Application Note AN068

SORCUS-Karten als komplettes Front-End für synchrone gepufferte Ein- und Ausgaben

Autor: MS an068.doc (20 Seiten)

Inhalt

Einleitung

Viele Meßaufgaben mit SORCUS-Karten bestehen darin, analoge oder digitale Signale kontinuierlich synchron abzutasten und die Meßwerte in Blöcken zum PC zu übertragen. Dieser Vorgang kann mit der Bibliothek MLXIO.DLL unter WIN32-Betriebssystemen drastisch vereinfacht werden. Die Bibliothek basiert auf dem Modul-Device-Treiber-Konzept und sorgt für die Erfassung, Zwischenpufferung und Übertragung der Meßdaten. Der Ablauf eines Meßprogramms mit MLXIO.DLL gliedert sich in folgende Schritte:

1. Bibliothek initialisieren
2. Meßkanäle definieren
3. Messung starten
4. Meßwerte lesen (bei Eingangskanälen) bzw. Meßwerte schreiben (bei Ausgangskanälen)
5. Messung stoppen

Darüber hinaus bietet MLXIO.DLL die Möglichkeit, getriggerte Messungen durchzuführen, d.h. zunächst wird ein festgelegtes Signal solange überprüft, bis die definierte Triggerbedingung auftritt. Dieser Vorgang läuft ausschließlich auf der CPU der SORCUS-

Karte ab, d.h. der PC wird damit nicht belastet. Nach Auftreten der Triggerbedingung wird die Übertragung von Meßdaten zum PC gestartet.

Programmier-Umgebung

Die Einbindung von MLXIO.DLL ist derzeit möglich in:
- Microsoft Visual C++ ab Version 4.2
- LabView
- Microsoft Visual Basic ab Version 5

Die Einbindung in Borland C++ und Delphi folgt in Kürze.

Systemvoraussetzungen

Die Bibliothek MLXIO.DLL läuft auf Windows 95/98 und Windows NT. Die SORCUS-Treiber für Multi-LAB/2 bzw. Modular-4/486 müssen installiert sein.
Weiterhin werden folgende Dateien benötigt:
- Modul Device Treiber für die verwendeten SPB-Module. Es empfiehlt sich, das Administrationstool SNW32.EXE zu installieren, da dieses die Modul Device Treiber installiert und Informationen darüber in der Registry anlegt, welche von MLXIO.DLL ausgelesen werden können.
- MLXW32.LIB, MLXIO.LIB, MLXW32.H und MLXIO.H für die Programmierung unter Microsoft Visual C++
- MLXW32.BAS und MLXIO.BAS für die Programmierung unter Visual Basic
- M-PC-IN.LIB, M-PC-OUT.LIB, TIMECTRL.LIB, TRIGGER.LIB bei Einsatz der Modular-4/486. Diese Dateien sollten sich in einem der folgenden Verzeichnisse befinden, um von MLXIO.DLL gefunden zu werden:
  1. Im MDD-Verzeichnis von SNW32.EXE: Dieser Pfad wird im Menü Options-MDD eingestellt. Beachten Sie, daß hier für jeden Kartentyp ein unterschiedlicher Pfadname eingestellt sein kann!
  2. In dem Verzeichnis, aus dem MLXIO.DLL geladen wird
  3. In einem beliebigen Verzeichnis, wenn dieses mit *mlxio_set_driver_path* eingestellt wird.
- PCINML2.LIB, PCOUTML2.LIB, TMCTML2.LIB, TRIGML2.LIB bei Einsatz der Multi-LAB/2. Für den Suchpfad gelten die zuvor beschriebenen Regeln. Beachten Sie bitte, daß für die Multi-LAB/2 mindestens die Version 7.B.004 von MLXW32.DLL erforderlich ist!

Beschreibung der Funktionen

Initialisierung

mlxio_set_driver_path

Funktion **mlxio_set_driver_path** definiert den Pfad, in dem MLXIO.DLL Echtzeitprogramme für die SORCUS-Karte sucht. Der Aufruf dieser Funktion ist optional. Falls diese Funktion nicht vor *mlxio_initialize* aufgerufen wird, gelten folgende Suchpfade:

1. Der Pfad, der in der Registry unter *HKEY_CURRENT_USER\SOFTWARE\SORCUS\SNW32\ML?\mdd_path* abgelegt ist. Dabei steht das Fragezeichen für den Kartentyp (2 = Multi-LAB/2, 6 = Multi-COM, 7 = kleine Modular-4/486, 8 = große Modular-4/486). Diese Einträge werden von SNW32.EXE vorgenommen.
2. Der Pfad, von dem MLXIO.DLL geladen wurde.

PASCAL
```
PROCEDURE mlxio_set_driver_path
(VAR pacPath: STRING);
```

C
```
void mlxio_set_driver_path
(CHAR *pacPath);
```

Parameter *pacPath* Suchpfad

Rückgabe Keine

mlxio_initialize

Funktion initialisiert die Bibliothek MLXIO.DLL.

Hinweis Vor dem Aufruf dieser Funktion muß die Bibliothek MLXW32.DLL initialisiert und die Kommunikation mit den benutzten SORCUS-Karten gestartet werden. Verwenden Sie dazu die Funktionen *mlx_bib_startup*[1] und *mlx_start/mlx_reset*.

[1] Buchstabe ‚x' in mlx... steht für den Kartentyp: 2 (Multi-LAB/2), 7 (kleine Modular-4/486), 8 (große Modular-4/486).

PASCAL FUNCTION mlxio_initialize
(dFreq: DOUBLE; ulBufSize, ulMode: ULONG): ULONG;

C ULONG mlxio_initialize
(DOUBLE dFreq, ULONG ulBufSize, ULONG ulMode);

Parameter

dFreq Basis-Abtasttakt in Hz. Die Abtastraten der Meßsignale werden als ganzzahlige Teiler dieses Taktes angegeben.

ulBufSize zeitliches Fassungsvermögen des Zwischenpuffers auf der PC-Seite. Beispiel: bei einem Basistakt von 1000 Hz bedeutet *ulBufSize*=2000 ein Fassungsvermögen von *ulBufSize/dFreq* = 2 Sekunden. *ulBufSize* entspricht der max. Anzahl der Werte im Zwischenpuffer, wenn das Signal mit dem Basisabtasttakt (und nicht mit einem Teiler davon) abgetastet wird.

ulMode bitweise ODER-Verknüpfung folgender Werte:

MLXIO_CONTINUOUS:
Meßdaten werden im Laufe der Messung blockweise übertragen.

MLXIO_BLOCK:
Während der Messung werden keine Daten zwischen PC und SORCUS-Karte ausgetauscht. Die Messung endet automatisch, wenn der Zwischenpuffer voll ist, d.h. die Messdauer beträgt *ulBufSize/dFreq.*

MLXIO_INPUT_FLEXIBLE:
Muß angegeben werden, wenn Eingangskanäle mit verschiedenen Abtastraten verwendet werden sollen.

MLXIO_INPUT_FAST:
Kann angegeben werden, wenn alle verwendeten Eingangskanäle die gleiche Abtastfrequenz haben.

MLXIO_OUTPUT_FLEXIBLE:
Muß angegeben werden, wenn Ausgangskanäle mit verschiedenen Abtastraten verwendet werden sollen.

MLXIO_OUTPUT_FAST:
Kann angegeben werden, wenn alle verwendeten Ausgangskanäle die gleiche Abtastfrequenz haben.

Rückgabe

0: kein Fehler

Sonst: Fehlernummer, siehe „Beschreibung der Fehlercodes“

mlxio_set_trigger_mode

Funktion	Diese Funktion wird nur benötigt, wenn Triggerbedingungen zum Starten/Stoppen des Meßvorgangs verwendet werden sollen. Die Funktion definiert Prä- und Posttriggerzeiten und den Wirkungsbereich von Triggerereignissen.	
PASCAL	PROCEDURE mlxio_set_trigger_mode (pretrigger, posttrigger : ULONG; global : BOOL);	
C	void mlxio_set_trigger_mode (ULONG pretrigger, ULONG posttrigger, BOOL global);	
Parameter	*pretrigger*	Prätriggerzeit in Einheiten der Basisabtastrate
	posttrigger	Posttriggerzeit in Einheiten der Basisabtastrate
	global	FALSE: Triggerevent startet/stoppt Messung lokal auf dem SORCUS-Board TRUE: Triggerevent startet/stoppt Messung global auf allen verwendeten SORCUS-Boards
Rückgabe	Keine	

Öffnen von Kanälen

mlxio_open_input_channel

Funktion	Diese Funktion definiert einen Dateneingabekanal und legt die erforderlichen Zwischenpuffer auf der PC-Seite an. Die ersten 3 Parameter entsprechen der SORCUS-Bibliotheksfunktion *mdd8_open_channel.* Im Gegensatz zu *mdd8_open_channel* wird hier der erforderliche Modul-Device-Treiber automatisch geladen, falls er nicht schon auf der Karte installiert ist.	
PASCAL	FUNCTION mlxio_open_input_channel (usMDD, usStrucSize : USHORT; VAR pCPS; usRate : USHORT) : HMLXIO;	
C	HMLXIO mlxio_open_input_channel (USHORT usMDD, USHORT usStrucSize, void *pCPS, USHORT usRate);	
Parameter	*usMDD*	Tasknummer des Modul-Device-Treibers = Steckplatznummer des SPB-Moduls. Für Devices auf der Basiskarte (z.B. bei der Multi-LAB/2) muß 0 angegeben werden.
	usStrucSize	Größe der CPS-Struktur in Bytes
	pCPS	Zeiger auf die CPS-Struktur. Der Aufbau der CPS-Struktur ist abhängig von der verwendeten Hardware
	usRate	Definiert die Abtastrate des Signals: 1 = Basisabtasttakt (siehe mlxio_initialize) n = ganzzahliger Teiler des Basisabtasttakts
Rückgabe	Handle des Dateneingabekanals oder 0 im Fehlerfall. Der Fehlercode kann mit *mlxio_get_previous_error* ermittelt werden.	

Hinweis	Modul-Device-Treiber und die Beschreibungen dazu sind auf der SORCUS-Homepage www.sorcus.com erhältlich.

mlxio_open_output_channel

Funktion	Diese Funktion definiert einen Datenausgabekanal und legt die erforderlichen Zwischenpuffer auf der PC-Seite an. Die ersten 3 Parameter entsprechen der SORCUS-Bibliotheksfunktion *mdd8_open_channel.* Im Gegensatz zu *mdd8_open_channel* wird hier der erforderliche Modul-Device-Treiber automatisch geladen, falls er nicht schon auf der Karte installiert ist.	
PASCAL	FUNCTION mlxio_open_output_channel (usMDD, usStrucSize : USHORT; VAR pCPS; usRate : USHORT) : HMLXIO;	
C	HMLXIO mlxio_open_output_channel (USHORT usMDD, USHORT usStrucSize, void *pCPS, USHORT usRate);	
Parameter	*usMDD*	Tasknummer des Modul-Device-Treibers = Steckplatznummer des SPB-Moduls. Für Devices auf der Basiskarte (z.B. bei der Multi-LAB/2) muß 0 angegeben werden.
	usStrucSize	Größe der CPS-Struktur in Bytes
	pCPS	Zeiger auf die CPS-Struktur. Der Aufbau der CPS-Struktur ist abhängig von der verwendeten Hardware
	usRate	Definiert die Abtastrate des Signals: 1 = Basisabtasttakt (siehe mlxio_initialize) n = ganzzahliger Teiler des Basisabtasttakts
Rückgabe	Handle des Datenausgabekanals oder 0 im Fehlerfall. Der Fehlercode kann mit *mlxio_get_previous_error* ermittelt werden.	

Hinweis	Modul-Device-Treiber und die Beschreibungen dazu sind auf der SORCUS-Homepage www.sorcus.com erhältlich.

mlxio_define_limit_trigger

Funktion	Definiert eine Triggerbedingung für ein analoges Eingangssignal	
PASCAL	FUNCTION mlxio_define_limit_trigger (VAR hSignal; action, mode : USHORT; limit : SHORT; min_time : USHORT; hysteresis : SHORT) : HMLXTRIG;	
C	HMLXTRIG mlxio_define_limit_trigger (void *hSignal, USHORT action, USHORT mode, SHORT limit, USHORT min_time, SHORT hysteresis);	
Parameter	*hSignal*	Handle des Triggersignals. Es kann ein Rückgabewert von *mlxio_open_input_channel* oder ein Rückgabewert von *mdd8_open_channel* verwendet werden. Im letzteren Fall werden die Meßwerte des Triggerkanals nur lokal auf dem SORCUS-Board verarbeitet und nicht zum PC übertragen.
	action	TRIGACT_START: Triggerereignis startet die Meßdatenerfassung TRIGACT_STOP: Triggerereignis stoppt die Meßdatenerfassung
	mode	TRIGMODE_LIMIT_GT: Triggerbedingung ist erfüllt, wenn der Meßwert größer als der Wert *limit* ist TRIGMODE_LIMIT_LT: Triggerbedingung ist erfüllt, wenn der Meßwert kleiner als der Wert *limit* ist.
	limit	Vergleichswert zur Überprüfung der Triggerbedingung
	min_time	Wenn ein Wert != 0 angegeben wird, wird das Triggerereignis erst dann ausgelöst, wenn die Triggerbedingung bei *min_time* aufeinanderfolgenden Abtastungen erfüllt ist. Der Wert 0 löst das Triggerereignis sofort aus, sobald die Triggerbedingung erfüllt ist.
	hysteresis	Hysteresebedingung, die erfüllt sein muß, bevor ein folgendes Triggerereignis ausgelöst werden kann. Wenn z.B. *mode = TRIGMODE_LIMIT_GT* angegeben wurde und *hysteresis* kleiner *limit* ist, muß nach Eintreten eines Triggerereignisses der Meßwert zunächst *hysteresis* unterschreiten, bevor das nächste Ereignis ausgelöst werden kann.
Rückgabe	0: Triggerobjekt konnte nicht erzeugt werden. Der Fehlercode kann mit *mlxio_get_previous_error* ermittelt werden. != 0: Handle des Triggerobjekts	

mlxio_define_bitmask_trigger

Funktion: definiert eine Triggerbedingung für digitale Eingangssignale

PASCAL:
```
FUNCTION mlxio_define_bitmask_trigger
(VAR hSignal; action, mode, bitmask, min_time : USHORT) :
HMLXTRIG;
```

C:
```
HMLXTRIG mlxio_define_bitmask_trigger
(void *hSignal, USHORT action, USHORT mode, USHORT bitmask,
USHORT min_time);
```

Parameter

hSignal — Handle des Triggersignals.
Es kann ein Rückgabewert von *mlxio_open_input_channel* oder ein Rückgabewert von *mdd8_open_channel* verwendet werden. Im letzteren Fall werden die Meßwerte des Triggerkanals nur lokal auf dem SORCUS-Board verarbeitet und nicht zum PC übertragen.

action — TRIGACT_START: Triggerereignis startet die Meßdatenerfassung
TRIGACT_STOP: Triggerereignis stoppt die Meßdatenerfassung

mode — TRIGMODE_ONE_BIT_CLR:
Triggerereignis wird ausgelöst, wenn mindestens eines der in *bit_mask* spezifizierten Bits gelöscht ist.

TRIGMODE_ALL_BITS_CLR:
Triggerereignis wird ausgelöst, wenn alle in *bit_mask* spezifizierten Bits gelöscht sind.

TRIGMODE_ONE_BIT_SET:
Triggerereignis wird ausgelöst, wenn mindestens eines der in *bit_mask* spezifizierten Bits gesetzt ist.

TRIGMODE_ALL_BITS_SET:
Triggerereignis wird ausgelöst, wenn alle in *bit_mask* spezifizierten Bits gesetzt sind.

bit_mask — Vergleichs-Bitmaske. Zu prüfende Bits müssen auf 1 gesetzt werden.

min_time — Wenn ein Wert != 0 angegeben wird, wird das Triggerereignis erst dann ausgelöst, wenn die Triggerbedingung bei *min_time* aufeinanderfolgenden Abtastungen erfüllt ist. Der Wert 0 löst das Triggerereignis sofort aus, sobald die Triggerbedingung erfüllt ist.

Rückgabe 0: Triggerobjekt konnte nicht erzeugt werden. Der Fehlercode kann mit *mlxio_get_previous_error* ermittelt werden.
!= 0: Handle des Triggerobjekts

mlxio_open_sample_trigger

Funktion Öffnet einen Kanal zur Abtast-Triggerung per Software. Dieser Kanal kann z.B. beim Modul M-SH12-8 verwendet werden, wenn die zeitgleiche Abtastung aller Kanäle des Moduls nicht durch eine Hardware-Bedingung ausgelöst wird. Die Triggerung wird zu Beginn des Abtastzyklus durchgeführt.

PASCAL
```
FUNCTION mlxio_open_sample_trigger
(usMDD, usStrucSize : USHORT; VAR pCPS; usRate : USHORT;
ulTrigVal : ULONG) : HSAMPTRIG;
```

C
```
HSAMPTRIG mlxio_open_sample_trigger
(USHORT usMDD, USHORT usStrucSize, void *pCPS, USHORT
usRate, ULONG ulTrigVal);
```

Parameter

usMDD	Steckplatznummer des zugehörigen SPB-Moduls
usStrucSize	Größe der CPS-Struktur in Bytes
pCPS	Zeiger auf die CPS-Struktur, die den Software-Abtasttriggerkanal beschreibt. Siehe Dokumentation der Modul-Device-Treiber.
usRate	Definiert die Trigger-Rate: 1 = Basisabtasttakt (siehe mlxio_initialize) n = ganzzahliger Teiler des Basisabtasttakts

Rückgabe Handle des Abtasttriggerkanals oder 0 im Fehlerfall. Der Fehlercode kann mit *mlxio_get_previous_error* ermittelt werden.

Vorbereitung der Messung

mlxio_set_private_data

Funktion	speichert einen beliebigen 32-bit-Wert mit den Kanaldaten. Damit kann der Kanal z.B. in Callback-Funktionen identifiziert werden.	
PASCAL	PROCEDURE mlxio_set_private_data (hmlxio : HMLXIO; ulPrivate : ULONG);	
C	void mlxio_set_private_data (HMLXIO hmlxio, ULONG ulPrivate);	
Parameter	*hmlxio*	Handle eines Datenein- oder –ausgabekanals
	ulPrivate	beliebiger 32-bit-Wert zur Identifikation des Kanals
Rückgabe	Keine	

mlxio_get_private_data

Funktion	gibt den zuvor mit *mlxio_set_private_data* gespeicherten 32-bit-Wert zurück	
PASCAL	FUNCTION mlxio_get_private_data (hmlxio : HMLXIO) : ULONG;	
C	ULONG mlxio_get_private_data (HMLXIO hmlxio);	
Parameter	*hmlxio*	Handle eines Datenein- oder –ausgabekanals
Rückgabe	zuvor gespeicherter 32-bit-Wert	

mlxio_get_data_size

Funktion	gibt die Anzahl Bytes pro Abtastzyklus für den gewählten Kanal zurück. Das Format der Datenbytes ist abhängig vom Typ des Kanals. Beachten Sie hierzu die Beschreibung der Modul-Device-Treiber.	
PASCAL	FUNCTION mlxio_get_data_size (hmlxio : HMLXIO) : USHORT;	
C	USHORT mlxio_get_data_size (HMLXIO hmlxio);	
Parameter	*hmlxio*	Handle eines Datenein- oder –ausgabekanals
Rückgabe	Anzahl der Bytes pro Abtastzyklus	

mlxio_get_mdd_handle

Funktion	gibt den Handle des MDD-Kanals zurück, der zum gewählten Kanal gehört. Dieser Handle kann verwendet werden, um die MDD-Diagnosefunktionen aufzurufen (z.B. Rücklesen der CPS-Struktur, Setzen/Lesen von Captions, siehe Beschreibung der Modul Device Treiber)
PASCAL	FUNCTION mlxio_get_mdd_handle (hmlxio : HMLXIO) : HMDD8;
C	HMDD8 mlxio_get_mdd_handle (HMLXIO hmlxio);
Parameter	*hmlxio* Handle des Datenein- oder –ausgabekanals
Rückgabe	Handle des zugehörigen MDD-Kanals

mlxio_set_chan_callback

Funktion

Mit dieser optionalen Prozedur kann eine kanalspezifische Rückruffunktion eingerichtet werden, die aufgerufen wird, wenn

- Meßdaten von Eingangskanälen zur Verfügung stehen oder
- Zwischenpuffer für Ausgangskanäle gefüllt werden müssen.

Alternativ zur Programmierung von Callback-Routinen kann auch der Zustand der Kanäle mit *mlxio_get_channel_state* gepollt werden. Für LabView kann nur die Polling-Methode verwendet werden, da dort kein Callback vorgesehen ist.

PASCAL

```
PROCEDURE mlxio_set_chan_callback
(hmlxio : HMLXIO; pfnCallback : POINTER);
```

C

```
void mlxio_set_chan_callback
(HMLXIO hmlxio, MLXIOCHANCALLBACK pfnCallback);
```

Parameter

hmlxio Handle des Ein- oder Ausgangskanals (= Rückgabewert von mlxio_open_input_channel oder mlxio_open_output_channel)

pfnCallback Zeiger auf die Callback-Routine. Die Callback-Routine hat keinen Rückgabewert und erhält als Übergabeparameter den Handle *hmlxio*

Rückgabe

Keine

mlxio_set_start_callback

Funktion

Mit dieser optionalen Prozedur kann eine Rückruffunktion eingerichtet werden, die aufgerufen wird, sobald die Meßdatenerfassung tatsächlich beginnt. Dieser Mechanismus kommt gewöhnlich zum Einsatz, wenn Triggerbedingungen für den Start der Messung vereinbart werden.

Alternativ zur Programmierung von Callback-Routinen kann der Zustand der Meßdatenerfassung auch mit *mlxio_get_acquisition_state* per Polling abgefragt werden. Für LabView kann nur die Polling-Methode verwendet werden, da dort kein Callback vorgesehen ist.

PASCAL

```
PROCEDURE mlxio_set_start_callback
(pfnCallback : POINTER);
```

C

```
void mlxio_set_start_callback (MLXIOSTARTCALLBACK
pfnCallback);
```

Parameter

pfnCallback Zeiger auf die Callback-Routine. Die Callback-Routine hat keinen Rückgabewert und erhält als Übergabeparameter die Nummer der betroffenden Karte als USHORT-Wert.

Rückgabe Keine

mlxio_set_stop_callback

Funktion Mit dieser optionalen Prozedur kann eine Rückruffunktion eingerichtet werden, die aufgerufen wird, sobald die Meßdatenerfassung beendet wird. Dieser Mechanismus kommt gewöhnlich zum Einsatz, wenn Triggerbedingungen für das Ende der Messung vereinbart werden.

Alternativ zur Programmierung von Callback-Routinen kann der Zustand der Meßdatenerfassung auch mit *mlxio_get_acquisition_state* per Polling abgefragt werden. Für LabView kann nur die Polling-Methode verwendet werden, da dort kein Callback vorgesehen ist.

PASCAL
```
PROCEDURE mlxio_set_stop_callback
(pfnCallback : POINTER);
```

C
```
void mlxio_set_stop_callback (MLXIOSTOPCALLBACK pfnCallback);
```

Parameter *pfnCallback* Zeiger auf die Callback-Routine. Die Callback-Routine hat keinen Rückgabewert und erhält als Übergabeparameter die Nummer der betroffenden Karte als USHORT-Wert.

Rückgabe Keine

mlxio_set_error_callback

Funktion Mit dieser optionalen Prozedur kann eine Rückruffunktion eingerichtet werden, die aufgerufen wird, sobald die Meßdatenerfassung durch eine Fehlermeldung der Karte beendet wird. Dies kann passieren, wenn die Datenpuffer auf der SORCUS-Karte nicht schnell genug vom PC bedient werden.

Alternativ zur Programmierung von Callback-Routinen kann der Zustand der Meßdatenerfassung auch mit *mlxio_get_acquisition_state* per Polling abgefragt werden. Für LabView kann nur die Polling-Methode verwendet werden, da dort kein Callback vorgesehen ist.

PASCAL
```
PROCEDURE mlxio_set_error_callback
(pfnCallback : POINTER);
```

C
```
void mlxio_set_error_callback (MLXIOERRORCALLBACK
pfnCallback);
```

Parameter *pfnCallback* Zeiger auf die Callback-Routine. Die Callback-Routine hat keinen Rückgabewert und erhält als Übergabeparameter die Nummer der betroffenden Karte als USHORT-Wert.

Rückgabe Keine

Meßbetrieb

mlxio_start_measurement

Funktion	Die Funktion installiert zunächst die erforderlichen Echtzeitprogramme auf die SORCUS-Karte. Die Meßdatenerfassung wird danach sofort gestartet, falls keine Triggerbedingung mit *action = TRIGACT_START* definiert wurde. Andernfalls wird die Triggerbedingung abgewartet.
PASCAL	FUNCTION mlxio_start_measurement : ULONG;
Rückgabe	0 oder Fehlermeldung, siehe „Beschreibung der Fehlercodes“

mlxio_stop_measurement

Funktion	Die Meßdatenerfassung wird gestoppt, unabhängig davon, ob Triggerbedingung mit *action = TRIGACT_START* definiert wurde.
PASCAL	FUNCTION mlxio_stop_measurement : ULONG;
Rückgabe	0 oder Fehlermeldung, siehe „Beschreibung der Fehlercodes“

mlxio_write_data

Funktion	Schreibt einen Block von Ausgabedaten in den PC-Zwischenpuffer für einen Ausgangskanal. Die Übertragung der Zwischenpuffer zur SORCUS-Karte erfolgt, sobald alle Ausgangs-Zwischenpuffer genügend Daten enthalten.	
PASCAL	FUNCTION mlxio_write_data (hmlxio : HMLXIO; VAR databuffer; numofbytes : ULONG; VAR error : ULONG) : ULONG;	
C	ULONG mlxio_write_data (HMLXIO hmlxio, void *databuffer, ULONG numofbytes, ULONG *error);	
Parameter	*hmlxio*	Handle des Datenausgabekanals
	databuffer	Zeiger auf den Puffer, der die Ausgabedaten enthält
	numofbytes	Größe des Puffers in Bytes
	error	Zeiger auf einen Wert, der nach dem Aufruf eine Fehlermeldung (siehe „Beschreibung der Fehlercodes“) enthalten kann. Es kann auch ein Nullpointer übergeben werden.
Rückgabe	Anzahl der Bytes, die in den Zwischenpuffer übertragen wurden.	

mlxio_read_data

Funktion	Liest einen Block von Eingabedaten aus dem PC-Zwischenpuffer eines Dateneingabekanals, sofern dort Daten bereitstehen.	
PASCAL	FUNCTION mlxio_read_data (hmlxio : HMLXIO; VAR databuffer; numofbytes : ULONG; VAR error, timebase : ULONG) : ULONG;	
C	ULONG mlxio_read_data (HMLXIO hmlxio, void *databuffer, ULONG numofbytes, ULONG *error, ULONG *timebase);	
Parameter	*hmlxio*	Handle des Dateneingabekanals
	databuffer	Zeiger auf den Puffer, in den die Eingabedaten kopiert werden sollen
	numofbytes	Größe des Puffers in Bytes
	error	Zeiger auf einen Wert, der nach dem Aufruf eine Fehlermeldung (siehe „Beschreibung der Fehlercodes“) enthalten kann. Es kann auch ein Nullpointer übergeben werden.
	timebase	Enthält nach dem Aufruf der Funktion den Zeitstempel des 1. Wertes im Puffer. Der Zeitstempel beginnt nach dem Aufruf von mlxio_start_measurement bei 0 und wird im Basisabtasttakt inkrementiert. Es kann auch ein Nullpointer übergeben werden.
Rückgabe	Anzahl der Bytes, die aus dem Zwischenpuffer kopiert wurden.	

mlxio_get_acquisition_state

Funktion	Gibt den Status der Meßdatenerfassung zurück	
PASCAL	FUNCTION mlxio_get_acquisition_state : ULONG;	
C	ULONG mlxio_get_acquisition_state (void);	
Parameter	keine	
Rückgabe	MLXIO_STOPPED	Datenerfassung beendet oder noch nicht gestartet
	MLXIO_WAITING	Warten auf Start-Triggerereignis
	MLXIO_RUNNING	Datenerfassung läuft
	MLXIO_TRIGSTOP	Datenerfassung durch Triggerereignis gestoppt
	MLXIO_BLOCKEND	Messung im Modus MLXIO_BLOCK beendet
	MLXIO_ERRORSTOP	Messung durch Fehlermeldung beendet

mlxio_get_channel_state

Funktion	Gibt den Status eines Datenerfassungskanals zurück	
PASCAL	PROCEDURE mlxio_get_channel_state (hmlxio : HMLXIO; VAR free, used, error : ULONG);	
C	void mlxio_get_channel_state (HMLXIO hmlxio, ULONG *free, ULONG *used, ULONG *error);	
Parameter	*hmlxio*	Handle des Kanals
	free	Zeiger auf einen Wert, der nach dem Aufruf die Anzahl der freien Bytes im PC-Zwischenpuffer des Kanals enthält. Falls ein Nullpointer übergeben wird, wird diese Information ignoriert.
	used	Zeiger auf einen Wert, der nach dem Aufruf die Anzahl der genutzten Bytes im PC-Zwischenpuffer des Kanals enthält. . Falls ein Nullpointer übergeben wird, wird diese Information ignoriert.
	error	Zeiger auf einen Wert, der nach dem Aufruf eine Fehlermeldung (siehe „Beschreibung der Fehlercodes“) enthalten kann. . Falls ein Nullpointer übergeben wird, wird diese Information ignoriert.
Rückgabe	keine	

Routinen zur Fehlerbehandlung

mlxio_reload

Funktion	öffnet alle konfigurierten MDD-Kanäle auf dem SORCUS-Board. Diese Funktion findet Verwendung, wenn z.B. mlxio_start_measurement nicht erfolgreich ausgeführt werden kann. In diesem Fall empfiehlt es sich, *mlx_reset* auszuführen und anschließend mit *mlxio_reload* die benötigten MDD-Kanäle neu zu konfigurieren.
PASCAL	PROCEDURE mlxio_reload;
C	void mlxio_reload (void);
Parameter	Keine
Rückgabe	Keine

mlxio_reset_error

Funktion	setzt den internen Fehlerstatus eines Ein- oder Ausgabekanals zurück. Die Routine sollte aufgerufen werden, wenn von *mlxio_write_data/mlxio_read_data* oder *mlxio_get_channel_state* ein Fehlercode zurückgegeben wurde.
PASCAL	PROCEDURE mlxio_reset_error (hmlxio : HMLXIO);
C	void mlxio_reset_error (HMLXIO hmlxio);
Parameter	*hmlxio* Handle eines Ein- oder Ausgabekanals
Rückgabe	Keine

Nachbereitung

mlxio_close_all

Funktion	gibt alle von MLXIO.DLL benutzten Resourcen auf dem PC und auf der SORCUS-Karte frei.
PASCAL	PROCEDURE mlxio_close_all;
C	void mlxio_close_all (void);
Parameter	Keine
Rückgabe	Keine

mlxio_get_previous_error

Funktion	liefert den Fehlerstatus einer zuvor gescheiterten „OPEN“-Funktion zurück.
PASCAL	FUNCTION mlxio_get_previous_error : ULONG;
C	ULONG mlxio_get_previous_error (void);
Parameter	Keine
Rückgabe	Fehlerstatus (siehe „Beschreibung der Fehlercodes“)

Beschreibung der Fehlercodes

Folgende Fehlercodes sind bisher definiert:

MLXIO_OK	Rückgabewert 0: kein Fehler
MLXIO_ERR_MDDPATH	Suchpfad für Treiber nicht korrekt (siehe „Systemvoraussetzungen“)
MLXIO_ERR_INVALID_STATE	*mlxio_initialize* wurde nicht aufgerufen
MLXIO_ERR_LOAD_DRIVER	*mlxio_reload* kann einen Modul Device Treiber nicht laden
MLXIO_ERR_MDD_OPEN	Fehler beim Öffnen eines MDD-Kanals: Parameter der CPS-Struktur fehlerhaft
MLXIO_ERR_MLXM_OPEN	Fehler beim Anlegen der Strukturen zum Datenaustausch mit der SORCUS-Karte
MLXIO_ERR_PREPARE	Fehler beim Vorbereiten der Messung, z.B. beim Laden der Echtzeitprogramme auf die Karte
MLXIO_ERR_START	Fehler beim Start der Messung, z.B. zu wenig Speicher auf der Karte
MLXIO_ERR_CHANNEL_TYPE	Falscher Kanaltyp: z.B. Handle eines Dateneingabekanals an *mlxio_write_data* übergeben
MLXIO_ERR_TRIGGERTASK	Fehler beim Installieren der Task zur Überwachung der Triggerbedingungen
MLXIO_ERR_OPEN_TRIGGER	Fehler beim Öffnen eines Triggerkanals
MLXIO_ERR_OPEN_SAMPTRIG	Fehler beim Öffnen eines Software-Abtasttriggerkanals
MLXIO_ERR_PRETRIGGER	Prätriggerzeit zu lang
MLXIO_ERR_INVALID_HANDLE	Handle ungültig
MLXIO_ERR_SAMPLE_RATE	Die gewählte Abtastrate ungültig. Im Übertragungsmodus „FAST“ wurden verschiedene Abtastraten angegeben.
MLXIO_ERR_MEMORY	Zu wenig Arbeitsspeicher (PC)
MLXIO_ERR_TOO_MUCH_OUTPUTS	Zu viele Ausgänge definiert
MLXIO_ERR_INVALID_CHANNEL	Ungültiger Kanaltyp
MLXIO_ERR_RELOAD bis MLXIO_ERR_RELOAD+15	Fehler bei *mlxio_start_measurement.* Kann beim wiederholten Starten auftreten, wenn zwischenzeitlich Konfigurationsänderungen (neue Kanäle) vorgenommen wurden. Die letzten 4 Bits der Fehlernummer beinhalten die Nummer der betroffenen Karte.
MLXIO_ERR_OPEN_INPUT bis MLXIO_ERR_OPEN_INPUT+15	Tritt bei *mlxio_start_measurement* auf, wenn das Echtzeitprogramm zur Übertragung von Eingabedaten zum PC nicht geladen oder initialisiert werden kann.
MLXIO_ERR_OPEN_OUTPUT bis MLXIO_ERR_OPEN_OUTPUT+15	Tritt bei *mlxio_start_measurement* auf, wenn das Echtzeitprogramm zur Übertragung von Ausgabedaten zur SORCUS-Karte nicht geladen oder initialisiert werden kann.

Anwendungsbeispiel

Das beiliegende Programm MLXIOBSP.EXE zeigt eine einfache Anwendung von MLXIO.DLL mit dem SPB-Modul M-5B-1/U. Der Quellcode befindet sich in der Datei MLXIOBSP.C. Das Programm initialisiert zunächst die Bibliothek, öffnet die benötigten Eingabekanäle und erfaßt Daten mit Hilfe von Callback-Funktionen. Das Beispiel wurde mit Microsoft Visual C++ als Konsolenapplikation übersetzt. Dabei wurden die Importbibliotheken MLXIO.LIB und MLXW32.LIB zum Projekt hinzugefügt.