X-Bus Intensivseminar

Die Programmierung des X-Bus-Systems

SORCUS Computer GmbH
Dipl.-Ing. Jens Daneke

Inhalt

- Plattformen und Compiler
- Programmierkonzept
  - Initialisierung
  - Fehlerbehandlung
- Programmierung vom PC
  - Setup-Vorgang
  - Initialisierung
- Echtzeitprogrammierung
  - Betriebssystem OsX
  - Programm-Initialisierung
  - Funktionen und Prozeduren
  - Service-Requests
  - Ringpuffer
  - Installation von Echtzeitprogrammen

Plattformen und Compiler

Es stehen Bibliotheken und Treiber für folgende Anwendungsfälle zur Verfügung:

- **Echtzeitprogramme** unter OsX-Echtzeit-Betriebssystem auf einem X-MAX-1: Borland C 3.1, 4.5 oder 5.0, Borland Pascal 7.
- Ansprechen eines X-Bus-Systems vom **Windows-Host-PC** (DLL mit Programmierschnittstelle zu C, Delphi, Visual Basic und LabView)
- Das X-Bus-System kann
  - intelligent (Trägerkarte mit X-MAX-1) oder
  - nicht intelligent (MAX6pci ohne X-MAX-1) sein.
- Windows-Treiber (98, NT, 2000, XP) stehen zur Verfügung für
  - Kommunikation über PCI-Bus
  - Direkte serielle Ankopplung
  - Netzwerk
  - Ethernet mit TCP/IP
  - Modem-Verbindung

Programmier-Konzept 1

- Die Programmier-Schnittstelle ist identisch für
  - Echtzeit- oder Windows-Programmierung (bis auf wenige Besonderheiten)
  - für intelligente und nicht-intelligente Systeme
  - Ankopplung über PCI, seriell, Modem oder Ethernet

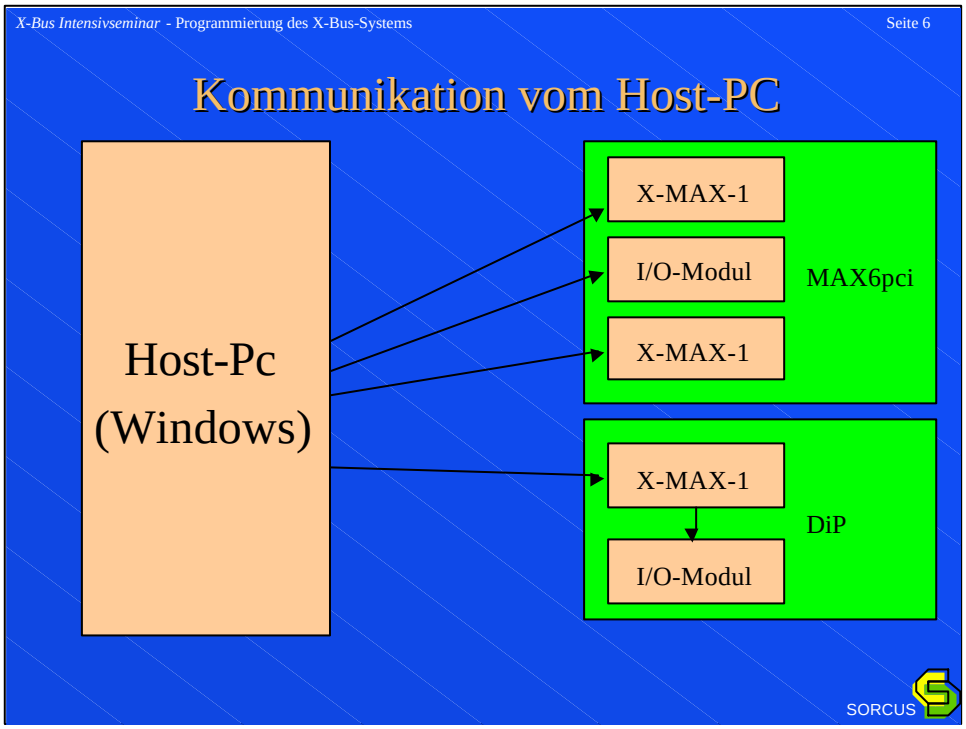
→Schnelle Konvertierung eines Windows-Programms in ein Echtzeit-Programm, wenn z.B. die Echtzeitfähigkeit von Windows nicht ausreicht

→Keine Änderungen, wenn die Applikation anstatt auf einer im PC steckenden MAX6pci auf einem externen z.B. über Ethernet angekoppelten DiP laufen soll.

→Optimale Flexibilität

Programmier-Konzept 2

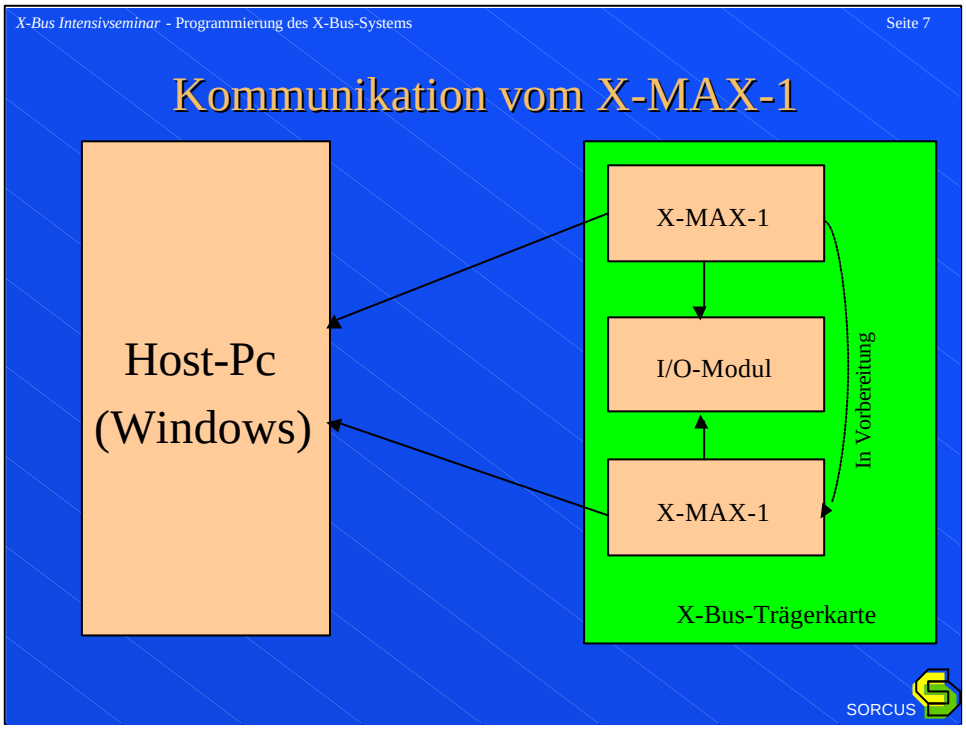
- Kanal-Orientiertes Konzept
- Um z.B. ein Modul oder eine Funktionseinheit auf einem Modul benutzen zu können, muss das Anwenderprogramm sich von der Bibliothek zunächst ein **Handle** holen.
- Mit dem Handle können dann die entsprechenden Bibliotheksfunktionen aufgerufen werden.
- Durch den Aufruf einer „Ende-Funktion“ kann das Handle ungültig gemacht und die belegten Ressourcen wieder freigegeben werden.



Von einem Windows-Programm kann über folgende Kanäle mit X-Bus-Systemen kommuniziert werden:

1. Mit CPU-Modulen tauschen die Treiber über sogenannte Makrobefehle (s. Application Note AN082) Daten aus. Dabei ist es egal, ob das Modul auf einer MAX6pci im eigenen PC steckt oder auf einer Remote angekoppelten Trägerkarte wie z.B. dem DiP. In dem Fall spielt die Art der Ankopplung ebenfalls keine Rolle. Aus einem Windows-Programm können auch mehrere CPU-Module angesprochen werden.

2. I/O-Module können auf einer MAX6pci direkt angesprochen werden. Dazu stehen für alle nicht-intelligenten Module Windows-Kernel-Mode-Treiber bereit. Auf Remote-angekoppelten Systemen läuft der Treiber für das I/O-Modul auf dem CPU-Modul, das auch die Kommunikation mit dem Host abwickelt. Diese Möglichkeit besteht auch auf einer MAX6pci mit CPU. Im Vergleich zum Zugriff über den Kernel-Mode-Treiber ist diese Variante langsamer. Aus Sicht den Anwenderprogramms unterscheiden sich beide Fälle nicht.



Echtzeitprogramme können von sich aus nur über Kurznachrichten (SRQs siehe S.28) andere CPUs oder Host-PCs über Ereignisse informieren.

Eine Erweiterung ist in Vorbereitung, so dass ein CPU Modul auf Programme, die auf einem anderen CPU-Modul laufen genauso zugreifen kann, wie auf die eigenen Programme.

Auch in der Kommuniaktion mit dem Host sind zukünfig Verbesserungen vorgesehen, indem z.B. gemeinsam genutzter Speicher verwendet wird.

Programmier-Konzept: Initialisierung

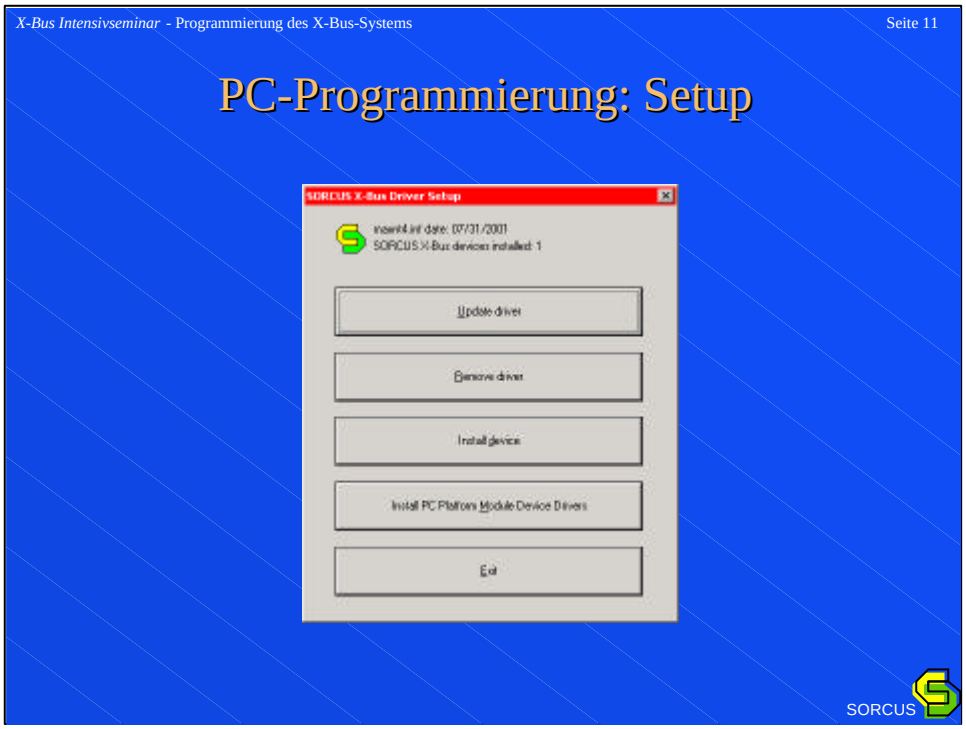
- **max_init_lib** startet die Bibliotheksbenutzung
- mit **max_connect_cpu** holt sich das Anwenderprogramm ein Handle für die CPU, die es ansprechen möchte:
- Ein **Echtzeitprogramm** muss sich ein Handle für das eigene X-MAX-1 holen, kann sich zusätzlich auch Handles für andere X-MAX-1 Module auf derselben Trägerkarte holen oder für einen angekoppelten Host-PC zum Versenden von SRQs (Kurznachrichten).
- Ein **Windows-Programm** kann sich ein Handle für ein X-MAX-1 auf einer Trägerkarte holen, mit dem es kommunizieren will. Zum Ansprechen nicht-intelligenter Systeme muss sich das Programm ein Handle für den Windows-PC selber holen.

Fehlerbehandlung

- (Fast) Alle Funktionen liefern einen Fehlercode zurück.
- Zu jedem Fehlercode gibt **max_get_error_message** eine Kurzbeschreibung zurück.
- Fehler müssen nicht nach jeder Funktion abgefragt werden. Aufgetretene Fehler werden in der Bibliothek gespeichert. Wenn beim Aufruf einer Funktion ein Fehler in der Bibliothek zwischengespeichert war, wird die eigentliche Funktionalität der Funktion nicht ausgeführt, der gespeicherte Fehler zurückgegeben.
- **max_clear_error** löscht den gespeicherten Fehler
- Performance-Verbesserung durch Unterdrückung der bibliotheks-internen Fehlerüberprüfung erreicht man auf der Echtzeitseite durch **max_set_error_check_level**.

Programmierung vom PC

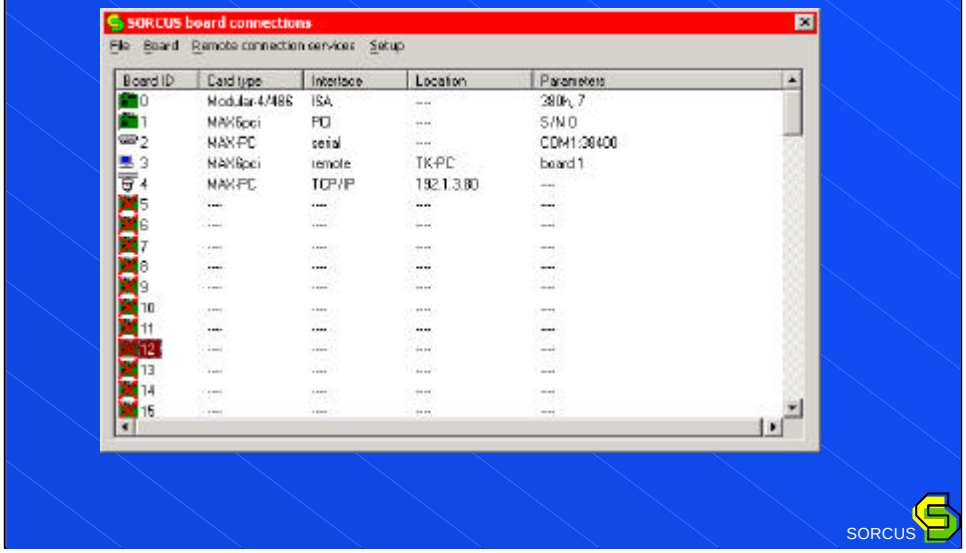
PC-Programmierung: Setup



Unter Win NT sind die Treiber mit dem oben abgebildeten Setup-Programm zu installieren.

Unter allen Plug and Play Betriebssystemen (Win98, Win2000, Win XP) werden Board und aufgesteckte Module selbstständig erkannt und die entsprechenden Treiber Plug and Play konform installiert. Das Setup-Programm wird nur zu Update-Zwecken benötigt.

Ergebnis der Karteninstallation in der Systemsteuerung

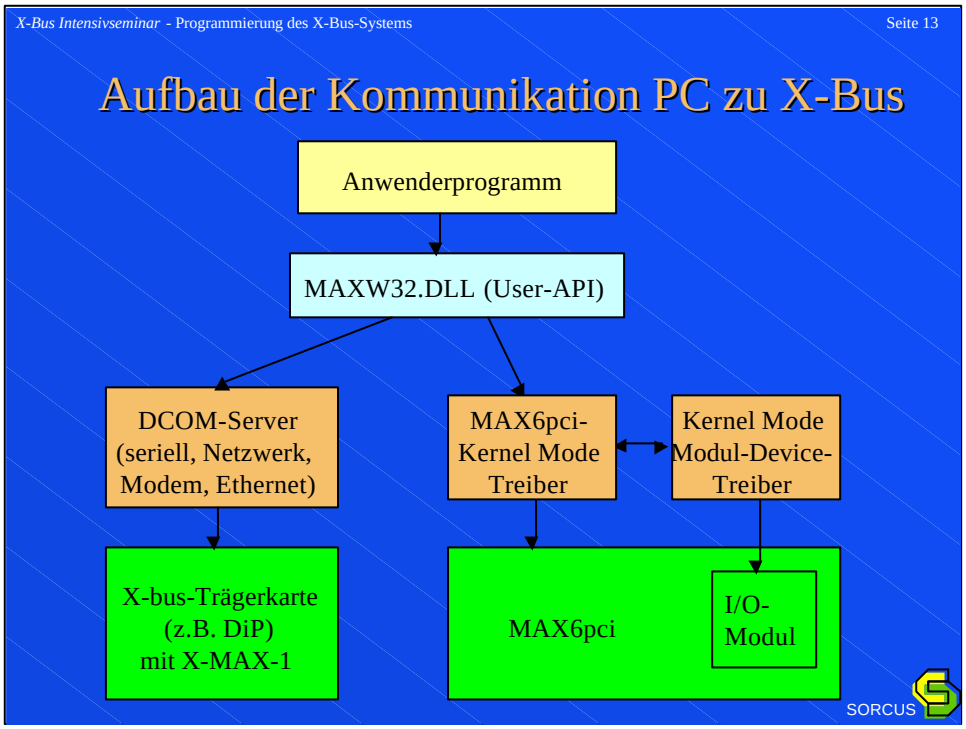


Als Ergebnis des Setup-Vorgangs erscheint in der Systemsteuerung der Punkt SORCUS Boards.

Darin finden sich die lokal im PC einsteckenden Karten (im obigen Beispiel eine MAX6pci mit BoardID 1). Auch andere SORCUS Systeme wie z.B. die MODULAR-4/486 werden dort aufgelistet.

Remote-angeschlossene Systeme werden über dieses Programm hinzugefügt bzw. konfiguriert. Dazu ist eine freie Board ID zu wählen. Per rechtem Mausklick kann nun ein neues System hinzugefügt werden. Die Konfiguration geschieht dialoggesteuert. Im obigen System sind 4 unterschiedlich angekoppelte X-Bus-Systeme als Board ID konfiguriert.

Die Board ID ist in allen PC-Anwenderprogrammen anzugeben um zu spezifizieren, welches Zielsystem angesprochen werden soll.



Zum besseren Verständnis ist hier der interne Aufbau der Treiberprogramme dargestellt:

Das Anwenderprogramm sieht nur die MAXW32 DLL mit den Bibliotheksfunktionen.

Die DLL entscheidet anhand der Board ID wie das Zielsystem anzusprechen ist:

- für die Kommunikation mit einer MAX6pci wird ein Dienst im Kernel-Mode Treiber aufgerufen

- für die Kommunikation mit einem Remote-angekoppelten System wird ein Dienst im MLXSERV DCOM Server aufgerufen, der für alle Remote-Ankopplungsarten die Kommunikation abwickelt.

Bei Verwendung der PC-basierten Modul-Device-Treiber für I/O-Module verzweigt der MAX6pci-Kernel-Mode-Treiber weiter in den entsprechenden Modul-spezifischen Treiber.

Hier ist eine typische Initialisierung eines X-Bus-Systems aus einem PC-Programm dargestellt:

1. Initialisierung der Bibliothek
2. Reset des Zielsystems
3. Abholen eines Handles für die Ziel CPU

Anschliessend ist die Funktionen zum Beenden der Kommunikation mit einer CPU sowie die Bibliotheks-Deinitialisierung gezeigt.

Echtzeitprogrammierung

OsX-Eigenschaften

- Multitasking-Betriebssystem mit bis zu 1024 Tasks
- Interrupt-, Nicht-Interrupt- und Timer-gesteuerte Tasks
- Möglichkeiten der Inter-Task-Kommunikation:
  - Datenaustausch über Daten- oder Parameterbereich (Speicherbereiche, die einer Task zugeordnet sind; Zugriffe sind durch Semaphoren schützbar)
  - Aufruf von Prozeduren oder Funktionen
- Weitere Features
  - Ringpuffer
  - Support der wichtigsten Einheiten auf dem CPU-Modul (Timer, Uhr, RAM, Flash, Interrupts)
  - weitere Einheiten werden durch separate Treiber-Tasks unterstützt (PCMCIA, Display, Tastatur, Druckerport usw.)

SORCUS

Die Hauptprozedur einer Interrupt-Task wird beim Auftreten des Interrupts aufgerufen.

Die Hauptprozedur einer Nicht-Interrupt-Task wird vom Betriebsystem immer dann aufgerufen, wenn gerade keine Interrupts bearbeitet werden müssen. Bei mehreren NI-Tasks werden diese in der Reihenfolge ihrer Aktivierung aufgerufen.

Die Hauptprozedur einer TI-Task wird nach einem beim Aktivieren angegebenen Zeitplan aufgerufen.

Besonderheiten bei der Programmierung von Echtzeitprogrammen

- **Program Descriptor Tabelle** mit den gewünschten Eigenschaften des Programms als globale Programmvariable (enthält Programm-Nr. und -Typ, Interrupt, Adressen der globalen Prozeduren usw.)
- Globale (d.h. durch andere Tasks aufrufbare) Funktionen und Prozeduren werden durch entry- und exit-Routinen eingerahmt
- Prozedur 0 einer Task wird durch das Betriebssystem aufgerufen (Interrupt-, Nicht-Interrupt- oder Timer gesteuert)
- Prozedur 1 einer Task kann durch das Betriebssystem bei der Installation aufgerufen werden

Initialisierung von Echtzeitprogrammen

- Besondere Initialisierung ist in der Hauptroutine aufzurufen

```
// Program Describtor Table muss als globale Variable im Programm enthalten sein
struct pdt_type
{
  MAX_PDT_HEAD    rcHead;        // Informationen über das Programm
  void* MAXEXPORT main_proc;     // Adresse d. Hauptprozedur
  void* MAXEXPORT auto_init;     // Adresse d. Initialisierungs-Prozedur
  // weitere Anwender-Prozeduren bzw. Funktionen
} rcPDT;

void main ()
{
  // Initialisierung der Program Describtor Table:
  InitPDT();
  // Dem OsX das Programm bekanntmachen:
  max_init_program ((MAX_PDT_HEAD*)&rcPDT);
}
```

SORCUS

Die main-Prozedur wird lediglich einmal beim Installieren durchlaufen. Darin muss die PDT initialisiert werden. Durch max_init_program werden alle PDT Informationen dem Betriebssystem bekannt gegeben. Anschlissend ist das Programm als Task installiert.

Die eigentliche Funktionalität eines Programmes liegt in seinen globalen Prozeduren und Funktionen.

Prozeduren und Funktionen

- Echtzeitprozedur

```
void MAXEXPORT MyProc ()
{
    USHORT  task;

    task = max_entry_proc ();
    ...
    max_exit_proc ();
}
```

- Aufruf

```
max_call_proc (hTargetCPU, TaskNr, ProcNr);
```

- Echtzeitfunktion

```
void MAXEXPORT MyFunc
    (USHORT *pusInSize,    void *pInData,
     USHORT *pusRetSize,  void *pRetData)
{
    USHORT  task;
    MAX_ERROR Error;

    task = max_entry_func ();
    *pusInSize = ...;       // Anzahl der  verwendeten Daten
     Error = ....;          // Fehlercode zurückgeben
    *pusRetSize = ...;      // Anzahl der  Rückgabedaten
    max_exit_func (Error);
}
```

- Aufruf

```
max_call_func (hTargetCPU, TaskNr, FuncNr,
               &usInSize, pInData, &usRetSize, pRetData);
```

SORCUS

Die Nummer der Prozeduren bzw. Funktionen sind durch die Reihenfolge der Funktionszeiger in der PDT festgelegt.

Ringpuffer - 1

- Als FIFO organisierte Speicherbereiche im RAM eines CPU-Moduls, die unter Angabe eines Handles beschrieben und gelesen werden

```
// Puffer anlegen:
MAX_BUFFER_TYPE BufferParam;
MAX_BUFHND hBuffer;

BufferParam.usStrategy = MAX_ALLOC_DOWN_ABS; // Puffer am Ende des Speichers anlegen
BufferParam.usAlignment = 2;
BufferParam.usTask = 0;
BufferParam.usUsage = 0;
ulSize = 100000;            // theoretisch bis max. der gesamte freie Speicher
Error = max_create_buffer  (hTargetCPU, &BufferParam, &ulSize, &hBuffer);
```

SORCUS

Ringpuffer sind die einfachste Möglichkeit Messdaten auf einem X-MAX-1 für die Abholung durch den Host-PC zwischenzuspeichern.

Ringpuffer - 2

- Das beim Anlegen des Puffers erhaltene Handle ist Prozess-spezifisch. Um einen Puffer in 2 Prozessen zu benutzen, muss der Prozess, der den Puffer anlegt eine Puffer-Kennung an den zweiten Prozess weiterreichen.

```
ULONG ulID;                 // Prozess-unabhängige Puffer-Kennung
Error = max_get_buffer_id (hBuffer, &ulID);       // Puffer-Kennung holen:
```

- Der zweite Prozess muss die Kennung entgegennehmen und sich seinerseits ein Handle für den Puffer holen.

```
MAX_BUFHND hMessBuffer;
Error = max_get_buffer_handle (hBuffer, ulID, &hMessBuffer);  // Puffer Handle holen
```

Ringpuffer - 3

- Zugriff mit max_get_buffer und max_set_buffer unter Angabe der zu lesenden bzw. schreibenden Datenzahl

```
// ulSize Bytes aus dem Puffer lesen
Error = max_get_buffer (hBuffer, usStrategy, &ulSize, pData);
```

- In *usStrategy* kann festgelegt werden, ob
  - genau *ulSize* Bytes gelesen werden oder maximal *ulSize* Bytes
  - Daten können aus dem Puffer kopiert oder daraus gelesen werden.
- Funktionen zum Löschen aller Puffer-Werte und zum Lesen des Puffer-Füllstands stehen zur Verfügung

Installieren von Echtzeitprogrammen 1

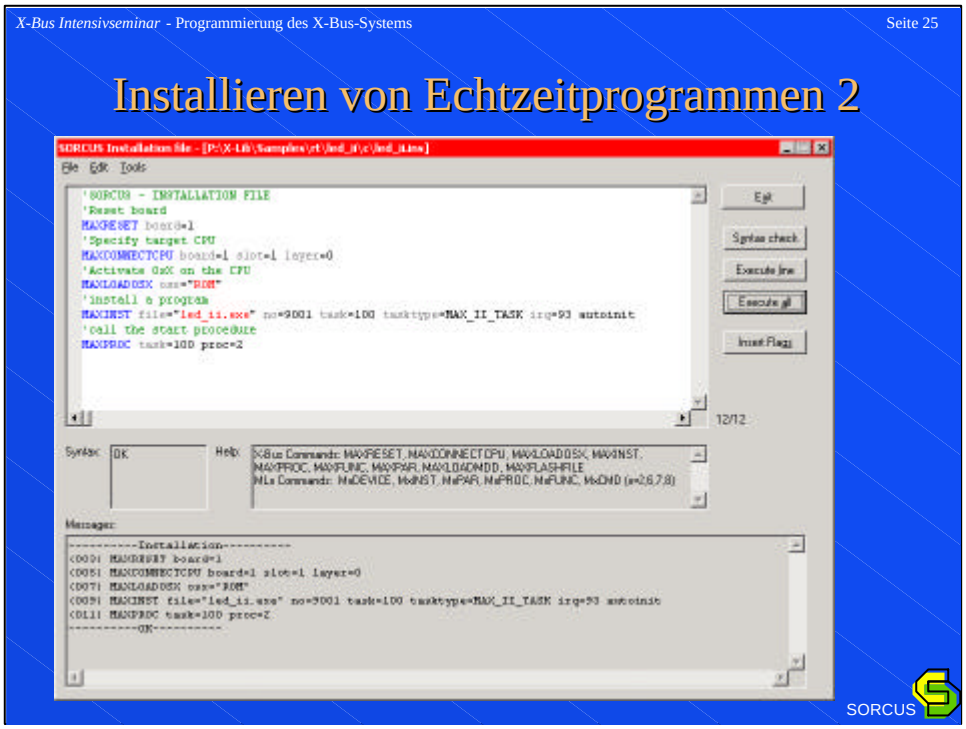
- Aus einem Windows-Programm mit dem Befehl max_transfer_and_install:

```
MAX_PGM_TYPE        PgmParam;

PgmParam.szName = „..\\rt\\Messpgm1.exe“;      // Dateiname des Programmes
PgmParam.usNo = 3000;                          // Programm Nr. entsprechend PDT
PgmParam.usUsage = 0;
PgmParam.usTask = 100;                         // gewünschte Task Nr.
PgmParam.usTaskType = MAX_II_TASK;             // Typ der Task (hier: Interrupt-Task)
PgmParam.Level = 0;
PgmParam.usIrq = MAX1_IRQ_TIMERA;              // Interrupt-Quelle (hier Timer A)
// Eigenschaften der Task :
PgmParam.usFlags = MAX_CALL_AUTO_INIT + MAX_TASKTYPE_BY_INSTALL;
PgmParam.ulDataSize = 0x10000;                 // gewünschte Größe des Datenbereiches

Error = max_transfer_and_install (hTargetCPU, &PgmParam, &TaskNr);
```

Installieren von Echtzeitprogrammen 2

- Programme können aus SNW32 mit einer INS Datei in das RAM eines X-MAX-1 geladen werden.
- In einer INS-Datei kann dafür eine Befehlssequenz definiert werden, z.B.:
  - Programme installieren
  - Parameter setzen
  - Prozeduren aufrufen

Installieren von Echtzeitprogrammen 2

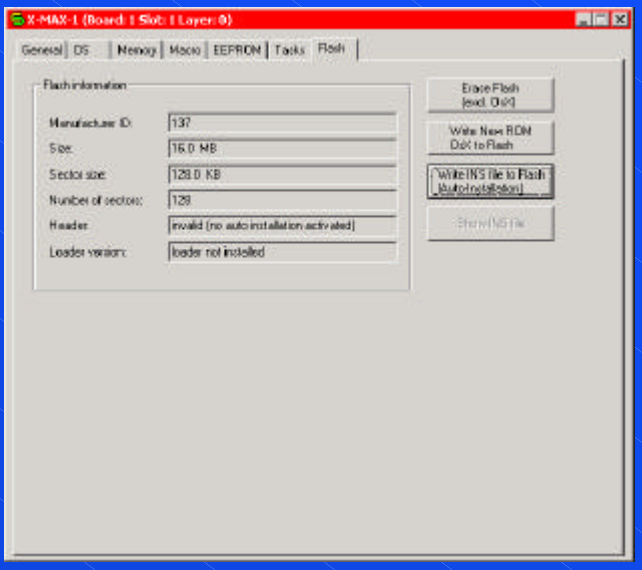


Das Bild zeigt den INS-Datei Editor im SNW32 Karten-Manager-Programm.

Installieren von Echtzeitprogrammen 3

- Automatisches Installieren eines Programmes aus dem Flash
- Echtzeitprogramme können im Flash des X-MAX-1 abgespeichert werden
- Dazu ist eine INS-Datei mit allen gewünschten Befehlen zu schreiben.
- Die INS-Datei und die zu ladenden Echtzeitprogramme können mit dem Flash Assistenten aus SNW32 ins Flash geschrieben werden.

Installieren von Echtzeitprogrammen 3



Das Bild zeigt den SNW32 Assistenten mit eine INS-Datei ins Flash geschrieben werden kann.

Zusätzlich bietet er die Möglichkeit das ebenfalls im Flash befindliche Betriebssystem zu aktualisieren.

Service-Requests (SRQ)

- Ein X-MAX-1 kann durch eine Kurznachricht (2 Byte) aktiv einen Host über besondere Ereignisse informieren.
- Dazu muss sich das Echtzeitprogramm mit **max_connect_cpu** ein Handle für den gewünschten Empfänger holen. Mit **max_send_srq** wird die Kurznachricht verschickt.
- Auf dem (Windows-) Host können sich Anwender-programme für den Empfang der SRQs anmelden. Mit **max_set_service** wird eine Anwenderfunktion benannt, die aufgerufen wird, wenn ein SRQ empfangen wird:
  MAX_CALLBACK MyService (
  MAXMODHND hSenderCPU,
  ULONG ulMessage);

SORCUS

Zu beachten: Der Aufruf der Funktion max_send_srq schickt die Kurznachricht nicht unmittelbar an den Host. Die Nachricht wird zunächst in einen Puffer geschrieben. Im OsX ist eine NI-Task für die Leerung dieses Puffers und das Absenden der Kurznachricht zuständig.

Unter Windows vergeht ebenfalls eine gewisse Zeit (die nicht vorhergesagt werden kann) vom Empfang der Nachricht im Kernel-Mode bis zum Aufruf der Anwender-Funktion.

Informationsquellen

- Handbuch ist Bibliotheks-Referenz- und Programmier-Handbuch gleichermaßen
- Software- oder Dokumentations-Updates werden unter www.sorcus.com zur Verfügung gestellt

Ende