MAX6pci,
MAX3pc104,
X-KiT-3,
MAX5dip,
MAX8dip
und MAX-Module
incl. MAX-PC's

SORCUS

Alle Angaben in diesem Handbuch sind ohne Gewähr und können ohne weitere Benachrichtigung geändert werden. Da sich trotz aller Bemühungen Fehler nie vollständig ausschließen lassen, sind wir für Hinweise dankbar. Eventuelle Erweiterungen und Korrekturen finden Sie auf der mitgelieferten CD.

3. Auflage
31.03.2004
SORCUS Computer GmbH
Im Breitspiel 11
69126 Heidelberg

Inhaltsverzeichnis

5. Das Multi-Tasking Betriebssystem "OsX" 5-1

6. Bibliotheken 6-1

9. Remote-Debugging mit RTDS 9-1

10. Modulbeschreibungen 10-1

Anhang

1. Einleitung

1.1. Das X-Bus-System

Das X-Bus-System ist ein modulares Messdatenerfassungs-, Steuerungs- und Kommunikationssystem. Es zeichnet sich insbesondere durch seine hohe Flexibilität in der Anpassung an neue Aufgaben aus. Das X-Bus-System besteht aus den sogenannten MAX-Modulen sowie dazugehörigen Trägerkarten. Als X-Bus wird der Bus bezeichnet, der die MAX-Module auf der Trägerkarte verbindet. Alle MAX-Module haben dafür eine X-Bus-Schnittstelle.

Die Trägerkarten unterscheiden sich durch die Bauform, die Anzahl der Modul-Steckplätze, und ihre Schnittstelle nach außen. Bislang sind folgende Trägerkarten für unterschiedliche Anwendungsfälle verfügbar:

- MAX6pci: PCI-Einsteckkarte für 6 MAX-Module
- MAX3pc104: PC104-Karte für 3 MAX-Module
- MAX5dip: Stand-Alone-System inkl. Stromversorgung für 5 MAX-Module
- MAX8dip: Stand-Alone-System inkl. Stromversorgung für 8 MAX-Module
- X-KiT-3: Evaluation-Board für 3 MAX-Module mit Display

Darüber hinaus können eigene Trägerkarten oder MAX-Module entwickelt, bzw. MAX-Module in begleitende Hardware eindesigned werden. Dazu stehen Application Notes auf der mitgelieferten CD und der SORCUS-Homepage zur Verfügung.

Aus Software-Sicht spielt die verwendete Trägerkarte keine Rolle – alle Karten verhalten sich bezüglich der aufsteckenden MAX-Module identisch. Unterschiede bestehen lediglich in direkt auf der Trägerkarte untergebrachten Funktionseinheiten wie z.B. LEDs.

Alle Trägerkarten können mit allen MAX-Modulen in beliebiger Kombination und Anordnung bestückt werden.

Die Stand-Alone-Systeme MAX5dip, MAX8dip, X-KIT-3 und BASiS-6 erfordern in jedem Fall das Aufstecken eines CPU-Moduls als lokale Intelligenz. Dafür stehen derzeit das X-MAX-1 und das X-MAX-E zur Verfügung. Beide sind jeweils komplette 486-PCs mit allen Standard-Schnittstellen und Speicher, wobei das X-MAX-E zusätzlich eine Ethernet-Schnittstelle enthält. In Vorbereitung ist derzeit das

X-MAX-400, das einen 400MHz ARM-Prozessor verwendet und damit eine deutlich schnellere Verarbeitung bieten wird.

Die CPU-Module werden mit dem Echtzeit-Multi-Tasking-Betriebssystem OsX ausgeliefert, das schnelle Reaktion auf Interrupts ermöglicht. Die Ankopplung der Stand-Alone-Systeme an einen Host-PC kann derzeit über Ethernet oder eine serielle Schnittstelle erfolgen.

Auf der MAX6pci-Karte ist ein CPU-Modul nicht unbedingt erforderlich. Diese Trägerkarte kann mit 6 I/O-Modulen (nicht intelligente Module werden als allgemein als I/O-Module bezeichnet) bestückt werden. In diesem Fall werden die Module komplett vom PC, in dem die MAX6pci eingesteckt ist, gesteuert. Die Echtzeitfähigkeit und Interrupt-Reaktionszeit ist in diesem Fall stark vom PC-Betriebssystem und dessen Auslastung durch andere laufende Programme abhängig. Um davon unabhängig zu werden, besteht auf der MAX6pci die Möglichkeit, durch Aufstecken eines CPU-Moduls ein intelligentes Sub-System zu schaffen, das dann parallel zum PC arbeitet. Jederzeit können beide miteinander kommunizieren.

Mit einer MAX6pci, einem CPU-Modul sowie bis zu 5 weiteren I/O-Modulen (durch die Stapelbarkeit einiger Module u.U. sogar noch mehr) kann ein System aufgebaut werden, das echte **Parallelverarbeitung** bietet. Eine typische Aufteilung der Aufgaben ist z.B. die Messdatenerfassung sowie ggf. eine Vorverarbeitung der Messdaten auf dem CPU-Modul und die Weiterverarbeitung (d.h. Darstellung, Analyse, Speicherung usw.) der erfassten Daten auf dem PC.

Als Besonderheit unterstützt der X-Bus den **Multi-Prozessor**-Betrieb, d.h. für Anlagen mit hohen Leistungsanforderungen besteht die Möglichkeit, mehrere CPU-Module auf eine Trägerkarte zu stecken. Die Aufgaben können dann auf die einzelnen CPUs aufgeteilt werden, z.B. erfasst eine CPU-Modul mit maximaler Geschwindigkeit Messwerte von einem Analog-Eingangs-Modul, ein anderes bedient ein Kommunikationsmodul mit hoher Übertragungsgeschwindigkeit und ein drittes kümmert sich im wesentlichen um Berechnungen und die Kommunikation mit einem Host-PC.

1.2. Einsatz und Programmierung

Es stehen prinzipiell zwei Möglichkeiten zum Einsatz des Systems zur Auswahl:

1. **Einsatz des Systems ohne eigene Programmierung.** In diesem Fall empfiehlt sich der Einsatz der Komplett-Software ARGUS. Neben einer Beschreibung finden Sie auf der mitgelieferten CD auch eine Testversion von ARGUS. Es bietet vielfältige Möglichkeiten in den Einsatzfeldern Messen, Analysieren und Dokumentieren, Anlagenüberwachung, Prüfstandsautomatisierung,

Qualitätskontrolle und Ferndiagnose. Für ARGUS gibt es ein eigenes Handbuch, so dass hier darauf nicht näher eingegangen wird.

2. **Erstellen eigener Programme für das X-Bus-System.** SORCUS liefert dafür mit jeder Karte ein umfangreiches Software-Packet mit. Es besteht aus Treibern, Bibliotheken, Beispielprogrammen und Hilfsprogrammen für die Diagnose und Entwicklung eigener Programme.

Dieses Buch dient als Referenz- und Programmierhandbuch bei der Erstellung eigener Programme für das X-Bus-System.

Es können Programme für den Host-PC geschrieben werden, die das System aus Windows (zukünftig auch LINUX) heraus ansprechen. Dazu werden Bibliotheken (s. Kapitel 6) mitgeliefert, die die Programmierung in den Programmiersprachen C, Delphi, Visual Basic sowie auch LabView erlauben.

Außerdem werden Bibliotheken mitgeliefert, mit deren Hilfe Programme für das Echtzeitbetriebssystem OsX auf den CPU-Modulen (im folgenden werden diese Programme auch als Echtzeitprogramme bezeichnet) geschrieben werden können. Dies kann in C oder Pascal geschehen. Die Besonderheiten des Betriebssystems OsX sind in Kapitel 5 beschrieben. Für die Entwicklung dieser Echtzeitprogramme findet sich auf der CD das Entwicklungs- und Debug-Programm RTDS (s. Kapitel 8).

Das Ansprechen der I/O-Module erfolgt über die sogenannten Modul-Device-Treiber (MDD). Deren grundsätzliche Eigenschaften und Verwendung ist in Kapitel 7 erläutert. Die speziellen Eigenschaften der jeweiligen MAX-Module und deren MDD sind in Kapitel 9 aufgeführt.

Das ebenfalls mitgelieferte Programm SNW32 ist ein hilfreicher Assistent bei der Inbetriebnahme und Diagnose (s. Kap. 3).

1.3. Kompatibilität zu anderen SORCUS-Karten

Die SORCUS-Karten MODULAR-4, Multi-LAB/2 und Multi-COM sind ISA-Bus Karten mit eigener CPU. Sie entsprechen vom Prinzip der MAX6pci-Karte mit einem CPU-Modul. Bei der MODULAR-4 ist die Funktionalität auch über Aufsteckmodule konfigurierbar.

Praktisch alle Funktionen, die mit den ISA-Karten ausgeführt werden können, lassen sich auch mit einer MAX6pci mit den entsprechenden MAX-Modulen erreichen. Die Programmierung der Systeme ist weitgehend identisch, da das auf den CPU-Modulen verwendete Betriebssystem OsX auch dort zum Einsatz kommt. Im Detail bestehen natürlich Unterschiede bei allen Bibliotheksfunktionen. Strukturelle Änderungen sind bei Portierung z.B. eines MODULAR-4 Programms auf eine MAX6pci nicht

erforderlich. Auf der MAX6pci müssen alle I/O-Module grundsätzlich über die Modul-Device-Treiber angesprochen werden.

1.4. CE-Kennzeichnung

Das X-Bus-System ist ein OEM-Produkt und als Zulieferteil für die Weiterverarbeitung durch Industrie, Handwerk oder sonstige auf dem Gebiet der elektromagnetischen Verträglichkeit fachkundigen Betriebe oder Personen bestimmt. Im Sinne des EMVG vom 18. September 1998 §6 Abs. 9 besteht daher für das X-Bus-System keine CE-Kennzeichnunspflicht.

Verkabelung, verwendeter PC und die Einsatzumgebung sind Faktoren, die sich auf die EMV eines Gerätes auswirken können. Ein Gerät, in das eine oder mehrere Trägerkarten (z.B. MAX6pci) bestückt mit einem MAX-PC und/oder MAX-Modulen eingesetzt wurden, muss in seiner Gesamtheit entsprechend den dafür gültigen Richtlinien bewertet werden, wenn mit dem CE-Kennzeichen Konformität dokumentiert werden soll oder muss.

Selbstverständlich wurden bei der Entwicklung des X-Bus-Systems alle möglichen Maßnahmen für einen EMV-gerechten Aufbau ergriffen.

1.5. Weitere Informationen

Dieses Handbuch bietet eine Installationsanleitung, beschreibt alle Komponenten des Systems und dient als Nachschlagewerk für die Programmierung des X-Bus-Systems.

Korrekturen und Erweiterungen dieses Buchs, die zum Zeitpunkt des Drucks noch nicht vorlagen, finden Sie ggf. auf der mitgelieferten CD. Aktuelle Versionen aller Software-Produkte und Beschreibungen stehen Ihnen darüber hinaus jederzeit auf der SORCUS-Homepage (www.sorcus.com) zum Download zur Verfügung.

Als weitere Unterstützung beim Einstieg in das X-Bus-System bietet SORCUS auch ein- bis zweitägige **Intensivseminare** an. Näheres dazu finden Sie ebenfalls auf der SORCUS-Homepage. Zwei Kapitel aus diesen Seminaren stehen dort auch als PDF-Dokumente bereit.

2. Trägersysteme

2.1. Installation und Einbau

2.1.1. Inbetriebnahme

Die MAX-Module bzw. die Trägerkarten sind elektrostatisch geschützt verpackt. Beim Auspacken sollte unbedingt darauf geachtet werden, dass die MAX-Module resp. die Trägerkarten nicht elektrostatischen Entladungen ausgesetzt werden.

Bei Beschädigungen oder sonstigen Fehlern setzen Sie sich bitte umgehend mit Ihrem Lieferanten oder mit SORCUS Computer GmbH in Verbindung.

Es empfiehlt sich folgender Arbeitsablauf:

1. Überprüfen auf Vollständigkeit und Unversehrtheit der Lieferung, sowohl der Trägerkarte als auch der einzelnen MAX-Module.
2. Auspacken der Trägerkarte bzw. der MAX-Module (Vorsicht vor elektrostatischen Aufladungen!).
3. Überprüfen (!) und Einstellen der gewünschten Konfiguration der Trägerkarte (z.B. bei MAX6isa, MAX3pc104 und X-KiT-3) und auf den Modulen (siehe Modulbeschreibungen). Hinweis: Beim X-KiT-3 muss der MAX-PC auf Steckplatz 1 gesteckt werden.
4. Wenn Sie die Uhr oder das statische RAM auf der Karte puffern wollen, ist hierfür eine externe Batterie oder die Batterie des PC bzw. des X-KiT-3 oder des MAX5dip oder MAX8dip zu verwenden.
5. a) X-KiT-3, MAX5dip, MAX8dip: Anschließen des mitgelieferten (nur bei X-KiT-3) Netzteils. Anschließen des mitgelieferten seriellen Verbindungskabels (Null-Modem) an den PC (z.B. COM1) und an das X-KiT-3 (St6 für X-MAX-1 bzw. St10 für X-MAX-E), dann Einschalten des PC.
   b) MAX6pci: Ausschalten des PC, PC öffnen. Wie das gemacht wird, steht in den Handbüchern zu Ihrem PC (vorher unbedingt Netzstecker herausziehen!) Einstecken der Trägerkarte in den PC (ohne Module und ohne dass irgendwelche Kabel an die Karte angeschlossen sind). Die MAX6pci-Karte wird, wie andere PC-Karten auch, in den PC eingesteckt und mit einer Schraube am Bügel gesichert. Der PCI-Steckplatz, in den die Karte eingesteckt wird, ist im Prinzip

gleichgültig. Aus Gründen der Störsicherheit wird aber empfohlen, die Karte möglichst direkt neben einer Seitenwand und mit Abstand zu den Floppy-Disk- und Hard-Disk-Laufwerken und Kabeln zu platzieren. Wenn noch weitere Steckplätze frei sind, sollte zwischen der Trägerkarte und der nächsten Karte ein Steckplatz frei bleiben. Einschalten des PC.

! *Es ist besonders darauf zu achten, dass die Trägerkarte MAX6pci weder mechanisch noch elektrisch Kontakt zu einer benachbarten Karte hat, ebenso, dass die Trägerkarte und die Module an der unteren Kante keinen Kontakt mit Bauteilen auf dem Motherboard des PC haben.*

6. Treiberinstallation siehe Kapitel 2.1.3.
7. Starten des Programms "SNW32" zur Prüfung der Funktionsfähigkeit der Karte.
8. Ausschalten des PC bzw. des X-KiT-3 oder MAX5dip oder MAX8dip, Aufstecken der Anschlusskabel (das andere Ende der Kabel darf nirgendwo angeschlossen sein!), Aufstecken der Module auf die Trägerkarte, Karte (MAX6pci) wieder in den PC einbauen und den PC bzw. X-KiT-3 oder MAX5dip oder MAX8dip einschalten.
9. Starten des Programms "SNW32" und Überprüfen aller Schnittstellen.

Falls beim Einsatz mit einer Karte einmal Probleme auftauchen sollten, ist das Programm SNW32 auch geeignet, um einen möglichen Fehler auf der Karte oder auf den Modulen zu lokalisieren. Dazu empfiehlt es sich, zunächst alle Kabelverbindungen zur Karte zu unterbrechen und z.B. mit Hilfe eines Funktionsgenerators, eines Oszillographen oder eines Voltmeters die Ein- und Ausgangssignale und einzelne Funktionsgruppen auf der Karte zu testen. Eine weitere Möglichkeit zum Testen eines Moduls ist das Test-Modul X-TEST-1. Es wird unter ein zu testendes Modul gesteckt und ermöglicht das Überprüfen aller I/O-Pins des zu testenden Moduls.

Sind die einzelnen Tests erfolgreich verlaufen, können die gewünschten externen Anschlüsse aufgesteckt werden.

2.1.2. Software auf CD

Die beiliegende SORCUS-CD ist nicht kopiergeschützt. Die Programme dürfen für Zwecke des Anwenders in Zusammenarbeit mit SORCUS-Produkten beliebig oft kopiert werden. Wiederverkäufer können die Programme auch verändern und weiter verkaufen. Eine besondere Genehmigung der SORCUS Computer GmbH ist dazu nicht erforderlich.

Alle Programme auf der CD sind als Hilfsmittel gedacht, um Ihnen den Einsatz des MAX-PC bzw. der Trägerkarten bzw. –systeme zu erleichtern und zu verdeutlichen. Eine Garantie für ein fehlerfreies Funktionieren dieser Programme wird von SORCUS Computer GmbH aber nicht übernommen. Insbesondere werden jede Haftung, Gewährleistung und eventuelle Schadenersatzansprüche, die sich aus dem Einsatz des MAX-PC bzw. der Trägerkarten bzw. –systeme sowie der mitgelieferten Software ergeben könnten, ausgeschlossen.

Aktuelle Software-Versionen können Sie vom Internet (www.sorcus.com) herunterladen! Die Struktur auf der Internet-Seite ist identisch mit der auf der CD.

2.1.3. Treiberinstallation

Bevor Sie MAX-Module unter Windows oder Linux nutzen können, müssen Sie einen Treiber installieren. Der Treiber befindet sich in der Datei *maxsetup.zip* (diese kann von der SORCUS Homepage *www.sorcus.com* unter *Software\X-Bus* heruntergeladen werden) bzw. auf der SORCUS-CD im Verzeichnis *files\instwin*. Momentan werden die Windows Betriebssysteme 98, ME, NT 4.0, 2000 und XP unterstützt. Der Installationsvorgang unterscheidet sich dabei zwischen NT 4.0 und den anderen Betriebssystemen. Die ZIP-Datei muss zunächst in ein beliebiges Verzeichnis entpackt werden.

2.1.3.1. Treiberinstallation Windows NT 4.0

Um den Treiber unter Windows NT 4.0 zu installieren, führen Sie die folgenden Schritte aus:

- Starten Sie das Programm *setup.exe*.
- Drücken Sie nun die Schaltfläche *Install driver files*, um die Treiberdateien auf Ihren Rechner zu kopieren.
- Drücken Sie nun die Schaltfläche *Install device* um eine Karte zu installieren. Dabei wird der Karte eine Nummer zugewiesen, die bei der Programmierung zur Identifizierung der Karte verwendet wird.
- Wenn Sie PC-basierte Modul-Device-Treiber (siehe Kapitel 8) verwenden möchten, dann drücken Sie die Schaltfläche *Install PC Platform Module Device Drivers*.

2.1.3.2. Treiberinstallation Windows 98, ME, 2000, XP

MAX6pci

Beim Starten des Rechners erkennt Windows die eingesteckte SORCUS Karte selbstständig und fordert Sie auf, einen Datenträger mit dem Treiber anzugeben. Wählen Sie dazu das Verzeichnis *files\instwin* von der SORCUS-CD aus oder entpacken Sie die Datei *maxsetup.zip* in ein beliebiges Verzeichnis und wählen Sie dieses Verzeichnis aus. Nun wird der Treiber samt aller PC-basierten Modul-Device-Treiber (siehe Kapitel 7) installiert.

X-KiT-3, MAX5dip, MAX8dip, BASiS-6

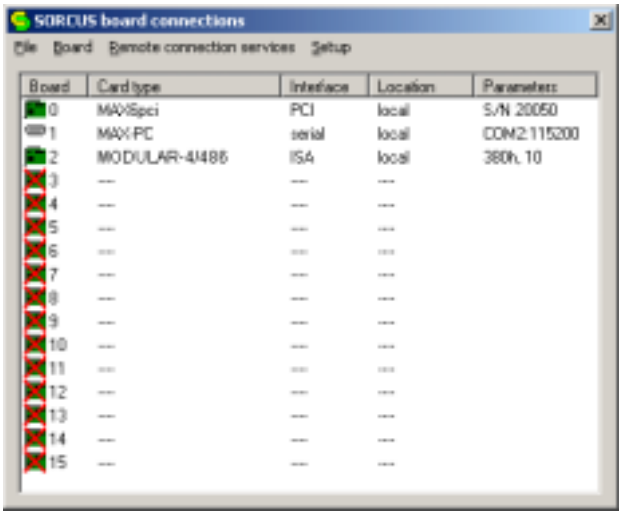
Wählen Sie das Verzeichnis *files/instwin* von der SORCUS-CD aus oder entpacken Sie die Datei *maxsetup.zip*. Starten Sie nun das Programm setup.exe. Nun wird der Treiber installiert.

2.1.4. Treiber-Updates

Wenn Sie den SORCUS Treiber updaten möchten, dann starten Sie das Programm *setup.exe* und drücken die Schaltfläche *Update drivers*. Es werden nun alle installierten SORCUS Treiber (inklusive eventuell installierter PC basierter Modul-Device-Treiber) aktualisiert.

2.1.5. Systemsteuerung

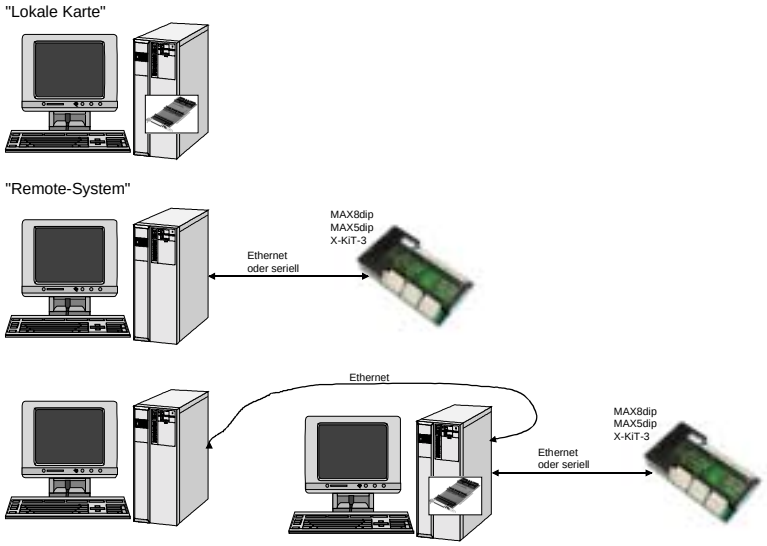
Nach der Treiberinstallation befindet sich in der Windows-Systemsteuerung ein Symbol *SORCUS boards*. Wenn Sie das Programm starten, wird Ihnen eine Übersicht aller installierten SORCUS-Karten angezeigt. Das Programm dient auch zur Einrichtung von Remote-Verbindungen zu Karten (seriell, Netzwerk).



2.1.6. Remote-Verbindungen

Als Remote-Verbindung wird eine Verbindung von einem Rechner, z.B. einem PC, zu einem externen X-Bus-System (z.B. X-KiT-3) bezeichnet. X-Bus-Systeme, die über eine Remote-Verbindung angesprochen werden, werden im folgenden als „Remote-Systeme“ bezeichnet. Karten, die über den internen Bus des Rechners (z.B. ISA oder PCI) angesprochen werden, werden im folgenden als „Lokale-Karten“ bezeichnet.

SORCUS erlaubt z.Zt. Remote-Verbindungen über eine serielle-Schnittstelle oder eine Ethernet-Schnittstelle. Zusätzlich kann auch eine Remote-Verbindung zu einer Karte aufgebaut werden, die mit einem anderen Rechner innerhalb eines Ethernet-Netzwerkes (als „Lokale-Karte“ oder als „Remote-System“) verbunden ist.



Ist die Remote-Verbindung eingerichtet, gibt es aus Programmierer- bzw. Anwendersicht keinen Unterschied zwischen einem „Remote-System“ und einer „Lokalen-Karte“.

Die CPU-Module X-MAX-1 und X-MAX-E werden im folgenden auch als „CPU-Modul“ bezeichnet. Das Modul, das die Ethernet-Schnittstelle für die Remote-Verbindung zur Verfügung stellt (z.B. X-ETH-10 oder X-MAX-E), wird im folgenden auch als „Ethernet-Modul“ bezeichnet. Ein X-MAX-E Modul kann daher sowohl „CPU-Modul“ als auch „Ethernet-Modul“sein.

Die im folgenden erwähnten Treiber bzw. das Programm SNW32 sind auf der SORCUS-CD enthalten oder können von unserer Internetseite (www.sorcus.com) heruntergeladen werden. SNW32 muss mindestens die Version 3.H.009 aufweisen.

2.1.6.1. Remote-Verbindung zu einem anderen Rechner

Führen Sie die nachfolgenden Schritte aus, um eine Remote-Verbindung zu einem X-Bus System aufzubauen, das mit einem anderen Rechner verbunden ist:

- Bitte beachten Sie, dass auf dem Remote-Rechner, mit dem Sie sich verbinden wollen, der „Remote connection service“ installiert werden muss. Starten Sie hierzu auf dem Remote-PC das Symbol *SORCUS boards* in der Windows-Systemsteuerung und wählen Sie im Menü *Remote connection services* den Punkt *Install mlxserv.exe*.
- Installieren Sie auf dem PC zunächst den Windows-Treiber und das Programm SNW32.
- Starten Sie auf dem PC in der Windows-Systemsteuerung das Symbol *SORCUS boards*.
- Jetzt können Sie in der Kartenliste mit der rechten Maustaste auf eine freie Karte klicken und aus dem Kontextmenü *New remote connection* auswählen. Als *Type* wählen Sie *Connection to remote PC*.
- Im nachfolgenden Dialog müssen Sie einen Remote-PC sowie ein dort installiertes X-Bus-System auswählen.
- Nach diesem Vorgang erscheint in der Liste ein neuer Eintrag.
- Starten Sie nun SNW32 und wählen Sie das X-Bus-System aus. Die X-Bus Module auf dem System erscheinen in einer Baumstruktur.

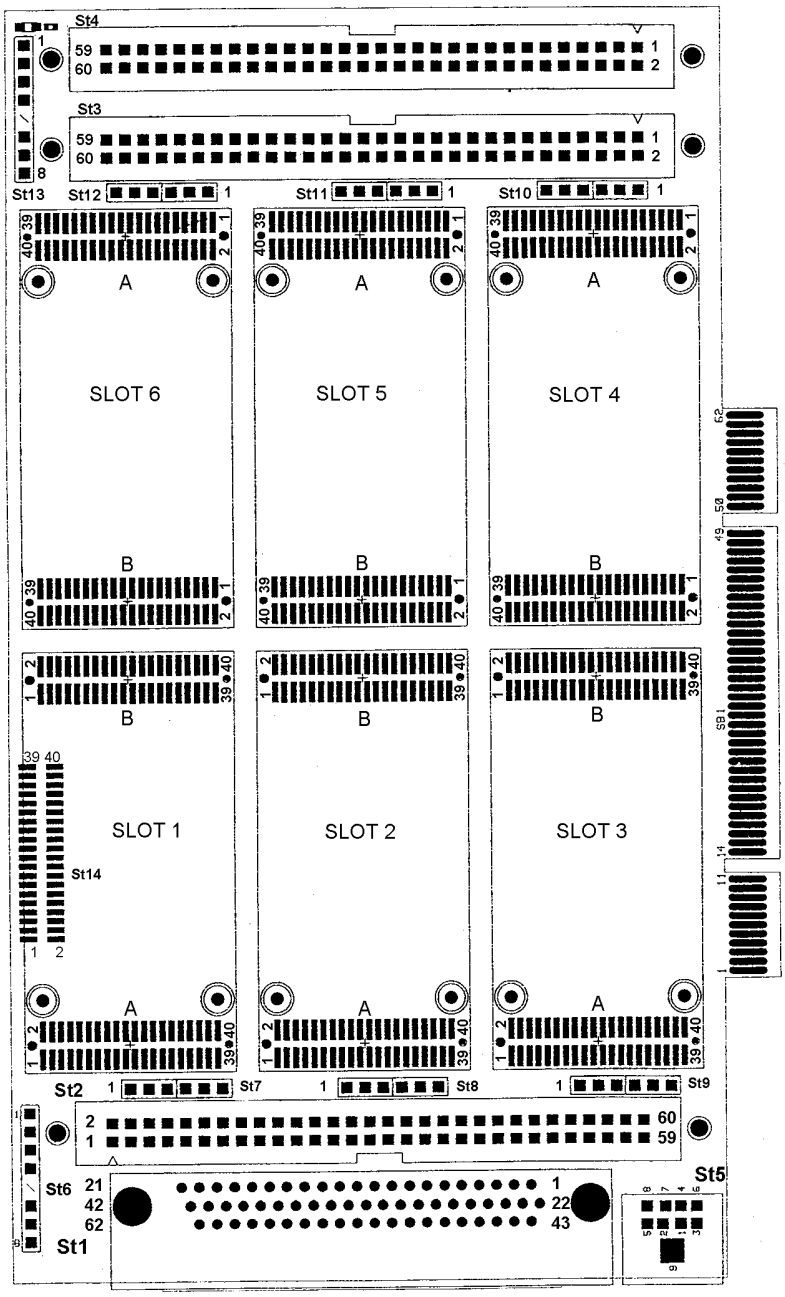
2.1.6.2. Entfernen von Remote Verbindungen

Zum Entfernen einer Remote-Verbindung klicken Sie mit der rechten Maustaste auf die zu löschende Verbindung und wählen aus dem Kontextmenü *Delete*. Die Verbindung wird nun aus der Kartenliste gelöscht. Falls keine Remote-Verbindungen mehr existieren, können Sie auch den DCOM-Server aus ihrem System entfernen. Wählen Sie dazu aus dem Menü *Remote connection services* den Menüpunkt *Remove mlxserv.exe* aus.

Notizen

2.2. MAX6pci

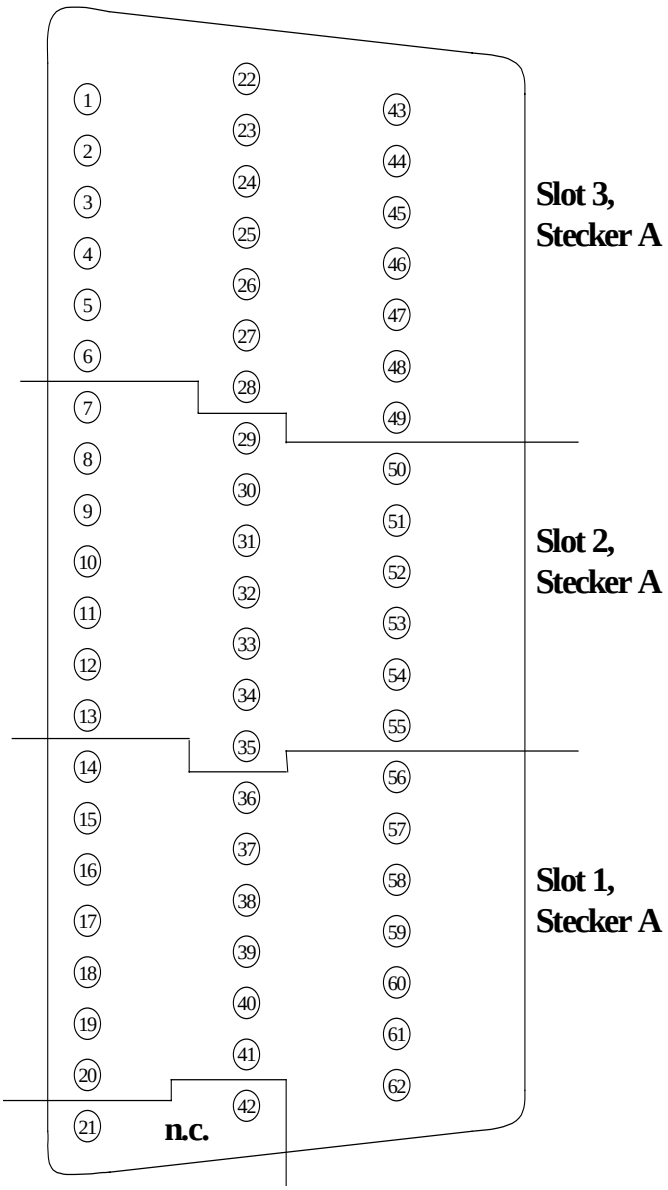
2.2.1. Lageplan der Trägerkarte



2.2.2. MAX6pci Stecker und Abschirmung

2.2.2.1. Steckerzuordnung MAX6pci

Pinbelegung der 62-poligen Buchse (St1) , Ansicht auf Buchse MAX6pci von außen.



Die nachfolgende Tabelle enthält die Zuordnung der Pins des Steckers A der MAX-Module des entsprechenden Slots (1 bis 6) zu den Steckern St1 bis St4 auf der Trägerkarte MAX6pci (siehe auch Lageplan).

Slot 1, Stecker A	**St1[1] (D-Sub-62)**	**Slot 2, Stecker A**	**St1[1] (D-Sub-62)**	**Slot 3, Stecker A**	**St1[1] (D-Sub-62)**
1	20	1	35	1	28
2	41	2	13	2	49
3	62	3	34	3	6
4	19	4	55	4	27
5	40	5	12	5	48
6	61	6	33	6	5
7	18	7	54	7	26
8	39	8	11	8	47
9	60	9	32	9	4
10	17	10	53	10	25
11	38	11	10	11	46
12	59	12	31	12	3
13	16	13	52	13	24
14	37	14	9	14	45
15	58	15	30	15	2
16	15	16	51	16	23
17	36	17	8	17	44
18	57	18	29	18	1
19	14	19	50	19	22
20	56	20	7	20	43

Slot 1, Stecker A	**St2**	**Slot 2, Stecker A**	**St2**	**Slot 3, Stecker A**	**St2**
21	1	21	21	21	41
22	2	22	22	22	42
23	3	23	23	23	43
24	4	24	24	24	44
25	5	25	25	25	45
26	6	26	26	26	46
27	7	27	27	27	47
28	8	28	28	28	48
29	9	29	29	29	49
30	10	30	30	30	50
31	11	31	31	31	51
32	12	32	32	32	52
33	13	33	33	33	53
34	14	34	34	34	54
35	15	35	35	35	55
36	16	36	36	36	56
37	17	37	37	37	57
38	18	38	38	38	58
39	19	39	39	39	59
40	20	40	40	40	60

[1] Bei St1 sind Pin 21 und 42 nicht angeschlossen

Slot 4, Stecker A	**St3**	**Slot 5, Stecker A**	**St3**	**Slot 6, Stecker A**	**St3**
1	1	1	21	1	41
2	2	2	22	2	42
3	3	3	23	3	43
4	4	4	24	4	44
5	5	5	25	5	45
6	6	6	26	6	46
7	7	7	27	7	47
8	8	8	28	8	48
9	9	9	29	9	49
10	10	10	30	10	50
11	11	11	31	11	51
12	12	12	32	12	52
13	13	13	33	13	53
14	14	14	34	14	54
15	15	15	35	15	55
16	16	16	36	16	56
17	17	17	37	17	57
18	18	18	38	18	58
19	19	19	39	19	59
20	20	20	40	20	60

Slot 4, Stecker A	**St4**	**Slot 5, Stecker A**	**St4**	**Slot 6, Stecker A**	**St4**
21	1	21	21	21	41
22	2	22	22	22	42
23	3	23	23	23	43
24	4	24	24	24	44
25	5	25	25	25	45
26	6	26	26	26	46
27	7	27	27	27	47
28	8	28	28	28	48
29	9	29	29	29	49
30	10	30	30	30	50
31	11	31	31	31	51
32	12	32	32	32	52
33	13	33	33	33	53
34	14	34	34	34	54
35	15	35	35	35	55
36	16	36	36	36	56
37	17	37	37	37	57
38	18	38	38	38	58
39	19	39	39	39	59
40	20	40	40	40	60

2.2.2.2. Abschirmung

Die Brücken St7 .. St12 können dafür verwendet werden, um Pin 10, 20, 30 oder 40 des Steckers A eines Moduls mit der Abschirmung der Trägerkarte zu verbinden. Die Abschirmung besteht aus einer Kupferfläche auf den äußeren Leiterplatten-Lagen der Trägerkarte, die die Leitungen von Stecker A umschließt. Die Abschirmungen aller Module sind galvanisch von einander getrennt und auch getrennt von der Trägerkarte.

St7	**Slot 1, Stecker A**	**St8**	**Slot 2, Stecker A**	**St9**	**Slot 3, Stecker A**
1	10	1	10	1	10
2[1]	–	2[1]	–	2[1]	–
3	20	3	20	3	20
4	30	4	–	4	30
5[1]	–	5[1]	30	5[1]	–
6	40	6	40	6	40

St10	**Slot 4, Stecker A**	**St11**	**Slot 5, Stecker A**	**St12**	**Slot 6, Stecker A**
1	10	1	10	1	10
2[1]	–	2[1]	–	2[1]	–
3	20	3	20	3	20
4	30	4	–	4	30
5[1]	–	5[1]	30	5[1]	–
6	40	6	40	6	40

[1] Pin 2 und 5 von St7 bis St12 gehen an die Abschirmung des Moduls auf Slot 1 bis Slot 6 auf der Trägerkarte, sofern das Modul über eine Abschirmung verfügt.

2.3. MAX3pc104

2.3.1. Lageplan der Trägerkarte MAX3pc104

Der endgültige Lageplan lag bei Drucklegung dieses Handbuchs noch nicht vor.

2.3.2. MAX3pc104 Stecker und Abschirmung

2.3.2.1. Steckerzuordnung MAX3pc104

Die nachfolgende Tabelle enthält die Zuordnung der Pins des Steckers A der MAX-Module des entsprechenden Slots (1 bis 3) zu den Steckern St1 bis St3 auf der Trägerkarte MAX3pc104 (siehe auch Lageplan).

Slot 1, Stecker A	**St1**	**Slot 2, Stecker A**	**St2**	**Slot 3, Stecker A**	**St3**
1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5
6	6	6	6	6	6
7	7	7	7	7	7
8	8	8	8	8	8
9	9	9	9	9	9
10	10	10	10	10	10
11	11	11	11	11	11
12	12	12	12	12	12
13	13	13	13	13	13
14	14	14	14	14	14
15	15	15	15	15	15
16	16	16	16	16	16
17	17	17	17	17	17
18	18	18	18	18	18
19	19	19	19	19	19
20	20	20	20	20	20

Slot 1, Stecker A	**St1**	**Slot 2, Stecker A**	**St2**	**Slot 3, Stecker A**	**St3**
21	21	21	21	21	21
22	22	22	22	22	22
23	23	23	23	23	23
24	24	24	24	24	24
25	25	25	25	25	25
26	26	26	26	26	26
27	27	27	27	27	27
28	28	28	28	28	28
29	29	29	29	29	29
30	30	30	30	30	30
31	31	31	31	31	31
32	32	32	32	32	32
33	33	33	33	33	33
34	34	34	34	34	34
35	35	35	35	35	35
36	36	36	36	36	36
37	37	37	37	37	37
38	38	38	38	38	38
39	39	39	39	39	39
40	40	40	40	40	40

2.3.2.2. Abschirmung

Die Brücken St4 .. St6 können dafür verwendet werden, um Pin 10, 20, 30 oder 40 des Steckers A eines Moduls mit der Abschirmung der Trägerkarte zu verbinden. Die Abschirmung besteht aus einer Kupferfläche auf den äußeren Leiterplatten-Lagen der Trägerkarte, die die Leitungen von Stecker A umschließt. Die Abschirmungen aller Module sind galvanisch von einander getrennt und auch getrennt von der Trägerkarte.

St4	**Slot 1, Stecker A**	**St5**	**Slot 2, Stecker A**	**St6**	**Slot 3, Stecker A**
1	10	1	10	1	10
2[1]	–	2[1]	–	2[1]	–
3	20	3	20	3	20
4	30	4	–	4	30
5[1]	–	5[1]	30	5[1]	–
6	40	6	40	6	40

[1] Pin 2 und 5 von St4 bis St6 gehen an die Abschirmung des Moduls auf Slot 1 bis Slot 6 auf der Trägerkarte, sofern das Modul über eine Abschirmung verfügt.

2.4. BASiS-6 – Trägersystem für 6 MAX-Module

2.4.1. Beschreibung der Baugruppe

BASiS-6 ist ein Trägersystem für 6 MAX-Module zur Integration in die Hardware des Anwenders. Alle I/O-Signale der aufgesteckten MAX-Module sind auf der Unterseite der Baugruppe auf Steckverbinder (Typ = SUYIN 127150FA040) geführt. Die Spannungsversorgung erfolgt extern mit 5V, alle intern benötigten Spannungen werden daraus abgeleitet. Ein zentraler Chip (FPGA) erzeugt den X-Bus Takt für die Module und verwaltet die Module. Über den zentralen Chip kann auch bei dafür geeigneten X-Bus Modulen ein Update des Moduldesigns vorgenommen werden.

Vom FPGA sind 11 Pins auf den Stecker 8 geführt, sog. User-Signale. Diese Signale (*FPGA01*..*FPGA11*) können vom Anwender gesetzt und gelesen werden, wobei *FPGA01* nur als Eingang verwendbar ist. Der Zustand der Eingänge wird im Register *User-Signal* gelesen und gesetzt. Im Register *User-Output-Enable* kann festgelegt werden, welche der Signale als Ausgang arbeiten sollen. Für den einfachen Zugriff auf diese Pins steht ein Modul-Device-Treiber zur Verfügung. Diese Ein- und Ausgänge sind 5V-tolerant, trotzdem muss bei Verwendung der Pins durch Schutzwiderstände und Dioden gegen die 3,3V Versorgung sichergestellt werden, dass die Spannung an den Pins zu keinem Zeitpunkt über der Versorgungsspannung der Baugruppe liegt.

2.4.1.1. Steckerposition und -belegung

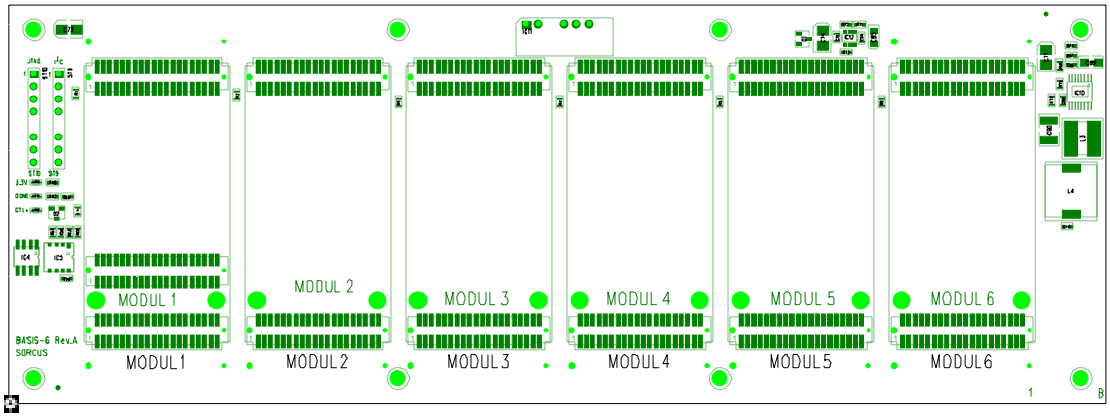
Folgende Steckverbinder sind auf BASiS-6 vorhanden:

Stecker / Buchse	Typ	Funktion
St1	SMD-Buchse 40-polig	Stecker A von Modul 1
St2	SMD-Buchse 40-polig	Stecker A von Modul 2
St3	SMD-Buchse 40-polig	Stecker A von Modul 3

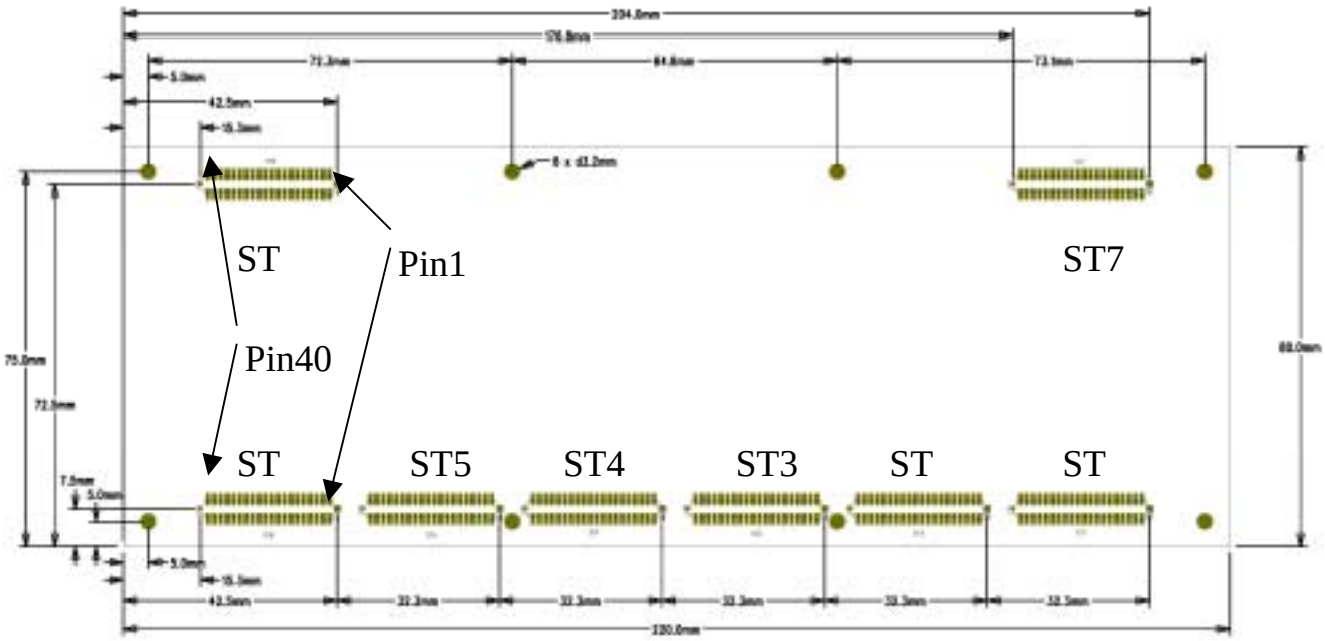
Stecker / Buchse	Typ	Funktion
St4	SMD-Buchse 40-polig	Stecker A von Modul 4
St5	SMD-Buchse 40-polig	Stecker A von Modul 5
St6	SMD-Buchse 40-polig	Stecker A von Modul 6
St7	SMD-Buchse 40-polig	Stecker C von Modul 1
St8	SMD-Buchse 40-polig	Spannungsversorgung und User-Signale
St9	Pfostenreihe 8-polig	I²C-Schnittstelle für Test
St10	Pfostenreihe 8-polig	JTAG-Schnittstelle für Test

Die Stecker St1 bis St6 sind 1:1 mit den Signalen der Modulstecker A der Module 1 bis 6 verbunden. D.h. Stecker 1 Pin 1 ist mit Stecker A Modul 1 Pin 1 verbunden. Stecker 7 ist 1:1 mit dem Modulstecker C des Modul 1 verbunden.

Blick auf die Modulseite von BASiS-6:



Blick auf die Steckerseite von BASiS-6:



2.4.1.2. Stecker St8: Spannungsversorgung und User-Signale

Der Stecker St8 ist für die Spannungsversorgung des Modulträgers und für die User-Signale verwendet. Die Steckerbelegung ist in folgender Tabelle aufgeführt:

Pin	Typ	Funktion
1-6	Spannungsversorgung	Ausgang, +3,3V von der Basisplatine
7-10	Spannungsversorgung	GND
11-12	Spannungsversorgung	Ausgang, +12V von der Basisplatine (ungefiltert)
13-14	Spannungsversorgung	GND
15-16	Spannungsversorgung	Ausgang, -12V von der Basisplatine (ungefiltert)
17-18	Spannungsversorgung	GND

Pin	Typ	Funktion
19	User-Signal	FPGA01, Eingang
20	User-Signal	FPGA02, als Ein- oder Ausgang konfigurierbar
21	User-Signal	FPGA03, als Ein- oder Ausgang konfigurierbar
22	User-Signal	FPGA04, als Ein- oder Ausgang konfigurierbar
23	User-Signal	FPGA05, als Ein- oder Ausgang konfigurierbar
24	User-Signal	FPGA06, als Ein- oder Ausgang konfigurierbar
25	User-Signal	FPGA07, als Ein- oder Ausgang konfigurierbar
26	User-Signal	FPGA08, als Ein- oder Ausgang konfigurierbar
27	User-Signal	FPGA09, als Ein- oder Ausgang konfigurierbar
28	User-Signal	FPGA10, als Ein- oder Ausgang konfigurierbar
29	Reset	Ausgang, active high
30	User-Signal	FPGA11, als Ein- oder Ausgang konfigurierbar
31-34	Spannungsver-sorgung	GND
35-40	Spannungsver-sorgung	Eingang, +5,0V zur Versorgung

Für die Spannungsversorgung müssen +5,0V an den Pins 35-40 gegen GND angelegt werden. Die +3,3V, +12V und –12V Ausgangsspannungen können zur Versorgung externer Baugruppen verwendet werden. Die gesamte Stromaufnahme aller Module und der externen Baugruppen darf bei 3,3V einen Strom von 3,0A nicht überschreiten. Bei +12V und –12V beträgt der maximale Gesamtstrom jeweils 85mA.

2.4.1.3. Stecker St9 und St10: I²C und JTAG für den zentralen Chip

St9 und St10 sind 8-pol. einreihige Pfostenstecker mit Rastermaß 2,54 mm. Pin 5 fehlt und dient der Codierung.

Pin an St9	Name
1	/PROG
2	CLK
3	/EN
4	DAT
5	Kein Pin
6	n.c.
7	GND
8	+3,3 Volt

Pin an St10	Name
1	TDI
2	TCK
3	TMS
4	TDO
5	Kein Pin
6	n.c.
7	GND
8	+3,3 Volt

2.4.1.4. Leuchtdioden

Auf BASiS-6 sind 3 Leuchtdioden vorhanden. Die rote LED, die mit 3.3V beschriftet ist, ist vom Benutzer ansteuerbar und intern mit dem Reset-Signal des X-Bus verknüpft. Während Reset ist die LED ausgeschaltet. Nach Reset leuchtet die LED und kann dann vom Benutzer aus- und eingeschaltet werden.

Die grüne LED ist mit DONE beschriftet und zeigt die erfolgreiche Konfigurierung des FPGA an. Diese LED leuchtet nach der Konfiguration dauerhaft.

Die blaue LED ist mit GTL+ beschriftet. Wenn diese LED nicht angesteuert wird, leuchtet sie schwach blau. Bei Ansteuerung leuchtet sie sehr hell blau. Diese LED zeigt bei zukünftigen Systemen die Fähigkeit der schnellen X-Bus Datenübertragung mit GTL+ Logikpegeln an.

2.4.2. Modul-Device-Treiber

Der Treiber ermöglicht den Zugriff auf den zentralen Chip des BASiS-6 Trägersystems. Der zentrale Chip hat die Slotummer 9 und die Layernummer 0.

2.4.2.1. Installation

Der Modul-Device-Treiber für OsX hat die Programmnummer 80F0h und den Dateinamen mbasis6.exe. Der Modul-Device-Treiber für Windows hat den Namen mbasis6.sys.

Die Installation aus einem PC-Programm:

Error = max_load_mdd (hModul, 9, 0, 0, 0x80F0, NULL, &hMDD);

Befehl in einer INS-Datei:

MAXLOADMDD slot=9 layer=0 progno=80F0

2.4.2.2. Kanaleigenschaftsstruktur CPS_BASIS6

Die CPS für das Modul hat den Namen CPS_BASIS6.

2.4.2.3. Register-Zugriffe

Um auf die Register uzugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*1*	Version dieser CPS Definition
.usDevice	*DEVICE_REGISTER*	Kanal zu einem Register
.usIndexFirst	*0, 2, 4, 6*	Nummer des Registers
.usIndexLast	*=.usIndexFirst*	Nummer des Registers
.usRegisterWidth	*BASIS6_WORD_REGISTER*	Breite des Registers
.ulFlags	*0*	reservierter Parameter
.usMode	*0*	reservierter Parameter

Strukturelement	Werte	Bedeutung
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff

Eingabe- und Ausgabedienst

Mit dem Ausgabedienst wird ein Register gesetzt, mit dem Eingabedienst wird ein Register ausgelesen.

Der Datentyp ist DATA_USHORT.

- **max_write_channel_ushort**
- **max_read_channel_ushort**

2.4.3. Besondere Eigenschaften

Parameter	Randbedingungen	min.	typ.	max.	Einheit	Anm.
Modulsteckplätze				6		
Für I/O Module geeignet		5				
Sonstiges						
Anzahl digitale Eingänge		1		11		
Anzahl digitale Ausgänge		0		10		
Spannungsversorgung der Platine		+4,5	+5	+5,5	V	
Stromaufnahme (ohne Module)	+5 V		Tbd.		mA	
Versorgungsstrom (für Module)	+3,3 V +12 V - 12 V			3 85 85	A mA mA	
Temperaturbereich						
Betrieb		0		+70	°C	
Lagerung		–40		+85	°C	

Notizen

2.5. MAX5dip: Dezentrale intelligente Peripherie

Das MAX5dip ist ein intelligentes, selbstständig arbeitendes, dezentrales Peripheriegerät. Es kann unabhängig von einem Host-Rechner zum Messen, Steuern und Regeln eingesetzt werden. Das kleine, kompakte Gehäuse ist für Hutschienenmontage vorgesehen.

Zur Anpassung an die Anwendung ist es mit 5 MAX-Modulen bestückbar, wovon 3 die eigentlichen Peripheriefunktionen des MAX5dip festlegen. Die Verbindung mit der Außenwelt geschieht über insgesamt 128 Schraubklemmen.

Eine ebenfalls über ein MAX-Modul konfigurierbare Kommunikationsschnittstelle zur Host-Anbindung erlaubt den Datenaustausch mit dem Host-System und z.B. den Download anwendungsbezogener Software ins lokale RAM oder ins Flash. Als Host-Schnittstelle möglich sind z.B. PROFIBUS-DP, CAN, Ethernet, USB, RS-232, RS-422, RS-485, etc.

Der fünfte Modul-Steckplatz ist für ein CPU-Modul vorgesehen. Hier kann z.B. der MAX-PC, ein MAX-Modul mit 100 MHz 486 CPU, 16 MByte RAM und 16 MByte Flash aufgesteckt werden.

Das Gerät ist in eine Terminalplatine mit den Schraubklemmen und eine Elektronikplatine aufgeteilt. Die Terminalplatine trägt die stehende Verdrahtung. Sie beinhaltet kein funktionsbestimmendes, elektrisches Bauteil, so daß seine Ausfallwahrscheinlichkeit gering ist. Das Gehäuse kann auf einer Hutschiene oder auch auf einer ebenen Fläche montiert werden. Die Elektronikplatine enthält die komplette Elektronik des Systems. Bei einem Austausch sind keine Leitungen zu lösen. Sie ist auf den Terminalblock aufgesteckt.

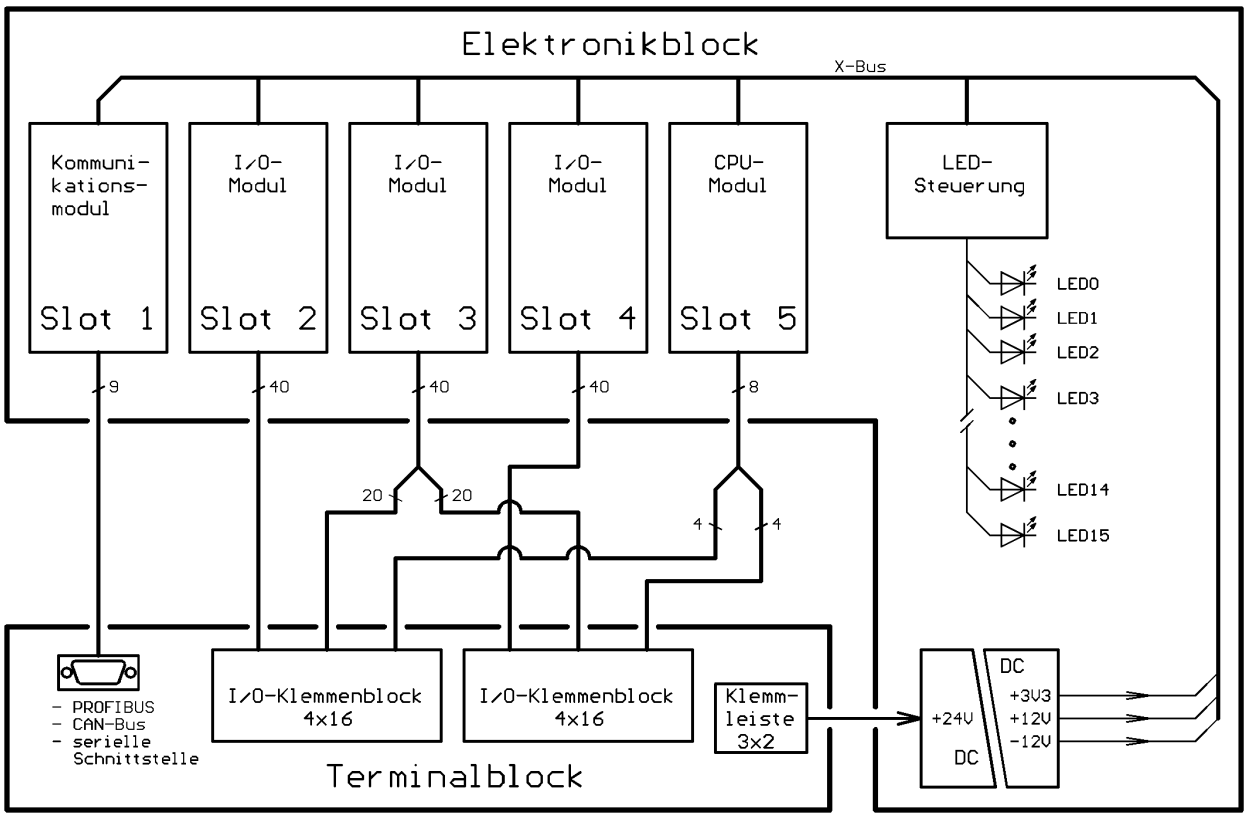
Mit Leuchtdioden können bis zu 16 Zustände von Ein- und Ausgangsleitungen angezeigt werden.

Das MAX5dip wird mit einer Gleichspannung von 24V (18,6V bis 36V) versorgt. Alle intern benötigten Betriebsspannungen werden daraus galvanisch getrennt erzeugt. Mit einem Schalter kann das Gerät ein- bzw. ausgeschaltet werden.

2.5.1. Besondere Eigenschaften

- 5 Steckplätze für MAX-Module, z.B. 3 I/O- Module, 1 CPU-Modul und eins für die Host-Schnittstelle (CAN, PROFIBUS, Ethernet, etc.)
- Pro Steckplatz können zusätzlich 2 Module übereinander gestapelt werden, z.B. CPU-Module
- 128 (3 x 40 + 8 Hilfsklemmen) Schraubklemmen für den Anschluß von I/O-Signalen
- 16 Leuchtdioden zur Anzeige von Signalzuständen
- Echtzeit-Multi-Tasking Betriebssystem auf CPU-Modul
- Anwendungsbezogene Echtzeit-Programme zum Messen, Steuern und Regeln können mit Standard-Tools auf dem PC entwickelt werden (in C oder Pascal)
- Remote-Debugging über Host-Schnittstelle
- Versorgungsspannung: 24V (18,6V....36V)
- Montierbar auf DIN-Schiene oder ebener Fläche

2.5.2. Blockschaltbild



2.5.3. Einsetzbare MAX-Module

Das Gerät weist insgesamt 5 Modul-Steckplätze Slot 1...5 auf. Sie können z.B. wie in folgender Tabelle beschrieben bestückt werden.

Slot-Nr.	mögliche Module	Funktion	Erläuterung
1	X-DPS-2i,	2 x PROFIBUS-DP Slave, isoliert	Die Host-Schnittstelle ist auf eine 9-polige D-Sub-Buchse geführt.
	X-DPM-1i	PROFIBUS-DP Master/Slave, isoliert	
	X-COM-4,	Seriell RS232, RS422, RS485	
	X-COM-8i,	” (isoliert)	
	X-SCC-2	”	
	X-CAN-2i	2 x CAN, isoliert	
	X-MAX-1	CPU-Modul mit RS232	

Slot-Nr.	**mögliche Module**	**Funktion**	**Erläuterung**
2 3 4	alle MAX-Module	Analog- und Digital-I/O, Zähler, Inkrementalgeber, SSI, serielle Schnittstellen, etc.	Die 40 I/O-Pins dieser 3 Module sind an 120 Schraubklemmen geführt.
5	X-MAX-1	CPU-Modul	Dieser Steckplatz ist für ein CPU-Modul vorgesehen. Falls sich auf Steckplatz 1 schon ein CPU-Modul befindet, kann hier auch ein zusätzliches CPU-Modul zur Erhöhung der Rechenleistung aufgesteckt werden. Mindestens ein CPU-Modul ist erforderlich.

2.5.4. Technische Daten

Parameter	**min.**	**typ.**	**max.**	**Einheit**	**Anmerkung**
Versorgungsspannung	18,6	24	36	V	
Leistungsaufnahme (abhängig von den aufgesteckten MAX-Modulen)	3		26	W	min.: ohne Module max.: begrenzt durch DC/DC-Wandler
Galvanische Trennung		500		V	zwischen Versorgungs-spannung und allen I/O-Signalen
Umgebungstemperatur	0	20	60	°C	bei waagrechter Montage, sonst max. 40°C
Anschlußleiterquerschnitt	0,5		1,5	mm^2	Schutzleiter PE bis $2{,}5mm^2$
Abmessungen Breite Höhe Tiefe		235 130 60		mm mm mm	Komplettes Gerät, Elektronikblock auf Terminalblock aufgesteckt
Schutzart		IP20			

2.5.5. Belegung der Host-Schnittstelle

Für die Host-Schnittstelle hat das Gerät eine 9-polige D-Sub-Buchse. Sie kann für PROFIBUS-DP direkt verwendet werden. Für CAN-Bus muß ein Adapter, für serielle Schnittstelle RS232 ein beidseitig männlicher Gender Changer aufgesteckt werden.

Je nach verwendetem Kommunikationsmodul ist diese Steckverbindung wie in der folgenden Tabelle beschrieben belegt. Hierbei ist zu beachten, daß bei den seriellen Schnittstellen auch unterschiedliche Betriebsarten möglich sind und einige Signale unterschiedliche Funktion haben können (siehe auch Datenblatt des verwendeten Kommunikationsmoduls).

Pin-Nr.	**PROFIBUS-DP** **mit X-DPS-2i** **X-DPM-1i**	**CAN-Bus(2)** **mit X-CAN-2i**	**Ethernet** **mit X-ETH-10**	**seriell, asynchron RS232 (1)** **mit X-MAX-1** **X-SCC-2/U** **X-SCC-2/R**	**seriell RS422(1)** **mit X-SCC-2/U**	**seriell RS485(1)** **mit X-SCC-2/U**
1	DP_PE	CAN_GND	TD+	DCD	RVC-	–
2	–	CAN_H	RD+	RxD	RCV+	–
3	DP_B	–	Absch.	TxD	TMT-	D-
4	DP_RTS	CAN_V5(3)	–	DTR	TMT+	D+
5	DP_GND	–	–	GND	GND	GND
6	DP_5V	CAN_L	TD-	DSR	CTS-/CLKin-	CTS-/CLKin-
7	–	CAN_GND	RD-	RTS	RTS-/CLKout-	RTS-/CLKout-
8	DP_A	–	–	CTS	CTS+/CLKin+	CTS+/CLKin+
9	–	CAN_GND	–	RI	RTS+/CLKout+/ Ri	RTS+/CLKout+/ Ri

(1) mit aufgestecktem Gender Changer
(2) mit aufgestecktem Adapter
(3) optional

2.5.6. Belegung der Schraubklemmen

Der nebenstehenden Darstellung kann man entnehmen, wie die I/O-Leitungen der Modulsteckplätze 2, 3 und 4 mit den Schraubklemmen verbunden sind.

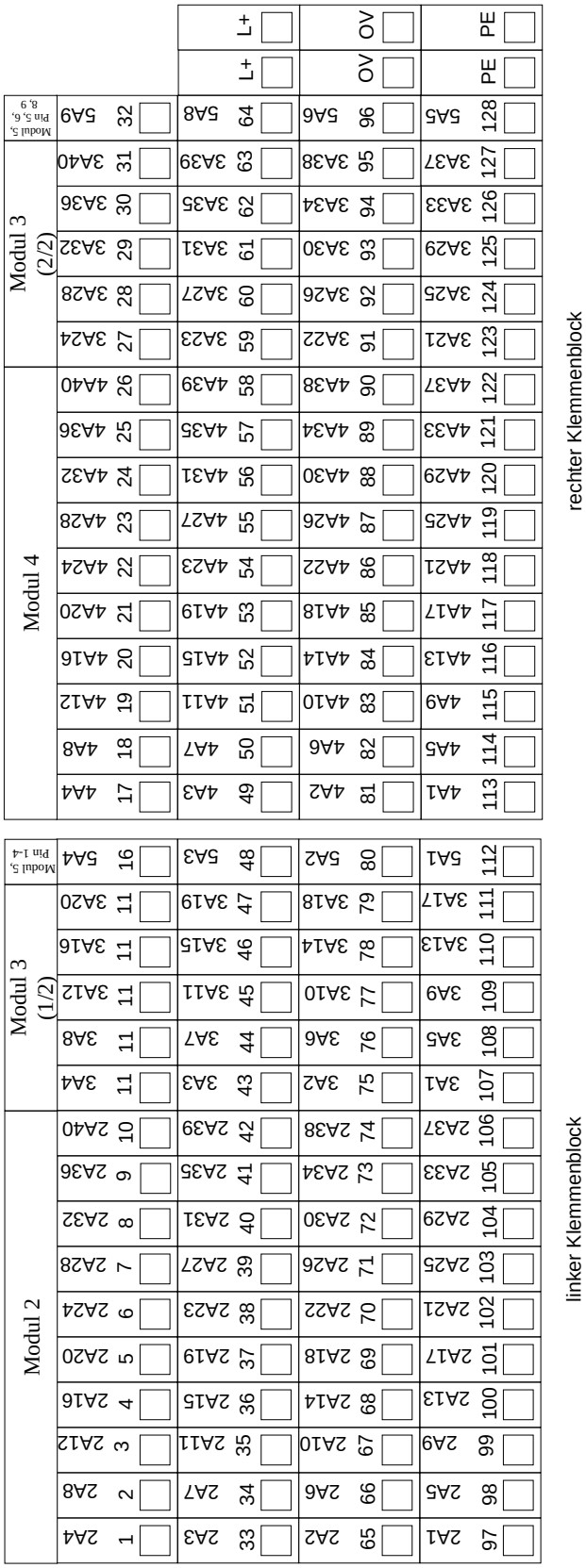
Die Anschlußbezeichnungen haben z.B. folgende Bedeutung:

2A23 = Modul-Steckplatz 2, Pin A23
Die tatsächliche Funktion des jeweiligen Anschlusses hängt von dem aufgesteckten MAX-Modul ab und kann dem jeweiligen Datenblatt entnommen werden.

L+ = Versorgungsspannung

0V = Bezugspotential der Versorgungsspannung

PE = Anschluß für Schutzleiter



2.5.7. Belegung der Jumper

Auf dem MXA5dip sind 3 Jumper vorhanden.
JP1 dient zur Verbindung des X-Bus Ground mit dem PE-Leiter. JP2 und JP3 aktivieren die Batteriepufferung für den Modul-Steckplatz 1 bzw. 5.

Jumper	Position	Funktion
1	1-2	PE mit XGND verbunden
1	2-3 (*)	PE nicht mit XGND verbunden
2	1-2	Batteriepufferung für Modul-Steckplatz 1 aktiviert
2	2-3 (*)	Keine Batteriepufferung für Modul-Steckplatz 1
3	1-2	Batteriepufferung für Modul-Steckplatz 5 aktiviert
3	2-3 (*)	Keine Batteriepufferung für Modul-Steckplatz 5

2.5.8. Remote-Verbindungen

2.5.8.1. Serielle Remote-Verbindung

Führen Sie die nachfolgenden Schritte aus, um eine serielle Remote-Verbindung aufzubauen:

- Installieren Sie auf dem PC zunächst den Windows-Treiber und das Programm SNW32.
- Starten Sie auf dem PC das Symbol *SORCUS boards* in der Windows-Systemsteuerung.
- Jetzt können Sie in der Kartenliste mit der rechten Maustaste auf eine freie Karte klicken und aus dem Kontextmenü *New remote connection* auswählen. Als *Type* wählen Sie *Serial connection* und als *Board* wählen Sie *MAX-PC*.
- Die nachfolgenden Dialoge ermöglichen Ihnen dann, eine Verbindung einzurichten. Wählen Sie hier die entsprechende serielle Schnittelle des PC. Die Schnittstelle muss folgendermaßen konfiguriert werden: Speed=38400; Parity=none; Stop bits=1.
- Nach diesem Vorgang erscheint in der Liste der Eintrag *MAX-PC*.
- Verbinden Sie die serielle Schnittstelle des X-MAX-1 bzw. X-MAX-E über eine Nullmodem-Verbindung (RCV und TMT Leitungen kreuzen, GND Leitungen verbinden) mit der seriellen Schnittstelle des PCs:
  Wenn Sie das X-MAX-1 auf Steckplatz 1 aufstecken, können Sie ein Standard-Nullmodemkabel (unter Zuhilfenahme eines Gender-Changers) verwenden. Wenn Sie das X-MAX-E auf Steckplatz 1 aufstecken, verwenden Sie bitte das Kabel KX-3943.
  Auf den anderen Steckplätzen müssen Sie die entsprechenden Schraubklemmen verwenden.
- Schalten Sie nun das MAX5dip ein. Das MAX5dip ist nun betriebsbereit.
- Starten Sie nun SNW32 und wählen Sie das MAX5dip aus. Die X-Bus Module auf dem MAX5dip erscheinen in einer Baumstruktur.

2.5.8.2. Ethernet Remote-Verbindung

Führen Sie die nachfolgenden Schritte aus, um eine Ethernet Remote-Verbindung aufzubauen:

- Installieren Sie auf dem PC zunächst den Windows-Treiber und das Programm SNW32.
- Starten Sie auf dem PC das Symbol *SORCUS boards* in der Windows-Systemsteuerung.
- Jetzt können Sie in der Kartenliste mit der rechten Maustaste auf eine freie Karte klicken und aus dem Kontextmenü *New remote connection* auswählen. Als *Type* wählen Sie *TCP/IP connection to MAX-PC*.
- Im nachfoldenden Dialog müssen Sie die IP-Adresse für die Verbindung eintragen. Wenn Sie das MAX5dip innhalb eines Netzwerkes einsetzen, erfragen Sie bitte eine freie IP-Adresse bei Ihrem Netzwerkadministrator.
- Nach diesem Vorgang erscheint in der Liste der Eintrag *MAX-PC*.
- Um eine Remote-Verbindung des MAX5dip per Ethernet aufzubauen, müssen auf dem „CPU-Modul“ (X-MAX-1 bzw. X-MAX-E) zunächst entsprechende Treiber installiert werden und auf dem „Ethernet-Modul“ (X-ETH-10 bzw. X-MAX-E) muss die IP-Adresse eingestellt werden. Für diesen Konfigurationsvorgang muss einmalig eine serielle Remote-Verbindung aufgebaut werden oder das „CPU-Modul“ und das „Ethernet-Modul“ müssen dafür auf eine MAX6pci Karte aufgesteckt werden:
- Serielle Remote-Verbindung: Wenn Sie das X-ETH-10 Modul als „Ethernet-Modul“ verwenden, dann stecken Sie das X-ETH-10 Modul auf Steckplatz 2 auf (zusätzlich zum „CPU-Modul“ auf Steckplatz 1).
- MAX6pci: Stecken Sie das „CPU-Modul“ und das „Ethernet-Modul“auf beliebige freie Steckplätze.
- Starten Sie SNW32 und wählen Sie dort das seriell angekoppelte MAX5dip bzw. die MAX6pci-Karte aus.
- Starten Sie den SNW32-Assistenten für das „Ethernet-Modul“ (X-ETH-10 bzw. X-MAX-E) (durch einen Doppelklick auf das Modul) und wählen Sie die Seite *TCP/IP Configuration* aus. Tragen Sie hier die IP-Adresse ein, die Sie zuvor in *SORCUS boards* in der Systemsteuerung eingetragen haben.
- Starten Sie nun den SNW32-Assistenten für das „CPU-Modul“ (X-MAX-1 bzw. X-MAX-E) und wählen Sie die Seite *System driver* aus. Auf dieser Seite werden

alle installierten System-Treiber angezeigt. Fügen Sie den Treiber *Ethernet Host Connection Driver* für die Ethernet Remote-Verbindung hinzu. Sie können bei der Konfiguration des Treibers das „Ethernet-Modul“ auswählen, über das die Ankopplung erfolgen soll, sowie dessen spätere (d.h., wenn es in der Ethernet Remote-Verbindung im Einsatz ist) Slot- und Layer-Nummer festlegen: hier sollte Slot=1 und Layer=0 eingetragen werden.

- Über den Punkt *Install into Flash* müssen Sie jetzt den Treiber in das Flash des „CPU-Moduls“ laden. Damit steht er nach jedem Reset automatisch zur Verfügung. Beenden Sie anschließend SNW32.

- Stecken Sie die Module auf das MAX5dip und stellen Sie eine Ethernet-Verbindung mit dem Rechner her:
  Verwenden Sie das X-ETH-10 Modul als „Ethernet-Modul“, dann stecken Sie dieses auf Steckplatz 1 und das „CPU-Modul“ auf einen beliebigen Steckplatz (vorzugsweise auf Steckplatz 5).
  Verwenden Sie das X-MAX-E als „Ethernet-Modul“ und „CPU-Modul“, dann stecken Sie dieses auf Steckplatz 1.
  Verbinden Sie die 9-polige D-Sub-Buchse des MAX5dip mit dem Kabel KX-3943 direkt mit dem PC bzw. mit einem Crossover-Kabel mit dem Netzwerk. Möchten Sie einen anderen Steckplatz verwenden, so müssen Sie das Kabel direkt an die Schraubklemmen anschließen.

- Schalten Sie nun das MAX5dip ein. Das MAX5dip ist nun betriebsbereit.

- Starten Sie nun SNW32 und wählen Sie das MAX5dip aus. Die X-Bus Module auf dem MAX5dip erscheinen in einer Baumstruktur.

2.6. MAX8dip: Dezentrale intelligente Peripherie mit 8 Steckplätzen

Das MAX8dip ist ein intelligentes, selbstständig arbeitendes, dezentrales Peripheriegerät. Es kann unabhängig von einem Host-Rechner zum Messen, Steuern und Regeln eingesetzt werden. Das kleine, kompakte Gehäuse ist für Hutschienenmontage vorgesehen, kann aber auch mit abmontierter Hutschienenhalterung als Tischgehäuse verwendet werden.

Zur Anpassung an die Anwendung ist es mit 8 MAX-Modulen bestückbar, von denen 6 die eigentlichen Peripheriefunktionen des MAX8dip festlegen. Die Verbindung mit der Außenwelt geschieht über insgesamt 240 Schraubklemmen.

Eine ebenfalls über ein aufsteckbares MAX-Modul konfigurierbare Kommunikationsschnittstelle zur Host-Anbindung erlaubt den Datenaustausch mit dem Host-System und z.B. den Download anwendungsbezogener Software ins lokale RAM oder ins Flash. Als Host-Schnittstelle möglich sind z.B. PROFIBUS-DP, CAN, Ethernet, USB, RS-232, RS-422, RS-485, etc.

Ein Modul-Steckplatz ist für ein CPU-Modul vorgesehen. Hier kann z.B. der MAX-PC, ein MAX-Modul mit 100 MHz 486 CPU, 16 MByte RAM und 16 MByte Flash aufgesteckt werden. Ein weiterer Steckplatz ist speziell für Kommunikationsmodule

vorgesehen. Hier stehen dem Anwender alle 40 Aussenwelt-Signale dieses Modul-Steckplatzes auf 3 D-SUB Steckverbindern zur Verfügung.

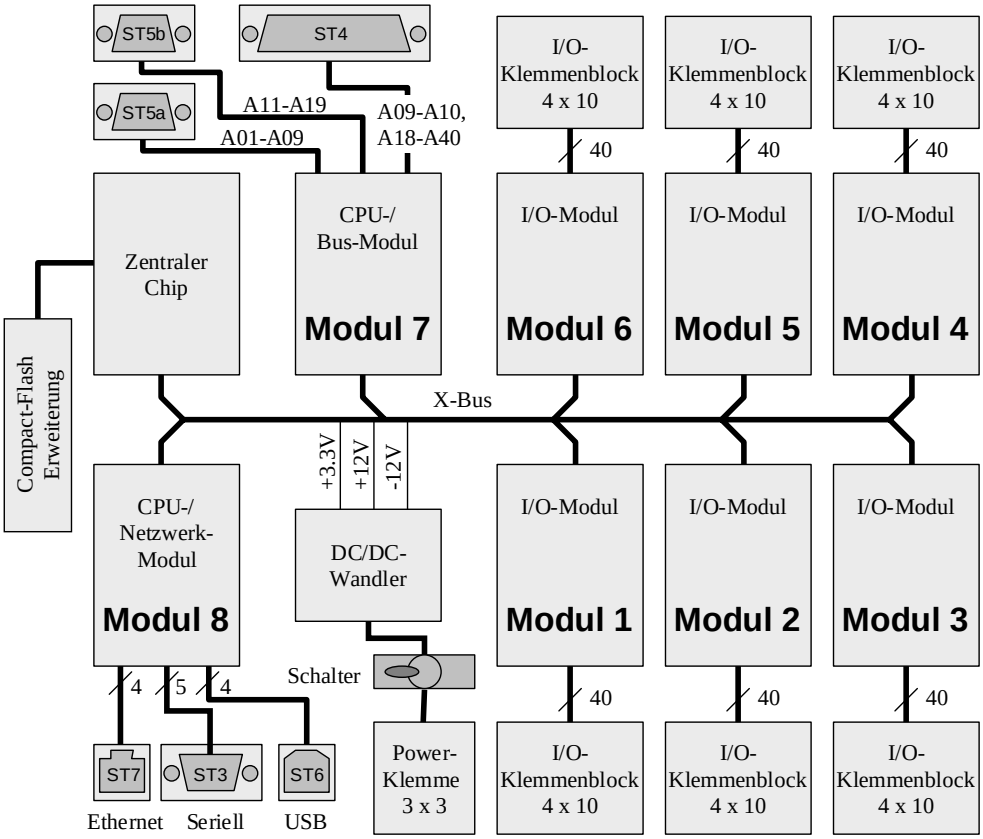
Auf den übrigen 6 Modulsteckplätzen können alle MAX-Module aufgesteckt werden.

Das MAX8dip wird mit einer Gleichspannung von 24V (18,6V bis 36V) versorgt. Alle intern benötigten Betriebsspannungen werden daraus galvanisch getrennt erzeugt. Mit einem Schalter kann das Gerät ein- bzw. ausgeschaltet werden.

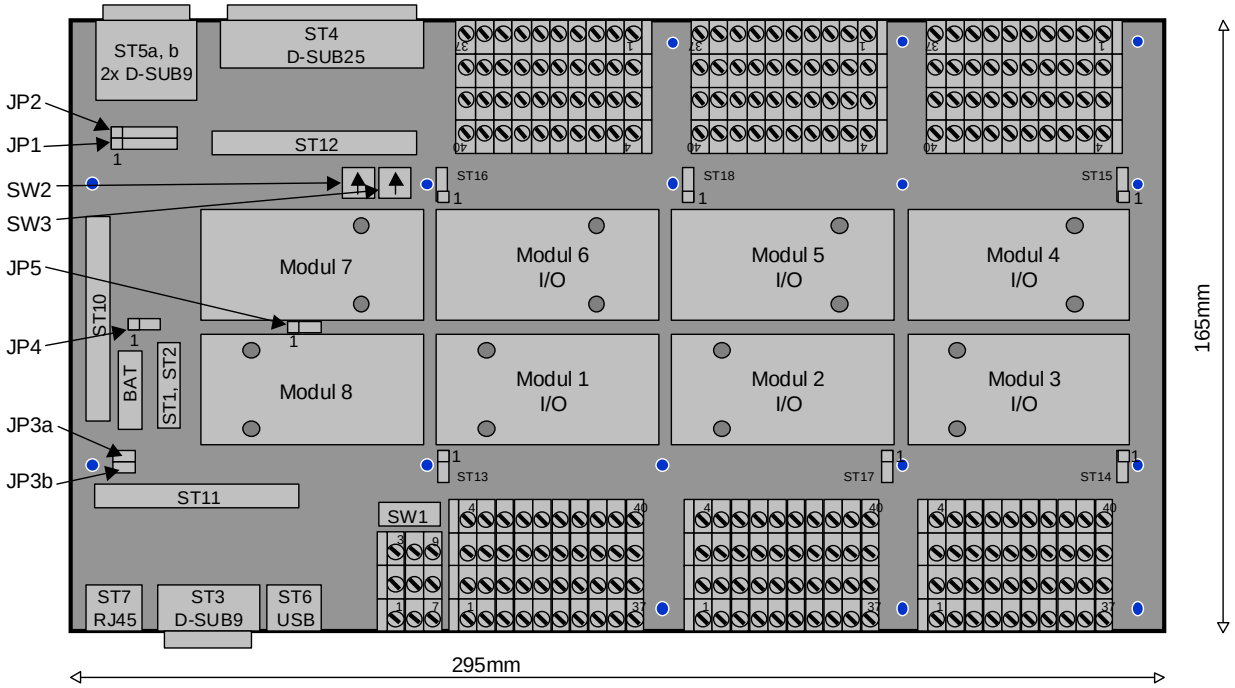
2.6.1. Besondere Eigenschaften

- 8 Steckplätze für MAX-Module, z.B. 6 I/O- Module, ein CPU-Modul und eines für die Host-Schnittstelle (CAN, PROFIBUS, Ethernet, etc.)
- Pro Steckplatz können zusätzlich 2 Module übereinander gestapelt werden, z.B. CPU-Modul und Display-Modul
- 240 (6 x 40) Schraubklemmen für den Anschluß von I/O-Signalen
- Erweiterbarkeit mit Compact-Flash Modul
- Echtzeit-Multi-Tasking Betriebssystem auf CPU-Modul
- Anwendungsbezogene Echtzeit-Programme zum Messen, Steuern und Regeln können mit Standard-Tools auf dem PC entwickelt werden (in C oder Pascal)
- Remote-Debugging über Host-Schnittstelle
- Versorgungsspannung: 24V (18,6V....36V)
- Montierbar auf DIN-Schiene oder ebener Fläche

2.6.2. Blockschaltbild



2.6.3. Lageplan der Module und Steckverbinder



2.6.4. Einsetzbare MAX-Module

Das Gerät weist insgesamt 8 Modul-Steckplätze auf. Sie können z.B. wie in folgender Tabelle beschrieben bestückt werden.

Slot-Nr.	mögliche Module	Funktion	Erläuterung
1-6	alle MAX-Module	Analog- und Digital-I/O, Zähler, Inkrementalgeber, SSI, serielle Schnittstellen, etc.	Die 40 I/O-Pins dieser 6 Module sind an einzelne Schraubklemmenblöcke geführt.
7	X-DPS-2i, X-DPM-1i	2 x PROFIBUS-DP Slave PROFIBUS-DP Master / Slave	Beide Schnittstellen des PROFIBUS bzw. CAN-Bus Moduls sind auf eigenen D-SUB-9 Stecker (St5a und St5b) gelegt.
	X-CAN-2i	2 x CAN	Bei CAN ist ein Adapterstecker notwendig
	X-COM-4, X-COM-8i, X-SCC-2	Seriell RS232, RS422, RS485	
	X-MAX-1	CPU-Modul mit RS232	Die Host-Schnittstelle ist auf die 9-polige D-Sub-Buchse St5a geführt.

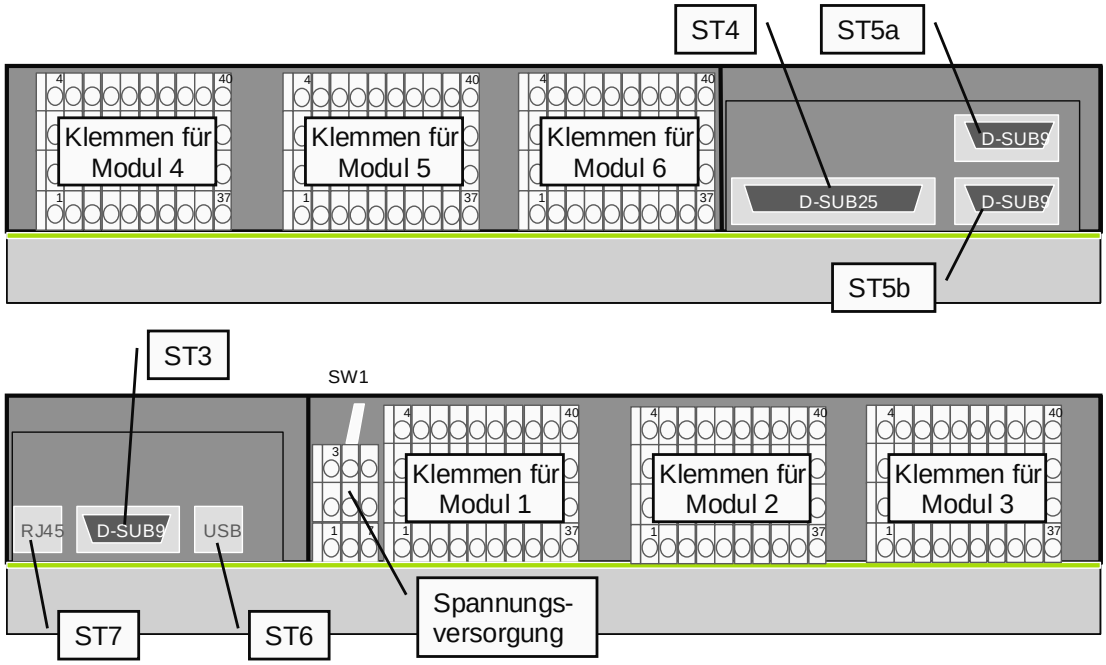
Slot-Nr.	mögliche Module	Funktion	Erläuterung
8	X-MAX-E, X-MAX-400 X-ETH-10, X-ETH-100	CPU-Modul mit Ethernet bzw. Ethernet-Modul	Dieser Steckplatz ist für ein MAX-Modul mit Ethernet vorgesehen. Falls sich auf Steckplatz 7 schon ein CPU-Modul befindet, kann hier auch ein zusätzliches CPU-Modul zur Erhöhung der Rechenleistung aufgesteckt werden. Mindestens ein CPU-Modul ist erforderlich.

2.6.5. Stecker und Stecker-Belegung des MAX8dip

Folgende Steckverbinder sind vorhanden:

Stecker / Buchse	Typ	Funktion
St1	Pfostenreihe 8-Polig	JTAG-Schnittstelle für Test
St2	Pfostenreihe 8-Polig	I²C-Schnittstelle für Test
St3	D-SUB-9 Stecker	Serielle Schnittstelle des CPU-Moduls X-MAX-E bzw. X-MAX-400
St4	D-SUB-25 Buchse	Signale von Modulsteckplatz 7
St5a	D-SUB-9 Buchse oben	Signale A01...A09 von Modulsteckplatz 7
St5b	D-SUB-9 Buchse unten	Signale A11...A19 von Modulsteckplatz 7
St6	USB-B	USB-Device-Schnittstelle des X-MAX-400 (zum Host-PC)
St7	RJ45	Ethernet-Schnittstelle des X-MAX-E, X-MAX-400, X-ETH-10 oder X-ETH-100

2.6.6. Steckeranordnung



2.6.7. Belegung der Stecker St3, St4, St5a, St5b, St6, St7

Die folgenden Tabellen geben die Zuordnung der Stecker zu den Modulen an. M7.A01 bezeichnet z.B. den Pin 1 am Stecker A des Moduls auf dem Steckplatz 7. In Klammern sind die Funktionen der Stecker-Pins angegeben, wenn ein CPU-Modul auf Steckplatz 8 aufgesteckt ist.

Pin	St3 (bei X-MAX-E und X-MAX400)	St5a	St5b	St7 Ethernet (bei X-MAX-E, X-ETH-x und X-MAX400)	St6 USB (bei X-MAX400)
1	-	M7.A01	M7.A11	M8.A01 (TD+)	M8.A28 (Vbus)
2	M8.A06 (RXD)	M7.A03	M7.A13	M8.A02 (TD–)	M8.A29 (D–)
3	M8.A08 (TXD)	M7.A05	M7.A15	M8.A03 (RD+)	M8.A30 (D+)
4	-	M7.A07	M7.A17	-	GND
5	M8.A05 (GND)	M7.A09	M7.A19	-	-
6	-	M7.A02	M7.A12	M8.A04 (RD–)	-
7	M8.A07 (RTS)	M7.A04	M7.A14	-	-
8	M8.A09 (CTS)	M7.A06	M7.A16	-	-
9	-	M7.A08	M7.A18	-	-

2.6.7.1. Stecker St3

Am Stecker St3 liegt die serielle Host-Schnittstelle des CPU-Moduls (X-MAX-E oder X-MAX-400) von Steckplatz 8. Diese kann direkt mit einem Null-Modem Kabel verwendet werden.

2.6.7.2. Stecker St4

Der Stecker St4 ist mit den übrigen Signalen von Modulsteckplatz 7 belegt, die nicht an den Steckern St5a und St5b vorhanden sind. Die Modulanschlüsse A09, A18 und A19 sind sowohl auf dem Stecker ST4, als auch auf dem Stecker St5a bzw. St5b vorhanden.

Pin	St4	Pin	St4
1	M7.A09	14	M7.A10
2	M7.A18	15	M7.A19
3	M7.A20	16	M7.A21
4	M7.A22	17	M7.A23
5	M7.A24	18	M7.A25
6	M7.A26	19	M7.A27
7	M7.A28	20	M7.A29
8	M7.A30	21	M7.A31
9	M7.A32	22	M7.A33
10	M7.A34	23	M7.A35
11	M7.A36	24	M7.A37
12	M7.A38	25	M7.A39
13	M7.A40		

2.6.7.3. Stecker St5a und St5b

Die Stecker St5a und St5b können für PROFIBUS-DP direkt verwendet werden. Für eine serielle Schnittstelle RS232 muß ein beidseitig männlicher Gender Changer aufgesteckt werden. Für CAN-Bus ist ein Adapterstecker notwendig. Je nach verwendetem Kommunikationsmodul ist dieser Stecker wie in der folgenden Tabelle beschrieben belegt. Hierbei ist zu beachten, daß bei den seriellen Schnittstellen auch unterschiedliche Betriebsarten möglich sind und einige Signale unterschiedliche Funktion haben können (siehe auch Datenblatt des verwendeten Kommunikationsmoduls). Wenn auf Steckplatz 7 ein X-MAX-1 steckt, kann mit St5a die Hostverbindung hergestellt werden.

Pin-Nr.	PROFIBUS-DP mit X-DPS-2i X-DPM-1i	CAN-Bus(3) mit X-CAN-2i	Ethernet mit X-ETH-10	seriell, asynchron RS232 (1) mit X-MAX-1 X-SCC-2/x	seriell RS422(1) mit X-SCC-2/U	seriell RS485(1) mit X-SCC-2/U
1	DP_PE	CAN_GND	TD+	DCD	RVC-	–
2	–	CAN_H	RD+	RxD	RCV+	–
3	DP_B	–	Absch.	TxD	TMT-	D-
4	DP_RTS	CAN_V5(2)	–	DTR	TMT+	D+
5	DP_GND	–	–	GND	GND	GND
6	DP_5V	CAN_L	TD-	DSR	CTS-/CLKin-	CTS-/CLKin-
7	–	CAN_GND	RD-	RTS	RTS-/CLKout-	RTS-/CLKout-
8	DP_A	–	–	CTS	CTS+/CLKin+	CTS+/CLKin+

Pin-Nr.	PROFIBUS-DP mit X-DPS-2i X-DPM-1i	CAN-Bus(3) mit X-CAN-2i	Ethernet mit X-ETH-10	seriell, asynchron RS232 (1) mit X-MAX-1 X-SCC-2/x	seriell RS422(1) mit X-SCC-2/U	seriell RS485(1) mit X-SCC-2/U
9	–	CAN_GND	–	RI	RTS+/CLKout+/ Ri	RTS+/CLKout+/ Ri

(1) mit aufgestecktem Gender Changer
(2) optional
(3) mit aufgestecktem Adapter

2.6.8. Belegung der Jumper

2.6.8.1. Jumper JP1, JP2 und JP4: Wake-Up Funktionalität (bei MAX8dip/SD)

Hiermit kann beim MAX8dip/SD die Standbyfunktion aktiviert werden. Mit Jumper JP1 kann das Wake-Up Signal und die Standby-Spannungsversorgung für den Modulsteckplatz 7 eingestellt werden. Jumper JP2 stellt die selben Funktionen für Steckplatz 8 ein. Ob das aufgesteckte MAX-Modul eine Wake-Up Funktion unterstützt und eine Stand-By Versorgung nutzen kann, muss dem jeweiligen Datenblatt entnommen werden. Wenn das aufgesteckte Modul keine Standby-Versorgung unterstützt, darf der Jumper JP1 bzw. JP2 **nicht** in Position 5-6 aufgesteckt werden, da sonst das MAX8dip beschädigt werden kann!

Mit Jumper JP4 kann die Standby- und Wake-Up Funktion konfiguriert werden.

Jumper	Position	Funktion
JP1	1-2 (*)	Keine Wake-Up Funktion durch Modulsteckplatz 7
JP2	1-2 (*)	Keine Wake-Up Funktion durch Modulsteckplatz 8
JP1	2-3	Wake-Up Funktion durch Modulsteckplatz 7 aktiviert
JP2	2-3	Wake-Up Funktion durch Modulsteckplatz 8 aktiviert
JP1	4-5 (*)	Keine Standby-Versorgung des Modulsteckplatz 7
JP2	4-5 (*)	Keine Standby-Versorgung des Modulsteckplatz 8
JP1	5-6	Standby-Versorgung auf Modulsteckplatz 7 geschaltet
JP2	5-6	Standby-Versorgung auf Modulsteckplatz 8 geschaltet

Jumper	Position	Funktion
JP4	1-2 (*)	Das MAX8dip/SD kann nicht in den Standby-Modus geschaltet werden, (erstes) Wake-Up durch ein Modul ist möglich.
JP4	2-3	Das MAX8dip/SD kann per Software den Standby-Modus geschaltet werden, Wake-Up durch ein Modul ist möglich.

(*) Defaulteinstellung

2.6.8.2. Jumper JP3a und JP3b: Batteriepufferung

Hiermit kann beim MAX8dip (ab Rev. B) eine Batteriepufferung für Module, die auf Steckplatz 7 oder 8 aufgesteckt sind, aktiviert werden. Ob das aufgesteckte MAX-Modul eine Batteriepufferung unterstützt, kann dem jeweiligen Datenblatt entnommen werden.

Jumper	Funktion
3a	Batteriepufferung für Modulsteckplatz 7
3b	Batteriepufferung für Modulsteckplatz 8

2.6.8.3. Jumper JP5: Erhöhung der Versorgungsspannung

Dieser Jumper ist nicht bestückt.

2.6.9. Funktion der Schalter

2.6.9.1. Schalter S1: Hauptschalter

Hiermit wird das MAX8dip eingeschaltet. Ein MAX8dip/SD kann anschliessend per Software in den Standby-Modus geschaltet werden.

2.6.9.2. Schalter S2 und S3: Drehcodierschalter für Adress-einstellung

Diese Schalter sind nicht bestückt.

2.6.10. Belegung der Schraubklemmen

2.6.10.1. Spannungsversorgung (Klemme 7)

Die Spannungsversorgung erfolgt galvanisch getrennt über einen eingebauten DC/DC-Wandler. Der Eingangsspannungsbereich beträgt 18-36V Gleichspannung. An den L+ Schraubklemmen (3, 6 oder 9) wird die Versorgungsspannung angeschlossen. An den 0V Schraubklemmen (2, 5 oder 8) wird das Bezugspotential der Versorgungsspannung angeschlossen. An den PE Schraubklemmen (1, 4 oder 7) kann ein Schutzleiter angeschlossen werden, der mit den Abschirmungen der diversen Stecker verbunden ist.

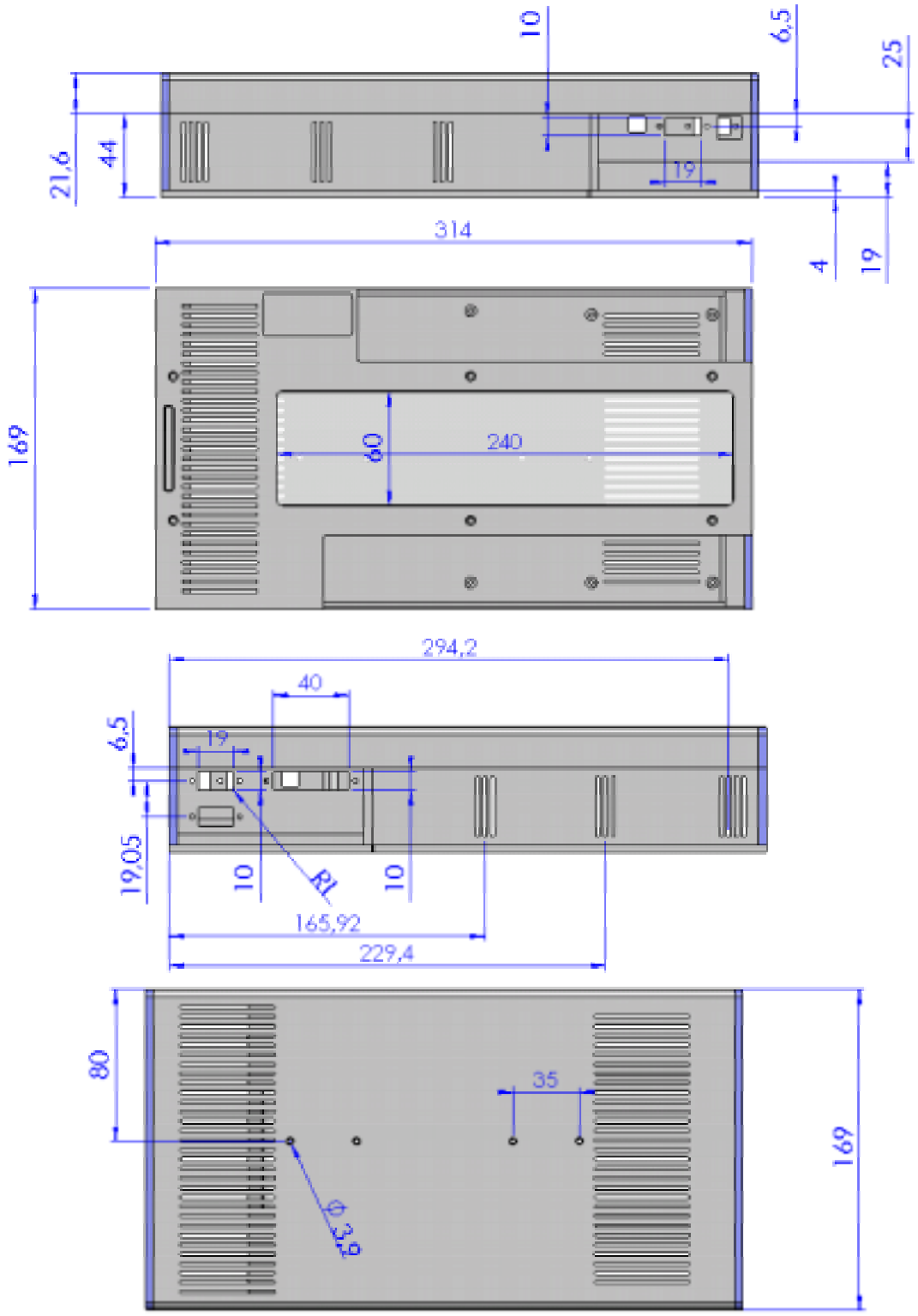
2.6.10.2. Modul I/O (Klemmen 1-6)

Der jeweils neben deb Modulsteckplätzen 1 bis 6 plazierte Schraubklemmenblock ist mit dem Stecker A des zugehörigen Moduls 1:1 verbunden, d.h. die Schraubklemme 1 neben Modulsteckplatz 1 ist mit dem Anschlußpin A1 des Moduls auf Steckplatz 1 verbunden. Die tatsächliche Funktion des jeweiligen Anschlusses hängt von dem aufgesteckten MAX-Modul ab und kann dem jeweiligen Datenblatt entnommen werden.

2.6.11. Sicherung

Zum Schutz des Gerätes vor Kurzschluss oder Überlastung ist eine Stecksicherung (4A träge) eingesetzt. Falls die Sicherung zerstört wird, ist zuerst der Schalter 1 auszuschalten und das MAX8dip auf Kurzschlüsse oder defekte Max-Module zu überprüfen. Ersetzen Sie die Sicherung nur durch eine gleiche Ausführung.

2.6.12. Gehäuse



2.6.13. Technische Daten

Parameter	min.	typ.	max.	Einheit	Anmerkung
Versorgungsspannung	18,6	24	36	V	
Leistungsaufnahme (abhängig von den aufgesteckten MAX-Modulen)	3 3		33 13	W W	min.: ohne Module max.: begrenzt durch DC/DC-Wandler Bei MAX8dip/SST
Galvanische Trennung		500		V	zwischen Versorgungs-spannung und allen I/O-Signalen
Umgebungstemperatur	0	20	60	°C	bei waagrechter Hutschienen-Montage, sonst max. 40°C
Anschlußleiterquerschnitt	0,5		1,5	mm^2	Schutzleiter PE bis $2{,}5mm^2$
Abmessungen Breite Höhe Tiefe		314 169 66		mm mm mm	Komplettes Gerät, ohne Hutschienen-halterung
Schutzart	IP20				

2.6.14. Remote-Verbindungen

2.6.14.1. Serielle Remote-Verbindung

Führen Sie die nachfolgenden Schritte aus, um eine serielle Remote-Verbindung aufzubauen:

- Installieren Sie auf dem PC zunächst den Windows-Treiber und das Programm SNW32.

- Starten Sie auf dem PC das Symbol *SORCUS boards* in der Windows-Systemsteuerung.
- Jetzt können Sie in der Kartenliste mit der rechten Maustaste auf eine freie Karte klicken und aus dem Kontextmenü *New remote connection* auswählen. Als *Type* wählen Sie *Serial connection* und als *Board* wählen Sie *MAX-PC*.
- Die nachfolgenden Dialoge ermöglichen Ihnen dann, eine Verbindung einzurichten. Wählen Sie hier die entsprechende serielle Schnittelle des PC. Die Schnittstelle muss folgendermaßen konfiguriert werden: Speed=38400; Parity=none; Stop bits=1.
- Nach diesem Vorgang erscheint in der Liste der Eintrag *MAX-PC*.
- Stecken Sie nun das „CPU-Modul“ auf das MAX8dip auf und stellen Sie eine Nullmodem-Verbindung (RCV und TMT Leitungen kreuzen, GND Leitungen verbinden) mit der seriellen Schnittstelle des PCs her:
  X-MAX-1:
  Stecken Sie das X-MAX-1 auf Steckplatz 7 und verbinden Sie die 9-polige D-Sub-Buchse ST5A des MAX8dip (unter Zuhilfenahme eines Gender-Changers) mit einem Nullmodemkabel mit der seriellen PC-Schnittstelle.
  X-MAX-E:
  Möglichkeit A: Stecken Sie das X-MAX-E auf Steckplatz 8 und verbinden Sie den 9-poligen D-Sub-Stecker ST3 des MAX8dip mit einem Nullmodem-Kabel mit der seriellen PC-Schnittstelle.
  Möglichkeit B: Stecken Sie das X-MAX-E auf Steckplatz 7 und verbinden Sie die 9-polige D-Sub-Buchse ST5A des MAX8dip mit dem Kabel KX-3943 mit der seriellen PC-Schnittstelle.
  Auf den anderen Steckplätzen müssen Sie die entsprechenden Schraubklemmen verwenden.
- Schalten Sie nun das MAX8dip ein. Das MAX8dip ist nun betriebsbereit.
- Starten Sie nun auf dem PC SNW32 und wählen Sie das MAX8dip aus. Die X-Bus Module auf dem MAX8dip erscheinen in einer Baumstruktur.

2.6.14.2. Ethernet Remote-Verbindung

Führen Sie die nachfolgenden Schritte aus, um eine Ethernet Remote-Verbindung aufzubauen:

- Installieren Sie auf dem PC zunächst den Windows-Treiber und das Programm SNW32.

- Starten Sie auf dem PC das Symbol *SORCUS boards* in der Windows-Systemsteuerung.
- Jetzt können Sie in der Kartenliste mit der rechten Maustaste auf eine freie Karte klicken und aus dem Kontextmenü *New remote connection* auswählen. Als *Type* wählen Sie *TCP/IP connection to MAX-PC*.
- Im nachfolgenden Dialog müssen Sie die IP-Adresse für die Verbindung eintragen. Wenn Sie das MAX8dip innerhalb eines Netzwerkes einsetzen, erfragen Sie bitte eine freie IP-Adresse bei Ihrem Netzwerkadministrator.
- Nach diesem Vorgang erscheint in der Liste der Eintrag *MAX-PC*.
- Um eine Remote-Verbindung des MAX8dip per Ethernet aufzubauen, müssen auf dem „CPU-Modul“ (X-MAX-1 bzw. X-MAX-E) zunächst entsprechende Treiber installiert werden und auf dem „Ethernet-Modul“ (X-ETH-10 bzw. X-MAX-E) muss die IP-Adresse eingestellt werden. Für diesen Konfigurationsvorgang muss einmalig eine serielle Remote-Verbindung aufgebaut werden, oder das „CPU-Modul“ und das „Ethernet-Modul“ müssen dafür auf eine MAX6pci Karte aufgesteckt werden:
- Serielle Remote-Verbindung: Wenn Sie das X-ETH-10 Modul als „Ethernet-Modul“ verwenden, dann stecken Sie das X-ETH-10 Modul auf Steckplatz 8 auf (zusätzlich zum „CPU-Modul“ auf Steckplatz 7).
- MAX6pci: Stecken Sie das „CPU-Modul“ und das „Ethernet-Modul“ auf beliebige freie Steckplätze.
- Starten Sie SNW32 und wählen Sie dort das seriell angekoppelte MAX8dip bzw. die MAX6pci-Karte aus.
- Starten Sie den SNW32-Assistenten für das „Ethernet-Modul“ (X-ETH-10 bzw. X-MAX-E) (durch einen Doppelklick auf das Modul) und wählen Sie die Seite *TCP/IP Configuration* aus. Tragen Sie hier die IP-Adresse ein, die Sie zuvor in *SORCUS boards* in der Systemsteuerung eingetragen haben.
- Starten Sie nun den SNW32-Assistenten für das „CPU-Modul“ (X-MAX-1 bzw. X-MAX-E) und wählen Sie die Seite *System driver* aus. Auf dieser Seite werden alle installierten System-Treiber angezeigt. Fügen Sie den Treiber *Ethernet Host Connection Driver* für die Ethernet Remote-Verbindung hinzu. Sie können bei der Konfiguration des Treibers das „Ethernet-Modul“ auswählen, über das die Ankopplung erfolgen soll, sowie dessen spätere (d.h., wenn es in der Ethernet Remote-Verbindung im Einsatz ist) Slot- und Layer-Nummer festlegen: hier sollte Slot=8 und Layer=0 eingetragen werden.

- Über den Punkt *Install into Flash* müssen Sie jetzt den Treiber in das Flash des „CPU-Moduls“ laden. Damit steht er nach jedem Reset automatisch zur Verfügung. Beenden Sie anschließend SNW32.
- Stecken Sie die Module auf das MAX8dip und stellen Sie eine Verbindung mit dem Rechner her:
  Verwenden Sie das X-ETH-10 Modul als „Ethernet-Modul“, dann stecken Sie dieses auf Steckplatz 8 und dsas „CPU-Modul“ auf Steckplatz 7. Verbinden Sie die RJ45 Buchse des MAX8dip mit einem Crossover-Kabel mit dem PC bzw. mit einem Patch-Kabel mit dem Netzwerk.
  Verwenden Sie das X-MAX-E als „Ethernet-Modul“ und „CPU-Modul“, dann stecken Sie dieses auf Steckplatz 8. Verbinden Sie die RJ45 Buchse des MAX8dip mit einem Crossover-Kabel mit dem PC bzw. mit einem Patch-Kabel mit dem Netzwerk.
  Auf den anderen Steckplätzen müssen Sie die entsprechenden Schraubklemmen verwenden.
- Schalten Sie nun das MAX8dip ein. Das MAX8dip ist nun betriebsbereit.
- Starten Sie nun SNW32 und wählen Sie das MAX8dip aus. Die X-Bus Module auf dem MAX8dip erscheinen in einer Baumstruktur.

2.7. X-KiT-3

2.7.1. Konfiguration der Jumper JP1 bis JP10 des X-KiT-3

Die 10 Jumper sind mit BAT, JTAG, BOOT, IRDA, HOST, DISPLAY, DESIGN, EXT.JTAG, IRQ0EN und RESERV. bezeichnet.

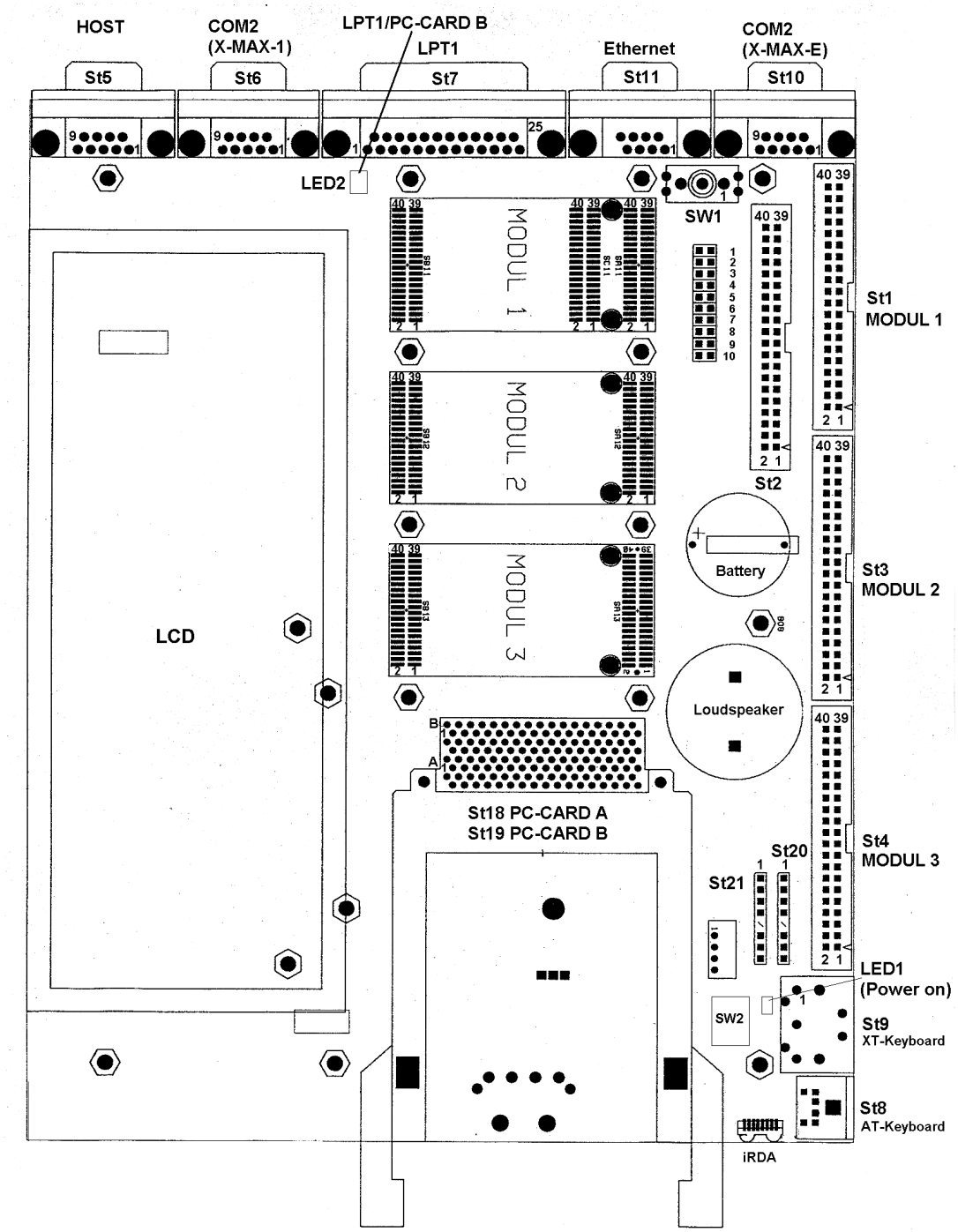
Jumper	**Nr.**	**Device**	**Funktion ohne Jumper**	**Funktion mit Jumper**
BAT	JP2	On-board Batterie	off	on
JTAG	JP7	Pins 10..13 von St2	JTAG	PCMCIA Card A
BOOT	JP1	Boot (erster opcode fetch nach Reset)	von on-board Flash	von PCMCIA Card A
IRDA	JP5	iRDA Transceiver	Enabled	Disabled
HOST	JP4	On-board Host-Schnittstelle	Enabled	Disabled
		Pin 10 von St1 (= HOST)	Enabled	Disabled
DISPLAY	JP3	On-board Display	On	Off
DESIGN	JP8	CPLD-Design Version von X-KiT-3	Disabled	Enabled
EXT.JTAG	JP6	JTAG-Kette (an St21)	Modul 1 .. 2 .. 3	Modul 1 .. 2 .. 3 .. CPU
IRQ0EN	JP9	Pin IRQ0 (= Pin 20 von St1)	Verwendet von Touch-Screen Controller	extern nutzbar
RESERV.	JP10	-	Keine	Keine

2.7.2. Versorgungsspannung des X-KiT-3

Die Versorgungsspannung für das X-KiT-3 wird über Stecker St23 oder St14 zugeführt. An Stecker St23 (Klinkenstecker) wird das mitgelieferte Netzteil (5 Volt) angeschlossen. Alternativ kann an Stecker St14 ein 4-pol. Anschlusskabel angeschlossen werden, so wie es für den Anschluss von CD-ROM- oder Floppy-Disk-Laufwerken verwendet wird.

Nur eine der beiden Spannungsquellen darf angeschlossen sein !

2.7.3. Lageplan des X-KiT-3, Rev. B



2.7.4. Steckerbelegung X-KiT-3

Folgende Stecker sind auf dem Evaluation-Board verfügbar:

Stecker/ Buchse	Typ auf dem Evaluation-Board	Funktion
St1[1]	40-pol. Pfostenstecker (2x20)	Anschluss von Stecker A von Modul 1 (MAX-PC)
St2[1]	40-pol. Pfostenstecker (2x20)	Anschluss von Stecker C von Modul 1 (MAX-PC)
St3	40-pol. Pfostenstecker (2x20)	Anschluss von Stecker A von Modul 2
St4	40-pol. Pfostenstecker (2x20)	Anschluss von Stecker A von Modul 3
St5	9-pol. D-Sub. Stecker	Serielle RS-232 Schnittstelle HOST, an Modul 1
St6	9-pol. D-Sub. Stecker	Serielle RS-232 Schnittstelle COM2 von Modul 1 (wenn Modul 1 = X-MAX-1)
St7	25-pol D-Sub. Buchse	Parallele Druckerschnittstelle von Modul 1
St8	6-pol. Mini-DIN Buchse	Anschluss für AT-Keyboard von Modul 1
St9	5-pol. DIN-Buchse	Anschluss für XT-Keyboard von Modul 1
St10	9-pol. D-Sub. Stecker	Serielle RS-232/-422/-485 Schnittstelle von Modul 1 (wenn Modul 1 = X-MAX-E)
St11	RJ45 (8-pol.) Buchse	Anschluss für Ethernet 10BaseT (wenn Modul 1 = X-MAX-E)
St12	33-pol. FPC-Steckver-binder (vorbereitet) TOP	Anschluss von LC-Display

[1] Die Pinbelegung entspricht nicht exakt der Pinbelegung des jeweiligen Moduls-Steckers, siehe 2.7.4.1.

Stecker/ Buchse	Typ auf dem Evaluation-Board	Funktion
St13	32-pol. FPC-Steckverbinder (vorbereitet) BOTTOM	Anschluss von LC-Display
St14[2]	4-pol. Stromversorgungsstecker	Anschluss von +5 Volt
St15	5-pol. Micro-Connector	Anschluss für Inverter für Display-Hintergrundbeleuchtung TDK CXA-K0505-HI
St16	6-pol. Micro-Connector	Anschluss für Inverter für Display-Hintergrundbeleuchtung Hitachi INV569
St17	24-pol. FPC-Steckverbinder	Anschluss für LC-Display Hitachi SX16M003-ZZA
St18	68-pol. PC-Card-Connector	Anschluss von PC-Card A
St19	68-pol. PC-Card-Connector	Anschluss von PC-Card B
St20	8-pol. Pfostenstecker	Anschluss von JTAG für FPGAs
St21	8-pol. Pfostenstecker	Anschluss von JTAG für Module
St22	8 Lötpads	Anschluss für Inverter für Display-Hintergrundbeleuchtung
St23[2]	2-pol. Klinkenbuchse	5 Volt Versorgungsspannung

Die hier angegebene Beschreibung der Pin-Funktionen bezieht sich auf das Evaluation-Board X-KiT-3, bestückt mit einem MAX-PC (X-MAX-1 bzw. X-MAX-E) auf Steckplatz 1. Die angegebenen Namen beziehen sich auf die Pinbelegung und Pinfunktionen nach Power-On bzw. nach Reset. Für viele Pins sind per Software anschließend andere Funktionen einstellbar.

Wenn ein Pin mehrere Funktionen unterstützt, z.B. digitaler Eingang, digitaler Ausgang, Interrupt-Eingang, etc., dann werden diese Funktionen per Software aktiviert durch Setzen von einem oder mehreren internen Registern des X-MAX-1 bzw. X-MAX-E.

Wenn ein Pin als digitaler Eingang verwendet werden kann, kann sein logischer Zustand jederzeit gelesen werden.

[2] Nur einer der beiden Stecker bzw. Buchsen darf im Betrieb gesteckt sein

Wenn ein Pin als digitaler Ausgang verwendet werden kann, kann er per Software gesetzt werden und der gesetzte Wert zurückgelesen werden.

Wenn ein Pin als Interrupt-Eingang (IRQx, NMI oder SMI) verwendet werden kann, muss er als digitaler Eingang konfiguriert werden (falls er auch eine andere Funktion haben kann). Zusätzlich kann ein IRQ Interrupt-Eingang intern mit einer der 15 Interrupt-Leitungen des MAX-PC verbunden werden.

Die Registerbeschreibung kann von SORCUS angefordert werden.

2.7.4.1. Stecker St1

Die meisten Pins entsprechen in ihrer Funktion denen von Stecker A von Modul X-MAX-1 bzw. X-MAX-E, bis auf die mit (*) gekennzeichneten Pins 14 und 22. Pin 14 von St1 ist auf dem X-KiT-3 nicht angeschlossen (wird für BOOT auf den MAX-PCs verwendet). Die mit (+) gekennzeichneten Pins sind 5-Volt-kompatibel (Pin 16), anders als bei X-MAX-1 oder X-MAX-E, wo die maximale Eingangs-Spannung bei vielen Pins 3,6 Volt beträgt.

Pin	**Name**	**Pin**	**Name**	**Pin**	**Name**	**Pin**	**Name**
1	DCD (X-MAX-1) TD+ (X-MAX-E)	6	CTS (X-MAX-1) RCV (X-MAX-E)	11	/WDO	16+	/BL1
2	DSR (X-MAX-1) TD- (X-MAX-E)	7	DTR (X-MAX-1) RTS (X-MAX-E)	12	/MRES	17	SIRIN
3	RCV (X-MAX-1) RD+ (X-MAX-E)	8	Ri (X-MAX-1) TMT (X-MAX-E)	13	SPKR	18	SIROUT
4	RTS (X-MAX-1) RD- (X-MAX-E)	9	GND (X-MAX-1) CTS (X-MAX-E)	14*	n.c. (BOOT)	19	LED
5	TMT (X-MAX-1) GND (X-MAX-E)	10	HOST	15	GP0	20	IRQ0

Pin	Name	Pin	Name	Pin	Name	Pin	Name
21	P3	26+	/ERROR	31	PD3	36+	PE
22*	BAT	27	PD1	32+	/ACK	37	PD6
23	/STRB	28	/INIT	33	PD4	38+	SLCT
24	/AFDT	29	PD2	34+	BUSY	39	PD7
25	PD0	30	/SLCTIN	35	PD5	40	GND

n.c. = not connected

RS-232 (Pins 1..9) bei X-MAX-1 bzw. 10BaseT und COM2 (Pins 1..4 und 5..9) bei X-MAX-E

Standard-Funktion (X-MAX-1): RS-232 Schnittstelle

Standard-Funktion (X-MAX-E): Pin A1..A4 stellen einen 10BaseT Ethernet Anschluss und die Pins A5..A9 das serielle Interface zur Verfügung. Die physikalische Schnittstelle kann softwaremäßig für RS-232/-422/-485 konfiguriert werden.

Von OsX wird die serielle Schnittstelle als Host-Schnittstelle Nr. 2 (COM2 mit IRQ-3) mit 38,4 kBaud initialisiert (8 Datenbit, 1 Stopbit, keine Parität).

2.7.4.2. HOST (Pin 10): Debug-Schnittstelle bzw. IO-Pin mit Interrupt

Der Pin ist direkt mit Pin A10 von Stecker A des X-MAX-1 bzw. X-MAX-E verbunden, seine Funktion ist abhängig von Jumper JP4 = „HOST“.

Standard-Funktion: Ein Jumper muss auf JP4 aufgesteckt werden (= HOST enabled). Dann wird der Pin intern für die Debug-Schnittstelle (siehe Stecker St5) verwendet. Er darf dann nicht mehr extern angeschlossen sein.

Alternative Funktionen:

Wenn kein Jumper auf JP4 aufgesteckt ist (= HOST disabled), entspricht der Pin dem Pin A10 von Stecker A des X-MAX-1 bzw. X-MAX-E und kann für die nachfolgenden Funktionen verwendet werden.

Alternativ-Funktion 1: Digitaler Ein-/Ausgang GP7

Alternativ-Funktion 2: Activity-/Wakeup-Eingang GP7

Alternativ-Funktion 3: SMI/NMI-Eingang GP7

Alternativ-Funktion 4: Interrupt-Eingang PIRQ1

Der Pin liegt zusätzlich am Interrupt Eingang PIRQ1, der per Software auf alle 15 Interrupt-Eingänge des Interrupt-Controllers gelegt werden kann.

/WDO (Pin 11): Watch-Dog Ausgang

Dieser Pin ist der Ausgang des Watch-Dogs des MAX-PC. Er ist active Low. Wenn der Watch-Dog nicht rechtzeitig per Software nachgetriggert wird, geht der Ausgang /WDO auf log. 0. Zum Nachtriggern gibt es in OsX eine System-Routine bzw. einen entsprechenden Makro-Befehl.

/MRES (Pin 12): Manual Reset

Hier kann ein Reset Taster gegen GND angeschlossen werden. Wenn der Eingang log. 0 ist, wird ein Reset des MAX-PC‘s ausgelöst. Der Eingang hat Schmitt-Trigger Charakteristik.

SPKR (Pin 13): Lautsprecher-Ausgang

Hier kann man, so wie als Beispiel im Schaltplan vom Evaluation-Board X-KiT-3 angegeben, einen Lautsprecher anschließen.

n.c. (Pin 14)

Dieser Pin ist nicht angeschlossen. Auf X-MAX-1 bzw. X-MAX-E wird er als BOOT-Pin verwendet, der bei X-KiT-3 an Jumper JP1 = „BOOT“ liegt. Wenn kein Jumper aufgesteckt ist, dann wird vom on-board Flash von X-MAX-1 bzw. X-MAX-E gebootet. Wenn er aufgesteckt ist, wird von PCMCIA Card A gebootet.

GP0 (Pin 15): allg. I/O-Pin

Standard-Funktion: digitaler Ein-/Ausgang GP0

Dieser Pin kann als universeller I/O-Pin eingesetzt werden. Nach Reset ist der interne Pull-Up Widerstand aktiv.

Alternativ-Funktion 1: Activity-/Wakeup-Eingang GP0

Alternativ-Funktion 2: SMI/NMI-Eingang GP0

/BL1 (Pin 16): Battery Low 1

Dieser Eingangs-Pin hat Schmitt-Trigger-Charakteristik. Er verfügt über einen Pull-Up Widerstand von 4,7 kOhm. Er kann so programmiert werden, dass eine positive oder negative Flanke die CPU in einen Stromspar-Modus schaltet. Der Status des Pins ist lesbar.

2.7.4.3. SIRIN und SIROUT (Pin 17 und 18): iRDA

Diese beiden Pins sind auch bei X-KiT-3 **nicht** 5-Volt-kompatibel, die maximale Eingangsspannung ist beschränkt auf 3,6 Volt. Ihre Funktion ist abhängig von Jumper „IRDA“ (= JP5). Wenn ein Jumper aufgesteckt ist (= iRDA enable), ist der on-board iRDA-Transceiver aktiv und die beiden Pins müssen **unbeschaltet** bleiben. Wenn kein Jumper aufgesteckt ist (iRDA disable), kann ein externer iRDA-Transceiver angeschlossen werden.

SIRIN ist der iRDA-Eingang, SIROUT der iRDA-Ausgang.

SIRIN verfügt über einen abschaltbaren Pull-Down Widerstand.

SIROUT ist im Tri-State Modus, wenn die iRDA-Schnittstelle per Register abgeschaltet ist. Er verfügt über einen Pull-Down Widerstand.

Ein Beispiel für den Anschluss eines iRDA-Transceivers finden Sie im Schaltplan des Evaluation-Board X-KiT-3 (Application Note AN087).

LEDint (Pin 19): LED Ausgang

Dieser Logik-Ausgang liefert den Zustand der on-board LED des MAX-PC.

IRQ-0 (Pin 20): Interrupt-Eingang PIRQ-0 bzw. Ein-/Ausgang GP8

Standard-Funktion: Universeller Ein-/Ausgang GP8

Nach Reset ist er als Eingang konfiguriert. Als Eingang kann er per Software mit einem Pull-Up Widerstand versehen werden.

Alternativ-Funktion 1: Wakeup/Activity-Eingang GP8

Als Eingang kann der Pin zusätzlich so konfiguriert werden, dass bei einer negativen Flanke das System aktiviert wird.

Alternativ-Funktion 2: SMI/NMI-Eingang GP8

Alternativ-Funktion 3: Interrupt-Eingang PIRQ0

P3 (Pin 21): 3,3 Volt Versorgung

Hier steht die 3,3 Volt Versorgungsspannung des Moduls X-MAX-1 bzw. X-MAX-E zur Verfügung.

2.7.4.4. BAT (Pin 22): Batterie-Eingang

Dieser Pin muss unbeschaltet bleiben, wenn der Jumper JP2 = „BAT“ aufgesteckt ist, auch wenn keine Batterie auf dem X-KiT-3 eingesetzt ist, weil der +Pol der Batteriefassung dann an GND liegt.

Wenn extern eine Batterie angeschlossen werden soll, darf der Jumper „BAT“ nicht bestückt sein.

Drucker-Port (Pin 23..39)

/STRB, /AFDT, /ERROR, /INIT, /SLCTIN, /ACK, BUSY, PE, SLCT, PD0..PD7

Diese Pins sind verfügbar auf den Steckern St7 (D-SUB) und St1. Zusätzlich werden sie verwendet für PCMCIA Slot B. Wenn eine PCMCIA-Karte in Slot B steckt, leuchtet eine rote LED (=LED2) neben dem Stecker St7 und die Pins dürfen nicht beschaltet werden. Nur wenn LED2 nicht leuchtet, können die Pins als Druckerport oder für die unten angegebenen Funktionen verwendet werden.

Standard-Funktion: Druckerport

Diese Pins zusammen können als unidirektionaler oder bidirektionaler Druckerport oder als Enhanced Parallel Port (EPP) konfiguriert werden.

Alternativ-Funktion 1: PCMCIA-Slot B

Wenn ein PCMCIA-Slot B im System vorhanden und aktiv ist, darf der Druckerport nicht benutzt werden.

Alternativ-Funktion 2: Digitale Ein-/Ausgänge PD0..PD7

Hierzu muss der Druckerport als bidirektionaler Port konfiguriert werden. Die Ein-/Ausgänge PD0..PD7 können nur alle 8 gemeinsam als Eingänge oder Ausgänge konfiguriert werden.

Alternativ-Funktion 3: Digitale Eingänge BUSY, /ACK, PE, SLCT, /ERROR

Der Status dieser Eingänge kann per Software über Register abgefragt werden: Bit 7 = BUSY invertiert, Bit 6 = /ACK, Bit 5 = PE, Bit 4 = SLCT, Bit 3 = /ERROR.

Alternativ-Funktion 4: Digitale Ausgänge /STRB, /AFDT, /INIT, /SLCTIN

Diese Ausgänge können einzeln über Register gesetzt werden: Bit 3 = /SLCTIN invertiert, Bit 2 = /INIT, Bit 1 = /AFDT invertiert, Bit 0 = /STRB invertiert.

Alternativ-Funktion 5: Interrupt-Eingang

Der Druckerport kann einen Interrupt auslösen, der auf IRQ7 gelegt ist.

Alternativ-Funktion 6: Spezialfunktionen

Durch werksseitige Umprogrammierung können die Pins PD0..PD7 auch andere Funktionen übernehmen, bitte anfragen.

GND (Pin 40)

Bezugspotential für das System bzw. den Druckerport.

2.7.4.5. Stecker St2

Viele Pins entsprechen in ihrer Funktion denen von Stecker C von X-MAX-1 bzw. X-MAX-E, bis auf die mit (*) gekennzeichneten Ausnahmen. Die mit (+) gekennzeichneten Pins sind 5-Volt-kompatibel, anders als bei X-MAX-1 oder X-MAX-E, wo die maximale Eingangs-Spannung bei manchen Pins auf 3,6 Volt begrenzt ist.

Die Pins 27 bis 40 stellen alle Signale für einen direkten Anschluss eines LC-Displays zur Verfügung.

Einige Signale für die PCMCIA-Slots können auch für andere Zwecke verwendet werden, wenn sie nicht als PCMCIA Schnittstellen-Signale verwendet werden. Wenn nur ein PCMCIA-Slot vorgesehen werden soll, muss das Slot A sein. Die Pins für Slot B sind trotzdem auch anders nutzbar.

Pin	Name	Pin	Name	Pin	Name	Pin	Name
1+	RSTDRV	6	XTCLK	11*	/MCEHA (TMS)	16*	VCON
2+	GP16	7+	/BL2	12*	RSTA (TDI)	17*	3,3 Volt
3+	GP17	8	XTDATA	13*	/REGA (TDO)	18*	VBR
4+	SUSRES	9+	JTAG	14+	GP18 (VPP2B)	19*	5 Volt
5+	ACIN	10*	/MCELA (TCK)	15*	GND	20*	DISPON

Pin	Name	Pin	Name	Pin	Name	Pin	Name
21*	12 Volt	26	GP20 (/CDA2)	31	LC	36	LCDD4
22*	CCFL	27	/LVDD	32	M	37	LCDD3
23+	GP14 (VPP1A)	28	/LVEE	33	LCDD7	38	LCDD2
24+	GP15 (VPP2A)	29	FRM	34	LCDD6	39	LCDD1
25+	GP13 (VCCA)	30	SCK	35	LCDD5	40	LCDD0

RSTDRV (Pin 1): Reset Ausgang

Dieser Pin liefert ein Reset Signal (active High), wenn das Modul X-MAX-1 bzw. X-MAX-E einen Reset durchführt.

GP16 (Pin 2): Digitaler Ein-/Ausgang

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion: Digitaler Ausgang

GP17 (Pin 3): Digitaler Ein-/Ausgang

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion 1: Digitaler Ausgang

Alternativ-Funktion 2: Wenn PCMCIA-Slot B vorgesehen ist (wie bei X-KiT-3), dient dieser Pin als Ausgang VPP1B.

SUSRES (Pin 4): Suspend Reset

Dieser Schmitt-Trigger-Eingang verfügt über einen Pull-Up Widerstand von 100 kOhm. Bei einer Flanke an diesem Eingang kann das System in den Power-Down Modus gesetzt werden.

ACIN (Pin 5): Ext. Versorgung vorhanden

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion: Dieser Pin dient dazu, dem System die Verfügbarkeit einer externen Spannungsversorgung anzuzeigen. Der Schmitt-Trigger-Eingang verfügt über einen Pull-Down Widerstand. Bei einer Flanke an diesem Eingang kann das System in den Power-Down Modus gesetzt werden.

2.7.4.6. XTCLK und XTDATA (Pin 6 und 8): XT-Keyboard

Diese beiden Pins liegen direkt an den gleichen Pins von Stecker C von X-MAX-1 bzw. X-MAX-E. Wenn eine XT-kompatible Tastatur an Stecker St9 angeschlossen ist, dann müssen diese beiden Pins an St2 unbeschaltet bleiben.

/BL2 (Pin 7): Battery Low 2

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion Dieser Eingangs-Pin hat Schmitt-Trigger-Charakteristik. Er verfügt über einen Pull-Up Widerstand von 100 kOhm. Er kann so programmiert

werden, dass eine positive oder negative Flanke die CPU in einen Stromspar-Modus schaltet.

2.7.4.7. JTAG, TCK, TMS, TDI, TDO (Pin 9..13)

Der Pin JTAG dient Testzwecken und muss unbeschaltet bleiben. Er liegt parallel an Pin 9 von Stecker C von X-MAX-1 bzw. X-MAX-E und am Jumper „JTAG“ (=JP7). Wenn hier ein Jumper aufgesteckt ist, werden die Pins /MCELA (Pin 10), MCEHA (Pin 11), RSTA (Pin 12) und /REGA (Pin 13) für PCMCIA-Slot A verwendet. Wenn kein Jumper aufgesteckt ist, dienen die 4 Pins als JTAG-Schnittstelle mit Pin 10 = TCK, Pin 11 = TMS, Pin 12 = TDI und Pin 13 = TDO für die CPU auf dem X-MAX-1 bzw. X-MAX-E.

GP18 (Pin 14): Digitaler Ein-/Ausgang

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion 1: Digitaler Ausgang

Alternativ-Funktion 2: VPP2B für PCMCIA-Slot B

Wenn PCMCIA-Slot B vorgesehen ist (wie bei X-KiT-3), dient dieser Pin als Ausgang VPP2B.

GND (Pin 15)

Bezugspotential für das X-KiT-3.

VCON (Pin16)

Ausgangsspannung des 12-Bit D/A-Wandlers zur Steuerung des Kontrasts des on-board LC-Displays.

3,3 Volt (Pin 17), 5 Volt (Pin 19) und 12 Volt (Pin 21)

Versorgungsspannungen des X-KiT-3.

VBR (Pin 18)

Ausgangsspannung des 12-Bit D/A-Wandlers zur Steuerung der Helligkeit (Hintergrundbeleuchtung) des on-board LC-Displays.

DISPON (Pin 20)

Digitaler Ausgang zur Steuerung der on/off-Funktion des on-board Displays.

CCFL (Pin 22)

Digitaler Ausgang zur Steuerung der on/off-Funktion der on-board Hintergrundbeleuchtung.

GP14 bzw. VPP1A (Pin 23)

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion 1: Digitaler Ausgang

Alternativ-Funktion 2: Digitaler Activity-/Wakeup-Eingang

Hierzu muss der Pin als Eingang konfiguriert werden

Alternativ-Funktion 3: SMI/NMI-Eingang

Hierzu muss der Pin als Eingang konfiguriert werden

Alternativ-Funktion 4: VPP1A für PCMCIA-Slot

Wenn PCMCIA-Slot A vorgesehen ist (wie bei X-KiT-3), dient dieser Pin als Ausgang VPP1A für PCMCIA-Slot A.

GP15 bzw. VPP2A (Pin 24)

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion 1: Digitaler Ausgang

Alternativ-Funktion 2: VPP2A für PCMCIA-Slot A

Wenn PCMCIA-Slot A genutzt wird, (wie bei X-KiT-3) dient dieser Pin als Ausgang VPP2A für PCMCIA-Slot A.

GP13 bzw. VCCA (Pin 25)

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion 1: Digitaler Ausgang

Alternativ-Funktion 2: Digitaler Activity-/Wakeup-Eingang

Hierzu muß der Pin als Eingang konfiguriert werden.

Alternativ-Funktion 3: VCCA für PCMCIA-Slot A

Wenn PCMCIA-Slot A vorgesehen ist (wie bei X-KiT-3), dient dieser Pin als Ausgang VCCA für PCMCIA-Slot A.

GP20 bzw. /CDA2 (Pin 26)

Standard-Funktion: Digitaler Eingang

Alternativ-Funktion 1: Digitaler Ausgang

Alternativ-Funktion 2: /CDA2 für PCMCIA-Slot A

Wenn PCMCIA-Slot A vorgesehen ist (wie bei X-KiT-3), kann dieser Pin als zweiter Card-Detect-Eingang /CDA2 für PCMCIA-Slot A dienen.

/LVDD, /LVEE, FRM, SCK, LC, M, LCDD0..LCDD7 (Pin 27..40)

Diese Pins (alles Ausgänge) dienen ausschließlich für den Anschluss eines LC-Displays.

2.7.4.8. Stecker St3: an Stecker A von Modul 2

Pin	Name	Pin	Name	Pin	Name	Pin	Name
1	A1	11	A11	21	A21	31	A31
2	A2	12	A12	22	A22	32	A32
3	A3	13	A13	23	A23	33	A33
4	A4	14	A14	24	A24	34	A34
5	A5	15	A15	25	A25	35	A35
6	A6	16	A16	26	A26	36	A36
7	A7	17	A17	27	A27	37	A37
8	A8	18	A18	28	A28	38	A38
9	A9	19	A19	29	A29	39	A39
10	A10	20	A20	30	A30	40	A40

2.7.4.9. Stecker St4: an Stecker A von Modul 3

Pin	Name	Pin	Name	Pin	Name	Pin	Name
1	A1	11	A11	21	A21	31	A31
2	A2	12	A12	22	A22	32	A32
3	A3	13	A13	23	A23	33	A33
4	A4	14	A14	24	A24	34	A34
5	A5	15	A15	25	A25	35	A35
6	A6	16	A16	26	A26	36	A36

Pin	Name	Pin	Name	Pin	Name	Pin	Name
7	A7	17	A17	27	A27	37	A37
8	A8	18	A18	28	A28	38	A38
9	A9	19	A19	29	A29	39	A39
10	A10	20	A20	30	A30	40	A40

2.7.4.10. Stecker St5: Serielle HOST-Schnittstelle (RS-232)

Diese Schnittstelle ist nur aktiv, wenn ein Jumper auf dem Jumperfeld „HOST“ aufgesteckt ist. Die Pinbelegung dieses 9-pol. D-Sub. Steckers entspricht denen einer seriellen PC-Schnittstelle.

Pin an St5	Name
1	n.c.
2	RCV (Eingang)
3	TMT (Ausgang)
4	n.c.
5	GND
6	n.c.
7	RTS (Ausgang)
8	CTS (Eingang)
9	n.c.

n.c. = not connected

2.7.4.11. Stecker St6: Serielle Schnittstelle COM2 für X-MAX-1

Dieser Stecker darf nicht angeschlossen werden, wenn das X-KiT-3 mit einem X-MAX-E bestückt ist. Die Pins dieses 9-pol. D-Sub. Steckers sind direkt mit den Pins von Stecker A von Modul 1 (bestückt mit X-MAX-1) verbunden.

Pin St6 (X-KiT-3)	**Pin St1 (X-KiT-3)**	**Pin an Stecker A von Modul 1 (=X-MAX-1)**	**Name**
1	1	A1	DCD
2	3	A3	RCV
3	5	A5	TMT
4	7	A7	DTR
5	9	A9	GND
6	2	A2	DSR
7	4	A4	RTS
8	6	A6	CTS
9	8	A8	Ri

2.7.4.12. Stecker St7: Druckerport LPT1 bzw. LPT2

Der Druckerport ist verfügbar an Stecker St7. Die Pinbelegung entspricht der üblichen 25-pol. D-Sub Buchse in PCs. Die gleichen Signale stehen an St1 zur Verfügung. Wie sie verwendet werden können, steht in der Beschreibung zum Stecker St1 in diesem Handbuch.

Pin St7 (X-KiT-3)	**Pin St1 (X-KiT-3)**	**Pin an Stecker A von Modul 1 (=X-MAX-1)**	**Name**
1	23	A23	/STRB
2	25	A25	PD0
3	27	A27	PD1
4	29	A29	PD2
5	31	A31	PD3
6	33	A33	PD4
7	35	A35	PD5
8	37	A37	PD6
9	39	A39	PD7
10	32	A32	/ACK
11	34	A34	BUSY
12	36	A36	PE
13	38	A38	SLCT
14	24	A24	/AFDT
15	26	A26	/ERROR
16	28	A28	/INIT
17	30	A30	/SLCTIN
18..25	40	A40	GND

2.7.4.13. Stecker St8: Anschluss für AT-Keyboard

Pin an St8	Name
1	GP20 (= Pin C26 von Modul 1)
2	-
3	GND
4	+5 Volt
5	GP0 (= Pin A15 von Modul 1)
6	-

2.7.4.14. Stecker St9: Anschluss für XT-Keyboard

Pin an St9	Name
1	XTCLK (= Pin C6 von Modul 1)
2	XTDATA (= Pin C8 von Modul 1)
3	/XTRES
4	GND
5	+5 Volt

2.7.4.15. Stecker St10: 9-pol. D-Sub. für serielle Schnittstelle für X-MAX-E

Dieser Stecker darf nicht angeschlossen werden, wenn Modul 1 = X-MAX-1.

Pin an St10 (X-KiT-3)	**X-MAX-E (Stecker A)**	**Signal RS-232**	**Signal RS-422**	**Signal RS-485**
1	-	n.c.	n.c.	n.c.
2	A6	RCV	RCV+	n.c.
3	A8	TMT	TMT-	DATA-
4	-	n.c.	n.c.	n.c.
5	-	GND	GND	GND
6	-	n.c.	n.c.	n.c.
7	A7	RTS	TMT+	DATA+
8	A9	CTS	RCV-	n.c.
9	-	-	n.c.	n.c.

n.c. = not connected

Die Umschaltung von RS-232, RS-422 und RS-485 erfolgt per Software über die Leitungen PDRQ0 und DTR des X-MAX-E gemäß nachfolgender Tabelle. Wenn RS-232 konfiguriert wird, kann RTS wie RTS verwendet werden. Wenn RS-422 oder RS-485 konfiguriert wird, enabled RTS den RS-422/RS-485 Treiber.

	RS-232	**RS-422**	**RS-485** Receive	**RS-485** Transmit
PDRQ0	0	1	0	0
DTR	0	1	1	1
RTS	=RTS	1	0	1

2.7.4.16. Stecker St11: RJ45 für Ethernet 10BaseT für X-MAX-E

Dieser Stecker darf nicht angeschlossen werden, wenn Modul 1 = X-MAX-1.

Pin von St11	Name
1	TD+
2	TD-
3	RD+
4	n.c.
5	n.c.
6	RD-
7	n.c.
8	n.c.

n.c. = not connected

2.7.4.17. Buchse St23 und Stecker St14: 5 Volt Versorgungsspannung

An Klinkenbuchse St23 wird das dem X-KiT-3 beiliegende 5-Volt-Netzteil angeschlossen. Alternativ kann an St14 der für Disketten- und Festplattenlaufwerke in PCs übliche Stecker zur Stromversorgung angeschlossen werden. Es kann direkt die Versorgung aus einem PC verwendet werden, die +5 Volt liefert. **Nur eine der beiden Möglichkeiten darf benutzt werden!**

Pin an St14	Name
1	n.c.
2	GND
3	GND
4	+5 Volt

n.c. = not connected

2.7.4.18. Stecker St15: TDK CXA-K0505-HI (Hintergrundbeleuchtung)

Pin an St15	Name
1	P5
2	GND
3	CCFL
4	VBR
5	-

2.7.4.19. Stecker St16: Hitachi INV569 (Hintergrundbeleuchtung)

Pin an St16	Name
1	P5
2	P5
3	VBR
4	CCFL
5	GND
6	GND

2.7.4.20. Stecker St17 (24-pol.), St12 und St13: LC-Display Hitachi

Pin St17	Pin St12	Pin St13	Name	Pin St17	Pin St12	Pin St13	Name
24	-	-	DFLM	12	9	25	DLCDD0
23	33	1	GND	11	2	32	DLCDD1
22	27	7	DLC	10	16	16	DLCDD2
21	32	2	GND	9	3	31	DLCDD3
20	23	11	DSCK	8	-	-	DLVEE (DISPON)
19	28	6	P3	7	29	5	P3
18	30	4	GND	6	-	-	VCON
17	4	30	DLCDD4	5	22	12	GND
16	-	-	DLCDD5	4	-	-	Y-
15	-	-	DLCDD6	3	-	-	X-
14	-	-	DLCDD7	2	-	-	Y+
13	26	8	GND	1	-	-	X+
-	31	3	P3	-	17, 18	16, 17	DLCDD2
-	1, 5-8, 12-15, 19-21	33, 12-15, 19-22, 26-29	GND	-	10, 11	23, 24	DLCDD0
-	24, 25	9, 10	DSCK				

2.7.4.21. Stecker St18 und St19: Anschluss für PC-CARD A und B

Siehe Lageplan von X-KiT-3.

2.7.4.22. Stecker St20: JTAG für CPLDs IC1 und IC2 des X-KiT-3

St20 ist ein 8-pol. einreihiger Pfostenstecker mit Rastermaß 2,54 mm. Pin 5 fehlt und dient der Codierung.

Pin an St20	Name
1	TDI
2	TCK
3	TMS
4	TDO
5	Kein Pin
6	n.c.
7	GND
8	+5 Volt

n.c. = not connected

2.7.4.23. Stecker St21: JTAG für Modul 1..3 und ggfls. CPU von X-MAX-1 bzw. X-MAX-E

St21 ist ein 8-pol. einreihiger Pfostenstecker mit Rastermaß 2,54 mm. Pin 5 fehlt und dient der Codierung. Wenn Jumper „EXT.JTAG“ aufgesteckt ist, wird auch die CPU in die Kette einbezogen, andernfalls nur die Module 1, 2 und 3.

Pin an St21	Name
1	TDI
2	TCK
3	TMS
4	TDO
5	Kein Pin
6	n.c.
7	GND
8	+5 Volt

n.c. = not connected

2.7.4.24. Stecker St22: Lötpads für Anschluss eines Inverters für Hintergrundbeleuchtung

Pin an St22	Name
1	+5 Volt
2	+5 Volt
3	VBR (DIM)
4	CCFL (ON/OFF)
5	GND
6	GND
7	+12 Volt
8	+12 Volt

2.7.4.25. Beschreibung der Pins von Stecker B (X-Bus Anschluss)

Diese Pins liefern alle Signale für den X-Bus und die Versorgungsspannungen der Module. Die Application Note AN084 kann als Beispiel für den Anschluss der Module untereinander dienen. Folgende Pins erfordern besondere Beachtung:

GND (Pins B1, B17, B39)

Diese Pins dienen als Bezugspotential für die Versorgungsspannungen von 3,3 Volt und die optionalen +/- 12 Volt.

P3 (Pins B5, B9, B25)

Hier wird die Versorgungsspannung von +3,3 Volt angeschlossen.

M12 (Pin 31)

Hier wird die Versorgungsspannung von -12 Volt angeschlossen. Auf dem Modul X-MAX-1 ist dieser Pin nicht angeschlossen.

P12 (Pin 35)

Hier wird die Versorgungsspannung von +12 Volt angeschlossen. Auf dem Modul X-MAX-1 ist dieser Pin nicht angeschlossen.

/XREQ, XCLK, /XAS, /XRDY, /XRES (Pins B2..B4, B6, B40)

Dies sind die X-Bus Steuersignale.

XC0..XC15 (Pins B8, B7, B10..B15, B18..B20, B23, B22, B27, B24, B28)

Dies sind die X-Bus Command/Address/Datenleitungen.

XLN0..XLN3 (Pins B16, B21, B26, B32)

Über diese Pins wird die 4-Bit Slot-Nummer eingeprägt.

XDL0..XDL2 (Pins B34, B36, B38)

Diese Pins verbinden die Module untereinander. Ihre Funktion wird durch die Module bestimmt.

TCK, TMS, TDI, TDO (Pins B29, B30, B37, B33)

Diese Pins bilden die JTAG-Schnittstelle des Funktionsteils des Moduls, Beschaltung siehe Kapitel 2.7.4.5. Wenn die JTAG-Schnittstelle nicht benutzt wird, können die Pins unbeschaltet bleiben.

2.7.5. Remote-Verbindungen

2.7.5.1. Serielle Remote-Verbindung

Führen Sie die nachfolgenden Schritte aus, um eine serielle Remote-Verbindung aufzubauen:

- Installieren Sie auf dem PC zunächst den Windows-Treiber und das Programm SNW32.
- Starten Sie auf dem PC das Symbol *SORCUS boards* in der Windows-Systemsteuerung.
- Jetzt können Sie in der Kartenliste mit der rechten Maustaste auf eine freie Karte klicken und aus dem Kontextmenü *New remote connection* auswählen. Als *Type* wählen Sie *Serial connection* und als *Board* wählen Sie *MAX-PC*.
- Die nachfolgenden Dialoge ermöglichen Ihnen dann, eine Verbindung einzurichten. Wählen Sie hier die entsprechende serielle Schnittelle des PC. Die Schnittstelle muss folgendermaßen konfiguriert werden: Speed=38400; Parity=none; Stop bits=1.
- Nach diesem Vorgang erscheint in der Liste der Eintrag *MAX-PC*.
- Verbinden Sie die serielle Schnittstelle des X-MAX-1 bzw. X-MAX-E über eine Nullmodem-Verbindung (RCV und TMT Leitungen kreuzen, GND Leitungen verbinden), mit der seriellen Schnittstelle des PCs:
  Stecken Sie das X-MAX-1 Modul auf Steckplatz 1 und verbinden Sie den 9-poligen D-Sub-Stecker ST6 des X-KiT-3 mit einem Nullmodem-Kabel mit der seriellen PC-Schnittstelle.
  Verwenden Sie ein X-MAX-E Modul, dann nehmen Sie bitte das Kabel KX-3795.
- Schalten Sie nun das X-KiT-3 ein. Das X-KiT-3 ist nun betriebsbereit.
- Starten Sie nun SNW32 und wählen Sie das X-KiT-3 aus. Die X-Bus Module auf dem X-KiT-3 erscheinen in einer Baumstruktur.

2.7.5.2. Ethernet Remote-Verbindung

Führen Sie die nachfolgenden Schritte aus, um eine Ethernet Remote-Verbindung aufzubauen:

- Installieren Sie auf dem PC zunächst den Windows-Treiber und das Programm SNW32.
- Starten Sie auf dem PC das Symbol *SORCUS boards* in der Windows-Systemsteuerung.
- Jetzt können Sie in der Kartenliste mit der rechten Maustaste auf eine freie Karte klicken und aus dem Kontextmenü *New remote connection* auswählen. Als *Type* wählen Sie *TCP/IP connection to MAX-PC*.
- Im nachfoldenden Dialog müssen Sie die IP-Adresse für die Verbindung eintragen. Wenn Sie das X-KiT-3 innhalb eines Netzwerkes einsetzen, erfragen Sie bitte eine freie IP-Adresse bei Ihrem Netzwerkadministrator.
- Nach diesem Vorgang erscheint in der Liste der Eintrag *MAX-PC*.
- Um eine Remote-Verbindung des X-KiT-3 per Ethernet aufzubauen, müssen auf dem „CPU-Modul“ (X-MAX-1 bzw. X-MAX-E) zunächst entsprechende Treiber installiert werden und auf dem „Ethernet-Modul“ (X-ETH-10 bzw. X-MAX-E) muss die IP-Adresse eingestellt werden. Für diesen Konfigurationsvorgang muss einmalig eine serielle Remote-Verbindung aufgebaut werden oder das „CPU-Modul“ und das „Ethernet-Modul“ müssen dafür auf eine MAX6pci Karte aufgesteckt werden:
- Serielle Remote-Verbindung: Wenn Sie das X-ETH-10 Modul als „Ethernet-Modul“ verwenden, dann stecken Sie das X-ETH-10 Modul auf Steckplatz 2 auf (zusätzlich zum „CPU-Modul“ auf Steckplatz 1).
- MAX6pci: Stecken Sie das „CPU-Modul“ und das „Ethernet-Modul“auf beliebige freie Steckplätze.
- Starten Sie SNW32 und wählen Sie dort das seriell angekoppelte X-KiT-3 bzw. die MAX6pci-Karte aus.
- Starten Sie den SNW32-Assistenten für das „Ethernet-Modul“ (X-ETH-10 bzw. X-MAX-E) (durch einen Doppelklick auf das Modul) und wählen Sie die Seite *TCP/IP Configuration* aus. Tragen Sie hier die IP-Adresse ein, die Sie zuvor in *SORCUS boards* in der Systemsteuerung eingetragen haben.
- Starten Sie nun den SNW32-Assistenten für das „CPU-Modul“ (X-MAX-1 bzw. X-MAX-E) und wählen Sie die Seite *System driver* aus. Auf dieser Seite werden alle installierten System-Treiber angezeigt. Fügen Sie den Treiber *Ethernet Host Connection Driver* für die Ethernet Remote-Verbindung hinzu. Sie können bei der Konfiguration des Treibers das „Ethernet-Modul“ auswählen, über das die Ankopplung erfolgen soll, sowie dessen spätere (d.h., wenn es in der Ethernet Remote-Verbindung im Einsatz ist) Slot- und Layer-Nummer festlegen:

- Verwenden Sie das X-ETH-10 als „Ethernet-Modul“, dann sollte hier Slot=2 und Layer=0 eingetragen werden.
- Verwenden Sie das X-MAX-E als „Ethernet-Modul“ und „CPU-Modul“, dann sollte hier Slot=1 und Layer=0 eingetragen werden.
- Über den Punkt *Install into Flash* müssen Sie jetzt den Treiber in das Flash des „CPU-Moduls“ laden. Damit steht er nach jedem Reset automatisch zur Verfügung. Beenden Sie anschließend SNW32.
- Stecken Sie die Module auf das X-KiT-3 und stellen Sie eine Verbindung mit dem Rechner her:
  Verwenden Sie das X-MAX-E als „Ethernet-Modul“ und „CPU-Modul“, dann stecken Sie dieses auf Steckplatz 1. Verbinden Sie den 9-poligen D-Sub-Stecker ST6 des X-KiT-3 mit dem Kabel KX-3795 direkt mit dem PC bzw. mit einem Crossover-Kabel mit dem Netzwerk.
  Verwenden Sie das X-ETH-10 Modul als „Ethernet-Modul“, dann stecken Sie dieses auf Steckplatz 2 und das „CPU-Modul“ auf Steckplatz 1. Verbinden Sie die entsprechenden PINs des Pfosten-Steckers ST3 mit dem PC bzw. mit dem Netzwerk.
- Schalten Sie nun das X-KiT-3 ein. Das X-KiT-3 ist nun betriebsbereit.
- Starten Sie nun SNW32 und wählen Sie das X-KiT-3 aus. Die X-Bus Module auf dem X-KiT-3 erscheinen in einer Baumstruktur.

2.7.6. Technische Daten

Maße: 230 x 182 mm

Betriebsspannung: 5V (4,75V...5,5V)

Notizen

3. Das Karten-Manager-Programm SNW32

3.1. Aufgabe

Das Programm „Schöne Neue Welt 32“ (SNW32) dient zum Steuern, Konfigurieren und Testen von SORCUS Karten und Modulen vom Host-PC aus. Es unterstützt alle MAX-Trägerkarten und MAX-Module. Das Programm arbeitet unter allen WIN32 und Linux Betriebssystemen.

SNW32 zeigt die einzelnen Komponenten des SORCUS Systems in einer übersichtlichen Baumstruktur an. Über diese erhält der Anwender Zugriff auf sämtliche zur Verfügung stehenden Funktionseinheiten der jeweils installierten Karte(n). Neben dem Zugriff auf die Hardwarekomponenten bietet das Programm auch die Möglichkeit zur Installation und Verwaltung von Softwarekomponenten auf der Karte. Ein Taskmanager kann beispielsweise alle auf der Karte installierten Echtzeitprogramme sowie deren Parameter anzeigen. Es ist auch möglich, ohne SNW32 starten zu müssen, über Kommandozeilenparameter mit Hilfe von Installationsdateien Echtzeitprogramme zu installieren.

3.2. Installation

Das Programm wird über ein Installationsprogramm auf Ihrem Rechner installiert. Beachten Sie, dass Sie auch einen entsprechenden Treiber für Ihre SORCUS Karte installiert haben (siehe Kapitel Treiberinstallation 2.1.3.)!

Das Installationsprogramm von SNW32 befindet sich im Download-Bereich unserer SORCUS-CD bzw. Homepage. Dort befindet sich auch eine Text-Datei mit weiteren Installationshinweisen.

3.3. Assistenten

Je Funktionseinheit steht dem Benutzer ein Assistent zur Verfügung, der einen komfortablen Zugriff auf das angewählte Device ermöglicht, und zwar sowohl für die Komponenten der MAX-Trägerkarte wie auch für die aufgesteckten MAX-Module. Damit ist es z.B. möglich, Ein- oder Ausgaben durchzuführen. Die Assistenten liegen in Form von 32-Bit DLLs vor und können auch problemlos in andere Windows-Applikationen eingebunden werden.

3.4. Installations-Dateien

Die SORCUS Karten bzw. der MAX-PC verfügen über ein eigenes Echtzeit Multi-Tasking-Betriebssystem OsX. Eine zentrale Aufgabe im Umgang mit den Karten ist es, Echtzeit-Programme als Tasks zu installieren. Die einfachste Methode, Programme zu installieren, Parameter zu setzen und Prozeduren zu starten, ist die Verwendung von Installationsdateien. Installationsdateien sind Textdateien und bestehen aus einer Folge von Anweisungen, die die oben genannten Aktionen ausführen. Dafür sind einige Schlüsselwörter definiert, die in den Hilfetexten und im Anhang G ausführlich erklärt sind. Installationsdateien erhalten standardmäßig die Extension '.INS'. Das Programm enthält einen komfortablen Editor, mit dem solche INS-Dateien erstellt und getestet werden können. Um INS-Dateien auszuführen, kann SNW32 auch mit Komandozeilenparametern aufgerufen werden.

Die .INS-Dateien können auch im on-board Flash eines MAX-Moduls gespeichert werden. Nach einem Reset des Systems werden dieses dann automatisch ausgeführt.

3.5. Flash Unterstützung

Das auf den SORCUS Karten bzw. MAX-PCs vorhandene Flash-ROM kann neben dem Betriebssystem OsX auch Anwenderprogramme enthalten. Diese können nach einem Power-On oder nach einem Reset der Karte automatisch installiert werden. Der Flash-Assistent von SNW32 übernimmt dabei die gesamte Programmierung des Flash, so dass der Anwender keinen eigenen Programmieraufwand mehr hat.

3.6. Hotline File

Um der SORCUS-Hotline die Suche nach möglichen Fehlern in Ihrem System zu erleichtern, stellt das Programm SNW32 die Möglichkeit zur Verfügung, eine sog. Hotline-Datei zu erstellen. Diese Text-Datei enthält verschiedene Informationen, die die SORCUS Karten betreffen.

3.7. Hilfe

Das Programm bietet eine Online-Hilfe, die die Bedienung der einzelnen Komponenten näher erläutert.

4. Programmierung eines X-Bus-Systems vom Host-PC unter Windows

Die Einbeziehung von X-Bus-Karten in Windows-Programme geschieht über mitgelieferte Bibliotheken. Hinweise zur Einbindung der Bibliotheken in Programme unter den verschiedenen Compilern sind in Kap. 6.1 aufgeführt.

Grundsätzlich muss zwischen folgenden X-Bus-Systemen unterschieden werden:

- MAX6pci ohne aufgestecktes CPU-Modul
- System mit aufgestecktem CPU-Modul mit OsX-Betriebssystem.

4.1. Programmierung einer MAX6pci ohne CPU-Modul

Wenn Zugriffe auf I/O-Module (z.B. das Auslesen von Messwerten) nicht unbedingt periodisch erfolgen müssen und keine kurzen Reaktionszeiten erforderlich sind, kann eine MAX6pci Karte ohne CPU-Module betrieben werden. Die gesamte Funktionalität kann dann von einem Windows-Programm erledigt werden. Dabei ist zu beachten, dass der Windows Scheduler (je nach Systemauslastung) den einzelnen Prozessen mehr oder weniger Rechenleistung zukommen lässt.

I/O-Module werden über sog. Modul-Device-Treiber angesprochen (s. Kap. 8). Für jedes Modul steht ein solcher Windows-Kernel-Mode-Treiber zur Verfügung. Bei den Plug and Play Betriebssystemen werden die aufgesteckten Module automatisch erkannt und der zugehörige Treiber installiert. Unter Windows NT muss die Installierung mit dem Setup-Programm (Kap. 2) erfolgen.

Ist kein CPU-Modul aufgesteckt, stehen nur die Bibliotheksfunktionen (s. Kap. 6) für die Initialisierung sowie die Modul-Device-Treiber- und EEPROM-Zugriffe zur Verfügung.

4.2. Programmierung eines intelligenten X-Bus-Systems

Für Anwendungsaufgaben, bei denen ein konstantes Zeitraster zwischen den Zugriffen auf die I/O-Module unerlässlich ist (z.B. äquidistante Erfassung von Messwerten) oder bei denen sehr schnell auf Ereignisse wie z.B. das Setzen eines digitalen Eingangs reagiert werden muss, bietet Windows nur ungenügende Echtzeitfähigkeiten.

In diesem Fall bietet das Aufstecken eines CPU-Moduls auf eine MAX6pci eine einfache Lösung. Die Programmieraufgabe wird dann auf die verschiedenen CPUs aufgeteilt. Schnelle Echtzeit-Aufgaben wie z.B. Messdatenerfassung oder Regelungen sollten auf das CPU-Modul mit OsX (Kap. 5) ausgelagert werden. Der Windows-PC erledigt die weniger zeitkritischen Aufgaben wie z.B. Visualisierung oder Archivierung der Messdaten, Parametrierung der Regler usw. Das Windows-Programm kann dabei auf alle unter OsX auf dem CPU-Modul laufenden Echtzeitprogramme und Parameter zugreifen.

Bei der seriellen oder Ethernet-Ankopplung eines X-Bus-Systems wie z.B. dem MAX5dip und dem MAX8dip befindet sich auf dem Ziel-System immer ein CPU-Modul, das die Kommunikation mit dem Host steuert. Auf diesem CPU-Modul wird normalerweise auch der eigentliche Programm-Algorithmus ablaufen. Für Anwendungen, bei denen sehr viel mit dem Host-PC kommuniziert wird, kann auch ein zweites CPU-Modul auf das MAX5dip und MAX8dip gesteckt werden, das die Messaufgabe bearbeitet, während das andere CPU-Modul ausschließlich mit der Kommunikation zum Host beschäftigt ist.

Damit der PC mit dem CPU-Modul optimal zusammen arbeitet, sollten im Windows-Programm folgende Punkte beachtet werden:

- Datenblöcke sollten in möglichst großen Einheiten übertragen werden, da für jede einzelne Bibliotheksfunktion auf der Windows-Seite ein recht aufwändiger Treiber-Aufruf nötig ist. Alle vom Host zum CPU-Modul verschickten Befehle enthalten Overhead, so dass große Blöcke auch auf der CPU-Seite eine bessere Performance bewirken. Auf der Karte gesammelte Daten sollten also im CPU-Modul gepuffert werden, um sie in großen Blöcken zum Host zu übertragen. Blockgrößen von einigen Kilobyte sind sinnvoll.

- Alle zeitkritischen Vorgänge sollten in Echtzeitprogrammen auf der Karte behandelt werden. Periodische Abtastungen von Signalen sollten nur in Ausnahmefällen vom PC aus gesteuert werden. Ist dies trotzdem erforderlich, sollte der Zugriff über den auf dem PC installierten Modul-Device-Treiber für das jeweilige I/O-Modul erfolgen und nicht über den Modul-Device-Treiber auf dem CPU-Modul, weil dann zusätzlich das CPU-Modul belastet wird.

- Man sollte sich nie auf Reaktionszeiten des PCs auf irgendwelche Ereignisse (z.B. Service Requests der Karte) verlassen, da diese sehr stark von der Gesamtauslastung des PC-Systems abhängen.

- Um die Bedienbarkeit des PC-Gesamtsystems zu gewährleisten, sollte immer ereignisorientiert programmiert werden. In DOS-Zeiten übliche Endlosschleifen zur Abfrage des Zustands eines Echtzeitprogramms sollten vermieden werden.

4.3. PC-Programmbeispiel

Der folgende Auszug aus einem C-Programm zeigt, wie eine X-Bus-Karte initialisiert wird.

Die Funktion *InitXBusBoard* ruft alle Bibliotheksfunktionen auf, um die Bibliothek und die ausgewählte X-Bus-Karte zu initialisieren. Der Parameter *usBoardNr* muss den Wert einer installierten Karte enthalten. Eine Übersicht über die auf dem Host-PC konfigurierten X-Bus-Karten ist nach dem Ausführen des Setup-Programms in der Systemsteuerung unter SORCUS Boards (2.1.5.) zu finden.

Der Aufruf von *max_init_lib* dient zunächst der Bibliotheksinitialisierung. Durch *max_reset_board* wird anschließend ein Reset der Karte ausgeführt.

Fast alle Bibliotheksfunktionen erfordern die Übergabe eines Handles, das spezifiziert, an welche CPU der Bibliotheksbefehl gerichtet ist. Ein solches Handle erhält man durch den Aufruf von *max_connect_cpu*. Beim Aufruf wird durch die Parameter *usBoardNr*, *usCPUSlot* und *usCPULayer* angegeben, für welche CPU ein Handle angefordert wird.

Die Funktion *ErrorHandler* demonstriert die Verwendung der Bibliotheksfunktionen für die Fehlerbehandlung.

Wenn ein Programm keine Zugriffe auf die Bibliothek mehr macht, muss es *max_exit_lib* aufrufen.

```
#include "max_lib.h"

/*****************************************************************************
/
// Initialisierung des durch usBoardNr (s. Systemsteuerung „SORCUS Boards“)
// gegebenen X-Bus-Systems

int InitXBusBoard (USHORT usBoardNr)
{
   USHORT    usCPUSlot, usCPULayer;
   MAX_ERROR Error;
   MAXMODHND hModul;
   MAX_OS_INFO rcOsInfo;

   // Start des Bibliothekszugriffs
   Error = max_init_lib (0);
   // Konnte die Bibliothek initialisiert werden ?
   if (Error != ERR_OK)
   {
      ErrorHandler(Error, "max_init_lib");
      return 1;
   }

   // X-Bus-Karte zuruecksetzen
   Error = max_reset_board (usBoardNr, 20);
   if (Error)
   {
      ErrorHandler(Error, "max_reset_board");
      return 1;
   }

   // CPU-Handle holen
   Error = max_connect_cpu (usBoardNr, usCPUSlot, usCPULayer,
                            &rcOsInfo, &hModul);
   if (Error)
   {
      ErrorHandler(Error, "max_connect_cpu");
      return 1;
   }
   return 0;
}

/*****************************************************************************
/
// Fehler-Anzeige Subroutine

void ErrorHandler (MAX_ERROR Error, char *sText)
{
   USHORT  usMsgSize = 100;
   char    sErrMsg[200];

   max_get_error_message(Error, &usMsgSize, sErrMsg);
   printf ("\n\rError %X in %s (%s)", Error, sText, sErrMsg);
   max_clear_error();
}
```

5. Das Multi-Tasking Betriebssystem "OsX"

Das Echtzeit-Multi-Tasking Betriebssystem OsX des MAX-PC wurde bereits 1986 von SORCUS Computer für Z80 Systeme und 1990 für Intel i486 Prozessoren entwickelt und hat bis heute in Tausenden von Anwendungen seine hohe Effizienz und Benutzerfreundlichkeit unter Beweis gestellt.

Jeder MAX-PC enthält ein eigenes Betriebssystem im Flash-ROM, das alle Schnittstellen und Funktionseinheiten des MAX-PC unterstützt. Es wird vom Host-PC aus über Makrobefehle und von Tasks auf dem MAX-PC über System-Subroutinen angesprochen. Die Makrobefehle bzw. System-Subroutinen liegen den Bibliotheken zugrunde, mit denen Sie das Betriebssystem direkt in einer Hochsprache benutzen können. Als Sprachen für die Echtzeitprogrammierung stehen Borland-Pascal, Borland C++ und Assembler zur Verfügung. Als Entwicklungsumgebung befindet sich die SORCUS-Entwicklungsumgebung RTDS auf der CD, mit der Echtzeitprogramme erstellt, kompiliert, auf dem MAX-PC geladen und debuggt werden können (s. Kap. 9). Alternativ können Programme auch mit Borland Pascal oder C++ Entwicklungsumgebungen erstellt werden. Von dort können sie jedoch nicht debuggt werden.

5.1. Die Echtzeitprogramme

Echtzeitprogramme sind Programme, die auf einem MAX-PC unter dem Echtzeitbetriebssystem OsX laufen. Dieses ist ein reaktionsschnelles Multi-Tasking Betriebssystem, auf dessen Eigenschaften im folgenden kurz eingegangen wird.

Das Betriebssystem OsX erlaubt dem Anwender, bis zu 1024 unabhängige Programme gleichzeitig auf einem MAX-PC laufen zu lassen. Die Programme können Interrupt-gesteuert (II-Tasks), Timer-gesteuert (TI-Tasks) oder Nicht-Interrupt-gesteuert (NI-Tasks) sein.

Damit ein Programm auf einem MAX-PC laufen kann, muss es zunächst in das RAM des MAX-PC geladen werden. Dieses kann auf drei Arten geschehen:

- Vom PC aus über eine entsprechende Funktion der Bibliothek (s. Kapitel 6.13)
- Vom PC aus über eine Installationsdatei (INS-FILE), die mit Hilfe des Karten-Manager-Programm SNW32 ausgeführt wird (s. Kapitel 3 sowie Anhang G)
- Das Programm kann im Flash-ROM des MAX-PC stehen. Nach einem Reset oder Power-On kann es dann automatisch aktiviert werden. Während der Installation werden die Programme ins RAM kopiert. Das Laden eines Programms in das Flash-ROM eines MAX-PC wird ebenfalls von einem Assistenten in SNW32 unterstützt.

Sobald das Programm im RAM des Moduls vorhanden ist, kann es als Task installiert werden. In der Regel geschieht dies im selben Schritt wie das Laden des Programms (Ausnahme: Mehrfachinstallation eines Programms, s. Seite 5-13). Unter dem **Installieren** wird die Bekanntgabe der erforderlichen Programm-Informationen an das Betriebssystem OsX verstanden. Das OsX legt daraufhin intern die erforderlichen Datenstrukturen für die Verwaltung des Programms als Task an.

Ein Echtzeitprogramm besteht aus folgenden Elementen:

- **Globale Prozeduren und Funktionen**, die vom Betriebssystem, von anderen Tasks, von anderen CPU-Modulen oder vom Host-PC aus aufgerufen werden können. Sie gehören zu einem Programm und stellen anderen Programmen bzw. Tasks bestimmte Funktionen zur Verfügung. Neben beliebigen Anwender-Prozeduren/Funktionen müssen zwei Prozeduren in jedem Echtzeitprogramm vorhanden sein: Hauptprozedur und Auto-Init-Prozedur.
  - **Hauptprozedur:** Sie wird nach dem Aktivieren einer Task vom Betriebssystem OsX aufgerufen, je nach Task-Typ Interrupt-, Timer- oder Nicht-Interrupt-gesteuert.
  - **Auto-Init-Prozedur:** Sie kann beim Installieren der Task einmal automatisch ausgeführt werden, um z.B. globale Variablen oder Parameter zu initialisieren. (gesteuert vom Flag MAX_CALL_AUTO_INIT beim Befehl **max_transfer_and_install**). Die Prozedur kann jederzeit auch manuell über den Befehl **max_call_proc** aufgerufen werden.
  - **beliebig viele anwenderdefinierte Prozeduren und Funktionen** (zu deren Besonderheiten s. Kapitel 6.16).
- **Parameterbereich**: Ein durchgängiger Speicherbereich, der bei der Installation einer Task angelegt und der Task zugeordnet wird. Seine Größe wird durch das

Echtzeitprogramm selbst festgelegt (max. 64 kB – 256). Er dient üblicherweise zur Aufnahme von Konfigurationsdaten (s. Kapitel 6.15).

- **Datenbereich**: Ein durchgängiger Speicherbereich, dessen Größe beim Installieren der Task angegeben werden kann. Er kann maximal den gesamten freien Speicher belegen. Üblicherweise werden darin z.B. Messwerte zwischengespeichert (s. Kapitel 6.15).
- Neben diesen drei Elementen sind, wie in Programmen für andere Betriebssysteme, die üblichen Programmelemente wie globale Variable, Prozeduren und Funktionen, die nur innerhalb des Programms benutzt werden, ohne Einschränkungen erlaubt. Auch objektorientierte Programmierung ist möglich.

Die Bibliotheken stellen Funktionen bereit, um globale Task-Prozeduren und Funktionen aufzurufen. Sie erlauben auch einen Zugriff auf den Parameter- und Datenbereich der einzelnen Tasks.

5.2. Die Task-Typen

Wann eine Task an der Reihe ist, also deren Haupt-Prozedur aufgerufen wird, hängt u.a. vom Task-Typ ab. Dieser wird durch das Programm selbst vorgegeben oder bei der Installation der Task festgelegt. Es gibt drei Task-Typen:

- Interrupt-Tasks (II-Tasks)
- Timer-Tasks (TI-Tasks)
- Nicht-Interrupt-Tasks (NI-Tasks)

5.2.1. Interrupt-Tasks (II-Tasks)

Sie haben die höchste Priorität. Wenn das entsprechende Interrupt-Ereignis auftritt, wird die Haupt-Prozedur des unter der Interrupt-Task installierten Programms einmal aufgerufen. Das Programm, also die Haupt-Prozedur, bestimmt selbst, wann es die Kontrolle an das Betriebssystem zurückgibt.

Die Priorität der II-Tasks untereinander wird durch die Hardware des MAX-PC weitgehend festgelegt.

Bei der Installation einer Interrupt-Task kann die Interrupt-Nummer mit angegeben werden, wenn sie nicht durch das Programm fest vorgegeben ist. Sie entspricht nicht der Interrupt-Vektor-Nummer. Im folgenden wird einfach von Interrupt xxx

gesprochen, wenn die Interrupt-Nummer gemeint ist. In Kapitel 10.24.4. finden Sie eine Zusammenstellung aller lokalen Interrupts auf dem MAX-PC.

5.2.2. Timer-Tasks (TI-Tasks)

Die Hauptprozedur einer TI-Task wird vom Betriebssystem in festen einstellbaren Zeitintervallen aufgerufen. Nach einer vom Benutzer angegebenen Anzahl von Aufrufen wird die Task automatisch deaktiviert, sofern die Anzahl der Aufrufe nicht auf "unendlich" gestellt wurde. Der erste Aufruf einer TI-Task kann sofort nach ihrer Aktivierung erfolgen oder erst nach einer einstellbaren Verzögerungszeit. Alle Zeitangaben, die zur Steuerung einer TI-Task benötigt werden, werden dem Betriebssystem beim Aktivieren als "Zeitplan" übergeben. Falls mehrere Aktivierungs-Befehle gegeben werden, während die zu aktivierende Task noch nicht aktiv ist, wird nur die letzte Aktivierung ausgeführt. Falls eine Task bereits aktiv ist und weitere Aktivierungs-Befehle gesendet werden, dann wird der letzte Aktivierungs-Befehl zwischengespeichert und nach Beendigung der Task ausgeführt.

Alle TI-Tasks werden von dem sogenannten TI-Task-Scheduler aufgerufen, der mit TIMER-C arbeitet und in festen Zeitintervallen (standardmäßig 1 ms) angesprochen wird. Dieses Zeitintervall wird als "Timer-Tic" bezeichnet, auf den sich alle TI-Tasks beziehen. Der Zeitplan der TI-Tasks, der beim Aktivieren übergeben wird, wird in Vielfachen des Timer-Tics angegeben. Um den Task-Scheduler müssen Sie sich im übrigen nicht weiter kümmern. Er wird automatisch eingerichtet, sobald die erste TI-Task installiert wird. Wenn Sie den Standard Timer-Tic von 1 ms verändern wollen, müssen Sie das vor Installation der ersten TI-Task tun (durch Änderung von Betriebssystemparameter 316).

Die TI-Tasks haben untereinander eine einstellbare Priorität, die bestimmt, welche Task gestartet wird, falls der Aufruf zweier TI-Tasks zeitlich zusammenfällt. Es ist auch möglich, einer TI-Task einmalig die höchste Priorität zuzuordnen, so dass sie auf jeden Fall als nächste TI-Task an die Reihe kommt, danach aber regulär weiterbearbeitet wird. Wenn eine TI-Task nicht zum vorgesehenen Zeitpunkt aufgerufen werden konnte, weil noch eine andere TI-Task lief oder eine TI-Task mit höherer Priorität zum selben Zeitpunkt gestartet werden sollte, wird der Aufruf später nachgeholt. Auch wenn mehrere Aufrufe hintereinander nicht bearbeitet werden konnten, geht keiner davon verloren. Sie werden alle nachgeholt, sobald die Tasks mit höherer Priorität ihre Aufrufe beendet haben. Wenn der Aufruf einer TI-Task aber über einen einstellbaren Zeitraum (Betriebssystemparameter 324) hinaus verzögert wird, löst das Betriebssystem einen Error-Request (Interrupt mit Übergabe eines Fehlercodes) zum Host-PC aus.

Bitte beachten Sie, dass sich TI-Tasks gegenseitig nicht unterbrechen. Eine laufende TI-Task wird immer erst beendet, bevor eine andere an die Reihe kommt, unabhängig von der Priorität. **!**

TI-Tasks bestimmen ebenso wie II-Tasks selbst, wann sie die Kontrolle an das Betriebssystem zurückgeben. TI-Tasks haben eine niedrigere Priorität als II-Tasks und können von diesen unterbrochen werden.

5.2.3. Nicht-Interrupt Tasks (NI-Tasks)

Sie haben die niedrigste Priorität. Sie können von II- und TI-Tasks unterbrochen werden. NI-Tasks werden in einer vom Anwender festgelegten Reihenfolge aufgerufen. Die Häufigkeit, mit der eine NI-Task aufgerufen wird, kann dadurch erhöht werden, dass sie mehrfach aktiviert wird. Dadurch ändert sich aber nur ihre Aufrufrequenz relativ zu anderen NI-Tasks. Auch NI-Tasks bestimmen selbst, wann sie die Kontrolle an das Betriebssystem zurückgeben.

Eine Priorität im herkömmlichen Sinne von NI-Tasks untereinander existiert nicht, weil NI-Tasks vom Betriebssystem einfach in einer bestimmten Reihenfolge aufgerufen werden, die durch die Reihenfolge ihrer Aktivierung gegeben ist. Da aber NI-Tasks auch mehrfach aktiviert werden können, kann man sagen, dass eine häufig aktivierte NI-Task statistisch auch häufiger aufgerufen wird und damit eine höhere Bedeutung hat.

Auch bei NI-Tasks besteht die Möglichkeit, eine bestimmte Task vorübergehend als die nächste Aufzurufende in der Reihenfolge ganz nach vorne zu schieben, so dass sie in jedem Fall als nächste NI-Task an der Reihe ist. Falls gerade eine NI-Task läuft, so wird damit gewartet, bis diese die Kontrolle an das Betriebssystem zurückgegeben hat.

5.3. Daten- und Parameterbereich

Bei der Installation eines Programms unter einer Task bekommt diese Task einen Datenbereich und einen Parameterbereich zugewiesen. Die Größe dieser beiden Bereiche wird vom Programm selbst, die des Datenbereichs kann auch beim Installieren festgelegt werden. Beide Bereiche sind später nicht mehr änderbar. Sie können auch die Länge 0 haben. Sie sind voneinander unabhängige, lineare, durchgängige Bereiche. Sie unterscheiden sich durch ihre maximal erlaubte Größe und durch die Art, wie auf ihre Inhalte zugegriffen wird.

5.3.1. Datenbereich

Der Zugriff auf den **Datenbereich** geschieht über Bibliotheksfunktionen unter Angabe der Tasknummer und des Offsets (in Bytes) bezogen auf den Anfang des Datenbereichs.

Der Datenbereich ist nicht segmentiert, so dass auch Speicherbereiche oberhalb 1 MB vom Betriebssystem genutzt werden können, seine Größe ist nicht beschränkt. Er eignet sich also besonders für fortlaufende Daten, wie sie zum Beispiel bei einem Messdatenerfassungsprogramm anfallen.

Der Datenbereich kann entweder im Programm selbst oder bei der Installation vom Betriebssystem reserviert werden.

Ist der Datenbereich im Programm enthalten, so ist seine Größe fest und kann nicht geändert werden (Bit-8 und Bit-9 in den PDT-Flags = 1, MAXPDT_DATA_IN-_PROG, MAXPDT_FIXED_DATASIZE). Üblicherweise ist der Datenbereich in diesem Fall als Array oder Struktur Teil der globalen Variablen des Programms und kann innerhalb des Programms mit normalen Variablenzugriffen benutzt werden. Seine Anfangsadresse und Größe müssen in die PDT eingetragen werden (ulDataAdr und ulDataSize).

Wird der Datenbereich vom Betriebssystem reserviert, gibt es mehrere Möglichkeiten, seine Größe festzulegen:

- Das Programm legt die Größe des Datenbereichs auf den in der PDT als Größe eingetragenen Wert fest (Bit-9 in den PDT-Flags = 1, MAXPDT_FIXED_DATA-SIZE)
- Die Größe wird erst beim Installieren angegeben (Bit-9 in den PDT-Flags=0). In diesem Fall wird im Installationsbefehl (genauer gesagt im Installationsflag, das von SNW32 bei MAXINST oder von der Host-PC-Bibliothek bei **max_trans-fer_and_install** übergeben wird) angegeben, ob der Eintrag in der PDT die Größe des Datenbereich bestimmt, oder ob die Datenbereichsgröße im Installationsbefehl übergeben wird. Dieses Verfahren wird zum Beispiel benutzt, um die Datenbe-reichsgröße dem aktuellen Messproblem anzupassen.

Um auf den Datenbereich zuzugreifen, der vom Betriebssystem reserviert wurde, müssen die Bibliotheksfunktionen verwendet werden (**max_get_data_sema, max_release_data_sema, max_get_data** und **max_set_data**).

5.3.2. Parameterbereich

Auf den **Parameterbereich** wird entweder direkt oder durch Bibliotheksfunktionen unter Angabe der Tasknummer und des Offsets (in Bytes) zugegriffen. Die Größe ist auf 64 kB – 256 beschränkt.

Der Parameterbereich ist ein Variablenbereich, der einer Task zugeordnet, aber über verschiedene Betriebssystemfunktionen auch für andere Tasks zugänglich ist. Jede Task kann also auf die Parameter einer anderen Task zugreifen. Über entsprechende PC-Bibliotheksroutinen können auch vom PC aus Parameter gelesen oder verändert werden.

Die Parameter einer Task sind von Null beginnend byteweise durchnummeriert (relative Adresse). Der Zugriff auf die Parameter von einer anderen Task oder vom PC aus erfolgt immer über die Angabe dieser Nummer. Variablen des Parameterbereiches, die aus mehreren Bytes bestehen (z.B. Integer-Werte), belegen mehrere Nummern. Um auf solch einen Parameter zuzugreifen, wird immer die erste Nummer angegeben.

Der Parameterbereich kann entweder Teil des Programms sein oder vom Betriebssystem bei der Installierung des Programms reserviert werden. Wenn das Programm den Parameterbereich selbst anlegt (Bit-10 in den PDT-Flags = 1, MAXPDT_PARAMETER_IN_PROG), dann geschieht das in der Regel mit einer Datenstruktur (**STRUCT** oder **RECORD**) im Variablenbereich. Die Adresse und die Größe dieser Struktur müssen in die PDT eingetragen werden (ulParAdr und ulParSize). Innerhalb des Programms ist der Zugriff auf den Parameterbereich dann einfach ein Zugriff auf die deklarierte Datenstruktur.

Wenn der Parameterbereich beim Installieren des Programms vom Betriebssystem angelegt wird (MAXPDT_PARAMETER_IN_PROG nicht gesetzt), muss nur die Größe des zu reservierenden Parameterbereichs in der PDT eingetragen sein. Der Eintrag ulParAdr in der PDT muss auf Null gesetzt werden. Um auf den Parameterbereich zugreifen zu können, muss das Programm mit Hilfe der Bibliotheksfunktionen Parameter lesen (max_read_par_xxxx) und schreiben (max_write_par_xxxx).

Üblicherweise werden im Parameterbereich Daten zur Konfiguration des Programms abgelegt, z.B. Abtastrate, Anzahl der zu messenden Kanäle, usw.

5.4. Datenpuffer

Das Betriebssystem stellt ringförmig organisierte Datenpuffer zur Verfügung. Es legt auf Anforderung einen Ringpuffer im Speicher des MAX-PC an und liefert ein Handle zurück, das für alle weiteren Zugriffe auf den Puffer benutzt wird. Zur Zeit können bis zu 256 voneinander unabhängige Puffer angelegt werden.

Ein Puffer arbeitet byteorientiert nach dem FIFO-Prinzip, d.h. die zuerst in den Puffer geschriebenen Byte werden auch als erste wieder ausgelesen.

5.5. Einige betriebssysteminterne Tasktabellen

Bei der Installation eines Programms unter einer Task werden außer Daten- und Parameterbereich auch einige Tabellen angelegt, die für die Verwaltung notwendig sind. Die beiden wichtigsten sind:

Programm-Deskriptor-Tabelle (PDT)

Task-Deskriptor-Tabelle (TDT)

Die Programm-Deskriptor-Tabelle (PDT) ist Teil des Programms, das auf dem MAX-PC geladen und unter einer Task installiert wird. Sie wird durch die Installation nicht verändert, auch das Programm selbst darf sie zur Laufzeit nicht mehr verändern. Es könnte ja z.B. sein, dass dasselbe Programm nochmals unter einer anderen Task installiert wird, dann werden die Informationen auch nochmals benötigt. Außerdem greift das Betriebssystem auch zur Laufzeit auf die Tabelle zu, allerdings nur lesend. Der Aufbau und eine genaue Erklärung der PDT finden Sie in Anhang D. Für den standardisierten Kopf der PDT ist in den Bibliotheken die Struktur MAX_PDT_HEAD definiert.

Die Task-Deskriptor-Tabelle (TDT) ist der jeweiligen Task zugeordnet. Sie enthält praktisch alle Informationen über die Task, über das unter der Task installierte Programm, z.B. auch die Adresse der PDT, und weitere Parameter, die vorwiegend zur Laufzeit benutzt werden. Die TDT steht direkt vor dem Parameterbereich der Task. Wenn der Parameterbereich im Programm enthalten ist, muss auch der Platz für die TDT im Programm reserviert werden. Dazu steht in den Echtzeitbibliotheken die Struktur **MAX_TDT_TYPE** zur Verfügung, mit deren Hilfe auch einfach auf die verschiedenen TDT-Einträge zugegriffen werden kann. Es werden außerdem weitere Informationen wie Interruptnummer und Tasknummer in dieser Tabelle gehalten. Eine ausführliche Beschreibung der TDT finden Sie in Anhang E.

5.6. Allgemeine Hinweise zur Programmierung

5.6.1. Unterschiede zur PC-Programmierung

Die wesentlichen Unterschiede zur PC-Programmierung ergeben sich aus der Struktur des Betriebssystems.

OsX ist ein sogenanntes Multi-Tasking Betriebssystem, das es erlaubt, mehrere Tasks „gleichzeitig“ auszuführen. Ein Scheduler des Betriebssystems teilt dazu den einzelnen Tasks Rechenzeit zu. Wird eine Task vom Scheduler aktiviert, ruft es dessen Hauptprozedur auf.

Ist die Hauptprozedur abgearbeitet, gibt das Programm die Kontrolle an den Scheduler zurück, d.h. an das Betriebssystem. Je nach Task-Typ kann eine Task auch während der Ausführung von einer höher priorisierten Task unterbrochen werden (siehe Kap. 5.3.). Abhängig vom Task-Typ wird das Programm automatisch später nochmals oder mehrmals wieder aufgerufen. Welcher Scheduler verwendet wird, wird festgelegt durch den Typ der Task, unter dem das Programm installiert wird.

Im Gegensatz zur Hauptprozedur wird die sogenannte Auto-Init-Prozedur nur einmal nach der Installation aufgerufen.

Die Prozedur, die bei einem DOS-Programm die Hauptprozedur ist (main) wird vom Betriebssystem des MAX-PC nur zum Zeitpunkt der Installation einmal aufgerufen. Sie hat die Aufgabe, die PDT zu initialisieren und dem Betriebssystem bekannt zu machen (**max_init_program**). Dieser Teil wird als PREPARE bezeichnet.

Im Unterschied zur DOS Programmierung müssen Sie sich beim Erstellen von Echtzeitprogrammen selbst um die "Anmeldung" des Programms kümmern, da die Compiler keine Informationen über das Betriebssystem des MAX-PC haben, um die notwendigen Tabellen selbst anzulegen. Das erledigen Sie, wie schon erwähnt, im PREPARE-Teil des Programms. Achten Sie unbedingt darauf, dass alle Einträge in der PDT zu Ihrem Programm passen.

5.6.2. Warten auf Ereignisse

Viele DOS-Programme verbringen den größten Teil der Zeit damit, auf Eingaben vom Benutzer zu warten. Unter DOS ist das kein Problem. Da die gesamte Rechenleistung ohnehin nur einem Programm zur Verfügung steht, kann dieses Programm die Tastatur einfach in einer Schleife abfragen und erst dann etwas tun, wenn eine Taste gedrückt ist.

Ganz anders sieht das im Multi-Tasking Betrieb des MAX-PC aus. Wenn eine Task auf das Schließen eines Schalters oder auf das Eintreffen eines Zeichens an einer Schnittstelle warten muss, bevor sie fortfahren kann, dann darf das nicht in einer Schleife wie bei DOS geschehen. In diesem Fall würde nicht nur die eine Task, sondern alle anderen Tasks (zumindest alle NI-Tasks) blockiert und kämen nicht an die Reihe. Statt einer direkten Schleife sollten Sie für die Abfrage eine NI-Task verwenden. Prüfen sie, ob das erwartete Ereignis eingetreten ist. Wenn ja, können Sie entsprechend der Aufgabenstellung fortfahren. Wenn das Ereignis noch nicht eingetreten ist, sollten Sie die NI-Task beenden und beim nächsten Durchlauf erneut prüfen, ob das Ereignis, eingetreten ist.

Beispiel:

Eine Task soll Programmteil I abarbeiten und anschließend warten, bis ein Schalter geschlossen ist. Dann soll Programmteil II abgearbeitet werden und wieder von vorne begonnen werden. In einem klassischen **DOS Programm** sähe das dann so aus:

1. Programmteil I ausführen
2. Schleife bis Schalter geschlossen
3. Programmteil II ausführen
4. Springe zu 1.

Als **NI-Task** könnte das Programm wie folgt aussehen:

Start-Prozedur oder Auto-Init Prozedur:

Statusvariable = 1 setzen

Haupt-Prozedur:

1. Statusvariable prüfen
   a) Status = 1: Weiter bei 2.1
   b) Status = 2: Weiter bei 3.1

2.1. Programmteil I ausführen
2.2. Statusvariable = 2 setzen
2.3. Hauptprozedur verlassen (oder bei 3.1 weiter)

3.1. Schalter prüfen
  a) Schalter offen: Hauptprozedur beenden
  b) Schalter geschlossen: Weiter bei 3.2.
3.2. Programmteil II ausführen
3.3. Statusvariable = 1 setzen
3.4. Hauptprozedur verlassen

Die Reaktionszeit auf das Schließen des Schalters ist in diesem Beispiel abhängig davon, wie stark der MAX-PC mit anderen Tasks ausgelastet ist, also letztlich davon, wie oft die NI-Task aufgerufen wird. Wenn Sie sehr schnell (im Mikrosekundenbereich) auf das Schließen des Schalters reagieren müssen, sollten Sie dafür sorgen, dass der Schalter einen Interrupt auslöst und den Programmteil II in einer Interrupt-Task erledigen.

5.6.3. Ermitteln der eigenen Tasknummer

Es gibt Fälle, in denen eine Task bestimmen muss, unter welcher Tasknummer sie selbst installiert ist, zum Beispiel dann, wenn mit einem Systemaufruf Daten in den Datenbereich geschrieben werden sollen (**max_set_data**). Je nach Installation der Task gibt es eine oder zwei Möglichkeiten, die Tasknummer zu bestimmen.

Wenn der Parameterbereich Teil des Programms ist, dann ist zwingend auch die Task-Deskriptor-Tabelle (TDT) im Programm enthalten. Die Tasknummer steht als Eintrag in der TDT. Sie können die Tasknummer direkt aus dem Eintrag usTaskNo der Struktur MAX_TDT_TYPE lesen.

In Fällen, in denen der Parameterbereich nicht vom Programm selbst, sondern vom Betriebssystem vergeben wird, kann die Tasknummer nicht aus der TDT ermittelt werden, da die TDT in diesem Fall nur unter Angabe der Tasknummer zugreifbar ist. Für diesen Fall liefern die Funktionen **max_entry_proc** und **max_entry_func** die Tasknummer zurück.

5.6.4. Verwenden von Fließkommaoperationen

Die Verwendung von Fließkommaoperationen unterliegt bei der Erstellung von Echtzeitprogrammen zwei Einschränkungen.

Die Emulation des Coprozessors ist nicht reentrant. Das heißt, dass eine laufende Berechnung nicht von einer Funktion unterbrochen werden darf, die ebenfalls Fließkommaoperationen enthält. Sicherheitshalber sollten deshalb bei MAX-PCs ohne Coprozessor keine Fließkommaberechnungen in Interrupt-Prozeduren bzw. in Prozeduren, die vom PC aus aufgerufen werden, enthalten sein. In der

Programmiersprache C gilt zusätzlich die Einschränkung, dass innerhalb von Prozeduren, die mit **max_entry_proc** oder **max_entry_func** beginnen, keine Fließkommaoperationen erlaubt sind. Statt dessen müssen die Berechnungen in einer eigenen (lokalen) Prozedur durchgeführt werden, die nicht mit **max_entry_proc** oder **max_entry_func** beginnt (und auch nicht in die PDT eingetragen wird). Diese Prozedur kann dann beliebig aufgerufen werden.

Beispiel:

```
void fp_calculations (void)
{
  ... /* beliebige Berechnungen */
}

void xyz (void)
{
  max_entry_proc();
  ...
  fp_calculations();
  ...
  max_exit_proc();
}
```

Beachten Sie, dass bereits die Zuweisung einer Fließkommazahl eine Fließkommaoperation ist, für die alle oben beschriebenen Einschränkungen gelten.

5.6.5. Objektorientierte Programmierung, dynamische Datenobjekte

Die Erstellung von objektorientierten Programmen mit C++ oder Pascal ist grundsätzlich möglich. Da aber die dynamische Speicherverwaltung (New, Dispose,...) von C++ bzw. Pascal auf dem MAX-PC nicht unterstützt wird, müssen alle Objekte statisch deklariert werden, dürfen also nicht auf dem Heap liegen. Achten Sie unbedingt darauf, dass auch die 'Vorfahren' eines Objektes keine dynamischen Speicherzugriffe (New, Dispose, ...) enthalten dürfen.

5.7. Installation von Programmen

Ein Anwendungsprogramm, das von OsX ausgeführt werden soll, muss zunächst in das RAM des MAX-PC geladen und anschließend installiert werden.

Zur komfortablen Installation der Programme können Sie das Hilfsprogramm

"SNW32"

nutzen.

Wenn Sie die Programme aus Ihrem eigenen PC-Programm heraus installieren wollen, stehen dafür Funktionen in der PC-Bibliothek zur Verfügung. Mehr darüber erfahren Sie in Kapitel 6.

On-board-Echtzeitprogrammdateien für OsX mit der Erweiterung ".EXE" dürfen nicht auf dem PC gestartet werden und umgekehrt für den PC geschriebene Programme nicht auf den MAX-PC geladen werden. **!**

Außerdem kann eine Liste zu installierender ROM-Programme im Flash-ROM abgelegt werden, die nach dem Einschalten des MAX-PC automatisch abgearbeitet wird.

5.7.1. Mehrfachinstallation von Echtzeitprogrammen

Die mehrfache Installation eines Programms bedeutet, dass der Programmcode nur einmal im RAM des MAX-PC vorhanden ist. Jede Instanz eines mehrfach installierten Programms hat jedoch ihre eigene Tasknummer, sowie ihren eigenen Parameter- und Datenbereich. Dies hat den Vorteil, dass weniger RAM-Speicher benötigt wird. Bei der Programmierung müssen jedoch einige Punkte beachtet werden.

In den Flags der Programm-Deskriptor-Tabelle (PDT) des Echtzeitprogramms dürfen die Flags MAXPDT_DATA_IN_PROG und MAXPDT_PARAMETER_IN_PROG nicht gesetzt sein (s. Anhang D). Dies bewirkt, dass der Parameter- und Datenbereich nicht vom Programm selbst, sondern vom Betriebssystem reserviert wird. Da dann Parameter- bzw. Datenbereich vom Programmcode getrennt sind, kann im Programm nicht direkt auf den eigenen Parameter- bzw. Datenbereich zugegriffen werden. Statt dessen müssen die Bibliotheksfunktionen dazu verwendet werden (z.B. **max_read_par_uchar** oder **max_set_data**).

Die PDT-Angabe 'Anfangsadresse Parameterbereich' wird in diesem Fall nicht ausgewertet. Dadurch ist eine Vereinbarung einer Variablen, die den Parameterbereich bildet, überflüssig.

Wenn in den globalen Prozeduren des Programms Funktionsaufrufe erfolgen, bei denen die Angabe der Tasknummer erforderlich ist, kann die Tasknummer durch den Rückgabewert von **max_entry_proc** bzw. **max_entry_func** ermittelt werden.

Die Mehrfachinstallation eines Programms kann nicht mit max_transfer_and_install geschehen, sondern erfordert zwei Schritte:

1. Laden des Programmcodes in das RAM des MAX-PC mit der Funktion **max_transfer_program**. Die Funktion liefert die Start-Adresse zurück.
2. Anschließend können mit der Funktion **max_install_task** unter Angabe der Tasknummer mehrere Instanzen des Programms erzeugt werden. Dazu übergeben Sie die von max_transfer_program gelieferte Start-Adresse an **max_install_task.**

5.8. Installieren eines neuen Betriebssystems

Nach einem Reset der Trägerkarte bzw. des MAX-PC ist zunächst immer das sogenannte ‘Mini-Betriebssystem’ im ROM des MAX-PC aktiv. Es dient im wesentlichen zum Laden und Aktivieren eines vollständigen Betriebssystems und zu Debugging-Zwecken (z.B. Post-mortem-Analyse nach Programmabsturz). Das Mini-Betriebssystem arbeitet ohne Verwendung von RAM und Interrupts und beherrscht nur einige Makrobefehle.

Neben dem Mini-Betriebssystem enthält das ROM des MAX-PC ein vollständiges Betriebssystem (im folgenden 'ROM-Betriebssystem' genannt), das nach einem Reset erst aktiviert werden muss (Ausnahme: Ist der MAX-PC per serieller Ankopplung mit einem Host-PC verbunden bzw. sind RxD und TxD der seriellen Schnittstelle kurzgeschlossen, wird das ROM-Betriebssystem automatisch aktiviert). Aus Geschwindigkeitsgründen wird dieses Betriebssystem vom ROM ins RAM kopiert und dann gestartet. Das Aktivieren geschieht durch die Funktion **max_load_osx** oder aus SNW32 im Assistenten zum jeweiligen MAX-PC. Ein Update des Betriebssystems im ROM kann mit SNW32 vorgenommen werden.

Das Betriebssystem des MAX-PC kann statt aus dem ROM auch vom Host-PC aus ins RAM des MAX-PC geladen werden. Auch dieser Vorgang kann sehr einfach und schnell mit der Bibliotheksfunktion **max_load_osx** und von SNW32 (unter Angabe des Namens einer Datei, die ein Betriebssystem enthält) erledigt werden. Wenn das Betriebssystem läuft, besteht kein Unterschied, ob es aus dem ROM oder vom Host-PC geladen wurde. Die Bibliotheken enthalten Funktionen, mit denen ermittelt werden kann, welcher Typ und welche Version des Betriebssystems aktuell auf dem MAX-PC läuft.

Der Name der Betriebssystemdatei enthält Versionsinformationen und ist wie folgt aufgebaut:

Xx-za.nnR Neues Betriebssystem
x = 1: MAX-PC X-MAX-1, x=E: MAX-PC X-MAX-E
z = Zahl 0 bis 9 (Version), a = Buchstabe A bis Z (Revision),
nn = lfd. Nr. des Betriebssystems)

Das Programm SNW32 sowie die Bibliotheksfunktion **max_load_osx** bieten die Möglichkeit, das jeweils aktuellste Betriebssystem zu laden. Dazu muss bei **max_load_osx** als Name MAX1_NEWEST_OSX angegeben werden. Aus mehreren im Verzeichnis 'OSX' stehenden Betriebssystem-Dateien wird dann automatisch die neueste Version geladen.

5.9. Aufbau des RAM-Bereichs des MAX-PC

Das Betriebssystem residiert in den unteren 64 kByte. Die Interrupt-Vektor-Tabelle beginnt ab Adresse 0 und belegt 1 kByte. Nach einem Hardware-Reset und der Aktivierung des Betriebssystems können Sie die Grenzen des noch freien RAM vom MAX-PC lesen (**max_get_ram_info**). Die Grenzen stehen im Parameterbereich des Betriebssystems, also der Task 0 (siehe Anhang F). Es werden immer physikalische 32-Bit-Adressen geliefert. Wenn ein Programm auf den MAX-PC geladen und installiert wird, wird die untere bzw. obere Grenze des freien RAM entsprechend der Größe des Programms und gegebenenfalls der Größe des Parameter- und Datenbereichs der gerade installierten Task verändert, also nach oben verschoben. Auch bestimmte Aktionen des Betriebssystems belegen zusätzlich etwas Speicherplatz.

Bei der aktuellen Version des Betriebssystems, das im Real Mode arbeitet, wird von den Anwendungsprogrammen erwartet, dass sie ebenfalls im Real Mode arbeiten. Nur die unteren 1 MByte des Speichers können also für Programme genutzt werden. Parameter- und Datenbereiche können auch im darüber hinausgehenden Bereich liegen (je nach Speicherausbau z.B. bis 32 MByte).

5.10. Parameterbereich des Betriebssystems

Das Betriebssystem selbst ist ebenfalls eine Task auf dem MAX-PC: Task 0. Damit stehen alle Taskbefehle auch für den Zugriff auf die Strukturen des Betriebssystems zur Verfügung. Das Betriebssystem hat dementsprechend auch einen eigenen Parameterbereich, der teilweise auch für den Anwender von Interesse sein kann. Hier nicht erklärte bzw. reservierte Parameter dürfen auf keinen Fall verändert werden.

Details zu den Parametern finden Sie im Anhang F.

5.11. Allgemeines zu den Programmbeispielen

Im folgenden soll an einfachen Programmbeispielen in Borland-Pascal und C++ gezeigt werden, wie Programme in Hochsprachen erstellt werden. Alle Programme finden Sie auch auf der mitgelieferten SORCUS-CD (oder auf unserer Homepage www.sorcus.com). Um erste Eindrücke zu sammeln, ist es hilfreich, wenn Sie zuerst ein wenig mit diesen Programmen experimentieren.

Die hier beschriebenen Programme lassen lediglich die Leuchtdiode auf dem MAX-PC blinken, einmal als NI-Task und einmal als II-Task. Es wird kein Datenbereich benutzt.

Als Entwicklungsumgebung können Sie Borland-Pascal für DOS oder Borland C++ oder RTDS (Kapitel 9) verwenden.

5.11.1. Programmbeispiele für Borland-Pascal

5.11.1.1. NI-Task in Borland-Pascal

Das folgende Programmbeispiel zeigt das Blink-LED Programm als NI-Task. Sie finden das komplette Programm im Sourcecode unter den Namen "LED_NI.PAS" auf der mitgelieferten SORCUS-CD.

```
{$F+}                                    { FAR Calls ! }
{$N-}                                    { No Coprocessor }
{$E-}                                    { No Emulation (application specific) }
{$R-}                                    { No Range-Check }
{$S-}                                    { No Stack-Checking }
{$I-}                                    { No IO-Checking }
{$V-}                                    { No strict var string check}

{$M 1024, 0, 0}                          { Stack und Heap auf Minimum }

uses
   Maxlib, X_error, maxtype, maxdefs, maxmdd;
```

```
{ the program uses the following SORCUS units
  Maxlib   - all functions and procedure declarations
  X_error  - defined error message constants
  maxtype  - all type definitions used in the library
  maxdefs  - various constant definitions
  maxmdd   - needed for MDDs: It defines CPS-types and constants
}

{ constant definitions }
const
   PROGRAMM_NUMBER : WORD = $9000;
   VERSION   : char       = '1';
   REVISION  : char       = 'C';
   SRQ_ERROR : WORD       = $8182;

type
   { program descriptor table }
   pdt_type = RECORD
      rcHead    : MAX_PDT_HEAD;        { standard PDT header }
      main_proc : POINTER;             { pointer to proc. 0  }
      auto_init : POINTER;             { pointer to proc. 1  }
      start     : POINTER;             { pointer to proc. 2 (optional) }
      stop      : POINTER;             { pointer to proc. 3 (optional) }
   end;

   { parameter area (is part of the program
     because PDT-flag MAXPDT_PARAMETER_IN_PROG is set) }
   parameter_type = RECORD
      rcTDT       : MAX_TDT_TYPE ;     { must be before the parameter area }
{ Offset 0: }
      Error       : MAX_ERROR;        { error code if error occurs }
      usLedState  : WORD;             { status of LED: 0 = off, 1 = on }
      usBlinkrate : WORD;             { Number of task calls after that the LED shall be
                                        switched over }
      usSendSRQs  : WORD;             { set to 1, if a SRQ has to be sent to the Host-PC,
                                        if LED is switched over }
      usCounter   : WORD;             { counter variable }
{ Offset 10: }
      usSRQMsg    : WORD;             { SRQ-word to be sent, if usSendSRQs = 1}
      usHostNo    : WORD;             { number of Host-PC, to that the SRQ has to be sent }
      hMyself     : MAXMODHND;        { handle of own CPU }
      hHost       : MAXMODHND;        { handle of own Host }
{ Offset 20: }
      rcOsX       : MAX_OS_INFO;      { version of OsX }
   end;

{ global variables }
var
   rcPDT               : pdt_type;
   parameter           : parameter_type;

{*************************************************************************}
{ in case of error send srq to PC                                         }
PROCEDURE ErrorHandler;
begin
   parameter.Error := max_send_srq (parameter.hHost, 0, SRQ_ERROR);
end;

{*************************************************************************}
{ Init task parameter                                                     }
PROCEDURE AutoInit; far;
begin
   max_entry_proc;
   parameter.usBlinkrate := 8000;
   parameter.usCounter   := 0;
   parameter.usSendSRQs  := 0;
   parameter.usSRQMsg    := $9190;
   parameter.usHostNo    := 1;         { PC connected via PCI }
   parameter.Error       := max_init_lib(0);
   if (parameter.Error = ERR_OK) then begin
```

5

```
      { get a handle for the X-MAX-1 }
      parameter.Error := max_connect_cpu (0, MAX_MYSELF, 0,
                                          parameter.rcOsX, parameter.hMyself);
      if (parameter.Error = ERR_OK) then
         parameter.Error := max_led_off (parameter.hMyself);
      parameter.usLedState := 0;       { Led is switched off }
   end;
   max_exit_proc;
end;

{*****************************************************************************}
{ main procedure of the task (called automatically by OsX after activating the task) }
PROCEDURE MainProc; far;
begin
   max_entry_proc;
   Inc(parameter.usCounter);
   if (parameter.usCounter > parameter.usBlinkrate) then begin
      if (parameter.usLedState = 0) then begin
         max_led_on(parameter.hMyself);
         parameter.usLedState := 1;
      end
      else begin
         max_led_off(parameter.hMyself);
         parameter.usLedState := 0;
      end;
      parameter.usCounter := 0;
      { transmit srq to the host pc if parameter usSendSRQs is set }
      if (parameter.usSendSRQs <> 0) then
         max_send_srq (parameter.hHost, 0, parameter.usSRQMsg);
   end;
   max_exit_proc;
end;

{*****************************************************************************}
{ Start the task                                                              }
PROCEDURE Start; far;
var
   usTask : WORD;
   Dummy  : MAX_OS_INFO;
begin
   usTask := max_entry_proc;
   { get a handle for the host PC (to send SRQs) }
   parameter.Error := max_connect_cpu (parameter.usHostNo, 0, 0, Dummy, parameter.hHost);
   parameter.Error := max_wakeup_ni_task (parameter.hMyself, usTask, 1);
   max_exit_proc;
end;

{*****************************************************************************}
{ Stop the task                                                               }
PROCEDURE Stop; far;
var
  usTask : WORD;
begin
   usTask := max_entry_proc;
   parameter.Error := max_sleep_task (parameter.hMyself, usTask);
   max_exit_proc;
end;

{*****************************************************************************}

begin
   { the PDT-flags determine the task-type and
    information how the data and parameter area are allocated }
   rcPDT.rcHead.usFlags         := MAX_NI_TASK +
                                   MAXPDT_PARAMETER_IN_PROG; { parameter are part of the
                                                               program }

   { interrupt source (0 for NI-tasks) }
   rcPDT.rcHead.usInterrupt     := 0;
```

```
   { the program number identifies the program uniquely }
   rcPDT.rcHead.usProgNo        := PROGRAMM_NUMBER;

   { number of global task procedures in the PDT }
   rcPDT.rcHead.usProcedures    := (sizeof(rcPDT) - sizeof (MAX_PDT_HEAD)) div 4;

   { the following lines are just for documentation and information purpose }
   rcPDT.rcHead.ucType          := 1;   { always ! }
   rcPDT.rcHead.ucSize          := sizeof(MAX_PDT_HEAD);
   rcPDT.rcHead.cVersion        := VERSION;
   rcPDT.rcHead.cRevision       := REVISION;
   rcPDT.rcHead.ucCPU           := MAX_CPU486;
   rcPDT.rcHead.ucCoProc        := _NOCOPROC;
   rcPDT.rcHead.ucLanguage      := MAXRT_LANGUAGE_PAS;
   rcPDT.rcHead.ucProgType      := 0;   { not used }
   rcPDT.rcHead.ulRes1          := 0;
   rcPDT.rcHead.ulRes2          := 0;
   rcPDT.rcHead.ulHyperAdr      := 0;   { feature not available yet }

   { address and size of the tasks data area
     only when MAXPDT_DATA_IN_PROG is set in PDT-flags }
   rcPDT.rcHead.ulDataAdr       := 0;
   rcPDT.rcHead.ulDataSize      := 0;

   { address and size of the tasks parameter area
     only when MAXPDT_PARAMETER_IN_PROG is set in PDT-flags }
   rcPDT.rcHead.ulParAdr  := max_get_physical_addr(@parameter) + sizeof(MAX_TDT_TYPE);
   rcPDT.rcHead.usParSize := sizeof(parameter) - sizeof(MAX_TDT_TYPE);

   { initialize global procedure addresses in PDT }
   rcPDT.main_proc        := @MainProc;
   rcPDT.auto_init        := @AutoInit;
   rcPDT.start            := @Start;
   rcPDT.stop             := @Stop;
   { transfer PDT information to the OsX }
   max_init_program (@rcPDT);
end.
```

Da das Programm insgesamt umfangreich kommentiert ist, soll jetzt nur noch auf Besonderheiten eingegangen werden.

Als erstes werden die SORCUS-Units eingebunden. Diese Units enthalten die in Kapitel 6 beschriebenen Funktionen und Definitionen Anschließend wird die PDT deklariert. Sie sehen am Ende der PDT, dass das Programm vier globale Prozeduren enthält.

Außerdem wird noch ein Record vereinbart, der die Parameter enthält. Die Parameter werden also lokal im Programm gehalten, der Parameterbereich wird nicht vom Betriebssystem reserviert. Beachten Sie bitte, dass vor den eigentlichen Parametern Platz für die TDT reserviert wird.

Alle globalen (also in der PDT aufgeführten Prozeduren) beginnen mit **max_entry_proc** und enden mit **max_exit_proc**. Dadurch ist sichergestellt, dass alle Prozessor-Register gesichert und für die Prozedur richtig eingestellt werden.

In der "Auto-Init-Prozedur" werden die Parameter initialisiert, Handle geholt und die LED ausgeschaltet. Der Teil, der zwischen "**begin**" und "**end.**" steht, enthält in einem "normalen" Pascal Programm das eigentliche Hauptprogramm. Dieser Teil wird während der Installierung abgearbeitet. Er sorgt dafür, dass die PDT initialisiert wird

und dem MAX-PC Betriebssystem die Adresse der PDT mitgeteilt wird (**max_init_program**), beachten Sie bitte die Einstellung der PDT-Flags.

5.11.1.2. Installation der NI-Task

Nachdem das Programm "LED_NI.PAS" einwandfrei übersetzt und die Datei "LED_NI.EXE" erzeugt wurde, kann das Programm auf den MAX-PC geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW32 ein, und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "LED_NI.INS" benutzen.

```
; Beispielinstallation: LED-Blinken als NI-Task (ohne Interrupt)
;
; Reset einer MAX-Trägerkarte
MAXRESET board=0
;
; MAX-PC anwählen und Slot-Layer-Nummer einstellen
MAXCONNECTCPU board=0 slot=1 layer=0
;
; Osx auf MAX-PC laden
MAXLOADOSX osx="ROM"
;
; Programmnummer 9000h unter Tasknummer 21h installieren
MAXINST file="LED_NI.EXE" no=9000 task=21 tasktype=MAX_NI_TASK irq=0 level=0
        datasize=0 autoinit
;
; Parameter der Task 21h setzen
; PAR-4,5 : Blinkrate der LED    => 1000h
;
MAXPAR task=21 start=4 para=00 10
;
; Durch Aufruf der Start-Prozedur Programm starten (Proz. 2)
MAXPROC task=21 proc=2
```

Diese Installationsdatei installiert die NI-Task, stellt die Blinkrate ein und startet das Programm durch Aufruf der Prozedur 2.

5.11.1.3. II-Task in Borland-Pascal

Das folgende Listing zeigt das Blink-LED Programm als II-Task für den MAX-PC. Dieses Programm finden Sie auch unter dem Namen "LED_II.PAS" auf der mitgelieferten CD. Das Programm wird unter dem Interrupt des TIMER-A installiert. Die Blinkrate kann in Hertz angegeben werden.

```
{$F+}                                    { FAR Calls ! }
{$N-}                                    { No Coprocessor }
{$E-}                                    { No Emulation (application specific) }
{$R-}                                    { No Range-Check }
{$S-}                                    { No Stack-Checking }
{$I-}                                    { No IO-Checking }
{$V-}                                    { No strict var string check}

{$M 1024, 0, 0}                          { Stack und Heap auf Minimum }

uses
   Maxlib, X_error, maxtype, maxdefs, maxmdd;
{ the program uses the following SORCUS units
  Maxlib   - all functions and procedure declarations
  X_error  - defined error message constants
  maxtype  - all type definitions used in the library
  maxdefs  - various constant definitions
  maxmdd   - needed for MDDs: It defines CPS-types and constants
}

{ constant definitions }
const
   PROGRAMM_NUMBER : WORD = $9001;
   VERSION   : char       = '1';
   REVISION  : char       = 'C';
   SRQ_ERROR : WORD       = $8182;

type
   { program descriptor table }
   pdt_type = RECORD
      rcHead    : MAX_PDT_HEAD;        { standard PDT header }
      main_proc : POINTER;             { pointer to proc. 0  }
      auto_init : POINTER;             { pointer to proc. 1  }
      start     : POINTER;             { pointer to proc. 2 (optional) }
      stop      : POINTER;             { pointer to proc. 3 (optional) }
   end;

   { parameter area (is part of the program
     because PDT-flag MAXPDT_PARAMETER_IN_PROG is set) }
   parameter_type = RECORD
      rcTDT       : MAX_TDT_TYPE ;     { must be before the parameter area }
{ Offset 0: }
      Error       : MAX_ERROR;        { error code if error occurs }
      usLedState  : WORD;             { status of LED: 0 = off, 1 = on }
      usBlinkrate : WORD;             { Number of task calls after that the LED shall be
                                        switched over }
      usSendSRQs  : WORD;             { set to 1, if a SRQ has to be sent to the Host-PC, if
                                        LED is switched over }
      usTimerFrequency : WORD;        { timer vaule in multiples of 0.84 µs }
{ Offset 10: }
      usSRQMsg    : WORD;             { SRQ-word to be sent, if usSendSRQs = 1}
      usHostNo    : WORD;             { number of Host-PC, to that the SRQ has to be sent }
      hMyself     : MAXMODHND;        { handle of own CPU }
      hHost       : MAXMODHND;        { handle of own Host }
{ Offset 20: }
      rcOsX       : MAX_OS_INFO;      { version of OsX }
end;
```

5

```
{ global variables }
var
   rcPDT               : pdt_type;
   parameter           : parameter_type;
   usTimerCounter      : WORD;          { global counter variable }

{*************************************************************************}
{ in case of error send srq to PC                                         }
PROCEDURE ErrorHandler;
begin
   parameter.Error := max_send_srq (parameter.hHost, 0, SRQ_ERROR);
end;

{*************************************************************************}
{ Init task parameter                                                     }
PROCEDURE AutoInit; far;
begin
   max_entry_proc;
   parameter.usBlinkrate := 8000;
   parameter.usTimerFrequency := 11892; { = 100Hz (1 unit = 0,84us)
             LED blink frequency = usTimerFrequency div usBlinkrate }
   parameter.usSendSRQs  := 0;
   parameter.usSRQMsg    := $9190;
   parameter.usHostNo    := 1;         { PC connected via PCI }
   usTimerCounter        := 0;
   parameter.Error       := max_init_lib(0);
   if (parameter.Error = ERR_OK) then begin
      { get a handle for the X-MAX-1 }
      parameter.Error := max_connect_cpu (0, MAX_MYSELF, 0,
                                          parameter.rcOsX, parameter.hMyself);
      if (parameter.Error = ERR_OK) then
         parameter.Error := max_led_off (parameter.hMyself);
      parameter.usLedState := 0;       { Led is switched off }
   end;
   max_exit_proc;
end;

{*************************************************************************}
{ main procedure of the task (called automatically by OsX after activating the task) }
PROCEDURE MainProc; far;
begin
   max_entry_proc;
   Inc(usTimerCounter);
   { LED blinking can be extended with a software counter }
   if (usTimerCounter = parameter.usBlinkrate)  then begin
      if (parameter.usLedState = 0) then begin
         max_led_on(parameter.hMyself);
         parameter.usLedState := 1;
      end
      else begin
         max_led_off(parameter.hMyself);
         parameter.usLedState := 0;
      end;
      usTimerCounter := 0;
      { transmit srq to the host pc if parameter usSendSRQs is set }
      if (parameter.usSendSRQs <> 0) then
         max_send_srq (parameter.hHost, 0, parameter.usSRQMsg);
   end;
   max_exit_proc;
end;

{*************************************************************************}
{ Start the task                                                           }
PROCEDURE Start; far;
var
   usTask : WORD;
   Dummy  : MAX_OS_INFO;
begin
   usTask := max_entry_proc;
   { get a handle for the host PC (to send SRQs) }
```

```
   parameter.Error := max_connect_cpu (parameter.usHostNo, 0, 0, Dummy, parameter.hHost);
   parameter.Error := max_set_timer (parameter.hMyself, MAX1_TIMER_C,
                                     MAX1_TIMER_MODE_INT, parameter.usTimerFrequency);
   parameter.Error := max_clear_int (parameter.hMyself, parameter.rcTDT.ucInterrupt);
   parameter.Error := max_unmask_int (parameter.hMyself, parameter.rcTDT.ucInterrupt);
   parameter.Error := max_wakeup_ii_task (parameter.hMyself, usTask);
   max_exit_proc;
end;

{*****************************************************************************}
{ Stop the task                                                               }
PROCEDURE Stop; far;
var
  usTask : WORD;
begin
   usTask := max_entry_proc;
   parameter.Error := max_sleep_task (parameter.hMyself, usTask);
   max_exit_proc;
end;

{*****************************************************************************}

begin
   { the PDT-flags determine the task-type and
    information how the data and parameter area are allocated }
   rcPDT.rcHead.usFlags         := MAX_II_TASK +    { parameter are part of the program }
                                   MAXPDT_PARAMETER_IN_PROG;
   { interrupt source (0 for NI-tasks) }
   rcPDT.rcHead.usInterrupt     := MAX1_IRQ_TIMERC;
   { the program number identifies the program uniquely }
   rcPDT.rcHead.usProgNo        := PROGRAMM_NUMBER;
   { number of global task procedures in the PDT }
   rcPDT.rcHead.usProcedures    := (sizeof(rcPDT) - sizeof (MAX_PDT_HEAD)) div 4;
   { the following lines are just for documentation and information purpose }
   rcPDT.rcHead.ucType          := 1;   { always ! }
   rcPDT.rcHead.ucSize          := sizeof(MAX_PDT_HEAD);
   rcPDT.rcHead.cVersion        := VERSION;
   rcPDT.rcHead.cRevision       := REVISION;
   rcPDT.rcHead.ucCPU           := MAX_CPU486;
   rcPDT.rcHead.ucCoProc        := _NOCOPROC;
   rcPDT.rcHead.ucLanguage      := MAXRT_LANGUAGE_PAS;
   rcPDT.rcHead.ucProgType      := 0;   { not used }
   rcPDT.rcHead.ulRes1          := 0;
   rcPDT.rcHead.ulRes2          := 0;
   rcPDT.rcHead.ulHyperAdr      := 0;   { feature not available yet }

   { address and size of the tasks data area
     only when MAXPDT_DATA_IN_PROG is set in PDT-flags }
   rcPDT.rcHead.ulDataAdr       := 0;
   rcPDT.rcHead.ulDataSize      := 0;

   { address and size of the tasks parameter area
     only when MAXPDT_PARAMETER_IN_PROG is set in PDT-flags }
   rcPDT.rcHead.ulParAdr  := max_get_physical_addr(@parameter) + sizeof(MAX_TDT_TYPE);
   rcPDT.rcHead.usParSize := sizeof(parameter) - sizeof(MAX_TDT_TYPE);

   { initialize global procedure addresses in PDT }
   rcPDT.main_proc        := @MainProc;
   rcPDT.auto_init        := @AutoInit;
   rcPDT.start            := @Start;
   rcPDT.stop             := @Stop;
   { transfer PDT information to the OsX }
   max_init_program (@rcPDT);
end.
```

Da auch hier das Listing ausführlich kommentiert ist, soll nur auf Besonderheiten eingegangen werden.

Die Start-Prozedur berechnet zuerst aus der angegebenen Frequenz die Timer-Werte. Beachten Sie bitte dabei, dass die Timer in der Regel mit einer Eingangsfrequenz von 1,1892 MHz getaktet werden, und dass jeder Timer 16 Bit breit ist. Da mit diesen Werten "nur" Blinkfrequenzen zwischen 1,1892 MHz und 17 Hz möglich sind, wurde der Hardware-Timer durch einen Softwarezähler erweitert. Erst wenn dieser Softwarezähler den Wert des Parameters *usBlinkrate* erreicht hat, wird die LED umgeschaltet. Bei der Programmierung wurde darauf geachtet, dass der Timer-Wert möglichst groß ist, damit die Anzahl der Interrupts vom Timer pro Zeiteinheit möglichst klein ist.

Die Hauptprozedur unterscheidet sich nicht von der Hauptprozedur der NI-Task.

! *Die Hauptroutine darf nicht als '**interrupt**'-Prozedur deklariert werden. Das durch die **interrupt**-Anweisung bewirkte Retten der Register wird mit **max_entry_proc** erledigt, das Beenden der Prozedur mit IRET mit **max_exit_proc** und durch OsX.*

5.11.1.4. Installation der II-Task

Nachdem das Programm "LED_II.PAS" einwandfrei übersetzt und die Datei "LED_II.EXE" erzeugt wurde, kann das Programm auf den MAX-PC geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW32 ein und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "LED_II.INS" benutzen.

```
; Beispielinstallation: LED-Blinken mit Timer-Interrupt (Timer-A)
;
; Reset einer MAX-Trägerkarte
MAXRESET board=0
;
; MAX-PC anwählen und Slot-Layer-Nummer einstellen
MAXCONNECTCPU board=0 slot=1 layer=0
;
; Osx auf MAX-PC laden
MAXLOADOSX osx="ROM"
;
; Programmnummer 9001h unter Tasknummer 22h installieren
MAXINST file="LED_II.EXE" no=9001 task=22 tasktype=MAX_II_TASK irq=78 level=0
        datasize=0 autoinit
;
; Parameter der Task 22h setzen
; PAR-4,5 : Blinkrate der LED    => 1000h
;
MAXPAR task=22 start=4 para=05 00 00 01
;
; Durch Aufruf der Start-Prozedur Programm starten (Proz. 2)
MAXPROC task=22 proc=2
```

Die Installationsdatei installiert die II-Task und aktiviert sie anschließend.

Beachten Sie bitte, dass dieses Programm im Gegensatz zur NI-Task nur einmal (!) installiert werden kann, da es auch nur einen TIMER-A auf dem MAX-PC gibt. Sie könnten als Übung versuchen, das vorhandene Programm unter einen anderen Namen zu kopieren und es so zu modifizieren, dass das neue Programm z.B. den TIMER-C benutzt.

5.11.2. Programmbeispiele für Borland C

5.11.2.1. NI-Task in Borland C

Das folgende Programmbeispiel für den MAX-PC zeigt das Blink-LED Programm als NI-Task. Sie finden dieses Programm auch unter dem Namen "LED_NI.C" auf der mitgelieferten CD. Zum Kompilieren benutzen Sie bitte die ebenfalls auf der CD vorhandene Projektdatei "LED_NI.PRJ" (bzw. LED_NI.IDE). Bitte überprüfen Sie vor dem Kompilieren die eingestellten Pfade ("Options/Directories").

```
#pragma option -r-

#define _RTLIB_                         // must be defined in every real time
                                        // program before including max_lib.h
#include "max_lib.h"
#include "x_error.h"
#include "dos.h"
#include "stdlib.h"

#define PROGRAMM_NUMBER 0x9000
#define VERSION         '1'
#define REVISION        'C'
#define SRQ_ERROR       0x8182

// PDT structure
struct pdt_type
{
   MAX_PDT_HEAD    rcHead;
   void* MAXEXPORT main_proc;
   void* MAXEXPORT auto_init;
   // any number of user defined procedures may follow (optionally)
   void* MAXEXPORT start;
   void* MAXEXPORT stop;
} rcPDT;

// parameter area structure
struct parameter_type
{
   MAX_TDT_TYPE rcTDT;                  // always before the first parameter !
// offset 0
   MAX_ERROR    Error;                  // Errorcode
   USHORT       usLedState;             // Actual state of the LED
   USHORT       usBlinkrate;            // Controls of the blinking frequency
   USHORT       usSendSRQs;             // Control if SRQ should be sent with every
                                        // change of the LED state
   USHORT       usCounter;              // Actual Frequency counter
// offset 10
```

```
   USHORT       usSRQMsg;              // SRQ message to be used if usSendSRQs=1
   USHORT       usHostNo;              // number of the target (PC) for srqs
   MAXMODHND    hMyself;               // Handle for the CPU module
   MAXMODHND    hHost;                 // Handle for the PC (to send SRQs)
// offset 20
   MAX_OS_INFO  rcOsX;                 // Version info about OsX
} parameter;

/*****************************************************************************/
// in case of error send srq to PC
// this is a local procedure - no max_entry / max_exit must be called
void ErrorHandler(void)
{
   parameter.Error = max_send_srq (parameter.hHost, 0, SRQ_ERROR);
}

/*****************************************************************************/
// main procedure of the task (called automatically by OsX after activating the task)
void MAXEXPORT MainProc(void)
{
   max_entry_proc();
   if (++parameter.usCounter >= parameter.usBlinkrate) {
      if (parameter.usLedState == 0) {
         max_led_on(parameter.hMyself);
         parameter.usLedState = 1;
      }
      else {
         max_led_off(parameter.hMyself);
         parameter.usLedState = 0;
      }
      parameter.usCounter = 0;
      // transmit srq to the host pc if parameter usSendSRQs is set
      if (parameter.usSendSRQs)
         max_send_srq (parameter.hHost, 0, parameter.usSRQMsg);
   }
   max_exit_proc();
}

/*****************************************************************************/
// Init task parameter
void MAXEXPORT AutoInit(void)
{
   max_entry_proc();
   parameter.usBlinkrate = 8000;
   parameter.usCounter   = 0;
   parameter.usSendSRQs  = 0;
   parameter.usSRQMsg    = 0x9190;
   parameter.usHostNo    = 1;          // number of target for SRQs
   parameter.Error = max_init_lib(0);
   if (parameter.Error == ERR_OK) {
      // get a handle for the X-MAX-1
      parameter.Error = max_connect_cpu (0, MAX_MYSELF, 0,
                                         &parameter.rcOsX, &parameter.hMyself);
      if (parameter.Error == ERR_OK) {
         max_led_off(parameter.hMyself);
         parameter.usLedState = 0;          // Led is switched off
      }
   }
   max_exit_proc();
};
```

```
/*****************************************************************************/
// start the task
void MAXEXPORT Start(void)
{
   USHORT      usTask;
   usTask = max_entry_proc();
   // get a handle for the host PC (to send SRQs via PCI-bus)
   parameter.Error = max_connect_cpu (parameter.usHostNo, 0, 0, NULL, &parameter.hHost);
   if (parameter.Error == ERR_OK)
      parameter.Error = max_wakeup_ni_task (parameter.hMyself, usTask, 1);
   max_exit_proc();
}

/*****************************************************************************/
// stop the task
void MAXEXPORT Stop(void)
{
   USHORT usTask;
   usTask = max_entry_proc();
   parameter.Error = max_sleep_task (parameter.hMyself, usTask);
   max_exit_proc();
}

/*****************************************************************************/
// initialize the Program Descriptor Table (PDT)
void main ()
{
   // the PDT-flags determine the task-type and
   // information how the data and parameter area are allocated
   rcPDT.rcHead.usFlags         = MAX_NI_TASK +          // parameter are part of the program
                                  MAXPDT_PARAMETER_IN_PROG;
   // interrupt source (0 for NI-tasks)
   rcPDT.rcHead.usInterrupt     = 0;

   // the program number identifies the program uniquely
   rcPDT.rcHead.usProgNo        = PROGRAMM_NUMBER;

   // number of global task procedures in the PDT
   rcPDT.rcHead.usProcedures    = (sizeof(rcPDT) - sizeof (MAX_PDT_HEAD)) / 4;

   // the following lines are just for documentation and information purpose
   rcPDT.rcHead.ucType          = 1;   // always !
   rcPDT.rcHead.ucSize          = sizeof(MAX_PDT_HEAD);
   rcPDT.rcHead.cVersion        = VERSION;
   rcPDT.rcHead.cRevision       = REVISION;
   rcPDT.rcHead.ucCPU           = MAX_CPU486;
   rcPDT.rcHead.ucCoProc        = _NOCOPROC;
   rcPDT.rcHead.ucLanguage      = MAXRT_LANGUAGE_C;
   rcPDT.rcHead.ucProgType      = 0;   // not used
   rcPDT.rcHead.ulRes1          = 0;
   rcPDT.rcHead.ulRes2          = 0;
   rcPDT.rcHead.ulHyperAdr      = 0;   // feature not available yet
   // address and size of the tasks data area
   // only when MAXPDT_DATA_IN_PROG is set in PDT-flags
   rcPDT.rcHead.ulDataAdr       = 0;
   rcPDT.rcHead.ulDataSize      = 0;
   // address and size of the tasks parameter area
   // only when MAXPDT_PARAMETER_IN_PROG is set in PDT-flags
   rcPDT.rcHead.ulParAdr  = max_get_physical_addr(&parameter) + sizeof(MAX_TDT_TYPE);
   rcPDT.rcHead.usParSize = sizeof(parameter) - sizeof(MAX_TDT_TYPE);

   // initialize global procedure addresses in PDT
   rcPDT.main_proc        = MainProc;
   rcPDT.auto_init        = AutoInit;
   rcPDT.start            = Start;
   rcPDT.stop             = Stop;

   // transfer PDT information to the OsX
   max_init_program ((MAX_PDT_HEAD*)&rcPDT);
}
```

Da das Programm insgesamt ausführlich kommentiert ist, soll nur auf Besonderheiten eingegangen werden.

Um dieses Programm zu kompilieren, öffnen Sie bitte die Projektdatei "LED_NI.PRJ" (bzw. LED_NI.IDE). Beachten Sie bitte, dass die Datei "MAXRT.LIB" in Ihre Projektdatei eingebunden werden muss. Nähere Informationen zu dieser Bibliothek finden Sie im Kapitel 6 dieses Handbuchs, wo alle Routinen der Bibliothek "MAXRT" ausführlich erläutert sind.

Zuerst wird der Aufbau der PDT als Struktur deklariert. Das Programm enthält vier Prozeduren (Main-Proc, Auto-Init, Start und Stop).

Dann wird eine Struktur für Parameter vereinbart. Der Speicher für den Parameterbereich wird vom Programm bereitgestellt. Bei der Deklaration ist es wichtig, dass vor den eigentlichen Parametern Platz für die TDT reserviert wird.

Alle globalen (also in der PDT aufgeführten Prozeduren) beginnen mit **max_entry_proc** und enden mit **max_exit_proc**. Dadurch ist sichergestellt, dass alle Prozessor-Register gesichert und für die Prozedur richtig eingestellt werden.

In der Auto-Init-Prozedur werden die Parameter initialisiert, das Handle für die folgenden Funktionen geholt und die LED ausgeschaltet.

Die Prozedur **main** enthält in einem "normalen" C-Programm das eigentliche Hauptprogramm. Dieser Teil, der hier als "PREPARE" bezeichnet wird, wird während der Installierung abgearbeitet. Er sorgt dafür, dass die PDT initialisiert wird und dem MAX-PC Betriebssystem die Adresse der PDT mitgeteilt wird (**max_init_program**). Beachten Sie bitte die Einstellung der PDT-Flags.

Die Datei "DOS.H" sollte in alle Programme eingebunden werden. Dadurch werden einige Prozeduren wesentlich schneller ausgeführt.

5.11.2.2. Installation der NI-Task

Nachdem das Projekt "LED_NI.PRJ" einwandfrei übersetzt und die Datei "LED_NI.EXE" erzeugt wurde, kann das Programm auf den MAX-PC geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW32 ein, und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "LED_NI.INS" benutzen.

```
; Beispielinstallation: LED-Blinken als NI-Task (ohne Interrupt)
;
; Reset einer MAX-Trägerkarte
MAXRESET board=0
;
; MAX-PC anwählen und Slot-Layer-Nummer einstellen
MAXCONNECTCPU board=0 slot=1 layer=0
;
; Osx auf MAX-PC laden
MAXLOADOSX osx="ROM"
;
; Programmnummer 9000h unter Tasknummer 21h installieren
MAXINST file="LED_NI.EXE" no=9000 task=21 tasktype=MAX_NI_TASK  irq=0 level=0
        datasize=0 autoinit
;
; Parameter der Task 21h setzen
; PAR-4,5 : Blinkrate der LED    => 1000h
;
MAXPAR task=21 start=4 para=00 10
;
; Durch Aufruf der Start-Prozedur Programm starten (Proz. 2)
MAXPROC task=21 proc=2
```

Diese Installationsdatei installiert die NI-Task, stellt die Blinkrate ein und aktiviert die Task durch Aufruf der Prozedur 2.

5.11.2.3. II-Task in Borland C

Das folgende Listing zeigt das Blink-LED Programm für den MAX-PC als II-Task. Sie finden das Programm auch unter dem Namen "LED_II.C" auf der mitgelieferten CD. Zum Kompilieren benutzen Sie bitte die Projektdatei "LED_II.PRJ" (bzw. LED_II.IDE), die sich ebenfalls auf der CD befindet. Das Programm soll unter dem Interrupt von TIMER-A installiert werden. Die Blinkrate soll in Hertz angegeben werden.

```
#pragma option -r-

#define _RTLIB_                         // must be defined in every real time
                                        // program before including max_lib.h
#include "max_lib.h"
#include "x_error.h"
#include "dos.h"
#include "stdlib.h"

#define PROGRAMM_NUMBER 0x9001
#define VERSION         '1'
#define REVISION        'C'
#define SRQ_ERROR       0x8182
```

```
// PDT structure
struct pdt_type
{
   MAX_PDT_HEAD    rcHead;
   void* MAXEXPORT main_proc;
   void* MAXEXPORT auto_init;
   // any number of user defined procedure can follow (optionally)
   void* MAXEXPORT start;
   void* MAXEXPORT stop;
} rcPDT;

// parameter area structure
struct parameter_type
{
   MAX_TDT_TYPE rcTDT;                 // always before the first parameter !
// offset 0
   MAX_ERROR    Error;                 // Errorcode
   USHORT       usLedState;            // Actual state of the LED
   USHORT       usBlinkrate;           // Controls of the blinking frequency
   USHORT       usSendSRQs;            // Control if SRQ should be sent with every
                                       // change of the LED state
   USHORT       usTimerFrequency;      // Timer value
// offset 10
   USHORT       usSRQMsg;              // SRQ message to be used if usSendSRQs=1
   USHORT       usHostNo;              // number of the target for srqs
   MAXMODHND    hMyself;               // Handle for the CPU module
   MAXMODHND    hHost;                 // Handle for the PC (to send SRQs)
// offset 20
   MAX_OS_INFO  rcOsX;                 // Version info about OsX
} parameter;

USHORT usTimerCounter;                 // global counter variable

/*****************************************************************************/
// in case of error send srq to PC
// this is a local procedure - no max_entry / max_exit must be called
void ErrorHandler(void)
{
   parameter.Error = max_send_srq (parameter.hHost, 0, SRQ_ERROR);
}

/*****************************************************************************/
// main procedure of the task (called automatically by OsX after activating the task)
// the procedure is called with the frequency in 'parameter.usTimerFrequency'
void MAXEXPORT MainProc(void)
{
   max_entry_proc();

   // LED blinking can be extended with a software counter
   if (++usTimerCounter == parameter.usBlinkrate) {
      if (parameter.usLedState == 0) {
         max_led_on(parameter.hMyself);
         parameter.usLedState = 1;
      }
      else {
         max_led_off(parameter.hMyself);
         parameter.usLedState = 0;
      }
      usTimerCounter = 0;
      // transmit srq to the host pc if parameter usSendSRQs is set
      if (parameter.usSendSRQs)
         max_send_srq (parameter.hHost, 0, parameter.usSRQMsg);
   }
   max_exit_proc();
}
```

```
/*****************************************************************************/
// Init task parameter
void MAXEXPORT AutoInit(void)
{
   max_entry_proc();
   parameter.usTimerFrequency = 11892; // = 100Hz (1 unit = 0,84us)
   // LED blink frequency = usTimerFrequency / usBlinkrate
   parameter.usBlinkrate = 100;        // 1 Hz
   parameter.usHostNo    = 1;          // default connection to PC via PCI bus
   usTimerCounter        = 0;
   parameter.usSendSRQs  = 0;
   parameter.usSRQMsg    = 0x9190;
   parameter.Error = max_init_lib(0);
   if (parameter.Error == ERR_OK) {
      // get a handle for the X-MAX-1
      parameter.Error = max_connect_cpu (0, MAX_MYSELF, 0,
                                         &parameter.rcOsX, &parameter.hMyself);
      if (parameter.Error == ERR_OK) {
         max_led_off(parameter.hMyself);
         parameter.usLedState = 0;          // Led is switched off
      }
   }
   max_exit_proc();
};

/*****************************************************************************/
// start the task
void MAXEXPORT Start(void)
{
   USHORT      usTask;

   usTask = max_entry_proc();
   // get a handle for the host PC (to send SRQs)
   parameter.Error = max_connect_cpu (parameter.usHostNo, 0, 0, NULL, &parameter.hHost);
   if (parameter.Error == ERR_OK) {
      parameter.Error = max_set_timer (parameter.hMyself, MAX1_TIMER_C,
                                       MAX1_TIMER_MODE_INT, parameter.usTimerFrequency);
      parameter.Error = max_clear_int (parameter.hMyself, parameter.rcTDT.ucInterrupt);
      parameter.Error = max_unmask_int (parameter.hMyself, parameter.rcTDT.ucInterrupt);
      parameter.Error = max_wakeup_ii_task (parameter.hMyself, usTask);
   }
   max_exit_proc();
}

/*****************************************************************************/
// stop the task
void MAXEXPORT Stop(void)
{
   USHORT usTask;

   usTask = max_entry_proc();
   parameter.Error = max_sleep_task (parameter.hMyself, usTask);
   max_exit_proc();
}

/*****************************************************************************/
// initialize the Program Descriptor Table (PDT)
void main ()
{
   // the PDT-flags determine the task-type and
   // information how the data and parameter area are allocated
   rcPDT.rcHead.usFlags         = MAX_II_TASK +
                                  MAXPDT_PARAMETER_IN_PROG; // parameter are part of the
                                                            program

   // interrupt source (0 for NI-tasks)
   rcPDT.rcHead.usInterrupt     = MAX1_IRQ_TIMERC;

   // the program number identifies the program uniquely
   rcPDT.rcHead.usProgNo        = PROGRAMM_NUMBER;
```

```
   // number of global task procedures in the PDT
   rcPDT.rcHead.usProcedures    = (sizeof(rcPDT) - sizeof (MAX_PDT_HEAD)) / 4;

   // the following lines are just for documentation and information purpose
   rcPDT.rcHead.ucType          = 1;   // always !
   rcPDT.rcHead.ucSize          = sizeof(MAX_PDT_HEAD);
   rcPDT.rcHead.cVersion        = VERSION;
   rcPDT.rcHead.cRevision       = REVISION;
   rcPDT.rcHead.ucCPU           = MAX_CPU486;
   rcPDT.rcHead.ucCoProc        = _NOCOPROC;
   rcPDT.rcHead.ucLanguage      = MAXRT_LANGUAGE_C;
   rcPDT.rcHead.ucProgType      = 0;   // not used
   rcPDT.rcHead.ulRes1          = 0;
   rcPDT.rcHead.ulRes2          = 0;
   rcPDT.rcHead.ulHyperAdr      = 0;   // feature not available yet

   // address and size of the tasks data area
   // only when MAXPDT_DATA_IN_PROG is set in PDT-flags
   rcPDT.rcHead.ulDataAdr       = 0;
   rcPDT.rcHead.ulDataSize      = 0;

   // address and size of the tasks parameter area
   // only when MAXPDT_PARAMETER_IN_PROG is set in PDT-flags
   rcPDT.rcHead.ulParAdr  = max_get_physical_addr(&parameter) + sizeof(MAX_TDT_TYPE);
   rcPDT.rcHead.usParSize = sizeof(parameter) - sizeof(MAX_TDT_TYPE);

   // initialize global procedure addresses in PDT
   rcPDT.main_proc        = MainProc;
   rcPDT.auto_init        = AutoInit;
   rcPDT.start            = Start;
   rcPDT.stop             = Stop;

   // transfer PDT information to the OsX
   max_init_program ((MAX_PDT_HEAD*)&rcPDT);
}
```

Da das Listing ausführlich dokumentiert ist, soll nur auf die Besonderheiten eingegangen werden. Zum Kompilieren benutzen Sie bitte die Projektdatei "LED_II.PRJ" (bzw. LED_II.IDE).

In der PDT werden Sie erkennen, dass das Programm vier Prozeduren enthält (Main-, Auto-Init-, Start- und die Stop-Prozedur).

Die Start-Prozedur berechnet zuerst aus der angegebenen Frequenz die Timer-Werte. Beachten Sie bitte dabei, dass die Timer mit einer Eingangsfrequenz von 1,1892 MHz getaktet werden, und dass jeder Timer 16 Bit breit ist. Da mit diesen Werten "nur" Blinkfrequenzen zwischen 1,1892 MHz und 17 Hz möglich sind, wurde der Hardware-Timer durch einen Softwarezähler erweitert. Erst wenn dieser Softwarezähler den Wert des Parameter *usBlinkrate* erreicht hat, wird die LED umgeschaltet. Bei der Programmierung wurde darauf geachtet, dass der Timer-Wert möglichst groß ist, damit die Anzahl der Interrupts vom Timer pro Zeiteinheit möglichst klein ist.

Die Hauptprozedur unterscheidet sich nicht von der Hauptprozedur der NI-Task.

5.11.2.4. Installation der II-Task

Nachdem das Projekt "LED_II.PRJ" einwandfrei übersetzt und die Datei "LED_II.EXE" erzeugt wurde, kann das Programm auf den MAX-PC geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW32 ein, und laden Sie diese *.INS Datei. Sie können auch die Beispielinstallationsdatei "LED_II.INS" benutzen.

```
; Beispielinstallation: LED-Blinken mit Timer-Interrupt (Timer-A)
;
; Reset einer MAX-Trägerkarte
MAXRESET board=0
;
; MAX-PC anwählen und Slot-Layer-Nummer einstellen
MAXCONNECTCPU board=0 slot=1 layer=0
;
; Osx auf MAX-PC laden
MAXLOADOSX osx="ROM"
;
; Programmnummer 9001h unter Tasknummer 22h installieren
MAXINST file="LED_II.EXE" no=9001 task=22 tasktype=MAX_II_TASK irq=78 level=0
        datasize=0 autoinit
;
; Parameter der Task 22h setzen
; PAR-4,5 : Blinkrate der LED    => 1000h
;
MAXPAR task=22 start=4 para= 00 10
;
; Durch Aufruf der Start-Prozedur Programm starten (Proz. 2)
MAXPROC task=22 proc=2
```

Diese Installationsdatei installiert die II-Task und aktiviert sie anschließend.

Sie könnten als Übung versuchen, das vorhandene Programm unter einen anderen Namen zu kopieren und es so zu modifizieren, dass dieses neue Programm z.B. den TIMER-C benutzt.

Notizen

6. Bibliotheken

Die Bibliotheken stellen die Programmierschnittstelle zu den MAX-Modulen und den MAX-Trägerkarten dar.

Bibliotheken werden für die Betriebssysteme OsX (für C und Pascal), Windows (98, CE[1], NT, ME, 2000 und XP) (für C, Pascal und Basic) mitgeliefert. Welche Compiler im einzelnen unterstützt werden, kann der mitgelieferten CD entnommen werden.

Die Bibliotheken können zum einen auf einem Host-PC eingesetzt werden, um lokale MAX-Trägerkarten und MAX-Module anzusprechen. Zum anderen kann auch auf Karten, wie z.B. dem „X-KiT-3“ oder dem „MAX5dip“ und „MAX8dip“, die über eine Remote-Verbindung angekoppelt sind, zugegriffen werden. Eine Remote-Verbindung kann z.B. über eine serielle Schnittstelle oder über ein Netzwerk hergestellt werden. Die Remote-Ankopplung verwendet die DCOM-Schnittstelle von Microsoft und wird daher z.Zt. nur unter Windows unterstützt. Die Bibliotheken erlauben das Ansprechen beliebig vieler MAX-Trägerkarten, bzw. beliebig vieler CPU-Module.

Außerdem können die Bibliotheken zur Erstellung sogenannter Echtzeitprogramme verwendet werden. Das sind Programme, die auf einem MAX-PC Modul unter dem Echtzeit-Multi-Tasking-Betriebssystem OsX laufen. Näheres zum Aufbau und Download solcher Echtzeitprogramme ist in den Kapiteln 6.13 bis 6.16 zu finden. Ein Zugriff aus einem Echtzeitprogramm auf einen angeschlossenen Host-PC ist **zur Zeit** nur über sogenannte Service-Requests möglich. Dabei kann der MAX-PC ein Wort an den PC senden, welches dort empfangen wird (s. Kapitel 6.12). Für die Steuerung eines weiteren Datenaustausches ist in diesem Fall der PC zuständig.

Auf der Host-PC-Seite ist vor Verwendung der Bibliothek eine Konfiguration bzw. Treiberinstallation der verwendeten MAX-Trägerkarten vorzunehmen (s. Kap. 2.5). Dies geschieht unter Verwendung der „Plug and Play“-Eigenschaften der MAX-Trägerkarte mit den mitgelieferten Konfigurations- bzw. Installationsprogrammen für die jeweiligen Betriebssysteme. Jeder im PC steckenden MAX-Trägerkarte sowie jedem per Remote-Verbindung angeschlossenen MAX-PC wird bei der Installation eine eindeutige Kartennummer zugeordnet. Diese Nummer wird bei einigen Bibliotheksfunktionen als Parameter zur Adressierung der Trägerkarte verlangt.

[1] Zukünftiges Produkt

Zu Beginn eines Programms ist in jedem Fall die Funktion **max_init_lib** aufzurufen. Bei der Beendigung der Bibliotheksbenutzung ist entsprechend **max_exit_lib** aufzurufen. Zu besonderen Konventionen in Echtzeitprogrammen s. auch Kapitel 6.13.

Die Programmierung mit den Bibliotheken verfolgt einen verbindungsorientierten Ansatz: Bevor mit einem MAX-PC kommuniziert werden kann, muss zunächst eine Verbindung zu dem Modul aufgebaut werden. Das geschieht unter Angabe der Nummer der MAX-Trägerkarte und des Steckplatzes des darauf befindlichen MAX-PC durch die Funktion **max_connect_cpu**. Diese liefert ein Handle zurück. Es muss im folgenden nahezu allen Bibliotheksfunktionen übergeben werden, um festzulegen welcher MAX-PC jeweils angesprochen werden soll. Solange bis mit der Funktion **max_exit_cpu** die Verbindung getrennt wird, kann mit dem MAX-PC kommuniziert werden.

Weitere Hinweise, die nicht mehr ins Handbuch aufgenommen werden konnten, finden sich auf der CD in der Datei "**libusage.pdf**".

6.1. Betriebssysteme

In den folgenden Abschnitten wird auf einige zu beachtende Besonderheiten bei der Benutzung der Bibliothek unter verschiedenen Betriebssystemen hingewiesen.

6.1.1. OsX

Die für das Erstellen eigener Programme benötigten Dateien stehen in der Datei maxlib.zip. In den darunter liegenden Unterverzeichnissen RT und COMMON sind alle Dateien für die Programmierung eigener Anwendungen in den Programmiersprachen C und Pascal enthalten.

Im Anwenderprogramm bzw. in den Projekteinstellungen ist die Konstante *_RTLIB_* zu definieren.

Als Code-Modell muss **large** eingestellt sein. Alle Funktionen, die dem Betriebssystem über die PDT bekannt gemacht werden, sind als '**far pascal**' zu deklarieren. Zu Besonderheiten bei der Verwendung von Floating-Point Arithmetik s.u.

Die Bibliothek wurde mit den C- und Pascal Compilern von Borland getestet.

6.1.1.1. Programmiersprache C/C++

Für die Programmierung in C/C++ werden die Borland C++ Compiler (Version 3.1, 4.5 und 5.0) unterstützt.

Die Bibliotheks-Datei *MAXRT.LIB* aus dem entsprechenden Verzeichnis *RT\BC31* (für Borland C 3.1 ohne Gleitkomma-Unterstützung), *RT\BC31_EMU* (für Borland C 3.1 mit Gleitkomma-Emulation) bzw. *RT\BC45* (für Borland C 4.5 und 5.0) ist zum Projekt hinzuzulinken.

Zusätzlich stehen in den *RT\BCxx*-Verzeichnissen die für das OsX-Betriebssystem angepassten Startup-Code-Objektdateien. Damit tatsächlich diese Startup-Codes anstelle der im Borland C enthaltenen in das Projekt eingebunden werden, muss unter *Option/Directories/Library-Directories* die Pfadangabe *RT\BCxx* vor derjenigen zu den Original Borland Bibliotheken stehen.

Beispiel: Sie arbeiten in der folgenden Arbeitsumgebung:

Pfad zu den Bibliotheksroutinen von C++:
C:\BORLANDC\LIB

Pfad zu Ihren MAX-PC Start-Up-Codes:
C:\SORCUS\MAX\MAXLIB\RT\BC45

Der Eintrag in "Library Directories" müsste dann wie folgt aussehen:
C:\SORCUS\MAX\MAXLIB\RT\BC45;C:\BORLANDC\LIB;

C sucht jetzt bei dem Linkvorgang zuerst in Ihrem Entwicklungsverzeichnis nach den Start-Up-Codes und dann erst in dem Bibliotheksverzeichnis von C.

Falls beim Linken eines Echtzeitprogramms die falschen Start-Up-Codes eingebunden werden, meldet der Linker 'undefined symbol **_wrong_startups_linked**'.

Die Header-Datei *MAX_LIB.H* mit den Prototypen der Funktionen aus dem Verzeichnis *COMMON\C* ist in jedem Fall per #include einzubinden. Sie bindet automatisch die Header-Files *X_DEFS.H* und *X_TYPES.H* ein. Diese enthalten Konstantendefinitionen bzw. Definitionen von in den Bibliotheksfunktionen verwendeten Datentypen.

Bei Verwendung der Modul-Device-Treiber ist zusätzlich die Datei *X_MDD.H* per #include einzubinden.

Möchten Sie in Ihrem Programm die für Fehlermeldungen definierten Konstanten verwenden, ist außerdem die Datei *X_ERROR.H* per #include einzubinden.

Nach der Einbindung der neuen Start-Up-Codes können Sie Programme mit C für den MAX-PC erstellen. Beachten Sie bei der Programmierung die folgenden Hinweise:

!

- *Bildschirm- (Grafik oder Text) oder Tastaturfunktionen sind nur bei Verwendung eines speziellen Treiberprogramms erlaubt (siehe Anhang H).*
- *Es darf keine Overlay-Technik verwendet werden.*
- *Es können keine Datei-Operationen durchgeführt werden.*
- *Es kann kein Speicher dynamisch über die Standardfunktionen von C reserviert oder freigegeben werden. Hierfür werden entsprechende Routinen von den Bibliotheken bereitgestellt.* ***Pointer-Operationen können jedoch ganz normal durchgeführt werden.***
- *Es dürfen keine DOS-spezifischen Funktionen bzw. Funktionen, die DOS-Interrupts verwenden, benutzt werden (siehe folgende Tabelle).*
- *Das Einfügen von Debug-Informationen hat keinen Einfluss auf die Ausführung des Programms.*
- *Benutzen Sie das Speichermodell "LARGE".*
- *Bei Floating-Point-Operationen darf der Stack innerhalb der Prozedur nicht verändert worden sein. Durch* ***max_entry_proc*** *wird der Stack verändert! Falls Sie also Fließkommaoperationen z.B. in der Hauptprozedur vornehmen möchten, so schreiben Sie bitte eine separate Prozedur, die dann innerhalb der Hauptprozedur aufgerufen wird.*
- *Verwenden Sie keine Registervariablen in Taskfunktionen.*
- *Keine Floating-Point-Operationen in II-Tasks oder Prozeduren bzw. Funktionen, die vom PC aus aufgerufen werden.*

Folgende Standardfunktionen von Borland C können ohne Einschränkungen verwendet werden:

fabs	cos	log	sizeof	isalpha	memcpy
exp	tan	log10	strlen	isascii	memmove
asin	ceil	poly	strcpy	isdigit	atoi
acos	floor	pow	strcmp	islower	itoa
atan	fmod	pow10	strcat	isupper	
sin	hypot	sqrt			

Dies ist nur ein Auszug der verwendbaren Routinen. Im Prinzip können alle Routinen zur Speichermanipulation und auch Stringfunktionen uneingeschränkt verwendet werden.

Alle Routinen, die unter die nachfolgenden Rubriken fallen, können bei der Echtzeitprogrammierung nicht verwendet werden:

- *Verzeichnisroutinen*
- *I/O-Routinen**
- *Grafikroutinen**
- *Bildschirmroutinen**
- *Datum und Uhrzeit*
- *Dynamische Speicherverwaltung (z.B. Allokierung und Freigabe)*
- *Funktionen wie 'delay' und 'sound'*

* nur bei Verwendung separater Treiber werden diese Routinen unterstützt. Die Treiber sind im Anhang H beschrieben.

Sie sollten die Compiler- und Linker-Optionen wie folgt einstellen:

Borland C 3.1:

Options:

- Compiler / Advanced-Code-Generation: Emulation
  80386-Code
  Generate Underbars
- Compiler / Entry-, Exit-Code: DOS standard
  Standard stack frame
- Compiler / Optimize / Optimize for: Speed
- Compiler / Optimize / Register Variables: None
- Compiler / Source-Options: BORLAND C++
- Linker / Libraries: keine zusätzlichen Libraries einbinden

Borland C 4.5 und 5.0:

TargetExpert:

- Zieltyp: Anwendung (.exe)
- Umgebung: DOS (Standard)
- Zielmodell: Large
- Standardbibliotheken: Laufzeit
  'Emulation' oder 'keine Mathe-Unterstützung'

Projektoptionen:

- Compiler / Compiler-Ausgabe: Unterstriche erzeugen
- Compiler / Code-Generierung / Registervariablen: nicht verwenden
- 16-Bit Compiler / Prozessor: i486
- Optimierungen / Spezielle Optimierungen: hinsichtlich Geschwindigkeit (**nur bei Borland 4.5**)

6.1.1.2. Programmiersprache Pascal

Für die Programmierung in Pascal wird der Borland-Pascal 7.0 Compiler unterstützt.

Im Verzeichnis *RT\BP7* befinden sich alle benötigten Unit-Dateien (TPUs). Diese wurden aus den Quellcode-Dateien im Verzeichnis *COMMON\PAS* erzeugt. Sie müssen per uses-Anweisung in das Projekt eingebunden werden. Die Datei *MAXMDD.TPU* braucht nur bei Verwendung der Modul-Device-Treiber-Funktionen und die Datei *X_ERROR.TPU* nur bei Verwendung der für Fehlermeldungen definierten Konstanten eingebunden werden.

Bevor Sie mit der eigentlichen Programmierung beginnen können, sind einige Vorbereitungen notwendig. Zuerst sollten Sie sich ein Entwicklungsverzeichnis anlegen, in dem Sie Ihre Programme für den MAX-PC ablegen können. Danach muss die System-Unit von Borland-Pascal durch eine neue ersetzt werden, die Sie in der Datei maxlib.zip auf der SORCUS-CD im Verzeichnis *\RT\BP7* finden können.

Die System-Unit enthält neben den Laufzeitbibliotheken auch den Initialisierungsteil von Borland-Pascal. Darin sind viele DOS-Aufrufe enthalten, die in einer speziellen System-Unit durch Aufrufe des MAX-PC Betriebssystems (OsX) ersetzt sind.

Die System-Unit wird automatisch in alle Programme eingebunden, ohne dass sie mit USES angegeben werden muss. Normalerweise liegt sie auch nicht explizit als SYSTEM.TPU vor, sondern ist mit den anderen Standard-Units von Borland-Pascal (z.B. PRINTER) in der Datei TURBO.TPL (TPL = Turbo-Pascal-Library) zusammengefasst, wodurch sich die Ladezeiten bei der Kompilierung deutlich verkürzen.

Die Original-SYSTEM.TPU muss zuerst aus TURBO.TPL entfernt werden. Der Inhalt von TURBO.TPL wird mit dem Programm TPUMOVER verändert. Die genaue Beschreibung dieses Programms finden Sie im Borland-Pascal Benutzerhandbuch im Kapitel "Die Zusatzprogramme".

Wechseln Sie bitte in Ihr Borland-Pascal Verzeichnis und geben folgende Zeilen ein:

```
tpumover turbo *system
tpumover turbo -system
```

Mit der Ausführung der ersten Zeile wird die Datei Original-SYSTEM.TPU extrahiert. Die zweite Zeile entfernt die System-Unit aus TURBO.TPL.

Nach wie vor müssen Sie die System-Unit nicht in der USES-Anweisung angeben, allerdings müssen Sie dem Compiler nun mitteilen, wo er die Datei SYSTEM.TPU findet. Das geschieht, wie für andere Units auch, unter *'Option/Verzeichnisse/Unit-Verzeichnisse'*. Um Verwechslungen auszuschließen, geben Sie den Pfad, unter dem

die SYSTEM.TPU zu finden ist, als ersten an. Für Echtzeitprogramme ist das normalerweise *C:\SORCUS\MAX\MAXLIB\RT\BP7*.

Der Eintrag in die Verzeichnisliste unter 'Option' kann entfallen, wenn die SYSTEM.TPU in das Verzeichnis kopiert wird, in dem Sie Ihre Echtzeitprogramme entwickeln. Das aktuelle Verzeichnis wird immer vor den in der Liste angegebenen Verzeichnissen nach Units durchsucht.

Falls Sie nicht die mitgelieferte SYSTEM.TPU verwenden, erhalten Sie beim Versuch, ein Echtzeitprogramm zu erstellen, die Meldung 'External Bezeichner **_wrong_startups_linked** nicht gefunden'. Im umgekehrten Fall, also beim Compilieren eines PC-Programms mit der SORCUS-System-Unit, ist die Fehlermeldung leider nicht so festgelegt. In der Regel wird die Meldung 'UNIT-Versionen stimmen nicht überein (*Name*)' lauten, wobei *Name* für eine beliebige, von Ihnen verwendete Unit stehen kann.

! *Nach der Einbindung der neuen SYSTEM.TPU können Sie Echtzeitprogramme mit Borland-Pascal für den MAX-PC erstellen. Beachten Sie dabei die folgenden Einschränkungen und Hinweise:*

- *Es sind z.Zt. keine Bildschirm- (Grafik und Text) oder Tastaturfunktionen möglich.*
- *Es darf keine Overlay-Technik verwendet werden.*
- *Es können keine Datei-Operationen durchgeführt werden.*
- *Es kann kein Speicher dynamisch reserviert bzw. freigegeben werden ("GETMEM", "NEW", "DISPOSE" etc.). Dafür stehen spezielle Bibliotheksroutinen zur Verfügung.* ***Pointer-Operationen können normal verwendet werden.***
- *Es darf kein Range-Check aktiviert sein. Eine Zusammenstellung der Compilerschalter folgt auf der nächsten Seite.*
- *Es dürfen keine DOS-spezifischen Funktionen benutzt werden (DOS-Interrupts)*
- *Das Einfügen von Debug-Informationen hat keinen Einfluss auf die Ausführung des Programms.*
- *Es dürfen keine Floating-Point-Operationen in II-Tasks, oder in Prozeduren bzw. Funktionen, die vom PC aus aufgerufen werden, vorgenommen werden.*

Folgende Standardfunktionen von Borland-Pascal können ohne Einschränkungen verwendet werden:

Abs	Delete	Int	Pred	SSeg
Addr	DSeg	Length	Ptr	Str
ArcTan	Exp	Ln	Round	Succ
Chr	Exit	Lo	Seg	Swap
Concat	FillChar	Move	Sin	Trunc
Copy	Frac	Odd	SizeOf	Upcase
Cos	Hi	Ofs	SPtr	Val
CSeg	Inc	Ord	Sqr	
Dec	Insert	Pos	Sqrt	

Weiterhin kann selbstverständlich wie bisher das Unit-Konzept benutzt werden, um beispielsweise eigene Bibliotheken zu generieren.

Sie sollten die Compiler-Optionen für Pascal wie folgt einstellen:

{$F+}	Prozeduren mit FAR-CALLS aufgerufen
{$R-}	Range-Check ausschalten
{$S-}	Stack-Check ausschalten
{$I-}	I/O-Check ausschalten
{$M 1024,0,0}	Stack und Heap auf ein Minimum einstellen.

Wenn Fließkommazahlen (nicht vom Typ REAL) benutzt werden:

{$N+}	Befehle für den mathematischen Co-Prozessor erzeugen
{$E+}	Co-Prozessor-Emulation einbinden.

Wenn keine Fließkommazahlen oder nur solche vom Typ REAL benutzt werden:

{$N-}	Keine Befehle für den mathematischen Co-Prozessor erzeugen
{$E-}	Keine Co-Prozessor-Emulation einbinden.

Das Einstellen der Compilerschalter können Sie entweder in der Entwicklungsumgebung unter 'Options' oder direkt im Quelltext mit Hilfe von Compiler-Switches vornehmen. Empfohlen wird die Einstellung im Quelltext.

6.1.2. Windows NT, 98, 2000, XP und ME

Die für das Erstellen eigener Programme benötigten Dateien stehen in der Datei maxlib.zip. In den darunter liegenden Unterverzeichnissen PC und COMMON sind alle Dateien für die Programmierung eigener Anwendungen in den Programmiersprachen C, Delphi und Visual Basic enthalten.

Für Windows steht die DLL **MAXW32.DLL** zur Verfügung. Sie setzt auf dem Gerätetreiber **MAXDRV.SYS** (unter NT) bzw. **MAXWDM.SYS** auf. Zur Laufzeit eines Programms muss der Gerätetreiber installiert sein und auf die DLL muss zugegriffen werden können (s. Kapitel 2.).

Die Aufrufkonvention der DLL-Funktionen ist 'stdcall'.

Die Bibliothek wurde mit den C-Compilern von Microsoft und Borland, mit Borland Delphi und Microsoft Visual Basic getestet.

6.1.2.1. Programmiersprache C/C++

Für die Programmierung in C/C++ werden die Compiler Microsoft Visual C (ab Version 4.0) und Borland C++ (ab Version 5.0) unterstützt.

Für diese Compiler werden Importbibliotheken und Header-Dateien als Schnittstelle zur Bibliothek *MAXW32.DLL* (wird bei der Treiberinstallation automatisch in das Windows/System32 Verzeichnis kopiert) mitgeliefert.

Die Importbibliothek-Datei *MAXW32.LIB* aus dem Verzeichnis *PC\Win32\BC5* (für den Borland C++ Compiler), *PC\WIN32\BCB* (für Borland C++ Builder) bzw. *PC\Win32\MSVC* (für den Microsoft Compiler) ist zum Projekt hinzuzulinken.

Die Header-Datei *MAX_LIB.H* mit den Prototypen der Bibliotheksfunktionen aus dem Verzeichnis *COMMON\C* ist in jedem Fall per #include einzubinden. Diese bindet automatisch die Header-Dateien *X_DEFS.H* und *X_TYPES.H* ein. Diese enthalten Konstantendefinitionen bzw. Definitionen von in den Bibliotheksfunktionen verwendeten Datentypen.

Bei Verwendung der Modul-Device-Treiber ist zusätzlich die Datei *X_MDD.H* per #include einzubinden.

Möchten Sie in Ihrem Programm die für Fehlermeldungen definierten Konstanten verwenden, ist zusätzlich die Datei *X_ERROR.H* einzubinden.

Stellt das Programm Callback-Funktionen für Service-Request (max_set_service) oder Module-Device-Treiber Kanäle (max_open_channel) bereit, so muss die Multi-Thread Version der Laufzeitbibliothek verwendet werden.

6.1.2.2. Programmiersprache Delphi

Für die Programmierung in Borland-Delphi werden die Compiler ab Version 2.0 unterstützt.

Es werden keine compilierten DCU-Dateien mitgeliefert. Statt dessen müssen die .PAS-Sourcefiles aus dem Verzeichnis *COMMON\PAS* per uses-Anweisung in das Projekt eingebunden werden.

Die Dateien *MAXLIB.PAS*, *MAXDEFS.PAS* und *MAXTYPE.PAS* müssen in jedem Fall eingebunden werden, die Datei *MAXMDD.PAS* nur bei Verwendung der Modul-Device-Treiber-Funktionen und die Datei *X_ERROR.PAS* nur bei Verwendung der für Fehlermeldungen definierten Konstanten.

6.1.2.3. Programmiersprache Visual Basic

Für das Ansprechen der X-Bus-Bibliothek *MAXW32.DLL* aus Visual Basic Programmen stehen folgende Importbibliotheken zur Verfügung:

- *max_lib.bas* enthält die Deklaration von Funktionen und Typen,
- *max_defs.bas* enthält die verwendeten Konstanten
- *max_mdd.bas* enthält die Konstanten und CPS-Typ-Deklarationen für die Verwendung der Modul-Device-Treiber
- *max_err.bas* enthält die definierten Fehler-Konstanten

Diese Dateien müssen als *Module* einem Visual Basic Projekt hinzugefügt werden.
Einige Datentypen wie z.B. *MAX_ERROR*, *MAX_VERSION* sowie alle MAX-Handle-Typen wurden in Visual Basic nicht eigens definiert. Dafür werden die Standard Typen *INTEGER* bzw. *LONG* verwendet. Im einzelnen geht das aus den mitgelieferten Basic-Bibliotheken hervor.

Bei Konstanten, die in den anderen Sprachen und im Handbuch mit einem Unterstrich beginnen, entfällt der Unterstrich in der Visual Basic Definition, da dieser nicht zulässig ist.

Einige in der DLL verwendete Datentypen sind auf Word-Alignment angewiesen. Wenn eine Variable eines solchen Typs an eine DLL-Funktion übergeben wird, erwartet die DLL alle Elemente der Struktur im Speicher direkt hintereinander stehend. Visual Basic richtet dagegen die Struktur-Elemente an 4-Byte-Adressen aus. Wenn z.B. ein Integer und ein Long Element hintereinander in einer Datenstruktur stehen, fügt Visual Basic zwischen beiden Elementen einen Integer Dummy-Wert ein.

Aus diesem Grund sind die Datenstrukturen *MAX_OS_INFO*, *MAX_TI_TYPE*, *MAX_BOARD_TYPE*, *MAX_CHANNEL_TRANSFER_TYPE* und *MAX_FLASH_-TYPE* sowie einige CPS-Strukturen in Basic anders definiert als im Handbuch. Der Typ Long wurde in diesen Strukturen durch MAX_LONG ersetzt. Dieser Typ ist nur in Visual Basic definiert. Zur Umwandlung von MAX_LONG in Long ist im Modul Max_lib.bas folgende Funktion enthalten:

```
Public Function ConvertMaxLongToLong (ByRef prcMaxLong As MAX_LONG) As Long
```

Zur Umwandlung von Long nach MAX_LONG steht die folgende Subroutine zur Verfügung:

```
Public Sub ConvertLongToMaxLong (ByVal ulLongValue As Long,
                                 ByRef prcMaxLong As MAX_LONG)
```

Die Funktion *max_set_service* ist unter Visual Basic nicht implementiert, da der Aufruf von Callback-Funktionen unter den derzeitigen Visual Basic Versionen (inkl. 6.0) nicht funktioniert, sofern der Aufruf aus einem anderen Thread heraus erfolgt. Dieses ist im Fall der Callback-Funktion für Service-Requests der Fall. Aus dem selben Grund muss der Parameter *pCbFunc* beim Aufruf der Funktion *max_open_channel* in Visual Basic-Programmen =NULL gesetzt werden.

Die Visual Basic Versionen 4 bis 6 werden unterstützt.

6.1.2.4. Verwendung einer MAX6pci ohne aufgestecktes CPU-Modul

Die Programmierung einer Karte ohne aufgestecktes X-MAX-1 Modul ist prinzipiell identisch zum vorherigen. Dadurch, dass viele der im Handbuch beschriebenen Funktionen sich direkt auf ein X-MAX-1 bzw. das darauf laufende OsX-Betriebssystem beziehen, steht aber natürlich nur ein eingeschränkter Funktionsumfang zur Verfügung. Er umfasst im wesentlichen die Modul-Device-Treiber-Zugriffsfunktionen für das Ansprechen der aufsteckenden I/O-Module.

Folgende Bibliotheksfunktionen stehen zur Verfügung:

Initialisierung:

- max_init_lib und max_exit_lib
- max_reset_board
- max_board_reacting
- max_set_timeout
- max_set_board_led
- max_connect_cpu (dabei ist slot=layer=0 zu setzen)
- max_exit_cpu

Version und andere Informationen abfragen

- max_get_drv_version, max_get_lib_version, max_get_version_string
- max_get_board_info, max_get_module_info und max_scan_boards

Zugriff auf Modul-EEPROMs

- alle Funktionen aus Kapitel 6.4.1

Modul-Device-Treiber-Zugriffe

- alle Funktionen aus Kapitel 6.9

Fehlerbehandlung

- alle Funktionen aus Kapitel 6.19

6.2. Wichtige Hinweise zu den Bibliotheksfunktionen

Alle Funktionen der Bibliothek liefern als Rückgabewert einen Fehlerstatus vom Typ MAX_ERROR. Wichtige Informationen zur Fehlerbehandlung finden sich in Kapitel 6.19. Die Bedeutung der Fehlerrückgabewerte ist im Anhang B aufgeführt.

Hinweise:

1. Alle Strings werden von den Bibliotheken nullterminiert erwartet bzw. zurückgeliefert.
2. Die Länge der Daten, die an Bibliotheksfunktionen übergeben bzw. von Bibliotheksfunktionen zurückgeliefert werden, ist bei einigen Funktionen begrenzt (z.B. **max_call_func**, **max_read_channel_block** usw.). Die maximale Länge kann mit der Funktion **max_get_max_blocksize** abgefragt werden.
3. In den Bibliotheken verwendete Datenstrukturen verwenden zum Teil 8-Bit und 16-Bit Werte, daher dürfen struct bzw. record Felder nicht ausgerichtet sein.

Der Punkt "Verwendung" in den Beschreibungen der Bibliotheksfunktionen listet die Einsatzmöglichkeiten der jeweiligen Funktion auf. Kann eine Funktion z.B. auf einem MAX-PC in einem Echtzeitprogramm und auf dem Windows Host-PC ausgeführt werden, steht unter "Verwendung": "Host-PC, MAX-PC OsX".

Manche Funktionen greifen auch auf ein anderes Zielsystem zu, indem sie ein Handle auf das CPU-Modul erwarten. Bei diesen Funktionen wird sowohl das System angegeben, auf dem die Funktion ausgeführt wird, als auch das System, auf das zugegriffen werden soll. Kann z.B. vom Windows Host-PC auf ein OsX MAX-PC und von einem OsX MAX-PC auf einen anderen OsX MAX-PC zugegriffen werden, steht unter "Verwendung": "Host-PC → MAX-PC OsX, MAX-PC OsX → MAX-PC OsX".

6.3. Allgemeine Verwaltungsfunktionen

max_init_lib — Initialisierung der Bibliothek

Pascal FUNCTION max_init_lib (language: WORD): MAX_ERROR;

C MAX_ERROR max_init_lib (USHORT language);

VB Declare Function max_init_lib Lib "maxw32.dll" (ByVal language As Integer) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Diese Funktion initialisiert die Bibliothek. Es wird je nach Betriebssystem geprüft, ob der Treiber geladen werden kann. Die Funktion muss in jedem Thread (PC), der die Bibliothek nutzt, bzw. Echtzeitprogramm (auf dem MAX-PC) einmal aufgerufen werden. Der Aufruf muss dabei vor allen anderen Aufrufen erfolgen.

Parameter *language* Der Parameter legt fest, in welcher Sprache die Funktion **max_get_error_message** (s. Kapitel 6.19) von der Bibliothek erzeugte Meldungen zurückgibt. Zur Zeit ist nur der Wert 0 (Englisch) definiert. In Echtzeitprogrammen hat der Parameter keine Bedeutung.

max_exit_lib — Beenden der Bibliotheks-Benutzung

Pascal FUNCTION max_exit_lib: MAX_ERROR;

C MAX_ERROR max_exit_lib (void);

VB Declare Function max_exit_lib Lib "maxw32.dll" () As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Diese Funktion muss aufgerufen werden, wenn keine weiteren Bibliotheksfunktionen mehr benötigt werden. Sie gibt von der Bibliothek verwendete Ressourcen wieder frei. Alle zuvor geöffneten Handles werden ungültig.

Parameter keine

max_get_lib_version max_get_drv_version — Bibliotheks- und Treiberversion

Pascal

PROCEDURE max_get_lib_version (VAR version: MAX_VERSION; VAR date: LONGWORD);

FUNCTION max_get_drv_version (VAR version: MAX_VERSION; VAR date: LONGWORD): MAX_ERROR;

C

void max_get_lib_version (MAX_VERSION *version, ULONG *date)

MAX_ERROR max_get_drv_version (MAX_VERSION *version, ULONG *date)

VB

Declare Sub max_get_lib_version Lib "maxw32.dll" (ByRef version As Long, ByRef date As Long)

Declare Function max_get_drv_version Lib "maxw32.dll" (ByRef version As Long, ByRef date As Long) As Integer

Verwendung: Host-PC, MAX-PC OsX

Funktion: Diese Funktionen liefern die Version und das Erstellungsdatum der Bibliothek bzw. des von der Bibliothek genutzten Treibers.

Parameter

version — Zeiger auf eine Variable, in die die Version in folgendem Format eingetragen wird:
Byte 0-1: Revisionsnummer
Byte 2: Revisionsbuchstabe (z.B. ‚A‘)
Byte 3: Versionsnummer

date — Zeiger auf eine Variable, in die das Erstellungsdatum in folgendem Format eingetragen wird:
Byte 0: Tag (1..31)
Byte 1: Monat (1..12)
Byte 2-3: Jahr (0.. 65535)

max_get_version_string Version in String konvertieren

Pascal PROCEDURE max_get_version_string (versioncode: MAX_VERSION; datecode: LONGWORD; VAR version, date: PChar);

C void max_get_version_string (MAX_VERSION versioncode, ULONG datecode, char *version, char *date)

VB Declare Sub max_get_version_string Lib "maxw32.dll" (ByVal versioncode As Long, ByVal datecode As Long, ByVal version As String, ByVal date As String)

Verwendung Host-PC, MAX-PC OsX

Funktion Diese Funktion konvertiert die von **max_get_lib_version, max_get_drv_version, max_get_prog_version, max_get_mdd-_version** und **max_get_os_version,** gelieferten Versionsangaben (Version und Erstellungsdatum) in nullterminierte ASCII-Strings.

Parameter

versioncode Von einer der oben genannten Versions-Funktionen zurückgelieferter Versionscode

datecode Von einer der oben genannten Versions-Funktionen zurückgelieferter Datumscode

version Zeiger auf eine Variable, in die die Versionsangabe als nullterminierter String eingetragen wird. Es ist Platz für 9 Bytes zur Verfügung zu stellen.
Format: „01.A.006“ (Version 1, Revision A, Revisions-zähler 006)

date Zeiger auf eine Variable, in die das Erstellungsdatum als nullterminierter String eingetragen wird. Es ist Platz für 11 Bytes zur Verfügung zu stellen.
Format: „tt.mm.yyyy“ (Tag, Monat, Jahr)

max_scan_boards Abfrage der installierten X-Bus-Boards

Pascal FUNCTION max_scan_boards (VAR totalnumber: WORD; boardlist: POINTER): MAX_ERROR;

C MAX_ERROR max_scan_boards (USHORT *totalnumber, MAX_CONNECTION_TYPE *boardlist);

VB Declare Function max_scan_boards Lib "maxw32.dll" (ByRef totalnumber As Integer, ByRef boardlist As MAX_CONNECTION_TYPE) As Integer

Verwendung Host-PC

Funktion Mit dieser Funktion können alle mit dem Setup-Programm konfigurierten X-Bus-Karten abgefragt werden. Sie liefert neben den jeweiligen Board-Nummern auch Information über die Art der Verbindung zu den Boards.

Parameter *totalnumber*: Zeiger auf eine Variable, die beim Aufruf der Funktion die Größe des Arrays *boardlist* enthalten muss, in das die Funktion die Daten eintragen soll. Die Funktion trägt in *totalnumber* anschließend ein, wie viele Einträge in das Array tatsächlich vorgenommen wurden. Zur Zeit können max. 100 Karten verwaltet werden.

boardlist: Zeiger auf ein Array, in das die Funktion die installierten Boards einträgt. Es werden maximal *totalnumber* Einträge vorgenommen (auch wenn mehr Boards installiert sind!). Der Datentyp MAX_CONNECTION_TYPE ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
usBoardNumber	USHORT	Nummer des installierten Boards
usConnectionType	USHORT	Verbindungsart zu dem Board (s.u:)

Zur Zeit sind folgende Verbindungstypen definiert:

MAX_LOCAL_CONNECTION (=PCI- oder ISA-Board)

MAX_SERIAL_CONNECTION (= Ankopplung eines X-MAX-1 über dessen serielle Schnittstelle)

MAX_SERVER_CONNECTION (=Ankopplung eines Boards über einen Netzwerk-Server)

MAX_TCPIP_CONNECTION (=Ankopplung über eine Ethernet-Schnittstelle)

MAX_UNKNOWN_CONNECTION (unbekannte Verbindung).

6.4. Funktionen zum Ansprechen der MAX-Trägerkarte und der Module

Die Funktionen in diesem Abschnitt beziehen sich auf die Devices der MAX-Trägerkarte, z.B. um Informationen über die Trägerkarte oder einzelne darauf steckende Module abzufragen.

max_get_board_info Information über die Trägerkarte

Pascal	FUNCTION max_get_board_info (card: WORD; VAR info: MAX_BOARD_TYPE): MAX_ERROR;
C	MAX_ERROR max_get_board_info (USHORT card, MAX_BOARD_TYPE *info);
VB	Declare Function max_get_board_info Lib "maxw32.dll" (ByVal card As Integer, ByRef info As MAX_BOARD_TYPE) As Integer
Verwendung	Host-PC
Funktion	Diese Funktion liefert Informationen über die angegebene Trägerkarte.
Parameter	*card* Nummer der Trägerkarte

info Zeiger auf eine Struktur vom Typ MAX_BOARD_TYPE, in die Informationen über die Trägerkarte eingetragen werden. Die Struktur MAX_BOARD_TYPE ist wie folgt aufgebaut.

Element	Datentyp	Beschreibung
usType	USHORT	Typ der Trägerkarte definiert sind: CARDTYPE_MAX6PCI CARDTYPE_XBUS_CPU CARDTYPE_XKIT3 CARDTYPE_MAX5DIP CARDTYPE_MAX8DIP
usReserved	USHORT	Reserviert
usVersion	USHORT	Version der Trägerkarte: Im High-Byte steht ein Buchstabe (′A′ .. ′Z′) für die Revision, im Low-Byte ist der Fertigungsstand enthalten
ulSerNr	ULONG	Seriennummer

max_board_reacting Test der Trägerkarte

Pascal FUNCTION max_board_reacting (card: WORD): MAX_ERROR;

C MAX_ERROR max_board_reacting (USHORT card);

VB Declare Function max_board_reacting Lib "maxw32.dll" (ByVal card As Integer) As Integer

Verwendung Host-PC

Funktion Diese Funktion prüft, ob die angegebene Trägerkarte vorhanden und ansprechbar ist. Wenn diese Funktion den Fehler ERR_BOARD_NOT_REACTING liefert, ist das Zurücksetzen der MAX-Trägerkarte erforderlich.

Parameter *card* Nummer der Trägerkarte.

max_reset_board — Reset der Trägerkarte

Pascal FUNCTION max_reset_board (card: WORD; timeout: WORD): MAX_ERROR;

C MAX_ERROR max_reset_board (USHORT card, USHORT timeout);

VB Declare Function max_reset_board Lib "maxw32.dll" (ByVal card As Integer, ByVal timeout As Integer) As Integer

Verwendung Host-PC

Funktion Diese Funktion setzt die Trägerkarte und damit den X-Bus mitsamt allen aufgesteckten Modulen zurück. Bei Remote-Verbindungen zu MAX-PCs arbeitet diese Funktion nur dann korrekt, wenn die Ziel-CPU, über die die Verbindung hergestellt wurde, angesprochen werden kann. Sonst muss ein Hardware-Reset des Zielsystems durchgeführt werden.

Parameter *card* Nummer der Trägerkarte.

timeout Zeit (als Vielfaches zu 100ms), die bei allen Zugriffen auf die Trägerkarte oder ein MAX-Modul maximal auf eine Reaktion gewartet wird.

max_set_timeout — Einstellen der Timeoutzeit

Pascal FUNCTION max_set_timeout (card: WORD; timeout: WORD): MAX_ERROR;

C MAX_ERROR max_set_timeout (USHORT card, USHORT timeout);

VB Declare Function max_set_timeout Lib "maxw32.dll" (ByVal card As Integer, ByVal timeout As Integer) As Integer

Verwendung Host-PC

Funktion Mit dieser Funktion kann die Timeoutzeit neu eingestellt werden, die bei allen Zugriffen auf die Trägerkarte oder ein darauf steckendes MAX-Modul maximal auf eine Reaktion gewartet wird

Parameter *card* Nummer der Trägerkarte

timeout Zeit (als Vielfaches von 100 ms)

max_get_module_info Information über ein MAX-Modul

Pascal FUNCTION max_get_module_info (card: WORD; slot: WORD; layer: WORD; VAR modulinfo: MAX_MODULINFO_TYPE): MAX_ERROR;

C MAX_ERROR max_get_module_info (USHORT card, USHORT slot, USHORT layer, MAX_MODULINFO_TYPE *modulinfo);

VB Declare Function max_get_module_info Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByRef modulInfo As MAX_MODULINFO_TYPE) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Diese Funktion liefert Informationen über ein auf der Trägerkarte aufgestecktes MAX-Modul. Befindet sich auf dem angegebenen Steckplatz kein Modul, liefert die Funktion den Fehler ERR_MODULE_NOT_AVAILABLE zurück.

Parameter

card Nummer der Trägerkarte, in Echtzeitprogrammen ohne Bedeutung (=0 setzen).

slot Steckplatz des Moduls (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0, die Etage darüber die Nummer 1 usw.

modulinfo Zeiger auf eine Datenstruktur vom Typ MAX_MODULINFO_TYPE, in die Informationen über ein auf dem Steckplatz befindliches Modul eingetragen werden. Die Struktur MAX_MODULINFO_TYPE ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
usType	USHORT	Typ des Moduls (die definierten Typen finden sich im Anhang A)
usVariant	USHORT	Enthält genauere Information über Bestückungsvariante.
usReserved	USHORT	Reserviert

Element	Datentyp	Beschreibung
usVersion	USHORT	Version der Modul-Hardware: Im High-Byte steht ein Buchstabe (′A′ .. ′Z′) für die Revision, im Low-Byte ist der Fertigungsstand enthalten
usCategory	USHORT	Enthält genauere Information über die Kategorie des Moduls, z.B. CPU, Prozess-I/O usw.
usStatus	USHORT	Status des Moduls (zur Zeit ist nur Bit 0 definiert: Bit-0=1: Modul ist potentieller Bus-Master (d.h. ein CPU-Modul))

max_set_board_led Schalte LED auf Trägerkarte

Pascal FUNCTION max_set_board_led (card: WORD; led: WORD; onoff: WORD): MAX_ERROR;

C MAX_ERROR max_set_board_led (USHORT card, USHORT led, USHORT onoff);

VB Declare Function max_set_board_led Lib "maxw32.dll" (ByVal card As Integer, ByVal led As Integer, ByVal onoff As Integer) As Integer

Verwendung Host-PC

Funktion Diese Funktion schaltet die LED auf der MAX6pci-Trägerkarte ein oder aus (nicht verfügbar für X-KiT-3, MAX5dip und MAX8dip).

Parameter *card* Nummer der Trägerkarte.

led reserviert (= 0 setzen)

onoff Zustand der LED (0=aus, 1=ein)

6.4.1. Zugriff auf die Modul-EEPROMs

Jedes MAX-Modul verfügt über ein EEPROM, in dem modulspezifische Daten gespeichert werden. Neben einigen Bereichen, die vom Betriebssystem oder den Modul-Device-Treibern genutzt werden, steht ein großer Bereich für Anwenderdaten zur Verfügung. Dieser ist jedoch nicht werksseitig angelegt, sondern muss zunächst mit **max_create_eeprom_area** angelegt werden.

Bevor mit den folgenden Funktionen Zugriffe auf ein EEPROM erfolgen, muss zunächst mit **max_open_eeprom** der Zugriff initialisiert werden.

max_get_eeprom_info — Information über EEPROM

Pascal: FUNCTION max_get_eeprom_info (card: WORD; slot: WORD; layer: WORD; VAR info: MAX_EEPINFO_TYPE): MAX_ERROR;

C: MAX_ERROR max_get_eeprom_info (USHORT card, USHORT slot, USHORT layer, MAX_EEPINFO_TYPE *info);

VB: Declare Function max_get_eeprom_info Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByRef info As MAX_EEPINFO_TYPE) As Integer

Verwendung: Host-PC, MAX-PC OsX

Funktion: Mit dieser Funktion können Angaben über das EEPROM abgefragt werden.

Parameter:

- *card* — Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).
- *slot* — Steckplatz des Moduls, auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)
- *layer* — Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.
- *info* — Zeiger auf eine Datenstruktur vom Typ MAX_EEPINFO_TYPE, in die die Informationen eingetragen werden. Die Struktur MAX_EEPINFO_TYPE hat folgenden Aufbau:

Element	Datentyp	Bedeutung
ulVersion	ULONG	Version der EEPROM-Inhaltsdefinition
ulLength	ULONG	Größe des EEPROMs in Bytes
ulOrg	ULONG	Organisation des EEPROMs (0=Byte, 1=Wort, 2=Doppelwort). Dieser Eintrag entscheidet darüber, wie bei den übrigen EEPROM-Funktionen die Parameter *size* und *offset* interpretiert werden.
ulBufferSize	ULONG	Größe des EEPROM-Schreibpuffers
ulFlags	ULONG	z.Zt. ist nur MAX_EEP_AUTO_INC definiert, welches bedeutet, dass das EEPROM mit Auto-Inkrement der Adressen arbeitet
ulSpeed	ULONG	max. Geschwindigkeit, mit der auf das serielle EEPROM zugegriffen wird

max_open_eeprom — Öffnen des EEPROMs

Pascal: FUNCTION max_open_eeprom (card: WORD; slot: WORD; layer: WORD; flags: WORD): MAX_ERROR;

C: MAX_ERROR max_open_eeprom (USHORT card, USHORT slot, USHORT layer, USHORT flags);

VB: Declare Function max_open_eeprom Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByVal flags As Integer) As Integer

Verwendung: Host-PC, MAX-PC OsX

Funktion: Mit dieser Funktion wird ein EEPROM-Zugriff initialisiert.

Parameter: *card* Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).

slot	Steckplatz des Moduls, auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)
layer	Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.
flags	Eigenschaften für den EEPROM-Zugriff. Zur Zeit ist nur das Flag MAX_CHECK_EEP_CHECKSUM definiert, dass eine Überprüfung des gesamten EEPROMs mit der ebenfalls gespeicherten Checksumme bewirkt. Die Checksumme wird nicht bei jedem Zugriff, sondern nur in den Funktionen **max_open_eeprom** und **max_close_eeprom** geprüft bzw. berechnet.

max_close_eeprom Schließen des EEPROMs

Pascal	FUNCTION max_close_eeprom (card: WORD; slot: WORD; layer: WORD; flags: WORD): MAX_ERROR;	
C	MAX_ERROR max_close_eeprom (USHORT card, USHORT slot, USHORT layer, USHORT flags);	
VB	Declare Function max_close_eeprom Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByVal flags As Integer) As Integer	
Verwendung	Host-PC, MAX-PC OsX	
Funktion	Mit dieser Funktion wird ein EEPROM-Zugriff beendet.	
Parameter	*card*	Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).
	slot	Steckplatz des Moduls, auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)
	layer	Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.
	flags	Eigenschaften für den EEPROM-Zugriff. Zur Zeit ist nur das Flag MAX_CHECK_EEP_CHECKSUM definiert, dass die Berechnung der Checksumme über das gesamte EEPROM bewirkt, die dann abgespeichert wird.

max_get_free_eeprom_block Größter freier Block im EEPROM

Pascal FUNCTION max_get_free_eeprom_block (card: WORD; slot: WORD; layer: WORD; VAR size: LONGWORD): MAX_ERROR;

C MAX_ERROR max_get_free_eeprom_block (USHORT card, USHORT slot, USHORT layer, ULONG *size);

VB Declare Function max_get_free_eeprom_block Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByRef size As Long) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Mit dieser Funktion kann die Größe des größten zusammenhängenden unbenutzten EEPROM-Bereichs ermittelt werden.

Parameter

card Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).

slot Steckplatz des Moduls, auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

size Zeiger auf eine Variable, in die die Größe (in Bytes) des größten freien Blocks eingetragen wird.

max_create_eeprom_area Anlegen eines EEPROM-Bereichs

Pascal FUNCTION max_create_eeprom_area (card: WORD; slot: WORD; layer: WORD; usage: WORD; size: WORD; flags: WORD): MAX_ERROR;

C MAX_ERROR max_create_eeprom_area (USHORT card, USHORT slot, USHORT layer, USHORT usage, USHORT size, USHORT flags);

VB Declare Function max_create_eeprom_area Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByVal usage As Integer, ByVal size As Integer, ByVal flags As Integer) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion

Mit dieser Funktion wird ein Speicherbereich im EEPROM eines Moduls angelegt. Wenn der gewünschte Bereich bereits angelegt worden ist, liefert die Funktion ERR_EEPROM_REGION_EXISTS zurück. Der Bereich kann vor Schreib- und Lesezugriffen per Passwort gesichert werden. Nach dem Anlegen ist das Passwort standardmäßig ein leerer String. Mit Hilfe der Funktion **max_change_eeprom_password** kann das Passwort geändert werden.

Parameter

card Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).

slot Steckplatz des Moduls, auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

usage Angabe, welcher EEPROM-Bereich angelegt werden soll. User-Programme dürfen diesen Parameter nur = MAX_EEPAREA_USER_DATA setzen. Andere Bereiche werden durch Systemprogramme bzw. beim Hersteller eingerichtet.

size Gewünschte Größe des Bereichs. Da einige Modul-Device-Treiber ebenfalls EEPROM-Speicher benötigen (z.B. für Initialisierungs- oder Korrekturdaten), sollte nicht der gesamte EEPROM-Speicher für Anwender-Daten reserviert werden bevor der MDD nicht seinen Bereich angelegt hat.

flags Flags mit Eigenschaften des Bereichs:
MAX_CHECK_EEP_CHECKSUM: in dem neu angelegten Bereich wird eine Checksumme angelegt, die mit jedem Schreibzugriff aktualisiert wird. Die Schreibzugriffe dauern dadurch länger, dafür ist die Datenkonsistenz des Bereichs gewährleistet.
MAX_EEP_PASSREAD: Bei Lesezugriffen muss das Passwort angegeben werden.
MAX_EEP_PASSWRITE: Bei Schreibzugriffen auf den Bereich muss ein Passwort angegeben werden. Auch das Löschen des Bereichs ist geschützt.

max_change_eeprom_password — Ändern des Passworts für einen EEPROM-Bereich

Pascal	FUNCTION max_change_eeprom_password (card: WORD; slot: WORD; layer: WORD; usage: WORD; Const oldpass: PChar; Const newpass: PChar): MAX_ERROR;
C	MAX_ERROR max_change_eeprom_password (USHORT card, USHORT slot, USHORT layer, USHORT usage, char *oldpass, char *newpass);
VB	Declare Function max_change_eeprom_password Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByVal usage As Integer, ByVal oldpass As String, ByVal newpass As String) As Integer
Verwendung	Host-PC, MAX-PC OsX
Funktion	Mit dieser Funktion kann das Passwort, mit dem ein EEPROM-Bereich geschützt wird, geändert werden. Werksseitig ist kein Passwort vergeben, so dass beim ersten Aufruf dieser Funktion ein leerer String übergeben werden muss.

Parameter

card	Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).
slot	Steckplatz des Moduls, auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)
layer	Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.
usage	EEPROM-Bereich, für den das Passwort geändert werden soll. User-Programme dürfen diesen Parameter nur = MAX_EEPAREA_USER_DATA setzen.
oldpass	Bisheriges Passwort
newpass	Neues Passwort mit max. 8 Zeichen.

max_get_eeprom_area_size Größe eines EEPROM-Bereichs

Pascal FUNCTION max_get_eeprom_area_size (card: WORD; slot: WORD; layer: WORD; usage: WORD; VAR size: WORD): MAX_ERROR;

C MAX_ERROR max_get_eeprom_area_size (USHORT card, USHORT slot, USHORT layer, USHORT usage, USHORT *size);

VB Declare Function max_get_eeprom_area_size Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByVal usage As Integer, ByRef size As Integer) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Diese Funktion liefert die Größe eines im EEPROM angelegten Bereichs.

Parameter

card Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).

slot Steckplatz des Moduls, auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

usage EEPROM-Bereich, dessen Größe ermittelt werden soll. User-Programme dürfen diesen Parameter nur = MAX_EEPAREA_USER_DATA setzen.

size Zeiger auf eine Variable, in die die Länge des Bereichs (in Bytes) eingetragen wird

max_delete_eeprom_area Löschen eines EEPROM-Bereichs

Pascal FUNCTION max_delete_eeprom_area (card: WORD; slot: WORD; layer: WORD; usage: WORD; Const sPass: PChar): MAX_ERROR;

C MAX_ERROR max_delete_eeprom_area (USHORT card, USHORT slot, USHORT layer, USHORT usage, char *sPass);

VB Declare Function max_delete_eeprom_area Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByVal usage As Integer, ByVal sPass As String) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Mit dieser Funktion wird ein Speicherbereich im EEPROM eines Moduls gelöscht.

Parameter

card Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).

slot Steckplatz des Moduls , auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

usage Bereich, der gelöscht werden soll. User-Programme dürfen diesen Parameter nur = MAX_EEPAREA_USER_DATA setzen.

sPass Falls der Bereich mit Passwort-Schutz versehen ist, muss hier das Passwort als nullterminierter String übergeben werden.

max_read_eeprom_area max_write_eeprom_area Zugriff auf einen EEPROM-Bereich

Pascal

FUNCTION max_read_eeprom_area (card: WORD; slot: WORD; layer: WORD; usage: WORD; offset: WORD; size: WORD; VAR data; Const sPass: PChar): MAX_ERROR;

FUNCTION max_write_eeprom_area (card: WORD; slot: WORD; layer: WORD; usage: WORD; offset: WORD; size: WORD; VAR data; Const sPass: PChar): MAX_ERROR;

C

MAX_ERROR max_read_eeprom_area (USHORT card, USHORT slot, USHORT layer, USHORT usage, USHORT offset, USHORT size, void *data, char *sPass);

MAX_ERROR max_write_eeprom_area (USHORT card, USHORT slot, USHORT layer, USHORT usage, USHORT offset, USHORT size, void *data, char *sPass);

VB

Declare Function max_read_eeprom_area Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByVal usage As Integer, ByVal offset As Integer, ByVal size As Integer, ByRef data As Any, ByVal sPass As String) As Integer

Declare Function max_write_eeprom_area Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByVal usage As Integer, ByVal offset As Integer, ByVal size As Integer, ByRef data As Any, ByVal sPass As String) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Mit diesen Funktionen kann ein Speicherbereich im EEPROM eines Moduls gelesen bzw. beschrieben werden..

Parameter

card Nummer der Trägerkarte. In Echtzeitprogrammen hat der Parameter keine Bedeutung (=0 setzen).

slot Steckplatz des Moduls, auf dessen EEPROM zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

usage Bereich, der gelesen bzw. beschrieben werden soll. User-Programme dürfen zum Schreiben nur MAX_EEPAREA_USER_DATA angeben. Gelesen werden können durch Angabe von MAX_EEPAREA_MODULNAME (Text mit der Kurzbezeichnung des Moduls) MAX_EEPAREA_INFOTEXT (kurze Beschreibung des Moduls als Text), MAX_EEPAREA_MANUFACTURER (Angaben zum Hersteller) oder MAX_EEPAREA_PROD_DATA (Produktionsdaten) auch weitere Informationen (soweit sie vorhanden sind).

offset Offset (in Bytes) zum Beginn des Bereichs

size Größe des zu lesenden oder zu schreibenden Blocks

data Zeiger auf die Daten

sPass Falls der Bereich mit Passwort-Schutz versehen ist, muss hier das Passwort als nullterminierter String übergeben werden. Andernfalls NULL setzen.

6.5. Funktionen zur Verwaltung von MAX-Modulen

Die Kommunikation eines Anwenderprogramms auf einem Host-PC oder MAX-PC mit einem Zielsystem erfolgt grundsätzlich über ein Handle. Dieses muss sich das Anwenderprogramm zunächst mit Hilfe der Funktion **max_connect_cpu** holen. Für alle weiteren Zugriffe auf das Zielsystem ist das erhaltene Handle den entsprechenden Bibliotheksfunktionen zu übergeben.

max_connect_cpu Verbindungsaufbau zum Zielsystem

Pascal FUNCTION max_connect_cpu (card: WORD; slot: WORD; layer: WORD; VAR os: MAX_OS_INFO; VAR hModul: MAXMODHND): MAX_ERROR;

C MAX_ERROR max_connect_cpu (USHORT card, USHORT slot, USHORT layer, MAX_OS_INFO *os, MAXMODHND *hModul)

VB Declare Function max_connect_cpu Lib "maxw32.dll" (ByVal card As Integer, ByVal slot As Integer, ByVal layer As Integer, ByRef os As MAX_OS_INFO, ByRef hModul As Long) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Diese Funktion stellt die Verbindung zu einem Zielsystem (CPU-Modul oder Host-PC oder PCI-Karte ohne CPU-Modul) her und liefert dafür ein Handle, das anschließend für die meisten Bibliotheksfunktionen benötigt wird. Die Funktion ist daher in jedem Programm (auch in einem Echtzeitprogramm!) während der Initialisierung aufzurufen.

Parameter *card*, *slot*, *layer* Diese Parameter spezifizieren das Zielsystem, zu dem eine Verbindung hergestellt werden soll. Die untenstehende Tabelle zeigt die möglichen Einstellungen.

os Zeiger auf eine Struktur vom Typ MAX_OS_INFO, in die eingetragen wird, welches Betriebssystem auf dem CPU-Modul aktiv ist. Wenn die Informationen nicht benötigt werden, kann der Zeiger = NULL gesetzt werden. Die Struktur MAX_OS_INFO ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
usType	USHORT	Typ des aktiven Betriebssystems. Die zur Zeit definierten Werte sind in der nachfolgenden Tabelle aufgeführt.
version	MAX_ VERSION	Versionsangabe des Betriebs-systems. Für OsX entspricht das Format dem der Funktion **max_get_lib_version**.
ulSub	ULONG	Sub-Versionsangabe. Wird zur Zeit nicht verwendet
ulDate	ULONG	Erstellungsdatum des Betriebs-systems. Für OsX entspricht das Format dem der Funktion **max_get_lib_version**.

hModul Zeiger auf eine Variable, in die das Handle eingetragen wird. Bei fehlerhafter Ausführung dieser Funktion wird 0 zurückgeliefert. Dieses Handle ist allen Bibliotheksfunktionen zu übergeben, die auf das Zielsystem zugreifen.

!

Das erhaltene Handle ist Prozess-spezifisch, d.h. es darf nicht an andere Anwendungen weitergereicht werden. Andere Anwendungen müssen sich selbst ein Handle für das Zielsystem holen.

Im Parameter *os.usType* sind folgende Betriebssysteme definiert:

MAX_MINI_OSX	OsX-Betriebssystem mit eingeschränktem Funktionsumfang, das nach Reset automatisch anläuft
MAX_ROM_OSX	Vollständiges Betriebssystem, muss aktiviert werden oder kann durch besondere Maßnahmen nach Reset auch automatisch anlaufen.
MAX_RAM_OSX	Vollständiges Betriebssystem, das vom Host aus ins RAM des CPU-Moduls geladen wurde.

In der folgenden Tabelle sind die Einstellungen für die Parameter *card*, *slot* und *layer* für verschiedene Verbindungsarten zusammengefasst:

Verbindung von	**Verbindung zu**	**Verbindung über**	***card***	***slot*[1]**	***layer*[2]**
Host-PC	MAX-PC OsX	PCI-Bus	Karten-Nr. aus der Konfiguration	Steckplatz-Nr. oder MAX_-AUTO_SLOT[3]	Etagen-Nr. oder 0
Host-PC	MAX-PC OsX	Serielle Schnittstelle auf dem MAX-PC	Karten-Nr. aus der Konfiguration	0	0
Host-PC	MAX6pci ohne MAX-PC	PCI-Bus	Karten-Nr. aus der Konfiguration	0	0
MAX-PC OsX (Echtzeitprogramm)	Host-PC	PCI-Bus	MAX_PCI_HOST	0	0
MAX-PC OsX (Echtzeitprogramm)	Host-PC	Serielle Schnittstelle auf dem MAX-PC	MAX_SERIAL_HOST	0	0
MAX-PC OsX (Echtzeitprogramm)	Host-PC	Ethernet-Schnittstelle	MAX_ETHERNET_HOST	Steckplatz-Nr. des Moduls, auf dem sich die verwendete Schnittstelle befindet.	Etagen-Nr. des Moduls, auf dem sich die verwendete Schnittstelle befindet.
MAX-PC OsX (Echtzeitprogramm)	Anderer MAX-PC OsX auf derselben Trägerkarte	X-Bus	MAX_LOCAL_BOARD	Steckplatz-Nr. des anderen MAX-PC	Etagen-Nr. des anderen MAX-PC
MAX-PC OsX (Echtzeitprogramm)	Sich selbst	Lokal	MAX_LOCAL_BOARD	MAX_MY_SELF	0

[1] Steckplatz: Die Nummerierung der Steckplätze beginnt ab 1 und ist auf der Trägerkarte aufgedruckt
[2] Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0
[3] falls genau ein CPU-Modul aufgesteckt

max_exit_cpu Verbindungsabbau mit dem Zielsystem

Pascal FUNCTION max_exit_cpu (VAR hModul: MAXMODHND): MAX_ERROR;

C MAX_ERROR max_exit_cpu (MAXMODHND *hModul);

VB Declare Function max_exit_cpu Lib "maxw32.dll" (ByRef hModul As Long) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX, Host-PC → MAX6pci

Funktion Diese Funktion beendet die Kommunikation der Bibliothek mit dem Zielsystem. Das Handle wird geschlossen. Auch alle zum Zielsystem gehörenden Handles für Buffer, MDD usw. werden geschlossen.

Parameter *hModul* Zeiger auf eine Variable, in der das Handle des Zielsystems steht. Das Handle wird ungültig gemacht.

max_get_os_version Betriebssystemversion des MAX-PC

Pascal FUNCTION max_get_os_version (hModul: MAXMODHND; VAR os: MAX_OS_INFO): MAX_ERROR;

C MAX_ERROR max_get_os_version (MAXMODHND hModul, MAX_OS_INFO *os);

VB Declare Function max_get_os_version Lib "maxw32.dll" (ByVal hModul As Long, ByRef os As MAX_OS_INFO) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktion liefert den Typ und die Version des auf dem MAX-PC aktiven Betriebssystems.

Parameter *hModul* Handle auf das CPU-Modul

os Zeiger auf eine Struktur, in die eingetragen wird, welches Betriebssystem auf dem MAX-PC aktiv ist. Die Struktur ist bei **max_connect_cpu** erläutert.

max_reacting Kommunikationstest

Pascal FUNCTION max_reacting (hModul: MAXMODHND): MAX_ERROR;

C MAX_ERROR max_reacting (MAXMODHND hModul);

VB Declare Function max_reacting Lib "maxw32.dll" (ByVal hModul As Long) As Integer

Verwendung Host-PC → MAX-PC OsX

Funktion Diese Funktion prüft, ob eine Kommunikation der Bibliothek mit dem MAX-PC funktioniert.

Parameter *hModul* Handle auf das CPU-Modul

max_load_osx Betriebssystem-Download

Pascal FUNCTION max_load_osx (hModul: MAXMODHND; CONST osx: PChar): MAX_ERROR;

C MAX_ERROR max_load_osx (MAXMODHND hModul, char *osx)

VB Declare Function max_load_osx Lib "maxw32.dll" (ByVal hModul As Long, ByVal osx As String) As Integer

Verwendung Host-PC → MAX-PC OsX

Funktion Diese Funktion lädt ein Betriebssystem auf das CPU-Modul.

Parameter *hModul* Handle auf das CPU-Modul

osx Zeiger auf einen nullterminierten String, der angibt welches Betriebssystem aktiviert werden soll:
'ROM' : es wird das im Flash-ROM des MAX-PC befindliche Betriebssystem aktiviert.
'Dateiname': Der String wird als Name einer Betriebssystem-Datei interpretiert. Diese wird in das RAM des MAX-PC geladen und aktiviert. Damit lassen sich auf einfache Weise Updates des Betriebssystems vornehmen. Die Joker '*' und '? ' im String sind erlaubt. Bei Angabe der Konstante MAX1_NEWEST_OSX verwendet die Funktion die aktuellste Betriebssystem-Datei aus dem Verzeichnis „SNW32\OSX“.

max_get_slot Steckplatz ermitteln

Pascal FUNCTION max_get_slot (hModul: MAXMODHND; VAR slot: WORD; VAR layer: WORD): MAX_ERROR;

C MAX_ERROR max_get_slot (MAXMODHND hModul, USHORT *slot, USHORT *layer)

VB Declare Function max_get_slot Lib "maxw32.dll" (ByVal hModul As Long, ByRef slot As Integer, ByRef layer As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit dieser Funktion kann ermittelt werden, auf welchem Steckplatz sich ein CPU-Modul befindet. Das wird z.B. in Echtzeitprogrammen benötigt oder bei einem per Remote-Verbindung angebundenen CPU-Modul. Die Steckplatznummer ist z.B. bei EEPROM- oder Flash-Zugriffen anzugeben.

Parameter
- *hModul* Handle auf das CPU-Modul
- *slot* Zeiger auf eine Variable, in die die Steckplatznummer des CPU-Moduls eingetragen wird
- *layer* Zeiger auf eine Variable, in die die Etagennummer des CPU-Moduls eingetragen wird

max_get_max_blocksize Kommunikationspufferlänge

Pascal FUNCTION max_get_max_blocksize (hModul: MAXMODHND; VAR receive: LONGWORD; VAR transmit: LONGWORD): MAX_ERROR;

C MAX_ERROR max_get_max_blocksize (MAXMODHND hModul, ULONG *receive, ULONG *transmit)

VB Declare Function max_get_max_blocksize Lib "maxw32.dll" (ByVal hModul As Long, ByRef receive As Long, ByRef transmit As Long) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX → MAX-PC OsX

Funktion Diese Funktion liefert Information darüber, wie lang die Datenblöcke bei bestimmten Funktionen maximal sein dürfen. Die Länge ist abhängig vom auf dem CPU-Modul aktiven Betriebssystem und davon, von wo das CPU-Modul angesprochen wird. Sie kann für Datenblöcke, die zum CPU-Modul gesendet werden unter Umständen anders sein als für Datenblöcke, die vom CPU-Modul angefordert werden.

Parameter *hModul* Handle auf das CPU-Modul

receive Zeiger auf eine Variable, in die die maximale Länge von Datenblöcken eingetragen wird, die das CPU-Modul empfangen kann (gilt für **max_write_channel_block, max_channel_control** und **max_call_func)**.

transmit Zeiger auf eine Variable, in die die maximale Länge von Datenblöcken eingetragen wird, die von einem CPU-Modul angefordert werden kann (gilt für **max_read_channel_block, max_channel_info** und **max_call_func)**.

6.6. Funktionen für die Verwendung von INS-Files

INS-Files sind Dateien, die verschiedene Befehle zum Ansprechen der X-Bus-Hardware enthalten. Zulässige Anweisungen sind im Anhang G beschrieben. Die Erstellung solcher Dateien und ihr Test erfolgt am besten mit SNW32.

Die Bibliothek stellt für die Verwendung von INS-Files in eigenen Programmen die Funktion **max_execute_ins_file** zum Ausführen der in einer INS-Datei enthaltenen Anweisungen zur Verfügung. Für das Schreiben einer INS-Datei inklusive der darin ggf. zum Download enthaltenen Echtzeitprogramme in das Flash eines CPU-Modules gibt es die Funktion **max_write_ins_file_to_flash**.

max_execute_ins_file Befehlsliste in einer INS-Datei abarbeiten

Pascal FUNCTION max_execute_ins_file (hModul: MAXMODHND; insfile: PChar; mode: WORD): MAX_ERROR;

C MAX_ERROR max_execute_ins_file (MAXMODHND hModul, char *insfile, USHORT mode);

VB Declare Function max_execute_ins_file Lib "maxw32.dll" (ByVal hModul As Long, ByVal insfile As String, ByVal mode As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX

Funktion Die in einer INS-Datei stehende Liste von Befehlen wird mit Hilfe dieser Funktion abgearbeitet. Liefert die Funktion den Fehler ERR_INS_FILE_SYNTAX, so wird empfohlen im INS-File-Editor von SNW32 den Syntax-Check auf die INS-Datei anzuwenden, um den Fehler genauer zu lokalisieren. Die Funktion greift auf die im SNW32\BIN-Verzeichnis befindliche A_MLX.DLL zurück.

Parameter
- *hModul* Handle auf das CPU-Modul
- *insfile* Name der INS-Datei als nullterminierter String. Die Namenserweiterung .INS muss enthalten sein. Wird hier NULL angegeben, öffnet die Funktion die Windows-Dateiauswahl-Dialogbox. Eventuell durch die INS-Datei ins Flash geladene Echtzeitprogramme müssen sich im selben Verzeichnis wie die INS-Datei befinden.
- *mode* reserviert = 0 setzen

max_write_ins_file_to_flash Schreiben eines INS-Files ins Flash

Pascal FUNCTION max_write_ins_file_to_flash (hModul: MAXMODHND; slot: WORD, layer: WORD, chip: WORD, insfile: PChar, mode: WORD): MAX_ERROR;

C MAX_ERROR max_write_ins_file_to_flash (MAXMODHND hModul, USHORT slot, USHORT layer, USHORT chip, char *insfile, USHORT mode);

VB Declare Function max_write_ins_file_to_flash Lib "maxw32.dll" (ByVal hModul As Long, ByVal slot as Integer, ByVal layer as Integer,

ByVal chip as Integer, ByVal insfile as String, ByVal mode as Integer) As Integer

Verwendung Host-PC → MAX-PC OsX

Funktion Diese Funktion lädt eine INS-Datei in das Flash-ROM eines CPU-Moduls. Echtzeitprogramme, die in der INS-Datei mittels MAXINST-Befehlen auf das CPU-Modul geladen werden sollen, werden zusammen mit einem Loader-Programm ebenfalls ins Flash geschrieben.

Zu beachten ist, dass sich nur eine einzige INS-Datei im Flash befinden darf. Eine bereits im Flash stehende INS-Datei wird durch einen Aufruf von **max_write_ins_file_to_flash** überschrieben. Die Befehle MAXRESET, MAXLOADOSX und MAXCONNECT_CPU in der INS-Datei werden ignoriert.

Die ins Flash geschriebenen Anweisungen werden automatisch abgearbeitet, sobald **max_reset_board** ausgeführt wird und das ROM-OsX aktiviert wird.

Die Funktion greift auf die im SNW32\BIN-Verzeichnis befindliche A_XMODULE.DLL zurück.

Parameter

hModul Handle auf das CPU-Modul

slot Steckplatz des Moduls, in dessen Flash die INS-Datei geschrieben werden soll

layer Etagennummer des Moduls, in dessen Flash die INS-Datei geschrieben werden soll

chip IC, in das die INS-Datei geschrieben werden soll. Dieser Parameter muss für das X-MAX-1 oder X-MAX-E =3 gesetzt werden.

insfile Name der INS-Datei als nullterminierter String (s. a. max_execute_ins_file)

mode reserviert = 0 setzen

6.7. Funktionen für Zugriffe auf das RAM eines MAX-PC

max_allocate_ram — Speicher reservieren

Pascal: FUNCTION max_allocate_ram (hModul: MAXMODHND; VAR par: MAX_ALLOC_TYPE; VAR size: LONGWORD; VAR address: LONGWORD): MAX_ERROR;

C: MAX_ERROR max_allocate_ram (MAXMODHND hModul, MAX_ALLOC_TYPE *par, ULONG *size, ULONG *address);

VB: Declare Function max_allocate_ram Lib "maxw32.dll" (ByVal hModul As Long, ByRef par As MAX_ALLOC_TYPE, ByRef size As Long, ByRef address As Long) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Mit dieser Funktion wird ein Speicherbereich reserviert. Die Zugriffe darauf müssen mit den Funktionen **max_read_memory** und **max_write_memory** erfolgen.

Parameter:

hModul Handle des Moduls

par Zeiger auf eine Datenstruktur vom Typ MAX_ALLOC_TYPE, in der die gewünschten Eigenschaften des zu reservierenden Speichers stehen. Die Struktur MAX_ALLOC_TYPE ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
usStrategy	USHORT	Strategie der Speicherreservierung. Die möglichen Werte sind in der unten stehenden Tabelle aufgeführt
usAlignment	USHORT	Ausrichtung des Anfangs des Speichers in Potenzen zu 2 (0 = Byte, 1 = Wort, 2 = Doppelwort, ...)

Element	Datentyp	Beschreibung
usTask	USHORT	Nummer der Task, die den Speicher anfordert (nur zu Protokollzwecken)
usUsage	USHORT	Reserviert (= 0 setzen)

size Zeiger auf eine Variable, in der die gewünschte Größe des zu reservierenden Speicherbereiches steht. Im Anschluss an den Aufruf wird dort die tatsächliche Größe eingetragen.

address Zeiger auf eine Variable, in die die physikalische Anfangsadresse des reservierten Speicherbereiches eingetragen wird. Wenn für *usStrategy* in einem Echtzeitprogramm der Wert MAX_ALLOC_DOS_-MEMORY verwendet wird, trägt die Funktion in diese Variable die Anfangsadresse des reservierten Speichers unterhalb 1MB als Segment:Offset Adresse ein (s.u.).

Mögliche Werte für die Strategie der Speicherreservierung:

MAX_ALLOC_ UP_ABS	Speicher von unten reservieren. Wenn nicht genügend Speicher zur Verfügung steht, wird ein Fehler gemeldet
MAX_ALLOC_ UP_MAX	Speicher von unten reservieren. Wenn nicht genügend Speicher zur Verfügung steht, wird so viel wie möglich reserviert.
MAX_ALLOC_ DOWN_ABS	Speicher von oben reservieren. Wenn nicht genügend Speicher zur Verfügung steht, wird ein Fehler gemeldet.
MAX_ALLOC_ DOWN_MAX	Speicher von oben reservieren. Wenn nicht genügend Speicher zur Verfügung steht, wird so viel wie möglich reserviert.

MAX_ALLOC_ DOS_MEMORY	Speicher entsprechend der Strategie MAX_ALLOC_UP_ABS reservieren. Zusätzlich prüft die Funktion, ob der gesamte angeforderte Speicher unterhalb der 1 MB Grenze angelegt werden konnte. Wenn das möglich ist, kann dann in einem Echtzeitprogramm direkt mit Real-Pointern gearbeitet werden, sonst liefert die Funktion einen Fehler. MAX_ALLOC_DOS_MEMORY steht nur in den Bibliotheken für das OsX (Echtzeitprogramme) zur Verfügung.

Wird in einem Echtzeitprogramm das Strukturelement *usStrategy* mit dem Wert MAX_ALLOC_DOS_MEMORY ausgefüllt, gibt *address* die Segment:Offset Anfangsadresse des reservierten Speicherbereiches zurück. Für *address* muss also in diesem Fall ein C- oder Pascal-Pointer eingesetzt werden, über den anschließend auf den reservierten Speicher direkt zugegriffen werden kann:

```
UCHAR           *pData;
ULONG           ulSize = 72; // allocate 72 bytes
MAX_ALLOC_TYPE  rcMemory;

rcMemory.usStrategy = MAX_ALLOC_DOS_MEMORY;
rcMemory.usAlignment = 1;
rcMemory.usTask = 100;
rcMemory.usUsage = 0;
Error = max_allocate_ram (hModul, &rcMemory, &ulSize, (ULONG*)&pData);
```

max_get_ram_info RAM-Informationen

Pascal FUNCTION max_get_ram_info (hModul: MAXMODHND; VAR info: MAX_RAM_TYPE): MAX_ERROR;

C MAX_ERROR max_get_ram_info (MAXMODHND hModul, MAX_RAM_TYPE *info);

VB Declare Function max_get_ram_info Lib "maxw32.dll" (ByVal hModul As Long, ByRef prcInfo As MAX_RAM_TYPE) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit dieser Funktion können Informationen über den RAM-Speicher des angegebenen MAX-PC ermittelt werden, u.a. wie viel freier RAM-Speicher zur Verfügung steht.

Parameter *hModul* Handle des Moduls

info Zeiger auf eine Datenstruktur vom Typ MAX_RAM_TYPE, in die die folgenden Informationen eingetragen werden. Die Struktur MAX_RAM_TYPE ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
ulFree	ULONG	Anzahl der zur freien Verfügung stehenden Bytes
ulTotal	ULONG	Größe des gesamten RAM-Speichers in Bytes
ulStart	ULONG	Anfangsadresse des freien Speichers
ulEnd	ULONG	Endadresse des freien Speichers

max_write_memory max_read_memory — Speicherzugriffe

Pascal

```
FUNCTION max_write_memory (hModul: MAXMODHND; address: LONGWORD; VAR size: LONGWORD; VAR data): MAX_ERROR;

FUNCTION max_read_memory (hModul: MAXMODHND; address: LONGWORD; VAR size LONGWORD; VAR data): MAX_ERROR;
```

C

```
MAX_ERROR max_write_memory (MAXMODHND hModul, ULONG address, ULONG *size, void *data);

MAX_ERROR max_read_memory (MAXMODHND hModul, ULONG address, ULONG *size, void *data);
```

VB

```
Declare Function max_write_memory Lib "maxw32.dll" (ByVal hModul As Long, ByVal address As Long, ByRef size As Long, ByRef data As Any) As Integer

Declare Function max_read_memory Lib "maxw32.dll" (ByVal hModul As Long, ByVal address As Long, ByRef size As Long, ByRef data As Any) As Integer
```

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktionen dienen zum Schreiben und Lesen des RAM-Speichers des angegebenen MAX-PC. Normalerweise kann ein Programm im Real-Mode nur die unteren 1 MByte des 4 GByte großen Speicherbereichs der CPU zugreifen.

Parameter *hModul* Handle des Moduls

address Physikalische Anfangsadresse, ab der geschrieben oder gelesen werden soll.

size Zeiger auf eine Variable, in der die Anzahl Bytes steht, die gelesen bzw. geschrieben werden sollen. Die Funktion trägt in die Variable ein, wie viele Bytes tatsächlich gelesen bzw. geschrieben worden sind.

data Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden, bzw. in der die zu schreibenden Daten stehen.

max_get_physical_addr Physikalische Adresse

Pascal FUNCTION max_get_physical_addr (VAR ptr): LONGWORD;

C ULONG max_get_physical_addr (void *ptr);

VB Declare Function max_get_physical_addr Lib "maxw32.dll" (ByRef ptr As Any) As Long

Verwendung Host-PC, MAX-PC OsX

Funktion Diese Funktion konvertiert eine Segment:Offset-Adresse in eine physikalische Adresse, wie sie z.B. von Bibliotheksfunktionen für RAM-
oder Flash-Zugriffe gefordert wird. Physikalische Adressen erlauben im Gegensatz zu den normalerweise in 16-Bit-Programmen verwendeten Segment:Offset-Adressen auch die Adressierung von Speicherzellen oberhalb 1 MB.

Parameter *ptr* Zu konvertierende Segment:Offset-Adresse

Rückgabewert Physikalische Adresse

6.8. Funktionen für den Zugriff auf das Flash des MAX-PC

max_get_flash_info Flash-ROM-Informationen

Pascal FUNCTION max_get_flash_info (hModul: MAXMODHND; slot: WORD; layer: WORD; chip: WORD; VAR info: MAX_FLASH_TYPE): MAX_ERROR;

C MAX_ERROR max_get_flash_info (MAXMODHND hModul, USHORT slot, USHORT layer, USHORT chip, MAX_FLASH_TYPE *info);

VB Declare Function max_get_flash_info Lib "maxw32.dll" (ByVal hModul As Long, ByVal slot As Integer, ByVal layer As Integer, ByVal chip As Integer, ByRef info As MAX_FLASH_TYPE) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktion liefert Informationen über den Flash-ROM-Speicher auf einem MAX-Modul.

Parameter

hModul Handle des Moduls

slot Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

chip Angabe über welches IC Informationen eingeholt werden sollen. Dieser Parameter muss für das X-MAX-1 oder X-MAX-E = 3 gesetzt werden.

info Zeiger auf eine Datenstruktur vom Typ MAX_FLASH_TYPE, in die die gelesenen Informationen eingetragen werden. Die Struktur MAX_FLASH_TYPE ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
usManufacturer	USHORT	Flash-Hersteller ID
usDevice	USHORT	Flash-Device ID
ulFlashSize	ULONG	Speichergröße (in Bytes)
ulSectorSize	ULONG	Sektorgröße (in Bytes)
ulSectorCount	ULONG	Anzahl an Sektoren
usOrganization	USHORT	Organisation des Flash (0 = Bit, 1 = Byte, 2= Wort)
usProtectable	USHORT	Welche Sektoren schreibgeschützt werden können (0 = keine, 3 = alle, 2 = nur der oberste Sektor, 1 = nur der unterste Sektor)
usBufferSize	USHORT	Größe des Schreibpuffers
usStatus	USHORT	Zustand des Chips

max_flash_sector_info Flash-Sektor-Informationen

Pascal FUNCTION max_flash_sector_info (hModul: MAXMODHND; slot: WORD; layer: WORD; chip: WORD; sector: WORD; VAR protection: WORD): MAX_ERROR;

C MAX_ERROR max_flash_sector_info (MAXMODHND hModul, USHORT slot, USHORT layer, USHORT chip, USHORT sector, USHORT *protection);

VB Declare Function max_flash_sector_info Lib "maxw32.dll" (ByVal hModul As Long, ByVal slot As Integer, ByVal layer As Integer, ByVal chip As Integer, ByVal sector As Integer, ByRef protection As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktion liefert Informationen über einen Sektor im Flash-ROM-Speicher auf dem CPU-Modul.

Parameter

hModul Handle des Moduls

slot Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

chip Angabe, über welches IC Informationen eingeholt werden sollen. Dieser Parameter muss für das X-MAX-1 oder X-MAX-E = 3 gesetzt werden.

sector Sektornummer

protection Zeiger auf eine Variable, in die eingetragen wird, ob der Sektor schreibgeschützt ist (=1) oder nicht (= 0)

max_erase_flash Flash-Sektoren löschen

Pascal FUNCTION max_erase_flash (hModul: MAXMODHND; slot: WORD; layer: WORD; chip: WORD; first: LONGWORD; last: LONGWORD): MAX_ERROR;

C MAX_ERROR max_erase_flash (MAXMODHND hModul, USHORT slot, USHORT layer, USHORT chip, ULONG first, ULONG last);

VB Declare Function max_erase_flash Lib "maxw32.dll" (ByVal hModul As Long, ByVal slot As Integer, ByVal layer As Integer, ByVal chip As Integer, ByVal first As Long, ByVal last As Long) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktion löscht einen Bereich im Flash-ROM-Speicher auf dem CPU-Modul. Es ist zu beachten, dass das Flash in Sektoren organisiert ist. Sektoren können nur als Ganzes gelöscht werden. Liegt die Anfangsadresse nicht am Anfang eines Sektors, so wird auch der Bereich zwischen Sektoranfang und Startadresse gelöscht. Das Löschen von Sektoren dauert relativ lange, so dass man diese Funktion nie in zeitkritischen Programmteilen einsetzen sollte.

Parameter	*hModul*	Handle des Moduls
	slot	Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)
	layer	Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.
	chip	Angabe, in welchem IC gelöscht werden soll. Dieser Parameter muss für das X-MAX-1 oder X-MAX-E = 3 gesetzt werden.
	first	Angabe der Adresse, ab der gelöscht werden soll
	last	Angabe der Adresse, bis zu der gelöscht werden soll

max_write_flash max_read_flash — Flash-Zugriff

Pascal

```
FUNCTION max_write_flash (hModul: MAXMODHND; slot: WORD; layer: WORD; chip: WORD; start: LONGWORD; size: LONGWORD: VAR data): MAX_ERROR;

FUNCTION max_read_flash (hModul: MAXMODHND; slot: WORD; layer: WORD; chip: WORD; start: LONGWORD; size: LONGWORD; VAR data): MAX_ERROR;
```

C

```
MAX_ERROR max_write_flash (MAXMODHND hModul, USHORT slot, USHORT layer, USHORT chip, ULONG start, ULONG size, void *data);

MAX_ERROR max_read_flash (MAXMODHND hModul, USHORT slot, USHORT layer, USHORT chip, ULONG start, ULONG size, void *data);
```

VB

```
Declare Function max_write_flash Lib "maxw32.dll" (ByVal hModul As Long, ByVal slot As Integer, ByVal layer As Integer, ByVal chip As Integer, ByVal start As Long, ByVal size As Long, ByRef data As Any) As Integer

Declare Function max_read_flash Lib "maxw32.dll" (ByVal hModul As Long, ByVal slot As Integer, ByVal layer As Integer, ByVal chip As Integer, ByVal start As Long, ByVal size As Long, ByRef data As Any) As Integer
```

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit diesen Funktionen können Daten in das Flash-ROM auf dem MAX-PC geschrieben bzw. ausgelesen werden. Bevor Daten in das Flash geschrieben werden können, müssen die entsprechenden Sektoren mit **max_erase_flash** gelöscht werden.

Parameter

hModul Handle des Moduls

slot Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

chip Angabe, in welchem IC geschrieben/gelesen werden soll. Dieser Parameter muss für das X-MAX-1 oder X-MAX-E = 3 gesetzt werden.

start Relative Adresse (bezogen auf den Anfang des Flash), ab der Daten zu lesen bzw. zu schreiben sind

size Anzahl Bytes, die gelesen bzw. geschrieben werden sollen

data Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden, bzw. in der die zu schreibenden Daten stehen.

6.9. Funktionen für Modul-Device-Treiber-Kommunikation

Das Ansprechen von MAX-Modulen erfolgt grundsätzlich über die sogenannten Modul-Device-Treiber (MDD, s. auch Kapitel 8). Dies sind Programme mit einer normierten Schnittstelle zur Bibliothek, über die alle Zugriffe auf die Hardware abgewickelt werden. Sie können auf einem MAX-PC oder dem PC laufen. Für jedes Modul steht in der Regel ein MDD zur Verfügung. Für Spezialzwecke kann es auch mehrere MDDs geben.

Damit ein MDD benutzt werden kann, muss er zunächst auf ein CPU-Modul geladen werden. Für Karten ohne CPU-Modul kann er auch auf dem Host-PC installiert werden. Dieser Vorgang ist mit **max_load_mdd** zu erledigen. Das von der Funktion gelieferte MDD-Handle ist anschließend beim Öffnen von Kanälen sowie weiteren im folgenden beschriebenen Funktionen zu übergeben.

Die Zugriffe auf die Hardware erfolgen nach einem kanalorientierten Konzept. Bevor man auf ein Device zugreifen kann, muss man zunächst einen Kanal zum MDD öffnen. Der MDD liefert bei Erfolg eine eindeutige Nummer (Handle) zurück. Die beim Öffnen eines Kanals anzugebenden Parameter sind Device-spezifisch und werden in der Beschreibung des jeweiligen Moduls erläutert (Kapitel 10). Die weiteren Zugriffe über die geöffneten Kanäle auf die Hardware des Moduls sind Hardware-unabhängig. Die entsprechenden Bibliotheksfunktionen sind für alle MDD’s die gleichen.

Weitere Details zu MDD’s finden Sie in Kapitel 8.

Das MDD Konzept hat den Vorteil, dass ein Hardwaretausch ohne irgendwelche Softwareänderungen möglich ist. Das garantiert eine lange Verfügbarkeit von SORCUS-Produkten. Selbst wenn ein Chip-Hersteller die Produktion eines Chips einstellt, z.B. ein A/D-Wandler, der auf einem Modul verwendet wird, hat das keine Bedeutung für einen SORCUS-Kunden (und das Produkt, in dem SORCUS-Produkte eingesetzt sind). Es braucht lediglich ein neuer MDD geladen werden, eventuell müssen zusätzlich einige Parameter beim Öffnen des Kanals verändert werden. Die eigentlichen Hardware-Zugriffe bleiben aber unverändert.

max_load_mdd — Download eines MDD

Pascal

```
FUNCTION max_load_mdd (hModul: MAXMODHND; slot: WORD;
layer: WORD; submodul: WORD; prognr: WORD; VAR program:
MAX_PGM_TYPE; VAR hMdd: MAXMDDHND): MAX_ERROR;
```

C MAX_ERROR max_load_mdd (MAXMODHND hModul, USHORT slot, USHORT layer, USHORT submodul, USHORT prognr, MAX_PGM_TYPE *program, MAXMDDHND *hMdd);

VB Declare Function max_load_mdd Lib "maxw32.dll" (ByVal hModul As Long, ByVal slot As Integer, ByVal layer As Integer, ByVal submodul As Integer, ByVal prognr As Integer, ByRef program As MAX_PGM_TYPE, ByRef hMdd As Long) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Diese Funktion prüft, ob für das angegebene Modul bzw. Sub-Modul bereits ein MDD installiert ist. Ist dies noch nicht der Fall, lädt sie den angegebenen MDD, im anderen Fall wird nur die Verbindung zum bereits geladenen MDD hergestellt. Die Funktion liefert ein Handle auf den MDD zurück.
Auch Echtzeitprogramme müssen diese Funktion vor der Verwendung eines MDDs einmal aufrufen, um ein MDD-Handle zu bekommen. Dabei wird jedoch vorausgesetzt, dass der MDD vorher bereits installiert worden ist.

Parameter

hModul Handle des Zielsystems (Host-PC oder MAX-PC) , auf das der MDD geladen werden soll.

slot Steckplatz des Moduls, das der MDD unterstützt (s. Aufdruck auf der Trägerkarte)

layer Etagennummer des Moduls. Die unterste Etage direkt auf der Trägerkarte erhält die Nummer 0.

submodul Für Module, die von mehr als einem MDD gleichzeitig unterstützt werden, wird hier angegeben, für welches Sub-Modul der MDD eingesetzt werden soll. Für Module, die von nur einem MDD unterstützt werden, ist submodul = 0 zu setzen.

prognr Programmnummer des zu ladenden MDD. Wird die Nummer hier angegeben, versucht die Bibliothek, den entsprechenden MDD automatisch zu laden (*Auto-Load*). Die Datenstruktur *program* braucht dann nicht ausgefüllt zu werden. Die Funktion sucht die entsprechende Datei zunächst in dem durch **max_set_mdd_path** festgelegten Verzeichnis (sofern max_set_mdd_path ausgeführt worden ist). Ist der MDD dort nicht zu finden, wird er im

SNW32/MDD-Verzeichnis gesucht (sofern vorhanden). Schlägt auch diese Suche fehl, wird als letztes das aktuelle Verzeichnis durchsucht.
Wird prognr = 0 gesetzt, muss die Datenstruktur *program* ausgefüllt werden.

program Soll **nicht** der für das Modul standardmäßig vorgesehene MDD geladen werden, ist *prognr* = 0 zu setzen. Nur in diesem Fall muss in *program* ein Zeiger auf eine Datenstruktur mit allen Informationen, die für die Installation des MDD benötigt werden, übergeben werden. Die Struktur ist unter der Funktion **max_transfer_and_-install** erläutert.
Falls der für das Modul standardmäßig vorgesehene MDD automatisch geladen werden soll, kann *program* =NULL gesetzt werden.
In Echtzeitprogrammen braucht der Parameter nicht ausgefüllt zu werden, die Kenntnisse über den MDD benötigt ein Echtzeitprogramm nicht.

hMdd Zeiger auf eine Variable, in die das Handle des MDD eingetragen wird. Bei fehlerhafter Ausführung der Funktion ist hMdd = 0. Dieses Handle ist allen Bibliotheksfunktionen zu übergeben, die auf den MDD zugreifen.

! *Das erhaltene Handle ist Prozess-spezifisch, d.h. es darf nicht an andere Anwendungen weitergereicht werden. Andere Anwendungen müssen sich selbst ein Handle für den MDD holen.*

max_set_mdd_path MDD-Verzeichnis

Pascal FUNCTION max_set_mdd_path (CONST path: PChar): MAX_ERROR;

C MAX_ERROR max_set_mdd_path (const char *path);

VB Declare Function max_set_mdd_path Lib "maxw32.dll" (ByVal path As String) As Integer

Verwendung Host-PC

Funktion Mit dieser Funktion kann der Verzeichnispfad zu den MDDs eingestellt werden, aus dem **max_load_mdd** bei Verwendung des *Auto-Load*-Mechanismus (Parameter *prognr* ungleich 0) den MDD auf des entsprechende CPU-Modul lädt.

Parameter *path* Zeiger auf einen nullterminierten String mit dem Pfadnamen

max_get_mdd_version MDD-Version

Pascal FUNCTION max_get_mdd_version (hMdd: MAXMDDHND; VAR version: MAX_VERSION; VAR date: LONGWORD): MAX_ERROR;

C MAX_ERROR max_get_mdd_version (MAXMDDHND hMdd, MAX_VERSION *version, ULONG *date);

VB Declare Function max_get_mdd_version Lib "maxw32.dll" (ByVal hMdd As Long, ByRef version As Long, ByRef date As Long) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Diese Funktion liefert die Version und das Erstellungsdatum des MDD´s, siehe auch **max_get_version_string**.

Parameter *hMdd* Handle auf den MDD

version Zeiger auf eine Variable, in die die Version des MDD in folgendem Format eingetragen wird:
Byte 0-1: Revisionsnummer
Byte 2: Revisionsbuchstabe (z.B. ‚A‘)
Byte 3: Versionsnummer

date Zeiger auf eine Variable, in die das Erstellungsdatum des MDD in folgendem Format eingetragen wird:
Byte 0: Tag (1..31)
Byte 1: Monat (1..12)
Byte 2-3: Jahr (0.. 65535)

max_reset_mdd — Reset eines MDD

Pascal FUNCTION max_reset_mdd (hMdd: MAXMDDHND): MAX_ERROR;

C MAX_ERROR max_reset_mdd (MAXMDDHND hMdd);

VB Declare Function max_reset_mdd Lib "maxw32.dll" (ByVal hMdd As Long) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Diese Funktion setzt den angegebenen MDD zurück. Alle zu dem MDD geöffneten Kanäle werden ungültig.

Parameter *hMdd* Handle auf den MDD

max_read_infotext — MDD-Informationen

Pascal FUNCTION max_read_infotext (hMdd: MAXMDDHND; page: WORD; line: WORD; VAR size: WORD; text: PChar): MAX_ERROR;

C MAX_ERROR max_read_infotext (MAXMDDHND hMdd, USHORT page, USHORT line, USHORT *size, char *text);

VB Declare Function max_read_infotext Lib "maxw32.dll" (ByVal hMdd As Long, ByVal page As Integer, ByVal line As Integer, ByRef size As Integer, ByVal text As String) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Mit dieser Funktion können im MDD gespeicherte Informationen ausgelesen werden. Auf bis zu 255 Seiten (page) können jeweils bis zu 255 Zeilen (line) Text gespeichert sein. Standardisiert sind jeweils die ersten beiden Zeilen von Seite 0 und Seite 1:
Seite 0: Zeile 0: Bezeichnung des MDD
Zeile 1: Revision des MDD

Seite 1: Zeile 0: Herstellerfirma des MDD
Zeile 1: Autor

Parameter *hMdd* Handle auf den MDD

page Seite des Infotextes

line Zeile des Infotextes

size Zeiger auf eine Variable, die die maximale Anzahl auszulesender Zeichen angibt (max. 80).

text Zeiger auf eine Variable, in die der Text als nullterminierter String eingetragen wird.

max_device_status Status eines MDD-Device

Pascal FUNCTION max_device_status (hMdd: MAXMDDHND; VAR device: MAX_DEV_SPEC_TYPE; VAR status: WORD): MAX_ERROR;

C MAX_ERROR max_device_status (MAXMDDHND hMdd, MAX_DEV_SPEC_TYPE *device, USHORT *status);

VB Declare Function max_device_status Lib "maxw32.dll" (ByVal hMdd As Long, ByRef device As MAX_DEV_SPEC_TYPE, ByRef status As Integer) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Mit dieser Funktion kann ausgelesen werden, wie viele Kanäle zu einem Device geöffnet sind.

Parameter *hMdd* Handle auf den MDD

device Zeiger auf eine Variable vom Typ MAX_DEV_SPEC_TYPE, die das Device spezifiziert. Die Datenstruktur MAX_DEV_SPEC_TYPE hat folgenden Aufbau:

Element	Datentyp	Beschreibung
ulType	ULONG	Typ des Device (entspricht der Angabe *usDevice* in der CPS)

Element	Datentyp	Beschreibung
ulIndex	ULONG	Nummer des Devices
ulSubIndex	ULONG	Für einige Devices, die sich extern befinden (z.B. über einen Feldbus verbunden sind) ist die Angabe eines Sub-Index erforderlich.

status Zeiger auf eine Variable, in die die Anzahl der zu dem Device geöffneten Kanäle eingetragen wird. Foldende Werte sind möglich:

Wert	Bedeutung
= 0	Zu dem Device ist kein Kanal geöffnet
= DSC_IS_EXCLUSIVE	Das Device ist exklusiv für einen einzigen Kanal reserviert.
Alle anderen Werte	Der Zählerstand gibt die Anzahl Kanäle an, die zu dem Device geöffnet sind.

6.10. Funktionen für den MDD-Kanalzugriff

Bevor Kanalzugriffe auf die vom MDD unterstützten Devices gemacht werden können, muss der MDD zunächst mit **max_load_mdd** installiert worden sein. Anschließend muss unter Angabe des Devices und der gewünschten Kanaleigenschaften mit **max_open_channel** ein Kanal zu dem entsprechenden Device geöffnet werden. Das dabei zurückgelieferte Handle wird für alle Kanalzugriffe benötigt.

max_open_channel — Öffnen eines MDD-Kanals

Pascal
FUNCTION max_open_channel (hMdd: MAXMDDHND; usCpsSize: WORD; VAR *pCPS; VAR pCbFunc: MAX_CHANNEL_CALLBACK_TYPE; ulPar: LONGWORD; VAR hChannel: MAXCHLHND): MAX_ERROR;

C
MAX_ERROR max_open_channel (MAXMDDHND hMdd, USHORT usCpsSize, void *pCPS, MAX_CHANNEL_CALLBACK_TYPE *pCbFunc, ULONG ulPar, MAXCHLHND *hChannel);

VB
Declare Function max_open_channel Lib "maxw32.dll" (ByVal hMdd As Long, ByVal usCpsSize As Integer, ByRef pCPS As Any, ByVal pCbFunc As Long, ByVal ulPar As Long, ByRef hChannel As Long) As Integer

Verwendung
Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion
Diese Funktion öffnet einen Kanal zu einem Device des durch *hMdd* angegebenen MDD. Die Funktion liefert ein Handle auf den neu geöffneten Kanal zurück. Darüber kann dann mit dem Kanal kommuniziert werden.

Parameter

hMdd Handle auf den MDD, in dem ein Kanal geöffnet werden soll.

usCpsSize Größe der in pCPS übergebenen CPS-Datenstruktur.

pCPS Zeiger auf die CPS (Channel Property Structure) des Kanals. In der CPS steht zu welchem Device und mit welchen Eigenschaften der Kanal geöffnet werden soll. Die Datenstruktur ist abhängig vom jeweiligen MDD und vom Device. Ihr Aufbau kann aus der Dokumentation des jeweiligen MDD entnommen werden (Kapitel 10).

pCbFunc Adresse einer Anwender-Callback-Routine. Einige Devices in einigen MDD´s können von sich aus das Anwenderprogramm über aufgetretene Ereignisse informieren (z.B. bei serieller Kommunikation beim Empfang eines Telegramms). Dieses geschieht, indem die hier angegebene Anwender-Funktion aufgerufen wird. Welche MDD-Devices diese Funktionalität unterstützen, kann aus der jeweiligen MDD-Dokumentation entnommen werden. Die Anwenderfunktion muss folgenden Aufbau haben (wird von VB nicht unterstützt):

Pascal
```
MAX_CALLBACK MyCallbackFunction
(handle: MAXCHLHND;
param: LONGWORD;
size: LONGWORD;
VAR pData);
```

C
```
MAX_CALLBACK MyCallbackFunction
(MAXCHLHND handle,
ULONG param,
ULONG size,
void *pData);
```

Beim Aufruf bekommt die Funktion das Handle des MDD-Kanals übergeben, der das Ereignis mitteilen möchte, den in *ulPar* beim Öffnen des Kanals übergebenen Wert, sowie einen Zeiger auf der Funktion übergebene Nutzdaten (deren Anzahl ebenso).[1] Die Callback-Routine darf keine Klassenfunktion sein.

ulPar Für Kanäle mit Callback-Funktionalität kann hier ein Parameter angegeben werden, der beim Aufruf der Callback-Anwender-Funktion übergeben wird.

hChannel Zeiger auf eine Variable, in die das Handle des Kanals eingetragen wird. Bei fehlerhafter Ausführung der Funktion ist hChannel = 0. Dieses Handle ist allen

[1] Für das Informieren eines Anwenderprogramms über ein aufgetretenes Ereignis greift der MDD auf den sogenannten Message-Modul-Device-Treiber zurück. Dies ist ein Hilfsprogramm mit dem Namen ‚mx_msg.exe‘ bzw. ‚mx_msg.sys‘, das für die Weiterleitung von Nachrichten von einem MDD an ein Anwenderprogramm benutzt wird. Der Message-MDD wird durch max_open_channel automatisch installiert, sobald die Funktion zum erstenmal mit pCbFunc != NULL aufgerufen wird. Bei der Verwendung in Echtzeitprogrammen ist zu beachten, dass der Message-MDD als Task auf dem CPU-Modul installiert sein muss. Das kann vom Host-PC aus durch Aufruf von max_load_mdd mit der Programmnummer 8FFFh (=MDD_X_MSG_MDD) geschehen oder über die Auto-Installation aus dem Flash: Installation durch max_load_mdd (hCPU, 0, 0, 0, 0x8FFF, NULL, &hMsgMdd);

Bibliotheksfunktionen zu übergeben, die auf den Kanal zugreifen.

Das erhaltene Handle ist Prozess-spezifisch, d.h. es darf nicht an andere Anwendungen weitergereicht werden. Andere Anwendungen müssen sich selbst ein Handle für das Device holen. !

Übertragen eines Prozess-spezifischen Kanal-Handles

Für solche Fälle, in denen die Konfiguration z.B. einer Messaufgabe nicht in einem Echtzeitprogramm sondern in einem PC-Programm vorgenommen werden soll, besteht das Problem, dass die bei max_open_channel erhaltenen Kanal-Handle Prozess-spezifisch sind, d.h. sie dürfen nicht einfach vom PC an ein Echtzeitprogramm übertragen werden, dass diese dann zum Lesen oder Beschreiben der Kanäle verwendet.

Zur Lösung dieser Problematik gibt es die beiden Funktionen max_export_channel_-handle und max_convert_channel_handle. Mit **max_export_channel_handle** wird die nicht-prozess-spezifische Kennung eines geöffneten Kanals sowie weitere auf der Empfängerseite zum Erzeugen eines eigenen Handles benötigte Daten an ein Echtzeitprogramm weitergegeben. Das Echtzeitprogramm muss als Task installiert sein und eine Funktion enthalten, die die Daten in Empfang nimmt (Aufbau: s.u.). max_export_channel_handle ruft diese Echtzeitfunktion auf und übergibt ihr die Daten.

Die Funktion im Echtzeitprogramm muss die empfangenen Daten an die Bibliotheksfunktion **max_convert_channel_handle** übergeben und erhält von dieser Funktion ein Handle, mit dem auf den Kanal zugegriffen werden kann.

Dieses Verfahren funktioniert nur dann, wenn das PC-Programm denselben MDD verwendet, auf den auch das Echtzeitprogramm zugreifen kann.

max_export_channel_handle Kanal-Handle exportieren

Pascal
```
FUNCTION max_export_channel_handle (hModul: MAXMODHND;
task, func: WORD; VAR hChannel: MAXCHLHND): MAX_ERROR;
```

C
```
MAX_ERROR max_export_channel_handle (MAXMODHND
hModul, USHORT task, USHORT func, MAXCHLHND *hChannel);
```

VB
```
Declare Function max_export_channel_handle Lib "maxw32.dll"
(ByVal hModul As Long, ByVal task As Integer, ByVal func As
Integer, ByRef hChannel As Long) As Integer
```

Verwendung Host-PC → MAX-PC OsX

Funktion Die Funktion exportiert eine nicht Prozess-spezifische Kennung zu dem übergebenen MDD-Kanal-Handle an ein Echtzeitprogramm. Damit es anschließend nicht zu Konflikten kommt, wenn der Kanal geschlossen wird, wird das Handle ungültig gemacht. In der Empfängerfunktion des Echtzeitprogramms liefert **max_convert_channel_handle** ein eigenes Handle für den Kanal.

Parameter

hModul Handle des MAX-PCs, auf das das übergebene Kanal-Handle exportiert werden soll.

task Tasknummer der Task, die die Empfängerfunktion enthält.

func Nummer der Empfängerfunktion entsprechend der PDT.

hChannel Zeiger auf eine Variable, die das Handle des MDD-Kanals enthält. Bei erfolgreicher Ausführung der Funktion wird das Handle = NULL gesetzt.

max_convert_channel_handle Kanal-Handle holen

Pascal FUNCTION max_convert_channel_handle (data: MAX_CHANNEL_TRANSFER_TYPE; VAR hChannel: MAXCHLHND): MAX_ERROR;

C MAX_ERROR max_convert_channel_handle (MAX_CHANNEL_TRANSFER_TYPE data, MAXCHLHND *hChannel);

VB Declare Function max_convert_channel_handle Lib "maxw32.dll" (ByRef data As MAX_CHANNEL_TRANSFER_TYPE, ByRef hChannel As Long) As Integer

Verwendung MAX-PC OsX

Funktion Die Funktion wandelt die nicht Prozess-spezifische Kennung eines im MDD geöffneten Kanals in ein Kanal-Handle um. Sie muss innerhalb der Funktion aufgerufen werden, die von max_export_channel_handle aufgerufen wird. Zu beachten ist, dass die Funktion sich immer auf das CPU-Modul bezieht, von dem aus sie aufgerufen wird (also das eigene CPU-Modul, dort muss sich auch der MDD befinden, zu dem der Kanal geöffnet worden ist.). Daher bekommt sie kein Handle für das CPU-Modul übergeben.

Parameter *data* Zeiger auf die Daten, die der Funktion, die von **max_export_channel_handle** aufgerufen wurde, übergeben worden sind. Die Daten innerhalb der Struktur dürfen nicht verändert werden, sondern müssen so an diese Funktion übergeben werden wie sie beim Aufruf der Echtzeitfunktion waren. Daher ist der Aufbau der Struktur hier nicht offengelegt.

hChannel Zeiger auf eine Variable, in die das Handle des MDD-Kanals eingetragen wird.

Beispiel einer Echtzeitfunktion, die durch max_export_channel_handle aufgerufen wird. Das erzeugte Handle wird im Parameterbereich der Task gespeichert.

C:

```
void MAXEXPORT ReceiveHandle (USHORT *pusInsize,
                              MAX_CHANNEL_TRANSFER_TYPE *pData,
                              USHORT *pusOutsize,
                              void *pDummy)
{
   MAX_ERROR Error;

   max_entry_func();

   /* get handle and store it in parameter range */
   Error=max_convert_channel_handle(pData,&parameter.hChannel1);

   max_exit_func(Error);
}
```

Pascal:

```
PROCEDURE ReceiveHandle (VAR pusInsize: WORD;
                         VAR pData: MAX_CHANNEL_TRANSFER_TYPE;
                         VAR pusOutsize: WORD; VAR pDummy); far;

VAR

  Error : MAX_ERROR;

BEGIN
  max_entry_func;
  { get a handle and store it in parameter range }
  Error           :=           max_convert_channel_handle(pData,
parameter.hChannel1);
  max_exit_func(Error);
END;
```

max_close_channel MDD-Kanal schließen

Pascal	FUNCTION max_close_channel (VAR hChannel: MAXCHLHND): MAX_ERROR;
C	MAX_ERROR max_close_channel (MAXCHLHND *hChannel);
VB	Declare Function max_close_channel Lib "maxw32.dll" (ByRef hChannel As Long) As Integer
Verwendung	Host-PC, MAX-PC OsX
Funktion	Diese Funktion schließt den angegebenen Kanal und macht das Handle ungültig. Das Schließen eines Kanals ist nicht unbedingt erforderlich. Es sollte aber gemacht werden, wenn ein Kanal nicht mehr benötigt wird, um den belegten Speicher wieder freizugeben. Keinesfalls sollte ein Kanal vor jedem Kanalzugriff geöffnet und hinterher wieder geschlossen werden!
Parameter	*hChannel* Zeiger auf eine Variable, die das Handle des Kanals enthält. Die Variable wird anschließend = 0 gesetzt.

max_read_channel_uchar
max_read_channel_short
max_read_channel_ushort
max_read_channel_long
max_read_channel_ulong

Lesen aus einem Kanal

Pascal

```
FUNCTION max_read_channel_uchar (hChannel: MAXCHLHND;
VAR data: BYTE): MAX_ERROR;

FUNCTION max_read_channel_short (hChannel: MAXCHLHND;
VAR data: SMALLINT): MAX_ERROR;

FUNCTION max_read_channel_ushort (hChannel: MAXCHLHND;
VAR data: WORD): MAX_ERROR;

FUNCTION max_read_channel_long (hChannel: MAXCHLHND;
VAR data: LONGINT): MAX_ERROR;

FUNCTION max_read_channel_ulong (hChannel: MAXCHLHND;
VAR data: LONGWORD): MAX_ERROR;
```

C

```
MAX_ERROR max_read_channel_uchar (MAXCHLHND hChannel,
UCHAR *data);

MAX_ERROR max_read_channel_short (MAXCHLHND hChannel,
SHORT *data);

MAX_ERROR max_read_channel_ushort (MAXCHLHND hChannel,
USHORT *data);

MAX_ERROR max_read_channel_long (MAXCHLHND hChannel,
LONG *data);

MAX_ERROR max_read_channel_ulong (MAXCHLHND hChannel,
ULONG *data);
```

VB

```
Declare Function max_read_channel_uchar Lib "maxw32.dll" (ByVal
hChannel As Long, ByRef data As Byte) As Integer

Declare Function max_read_channel_short Lib "maxw32.dll" (ByVal
hChannel As Long, ByRef data As Integer) As Integer

Declare Function max_read_channel_ushort Lib "maxw32.dll" (ByVal
hChannel As Long, ByRef data As Integer) As Integer

Declare Function max_read_channel_long Lib "maxw32.dll" (ByVal
hChannel As Long, ByRef data As Long) As Integer
```

Declare Function max_read_channel_ulong Lib "maxw32.dll" (ByVal hChannel As Long, ByRef data As Long) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Diese Funktionen lesen Daten von dem angegebenen Kanal. Welche der Funktionen vom angesprochenen MDD-Kanal unterstützt wird, steht in der jeweiligen MDD-Beschreibung oder kann mit **max_channel_info** (infotype = INFO_DATA_TYPE) abgefragt werden.

Parameter *hChannel* Handle des Kanals

data Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden

max_read_channel_block Lesen eines Blocks aus einem Kanal

Pascal FUNCTION max_read_channel_block (hChannel: MAXCHLHND; VAR size: LONGWORD; VAR data): MAX_ERROR;

C MAX_ERROR max_read_channel_block (MAXCHLHND hChannel, ULONG *size, void *data);

VB Declare Function max_read_channel_block Lib "maxw32.dll" (ByVal hChannel As Long, ByRef size As Long, ByRef data As Any) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Diese Funktion liest einen Block von Daten aus dem angegebenen Kanal. Die Anzahl wird in der Variablen *size* angegeben. Ob die Funktion vom angesprochenen MDD-Kanal unterstützt wird, steht in der jeweiligen MDD-Beschreibung oder kann mit **max_channel_info** (infotype = INFO_DATA_TYPE) abgefragt werden.

Parameter *hChannel* Handle des Kanals

size Zeiger auf eine Variable, in der die zu lesende Anzahl an Bytes steht. Nach der Rückkehr aus der Funktion enthält die Variable die Anzahl der tatsächlich gelesenen Bytes.

data Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden

max_trigger_channel — Trigger Kanal

Pascal FUNCTION max_trigger_channel (hChannel: MAXCHLHND): MAX_ERROR;

C MAX_ERROR max_trigger_channel (MAXCHLHND hChannel);

VB Declare Function max_trigger_channel Lib "maxw32.dll" (ByVal hChannel As Long) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Diese Funktion triggert einen Kanal, d.h. der Schreibdienst des MDD-Kanals wird aufgerufen. Dieser bekommt keine Daten übergeben.

Parameter *hChannel* Handle des Kanals

max_write_channel_uchar / max_write_channel_short / max_write_channel_ushort / max_write_channel_long / max_write_channel_ulong — Schreiben in einen Kanal

Pascal FUNCTION max_write_channel_uchar (hChannel: MAXCHLHND; data: BYTE): MAX_ERROR;

FUNCTION max_write_channel_short (hChannel: MAXCHLHND; data: SMALLINT): MAX_ERROR;

FUNCTION max_write_channel_ushort (hChannel: MAXCHLHND; data: WORD): MAX_ERROR;

FUNCTION max_write_channel_long (hChannel: MAXCHLHND; data: LONGINT): MAX_ERROR;

FUNCTION max_write_channel_ulong (hChannel: MAXCHLHND; data: LONGWORD): MAX_ERROR;

C MAX_ERROR max_write_channel_uchar (MAXCHLHND hChannel, UCHAR data);

MAX_ERROR max_write_channel_short (MAXCHLHND hChannel, SHORT data);

MAX_ERROR max_write_channel_ushort (MAXCHLHND hChannel, USHORT data);

MAX_ERROR max_write_channel_long (MAXCHLHND hChannel, LONG data);

MAX_ERROR max_write_channel_ulong (MAXCHLHND hChannel, ULONG data);

VB

Declare Function max_write_channel_uchar Lib "maxw32.dll" (ByVal hChannel As Long, ByVal data As Byte) As Integer

Declare Function max_write_channel_short Lib "maxw32.dll" (ByVal hChannel As Long, ByVal data As Integer) As Integer

Declare Function max_write_channel_ushort Lib "maxw32.dll" (ByVal hChannel As Long, ByVal data As Integer) As Integer

Declare Function max_write_channel_long Lib "maxw32.dll" (ByVal hChannel As Long, ByVal data As Long) As Integer

Declare Function max_write_channel_ulong Lib "maxw32.dll" (ByVal hChannel As Long, ByVal data As Long) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Diese Funktionen schreiben Daten in den angegebenen Kanal. Welche der Funktionen vom angesprochenen MDD-Kanal unterstützt wird, steht in der jeweiligen MDD-Beschreibung oder kann mit **max_channel_info** (infotype = INFO_DATA_TYPE) abgefragt werden.

Parameter *hChannel* Handle des Kanals

data Zu schreibendes Datum.

max_write_channel_block Schreiben eines Blocks an einen Kanal

Pascal FUNCTION max_write_channel_block (hChannel: MAXCHLHND; VAR size: LONGWORD; VAR data): MAX_ERROR;

C MAX_ERROR max_write_channel_block (MAXCHLHND hChannel, ULONG *size, void *data);

VB Declare Function max_write_channel_block Lib "maxw32.dll" (ByVal hChannel As Long, ByRef size As Long, ByRef data As Any) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Diese Funktion schreibt einen Block von Daten an den angegebenen Kanal. Die Anzahl wird in der Variablen *size* angegeben. Ob die Funktion vom angesprochenen MDD-Kanal unterstützt wird, steht in der jeweiligen MDD-Beschreibung oder kann mit **max_channel_info** abgefragt werden.

Parameter *hChannel* Handle des Kanals

size Zeiger auf eine Variable, in der die Anzahl an Bytes steht. Nach der Rückkehr aus der Funktion enthält die Variable die Anzahl der tatsächlich geschriebenen Bytes.

data Zeiger auf eine Variable, in die die zu schreibenden Daten eingetragen werden müssen.

max_channel_control Kanal-Steuerkommando

Pascal FUNCTION max_channel_control (hChannel: MAXCHLHND; cmd: WORD; size: WORD; VAR par): MAX_ERROR;

C MAX_ERROR max_channel_control (MAXCHLHND hChannel, USHORT cmd, USHORT size, void *par);

VB Declare Function max_channel_control Lib "maxw32.dll" (ByVal hChannel As Long, ByVal cmd As Integer, ByVal size As Integer, ByRef par As Any) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Mit dieser Funktion kann ein Steuerbefehl an einen MDD-Kanal gesendet werden.

Parameter *hChannel* Handle des Kanals

cmd Steuerbefehl. Befehle, die von allen Kanälen unterstützt werden, sind in der unten stehenden Tabelle aufgeführt. Erlaubt ein Kanal weitere Kommandos, ist das in der zugehörigen MDD-Beschreibung aufgeführt.

size Länge eventuell erforderlicher Parameterdaten. Die maximal mögliche Länge kann mit **max_get_max_blocksize** ermittelt werden.

par Für einige Steuerbefehle müssen zusätzlich zum Befehl weitere Parameter angegeben werden. In diesem Fall wird in diesem Parameter ein Zeiger auf die Übergabeparameter übergeben.

cmd	**Bedeutung**
CTRL_CAPTION	Text (als nullterminierten String mit *size*=Länge) in den Kanal eintragen
CMD_SUSP_WRITE	Datenausgabedienst sperren (*size*= 0)
CMD_SUSP_READ	Dateneingabedienst sperren (*size*= 0)
CMD_CONT_WRITE	Sperrung des Datenausgabedienstes aufheben (*size*= 0)
CMD_CONT_READ	Sperrung des Dateneingabedienstes aufheben (*size*= 0)

max_channel_info Kanal-Information

Pascal FUNCTION max_channel_info (hChannel: MAXCHLHND; infotype: WORD; VAR size: WORD; VAR data): MAX_ERROR;

C MAX_ERROR max_channel_info (MAXCHLHND hChannel, USHORT infotype, USHORT *size, void *data);

VB Declare Function max_channel_info Lib "maxw32.dll" (ByVal hChannel As Long, ByVal infotype As Integer, ByRef size As Integer, ByRef data As Any) As Integer

Verwendung Host-PC, MAX-PC OsX, Host-PC → MAX-PC OsX

Funktion Mit dieser Funktion können Informationen über einen Kanal abgefragt werden.

Parameter *hChannel* Handle des Kanals

infotype Angabe welche Information angefordert wird. Werte, die von allen Kanälen unterstützt werden, sind in der unten stehenden Tabelle aufgeführt. Bietet ein Kanal weitere Abfragemöglichkeiten, ist das in der zugehörigen MDD-Beschreibung aufgeführt.

size Zeiger auf eine Variable, in der die maximal zu lesende Anzahl an Bytes steht. Die maximal mögliche Länge kann mit **max_get_max_blocksize** ermittelt werden. Nach der Rückkehr aus der Funktion enthält die Variable die Anzahl der tatsächlich gelesenen Bytes.

data Zeiger auf eine Variable, in die Informationen eingetragen werden.

infotype	**Bedeutung der Antwortdaten (*data*)**
INFO_DATA_ TYPE	Datentyp, den der Kanal unterstützt (DATA_-VOID, DATA_UCHAR, DATA_SHORT, DATA_USHORT, DATA_LONG, DATA_-ULONG, DATA_TRIGGER, DATA_BLOCK) (size=4)
INFO_ TRANSFER_ MODE	Angabe, ob der Kanal gelesen und/oder geschrieben werden kann und ob der Kanal die Möglichkeit hat, Anwender-Callback-Funktions-aufrufe zu initialisieren. In der Antwort gibt Bit 0 an, ob der Kanal gelesen werden kann, Bit 1, ob der Kanal geschrieben werden kann und Bit 2, ob der Kanal Callback-fähig ist. (size=4)
INFO_MINSIZE, INFO_MAXSIZE	Minimale bzw. maximale Anzahl Datenbyte, die bei einem Kanalzugriff übergeben werden. (size=4)
INFO_DEVICE	Statusinformation zu dem mit dem Kanal verknüpften Device (s. MDD-Beschreibung) (size =4)
INFO_CAPTION	Zu einem Kanal eingetragener Text als nullterminierter String. (max. Länge steht in der CDT)

infotype	**Bedeutung der Antwortdaten (*data*)**
INFO_CPS	CPS-Datenstruktur, mit der der Kanal geöffnet wurde
INFO_CDT	CDT-Datenstruktur (Channel Descriptor Table) des Kanals
INFO_PAR	Zusätzliche Kanalparameter (ob und wie viele vorhanden sind steht in der CDT)
INFO_STATUS	Status des Kanals (Bitfeld) (size=4): Bit-0 =1: Kanal ist exklusiv geöffnet Bit-1 = 1: Daten werden unformatiert übertragen Bit-2 = 1: Daten werden ohne Gain-Offset-Korrektur übertragen Bit-16 = 1: Datenausgabedienst ist gesperrt Bit-17 = 1: Dateneingabedienst ist gesperrt

6.11. Ringpuffer-Funktionen

Das Betriebssystem OsX auf einem MAX-PC stellt bis zu 256 ringförmig organisierte Datenpuffer zur Verfügung. Die Puffer arbeiten byteorientiert nach dem FIFO-Prinzip, d.h. die zuerst in den Puffer geschriebenen Bytes werden auch als erste wieder ausgelesen. Ein Puffer kann mit der Funktion **max_create_buffer** im RAM des MAX-PC angelegt werden. Das Betriebssystem weist dem Puffer eine eindeutige Kennung zu. Diese kann mit **max_get_buffer_id** ausgelesen werden.

Alle Zugriffe auf die Puffer erfolgen über ein Handle, das entweder beim Anlegen des Puffers von **max_create_buffer** oder von der Funktion **max_get_buffer_handle** geliefert wird. Das Handle ist Prozess-spezifisch und darf deshalb nicht an andere Anwendungen (z.B. von einem PC-Programm an ein Echtzeitprogramm) weitergereicht werden. Wenn ein Puffer von zwei Anwendungen benutzt werden soll, muss die Anwendung, die den Puffer erzeugt hat, sich mit **max_get_buffer_id** die Pufferkennung holen. Diese kann dann eine andere Anwendung weitergereicht werden, die sich daraufhin mit **max_get_buffer_handle** ein eigenes Handle für den Puffer holen kann.

Die Puffer sind voll Multi-Tasking-fähig, d.h. sie dürfen gleichzeitig von mehreren Anwendungen angesprochen werden. Während eine Anwendung z.B. Daten in einen Puffer schreibt, kann eine andere Anwendung einen älteren Datenblock aus dem

Puffer auslesen. Gleichzeitiges Beschreiben eines Puffers von zwei Anwendungen führt dagegen zu einer Fehlermeldung. In diesem Fall muss die Anwendung, die die Fehlermeldung erhalten hat, den Schreibversuch später wiederholen. Gleiches gilt für gleichzeitiges Lesen von zwei Anwendungen.

max_create_buffer — Puffer erzeugen

Pascal: FUNCTION max_create_buffer (hModul: MAXMODHND; VAR par: MAX_BUFFER_TYPE; VAR size: LONGWORD; VAR pBufHandle: MAXBUFHND): MAX_ERROR;

C: MAX_ERROR max_create_buffer (MAXMODHND hModul, MAX_BUFFER_TYPE *par, ULONG *size, MAXBUFHND *pBufHandle);

VB: Declare Function max_create_buffer Lib "maxw32.dll" (ByVal hModul As Long, ByRef par As MAX_BUFFER_TYPE, ByRef size As Long, ByRef pBufHandle As Long) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Mit dieser Funktion wird ein ringförmig organisierter FIFO-Datenpuffer im RAM des MAX-PC angelegt. Die Verwaltung des Puffers wird komplett vom Betriebssystem OsX übernommen. Es stellt auch sicher, dass es keine Zugriffskonflikte gibt. Die Funktion liefert ein Handle zurück, mit dem anschließend auf den Puffer zugegriffen wird.

Parameter:

hModul Handle des MAX-PC

par Zeiger auf eine Datenstruktur vom Typ MAX_BUFFER_TYPE, in der die gewünschten Eigenschaften des Ringpuffers stehen. Die Struktur MAX_BUFFER_TYPE ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
usStrategy	USHORT	Strategie der Speicherreservierung. Die möglichen Werte sind in der unten stehenden Tabelle aufgeführt
usAlignment	USHORT	Ausrichtung des Anfangs des Puffers in Potenzen zu 2 (0 =

Element	Datentyp	Beschreibung
		Byte, 1 = Wort, 2 = Doppelwort, ...). Unter Geschwindigkeitsaspekten sollte mindestens 2 gewählt werden.

usTask	USHORT	Nummer der Task, die den Puffer anfordert (nur zu Protokollzwecken)
usUsage	USHORT	Reserviert (= 0 setzen)

size Zeiger auf eine Variable, in der die gewünschte Größe des Puffers steht. Im Anschluss an den Aufruf wird dort die tatsächliche Größe des Puffers eingetragen.

pBufHandle Zeiger auf eine Variable, in die ein Handle eingetragen wird, über das auf den Puffer zugegriffen wird.

Mögliche Werte für die Strategie der Puffererzeugung:

MAX_ALLOC_UP_ABS	Speicher von unten reservieren. Wenn nicht genügend Speicher zur Verfügung steht, wird ein Fehler gemeldet
MAX_ALLOC_UP_MAX	Speicher von unten reservieren. Wenn nicht genügend Speicher zur Verfügung steht, wird so viel wie möglich reserviert.
MAX_ALLOC_DOWN_ABS	Speicher von oben reservieren. Wenn nicht genügend Speicher zur Verfügung steht, wird ein Fehler gemeldet.
MAX_ALLOC_DOWN_MAX	Speicher von oben reservieren. Wenn nicht genügend Speicher zur Verfügung steht, wird so viel wie möglich reserviert.

Das erhaltene Handle ist Prozess-spezifisch, d.h. es darf nicht an andere Anwendungen weitergereicht werden. Andere Anwendungen müssen sich selbst ein Handle für den Puffer holen.

max_get_buffer_id Puffer-Kennung

Pascal FUNCTION max_get_buffer_id (hBuffer: MAXBUFHND; VAR id: LONGWORD): MAX_ERROR;

C MAX_ERROR max_get_buffer_id (MAXBUFHND hBuffer, ULONG *id);

VB Declare Function max_get_buffer_id Lib "maxw32.dll" (ByVal hBuffer As Long, ByRef id As Long) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktion liefert die Kennung eines Puffers. Diese Kennung kann im Gegensatz zum Pufferhandle an andere Prozesse weitergereicht werden, damit diese sich ebenfalls ein Handle für den Puffer holen können.

Parameter *hBuffer* Handle des Puffers

id Zeiger auf eine Variable, in die die auf dem entsprechenden MAX-PC eindeutige Kennung für den Puffer eingetragen wird.

max_get_buffer_handle — Puffer-Handle

Pascal: FUNCTION max_get_buffer_handle (hModul: MAXMODHND; id: LONGWORD; VAR pBufHandle: MAXBUFHND): MAX_ERROR;

C: MAX_ERROR max_get_buffer_handle (MAXMODHND hModul, ULONG id, MAXBUFHND *pBufHandle);

VB: Declare Function max_get_buffer_handle Lib "maxw32.dll" (ByVal hModul As Long, ByVal id As Long, ByRef pBufHandle As Long) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Mit dieser Funktion kann sich eine Anwendung ein Handle auf einen bestehenden Puffer holen, mit dem anschließend auf den Puffer zugegriffen wird.

Parameter:

hModul Handle des MAX-PC

id Eindeutige Kennung für den Puffer auf dem entsprechenden CPU-Modul.

pBufHandle Zeiger auf eine Variable, in die ein Handle eingetragen wird, über das auf den Puffer zugegriffen wird.

! *Das erhaltene Handle ist Prozess-spezifisch, d.h. es darf nicht an andere Anwendungen weitergereicht werden. Andere Anwendungen müssen sich selbst ein Handle für den Puffer holen.*

max_get_buffer_status — Pufferstatus

Pascal: FUNCTION max_get_buffer_status (hBuffer: MAXBUFHND; VAR valid: LONGWORD; VAR free: LONGWORD): MAX_ERROR;

C: MAX_ERROR max_get_buffer_status (MAXBUFHND hBuffer, ULONG *valid, ULONG *free);

VB: Declare Function max_get_buffer_status Lib "maxw32.dll" (ByVal hBuffer As Long, ByRef valid As Long, ByRef free As Long) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Diese Funktion liefert den Füllstatus eines Puffers.

Parameter	*hBuffer*	Handle des Puffers
	valid	Zeiger auf eine Variable, in die eingetragen wird, wie viele Bytes im Puffer benutzt sind.
	free	Zeiger auf eine Variable, in die eingetragen wird, wie viele Bytes im Puffer frei sind.

max_clear_buffer — Pufferinhalt löschen

Pascal	FUNCTION max_clear_buffer (hBuffer: MAXBUFHND): MAX_ERROR;
C	MAX_ERROR max_clear_buffer (MAXBUFHND hBuffer);
VB	Declare Function max_clear_buffer Lib "maxw32.dll" (ByVal hBuffer As Long) As Integer
Verwendung	Host-PC → MAX-PC OsX, MAX-PC OsX
Funktion	Mit dieser Funktion wird der Inhalt des Puffers gelöscht.
Parameter	*hBuffer* Handle des Puffers

max_get_buffer max_set_buffer — Pufferzugriff

Pascal	FUNCTION max_get_buffer (hBuffer: MAXBUFHND; strategy: WORD; VAR size: LONGWORD; VAR data: LONGWORD): MAX_ERROR; FUNCTION max_set_buffer (hBuffer: MAXBUFHND; strategy: WORD; VAR size: LONGWORD; VAR data: LONGWORD): MAX_ERROR;

C MAX_ERROR max_get_buffer (MAXBUFHND hBuffer, USHORT strategy, ULONG *size, ULONG *data);

MAX_ERROR max_set_buffer (MAXBUFHND hBuffer, USHORT strategy, ULONG *size, ULONG *data);

VB Declare Function max_get_buffer Lib "maxw32.dll" (ByVal hBuffer As Long, ByVal strategy As Integer, ByRef size As Long, ByRef data As Any) As Integer

Declare Function max_set_buffer Lib "maxw32.dll" (ByVal hBuffer As Long, ByVal strategy As Integer, ByRef size As Long, ByRef data As Any) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit diesen Funktionen werden Daten aus einem Puffer gelesen bzw. in einen Puffer hinein geschrieben.

Parameter *hBuffer* Handle des Puffers

strategy Strategie des Zugriffs. Folgende Werte sind möglich:
MAX_BUFFER_MOVE_ABS: Es wird versucht genau die angegebene Anzahl Bytes zu schreiben bzw. zu lesen. Gelingt das nicht, wird nichts gelesen bzw. geschrieben, sondern mit einer Fehlermeldung reagiert. Beim Lesen werden die gelesenen Daten aus dem Puffer entfernt.
MAX_BUFFER_MOVE_MAX: Wenn nicht so viele Bytes wie gefordert gelesen bzw. geschrieben werden können, wird die maximal mögliche Anzahl Bytes gelesen bzw. geschrieben. Beim Lesen werden die gelesenen Daten aus dem Puffer entfernt.
MAX_BUFFER_COPY_ABS (nur für **max_get_buffer**): wie MAX_BUFFER_MOVE_ABS, die Daten werden jedoch aus dem Puffer kopiert, d.h. die Daten bleiben im Puffer.
MAX_BUFFER_COPY_MAX (nur für **max_get_buffer**): wie MAX_BUFFER_MOVE_MAX, die Daten werden jedoch aus dem Puffer kopiert, d.h. die Daten bleiben im Puffer.

size Zeiger auf eine Variable, in der die Anzahl Bytes steht, die gelesen bzw. geschrieben werden sollen. Die Funktion trägt in die Variable ein, wie viele Bytes tatsächlich gelesen bzw. geschrieben worden sind. Bei

max_get_buffer mit *strategy* = MAX_BUFFER_COPY_ABS oder MAX_BUFFER_COPY_MAX ist die Länge auf die durch den von **max_get_max_blocksize** im Parameter *transmit* gelieferten Wert beschränkt.

data Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden, bzw. in der die zu schreibenden Daten stehen.

6.12. Service-Request Funktionen

Service-Requests sind ein Mechanismus, mit dem ein MAX-PC einem anderen MAX-PC oder dem Host-PC eine Nachricht schicken kann. Der MAX-PC sendet dazu ein Wort an den Adressaten. Die Bibliothek auf der Seite des Absenders stellt dafür die Funktion **max_send_srq** zur Verfügung.

Vor dem Aufruf von **max_send_srq** muss sich das Echtzeitprogramm zunächst mit **max_connect_cpu** ein Handle auf die Empfänger-CPU holen. Auf dem Ziel-Modul bzw. dem PC wird beim Empfang des SRQ-Wortes ein Interrupt ausgelöst. Auf dieser Seite stellt die Bibliothek die Funktion **max_set_service** zur Verfügung. Diese erlaubt die Angabe einer Anwenderprozedur, die dann von der Bibliothek aufgerufen wird, wenn ein SRQ-Wort empfangen wird (Callback-Mechanismus).

Der SRQ-Mechanismus wird auch vom Betriebssystem OsX auf den CPU-Modulen verwendet, um Fehler an andere CPU-Module oder den PC zu melden (sogenannte Error-Requests, ERQs) oder sonstige Ereignisse zu signalisieren. Die Unterscheidung zwischen Error-Requests und Anwender-definierten SRQs geschieht anhand des gesendeten SRQ-Wortes (Anhang B).

max_send_srq Service-Request senden

Pascal	FUNCTION max_send_srq (hModul: MAXMODHND; mode: WORD; message: WORD): MAX_ERROR;
C	MAX_ERROR max_send_srq (MAXMODHND hModul, USHORT mode, USHORT message);
VB	-
Verwendung	MAX-PC OsX → Host-PC, MAX-PC OsX → MAX-PC OsX
Funktion	Mit dieser Funktion wird ein Request an den angegebenen Host gesendet.
Parameter	*hModul* Handle des Moduls, das den SRQ empfangen soll
	mode Reserviert (zur Zeit =0 setzen)
	message Zu sendendes Wort. Das High-Byte steht zur freien Verfügung. Im Low-Byte stehen nur die Werte von 80h bis BFh zur freien Verfügung. Andere Werte sind für Fehlermeldungen des Betriebssystems OsX reserviert (s. Anhang B).

max_set_service Callback-Routine installieren

Pascal FUNCTION max_set_service (hModul: MAXMODHND; service: MAX_CALLBACK_TYPE): MAX_ERROR;

C MAX_ERROR max_set_service (MAXMODHND hModul, MAX_CALLBACK_TYPE *service);

VB -

Verwendung Host-PC

Funktion Mit dieser Funktion wird der Bibliothek die Adresse einer Anwender-Callback-Funktion übergeben. Die Bibliothek ruft diese Funktion auf, sobald sie vom angegebenen Modul einen User-Service-Request oder einen Error-Request empfängt.

Die Callback-Funktion darf keine Methode einer Klasse sein!

Unter den derzeitigen Versionen von Visual Basic funktioniert der Callback-Mechanismus nicht! Der Grund ist, dass Basic nicht Multi-Threading-fähig ist. Diese Funktion ist daher nicht für Basic implementiert. Soll unter Windows mit Service-Requests gearbeitet werden, müssen Sie daher zur Zeit die C++ oder Delphi Compiler verwenden.

Die Funktion muss folgende Deklaration besitzen:

Pascal PROCEDURE MyService (hModul: MAXMODHND; message: LONGWORD);

C MAX_CALLBACK MyService (MAXMODHND hModul, ULONG message);

Wenn ein Request vom entsprechenden Modul empfangen wird, wird der Funktion bei ihrem Aufruf im Parameter *hModul* das Handle des Moduls übergeben. Damit ist es möglich, dieselbe Service-Prozedur für mehrere CPU-Module zu verwenden. Im Parameter *message* steht das empfangene SRQ-Wort. Steht im Low-Byte von *message* ein Wert von 80h bis BFh, handelt es sich um einen Service-Request, andere Werte geben Fehler- oder sonstige Meldungen des Betriebssystems OsX an (s. Anhang B).

Parameter *hModul* Handle des Moduls, von dem SRQs empfangen werden sollen.

service Zeiger auf die Anwenderfunktion.

6.13. Download und Installation von Echtzeitprogrammen

max_transfer_and_install — Download und Installation eines Echtzeitprogramms

Pascal
FUNCTION max_transfer_and_install (hModul: MAXMODHND; VAR program: MAX_PGM_TYPE, var task: WORD): MAX_ERROR;

C
MAX_ERROR max_transfer_and_install (MAXMODHND hModul, MAX_PGM_TYPE *program, USHORT *task);

VB
Declare Function max_transfer_and_install Lib "maxw32.dll" (ByVal hModul As Long, ByRef program As MAX_PGM_TYPE, ByRef task As Integer) As Integer

Verwendung
Host-PC → MAX-PC OsX

Funktion
Diese Funktion lädt ein als Datei vorliegendes Echtzeitprogramm in das RAM des MAX-PC und installiert das Programm als Task unter dem Betriebssystem OsX. Die Funktion ist nicht in der Echtzeitbibliothek enthalten.

Parameter

hModul Handle auf das CPU-Modul

program Zeiger auf eine Variable vom Typ MAX-PGM_TYPE, in der das Programm spezifiziert ist. Die Struktur MAX_PGM_TYPE hat folgenden Aufbau:

Element	Datentyp	Beschreibung
szName	char*	Dateiname des Programms als nullterminierter String. Programme, die in C oder Pascal geschrieben sind, müssen die Namenserweiterung „.EXE“ haben, Assembler-Programme „.LIB“.
usNo	USHORT	In der PDT (Programm Deskriptor Tabelle) eingetragene Nummer des Programms Programm-Nummern ab 8000h

Element	Datentyp	Beschreibung
		sind für SORCUS reserviert.
usUsage	USHORT	reserviert (= 0 setzen)
usTask	USHORT	Tasknummer, unter der das Programm installiert werden soll. Erlaubte Werte sind 1 bis 1023. Die Nummer hat keinen Einfluss auf die Priorität des Programms.
usTaskType	USHORT	Tasktyp als der das Programm installiert werden soll, mögliche Werte sind: - MAX_NI_TASK, - MAX_II_TASK - MAX_TI_TASK - MAX_TM_TASK.
usLevel	USHORT	Privilegstufe des Programms (3=niedrigste bis 0=höchste)
usIrq	USHORT	Nummer des lokalen Interrupts, unter dem die Task installiert werden soll (bei Nicht-Interrupt-Tasks = 0 setzen)
usFlags	USHORT	Installations-Flags, Erklärung siehe Tabelle unten
ulDataSize	ULONG	Länge des Datenbereiches

task Zeiger auf eine Variable, in die die Tasknummer eingetragen wird. Wenn im Element *usTask* in der übergebenen Datenstruktur ein gültiger Wert übergeben wird, wird dieser in *task* zurückgegeben.

Bedeutung der Bits im Element usFlags:

Flag	Bedeutung dieses Flag
MAX_CALL_AUTO_INIT	Wenn dieses Flag gesetzt ist, wird automatisch die auto_init-Prozedur aufgerufen.
MAX_AUTO_ACTIVATE	Wenn dieses Flag gesetzt ist, wird die Task nach dem Installieren automatisch aktiviert.

Flag	Bedeutung dieses Flag
MAX_TASKTYPE_BY_ INSTALL	Wenn dieses Flag gesetzt ist, werden der Tasktyp und die Interruptnummer durch die Einträge in der PDT des Echtzeitprogramms festgelegt, ansonsten durch die Parameter der Funktion **max_transfer_and_install**. Das Flag ist nur wirksam, wenn Bit 3 in den Flags der PDT = 0 ist.
MAX_DATASIZE_IN_ PDT	Wenn dieses Flag gesetzt ist, wird die Datenbereichsgröße der Task durch den in *ulDataSize* in der PDT des Echtzeitprogramms angegebenen Wert festgelegt, ansonsten durch den Parameter *ulDataSiz*e der Funktion **max_transfer_and_install**. Das Flag ist nur wirksam, wenn Bit 9 in den Flags der PDT = 0 ist.

max_transfer_program Download eines Echtzeitprogramms

Pascal FUNCTION max_transfer_program (hModul: MAXMODHND; const filename: PChar; prognr: WORD; VAR address: LONGWORD): MAX_ERROR;

C MAX_ERROR max_transfer_program (MAXMODHND hModul, char *filename, USHORT prognr, ULONG *address);

VB Declare Function max_transfer_program Lib "maxw32.dll" (ByVal hModul As Long, ByVal filename As String, ByVal prognr As Integer, ByRef address As Long) As Integer

Verwendung Host-PC → MAX-PC OsX

Funktion Diese Funktion lädt ein als Datei vorliegendes Echtzeitprogramm in das RAM des MAX-PC. Um das Programm anschließend als Task unter dem Betriebssystem OsX zu installieren, muss noch die Funktion **max_install_task** aufgerufen werden.

Die Aufteilung der Funktion **max_transfer_and_install** ist nur dann erforderlich, wenn der Programmcode nur einmal in das RAM des MAX-PC geladen werden soll, jedoch mehrere Instanzen davon als OsX-Task installiert werden sollen (**Mehrfachinstallation**). Dabei befindet sich der Programmcode und die PDT nur einmal im Speicher, jede Instanz des Programms hat jedoch ihren eigenen Daten- und

Parameterbereich sowie ihre eigene TDT. Der Sinn der Mehrfachinstallation liegt im geringeren Speicherbedarf.

Damit ein Echtzeitprogramm mehrfach installierbar ist, müssen folgende Punkte beachtet werden:

- In den PDT-Flags (im Element *usFlags* der Struktur MAX_PDT_HEAD) dürfen die Flags MAXPDT_DATA_IN_PROG und MAXPDT_PARAMETER_IN_PROG *nicht* angegeben sein. Dadurch reserviert das Betriebssystem den Parameter- und Datenbereich.
- Das Programm darf dann nur mit den Bibliotheksfunktionen auf den eigenen Parameter- und Datenbereich zugreifen.
- Die Tasknummer der entsprechenden Instanz des Programms kann bei Bedarf aus der TDT mit **max_task_info** ausgelesen werden.

Parameter *hModul* Handle auf das CPU-Modul

filename Zeiger auf einen nullterminierten String, der den zu ladenden Dateinamen angibt

prognr In der PDT (Programm Deskriptor Tabelle) eingetragene Nummer des Programms

address Zeiger auf eine Variable, in die die Startadresse des Programms eingetragen wird. Dieser Wert ist anschließend der Funktion **max_install_task** zu übergeben.

max_install_task OsX-Task installieren

Pascal
```
FUNCTION max_install_task (hModul: MAXMODHND;
VAR program: MAX_PGM_TYPE; address: LONGWORD; VAR
task: WORD): MAX_ERROR;
```

C
```
MAX_ERROR max_install_task (MAXMODHND hModul,
MAX_PGM_TYPE *program, ULONG address, USHORT *task);
```

VB
```
Declare Function max_install_task Lib "maxw32.dll" (ByVal hModul
As Long, ByRef program As MAX_PGM_TYPE, ByVal address As
Long, ByRef task As Integer) As Integer
```

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktion installiert ein im RAM des MAX-PC vorliegendes Echtzeitprogramm als Task unter dem Betriebssystem OsX.

Die Funktion muss nur dann verwendet werden, wenn das Programm mit **max_transfer_program** geladen worden ist.

Parameter

hModul Handle auf das CPU-Modul

program Zeiger auf eine Variable vom Typ MAX-PGM_TYPE, in der das Programm spezifiziert ist. Die Struktur MAX_PGM_TYPE ist bei der Funktion **max_transfer_and_install** erläutert.

address Startadresse des Programms, die von der Funktion **max_transfer_program** geliefert worden ist.

task Zeiger auf eine Variable, in die die Tasknummer eingetragen wird.

max_init_program Anmelden eines Echtzeitprogramms

Pascal FUNCTION max_init_program (pdt: POINTER): MAX_ERROR;

C MAX_ERROR max_init_program (MAX_PDT_HEAD *pdt);

VB -

Verwendung MAX-PC OsX

Funktion

Mit dieser Funktion meldet sich ein Programm beim Betriebssystem OsX als Task an. Dazu wird dem Betriebssystem die Adresse der ausgefüllten PDT (Programm Deskriptor Tabelle) des Echtzeitprogramms übergeben. Das Betriebssystem entnimmt daraus die für die Installation des Programms als Task benötigten Informationen.
Die PDT ist Teil des Programms, das unter einer Task installiert wird, und muss von diesem Programm initialisiert werden. Sie enthält Informationen, die das Betriebssystem über das Programm wissen muss, wie z.B. die Anzahl und die Anfangsadressen der Prozeduren und Funktionen, die Größe des Daten- und Parameterbereiches, die Programmnummer (mitsamt Versionsangabe). Die Struktur ist in der Bibliothek unter der Bezeichnung MAX_PDT_HEAD definiert. Zusätzlich sind direkt dahinter Zeiger auf die Taskprozeduren und Funktionen zu definieren (s. Beispiel).

Andere für die Installation benötigte Informationen werden dem OsX erst durch den Installationsbefehl mitgeteilt, wie z.B. die Tasknummer. Der Tasktyp, die Größe des Datenbereiches sowie bei II-Tasks die Interruptnummer können entweder bei der Installation einer Task aus der PDT des Programms entnommen werden oder dem OsX durch **max_transfer_and_install** mitgeteilt werden. Welche der beiden Möglichkeiten verwendet wird, kann durch Flags in der PDT und bei **max_transfer_and_install** festgelegt werden.

Die PDT ist im Programm als globale Variable zu deklarieren und auszufüllen (s. Beispiel). Die Angaben in der PDT sind statisch, d.h. sie werden weder vom Betriebssystem verändert noch darf das Echtzeitprogramm sie zur Laufzeit ändern.

Die Funktion muss im Prepare-Teil (main-Routine = Einsprungpunkt des Programms) des Echtzeitprogramms aufgerufen werden, nachdem die PDT ausgefüllt ist.

Parameter

pdt Zeiger auf den Anfang der PDT des Echtzeitprogramms.

Beispiel für die main-Routine eines Echtzeitprogramms:

```
// Deklaration des Parameterbereiches
struct parameter_type {
 MAX_TDT_TYPE rcTDT;         // Task-Deskriptor-Tabelle (steht immer direkt vor den Parametern)
                             //... (bis zu 64kByte eigene Parameter)
} parameter;

// Deklaration der Program Deskriptor Tabelle (PDT) (im Beispiel mit zwei eigenen Prozeduren)
struct pdt_type {
 MAX_PDT_HEAD  rcHead;       // PDT-Header
 void* MAXEXPORT main_proc;  // Adr. d. Main-Prozedur (= Prozedur 0)
 void* MAXEXPORT auto_init;  // Adr. d. Init-Prozedur (= Prozedur 1)
 void* MAXEXPORT start;      // Adr. d. Start-Prozedur (optional) (= Prozedur 2)
 void* MAXEXPORT stop;       // Adr. d. Stop-Prozedur (optional) (= Prozedur 3)
} rcPDT;

void main ()  // Prepare Teil des Programmes
{
 rcPDT.rcHead.ucType         = 1;
 rcPDT.rcHead.ucSize         = sizeof(MAX_PDT_HEAD);
 rcPDT.rcHead.usProcedures   = (sizeof(rcPDT)-sizeof(MAX_PDT_HEAD))/4;// Anzahl der Prozeduren
 rcPDT.rcHead.usProgNo       = 0x300;// Programm-Nummer
 rcPDT.rcHead.cVersion       = ´1´;  // Programm-Version
 rcPDT.rcHead.cRevision      = ´A´;  // Programm-Revision
 rcPDT.rcHead.ucCPU          = MAX_CPU486;
 rcPDT.rcHead.ucCoProc       = NOCOPROC;
 rcPDT.rcHead.ucLanguage     = MAXRT_LANGUAGE_C;
 rcPDT.rcHead.ucProgType     = 0;
 rcPDT.rcHead.usFlags        = MAX_NI_TASK + MAXPDT_PARAMETER_IN_PROG;
 rcPDT.rcHead.usInt          = 0;    // Interrupt-Nummer
 rcPDT.rcHead.ulDataAdr      = 0;    // Adresse des Datenbereichs
                                     (nur, wenn das Programm selbst den Datenbereich festlegt)
 rcPDT.rcHead.ulDataSize     = 0;    // Größe des Datenbereichs
 rcPDT.rcHead.ulRes1         = 0;    // zur Zeit nicht verwendet
 rcPDT.rcHead.ulRes2         = 0;    // zur Zeit nicht verwendet
 rcPDT.rcHead.ulHyperAdr     = 0;    // zur Zeit nicht verwendet
 // physikalische Anfangsadresse des Parameterbereichs
 rcPDT.rcHead.ulParAdr       = max_get_phys_addr(&parameter) + sizeof(MAX_TDT_TYPE);
 // Anzahl der Parameter
 rcPDT.rcHead.usParSize      = sizeof(parameter) -  sizeof(MAX_TDT_TYPE);
 // Adressen der Taskprozeduren bzw. Funktionen eintragen
 rcPDT.main_proc             = main_task;
 rcPDT.auto_init             = auto_init;
 rcPDT.start                 = start;
 rcPDT.stop                  = stop;
 max_init_program ((MAX_PDT_HEAD*)&rcPDT);
}
```

PDT-Flags

Im Element *usFlags* der Datenstruktur MAX_PDT_HEAD werden die folgenden Eigenschaften des Programms festgelegt:

Bit	Bedeutung
0..2	Gibt den Tasktyp an. Als Konstanten stehen MAX_NI_TASK, MAX_TI_TASK, MAX_II_TASK und MAX_TM_TASK zur Verfügung.

Bit	Bedeutung
3	Ist dieses Flag =1 gesetzt, werden der Tasktyp und die Interrupt-Nummer für die Task, unter der das Programm installiert wird, auf jeden Fall aus den Angaben in der PDT (*usInterrupt* und *usFlags*, Bit 0 bis 2) entnommen, ansonsten können Tasktyp und Interrupt-Nummer beim Installieren festgelegt werden. Konstante: MAXPDT_TASKTYPE_IN_PDT.
4..7	Reserviert, = 0 setzen.
8	Ist dieses Flag =1 gesetzt, wird der Datenbereich vom Programm selbst angelegt (als globale Variable im Programm). Der Parameter *ulDataAdr* in der MAX_PDT_HEAD Datenstruktur muss in diesem Fall mit der physikalischen Anfangsadresse der globalen Variablen ausgefüllt werden. Dieses geschieht genau wie im vorstehenden Beispiel für die Initialisierung der Parameterbereich-Anfangsadresse (*rcPDT.rcHead.ulParAdr*) gezeigt. Empfohlen wird, dieses Flag = 0 zu setzen. Dann legt das Betriebssystem den Datenbereich an. Das Programm kann dann nur mit den Bibliotheksfunktionen auf den eigenen Datenbereich zugreifen. Der Parameter *ulDataAdr* in der MAX_PDT_HEAD Datenstruktur braucht nicht ausgefüllt zu werden. Konstante: MAXPDT_DATA_IN_PROG.
9	Ist dieses Flag =1 gesetzt, wird die Größe des Datenbereiches durch den Parameter *ulDataSize* in der MAX_PDT_HEAD Datenstruktur festgelegt, andernfalls wird die Größe des Datenbereiches beim Installieren angegeben. Konstante: MAXPDT_FIXED_DATASIZE.
10	Ist dieses Flag =1 gesetzt, wird der Parameterbereich vom Programm selbst angelegt (als globale Variable im Programm). Der Parameter *ulParAdr* in der MAX_PDT_HEAD Datenstruktur muss in diesem Fall mit der physikalischen Anfangsadresse der globalen Variablen ausgefüllt werden (s. Beispiel oben). Das Programm kann auf den eigenen Parameterbereich direkt wie auf jede andere globale Variable zugreifen. Ist dieses Flag = 0 gesetzt, legt das Betriebssystem den Parameterbereich an. Das Programm kann dann nur mit den Bibliotheksfunktionen auf den eigenen Parameterbereich zugreifen. Der Parameter *ulParAdr* in der MAX_PDT_HEAD Datenstruktur braucht nicht ausgefüllt zu werden. Konstante: MAXPDT_PARAMETER_IN_PROG.
11..15	reserviert, = 0 setzen

Das Setzen von mehreren Flags kann durch Addieren der angegebenen Konstanten erfolgen.

6.14. Taskverwaltung

max_get_prog_version — Version eines Echtzeitprogramms

Pascal
FUNCTION max_get_prog_version (hModul: MAXMODHND; task: WORD; VAR version: MAX_VERSION; VAR date: LONGWORD); MAX_ERROR;

C
MAX_ERROR max_get_prog_version (MAXMODHND hModul, USHORT task, MAX_VERSION *version, ULONG *date);

VB
Declare Function max_get_prog_version Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByRef version As Long, ByRef date As Long) As Integer

Verwendung
Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion
Diese Funktion liefert die in der PDT eingetragene Version und das Erstellungsdatum des unter einer Task installierten Echtzeitprogramms.

Parameter

hModul	Handle auf das CPU-Modul
task	Nummer der Task
version	Zeiger auf eine Variable, in die die Version eingetragen wird (Format s. Kapitel 6.3)
date	Zeiger auf eine Variable, in die das Erstellungsdatum eingetragen wird (Format s. Kapitel 6.3)

max_wakeup_ni_task / max_wakeup_ti_task / max_wakeup_ii_task — Task-Aktivierung

Pascal
FUNCTION max_wakeup_ni_task (hModul: MAXMODHND; task: WORD; option: WORD): MAX_ERROR;

FUNCTION max_wakeup_ti_task (hModul: MAXMODHND; task: WORD; VAR data: MAX_TI_TYPE): MAX_ERROR;

FUNCTION max_wakeup_ii_task (hModul: MAXMODHND; task: WORD): MAX_ERROR;

C

MAX_ERROR max_wakeup_ni_task (MAXMODHND hModul, USHORT task, USHORT option);

MAX_ERROR max_wakeup_ti_task (MAXMODHND hModul, USHORT task, MAX_TI_TYPE *data);

MAX_ERROR max_wakeup_ii_task (MAXMODHND hModul, USHORT task);

VB

Declare Function max_wakeup_ni_task Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal option As Integer) As Integer

Declare Function max_wakeup_ti_task Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByRef data As MAX_TI_TYPE) As Integer

Declare Function max_wakeup_ii_task Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktionen aktivieren eine Task.

Parameter

hModul Handle auf das CPU-Modul

task Nummer der zu aktivierenden Task

option Optionen für die Aktivierung einer **NI-Task**:

= 1: Task wird ab Aktivierung zyklisch aufgerufen.

= 2: Task kommt als nächste NI-Task an die Reihe. Wenn eine andere Task bereits vorgezogen werden soll und immer noch auf die Ausführung wartet, wird sie durch diese Task ersetzt. Wenn die Task vorher nicht aktiviert war, wird sie nur ein einziges Mal aufgerufen.

= 3: Wie 2, wenn bereits eine NI-Task vorgezogen, aber noch nicht begonnen wurde, wird ein Fehler geliefert.

data Zeiger auf eine Struktur vom Typ MAX_TI_TYPE mit Informationen zur Aktivierung einer TI-Task. Die Struktur hat folgenden Aufbau:

Element	Datentyp	Beschreibung
usPriority	USHORT	Priorität (0= höchste bis 255 = niedrigste)
ulHoldOff	ULONG	Zeit in Timer-Ticks nach der die TI-Task zum ersten Mal aufgerufen wird
ulIntervall	ULONG	Zeit in Timer-Ticks zwischen zwei Aufrufen
ulCalls	ULONG	Anzahl der Aufrufe bis die TI-Task automatisch wieder deaktiviert wird (0=TI-Task wird nicht automatisch deaktiviert)

max_sleep_task Task-Deaktivierung

Pascal FUNCTION max_sleep_task (hModul: MAXMODHND; task: WORD): MAX_ERROR;

C MAX_ERROR max_sleep_task (MAXMODHND hModul, USHORT task);

VB Declare Function max_sleep_task Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktion deaktiviert eine Task.

Parameter *hModul* Handle auf das CPU-Modul

task Nummer der zu deaktivierenden Task

max_get_task_status — Task-Status

Pascal: FUNCTION max_get_task_status (hModul: MAXMODHND; task: WORD; VAR activations: WORD): MAX_ERROR;

C: MAX_ERROR max_get_task_status (MAXMODHND hModul, USHORT task, USHORT *activations);

VB: Declare Function max_get_task_status Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByRef activations As Integer) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Mit dieser Funktion wird ermittelt, ob und wie oft eine Task aktiviert ist.

Parameter:

- *hModul* — Handle auf das CPU-Modul
- *task* — Nummer der Task
- *activations* — Zeiger auf eine Variable, in die die Anzahl der Aktivierungen der Task eingetragen wird. NI-Tasks können mehrfach aktiviert werden, TI- und Interrupt-Tasks nur einmal.

max_get_task_number_prog — Task-Nummer eines Programms ermitteln

Pascal: FUNCTION max_get_task_number_prog (hModul: MAXMODHND; prog: WORD; VAR instance: WORD, VAR task: WORD): MAX_ERROR;

C: MAX_ERROR max_get_task_number_prog (MAXMODHND hModul, USHORT prog, USHORT *instance, USHORT *task);

VB: Declare Function max_get_task_number_prog Lib "maxw32.dll" (ByVal hModul As Long, ByVal prog As Integer, ByRef instance As Integer, ByRef task As Integer) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Mit dieser Funktion kann die Tasknummer ermittelt werden, unter der ein Programm installiert ist, sowie die Anzahl der Installationen des Programms.

Parameter	*hModul*	Handle auf das CPU-Modul
	prog	In der PDT (Programm Deskriptor Tabelle) eingetragene Programmnummer.
	instance	Zeiger auf eine Variable, die die Instanz des Programms, zu dem die Tasknummer gesucht ist, angibt. Die Funktion liefert dann in *task* die Tasknummer, die sich, wenn man die Tasks ihren Nummern (0 bis Anzahl der Instanzen) nach ordnet, an der Stelle *instance* befindet. Ist ein Programm mehrfach z.B. unter den Tasknummern 7, 8 und 9 installiert, und wird die Funktion mit instance=1 aufgerufen, wird in *task* der Wert 8 eingetragen. Die Funktion trägt in die Variable ein, wie viele Instanzen des Programms als Tasks installiert sind.
!		*Dieser Wert ist auch dann gültig, wenn die Funktion den Fehler ERR_TASK_NOT_INSTALLED zurückliefert!*
	task	Zeiger auf eine Variable, in die die Tasknummer der Instanz des Programms eingetragen wird.

max_get_task_number_int — Ermitteln, welche Task unter einem Interrupt installiert ist

Pascal	FUNCTION max_get_task_number_int (hModul: MAXMODHND; intr: WORD, VAR task: WORD): MAX_ERROR;
C	MAX_ERROR max_get_task_number_int (MAXMODHND hModul, USHORT intr, USHORT *task);
VB	Declare Function max_get_task_number_int Lib "maxw32.dll" (ByVal hModul As Long, ByVal intr As Integer, ByRef task As Integer) As Integer
Verwendung	Host-PC → MAX-PC OsX, MAX-PC OsX
Funktion	Mit dieser Funktion kann ermittelt werden, welche Task unter einem bestimmten Interrupt installiert ist.

Parameter *hModul* Handle auf das CPU-Modul

intr Interruptnummer, zu der die Tasknummer gesucht werden soll. Mögliche Werte: siehe Kapitel 6.18 und Kapitel 9.3.3.

task Zeiger auf eine Variable, in die die Nummer der Task eingetragen wird.

max_task_info Task-Informationen

Pascal FUNCTION max_task_info (hModul: MAXMODHND; task: WORD; infotype: WORD; VAR result: LONGWORD): MAX_ERROR;

C MAX_ERROR max_task_info (MAXMODHND hModul,USHORT task, USHORT infotype, ULONG *result);

VB Declare Function max_task_info Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal infoType As Integer, ByRef result As Long) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit dieser Funktion können Systeminformationen aus der PDT (Programm Deskriptor Tabelle) und TDT (Task Deskriptor Tabelle) einer Task abgefragt werden. Die Funktion bietet u.a. die Möglichkeit zu prüfen, unter welchen Tasknummern Programme installiert sind.

Ist die Task nicht installiert, liefert die Funktion den Fehler ERR_TASK_NOT_INSTALLED.

Parameter *hModul* Handle auf das CPU-Modul

task Nummer der Task

infotype Spezifikation der gewünschten Information. Die möglichen Werte sind in der Tabelle unten aufgelistet.

result Zeiger auf eine Variable, in die das Ergebnis eingetragen wird. Für einige Werte von *infotype* liefert die Funktion mehrere Informationen zusammengefasst. In der unten stehenden Tabelle kennzeichnet dann (1) das niederwertigste Byte, (4) ist das höchstwertigste Byte.

Für *infotype* definierte Werte:

infotype	**Angeforderte Information**
0	Typ der PDT (1), Länge des Vorspanns der PDT (2), Anzahl der in der PDT eingetragenen Prozeduren (3-4)
4	Programmnummer (1-2), Version (3) und Revision (4) des Programms
8	Prozessortyp (1), Co-Prozessor (2), Programmiersprache des Programms (3), Programmtyp (4)
12	PDT-Flags (1-2, s. Anhang D), Interrupt (3), (4) = 0
x+0	Anfangsadresse (Seg:Offs) der Main-Prozedur (Prozedur 0), x=Länge des PDT Vorspanns (infotype= 0)
x+4	Anfangsadresse (Seg:Offs) der Auto-Init-Prozedur (Prozedur 1), x=Länge des PDT Vorspanns (infotype= 0)
x+8	Anfangsadresse (Seg:Offs) der ersten Anwenderprozedur (Prozedur 2), x=Länge des PDT Vorspanns (infotype= 0)
x+n	Anfangsadresse (Seg:Offs) einer Anwenderprozedur bzw. –funktion n/4, x=Länge des PDT Vorspanns (infotype= 0)
256+0	Typ der TDT (1), Länge der TDT (2), Tasknummer (3-4)
256+4	TDT-Flags (1-2), Anzahl der Aktivierungen (für NI-Tasks) bzw. Interrupt, unter dem das Programm installiert ist (3), (4) = 0
256+8	Größe des Parameterbereiches in Bytes (1-2), Anzahl in der PDT eingetragener Prozeduren (3-4)
256+12	Physikalische Anfangsadresse der PDT
256+16	Physikalische Anfangsadresse des Hypertext
256+20	Physikalische Anfangsadresse des Datenbereiches
256+24	Physikalische Endadresse des Datenbereiches
512+0	Physikalische Anfangsadresse der TDT

Hinweis: Zur Ermittlung der **Anfangsadresse des Parameterbereiches** einer Task ist die Anfangsadresse der TDT zu ermitteln (infotype =512+0). Der Parameterbereich beginnt immer direkt nach der TDT (Größe der TDT kann mit infotype =256+0 ermittelt werden).

6.15. Daten- und Parameterbereich eines Echtzeitprogramms

Bei der Installation eines Programms unter einer Task bekommt diese Task einen Daten- und einen Parameterbereich zugewiesen. Das sind voneinander unabhängige lineare, durchgängige Bereiche im RAM-Speicher des MAX-PC, die für die Interprozess-Kommunikation genutzt werden können.

Die Größe dieser beiden Bereiche wird vom Programm selbst festgelegt (in der PDT). Die Größe des Datenbereiches kann alternativ auch beim Installieren angegeben werden. Die Bereiche sind nach dem Installieren nicht in ihrer Größe veränderbar. Ihre Größe kann auch = 0 sein.

Der Parameterbereich dient üblicherweise zur Aufnahme von Konfigurationsdaten des Programms, z.B. Grenzwerte, Abtastrate, Baudrate usw. Seine Länge ist auf 65280 Bytes beschränkt. Der Datenbereich ist für die Aufnahme von fortlaufenden Daten (z.B. Messdaten) gedacht.

Andere Tasks, sowie andere CPU-Module oder auch der PC können mittels der nachfolgenden Bibliotheksfunktionen über das OsX auf die beiden Bereiche zugreifen. Die Zugriffe erfolgen unter Angabe der Tasknummer und des Offsets (in Bytes) bezogen auf den Anfang des jeweiligen Bereichs.

Die Task kann auf ihren eigenen Parameter- bzw. Datenbereich auch direkt zugreifen, wenn es ihn selbst anlegt. Dazu müssen in der PDT des Programms die Flags MAXPDT_PARAMETER_IN_PROG bzw. MAXPDT_DATA_IN_PROG gesetzt sein und die Anfangsadressen der Bereiche ebenfalls in der PDT eingetragen werden. In diesem Fall sind der Datenbereich bzw. der Parameterbereich als globale Variable (in der Regel vom Typ struct bzw. Record) innerhalb des Echtzeitprogramms anzulegen, auf die wie auf jede andere globale Variable zugegriffen werden kann. Die direkten Zugriffe sind auf den Speicherbereich unterhalb von 1MB beschränkt. Die Lage der Bereiche kann mit **max_task_info** überprüft werden. Für den Datenbereich wird empfohlen, ihn vom Betriebssystem anlegen zu lassen und nur über die Bibliotheksfunktion darauf zuzugreifen.

max_get_data_sema Datenbereichs-Semaphore
max_release_data_sema

Pascal FUNCTION max_get_data_sema (hModul: MAXMODHND; task: WORD): MAX_ERROR;

FUNCTION max_release_data_sema (hModul: MAXMODHND; task: WORD): MAX_ERROR;

C MAX_ERROR max_get_data_sema (MAXMODHND hModul, USHORT task);

MAX_ERROR max_release_data_sema (MAXMODHND hModul, USHORT task);

VB Declare Function max_get_data_sema Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer) As Integer

Declare Function max_release_data_sema Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit diesen Funktionen wird der Zugriff auf den Datenbereich einer Task gesteuert. Damit ein konsistenter Zugriff gewährleistet ist, muss ein Programm, das auf einen Datenbereich zugreifen will, zunächst versuchen, mit **max_get_data_sema** die Semaphore für den Datenbereich zu bekommen. Gelingt dies, kann das Programm beliebig auf den Datenbereich zugreifen. Hat sich bereits eine andere Anwendung die Semaphore geholt, liefert die Funktion eine Fehlermeldung. Mit **max_release_data_sema** wird die Semaphore wieder frei gegeben.

! *Das Betriebssystem prüft bei den Datenbereichszugriffen selbst nicht, ob die Semaphore frei ist. Um konsistente Daten sicherzustellen, müssen alle Anwendungen vor einem Zugriff auf den Datenbereich einer Task zunächst die Semaphore abprüfen!*

Parameter *hModul* Handle auf das CPU-Modul

task Nummer der Task

max_get_data max_set_data — Datenbereichszugriff

Pascal

FUNCTION max_get_data (hModul: MAXMODHND; task: WORD; offset: LONGWORD; VAR size: LONGWORD; VAR data): MAX_ERROR;

FUNCTION max_set_data (hModul: MAXMODHND; task: WORD; offset: LONGWORD; VAR size: LONGWORD; VAR data): MAX_ERROR;

C

MAX_ERROR max_get_data (MAXMODHND hModul, USHORT task, ULONG offset, ULONG *size, void *data);

MAX_ERROR max_set_data (MAXMODHND hModul, USHORT task, ULONG offset, ULONG *size, void *data);

VB

Declare Function max_get_data Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal offset As Long, ByRef size As Long, ByRef data As Any) As Integer

Declare Function max_set_data Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal offset As Long, ByRef size As Long, ByRef data As Any) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Mit diesen Funktionen wird auf den Datenbereich einer Task zugegriffen. Vor jedem Zugriff sollte mit **max_get_data_sema** die entsprechende Semaphore geholt werden, um einen konsistenten Zugriff zu gewährleisten.

Parameter

hModul Handle auf das CPU-Modul

task Nummer der Task

offset Offset (in Bytes) bezogen auf den Anfang des Datenbereiches, ab dem die Daten gelesen bzw. geschrieben werden sollen.

size Zeiger auf eine Variable, in der steht wie viele Bytes (maximal) gelesen bzw. geschrieben werden sollen. Nach Beendigung der Funktion ist dort eingetragen, wie viele Bytes tatsächlich gelesen bzw. geschrieben worden sind. Die Größe des Datenbereiches kann aus der TDT mit Hilfe der Funktion **max_task_info** ausgelesen werden.

data Zeiger auf die Daten

max_get_par_sema max_release_par_sema Parameterbereichs-Semaphore

Pascal FUNCTION max_get_par_sema (hModul: MAXMODHND; task: WORD): MAX_ERROR;

FUNCTION max_release_par_sema (hModul: MAXMODHND; task: WORD): MAX_ERROR;

C MAX_ERROR max_get_par_sema (MAXMODHND hModul, USHORT task);

MAX_ERROR max_release_par_sema (MAXMODHND hModul, USHORT task);

VB Declare Function max_get_par_sema Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer) As Integer

Declare Function max_release_par_sema Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit diesen Funktionen wird der Zugriff auf den Parameterbereich einer Task gesteuert. Damit ein konsistenter Zugriff gewährleistet ist, muss ein Programm, das auf einen Parameterbereich zugreifen will, zunächst versuchen mit **max_get_par_sema** die Semaphore für den Parameterbereich zu bekommen. Gelingt dies, kann das Programm beliebig Daten in den Parameterbereich schreiben oder lesen. Hat sich bereits eine andere Anwendung die Semaphore geholt, liefert die Funktion eine Fehlermeldung und darf anschließend nicht auf den Parameterbereich zugreifen. Mit **max_release_par_sema** wird die Semaphore wieder frei gegeben.

! *Das Betriebssystem prüft bei den Parameterbereichszugriffen selbst nicht, ob die Semaphore frei ist. Um konsistente Daten sicherzustellen müssen alle Anwendungen vor einem Zugriff auf den Parameterbereich einer Task zunächst die Semaphore abprüfen!*

Parameter *hModul* Handle auf das CPU-Modul

task Nummer der Task

max_read_par_uchar
max_read_par_ushort
max_read_par_ulong

Einzelwerte aus dem Parameterbereich lesen

Pascal

FUNCTION max_read_par_uchar (hModul: MAXMODHND; task: WORD; offset: WORD; VAR data: BYTE): MAX_ERROR;

FUNCTION max_read_par_ushort (hModul: MAXMODHND; task: WORD; offset: WORD; VAR data: WORD): MAX_ERROR;

FUNCTION max_read_par_ulong (hModul: MAXMODHND; task: WORD; offset: WORD; VAR data: LONGWORD): MAX_ERROR;

C

MAX_ERROR max_read_par_uchar (MAXMODHND hModul, USHORT task, USHORT offset, UCHAR *data);

MAX_ERROR max_read_par_ushort (MAXMODHND hModul, USHORT task, USHORT offset, USHORT *data);

MAX_ERROR max_read_par_ulong (MAXMODHND hModul, USHORT task, USHORT offset, ULONG *data);

VB

Declare Function max_read_par_uchar Lib "maxw32.dll" (ByVal hModul As Long, ByVal usTask As Integer, ByVal usOffset As Integer, ByRef pucPar As Byte) As Integer

Declare Function max_read_par_ushort Lib "maxw32.dll" (ByVal hModul As Long, ByVal usTask As Integer, ByVal usOffset As Integer, ByRef pusPar As Integer) As Integer

Declare Function max_read_par_ulong Lib "maxw32.dll" (ByVal hModul As Long, ByVal usTask As Integer, ByVal usOffset As Integer, ByRef pulPar As Long) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Mit diesen Funktionen können einzelne Parameter im Parameterbereich einer Task gelesen werden. Die Zugriffe sind in sich konsistent. Daher braucht die Semaphore nicht vorher geholt werden.

Parameter

hModul	Handle auf das CPU-Modul
task	Nummer der Task
offset	Offset (in Bytes) bezogen auf den Anfang des Parameterbereiches, ab dem gelesen werden soll.
data	Zeiger auf eine Variable, in die die Daten eingetragen werden

max_write_par_uchar max_write_par_ushort max_write_par_ulong — Einzelwerte in den Parameterbereich schreiben

Pascal

FUNCTION max_write_par_uchar (hModul: MAXMODHND; task: WORD; offset: WORD; data: BYTE): MAX_ERROR;

FUNCTION max_write_par_ushort (hModul: MAXMODHND; task: WORD; offset: WORD; data: WORD): MAX_ERROR;

FUNCTION max_write_par_ulong (hModul: MAXMODHND; task: WORD; offset: WORD; data: LONGWORD): MAX_ERROR;

C

MAX_ERROR max_write_par_uchar (MAXMODHND hModul, USHORT task, USHORT offset, UCHAR data);

MAX_ERROR max_write_par_ushort (MAXMODHND hModul, USHORT task, USHORT offset, USHORT data);

MAX_ERROR max_write_par_ulong (MAXMODHND hModul, USHORT task USHORT offset, ULONG data);

VB

Declare Function max_write_par_uchar Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal offset As Integer, ByVal data As Byte) As Integer

Declare Function max_write_par_ushort Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal offset As Integer, ByVal data As Integer) As Integer

Declare Function max_write_par_ulong Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal offset As Integer, ByVal data As Long) As Integer

Verwendung: Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion: Mit diesen Funktionen können einzelne Parameter im Parameterbereich einer Task geschrieben werden. Die Zugriffe sind in sich konsistent. Daher braucht die Semaphore nicht vorher geholt werden.

Parameter

hModul	Handle auf das CPU-Modul
task	Nummer der Task
offset	Offset (in Bytes) bezogen auf den Anfang des Parameterbereiches, ab dem geschrieben werden soll.
data	Zu schreibende Daten

max_read_par_block max_write_par_block — Parameterbereich: Blockzugriffe

Pascal

FUNCTION max_read_par_block (hModul: MAXMODHND; task: WORD; offset: LONGWORD; VAR size: WORD; VAR data): MAX_ERROR;

FUNCTION max_write_par_block (hModul: MAXMODHND; task: WORD; offset: LONGWORD; VAR size: WORD; VAR data): MAX_ERROR;

C

MAX_ERROR max_read_par_block (MAXMODHND hModul, USHORT task, ULONG offset, USHORT *size, void *data);

MAX_ERROR max_write_par_block (MAXMODHND hModul, USHORT task, ULONG offset, USHORT *size, void *data);

VB

Declare Function max_read_par_block Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal offset As Integer, ByRef size As Integer, ByRef data As Any) As Integer

Declare Function max_write_par_block Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal offset As Integer, ByRef size As Integer, ByRef data As Any) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit diesen Funktionen wird auf den Parameterbereich einer Task zugegriffen. Vor jedem Zugriff sollte mit **max_get_par_sema** die entsprechende Semaphore geholt werden, um einen konsistenten Zugriff zu gewährleisten.

Parameter

hModul Handle auf das CPU-Modul

task Nummer der Task

offset Offset (in Bytes) bezogen auf den Anfang des Parameterbereiches, ab dem gelesen bzw. geschrieben werden soll.

size Zeiger auf eine Variable, in der steht wie viele Bytes (maximal) gelesen bzw. geschrieben werden sollen. Nach Beendigung der Funktion ist dort eingetragen wie viele Bytes tatsächlich gelesen bzw. geschrieben worden sind. Die Größe des Parameterbereichs kann aus der TDT mit der Funktion **max_task_info** ermittelt werden.

data Zeiger auf die Daten

6.16. Prozeduren und Funktionen in Echtzeitprogrammen

Echtzeitprogramme enthalten als ein wesentliches Element Prozeduren bzw. Funktionen, die von anderen Tasks auf dem gleichen CPU-Modul oder vom Host-PC aus aufgerufen werden können. Voraussetzung dafür ist, dass die Prozeduren bzw. Funktionen in der PDT (Programm Deskriptor Tabelle) des Echtzeitprogramms eingetragen sind (s. Kapitel 6.13).

Der Aufruf erfolgt über das Betriebssystem OsX mit Hilfe der Bibliotheksfunktionen **max_call_proc** bzw. **max_call_func** unter Angabe der Task- und Funktionsnummer der aufzurufenden Prozedur bzw. Funktion.

Einer Funktion können Parameter übergeben werden. Beim Aufruf wird der Funktion die Anzahl der übergebenen und der als Antwort zurück erwarteten Bytes übergeben. Die Funktion liefert die Information, wie viele der übergebenen Bytes verwendet wurden, und wie viele Bytes sie in ihrer Antwort tatsächlich zurückliefert. Die maximal erlaubte Blocklänge wird von der Funktion **max_get_max_blocksize** geliefert.

Prozeduren bekommen keine Parameter übergeben, Prozeduren und Funktionen liefern einen Fehlerstatus zurück.

Da das Echtzeitbetriebssystem OsX eine andere Parameterübergabe-Konvention erfordert, als die mit C oder Pascal Compilern erzeugten Programme, sind am Anfang und Ende einer Prozedur bzw. Funktion jeweils bestimmte Einsprung- und Rücksprung-Routinen einzufügen.

max_entry_proc max_entry_proc_di max_exit_proc max_exit_proc_ei — Prozedur-Rahmen

Pascal

```
FUNCTION max_entry_proc: WORD;
FUNCTION max_entry_proc_di: WORD;
PROCEDURE max_exit_proc;
PROCEDURE max_exit_proc_ei;
```

C USHORT max_entry_proc (void);

USHORT max_entry_proc_di (void);

void max_exit_proc (void);

void max_exit_proc_ei (void);

VB -

Verwendung MAX-PC OsX

Funktion Diese Funktionen bilden zwei unterschiedliche Sätze von Rahmen einer in der PDT eines Echtzeitprogramms eingetragenen Prozedur (also eine Routine, die mit **max_call_proc** aufgerufen wird). Der erste Satz (max_entry_proc und max_exit_proc) hat die Aufgabe, Register zu retten bzw. wiederherzustellen. Der zweite Satz (max_entry_proc_di und max_exit_proc_ei) sperrt zusätzlich die maskierbaren CPU-Interrupts während des Prozeduraufrufs.

Der Aufbau einer Echtzeit-Prozedur muss wie folgt aussehen:

```
void MAXEXPORT MyProc()
{
  USHORT task;

  //Zuweisungen an lokale Variable sind vor max_entry_proc nicht zulässig

  task = max_entry_proc();

  ...

  max_exit_proc();
}
```

Parameter keine

Rückgabewert **max_entry_proc** und **max_entry_proc_di** liefern die Nummer der Task zurück.

max_entry_func max_entry_func_di max_exit_func max_exit_func_ei — Funktions-Rahmen

Pascal
FUNCTION max_entry_func: WORD;
FUNCTION max_entry_func_di: WORD;
PROCEDURE max_exit_func (Error: MAX_ERROR);
PROCEDURE max_exit_func_ei (Error: MAX_ERROR);

C
USHORT max_entry_func (void);
USHORT max_entry_func_di (void);
void max_exit_func (MAX_ERROR Error);
void max_exit_func_ei (MAX_ERROR Error);

VB
-

Verwendung
MAX-PC OsX

Funktion
Diese Funktionen bilden zwei unterschiedliche Sätze für den Rahmen einer in der PDT (Programm Deskriptor Tabelle) eines Echtzeitprogramms eingetragenen Funktion (also eine Routine, die mit **max_call_func** aufgerufen wird). Der erste Satz (max_entry_func und max_exit_func) hat die Aufgabe, die Parameterübergabe-Konvention anzupassen und Register zu retten bzw. wiederherzustellen. Der zweite Satz (max_entry_func_di und max_exit_func_ei) sperrt zusätzlich die maskierbaren CPU-Interrupts während des Funktionsaufrufs.

Der Aufbau einer Echtzeit-Funktion muss wie folgt aussehen:

C:

```
void MAXEXPORT MyFunc (USHORT *pusInsize,  void *pIndata,
                       USHORT *pusRetsize, void *pRetData)
{
  MAX_ERROR Error;
  USHORT task;
  //Zuweisungen an lokale Variable sind vor max_entry_func nicht zulässig

  task = max_entry_func();
  Error = ERR_OK;
  *pusRetsize = ...;       // Rückgabe-Parameter-Länge
  *pusInsize = ...;        // Anzahl der verwendeten Übergabe-Parameter-Bytes

  ...

  max_exit_func(Error);
}
```

Pascal:

```
PROCEDURE MyFunc (VAR pusInsize: WORD; VAR pData;
                  VAR pusRetsize: WORD; VAR pRetData); far;
VAR
   Error : MAX_ERROR;
   usTask : WORD;
   { Zuweisungen an lokale Variable sind vor max_entry_func nicht zulässig }
BEGIN
   usTask := max_entry_func;
   Error := ERR_OK;

   ...

   { wenn nicht alle erhaltenen Daten verwendet werden, pusInsize setzen: }
   pusInsize   := ...;       { Anzahl der verwendeten Parameter Bytes }
   { wenn nicht so viele Daten wie angefordert zurückgeliefert werden, pusRetsize
    setzen: }
   pusRetsize := ...;       { Rückgabe-Parameter Länge }
   max_exit_func(Error);
END;
```


Parameter *error* Fehler, der von der Funktion zurückgeliefert wird. Ist kein Fehler aufgetreten, muss der Wert in der Funktion = 0 gesetzt werden.
Zur Signalisierung eines Fehlers ist Error in der Funktion = xxE0h zu setzen. In xx kann ein beliebiger Fehlerwert eingetragen werden. Das Low-Byte der Fehlervariablen muss den Wert E0h haben.

Rückgabewert: **max_entry_func** und **max_entry_func_di** liefern die Nummer der Task zurück.

max_call_func max_call_proc — Task-Funktionsaufruf

Pascal FUNCTION max_call_func (hModul: MAXMODHND; task: WORD; func: WORD; VAR insize: WORD; VAR indata; VAR outsize: WORD; VAR outdata): MAX_ERROR;

FUNCTION max_call_proc (hModul: MAXMODHND; task: WORD; func: WORD): MAX_ERROR;

C MAX_ERROR max_call_func (MAXMODHND hModul, USHORT task, USHORT func, USHORT *insize, void *indata, USHORT *outsize, void *outdata);

MAX_ERROR max_call_proc (MAXMODHND hModul, USHORT task, USHORT func);

VB Declare Function max_call_func Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal func As Integer, ByRef insize As Integer, ByRef indata As Any, ByRef outsize As Integer, ByRef outdata As Any) As Integer

Declare Function max_call_proc Lib "maxw32.dll" (ByVal hModul As Long, ByVal task As Integer, ByVal func As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit diesen Funktionen wird eine Funktion bzw. Prozedur aus einer Task auf dem angegebenen MAX-PC aufgerufen. Die Funktion bzw. Prozedur muss in der PDT der Task eingetragen sein.

Prozeduren bekommen keine Daten übergeben und liefern keine Ergebnisse zurück. Beim Aufruf von **max_call_proc** wird lediglich überprüft, ob die angegebene Prozedur vorhanden ist. Ist dies nicht der Fall, liefert die Funktion einen Fehler zurück. Die Deklaration einer Prozedur lautet folglich:

Pascal PROCEDURE MyProc

C void MAXEXPORT MyProc (void);

Funktionen können Daten übergeben bekommen und Daten zurückliefern. Sie müssen wie folgt deklariert werden:

Pascal PROCEDURE MAXEXPORT MyFunc (VAR pusInsize: WORD; VAR pDataIn; VAR pusOutsize: WORD; VAR pDataOut);

C void MAXEXPORT MyFunc (USHORT *pusInsize, void *pDataIn, USHORT *pusOutsize, void *pDataOut);

Durch den Zeiger *pusInsize* erfährt die Funktion, wie viele Daten sie durch den Zeiger *pDataIn* übergeben bekommt. Die Funktion muss in *pusInsize* anschließend die Anzahl der verwendeten Bytes eintragen. Durch den Zeiger *pusOutsize* erfährt die Funktion, wie viele Daten sie maximal zurückliefern darf. Die Funktion muss die Rückgabedaten an die Adresse, auf die *pDataOut* zeigt, schreiben. In *pusOutsize* muss sie anschließend die Anzahl der tatsächlich zurückgelieferten Bytes eintragen.

Parameter *hModul* Handle auf das CPU-Modul

task Nummer der Task

func Nummer der Funktion bzw. Prozedur, laut Reihenfolge in der PDT

insize Zeiger auf eine Variable, in der steht, wie viele Bytes an die Funktion übergeben werden sollen. Die maximal erlaubte Datenlänge kann mit der Funktion **max_get_max_blocksize** ermittelt werden. Nach Beendigung der Funktion ist dort eingetragen, wie viele Bytes von der Funktion tatsächlich benutzt worden sind.

indata Zeiger auf die Daten, die der Funktion übergeben werden sollen

outsize Zeiger auf eine Variable, in der steht, wie viele Bytes die Funktion maximal zurückliefern soll. Die maximal erlaubte Datenlänge kann mit der Funktion **max_get_max_blocksize** ermittelt werden. Nach Beendigung der Funktion ist dort eingetragen, wie viele Bytes die Funktion tatsächlich zurückgeliefert hat.

outdata Zeiger auf die Daten, die von der Funktion geliefert werden sollen

6.17. Zugriff auf die Hardware-Devices des MAX-PC

max_set_timer — Setze und starte einen Timer

Pascal FUNCTION max_set_timer (hModul: MAXMODHND; timer: WORD; mode: WORD; value: WORD): MAX_ERROR;

C MAX_ERROR max_set_timer (MAXMODHND hModul, USHORT timer, USHORT mode, USHORT value);

VB -

Verwendung MAX-PC OsX

Funktion Diese Funktion setzt einen Timer und startet ihn. Die Timer sind 16 Bit breit und werden mit jedem Takt um eins dekrementiert. Sobald der Timer auf 0 gezählt hat, wird ein Interrupt ausgelöst und der Timer automatisch wieder mit dem Startwert geladen. Der Timer läuft dann erneut los. Unter dem zugehörigen Interrupt kann z.B. eine II-Task installiert werden.

Parameter *hModul* Handle auf das CPU-Modul

timer Gibt an, welcher Timer gesetzt werden soll (MAX1_TIMER_A, MAX1_TIMER_B oder MAX1_TIMER_C).

mode Betriebsart des Timers. Derzeit ist dieser Parameter immer mit der Konstanten **MAX1_TIMER_MODE_INT** zu beschreiben**.** Timer B kann keinen Interrupt auslösen

value Initialisierungswert des Timers. Die Auflösung der Timer beträgt 0,84µs und die maximale Laufzeit 65535 * 0,840915µs = 55,11 ms.

max_get_timer — Lies Zählerstand eines Timers

Pascal FUNCTION max_get_timer (hModul: MAXMODHND; timer: WORD; VAR status: MAX_TIMER_STATE_TYPE): MAX_ERROR;

C MAX_ERROR max_get_timer (MAXMODHND hModul, USHORT timer, MAX_TIMER_STATE_TYPE *status);

VB -

Verwendung MAX-PC OsX

Funktion Diese Funktion liest den aktuellen Zählerstand eines Timers. Die Timer sind 16 Bit breit und werden mit jedem Takt um eins dekrementiert.

Parameter *hModul* Handle auf das CPU-Modul

timer Gibt an, welcher Timer gelesen werden soll (MAX1_TIMER_A, MAX1_TIMER_B oder MAX1_TIMER_C).

status Zeiger auf eine Struktur vom Typ MAX_TIMER_STATE_TYPE mit Informationen zur Status des Timers. Die Struktur ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
usValue	USHORT	Timerwert
usMode	USHORT	Aktuell eingestellte Betriebsart des Timers

usFlags	USHORT	Flag, das den Zustand des Timer-Ausgangs anzeigt (MAX_TIMER_-OUTPUT), ob der Timer dezimal oder binär zählt (MAX_TIMER_-MODE) und ob der Timer auf 0 gelaufen ist (MAX_TIMER_ZERO)

Einzelheiten finden Sie im Datenblatt des Timer-Bausteins 8254 von Intel bzw. im Handbuch zur CPU (AMD SC400).

max_set_rtc_mode max_get_rtc_status — Betriebsart der Uhr

Pascal

FUNCTION max_set_rtc_mode (hModul: MAXMODHND; mode: WORD): MAX_ERROR;

FUNCTION max_get_rtc_status (hModul: MAXMODHND; VAR status: WORD): MAX_ERROR;

C

MAX_ERROR max_set_rtc_mode (MAXMODHND hModul, USHORT mode);

MAX_ERROR max_get_rtc_status (MAXMODHND hModul, USHORT *status);

VB

Declare Function max_set_rtc_mode Lib "maxw32.dll" (ByVal hModul As Long, ByVal mode As Integer) As Integer

Declare Function max_get_rtc_status Lib "maxw32.dll" (ByVal hModul As Long, ByRef status As Integer) As Integer

Verwendung

Host-PC → MAX-PC OSX, MAX-PC OsX

Funktion

Mit diesen Funktionen wird die Betriebsart der Uhr eingestellt bzw. abgefragt.
Eine Anwendung: Wenn eine Task als II-Task unter dem Interrupt MAX1_IRQ_RTC_PERIODIC installiert wird, erhält man einen weiteren Timer mit den in der Tabelle aufgeführten Zeiten.

Parameter *hModul* Handle auf das CPU-Modul

mode, status Bitfeld mit Kontrollflags, s. nachfolgende Tabelle.

Bit	**Bedeutung**
15	Update in progress (=1)
14-12	Betriebsart, standardmäßig = 010b
11-8	Frequenz am Interruptausgang (IRQ8): 0 = disabled 1 = 3,906 ms 2 = 7,812 ms 3 = 122,070 µs 4 = 244,141 µs 5 = 488,281 µs 6 = 976,563 µs 7 = 1,953 ms 8 = 3,906 ms 9 = 7,812 ms 10 = 15,625 ms 11 = 31,25 ms 12 = 62,5 ms 13 = 125 ms 14 = 250 ms 15 = 500 ms
7	Uhr: 1 = Stop, 0 = Run
6	Periodic Interrupt enabled (=1)
5	Alarm Interrupt enabled (=1)
4	Update-Ended Interrupt enabled (=1)
3	=0 (reserviert)
2	Data Mode (0=BCD, 1=Binary)
1	24/12 (1=24 Hour Mode)
0	Daylight Saving (1=enabled)

max_get_rtc_interrupt_status Lies Interruptzustand der Uhr

Pascal FUNCTION max_get_rtc_interrupt_status (hModul: MAXMODHND; VAR status: BYTE): MAX_ERROR;

C MAX_ERROR max_get_rtc_interrupt_status (MAXMODHND hModul, UCHAR *status);

VB Declare Function max_get_rtc_interrupt_status Lib "maxw32.dll" (ByVal hModul As Long, ByRef status As Byte) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktion liest den aktuellen Interrupt-Zustand der Uhr.

Parameter *hModul* Handle auf das CPU-Modul

status Interrupt-Zustand der Uhr:

Bit	Bedeutung
7	Interrupt-Request-Flag, wird nur gesetzt, wenn einer der Interrupts im RTC enabled ist.
6	Periodic interrupt flag
5	Alarm interrupt flag
4	Update-Ended flag
3-0	Reserviert

max_get_date_and_time max_set_date_and_time — Lies/Setze Datum und Uhrzeit

Pascal
FUNCTION max_get_date_and_time (hModul MAXMODHND; VAR datetime: MAX_TIME_TYPE): MAX_ERROR;

FUNCTION max_set_date_and_time (hModul: MAXMODHND; VAR datetime: MAX_TIME_TYPE): MAX_ERROR;

C
MAX_ERROR max_get_date_and_time (MAXMODHND hModul, MAX_TIME_TYPE *datetime);

MAX_ERROR max_set_date_and_time (MAXMODHND: hModul, MAX_TIME_TYPE *datetime);

VB
Declare Function max_get_date_and_time Lib "maxw32.dll" (ByVal hModul As Long, ByRef datetime As MAX_TIME_TYPE) As Integer

Declare Function max_set_date_and_time Lib "maxw32.dll" (ByVal hModul As Long, ByRef datetime As MAX_TIME_TYPE) As Integer

Verwendung
Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion
Diese Funktionen lesen bzw. setzen das Datum und die Uhrzeit der Echtzeituhr. Die Echtzeituhr muss mit der Funktion max_set_rtc_mode gestartet werden.

Parameter
hModul Handle auf das CPU-Modul

datetime Zeiger auf eine Struktur vom Typ MAX_TIME_TYPE mit Informationen zu Datum und Uhrzeit. Die Struktur hat folgenden Aufbau:

Element	Datentyp	Beschreibung
sec	USHORT	Sekunde
min	USHORT	Minute
hour	USHORT	Stunde
mday	USHORT	Tag
mon	USHORT	Monat
year	USHORT	Jahr
wday	USHORT	Wochentag (0= Sonntag)

max_get_alarm Lies/Setze Alarmzeit
max_set_alarm

Pascal FUNCTION max_get_alarm (hModul MAXMODHND; VAR alarmtime: MAX_ALARM_TYPE): MAX_ERROR;

FUNCTION max_set_alarm (hModul: MAXMODHND; VAR alarmtime: MAX_ALARM_TYPE): MAX_ERROR;

C MAX_ERROR max_get_alarm (MAXMODHND hModul, MAX_ALARM_TYPE *alarmtime);

MAX_ERROR max_set_alarm (MAXMODHND: hModul, MAX_ALARM_TYPE *alarmtime);

VB Declare Function max_get_alarm Lib "maxw32.dll" (ByVal hModul As Long, ByRef alarmtime As MAX_ALARM_TYPE) As Integer

Declare Function max_set_alarm Lib "maxw32.dll" (ByVal hModul As Long, ByRef alarmtime As MAX_ALARM_TYPE) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktionen lesen bzw. setzen die Alarmeinstellung der Echtzeituhr. Es bestehen zwei Möglichkeiten zur Alarmeinstellung:

1. Zu der durch *alarmtime* festgelegten Uhrzeit wird der Alarm ausgelöst (je nachdem, ob mit **max_set_rtc_mode** 12- oder 24-Stunden-Mode eingestellt wurde, einmal oder zweimal täglich).

2. Durch Setzen eines (oder mehrerer) der Parameter *sec*, *min*, *hour* auf den Wert MAX1_CYCLIC_ALARM wird zyklisch ein Alarm ausgelöst. Um jede Sekunde einen Alarm auszulösen, müssen alle drei Parameter auf MAX1_CYCLIC_ALARM gesetzt werden. Soll jede Minute ein Alarm erfolgen, müssen *min* und *hour* auf MAX1_CYCLIC_ALARM gesetzt werden, *sec* gibt die Zeit an, zu der der Alarm ausgelöst wird. Soll jede Stunde ein Alarm ausgelöst werden, muss *hour* auf MAX1_CYCLIC_ALARM gesetzt werden, *min* und *sec* geben die Zeit an, zu der der Alarm ausgelöst wird.

   Wenn eine Task als II-Task unter dem Interrupt MAX1_IRQ_RTC_ALARM installiert wird, wird beim Auftreten des Alarms ein Interrupt ausgelöst.

Parameter *hModul* Handle auf das CPU-Modul

alarmtime Zeiger auf eine Struktur vom Typ MAX_ALARM_TYPE mit Informationen zur Alarmeinstellung. Die Struktur MAX_ALARM_TYPE ist wie folgt aufgebaut:

Element	Datentyp	Beschreibung
sec	USHORT	Sekunde
min	USHORT	Minute
hour	USHORT	Stunde

max_led_on max_led_off — Schalte LED ein/aus

Pascal FUNCTION max_led_on (hModul: MAXMODHND): MAX_ERROR;

FUNCTION max_led_off (hModul: MAXMODHND): MAX_ERROR;

C MAX_ERROR max_led_on (MAXMODHND hModul);

MAX_ERROR max_led_off (MAXMODHND hModul);

VB Declare Function max_led_on Lib "maxw32.dll" (ByVal hModul As Long) As Integer

Declare Function max_led_off Lib "maxw32.dll" (ByVal hModul As Long) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Diese Funktionen schalten die LED auf dem MAX-PC ein (on) bzw. aus (off).

Parameter *hModul* Handle auf das CPU-Modul.

max_led_status — Lies den Zustand der LED

Pascal FUNCTION max_led_status (hModul: MAXMODHND; VAR status: WORD): MAX_ERROR;

C MAX_ERROR max_led_status (MAXMODHND hModul, USHORT *status);

VB Declare Function max_led_status Lib "maxw32.dll" (ByVal hModul As Long, ByRef status As Integer) As Integer

Verwendung Host-PC, MAX-PC OsX

Funktion Die Funktion gibt den Zustand der LED auf dem MAX-PC zurück.

Parameter *hModul* Handle auf das CPU-Modul.

status Zustand der LED: Wenn Bit-0 = 0 ist, dann ist die LED ausgeschaltet, wenn Bit-0 = 1 ist, dann ist sie eingeschaltet.

max_macro_interruptible — Mache Makrobefehle unterbrechbar

Pascal FUNCTION max_macro_interruptible (hModul: MAXMODHND; control: WORD): MAX_ERROR;

C MAX_ERROR max_macro_interruptible (MAXMODHND hModul, USHORT control);

VB Declare Function max_macro_interruptible Lib "maxw32.dll" (ByVal hModul As Long, ByVal control As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX

Funktion Die Funktion dient dazu, Makrobefehle auf dem MAX-PC unterbrechbar zu machen. Wenn sie unterbrechbar sind, bewirkt ein Bibliotheksaufruf nur eine kurze Interrupt-Sperrung auf dem MAX-PC zu Beginn des Aufrufs. Andernfalls ist der gesamte Bibliotheksaufruf nicht von lokalen Hardware-Interrupts unterbrechbar. In diesem Fall kann - z. B. beim Austausch großer Datenmengen mit dem Host-PC - die Reaktionszeit der auf dem MAX-PC laufenden Programme deutlich verlängert werden. Bei schneller Messdatenerfassung können Messwerte verloren gehen (Interrupt-Überlauf). Der Interrupt-Controller des MAX-PC kann keine Interrupt-Überläufe erkennen, aber einige MAX-Module wie das X-C16-3 und X-DiO-40[1].

Parameter *hModul* Handle auf das CPU-Modul.

control = TRUE (1): unterbrechbar
= FALSE (0): nicht unterbrechbar (Voreinstellung).

[1] Die Einstellung ist zur Zeit nur für Ankopplung über PCI-Schnittstelle implementiert. Die Makrobefehlsbearbeitung über andere Schnittstellen ist immer unterbrechbar.

max_cache_control MAX-PC Cache-Kontrolle

Pascal FUNCTION max_cache_control (hModul: MAXMODHND; VAR mode: WORD): MAX_ERROR;

C MAX_ERROR max_cache_control (MAXMODHND hModul, USHORT *mode);

VB Declare Function max_cache_control Lib "maxw32.dll" (ByVal hModul As Long, ByRef mode As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Die Funktion dient zum Ein- und Ausschalten des Prozessor-Caches auf dem MAX-PC.

Parameter *hModul* Handle auf das CPU-Modul.

mode Zeiger auf eine Variable, in der der gewünschte Zustand des Cache steht.
0 = Lies Cache Mode
1 = Cache ungültig machen und ausschalten
2 = Cache einschalten (Write-through)
3 = Cache einschalten. (Write-back)
Nach Verlassen der Funktion steht dort der tatsächliche Zustand.

max_cpu_speed_control Setzen des CPU-Taktes

Pascal FUNCTION max_cpu_speed_control (hModul: MAXMODHND; mode: WORD; speed: WORD): MAX_ERROR;

C MAX_ERROR max_cpu_speed_control (MAXMODHND hModul, USHORT mode, USHORT speed);

VB Declare Function max_cpu_speed_control Lib "maxw32.dll" (ByVal hModul As Long, ByVal mode As Integer, ByVal speed As Integer) As Integer

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Mit dieser Funktion kann die Geschwindigkeit der CPU verändert werden. Die aktuelle CPU-Taktfrequenz kann aus OsX-Parameter 148 mit max_read_par_ushort ausgelesen werden (Task 0). Der CPU-Takt

beeinflusst neben der Verarbeitungsgeschwindigkeit auch die Stromaufnahme und Wärmeentwicklung des Moduls.

Zukünftige CPU-Module bieten unter Umständen andere Einstellmöglichkeiten.

Parameter *hModul* Handle auf das CPU-Modul

mode Die auf dem X-MAX-1 verwendete CPU SC400 bietet die 3 Betriebsarten MAX1_CPU_LOW_SPEED, MAX1_CPU_HIGH_SPEED und MAX1_CPU_HYPER_SPEED.

speed In jeder der 3 durch *mode* eingestellten Betriebsarten lässt sich die CPU mit verschiedenen Taktfrequenzen betreiben:

MAX1_CPU_LOW_SPEED	1, 2, 4 oder 8 MHz.
MAX1_CPU_HIGH_SPEED	8, 16 oder 33 MHz.
MAX1_CPU_HYPER_SPEED	66 oder 100 MHz.

Dafür stehen Konstanten zur Verfügung, z.B. MAX1_SPEED_33MHZ usw.

6

max_write_cmos max_read_cmos — Zugriff auf CMOS-RAM

Pascal

```
FUNCTION max_write_cmos (hModul: MAXMODHND; addr: WORD; size: WORD; Var data): MAX_ERROR;

FUNCTION max_read_cmos (hModul: MAXMODHND; addr: WORD; size: WORD; VAR data): MAX_ERROR;
```

C

```
MAX_ERROR max_write_cmos (MAXMODHND hModul, USHORT addr, USHORT size, void *data);

MAX_ERROR max_read_cmos (MAXMODHND hModul, USHORT addr, USHORT size, void *data);
```

VB

```
Declare Function max_write_cmos Lib "maxw32.dll" (ByVal hModul As Long, ByVal addr As Integer, ByVal size As Integer, ByRef data As Any) As Integer

Declare Function max_read_cmos Lib "maxw32.dll" (ByVal hModul As Long, ByVal addr As Integer, ByVal size As Integer, ByRef data As Any) As Integer
```

Verwendung Host-PC → MAX-PC OsX, MAX-PC OsX

Funktion Die Funktionen schreiben bzw. lesen einen Block in das bzw. aus dem CMOS-RAM des CPU-Moduls. Das X-MAX-1 stellt insgesamt 114 Bytes statisches RAM (in der Echtzeituhr) zur Verfügung. Wird das Modul durch eine externe Batterie gepuffert, bleiben die dort gespeicherten Daten auch beim Abschalten der Versorgungsspannung erhalten.

Parameter

hModul	Handle auf das CPU-Modul.
addr	Adresse, auf die zugegriffen werden soll (14 .. 127).
size	Anzahl Byte, max. 114
data	Zeiger auf Block, der geschrieben werden soll, bzw. in den gelesen werden soll.

max_trigger_watchdog Watchdog Triggerung

Pascal FUNCTION max_trigger_watchdog (hModul: MAXMODHND): MAX_ERROR;

C MAX_ERROR max_trigger_watchdog (MAXMODHND hModul);

VB -

Verwendung MAX-PC OsX

Funktion Diese Funktion dient zur Triggerung des Watchdog. Die Funktionsweise des Watchdog auf dem X-MAX-1 ist in Kapitel 9.3.2.3 beschrieben.

Parameter keine

6.18. Steuerung der lokalen Interrupts auf einem CPU-Modul

max_mask_int max_unmask_int — Interrupt-Maskierung

Pascal FUNCTION max_mask_int (hModul: MAXMODHND; irq: WORD): MAX_ERROR;

FUNCTION max_unmask_int (hModul: MAXMODHND; irq: WORD): MAX_ERROR;

C MAX_ERROR max_mask_int (MAXMODHND hModul, USHORT irq);

MAX_ERROR max_unmask_int (MAXMODHND hModul, USHORT irq);

VB -

Verwendung MAX-PC OsX

Funktion Mit diesen Funktionen kann ein auf dem CPU-Modul lokaler Interrupt gesperrt bzw. freigegeben werden.

Parameter *irq* Interruptnummer: 78h..7fh, 90h..97h, 98h..9ah

max_clear_int — Pending-Interrupt löschen

Pascal FUNCTION max_clear_int (hModul: MAXMODHND; irq: WORD): MAX_ERROR;

C MAX_ERROR max_clear_int (MAXMODHND hModul, USHORT irq);

VB -

Verwendung MAX-PC OsX

Funktion Diese Funktion löscht einen Interrupt im Interrupt-Controller (Interrupt Request Register). Sie wird in der Regel vor **max_unmask_int** verwendet, um sicherzustellen, dass ein vorher aufgetretener Interrupt nicht zur Ausführung kommt.

Parameter *irq* Interruptnummer: 78h..7fh, 90h..97h, 98h..9ah

max_get_nmi_reason NMI-Ursache

Pascal FUNCTION max_get_nmi_reason (hModul: MAXMODHND; VAR reason: WORD): MAX_ERROR;

C MAX_ERROR max_get_nmi_reason (MAXMODHND hModul, USHORT *reason);

VB -

Verwendung MAX-PC OsX

Funktion Mit dieser Funktion lässt sich die Ursache für einen aufgetretenen NMI-Interrupt ermitteln. Sie kommt normalerweise in der Hauptprozedur einer unter dem NMI-Interrupt installierten II-Task zum Einsatz. Informationen zum NMI-Interrupt auf dem X-MAX-1 stehen im Kapitel 10.22.6.

Parameter *reason* Zeiger auf eine Variable, in die die NMI-Interruptursache eingetragen wird.

max_disable_int max_enable_int Interrupt-Sperrung

Pascal PROCEDURE max_disable_int: MAX_ERROR;

PROCEDURE max_enable_int: MAX_ERROR;

C void max_disable_int (void);

void max_enable_int (void);

VB -

Verwendung MAX-PC OsX

Funktion Diese Funktionen dienen dazu, eine Subroutine in einem Echtzeitprogramm gegen Unterbrechung durch Interrupts zu schützen. Dazu muss zunächst **max_disable_int** aufgerufen werden. Bis zum Befehl **max_enable_int** sind in der Folge alle Befehle gegen die Unterbrechung durch Interrupts geschützt. Die beiden Aufrufe bilden immer ein Paar, d.h. sie müssen in derselben Subroutine stehen.

Parameter keine

6.19. Fehlerbehandlung

Alle Funktionen der Bibliothek, in denen Fehler auftreten können, liefern einen Fehlerstatus als Funktionsrückgabewert. Dieser kann z.B. angeben, dass ein übergebener Parameter einen unerlaubten Wert hat oder dass es zu einem Fehler in der Kommunikation mit der Hardware gekommen ist. Wenn kein Fehler aufgetreten ist, liefern die Funktionen ERR_OK (= 0) zurück. Beim Auftreten eines Fehlers wird der entsprechende Fehlercode innerhalb der Bibliothek gespeichert. Der Rückgabewert muss im Anschluss an jeden Funktionsaufruf überprüft werden.

Außerdem kann es zu (spontanen) Fehler- oder sonstigen Meldungen von einem MAX-PC kommen. Diese werden durch das Verschicken eines Error-Requests signalisiert. Die Bibliothek nimmt Error-Requests entgegen und reagiert durch den Aufruf der Anwender-Service-Request-Routine, sofern eine solche installiert worden ist (s. Kapitel 6.12). Enthält das Anwenderprogramm keine solche Routine, wird eine interne Behandlungsroutine aufgerufen und die Bibliothek speichert den Fehlercode in der Bibliothek. Unter Windows werden die Fehler Thread-spezifisch gespeichert.

Die Funktionalität der nachfolgenden Funktionsaufrufe wird nicht ausgeführt, wenn für dasselbe Zielsystem ein Fehler in der Bibliothek zwischengespeichert ist. Sie liefern statt dessen den in der Bibliothek gespeicherten Fehlercode zurück! Funktionen, die sich auf ein anderes Zielsystem beziehen, werden normal ausgeführt.

Innerhalb der Bibliothek gespeicherte Fehler müssen mit **max_clear_error** wieder zurückgesetzt werden.

max_get_error_message — Klartext-Fehlermeldungen

Pascal	FUNCTION max_get_error_message (error: MAX_ERROR; VAR size: WORD; message: PChar): MAX_ERROR;
C	MAX_ERROR max_get_error_message (MAX_ERROR error, USHORT *size, char *message);
VB	Declare Function max_get_error_message Lib "maxw32.dll" (ByVal error As Integer, ByRef size As Integer, ByVal message As String) As Integer
Verwendung	Host-PC, MAX-PC OsX
Funktion	Diese Funktion liefert eine zu einem Fehlercode gehörige Textmeldung. Ist der Fehlercode nicht definiert, gibt die Funktion ihrerseits eine

Fehlermeldung zurück. Die Sprache der Fehlermeldung wird durch **max_init_lib** festgelegt.

Parameter *error* Fehlercode

size Zeiger auf eine Variable, in der die maximal erlaubte Länge des zurückgelieferten Textes angegeben wird. Eine Angabe von 200 ist dafür ausreichend. Die Funktion trägt anschließend die Länge des Strings ein.

message Zeiger auf eine Variable, in die der zugehörige Text als nullterminierter String eingetragen wird.

max_clear_error Fehler löschen

Pascal PROCEDURE max_clear_error;

C void max_clear_error (void);

VB Declare Sub max_clear_error Lib "maxw32.dll" ()

Verwendung Host-PC, MAX-PC OsX

Funktion Diese Funktion setzt nach einem aufgetretenen Fehler den in der Bibliothek gespeicherten Fehlerstatus zurück.

Parameter keine

max_set_error_check_level Fehlerüberprüfungs-Mode einstellen

Pascal PROCEDURE max_set_error_check_level (level: WORD);

C void max_set_error_check_level (USHORT level);

VB -

Verwendung MAX-PC OsX

Funktion Mit dieser Funktion kann die Verarbeitungsgeschwindigkeit der geschwindigkeitskritischen Bibliotheksaufrufe erhöht werden. In den MDD-Schreib- und Lesefunktionen kann damit die Gültigkeitsprüfung der übergebenen Handles sowie die Prüfung, ob die richtige Funktion für den Kanal aufgerufen wurde, abgeschaltet werden. Funktionelle Fehler, die der Modul-Device-Treiber in der Hardware feststellt, werden davon nicht beeinflusst.

Diese Überprüfungen werden standardmäßig von der Bibliothek vorgenommen. Sobald ein Programm fertiggestellt und getestet ist, kann man normalerweise darauf verzichten. Es muss aber dann unbedingt überprüft werden, dass max_open_channel ein gültiges Handle zurückliefert und dass das Schließen des Handles nicht zwischenzeitlich erfolgt.

Achtung: Bei abgeschalteter Fehler-Überprüfung kann die Übergabe eines ungültigen Handles oder der Aufruf einer nicht vom Kanal unterstützten Datenfunktion (z.B. max_read_channel_uchar für Blockkanal) zum Absturz des CPU-Moduls führen.

Parameter

level: Gibt an, wie streng die Bibliothek Fehler abprüfen soll:

- 0: In den Geschwindigkeits-kritischen Funktionen wird auf die Überprüfung der Übergabeparameter verzichtet.
- 10: Die strengst mögliche Fehlerprüfung wird vorgenommen (default-Einstellung).
- 1..9: zur Zeit reserviert.

Notizen

7. Treibersoftware für OsX

7.1. Display-Grafiktreiber

7.1.1. Allgemeines

Das OsX-Programm OSXTXTKB.EXE dient neben der Textausgabe und Ansteuerung der Tastatur zusätzlich zur Grafikansteuerung der Displays Hitachi SX16H003-ZZA und Sharp LQ057Q3DC02.

Zur Nutzung der nachfolgend aufgelisteten Funktionen muss der Treiber OSXTXTKB.EXE installiert sein und die Bibliothek OSXGRAPH.LIB und die Header-Datei OSXGRAPH.H in das OsX-Echtzeitprogramm eingebunden werden.

Die linke obere Ecke des Displays hat die Koordinaten X=0 ,Y=0. Die rechte untere Ecke des Sharp Displays hat die Koordinaten X=319, Y=239. Die rechte untere Ecke des Hitachi Displays hat die Koordinaten X=639, Y=239.

Der Treiber OSXTXTKB.EXE hat die Programmnummer A002h. Die Installation mit einer .INS-Datei:

MAXINST file="osxtxtkb.exe" no=A002 task=3f0 tasktype=MAX_NI_TASK autoinit

Übersicht der verfügbaren Funktionen:

max_init_graphic	Grafik-Initialisierung
max_set_pen_color	Setze-Stift-Farbe
max_draw_pixel	Zeichne-Bildpunkte
max_draw_line	Zeichne-Linie
max_draw_rectangle	Zeichne-Rechteck
max_fill_rectangle	Fülle-Rechteck
max_draw_ellipse	Zeichne-Ellipse
max_fill_ellipse	Fülle-Ellipse

Übersicht der verfügbaren Funktionen:

max_set_front	Setze-Schrifttyp
max_draw_text	Schreibe-Text
max_set_brush_color	Setze-Pinsel-Farbe
max_plot_pixel	Zeichne-Bildpunkte
max_plot_line	Zeichne-Linie
max_plot_box / max_plot_filled_box	Zeichne-Rechteck
max_plot_ellipse / max_plot_filled_ellipse	Zeichne-Ellipse
max_plot_text	Zeichne-Text

7.1.2. Funktionen

max_init_graphic **Grafik-Initialisierung**

C	MAX_ERROR max_init_graphic(MAXMODHND hModule, USHORT usDisplayType, USHORT usDisplayMode, USHORT usFlags)

Funktion	Diese Funktion führt die Initialisierung des Displays durch. Sie muss vor allen anderen Grafik-Funktionen aufgerufen werden.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usDisplayType	Verwendetes Display. Bisher werden das Hitachi-Display SX16H003-ZZA (640*240) und das Sharp-Display LQ057Q3DC02 (320*240) unterstützt. Hierfür sind zwei Konstanten definiert: HITACHI_SX16H003ZZA_DISPLAY SHARP_LQ057Q3DC02_DISPLAY
	usDisplayMode	Hier wird definiert, ob das Display in den Grafik-Modus oder Text-Modus geschaltet werden soll: GRAPHIC_MODE TEXT_MODE Für die weiteren Grafikfunktionen muss das Display in den Grafik-Modus geschaltet werden.
	usFlags	Reservierter Parameter. Muss 0 gesetzt sein!

max_set_pen_color **Setze-Stift-Farbe**

C	max_set_pen_color(MAXMODHND hModule, UCHAR ucRed, UCHAR ucGreen, UCHAR ucBlue)

Funktion	Setzt die Farbe des Stiftes. Dieser wird für alle weiteren Grafikfunktionen verwendet.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	ucRed	Rot-Anteil
	ucGreen	Grün-Anteil
	ucBlue	Blau-Anteil

max_draw_pixel **Zeichne-Bildpunkt**

C	max_draw_pixel(MAXMODHND hModule, USHORT usX, USHORT usY)

Funktion	Mit Hilfe dieser Funktion ist es möglich, einzelne Bildpunkte mit der aktuellen Farbe des Stiftes zu setzen.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate
	usY	Y-Koordinate

max_draw_line **Zeichne-Linie**

C	max_draw_line(MAXMODHND hModule, USHORT usX1, USHORT usY1, USHORT usX2, USHORT usY2)

Funktion	Mit Hilfe dieser Funktion ist es möglich, eine Linie zwischen zwei Punkten zu zeichnen. Die Linie hat dabei die Farbe des aktuellen Stiftes.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usX1	X-Koordinate Startpunkt
	usY1	Y-Koordinate Startpunkt
	usX2	X-Koordinate Endpunkt
	usY2	Y-Koordinate Endpunkt

max_draw_rectangle **Zeichne-Rechteck**

C	max_draw_rectangle(MAXMODHND hModule, USHORT usX, USHORT usY, USHORT usWidth, USHORT usHeight)

Funktion	Mit Hilfe dieser Funktion ist es möglich, ein Rechteck zu zeichnen. Die Linie hat dabei die Farbe des aktuellen Stiftes.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate der linken oberen Ecke
	usY	Y-Koordinate Startpunkt linken oberen Ecke
	usWidth	Breite des Rechtecks
	usHeight	Höhe des Rechtecks

max_fill_rectangle **Fülle-Rechteck**

C	max_fill_rectangle(MAXMODHND hModule, USHORT usX, USHORT usY, USHORT usWidth, USHORT usHeight)

Funktion	Mit Hilfe dieser Funktion ist es möglich, ein Rechteck mit der Farbe des aktuellen Stiftes auszufüllen.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate der linken oberen Ecke
	usY	Y-Koordinate Startpunkt linken oberen Ecke
	usWidth	Breite des Rechtecks
	usHeight	Höhe des Rechtecks

max_draw_ellipse **Zeichne-Ellipse**

C	max_draw_ellipse(MAXMODHND hModule, USHORT usX, USHORT usY, USHORT usWidth, USHORT usHeight)

Funktion	Mit Hilfe dieser Funktion ist es möglich, eine Ellipse zu zeichnen. Die Linie hat dabei die Farbe des aktuellen Stiftes.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate der linken oberen Ecke des Rechtecks, das die Ellipse umschließt
	usY	Y-Koordinate der linken oberen Ecke des Rechtecks, das die Ellipse umschließt
	usWidth	Breite des Rechtecks, das die Ellipse umschließt
	usHeight	Höhe des Rechtecks, das die Ellipse umschließt

max_fill_ellipse **Fülle-Ellipse**

C	max_fill_ellipse(MAXMODHND hModule, USHORT usX, USHORT usY, USHORT usWidth, USHORT usHeight)

Funktion	Mit Hilfe dieser Funktion ist es möglich, eine Ellipse mit der Farbe des aktuellen Stiftes auszufüllen.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate der linken oberen Ecke des Rechtecks, das die Ellipse umschließt
	usY	Y-Koordinate der linken oberen Ecke des Rechtecks, das die Ellipse umschließt
	usWidth	Breite des Rechtecks, das die Ellipse umschließt
	usHeight	Höhe des Rechtecks, das die Ellipse umschließt

max_set_font **Setze-Schrifttyp**

C	max_set_font(MAXMODHND hModule, USHORT usFontName, USHORT usFontStyle, USHORT usFontSize)

Funktion	Mit Hilfe dieser Funktion ist es möglich, eine Schrift für die Ausgabe von Text festzulegen.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usFontName	FONT_MONOSPACED
	usFontStyle	FONT_STYLE_PLAIN
	usFontSize	Für die Ausgabe von Text stehen momentan drei Schriftgrößen zur Verfügung: FONT_8_8 (8*8 Pixel) FONT_8_12 (8*12 Pixel) FONT_8_14 (8*14 Pixel)

max_draw_text **Schreibe-Text**

C	max_draw_text(MAXMODHND hModule, USHORT usX, USHORT usY, CHAR* pcText, USHORT usSize)

Funktion	Mit Hilfe dieser Funktion ist es möglich, Text mit der aktuellen Schriftart auszugeben. Der Text hat die Farbe des ausgewählten Stiftes.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate der linken oberen Ecke
	usY	Y-Koordinate linken oberen Ecke
	pcText	Text der ausgegeben werden soll (maximal 200 Zeichen).
	usSize	Anzahl der Textzeichen (maximal 200).

max_set_brush_color **Setze-Pinsel-Farbe**

C	max_set_brush_color(MAXMODHND hModule, UCHAR ucRed, UCHAR ucGreen, UCHAR ucBlue)

Funktion	Setzt die Farbe des Pinsels.

Parameter	hModule	Handle des CPU-Moduls, auf dem die Treiber-Task installiert ist.
	ucRed	Rot
	ucGreen	Grün
	ucBlue	Blau

max_plot_pixel **Zeichne-Bildpunkt**

C	max_plot_pixel(MAXMODHND hModule, USHORT usX, USHORT usY, ULONG ulColour)

Funktion	Mit Hilfe dieser Funktion ist es möglich einzelne Bildpunkte des Displays zu setzen.

Parameter	hModule	Handle des CPU-Moduls auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate
	usY	Y-Koordinate
	ulColour	Farbe des Punktes.

max_plot_line **Zeichne-Linie**

C	max_plot_line(MAXMODHND hModule, USHORT usX1, USHORT usY1, USHORT usX2, USHORT usY2, ULONG ulColour)

Funktion	Mit Hilfe dieser Funktion ist es möglich eine Linie zwischen zwei Punkten zu zeichnen.

Parameter	hModule	Handle des CPU-Moduls auf dem die Treiber-Task installiert ist.
	usX1	X-Koordinate Startpunkt
	usY1	Y-Koordinate Startpunkt
	usX2	X-Koordinate Endpunkt
	usY2	Y-Koordinate Endpunkt
	ulColour	Farbe der Linie.

max_plot_box / max_plot_filled_box Zeichne-Rechteck

C	max_plot_box(MAXMODHND hModule, USHORT usX1, USHORT usY1, USHORT usX2, USHORT usY2, ULONG ulColour)
	max_plot_filled_box(MAXMODHND hModule, USHORT usX1, USHORT usY1, USHORT usX2, USHORT usY2, ULONG ulColour)

Funktion	Mit Hilfe dieser Funktion ist es möglich ein Rechteck (nicht ausgefüllt oder ausgefüllt) zu zeichnen.

Parameter	hModule	Handle des CPU-Moduls auf dem die Treiber-Task installiert ist.
	usX1	X-Koordinate der linken oberen Ecke
	usY1	Y-Koordinate Startpunkt linken oberen Ecke
	usX2	X-Koordinate der rechten unteren Ecke.
	usY2	Y-Koordinate der rechten unteren Ecke.
	ulColour	Farbe des Rechtecks.

max_plot_ellipse / max_plot_filled_ellipse Zeichne-Ellipse

C	max_plot_ellipse(MAXMODHND hModule, USHORT usX, USHORT usY, USHORT usXDiameter, USHORT usYDiameter, ULONG ulColour)
	max_plot_filled_ellipse(MAXMODHND hModule, USHORT usX, USHORT usY, USHORT usXDiameter, USHORT usYDiameter, ULONG ulColour)

Funktion	Mit Hilfe dieser Funktion ist es möglich eine Ellipse (nicht ausgefüllt oder ausgefüllt) zu zeichnen.

Parameter	hModule	Handle des CPU-Moduls auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate der linken oberen Ecke
	usY	Y-Koordinate linken oberen Ecke
	usXDiameter	X-Durchmesser der Ellipse.
	usYDiameter	Y-Durchmesser der Ellipse.
	ulColour	Farbe der Ellipse.

max_plot_text **Zeichne-Text**

C	max_plot_text(MAXMODHND hModule, USHORT usX, USHORT usY, USHORT usFont, ULONG ulForeColour, ULONG ulBackColour, CHAR* pcText, USHORT usSize)

Funktion	Mit Hilfe dieser Funktion ist es möglich Text auszugeben.

Parameter	hModule	Handle des CPU-Moduls auf dem die Treiber-Task installiert ist.
	usX	X-Koordinate der linken oberen Ecke
	usY	Y-Koordinate linken oberen Ecke

	usFont	Für die Ausgabe von Text stehen momentan drei verschiedene Schriftarten (8, 12 und 14 Punkte) zur Verfügung: FONT_8 FONT_12 FONT_14
	ulForeColour	Textfarbe
	ulBackColour	Hintergrundfarbe
	pcText	Text der ausgegeben werden soll (maximal 200 Zeichen).
	usSize	Anzahl der Textzeichen (maximal 200).

7.2. Display-Text und XT-Tastatur Treiber

Das OsX-Programm OSXTXTKB.EXE dient zur Textausgabe auf dem Display (Typ Hitachi SX16H003-ZZA) und zur Ansteuerung einer XT-Tastatur.

Das Programm ermöglicht die Benutzung folgender Standard-Bibliotheksfunktionen von Borland C++ für die Textausgabe:

gotoxy, textcolor, textbackground, textattr, highvideo, lowvideo, normvideo, wherex, wherey, clrscr, cprintf.

Zusätzlich werden 2 Funktionen für die Tastaturabfrage zur Verfügung gestellt:

kbhit und *getch.*

Alle aufgezählten Funktionen sind in *conio.h* deklariert. Um diese in OSXTXTKB.EXE realisierten Funktionen zu benutzen, müssen Sie die Importdatei TXTKB.OBJ in Ihr Projekt einfügen.

Bei der Benutzung von *getch()* ist zu beachten, dass die Funktion im Unterschied zur DOS-Funktion nicht auf ein Zeichen von der Tastatur wartet. Dies würde der Philosophie des Betriebssystems OsX widersprechen. Sie sollte also nur aufgerufen werden, wenn *kbhit()* zuvor einen Wert <> 0 zurückgegeben hat. Wird *getch()* aufgerufen, wenn sich kein Zeichen im Tastaturpuffer befindet, wird 0 zurückgegeben.

Hinweis: Zur Benutzung der Tastaturfunktionen muss OSXTXTKB.EXE als II-Task unter Interrupt 79h installiert werden!

Installation der Task OSXTXTKB.EXE

Bevor andere OsX-Tasks die Text- und Tastaturfunktionen von OSXTXTKB.EXE nutzen können, muss OSXTXTKB.EXE installiert, parametriert und gestartet werden. Dies kann mit Hilfe einer Installationsdatei oder in einem PC-Programm durchgeführt werden.

Einstellbar sind die Zeichengröße und der Display-Kontrast und die Helligkeit der Hintergrundbeleuchtung.

Die Zeichengröße wird in Parameter 12 (WORD) der Task OSXTXTKB.EXE vor Aufruf der Startprozedur (Prozedur 2) eingestellt. Mögliche Werte sind: 8, 12, 14, 16, 18, 24. Als Default-Wert ist 8 eingestellt.

Die Kontrastspannung für das Display wird in Parameter 8 (WORD) eingestellt. Der Wertebereich ist 0 (niedrigster Kontrast) – fffh (stärkster Kontrast). Default-Wert ist c00h.

Die Helligkeit der Hintergrundbeleuchtung wird in Parameter 10 (WORD) eingestellt. Der Wertebereich ist 0-fffh. Default-Wert ist 0. Höhere Werte bedeuten geringere Helligkeit.

Die Werte aus Parameter 8 und 10 werden zunächst in der Startprozedur eingestellt. Sollen die Werte nachträglich verändert werden, muss nach Einstellen der Parameter die Prozedur 4 der Task aufgerufen werden.

Die Tastaturfunktionen sind nur verfügbar, wenn OSXTXTKB.EXE als II-Task unter Interrupt 79h installiert wird. Andernfalls geben *kbhit()* und *getch()* stets 0 zurück.

Beispiel für die Installation aus einer .INS-Datei:

```
; Task installieren
MAXINST file="osxtxtkb.exe" no=A002 task=03f0 tasktype=MAX_II_TASK irq=79
        autoinit
; Zeichengröße 16 einstellen
MAXPAR task=03f0 start=000c para=10 00
; Startprozedur aufrufen
MAXPROC task=03f0 proc=0002
```

Prozeduren und Funktionen der Task OSXTXTKB.EXE

Prozedur 0:

Hauptprozedur, dient als Interrupt-Handler für die Tastatur, sofern die Task als II-Task installiert wird

Prozedur 1:

AutoInit: Initialisierung der OsX-Bibliothek und der Taskparameter. Muss nach der Installation vor allen anderen Aufrufen ausgeführt werden (kann beim Installieren automatisch ausgeführt werden).

Prozedur 2:

Startprozedur: Initialisiert die Grafik- und Tastaturfunktionen, stellt Kontrast und Helligkeit ein.

Prozedur 3:

Stop-Prozedur: Schaltet Grafik und Tastatur aus

Prozedur 4:

Übernahme der Veränderung von Kontrast und Helligkeit (Parameter 8 und 10)

Funktion 5:

Textausgabe auf dem Display an der aktuellen Cursor-Position mit den aktuellen Text-Attributen

Daten HIN: nullterminierter String
Länge HIN: String-Länge inkl. Nullterminierung
Daten ZURÜCK: keine
Länge ZURÜCK: 0

Funktion 6:
Einstellen der Textattribute und der Cursorposition für Textausgabe mit Funktion 5.
Parameter:

Daten HIN:

```
struct _ATTRIBUTES
{
  USHORT usX;            // 0xFFFF: Cursorposition wird nicht verändert
                         // sonst:  Cursor wird auf Position usX|usY gesetzt
  USHORT usY;            // nur ausgewertet, wenn usX != 0xFFFF
  USHORT usAttr;         // 0xFFFF: Textattribute nicht ändern
                         // sonst: Textattribute
                         //     Bit 7: BLINK
                         //     Bits 6-4:     Hintergrundfarbe
                         //     Bits 3-0:     Vordergrundfarbe
                         // entspr. den Definitionen in graphics.h
                            (Borland C)
USHORT usClr;            // 1: Bildschirm wird gelöscht
};
```

Länge HIN: 8
Daten ZURÜCK: keine
Länge ZURÜCK: 0

Parameter der Task OSXTXTKB.EXE

Nr.	Typ	Zugr.	Init	Bedeutung
8	WORD (2)	R/W	0C00h	Kontrasteinstellung für das Display, Wertebereich: 0-0fffh
10	WORD (2)	R/W	0	Helligkeit der Hintergrundbeleuchtung, Wertebereich: 0-0fffh. Höhere Werte bedeuten geringere Helligkeit.
12	WORD (2)	R/W	8	Zeichenhöhe (Pixel) des verwendeten Fonts
14	WORD (2)	R/W	1	Display-Typ 1 = Hitachi SX16H003-ZZA 2 = Sharp LQ057Q3DC02

7.3. Touch-Screen

Das OsX-Programm OSXTOUCH.EXE steuert den Touchscreen-Controller für das Hitachi-Display auf dem X-KIT-3.

Das Programm unterstützt 3 Betriebsarten:

- Normalbetrieb: Nutzung der Touchscreen-Funktionalität in OsX-Tasks
- Kalibrierbetrieb: Es wird zunächst oben links auf dem Display ein Kreuz eingeblendet. Der Benutzer muss dieses mit einem Stift berühren. Das Kreuz wird daraufhin unten links eingeblendet. Nach Berühren dieses Kreuzes werden neue Kalibrierdaten errechnet und der Kalibrierbetrieb wird beendet.
- Testbetrieb: Zeichnen mit dem Stift auf dem Display

Installation der Task OSXTOUCH.EXE

Bevor andere OsX-Tasks die Touchscreen-Funktionen nutzen können, muss OSXTOUCH.EXE installiert, parametriert und gestartet werden. Dies kann mit Hilfe einer Installationsdatei oder in einem PC-Programm durchgeführt werden.

Einstellbar sind die Kalibrierwerte, der Display-Kontrast und die Helligkeit der Hintergrundbeleuchtung.

Bei der Initialisierung des Programms werden zunächst Standard-Kalibrierwerte benutzt. Diese können durch Zugriff auf den Parameterbereich vor dem Start der Task überschrieben werden.

Beispiel für die Installation aus einer .INS-Datei:

```
; Task installieren
MAXINST file="osxtouch.exe" no=30a task=03f1 tasktype=MAX_NI_TASK autoinit
; Kalibrierdaten eintragen
MAXPAR task=03f1 start=000a  99 01 68 0e 47 0c 6b 03
; Task starten
MAXPROC task=03f1 proc=0002
```

Prozeduren der Task OSXTOUCH.EXE

Prozedur 0:

Hauptprozedur, bedient Service-Anforderungen des Touchscreen-Controllers

Prozedur 1:

AutoInit: Initialisierung der OsX-Bibliothek und der Taskparameter. Muss nach der Installation vor allen anderen Aufrufen ausgeführt werden.

Prozedur 2:

Startprozedur: Startet die Touchscreen-Task im Normalbetrieb

Prozedur 3:

Stop-Prozedur: Beendet die Auswertung der Touchscreen-Daten. Falls der Grafikbetrieb aktiviert wurde, wird das Display abgeschaltet.

Prozedur 4:

Startet die Kalibrierprozedur. Nach Abschluss der Kalibrierung können die neuen Kalibrierdaten aus den Parametern 10-17 ausgelesen werden.

Prozedur 5:

Zeichnen auf dem Display

Prozedur 6:

Display im Grafikmodus initialisieren, ohne die Betriebsart der Touchscreen-Task zu ändern. Diese Prozedur kann zur Grafikcontroller-Initialisierung verwendet werden, wenn das Display aus einer anderen OsX-Task im Grafikmodus verwendet werden soll. Die Adresse des Frame-Buffers steht im Parameterbereich der Task.

Prozedur 7:

Grafikmodus abschalten, falls er durch Prozedur 4, 5 oder 6 aktiviert wurde

Nutzen der Touchscreen-Funktionalität in anderen OsX-Tasks

Wenn die Touchscreen-Task im Normalbetrieb läuft, können andere Tasks den aktuellen Zustand des Touch-Pens im Parameterbereich abfragen. Der Zustand des Stifts (gedrückt oder nicht gedrückt) steht in Parameter 0, die Koordinaten (X und Y) stehen in den Parametern 2 und 4. Die Koordinaten sind nur gültig, wenn Parameter 0 = 1 ist!

Direkter Zugriff auf den Framebuffer im Grafikbetrieb

Für Anwendungen, die durch direkten Schreibzugriff auf den Framebuffer Grafik darstellen sollen, können die Prozeduren 6 und 7 verwendet werden, um den Grafikcontroller zu bedienen. Aus dem Parameterbereich kann die Startadresse des Framebuffers ausgelesen werden. Das erste Byte im Framebuffer stellt die beiden Pixel links oben auf dem Display dar, wobei das linke Pixel in den 4 höherwertigen Bits gespeichert wird. Es sind 16 Farbwerte pro Pixel möglich. Das Display hat eine Auflösung von 640x240 Punkten. Eine Zeile im Display umfasst 320 Bytes. Der gesamte Framebuffer ist 76800 Bytes lang.

Kalibrierdaten

Gerätespezifische Kalibrierdaten können mit Hilfe der Prozedur 4 ermittelt werden. Nach Abschluss der Kalibrierprozedur werden diese im Parameterbereich der Task abgelegt. Die Kalibrierdaten werden derzeit noch nicht im nichtflüchtigen Speicher abgelegt. Deshalb muss sich eine andere OsX-Task oder der PC um die Verwaltung dieser Daten kümmern. Sinnvollerweise sollten die Daten nach Ausführen der Kalibrierprozedur ausgelesen und zwischengespeichert werden.

Um die zwischengespeicherten Kalibrierdaten später wiederzuverwenden, müssen sie vor Aufruf der Startprozedur in den Parameterbereich eingetragen werden.

Parameter der Touchscreen-Task

Nr.	Typ	Zugr.	Init	Bedeutung
0	WORD (2)	R		Status des Touch-Pen: 0: nicht gedrückt 1: gedrückt
2	WORD (2)	R		X-Koordinate des Stifts nur gültig, wenn Parameter 0 == 1!
4	WORD (2)	R		Y-Koordinate des Stifts nur gültig, wenn Parameter 0 == 1!
10	8 Byte	R/W		Kalibrierdaten
34	WORD (2)	R/W	0C00h	Kontrasteinstellung für das Display, Wertebereich: 0-0fffh
36	WORD (2)	R/W	0	Helligkeit der Hintergrundbeleuchtung, Wertebereich: 0-0fffh. Höhere Werte bedeuten geringere Helligkeit.
38	DWORD (4)	R	-	Phys. Adresse des Framebuffers im Grafikbetrieb

7.4. Software für PCMCIA-Karten

Diese Software wird in Application Notes beschrieben.

7.5. FAT-Dateisystem für PCMCIA- bzw. Compact-Flash-Karten an einem MAX-PC

Im folgenden wird die Verwendung des Treiberprogramms OSXFAT16 beschrieben. Damit können Dateien von einem MAX-PC auf eine PCMCIA- bzw. Compact-Flash Speicherkarte geschrieben bzw. davon gelesen werden, die an einer der beiden PCMCIA-Schnittstellen eines MAX-PC angeschlossen ist. Für Evaluierungszwecke kann das Programm z.B. auf dem X-KIT-3 ausprobiert werden.

Verwendet wird das von MS-DOS bekannte FAT16-Dateisystem.

Das Treiber-Programm ermöglicht alle üblichen Datei-Operationen, wie z.B. Anlegen, Lesen, Schreiben, Datei-Informationen ermitteln usw. Dazu dürfen jedoch nicht die Standard-Stream-Funktionen der Compiler-Bibliotheken verwendet werden wie z.B. fopen, fread usw. Die Schnittstelle besteht statt dessen aus einer Reihe von aufrufbaren Funktionen (s.u.).

Karteninformationen können abgefragt und eine Karte mit FAT16 formatiert werden.

Das Einstecken bzw. Entfernen einer Karte wird selbstständig erkannt und behandelt. Zusätzlich kann das Anwenderprogramm mittels einer Callback-Funktion darüber benachrichtigt werden.

Mit dem Treiberprogramm können Unterverzeichnisse in beliebiger Verschachtelungstiefe verwaltet werden. Alle Datei-bezogenen Funktionen können auf Dateien in Unterverzeichnissen genauso wie auf solche im Hauptverzeichnis angewendet werden. Gegenüber anderen FAT16-Implementierungen gibt es folgende **Einschränkung**:

Jedes Unterverzeichnis darf maximal nur die Anzahl der im Hauptverzeichnis erlaubten Dateien enthalten. Diese kann mit Funktion 12 ermittelt werden. Dabei ist zu berücksichtigen, dass jedes Unterverzeichnis immer 2 System-Dateien („.“ und „..“) enthält. Ein Unterverzeichnis wird in jedem Fall als ein Verzeichniseintrag in seinem Basisverzeichnis gezählt.

Enthält eine Karte in einem Unterverzeichnis mehr Dateien bzw. Unterverzeichnisse als erlaubt, so wird nur die maximal erlaubte Zahl dargestellt bzw. vom Treiberprogramm verwaltet.

Weiterhin erlauben die Funktionen in der derzeitigen Version des Treiberprogramms keine verschachtelten Verzeichnisnamen, wie z.B. subdir1\subdir2\subdir3. Es darf nur jeweils ein Verzeichnisname mit jedem Funktionsaufruf angegeben werden. Damit also in vorgenanntem Beispiel von subdir1 zu subdir3 gewechselt werden

kann, muss Funktion 17 zweimal aufgerufen werden, zunächst mit dem Argument subdir2 und anschließend mit subdir3.

Für die Suche nach einer Datei müssen die Dateien der Reihe nach untersucht werden, da z.Zt. keine Suchfunktionen wie z.B. findfirst und findnext zur Verfügung stehen.

7.5.1. Installation des Treiberprogramms

Das Treiber-Programm hat den Namen OSXFAT16.EXE und die Programm Nummer A00Eh. Die Installation erfolgt z.B. mit folgender INS-Datei:

```
MAXRESET board=0

MAXCONNECTCPU board=0 slot=1 layer=0

MAXINST file="osxfat16.exe" no=a00e task=65 tasktype=MAX_NI_TASK autoinit
```

Das Treiberprogramm darf nur einmal installiert werden. Die AutoInit-Prozedur muss aufgerufen werden. Das Programm kann entweder als NI-oder II-Task (dann unter IRQ 97h) installiert werden. Beide Möglichkeiten unterscheiden sich durch die Art, wie ein Medien-Wechsel erkannt und behandelt wird: Die NI-Task entdeckt den Medienwechsel durch Abfragen (Polling) der Schnittstelle im Hintergrund, die II-Task reagiert sofort auf den ausgelösten Interrupt.

Bei einem Medienwechsel werden im Treiberprogramm umfangreiche Aktionen ausgeführt. Diese nehmen u.U. mehrere Sekunden in Anspruch. Ist das Treiberprogramm als II-Task installiert, kommen während dieser Aktionen keine anderen Tasks an die Reihe. Dies ist nur in Ausnamefällen akzeptabel, so dass empfohlen wird, das Programm als NI-Task zu installieren. In diesem Fall wird das Echtzeitverhalten durch Medienwechsel nur geringfügig beeinflußt.

7.5.2. Hinweise zur Verwendung

Für PCMCIA- bzw. Compact-Flash-Karten in Slot A braucht die Funktion für die Karteninitialisierung (Funktion Nr. 2) nicht aufgerufen zu werden. Das Treiberprogramm erkennt für diese Schnittstelle automatisch das Vorhandensein einer Karte und führt die erforderlichen Aktionen selbsttätig durch.

Slot B kann nicht uneingeschränkt verwendet werden. Auf dem MAX-PC werden CPU-Resourcen von PCMCIA-Slot B, dem Parallelport und einigen Universal- Ein-/Ausgängen (GPIO21 bis GPIO31) gemeinsam genutzt. Nur eine der 3 Funktionen kann verwendet werden. Um Slot B zu nutzen, ist der Aufruf von Funktion 2 bei

eingesteckter Karte einmalig erforderlich. Erst dadurch wird Slot B initialisiert und aktiviert. Die parallele Schnittstelle und GPIO21 bis GPIO31 stehen damit nicht mehr zur Verfügung (s. auch Application Note AN081).

Durch eine Semaphore pro PCMCIA-Slot ist das Programm Multi-Tasking fähig. Greift eine Task auf eine Karte zu, während bereits ein Zugriff durch eine andere Task läuft, liefert die aufgerufene Funktion den Fehler ERR_SEMAPHORE_BUSY. Der Zugriff muss zu einem späteren Zeitpunkt wiederholt werden.

7.5.3. Aufruf der Funktionen des Treiberprogramms

Die Schnittstelle des Treiber-Programms besteht aus einer Reihe von Funktionen, die aus Anwenderprogrammen heraus aufgerufen werden können. Im folgenden ist diese Schnittstelle detailliert beschrieben. Dabei sind für jede Funktion die folgenden Parameter aufgeführt:

- insize Anzahl der Bytes, die der Funktion (in indata) übergeben werden
- indata Zeiger auf die Übergabeparameter
- outsize Anzahl der Bytes, die die Funktion (in outdata) zurückgeben soll
- outdata Zeiger auf die Rückgabeparameter

Der Funktionsaufruf sieht wie folgt aus:

```
Error = max_call_func (hCpuModul, usTask, usFunction,
                       &insize, &indata,
                       &outsize, &outdata);
```

Dabei sind indata und outdata jeweils USHORT Variable. In `usFunction` ist die bei der folgenden Funktionsbeschreibung angegebene Nummer anzugeben. Es wird immer zurückgeliefert, ob die Funktion erfolgreich ausgeführt werden konnte.

MAX-PCs bieten die Möglichkeit, zwei PCMCIA- bzw. Compact-Flash-Karten anzuschließen, so dass es beim Aufruf einiger Funktionen erforderlich ist, zu spezifizieren, welche Schnittstelle angesprochen werden soll. Zulässige Werte sind 0 und 1 für die Schnittstellen A und B.

Ist bei einem Funktionsaufruf keine Karte eingesteckt, wird die Fehlermeldung ERR_DEVICE_NOT_AVAILABLE bzw. bei solchen Funktionen, die ein Datei-Handle übergeben bekommen, ERR_INVALID_HANDLE zurück gegeben.

7.5.4. Beschreibung der Funktionen des Treiberprogramms

Prozedur 0 **Status-Routine**

Diese Prozedur darf nicht durch das Anwenderprogramm aufgerufen werden. Sie überwacht an beiden Schnittstellen des PCMCIA-Controllers, ob ein Kartenwechsel stattgefunden hat. Je nach Installation der Task geschieht das Interrupt-gesteuert oder im Polling-Betrieb.

Prozedur 1 **Programm-Initialisierung**

Diese Prozedur führt die Basis-Initialisierung durch. Sie muss beim Installieren durch das autoinit-Flag einmal automatisch aufgerufen werden. Anschliessend darf sie nicht noch einmal aufgerufen werden. Da die Prozedur keinen Fehler zurückliefern kann, werden eventuell auftretende Fehler im Parameter 0 der Task als USHORT-Wert gespeichert.

Funktion 2 **Karten-Initialisierung**

Diese Funktion muss für Schnittstelle B einmal im Anschluss an die Installation aufgerufen werden (mit einsteckender Karte). Für Schnittstelle A ist der Aufruf nicht erforderlich.

insize 2

indata Schnittstelle (0 oder 1)

outsize 0

outdata -

Funktion 3 **Datei öffnen**

Mit dieser Funktion öffnet man eine Datei. Als Ergebnis erhält man ein Handle zurück, das man für den anschließenden Zugriff auf die Datei benötigt.

Nur eine begrenzte Anzahl von Dateien darf gleichzeitig geöffnet sein. Die maximale Zahl kann aus dem Task-Parameter 4 (als UCHAR) ausgelesen werden (z.Zt. 16). Wird diese Zahl erreicht, müssen vor dem erneuten Aufruf dieser Funktion zunächst Dateien geschlossen werden.

insize sizeof (MAX_FILE_OPEN_TYPE)

indata Variable des Typs MAX_FILE_OPEN_TYPE

outsize 4

outdata Handle für die Datei. Dieses wird für alle anschließenden Dateioperationen benötigt.

Der Datentyp MAX_FILE_OPEN_TYPE ist wie folgt aufgebaut:

acName	char[80]	Dateiname als nullterminierter ASCII-String. Dieser muss der DOS-Konvention entsprechen, d.h. der Name darf max. 8 Zeichen lang sein mit max. 3 Zeichen Erweiterung. Um zu kennzeichnen, auf welcher Karte die Datei geöffnet werden soll, kann der Laufwerkbuchstabe gefolgt von einem Doppelpunkt als erstes angegeben werden. Beispiel: a:\messdata.123 öffnet die Datei ‚messdata.123‘ auf Laufwerk A (entspricht Schnittstelle 0). Wird kein Laufwerk angegeben, wird default-mäßig Schnittstelle 0 verwendet. Längere Namen werden einfach nach 8 Zeichen abgeschnitten.
usMode	USHORT	Festlegung von Eingenschaften für das Öffnen und den späteren Zugriff auf die Datei 0: die Datei wird im Hauptverzeichnis angelegt bzw. wenn dort eine Datei mit dem angegebenen Namen bereits existiert, wird diese geöffnet. Der Schreibzeiger wird auf das Ende der Datei gesetzt (d.h. Schreiben bedeutet dann, an das Ende der Datei anhängen). ≠0: Zur Zeit nicht erlaubt.

Beispiel:

```
MAX_FILE_OPEN_TYPE rcFile;

Insize  = sizeof(MAX_FILE_OPEN_TYPE);
Outsize = sizeof(ULONG);

rcFile.usMode = 0;
strcpy (rcFile.acName, „testfile.bin“);
Error = max_call_func (hMyself, FatTaskNr, 3,
                       &Insize, &rcFile,
                       &Outsize, &hFile);
```

Funktion 4 **Datei schreiben**

Mit dieser Funktion kann in eine Datei geschrieben werden. Die optimale Schreibgeschwindigkeit erzielt man, wenn die Anzahl der zu schreibenden Daten ein Vielfaches der Clustergröße (s.Funktion 12: usSectorSize * ucSectorPerCluster) ist. Die Funktion wertet die Dateiattribute aus, d.h. Read-Only-Dateien werden nicht beschrieben.

insize sizeof (MAX_FILE_WRITE_TYPE) (=16)

indata Variable des Typs MAX_FILE_WRITE_TYPE

outsize 0

outdata -

Der Datentyp MAX_FILE_WRITE_TYPE ist wie folgt aufgebaut:

ulFileHandle	ULONG	Handle für die Datei
ulSize	ULONG	Anzahl der zu schreibenden Daten
ulDataAddr	ULONG	physikalische Anfangsadresse der zu schreibenden Daten
ulMode	ULONG	z.Zt. immer =0: Beim Schreiben in eine bestehende Datei werden die neuen Daten an das Ende der Datei angehängt.

Beispiel:

```
Insize  = sizeof(MAX_FILE_WRITE_TYPE);
Outsize = 0;

FileData.ulFileHandle = Handle;
FileData.ulSize = 12345;
FileData.ulDataAddr = 0x20000;
FileData.ulMode = 0;
// Schreibe 12345 Bytes in die Datei
Error = max_call_func (hMyself, FatTaskNr, 4,
                       &Insize, &FileData, &Outsize, &Dummy);
```

Funktion 5 **Datei lesen**

Mit dieser Funktion kann eine Datei gelesen werden.

insize sizeof (MAX_FILE_READ_TYPE) (=16)

indata Variable des Typs MAX_FILE_READ_TYPE (der Aufbau entspricht MAX_FILE_ WRITE_TYPE, s.o.). In *ulSize* steht die Anzahl an Bytes, die gelesen werden sollen. In *ulDataAddr* wird die (physikalische) Anfangsadresse eingetragen, wo die gelesenen Daten hingeschrieben werden sollen. In *ulMode* steht der Offset in Bytes bezogen auf den Dateianfang, ab dem gelesen werden soll.

outsize 0

outdata -

Beispiel:

```
Insize = sizeof(MAX_FILE_READ_TYPE);
Outsize = 0;

FileData.ulFileHandle = Handle;
FileData.ulSize = 1000;
FileData.ulDataAddr = 0x20000;
FileData.ulMode = 0;

// lese 1000 Bytes ab Dateianfang der Datei
Error = max_call_func (hMyself, FatTaskNr, 5,
                       &Insize, &FileData, &Outsize, &Dummy);
```

Funktion 6 **Datei schließen**

Mit dieser Funktion wird eine geöffnete Datei geschlossen. Anschließend können keine weiteren Operationen auf die Datei durchgeführt werden. Nur eine begrenzte Anzahl von Dateien darf gleichzeitig geöffnet sein. Die maximale Zahl kann aus dem Task-Parameter 4 (als UCHAR) ausgelesen werden (z.Zt. 16).

insize 4

indata Handle für die Datei.

outsize 0

outdata -

Funktion 7 **Datei löschen**

Mit dieser Funktion wird eine geöffnete Datei gelöscht. Das Handle wird geschlossen.

insize 4

indata Handle für die Datei.

outsize 0

outdata -

Funktion 8 Medium formatieren

Diese Funktion dient zur (Neu-) Formatierung der Karte. Funktion 12 kann verwendet werden, um festzustellen, ob die Karte mit FAT16 formatiert ist. Die Funktion setzt bei Erfolg Parameter 6 bzw. 7 auf den Wert 1, um zu signalisieren, dass eine gültige Karte einsteckt.

Die Funktion schreibt auch die Partitionstabelle des Datenträgers neu. Dabei wird eine Partition über die gesamte Größe des Datenträgers angelegt und anschließend als FAT16 formatiert. Durch die Beschränkungen von FAT16 können bis maximal 2GB formatiert werden. Die Anzahl der Sektoren pro Cluster ist abhängig von der Speicherkapazität der Karte, für Dateien im Hauptverzeichnis werden fest 512 Einträge reserviert, als OEM wird „_SORCUS_“ eingetragen und eine FAT-Kopie wird angelegt.

insize 2

indata Schnittstelle (0 oder 1)

outsize 0

outdata -

Funktion 9 Sektor direkt lesen

Mit dieser Funktion kann ein einzelner Sektor der Karte gelesen werden. Die Funktion dient nur zu Testzwecken.

insize 4

indata ULONG-Variable, in deren High-Word die Schnittstelle angegeben ist (0 oder 1), und deren Low-Word die Nummer des zu lesenden Sektors enthält. Die Anzahl vorhandener Sektoren kann mit Funktion 12 ermittelt werden.

outsize 512

outdata Array, in das alle Datenbytes des Sektors eingetragen werden

Funktion 10 Information eines Verzeichniseintrags lesen

Mit dieser Funktion können die auf der Karte gespeicherten Dateien abgefragt werden. Es ist der Index des Hauptverzeichnis-Eintrags einer Datei anzugeben, zu der die Funktion dann die verfügbaren Informationen liefert. Mit Hilfe dieser Funktion ist es z.B. möglich, nach einer Datei zu suchen. Dabei müssen u.U. alle Hauptverzeichnis-Einträge durchsucht werden, da es (durch Löschen von Dateien) zu Lücken im Hauptverzeichnis kommen kann. Bei freien Hauptverzeichnis-Einträgen liefert die Funktion den ERR_DEVICE_NOT_AVAILABLE. Bei Auftreten dieses Fehlers muss *max_clear_error* aufgerufen werden, bevor die Funktion für den nächsten Eintrag erneut aufgerufen werden darf.

Beim Lesen der einzelnen Einträge des Hauptverzeichnis sind die Attribute der Datei zu beachten. So wird z.B. die Bezeichnung des Datenträgers als ein Verzeichnis-Eintrag (normalerweise Index 0) mit gesetztem Attribut-Bit 3 gespeichert. Unterverzeichnisse sind ebenfalls Einträge im Hauptverzeichnis. Dabei ist Bit 4 gesetzt.

insize 4

indata ULONG-Variable, in deren High-Word die Schnittstelle angegeben ist (0 oder 1), und deren Low-Word den Index des zu lesenden Hauptverzeichnis-Eintrags enthält. Die Anzahl von Hauptverzeichnis-Einträgen kann mit Funktion 12 ermittelt werden.

outsize sizeof(MAX_FILE16_INFO_TYPE)

outdata Variable vom Typ MAX_FILE16_INFO_TYPE, in die die Informationen des Verzeichniseintrags eingetragen werden.

Der Datentyp MAX_FILE16_INFO_TYPE ist wie folgt aufgebaut:

acName	char[14]	Dateiname als nullterminierter ASCII-String
ulSize	ULONG	Aktuelle Dateigröße
usDate	USHORT	Datum (des letzten Schreibzugriffs)
usTime	USHORT	Uhrzeit (des letzten Schreibzugriffs)
usMode	USHORT	Dateiattribute
usDrive	USHORT	Laufwerk (0 oder 1), auf dem sich die Datei befindet
usExtra	USHORT	Start-Cluster (nur für interne Zwecke benötigt)

Damit Datum und Uhrzeit gültig sind, ist es erforderlich, im Anschluss an ein Reset die Echtzeituhr auf einem MAX-PC zu stellen (max_set_date_and_time). Die Werte werden in folgendem Format geliefert:

Datum Bit `15 11 7 3 0`

Bedeutung `yyyy yyym mmmd dddd`

y enthält die Jahreszahl als Offset zu 1980, *m* den Monat und *d* den Tag.

Zeit Bit `15 11 7 3 0`

Bedeutung `hhhh hmmm mmms ssss`

h enthält die Stunde, *m* die Minute und *s* die Sekunde/2.

usMode enthält die Attribute der Datei. Die Bedeutung der einzelnen Bits entspricht DOS:

Bit	**Bedeutung**
0	1=Read-Only (die Datei kann nicht beschrieben werden)
1	1=Versteckte Datei
2	1=System-Datei
3	1=Volume-Name
4	1=Unterverzeichnis
5	1=Archivierungs-Bit
6, 7	reserviert

Beispiel:

```
/*********************************************************************/
// Implementation of the DIR command
/*********************************************************************/

MAX_FILE16_INFO_TYPE rcFileInfo;
MAX_DRIVE_INFO rcDrvInfo;

// Get number of entries in the root directory
Insize  = sizeof(USHORT);
Outsize = sizeof(MAX_DRIVE_INFO);
Error = max_call_func (hMyself, usTask, 12,
                       &Insize, &Drive,
                       &Outsize, &rcDrvInfo);

// scan entries of the root directory
for (entry=0; entry<rcDrvInfo.usRootDirSize; entry++)
{
   Insize  = sizeof(ULONG);
   Outsize = sizeof(MAX_FILE16_INFO_TYPE);
   Error = max_call_func (hMyself, usTask, 10,
                          &Insize, &entry,
                          &Outsize, &rcFileInfo);
   if (Error)
   {
      if (Error == ERR_DEVICE_NOT_AVAILABLE) // free directory entry
         max_clear_error();                  // continue
      else                                   // error has occured
         break;                              // handle it ...
   }
   else
   {
      if (rcFileInfo.usMode & 0x18)   // volume name or sub-directory
      {
         ....
      }
      else       // this is a file
      {
         // convert date and time and display file information
         day  = rcFileInfo.usDate & 0x001F;
         mon  = (rcFileInfo.usDate & 0x01E0) >> 5;
         year = ((rcFileInfo.usDate & 0xFE00) >> 9) + 1980;
         hour = (rcFileInfo.usTime & 0xF800) >> 11;
         min  = (rcFileInfo.usTime & 0x07E0) >> 5;
         sec  = (rcFileInfo.usTime & 0x001F) << 1;
         printf("%s, %d Bytes, %02d.%02d.%04d %02d:%02d:%02d
                (Attr.=%X) \n\r",
                rcFileInfo.acName, rcFileInfo.ulSize,
                day, mon, year, hour, min, sec, rcFileInfo.usMode);
      }
   }
}
```

Funktion 11 Eigenschaften einer geöffneten Datei lesen

Mit dieser Funktion können die Eigenschaften einer geöffneten Datei ausgelesen werden.

insize 4

indata Handle einer Datei

outsize sizeof(MAX_FILE16_INFO_TYPE)

outdata Variable vom Typ MAX_FILE16_INFO_TYPE, in die die Eigenschaften der Datei eingetragen werden (Aufbau siehe Funktion 10).

Funktion 12 Information über die Karte abfragen

Diese Funktion liefert Informationen über die angeschlossene Flash-Karte. Diese Funktion darf auch aufgerufen werden, wenn die Karte nicht mit FAT16 formatiert ist, dann sind alle Werte in der u.g. Struktur =0 (außer *aucManufacturer*) gesetzt. Über das Element *usFormat* kann am einfachsten überpüft werden, ob die eingesteckte Karte durch das OSXFAT16 Treiberprogramm unterstützt wird.

insize 2

indata Schnittstelle (0 oder 1)

outsize sizeof(MAX_DRIVE_INFO)

outdata Variable vom Typ MAX_DRIVE_INFO, in die die Informationen eingetragen werden.

Der Datentyp MAX_DRIVE_INFO ist wie folgt aufgebaut:

ulCapacity	ULONG	Kapazität der Flash-Karte (total)
ulFree	ULONG	Freier Speicher auf dem Medium
usRootDirSize	USHORT	Anzahl der Einträge im Hauptverzeichnis (total) = Anzahl möglicher Dateien
usSectorSize	USHORT	Anzahl Bytes pro Sektor
ucSectorPerCluster	UCHAR	Anzahl Sektoren pro Cluster
ucMediaDesc	UCHAR	Media-Descriptor
usFormat	USHORT	Format des Dateisystems (16 = FAT16)
aucManufacturer	char[8]	Hersteller (bei Formatierung mit Funktion 8 steht darin _SORCUS_)

Funktion 13 Anmelden für Benachrichtigung über Medienwechsel

Über diese Funktion meldet sich ein Anwenderprogramm dafür an, dass es bei einem Wechsel der Flash-Karte informiert werden soll. Die Benachrichtigung geschieht durch den Funktionsaufruf einer beim Anmelden anzugebenen Anwenderfunktion durch das Treiberprogramm. Bei einem Medienwechsel ruft das Treiberprogramm vorher selbstständig die Karten-Initialisierung-Funktion (Nr. 2) auf.

Zusätzlich zu der aktiven Benachrichtigung kann ein Medienwechsel auch über die Parameter 6 (für Schnittstelle 0) und 7 (für Schnittstelle 1, jeweils ein Byte) jederzeit abgefragt werden (Polling). Dort trägt das Treiberprogramm eine 1 ein, sobald eine Karte mit gültiger FAT16-Formatierung im laufenden Betrieb eingesteckt worden ist bzw. eine 2, wenn eine Karte entfernt wurde. Das Einstecken einer nicht mit FAT16 formatierten Karte führt zum Wert 3 im entsprechenden Parameter. Das Zurücksetzen (=0) des entsprechenden Parameters ist Aufgabe des Anwenderprogramms.

Bei der Programmierung der Anwenderfunktion sollte bedacht werden, dass der Aufruf der Anwenderfunktion aus der Interrupt-Routine heraus erfolgt – sofern das Programm als II-Task installiert ist. Die Interrupt-Routine des Treiberprogramms wird also um die Dauer der Anwenderfunktion verlängert.

insize 4

indata Aufzurufende Anwenderfunktion, spezifiziert durch Task-Nr. im High-Word und Funktions-Nr. im Low-Word. Die Funktion muss der Konvention von globalen OsX-Funktionen entsprechen (s. Handbuch). Das Treiberprogramm ruft die Funktion mittels *max_call_func* auf, sobald es feststellt, dass eine Karte gewechselt wurde.

outsize 0

outdata -

Die Anwenderfunktion erhält bei ihrem Aufruf einen ULONG-Parameter übergeben. In dessen High-Word wird ihr die Schnittstelle (0 oder 1) mitgeteilt, an der der Kartenwechsel stattgefunden hat. Das Low-Word kodiert die Änderung (1=neue mit FAT16-formatierte Karte, 2=Karte entfernt, 3=neue Karte mit nicht unterstütztem Format). Die Funktion darf keine Daten zurückliefern.

Beispiel für die Callback-Funktion:

```
void MAXEXPORT AlarmDummy (USHORT *pusInsize, ULONG *ulEvent,
                      USHORT *pusOutsize, void *dummy)
{
   USHORT usCard;

   max_entry_func();
   if (*ulEvent & 0x10000)
      usCard = 1;
   else
      usCard = 0;
   switch (*ulEvent & 0xFFFF)
   {
      case 1: ...    // mit FAT16 formatierte Karte eingesteckt
              break;
      case 2: ...    // Karte entfernt
              break;
      case 3: ...    // falsch bzw. nicht formatierte Karte eingesteckt
              break;
   }
   max_exit_func(ERR_OK);
}
```

Achtung: Die Angabe einer nicht vorhandenen oder falschen Funktion kann u.U. zum Absturz führen, wenn ein Medienwechsel erfolgt!

Funktion 14 Default-Callback-Funktion

Nur für interne Zwecke.

Funktion 15 Information zu einem geöffneten Handle lesen

Mit dieser Funktion können die zu einer geöffneten Datei gespeicherten Verwaltungs-Informationen ausgelesen werden. Die Struktur der Rückgabedaten ist nicht offengelegt, da die Funktion nur für interne Zwecke enthalten ist.

insize 4

indata Handle einer Datei

outsize sizeof (SftEntry)

outdata Variable vom Typ SftEntry, in die die Informationen über die Datei eingetragen werden.

Funktion 16 **Neues Verzeichnis erzeugen**

Mit dieser Funktion wird ein neues Unterverzeichnis im gerade aktuellen Verzeichnis angelegt.

insize sizeof (MAX_FILE_OPEN_TYPE)

indata Variable des Typs MAX_FILE_OPEN_TYPE

outsize 0

outdata -

Der Datentyp MAX_FILE_OPEN_TYPE ist wie folgt aufgebaut:

acName	char[80]	Verzeichnisname als nullterminierter ASCII-String. Dieser muss der DOS-Konvention entsprechen, d.h. der Name darf max. 8 Zeichen lang sein mit max. 3 Zeichen Erweiterung. Längere Namen werden einfach nach 8 Zeichen abgeschnitten.
usMode	USHORT	=0: reserviert

Funktion 17 **Verzeichnis wechseln**

Mit dieser Funktion wird in ein anderes Verzeichnis gewechselt. Mit jedem Aufruf dieser Funktion kann nur eine Verschachtelungsebene gewechselt werden. Durch Angabe von „..“ gelangt man in das übergeordnete Verzeichnis, durch „\“ in das Hauptverzeichnis.

insize Länge des Verzeichnisnamens

indata Verzeichnisname als nullterminierter ASCII-String (max. 12 Zeichen inkl. Extension)

outsize 0

outdata -

Funktion 18 **Verzeichnis löschen**

Mit dieser Funktion kann ein Unterverzeichnis des aktuellen Verzeichnis vom Datenträger entfernt werden. Dies ist nur dann möglich, wenn sich keine Dateien in dem angegebenen Verzeichnis befinden, ansonsten wird die Fehlermeldung ERR_FUNCTION_LOCKED zurückgeliefert. Der Aufruf kann nur aus der nächsthöheren Verzeichnisebene erfolgen.

insize Länge des Verzeichnisnamens

indata Verzeichnisname als nullterminierter ASCII-String (max. 12 Zeichen inkl. Extension)

outsize 0

outdata -

Funktion 19 **Aktuelles Verzeichnis ermitteln**

Diese Funktion liefert den kompletten Pfad des aktuell angewählten Verzeichnis

insize 0

indata -

outsize Länge des Strings

outdata Verzeichnisname als nullterminierter ASCII-String z.B. a:\subdir1\subdir2

7.5.5. Einige Parameter des Treiberprogramms (gültig ab Version 2.E)

Offset	Typ	Bedeutung
0	USHORT	Fehler-Code aus der auto_init Prozedur (#1)
4	UCHAR	Mögliche Anzahl gleichzeitig geöffneter Dateien (nur lesen)
6	UCHAR	Status von Karte 0 (0 nach init, 1=Karte eingesteckt, 2=Karte entfernt, 3=Karte mit falscher Formatierung eingesteckt)
7	UCHAR	Status von Karte 1 (0 nach init, 1=Karte eingesteckt, 2=Karte entfernt, 3=Karte mit falscher Formatierung eingesteckt)
14	USHORT	Timeout Wartezeit für Kartenzugriffe (Default=20)

Alle übrigen Parameter dienen nur internen Zwecken.

7.5.6. Kurzübersicht über die Funktionen des Treiberprogramms

Funktion	Bedeutung
0	Status Routine
1	Programm-Initialisierung
2	Karten-Initialisierung
3	Datei öffnen / erzeugen
4	Datei schreiben
5	Datei lesen
6	Datei schließen
7	Datei löschen
8	Karte formatieren
9	Sektor direkt lesen (intern)
10	Information zu einem Verzeichnis-Eintrag lesen
11	Datei-Information lesen
12	Karten Informationen lesen

Funktion	Bedeutung
13	Anmelden für die Benachrichtigung bei Medienwechsel
14	Default Callback Funktion (intern)
15	SFT-Eintrag lesen (intern)
16	Verzeichnis anlegen
17	Verzeichnis wechseln
18	Verzeichnis löschen
19	Aktuelles Verzeichnis ermitteln

7.5.7. Fehlercodes

Die Funktionen des Treiberprogramms liefern folgende Fehler zurück:

Fehlercode	Konstante	Bedeutung
0102h	ERR_MEMORY_ALLOCATION	RAM für interne Datenstrukturen konnte nicht reserviert werden
02E0h	ERR_INVALID_PARAMETER	Ungültiger Datei bzw. Verzeichnisname oder Lese-Offset zu gross (Funktion 5)
09E0h	ERR_NOT_IMPLEMENTED	Feature nicht verfügbar
0EE0h	ERR_NOT_ENOUGH_MEMORY	Kein Speicherplatz auf der Karte frei
11E0h	ERR_TIMEOUT	Timeout beim Hardwarezugriff
12E0h	ERR_READ_ONLY_DEVICE	Schreibzugriff nicht erlaubt
14E0h	ERR_INVALID_HANDLE	Falsches Datei-Handle
15E0h	ERR_DEVICE_NOT_AVAILABLE	Karte nicht vorhanden, bzw. angegebene Datei oder Verzeichnis existiert nicht
16E0h	ERR_WRONG_NUMBER_OF_ INPUTS	Falscher Datentyp der Übergabeparameter
17E0h	ERR_WRONG_NUMBER_OF_ RETURNS	Der vom Anwender zur Verfügung gestellte Speicher für die Rückgabedaten ist zu klein.
24E0h	ERR_OSX_BUFFER_FULL	Keine freien Einträge im aktuellen Verzeichnis
36E0h	ERR_DEVICE_NOT_READY	Initialisierung in der auto_init ist liefert Fehler
84E0h	ERR_FLASH_WRONG_STATE	Ungültige Partitionstabelle
85E0h	ERR_FLASH_NOT_SUPPORTED	Der verwendete Kartentyp wird nicht unterstützt (falsche Formatierung)
99E0h	ERR_RANGE_CROSS_OVER	Anzahl möglicher geöffneter Dateien überschritten
9CE0h	ERR_FUNCTION_LOCKED	Löschen des Verzeichnisses nicht möglich (es ist nicht leer)
A4E0h	ERR_SEMAPHORE_BUSY	Zugriff aus einer anderen Task auf die Karte läuft gerade
A9E0h	ERR_DEVICE_NOT_INIT	Karte nicht initialisiert

Fehler, die von OsX-Systemfunktions- Aufrufen zurückgegeben werden können, werden in der Regel vom Treiber direkt an den Anwender weitergereicht, so dass auch andere Fehlercodes auftreten können.

8. Modul-Device-Treiber

8.1. Allgemeines

Zur Programmierung der I/O-Devices auf den MAX-Modulen und dem MAX-PC müssen die Modul-Device-Treiber (MDD) verwendet werden. Für jedes Modul gibt es ein solches Treiberprogramm. Dieses kann entweder auf einem CPU-Modul als OsX-Task installiert werden (OsX-MDD) oder als Windows-Kernel-Mode Treiber auf dem Host-PC (PC-MDD).

Die Modul-Device Treiber gehen von einem Kanal-orientierten Ansatz aus. Der Anwender 'öffnet' einen Kanal zu einem oder mehreren Devices, z.B. einem Analog-Eingang. Beim Öffnen werden Kanal-spezifische Parameter wie z.B. der Messbereich übergeben und ein gültiges Handle zurückgeliefert. Mit dem Handle kann nun sehr einfach auf das Device zugegriffen werden (z.B. Lesen eines Analogwertes). Wird ein Kanal nicht mehr benötigt, kann er geschlossen werden. Dabei wird der vorher belegte Speicherplatz wieder freigegeben.

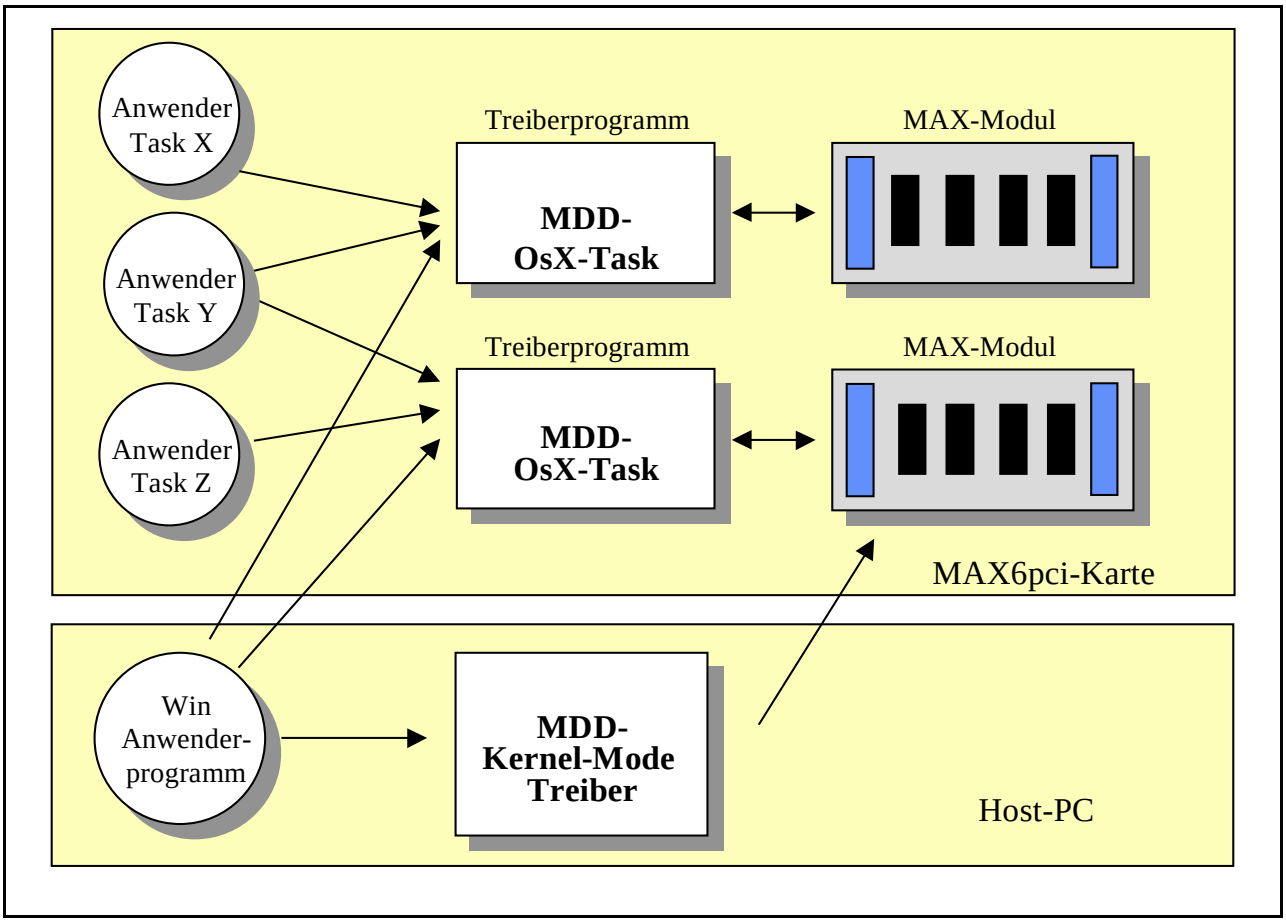
Modul-Device-Treiber sind voll Multi-Tasking-tauglich. Ein MDD verwaltet sämtliche Funktionseinheiten eines Moduls und stellt eine normierte Schnittstelle für den Zugriff auf die Funktionseinheiten zur Verfügung.

Jeder MDD stellt bestimmte Standard-Funktionen zur Verfügung, wie z.B. Versionsabfrage, Statusabfrage zu den unterstützten Devices, Infotext lesen, Reset sowie das Öffnen von Kanälen zu den Devices.

Über die geöffneten Kanäle lassen sich die Devices ansprechen, z.B. einen Wert an einen Ausgang schreiben oder einen Wert von einem Eingang lesen. Jeder Kanal kann einen Ausgabe- und einen Eingabedienst haben, er muss aber nicht beide zur Verfügung stellen. Zusätzlich zu diesen Diensten stellen Kanäle eine Reihe von Sonderdiensten für Steuerung der Kanalfunktion, für Informationsabfragen zum Kanal sowie zum Schließen der Kanals bereit. Die Zugriffe auf alle Devices auf den Modulen erfolgen nur über den MDD mit den standardisierten Bibliotheksfunktionen **max_read_channel_xxx** und **max_write_channel_xxx** (xxx = **uchar**, **ushort**, **short**, **ulong, long** oder **block**), so dass das Ersetzen eines Moduls durch ein anderes keinen Einfluss auf den Zugriff hat, nur die Parameter beim Öffnen des Kanals ändern sich unter Umständen.

Welche Funktion für ein Device jeweils zu verwenden ist, ist der Beschreibung des jeweiligen Moduls zu entnehmen.

Der Zugriff auf ein Modul kann gleichzeitig über mehrere MDDs erfolgen (siehe Abbildung). So kann z.B. ein PC-Anwenderprogramm über einen PC-MDD auf ein Modul zugreifen und gleichzeitig eine OsX-Task das Modul über einen OsX-MDD ansprechen. Ein PC-Programm kann zusätzlich auch über einen OsX-MDD auf ein Modul zugreifen.



Vorteile von MDDs

- Aufgrund der standardisierten Schnittstelle aller MDDs ist es möglich, alle Devices mit einheitlichen Bibliotheks-Funktionen anzusprechen. Die Funktionen zum Ansprechen der MDDs sind somit für alle Module gleich. Dies erleichtert das Programmieren insbesondere beim Einsatz von unterschiedlichen Modultypen.
- Ein Device kann aus beliebig vielen Anwendungen angesprochen werden.
- Falls bei einem Modul Änderungen vorgenommen werden, die das Ansprechverhalten eines Device verändern, muss nur ein Update des MDD auf dem MAX-PC oder dem Host-PC installiert werden. Bereits bestehende

Anwenderprogramme müssen nicht verändert und somit nicht nochmals übersetzt werden. Gleiches gilt für den kompletten Ersatz eines Moduls durch einen kompatiblen Nachfolgetyp.

- In Echtzeitprogrammen erfolgt der MDD-Zugriff immer direkt, d.h. ohne Zwischenschaltung des Betriebssystems.

8.2. Treiberspezifische Begriffe

8.2.1. Handle

Ein 'Handle' wird vom Treiber beim Öffnen eines Kanals für den Kanal vergeben und ermöglicht den Zugriff auf den geöffneten Kanal. Das Handle eines geöffneten Kanals ist dabei immer eindeutig, d.h. kein anderer Kanal kann ein Handle mit demselben numerischen Wert haben. Das Handle ist ein 32-Bit-Wert. Bei einem geöffneten Kanal ist es immer $\neq 0$.

8.2.2. Device

Unter den 'Devices' eines Moduls versteht man die Funktionseinheiten eines MAX-Moduls oder des MAX-PCs. Der Eingang eines Analog-Moduls ist z.B. ein Device.

Damit sich innerhalb einer Gruppe ein bestimmtes Device gezielt ansprechen lässt, muss der sogenannte **Device-Index**, angegeben werden (bestehend aus erstem und letztem Device).

8.2.3. Dienste

Die verfügbaren Funktionen eines Kanals werden als die Dienste bezeichnet.

Wenn z.B. ein Kanal zu einem Eingang geöffnet wird, so verfügt der Kanal über einen **Eingabedienst**, der bei einem entsprechenden Kanalzugriff genau diesen Eingang liest.

Wenn z.B. ein Kanal zu einem Ausgang geöffnet wird, so verfügt der Kanal über einen **Ausgabedienst**, der bei einem entsprechenden Kanalzugriff diesen Ausgang nach Ihren Wünschen setzt. Ist der Ausgang zusätzlich rücklesbar, so verfügt der Kanal auch über einen **Eingabedienst**, der beim Abruf den Ausgang zurückliest.

Unabhängig davon, ob ein Kanal zu einem Ein- oder Ausgang geöffnet wird, verfügt er auch über einige **Sonderdienste**. Mit Hilfe dieser Sonderdienste kann der Kanal gesteuert (z.B. Schließen des Kanals), parametriert oder diagnostiziert werden.

8.2.4. Kanalname

Dieser ist eine frei wählbare Bezeichnung eines Kanals. Der Kanalname hat keinen Einfluss auf die Funktionalität des Kanals und ist lediglich für Diagnosezwecke gedacht. Es besteht z.B. die Möglichkeit, dass der PC die Kanalnamen aller auf der Trägerkarte MAX6pci geöffneten Kanäle liest und auf dem Bildschirm darstellt. Der Kanalname eines Kanals besteht aus maximal 80 Zeichen.

Das Setzen und Lesen des Kanalnamens erfolgt über Sonderdienste des jeweiligen Kanals. Das Setzen erfolgt mit der Funktion **max_channel_control**, das Lesen mit **max_channel_info**.

8.2.5. Infotext

Unter 'Infotext' versteht man eine Zeichenkette (String), der in Klartext Informationen zu einem MDD liefert. Ein MDD kann über maximal 256 Infotextseiten, die jeweils maximal 256 Infotexte enthalten können, verfügen. Das Lesen eines Infotextes erfolgt mit der Funktion **max_read_infotext**, der die Seitennummer und Zeilennummer des gewünschten Infotextes übergeben werden muss.

Zur Zeit sind folgende Seiten definiert:

Seite 0 enthält Angaben zum eingesetzten Modul:

Zeile	Eintrag
0	Modulbezeichnung
1	Revision des Moduls
ab 2	Zusätzliche Informationen zum Modul

Seite 1 enthält Angaben zum Hersteller:

Zeile	Eintrag
0	Firmenbezeichnung
1	Autor
ab 2	Zusätzliche Informationen zum Hersteller

8.3. Verwaltung der Devices

Zur Verwaltung der Devices ist im zugehörigen MDD für jedes verfügbare Device ein **Device-Status-Zähler** reserviert. Dieser Zähler gibt an, wie viel Kanäle zum Device bereits geöffnet sind.

Beim Öffnen eines Kanals kann der MDD durch Setzen des Flags *_CP_EXCLUSIVE* dazu angewiesen werden, das Device **exklusiv** für den Kanal zu reservieren. Es können dann keine weiteren Kanäle mehr auf dieses Device geöffnet werden.

Der aktuelle Stand eines Device-Status-Zählers kann mit Hilfe der Funktion **max_device_status** ermittelt werden.

8.4. Datentypen

In den Beschreibungen der einzelnen MDDs ist aufgeführt, mit welchen Funktionen auf die einzelnen Kanäle zugegriffen wird und in welchem Datenformat die Daten übergeben werden. Zu jedem Kanal kann das Datenformat mit **max_channel_info** ermittelt werden.

Bei digitalen I/O s werden die Daten in der Regel rechtsbündig angegeben, d.h. Bit 0 entspricht dem niedrigsten im Kanal enthaltenen I/O-Bit. Umfasst beispielsweise ein Kanal die digitalen Eingänge 4 bis 7, steht beim Lesen der Wert von Eingang 4 im Bit 0 des beim Lesezugriff erhaltenen Bytes bzw. Wortes.

Analoge I/Os werden standardmäßig von allen MDDs als LONG-Werte (vorzeichenbehafteter 32 Bit-Wert) übergeben. Dabei entspricht der Wert 1 einem Mikrovolt (1 µV) bei Spannungs-Ein- oder Ausgängen. Bei Strom-Ein- oder Ausgängen (z.B. RANGE_20MA) entspricht der Wert 1 einem Nanoampere (1 nA). Die Verwendung dieses Formats bei allen Zugriffen auf analoge Kanäle garantiert Software-Kompatibilität auch bei einer neuen Hardware-Revision eines Moduls (z.B. der Bestückung mit einem anderen Analog-Digital-Umsetzer). Es wird dann nur eine neue MDD-Version benötigt.

Um Lese- oder Schreibzugriffe auf analoge Kanäle schneller zu machen, haben die meisten MDDs auch die Möglichkeit implementiert, die Werte direkt in dem Format zu übergeben, wie der Analog-Digital-Umsetzer bzw. der Digital-Analog-Umsetzer auf dem Modul sie erwartet. Dazu muss beim Öffnen das Flag *_CP_HW_FORMAT* gesetzt werden. In der MDD-Beschreibung ist dann das erforderliche Format der Daten dokumentiert. Bei Verwendung dieses Formats kann der Austausch eines Moduls durch einen anderen Typ (z.B. ein Analogmodul mit einer höheren Auflösung) die Anpassung des Anwenderprogramms notwendig machen.

Durch die MDD-interne Umrechnung vom Hardware-Format in das normierte Analogformat sind die Zugriffe mit dem normierten Format langsamer.

8.5. Kanaleigenschaftsstrukturen (CPS)

Eine CPS ist eine Datenstruktur, die alle erforderlichen Informationen enthält, die der Modul-Device-Treiber zum Öffnen eines Kanals benötigt. Für jeden MDD ist eine spezielle CPS definiert (siehe Dokumentation zum jeweiligen MDD in Kapitel 9).

Die Strukturelemente **.usDevice, .usIndexFirst und .usIndexLast** spezifizieren das Device (bzw. die Device-Gruppierung) zu dem (bzw. zu der) der Kanal geöffnet werden soll. Ein einzelnes Device wird ausgewählt, indem **.usIndexLast** = **.usIndexFirst** gesetzt wird., wobei **.usIndexFirst** das Device angibt. Durch Setzen von **.usIndexLast** > **.usIndexFirst** können die Daten von einer Gruppe von Devices in einem einzigen Zugriff erfasst werden.

Das Element **.usFlags** bzw. **.ulFlags** gibt sonstige Eigenschaften des Kanals an. Sollen keine sonstigen Eigenschaften angegeben werden, so muss **.usFlags** bzw. **.ulFlags** = 0 gesetzt werden. Die folgenden Flags sind für viele MDDs definiert.

Flag	**Bedeutung**
_CP_EXCLUSIVE	Der Kanal soll exklusiv geöffnet werden, d.h. zu diesem Device kann kein anderer Kanal geöffnet werden.
_CP_HW_FORMAT	Die Daten sollen in dem Format übertragen werden, das die Hardware liefert bzw. benötigt (s. Kapitel 7.4).
_CP_UNCORRECTED	Dieses Flag kann bei analogen Kanälen gesetzt werden und bewirkt, dass die vom Device ermittelten Werte nicht einer Gain/Offset-Korrektur unterzogen werden. Diese Option sollte nur für Test- oder Abgleichzwecke verwendet werden.
_CP_SYNC_CALLBACK	Kann bei Callback-Kanälen gesetzt werden. Es bewirkt, dass die Callback-Funktion direkt aufgerufen wird (ohne zeitlichen Verzug). Ist dieses Flag nicht gesetzt, wird der Aufruf gepuffert. Wenn der MDD auf dem PC läuft, ist nur dieser Mode möglich. Unter OSX erfolgt der Aufruf der Callback-Funktion auf dem als NI-Task laufenden Message-MDD, d.h. asynchron zum Ereignis.

Für die in vielen CPS-Strukturen enthaltenen Elemente **.usReadMode** und **.usWriteMode** kommen die folgenden Werte zur genauen Spezifizierung des Device-Zugriffs zum Einsatz:

Mode	**Bedeutung**
IO_MODE_DIRECT	Es wird direkt vom Eingang gelesen bzw. direkt in den Ausgang geschrieben.
IO_MODE_LATCH	Es wird aus dem Eingangspuffer (Hardware) des Moduls gelesen bzw. es wird in den Ausgangspuffer (Hardware) des Moduls geschrieben. Damit geschriebene Daten am Ausgang effektiv werden, muss ein Trigger ausgelöst werden..
IO_MODE_RAM	Es wird aus dem RAM gelesen. Dieser Mode wird dazu verwendet, um Daten, die in den Kanal geschrieben wurden und im Treiber gepuffert sind, zurückzulesen.
IO_MODE_RAM_LATCH	Für Ausgänge gilt: Es wird in das RAM geschrieben. Damit geschriebene Daten am Ausgang effektiv werden, muss per Software ein Trigger ausgelöst werden. Dazu ist zusätzlich ein entsprechender Trigger-Kanal zu öffnen.

Notizen

9. Remote-Debugging mit RTDS

9.1. Was ist RTDS?

SORCUS RTDS ("Real-Time Development Studio") ist eine Windows-Umgebung für die Entwicklung von Echtzeitprogrammen für die SORCUS-Karten Multi-LAB/2, Multi-COM, MODULAR-4/486 und MAX-PC.
Die Entwicklungsumgebung besteht aus einem komfortablen Editor, einem integrierten Debugger und einer Projektverwaltung zur automatischen Erzeugung von Make-Dateien. Im RTDS nicht enthalten sind allerdings Compiler, Linker und das Make-Tool selbst. Zur eigentlichen Programmerzeugung wird daher auf die entsprechenden Tools des Borland-C++-Compilers zurückgegriffen, empfohlen werden die Versionen 4.5, 5.0 oder 5.2, da das Make-Tool der Version 3.1 nicht sauber unter Windows arbeitet. Für Borland Pascal fehlt die Unterstützung zur Zeit noch.

9.2. RTDS konfigurieren

Wenn Sie RTDS zum ersten Mal einsetzen, müssen Sie einige Einstellungen für die Anpassung an Ihr System vornehmen. Wählen Sie dafür aus dem Menü "Project" den Menüpunkt "Settings" (s.u.).
Nehmen Sie folgende Einstellungen vor:

- Unter "General" die Nummer der SORCUS-Karte, mit der Sie arbeiten wollen, entsprechend 'SORCUS-Boards' in der Systemsteuerung.
- Unter "Connection" die Debug-Schnittstelle vom PC zur SORCUS-Karte.
- Unter "Connection" das Verzeichnis, in dem sich die Programmdateien des SORCUS-Remote-Kernels befinden (normalerweise stehen diese im RTDS-Verzeichnis).
- Unter "Tools" beim Punkt "Tools Directory" das "bin"-Verzeichnis Ihres Borland-Compilers. In diesem Verzeichnis müssen sich die Dateien "bcc.exe", "tlink.exe" und "make.exe" befinden. Falls Sie den Turbo-Assembler verwenden wollen, muss sich auch die Datei "tasm.exe" in diesem Verzeichnis befinden.
- Unter "Directories" beim Punkt "Includes" das "Include"-Verzeichnis Ihres Borland-Compilers sowie das Verzeichnis, in dem sich die SORCUS-Header-Dateien befinden.

- Unter "Directories" beim Punkt "Libraries" das "Lib"-Verzeichnis Ihres Borland-Compilers sowie das Verzeichnis, in dem sich die SORCUS-Bibliotheken befinden.

9.3. Das Project-Menü

- **New Project**:
  Wählen Sie diesen Eintrag, um ein neues Projekt anzulegen.
- **Open Project**:
  Wählen Sie diesen Eintrag, um ein bestehendes Projekt zu öffnen.
- **Close Project:**
  Wählen Sie diesen Eintrag, um das aktuelle Projekt zu schließen. Falls Sie Änderungen am Projekt durchgeführt haben, werden Sie gefragt, ob Sie diese speichern möchten.
- **Insert File Into Project:**
  Wählen Sie diesen Eintrag, um dem aktuellen Projekt bestehende Dateien hinzuzufügen. Diese Möglichkeit ist ab Version 1.C.003 implementiert.
- **Build:**
  Wenn Sie diesen Eintrag anwählen, wird automatisch eine Make-Datei (<Name des Projekts>.mak) erzeugt und das Make-Programm mit dieser Datei aufgerufen. Falls Sie seit dem letzten Aufruf dieses Eintrags Änderungen an den Quelldateien Ihres Projekts durchgeführt haben, werden vom Make-Programm automatisch Compiler und Linker aufgerufen, um eine aktuelle Version des zum Projekt gehörenden Echtzeitprogramms zu erzeugen.
- **Install:**
  Beim Anwählen dieses Menüpunkts wird die zum Projekt gehörende Installationsdatei ausgeführt. Diese Datei wird nicht automatisch erzeugt, sie muss per Hand angelegt werden!
- **Settings:**
  Beim Anwählen dieses Eintrags wird ein Dialog angezeigt, in dem Einstellungen für das aktuelle Projekt durchgeführt werden können. Die Struktur der Projekteinstellungen entspricht der der Borland-C++-IDE. Schlagen Sie zur Information über die Einstellungen bitte in der Borland-Hilfe nach. Beim Anwählen des Buttons "Set Default" werden diese Einstellungen als Default-Einstellungen für alle neuen Projekte dieses Typs (d.h. dieser Art SORCUS-Karte) verwendet.

9.4. Neues Projekt anlegen

Wählen Sie aus dem "Project"-Menü (s.o.) den Eintrag "New Project", eine Dialogbox "New Project" erscheint.
Wählen Sie Im Feld "Type" den Typ des Projekts aus. Da im Moment nur die Entwicklung von Echtzeitprogrammen mit dem Borland C++-Compiler unterstützt wird, beschränkt sich die Auswahl auf den Typ der SORCUS-Karte, auf der Ihr Programm laufen soll.
Geben Sie im Feld "Location" einen Pfad an, in dem Ihr Projekt angelegt werden soll. Wenn Sie hier nichts eintragen, wird Ihr Projekt im Verzeichnis "c:\" angelegt.
Drücken Sie jetzt den Button "Create". Nun wird in dem unter "Location" angegebenen Pfad ein Verzeichnis mit dem Namen Ihres Projekts angelegt. In diesem Verzeichnis wird eine Datei <*Name des Projekts*>.rtp angelegt, in der die Projekteinstellungen gespeichert werden. Sollte bereits ein Verzeichnis mit dem Namen des Projekts vorhanden sein, so wird dieses nicht gelöscht.
Standardmäßig sind bereits die notwendigen SORCUS-Libraries, der Startup-Code sowie eine Installationsdatei namens <*Name des Projekts*>.ins im Projektverzeichnis in dem Projekt enthalten. Die Installationsdatei selbst wird zu dem Zeitpunkt allerdings noch nicht erzeugt. Ist in dem Projektverzeichnis bereits eine Installationsdatei <*Name des Projekts*>.ins vorhanden, so wird diese auch nicht gelöscht oder verändert.
Quellcodedateien werden beim Anlegen eines neuen Projekts nicht erzeugt, Sie müssen diese selber anlegen und dem Projekt hinzufügen.

9.5. Debugger starten

Die Installation des Echtzeitprogramms, das debugged werden soll, kann aus RTDS durch die Anwahl von "Install" im Menü "Projekt" erfolgen. Die dafür benötigte Installationsdatei muss allerdings per Hand erstellt werden (siehe Anhang G).
Um das zum aktuellen Projekt gehörende Echtzeitprogramm zu debuggen, muss aus dem Menü "Debug" (s.u.) der Eintrag "Start Debugging" angewählt werden. Daraufhin wird als erstes überprüft, ob der Debug-Kernel auf der Karte vorhanden ist. Gegebenenfalls wird dieser installiert. Dann wird versucht, über den Debug-Kernel das Echtzeitprogramm auf der Karte zu finden. Wird es gefunden, so wird geprüft, ob die Version des Programms auf der Karte älter ist als die auf der Festplatte. Ist das Programm auf der Karte älter oder gar nicht vorhanden, so wird die zum Projekt gehörige Installationsdatei ausgeführt und noch mal geprüft, ob das Programm auf der Karte vorhanden und aktuell ist.
Ist jetzt das Programm in einer aktuellen Version auf der Karte vorhanden, so wird der Debugger gestartet. Der Instruction Pointer wird (erkennbar am gelben Pfeil am linken Rand) auf den Anfang der main()-Funktion gesetzt.

9.6. Das Debug-Menü

- **Start Debugging:**
  Beim Anwählen dieses Eintrags wird der Debugger gestartet. Falls sie nicht im View-Menü deaktiviert wurden, erscheinen dann die Variablen- und Registeransicht sowie die Aufrufliste. Außerdem wird der Instruction Pointer auf den Anfang der main()-Funktion positioniert.
- **End Debugging:**
  Beim Anwählen dieses Eintrags wird der Debugger beendet. Waren Variablen- und Registeransicht oder Aufrufliste dargestellt, so werden sie geschlossen.
- **Break:**
  Mit diesem Eintrag wird das Programm auf der Karte unterbrochen (noch nicht unterstützt).
- **Go:**
  Nach Anwählen dieses Eintrags läuft das zu debuggende Programm weiter, solange bis es auf einen Breakpoint trifft oder das Programmende (d.h. der Rücksprung aus der main()-Funktion) erreicht ist.
- **Trace Into:**
  Nach Anwählen dieses Eintrags wird im zu debuggenden Programm eine Quellcodezeile bzw. bei aktivem Disassembly-Fenster ein Maschinenbefehl ausgeführt. Falls hierbei eine Funktion aufgerufen wird, so wird in der ersten Zeile der aufgerufenen Funktion angehalten.
- **Step Over:**
  Wie beim Trace Into wird bei diesem Eintrag eine Quellcodezeile bzw. ein Maschinenbefehl ausgeführt. Der Unterschied besteht allerdings darin, dass bei Funktionsaufrufen erst dann angehalten wird, wenn zur aufrufenden Funktion zurückgekehrt wurde.
- **Step Out:**
  Wird dieser Eintrag gewählt, so läuft das zu debuggende Programm so lange, bis aus der Funktion, in der sich der Instruction Pointer momentan befindet, zurückgekehrt wurde.
- **Run To Cursor:**
  Nach Anwählen dieses Menüpunkts läuft das Programm so lange, bis die Quellcodezeile bzw. die Adresse, an der momentan der Cursor steht, erreicht wird.
- **Set IP To Cursor**:
  Mit diesem Eintrag wird der Instruction Pointer auf die momentane Cursorposition gesetzt, d.h. beim nächsten Go, Step Over, Trace Into oder Step Out-Befehl wird die Ausführung an der momentanen Cursorposition fortgesetzt.
- **Reset:**
  Hiermit wird das zu debuggende Programm wieder in den Anfangszustand

versetzt, d.h. die Register werden restauriert und der Instruction Pointer wird auf den Anfang der main-Funktion() gesetzt.

- **Toggle Breakpoint:**
  Ist an der momentanen Cursorposition im Editor- oder Disassembly-Fenster kein Breakpoint gesetzt, so wird einer gesetzt. Ist ein Breakpoint gesetzt, so wird er entfernt.
- **Watch Expression:**
  Nach Anwählen dieses Menüpunkts wird ein Dialog dargestellt, in dem ein Ausdruck eingetragen werden kann. Dieser Ausdruck wird dann in die Variablenansicht aufgenommen.
- **Watch Selection:**
  Ist im aktuellen Editorfenster ein Ausdruck markiert (invertiert dargestellt), so wird er in die Variablenansicht aufgenommen.

Notizen

10. Modulbeschreibungen

10.1. Einführung

10.1.1. Abkürzungen

Für die Ein- und Ausgänge der MAX-Module werden in diesem Handbuch die folgenden Abkürzungen verwendet:

AGND	Masse für analogen Ein- oder Ausgang (= Analog Ground)
AIN	Analoger Eingang
AOUT	Analoger Ausgang
DGND	Masse für digitalen Ein- oder Ausgang (= Digital Ground)
DIN	Digitaler Eingang
DIO	Digitaler Ein- und Ausgang
DOUT	Digitaler Ausgang
GND	Masse (= Ground)
INT	Interrupt-Eingang
TRIG	Trigger-Eingang

Eine mittels Bindestrich angehängte Zahl steht für die Nummer des Ein- oder Ausgangs. Die Bezeichnung 'AOUT-3' bedeutet also: Analogausgang Nr. 3.

10.1.2. Analogeingänge

Das MAX-PC System bietet Module mit Masse-bezogenen Eingängen und mit Differenzeingängen. Je nach Modul liegt der Eingangstyp fest oder kann konfiguriert werden.

10.1.2.1. Masse-bezogene Eingänge (single-ended)

Bei einem Masse-bezogenen Eingang wird die Spannung an einem Eingang gegenüber einem Bezugspunkt gemessen. Meist ist dieser Bezugspunkt die Systemmasse. Wenn die Signalquellen nicht von vornherein gemeinsame Massen besitzen, muss darauf geachtet werden, dass beim Zusammenschalten der Quellen keine Kurzschlüsse über die Masseanschlüsse auf dem Modul entstehen. Aus Gründen der Störsicherheit wird empfohlen, die Masseleitungen möglichst nahe am

Messpunkt - also am Modul - zusammenzuführen. In diesem Fall werden die Signal- und Masseleitungen der einzelnen Quellen getrennt zum Eingang geführt und dort angeschlossen. Alle Analogeingänge haben eigene Masseeingänge, die auf dem Modul miteinander verbunden sind. Bei Modulen ohne galvanische Trennung sind alle Masseeingänge außerdem mit der Masse der Trägerkarte und damit mit der PC-Masse verbunden.

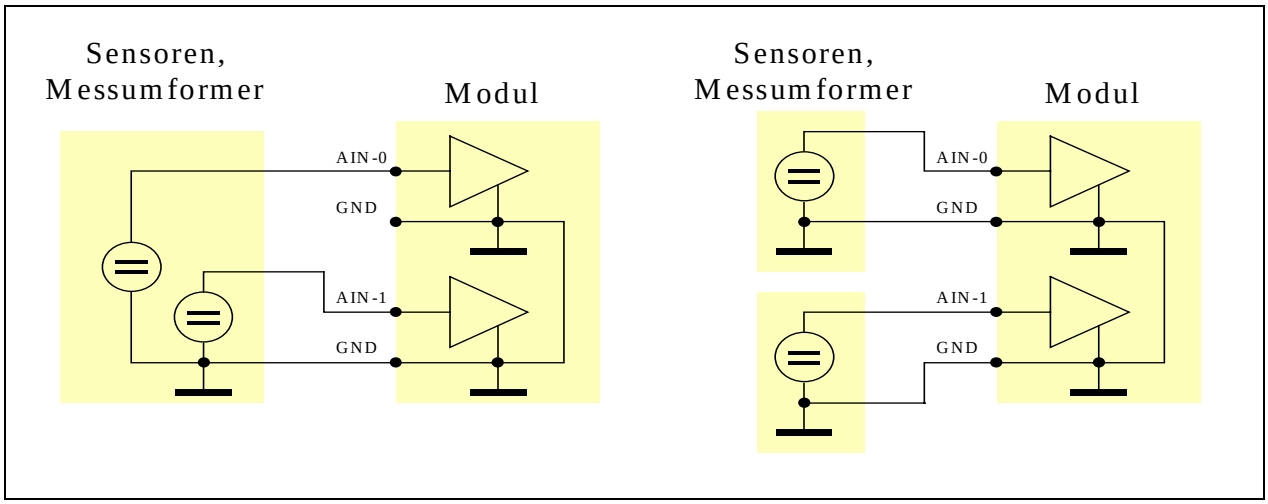


Abb. 10-1: Anschluss von zwei Signalquellen mit gemeinsamer Masse (links) oder getrennter Masse (rechts) an Masse-bezogene Eingänge.

10.1.2.2. Differenzeingänge

Bei einem Differenzeingang wird auf dem Modul die Differenz zwischen zwei Signalen gebildet und gemessen (2 Signalpole, 1 Pol für Bezugsmasse). Wichtig ist, dass auf jeden Fall die Bezugsmasse der beiden Signale angeschlossen werden muss.

Zu beachten sind zwei Punkte:

a) Die erlaubte **Differenzspannung** zwischen den beiden Signaleingängen entspricht meistens dem Eingangsbereich (siehe technische Daten des Moduls).

b) Der sog. **Common Mode Range** (Gleichtaktbereich) ist die max. Spannung zwischen einem der Signaleingänge und Masse, bei der die Messung über den gesamten Eingangsbereich noch funktioniert. So kann z.B. bei einem 5 Volt Eingangsbereich die Spannung an einem Eingang +10 Volt (gegen Masse) sein, beim anderen +5 Volt (gegen Masse). Dann beträgt die Differenz 5 Volt. Damit diese Spannung richtig gemessen werden kann, muss der Common Mode Range ≥ ±10 Volt sein. Es gibt auch Module mit einem größeren Common Mode Range. Damit kann in vielen Fällen ein Modul mit galvanischer Trennung überflüssig werden.

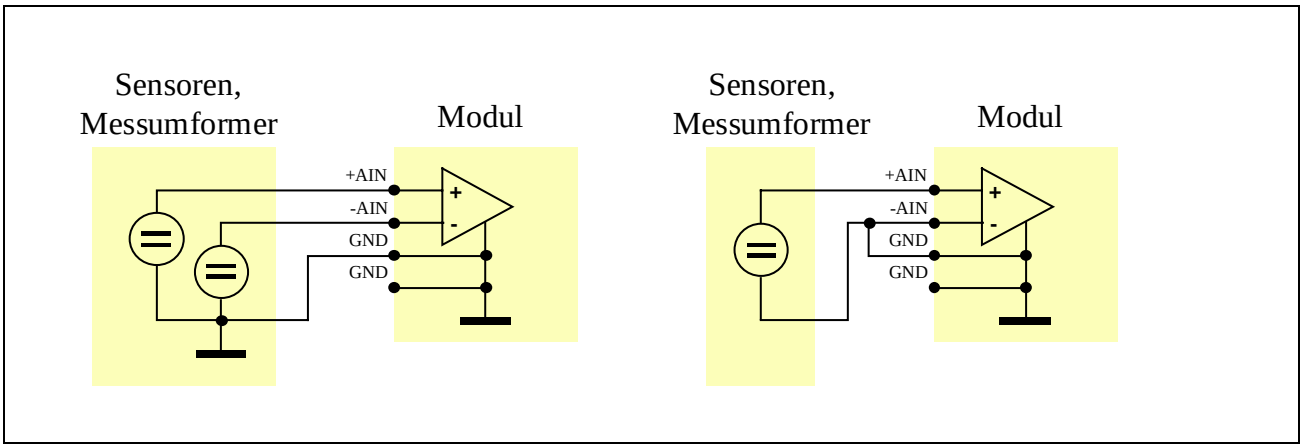


Abb. 10-2: Anschluss einer 3-poligen Signalquelle an einen Differenzeingang (links) und Anschluss einer 2-poligen Signalquelle an einen Differenzeingang (rechts)

- *Bei 2-poligen Signalquellen (z.B. bei einem Messverstärker mit galvanischer Trennung oder einer Batterie) ist eine Differenzmessung eigentlich nicht erforderlich. Wenn sie trotzdem verwendet wird, muss einer der beiden Pole zusätzlich mit einem der beiden Masseeingänge verbunden werden (siehe Abbildung 10-2, rechts).*
- *Unabhängig von der Spannungsdifferenz zwischen den beiden Signalen sind korrekte Messungen nur dann möglich, wenn keine der beiden Spannungen den Common Mode Range überschreitet. Details dazu finden Sie in den technischen Daten der einzelnen Module.*

10.1.2.3. Einschwingzeit

Alle Module des MAX-PC Systems mit analogen Eingängen verfügen jeweils über einen Analog/Digital-Wandler (abgekürzt A/D-Wandler). Um mehrere Eingänge zu realisieren, ist dem A/D-Wandler meistens ein Multiplexer vorgeschaltet, mit dem der zu wandelnde Eingang ausgewählt wird. Nach dem Umschalten auf einen Eingang dauert es eine gewisse Zeit, bis der Ausgang des Multiplexers auf die Spannung des Eingangs (= Signal) eingeschwungen ist. Diese Zeit wird als Einschwingzeit oder Settle-Time bezeichnet. Sie ist von Modul zu Modul verschieden und kann einige Mikrosekunden betragen. Bei einigen Modulen ist sie vom Ausgangswiderstand der Signalquelle abhängig, insbesondere bei großen Quellwiderständen.

Um korrekte Messungen zu erhalten, muss sichergestellt werden, dass die A/D-Wandlung erst nach Ablauf der Settle-Time gestartet wird. Einige Module verfügen zu diesem Zweck über einen sogenannten Settle-Timer. Dieser ist ein programmierbares Zeitglied, das verhindert, dass die A/D-Wandlung nach einer Eingangsumschaltung zu früh gestartet wird. Wenn ein solcher Timer nicht auf dem

Modul vorhanden ist, sorgt der Modul-Device-Treiber dafür, dass genügend Zeit zum Einschwingen bleibt.

10.1.2.4. Sample & Hold

Analog/Digital-Wandler benötigen pro Wandlung eine gewisse Zeit. In dieser Zeit darf sich die am A/D-Wandler anliegende Spannung nicht verändern. Dafür sorgt der sogenannte Sample-and-Hold-Baustein (abgekürzt mit S&H-Baustein), der den analogen Spannungswert für die Zeit der Wandlung konstant hält.

Wenn mehrere Eingänge zu einem bestimmten Zeitpunkt abgetastet werden müssen, entsteht - bedingt durch die Wandlungs- und Einschwingzeit - ein geringer zeitlicher Versatz zwischen den einzelnen ermittelten Eingangsspannungen, der nicht bei jeder Anwendung toleriert werden kann. Mit einem S&H-Baustein pro Eingang kann dieses Problem gelöst werden. Alle Eingangsspannungen werden zeitgleich in S&H-Bausteinen gespeichert und dann nacheinander gewandelt.

10.1.2.5. Messbereiche und Zahlendarstellung

Analoge Signale können in verschiedenen Messbereichen erfasst werden. Die Messverstärker dafür befinden sich vor dem A/D-Wandler. Module mit einstellbarem Messbereich erlauben es, jeden Eingang mit unterschiedlichem Messbereich zu erfassen. Beachten Sie, dass auch nach einer Messbereichsumschaltung eine gewisse Zeit vergeht, bis die Verstärker neu eingeschwungen sind.

Die Zuordnung eines von einem A/D-Wandler gelieferten Digitalwertes zu der analogen Eingangsspannung ist vom gewählten Messbereich abhängig. Die Zahlendarstellung wird von den Modul-Device-Treibern vereinheitlicht, so dass sie - unabhängig von der Hardware - für alle Module gleich ist.

10.1.2.6. Abgleich

Module mit Analogeingängen können digital abgeglichen werden.

Der Abgleich wird mit zwei **Korrekturfaktoren** durchgeführt, einer für Verstärkungsfehler (*GAIN*) und einer für Offset-Fehler (*OFFSET*). Die Durchführung des Abgleichs ist gegebenenfalls unter dem jeweiligen Modul in den folgenden Abschnitten beschrieben. Die Korrekturfaktoren werden für jedes einzelne Modul im Werk bestimmt und in das EEPROM eingetragen.

10.1.3. Analogausgänge

Analogausgänge gibt es als Strom- und als Spannungsausgänge in verschiedenen Bereichen. In der Regel können die Ausgangsbereiche für alle Ausgänge getrennt per Software eingestellt werden. Die Zahlendarstellung beim Arbeiten mit Modul-Device-Treibern ist die gleiche wie bei analogen Eingängen (siehe Seite 9-4).

Vermeiden Sie eine Fehlbeschaltung eines Ausganges. Für Spannungsausgänge finden Sie in den technischen Daten die maximale Strombelastung. Für Stromausgänge ist die minimale Größe, die ein angeschlossener Lastwiderstand haben darf, angegeben. Die Stromausgänge werden bei zu kleinem Widerstand nicht zerstört, können aber den gewünschten Strom nicht mehr liefern.

10.1.3.1. Abgleich

Ebenso wie analoge Eingänge können auch analoge Ausgänge digital abgeglichen werden. Auch hier werden die Korrekturwerte (*GAIN* und *OFFSET*) im EEPROM auf dem Modul gespeichert.

10.1.3.2. Setzen der Ausgänge

Gleichzeitiges Setzen aller Ausgänge wird dadurch erreicht, dass die digitalen Werte zunächst nacheinander für jeden Ausgang in ein Zwischenregister eingetragen werden und dann gleichzeitig in die Ausgaberegister übernommen werden (siehe Abbildung 10-3).

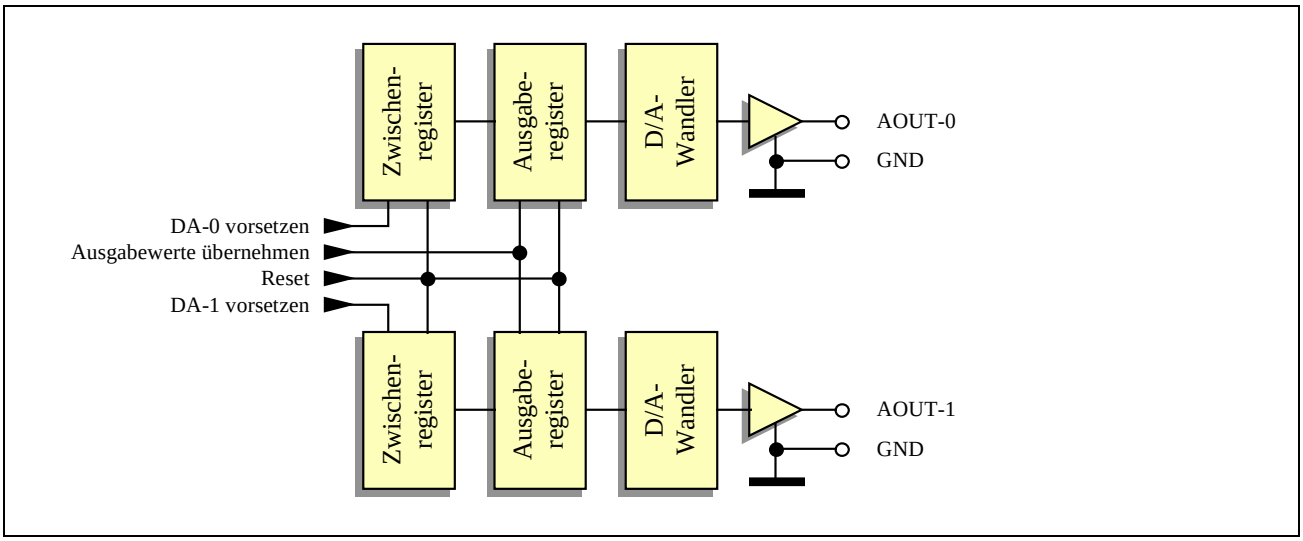


Abb. 10-3: Blockschaltbild für analoge Ausgänge (Beispiel für zwei Ausgänge)

10.1.4. Digitaleingänge

Im MAX-PC System stehen verschiedene Eingangstypen zur Verfügung: TTL-Eingänge, Schmitt-Trigger-Eingänge und opto-entkoppelte Eingänge, Zähleingänge, Inkrementalgeberinterface, Interrupt-Eingänge und Trigger-Eingänge.

Wichtige Kenngrößen von Digitaleingängen sind die Eingangspegel. Sie geben an, welche Spannung bzw. welcher Strom für eine logische Eins oder eine logische Null stehen.

10.1.4.1. TTL-Eingänge

Bei TTL-Eingängen werden **Spannungen** für die Unterscheidung der logischen Zustände benutzt. 0 bis 0,8 Volt gelten als logische Null, 2,0 bis 5,0 Volt als logische Eins. Eingangsspannungen zwischen 0,8 und 2,0 Volt erzeugen keinen vorhersehbaren Logikzustand. In den technischen Daten zu den einzelnen Modulen können Sie nachlesen, welche Pegel für das jeweilige Modul gelten. Alle Spannungen beziehen sich auf eine gemeinsame Masse, die mit der Masse der Trägerkarte verbunden ist.

Wenn z.B. ein Schalter an einen TTL-Eingang angeschlossen werden soll, ist zu beachten, dass der Eingang bei offenem Schalter nicht unbeschaltet sein darf. Unbeschaltete Eingänge verhalten sich undefiniert und sind deshalb in den meisten Fällen nicht zulässig.

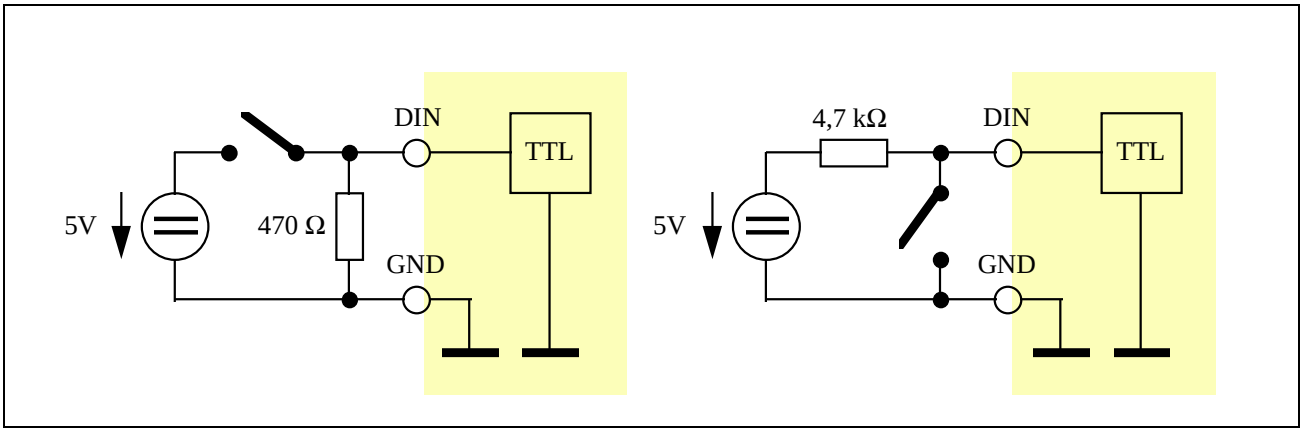


Abb. 10-4: Anschließen eines Schalters an einen TTL-Eingang

Abbildung 10-4 zeigt, wie ein Schalter an einen TTL-Eingang angeschlossen wird. In der linken Schaltung gilt ein geschlossener Schalter als logische Eins, in der rechten als logische Null. Die Widerstände sind unbedingt erforderlich, da der Eingang sonst

bei offenem Schalter nicht beschaltet ist (links) bzw. die Signalquelle bei geschlossenem Schalter kurzgeschlossen wird (rechts).

10.1.4.2. Schmitt-Trigger-Eingänge

Schmitt-Trigger-Eingänge verhalten sich ähnlich wie TTL-Eingänge. Der Unterschied besteht darin, dass es bei Schmitt-Trigger-Eingängen keinen unzulässigen Eingangsspannungsbereich gibt. Statt dessen werden zwei sogenannte Trigger-Schwellen angegeben, bei deren Über- bzw. Unterschreiten der Eingangspegel von einer logischen Null in eine logische Eins (bzw. umgekehrt) umschaltet. Die Schwellen für Über- und Unterschreiten haben unterschiedliche Werte, die Differenz wird als Hysterese bezeichnet. Andernfalls würden bei einem Eingangssignal nahe der Trigger-Schwelle kleinste Störungen genügen, um zwischen logisch Null und Eins umzuschalten.

10.1.4.3. Opto-entkoppelte Eingänge

Dieser Eingangstyp unterscheidet die logischen Zustände nicht nach Spannungen, sondern danach, ob Strom fließt oder nicht. Je nach Modul werden Ströme innerhalb eines bestimmten Bereichs (z.B. von 5 bis 10 mA) als logische Eins, schwächere Ströme als logische Null gewertet. Die jeweilige Schwelle entnehmen Sie bitte den technischen Daten der einzelnen Module.

Auf den Modulen wird mit Vorwiderständen eingestellt, bei welcher Eingangsspannung der für eine logische Eins notwendige Strom fließt.

Zur Anpassung an andere Spannungen müssen Sie entsprechende Bestückungsvarianten des gewünschten Moduls einsetzen, standardmäßig gibt es Versionen für Logik- und Prozesspegel.

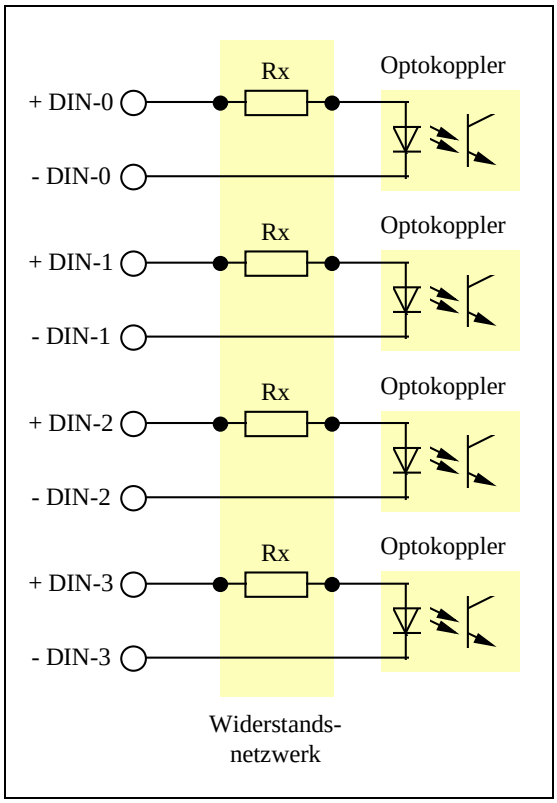


Abb. 10-5: Eingangsbeschaltung von vier benachbarten Eingängen

Bei Modulen mit **Logik-Pegeln** wird als log. 0 eine Spannung < 2,2 Volt und als log. 1 eine Spannung > 4 Volt erkannt.

Bei Modulen mit **Prozess-Pegeln** wird als log. 0 eine Spannung < 5 Volt und als log. 1 eine Spannung > 13 Volt erkannt.

10.1.4.4. Lesen von Digitaleingängen

Digitaleingänge werden einzeln oder in Gruppen gleichzeitig eingelesen. Auf einigen Modulen muss ein Steuerbefehl gegeben werden, damit eine Gruppe von Eingängen zeitgleich erfasst wird. Dadurch werden die Zustände der Eingänge zunächst in einem Zwischenspeicher (Latch) gespeichert. Anschließend werden die Daten aus dem Zwischenspeicher gelesen. Die Übernahme der digitalen Werte in das Latch kann entweder per Software erfolgen (intern) oder von einem digitalen Eingang ausgelöst werden (extern) und wird als internes bzw. externes Triggern des Latches bezeichnet.

10.1.4.5. Interrupt-Eingänge

Diese Eingänge werden für Signale benutzt, auf die sofort reagiert werden muss (zum Beispiel wenn eine Maschine an einen Endschalter stößt oder ein Bimetallschalter anzeigt, dass eine Temperaturgrenze überschritten ist). Signale an Interrupt-Eingängen werden in der Regel direkt als Interrupts zum Prozessor weitergeleitet, der dann entsprechend in einer Interrupt-Behandlungs-Routine darauf reagieren kann.

10.1.4.6. Trigger-Eingänge

Diese Eingänge werden für Signale benutzt, die auf dem Modul eine bestimmte Aktion auslösen sollen (z.B. externes Triggern zum zeitgleichen Abtasten aller Eingänge). Die auslösende Flanke kann meist per Software konfiguriert werden.

10.1.4.7. Zähleingänge

Mit diesen Eingängen werden Impulse gezählt. Zählbeginn und -ende können sowohl per Software als auch von anderen digitalen Eingängen gesteuert werden. Um während des Zählens den Zählerstand auslesen zu können und um mehrere Zählerstände gleichzeitig zu erfassen, verfügen auch die Zähler über Latches, die per Software
oder durch einen digitalen Eingang getriggert werden können.

Neben der Verwendung von Hardwarezählern gibt es auch die Möglichkeit, mit geeigneten Programmen die Impulse an digitalen Eingängen oder an Interrupt-Eingängen zu zählen. Allerdings ist die maximale Zählrate dabei durch die Prozessorgeschwindigkeit begrenzt (mehrere kHz) und die Belastung des Prozessors je nach Frequenz sehr hoch.

10.1.4.8. Inkrementalgeberinterface

Inkrementalgeber sind Sensoren, die bei Positionsänderungen Impulse abgeben. Durch die Auswertung dieser Impulse kann z.B. die Position eines Schrittmotors ermittelt werden. Dabei hängt es vom Typ des Gebers ab, wie viele Impulse er pro Umdrehung bzw. pro Millimeter abgibt. Um die Richtung der Positionsänderung zu bestimmen, werden zwei um 90 Grad phasenverschobene Rechtecksignale (Phase A und Phase B) geliefert, die bei konstanter Bewegung die gleiche Frequenz haben. Je nach Bewegungsrichtung eilt Phase A oder Phase B voraus (siehe Abbildung 10-6). Dadurch kann das Inkrementalgeberinterface erkennen, ob aufwärts oder abwärts gezählt werden muss. Jeder detektierte Impuls auf dem Inkrementalgeber erzeugt eine Rechteckperiode. Die Kombination beider Signale ermöglicht es dem Interface, die Auflösung des Inkrementalgebers über die Impulszahl hinaus zu erhöhen, indem es innerhalb einer Rechteckperiode zwei (Zweifach-Auswertung) oder vier (Vierfach-Auswertung) Zählimpulse erzeugt (siehe Abbildung 10-7).

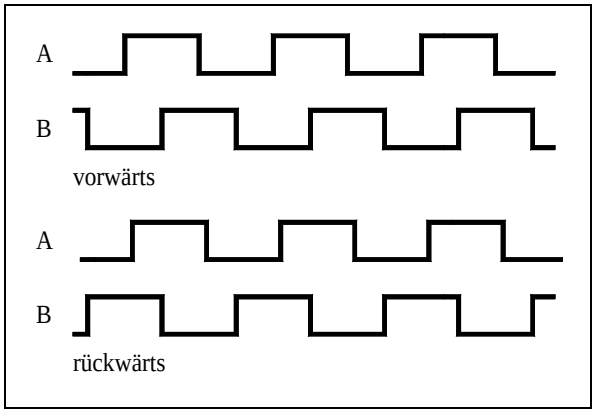


Abb. 10-6: Inkrementalgebersignale

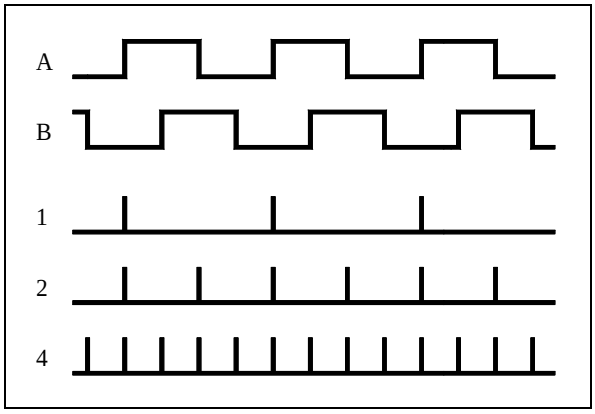


Abb. 10-7: Auswertung für Vorwärtsrichtung (ein-, zwei- oder vierfach)

10.1.4.9. Pulsbreiten-, Periodendauer- und Frequenzmessung

Durch die Kombination von Zähleingängen und zeitlich bekannten Referenzsignalen lassen sich Pulsbreiten-, Periodendauer- und Frequenzmessungen durchführen. Bei der Pulsbreitenmessung wird für die Dauer des Pulses ein vom MAX-Modul vorgegebener oder eingespeister Referenztakt gezählt. Die Periodendauermessung arbeitet nach dem gleichen Prinzip, nur dass über eine ganze Periode gezählt wird.

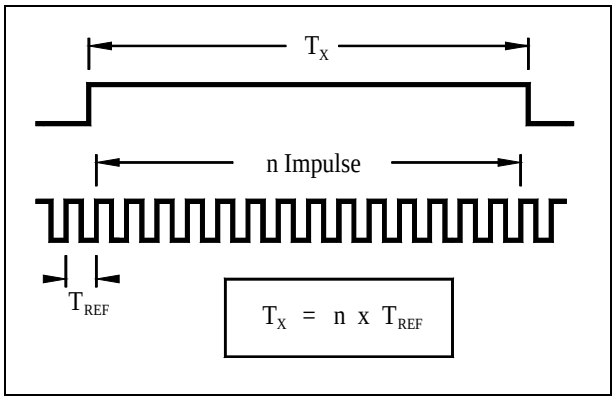


Abb. 10-8: Pulsbreitenmessung

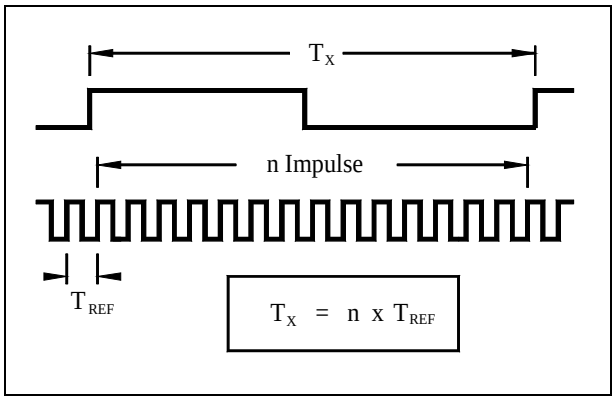


Abb. 10-9: Periodendauermessung

Bei der Frequenzmessung wird vom Modul (z.B. durch einen Timer) oder an einem Eingang des Moduls eine Zeit vorgegeben (Gate-Time), während der die Impulse des Eingangssignals gezählt werden. Je länger die vorgegebene Zeit ist, um so größer ist die Genauigkeit, um so länger dauert jedoch die Messung.

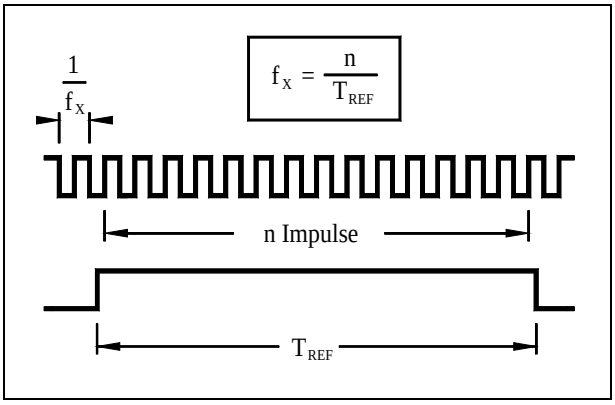


Abb. 10-10: Frequenzmessung

10.1.5. Digitalausgänge

Digitalausgänge gibt es im MAX-PC System in verschiedenen Typen: als TTL-Ausgänge, als opto-entkoppelte Open-Collector-Ausgänge und als Relais.

10.1.5.1. TTL-Ausgänge

Der TTL-Standard sieht vor, dass der Ausgangspegel bei einer logischen Null maximal 0,4 Volt und bei einer logischen Eins minimal 2,4 Volt beträgt. Die genauen Werte für die einzelnen Module und ihre Abhängigkeit vom Ausgangsstrom entnehmen Sie den jeweiligen technischen Daten. TTL-Ausgänge sind nicht für hohe Ausgangsströme ausgelegt. In der Regel sind die maximalen Ströme kaum höher als 30 mA. Allerdings können TTL-Ausgänge sowohl Strom liefern (bei logisch Eins) als auch Strom aufnehmen (bei logisch Null). Abbildung 10-11 zeigt, wie TTL-Ausgänge beschaltet werden können. Im linken Bild fließt bei einer logischen Eins ein positiver Strom durch den Lastwiderstand R_L, im rechten Bild fließt bei einer logischen Null ein negativer Strom durch R_L.

Neben den logischen Pegeln Eins und Null können TTL-Ausgänge oft noch in einen dritten Zustand geschaltet werden, der als 'hochohmig' (Tri-State) bezeichnet wird. In diesem Fall verhält sich der Ausgang so, als wäre er modulseitig nicht angeschlossen (offen). Das Schalten in den hochohmigen Zustand wird auch als 'Disable' (Sperren), das Zurückschalten als 'Enable' (Aktivieren) bezeichnet.

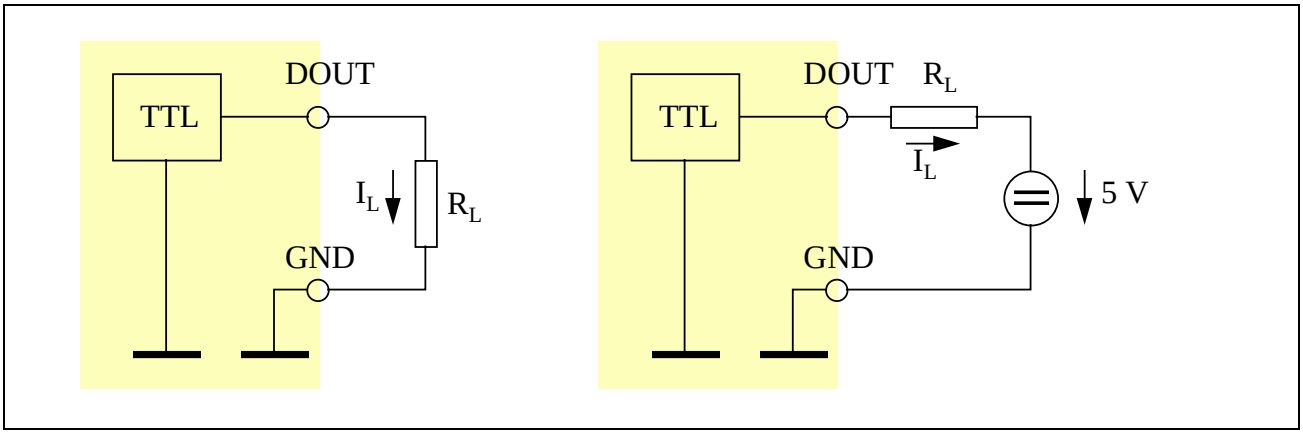


Abb. 10-11: Anschluss von TTL-Ausgängen

10.1.5.2. Opto-entkoppelte Open-Collector-Ausgänge

Bei allen opto-entkoppelten digitalen Ausgängen des MAX-PC Systems handelt es sich um Ausgänge, bei denen Kollektor und Emitter der Ausgangstransistoren für jeden Ausgang getrennt herausgeführt werden. Dadurch können alle Ausgänge vollständig unabhängig voneinander beschaltet werden. Der Ausgangstransistor kann idealisiert als Schalter betrachtet werden, wobei zwei Einschränkungen gelten:

1. Der Transistor leitet nur in einer Richtung Strom (siehe Pfeil an Emitter).
2. Im eingeschalteten (leitenden) Zustand ist die Ausgangsspannung nie exakt 0 Volt, sondern kann je nach Modul und Ausgangsstrom bis zu 1,4 Volt betragen (siehe technische Daten).

In der Regel gilt ein eingeschalteter (leitender) Transistor als logische Null, ein ausgeschalteter (nicht leitender) Transistor als logische Eins.

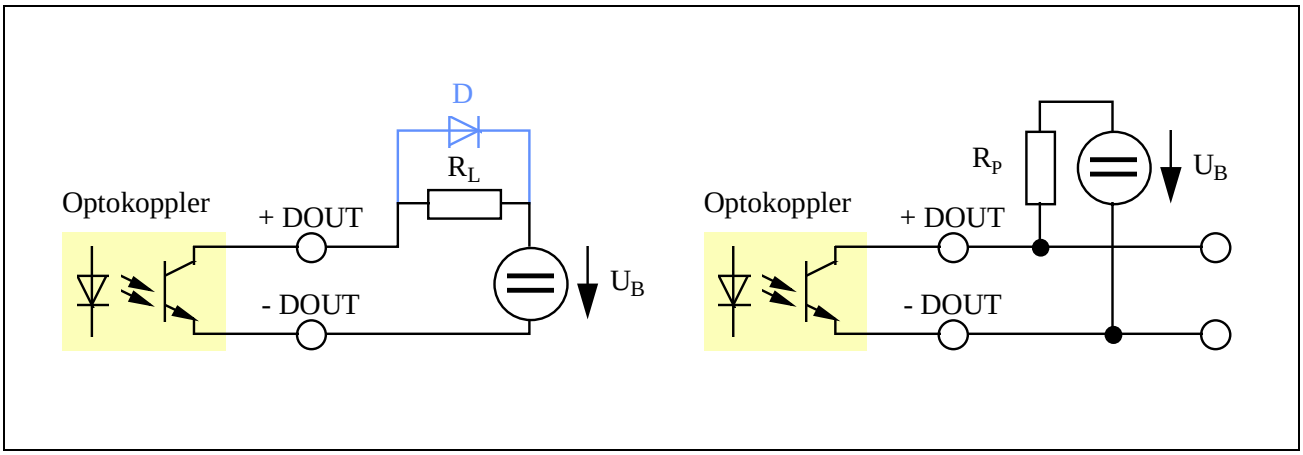


Abb. 10-12: Anschluss von opto-entkoppelten Open-Collector-Ausgängen

Abbildung 10-12 zeigt den Anschluss opto-entkoppelter Ausgänge. Das linke Schaltbild verdeutlicht, wie Lasten (z.B. Lampen oder Ventile) üblicherweise angeschlossen werden. Beachten Sie, dass beim Schalten von induktiven Lasten (z.B. Motoren und Relais) zusätzlich die Diode D zum Schutz des Ausganges notwendig ist. Die Last wird mit einer logischen Null eingeschaltet (z.B. Lampe leuchtet) und mit einer logischen Eins ausgeschaltet (z.B. Lampe aus). Beachten Sie stets, dass der maximale Ausgangsstrom und die maximale Spannung U_B nicht überschritten werden. Besonders beim Anschließen von Motoren ist zu beachten, dass der Einschaltstrom erheblich größer als der Nennstrom sein kann.

Wenn der Ausgang an einen digitalen Steuereingang eines anderen Gerätes (z.B. einer SPS) angeschlossen werden soll, kann die in Abbildung 10-12 rechts angegebene Schaltung verwendet werden. Der Ausgangspegel bei einer logischen Eins ist gleich der angelegten Spannung U_B. Bei einer logischen Null beträgt er je nach Modul zwischen 0 und 1,4 Volt und ist vom Strom abhängig, der durch R_P und U_B vorgegeben ist. R_P liegt typischerweise im Bereich von 270 Ω bis 4,7 kΩ. Beachten Sie, dass die Einschaltzeit kürzer und die Flankensteilheit größer ist, je kleiner der Widerstand R_P gewählt wird.

10.1.5.3. Relais-Ausgänge

Relaisausgänge sind einfache Umschalter mit einem gemeinsamen Umschaltkontakt. Bei logisch Null befindet sich der Schalter auf dem 'Ruhekontakt', bei logisch Eins auf dem 'Arbeitskontakt'.

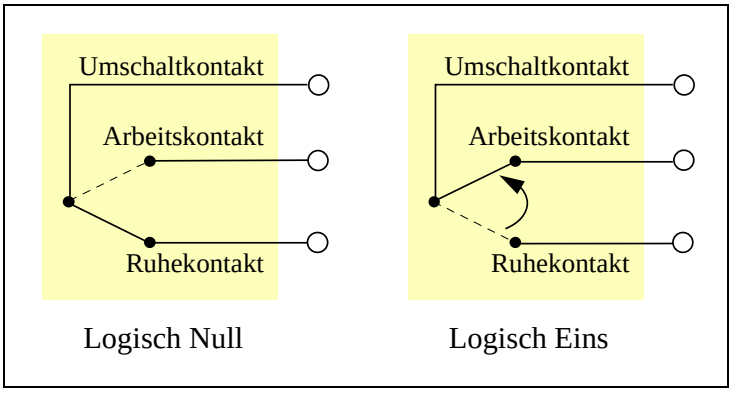


Abb. 10-13: Relaisausgang: links auf Ruhekontakt, rechts auf Arbeitskontakt

10.1.5.4. Watchdog

Digitale Ausgänge sind oft Prozesssteuerausgänge, ihre einwandfreie Funktion ist häufig sicherheitsrelevant. Um das Sicherheitsrisiko bei einem Programmabsturz zu verringern, verfügen einige Module mit digitalen Ausgängen über einen Watchdog. Diese Einrichtung muss von der Software in regelmäßigen Intervallen angesprochen (getriggert) werden, um anzuzeigen, dass das Programm noch einwandfrei läuft. Wenn auf den Watchdog innerhalb einer festgelegten Zeit (Timeout-Zeit) nicht zugegriffen wird, schaltet er alle Ausgänge ab. Das heißt bei opto-entkoppelten Ausgängen, dass alle Ausgangstransistoren stromlos (gesperrt) geschaltet werden, und bei Relais, dass alle Ausgänge auf Ruhekontakt geschaltet werden. Sie müssen eine Anlage also so auslegen, dass sie in diesem Zustand sicher ist.

Wenn Ihre Anwendung nicht sicherheitskritisch ist, können Sie den Watchdog auch abschalten.

10.1.6. EEPROM

Alle Module verfügen über einen nichtflüchtigen Speicher, ein sogenanntes EEPROM, in dem Initialisierungs-, Konfigurations- und Korrekturdaten abgelegt sind.

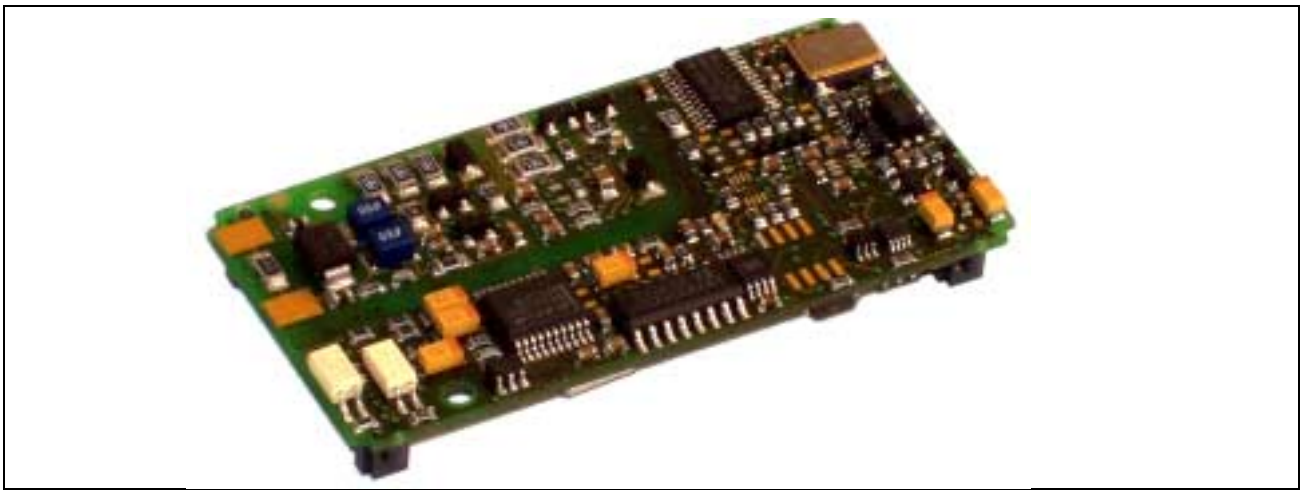
10.1.7. Testen der Module mit SNW32

Alle Module können mit dem Programm SNW32 getestet werden. Doppelklicken Sie dazu auf das entsprechende Modul im Konfigurationsbaum. Ein Assistent erlaubt Ihnen nun den Zugriff auf die Konfiguration und Test des ausgewählten Moduls.

Notizen

X-56K-FU
X-33K-FU

Modem V.90, V.34 mit UART,
Sprach-CODEC, Funkuhr



10.2. X-56K-FU und X-33K-FU

Inhaltsverzeichnis

10.2.1. Funktionsbeschreibung

Das Modem enthält einen Modem-Chipsatz für Übertragungsraten bis zu 56kBit/s (X-56k-FU) bzw. 33,6kBit/s (X-33k-FU). Für die Ein- und Ausgabe von Sprache steht bei einigen Modul-Varianten ein Codec zur Verfügung, der Audiosignale mit 8bit Auflösung und 8khz Samplerate verarbeitet. Dabei ist ein Anschluss von einem Lautsprecher (8 Ohm, 500mW, mono), einem Headset (Kondensatormikrophon, mono und Kopfhörer 32 Ohm, stereo) sowie einem Handapparat (analoges Telefon) gleichzeitig möglich.

Zusätzlich steht eine serielle Schnittstelle mit einem 16550-kompatiblen UART (TL16C752) und einem umschaltbaren Treiber (RS232/RS485) mit RTS und CTS zur Verfügung.

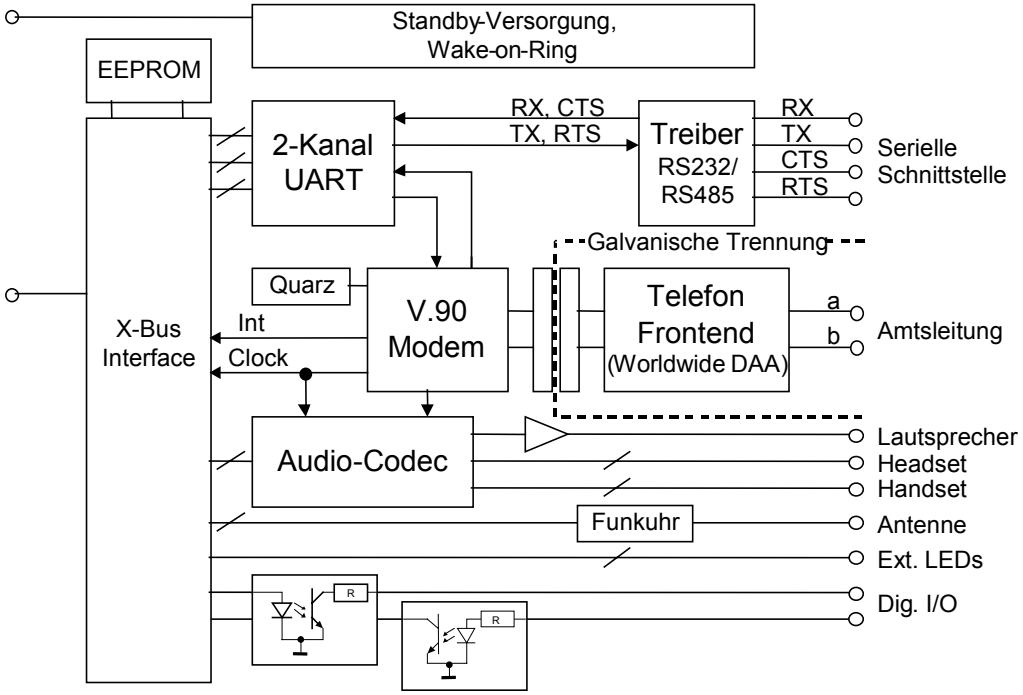
Für die Signalisierung des Modem-Status können extern 5 LEDs angeschlossen werden, mit denen dann der Zustand der Modem-Verbindung angezeigt werden kann. Die LEDs sind über Konfigurationsregister abschaltbar, so dass Strom gespart werden kann. Für digitale Signale ist jeweils ein Eingang und ein Ausgang an den Stecker geführt.

Zusätzlich ist auf dem Modul eine DCF-77 Funkuhr untergebracht, für die eine externe Antenne angeschlossen werden muss.

10.2.1.1. Modul-Versionen

Typ	Sub-typ	Version	Übertragungs-rate	Sonstiges	CODEC	Temp.-bereich	Bestell-Nr.
98	3	iX-33k-FU	33,6kBit/s	UART, Funkuhr	Nein	-40..85°C	FM-3365
98	0	X-56k-FU	56kBit/s	UART, Funkuhr	Ja	0..70°C	FM-2714
98	4	X-14k-1	14,4kBit/s	-	Nein	0..70°C	FM-3970

10.2.1.2. Blockschaltbild



10.2.1.3. Galvanische Trennung

Auf dem Modul ist das Modem-Frontend galvanisch vom X-Bus getrennt. Die Schaltung genügt den Tests nach UL1950.

Die übrigen Anschlüsse, insbesondere auch der Anschluss für den Handapparat, sind nicht galvanisch getrennt. Der digitale Ein- und Ausgang ist zum Schutz des FPGA über Optokoppler geführt.

10.2.2. Modemteil

Das Modem bietet alle mit PC-Modems vergleichbaren Übertragungsstandards und Protokolle. Das Line-Interface ist durch Programmierung an alle weltweit genutzten Telefonstandards anpassbar. Die Steuerung des Wahlvorganges und der weiteren Modemeinstellungen geschieht über AT-Befehle. Für die optimale Anpassung an neue Übertragungsprotokolle ist die Firmware des DSP des Modems per Software updatebar.

Während des Verbindungsaufbaus meldet das Modem den Status mit *BUSY*, *CONNECT (rate)*, *NO ANSWER*, *NO CARRIER*, *NO DIALTONE*, *OK*, *RING* und *RINGING* zurück. Zur Überwachung des Modem-Status können extern 5 LEDs angeschlossen werden. Die LEDs zeigen die Zustände *Senden*, *Empfangen*,

Trägersignal erkannt, *Ankommender Ruf* und *Online* an. Die Ansteuerung der LEDs kann über Konfigurationsregister deaktiviert werden.

Besondere Eigenschaften:	
Übertragungsprotokolle	V.90, V.34, V.32bis, V.32, V.22bis, V.21 und Bell 103
Fehlerkorrektur und Datenkomprimierung	V42, V42bis und MNP2-5
Übertragungssteuerung	Erweiterter AT-Befehlssatz
Weiter Eigenschaften	Firmware updatebar Rufnummernanzeige (Caller-ID nach US Bellcore und ETSI) Erkennung parallelgeschalteter Telefone (on-Hook und Off-Hook) Sprachübertragung (Samplerate 8kHz, µLaw-Codiert) Anpassung an weltweite Telefonnetze

Zusammen mit dem Codec, der auf den Modems im kommerziellen Temperaturbereich (X-56k-FU und X-33k-FU) vorhanden ist, kann das Modem auch als Anrufbeantworter, für Sprachmitteilungen und Sprachsteuerung verwendet werden. Für die Sprachein- und Ausgabe kann entweder ein Headset (Kopfhörer-Mikrophon-Kombination) oder ein Handset (analoges Telefon) verwendet werden. Bei allen Modem-Versionen ist der Anschluss eines externen Lautsprechers möglich, über den der Verbindungsaufbau kontrolliert werden kann.

Ohne den Codec ist nur die Wiedergabe von gespeicherten Sprachdateien möglich, wenn diese mit 8bit Auflösung, 8kHz Samplerate in A-Law Codierung vorliegen. Damit sind z.B. gesprochene Statusmeldungen des X-Bus Systems möglich.

10.2.3. Funkuhr

Auf dem Modem ist ein DCF-77 Empfänger integriert. Extern muss nur noch eine passende Antenne angeschlossen werden. Eine Nutzung der Funkuhr mit anderen Sendern ist nur nach Umrüstung des Empfängers möglich.

10.2.4. Modul-Device-Treiber

Zur Drucklegung dieses Handbuchs war die Beschreibung noch nicht verfügbar. Die aktuelle Version finden Sie unter www.sorcus.com.

10.2.5. Anschlusspins des Moduls (bezogen auf Stecker A)

Pin	Bedeutung
1	Telefonleitung a
2	Telefonleitung b
3-16	Dürfen nicht belegt werden
17	/CTS, Handshake 2. Serielle Schnittstelle
18	RXD, Empfangsdaten 2. Serielle Schnittstelle
19	TXD, Sendedaten 2. Serielle Schnittstelle
20	/RTS, Handshake 2. Serielle Schnittstelle
21	Anschluss TX-LED Anode
22	Anschluss RX-LED Anode
23	Anschluss DCD-LED Anode
24	Anschluss RI-LED Anode
25	Anschluss Off-Hook-LED Anode
26	Digitaler Ausgang, Open-Collector, max. 50mA
27	Digitaler Eingang, Optokoppler-LED, max. 50mA
28	Masse
29	Lautsprecher -
30	Lautsprecher +
31	Headset Mikrophon +
32	Headset Mikrophon -
33	Headset Kopfhörer rechts
34	Headset Kopfhörer links
35, 36	Masse
37	DCF77 Antennenanschluss 1
38	DCF77 Antennenanschluss 2
39	Handapparat +
40	Handapparat -

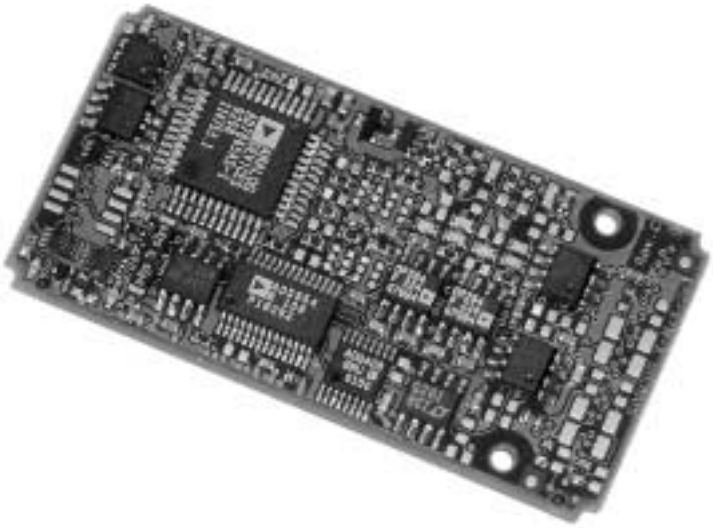
10.2.6. Technische Daten

Parameter	Randbedingungen	min.	typ.	max.	Einheit	Anm.
Modem						
Übertragungsrate	X-56k-FU, senden dto., empfangen X-33k6-FU, senden dto., empfangen			33,6 k 56 k 33,6 k 33,6 k	Bit/s	
FIFO-Buffer				64	Byte	
Codec						
Samplingrate	Aufnahme und Wiedergabe		8		kHz	
Impedanz Lautsprecher		8			Ohm	
Impedanz Kopfhörer			32		Ohm	
Sonstiges						
Digitaler Eingang	Eingangswiderstand Spannung Low-Pegel Spannung High-Pegel	 2,5	330	 1,5 12	Ohm V V	
Digitaler Ausgang	Ausgangswiderstand Ausgangsstrom Spannung bei 2mA	47	 0,5	 50	Ohm mA V	
ESD-Schutz				2000	V	
Versorgungsspannungen des Moduls			+3,3 +12		V V	
Versorgungsstrom (ohne Last an den Ausgängen)	für +3,3 V für +12 V		105 20		mA mA	1 2
Temperaturbereich Betrieb	X-56k-FU, X-33k6-FU	0		+70	°C	
	iX-56k-FU, iX-33k6-FU	–40		+85	°C	
Lagerung		–40		+85	°C	

Anm. 1: zuzüglich ca. 20mA, wenn die externen LEDs geschaltet sind, **Anm. 2**: der Strom auf der +12V-Leitung fliesst nur dann, wenn ein angeschlossener Handapparat abgenommen wird (Telefon-Schleifenstrom), was nur bei den Modulen mit Codec möglich ist.

X-5B-1

1 Analogeingang, 1 Analogausgang, 14 digitale Ausgänge zur Ankopplung von 5B-Umformern



10.3. X-5B-1

Inhaltsverzeichnis

10.3.1. Beschreibung

Das Modul ermöglicht die Erfassung und Ausgabe von analogen Signalen, die über 5B-Umformer aufgenommen werden. Es dient zur Ansteuerung mehrerer analoger 5B-Multiplexer-Panels (19"“Trägerkarten 5Bx02 oder 5BA32).

Der analoge Eingang und der analoge Ausgang besitzen jeweils eine Auflösung von 14 Bit bei einem Spannungsbereich von –5V bis +5V. Die Wandlungszeit des analogen Eingangs beträgt 5µs.

Die digitalen Ausgänge sind TTL-kompatibel.

10.3.2. Modul-Device-Treiber

10.3.2.1. Installation

Der Modul-Device-Treiber für OsX hat die Programmnummer 802eh und den Dateinamen mx5b1.exe. Der Modul-Device-Treiber für Windows hat den Namen mx5b1.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x802e, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=802e

10.3.2.2. Kanaleigenschaftsstruktur CPS_X5B1

Die CPS für das Modul hat den Namen CPS_X5B1.

10.3.2.3. Analoge Eingänge

Mit dem Modul können 64 analoge Eingänge über folgende CPS angesprochen werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AIN_DIFF*	Messung von AIN
.usIndexFirst	*0..63*	Adresse des ersten Eingangs
.usIndexLast	*0..63 (usIndexLast ≥ usIndexFirst)*	Adresse des letzten Einganges
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*0*	Kein Schreibzugriff
.usFlags	*0*	Keine Bedeutung

Strukturelement	Werte	Bedeutung
	_CP_EXCLUSIVE *_CP_UNCORRECTED*	Der Zugriff auf die Eingänge erfolgt exklusiv Die Meßdaten werden nicht korrigiert
.usRange	*RANGE_BIP_5V*	Eingangsbereich +/-5V
.usOversampling	*0..65535*	Anzahl der Messungen zur Mittelwertbildung
.usSettleTime	*0..65535*	Länge des ‚Read Enable' Pulses in µs

Eingabedienst

Der Zugriff auf einen Einzelkanal erfolgt mit

- **max_read_channel_long**

Bei Blockzugriffen muß die Größe des Datenpuffers durch `sizeof(LONG)` teilbar sein.

Der Rückgabewert enthält jeweils die gemessene Spannung in µV.

Bei Verwendung des Flags _CP_HW_FORMAT[1] erfolgt der Zugriff mit

- **max_read_channel_ushort**

Für den entsprechenden Blockzugriff muß die Größe des Datenpuffers durch `sizeof(USHORT)` teilbar sein.

10.3.2.4. Analoge Ausgänge

Mit dem Modul können 64 analoge Ausgänge über folgende CPS angesprochen werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AOUT*	Analoger Ausgang
.usIndexFirst	*0..63*	Adresse des ersten Eingangs
.usIndexLast	*0..63 (usIndexLast ≥ usIndexFirst)*	Adresse des letzten Einganges
.usReadMode	*IO_MODE_RAM*	Lesen nur aus dem RAM
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0* *_CP_EXCLUSIVE* *_CP_UNCORRECTED*	Keine Bedeutung Der Zugriff auf die Eingänge erfolgt exklusiv Die Meßdaten werden nicht korrigiert

[1] Die Daten werden im Zweierkomplement geliefert: 0h entspricht +0V; ; 1fffh entspricht +5V; 2000h entspricht –5V 3fffh entspricht –0V

Strukturelement	Werte	Bedeutung
.usRange	*RANGE_BIP_5V*	+/- 5V Ausgangsbereich
.usSettleTime	*0..65535*	Länge des ‚Write Enable' Pulses in µs

Eingabe- und Ausgabedienst

Der Zugriff auf einen Einzelkanal erfolgt mit

- **max_write_channel_long** und
- **max_read_channel_long**.

Bei Blockzugriffen muß die Größe des Datenpuffers durch `sizeof(LONG)` teilbar sein.

Der Übergabewert für **max_write_channel** muß die gewünschte Ausgangsspannung in µV enthalten.

max_read_channel liefert den zuletzt geschriebenen Wert zurück. Falls in das betreffende Device noch kein Wert geschrieben wurde, wird der Fehler ERR_BUFFER_EMPTY zurückgegeben.

Bei Verwendung des Flags _CP_HW_FORMAT[2] erfolgt der Zugriff mit

- **max_write_channel_ushort**.

Für den entsprechenden Blockzugriff muß die Größe des Datenpuffers durch `sizeof(USHORT)` teilbar sein.

10.3.2.5. Triggern der analogen Ausgänge

Da die analogen Ausgänge auf einem 5B-Panel ihren Ausgabewert wieder verlieren, müssen sie regelmäßig neu gesetzt werden. Dies kann über einen separaten Trigger-Kanal geschehen, der wie folgt zu initialisieren ist:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TRIGGER*	Kanal zu einem digitalen Ausgang
.usIndexFirst	*0..63*	Adresse des ersten Eingangs
.usIndexLast	*0..63 (usIndexLast ≥ usIndexFirst)*	Adresse des letzten Einganges
.usReadMode	*0*	Kein Lesezugriff
.usWriteMode	*0*	Kein Lesezugriff
.usFlags	*0*	Keine Bedeutung

[2] Die Daten werden im folgenden Format erwartet: 3fffh -> -5V, 1fffh -> 0V, 0h -> +5V

Strukturelement	Werte	Bedeutung
.usRange	*RANGE_BIP_5V*	+/- 5V Ausgangsbereich
.usSettleTime	*0..65535*	Länge des ‚Write Enable' Pulses in µs

Ausgabedienst

Das Setzen der gewählten Kanäle erfolgt mit

- **max_trigger_channel**.

Dabei werden alle in der CPS angegebenen Kanäle nacheinander neu beschrieben. Ein Ausgang wird nur neu gesetzt, wenn bereits ein Wert geschrieben wurde und zur Zeit ein Schreibkanal darauf geöffnet ist.

10.3.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

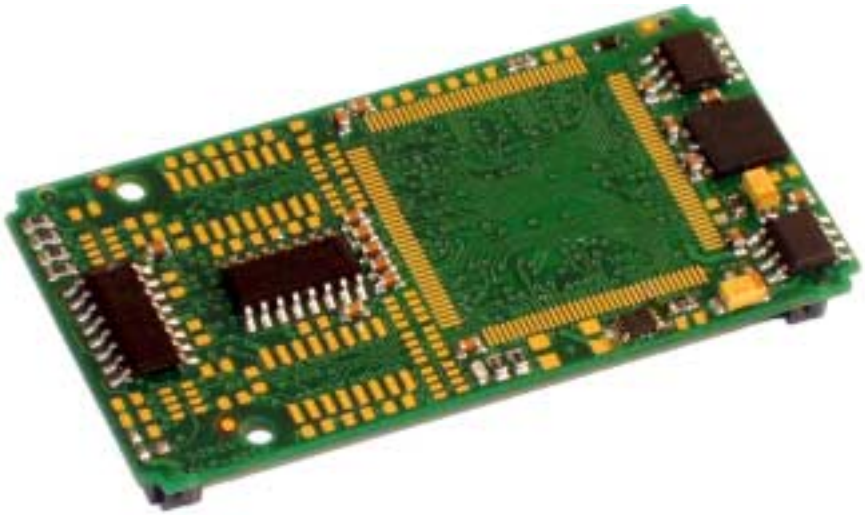
Pin	Funktion	Bedeutung
1	AIN0	pos. Analogeingang für Kanal A
2	AGND0	Analog-GND
3	AOUT	Analoger Ausgang
4	AIN1	neg. Analogeingang für Kanal A
5	AGND1	Analog-GND
6	GND	Modul-GND
7..12	RA0..RA5	Read-Address
13..18	WA0..WA5	Write-Address
19	/RE	Read-Address Enable (active low)
20	/WE	Write-Address Enable (active low)
21..40		i.c. (nicht anschließen!)

10.3.4. Besondere Eigenschaften

Parameter	Wert	Einheit
Analogeingänge, Anzahl	1	
Auflösung	14	Bit
Wandlungszeit	5	µs
Genauigkeit		LSB
Analogausgänge, Anzahl	1	
Auflösung	14	Bit
Einschwingzeit	1	µs
Genauigkeit		LSB
Digitalausgäng, Anzahl	14	
Spannungsbereich	0..3,3	V

X-5Bx64-8

Digitales I/O-Modul zum Anschluss
von bis zu 8 5Bx64-Panels



10.4. X-5Bx64-8

Inhaltsverzeichnis

10.4.1. Beschreibung

Das Modul X-5Bx64-8 stellt alle erforderlichen Kontroll- und Datenleitungen zur Ansteuerung von bis zu 8 5Bx64-Panels, 3 Timer und 5 unabhängige digitale TTL-Ein-/Ausgänge zur Verfügung.

Das Digital-I/O-Panel 5Bx64 bzw. 5Bx64i stellt 64 digitale galvanisch getrennte Leitungen zur Verfügung. Die 64 Leitungen des 5Bx64 können hardwaremäßig als Ein- oder als Ausgänge konfiguriert werden. Beim 5Bx64i sind alle 64 Leitungen digitale Eingänge.

Der Anschluss an die Ein- oder Ausgänge des Panels erfolgt über Schraubklemmen. Das 5Bx64 Panel ist für 19" Montage mit den Einbaurahmen AC1363 vorgesehen. Es benötigt eine Versorgungsspannung von 5 Volt.

Vier der Eingangsleitungen können per Jumper auf Interrupt-Leitungen des MAX-PC (X-MAX-1 bzw. X-MAX-E) durchgeschaltet werden.

Die Eingangszustände aller 64 Leitungen können gleichzeitig erfasst werden. Dieses kann per Software getriggert werden oder von einem externen Signal, das auf einen Eingang des 5Bx64 Panel gelegt wird. Dieses Signal kann zugleich einen Interrupt auf dem MAX-PC auslösen.

Das Panel 5Bx64 verfügt außerdem über einen eigenen Watchdog-Timer und über eine Spannungsüberwachung (nicht beim 5Bx64i). Im Falle einer Störung werden die Ausgänge aller Leitungen in den hochohmigen Zustand geschaltet. Eine aufgetretene Störung auf dem Panel, z.B. ein kurzzeitiger Ausfall der Spannungsversorgung, wird gespeichert und kann abgefragt werden.

Die für den Anschluss der 5Bx64-Panels nicht benötigten Pins können einzeln als digitale Ein- oder Ausgänge programmiert werden.

Außerdem verfügt das Modul über eine per Software schaltbare on-board LED.

Insgesamt kann das Modul 7 Interrupts auslösen. Vier davon kommen von bestimmten digitalen Eingängen der 5Bx64-Panels an die Modul-Pins A5..A8 (= INT-1, INT-7, INT-4 und INT-3). Das ist auf den angeschlossenen 5Bx64-Panels per Jumper J2 einstellbar. Drei Interrupts kommen von den 3 Timern auf dem Modul.

Jeder der 7 Eingänge des Interrupt-Controllers kann einen Interrupt bei einer positiven, negativen oder bei beiden Flanken auslösen. Wenn ein Interrupt noch nicht

bedient wurde und der nächste Interrupt am selben Eingang auftritt, wird ein Interrupt-Overrun im Modul gespeichert.

Einschränkungen mit Modul X-5Bx64-8, Rev. B:

Mit der Leiterplatte X-DIO-40, Rev. B ist die Adressleitung BA3 fest auf 0 gesetzt. Dadurch können zur Zeit nur vier 5Bx64-Panels je Modul angeschlossen werden. Bei Rev. C der Leiterplatte ist diese Einschränkung beseitigt.

10.4.2. Modul-Device-Treiber

10.4.2.1. Installation

Der Modul-Device-Treiber für das OsX hat die Programmnummer 802ah und den Dateinamen m5bx648.exe. Der Modul-Device-Treiber für Windows hat den Namen mx5bx648.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd(hModul, 1, 0, 0, 0x802A, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=802A

10.4.2.2. Kanaleigenschaftsstruktur CPS_X5BX648

Die CPS für das Modul hat den Namen CPS_X5BX648.

10.4.2.3. Digitale Eingänge des Moduls

Um auf einen der fünf lokal auf dem Modul vorhandenen digitalen Eingänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DIN*	Kanal auf ein digitalen Eingang
.usIndexFirst	*35, 36, 37, 38, 39*	Nummer des Eingangs
.usIndexLast	*.usIndexFirst*	Nummer des Eingangs
.usAddress	*X5BX648_MODULE_ADDRESS*	Der Eingang befindet sich auf dem Modul
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*0*	Reservierter Parameter
.usFlags	*0* *_CP_EXCLUSIVE*	Keine Bedeutung Zugriff erfolgt exklusiv

Eingabedienst

Der Datentyp des Kanals ist DATA_UCHAR. Der Zugriff auf das Device erfolgt mit:

- **max_read_channel_uchar**

10.4.2.4. Digitale Ausgänge des Moduls

Um auf einen der fünf lokal auf dem Modul vorhandenen digitalen Ausgänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DOUT*	Kanal auf einen digitalen Ausgang
.usIndexFirst	*35, 36, 37, 38, 39*	Nummer des Ausgangs
.usIndexLast	*.usIndexFirst*	Nummer des Ausgangs
.usAddress	*X5BX648_MODULE_ADDRESS*	Der Ausgang befindet sich auf dem Modul
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0* *_CP_EXCLUSIVE*	Keine Bedeutung Zugriff erfolgt exklusiv

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR. Die Zustände der Ausgänge können mit einem Lesebefehl zurückgelesen werden. Der Zugriff auf das Device erfolgt mit:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

10.4.2.5. Digitale Eingänge der Panels

Um auf jeweils acht digitale Eingänge eines 5Bx64-Panels zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DIN*	Kanal auf einen digitalen Eingang
.usIndexFirst	*0,8,16,24,32,40,48,56*	Nummer des ersten Eingangs
.usIndexLast	*.usIndexFirst+7*	Nummer des letzten Eingangs
.usAddress	*4 ... 11*	Adresse des 5Bx64 Panels
.usReadMode	*IO_MODE_LATCH*	Lesezugriff über den internen Zwischenspeicher
.usWriteMode	*0*	Reservierter Parameter

Strukturelement	Werte	Bedeutung
.usFlags	*0* *_CP_EXCLUSIVE*	Keine Bedeutung Zugriff erfolgt exklusiv

Eingabedienst

Die Zustände der digitalen Eingänge werden immer aus einem internen Zwischenspeicher (Latch) auf dem Panel entnommen. Um die Eingangszustände per Software-Befehl in diesen Zwischenspeicher zu übertragen, steht der Trigger-Kanal *DEVICE_TRIGGER* zur Verfügung.

Der Datentyp des Kanals ist DATA_UCHAR. Der Zugriff auf das Device erfolgt mit:

- **max_read_channel_uchar**

10.4.2.6. Digitale Ausgänge der Panels

Um auf jeweils acht digitale Ausgänge eines 5Bx64-Panels zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DOUT*	Kanal zu einem digitalen Ausgang
.usIndexFirst	*0,8,16,24,32,40,48,56*	Nummer des ersten Ausgangs
.usIndexLast	*.usIndexFirst+7*	Nummer des letzten Ausgangs
.usAddress	*4 ... 11*	Adresse des 5Bx64 Panels
.usReadMode	*IO_MODE_LATCH*	Lesezugriff über den internen Zwischenspeicher
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0* *_CP_EXCLUSIVE*	Keine Bedeutung Zugriff erfolgt exklusiv

Eingabe- und Ausgabedienste

Wenn die Ausgangszustände zurückgelesen werden, werden sie aus einem Latch entnommen, daher muss zuvor ein Latchbefehl über den Triggerkanal erfolgen.

Der Datentyp des Kanals ist DATA_UCHAR. Der Zugriff auf das Device erfolgt mit:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

10.4.2.7. Abtast-Trigger der Panels

Das Einlesen der digitalen Eingänge bzw. das Rücklesen der digitalen Ausgänge der Panels erfolgt immer aus den Zwischenspeichern (Latches). Zur Übernahme der aktuellen Leitungspegel in diese Zwischenspeicher, muss ein Abtast-Trigger ausgelöst werden. Dieser Abtast-Trigger kann durch einen externen Impuls am Panel oder per Software ausgelöst werden.

Das Triggern per Software erfolgt für alle angeschlossenen Panels gleichzeitig. Um auf den Abtast-Trigger zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TRIGGER*	Kanal auf einen Trigger
.usIndexFirst	*X5BX648_SAMPLE_TRIGGER*	Abtast-Trigger
.usIndexLast	*X5BX648_SAMPLE_TRIGGER*	Abtast-Trigger
.usAddress	*0*	Es werden alle Panels getriggert
.usReadMode	*0*	Reservierter Parameter
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0*	Reservierter Parameter

Ausgabedienst

Der Datentyp des Kanals ist DATA_VOID. Beim Zugriff auf den Kanal mit

- **max_trigger_channel**

werden die aktuellen Zustände der Eingänge in den Zwischenspeicher übertragen.

10.4.2.8. Watchdog-Trigger der Panels

Das Panel 5Bx64 verfügt über einen eigenen Watchdog, der (falls aktiviert) innerhalb einer bestimmten Zeit nachgetriggert werden muss. Die Aktivierung und Nachtriggerzeit wird auf dem Panel per Jumper eingestellt. Das Nachtriggern erfolgt für alle angeschlossenen Panels gleichzeitig. Um auf einen Watchdog-Trigger zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TRIGGER*	Kanal auf einen Trigger
.usIndexFirst	*X5BX648_WATCHDOG_TRIGGER*	Watchdog-Trigger
.usIndexLast	*X5BX648_WATCHDOG_TRIGGER*	Watchdog-Trigger
.usAddress	*0*	Es werden alle Panels getriggert
.usReadMode	*0*	Reservierter Parameter
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0*	Reservierter Parameter

Ausgabedienst

Der Datentyp des Kanals ist DATA_VOID. Beim Zugriff auf den Kanal mit

- **max_trigger_channel**

werden die Watchdogs aller angeschlossenen Panels nachgetriggert.

10.4.2.9. Status des Panels

Um auf den Status des 5Bx64-Panels zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Kanal auf ein Kontroll-Device
.usIndexFirst	*X5BX648_STATUS*	Status-Information
.usIndexLast	*X5BX648_STATUS*	Status-Information
.usAddress	*4 ... 11*	Adresse des 5Bx64-Panels
.usReadMode	*IO_MODE_LATCH*	Lesezugriff über den internen Zwischenspeicher
.usWriteMode	*0*	Reservierter Parameter
.usFlags	*0*	Reservierter Parameter

Eingabedienst

Der Datentyp ist DATA_USHORT. Der Status eines Panels kann mit einem Lesezugriff mit

- **max_read_channel_ushort**

ermittelt werden.

Mögliche Werte beim Lesen sind:

Wert	Bedeutung
X5BX648_STATUS_OK	Alles in Ordnung. Die Ausgänge sind aktiviert.
X5BX648_STATUS_WARNING	Die Versorgungsspannung des Panels ist zwischenzeitlich ausgefallen, ist jetzt aber wieder in Ordnung. Die Ausgänge wurden durch das Lesen dieser Statusinformationen wieder aktiviert.
X5BX648_STATUS_ERROR	Die Versorgungsspannung des Panels ist ausgefallen und/oder der Watchdog wurde nicht nachgetriggert. Die Ausgänge wurden deaktiviert.

10.4.2.10. LED des Moduls

Das Modul verfügt über eine lokale LED. Um auf diese zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_LED*	Kanal zu einer LED
.usIndexFirst	*0*	Nummer der LED
.usIndexLast	*0*	Nummer der LED
.usAddress	*X5BX648_MODULE_ADDRESS*	Die LED befindet sich auf dem Modul
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0*	Reservierter Parameter

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR. Der Zugriff auf das Device erfolgt mit:

- **max_read_channel_uchar**
- **max_write_channel_uchar**

10.4.2.11. Timer

Die drei 16-Bit Timer (0, 1 und 2) sind 8254-kompatibel (= Timer-Chip in PCs). Für jeden kann eine von 5 Betriebsarten gewählt werden. Alle Timer werden mit dem on-board Quarzoszillator von 1 MHz getaktet. Die Pins A1..A34 des Moduls werden mit Pin 1..34 der 5Bx64-Panels verbunden. Timer 0 liefert sein Ausgangssignal an Pin 3 der 5Bx64-Panels (= TA) und kann dazu dienen (wenn Jumper J2: 11-12 aufgesteckt ist), alle 5Bx64-Eingänge gleichzeitig auf allen angeschlossenen 5Bx64-Panels zu latchen. Gleichzeitig kann dabei ein Interrupt ausgelöst werden, der das Auslesen der

Latches übernimmt. Die Timer 1 und 2 können als zusätzliche Timer eingesetzt werden.

10.4.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Funktion (5Bx64)	Pin (X-5BX64-8)	Funktion (5Bx64)	Pin (X-5BX64-8)
GND	1	BA4	21
GND	2	BA5	22
TA	3	BA6	23
- (n.c.)	4	- (n.c.)	24
INT-1	5	DIR	25
INT-7	6	/WE	26
INT-4	7	/RE	27
INT-3	8	/LI	28
BD0	9	WDI	29
BD1	10	CLK	30
BD2	11	VB	31
BD3	12	CHECKOUT	32
BD4	13	WDST	33
BD5	14	CHECKIN	34
BD6	15	PINA35	35
BD7	16	PINA36	36
BA0	17	PINA37	37
BA1	18	PINA38	38
BA2	19	PINA39	39
GND (BA3)	20	GND	40

10.4.4. Besondere Eigenschaften

Parameter	Wert	Einheit
Anzahl digitaler Eingänge	7	-
davon interruptfähig	4	-
Anzahl digitaler Ausgänge	15	-
Anzahl bidirektionaler digitaler Ausgänge	13	-
Eingangsspannung (andere Standards auf Anfrage)		
(kompatibel mit 5V TTL und 5V, 3,3V und 2,5V CMOS)		
log. 0	< 0,8	V
log. 1	> 2,0	V
Input Leakage Current, max.	10	µA
Ausgangsspannung		
(kompatibel mit 5V TTL und 3,3V CMOS)		
log. 0, max. (IOL = 8mA)	0,4	V
log. 1, min. (IOH = -4mA)	2,4	V
Ausgangstrom log. 0, max.	-12	mA
log. 1, min.	+12	mA
Überspannungsfestigkeit der Eingänge	-0,5 .. +5,5	V
für Impulse < 10 ns und < 200mA	-2,0 .. +7,0	V
Timer, Anzahl	3	-
Auflösung	16	Bit
Betriebsarten (z.B. Rechteckgenerator, Ratengenerator, Impulsgenerator)	6	-
Temperatur-Bereich, Betrieb	0 .. +70	°C
optional (bitte anfragen)	-40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	8,85	g
Stromaufnahme (3,3V) min./max.	370/tbd	mA
(die X-Bus Spannungen ±12V werden nicht benötigt)		

X-AD14-20
X-AD12-16

20 analoge Eingänge, bis 14 Bit Auflösung
und max. 3 Msps Abtastrate

10.5. X-AD14-20 und X-AD12-16

Inhaltsverzeichnis

10.5.1. Beschreibung

Diese Module stellen bis zu 20 externe analoge Eingänge zur Verfügung. Jeder der Eingänge kann als Masse-bezogener Kanal oder mit einem anderen Eingang zusammen als Differenzkanal gemessen werden. Die Umschaltung kann je Kanal per Software vorgenommen werden. Maximal sind also 20 Masse-bezogene oder 10 Differenz-Kanäle möglich. Außerdem kann jeder der Kanäle per Software auf 8 Eingangsspannungsbereiche (von ±10 Volt bis ±250 mV) eingestellt werden. Eine Ausnahme hiervon macht das Low-Cost Modul X-AD12-16/L, das nur zwei Eingangsspannungsbereiche zur Verfügung stellt. Die verschiedenen Module unterscheiden sich durch die Auflösung und die max. Wandlungsrate (0,4 Msps bis 3 Msps).

Folgende Bestückungsvarianten sind lieferbar:

Typ	**Sub-typ**	**Modul-Version**	**Ein-gänge**	**Auf-lösung**	**max. Abtastrate**	**Bemerkung**
36	1	X-AD14-20/F	20	14	2,2 Msps	Temperatur
36	2	X-AD14-20/M	20	14	0,8 Msps	Low-Power Mode, Temperatur
36	3	X-AD14-20/S	20	14	0,4 Msps	Low-Power Mode, Temperatur
36	7	X-AD12-16/L	16	12	0,8 Msps	Eingangsbereich ±2,5V und ±10V
36	8	X-AD12-16/5	16	12	0,8 Msps	Eingangsbereich ±2,5V und ±5V

Einige dieser Module bieten einige Sonderfunktionen, z.B. kann die on-board Temperatur zwischen –40°C und +125°C gemessen werden. Für Eich- und Abgleichzwecke kann außerdem die Offsetspannung an mehreren Stellen der analogen Verstärkerkette, z.B. des Differenzverstärkers und des A/D-Wandlers gemessen werden. Zur Bestimmung der Settle-Time bei Kanalumschaltung stehen zwei Kanäle mit +10 Volt und –10 Volt zur Verfügung. Der on-board Settle-Timer erspart umständliche Programmierung. Er ist programmierbar von 0 bis 65536ns in Schritten von 1ns. Zu beachten ist, dass bei anderen Bestückungen für die größten Eingangsbereiche die Settle-Time auch vom Quell-Widerstand der zu messenden Spannungsquelle abhängig ist und berücksichtigt werden muss.

Bestückungsvarianten größter Eingangsbereich

Es gibt das Modul auch als Bestückungsvarianten mit 0..20 mA Eingängen oder mit einem max. Eingangsbereich größer ±10 Volt, z.B. ±50 Volt oder ±100 Volt. Auf Wunsch sind auch andere Bereiche je Kanal möglich.

Größter Eingangsbereich	Modul-Version
±10 Volt	/x/10 (Standardversion)
±20 Volt	/x/20
±50 Volt	/x/50
±100 Volt	/x/100
0..20 mA	/x/i

10.5.2. Modul-Device-Treiber

10.5.2.1. Installation

Der Modul-Device-Treiber für das OsX hat die Programmnummer 8024h und den Dateinamen mxad1420.exe. Der Modul-Device-Treiber für Windows hat den Namen mxad1420.sys. Dies gilt für alle Subtypen des Moduls. Diese Beschreibung gilt ab der MDD Version 1.F.001.

Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8024, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8024

10.5.2.2. Kanaleigenschaftsstruktur CPS_XAD1420

Die CPS für das Modul hat den Namen CPS_XAD1420 bzw. CPS_XAD1420_A.

CPS_XAD1420_A wurde gegenüber CPS_XAD1420 um die Elemente usHWOversampling und usSWOversampling erweitert.

10.5.2.3. Analoge Eingänge (Differenz)

Das Modul bietet 10 bzw. 8 (X-AD12-16) analoge Differenzeingänge, die über folgende CPS angesprochen werden können:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AIN_DIFF*	Kanal zu einem analogen Differenzeingang
.usIndexFirst	X-AD14-20: *0 ... 9* X-AD12-16: *0 ... 7*	Nummer des ersten Eingangs
.usIndexLast	X-AD14-20: *0 ... 9* X-AD12-16: *0 ... 7*	Nummer des letzten Eingangs
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usFlags	*0*	Die Daten werden im normierten, korrigierten Datenformat zurückgeliefert. Der Zugriff ist nicht exklusiv.
	_CP_EXCLUSIVE	Wie bei *0*, nur dass der Zugriff exklusiv erfolgt.
	_CP_UNCORRECTED	Die Werte werden keiner internen Korrektur unterzogen. In der momentanen Version des Treibers muss dieses Flag gesetzt werden, sofern das Flag *_CP_HW_FORMAT* nicht gesetzt ist.
	_CP_HW_FORMAT	
	_CP_FORCE_BLOCK	Es werden direkt die vom Wandler gelieferten Rohwerte (siehe 9.2.2.6.) zurückgegeben (das Flag *_CP_UNCORRECTED* hat keine Auswirkungen). Dieses Flag legt fest, dass die Rückgabewerte mit der Funktion *max_read_channel_block* gelesen werden müssen, auch wenn *usIndexFirst == usIndexLast* ist.
.usRange	*RANGE_BIP_10V*	Bipolar ±10 V (nicht bei X-AD12-16/5)
	RANGE_BIP_5V	Bipolar ±5 V (nicht bei X-AD12-16/L)
	RANGE_BIP_2V5	Bipolar ±2,5 V
	RANGE_BIP_2V	Bipolar ±2 V (nicht bei X-AD12-16)
	RANGE_BIP_1V25	Bipolar ±1,25 V (nicht bei X-AD12-16)
	RANGE_BIP_1V	Bipolar ±1 V (nicht bei X-AD12-16)
	RANGE_BIP_500MV	Bipolar ±500 mV (nicht bei X-AD12-16)
	RANGE_BIP_250MV	Bipolar ±250 mV (nicht bei X-AD12-16)
.usSettleTime	*0 ... 65535*	Settle-Time in Vielfachen von 1ns.
.usHWOversampling	*1, 2, 4, 8, 16, 32*	Erst ab Modul Rev. F verfügbar! Der Wert gibt an, über wieviele Messungen eine Mittelung durchgeführt werden soll. Bei einem Wert =1 wird der Kanal nur einmal gemessen. Die Messungen und Mittelungen erfolgen direkt per Hardware.
.usSWOversampling	*1 ... 65535*	Erst ab Modul Rev. D verfügbar! Der Wert gibt an, über wieviele Messungen der MDD eine Mittelung durchführen soll. Bei einem Wert =1 wird der Kanal nur einmal gemessen.

Eingabedienst

Wenn das Flag *_CP_HW_FORMAT* gesetzt ist, ist der Datentyp des Kanals DATA_SHORT:

- **max_read_channel_short** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Wenn das Flag *_CP_HW_FORMAT* nicht gesetzt ist, ist der Datentyp des Kanals DATA_LONG:

- **max_read_channel_long** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Wenn das Flag *_CP_FORCE_BLOCK* und das Flag *_CP_HW_FORMAT* gesetzt sind, erfolgt der Zugriff mit:

- **max_read_channel_block** (Mehrkanal)

Wenn das Flag *_CP_FORCE_BLOCK* gesetzt ist und das Flag *_CP_HW_FORMAT* nicht gesetzt ist, erfolgt der Zugriff mit:

- **max_read_channel_block** (Mehrkanal)

10.5.2.4. Analoge Eingänge (massebezogen)

Das Modul bietet 20 bzw. 16 (X-AD12-16) analoge Eingänge (massebezogen), auf die über folgende CPS zugegriffen werden kann:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AIN_SE*	Kanal zu einem analogen massebezogenen Eingang
.usIndexFirst	X-AD14-20: *0 ... 19* X-AD12-16: *0 ... 7, 10 ... 17*	Nummer des ersten Eingangs
.usIndexLast	X-AD14-20: *0 ... 19* X-AD12-16: *0 ... 7, 10 ... 17*	Nummer des letzten Eingangs
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usFlags	*0*	Die Daten werden im normierten, korrigierten Datenformat zurückgeliefert. Der Zugriff ist nicht exklusiv.
	_CP_EXCLUSIVE	Wie bei *0*, nur dass der Zugriff exklusiv erfolgt.
	_CP_UNCORRECTED	Die Werte werden keiner internen Korrektur unterzogen. In der momentanen Version des Treibers muss dieses Flag gesetzt werden, sofern

Strukturelement	Werte	Bedeutung
	_CP_HW_FORMAT	das Flag *_CP_HW_FORMAT* nicht gesetzt ist.
	_CP_FORCE_BLOCK	Es werden direkt die vom Wandler gelieferten Rohwerte (siehe 9.2.2.6.) zurückgegeben (das Flag *_CP_UNCORRECTED* hat keine Auswirkungen). Dieses Flag legt fest, dass die Rückgabewerte mit der Funktion *max_read_channel_block* gelesen werden müssen, auch wenn *usIndexFirst* == *usIndexLast* ist.
.usRange	*RANGE_BIP_10V* *RANGE_BIP_5V* *RANGE_BIP_2V5* *RANGE_BIP_2V* *RANGE_BIP_1V25* *RANGE_BIP_1V* *RANGE_BIP_500MV* *RANGE_BIP_250MV*	Bipolar ±10 V (nicht bei X-AD12-16/5) Bipolar ±5 V (nicht bei X-AD12-16/L) Bipolar ±2,5 V Bipolar ±2 V (nicht bei X-AD12-16) Bipolar ±1,25 V (nicht bei X-AD12-16) Bipolar ±1 V (nicht bei X-AD12-16) Bipolar ±500 mV (nicht bei X-AD12-16) Bipolar ±250 mV (nicht bei X-AD12-16)
.usSettleTime	*0 ... 65535*	Settle-Time in Vielfachen von 1ns
.usHWOversampling	*1, 2, 4, 8, 16, 32*	Erst ab Modul Rev. F verfügbar! Der Wert gibt an, über wieviele Messungen eine Mittelung durchgeführt werden soll. Bei einem Wert =1 wird der Kanal nur einmal gemessen. Die Messungen und Mittelungen erfolgen direkt per Hardware.
.usSWOversampling	*1 ... 65535*	Erst ab Modul Rev. D verfügbar! Der Wert gibt an, über wieviele Messungen der MDD eine Mittelung durchführen soll. Bei einem Wert =1 wird der Kanal nur einmal gemessen.

Eingabedienst

Wenn das Flag *_CP_HW_FORMAT* gesetzt ist, ist der Datentyp des Kanals DATA_SHORT:

- **max_read_channel_short** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Wenn das Flag *_CP_HW_FORMAT* nicht gesetzt ist, ist der Datentyp des Kanals DATA_LONG:

- **max_read_channel_long** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Wenn das Flag *_CP_FORCE_BLOCK* und das Flag *_CP_HW_FORMAT* gesetzt sind, erfolgt der Zugriff mit:

- **max_read_channel_block** (Mehrkanal)

Wenn das Flag *_CP_FORCE_BLOCK* gesetzt ist und das Flag *_CP_HW_FORMAT* nicht gesetzt ist, erfolgt der Zugriff mit:

- **max_read_channel_block** (Mehrkanal)

10.5.2.5. High-Speed-Messung eines Einzelkanals (massebezogen/Differenz)

Unter Verwendung der folgenden CPS kann eine High-Speed-Messung eines Einzelkanals durchgeführt werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AIN_SE* *DEVICE_AIN_DIFF*	Kanal zu einem analogen Differenz- oder massebezogenen Eingang
.usIndexFirst	siehe Analoge Eingänge	Nummer des ersten Eingangs
.usIndexLast	*.usIndexFirst*	Nummer des letzten Eingangs
.usReadMode	*IO_MODE_DIRECT* *IO_MODE_MULTIPLE*	Direkter Lesezugriff Der Treiber liefert mehrere Messwerte des Kanals, die im Treiber nacheinander erfasst werden.
.usFlags	*_CP_HIGH_SPEED*	High-Speed-Messung
.usRange	siehe Analoge Eingänge	Messbereich
.usSettleTime	*0*	Reservierter Parameter

Anmerkungen

Mit diesem Mode ist die schnellstmögliche Abtastrate des Moduls erreichbar. In diesem Mode liefert der Kanal ohne Umrechnung die vom Wandler gelesenen Rohwerte (siehe 9.2.2.6.) zurück. Vor Beginn und nach dem Ende von Messungen muss mit Hilfe von Sonderdiensten die Messung „verriegelt“ werden.

Eingabedienst

Der Datentyp des Kanals ist DATA_SHORT. Ist *.usReadMode* = *IO_MODE_DIRECT*, erfolgt der Zugriff mit:

- **max_read_channel_short**

Ist *.usReadMode* = *IO_MODE_MULTIPLE* erfolgt der Zugriff mit:

- **max_read_channel_block**

Sonderdienst

- **max_channel_control,** Steuerbefehle CMD_HIGH_SPEED_START, CMD_HIGH_SPEED_STOP: Zu Beginn einer Messung muss der Steuerbefehl *CMD_HIGH_SPEED_START* gesendet werden. Danach können die Messwerte gelesen werden. Am Ende der Messung muss der Steuerbefehl *CMD_HIGH_SPEED_STOP* gesendet werden. Zwischen dem Aufruf *CMD_HIGH_SPEED_START* und *CMD_HIGH_SPEED_STOP* kann nicht mit einem anderen MDD-Kanal auf dieses Modul zugegriffen werden! Dem Dienst werden keine Daten übergeben.

Beispiel:

```
// Kanal öffnen
Error = max_open_channel(hMdd, sizeof (rcAin), &rcAin, NULL, NULL, &hAin);

// Am Anfang der Messung das Steuerkommando CMD_HIGH_SPEED_START senden
// Das Modul ist jetzt für andere MDD-Kanäle gesperrt
max_channel_control(hAin, CMD_HIGH_SPEED_START, 0, NULL);

// Jetzt können High-Speed Messungen des Kanal durchgeführt werden
// Wenn IO_MODE_MULTIPLE nicht gesetzt ist
max_read_channel_short(hAin, &sData);
...
// Wenn IO_MODE_MULTIPLE gesetzt ist
// der Treiber soll 100 Messwerte a 2 Byte (short) liefern
ulSize = 100 * 2;
max_read_channel_block(hAin, &ulSize, (void*)asData);
...
// Am Ende der Messung das Steuerkommando CMD_HIGH_SPEED_STOP senden
// Das Modul ist jetzt für andere MDD-Kanäle wieder freigegeben
max_channel_control(hAin, CMD_HIGH_SPEED_STOP, 0, NULL);
```

10.5.2.6. Hardwareformate

Bereich	Rohwert	Spannungswert/Stromwert
Alle Spannungsbereiche (RANGE_BIP_...)	*-8192...+8191*	Neg. Vollausschlag...pos. Vollausschlag
RANGE_20MA	*0...+8191*	0...20mA

Die 12-Bit-Varianten liefern ebenfalls 14 Bit, wobei die Genauigkeit auf 12 Bit beschränkt ist.

10.5.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Kanal	SE-Kanal Eingangs-Pin	SE-Kanal GND-Pin	Differenz-Kanal: Pin für +Eingang	Differenz-Kanal: Pin für -Eingang	Differenz-Kanal: Pins für GND
AIN-0	1	2	1	3	2, 4
AIN-10	3	4	-	-	-
AIN-1	5	6	5	7	6, 8
AIN-11	7	8	-	-	-
AIN-2	9	10	9	11	10, 12
AIN-12	11	12	-	-	-
AIN-3	13	14	13	15	14, 16
AIN-13	15	16	-	-	-
AIN-4	17	18	17	19	18, 20
AIN-14	19	20	-	-	-
AIN-5	21	22	21	23	22, 24
AIN-15	23	24	-	-	-
AIN-6	25	26	25	27	26, 28
AIN-16	27	28	-	-	-
AIN-7	29	30	29	31	30, 32
AIN-17	31	32	-	-	-
AIN-8	33 [1]	34	33 [1]	35 [1]	34, 36
AIN-18	35 [1]	36	-	-	-
AIN-9	37 [1]	38	37 [1]	39 [1]	38, 40
AIN-19	39 [1]	40	-	-	-

[1] bei X-AD12-16 sind diese Pins not connected

10.5.4. Besondere Eigenschaften

(Angaben gelten für Standardversion Version X-AD14-20/F)

Parameter	Wert	Einheit
Anzahl externe Analogeingänge je Kanal wählbar per Software als Masse-bezogener Eingang (SE) oder je 2 Eingänge als Differenzeingang, auch gemischter Betrieb möglich	20	
Auflösung, max.	14	Bit
Eingangsbereiche [1] (Standardbestückung) ±10V, ±5V, ±2,5V, (±2,0V), ±1,25V, (±1V), ±500mV, ±250mV Mögliche max. Eingangsbereiche (Bestückungsvarianten) z.B.: 0 .. 20 mA, 4 .. 20 mA, ±250 V, ±100 V, ±50 V	6 a.A.	
Wandlungszeit	450	ns
Überspannungsfestigkeit der Eingänge (Power on oder off)	-37 .. +52	V
Settle-Timer, einstellbar von 30 ns bis 60 µs	on-board	
Sonderkanäle (on-board Temperatur, Offset, ±10V)	8	
Temperaturmessung für automatischen Abgleich, Bereich	-40 .. +125	°C
Genauigkeit bei 25°C / -40°C bis +125°C	±3/±4	°C
Nichtlinearität	±0,8	°C
Ausgangsspannung bei 0°C (typ.)	+424	mV
Kennliniensteilheit (typ.)	+6,25	mV/°C
Langzeitstabilität, bei 125°C für 1000 Stunden (typ.)	±0,2	°C
Temperatur-Bereich (Betrieb)	–40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	9,75	g

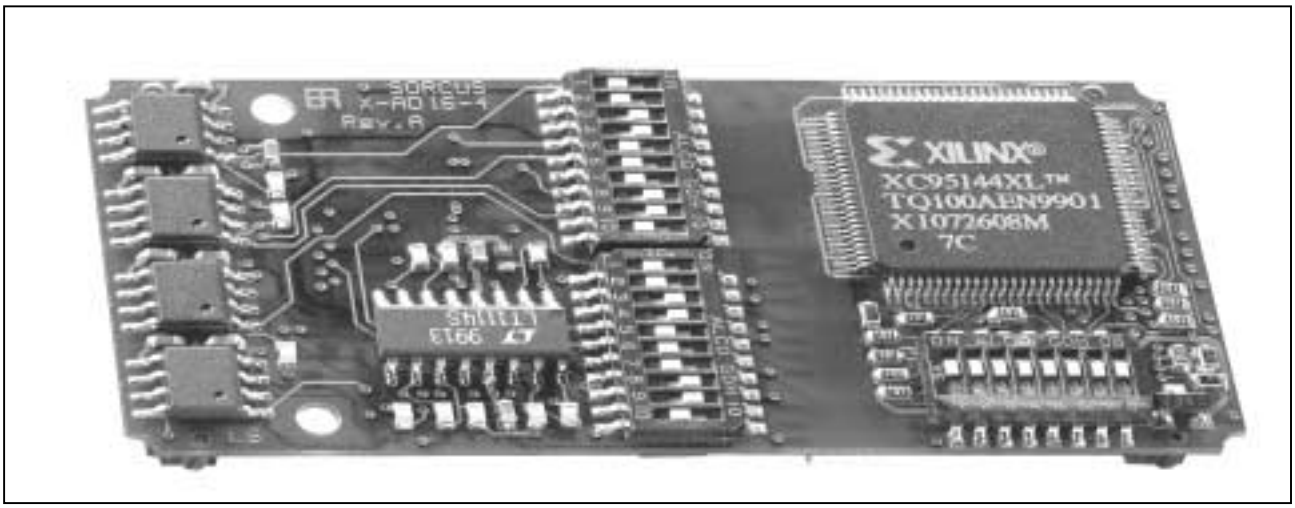
[1] Anmerkung: Die in (Klammern) angegebenen Bereiche stehen ebenfalls zur Verfügung, die anderen Bereiche bieten aber eine höhere Genauigkeit.

Stromaufnahme max. Wandlungsrate / Low-Power Mode		
Modul-Revisionen A, B, C und D		
3,3V	160/150	mA
12V	45/25	mA
–12V	50/30	mA
Modul ab Rev. E:		
3,3 V	65/tbd	mA
12 V	36/tbd	mA
–12 V	34/tbd	mA

Notizen

X-AD16i-4

4 analoge Spannungseingänge mit 16 Bit Auflösung, galvanisch getrennt



10.6. X-AD16i-4

Inhaltsverzeichnis

10.6.1. Beschreibung

Das Modul bietet 4 voneinander unabhängige Spannungseingänge, die von der Trägerkarte, aber nicht untereinander galvanisch getrennt sind. Der Messbereich jedes Kanals wird über DIL-Schalter auf dem Modul eingestellt. Die Kanäle sind voneinander unabhängig konfigurierbar. Folgende Bereiche sind möglich: 2,5V; 5V; ±5V; 10V; ±10V.

10.6.2. Konfiguration des Moduls

Die Eingangsbereiche werden mit den DIL-Schaltern SW1 und SW2 konfiguriert. Die Schalter dürfen nur bei ausgebautem Modul eingestellt werden.

Kanal AIN-0:

Bereich	SW1-1	SW1-2	SW1-3	SW1-4	SW1-5	SW3-1	SW3-2
0..2,5V	ON	ON	OFF	OFF	ON	OFF	OFF
0...5V	OFF	ON	OFF	OFF	ON	ON	OFF
±5V	ON	OFF	ON	ON	OFF	OFF	ON
±10V	OFF	OFF	ON	ON	OFF	ON	ON

Kanal AIN-1:

Bereich	SW1-6	SW1-7	SW1-8	SW1-9	SW1-10	SW3-3	SW3-4
0..2,5V	ON	ON	OFF	OFF	ON	OFF	OFF
0..5V	OFF	ON	OFF	OFF	ON	ON	OFF
±5V	ON	OFF	ON	ON	OFF	OFF	ON
±10V	OFF	OFF	ON	ON	OFF	ON	ON

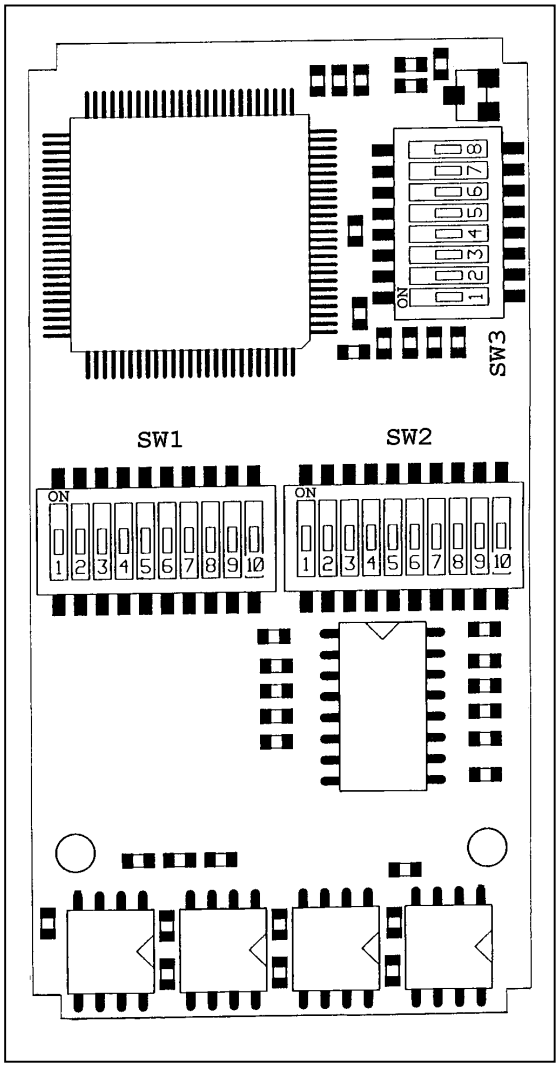
Kanal AIN-2:

Bereich	SW2-1	SW2-2	SW2-3	SW2-4	SW2-5	SW3-5	SW3-6
0..2,5V	ON	ON	OFF	OFF	ON	OFF	OFF
0..5V	OFF	ON	OFF	OFF	ON	ON	OFF
±5V	ON	OFF	ON	ON	OFF	OFF	ON
±10V	OFF	OFF	ON	ON	OFF	ON	ON

Kanal AIN-3:

Bereich	SW2-6	SW2-7	SW2-8	SW2-9	SW2-10	SW3-7	SW3-8
0..2,5V	ON	ON	OFF	OFF	ON	OFF	OFF
0..5V	OFF	ON	OFF	OFF	ON	ON	OFF
±5V	ON	OFF	ON	ON	OFF	OFF	ON
±10V	OFF	OFF	ON	ON	OFF	ON	ON

10.6.3. Lageplan des Moduls



10.6.4. Modul-Device-Treiber

10.6.4.1. Installation

Der Modul-Device-Treiber für das OsX hat die Programmnummer 8020h und den Dateinamen mxad164.exe. Der Modul-Device-Treiber für Windows hat den Namen mxad1648.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8020, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8020

10.6.4.2. Kanaleigenschaftsstruktur CPS_XAD164

Die CPS für das Modul hat den Namen CPS_XAD164.

10.6.4.3. Analoge Eingänge

Ein Kanal zu einem der 4 analogen Differenz-Spannungseingänge wird mit folgender CPS geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AIN_DIFF*	Kanal zu analogem Differenz-Eingang
.usIndexFirst	*0 ... 3*	Nummer des ersten Eingangs
.usIndexLast	*0 ... 3*	Nummer des letzten Eingangs
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usFlags	*0*	Die Daten werden im normierten, korrigierten Datenformat zurückgeliefert
	_CP_UNCORRECTED	Die Werte werden keiner internen Korrektur unterzogen. In der momentanen Version des Treibers muss dieses Flag gesetzt werden, sofern das Flag *_CP_HW_FORMAT* nicht gesetzt ist.
	_CP_HW_FORMAT	Es werden direkt die vom Wandler gelieferten Rohwerte zurückgegeben (das Flag *_CP_UNCORRECTED* hat keine Auswirkungen). Die Zuordnung zwischen Rohwert und Spannungswert können Sie der Tabelle entnehmen.
.usRange	*RANGE_BIP_10V*	Bipolar ±10 V
	RANGE_BIP_5V	Bipolar ±5 V

Strukturelement	Werte	Bedeutung
	RANGE_UPP_5V *RANGE_UPP_2V5*	Unipolar 0 ... 5 V Unipolar 0 ... 2,5 V Der übergebene Eingangsbereich muss mit den DIL-Schaltern übereinstimmen.
.usOversampling	*1 ... 65535*	Der Wert gibt an, wie oft ein Eingang gemessen werden soll. Wird ein Wert > 1 eingetragen, führt der MDD eine Mittelung durch.

Rohwerte des Wandlers:

Bereich	Auflösung	0	8000h	ffffh
0 ... 2,5 V	38 µV	0	1,25 V	2,5 V - 38 µV
0 ... 5 V	76 µV	0	2,5 V	5 V - 76 µV
-5 ... 5 V	152,5 µV	- 5 V	0 V	5 V - 152,5 µV
-10 ... 10 V	305 µV	- 10 V	0 V	10 V - 305 µV

Eingabedienst

Wenn das Flag *_CP_HW_FORMAT* gesetzt ist, ist der Datentyp des Kanals DATA_USHORT:

- **max_read_channel_ushort** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Wenn das Flag *_CP_HW_FORMAT* nicht gesetzt ist, ist der Datentyp des Kanals DATA_LONG:

- **max_read_channel_long** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

10.6.4.4. Abfrage der Eingangsbereiche

Dieses Device bietet die Möglichkeit, die auf dem Modul (per DIL-Schalter) eingestellten Eingangsbereiche auszulesen. Dazu wird ein Kanal mit folgender CPS geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Kanal zur Abfrage der Eingangsbereiche
.usIndexFirst	*XAD164_RANGE_SETTING*	Eingangsbereich
.usIndexLast	*XAD164_RANGE_SETTING*	Eingangsbereich
.usFlags	*0*	Reservierter Parameter
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff

Anmerkungen

Mit einem Blocklesezugriff auf diesen Kanal werden die Eingangsbereiche aller 4 Kanäle ermittelt. Für jeden Kanal wird der Eingangsbereich in einem USHORT Wert (2 Byte) abgelegt. Insgesamt müssen bei einem Lesezugriff also 8 Byte an Daten gelesen werden. Die Werte entsprechen den Konstanten *RANGE_BIP_10V*, *RANGE_BIP_5V*, *RANGE_UPP_5V* und *RANGE_UPP_2V5*, die auch beim Öffnen eines Eingangskanals angegeben werden müssen.

Eingabedienst

Der Datentyp ist DATA_USHORT. Der Kanalzugriff erfolgt mit:

- **max_read_channel_block**

Beispiel

Das folgende Beispiel liest die Eingangsbereiche der 4 Eingänge:

```
USHORT ausRange[4];     // Bereiche für Kanal-0 ... 3
ULONG ulSize;
ulSize = 8;             // 2 Byte pro Kanal * 4 Kanäle

max_read_channel_block(hChan, &ulSize, (void*) ausRange);
```

10.6.5. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Kanal	Pin-Bezeichnung	Pin-Nr.
AIN-0	+AIN-0	4
	–AIN-0	6
	GND-0	1; 2; 3; 5; 7; 9; 10
AIN-1	+AIN-1	14
	–AIN-1	16
	GND-1	11; 12; 13; 15; 17; 19; 20
AIN-2	+AIN-2	24
	–AIN-2	26
	GND-2	21; 22; 23; 25; 27; 29; 30
AIN-3	+AIN-3	34
	–AIN-3	36
	GND-3	31; 32; 33; 35; 37; 39; 40

10.6.6. Besondere Eigenschaften

Parameter	**Wert**	**Einheit**
Anzahl der Eingänge	4	-
Eingangsspannungsbereiche		
bipolar	±5, ±10	V
unipolar	0 .. +2,5, 0 .. +5	V
Auflösung	16	Bit
Linearität	2	LSB
Full-Scale-Error (inkl. Offset), max.	0,5	%
Offsetfehler	±10	mV
Überspannungsfestigkeit	±40	V
Eingangswiderstand	10^9	Ω
Eingangskapazität	4	pF
Wandlungsdauer	5	µs
Temperatur-Bereich, Betrieb	0 .. 50	°C
Abmessungen	29x58x8	mm
Gewicht	11,6	g
Stromaufnahme +3,3V	160	mA
+12V	105	mA

Notizen

X-AD24-4i

4 differentielle analoge Eingänge für
direkten Sensoranschluss

10.7. X-AD24-4i

Inhaltsverzeichnis

10.7.1. Beschreibung

Zur Messung von Temperatur, Schwingung, Druck oder elektrischem Widerstand können an das MAX-Modul X-AD24-4i vier Sensorelemente wie Dehnmessbrücken (4- und 6-Leiter), Temperaturmesswiderstände (z.B. Pt100) oder Thermoelemente (alle Typen) direkt angeschlossen werden. Jeder Sensorkanal liefert 24 Bit Auflösung und ist galvanisch vom Rest des Moduls getrennt. Die Wandlung erfolgt mit einer integrierten Störunterdrückung der 50Hz bzw. 60Hz Netzfrequenz. Bei den Versionen /S, /P und /T entspricht dies Ergebnisraten von 6,25SPS bzw. 7,5SPS. Alternativ kann die Wandlung auch mit 97,5SPS (Version /S, /P, /T) bzw. 42kSPS (Version /V, /C, /F, /I) erfolgen.

Die für Dehnmessbrücken benötigte Versorgungsspannung wird für jeden Sensor direkt von dem Modul geliefert und ist galvanisch getrennt. Für die Messung von sehr hohen oder sehr niedrigen Temperaturen werden verschiedenen Typen von Thermoelementen eingesetzt. Diese können unabhängig vom Typ direkt an das Modul angeschlossen werden. Die Linearisierung und die Kaltstellenkompensation wird von der Treibersoftware vorgenommen. Für die präzise Temperaturmessung mit Hilfe von Pt100 bzw. Pt1000 Temperaturmesswiderständen wird eine 4-Leiter Verkabelung genutzt, um die parasitären Widerstände in den Zuleitungen zu kompensieren.

Eingangsbereiche

Die Module bieten 4 AD-Wandler mit je einem differentiellen Messeingang und einem differentiellen Referenzeingang**.** Jeder AD-Wandler ist gegen die übrigen Wandler und gegen den X-Bus galvanisch getrennt. Die Eingangsbereiche werden durch die Versorgungsspannung der Wandler auf 0-5V begrenzt, jedoch wird durch einen Operationsverstärker auch ein Eingangsbereich von –10V bis +10V ermöglicht (Version /V). Die Module eignen sich zum Anschluss von Sensorelementen wie Dehnmessbrücken (4- und 6-Leiter), Temperaturmesswiderständen (z.B. Pt100) oder Thermoelementen (alle Typen) und bieten eine galvanisch getrennte 5V-Versorgung für die Sensoren. Für ICP Sensoren beträgt die externe Spannungsversorgung 24V@4mA. Für Sensoren mit Stromschnittstelle steht eine Modulversion mit 0..20mA Eingangsstrombereich zur Verfügung.

Andere Bereiche sind geänderter Eingangsbeschaltung des Operationsverstärkers je Kanal möglich.

Folgende Bestückungsversionen sind standard-mässig lieferbar:

Typ	Sub typ	Version	Sensoranschluss	Max. Wandel-rate	Sensor-versorgung	Bemerkung
20	0	X-AD24-4i/S	Messbrücken	97,5 SPS	4x 5V, 15mA	Für 350Ω Brücken
20	1	X-AD24-4i/T	Thermoelemente	97,5 SPS	4x 2mA	Externe Kompensation
20	2	X-AD24-4i/P	Pt100 Temp. Sensoren	97,5SPS	4x 2mA	
20	3	X-AD24-4i/I	ICP-Sensoren	42k SPS	4x 24V,4mA	Keine Galvanische Trennung
20	4	X-AD24-4i/V	-10V..+10V	42k SPS	-	
20	5	X-AD24-4i/C	0-20mA	42k SPS	-	100Ω Eingangswiderstand
20	6	X-AD24-4i/F	Messbrücken	42k SPS	4x 5V, 5mA	
20	-	X-AD24-4i/x	Kundenspez. Kombinationen	Wie oben.	Wie oben.	

Verwendete Wandler, Anti-Aliasing und Oversampling

Auf den Modulen werden AD-Wandler eingesetzt, die nach dem Sigma-Delta Verfahren arbeiten. Diese Wandler tasten das Analogsignal mit einer hohen internen Frequenz ab und führen das Wandelergebnis durch ein digitales SINC-Filter. Die Analogbandbreite des Einganges ist wesentlich niedriger als die Ergebnisrate, so dass in den meisten Fällen Anti-Aliasing Filter überflüssig sind. In Anwendungen, in denen die Bandbreite des Eingangssignals grösser als die 256-fache Ergebnisrate sein kann, sind zusätzliche Tiefpassfilter notwendig. Diese können aber aufgrund des hohen Abstandes zum Nutzsignal mit einfachen RC-Filtern ausgeführt werden.

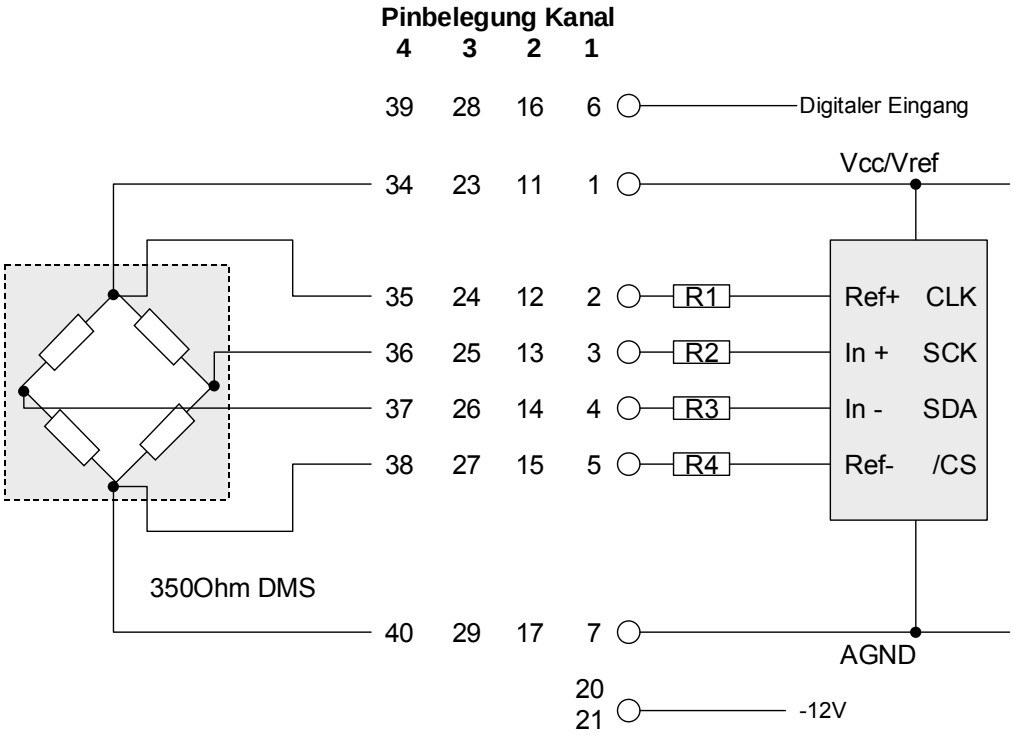
10.7.2. Anschluss von Dehnmessbrücken an das X-AD24-4i/S

Für Dehnmessbrücken liefert das Modul je Kanal eine galvanisch getrennte Sensorversorgung von 5V, die mit 15mA belastbar ist. Die Dehnmessbrücken sollten immer in 6-Leiter-Anschlusstechnik verdrahtet werden, um Messfehler zu minimieren. Wenn Sie einen 4-Leiter DMS an das Modul anschliessen, sollte der Anschluss der Sensorversorgung mit getrennten Kabeln ausgeführt werden und so dicht am DMS wie möglich mit den Referenzanschlüssen verbunden werden.

Beim Anschluss von Halbbrücken muss eine Ergänzung zur Vollbrücke mit zwei Widerständen erfolgen. Der Wert dieser Widerstände muss mit dem Widerstandswert der verwendtetn Halbbrücke übereinstimmen, um Offsetfehler zu minimieren.

Der Wandler bildet ein ratiometrisches Messergebnis mit einem Wertebereich von –1 bis +1, das nur noch durch die Empfindlichkeit der Messbrücke dividiert werden muss. Die Belastung der Messbrücke kann wie folgt umgerechnet werden:

Messgrösse = Wandelergebnis/(Vollaussteuerung/Einheit)

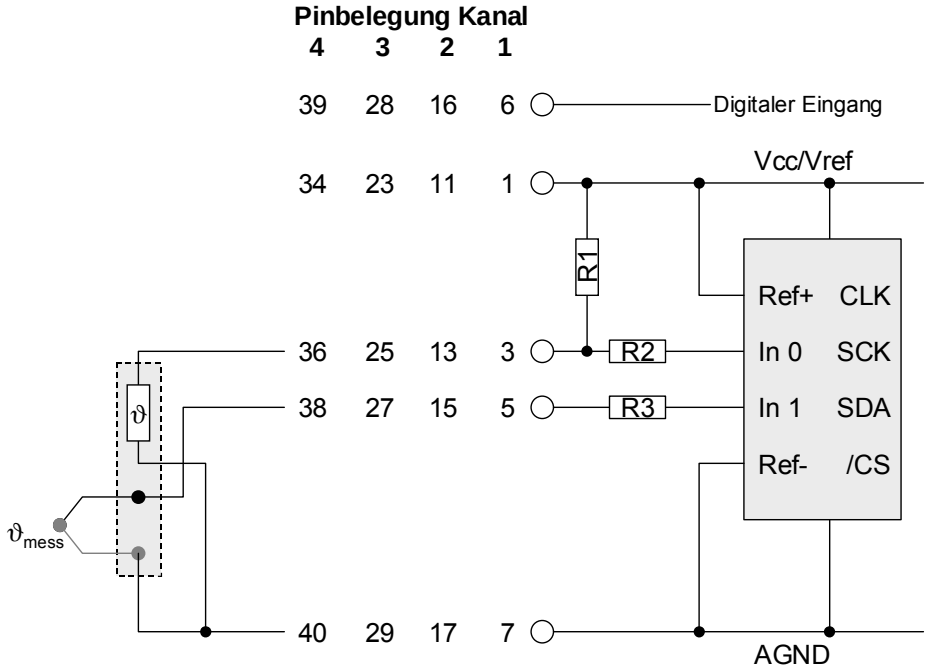


10.7.3. Anschluss von Thermoelementen an das X-AD24-4i/T

Thermoelemente können unabhängig vom Typ direkt an das Modul angeschlossen werden. Die Linearisierung und die Kaltstellenkompensation wird von der Treibersoftware vorgenommen.

Für die Temperaturmessung mit Thermoelementen ist die Messung einer Vergleichstemperatur notwendig. Die Platzierung dieses Temperatursensors ist in einer gesonderten Application-Note beschrieben. Über den mitgelieferten Modul-Device-Treiber (MDD) wird eine Korrektur mit der Kaltstellentemperatur sowie eine Linearisierung je nach Thermoelementtyp vorgenommen.

Für jeden Kanal ist eine gesonderte Messung der Kaltstellenkompensation vorgesehen. Eine Kaltstellenkompensation mit nur einem Pt100 für alle Kanäle führt zu vergrösseren Messungenauigkeiten. Der Messfühler für die Kaltstellentemperatur ist intern mit einem 2,5kΩ Vorwiderstand gegen die 5V Referenzspannung an das Modul angeschlossen. Für die Kaltstellenkompensation eigenen sich Pt100 Temperatursensoren.

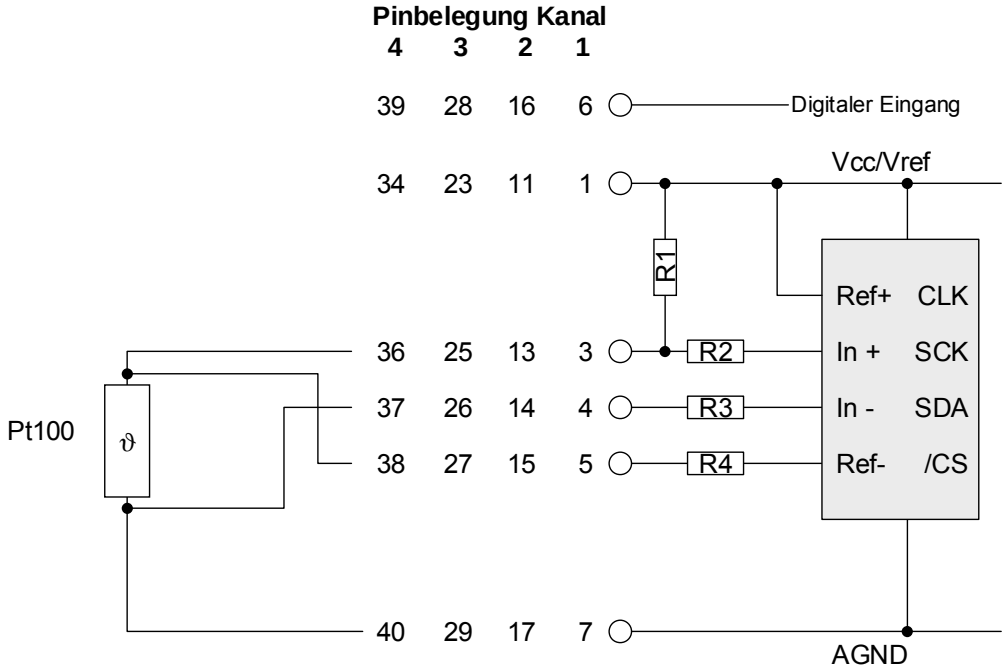


Die Messwerte des Wandlers geben hier für den Kanal 0 das ratiometrische Verhältnis des 2,5kΩ Widerstandes zur Kaltstellenkompensation an (siehe Beschreibung des X-AD24-4i/P). Für Kanal 1 wird die Spannung des Thermoelementes ausgegeben. 1LSB entspricht dann $5V/2^{24}$, also 0,29802µV. Die Umrechnung (Linearisierung und Kaltstellenkompensation) in die Temperatur der Messstelle wird vom MDD vorgenommen.

10.7.4. Anschluss von Temperaturmesswiderständen an das X-AD24-4i/P

Wenn präzise Temperaturen mit Hilfe von Pt100 bzw. Pt1000 Temperaturmesswiderständen gemessen werden sollen, wird der differentielle

Referenzeingang dazu benutzt, die parasitären Widerstände in den Zuleitungen zu kompensieren. Dazu ist eine 4-Leiter Verkabelung notwendig.



Auf dem Modul wird eine Schaltung mit einem Pt100 und einem Vorwidertand[3] R1 von 2,5kΩ/0,1% verwendet. Ein Pt100 hat im Bereich von –200°C bis +850°C die in der Tabelle gegebenen Widerstandswerte (nach DIN43760).

Temperatur	-200°C	-100°C	0°C	100°C	200°C	400°C	800°C	850°C
Widerstand	21,16 Ω	60,25Ω	100Ω	138,5Ω	175,8Ω	247Ω	375,5Ω	390,3Ω
Wandelergebnis	0x045564	0x0C56D5	0x147AE1	0x1C5D63	0x2400FB	0x3295E9	0x4CE703	0x4FEEF5

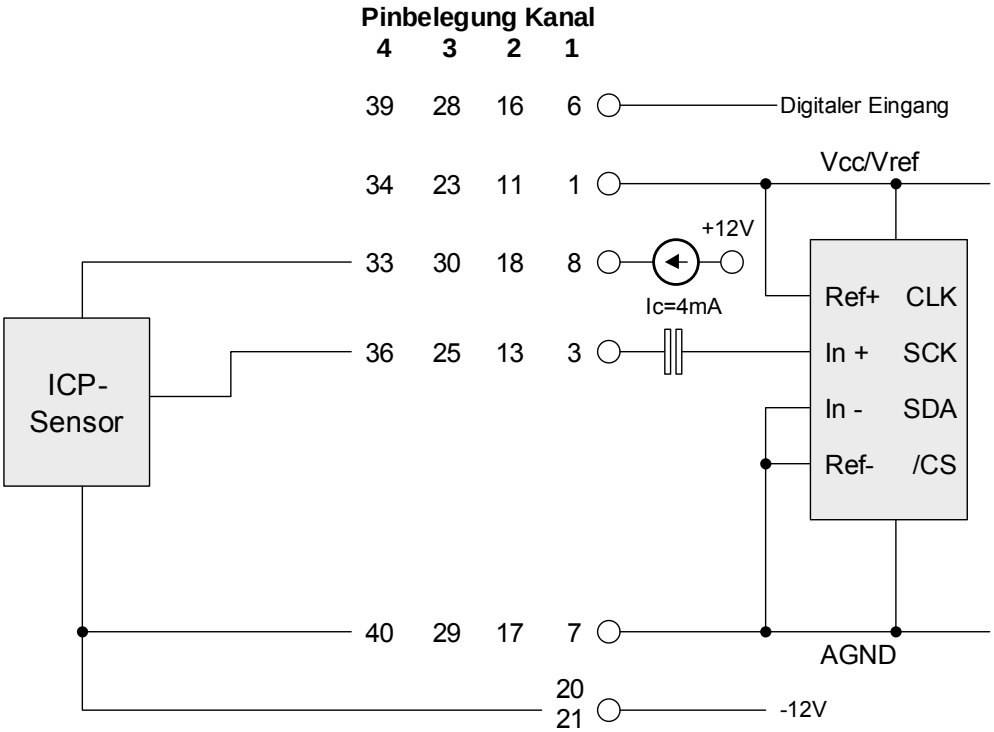
Der Wandler bildet ein ratiometrisches Messergebnis als den Quotienten aus dem halben Vorwiderstand zum Widerstandswert des Pt100. Dieser Quotient wird direkt als Wandelergebnis ausgegeben.

10.7.5. Anschluss von ICP®-Sensoren an das X-AD24-4i/I

Das Modul liefert eine Sensorstromversorgung von 4mA bei 24V. Diese Sensorstromversorgung ist jedoch nicht galvanisch vom X-Bus getrennt, so dass hier der Verkabelung besondere Beachtung zukommen muss. Die galvanisch getrennten

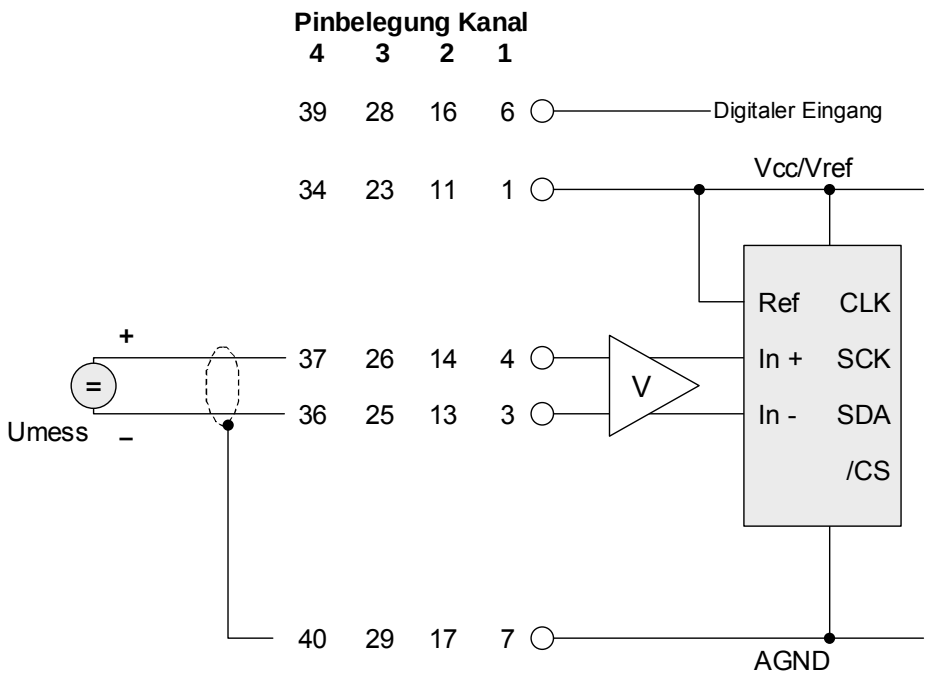
[3] Der Strom beträgt dann max. 2mA was bei maximalem Widerstand zu einer zusätzlichen Verlustleistung im Messwiderstand von 1,2mW führt.

Masseanschlüsse der einzelnen Kanäle müssen hierbei mit der –12V Leitung verbunden werden.



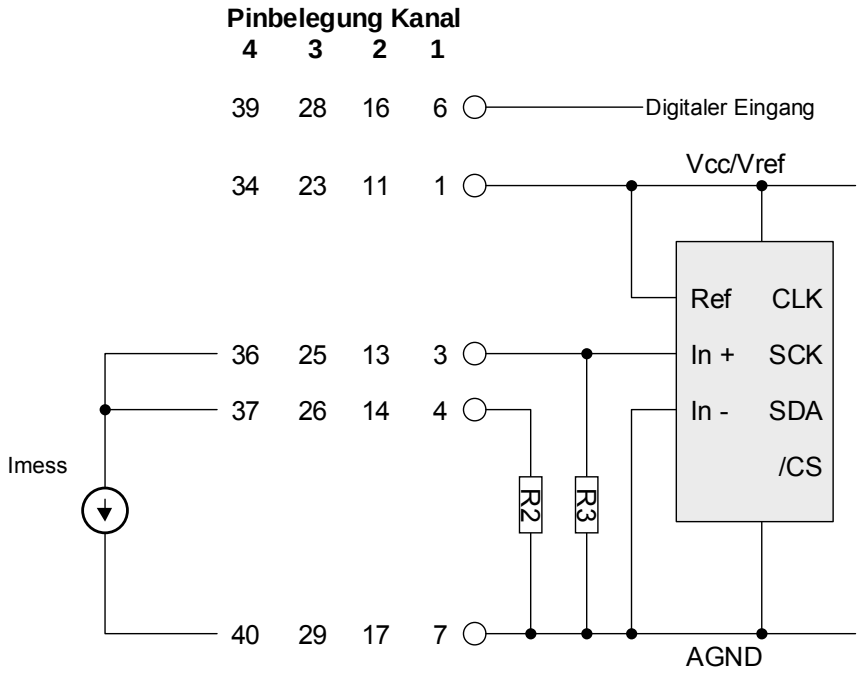
10.7.6. Spannungsmessung mit dem X-AD24-4i/V

Zur Messung von Spannungen mit dem X-AD24-4i/V werden die Spannungsquellen differentiell an das Modul angeschlossen. Dabei darf keiner der Eingänge mit der galvanisch getrennten Masse des Moduls verbunden werden, da die Eingänge intern auf einem Referenzpotential liegen. An die galvanisch getrennte Masse sollte die Abschirmung des Anschlusskabels angeschlossen werden. Der maximale Eingangsspannungsbereich beträgt –12,5 V ... +12,5 V.



10.7.7. Strommessung mit dem X-AD24-4i/C

Auf dem X-AD24-4i/C sind zur Strommessung zwei getrennte Shunt-Widerstände mit je 200Ω 0,1% bestückt, die jeweils mit 12mA belastet werden dürfen. Daher müssen für einen Messbereich von 0..20mA jeweils zwei Anschlüsse gemeinsam angeschlossen werden.



10.7.8. Galvanische Trennung

Das Modul hat vier galvanisch getrennte Kanäle, die sowohl vom X-Bus als auch untereinander getrennt sind. Die galvanische Trennung zum X-Bus hat eine Isolationsspannung von 500V, zwischen den Kanälen beträgt die Isolationsspannung jeweils 100V zwischen benachbarten Gruppen. Wenn höhere Isolationsspannungen benötigt werden, müssen die Signale auf verschiedene X-AD24-4i Module gelegt werden.

10.7.9. Modul-Device-Treiber

10.7.9.1. Installation

Der Modul-Device-Treiber für das OsX hat den Dateinamen mxad244i.exe. Der Modul-Device-Treiber für Windows hat den Namen mxad244i.sys.

Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0), die Programm-Nr. ist abhängig vom Modul-Typ:

Error = max_load_mdd (hModul, 1, 0, 0, 0x8014, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0) , die Programm-Nr. ist abhängig vom Modul-Typ:

MAXLOADMDD slot=1 layer=0 progno=8014

10.7.9.2. Kanaleigenschaftsstruktur CPS_XAD244I

Die CPS für das Modul hat den Namen CPS_XAD244I.

10.7.9.3. Analoge Eingänge

Das Modul stellt 4 differenzielle oder single ended analoge Eingänge zur Verfügung. Um auf diese zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*0*	Version dieser CPS-Definition
.usDevice	*DEVICE_AIN_DIFF*	Kanal auf einen analogen Differenzeingang
	DEVICE_AIN_DIFF	Kanal auf einen analogen massebezogenen Eingang
.usIndexFirst	0 ... 3	Nummer des ersten Eingangs
.usIndexLast	0 ... 3	Nummer des letzten Eingangs
.usReadMode	*IO_MODE_LATCH*	Die Daten werden aus einem Zwischenspeicher (Latch) gelesen (siehe auch *ulMode*)
.usWriteMode	*0*	reservierter Parameter
.ulMode	*XAD244I_MODE_TRIGGER*	Die Daten werden per Software-Trigger (siehe auch Kanal Software-Trigger) und/oder DIN gelatcht (siehe auch *ulTrigger*)
	XAD244I_MODE_CONTINUOUS	Die Daten werden kontinuierlich gelatcht
.ulFilter	*XAD244I_MODE_CONT_50_HZ_FILTER*	es wird eine 50Hz Störunterdrückung durchgeführt wird
	XAD244I_MODE_CONT_60_HZ_FILTER	es wird eine 60Hz Störunterdrückung durchgeführt wird
	XAD244I_MODE_CONT_NO_FILTER	es wird keine Störunterdrückung durchgeführt wird
.ulFlags	*0*	Keine Bedeutung
	_CP_HW_FORMAT	Es werden direkt die Werte vom Wandler geliefert.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.

Strukturelement	Werte	Bedeutung
.ulCallbackEvents		Dieser Parameter regelt, wann die Anwender-Callback-Funktion aufgerufen werden soll:
	0	es wird keine Callback-Funktion aufgerufen
	XAD244I_EVENT_NEW_DATA	wenn neue Wandlungsdaten bereit stehen
.usOversampling	*1 ... 65535*	Oversampling: der MDD führt eine Mittelung über die letzten 1 bis 65535 Messwerte durch
.ulSensor	*0*	Typ des angeschlossenen Sensors
.ulTrigger	*0* *XAD2444I_DIN_POS_EDGE* *XAD244I_DIN_NEG_EDGE*	Wenn der Mode XAD244I_MODE_TRIGGER gewählt wurde, kann hier angegeben werden, ob eine positive und/oder negative Flanke am digitalen Trigger-Eingang den analogen Eingang latchen soll. Zusätzlich kann noch über einen DEVICE_TRIGGER-Kanal per Software gelatcht werden. Dem AIN-0 ist der DIN-0, dem AIN-1 der DIN-1, ... als Triggereingang zugeordnet

Eingabedienste DEVICE_AIN_DIFF

Der Datentyp des Kanals ist ULONG:

- **max_read_channel_block**

Eingabedienste DEVICE_AIN_SE

Der Datentyp des Kanals ist ULONG:

- **max_read_channel_block**

Callback-Funktion

Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen wenn das in *ulCallbackEvents* angegebenen Ereigniss aufgetreten ist. Die Callback-Funktion bekommt einen ULONG-Wert übergeben, der die angegebene Konstante enthält.

10.7.9.4. Software-Trigger zum Latchen der analogen Eingänge

Um einen Software-Trigger zum latchen der analogen Eingänge zu nutzen, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*0*	Version dieser CPS-Definition
.usDevice	*DEVICE_TRIGGER*	Trigger Kanal
.usIndexFirst	*XAD244I_AIN_TRIGGER*	Trigger für die analogen Eingänge
.usWriteMode	*IO_MODE_DIRECT*	
.usReadMode	*0*	reservierter Parameter
.ulFlags	*0*	reservierter Parameter
.ulMode	*0*	reservierter Parameter

Ausgabedienste

Der Datentyp des Kanals ist ULONG:

- **max_write_channel_ulong**

Als Wert für den Schreibdienst ist momentan der Wert *XAD244I_LATCH_ALL_AIN* möglich. Mit einem Schreibbefehl werden alle analogen Eingänge gelatcht.

10.7.10. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Für die Modulversionen /S, /P, /I und /F gilt folgende Anschlussbelegung. Einige der Pins sind je nach Modulversion unbeschaltet. Für den korrekten Anschluss der Sensoren siehe vorige Kapitel.

Pin	Bedeutung	Kanal 1	Kanal 2	Kanal 3	Kanal 4
IN+	Differenzeingang, positive Seite	3	13	25	36
IN-	Differenzeingang, negative Seite	4	14	26	37
V5x	Galvanisch getrennte 5V-Versorgung für Sensoren	1	11	23	34
Ref+	Differentieller Referenzeingang +	2	12	24	35
Ref-	Differentieller Referenzeingang -	5	15	27	38
D_In	Digitaler Eingang	6	16	28	39

Pin	Bedeutung	Kanal 1	Kanal 2	Kanal 3	Kanal 4
GNDx	Galvanisch getrennte Masse für Sensorversorgung und digitalen Eingang	7	17	29	40
NC	Nicht angeschlossen	9, 10	19	22	31, 32
ICC	Konstantstrom 4mA für ICP-Modul	8	18	30	33
VM12	-12V für ICP-Sensoren		20	21	

Bei den Modulversionen /T, /V und /C gelten folgende abweichende Steckerbelegungen

Modul X-AD24-4i/T

Pin	Bedeutung	Kanal 1	Kanal 2	Kanal 3	Kanal 4
IN1	Eingang Kaltstellenkompensation (CH0*)	3	13	25	36
IN2	Eingang Thermoelement (CH1*)	5	15	27	38
V5x	Galvanisch getrennte 5V-Versorgung für Sensoren	1	11	23	34
D_In	Digitaler Eingang	6	16	28	39
GNDx	Galvanisch getrennte Masse für Sensoren und digitalen Eingang	7	17	29	40
NC	Nicht angeschlossen	2, 4, 8, 9, 10	12, 14, 18,19	22, 24, 26, 30	31..33, 35, 37
VM12	-12V für ICP-Sensoren		20	21	

* Siehe Beschreibung des Hardware-Formats

Modul X-AD24-4i/V

Pin	Bedeutung	Kanal 1	Kanal 2	Kanal 3	Kanal 4
IN+	Differenzeingang, positive Seite	4	14	26	37
IN-	Differenzeingang, negative Seite	3	13	25	36
D_In	Digitaler Eingang	6	16	28	39
GNDx	Galvanisch getrennte Masse für Sensorversorgung und digitalen Eingang	7	17	29	40
NC	Nicht angeschlossen	2, 5, 8, 9, 10	12, 15, 18,19	22, 24, 27, 30	31..33, 35, 38
VM12	-12V für ICP-Sensoren		20	21	

Modul X-AD24-4i/C

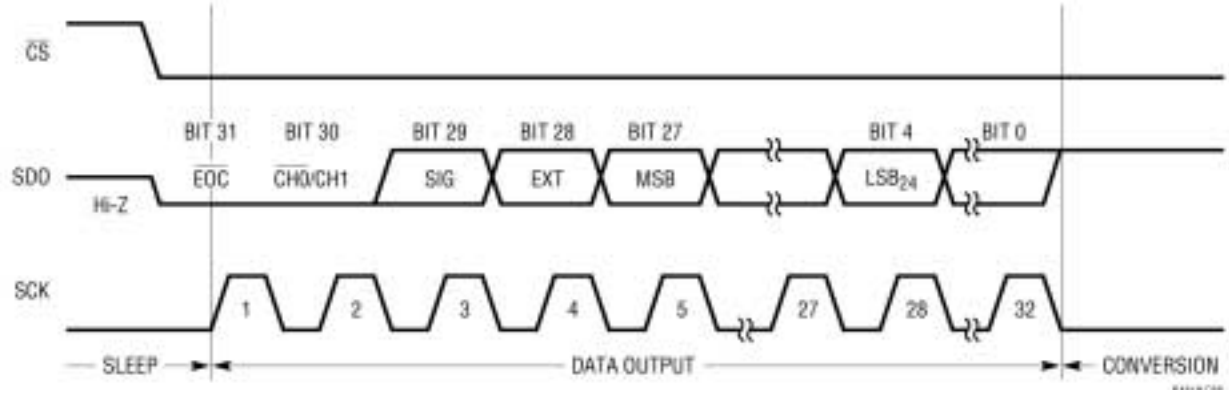
Pin	Bedeutung	Kanal 1	Kanal 2	Kanal 3	Kanal 4
IN1	Stromeingang 1, mit 2 verbinden!	3	13	25	36
IN2	Stromeingang 2, mit 1 verbinden!	4	14	26	37
D_In	Digitaler Eingang	6	16	28	39
GNDx	Galvanisch getrennte Masse für Stromeingang und digitalen Eingang	7	17	29	40
NC	Nicht angeschlossen	2, 5, 8, 9, 10	12, 15, 18,19	22, 24, 27, 30	31..33, 35, 38
VM12	-12V für ICP-Sensoren		20	21	

10.7.11. Hardware Datenformat

Je nach Modultyp gibt es zwei unterschiedliche Hardwareformate.

Hardwareformat 1: Modulversion /S, /T, /P, /I

Die Module /S und /P setzen den AD-Wandler LTC2411 ein. Der Modultyp /T verwendet den Zweikanal-Wandler LTC2402.



	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	…	Bit 4	Bit 3-0
Input Voltage	EOC	CH SELECT	SIG	EXR	MSB						LSB	SUB LSBs(1)
$V_{IN} > 9/8 \cdot V_{REF}$	0	CH0/CH1	1	1	0	0	0	1	1	...	1	X
$9/8 \cdot V_{REF}$	0	CH0/CH1	1	1	0	0	0	1	1	...	1	X
V_{REF} + 1LSB	0	CH0/CH1	1	1	0	0	0	0	0	...	0	X
V_{REF}	0	CH0/CH1	1	0	1	1	1	1	1	...	1	X
$3/4V_{REF}$ + 1LSB	0	CH0/CH1	1	0	1	1	0	0	0	...	0	X
$3/4V_{REF}$	0	CH0/CH1	1	0	1	0	1	1	1	...	1	X
$1/2V_{REF}$ + 1LSB	0	CH0/CH1	1	0	1	0	0	0	0	...	0	X
$1/2V_{REF}$	0	CH0/CH1	1	0	0	1	1	1	1	...	1	X
$1/4V_{REF}$ + 1LSB	0	CH0/CH1	1	0	0	1	0	0	0	...	0	X
$1/4V_{REF}$	0	CH0/CH1	1	0	0	0	1	1	1	...	1	X
0+/0–	0	CH0/CH1	1/0(2)	0	0	0	0	0	0	...	0	X
–1LSB	0	CH0/CH1	0	1	1	1	1	1	1	...	1	X
$-1/8 \cdot V_{REF}$	0	CH0/CH1	0	1	1	1	1	0	0	...	0	X
$V_{IN} < -1/8 \cdot V_{REF}$	0	CH0/CH1	0	1	1	1	1	0	0	...	0	X

(1): Die SUB-LSB-Werte sind gültig und können bei Mittelwertbildung berücksichtigt werden.

(2): Im 0-Durchgang ändert sich das Vorzeichenbit

Das Bit 31 zeigt das Ende der Wandlung an und muss immer 0 sein.

Das Bit 30 zeigt den gewandelten Kanal an. Bei den Modulen mit einem Differenzkanal (/S und /P) ist das Bit 30 immer 0, bei dem Modul /T ist beim Kanal der Kaltstellenkompensation das Bit 30 auf 0 und beim Thermoelement ist das Bit 30 auf 1.

Das Bit 28 zeigt eine (zulässige) Überschreitung des Eingangsbereiches an. Wenn die Eingangsspannung über 9/8 V_{REF} oder unter –1/8 V_{REF} liegt, bleibt das Wandelergebnis auf einem Maximalwert stehen.

Hardwareformat 2: Modulversion /V, /C, /I, /F

Die Module /V, /C, /I und /F setzen den AD-Wandler ADS1252 ein. Dieser Wandler liefert ein 24Bit Ergebnis im 2er Komplement. Die unteren 8 Bits werden mit 0 aufgefüllt.

Eingangssignal	Wandelergebnis
+Vollaussteuerung	`0x7FFFFF00`
+ 1 LSB	`0x00000100`
Null	`0x00000000`
- 1 LSB	`0xFFFFFF00`
–Vollaussteuerung	`0x80000000`

10.7.12. Technische Daten

Parameter	Randbedingungen	Wert min.	Wert typ.	Wert max.	Einheit	Anm.
Analoge Eingänge						
Anzahl	Alle Modultypen ausser /T		4 differenz			
	Modultyp /T		8 single-ended			
Auflösung				24	Bit	
Messgenauigkeit	7,5SPS	17	19	24	Bit	4
	97,5SPS	16	18	24		
	42,5kSPS	14	15	21		
Analogbandbreite	Modultyp /S, /P und /T					
	bei 50Hz Unterdrückung	0		3,02	Hz	
	bei 60Hz Unterdrückung	0		3,63	Hz	
	bei 97,5 SPS	0		47,1	Hz	
Analogbandbreite (-3dB)	Modultyp /C und /V					
	bei 50Hz Unterdrückung	0		10,8	Hz	
	bei 60Hz Unterdrückung	0		12,9	Hz	
	bei 42,5 kSPS	0		9100	Hz	
	Modultyp /I					
	bei 42,5 kSPS	1,5		9100	Hz	
Integrale Nichtlinearität	Tmin...Tmax			+/- 4	LSB	2
Differentielle Nichtlinearität	Tmin...Tmax			+/- 1,5	LSB	2
Verstärkungsfehler	Tmin...Tmax, per Software korrigierbar			+/- 2	mV	2
Verstärkungs-Temperatur-Koeffizient	Tmin...Tmax, per Software korrigierbar					
	Modultyp /S, /P und /T		2		ppm/°C	
	Modultyp /V, /C und /I		7,5		ppm/°C	
Offset-Fehler	Tmin...Tmax, per Software korrigierbar			+/- 4	mV	2
Ergebnisrate pro Kanal	Modultyp /S, /P und /T					
	bei 50Hz Unterdrückung			7,5	Hz	
	bei 60Hz Unterdrückung			6,25	Hz	
	Maximal			97,5	Hz	
	Modultyp /V, /C und /I					
	bei 50Hz Unterdrückung			50	Hz	
	bei 60Hz Unterdrückung			60	Hz	
	Maximal			42,5	kHz	

Parameter	**Randbedingungen**	**Wert** min.	typ.	max.	**Einheit**	**Anm.**
Versorgungsspannungen des Moduls						
Versorgungsstrom	für +3,3 V		165		mA	
	für +/-12 V (nicht /I)		0		mA	
	für +/-12V (/I-Version)		16		mA	
Temperaturbereich Betrieb		–40		+85	°C	
Lagerung		–40		+85	°C	

Anm. 1: steht für einen der Kanäle A, B, C oder D, **Anm. 2:** Tmin = 0 °C, Tmax. = 70 °C, **Anm. 3:** ohne Software-Korrektur, **Anm. 4:** Maximalwert mit Oversampling

X-C16-3i

Zähler-Modul mit 3 universellen Zählerkanälen,
12 Opto-entkoppelten Ein- und 8 Ausgängen

10.8. X-C16-3i

Inhaltsverzeichnis

10.8.1. Beschreibung

Das Modul stellt 3 16-Bit-Zähler zur Verfügung. Diese sind kaskadierbar. Weiterhin hat das Modul 12 externe, einzeln opto-entkoppelte High-Speed Eingänge. Die Eingänge können per Software für unterschiedliche Funktionen konfiguriert werden. Alle 12 Eingänge sind Interrupt-fähig und können als allgemeine Eingänge per Software abgefragt werden. Weiterhin können sie auf verschiedene Art und Weise als Zähler- und Steuereingänge für die 3 Zählerkanäle konfiguriert werden.

Das Modul ist in 3 Bestückungsvarianten lieferbar, die sich durch die Eingangspegel unterscheiden:

Typ	Subtyp	Modul	Pegel	Schwelle log. 0	Schwelle log. 1
54	0	X-C16-3i/L	Logik	< 2,2 Volt	> 4 Volt
54	1	X-C16-3i/P	Prozess	< 5 Volt	> 13 Volt
54	2	X-C16-3i/T	TTL	< 0,8 Volt	> 2,4 Volt

Auf Wunsch können auch gemischte Bestückungen und andere Schwellen eingestellt werden.

Zusätzlich bietet das Modul 8 einzeln opto-entkoppelte Ausgänge. Sie können per Software gesetzt werden und haben keine feste Zuordnung zu den Zählerkanälen oder zu den Eingängen.

10.8.2. Modul-Device-Treiber

10.8.2.1. Installation

Der Modul-Device-Treiber für das OsX hat die Programmnummer 8036h und den Dateinamen mxc163.exe. Der Modul-Device-Treiber für Windows hat den Namen mxc163.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8036, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8036

10.8.2.2. Kanaleigenschaftsstruktur CPS_XC163_A

Die CPS für das Modul hat den Namen CPS_XC163_A. CPS_XC163_A wurde gegenüber CPS_XC163 um den Parameter *ulCallbackEvent* erweitert.

10.8.2.3. LED

Um auf die LED des Moduls zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_LED*	Kanal zu einer LED
.usIndexFirst	*0*	Nummer der LED
.usIndexLast	*0*	Nummer der LED
.usFlags	*0*	Keine Bedeutung
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usMode	*0*	Keine Bedeutung

Eingabe- und Ausgabedienst

Um die LED ein- bzw. auszuschalten muss eine 1 bzw. 0 in den Kanal geschrieben werden. Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_uchar**
- **max_read_channel_uchar**

10.8.2.4. Digitale Eingänge

Das Modul hat 12 digitale Eingänge (DIN-0...DIN-11). Die Eingänge können entweder direkt oder aus einem internen Zwischenspeicher (Latch) gelesen werden.

Das Übertragen der aktuellen Eingangszustände in den Zwischenspeicher (Latchen) kann über einen externen Eingang und/oder per Software erfolgen. Das softwaremäßige Latchen erfolgt über einen DEVICE_TRIGGER-Kanal (siehe „Latchen der digitalen Eingänge per Software“) und ist auch dann möglich, wenn bereits ein externer Latcheingang verwendet wird. Die Quelle für das externe Latchen wird mit einem Steuer-Kanal (siehe „Definition des externen Latch-Eingangs für die digitalen Eingänge“) festgelegt. Um auf die digitalen Eingänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DIN*	Kanal auf einen digitalen Eingang
.usIndexFirst	*0 ... 11*	Nummer des ersten Eingangs
.usIndexLast	*0 ... 11*	Nummer des letzten Eingangs
.usFlags	*0*	Keine Bedeutung
	_CP_EXCLUSIVE	Falls der Kanal mit Callback Funktionalität geöffnet wird, muss diese Flag gesetzt sein
		Es wird kein Ereignis signalisiert
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
	IO_MODE_LATCH	Werte aus internem Zwischenspeicher lesen.
.usWriteMode	*0*	Keine Bedeutung
.usMode	*0*	Keine Bedeutung
.ulCallbackEvent[1]	*0*	Es wird kein Ereignis signalisiert
	XC163_EVENT_NEG_EDGE	Eine positive Flanke wird signalisiert
	XC163_EVENT_POS_EDGE	Eine negative Flanke wird signalisiert
		Da ein DIN nur eine positive oder negative Flanke signalisieren kann, dürfen die beiden Werte nicht gleichtzeitig gesetzt werden.

[1] bei Benutzung von CPS-Typ CPS_XC163_A

Eingabedienst

Der Datentyp des Kanals ist DATA_USHORT, der Zugriff erfolgt mit:

- **max_read_channel_ushort**

Callback-Funktion

Über das Strukturelement ulCallbackEvent kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 2 ULONG-Werte übergeben. Im ersten ULONG-Wert wird die entsprechende Konstante die in *ulCallbackEvent* angegeben wurde zurückgegeben. Der zweite ULONG-Wert zeigt an, bei welchen DINs eine Flanke aufgetreten ist. Die unteren 2 Byte signalisieren dabei, dass eine positive Flanke eines DINs aufgetreten ist. Die oberen 2 Byte signalisieren dabei, dass eine negative Flanke eines DINs aufgetreten ist. Jedes Bit repräsentiert einen DIN. Die Daten sind rechtsbündig angeordnet. Bei einem Kanal zu einem DIN-5...11 würde also Bit-0...6 eine pos. Flanke und Bit-16...22 eine neg. Flanke an DIN-5...11 anzeigen.

10.8.2.5. Definition des externen Latch-Eingangs für die digitalen Eingänge

Mit diesem Device wird festgelegt, welcher digitale Eingang die digitalen Eingänge in den internen Zwischenspeicher (Latch) des Moduls übertragen soll. Um darauf zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Kanal auf ein Kontroll-Device
.usIndexFirst	*XC163_DIN_LATCH_CTRL*	Nummer des Kontroll-Devices
.usIndexLast	*XC163_DIN_LATCH_CTRL*	Nummer des Kontroll-Devices
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usMode	*0*	Keine Bedeutung

Eingabedienst

Der Datentyp des Kanals ist DATA_USHORT, der Zugriff erfolgt mit:

- **max_write_channel_ushort**
- **max_read_channel_ushort**

Der zu schreibende bzw. gelesene Wert resultiert aus einer Oder-Verknüpfung der folgenden Flags:

Flags	Bedeutung
_XC163_DIN_x	DIN-x (x = 0 ... 11) ist Latch-Eingang
_XC163_POS_EDGE	Positive Flanke löst Latch-Impuls aus
_XC163_NEG_EDGE	Negative Flanke löst Latch-Impuls aus
_XC163_ENABLE	Externer Latch-Eingang ist aktiviert
_XC163_DISABLE	Externer Latch-Eingang ist deaktiviert

Beispiel:

Der folgende Befehl definiert, dass eine positive Flanke an DIN-7 alle DINs latcht:

```
max_write_channel_ushort (hChan, _XC163_DIN_7 | _XC163_POS_EDGE | _XC163_ENABLE)
```

10.8.2.6. Latchen der digitalen Eingänge per Software

Mit diesem Device können die digitalen Eingänge softwaremäßig in den Zwischenspeicher übertragen (gelatcht) werden. Das softwaremäßige Zwischenspeichern kann auch dann erfolgen, wenn bereits ein externer Latch-Eingang aktiviert ist. Um darauf zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TRIGGER*	Kanal auf einen Trigger
.usIndexFirst	*XC163_DIN_TRIGGER*	Nummer des Triggers
.usIndexLast	*XC163_DIN_TRIGGER*	Nummer des Triggers
.usFlags	*0*	Keine Bedeutung
.usMode	*0*	Keine Bedeutung

Ausgabedienst

Der Datentyp des Kanals ist DATA_VOID, der Zugriff erfolgt mit:

- **max_trigger_channel**

10.8.2.7. Filterfunktion der digitalen Eingänge

Mit Hilfe dieses Kanals kann die Abtastfrequenz, mit der die digitalen Eingänge abgetastet wrden, eingestellt werden. Störungen, die während des Flankeneinstiegs bzw. Flankenabfalls auftreten, können damit herausgefiltert werden, wenn innerhalb

dieser Zeitspanne nur ein Abtastzeitpunkt liegt. Die Einstellungen gelten für alle digitalen Eingänge.

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Kanal auf ein Kontroll-Device
.usIndexFirst	*XC163_INPUT-FILTER*	
.usFlags	*0*	reservierter Parameter
.usMode	*0*	reservierter Parameter

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_ULONG. Um die Abtastfrequenz zu setzten bzw. zurückzulesen, können folgende Funktionen verwendet werden:

- **max_write_channel_ulong**
- **max_read_channel_ulong**

Mögliche Werte für das Schreiben bzw. Lesen sind:

XC163_20MHZ
XC163_10MHZ
XC163_5MHZ
XC163_2MHZ
XC163_1MHZ
XC163_500KHZ
XC163_200KHZ
XC163_100KHZ
XC163_50KHZ
XC163_20KHZ
XC163_10KHZ
XC163_5KHZ
XC163_2KHZ
XC163_1KHZ

10.8.2.8. Digitale Ausgänge

Um auf die digitalen Ausgänge des Moduls zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DOUT*	Kanal auf einen digitalen Ausgang
.usIndexFirst	*0 ... 7*	Nummer des ersten Ausgangs
.usIndexLast	*0 ... 7*	Nummer des letzten Ausgangs
.usFlags	*0*	Keine Bedeutung
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usMode	*0*	Keine Bedeutung

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR, der Zugriff erfolgt mit:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

10.8.2.9. Grundlagen Zähler

Zähleingänge

Das Modul verfügt über drei Zähler (0 ... 2). Jeder Zähler besitzt zwei Zähleingänge (A, B), die mit verschiedenen digitalen Eingängen (DIN) verbunden werden können. Hierzu kann beim Öffnen eines MDD-Kanals im Strukturelement *.usCounterInput* angegeben werden, welche DINs man mit dem jeweiligen Zähler verbinden will. Wird beim Öffnen eines Zählerkanals das Flag *_XC163_NEG_EDGE_x* (x = A, B) gesetzt, wird auf eine negative Flanke am entsprechenden Zähleingang reagiert, ansonsten auf eine positive Flanke.

Zählerausgänge

Bei einigen Betriebsarten der Zähler ist es möglich, bei einem Zählerunterlauf einen digitalen Ausgang (DOUT) umzuschalten. Dabei ist dem Zähler 0 DOUT-0, dem Zähler 1 DOUT-1 und dem Zähler 2 DOUT-2 zugeordnet. Sind mehrere Zähler kaskadiert, wird immer der DOUT des ersten Zählers verwendet.

Kaskadierung

Jeder Zähler hat eine Breite von 16 Bit. Durch ein Zusammenschalten von zwei bzw. drei Zählern (Kaskadieren) kann man die Breite auf 32 bzw. 48 Bit erweitern. Das

Kaskadieren erfolgt beim Öffnen eines MDD-Kanals dadurch, dass man für die Strukturelemente *.usIndexFirst* und *.usIndexLast* unterschiedliche Werte angibt.

Zugriff auf einen Zählerkanal

Der Datentyp ist immer DATA_UCHAR. Das Schreiben bzw. Lesen eines Zählerkanals erfolgt immer mit einem **max_read_channel_block** bzw. **max_write_channel_block** Befehl. Die Anzahl der Datenbytes, die dabei übertragen werden, hängt dabei von der Breite des Zählers ab:

Anzahl der (kaskadierten) Zähler	Anzahl der Datenbytes
1	2
2	4
3	6

Steuern der Zähler

Das Steuern (Starten, Stoppen, Latchen, Laden und Zurücksetzen) von Zählern kann auf zwei Arten erfolgen:
Zum einen können Flanken an den digitalen Eingängen Steuerbefehle auslösen (siehe „Definition der externen Zähler-Steuereingänge“).
Zum anderen können die Zähler per Software gesteuert werden (siehe „Steuern der Zähler per Software“).

Umschalten der Referenz-Frequenzen

Der im CPS-Element *usReferenceTime* angegebenen Wert kann dynamisch während einer Messung mit Hilfe des Kanal-Steuerkommandos *CTRL_SET_TIME* umgeschaltet werden.
Das folgende Beispiel schaltet die Referenz-Frequenz auf 1 KHz um:

```
ulTime = XC163_1KHZ;
max_channel_control(hChannel, CTRL_SET_TIME, 4, (void*)&ulTime);
```

Status der Messung

Mit Hilfe des Kanal-Steuerkommandos INFO_DATA kann der Status der aktuellen Messung ermittelt werden.
Falls neue Daten vorliegen ist das Bit *_XC163_NEW_DATA* gesetzt.

Beispiel:

```
usSize = 4;
max_channel_info(hChannel, INFO_DATA, &usSize, (void*)&ulStatus);
```

Zähler-Über- / Unterlauf

Liegt seit dem letzten Auslesen des Zählers ein Zähler-Überlauf bzw. ein Unterlauf vor, so liefert der Eingabedienst der Fehler ERR_OVERFLOW bzw. ERR_UNDERFLOW zurück. Liegen mehrere Überläufe/Unterläufe vor, so liefert der Dienst den Fehler ERR_INVALID_DATA. Diese Rückgabewerte treten nur auf, wenn keine Callback-Funktion für den Overflow bzw. Underflow definiert ist.

10.8.2.10. Definition der externen Zähler-Steuereingänge

Mit diesem Device kann definiert werden, welche digitalen Eingänge als Steuereingänge für die Zähler verwendet werden können. Insgesamt stehen vier Steuereingänge (0 ... 3) zur Verfügung. Diese können verwendet werden, um die Zähler zu steuern, d.h.:

- der Zähler kann mit den Werten geladen werden, die zuvor mit einem Schreibbefehl in einen internen Zwischenspeicher des Zählerkanals geschrieben wurden
- der Zähler kann gestartet werden
- der Zähler kann angehalten werden
- der aktuelle Zählerstand kann in einem Zwischenspeicher abgelegt werden
- der aktuelle Zählerstand kann auf 0 zurückgesetzt werden

Um einen Steuereingang für einen Zähler zu nutzen, muss beim Öffnen des entsprechenden Zählerkanals angegeben werden, welcher Steuereingang (0 ... 3) einen Befehl auslösen soll.

Um einen Steuereingang zu definieren, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Kanal auf ein Kontroll-Device
.usIndexFirst	*XC163_COUNTER_HW_CTRL*	Externer Zähler-Steuereingang
.usIndexLast	*XC163_COUNTER_HW_CTRL*	Externer Zähler-Steuereingang
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usMode	*0*	Keine Bedeutung

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_USHORT, der Zugriff erfolgt mit:

- **max_write_channel_ushort**
- **max_read_channel_ushort**

Mögliche Werte bei einen Schreib- bzw. Lesezugriff sind:

Wert	Bedeutung
_XC163_DIN_x_y_TO_CTRL_0_1	Verbindet die digitalen Eingänge x/y (x/y = 0/1, 2/3, 6/7, 8/9) mit den Steuereingängen 0 und 1
_XC163_DIN_x_y_TO_CTRL_2_3	Verbindet die digitalen Eingänge x/y (x/y = 0/1, 4/5, 6/7, 10/11) mit den Steuereingängen 2 und 3
_XC163_CTRL_x_NEG_EDGE	Dieses Flag legt fest, dass eine negative Flanke an dem Steuereingang x (x = 0 ... 3) einen Befehl auslöst.

Durch eine Oder-Verknüpfung der Konstanten können die Steuereingänge gleichzeitig gesetzt werden.

Beispiel:

Der folgende Befehl verbindet DIN-0 und DIN-1 mit dem Steuereingang 0 und 1 und DIN-10 und DIN-11 mit dem Steuereingang 2 und 3.

```
max_write_channel_ushort (hChan,
                    _XC163_DIN_0_1_TO_CTRL_0_1 | _XC163_DIN_10_11_TO_CTRL_2_3)
```

10.8.2.11. Steuern der Zähler per Software

Dieses Device erlaubt es, die Zähler zu steuern, d.h.:

- der Zähler kann mit den Werten geladen werden, die zuvor mit einem Schreibbefehl in einen internen Zwischenspeicher des Zählerkanals geschrieben wurden
- der Zähler kann gestartet werden
- der Zähler kann angehalten werden
- der aktuelle Zählerstand kann in einem Zwischenspeicher abgelegt werden
- der aktuelle Zählerstand kann auf 0 zurückgesetzt werden

Um darauf zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Kanal auf ein Kontroll-Device
.usIndexFirst	*XC163_COUNTER_SW_CTRL*	Zähler Software-Steuerung
.usIndexLast	*XC163_COUNTER_SW_CTRL*	Zähler Software-Steuerung
.usFlags	*0*	Keine Bedeutung
.usReadMode	*0*	Keine Bedeutung
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usMode	*0*	Keine Bedeutung

Ausgabedienst

Der Datentyp des Kanals ist DATA_USHORT, der Zugriff erfolgt mit:

- **max_write_channel_ushort**

Mögliche Werte für einen Schreibzugriff sind:

Wert	Bedeutung
_XC163_COUNTER_x	Zähler x (x = 0 ... 2), an den das Kommando gesendet werden soll.
_XC163_START_COUNTER	Zähler starten
_XC163_STOP_COUNTER	Zähler anhalten
_XC163_LOAD_COUNTER	Zähler laden
_XC163_LATCH_COUNTER	Die aktuellen Zählerstände in den internen Zwischenspeicher übernehmen
_XC163_RESET_COUNTER	Zählerstände auf 0 setzen

Durch eine Oder-Verknüpfung der Konstanten können auch mehrere Zähler gleichzeitig gesteuert werden.

Beispiel:

Der folgende Befehl startet den Zähler 0 und den Zähler 2:

```
max_write_channel_ushort (hChannel, _XC163_COUNTER_0 | _XC163_COUNTER_2 |
_XC16_START_COUNTER)
```

10.8.2.12. Aufwärtszähler

Mit diesem Device kann ein Aufwärtszähler realisiert werden. An den Zähleingang A werden die Zählimpulse gelegt. Zähleingang B wird nicht benutzt. Um auf den Zähler zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_UP_COUNTER*	Kanal zu einem Aufwärtszähler
.usIndexFirst	*0 ... 2*	Nummer des ersten Aufwärtszählers
.usIndexLast	*0 ... 2*	Nummer des letzten Aufwärtszählers (Ist dieser Wert größer als *.usIndexFirst*, werden die Zähler von *.usIndexFirst* bis *.usIndexLast* kaskadiert.)
.usFlags	*_CP_EXCLUSIVE* *_CP_SYNC_CALLBACK*	Der Zugriff erfolgt immer exklusiv Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt.

Strukturelement	Werte	Bedeutung
		Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_DIRECT* *IO_MODE_LATCH*	Direkter Lesezugriff Zählerwert wird aus Zwischenspeicher gelesen
.usWriteMode	*IO_MODE_DIRECT* *IO_MODE_LATCH*	Direkter Schreibzugriff Der Zählerwert wird in Zwischenspeicher geschrieben
.usMode	*0* *_XC163_AUTORUN*	Keine Bedeutung Zähler startet sofort nach Öffnen des Kanals mit Startwert = 0.
.usCounterInput	*_XC163_DIN_x_y* *_XC163_NEG_EDGE_A* *_XC163_NEG_EDGE_B*	Zähleingang (siehe Tabelle) Zählen bei negativer Flanke an Eingang A Zählen bei negativer Flanke an Eingang B
.ulControlInput	*0* *_XC163_START_CTRL_x* *_XC163_STOP_CTRL_x* *_XC163_LOAD_CTRL_x* *_XC163_LATCH_CTRL_x* *_XC163_RESET_CTRL_x*	Keine Bedeutung Start, Stop, Laden, Zwischenspeichern und Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)
.ulCallbackEvent	*0* *XC163_EVENT_OVERFLOW* *XC163_EVENT_START* *XC163_EVENT_STOP* *XC163_EVENT_LOAD* *XC163_EVENT_LATCH* *XC163_EVENT_RESET*	Es wird kein Ereignis signalisiert Der Zähler ist übergelaufen Der Zähler wurde durch einem Steuerkanal gestartet Der Zähler wurde durch einem Steuerkanal gestoppt Der Zähler wurde durch einem Steuerkanal geladen Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße hängt dabei von der Anzahl der Zähler ab. Je Zähler werden 2 Byte benötigt.

Anmerkungen

1. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist *.usMode = _XC163_AUTORUN*, startet der Zähler nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zähler nach dem Beschreiben des Kanals mit dem geschriebenem Wert.
- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibzugriff der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden sollen, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

2. Lesen der Zählerwerte

Ist *.usReadMode = IO_MODE_DIRECT*, kann der aktuelle Zählerstand durch einen Lesebefehl ermittelt werden. Wurde der Lesemodus auf *IO_MODE_LATCH* eingestellt, wird durch einen Lesezugriff der im Latch zwischengespeicherte Zählerstand ausgelesen. Ein Latchbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y***
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11
1	x/y = 2/3, 8/9
2	x/y = 4/5, 10/11

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

Beispiel:

Nachfolgend wird ein Kanal auf einen Aufwärtszähler geöffnet. Dabei werden die Zähler 0 und 1 zu einem Zähler kaskadiert. Der Zähler startet sofort beim Öffnen des Kanals.

```
CPS_XC163 rcUpCounter;
MAXCHLHND hUpCounter;
MAX_ERROR Error;
ULONG ulSize;
UCHAR aucData[4];

// CPS ausfuellen
rcUpCounter.usDevice = DEVICE_UP_COUNTER;
rcUpCounter.usIndexFirst = 0;
rcUpCounter.usIndexLast = 1;
rcUpCounter.usFlags = _CP_EXCLUSIVE;
rcUpCounter.usReadMode = IO_MODE_DIRECT;
rcUpCounter.usWriteMode = IO_MODE_DIRECT;
rcUpCounter.usMode = _XC163_AUTORUN;
rcUpCounter.usCounterInput = _XC163_DIN_0_1;
rcUpCounter.ulControlInput = 0;
// Kanal oeffnen
Error = max_open_channel(hMdd, sizeof (rcUpCounter), &rcUpCounter,
                                                NULL, NULL, &hUpCounter);
// Kanal lesen
ulSize = 4;
Error = max_read_channel_block(hUpCounter, &ulSize, (void*)aucData);
```

10.8.2.13. Abwärtszähler

Mit diesem Device kann ein Abwärtszähler realisiert werden. An den Zähleingang A werden die Zählimpulse gelegt. Zähleingang B wird nicht benutzt. Um auf den Zähler zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DOWN_COUNTER*	Kanal auf einen Abwärtszähler
.usIndexFirst	*0 ... 2*	Nummer des ersten Zählers
.usIndexLast	*0 ... 2*	Nummer des letzten Zählers (Ist dieser Wert größer als *.usIndexFirst*, werden die Zähler von *.usIndexFirst* bis *.usIndexLast* kaskadiert.)
.usFlags	*_CP_EXCLUSIVE* *_CP_SYNC_CALLBACK*	Der Zugriff erfolgt immer exklusiv Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff

Strukturelement	Werte	Bedeutung
	IO_MODE_LATCH	Der Zählerwert wird aus Zwischenspeicher gelesen
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
	IO_MODE_LATCH	Der Zählerwert wird in Zwischenspeicher geschrieben
.usMode	*0*	Keine Bedeutung
	_XC163_AUTORUN	Zähler startet sofort nach Öffnen des Kanals mit Startwert = 0.
	_XC163_OUTPUT	Der zum Zähler gehörige DOUT schaltet bei jedem Zählerunterlauf.
	_XC163_AUTOLOAD_ UNDERFLOW	Nach einem Zählerunterlauf wird der Zähler wieder mit dem Startwert geladen. Ansonsten wird er mit dem Maximalwert geladen.
.usCounterInput	*_XC163_DIN_x_y*	Zähleingang (siehe Tabelle)
	_XC163_NEG_EDGE_A	Zählen bei negativer Flanke an Eingang A
	_XC163_NEG_EDGE_B	Zählen bei negativer Flanke an Eingang B
.ulControlInput	*0*	Keine Bedeutung
	_XC163_START_CTRL_x	Start,
	_XC163_STOP_CTRL_x	Stop,
	_XC163_LOAD_CTRL_x	Laden,
	_XC163_LATCH_CTRL_x	Zwischenspeichern und
	_XC163_RESET_CTRL_x	Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)
ulCallbackEvent	*0*	Es wird kein Ereignis signalisiert
	XC163_EVENT_ UNDERFLOW	Der Zähler ist untergelaufen
	XC163_EVENT_START	Der Zähler wurde durch einem Steuerkanal gestartet
	XC163_EVENT_STOP	Der Zähler wurde durch einem Steuerkanal gestoppt
	XC163_EVENT_LOAD	Der Zähler wurde durch einem Steuerkanal geladen
	XC163_EVENT_LATCH	Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert
	XC163_EVENT_RESET	Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße hängt dabei von der Anzahl der Zähler ab. Je Zähler werden 2 Byte benötigt.

Anmerkungen

1. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist der Mode *_XC163_AUTORUN* aktiviert, startet der Zähler nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zähler nach dem Beschreiben des Kanals mit dem geschriebenem Wert.
- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibzugriff der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

2. Lesen der Zählerwerte

Ist *.usReadMode = IO_MODE_DIRECT*, kann der aktuelle Zählerstand durch einen Lesebefehl ermittelt werden. Wurde der Lesemodus auf *IO_MODE_LATCH* eingestellt, wird durch einen Lesezugriff der im Latch zwischengespeicherte Zählerstand ausgelesen. Ein Latchbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y***
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11
1	x/y = 2/3, 8/9
2	x/y = 4/5, 10/11

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

Beispiel:

Siehe Aufwärtszähler.

10.8.2.14. Auf- und Abwärtszähler

Mit diesem Device kann ein Auf- und Abwärtszähler realisiert werden. Um darauf zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE _UP_DOWN_COUNTER*	Kanal auf einen Auf- und Abwärtszähler
.usIndexFirst	*0 ... 2*	Nummer des ersten Zählers
.usIndexLast	*0 ... 2*	Nummer des letzten Zählers (Ist dieser Wert größer als *.usIndexFirst*, werden die Zähler von *.usIndexFirst* bis *.usIndexLast* kaskadiert.)
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werde die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
	IO_MODE_LATCH	Der Zählerwert wird aus Zwischenspeicher gelesen.
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
	IO_MODE_LATCH	Der Zählerwert wird in Zwischenspeicher geschrieben.
.usMode	*_XC163_MODE_A*	Impulse am Zähleingang A werden in Aufwärtsrichtung gezählt, Impulse am Zähleingang B in Abwärtsrichtung.
	_XC163_MODE_B	Die Zählimpulse werden am Zähleingang A angelegt. Über den Zähleingang B wird die Zählrichtung festgelegt (0 = aufwärts; 1 = abwärts).
	_XC163_AUTORUN	Der Zähler startet nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
	_XC163_OUTPUT	Der zum Zähler gehörige DOUT schaltet bei jedem Zählerunterlauf.
	_XC163_AUTOLOAD_ UNDERFLOW	Nach einem Zählerunterlauf wird der Zähler wieder mit dem Startwert geladen. Ansonsten wird er mit dem Maximalwert geladen.
.usCounterInput	*_XC163_DIN_x_y*	Zähleingang (siehe Tabelle)
	_XC163_NEG_EDGE_A	Zählen bei negativer Flanke an Eingang A

Strukturelement	Werte	Bedeutung
	_XC163_NEG_EDGE_B	Zählen bei negativer Flanke an Eingang B
.ulControlInput	*0*	Keine Bedeutung
	_XC163_START_CTRL_x	Start,
	_XC163_STOP_CTRL_x	Stop,
	_XC163_LOAD_CTRL_x	Laden,
	_XC163_LATCH_CTRL_x	Zwischenspeichern und
	_XC163_RESET_CTRL_x	Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)
ulCallbackEvent	*0*	Es wird kein Ereignis signalisiert
	XC163_EVENT_OVERFLOW	Der Zähler ist übergelaufen
	XC163_EVENT_ UNDERFLOW	Der Zähler ist untergelaufen
	XC163_EVENT_START	Der Zähler wurde durch einem Steuerkanal gestartet
	XC163_EVENT_STOP	Der Zähler wurde durch einem Steuerkanal gestoppt
	XC163_EVENT_LOAD	Der Zähler wurde durch einem Steuerkanal geladen
	XC163_EVENT_LATCH	Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert
	XC163_EVENT_RESET	Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße hängt dabei von der Anzahl der Zähler ab. Je Zähler werden 2 Byte benötigt.

Anmerkungen

1. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist der Mode *_XC163_AUTORUN* aktiviert, startet der Zähler nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zähler nach dem Beschreiben des Kanals mit dem geschriebenem Wert.
- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibzugriff der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des

geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

2. Lesen der Zählerwerte

Ist *.usReadMode = IO_MODE_DIRECT*, kann der aktuelle Zählerstand durch einen Lesebefehl ermittelt werden. Ist *.usReadMode = IO_MODE_LATCH*, wird durch einen Lesezugriff der im Latch zwischengespeicherte Zählerstand ausgelesen. Ein Latchbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y***
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11
1	x/y = 2/3, 8/9
2	x/y = 4/5, 10/11

Beispiel:

Nachfolgend wird ein Kanal auf einen Auf- / Abwärtszähler im Mode A geöffnet. Dabei werden die Zähler 0 bis 2 zu einem Zähler kaskadiert. Der Zähler startet sofort beim Öffnen des Kanals.

```
CPS_XC163 rcUpDownCounter;
MAXCHLHND hUpDownCounter;
MAX_ERROR Error;
ULONG ulSize;
UCHAR aucData[6];

// CPS ausfuellen
rcUpDownCounter.usDevice = DEVICE_UP_DOWN_COUNTER;
rcUpDownCounter.usIndexFirst = 0;
rcUpDownCounter.usIndexLast = 2;
rcUpDownCounter.usFlags = _CP_EXCLUSIVE;
rcUpDownCounter.usReadMode = IO_MODE_DIRECT;
rcUpDownCounter.usWriteMode = IO_MODE_DIRECT;
```

```
rcUpDownCounter.usMode = _XC163_MODE_A | _XC163_AUTORUN;
rcUpDownCounter.usCounterInput = _XC163_DIN_0_1;
rcUpDownCounter.ulControlInput = 0;

// Kanal oeffnen
Error = max_open_channel(hMdd, sizeof (rcUpDownCounter), &rcUpDownCounter,
                                                NULL, NULL, &hUpDownCounter);

// Kanal lesen
ulSize = 6;
Error = max_read_channel_block(hUpDownCounter, &ulSize, (void*)aucData);
```

Beispiel:

Das nachfolgende Beispiel zeigt einen Auf- Abwärtszahler, der die Über- und Unterläufe signalisiert.
In der Callback-Funktion werden diese Ereignisse unteschieden.

```
MAX_CALLBACK MyCallbackFunction(MAXCHLHND handle, ULONG param, ULONG size, void*
pData)
{
    ULONG ulData;

    ulData = *(ULONG*)pData;

    if ( (ulData & XC163_EVENT_OVERFLOW) != 0)
    {
        ... es ist ein Überlauf aufgetreten
    }

    if ( (ulData & XC163_EVENT_UNDERFLOW) != 0)
    {
        ... es ist ein Unterlauf aufgetreten
    }
}

void OpenChannel(void)
{
    CPS_XC163_A  cps;

    cps.usDevice = DEVICE_UP_DOWN_COUNTER;
    cps.usIndexFirst = 0;
    cps.usIndexLast = 0;
    cps.usFlags = _CP_EXCLUSIVE;
    cps.usReadMode = IO_MODE_DIRECT;
    cps.usWriteMode = IO_MODE_DIRECT;
    cps.usMode = _XC163_MODE_A | _XC163_AUTORUN;
    cps.usCounterInput = _XC163-DIN_0_1;
    cps.ulControlInput = 0;
    cps.ulCallbackEvent = XC163_EVENT_OVERFLOW | XC163_EVENT_UNDERFLOW;

    // öffnen des Kanals mit Callback-Funktionalität
    max_open_channel(g_hMdd, sizeof (cps), &cps, MyCallbackFunction, NULL,
&hUpCounter);

}
```

10.8.2.15. Timer

Mit diesem Device kann ein Timer realisiert werden. Der Zähler arbeitet als Abwärtszähler. Die Zählsignale kommen entweder von einem modulinternen Taktgeber oder von einem externen Signal, das an Zähleingang A anliegen muss. Zähleingang B wird nicht benutzt. Es muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TIMER*	Kanal auf einen Timer
.usIndexFirst	*0 ... 2*	Nummer des ersten Timers
.usIndexLast	*0 ... 2*	Nummer des letzten Timers (Ist dieser Wert größer als *.usIndexFirst*, werden die Zähler von *.usIndexFirst* bis *.usIndexLast* kaskadiert.)
.usFlags	*_CP_EXCLUSIVE* *_CP_SYNC_CALLBACK*	Der Zugriff erfolgt immer exklusiv. Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_DIRECT* *IO_MODE_LATCH*	Direkter Lesezugriff Der Zählerwert wird aus Zwischenspeicher gelesen.
.usWriteMode	*IO_MODE_DIRECT* *IO_MODE_LATCH*	Direkter Schreibzugriff Der Zählerwert wird in Zwischenspeicher geschrieben.
.usMode	*0* *_XC163_CONTINUOUS* *_XC163_OUTPUT*	Keine Bedeutung Der Timer startet nach dem Nulldurchlauf automatisch. Der zum Zähler gehörige DOUT schaltet bei jedem Zählerunterlauf.
.ulReferenceTime	*XC163_EXTERNAL* *XC163_20MHZ* *XC163_10MHZ* *XC163_5MHZ* *XC163_1MHZ* *XC163_500KHZ* *XC163_200KHZ* *XC163_100KHZ* *XC163_50KHZ* *XC163_20KHZ* *XC163_10KHZ* *XC163_5KHZ* *XC163_2KHZ* *XC163_1KHZ*	Referenz-Frequenz

Strukturelement	Werte	Bedeutung
	XC163_500HZ *XC163_200HZ* *XC163_100HZ* *XC163_50HZ* *XC163_20HZ* *XC163_10HZ*	
.usCounterInput	*_XC163_DIN_x_y* *_XC163_NEG_EDGE_A* *_XC163_NEG_EDGE_B*	Zähleingang (siehe Tabelle) Zählen bei negativer Flanke an Eingang A Zählen bei negativer Flanke an Eingang B
.ulControlInput	*0* *_XC163_START_CTRL_x* *_XC163_STOP_CTRL_x* *_XC163_LOAD_CTRL_x* *_XC163_LATCH_CTRL_x* *_XC163_RESET_CTRL_x*	Keine Bedeutung Start, Stop, Laden, Zwischenspeichern und Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)
ulCallbackEvent	*0* *XC163_EVENT_ UNDERFLOW* *XC163_EVENT_START* *XC163_EVENT_STOP* *XC163_EVENT_LOAD* *XC163_EVENT_LATCH* *XC163_EVENT_RESET*	Es wird kein Ereignis signalisiert Der Zähler ist untergelaufen Der Zähler wurde durch einem Steuerkanal gestartet Der Zähler wurde durch einem Steuerkanal gestoppt Der Zähler wurde durch einem Steuerkanal geladen Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße hängt dabei von der Anzahl der Zähler ab. Je Zähler werden 2 Byte benötigt.

Anmerkungen

1. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zähler nach dem Beschreiben des Kanals mit dem geschriebenem Wert.
- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibzugriff der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.usControlInput* mit der entsprechenden Konstanten belegt werden.

2. Lesen der Zählerwerte

Ist *usReadMode = IO_MODE_DIRECT*, kann der aktuelle Zählerstand durch einen Lesebefehl ermittelt werden. Wurde der Lesemodus auf *IO_MODE_LATCH* eingestellt, wird durch einen Lesezugriff der im Latch zwischengespeicherte Zählerstand ausgelesen. Ein Latchbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y***
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11
1	x/y = 2/3, 8/9
2	x/y = 4/5, 10/11

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

Beispiel:

Nachfolgend wird ein Kanal auf einen Timer mit einer Zählfrequenz von 1 kHz geöffnet. Durch das Beschreiben des Kanals startet der Timer und schaltet bei jedem Unterlauf den digitalen Ausgang DOUT-0 um.

```
CPS_XC163  rcTimer;
MAXCHLHND hTimer;
MAX_ERROR Error;
ULONG ulSize;
UCHAR aucData[2];
```

```
// CPS ausfuellen
rcTimer.usDevice = DEVICE_TIMER;
rcTimer.usIndexFirst = 0;
rcTimer.usIndexLast = 0;
rcTimer.usFlags = _CP_EXCLUSIVE;
rcTimer.usReadMode = IO_MODE_DIRECT;
rcTimer.usWriteMode = IO_MODE_DIRECT;
rcTimer.usMode = _XC163_CONTINUOUS | _XC163_OUTPUT;
rcTimer.ulReferenceTime = XC163_1KHZ;
rcTimer.usCounterInput = _XC163_DIN_0_1;
rcTimer.ulControlInput = 0;
// Kanal oeffnen
Error = max_open_channel(hMdd, sizeof (rcTimer), &rcTimer,
                                                NULL, NULL, &hTimer);
// Kanal mit Wert 1000 (=3E8hex) beschreiben und starten
ulSize = 2;
aucData[0] = 0xE8;
aucData[1] = 0x03;
Error = max_write_channel_block(hTimer, &ulSize, (void*)aucData);
```

10.8.2.16. Pulsbreitenmessung

Das Signal, dessen Pulsbreite gemessen werden soll, wird an den Zähleingang B angelegt. Die Referenz-Frequenz, die während der Pulsdauer des Messsignals gezählt wird, kann intern generiert werden oder über den Zähleingang A vorgegeben werden.

Um eine Pulsbreitenmessung durchführen zu können, muss nachfolgende CPS verwendet werden.

Strukturelement	**Werte**	**Bedeutung**
.usDevice	*DEVICE _PULSEWIDTH_COUNTER*	Kanal auf einen Zähler für Pulsbreitenmessung
.usIndexFirst	*0 ... 2*	Nummer des ersten Zählers
.usIndexLast	*0 ... 2*	Nummer des letzten Zählers (Ist dieser Wert größer als *.usIndexFirst*, werden die Zähler von *.usIndexFirst* bis *.usIndexLast* kaskadiert.)
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
	_XC163_NEW_DATA_ONLY	Ist dieses Flag gesetzt, liefert der Lesedienst den Fehler ERR_DATA_NOT_READY zurück falls noch keine neuen Daten bereitstehen. Andernfalls wird das alte Ergebnis zurückgeliefert.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_LATCH*	Der Zählerwert wird aus Zwischenspeicher gelesen.

Strukturelement	Werte	Bedeutung
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
	IO_MODE_LATCH	Der Zählerwert wird in den Zwischenspeicher geschrieben.
.usMode	*0*	Keine Bedeutung
	_XC163_AUTORUN	Der Zählvorgang startet sofort nach dem Öffnen des Kanals (Zähler wird mit 0 vorinitialisiert).
	_XC163_CONTINUOUS	Am Ende der Messung wird sofort eine neue Messung gestartet. Wurde der Zähler mit einem Schreibzugriff auf einen bestimmten Wert voreingestellt, wird dieser Wert jeweils übernommen.
	_XC163_OVERRIDE	Im Zwischenspeicher liegt jeweils das Ergebnis der letzten Messung. Ist dieser Mode nicht aktiviert, wird nur ein neuer Wert in den Zwischenspeicher geschrieben, wenn der alte Wert ausgelesen wurde.
.ulReferenceTime	*XC163_EXTERNAL*	Referenz-Frequenz
	XC163_20MHZ	
	XC163_10MHZ	
	XC163_5MHZ	
	XC163_1MHZ	
	XC163_500KHZ	
	XC163_200KHZ	
	XC163_100KHZ	
	XC163_50KHZ	
	XC163_20KHZ	
	XC163_10KHZ	
	XC163_5KHZ	
	XC163_2KHZ	
	XC163_1KHZ	
	XC163_500HZ	
	XC163_200HZ	
	XC163_100HZ	
	XC163_50HZ	
	XC163_20HZ	
	XC163_10HZ	
.usCounterInput	*_XC163_DIN_x_y*	Zähleingang (siehe Tabelle)
	_XC163_NEG_EDGE_A	Zählen bei negativer Flanke an Eingang A
	_XC163_NEG_EDGE_B	Zählen bei negativer Flanke an Eingang B
.ulControlInput	*0*	Keine Bedeutung
	_XC163_START_CTRL_x	Start,
	_XC163_STOP_CTRL_x	Stop,
	_XC163_LOAD_CTRL_x	Laden,
	_XC163_LATCH_CTRL_x	Zwischenspeichern und

Strukturelement	Werte	Bedeutung
	_XC163_RESET_CTRL_x	Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)
ulCallbackEvent	*0*	Es wird kein Ereignis signalisiert
	XC163_END_OF_MEASUREMENT	Eine Messung ist beendet worden
	XC163_EVENT_OVERFLOW	Der Zähler ist übergelaufen
	XC163_EVENT_START	Der Zähler wurde durch einem Steuerkanal gestartet
	XC163_EVENT_STOP	Der Zähler wurde durch einem Steuerkanal gestoppt
	XC163_EVENT_LOAD	Der Zähler wurde durch einem Steuerkanal geladen
	XC163_EVENT_LATCH	Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert
	XC163_EVENT_RESET	Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße hängt dabei von der Anzahl der Zähler ab. Je Zähler werden 2 Byte benötigt.

Anmerkungen

1. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist *.usMode = _XC163_AUTORUN*, startet der Zähler nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zählvorgang nach dem Beschreiben des Kanals. Der Zähler wird dabei mit dem geschriebenem Wert vor-initialisiert.
- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibbefehl der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

2. Lesen der Zählerwerte

Gelesen wird immer aus einem internen Zwischenspeicher.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y***
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11
1	x/y = 2/3, 8/9
2	x/y = 4/5, 10/11

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

Beispiel:

Nachfolgend wird ein Kanal für eine Pulsbreitenmessung mit einer Zählfrequenz von 1 kHz geöffnet.

```
CPS_XC163 rcPulseWidth;
MAXCHLHND hPulseWidth;
MAX_ERROR Error;
ULONG ulSize;
UCHAR aucData[2];

// CPS ausfuellen
rcPulseWidth.usDevice = DEVICE_PULSEWIDTH_COUNTER;
rcPulseWidth.usIndexFirst = 0;
rcPulseWidth.usIndexLast = 0;
rcPulseWidth.usFlags = _CP_EXCLUSIVE;
rcPulseWidth.usReadMode = IO_MODE_LATCH;
rcPulseWidth.usWriteMode = IO_MODE_DIRECT;
rcPulseWidth.usMode = _XC163_AUTORUN | _XC163_OVERRIDE | _XC163_CONTINUOUS;
rcPulseWidth.ulReferenceTime = XC163_1KHZ;
rcPulseWidth.usCounterInput = _XC163_DIN_0_1;
rcPulseWidth.ulControlInput = 0;

// Kanal oeffnen
Error = max_open_channel(hMdd, sizeof (rcPulseWidth), &rcPulseWidth,
                                                NULL, NULL, &hPulseWidth);

// Messwerte lesen
ulSize = 2;
Error = max_read_channel_block(hPulseWidth, &ulSize, (void*)aucData);
```

10.8.2.17. Frequenzmessung

Die zu messende Frequenz wird an den Zähleingang A angelegt. Die Torzeit, während der die Messfrequenz gezählt wird, kann intern generiert werden oder über den Zähleingang B vorgegeben werden.

Um eine Frequenzmessung durchführen zu können, muss nachfolgende CPS verwendet werden.

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE _FREQUENCY_COUNTER*	Kanal auf einen Zähler für Frequenzmessung
.usIndexFirst	*0 ... 2*	Nummer des ersten Zählers
.usIndexLast	*0 ... 2*	Nummer des letzten Zählers (Ist dieser Wert größer als *.usIndexFirst*, werden die Zähler von *.usIndexFirst* bis *.usIndexLast* kaskadiert.)
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv.
	_XC163_NEW_DATA_ONLY	Ist dieses Flag gesetzt, liefert der Lesedienst den Fehler ERR_DATA_NOT_READY zurück falls noch keine neuen Daten bereitstehen. Andernfalls wird das alte Ergebnis zurückgeliefert.
.usReadMode	*IO_MODE_LATCH*	Der Zählerwert wird aus Zwischenspeicher gelesen
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
	IO_MODE_LATCH	Der Zählerwert wird in Zwischenspeicher geschrieben.
.usMode	*0*	Keine Bedeutung
	_XC163_AUTORUN	Der Zählvorgang startet sofort nach dem Öffnen des Kanals (Zähler wird mit 0 vorinitialisiert).
	_XC163_CONTINUOUS	Am Ende der Messung wird sofort eine neue Messung gestartet. Wurde der Zähler mit einem Schreibzugriff auf einen bestimmten Wert voreingestellt, wird dieser Wert jeweils übernommen.
	_XC163_OVERRIDE	Im Zwischenspeicher liegt jeweils das Ergebnis der letzten Messung. Ist dieser Mode nicht aktiviert, wird nur ein neuer Wert in den Zwischenspeicher geschrieben, wenn der alte Wert ausgelesen wurde.

Strukturelement	Werte	Bedeutung
.ulReferenceTime	*XC163_EXTERNAL* *XC163_50NS* *XC163_100NS* *XC163_200NS* *XC163_500NS* *XC163_1US* *XC163_2US* *XC163_5US* *XC163_10US* *XC163_20US* *XC163_50US* *XC163_100US* *XC163_200US* *XC163_500US* *XC163_1MS* *XC163_2MS* *XC163_5MS* *XC163_10MS* *XC163_20MS* *XC163_50MS* *XC163_100MS*	Torzeit
.usCounterInput	*_XC163_DIN_x_y* *_XC163_NEG_EDGE_A* *_XC163_NEG_EDGE_B*	Zähleingang (siehe Tabelle) Zählen bei negativer Flanke an Eingang A Zählen bei negativer Flanke an Eingang B
.ulControlInput	*0* *_XC163_START_CTRL_x* *_XC163_STOP_CTRL_x* *_XC163_LOAD_CTRL_x* *_XC163_LATCH_CTRL_x* *_XC163_RESET_CTRL_x*	Keine Bedeutung Start, Stop, Laden, Zwischenspeichern und Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)

Strukturelement	Werte	Bedeutung
ulCallbackEvent	*0*	Es wird kein Ereignis signalisiert
	XC163_END_OF_ MEASUREMENT	Eine Messung ist beendet worden
	XC163_EVENT_OVERFLOW	Der Zähler ist übergelaufen
	XC163_EVENT_START	Der Zähler wurde durch einem Steuerkanal gestartet
	XC163_EVENT_STOP	Der Zähler wurde durch einem Steuerkanal gestoppt
	XC163_EVENT_LOAD	Der Zähler wurde durch einem Steuerkanal geladen
	XC163_EVENT_LATCH	Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert
	XC163_EVENT_RESET	Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße hängt dabei von der Anzahl der Zähler ab. Je Zähler werden 2 Byte benötigt.

Anmerkungen

1. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist *.usMode = _XC163_AUTORUN*, startet der Zähler nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zählvorgang nach dem Beschreiben des Kanals. Der Zähler wird dabei mit dem geschriebenem Wert vor-initialisiert.
- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibbefehl der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

2. Lesen der Zählerwerte

Gelesen wird immer aus einem internen Zwischenspeicher.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y***
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11
1	x/y = 2/3, 8/9
2	x/y = 4/5, 10/11

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

Beispiel:

Nachfolgend wird ein Kanal auf eine Frequenzmessung mit einer Torzeit von 1 ms geöffnet.

```
CPS_XC163  rcFrequency;
MAXCHLHND hFrequency;
MAX_ERROR Error;
ULONG ulSize;
UCHAR aucData[2];

// CPS ausfuellen
rcFrequency.usDevice = DEVICE_FREQUENCY_COUNTER;
rcFrequency.usIndexFirst = 0;
rcFrequency.usIndexLast = 0;
rcFrequency.usFlags = _CP_EXCLUSIVE;
rcFrequency.usReadMode = IO_MODE_LATCH;
rcFrequency.usWriteMode = IO_MODE_DIRECT;
rcFrequency.usMode = _XC163_AUTORUN | _XC163_OVERRIDE | _XC163_CONTINUOUS;
rcFrequency.ulReferenceTime = XC163_1MS;
rcFrequency.usCounterInput = _XC163_DIN_0_1;
rcFrequency.ulControlInput = 0;

// Kanal oeffnen
Error = max_open_channel(hMdd, sizeof (rcFrequency), &rcFrequency,
                                                NULL, NULL, &hFrequency);
// Messwerte lesen
ulSize = 2;
Error = max_read_channel_block(hFrequency, &ulSize, (void*)aucData);
```

10.8.2.18. Periodendauermessung

Das Signal, dessen Periodendauer bestimmt werden soll, wird an Zähleingang B gelegt. Die Referenz-Frequenz, die während der Periodendauer des Messsignals gezählt wird, kann intern generiert werden oder über Zähleingang A zugeführt werden.

Um eine Periodendauermessung durchführen zu können, muss nachfolgende CPS verwendet werden.

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE _PERIOD_COUNTER*	Kanal auf einen Zähler für Periodendauermessung
.usIndexFirst	*0 ... 2*	Nummer des ersten Zählers
.usIndexLast	*0 ... 2*	Nummer des letzten Zählers (Ist dieser Wert größer als *.usIndexFirst*, werden die Zähler von *.usIndexFirst* bis *.usIndexLast* kaskadiert.)
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
	_XC163_NEW_DATA_ONLY	Ist dieses Flag gesetzt, liefert der Lesedienst den Fehler ERR_DATA_NOT_READY zurück falls noch keine neuen Daten bereitstehen. Andernfalls wird das alte Ergebnis zurückgeliefert.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_LATCH*	Der Zählerwert wird aus Zwischenspeicher gelesen
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
	IO_MODE_LATCH	Der Zählerwert wird in Zwischenspeicher geschrieben
.usMode	*0*	Keine Bedeutung
	_XC163_AUTORUN	Der Zählvorgang startet sofort nach dem Öffnen des Kanals (Zähler wird mit 0 vorinitialisiert).
	_XC163_CONTINUOUS	Am Ende der Messung wird sofort eine neue Messung gestartet. Wurde der Zähler mit einem Schreibzugriff auf einen bestimmten Wert voreingestellt, wird dieser Wert jeweils übernommen.

Strukturelement	Werte	Bedeutung
	_XC163_OVERRIDE	Im Zwischenspeicher liegt jeweils das Ergebnis der letzten Messung. Ist dieser Mode nicht aktiviert, wird nur ein neuer Wert in den Zwischenspeicher geschrieben, wenn der alte Wert ausgelesen wurde.
.ulReferenceTime	*XC163_EXTERNAL* *XC163_20MHZ* *XC163_10MHZ* *XC163_5MHZ* *XC163_1MHZ* *XC163_500KHZ* *XC163_200KHZ* *XC163_100KHZ* *XC163_50KHZ* *XC163_20KHZ* *XC163_10KHZ* *XC163_5KHZ* *XC163_2KHZ* *XC163_1KHZ* *XC163_500HZ* *XC163_200HZ* *XC163_100HZ* *XC163_50HZ* *XC163_20HZ* *XC163_10HZ*	Referenz-Frequenz
.usMultiple	*0*	Reserviert
.usCounterInput	*_XC163_DIN_x_y* *_XC163_NEG_EDGE_A* *_XC163_NEG_EDGE_B*	Zähleingang (siehe Tabelle)
.ulControlInput	*0* *_XC163_START_CTRL_x* *_XC163_STOP_CTRL_x* *_XC163_LOAD_CTRL_x* *_XC163_LATCH_CTRL_x* *_XC163_RESET_CTRL_x*	Keine Bedeutung Start, Stop, Laden, Zwischenspeichern und Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)
ulCallbackEvent	*0* *XC163_END_OF_ MEASUREMENT* *XC163_EVENT_OVERFLOW*	Es wird kein Ereignis signalisiert Eine Messung ist beendet worden Der Zähler ist übergelaufen

Strukturelement	Werte	Bedeutung
	XC163_EVENT_START	Der Zähler wurde durch einem Steuerkanal gestartet
	XC163_EVENT_STOP	Der Zähler wurde durch einem Steuerkanal gestoppt
	XC163_EVENT_LOAD	Der Zähler wurde durch einem Steuerkanal geladen
	XC163_EVENT_LATCH	Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert
	XC163_EVENT_RESET	Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße hängt dabei von der Anzahl der Zähler ab. Je Zähler werden 2 Byte benötigt.

Anmerkungen

1. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist *.usMode = _XC163_AUTORUN*, startet der Zähler nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zählvorgang nach dem Beschreiben des Kanals. Der Zähler wird dabei mit dem geschriebenem Wert vor-initialisiert.
- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibbefehl der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

2. Lesen der Zählerwerte

Gelesen wird immer aus einem internen Zwischenspeicher.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y***
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11
1	x/y = 2/3, 8/9
2	x/y = 4/5, 10/11

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

Beispiel:

Nachfolgend wird ein Kanal auf eine Periodendauermessung mit einer Zählfrequenz von 1 kHz geöffnet.

```
CPS_XC163  rcPeriod;
MAXCHLHND hPeriod;
MAX_ERROR Error;
ULONG ulSize;
UCHAR aucData[2];

// CPS ausfuellen
rcPeriod.usDevice = DEVICE_PERIOD_COUNTER;
rcPeriod.usIndexFirst = 0;
rcPeriod.usIndexLast = 0;
rcPeriod.usFlags = _CP_EXCLUSIVE;
rcPeriod.usReadMode = IO_MODE_LATCH;
rcPeriod.usWriteMode = IO_MODE_DIRECT;
rcPeriod.usMode = _XC163_AUTORUN | _XC163_OVERRIDE | _XC163_CONTINUOUS;
rcPeriod.ulReferenceTime = XC163_1KHZ;
rcPeriod.usMultiple = 0;
rcPeriod.usCounterInput = _XC163_DIN_0_1;
rcPeriod.ulControlInput = 0;

// Kanal oeffnen
Error = max_open_channel(hMdd, sizeof (rcPeriod), &rcPeriod,
                                                  NULL, NULL, &hPeriod);

// Messwerte lesen
ulSize = 2;
Error = max_read_channel_block(hPeriod, &ulSize, (void*)aucData);
```

10.8.2.19. Periodendauermessung über mehrere Perioden

Das Signal, dessen Periodendauer bestimmt werden soll, wird an den Zähleingang B gelegt. Die Referenz-Frequenz, die während der Periodendauer des Messsignals gezählt wird, kann intern generiert werden oder über den Zähleingang A zugeführt werden.

Um eine Periodendauermessung über mehrere Perioden durchführen zu können, muss nachfolgende CPS verwendet werden.

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_PERIOD_COUNTER*	Kanal auf einen Zähler für Periodenmessung über mehrere Perioden
.usIndexFirst	*0*	Nummer des ersten Zählers
.usIndexLast	*1*	Nummer des letzten Zählers
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
	_XC163_NEW_DATA_ONLY	Ist dieses Flag gesetzt, liefert der Lesedienst den Fehler ERR_DATA_NOT_READY zurück falls noch keine neuen Daten bereitstehen. Andernfalls wird das alte Ergebnis zurückgeliefert.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_LATCH*	Der Zählerwert wird aus Zwischenspeicher gelesen.
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
	IO_MODE_LATCH	Der Zählerwert wird in Zwischenspeicher geschrieben.
.usMode	*_XC163_AUTORUN*	Der Zählvorgang startet sofort nach dem Öffnen des Kanals (Zähler wird mit 0 vorinitialisiert).
	_XC163_OVERRIDE	Im Zwischenspeicher liegt jeweils das Ergebnis der letzten Messung. Ist dieser Mode nicht aktiviert, wird nur ein neuer Wert in den Zwischenspeicher geschrieben, wenn der alte Wert ausgelesen wurde.
	_XC163_MULTIPLE	Dieses Flag muss gesetzt sein.

Strukturelement	Werte	Bedeutung
.ulReferenceTime	*XC163_EXTERNAL* *XC163_20MHZ* *XC163_10MHZ* *XC163_5MHZ* *XC163_1MHZ* *XC163_500KHZ* *XC163_200KHZ* *XC163_100KHZ* *XC163_50KHZ* *XC163_20KHZ* *XC163_10KHZ* *XC163_5KHZ* *XC163_2KHZ* *XC163_1KHZ* *XC163_500HZ* *XC163_200HZ* *XC163_100HZ* *XC163_50HZ* *XC163_20HZ* *XC163_10HZ*	Referenz-Frequenz
.usMultiple	*1 ... 65535*	Anzahl der Perioden, die gemessen werden sollen
.usCounterInput	*_XC163_DIN_x_y* *_XC163_NEG_EDGE_A* *_XC163_NEG_EDGE_B*	Zähleingang (siehe Tabelle) Zählen bei negativer Flanke an Eingang A Zählen bei negativer Flanke an Eingang B
.ulControlInput	*0* *_XC163_START_CTRL_x* *_XC163_STOP_CTRL_x* *_XC163_LOAD_CTRL_x* *_XC163_LATCH_CTRL_x* *_XC163_RESET_CTRL_x*	Keine Bedeutung Start, Stop, Laden, Zwischenspeichern und Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)
ulCallbackEvent	*0* *XC163_END_OF_ MEASUREMENT* *XC163_EVENT_OVERFLOW* *XC163_EVENT_START* *XC163_EVENT_STOP* *XC163_EVENT_LOAD* *XC163_EVENT_LATCH*	Es wird kein Ereignis signalisiert Eine Messung ist beendet worden Der Zähler ist übergelaufen Der Zähler wurde durch einem Steuerkanal gestartet Der Zähler wurde durch einem Steuerkanal gestoppt Der Zähler wurde durch einem Steuerkanal geladen Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert

Strukturelement	Werte	Bedeutung
	XC163_EVENT_RESET	Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße umfaßt 2 Byte.

Anmerkung

1. Periodendauermessung über mehrere Perioden

Ist der Mode *_XC163_MULTIPLE* aktiviert, wird eine Periodendauermessung über mehrere Perioden durchgeführt. Hierzu muss *usIndexFirst = 0* und *usIndexLast = 1* eingestellt sein. Im Element *usMultiple* kann dann die Anzahl der Perioden eingetragen werden, die gemessen werden sollen. Die Steuerkommandos zum Starten, Stoppen, Laden, Latchen und Zurücksetzen müssen sowohl an Zähler 0 als auch an Zähler 1 gesendet werden!

2. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist *.usMode = _XC163_AUTORUN*, startet der Zähler nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zählvorgang nach dem Beschreiben des Kanals. Der Zähler wird dabei mit dem geschriebenem Wert vor-initialisiert.
- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibbefehl der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

3. Lesen der Zählerwerte

Gelesen wird immer aus einem internen Zwischenspeicher.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y***
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11
1	x/y = 2/3, 8/9
2	x/y = 4/5, 10/11

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

Beispiel:

Nachfolgend wird ein Kanal auf eine Periodendauermessung über 2 Perioden mit einer Zählfrequenz von 1 kHz geöffnet.

```
CPS_XC163  rcPeriod;
MAXCHLHND hPeriod;
MAX_ERROR Error;
ULONG ulSize;
UCHAR aucData[2];

// CPS ausfuellen
rcPeriod.usDevice = DEVICE_PERIOD_COUNTER;
rcPeriod.usIndexFirst = 0;
rcPeriod.usIndexLast = 1;
rcPeriod.usFlags = _CP_EXCLUSIVE;
rcPeriod.usReadMode = IO_MODE_LATCH;
rcPeriod.usWriteMode = IO_MODE_DIRECT;
rcPeriod.usMode = _XC163_AUTORUN | _XC163_OVERRIDE | _XC163_MULTIPLE;
rcPeriod.ulReferenceTime = XC163_1KHZ;
rcPeriod.usMultiple = 2;
rcPeriod.usCounterInput = _XC163_DIN_0_1;
rcPeriod.ulControlInput = 0;

// Kanal oeffnen
Error = max_open_channel(hMdd, sizeof (rcPeriod), &rcPeriod,
                                                NULL, NULL, &hPeriod);

// Messwerte lesen
ulSize = 2;
Error = max_read_channel_block(hPeriod, &ulSize, (void*)aucData);
```

10.8.2.20. Inkrementalgeber

Mit diesem Device kann eine Inkrementalgebererfassung realisiert werden. Es stehen zwei unterschiedliche Betriebsarten (Mode A und Mode B) zur Verfügung:

Um auf einen Inkrementalgeber-Kanal zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_INCREMENTAL _COUNTER*	Kanal auf einen Zähler für Inkrementalgeber
.usIndexFirst	*0 ... 2* (Mode B: 0)	Nummer des ersten Zählers
.usIndexLast	*0 ... 2* (Mode B: 1)	Nummer des letzten Zählers
.usFlags	*_CP_EXCLUSIVE* *_CP_SYNC_CALLBACK*	Der Zugriff erfolgt immer exklusiv Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_DIRECT* *IO_MODE_LATCH*	Direkter Lesezugriff Der Zählerwert wird aus Zwischenspeicher gelesen
.usWriteMode	*IO_MODE_DIRECT* *IO_MODE_LATCH*	Direkter Schreibzugriff Der Zählerwert wird in Zwischenspeicher geschrieben.

Strukturelement	Werte	Bedeutung
.usMode	*_XC163_AUTORUN*	Der Zählvorgang startet sofort nach dem Öffnen des Kanals (Zähler wird mit 0 vorinitialisiert).
	----------	Mode Flags: Es muss genau ein Flag gesetzt sein.
	_XC163_MODE_A	In dieser Betriebsart arbeitet der Zähler als Auf-/Abwärtszähler. Die Zählimpulse werden von beiden Phasen des Inkrementalgebers abgeleitet. Phase A des Inkrementalgebers wird an den Zähleingang A des Zählers und Phase B des Inkrementalgebers an Phase B des Zählers angeschlossen.
	_XC163_MODE_B	In dieser Betriebsart werden zwei Zähler des Moduls verwendet (0 und 1). Zähler 0 zählt die Impulse in Vorwärtsrichtung, Zähler 1 die Impulse in Rückwärtsrichtung. Die Position ergibt sich aus der Differenz beider Zählerstände. Die Phasen A und B werden an den Zähleingang A und B der Zähler angeschlossen (es muss ein Zähleingang verwendet werden, der an beiden Zählern verfügbar ist).
	----------	Flanken Flags: Es muss genau ein Flag gesetzt sein.
	_XC163_SINGLE_A	Zähler zählt pos. bzw. neg. Flanke an Eingang A
	_XC163_SINGLE_B	Zähler zählt pos. bzw. neg. Flanke an Eingang B
	_XC163_DOUBLE_A	Zähler zählt pos. und neg. Flanke an Eingang A
	_XC163_DOUBLE_B	Zähler zählt pos. und neg. Flanke an Eingang B
	_XC163_QUADRUPLE	Zähler zählt pos. und neg. Flanke an Eingang A und B
.usCounterInput	*_XC163_DIN_x_y*	Zähleingang (siehe Tabelle)
	_XC163_NEG_EDGE_A	Zählen bei negativer Flanke an Eingang A
	_XC163_NEG_EDGE_B	Zählen bei negativer Flanke an Eingang B

Strukturelement	Werte	Bedeutung
.ulControlInput	*0*	Keine Bedeutung
	_XC163_START_CTRL_x	Start,
	_XC163_STOP_CTRL_x	Stop,
	_XC163_LOAD_CTRL_x	Laden,
	_XC163_LATCH_CTRL_x	Zwischenspeichern und
	_XC163_RESET_CTRL_x	Reset des Zählers erfolgt jeweils durch Steuerkanal x (x = 0 ... 3)
ulCallbackEvent	*0*	Es wird kein Ereignis signalisiert
	XC163_EVENT_ OVERFLOW	Der Zähler ist übergelaufen
	XC163_EVENT_ UNDERFLOW	Der Zähler ist untergelaufen
	XC163_EVENT_START	Der Zähler wurde durch einem Steuerkanal gestartet
	XC163_EVENT_STOP	Der Zähler wurde durch einem Steuerkanal gestoppt
	XC163_EVENT_LOAD	Der Zähler wurde durch einem Steuerkanal geladen
	XC163_EVENT_LATCH	Der Zählerstand wurde durch einem Steuerkanal zwischengespeichert
	XC163_EVENT_RESET	Der Zähler wurde durch einem Steuerkanal zurückgesetzt

Eingabe- und Ausgabedienst

Der Zugriff auf den Kanal erfolgt mit:

- **max_write_channel_block**
- **max_read_channel_block**

Die Blockgröße hängt dabei von der Anzahl der Zähler ab. Je Zähler werden 2 Byte benötigt.

Anmerkungen

1. Betriebsart

Ist Mode B aktiviert, muss *.usIndexFirst = 0* und *.usIndexLast = 1* sein.

2. Starten des Zählers

Das Starten des Zählers kann unterschiedlich erfolgen:

- Ist *.usMode = _XC163_AUTORUN*, startet der Zähler nach dem Öffnen des Kanals sofort (mit dem Anfangswert 0).
- Ist *.usWriteMode = IO_MODE_DIRECT*, startet der Zählvorgang nach dem Beschreiben des Kanals. Der Zähler wird dabei mit dem geschriebenem Wert vor-initialisiert.

- Ist *.usWriteMode = IO_MODE_LATCH*, wird mit einem Schreibbefehl der Zählerwert in einen Zwischenspeicher geschrieben. Zum Aktivieren des geschriebenen Wertes muss noch ein Lade- und evtl. Startbefehl (falls der Zähler noch nicht läuft) gesendet werden. Der Lade- und Startbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.usControlInput* mit der entsprechenden Konstanten belegt werden.

3. Lesen der Zählerwerte

Ist *usReadMode = IO_MODE_DIRECT*, kann der aktuelle Zählerstand durch einen Lesebefehl ermittelt werden. Wurde der Lesemodus auf *IO_MODE_LATCH* eingestellt, wird durch einen Lesezugriff der im Latch zwischengespeicherte Zählerstand ausgelesen. Ein Latchbefehl kann entweder per Software erfolgen und/oder über eine externe Steuerleitung. Wenn eine externe Steuerleitung verwendet werden soll, muss das Strukturelement *.ulControlInput* mit der entsprechenden Konstanten belegt werden.

Tabelle Zähleingang *_XC163_DIN_x_y:*

.usIndexFirst	***_XC163_DIN_x_y* für Mode A**	***_XC163_DIN_x_y* für Mode B**
0	x/y = 0/1, 2/3, 4/5, 6/7, 8/9, 10/11	x/y = 2/3, 8/9
1	x/y = 2/3, 8/9	
2	x/y = 4/5, 10/11	

Callback-Funktion

Über das Strukturelement *ulCallbackEvent* kann festgelegt werden, über welches Ereignis man informiert werden möchte. Die Anwenderfunktion bekommt 4 Byte Nutzdaten übergeben, die angeben, welche Ereignisse aufgetreten sind. Dazu werden die entsprechenden Konstanten, die in *ulCallbackEvent* anzugeben sind, zurückgegeben.

10.8.2.21. Pulsbreitenmodulation

Mit Hilfe dieses Kanals kann ein Pulsbreitenmoduliertes Signal mit den digitalenAusgängen DOUT-1 bzw. DOUT-2 erzeugt werden.

Strukturelement	**Werte**	**Bedeutung**
.usDevice	*DEVICE_PWM*	Kanal auf für pulsbreitenmoduliertes Signal
.usIndexFirst	*1, 2*	DOUT auf dem das Signal ausgegeben werden soll
.usIndexLast	*.usIndexFirst*	

Strukturelement	Werte	Bedeutung
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
.usMode	*0*	reservierter Parameter
.usWriteMode	*0*	reservierter Parameter
.usReadMode	*0*	reservierter Parameter
.usControlInput	*0*	reservierter Parameter

Der Kanal belegt jeweils den Zähler 0 und 1 (für usIndexFirst = 0)
oder den Zähler 0 und 2 (für usIndexFirst = 1).

Eingabe- und Ausgabedienst

Der Kanal verfügt über keinen Ein- bzw. Ausgabedienst

Sonderdienste

Mit Hilfe von Sonderdiensten kann die Periodendauer und das Puls-/Pausenverhältnis verändert werden.

- **max_channel_control,** Steuerbefehl XC163_SET_PERIOD_TIME: dieser Befehl ändert die Periodendauer des Signals.

Dem Sonderdienst muss die Länge der Periodendauer in Form der folgenden Struktur übergeben werden:

```
Struct
{
    USHORT usSec;    // 0 ... 6500
    USHORT usMSec;   // 0 ... 999
    USHORT usUSec;   // 0 ... 999
    USHORT usNSec;   // 50 ... 999
} XC163_TIME;
```

- **max_channel_control,** Steuerbefehl XC163_SET_PULSE_PAUSE_RATIO: mit diesem Befehl kann das Puls-/Pausenverhältnis in % geändert werden. Dem Dienst wird ein USHORT-Wert mit dem Prozentwert übergeben.

- **max_channel_control,** Steuerbefehl CMD_START: mit diesem Befehl kann die Ausgabe des Signals gestartet werden (nachdem zuvor die Periodendauer und das Puls-/Pausenverhältnis eingestellt wurden).

10.8.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Eingang	+Eingang (= Anode)	-Eingang (= Kathode)	+Ausgang (= Kollektor)	-Ausgang (= Emitter)
DIN-0	1	2	-	-
DIN-1	3	4	-	-
DIN-2	5	6	-	-
DIN-3	7	8	-	-
DIN-4	9	10	-	-
DIN-5	11	12	-	-
DOUT-0	-	-	13	14
DOUT-1	-	-	15	16
DOUT-2	-	-	17	18
DOUT-3	-	-	19	20
DIN-6	21	22	-	-
DIN-7	23	24	-	-
DIN-8	25	26	-	-
DIN-9	27	28	-	-
DIN-10	29	30	-	-
DIN-11	31	32	-	-
DOUT-4	-	-	33	34
DOUT-5	-	-	35	36
DOUT-6	-	-	37	38
DOUT-7	-	-	39	40

10.8.4. Besondere Eigenschaften

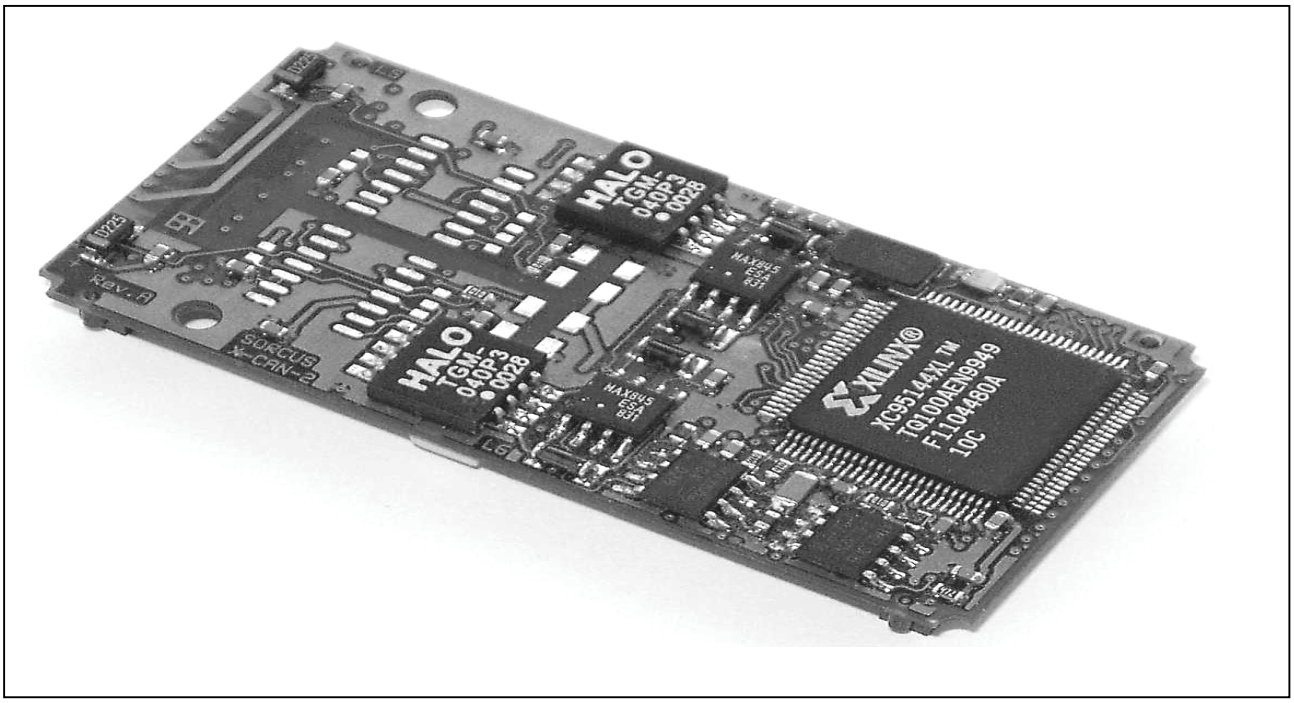
(Angaben gelten für Version X-C16-3i/L und X-C16-3i/P)

Parameter	**Wert**	**Einheit**
Zählerkanäle, kaskadierbar,	3	-
per Software verlängerbar bis 256 Bit		
Auflösung	16	Bit
Eingangsfrequenz, max.	10	MHz
Zählfrequenz, max.	20	MHz
Externe Eingänge (X-C16-3i/L), Anzahl	12	-
Eingangsspannung, max.	±18	V
Schwelle max., Erkennung einer log. 0	2,2	V
Schwelle min., Erkennung einer log. 1	4	V
Schwelle Eingangsstrom, typ.	2	mA
Externe Eingänge (X-C16-3i/P), Anzahl	12	-
Eingangsspannung, max.	±30	V
Schwelle max., Erkennung einer log. 0	5	V
Schwelle min., Erkennung einer log. 1	13	V
Schwelle Eingangsstrom, typ.	2	mA
Externe Ausgänge, Anzahl	8	-
Ausgangsspannung, max.	80	V
Ausgangsspannung, max. bei log. 0 (Iout = 10 mA)	< 0,4	V
Ausgangsleistung, max. je Kanal	150	mW
Temperatur-Bereich, Betrieb	-40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	10,6	g
Stromaufnahme (min./max.)		
3,3V	310/405	mA
(die X-Bus Spannungen ±12V werden nicht benötigt)		

Notizen

X-CAN-2i

2 unabhängige, isolierte CAN-Bus-Schnittstellen



10

10.9. X-CAN-2i/H, X-CAN-2i/M, X-CAN-2i/F

Inhaltsverzeichnis

10.9.1. Beschreibung

Das Modul X-CAN-2i bietet zwei unabhängige CAN-Bus-Schnittstellen nach CAN-Spezifikation 2.0 A und 2.0 B (11- und 29-Bit Identifier). Beide Schnittstellen sind sowohl gegenüber der X-Bus-Seite als auch gegeneinander galvanisch getrennt.

Je nach Bestückungsvariante ist die physikalische Busankopplung der beiden Schnittstellen wie folgt ausgeführt:

- zwei High-Speed-Schnittstellen (X-CAN-2i/H = Typ 58, Subtyp 1) nach ISO-DIS 11898 mit Philips 82C250 Transceiver
- zwei fehlertolerante Low-Speed-Schnittstellen (X-CAN-2i/F = Typ 58, Subtyp 2) nach ISO-DIS 11519-1 mit Philips TJA-1054 Transceiver
- je eine High-Speed und eine Low-Speed-Schnittstelle (X-CAN-2i/M = Typ 58, Subtyp 3)

Die maximale Übertragungsgeschwindigkeit beträgt 1 MBit/s bzw. bei der Low-Speed-Schnittstelle 125 kBit/s.

High- und Low-Speed Schnittstellen können nicht im selben Netzwerk betrieben werden!

Die CAN-Bus-Kommunikation wird durch zwei CAN-Bus-Controller INTEL 82527 abgewickelt. Diese sorgen auch für Fehlererkennung und –begrenzung (Bit-Monitoring, Frame- und CRC-Check, Acknowledgement-Überwachung, Überwachung der Bit-Stuffing-Codierungsregel, wiederholtes Senden im Fehlerfall). Durch ihre Full-CAN-Funktionalität reagieren die CAN-Controller nur auf die Bus-Nachrichten, die für die jeweilige Anwendung von Interesse sind.

Bei High-Speed-Schnittstellen kann der Busabschlusswiderstand per Software zugeschaltet werden. Bei Low-Speed-Schnittstellen können die Busabschluss-

Widerstände zwischen den Werten 5,6 kΩ und 510 Ω per Software umgeschaltet werden.

Das Modul ist Interrupt-fähig. Ein Interrupt wird durch eines der folgenden Ereignisse in einer der beiden Schnittstellen ausgelöst:

- Ein CAN-Telegramm ist erfolgreich versendet worden. Das Telegramm muss dazu von mindestens einem anderen Busteilnehmer richtig empfangen worden sein und darf von keinem Busteilnehmer als fehlerhaft erkannt worden sein.
- Ein CAN-Telegramm ist fehlerfrei empfangen worden.
- Ein CAN-Controller erkennt einen stark gestörten Bus bzw. schaltet sich aufgrund gehäufter Übertragungsfehler von der CAN-Bus-Kommunikation ab.
- Der fehlertolerante CAN-Bus-Transceiver erkennt, dass eine der beiden CAN-Leitungen unterbrochen oder mit Masse bzw. der Versorgungsspannung kurzgeschlossen ist oder beide CAN-Leitungen kurzgeschlossen sind (nur für die Low-Speed-Schnittstelle).

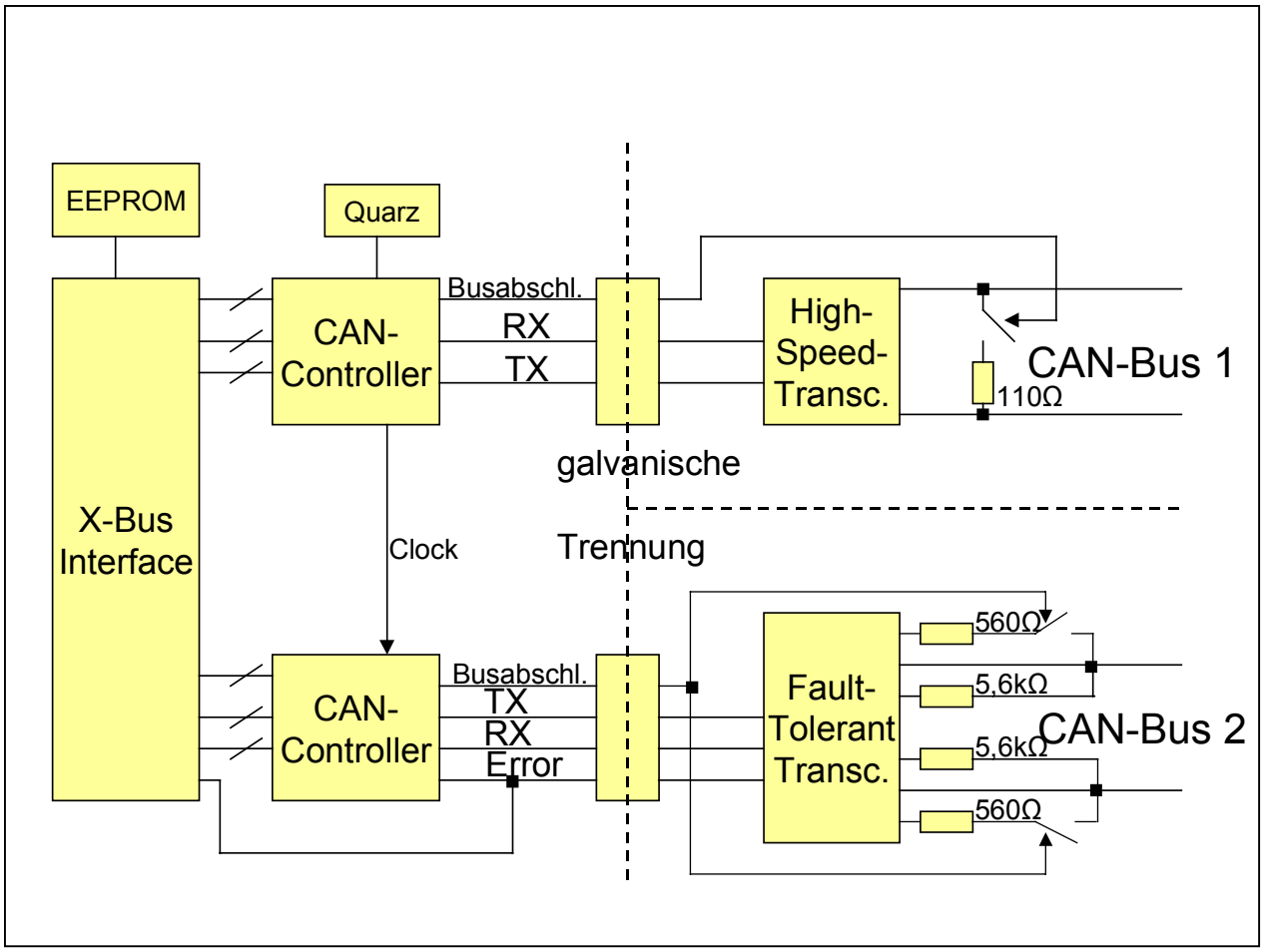
Der Transceiver schaltet bei Erkennen eines Leitungsfehlers automatisch auf eine Ein-Draht-Übertragung (gegen Masse) um.

Bei Abnahme größerer Stückzahlen sind die folgenden weiteren Bestückungsvarianten möglich (für eine oder beide Schnittstellen):

- Bereitstellung der 12 Volt Versorgungsspannung des X-Bus. Diese ist nicht galvanisch getrennt und wird vom Modul selbst nicht verwendet. Die galvanische Trennung wird dadurch aufgehoben.
- Verbindung der Abschirmung des Bus-Kabels (Shield) mit dem galvanisch getrennten Ground.
- Externe Versorgung (7 – 24 Volt) des galvanisch getrennten Teils der Schnittstelle
- Bestückung nur einer Schnittstelle

10.9.1.1. Blockschaltbild

Das Blockschaltbild zeigt den Aufbau des X-CAN-2i/M.



10.9.1.2. Der CAN-Bus

Ein CAN-Bus-Netzwerk ist ein Bussystem, bei dem mehrere Teilnehmer den Bus gleichzeitig anfordern können. Ein Teilnehmer sendet eine Nachricht, alle anderen Teilnehmer hören diese Nachricht mit. Mit einer Nachricht können sowohl Daten an andere Teilnehmer verschickt werden als auch Daten eines anderen Teilnehmers angefordert werden.

Die Daten werden hierbei immer zusammen mit einem CAN-Identifier der im folgenden als ID bezeichnet wird, auf dem CAN-Bus verschickt. Die Länge der Daten ist auf maximal 8 Byte pro Telegramm begrenzt. Die ID wird von anderen Teilnehmern als Kriterium verwendet, ob die Nachricht für sie von Interesse ist oder nicht. Weiterhin legt der Wert der ID die Priorität der Nachricht auf dem CAN-Bus fest. Wenn mehrere Nachrichten gleichzeitig gesendet werden, setzt sich im zeitlichen

Verlauf die Nachricht mit der kleinsten ID gegenüber den anderen Nachrichten durch.

10.9.1.3. Message-Objekte

Nutzdaten und ID werden zu einem Message-Objekt zusammengefasst und vom Modul-Device-Treiber verwaltet. Die eindeutige Kennzeichnung eines Message-Objektes erfolgt durch den Identifier.

Bei der Vergabe der ID an ein Message-Objekt muss folgendes beachtet werden:

- Die ID muss für einen Busteilnehmer eindeutig sein, d.h. er darf sie höchstens einem Nutzdatensatz zuordnen.
- Eine ID darf nur einem Teilnehmer als aktiver Sender zugeordnet sein, d.h. nur dieser eine Teilnehmer darf die der ID zugeordneten Nutzdaten auf dem Bus versenden. Es können mehrere Busteilnehmer als Empfänger der Daten auftreten.
- Für jede der beiden CAN-Schnittstellen muss festgelegt werden, ob sie mit 11- oder 29-Bit-IDs benutzt werden soll.
- Nicht alle auf dem Markt erhältlichen CAN-Controller unterstützen 29-Bit-IDs! Befinden sich solche Busteilnehmer im CAN-Netzwerk, wird es bei der Verwendung von 29-Bit-IDs zu Fehlern kommen. Verwenden Sie in diesem Fall 11-Bit-IDs.

Message-Objekte werden in zwei Typen unterteilt:

- Sendeobjekte zum Verschicken von Daten
- Empfangsobjekte zum Empfangen von Daten

Ein ***Sendeobjekt*** kann *aktiv* oder *passiv* sein:

- Bei einem aktiven Sendeobjekt geben Sie den Zeitpunkt, wann das Message-Objekt Daten sendet, in Ihrem Anwendungsprogramm durch einen Schreibzugriff auf einen zu dem Sendeobjekt geöffneten Kanal selbst vor. In diesem Falle wird ein *Data-Frame* auf dem CAN-Bus verschickt.
- Bei einem passiven Sendeobjekt wird der Zeitpunkt, wann das Message-Objekt Daten sendet, von einem anderen Busteilnehmer, der von Ihrem Anwendungsprogramm Daten per *Remote-Frame* anfordert, bestimmt. Es werden *automatisch* die zu diesem Zeitpunkt gültigen Daten des passiven Sendeobjekts durch Verschicken eines *Data-Frames* auf den Bus gesendet.

Ein ***Empfangsobjekt*** kann *aktiv* oder *passiv* sein:

- Bei einem aktiven Empfangsobjekt geben Sie den Zeitpunkt, wann das Message-Objekt Daten empfängt, in Ihrem Anwendungsprogramm durch einen Lesezugriff auf einen zu dem Empfangsobjekt geöffneten Kanal selbst vor. In diesem Falle werden Daten durch Senden eines *Remote-Frames* von einem anderen Busteilnehmer angefordert.
- Ein passives Empfangsobjekt empfängt Daten, ohne dass Sie diese in ihrem Anwendungsprogramm speziell anfordern. Die Daten stammen von einem anderen Busteilnehmer, der einen *Data-Frame* verschickt hat.

Die Anzahl der Message-Objekte ist dynamisch. Im Initialisierungszustand sind keine Message-Objekte konfiguriert. Die Konfiguration eines Message-Objekts erfolgt beim Öffnen des ersten Kanals mit einem bestimmten Identifier-Wert.

Werden weitere Kanäle zum selben Message-Objekt geöffnet (d.h. in der CPS ist der selbe Wert im Parameter *ulCanID* angegeben), so wird kein neues Message-Objekt angelegt. Es wird lediglich ein neuer Kanal zum bereits konfigurierten Message-Objekt geöffnet. Dabei ist zu beachten, dass die Eigenschaften eines Message-Objekts bei seiner Konfiguration festgelegt werden. Dieses hat zur Folge, dass sich die CPS-Parameter beim Öffnen eines Kanals zu einem bestehenden Message-Objekt nicht von denen bei seiner Konfiguration unterscheiden dürfen.

10.9.1.4. Verwaltung der Message-Puffer eines CAN-Controllers

Die folgende Erläuterung ist nur dann von Interesse, wenn über eine der beiden CAN-Schnittstellen mehr als 14 verschiedene Message-Objekte kommunizieren sollen.

Damit eine Nachricht gesendet oder empfangen werden kann, muss das zugehörige Message-Objekt in einem der Message-Puffer eines CAN-Controllers auf dem Modul eingetragen sein. Jeder CAN-Controller verfügt hierzu über 14 Message-Puffer (Puffer 1 bis 14), die zum Senden und Empfangen genutzt werden können, und einen speziellen Message-Puffer (Puffer 15), der ausschließlich zum Empfangen genutzt werden kann.

Das Eintragen eines Message-Objekts in einen Message-Puffer nimmt eine gewisse Zeit (einige µs) in Anspruch. Wenn ein Message-Objekt zu dem Zeitpunkt, an dem es z.B. einen Sendewunsch vom Anwenderprogramm erhält, bereits in einem Message-Puffer eingetragen ist, entfällt diese Zeit – es wird schneller reagiert.

Der Modul-Device-Treiber ist darauf ausgerichtet, die Puffer möglichst effektiv nutzen zu können. Hierbei hat das richtige Verständnis zur Pufferverwaltung entscheidenden Einfluss auf die Effektivität Ihres Anwendungsprogramms.

Wenn Sie für eine CAN-Schnittstelle des X-CAN-2i maximal 14 Message-Objekte konfigurieren, dann wird im entsprechenden CAN-Controller jedes Message-Objekt dauerhaft in einen dafür **exklusiv** reservierten **Puffer** (Nr. 1 bis 14) eingetragen. Dieser Fall ist die Optimal-Konfiguration: Weder muss zum aktiven Senden oder Anfordern von Daten die ID in einen Puffer eingetragen werden, noch muss zum Empfangen einer Nachricht die zugehörige ID aus einem Puffer gelesen und ausgewertet werden. Weiterhin wird das zugehörige CPU-Modul minimal belastet, da ausschließlich die gewünschten (d.h. die konfigurierten) Nachrichten empfangen werden.

Mehr als 14 Message-Objekte

Benötigt eine CAN-Schnittstelle des X-CAN-2i mehr als 14 Message-Objekte, so können zumindest einige davon zwangsläufig nur temporär in einen Puffer des CAN-Controllers eingetragen werden.

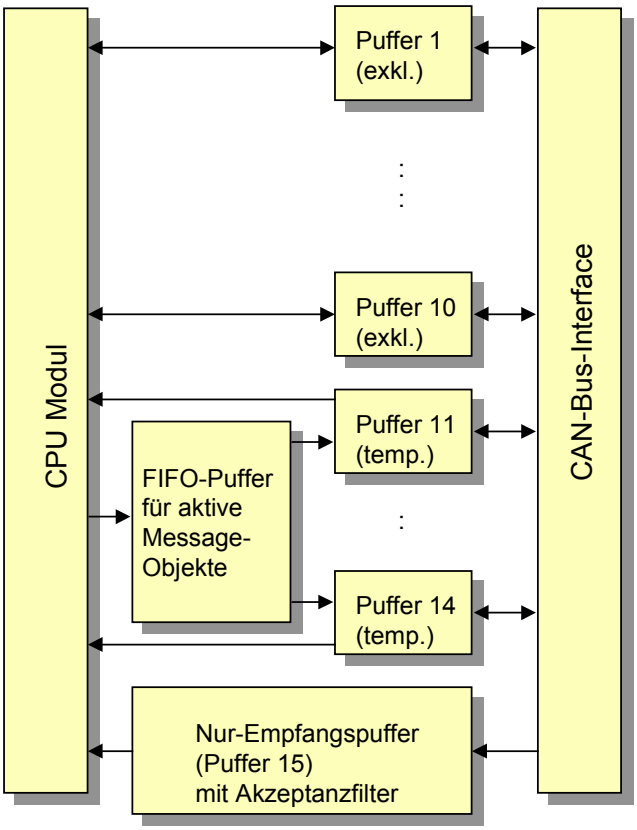
Wenn Sie beim Konfigurieren des Message-Objekts das Flag *XCAN_EXCL_ BUFFER* setzen, so zwingen Sie die Pufferverwaltung dazu, einen der Puffer 1 bis 13 abzugeben und ihn exklusiv für dieses Message-Objekt zu reservieren. In diesem Falle wird das Message-Objekt bereits zum Zeitpunkt der Konfiguration dauerhaft in einen der freien Puffer 1 bis 13 eingetragen und der dafür genutzte Puffer exklusiv für dieses Message-Objekt reserviert. Bei einem **exklusiven Puffer** muss weder zum aktiven Senden oder Anfordern von Daten die ID in den Puffer eingetragen werden, noch muss zum Empfangen einer Nachricht die zugehörige ID aus dem Puffer gelesen und ausgewertet werden.

Die von den Puffern 1 bis 13 nicht exklusiv reservierten Puffer sowie Puffer 14 werden von der Pufferverwaltung als **temporäre Puffer** verwendet. Diese Puffer stehen für aktive Message-Objekte zur Verfügung. Ein aktives Empfangsobjekt sendet über einen solchen Puffer einen Remote-Frame, ein aktives Sendeobjekt sendet über einen solchen Puffer einen Data-Frame. Der Modul-Device-Treiber trägt hierzu das Message-Objekt - falls es nicht bereits in einem temporären Puffer gespeichert ist - in einen dieser temporären Puffer ein, sobald es gesendet werden soll. Der so belegte temporäre Puffer ist danach solange nicht verfügbar, bis das gesendete Objekt die Empfangsbestätigung (signalisiert durch Interrupt) erhält. Danach bleibt das Message-Objekt in diesem Puffer solange gespeichert, bis es durch ein anderes Message-Objekt überschrieben wird.

Wenn gerade kein temporärer Puffer verfügbar ist, so wird die Nachricht in einen **FIFO-Puffer** des MDD eingetragen. Wenn einer der temporären Puffer frei wird, so rückt eine Nachricht aus dem FIFO-Puffer nach und wird automatisch gesendet.

Die Abbildung rechts zeigt, wie die Pufferverwaltung des MDD

die Message-Puffer nutzt, wenn mehr als 14 Message-Objekte konfiguriert sind und davon 10 Message-Objekte die Eigenschaft *XCAN_EXCL_BUFFER* haben:



Nur-Empfangspuffer

Alle **passiven** Message-Objekte, die nicht in einen exklusiven Puffer eingetragen werden, werden zum Zeitpunkt der Konfiguration in den **Nur-Empfangspuffer** (Puffer 15) eingetragen. Hierbei werden bereits eingetragene Message-Objekte nicht überschrieben. Statt dessen überlagern sich die IDs aller eingetragenen Message-Objekte und bilden ein digitales Akzeptanzfilter, das zur Entlastung des MDD nicht gewünschte Nachrichten per Hardware ausfiltert. Bei den passiven Message-Objekten ist eine durch den CAN-Controller bedingte Einschränkung zu beachten: Alle Message-Objekte, die über den Nur-Empfangspuffer empfangen werden sollen, müssen vom selben Typ sein, d.h. es können darüber entweder nur passive Sendeobjekte oder passive Empfangsobjekte empfangen werden. Die exklusiv genutzten Puffer sind hiervon nicht betroffen. Darin können sowohl passive Sende- als auch Empfangsobjekte eingetragen werden.

Das **digitale Akzeptanzfilter** besteht aus einer ID-Maske und einer Akzeptanzmaske. Beide Masken werden bitweise aus allen in den Nur-Empfangspuffer eingetragenen IDs gebildet. An den Bitpositionen, an denen alle eingetragenen IDs den gleichen Bitwert haben, ist das Bit in der Akzeptanzmaske = 1 und in der ID-Maske gleich dem Bitwert. An den übrigen Bitposition enthalten beide Masken eine 0. Logisch bedeutet dies: Die Akzeptanzmaske entsteht durch eine XNOR-Verknüpfung aller eingetragenen IDs, die ID-Maske entsteht durch eine AND-Verknüpfung aller eingetragenen IDs.

Nur passive Message-Objekte gehen in die Berechnung des Akzeptanzfilters ein! !

Eine Nachricht passiert das Akzeptanzfilter nur dann, wenn in der ID der Nachricht an allen Positionen, an denen in der Akzeptanzmaske eine 1 steht, der Bitwert gleich dem Bitwert in der ID-Maske ist.

Beispiel: In den Nur-Empfangspuffer werden die IDs 03h und 07h eingetragen.

IDs:	03h=	0	0	0	0	0	0	0	0	0	1	1
	07h=	0	0	0	0	0	0	0	0	1	1	1
Akzeptanzmaske:		1	1	1	1	1	1	1	1	0	1	1
ID-Maske:		0	0	0	0	0	0	0	0	0	1	1

Es gilt: Akzeptanzmaske = 03h XNOR 07h, ID-Maske = 03h AND 07h. Die IDs, die das Filter passieren können, lassen sich leicht aus der entstehenden ID-Maske herauslesen. Bis auf die Position, die grau unterlegt ist (d.h. das Bit in der Akzeptanzmaske ist = 0), müssen die IDs mit der ID-Maske übereinstimmen. An der grau unterlegten Position spielt der Bitwert der ID keine Rolle. Schlussfolgerung: Das Akzeptanzfilter akzeptiert in diesem Falle tatsächlich nur die IDs 03h und 07h.

Allerdings ist es nicht immer möglich, das Akzeptanzfilter so optimal einzustellen, dass wirklich nur die gewünschten Nachrichten vom CAN-Controller empfangen werden. Bei ungünstiger Vergabe der IDs kann es dazu kommen, dass die Masken des Akzeptanzfilters so ungünstig gebildet werden, dass sehr viele unerwünschte Nachrichten das Filter passieren. Die unerwünschten Nachrichten werden zwar vom MDD verworfen, belasten aber dennoch die CPU!

Beispiel: In den Nur-Empfangspuffer werden die IDs 20h und 07h eingetragen.

IDs:	20h=	0	0	0	0	0	1	0	0	0	0	0
	07h=	0	0	0	0	0	0	0	0	1	1	1
Akzeptanzmaske:		1	1	1	1	1	0	1	1	0	0	0
ID-Maske:		0	0	0	0	0	0	0	0	0	0	0

Im Vergleich zum vorigen Beispiel sind nun mehrere Positionen grau unterlegt (d.h. das Bit in der Akzeptanzmaske ist = 0). Das Filter lässt in diesem Falle die IDs 0h bis 7h und 20h bis 27h passieren. Schlussfolgerung: Das Akzeptanzfilter akzeptiert in diesem Falle außer den gewünschten IDs 20h und 07 auch weitere 14 unerwünschte IDs.

Zusammenfassung

Die Tabelle bietet nochmals eine Übersicht über die unterschiedliche Nutzung der Puffer:

Typ von Message-Objekt	Art der Puffernutzung
Beliebiges Message-Objekt XCAN_EXCL_BUFFER gesetzt	Das Message-Objekt wird zum Zeitpunkt seiner Konfiguration dauerhaft in einen der dafür exklusiv reservierten Puffer eingetragen. Es ist das Senden und Empfangen von Nachrichten, sowohl passiv als auch aktiv, möglich, ohne dass das Message-Objekt vorher nochmals in einen Puffer eingetragen werden muss.
Aktives Message-Objekt XCAN_EXCL_BUFFER nicht gesetzt	Das Message-Objekt wird zum Zeitpunkt, an dem es aktiv Daten sendet oder anfordert, in einen der freien temporären Puffer eingetragen (falls es nicht schon eingetragen ist). Der genutzte Puffer wird vorübergehend gesperrt. Daraufhin wird die Nachricht auf dem CAN-Bus verschickt. Nach der Empfangsbestätigung steht der verwendete temporäre Message-Puffer wieder zur Verfügung.
Passives Sendeobjekt XCAN_EXCL_BUFFER nicht gesetzt	Das Message-Objekt wird zum Zeitpunkt seiner Konfiguration dauerhaft in den Nur-Empfangspuffer eingetragen. Zum Zeitpunkt, an dem es aktiv Daten sendet oder anfordert, wird es in einen der temporären Message-Puffer eingetragen (falls es nicht schon eingetragen ist). Daraufhin wird die Nachricht auf dem CAN-Bus verschickt.
Passives Empfangsobjekt XCAN_EXCL_BUFFER nicht gesetzt	Das Message-Objekt wird zum Zeitpunkt seiner Konfiguration dauerhaft in den Nur-Empfangspuffer eingetragen und wartet darauf, bis es eine Nachricht erhält.

- Um eine optimale Bearbeitungsgeschwindigkeit zu erreichen (d.h. die Anzahl und die Dauer der Zugriffe auf das Modul zu minimieren), legen Sie eine optimale Anzahl an exklusiven Puffern fest und weisen den gewünschten Message-Objekten die Eigenschaft *XCAN_EXCL_BUFFER* zu. Dies sollte für Message-Objekte gelten, die sich durch einen besonders häufigen Zugriff auf den CAN-Bus auszeichnen, oder die sehr zeitkritisch sind.
- Gibt es eine sehr hohe Zahl an Message-Objekten, die alle gleich häufig an der CAN-Bus-Kommunikation teilnehmen, kann es sinnvoll sein, keine exklusiven Puffer zu verwenden, damit der MDD die Message-Objekte möglichst gleichmäßig auf die 14 Puffer verteilen kann. Würden sich alle Message-Objekte z.B. nur einen temporären Puffer teilen, müsste dieser praktisch vor jeder Kommunikation neu mit dem jeweiligen Message-Objekt geladen werden.
- Achten Sie bei der Vergabe der IDs für die Message-Objekte, die in den Nur-Empfangspuffer eingetragen werden, darauf, dass nicht zu viele unerwünschte

Nachrichten das Akzeptanzfilter des Nur-Empfangspuffers passieren. Dies erreichen Sie, in dem Sie IDs verwenden, die möglichst ähnliche Bitmuster aufweisen.

- Wenn sie ein Message-Objekt als passives Sendeobjekt konfigurieren, so sollten Sie die Eigenschaft *XCAN_EXCL_BUFFER* zuweisen.

10.9.2. Modul Device Treiber

10.9.2.1. Installation

Der Modul-Device-Treiber für OsX hat die Programmnummer 803Ah und den Dateinamen mxcan2.exe. Der Modul-Device-Treiber für Windows hat den Namen mxcan2.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x803A, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=803A

10.9.2.2. Kanaleigenschaftsstruktur CPS_XCAN

Die CPS für das Modul hat den Namen CPS_XCAN.

10.9.2.3. Aktive Sendeobjekte

Aktive Sendeobjekte versenden ihre Daten durch einen (aktiven) Aufruf des Kanal-Ausgabedienstes durch das Anwenderprogramm. Das Datentelegramm wird verschickt, sobald der Bus frei ist. Ein solches Objekt wird mit folgender CPS erzeugt bzw. ein Kanal zu einem solchen Objekt geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_BUS_OUT*	Kanal zu einem Sendeobjekt
.usChannel	*0 oder 1*	CAN-Schnittstelle auf dem Modul
.ulCanID	*0 bis 7F0h*	CAN-Bus Identifier für 11-Bit- Identifier
	0 bis 1FFFFFF0h	CAN-Bus Identifier für 29-Bit- Identifier
.usFlags	*_CP_SYNC_CALLBACK*	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
	XCAN_NO_INT	Die Callback-Funktion wird nicht aufgerufen. Statt dessen wird solange im Ausgabedienst gewartet,

Strukturelement	Werte	Bedeutung
		bis das Telegramm auf den Bus versendet werden kann (max. die durch *usTimeout* gegebene Zeit).
	XCAN_EXCL_BUFFER	Kann zusätzlich gesetzt werden, um einen Puffer des CAN-Controllers für das Objekt zu reservieren.
.usTimeout	*0 bis 65535*	Zeit (in µs), die max. vergehen darf, bis ein Sendewunsch tatsächlich auf den Bus abgesetzt werden kann.

Eingabe- und Ausgabedienst

Der Datentyp ist DATA_UCHAR. Ein Block besteht aus 0 bis 8 Bytes.

- **max_write_channel_block**
- **max_read_channel_block** (Zurücklesen der Daten)

Sonderdienst

- **max_channel_control,** Steuerbefehl CMD_START: die zuvor versendeten Daten nochmals senden. Es werden keine Daten übergeben.

Callback-Funktion

Sofern im Element *usFlags* der CPS das Flag *XCAN_NO_INT* nicht gesetzt ist, und beim Öffnen des Kanals eine Callback-Funktion angegeben wurde, wird diese aufgerufen, wenn das Telegramm erfolgreich versendet werden konnte. Der Callback-Funktion werden keine Daten übergeben.

10.9.2.4. Passive Sendeobjekte

Passive Sendeobjekte verschicken ihre Daten nur auf Anforderung (über den CAN-Bus) durch einen anderen Teilnehmer. Sobald ein von einem anderen Teilnehmer gesendeter Remote-Frame empfangen wird, werden die zu dem Zeitpunkt gültigen Daten versendet. Ein Schreibzugriff auf den Kanal stellt dem Sendeobjekt neue Daten zur Verfügung, ohne dass die Daten unmittelbar auf den Bus versendet werden.

Ein solches Objekt wird mit folgender CPS erzeugt bzw. ein Kanal zu einem solchen Objekt geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_BUS_OUT*	Kanal zu einem Sendeobjekt
.usChannel	*0 oder 1*	CAN-Schnittstelle auf dem Modul
.ulCanID	*0 bis 7F0h*	CAN-Bus Identifier für 11-Bit- Identifier
	0 bis 1FFFFFF0h	CAN-Bus Identifier für 29-Bit- Identifier
.usFlags	*XCAN_REMOTE*	Muss gesetzt sein.
	XCAN_NO_INT	Das Versenden der Daten erzeugt keinen Interrupt. Die Callback-Funktion wird nicht aufgerufen.
	XCAN_EXCL_BUFFER	Kann zusätzlich gesetzt werden, um einen Puffer des CAN-Controllers für das Objekt zu reservieren.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usTimeout	*0*	reserviert

Eingabe- und Ausgabedienst

Der Datentyp ist DATA_UCHAR. Ein Block besteht aus 0 bis 8 Bytes.

- **max_write_channel_block**
- **max_read_channel_block** (Zurücklesen der Daten)

Sonderdienst

- **max_channel_control,** Steuerbefehl CMD_START: die gerade aktuellen Daten aktiv versenden. Es werden keine Daten übergeben.

Callback-Funktion

Sofern im Element *usFlags* der CPS das Flag *XCAN_NO_INT* nicht gesetzt ist, und beim Öffnen des Kanals eine Callback-Funktion angegeben wurde, wird diese aufgerufen, wenn das Telegramm erfolgreich versendet werden konnte. Der Callback-Funktion werden keine Daten übergeben.

10.9.2.5. Aktive Empfangsobjekte

Durch einen Lesezugriff auf ein aktives Empfangsobjekt wird (aktiv) ein Remote-Frame versendet, um von einem anderen Busteilnehmer Daten anzufordern. Im Kanal-Eingabedienst wird solange gewartet, bis die Antwort vom anderen

Busteilnehmer eintrifft, jedoch maximal nur die im Parameter *usTimeout* angegebene Zeit. Es wird kein Interrupt ausgelöst.

Ein solches Objekt wird mit folgender CPS erzeugt bzw. ein Kanal zu einem solchen Objekt geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_BUS_IN*	Kanal zu einem Empfangsobjekt
.usChannel	*0 oder 1*	CAN-Schnittstelle auf dem Modul
.ulCanID	*0 bis 7F0h*	CAN-Bus Identifier für 11-Bit- Identifier
	0 bis 1FFFFFF0h	CAN-Bus Identifier für 29-Bit- Identifier
.usFlags	*XCAN_REMOTE + XCAN_NO_INT*	Müssen gesetzt sein
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
	XCAN_EXCL_BUFFER	Kann zusätzlich gesetzt werden, um einen Puffer des CAN-Controllers für das Objekt zu reservieren.
.usTimeout	*0 bis 65535*	Zeit (in µs), die max. vergehen darf, bis die Antwort empfangen wird.

Eingabedienst

Der Datentyp ist DATA_UCHAR. Ein Block besteht aus 0 bis 8 Bytes.

- **max_read_channel_block**

Sonderdienst

- **max_channel_control,** Steuerbefehl CMD_START: Remote-Telegramm versenden. In diesem Fall wird beim Eintreffen der Antwort die Callback-Funktion aufgerufen. Es werden keine Daten übergeben.

Callback-Funktion

Wenn eine Callback-Funktion angegeben wurde, wird diese nur aufgerufen, wenn über den o.g. Sonderdienst Daten angefordert werden. Die empfangenen Daten (bis zu 8 Bytes) werden der Callback-Funktion übergeben.

10.9.2.6. Passive Empfangsobjekte

Passive Empfangsobjekte empfangen Daten vom Bus nur, wenn ein anderer Teilnehmer ihnen etwas sendet. Sie lösen selbst keine Aktivität auf dem CAN-Bus aus.

Ein solches Objekt wird mit folgender CPS erzeugt bzw. ein Kanal zu einem solchen Objekt geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_BUS_IN*	Kanal zu einem Empfangsobjekt
.usChannel	*0 oder 1*	CAN-Schnittstelle auf dem Modul
.ulCanID	*0 bis 7F0h*	CAN-Bus Identifier für 11-Bit- Identifier
	0 bis 1FFFFFF0h	CAN-Bus Identifier für 29-Bit- Identifier
.usFlags	*_CP_SYNC_CALLBACK*	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
	XCAN_EXCL_BUFFER	Kann zusätzlich gesetzt werden, um einen Puffer des CAN-Controllers für das Objekt zu reservieren.
.usTimeout	*0*	reserviert

Eingabedienst

Die gerade aktuellen Daten eines Empfangsobjekts können jederzeit durch einen Lesezugriff gelesen werden.

Der Datentyp ist DATA_UCHAR. Ein Block besteht aus 0 bis 8 Bytes.

- **max_read_channel_block**

Sonderdienst

- **max_channel_control,** Steuerbefehl CMD_START: Remote-Telegramm für ein passives Empfangsobjekt versenden (dadurch wird es zum sowohl aktiven als auch passiven Empfangsobjekt). Es werden keine Daten übergeben.

Callback-Funktion

Der Empfang der Daten wird dem Anwenderprogramm durch den Aufruf der Callback-Prozedur mitgeteilt, sofern diese beim Öffnen des Kanals angegeben wurde. Dabei werden die empfangenen Daten (max. 8 Bytes) der Callback-Funktion übergeben.

10.9.2.7. Bus-Steuerung

Das Bus-Steuerungs-Device erlaubt das Abschalten einer CAN-Bus-Schnittstelle von der Bus-Kommunikation, die Einstellung der Bit-Timing-Parameter sowie die Festlegung einer Fehlerbehandlungs-Funktion (s. 10.9.2.9.).

Ein solches Objekt wird mit folgender CPS erzeugt bzw. ein Kanal zu einem solchen Objekt geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Steuerkanal
.usChannel	*0 oder 1*	CAN-Schnittstelle auf dem Modul
.usIndexFirst	*XCAN_BUS_CTRL_INDEX*	Kanal-Nummer
.usIndexLast	*= .usIndexFirst*	Kanal-Nummer
.usFlags	*0*	Keine Bedeutung
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.

Eingabe- und Ausgabedienst

Mit dem Ausgabedienst wird der CAN-Bus parametriert. Die Bit-Timing-Parameter, der Busabschlußwiderstand und das ID-Format werden für die betreffende Schnittstelle festgelegt. Der Datentyp ist DATA_USHORT. Der Kanal ist rücklesbar.

- **max_write_channel_block**
- **max_read_channel_block** (Zurücklesen der eingestellten Daten)

Eine folgende Datenstruktur vom Typ XCAN_BITPARAMS wird von den Kanaldiensten erwartet:

Name	Typ	Bereich	Bedeutung
Tq	USHORT	125-8000	Time-Quantum in ns (Vielfaches von 125)
Tseg1	USHORT	3-16	Zeit tseg1/Tq vor dem nominellen Abtastzeitpunkt
Tseg2	USHORT	2-8	Zeit tseg2/Tq nach dem nominellen Abtastzeitpunkt
Tsjw	USHORT	1-4	Synchronisationssprungweite tsjw/Tq
SpB	USHORT	1 oder 3	Zahl der Abtastungen pro Bit

Name	Typ	Bereich	Bedeutung
Rbus[1]	USHORT	0 oder 1	Busabschlusswiderstand aus bzw. eingeschaltet (für High Speed CAN-Bus-Schnittstelle) bzw. Busabschlusswiderstände 5,6 kΩ oder 510 Ω (für Low-Speed CAN-Bus-Schnittstelle)
MsgFormat	USHORT	11 oder 29	Anzahl der Bits im CAN-Identifier

Sonderdienste

- **max_channel_control,** Steuerbefehle CMD_START, CMD_STOP: CAN-Bus ein- bzw. ausschalten. Es werden keine Daten übergebe.

Nach dem Öffnen des ersten Kanals zu einem Sende- oder Empfangsobjekt ist der Bus standardmäßig eingeschaltet.

Callback-Funktion

Der Aufruf einer Callback-Funktion erfolgt bei Kommunikationsfehlern auf dem CAN-Bus. Näheres dazu im Kap. 10.9.2.9.

10.9.2.8. Erläuterung der Bit-Timing-Parameter

Laut CAN-Spezifikation ist ein Bit in vier Zeitabschnitte unterteilt:

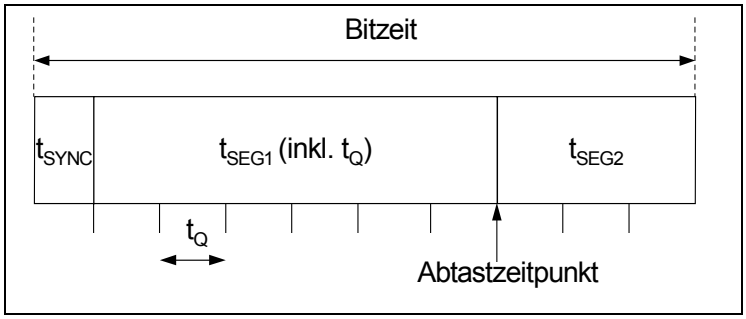
- Zeitabschnitt t_{SYNC} zur Synchronisation
- Zeitabschnitt t_{PROP} zur Kompensation der physikalischen Verzögerungszeiten
- Zeitabschnitt t_{SEG1} vor dem nominellen Abtastzeitpunkt
- Zeitabschnitt t_{SEG2} nach dem nominellen Abtastzeitpunkt

Die beiden Zeitabschnitte t_{SEG1} und t_{SEG2} dienen der Nachsynchronisation zum Ausgleich der Phasendifferenz zwischen Sende- und Empfangsoszillator. Für die auf dem

[1] Bei fehlertoleranten CAN-Bus-Systemen muss der CAN-Bus mit einem Busabschlusswiderstand von 100 Ohm abgeschlossen werden. Dieser ist dabei über das gesamte Netzwerk verteilt, wobei jeder CAN-Knoten nur einen Teil des Gesamtwiderstandes enthält. Der Busabschlusswiderstand berechnet sich aus der Parallelschaltung der Widerstände aller Knoten. Die Widerstände an den einzelnen Bus-Teilnehmern können verschieden sein.
Auf dem X-CAN-2 stehen für die fehlertoleranten Schnittstellen die beiden Widerstandswerte 5,6 kOhm (für Systeme mit vielen Teilnehmern) und 510 Ohm (für Systeme mit wenigen Teilnehmern) zur Auswahl. Dass in Systemen mit weniger als 5 Teilnehmern der optimale Gesamtwiderstand von 100 Ohm nicht erreicht werden kann, stört in Systemen mit geringer Ausdehnung in der Regel nicht. Details dazu finden sich z.B. in den „Application Hints Fault-Tolerant CAN-Transceiver“ von Philips Semiconductors.

Modul verwendeten CAN-Controller INTEL 82527 werden t_{PROP} und t_{SEG1} zu einem Segment t_{SEG1} zusammengefasst.

Alle Zeitabschnitte werden in Vielfachen des Time-Quantums t_Q angegeben. Das Time-Quantum t_Q ist immer ein ganzzahliges Vielfaches der Periodendauer des CAN-Controller-Taktes auf dem Modul. Diese beträgt 125 ns.



Die Bitzeit berechnet sich nach der folgenden Formel:

$$\text{Bitzeit} = (t_{SEG1}/t_Q + t_{SEG2}/t_Q + 1) \cdot t_Q$$

Zusätzlich kann die *maximale Synchronisationssprungweite* (Resynchronisation Jump Width t_{SJW}) eingestellt werden. Dies ist die maximale, pro Nachsynchronisation zulässige Verlängerung bzw. Verkürzung von t_{SEG1} und t_{SEG2}. Diese Größe wird ebenfalls als Vielfaches von t_Q angegeben.

Zur Verbesserung der Störsicherheit kann die *Abtastrate pro Bit* (Samples per Bit) auf den Wert = 3 eingestellt werden.

Die Bit-Timing-Parameter sind frei konfigurierbar. Allerdings müssen folgende Bedingungen[2] erfüllt sein:

- $t_{SEG1}/t_Q + t_{SEG2}/t_Q \geq 7$
- $t_{SEG2} \geq t_{SJW}$
- $t_{SEG1} \geq t_{SJW} + t_{PROP}$ für Samples per Bit=1
- $t_{SEG1} \geq t_{SJW} + t_{PROP} + 2 \cdot t_Q$ für Samples per Bit=3

Wenn Sie sich an der CiA-Empfehlung 102, Version 2.0 orientieren wollen, so verwenden Sie eine der in der folgenden Tabelle aufgeführten *Standard-Bitraten* und stellen die dafür angegebenen Bit-Timing-Parameter ein:

[2] Der Zeitabschnitt t_{PROP} ist physikalisch vorgegeben: $t_{PROP} = 2 \cdot (t_{PHYS} + t_{DRV} + t_{CC})$. t_{PHYS}, t_{DRV} und t_{CC} sind die maximalen Verzögerungen, entstehend durch die physikalische Signallaufzeit auf dem Bus (berechnet sich aus Materialkonstante (typ < 10 ns/m) · Länge), durch den Ausgangstreiber (max. 80 ns) sowie durch den Eingangskomparator des CAN-Controllers (max. 60 ns).

Bitrate (kBit/s)	**t_Q (ns)**	**t_{SEG1}/t_Q**	**t_{SEG2}/t_Q**	**t_{SJW}/t_Q**	**Samples per Bit**
10	5000	16	3	2	1
20 [3]	2500	16	3	2	1
50	1000	16	3	2	1
125	500	13	2	1	1
250 [4]	250	13	2	1	1
500 [4]	125	13	2	1	1
800 [4]	125	7	2	1	1
1000 [4]	125	5	2	1	1

Beachten Sie auch, dass das Erzielen hoher Bitraten Auswirkungen auf die maximal zulässige Buslänge hat!

10.9.2.9. CAN-Bus-Fehlerbehandlungs-Funktion

Durch die Angabe einer Callback-Funktion beim Öffnen eines Kanals zum Bus-Control-Device kann eine Fehlerbehandlungsfunktion installiert werden, die vom MDD automatisch aufgerufen wird, wenn der CAN-Controller einen Fehler signalisiert.

Die Fehlerbehandlungsfunktion bekommt bei ihrem Aufruf einen ULONG-Parameter übergeben. Die Bedeutung dieses Parameters ist wie folgt:

[3] Werkseitig eingetragene Bit-Timing-Parameter.

[4] Nur für High-Speed Schnittstellen

Das Low-Word gibt mit einem der folgenden Werte den Zustand des CAN-Bus an:

- XCAN_BUS_ON Normalbetrieb
- XCAN_BUS_HEAVY Stark gestörter Bus
- XCAN_BUS_OFF Aufgrund schwerer Busstörungen hat der CAN-Controller sich automatisch vom CAN-Bus zurückgezogen
- XCAN_LINE_ERROR Vom Transceiver signalisierter Leitungsfehler (nur für Low-Speed Schnittstelle)

Im High-Word ist nähere Information über den zuletzt aufgetretenen Fehler codiert:

Wert	Bedeutung
1	Stuff Error (mehr als 5 gleiche Bits in einem Telegramm hintereinander)
2	Form Error (Format-Fehler im Telegramm)
3	Ack Error (Gesendetes Telegramm wurde von keinem Teilnehmer quittiert)
4, 5	Bit Error (beim Senden eines Bits wurde der falsche Wert zurückgelesen)
6	CRC-Error (fehlerhafte Checksumme)
übrige	reserviert

Die Fehlererkennung und –signalisierung wird vollständig vom CAN-Controller übernommen. Dazu besitzt jeder CAN-Controller einen Sende- und einen Empfangsfehlerzähler, die er beim Auftreten von Fehlern um einen vom Fehlertyp abhängigen Betrag inkrementiert. Nach jeder erfolgreichen Übertragung wird der Zähler um einen bestimmten Betrag dekrementiert.

In Abhängigkeit dieser beiden Zähler nimmt der Controller einen der drei folgenden Zustände ein:

- Beide Zähler < 128 (Betriebsart 'Fehleraktiv'): Die CAN-Schnittstelle nimmt voll an der Bus-Kommunikation teil und sendet bei der Erkennung von Fehlern automatisch einen aktiven Error-Frame. Die als fehlerhaft erkannte Nachricht wird hierbei durch gezieltes Verletzen des Nachrichtenprotokolls zerstört.
- Einer der beiden Zähler überschreitet den Wert 128 (Betriebsart 'Fehlerpassiv'): Die CAN-Schnittstelle nimmt weiter voll an der Bus-Kommunikation teil, sendet nun aber bei der Erkennung von Fehlern einen passiven Error-Frame, der das Nachrichtenprotokoll nicht zerstört.

- Einer der beiden Zähler überschreitet den Wert 256 (Betriebsart 'Bus Off'): Die CAN-Schnittstelle wird von der Bus-Kommunikation abgeschaltet. Diese Betriebsart kann nur durch die Sonderdienste CMD_STOP und anschließend CMD_START (s.o.) verlassen werden. In diesem Falle werden beide Fehlerzähler wieder zurückgesetzt.

Der CAN-Controller schaltet zwischen den Betriebsarten 'Fehleraktiv' und 'Fehlerpassiv' selbstständig hin und her.

Ein Fehlerstand > 96 wird als 'Stark gestörter Bus' interpretiert und genau wie der Übergang in die Betriebsart 'Bus-Off' mit einem Interrupt signalisiert. In diesen beiden Fällen wird die angegebene Fehler-Funktion aufgerufen.

10.9.2.10. Akzeptanzfilter für den Nur-Empfangspuffer

Das Akzeptanzfilter dient dazu, eine Anzahl von CAN-Telegrammen mit unterschiedlichen Identifiern zu empfangen, ohne dass für jeden Identifier ein eigenes Message-Objekt angelegt werden muss.

Soweit Message-Objekte nicht in eigenen Puffern des CAN-Controllers eingetragen sind, geschieht ihr Empfang über den Nur-Empfangspuffer (s. Kap. 10.9.1.4). Für diesen Message-Puffer gibt es außer dem Identifier-Wert noch einen Akzeptanzfilter. Dieses bestimmt, welche Bits der empfangenen ID exakt mit dem abgespeicherten ID-Wert übereinstimmen müssen und welche nicht. Alle konfigurierten Message-Objekte, die über den Nur-Empfangspuffer empfangen werden sollen, gehen automatisch in die Bildung des Wertes des Akzeptanzfilters ein.

Über das Akzeptanzfilter-Device ist es möglich, den Filter manuell zu verändern, so dass das Modul ganze Gruppen von Message-Objekten oder sogar alle Message-Objekte empfangen kann, ohne die Message-Objekte einzeln zu konfigurieren. Dazu ist ein Kanal mit folgender CPS zu öffnen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Steuerkanal
.usChannel	*0 oder 1*	CAN-Schnittstelle auf dem Modul
.usIndexFirst	*0 bis 10 bzw.*	Offset des niederwertigsten Bits im 11 bzw.
	0 bis 28 [1]	29 Bit umfassenden Akzeptanzfilter
.usIndexLast	*0 bis 10 bzw.*	Offset des höchstwertigsten Bits im 11 bzw.
	0 bis 28	29 Bit umfassenden Akzeptanzfilter
.usFlags	*XCAN_REMOTE*	Nur-Empfangspuffer empfängt nur Remote-Frames; wenn das Flag nicht gesetzt ist, empfängt er nur Data-Frames.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_ULONG.

- **max_write_channel_ulong**
- **max_read_channel_ulong** (zum Zurücklesen der eingestellten Maske)

Ist ein Bit in der Akzeptanzmaske = 0, so empfängt der Nur-Empfangspuffer alle Telegramme, deren Identifier an dieser Bit-Stelle den Wert 0 oder 1 haben. Ist das Bit in der Akzeptanzmaske = 1, so empfängt der Nur-Empfangspuffer nur die Telegramme, deren Identifier an dieser Bit-Stelle mit dem automatisch errechneten Identifier-Wert übereinstimmen. Sind alle Bits der Akzeptanzmaske =0, bedeutet dies folglich, dass alle Identifier-Werte über den Nur-Empfangspuffer empfangen werden.

Im Initialisierungszustand (d.h. es sind keine CAN-Kanäle geöffnet und die Akzeptanzmaske ist noch nicht mit Hilfe des Akzeptanzfilter-Devices manuell gesetzt worden), ist der Wert der Akzeptanzmaske = FFFFFFFFh.

Beachten Sie, dass in der automatisch erzeugten Akzeptanzmaske niemals ein 0 gesetztes Bit über das Akzeptanzfilter-Device auf 1 gesetzt werden darf. Ansonsten würden konfigurierte Message-Objekte nicht mehr durch das Akzeptanzfilter gelangen. Der Versuch wird vom MDD mit einer Fehlermeldung quittiert.

[1] Zur Zeit muss *usIndexFirst* = *usIndexLast* = 0 gesetzt werden. D.h. das Akzeptanzfilter kann nur als Ganzes verändert werden.

Callback-Funktion

Die Callback-Funktion wird immer dann aufgerufen, wenn eine unkonfigurierte Nachricht empfangen wird, d.h. das Akzeptanzfilter passiert hat.

Die Callback-Prozedur bekommt die folgende Datenstruktur vom Typ XCAN_MESSAGE übergeben:

Name	Typ	Bedeutung
ulCanID	ULONG	Identifier
usDataSize	USHORT	Anzahl der im Array aucData eingetragenen gültigen Daten. Bei der Konfiguration des Nur-Empfangspuffers für den Empfang von Remote-Frames wird hier 0 eingetragen.
usFlags	USHORT	reserviert
aucData	UCHAR	Array aus 8 Byte, in das die empfangenen Daten eingetragen werden.

10.9.3. Pinbelegung

Signal	Modulstecker A	D-SUB-9 Stecker
CAN-Bus 1, GND	1, 21	6
CAN-Bus 1, CAN-L	2, 22	2
CAN-Bus 1, CAN-H	3, 23	7
CAN-Bus 1, GND	4, 24	3
CAN-Bus 1, 12V out [1]	5, 25	8
CAN-Bus 1, 7-24V in [1]	6, 26	4
CAN-Bus 1, 5V out	7, 27	9
CAN-Bus 1, Schirm [1]	8, 28	5
n.c.	9, 29	-
n.c.	10, 30	-
CAN-Bus 2, GND	11, 31	6
CAN-Bus 2, CAN-L	12, 32	2
CAN-Bus 2, CAN-H	13, 33	7
CAN-Bus 2, GND	14, 34	3
CAN-Bus 2, 12V out [1]	15, 35	8
CAN-Bus 2, 7-24V in [1]	16, 36	4
CAN-Bus 2, 5V out	17, 37	9
CAN-Bus 2, Schirm [1]	18, 38	5
n.c.	19, 39	-
n.c.	20, 40	-

[1] Die Pins <12V out>, <7-24V in> und <Schirm> sind nur in speziellen Bestückungsvarianten verfügbar, die erst bei Abnahme einer bestimmten Stückzahl möglich sind. Standardmäßig sind die Pins unbeschaltet.

Die Pinbelegung ist so ausgeführt, dass ein D-SUB-9 Stecker, der mit Modulstecker A verbunden ist, die CiA-Empfehlung 102 einhält. Dabei ist Pin 1 des D-SUB-9 Steckers nicht belegt. Bei der Variante X-CAN-2i/M ist Bus 1 die High-Speed-Schnittstelle und Bus 2 die Low-Speed-Schnittstelle.

10.9.4. Besondere Eigenschaften

Parameter	**Wert**	**Einheit**
CAN-Controller	INTEL 82527 (2 Stück)	-
CAN-Transceiver		
High Speed (X-CAN-2i/H)	Philips 82C250 o.ä.	-
Low Speed (X-CAN-2i/F)	Philips TJA 1054	-
Einstellbare Bitrate		
High Speed	5 bis 1000	kBit/s
Low Speed	5 bis 125	kBit/s
Busabschlusswiderstand, schaltbar		
High Speed typ. (±10%)	110	Ω
Low Speed typ.	510 oder 5600[1]	Ω
Trennspannung	500	V rms
Ausgänge Versorgungsspannung (5V out)		
Spannung min./typ./max.	4,7/5/5,1	V
Strom max. bei funktionierender Bus Kommunikation	15	mA
Temperatur-Bereich, Betrieb	0 bis 70	°C
optional	-40 bis +85	°C
Abmessungen	29x58x8	mm
Gewicht	10	g
Stromaufnahme (3,3V)		
High Speed typ. (X-CAN-2i/H)	510	mA
Low Speed typ. (X-CAN-2i/F)	410	mA
(die X-Bus Spannungen ±12V werden nicht benötigt)		

[1] An beiden CAN-Bus-Leitungen sind Busabschlusswiderstände von je 5,6 kΩ vorhanden. Durch das Einschalten wird ein Widerstand von je 560 Ω dazu parallel geschaltet. Für Netzwerke mit wenig Teilnehmern sollte der niedrige Widerstand gewählt werden, in großen Netzwerken ein hoher Busabschlusswiderstand.
Die CAN-L Leitung wird immer nach VCC (5V) terminiert.

Notizen

X-COM-4

4 serielle Schnittstellen
(2 x RS-232 und 2 x RS-232, RS-422 oder RS-485)

10.10. X-COM-4

Inhaltsverzeichnis

10.10.1. Beschreibung

Das X-COM-4 bietet 4 serielle Schnittstellen. Zwei davon sind als RS-232 mit allen Modem-Steuersignalen ausgeführt, die beiden anderen können per Software als RS-232, RS-422 oder RS-485 umgeschaltet werden. Als RS-232 verfügen sie über 2 Modem-Steuersignale RTS und CTS. Jede Schnittstelle bietet für jede Kommunikationsrichtung ein 16 Byte großes FIFO. Die max. Baudrate beträgt 460,8 kBaud je Kanal. Das X-COM-4 hat die Typ-Nr. 82, Subtyp 1.

10.10.2. Modul-Device-Treiber

10.10.2.1. Installation

Der Modul-Device-Treiber für OsX hat die Programmnummer 8052h und den Dateinamen mxcom4.exe. Der Modul-Device-Treiber für Windows hat den Namen mxcom4.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8052, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8052

10.10.2.2. Kanaleigenschaftsstruktur CPS_XCOM4

Die CPS für das Modul hat den Namen CPS_XCOM4.

10.10.2.3. Funktionsweise

Der MDD bietet eine Zwischenpufferung der Sende- und Empfangdaten.

Senden

Die Daten, die dem MDD zum Senden übergeben werden, werden zunächst im MDD zwischengespeichert. Der MDD reagiert auf den Interrupt, mit dem das Modul signalisiert, dass im Sende-FIFO-Puffer des Moduls Platz frei ist. Der MDD schreibt dann die zwischengepufferten Daten in das Modul, von wo aus sie versendet werden. Auf diese Weise kann das Anwenderprogramm beliebig Sendedaten nachliefern, ohne dass es sich um die Abwicklung des Sendevorgangs kümmern muss. Die Beschränkung liegt in der Größe des MDD-internen Zwischenpuffers, die beim Öffnen des Kanals festgelegt wird.

Empfangen

Der MDD reagiert auf den Interrupt, mit dem das Modul signalisiert, dass Daten im Empfangs-FIFO der jeweiligen Schnittstelle vorliegen. Der MDD entnimmt daraufhin die Daten aus dem FIFO und puffert sie intern zwischen. Das Anwenderprogramm kann zu jedem Zeitpunkt Daten abholen. Falls gerade keine vorliegen, wird dies gemeldet. Die Größe des MDD-internen Zwischenpuffers wird beim Öffnen des Kanals festgelegt.

Handshake

Das Modul unterstützt 2 Protokolle, mit denen ein Überlauf des Empfangspuffers verhindert werden kann:

- RTS-CTS Handshake: Die RTS- und CTS-Signale von Sender und Empfänger sind gekreuzt. Der Sender signalisiert über sein RTS-Signal, dass er Daten zum Senden vorliegen hat. Wenn der Empfänger bereit für den Empfang ist, muss er dies wiederum über sein RTS-Signal melden. Daraufhin beginnt der Sender mit dem Versenden der Daten. Wenn sich eine bestimmte (in *ulXoffLim* einstellbare) Anzahl von Zeichen im Empfangspuffer befinden, setzt der Empfänger RTS =0. Der Sender muss auf diesen Signalwechsel an seinem CTS Eingang mit der Unterbrechung des Sendens reagieren. Sobald der Füllstand des Empfangspuffers im Empfänger eine bestimmte (in *ulXonLim* einstellbare) Schwelle wieder unterschreitet (dadurch dass die Daten vom Anwenderprogramm abgeholt wurden), setzt der Empfänger RTS wieder =1. Daraufhin kann der Sender wieder Daten senden. Hat der Sender keine weiteren Sendedaten vorliegen, beendet er die Sendung durch Zurücksetzen seines RTS-Signals.

- XON/XOFF Handshake: Dieses Protokoll arbeitet ohne Steuerleitungen. Statt dessen wird ein Zeichen als XOFF-Zeichen (üblicherweise 13h) und eines als XON-Zeichen (üblicherweise 11h) auf Sender- und Empfängerseite vereinbart. Der Empfänger kann den Sender durch Senden von XOFF anhalten. Durch Senden von XON wird dem Sender mitgeteilt, dass er Daten senden kann. Die vereinbarten Zeichen dürfen in den Sendedaten nicht vorkommen. Sie werden auch nicht in den Empfangspuffer eingetragen.

10.10.2.4. Serielle Schnittstellen

Ein Zugriff auf eine der vier seriellen Schnittstellen kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_COM*	Serielle Schnittstelle
.usIndexFirst	*0 ... 3*	Nummer der Schnittstelle
.usIndexLast	.usIndexFirst	Nummer der Schnittstelle
.usType	*COM_RS232* *COM_RS422* *COM_RS485*	RS232 Schnittstelle (Schnittstelle 0 ... 3) RS422 Schnittstelle (Schnittstelle 0 und 1) RS485 Schnittstelle (Schnittstelle 0 und 1)
.ulFlags	*_CP_EXCLUSIVE*	Dieses Flag muss gesetzt sein! Der Zugriff auf eine Schnittstelle erfolgt immer exklusiv.
	_COM_REPLACE_ERR_CHAR	Ein empfangenes Zeichen, das einen Fehler aufweist, wird durch das *cErrorChar* Zeichen ersetzt.
.usReadMode	*IO_MODE_RAM*	Die Empfangs-Daten werden aus einem Empfangspuffer des MDDs entnommen. Der MDD trägt die empfangenen Daten selbstständig in diesen Puffer ein.
.usWriteMode	*IO_MODE_RAM*	Die Sende-Daten werden in einen internen Sendepuffer des MDDs geschrieben. Der Puffer wird automatisch vom MDD geleert.
.usFlowControl		Hier wird festgelegt, welches Protokoll verwendet werden soll. Einer der 3 folgenden Werte muss gesetzt werden:
	COM_NO_FLOW	Es wird kein Protokoll verwendet.
	COM_XONXOFF_FLOW	Das Software Protokoll XON/XOFF wird zum Senden und Empfangen benutzt.
	COM_RTSCTS_FLOW	Die Steuerleitungen RTS und CTS werden für ein Handshake benutzt.

Strukturelement	Werte	Bedeutung
.ulBaudRate	*7 ... 460800*	Baudrate (in Baud) Alle Standardwerte sind einstellbar.[4]
.usByteSize		Anzahl der Bits pro Zeichen:
	COM_5_BITS	5 Bits
	COM_6_BITS	6 Bits
	COM_7_BITS	7 Bits
	COM_8_BITS	8 Bits
.usStopBits		Anzahl der Stopp-Bits:
	COM_ONE_STOPBIT	1 Stoppbit
	COM_ONE5_STOPBITS	1,5 Stoppbits
	COM_TWO_STOPBITS	2 Stoppbits
.usParity		Paritäts-Bit:
	COM_NO_PARITY	kein Paritätsbit
	COM_EVEN_PARITY	gerades Parität
	COM_ODD_PARITY	ungerade Parität
	COM_MARK_PARITY	Paritätsbit = Mark
	COM_SPACE_PARITY	Paritätsbit = Space
.ulXonLim	*1 ... ulXoffLim*	Erreicht der Empfangspuffer diesen Füllstand, wird die Gegenstelle mittels des gewählten Protokolls veranlasst, Zeichen zu senden.
.ulXoffLim	*1 ... ulRcvBuffer*	Erreicht der Empfangspuffer diesen Füllstand, wird die Gegenstelle mittels des gewählten Protokolls veranlasst, keine Zeichen mehr zu senden.
.cXonChar		Das XON Zeichen für das XON/XOFF Software Protokoll.
.cXoffChar		Das XOFF Zeichen für das XON/XOFF Software Protokoll.
.cErrorChar		Fehlerzeichen. Dieses Zeichen wird benutzt, um fehlerhafte Zeichen im Empfangsbuffer zu überschreiben.
.ulTmtBuffer	*0 ... 2048*	Größe des Sendepuffers
.ulRcvBuffer	*0 ... 10240*	Größe des Empfangspuffers
.ulTimeout	*0*	reserviert
.ulCallBackEvents	*0*	reserviert

[4] Die seriellen Schnittstellen Controller werden mit einem Takt von 7,3728 MHz versorgt. Die möglichen Baudraten berechnen sich nach folgender Formel: Baudrate = 460800 : x mit x=1..65535. Wird eine Baudrate angegeben, die nicht möglich ist, stellt der MDD den nächstgelegenen möglichen Wert ein. Der tatsächlich eingestellte Wert kann in diesem Fall mit einem Sonderdienst abgefragt werden.

Anmerkungen

Nach dem Öffnen eines RS-485 Kanals ist dieser standardmäßig auf Empfang geschaltet. Zum Umschalten stehen Sonderdienste zur Verfügung.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Ist bei der Übertragung ein Fehler aufgetreten, liefert der Dienst einen Fehler zurück (XCOM4_UART_ERROR). Die genaue Fehlerursache kann mit einem Sonderdienst ermittelt werden.

Sind bei einem Lesevorgang nicht die Anzahl der gewünschten Zeichen vorhanden, werden die im Puffer vorhandenen Zeichen zurückgegeben. Der Parameter ulSize enthält nach dem Aufruf die Anzahl der zurückgegebenen Zeichen.

Ist bei einem Schreibzugriff im Puffer nicht genügend Platz vorhanden, werden keine Daten übernommen und der Dienst liefert den Fehler ERR_BUFFER_FULL zurück.

Sonderdienste

- **max_channel_info,** Infotyp INFO_ERROR: um im Falle eines Fehlers eine genauere Fehlerursache zu ermitteln (zuvor **max_clear_error** aufrufen!). Es werden 4 Byte Daten zurückgeliefert, die aus einer Oder-Verknüpfung der folgenden Werte bestehen:
  - _XCOM4_ERROR_OVERRUN
  - _XCOM4_ERROR_FRAME
  - _XCOM4_ERROR_BREAK
  - _XCOM4_ERROR_RCV_BUFFER_OVERFLOW
  - _XCOM4_ERROR_PARITY
  - _XCOM4_ERROR_TMT
- **max_channel_control,** Steuerbefehle CMD_DIR_OUTPUT, CMD_DIR_INPUT: eine serielle RS-485 Schnittselle auf Senden bzw. Empfang umschalten. Der Funktion werden keine Daten übergeben.
- **max_channel_control**, Steuerbefehle CMD_XCOM4_CLEAR_RCV_BUFFER und CMD_XCOM4_CLEAR_TMT_BUFFER: Empfangs- bzw. Sendepuffer des MDDs löschen. Es werden keine Daten übergeben.

- **max_channel_info,** Infotyp INFO_XCOM4_BAUDRATE: die tatsächlich eingestellte Baudrate abfragen. Der Wert wird als ULONG-Wert zurückgegeben.
- **max_channel_control,** Steuerbefehel CTRL_XCOM4_MODEM_LINES: Steuerleitungen manuell schalten.

  In dem zu übergebenen ULONG-Wert bestimmen die Flags _XCOM4_DTR_0 und _XCOM4_DTR_1 den gewünschten Zustand des DTR-Signals (nur für RS-232 Schnittstellen 2 und 3) und die Flags _XCOM4_RTS_0 und _XCOM4_RTS_1 den Zustand des RTS-Signals (nur für RS-232, steht nicht zur Verfügung, wenn *.usFlowControl* = COM_RTSCTS_FLOW gesetzt ist).
- **max_channel_info,** Infotyp INFO_XCOM4_MODEM_LINES: Status der Steuerleitungen CTS, DSR, RI, DCD abfragen. Diese stehen nur für RS-232 Schnittstellen zur Verfügung. Es wird ein ULONG-Wert zurückgeliefert.

10.10.3. Treiberprogramm CQmax

Das in den bisherigen Handbuch-Auflagen beschriebene Treiberprogramm CQmax (bestehend aus den OsX-Programmen X1PA007 und X1PA008) ist weiterhin verfügbar. Die Beschreibung steht als Application Note AN098 zur Verfügung.

10.10.4. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

		Signal		
Pin	**Kanal**	**RS-232**	**RS-422**	**RS-485**
1..9	-	n.c.	n.c.	n.c.
10	-	GND	GND	GND
11	0	RCV	IN+	-
12	0	RTS	OUT+	DATA+
13	0	TMT	OUT-	DATA-
14	0	CTS	IN-	-
15	0	GND	GND	GND
16	1	RCV	IN+	-
17	1	RTS	OUT+	DATA+
18	1	TMT	OUT-	DATA-
19	1	CTS	IN-	-
20	1	GND	GND	GND
21	2	DCD	-	-
22	2	DSR	-	-
23	2	RCV	-	-
24	2	RTS	-	-
25	2	TMT	-	-
26	2	CTS	-	-
27	2	DTR	-	-
28	2	Ri	-	-
29	2	GND	-	-
30	2	-	-	-
31	3	DCD	-	-
32	3	DSR	-	-
33	3	RCV	-	-
34	3	RTS	-	-
35	3	TMT	-	-
36	3	CTS	-	-
37	3	DTR	-	-
38	3	Ri	-	-
39	3	GND	-	-
40	3	-	-	-

10.10.5. Besondere Eigenschaften

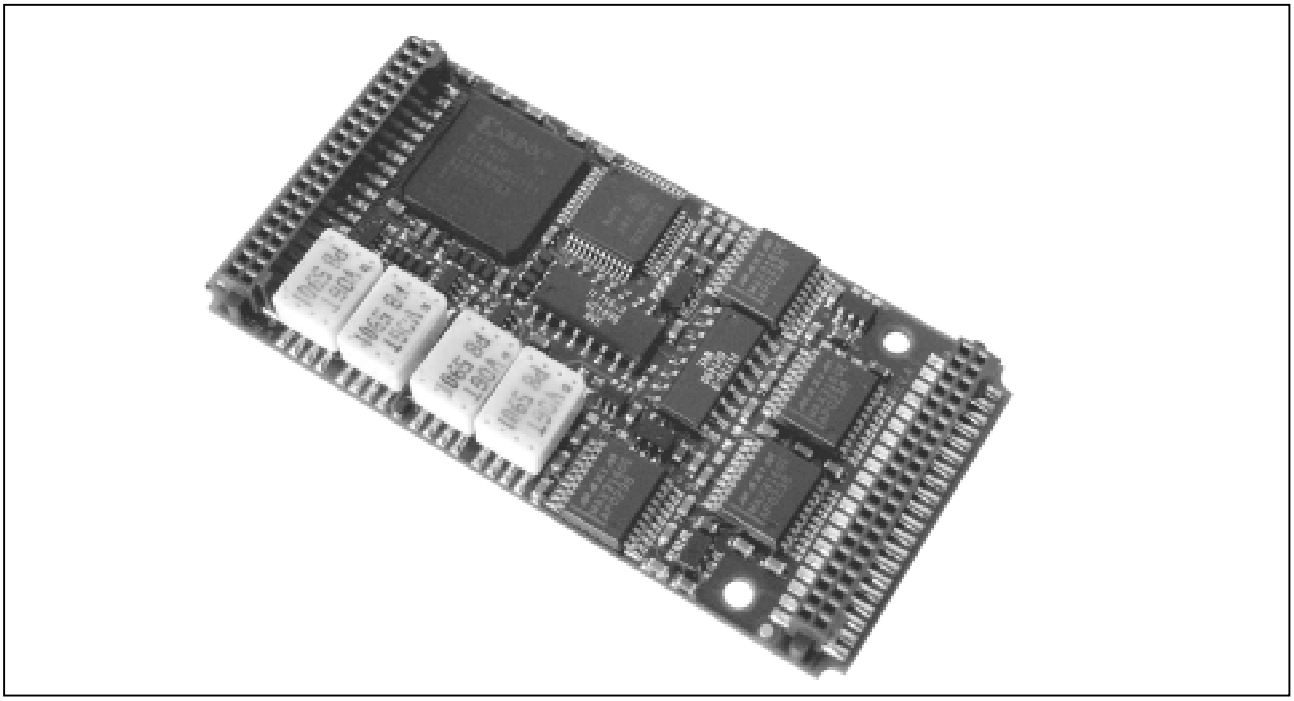
Parameter	**Wert**	**Einheit**
Serielle Schnittstellen (Kanal A und B)		
per Software wählbar	RS-232, RS-422 oder RS-485	
max. Baudrate	460,8	kBaud
FIFO (je Kanal und je Kommunikationsrichtung)	16	Byte[5]
RS-232		
Eingangsspannung, max.	±25	V
Eingangsschwelle Low, min.	0,6	V
Eingangsschwelle High, max.	2,0	V
Hysterese, typ.	0,5	V
Eingangswiderstand, min./typ./max.	3/5/7	kΩ
Ausgangsspannung (3 kΩ an GND), min./typ.	±5/±5,4	V
Ausgangsstrom (Ausgang an GND), max.	±60	mA
RS-422 und RS-485		
Eingangswiderstand, min.	48	kΩ
Eingangsstrom, max. (-7V .. +12V)	–0,15/0,25	mA
Differentielle Eingangsschwelle, min./max.	–50/-200	mV
Hysterese, typ.	30	mV
Differentielle Ausgangsspannung, min. (RS-422/RS-485) (RS-485 = 27 Ω, RS-422 = 50 Ω)	2/1,5	V
Temperatur-Bereich, Betrieb	–40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	8	g
Stromaufnahme 3,3V, typ. (die X-Bus Spannungen ±12V werden nicht benötigt)	160	mA

[5] Auf Anfrage auch größere FIFOs möglich.

Notizen

X-COM-8i

8 serielle Schnittstellen
(8 x RS-232 oder 8 x RS-4xx)



10.11. X-COM-8i

Inhaltsverzeichnis

10.11.1. Beschreibung

Das Modul ist in 2 Versionen verfügbar:

Typ	Subtyp	Modul-Typ	Serielle Schnittstellen	Bemerkung
88	0	X-COM-8i/232	8	8x RS-232
88	1	X-COM-8i/4xx	8	8x RS-422 bzw. RS-485

Die 8 seriellen Schnittstellen sind PC-kompatibel. Das Modul ist in einer Variante mit 8 RS-232 Schnittstellen und einer Variante mit 8 RS-4xx Schnittstellen (per Software zwischen RS-422 und RS-485 umschaltbar) lieferbar. Jede Schnittstelle verfügt über 64 Byte große Empfangs- bzw. Sendepuffer. Die max. Datenrate beträgt 3Mb/s (RS-485) bzw. 1Mb/s (RS-232);

10.11.2. Software

Zur Zeit der Drucklegung dieses Handbuches lag noch keine Spezifikation der Treibersoftware vor. Die Dokumentation zum MDD des Moduls wird auf www.sorcus.com zum Download bereitgestellt.

10.11.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

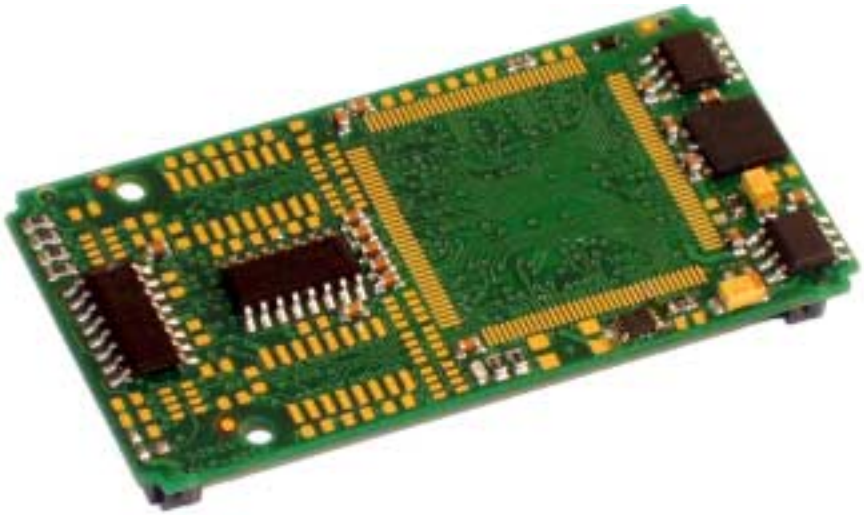
		Signal		
Pin	**Kanal**	**RS-232**	**RS-422**	**RS-485**
1	0	RCV	RCV+	Do not connect
2	0	RTS	TMT+	RCV+/TMT+
3	0	TMT	TMT-	RCV-/TMT-
4	0	CTS	RCV-	Do not connect
5	0	GND	-	-
6	1	RCV	RCV+	Do not connect
7	1	RTS	TMT+	RCV+/TMT+
8	1	TMT	TMT-	RCV-/TMT-
9	1	CTS	RCV-	Do not connect
10	1	GND	-	-
11	2	RCV	RCV+	Do not connect
12	2	RTS	TMT+	RCV+/TMT+
13	2	TMT	TMT-	RCV-/TMT-
14	2	CTS	RCV-	Do not connect
15	2	GND	-	-
16	3	RCV	RCV+	Do not connect
17	3	RTS	TMT+	RCV+/TMT+
18	3	TMT	TMT-	RCV-/TMT-
19	3	CTS	RCV-	Do not connect
20	3	GND	-	-
21..25	4	Wie Pin 1 bis 5	Wie Pin 1 bis 5	Wie Pin 1 bis 5
26..30	5	Wie Pin 1 bis 5	Wie Pin 1 bis 5	Wie Pin 1 bis 5
31..35	6	Wie Pin 1 bis 5	Wie Pin 1 bis 5	Wie Pin 1 bis 5
36..40	7	Wie Pin 1 bis 5	Wie Pin 1 bis 5	Wie Pin 1 bis 5

10.11.4. Besondere Eigenschaften

Parameter	Wert	Einheit
RS-232		
Eingangsspannung, max.	±25	V
Eingangsschwelle Low, min.	0,6	V
Eingangsschwelle High, max.	2,0	V
Hysterese, typ.	0,5	V
Eingangswiderstand, min./typ./max.	3/5/7	V
Ausgangsspannung (3 kΩ an GND), min./typ.	±5/±5,4	V
Ausgangsstrom (Ausgang an GND), max.	±60	mA
RS-422 und RS-485		
Eingangswiderstand, min.	48	kΩ
Eingangsstrom, max. (-7V .. +12V)	–0,15/0,25	mA
Differentielle Eingangsschwelle, min./max.	–50/-200	mV
Hysterese, typ.	30	mV
Differentielle Ausgangsspannung, min. (RS-422/RS-485) (RS-485 = 27 Ω, RS-422 = 50 Ω)	2/1,5	V

X-CPLD-38

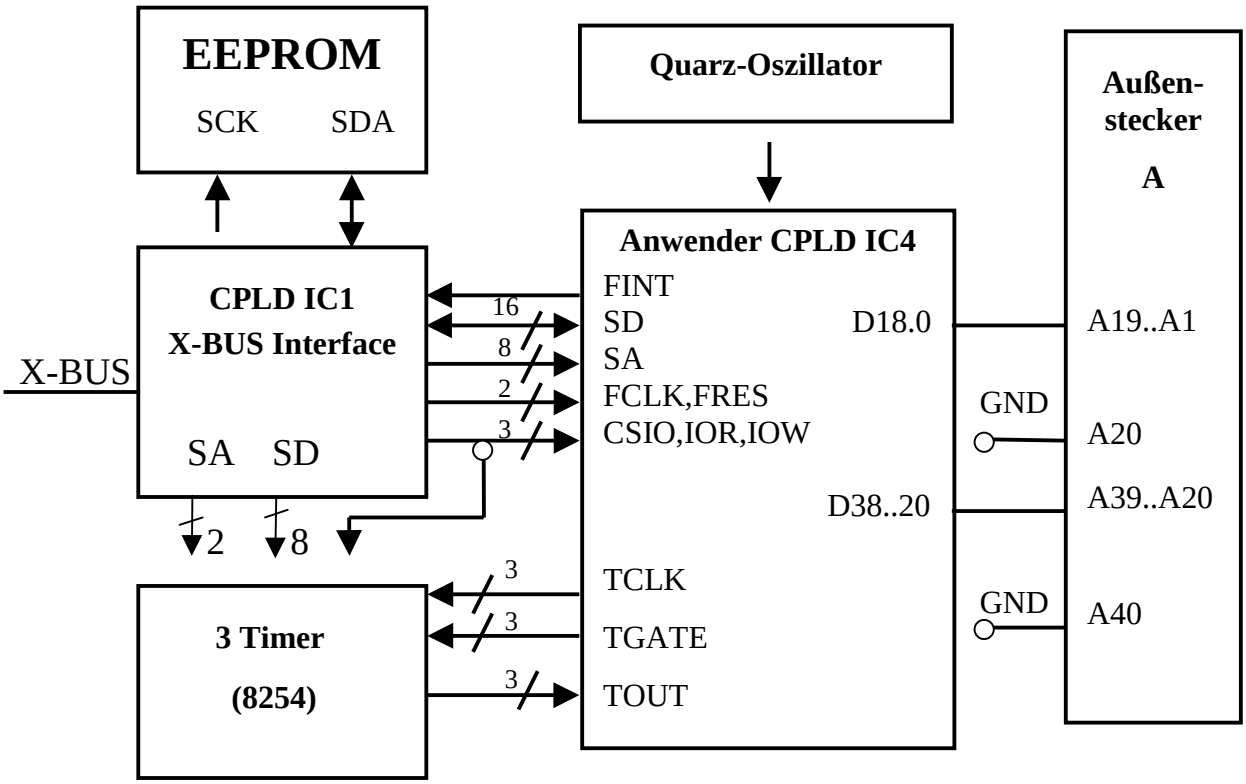
Frei programmierbares Digital-I/O-Modul



10.12. X-CPLD-38

Inhaltsverzeichnis

10.12.1. Blockschaltbild



10.12.2. Beschreibung

Das Modul X-CPLD-38 bietet die Möglichkeit, eigene Hardwaredesigns zu entwickeln und diese in die X-BUS Welt zu integrieren. Auf dem Modul befindet sich ein frei programmierbares CPLD (= IC4) von XILINX mit 288 Makrozellen (XC95288XL), das über ein weiteres CPLD (= IC1) an den X-BUS angekoppelt ist. Der dem Anwender zur Verfügung stehende Funktionsbereich im I/O Bereich des X-BUS umfasst 224 Byte. Es können wahlweise Wort- oder Byte-Zugriffe durchgeführt werden.

Außerdem sind 38 Pins des CPLDs nach außen auf den Modul-Stecker A geführt und können somit direkt als Aus- oder Eingänge verwendet werden.

Verwendung des 82C54 Timer-Chips von Intel

Auf dem Modul befindet sich außerdem noch ein Timer Baustein. Dabei handelt es sich um den 82C54-Chip von Intel. Die Programmierung erfolgt über das CPLD IC1. Die jeweils 3 Clock-, Gate- und Out-Leitungen des Timer-Chips sind direkt zum CPLD IC4 durchgeschleift und können vom Anwender frei benutzt werden. Für die

Clock-Leitungen steht dem Anwender eine von außen eingespeiste 1 MHz Clock zur Verfügung.

Alles Weitere über die Verwendungsmöglichkeiten der Timer kann aus dem Datenblatt entnommen werden. Herunterladbar z.B. unter:

www.developer.intel.com/design/periphrl/datashts/231244.htm

Programmiersoftware

Zum Programmieren des CPLDs kann das Programmpaket WebPACK von XILINX verwendet werden. Es enthält auch eine kostenlose Programmiersoftware, die Designs in VHDL, Verilog oder ABEL ermöglicht.

Herunterladbar über: www.xilinx.com

Pinbelegung des Kunden-programmierbaren CPLDs

Es muss beim Programmieren des CPLDs IC4 darauf geachtet werden, dass die Pinbelegung des gefitteten Designs zur Verschaltung des CPLDs passt. Dafür gibt es von SORCUS .ucf-Files, xcpld38a.ucf für ABEL-Designs und xcpld38v.ucf für VHDL-Designs. Beide Files können über die SORCUS Homepage heruntergeladen werden. Das entsprechende .ucf-File kopiert man in das Projektverzeichnis, indem das Design entstehen soll, und gibt ihm den selben Namen, den auch das Projekt trägt. Führt man nun einen „Fitting“-Vorgang durch, so werden die Informationen dieses Files automatisch in das Projekt übernommen. Zur Sicherheit empfiehlt es sich, vor dem Programmieren des Designs in das CPLD im „Fitting“-Report nachzuschauen, ob die Pinbelegung auch tatsächlich korrekt ist, da sonst durch eventuell verursachte Kurzschlüsse Schäden am Modul entstehen können.

Im folgenden ist das xcplda.ucf-File für ABEL abgedruckt. Benutzt man VHDL, so müssen alle nummerierten Pins geändert werden : sd3 -> sd<3>.

```
#PINLOCK_BEGIN

NET "ior"       LOC =  "S:PIN2";
NET "d36"       LOC =  "S:PIN4";
NET "d3"        LOC =  "S:PIN5";
NET "sd3"       LOC =  "S:PIN7";
NET "d17"       LOC =  "S:PIN10";
NET "d1"        LOC =  "S:PIN11";
NET "d21"       LOC =  "S:PIN12";
NET "d22"       LOC =  "S:PIN13";
NET "d23"       LOC =  "S:PIN14";
NET "d2"        LOC =  "S:PIN15";
NET "sd1"       LOC =  "S:PIN17";
NET "sd6"       LOC =  "S:PIN20";
NET "d0"        LOC =  "S:PIN21";
NET "d12"       LOC =  "S:PIN22";
NET "d15"       LOC =  "S:PIN23";
NET "d16"       LOC =  "S:PIN24";
NET "d20"       LOC =  "S:PIN25";
NET "fclk"      LOC =  "S:PIN30";
NET "d24"       LOC =  "S:PIN35";
NET "d25"       LOC =  "S:PIN39";
NET "d37"       LOC =  "S:PIN40";
NET "d4"        LOC =  "S:PIN41";
NET "d5"        LOC =  "S:PIN43";
NET "d31"       LOC =  "S:PIN46";
NET "d11"       LOC =  "S:PIN49";
NET "d13"       LOC =  "S:PIN51";
NET "d14"       LOC =  "S:PIN52";
NET "d27"       LOC =  "S:PIN53";
NET "d7"        LOC =  "S:PIN54";
NET "sd9"       LOC =  "S:PIN58";
NET "sd11"      LOC =  "S:PIN60";
NET "sd7"       LOC =  "S:PIN66";
NET "sd15"      LOC =  "S:PIN69";
NET "sd14"      LOC =  "S:PIN70";
NET "fint"      LOC =  "S:PIN71";
NET "sa7"       LOC =  "S:PIN91";
NET "d29"       LOC =  "S:PIN79";
NET "tclk2"     LOC =  "S:PIN80";
NET "d38"       LOC =  "S:PIN81";
NET "sd2"       LOC =  "S:PIN82";
NET "d9"        LOC =  "S:PIN83";
NET "tclk1"     LOC =  "S:PIN86";
NET "tgate2"    LOC =  "S:PIN88";
NET "sa6"       LOC =  "S:PIN138";
NET "d26"       LOC =  "S:PIN92";
NET "d35"       LOC =  "S:PIN93";
NET "d6"        LOC =  "S:PIN94";
NET "d10"       LOC =  "S:PIN98";
NET "d18"       LOC =  "S:PIN100";
NET "d33"       LOC =  "S:PIN102";
NET "´sd0"      LOC =  "S:PIN103";
NET "iow"       LOC =  "S:PIN104";
NET "tgate0"    LOC =  "S:PIN105";
NET "tgate1"    LOC =  "S:PIN106";
NET "tclk0"     LOC =  "S:PIN107";
NET "sd13"      LOC =  "S:PIN111";
NET "csio"      LOC =  "S:PIN112";
NET "qclk"      LOC =  "S:PIN113";
NET "tout1"     LOC =  "S:PIN115";
NET "tout0"     LOC =  "S:PIN116";
NET "sa3"       LOC =  "S:PIN128";
NET "sd12"      LOC =  "S:PIN118";
NET "d32"       LOC =  "S:PIN119";
NET "d34"       LOC =  "S:PIN120";
NET "led"       LOC =  "S:PIN121";
NET "sa2"       LOC =  "S:PIN129";
NET "sa1"       LOC =  "S:PIN130";
NET "sa0"       LOC =  "S:PIN74";
NET "d28"       LOC =  "S:PIN131";
NET "d8"        LOC =  "S:PIN132";
NET "sd4"       LOC =  "S:PIN133";
NET "sd8"       LOC =  "S:PIN134";
NET "sd5"       LOC =  "S:PIN135";
NET "sa4"       LOC =  "S:PIN117";
NET "sd10"      LOC =  "S:PIN137";
NET "sa5"       LOC =  "S:PIN136";
NET "d30"       LOC =  "S:PIN139";
NET "tout2"     LOC =  "S:PIN140";
NET "fres"      LOC =  "S:PIN143";

#PINLOCK_END
```

Beispieldesign

Als Ausgang für eine Eigenentwicklung steht ein Beispieldesign in ABEL und in VHDL zur Verfügung, welches die Verwendung der Register, das Auslesen der Eingänge, das Schalten der Ausgänge und die Verwendung der Modul-LED erläutert. Das Beispieldesign kann von der SORCUS Homepage heruntergeladen werden: www.sorcus.com

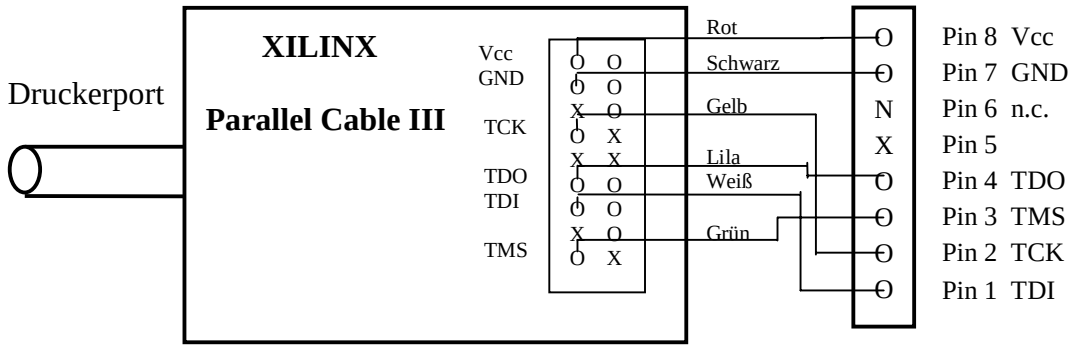
Programmieren mittels XILINX Parallel Cable III

Programmieren lässt sich das CPLD IC4 mit Hilfe einer Programmiersoftware, die im WebPACK Paket enthalten ist. Startet man diese Software, so werden beide CPLDs (IC1 und IC4) des Moduls angezeigt. Für das erste CPLD wählt man ein „Dummy“-Design (dummy.jed) aus, welches sich wie das Beispieldesign über unsere Homepage herunterladen lässt. Ausgewählt und programmiert wird allerdings nur das zweite CPLD mit dem selbst erzeugten JEDEC-File

Des weiteren benötigt man einen Programmieradapter: XILINX Parallel Cable III. Dieser kann über SORCUS bezogen werden unter der Bestellnummer KF-3199. Der Programmieradapter wird an den Druckerport des Rechners gesteckt und bietet am anderen Ende zwei 9 polige Steckerleisten mit jeweils 6 herausgeführten Pins. Verwendet wird die Steckerleiste, die für JTAG vorgesehen ist. Diese Pins werden über ein Kabel mit dem JTAG Stecker für das Modul verbunden. Die Pinbelegung der JTAG Stecker der SORCUS Produkte werden wie folgt mit dem XILINX Adapter verbunden:

MAX6pci: Im Falle der MAX6pci-Karte wird als JTAG-Stecker Stecker 13 auf der Max6pci verwendet. Beim Programmiervorgang muss die Max6pci Karte mit Strom versorgt sein, das zu programmierende Modul X-CPLD-38 muss sich auf Steckplatz 6 befinden. Die anderen Steckplätze dürfen nicht besetzt sein.

SORCUS Stecker 13 MAX6pci



X: Pin nicht vorhanden N: Pin wird nicht angeschlossen

10.12.3. Modul-Device-Treiber

10.12.3.1. Installation

Der Modul-Device-Treiber für das OsX hat die Programmnummer 802ah und den Dateinamen mcpld38.exe. Der Modul-Device-Treiber für Windows hat den Namen mxcpld38.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd(hModul, 1, 0, 0, 0x802b, NULL, &hMDD);

Befehl in INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=802b

10.12.3.2. Kanaleigenschaftsstruktur CPS_XCPLD38

Die CPS für das Modul hat den Namen CPS_XCPLD38.

10.12.3.3. Registerzugriffe

Um auf ein Register des Anwender-CPLDs zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_REGISTER*	Kanal zu einem Register des Anwender-CPLDs
.usIndexFirst	*10h ... ffh*	Nummer des Registers
.usIndexLast	*= .usIndexFirst*	Nummer des Registers
.usRegisterWidth	XCPLD38_BYTE_ REGISTER XCPLD38_WORD_ REGISTER	Breite des Registers
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0*	Reservierter Parameter
.usMode	*0*	Reservierter Parameter

Eingabe- und Ausgabedienst

In Abhängigkeit von *usRegisterWidth* ist der Datentyp des Kanals DATA_UCHAR (BYTE_REGISTER) oder DATA_USHORT (WORD_REGISTER):

Datentyp DATA_UCHAR:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

Datentyp DATA_UCHORT:

- **max_write_channel_ushort**
- **max_read_channel_ushort**

Anmerkung

Gerade Register (10h, 12h, 14h, ...) können eine Breite von 8-Bit (BYTE_REGISTER) oder 16-Bit (WORD_REGISTER) haben.

Ungerade Register (11h, 13h, 15h, ...) können nur eine Breite von 8-Bit (BYTE_REGISTER) haben.

10.12.3.4. Interrupts

Zum Empfang von Interrupts in einer Anwender-Callback-Funktion, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_HW_INT*	Kanal für den Empfang von Interrupts
.usIndexFirst	*0 ... 15*	Nummer des ersten Interrupts bei dem die Callback-Funktion aufgerufen werden soll
.usIndexLast	*0 ... 15*	Nummer des letzten Interrupts bei dem die Callback-Funktion aufgerufen werden soll
.usRegisterWidth	*0*	Reservierter Parameter
.usFlags	*_CP_EXCLUSIVE* *_CP_SYNC_* *CALLBACK*	Der Zugriff erfolgt immer exklusiv Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Anwender-Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im "Hintergrund" (NI-Task) abgearbeitet.
.usMode	*0*	Reservierter Parameter

Eingabe- und Ausgabedienst

Der Kanal verfügt über keinen Ein- bzw. Ausgabedienst. Der Kanal signalisiert über den Aufruf der Callback-Funktion das Auftreten eines Interrupts auf dem Modul.

Callback-Funktion

Tritt auf dem Modul ein Interrupt auf, so wird die beim Öffnen des Kanals angegebene Anwenderfunktion aufgerufen.
Die Anwenderfunktion bekommt 2 Byte Nutzdaten übergeben. In diesen Daten ist der Zustand der Interrupt-Pending-Bits festgehalten.

Beispiel

Der folgende Programmauschnitt zeigt das Öffnen eines Kanals. Tritt ein Interrupt auf dem Modul auf, so wird die Funktion "MyCallbackFunction" aufgerufen.

```
void OpenChannel()
{
    CPS_XCPLD38 cps;
    cps.usDevice = DEVICE_HW_INT;
    cps.usIndexFirst = 0;
    cps.usIndexLast = 15;
    cps.usRegisterWidth = 0;
    cps.usFlags = _CP_EXCLUSIVE | _CP_SYNC_CALLBACK;
    cps.usMode = 0;
    max_open_channel(hMdd, sizeof (cps), &cps, MyCallbackFunction, NULL, &hInterrupt);
}

MAX_CALLBACK MyCallbackFunction(MAXCHLHND handle, ULONG param, ULONG size, void* pData)
{

    return ERR_OK;
}
```

Voraussetzungen für den Einsatz unter OsX

Damit unter OsX ein Kanal mit Callback-Funkionalität geöffnet werden kann, müssen folgende Voraussetzungen erfüllt sein:

- Der Interrupt-Manager "irqmngr.exe" muss installiert sein.
  Befehl für die INS-Datei:
  MAXINST file="irqmngr.exe" no=a00f task=65 tasktype=MAX_NI_TASK autoinit
  Die Tasknummer kann beliebig gewählt werden.
- Ist das Flag _CP_SYNC_CALLBACK im Strukturelement usFlags beim Öffnen des Kanals nicht gesetzt, so muss zusätzlich der Message-Modul-Device-Treiber installiert sein.
  Befehl für die INS-Datei:
  MAXLOADMDD slot=0 layer=0 progno=8FFF
  Ist das Flag gesetzt, muss diser nicht installiert werden.

10.12.4. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Die 38 Pins des CPLDs sind direkt auf den Modul-Stecker herausgeführt:

Pin am CPLD	Pin am Modul-Stecker	Pin am CPLD	Pin am Modul-Stecker
D0	1	D20	21
D1	2	D21	22
D2	3	D22	23
D3	4	D23	24
D4	5	D24	25
D5	6	D25	26
D6	7	D26	27
D7	8	D27	28
D8	9	D28	29
D9	10	D29	30
D10	11	D30	31
D11	12	D31	32
D12	13	D32	33
D13	14	D33	34
D14	15	D34	35
D15	16	D35	36
D16	17	D36	37
D17	18	D37	38
D18	19	D38	39
GND	20	GND	40

10.12.5. Besondere Eigenschaften

Parameter	Wert	Einheit
Anzahl externer digitaler Ein- bzw. Ausgänge	38	-
Eingangsspannung (andere Standards auf Anfrage)		
(kompatibel mit 5V TTL und 5V, 3,3V und 2,5V CMOS)		
log. 0	< 0,8	V
log. 1	> 2,0	V
Input Leakage Current, max.	10	µA
Ausgangsspannung		
(kompatibel mit 5V TTL und 3,3V CMOS)		
log. 0, max. (IOL = 8mA)	0,4	V
log. 1, min. (IOH = -4mA)	2,4	V
Ausgangstrom log. 0, min./typ.	12/36	mA
log. 1, min./typ.	-12/-24	mA
Überspannungsfestigkeit der Eingänge	-0,5 .. +5,5	V
für Impulse < 10 ns und < 200mA	-2,0 .. +7,0	V
Watchdog-Timer, programmierbar	80 .. 270	ms
Timer, Anzahl	3	-
Auflösung	16	Bit
Betriebsarten (z.B. Rechteckgenerator, Ratengenerator, Impulsgenerator)	6	-
Temperatur-Bereich, Betrieb	0 .. +70	°C
optional (bitte anfragen)	-40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	8,75	g
Stromaufnahme (3,3V) (abhängig vom Anwender-Design)	250	mA
(die X-Bus Spannungen ±12V werden nicht benötigt)		

X-DA16i-4
X-DA14i-4

4 analoge Spannungs- und Stromausgänge
mit 16 bzw. 14 Bit Auflösung,
galvanisch getrennt

10.13. X-DA16i-4 und X-DA14i-4

Inhaltsverzeichnis

10.13.1. Beschreibung

Die Module X-DA14i-4/U, X-DA14i-4/Ui, X-DA16i-4/Ui und X-DA16i-4/U bieten 4 voneinander unabhängige Spannungsausgänge mit 14 bzw. 16 Bit Auflösung. Auf den beiden Modul-Varianten X-DA14i-4/Ui und X-DA16i-4/Ui ist jeder Ausgang zusätzlich auch als Stromausgang ausgeführt.

Der Ausgabebereich kann für jeden Ausgang per Software eingestellt werden. Jeder der 4 Ausgänge besitzt 2 bzw. 3 Ausgabe-Pins: einen für bipolare Spannung, einen für unipolare Spannung sowie bei den Modul-Varianten X-DA14i-4/Ui und X-DA16i-4/Ui zusätzlich einen für Strom. Prinzipiell wirken sich Ausgaben immer auf alle 2 bzw. 3 Pins aus. Den „richtigen“ Wert erhält man aber nur immer an einem der Pins. Welcher das jeweils ist, wird durch die Angabe des gewünschten Bereichs beim Öffnen des Kanals (s. Kap. 10.13.3.3) festgelegt.

Folgende Bereiche sind möglich:

- Spannung: 0..2,5V, ±2,5V, 0..5V, ±5V, 0..10V und ±10V
- Strom: 0..20mA, 0..-20mA
- Durch das Anlegen einer externen Referenzspannung kann für jeden Kanal zusätzlich ein im Bereich ±10V frei einstellbarer Ausgangsbereich definiert werden.

Die Ausgabe der Spannungs- oder Stromwerte kann per Software oder über einen externen Triggereingang gesteuert werden. Bei der externen Triggerung werden alle 4 Ausgänge zeitgleich ausgegeben.

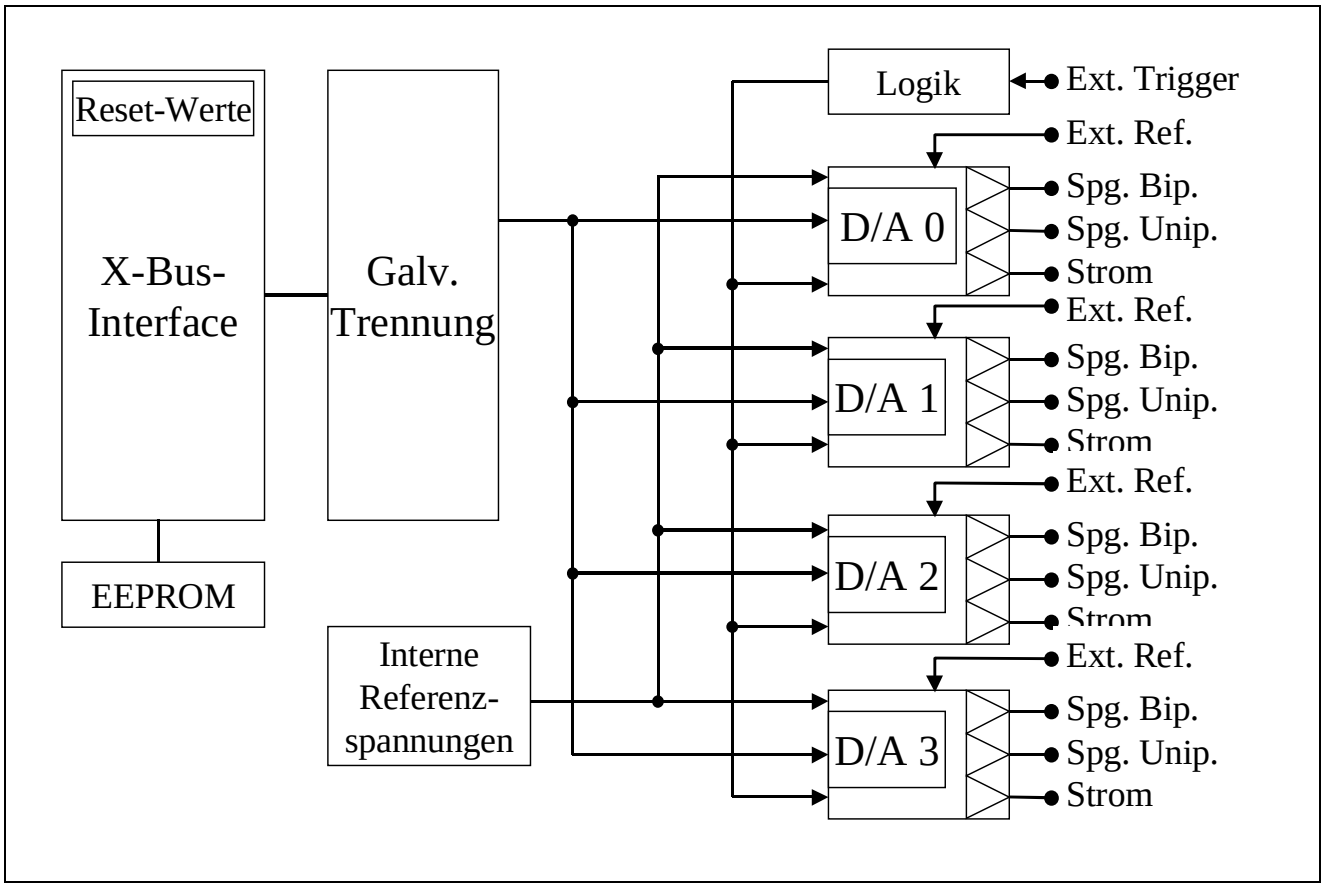
Nach Power-On-Reset liegen an allen Ausgängen 0V bzw. 0mA an. Die auszugebende Spannung bzw. Strom nach einem Hardware-Reset kann für jeden Kanal frei voreingestellt werden.

Alle analogen Ausgänge und der Triggereingang sind galvanisch vom X-Bus getrennt. Zwischen den analogen Ausgängen selbst besteht keine galvanische Trennung.

Zur Kompensation von Temperatur-Einflüssen wird die Temperatur auf dem Modul gemessen. Der aktuelle Wert kann jederzeit gelesen werden.

Das Modul hat die Typ-Nr. 50.

10.13.2. Blockschaltbild



10.13.3. Modul-Device-Treiber

10.13.3.1. Installation

Der Modul-Device-Treiber für alle 4 Modul-Varianten für das OsX hat die Programmnummer 8032h und den Dateinamen mxda164.exe. Der Modul-Device-Treiber für Windows hat den Namen mxda164.sys. Dieser gilt für alle hier beschriebenen Varianten des Modul. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0) erfolgt mit folgendem Funktionsaufruf:

Error = max_load_mdd (hModul, 1, 0, 0, 0x8032, NULL, &hMDD);

Der entsprechende Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0) lautet:

MAXLOADMDD slot=1 layer=0 progno=8032

10.13.3.2. Kanaleigenschaftsstruktur CPS_XDA164

Die CPS des Moduls hat den Namen CPS_XDA164.

10.13.3.3. Analoge Ausgänge

Ein Kanal zu einem der 4 analogen Ausgänge wird mit folgender CPS geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AOUT*	Kanal zu einem analogen Ausgang
.usIndexFirst	*0 ... 3*	Nummer des ersten Ausgangs
.usIndexLast	*0 ... 3*	Nummer des letzten Ausgangs
.usReadMode	*IO_MODE_RAM*	Die zuletzt geschriebenen Ausgabewerte können per Software zurückgelesen werden.
.usFlags	*_CP_EXCLUSIVE*	Muss gesetzt sein! Der Zugriff erfolgt exklusiv.
	_CP_UNCORRECTED	Die Werte werden keiner internen Korrektur unterzogen.
	_CP_HW_FORMAT	Die Werte werden direkt (ohne Umrechnung) an den/die DA-Wandler weitergegeben. (muss für usRange = *RANGE_USER_...* gesetzt sein). Ist dieses Flag gesetzt, hat das Flag *_CP_UNCORRECTED* keine Auswirkung.
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff.
	IO_MODE_RAM	Die Werte werden in einen Puffer geschrieben. Damit sie an den Ausgängen wirksam werden, ist zusätzlich ein Triggerkanal zu öffnen (s.u.).
		Alle zu einem Modul geöffneten AOUT-Kanäle müssen den selben Wert in *usWriteMode haben!*
.ausRange	Siehe Tabelle	Ausgabebereich

Eingabe- und Ausgabedienst

Wenn das Flag *_CP_HW_FORMAT* gesetzt ist, ist der Datentyp des Kanals *DATA_USHORT*:

- **max_write_channel_ushort** (Einzelkanal)
- **max_write_channel_block** (Mehrkanal)
- **max_read_channel_ushort** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Die Daten müssen dann in folgendem Format übergeben werden:

Digitalwert		**Spannungsausgänge**			**Stromausgang**
		X-DA16i-4	**X-DA14i-4**		
X-DA16i-4	X-DA14i-4	Bipolar	Bipolar	Unipolar	
FFFF	3FFF	-FSR	+FSR	FSR	FSR
8000	2000	0V	0V	½FSR	½FSR
0	0	+FSR	-FSR	0V	0mA

FSR (Full Scale Range) ist je nach gewähltem Bereich 10V, 5V, 2,5V oder die am Referenzeingang anliegende externe Spannung.

Wenn das Flag *_CP_HW_FORMAT* nicht gesetzt ist, ist der Datentyp des Kanals *DATA_LONG*:

- **max_write_channel_long** (Einzelkanal)
- **max_write_channel_block** (Mehrkanal)
- **max_read_channel_long** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Anmerkungen

Die Werte werden sowohl am bipolaren als auch am unipolaren Ausgang ausgegeben (bei X-DA14i-4/Ui und X-DA16i-4/Ui zusätzlich am Stromausgang). Der korrekte Wert liegt aber nur an dem Ausgangs-Pin an, der dem eingestellten Bereich (*.usRange*) entspricht.

Der Stromausgang darf nur verwendet werden, wenn *.usRange = RANGE_20MA* oder *RANGE_N20MA* eingestellt ist. **Bei anderen Einstellungen muss der Stromausgang unbeschaltet bleiben.** Bei der Verwendung einer externen Referenzspannung berechnet sich der max. Strom am Stromausgang zu $U_{Ref}/8$ (U_{Ref} in V, Strom in mA). Dabei ist unbedingt der max. zulässige Strom zu beachten!

Nur wenn *.usWriteMode = IO_MODE_DIRECT* gesetzt wird, werden die Werte, die dem Ausgabedienst übergeben werden, unmittelbar an den Ausgängen aktiv. Falls dort *IO_MODE_RAM* angegeben wird, schreibt der Ausgabedienst die Werte nur in einen Puffer. Damit sie von dort aktiviert werden, ist ein Trigger erforderlich. Dafür muss ein separater-Trigger-Kanal geöffnet werden. Dieser wirkt sich immer auf alle 4 Ausgänge aus. Wenn mehrere Kanäle geöffnet werden, müssen entweder alle *.usWriteMode = IO_MODE_DIRECT* oder alle *.usWriteMode = IO_MODE_RAM* verwenden!

Folgende Ausgabebereiche sind möglich (im CPS-Element *.usRange*):

Konstante	**Bedeutung**
RANGE_BIP_10V	-10 ... 10 V
RANGE_BIP_5V	-5 ... 5 V
RANGE_BIP_2V5	-2,5 ... 2,5 V
RANGE_UPP_10V	0 ... 10 V
RANGE_UPP_5V	0 ... 5 V
RANGE_UPP_2V5	0 ... 2,5 V
RANGE_20MA[1]	0 ... 20 mA
RANGE_N20MA[4]	0 ... -20 mA
RANGE_USER[3]	0 ... $U_{RefExtern}$ bzw. $\pm U_{RefExtern}$
Folgende Werte können nur bei Modul-Rev . A und B eingestellt werden:	
RANGE_USER_A_UP[1]	0 ... $-U_{RefExternA}$
RANGE_USER_A_BIP[6]	$-U_{RefExternA}$ ... $+U_{RefExternA}$
RANGE_USER_B_UP[4]	0 ... $-U_{RefExternB}$
RANGE_USER_B_BIP[4]	$-U_{RefExternB}$ ... $+U_{RefExternB}$
RANGE_USER_C_UP[4]	0 ... $-U_{RefExternC}$
RANGE_USER_C_BIP[4]	$-U_{RefExternC}$ ... $+U_{RefExternC}$
RANGE_USER_D_UP[4]	0 ... $-U_{RefExternD}$
RANGE_USER_D_BIP[4]	$-U_{RefExternD}$ ... $+U_{RefExternD}$

Mit Hilfe der externen Referenzspannungen (*RANGE_USER_...* bzw. *RANGE_USER*) können benutzerdefinierte Ausgangsbereiche realisiert werden. Dabei ist zu beachten, dass die angelegte Spannung max. 10V sein darf.

[1] nur bei X-DA16i-4/Ui und X-DA14i-4/Ui
[2] nur bei X-DA16i-4/Ui und X-DA14i-4/Ui ab Rev. E
[3] ab Rev. C
[6] nur bei *Rev. A und B*

Falls ein Kanal mit usRange = *RANGE_USER_...*[1] bzw. *RANGE_USER*[3] geöffnet werden soll, muss auch das Flag *_CP_HW_FORMAT* gesetzt sein!

Sonderdienste

- **max_channel_control** Steuerbefehl *CTRL_XDA16_SET_RESET_VAL*[7]: zum Festlegen der Ausgabewerte, die bei einem Reset an den Ausgängen aktiv werden. Dem Steuerbefehl muss ein Array vom Typ *XDA164_CONTROL_TYPE* übergeben werden:

  ```
  struct
  {
  USHORT usControl;
  USHORT usData;
  LONG lData;
  }XDA164_CONTROL_TYPE;
  ```

  Es müssen genau so viele Array-Elemente übergeben werden, wie der geöffnete Kanal Ausgänge umfasst.
  In *usControl* wird die Wirkung eines Reset festgelegt. Folgende Werte sind definiert:

  - *XDA164_USE_RESET_VALUE*: Nach Reset gibt der Ausgang den in *usData* bzw. *lData* angegebenen Wert aus. Ist in *.usFlags* der CPS das Bit _CP_HW_FORMAT gesetzt, wird der Wert aus *usData* verwendet, ansonsten der Wert aus *lData*.
  - *XDA164_ALL_ZERO*: Alle 3 Ausgänge eines Kanals (also Strom, bipolare Spannung, unipolare Spannung) geben nach Reset 0V bzw. 0 mA aus.
  - *XDA164_KEEP_LAST_VALUE*: Nach Reset bleibt der zuletzt ausgegebene Wert am Ausgang stehen.

  Die Werte aus *usData* bzw. *lData* müssen nur ausgefüllt werden, wenn *usControl = XDA164_USE_RESET_VALUE* gesetzt ist.

  Wird der Dienst nicht aufgerufen, bleiben nach einem Reset die zuletzt ausgegebenen Werte an den Ausgängen aktiv.

- **max_channel_control** Steuerbefehl *CTRL_CHANGE_RANGE:* Nachträgliches Verändern der Ausgabebereiche. Nach dem Öffnen eines Analog-Ausgabe-Kanals wird für alle zum Kanal gehörenden Ausgänge der in *.usRange* festgelegte Spannungs- oder Strombereich eingestellt. Sollen bei einem mehrere Ausgänge umfassenden Kanal die Ausgänge Werte in unterschiedlichen Bereichen ausgeben,

[7] Der Sonderdienst CTRL_XDA16_SET_RESET_VAL steht erst ab Modul-Rev. E zur Verfügung.

kann dies durch diesen Sonderdienst nachträglich verändert werden. Dazu ist ein Array vom Typ *XDA164_CONTROL_TYPE* (s.o.) zu übergeben. Es müssen genau so viele Array-Elemente übergeben werden, wie der geöffnete Kanal Ausgänge umfasst.
us*Control* legt fest, ob der Ausgangsbereich für den entsprechenden Ausgang geändert werden soll (dazu = *XDA164_CHANGE_RANGE* setzten). Der neue Ausgangsbereich ist in *usData* einzutragen. Ist *usControl* = 0, wird der Bereich für den Ausgang nicht verändert.

10.13.3.4. Trigger (ab Rev. E)

Ein Kanal, mit dem die Ausgabe an den analogen Ausgängen per Software und/oder Hardware getriggert werden kann, wird mit folgender CPS geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TRIGGER*[8]	Trigger für Analogausgabe
.usIndexFirst	*0*	Reserviert
.usIndexLast	*0*	Reserviert
.usReadMode	*0*	Reserviert
.usFlags	*_CP_EXCLUSIVE*	Muss gesetzt sein. Der Zugriff erfolgt immer exklusiv.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usWriteMode	*0*	Externer Trigger wird nicht aktiviert.
	TRIGGER_POS_EDGE	Die Triggerung erfolgt durch eine positive Flanke am externen Triggereingang.
	TRIGGER_NEG_EDGE	Die Triggerung erfolgt durch eine negative Flanke am externen Triggereingang.
	TRIGGER_HIGH_LEVEL	Die Triggerung erfolgt durch einen High-Pegel am externen Triggereingang.
	TRIGGER_LOW_LEVEL	Die Triggerung erfolgt durch einen Low-Pegel am externen Triggereingang.

[8] ab Rev. E

Ausgabedienst

Der Datentyp des Kanals ist *DATA_VOID*:

- **max_trigger_channel**.

Anmerkung

Unabhängig von der Einstellung in *usWriteMode* steht der Ausgabedienst immer zur Verfügung, d.h., auch wenn externe Triggerung aktiviert ist, steht zusätzlich die getriggerte Ausgabe per Software zur Verfügung.

Ein Triggersignal bzw. –befehl wirkt sich immer auf alle 4 Ausgänge des Moduls aus, auch wenn einzelne AOUT-Kanäle mit *usWriteMode = IO_MODE_DIRECT* geöffnet wurden.

Callback-Funktion

Wenn beim Öffnen des Trigger-Kanals eine Callback-Funktion angegeben wird, wird diese bei Eintreten der externen Triggerbedingung aufgerufen. Sie bekommt keine Werte übergeben. Wird für die Triggerereignisse *TRIGGER_HIGH_LEVEL* und *TRIGGER_LOW_LEVEL* eine Callback-Funktion angegeben, so wird beim Eintreten eines Triggers zunächst die Verarbeitung weiterer Ereignisse unterbunden, um eine Blockierung des Rechners zu vermeiden. Der Trigger kann mit **max_trigger_channel** in der Callback-Funktion wieder aktiviert werden.

10.13.3.5. Temperaturmessung (ab Rev. E)

Ein Kanal zur Messung der Modul-Temperatur wird mit folgender CPS geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AIN_SE*[9]	Trigger für Analogausgabe
.usIndexFirst	*0*	Reserviert
.usIndexLast	*0*	Reserviert
.usReadMode	*IO_MODE_RAM*	Die Temperatur wird im Hintergrund zyklisch gemessen. Beim Lesezugriff wird der zuletzt gemessene Wert zurückgeliefert.
	IO_MODE_DIRECT	Die Temperatur wird nur bei einem Lesezugriff neu gemessen.
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt exklusiv
.usWriteMode	*0*	Reserviert

[9] ab Rev. E

Eingabedienst

Der Datentyp des Kanals ist *DATA_USHORT*.

- **max_read_channel_ushort**

Es wird ein Wert im Bereich 0..FFFh zurückgeliefert. Die Temperatur wird daraus nach folgender Formel berechnet:
Temperatur[°C] = Wert / 3,44064 –273,24

10.13.3.6. LED (ab Rev. E)

Ein Kanal zur Ansteuerung der Modul-LED wird mit folgender CPS geöffnet:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_LED*[10]	Leuchtdiode
.usIndexFirst	*0*	Reserviert
.usIndexLast	*0*	Reserviert
.usReadMode	*IO_MODE_DIRECT*	Reserviert
.usFlags	0	Reserviert
.usWriteMode	*IO_MODE_DIRECT*	Reserviert

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist *DATA_UCHAR***.**

- **max_write_channel_uchar**
- **max_read_channel_uchar**

[10] ab Rev. E

10.13.4. Anschlusspins des Moduls

Kanal	Signal	Pin	Erläuterung
0	UNIOUT-0	6	Unipolarer Spannungsausgang
	BIPOUT-0	4	Bipolarer Spannungsausgang
	GND-0	1,5	Ground für Spannungsausgänge
	IOUT-0	2	Stromausgang[1]
	GND-0	3	Ground für Stromausgang
	REFEXT-0	9	Externe Referenzspannung
	GND-0	10	Ground für Referenzspannung
1	UNIOUT-1	16	Unipolarer Spannungsausgang
	BIPOUT-1	14	Bipolarer Spannungsausgang
	GND-1	11,15	Ground für Spannungsausgänge
	IOUT-1	12	Stromausgang[1]
	GND-1	13	Ground für Stromausgang
	REFEXT-1	19	Externe Referenzspannung
	GND-1	20	Ground für Referenzspannung
2	UNIOUT-2	26	Unipolarer Spannungsausgang
	BIPOUT-2	24	Bipolarer Spannungsausgang
	GND-2	21,25	Ground für Spannungsausgänge
	IOUT-2	22	Stromausgang[1]
	GND-2	23	Ground für Stromausgang
	REFEXT-2	29	Externe Referenzspannung
	GND-2	30	Ground für Referenzspannung
3	UNIOUT-3	36	Unipolarer Spannungsausgang
	BIPOUT-3	34	Bipolarer Spannungsausgang
	GND-3	31,35	Ground für Spannungsausgänge
	IOUT-3	32	Stromausgang[1]
	GND-3	33	Ground für Stromausgang
	REFEXT-3	39	Externe Referenzspannung
	GND-3	40	Ground für Referenzspannung
-	n.c.	17,18,27, 28,37,38	Nicht beschalten
-	TRIG[2]	8	Triggereingang
-	GND	7	Ground für Triggereingang

[1] nur bei X-DA16i-4/Ui und X-DA14i-4/Ui, bei X-DA16i-4/U und X-DA14i-4/Ui nicht belegt
[2] ab Rev E, bei Rev A bis D nicht beschalten

10.13.5. Besondere Eigenschaften

Parameter	Wert	Einheit
Anzahl der Kanäle	4	
Auflösung X-DA16i-4[11]	16	Bit
Auflösung X-DA14i-4[12]	14	Bit
Referenzspannungen, intern	2,5, 5, 10	V
Ausgangsbereiche		
Spannung, unipolar	0 .. +|U_{REF}|	V
Spannung, bipolar	±|U_{REF}|	V
Strom	0 .. 20, 0..-20	mA
Monotonität X-DA14i-4	13	Bit
Monotonität X-DA16i-4	15	Bit
Referenzspannungseingang		
max. Eingangsspannung	±10	V
ESD-Schutz der Referenzeingänge, max.	2000	V
Triggereingang		
Eingangsspannung, max.	–5..+12	V
Eingangsspannung, typ.	0..5	V
log. 0	<1,6	V
log. 1	>4,3	V
Temperatur-Bereich, Betrieb	0 .. 50[13]	°C
Abmessungen	29x58x8	mm
Gewicht	10	g

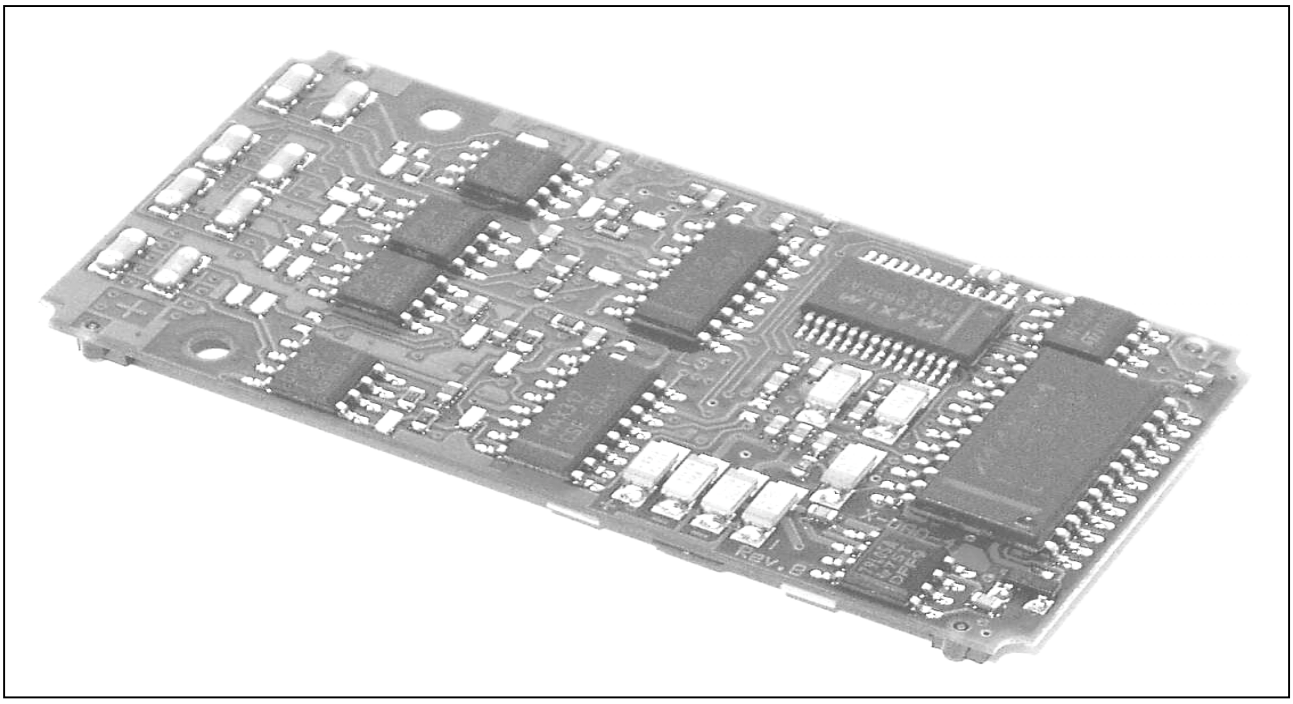
[11] im unipolaren Modus (VUNIOUT) gilt: 1 LSB = $|U_{REF}| / 2^{16}$ = 0,0015% FSR
im bipolaren Modus (VBIPOUT) gilt: 1 LSB = ½ $|U_{REF}| / 2^{16}$ = 0,0015% FSR

[12] im unipolaren Modus (VUNIOUT) gilt: 1 LSB = $|U_{REF}| / 2^{14}$ = 0,0061% FSR
im bipolaren Modus (VBIPOUT) gilt: 1 LSB = ½ $|U_{REF}| / 2^{14}$ = 0,0061% FSR

[13] ab Rev. E: 0..70°C

X-DAD-4

4 analoge Eingänge, 4 analoge Ausgänge,
18 digitale I/O-Pins



10.14. X-DAD-4

Inhaltsverzeichnis

10.14.1. Beschreibung

Das Modul X-DAD-4 stellt 4 analoge Differenzeingänge mit 12 Bit Auflösung zur Verfügung. Jeder dieser Eingänge kann bei Modul-Version /U per Software auf einen von 4 Eingangsspannungsbereichen (0..5 Volt, 0..10 Volt, ±5 Volt, ±10 Volt) eingestellt werden. Zusätzlich kann jeder Eingang auch per einlötbarem Widerstand auf Verstärkungsfaktoren von 1 bis 1000 eingestellt werden. Die Modul-Version /i hat den festen Eingangsbereich 0..20 mA.

Jeder der 4 anlogen Ausgänge kann bei Version /U per Software auf einen von 4 Spannungsbereichen (0..5 Volt, 0..10 Volt, ±5 Volt, ±10 Volt) eingestellt werden. Es gibt das Modul auch als Bestückungsvariante /i mit 0..20 mA Konstantstromausgängen.

Die Ist-Ausgangsspannung von allen 4 Kanälen kann zurückgelesen werden.

Das Modul hat außerdem 18 digitale Ein- bzw. Ausgänge mit TTL-Pegeln.

10.14.2. Modul-Device-Treiber

10.14.2.1. Installation

Der Modul-Device-Treiber des X-DAD-4/u für OsX hat die Programmnummer 802Ch und den Dateinamen mxdad4u.exe. Der Modul-Device-Treiber für Windows hat den Namen mxdad4u.sys. Für das X-DAD-4/i ist die Programmnummer 802Dh, der Programmname mxdad4i.exe (für OsX) bzw. mxdad4i.sys (für Windows).

Die Installation aus einem PC-Programm (z.B. für X-DAD-4/u auf Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x802C, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für X-DAD-4/u auf Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=802C

10.14.2.2. Kanaleigenschaftsstruktur CPS_XDAD4

Die CPS für das Modul hat den Namen CPS_XDAD4.

10.14.2.3. Digitale Kanäle

Die insgesamt 18 digitalen Leitungen des Moduls können als Eingänge, Ausgänge oder als umschaltbare I/O Pins verwendet werden. Die Richtung der Digitalleitungen kann nur in Vierer-Gruppen konfiguriert werden: D0..D3, D4..D7, D9..D12, D13..D16 bilden jeweils eine zusammenhängende Gruppe, deren einzelne Leitungen entweder alle als Eingang oder alle als Ausgang geschaltet sein können. Die Leitungen D8 und D17 sind einzeln konfigurierbar. Es sind folgende Gruppenkanäle zulässig:

4er-Gruppen: D0..D3, D4..D7, D9..D12, D13..D16

8er-Gruppen: D0..D7 und D9..D16

16er-Gruppe: D0..D16, darin ist die Leitung D8 nicht enthalten

Um auf einen digitalen Kanal oder eine Gruppe zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DIN* *DEVICE_DOUT* *DEVICE_DIO*	Kanal auf einen digitalen Eingang Kanal auf einen digitalen Ausgang Kanal auf eine umschaltbare digitale Leitung
.usIndexFirst	0, 4, 9, 13 für Gruppenkanäle bzw. 8 oder 17 für die Einzelkanäle	Nummer der ersten Leitung
.usIndexLast	3, 7, 12, 16 für Gruppenkanäle bzw. 8 oder 17 für die Einzelkanäle	Nummer der letzten Leitung
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*0* *IO_MODE_DIRECT*	Ohne Bedeutung (für DEVICE_DIN) Direkter Schreibzugriff (für DEVICE_DIO und DEVICE_DOUT)
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt exklusiv

Anmerkung

Wird ein DEVICE_DIO Kanal geöffnet, so sind die Pins beim Öffnen als Eingang konfiguriert.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_USHORT, dabei sind die Daten rechtsbündig angegeben. Alle Zugriffe auf Einzelkanäle erfolgen mit

- **max_read_channel_ushort** bzw.
- **max_write_channel_ushort**

Sonderdienste

- **max_channel_control**, Steuerbefehle CMD_DIR_INPUT, CMD_DIR_OUTPUT: Für Kanäle vom Typ DEVICE_DIO stehen die Befehle zum Umschalten der Richtung zur Verfügung. Es werden keine Daten übergeben.

10.14.2.4. Analoge Eingänge

Die externen Analog-Eingänge sind den Kanälen AIN-0 bis AIN-3 (*usIndexFirst* = 0 bis 3) zugeordnet. Weiterhin sind folgende interne Spannungskanäle verfügbar:

- 5V-Referenzspannung (*usIndexFirst* = 4)
- 3,3V-Versorgungsspannung (*usIndexFirst* = 5)
- Analog-Masse (*usIndexFirst* = 6)
- Temperatur (*usIndexFirst* = 7)

Um auf einen der analogen Eingänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AIN_DIFF*	Kanal auf einen analogen Eingang
.usIndexFirst	*0 bis 7 (s.o.)*	Nummer des ersten Eingangs
.usIndexLast	*0 bis 7*	Nummer des letzten Eingangs (beim X-DAD-4/i können keine internen Spannungskanäle mit externen Analog-Eingängen in einem Block gemischt werden, d.h. wenn usIndexFirst <4 ist, muss auch usIndexLast <4 sein.)
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*0*	Keine Bedeutung

Strukturelement	Werte	Bedeutung
.usFlags	*0*	Keine Bedeutung
	_CP_EXCLUSIVE	Der Zugriff erfolgt exklusiv
	_CP_HW_FORMAT	Es werden direkt die vom Wandler gelieferten Rohwerte zurückgegeben (das Flag *_CP_UNCORRECTED* hat keine Auswirkungen)
	_CP_UNCORRECTED	Die Werte werden keiner internen Korrektur unterzogen.
.usRange	**X-DAD-4/u:**	Eingangsbereich:
	RANGE_BIP_5V	Bipolar ±5V
	RANGE_BIP_10V	Bipolar ±10V
	RANGE_UPP_5V	Unipolar 0 ... 5V
	RANGE_UPP_10V	Unipolar 0 ... 10V
	X-DAD-4/i:	
	externe Analog-Eingänge:	
	RANGE_20MA	Unipolar 0 ... 20mA
	interne Spannungs-kanäle:	
	RANGE_BIP_5V	Bipolar ±5V
	RANGE_BIP_10V	Bipolar ±10V
	RANGE_UPP_5V	Unipolar 0 ...5 V
	RANGE_UPP_10V	Unipolar 0 ... 10V

Eingabedienst

Wenn das Flag *_CP_HW_FORMAT* gesetzt ist, ist der Datentyp des Kanals DATA_USHORT:

- **max_read_channel_ushort** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Der vom A/D-Wandler kommende Wert wird in folgendem Format (2er-Komplement) geliefert:

Unipolare Bereiche: 0 Volt = 0, max. Wert = FFFh

Bipolare Bereiche: min. Wert = 800h, max. Wert = 7FFh

Wenn das Flag *_CP_HW_FORMAT* nicht gesetzt ist, ist der Datentyp des Kanals DATA_LONG:

- **max_read_channel_long** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Der Temperaturfühler liefert bei +25°C eine Spannung von 608mV. Die Abhängigkeit von Spannung und Temperatur ist annähernd linear mit einer Steigung von 2mV/K.

10.14.2.5. Analoge Ausgänge

Um auf einen der analogen Ausgänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_AOUT*	Kanal auf einen analogen Ausgang
.usIndexFirst	*0 bis 3*	Nummer des ersten Ausgangs
.usIndexLast	*0 bis 3*	Nummer des letzten Ausgangs
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff: Ausgänge werden über den A/D-Wandler zurückgemessen (**nur bei X-DAD-4/u**)
	IO_MODE_RAM	(Softwaremäßiges) Zurücklesen des geschriebenen Wertes
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff: Ausgänge werden gesetzt
.usFlags	*0*	Normiertes Datenformat
	_CP_EXCLUSIVE	Der Zugriff erfolgt exklusiv
	_CP_HW_FORMAT	Die Ausgabedaten werden als Rohwerte im Format des Wandlers übergeben
.usRange	**X-DAD-4/u:**	Ausgangsbereich:
	RANGE_BIP_5V,	Bipolar ±5V
	RANGE_BIP_10V,	Bipolar ±10V
	RANGE_UPP_5V,	Unipolar 0 ... 5V
	RANGE_UPP_10V	Unipolar 0 ... 10V
	X-DAD-4/i:	
	RANGE_20MA	0 ... 20mA

Eingabe- und Ausgabedienst

Wenn das Flag *_CP_HW_FORMAT* gesetzt ist, ist der Datentyp des Kanals DATA_USHORT:

- **max_write_channel_ushort** (Einzelkanal)
- **max_write_channel_block** (Mehrkanal)
- **max_read_channel_ushort** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Wenn das Flag *_CP_HW_FORMAT* nicht gesetzt ist, ist der Datentyp des Kanals DATA_LONG:

- **max_write_channel_long** (Einzelkanal)
- **max_write_channel_block** (Mehrkanal)
- **max_read_channel_long** (Einzelkanal)

- **max_read_channel_block** (Mehrkanal)

Anmerkungen

Wenn das Flag *_CP_HW_FORMAT* gesetzt ist, wird folgendes Datenformat verwendet:

Unipolare Bereiche

Ausgabe = Max. Wert * geschriebener Wert / 4096

z.B. max_write_channel_ushort (.., 0x456) setzt im 0-10V Spannungsbereich den Ausgang auf 10*0x456 / 4096 = 2.7099 V.

Bipolare Bereiche

Ausgabe = Max. Wert * ((geschriebener Wert / 2048) –1)

z.B. max_write_channel_ushort (.., 0x456) setzt im ±10V Spannungsbereich den Ausgang auf 10* ((0x456 / 2048)-1) = -4.58 V.

10.14.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Pin	Funktion	Pin	Funktion
1	AIN-0+	21	AIN-2+
2	AIN-0-	22	AIN-2-
3	AIN-0 GND	23	AIN-2 GND
4	AIN-1+	24	AIN-3+
5	AIN-1-	25	AIN-3-
6	AIN-1 GND	26	AIN-3 GND
7	AOUT-0	27	AOUT-2
8	AOUT-0 GND	28	AOUT-2 GND
9	AOUT-1	29	AOUT-3
10	AOUT-1 GND	30	AOUT-3 GND
11	DIO-0	31	DIO-9
12	DIO-1	32	DIO-10
13	DIO-2	33	DIO-11
14	DIO-3	34	DIO-12
15	DIO-4	35	DIO-13
16	DIO-5	36	DIO-14
17	DIO-6	37	DIO-15
18	DIO-7	38	DIO-16
19	DIO-8	39	DIO-17
20	GND_0_8	40	GND_9_17

10.14.4. Besondere Eigenschaften

Parameter		Wert	Einheit
Analoge Eingänge (extern), Anzahl		4	-
Auflösung		12	Bit
Wandlungsgeschwindigkeit, min./typ./max.		6/7/10	µs
Acquisition Time, min./max.		3/5	µs
Eingangsspannungsbereiche,	Version /U	4[1]	-
	Version /i	2[2]	-
Common Mode Range,	Version /U	±10[3]	V
Überspannungsschutz,	Version /U	±40	V
Erholungszeit aus Stand-By-Mode		tbd	
Erholungszeit aus Power-Down-Mode		tbd	
Eingangsimpedanz (Common Mode/Diff.), Version U		>100/>100	MΩ
Analoge Ausgänge, Anzahl		4	-
Auflösung		12	Bit
Settle Time, max.		3	µs
Ausgangsspannungsbereiche,	Version /U	4[1]	-
	Version /i	2[2]	-
Digitale Ein-/Ausgänge, Anzahl		18	
4 Gruppen mit je 4 und 2 einzeln als Ein-/Ausgänge konfigurierbar			
Eingangsspannung (kompatibel mit 5V TTL und 5V, 3,3V und 2,5V CMOS)			
	log. 0	< 0,8	V
	log. 1	> 2,0	V
Input Leakage Current, max.		10	µA
Ausgangsspannung (kompatibel mit 5V TTL und 3,3V CMOS)			
	log. 0, max. (IOL = 8mA)	0,4	V
	log. 1, min. (IOH = -4mA)	2,4	V
Ausgangstrom, max.	Low Pegel	-12	mA
	High Pegel	+12	mA
Überspannungsfestigkeit der Eingänge		-0,5 .. +5,5	V
für Impulse < 10ns und < 200mA		–2,0 .. +7,0	V

[1] Eingangsbereiche 0 .. 5V, 0 .. 10V, ±5V, ±10V
[2] Eingangsbereiche 0 .. 20mA, 4 .. 20mA
[3] Common Mode Range abhängig von Eingangsspannung, typ. ±10V

Parameter	**Wert**	**Einheit**
Temperatur-Bereich, Betrieb	0 .. +70	°C
optional (bitte anfragen)	–40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht, X-DAD-4/i / X-DAD-4/U	9,9/10,3	g
Stromaufnahme (Ausgänge offen), 3,3V	10	mA
+12V (X-DAD-4/U / X-DAD-4/i)	40/80	mA
-12V (X-DAD-4/U / X-DAD-4/i)	30/65	mA

X-DIO-40/i

38 digitale Ein-/Ausgänge, 3 Timer, LED,
12-Kanal Interrupt Controler

10.15. X-DIO-40/i, X-DIO-40 und X-DIO-32

Inhaltsverzeichnis

10.15.1. Beschreibung

Das Modul X-DIO-40/i stellt 38 externe digitale TTL-Ein-/Ausgänge, 3 Counter/Timer, eine LED und einen 12-Kanal Interrupt-Controller mit Interrupt-Overrun Erkennung zur Verfügung.

6 Ein-/Ausgänge können einzeln als Ein- oder Ausgänge konfiguriert werden. Jeder ist Interrupt-fähig. Sie können zusätzliche Funktionen bei Timer 0 bzw. 1 übernehmen. Die übrigen 32 Ein-/Ausgänge sind in 4-er Gruppen organisiert, wobei jeweils nur die ganze Gruppe als Ein- oder Ausgang konfigurierbar ist. Dadurch können gegebenenfalls auch benachbarte Kanäle vom Öffnen eines Kanals betroffen sein. Wenn z.B. ein Kanal mit DIO-2 als Eingang geöffnet wird, sind damit auch DIO-0, DIO-1 und DIO-3 als Eingänge festgelegt. Jeweils 16 Eingänge können zeitgleich abgetastet werden.

Jeder der 12 Eingänge des Interrupt-Controllers kann einen Interrupt bei einer positiven, negativen oder bei beiden Flanken auslösen. Wenn ein Interrupt noch nicht bedient wurde und der nächste Interrupt am selben Eingang auftritt, wird ein Interrupt-Overrun gespeichert. Die 12 Interrupt-Eingänge sind mit den Ausgängen der 3 Timer, mit den Ein-/Ausgängen DIO-16, DIO-17, DIO-18, DIO-36, DIO-37 und DIO-38, und mit 3 frei wählbaren Eingängen verbunden. Per Software kann dabei je einer der Ein-/Ausgänge DIO-0..DIO-15 und DIO-20..DIO-35 angewählt werden. So kann z.B. auch ein Ausgangskanal Interrupt-fähig sein. Das Auftreten von Interrupts wird vom MDD durch Aufrufe von Anwender-Callback-Funktionen signalisiert (s.u.). Die Priorität der einzelnen Interrupt-Quellen bestimmt der Anwender durch die Reihenfolge, in der er Kanäle mit Callback-Funktionen öffnet: Der erste Kanal, der mit einer Callback-Funktion geöffnet wird, hat die höchste Priorität.

Die drei 16-Bit Timer sind 8254-kompatibel (Timer-Chip in PCs). Für jeden kann eine von 5 Betriebsarten gewählt werden. Von Timer 0 und 1 kann jeweils der Clock-Eingang CLK, der Gate-Eingang GATE und der Ausgang OUT einzeln an einen externen Ein- bzw. Ausgang gelegt werden. Alle 3 Timer sind Interrupt-fähig. Timer 2 kann auch als Watchdog geschaltet werden. Wenn er nicht rechtzeitig nachgetriggert wird, werden alle als Ausgänge geschalteten digitalen Kanäle hochohmig geschaltet. Außerdem verfügt das Modul über eine per Software schaltbare on-board LED.

Das Modul X-DIO-40 enthält keine Timer und keinen Interrupt-Controller. Das Modul X-DIO-32 ist eine Low-Cost-Variante ohne Timer und ohne Interrupt-Controller. Zusätzlich entfallen die 6 einzelnen konfigurierbaren Ein-/Ausgänge.

Das Modul ist in folgenden Bestückungsvarianten lieferbar:

Typ	Sub-typ	Modul-Version	Funktion
40	1	X-DIO-40/i	38 Digital IO, 3 Timer, Watchdog, 12 Interrupts
40	0	X-DIO-40	38 Digital IO
40	2	X-DIO-32	32 digital I/O

10.15.2. Modul-Device-Treiber

10.15.2.1. Installation

Der Modul-Device-Treiber für OsX hat die Programmnummer 8028h und den Dateinamen mxdio40.exe. Der Modul-Device-Treiber für Windows hat den Namen mxdio40.sys. Dies gilt sowohl für die X-DIO-40 als auch die X-DIO-32 Module. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8028, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8028

10.15.2.2. Kanaleigenschaftsstruktur CPS_XDIO40

Die CPS für das Modul hat den Namen CPS_XDIO40 bzw. CPS_XDIO40_A. Der Typ CPS_XDIO40.A wird ab MDD-Version 1.H unterstützt. Die Struktur wurde gegenüber CPS_XDIO40 um die Parameter ulTimeout, ulCallbackEvents und rcCallbackEvents erweitert.

10.15.2.3. Digitale Ein- und Ausgänge (umschaltbar)

Um auf die umschaltbaren digitalen Ein- und Ausgänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DIO*	Umschaltbarer DIN/DOUT

Strukturelement	Werte	Bedeutung
		Kanal. Zum Umschalten der Richtung stehen die Kanal-Steuerkommandos *CMD_DIR_OUTPUT* und *CMD_DIR_INPUT* zur Verfügung.
.usIndexFirst	siehe Tabelle unter 9.2.2.9.	Nummer des ersten DIOs
.usIndexLast	siehe Tabelle unter 9.2.2.9.	Nummer des letzten DIOs
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usMode	*0*	Reservierter Parameter

.rcCallbackEvents
(vom Typ:
`struct`
`{`
`USHORT ausPosEdge[3];`
`USHORT ausNegEdge[3]`
`}`
`XDIO40_CALLBACK_EVENTS`
)
		Dieser Parameter regelt, welche DIOs Interrupts auslösen und damit die Anwender-Callback-Funktion aufrufen sollen. Das Modul kann bei den DIOs-16, 17, 18, 36, 37 und 38 sowie bei 3 frei wählbaren DIOs Interrupts auslösen. Mit den Elementen ausPosEdge und ausNegEdge der Struktur kann festgelegt werden, welche DIOs Interrupts auslösen sollen. Die DIOs sind innerhalb der Arrays rechtsbündig angeordnet, d.h. ist der Kanal auf die DIOs-20 ... 35 geöffnet, so legt Bit 0 in ausPosEdge[0] fest, ob DIO-20 und Bit-15 ob DIO-35 einen Interrupt auslösen soll.

Anmerkung

Nach dem Öffnen ist der Kanal zunächst als DIN-Kanal konfiguriert.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist abhängig von den Werten in *.usIndexFirst* und *.usIndexLast* (siehe Tabelle). Der Zugriff auf das Device erfolgt, je nach Datentyp mit:

Datentyp DATA_UCHAR:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

oder Datentyp DATA_USHORT:

- **max_write_channel_ushort**
- **max_read_channel_ushort**

Sonderdienst

- **max_channel_control,** Steuerbefehl CMD_DIR_INPUT, CMD_DIR_OUTPUT: Umschalten zwischen DIN und DOUT. Dem Dienst werden keine Daten übergeben.

Callback-Funktion (nur für X-DIO-40/i)

Eine Callback-Funktion kann nur bei solchen Kanälen angegeben werden, bei denen in rcCallbackEvents zumindest ein Bit = 1 gesetzt ist. Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen wenn eine positive bzw. negative Flanke an einem der dafür konfigurierten DIOs auftritt (Interrupt). Die Callback-Funktion bekommt eine Struktur vom Typ XDIO40_INTERRUPT übergeben.

```
struct
{
     USHORT ausPending[3];
     USHORT ausOverrun[3];
}XDIO40_INTERUPT;
```

Das Element ausPending kennzeichnet, welche DIOs einen Interrupt ausgelöst haben. Das Element ausOverrun kennzeichnet, bei welchen DIOs ein Interrupt-Überlauf aufgetreten ist. Die Daten sind dabei rechtsbündig angeordnet: wurde z.B. ein Kanal zu den DIOs 4 ... 8 geöffnet, zeigt das unterste Bit in ausPending[0] an, dass DIO-4 einen Interrupt verursacht hat. Nur ausPending [0] und ausOverrun [0] werden derzeit verwendet. Die restlichen Array Elemente müssen = 0 gesetzt werden.

10.15.2.4. Digitale Ausgänge

Um auf die digitale Ausgänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DOUT*	Kanal zu einem digitalen Ausgang
.usIndexFirst	siehe Tabelle unter 9.2.2.9.	Nummer des ersten Ausgangs
.usIndexLast	siehe Tabelle unter 9.2.2.9.	Nummer des letzten Ausgangs
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet
.usMode	*0*	Reservierter Parameter

.rcCallbackEvents
(vom Typ:
`struct`
`{`
`USHORT ausPosEdge[3];`
`USHORT ausNegEdge[3]`
`}XDIO40_CALLBACK_EVENTS`
)
		Dieser Parameter regelt, welche DOUTs Interrupts auslösen und damit die Anwender-Callback-Funktion aufrufen sollen. Das Modul kann bei den DOUTs-16, 17, 18, 36, 37 und 38 sowie bei 3 frei wählbaren DOUTs Interrupts auslösen. Mit den Elementen ausPosEdge und ausNegEdge der Struktur kann festgelegt werden, welche DOUTs Interrupts auslösen sollen. Die DOUTs sind innerhalb der Arrays rechtsbündig angeordnet, d.h. ist der Kanal auf die DOUTs-20 ... 35 geöffnet, so legt Bit 0 in ausPosEdge[0] fest, ob DOUT-20 und Bit-15 ob DOUT-35 einen Interrupt auslösen soll.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist abhängig von den Werten in *.usIndexFirst* und *.usIndexLast* (siehe Tabelle), der Zugriff erfolgt, je nach Datentyp mit:

Datentyp DATA_UCHAR:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

oder Datentyp DATA_USHORT:

- **max_write_channel_ushort**
- **max_read_channel_ushort**

Callback-Funktion (nur für X-DIO-40/i)

Eine Callback-Funktion kann nur bei solchen Kanälen angegeben werden, bei denen in rcCallbackEvents zumindest ein Bit = 1 gesetzt wird. Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen wenn eine positive bzw. negative Flanke an einem der dafür konfigurierten DOUTs auftritt, d.h. wenn der Ausgang geschaltet wird. Die Callback-Funktion bekommt eine Struktur vom Typ XDIO40_INTERRUPT übergeben.

```
struct
{
     USHORT ausPending[3];
     USHORT ausOverrun[3];
}XDIO40_INTERUPT;
```

Das Element ausPending kennzeichnet, welche DOUTs einen Interrupt ausgelöst haben. Das Element ausOverrun kennzeichnet, bei welchen DOUTs ein Interrupt-Überlauf aufgetreten ist. Die Daten sind dabei rechtsbündig angeordnet: wurde z.B. ein Kanal zu den DOUTs 4 ... 8 geöffnet, zeigt das unterste Bit in ausPending[0] an, dass DOUT-4 einen Interrupt verursacht hat. Nur ausPending [0] und ausOverrun [0] werden derzeit verwendet. Die restlichen Array Elemente müssen = 0 gesetzt werden.

10.15.2.5. Digitale Eingänge

Um auf die digitalen Eingänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DIN*	Kanal zu einem digitalen Eingang
.usIndexFirst	siehe Tabelle unter 9.2.2.9.	Nummer des ersten Eingangs
.usIndexLast	siehe Tabelle unter 9.2.2.9.	Nummer des letzten Eingangs
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*0*	Kein Schreibzugriff
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet
.usMode	*0*	Reservierter Parameter

.rcCallbackEvents
(vom Typ:
`struct`
`{`
`USHORT ausPosEdge[3];`
`USHORT ausNegEdge[3]`
`}XDIO40_CALLBACK_EVENTS`
)
		Dieser Parameter regelt, welche DINs Interrupts auslösen und damit die Anwender-Callback-Funktion aufrufen sollen. Das Modul kann bei den DINs-16, 17, 18, 36, 37 und 38 sowie bei 3 frei wählbaren DINs Interrupts auslösen. Mit den Elementen ausPosEdge und ausNegEdge der Struktur kann festgelegt werden, welche DINs Interrupts auslösen sollen. Die DINs sind innerhalb der Arrays rechtsbündig angeordnet, d.h. ist der Kanal auf die DINs-20 ... 35 geöffnet, so legt Bit 0 in ausPosEdge[0] fest, ob DIN-20 und Bit-15 ob DIN-35 einen Interrupt auslösen soll.

Eingabedienst

Der Datentyp des Kanals ist abhängig von den Werten in *.usIndexFirst* und *.usIndexLast* (siehe Tabelle). Der Zugriff auf das Device erfolgt, je nach Datentyp mit:

Datentyp DATA_UCHAR:

- **max_read_channel_uchar**

oder Datentyp DATA_USHORT:

- **max_read_channel_ushort**

Callback-Funktion (nur für X-DIO-40/i)

Eine Callback-Funktion kann nur bei solchen Kanälen angegeben werden, bei denen in rcCallbackEvents zumindest ein Bit = 1 gesetzt ist. Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen wenn eine positive bzw. negative Flanke an einem der dafür konfigurierten DINs auftritt (Interrupt). Die Callback-Funktion bekommt eine Struktur vom Typ XDIO40_INTERRUPT übergeben.

```
struct
{
     USHORT ausPending[3];
     USHORT ausOverrun[3];
}XDIO40_INTERUPT;
```

Das Element ausPending kennzeichnet, welche DINs einen Interrupt ausgelöst haben. Das Element ausOverrun kennzeichnet, bei welchen DINs ein Interrupt-Überlauf aufgetreten ist. Die Daten sind dabei rechtsbündig angeordnet: wurde z.B. ein Kanal zu den DINs 4 ... 8 geöffnet, zeigt das unterste Bit in ausPending[0] an, dass DIN-4 einen Interrupt verursacht hat. Nur ausPending [0] und ausOverrun [0] werden derzeit verwendet. Die restlichen Array Elemente müssen = 0 gesetzt werden.

10.15.2.6. Timer (nur für X-DIO-40/i)

Um auf einen der 3 Timer des Moduls zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TIMER*	Timer
.usIndexFirst	*0 ... 2*	Nummer des Timers
.usIndexLast	*.usIndexFirst*	Nummer desTimers
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff

Strukturelement	Werte	Bedeutung
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags		

	_CP_EXCLUSIVE	Muss gesetzt sein. Es kann nur ein Kanal pro Timer geöffnet werden.

		Eines der beiden folgenden Flags muss gesetzt sein:
	_XDIO40_CLOCK_1MHZ	Interner Takt (1 MHz) an CLOCK.
	_XDIO40_CLOCK_DIO	Externer Takt an CLOCK. Nur für Timer-0 und -1 verwendbar! CLOCK wird mit DIO-16 (Timer-0) bzw. DIO-36 (Timer-1) verbunden.

		Eines der 3 folgenden Flags muss gesetzt sein:
	_XDIO40_GATE_SOFTWARE	GATE ist per Software schaltbar.
	_XDIO40_GATE_T2OUT	Nur für Timer-0 und -1 verwendbar! GATE wird mit OUT von Timer-2 verbunden.
	_XDIO40_GATE_DIO	Nur für Timer-0 und -1 verwendbar! GATE wird mit DIO-17 (Timer-0) bzw. DIO-37 (Timer-1) verbunden.

	_XDIO40_OUT_DIO	Nur für Timer-0 und -1 verwendbar! OUT wird mit DIO-18 (Timer-0) bzw. DIO-38 (Timer-1) verbunden.

	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar!
		Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet
.usMode		Aus den folgenden zwei Gruppen muss jeweils genau ein Flag gesetzt sein!

	_XDIO40_TIMER_MODE_0	Interrupt bei Zählende
	_XDIO40_TIMER_MODE_1	Programmierbares Monoflop
	_XDIO40_TIMER_MODE_2	Ratengenerator
	_XDIO40_TIMER_MODE_3	Rechteckgenerator
	_XDIO40_TIMER_MODE_4	Software-getriggerter Impuls
	_XDIO40_TIMER_MODE_5	Hardware-getriggerter Impuls

	_XDIO40_TIMER_MODE_BCD	Der Timer zählt im BCD-Modus
	_XDIO40_TIMER_MODE_ BINARY	Der Timer zählt im Binär-Modus

Strukturelement	Werte	Bedeutung
.ulCallbackEvents	*XDIO40_EVENT_POS_EDGE*	Eine positive Flanke an OUT löst einen Interrupt aus.
	XDIO40_EVENT_NEG_EDGE	Eine negative Flanke an OUT löst einen Interrupt aus.

Anmerkung

Nach dem Öffnen eines Kanals muss der Timer mit dem Ausgabedienst gestartet werden. Soll ein DIO als CLOCK, GATE oder OUT verwendet werden, so kann dieses Signal nicht als „normaler“ DIO verwendet werden und umgekehrt.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_USHORT:

- **max_write_channel_ushort**
- **max_read_channel_ushort**

Sonderdienst

- **max_channel_control,** Steuerbefehl CTRL_XDIO40_SET_GATE:
  Ist das Gate per Software schaltbar, kann dies mit dem Steuerkommando erfolgen. Dem Dienst wird ein ULONG-Wert übergeben, in den je nach gewünschtem Zustand eine 0 oder 1 eingetragen werden muss.

Callback-Funktion

Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen, wenn eine positive bzw. negative Flanke am OUT des Timers auftritt.
Die Callback-Funktion bekommt einen ULONG Wert übergeben, der aus einer ODER-Verknüpfung der folgenden Konstanten bestehen kann:

- XDIO40_INTERRUPT_PENDING: Interrupt aufgetreten
  XDIO40_INTERRUPT_OVERRUN: Interrupt-Überlauf aufgetreten (d.h. bevor ein Interrupt bedient werden konnte ist bereits ein weiterer Interrupt aufgetreten)

Timer Mode

Die Timer Modi 0 ... 5 entsprechen den Modi des programmierbaren Intervall-Timers 82C54. Der Timer arbeitet in allen Modi als Abwärtszähler.

_XDIO40_TIMER_MODE_0 (Interrupt bei Zählende)

Nach dem Beschreiben des Timers, wird der Timer mit dem nächsten CLK-Impuls geladen. Der OUT-Pin ist ab Zählbeginn auf 0. Erreicht der Zähler den Wert 0, so steigt OUT auf 1. GATE=1 aktiviert den Zähler, GATE=0 deaktiviert ihn.

_XDIO40_TIMER_MODE_1 (programmierbares Monoflop)

Nach dem Beschreiben eines Timers liegt OUT zunächst auf 1. Ein Trigger am GATE (von 0 auf 1) lädt den Zähler. Beim nächsten CLK-Impuls fällt OUT auf 0 und bleibt auf diesem Pegel, bis der Zähler den Wert 0 erreicht hat. OUT steigt dann wieder auf 1. Erst einen CLK-Impuls nach dem nächsten Trigger fällt OUT erneut auf 0.

Tritt während des Zählvorgangs ein neuer Trigger-Impuls auf, wird der Zähler erneut mit dem Anfangswert geladen. Ein Schreibvorgang hat solange keine Auswirkungen bis ein neuer Trigger-Impuls auftritt.

_XDIO40_TIMER_MODE_2 (Ratengenerator) (periodisch)

Nach dem Beschreiben des Timers beginnt der Zähler mit dem nächsten CLK-Impuls mit der Zählung. Erreicht der Zähler den Wert 1, so fällt OUT für einen CLK-Impuls auf 0. Anschließend wird Anfangszählerwert automatisch neu geladen und der Zählvorgang beginnt erneut. GATE=1 aktiviert, GATE=0 deaktiviert den Zähler. Fällt GATE während des Zählvorgangs auf 0 und steigt später wieder auf 1, wird beim Übergang von 0 auf 1 der Zähler neu geladen und der Zählvorgang gestartet. Ein Schreibbefehl lädt und startet den Zähler ebenfalls erneut.

_XDIO40_TIMER_MODE_3 (Rechteckgenerator) (periodisch)

In diesem Modus wird ein periodisches Rechtecksignal erzeugt. Beim Start liegt OUT auf 1, ist der Zähler zur Hälfte abgelaufen, fällt OUT auf 0. Erreicht der Zähler den Wert 0, steigt OUT auf 1 und der Zähler wird erneut geladen. GATE=1 aktiviert, GATE=0 deaktiviert den Zähler. Fällt GATE auf 0, während auch OUT auf 0 liegt, so steigt OUT sofort auf 1. Ein Trigger an GATE (von 0 auf 1) lädt und startet den Zähler. Das Beschreiben des Timers beeinflusst den gegenwärtigen Zählvorgang nicht. Erst nach dem Ende des aktuellen Halbzyklus wird der Wert geladen.

_XDIO40_TIMER_MODE_4 (Sofware-getriggerter Impuls)

Zu Beginn liegt OUT auf 1. Hat der Zähler 0 erreicht, so fällt OUT für einen CLK-Impuls auf 0 und steigt dann wieder auf 1. GATE=1 aktiviert, GATE=0 deaktiviert den Zähler. Nach dem Beschreiben des Timers wird der Zähler mit dem nächsten CLK-Impuls geladen und mit dem nächsten CLK-Imouls gestartet. Erfolgt während des Zählvorgangs ein Schreibzugriff, wird der Wert beim nächsten CLK-Impuls geladen und der Zählvorgang mit diesem Wert fortgesetzt.

_XDIO40_TIMER_MODE_5 (Hardware-getriggerter Impuls)

Die Impulsform stimmt mit Mode 4 überein. Die Triggerung erfolgt jedoch durch einen Übergang von 0 auf 1 von GATE. Hierdurch wird durch den nächsten CLK-Impuls der Zähler geladen und gestartet. Ist der Zählerwert 0 erreicht, so fällt OUT für einen CLK-Impuls auf 0. Tritt während des Zählvorgang ein Trigger-Impuls auf, wird der Anfangswert erneut geladen und der Zählvorgang mit diesem Wert fortgesetzt. Ein Schreibvorgang wirkt sich auf den aktuellen Zählvorgang nicht aus. Erst beim nächsten Trigger-Impuls wird der Wert übernommen.

10.15.2.7. Watchdog (nur X-DIO-40/i)

Der Watchdog wird über den Timer-2 des Moduls realisiert, d.h. wenn Timer-2 bereits verwendet wird, kann kein Watchdog mehr benutzt werden (und umgekehrt).

Um auf den Watchdog zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_WATCHDOG*	Kanal auf einen Watchdog
.usIndexFirst	*0*	Nummer des Watchdog
.usIndexLast	*0*	Nummer des Watchdog
.ulTimeout	*1 ... 260*	Timeout für Watchog in ms
.usReadMode	*0*	Reservierter Parameter
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt exklusiv
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.ulCallbackEvents	*XDIO40_EVENT_WATCHDOG*	Ist der Watchdog abgelaufen, ruft er die Callback-Funktion auf.

Ausgabedienst

Der Datentyp ist DATA_VOID. Der Zugriff erfolgt mit:

- **max_trigger_channel**

Anmerkung

Damit der Watchdog nicht aktiv wird, muss er zumindest einmal während der in der CPS angegebenen Timeout-Zeit nachgetriggert werden. Bleibt das aus, so dass der

Watchdog-Timer abläuft, werden alle Ausgänge hochohmig geschaltet. Das Deaktivieren des Watchdog erfolgt beim Schließen des Kanals.

Callback-Funktion

Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen wenn der Watchdog nicht rechtzeitig nachgetriggert wurde. Die Callback-Funktion bekommt bei ihrem Aufruf keine Parameter übergeben.

Sonderdienst

- **max_channel_info,** Infotyp INFO_DEVICE: Der Zustand des Watchdog kann jederzeit gelesen werden. Die Funktion liefert den Status des Watchdog als ULONG-Wert zurück, wobei 0 bedeutet, dass der Watchdog nicht abgelaufen ist, 1 bedeutet, dass der Watchdog abgelaufen ist.
- **max_channel_control,** Steuerbefehl CMD_START: Mit diesem Sonderdienst kann der Watchdog anschließend wieder gestartet werden. Dem Dienst werden keine Daten übergeben.

10.15.2.8. LED

Das Modul bietet eine LED, die über folgende CPS angesprochen werden kann:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_LED*	Kanal zu einer LED
.usIndexFirst	*0*	Nummer der LED
.usIndexLast	*0*	Nummer der LED
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0* *_CP_EXCLUSIVE*	Keine Bedeutung Der Zugriff auf die LED erfolgt exklusiv
.usMode	*0*	Reservierter Parameter

Eingabe- und Ausgabedienst

Um die LED ein- bzw. auszuschalten muss eine 1 bzw. 0 in den Kanal geschrieben werden. Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_uchar**
- **max_read_channel_uchar**

10.15.2.9. Device-Index und Datentypen für DIN, DOUT und DIO-Kanäle

In der folgenden Tabelle sind für DIN, DOUT und DIO-Kanäle die möglichen Kombinationen von.*usIndexFirst* und *.usIndexLast* sowie der Datentyp des Kanals aufgelistet. Da die Ein-/Ausgänge jeweils in Gruppen organisiert sind, kann ein Kanal nur über eine oder mehrere Gruppen geöffnet werden.

.usIndexFirst	.usIndexLast	Datentyp	Anmerkung
0	3	DATA_UCHAR	
0	7	DATA_UCHAR	
0	11	DATA_USHORT	
0	15	DATA_USHORT	
4	7	DATA_UCHAR	
4	11	DATA_UCHAR	
4	15	DATA_USHORT	
8	11	DATA_UCHAR	
8	15	DATA_UCHAR	
12	15	DATA_UCHAR	
16	16	DATA_UCHAR	nicht bei X-DIO-32
17	17	DATA_UCHAR	nicht bei X-DIO-32
18	18	DATA_UCHAR	nicht bei X-DIO-32
20	23	DATA_UCHAR	
20	27	DATA_UCHAR	
20	31	DATA_USHORT	
20	35	DATA_USHORT	
24	27	DATA_UCHAR	
24	31	DATA_UCHAR	
24	35	DATA_USHORT	
28	31	DATA_UCHAR	
28	35	DATA_UCHAR	
32	35	DATA_UCHAR	
36	36	DATA_UCHAR	nicht bei X-DIO-32
37	37	DATA_UCHAR	nicht bei X-DIO-32
38	38	DATA_UCHAR	nicht bei X-DIO-32

10.15.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Kanal	Pin	Kanal	Pin
DIO-0	1	DIO-20	21
DIO-1	2	DIO-21	22
DIO-2	3	DIO-22	23
DIO-3	4	DIO-23	24
DIO-4	5	DIO-24	25
DIO-5	6	DIO-25	26
DIO-6	7	DIO-26	27
DIO-7	8	DIO-27	28
DIO-8	9	DIO-28	29
DIO-9	10	DIO-29	30
DIO-10	11	DIO-30	31
DIO-11	12	DIO-31	32
DIO-12	13	DIO-32	33
DIO-13	14	DIO-33	34
DIO-14	15	DIO-34	35
DIO-15	16	DIO-35	36
DIO-16 bzw. Timer 0 Clock	17 [1]	DIO-36 bzw. Timer 1 Clock	37 [1]
DIO-17 bzw. Timer 0 Gate	18 [1]	DIO-37 bzw. Timer 1 Gate	38 [1]
DIO-18 bzw. Timer 0 Output	19 [1]	DIO-38 bzw. Timer 1 Output	39 [1]
GND	20	GND	40

[1] bei X-DIO-32 nicht angeschlossen

10.15.4. Besondere Eigenschaften

Parameter	**Wert**	**Einheit**
Anzahl externer digitaler Ein- bzw. Ausgänge		
(X-DIO-40 und X-DIO-40/i)	38	-
(X-DIO-32)	32	-
Eingangsspannung (andere Standards auf Anfrage)		
(kompatibel mit 5V TTL und 5V, 3,3V und 2,5V CMOS)		
log. 0	< 0,8	V
log. 1	> 2,0	V
Input Leakage Current, max.	10	µA
Ausgangsspannung		
(kompatibel mit 5V TTL und 3,3V CMOS)		
log. 0, max. (IOL = 8mA)	0,4	V
log. 1, min. (IOH = -4mA)	2,4	V
Ausgangstrom log. 0, min./typ.	12/36	mA
log. 1, min./typ.	-12/-24	mA
Überspannungsfestigkeit der Eingänge	-0,5 .. +5,5	V
für Impulse < 10 ns und < 200mA	-2,0 .. +7,0	V
Watchdog-Timer (nur bei X-DIO-40/i), programmierbar	80 .. 270	ms
Timer (nur bei X-DIO-40/i), Anzahl	3	-
Auflösung	16	Bit
Betriebsarten (z.B. Rechteckgenerator, Ratengenerator, Impulsgenerator)	6	-
Temperatur-Bereich, Betrieb	0 .. +70	°C
optional (bitte anfragen)	-40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	8,85	g
Stromaufnahme (3,3V) min./max.	455/500	mA
(die X-Bus Spannungen ±12V werden nicht benötigt)		

Notizen

X-DPM-1i

PROFIBUS-DP Master

in Entwicklung

10.16. X-DPM-1i

Inhaltsverzeichnis

10.16.1. Beschreibung

Das Modul X-DPM-1i (X-Bus-Modul-Typ 90, Subtyp 1) ist ein intelligentes PROFIBUS-DP-Master Modul. Alle Baudraten bis zu 12 MBaud werden unterstützt. Als lokale Intelligenz ist das Modul mit einem C165 Mikrocontroller bestückt. Das PROFIBUS –Protokoll wird in einem ASIC (ASPC2) abgewickelt. Es verfügt über eine galvanisch getrennte RS-485 Schnittstelle.

10.16.1.1. Erstellen der Master-Konfiguration

Um als Master in einem PROFIBUS Netzwerk arbeiten zu können, ist zunächst die Konfiguration des Systems erforderlich. Dazu wird die Software COM-Profibus von Siemens verwendet. Diese kann über SORCUS bezogen werden. Bei der Installation von COM-Profibus müssen unter Optionen Memory-Card-Treiber deaktiviert werden.

Wenn Sie dieses Programm installiert haben, müssen Sie zunächst die mitgelieferte GSD-Datei (X-DPM-1.GSD) und die BMP-Dateien in die entsprechenden Verzeichnisse von COM-Profibus kopieren. Dadurch steht das X-DPM-1i dort symbolisch zur Auswahl. Die GSD-Datei beschreibt die PROFIBUS-Eigenschaften des Masters.

Eine PROFIBUS-Konfiguration wird durch das (grafische) Zusammenstellen aller im Netzwerk vorhandenen Teilnehmer erstellt. Dabei werden dem Master und allen angeschlossenen Slaves Adressen zugeteilt und sonstige Eigenschaften, wie z.B. die verwendete Baudrate, festgelegt.

Das fertig konfigurierte PROFIBUS-System muss dann als 2bf-Datei exportiert werden. Diese Datei muss einmalig in den auf dem Modul vorhandenen Flash-Speicher geladen werden. Das kann mit Hilfe von SNW32 oder durch einen MDD-Dienst geschehen.

10.16.2. Modul-Device-Treiber

Die Beschreibung des Modul-Device-Treibers lag bei Drucklegung dieses Handbuchs noch nicht vor. Sobald diese fertig ist, wird sie auf der SORCUS-Homepage bereitgestellt.

10.16.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Pin	Funktion	Bedeutung	
1	DPPE	Optional	
2	DP5V	+5 Volt, isoliert	
3	-	n.c.	
4	-	n.c.	
5	DPB	DPB, isoliert	
6	DPA	DPA, isoliert	
7	DPRTS	Request To Send, isoliert	
8	-	n.c.	
9	DPGND	Ground, isoliert	
10	-	n.c.	
11	RCV	Receive-Signal	RS-232 Schnittstelle
12	RTS	Request to Send-Signal	RS-232 Schnittstelle
13	TMT	Transmit-Signal	RS-232 Schnittstelle
14	CTS	Clear to Send-Signal	RS-232 Schnittstelle
15	GND	GND	
16	LED2	LED2	
17	LED3	LED3	
18..20		reserviert	
21..40		reserviert	

n.c.: not connected

10.16.4. Besondere Eigenschaften

Parameter	**Wert**	**Einheit**
Anzahl Kanäle	1	-
Physikalischer Anschluss	PROFIBUS RS-485	
Galvanische Trennung	Ja	
Übertragungsrate, max.	12	MBaud
PROFIBUS-Controller	Siemens ASPC2	-
Überspannungsfestigkeit der Ein-/Ausgänge	Tbd	V
Temperatur-Bereich, Betrieb	-40..85	°C
Stromaufnahme	Tbd	mA

X-DPS-1i
X-DPS-2i

Modul mit 2 PROFIBUS-Slave Kontrollern

10.17. X-DPS-1i, X-DPS-2i

Inhaltsverzeichnis

10.17.1. Beschreibung

Mit dem Modul X-DPS-2i (X-Bus-Modul-Typ 91, Subtyp 1) können aus einem X-Bus-System zwei intelligente modulare Profibus-Slaves werden. Jeder Slave ist Sync- und Freeze-fähig. Beide Kanäle sind voneinander unabhängig und gegeneinander sowie zum X-Bus isoliert. Für jeden Kanal ist ein Profibus-Controller vorhanden, der die Abwicklung des Profibus-DP Protokolls übernimmt. Das Modul ist für Übertragungsraten bis 12MBit/s geeignet, wobei die tatsächliche Baudrate automatisch erkannt wird.

Die Variante X-DPS-1i (Subtyp 0) stellt nur einen Kanal zur Verfügung. Alle anderen Eigenschaften sind identisch zum X-DPS-2i.

10.17.1.1. Einbinden in die Master-Konfiguration

Dem Master wird mit Hilfe grafischer Tools wie z.B. COM-Profibus oder Step 7 (beides von Siemens) bekannt gegeben, mit welchen Slaves er zu kommunizieren hat und welche Eigenschaften die einzelnen Slaves haben. Um die Einbindung des X-Bus-Slaves in das Master-System zu vollziehen, sind zunächst die mitgelieferte GSD-Datei (X-DPS-2.GSD) und die BMP-Dateien in die entsprechenden Verzeichnisse der Konfigurations-Software zu kopieren. Dadurch steht das X-DPS-2i im Konfigurationsprogramm symbolisch zur Auswahl.

Die GSD-Datei beschreibt die Eigenschaften des Slaves. Bei den meisten Profibus-Slaves ist darin direkt festgelegt, wie viele Ein- und Ausgänge der Slave hat, und wie diese beschaffen sind. Die GSD-Datei des X-DPS-2i stellt alle mit dem Modul möglichen Konfigurationen zur Auswahl, da das Modul ein modularer Slave ist.

Wie viele Daten der X-DPS-2i Slave in einem konkreten Anwendungsfall tatsächlich auf dem Bus bereitstellen soll, muss im Master-Konfigurationsprogramm konfiguriert werden. Dazu werden aus den (durch die GSD-Datei) vorgegebenen Konfigurationsmöglichkeiten passende Baugruppen ausgewählt, z.B. eine Baugruppe mit 16 digitalen Eingängen und eine mit 8 analogen Ausgängen. Eine Baugruppe entspricht einem Subkanal, über den Daten vom Master zum Slave oder vom Slave zum Master übertragen werden sollen. Sie legt fest, ob es sich bei dem Kanal um Ein- oder Ausgänge (Bezeichnung immer aus der Sicht des Masters) handelt, wie viele Daten er umfasst und wie die Daten organisiert sind. Weiterhin kann angegeben werden, dass die Daten der Baugruppe konsistent übertragen werden. Jede Baugruppe ist durch eine Kennung (auch ID) gekennzeichnet. Dieses Kennungs-Byte beschreibt den zugehörigen Kanal eindeutig.

Beispiel:

Der Slave soll 2 analoge (16-Bit-) Werte an den Master geben (d.h. Ausgänge) und 1 Byte vom Master empfangen (d.h. Eingänge), mit denen er z.B. 8 digitale Ausgänge auf einem anderen X-Bus-Modul setzt. In diesem Fall werden zwei Baugruppen benötigt. Die Konfiguration der beiden Baugruppen ergibt für Baugruppe 0 die Kennung 97 und für Baugruppe 1 die Kennung 16 (jeweils ohne Konsistenz).

Alle in der Masterkonfiguration für den Slave eingegebenen Baugruppen-Kennungen werden auch auf der Slave-Seite benötigt. Diese sollte man sich bei der Konfiguration des Mastersystems notieren. Zusätzlich wird für den Slave die im Master-System konfigurierte Stationsadresse des Slaves benötigt.

10.17.1.2. Erstellen eines Programms auf der Slave-Seite

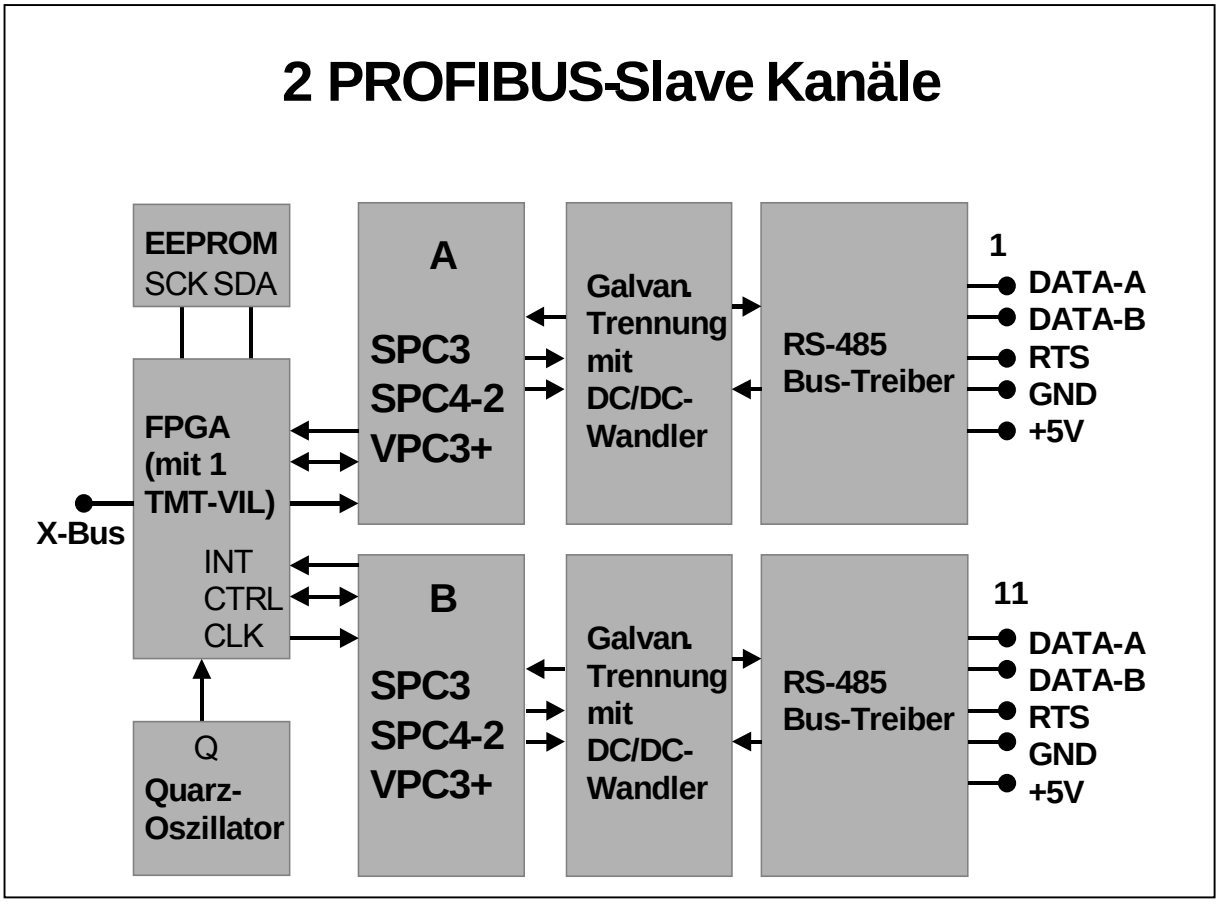
Das Anwenderprogramm muss unter Verwendung des Modul Device Treibers (s.u.) den Slave zunächst konfigurieren, d.h. die Stationsadresse einstellen und festlegen wie viele Ein- und Ausgänge der Slave auf dem Profibus bereitstellen soll und welcher Art diese sind. Dabei ist es entscheidend, dass hier dieselben Kennungsbytes angegeben werden, die auch auf der Seite des Masters für diesen Slave eingetragen wurden. Die Reihenfolge muss ebenfalls dieselbe sein.

Werden auf der Master und Slave-Seite unterschiedliche Konfigurationen eingestellt, erkennt der Master den Slave nicht.

Beispiel:

Auf der Slave-Seite ist das X-DPS-2i über den Modul Device Treiber zu konfigurieren, indem die Stationsadresse eingestellt wird und für die Slave-Konfiguration die beiden Kennungen 97 und 16 für die beiden Baugruppen (s.o.) zu übergeben sind. Anschließend hat das Programm die Aufgabe, die Eingänge (aus Master-Sicht) mit Daten zu versorgen und die vom Master empfangenen Ausgangsdaten an die Peripherie weiterzugeben. Es müssten also z.B. zwei Kanäle eines Analog-Eingangsmoduls gelesen werden und die gemessenen Werte auf den Bus geschrieben werden. Das Byte mit den Informationen für die digitalen Ausgänge müsste vom X-DPS-2i ausgelesen werden um diese Werte über ein digitales Ausgangsmodul auszugeben.

10.17.2. Blockschaltbild



10.17.3. Modul-Device-Treiber

10.17.3.1. Installation

Der Modul-Device-Treiber für OsX hat die Programmnummer 805Bh und den Dateinamen mxdps2.exe. Der Modul-Device-Treiber für Windows hat den Namen mxdps2.sys. Die beiden Profibus-Kanäle des Moduls können entweder durch einen MDD oder zwei unabhängige MDDs auf unterschiedlichen CPUs genutzt werden. Es ist nicht möglich einen der beiden Kanäle mit mehreren MDDs zu benutzen. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x805B, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=805B

10.17.3.2. Kanaleigenschaftsstruktur CPS_XDPS

Die CPS für das Modul hat den Namen CPS_XDPS.

10.17.3.3. Konfiguration

Bevor ein Slave an der Bus-Kommunikation teilnehmen kann, muss er zunächst konfiguriert werden. Dadurch wird dem Slave mitgeteilt, welche Daten er mit dem Master auf dem Profibus austauschen soll: die Art und Anzahl der Ein- und Ausgänge. Dazu ist ein Kanal mit folgender CPS zu öffnen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Steuerkanal für die Konfiguration des Slaves
.usChannel	*0 oder 1*	Schnittstelle auf dem Modul (bei X-DPS-1i: 0)
.usFlags	*_CP_EXCLUSIVE*	Muss gesetzt sein: für die Konfiguration kann nur ein Kanal exklusiv geöffnet werden.
	_XDPS_SET_ADDR_ SUPPORT	Der Slave unterstützt die dynamische Zuweisung einer Stationsadresse durch den Master
.usMode	*0*	Betriebsart (reserviert)
.usAddress	*3 bis 123*	Profibus-Stationsadresse des Slaves
.usTimeout		reserviert (=0 setzen)

Eingabe- und Ausgabedienst

Mit dem Ausgabedienst wird dem Modul die Slave-Konfiguration übergeben, d.h. es wird konfiguriert, wie viele Ein-und Ausgabedaten der Slave dem Bus zur Verfügung stellen bzw. vom Bus empfangen soll.

Der Datentyp ist DATA_UCHAR. Ein Block besteht aus 1 bis 244 Bytes.

- **max_write_channel_block**
- **max_read_channel_block** (Zurücklesen der Daten)

Jedes Byte im übergebenen Konfigurationsdaten-Array konfiguriert dabei einen Subkanal auf dem Slave. *ulSize* gibt somit die Anzahl der Subkanäle an.

Der Inhalt des Konfigurationsdaten-Arrays entspricht exakt den Kennungen, die für die Subkanäle dieses Slaves (gekennzeichnet durch die in der CPS angegebene Stationsadresse) bei der Konfiguration des Profibus-Systems im Master anzugeben sind. In der Regel ist jedem Subkanal ein Byte zugeordnet, das Auskunft über die Art und Anzahl der Ein- bzw. Ausgänge dieses Subkanals gibt. Bei Verwendung des speziellen Kennungsformats (s. folgende Tab.) gehören zu einem Subkanal mehrere Konfigurationsbytes.

Die Kodierung der einzelnen Bytes entspricht der Profibus Norm:

Bit	Wert	Bedeutung
0..3	0..15	Länge der Daten + 1, d.h. der Wert 0 entspricht Länge 1
4..5	00	spezielles Kennungsformat[14]
	01	Eingabe-Subkanal (Eingangsdaten vom Prozess zum Slave)
	10	Ausgabe-Subkanal (Ausgangsdaten vom Slave zum Prozess)
	11	Ein-Ausgabe-Subkanal
6	0	Byte, Bytestruktur
	1	Wort
7	0	Datenkonsistenz über 1 Byte bzw. Wort
	1	Datenkonsistenz über die gesamte Länge des Subkanals

Sonderdienste

- **max_channel_control,** Steuerbefehle CMD_START, CMD_STOP:
  Bus ein-bzw. ausschalten. Es werden keine Daten übergeben.
- **max_channel_info,** Infotyp INFO_DEVICE: Diagnose

Die Diagnose liefert folgende Status-Informationen über den Slave als USHORT-Wert:

Bit	Bedeutung
0	0: Offline-Zustand, 1= Passiv-Idle-Zustand
1	1=FDL-Indication liegt vor
2	1= Diagnose-Puffer wurde noch nicht vom Master abgeholt
3	1= interner Speicherfehler aufgetreten
4,5	DP-Zustand 00=Warte auf Parametrierung, 01=Warte auf Konfigurierung, 10=Normalbetrieb
6,7	Watchdog-Zustand: 00 Baud-Search, 01 Baud-Control, 10 DP-Control
8..11	aktuelle Baudrate (s.u.)
12..15	Version des SPC3 Profibus-Controllers

[14] Das spezielle Kennungsformat wird nicht von allen Mastern unterstützt und sollte nach Möglichkeit vermieden werden. Die genaue Kodierung dieses Formats ist aus der Norm zu entnehmen.

Kodierung der Baudrate:

Wert	0	1	2	3	4	5	6	7	8	9
Baud	12M	6M	3M	1,5M	500k	187,5k	93,75k	45,45k	19,2k	9600

Anmerkungen

Der Ausgabedienst muss aufgerufen werden, bevor andere Kanäle zu dem Modul geöffnet werden können. Danach ist der Slave konfiguriert. Der Ausgabedienst kann nur einmal aufgerufen werden.

Um an der Bus-Kommunikation teilzunehmen muss der Sonderdienst CMD_START aufgerufen werden. Soll der Slave zusätzlich Diagnosemeldungen übertragen, muss **vor** dem Start-Befehl der Diagnosekanal konfiguriert werden (siehe Kapitel 9.16.3.6. Diagnosemeldungen des Slaves).

Callback-Funktion

Beim Öffnen des Bus-Konfigurations-Kanals (max_open_channel) kann eine Callback-Funktion angegeben werden. Die Funktion bekommt in jedem Fall einen USHORT-Wert als Daten übergeben. Das Low-Byte kennzeichnet darin, welches Ereignis eingetreten ist. Bei einigen Ereignissen ist im High-Byte eine genauere Aufschlüsselung des Ereignisses enthalten (s.u.). Die Callback-Funktion wird für den Normal-Betrieb nicht benötigt, ist aber für Diagnosezwecke sehr hilfreich.

Ereignis	**Daten-Low-Byte**	**Daten-High-Byte**
Offline-Zustand erreicht	XDPS_OFFLINE	0
Data-Exchange-Zustand erreicht oder verlassen	XDPS_DATA_XCHG	Aktueller Zustand des Profibus-Controllers: 00h: Warte auf Parametrierung 10h: Warte auf Konfiguration 20h: Daten-Austausch
Neuer Steuerbefehl empfangen	XDPS_RCV_CTRL_CMD	Empfangener Befehl, bitweise kodiert: Bit 1 Clear-Befehl Bit 2 Unfreeze-Befehl Bit 3 Freeze-Befehl Bit 4 Unsync-Befehl Bit 5 Sync-Befehl
Fehler in der Parametrierung	XDPS_PARAM_ERROR	0
Fehler in der Konfiguration	XDPS_CFG_ERROR	0..243: Offset des ersten fehlerhaften Konfigurationsbytes 254..255: Länge der Konfiguration ist falsch

Ereignis	Daten-Low-Byte	Daten-High-Byte
Baudrate gefunden	XDPS_BAUDRATE_ DETECT	Baudrate, s.o.
Watchdog-Timeout[15]	XDPS_WATCHDOG_ TIMEOUT	0
Neue Stations-Adresse	XDPS_SLAVE_ADDR	Neue Slave-Adresse

10.17.3.4. User-Watchdog

Der in jedem der beiden Profibus-Controller enthaltene User-Watchdog-Timer dient dazu, sicherzustellen, dass Anwenderprogramme das Slave-Modul korrekt bedienen. Der Watchdog wird mit dem im usTimeout angegebenen Zählerwert initialisiert. Jedes vom Master empfangene Datentelegramm dekrementiert den Watchdogzähler. Wird der Zähler bis auf 0 heruntergezählt, wird der Slave inaktiv (Zustand „Warte auf Parametrierung“), d.h. er nimmt nicht mehr an der Bus-Kommunikation teil. Um das zu verhindern, muss der Ausgabedienst zyklisch aufgerufen werden (zumindest einmal, bevor der Watchdogzähler abläuft). Mit folgender CPS muss ein Kanal für die Triggerung des User-Watchdog geöffnet werden.

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_WATCHDOG*	
.usChannel	*0 oder 1*	Schnittstelle auf dem Modul (bei X-DPS-1: 0)
.usMode	*0*	reserviert
.usFlags	*_CP_EXCLUSIVE*	Kann gesetzt werden, um nur einen Kanal zu erlauben.
usTimeout	*1..65535*	Timeoutzähler

Ausgabedienst

- **max_trigger_channel**

Der Datentyp des Kanals ist DATA_VOID.

[15] Der Watchdog dient zur Überwachung der Master-Slave-Kommunikation. Er wird beim Empfang von Daten-Telegrammen automatisch getriggert. Der Timeoutwert des Watchdog-Timers wird vom Master durch das Standard-Parametrier-Telegramm eingestellt. Er liegt zwischen 20ms und 650s. Der Watchdog wird nur verwendet, wenn im Master für den Slave die Ansprechüberwachung aktiviert ist. Kommt es zum Ablauf der Überwachungszeit, nimmt der Watchdog-Zustand den Wert „Band control“ an und der DP-Zustand den Wert „Warte auf Parametrierung“.

10.17.3.5. Bus-Daten

Mit folgender CPS werden Kanäle geöffnet, über die der Slave dem Master Nutzdaten zur Verfügung stellt bzw. vom Master erhaltene Nutzdaten abholt.

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_BUS_IN*	Daten werden vom Slave an den Master gegeben
	DEVICE_BUS_OUT	Daten werden vom Master an den Slave gegeben
.usChannel	*0 oder 1*	Schnittstelle auf dem Modul (bei X-DPS-1: 0)
.usIndexFirst	*0..Anz. der Subkanäle*	Erster Subkanal
.usIndexLast	*.usIndexFirst bis Anz. der Subkanäle*	Letzter Subkanal
.usFlags	*0*	reserviert
usMode	0	für DEVICE_BUS_OUT
	IO_MODE_DIRECT	für DEVICE_BUS_IN: Daten werden sofort auf den Bus geschrieben
	IO_MODE_RAM	für DEVICE_BUS_IN: Daten werden durch den Ausgabedienst nur ins RAM geschrieben. Erst durch den Aufruf eines Sonderdienstes CMD_LATCH_ OUT werden sie aktiv (s.u.)

Eingabe- und Ausgabedienst

Hinweis: Die Bezeichnungen Ein- und Ausgänge sind aus der Sicht des Masters zu verstehen. D.h. in einen Kanal vom Typ DEVICE_BUS_IN muss der Slave die Daten, die er an den Master übertragen soll, hineinschreiben:

- **max_write_channel_block**

Aus einem Kanal vom Typ DEVICE_BUS_OUT liest der Slave die Daten, die ihm der Master sendet:

- **max_read_channel_block**

Der Datentyp ist in beiden Fällen DATA_UCHAR. Die Länge ist zuvor mit einem Sonderdienst zu ermitteln (s.u.).

Anmerkungen

Durch *usIndexFirst* und *usIndexLast* wird der Subkanal spezifiziert, zu dem ein Kanal geöffnet werden soll. Der Wert entspricht der Position in den Konfigurationsdaten, die dem Slave mit dem Aufruf des Ausgabedienstes des Slave-Konfigurationskanals übergeben wurden. Ein *DEVICE_BUS_OUT*-Kanal kann nur dann geöffnet werden, wenn in dem Konfigurationsdaten-Array an der Position

usIndexFirst ein Konfigurationsbyte steht, das besagt, dass dieser Subkanal ein Ausgang ist. Dementsprechend kann ein *DEVICE_BUS_IN*-Kanal kann nur dann geöffnet werden, wenn in dem Konfigurationsdaten-Array an der Position *usIndexFirst* ein Konfigurationsbyte steht, das besagt, dass dieser Subkanal ein Eingang ist.

Ein Blockkanal ist möglich, indem in *usIndexLast* ein anderer Wert als in *usIndexFirst* angegeben wird. Dabei ist darauf zu achten, dass Ausgänge und Eingänge nicht in einem Block gemischt werden können.

Sonderdienste

- **max_channel_info,** Infotyp INFO_MAXSIZE:
  liefert die Länge eines Datenkanals in einer ULONG-Variablen
- **max_channel_control,** Steuerbefehl CMD_LATCH_OUT:
  Für DEVICE_BUS_IN-Kanäle werden die im RAM zwischengespeicherten Werte aller DEVICE_BUS_IN-Kanäle gleichzeitig auf dem Profibus aktiv. Für Konfigurationen mit mehreren Eingabe-Baugruppen ermöglicht das Datenkonsistenz aller Baugruppen. Dazu müssen die Kanäle mit usMode=IO_MODE_RAM geöffnet werden. Es werden keine Daten übergeben.

Callback-Funktion

Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen, wenn der Slave ein neues Daten-Telegramm vom Master empfangen hat. Die Funktion bekommt 0 als USHORT-Wert übergeben. Die Callback-Funktionalität ist nur für DEVICE_BUS_OUT-Kanäle verfügbar.

Da dadurch u.U. ein hohes Interrupt-Aufkommen auf dem CPU-Modul entsteht, ist zu prüfen, ob Callbacks tatsächlich benötigt werden. Da die Daten für alle DEVICE_BUS_OUT-Kanäle gleichzeitig empfangen werden, werden auch alle zugehörigen Callback-Funktionen (nacheinander) aufgerufen. Es reicht daher prinzipiell aus, nur bei einem DEVICE_BUS_OUT-Kanal eine Callback-Funktion anzugeben.

10.17.3.6. Diagnosemeldungen des Slaves

Jeder Slave stellt dem Master Diagnosedaten zur Verfügung. 6 Bytes sind von der Profibus-Norm fest vorgeschrieben. Davon geben die ersten 3 Bytes Information über den Status des Slave (s. Norm). Das vierte Byte enthält die Adresse des DP-Masters, der den Slave parametriert hat. Das fünfte und sechste Byte enthalten die Herstellerkennung des Slaves. Um diese Standard-Diagnose braucht sich der Anwender nicht zu kümmern, sie wird automatisch vom Profibus-Controller auf dem Modul gesetzt.

Soll der Slave darüber hinaus weitere anwenderspezifsche Diagnosedaten liefern, muss ein Kanal mit folgender CPS geöffnet werden, der dies konfiguriert:

Strukturelement	**Werte**	**Bedeutung**
.usDevice	*DEVICE_ DIAGNOSIS*	Diagnosekanal des Slaves
.usChannel	*0 oder 1*	Schnittstelle auf dem Modul (bei X-DPS-1: 0)
.usIndexFirst	*0*	reserviert
.usIndexLast	*0*	reserviert
.usFlags	*_CP_EXCLUSIVE*	Kann gesetzt werden, um nur einen Diagnosekanal zu erlauben.
usMode	*XDPS_DIAG_FOR_ DEVICE*	Der Kanal dient zum Schreiben von Gerätebezogenen Diagnosedaten (Kodierung ist gerätespezifisch)
	XDPS_DIAG_FOR_ ID	Der Kanal dient zum Schreiben von Kennungs-bezogenen Diagnosedaten (s.u.).
	XDPS_DIAG_FOR_ CHANNEL	Der Kanal dient zum Schreiben von Kanal-bezogenen Diagnosedaten (s.u.).

Ausgabedienst

Der Ausgabedienst übergibt die Diagnosemeldungen.

Der Datentyp ist DATA_UCHAR. Die maximale Länge beträgt 40 Bytes (inkl. Standard Diagnose). Die Länge ist durch den MDD sowie die GSD-Datei festgelegt.

- **max_write_channel_block**

Anmerkungen

Entsprechend der Profibus Norm können 3 verschiedenartige Diagnosemeldungen gesetzt werden. In welchem Format die Diagnose geschrieben wird, hängt vom CPS-Element *usMode* ab:

- *XDPS_DIAG_FOR_DEVICE*: Die Diagnose besteht aus max. 32 Bytes gerätespezifischer Information. Dem Ausgabedienst müssen nur die reinen Diagnosedaten übergeben werden. Der erforderliche Header wird vom MDD automatisch den Diagnosedaten vorangestellt.
- *XDPS_DIAG_FOR_ID*: Die Diagnose ist ein Bitfeld dessen Größe durch die Anzahl der Konfigurationsbytes festgelegt ist. Jedes Bit zeigt darin an, ob das zugehörige Konfigurationsbyte (und damit der entsprechende I/O-Bereich) eine Diagnose hat (Bit=1) oder nicht (Bit=0). Dem Ausgabedienst müssen nur die reinen Diagnosedaten übergeben werden. Der erforderliche Header wird vom MDD automatisch den Diagnosedaten vorangestellt.
- *XDPS_DIAG_FOR_CHANNEL:* Die Diagnose ist kanalbezogen. Für jeden Kanal werden 3 Bytes erwartet. Insgesamt kann also durch einen Aufruf des Ausgabedienstes für max. 11 Kanäle eine Diagnose gesetzt werden. Es ist zu beachten, dass in dieser Betriebsart die Diagnosedaten für jeden Kanal inkl. der erforderlichen Headerbytes übergeben werden müssen:

Bit	7	6	5	4	3	2	1	0
1. Byte	1	0	Kennungsnummer					
2. Byte	Ein-Ausgabe		Kanalnummer					
3. Byte	Kanaltyp				Fehlertyp			

Die Kodierung für Ein-Ausgabe, Kanaltyp und Fehlertyp ist der Norm zu entnehmen.

Callback-Funktion

Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen, wenn die Diagnose auf dem Profibus zur Verfügung gestellt worden ist. Die Funktion bekommt 0 als USHORT-Wert übergeben.

10.17.4. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Pin	Funktion	Bedeutung
1	Kanal A, DPPE	Optional (Schirm)
2	Kanal A, DP5V	+5 Volt, isoliert
3	-	n.c.
4	-	n.c.
5	Kanal A, DPB	DPB, isoliert
6	Kanal A, DPA	DPA, isoliert
7	Kanal A, DPRTS	Request To Send, isoliert
8	-	n.c.
9	Kanal A, DPGND	Ground, isoliert
10	-	n.c.
11	Kanal B, DPPE	Optional (Schirm)
12	Kanal B, DP5V	+5 Volt isoliert
13		n.c.
14		n.c.
15	Kanal B, DPB	DPB, isoliert
16	Kanal B, DPA	DPA, isoliert
17	Kanal B, DPRTS	Request To Send, isoliert
18		n.c.
19	Kanal B, DPGND	Ground, isoliert
20		n.c.
21..40		n.c.

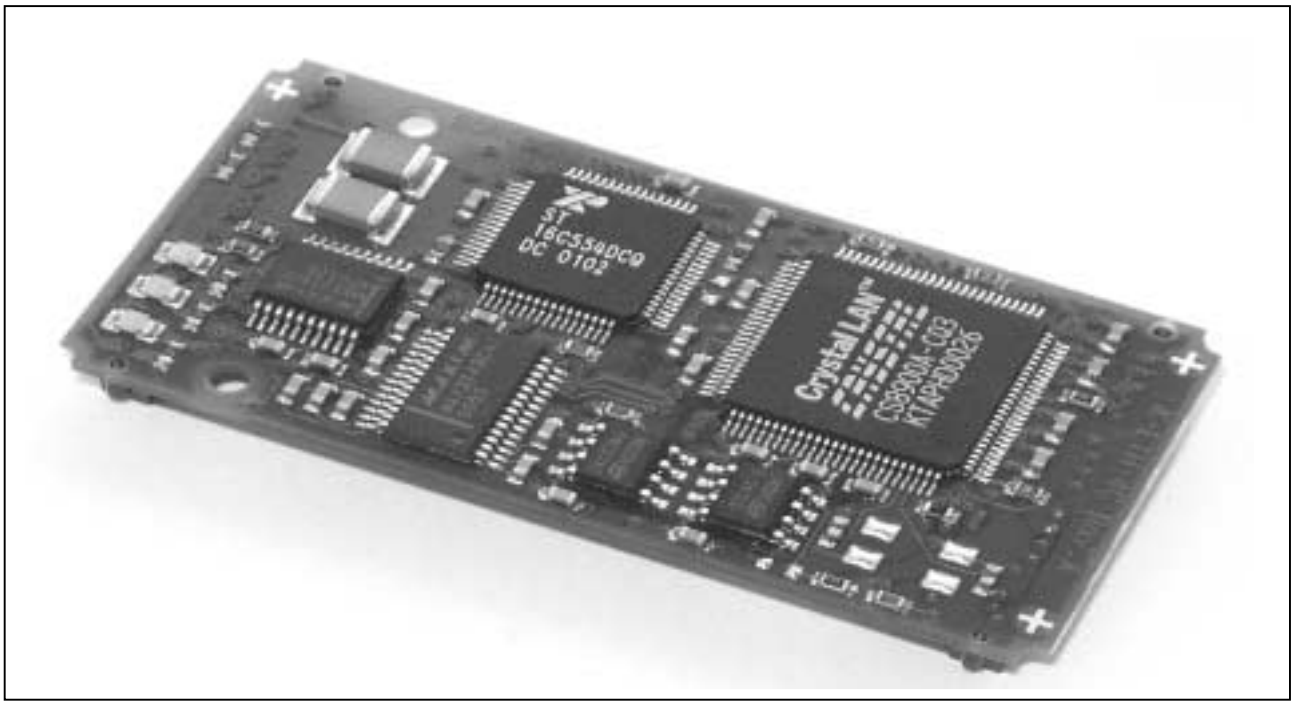
n.c.: not connected

10.17.5. Besondere Eigenschaften

Parameter	Wert	Einheit
Anzahl Kanäle	1 bzw. 2	-
Physikalischer Anschluss: Kanal A und B	PROFIBUS	
Galvanische Trennung	je Kanal isoliert	
Übertragungsrate, max.	12	MBaud
PROFIBUS-Controller	SPC3	-
Überspannungsfestigkeit der Ein-/Ausgänge	tbd	V
Temperatur-Bereich, Betrieb optional	0 bis 70 -40 bis +85	°C °C
Stromaufnahme (3,3V) (die X-Bus Spannungen ±12V werden nicht benötigt)	tbd	mA

X-ETH-10

10Base-T Ethernet Schnittstelle mit TCP/IP



10.18. X-ETH-10

Inhaltsverzeichnis

10.18.1. Beschreibung

Mit dem Modul X-ETH-10 (X-Bus-Modul-Typ 81, Subtyp 1) kann ein X-Bus-System an ein Ethernet-Netzwerk angeschlossen werden.

Die 10Base-T Schnittstelle erlaubt Ethernet Kommunikation mit 10 Mbit/s. Sie kann entweder für die Ankopplung eines X-Bus-Systems über Ethernet oder über den Modul Device Treiber für unabhängige TCP/IP-Kommunikation verwendet werden. Das Modul hat die Typ-Nr. 80, Subtyp 1.

10.18.2. Modul-Device-Treiber

10.18.2.1. Installation

Der Modul-Device-Treiber für OsX hat die Programmnummer 8051h und den Dateinamen mxeth10.exe. Der Modul-Device-Treiber für Windows hat den Namen mxeth10.sys.

Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8051, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8051

10.18.2.2. Kanaleigenschaftsstruktur CPS_XETH

Die CPS für das Modul hat den Namen CPS_XETH.

10.18.2.3. TCP/IP-Sockets

Ein Zugriff auf ein TCP/IP-Socket kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TCP_SOCKET*	TCP/IP-Socket
.usType	*TCP_SERVER*	Server
	TCP_CLIENT	Client
.usFlags	*_CP_EXCLUSIVE*	Dieses Flag muss gesetzt sein! Der Zugriff erfolgt immer exklusiv.
.ulCallbackEvents	*XETH_EVENT_ESTABLISHED*	Benachrichtigung bei Verbindung
	XETH_EVENT_CLOSED	Benachrichtigung beim Schließen
	XETH_EVENT_NEW_DATA	Benachrichtigung bei neuen Daten
.usReadMode	*IO_MODE_RAM*	Die Empfangs-Daten werden aus einem Empfangspuffer des MDDs entnommen. Der MDD trägt die empfangenen Daten selbstständig in diesen Puffer ein.
.usWriteMode	*IO_MODE_RAM*	Die Sende-Daten werden in einen internen Sendepuffer des MDDs geschrieben. Der Puffer wird automatisch vom MDD geleert.
	IO_MODE_DIRECT	Die Daten werden unter Verwendung des PSH-Flags versendet.
.ulServerIp		IP-Adresse des Servers bei Client
.usServerPort	*0 – 65535*	Eigener Port bei Server, Port des Servers bei Client

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Ist bei einem Lesevorgang nicht die gewünschte Anzahl Zeichen vorhanden, werden die im Puffer vorhandenen Zeichen zurückgegeben. Ist bei einem Schreibzugriff im Puffer nicht genügend Platz vorhanden, werden die Daten nicht übernommen.
Wenn die Verbindung noch nicht hergestellt wurde, so liefert der Schreibdienst den Fehler ERR_NOT_CONNECTED zurück.
Wurde die Verbindung geschlossen, so wird der Fehler ERR_CLOSED zurückgegeben.

Sonderdienste

- **max_channel_control,** Steuerwert CMD_START: geschlossenes Socket wieder öffnen. Ein Server geht in den Listen-Modus über, ein Client versucht ein Connect mit der Server-Seite.
- **max_channel_control,** Steuerwert CMD_STOP: offenes Socket schliessen. Ein geschlossenes Socket nimmt keine Daten vom Netzwerk entgegen und kann keine Daten versenden.
- **max_channel_control,** Steuerwert CMD_RESET: Empfangs- und Sendepuffer eines Sockets leeren.

  Bei allen 3 Sonderdiensten werden keine Daten übergeben.
- **max_channel_info,** Infotyp INFO_DEVICE: aktuellen Status des Sockets abfragen. Es wird ein ULONG mit folgender Bedeutung geliefert:

Wert	Bedeutung
XETH_LISTEN	Server wartet auf Verbindung durch Client
XETH_CONNECTING	Verbindungsaufbau
XETH_ESTABLISHED	Verbindung ist aktiv
XETH_CLOSING	Verbindung wird beendet
XETH_CLOSED	Verbindung geschlossen (inaktiv)

Hinweise

Beim Öffnen des Kanals geht ein Server in den Listen-Modus, ein Client versucht eine Verbindung zum Server aufzubauen.
Wird die Schnittstelle für eine Host-Ankopplung verwendet, so stehen die Ports 1024 bis 1040 nicht zur Verfügung.

10.18.2.4. IP-Einstellungen

Ein Zugriff auf die IP-Einstellungen kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Control-Kanal
.usIndex	*XETH_IP_SETTING*	IP-Einstellungen
.usReadMode	*IO_MODE_DIRECT*	Die aktuell verwendeten IP-Einstellungen werden zurückgegeben.
.usWriteMode	*IO_MODE_DIRECT*	Die aktuell verwendeten IP-Einstellungen werden gesetzt und im EEPROM gespeichert.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Die IP-Einstellungen der ETHERNET-Schnittstelle werden im EEPROM des Moduls gespeichert. Beim Laden des MDDs werden die Einstellungen aus dem EEPROM gelesen, so daß ein Setzen der Einstellungen mit Hilfe dieses Kanals nur bei einer Änderung der Einstellungen notwendig ist.

Zum Lesen und Schreiben der IP-Einstellungen steht die folgende Struktur XETH_IP_SETTINGS zur Übergabe zur Verfügung:

Strukturelement	Werte	Bedeutung
.usVersion	*0*	Version
.ulIp		IP-Adresse
.ulSubnetmask		Subnetzmaske
.ulGateway		Standard-Gateway

Wird der Schreibdienst IO_MODE_DIRECT verwendet, so werden geöffnete Sockets in den Modus *XETH_CLOSED* gesetzt.

10.18.2.5. Adapter-Einstellungen

Ein Zugriff auf die Adapter-Einstellungen kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Control-Kanal
.usIndex	*XETH_ADAPTER_SETTING*	Adapter-Einstellungen
.usReadMode	*IO_MODE_DIRECT*	Die aktuell verwendeten Adapter-Einstellungen werden zurückgegeben.
.usWriteMode	*IO_MODE_DIRECT*	Die aktuell verwendeten Adapter - Einstellungen werden gesetzt und im EEPROM gespeichert.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Die Adapter-Einstellungen werden im EEPROM des Moduls gespeichert. Beim Laden des MDDs werden die Einstellungen aus dem EEPROM gelesen, so dass ein Setzen der Einstellungen mit Hilfe dieses Kanals nur bei einer Änderung notwendig ist.

Die Adapter-Einstellungen werden mit der Struktur XETH_ADAPTER_SETTINGS an den Eingabe- bzw. Ausgabedienst übergeben:

Strukturelement	Werte	Bedeutung
.usVersion	*0*	Version
.usTransMode	XETH_HALFDUPLEX_10 XETH_FULLDUPLEX_10	Übertragungs-Modus

10.18.2.6. MAC-Adresse

Ein Zugriff auf die MAC-Adresse kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Control-Kanal
.usIndex	*XETH_MAC_ADDRESS*	MAC-Adresse
.usReadMode	*IO_MODE_DIRECT*	Die aktuell verwendete MAC-Adresse wird zurückgegeben.
.usWriteMode	*IO_MODE_DIRECT*	Die aktuell verwendete MAC-Adresse wird gesetzt und im EEPROM gespeichert.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Die MAC-Adresse der ETHERNET-Schnittstelle ist eine vom Hersteller eindeutig vergebene Adresse, die aus 6 Byte besteht. Normalerweise wird diese Einstellung für Anwenderprogramme nicht benötigt. Sie sollte nicht verändert werden. Wird der Schreibdienst benutzt, so werden geöffnete Sockets in den Modus *XETH_CLOSED* gesetzt.

10.18.2.7. Sonstige Dienste

Nach der Installation stellt der MDD bereits ohne Öffnen eines Kanals den ARP-Reply-Dienst und ICMP-Echo-Reply-Dienst zur Verfügung.
Mittels ARP-Reply können andere Netzwerkteilnehmer die MAC-Adresse der Schnittstelle über die IP-Adresse erfragen
Durch ICMP-Echo-Reply kann ein Ping auf die Schnittstelle erfolgen.

10.18.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

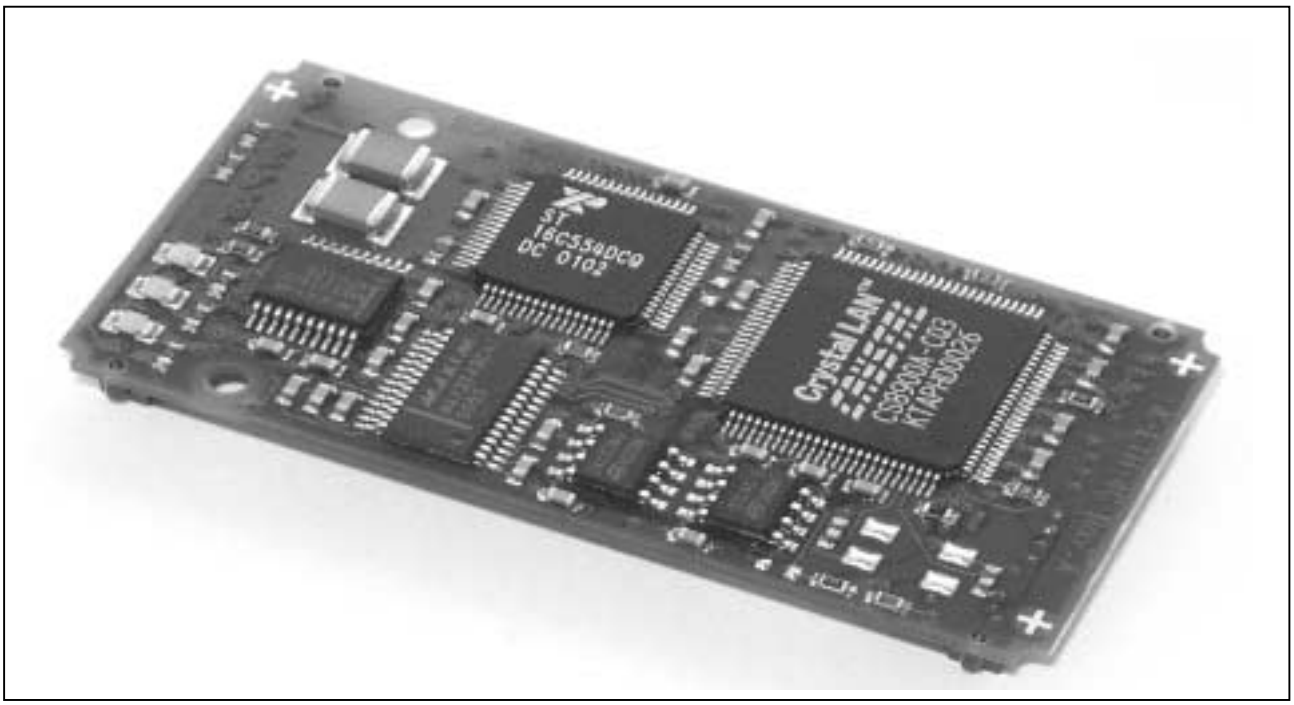
Pin	Funktion
1	TD+
2	TD-
3	RD+
4	RD-
5	Abschirmung
6	+3,3V
7	LAN-LED
8	LINK-LED
9	HC1-LED
10, 15, 20, 29, 39	GND
Rest	not connected

10.18.4. Besondere Eigenschaften

Parameter	Wert	Einheit
Physikalische Schnittstelle (10MBit)	10Base-T	
Controller	CS8900A	-
Abmessungen	29x58x8	mm
Temperatur-Bereich, Betrieb	-40 bis +85	°C
Stromaufnahme (3,3V) (die X-Bus Spannungen ±12V werden nicht benötigt)	190	mA

X-ETH-4C

10Base-T Ethernet und 4 serielle Schnittstellen (2 x RS-232 und 2 x RS-232, RS-422 oder RS-485)



10.19. X-ETH-4C

Inhaltsverzeichnis

10.19.1. Beschreibung

Das Modul X-ETH-4C stellt eine 10BaseT-Ethernet-Schnittstelle sowie 4 serielle Schnittstellen zur Verfügung.

Die 4 seriellen Schnittstellen sind PC-kompatibel. Zwei davon sind als RS-232 mit allen Modem-Steuersignalen ausgeführt, die beiden anderen können per Software als RS-232, RS-422 oder RS-485 umgeschaltet werden. Als RS-232 verfügen sie über 2 Modem-Steuersignale RTS und CTS. Jede Schnittstelle bietet für jede Kommunikationsrichtung ein 16 Byte großes FIFO. Die max. Baudrate beträgt 460,8 kBaud je Kanal.

Die 10Base-T Schnittstelle erlaubt Ethernet Kommunikation mit 10 Mbit/s. Sie kann entweder für die Ankopplung eines X-Bus-Systems über Ethernet oder über den Modul Device Treiber für unabhängige TCP/IP-Kommunikation verwendet werden. Das Modul hat die Typ-Nr. 80, Subtyp 1.

10.19.2. Software

Die Beschreibung des MDDs lag zur Drucklegung noch nicht vor. Sobald sie verfügbar ist, wird sie auf der SORCUS-Homepage bereitgestellt.

10.19.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

		Signal			
Pin	**Kanal**	**RS-232**	**RS-422**	**RS-485**	**Ethernet**
1	Ethernet	-	-	-	TD+
2	Ethernet	-	-	-	TD-
3	Ethernet	-	-	-	RD+
4	Ethernet	-	-	-	RD-
5	Ethernet	-	-	-	Abschirmung
6	Ethernet	-	-	-	+3,3V
7	Ethernet	-	-	-	LAN-LED
8	Ethernet	-	-	-	LINK-LED
9	Ethernet	-	-	-	HC1-LED
10	-	-	-	-	GND
11	0	RCV	IN+	-	-
12	0	RTS	OUT+	DATA+	-
13	0	TMT	OUT-	DATA-	-
14	0	CTS	IN-	-	-
15	0	GND	GND	GND	-
16	1	RCV	IN+	-	-
17	1	RTS	OUT+	DATA+	-
18	1	TMT	OUT-	DATA-	-
19	1	CTS	IN-	-	-
20	1	GND	GND	GND	-
21	2	DCD	-	-	-
22	2	DSR	-	-	-
23	2	RCV	-	-	-
24	2	RTS	-	-	-
25	2	TMT	-	-	-
26	2	CTS	-	-	-

		Signal			
Pin	**Kanal**	**RS-232**	**RS-422**	**RS-485**	**Ethernet**
27	2	DTR	-	-	-
28	2	Ri	-	-	-
29	2	GND	-	-	-
30	2	-	-	-	-
31	3	DCD	-	-	-
32	3	DSR	-	-	-
33	3	RCV	-	-	-
34	3	RTS	-	-	-
35	3	TMT	-	-	-
36	3	CTS	-	-	-
37	3	DTR	-	-	-
38	3	Ri	-	-	-
39	3	GND	-	-	-
40	3	-	-	-	-

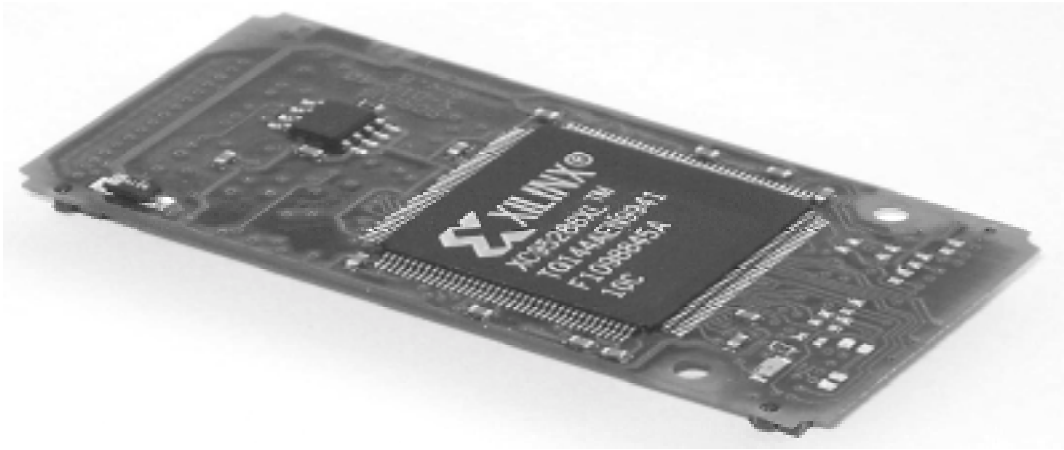
10.19.4. Besondere Eigenschaften

Parameter	Wert	Einheit
Ethernet-Schnittstelle		
Physikal. Schnittstelle (10 MBit)	10Base-T	
Controller	CS8900A	
Serielle Schnittstellen (Kanal A und B)		
per Software wählbar	RS-232, RS-422 oder RS-485	
max. Baudrate	460,8	kBaud
FIFO (je Kanal und je Kommunikationsrichtung)	16	Byte
RS-232		
Eingangsspannung, max.	±25	V
Eingangsschwelle Low, min.	0,6	V
Eingangsschwelle High, max.	2,0	V
Hysterese, typ.	0,5	V
Eingangswiderstand, min./typ./max.	3/5/7	kΩ
Ausgangsspannung (3 kΩ an GND), min./typ.	±5/±5,4	V
Ausgangsstrom (Ausgang an GND), max.	±60	mA
RS-422 und RS-485		
Eingangswiderstand, min.	48	kΩ
Eingangsstrom, max. (-7V .. +12V)	–0,15/0,25	mA
Differentielle Eingangsschwelle, min./max.	–50/-200	mV
Hysterese, typ.	30	mV
Differentielle Ausgangsspannung, min. (RS-422/RS-485) (RS-485 = 27 Ω, RS-422 = 50 Ω)	2/1,5	V
Temperatur-Bereich, Betrieb	–40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	10,4	g
Stromaufnahme		
3,3V, typ. (die X-Bus Spannungen ±12V werden nicht benötigt)	tbd	mA

Notizen

X-IDE-1

IDE Schnittstelle



10.20. X-IDE-1

Inhaltsverzeichnis

10.20.1. Beschreibung

Das Modul X-IDE-1 stellt eine Standard IDE-Schnittstelle zur Verfügung.

10.20.2. Modul-Device-Treiber

Die Beschreibung des Modul-Device-Treibers lag bei Drucklegung dieses Handbuchs noch nicht vor. Sobald diese fertig ist, wird sie auf der SORCUS-Homepage bereitgestellt.

10.20.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

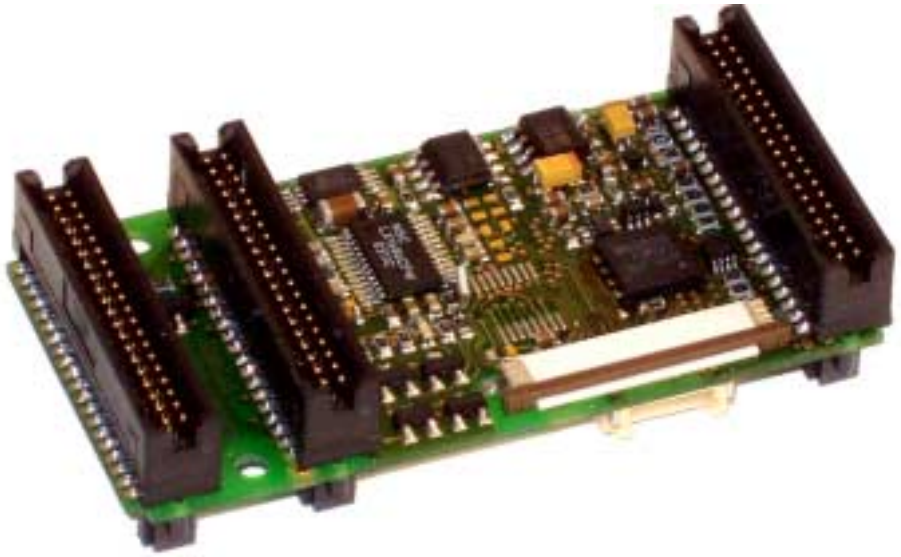
Pin	Funktion	Bedeutung
1	/RESET	Reset
2	GROUND	Masse
3	DD7	Data Bus Bit 7
4	DD8	Data Bus Bit 8
5	DD6	Data Bus Bit 6
6	DD9	Data Bus Bit 9
7	DD5	Data Bus Bit 5
8	DD10	Data Bus Bit 10
9	DD4	Data Bus Bit 4
10	DD11	Data Bus Bit 11
11	DD3	Data Bus Bit 3
12	DD12	Data Bus Bit 12
13	DD2	Data Bus Bit 2
14	DD13	Data Bus Bit 13
15	DD1	Data Bus Bit 1
16	DD14	Data Bus Bit 14
17	DD0	Data Bus Bit 0
18	DD15	Data Bus Bit 15
19	GROUND	Masse
20	n.c.	not connected

Pin	Funktion	Bedeutung
21	DMARQ	DMA Request
22	GROUND	Masse
23	/DIOW	IO Write
24	GROUND	Masse
25	/DIOR	IO Read
26	GROUND	Masse
27	IORDY	IO Ready
28	SPSYNC, CSEL	Spindle Sync/Cable Select
29	/DMACK	DMA Acknowledge
30	GROUND	Masse
31	INTRQ	Interrupt Request
32	/IOCS16	16 Bit IO
33	DA1	Address Bit 1
34	/PDIAG	Passed Diagnostic
35	DA0	Address Bit 0
36	DA2	Address Bit 2
37	/CS0, /CS1FX	Chip Select 0
38	/CS1, /CS3FX	Chip Select 1
39	/DASP	Drive Active
40	GROUND	Masse

Notizen

X-LCD-S1
X-LCD-H1

Untersteckbares Modul zum Anschluss von LCD-Displays



10.21. X-LCD-S1, X-LCD-H1

Inhaltsverzeichnis

10.21.1. Beschreibung

Die Module X-LCD-S1 und X-LCD-H1 ermöglichen den einfachen Anschluss von LC-Displays an X-Bus Systeme. Das Modul wird unter ein CPU-Modul gesteckt und führt die Signale des LCD-Controllers auf geeignete Steckverbinder heraus. Die Spannungsversorgung des angeschlossenen LCD-Panels erfolgt ebenfalls über das X-LCD. Zusätzlich ist der Hochspannungs-Inverter für die Hintergrundbeleuchtung integriert. Per Software kann sowohl die Helligkeit der Hintergrundbeleuchtung als auch der Kontrast des Displays eingestellt werden.

Die Signale des LCD-Controllers des CPU-Moduls werden nicht auf den unteren Stecker C des X-LCD Moduls weitergegeben. Beim Einsatz auf dem X-KiT-3 wird das Display des X-KiT-3 daher nicht mehr angesteuert.

Das X-LCD-S1 eignet sich für den Anschluss des 5,7" TFT-Displays SHARP LQ057Q3DC02 und mit einer Adapterplatine (auf Anfrage) auch für das 3,8" TFT-Display SHARP LQ038Q5DR01.

Das X-LCD-H1 eignet sich für den Anschluss des 6,3" DSTN-Displays HITACHI SX16H003. Dieser Modultyp hat auch einen integrierten Baustein für die Auswertung des Touch-Displays.

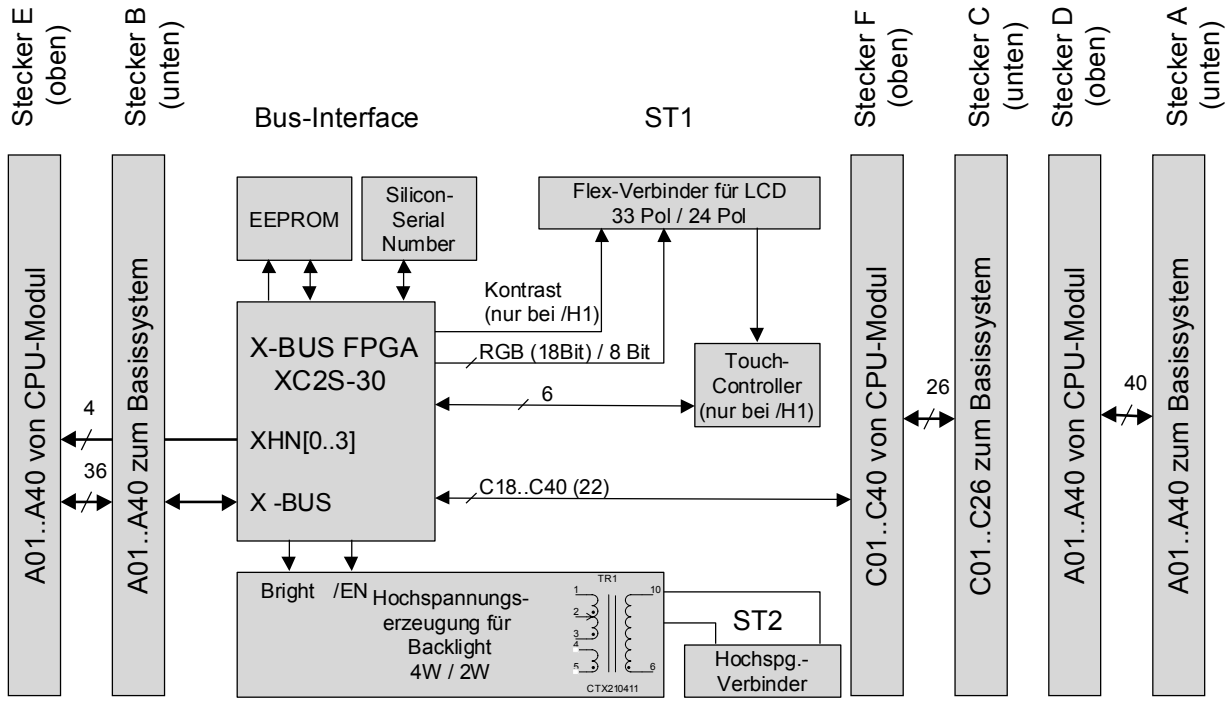
Anpassungen an andere Displaytypen sind auf Anfrage möglich.

!

Vorsichtsmassnahmen:

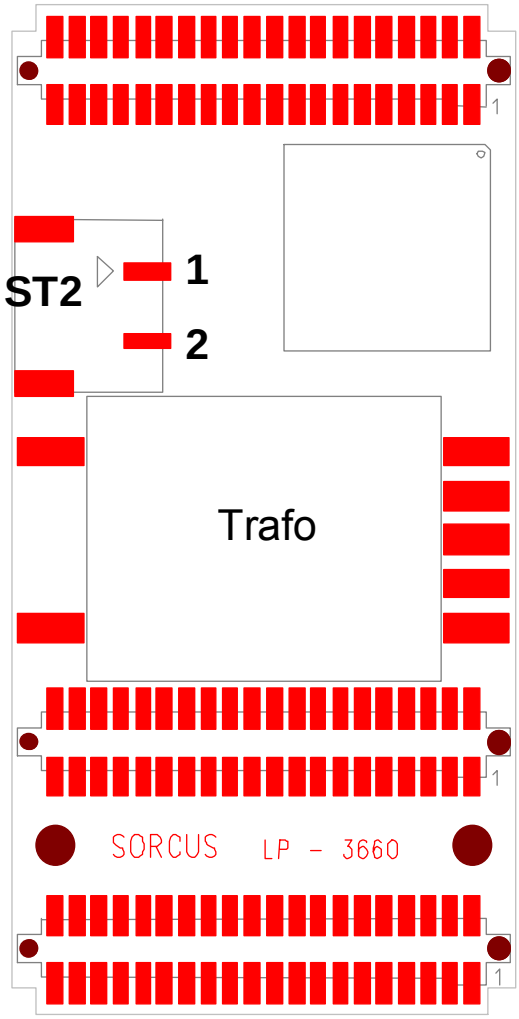
Auf dem Modul wird eine Hochspannung von bis zu 1500V erzeugt. Die abgegebene Leistung beträgt bis zu 4W. Schliessen Sie daher das Kabel für die Hintergrundbeleuchtung nur im abgeschalteten Zustand an.

10.21.2. Blockschaltbild

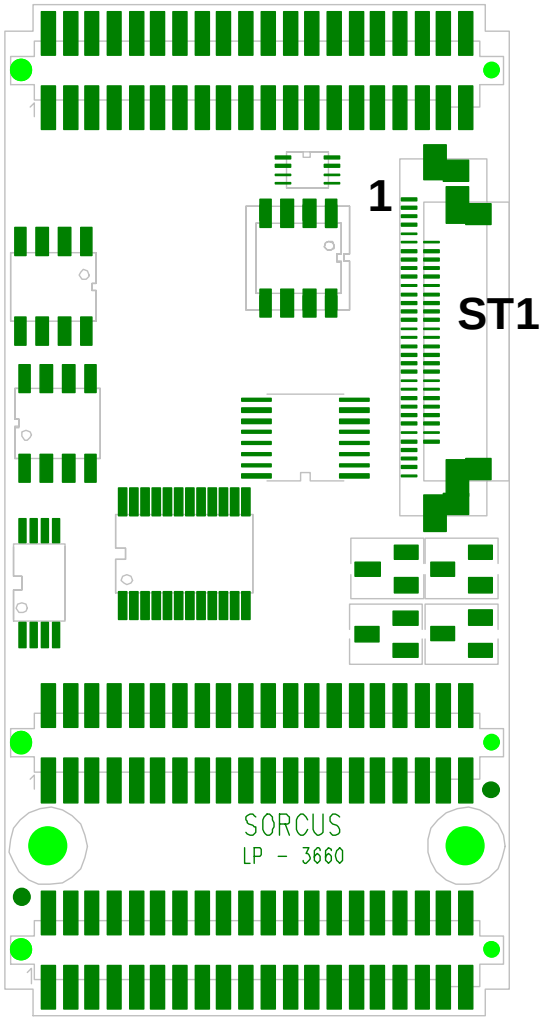


10.21.3. Lageplan

Ansicht von unten



Ansicht von oben



10.21.3.1. Steckerbelegung

Je nach Modulversion ist ST1 mit einer 33-Poligen Filmkabelbuchse oder mit einer 24-Poligen Filmkabelbuchse bestückt. In diesen Verbinder muss das Filmkabel mit den Kontakten nach unten eingelegt werden. Überprüfen Sie unbedingt die richtige Zuordnung von Pin 1 des ST1 zu Pin 1 des LCD-Panels.

!

Achtung:

Bei falscher Verbindung des Filmkabels kann sowohl das Modul als auch das LCD-Panel zerstört werden.

Pin-Belegung ST1:

Pin	X-LCD-S1	X-LCD-H1
1	GND	FLM
2	CLK	GND
3	HSYNC	CL1
4	VSYNC	GND
5	GND	CL2
6	R0	VCC
7	R1	GND
8	R2	D0
9	R3	D1
10	R4	D2
11	R5	D3
12	GND	GND
13	G0	D4
14	G1	D5
15	G2	D6
16	G3	D7
17	G4	/DISP_Off

Pin	X-LCD-S1	X-LCD-H1
18	G5	VCC
19	GND	VCON
20	B0	GND
21	B1	Y(-)
22	B2	X(-)
23	B3	Y(+)
24	B4	X(+)
25	B5	-
26	GND	-
27	ENAB	-
28	VCC	-
29	VCC	-
30	R/L	-
31	U/D	-
32	V/Q	-
33	GND	-

Stecker ST2 liefert die Hochspannung für die Hintergrundbeleuchtung. Pin 1 ist die "kalte" Seite, Pin 2 die Hochspannungsseite.

Pin-Belegung ST2:

Pin	Funktion
1	Rückführung der Hochspannung
2	Hochspannung

Stecker A des Moduls ist 1:1 mit den Signalen des Steckers A des aufgesteckten CPU-Moduls verbunden. Die Steckerbelegung entnehmen Sie bitte der Beschreibung des eingesetzten CPU-Moduls.

Der Stecker C des Moduls gibt nur einen Teil der Signale des Steckers C des aufgesteckten CPU-Moduls weiter. Die Belegung der Signale entnehmen Sie bitte der Beschreibung des eingesetzten CPU-Moduls.

Pin-Belegung Stecker C:

Pin	Funktion
1-26	siehe Beschreibung CPU-Modul
27-40	Unbeschaltet

10.21.3.2. Montage der Kabel

Achten Sie bei der Verlegung der Kabel zum LCD-Panel darauf, das Filmkabel nicht abzuknicken. Für die Verlängerung des Filmkabels sind auf Anfrage Adapterplatinen lieferbar.

Vermeiden Sie die Führung des Hochspannungskabels neben Analog-Modulen, da sonst die Genauigkeit der Wandler beeinträchtigt wird. Führen Sie das Hochspannungskabel auch nicht direkt über andere Module hinweg.

10.21.4. Modul-Device-Treiber

Die Funktionen des Moduls werden nicht über einen Modul-Device-Treiber angesteuert, sondern über Funktionen aus der Grafikbibliothek.

Schlagen Sie hierzu im Kapitel 7. nach.

10.21.5. Besondere Eigenschaften

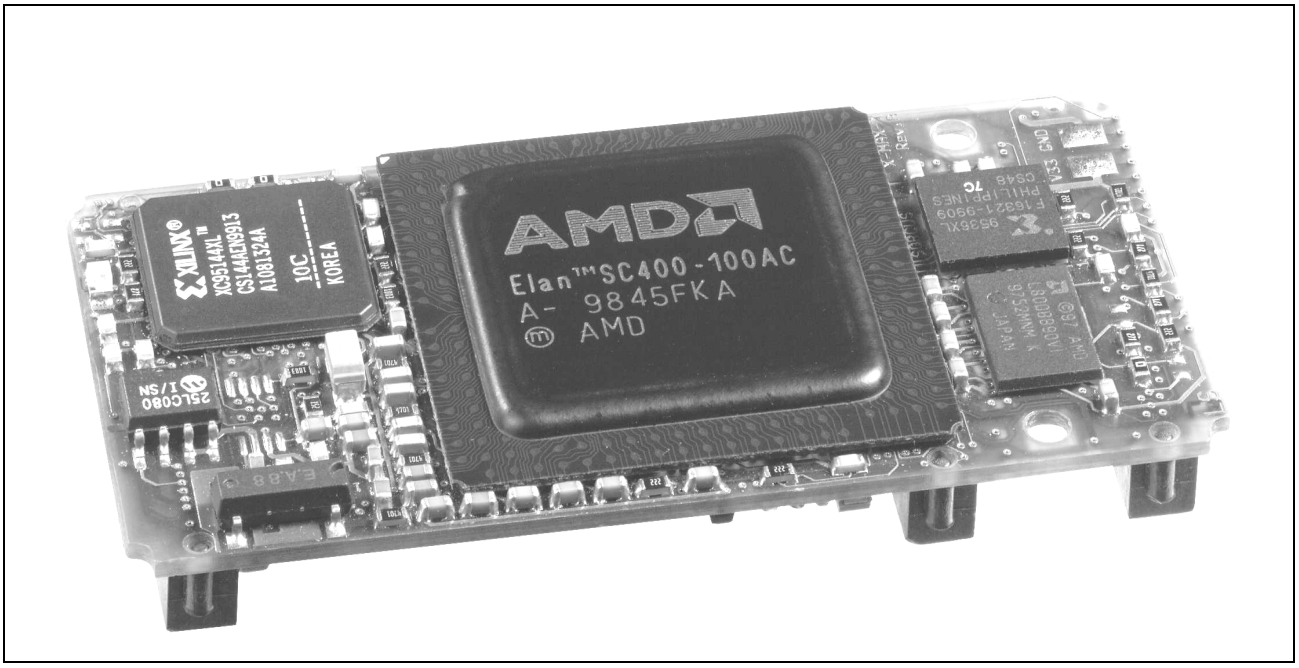
Parameter	Randbedingungen	Wert min.	Wert typ.	Wert max.	Einheit	Anm.
Hochspannungs-erzeugung						
Spannung, Betrieb	X-LCD-S1		690		V	
	X-LCD-H1		560		V	
Spannung, Zündung	X-LCD-S1	1500			V	
	X-LCD-H1	1200			V	
Lampenstrom	X-LCD-S1	2,5	5,0	5,5	mA	
	X-LCD-H1	0,5	1,5	3,0	mA	
Dimmbereich		20		100	%	
Frequenz			60		kHz	
Versorgungsspannungen des Moduls						
Versorgungsstrom	für +3,3 V,					(1)
	Hintergundbeleuchtung aus		65		mA	
	Hintergundbeleuchtung ein			1450	mA	
	für +/-12 V		0		mA	
Temperaturbereich Betrieb		0		+70	°C	
Lagerung		–40		+85	°C	

(1) Hinzu kommt der Versorgungsstrom des Displays von 30mA (SX16H003) bzw. 160mA (LQ057Q3DC02)

Notizen

X-MAX-1

Der kleinste PC der Welt



10

10.22. X-MAX-1, Rev. D, E und F

Inhaltsverzeichnis

10.22.1. Beschreibung

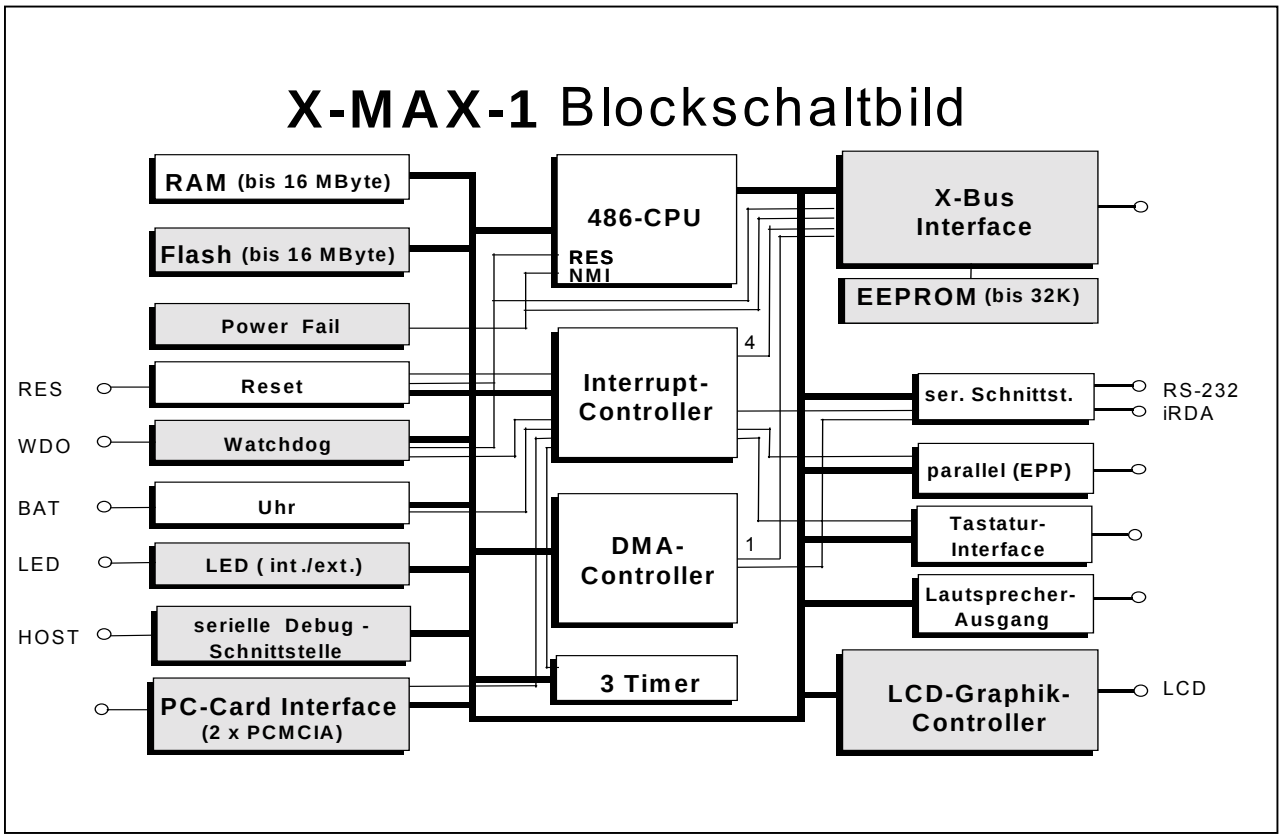
Das CPU-Modul X-MAX-1 ist mit einer 486-CPU bestückt und enthält einen 8 KByte großen Cache. Die Taktfrequenz der lokalen CPU beträgt je nach Version zwischen 33 MHz und 100 MHz (interner Takt).

Das Modul ist bereits mit umfangreicher Peripherie ausgestattet. Außer Flash-ROM (max.16 MByte) und RAM (max.16 MByte) verfügt es über 3 Timer, Interrupt-und DMA-Controller, eine serielle Schnittstelle und eine Druckerschnittstelle, eine Uhr, einen Watch-Dog und Spannungsüberwachung. Die Uhr kann mit einer externen Batterie gepuffert werden. Alle Schnittstellen sind PC-kompatibel.

Im Flash-ROM befindet sich das SORCUS-eigene Echtzeit Multi-Tasking Betriebssystem OsX für bis zu 1024 Tasks.

Das CPU-Modul ist beliebig mischbar als Multiprozessorsystem auf einer Trägerkarte einsetzbar. Das CPU-Modul läßt sich auch im sogenannten Stand-alone Betrieb einsetzen, z.B. auf einer kundenspezifischen Trägerkarte. Dabei befinden sich die Anwendungsprogramme im Flash-ROM des CPU-Moduls. Die Kommunikation mit einem Hostrechner erfolgt dann z.B. über Ethernet (inkl. TCP/IP), über eine serielle Schnittstelle (RS-232,RS- 485,etc.) oder über ein Modem-Modul.

10.22.2. Blockschaltbild



10.22.3. Stecker A

Nachfolgend werden die Standardfunktionen des A-Steckers beschrieben. Erweiterte PIN-Funktionen sowie eine Beschreibung der Stecker B und C finden Sie in der Application Note AN081 auf unserer Homepage.

Pin	Name	Standardfunktion	Pegel	Eingang/Ausgang
1	DCD	Serielle Schnittstelle	RS-232	Eingang
2	DSR	Serielle Schnittstelle	RS-232	Eingang
3	RCV	Serielle Schnittstelle	RS-232	Eingang
4	RTS	Serielle Schnittstelle	RS-232	Ausgang
5	TMT	Serielle Schnittstelle	RS-232	Ausgang
6	CTS	Serielle Schnittstelle	RS-232	Eingang
7	DTR	Serielle Schnittstelle	RS-232	Ausgang
8	Ri	Serielle Schnittstelle	RS-232	Eingang
9	GND	Serielle Schnittstelle	GND	-
10	HOST	Debug Schnittstelle	5V-Logik gepuffert mit 1 kΩ Vorwiderstand	Eingang/Ausgang
11	/WDO	Watchdog Out	3,3V-Logik	Ausgang
12	/MRES	Manuell Reset	3,3V-Logik	Eingang
13	SPKR	Lautsprecher-Ausgang	5V-Logik	Ausgang
14	BOOT	n.c. (Pin darf nicht angeschlossen werden)	3,3V-Logik	Eingang
15	/GP0	allg. I/O	5V-Logik	Eingang/Ausgang
16	/BL1	Battery Low	3,3V-Logik	Eingang
17	SIRin	Infrarot-In	3,3V-Logik	Eingang
18	SIRout	Infrarot-Out	3,3V-Logik	Ausgang
19	LEDint	LED intern	5V-Logik gepuffert mit 270 Ω Vor-widerstand	Ausgang
20	IRQ0	Interrupt-In	5V-Logik	Eingang

Pin	Name	Standardfunktion	Pegel	Eingang/Ausgang
21	V33	Versorgungs-Spannung 3,3V	3,3 V	Ausgang
22	BAT	Batterie-Eingang	Batterie (+2,5 .. +3,6V)	Eingang
23	/STRB	Parallel-Port	5V-Logik gepuffert	Ausgang
24	/AFDT	Parallel-Port	5V-Logik gepuffert	Ausgang
25	PD0	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
26	/ERROR	Parallel-Port	5V-Logik gepuffert	Eingang
27	PD1	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
28	/INIT	Parallel-Port	5V-Logik gepuffert	Ausgang
29	PD2	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
30	/SLCTIN	Parallel-Port	5V-Logik gepuffert	Ausgang
31	PD3	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
32	/ACK	Parallel-Port	5V-Logik gepuffert	Eingang
33	PD4	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
34	BUSY	Parallel-Port	5V-Logik gepuffert	Eingang
35	PD5	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
36	PE	Parallel-Port	5V-Logik gepuffert	Eingang
37	PD6	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang

Pin	Name	Standardfunktion	Pegel	Eingang/Ausgang
38	SLCT	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
39	PD7	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
40	GND	GND	-	-

Serielle Schnittstelle (Stecker A, Pin 1..9)

Die Pins 1 bis 9 stellen eine 16550-kompatible (Standard-PC) serielle Schnittstelle zur Verfügung. Sie kann als COM1 oder COM2 konfiguriert und als RS-232 oder Infrarot-Schnittstelle betrieben werden. Beide Interfaces können in einem System auch gleichzeitig angeschlossen sein, zwischen beiden kann dann jederzeit per Software umgeschaltet werden.

Die RS-232 Schnittstelle wird z.Zt. von OsX als COM2 mit der Baudrate 38,4 kBaud (8 Datenbit, 1 Stop-Bit, keine Parität) als Host-Kommunikationsschnittstelle konfiguriert. Die Schnittstelle meldet, ob eine Gegenstelle angeschlossen ist. Wenn kein Signal anliegt, geht sie in einen Power-Down Modus. Um Strom zu sparen, kann sie per Software komplett abgeschaltet werden, dann muss auch der zugehörige Interrupt IRQ3 maskiert werden.

Als Infrarot-Schnittstelle liefert sie die Signale für den direkten Anschluss eines Infrarot-Senders/Empfängers. Die Baudrate ist hier ebenfalls programmierbar bis max. 115,2 KBit/s oder im DMA-Mode fix mit 1.152 MBit/s.

Debug Schnittstelle (Stecker A, Pin 10)

Dieser bidirektionale Pin kann für eine sehr einfache serielle Debug-Schnittstelle verwendet werden. Er liefert bzw. benötigt Logikpegel (5-Volt-kompatibel). Um ihn als serielle Schnittstelle zu nutzen, ist ein externer Pegelkonverter erforderlich, siehe Schaltung des Evaluation-Boards X-KiT-3.[1]

Watchdog Out (Stecker A, Pin 11)

Dieser Pin ist der Ausgang des Watchdogs. Er ist active Low. Wenn der Watchdog nicht rechtzeitig per Software nachgetriggert wird, geht der Ausgang auf log. 0. Soll das Ablaufen des Watchdog-Timers (siehe unten) einen Hardware-Reset des X-MAX-1 auslösen, besteht die Möglichkeit, an Stecker A die Pins 11 (Watchdog Out) und 12 (Manuel Reset) zu verbinden.

[1] Beschreibung kann bei SORCUS Computer separat bezogen werden.

Manuel Reset (Stecker A, Pin 12)

An den Eingang kann z.B. ein Taster (einpolig schließend, zwischen Pin 9 und 12) angeschlossen werden, um den X-MAX-1 manuell zurückzusetzen.

Wenn der Eingang nicht verwendet werden soll, kann er unbeschaltet bleiben, er verfügt über einen internen Pull-up-Widerstand.

Der Eingang hat Schmitt-Trigger Charakteristik.

Lautsprecher-Ausgang (Stecker A, Pin 13)

Hier kann man einen Lautsprecher anschließen. Er wird dann so angesteuert wie vom PC/AT bekannt. Näheres dazu steht in der Application Note AN087.

Allg. I/O (Stecker A, Pin 15)

Dieser Pin kann als universeller I/O-Pin eingesetzt werden. Als Eingang ist er 5-Volt-kompatibel. Nach Reset ist der interne Pull-Up Widerstand aktiv.

Über Register (Port22h, Index Abh) kann der Status abgefragt bzw. der Ausgang gesetzt werden.

Battery Low (Stecker A, Pin 16)

Dieser Eingangs-Pin mit Schmitt-Trigger-Charakteristik ist **nicht** 5-Volt-kompatibel. Er verfügt über einen externen Pull-Up Widerstand von 4,7 kΩ. Er kann so programmiert werden, dass eine positive oder negative Flanke die CPU in einen Stromspar-Modus schaltet. Der Status des Pins ist lesbar über Register.

Infrarot -In (Stecker A, Pin 17)
Infrarot-Out (Stecker A, Pin 18)

Der iRDA Eingang ist **nicht** 5-Volt-kompatibel. Er verfügt über einen abschaltbaren Pull-Down Widerstand.

Der iRDA Ausgang ist im Tri-State Modus, wenn die iRDA-Schnittstelle abgeschaltet ist. Der Pin ist **nicht** 5-Volt-kompatibel. Er verfügt über einen Pull-Down Widerstand. Ein Beispiel für den Anschluss eines iRDA-Transceivers finden Sie in der Application Note AN087.

LED intern (Stecker A, Pin 19)

Dieser Logik-Ausgang liefert den Zustand der on-board LED.

Interrupt-In (Stecker A, Pin 20)

Dieser Pin kann als Interrupt-Eingang verwendet werden. Es wird ein Interrupt bei einer positiven Flanke ausgelöst. Der Eingang ist 5-Volt-kompatibel.

Versorgungsspannung 3,3 Volt (Stecker A, Pin 21)

Hier steht die 3,3 Volt Versorgungsspannung des Moduls zur Verfügung.

Batterie-Eingang (Stecker A, Pin 22)

Hier kann eine Batterie > 2,4 Volt angeschlossen werden (max. 3,6 Volt), um die on-board Uhr (RTC) zu versorgen. Wenn das nicht erforderlich ist, sollte der Pin an GND gelegt werden.

Parallel-Port (Stecker A, Pin 23..39)

Standardfunktion der Leitungen /STRB, /AFDT, /ERROR, /INIT, /SLCTIN, /ACK, BUSY, PE, SLCT, PD0..PD7: Druckerport

Diese Pins zusammen können als Druckerport verwendet werden, der als Standard Druckerport (unidirektional), PS/2 Druckerport (bidirektional) oder als Enhanced Parallel Port (EPP) konfiguriert werden kann. Die Ein- und Ausgänge sind 5-Volt-kompatibel.

GND (Stecker A, Pin 40)

Bezugspotential für das System bzw. den Druckerport.

10.22.4. Lokale Interrupts

Interrupt-Nummer	Typ	Quelle/Erklärung
0 (00h)	Fault	CPU: Divide Error (z.B. / 0)
1 (01h)	Fault	CPU: Debug Exception (Trap/Fault)
2 (02h)	NMI	Ext.: INT 2 oder NMI
3 (03h)	Trap	CPU: Soft-Interrupt (INT 3) (1-Byte Opcode)
4 (04h)	Trap	CPU: Overflow (INT 0)
5 (05h)	Fault	CPU: Array Bounds Check
6 (06h)	Fault	CPU: Invalid Opcode
7 (07h)	Fault	CPU: Device Not Available
8 (08h)	Abort	CPU: Double Fault
9 (09h)	-	Reserviert (Intel)
10 (0ah)	Fault	CPU: Invalid TSS (Protected Mode)
11 (0bh)	Fault	CPU: Segment Not Present
12 (0ch)	Fault	CPU: Stack Fault
13 (0dh)	Fault	CPU: General Protection Fault
14 (0eh)	Fault	CPU: Page Fault
15 (0fh)	-	Reserviert (Intel)
16 (10h)	Fault	CPU: Floating Point Error (tritt bei i486SX nicht auf)
17 (11h)	Fault	CPU: Alignment Check
18 - 31 (12h - 1fh)	-	Reserviert (Intel)
32 - 119 (20h - 77h)	Trap	Soft-Interrupt (2-Byte Opcode), reserviert für SORCUS
120 - 127 (78h - 7fh)	Hardware	**Master-Interrupt-Controller** auf dem X-MAX-1: Master-0 bis Master-7 (siehe Tab. Seite 10-265)
128 - 143 (80h - 8fH)	-	Reserviert
144 - 151 (90h - 97h)	Hardware	**Slave-Interrupt-Controller** auf dem X-MAX-1 an Master-2 vom Master-Interrupt-Controller (siehe Tabelle unten)
152 (98h)	Hardware	RTC-Interrupt: Update Ended
153 (99h)	Hardware	RTC-Interrupt: Alarm
154 (9ah)	Hardware	RTC-Interrupt: Periodic
155 - 207 (9bh - cfh)	-	Reserviert

Interrupt-Nummer	Typ	Quelle/Erklärung
208 - 239 (d0h - efh)	Trap	Soft-Interrupt (2-Byte Opcode), reserviert für SORCUS
240 - 255 (f0h - ffh)	Trap	Soft-Interrupt (2-Byte Opcode), frei für Anwender

Hardware Interrupts

Prio-rität	Interrupt Dez.	Hex	Quelle	Beschreibung	Interrupt-Eingang
Hoch	2	02h	NMI	Nicht maskierbarer Interrupt	NMI (CPU)
	120	78h	TIMER-A	Zeitgeber A	Master-0
	121	79h	Keyboard	Keyboard	Master-1
	122	7ah	Slave	von Slave-Controller	Master-2
	144	90h	Uhr	Uhr	Slave-0
	145	91h	PIRQ-2	X-Bus / Grafik	Slave-1
	146	92h	PIRQ-3	X-Bus	Slave-2
	147	93h	PIRQ-5	Zeitgeber-C	Slave-3
	148	94h	Mouse	Mouse	Slave-4
	149	95h	PIRQ-0	Externer Pin	Slave-5
	150	96h	PIRQ-1	Host Pin	Slave-6
	151	97h	PC-Card	PC-Card	Slave-7
	123	7bh	COM2	COM2	Master-3
	124	7ch	PIRQ-6	X-Bus / COM2	Master-4
	125	7dh	PIRQ-7	X-Bus / EPP	Master-5
	126	7eh	PIRQ-4	Mailbox RBF	Master-6
Niedrig	127	7fh	EPP	EPP	Master-7

10.22.5. Direkter Zugriff auf I/O-Ports bzw. CPU-Register

Die auf dem X-MAX-1 verwendete CPU AMD SC400 bietet über die Standard-PC-Funktionalität (486-kompatibel) hinaus eine Vielzahl von speziellen Features. Nur ein Teil davon ist bzw. wird zukünftig direkt durch die SORCUS-Bibliotheken bzw. Modul-Device-Treiber unterstützt. Sollen weitere, nicht von dieser Software unterstützte Funktionen verwendet werden, muss auf die direkte Programmierung der Register zurückgegriffen werden. Dazu werden die AMD-Handbücher benötigt. Diese stehen unter www.amd.com zur Verfügung.

In diesem Fall kann SORCUS jedoch keine Gewähr dafür übernehmen, dass bei einem Nachfolge-Modell des X-MAX-1 alle Features der derzeitigen CPU SC400 enthalten sein werden. Eine solche Garantie auf Kompatibilität gibt es nur bezüglich der von den SORCUS-Bibliotheken und Modul-Device-Treibern unterstützten Funktionen.

Wenn die Register der CPU direkt angesprochen werden müssen, ist folgendes zu beachten:

Auf andere Adressen als die in den AMD-Handbüchern dokumentierten darf nicht zugegriffen werden. Alle I/O-Zugriffe müssen über die nachfolgend beschriebenen Bibliotheksfunktionen erfolgen, Zugriffe über inport bzw. outport-Befehle sind nicht zulässig.

max_read_io_byte
max_read_io_word
max_read_io_long — Lesen eines Registers

Pascal

```
FUNCTION max_read_io_byte (hModul: MAXMODHND; port: WORD; VAR data: BYTE) : MAX_ERROR;

FUNCTION max_read_io_word (hModul: MAXMODHND; port: WORD; VAR data: WORD) : MAX_ERROR;

FUNCTION max_read_io_long (hModul: MAXMODHND; port: WORD; VAR data: LONGWORD) : MAX_ERROR;
```

C

```
MAX_ERROR max_read_io_byte (MAXMODHND hModul, USHORT port, UCHAR *data);

MAX_ERROR max_read_io_word (MAXMODHND hModul, USHORT port, USHORT *data);

MAX_ERROR max_read_io_long (MAXMODHND hModul, USHORT port, ULONG *data);
```

VB

Declare Function max_read_io_byte Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByRef data As Byte) As Integer

Declare Function max_read_io_word Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByRef data As Integer) As Integer

Declare Function max_read_io_long Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByRef data As Long) As Integer

Verwendung Host-PC, X-MAX-1 OsX

Funktion Mit diesen Funktionen können einzelne Adressen im I/O-Bereich eines CPU-Moduls Byte-, Wort- oder Doppelwortweise gelesen werden. Sie dürfen nur für die sogenannten „direct-mapped“-Register verwendet werden (das sind die PC/AT kompatiblen Registeradressen).

Parameter

hModul Handle auf das CPU-Modul

port Registeradresse im I/O-Bereich der CPU

data Zeiger auf eine Variable, in die der gelesene Wert eingetragen wird.

max_write_io_byte
max_write_io_word
max_write_io_long Schreiben eines Registers

Pascal

FUNCTION max_write_io_byte (hModul: MAXMODHND; port: WORD; data: BYTE) : MAX_ERROR;

FUNCTION max_write_io_word (hModul: MAXMODHND; port: WORD; data: WORD): MAX_ERROR;

FUNCTION max_write_io_long (hModul: MAXMODHND; port: WORD; data: LONGWORD): MAX_ERROR;

C

MAX_ERROR max_write_io_byte (MAXMODHND hModul, USHORT port, UCHAR data);

MAX_ERROR max_write_io_word (MAXMODHND hModul, USHORT port, USHORT data);

MAX_ERROR max_write_io_long (MAXMODHND hModul, USHORT port, ULONG data);

VB Declare Function max_write_io_byte Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal data As Byte) As Integer

Declare Function max_write_io_word Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal data As Integer) As Integer

Declare Function max_write_io_long Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal data As Long) As Integer

Verwendung Host-PC, X-MAX-1 OsX

Funktion Mit diesen Funktionen können Ports im I/O-Bereich eines CPU-Moduls Byte-, Wort- oder Doppelwortweise beschrieben werden. Sie dürfen nur für die sogenannten „direct-mapped“-Register verwendet werden.

Parameter *hModul* Handle auf das CPU-Modul

port Registeradresse im I/O-Bereich der CPU

data Wert

max_read_io_indexed Lesen eines indizierten Registers

Pascal FUNCTION max_read_io_indexed (hModul: MAXMODHND; port: WORD; index: BYTE; VAR data: BYTE) : MAX_ERROR;

C MAX_ERROR max_read_io_indexed (MAXMODHND hModul, USHORT port, UCHAR index, UCHAR *data);

VB Declare Function max_read_io_indexed Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal index As Byte, ByRef data As Byte) As Integer

Verwendung Host-PC, X-MAX-1 OsX

Funktion Mit dieser Funktion wird ein sogenanntes „indexed“-Register gelesen. Dazu gehören die meisten der SC400-spezifischen Konfigurations- und Kontroll-Register.

Parameter	*hModul*	Handle auf das CPU-Modul
	port	I/O-Registeradresse. Folgende Werte sind zulässig: 22h: ChipSetup and Control 70h: Real-Time-Clock und CMOS-RAM 3B4h: LCD Graphics Controller im monochrom Mode 3D4h: LCD Graphics Controller im CGA-Mode 3E0h: PC-Card Controller Der Wert *index* wird in die Adresse *port* hineingeschrieben und von der Adresse *port+1* wird gelesen.
	index	Subadresse
	data	Zeiger auf eine Variable, in die der gelesene Wert eingetragen wird.

max_write_io_indexed Beschreiben eines indizierten Registers

Pascal	FUNCTION max_write_io_indexed (hModul: MAXMODHND; port: WORD; index: BYTE; mask: BYTE; data: BYTE) : MAX_ERROR;
C	MAX_ERROR max_write_io_indexed (MAXMODHND hModul, USHORT port, UCHAR index, UCHAR mask, UCHAR data);
VB	Declare Function max_write_io_indexed Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal index As Byte, ByVal mask As Byte, ByVal data As Byte) As Integer
Verwendung	Host-PC, X-MAX-1 OsX
Funktion	Mit dieser Funktion kann ein sogenanntes „indexed“-Register beschrieben werden. Dazu gehören die meisten der SC400-spezifischen Konfigurations- und Kontroll-Register. Verwendet wird in dieser Funktion eine Read-Modify-Write-Sequenz: Zunächst wird das durch *index* und *port* spezifizierte Register gelesen, anschließend wird der gelesene Wert mit dem durch *mask* gegebenen Wert verundet und dann mit dem Wert in *data* verodert. Das Ergebnis wird wieder in das Register geschrieben. Dadurch können mit einem Funktionsaufruf einzelne Bits eines Registers gesetzt oder gelöscht werden.

Parameter	*hModul*	Handle auf das CPU-Modul
	port	I/O-Registeradresse. Folgende Werte sind zulässig: 22h: ChipSetup and Control 70h: Real-Time-Clock und CMOS-RAM 3B4h: LCD Graphics Controller im monochrom Mode 3D4h: LCD Graphics Controller im CGA-Mode 3E0h: PC-Card Controller Der Wert *index* wird in die Adresse *port* hineingeschrieben und in die Adresse *port+1* wird der Datenwert geschrieben.
	index	Subadresse
	mask	Wert der Bitmaske, mit der der vom durch *port* und *index* gegebenen Register gelesene Wert verunded wird. Dadurch kann z.B. ein einzelnes Bit ausmaskiert werden.
	data	Wert der Bitmaske, mit der der vom durch *port* und *index* gegebenen Register gelesene Wert verodert wird.

10.22.6. Watchdog-Timer und Spannungsüberwachung

Zur Überwachung eines ordnungsgemäßen Programmablaufs sind auf jedem X-MAX-1 verschieden Überwachungsmöglichkeiten vorhanden:

Einrichtung	**spricht an, wenn**	**bewirkt**
Watchdog	Watchdog-Timer nicht rechtzeitig nachgetriggert wird	NMI
Spannungs-überwachung A		NMI
Spannungs-überwachung B		Hardware-Reset, Uhr auf Batterie-Pufferung

Watchdog-Timer

Zur Überwachung der laufenden Echtzeit-Software (Tasks) stellt der X-MAX-1 einen Watchdog-Timer zur Verfügung. Dieser muss durch eine Task in regelmäßigen Abständen neu gesetzt werden (re-triggern). Um ein gewünschtes Zeitschema einzuhalten, sollte die Re-Triggerung an der "zeitkritischsten" Stelle aller installierten Tasks erfolgen (nur an einer Stelle!).

In Anwendungen, in denen die Interrupt-Belastung des X-MAX-1 sehr hoch ist, kann es z.B. dazu kommen, dass NI-Tasks nur sehr selten vom Betriebssystem aufgerufen werden. Soll überwacht werden, dass eine bestimmte NI-Task trotzdem rechtzeitig an die Reihe kommt, muss die Re-Triggerung des Watchdog-Timers in der Hauptprozedur dieser NI-Task stattfinden.

Kommt das Betriebssystem nicht dazu, die Task, die die Re-Triggerung übernimmt, rechtzeitig aufzurufen, weil ein Programm abgestürzt ist, oder andere Tasks zu viel Zeit beanspruchen, bleibt die Re-Triggerung aus. Dadurch läuft der Watchdog-Timer ab. Der X-MAX-1 reagiert darauf mit einem NMI-Interrupt. Wenn für diesen Interrupt eine Task installiert ist und der NMI freigegeben ist, ruft das OsX-Betriebssystem die Hauptprozedur dieser Task auf. Dort besteht die Möglichkeit, einen sicheren Betriebszustand herzustellen, Daten zu retten oder den Host-PC durch einen Service-Request zu informieren.

Die Watchdog-Timeout-Zeit, innerhalb der nachgetriggert werden muss, beträgt ca. 1,6 s.

Um die Funktionalität des Watchdogs nutzen zu können, sind folgende Schritte notwendig:

1. Der Watchdog wird standardmäßig vom Betriebssystem in einer NI-Task nachgetriggert. Um dies abzuschalten, ist der Betriebssystem-Parameter 214 (Byte) = 0 zu setzen (Auto-Retrigger = off).
2. Anstelle des Betriebssystems muss jetzt eine Anwender-Task die Nachtriggerung des Watchdog-Timers übernehmen. Das kann mit Hilfe der Bibliotheksfunktion **max_trigger_watchdog** geschehen.
3. Eine II-Task sollte unter dem NMI-Interrupt installiert werden, um bei Ablauf des Watchdog-Timers reagieren zu können. Zu beachten ist, dass der NMI demaskiert werden muss (**max_unmask_int(NMI)**).

Spannungsüberwachung

Unterschreitet die Versorgungsspannung des Moduls einen kritischen Wert, dann wird mit einem NMI reagiert. In einer unter dem NMI installierten Task kann dann versucht werden, wichtige Daten zu retten und einen sicheren Betriebszustand herzustellen.

Sinkt die Versorgungsspannung noch weiter ab, wird ein Hardware-Reset durchgeführt.

Auslesen der NMI-Interruptquelle

Werden mehrere NMI-Quellen freigegeben, kann die Ursache eines NMI durch Aufruf einer Bibliotheksfunktion (**max_get_nmi_reason**) herausgefunden werden. Dieser liefert einen 16-Bit Wert zurück. Darin ist:

Bit 9 des gelesenen Statuswortes = 0, wenn die Spannungsüberwachung den NMI ausgelöst hat.

Bit 10 = 1, wenn der Watchdog-Timer einen NMI verursacht hat.

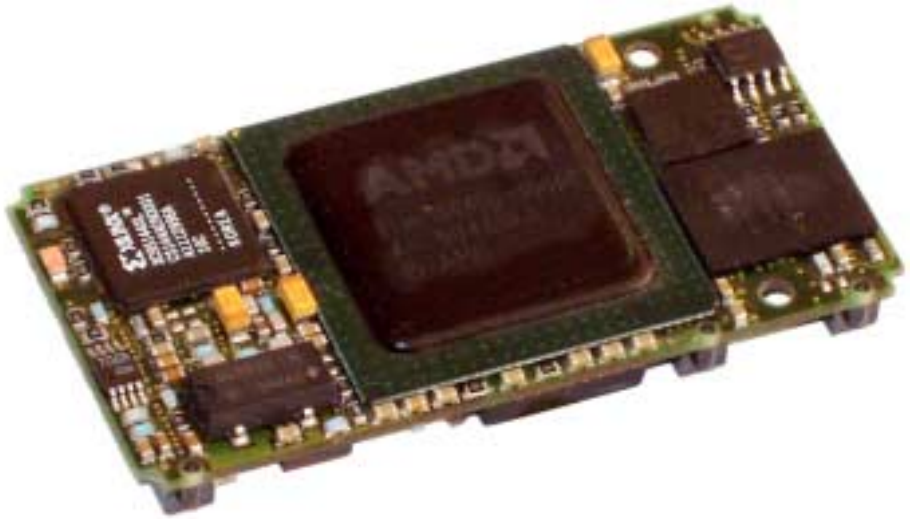
10.22.7. Besondere Eigenschaften

CPU:	486-100, 486-33
RAM:	2 MByte bis 16 MByte
ROM:	Flash-EPROM, 4 MByte bis 16 MByte
EEPROM:	4kByte, seriell
Timer:	3 x 16 Bit Breite, Eingangsfrequenz 1,1892 MHz, Timer A und C interruptfähig, Echtzeituhr.
Interrupts:	15 Eingänge, davon 4 an X-Bus, 1 an Mailbox-Register, 2 an externen Eingängen, 2 an Timer, 1 an serielle Schnittstellen, 1 an Uhr
Serielle Schnittstellen:	RS-232 Schnittstelle mit den üblichen Modem-Steuersignalen (nur X-MAX-1), 16550-kompatibel
Echtzeituhr:	Datum (Tag, Monat, Jahr, Wochentag) und Uhrzeit (Std., Min. Sek.), batteriepufferbar, interruptfähig
Multi-Tasking-Betriebssystem:	Voll echtzeitfähig, max. 1024 Tasks, Interrupt-, Timer-initiierte und Nicht-Interrupt-Tasks. Im Flash des X-MAX-1 enthalten, wird ins RAM kopiert, belegt 64 KB RAM.
Stromaufnahme:	+3,3 V: 594 mA (typ. bei 100 MHz)
Abmessungen:	29 x 58 x 8 mm
Temperaturverträglichkeit:	0 bis 55°C (optional –40° .. + 85° C)
Luftfeuchtigkeitsverträglichkeit:	5 bis 95% (nicht kondensierend)

Notizen

X-MAX-E

Der kleinste PC der Welt mit Ethernet



10.23. X-MAX-E, Rev. D

Inhaltsverzeichnis

10.23.1. Beschreibung

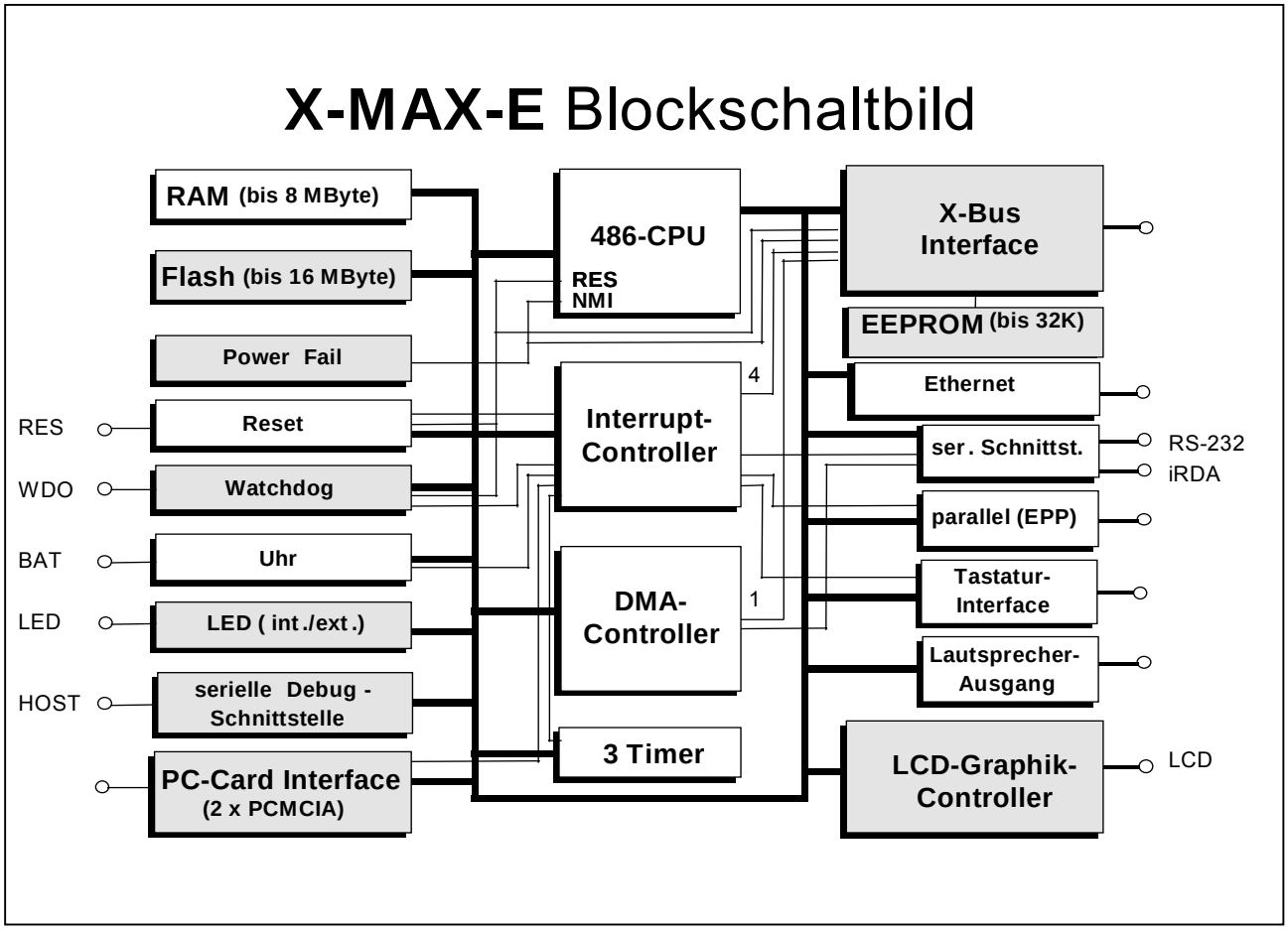
Das CPU-Modul X-MAX-E ist mit einer 486-CPU bestückt und enthält einen 8 KByte großen Cache. Die Taktfrequenz der lokalen CPU beträgt je nach Version zwischen 33 MHz und 100 MHz (interner Takt).

Das Modul ist bereits mit umfangreicher Peripherie ausgestattet. Außer Flash-ROM (max.16 MByte) und RAM (max. 8 MByte) verfügt es über 3 Timer, Interrupt-und DMA-Controller, eine serielle Schnittstelle (RS-232/422/485) und eine Druckerschnittstelle, eine Uhr, einen Watch-Dog und Spannungsüberwachung und eine 10 MBit Ethernet Schnittstelle (10BaseT). Die Uhr kann mit einer externen Batterie gepuffert werden. Alle Schnittstellen sind PC-kompatibel.

Im Flash-ROM befindet sich das SORCUS-eigene Echtzeit Multi-Tasking Betriebssystem OsX für bis zu 1024 Tasks.

Das CPU-Modul ist beliebig mischbar als Multiprozessorsystem auf einer Trägerkarte einsetzbar. Das CPU-Modul läßt sich auch im sogenannten Stand-alone Betrieb einsetzen, z.B. auf einer kundenspezifischen Trägerkarte. Dabei befinden sich die Anwendungsprogramme im Flash-ROM des CPU-Moduls. Die Kommunikation mit einem Hostrechner erfolgt dann z.B. über Ethernet (inkl. TCP/IP), über eine serielle Schnittstelle (RS-232, RS-485, etc.) oder über ein Modem-Modul.

10.23.2. Blockschaltbild



10.23.3. Stecker A

Nachfolgend werden die Standardfunktionen des A-Steckers beschrieben. Erweiterte PIN-Funktionen sowie eine Beschreibung der Stecker B und C finden Sie in der Application Note AN081 auf unserer Homepage.

Pin	Name	Standardfunktion	Pegel	Eingang/Ausgang
1	TD+	Ethernet-Schnittstelle	-	Ausgang
2	TD-	Ethernet-Schnittstelle	-	Ausgang
3	RD+	Ethernet-Schnittstelle	-	Eingang
4	RD-	Ethernet-Schnittstelle	-	Eingang
5	GND	Serielle Schnittstelle	GND	Eingang/Ausgang
6	RCV	Serielle Schnittstelle	RS-232	Eingang
7	RTS	Serielle Schnittstelle	RS-232	Ausgang
8	TMT	Serielle Schnittstelle	RS-232	Eingang
9	CTS	Serielle Schnittstelle	RS-232	Eingang
10	HOST	Debug Schnittstelle	Logik gepuffert mit 1 kΩ Vorwiderstand	Eingang/Ausgang
11	/WDO	Watchdog Out	3,3V-Logik	Ausgang
12	/MRES	Manuell Reset	3,3V-Logik	Eingang
13	SPKR	Lautsprecher-Ausgang	5V-Logik	Ausgang
14	BOOT	n.c. (Pin darf nicht angeschlossen werden)	3,3V-Logik	Eingang
15	/GP0	allg. I/O	5V-Logik	Eingang/Ausgang
16	/BL1	Battery Low	3,3V-Logik	Eingang
17	SIRin	Infrarot-In	3,3V-Logik	Eingang
18	SIRout	Infrarot-Out	3,3V-Logik	Ausgang
19	LEDint	LED intern	5V-Logik gepuffert mit 270 Ω Vorwiderstand	Ausgang
20	IRQ0	Interrupt-In	5V-Logik	Eingang/Ausgang

Pin	Name	Standardfunktion	Pegel	Eingang/Ausgang
21	V33	Versorgungs-Spannung 3,3V	3,3 V	Ausgang
22	BAT	Batterie-Eingang	Batterie (+2,5 .. +3,6V)	Eingang
23	/STRB	Parallel-Port	5V-Logik gepuffert	Ausgang
24	/AFDT	Parallel-Port	5V-Logik gepuffert	Ausgang
25	PD0	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
26	/ERROR	Parallel-Port	5V-Logik gepuffert	Eingang
27	PD1	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
28	/INIT	Parallel-Port	5V-Logik gepuffert	Ausgang
29	PD2	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
30	/SLCTIN	Parallel-Port	5V-Logik gepuffert	Ausgang
31	PD3	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
32	/ACK	Parallel-Port	5V-Logik gepuffert	Eingang
33	PD4	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
34	BUSY	Parallel-Port	5V-Logik gepuffert	Eingang
35	PD5	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
36	PE	Parallel-Port	5V-Logik gepuffert	Eingang
37	PD6	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
38	SLCT	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
39	PD7	Parallel-Port	5V-Logik gepuffert	Eingang/Ausgang
40	GND	GND	-	-

Ethernet-Schnittstelle (Stecker A, Pin 1..4)

Die Pins 1..4 stellen einen 10Base-T Ethernet Anschluss zur Verfügung.

Serielle Schnittstelle (Stecker A, Pin 5..9)

Die Pins 5..9 stellen eine 16550-kompatible (Standard-PC) serielle Schnittstelle zur Verfügung. Sie kann als COM1 oder COM2 konfiguriert und als RS-232/422/485 oder Infrarot-Schnittstelle betrieben werden. Beide Interfaces können in einem System auch gleichzeitig angeschlossen sein, zwischen beiden kann dann jederzeit per Software umgeschaltet werden.

Debug Schnittstelle (Stecker A, Pin 10)

Dieser bidirektionale Pin kann für eine sehr einfache serielle Debug-Schnittstelle verwendet werden. Er liefert bzw. benötigt Logikpegel (5-Volt-kompatibel). Um ihn als serielle Schnittstelle zu nutzen, ist ein externer Pegelkonverter erforderlich, siehe Schaltung des Evaluation-Boards X-KiT-3.[1]

Watchdog Out (Stecker A, Pin 11)

Dieser Pin ist der Ausgang des Watchdogs. Er ist active Low. Wenn der Watchdog nicht rechtzeitig per Software nachgetriggert wird, geht der Ausgang auf log. 0. Soll das Ablaufen des Watchdog-Timers (siehe unten) einen Hardware-Reset des X-MAX-E auslösen, besteht die Möglichkeit, an Stecker A die Pins 11 (Watchdog Out) und 12 (Manuell Reset) zu verbinden.

Manuell Reset (Stecker A, Pin 12)

An den Eingang kann z.B. ein Taster (einpolig schließend, zwischen Pin 9 und 12) angeschlossen werden, um den X-MAX-E manuell zurückzusetzen.

Wenn der Eingang nicht verwendet werden soll, kann er unbeschaltet bleiben, er verfügt über einen internen Pull-up-Widerstand.

Der Eingang hat Schmitt-Trigger Charakteristik.

Lautsprecher-Ausgang (Stecker A, Pin 13)

Hier kann man einen Lautsprecher anschließen. Er wird dann so angesteuert wie vom PC/AT bekannt. Näheres dazu steht in der Application Note AN087.

[1] Beschreibung kann bei SORCUS Computer separat bezogen werden.

Allg. I/O (Stecker A, Pin 15)

Dieser Pin kann als universeller I/O-Pin eingesetzt werden. Als Eingang ist er 5-Volt-kompatibel. Nach Reset ist der interne Pull-Up Widerstand aktiv. Die Funktion des Pins wird über ein Register Bit eingestellt.

Battery Low (Stecker A, Pin 16)

Dieser Eingangs-Pin mit Schmitt-Trigger-Charakteristik ist **nicht** 5-Volt-kompatibel. Er verfügt über einen externen Pull-Up Widerstand von 4,7 kΩ. Er kann so programmiert werden, dass eine positive oder negative Flanke die CPU in einen Stromspar-Modus schaltet. Der Status des Pins ist lesbar über Register.

Infrarot-In (Stecker A, Pin 17)
Infrarot-Out (Stecker A, Pin 18)

Der iRDA Eingang ist **nicht** 5-Volt-kompatibel. Er verfügt über einen abschaltbaren Pull-Down Widerstand.

Der iRDA Ausgang ist im Tri-State Modus, wenn die iRDA-Schnittstelle abgeschaltet ist. Der Pin ist **nicht** 5-Volt-kompatibel. Er verfügt über einen Pull-Down Widerstand. Ein Beispiel für den Anschluss eines iRDA-Transceivers finden Sie in der Application Note AN087.

LED intern (Stecker A, Pin 19)

Dieser Logik-Ausgang liefert den Zustand der on-board LED.

Interrupt-In (Stecker A, Pin 20)

Dieser Pin kann als Interrupt-Eingang verwendet werden. Es wird ein Interrupt bei einer positiven Flanke ausgelöst. Der Eingang ist 5-Volt-kompatibel.

Versorgungsspannung 3,3 V (Stecker A, Pin 21)

Hier steht die 3,3 Volt Versorgungsspannung des Moduls zur Verfügung.

Batterie-Eingang (Stecker A, Pin 22)

Hier kann eine Batterie > 2,4 Volt angeschlossen werden (max. 3,6 Volt), um die on-board Uhr (RTC) zu versorgen. Wenn das nicht erforderlich ist, sollte der Pin an GND gelegt werden.

Parallel-Port (Stecker A, Pin 23..39)

Standardfunktion der Leitungen /STRB, /AFDT, /ERROR, /INIT, /SLCTIN, /ACK, BUSY, PE, SLCT, PD0..PD7: Druckerport

Diese Pins zusammen können als Druckerport verwendet werden, der als Standard Druckerport (unidirektional), PS/2 Druckerport (bidirektional) oder als Enhanced Parallel Port (EPP) konfiguriert werden kann. Die Ein- und Ausgänge sind 5-Volt-kompatibel.

GND (Stecker A, Pin 40)

Bezugspotential für das System bzw. den Druckerport.

10.23.4. Lokale Interrupts

Interrupt-Nummer	Typ	Quelle/Erklärung
0 (00h)	Fault	CPU: Divide Error (z.B. / 0)
1 (01h)	Fault	CPU: Debug Exception (Trap/Fault)
2 (02h)	NMI	Ext.: INT 2 oder NMI
3 (03h)	Trap	CPU: Soft-Interrupt (INT 3) (1-Byte Opcode)
4 (04h)	Trap	CPU: Overflow (INT 0)
5 (05h)	Fault	CPU: Array Bounds Check
6 (06h)	Fault	CPU: Invalid Opcode
7 (07h)	Fault	CPU: Device Not Available
8 (08h)	Abort	CPU: Double Fault
9 (09h)	-	Reserviert (Intel)
10 (0ah)	Fault	CPU: Invalid TSS (Protected Mode)
11 (0bh)	Fault	CPU: Segment Not Present
12 (0ch)	Fault	CPU: Stack Fault
13 (0dh)	Fault	CPU: General Protection Fault
14 (0eh)	Fault	CPU: Page Fault
15 (0fh)	-	Reserviert (Intel)
16 (10h)	Fault	CPU: Floating Point Error (tritt bei i486SX nicht auf)
17 (11h)	Fault	CPU: Alignment Check
18 - 31 (12h - 1fh)	-	Reserviert (Intel)
32 - 119 (20h - 77h)	Trap	Soft-Interrupt (2-Byte Opcode), reserviert für SORCUS
120 - 127 (78h - 7fh)	Hardware	**Master-Interrupt-Controller** auf dem X-MAX-E: Master-0 bis Master-7 (siehe Tabelle unten)
128 - 143 (80h - 8fH)	-	Reserviert

Interrupt-Nummer	Typ	Quelle/Erklärung
144 - 151 (90h - 97h)	Hardware	**Slave-Interrupt-Controller** auf dem X-MAX-E an Master-2 vom Master-Interrupt-Controller (siehe Tabelle unten)
152 (98h)	Hardware	RTC-Interrupt: Update Ended
153 (99h)	Hardware	RTC-Interrupt: Alarm
154 (9ah)	Hardware	RTC-Interrupt: Periodic
155 - 207 (9bh - cfh)	-	Reserviert
208 - 239 (d0h - efh)	Trap	Soft-Interrupt (2-Byte Opcode), reserviert für SORCUS
240 - 255 (f0h - ffh)	Trap	Soft-Interrupt (2-Byte Opcode), frei für Anwender

Hardware Interrupts

Prio-rität	Interrupt Dez.	Hex	Quelle	Beschreibung	Interrupt-Eingang
Hoch	2	02h	NMI	Nicht maskierbarer Interrupt	NMI (CPU)
	120	78h	TIMER-A	Zeitgeber A	Master-0
	121	79h	Keyboard	Keyboard	Master-1
	122	7ah	Slave	von Slave-Controller	Master-2
	144	90h	Uhr	Uhr	Slave-0
	145	91h	PIRQ-2	X-Bus / Grafik	Slave-1
	146	92h	PIRQ-3	X-Bus	Slave-2
	147	93h	PIRQ-5	Zeitgeber-C	Slave-3
	148	94h	Mouse	Mouse	Slave-4
	149	95h	PIRQ-0	Externer Pin	Slave-5
	150	96h	PIRQ-1	Host Pin	Slave-6
	151	97h	PC-Card	PC-Card	Slave-7
	123	7bh	COM2	COM2	Master-3
	124	7ch	PIRQ-6	X-Bus / COM2	Master-4
	125	7dh	PIRQ-7	X-Bus / LAN-Controller	Master-5
	126	7eh	PIRQ-4	Mailbox RBF	Master-6
Niedrig	127	7fh	EPP	EPP	Master-7

10.23.5. Direkter Zugriff auf I/O-Ports bzw. CPU-Register

Die auf dem X-MAX-E verwendete CPU AMD SC400 bietet über die Standard-PC-Funktionalität (486-kompatibel) hinaus eine Vielzahl von speziellen Features. Nur ein Teil davon ist bzw. wird zukünftig direkt durch die SORCUS-Bibliotheken bzw. Modul-Device-Treiber unterstützt. Sollen weitere, nicht von dieser Software unterstützte Funktionen verwendet werden, muss auf die direkte Programmierung der Register zurückgegriffen werden. Dazu werden die AMD-Handbücher benötigt. Diese stehen unter www.amd.com zur Verfügung.

In diesem Fall kann SORCUS jedoch keine Gewähr dafür übernehmen, dass bei einem Nachfolge-Modell des X-MAX-E alle Features der derzeitigen CPU SC400 enthalten sein werden. Eine solche Garantie auf Kompatibilität gibt es nur bezüglich der von den SORCUS-Bibliotheken und Modul-Device-Treibern unterstützten Funktionen.

Wenn die Register der CPU direkt angesprochen werden müssen, ist folgendes zu beachten:

Auf andere Adressen als die in den AMD-Handbüchern dokumentierten darf nicht zugegriffen werden. Alle I/O-Zugriffe müssen über die nachfolgend beschriebenen Bibliotheksfunktionen erfolgen, Zugriffe über inport bzw. outport-Befehle sind nicht zulässig.

max_read_io_byte
max_read_io_word
max_read_io_long — Lesen eines Registers

Pascal

```
FUNCTION max_read_io_byte (hModul: MAXMODHND; port: WORD; VAR data: BYTE) : MAX_ERROR;

FUNCTION max_read_io_word (hModul: MAXMODHND; port: WORD; VAR data: WORD) : MAX_ERROR;

FUNCTION max_read_io_long (hModul: MAXMODHND; port: WORD; VAR data: LONGWORD) : MAX_ERROR;
```

C

```
MAX_ERROR max_read_io_byte (MAXMODHND hModul, USHORT port, UCHAR *data);

MAX_ERROR max_read_io_word (MAXMODHND hModul, USHORT port, USHORT *data);

MAX_ERROR max_read_io_long (MAXMODHND hModul, USHORT port, ULONG *data);
```

VB

Declare Function max_read_io_byte Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByRef data As Byte) As Integer

Declare Function max_read_io_word Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByRef data As Integer) As Integer

Declare Function max_read_io_long Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByRef data As Long) As Integer

Verwendung Host-PC, X-MAX-E OsX

Funktion Mit diesen Funktionen können einzelne Adressen im I/O-Bereich eines CPU-Moduls Byte-, Wort- oder Doppelwortweise gelesen werden. Sie dürfen nur für die sogenannten „direct-mapped“-Register verwendet werden (das sind die PC/AT kompatiblen Registeradressen).

Parameter

hModul Handle auf das CPU-Modul

port Registeradresse im I/O-Bereich der CPU

data Zeiger auf eine Variable, in die der gelesene Wert eingetragen wird.

max_write_io_byte
max_write_io_word
max_write_io_long Schreiben eines Registers

Pascal

FUNCTION max_write_io_byte (hModul: MAXMODHND; port: WORD; data: BYTE) : MAX_ERROR;

FUNCTION max_write_io_word (hModul: MAXMODHND; port: WORD; data: WORD): MAX_ERROR;

FUNCTION max_write_io_long (hModul: MAXMODHND; port: WORD; data: LONGWORD): MAX_ERROR;

C

MAX_ERROR max_write_io_byte (MAXMODHND hModul, USHORT port, UCHAR data);

MAX_ERROR max_write_io_word (MAXMODHND hModul, USHORT port, USHORT data);

MAX_ERROR max_write_io_long (MAXMODHND hModul, USHORT port, ULONG data);

VB Declare Function max_write_io_byte Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal data As Byte) As Integer

Declare Function max_write_io_word Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal data As Integer) As Integer

Declare Function max_write_io_long Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal data As Long) As Integer

Verwendung Host-PC, X-MAX-E OsX

Funktion Mit diesen Funktionen können Ports im I/O-Bereich eines CPU-Moduls Byte-, Wort- oder Doppelwortweise beschrieben werden. Sie dürfen nur für die sogenannten „direct-mapped“-Register verwendet werden.

Parameter *hModul* Handle auf das CPU-Modul

port Registeradresse im I/O-Bereich der CPU

data Wert

max_read_io_indexed Lesen eines indizierten Registers

Pascal FUNCTION max_read_io_indexed (hModul: MAXMODHND; port: WORD; index: BYTE; VAR data: BYTE) : MAX_ERROR;

C MAX_ERROR max_read_io_indexed (MAXMODHND hModul, USHORT port, UCHAR index, UCHAR *data);

VB Declare Function max_read_io_indexed Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal index As Byte, ByRef data As Byte) As Integer

Verwendung Host-PC, X-MAX-E OsX

Funktion Mit dieser Funktion wird ein sogenanntes „indexed“-Register gelesen. Dazu gehören die meisten der SC400-spezifischen Konfigurations- und Kontroll-Register.

Parameter	*hModul*	Handle auf das CPU-Modul
	port	I/O-Registeradresse. Folgende Werte sind zulässig: 22h: ChipSetup and Control 70h: Real-Time-Clock und CMOS-RAM 3B4h: LCD Graphics Controller im monochrom Mode 3D4h: LCD Graphics Controller im CGA-Mode 3E0h: PC-Card Controller Der Wert *index* wird in die Adresse *port* hineingeschrieben und von der Adresse *port+1* wird gelesen.
	index	Subadresse
	data	Zeiger auf eine Variable, in die der gelesene Wert eingetragen wird.

max_write_io_indexed Beschreiben eines indizierten Registers

Pascal	FUNCTION max_write_io_indexed (hModul: MAXMODHND; port: WORD; index: BYTE; mask: BYTE; data: BYTE) : MAX_ERROR;
C	MAX_ERROR max_write_io_indexed (MAXMODHND hModul, USHORT port, UCHAR index, UCHAR mask, UCHAR data);
VB	Declare Function max_write_io_indexed Lib "maxw32.dll" (ByVal hModul As Long, ByVal port As Integer, ByVal index As Byte, ByVal mask As Byte, ByVal data As Byte) As Integer
Verwendung	Host-PC, X-MAX-E OsX
Funktion	Mit dieser Funktion kann ein sogenanntes „indexed“-Register beschrieben werden. Dazu gehören die meisten der SC400-spezifischen Konfigurations- und Kontroll-Register. Verwendet wird in dieser Funktion eine Read-Modify-Write-Sequenz: Zunächst wird das durch *index* und *port* spezifizierte Register gelesen, anschließend wird der gelesene Wert mit dem durch *mask* gegebenen Wert verundet und dann mit dem Wert in *data* verodert. Das Ergebnis wird wieder in das Register geschrieben. Dadurch können mit einem Funktionsaufruf einzelne Bits eines Registers gesetzt oder gelöscht werden.

Parameter *hModul* Handle auf das CPU-Modul

port I/O-Registeradresse. Folgende Werte sind zulässig:
22h: ChipSetup and Control
70h: Real-Time-Clock und CMOS-RAM
3B4h: LCD Graphics Controller im monochrom Mode
3D4h: LCD Graphics Controller im CGA-Mode
3E0h: PC-Card Controller

Der Wert *index* wird in die Adresse *port* hineingeschrieben und in die Adresse *port+1* wird der Datenwert geschrieben.

index Subadresse

mask Wert der Bitmaske, mit der der vom durch *port* und *index* gegebenen Register gelesene Wert verunded wird. Dadurch kann z.B. ein einzelnes Bit ausmaskiert werden.

data Wert der Bitmaske, mit der der vom durch *port* und *index* gegebenen Register gelesene Wert verodert wird.

10.23.6. Watchdog-Timer und Spannungsüberwachung

Zur Überwachung eines ordnungsgemäßen Programmablaufs sind auf jedem X-MAX-E verschieden Überwachungsmöglichkeiten vorhanden:

Einrichtung	**spricht an, wenn**	**bewirkt**
Watchdog	Watchdog-Timer nicht rechtzeitig nachgetriggert wird	NMI
Spannungs-überwachung A		NMI
Spannungs-überwachung B		Hardware-Reset, Uhr auf Batterie-Pufferung

Watchdog-Timer

Zur Überwachung der laufenden Echtzeit-Software (Tasks) stellt der X-MAX-E einen Watchdog-Timer zur Verfügung. Dieser muss durch eine Task in regelmäßigen Abständen neu gesetzt werden (re-triggern). Um ein gewünschtes Zeitschema einzuhalten, sollte die Re-Triggerung an der "zeitkritischsten" Stelle aller installierten Tasks erfolgen (nur an einer Stelle!).

In Anwendungen, in denen die Interrupt-Belastung des X-MAX-E sehr hoch ist, kann es z.B. dazu kommen, dass NI-Tasks nur sehr selten vom Betriebssystem aufgerufen werden. Soll überwacht werden, dass eine bestimmte NI-Task trotzdem rechtzeitig an die Reihe kommt, muss die Re-Triggerung des Watchdog-Timers in der Hauptprozedur dieser NI-Task stattfinden.

Kommt das Betriebssystem nicht dazu, die Task, die die Re-Triggerung übernimmt, rechtzeitig aufzurufen, weil ein Programm abgestürzt ist, oder andere Tasks zu viel Zeit beanspruchen, bleibt die Re-Triggerung aus. Dadurch läuft der Watchdog-Timer ab. Der X-MAX-E reagiert darauf mit einem NMI-Interrupt. Wenn für diesen Interrupt eine Task installiert ist und der NMI freigegeben ist, ruft das OsX-Betriebssystem die Hauptprozedur dieser Task auf. Dort besteht die Möglichkeit, einen sicheren Betriebszustand herzustellen, Daten zu retten oder den Host-PC durch einen Service-Request zu informieren.

Die Watchdog-Timeout-Zeit, innerhalb der nachgetriggert werden muss, beträgt ca. 1,6 s.

Um die Funktionalität des Watchdogs nutzen zu können, sind folgende Schritte notwendig:

1. Der Watchdog wird standardmäßig vom Betriebssystem in einer NI-Task nachgetriggert. Um dies abzuschalten, ist der Betriebssystem-Parameter 214 (Byte) = 0 zu setzen (Auto-Retrigger = off).
2. Anstelle des Betriebssystems muss jetzt eine Anwender-Task die Nachtriggerung des Watchdog-Timers übernehmen. Das kann mit Hilfe der Bibliotheksfunktion **max_trigger_watchdog** geschehen.
4. Eine II-Task sollte unter dem NMI-Interrupt installiert werden, um bei Ablauf des Watchdog-Timers reagieren zu können. Zu beachten ist, dass der NMI demaskiert werden muss (**max_unmask_int(NMI)**).

Spannungsüberwachung

Unterschreitet die Versorgungsspannung des Moduls einen kritischen Wert, dann wird mit einem NMI reagiert. In einer unter dem NMI installierten Task kann dann versucht werden, wichtige Daten zu retten und einen sicheren Betriebszustand herzustellen.

Sinkt die Versorgungsspannung noch weiter ab, wird ein Hardware-Reset durchgeführt.

Auslesen der NMI-Interruptquelle

Werden mehrere NMI-Quellen freigegeben, kann die Ursache eines NMI durch Aufruf einer Bibliotheksfunktion (**max_get_nmi_reason**) herausgefunden werden. Dieser liefert einen 16-Bit Wert zurück. Darin ist:

Bit 9 des gelesenen Statuswortes = 0, wenn die Spannungsüberwachung den NMI ausgelöst hat.

Bit 10 = 1, wenn der Watchdog-Timer einen NMI verursacht hat.

10.23.7. Modul-Device-Treiber

10.23.7.1. Installation

Der Modul-Device-Treiber für OsX hat die Programmnummer 8051h und den Dateinamen mxeth10.exe. Der Modul-Device-Treiber für Windows hat den Namen mxeth10.sys.

Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x800C, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=800C

10.23.7.2. Kanaleigenschaftsstruktur CPS_XMAXE

Die CPS für das Modul hat den Namen CPS_XMAXE

10.23.7.3. TCP/IP-Sockets

Ein Zugriff auf ein TCP/IP-Socket kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_TCP_SOCKET*	TCP/IP-Socket
.usType	*TCP_SERVER*	Server
	TCP_CLIENT	Client
.usFlags	*_CP_EXCLUSIVE*	Dieses Flag muss gesetzt sein! Der Zugriff erfolgt immer exklusiv.
.ulCallbackEvents	*XETH_EVENT_ESTABLISHED*	Benachrichtigung bei Verbindung
	XETH_EVENT_CLOSED	Benachrichtigung beim Schließen

Strukturelement	Werte	Bedeutung
	XETH_EVENT_NEW_DATA	Benachrichtigung bei neuen Daten
.usReadMode	*IO_MODE_RAM*	Die Empfangs-Daten werden aus einem Empfangspuffer des MDDs entnommen. Der MDD trägt die empfangenen Daten selbstständig in diesen Puffer ein.
.usWriteMode	*IO_MODE_RAM*	Die Sende-Daten werden in einen internen Sendepuffer des MDDs geschrieben. Der Puffer wird automatisch vom MDD geleert.
	IO_MODE_DIRECT	Die Daten werden unter Verwendung des PSH-Flags versendet.
.ulServerIp		IP-Adresse des Servers bei Client
.usServerPort	*0 – 65535*	Eigener Port bei Server, Port des Servers bei Client

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Ist bei einem Lesevorgang nicht die gewünschte Anzahl Zeichen vorhanden, werden die im Puffer vorhandenen Zeichen zurückgegeben. Ist bei einem Schreibzugriff im Puffer nicht genügend Platz vorhanden, werden die Daten nicht übernommen.
Wenn die Verbindung noch nicht hergestellt wurde, so liefert der Schreibdienst den Fehler ERR_NOT_CONNECTED zurück.
Wurde die Verbindung geschlossen, so wird der Fehler ERR_CLOSED zurückgegeben.

Sonderdienste

- **max_channel_control,** Steuerbefehl CMD_START: geschlossenes Socket wieder öffnen. Ein Server geht in den Listen-Modus über, ein Client versucht ein Connect mit der Server-Seite. Dem Dienst werden keine Daten übergeben.
- **max_channel_control,** Steuerbefehl CMD_STOP: offenes Socket schliessen. Ein geschlossenes Socket nimmt keine Daten vom Netzwerk entgegen und kann keine Daten versenden. Dem Dienst werden keine Daten übergeben.
- **max_channel_control,** Steuerbefehl CMD_RESET: Empfangs- und Sendepuffer eines Sockets leeren. Dem Dienst werden keine Daten übergeben.

- **max_channel_info,** Infotyp INFO_DEVICE: den aktuellen Status des Sockets abfragen. Es wird eine ULONG mit folgender Bedeutung geliefert:

Wert	Bedeutung
XETH_LISTEN	Server wartet auf Verbindung durch Client
XETH_CONNECTING	Verbindungsaufbau
XETH_ESTABLISHED	Verbindung ist aktiv
XETH_CLOSING	Verbindung wird beendet
XETH_CLOSED	Verbindung geschlossen (inaktiv)

Hinweise

Beim Öffnen des Kanals geht ein Server in den Listen-Modus, ein Client versucht eine Verbindung zum Server aufzubauen.
Wird die Schnittstelle für eine Host-Ankopplung verwendet, so stehen die Ports 1024 bis 1040 nicht zur Verfügung.

10.23.7.4. IP-Einstellungen

Ein Zugriff auf die IP-Einstellungen kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Control-Kanal
.usIndex	*XETH_IP_SETTING*	IP-Einstellungen
.usReadMode	*IO_MODE_DIRECT*	Die aktuell verwendeten IP-Einstellungen werden zurückgegeben.
.usWriteMode	*IO_MODE_DIRECT*	Die aktuell verwendeten IP-Einstellungen werden gesetzt und im EEPROM gespeichert.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Die IP-Einstellungen der ETHERNET-Schnittstelle werden im EEPROM des Moduls gespeichert. Beim Laden des MDDs werden die Einstellungen aus dem EEPROM gelesen, so daß ein Setzen der Einstellungen mit Hilfe dieses Kanals nur bei einer Änderung der Einstellungen notwendig ist.

Zum Lesen und Schreiben der IP-Einstellungen steht die folgende Struktur XETH_IP_SETTINGS zur Übergabe zur Verfügung:

Strukturelement	Werte	Bedeutung
.usVersion	*0*	Version
.ulIp		IP-Adresse
.ulSubnetmask		Subnetzmaske
.ulGateway		Standard-Gateway

Wird der Schreibdienst IO_MODE_DIRECT verwendet, so werden geöffnete Sockets in den Modus *XETH_CLOSED* gesetzt.

10.23.7.5. Adapter-Einstellungen

Ein Zugriff auf die Adapter-Einstellungen kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Control-Kanal
.usIndex	*XETH_ADAPTER_SETTING*	Adapter-Einstellungen
.usReadMode	*IO_MODE_DIRECT*	Die aktuell verwendeten Adapter-Einstellungen werden zurückgegeben.
.usWriteMode	*IO_MODE_DIRECT*	Die aktuell verwendeten Adapter - Einstellungen werden gesetzt und im EEPROM gespeichert.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Die Adapter-Einstellungen werden im EEPROM des Moduls gespeichert. Beim Laden des MDDs werden die Einstellungen aus dem EEPROM gelesen, so dass ein Setzen der Einstellungen mit Hilfe dieses Kanals nur bei einer Änderung notwendig ist.

Die Adapter-Einstellungen werden mit der Struktur XETH_ADAPTER_SETTINGS an den Eingabe- bzw. Ausgabedienst übergeben:

Strukturelement	Werte	Bedeutung
.usVersion	*0*	Version
.usTransMode	XETH_HALFDUPLEX_10 XETH_FULLDUPLEX_10	Übertragungs-Modus

10.23.7.6. MAC-Adresse

Ein Zugriff auf die MAC-Adresse kann mit folgender CPS erfolgen:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Control-Kanal
.usIndex	*XETH_MAC_ADDRESS*	MAC-Adresse
.usReadMode	*IO_MODE_DIRECT*	Die aktuell verwendete MAC-Adresse wird zurückgegeben.
.usWriteMode	*IO_MODE_DIRECT*	Die aktuell verwendete MAC-Adresse wird gesetzt und im EEPROM gespeichert.

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist DATA_UCHAR.

- **max_write_channel_block**
- **max_read_channel_block**

Die MAC-Adresse der ETHERNET-Schnittstelle ist eine vom Hersteller eindeutig vergebene Adresse, die aus 6 Byte besteht. Normalerweise wird diese Einstellung für Anwenderprogramme nicht benötigt. Sie sollte nicht verändert werden. Wird der Schreibdienst benutzt, so werden geöffnete Sockets in den Modus *XETH_CLOSED* gesetzt.

10.23.7.7. Sonstige Dienste

Nach der Installation stellt der MDD bereits ohne Öffnen eines Kanals den ARP-Reply-Dienst und ICMP-Echo-Reply-Dienst zur Verfügung.
Mittels ARP-Reply können andere Netzwerkteilnehmer die MAC-Adresse der Schnittstelle über die IP-Adresse erfragen
Durch ICMP-Echo-Reply kann ein Ping auf die Schnittstelle erfolgen.

10.23.8. Besondere Eigenschaften

CPU:	486-100, 486-33
RAM:	2 MByte bis 8 MByte
ROM:	Flash-EPROM, 4 MByte bis 16 MByte
EEPROM:	4kByte, seriell
Timer:	3 x 16 Bit Breite, Eingangsfrequenz 1,1892 MHz, Timer A und C interruptfähig, Echtzeituhr.
Interrupts:	15 Eingänge, davon 4 an X-Bus, 1 an Mailbox-Register, 2 an externen Eingängen, 2 an Timer, 1 an serielle Schnittstellen, 1 an Uhr
Serielle Schnittstellen:	RS-232 Schnittstelle mit den Modem-Steuersignalen RTS und CTS, 16550-kompatibel
Ethernet-Schnittstelle:	10base-T Ethernet-Anschluss
Echtzeituhr:	Datum (Tag, Monat, Jahr, Wochentag) und Uhrzeit (Std., Min. Sek.), batteriepufferbar, interruptfähig
Multi-Tasking-Betriebssystem:	Voll echtzeitfähig, max. 1024 Tasks, Interrupt-, Timer-initiierte und Nicht-Interrupt-Tasks. Im Flash des X-MAX-E enthalten, wird ins RAM kopiert, belegt 64 KB RAM.
Stromaufnahme:	+3,3 V: 594 mA (typ. bei 100 MHz)
Abmessungen:	29 x 58 x 8 mm
Temperaturverträglichkeit:	0 bis 55°C (optional –40° .. + 85° C)
Luftfeuchtigkeitsverträglichkeit:	5 bis 95% (nicht kondensierend)

Notizen

X-MAX-200
X-MAX-400

Der kleinste PDA der Welt mit Ethernet und USB-OTG

10.24. X-MAX-200 und X-MAX-400, Rev. A

Inhaltsverzeichnis

10.24.1. Beschreibung

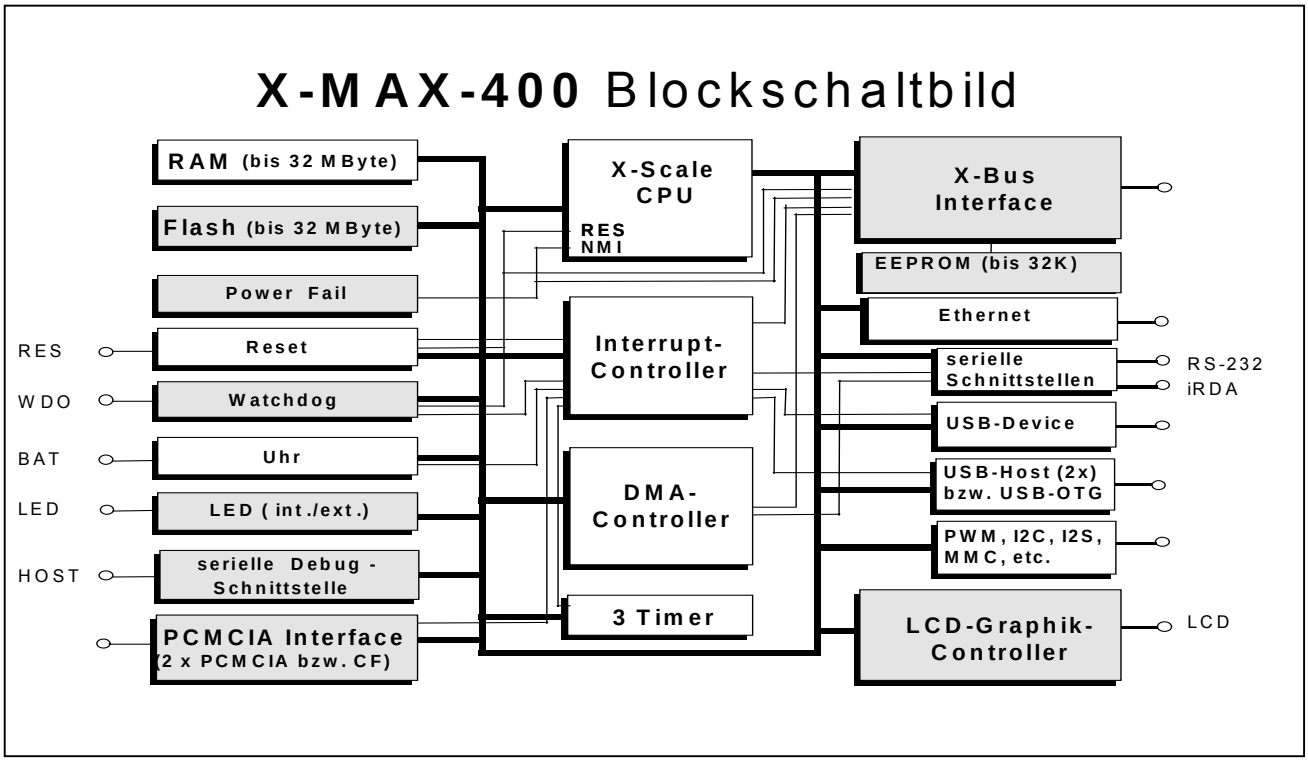
Die CPU-Module X-MAX-200 und X-MAX-400 sind mit einer X-Scale CPU (Intel) bestückt. Die Taktfrequenz der lokalen CPU beträgt je nach Version zwischen 200 MHz und 400 MHz und ist zur Reduzierung der Leistungsaufnahme per Software umschaltbar.

Die Module sind bereits mit umfangreichem Speicher und Peripherie ausgestattet. Außer Flash-ROM (max. 32 MByte), RAM (max. 32 MByte) und 66 KByte Cache verfügen sie über diverse Timer, Interrupt- und DMA-Controller, drei serielle Schnittstellen (eine davon mit RS-232 Pegeln), eine parallele Druckerschnittstelle, eine Uhr, Watch-Dog und eine mit einer externen Batterie pufferbaren Uhr. Das X-MAX-400 ist außerdem noch mit einer 10/100 MBit Ethernet Schnittstelle (10BaseT und 100BaseTx) ausgestattet. Zusätzlich bieten beide Versionen eine USB-Device Schnittstelle und das X-MAX-400 außerdem 2 USB-Host-Schnittstellen, von denen eine auch als OTG-Schnittstelle konfiguriert werden kann.

Im Flash-ROM befindet sich standardmäßig das SORCUS-eigene Echtzeit Multi-Tasking Betriebssystem OsX für bis zu 1024 Tasks. Alternativ ist auch Windows CE verfügbar.

Das CPU-Modul ist auf allen SORCUS Trägerkarten einsetzbar. Es kann ohne Änderung auch mehrfach auf dasselbe Board gesteckt werden und bildet dann ein Multiprozessorsystem, z.B. um zeitkritische Echtzeitaufgaben zu bewältigen. Das CPU-Modul läßt sich auch stand-alone einsetzen, z.B. auf einer kundenspezifischen Trägerkarte. Die Anwendungsprogramme befinden sich dann im Flash-ROM des CPU-Moduls. Die Kommunikation mit einem Hostrechner kann, wenn gewünscht, z.B. über Ethernet (inkl. TCP/IP), USB, seriell oder über ein Modem-Modul erfolgen.

10.24.2. Blockschaltbild



10.24.3. Stecker A

Nachfolgend werden die Standardfunktionen des A-Steckers beschrieben. Erweiterte Pin-Funktionen sowie eine Beschreibung der Stecker B und C finden Sie in der Application Note AN081 auf unserer Homepage **www.sorcus.com**.

Erläuterungen:
1) Wenn in der Spalte „Standardfunktion“ GPxx (mit xx = 0 bis 87) erscheint, dann ist dieser Pin als Ein- oder Ausgang schaltbar und Interrupt-fähig.
2) Alle Pins sind für 3,3V Logik-Pegel ausgelegt und nicht 5V-spannungsfest, außer wenn in der Spalte „Pegel“ etwas anderes vermerkt ist.
3) PU, PD bzw. Rv in der Spalte „Pegel“ bedeuten, dass dieser Pin einen Pull-Up, Pull-Down bzw. Vorwiderstand on-board hat.

Pin	Name	Standardfunktion	Pegel	Eingang/Ausgang
1	TXp	Ethernet: TX+	100BaseTx	Ausgang
2	TXm	Ethernet: TX-	100BaseTx	Ausgang
3	RXp	Ethernet: RX+	100BaseTx	Eingang
4	RXm	Ethernet: RX-	100BaseTx	Eingang
5	GND	B-UART: GND	GND	-
6	RCV	B-UART: RCV	RS-232	Eingang
7	RTS	B-UART: RTS	RS-232	Ausgang
8	TMT	B-UART: TMT	RS-232	Ausgang
9	CTS	B-UART: CTS	RS-232	Eingang
10	HOST	Debug-I/O, GP4	Rv=330 Ω, PU	Ein-/Ausgang
11	/WDO	Watchdog Out	-	Ausgang
12	/MRES	Manuell Reset	Rv=330 Ω, PU	Eingang
13	SCL	I2C-Schnittstelle, Lautsprecher (digital Out, Rv=330 Ω)	PU	Ein-/Ausgang
14	SDA	I2C-Schnittstelle	PU	Ein-/Ausgang
15	DDm	USB-Device: D-, I/O	-	Ein-/Ausgang
16	DDp	USB-Device: D+	-	Ein-/Ausgang
17	iRXD	S-UART: RCV, Infrarot-In, GP46	-	Ein-/Ausgang
18	iTXD	S-UART: TMT, Infrarot-Out, GP47	-	Ein-/Ausgang
19	LED	LED, GP27	Rv=330 Ω	Ein-/Ausgang
20	IRQ0	GP11	PU	Ein-/Ausgang

Pin	Name	Standardfunktion	Pegel	Eingang/Ausgang
21	H1PSW	USB Host 1: Überstromausgang	PU	Ausgang
22	BAT	Batterie	+2,8 .. +3,6V	Eingang
23	H2PSW (/STRB)	USB Host 2: Überstromausgang, GP28	-	Ein-/Ausgang
24	H2Dp (/AFDT)	USB Host 2: D+, GP29	-	Ein-/Ausgang
25	H2Dm (PD0)	USB Host 2: D-, GP32	-	Ein-/Ausgang
26	/H2OC (/ERROR)	USB Host 2: Überstromeingang, GP31	-	Ein-/Ausgang
27	PD1	GP33	PU	Ein-/Ausgang
28	H1BUS (/INIT)	USB Host 1: Vbus, I/O	3,3V-Logik bzw. 5V-Out	Ein-/Ausgang
29	PD2	GP34, F-UART: RCV	-	Ein-/Ausgang
30	H1Dp (/SLCTIN)	USB Host 1: D+, GP24	-	Ein-/Ausgang
31	PD3	GP35, F-UART: CTS	-	Ein-/Ausgang
32	H1Dm (/ACK)	USB Host 1: D-, GP25	-	Ein-/Ausgang
33	PD4	GP36, F-UART: DCD	-	Ein-/Ausgang
34	/H1OC (BUSY)	USB Host 1: Überstromeingang, GP26	-	Ein-/Ausgang
35	PD5	GP37, F-UART: DSR	-	Ein-/Ausgang
36	PE	GP40, F-UART: DTR	-	Ein-/Ausgang
37	PD6	GP38, F-UART: Ri	-	Ein-/Ausgang
38	SLCT	GP41, F-UART: RTS	-	Ein-/Ausgang
39	PD7	GP39, F-UART: TMT	-	Ein-/Ausgang
40	GND	GND	-	-

Ethernet-Schnittstelle (Stecker A, Pin 1..4)

Pin 1 bis 4 stellen einen 10Base-T bzw. 100BaseTx Ethernet Anschluss mit galvan. Trennung zur Verfügung.

Serielle RS-232 Schnittstelle (Stecker A, Pin 5..9)

Pin 5 bis 9 stellen eine serielle Schnittstelle mit RS-232 Pegeln incl. der beiden Modem-Steuerleitungen RTS und CTS zur Verfügung.

Debug Schnittstelle (Stecker A, Pin 10)

Dieser bidirektionale Pin kann z.B. für eine sehr einfache, serielle Debug-Schnittstelle verwendet werden. Er liefert bzw. benötigt Logikpegel.

Watchdog Out (Stecker A, Pin 11)

Dieser Pin ist der Ausgang des Watchdogs. Er ist active Low. Wenn der Watchdog nicht rechtzeitig per Software nachgetriggert wird, geht der Ausgang auf log. 0. Soll das Ablaufen des Watchdog-Timers (siehe unten) einen Hardware-Reset des Moduls auslösen, besteht die Möglichkeit, die Pins 11 (Watchdog Out) und 12 (Manual Reset) von Stecker A zu verbinden.

Manual Reset (Stecker A, Pin 12)

An diesen Eingang kann z.B. ein Taster (einpolig schließend, zwischen Pin 9 und GND) angeschlossen werden, um das CPU-Modul manuell zurückzusetzen.

Der Eingang hat Schmitt-Trigger Charakteristik. Wenn er nicht verwendet werden soll, kann er unbeschaltet bleiben. Er verfügt über einen internen Pull-up-Widerstand.

I2C-Schnittstelle (Stecker A, Pin 13 und 14)

Diese beiden Pins bilden eine I2C-Schnittstelle. Beide Pins verfügen über je einen internen Pull-Up Widerstand. Aus Kompatibilitätsgründen mit X-MAX-1 und X-MAX-E kann Pin 13 auch als universeller Ein-/Ausgang, z.B. auch als Lautsprecherausgang verwendet werden. Die I2C-Schnittstelle steht dann nicht mehr zur Verfügung.

USB-Device Schnittstelle

Pin 15 kann aus Kompatibilitätsgründen mit X-MAX-1 und X-MAX-E alternativ auch als universeller I/O-Pin eingesetzt werden. Die USB-Device-Schnittstelle steht dann nicht mehr zur Verfügung.

Standard-UART (= S-UART) bzw. Infrarot Schnittstelle (Stecker A, Pin 17 und 18)

Diese beiden Pins können als UART verwendet werden (Pin 17 = RCV, Pin 18 = TMT) oder als iRDA Schnittstelle eingesetzt werden. Zusätzlich kann jeder Pin auch als universeller Ein-/Ausgang konfiguriert werden.

LED (Stecker A, Pin 19)

Dieser Logik-Ausgang kann zum Anschluß einer per Software schaltbaren LED dienen. Er kann auch als universeller Ein-/Ausgang konfiguriert werden oder den Zustand der on-board LED´s liefern.

Interrupt-In (Stecker A, Pin 20)

Dieser Pin kann als universeller Ein-/Ausgang z.B. als Interrupt-Eingang verwendet werden. Die Interrupt-Flanke ist programmierbar.

USB Host 1: (Stecker A, Pin 21, 28, 30, 32, 34)

Diese 5 Pins bilden die USB Host Schnittstelle 1. Sie kann auch als USB-OTG konfiguriert werden. Pin 28 kann dann auch die 5 Volt Versorgungsspannung Vbus dafür liefern (abschaltbar).

Batterie-Eingang (Stecker A, Pin 22)

Hier kann eine Batterie angeschlossen werden (max. 3,6 Volt), um die on-board Uhr (RTC) zu versorgen. Wenn das nicht erforderlich ist, sollte der Pin offen bleiben.

Parallel-Port (Stecker A, Pin 23..39)

Standardfunktion der Leitungen /STRB, /AFDT, /ERROR, /INIT, /SLCTIN, /ACK, BUSY, PE, SLCT, PD0..PD7: Druckerport

Diese Pins zusammen können als Druckerport verwendet werden, der als Standard Druckerport (unidirektional) oder PS/2 Druckerport (bidirektional) konfiguriert werden kann. Alle Pins können aber auch als universelle Ein-/Ausgänge und für andere Funktionen konfiguriert werden.

USB Host 2: (Stecker A, Pin 23, 24, 25, 26)

Diese 4 Pins bilden die USB Host Schnittstelle 2. Alle 4 Pins können auch als universelle Ein-/Ausgänge konfiguriert werden oder andere Funktionen übernehmen.

GND (Stecker A, Pin 40)

Bezugspotential für das System bzw. den Druckerport.

10.24.4. Lokale Interrupts

Diese Beschreibung war zum Zeitpunkt der Drucklegung dieses Handbuchs noch nicht verfügbar.

10.24.5. Indirekter Zugriff auf I/O-Ports bzw. CPU-Register

Die auf dem X-MAX-200 und –400 verwendete CPU PXA255 (Intel) bietet über die Standard-PDA-Funktionalität hinaus eine Vielzahl von speziellen Features. Nur ein Teil davon ist bzw. wird zukünftig direkt durch die SORCUS-Bibliotheken bzw. Modul-Device-Treiber unterstützt. Sollen weitere, nicht von dieser Software unterstützte Funktionen verwendet werden, muss auf die direkte Programmierung der Register zurückgegriffen werden. Dazu werden die entsprechenden Handbücher von Intel benötigt. Diese stehen unter www.intel.com zur Verfügung.

In diesem Fall kann SORCUS jedoch keine Gewähr dafür übernehmen, dass bei einem Nachfolge-Modell der Module alle Features der derzeitigen CPU enthalten sein werden. Eine solche Garantie auf Kompatibilität gibt es nur bezüglich der von den SORCUS-Bibliotheken und Modul-Device-Treibern unterstützten Funktionen.

Wenn die Register der CPU direkt angesprochen werden müssen, ist folgendes zu beachten:

Auf andere Adressen als die in den Intel-Handbüchern dokumentierten darf nicht zugegriffen werden. Alle Zugriffe müssen über die zukünftig hier beschriebenen Bibliotheksfunktionen erfolgen.

10.24.6. Modul-Device-Treiber

Diese Beschreibung war zum Zeitpunkt der Drucklegung dieses Handbuchs noch nicht verfügbar.

10.24.7. Besondere Eigenschaften

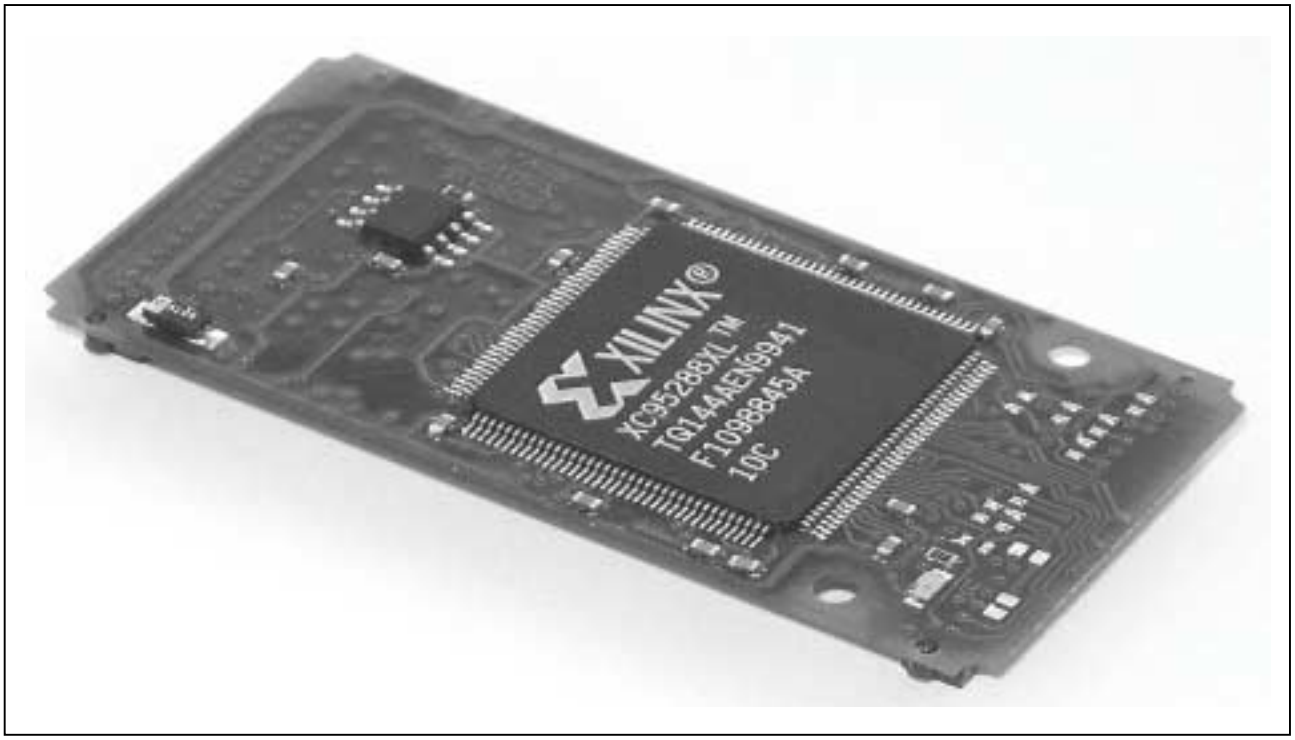
Hier sind nur die für Mess-, Steuer- und Regelungstechnik wichtigen Eigenschaften und Schnittstellen aufgeführt, nicht die für embedded Anwendungen.

CPU:	PXA255 (Intel X-Scale), 100 bis 400 MHz
RAM	8, 16 oder 32 Mbyte
ROM	Flash-EPROM: 4, 8, 16 oder 32 MByte
EEPROM:	4, 8, 16, 32 oder 64 kByte, seriell
Timer	diverse
Interrupts	87 Eingänge, davon 16 an X-Bus
Ethernet:	10Base-T und 100BaseTx Ethernet-Anschluss
USB:	USB-Device in der CPU USB-Host 1 und Host 2, je mit Überstromüberwachung für Vbus. USB-Host 1 auch als USB-OTG konfigurierbar, incl. 5V Erzeugung für Vbus
Serielle Schnittstellen:	B-UART: RS-232 Schnittstelle mit den Modem-Steuersignalen RTS und CTS S-UART: 3,3V-Logik RCV-Eingang und TMT-Ausgang F-UART: 3,3V-Logik mit allen Modem-Steuerleitungen
I2C-Schnittstelle	Multi-Master-fähig, Pull-Up Widerstände on-board
Echtzeituhr	Datum (Tag, Monat, Jahr, Wochentag) und Uhrzei (Std., Min., Sek.), mit ext. Batterie pufferbar interruptfähig
Multi-Tasking-Betriebssystem	Voll echtzeitfähig, max. 1024 Tasks, Interrupt-, Timer-initiierte und Nicht-Interrupt-Tasks. Im Flash des Moduls enthalten, wird ins RAM kopiert, belegt ca. 64 KB RAM.
Stromaufnahme	Nur +3,3 V erforderlich: tbd
Abmessungen	29 x 58 x 8 mm
Temperaturbereich (Betrieb)	0 bis 70°C (optional –40° .. + 85° C)
Luftfeuchtigkeit:	5 bis 95% (nicht kondensierend)

Notizen

X-MIX-26

Universelles Analog und Digital IO-Modul



10

10.25. X-MIX-26: Modul mit analogen und digitalen Ein- und Ausgängen

Inhaltsverzeichnis

10.25.1. Beschreibung

Das Modul X-MIX-26 bietet 4 analoge Eingänge und 4 analoge Ausgänge. Zusätzlich bietet es 19 digitale Kanäle mit TTL-Pegeln, die einzeln als Aus- oder Eingänge geschaltet werden können. Sowohl die Ausgabe von Analogwerten, als auch die Wandlung der analogen Eingänge kann über einen externen digitalen Trigger ausgelöst werden.

Auf dem Modul befindet sich ein Temperatursensor, der für Korrekturzwecke ausgelesen werden kann.

Ausserdem verfügt das Modul noch über einen Watch-Dog, der in 100 ms Schritten von 100 ms bis 1,6 s variiert werden kann. Falls dieser nicht rechtzeitig nachtgeriggert wird, schaltet er alle Ausgänge ab.

10.25.2. Modul-Device-Treiber

Die Beschreibung des Modul-Device-Treibers lag bei Drucklegung dieses Handbuchs noch nicht vor. Sobald diese fertig ist, wird sie auf der SORCUS-Homepage bereitgestellt.

10.25.3. Anschlusspins des Moduls

(bezogen auf den Modul-Stecker A)

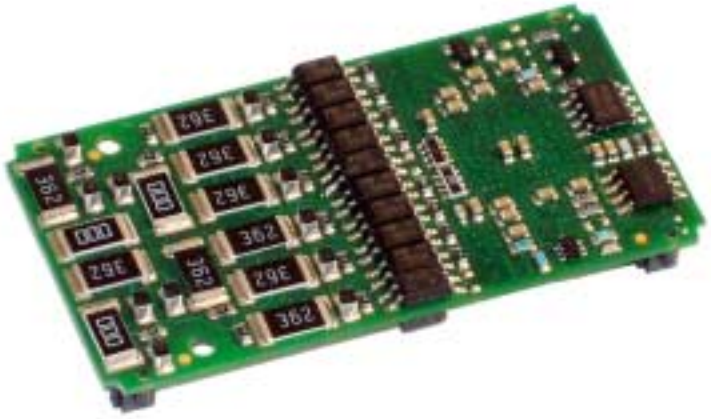
Pin	Funktion	Pin	Funktion
1	AIN-0+	21	AIN-2+
2	AIN-0-	22	AIN-2-
3	AIN-0 GND	23	AIN-2 GND
4	AIN-1+	24	AIN-3+
5	AIN-1-	25	AIN-3-
6	AIN-1 GND	26	AIN-3 GND
7	AOUT-0	27	AOUT-2
8	AGND0	28	AGND2
9	AOUT-1	29	AOUT-3
10	AGND1	30	AGND3
11	DIO-0	31	DIO-10
12	DIO-1	32	DIO-11
13	DIO-2	33	DIO-12
14	DIO-3	34	DIO-13
15	DIO-4	35	DIO-14
16	DIO-5	36	DIO-15
17	DIO-6	37	DIO-16
18	DIO-7	38	DIO-17
19	DIO-8	39	DIO-18
20	DIO-9	40	GND

10.25.4. Besondere Eigenschaften

Parameter	Wert	Einheit
Analoge Eingänge (extern), Anzahl	4	Bit
Auflösung	14	Bit
Wandlungsgeschwindigkeit, max.	2,4	µs
Acquisition Time, max.	0,35	V
Eingangsspannungsbereich	±10	V
Eingangsspannung, max.	±18	V
Erholungszeit aus Stand-By-Mode	1	µs
Analoge Ausgänge, Anzahl	4	-
Auflösung	14	Bit
Ausgangsspannungsbereich	±10	V
Digitale Ein-/Ausgänge, Anzahl	19	-
einzeln als Ein-/Ausgänge konfigurierbar		
Eingangsspannung (kompatibel mit 5V TTL und 5V, 3,3V und 2,5V CMOS)		
log. 0	< 0,8	V
log. 1	> 2,0	V
Input/Output Leakage Current	±10	µA
Ausgangsspannung (kompatibel mit 5V TTL und 3,3V CMOS)		
log. 0, max. (IOL = 8mA)	0,4	V
log. 1, min. (IOH = -4mA)	2,4	V
Ausgangstrom, max. log. 0	-12	mA
min. log. 1	+12	mA
Überspannungsfestigkeit der Eingänge	-0,5 .. +5,5	V
für Impulse < 10ns und < 200mA	–2,0 .. +7,0	V
Temperatur-Bereich, Betrieb	-40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	tbd	g
Stromaufnahme (Ausgänge offen), 3,3V	tbd	mA
+12V	tbd	mA
-12V	tbd	mA

X-OPT-io

Digitales I/O-Modul mit
20 Opto-entkoppelten Ein- bzw. Ausgängen



10.26. X-OPT-io

Inhaltsverzeichnis

10.26.1. Beschreibung

Das Modul X-OPT-io bietet 20 einzeln galvanisch per Optokoppler getrennte Kanäle, wobei jeder Kanal je nach Bestückung Ein- oder Ausgang sein kann. In der Modulbezeichnung X-OPT-io steht das „i“ für die Anzahl der Eingänge, das „o“ für die Anzahl der Ausgänge. Folgende Bestückungsversionen sind standard-mässig lieferbar:

Typ	Subtyp	Modul mit Logikeingängen/Ausgängen	Modul mit Prozesseingängen /Ausgängen	Anzahl Eingänge	Anzahl Ausgänge
24	0/1	X-OPT-200/L	X-OPT-200/P	20	0
24	3/2	X-OPT-200/ T	X-OPT-200/H	20	0
25	0/1	X-OPT-164/L	X-OPT-164/P	16	4
25	3/2	X-OPT-164/T	X-OPT-164/H	16	4
26	0/1	X-OPT-128/L	X-OPT-128/P	12	8
26	3/2	X-OPT-128/T	X-OPT-128/H	12	8
27	0/1	X-OPT-812/L	X-OPT-812/P	8	12
27	3/2	X-OPT-812/T	X-OPT-812/H	8	12
28	0/1	X-OPT-416/L	X-OPT-416/P	4	16
28	3/2	X-OPT-416/T	X-OPT-416/H	4	16
29	0	X-OPT-020		0	20
29	1	X-OPT-020/X		0	20

Eingangspegel

Die High-Speed Eingänge sind Interrupt-fähig und können als allgemeine Eingänge per Software abgefragt werden. Jedes Modul ist in 4 Bestückungsvarianten lieferbar: für TTL-Logikpegel, 5V-Logikpegel und für 24V-Prozesspegel und 48V-Prozesspegel.

Auf Wunsch können auch gemischte Bestückungen und andere Schwellen eingestellt werden.

Modul-Typ	Pegel	Schwelle log. 0	Schwelle log. 1
X-OPT-io/T	TTL-Logik	< 0,8 Volt	> 2,4 Volt
X-OPT-io/L	5V-Logik	< 2,4 Volt	> 4 Volt
X-OPT-io/P	Prozess	< 5 Volt	> 13 Volt
X-OPT-io/H	Prozess, hohe Spannung	< 5 Volt	> 13 Volt

Ausgänge

Module mit der Bezeichnung /x kennzeichnen Module mit Ausgängen für 40 V bei 80 mA, Module ohne diese Bezeichnung haben Ausgänge für 80 V bei 10 mA.

10.26.2. Modul-Device-Treiber

10.26.2.1. Installation

Der Modul-Device-Treiber für das OsX hat (je nach Modultyp) den Dateinamen mxopt200.exe, mxopt164.exe, mxopt128.exe, mxopt812.exe, mxopt416.exe oder mxopt020.exe. Der Modul-Device-Treiber für Windows hat (je nach Modultyp) den Namen mxopt200.sys, mxopt164.sys, mxopt128.sys, mxopt812.sys, mxopt416.sys oder mxopt020.sys.

Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0), die Programm-Nr. ist abhängig vom Modul-Typ:

Error = max_load_mdd (hModul, 1, 0, 0, *siehe Tabelle*, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0) , die Programm-Nr. ist abhängig vom Modul-Typ:

MAXLOADMDD slot=1 layer=0 progno= *siehe Tabelle*

Modul	progno
X-OPT-200	0x8018
X-OPT-164	0x8019
X-OPT-128	0x801A
X-OPT-812	0x801B
X-OPT-416	0x801C
X-OPT-020	0x801D

10.26.2.2. Kanaleigenschaftsstruktur CPS_XOPTIO

Die CPS für das Modul hat den Namen CPS_XOPTIO bzw. CPS_XOPTIO_A.

CPS_XOPTIO_A wurde gegenüber der CPS_XOPTIO um die Parameter ulCallbackEvents und ulTimeout erweitert.

10.26.2.3. LED

Das Modul bietet eine LED, auf die mit folgender CPS zugegriffen werden kann.

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_LED*	Kanal zu einer LED
.usIndexFirst	*0*	Nummer der LED
.usIndexLast	*0*	Nummer der LED
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0* *_CP_EXCLUSIVE*	Keine Bedeutung Der Zugriff erfolgt exklusiv

Eingabe- und Ausgabedienst

Um die LED ein- bzw. auszuschalten muss eine 1 bzw. 0 in den Kanal geschrieben werden. Der Datentyp ist DATA_UCHAR.

- **max_write_channel_uchar**
- **max_read_channel_uchar**

10.26.2.4. Digitale Eingänge

Das Modul stellt, je nach Bestückungsvariante, 0, 4, 8, 12, 16 oder 20 digitale Eingänge zur Verfügung. Um auf diese zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DIN*	Kanal auf einen digitalen Eingang
.usIndexFirst	siehe Tabelle	Nummer des ersten Eingangs (siehe Tabelle)
.usIndexLast	siehe Tabelle	Nummer des letzten Eingangs (siehe Tabelle)
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*0*	Reservierter Parameter
.usFlags	*0*	Keine Bedeutung
	_CP_EXCLUSIVE	Der Zugriff erfolgt immer exklusiv. Muss bei Callback-Funktionalität gesetzt sein!
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.ulCallbackEvents		Dieser Parameter regelt, wann die Anwender-Callback-Funktion aufgerufen werden soll:
	XOPTIO_EVENT_POS_EDGE	Bei einer positiven Flanke am DIN wird die Callback-Funktion aufgerufen
	XOPTIO_EVENT_NEG_EDGE	Bei einer negativen Flanke am DIN wird die Callback-Funktion aufgerufen

Eingabedienst

Der Datentyp des Kanals ist abhängig von den Werten in *.usIndexFirst* und *.usIndexLast* (siehe Tabelle). Der Zugriff auf das Device erfolgt je nach Datentyp mit:

Datentyp DATA_UCHAR

- **max_read_channel_uchar**

oder Datentyp DATA_ULONG

- **max_read_channel_ulong**

Callback-Funktion

Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen, wenn eine positive bzw. negative Flanke an einem DIN auftritt (Interrupt). Die Callback-Funktion bekommt einen ULONG-Wert übergeben. Die Bits kennzeichnen, welcher Eingang einen Interrupt ausgelöst hat. Die Daten sind dabei rechtsbündig angeordnet (wurde z.B. ein Kanal zu den DINs 4 ... 8 geöffnet, so zeigt das unterste Bit an, dass DIN-4 einen Interrupt verursacht hat). Bei Modulen der Revision A können maximal 3 verschiedene DINs Interrupts auslösen. Bei Modulen ab Revision B kann jeder DIN einen Interrupt auslösen.

10.26.2.5. Digitale Ausgänge

Das Modul stellt, je nach Bestückungsvariante, 0, 4, 8, 12, 16 oder 20 digitale Ausgänge zur Verfügung. Um auf diese zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	**Werte**	**Bedeutung**
.usDevice	*DEVICE_DOUT*	Kanal zu einem digitalen Ausgang
.usIndexFirst	siehe Tabelle	Nummer des ersten Ausgangs (siehe Tabelle)
.usIndexLast	siehe Tabelle	Nummer des letzten Ausgangs (siehe Tabelle)
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt immer exklusiv
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.ulCallbackEvents		Dieser Parameter regelt, wann die Anwender-Callback-Funktion aufgerufen werden soll:
	XOPTIO_EVENT_POS_EDGE	Bei einer positiven Flanke am DOUT wird die Callback-Funktion aufgerufen
	XOPTIO_EVENT_NEG_EDGE	Bei einer negativen Flanke am DOUT wird die Callback-Funktion aufgerufen

Eingabe- und Ausgabedienst

Der Datentyp des Kanals ist abhängig von den Werten in *.usIndexFirst* und *.usIndexLast* (siehe Tabelle). Der Zugriff auf das Device erfolgt je nach Datentyp:

Datentyp DATA_UCHAR:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

oder Datentyp DATA_ULONG:

- **max_write_channel_ulong**
- **max_read_channel_ulong**

Callback-Funktion

Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen, wenn eine positive bzw. negative Flanke an einem DOUT auftritt (Interrupt). Die Callback-Funktion bekommt einen ULONG-Wert übergeben. Die Bits kennzeichnen, welcher Ausgang einen Interrupt ausgelöst hat. Die Daten sind dabei rechtsbündig angeordnet (wurde z.B. ein Kanal zu den DOUTs 4 ... 8 geöffnet so zeigt das unterste Bit an, dass DOUT-4 einen Interrupt verursacht hat). Bei Modulen der Revision A können maximal 3 verschiedene DOUTs Interrupts auslösen. Bei Modulen ab Revision B kann jeder DOUT einen Interrupt auslösen.

10.26.2.6. Watchdog

Um auf den Watchdog zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_WATCHDOG*	Kanal auf einen Watchdog
.usIndexFirst	*0*	Nummer des Watchdog
.usIndexLast	*0*	Nummer des Watchdog
.ulTimeout	Rev. A: *200,1600* ab. Rev. B: 200, 400, 600, ... 1600	Timeout für Watchog in ms
.usReadMode	*0*	Reservierter Parameter
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*CP_EXCLUSIVE*	Der Zugriff erfolgt exklusiv

Strukturelement	Werte	Bedeutung
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.ulCallbackEvent	*XOPTIO_EVENT_WATCHDOG*	

Ausgabedienst

Der Datentyp ist DATA_VOID. Der Zugriff erfolgt mit:

- **max_trigger_channel**

Anmerkung

Damit der Watchdog nicht aktiv wird, muss er zumindest einmal während der in der CPS angegebenen Timeout-Zeit nachgetriggert werden. Bleibt das aus, so dass der Watchdog-Timer abläuft, werden alle Ausgänge hochohmig geschaltet. Zusätzlich kann das Anwenderprogramm benachrichtigt werden. Dazu ist bei **max_open_channel** der Parameter *pCbFunc* mit der Adresse der im Benachrichtigungsfall aufzurufenden Funktion auszufüllen. Sie bekommt bei ihrem Aufruf keine Parameter übergeben. Das Deaktivieren des Watchdog erfolgt beim Schließen des Kanals.

Sonderdienst

- **max_channel_info,** Infotyp INFO_DEVICE: Zustand des Watchdogs ermitteln. . Die Funktion liefert den Status des Watchdog als ULONG-Wert zurück, wobei 0 bedeutet, dass der Watchdog nicht abgelaufen ist, 1 bedeutet, dass der Watchdog abgelaufen ist.
- **max_channel_control,** Steuerbefehl CMD_START: mit diesem Befehl kann der Watchdog anschließend wieder gestartet werden. Dem Dienst werden keine Daten übergeben.

10.26.2.7. Filter für die digitalen Eingänge

Erst ab Modul Revision B verfügbar!

Mit diesem Kanal kann ein Eingangsfilter für die digitalen Eingänge des Moduls zugeschaltet werden. Eine Änderung an den Eingängen wird erst dann akzeptiert, wenn diese für eine gewisse Zeit anliegt.

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Kanal auf einen Kontrollkanal
.usIndexFirst	*XOPTIO_INPUT_FILTER*	Eingangsfilter
.usIndexLast	*XOPTIO_INPUT_FILTER*	Eingangsfilter
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*0*	Reservierter Parameter

Eingabe- und Ausgabedienst

Der Datentyp ist DATA_ULONG. Der Zugriff erfolgt mit:

- **max_write_channel_ulong**
- **max_read_channel_ulong**

Als Daten werden die X-Bus-Takte angegeben, die das Signal anliegen muss um akzeptiert zu werden. Als Werte sind 0 (Filter deaktiviert), 1 (das Signal muss mind. einen Takt anliegen) oder 2 (das Signal muss mind. 2 Takte anliegen) erlaubt.

10.26.2.8. Device-Index und Datentypen

Tabelle: Mögliche Werte für *.usIndexFirst und .usIndexLast*

Modul-Typ	**DEVICE_DIN** ***.usIndexFirst/.usIndexLast***	**DEVICE_DOUT** ***.usIndexFirst/usIndexLast***	**Datentyp**
X-OPT-200	0 ... 19	-	DATA_ULONG
X-OPT-164	0 ... 7 10 ... 17	8 ... 9 18 ... 19	DATA_UCHAR DATA_UCHAR
X-OPT-128	0 ... 5 10 ... 15	6 ... 9 16 ... 19	DATA_UCHAR DATA_UCHAR
X-OPT-812	0 ... 3 10 ... 13	4 ... 9 14 ... 19	DATA_UCHAR DATA_UCHAR
X-OPT-416	0 ... 1 10 ... 11	2 ... 9 12 ... 19	DATA_UCHAR DATA_UCHAR
X-OPT-020	-	0 ... 19	DATA_ULONG

10.26.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Kanal	Anode bzw. Kollektor	Kathode bzw. Emitter	Modul X-OPT-200	Modul X-OPT-164	Modul X-OPT-128	Modul X-OPT-812	Modul X-OPT-416	Modul X-OPT-020
CH-0	1	2	Input	Input	Input	Input	Input	Output
CH-1	3	4	Input	Input	Input	Input	Input	Output
CH-2	5	6	Input	Input	Input	Input	Output	Output
CH-3	7	8	Input	Input	Input	Input	Output	Output
CH-4	9	10	Input	Input	Input	Output	Output	Output
CH-5	11	12	Input	Input	Input	Output	Output	Output
CH-6	13	14	Input	Input	Output	Output	Output	Output
CH-7	15	16	Input	Input	Output	Output	Output	Output
CH-8	17	18	Input	Output	Output	Output	Output	Output
CH-9	19	20	Input	Output	Output	Output	Output	Output
CH-10	21	22	Input	Input	Input	Input	Input	Output
CH-11	23	24	Input	Input	Input	Input	Input	Output
CH-12	25	26	Input	Input	Input	Input	Output	Output
CH-13	27	28	Input	Input	Input	Input	Output	Output
CH-14	29	30	Input	Input	Input	Output	Output	Output
CH-15	31	32	Input	Input	Input	Output	Output	Output
CH-16	33	34	Input	Input	Output	Output	Output	Output
CH-17	35	36	Input	Input	Output	Output	Output	Output
CH-18	37	38	Input	Output	Output	Output	Output	Output
CH-19	39	40	Input	Output	Output	Output	Output	Output

10.26.4. Besondere Eigenschaften

Parameter	Wert	Einheit
Externe Ausgänge (Versionen /L, /P und /H)		-
Anzahl max.	20	
Ausgangsspannung, max.	70	V
Ausgangsspannung, max. bei log. 0 (Iout = 10 mA)	<0,4	V
Ausgangsstrom, nominell je Kanal	10	mA
Ausgangsstrom, max. je Kanal	30	mA
Externe Ausgänge (Versionen /X)		
Anzahl max.	20	-
Ausgangsspannung, max.	40	V
Ausgangsspannung, max. bei log. 0 (Iout = 60 mA)	<1,0	V
Ausgangsstrom, nominell je Kanal	60	mA
Ausgangsstrom, max. je Kanal	90	mA
Externe Eingänge (Versionen /T)		
Anzahl max.	20	-
Eingangsspannung, max.	±12	V
Schwelle max., Erkennung einer log. 0	0,8	V
Schwelle min., Erkennung einer log. 1	2,4	V
Schwelle Eingangsstrom, typ.	2	mA
Eingensimpedanz	500	Ohm
Externe Eingänge (Versionen /L)		
Anzahl max.	20	-
Eingangsspannung, max.	±18	V
Schwelle max., Erkennung einer log. 0	2,2	V
Schwelle min., Erkennung einer log. 1	4	V
Schwelle Eingangsstrom, typ.	2	mA
Eingensimpedanz	500	Ohm
Externe Eingänge (Versionen /P)		
Anzahl max.	20	-
Eingangsspannung, max.	±30	V
Schwelle max., Erkennung einer log. 0	5	V
Schwelle min., Erkennung einer log. 1	13	V
Schwelle Eingangsstrom, typ.	2	mA
Eingansimpedanz	3,9	kOhm
Externe Eingänge (Versionen /H)		
Anzahl max.	20	-
Eingangsspannung, max.	±72	V

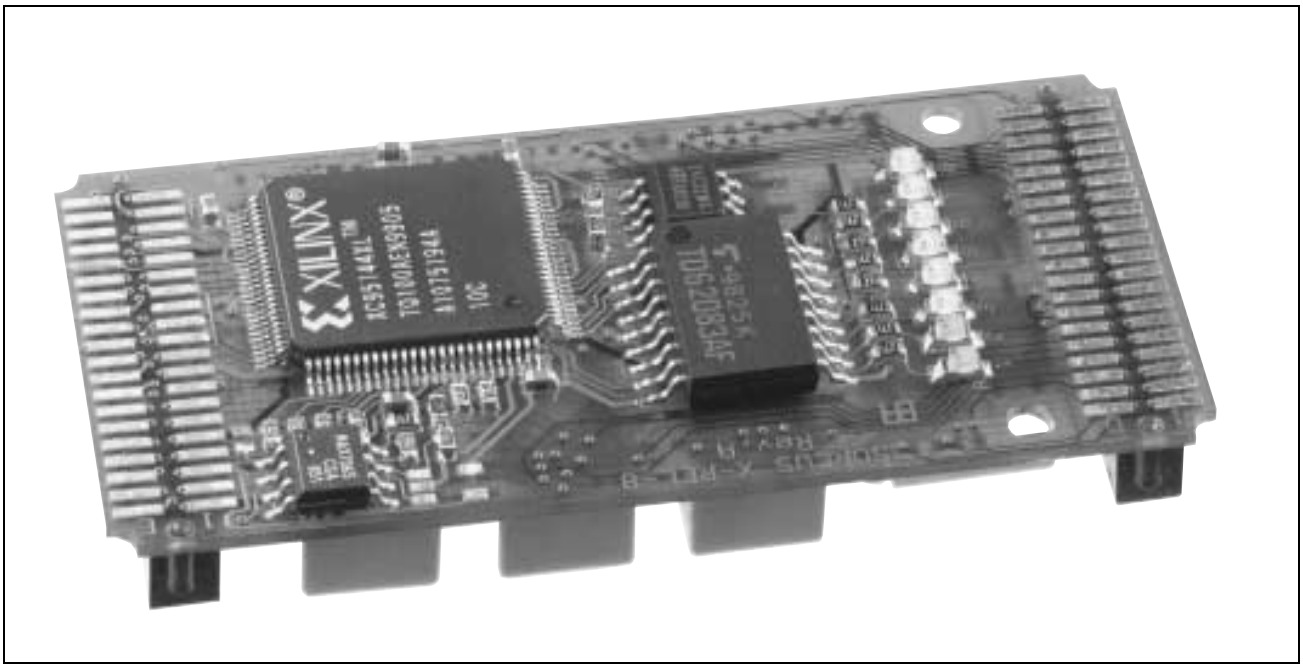
Parameter	**Wert**	**Einheit**
Schwelle max., Erkennung einer log. 0	5	V
Schwelle min., Erkennung einer log. 1	13	V
Schwelle Eingangsstrom, typ.	2,4	mA
Eingansimpedanz	Konstant-strom	-
Temperatur-Bereich, Betrieb	0 .. +70	°C
Abmessungen	29x58x8	mm
Gewicht, min./max.	8,6/9,35	g
Stromaufnahme		
3,3V	325+10*x	mA
(die X-Bus Spannungen ±12V werden nicht benötigt)		

x = Anzahl aktiver DOUTs

Notizen

X-REL-8

8 Relais-Ausgänge
(je Kanal 1 x Um und 1 x Ein bzw. 1 x Aus)



10.27. X-REL-8

Inhaltsverzeichnis

10.27.1. Beschreibung

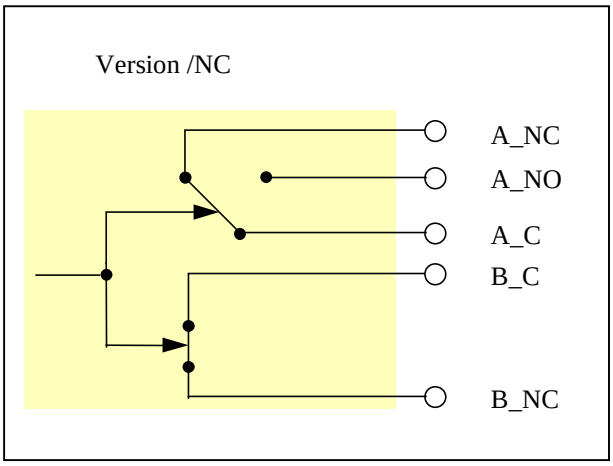
Das Modul X-REL-8 stellt 8 Relaisausgänge zur Verfügung. Jedes Relais verfügt über einen Umschalter und einen Ein- bzw. Ausschalter, abhängig von der Modul-Version. Alle 16 Schalter sind untereinander galvanisch getrennt.

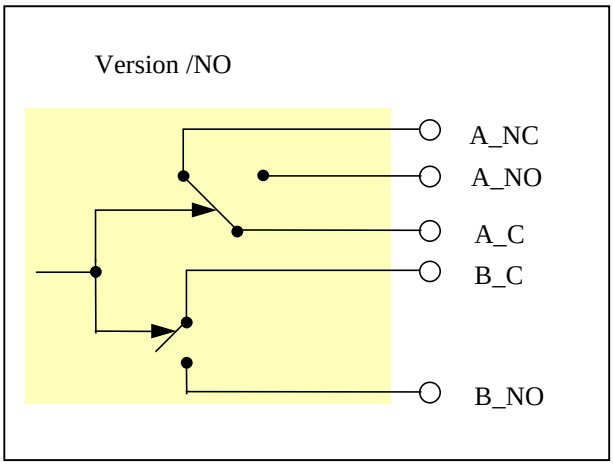
Zusätzlich verfügt das Modul über einen Watchdog, der, wenn er nicht rechtzeitig nachgetriggert wird, alle 8 Relais in den Ruhezustand schaltet. Optional kann dabei auch ein Interrupt ausgelöst werden.

Das Modul ist in folgenden Bestückungsvarianten lieferbar:

Typ	Subtyp	Relaiskonfiguration	Modul-Version
16	0	1 x Um, 1 x Ein	X-REL-8/NO
16	1	1 x Um, 1 x Aus	X-REL-8/NC

10.27.2. Blockschaltbild





10.27.3. Modul-Device-Treiber

10.27.3.1. Installation

Der Modul-Device-Treiber für das OsX hat die Programmnummer 8010h und den Dateinamen mxrel8.exe, für Windows hat er den Namen mxrel8.sys.

Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8010, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8010

10.27.3.2. Kanaleigenschaftsstruktur CPS_XREL8

Die CPS für das Modul hat den Namen CPS_XREL8.

10.27.3.3. Digitale Ausgänge

Um auf die digitalen Ausgänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_DOUT*	Kanal zu einem digitalen Ausgang
.usIndexFirst	*0 ... 7*	Nummer des ersten Ausgangs
.usIndexLast	*0 ... 7*	Nummer des letzten Ausgangs
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Kanal mit Lese- und Schreibfunktion
.usFlags	*_CP_EXCLUSIVE*	Der Zugriff erfolgt exklusiv

Eingabe- und Ausgabedienst

Alle Kanalzugriffe erfolgen mit:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

Die Daten werden rechtsbündig übergeben. Wird das entsprechende Bit =1 gesetzt, ist der Schalter im zugehörigen Relais geschlossen (es fließt Strom durch das Relais). Je nach Modul-Version sind die Relais-Kontakte offen oder geschlossen. Der Datentyp ist DATA_UCHAR.

10.27.3.4. Watchdog

Um auf den Watchdog zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_WATCHDOG*	Kanal auf einen Watchdog
.usIndexFirst	*0*	Nummer des Watchdog
.usIndexLast	*0*	Nummer des Watchdog
.usTimeout	*200* *1600*	Timeout für Watchog: 200 ms oder 1,6 s
.usReadMode	*0*	Reservierter Parameter
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usFlags	*CP_EXCLUSIVE*	Der Zugriff erfolgt exklusiv

Ausgabedienst

Der Datentyp ist DATA_VOID. Der Zugriff erfolgt mit:

- **max_trigger_channel**

Anmerkung

Damit der Watchdog nicht aktiv wird, muss er zumindest einmal während der in der CPS angegebenen Timeout-Zeit nachgetriggert werden. Bleibt das aus, so dass der Watchdog-Timer abläuft, werden alle Relais stromlos geschaltet. Zusätzlich kann das Anwenderprogramm benachrichtigt werden. Dazu ist bei **max_open_channel** der Parameter *pCbFunc* mit der Adresse der im Benachrichtigungsfall aufzurufenden Funktion auszufüllen. Sie bekommt bei ihrem Aufruf keine Parameter übergeben. Das Deaktivieren des Watchdog erfolgt beim Schließen des Kanals.

Sonderdienste

- **max_channel_info,** Infotyp INFO_DEVICE: Zustand des Watchdog lesen. Die Funktion liefert den Status des Watchdog als ULONG-Wert zurück, wobei 0 bedeutet, dass der Watchdog nicht abgelaufen ist, 1 bedeutet, dass der Watchdog abgelaufen ist und die Relais zurückgesetzt wurden.
- **max_channel_control,** Steuerbefehl CMD_START: Watchdog anschließend wieder starten.

10.27.4. Anschlusspins des Moduls

Diese Module besitzen für jeden der 8 Kanäle 2 unabhängige Schalter A und B (siehe Blockschaltbild).
In der folgenden Tabelle sind die Pin-Nummern von Stecker A für alle Kontakte aller 8 Kanäle aufgelistet.

Kanal	**A_ NC**	**A_NO**	**A_C**	**B_NC bzw. B_NO**	**B_C**
0	1	2	3	5	4
1	6	7	8	10	9
2	11	12	13	15	14
3	16	17	18	20	19
4	21	22	23	25	24
5	26	27	28	30	29
6	31	32	33	35	34
7	36	37	38	40	39

10.27.5. Besondere Eigenschaften

Parameter		Wert	Einheit
Relais, Anzahl		8	
je Relais 1 Umschalt und 1 Ein- bzw. Ausschaltkontakt			
Kontaktwiderstand, initial (bei 6V DC und 1A), max.		100	mΩ
Schaltleistung, max.		30	W
Schaltspannung, max.	AC	110	V
	DC	125	V
Schaltstrom, max.		1	A
Lebenserwartung,	elektrisch bei 30V, 1A	10**5	Operationen
	mechanisch	5 * 10**7	Operationen
Einschaltzeit, max. ohne Prellen		4	ms
Ausschaltzeit, max. ohne Prellen		4	ms
Watchdog:			
Time-Out Zeit		200 bzw.	ms
		1600	ms
Temperatur-Bereich, Betrieb		-40 .. +85	°C
Abmessungen		29x58x8	mm
Gewicht, min./max.		12,7/12,85	g
Stromaufnahme, alle Relais offen /geschlossen			
3,3V		175/485	mA
(die X-Bus Spannungen ±12V werden nicht benötigt)			

X-SCC-2

Modul für serielle Kommunikation

10.28. X-SCC-2

Inhaltsverzeichnis

Das Modul X-SCC-2 bietet 2 unabhängige, serielle Kommunikationskanäle. Es verwendet dazu den „Enhanced Serial Communication Controller“ ESCC Z85230 von ZILOG in der 16 MHz Version.

10.28.1. Bestückungsversionen

Das Modul ist in drei Versionen lieferbar:

Bestückungs-Version: X-SCC-2/U

- 2 serielle Schnittstellen für asynchron, Bit-synchron, Byte-synchron, HDLC, SDLC, bisync., etc.
- 8 Byte Empfangs-FIFO und 4 Byte Sende-FIFO je Kanal
- Quarzoszillator 14,732 MHz für Baudratenerzeugung
- Baudratengenerator und PLL je Kanal
- Per Software umschaltbar:
  RS-232, RS-422, RS-485/1, RS-485/2, 20 mA
- RS-xxx an Pin 1..10 bzw. 21..30
- 20 mA an Pin 11..20 bzw. 31..40
- Diverse CLKin- und CLKout-Konfigurationen programmierbar
- Abschlusswiderstände programmierbar (bei RS-4xx)

Bestückungs-Version: X-SCC-2/R

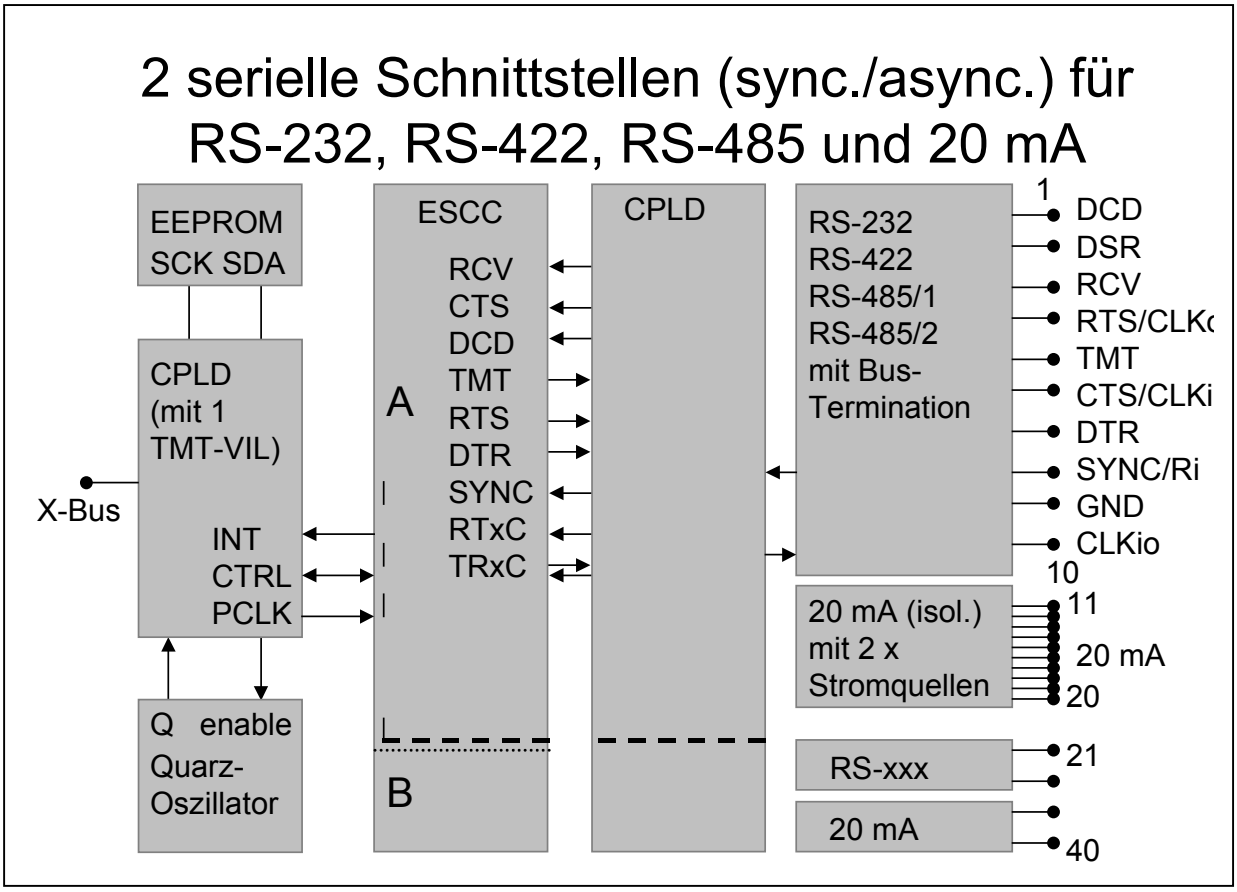
- Nur RS-232 möglich
- Modem-Steuersignale: RTS, CTS, DTR, DSR, Ri, DCD, zusätzlicher Clock-Eingang an CLKio
- Ri und/oder CTS als Clock-Eingänge programmierbar
- RTS als Clock-Ausgang programmierbar
- CLKio-Pin als Clock-Eingang programmierbar

- max. 120 KBaud
- 3,3 V und +/- 12 V erforderlich
- Kein Shut-Down

Bestückungsversion: X-SCC-2i/C

- Wie X-SCC-2/R
- Zusätzlich 20mA Current Loop Sender/Empfänger (galv. getrennt)
- Zusätzlich 2 Konstantstromquellen je Kanal

10.28.2. Blockschaltbild



10.28.3. Software

10.28.3.1. Asynchrone serielle Kommunikation

Es stehen Treiberprogramme für OsX zur Verfügung. Diese laufen auf einem MAX-PC, können aber auch vom Host-PC aus angesprochen werden.

Die Intelligenz des MAX-PC wird zum einen für die Pufferung der Daten und zum anderen für die Abarbeitung beliebiger Protokolle benutzt. Beides läuft komplett unabhängig vom Host-PC. Der Host-PC gibt die Nutzdaten zu beliebiger Zeit mit hoher Geschwindigkeit zum MAX-PC, der sich von da ab nur um das Senden der Daten und um die Erfüllung aller Protokollanforderungen kümmert.

Die Echtzeit-Programme für die serielle Kommunikation lassen sich in zwei Teile zerlegen. Der eine Teil übernimmt das Senden und das Empfangen von Zeichen aus dem bzw. in den Speicher des MAX-PC. Er setzt direkt auf der Hardware auf und stellt alle zeitkritischen Funktionen zur Verfügung. Er ist so programmiert, dass auch

bei hohen Baudraten keine Zeichen verloren gehen. Dieser Teil wird als **Basiskommunikation** bezeichnet. Der zweite Teil ist eher nicht zeitkritisch. Er übernimmt das gesamte **Protokollhandling**, also die Erzeugung und Überprüfung von Steuerzeichen und -signalen, Anforderung von Wiederholungen und alle anderen Aktionen, die zur Erfüllung der Protokollspezifikationen nötig sind. Natürlich kann auch in diesem Teil des Programms die Zeit eine wichtige Rolle spielen, wenn zum Beispiel Reaktionstelegramme innerhalb einer bestimmten Zeitspanne versandt werden müssen.

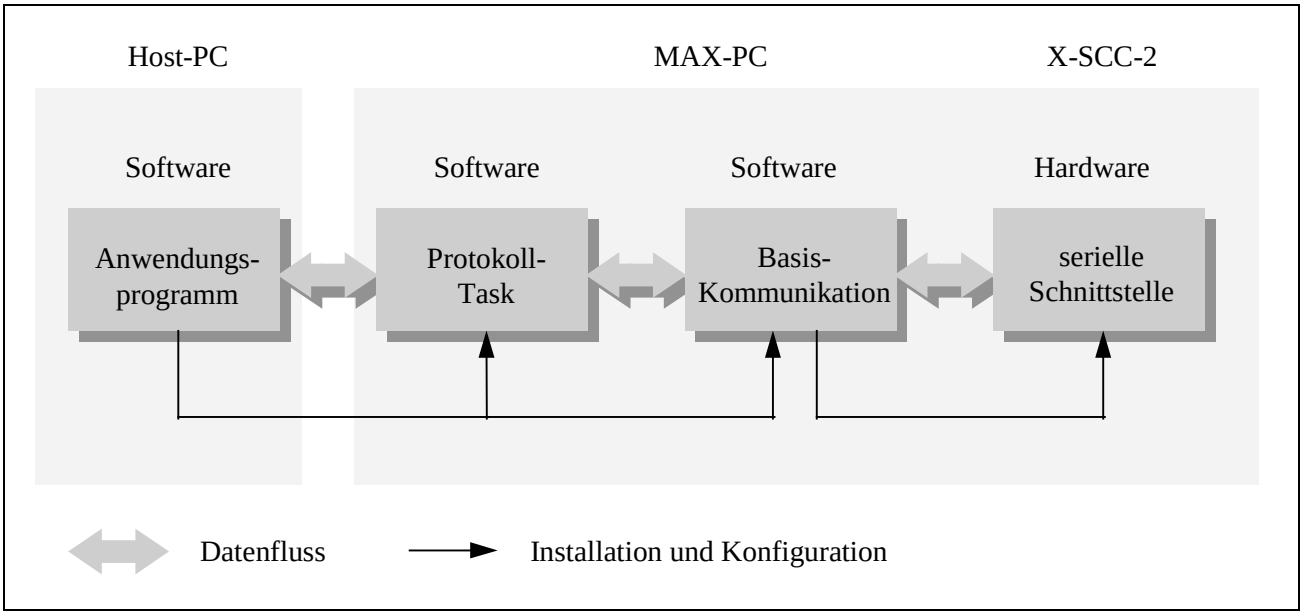


Abb. 1: Grundstruktur der asynchronen seriellen Kommunikation

Der PC tauscht mit dem Teil bzw. mit der Task, die das Protokollhandling übernimmt, die Sende- und Empfangsdaten aus. Wenn ganz ohne Protokoll, oder nur mit einem einfachen Handshake (RTS/CTS, XON/XOFF) kommuniziert wird, entfällt die Protokolltask, und der PC tauscht die Daten direkt mit den Tasks der Basiskommunikation aus.

10.28.3.2. Basiskommunikation

Die Programme der Basiskommunikation lösen, wie bereits erwähnt, den zeitkritischen Teil der Kommunikation und arbeiten direkt mit der Hardware. Die Basiskommunikation für eine serielle Schnittstelle setzt sich aus mehreren Programmen bzw. Tasks zusammen. Die Installation der Programme auf einem MAX-PC geschieht am einfachsten mit SNW32.

10.28.3.3. Die Grundstruktur der Basiskommunikation

Jede der 2 seriellen Schnittstellen auf dem Modul kann aus mehreren Gründen Interrupts auslösen. Für jede Schnittstelle und jeden ihrer Interrupts wird eine eigene Task installiert. Da aber jedes MAX-Modul X-SCC-2 nur einen Interrupt auslösen kann, muss zunächst entschieden werden, von welcher Schnittstelle und aus welchem Grund der Interrupt ausgelöst wurde, um zu bestimmen, welche Task den Interrupt bearbeiten muss. Diese Aufgabe wird von einer Task übernommen, die unter dem jeweiligen Interrupt des X-SCC-2 installiert und als **Interruptmanager** bezeichnet wird.

Die Basiskommunikationsprogramme bestehen aus einem Interruptmanager (Programm X1PA004.LIB) und einem Kommunikationsprogramm (Programm X1PA005.LIB), die für die Schnittstellen des MAX-Moduls X-SCC-2 geeignet sind. Pro X-SCC-2 Modul muss eine Interruptmanager-Task installiert sein und für jeden verwendeten Kanal eine Kommunikations-Task.

Nur mit den Kommunikationstasks werden Daten ausgetauscht, der Interruptmanager wird nie aus der Anwendungsebene heraus angesprochen.

10.28.3.4. Handshake

Für asynchrone Kommunikation bietet die Basiskommunikation zwei einfache Handshakemechanismen (RTS/CTS und XON/XOFF), mit denen verhindert werden kann, dass der Empfangspuffer überläuft. In beiden Fällen meldet die Empfangsstation, wenn der Empfangspuffer einen bestimmten (einstellbaren) 'Füllstand' erreicht hat. Der Sender sendet der Gegenstation dann vorübergehend keine Zeichen mehr, bis gemeldet wird, dass der 'Füllstand' wieder unter eine (einstellbare) Grenze gefallen ist.

Beim **RTS/CTS**-Handshake muss die RTS-Leitung des Empfängers mit der CTS-Leitung des Senders verbunden werden. Wenn ein bestimmter Füllstand (Meldegrenze für 'Puffer voll') erreicht ist, setzt der Empfänger RTS auf Null, der Sender unterbricht dann den laufenden Sendevorgang. Sobald der Puffer wieder leer ist (Meldegrenze für 'Puffer leer'), wird RTS wieder auf eins gesetzt, und der Sender fährt mit dem Senden von Zeichen fort.

Der **XON/XOFF**-Handshake benötigt im Gegensatz zu RTS/CTS keine zusätzlichen Steuerleitungen. Der Empfänger sendet, sobald die obere Meldegrenze erreicht ist, das Steuerzeichen XOFF zum Sender. XOFF ist ein beliebiges Zeichen, das auf Empfänger- und Senderseite gleich sein muss (üblicherweise 13h). Sobald auf der Sendeseite XOFF empfangen wird, wird das Senden weiterer Zeichen unterbunden, bis der Empfänger durch das Senden von XON wieder Empfangsbereitschaft meldet. XON ist ebenfalls ein frei definierbares Zeichen (üblicherweise 11h). Beim Arbeiten

mit XON/XOFF ist zu beachten, dass sowohl XON als auch XOFF reservierte Steuerzeichen sind und in den Nutzdaten nicht vorkommen dürfen. Sie werden immer als Steuerzeichen interpretiert und gelangen nie in den Empfangspuffer.

10.28.3.5. Senden und Empfangen

Die für Senden und Empfangen benutzten Parameter, Prozeduren und Funktionen sind für alle von SORCUS gelieferten Basiskommunikationsprogramme gleich. Je nachdem, ob die Sendedaten vom PC oder von einer übergeordneten Protokolltask kommen, müssen folgende Schritte mit den Bibliotheken durchgeführt werden. Alle in diesem Kapitel beschriebenen Aufrufe beziehen sich auf die Kommunikationstask des jeweiligen Kanals. Die Tasknummern können im Prinzip frei gewählt werden; empfohlen wird jedoch folgende Zuordnung:

Interrupt-kanal	**Tasknummer des Interrupt-Managers**	**Tasknummer der Kommunikationstask**
PIRQ-6 (7ch)	301h	318h + Kanalnummer (0..3)
PIRQ-7 (7dh)	302h	320h + Kanalnummer (0..3)
PIRQ-2 (91h)	303h	328h + Kanalnummer (0..3)
PIRQ-3 (92h)	304h	330h + Kanalnummer (0..3)

Tab. 1: Empfohlene Verteilung von Tasknummern und Interruptnummern

10.28.3.6. Protokollhandling

Die von SORCUS lieferbaren Protokollprogramme unterstützen alle seriellen Schnittstellen des MAX-Moduls X-SCC-2. Sie setzen auf der Basiskommunikation auf. Das Senden und Empfangen von Daten wird bei jedem Protokoll unterschiedlich gehandhabt.

10.28.3.7. Das Kommunikationsprogramm X1PA005.LIB

Das Kommunikationsprogramm X1PA005.LIB bedient **eine** serielle Schnittstelle auf dem MAX-Modul X-SCC-2. Dies betrifft sowohl die Initialisierung der Schnittstelle als auch den laufenden Betrieb. Es ermöglicht den Zugriff auf alle für eine serielle Schnittstelle relevanten Einstellungen und Zustände (Fehlerstatus, Statusleitungen, etc.).

Das Basiskommunikationsprogramm enthält auch die für die beiden Datenfluss-Protokolle XON/XOFF und RTS/CTS nötigen Funktionen, die sich durch Parameter aktivieren lassen.

10.28.3.8. Initialisierung

Zunächst müssen die Kommunikationsparameter im Parameterbereich der für den Kanal zuständigen Kommunikationstask eingestellt werden. Anschließend wird die Task durch Aufruf von Prozedur 2 gestartet, so dass sie zum Senden und Empfangen bereit ist. Die Datenpuffer sind dadurch angelegt. Falls Sie aber Parameter (wie z.B. Baudrate) ändern oder die Kommunikation neu beginnen möchten, muss das Programm reinitialisiert werden. Nach dem Ändern von Parametern muss Prozedur 2 aufgerufen werden, damit die Änderungen wirksam werden. Mit den Prozeduren 5 und 15 können Sende- und Empfangsbereitschaft ein- und ausgeschaltet sowie Sende- und Empfangspuffer gelöscht werden.

10.28.3.9. Empfangen

Die empfangenen Daten stehen im Empfangspuffer der Basiskommunikationstask, sie können mit der Funktion 17 (11h) der Kommunikationstask abgeholt werden. Dazu müssen Sie zuerst eine Datenstruktur reservieren, die die empfangenen Daten aufnehmen soll, und die Adresse dieser Struktur übergeben. Werten Sie unbedingt den von der Funktion zurückgelieferten Fehlercode aus. Er gibt zum Beispiel an, ob die übergebenen Daten gültig sind.

Das folgende Beispiel kopiert empfangene Zeichen in einen String. Wenn das Funktionsergebnis 0 ist, sind die Daten gültig. Der beim Aufruf der Funktion übergebene String muss für mindestens 255 Zeichen Speicher reserviert haben.

Pascal:

```
FUNCTION rcv_string (channel: BYTE; VAR data: STRING): MAX_ERROR;
VAR
  dummyout: BYTE;
  insize: WORD;
  outsize: WORD;
  error: MAX_ERROR;
  task: WORD;
BEGIN
  task := $318 + channel;
  insize := 0;
  outsize := 255;

  { Die Empfangsdaten werden nicht an data, sondern an data[1] geschrieben,}
  { da das erste Byte von Data die Stringlänge enthält. }

  error := max_call_func(hModul,task,17,insize,dummyout,outsize,data[1]);

  data[0] := chr(outsize);        {Stringlänge einstellen}
  rcv_string := error;
END;
```

C:

```
MAX_ERROR rcv_string (UCHAR channel, char *data)
{
  void   *dummyout;
  USHORT insize, outsize;
  UCHAR   error;
  USHORT task;

  task = 0x318 + channel;
  insize = 0;
  outsize = 255;
  error = max_call_func(hModul,task,17,&insize,&dummyout,&outsize,data);

  data[outsize] = 0;  /* String mit 'Null' beenden */
  return(error);
}
```

Die Funktion überprüft dann den Status des Empfangspuffers, kopiert die empfangenen Zeichen in den angegebenen Puffer und liefert die Anzahl der tatsächlich gelesenen Zeichen zurück oder gegebenenfalls eine Fehlermeldung.

Um den Empfangsstatus zu ermitteln, liest man die zugehörigen Parameter (Nummern 262 und 264), in denen Fehlermeldungen sowie die Anzahl der Zeichen im Puffer enthalten sind.

10.28.3.10. Senden

Um Daten zu senden, müssen Sie die Funktion 7 der Basiskommunikation aufrufen und dabei die Sendedaten (max. 256 Bytes) übergeben. Werten Sie unbedingt die von der Funktion zurückgegebene Fehlermeldung aus! Sie enthält zum Beispiel Informationen darüber, ob die Daten in den Puffer eingetragen werden konnten.

Das folgende Beispiel zeigt eine Funktion zum Senden eines Strings. Der Rückgabewert ist 0, wenn der String in den Sendepuffer eingetragen werden konnte.

Pascal:

```
FUNCTION send_string (channel: BYTE; data: STRING): MAX_ERROR;
VAR
  dummyin: BYTE;
  error: MAX_ERROR;
  task, datalen, retlen: WORD;
BEGIN
  task := $318 + channel;

  { Die Sendedaten werden mit data[1] übergeben, da bei einer Übergabe mit }
  { data auch das erste Byte (= Stringlänge) übertragen würde.  }

  datalen := length(data);
  retlen := 0;
  error  := max_call_func(hModul,task,7,datalen,data[1],retlen,dummyin);
  send_string := error;
END;
```

C:

```
MAX_ERROR send_string (UCHAR channel, char *data)
{
  void *dummyin;
  USHORT datalen, retlen;
  MAX_ERROR error;
  USHORT task;

  task = 0x318 + channel;
  datalen = strlen(data);
  retlen = 0;
  error = max_call_func(hModul,task,7,&datalen,&data,&retlen,dummyin);
  return(error);
}
```

Das Basiskommunikationsprogramm sendet dann die Zeichen. Eine Statusabfrage, ob die Zeichen gesendet oder in den Sendepuffer übernommen werden können, wird durch die Funktion vorgenommen. Im Fehlerfall wird ein Fehlercode zurückgeliefert (siehe Tabelle der Fehlercodes, Anhang B-1).

Um den Sendestatus zu ermitteln (z.B. ob alle Zeichen gesendet wurden), liest man die zugehörigen Parameter (256 und 258), die Statusmeldungen (2 Byte) und die Anzahl der im Sendepuffer (4 Byte) enthaltenen Zeichen.

10.28.3.11. Die Parameter des Programms X1PA005.LIB

Alle Parameter, die in der folgenden Tabelle nicht beschrieben sind, sind reserviert und dürfen nicht geändert werden.

Nr.	Typ	Init	Zugr.[1]	Bedeutung des Parameters
0	UCHAR	0	R	Genereller Programmstatus: 0 = geladen, 1 = läuft, 2 = gestoppt, 3 = gestoppt nach Fehler
1	UCHAR	0	R	Fehlerinformation: gültig, wenn Parameter 0 = 3, siehe Fehlertabelle auf Seite 9-210
4	USHORT	0	R/W	Programmnummer des zugeh. Interruptmanagers
6	USHORT	0	R/W	Tasknummer des zugehörigen Interruptmanagers
8	UCHAR	0	R/W	Hier muss die Slot^Layer-Nr. angegeben werden, auf dem das MAX-Modul X-SCC-2 steckt (Bit 7..4 = Slot, Bit 3..0 = Layer).
9	UCHAR	0	R/W	Kanalnummer: Kanal A = 0, und B = 1
10	UCHAR	1	R/W	Physikalische Verbindung: 1 = RS-232, 3 = RS-422, 4 = RS-485
24	LONG	9600	R/W	Baudrate für Senden[2]
28	UCHAR	8	R/W	Zeichenlänge für Senden in Anzahl Bit: 5, 6, 7, 8
29	UCHAR	1	R/W	Anzahl Stopbit (Senden und Empfangen) 1 = ein Stopbit, 2 = zwei Stopbits, 3 = 1,5 Stopbits
30	UCHAR	0	R/W	Paritätsbit-Generierung (Senden und Empfangen): 0 = keine, 1 = ungerade, 2 = gerade
34	LONG	9600	R/W	Baudrate für Empfangen[2]
38	UCHAR	8	R/W	Zeichenlänge für Empfangen in[3] Anz. Bit: 5, 6, 7, 8
39	UCHAR	0FFh	R	Bitmaske für Zeichenlänge[4]

[1] Zugriff auf Parameter: R= Nur Lesen, R/W = Lesen und Schreiben

[2] Es sind beliebige Baudraten einstellbar. Nach Aufruf der Prozedur 2 wird hier die nächste realisierbare Baudrate eingetragen.

[3] Standardmässig liefert der Kommunikationsbaustein immer 8 Bit zurück, wobei bei Zeichenlängen kleiner 8 das höchstwertige Bit entsprechend der Parität gesetzt wird. Das Programm liefert normalerweise nur die gewünschte Anzahl von Bits/Zeichen. Alle anderen Bits werden = 0 gesetzt. Soll das Paritätsbit mitgeliefert werden, kann das über eine Bitmaske definiert werden (siehe Parameter 39)

[4] Bitmaske die definiert, welche Bits zurückgeliefert werden sollen. Bit = 1 bedeutet: Bit wird nicht gelöscht.

Nr.	Typ	Init	Zugr.[1]	Bedeutung des Parameters
40	LONG	0	R/W	Sendepuffer-Größe (in Byte)
48	LONG	0	R/W	Empfangspuffer-Größe (in Byte)
54	UCHAR	0	R/W	Empfangenes Byte bei Paritäts-Fehler verwerfen oder speichern: 0 = speichern, 1 = verwerfen
256	USHORT	0	R	Sendestatus (Bitmap) Bit 0 = Sendepuffer (Software) ist leer Bit 1 = alle Zeichen gesendet (Hardware) Bit 2 bis 14 sind reserviert Bit 15 = Sendeteil angehalten (durch PC oder Protokoll)
258	LONG	0	R	Anzahl Zeichen im Sendepuffer
262	USHORT	0	R	Empfangsstatus (Bitmap) Einmaliges Auftreten der Bedingung setzt das zugehörige Bit (Ausnahme Bit 15). Rücksetzung Bit 0 bis 4 durch Funktion 16 Bit 0 = Zeichenverlust wegen übergelaufenem Empfangspuffer Bit 1 = Zeichenverlust wegen Überlastung im ser. Controller Bit 2 = Paritäts-Fehler Bit 3 = Frame-Fehler Bit 4 = Break-Detection Bit 5 bis 14 sind reserviert Bit 15 = Empfangsteil angehalten (durch PC oder Protokoll)
264	LONG	0	R	Anzahl Byte im Empfangspuffer
268	USHORT	0	R	Zustand der Eingangs-Steuerleitungen (Bitmap) Bit 0 = CTS (Clear To Send) Bit 1 = DCD (Data Carrier Detected) Bit 2 = RI (Ring Indicator) Bit 3 bis 15 = reserviert
318	USHORT	0	R/W	Protokolltyp 0 = kein Protokoll 1 = XON/XOFF 2 = RTS/CTS

Tabelle der möglichen Fehler für einen Programmabbruch

(Wenn Parameter 0 = 3 ist, dann steht in Parameter 1 die Fehlerursache.)

Fehlernummer in Parameter 1	**Erklärung**
0	Reserviert
1	Nicht genügend Platz für Sendepuffer-Reservierung
2	Falscher Parameter für Sendepuffer-Reservierung
3	Unbekannter Fehler während Sendepuffer-Reservierung
4	Falscher Parameter für Sendepuffer-Reservierung
7	Ungültige Sendepuffer-Nr.
8	Sendepuffer wird gerade benutzt (locked)
11	Nicht genügend Platz für Empfangspuffer-Reservierung
12	Falscher Parameter für Empfangspuffer-Reservierung
13	Unbekannter Fehler während Empfangspuffer-Reservierung
14	Falscher Parameter für Empfangspuffer-Reservierung
17	Ungültige Puffernummer für Empfangspuffer
18	Empfangspuffer wird gerade benutzt (locked)
30	Fehler beim Aufruf einer externen Funktion (aufgerufen durch Aktions-Filter)
40	Fehler bei Interrupt-Service-Aufruf
50	Keine Quarzfrequenz in EEPROM eingetragen
99	Unbekannter Fehler

10.28.3.12. Die Funktionen und Prozeduren des Basiskommunikationsprogramms

Das Basiskommunikationsprogramm umfasst globale Prozeduren (ohne Übergabeparameter und Antwort) und globale Funktionen (mit Übergabe von Parametern und Antwort). In der folgenden Tabelle sind die Prozeduren in der Spalte 'Typ' mit P gekennzeichnet, die Funktionen mit F. Beim Aufruf von Funktionen müssen eine Reihe von Parametern übergeben werden. Sie sind in der folgenden Tabelle mit den Bezeichnern angegeben, mit der die Funktion über die Bibliotheken aufgerufen wird.

Nr.	Typ	Bedeutung der Funktion
2	P	Start/Restart der Kommunikation (Konfiguration, Speicherreserv.)
4	P	Baudrate aus Parameter 24 und 34 übernehmen und einstellen
5	F	**Sendebereitschaft ein- und ausschalten,** optional Sendepuffer löschen Hin: insize = 2 outsize = 0 indata = *Kontroll-Wort* (s.u.) *Kontroll-Wort:* 0 = Senden anhalten, Puffer nicht löschen 1 = Senden starten, Puffer nicht löschen 2 = Senden anhalten, Puffer löschen 3 = Senden starten, Puffer löschen
6	F	**Sendestatus melden** Hin: insize = 0 outsize = 6 outdata = *Rückgabe-Struktur* (s.u.) Rück: outsize = 6 *Rückgabe-Struktur:* Sendestatus (USHORT), Anzahl Zeichen im Sendepuffer (LONG), Bedeutung wie Parameter 256 und 258
7	F	**Zeichenkette an Sendepuffer übergeben** Hin: insize = Anzahl zu übergebender Zeichen outsize = 0 indata = *Übergabe-Puffer* (s.u.) *Übergabe-Puffer:* Datenstruktur, die *insize* zu sendende Bytes enthält.

Nr.	Typ	Bedeutung der Funktion
15	F	**Empfangsbereitschaft ein- und ausschalten**, optional Empfangspuffer löschen Hin: insize = 2 outsize = 0 indata = *Kontroll-Wort* (s.u.) *Kontroll-Wort:* 0 = Empfangen anhalten, Puffer nicht löschen 1 = Empfangen starten, Puffer nicht löschen 2 = Empfangen anhalten, Puffer löschen 3 = Empfangen starten, Puffer löschen
16	F	**Empfangsstatus melden** Hin: insize = 2 outsize = 6 outdata = *Rückgabe-Struktur* (s.u.) Rück: outsize = 6 *Rückgabe-Struktur:* Empfangsstatus (USHORT), Anzahl Zeichen im Empfangspuffer (LONG), Bedeutung wie Parameter 262 und 264
17	F	**Zeichenkette aus Empfangspuffer übernehmen** Hin: insize = 0 outsize (1) = Anzahl angeforderter Zeichen outdata = *Rückgabe-Puffer* (s.u.) Rück: outsize (2) = Anzahl tatsächlich gelesener Zeichen *Rückgabe-Puffer:* Datenstruktur, die die gelesenen Zeichen aufnehmen kann, also mindestens *outsize* (1) Byte umfasst. Nach dem Aufruf sind die ersten *outsize* (2) Byte gültig.

Nr.	Typ	Bedeutung der Funktion
33	F	**Steuerleitungs-Ausgänge setzen** Hin: insize = 4 outsize = 0 indata = *Übergabe-Struktur* (s.u.) *Übergabe-Struktur*: Wort 1: Maske für die betroffenen Steuerleitungen (USHORT), Bit = 1: Steuerleitung soll geändert werden Wort 2: Information, wie die Steuerleitung gesetzt werden soll Bitmap (Wort 1 und 2): Bit 0 = RTS Bit 1 = DTR Bit 2 = TMT-Break Bit 3 bis 15 reserviert
34	F+	**Zustand der Steuerleitungs-Eingänge lesen** Hin: insize = 0 outsize = 2 outdata = *Rückgabe-Struktur* (s.u.) *Rückgabe-Struktur:* Information, wie die Steuerleitung gesetzt wurde (USHORT) Bitmap: Bit 0 = CTS Bit 1 = DCD Bit 2 = RI Bit 3 = DSR Bit 4 bis 15 reserviert

Fehlerrückgabecodes von Funktionen des Programms X1PA005

Alle zurückgelieferten Fehler sind Meldungen vom Betriebssystem. Die folgende Tabelle zeigt, welche Fehler bei den einzelnen Funktionen auftauchen können und was sie bedeuten:

Funktion	Fehlernummer	Bedeutung
5	22h oder 23h 26h	Sendepuffer gesperrt Sendepuffer-Nummer ungültig
6	26h	Sendepuffer-Nummer ungültig
7	22h oder 23h 24h 26h	Sendepuffer gesperrt Sendepuffer voll Sendepuffer-Nummer ungültig
15	22h oder 23h 26h	Empfangspuffer gesperrt Empfangspuffer-Nummer ungültig
16	26h	Empfangspuffer-Nummer ungültig
17	22h oder 23h 26h	Empfangspuffer gesperrt Empfangspuffer-Nummer ungültig

Bei den Fehlermeldungen 'Puffer voll' bzw. 'Puffer gesperrt' können Sie den Funktionsaufruf zu einem späteren Zeitpunkt wiederholen. Damit sich der Pufferzustand ändern kann, müssen Sie die Kontrolle nach dem ersten Funktionsaufruf (der den Fehler gemeldet hat) zuerst wieder an das Betriebssystem zurückgeben. !

10.28.4. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Angaben in Klammern sind Pins, die in dieser Betriebsart nicht angeschlossen werden sollten.

Pin		**RS-232**	**RS-422**	**RS-485/1**	**RS-485/2**	**20 mA**
A	**B**					**Current Loop**
1	11	**DCD**	**RCV-**	**n.c.**	**n.c.**	(DCD RS-232)
2	12	**DSR**	**CTS-**	**n.c.**	**CTS-**	(DSR RS-232)
3	13	**RCV**	**RCV+**	**n.c.**	**n.c.**	(RCV RS-232, ohne Funktion)
4	14	**RTS**	**RTS-**	**RTS-/CTS-**	**RTS-**	(RTS RS-232)
5	15	**TMT**	**TMT-**	**RCV-/TMT-**	**RCV-/TMT-**	(TMT RS-232)
6	16	**CTS**	**CTS+**	**n.c.**	**CTS+**	(CTS RS-232)
7	17	**DTR**	**TMT+**	**RCV+/TMT+**	**RCV+/TMT+**	(DTR RS-232)
8	18	**Ri**	**(Ri-Input)**	**(Ri-Input)**	**(Ri-Input)**	(Ri RS-232)
9	19	**GND**	**GND**	**GND**	**GND**	(GND)
10	20	**CLKio**	**RTS+**	**RTS+/CTS+**	**RTS+**	(CLKio RS-232)
21	31	(-12V)	(-12V)	(-12V)	(-12V)	**-12V Ausgang**
22	32	(TMT+)	(TMT+)	(TMT+)	(TMT+)	**TMT+**
23	33	(TMT-)	(TMT-)	(TMT-)	(TMT-)	**TMT-**
24	34	(GND)	(GND)	(GND)	(GND)	**GND**
25	35	(CCS1+)	(CCS1+)	(CCS1+)	(CCS1+)	**CCS1+**
26	36	(RCV+)	(RCV+)	(RCV+)	(RCV+)	**RCV+**
27	37	(RCV-)	(RCV-)	(RCV-)	(RCV-)	**RCV-**
28	38	(GND)	(GND)	(GND)	(GND)	**GND**
29	39	(CCS2+)	(CCS2+)	(CCS2+)	(CCS2+)	**CCS2+**
30	40	(+12V)	(+12V)	(+12V)	(+12V)	**+12V Ausgang**

<n.c.> Diese Eingänge haben in dieser Betriebsart keine Funktion, sind aber hochohmig galvanisch angeschlossen. Die Spannungen an diesen Pins darf die max. erlaubte Eingangsspannung nicht überschreiten (siehe Besondere Eigenschaften).

<1> CCSn (Constant Current Source) sind 4 Konstantstromquellen, je 2 für jeden Kanal.

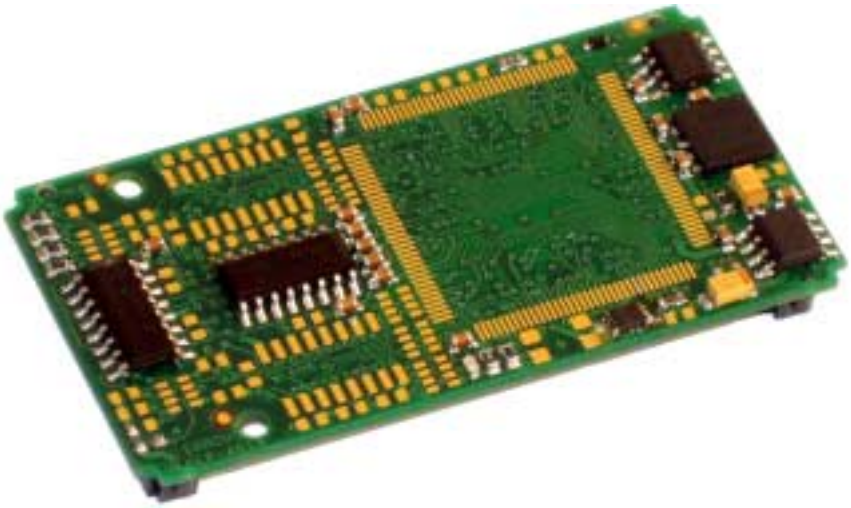
10.28.5. Besondere Eigenschaften

Parameter	Wert	Einheit
Anzahl serieller Schnittstellen	2	
per Software wählbar	RS-232, RS-422, RS-485 oder 20 mA Current Loop (dann galv. getrennt)	
max. Baudrate je Kanal	920,75	kBaud
FIFO (je Kanal)	8	Byte
Controller	16 MHz ESCC	
RS-232		
Eingangsspannung, max.	±25	V
Eingangsschwelle Low, min.	0,6	V
Eingangsschwelle High, max.	2,0	V
Hysterese, typ.	0,5	V
Eingangswiderstand, min./typ./max.	3/5/7	kΩ
Ausgangsspannung (3 kOhm an GND), min./typ.	±5/±5,4	V
Ausgangsstrom (Ausgang an GND), max.	±60	mA
RS-422 und RS-485		
Eingangswiderstand, min.	48	kΩ
Eingangsstrom, max. (-7V .. +12V)	-0,15/0,25	mA
Differentielle Eingangsschwelle, min./max.	-200/-50	mV
Hysterese, typ.	30	mV
Differentielle Ausgangsspannung, min. (RS-422/RS-485) (RS-485 = 27 Ohm, RS-422 = 50 Ohm)	2/1,5	V
Temperatur-Bereich, Betrieb	-40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht, X-SCC-2/R / X-SCC-2/U	9,45/10,85	g
Stromaufnahme		
3,3V, typ. (X-SCC-2/R / X-SCC-2/U)	255/305	mA
12V (X-SCC-2/R)	55	mA
-12V (X-SCC-2/R)	25	mA

Notizen

X-SSI-2

2 synchron serielle Schnittstellen



10.29. X-SSI-2: Synchron Serielles Interface mit 2 Kanälen

Inhaltsverzeichnis

10.29.1. Beschreibung

Das Modul X-SSI-2 stellt 2 SSI-Kanäle zur Verfügung. Die angeschlossenen Geber werden zeitgleich abgetastet und können anschließend eingelesen werden.

Die Zahl der Bits pro Kanal ist programmierbar von 2 bis 32 Bit, ebenso die Codierung des Ergebnisses (Gray-Code oder binär). Die Taktrate kann zwischen 16,2 kHz bis 8,25 MHz programmiert werden.

Achtung: Einige Weggeber (z.B. Stegmann) brauchen nach der ersten Taktflanke bis zum Auslesen etwas länger als bei den folgenden Bits. In solchen Fällen muss der Takt sich nach der längeren Zeit richten. Andere Weggeber haben eine Spannungsüberwachung, deren Zustand in einem Datenbit mitgeliefert wird. Ist dieses Bit gesetzt, ist das Ergebnis ungültig. Auch Gray-Codierung ist bei diesen Gebern möglich, weil das Status-Bit im MSB übertragen wird.

Jeder Kanal hat einen eigenen, differentiellen Taktausgang und einen differentiellen Dateneingang. Nach dem Einschalten des Systems ist CLK+ = log. 1, CLK- = log. 0. Beide sind über RS-422 Treiber mit dem Logikteil des Moduls verbunden.

Zusätzlich stehen zwei digitale Eingänge und zwei digitale Ausgänge zur Verfügung. Diese werden ebenfalls über RS-422 Treiber zum Logikteil geführt.

Ausserdem verfügt das Modul über eine per Software schaltbare LED.

10.29.2. Modul-Device-Treiber

10.29.2.1. Installation

Der Modul-Device-Treiber für das OsX hat die Programmnummer 8044h und den Dateinamen mxssi2.exe. Der Modul-Device-Treiber für Windows hat den Namen mxssi2.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8044, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8044

10.29.2.2. Kanaleigenschaftsstruktur CPS_XSSI2_A

Die CPS für das Modul hat den Namen CPS_XSSI2_A.
CPS_XSSI2_A wurde gegenüber CPS_XSSI2 um den Parameter *usVersion* erweitert.

10.29.2.3. SSI-Schnittstellen

Das Modul enthält zwei SSI-Kanäle (0 .. 1). Um auf die Schnittstellen zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*1*	Version dieser CPS Definition
.usDevice	*DEVICE_SSI*	Kanal auf eine SSI-Schnittstelle
.usIndexFirst	*0 ... 1*	Nummer der ersten Schnittstelle
.usIndexLast	*0 ... 1*	Nummer der letzten Schnittstelle
.usFlags[1]	*_CP_EXCLUSIVE*	Muss gesetzt sein! Der Zugriff erfolgt immer exklusiv.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet.
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*0*	Keine Bedeutung
.usMode	*XSSI2_AUTO_MODE*	Ist dieser Modus aktiviert, startet der Eingabedienst eine neue Mesung und liefert die Daten zurück. Können innerhalb einer Timeoutzeit keine Daten empfangen werden, liefert der Eingabedienst einen Fehler zurück. In diesem Modus kann keine Callback-Funktion verwendet werden.
	XSSI2_MANUAL_MODE	Ist dieser Modus aktiviert, wird die Messung mit einem Steuerkommando gestartet (siehe Sonderdienste). Das Ende der Messung kann über eine Callback-Funktion[2] signalisiert werden. Dieser wird der Status und das Ergebnis der Messung übergeben. Danach kann die nächste Messung gestartet werden. Wurde der Kanal über mehrere SSI-Schnittstellen geöffnet, so wird die Callback-Funktion erst dann aufgerufen, wenn alle Kanäle die Messung beendet haben. Wird keine Callback-Funktion verwendet, liefert der Eingabedienst solange die Messung noch läuft einen Fehler

[1] Das Flag _XSSI2_MANUAL_MODE der CPS_XSSI2 wurde nach usMode verschoben
[2] Erst ab Rev. D des Moduls verfügbar

Strukturelement	Werte	Bedeutung
		ERR_DATA_NOT_READY zurück. Kann innerhalb einer Timeoutzeit kein gültiges Ergebnis empfangen werden, liefert der Eingabedienst den Fehler ERR_INVALID_DATA zurück. Danach kann die nächste Messung gestartet werden.
.usDataMode[3]	*0*	Binary-Code
	XSSI_GRAY_TO_BINARY	Gray-Code direkt in Binary-Code umwandeln
.ulClock[3]	*16200 ... 8250000*	Taktfrequenz der Ssi-Übertragung in Hz (16,2 KHz ... 8,25 MHz)
.usBit[3]	*2 ... 32*	Anzahl der Datenbits

Eingabedienst

Der Datentyp des Kanals ist DATA_ULONG, der Zugriff erfolgt mit:

- **max_read_channel_ulong** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Liegen keine Daten vor, so liefert die Funktion den Fehler ERR_DATA_NOT_READY bzw. ERR_INVALID_DATA zurück.

Sonderdienst

- **max_channel_control,** Steuerbefehl CMD_START: Ist der Modus XSSI2_MANUAL_MODE gesetzt, kann die Messung mit dem Kanal-Steuerkommando gestartet werden. Dem Dienst werden keine Daten übergeben.

[3] Rev. A und B: Beide Kanäle müssen die gleichen Einstellungen haben

Callback-Funktion[4]

Wenn beim Öffnen des Kanals im Modus XSSI2_MANUAL_MODE eine Callback-Funktion angegeben wird, wird diese aufgerufen wenn neue Daten vorliegen.

Die Callback-Funktion bekommt pro SSI-Kanal zwei ULONG-Werte übergeben. Der erste ULONG-Wert liefert den Status des Kanals, der zweite ULONG-Wert den Messwert des Kanals. Der Status des Kanals kann folgende Werte annehmen: ERR_OK – die Daten des Kanals sind gültig; ERR_INVALID_DATA – der Kanal konnte keine gültigen Daten empfangen, die Daten sind daher ungültig.

10.29.2.4. Digitale Eingänge[4]

Das Modul verfügt über zwei digitale Eingänge DIN-0 und DIN-1 die über RS-422 Treiber mit dem Logikteil des Moduls verbunden sind. Um auf die digitale Eingänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*1*	Version dieser CPS Definition
.usDevice	*DEVICE_DIN*	Kanal auf einen digitalen Eingang
.usIndexFirst	*0 ... 1*	Nummer der ersten Eingangs
.usIndexLast	*0 ... 1*	Nummer der letzten Eingangs
.usFlags	*0*	Keine Bedeutung
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usMode	0	Keine Bedeutung

Eingabedienst

Die Daten sind rechtsbündig angegeben. Der Datentyp des Kanals ist DATA_UCHAR, der Zugriff erfolgt mit:

- **max_read_channel_uchar**

[4] Erst ab Rev. D des Moduls verfügbar

10.29.2.5. Digitale Ausgänge[4]

Das Modul verfügt über zwei digitale Ausgänge DOUT-0 und DOUT-1 die über RS-422 Treiber mit dem Logikteil des Moduls verbunden sind. Um auf die digitalen Ausgänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*1*	Version dieser CPS Definition
.usDevice	*DEVICE_DOUT*	Kanal auf einen digitalen Ausgang
.usIndexFirst	*0 ... 1*	Nummer der ersten Eingangs
.usIndexLast	*0 ... 1*	Nummer der letzten Eingangs
.usFlags	*_CP_EXCLUSIVE*	Muss gesetzt sein. Der Zugriff erfolg immer exklusiv.
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibugriff
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usMode	0	Keine Bedeutung

Ausgabe- und Eingabedienst

Die Daten sind rechtsbündig angegeben. Der Datentyp des Kanals ist DATA_UCHAR, der Zugriff erfolgt mit:

- **max_write_channel_uchar**
- **max_read_channel_uchar**

10.29.2.6. LED[5]

Das Modul verfügt über eine schaltbare LED. Um auf die LED zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*1*	Version dieser CPS Definition
.usDevice	*DEVICE_LED*	Kanal auf die LED
.usIndexFirst	*0*	Nummer der LED
.usIndexLast	*0*	Nummer der LED
.usFlags	*0*	Keine Bedeutung
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff

[4] Erst ab Rev. D des Moduls verfügbar
[5] Erst ab Rev. B des Moduls verfügbar

Strukturelement	Werte	Bedeutung
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff
.usMode	0	Keine Bedeutung

Eingabe- und Ausgabedienst

Um die LED ein- bzw. auszuschalten muss eine 1 bzw. 0 in den Kanal geschrieben werden. Der Datentyp des Kanals ist DATA_UCHAR, der Zugriff erfolgt mit:

- **max_read_channel_uchar**
- **max_write_channel_uchar**

10.29.2.7. X-Bus Takt

Das Modul benötigt zur Erzeugung des SSI-Taktes den X-Bus Takt. Falls der Treiber die Taktfrequenz des X-Busses nicht ermitteln kann, erhält man beim Öffnen des Kanals die Fehlermeldung ERR_SYSTEM_CLOCK. In diesem Fall muss dem Treiber mit Hilfe eines DEVICE_CTRL Kanals die Taktfrequenz des X-Bus mitgeteilt werden. Mit folgender CPS kann sie dem Treiber mitgeteilt werden:

Strukturelement	Werte	Bedeutung
.usDevice	*DEVICE_CTRL*	Steuerkanal
.usIndexFirst	*XSSI2_SYSTEM_CLOCK*	Index für Systemclock
.usIndexLast	*XSSI2_SYSTEM_CLOCK*	Index für Systemclock
.usFlags	*0*	Reservierter Parameter

Ausgabedienst

Der Datentyp des Kanals ist DATA_ULONG. Mit einem Schreibzugriff auf den Kanal kann dem MDD die Taktfrequenz (in Hz) des X-Bus mitgeteilt werden. Standardmäßig ist der X-Bus mit 33 MHz getaktet (XSSI2_33MHz).

- **max_write_channel_ulong**

10.29.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Kanal 0		Kanal 1	
Pin	**Beduetung**	**Pin**	**Bedeutung**
1	CLK+	6	CLK+
2	CLK-	7	CLK-
3	DATA+	8	DATA+
4	DATA-	9	DATA-
5	GND	10	GND
20	GND	40	GND

Alle nicht angegebenen Pins müssen frei bleiben (Pin 5, 10, 11 bis 19, 21 bis 39), sie sind ggfls. intern auf dem Modul angeschlossen. Sie sind für zukünftige Erweiterungen vorgesehen.

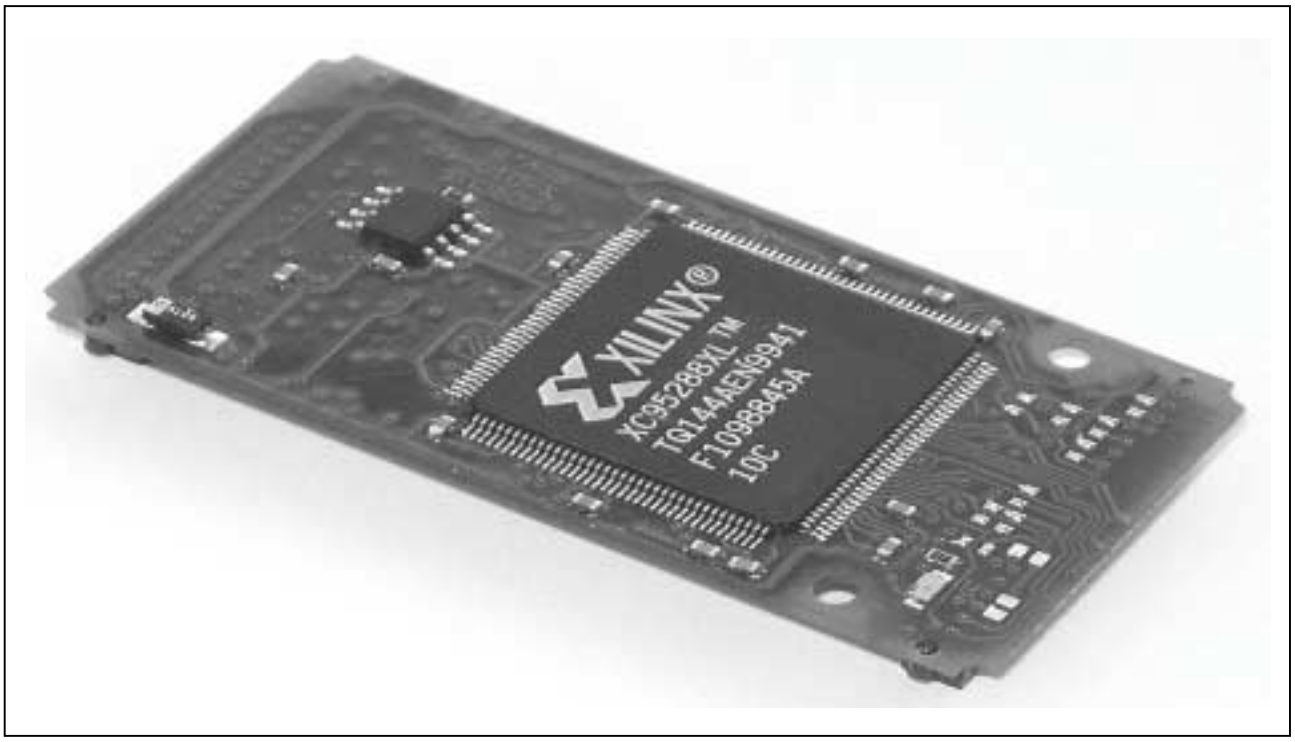
10.29.4. Besondere Eigenschaften

Parameter		Wert	Einheit
Anzahl SSI-Kanäle		2	-
Takt programmierbar (bei X-Bus Takt = 33 MHz)		16,1448 .. 8250	kHz
Länge der Daten programmierbar		2 .. 32	Bit
Codierung, programmierbar		binär, Gray-Code	-
Eingangsspannung (kompatibel mit 5V TTL und 5V, 3,3V und 2,5V CMOS)			
	log. 0	< 0,8	V
	log. 1	> 2,0	V
Input Leakage Current, max.		10	µA
Ausgangsspannung (kompatibel mit 5V TTL und 3,3V CMOS)			
	log. 0, max. (IOL = 8mA)	0,4	V
	log. 1, min. (IOH = -4mA)	2,4	V
Ausgangstrom,	max. log. 0	-12	mA
	min. log. 1	+12	mA
Überspannungsfestigkeit der Eingänge		-0,5 .. +5,5	V
für Impulse < 10 ns und < 200mA		−2,0 .. +7,0	V
Temperatur-Bereich, Betrieb		0 .. +70	°C
	optional (bitte anfragen)	−40 .. +85	°C
Abmessungen		29x58x8	mm
Gewicht		7,8	g
Stromaufnahme (3,3V) (die X-Bus Spannungen ±12V werden nicht benötigt)		280	mA

Notizen

X-SSI-2/M

Mithören von 2 synchron seriellen Schnittstellen



10.30. X-SSI-2/M: Synchron Serielles Mithörmodul mit 2 Kanälen

Inhaltsverzeichnis

10.30.1. Beschreibung

Das Modul X-SSI-2/M kann 2 bestehende SSI-Verbindungen protokollieren. Jede der beiden Verbindungen kann unabhängig mitgehört werden.

Die Zahl der Bits pro Kanal ist programmierbar von 2 bis 32 Bit, ebenso die Codierung des Ergebnisses (Gray-Code oder binär). Mitgehört werden können Module bis zu einer Taktrate von 8,25 MHz. Diese Einstellungen können für beide Kanäle unabhängig vorgenommen werden.

Jeder Kanal hat einen eigenen, differentiellen Takteingang und einen differentiellen Dateneingang. Beide sind über RS-422 Treiber mit dem Logikteil des Moduls verbunden.

Zusätzlich stehen vier digitale Eingänge zur Verfügung. Diese werden ebenfalls über RS-422 Treiber zum Logikteil geführt.

Ausserdem verfügt das Modul über eine per Software schaltbare LED

10.30.2. Modul-Device-Treiber

10.30.2.1. Installation

Der Modul-Device-Treiber für das OsX hat die Programmnummer 8044h und den Dateinamen mxssi2.exe. Der Modul-Device-Treiber für Windows hat den Namen mxssi2.sys. Die Installation aus einem PC-Programm (z.B. für Steckplatz 1, Layer 0):

Error = max_load_mdd (hModul, 1, 0, 0, 0x8044, NULL, &hMDD);

Befehl in einer INS-Datei (z.B. für Steckplatz 1, Layer 0):

MAXLOADMDD slot=1 layer=0 progno=8044

10.30.2.2. Kanaleigenschaftsstruktur

Die CPS für das Modul hat den Namen CPS_XSSI2_A.

CPS_XSSI2_A wurde gegenüber CPS_XSSI2 um den Parameter *usVersion* erweitert.

10.30.2.3. SSI-Kanäle

Das Modul kann zwei SSI-Verbindungen mithören, Kanäle (0 .. 1). Um auf diese Kanäle zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVerion	*1*	Version dieser CPS Definition
.usDevice	*DEVICE_SSI*	Kanal auf eine SSI-Schnittstelle
.usIndexFirst	*0 ... 1*	Nummer der ersten Schnittstelle
.usIndexLast	*0 ... 1*	Nummer der letzten Schnittstelle
.usFlags[16]	*_CP_EXCLUSIVE*	Muss gesetzt sein!. Der Zugriff erfolgt immer exklusiv.
	_CP_SYNC_CALLBACK	Nur in Echtzeitprogrammen verwendbar! Der Aufruf der Callback-Funktion erfolgt direkt. Ansonsten werden die Aufrufe gepuffert und im „Hintergrund“ (NI-Task) abgearbeitet
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*0*	Keine Bedeutung
.usMode	*XSSI2_MANUAL_MODE*	Ist dieser Modus aktiviert, wird die Messung mit einem Steuerkommando gestartet (siehe Sonderdienste). Das Ende der Messung kann über eine Callback-Funktion[2] signalisiert werden. Dieser wird der Status und das Ergebnis der Messung übergeben. Danach kann die nächste Messung gestartet werden. Wurde der Kanal über mehrere SSI-Schnittstellen geöffnet, so wird die Callback-Funktion erst dann aufgerufen, wenn alle Kanäle die Messung beendet haben. Wird keine Callback-Funktion verwendet, liefert der Eingabedienst solange den Fehler ERR_DATA_NOT_READY bis eine Messung vorliegt. Danach kann die nächste Messung gestartet werden.

[16] Das Flag _XSSI2_MANUAL_MODE der CPS_XSSI2 wurde nach usMode verschoben
[2] Erst ab Rev. B des Moduls verfügbar

Strukturelement	Werte	Bedeutung
	XSSI2_CONTINUOUS_MODE[2]	Ist dieser Modus aktiviert, misst das Modul kontinuierlich. Der Eingabedienst liefert das Ergebnis der letzten Messung zurück. Ist seit dem letzten Auslesen kein neuer Messwert mehr empfangen worden, liefert der der Dienst den Fehler ERR_DATA_NOT_READY zurück. Das Ende der Messungen kann über eine Callback-Funktion signalisiert werden. Dieser wird der Status und das Ergebnis der Messung übergeben.
.usDataMode	*0* *XSSI_GRAY_TO_BINARY*	Binary-Code Gray-Code direkt in Binary-Code umwandeln
.ulClock	*16200 ... 8250000*	Taktfrequenz der Ssi-Übertragung in Hz (16,2 KHz ... 8,25 MHz)
.usBit	*2 ... 32*	Anzahl der Datenbits

Eingabedienst

Der Datentyp des Kanals ist DATA_ULONG, der Zugriff erfolgt mit:

- **max_read_channel_ulong** (Einzelkanal)
- **max_read_channel_block** (Mehrkanal)

Liegen keine Daten vor, so liefert die Funktion den Fehler ERR_DATA_NOT_READY bzw. ERR_READ-IN_PROGRESS zurück.

Sonderdienst

- **max_channel_control,** Steuerbefehl CMD_START: Ist der Modus XSSI2_MANUAL_MODE gesetzt, kann die Messung mit dem Kanal-Steuerkommando gestartet werden. Dem Dienst werden keine Daten übergeben.

Callback-Funktion[2]

Wenn beim Öffnen des Kanals eine Callback-Funktion angegeben wird, wird diese aufgerufen wenn neue Daten vorliegen.

Die Callback-Funktion bekommt pro SSI-Kanal zwei ULONG-Werte übergeben. Der erste ULONG-Wert liefert den Status der Kanals, der zweite ULONG-Wert den Messwert des Kanals. Ist der Status ungleich ERR_OK, wurde bei der Messung des Kanals ein Fehler entdeckt und die Daten sind daher ungültig.

[2] Erst ab Rev. B des Moduls verfügbar

10.30.2.4. Digitale Eingänge[2]

Das Modul verfügt über vier digitale Eingänge DIN-0 ... DIN-3 die über RS-422 Treiber mit dem Logikteil des Moduls verbunden sind. Um auf die digitale Eingänge zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*1*	Version dieser CPS Definition
.usDevice	*DEVICE_DIN*	Kanal auf einen digitalen Eingang
.usIndexFirst	*0 ... 3*	Nummer der ersten Eingangs
.usIndexLast	*0 ... 3*	Nummer der letzten Eingangs
.usFlags	*0*	Keine Bedeutung
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usMode	0	Keine Bedeutung

Eingabedienst

Die Daten sind rechtsbündig angegeben. Der Datentyp des Kanals ist DATA_UCHAR, der Zugriff erfolgt mit:

- **max_read_channel_uchar**

10.30.2.5. LED[2]

Das Modul verfügt über eine schaltbare LED. Um auf die LED zugreifen zu können, muss folgende CPS verwendet werden:

Strukturelement	Werte	Bedeutung
.usVersion	*1*	Version dieser CPS Definition
.usDevice	*DEVICE_LED*	Kanal auf die LED
.usIndexFirst	*0*	Nummer der LED
.usIndexLast	*0*	Nummer der LED
.usFlags	*0*	Keine Bedeutung
.usReadMode	*IO_MODE_DIRECT*	Direkter Lesezugriff
.usWriteMode	*IO_MODE_DIRECT*	Direkter Schreibzugriff

[2] Erst ab Rev. B des Moduls verfügbar

Strukturelement	Werte	Bedeutung
.usMode	0	Keine Bedeutung

Eingabe- und Ausgabedienst

Um die LED ein- bzw. auszuschalten muss eine 1 bzw. 0 in den Kanal geschrieben werden. Der Datentyp des Kanals ist DATA_UCHAR, der Zugriff erfolgt mit:

- **max_read_channel_uchar**
- **max_write_channel_uchar**

10.30.3. Anschlusspins des Moduls (bezogen auf den Modul-Stecker A)

Pin	Bedeutung	Pin	Bedeutung
1	CLK0+: Takt-Eingang Kanal 0, +	21	n.c.
2	CLK0-: Takt-Eingang Kanal 0, -	22	n.c.
3	DATA0+: Daten-Eingang Kanal 0, +	23	n.c.
4	DATA0-: Daten-Eingang Kanal 0, -	24	n.c.
5	GND	25	n.c.
6	CLK1+, Takt-Eingang Kanal 1, +	26	n.c.
7	CLK1-, Takt-Eingang Kanal 1, -	27	n.c.
8	DATA1+, Daten-Eingang Kanal 1, +	28	n.c.
9	DATA1-, Daten-Eingang Kanal 1, -	29	n.c.
10	GND	30	n.c.
11	RS422in0+, extra Eingang 0, +	31	n.c.
12	RS422in0-, extra Eingang 0, -	32	n.c.
13	RS422in1+, extra Eingang 1, +	33	n.c.
14	RS422in1-, extra Eingang 1, -	34	n.c.
15	GND	35	n.c.
16	RS422in2+, extra Eingang 2, +	36	n.c.
17	RS422in2-, extra Eingang 2, -	37	n.c.

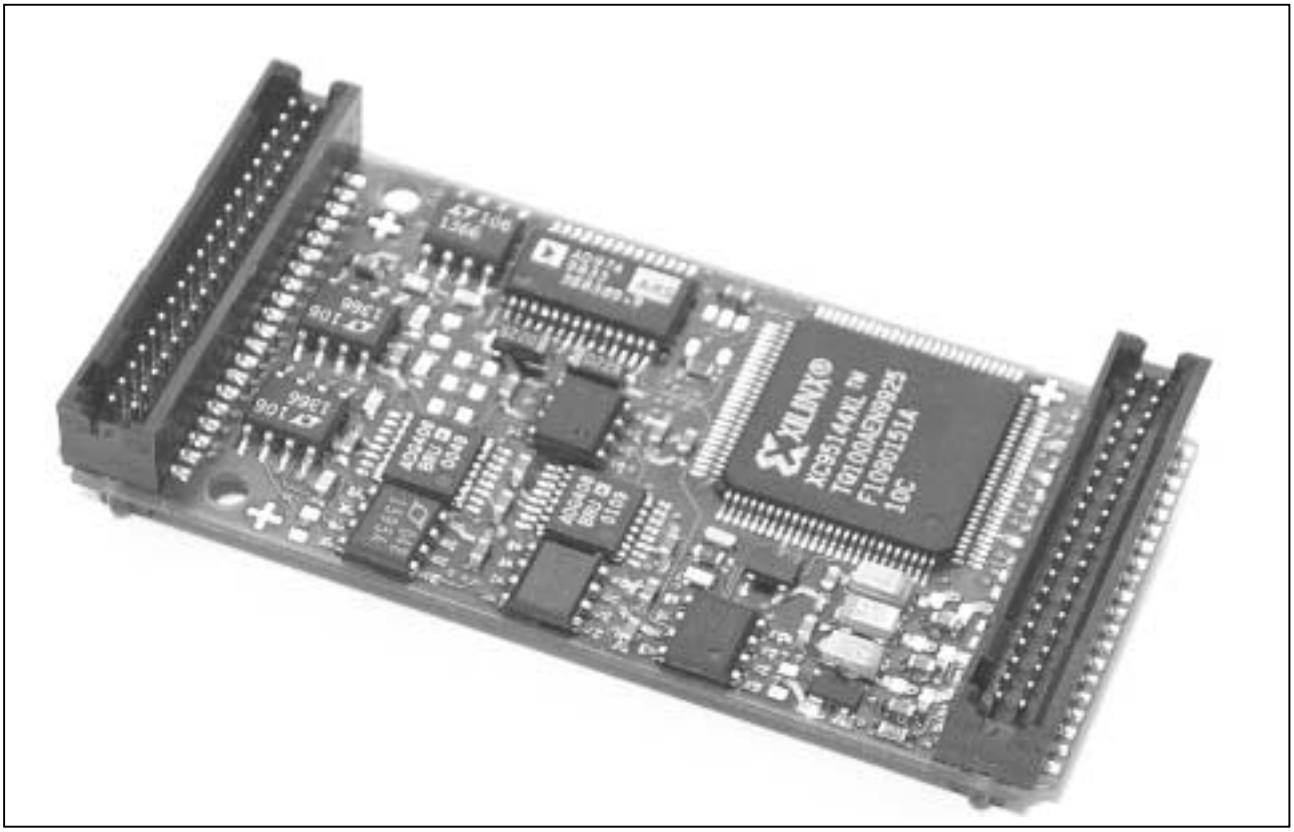
Pin	**Bedeutung**	**Pin**	**Bedeutung**
18	RS422in3+, extra Eingang 3, +	38	n.c.
19	RS422in3-, extra Eingang 3, -	39	n.c.
20	GND	40	n.c.

10.30.4. Besondere Eigenschaften

Parameter		Wert	Einheit
Anzahl mitzuhörender SSI-Kanäle		2	-
Takt programmierbar (bei X-Bus Takt = 33 MHz)		16,1448 .. 8250	kHz
Länge der Daten programmierbar		2 .. 32	Bit
Codierung, programmierbar		binär, Gray-Code	-
Eingangsspannung (kompatibel mit 5V TTL und 5V, 3,3V und 2,5V CMOS)			
	log. 0	< 0,8	V
	log. 1	> 2,0	V
Input Leakage Current, max.		10	µA
Ausgangsspannung (kompatibel mit 5V TTL und 3,3V CMOS)			
	log. 0, max. (IOL = 8mA)	0,4	V
	log. 1, min. (IOH = -4mA)	2,4	V
Ausgangstrom,	max. log. 0	-12	mA
	min. log. 1	+12	mA
Überspannungsfestigkeit der Eingänge		-0,5 .. +5,5	V
für Impulse < 10 ns und < 200mA		−2,0 .. +7,0	V
Temperatur-Bereich, Betrieb		0 .. +70	°C
	optional (bitte anfragen)	−40 .. +85	°C
Abmessungen		29x58x8	mm
Gewicht		7,8	g
Stromaufnahme (3,3V) (die X-Bus Spannungen ±12V werden nicht benötigt)		280	mA

X-TEST-1

Testmodul für alle X40-Bus Module



10.31. X-TEST-1

Inhaltsverzeichnis

10.31.1. Beschreibung

Dieses Modul kann alle 40 Pins des A-Steckers eines zu testenden Moduls testen. Dabei kann jeder Pin galvanisch getrennt oder nicht getrennt, analog oder digital, Eingang, Ausgang oder bidirektional sein. Lediglich Spannungen über ±10 Volt können nicht getestet werden.

Zusätzlich kann die Stromaufnahme der Versorgungsspannungen des zu testenden Moduls (3,3 Volt, +12 Volt und –12 Volt) gemessen werden.

Das zu testende Modul wird einfach auf dieses Modul aufgesteckt. Mit Hilfe des SORCUS Programms SNW32 kann dann ein automatischer Test des zu testenden Moduls durchgeführt werden.

10.31.2. Besondere Eigenschaften

Parameter	Wert	Einheit
Analogeingänge, Anzahl extern	40	-
je Kanal wählbar per Software als Masse-bezogener Eingang (SE) oder je 2 Eingänge als Differenzeingang, auch gemischter Betrieb möglich		
Auflösung	16	Bit
Eingangsbereich	±10	V
Wandlungszeit A/D-Wandler	25	µs
Überspannungsfestigkeit (Power on oder off)	–12 .. +12	V
Messung der Stromaufnahme des zu testenden Moduls		
Bereich (für +3,3V, +12V, -12V)	0 .. 500	mA
Fehler	< 1	%
Temperaturmessung on-board, Bereich	-55 .. +125	°C
Genauigkeit	±2	°C
Analogausgänge, Anzahl extern	2	
Ausgangsbereich	±10	V
Auflösung	16	Bit
Messung des Stroms in/aus Pins des zu testenden Moduls	±100	mA
Temperatur-Bereich, Betrieb	-40 .. +85	°C
Abmessungen	29x58x8	mm
Gewicht	tbd	g

Parameter	**Wert**	**Einheit**
Stromaufnahme (ohne Last)		
3,3V	tbd	mA
12V	tbd	mA
–12V	tbd	mA

Notizen

A. Modulübersicht

Die hier aufgeführten Module sind ohne Einschränkungen zusammen mit dem MAX-PC einsetzbar, sofern nichts anderes vermerkt ist. Bitte beachten Sie folgende Hinweise:

Typ	Sub-typ	Modul	Prog.-Nr. des MDD	Beschreibung
11	0	X-MAX-1	800bh	MAX-PC
12	0	X-MAX-E	800ch	MAX-PC mit Ethernet
14	0, 1	X-MAX-400		MAX-PDA mit 400 MHz CPU
	2	X-MAX-200		MAX-PDA mit 200 MHz CPU
16	0	X-REL-8/NO	8010h	8 Relaisausgänge (1 x Um und 1x Ein).
	1	X-REL-8/NC		8 Relaisausgänge (1 x Um und 1x Aus).
	2	X-REL-8/U		8 Relaisausgänge
20	0	X-AD24-4i/S	8014h	4 einzeln isol. DMS-Kanäle, 24 Bit, 100 sps
	1	X-AD24-4i/T		4 einzeln isol. Thermoelement-Kanäle, 24 Bit, 100 sps
	2	X-AD24-4i/P		4 einzeln isol. Pt100-Kanäle, 24 Bit, 100 sps
	3	X-AD24-4i/I		4 einzeln isol. Kanäle für ICP-Sensoren, 24 Bit, 100 sps
	4	X-AD24-4i/C		4 einzeln isol. Stromeingänge, 0..20 mA, 24 Bit, 40 Ksps
	5	X-AD24-4i/V		4 einzeln isol. Spannungseingänge, +/-10 V, 24 Bit, 40 Ksps
	6	X-AD24-4i/F		4 einzeln isol. DMS-Eingänge, 24 Bit, 40 Ksps
24	0	X-OPT-200/L	8018h	20 opto-entkoppelte digitale Eingänge
	1	X-OPT-200/P		
	2	X-OPT-200/H		
25	0	X-OPT-164/L	8019h	16 opto-entkoppelte digitale Ein- und 4 Ausgänge
	1	X-OPT-164/P		
	2	X-OPT-164/H		
26	0	X-OPT-128/L	801ah	12 opto-entkoppelte digitale Ein- und 8 Ausgänge
	1	X-OPT-128/P		
	2	X-OPT-128/H		
27	0	X-OPT-812/L	801bh	8 opto-entkoppelte digitale Ein- und 12 Ausgänge
	1	X-OPT-812/P		
	2	X-OPT-812/H		
28	0	X-OPT-416/L	801ch	4 opto-entkoppelte digitale Ein- und 16 Ausgänge
	1	X-OPT-416/P		
	2	X-OPT-416/H		
29	0	X-OPT-020	801dh	20 opto-entkoppelte digitale Ausgänge
	2	X-OPT-020/x		
32	0	X-AD16i-4	8020h	4 galvanisch getrennte analoge Eingänge, 16 Bit

Typ	Sub-typ	Modul	Prog.-Nr. des MDD	Beschreibung
36	1, 11	X-AD14-20/F	8024h	20 analoge Eingänge, 14 Bit, 2,2 Msps
	2, 12	X-AD14-20/M		20 analoge Eingänge, 14 Bit, 800 Ksps
	3	X-AD14-20/S		20 analoge Eingänge, 14 Bit, 400 Ksps
	7	X-AD12-16/L		16 analoge Eingänge, 12 Bit, 400 Ksps
	8	X-AD12-16/5		16 analoge Eingänge, 12 Bit, 400 Ksps
	9	X-AD14-20/F/i		20 analoge Eingänge, 14 Bit, 2,2 Msps, 0..20 mA
40	1	X-DIO-40/i	8028h	38 digital I/O, 3 Timer, LED, Interrupt-fähig
	0	X-DIO-40		38 digital I/O, LED
40	2	X-DIO-32	8028h	32 digital I/O
42	0	X-5Bx64-8	802ah	Anschluss-Modul für acht 5Bx64-Panels
43	0	X-CPLD-38	802bh	Frei programmierbares Digital-I/O-Modul
44	0	X-DAD-4/U	802ch	4 analoge Eingänge, 4 analoge Ausgänge, 18 digitale I/Os, Spannungs-Ein- und Ausgänge
45	0	X-DAD-4/i	802dh	4 analoge Eingänge, 4 analoge Ausgänge, 18 digitale I/Os, Strom-Ausgänge
46	1	X-5B-1	802eh	Anschluss-Modul für acht 5Bx02-Panels
48	0	X-DA12-4/Ui	8030h	4 galvanisch getrennte analoge Ausgänge, 12 Bit
50	0	X-DA16i-4/Ui	8032h	4 galv. getrennte Ausgänge, 16 Bit Auflösung
	1	X-DA16i-4/U		4 galv. getrennte Ausgänge, 16 Bit Auflösung
	2	X-DA14i-4/Ui		4 galv. getrennte Ausgänge, 14 Bit Auflösung
	3	X-DA14i-4/U		4 galv. getrennte Ausgänge, 14 Bit Auflösung
	4	X-DA16i-4/F	8132h	4 galv. getrennte Ausgänge, 16 Bit mit FIFO-Mode
54	0	X-C16-3i/L	8036h	3 kaskadierbare 16-Bit-Zähler, 12 Opto-Eingänge, 8 Opto-Ausgänge
	1	X-C16-3i/P		
	2	X-C16-3i/T		
58	1	X-CAN-2i/H	803ah	CAN-Bus-Interface mit 2 High-Speed Kanälen, einzeln isoliert
	2	X-CAN-2i/F		CAN-Bus-Interface, 2 fehlertolerante Kanäle, einzeln isoliert
	3	X-CAN-2i/M		CAN-Bus-Interface, 1x High-Speed, 1x fehlertolerant, einzeln isol.
64	1	X-IDE-1	8040h	IDE-Schnittstelle
68	0	X-SSI-2	8044h	2 SSI-Schnittstellen, Host
	1	X-SSI-M		SSI-Monitor, 1 Kanal
	2	X-SSI-4		4 SSI-Schnittstellen, Host
	3	X-SSI-8		8 SSI-Schnittstellen, Host
	4	X-SSI-2/M		SSI-Monitor, 2 Kanäle
74	0	X-SH12-8	804ah	12-Bit Analog-Eingangsmodul mit Speicher
80	1	X-ETH-4C	8050h	Ethernet und 4 asynchrone, serielle Schnittstellen
81	1	X-ETH-10	8051h	Ethernet-Schnittstelle, 10 MBit
82	1	X-COM-4	8052h	4 asynchrone serielle Schnittstellen
84	1	X-SCC-2/U	8054h	2-Kanal sync., async. ESCC, RS-232, -422, -485
	2	X-SCC-2/R		2-Kanal sync., async. ESCC, RS-232
	3	X-SCC-2i/C		2-Kanal sync., async. ESCC, RS-232, 20 mA

Typ	Sub-typ	Modul	Prog.-Nr. des MDD	Beschreibung
88	0, 1	X-COM-8i	8058h	8 asynchrone serielle Schnittstellen, einzeln isoliert
90	0	X-DPM-1i	805ah	PROFIBUS-DP-Master
91	0 1	X-DPS-1i X-DPS-2i	805bh	1 isol. PROFIBUS-DP-Slave, 12 Mbit/s 2 PROFIBUS-DP-Slaves, 12 Mbit/s, einzeln isoliert
96	0	X-ISDN-1	8060h	ISDN-Modem
98	0 3 4	X-56K-FU X-33K-FU X-14K-FU		Analog-Modem, 56 kbit/s, Codec. Funkuhr, UART Analog-Modem, 33,6 kbit/s, Funkuhr, UART Analog-Modem, 14,4 kbit/s
100	1	X-ETH-100	8064h	Ethernet
104	0	X-IEC-1	8068h	IEC-Bus
112	0	X-MiX-26	8070h	4 simultan. sampled analog In, 4 analog Out, 14 Bit, 18 digital I/O
128	0	X-TEST-1	8080h	Test-Modul für MAX-Module
129	0	X-TEST-D	8081h	Test-Modul für digital I/O
132	1 2	X-LCD-S1 X-LCD-H1		LCD-Modul für Sharp-TFT Display LQ057 LCD-Modul für Hitachi-LCD Display SX16
238	1	MAX5dip	8500h	Zentraler Chip auf MAX5dip
240	1, 3, 4	MAX8dip	85F0h	Zentraler Chip auf MAX8dip
242	1	BASiS-6		Zentraler Chip auf BASiS-6

Notizen

B. Fehlermeldungen

Beim Einsatz von MAX-PCs und Trägerkarten (z.B. MAX6pci) werden Fehler von den Bibliotheken, vom Windows-Treiber, von den MDDs oder vom Betriebssystem auf einem angesprochenen CPU-Modul erkannt und als Funktionsrückgabewert an das aufrufende Programm zurückgeliefert. Als weitere Möglichkeit können auch Funktionen in OsX-Anwenderprogrammen, die mit **max_call_func** aufgerufen werden können, Fehler zurückliefern. Dabei ist zu beachten, dass der Wert des Low-Bytes =E0h sein muss, wenn die Funktion einen Fehler signalisieren soll.

Die vom Betriebssystem OsX gemeldeten Fehler haben im Low-Byte Werte von C0h bis FFh. Im High-Byte steht in der Regel eine nähere Erklärung zum Fehler.

In der folgenden Tabelle sind die zur Zeit definierten Fehlercodes aufgelistet. Die Konstanten finden sich in den Dateien X_ERROR.H, X_ERROR.PAS und MAX_ERR.BAS. Alle nicht erwähnten Fehlercodes sind reserviert. Eine kurze Beschreibung kann zu jedem Fehlercode mit der Funktion **max_get_error_message** abgerufen werden.

Wert (hex)	**Konstante**	**Beschreibung**
0000	ERR_OK	Kein Fehler
0101	ERR_TBF_TIMEOUT	Gegenstelle hat das gesendete Wort nicht innerhalb der Timeout-Zeit abgeholt
0201	ERR_RBE_TIMEOUT	Antwort der Gegenstelle kam nicht innerhalb der Timeout-Zeit
0401	ERR_MACRO_PROCESSING	Fehler beim Abarbeiten einer Befehlssequenz
0102	ERR_MEMORY_ ALLOCATION	Es konnte kein Speicher für bibliotheks-interne Variable allokiert werden.
0202	ERR_BOARD_NOT_ AVAILABLE	Die angegebene Trägerkarte ist nicht verfügbar. Sie ist nicht konfiguriert.
0302	ERR_BOARD_NOT_ REACTING	Die angegebene Trägerkarte ist nicht ansprechbar. Dieser Zustand kann nur durch **max_reset_board** verlassen werden.
0402	ERR_WRONG_CARDTYPE	Falscher Kartentyp-Parameter

Wert (hex)	Konstante	Beschreibung
0502	ERR_MODULE_NOT_ AVAILABLE	Auf dem angegebenen Steckplatz ist kein Modul vorhanden.
0602	ERR_MINI_OS_NOT_ACTIVE	Das nach einem Reset normalerweise automatisch aktivierte Mini-OsX ist nicht aktiv, **max_reset_board** sollte aufgerufen werden.
0702	ERR_ROM_OS_NOT_ACTIVE	Das OsX ist nicht aktiv, **max_reset_-board** sollte aufgerufen werden.
0802	ERR_OSX_NOT_READ	Fehler beim Lesen der OsX-Betriebs-systemdatei
0902	ERR_OSX_NOT_FOUND	OsX-Betriebssystemdatei nicht gefunden
0A02	ERR_MORE_THAN_ONE_CPU	Bei Verwendung der Konstanten MAX_AUTO_SLOT in **max_connect_ cpu** darf sich nur ein CPU-Modul auf der Trägerkarte befinden.
0B02	ERR_SW_RESET_NOT_ POSSIBLE	Das Zurücksetzen des Systems ist nicht möglich. Das ist z.B. bei Remote-Ankopplungen der Fall, wenn die Kommunikation nicht mehr funktioniert. In diesem Fall muss ein Hardware-Reset des Systems erfolgen.
0203	ERR_NO_DEBUG_INFO	Das Programm kann nicht debugged werden. Die dafür benötigten Informationen konnten nicht auf den MAX-PC geladen werden. Das Programm ist mit Debug-Informationen neu zu kompilieren.
0303	ERR_INT_NOT_IN_USE	Der angegebene Interrupt wird von keiner Task benutzt.
0403	ERR_FLASH_STATUS	Fehler beim Zugriff auf das Flash-ROM
0D03	ERR_MDD_NOT_STARTED	MDD-Task noch nicht gestartet
0F03	ERR_BUFFER_EMPTY	Aus dem Puffer konnten keine Daten gelesen werden, da der Puffer leer ist.
1003	ERR_BUFFER_FULL	In den Puffer konnten keine Daten

Wert (hex)	Konstante	Beschreibung
		geschrieben werden, da der Puffer voll ist.
1203	ERR_PAR_SEMAPHORE_ LOCKED	Die Semaphore für den Parameterbereich der Task ist belegt.
1303	ERR_DATA_SEMAPHORE_ LOCKED	Die Semaphore für den Datenbereich der Task ist belegt.
1403	ERR_TYPE_CHECK	Ungültiges Betriebssystem ist aktiv
1503	ERR_PROG_NOT_INSTALLED	Das Programm ist nicht als Task auf dem CPU-Modul installiert
1603	ERR_INVALID_ERROR	Ungültiger Fehlercode
1703	ERR_MSG_MDD_NOT_ FOUND	Die benötigte Treiberdatei ‚mx_msg.exe' konnte nicht in den MDD-Suchpfaden gefunden werden.
1803	ERR_READING_MSG_MDD	Fehler beim Lesen der Daten vom Message-MDD.
1903	ERR_MDD_WITH_WRONG_ VERSION	Es wurde ein MDD für das Modul gefunden. Dieser passt jedoch nicht mit der Revision des Moduls zusammen.
1A03	ERR_MDD_WITH_WRONG_ VARIANT	Es wurde ein MDD für das Modul gefunden. Dieser ist aber für die Variante des Moduls nicht geeignet.
1B03	ERR_MODULE_TYPE_ MISMATCH	Es wurde ein MDD mit der angegebenen Programmnummer gefunden. Dieser passt jedoch nicht zum angegebenen Modul.
1C03	ERR_PROG_NR_MISMATCH	Es wurde kein MDD mit der angegebenen Programmnummer gefunden.
xx04	ERR_PARAMETER_RANGE	Ein Wert im Parameterbereich einer Task hat einen nicht zulässigen Wert. Der Offset wird im High-Byte angegeben (xx=00 .. FF).
xx05	ERR_EEPROM_WORD	Ein Wert im EEPROM hat einen nicht zulässigen Wert. Der Offset wird im

Wert (hex)	Konstante	Beschreibung
		High-Byte angegeben (xx=00 .. FF).
xx06	ERR_WRONG_PARAMETER	Ein Übergabeparameter hat einen unzulässigen Wert. Die Nummer des Parameters wird im High-Byte angegeben (xx=00 .. FF).
0207	ERR_FILE_NOT_FOUND	Die angegebene Datei konnte nicht gefunden werden.
0307	ERR_PROG_FORMAT	Das Programm hat ein falsches Format.
0407	ERR_LINKER_SIGNATURE	Das Programm hat ein falsches Format.
0507	ERR_READ_FILE	Fehler beim Lesen einer Datei
0607	ERR_MACRO_NOT_ AVAILABLE	Der angegebene Makrobefehl ist nicht implementiert.
0707	ERR_PROG_NOT_FOUND	Das Programm konnte nicht gefunden werden.
0E07	ERR_ILLEGAL_COMMAND	Befehl nicht erlaubt.
0F07	ERR_CALLBACK_NOT_ SUPPORTED	Das Device, zu dem ein Kanal geöffnet werden soll, unterstützt keine Callback-Funktionalität.
1007	ERR_AXMODULE_DLL_ VERSION	Die benötigte DLL A_XMODULE ist zu alt. Die Datei befindet sich im SNW32-Verzeichnis.
1107	ERR_AXMODULE_DLL_ NOT_FOUND	Die benötigte A_XMODULE.DLL (aus SNW32) konnte nicht gefunden werden.
1207	ERR_INS_FILE_SYNTAX	Syntax-Fehler in der INS-Datei. Zur genaueren Analyse sollte der SNW32-INS-File-Editor verwendet werden.
1307	ERR_AMLX_DLL_VERSION	Die benötigte DLL A_MLX ist zu alt. Die Datei befindet sich im SNW32-Verzeichnis.
1407	ERR_AMLX_DLL_ NOT_FOUND	Die benötigte A_MLX.DLL (aus SNW32) konnte nicht gefunden werden.
0408	ERR_HOLD_REQUEST_ FAILED	Fehler beim Aufruf des Treiber-Dienstes zum Abholen eines SRQs

Wert (hex)	**Konstante**	**Beschreibung**
0608	ERR_DRV_SERVICE	Fehler beim Aufruf des Treiber-Dienstes
0708	ERR_DRIVER_NOT_OPENED	Der Treiber ist nicht korrekt installiert.
0808	ERR_CREATING_SRQ_ THREAD	Der Thread für den Empfang von SRQs von CPU-Modulen konnte auf dem PC nicht gestartet werden.
0908	ERR_SRQ_THREAD_ PRIORITY	Die Priorität des SRQ-Threads konnte nicht auf den höchsten Level gesetzt werden.
0A08	ERR_DRIVER_GENERAL	Allgemeiner (PC-)Treiber-Fehler
0B08	ERR_PROCESSMEM_NOT_ ALLOCATED	(PC-)Treiber konnte benötigten Speicher nicht allokieren.
0D08	ERR_MDD_NOT_LOADED	Der benötigte MDD konnte nicht installiert werden. In den durchsuchten Verzeichnissen (s. **max_load_mdd**) konnte kein passender Treiber gefunden werden. Das kann folgende Ursachen haben: • kein Treiber für das Modul gefunden (Programmnummer stimmt nicht) • Treiber mit passender Nummer gefunden, aber dieser passt nicht zum angegebenen Modul (Modul-Typ, Version und Variante werden zur Kompatibilitätsprüfung ausgewertet)
0E08	ERR_INSTALL_PARAMETER	Fehlerhafte Installationsparameter in der Windows-Registry
1108	ERR_WRONG_PARAMETER_ READ	Fehlerhafte Parameter beim Lesen von Daten vom Treiber
1208	ERR_WRONG_PARAMETER_ WRITE	Fehlerhafte Parameter beim Schreiben von Daten zum Treiber
1308	ERR_CANNOT_WRITE_PARA METER	Parameter kann nicht beschrieben werden
1408	ERR_WRONG_PARAMETER_ TYPE	Unbekannter Parametertyp

Wert (hex)	Konstante	Beschreibung
1508	ERR_PROCESS_ID_NOT_ VALID	Unbekannte Prozess-ID
1608	ERR_LOGBUFFER_NOT_ READABLE	Falsche Funktion beim Lesen des Logbuffers
1908	ERR_EXCLUSIVE_LOCK_ DENIED	Exklusiver Kartenzugriff verwehrt
1A08	ERR_MAX_ACCESS_DENIED	Zugriff verweigert, Karte wird von anderem Prozess exklusiv genutzt
1C08	ERR_ACCESS_VIOLATION	Zugriff auf unzulässigen Bereich
1D08	ERR_SRP_CANCELLED	Service-Anforderung wurde vom Treiber abgebrochen
1E08	ERR_INVALID_LIST_TYPE	Ungültiger Listentyp
1F08	ERR_CORRECTION_NOT_ AVAILABLE	Für den Kanal sind keine Korrekturwerte verfügbar.
010D	ERR_XBUS_NOT_READY	X-Bus nicht kommunikationsbereit
020D	ERR_XBUS_SEMAPHORE_ TIMEOUT	Semaphoren für die Makrokommunikation nicht innerhalb der Timeout-Zeit erhalten
030D	ERR_XBUS_HARDWARE_ RESET	Es ist ein Hardware-Reset auf dem X-Bus aufgetreten
040D	ERR_XBUS_ACCESS_ TIMEOUT	X-Bus-Zyklus wurde mit Timeout abgeschlossen
050D	ERR_XBUS_XRES_TIMEOUT	Dauer-Reset auf dem X-Bus
060D	ERR_XBUS_XREQ_TIMEOUT	Fehlerhafter Zustand auf dem X-Bus
070D	ERR_XBUS_NO_CONFIG_ MASTER	Fehlerhafte X-Bus-Konfiguration: Kein Konfigurationsmaster entdeckt
080D	ERR_TOO_MANY_MASTERS	Max. zulässige Anzahl von Mastern auf dem X-Bus überschritten
0A0D	ERR_XBUS_CFG_MASTER_ FAIL	Fehler bei der Kommunikation mit dem Konfigurationsmaster
0B0D	ERR_EEP_SEMA_BUSY	Fehler beim Zugriff auf das Modul EEPROM: Semaphore ist nicht frei.

Wert (hex)	Konstante	Beschreibung
0C0D	ERR_EEP_BYTE_ACCESS	Es können Blöcke mit geradzahliger Länge ins EEPROM geschrieben bzw. gelesen werden.
0D0D	ERR_EEP_ACCESS	Fehler beim EEPROM-Zugriff
xxFF		CPU-Hardware-Defekt, Details im High-Byte
xxFE		Hardware-Defekt im CPU-Teil, Details im High-Byte
xxFD		Hardware-Defekt auf Modul, Details im High-Byte
xxFB		Systemabsturz, Hinweis auf Ursache im High-Byte
xxFA		Unerwarteter Trap, Fault, Int, nicht korrigierbar, Vektor-Nr. im High-Byte
xxF9		Fehler aus Prepare
xxF3 xxF2 xxF1 xxF0		Systemabsturz, verursachende Task steht im High-Byte, Low-Byte=F0h: Task 00 .. FFh, Low-Byte=F1h: Task 100h .. 1FFh, usw.
xxEF xxEE xxED xxEC		Taskabsturz, Tasknummer steht im High-Byte, Low-Byte=ECh: Task 00 .. FFh, Low-Byte=EDh: Task 100h .. 1FFh, usw.
xxEB xxEA xxE9 xxE8		Task deaktiviert, Tasknummer steht im High-Byte, Low-Byte=E8h: Task 00 .. FFh, Low-Byte=E9h: Task 100h .. 1FFh, usw.
xxE7 xxE6 xxE5 xxE4		Befehl wurde abgebrochen wegen Sende-Timeout (E7h), Empfangs-Timeout (E6h), unbekanntem Befehlscode (E5h), Fehler in Ausführung (E4h), im High-Byte steht der Makrobefehlscode
xxE3		Befehl wurde abgebrochen, im High-Byte steht der Typ des Fehlers

Wert (hex)	Konstante	Beschreibung
xxE2		Fehler bei Host-Kommunikation
xxE1		Fehler bei Zugriff auf Device, im High-Byte steht der Typ des Fehlers
xxE0		Fehler bei Aufruf einer Subroutine, im High-Byte steht der Typ des Fehlers
02E0	ERR_INVALID_PARAMETER	Fehlerhafte Parameter bei der Kommunikation zwischen Bibliothek und Treiber
08E0	ERR_TASK_NOT_INSTALLED	Die Task ist nicht installiert.
09E0	ERR_NOT_IMPLEMENTED	Die Funktion bzw. das Feature ist nicht implementiert.
0BE0	ERR_PLAUSIBILITY	Unerwartete Antwort auf einen Makrobefehl
0EE0	ERR_NOT_ENOUGH_ MEMORY	Der gewünschte Speicher konnte nicht reserviert werden. Mit **max_get_ ram_info** kann der zur Verfügung stehende Speicherplatz geprüft werden.
14E0	ERR_INVALID_HANDLE	Das übergebene Handle ist ungültig.
15E0	ERR_DEVICE_NOT_ AVAILABLE	Das Device steht nicht zur Verfügung.
16E0	ERR_WRONG_NUMBER_OF_ INPUTS	Falsche Anzahl an Übergabeparametern (hin)
17E0	ERR_WRONG_NUMBER_OF_ RETURNS	Falsche Anzahl an Übergabeparametern (zurück)
18E0	ERR_FUNCTION_NOT_ AVAILABLE	Die angegebene Funktion ist in der Task nicht vorhanden.
19E0	ERR_DEBUG_NOT_POSSIBLE	Debugging nicht möglich. Programm enthält keine Debug-Information.
20E0	ERR_DEVICE_ALREADY_IN_ USE	Der MDD-Kanal konnte nicht geöffnet werden, weil zum angegebenen Device bereits ein anderer Kanal exklusiv geöffnet ist.
84E0	ERR_FLASH_WRONG_STATE	Das Flash ist im falschen Zustand.

Wert (hex)	Konstante	Beschreibung
85E0	ERR_FLASH_NOT_ SUPPORTED	Der Flash-Typ wird nicht unterstützt.
86E0	ERR_FLASH_WRONG_ SERVICE	Der aufgerufene Dienst für den Zugriff auf das Flash ist nicht erlaubt.
87E0	ERR_FLASH_NOT_ AVAILABLE	Das Flash ist nicht vorhanden.
88E0	ERR_FLASH_WRONG_SLOT	Die Slot-Nummer, die Layer-Nummer oder die IC-Nr. ist falsch.
89E0	ERR_FLASH_ADDR_NOT_ ALIGNED	Die angegebene Adresse hat nicht das richtige Alignment.
8AE0	ERR_FLASH_WRONG_ADDR	Ungültige Adresse beim Zugriff auf das Flash.
8BE0	ERR_FLASH_SECTOR_NOT_ ERASED	Der Sektor, in den geschrieben werden soll, muss zunächst gelöscht werden.
8CE0	ERR_FLASH_TIMEOUT	Timeout-Fehler beim Schreiben bzw. Löschen des Flash.
94E0	ERR_CPS_NOT_SUPPORTED	Die übergebene CPS ist falsch.
95E0	ERR_INFOTEXT_NOT_ AVAILABLE	Infotext für den Kanal nicht verfügbar.
96E0	ERR_SERVICE_NOT_ AVAILABLE	MDD-Dienst nicht verfügbar
97E0	ERR_NO_CAPTION_BUFFER	Der Kanal besitzt keinen Puffer für eine Kurzbeschreibung. Der Puffer wird beim Öffnen des Kanals mit der im MDD-Parameter 1 angegebenen Größe angelegt.
98E0	ERR_WRONG_DATA_TYPE	Der MDD-Dienst verwendet den falschen Datentyp.
99E0	ERR_RANGE_CROSS_OVER	Bereichsüberschreitung
9AE0	ERR_RANGE_CROSS_UNDER	Bereichsunterschreitung
9BE0		Interner Hardwarefehler bei Zugriff
9CE0	ERR_FUNCTION_LOCKED	Funktion ist im MDD gesperrt

Wert (hex)	Konstante	Beschreibung
9DE0	ERR_EEPROM_REGION_ LENGTH	Die angegebene Anzahl an Bytes ist kleiner als die in der angegebenen Region enthaltenen Bytes.
9EE0	ERR_EEPROM_ACCESS	Fehler beim Zugriff auf EEPROM
9FE0	ERR_EEPROM_HEADER	Die EEPROM-Kennung (Wort 0) ist ungültig.
A0E0	ERR_EEPROM_REGION_ HEADER	Der Header der angegebenen Region ist ungültig.
A1E0	ERR_ARITHMETICS	Fehler in der Berechnung.
A2E0	ERR_XBUS_INVALID_ CONFIG	Fehler in der X-Bus-Konfiguration (Modul-Basisadresse ist bereits konfiguriert).
A3E0	ERR_SETTLE_TIMER	Settle-Time ist abgelaufen, ohne dass ein Wert gelesen werden konnte.
A4E0	ERR_SEMAPHORE_BUSY	Semaphore eines Moduls nicht erhalten
A5E0	ERR_EEPROM_REGION_NOT _CREATED	Der angegebene Bereich ist noch nicht angelegt (zuerst max_create_eeprom_ area aufrufen).
A6E0	ERR_EEPROM_REGION_ EXISTS	Der angegebene Bereich existiert bereits.
xxDF		Unerwarteter Trap, Fault, Int, korrigiert, Vektor-Nr. im High-Byte
xxDE		Fehler von MDD, High-Byte = Parameter-Offset des fehlerhaften Parameters
xxDB xxDA xxD9 xxD8		Warnung von Task. Die Tasknummer steht im High-Byte, Low-Byte=D8h: Task 00 .. FFh, Low-Byte=D9h: Task 100h .. 1FFh, usw.
xxD7		Warnung vom Betriebssystem, im High-Byte steht der Typ des Fehlers
xxD4		Warnung: nicht gesendet, im High-Byte steht der Grund als Bitmap

Wert (hex)	**Konstante**	**Beschreibung**
xxD3		Befehl komplett abgewickelt, aber nicht ausgeführt, im High-Byte steht der Makrobefehlscode
xxD2		Befehl komplett abgewickelt, aber nicht ausgeführt, im High-Byte steht der Typ des Fehlers
xxD1		Warnung von Befehl, ausgeführt, im High-Byte steht der Makrobefehlscode
xxD0		Warnung von Befehl, ausgeführt, im High-Byte steht der Typ des Fehlers
00CF	ERR_SRQ_BOARD_RESET	Die Karte wurde zurückgesetzt. Das High-Byte hat keine Bedeutung.
01CF	DRVMSG_SRQBUF_ WARNING	Warnung vom Treiber: Der SRQ-Buffer ist zu mehr als 2/3 gefüllt.
02CF	DRVMSG_SRQBUF_ERROR	Der SRQ-Buffer ist übergelaufen.
XxCE xxCC xxCB xxCA		Reservierte System SRQs

Wert (hex)	Konstante	Beschreibung
xxC3 xxC2 xxC1 xxC0		SRQ von Task. Die Tasknummer steht im High-Byte, Low-Byte=C0h: Task 00 .. FFh, Low-Byte=C1h: Task 100h .. 1FFh, usw.

Der Fehlertyp und Erklärungen zu Warnungen wird im **High-Byte** einer Fehlermeldung (SRQ) und als Fehlermeldung bei bestimmten Makrobefehlen geliefert.

Nummer Dez	Hex	Bedeutung
1	01h	Timeout
2	02h	Falscher Parameter bei Makrobefehl
3	03h	Reserviert
4	04h	Makrobefehl nicht implementiert
5	05h	ROM-Programm nicht vorhanden
6	06h	Protected Mode nicht erlaubt
7	07h	Programmnummer bei Installierung und PDT verschieden
8	08h	Task bzw. Programm nicht installiert
9	09h	Feature (noch) nicht implementiert
10	0ah	NI-Task-Tabelle voll
11	0bh	Systemfehler (unlogisch)
12	0ch	Task ist nicht aktiv
13	0dh	Falsches Format
14	0eh	Kein Platz im RAM
15	0fh	Subroutine nicht implementiert
16	10h	Falscher Parameter bei Aufruf einer Subroutine
17	11h	Timeout aufgetreten
18	12h	Device lässt sich nicht beschreiben
19	13h	Nicht genug Speicher
20	14h	Falsche Kennung
21	15h	Device nicht vorhanden
22	16h	Anzahl Parameter Hin falsch (bei Funktionsaufruf)
23	17h	Anzahl Parameter Rück falsch (bei Funktionsaufruf)
24	18h	Funktionsnummer ungültig oder Task nicht installiert
25	19h	Debugging mit RTDS nicht möglich, keine DDT
26	1ah	Unbekannter Fehler bzw. falsche Fehlergruppe von Funktion oder Prozedur gemeldet
27	1bh	Falscher Modul-Steckplatz

Nummer		**Bedeutung**
Dez	**Hex**	
28	1ch	EEPROM-Inhalt falsch
29	1dh	Erster zu schreibender Parameter/Data außerhalb Bereich
30	1eh	Anzahl zu schreibender Parameter/Daten zu groß
31	1fh	Es läuft gerade ein PC-Makrobefehl
32	20h	Device wird vom Betriebssystem benutzt
33	21h	Unerlaubte Zeitangabe
34	22h	Puffer wird gerade beschrieben
35	23h	Puffer wird gerade gelesen
36	24h	Nicht genügend Platz im Puffer
37	25h	Nicht genügend Zeichen im Puffer
38	26h	Puffernummer ungültig
39	27h	Tasktyp ungültig
40	28h	Cache ungültig (Puffer)
41-47	29h-2fh	Reserviert
48	30h	RAM (0 bis 64 KByte) defekt
49	31h	Interruptnummer falsch
50	32h	Reserviert
51	33h	Tasknummer schon benutzt
52	34h	TI-Task Timeout
53	35h	Linker-Signatur falsch
54	36h	Device noch nicht initialisiert
55-63	37h-3fh	Reserviert
64	40h	Fehler aus MDD: CDT-Kennung falsch
65-69	41h-45h	Reserviert
70	46h	Fehler aus MDD mit falscher Fehlergruppe (≠e0h)
71-95	47h-5fh	Reserviert
96-127	**60h-7fh**	**Benutzerspezifische Fehler**
128	80h	Kein Modul auf Steckplatz vorhanden
129	81h	Modul benötigt keine Initialisierung
130	82h	Modul wird nicht initialisiert wegen EEPROM-Info in Wort 1
131	83h	Modul wird nicht initialisiert, weil Subroutine fehlt
132	84h	Falscher Zustand
133	85h	Device-Typ wird nicht unterstützt
134	86h	Falscher Dienst (Flash-EPROM)
135	87h	Kein Flash lt. EEPROM (Flash-EPROM)

Nummer Dez	Hex	Bedeutung
136	88h	Falsche SLN (Slot^Layer-Nummer) oder IC-Nr. (Flash-EPROM)
137	89h	Adresse nicht aligned oder nicht auf Sektorgrenze (Flash-EPROM)
138	8ah	Adresse falsch (Flash-EPROM)
139	8bh	Flash bzw. Sektor nicht gelöscht (Flash-EPROM)
140	8ch	Fehler (Timeout) beim Programmieren/Löschen (Flash-EPROM)
141-143	8dh-8fh	Reserviert
144	90h	Host schon/noch installiert
145	91h	Host nicht installiert
146	92h	Fehler im Konfigurations-Bereich
147	93h	Reserviert
148	94h	Die übergebene CPS ist falsch.
149	95h	Infotext für den Kanal nicht verfügbar.
150	96h	MDD-Dienst nicht verfügbar
151	97h	Der Kanal besitzt keinen Puffer für eine Kurzbeschreibung. Der Puffer wird beim Öffnen des Kanals mit der im MDD-Parameter 1 angegebenen Größe angelegt.
152	98h	Der MDD-Dienst verwendet den falschen Datentyp.
153	99h	Bereichsüberschreitung
154	9ah	Bereichsunterschreitung
155	9bh	Interner Hardwarefehler bei Zugriff
156	9ch	Funktion ist im MDD gesperrt
157	9dh	Die angegebene Anzahl an Bytes ist kleiner als die in der angegebenen Region enthaltenen Bytes.
158	9eh	Fehler beim Zugriff auf EEPROM
159	9fh	Die EEPROM-Kennung (Wort 0) ist ungültig.
160	a0h	Der Header der angegebenen Region ist ungültig.
161	a1h	Fehler in der Berechnung.
162	a2h	Fehler in der X-Bus-Konfiguration (Modul-Basisadresse ist bereits konfiguriert).

Nummer		**Bedeutung**
Dez	**Hex**	
163	a3h	Settle-Time ist abgelaufen, ohne dass ein Wert gelesen werden konnte.
164	a4h	Semaphore eines Moduls nicht erhalten
165	a5h	I/O-Bereich zu klein (X-Bus Adressverteilung)
166-171	a6h-abh	Reserviert
172	ach	Ergebnis (Daten) ungültig
173-223	adh-dfh	Reserviert
224-255	**e0h-ffh**	**Benutzerspezifische Warnungen**

Notizen

C. Taskinformationen

Die im folgenden beschriebenen Informationen können mit der Funktion **max_task_info** abgerufen werden.

Alle System-Calls beziehen sich auf die beim Aufruf angegebene Task bzw. auf das darunter installierte Programm. Der Rückgabewert ist immer 4 Byte lang, die einzelnen Bytes werden mit aE, aF, aG und aH bezeichnet. Wenn mehrere Bytes zusammengefasst werden, ist aE das niederwertigste und aH das höchstwertige. Wenn kein Programm unter der Task installiert ist, wird immer 0 zurückgeliefert. Zurückgelieferte Adressen sind immer 32 Bit absolute physikalische Adressen, sofern nichts anderes vermerkt ist. Wenn nur Teile der Antwort angegeben sind, dann sind auch nur diese Teile gültig.

Die Funktions- bzw. Prozeduradressen können nur mit den oben angeführten Funktionen für die Funktionen bzw. Prozeduren 0 bis 31 ermittelt werden. Dabei muß für "x" die Länge des Vorspanns der Programm-Deskriptor-Tabelle (PDT) eingesetzt werden, die ebenfalls per System-Call ermittelt werden kann.

PDT-Informationen

callnr (lo)	callnr (hi)	Anforderung von
0	0	Typ PDT (aE), Länge (x) des Vorspanns der PDT (aF), Anzahl der Prozeduren (aG, aH)
4	0	Programm-Nr. (aE, aF), Version (aG) und Revision (aH) des installierten Programms: aG = Version : 30h (="0") bis 39h (="9") aH = Revision : 41h (="A") bis 5Ah (="Z")
8	0	Prozessor-Typ (aE), Co-Prozessor-Typ (aF), Programmiersprache (aG) und Programmtyp (aH)
12	0	Flag (aE, aF) (s. Anhang D), Interrupt-Nr. (aG), (aH = 0)
16	0	Anfangsadresse Datenbereich (physikalisch) (Entspricht dem Wert, der in der PDT eingetragen wurde. Die tatsächliche Anfangsadresse des Datenbereichs einer Task finden Sie in der TDT einer Task (siehe unten)!)
20	0	Größe Datenbereich (Anzahl Bytes)
32	0	Anfangsadresse des Parameterbereichs (physikalisch), nur gültig, wenn vom Programm selbst reserviert. Der Parameterbereich beginnt immer direkt "nach" der TDT.
36	0	Größe des Parameterbereichs (aE, aF) (in Anzahl Bytes)
38	0	Anfangsadresse Hypertextbereich (physikalisch)
x+0	0	Adresse der Main-Prozedur (Proz. 0) (Segment:Offset) (x=48 bei PDT Typ 1)
x+4	0	Adresse der Auto-Init-Prozedur (Proz. 1) (Segment:Offset) (x=48 bei PDT Typ 1)
x+8	0	Adresse der 1. Anwenderprozedur (Proz. 2) (Segment:Offset) (x=48 bei PDT Typ 1)

TDT-Informationen

callnr (lo)	callnr (hi)	Anforderung von
0	1	Typ TDT (aE), Länge TDT (aF), Task-Nr. (aG, aH)
4	1	Flag (aE, aF), Interrupt-Nummer (II-Task) bzw. Anzahl der Aktivierungen (NI-Task) (aG) (aH = 0)
8	1	Anzahl Parameter (aE, aF) (in Anzahl Bytes), Anzahl Prozeduren (aG, aH)
12	1	Adresse PDT (physikalisch)
16	1	Adresse Hypertext (physikalisch)
20	1	Anfangsadresse Datenbereich (physikalisch)
24	1	Endadresse Datenbereich (physikalisch)

TDT-Adresse

callnr (lo)	callnr (hi)	Anforderung von
0	2	Anfangsadresse der TDT

Notizen

D. OsX-Programm-Deskriptor-Tabelle (PDT)

Die einzelnen Einträge der Tabelle werden im folgenden ausführlich erläutert. Die Spalte 'rel. Adr.' steht für die Position innerhalb der PDT, an der der beschriebene Eintrag beginnt (in Byte). Die Angaben gelten für PDT-Typ = 1.

rel. Adr.	Typ	Erklärung
0	UCHAR	Typ der PDT (z.Zt. = 1)
1	UCHAR	Länge des Vorspanns der PDT (bei Typ 1 immer = 48)
2	USHORT	Anzahl der Prozeduren (z.B. = 3)
4	USHORT	Programmnummer
6	CHAR	Programmversion (z.B. '1')
7	CHAR	Programmrevision (z.B. 'A')
8	UCHAR	Prozessor-Typ (4 = 486)
9	UCHAR	Coprozessor-Typ
10	UCHAR	Programmiersprache des Programms
11	UCHAR	Programmtyp
12	USHORT	Flags (16 Bit)
14	USHORT	Interrupt-Nummer, für die das Programm gedacht ist
16	ULONG	Anfangsadresse des Datenbereichs (physikalisch)
20	ULONG	Größe des Datenbereichs in Byte
24	ULONG	Min. Größe Datenbereich
28	ULONG	Max. Größe Datenbereich
32	ULONG	Anfangsadresse des Parameterbereichs (physikalisch)*
36	USHORT	Größe des Parameterbereichs in Byte
38	ULONG	Anfangsadresse des Hypertextbereichs (physikalisch)
42	USHORT	Reserviert
44	ULONG	Anfang DDT
48	ULONG	Adresse der Haupt-Prozedur (Prozedur 0)
52	ULONG	Adresse der Auto-Init-Prozedur (Prozedur 1)
56	ULONG	Adresse der 1. Anwenderprozedur (Prozedur 2)

* wenn vom Programm selbst reserviert

rel. Adr.	Typ	Erklärung
...	...	... (gegebenenfalls weitere Anwenderprozeduren)

Typ der PDT (UCHAR, rel. Adr. 0)

Im Moment wird nur der PDT-Typ 1 unterstützt. Setzen Sie also in Ihren Programmen den Typ der PDT immer gleich 1.

Länge des Vorspanns der PDT (UCHAR, rel. Adr. 1)

Sie ist konstant und zur Zeit bei PDT-Typ 1 = 48 Byte.

Anzahl der Prozeduren (USHORT, rel. Adr. 2)

Sie beträgt mindestens 2 (Haupt-Prozedur und Auto-Init-Prozedur) und maximal 16320. Die Gesamtlänge der PDT in Byte errechnet sich aus der Länge des Vorspanns + (Anzahl der Prozeduren * 4).

Programmnummer (USHORT, rel. Adr. 4)

Sie kann von 1 bis 65534 gewählt werden. Programmnummer 0 ist reserviert für das Betriebssystem. Die Programmnummern ab 32768 sind für SORCUS reserviert und sollten nicht verwendet werden. Die Programmnummer 0ffffh ist ebenfalls reserviert, es handelt sich um ein Dummy-Programm ohne Funktion und wird gemeldet, wenn kein Programm unter einer Task installiert ist.

Programmversion und Programmrevision (2 CHAR, rel. Adr. 6 und 7)

Beide Einträge werden als ASCII-Zeichen angegeben (Version von "1" bis "9", Revision von "A" bis "Z") und dienen nur der Versionsverwaltung des Programmierers. Bei Verwendung der Hypertext-Struktur werden sie auch dort verwendet.

Prozessor-Typ (UCHAR, rel. Adr. 8)

Diese Angabe beschreibt, für welchen Prozessor das Programm geschrieben wurde (**MAX_CPU486**).

Coprozessor-Typ (UCHAR, rel. Adr. 9)

Diese Angabe beschreibt, ob und welchen Coprozessor das Programm erwartet (= 0 setzen).

Falls Sie Fließkommaoperationen verwenden wollen, so benutzen Sie **_487SX.** Falls Sie keine Fließkommaoperationen verwenden, so benutzen Sie **_NOCOPROZ**.

Falls Sie **_487SX** benutzen, sind die Programme sowohl auf einem MAX-PC mit 486 CPU als auch auf einem MAX-PC mit Pentium CPU (Coprozessor wird benutzt) lauffähig. Mit **_487DX** kompilierte Programme setzen voraus, dass eine MAX-PC mit Pentium CPU verwendet wird.

Programmiersprache des Programms (UCHAR, rel. Adr. 10)

Durch die Angabe der Programmiersprache, in der das Programm geschrieben ist, wird das Betriebssystem in die Lage versetzt, bestimmte sprachspezifische Eigenheiten zu berücksichtigen (**MAXRT_LANGUAGE_C, MAXRT_LANGUAGE_PAS, MAXRT_LANGUAGE_ASM**).

Programmtyp (UCHAR, rel. Adr. 11)

Dieser Eintrag definiert die Aufgabe des Programms. (Zur Zeit = 0 setzen)

Flags (USHORT = 16 Bit, rel. Adr. 12)

Die Angaben in Klammern sind die Namen der Konstanten, die für den jeweiligen Eintrag in MAXRT definiert sind.

Bit-Nr.	Bedeutung	
0..2	Tasktyp: Bits	210 000: Nicht-Interrupt-Task (**MAX_NI_TASK**) 001: Indirekte Interrupt-Task (**MAX _II_TASK**) 011: Timer-Initiierte Task (**MAX _TI_TASK**)
3[4]	Tasktyp und Interrupt-Nummer werden durch PDT festgelegt (Bit = 1). (**MAXPDT_TASKTYP_IN_PDT** = 8)	
4	Real-Mode- (Bit = 0) oder Protected-Mode-Programm (Bit = 1)	
5	Code cacheable (Bit = 0) oder nicht (Bit = 1). Reserviert	
6	Reserviert.	
7[4]	Hypertext vorhanden (Bit = 1).	
8[4]	Der Datenbereich wird vom Betriebssystem (Bit = 0) oder vom Programm selbst reserviert (Bit = 1). (**MAXPDT_DATA_IN_PROG** = 256)	
9[4]	Datenbereichsgröße variabel (Bit = 0) oder fest (Bit = 1). (**MAXPDT_FIXED_DATASIZE** = 512)	
10[4]	Der Parameterbereich wird vom Betriebssystem (Bit = 0) oder vom Programm reserviert (Bit = 1). (**MAXPDT_PARAMETER_IN_PROG** = 1024)	
11..15	Reserviert.	

Flag Bit-3: Tasktyp und Interrupt-Nummer liegen fest?

Wenn dieses Bit = 0 gesetzt ist, können Tasktyp und Interrupt-Nummer beim Installieren festgelegt werden. Wenn das Bit = 1 gesetzt wird, werden die Angaben zum Tasktyp und zur Interrupt-Nummer auf jeden Fall aus der PDT entnommen. Natürlich müssen die Angaben von Tasktyp und Interrupt-Nummer in diesem Fall gültige Werte aufweisen.

Flag Bit-8: Wer reserviert Datenbereich und liefert dessen Adresse?

[4] Erklärung siehe folgende Seiten

- Bit-8 = 0
  Standardmäßig wird der Datenbereich vom Betriebssystem reserviert, und damit Anfangs- und Endadresse festgelegt, entsprechend dem zum Zeitpunkt der Installation des Programms zur Verfügung stehenden Speicherplatz. Die Angabe "Anfangsadresse Datenbereich" in der PDT wird dann nicht verwendet. Ob die Angabe "Größe des Datenbereichs" ausgewertet wird, hängt von Bit-9 und von der Installation ab (s.u.). Der Zugriff vom Programm aus auf den eigenen Datenbereich muss dann immer über Bibliotheksaufrufe geschehen. Durch diese Technik ist ein Programm, das nur einmal auf dem MAX-PC vorhanden ist, mehrfach installierbar, weil die Task bei der Installation eines Programms einen eigenen Datenbereich (und Parameterbereich) erhält. Damit das Programm mehrfach installierbar ist, muss bei der Reservierung des Parameterbereichs genauso vorgegangen werden (siehe Flag Bit-10).

- Bit-8 = 1
  In diesem Fall, wird der Datenbereich vom Programm selbst reserviert oder zur Verfügung gestellt. Die Angaben in dieser PDT "Anfangsadresse Datenbereich" und "Größe Datenbereich" sind dann gültig und werden bei der Installation des Programms verwendet. Das hat den Vorteil, dass dann der Datenbereich vom Programm aus direkt zugreifbar ist (ohne Pointer). Allerdings ist das Programm dann nicht mehrfach installierbar.

Flag Bit-9: Größe des Datenbereichs fest oder variabel?

- Bit-9 = 0
  Die Größe des Datenbereichs ist variabel und kann beim Installieren vorgegeben werden. Auch wenn die Größe als variabel deklariert ist, kann beim Installieren angegeben werden, dass die in der PDT vorgegebene Größe verwendet werden soll.

- Bit-9 = 1
  Die Größe des Datenbereichs ist durch den in der PDT vorgegebenen Wert fest vorgegeben, sie kann beim Installieren nicht geändert werden. Der bei der Installation angegebene Wert für die Größe wird verworfen.

Flag Bit-10: Wer reserviert den Parameterbereich und legt dessen Anfangsadresse fest?

- Bit-10 = 0
  Standardmäßig reserviert das Betriebssystem den Platz für den Parameterbereich und legt die Anfangsadresse fest. Die Angabe in der PDT "Anfangsadresse Parameterbereich" ist dann ungültig, die Angabe "Größe Parameterbereich" in der PDT wird immer als Größe ausgewertet (s.u.). Der Zugriff vom Programm aus auf den eigenen Parameterbereich muss dann immer mit den Bibliotheksfunktionen

geschehen. Durch diese Technik ist ein Programm, das nur einmal auf dem MAX-PC vorhanden ist, mehrfach installierbar, weil jede Task bei der Installation eines Programms unter dieser Task einen eigenen Parameterbereich (und Datenbereich) erhält. Genauso muss dann auch bei der Reservierung des Datenbereichs vorgegangen werden (siehe Flag Bit-8).

- Bit-10 = 1
  Der Parameterbereich wird vom Programm selbst reserviert. Die Angabe in dieser PDT "Anfangsadresse Parameterbereich" ist dann gültig und wird bei der Installation des Programms verwendet. Das hat den Vorteil, dass dann der Parameterbereich vom Programm aus direkt zugreifbar ist. Allerdings ist das Programm dann nicht mehrfach installierbar.

Interrupt-Nummer (USHORT, rel. Adr. 14)

Diese Angabe wird bei II-, TI- und TM-Tasks verwendet.

Anfangsadresse des Datenbereichs, physikalisch (ULONG, rel. Adr. 16)

Diese Angabe wird nur für die Installation verwendet und muss nur dann gültig sein, wenn die Lage des Datenbereichs vom Programm vorgegeben wird. Soll der Datenbereich vom Betriebssystem reserviert werden, dann braucht hier nichts eingetragen werden.

Größe des Datenbereichs (ULONG, rel. Adr. 20)

Auch diese Angabe wird nur für die Installation verwendet. Wenn die Reservierung von Speicher und die Lage des Datenbereichs vom Programm selbst vorgegeben wird, dann muss hier die Größe stehen (Anzahl Byte). Wenn die Größe des Datenbereichs fest vorgegeben werden soll, steht hier ebenfalls die Größe des Datenbereichs.

Anfangsadresse des Parameterbereichs, physikalisch (ULONG, rel. Adr. 32)

Diese Angabe wird nur für die Installation verwendet und muss nur gültig sein, wenn die Lage des Parameterbereichs vom Programm vorgegeben wird (siehe oben). Unmittelbar vor dem Parameterbereich liegt immer die TDT (Task-Deskriptor-Tabelle), unabhängig davon, wer den Parameterbereich reserviert und dessen Anfangsadresse festgelegt hat. Es muss die physikalische Adresse des Parameterbereichs angegeben werden.

Größe des Parameterbereichs (USHORT, rel. Adr. 36)

Diese Angabe ist immer zum Installieren erforderlich. Die Größe (in Anzahl Byte) wird fest vorgegeben und kann durch das Installieren nicht verändert werden. Die maximale Größe beträgt 65280. Unmittelbar vor dem Parameterbereich liegt immer die TDT. Wenn der Parameterbereich vom Programm selbst reserviert wird, muss das Programm vor dem Parameterbereich auch den Platz für die TDT reservieren.

Adressen der Prozeduren, Segment:Offset (ULONG, ab rel. Adr. 48)

Die Adressen sind Segment:Offset Adressen, wenn das Programm für Real Mode geschrieben wurde. Die ersten beiden Prozeduren müssen immer vorhanden sein, also die Haupt-Prozedur (Prozedur #0) und die Auto-Init Prozedur (Prozedur #1). Die übrigen sind optional. Wenn mehrere Prozeduren vorhanden sind, müssen sie fortlaufend (ohne Lücken) durchnumeriert sein. Entsprechend der angegebenen Anzahl Prozeduren in dieser PDT müssen auch gültige Adressen vorhanden sein. Der Code einer Prozedur besteht mindestens aus einem "RET FAR".

Die Haupt-Prozedur wird bei NI-Tasks vom Scheduler bzw. bei II-Tasks bei Auftreten des zugehörigen Interrupts aufgerufen. Die Auto-Init-Prozedur wird immer nach Abschluss des Installationsvorgangs des Programms unter einer Task aufgerufen, wenn im Flag beim Installieren Bit-11 = 1 gesetzt ist. In Zukunft ist geplant, Prozedur 2 und 3 auch für Betriebssystemzwecke zu benutzen.

Notizen

E. OsX-Task-Deskriptor-Tabelle (TDT)

rel. Adr.	Typ	Erklärung
0	UCHAR	TDT-Typ (z.Zt. immer = 1)
1	UCHAR	Länge der TDT in Byte (bei Typ 1 immer = 36)
2	USHORT	Task-Nummer
4	USHORT	Flags
6	UCHAR	Interrupt-Nummer (bei II-Tasks) bzw. Zahl der Aktivierungen (bei NI-Tasks)
7	UCHAR	Reserviert (=0)
8	USHORT	Größe des Parameterbereichs (Anzahl Byte)
10	USHORT	Anzahl der Prozeduren
12	ULONG	Adresse der PDT (physikalisch)
16	ULONG	Adresse der Hypertextinformationen (physikalisch)
20	ULONG	Anfangsadresse des Datenbereichs (physikalisch)
24	ULONG	Endadresse+1 des Datenbereichs (physikalisch)
28	ULONG	Reserviert
32	ULONG	Reserviert

Typ der TDT (UCHAR, rel. Adr. 0)

Der Typ der TDT ist eine Konstante, die für zukünftige Entwicklungen vorgesehen ist. Sie ist zur Zeit immer = 1 gesetzt.

Länge der TDT (UCHAR, rel. Adr. 1)

Die Länge der TDT ist bei Typ 1 immer = 36 Byte.

Task-Nummer (USHORT, rel. Adr. 2)

Die Tasknummer ist die Nummer, unter der das Programm installiert wurde.

Flags (USHORT, rel. Adr. 4)

Bit-Nr.	Bedeutung	
0..2	Task-Typ: Bits	210 000: Nicht-Interrupt-Task 001: Indirekte-Interrupt-Task 011: Timer-Initiierte Task 011: System-Task
3, 4	Privilegstufe des Programms 00: höchste Privilegstufe (z.B. Betriebssystem) 11: niedrigste Priorität (Anwendungsprogramme)	
5	Programm läuft im Real-Mode (Bit=0) oder Protected-Mode (Bit=1).	
6	Task ist aktiviert (Bit=1)	
7	Programm installiert (Bit=1)	
8	Angaben zu Hypertext vorhanden (Bit=1) oder nicht vorhanden (Bit=0)	
9	Zugriff auf Datenbereich erlaubt (Bit=0) oder nicht erlaubt (Bit=1) (Semaphore für Datenbereich)	
10	Zugriff auf Parameterbereich erlaubt (Bit=0) oder nicht erlaubt (Bit=1) (Semaphore für Parameterbereich)	
11..15	Reserviert	

Interrupt-Nummer / Anzahl Aktivierungen (USHORT, rel. Adr. 6)

Bei Interrupt-Tasks (II-Tasks) steht hier die Interruptnummer, bei Nicht-Interrupt-Tasks (NI-Tasks) die Anzahl der Aktivierungen.

Anzahl Parameter (USHORT, rel. Adr. 8)

Dieser Eintrag ist eine Kopie aus dem entsprechenden Eintrag in der PDT des Programms.

Anzahl Prozeduren (USHORT, rel. Adr. 10)

Dieser Eintrag ist eine Kopie aus dem entsprechenden Eintrag in der PDT des Programms.

Adresse der PDT, physikalisch (ULONG, rel. Adr. 12)

Die Adresse der PDT ist nur gültig, wenn ein Programm unter der Task installiert wurde (Bit-7 in Flags =1).

Anfangsadresse des Datenbereichs, physikalisch (ULONG, rel. Adr. 20)

Diese Adresse gibt den Anfang des Datenbereichs an. Nach der Installation ist diese Angabe immer gültig, unabhängig davon, ob Lage und Größe des Datenbereichs vom Betriebssystem oder vom Programm festgelegt wurden.

Endadresse des Datenbereichs, physikalisch (ULONG, rel. Adr. 24)

Diese Adresse gibt das Ende des Datenbereichs (letzte Adresse + 1) an. Nach der Installation ist diese Angabe immer gültig, unabhängig davon, ob Lage und Größe des Datenbereichs vom Betriebssystem oder vom Programm festgelegt wurden.

Notizen

F. Parameter des Betriebssystems OsX

Das Betriebssystem OsX selbst ist ebenfalls eine Task auf dem MAX-PC: Task 0. Damit stehen alle Taskbefehle auch für den Zugriff auf die Strukturen des Betriebssystems zur Verfügung. Das Betriebssystem hat dementsprechend auch einen eigenen Parameterbereich, der teilweise auch für den Anwender von Interesse sein kann. Hier nicht erklärte bzw. reservierte Parameter dürfen auf keinen Fall verändert werden.

In der folgenden Tabelle bedeutet in der Spalte "Typ":

B = Ein-Byte Parameter
nB = n-Byte Parameter
W = Wort Parameter (2 Byte)
D = Doppelwort Parameter (4 Byte)
nS = n-Byte String (ASCII-Zeichen)
P = Physikalische Adresse (4 Byte)

In Spalte "Zugr" bedeutet: R = Parameter darf nur gelesen werden
RW = Parameter darf gelesen und geschrieben werden

Parameter	Typ	Zugr	Bedeutung
0 (00h)	B	R	Version dieser Parameterdefinition: z.Zt. = 01h
1 (01h)	W	R	Anzahl gültiger Parameter des Betriebssystems
4 (04h)	B	R	Kartentyp (11 = MAX-PC, Version X-MAX-1, 12 = MAX-PC, Version X-MAX-E)
5 (05h)	B	R	Kartenrevision (1=A, 4=D...)
10 (0ah)	2S	R	Jahrhundert der Herstellung des Betriebssystems, als Ergänzung des Datums in Parameter 22.
12 (0ch)	10S	R	Betriebssystem Name, Version und Revision, z.B.: "MAX11A.01x" x = "R" für RAM-Version, x = "E" für (EP)ROM-Version x = "B" für RAM-Beta-Test-Version
22 (16h)	8S	R	Datum der Herstellung des Betriebssystems, z.B.: "08/08/01" (tt/mm/jj)
30 (1eh)	8S	R	Uhrzeit der Herstellung des Betriebssystems, z.B.: "12:42:59" (hh:mm:ss)

Parameter	Typ	Zugr	Bedeutung
38 (26h)	B	R	CPU-Typ: 01h = V20, 04h = 486, 05h = Pentium
39 (27h)	B	R	CPU-Revision (siehe Intel-/AMD-Spezifikation)
40 (28h)	B	R	CPU-Model (z.Zt. nur gültig für Intel-Pentium)
41 (29h)	B	R	CPU-Hersteller: 1=Intel, 2=AMD, 3=UMC, 4=TI 5=Cyrix, 6=IBM, 7=STM, 8=NexGen, 9=NEC, 10 = Transmeta, 255 = nicht ermittelt
42 (2ah)	D	R	Ergebnis des Hardware-Selbst-Tests der CPU (0 = ok, <> 0 = Fehler)
46 (2eh)	B	R	Co-Proz.-Typ: 0 = keiner, 4 = 487, 5 = Pentium
47 (2fh)	B	R	Co-Proz.-Hersteller: 1 = Intel, 255 = nicht ermittelt
48 (30h)	D	R	CPU-Features (z. Zt. nur gültig für Intel-Pentium)
56 (38h)	W	R	Betriebssystem-Features: Bit-0: 0 = Datenzugriffe auf 1 MB beschränkt 1 = Datenzugriffe bis 4 GB erlaubt Bit-1: 0 = Datenbereiche werden von unten nach oben im Speicher reserviert 1 = von oben nach unten Bit-2: 0 = freies RAM wird nicht initialisiert 1 = freies RAM wird mit 0 initialisiert Bit-3: 0 = RAM-Größen Erkennung automatisch 1 = nicht automatisch, fix
64 (40h)	P	R	Physikal. RAM-Anfang (physikal. Adresse)
68 (44h)	P	R	Physikal. RAM-Ende (letzte Stelle + 1)
72 (48h)	P	R	Freies RAM-Anfang (erste freie Stelle)
76 (4ch)	P	R	Freies RAM-Ende (letzte Stelle + 1)
96 (60h)	W	R	Länge des Empfangs-Puffers für PC-Kommunikation, wenn = 0, dann max. 256
98 (62h)	W	R	Länge des Sende-Puffers für PC-Kommunikation, wenn = 0, dann max. 256
100 (64h)	W	R	Parameter-Nr., ab der die Modul-EEPROM-Tabelle steht, siehe separate Tabelle im Anschluss an diese Beschreibung

Parameter	Typ	Zugr	Bedeutung
146 (92h)	B	R	Zustand Batteriespannung an RTC während POWER ON: = 0: zum Zeitpunkt von POWER ON war keine Batterie angeschlossen bzw. die anliegende Spannung war < 2,4 V = 1: zum Zeitpunkt von POWER ON war eine Batterie angeschlossen bzw. die anliegende Spannung war ≥ 2,4 V Ist dieser Parameter = 1, dann wird angenommen, dass Uhrzeit und Datum gültig sind.
148 (94h)	W	R	CPU-Taktfrequenz (in Vielfachen von 1 MHz)
150 (96h)	W	R	Timer-Eingangstakt (in Vielfachen von 10 kHz)
168 (a8h)	W	R	Max. Anzahl Aktivierungen für NI-Tasks
170 (aah)	W	R	Max. Anzahl Tasks (alle Typen)
176 (b0h)	W	R	Max. Anzahl Puffer
200 (c8h)	B	RW	SRQ-Mode (PC-Schnittstelle): z.Zt. = 1
201 (c9h)	B	RW	SRQ-Delay (PC-Schnittstelle): z.Zt. = 64
202 (cah)	W	R	Größe des SRQ-Puffers in Bytes (Standardeinst. = 1024)
204 (cch)	W	R	Nummer des SRQ-Puffers
210 (d2h)	W	R	Zähler für Interrupts an IRQ-7 Master
212 (d4h)	W	R	Zähler für Interrupts an IRQ-7 Slave
214 (d6h)	B	RW	Watchdog: 0 = Auto-Retrigger off, 1 = on
215 (d7h)	B	R	NMI enabled (=0), disabled (=1)
217 (d9h)	B	R	Typ der CMD-Tabelle (0=2 Byte oder 1 = 4 Byte)
218 (dah)	D	R	Adresse Anfang der CMDTAB
240 (f0h)	B	R	Status der externen LED (1 = an, 0 = aus)
241 (f1h)	B	R	Status der lokalen LED (1 = an, 0 = aus)

Parameter	Typ	Zugr	Bedeutung
308 (134h)	W	R	Max. Anzahl TI-Tasks
310 (136h)	B	R	Typ des Timer-Chips (0 = 8254)
311 (137h)	B	RW[5]	Timer-Kanal für TI-Tasks (0 = Timer-A, 1=B, 2=C) (Standardeinstellung: Timer-C)
312 (138h)	B	R	Wo befindet sich der Timer ? (0 = MAX-PC)
313 (139h)	B	RW[5]	Interrupt-Nr. für TI-Task Timer (Standardeinst.: 93h)
316 (13ch)	D	RW[5]	Timer-Tic für TI-Tasks in Mikrosekunden Es sind Werte zwischen 100 und 26000 zulässig. Sonderfall: 0 bedeutet 1 ms (=Standardeinstellung)
320 (142h)	W	R	Längste Verzögerungszeit einer TI-Task seit Reset (Tics)
322 (144h)	W	R	TI-Task, die das Maximum (Parameter 320) ausgelöst hat
324 (146h)	W	RW	Meldegrenze für Zeitverzögerung einer TI-Task (in Timer-Tics, ffffh = keine Meldung)
326 (148h)	W	RW	Error-Request-Wort bei Erreichen der Meldegrenze (Parameter 320 > Parameter 324)
398 (16eh)	W	R	Aktuell bearbeitete NI-Task
400 (170h)	W	R	Parameter-Nr., wo Fehler bei System-Subroutinen gemerkt werden
402 (172h)	W	R	Parameter-Nr., wo Fehler bei Makro-Befehlen gemerkt werden
442 (1bah)	B	R	Eigene Steckplatz-Nr. (Slot^Layer-Nr)
484 (1e4h)	W		Status Control-Register

[5] Die Parameter für TI-Tasks können nur vor der ersten Installation einer TI-Task eingestellt werden, spätere Einstellungen sind wirkungslos (Zur Zeit nur Timer-C möglich).

Modul-EEPROM-Tabelle (8 Byte Header, dann je Modul 2 Byte Slot^Layer-Nummer und Status und eine Kopie) Die Parameter-Nr. des Anfangs der Tabelle steht in Parameter 100.

Parameter	Typ	Zugr	Bedeutung
x+0	B	R	Typ der Tabelle (z.Zt. = 1)
x+1	B	R	Länge Vorspann (z.Zt. = 8)
x+2	B	R	Anzahl Statusbytes pro Modul (z.Zt. = 2)
x+3	B	R	Max. Anzahl Module in dieser Tabelle (z.Zt. = 16)
x+4	W	R	Anzahl Byte je Modul (z.Zt. = 64)
x+6	B	R	Anzahl gültiger Module in dieser Tabelle
x+7	B	R	Reserviert
x+8	B	R	Slot^Layer-Nummer des 1. Eintrags
x+9	B	R	Status des EEPROMs des 1. Eintrags 00h = EEPROM ohne Fehler kopiert 14h = Falsche Kennung im EEPROM 20h = Semaphore nicht bekommen 2ah = Modul nicht bereit 2bh = EEPROM nicht bereit 80h = Kein Modul auf Steckplatz 85h = Falscher EEPROM-Typ
x+10	B	R	Slot^Layer-Nummer des 2. Eintrags
x+11	B	R	Status des EEPROMs des 2. Eintrags (s.o.)
..	..	..	..
x+38	B	R	Slot^Layer-Nummer des 16. Eintrags
x+39	B	R	Status des EEPROMs des 16. Eintrags (s.o.)
x+40	64B	R	Kopie von Byte 0..63 aus EEPROM des 1. Eintrags
..	..	..	..
x+1000	64B	R	Kopie von Byte 0..63 aus EEPROM des 16. Eintrags

x = Parameter-Nr, an der EEPROM-Tabelle für MAX-Module steht s.o.

G. Installationsdateien

Allgemeines

Installationsdateien bestehen aus einer Folge von Anweisungen, die verschiedene Aktionen ausführen. Das Programm SNW32 kann diese Installationsdateien ausführen. Die PC-Bibliotheken enthalten ebenfalls Routinen, um Installationsdateien abzuarbeiten. Zusätzlich können .INS-Dateien auch in das On-board Flash eines CPU-Moduls geschrieben werden. Nach einem Reset des Moduls werden sie dann automatisch ausgeführt (Autoinstallation).

Installationsdateien haben immer die Namenserweiterung '.INS'. Sie sind reine Textdateien, die mit jedem beliebigen Editor erstellt werden können. Ein komfortabler Editor ist in dem Programm SNW32 enthalten. Jede Zeile beginnt mit einem Schlüsselwort (=Befehl) oder mit einem Kommentarzeichen, Leerzeilen sind zulässig.

Schlüsselwörter

Alle zu einem Schlüsselwort gehörigen Parameter müssen in einer Zeile stehen. Alle Parameter sind hexadezimal anzugeben. Die kursiv gedruckten Bezeichner müssen durch Zahlenwerte oder Texte ersetzt werden.

Schlüsselwort	Bedeutung
MAXRESET	Reset einer MAX-Trägerkarte
MAXCONNECTCPU	Anwahl eines MAX-PC Moduls
MAXLOADOSX	Laden eines Betriebsystems auf einen MAX-PC
MAXINST	Installieren eines Echtzeitprogramms auf einen MAX-PC
MAXPROC	Prozedur einer Task aufrufen
MAXFUNC	Funktion einer Task aufrufen
MAXPAR	Parameter einer Task setzen
MAXLOADMDD	Modul-Device-Treiber eines Moduls laden
'	Kommentartext
;	Kommentartext

MAXRESET

Dieser Befehl setzt eine MAX-Trägerkarte und damit den X-Bus mitsamt allen aufgesteckten Modulen zurück.

Syntax

MAXRESET board=*board*

board: Nummer der MAX-Trägerkarte

MAXCONNECTCPU

Dieser Befehl stellt die Verbindung zu einem MAX-PC Modul her. **Alle weiteren Befehle beziehen sich dann auf dieses CPU-Modul – bis zum nächsten MAXCONNECTCPU Befehl.**

Syntax:

MAXCONNECTCPU board=*board* slot=*slot* layer=*layer*

board: Nummer der MAX-Trägerkarte.

slot: Slot-Nummer des Moduls.

layer: Layer-Nummer des Moduls.

MAXLOADOSX

Dieser Befehl lädt ein Betriebssystem auf das CPU-Modul.

Syntax:

MAXLOADOSX osx="*osx*"

osx: Name des Betriebssystems, das auf das CPU-Modul geladen werden soll:
„ROM“: es wird das im Flash-ROM befindliche Betriebssystem aktiviert.
„*Dateiname*“: der String wird als Name einer Betriebssystem-Datei interpretiert.

MAXINST

Dieser Befehl lädt ein Echtzeitprogramm auf das CPU-Modul.

Syntax:

MAXINST file="*file*" no=*no* [task=*task*] [tasktype=*tasktype*] [irq=*irq*] [level=*level*] [datasize=*datasize*] [autoinit] [activate]

Parameter in eckigen Klammern sind optional.

file: Dateiname des Programms.

no: In der PDT eingetragene Nummer des Programms.

task: Tasknummer, unter der das Programm installiert werden soll. Erlaubte Werte sind 1 bis 1023. Wenn *task* nicht angegeben wird, vergibt das Betriebssystem die Tasknummer.

tasktype: Tasktyp: gibt an, welchen Typ das Programm haben soll, wenn es installiert wird. Mögliche Werte sind:
MAX_NI_TASK
MAX_II_TASK
MAX_TI_TASK
Wenn *tasktype* nicht angegeben wird, wird der in der PDT angegebene Tasktyp verwendet.

irq: Nummer des lokalen Interrupts, unter dem die Task installiert werden soll. Dieser Parameter muss nur bei Interrupt-Tasks ausgefüllt werden! Wenn *tasktype* weggelassen wird, wird auch *irq* ignoriert und aus der PDT genommen.

level: Privilegstufe des Programms. (0=niedrigste bis 3=höchste). Wenn *level* nicht angegeben wird, wird Privilegstufe 0 verwendet.

datasize: Länge des Datenbereichs. Wenn *datasize* weggelassen wird, wird der in der PDT angegebene Wert verwendet.

autoinit: Wenn dieser Parameter angegeben wird, wird sofort nach der Installation die Prozedur 1 des Programms (AUTO_INIT) ausgeführt.

activate: Wenn dieser Parameter angegeben wird, wird die Task nach der Installation vom Betriebssystem aktiviert.

MAXPROC

Dieser Befehl ruft die Prozedur einer Task auf.

Syntax:

MAXPROC task=*task* proc=*proc*

task: Nummer der Task, deren Prozedur aufgerufen werden soll.

proc: Nummer der Prozedur, die aufgerufen werden soll.

MAXFUNC

Dieser Befehl ruft die Funktion einer Task auf. (Falls die Funktion eine Antwort zurückgibt, erscheint diese im Informationsfenster des SNW32 INS-Datei Editors.)

Syntax:

MAXFUNC task=*task* func=*func* para=*p1 p2 p3 ... pn*

task: Nummer der Task, deren Funktion aufgerufen werden soll.

func: Nummer der Funktion, die aufgerufen werden soll.

p1... pn: Parameterbytes, die der Funktion übergeben werden. Die einzelnen Parameterbytes müssen jeweils durch Leerzeichen voneinander getrennt werden.

MAXPAR

Mit diesem Befehl können ein oder mehrere Parameter einer Task gesetzt werden. Die Daten werden byteweise übergeben.

Syntax:

MAXPAR task=*task* start=*start* para=*p1 p2 p3 ... pn*

task: Nummer der Task, deren Parameter gesetzt werden sollen.

start: Nummer des ersten Parameters, der gesetzt werden soll.

p1 ... pn: Wert, auf den die ab *start* angegebenen Parameter gesetzt werden sollen.

MAXLOADMDD

Dieser Befehl dient zum Laden eines Modul-Device-Treibers für ein Modul. Der Treiber wird auf dem mit MAXCONNECTCPU angewählten MAX-PC installiert.

Syntax:

MAXLOADMDD slot=*slot* layer=*layer* progno=*progno*

slot: Slot-Nummer des Moduls.

layer: Layer-Nummer des Moduls.

progno: Programm-Nummer des Modul-Device-Treibers.

; '

Nach einem Strichpunkt (;) oder einem Hochkomma (') kann beliebiger Text stehen. Er wird komplett (bis zum Ende der Zeile) ignoriert. Eventuell enthaltene Befehls-Schlüsselwörter werden natürlich auch ignoriert.

H. Application Notes zu X-Bus, X-KiT-3 und MAX-PCs

(auf CD oder über Internet www.sorcus.com)

AN079: Design von X40-Bus-Modulen

AN077: Trägerboard für X40-Bus Module

AN087: Schaltplan X-KiT-3

AN085: Windows CE auf MAX-PC

AN086: Linux auf MAX-PC

AN082: Makrobefehle für den MAX-PC

AN083: Assembler-Systemroutinen für OsX auf dem MAX-PC

Notizen

I. Stichwortverzeichnis

K

L

M

R

S

T

Z