Application Note AN082
Makrobefehle für den MAX-PC

Autor: HB (+HK) Datei: AN082_F.DOC (39 Seiten)

Inhaltsverzeichnis

1. Makrobefehle für den MAX-PC

Makrobefehle sind Abfolgen von Daten, die an einen MAX-PC gesendet werden, um auf dem MAX-PC bestimmte Aktionen auszulösen oder Daten mit dem MAX-PC auszutauschen. Das Betriebssystem OsX auf dem MAX-PC nimmt die Befehle entgegen, führt sie aus und beantwortet sie.

Einzelheiten über die Hardware für die Kommunikation zwischen MAX-PCs oder zwischen MAX-PC und PC sind in einer separaten internen Note beschrieben. Alle im folgenden aufgeführten Makrobefehle sind im Betriebssystem OsX des MAX-PC enthalten. Sie sind nur für jene Anwender interessant, die eigene Treiber schreiben wollen.

Die Programm-Bibliotheken von SORCUS beinhalten alle Makrobefehle.

Makrobefehle können auch gesendet werden, während Anwenderprogramme auf dem MAX-PC laufen. Es ist Vorsicht geboten, wenn mit den Befehlen die gleichen Funktionseinheiten auf der Karte angesprochen werden, die auch von einem gerade auf dem MAX-PC laufenden Anwenderprogramm benutzt werden. Zur Lösung dieses Problems stehen Semaphoren und Modul Device Treiber zur Verfügung (s.u.).

Bei der Kommunikation mit einem MAX-PC kann es nur die beiden folgenden Fälle geben:

Fall 1: Der Initiator fordert per Makrobefehl bestimmte Daten oder Aktionen des MAX-PC an. Der führt das aus und sendet immer eine Antwort zurück. Der Initiator sendet erst dann den nächsten Befehl, wenn er die Antwort komplett empfangen hat. Das erste Byte des Makrobefehls und das erste Byte der Antwort sind identisch.

Fall 2: Der MAX-PC kann unabhängig davon jederzeit selbst aktiv werden und per Service-Request (SRQ) einen anderen PC zu bestimmten Aktionen auffordern. Dies tut er, indem er ein Wort zum PC oder zu einem anderen MAX-PC sendet (eventuell mit Interruptauslösung auf dem Target). An Bit 7 des ersten Byte kann der PC unterscheiden, ob es sich um die Antwort auf einen zuvor gesendeten Makrobefehl (Bit 7 = 0) oder um eine spontane Aktivität, also einen SRQ, (Bit 7 = 1) handelt.

1.1. Das Format der Makrobefehle

Jeder Makrobefehl besteht aus mindestens 2 Byte hin und 2 Byte zurück. Das erste Byte enthält den Befehlscode, das zweite Byte enthält zwei Angaben: in den unteren 4 Bit steht ein Formatcode für den Befehl bzw. die Länge des Befehls – 2 (FCB= Format Code Befehl), in den oberen 4 Bit steht ein Formatcode für die Antwort bzw. die Länge der Antwort – 2 (FCA= Format Code Antwort).

Beim Formatcode für den Befehl bedeutet eine Angabe zwischen 0 und 12 die Anzahl Byte, die nach den ersten beiden Byte noch folgen. Angaben > 12 sind Codierungen für Sonderformate des Befehls (s.u.).

Beim Formatcode für die Antwort bedeutet eine Angabe zwischen 0 und 12 die Anzahl angeforderter Byte - 2. Angaben > 12 sind Codierungen für Sonderformate der Antwort. Dabei ist es auch möglich, die Länge der Antwort zunächst offen zu lassen und nur die maximale Länge vorzugeben. Die tatsächliche Länge der Antwort wird vom Max-PC selbst ermittelt und als Teil der Antwort zurückgegeben.

Eine Antwort hat folgenden Aufbau: Das erste Byte enthält wieder den Befehlscode, das zweite Byte ist immer eine Statusmeldung bzw. Längenangabe der Antwort, dann folgen das erste Byte der Nutzdaten bzw. die Längenangabe der Antwort oder die Anzahl verworfener Byte.

Bei einer Fehlermeldung werden immer 4 Byte zurückgeschickt. Eine Übersicht aller Fehlermeldungen (Fehlergruppe und Fehlertyp) finden Sie in Anhang B des MAX6pci-Handbuchs.

Das erste Byte eines Makrobefehls wird immer im Low Byte, das zweite im High Byte übertragen, etc.

Im folgenden ist jeweils nur das Antwortformat „lang“ angegeben. In allen Fällen kann aber auch das „kurze“ Format zurückkommen.

1.2. Erklärung der Abkürzungen

A	Alignment
a1...av	Nutzdatenbytes der Antwort
aE aF aG aH	4-Byte Adresse, E = Low Byte, H = High Byte
b	Byte
b1...bn	n Nutzdatenbytes beim Befehl (Byte 1 bis Byte n)
c	Code für Makrobefehl
cL cH	Call-Nr.
CA	Makrobefehls-Code in der Antwort (0...7fh)
CB	Makrobefehls-Code im Befehl (0...7fh)
dE dF dG dH	Datenbereich Länge, E = Low Byte, H = High Byte
eL eH	Fehlermeldung: eH = Typ und eL = Gruppe (siehe Handbuch)
f	Flag
FCA	Format-Code Antwort:

	FCA = 0...12: Länge der Antwort – 2 FCA = 15: Die Anzahl angeforderter Byte folgt als Wort, ggfls. nach der Angabe der Befehlslänge. Der Befehl muß die angeforderte Anzahl liefern oder einen Fehler melden. FCA = 14: Die Anzahl angeforderter Byte folgt als Wort, ggfls. nach der Angabe der Befehlslänge. Der Befehl kann die tatsächliche Länge selbst bestimmen, max. aber bis zur vorgegebenen Länge.
FCB	Format-Code Befehl: FCB = 0...12: Länge des Befehls – 2 FCB = 15: Die Länge des Befehls – 2 folgt als Wort. Der Befehl muß alle Daten übernehmen oder alle verwerfen und einen Fehler melden. Die Länge der Antwort ist grösser als die Anzahl angeforderter Byte. FCB = 14: Die Länge des Befehls –2 folgt als Wort. Der Befehl kann soviel Daten nehmen wie er will (dabei beginnt er mit den zuerst übergebenen bzw. ab der niedrigsten Adresse) und meldet die Anzahl verworfener Byte zurück.
m	Anzahl angefordeter Nutzdatenbyte der Antwort (mL, mH = Low Byte, High Byte von m)
n	Anzahl Nutzdatenbyte beim Befehl (0 bis 255)
nL nH	Anzahl Nutzdatenbyte beim Befehl, Low Byte, High Byte
oE oF oG oH	Offset
p	Priorität
(p)	Parameter-Nummer (2 Byte): pL pH
pL pH	Programmnummer
rE rF rG rH	relative Adresse (4 Byte, E = Low, H = High Byte)
s	Statusmeldung, Segment bzw. Länge der Antwort s = 0: Fehlermeldung folgt als Wort s = 2...255: Länge der Antwort (incl. CA und s) s = 1: Länge der Antwort folgt als Wort (sL, sH = Low Byte, High Byte von s)
SLN	Slot^Layer-Nummer
(t)	= Task-Nummer (2 Byte): tL tH (0 bis 1023)
v	Tatsächliche Länge der Antwort (vL, vH = Low Byte, High Byte von v)
w	Anzahl verworfener Nutzdatenbyte des Befehls (wL, wH = Low Byte, High Byte von w)
z	= Prozedur-Nr. (0 bis 255)

Allgemeines Format der Makrobefehle (1 Kästchen = 1 Byte):

Wenn ein Fehler gemeldet wird, ist das 2. Byte der Antwort = 0. Es kommen keine gültigen Daten zurück. Die an den Befehl übergebenen Daten werden bzw. wurden komplett verworfen. Die Antwort besteht im Fehlerfall immer aus 4 Byte:

Antwort	CA	0	eL	eH	Fehlermeldung

Die Antwort kann im Kurz- oder Langformat erfolgen. Das kann beim Aufruf nicht vorgegeben werden. Der Befehl entscheidet selbst darüber. Beim Kurzformat kann s und damit die Antwortlänge zwischen 2 und 255 liegen, beim Langformat ist s = 1, die Antwortlänge folgt und kann zwischen 2 und 65535 liegen.

Befehl kurz fix, Antwort kurz fix:

FCB = 0...n (n = 0...12), FCA = 0...m (m = 0...12), s = 2...m+2

Befehl	CB	FCA ⋮ FCB	b1	:::	bn	
Antwort	CA	s	a1	:::	am	

Befehl lang fix, Antwort kurz fix:

FCB = 15, FCA = 0...m (m = 0...12), s = 2...m+2

Befehl	CB	FCA ⋮ FCB	n+2L	n+2H	b1	:::	bn	
Antwort	CA	s	a1	:::	am			

Befehl kurz fix, Antwort lang fix:

FCB = 2...n+2 (n= 0...10), FCA = 15, s = 2...m+2 bzw. s = 1 (= langes Antwortformat)

Befehl	CB	FCA ⋮ FCB	mL	mH	b1	:::	bn	
Antwort (Kurzformat)	CA	s	a1	:::	am			
Antwort (Langformat)	CA	01h	m+4L	m+4H	a1	:::	am	

Befehl lang fix, Antwort lang fix:

FCB = 15, FCA = 15, s = 2...m+2 (m= 0...252) bzw. s = 1 (langes Antwortformat)

Befehl	CB	0ffh	n+4L	n+4H	mL	mH	b1	:::	bn
Antwort (Kurzformat)	CA	s	a1	:::	am				
Antwort (Langformat)	CA	01h	m+4L	m+4H	a1	:::	am		

Befehl lang Angebot, Antwort kurz fix:

FCB = 14, FCA = 0...m+2 (m = 0...10), s = 2...m+4 bzw. s = 1 (langes Antwortformat)

Befehl	CB	FCA ⋮ FCB	n+2L	n+2H	b1	:::	bn		
Antwort (Kurzformat)	CA	s	wL	wH	a1	:::	am		
Antwort (Langformat)	CA	01h	m+6L	m+6H	wL	wH	a1	:::	am

Befehl kurz fix, Antwortwunsch:

FCB = 2...n+2 (n = 0...10), FCA = 14, m = Anzahl Wunsch Nutzdaten, s = 2...v+2 (v ≤ m), v = tatsächliche Länge (v ≤ m)

Befehl	CB	FCA ⋮ FCB	mL	mH	b1	:::	bn	
Antwort (Kurzformat)	CA	s	a1	:::	av			
Antwort (Langformat)	CA	01h	v+4L	v+4H	a1	:::	av	

Befehl lang Angebot, Antwortwunsch:

FCB = 14, FCA = 14, s = 4...v+4 (v ≤ m), m = Anzahl Wunsch Nutzdaten, v = tatsächliche Anzahl gelieferter Nutzdaten (v ≤ m), w = Anzahl verworfener Daten des Angebots

Befehl	CB	0eeh	n+4L	n+4H	mL	mH	b1	:::	bn
Antwort (Kurzformat)	CA	s	wL	wH	a1	:::	av		
Antwort (Langformat)	CA	01h	v+6L	v+6H	wL	wH	a1	:::	am

1.3. Kommunikation PC – MAX-PC

Bei "Code/Data" sind die einzelnen Byte in der Reihenfolge angegeben, wie sie zur Karte geschickt werden. Das gilt auch für solche Angaben, die aus zwei Byte bestehen, wie z.B. die Task-Nummer (tL und tH). Das erste Byte des Befehls wird im Low Byte, das zweite im High Byte, das dritte dann wieder im Low Byte, etc., gesendet.

Code/Data	Funktion
00h[1] 00h	**Anforderung: Überprüfen, ob der MAX-PC bereit ist** Dieser Befehl sollte so oft gesendet werden, bis als Antwort 00h 02h zurückkommt.
00h 02h	Antwort: MAX-PC ist bereit für Kommunikation
01h[1] 20h	**Anforderung: Typ: des Zielsystems und Betriebssystem melden**
01h 04h 1xh 00h	Antwort: Der MAX-PC sendet 01h 04h 1xh 00h zurück, andere Karten nur 04h. Das 3. Byte kann beim MAX-PC weiter ausgewertet werden. Es zeigt an, welches Betriebssystem aktiv ist: x = 0: Mini-Betriebssystem im EPROM (nach Power-On), es sind nur wenige Makrobefehle verfügbar. x = 1: Volles Betriebssystem (Kopie aus EPROM) x = 2: Volles Betriebssystem (von einem Host aus geladen)

1.4. Konfiguration

Mit dem Makrobefehl 1bh werden verschiedene Optionen auf dem MAX-PC einstellbar sein, die den Programmablauf in Bezug auf PC-Makrobefehle regeln.

Code/Data	Funktion
1bh 02h 10h 10h	**PC-Makrobefehle unterbrechbar durch Interrupt-Tasks** (= Einstellung nach Reset)
1bh 02h	Antwort
1bh 02h 00h 10h	**PC-Makrobefehle nicht unterbrechbar**
1bh 02h	Antwort
3bh 02h SLN f	**Initialisiere Modul entsprechend dem Inhalt des zugehörigen EEPROMs** SLN: Steckplatz = Slot^Layer-Nummer Flag: f = 0: Initialisiere nur, wenn Bit 0 von WORT-1 des zugehörigen EEPROMs = 1 f = 1: Initialisiere in jedem Fall
3bh 02h	Antwort

[1] Im Mini-OsX enthalten

1.5. Zugriff auf Modul-Konfigurations-Bereich (privilegierte Befehle)

Code/Data	Funktion
1eh[1] 22h SLN a	**Anforderung: Konfigurations-Register lesen** SLN = Slot^Layer-Nummer, a = Registeradresse
1eh 04h wL wH	Antwort: w = Wort
1fh[1] 04h SLN a wL wH	**Konfigurations-Register schreiben** SLN = Slot^Layer-Nummer, a = Registeradresse w = Wort
1fh 02h	Antwort

1.6. Zugriff auf Speicher (privilegierte Befehle)

Code/Data	Funktion
24h 06h 00h 00h 00h 00h 55h aah	**Hardware Reset** (X-Bus Reset)
24h 02h	Antwort: Im lokalen Antwortpuffer steht: 24h 00h 55h aah 00h 00h 00h 00h.
24h[1] 06h aE aF aG aH 55h aah	**Start Programm ab absoluter Adresse (Real Mode)** Physikalische Adresse (32 Bit)
24h 02h	Antwort: Erst wenn die empfangen wurde, wird das Programm gestartet. Im lokalen Antwortpuffer steht: 24h 00h 55h aah aE aF aG aH.
24h[1] 08h aE aF aG aH sL sH 55h aah	**Start Programm ab absoluter Adresse (Protected Mode)** Adresse (Offset 32 Bit) Segment (immer = 0)
24h 02h	Antwort: Erst wenn die empfangen wurde, wird das Programm gestartet. Im lokalen Antwortpuffer steht: 24h 00h 55h aah aE aF aG aH sL sH.
25h[1] 25h aE aF aG aH SLN	**Hinterlege Real Mode Adresse (physikalisch) und Absender-SLN für SRQ cbh (Start Programm)** Physikalische Adresse (32 Bit) Slot^Layer-Nummer des Absenders
25h 04h s sSLN	Antwort: s = Status: 0 = Start-Mechanismus war frei (sSLN <=SLN), 1 = used by sSLN (wenn sSLN >< SLN, werden neue Adresse und SLN verworfen, nur sSLN kann Adresse ändern). Slot^Layer-Nummer

[1] Im Mini-OsX enthalten

Code/Data	Funktion
25h[1] 27h	**Hinterlege Protected Mode Adresse (Segment:Offset) und Absender-SLN für SRQ cbh (Start Programm)**
aE aF aG aH	Adresse (Offset 32 Bit)
sL sH	Segment (immer = 0)
SLN	Slot^Layer-Nummer des Absenders
25h 04h	Antwort:
s	s = Status: 0 = Start-Mechanismus war frei (sSLN <=SLN), 1 = used by sSLN (wenn sSLN >< SLN, werden neue Adresse und SLN verworfen, nur sSLN kann Adresse ändern).
sSLN	Slot^Layer-Nummer
25h[1] 21h	**Kopiere ROM-OsX ins RAM und trage Adresse und Absender-SLN für SRQ cbh (Start Programm) ein (nur in Mini-OsX erlaubt):**
SLN	Slot^Layer-Nummer des Absenders
25h 04h 01h SLN	Antwort: Slot^Layer-Nummer
26h[1] f6h	**Anforderung: Block von physikalischer Adresse a lesen**
mL mH	m = Anzahl Byte (0 bis 65531)
aE aF aG aH	Adresse (32 Bit)
26h 01h m+4L m+4H b1...bm	Antwort: m = Anzahl Byte 0...65531
2eh[1] 0fh	**Block an physikalische Adresse a schreiben**
n+6L n+6H	n = Anzahl Byte (0 bis 65529)
aE aF aG aH	Adresse (32 Bit)
b1...bn	
2eh 02h	Antwort
22h 42h	**Anforderung: Freien Speicherplatz melden**
00h A	Alignment: Der Anfang des freien Bereichs kann auf eine Grenze gelegt werden, deren Adresse durch eine 2er Potenz teilbar ist: A=0: Byte, A=1: Wort, A=2: Doppelwort, ...
22h 06h aE aF aG aH	Antwort: Größe des freien Speichers in Anzahl Byte

[1] Im Mini-OsX enthalten

Code/Data	Funktion
22h 8ah	**Anforderung: Speicherplatz reservieren**
s	Strategie: s=0: nur Größe des freien Speichers melden s=1: von unten nach oben, soviel wie angegeben s=2: von unten nach oben, soviel wie möglich, aber nicht mehr als angegeben s=3: von oben nach unten, soviel wie angegeben s=4: von oben nach unten, soviel wie möglich, aber nicht mehr als angegeben
A	Alignment: Der Anfang des zu reservierenden Bereichs kann auf eine Grenze gelegt werden, deren Adresse durch eine 2er Potenz teilbar ist A=0: Byte, A=1: Wort, A=2: Doppelwort, ...
tL tH hL hH	t = Task-Nummer (nur für statistische Zwecke) h = Speichernutzung (nur für statistische Zwecke) (dyn., stat., Code, Daten, Parameter, etc.)
nE nF nG nH	Größe des zu reservierenden Bereichs in Anzahl Byte (s = 1, 2, 3, 4), sonst ohne Bedeutung
22h 0ah gE gF gG gH	Antwort: Größe des reservierten Bereichs (s = 1, 2, 3, 4) bzw. ohne Bedeutung (s = 0)
aE aF aG aH	Anfangsadresse des reservierten Bereichs (physikal. Adresse) (s = 1, 2, 3, 4) bzw. Größe des freien Speichers (s = 0)
39h 11h s	**Anforderung: Cache ein / aus** s = 0 : Status von Cache melden s = 1 : Cache aus s = 2 : Cache Write-through s = 3 : Cache Write-back
39h 03h s	Antwort: Status, s.o.

1.7. Systemzugriffe: I/O (privilegierte Befehle) *

Code/Data	Funktion
27h[1] 12h pL pH	**Anforderung: Byte von 8-Bit breitem Port p lesen** p = Port (I/O-Adresse)
27h 03h b	Antwort: b = Byte
27h[1] 22h pL pH	**Anforderung: Wort von 16-Bit breitem Port p lesen** p = Port (I/O-Adresse)
27h 04h wL wH	Antwort: w = Wort
27h[1] 42h pL pH	**Anforderung: Doppelwort von 32-Bit breitem Port p lesen** p = Port (I/O-Adresse)
27h 06h wE wF wG wH	Antwort: w = Doppelwort
27h[1] 13h pL pH i	**Anforderung: Index i an 8-Bit Port p schreiben, dann Byte von p+1 lesen** p = Port (I/O-Adresse), i = Index
27h 03h b	Antwort: b = Byte
23h[1] 03h pL pH b	**Byte an 8-Bit breiten Port p schreiben** p = Port (I/O-Adresse), b = Byte
23h 02h	Antwort
23h[1] 04h pL pH wL wH	**Wort an 16-Bit breiten Port p schreiben** p = Port (I/O-Adresse), w = Wort
23h 02h	Antwort
23h[1] 06h pL pH wE wF wG wH	**Doppelwort an 32-Bit breiten Port p schreiben** p = Port (I/O-Adresse), w = Doppelwort
23h 02h	Antwort
23h[1] 05h pL pH i m b	**Index i an 8-Bit Port p schreiben, dann Byte von p+1 lesen, mit Maske m eine UND-Verknüpfung und mit Byte b eine ODER-Verknüpfung durchführen, dann das Ergebnis an p+1 schreiben** p = Port (I/O-Adresse) i = Index, m = Maske, b = Byte
23h 02h	Antwort

[1] Im Mini-OsX enthalten

* Auf die I/O-Adressen 0eeh und 0efh kann nicht zugegriffen werden.

1.8. Zugriff auf Puffer

Erläuterungen für das Arbeiten mit den Puffern finden Sie in Kapitel 5 des MAX6pci-Handbuchs. Der Zugriff auf die Puffer vom PC aus ist gleichwertig mit dem Zugriff von irgendeiner on- board Task aus.

Code/Data	Funktion
60h 8ah s	**Anforderung: Lege einen Puffer im lokalen Speicher an** Strategie: s=1: von unten nach oben, soviel wie angegeben s=2: von unten nach oben, soviel wie möglich, aber nicht mehr als angegeben s=3: von oben nach unten, soviel wie angegeben s=4: von oben nach unten, soviel wie möglich, aber nicht mehr als angegeben
A	Alignment: Der Anfang des Puffers kann auf eine Grenze gelegt werden, deren Adresse durch eine 2er Potenz teilbar ist: A=0: Byte, A=1: Wort, A=2: Doppelwort, ...
tL tH hL hH nE nF nG nH	t = Task-Nummer (nur für statistische Zwecke) h = Speichernutzung (nur für statistische Zwecke) n = Größe des zu reservierenden Puffers (Sollgröße in Anzahl Byte)
60h 0ah pE pF pG pH gE gF gG gH	Antwort: p = Puffer-Nr. g = Größe des Puffers (Ist-Größe in Anzahl Byte)
61h 04h pE pF pG pH	**Entferne einen Puffer aus dem Speicher der Karte** p = Puffer-Nr. ***(noch nicht implementiert!)***
61h 02h	Antwort
62h 04h pE pF pG pH	**Lösche den Inhalt eines Puffers** p = Puffer-Nr.
62h 02h	Antwort
63h 84h pE pF pG pH	**Anforderung: Melde den Status eines Puffers** p = Puffer-Nr.
63h 0ah nE nF nG nH eE eF eG eH	Antwort: n = Anzahl gültiger Zeichen im Puffer e = freier Platz im Puffer (in Anzahl Byte)
64h 14h pE pF pG pH	**Anforderung: Lies Byte aus Puffer** p = Puffer-Nr.
64h 03h b	Antwort: b = Datenbyte

Code/Data	Funktion
65h 24h pE pF pG pH	**Anforderung: Lies Wort aus Puffer** p = Puffer-Nr.
64h 04h wL wH	Antwort: w = Datenwort
66h 44h pE pF pG pH	**Anforderung: Lies Doppelwort aus Puffer** p = Puffer-Nr.
66h 06h dE dF dG dH	Antwort: d = Daten-Doppelwort
67h e8h mL mH s 00h	**Anforderung: Lies bzw. kopiere Block aus Puffer „soviel wie möglich“** m = max. Größe des Blocks (in Anzahl Byte) s = Strategie: Bit 0 = 0: reserviert Bit 1 = 0: Block aus Puffer auslesen Bit 1 = 1: Block kopieren (bleibt im Puffer) Bit 2 = 0: wenn es nicht sofort klappt, abbrechen Bit 2 = 1: mehrmals probieren bis es klappt
pE pF pG pH	p = Puffer-Nr.
67h 01h v+4L v+4H a1 ... av	Antwort: v = tatsächliche Größe des gelesenen/kopierten Blocks (Anzahl Byte) a1 bis av = Byte
67h f8h mL mH s 00h	**Lies bzw. kopiere Block aus Puffer „alles oder nichts“** m = max. Größe des Blocks (in Anzahl Byte) s = Strategie: Bit 0 = 0: reserviert Bit 1 = 0: Block aus Puffer auslesen Bit 1 = 1: Block kopieren (bleibt im Puffer) Bit 2 = 0: wenn es nicht sofort klappt, abbrechen Bit 2 = 1: mehrmals probieren bis es klappt
pE pF pG pH	p = Puffer-Nr.
67h 01h m+4L m+4H a1 ... am	Antwort: m = Anzahl zurückgelieferter Byte a1 bis am = Byte
68h 05h pE pF pG pH b	**Schreibe Byte in Puffer** p = Puffer-Nr. b = Datenbyte
68h 02h	Antwort

Code/Data	Funktion
69h 06h pE pF pG pH wL wH	**Schreibe Wort in Puffer** p = Puffer-Nr. w = Datenwort
69h 02h	Antwort
6ah 08h pE pF pG pH dE dF dG dH	**Schreibe Doppelwort in Puffer** p = Puffer-Nr. d = Daten-Doppelwort
6ah 02h	Antwort
6bh 2eh n+8L n+8H s 00h	**Anforderung: Schreibe Block in Puffer „soviel wie möglich“** n = Größe des Blocks (in Anzahl Byte) s = Strategie: Bit 0 = 0: reserviert Bit 1 = 0: reserviert Bit 2 = 0: wenn es nicht sofort klappt, abbrechen Bit 2 = 1: mehrmals probieren bis es klappt
pE pF pG pH b1...bn	p = Puffer-Nr. b1 bis bn = Byte
6bh 04h wL wH	Antwort: w = Anzahl nicht verwendeter Byte des Blocks
6bh 0fh n+8L n+8H s 00h	**Schreibe Block in Puffer „alles oder nichts“** n = Größe des Blocks (in Anzahl Byte) s = Strategie: Bit 0 = 0: reserviert Bit 1 = 0: reserviert Bit 2 = 0: wenn es nicht sofort klappt, abbrechen Bit 2 = 1: mehrmals probieren bis es klappt
pE pF pG pH b1...bn	p = Puffer-Nr. b1 bis bn = Byte
6bh 02h	Antwort

1.9. EEPROM

Code/Data	Funktion
28h f6h mL mH SLN aE aF aG	**Anforderung: Lesen eines Blocks aus dem EEPROM** m = Anzahl Byte zu lesen SLN = Slot^Layer-Nummer a = Rel. Adresse des zu lesenden Blocks im EEPROM
28h 01h m+4L m+4H b1...bm	Antwort: m = Anzahl zurückgelieferter Byte EEPROM-Bytes
29h 0fh n+6L n+6H SLN aE aF aG b1...bn	**Schreiben eines Blocks in das EEPROM** n = Anzahl Byte zu schreiben sp = Modulsteckplatz a = Rel. Adresse des zu schreibenden Blocks im EEPROM Zu schreibender Block
29h 02h	Antwort

Die gesamte EEPROM-Information steht nach einem Reset im Parameterbereich des Betriebssystems zur Verfügung, sofern das Lesen des EEPROMs ohne Probleme erfolgte. Dafür ist in Parameter 100 die Parameter-Nr. eingetragen, ab der die Modul-EEPROM-Tabelle steht (siehe auch Anhang F des MAX6pci-Handbuchs.

1.10. Echtzeit-Uhr

Die Uhr gibt die Zeit in Sekunden, Minuten und Stunden an. Das Datum wird mit Tag, Monat, Jahr, Jahrhundert und Wochentag ausgegeben. Schaltjahre werden automatisch berücksichtigt. Der Wochentag wird mit 0 = Sonntag bis 6 = Samstag angegeben.

Zusätzlich verfügt die Uhr über einen „Periodic Interrupt“, der zusammen mit dem „Alarm Interrupt“ und dem „Update-Ended Interrupt“ mit Interrupt-Eingang IRQ8 (= IRQ´-0 des Slave-Interrupt-Controllers) verbunden ist. Die Uhr kann also zusätzlich als Timer verwendet werden. Zwischen 15 Impulsfrequenzen kann gewählt werden (s.u.).

Der Alarm-Ausgang ist ebenfalls Interrupt-fähig. Durch Angabe von sog. Wildcards kann auch der Alarm als Timer verwendet werden, allerdings mit eingeschränkten Einstellmöglichkeiten.

Die Uhr kann mit einer externen Batterie gepuffert werden, wenn erforderlich. Hierzu kann z.B. auch die Batterie des PC verwendet werden (siehe Kapitel 2, MAX6pci-Handbuch).

Code/Data	**Funktion**
34h 20h	**Uhr-Status lesen**
34h 04h sL sH	Antwort: s = Status der Uhr: Bit 0: Daylight Saving Bit 1: 12- oder 24-Stunden Betriebsart (1 = 24h-Mode) Bit 2: Data Mode (0 = BCD, 1 = Binary) Bit 3: = 0 (reserviert) Bit 4: Update-Ended Interrupt enabled (= 1) Bit 5: Alarm Interrupt enabled (= 1) Bit 6: Periodic Interrupt enabled (= 1) Bit 7: Uhr: 1 = Stop, 0 = Run Bit 11..8: Periodendauer am Periodic Interrupt: 0000b = disabled — 1000b = 3,906 ms 0001b = 3,906 ms — 1001b = 7,812 ms 0010b = 7,812 ms — 1010b = 15,625 ms 0011b = 122,070 µs — 1011b = 31,250 ms 0100b = 244,141 µs — 1100b = 62,500 ms 0101b = 488,281 µs — 1101b = 125 ms 0110b = 976,563 µs — 1110b = 250 ms 0111b = 1,953 ms — 1111b = 500 ms Bit 14..12: Interne Kontroll-Bits, standardmäßig = 010b Bit 15: Interner Update in der Uhr läuft gerade (= 1)

Code/Data	Funktion
34h 10h	**Uhr-Interrupt-Status lesen**
34h 03h s	Antwort: s = Status der Uhr: alle Bits werden durch das Lesen = 0 gesetzt Bit 3..0: reserviert (= 0) Bit 4: Update-Ended Interrupt Flag Bit 5: Alarm Interrupt Flag Bit 6: Periodic Interrupt Flag Bit 7: Interrupt Request Flag (wird nur gesetzt, wenn mind. einer der Interrupts enabled ist)
35h 02h sL sH	**Uhr: Betriebsart setzen:** s = Betriebsart der Uhr: Bit 0: Daylight Saving (1 = enable) Bit 1: 12- oder 24-Stunden Mode (1 = 24h-Mode) Bit 2: Data Mode (0 = BCD, 1 = Binary) Bit 3: = 0 (reserviert) Bit 4: Update-Ended Interrupt enable (= 1) Bit 5: Alarm Interrupt enable (= 1) Bit 6: Periodic Interrupt enable (= 1) Bit 7: Uhr: 1= Stop, 0 = Run Bit 11..8: 0000b = disabled; 0001b = 3,906 ms; 0010b = 7,812 ms; 0011b = 122,070 µs; 0100b = 244,141 µs; 0101b = 488,281 µs; 0110b = 976,563 µs; 0111b = 1,953 ms; 1000b = 3,906 ms; 1001b = 7,812 ms; 1010b = 15,625 ms; 1011b = 31,250 ms; 1100b = 62,500 ms; 1101b = 125 ms; 1110b = 250 ms; 1111b = 500 ms Bit 14..12: Interne Kontroll-Bits, standardmäßig = 010b setzen Bit 15: = 0 setzen
35h 02h	Antwort
36h 30h	**Uhr: Zeit lesen**
36h 05h h m s	Antwort: h = Stunden (0 bis 23) m = Minuten (0 bis 59) s = Sekunden (0 bis 59)

Code/Data	Funktion
36h 80h	**Uhr: Datum und Zeit lesen**
36h 0ah H J M T W h m s	Antwort: H = Jahrhundert J = Jahr (0 bis 99) M = Monat (1 bis 12) T = Tag (1 bis 31) W = Wochentag (0 bis 6, 0 = Sonntag) h = Stunden (0 bis 23) m = Minuten (0 bis 59) s = Sekunden (0 bis 59)
37h 08h H J M T W h m s	**Uhr: Datum und Zeit stellen** (Wenn eine der Angaben außerhalb des erlaubten Bereichs liegt, wird nichts neu gesetzt). H = Jahrhundert J = Jahr (0 bis 99) M = Monat (1 bis 12) T = Tag (1 bis 31) W = Wochentag (0 bis 6, 0 = Sonntag) h = Stunden (0 bis 23) m = Minuten (0 bis 59) s = Sekunden (0 bis 59)
37h 02h	Antwort
37h 04h 0 h m s	**Uhr: Alarmzeit setzen** h = Stunden (0 bis 23) m = Minuten (0 bis 59) s = Sekunden (0 bis 59)
35h 02h	Antwort
36h 40h	**Uhr: Alarmzeit lesen**
36h 06h 0 h m s	Antwort: h = Stunden (0 bis 23) m = Minuten (0 bis 59) s = Sekunden (0 bis 59)

Code/Data	Funktion
47h 0fh n+4L n+4H aL aH b1..bn	**CMOS-RAM: Block schreiben** n = Anzahl Byte zu schreiben (0 bis 114 (=72h)) a = Erste CMOS-Adresse (0eh bis 7fh) b = Data
47h 02h	Antwort
46h f4h mL mH aL aH	**CMOS-RAM: Block lesen** m = Anzahl Byte zu lesen (0 bis 114 (= 72h)) a = Erste Adresse (14 bis 127 (= 0eh bis 7fh)
46h m+2 d1..dm	Antwort: m = Länge der Antwort d = Data

1.11. Kontroll-LED

Code/Data	Funktion
30h[1] 00h	**on-board LED ein**
30h 02h	Antwort
31h[1] 00h	**on-board LED aus**
31h 02h	Antwort
33h[1] 10h	**Anforderung: Zustand der on-board LED**
33h 03h b	Antwort: Zustand der LED (Bit 0: 0 = aus, 1 = ein)

[1] Im Mini-OsX enthalten

1.12. Makro-Befehle für das Flash-EPROM

Code/Data	Funktion
2ah 24h	**Anforderung: FLASH_STATUS setzen bzw. lesen**
SLN IC d p	SLN=Modul-Steckplatz (= Slot^Layer-Nummer), IC=IC-Nr. d=Dienst (s.u.), p=Parameter

d	p	Funktion
0	0	Hersteller-ID des Flash-EPROMs ermitteln
1	0	Device-ID des Flash (Device-ID) ermitteln
2	0	Länge Flash (L) und Länge Sector (H) ermitteln: Länge (in Anzahl Byte) = 2**L bzw. 2**H
3	0	**Organisation** des Flash lesen (L=Org., H=0) L (untere 4 Bit) =0: Bit, =1: Byte, =2: Word L (obere 4 Bit) =0: kein Sector schützbar, =1: Bottom, =2: Top, =3: alle Sektoren schützbar, H=Größe Write-Buffer L=1: Write-Protected, L=0: Not-Protected, H=Größe Write-Buffer
5	s	**Status nach Erase von Sector/en s** L=0, H=0: Löschen fertig L=1, H=0: Löschen noch nicht fertig L=2, H=0: Fehler bei Löschen aufgetreten
10	0	**Lies Soft-State** (L = Soft-State, H=0)
11	0	**Setze "Erase enable"** (Soft-State 0 => 1)
-	-	Makro-Befehl: Erase Flash (Soft-State 1 => 2)
12	0	**Setze "Erase disable"** (Soft-State 0, 1 oder 2 => 0)
13	0	**Setze "Program enable"** (Soft-State 0 => 3)
-	-	Makro-Befehl "Programmiere Flash"
14	0	**Setze "Program disable"**

2ah 04h L H	Antwort: L, H: Ergebnis je nach Dienst
2bh 2ah SLN IC aE aF aG aH eE eF eG eH	**Anforderung: FLASH_ERASE: Erase Flash-EPROM direkt** SLN=Slot^Layer-Nummer, IC=IC-Nr. a = Erste zu löschende Adresse (bzg. auf Anfang Flash) e = Letzte zu löschende Adresse (bzg. auf Anfang Flash)
2bh 04h x 0	Antwort: x = Nr. eines gelöschten Sectors
2ch 0fh n+8L n+8H SLN IC rE rF rG rH d1...dn	**FLASH_PROGRAM: Flash direkt programmieren** n = Anzahl Byte SLN=Slot^Layer-Nummer, IC=IC-Nr. r = Relative Adresse des ersten zu progr. Bytes d = Data 1 bis n
2ch 02h	Antwort

Code/Data	Funktion
2dh f8h mL mH SLN IC rE rF rG rH	**Anforderung: FLASH_READ: Flash direkt lesen** m = Anzahl zu lesender Byte SLN=Slot^Layer-Nummer, IC=IC-Nr. r = Rel. Adresse des ersten zu lesenden Bytes
2dh 01h m+4L m+4H d1...dm	Antwort: m = Anzahl gelesener Bytes Data 1...m

1.13. Makro-Befehle für MDD-Dienste

Code/Data	Funktion
78h 06h hE hF hG hH sL sH	**Sonderdienst ohne Parameter** h = Handle s = Sonderdienst-Kennung
78h 02h	Antwort
78h 07h hE hF hG hH sL sH d	**Sonderdienst mit Parameter Byte Hin** h = Handle s = Sonderdienst-Kennung d = Parameter Byte
78h 02h	Antwort
78h 08h hE hF hG hH sL sH dL dH	**Sonderdienst mit Parameter Wort Hin** h = Handle s = Sonderdienst-Kennung d = Parameter Wort
78h 02h	Antwort
78h 0ah hE hF hG hH sL sH dE dF dG dH	**Sonderdienst mit Parameter Doppelwort Hin** h = Handle s = Sonderdienst-Kennung d = Parameter Doppelwort
78h 02h	Antwort
78h 16h hE hF hG hH sL sH	**Anforderung: Sonderdienst mit Parameter Byte Rück** h = Handle s = Sonderdienst-Kennung
78h 03h d	Antwort: d = Parameter Byte
78h 26h hE hF hG hH sL sH	**Anforderung: Sonderdienst mit Parameter Wort Rück** h = Handle s = Sonderdienst-Kennung
78h 04h dL dH	Antwort: d = Parameter Wort

Code/Data	Funktion
78h 46h hE hF hG hH sL sH	**Anforderung: Sonderdienst mit Parameter Doppelwort Rück** h = Handle s = Sonderdienst-Kennung
78h 06h dE dF dG dH	Antwort: d = Parameter Doppelwort
78h 2eh n+8L n+8H hE hF hG hH sL sH b1...bn	**Anforderung: Sonderdienst mit Stream Hin** n = Anzahl Byte Nutzdaten h = Handle s = Sonderdienst-Kennung Nutzdaten
78h 04h wL wH	Antwort: w = Anzahl verworfener Nutzdaten
78h 0e8h mL mH hE hF hG hH sL sH	**Anforderung: Sonderdienst mit Stream Rück** m = max. Anzahl Byte Nutzdaten Rück h = Handle s = Sonderdienst-Kennung
78h 01h v+4L v+4H b1...bv	Antwort: v = tatsächliche Anzahl Nutzdaten Rück Nutzdaten
78h 0eeh n+10L n+10H mL mH hE hF hG hH sL sH b1...bn	**Anforderung: Sonderdienst mit Stream Hin und Rück** n = Anzahl Nutzdaten hin m = max. Anzahl Byte Nutzdaten Rück h = Handle s = Sonderdienst-Kennung Nutzdaten
78h 01h v+6L v+6H wL wH b1...bv	Antwort: v = tatsächliche Anzahl Nutzdaten Rück w = Anzahl verworfener Nutzdaten (w ≤ n) Nutzdaten
79h 14h hE hF hG hH	**Anforderung: Lies Einzel-Data: Bit** h = Handle
79h 03h d	Antwort: d = Bit
79h 14h hE hF hG hH	**Anforderung: Lies Einzel-Data: Byte** h = Handle
79h 03h d	Antwort: d = Byte

Code/Data	Funktion
79h 24h hE hF hG hH	**Anforderung: Lies Einzel-Data: Wort** h = Handle
79h 04h dL dH	Antwort: d = Wort
79h 44h hE hF hG hH	**Anforderung: Lies Einzel-Data: Doppelwort** h = Handle
79h 06h dE dF dG dH	Antwort: d = Doppelwort
7ah 04h hE hF hG hH	**Schreibe Einzel-Data: Trigger** h = Handle
7ah 02h	Antwort
7ah 05h hE hF hG hH d	**Schreibe Einzel-Data: Bit** h = Handle d = Bit
7ah 02h	Antwort
7ah 05h hE hF hG hH d	**Schreibe Einzel-Data: Byte** h = Handle d = Byte
7ah 02h	Antwort
7ah 06h hE hF hG hH dL dH	**Schreibe Einzel-Data: Wort** h = Handle d = Wort
7ah 02h	Antwort
7ah 08h hE hF hG hH dE dF dG dH	**Schreibe Einzel-Data: Doppelwort** h = Handle d = Doppwort
7ah 02h	Antwort
7bh 0e6h mL mH hE hF hG hH	**Anforderung: Lies Daten-Stream** m = max. Anzahl Byte Nutzdaten Rück h = Handle
7bh 01h v+4L v+4H a1...av	Antwort: v = tatsächliche Anzahl Nutzdaten Rück Nutzdaten
7ch 2eh n+6L n+6H hE hF hG hH d1...dn	**Anforderung: Schreibe Daten-Stream** n = Anzahl Byte Nutzdaten h = Handle Nutzdaten
7ch 04h wL wH	Antwort: w = Anzahl verworfener Nutzdaten

1.14. Die Taskbefehle

1.14.1. Das Prinzip

Das Betriebssystem ordnet der Task, unter der ein Programm installiert wird, einen Parameter- und einen Datenbereich zu. Die Länge beider Bereiche kann auch = 0 sein. Die Länge des Parameterbereichs wird vom Programm selbst vorgegeben und ist damit konstant. Für die Angabe der Länge des Datenbereichs gibt es verschiedene Möglichkeiten, sie kann auch vom Anwender beim Installieren angegeben werden.

Das Programm selbst besteht aus allgemein aufrufbaren Prozeduren bzw. Funktionen, die auch von einer anderen Task oder vom PC aus aufgerufen werden können.

Vom PC aus kann mit Makrobefehlen auf die Strukturen (Prozeduren, Funktionen, Parameter, Daten) jeder Task zugegriffen werden: Es können Funktionen aufgerufen werden, Parameter gesetzt und gelesen werden und Daten geschrieben oder gelesen werden.

1.14.2. Befehle zur Installierung und Taskverwaltung

1.14.2.1. Makrobefehl 40h zum Installieren

Vom PC aus wird mit diesem Makrobefehl ein Programm unter einer Task installiert. Hierbei sind verschiedene Optionen möglich.

Code/Data	Funktion
40h 2fh 14h 00h	**Anforderung: Installiere Programm** (Erklärung s. unten)
tL th	t = Tasknummer
pL ph	p = Programmnummer
iL 00h	i = Interrupt-Nummer
fE fF fG fH	f = Flag
dE dF dG dH	d = Größe des zu reservierenden Datenbereichs
aE aF aG aH	a = Adresse
40h 04h tL tH	Antwort: Task-Nr.

Erläuterungen zum Makrobefehl 40h zum Installieren

t = Task-Nummer (Wort)

Dies ist die Task-Nummer, unter der das Programm installiert werden soll. Erlaubt sind Angaben von 1 bis 1023. Task 0 ist immer für das Betriebssystem reserviert. Task 1 bis 15 sollten nicht für Anwendungsprogramme benutzt werden.

p = Programmnummer (Wort)

Die Programmnummer kann von 1 bis 65534 betragen. Die im Makrobefehl angegebene Nummer muß mit der Nummer in der PDT übereinstimmen. Programmnummer 0 ist reserviert für das Betriebssystem, Programmnummer ffffh (=-1) ist ein Dummy-Programm ohne Funktion. Sie wird gemeldet, wenn kein Programm unter einer Task installiert ist. Je nach Typ der Karte heißen die Programme X1P... (MAX-PC), M7P... („kleine“ MODULAR-4/486) oder M8P... („große“ MODULAR-4/486). Die Programmnummer erscheint auch als 4-stellige hexadezimale Zahl im Namen von Programmdateien und in dazugehörigen Dateien:

X1Pnnnn.LIB = Anwenderprogramm (identisch mit .OBJ)
X1Pnnnn.LAB = Relocated Anwenderprogramm
X1Pnnnn.SEX = SORCUS EXE Datei (identisch mit .EXE)
X1Pnnnn.SRX = SORCUS Relocated EXE Datei
X1Pnnnn.SHX = SORCUS Hypertext Datei

i = Interrupt-Nummer (Byte) (siehe Kapitel 10, MAX6pci-Handbuch)

Bei NI-Tasks wird diese Angabe nicht ausgewertet. Bei Interrupt-Tasks gibt die Interrupt-Nummer an, welcher Interrupt der Task, die gerade installiert wird, zugeordnet wird. Wenn dieser Interrupt auftritt, wird die Hauptprozedur des Programms aufgerufen, das unter der Task installiert ist.

Für die Festlegung des Interrupts gibt es zwei Möglichkeiten, durch die PDT (= Programm Deskriptor Tabelle) oder mit Makrobefehl 40h zum Installieren (siehe auch bei „Task-Typ“). Hierfür sind zwei Bit zuständig: Bit 3 im Flag in der PDT und Bit 3 im Flag im Makrobefehl. Dies sind die gleichen Bit, die auch für die Festlegung des Task-Typs verwendet werden.

f = Flag (Doppelwort = 32 Bit)

Die folgende Tabelle gibt eine Kurzerklärung für die Verwendung der 32 Bit im Flag.

Bit-Nr.	Anzahl Bit	Erklärung
0 – 2	3	Task-Typ (NI-, II-, TI-Task)
3	1	Wer legt Task-Typ und Interrupt-Nr. fest?
4, 5	2	Privilegstufe des Programms
6 – 8	3	Programm im RAM oder ROM?, in welchem Format?
9, 10	2	Wie wird Größe von Datenbereich festgelegt?
11	1	Auto-Init-Prozedur nach Installieren aufrufen? (1 = ja)
12	1	Task sofort nach Installieren aktivieren? (1 = ja)
13-14	2	DDT anlegen? (0 = nein)
15	1	Wer vergibt die Task-Nr.? (0=Host, 1=OsX)
16-		Reserviert für SORCUS

Task-Typ: NI-, II-, TI-Task (Bit 0 bis 2)

Hiermit wird angegeben, welchen Typ die installierte Task haben soll:

0 (=000b): NI-Task (Nicht-Interrupt-Task)
1 (=001b): II-Task (Indirekte Interrupt-Task)
3 (=011b): TI-Task (Timer-Initiierte-Task)

Für die Festlegung des Task-Typs (und der Interrupt-Nummer) gibt es zwei Möglichkeiten, entweder durch das zu installierende Programm selbst oder mit Makrobefehl 40h zum Installieren. Hierfür sind zwei Bit zuständig: Bit 3 im Flag in der PDT und Bit 3 im Flag im Makrobefehl. Diese Bit werden auch für die Festlegung der Interrupt-Nummer verwendet.

Wer legt Task-Typ und Interrupt-Nummer fest? (Bit 3)

Wenn Bit 3 im Flag der PDT = 1 gesetzt ist, wird dieses Bit ignoriert und Task-Typ und Interrupt-Nummer werden in jedem Fall aus der PDT verwendet. Die Angaben „Task-Typ“ und „Interrupt-Nummer“ im Makrobefehl werden verworfen.

Wenn Bit 3 im Flag der PDT = 0 gesetzt ist, erlaubt das Programm, beide Angaben auch per Makrobefehl festzulegen. Hierzu muß dieses Bit = 1 gesetzt werden. Natürlich müssen die Angaben von Task-Typ und Interrupt-Nummer gültige Werte aufweisen. Wenn dieses Bit = 0 ist, werden wiederum die Angaben für Task-Typ und Interrupt-Nummer aus der PDT verwendet.

Privilegstufe des Programms (Bit 4 und 5)

0 (=00b) ist die höchste Privilegstufe, 3 (=11b) die niedrigste. Anwendungsprogramme haben immer die Privilegstufe 3, Systemprogramme die höchste, also 0. Diese Flags werden beim MAX-PC nicht ausgewertet und können immer = 0 gesetzt werden.

Programm im ROM oder RAM und Programmformat (Bit 6 bis 8)

Wenn das zu installierende Programm im RAM steht, wird die Adresse a als absolute physikalische Adresse angesehen, ihre Bedeutung richtet sich danach, in welchem Format das Programm im RAM der Karte steht. Dies wird dem Betriebssystem über Bit 6 bis 8 im Flag des Makrobefehls mitgeteilt:

Wenn das zu installierende Programm im ROM vorhanden ist, wird die im Makrobefehl angegebene Adresse a ignoriert, entscheidend ist dann die im Makrobefehl angegebene Programmnummer.

Wo?	**Programmformat**	**Adresse a**	**Format Adresse**	**Flag-Bit* 8 7 6**
RAM	PDT (tiny)	Anfang PDT	physikal.	0 0 0
RAM	PDT (large)	Anfang PDT	physikal.	1 0 0
RAM	EXE nicht relocated	Anfang EXE-Header	physikal.	1 1 0
RAM	EXE relocated	PREPARE-Code	Seg:Offs.	0 1 0
ROM	PDT	wird ignoriert	-	0 0 1

* alle anderen Kombinationen sind zur Zeit nicht erlaubt

Wie wird die Größe des Datenbereichs festgelegt? (Bit 9 und 10)

Hierfür gibt es mehrere Möglichkeiten (siehe folgende Tabelle). Wenn Bit 9 im Flag der PDT = 1 gesetzt ist, kann die Größe durch den Makrobefehl nicht verändert werden. Der Normalfall ist der, daß die Größe durch d im Makrobefehl beim Installieren angegeben wird. Die maximal und minimal mögliche Größe kann durch die Vorgaben in der PDT eingeschränkt werden, falls ein Programm z.B. nur einen max. 64 K großen Datenbereich unterstützt.

		Flag in PDT	**Flag in Makrobefehl**	
fix/variabel	**Größe richtet sich nach**	**Bit 9**	**Bit 10**	**Bit 9**
fix	„Größe“ in PDT	1	x	x
fix	„Größe“ in PDT	0	1	1
fix	„Minimum“ in PDT	0	1	0
fix	„Maximum“ in PDT	0	0	1
variabel	d in Makrobefehl	0	0	0

Auto-Init aufrufen? (Bit 11)

Wenn dieses Bit = 1 gesetzt ist, wird unmittelbar nach dem Installieren sofort die Auto-Init Prozedur (Prozedur 1) aufgerufen. Wenn das Bit = 0 gesetzt ist, nicht.

Task nach Installieren sofort aktivieren? (Bit 12)

Wenn dieses Bit = 1 gesetzt ist, wird nach dem Installieren und ggfls. nach dem Aufruf der Auto-Init Prozedur die Task auch sofort aktiviert. Wenn das Bit = 0 gesetzt ist, nicht.

Debug-Informationen (DDT) anlegen ? (Bit 13-14)

Bit 13-14 können folgende Werte annehmen: 00b=nein, 01b=für Borland Turbo Debugger, 10b und 11b reserviert.

Wer vergibt die Tasknummer? (Bit 15)

Wenn dieses Bit = 0 gesetzt ist, wird die Task-Nummer aus dem Makro-Befehl verwendet. Wenn dieses Bit = 1 gesetzt ist, wird die Task-Nummer von OsX vergeben.

Reserve (Bit 16 bis 31)

d = Größe Datenbereich (Doppelwort)

Die Größe wird in Anzahl Byte angegeben. Weitere Erklärungen finden sich beim Flag zu Bit 9 und 10 (s.o.).

a = Adresse (Doppelwort)

Die Adresse wird als physikalische oder Segment:Offset Adresse angegeben, siehe Flag, Bit 6 bis 8. Bei Verwendung der mitgelieferten Bibliothek wird die Formatanpassung automatisch übernommen.

1.14.2.2. Makrobefehle zur Taskverwaltung

Code/Data	Funktion
2fh 44h tL tH cL cH	**Anforderung: Task-Info** Task-Nr. (t = 0: Betriebssystem) Call-Nr. (Erklärung siehe Anhang C des MAX6pci-Handbuchs)
2fh 06h aE aF aG aH	Antwort: 4 Byte Ergebnis, z.B. Adresse
38h 22h pL pH	**Anforderung: Melde, ob Programm im ROM vorhanden ist** pL, pH = Nr. des Programms
38h 04h pL pH	Antwort: Wenn p = 0, dann ist das Programm nicht im ROM.
3ah 31h i	**Anforderung: Melde Nummer und Typ der Task, die den Interrupt i nutzt** i = Interrupt-Nr.
3ah 05h tL tH T	Antwort: Task-Nr. (wenn t = -1 (ffffh), dann ist der Interrupt i nicht benutzt) T = Task-Typ
4eh 44h nL nH pL pH	**Anforderung: Melde die Task, unter der Programm p installiert ist** n = Ordnungszahl der gesuchten Installierung p = Programmnummer
4eh 06h mL mH tL tH	Antwort: m = Ordnungszahl einer gefundenen Installierung (m ≤ n) t = Tasknummer der m-ten Installierung
41h[1] 02h tL tH	**Aktivieren einer II- bzw. DI-Task** tL, tH = Task-Nr. Der zugehörige Interrupt wird demaskiert. Dieser Befehl steht auch im ROM im Mini-OsX (= Mini-Betriebssystem) zur Verfügung (nur für Task 0). Es wird ein voll funktionsfähiges Betriebssystem OsX aus dem ROM des Moduls ins RAM des Moduls kopiert und aktiviert.
41h 02h	Antwort

[1] Mit t = 0 im Mini-OsX enthalten

Code/Data	Funktion
41h 03h tL tH p	**Aktivieren einer NI-Task** tL, tH = Task-Nr. p = 1: Aktiviere Task p = 2: Einmaliges Vorziehen einer NI-Task (unbedingt) p = 3: Einmaliges Vorziehen einer NI-Task. Wenn schon eine vorgezogen ist, erfolgt eine Fehlermeldung.
41h 02h	Antwort
41h 0fh 12h 00h tL tH p 00h zE zF zG zH iE iF iG iH nE nF nG nH	**Aktivieren einer TI-Task** (andere Tasktypen s.o.) t = Task-Nr. p = Prioritätszahl (0 = höchste Priorität) z = Hold-Off Zeit vom Aktivieren bis zum 1. Aufruf der Task (in Anzahl Timer-Tics[1]) i = Intervall zwischen 2 Aufrufen der Task (in Anzahl Timer-Tics) n = Anzahl Aufrufe, bis die Task automatisch wieder deaktiviert wird (0 = unendlich)
41h 02h	Antwort
42h 02h tL tH	**Deaktivieren einer Task (alle Tasktypen)** tL, tH = Task-Nr. (Task 0 ist nicht erlaubt) Ein laufendes Programm wird abgebrochen, ein gerade laufender Aufruf wird aber ordnungsgemäß beendet. Bei II-Tasks wird der zugehörige Interrupt maskiert. Eine mehrfach aktivierte NI-Task muß auch mehrfach wieder deaktiviert werden.
42h 02h	Antwort
43h 12h tL tH	**Anforderung: Melde, ob Task t aktiviert ist** tL, tH = Task-Nr.
43h 03h n	Antwort: n = Anzahl Aktivierungen (n = 0: Task ist nicht aktiviert)

[1] Der Timer-Tic kann einmalig vor der ersten Installierung einer TI-Task durch Setzen von Parameter 316 des Betriebssystems (siehe Anhang F des MAX6pci-Handbuchs) eingestellt werden. Defaultmäßig ist 1ms eingestellt.

Code/Data	Funktion
4fh f2h xL xH	**Anforderung: Melde alle installierten Tasks,** je Task 1 Bit (1=installiert) x = 0 (reserviert)
4fh 01h 84h 00h b1..b128	Antwort: b1...b128: 1024 Bit (Bit 0 von b1 = Task 0)
44h[1] 0eeh 0ah 00h 02h 04h 00h 00h 0eh 00h 01h 00h	**Anforderung: Master-Liste lesen/updaten (= Funktion 14 von OsX)** mL mH (= 1026 für max. 256 Master) Task 0 (=OsX) Funktion 14 Aufgerufene Unterfunktion: 1 = Master-Liste lesen/updaten
44h 01h v+6L v+6H 00h 00h 01h y mnr SP Status res.	Antwort: Länge der Antwort, v = Anzahl Nutzdatenbyte = Anzahl Master +2 Anzahl verworfener Byte = 0 1 = Typ der Liste y = Anzahl der Master, je Master folgen 4 Byte: Master-Nr. Steckplatz Status: 0 = keiner, 1 = passiv, 2 = aktiv schlafend, 3 = wach reserviert
44h[1] 0eeh 0ah 00h 02h 00h 00h 00h 0eh 00h 02h mnr	**Anforderung: Master-Status lesen (= Funktion 14 von OsX)** mL mH Task 0 (=OsX) Funktion 14 Aufgerufene Unterfunktion: 2 = Status des Masters mnr lesen
44h 01h 08h 00h 00h 00h mnr Status	Antwort: Länge der Antwort Anzahl verworfener Byte = 0 Master-Nr. Status: 0 = keiner, 1 = passiv, 2 = aktiv schlafend, 3 = wach

[1] Im Mini-OsX enthalten

1.14.3. Aufruf einer Prozedur bzw. Funktion

Beim Aufruf einer **Prozedur** einer Task werden keine Parameter an sie übergeben, als Antwort kommen immer zwei Byte zurück, die ggfls. einen Fehler melden können.

Beim Aufruf einer **Funktion** können Parameter hin und zurück übergeben werden (Anzahl Hin und Rück werden immer als Anzahl Byte angegeben und können auch = 0 sein):

Bezüglich der an die Funktion übergebenen Parameter meldet die Funktion zurück, wieviel Byte sie davon verwerten konnte. Sie beginnt mit der Verwertung sequentiell ab dem zuerst übergebenen Parameterbyte.

Beim Aufruf muß angegeben werden, wieviele Parameter (Anzahl Byte) von der Funktion maximal zurückerwartet werden. Die Funktion meldet immer zurück, wieviel Byte sie tatsächlich liefern konnte.

Code/Data	Funktion
45h 04h tL tH fL fH	**Aufruf der Prozedur f einer Task t** (es werden keine Parameter an die Prozedur über- geben oder von ihr zurückerwartet)
45h 02h	Antwort [1]
44h eeh n+8L n+8H mL mH tL tH fL fH b1...bn	**Anforderung: Aufruf der Funktion f einer Task t** n = Anzahl Byte Hin (= Anzahl der an die Funktion übergebenen Parameterbyte) m = Anzahl der max. zurückerwarteten Byte t = Task-Nr. f = Funktions-Nr. Parameter, die an die Funktion übergeben werden
44h 01h v+6L v+6H wL wH b1...bv	Antwort: [2] Anzahl der tatsächlich zurückgelieferten Byte Anzahl der verworfenen Parameterbyte Hin Parameter, die von der Funktion zurückgeliefert werden

1.14.4. Zugriff auf Parameter

Während des Zugriffs sind die Parameter vor dem gleichzeitigen Zugriff z.B. durch eine andere Task oder durch den PC geschützt. Um auf einzelne Parameter (Byte, Wort, Doppelwort) einer Task konsistent zugreifen zu können, können die bestehenden Funktionen Byte lesen/schreiben, Wort lesen/schreiben und Doppelwort lesen/schreiben eingesetzt werden. Die Funktionen zum Blocklesen (Makrobefehle 4ch und 4dh) dagegen sind durch Interrupts unterbrechbar, Konsistenz ist nicht mehr garantiert.

---

[1] Die im Fehlerfall in eL eH übergebenen Fehlercodes entsprechen den übrigen Fehlermeldungen, allerdings wird hier nur der Fehler-Typ eH gemeldet (siehe Anhang B des MAX6pci-Handbuchs), wenn die von der aufgerufenen Funktion gelieferte Fehlermeldung eL der Gruppe e0h ("Fehler bei Aufruf der Funktion") angehört, andernfalls wird eH = 1ah ("unbekannter Fehler, bzw. falsche Fehlergruppe von Funktion") gemeldet.

[2] Die im Fehlerfall in eL eH übergebenen Fehlercodes entsprechen den übrigen Fehlermeldungen, allerdings wird hier nur der Fehler-Typ eH gemeldet (siehe Anhang B des MAX6pci-Handbuchs), wenn die von der aufgerufenen Funktion gelieferte Fehlermeldung eL der Gruppe e0h ("Fehler bei Aufruf der Funktion") oder deh („MDD-Fehler“) angehört, andernfalls wird eH = 1ah ("unbekannter Fehler, bzw. falsche Fehlergruppe von Funktion") gemeldet.

Mit dem Makrobefehl 3ch kann sich eine Task eine Semaphore eines Parameterbereichs einer anderen oder auch der eigenen Task holen, um einen konsistenten Zugriff zu gewährleisten. Mit Makrobefehl 3dh gibt sie die Semaphore wieder frei. Je Task steht eine Semaphore für den Parameterbereich zur Verfügung, die automatisch verwaltet wird. Alle Tasks, die auf den Parameterbereich derselben Task zugreifen wollen, müssen immer zunächst versuchen, sich die zugehörige Semaphore zu holen. Die Task, die die Semaphore bekommen hat, kann dann mit allen angegebenen Funktionen konsistent auf den Parameterbereich zugreifen. Danach muß sie sie wieder freigeben. Die einzelnen Zugriffe selbst kümmern sich um die Semaphore nicht, gleichgültig, ob auf einzelne Parameter, auf einen Block oder ob direkt über die Adresse von Parametern zugegriffen wird.

Code/Data	**Funktion**
3ch 22h tL tH	**Anforderung: Parameter-Semaphore anfordern** t = Task-Nummer des Parameterbereichs
3ch 04h sL sHb	Antwort: s = 0: Semaphore bekommen, s <> 0: Semaphore nicht bekommen
3dh 02h tL tH	**Parameter-Semaphore freigeben** t = Task-Nummer des Parameterbereichs
3dh 02h	Antwort
58h 14h tL tH pL pH	**Anforderung: Parameter Byte lesen** t = Task-Nummer p = Nummer des zu lesenden Parameters
58h 03h b	Antwort: b = gelesenes Parameter Byte
59h 24h tL tH pL pH	**Anforderung: Parameter Wort lesen** t = Task-Nummer p = Nummer des zu lesenden Parameters
59h 04h wL wH	Antwort: w = gelesenes Parameter Wort
5ah 44h tL tH pL pH	**Anforderung: Parameter Doppelwort lesen** t = Task-Nummer p = Nummer des zu lesenden Parameters
5ah 06h dE dF dG dH	Antwort: d = gelesenes Parameter Doppelwort
4ch[1] f6h mL mH tL tH pL pH	**Anforderung: Parameter Block lesen** m = Anzahl zu lesender Parameterbyte t = Task-Nummer p = Nummer des ersten zu lesenden Parameters
4ch 01h m+4L m+4H b1...bm	Antwort: m = Anzahl gelesener Parameterbyte b = Data

[1] Mit t = 0 im Mini-OsX enthalten

Code/Data	Funktion
5ch 05h tL tH pL pH b	**Parameter Byte schreiben** t = Task-Nummer p = Nummer des zu schreibenden Parameters b = Parameter Byte
5ch 02h	Antwort
5dh 06h tL tH pL pH wL wH	**Parameter Wort schreiben** t = Task-Nummer p = Nummer des zu schreibenden Parameters w = Parameter Wort
5dh 02h	Antwort
5eh 08h tL tH pL pH dE dF dG dH	**Parameter Doppelwort schreiben** t = Task-Nummer p = Nummer des zu schreibenden Parameters d = Parameter Doppelwort
5eh 02h	Antwort
4dh 0fh n+6L n+6H tL tH pL pH b1...bn	**Parameter Block schreiben** n = Anzahl zu schreibender Parameterbyte t = Task-Nummer p = Nummer des ersten zu schreibenden Parameters Data
4dh 02h	Antwort

1.14.5. Zugriff auf Datenbereich

Der Datenbereich einer Task ist ein durchgängiger Speicherbereich ohne Lücken oder Überlappungen.

Alle Zugriffe auf den Datenbereich einer Task erfolgen unter Angabe eines Offsets als 32-Bit Wert zum Anfang des Datenbereichs.

Code/Data	Funktion
3eh 22h tL tH	**Anforderung: Datenbereichs-Semaphore anfordern** t = Task-Nummer des Datenbereichs
3eh 04h sL sHb	Antwort: s = 0: Semaphore bekommen, s <> 0: Semaphore nicht bekommen
3fh 02h tL tH	**Datenbereichs-Semaphore freigeben** t = Task-Nummer des Datenbereichs
3fh 02h	Antwort

Code/Data	Funktion
50h 16h tL tH oE oF oG oH	**Anforderung: Lies Datenbyte** von (Anfang Datenbereich + Offset o) t = Task-Nummer o = Offset (32 Bit)
50h 03h b	Antwort: b = Datenbyte
51h 26h tL tH oE oF oG oH	**Anforderung: Lies Datenwort** von (Anfang Datenbereich + Offset o) t = Task-Nummer o = Offset (32 Bit)
51h 04h wL wH	Antwort: w = Datenwort
52h 46h tL tH oE oF oG oH	**Anforderung: Lies Daten-Doppelwort** von (Anfang Datenbereich + Offset o) t = Task-Nummer o = Offset (32 Bit)
52h 06h dE dF dG dH	Antwort: d = Daten-Doppelwort
53h f8h mL mH tL tH oE oF oG oH	**Anforderung: Lies Datenblock** von (Anfang Datenbereich + Offset o) m = Blockgröße (0 bis 65535) t = Task-Nummer o = Offset (32 Bit)
53h 01h m+4L m+4H b1...bm	Antwort: m = Anzahl zurückgelieferter Byte Datenbyte (Blockgröße = m Byte)
54h 07h tL tH oE oF oG oH b	**Schreibe Datenbyte** an (Anfang Datenbereich + Offset o) t = Task-Nummer o = Offset (32 Bit) b = Datenbyte
54h 02h	Antwort
55h 08h tL tH oE oF oG oH wL wH	**Schreibe Datenwort** an (Anfang Datenbereich + Offset o) t = Task-Nummer o = Offset (32 Bit) w = Datenwort
55h 02h	Antwort

Code/Data	**Funktion**
56h 0ah tL tH oE oF oG oH dE dF dG dH	**Schreibe Daten-Doppelwort** an (Anfang Datenbereich + Offset o) t = Task-Nummer o = Offset (32 Bit) d = Daten-Doppelwort
56h 02h	Antwort
57h 0fh n+8L n+8H tL tH oE oF oG oH b1...bn	**Schreibe Datenblock** an (Anfang Datenbereich + Offset o) n = Anzahl zu schreibender Bytes t = Task-Nummer o = Offset (32 Bit) b = Datenbyte (Blockgröße = n Byte)
57h 02h	Antwort

1.14.6. Service-Requests (SRQs)

Sie lösen bestimmte Aktionen auf dem Empfänger aus. Sie werden wie Makro-Befehle behandelt. **Service-Requests liefern keine Antwort an den Absender zurück**. Sie liefern aber nach Aufruf von CALL_CMD trotzdem im Antwortpuffer einen Code zurück, der vom Aufrufer von CALL_CMD ausgewertet werden muß. Service-Requests haben einen CMD-Code ab 80h. Einige sind reserviert bzw. schon fest vergeben, andere können vom Anwender vergeben werden.

Code/Data	**Funktion**
cah 00h	**Service-Request Dummy Befehl:** macht nichts, dient dazu, bei X-MAX-1 und X-MAX-E die Mailbox-Semaphore zu löschen bzw. nach Aktivierung durch das Mail-Semaphoren-Lesen die CPU aus der „Attention“-Situation zu befreien.
-	Reaktion: keine
cbh	**Service-Request JUMP:** übergibt die Programm-Kontrolle an die mit CMD25 gesetzte Adresse. (Real und Protected Mode)
SLN	Slot-Layer-Nummer des Absenders. Sie muß identisch sein mit der bei CMD25 angegebenen SLN, andernfalls wird der Befehl nicht ausgeführt.
-	Reaktion: Im Antwortpuffer steht nach CALL_CMD: cbh 00h SLN Action aE aF aG aH sL* sH* Dabei ist a die mit CMD25 gesetzte physikal. Adresse und SLN die Slot-Layer-Nummer des SRQ-Absenders. Action kann 0,1 oder 3 sein. Wenn Action = 0 ist, dann wird die Reaktion nicht ausgeführt, weil entweder noch keine Adresse mit CMD25 gesetzt war oder weil SLN in CMD25 und in SRQcb nicht überein stimmten. Die Adresse a im Antwortpuffer ist dann ungültig. Wenn Action = 1, wird ein Jump im Real Mode auf Adresse a ausgeführt. Wenn Action = 3 ist, wird ein Jump im Protected Mode auf Adresse a ausgeführt. * nur im Protected Mode
cch	**Service-Request Master-List-Ready:** wird vom Konfigurations-Master an alle anderen Master geschickt. Die müssen sich dann mit CMD44 (= Funktion 14 von Task 0 (= OsX)) die Master-Liste abholen.
SLN	Slot-Layer-Nummer des Absenders
-	Reaktion: X-Bus Locking aufheben und mit CMD44 die Master-Liste abholen. Erst danach darf die eigene X-Bus Mailbox-Semaphore wieder gelöscht werden. Falls der Empfänger des SRQcc in ein anderes Betriebssystem als OsX geht, darf diese Semaphore erst gelöscht werden, wenn das neue Betriebssystem bereit für eine Kommunikation mit anderen Mastern ist.
cdh	**Service-Request**
-	Reaktion:

Code/Data	Funktion
ceh	**Service-Request:** (in Message MDD benutzt)
-	Reaktion:
cfh	**Service-Request: Karte wurde per Hardware-Reset zurückgesetzt** (in Windows verwendet)
-	Reaktion:

Historie dieses Dokumentes

Datum	Autor	Änderung
30.01.02	hk	AN082_F.DOC: Fehler in Beschreibung CMD24 (Hardware Reset) korrigiert
20.12.01	hk	AN082_E.DOC: Ergänzung zu CMD24 und CMD25 und SRQ´s dazu
26.10.01	hk, hb	AN082_D.DOC: Beschreibung CMD24 und CMD25 überarbeitet, Kapitel über SRQs dazu
26.6.01	hb	AN082_C.doc: Uhr komplett überarbeitet
26.6.01	hb	von hk übernommen (AN082_B.doc)