



Handbuch

Protokolle

ComUserTask



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

HIQuad®, HIQuad®X, HIMax®, HIMatrix®, SILworX®, XMR®, HICore® und FlexSILon® sind eingetragene Warenzeichen der HIMA Paul Hildebrandt GmbH.

Alle technischen Angaben und Hinweise in diesem Handbuch wurden mit größter Sorgfalt erarbeitet und unter Einschaltung wirksamer Kontrollmaßnahmen zusammengestellt. Bei Fragen bitte direkt an HIMA wenden. Für Anregungen, z. B. welche Informationen noch in das Handbuch aufgenommen werden sollen, ist HIMA dankbar.

Technische Änderungen vorbehalten. Ferner behält sich HIMA vor, Aktualisierungen des schriftlichen Materials ohne vorherige Ankündigungen vorzunehmen.

Alle aktuellen Handbücher können über die E-Mail-Adresse documentation@hima.com angefragt werden.

© Copyright 2019, HIMA Paul Hildebrandt GmbH

Alle Rechte vorbehalten.

Kontakt

HIMA Paul Hildebrandt GmbH

Postfach 1261

68777 Brühl

Tel.: +49 6202 709-0

Fax: +49 6202 709-107

E-Mail: info@hima.com

| Revisions- index | Änderungen | Art der Änderung | |
|---------------------|---|------------------|--------------|
| | | technisch | redaktionell |
| 11.00 | Erstausgabe des Handbuchs für SILworX V11 | | |
| | | | |
| | | | |
| | | | |

Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | Einleitung | 5 |
| 1.1 | Aufbau und Gebrauch des Handbuchs | 5 |
| 1.2 | Zielgruppe | 6 |
| 1.3 | Darstellungskonventionen | 7 |
| 1.3.1 | Sicherheitshinweise | 7 |
| 1.3.2 | Gebrauchshinweise | 8 |
| 1.4 | Safety Lifecycle Services | 9 |
| 2 | Sicherheit | 10 |
| 2.1 | Bestimmungsgemäßer Einsatz | 10 |
| 2.2 | Restrisiken | 10 |
| 2.3 | Sicherheitsvorkehrungen | 10 |
| 2.4 | Notfallinformationen | 10 |
| 2.5 | Automation Security bei HIMA Systemen | 10 |
| 3 | ComUserTask | 12 |
| 3.1 | Systemanforderung | 12 |
| 3.2 | Eigenschaften der ComUserTask: | 12 |
| 3.2.1 | Anlegen einer ComUserTask | 13 |
| 3.3 | Entwicklungsumgebung für ComUserTask | 13 |
| 3.4 | Abkürzungen | 13 |
| 3.5 | CUT-Schnittstelle in SILworX | 14 |
| 3.5.1 | Schedule-Intervall [ms] | 14 |
| 3.5.2 | Scheduling-Vorlauf | 14 |
| 3.5.3 | Scheduling-Nachlauf | 14 |
| 3.5.4 | STOP_INVALID_CONFIG | 15 |
| 3.5.5 | Variablen der CUT-Schnittstelle (CPU<->CUT) | 15 |
| 3.5.6 | Menüfunktion Eigenschaften | 16 |
| 3.5.7 | Menüfunktion Edit | 17 |
| 3.6 | CUT Funktionen | 19 |
| 3.6.1 | COM-User-Callback-Funktionen | 19 |
| 3.6.2 | COM-User-Library-Funktionen | 19 |
| 3.6.3 | Header-Files | 19 |
| 3.6.4 | Code-/Datenbereich und Stack für die CUT | 20 |
| 3.6.5 | Startfunktion CUCB_TaskLoop | 20 |
| 3.6.6 | Serielle Schnittstellen RS485 / RS232 IF | 21 |
| 3.6.7 | UDP/TCP-Socket-IF | 29 |
| 3.6.8 | Timer-IF | 43 |
| 3.6.9 | Diagnose | 44 |
| 3.7 | Funktionen für HIMatrix F*03 und HM31 | 45 |
| 3.7.1 | COM-User-IRQ-Task | 45 |
| 3.7.2 | NVRAM-IF | 48 |
| 3.7.3 | Semaphore-IF | 49 |
| 3.8 | Installation der Entwicklungsumgebung | 52 |
| 3.8.1 | Installation der Cygwin-Umgebung | 52 |
| 3.8.2 | Installation des GNU Compilers | 54 |

| | | |
|------------|---|-----------|
| 3.9 | Neues CUT-Projekt anlegen | 56 |
| 3.9.1 | CUT-Makefiles | 57 |
| 3.9.2 | C-Quellcode bearbeiten | 60 |
| 3.9.3 | ComUserTask in das Projekt einbinden | 64 |
| 3.9.4 | Fehler beim Laden einer Konfiguration mit CUT | 67 |
| 4 | Synchron Serielle Schnittstelle | 68 |
| 4.1 | Systemanforderung | 68 |
| 4.2 | Blockschaltbild | 68 |
| 4.3 | D-Sub-Buchsen FB1 und FB2 | 69 |
| 4.4 | Konfiguration zwischen COM und SSI-Submodul | 69 |
| 4.5 | Konfiguration der SSI-Schnittstelle | 69 |
| 4.5.1 | Leitungslänge und empfohlene Taktraten | 70 |
| 4.6 | Applikationshinweise | 71 |
| 5 | Allgemein | 72 |
| 5.1 | Maximale Kommunikationszeitscheibe | 72 |
| 5.1.1 | Ermitteln der maximalen Dauer der Kommunikationszeitscheibe | 72 |
| 5.2 | Lastbegrenzung | 72 |
| | Anhang | 75 |
| | Glossar | 75 |
| | Abbildungsverzeichnis | 76 |
| | Tabellenverzeichnis | 76 |

1 Einleitung

Das ComUserTask Handbuch beschreibt die Eigenschaften und die Konfiguration der ComUserTask für die sicherheitsbezogenen HIMA Steuerungssysteme mit dem Programmierwerkzeug SILworX.

Die Kenntnis von Vorschriften und das technisch einwandfreie Umsetzen der in diesem Handbuch enthaltenen Hinweise durch qualifiziertes Personal sind Voraussetzung für die Planung, Projektierung, Programmierung, Installation, Inbetriebnahme, Betrieb und Instandhaltung der HIMA Steuerungen.

Bei nicht qualifizierten Eingriffen in die Geräte, bei Abschalten oder Umgehen (Bypass) von Sicherheitsfunktionen oder bei Nichtbeachtung von Hinweisen dieses Handbuchs (und dadurch verursachten Störungen oder Beeinträchtigungen von Sicherheitsfunktionen) können schwere Personen-, Sach- oder Umweltschäden eintreten, für die HIMA keine Haftung übernehmen kann.

HIMA Automatisierungsgeräte werden unter Beachtung der einschlägigen Sicherheitsnormen entwickelt, gefertigt und geprüft. Nur für die in den Beschreibungen vorgesehenen Einsatzfälle mit den spezifizierten Umgebungsbedingungen verwenden.

1.1 Aufbau und Gebrauch des Handbuchs

Das Handbuch enthält die folgenden Hauptkapitel:

- Einleitung
- Sicherheit
- Produktbeschreibung
- Beschreibung der Konfiguration der ComUserTask in SILworX

Zusätzlich sind die folgenden Dokumente zu beachten:

| Name | Inhalt | Dokumenten-Nr. |
|---------------------------------|--|----------------|
| HIMax Systemhandbuch | Hardware-Beschreibung HIMax System | HI 801 000 D |
| HIMax Sicherheitshandbuch | Sicherheitsfunktionen HIMax Systems | HI 801 002 D |
| HIMatrix Sicherheitshandbuch | Sicherheitsfunktionen HIMatrix Systems | HI 800 022 D |
| HIMatrix Kompakt Systemhandbuch | Hardware-Beschreibung HIMatrix Kompakt System | HI 800 140 D |
| HIMatrix Modular Systemhandbuch | Hardware-Beschreibung HIMatrix Modular System F 60 | HI 800 190 D |
| HIQuad X Systemhandbuch | Hardware-Beschreibung HIQuad X System | HI 803 210 D |
| HIQuad X Sicherheitshandbuch | Sicherheitsfunktionen HIQuad X System | HI 803 208 D |
| Automation Security Handbuch | Beschreibung von Automation Security Aspekten bei HIMA Systemen | HI 801 372 D |
| SILworX Erste Schritte | Einführung in SILworX | HI 801 102 D |

Tabelle 1: Zusätzlich geltende Handbücher

Alle aktuellen Handbücher können über die E-Mail-Adresse documentation@hima.com angefragt werden. Für registrierte Kunden stellt HIMA die Dokumentationen im Download-Bereich <https://www.hima.com/de/downloads/> zur Verfügung.

1.2 Zielgruppe

Dieses Dokument wendet sich an Planer, Projektoren, Programmierer und Personen die zur Inbetriebnahme, zur Wartung und zum Betreiben von Automatisierungsanlagen berechtigt sind. Vorausgesetzt werden spezielle Kenntnisse auf dem Gebiet der sicherheitsbezogenen Automatisierungssysteme.

1.3 Darstellungskonventionen

Zur besseren Lesbarkeit und zur Verdeutlichung gelten in diesem Dokument folgende Schreibweisen:

| | |
|----------------------|---|
| Fett | Hervorhebung wichtiger Textteile. Bezeichnungen von Schaltflächen, Menüpunkten und Registern im Programmierwerkzeug, die angeklickt werden können. |
| <i>Kursiv</i> | Parameter und Systemvariablen, Referenzen. |
| <code>Courier</code> | Wörtliche Benutzereingaben. |
| RUN | Bezeichnungen von Betriebszuständen (Großbuchstaben). |
| Kap. 1.2.3 | Querverweise sind Hyperlinks, auch wenn sie nicht besonders gekennzeichnet sind. Im elektronischen Dokument (PDF): Wird der Mauszeiger auf einen Hyperlink positioniert, verändert er seine Gestalt. Bei einem Klick springt das Dokument zur betreffenden Stelle. |

Sicherheits- und Gebrauchshinweise sind besonders gekennzeichnet.

1.3.1 Sicherheitshinweise

Um ein möglichst geringes Risiko zu gewährleisten, sind die Sicherheitshinweise unbedingt zu befolgen.

Die Sicherheitshinweise im Dokument sind wie folgt dargestellt.

- Signalwort: Warnung, Vorsicht, Hinweis.
- Art und Quelle des Risikos.
- Folgen bei Nichtbeachtung.
- Vermeidung des Risikos.

Die Bedeutung der Signalworte ist:

- Warnung: Bei Missachtung droht schwere Körperverletzung bis Tod.
- Vorsicht: Bei Missachtung droht leichte Körperverletzung.
- Hinweis: Bei Missachtung droht Sachschaden.

SIGNALWORT



Art und Quelle des Risikos!
Folgen bei Nichtbeachtung.
Vermeidung des Risikos.

HINWEIS



Art und Quelle des Schadens!
Vermeidung des Schadens.

1.3.2 Gebrauchshinweise

Zusatzinformationen sind nach folgendem Beispiel aufgebaut:

i

An dieser Stelle steht der Text der Zusatzinformation.

Nützliche Tipps und Tricks erscheinen in der Form:

TIPP

An dieser Stelle steht der Text des Tipps.

1.4 Safety Lifecycle Services

HIMA unterstützt Sie in allen Phasen des Sicherheitslebenszyklus der Anlage: Von der Planung, der Projektierung, über die Inbetriebnahme, bis zur Aufrechterhaltung der Sicherheit.

Für Informationen und Fragen zu unseren Produkten, zu Funktionaler Sicherheit und zu Automation Security stehen Ihnen die Experten des HIMA Support zur Verfügung.

Für die geforderte Qualifizierung gemäß Sicherheitsstandards, führt HIMA produkt- oder kundenspezifische Seminare in eigenen Trainingszentren, oder bei Ihnen vor Ort durch. Das aktuelle Seminarangebot zu Funktionaler Sicherheit, Automation Security und zu HIMA Produkten finden Sie auf der HIMA Webseite.

Safety Lifecycle Services:

| | |
|--|--|
| Onsite+ / Vor-Ort-Engineering | In enger Abstimmung mit Ihnen führt HIMA vor Ort Änderungen oder Erweiterungen durch. |
| Startup+ / Vorbeugende Wartung | HIMA ist verantwortlich für die Planung und Durchführung der vorbeugenden Wartung. Wartungsarbeiten erfolgen gemäß der Herstellervorgabe und werden für den Kunden dokumentiert. |
| Lifecycle+ / Lifecycle-Management | Im Rahmen des Lifecycle-Managements analysiert HIMA den aktuellen Status aller installierten Systeme und erstellt konkrete Empfehlungen zu Wartung, Upgrade und Migration. |
| Hotline+ / 24-h-Hotline | HIMA Sicherheitsingenieure stehen Ihnen für Problemlösung rund um die Uhr telefonisch zur Verfügung. |
| Standby+ / 24-h-Rufbereitschaft | Fehler, die nicht telefonisch gelöst werden können, werden von HIMA Spezialisten innerhalb vertraglich festgelegter Zeitfenster bearbeitet. |
| Logistic+/ 24-h-Ersatzteilservice | HIMA hält notwendige Ersatzteile vor und garantiert eine schnelle und langfristige Verfügbarkeit. |

Ansprechpartner:

| | |
|----------------------------------|---|
| Safety Lifecycle Services | https://www.hima.com/de/unternehmen/ansprechpartner-weltweit/ |
| Technischer Support | https://www.hima.com/de/produkte-services/support/ |
| Seminarangebot | https://www.hima.com/de/produkte-services/seminarangebot/ |

2 Sicherheit

Sicherheitsinformationen, Hinweise und Anweisungen in diesem Dokument unbedingt lesen. Das Produkt nur unter Beachtung aller Richtlinien und Sicherheitsrichtlinien einsetzen.

Dieses Produkt wird mit SELV oder PELV betrieben. Vom Produkt selbst geht kein Risiko aus. Einsatz im Ex-Bereich nur mit zusätzlichen Maßnahmen erlaubt.

2.1 Bestimmungsgemäßer Einsatz

Für den Einsatz von HIMA Steuerungen sind die jeweiligen Bedingungen einzuhalten, siehe zusätzlich geltende Handbücher Tabelle 1.

2.2 Restrisiken

Von einem HIMA System selbst geht kein Risiko aus.

Restrisiken können ausgehen von:

- Fehlern in der Projektierung.
- Fehlern im Anwenderprogramm.
- Fehlern in der Verdrahtung.

2.3 Sicherheitsvorkehrungen

Am Einsatzort geltende Sicherheitsbestimmungen beachten und vorgeschriebene Schutzausrüstung tragen.

2.4 Notfallinformationen

Ein HIMA System ist Teil der Sicherheitstechnik einer Anlage. Der Ausfall einer Steuerung bringt die Anlage in den sicheren Zustand.

Im Notfall ist jeder Eingriff, der die Sicherheitsfunktion des HIMA Systems verhindert, verboten.

2.5 Automation Security bei HIMA Systemen

Industrielle Steuerungen müssen gegen IT-typische Problemquellen geschützt werden. Diese Problemquellen sind:

- Angreifer innerhalb und außerhalb der Kundenanlage
- Bedienungsfehler
- Software-Fehler

Die Anforderungen der Sicherheits- und Anwendungsnormen bezüglich des Schutzes vor Manipulationen sind zu beachten. Die Autorisierung von Personal und die notwendigen Schutzmaßnahmen unterliegen der Verantwortung des Betreibers.

WARNUNG



Personenschaden durch unbefugte Manipulation an der Steuerung möglich!

Die Steuerung ist gegen unbefugte Zugriffe zu schützen!

Beispielsweise:

- die Standardeinstellungen für Login und Passwort ändern.
- physischen Zugang zur Steuerung und zum PADT kontrollieren!

Sorgfältige Planung sollte die zu ergreifenden Maßnahmen nennen. Nach erfolgter Risikoanalyse sind die benötigten Maßnahmen zu ergreifen. Solche Maßnahmen sind beispielsweise:

- Sinnvolle Einteilung von Benutzergruppen.
- Gepflegte Netzwerkpläne helfen sicherzustellen, dass secure Netzwerke dauerhaft von öffentlichen Netzwerken getrennt sind und, falls nötig, nur ein definierter Übergang (z. B. über eine Firewall oder eine DMZ) besteht.
- Verwendung geeigneter Passwörter.

Ein regelmäßiges Review (z. B. jährlich) der Security-Maßnahmen ist ratsam.

Die für eine Anlage geeignete Umsetzung der benötigten Maßnahmen liegt in der Verantwortung des Anwenders!

Weitere Einzelheiten siehe HIMA Automation-Security Handbuch HI 801 372 D.

3 ComUserTask

Neben dem Logikprogramm, das mit SILworX erstellt wird, kann zusätzlich ein C-Programm des Anwenders mit Anbindung an diverse Kommunikationsschnittstellen des COM-Moduls implementiert werden.

Dieses nicht sichere C-Programm läuft als ComUserTask auf dem Kommunikationsmodul rückwirkungsfrei zum sicheren Prozessormodul der Steuerung.

Die ComUserTask hat einen eigenen Zyklus, der unabhängig vom Zyklus der CPU ist. Damit können beliebige Anwendungen in C erstellt und als ComUserTask implementiert werden z. B.:

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

3.1 Systemanforderung

Benötigte Ausstattung und Systemanforderung:

| Element | Beschreibung |
|-------------|---|
| Steuerung | HIMax mit COM-Modul HIQuad X mit COM-Modul HIMatrix F*03 HIMatrix F*01/02 |
| CPU-Modul | Die Ethernet-Schnittstellen des Prozessormoduls können für ComUserTask nicht verwendet werden. |
| COM-Modul | Ethernet 10/100BaseT D-Sub Anschlüsse FB1 und FB2 z. B. für RS232 Werden die seriellen Feldbus-Schnittstelle (FB1 oder FB2) oder FB3 bei HIMatrix (RS485) verwendet, müssen diese mit einem optionalen HIMA Submodul ausgerüstet sein, siehe Kommunikationshandbuch HI 801 100 D. |
| Aktivierung | Die Freischaltung erfolgt per Software-Freischaltcode, siehe Kommunikationshandbuch HI 801 100 D. |

Tabelle 2: Systemanforderung und Ausstattung ComUserTask

3.2 Eigenschaften der ComUserTask:

| Element | Beschreibung |
|------------------------|---|
| ComUserTask | Es kann für jede HIMax X-COM 01, HIQuad X F-COM 01 oder HIMatrix COM eine ComUserTask konfiguriert werden, siehe Kommunikationshandbuch HI 801 100 D. |
| Sicherheitsbezogen | Nein |
| Datenaustausch | Konfigurierbar (Feldbus- und/oder Ethernet-Schnittstelle) |
| Code- und Datenbereich | Siehe Kapitel 3.6.4 |
| Stack | Der Stack liegt in einem dafür reservierten Speicher außerhalb des Code-/Datenbereichs. Siehe Kapitel 3.6.4 |

Tabelle 3: Eigenschaften ComUserTask

3.2.1 Anlegen einer ComUserTask

Eine neue ComUserTask anlegen:

1. Im Strukturbaum **Konfiguration, Ressource, Protokolle** selektieren.
2. Im Kontextmenü von Protokolle **Neu**, ComUserTask wählen, um eine neue ComUserTask hinzuzufügen.
3. Im Kontextmenü der ComUserTask **Eigenschaften** das **COM-Modul** auswählen.
Die Standardeinstellungen können für die erste Konfiguration beibehalten werden.

3.3 Entwicklungsumgebung für ComUserTask

Für die Programmierung der ComUserTask steht zusätzlich zum normalen Befehlsumfang von C eine eigene Library (siehe Kapitel 3.5) mit definierten Funktionen zur Verfügung.

Entwicklungsumgebung

Im Kapitel 3.8 wird die Installation der Entwicklungsumgebung und die Erstellung einer ComUserTask beschrieben.

Zu der Entwicklungsumgebung gehören der GNU C Compiler und Cygwin, die den Bedingungen der GNU General Public License unterliegen (siehe www.gnu.org).

Die neueste Entwicklungsumgebung befindet sich jeweils auf der aktuellen HIMA DVD.

Steuerung

Bei den HIMA Steuerungen hat die ComUserTask 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 Variablen erforderlich, siehe Kapitel 3.5.5.

3.4 Abkürzungen

| Abkürzung | Bedeutung |
|-----------|---|
| CUCB | COM User Callback (CUCB_ Funktionen werden von der COM aufgerufen) |
| CUIT | COM User IRQ Task |
| CUL | COM User Library (CUL_ Funktionen werden in der CUT aufgerufen) |
| CUT | ComUserTask |
| CUT_PDI | CUT Prozessdaten-Eingangsbereich auf der COM, der die Daten enthält die von der CPU geschrieben werden. |
| CUT_PDO | CUT Prozessdaten-Ausgangsbereich auf der COM, der die Daten enthält die von der CPU gelesen werden. |
| GNU | GNU Projekt |
| IF | InterFace |
| FB | Feldbus-Schnittstelle der Steuerung |
| FIFO | First In First Out (Datenspeicher) |
| NVRAM | Non Volatile Random Access Memory, nicht flüchtiger Speicher |
| SSI | S ynchron S erial I nterface |

Tabelle 4: Abkürzungen

3.5 CUT-Schnittstelle in SILworX

Die Prozessdatenkommunikation der ComUserTask läuft zwischen der COM und der CPU ab.

WARNUNG



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.

3.5.1 Schedule-Intervall [ms]

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

Das Schedule-Intervall [ms] wird in SILworX in den *Eigenschaften* der ComUserTask eingestellt.

| Schedule-Intervall [ms] | |
|-------------------------|--------------|
| Wertebereich: | 2 ... 255 ms |
| Standardwert: | 15 ms |

Tabelle 5: Schedule-Intervall [ms]

·
i

Die der CUT verfügbare COM-Prozessor-Zeit hängt von den anderen parametrierten Funktionen der COM, wie z. B. safe**ethernet**, 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.

3.5.2 Scheduling-Vorlauf

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 Prozessdaten-Austausch von der COM zur sicheren CPU statt.

3.5.3 Scheduling-Nachlauf

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 Prozessdaten-Austausch von der COM zur sicheren CPU statt.

3.5.4 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.

3.5.5 Variablen der CUT-Schnittstelle (CPU<->CUT)

Parametrierung einer nicht sicherheitsgerichtete Prozessdatenkommunikation zwischen sicherer CPU und COM (CUT). Dabei kann je Datenrichtung die maximale Prozessdatenmenge ausgetauscht werden.

Die maximale Prozessdatenmenge für Standardprotokolle ist vom Steuerungs-Typ abhängig, siehe Kapitel 3.6.4.



1 CPU (sicheres Anwenderprogramm)
2 CUT Prozessdaten-Eingangsbereich (CUT_PDI)

3 COM (CUT)
4 CUT Prozessdaten-Ausgangsbereich (CUT_PDO)

Bild 1: Prozessdaten-Austausch zwischen CPU und COM (CUT)

Es können alle Datentypen ausgetauscht werden, die in SILworX verwendet werden.

Die Struktur der Daten muss in SILworX parametriert werden.

Die Größe der Datenstrukturen CUT_PDI und CUT_PDO (im compilierten C-Code der CUT) müssen den gleichen Größen der konfigurierten Datenstrukturen in SILworX entsprechen.

i

Sind im compilierten C-Code die Datenstrukturen CUT_PDI und CUT_PDO nicht vorhanden oder haben nicht die gleiche Größe wie die in SILworX parametrierten Prozessdaten, dann ist die Konfiguration ungültig und die COM nimmt den Zustand STOP_INVALID_CONFIG an. Die Prozessdatenkommunikation findet nur im Betriebsmodus RUN statt.

3.5.6 Menüfunktion Eigenschaften

Die Menüfunktion **Eigenschaften** aus dem Kontextmenü der ComUserTask öffnet den Dialog **Eigenschaften**.

| Element | Beschreibung |
|--|--|
| Typ | ComUserTask |
| Name | Beliebiger, eindeutiger Name, für eine ComUserTask |
| Aktualisierungsintervall der Prozessdaten [ms] | Aktualisierungszeit in Millisekunden, mit der die Prozessdaten des Protokolls zwischen Kommunikationsmodul und Prozessormodul ausgetauscht werden. Ist das <i>Aktualisierungsintervall der Prozessdaten</i> Null oder kleiner als die Zykluszeit der Steuerung, dann erfolgt der Datenaustausch so schnell wie möglich. Wertebereich: $(2^{31}-1)$ ms Standardwert: 0 |
| Mehrere Fragmente pro Zyklus zulassen | Aktiviert: Transfer der gesamten Prozessdaten des Protokolls zwischen Kommunikationsmodul und Prozessormodul innerhalb eines Zyklus des Prozessormoduls. Deaktiviert: Transfer der gesamten Prozessdaten des Protokolls zwischen Kommunikationsmodul und Prozessormodul, verteilt über mehrere Zyklen des Prozessormoduls. Standardwert: Aktiviert |
| Modul | Auswahl des COM-Moduls auf dem dieses Protokoll abgearbeitet wird. |
| Max. μ P-Budget aktivieren ¹⁾ | Aktiviert: Limit des μ P-Budget aus dem Feld <i>Max. μP-Budget in [%]</i> übernehmen. Deaktiviert: Kein Limit des μ P-Budget, für dieses Protokoll verwenden. Standardwert: Aktiviert |
| Max. μ P-Budget in [%] ¹⁾ | Maximales μ P-Budget des Moduls, welches bei der Abarbeitung des Protokolls produziert werden darf. Wertebereich: 1 ... 100 % Standardwert: 30 % |
| Verhalten bei CPU/COM Verbindungsverlust | Bei Verbindungsverlust des Prozessormoduls zum Kommunikationsmodul werden in Abhängigkeit dieses Parameters die Eingangsvariablen entweder initialisiert oder unverändert im Prozessormodul verwendet. (z. B. wenn Kommunikationsmodul bei laufender Kommunikation gezogen wird). Soll ein Projekt von kleiner SILworX V3 konvertiert werden, muss dieser Wert auf "Letzten Wert beibehalten" gesetzt sein, um den CRC nicht zu ändern. Für HIMatrix Steuerungen kleiner CPU BS V8 muss dieser Wert immer auf "Letzten Wert beibehalten" gesetzt sein. Initialdaten annehmen Eingangsvariablen werden auf die Initialwerte zurückgesetzt. Letzten Wert behalten Eingangsvariablen behalten den letzten Wert. Standardwert: Letzten Wert behalten |
| Schedule-Intervall [ms] | Die ComUserTask wird in einem parametrierten Schedule-Intervall[ms] der Steuerung (COM-Modul) aufgerufen, siehe Kapitel 3.5.1. Wertebereich: 2 ... 255 ms Standardwert: 15 ms |
| User Task | Pfad zur Loadable falls bereits geladen |
| ¹⁾ Funktion verfügbar für HIQuad X, HIMax mit COM Betriebssystem ab V6 und HIMatrix mit COM Betriebssysteme ab V15. | |

Tabelle 6: Allgemeine Eigenschaften der CUT

3.5.7 Menüfunktion Edit

Die Menüfunktion **Edit** aus dem Kontextmenü der ComUserTask öffnet den Dialog **Edit**. In diesem sind die Register *Prozessvariablen* und *Systemvariablen* enthalten.

3.5.7.1 Systemvariablen

Das Register **Systemvariablen** enthält die folgenden Systemparameter zur Überwachung und Steuerung des CUT:

| Name | Funktion | | | | | | | | | | |
|--|---|----------|--------------|--------------------|---|-------------------------|--|-------------------------|--|--------------------------|--|
| Ausführungszeit [DWORD] | Ausführungszeit der ComUserTask in µs | | | | | | | | | | |
| Reales Schedule-Intervall [DWORD] | Zeitabstand zwischen zwei ComUserTask Durchläufen in ms | | | | | | | | | | |
| Steuerung des User Task-Zustands [WORD] | <p>Die folgende Tabelle zeigt die Möglichkeiten, wie der Anwender mit dem Systemparameter <i>Steuerung des User Task-Zustandes</i> die ComUserTask steuern kann:</p> <table> <tr> <th>Funktion</th><th>Beschreibung</th></tr> <tr> <td>DISABLED 0x8000</td><td>Das Anwenderprogramm sperrt die CUT (d. h. die CUT wird nicht gestartet).</td></tr> <tr> <td>TOGGLE_MODE_0 0x0100</td><td>Nach Terminierung der CUT ist ein Start der CUT erst wieder nach dem Setzen von TOGGLE_MODE_1 erlaubt.</td></tr> <tr> <td>TOGGLE_MODE_1 0x0101</td><td>Nach Terminierung der CUT ist ein Start der CUT erst wieder nach dem Setzen von TOGGLE_MODE_0 erlaubt.</td></tr> <tr> <td>AUTOSTART 0 (Default)</td><td>Nach Terminierung der CUT startet die CUT automatisch nachdem die Störung oder der Fehler beseitigt wurde.</td></tr> </table> | Funktion | Beschreibung | DISABLED 0x8000 | Das Anwenderprogramm sperrt die CUT (d. h. die CUT wird nicht gestartet). | 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. | AUTOSTART 0 (Default) | Nach Terminierung der CUT startet die CUT automatisch nachdem die Störung oder der Fehler beseitigt wurde. |
| Funktion | Beschreibung | | | | | | | | | | |
| DISABLED 0x8000 | Das Anwenderprogramm sperrt die CUT (d. h. die CUT wird nicht gestartet). | | | | | | | | | | |
| 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. | | | | | | | | | | |
| AUTOSTART 0 (Default) | Nach Terminierung der CUT startet die CUT automatisch nachdem die Störung oder der Fehler beseitigt wurde. | | | | | | | | | | |
| Zustand der User Task [BYTE] | 1 = RUNNING (CUT läuft) 0 = ERROR (CUT läuft nicht wegen eines Fehlers) | | | | | | | | | | |

Tabelle 7: Systemvariablen der ComUserTask

TIPP

Wird die CUT terminiert und neu gestartet, wird kurzzeitig der COM-Zustand *Stopp/übernehme Daten aus Flash* aus dem Flash angezeigt, obwohl sich die CUT im Zustand RUN befindet.

3.5.7.2 Prozessvariablen

Eingangssignale (COM->CPU)

In das Register **Eingangssignale** werden die Variablen eingetragen, die von der COM (CUT) zur CPU übertragen werden sollen (Eingangsbereich der CPU).

VORSICHT



Die ComUserTask ist nicht sicherheitsbezogen!

Die nicht sicheren Variablen der ComUserTask dürfen nicht für die Sicherheitsfunktionen des CPU Anwenderprogramms verwendet werden.

Erforderlicher Eintrag im C-Code

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

```
/* SILworX Input Records (COM->CPU) */  
uword CUT_PDO[1] __attribute__((section("CUT_PD_OUT_SECT"), aligned(1)));
```

Die Größe der Datenstruktur CUT_PDO muss der Größe der konfigurierten Daten-Eingänge in SILworX entsprechen.

Ausgangssignale (CPU->COM)

In das Register **Ausgangssignale** werden die Variablen eingetragen, die von der CPU (Ausgangsbereich der CPU) zur COM (CUT) übertragen werden sollen.

Erforderlicher Eintrag im C-Code

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

```
/* SILworX Output Records (CPU->COM) */  
uword CUT_PDI[1] __attribute__((section("CUT_PD_IN_SECT"), aligned(1)));
```

Die Größe der Datenstruktur CUT_PDI muss der Größe der konfigurierten Daten-Ausgänge in SILworX entsprechen.

3.6 CUT Funktionen

3.6.1 COM-User-Callback-Funktionen

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

i

Alle COM-User-Callback Funktionen müssen im C-Code des Anwenders definiert werden! Auch die Funktion CUCB_IrqService die für HIMax und HIMatrix F*01/02 nicht unterstützt wird (nur für HIMatrix F*03 mit CAN-Modul), muss im C-Code des Anwenders aus Kompatibilitätsgründen definiert werden.

Funktionsprototyp: void CUCB_IrqService(udword devNo) {}

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.

3.6.2 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.

3.6.3 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
```

3.6.4 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 (Gilt für HIMA und HM31).

| Element | HIMax ab V.4 | HIMax vor V.4, HIMatrix F*01/02 | HIMatrix F*03, HIQuad X |
|--------------|--------------|---------------------------------|-------------------------|
| Startadresse | 0x780000 | 0x790000 | 0x800000 |
| Länge | 512 kByte | 448 kByte | 4 MB |

Tabelle 8: Speicherbereich für Code und Daten

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

| Element | HIMax, HIMatrix F*01/02 | HIMatrix F*03, HIQuad X |
|------------|-----------------------------|-----------------------------|
| Endadresse | Dynamisch aus Sicht der CUT | Dynamisch aus Sicht der CUT |
| Länge | ca. 64 kByte | ca. 500 kByte |

Tabelle 9: Stack-Speicher

3.6.5 Startfunktion CUCB_TaskLoop

Die Funktion `CUCB_TaskLoop()` ist die Startfunktion zur ComUserTask.

Die Programmausführung der ComUserTask beginnt mit dem Aufruf dieser Funktion (siehe `Schedule-Intervall [ms]` Kapitel 3.5.1).

Funktionsprototyp:

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

Parameter:

Die Funktion hat den folgenden Parameter (liefert Wert zurück)

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

Tabelle 10: Parameter CUCB_TaskLoop

3.6.6 Serielle Schnittstellen RS485 / RS232 IF

Die verwendeten Feldbus-Schnittstellen müssen mit dem entsprechenden Feldbus Submodul (Hardware) ausgestattet sein, siehe Systemdokumentation HIMA.

Telegramme empfangen aus Sicht der CUT Applikation

Die Anzahl der Idle Zeichen zur Erkennung von Telegrammgrenzen (Idle Zeit) ist für den seriellen Empfang über die ComUserTask auf 5 Zeichen fest eingestellt.

Ein Kommunikationspartner, welcher der CUT Applikation mehrere Telegramme hintereinander sendet, muss mindestens 5 Idle Zeichen zwischen den gesendeten Telegrammen einfügen. Somit erkennt der UART Treiber diese Telegramme als einzelne Telegramme.

Die vom UART Treiber empfangene Telegramme werden erst nach Empfang von mindestens 5 Idle Zeichen an die CUT Applikation weitergeleitet.

Telegramme senden aus Sicht der CUT Applikation

Die Anzahl der Idle Zeichen zur Erkennung von Telegrammgrenzen (Idle Zeit) ist für das serielle senden über die ComUserTask auf 5 Zeichen fest eingestellt.

Die ComUserTask Applikationsschnittstelle sorgt dafür, dass die von der Applikation direkt hintereinander gesendeten seriellen Telegramme mindestens die Idle Zeit als Abstand einhalten. Hierbei ist zu beachten, dass das Speichervermögen der CUT für zu sendende Telegramme auf ein Telegramm limitiert ist.

Erfolgt das Senden (Aufruf der Sendeschnittstelle) in kürzeren Abständen als die physikalische Übertragung benötigt, wird das Speichervermögen der CUT überschritten und der Fehlercode CUL_WOULDBLOCK wird zurückgegeben, siehe Kapitel 3.6.6.5.

3.6.6.1 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 1 kByte je initialisierter serieller Schnittstelle gespeichert.

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

Neu empfangenen Daten werden verworfen, wenn das Auslesen der Daten aus dem FIFO langsamer ist als der Empfang neuer Daten.

Funktionsprototyp:

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

Parameter:

Die Funktion hat die folgenden Parameter:

| Parameter | Beschreibung |
|---|---|
| comId | Feldbus-Schnittstelle (RS485, RS232) 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 (maximale baud Rate HIMax vor V.4) 7 = 57600 Bit (maximale baud Rate HIMax ab V.4) 8 = 115000 Bit (nur HIMatrix) |
| 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 |

Tabelle 11: Parameter CUL_AscOpen

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 |

Tabelle 12: Rückgabewert CUL_AscOpen

3.6.6.2 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 |
|--------------|--|
| <i>comId</i> | Feldbus-Schnittstelle (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE |

Tabelle 13: Parameter `CUL_AscClose`

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> | Die Schnittstelle wurde erfolgreich geschlossen. |
| <code>CUL_NOT_OPENED</code> | Die Schnittstelle war nicht (durch die CUT) geöffnet. |
| <code>CUL_INVALID_PARAM</code> | Es wurden unzulässige Parameter oder Parameterkombinationen übergeben. |
| <code>CUL_DEVICE_ERROR</code> | Sonstige Fehler |

Tabelle 14: Rückgabewert `CUL_AscClose`

3.6.6.3 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 und 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.

i

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 reserved1;   // 2 unbenutzte bytes
    udword mDataIdx;   // Byte-Offset in aData, ab dem die Daten liegen
    udword mDataMax;   // max. Byte-Offset-1 fuer Daten in aData,
                      //
    udword aData[0];   // Beginn des Datenkopierbereichs
}CUCB_ASC_BUFFER;
```

Parameter:

Die Funktion hat die folgenden Parameter:

| Parameter | Beschreibung |
|--------------------|--|
| <code>comId</code> | Feldbus-Schnittstelle (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE |
| <code>pBuf</code> | Definiert die angeforderte Datenmenge und den Ort, an den sie kopiert werden soll, bevor dann <code>CUCB_AscReady()</code> aufgerufen wird. Sind bereits ausreichend Daten im FIFO vorhanden, wird während <code>CUL_AscRcv()</code> <code>CUCB_AscRcvReady()</code> aufgerufen. |

Tabelle 15: Parameter `CUL_AscRcv`

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. |

Tabelle 16: Rückgabewert CUL_AscRcv

Restriktionen:

- Falls der durch CUCB_ASC_BUFFER definierte Speicherbereich nicht im Datensegment der CUT liegt, werden die CUIT und die CUT terminiert.
- Es können maximal 1024 Byte Daten angefordert werden.
- Es kann minimal 1 Byte Daten angefordert werden.

3.6.6.4 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 |
|-----------|--|
| comId | Feldbus-Schnittstelle (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE |

Tabelle 17: Parameter `CUCB_AscRcvReady`

Restriktionen:

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

3.6.6.5 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 ≤ 1 kByte sein.

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

Im Fehlerfall wird die definierte Datenmenge:

- Nicht gesendet
- 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> | Feldbus-Schnittstelle (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE |
| <code>pBuf</code> | Definiert die zu sendende Datenmenge. |

Tabelle 18: Parameter `CUL_AscSend`

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. |

Tabelle 19: Rückgabewert `CUL_AscSend`

Restriktionen:

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

3.6.6.6 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 | Feldbus-Schnittstelle (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE |

Tabelle 20: Parameter CUCB_AscSendReady

3.6.7 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.

3.6.7.1 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. |

Tabelle 21: Parameter CUL_SocketOpenUDPBind

Rückgabewert:

Es wird ein Error code (dword) 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_SOCKET_ERROR | Andere Socket Fehler. |

Tabelle 22: Rückgabewert CUL_SocketOpenUDPBind

Restriktionen:

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

3.6.7.2 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 (dword) 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 |

Tabelle 23: Rückgabewert `CUL_SocketOpenUDP`

3.6.7.3 CUL_NetMessageAlloc

Die Funktion `CULMessageAlloc()` alloziert Messagespeicher für die Nutzung der folgenden Funktionen:

- `CUL_SocketSendTo()` bei UDP
- `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 | |
| | HIMatrix F*01/02, HIMax bis V.4 | HIMatrix F*03, HIMax ab V.4 |
| | ≥ 1 Byte und ≤ 1400 Byte | ≥ 1 Byte und ≤ 1472 Byte |
| proto | 0 = TCP 1 = UDP | |

Tabelle 24: Parameter CUL_NetMessageAlloc

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.

3.6.7.4 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 $\neq 0$. |

Tabelle 25: Parameter CUL_SocketSendTo

Rückgabewert:

Es wird ein Error code (dword) 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. |

Tabelle 26: Rückgabewert CUL_SocketSendTo

Restriktionen:

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

3.6.7.5 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. |

Tabelle 27: Parameter CUCB_SocketUDPRcv

3.6.7.6 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 <code>CUL_NetMessageAlloc()</code> reservierter Speicher. |

Tabelle 28: Parameter CUL_NetMessageFree

Restriktionen:

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

3.6.7.7 CUL_SocketOpenTcpServer_TCP

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, uword 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. Ist der Wert Null, wird der Defaultwert 10 verwendet. Die maximale Obergrenze für diesen Parameter liegt bei 50. Größere Werte werden auf 50 begrenzt. |

Tabelle 29: Parameter CUL_SocketOpenTcpServer_TCP

Rückgabewert:

Es wird ein Error code (dword) zurückgegeben.

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

| Error code | Beschreibung |
|---------------------|---|
| socketNummer | Vergebene SocketNummer für TCP 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. |

Tabelle 30: Rückgabewert CUL_SocketOpenTcpServer_TCP

Restriktionen:

Im erfolgreichen Fall wird 1 Socket verbraucht.

3.6.7.8 CUCB_SocketTryAccept

Die COM ruft die Funktion `CUCB_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 <i>CUL_SocketOpenTcpServer()</i> erzeugter Socket. |

Tabelle 31: Parameter CUCB_SocketTryAccept

3.6.7.9 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. |

Tabelle 32: Parameter CUL_SocketAccept

Rückgabewert:

Es wird ein Error code (DWORD) zurückgegeben.

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

| Error code | Beschreibung |
|---------------------|---|
| Socket | Falls > 0, neu erzeugter Socket. Falls < 0 Errorcode. |
| CUL_WRONG SOCK | Falscher Socket-Typ oder Socket nicht vorhanden. |
| CUL_NO_MORE_SOCKETS | Es sind keine Socket-Ressourcen mehr verfügbar. |
| CUL_SOCKET_ERROR | Andere Socket Fehler. |

Tabelle 33: Rückgabewert CUL_SocketAccept

Restriktionen:

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

3.6.7.10 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. |

Tabelle 34: Parameter `CUL_SocketOpenTcpClient`

Rückgabewert:

Es wird ein Error code (dword) 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. |

Tabelle 35: Rückgabewert `CUL_SocketOpenTcpClient`

3.6.7.11 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 |
|--------------|---|
| socket | Zuvor mit <code>CUL_SocketOpenTcpClient()</code> erzeugter und beauftragter Socket. |
| successfully | TRUE, falls der Verbindungsversuch erfolgreich war, ansonsten FALSE. |

Tabelle 36: Parameter CUCB_SocketConnected

3.6.7.12 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. |

Tabelle 37: Parameter CUL_SocketSend

Rückgabewert:

Es wird ein Error code (dword) zurückgegeben.

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

| Error code | Beschreibung |
|-----------------|--|
| CUL_OKAY | Message erfolgreich versendet. |
| CUL_WRONG SOCK | Falscher Socket-Typ oder Socket nicht vorhanden. |
| CUL_WOULD_BLOCK | Message kann nicht versendet werden, da sonst Socket blockieren würde. |
| CUL_SOCK_ERROR | Andere Socket Fehler. |

Tabelle 38: Rückgabewert CUL_SocketSend

Restriktionen:

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

3.6.7.13 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.

i

TCP ist ein Stream Socket. Im Gegensatz zu UDP (Datagram Socket) kann beim Empfang von Daten mittels `CUCB_SocketTcpRcv()` nicht davon ausgegangen werden, dass immer genau eine ganze Sendung pro Aufruf ankommt.

Es können mehr, aber auch weniger Daten sein. Die Datenlänge wird als `dataLength` mitgeteilt.

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. |

Tabelle 39: Parameter `CUCB_SocketTcpRcv`

3.6.7.14 CUL_SocketClose

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

Der Socket wird innerhalb von 90 Sekunden geschlossen. Die Funktion `SocketOpen` kann erst wieder ausgeführt werden, wenn der Socket geschlossen wurde.

Funktionsprototyp:

`dword CUL_SocketClose(dword socket)`

Parameter:

Die Funktion hat den folgenden Parameter:

| Parameter | Beschreibung |
|-----------|-------------------------|
| socket | Zuvor erzeugter Socket. |

Tabelle 40: Parameter CUL_SocketClose

Rückgabewert:

Es wird ein Error code (dword) 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. |

Tabelle 41: Rückgabewert CUL_SocketClose

3.6.8 Timer-IF

3.6.8.1 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)

3.6.8.2 CUL_GetDateAndTime

Die Funktion `CUL_GetDateAndTime()` liefert an die übergebene Speicherstelle `*pSec` die Sekunden seit 1. Januar 1970, 00:00 und in `*pMsec` die zugehörigen Millisekunden. Die Werte werden mit der sicheren CPU abgeglichen und können je nach Parametrierung über SNTP extern synchronisiert werden, siehe Kommunikationshandbuch HI 801 100 D.

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.

3.6.9 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. |

Tabelle 42: Parameter Diagnose

3.7 Funktionen für HiMatrix F*03 und HM31

Die folgenden Funktionen gelten nur für die Steuerung HiMatrix F*03 und HM31.

3.7.1 COM-User-IRQ-Task

Die COM-User-IRQ-Task (CUIT) muss sich den Code- und Datenspeicher mit der CUT teilen. Diese hat einen eigenen Stack und wird wie bei der CUT-Stack durch das COM-BS (aus Sicht der CUIT dynamisch) festgelegt und ist 32 kByte 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.

i

Die Größe des CUIT-Stack darf 32 kByte nicht überschreiten!

Wird der CUIT-Stack überschritten, werden die Daten in den Speicher der COM geschrieben und können zu Fehlfunktionen führen!

Restriktionen:

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

3.7.1.1 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.

i

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.

3.7.1.2 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 |

Tabelle 43: Parameter CUL_IrqServiceEnable

Restriktionen:

Werden für *devNo* Werte ungleich 1 oder 2 verwendet, oder bei Verwendung einer nicht mit CAN belegten Feldbusschnittstelle, so werden CUIT/CUT terminiert.

3.7.1.3 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 |

Tabelle 44: Parameter CUL_IrqServiceDisable

Restriktionen:

Werden für *devNo* Werte ungleich 1 oder 2 verwendet, oder bei Verwendung einer nicht mit CAN belegten Feldbusschnittstelle, so werden CUIT/CUT terminiert.

3.7.1.4 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 |

Tabelle 45: Parameter CUL_DeviceBaseAddr

Restriktionen:

Werden für *devNo* Werte ungleich 1 oder 2 verwendet, oder bei Verwendung einer nicht mit CAN belegten Feldbuschnittstelle, so werden CUIT/CUT terminiert.

3.7.2 NVRAM-IF

Die CUT und die CUIT können den verfügbaren Bereich schreiben und lesen.

Die Größe des verfügbare NVRAM ist abhängig von der verwendeten Steuerungen.

| Element | HIMax ab V.4 | HIMatrix F*03 | HIMatrix F*01/02 |
|------------|-------------------------------------|--------------------------------------|------------------|
| NVRAM Size | 24576 Bytes (64 kByte -40 kByte) | 483328 Bytes (512 kByte-40 kByte) | Nicht Verfügbar |

Tabelle 46: Speicherbereich für Code und Daten

i

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.

Eine Unterscheidung für Speicherbereiche die häufig oder selten beschrieben werden erfolgt nicht.

3.7.2.1 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, siehe Kapitel 3.7.2 |
| source | Speicherbereich in CUT Datensegment, der ins NVRAM kopiert werden soll. |
| size | Anzahl Bytes, die kopiert werden sollen. |

Tabelle 47: Parameter CUL_NVRamWrite

Restriktionen:

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

- $\text{offset} \geq \text{Größe des verfügbare NVRAM}$.
- $\text{offset} + \text{size} > \text{Größe des verfügbare NVRAM}$.
- `source` nicht im CUT Datensegment.
- $\text{source} + \text{size}$ nicht im CUT Datensegment.

3.7.2.2 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, siehe Kapitel 3.7.2 |
| destination | Speicherbereich in CUT Datensegment, in den die Daten aus dem NVRAM kopiert werden soll. |
| size | Anzahl Bytes, die kopiert werden sollen. |

Tabelle 48: Parameter CUL_NVRamRead

Restriktionen:

Bei ungültigen Parametern werden die CUT und die CUIT terminiert:

- `offset` \geq Größe des verfügbare NVRAM.
- `offset+size` $>$ Größe des verfügbare NVRAM.
- `destination` nicht im CUT Datensegment.
- `destination+size` nicht im CUT Datensegment.

3.7.3 Semaphore-IF

Die CUT und die CUIT haben zur Prozesssynchronisation zusammen **eine** Semaphore.

Die Daten der CUCB- und CUL- Funktionen, die gemeinsam mit der Funktion `CUCB_IrqService()` verwendet werden, müssen über eine Semaphore geschützt werden. Damit wird die Konsistenz der gemeinsam mit der Funktion `CUCB_IrqService()` verwendeten Daten sichergestellt.

3.7.3.1 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 |
|-----------|---|
| pContext | 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. |

Tabelle 49: Parameter Semaphore-IF

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.

3.7.3.2 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 |
|-----------|---|
| pContext | Mit dem gleichen Wert für <code>*pContext</code> , wie er durch <code>CUL_SemaRequest</code> oder <code>CUL_SemaTry</code> geschrieben wurde. |

Tabelle 50: Parameter CUL_SemaRelease

Restriktionen:

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

3.7.3.3 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. |

Tabelle 51: Parameter CUL_SemaTry

Rückgabewert:

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

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

| Rückgabewert | Beschreibung |
|--------------------|---------------------------------------|
| <code>TRUE</code> | Semaphore konnte belegt werden. |
| <code>FALSE</code> | Semaphore konnte nicht belegt werden. |

Tabelle 52: Rückgabewert CUL_SemaTry

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.

i

Die Funktionen `CUL_SemaTry()` und `CUL_SemaRequest()` dürfen auch dann ohne Blockade aufgerufen werden, wenn die aufrufende Task die Semaphore schon belegt hat; nur müssen dann auch gleich viele `CUL_SemaRelease` 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.

3.8 Installation der Entwicklungsumgebung

In diesem Kapitel wird die Installation der Entwicklungsumgebung und die Erstellung einer ComUserTask beschrieben.

Die neueste Entwicklungsumgebung befindet sich jeweils auf der aktuellen HIMA DVD.

3.8.1 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 7/ Windows 10 installiert werden.

i

Voraussetzungen zur Installation, siehe Kapitel 3.1. Den **Virens scanner** auf dem PC deaktivieren, auf dem Cygwin installiert werden soll, um Probleme bei der Installation von Cygwin zu verhindern.

Die folgenden Schritte ausführen, um die Cygwin-Umgebung zu installieren:

Setupprogramm zur Installation von Cygwin starten:

1. Das Cygwin-Installationsarchiv `cygwin-1.7.5-1` von der Installations-CD auf Ihre lokale Festplatte (z. B. Laufwerk C:\) kopieren.
2. Im Windows-Explorer das Cygwin-Verzeichnis `C:\cygwin-1.7.5-1` öffnen.
3. Die Installation von Cygwin mit einem Doppelklick auf die Datei **setup-2.697.exe** starten.
4. Im Cygwin-Dialogfenster auf die Schaltfläche **Weiter** klicken, um das Setup auszuführen.

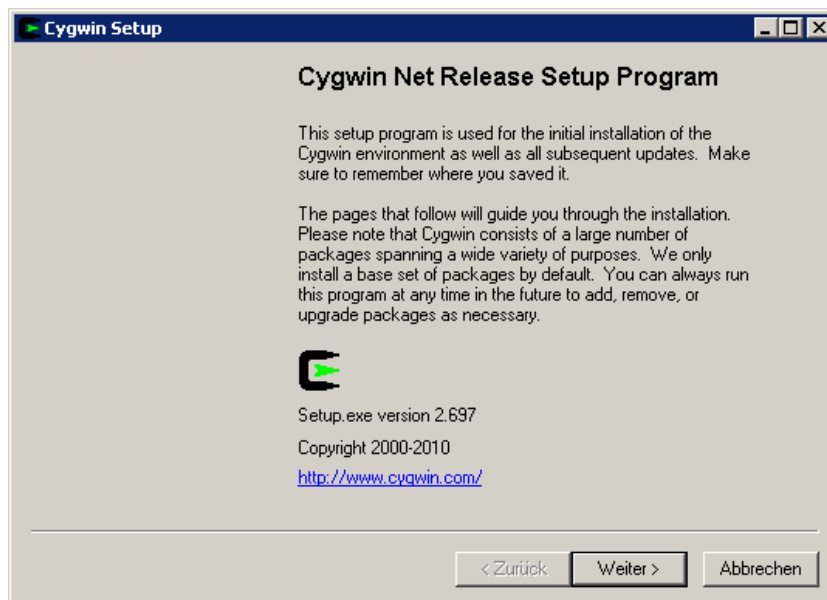


Bild 2: Cygwin Setup-Dialog *Cygwin Setup*

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

Diesen Schritt ausführen, um den Virusscanner für die Installation von Cygwin zu deaktivieren.

i

Den Virusscanner vor der Cygwin-Installation deaktivieren, da es abhängig vom verwendeten Virusscanner vorkommen kann, dass dieses Fenster nicht erscheint, obwohl der Virusscanner läuft.

1. **Disable Virus scanner** wählen, um Probleme während der Installation durch den Virusscanner zu verhindern.
2. Auf die Schaltfläche **Weiter** klicken, um die Eingabe zu bestätigen.

Im Dialog *Choose Installation Type* die Installations-Quelle von Cygwin auswählen:

1. Als Installations-Quelle **Install from Local Directory** auswählen.
2. Auf die Schaltfläche **Weiter** klicken, um die Eingabe zu bestätigen.

Im Dialog *Choose Installation Directory* das Installations-Ziel von Cygwin auswählen:

1. Das Verzeichnis angeben, in welches Cygwin installiert werden soll.
2. Alle weiteren Voreinstellungen des Dialogs übernehmen.
3. Auf die Schaltfläche **Weiter** klicken, um die Eingabe zu bestätigen.

Im Dialog *Select Local Package Directory* das Cygwin-Installationsarchiv wählen:

1. Im Feld *Local Package Directory* das Cygwin-Installationsarchiv angeben, in dem sich die Installationsdateien befinden.
2. Auf die Schaltfläche **Weiter** klicken, um die Eingabe zu bestätigen.

Im Dialog mit *Select Packages* alle Packages zur Installation auswählen:

1. Den radio button **Curr** auswählen.
2. Im Anzeigefeld mehrmals langsam auf die Installationsoption neben **All** klicken, bis **Install** für eine vollständige Installation aller packages angezeigt wird (ca. 1,86 GB Speicherbedarf).

i

Darauf achten, 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!

3. Auf die Schaltfläche **Weiter** klicken, um die Eingabe zu bestätigen.

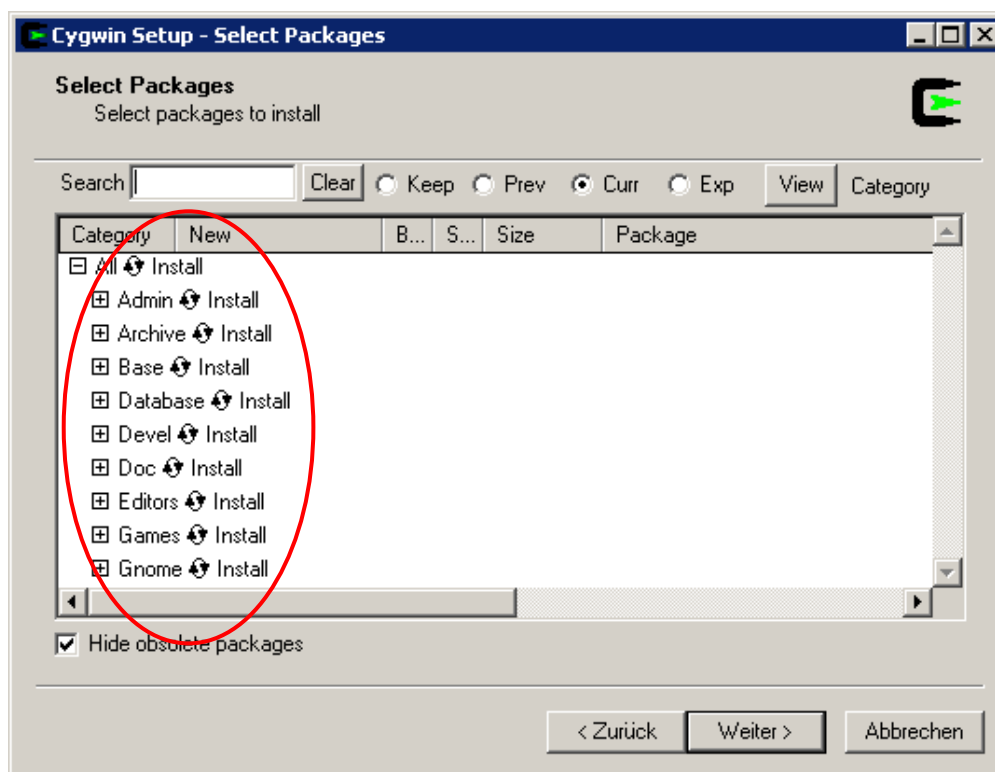


Bild 3: Cygwin Setup-Dialog *Select Packages*

Die Cygwin-Installation mit den folgenden Einträgen abschließen:

1. Eintrag in das **Startmenü** wählen.
2. Eintrag **Desktop-Icon** wählen.
3. Auf die Schaltfläche **Fertigstellen** klicken, 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 53: Befehle in Cygwin (Bash Shell)

3.8.2 Installation des GNU Compilers

Die folgenden Schritte ausführen, um den GNU Compiler zu installieren:

1. Im Windows-Explorer das Verzeichnis der Installations CD öffnen.
2. Auf das zip-File `gcc-ppc-v3.3.2_binutils-v2.15.zip` doppelklicken.
3. Alle Dateien in das Cygwin-Verzeichnis (z. B. `C:\cygwin\...`) extrahieren.
Der GNU Compiler wird im Cygwin-Verzeichnis in den Ordner **gcc-ppc** entpackt.
4. In der Systemsteuerung die Umgebungsvariablen eintragen:

- Die Systemeigenschaften über das Windows-Startmenü **Einstellungen->Systemsteuerung->System** öffnen.
- Register **Erweitert** wählen.
- Auf die Schaltfläche **Umgebungsvariablen** klicken.
- Im Feld *Systemvariablen* die Systemvariable **Path** wählen. Die Systemvariable mit dem Eintrag: `C:\cygwin\gcc-ppc\bin` erweitern.

Von der Installations-CD den Ordner `cut_src` in das Home-Verzeichnis kopieren. Der Ordner `cut_src` enthält alle für die Erstellung eines ComUserTask benötigten "include"- und "lib"-Verzeichnisse.

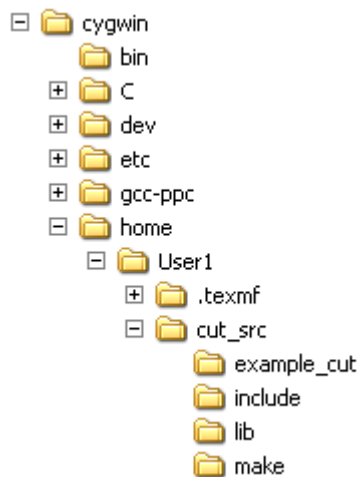


Bild 4: Strukturbaum von cygwin

i

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

Wenn ein anderes Home-Verzeichnis für die Cygwin Bash Shell angelegt werden soll, dann muss die Batchdatei `c:\cygwin\cygwin.bat` mit dem Befehl `set Home` ergänzt werden.

```
@echo off
```

```
C:
```

```
chdir C:\cygwin\bin
```

```
set Home=C:\User1
```

```
bash --login -i
```

Bild 5: Batchdatei *Cygwin.bat*

i

Um ausführbaren Code für das auf der HIMA DVD mitgelieferte Programm *example_cut* zu generieren, siehe Kapitel 3.9.2.4.

3.9 Neues CUT-Projekt anlegen

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

i

Das CUT-Projekt **example_cut** befindet sich fertig angepasst auf der HIMA DVD.

Um ausführbaren Code für das mitgelieferte Programm *example_cut* zu generieren, siehe Kapitel 3.9.2.4.

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 ldb-Datei im Verzeichnis *make* heißt dann **example_cut.ldb**.

Für eine neue ComUserTask den Ordner *example_cut* erstellen.

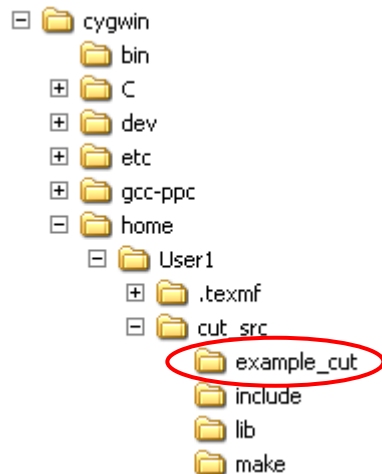


Bild 6: Strukturbaum von cygwin

1. Folgende Dateien in das Verzeichnis *example_cut* kopieren

- Example_cut.c
- Example_cut.mke
- makefile

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

3.9.1 CUT-Makefiles

Konfiguration der CUT-Makefiles für unterschiedliche Sourcefiles und ldb-Dateien

Insgesamt müssen drei Makefiles, wie in den folgenden Absätzen beschrieben, angepasst werden.

3.9.1.1 Makefile mit der Erweiterung „.mke“

Die mke-Datei befindet sich im jeweiligen Quellcode-Verzeichnis
z. B. *cut_src\example_cut\example_cut.mke*.

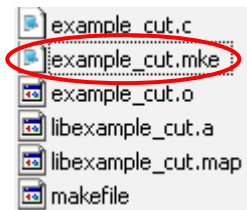


Bild 7: mke-Datei im Ordner example_cut

In der mke-Datei die folgenden Änderungen durchführen:

1. Der Variablen **module** muss der gleiche Loadable-Namen zugewiesen werden wie der .mke Datei (z. B. example_cut).
2. 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: example_cut.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=
```

Bild 8: mke-Datei

Makefile

Die makefile-Datei befindet sich im jeweiligen Quellcode-Verzeichnis.
z. B. cut_src\example_cut\makefile

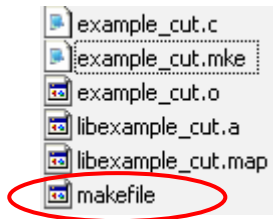


Bild 9: makefile im Ordner *example_cut*

In der makefile-Datei die folgenden Änderungen durchführen:

1. Die Include-Zeile für die mke-Datei nach oben ziehen und die .mke Datei auf den aktuellen Namen ändern.
2. Den Make-Aufruf mit den beiden Variablen **SUBMOD_DIRS** und **CUT_NAME** erweitern.

```
#
#
# $Id: makefile 82791 2007-05-07 09:52:54Z es_gb $
#
# $Log$
# Revision 1.1.8.2 2005/10/11 12:35:46 es_fp
# Initial checking - files moved from HEAD.
#
# Revision 1.1 2005/02/02 13:48:50 es_lx
# init rev
#
#
module=cut_src

INCLUDE_DIRS= ./ cut_src
SUBMOD_DIRS = cut_src/cutapp

SUBMOD1_DIRS=$(foreach dir,$(SUBMOD_DIRS),../$(dir))

SUBMOD_LIBS=$(foreach dir,$(SUBMOD_DIRS),$(dir)/lib$(notdir $(dir)).${LIBEXT})

cut_src: $(foreach dir,$(SUBMOD_DIRS),../$(dir).LibToBuild)
    @echo did make for $(SUBMOD1_DIRS); echo

%.LibToBuild:
    @$(MAKE) -C $(@:.LibToBuild=) -f `basename $@ .LibToBuild`.mke all

# end of file
```

Bild 10: makefile (exemplarisch)

3.9.1.2 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 befindet sich im cut_src-Verzeichnis
z. B. *cut_src\makeinc.inc.app*.

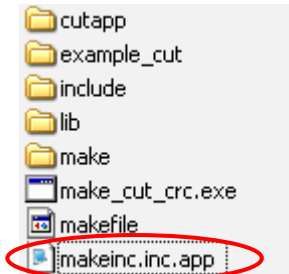


Bild 11: makeinc.inc.app Datei im Ordner *example_cut*

In der makeinc.inc.app-Datei die folgenden Änderungen durchführen:

1. Die Datei mit der Variablen **CUT_NAME** erweitern.

```
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
```

Bild 12: makeinc.inc.app

3.9.2 C-Quellcode bearbeiten

Folgende Schritte ausführen, um die Quellcodedatei zu öffnen:

1. Das Projektverzeichnis *cut_src\example_cut* öffnen, welches in den vorangegangenen Schritten erstellt und konfiguriert wurde.
2. Mit einem Editor (z. B. Notepad) die C-Quellcodedatei mit der Erweiterung **.c** öffnen.

3.9.2.1 Eingangs- und Ausgangsvariablen konfigurieren

Die folgenden Schritte ausführen, um die Ein- und Ausgangsvariablen in der Quellcodedatei zu konfigurieren:

1. Die Datengröße der Variablen, die im Register **Daten Ausgänge** in SILworX angelegt werden sollen, muss in der Quellcodedatei im Array **CUT_PDI[X]** angelegt werden.
2. Die Datengröße der Variablen, die im Register **Daten Eingänge** in SILworX angelegt werden sollen, muss in der Quellcodedatei im Array **CUT_PDO[X]** angelegt werden.

3.9.2.2 Startfunktion im CUT

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

3.9.2.3 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 SILworX Anwenderprogramm zurück.

i

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

```
/* Example for the CUT implemetation */
#include "include/cut_types.h"
#include "include/cut.h"
#ifdef __cplusplus
extern "C" {
#endif
/*****
/* SILworX Output Records (CPU->COM) */
uword CUT_PDI[1] __attribute__((section("CUT_PD_IN_SECT"), aligned(1)));
/* SILworX Input Records (COM->CPU) */
uword CUT_PDO[1] __attribute__((section("CUT_PD_OUT_SECT"), aligned(1)));
*****/

/* Callback function for starting the CUT */
void CUCB_TaskLoop(udword mode)
{
    if (CUT_PDI[0] > CUT_PDO[0]) /*This is executed only, if the          */
    {                               /*SILworX application program      */
                                    /*was processed.                  */
                                    /*The SILworX application program*/
                                    /*adds the value 1 to CUT_PDO[0] and */
                                    /*writes the result into CUT_PDI[0]      */

        CUT_PDO[0] = CUT_PDI[0]; /*Copies the value from input CUT_PDI[0] */
                                /*into output CUT_PDO[0] of the 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 */

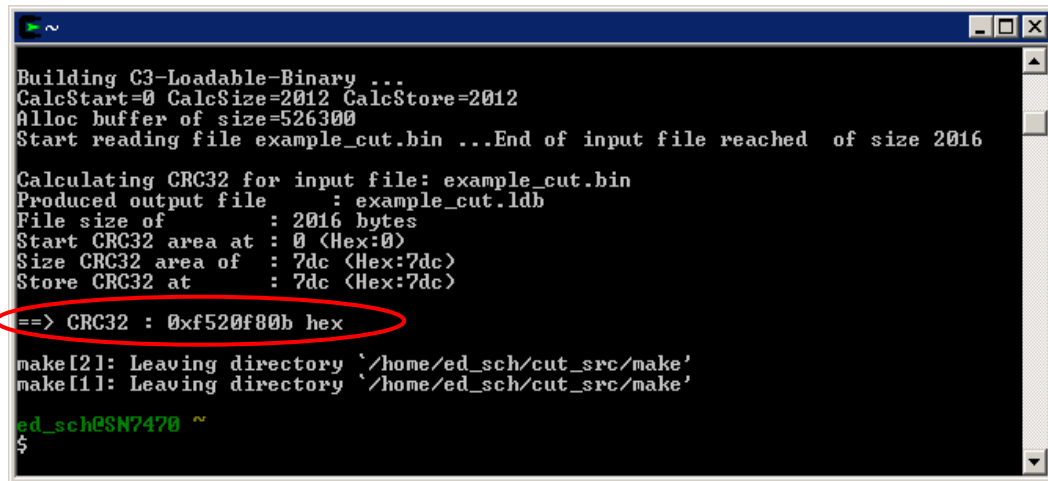
```

Bild 13: C-Code example_cut.c

3.9.2.4 Erstellung des Ausführbaren Code (ldb-Datei)

Die folgenden Schritte zur Erstellung des ausführbaren Codes (ldb-Datei) ausführen:

1. Die **Cygwin Bash Shell** starten.
2. In das Verzeichnis `.../cut_src/example_cut/` wechseln.
3. Die Codegenerierung mit der folgenden Eingabe starten
`make cut_HIMax` für HIMax
`make cut_I2` für HIMatrix F*01/02
`make cut_I3` für HIMatrix F*03.
Das Binärfile **cut.ldb** wird im Verzeichnis `/cut_src/make/` automatisch erzeugt.
4. Wenn der CRC32 erzeugt wurde, dann wurde auch ausführbarer Code erzeugt (siehe rote Markierung in Bild 14).



```
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 ~
$
```

Bild 14: Cygwin Bash Shell

Dieser ausführbare Code (ldb-Datei) muss in das Projekt als ComUserTask geladen werden (siehe Kapitel 3.9.3).

3.9.3 ComUserTask in das Projekt einbinden

In SILworX die folgenden Schritte ausführen, um den ComUserTask in das Projekt einzubinden:

3.9.3.1 ComUserTask anlegen

Eine neue ComUserTask anlegen:

1. Im Strukturbaum **Konfiguration, Ressource, Protokolle** selektieren.
2. Im Kontextmenü von Protokolle **Neu**, ComUserTask wählen, um eine neue ComUserTask hinzuzufügen.
3. Im Kontextmenü der ComUserTask **Eigenschaften** das **COM-Modul** auswählen. Standardeinstellungen können für die erste Konfiguration beibehalten werden.



Es kann immer nur ein ComUserTask pro Ressource angelegt werden.

3.9.3.2 Programmcode in das Projekt laden

Eine neue ComUserTask in das Projekt laden:

1. Im Strukturbaum **Konfiguration, Ressource, Protokolle** öffnen.
2. Rechtsklick auf ComUserTask und im Kontextmenü **User Task laden** wählen. Das Verzeichnis `../cut_src/make/` öffnen.
3. Die **ldb-Datei** auswählen, welche im ComUserTask ausgeführt werden soll.



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.

3.9.3.3 Variablen 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 abhängig der Steuerung (siehe Kapitel 3.2) Daten ausgetauscht werden.

Die folgenden beiden Globalen Variablen erstellen:

| Variable | Typ |
|----------|------|
| COM_CPU | UINT |
| CPU_COM | UINT |

3.9.3.4 Prozessvariablen verbinden

Prozessvariablen im ComUserTask:

1. Rechtsklick auf ComUserTask und im Kontextmenü **Edit** wählen.
2. Im Dialog **Edit** das Register **Prozessvariablen** wählen.

Ausgangsvariablen (CPU->COM)

In das Register **Ausgangsvariablen** werden die Variablen eingetragen, die von der CPU zur COM übertragen werden sollen.

| Name | Typ | Offset | Globale Variable |
|---------|------|--------|------------------|
| CPU_COM | UINT | 0 | CPU_COM |

Tabelle 54: Ausgangsvariablen (CPU->COM)

1. Aus der Objektauswahl die Globalen Variablen zum Versenden per Drag&Drop in das Register **Ausgangsvariablen** ziehen.
2. Rechtsklick auf eine leere Stelle im Bereich **Ausgangsvariablen** um Kontextmenü zu öffnen.
3. Im Kontextmenü **Neue Offsets** wählen, um die Offsets der Variablen neu zu generieren.

Eingangsvariablen (COM->CPU)

In das Register **Eingangsvariablen** werden die Variablen eingetragen, die von der COM zur CPU übertragen werden sollen.

| Name | Typ | Offset | Globale Variable |
|---------|------|--------|------------------|
| COM_CPU | UINT | 0 | COM_CPU |

Tabelle 55: Eingangsvariablen (COM->CPU)

1. Aus der Objektauswahl die Globalen Variablen zum Empfangen per Drag&Drop in das Register **Eingangsvariablen** ziehen.
2. Rechtsklick auf eine leere Stelle im Bereich **Eingangsvariablen** um Kontextmenü zu öffnen.
3. Im Kontextmenü **Neue Offsets** wählen, um die Offsets der Variablen neu zu generieren.

ComUserTask Konfiguration verifizieren:

1. Im Strukturbaum **Konfiguration, Ressource, Protokolle**, ComUserTask wählen.
2. Rechtsklick auf **Verifikation**, um die CUT Konfiguration zu verifizieren.
3. Einträge im **Logbuch** sorgfältig überprüfen, gegebenenfalls korrigieren.

3.9.3.5 Erstellung des SILworX Anwenderprogramms

Das SILworX Anwenderprogramm erstellen

1. Im Strukturbaum **Konfiguration, Ressource** und im Kontextmenü **Edit** wählen.
2. Aus der Objektauswahl die Globalen Variablen **COM_CPU** und **CPU_COM** per Drag & Drop aus der Objektauswahl in das Zeichenfeld ziehen.
3. Das Anwenderprogramm wie in der folgenden Abbildung dargestellt erstellen.

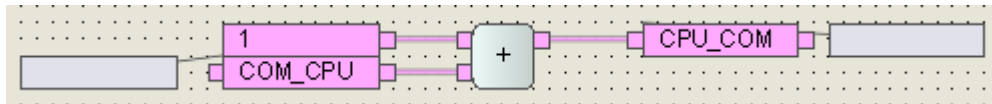


Bild 15: SILworX FBS-Editor

Das Schedule-Intervall [ms] konfigurieren

1. Rechtsklick auf ComUserTask und im Kontextmenü **Eigenschaften** wählen.
2. Im Eingabefeld *Schedule-Intervall [ms]* eintragen, in welchen Intervallen die ComUserTask aufgerufen werden soll.

i

Die Konfiguration des ComUserTask muss mit dem Anwenderprogramm der Ressource neu kompiliert und in die Steuerung übertragen werden, bevor dieser in der HIMA Steuerung wirksam wird.

Die ComUserTask mit dem Online-Test testen

1. Im Strukturbaum **Konfiguration, Ressource, Programm** selektieren.
2. Rechtsklick auf **Programm** und im Kontextmenü **Online** wählen; System-Login durchführen.

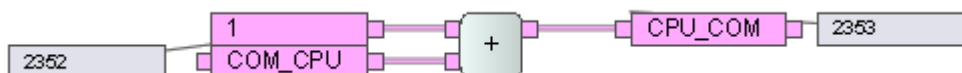


Bild 16: SILworX Online-Test

Funktion des SILworX Anwenderprogramms:

Das SILworX 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 (Schedule-Intervall [ms]) wird das Signal **CPU_COM** an die CUT-Funktion (siehe Beispielcode Kapitel 3.9.2) übergeben.

Die ComUserTask empfängt das Signal **CPU_COM** und sendet den Wert mit dem Signal **COM_CPU** unverändert zurück.

3.9.4 Fehler beim Laden einer Konfiguration mit CUT

Laufzeitprobleme (z. B. ComUserTask 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 des HIMax/HIQuad X Kommunikationsmoduls oder der HIMatrix Steuerung:

- In der Online-Ansicht des Hardware-Editors über die Funktion **Wartung/Service, Modul (Neustart)** einen Reset des Kommunikationsmoduls ausführen (Reset der HIMatrix über Reset-Taster, siehe Datenblatt der jeweiligen Steuerung).
- Eine neue CUT erzeugen (möglichst ohne Laufzeitfehler, Endlosschleife).
- CUT (ldb-Datei) in das Projekt laden.
- Code generieren.
- Code in die Steuerung laden.

4.3 D-Sub-Buchsen FB1 und FB2

Bei Einsatz des SSI-Submoduls gilt die Pin-Belegung der D-Sub-Buchsen FB1 und FB2, siehe Kommunikationshandbuch HI 801 100 D.

4.4 Konfiguration zwischen COM und SSI-Submodul

Der Datenaustausch zwischen dem Kommunikationsmodul (COM) **2** und dem SSI-Submodul **4** erfolgt über die interne serielle Schnittstelle **3** und muss durch die ComUserTask realisiert werden, siehe Kapitel ComUserTask.

Das in der ComUserTask erstellte Datenprotokoll für den Datenaustausch zwischen der COM und dem SSI-Submodul muss wie folgt eingestellt sein:

- Baudrate 115,2 kBit/s
- Datenlänge 8 Bit,
- Parity even,
- 1 Stoppbit

4.5 Konfiguration der SSI-Schnittstelle

Die SSI-Schnittstelle **5** wird durch das Startkommando-Byte SSISTART konfiguriert, siehe Tabelle 57.

Zur Anforderung eines neuen Geber-Datensatzes (aktuelle Position) muss das Startkommando-Byte SSISTART jedesmal erneut im Anwenderprogramm gesetzt und an das SSI-Submodul **4** weitergeleitet werden.

Das zu sendende Startkommando-Byte SSISTART hat folgendes Format:

| Bit | Beschreibung |
|----------------|---|
| 7 | Auxiliary Bit Dieses Bit legt die Belegung des Startkommando-Bytes fest. |
| Wenn Bit 7 = 1 | |
| 6 ... 4 | Folgende Einstellung für den SSI-Schiebetakt ist möglich 000 62,5 kHz 001 125 kHz 010 250 kHz 011 500 kHz |
| 3 | Schaltet Pin 3 und 8 als Dateneingang oder Taktausgang. 0: D3+, D3- als Eingang geschaltet 1: CL2+, CL2- als Taktausgang geschaltet |
| 2 | Taktausgang CL1 0: aktiviert 1: deaktiviert |
| 1 | Nicht verwendet |
| 0 | Taktausgang CL2 0: aktiviert 1: deaktiviert |
| Wenn Bit 7 = 0 | |
| 6 ... 0 | Nicht verwendet |

Tabelle 57: Datenformat des Startkommando-Bytes SSISTART

Die über die SSI-Schnittstelle ermittelten Geber-Daten werden in folgendem Format und Reihenfolge über die interne serielle Schnittstelle **3** vom SSI-Submodul **4** zur COM **2** gegeben.

Im Anwenderprogramm können diese Geber-Daten zur Ermittlung der X-Y-Z Position ausgewertet werden (Anmerkung: die Datenbits werden in der gleichen Reihenfolge zum Anwenderprogramm durchgereicht, wie diese vom Geber geliefert werden).

| Nr. | Kanal | Datenbit |
|-----|--------|-------------|
| 1 | Kanal1 | D47 ... D40 |
| 2 | | D39 ... D32 |
| 3 | | D31 ... D24 |
| 4 | | D23 ... D16 |
| 5 | | D15 ... D8 |
| 6 | | D7 ... D0 |
| 7 | Kanal2 | D47 ... D40 |
| 8 | | D39 ... D32 |
| 9 | | D31 ... D24 |
| 10 | | D23 ... D16 |
| 11 | | D15 ... D8 |
| 12 | | D7 ... D0 |
| 13 | Kanal3 | D47 ... D40 |
| 14 | | D39 ... D32 |
| 15 | | D31 ... D24 |
| 16 | | D23 ... D16 |
| 17 | | D15 ... D8 |
| 18 | | D7 ... D0 |

Tabelle 58: Format und Reihenfolge der Geber-Daten

4.5.1 Leitungslänge und empfohlene Taktraten

Die folgende Tabelle zeigt empfohlene Taktraten für die SSI-Schnittstelle in Abhängigkeit von der Feldleitungslänge.

| Leitungslänge /m | Taktrate /kHz |
|------------------|---------------|
| < 25 | ≤ 500 |
| < 50 | < 400 |
| < 100 | < 300 |
| < 200 | < 200 |
| < 400 | < 100 |

Tabelle 59: Empfohlene Taktraten in Abhängigkeit von Feldleitungslängen

4.6 Applikationshinweise

Das SSI-Submodul ist von der HIMA Steuerung galvanisch getrennt, daher müssen die SSI-Geber im Feld mit einer externen Spannungsversorgung versorgt werden.

Folgende Applikationen sind mit einem SSI-Submodul möglich:

- Anschluss von 3 Gebern zur gleichzeitigen Ermittlung von X-Y-Z-Koordinaten
1 SSI-Schiebetakt (CL1+, CL1-) und 3 Datenkanäle (D1+, D1-, D2+, D2-, D3+, D3-) zur gleichzeitigen Ermittlung von XYZ-Koordinaten.
Alle 3 Geber werden vom gleichem Takt (CL1+, CL1-) und somit auch mit der gleichen Taktfrequenz versorgt.
- Anschluss von 2 Gebern zur gleichzeitigen Ermittlung von X-Y-Koordinaten
2 SSI-Schiebetakte (CL1+, CL1-, CL2+, CL2-) und 2 Datenkanäle (D1+, D1-, D2+, D2-) zur gleichzeitigen Ermittlung von XY-Koordinaten.
Beide Geber werden aus zwei getrennten Takten (CL1+, CL1-, CL2+, CL2-) angesteuert.
Beide haben die gleiche Taktfrequenz.
- Anschluss von 1 Geber
1 SSI-Schiebetakt (CL1+, CL1- oder CL2+, CL2-) und 1 Datenkanal (D1+, D1- oder D2+, D2-).

i

Der Einbau des SSI-Submoduls in die Zone 2 (EG-Richtlinie 94/9/EG, ATEX) ist bei Beachtung der besonderen Bedingungen X zulässig.

5 Allgemein

In diesem Kapitel sind Parameter gesammelt, die für alle Kommunikationsprotokolle relevant sind.

5.1 Maximale Kommunikationszeitscheibe

Die maximale Kommunikationszeitscheibe ist die zugeteilte Zeit in Millisekunden (ms) pro CPU-Zyklus, innerhalb der das Prozessormodul die Kommunikationsaufgaben abarbeitet. Wenn die Protokollverarbeitung innerhalb der Dauer einer Kommunikationszeitscheibe nicht beendet werden konnte, führt die CPU dennoch die sicherheitsrelevanten Überwachungen für alle Protokolle in einem CPU-Zyklus aus.

i

Wenn nicht alle in einem CPU-Zyklus anstehenden Kommunikationsaufgaben ausgeführt werden können, erfolgt die komplette Übertragung der Kommunikationsdaten über mehrere CPU-Zyklen. Die Anzahl der Kommunikationszeitscheiben ist dann größer 1.

Für die Berechnungen der zulässigen maximalen Reaktionszeiten gilt die Bedingung, dass die Anzahl der Kommunikationszeitscheiben genau 1 ist.

5.1.1 Ermitteln der maximalen Dauer der Kommunikationszeitscheibe

Für eine erste Abschätzung der maximalen Dauer der Kommunikationszeitscheibe müssen die folgenden Zeiten aufsummiert und das Ergebnis in den Systemparameter *Max. Kom.-Zeitscheibe [ms]* in den Eigenschaften der Ressource eingetragen werden:

- Pro COM-Modul 3 ms.
- Pro redundante safe**ethernet** Verbindung 1 ms.
- Pro nicht redundante safe**ethernet** Verbindung 0,5 ms.
- Pro KByte Nutzdaten bei nichtsicheren Protokollen (z. B. Modbus) 1 ms.

HIMA empfiehlt, den abgeschätzten Wert *Max. Kom.-Zeitscheibe [ms]* mit dem im Control Panel angezeigten Wert zu vergleichen und gegebenenfalls in den Eigenschaften der Ressource zu korrigieren. Dies kann z. B. in einem FAT (Factory Acceptance Test) oder SAT (Site Acceptance Test) durchgeführt werden.

Ermitteln der tatsächlichen Dauer der maximalen Kommunikationszeitscheibe

1. Das HIMA System unter voller Last betreiben (FAT, SAT):
Alle Kommunikationsprotokolle sind in Betrieb (safe**ethernet** und Standardprotokolle).
2. Das **Control Panel** öffnen und im Strukturbaum das Verzeichnis **Kom.-Zeitscheibe** wählen.
3. Anzeige *Maximale Kom.-Zeitscheibe Dauer pro Zyklus [ms]* auszulesen.
4. Anzeige *Maximale Anzahl benötigter Kom.-Zeitscheibe Zyklen* auszulesen.

Die Dauer der Kommunikationszeitscheibe ist so hoch einzustellen, dass der CPU-Zyklus die vom Prozess vorgegebene Watchdog-Zeit nicht überschreiten kann, wenn er die eingestellte Kommunikationszeitscheibe ausnutzt.

5.2 Lastbegrenzung

Für jedes Kommunikationsprotokoll kann ein Rechenzeitbudget in % (*μP-Budget*) vorgegeben werden. So kann die verfügbare Rechenzeit zwischen den konfigurierten Protokollen verteilt werden. Die Summe der Rechenzeitbudgets aller parametrisierten Kommunikationsprotokolle eines CPU- oder COM-Moduls darf nicht größer als 100 % sein.

Die festgelegten Rechenzeitbudgets der einzelnen Kommunikationsprotokolle werden überwacht. Hat ein Kommunikationsprotokoll sein Rechenzeitbudget erreicht oder überschritten und es steht keine zusätzliche Rechenzeit als Reserve zur Verfügung, so wird das Kommunikationsprotokoll nicht komplett abgearbeitet.

Wenn noch genügend zusätzliche Rechenzeit vorhanden ist, wird diese verwendet, um ein Kommunikationsprotokoll, das sein Rechenzeitbudget erreicht oder überschritten hat noch abzuarbeiten. Dadurch kann es vorkommen, dass ein Kommunikationsprotokoll tatsächlich ein höheres Rechenzeitbudget verwendet als ihm zugeteilt wurde.

Eventuell werden über 100 % Rechenzeitbudget online angezeigt. Dies ist kein Fehler, das Rechenzeitbudget über 100 % ist die zusätzlich verwendete Rechenzeit.

i

Das zusätzliche Rechenzeitbudget ist keinesfalls eine Zusicherung für ein bestimmtes Kommunikationsprotokoll und kann jederzeit vom System zurückgenommen werden.

Anhang

Glossar

| Begriff | Beschreibung |
|-------------------|--|
| ARP | Address Resolution Protocol: Netzwerkprotokoll zur Zuordnung von Netzwerkadressen zu Hardwareadressen. |
| Bit-Variable | Variable, die bitweise adressiert wird. |
| CENELEC | Comité Européen de Normalisation Électrotechnique (Europäisches Komitee für elektrotechnische Normung) |
| Connector Board | Anschlusskarte für HIMax Modul. |
| COM | Kommunikationsmodul |
| CPU | Prozessormodul |
| CRC | Cyclic Redundancy Check, Prüfsumme |
| Dataview | Einer Dataview sind die Globalen Variablen für Eingangs- und Ausgangsdaten für den Zugriff durch Modbus-Quellen zugeordnet. |
| EN | Europäische Normen |
| Exportbereich | Als Exportbereich wird die Prozessdatenmenge bezeichnet, die vom System (aus einem Anwenderprogramm, HW-Eingang oder einem anderen Protokoll) geschrieben und vom Modbus Master gelesen werden kann. |
| FB | Feldbus |
| FBS | Funktionsbausteinsprache |
| ICMP | Internet Control Message Protocol: Netzwerkprotokoll für Status- und Fehlermeldungen. |
| IEC | Internationale Normen für die Elektrotechnik. |
| Importbereich | Als Importbereich wird die Prozessdatenmenge bezeichnet, die vom Modbus-Master geschrieben wird und als Eingangsdaten für das System (in einem Anwenderprogramm, HW-Ausgang oder einem anderen Protokoll) verwendet werden kann. |
| KE | Kommunikationsendpunkt |
| MAC-Adresse | Hardware-Adresse eines Netzwerkanschlusses (Media Access Control). |
| NSIP | Nicht-sicherheitsbezogenes Protokoll. |
| PADT | Programming and Debugging Tool (nach IEC 61131-3), PC mit SILworX. |
| PE | Schutzerde |
| PELV | Protective Extra Low Voltage: Funktionskleinspannung mit sicherer Trennung. |
| PES | Programmierbares Elektronisches System |
| R | Read |
| Rack-ID | Identifikation eines Basisträgers (Nummer). |
| rückwirkungsfrei | Es seien zwei Eingangsschaltungen an dieselbe Quelle (z. B. Transmitter) angeschlossen. Dann wird eine Eingangsschaltung „rückwirkungsfrei“ genannt, wenn sie die Signale der anderen Eingangsschaltung nicht verfälscht. |
| R/W | Read/Write |
| Register-Variable | Variable, die wortweise adressiert wird. |
| SB | Systembusmodul |
| SFF | Safe Failure Fraction, Anteil der sicher beherrschbaren Fehler. |
| SIF | Sicherheitstechnische Funktion |
| SIL | Safety Integrity Level (nach IEC 61508) |
| SILworX | Programmiersoftware für HIMax, HIQuad X und HIMatrix. |
| SIP | Sicherheitsbezogenes Protokoll |
| SNTP | Simple Network Time Protocol (RFC 1769) |
| SRS | System.Rack.Slot |
| SW | Software |

| Begriff | Beschreibung |
|---------|---------------|
| TMO | Timeout |
| W | Write |
| WD | Watchdog |
| WDZ | Watchdog-Zeit |

Abbildungsverzeichnis

| | | |
|----------|--|----|
| Bild 1: | Prozessdaten-Austausch zwischen CPU und COM (CUT) | 15 |
| Bild 2: | Cygwin Setup-Dialog <i>Cygwin Setup</i> | 52 |
| Bild 3: | Cygwin Setup-Dialog <i>Select Packages</i> | 54 |
| Bild 4: | Strukturbaum von cygwin | 55 |
| Bild 6: | Strukturbaum von cygwin | 56 |
| Bild 7: | mke-Datei im Ordner <i>example_cut</i> | 57 |
| Bild 8: | mke-Datei | 57 |
| Bild 9: | makefile im Ordner <i>example_cut</i> | 58 |
| Bild 10: | makefile (exemplarisch) | 58 |
| Bild 11: | makeinc.inc.app Datei im Ordner <i>example_cut</i> | 59 |
| Bild 12: | makeinc.inc.app | 60 |
| Bild 13: | C-Code <i>example_cut.c</i> | 62 |
| Bild 14: | Cygwin Bash Shell | 63 |
| Bild 15: | SILworX Programm-Editor | 66 |
| Bild 16: | SILworX Online-Test | 66 |
| Bild 17: | Blockschaltbild | 68 |

Tabellenverzeichnis

| | | |
|-------------|---|----|
| Tabelle 1: | Zusätzlich geltende Handbücher | 5 |
| Tabelle 2: | Systemanforderung und Ausstattung ComUserTask | 12 |
| Tabelle 3: | Eigenschaften ComUserTask | 12 |
| Tabelle 4: | Abkürzungen | 13 |
| Tabelle 5: | Schedule-Intervall [ms] | 14 |
| Tabelle 6: | Allgemeine Eigenschaften der CUT | 16 |
| Tabelle 7: | Systemvariablen der ComUserTask | 17 |
| Tabelle 8: | Speicherbereich für Code und Daten | 20 |
| Tabelle 9: | Stack-Speicher | 20 |
| Tabelle 10: | Parameter CUCB_TaskLoop | 20 |
| Tabelle 11: | Parameter CUL_AscOpen | 22 |
| Tabelle 12: | Rückgabewert CUL_AscOpen | 22 |
| Tabelle 13: | Parameter CUL_AscClose | 23 |
| Tabelle 14: | Rückgabewert CUL_AscClose | 23 |

| | |
|--|----|
| Tabelle 15: Parameter CUL_AscRcv | 24 |
| Tabelle 16: Rückgabewert CUL_AscRcv | 25 |
| Tabelle 17: Parameter CUCB_AscRcvReady | 26 |
| Tabelle 18: Parameter CUL_AscSend | 27 |
| Tabelle 19: Rückgabewert CUL_AscSend | 27 |
| Tabelle 20: Parameter CUCB_AscSendReady | 28 |
| Tabelle 21: Parameter CUL_SocketOpenUDPBind | 29 |
| Tabelle 22: Rückgabewert CUL_SocketOpenUDPBind | 29 |
| Tabelle 23: Rückgabewert CUL_SocketOpenUDP | 30 |
| Tabelle 24: Parameter CUL_NetMessageAlloc | 31 |
| Tabelle 25: Parameter CUL_SocketSendTo | 32 |
| Tabelle 26: Rückgabewert CUL_SocketSendTo | 32 |
| Tabelle 27: Parameter CUCB_SocketUDPRcv | 33 |
| Tabelle 28: Parameter CUL_NetMessageFree | 34 |
| Tabelle 29: Parameter CUL_SocketOpenTcpServer_TCP | 35 |
| Tabelle 30: Rückgabewert CUL_SocketOpenTcpServer_TCP | 35 |
| Tabelle 31: Parameter CUCB_SocketTryAccept | 36 |
| Tabelle 32: Parameter CUL_SocketAccept | 37 |
| Tabelle 33: Rückgabewert CUL_SocketAccept | 37 |
| Tabelle 34: Parameter CUL_SocketOpenTcpClient | 38 |
| Tabelle 35: Rückgabewert CUL_SocketOpenTcpClient | 38 |
| Tabelle 36: Parameter CUCB_SocketConnected | 39 |
| Tabelle 37: Parameter CUL_SocketSend | 40 |
| Tabelle 38: Rückgabewert CUL_SocketSend | 40 |
| Tabelle 39: Parameter CUCB_SocketTcpRcv | 41 |
| Tabelle 40: Parameter CUL_SocketClose | 42 |
| Tabelle 41: Rückgabewert CUL_SocketClose | 42 |
| Tabelle 42: Parameter Diagnose | 44 |
| Tabelle 43: Parameter CUL_IrqServiceEnable | 46 |
| Tabelle 44: Parameter CUL_IrqServiceDisable | 46 |
| Tabelle 45: Parameter CUL_DeviceBaseAddr | 47 |
| Tabelle 46: Speicherbereich für Code und Daten | 48 |
| Tabelle 47: Parameter CUL_NVRamWrite | 48 |
| Tabelle 48: Parameter CUL_NVRamRead | 49 |
| Tabelle 49: Parameter Semaphore-IF | 49 |
| Tabelle 50: Parameter CUL_SemaRelease | 50 |
| Tabelle 51: Parameter CUL_SemaTry | 51 |
| Tabelle 52: Rückgabewert CUL_SemaTry | 51 |
| Tabelle 53: Befehle in Cygwin (Bash Shell) | 54 |
| Tabelle 54: Ausgangsvariablen (CPU->COM) | 65 |

| | |
|--|-----------|
| Tabelle 55: Eingangsvariablen (COM->CPU) | 65 |
| Tabelle 56: Systemanforderung und Ausstattung ComUserTask | 68 |
| Tabelle 57: Datenformat des Startkommando-Bytes SSISTART | 69 |
| Tabelle 58: Format und Reihenfolge der Geber-Daten | 70 |
| Tabelle 59: Empfohlene Taktraten in Abhängigkeit von Feldleitungslängen | 70 |

Für weitere Informationen kontaktieren Sie:

HIMA Paul Hildebrandt GmbH

Albert-Bassermann-Str. 28
68782 Brühl, Germany

Telefon +49 6202 709-0
Fax +49 6202 709-107
E-Mail info@hima.com

Erfahren Sie online mehr über HIMA Lösungen:



www.hima.com/de/