

Programmierbare Systeme
System *HIMatrix*

***COM USER TASK
(CUT)***

Handbuch



HIMA Paul Hildebrandt GmbH
Industrie-Automatisierung

HI 800 328 CDA

Wichtige Hinweise

Alle in diesem Handbuch genannten HIMA-Produkte sind mit dem HIMA-Warenzeichen geschützt. Dies gilt gegebenenfalls, soweit nicht anders vermerkt, auch für andere genannte Hersteller und deren Produkte.

Alle technischen Angaben und Hinweise in diesem Handbuch wurden mit größter Sorgfalt erarbeitet und unter Einschaltung wirksamer Kontrollmaßnahmen zusammengestellt. Trotzdem sind Fehler nicht ganz auszuschließen.

HIMA sieht sich deshalb veranlasst, darauf hinzuweisen, dass weder eine Garantie noch die juristische Verantwortung oder irgend eine Haftung übernommen werden kann für die Folgen, die auf fehlerhafte Angaben zurückgehen. Für die Mitteilung eventueller Fehler ist HIMA dankbar.

Technische Änderungen vorbehalten.

Weitere Informationen sind in der Dokumentation auf der CD-ROM und auf unserer Website unter www.hima.de zu finden.

Informationsanfragen sind zu richten an:

HIMA Paul Hildebrandt GmbH
Postfach 1261
68777 Brühl

Tel: +49(6202)709 0
Fax: +49(6202)709 107

e-mail: info@hima.com

Inhaltsverzeichnis

1	COM User Task (CUT)	5
1.1	Voraussetzungen	5
1.1.1	ELOP II Factory	5
1.1.2	Entwicklungsumgebung	5
1.1.3	Verwendbare Steuerungen	5
1.2	Abkürzungen	6
2	CUT-Schnittstelle in ELOP II Factory	7
2.1	Schedule-Intervall [ms]	7
2.2	Scheduling-Vorbereitung	7
2.3	Scheduling-Nachbereitung	7
2.4	STOP_INVALID_CONFIG	8
2.5	Signale der CUT-Schnittstelle (CPU<->CUT)	8
2.5.1	Status	9
2.5.2	Steuerung	9
2.5.3	Daten-Eingänge (COM->CPU)	10
2.5.4	Daten-Ausgänge (CPU->COM)	11
3	CUT Funktionen	12
3.1	COM-User-Callback-Funktionen	12
3.2	COM-User-Library-Funktionen	12
3.3	Header-Files	12
3.4	Code-/Datenbereich und Stack für die CUT	12
3.5	Startfunktion CUCB_TaskLoop	13
3.6	Serielle Schnittstellen RS485 / RS232 IF	13
3.6.1	CUL_AscOpen	13
3.6.2	CUL_AscClose	15
3.6.3	CUL_AscRcv	15
3.6.4	CUCB_AscRcvReady	17
3.6.5	CUL_AscSend	18
3.6.6	CUCB_AscSendReady	19
3.7	UDP/TCP-Socket-IF	20
3.7.1	CUL_SocketOpenUdpBind	20
3.7.2	CUL_SocketOpenUdp	21
3.7.3	CUL_NetMessageAlloc	21
3.7.4	CUL_SocketSendTo	22
3.7.5	CUCB_SocketUdpRcv	23
3.7.6	CUL_NetMessageFree	23
3.7.7	CUL_SocketOpenTcpServer	24
3.7.8	CUCB_SocketTryAccept	25
3.7.9	CUL_SocketAccept	25
3.7.10	CUL_SocketOpenTcpClient	26
3.7.11	CUCB_SocketConnected	26
3.7.12	CUL_SocketSend	27
3.7.13	CUCB_SocketTcpRcv	28
3.7.14	CUL_SocketClose	28
3.8	Timer-IF	29
3.8.1	CUL_GetTimeStampMS	29
3.8.2	CUL_GetDateAndTime	29
3.9	COM-User-IRQ-Task (Nur für MHS31A)	30
3.9.1	CUCB_IrqService	30
3.9.2	CUL_IrqServiceEnable	30
3.9.3	CUL_IrqServiceDisable	31
3.9.4	CUL_DeviceBaseAddr	31
3.10	NVRam-IF (Nur für MHS31A)	32
3.10.1	CUL_NVRamWrite	32
3.10.2	CUL_NVRamRead	32
3.11	Semaphore-IF (Nur für MHS31A)	33
3.11.1	CUL_SemaRequest	33
3.11.2	CUL_SemaRelease	33

3.11.3	CUL_SemaTry.....	34
3.12	COM-IO-IF (Nur MHS31A).....	35
3.12.1	CUL_IORead.....	35
3.12.2	CUL_IOWrite.....	35
3.12.3	CUL_IOConfigure.....	36
3.13	Diagnose.....	36
4	Installation der Entwicklungsumgebung	37
4.1	Installation der Cygwin-Umgebung.....	37
4.2	Installation des GNU Compilers.....	40
5	Neues CUT-Projekt anlegen	41
5.1	CUT-Makefiles	42
5.1.1	Makefile mit der Erweiterung „mke“	42
5.1.2	Makefile	43
5.1.3	Makefile mit der Erweiterung „makeinc.inc.app“	44
5.2	C-Quellcode bearbeiten.....	45
5.2.1	Eingangs- und Ausgangssignale konfigurieren.....	45
5.2.2	Startfunktion im CUT	45
5.2.3	Beispielcode „example_cut.c“	45
5.2.4	Erstellung des Ausführbaren Code (ldb-Datei)	47
5.3	COM User Task in das Projekt einbinden.....	48
5.3.1	COM User Task anlegen.....	48
5.3.2	Programmcode in das Projekt laden	48
5.3.3	Signale mit dem CUT verbinden	49
5.3.3.1	Signale im Signaleditor erstellen	49
5.3.3.2	CUT Dialog „Signal-Zuordnungen“	49
5.3.3.3	Konfigurieren der Eingangssignale (COM->CPU)	49
5.3.3.4	Konfigurieren der Ausgangssignale (CPU->COM)	50
5.3.4	Erstellen Sie das ELOP II Factory Anwenderprogramm.....	50
5.3.5	Konfigurieren Sie das Schedule-Intervall [ms]	51
5.3.6	Erstellen und Laden des Codes in die Steuerung.....	51
5.3.7	Testen der COM User Task	51
5.4	Tipps und Tricks.....	52
5.4.1	Fehler beim Laden einer Konfiguration mit CUT.....	52

1 COM User Task (CUT)

Neben dem Anwenderprogramm, das mit ELOP II Factory erstellt wird, hat der Anwender zusätzlich die Möglichkeit ein C-Programm auf der Steuerung zu betreiben. Dieses nicht sichere C-Programm läuft als COM User Task rückwirkungsfrei zur sicheren CPU auf dem COM-Modul der Steuerung. Die COM User Task hat einen eigenen Zyklus, der unabhängig von dem Zyklus der CPU ist. Damit können beliebige Anwendungen in C erstellt und als COM User Task implementiert werden z.B.:

- Kommunikationsschnittstellen für spezielle Protokolle (TCP, UDP usw.).
- Gateway Funktion zwischen TCP/UDP und serieller Kommunikation.

1.1 Voraussetzungen

Für die Programmierung der COM User Task wird zusätzlich zum normalen Befehlsumfang von C eine eigene Library (siehe Kapitel 2) mit definierten Funktionen verwendet.

1.1.1 ELOP II Factory

ELOP II Factory verfügt ab Hardware-Management Version 8.36 über die entsprechenden Funktionen für die COM User Task.

Das Laden von Konfigurationen mit COM-User-Task erfordert eine entsprechende Lizenz.

1.1.2 Entwicklungsumgebung

Zu der Entwicklungsumgebung gehören der GNU C Compiler und Cygwin, die auf einer gesonderten Installations-CD vorhanden sind (nicht bei ELOP II Factory mit dabei) und den Bedingungen der GNU General Public License unterliegen (siehe www.gnu.org).

Die neuesten Versionen und Dokumentationen zu der Entwicklungsumgebung können Sie sich von den entsprechenden Seiten im Internet herunterladen www.cygwin.com und www.gnu.org.

1.1.3 Verwendbare Steuerungen

Eine COM-User-Task kann im Anwenderprogramm folgender Ressourcen angelegt werden:

Steuerung	CPU-BS	COM-BS
HIMatrix F20 HIMatrix (HW-Rev. 02) F30, F31, F35, F60,	Ab Version 6.44	Ab Version 11.24
MHS31A	Ab Version 6.44	Ab Version 11.24

Tabelle 1: Verwendbare Steuerungen für COM User Task

Bei den *HIMatrix* Steuerungen hat die COM User Task keinen Zugriff auf die sicheren Hardware Ein- und Ausgänge. Wenn ein Zugriff auf die Hardware Ein- und Ausgänge benötigt wird, dann ist ein Anwenderprogramm der CPU zur Verbindung der Signale erforderlich (siehe 2.5).

Bei der Steuerung MHS31A hat die COM User Task keinen Zugriff auf die sicheren Hardware Ein- und Ausgänge.

Die MHS31A besitzt zusätzlich nicht sichere Ein- und Ausgänge, auf die die COM User Task direkt zugreifen kann. Ein Anwenderprogramm der CPU zur Verbindung der Signale (siehe 2.5) ist für die nicht sicheren Ein- und Ausgänge nicht erforderlich.

1.2 Abkürzungen

Abkürzung	Bedeutung
CPU	Zentraler Prozessor der Steuerung
COM	Kommunikationsprozessor der Steuerung
CUCB	COM User Callback (CUCB_ Funktionen werden von der COM aufgerufen)
CUIT	COM User IRQ Task (Nur für MHS31A)
CUL	COM User Library (CUL_ Funktionen werden in der CUT aufgerufen)
CUT	COM User Task
GNU	GNU Projekt
IF	InterFace
FB	Feldbusschnittstelle der Steuerung
FIFO	First In First Out (Datenspeicher)
NVRam	Non Volatile Random Access Memory, nicht flüchtiger Speicher

Tabelle 2: Abkürzungen

2 CUT-Schnittstelle in ELOP II Factory

Die Prozessdatenkommunikation der COM User Task läuft zwischen der COM und der CPU ab.



Die geladene COM User Task darf keine privilegierten Befehle des COM Prozessors benutzen.

Der Code der CUT läuft auf der COM rückwirkungsfrei zur CPU. Damit ist die sichere CPU vor dem Code der CUT geschützt.

Es ist jedoch zu beachten, dass Fehler im CUT Code die Funktion der COM als Ganzes und damit auch die Funktion der Steuerung stören und sogar unterbinden können.

Die Sicherheitsfunktionen der CPU werden dadurch nicht beeinträchtigt.

2.1 Schedule-Intervall [ms]

Die COM User Task wird in einem parametrierten Schedule-Intervall[ms] in den Zuständen RUN und STOP_VALID_CONFIG der Steuerung (COM-Prozessor) aufgerufen.

Das Schedule-Intervall[ms] kann der Anwender in ELOP II Factory in den Eigenschaften der COM User Task einstellen.

Schedule-Intervall[ms]	
Wertebereich:	10 .. 255 ms
Standardwert:	15 ms

Tabelle 3: Parameter im ELOP II Factory Dialog „Eigenschaften“ der COM User Task

Hinweis Die der CUT verfügbare COM-Prozessor-Zeit hängt von den anderen parametrierten Funktionen der COM, wie z.B. SafeEthernet, Modbus-TCP etc. ab.

Wenn die CUT nicht innerhalb des Schedule-Intervalls fertig wird, dann wird jeder Aufruf zum Neustart der CUT abgewiesen, bis die CUT abgearbeitet ist.

2.2 Scheduling-Vorbereitung

Im Betriebsmode RUN der Steuerung:

Vor jedem Aufruf der CUT stellt die COM die Prozessdaten von der sicheren CPU der CUT in einem von der CUT definierten Speicherbereich zur Verfügung.

Im Betriebsmodus STOP der Steuerung:

Es findet kein Prozessdatenaustausch von der sicheren CPU zur COM statt.

2.3 Scheduling-Nachbereitung

Im Betriebsmode RUN der Steuerung:

Nach jedem Aufruf der CUT stellt die COM die Prozessdaten von der CUT der sicheren CPU zur Verfügung.

Im Betriebsmodus STOP der Steuerung:

Es findet kein Prozessdatenaustausch von der COM zur sicheren CPU statt.

2.4 STOP_INVALID_CONFIG

Befindet sich die COM Im Zustand STOP_INVALID_CONFIG, dann wird die CUT nicht ausgeführt.

Wechselt die COM in den Zustand STOP_INVALID_CONFIG und führt die CUT oder die CUIT aus, so werden diese terminiert.

2.5 Signale der CUT-Schnittstelle (CPU<->CUT)

Der Anwender kann eine nicht sicherheitsgerichtete Prozessdatenkommunikation zwischen sicherer CPU und COM (CUT) definieren.

Senderichtung	Maximale Größe der Prozessdaten
COM->CPU	16375 Bytes Daten (16384 Bytes – 9 Statusbytes)
CPU->COM	16382 Bytes Daten (16384 Bytes – 2 Steuerungsbytes)

Hinweis: Neben der Prozessdatenkommunikation können gleichzeitig noch weitere Protokolle (z.B. Modbus, Profibus-DP und Ethernet I/P) auf der Steuerung betrieben werden.
Insgesamt können 16384 Bytes Daten gesendet und 16384 Bytes Daten empfangen werden.
Die Sende- und Empfangsdaten können beliebig zwischen den Protokollen aufgeteilt werden.

Prozessdatenaustausch mit COM-User-Task (CUT)

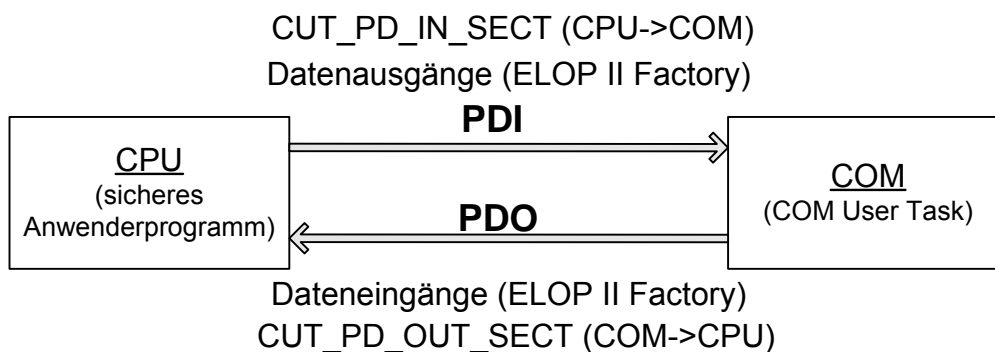


Abbildung 1: Prozessdatenaustausch zwischen CPU und COM (CUT)

Es können alle Datentypen ausgetauscht werden, die in ELOP II Factory als Signal dargestellt werden können.

Die Struktur der Daten muss in ELOP II Factory parametrisiert werden.
Die Größe der Datenstrukturen CUT_PDI und CUT_PDO (im kompilierten C-Code der CUT) müssen der gleichen Größe der konfigurierten Datenstruktur in ELOP II Factory entsprechen.

Hinweis Sind im kompilierten C-Code die Datenstrukturen CUT_PDI und CUT_PDO nicht vorhanden oder haben nicht die gleiche Größe wie die Datenstruktur in ELOP II Factory parametrisierten Prozessdaten, dann ist die Konfiguration ungültig und die COM nimmt den Zustand STOP_INVALID_CONFIG an.
Die Prozessdatenkommunikation findet nur im Betriebsmodus RUN statt.

Im Dialog *Signal-Zuordnungen* der COM User Tasks befinden sich die in den folgenden Abschnitten beschriebenen Register für den Prozessdatenaustausch.

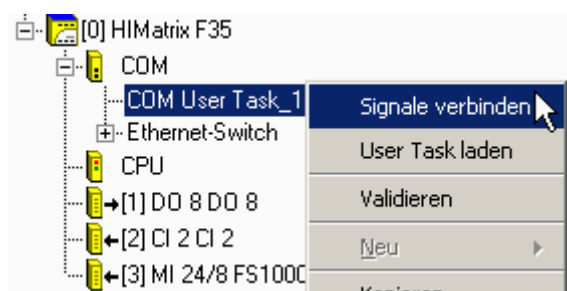


Abbildung 2: Kontextmenü der COM User Task

2.5.1 Status

Das Register **Status** enthält die folgenden Systemparameter zur Status-Überwachung der COM (CUT):

Name	Typ	Funktion
Ausführungszeit	DWORD	Ausführungszeit der COM User Task in μ s
Reales Schedule-Intervall	DWORD	Zeitabstand zwischen zwei COM User Task Durchläufen in ms
Zustand der User Task	BYTE	1 = RUNNING (CUT läuft) 0 = ERROR (CUT läuft nicht wegen eines Fehlers)

Tabelle 4: Signale im Register „Status“ der COM User Task

2.5.2 Steuerung

Das Register **Steuerung** enthält den Systemparameter *Steuerung des User Task-Zustandes*.

Die folgende Tabelle zeigt die Möglichkeiten, wie der Anwender mit dem Systemparameter *Steuerung des User Task-Zustandes* die COM User Task steuern kann:

Steuerung des User Task-Zustandes (WORD)		
Funktion	Wert	Beschreibung
DISABLED	0x8000	Das Anwenderprogramm sperrt die CUT (d.h. die CUT wird nicht gestartet).
AUTOSTART (Default)	0	Nach Terminierung der CUT startet die CUT automatisch nachdem die Störung oder der Fehler beseitigt wurde.
TOGGLE_MODE_0	0x0100	Nach Terminierung der CUT ist ein Start der CUT erst wieder nach dem Setzen von TOGGLE_MODE_1 erlaubt.
TOGGLE_MODE_1	0x0101	Nach Terminierung der CUT ist ein Start der CUT erst wieder nach dem Setzen von TOGGLE_MODE_0 erlaubt.

Tabelle 5: Signale im Register „Steuerung“ der COM User Task

2.5.3 Daten-Eingänge (COM->CPU)

In das Register **Daten-Eingänge** werden die Signale eingetragen, die von der COM (CUT) zur CPU übertragen werden sollen (Eingangsbereich der CPU). Hier werden die Datentypen und die Größe der Datenstruktur (des Eingangsbereichs der CPU) konfiguriert.



Beachten Sie, dass die nicht sicheren Signale der COM User Task die Sicherheitsfunktionen des CPU Anwenderprogramms nicht behindern dürfen.

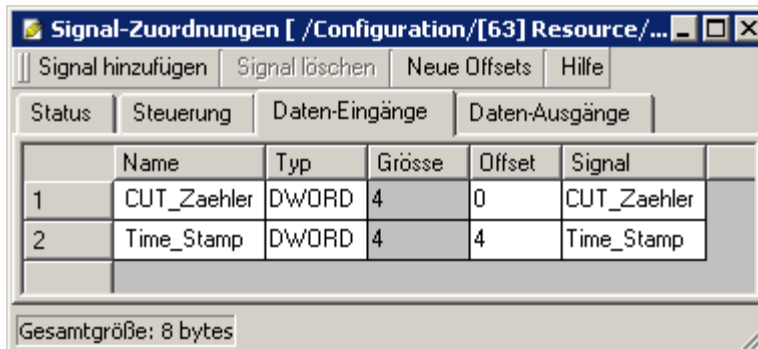


Abbildung 3: Register „Daten-Eingänge“ des COM User Task

Erforderlicher Eintrag im C-Code

Der C-Code des COM User Tasks muss für die Ausgänge der COM (Eingangsbereich der CPU) die folgende Datenstruktur CUT_PDO enthalten:

```
/* ELOP II Factory Daten-Eingänge (COM->CPU) */
udword CUT_PDO[2] __attribute__((section("CUT_PD_OUT_SECT"),aligned(1))));
```

Die Größe der Datenstruktur CUT_PDO muss der Größe der konfigurierten Daten-Eingänge in ELOP II Factory entsprechen.

2.5.4 Daten-Ausgänge (CPU->COM)

In das Register **Daten-Ausgänge** werden die Signale eingetragen, die von der CPU (Ausgangsbereich der CPU) zur COM (CUT) übertragen werden sollen. Hier werden die Datentypen und die Größe der Datenstruktur (des Ausgangsbereichs der CPU) konfiguriert.

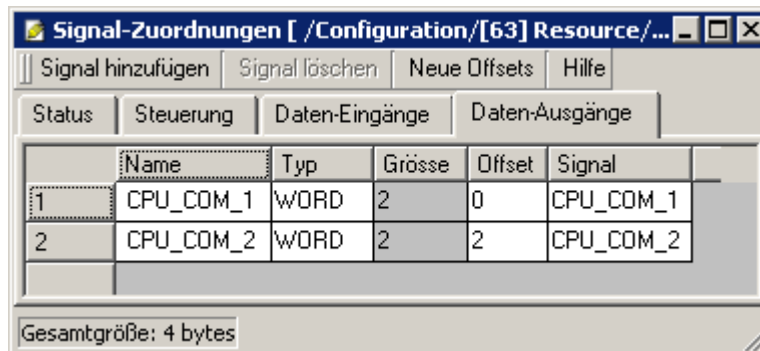


Abbildung 4: Register „Daten-Ausgänge“ des COM User Task

Erforderlicher Eintrag im C-Code

Der C-Code des COM User Tasks muss für die Eingänge der COM (Ausgangsbereich der CPU) die folgende Datenstruktur CUT_PDI enthalten:

```
/* ELOP II Factory Daten-Ausgänge (CPU->COM) */
uword CUT_PDI[2] __attribute__((section("CUT_PD_IN_SECT"),aligned(1)));
```

Die Größe der Datenstruktur CUT_PDI muss der Größe der konfigurierten Daten-Ausgänge in ELOP II Factory entsprechen.

3 CUT Funktionen

3.1 COM-User-Callback-Funktionen

Die COM-User-Callback Funktionen haben alle den Prefix „**CUCB_**“ und werden bei Ereignissen direkt von der COM aufgerufen.

Hinweis Alle COM-User-Callback Funktionen müssen im C-Code des Anwenders definiert werden!

Die COM-User-Callback (CUCB) und die COM-User-Library (CUL) Funktionen teilen sich den gleichen Code- und Datenspeicher sowie den Stack. Diese Funktionen stellen gegenseitig die Konsistenz der gemeinsam verwendeten Daten (Variablen) sicher. Die einzige Ausnahme stellt die Funktion `CUCB_IrqService()` dar, die einen eigenen Stack besitzt. Die Daten (Variablen) die gemeinsam von den CUCB-/CUL-Funktionen mit der Funktion `CUCB_IrqService()` genutzt werden, müssen über Semaphoren geschützt werden.

3.2 COM-User-Library-Funktionen

Alle COM-User-Library-Funktionen und Variablen haben den Prefix „**CUL_**“ und werden in der CUT aufgerufen.

Diese CUL Funktionen sind alle über das Objekt-File **libcut.a** verfügbar.

3.3 Header-Files

Die beiden Header Files **cut.h** und **cut_types.h** enthalten alle Funktionsprototypen für CUL/CUCB und die zugehörigen Datentypen und Konstanten.

Zur verkürzten Schreibweise werden die folgenden Datentypen im Header-File **cut_types.h** definiert:

```
typedef unsigned long  udword;
typedef unsigned short uword;
typedef unsigned char  ubyte;
typedef signed long    dword;
typedef signed short   word;
typedef signed char    sbyte;
#ifndef HAS_BOOL
typedef unsigned char  bool; // mit 0=FALSE, sonst TRUE
#endif
```

3.4 Code-/Datenbereich und Stack für die CUT

Der Code-/Datenbereich ist ein zusammenhängender Speicherbereich, der mit dem Code-Segment und dem Initialdaten-Segment beginnt und mit den Datensegmenten fortgesetzt wird. Im HIMA Linkersteuerfile (**makeinc.inc.app** und **section.dld**) ist die beschriebene Reihenfolge der Segmente und die verfügbare Speichermenge festgelegt.

Die COM User Task teilt mithilfe des HIMA Linkersteuerfiles den verfügbaren Speicherbereich optimal zwischen dem Code und den Daten auf.

Startadresse 0x790000

Länge 448kByte

Der Stack liegt in einem reservierten Speicherbereich, der zur Laufzeit des COM-Betriebssystems festgelegt wird.

Endadresse Dynamisch aus Sicht der CUT

Länge 64kByte

3.5 Startfunktion CUCB_TaskLoop

Die Funktion `CUCB_TaskLoop()` ist die Startfunktion zur COM User Task.

Die Programmausführung der COM User Task beginnt mit dem Aufruf dieser Funktion (siehe `Schedule-Intervall[ms]` Kapitel 2.1).

Funktionsprototyp:

```
void CUCB_TaskLoop(udword mode)
```

Parameter:

Die Funktion hat den folgenden Parameter

Parameter	Beschreibung
mode	1 = MODE_STOP entspricht dem Modus STOP_VALID_CONFIG 2 = MODE_RUN normaler Betrieb der Steuerung

3.6 Serielle Schnittstellen RS485 / RS232 IF

Die verwendeten Feldbusschnittstellen müssen mit den entsprechenden Aufsteckmodulen (Hardware) bestückt sein.

Hinweis	Für jeden <i>HIMatrix</i> -Steuerungstyp steht die jeweilige Systemdokumentationen zur Verfügung. (Siehe Systemhandbuch HI 800 140 und Datenblätter der <i>HIMatrix</i> -Steuerungen).
----------------	--

3.6.1 CUL_AscOpen

Die Funktion `CUL_AscOpen()` initialisiert die eingegebene serielle Schnittstelle (*comId*) mit den übergebenen Parametern. Nach dem Aufruf der Funktion `CUL_AscOpen()` beginnt die COM sofort mit dem Empfang von Daten über diese Schnittstelle.

Die empfangenen Daten werden in einem Software FIFO der Größe 1kByte für **jede** initialisierte serielle Schnittstelle gespeichert.

Die Daten werden solange gespeichert, bis diese mit der Funktion `CUL_AscRcv()` von der CUT ausgelesen werden.

Hinweis	Ist das Auslesen der Daten aus dem FIFO langsamer als der Empfang neuer Daten, so werden die neu empfangenen Daten verworfen.
----------------	---

Funktionsprototyp:

```
udword CUL_AscOpen( Udword comId,
                    Ubyte duplex,
                    udword baudRate,
                    ubyte parity,
                    ubyte stopBits)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
comId	Feldbusschnittstelle (RS485, RS 232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE
duplex	0 = Full-Duplex (nur für FB4 falls RS232 erlaubt) 1 = Halb-Duplex
baudRate	1 = 1200 Bit 2 = 2400 Bit 3 = 4800 Bit 4 = 9600 Bit 5 = 19200 Bit 6 = 38400 Bit 7 = 57600 Bit 8 = 115000 Bit
Die Datenbitlänge ist fest auf 8 Datenbits eingestellt. Zu diesen 8 Datenbits kommen die parametrisierten Bits für Parity und Stopbits, sowie ein Startbit hinzu.	
parity	0 = NONE 1 = EVEN 2 = ODD
stopBits	1 = 1 Bit 2 = 2 Bits

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.

Die Error codes sind im Header-Datei cut.h definiert.

Error code	Beschreibung
CUL_OKAY	Die Initialisierung wurde erfolgreich ausgeführt.
CUL_ALREADY_IN_USE	Die Schnittstelle wird durch andere Funktionen der COM benutzt oder ist bereits offen.
CUL_INVALID_PARAM	Es wurden unzulässige Parameter oder Parameterkombinationen übergeben.
CUL_DEVICE_ERROR	Sonstige Fehler

3.6.2 CUL_AscClose

Die Funktion `CUL_AscClose()` schließt die in `comId` eingetragene serielle Schnittstelle. Dabei werden die bereits empfangenen, aber noch nicht mit der Funktion `CUL_AscRcv()` ausgelesenen Daten im FIFO gelöscht.

Funktionsprototyp :

```
Udword CUL_AscClose(udword comId)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
comId	Feldbusschnittstelle (RS485, RS 232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.
Die Error codes sind im Header-File `cut.h` definiert.

Error code	Beschreibung
CUL_OKAY	Die Schnittstelle wurde erfolgreich geschlossen.
CUL_NOT_OPENED	Die Schnittstelle war nicht (durch die CUT) geöffnet.
CUL_INVALID_PARAM	Es wurden unzulässige Parameter oder Parameterkombinationen übergeben.
CUL_DEVICE_ERROR	Sonstige Fehler

3.6.3 CUL_AscRcv

Die Funktion `CUL_AscRcv()` beauftragt die COM, eine definierte Datenmenge aus dem FIFO zur Verfügung zu stellen.

Sobald die angefragte Datenmenge verfügbar ist (und die CUL bzw. das Scheduling es zulassen), ruft die COM die Funktion `CUCB_AscRcvReady()` auf.

Sind nicht genug Daten im FIFO, so kehrt die Funktion `CUL_AscRcv()` sofort zurück.

Der Auftrag für den Datenempfang bleibt solange gespeichert bis:

- Der Auftrag vollständig abgearbeitet wurde oder
- die Funktion `CUL_AscClose()` aufgerufen wird oder
- durch einen neuen Auftrag neu definiert wird.

Hinweis Bis der Auftrag fertig ist, darf der Inhalt von `*pBuf` nur noch über die Funktion `CUCB_AscRcvReady()` geändert werden.

Funktionsprototyp:

```
Udword CUL_AscRcv(udword comId, CUCB_ASC_BUFFER *pBuf)
```

```
typedef struct CUCB_AscBuffer {
    bool  bAscState; // zur Verwendung durch CUT/CUCB
    bool  bError;    // zur Verwendung durch CUT/CUCB
    uword align;     // COM ist 4 aligned, long's sind performanter
    udword mDataIdx; // Byte-Offset in aData, ab dem die Daten liegen
    udword mDataMax; // max. Byte-Offset: (mDataMax-mDataIdx) gibt an,
                    // // wieviele Bytes in aData gesendet oder
                    // // empfangen werden müssen
    udword aData[1]; // Beginn des Datenkopierbereichs
}CUCB_ASC_BUFFER;
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
comId	Feldbusschnittstelle (RS485, RS 232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE
pBuf	Definiert die angeforderte Datenmenge und den Ort, an den sie kopiert werden soll, bevor dann CUCB_AscReady() aufgerufen wird. Sind bereits ausreichend Daten im FIFO vorhanden, wird während CUL_AscRcv() CUCB_AscRcvReady() aufgerufen.

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.

Die Error codes sind im Header-File cut.h definiert.

Error code	Beschreibung
CUL_OKAY	wenn der Auftrag erfolgreich war, sonst Fehlercode.
CUL_NOT_OPENED	falls die Schnittstelle nicht durch die CUT geöffnet wurde
CUL_INVALID_PARAM	Es wurden unzulässige Parameter oder Parameterkombinationen übergeben.
CUL_DEVICE_ERROR	Sonstige Fehler

Restriktionen:

- Falls der durch CUCB_ASC_BUFFER definierte Speicherbereich nicht im Datensegment der CUT liegt, werden die CUT und die CUT terminiert.
- Es können maximal 1024 Byte Daten angefordert werden.

3.6.4 CUCB_AscRcvReady

Wenn die COM die Funktion `CUCB_AscRcvReady()` aufruft, dann liegt die angeforderte Datenmenge im FIFO bereit (Daten von der im Parameter `comId` definierten seriellen Schnittstelle).

Die Daten wurde zuvor mit der Funktion `CUL_AscRcv()` angefordert.

Der Aufruf der Funktion `CUCB_AscRcvReady()` kann außerhalb und während des Aufrufs der Funktion `CUL_AscRcv()` erfolgen. Der Task-Kontext ist immer der der CUT.

Die Funktion `CUCB_AscRcvReady()` darf alle CUT-Library-Funktionen aufrufen.

Ebenfalls erlaubt ist

- die Erhöhung von `mDataMax`, bzw.
- die neue Parametrierung von `mDataIdx` und `mDataMax` von der `comId` zugeordneten `*pBuf` Daten (zum Weiterlesen).

Das Strukturelement von `CUCB_ASC_BUFFER.mDataIdx` hat den Wert von `CUCB_ASC_BUFFER.mDataMax`.

Funktionsprototyp:

```
void CUCB_AscRcvReady(udword comId)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
<code>comId</code>	Feldbusschnittstelle (RS485, RS 232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE

Restriktionen:

Falls der durch `CUCB_ASC_BUFFER` definierte Speicherbereich nicht im Datensegment von CUT liegt, werden CUIT und CUT terminiert.

3.6.5 CUL_AscSend

Die Funktion `CUCB_AscSend` sendet die durch den Parameter `pBuf` definierte Datenmenge über die serielle Schnittstelle `comId`.

Die definierte Datenmenge muss ≥ 1 Byte und $\leq 1\text{kByte}$ sein.

Nach erfolgreichem Senden wird die Funktion `CUCB_AscSendReady()` aufgerufen.

Im Fehlerfall wird

- nicht gesendet und
- die Funktion `CUCB_AscSendReady()` wird nicht aufgerufen.

Funktionsprototyp:

```
Udword CUL_AscSend(udword comId, CUCB_ASC_BUFFER *pBuf)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
<code>comId</code>	Feldbusschnittstelle (RS485, RS 232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE
<code>pBuf</code>	Definiert die zu sendende Datenmenge

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.

Die Error codes sind im Header-File `cut.h` definiert.

Error code	Beschreibung
<code>CUL_OKAY</code>	falls das Versenden erfolgreich durchgeführt wurde
<code>CUL_WOULDBLOCK</code>	falls eine zuvor versendete Nachricht noch nicht versendet wurde
<code>CUL_NOT_OPENED</code>	falls die Schnittstelle nicht von CUT geöffnet wurde
<code>CUL_INVALID_PARAM</code>	Es wurden unzulässige Parameter oder Parameterkombinationen übergeben.
<code>CUL_DEVICE_ERROR</code>	Sonstige Fehler

Restriktionen:

Falls der durch `CUCB_ASC_BUFFER` definierte Speicherbereich nicht im Datensegment von CUT liegt, werden CUIT und CUT terminiert.

3.6.6 CUCB_AscSendReady

Wenn die COM die Funktion `CUCB_AscSendReady()` aufruft, dann ist das Senden der Daten mit der Funktion `CUCB_AscSend()` über die serielle Schnittstelle abgeschlossen.

Der Task-Kontext ist immer der der CUT. Die Funktion `CUCB_AscSendReady()` darf alle CUT-Library-Funktionen aufrufen.

Funktionsprototyp:

```
void CUCB_AscSendReady(udword comId)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
comId	Feldbusschnittstelle (RS485, RS 232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE

3.7 UDP/TCP-Socket-IF

Maximal 8 Sockets stehen unabhängig vom verwendeten Protokoll zur gleichzeitigen Nutzung zur Verfügung.
Die physikalische Verbindung erfolgt über die 10/100BaseT Ethernet Schnittstellen der Steuerung.

3.7.1 CUL_SocketOpenUdpBind

Die Funktion `CUL_SocketOpenUdpBind()` erzeugt einen Socket vom Typ UDP und bindet den Socket an den ausgewählten Port.

Die Adresse für das Binden ist immer `INADDR_ANY`, d.h. alle an die COM adressierten Nachrichten für UDP/port werden empfangen. Sockets werden immer im non-blocking Mode betrieben; d.h. diese Funktion blockiert nicht.

Funktionsprototyp:

```
dword CUL_SocketOpenUdpBind( uword port, uword *assigned_port_ptr )
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
port	Eine freie, durch die COM nicht belegte Portnummer ≥ 0 . Ist der Parameter port = 0, dann wird der Socket an den ersten freien Port gebunden.
assigned_port_ptr	Adresse, an die die gebundene Portnummer kopiert werden soll, falls port = 0 ist, oder NULL falls nicht

Rückgabewert:

Es wird ein Error code (uword) zurückgegeben.
Die Error codes sind im Header-File `cut.h` definiert.

Error code	Beschreibung
socketNummer	Vergebene SocketNummer für UDP falls > 0 ; Fehlercodes sind < 0
CUL_ALREADY_BOUND	Binden an port für UDP nicht möglich
CUL_NO_MORE_SOCKETS	Keine Ressourcen für Socket mehr verfügbar
CUL_SOCK_ERROR	Andere Socket Fehler

Restriktionen:

Ist `assigned_port_ptr` nicht im Besitz der CUT, so werden CUT/CUIT terminiert.

3.7.2 CUL_SocketOpenUdp

Die Funktion `CUL_SocketOpenUdp()` erzeugt einen Socket vom Typ UDP ohne Anbindung an einen Port. Danach können die Nachrichten über den Socket nur versendet werden, kein Empfang.

Funktionsprototyp:

```
dword CUL_SocketOpenUdp ( void )
```

Parameter:

Keine

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.
Die Error codes sind im Header-File cut.h definiert.

Error code	Beschreibung
socketNummer	Vergebene SocketNummer für UDP falls > 0 Fehlercodes sind < 0
CUL_NO_MORE_SOCKETS	Keine Ressourcen für Socket mehr verfügbar
CUL SOCK_ERROR	Andere Socket Fehler

3.7.3 CUL_NetMessageAlloc

Die Funktion `CULNetMessageAlloc()` alloziert Messagespeicher für die Nutzung von

- `CUL_SocketSendTo()` bei UDP und
- `CUL_SocketSend()` bei TCP

Es können maximal 10 Messages gleichzeitig in der CUT in Benutzung sein.

Funktionsprototyp:

```
void *CUL_NetMessageAlloc(udword size, ubyte proto)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
size	Benötigte Speichermenge in Bytes, muss ≥ 1 Byte und ≤ 1400 Byte sein
proto	0 = TCP 1 = UDP

Rückgabe:

Puffer Adresse, an die die zu sendenden Nutzdaten kopiert werden müssen. Es dürfen niemals Speicherbereiche außerhalb des allozierten Bereichs beschrieben werden. Es stehen keine Bereiche für die verwendeten Transportprotokolle zur Verfügung (Ethernet/IP/UDP oder TCP).

Restriktionen:

Falls keine Speicherressourcen mehr zur Verfügung stehen oder die Parametergröße zu groß oder proto > 1 ist, werden die CUT und die CUIT terminiert.

3.7.4 CUL_SocketSendTo

Die Funktion `CUL_SocketSendTo()` versendet die zuvor mit `CULNetMessageAlloc()` allozierte und gefüllte Nachricht als UDP Paket an die Zieladresse `destIp/destPort`. Nach der Sendung wird der Messagespeicher `pMsg` wieder automatisch freigegeben. Bei jeder Sendung muss mit der Funktion `CULMessageAlloc()` zuerst Messagespeicher alloziert werden.

Funktionsprototyp:

```
dword CUL_SocketSendTo( dword socket,
                        void *pMsg,
                        udword size,
                        udword destIp,
                        uword destPort)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
Socket	zuvor mit <code>CUL_SocketOpenUdp()</code> erzeugter Socket
pMsg	zuvor mit <code>CULNetMessageAlloc()</code> reservierter Speicher der UDP Nutzdaten
Size	Speichermenge in Bytes, muss \leq der zuvor allozierten Menge sein
destIp	Zieladresse $\neq 0$, auch <code>0xffffffff</code> als Broadcast erlaubt
destPort	Zielport $\neq 0$

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.
Die Error codes sind im headerfile `cut.h` definiert.

Error code	Beschreibung
CUL_OKAY	Message erfolgreich versendet
CUL_NO_ROUTE	Kein Routing vorhanden um <code>destIp</code> zu erreichen
CUL_WRONG SOCK	Falscher Socket-Typ oder Socket nicht vorhanden
CUL_SOCKET_ERROR	Andere Socket Fehler

Restriktionen:

Ist `pMsg` keine im Besitz der CUT befindliche Message oder ist `size` für `pMsg` zu groß, so werden CUT/CUIT terminiert.

3.7.5 CUCB_SocketUdpRcv

Die COM ruft die Funktion CUCB_SocketUdpRcv() auf, wenn Daten vom Socket bereit liegen.. Im Callback müssen die Daten bei Bedarf aus *pMsg nach CUT-Data kopiert werden. Nach dem return der Funktion darf auf *pMsg nicht mehr zugegriffen werden.

Funktionsprototyp:

```
void CUCB_SocketUdpRcv( dword socket,
                        void *pMsg,
                        udword packetLength,
                        udword dataLength)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
socket	zuvor mit CUL_SocketOpenUdp() erzeugter Socket
pMsg	pMsg zeigt auf den Beginn des UDP-Paket inklusive dem Ethernet-Header. Über den Ethernet-Header kann der Sender der Message ermittelt werden.
packetLength	Die Länge des Paketes steht in packetLength, dabei ist die Länge des Headers mit enthalten.
dataLength	Die Länge des UDP Nutzdatenanteils steht in dataLength.

3.7.6 CUL_NetMessageFree

Die Funktion CUL_NetMessageFree() gibt die zuvor mit CUL_NetMessageAlloc() allozierten Nachricht frei.

Diese Funktion ist im Normalfall nicht notwendig, da durch den Aufruf der Funktion CUL_SocketSendTo() eine automatische Freigabe erfolgt.

Funktionsprototyp:

```
void CUL_NetMessageFree(void *pMsg)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
pMsg	zuvor mit CUL_NetMessageAlloc() reservierter Speicher

Restriktionen:

Ist pMsg keine im Besitz der CUT befindliche Message, so werden CUT/CUIT terminiert.

3.7.7 CUL_SocketOpenTcpServer

Die Funktion `CUL_SocketOpenServer()` erzeugt einen Socket vom Typ TCP und bindet den Socket an den ausgewählten Port.

Die Adresse für das Binden ist immer `INADDR_ANY`. Zusätzlich wird die COM beauftragt auf dem Stream-Socket ein listen auszuführen. Sockets werden immer im non-blocking Mode betrieben; d.h. diese Funktion hier blockiert nicht.

Für die weitere Bedienung des Sockets siehe `CUCB_SocketTryAccept()` und `CUL_SocketAccept()`.

Funktionsprototyp:

```
dword CUL_SocketOpenTcpServer(uword port, udword backlog)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
port	durch die COM nicht belegte Portnummer > 0
backlog	maximale Anzahl wartender Verbindungsaufnahmen für Socket

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.

Die Error codes sind im Header-File `cut.h` definiert.

Error code	Beschreibung
socketNummer	Vergebene SocketNummer für UDP falls > 0 Fehlercodes sind < 0
CUL_ALREADY_BOUND	Binden an port/proto nicht möglich
CUL_NO_MORE_SOCKETS	Keine Ressourcen für Socket mehr verfügbar
CUL_SOCK_ERROR	Andere Socket Fehler

Restriktionen:

Im erfolgreichen Fall wird 1 Socket verbraucht.

3.7.8 CUCB_SocketTryAccept

Die COM ruft die Funktion `CUL_SocketTryAccept()` auf, wenn eine TCP-Verbindungsanfrage ansteht.

Mit dieser Anfrage kann dann mit der Funktion `CUL_SocketAccept()` ein Socket angelegt werden.

Funktionsprototyp:

```
void CUCB_SocketTryAccept(dword serverSocket)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
serverSocket	zuvor mit <code>CUL_SocketOpenTcpServer()</code> erzeugter Socket.

3.7.9 CUL_SocketAccept

Die Funktion `CUL_SocketAccept()` erzeugt für die zuvor mit `CUCB_SocketTryAccept()` signalisierte Verbindungsanfrage einen neuen Socket.

Funktionsprototyp:

```
dword CUL_SocketAccept( dword serverSocket,
                        udword *pIpAddr,
                        uword *pTcpPort)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
serverSocket	unmittelbar zuvor mit <code>CUCB_SocketTryAccept()</code> signalisierter serverSocket
pIpAddr	Adresse, an die die IP Adresse des Peers kopiert werden soll oder 0 falls nicht
pTcpPort	Adresse, an die die TCP Portnummer des Peers kopiert werden soll oder 0 falls nicht.

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.

Die Error codes sind im Header-File `cut.h` definiert.

Error code	Beschreibung
socket	falls > 0, neu erzeugter Socket
CUL_WRONG_SOCKET	Falscher Socket-Typ oder Socket nicht vorhanden
CUL_NO_MORE_SOCKETS	es sind keine Socket-Ressourcen mehr verfügbar
CUL_SOCKET_ERROR	andere Socket Fehler

Restriktionen:

Sind pIpAddr und pTcpPort nicht im Besitz der CUT, so werden CUT/CUIT terminiert.

3.7.10 CUL_SocketOpenTcpClient

Die Funktion `CUL_SocketOpenTcpClient()` erzeugt einen Socket vom Typ TCP mit freiem lokalem Port und beauftragt eine Verbindung zu `destIp` und `destPort`. Sockets werden immer im non-blocking Mode betrieben; d.h. diese Funktion blockiert nicht. Sobald die Verbindung hergestellt wurde, wird `CUCB_SocketConnected()` aufgerufen.

Funktionsprototyp:

```
dword CUL_SocketOpenTcpClient(udword destIp, uword destPort)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
<code>destIp</code>	IP-Adresse des Kommunikationspartners
<code>destPort</code>	Portnummer des Kommunikationspartners

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.
Die Error codes sind im Header-File `cut.h` definiert.

Error code	Beschreibung
<code>socketNummer</code>	falls > 0; Fehlercodes sind < 0
<code>CUL_NO_MORE_SOCKETS</code>	Keine Ressourcen für Socket mehr verfügbar
<code>CUL_NO_ROUTE</code>	kein Routing vorhanden, um <code>destIp</code> zu erreichen
<code>CUL SOCK ERROR</code>	andere Socket Fehler

3.7.11 CUCB_SocketConnected

Die Funktion `CUCB_SocketConnected()` wird von der COM aufgerufen, wenn mit der Funktion `CUL_SocketOpenTcpClient()` eine TCP-Verbindung aufgebaut wurde.

Funktionsprototyp:

```
void CUCB_SocketConnected(dword socket, bool successfully )
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
<code>socket</code>	zuvor mit <code>CUL_SocketOpenTcpClient()</code> erzeugter und beauftragter Socket
<code>successfully</code>	TRUE, falls der Verbindungsversuch erfolgreich war, ansonsten FALSE

3.7.12 CUL_SocketSend

Die Funktion `CUL_SocketSend()` versendet die zuvor mit `CULNetMessageAlloc()` allozierte und gefüllte Nachricht als TCP Paket.
Nach der Sendung wird der Messagespeicher `pMsg` wieder automatisch freigegeben.
Bei jeder Sendung muss mit der Funktion `CULMessageAlloc()` zuerst Messagespeicher alloziert werden.

Funktionsprototyp:

```
DWORD CUL_SocketSend(  DWORD socket,
                       void *pMsg,
                       UDWORD size)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
socket	zuvor mit <code>CUL_SocketAccept()</code> / <code>CUL_SocketOpenTcpClient()</code> erzeugter Socket
pMsg	zuvor mit <code>CULNetMessageAlloc()</code> reservierter Speicher der TCP Nutzdaten
size	Speichermenge in Bytes, muss \leq der zuvor allozierten Menge sein

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.
Die Error codes sind im Header-File `cut.h` definiert.

Error code	Beschreibung
CUL_OKAY	Message erfolgreich versendet
CUL_WRONG_SOCKET	Falscher Socket-Typ oder Socket nicht vorhanden
CUL_WOULD_BLOCK	Message kann nicht versendet werden, da sonst Socket blockieren würde
CUL_SOCKET_ERROR	andere Socket Fehler

Restriktionen:

Ist `pMsg` keine im Besitz der CUT befindliche Message oder ist `size` für `pMsg` zu groß, so werden CUT/CUIT terminiert.

3.7.13 CUCB_SocketTcpRcv

Die Funktion `CUCB_SocketTcpRcv()` wird von der COM aufgerufen, wenn die Nutzdaten vom Socket bereit liegen.

Nach dem Verlassen der Funktion `CUCB_SocketTcpRcv()` darf auf `*pMsg` nicht mehr zugegriffen werden.

Werden die Nutzdaten auch außerhalb der Funktion `CUCB_SocketTcpRcv()` benötigt, müssen die Nutzdaten aus `*pMsg` in einen dafür angelegten Bereich kopiert werden.

Hinweis Falls die TCP Verbindung asynchron getrennt wird (nach einem Fehler oder auf einen Request der anderen Seite), wird `CUCB_SocketTcpRcv()` mit `dataLength = 0` aufgerufen.

Durch diesen Aufruf wird der CUT signalisiert, dass sie den Socket schließen muss, um die Kommunikation neu zu synchronisieren.

Funktionsprototyp:

```
void CUCB_SocketTcpRcv( dword socket,
                        void *pMsg,
                        udword dataLength)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
socket	Socket, über den die Nutzdaten empfangen wurden.
pMsg	Der Parameter <code>pMsg</code> zeigt auf den Beginn der Nutzdaten ohne Ethernet-/IP-/TCP-Header.
dataLength	Die Länge der Nutzdaten in Bytes

3.7.14 CUL_SocketClose

Die Funktion `CUL_SocketClose()` schließt einen zuvor erzeugten Socket.

Funktionsprototyp:

```
dword CUL_SocketClose(dword socket)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
socket	zuvor erzeugter Socket

Rückgabewert:

Es wird ein Error code (udword) zurückgegeben.

Die Error codes sind in der im Header-Datei `cut.h` definiert.

Error code	Beschreibung
CUL_OKAY	Socket geschlossen und eine Socket-Ressource wieder frei.
CUL_WRONG_SOCKET	Socket nicht vorhanden

3.8 Timer-IF

3.8.1 CUL_GetTimeStampMS

Die Funktion `CUL_GetTimeStampMS()` liefert einen Millisekunden-Tick. Dieser ist geeignet in der CUT/CUIT eigene Timer zu implementieren. Der Zähler wird vom Quarz des COM-Prozessors abgeleitet und hat damit dieselbe Genauigkeit.

Funktionsprototyp:

```
udword CUL_GetTimeStampMS(void)
```

3.8.2 CUL_GetDateAndTime

Die Funktion `CUL_GetDateAndTime()` liefert an die übergebene Speicherstelle `*pSec` die Sekunden seit 1970 und in `*pMsec` die zugehörigen Millisekunden. Die Werte werden mit der sicheren CPU abgeglichen und können je nach Parametrierung über SNTP¹ extern synchronisiert werden.

Die Werte von `CUL_GetDateAndTime()` sollten **nicht** für Zeitmessungen, Timer oder ähnliches verwendet werden, da sie durch die Synchronisation und/oder durch den Anwender im Betrieb gestellt werden können.

Funktionsprototyp:

```
void CUL_GetDateAndTime(udword *pSec, udword *pMsec)
```

Restriktionen:

Ist der Speicher von `pSec` oder `pMsec` nicht im CUT Daten-Segment, werden CUT/CUIT terminiert.

¹ HIMatrix Standardfunktion

3.9 COM-User-IRQ-Task (Nur für MHS31A)

Die COM-User-IRQ-Task (CUIT) muss sich den Code- und Datenspeicher mit der CUT teilen. Sie hat einen eigenen Stack und wird wie bei der CUT-Stack durch das COM-BS (aus Sicht der CUIT dynamisch) festgelegt und ist 8kByte groß. Es gibt nur eine CUIT in der COM. Initial sind die IrqServices disabled, d.h. nach Power-On oder nach dem Laden der Konfiguration.

Restriktionen:

Aus der CUIT dürfen nur die CUL-Funktionen für das Semaphore-Handling aufgerufen werden.

3.9.1 CUCB_IrqService

Die Funktion `CUCB_IrqService()` wird von der COM nach dem Auslösen eines der beiden möglichen CAN IRQs aufgerufen.

Diese Funktion ist für die Bedienung der IRQ-Quelle `devNo` des jeweiligen CAN-Chip zuständig und muss dafür sorgen, dass der CAN-Chip seine IRQ-Anforderung wieder zurücknimmt.

Funktionsprototyp:

```
void CUCB_IrqService(udword devNo)
```

Restriktionen:

Die IRQ Verwaltung des COM-Prozessors wird vom COM-BS durchgeführt und darf nicht von der Funktion `CUCB_IrqService()` übernommen werden.

Hinweis

Die Funktion `CUCB_IrqService()` muss sehr effizient implementiert werden, um unnötige Latenzen anderer COM-Prozessor Funktionen zu minimieren.

Andernfalls ist es möglich, dass bei hoher Last des COM-Prozessors Funktionen nicht mehr ausgeführt werden können, wodurch z.B. auch die sichere Kommunikation der sicheren CPU gestört wird.

Die CUT-Library ermöglicht das Freischalten und Abschalten des COM-IRQ-Kanals, an den der CAN Controller angeschlossen ist.

3.9.2 CUL_IrqServiceEnable

Die Funktion `CUL_IrqServiceEnable()` schaltet den COM-IRQ-Kanal für den ausgewählten CAN Controller `devNo` frei. Ab jetzt lösen CAN-IRQs den Aufruf der CUT-IRQ-Task aus.

Funktionsprototyp:

```
void CUL_IrqServiceEnable(udword devNo)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
devNo	1 = CAN Controller A 2 = CAN Controller B

Restriktionen:

Werden für `devNo` Werte ungleich 1 oder 2 verwendet, so werden CUIT/CUT terminiert.

3.9.3 CUL_IrqServiceDisable

Die Funktion `CUL_IrqServiceDisable()` sperrt den COM-IRQ-Kanal für den CAN Controller *devNo*. Ab jetzt lösen CAN-IRQs nicht mehr den Aufruf der CUT-IRQ-Task aus. Noch nicht vollständig verarbeitete IRQ Behandlungen werden jedoch noch durchgeführt.

Funktionsprototyp:

```
void CUL_IrqServiceDisable(udword devNo)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
devNo	1 = CAN Controller A 2 = CAN Controller B

Restriktionen:

Werden für *devNo* Werte ungleich 1 oder 2 verwendet, so werden CUIT/CUT terminiert.

3.9.4 CUL_DeviceBaseAddr

Die Funktion `CUL_DeviceBaseAddr()` liefert die 32 Bit Basisadresse der CAN Controller.

Funktionsprototyp:

```
void* CUL_DeviceBaseAddr(udword devNo)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
devNo	1 = CAN Controller A 2 = CAN Controller B

Restriktionen:

Werden für *devNo* Werte ungleich 1 oder 2 verwendet, so werden CUIT/CUT terminiert.

3.10 NVRam-IF (Nur für MHS31A)

Die CUT und die CUIT können den verfügbaren Bereich schreiben und lesen. Das für diese Funktionen verfügbare NVRam ist 9kBytes groß.

Hinweis Die COM sorgt **nicht** für die Konsistenz der Daten bei Ausfall der Betriebsspannung während eines Zugriffs und auch nicht während gleichzeitigen Zugriffen aus zwei Tasks.

3.10.1 CUL_NVRamWrite

Die Funktion `CUL_NVRamWrite()` schreibt Daten in das NVRam.

Funktionsprototyp:

```
void CUL_NVRamWrite(udword offset, void *source, udword size)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
offset	im Bereich des NVRams, gültige Werte sind 0 – 9215
source	Speicherbereich in CUT Datensegment, der ins NVRam kopiert werden soll.
size	Anzahl Bytes, die kopiert werden sollen

Restriktionen:

Bei ungültigen Parametern werden die CUT und die CUIT terminiert; d.h. bei:

- offset \geq 9216
- offset+size > 9216
- source nicht im CUT Datensegment
- source+size nicht im CUT Datensegment

3.10.2 CUL_NVRamRead

Die Funktion `CUL_NVRamRead()` liest Daten aus dem NVRam.

Funktionsprototyp:

```
void CUL_NVRamRead(udword offset, void *destination, udword size)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
offset	im Bereich des NVRams, gültige Werte sind 0 – 9215
destination	Speicherbereich in CUT Datensegment, der ins NVRam kopiert werden soll.
size	Anzahl Bytes, die kopiert werden sollen

Restriktionen:

Bei ungültigen Parametern werden die CUT und die CUIT terminiert:

- offset \geq 9216
- offset+size > 9216
- destination nicht im CUT Datensegment
- destination+size nicht im CUT Datensegment

3.11 Semaphore-IF (Nur für MHS31A)

Die CUT und die CUIT haben zur Prozesssynchronisation zusammen **einen** Semaphor. Die Daten der CUCB- und CUL- Funktionen, die gemeinsam mit der Funktion `CUCB_IrqService()` verwendet werden, müssen über einen Semaphor geschützt werden. Damit wird die Konsistenz der gemeinsam mit der Funktion `CUCB_IrqService()` verwendeten Daten sichergestellt.

3.11.1 CUL_SemaRequest

Die Funktion `CUL_SemaRequest()` fordert die Semaphore der CUT/CUIT an.

Ist die Semaphore

- frei, so kehrt die Funktion mit dem Wert `pContext` zurück.
- nicht frei, wird die aufrufende Task blockiert, bis die Semaphore durch eine andere Task freigegeben wird und kehrt mit dem Wert `pContext` zurück.

Der Kontext, der durch Parameter `pContext` referenziert wird, wird nur von den CUL- Funktionen für die aufrufende Task genutzt und darf zwischen Request und Release nicht verändert werden.

Funktionsprototyp:

```
void CUL_SemaRequest(udword *pContext)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
<code>pContext</code>	wird nur innerhalb der aufrufenden Task von CUL benutzt. Der Kontext wird über <code>pContext</code> zurückgegeben und muss bei der Funktion <code>CUL_SemaRelease()</code> wieder angegeben werden.

Restriktionen:

Wird die Anzahl der zulässigen Rekursionen überschritten, werden CUT/CUIT terminiert. Wenn die CUT durch eine Semaphore blockiert wird, werden mit Ausnahme von `CUCB_IrqService()` keine CUCB_'s mehr durchgeführt.

3.11.2 CUL_SemaRelease

Die Funktion `CUL_SemaRelease()` gibt die Semaphore, die durch `*pContext` definiert wird, wieder frei.

Funktionsprototyp:

```
void CUL_SemaRelease(udword *pContext)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
<code>pContext</code>	mit dem gleichen Wert für <code>*pContext</code> , wie er durch <code>CUL_SemaRequest</code> oder <code>CUL_SemaTry</code> geschrieben wurde.

Restriktionen:

Falls häufiger Release aufgerufen wird als Request/Try, werden CUT/CUIL terminiert.

3.11.3 CUL_SemaTry

Die Funktion `CUL_SemaTry()` versucht die Semaphore der CUT/CUIT anzufordern.

Ist die Semaphore

- frei, so kehrt die Funktion mit `TRUE` zurück und belegt die Semaphore.
- nicht frei, so kehrt die Funktion mit `FALSE` zurück und belegt die Semaphore nicht.

Das `udword`, das durch `pContext` referenziert wird, wird nur von der CUL für die aufrufende Task genutzt und darf zwischen Request/Release nicht verändert werden.

Funktionsprototyp:

```
bool CUL_SemaTry(udword *pContext)
```

Parameter:

Die Funktion hat den folgenden Parameter:

Parameter	Beschreibung
<code>pContext</code>	wird nur innerhalb der aufrufenden Task von CUL benutzt

Rückgabewert:

Es wird ein Error code (`udword`) zurückgegeben.

Die Error codes sind im Header-File `cut.h` definiert.

Error code	Beschreibung
<code>TRUE</code>	Semaphore konnte belegt werden
<code>FALSE</code>	Semaphore konnte nicht belegt werden

Kontext wird über `pContext` zurückgegeben und muss bei der Funktion `CUL_SemaRelease` wieder angegeben werden.

Restriktionen:

Wird die Anzahl der zulässigen Rekursionen überschritten, werden CUT/CUIT terminiert.

Wenn die CUT durch eine Semaphore blockiert wird, werden mit Ausnahme von

`CUCB_IrqService()` keine `CUCB_'s` mehr durchgeführt.

Hinweis	Die Funktionen <code>CUL_SemaTry()</code> und <code>CUL_SemaRequest()</code> dürfen auch dann ohne Blockade aufgerufen werden, wenn die aufrufende Task die Semaphore schon belegt hat; nur müssen dann auch gleich viele <code>CUL_SemaRelease</code> erfolgen, bis die Semaphore wieder frei ist. Die Rekursion lässt mindestens 32000 Schritte zu. Ob mehr Schritte möglich sind, hängt von der jeweiligen Ausgabe der COM ab.
----------------	---

3.12 COM-IO-IF (Nur MHS31A)

Schnittstelle zur Bedienung der Standard-IO der COM. Die Filterung der DI Kanäle ist nicht parametrierbar; vielmehr sind immer gefilterte und nicht gefilterte Werte verfügbar.

```
struct CUL_IoBufferIn {
    uword mDIRaw;          //ungefilterte DI Werte,
                           //mit Bit[0]=DI[1]..Bit[11]=DI[12]
                           //Bit[12]=DI[13]/DO[1]..Bit[15]=DI[16]/DO[4]

    uword mDIFilter;       //gefilterte DI Werte,
                           //mit Bit[0]=DI[1]..Bit[15]=DI[16]
}CUL_IO_BUFFER_IN;

struct CUL_IoBufferOut {
    uword mDO;             // DO Wert 1=an, 0=aus,
                           // mit Bit[0]=DO[1] .. Bit[3]=DO[4]
}CUL_IO_BUFFER_OUT; // passend zu mDIDODir und Bit[4]..Bit[15]=0

struct CUL_IoBufferCfg {
    uword mDIDODir;        // Wert 1=Ausgabe, 0=Eingabe,
                           // mit Bit[0]=DI/DO[1]..Bit[3]=DI/DO[4]
                           // Bit[4..15] undefiniert, müssen 0 sein
} CUL_IO_BUFFER_CFG;
```

Die parametrierbaren DI/DO Kanäle, sind die Kanäle 1 bis 4. Die Kanalnummern gelten sowohl für DI wie auch für DO in obigen Strukturen. Ist ein DI/DO Kanal als Ausgang parametrierbar, so sind sein mDIRaw und mDIFilter nicht gültig.

3.12.1 CUL_IORead

Liest die Eingänge. Die Eigenschaften der DI bzgl. der Filter-/Raw-Werte werden durch die HW-Spezifikation definiert.

Funktionsprototyp:

```
void CUL_IORead(CUL_IO_BUFFER_IN *pIN)
```

Restriktionen:

Ist der übergebene Pointer nicht in Besitz der CUT, so werden die CUT/CUIT terminiert.

3.12.2 CUL_IOWrite

Schreibt die Ausgänge. Bevor CUL_IOWrite() erstmalig aufgerufen wird, werden an den Ausgängen 0-Werte ausgegeben.

Funktionsprototyp:

```
void CUL_IOWrite(CUL_IO_BUFFER_OUT *pOUT)
```

Restriktionen:

Ist der übergebene Pointer nicht in Besitz der CUT, so werden die CUT/CUIT terminiert.

3.12.3 CUL_IOConfigure

Die Funktion `CUL_IOConfigure()` konfiguriert die Ein-/Ausgänge.

Bevor `CUL_IOConfigure()` erstmalig aufgerufen wird, sind die DI/DO-Kanäle als Eingang parametrierung. `CUL_IOConfigure()` darf zum Rekonfigurieren der DI/DO-Kanäle mehrmals aufgerufen werden.

DI/DO-Kanäle, deren Parametrierung von

- Ausgang auf Eingang wechselt, geben keine Energie mehr nach außen,
- Eingang auf Ausgang wechselt, geben den zuvor mit `CUL_IOWrite()` definierten Wert aus.

Die COM_IO Funktionen dürfen durch CUIT/CUT auch mehrfach während eines CUCB-Aufrufs aufgerufen werden.

Funktionsprototyp:

```
void CUL_IOConfigure(CUL_IO_BUFFER_CFG *pCFG)
```

Restriktionen:

Ist der übergebene Pointer nicht in Besitz der CUT, so werden CUT/UIT terminiert.

3.13 Diagnose

Die Funktion `CUL_DiagEntry()` trägt ein Ereignis in die COM-Kurzzeitdiagnose ein, welches über das PADT ausgelesen werden kann.

Funktionsprototyp:

```
void CUL_DiagEntry( udword severity,
                   udword code,
                   udword param1,
                   udword param2)
```

Parameter:

Die Funktion hat die folgenden Parameter:

Parameter	Beschreibung
severity	severity dient der Klassifizierung des Ereignisses 0x45 ('E') == Fehler, 0x57 ('W') == Warnung, 0x49 ('I') == Information
code	Der Anwender definiert den Parameter code mit einer beliebigen Nummer für entsprechende Ereignisse. Beim Eintritt des Ereignisses wird die Nummer in der Diagnose angezeigt.
param1, param2	Zusätzliche Informationen über das Ereignis

4 Installation der Entwicklungsumgebung

In diesem Kapitel wird die Installation der Entwicklungsumgebung und die Erstellung einer COM User Task beschrieben.

Die Entwicklungsumgebung befindet sich auf der Installations-CD (siehe auch Kapitel 1.1.2).

4.1 Installation der Cygwin-Umgebung

Die Cygwin-Umgebung ist erforderlich, da die GNU C Compiler Tools nur unter der Cygwin-Umgebung lauffähig sind.

Die Cygwin-Umgebung muss unter Windows 2000/XP installiert werden.

Hinweis	Beachten Sie die Voraussetzungen zur Installation im Kapitel 1.1. Deaktivieren Sie den Virens Scanner auf dem PC, auf dem Cygwin installiert werden soll, um Probleme bei der Installation von Cygwin zu verhindern.
----------------	--

Führen Sie die folgenden Schritte aus, um die Cygwin-Umgebung zu installieren:

Schritt 1: Starten Sie das Setupprogramm zur Installation von Cygwin:

- ☐ Kopieren Sie das Cygwin-Installationsarchiv `cygwin_2005-03-11` von der Installations-CD auf Ihre lokale Festplatte (z. B. Laufwerk `C:\`).
- ☐ Öffnen Sie im Windows-Explorer das Cygwin-Verzeichnis `C:\ cygwin_2005-03-11`.
- ☐ Starten Sie die Installation von Cygwin mit einem Doppelklick auf die Datei **setup.exe**.
- ☐ Klicken Sie im Cygwin-Dialogfenster auf die Schaltfläche **Weiter**, um das Setup auszuführen.

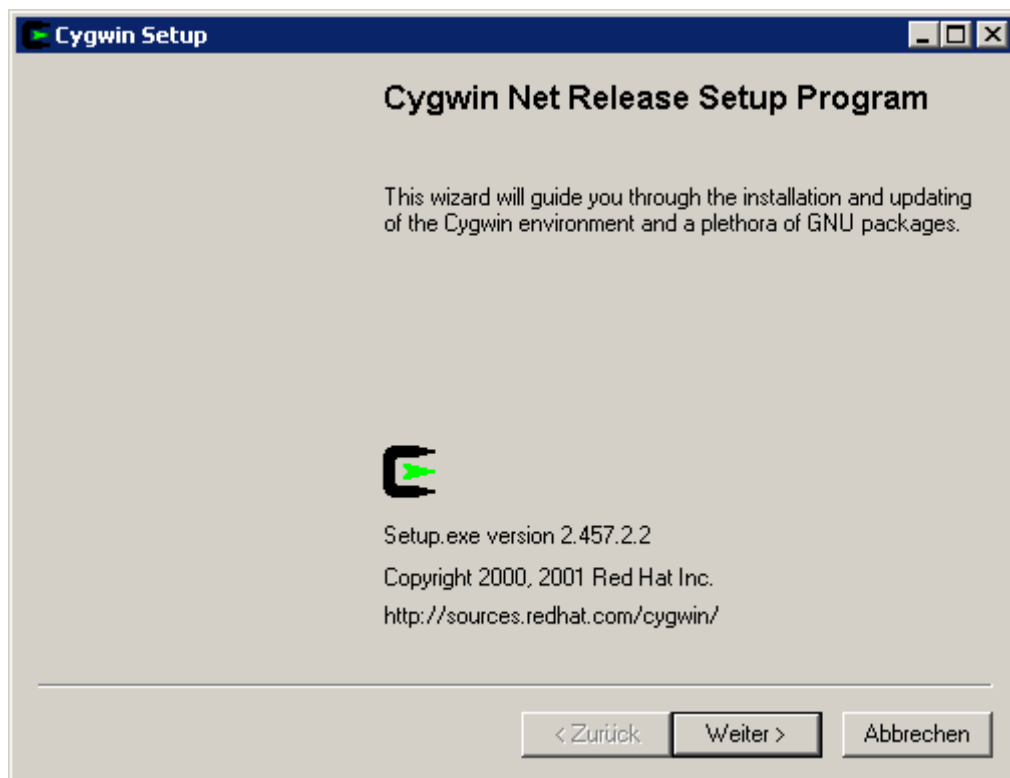


Abbildung 5: Cygwin Setup-Dialog *Cygwin Setup*

Schritt 2:

Der Dialog Disable Virus Scanner erscheint, wenn der Virusscanner nicht deaktiviert wurde.

Führen Sie diesen Schritt aus, um den Virusscanner für die Installation von Cygwin zu deaktivieren.

Hinweis Deaktivieren Sie den Virusscanner vor der Cygwin-Installation, da es abhängig vom verwendeten Virusscanner vorkommen kann, dass dieses Fenster nicht erscheint, obwohl der Virusscanner läuft.

- ☐ Wählen Sie **Disable Virus scanner**, um Probleme während der Installation durch den Virusscanner zu verhindern.
- ☐ Klicken Sie auf die Schaltfläche **Weiter**, um die Eingabe zu bestätigen.

Schritt 3:

Wählen Sie im Dialog *Choose Installation Type* die Installations-Quelle von Cygwin aus:

- ☐ Wählen Sie als Installations-Quelle **Install from Local Directory** aus.
- ☐ Klicken Sie auf die Schaltfläche **Weiter**, um die Eingabe zu bestätigen.

Schritt 4:

Wählen Sie im Dialog *Choose Installation Directory* das Installations-Ziel von Cygwin aus:

- ☐ Geben Sie das Verzeichnis an, in welches Cygwin installiert werden soll.
- ☐ Übernehmen Sie alle weiteren Voreinstellungen des Dialogs.
- ☐ Klicken Sie auf die Schaltfläche **Weiter**, um die Eingabe zu bestätigen.

Schritt 5:

Wählen Sie im Dialog *Select Local Package Directory* das Cygwin-Installationsarchiv

- ☐ Geben Sie im Feld *Local Package Directory* das Cygwin-Installationsarchiv an, in dem sich die Installationsdateien befinden.
- ☐ Klicken Sie auf die Schaltfläche **Weiter**, um die Eingabe zu bestätigen.

Schritt 6: Wählen Sie Dialog *Select Packages* alle Packages zur Installation aus:

- ☐ Wählen Sie den radio button **Curr.**
- ☐ Klicken Sie im Anzeigefeld mehrmals langsam auf die Installationsoption neben **All**, bis **Install** für eine vollständige Installation aller packages angezeigt wird (ca. 1,86 GB Speicherbedarf).

Hinweis Achten Sie darauf, dass hinter jedem package Install steht.
Wenn die packages nicht vollständig Installiert werden, dann fehlen wichtige Funktionen, um später den C-Code der CUT zu compilieren!

- ☐ Klicken Sie auf die Schaltfläche **Weiter**, um die Eingabe zu bestätigen.

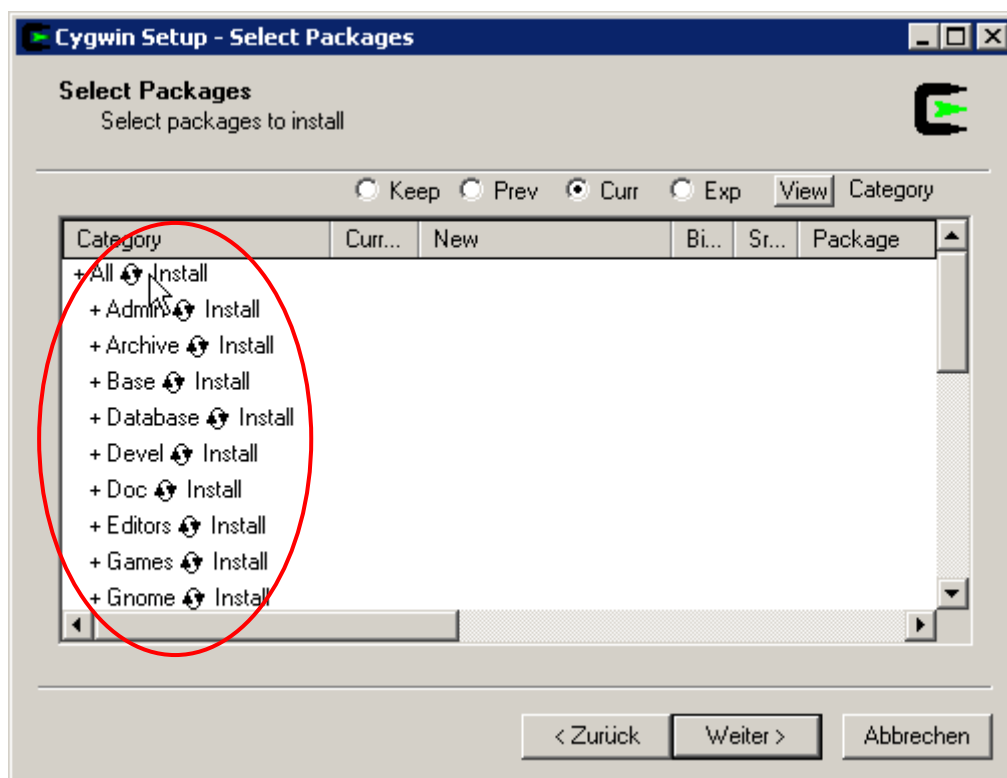


Abbildung 6: Cygwin Setup-Dialog *Select Packages*

Schritt 7: Schließen Sie die Cygwin-Installation mit den folgenden Einträgen ab:

- ☐ Wählen Sie Eintrag in das **Startmenü**.
- ☐ Wählen Sie den Eintrag **Desktop-Icon**.
- ☐ Klicken Sie auf die Schaltfläche **Fertigstellen**, um die Installation von Cygwin abzuschließen.

Cygwin Befehle	Beschreibung
cd (Name Verzeichnis)	Verzeichnis wechseln
cd ..	In höheres Verzeichnis wechseln
ls -l	Alle Dateien eines Verzeichnisses anzeigen
help	Übersicht über Bash Shell Kommandos

Tabelle 6: Befehle in Cygwin (Bash Shell)

4.2 Installation des GNU Compilers

Führen Sie die folgenden Schritte aus, um den GNU Compiler zu installieren:

- Öffnen Sie im Windows-Explorer das Verzeichnis der Installations CD.
- Doppelklicken Sie auf das zip-File `gcc-ppc-v3.3.2_binutils-v2.15.zip`.
- Extrahieren Sie alle Dateien in das Cygwin-Verzeichnis (z.B. `C:\cygwin\...`). Der GNU Compiler wird im Cygwin-Verzeichnis in den Ordner **gcc-ppc** entpackt.
- Tragen Sie in der Systemsteuerung die Umgebungsvariablen ein:
 - Öffnen Sie die Systemeigenschaften über das Windows-Startmenü **Einstellungen->Systemsteuerung->System**.
 - Wählen Sie das Register **Erweitert**.
 - Klicken Sie auf die Schaltfläche **Umgebungsvariablen**.
 - Wählen Sie im Feld *Systemvariablen* die Systemvariable **Path** und erweitern Sie die Systemvariable mit dem Eintrag: `C:\cygwin\gcc-ppc\bin`.

Kopieren Sie von der Installations-CD den Ordner **cut_src** in das Home-Verzeichnis. Der Ordner **cut_src** enthält alle für die Erstellung eines COM User Task benötigten "include"- und "lib"-Verzeichnisse. Die mitgelieferte Sourcdatei **cut_cbf.c** befindet sich standardmäßig im Verzeichnis `...\cut_src\cutapp\`.

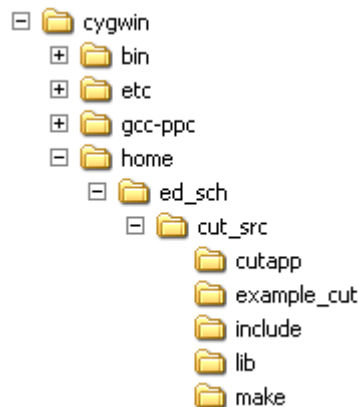


Abbildung 7: Strukturbaum von cygwin

Hinweis Wenn das Home-Verzeichnis nicht automatisch angelegt wurde, dann legen Sie das Home-Verzeichnis mit dem Windows-Explorer an (z.B. `C:\cygwin\home\User1`).

Wenn Sie ein anderes Home-Verzeichnis für die Cygwin Bash Shell anlegen wollen, dann müssen Sie die Batchdatei `cygwin.bat` mit dem Befehl `set Home` ergänzen.

```
@echo off
```

```
C:
chdir C:\cygwin\bin
set Home=C:\User1
bash --login -i
```

Abbildung 8: Batchdatei *Cygwin.bat*

5 Neues CUT-Projekt anlegen

Dieses Kapitel zeigt, wie ein neues CUT-Projekt angelegt wird und welche Dateien angepasst werden müssen.

Hinweis Das CUT-Projekt **example_cut** befindet sich fertig angepasst auf der Installations-CD.

Beim Anlegen weiterer neuer CUT-Projekte wird empfohlen, für jedes CUT-Projekt ein neues Verzeichnis unterhalb ...`\cut_src\` einzurichten.

Beispiel:

Zum Test wird das Verzeichnis *example_cut* angelegt, die C-Source heißt **example_cut.c**, die erzeugte Idb-Datei im Verzeichnis *make* heißt dann **example_cut.idb**.

Erstellen Sie für eine neue COM User Task den Ordner *example_cut*.

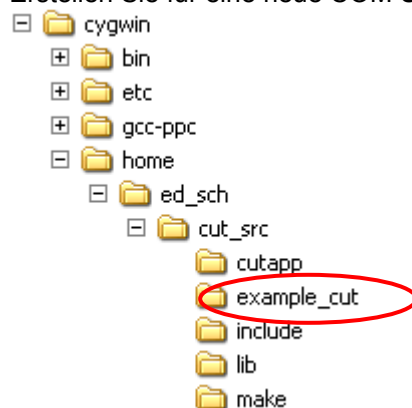


Abbildung 9: Strukturbaum von cygwin

- Kopieren Sie die Dateien
 - cutapp.c
 - cutapp.mke
 - makefile
 aus dem Verzeichnis *cutapp* in das Verzeichnis *example_cut*.
- Die Dateinamen müssen in den Loadable-Namen umbenannt werden (z.B. von **cutapp** in **example_cut**).

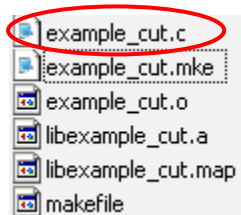


Abbildung 10: C-Code Datei im Ordner *example_cut*

Die Änderungen in der mke-Datei und im makefile müssen, wie in den nächsten Kapiteln beschrieben, bei einem neuen Projekt durchgeführt werden.

5.1 CUT-Makefiles

Konfiguration der CUT-Makefiles für unterschiedliche Sourcefiles und Idb-Dateien
Insgesamt müssen drei Makefiles, wie in den folgenden Absätzen beschrieben, angepasst werden.

5.1.1 Makefile mit der Erweiterung „.mke“

Die mke-Datei finden Sie im jeweiligen Quellcode-Verzeichnis
z.B. `cut_src\example_cut\example_cut.mke`.

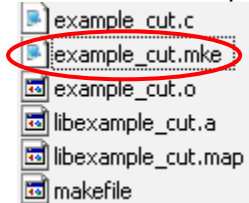


Abbildung 11: mke-Datei im Ordner example_cut

Führen Sie in der mke-Datei die folgenden Änderungen durch:

- Der Variablen **module** muss der gleiche Loadable-Namen zugewiesen werden wie der .mke Datei (z.B. example_cut).
- Der Variablen **c_sources** können eine oder mehrere C-Dateien zugewiesen werden, welche für die Erstellung des Ziel-Codes (Loadable-Datei) benötigt werden.

```
#####
#
# make file (DOS/NT)
# $Id: cutapp.mke 58869 2005-10-11 12:35:46Z es_fp $
#####
#
# assign name of module here (e.g. nl for NetworkLayer)
module= example_cut
#
# assign module sources here
sources=

c_sources= $(module).c
asm_sources=
```

Abbildung 12: mke-Datei ab der Zeile 1

5.1.2 Makefile

Die makefile-Datei finden Sie im jeweiligen Quellcode-Verzeichnis.
z.B. cut_src\example_cut\makefile

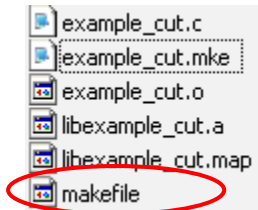


Abbildung 13: makefile im Ordner *example_cut*

Führen Sie in der makefile-Datei die folgenden Änderungen durch:

- Ziehen Sie die Include-Zeile für die mke-Datei nach oben und ändern Sie die .mke Datei auf den aktuellen Namen.
- Erweitern Sie den Make-Aufruf mit den beiden Variablen **SUBMOD_DIRS** und **CUT_NAME**.

all_objects:

include example_cut.mke

cut:

\$(MAKE) -C ../make elf **SUBMOD_DIRS=cut_src/\$(module)** **CUT_NAME=\$(module)**

all_objects: \$(c_objects) \$(asm_objects) \$(objects)

Abbildung 14: makefile ab der Zeile 34

5.1.3 Makefile mit der Erweiterung „makeinc.inc.app“

Als einmalige Änderung für dieses und alle weiteren CUT-Projekte wird der Name des CUT-Loadable über eine Make-Variable änderbar gemacht.

Die makeinc.inc.app-Datei finden Sie im cut_src-Verzeichnis
z.B. *cut_src\makeinc.inc.app*.

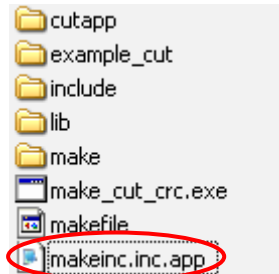


Abbildung 15: makeinc.inc.app Datei im Ordner *example_cut*

Führen Sie in der makeinc.inc.app-Datei die folgenden Änderungen durch:

- Erweitern Sie die Datei mit der Variablen **CUT_NAME**.

```
all : lib$(module).$(LIBEXT)
@echo 'did make for module ['lib$(module).$(LIBEXT)']'

lib$(module).$(LIBEXT) : $(objects) $(c_objects) $(asm_objects) $(libraries)

SUBMOD2_LIBS=$(foreach lib,$(SUBMOD_LIBS),../$(lib))
```

CUT_NAME=cut

```
makeAllLibs:
$(MAKE) -C ../cut_src cut_src
```

```
makeLoadable:
@echo; \
BGTYPE=" $(CUT_NAME)"; \
if [ ! -f $$BGTYPE.map ] ; then \
echo "Error: MAP-Datei $$BGTYPE.map existiert nicht"; \
exit 1; \
fi; \
OS_LENGTH=$(gawk '/__OS_LENGTH/ {print substr($$1,3,8)}' $$BGTYPE.map); \
echo; \
$(OBJCOPY) --strip-all --strip-debug -O binary $$BGTYPE.elf $$BGTYPE.bin; \
echo; \
echo "Building C3-Loadable-Binary ..."; \
$(MCR)C) $$BGTYPE.bin 0 $$OS_LENGTH $$OS_LENGTH $$BGTYPE.ldb; \
echo; \

$(CUT_NAME).elf: makeAllLibs $(SUBMOD2_LIBS)

elf:
@echo; test -f section.dld && $(MAKE) $(CUT_NAME).elf && $(MAKE) makeLoadable \
|| { echo "ERROR: Wrong subdir. Please invoke elf target only from make/ subdirectory." && \
echo && false ; } ;
```

end of file: makeinc.inc

Abbildung 16: makeinc.inc.app ab der Zeile 247

5.2 C-Quellcode bearbeiten

Führen Sie die folgenden Schritte aus, um die Quellcodedatei zu öffnen:

- Öffnen Sie das Projektverzeichnis *cut_srcexample_cut*, das Sie in den vorangegangenen Schritten erstellt und konfiguriert haben.
- Öffnen Sie mit einem Editor (z.B. Notepad) die C-Quellcodedatei mit der Erweiterung *.c*.

5.2.1 Eingangs- und Ausgangssignale konfigurieren

Führen Sie die folgenden Schritte aus, um die Ein- und Ausgangssignale in der Quellcodedatei zu konfigurieren:

- Die Datengröße der Signale, die im Register **Daten Ausgänge** im ELOP II Factory Hardware-Management angelegt werden sollen, muss in der Quellcodedatei im Array **CUT_PDI[X]** angelegt werden.
- Die Datengröße der Signale, die im Register **Daten Eingänge** im ELOP II Factory Hardware-Management angelegt werden sollen, muss in der Quellcodedatei im Array **CUT_PDO[X]** angelegt werden.

5.2.2 Startfunktion im CUT

Die C-Funktion `void CUCB_TaskLoop (udword mode)` ist die Startfunktion und wird vom Anwenderprogramm zyklisch aufgerufen.

5.2.3 Beispielcode „example_cut.c“

Der folgende C-Code kopiert den Wert vom Eingang **CUT_PDI[0]** in den Ausgang **CUT_PDO[0]** und gibt den Wert unverändert an das ELOP II Factory Anwenderprogramm zurück.

Hinweis Der C-Code **example_cut.c** befindet sich auf der Installations-CD.

```
/* Ein Beispiel fuer die CUT-Implementation */
#include "include/cut_types.h"
#include "include/cut.h"
#ifdef __cplusplus
extern "C" {
#endif
/*****
/* ELOP II Factory Daten-Ausgaenge (CPU->COM) */
udword CUT_PDI[1] __attribute__((section("CUT_PD_IN_SECT"), aligned(1)));
/* ELOP II Factory Daten-Eingaenge (COM->CPU) */
udword CUT_PDO[1] __attribute__((section("CUT_PD_OUT_SECT"), aligned(1)));
*****/

/* Callback-Funktion zum zyklischen Starten der CUT */
void CUCB_TaskLoop(udword mode)
{
    if (CUT_PDI[0] > CUT_PDO[0]) /*Wird nur dann ausgeführt, wenn das */
    {                               /*ELOP II Factory Anwenderprogramm */
                                    /*abgearbeitet wurde */
                                    /*Das ELOP II Factory Anwenderprogramm */
                                    /*addiert zu CUT_PDO[0] den Wert 1 und */
                                    /*schreibt das Ergebnis in CUT_PDI[0] */

        CUT_PDO[0] = CUT_PDI[0]; /*Kopiert den Wert vom Eingang CUT_PDI[0] */
                                /*in den Ausgang CUT_PDO[0] der SPS */

        if (CUT_PDO[0] == 65535)
            {CUT_PDO[0] = 0;}
    }
}
/*****/
```

```

/*****
void CUCB_AscRcvReady(udword comId)
{
    CUL_DiagEntry(0x49, 1, comId, 0);
}
*****/
/*****
void CUCB_AscSendReady(udword comId)
{
    CUL_DiagEntry(0x49, 2, comId, 0);
}
*****/
/*****
void CUCB_SocketTryAccept(dword serverSocket)
{
    CUL_DiagEntry(0x49, 3, serverSocket, 0);
}
*****/
/*****
void CUCB_SocketConnected(dword socket, bool Okay)
{
    CUL_DiagEntry(0x49, 4, socket, Okay);
}
*****/
/*****
void CUCB_SocketTcpRcv(dword socket, void *pMsg, udword dataLength)
{
    CUL_DiagEntry(0x49, 5, socket, dataLength);
}
*****/
/*****
void CUCB_SocketUdpRcv(dword socket, void *pMsg, udword packetLength,
                      udword dataLength)
{
    CUL_DiagEntry(0x49, 6, socket, dataLength);
}
*****/
/*****
void CUCB_IrqService(udword devNo)
{
    CUL_DiagEntry(0x49, 7, devNo, 0);
}
*****/

#ifdef __cplusplus
} /* end extern "C" */
#endif

/* end of file */

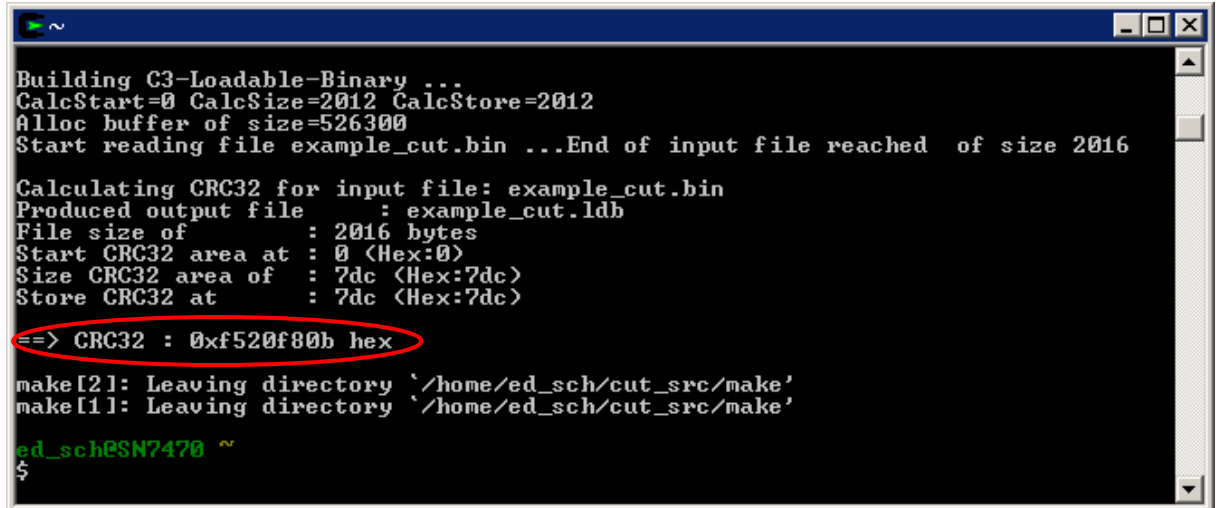
```

Abbildung 17: C-Code example_cut.c

5.2.4 Erstellung des Ausführbaren Code (ldb-Datei)

Führen Sie die folgenden Schritte zur Erstellung des ausführbaren Codes (ldb-Datei) aus:

- Starten Sie die **Cygwin Bash Shell**.
- Wechseln Sie in das Verzeichnis `/cut_src/example_cut/`.
- Starten Sie die Codegenerierung mit der Eingabe `make cut`.
Das Binärfile **cut.ldb** wird im Verzeichnis `/cut_src/make/` automatisch erzeugt.
- Wenn der CRC32 erzeugt wurde, dann wurde auch ausführbarer Code erzeugt (siehe rote Markierung in Abbildung 18).



```
Building C3-Loadable-Binary ...
CalcStart=0 CalcSize=2012 CalcStore=2012
Alloc buffer of size=526300
Start reading file example_cut.bin ...End of input file reached of size 2016

Calculating CRC32 for input file: example_cut.bin
Produced output file      : example_cut.ldb
File size of              : 2016 bytes
Start CRC32 area at      : 0 (Hex:0)
Size CRC32 area of       : 7dc (Hex:7dc)
Store CRC32 at           : 7dc (Hex:7dc)
=> CRC32 : 0xf520f80b hex
make[2]: Leaving directory `/home/ed_sch/cut_src/make'
make[1]: Leaving directory `/home/ed_sch/cut_src/make'
ed_sch@SN7470 ~
$
```

Abbildung 18: Cygwin Bash Shell

Dieser ausführbare Code (ldb-Datei) wird in die COM User Task geladen (siehe Kapitel 5.3).

5.3 COM User Task in das Projekt einbinden

Führen Sie in ELOP II Factory die folgenden Schritte aus, um den COM User Task in das Projekt einzubinden:

5.3.1 COM User Task anlegen

- Wählen Sie im ELOP II Factory Hardware-Management die Ressource, in welcher die CUT angelegt werden soll.
- Wählen Sie im Kontextmenü des COM-Modul **Neu->COM User Task**, um den COM User Task anzulegen.

Hinweis Es kann immer nur ein COM User Task pro Ressource angelegt werden.

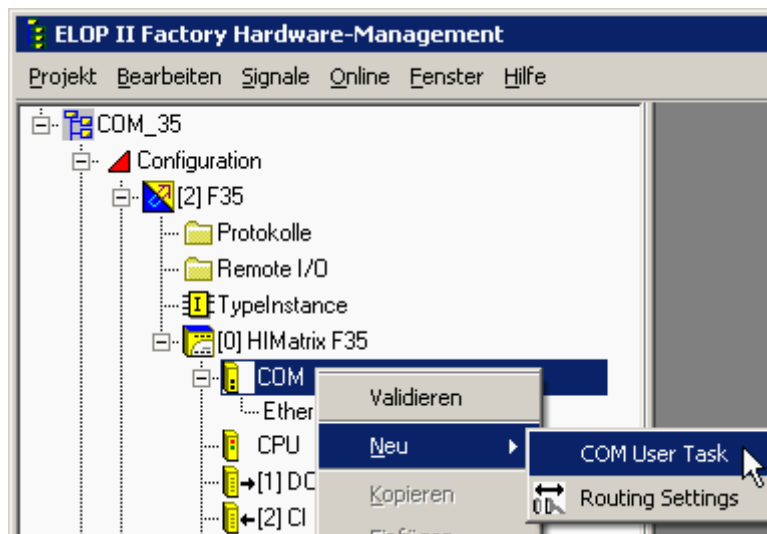


Abbildung 19: Anlage einer COM User Task

5.3.2 Programmcode in das Projekt laden

- Wählen Sie im Kontextmenü des COM User Task den Menüpunkt **User Task laden**. Es öffnet sich ein Standarddialog zum Öffnen einer Datei.
- Öffnen Sie das Verzeichnis `.../cut_src/make/`
- Wählen Sie die ldb-Datei aus, die im COM User Task ausgeführt werden soll.

Hinweis Durch erneutes Laden des ausführbaren Codes (ldb-Datei) können neue Versionen der ldb-Datei übernommen werden. Der Inhalt der ldb-Datei wird beim Laden nicht auf Korrektheit geprüft. Die ldb-Datei wird anschließend zusammen mit der Ressourcekonfiguration im Projekt kompiliert und kann in die Steuerung geladen werden. Wird die ldb-Datei verändert muss das Projekt erneut kompiliert und geladen werden.

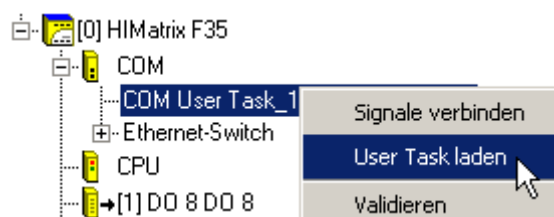


Abbildung 20: Laden einer COM User Task

5.3.3 Signale mit dem CUT verbinden

Der Anwender kann eine nicht sicherheitsgerichtete Prozessdatenkommunikation zwischen der sicheren CPU und der nicht sicheren COM (CUT) definieren. Dabei können in jeder Richtung maximal 16kByte Daten ausgetauscht werden.

5.3.3.1 Signale im Signaleditor erstellen

Erstellen Sie die beiden folgenden Signale im Signaleditor:

Signal	Typ
COM_CPU	UINT
CPU_COM	UINT

5.3.3.2 CUT Dialog „Signal-Zuordnungen“

Wählen Sie im Kontextmenü des **COM User Task** den Menüpunkt **Signale verbinden**, um den Dialog *Signal-Zuordnungen* zu öffnen.

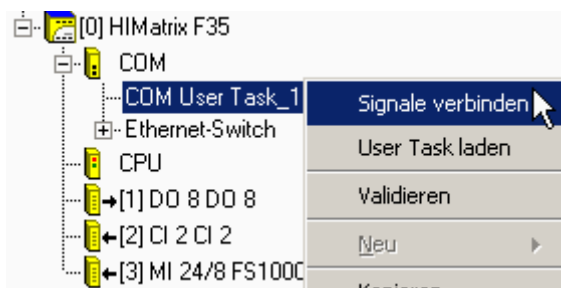


Abbildung 21: Öffnen des Dialogfensters *Signal-Zuordnungen*

5.3.3.3 Konfigurieren der Eingangssignale (COM->CPU)

In das Register **Daten-Eingänge** werden die Signale eingetragen, die von der COM zur CPU übertragen werden sollen (Daten-Eingänge).

- Wählen Sie das Register **Daten-Eingänge**.
- Klicken Sie im Signaleditor auf den Namen des Signals und ziehen Sie das Signal per Drag & Drop in das Register **Daten-Eingänge** des Dialogfensters *Signal-Zuordnungen*.
- Klicken Sie im Dialogfenster „Signal-Zuordnungen“ auf die Schaltfläche **Neue Offsets**.
- Klicken Sie im Popup-Fenster *Offsets nummerieren* auf die Schaltfläche **Nummerieren**.
- Schließen Sie das Dialogfenster.

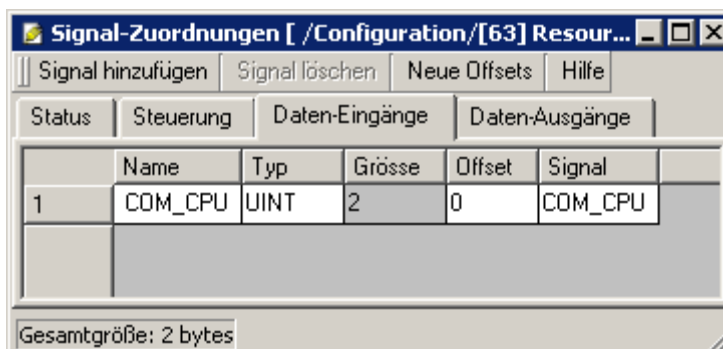


Abbildung 22: Register *Daten-Eingänge* des COM User Task

5.3.3.4 Konfigurieren der Ausgangssignale (CPU->COM)

In das Register **Daten-Ausgänge** werden die Signale eingetragen, die von der CPU zur COM übertragen werden sollen (Daten-Ausgänge).

- Wählen Sie das Register **Daten-Ausgänge**.
- Klicken Sie im Signaleditor auf den Namen des Signals und ziehen Sie das Signal per Drag & Drop in das Register **Daten-Ausgänge** des Dialogfensters *Signal-Zuordnungen*.
- Klicken Sie im Dialogfenster *Signal-Zuordnungen* auf die Schaltfläche **Neue Offsets**.
- Klicken Sie im Popup-Fenster *Offsets nummerieren* auf die Schaltfläche **Nummerieren**.
- Schließen Sie das Dialogfenster.

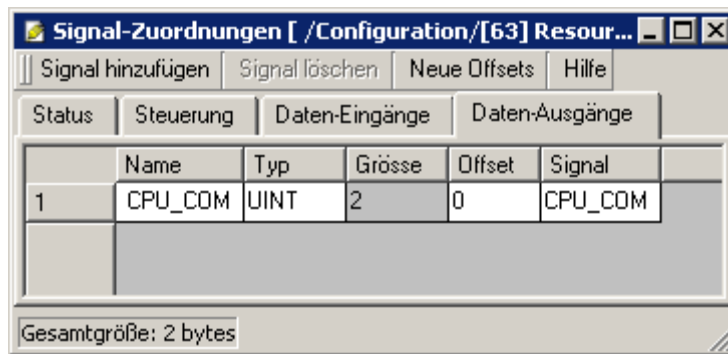


Abbildung 23: Register *Daten-Ausgänge* des COM User Task

5.3.4 Erstellen Sie das ELOP II Factory Anwenderprogramm

Erstellen Sie im Projektmanagement das folgende Anwenderprogramm:

- Öffnen Sie die *Programminstanz* im Projektmanagement.
- Ziehen Sie die Signale **COM_CPU** und **CPU_COM** per Drag & Drop vom Signaleditor in das Anwenderprogramm.
- Erstellen Sie das Anwenderprogramm wie in der folgenden Abbildung dargestellt.
- Schließen Sie die *Programminstanz*.

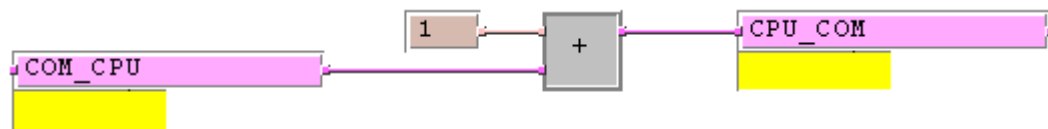


Abbildung 24: ELOP II Factory Anwenderprogramm

Funktion des ELOP II Factory Anwenderprogramm:

Das ELOP II Factory Anwenderprogramm addiert zu dem Signal **COM_CPU** (Daten-Eingänge) den Wert **1** und übergibt das Ergebnis dem Signal **CPU_COM** (Daten-Ausgänge). Beim nächsten CUT-Aufruf (siehe Kapitel 5.3.5) wird das Signal **CPU_COM** an die CUT-Funktion (siehe Kapitel 5.2.3) übergeben. Die COM User Task empfängt das Signal **CPU_COM** und sendet den Wert mit dem Signal **COM_CPU** unverändert zurück.

5.3.5 Konfigurieren Sie das Schedule-Intervall [ms]

- Wählen Sie im Kontextmenü des **COM User Task** den Menüpunkt **Eigenschaften**, um den Dialog *Eigenschaften* zu öffnen.
- Tragen Sie in das Eingabefeld *Schedule-Intervall [ms]* ein, in welchen Intervallen die COM User Task aufgerufen werden soll.

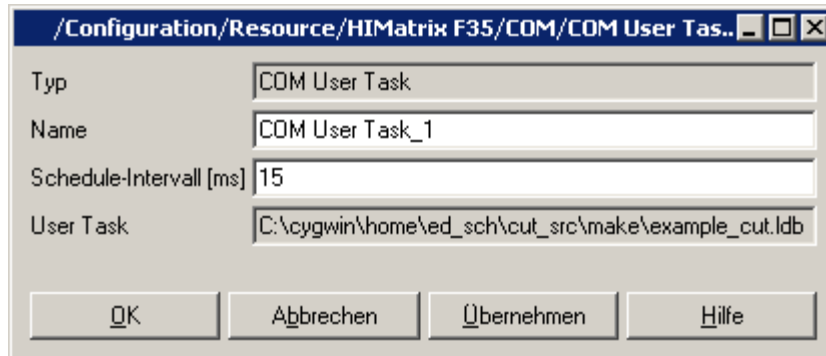


Abbildung 25: Dialog *Eigenschaften* des COM User Task

5.3.6 Erstellen und Laden des Codes in die Steuerung

- Starten Sie den Codegenerator.
- Stellen Sie sicher, dass der Code fehlerfrei generiert wurde (siehe Fehler-Status-Anzeige).
- Öffnen Sie das Control Panel
- Laden Sie den Code in die Steuerung.
- Starten Sie die Steuerung.

5.3.7 Testen der COM User Task

Testen Sie die COM User Task mit dem Online Test:

- Wechseln Sie in das ELOP II Factory Projektmanagement.
- Öffnen Sie über das Kontextmenü der Ressource und wählen Sie **Online-Test**.
- Öffnen Sie die TypeInstance mit einem Doppelklick auf das Symbol der **TypeInstance**.

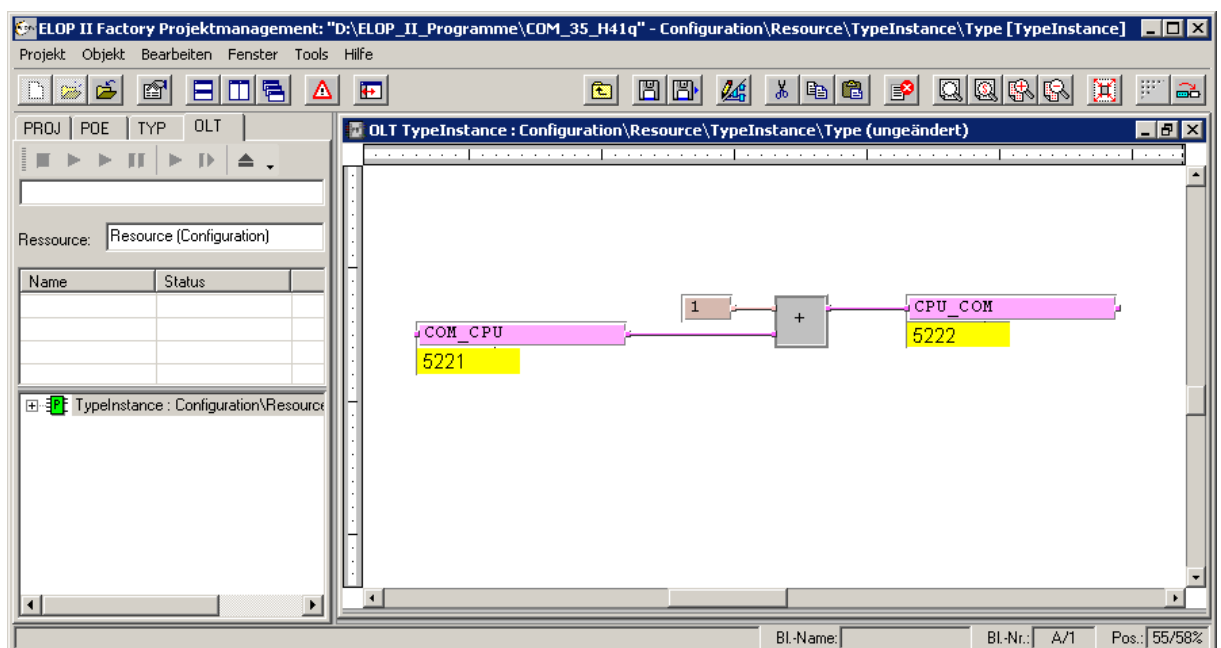


Abbildung 26: Online Test

5.4 Tipps und Tricks

5.4.1 Fehler beim Laden einer Konfiguration mit CUT

Laufzeitprobleme (z.B. COM-User-Task in endlos-Schleife):

Ursache für Laufzeitprobleme:

Wenn in dem betreffenden CUT-Quellcode eine Schleife programmiert wurde, die sehr lange läuft kann dies zu einer „Verklemmung“ des COM-Prozessors führen.

Das hat zur Folge, dass zur Steuerung keine Verbindung mehr hergestellt werden kann und das Löschen der Ressourcenkonfiguration nicht mehr möglich ist.

Lösung: Reset der Steuerung:

- Führen Sie den Reset der Steuerung aus (siehe Datenblatt der Steuerung).
- Löschen Sie die Ressourcenkonfiguration über
Control Panel->Extra->Löschen der Ressourcenkonfiguration
- Erzeugen Sie eine neue CUT (möglichst ohne Laufzeitfehler, Endlosschleife).
- Laden Sie die CUT (ldb-Datei) in das Projekt.
- Generieren Sie den Code.
- Laden Sie die den Code in die Steuerung.
Kann die Steuerung nicht geladen werden, dann schalten Sie die Steuerung noch einmal Aus und wieder Ein. Danach noch einmal den neuen Code laden.

HIMA
...die sichere Entscheidung.



HIMA Paul Hildebrandt GmbH
Industrie-Automatisierung
Postfach 1261 • 68777 Brühl

Telefon: (06202) 709-0 • Telefax: (06202) 709-107
E-mail: info@hima.com • Internet: www.hima.de

(0637)