Manual

Protocols

ComUserTask

Contact

HIMA Paul Hildebrandt GmbH

P.O. Box 1261

68777 Brühl

Phone: +49 6202 709-0

Fax: +49 6202 709-107

E-mail: info@hima.com

Document designation	Description
HI 801 517 D, Rev. 11.00 (1930)	German original document
HI 801 521 E, Rev. 11.00.00 (1933)	English translation of the German original document

Table of Contents

1 Introduction

The ComUserTask manual describes the properties of the ComUserTask and its configuration in SILworX for the safety-related HIMA controller systems.

Knowledge of regulations and the proper technical implementation of the instructions detailed in this manual performed by qualified personnel are prerequisites for planning, engineering, programming, installing, starting up, operating and maintaining the HIMA controllers.

HIMA shall not be held liable for severe personal injuries, damage to property or the environment caused by any of the following: unqualified personnel working on or with the devices, de-activation or bypassing of safety functions, or failure to comply with the instructions detailed in this manual (resulting in faults or impaired safety functionality).

HIMA automation devices have been developed, manufactured and tested in compliance with the pertinent safety standards and regulations. They may only be used for the intended applications under the specified environmental requirements.

1.1 Structure and Use of This Manual

The manual contains the following chapters:

- Introduction
- Safety
- Product description
- Configuring the ComUserTask in SILworX

Additionally, the following documents must be taken into account:

Name	Content	Document no.
HIMax system manual	Hardware description HIMax system	HI 801 001 E
HIMax safety manual	Safety function HIMax systems	HI 801 003 E
HIMatrix safety manual	Safety function HIMatrix systems	HI 800 023 E
HIMatrix compact system manual	Hardware description HIMatrix compact system	HI 800 141 E
HIMatrix modular system manual	Hardware description HIMatrix modular F 60 system	HI 800 191 E
HIQuad X system manual	Hardware description HIQuad X system	HI 803 211 E
HIQuad X safety manual	Safety function HIQuad X system	HI 803 209 E
Automation security manual	Description of automation security aspects related to the HIMA systems	HI 801 373 E
SILworX first steps manual	Introduction to SILworX.	HI 801 103 E

Table 1: Additional Applicable Manuals

All the current manuals can be obtained upon request by sending an e-mail to: documentation@hima.com. The documentation is available for registered HIMA customers in the download area https://www.hima.com/en/downloads/.

1.2 Target Audience

This document is aimed at the planners, design engineers, programmers and the persons authorized to start up, operate and maintain the automation systems. Specialized knowledge of safety-related automation systems is required.

1.3 Writing Conventions

To ensure improved readability and comprehensibility, the following writing conventions are used in this document:

Bold	To highlight important parts. Names of buttons, menu functions and tabs that can be clicked and used in the programming tool.
Italics	Parameters and system variables, references.
`Courier`	Literal user inputs.
RUN	Operating states are designated by capitals.
Chapter 1.2.3	Cross-references are hyperlinks even if they are not specially marked. In the electronic document (PDF): When the mouse pointer hovers over a hyperlink, it changes its shape. Click the hyperlink to jump to the corresponding position.

Safety notices and operating tips are specially marked.

1.3.1 Safety Notices

Safety notices must be strictly observed to ensure the lowest possible risk.

The safety notices are represented as described below.

- Signal word: warning, caution, notice.
- Type and source of risk.
- Consequences arising from non-observance.
- Risk prevention.

The signal words have the following meanings:

- Warning indicates hazardous situations which, if not avoided, could result in death or serious injury.
- Caution indicates hazardous situation which, if not avoided, could result in minor or moderate injury.
- Notice indicates a hazardous situation which, if not avoided, could result in property damage.

⚠ SIGNAL WORD

Type and source of risk!

Consequences arising from non-observance.

Risk prevention.

NOTICE

Type and source of damage!

Damage prevention.

1.3.2 Operating Tips

Additional information is structured as presented in the following example:

i The text giving additional information is located here.

Useful tips and tricks appear as follows:

TIP The tip text is located here.

1.4 Safety Lifecycle Services

HIMA provides support throughout all the phases of the plant's safety lifecycle, from planning and engineering through commissioning to maintenance of safety and security.

HIMA's technical support experts are available for providing information and answering questions about our products, functional safety and automation security.

To achieve the qualification required by the safety standards, HIMA offers product or customer-specific seminars at HIMA's training center or on site at the customer's premises. The current seminar program for functional safety, automation security and HIMA products can be found on HIMA's website.

Safety Lifecycle Services:	
Onsite+ / On-Site Engineering	In close cooperation with the customer, HIMA performs changes or extensions on site.
Startup+ / Preventive Maintenance	HIMA is responsible for planning and executing preventive maintenance measures. Maintenance actions are carried out in accordance with the manufacturer's specifications and are documented for the customer.
Lifecycle+ / Lifecycle Management	As part of its lifecycle management processes, HIMA analyzes the current status of all installed systems and develops specific recommendations for maintenance, upgrading and migration.
Hotline+ / 24 h Hotline	HIMA's safety engineers are available by telephone around the clock to help solve problems.
Standby+ / 24 h Call-Out Service	Faults that cannot be resolved over the phone are processed by HIMA's specialists within the time frame specified in the contract.
Logistics+/ 24 h Spare Parts Service	HIMA maintains an inventory of necessary spare parts and guarantees quick, long-term availability.

Contact details:	
Safety Lifecycle Services	https://www.hima.com/en/about-hima/contacts-worldwide/
Technical Support	https://www.hima.com/en/products-services/support/
Seminar Program	https://www.hima.com/en/products-services/seminars//

2 Safety

All safety information, notes and instructions specified in this document must be strictly observed. The product may only be used if all guidelines and safety instructions are adhered to.

The product is operated with SELV or PELV. No imminent risk results from the product itself. Use in the Ex zone is only permitted if additional measures are taken.

2.1 Intended Use

To use the HIMA controllers, all pertinent requirements must be met, see relevant manuals in Table 1.

2.2 Residual Risk

No imminent risk results from a HIMA system itself.

Residual risk may result from:

- Faults related to engineering.
- Faults in the user program.
- Faults related to the wiring.

2.3 Safety Precautions

Observe all local safety requirements and use the protective equipment required on site.

2.4 Emergency Information

A HIMA system is a part of the safety equipment of an overall system. If the controller fails, the system enters the safe state.

In case of emergency, no action that may prevent the HIMA system from operating safely is permitted.

2.5 Automation Security for HIMA Systems

Industrial controllers must be protected against IT-specific problem sources. Those problem sources are:

- Attackers inside and outside of the customer's plant.
- Operating failures.
- Software failures.

All requirements for protection against manipulation specified in the safety and application standards must be met. The operator is responsible for authorizing personnel and implementing the required protective actions.

⚠ WARNING

Physical injury possible due to unauthorized manipulation of the controller!

Protect the controller against unauthorized access!

For example:

- **Changing the default settings for login and password!**
- **Controlling the physical access to the controller and PADT!**

Careful planning should identify the measures to implement. The required measures are to be implemented after the risk analysis is completed. Such measures can include:

- Meaningful allocation of user groups.
- Maintained network maps help to ensure that secure networks are permanently separated from public networks and, if required, only a well-defined connection exists (e.g., via a firewall or a DMZ).
- Use of appropriate passwords.

A periodical review of the security measures is recommended, e.g., every year.

The user is responsible for implementing the necessary measures in a way suitable for the plant!

For further details, refer to the HIMA automation security manual (HI 801 373 E).

3 ComUserTask

In addition to the logic program created in SILworX, a C program of the user can also be implemented with connection to various communication interfaces of the COM module.

The non-safety-related C program runs as ComUserTask on the communication module and is interference-free from the safe processor module of the controller.

The ComUserTask has a specific cycle time that does not depend on the CPU cycle. This allows the users to program in C any kind of applications and implement them as ComUserTask, for example:

- Communication interfaces for special protocols (SSI, TCP, UDP, etc.).
- Gateway function between TCP/UDP and serial communication.

3.1 System Requirements

Equipment and system requirements

Element	Description
Controller	HIMax with COM module HIQuad X with COM module HIMatrix
CPU module	The Ethernet interfaces on the processor module may not be used for ComUserTask.
COM module	Ethernet 10/100BaseT D-sub connectors FB1 and FB2, e.g., for RS232. If the serial fieldbus interfaces (FB1 or FB2) or FB3 in HIMatrix (RS485) are used, they must be equipped with an optional HIMA submodule. Refer to the communication manual (HI 801 101 E) for details.
Activation	A software activation code is required for activation, refer to the communication manual (HI 801 001 E) for details.

Table 2: Equipment and System Requirements for ComUserTask

3.2 Properties of ComUserTask

Element	Description
ComUserTask	One ComUserTask can be configured for each HIMax X-COM 01, HIQuad X F-COM 01 or HIMatrix COM. Refer to the communication manual for details (HI 801 101 E).
Safety-related	No
Data exchange	Configurable (fieldbus and/or Ethernet interface).
Code and data area	See Chapter 3.6.4.
Stack	The stack is located in a reserved memory outside the code or data area, see Chapter 3.6.4.

Table 3: Properties of ComUserTask

3.2.1 Creating a ComUserTask

To create a new ComUserTask

1. In the structure tree, select **Configuration**, **Resource**, **Protocols**.
2. Select **New**, ComUserTask from the context menu of protocols to add a new ComUserTask.
3. Select **Properties, COM Module** from the context menu of ComUserTask.
   The default settings may be retained for the first configuration.

3.3 The ComUserTask Development Environment

In addition to the normal C commands, a specific library with defined functions (see Chapter 3.5) is available for programming a ComUserTask.

Development environment

Chapter 3.8 describes how to install the development environment and create a ComUserTask.

The development environment includes the GNU C compiler and Cygwin, which are subject to the conditions of the GNU General Public License (see www.gnu.org).

The latest development environment is included in the current HIMA DVD.

Controller

In the HIMA controllers, the ComUserTask has no access to the safe hardware inputs and outputs. If an access to the safe hardware inputs and outputs is required, a CPU user program must exist for connecting the variables, see Chapter 3.5.5.

3.4 Abbreviations

Abbreviation	Description
CUCB	COM User Callback (CUCB_ Functions invoked by the COM)
CUIT	COM user IRQ task
CUL	COM User Library (CUL_ Functions invoked in the CUT)
CUT	ComUserTask
CUT_PDI	CUT process data input area on the COM that contains the data written by the CPU.
CUT_PDO	CUT process data output area on the COM that contains the data to be read by the CPU.
GNU	GNU project
IF	Interface
FB	Fieldbus interface of the controller
FIFO	First in first out (Data memory)
NVRAM	Non-volatile random access memory
SSI	Synchronous serial interface

Table 4: Abbreviations

3.5 CUT Interface in SILworX

The process data communication of the ComUserTask runs between COM and CPU.

⚠ WARNING

The CUT code operates in the COM in an interference-free manner with the safe CPU. This ensures that the safe CPU is protected against the CUT code.
Note that errors in the CUT code can disturb the entire COM function, thus affecting or stopping the controller's function.
The CPU safety functions, however, are not compromised.

3.5.1 Schedule Interval [ms]

The ComUserTask is invoked in a predefined schedule interval [ms] during the controller's states RUN and STOP_VALID_CONFIG (COM module).

In SILworX, *Schedule Interval [ms]* can be set in the *Properties* of the ComUserTask.

Schedule Interval [ms]	
Range of values:	2…255 ms
Default value:	15 ms

Table 5: Schedule Interval [ms]

i The COM processor time available to the CUT depends on the other configured COM functions such as safe**ethernet** or Modbus TCP.
If CUT is not finished within the schedule interval, each call to restart the CUT is ignored until CUT has been processed.

3.5.2 Scheduling Preprocessing

In the controller's state RUN:

Before each CUT call, the COM module provides the process data of the safe CPU to the CUT in a memory area defined by CUT.

In the controller's state STOP:

No process data is exchanged between COM and safe CPU.

3.5.3 Scheduling Post-processing

In the controller's state RUN:

After each CUT call, the COM module provides the process data of the CUT to the safe CPU.

In the controller's state STOP:

No process data is exchanged between COM and safe CPU.

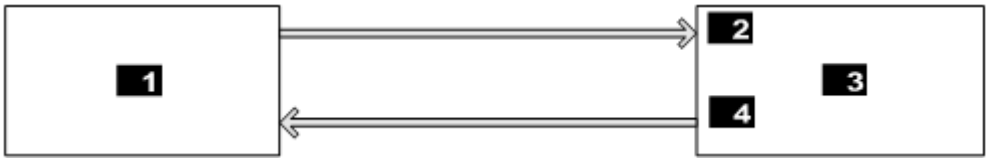
3.5.4 STOP_INVALID_CONFIG

If the COM is in the STOP_INVALID_CONFIG state, CUT is not executed.

If the COM module enters the STOP_INVALID_CONFIG state and executes CUT or CUIT, these functions are terminated.

3.5.5 CUT Interface Variables (CPU<->CUT)

Configuration of non-safety-related process data communication between safe CPU and COM (CUT). In this context, the maximum process data volume may be exchanged per data direction. The maximum process data volume for standard protocols depends on the controller type, see Chapter 3.6.4.



1 CPU (safe user program)
2 CUT process data input area (CUT_PDI)
3 COM (CUT)
4 CUT process data output area (CUT_PDI)

Figure 1: Process Data Exchange between CPU and COM (CUT)

All data types that are used in SILworX can be exchanged.

The data structure must be configured in SILworX.
The size of the data structures CUT_PDI and CUT_PDO (in the compiled CUT C code) must correspond to the size of the data structures configured in SILworX.

i If the data structures CUT_PDI and CUT_PDO are not available in the compiled C code or do not have the same size as the process data configured in SILworX, the configuration is invalid and the COM module enters the STOP_INVALID_CONFIG state.
Process data communication takes place in the RUN state only.

3.5.6 Menu Function: Properties

The **Properties** function of the ComUserTask context menu opens the **Properties** dialog box.

Element	Description
Type	ComUserTask
Name	Any unique name for a ComUserTask.
Refresh Rate [ms]	Refresh rate in milliseconds at which the communication module and processor module exchange protocol data. If the *Process Data Refresh Rate* is zero or lower than the cycle time for the controller, data is exchanged as fast as possible. Range of values: (2^{31}-1) ms Default value: 0
Allow Multiple Fragments per Cycle	Activated: Transmission of all process data of the protocol between communication module and processor module within one processor module's cycle. Deactivated: Transmission of all process data of the protocol between communication module and processor module, distributed over multiple processor module's cycles. Default value: Activated
Module	Selection of the COM module within which the protocol is processed.
Activate Max. µP Budget [1)]	Activated: Use the µP budget limit from the *Max. µP Budget in [%]* field. Deactivated: Do not use the µP budget limit for this protocol. Default value: Activated
Max. µP Budget in [%] [1)]	Maximum µP budget of the module that can be used for processing the protocols. Range of values: 1…100% Default value: 30%
Behavior on CPU/COM Connection Loss	If the connection of the processor module to the communication module is lost, the input variables are either initialized or are still used unchanged in the process module, depending on this parameter. For instance, if the communication module is removed when communication is running. **If a project created with a SILworX prior to V3 should be converted, this value must be set to Retain Last Value to ensure that the CRC does not change.** **For HIMatrix controllers with CPU OS prior to V8, this value must always be set to Retain Last Value.** Apply Initial Data: Input variables are reset to their initial values. Retain Last Value: The input variables retain the last value. Default value: Retain Last Value
Schedule Interval [ms]	The ComUserTask is invoked in a predefined schedule interval [ms] of controller (COM module), see Chapter 3.5.1. Range of values: 2…255 ms Default value: 15 ms
User Task	Loadable file path, if already loaded
1) The function is available for HIQuad X, HIMax with COM operating system as of V6 and HIMatrix with COM operating system as of V15.	

Table 6: General Properties of the ComUserTask

3.5.7 The Edit Menu Function

The **Edit** function on the ComUserTask context menu is used to open the **Edit** dialog box, which includes the tabs *Process Variables* and *System Variables.*

3.5.7.1 System Variables

The **System Variables** tab contains the following system parameters for monitoring and controlling the CUT:The **System Variables** tab contains the following system parameters for monitoring and controlling the CUT:

<table>
<tr><th>Name</th><th>Function</th></tr>
<tr><td>Execution Time
[DWORD]</td><td>Execution time of the ComUserTask in µs</td></tr>
<tr><td>Real Schedule Interval
[DWORD]</td><td>Time between two ComUserTask cycles in ms</td></tr>
<tr><td>User Task State Control
[WORD]</td><td>The following table shows how the user can control the ComUserTask with the User Task State Control system parameter:
<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>DISABLED
0x8000</td><td>The application program locks the CUT (CUT is not started).</td></tr>
<tr><td>TOGGLE_MODE_0
0x0100</td><td>After the CUT was terminated, it can only be started again after TOGGLE_MODE_1 has been set.</td></tr>
<tr><td>TOGGLE_MODE_1
0x0101</td><td>After the CUT was terminated, it can only be started again after TOGGLE_MODE_0 has been set.</td></tr>
<tr><td>Autostart
Default: 0</td><td>After termination a new start of the CUT is automatically allowed if the error is eliminated.</td></tr>
</table></td></tr>
<tr><td>State of the User Task
[BYTE]</td><td>1 = RUNNING (CUT is running)
0 = ERROR (CUT is not running due to an error)</td></tr>
</table>

Table 7: ComUserTask System Variables

TIP If the CUT is terminated and restarted, the COM state `STOP / LOADING DATA FROM FLASH` is displayed from the flash memory even if the ComUserTask is in the RUN state.

3.5.7.2 Process Variables

Input Signals (COM->CPU)

The **Input Signals** tab contains the variables that should be transferred from the COM module (CUT) to the CPU module (CPU input area).

⚠ CAUTION

The ComUserTask is not safety-related!

Non-safe variables of the ComUserTask must not be used for the safety functions of the CPU user program.

Required entry in the C code

The C code of the ComUserTask must have the following CUT_PDO data structure for the COM outputs (CPU input area):

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

The size of the CUT_PDO data structure must correspond to the size of the data inputs configured in SILworX.

Output Signals (CPU->COM)

The **Output Signals** tab contains the signals that should be transferred from the CPU (CPU output area) to the COM module (CUT).

Required entry in the C code

The C code of the ComUserTask must have the following CUT_PDI data structure for the COM inputs (CPU output area):

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

The size of the CUT_PDI data structure must correspond to the size of the data outputs configured in SILworX.

3.6 CUT Functions

3.6.1 COM User Callback Functions

The COM User Callback functions have all the **CUCB** _ prefix and are directly invoked from the COM when events occur.

---

i All COM user callback functions must be defined in the user's C code!
Also the CUCB_IrqService function, which is not supported for HIMax (but only for HIMatrix with CAN module), must be defined in the user C-Code for reasons of compatibility.
Function prototype: void CUCB_IrqService(udword devNo) {}

The COM user callback (CUCB) and the COM user library (CUL) functions share the stack and the same code and data memory. These functions mutually ensure the consistency of the data shared (variables).

3.6.2 COM User Library Functions

All COM User Library functions and variables have the **CUL**_ prefix and are invoked in the CUT.

All the CUL functions are available via the **libcut.a** object file.

3.6.3 Header Files

The two header files **cut.h** and **cut_types.h** contain all function prototypes for CUL/CUCB and the related data types and constants.

For the data types, the following short cuts are defined in the header file **cut_types.h**:

```
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; // with 0=FALSE, otherwise TRUE
#endif
```

3.6.4 Code/Data Area and Stack for CUT

The code/data area is a coherent memory area that begins with the code segment and the initial data segment and continues with the data segments. In the HIMA linker files (**makeinc.inc.app** and **section.dld**), the written segment sequence and the available storage capacity are predefined (this applies to HIMA and HM31).

Element	HIMax as of V4	HIMax prior to V4	HIMatrix, HIQuad X
Start address	0x780000	0x790000	0x800000
Length	512 kBytes	448 kBytes	4 MB

Table 8: Memory Area for Code and Data

The stack is located in a reserved memory area defined when the COM operating system is started.

Element	HIMax	HIMatrix, HIQuad X
End address	Dynamically (from the point of view of CUT)	Dynamically (from the point of view of CUT)
Length	Approx. 64 kBytes	Approx. 500 kBytes

Table 9: Stack Memory

3.6.5 Start Function CUCB_TaskLoop

`CUCB_TaskLoop()` is the starting function associated with the ComUserTask.

The ComUserTask program execution begins when this function is invoked (see Chapter 3.5.1).

Function prototype:

void CUCB_TaskLoop(udword mode)

Parameter:

The function has the following parameter (returns the value):

Parameters	Description
mode	1 = MODE_STOP corresponds to the mode STOP_VALID_CONFIG 2 = MODE_RUN controller's normal operation

Table 10: Parameter CUCB_TaskLoop

3.6.6 RS485 / RS232 IF Serial Interfaces

The used fieldbus interfaces must be equipped with the corresponding fieldbus submodule (hardware). Refer to the HIMA system documentation for details.

Receive telegrams from the perspective of the CUT application

The number of idle characters for identification of telegram boundaries (idle time) is set to 5 characters for the serial receiving via the ComUserTask.

A communication partner which consecutive sends several telegrams to CUT application, must insert at least 5 idle characters between the sent telegrams. By this, the UART driver identifies these telegrams as single telegrams.

The telegrams received by the UART driver are only forwarded to the CUT application, if at least 5 idle characters have been received.

Send telegrams from the perspective of the CUT application

The number of idle characters for identifying the telegram boundaries (idle time) is set to 5 characters for serial transmission via ComUserTask.

The ComUserTask application interface ensures that the serial telegrams are sent by the application consecutively at intervals of at least the idle time. It should be considered that the storage capacity of the CUT for sending telegrams is limited to one telegram.
If the transmission (call of the sending interface) occurs at shorter intervals than the physical transmission requires, the storage capacity of the CUT is exceeded and the CUL_WOULDBLOCK error code is returned, see Chapter 3.6.6.5.

3.6.6.1 CUL_AscOpen

The `CUL_AscOpen()` function initializes the entered serial interface (*comId*) with the given parameters. After invoking the `CUL_AscOpen()` function, the COM immediately begins receiving data via this interface.

The received data is stored in a FIFO software with a size of 1 kByte **for each** initialized serial interface.

The data is stored until it is read out with the CUT function `CUL_AscRcv()`.

Newly received data are discarded if reading the data from the FIFO is slower than receiving new data.

Function prototype:

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

Parameters:

The function has the following parameters:

Parameters	Description
comId	Fieldbus interface (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE
duplex	0 = Full duplex (only permitted for FB4 if RS232) 1 = Half duplex
baudRate	1 = 1200 bit 2 = 2400 bit 3 = 4800 bit 4 = 9600 bit 5 = 19200 bit 6 = 38400 bit (maximum baud rate for HIMax prior to V4) 7 = 57600 bit (maximum baud rate for HIMax as of V4) 8 = 115000 bit (HIMatrix only)
The data bit length is fixed and set to 8 data bits. Start, parity and stop bits must be added to these 8 data bits.	
parity	0 = NONE 1 = EVEN 2 = ODD
stopBits	1 = 1 bit 2 = 2 bits

Table 11: Parameter CUL_AscOpen

Return value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
CUL_OKAY	The interface was initialized successfully.
CUL_ALREADY_IN_USE	The interface is used by other COM functions or is already open.
CUL_INVALID_PARAM	Incorrect parameters or parameter combinations transmitted.
CUL_DEVICE_ERROR	Other errors

Table 12: Return Value CUL_AscOpen

3.6.6.2 CUL_AscClose

The `CUL_AscClose()` function closes the serial interface entered in *comId*.
In doing so, the data that has already been received but not read out with the function `CUL_AscRcv()` is deleted in FIFO.

Function prototype:

Udword CUL_AscClose(udword comId)

Parameter:

The function has the following parameter:

Parameters	Description
comId	Fieldbus interface (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE

Table 13: Parameter CUL_AscRcv

Return value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
CUL_OKAY	The interface was closed successfully .
CUL_NOT_OPENED	The interface was not opened (by the CUT).
CUL_INVALID_PARAM	Incorrect parameters or parameter combinations transmitted.
CUL_DEVICE_ERROR	Other errors

Table 14: Return Value CUL_AscClose

3.6.6.3 CUL_AscRcv

The `CUL_AscRcv()` function instructs the COM module to provide a defined data volume from the FIFO.

As soon as the requested data is available (and the CUL and scheduling allow it), the COM invokes the *CUCB_AscRcvReady()* function.

If not enough data is contained in FIFO, the `CUL_AscRcv()` function returns immediately.

The instruction to receive data is stored until:

- The instruction was completely processed or
- the *CUL_AscClose()* function is invoked or
- redefined due to a new instruction

i Until the instruction is completely processed, the *pBuf content may only be changed using the *CUCB_AscRcvReady()* function.

Function prototype:

udword CUL_AscRcv(udword comId, CUCB_ASC_BUFFER *pBuf)

```
typedef struct CUCB_AscBuffer {

  bool  bAscState;    // for using by CUT/CUCB
  bool  bError;       // for using by CUT/CUCB
  uword reserved1;    // 2 unused bytes
  udword mDataIdx;    // byte offset in aData from which the data are availa
  udword mDataMax;    // max. byte offset 1 for data in aData,
                      //
  udword aData[0];    // Start point of the data copy range
  }CUCB_ASC_BUFFER;
```

Parameter:

The function has the following parameters:

Parameters	Description
comId	Fieldbus interface (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE
pBuf	It defines the requested amount of data and the location to copy the data before CUCB_AscReady() is invoked. If sufficient data has already been received in the FIFO, the CUCB_AscRcvReady() function is invoked during CUL_AscRcv().

Table 15: Parameter CUL_AscRcv

Return value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
CUL_OKAY	If the order was successful, otherwise error code.
CUL_NOT_OPENED	If the interface was not opened by the CUT.
CUL_INVALID_PARAM	Incorrect parameters or parameter combinations transmitted.
CUL_DEVICE_ERROR	Other errors.

Table 16: Return Value CUL_AscRcv

Restrictions:

- If the memory area (defined by CUCB_ASC_BUFFER) is not allocated in the CUT data segment, CUT and CUIT are terminated.
- A maximum of 1024 bytes of data can be requested.
- A maximum of 1 byte of data can be requested.

3.6.6.4 CUCB_AscRcvReady

If the COM module invokes the `CUCB_AscRcvReady()` function, the requested amount of data is ready in FIFO (data from the serial interface defined in the `comId` parameter).

The data has been previously requested with the `CUL_AscRcv()` function.

The `CUCB_AscRcvReady()` function can be invoked before, after or while invoking the `CUL_AscRcv()` function. The task context is always that of CUT.

The `CUCB_AscRcvReady()` function may invoke all CUT library functions.

These actions are also permitted:

- Increasing `mDataMax` or
- Reconfiguring `mDataIdx` and `mDataMax` by the `*.pBuf` data assigned to `comId` (for further reading).

The structure element of CUCB_ASC_BUFFER.`mDataIdx` has the value of CUCB_ASC_BUFFER.`mDataMax`.

Function prototype:

void CUCB_AscRcvReady(udword comId)

Parameter:

The function has the following parameter:

Parameters	Description
comId	Fieldbus interface (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE

Table 17: Parameter CUCB_AscRcvReady

Restrictions:

If the memory area (defined by CUCB_ASC_BUFFER) is not located in the CUT data segment, CUIT and CUT are terminated.

3.6.6.5 CUL_AscSend

The `CUCB_AscSend` sends the amount of data defined by the parameter pBuf via the comId serial interface.

The defined amount of data must be ≥ 1 byte and ≤ 1kByte.

After data has been sent, the `CUCB_AscSendReady()` function is invoked.

If an error occurs:

- The defined data volume is not sent.
- The `CUCB_AscSendReady()` function will not be invoked.

Function prototype:

udword CUL_AscSend(udword comId, CUCB_ASC_BUFFER *pBuf)

Parameter:

The function has the following parameters:

Parameters	Description
comId	Fieldbus interface (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE
pBuf	Defines the amount of data to be sent.

Table 18: Parameter CUL_AscSend

Return value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
CUL_OKAY	If data was sent successfully.
CUL_WOULDBLOCK	If a message previously sent has not been sent yet.
CUL_NOT_OPENED	If the interface was not opened by the CUT.
CUL_INVALID_PARAM	Incorrect parameters or parameter combinations transmitted.
CUL_DEVICE_ERROR	Other errors.

Table 19: Return Value CUL_AscSend

Restrictions:

If the memory area (defined by CUCB_ASC_BUFFER) is not located in the CUT data segment, CUIT and CUT are terminated.

3.6.6.6 CUCB_AscSendReady

If the COM invokes the `CUCB_AscSendReady()` function, data is completely sent with the `CUCB_AscSend()` function via the serial interface.

The task context is always that of CUT. The `CUCB_AscSendReady()` function may invoke all CUT library functions.

Function prototype:

void CUCB_AscSendReady(udword comId)

Parameter:

The function has the following parameter:

Parameters	Description
comId	Fieldbus interface (RS485, RS232) 1 = FB1 2 = FB2 3 = FB3 4 = FB4_SERVICE

Table 20: Parameter CUCB_AscSendReady

3.6.7 UDP/TCP Socket IF

A maximum of 8 sockets can simultaneously be used irrespective of the used protocol.

The physical connection runs over the 10/100BaseT Ethernet interfaces of the controller.

3.6.7.1 CUL_SocketOpenUDPBind

The `CUL_SocketOpenUdpBind()` function creates a socket of UDP type and binds the socket to the selected port.

The binding address is always INADDR_ANY, i.e., all messages for UDP/port addressed to the COM module are received. Sockets are always run in non-blocking mode, i.e., this function does not block.

Function prototype:

dword CUL_SocketOpenUDPBind(uword port, uword *assigned_port_ptr)

Parameter:

The function has the following parameters:

Parameters	Description
port	An available port number, not occupied by the COM ≥ 0. If the port parameter = 0, the socket is bound to the first available port.
assigned_port_ptr	Address to which the bounded port number should be copied, if the port parameter = 0 or NULL if not.

Table 21: Parameter CUL_SocketOpenUDPBind

Return value:

An error code (dword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
Socket number	Socket number assigned to UDP if > 0. Error codes are < 0.
CUL_ALREADY_BOUND	Binding to a port for UDP not possible.
CUL_NO_MORE_SOCKETS	No more resources are available for socket.
CUL_SOCK_ERROR	Other socket error.

Table 22: Return Value CUL_SocketOpenUDPBind

Restrictions:

If the CUT has no assigned_port_ptr, CUT and CUIT are terminated.

3.6.7.2 CUL_SocketOpenUDP

The `CUL_SocketOpenUdp()` function creates a socket of UDP type without binding to a port. Afterwards, messages can only be sent, but not received via the socket.

Function prototype:

dword CUL_SocketOpenUDP (void)

Parameter:

None

Return value:

An error code (dword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
Socket number	Socket number assigned to UDP if > 0. Error codes are < 0.
CUL_NO_MORE_SOCKETS	No more resources are available for socket.
CUL_SOCK_ERROR	Other socket error.

Table 23: Return Value CUL_SocketOpenUDP

3.6.7.3 CUL_NetMessageAlloc

The `CULMessageAlloc()` function allocates message memory for using the following functions:

- `CUL_SocketSendTo()` with UDP
- `CUL_SocketSend()` with TCP

A maximum of 10 messages can be simultaneously used in CUT.

Function prototype:

void *CUL_NetMessageAlloc(udword size, ubyte proto)

Parameter:

The function has the following parameters:

Parameters	Description
Size	Memory requirements in bytes HIMax up to V4: ≥ 1 byte and ≤ 1400 bytes HIMatrix, HIMax as of V4: ≥ 1 byte and ≤ 1472 bytes
proto	0 = TCP 1 = UDP

Table 24: Parameter CUL_NetMessageAlloc

Return value:

Buffer address to which the data to be sent must be copied. Memory ranges must never be written outside the allocated area. There are no ranges for the used transmission protocols (EtherNet/IP/UDP or TCP).

Restrictions:

If no more memory resources are available or if the parameter size is too big or proto > 1, CUT and CUIT are terminated.

3.6.7.4 CUL_SocketSendTo

The `CUL_SocketSendTo()` function sends the message previously allocated and filled with the `CUL_NetMessageAlloc()` function as UDP package to the destIp/destPort target address.

After the message has been sent, the pMsg message memory is released automatically.

For each transmission, the CULMessageAlloc() function must first be used to allocate message memory.

Function prototype:

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

Parameter:

The function has the following parameters:

Parameters	Description
Socket	Socket created with `CUL_SocketOpenUdp()`.
pMsg	UDP user data memory previously reserved with `CUL_NetMessageAlloc()`.
Size	Memory size in bytes, it must be ≤ than the allocated memory.
destIp	Target address != 0, also 0xffffffff is allowed as broadcast.
destPort	Target port != 0.

Table 25: Parameter CUL_SocketSendTo

Return value:

An error code (dword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
CUL_OKAY	Message was sent successfully.
CUL_NO_ROUTE	No routing available to reach destIp.
CUL_WRONG_SOCK	Invalid socket type or socket not available.
CUL_SOCK_ERROR	Other socket error.

Table 26: Return Value CUL_SocketSendTo

Restrictions:

If the CUT has no pMsg message or if the size of pMsg is too large, CUT and CUIT are terminated.

3.6.7.5 CUCB_SocketUDPRcv

The COM invokes the CUCB_SocketUDPRcv() function if data from the socket is available. In callback, data must be copied from `*pMsg` to CUT data, if required. After the `return` function, no access to `*pMsg` is allowed.

Function prototype:

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

Parameter:

The function has the following parameters:

Parameters	Description
Socket	Socket created with `CUL_SocketOpenUdp()`.
`pMsg`	`pMsg` points to the UDP package begin including the Ethernet header. The transmitter of the message can be identified via the Ethernet header.
packetLength	The length of the package is stored in packetLength, included the length of the header.
dataLength	The length of the UDP user data part is stored in dataLength.

Table 27: Parameter CUCB_SocketUDPRcv

3.6.7.6 CUL_NetMessageFree

The `CUL_NetMessageFree()` function releases the message previously allocated with `CUL_NetMessageAlloc()`.

This function is usually not required since invoking the `CUL_SocketSendTo()` function results in an automatic release.

Function prototype:

void CUL_NetMessageFree(void *pMsg)

Parameter:

The function has the following parameter:

Parameters	Description
`pMsg`	TCP user data memory previously reserved with `CUL_NetMessageAlloc()`.

Table 28: Parameter CUL_NetMessageFree

Restrictions:

If the CUT has no `pMsg` message, CUT and CUIT are terminated.

3.6.7.7 CUL_SocketOpenTcpServer_TCP

The `CUL_SocketOpenServer()` function creates a socket of type TCP and binds the socket to the selected port.

The address for binding is always INADDR_ANY. Additionally, the COM module is requested to perform a *listen* on the stream socket. Sockets are always running in non-blocking mode, i.e., this function does not block.

For further information on how to use the socket, refer to `CUCB_SocketTryAccept()` and `CUL_SocketAccept()`.

Function prototype:

dword CUL_SocketOpenTcpServer(uword port, udword backlog)

Parameter:

The function has the following parameters:

Parameters	Description
port	Port number not occupied by the COM > 0.
backlog	Maximum number of waiting connections for socket. If the value is 0, the default value 10 is used. The maximum upper limit for this parameter is 50. Higher values are limited to 50 Larger values are limited to 50.

Table 29: Parameter CUL_SocketOpenTcpServer_TCP

Return value:

An error code (dword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
Socket number	Socket number already assigned to TCP if > 0. Error codes are < 0.
CUL_ALREADY_BOUND	Binding to a port/proto not possible.
CUL_NO_MORE_SOCKETS	No more resources are available for socket.
CUL_SOCK_ERROR	Other socket error.

Table 30: Return Value CUL_SocketOpenTcpServer_TCP

Restrictions:

If successfully one socket is used.

3.6.7.8 CUCB_SocketTryAccept

The COM invokes the `CUCB_SocketTryAccept()` function if a TCP connection request is pending.

This request can be used to create a socket with the `CUL_SocketAccept()` function.

Function prototype:

void CUCB_SocketTryAccept(dword serverSocket)

Parameter:

The function has the following parameter:

Parameters	Description
serverSocket	Socket previously created by *CUL_SocketOpenTcpServer()*.

Table 31: Parameter CUCB_SocketTryAccept

3.6.7.9 CUL_SocketAccept

The `CUL_SocketAccept()` function creates a new socket for the connection request previously signalized with `CUCB_SocketTryAccept()`.

Function prototype:

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

Parameter:

The function has the following parameters:

Parameters	Description
serverSocket	serverSocket signalized immediately before with CUCB_SocketTryAccept()
pIpAddr	Address to which the IP address of the peer should be copied or 0 if not.
pTcpPort	Address to which the TCP port number of the peer should be copied or 0 if not.

Table 32: Parameter CUL_SocketAccept

Return value:

An error code (dword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
Socket	If > 0, then new socket created. If < 0, then error code.
CUL_WRONG_SOCK	Invalid socket type or socket not available.
CUL_NO_MORE_SOCKETS	No more socket resources are available.
CUL_SOCK_ERROR	Other socket error.

Table 33: Return Value CUL_SocketAccept

Restrictions:

If the CUT has no pIpAddr or pTcpPort, CUT and CUIT are terminated.

3.6.7.10 CUL_SocketOpenTcpClient

The `CUL_SocketOpenTcpClient()` function creates a socket of type TCP with free local port and requests a connection to `destIp` and `destPort`. Sockets are always run in non-blocking mode, i.e., this function does not block. As soon as the connection has been established, the `CUCB_SocketConnected()` function is invoked.

Function prototype:

dword CUL_SocketOpenTcpClient(udword destIp, uword destPort)

Parameter:

The function has the following parameters:

Parameters	Description
destIp	IP address of the communication partner.
destPort	Port number of the communication partners.

Table 34: Parameter CUL_SocketOpenTcpClient

Return value:

An error code (dword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
Socket number	If > 0; error codes are < 0.
CUL_NO_MORE_SOCKETS	No more resources are available for socket.
CUL_NO_ROUTE	No routing available to reach destIp.
CUL_SOCK_ERROR	Other socket error.

Table 35: Return Value CUL_SocketOpenTcpClient

3.6.7.11 CUCB_SocketConnected

The `CUCB_SocketConnected()` function is invoked by the COM module if a TCP connection was established with the `CUL_SocketOpenTcpClient()` function.

Function prototype:

void CUCB_SocketConnected(dword socket, bool successfully)

Parameter:

The function has the following parameters:

Parameters	Description
Socket	Socket previously created and requested with `CUL_SocketOpenTcpClient()`.
Successfully	TRUE if the connection attempt was successfully, otherwise FALSE.

Table 36: Parameter CUCB_SocketConnected

3.6.7.12 CUL_SocketSend

The `CUL_SocketSend()` function sends the message allocated and filled with the `CUL_NetMessageAlloc()` function as TCP package.

After the message has been sent, the pMsg message memory is released automatically.

For each transmission, the CULMessageAlloc() function must first be used to allocate message memory.

Function prototype:

```
dword CUL_SocketSend(  dword socket,
                       void *pMsg,
                       udword size)
```

Parameter:

The function has the following parameters:

Parameters	Description
Socket	Socket previously created with `CUL_SocketAccept()/CUL_SocketOpenTcpClient()`.
pMsg	TCP user data memory previously reserved with `CUL_NetMessageAlloc()`.
Size	Memory size in bytes, it must be ≤ than the allocated memory.

Table 37: Parameter CUL_SocketSend

Return value:

An error code (dword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
CUL_OKAY	Message was sent successfully.
CUL_WRONG_SOCK	Invalid socket type or socket not available.
CUL_WOULD_BLOCK	Message cannot be sent, otherwise the socket would be blocked
CUL_SOCK_ERROR	Other socket error.

Table 38: Return Value CUL_SocketSend

Restrictions:

If the CUT has no `pMsg` message or if the size of `pMsg` is too large, CUT and CUIT are terminated.

3.6.7.13 CUCB_SocketTcpRcv

The `CUCB_SocketTcpRcv()` function is invoked by the COM module if user data from the socket is available.

After quitting the `CUCB_SocketTcpRcv()` function, no access to *pMsg is allowed.

If the user data is also required outside the `CUCB_SocketTcpRcv()` function, the user data must be copied from `*pMsg` to an area created for it.

i TCP is a stream socket. When data is received using CUCB_SocketTcpRcv(), it cannot be assumed that exactly one whole transmission is received for each call, as it is the case for UDP (Datagram Socket).
It can be more or less data. The data length is indicated as dataLength.

If the TCP connection is closed asynchronously (after an error or due to a request from the other side), the `CUCB_SocketTcpRcv()` function with dataLength = 0 is selected.
The call signalized to CUT that the socket must be closed to re-synchronized communication.

Function prototype:

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

Parameter:

The function has the following parameters:

Parameters	Description
Socket	Socket used to receive the user data.
pMsg	The `pMsg` parameter points to the user data begin **without** Ethernet/IP/TCP header.
dataLength	Length of the user data in bytes.

Table 39: Parameter CUCB_SocketTcpRcv

3.6.7.14 CUL_SocketClose

The `CUL_SocketClose()` function closes a socket previously created.

The socket is closed within 90 seconds. Only if the socket has been closed, the SocketOpen function can be executed once again.

Function prototype:

dword CUL_SocketClose(dword socket)

Parameter:

The function has the following parameter:

Parameters	Description
Socket	Socket previously created.

Table 40: Parameter CUL_SocketClose

Return value:

An error code (dword) is returned.

The error codes are defined in the cut.h header file.

Error code	Description
CUL_OKAY	Socket closed and one socket resource free again.
CUL_WRONG_SOCK	Socket not available.

Table 41: Return Value CUL_SocketClose

3.6.8 Timer-IF

3.6.8.1 CUL_GetTimeStampMS

The `CUL_GetTimeStampMS()` function provides a millisecond tick. This tick is suitable for implementing an own timer in CUT/CUIT. The counter is derived from the quartz of the COM processor and has therefore the same precision.

Function prototype:

udword CUL_GetTimeStampMS(void)

3.6.8.2 CUL_GetDateAndTime

The `CUL_GetDateAndTime()` function provides the seconds since the 1st January 1970, 00:00 to the `*pSec` memory and the milliseconds to the `*pMsec` memory. The values are aligned with the safe CPU and, depending on the configuration, they can be externally synchronized via SNTP. Refer to the communication manual (HI 801 101 E) for details.

The values for `CUL_GetDateAndTime()` should **not** be used for time measurement, timer or something else because the values can be provided by the synchronization and/or by the user during operation.

Function prototype:

void CUL_GetDateAndTime(udword *pSec, udword *pMsec)

Restrictions:

If the memory of pSec or pMsec are not allocated in the CUT data segment, CUT and CUIT are terminated.

3.6.9 Diagnostics

The `CUL_DiagEntry()` function records an event in the COM short-term diagnostics that can be read out with the PADT.

Function prototype:

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

Parameter:

The function has the following parameters:

Parameters	Description
severity	It is used to classify events. 0x45 ('E') == error, 0x57 ('W') == warning, 0x49 ('I') == information
Code	The user defines the parameter code with an arbitrary number for the corresponding events. If the event occurs, the number is displayed in the diagnosis.
param1, param2	Additional information about the event.

Table 42: Parameter Diagnostics

3.7 Functions for HIMatrix and HM31

The following functions only apply to HIMatrix and HM31 controllers.

3.7.1 ComUserIRQTask

The ComUserIRQTask (CUIT) shares the code and data memory with the CUT. This has its own stack, is 32 kByte in size and is determined like the CUT stack through the COM operating system (dynamically, from the CUIT view point). The stack has a size of 32kByte. Initially, the IrqServices are disabled, i.e., after power on or after loading the configuration.

i The size of the CUIT stack must not exceed 32 kByte!
If the maximum CUIT stack allowed is exceeded, data is written to the COM memory and can lead to malfunctions!

Restrictions:

From the CUIT only the CUL functions for the semaphore handling are invoked.

3.7.1.1 CUCB_IrqService

The `CUCB_IrqService()` function is invoked by the COM module after activation of one of the two possible CAN IRQs.

This function is responsible for the operation of the `devNo` IRQ source of the corresponding CAN chip and must ensure that the CAN chip cancels its IRQ request.

Function prototype:

void CUCB_IrqService(udword devNo)

Restrictions:

The IRQ management of the COM processor is performed by the COM operating system and must not be performed by the `CUCB_IrqService()` function.

i The `CUCB_IrqService()` function must be implemented very efficiently to minimize unneeded latencies of other COM processor functions.
Otherwise, functions could no longer be performed at high COM processor load which could interfere with the safe CPU's safe communication.

The CUT library allows the COM IRQ channel to which the CAN controller is connected, to be unlocked and switched off.

3.7.1.2 CUL_IrqServiceEnable

The `CUL_IrqServiceEnable()` function enables the COM IRQ channel for the selected *devNo* CAN controller. From now on, CAN IRQs trigger the call of the CUT IRQ task.

Function prototype:

void CUL_IrqServiceEnable(udword devNo)

Parameter:

The function has the following parameter:

Parameters	Description
devNo	1 = CAN controller A 2 = CAN controller B

Table 43: Parameter CUL_IrqServiceEnable

Restrictions:

If devNo values other than 1 or 2 are used or a fieldbus interface not set with CAN is employed, CUIT/CUT are terminated.

3.7.1.3 CUL_IrqServiceDisable

The `CUL_IrqServiceDisable()` function disables the COM IRQ channel for the *devNo* CAN controller. From now on, the CAN IRQs no longer trigger the call of the CUT IRQ task. However, incompletely handled IRQs are processed.

Function prototype:

void CUL_IrqServiceDisable(udword devNo)

Parameter:

The function has the following parameter:

Parameters	Description
devNo	1 = CAN controller A 2 = CAN controller B

Table 44: Parameter CUL_IrqServiceDisable

Restrictions:

If *devNo* values other than 1 or 2 are used or a fieldbus interface not set with CAN is employed, CUIT and CUT are terminated.

3.7.1.4 CUL_DeviceBaseAddr

The `CUL_DeviceBaseAddr()` function provides the 32-bit basic address of the CAN controllers.

Function prototype:

void* CUL_DeviceBaseAddr(udword devNo)

Parameter:

The function has the following parameter:

Parameters	Description
devNo	1 = CAN controller A 2 = CAN controller B

Table 45: Parameter CUL_DeviceBaseAddr

Restrictions:

If *devNo* values other than 1 or 2 are used or a fieldbus interface not set with CAN is employed, CUIT and CUT are terminated.

3.7.2 NVRAM IF

The CUT and the CUIT can read and write the available range.

The size of the available NVRAM depends on the controller in use.

Element	HIMax as of V4	HIMatrix
NVRAM size	24576 bytes (64 kBytes -40 kBytes)	483328 bytes (512 kBytes -40 kBytes)

Table 46: Memory Area for Code and Data

i The COM **cannot** ensure data consistency if the operation voltage fails while the COM is being accessed or two tasks are accessing it simultaneously.

No difference is made between memory areas that are written often and those that are written seldom.

3.7.2.1 CUL_NVRamWrite

The `CUL_NVRamWrite()` function writes data to the NVRAM.

Function prototype:

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

Parameter:

The function has the following parameters:

Parameters	Description
Offset	Values valid in the NVRAM area, see Chapter 3.7.2.
Source	Memory area in CUT data segment which should be copied to the NVRAM.
Size	Number of bytes that should be copied.

Table 47: Parameter CUL_NVRamWrite

Restrictions:

With invalid parameters, the CUT and the CUIT are terminated, for example:

- Offset ≥ size of the available NVRAM.
- offset+size > size of the available NVRAM.
- Source out of the CUT data segment.
- Source+size out of the CUT data segment.

3.7.2.2 CUL_NVRamRead

The `CUL_NVRamRead()` function reads data from the NVRAM.

Function prototype:

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

Parameter:

The function has the following parameters:

Parameters	Description
Offset	Values valid in the NVRAM area, see Chapter 3.7.2.
destination	Memory area in CUT data segment to which the data from the NVRAM is to be copied.
Size	Number of bytes that should be copied.

Table 48: Parameter CUL_NVRamRead

Restrictions:

With invalid parameters, the CUT and the CUIT are terminated:

- Offset ≥ size of the available NVRAM.
- offset+size > size of the available NVRAM.
- Destination out of the CUT data segment.
- Destination+size out of the CUT data segment.

3.7.3 Semaphore IF

The CUT and the CUIT have **one** common semaphore for process synchronization.

A semaphore must be used to protect the data of CUCB and CUL functions that is shared with the `CUCB_IrqService()` function. This ensures that data shared with the `CUCB_IrqService()` function is consistent.

3.7.3.1 CUL_SemaRequest

The `CUL_SemaRequest()` function sends a request to the CUT/CUIT semaphore.

If the semaphore

- is available, the function returns with the pContext value.
- is not available, the invoking task is blocked until the semaphore is released by another task and is returns with the pContext value.

The context, which is referenced by the pContext parameter, is only used by the CUL functions for the invoking task and must not change between request and release.

Function prototype:

void CUL_SemaRequest(udword *pContext)

Parameter:

The function has the following parameter:

Parameters	Description
pContext	Only used by the CUL function within the invoking task. The context is returned via `pContext` and must be specified again in the `CUL_SemaRelease()` function.

Table 49: Parameter Semaphore-IF

Restrictions:

If the number of admissible recursions is exceeded then the CUT/CUIT are terminated.

If the CUT is blocked by a semaphore no more CUCB_'s are executed with the exception of the function `CUCB_IrqService()`.

3.7.3.2 CUL_SemaRelease

The `CUL_SemaRelease()` function releases again the semaphore defined by `*pContext`.

Function prototype:

void CUL_SemaRelease(udword *pContext)

Parameter:

The function has the following parameter:

Parameters	Description
`pContext`	With the same value for `*pContext` as written by `CUL_SemaRequest` or `CUL_SemaTry`.

Table 50: Parameter CUL_SemaRelease

Restrictions:

If Release is invoked more times than Request/Try, then the CUT/CUIL are terminated.

3.7.3.3 CUL_SemaTry

The `CUL_SemaTry()` function tries to request the semaphore of the CUT/CUIT.

If the semaphore

- is free the function returns with the value TRUE and reserves the semaphore.
- is not free the function returns with the value FALSE and does not reserve the semaphore.

The udword referenced by pContext is only used by CUL for the invoking task and must not be changed between request and release.

Function prototype:

bool CUL_SemaTry(udword *pContext)

Parameter:

The function has the following parameter:

Parameters	Description
pContext	Only used by the CUL function within the invoking task.

Table 51: Parameter CUL_SemaRelease

Return value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Return Value	Description
TRUE	Semaphore could be reserved.
FALSE	Semaphore could not be reserved.

Table 52: Return Value CUL_SemaTry

The context is returned via pContext and must be specified again in the `CUL_SemaRelease()` function.

Restrictions:

If the number of admissible recursions is exceeded then the CUT/CUIT are terminated.

If the CUT is blocked by a semaphore no more CUCB_'s are executed with the exception of the function `CUCB_IrqService()`.

i The functions `CUL_SemaTry()and CUL_SemaRequest()` may also be invoked without blockade if the invoking task has already reserved the semaphore; in such a case, however, the same number of CUL_SemaRelease must occur until the semaphore is available again. The recursion allows a minimum of 32000 steps. If more steps are allowed, depends on the corresponding COM version.

3.8 Installing the Development Environment

This chapter describes how to install the development environment and create a ComUserTask.

The latest development environment is included in the current HIMA DVD.

3.8.1 Installing the Cygwin Environment

The Cygwin environment is required since the GNU C compiler tools only runs under the Cygwin environment.

The Cygwin environment must be installed under Windows 7/Windows 10.

i For further details on the installation requirements, refer to Chapter 3.1. Deactivate the **virus scanner** on the PC on which Cygwin should be installed to avoid problems when installing Cygwin.

Perform the following steps to install the Cygwin environment:

To start the setup program for installing Cygwin

1. Copy the `cygwin_1.7.5-1` installation archive from the installation CD to the local hard disk (e.g., the `C:\ drive`).
2. Open the `C:\ cygwin_1.7.5-1` directory in Windows Explorer.
3. Double-click the **setup-2.697.exe** file to start the Cygwin installation.
4. Click the **Next** button in the Cygwin dialog box to start the setup.

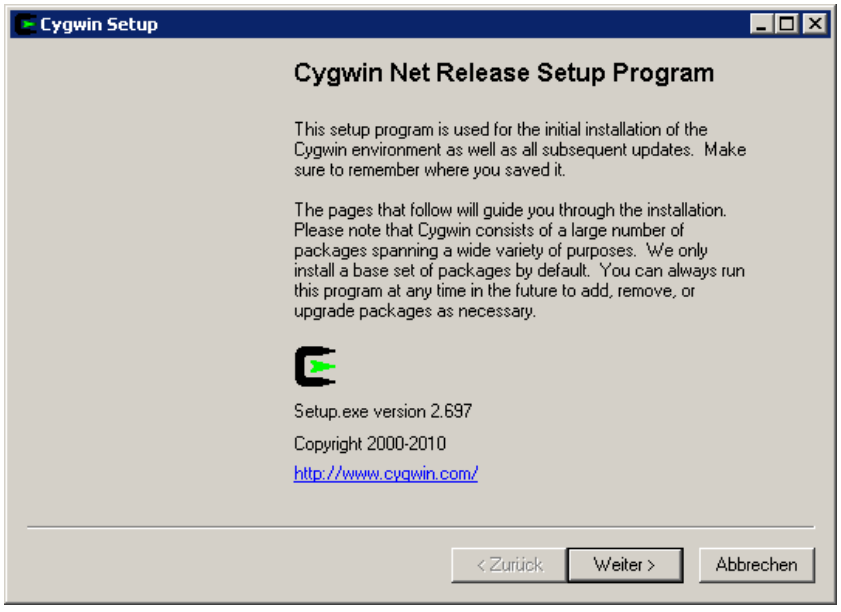


Figure 2: *Cygwin Setup* Dialog Box

The Disable Virus Scanner dialog box appears if the virus scanner was not deactivated.

Perform this step to deactivate the virus scanner during the installation of Cygwin.

i Deactivate the virus scanner before installing Cygwin since, depending on the virus scanner in use, the warning dialog box could not appear although the virus scanner is running.

1. Select **Disable Virus Scanner** to prevent potential problems during the installation due to the virus scanner.
2. Click the **Next** button to confirm.

In the *Choose Installation Type* dialog box, select the Cygwin installation source:

1. Select **Install from Local Directory** as installation source.
2. Click the **Next** button to confirm.

In the *Choose Installation Directory* dialog box, select the Cygwin installation destination:

1. Enter the directory in which Cygwin should be installed.
2. Accept all presettings of the dialog box.
3. Click the **Next** button to confirm.

In the *Select Local Package Directory* dialog box, select the Cygwin installation archive:

1. In the *Local Package Directory* field, specify the Cygwin installation archive containing the installation files.
2. Click the **Next** button to confirm.

In the *Select Packages* dialog box, select all installation packages:

1. Select the **Curr** radio button.
2. In the view box, slowly click the installation option next to **All** until **Install** is displayed for a complete installation of all packages (approx. 1.86 GB memory requirements).

i Make sure that **Install** is placed behind each package.
If the packages are not completely installed, important functions might be missing for compiling the CUT C code!

3. Click the **Next** button to confirm.

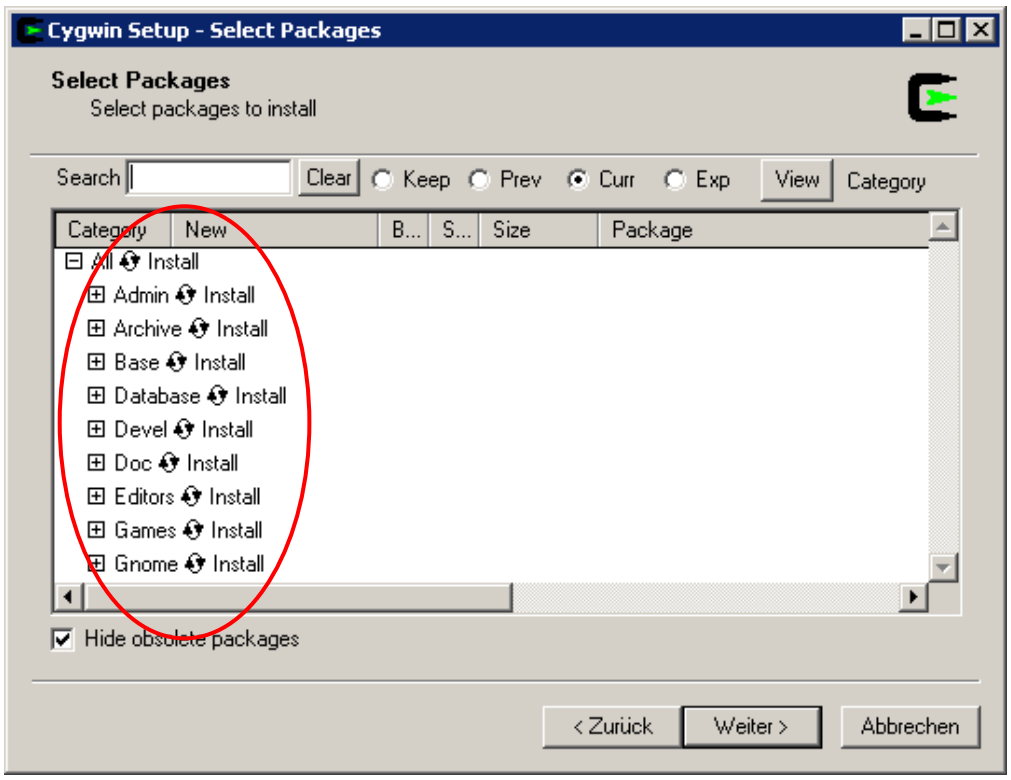


Figure 3: *Select Packages* Cygwin Setup Dialog Box

Complete the Cygwin installation with the following entries:

1. Select the entry in the **Start Menu**.
2. Select **Desktop Icon** entry.
3. Click the **Finish** button to complete the Cygwin installation.

Cygwin Commands	Description
cd (directory name)	Change directory
cd ..	Move to parent directory
ls -l	Display all files of a directory
help	Overview of bash shell commands

Table 53: Commands in Cygwin (Bash Shell)

3.8.2 Installing the GNU Compiler

To install the GNU compiler

1. In Windows Explorer, open the directory of the installation CD.
2. Double-click the `gcc-ppc-v3.3.2_binutils-v2.15.zip` zip file.
3. Extract all files in the Cygwin directory (e.g., `c:\cygwin\…`).
   The GNU compiler is unpacked in the **gcc-ppc** subfolder.
4. Set the environment variables in the control panel:
   - Use the Windows start menu **Settings->System Control->System** to open the system properties.
   - Select the **Advanced** tab.
   - Click the **Environment Variables** button.
   - In the *System Variables* field, select the **Path** system variable. Extend the system variable with `C:\cygwin\gcc-ppc\bin`.

Copy the cut_src folder from the installation CD to the home directory.
The cut_src folder contains all the 'include' and 'lib' directories required to create a ComUserTask.

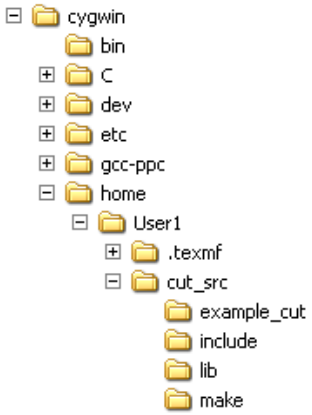


Figure 4: cygwin Structure Tree

i If the home directory was not created automatically, create it with Windows Explorer (e.g., `C:\cygwin\home\User1`).

To create another home directory for cygwin bash shell, add the `set Home` command to the `c:\cygwin\cygwin.bat` batch file.

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

Figure 5: *Cygwin.bat* Batch File

i Refer to Chapter 3.9.2.4 for further details on how to generate executable code for the *example_cut* program provided on the HIMA DVD.

3.9 Creating New CUT Projects

The *example_cut* provided in this chapter shows how to create a new CUT project and specifies which files must be adapted.

i The **example cut** project is on the HIMA DVD, completely adjusted.
Refer to Chapter 3.9.2.4 for further details on how to generate executable code for the *example_cut* program provided on the HIMA DVD.

For creating new CUT projects, HIMA recommends creating a new subdirectory of *...\cut_src* for each CUT project.

Example:

As a test, create the *example cut* directory, name the C source **example_cut.c** and the ldb file, which was created in the *make* directory, **example_cut.ldb**.

For a new ComUserTask, create the example_cut folder.

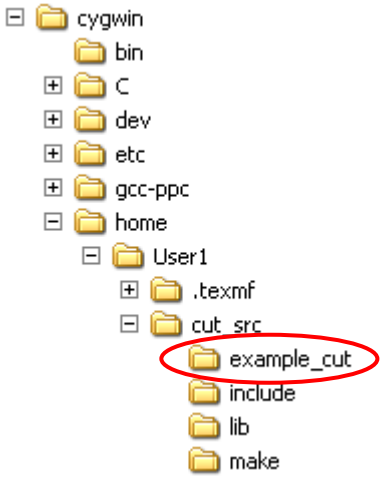


Figure 6: cygwin Structure Tree

1. **Copy the following to the example_cut directory**
   - Example_cut.c
   - Example_cut.mke
   - makefile

As described in the following chapters, perform all changes to .mke file and makefile for every new project.

3.9.1 CUT Makefiles

Configuration of CUT makefiles for different source files and ldb files.

As described in the following chapters, a total of three makefiles must be adapted.

3.9.1.1 Makefile with ".mke" Extension

The .mke file is located in the corresponding source code directory,
e.g., *cut_src\example_cut\example_cut.mke*.

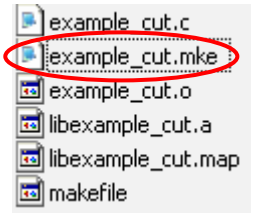


Figure 7: mke File in the example_cut Folder

Perform the following changes in the mke file:

1. The **module** variable must have the same loadable name as the .mke file (e.g., example_cut).
2. One or multiple C files required for creating the target code (loadable file) are assigned to the **c_sources** variable.

```
##############################################################################
#
# 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=
```

Figure 8: mke file

Makefile

The makefile file is located in the corresponding source code directory, e.g., `cut_src\example_cut\makefile.`

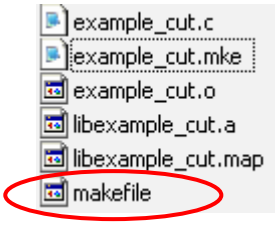


Figure 9: makefile in the *example_cut* Folder

Perform the following changes in the makefile file:

1. Drag the include line for the .mke file upwards and enter the current name for the the .mke file.
2. Expand the make call with the two variables **SUBMOD_DIRS** and **CUT_NAME**.

```
#
#
# $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

INCLUD_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
```

Figure 10: Example of makefile

3.9.1.2 Makefile with the 'makeinc.inc.app' Extension

The only change made to this and all the following CUT projects is that the name of the CUT loadable is made changeable via a make variable.

The makeinc.inc.app file is located in the cut_src directory, e.g., *cut_src\makeinc.inc.app*.

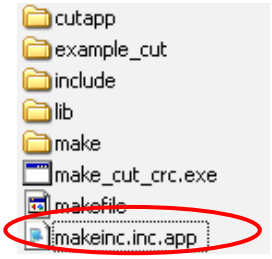


Figure 11: makeinc.inc.app File in the *example_cut F*older

Perform the following changes in the makeinc.inc.app file:

1. Expand the file with the **CUT_NAME** variable.

```
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 $$BGTYPE.map ] ; then \
echo "Error: MAP file $$BGTYPE.map does not exist"; \
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 $$BGTYPE.bin;\
echo; \
echo "Building C3-Loadable-Binary …"; \
$(MCRC) $$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: Invalid subdir. Please invoke elf target only from make/ subdirectory." && echo && false ; } ;

# end of file: makeinc.inc
```

Figure 12: makeinc.inc.app

3.9.2 Adapting C Source Codes

Perform the following steps to open the source code file:

1. Open the *cut_src\example_cut* project directory that was created and configured in the previous steps.
2. Open the C source code file with the extension .c using an editor (e.g. notepad).

3.9.2.1 Configuring Input and Output Variables

To configure the input and output variables in the source code file

1. The data size of the variables that should be created in the SILworX **Output Variables** tab, must be set in the **CUT_PDI[X]** array of the source code file.
2. The data size of the variables that should be created in the SILworX **Input Variables** tab, must be set in the **CUT_PDO[X]** array of the source code file.

3.9.2.2 Start Function in CUT

The `void CUCB_TaskLoop (udword mode)` function is the start function and is cyclically invoked by the user program.

3.9.2.3 Example Code "example_cut.c"

The following C code copies the value from the **CUT_PDI[0]** input to the **CUT_PDO[0]** output and returns the value unchanged to the SILworX user program.

---

i The **example_cut.c** C code is on the installation 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 */
```

Figure 13: C Code example_cut.c

3.9.2.4 Creating Executable Codes (ldb file)

To create the executable code (ldb file)

1. Start the **Cygwin Bash Shell.**
2. Move to the *.../cut_src/example_cut/* directory.
3. Start the code generation by specifying:
   *make cut_*HIMax for HIMax
   make cut_l3 for HIMatrix **.**
   The **cut.ldb** binary file is automatically created in the /cut_src/make/ directory.
4. If CRC32 was generated, also the executable code was generated (see red marking in Figure 14).

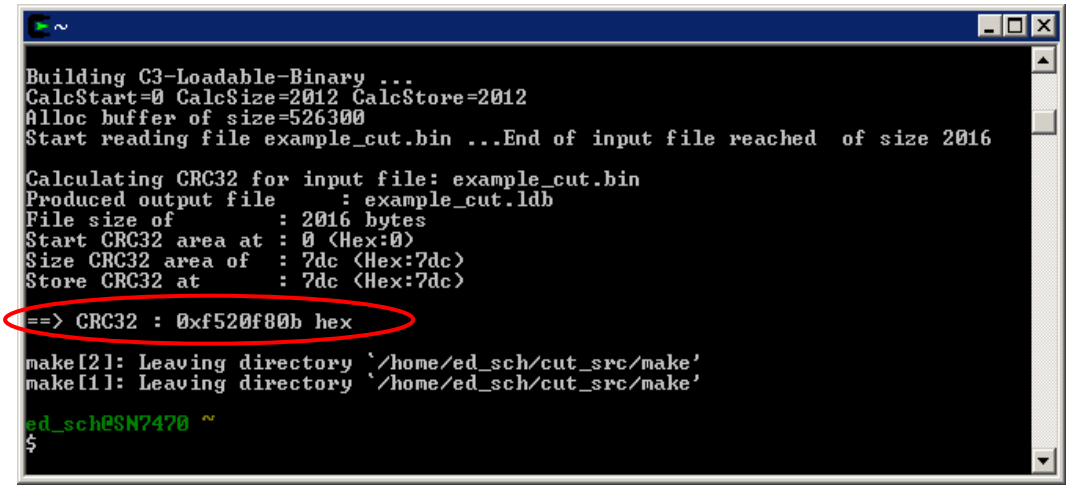


Figure 14: Cygwin Bash Shell

This executable code (ldb file) must be loaded into the ComUserTask, see Chapter 3.9.3.

3.9.3 Implementing the ComUserTask in the Project

Perform the following steps in SILworX to integrate the ComUserTask into the project:

3.9.3.1 Creating the ComUserTask

To create a new ComUserTask

1. In the structure tree, select **Configuration**, **Resource**, **Protocols**.
2. Select **New**, ComUserTask from the context menu of protocols to add a new ComUserTask.
3. Select **Properties, COM Module** from the context menu of ComUserTask.
   Accept the default settings for the first configuration.

---

i Only one ComUserTask can be created per resource.

3.9.3.2 Loading Program Code into the Project

To load a ComUserTask into the project

1. In the structure tree, open **Configuration**, **Resource**, **Protocols**.
2. Right-click ComUserTask and select **Load User Task** from the context menu. Open the *.../cut_src/make/* directory.
3. Select the **ldb file** that should be processed in the ComUserTask.

i Different versions of the ldb file can be integrated by reloading the executable code (ldb file). While the ldb file is being loaded, its content is not checked for correctness. The ldb file is then compiled in the project together with the resource configuration and can be loaded into the controller. If the ldb file changes, the project must be recompiled and reloaded.

3.9.3.3 Connecting Variables to CUT

The user can define a not safety-related process communication between the safe CPU and the not safe COM (CUT). Depending on the controller, a given data amount can be exchanged in each direction (see Chapter 3.2).

Create the following global variables:

Variable	Type
COM_CPU	UINT
CPU_COM	UINT

3.9.3.4 Connecting Process Variables

Process variables in the ComUserTask:

1. Right-click ComUserTask and select **Edit** from the context menu.
2. In the **Edit** dialog box, select the Process Variables tab.

Output Variables (CPU->COM)

The **Output Variables** tab contains the variables that should be transferred from the CPU to the COM module.

Name	Type	Offset	Global Variable
CPU_COM	UINT	0	CPU_COM

Table 54: Output Variables (CPU->COM)

1. Drag the global variables to be sent from the Object Panel onto the **Output Variables** tab.
2. Right-click a free space in the **Output Variables** area to open the context menu.
3. Select **New Offsets** from the context menu to re-generate the offsets of the variables.

Input Variables (COM->CPU)

The **Input Variables** tab contains the variables that should be transferred from the COM to the CPU module.

Name	Type	Offset	Global Variable
COM_CPU	UINT	0	COM_CPU

Table 55: Input Variables(COM->CPU)

1. Drag the global variables to be received from the Object Panel onto the **Input Variables** area.
2. Right-click a free space in the the **Input Variables** area to open the context menu.
3. Select **New Offsets** from the context menu to re-generate the offsets of the variables.

To verify the ComUserTask configuration

1. In the structure tree, select **Configuration, Resource, Protocols**, ComUserTask.
2. Right-click **Verification** to verify the CUT configuration.
3. Thoroughly verify the messages in the **logbook** and correct potential errors.

3.9.3.5 Creating the SILworX User Program

To create the SILworX user program

1. In the structure tree, open **Configuration**, **Resource**, and then select **Edit** from the context menu.
2. Drag the global variables **COM_CPU** and **CPU_COM** from the Object Panel to the drawing area.
3. Create the user program as specified in the following figure.

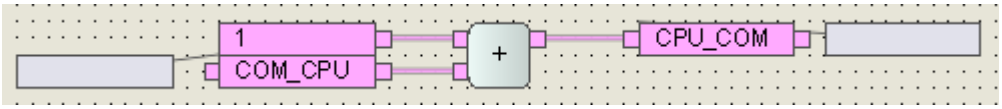


Figure 15: SILworX FBD Editor

To configure the schedule interval [ms]

1. Right-click ComUserTask and select **Properties** from the context menu.
2. In the *Schedule Interval [ms]* input box, specify in which intervals the ComUserTask should be invoked.

i Use the resource's user program to recompile the ComUserTask configuration and load it into the controller. The new configuration can only be used with the HIMA controller after this step is completed.

To check the ComUserTask with the online test

1. In the structure tree, open **Configuration, Resource.**
2. Right-click **Program** and select **Online** from the context menu. Log in to the system.

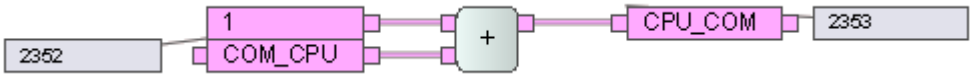


Figure 16: SILworX Online Test

Function of the SILworX user program:

The SILworX user program adds the value **1** to the **COM_CPU** signal (data inputs) and transmits the result to the **CPU_COM** signal (data outputs).

With the next CUT call (schedule interval [ms]), the signal **CPU_COM** is transmitted to the CUT function (see example code in Chapter 3.9.2).

The ComUserTask receives the signal **CPU_COM** and returns the value unchanged with the signal **COM_CPU**.

3.9.4 DCP: Error in loading a configuration with CUT

Run time problems (e.g., ComUserTask in infinite loop):

Reason for runtime problems:

Programming a loop, which runs for a long time, in the corresponding CUT source code results in a “deadlock“ of the COM processor.

As a consequence, no connection can be established to the controller and the resource configuration can no longer be deleted.

Solution: Reset the HIMax/HIQuad X communication module or the HIMatrix controller.

- In the online view of the Hardware Editor, use the **Maintenance/Service, Module (Restart)** function to reset the communication module (or the reset push-button to reset the HIMatrix system, see the data sheet of the corresponding controller).
- Create a new CUT (without runtime errors, endless loops).
- Load the CUT (ldb file) into the project.
- Generate the code.
- Load the code into the controller.

4 Synchronous Serial Interface

The synchronous serial interface (*SSI*) is a non-safety-related interface for absolute encoders (angle measurement systems).

The SSI submodule allows up to 3 absolute encoders to be connected to a common pulse and simultaneously receive the X-Y-Z position.

Prior to starting up the SSI submodule, the COM must be configured with the ComUserTask and the user program logic must be created with SILworX.

4.1 System Requirements

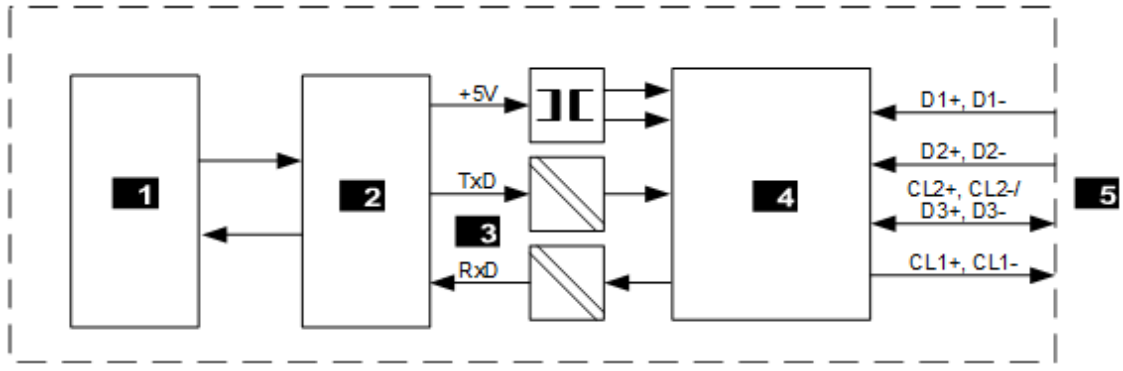
Equipment and system requirements

Element	Description
Controller	HIMax with X-COM 01 module HIQuad X with F-COM 01 module HIMatrix
CPU module	The interfaces on the processor module may not be used for SSI.
COM module	If the serial fieldbus interfaces (FB1 or FB2) are used, they must be equipped with the SSI submodule. Refer to the communication manual (HI 801 101 E) for details.
Activation	A software activation code is required for activation, refer to the communication manual (HI 801 001 E) for details.

Table 56: Equipment and System Requirements for ComUserTask

4.2 Block Diagram

The SSI submodule is galvanically separated from the HIMA controller.



1 Processor module (CPU)
2 Communication module (COM)
3 Internal serial interface
4 SSI submodule
5 SSI, FB1 or FB2

Figure 17: Block Diagram

4.3 D-Sub Connectors FB1 and FB2

If the SSI submodule is used, the pin assignment of D-sub connectors FB1 and FB2 applies. Refer to the communication manual (HI 801 101 E) for details.

4.4 Configuring Data Exchange between COM Module and SSI Submodule

Data is exchanged between the communication module (COM) 2 and the SSI submodule 4 via the internal serial interface 3 and must be implemented with the ComUserTask. Refer to Chapter ComUserTask.

The data protocol created in the ComUserTask and used to exchange data between the COM module and the SSI submodule must be set up as follows:

- Baud rate 115.2 kBit/s
- Data length: 8 bits
- Parity even
- 1 stop bit

4.5 Configuration of the SSI

The SSI 5 is configured using the SSISTART start command byte, see Table 57.

Whenever a new sensor data record (current position) is required, the SSISTART start command byte must be set in the user program and forwarded to the the SSI submodule 4.

The SSISTART start command byte to be sent has the following format:

Bit	Description
7	Auxiliary bit This bit defines the assignment of the start command byte.
If bit 7 = 1	
6 … 4	The following setting is possible for the SSI clock 000 62.5 kHz 001 125 kHz 010 250 kHz 011 500 kHz
3	It switches pins 3 and 8 either for use as a data input or as a pulsed output. 0: D3+, D3- switched for use as an input 1: CL2+, CL2- switched for use as a pulsed output
2	Pulsed output CL1 0: Activated 1: Deactivated:
1	Not used
0	Pulsed output CL2 0: Activated 1: Deactivated:
If bit 7 = 0	
6 … 0	Not used

Table 57: Data Format of the SSISTART Start Command Bytes

The SSI submodule 3 transmits the sensor data determined via the SSI to the COM 4 via the internal serial interface 2 in the following format and order.

The sensor data can be evaluated in the user program for determining the X-Y-Z position (note: the data bits are transferred to the user program in the same order as provided by the sensor).

No.	Channel	Data bit
1	Channel 1	D47…D40
2		D39…D32
3		D31…D24
4		D23…D16
5		D15…D8
6		D7…D0
7	Channel 2	D47…D40
8		D39…D32
9		D31…D24
10		D23…D16
11		D15…D8
12		D7…D0
13	Channel 3	D47…D40
14		D39…D32
15		D31…D24
16		D23…D16
17		D15…D8
18		D7…D0

Table 58: Format and Order of the Sensor Data

4.5.1 Wire Lengths and Recommended Clock Rates

The following table specifies the recommended clock rates for the SSI according with the field wire lengths.

Cable length /m	Clock rate /kHz
< 25	≤ 500
< 50	< 400
< 100	< 300
< 200	< 200
< 400	< 100

Table 59: Recommended Clock Rates According to the Field Wire Lengths

4.6 Application Notes

The SSI sensors must be supplied in the field with an external power supply since the SSI submodule is galvanically separated from the HIMA controller.

The following applications are possible with an SSI submodule:

- Connection of 3 sensors for simultaneously determining the x, y and z coordinates
  1 SSI clock (CL1+, CL1-) and 3 data channels (D1+, D1-, D2+, D2-, D3+, D3-) for simultaneously determining the x, y and z coordinates.
  All the 3 sensors are energized from the same clock (CL1+, CL1-) and thus from the same clock rate.
- Connection of 2 sensors for simultaneously determining the x and y coordinates
  2 SSI clocks (CL1+, CL1-, CL2+, CL2-) and 2 data channels (D1+, D1-, D2+, D2-) for simultaneously determining the x and y coordinates.
  Both sensors are energized from 2 different clocks (CL1+, CL1-, CL2+, CL2-). They have the same clock rate.
- Connection of 1 sensor
  1 SSI clock (CL1+, CL1- or CL2+, CL2-) and 1 data channel (D1+, D1- or D2+, D2-).

i If the special conditions X are observed, the SSI submodule may be mounted in zone 2 (EC Directive 94/9/EC, ATEX).

5 General

This chapter describes parameters that are relevant for all communication protocols.

5.1 Maximum Communication Time Slice

The maximum communication time slice is the time period in milliseconds (ms) per CPU cycle assigned to the processor module for processing the communication tasks. Even if the protocol processing could not be completed within one communication time slice, the CPU still executes the safety-relevant monitoring for all protocols within one CPU cycle.

i If not all upcoming communication tasks can be processed within one CPU cycle, the whole communication data is transferred over multiple CPU cycles. The number of communication time slices is then greater than 1.

For calculating the maximum response time, the number of communication time slices must be equal to 1.

5.1.1 Determining the Maximum Duration of the Communication Time Slice

For a first estimate of the maximum duration of the communication time slice, the sum of the following times must be entered in the *Max. Com. Time Slice [ms]* system parameter located in the properties of the resource.

- For each COM module: 3 ms.
- For each redundant safe**ethernet** connection: 1 ms.
- For non-redundant safe**ethernet** connection: 0.5 ms.
- For each kilobyte user data of non-safety-related protocols, e.g., Modbus: 1 ms.

HIMA recommends comparing the value estimated for *Max. Com. Time Slice [ms]* with the value displayed in the Control Panel and, if necessary, correcting it in the properties of the resource. This can be done during an FAT (factory acceptance test) or SAT (site acceptance test).

To determine the actual duration of the maximum communication time slice

1. Operate the HIMA system under full load (FAT, SAT):
   All communication protocols are in operation (safe**ethernet** and standard protocols).
2. Open the **Control Panel** and select the **Com. Time Slice** structure tree folder.
3. Read the vale displayed for *Maximum Com. Time Slice Duration per Cycle [ms].*
4. Read the value displayed for *Maximum Number of Required Com. Time Slice Cycles*.

The duration of the communication time slice must be set so that, when using the communication time slice, the CPU cycle cannot exceed the watchdog time specified by the process.

5.2 Load Limitation

A computing time budget expressed in % *(µP budget)* can be defined for each communication protocol. It allows the available computing time to be distributed among the configured protocols. The sum of the computing time budgets configured for all communication protocols on a CPU or COM module may not exceed 100%.

The defined computing time budgets of the individual communication protocols are monitored. If a communication protocol has already achieved or exceeded its budget and no reserve computing time is available, the communication protocol cannot be processed.

If sufficient additional computing time is available, it is used to process the communication protocol that has already achieved or exceeded its budget. It can therefore happen that a communication protocol uses more computing time budget than has been allocated to it.

It is possible that more than 100% computing time budget is displayed online. This is not a fault; the computing time budget exceeding 100% indicates the additional computing time used.

i The additional computing time budget is not a guarantee for a certain communication protocol and can be revoked from the system at any time.

Appendix

Glossary

Term	Description
ARP	Address resolution protocol, network protocol for assigning the network addresses to hardware addresses
Bit variable	Variable that is addressed bit by bit.
CENELEC	Comité Européen de Normalisation Électrotechnique (European Commitee for Electrotechnical Standardization)
COM	Communication module
Connector board	Connector board for the HIMax module.
CPU	Processor module
CRC	Cyclic redundancy check
Data view	The global variables for output and output data are assigned to a data view to allow access to Modbus sources.
EN	European standard
Export area	The export area is the process data volume that is written to by the system (a user program, hardware input or another protocol) and is read by the Modbus master.
FB	Fieldbus
FBD	Function block diagrams
ICMP	Internet control message protocol, network protocol for status or error messages.
IEC	International electrotechnical commission.
Import area	Process data volume that is written to by the Modbus master and can be used as input data for the system (in a user program, hardware output or another protocol).
Interference-free	Supposing that two input circuits are connected to the same source (e.g., a transmitter). An input circuit is termed "interference-free" if it does not distort the signals of the other input circuit.
KE	Communication end point
MAC address	Media access control address, hardware address of one network connection
NSIP	Non-safety-instrumented protocol.
PADT	Programming and debugging tool (acc. to IEC 61131-3), PC with SILworX.
PE	Protective ground
PELV	Protective extra low voltage
PES	Programmable electronic system
R	Read
R/W	Read/Write
Rack ID	Base rack identification (number)
Register variable	Variable that is addressed word by word.
SB	System bus
SFF	Safe failure fraction, i.e., portion of faults that can be safely controlled
SIF	Safety-instrumented function
SIL	Safety integrity level (in accordance with IEC 61508)
SILworX	Programming tool for HIMax, HIQuad X und HIMatrix.
SIP	Safety-instrumented protocol
SNTP	Simple network time protocol (RFC 1769)
SRS	System.Rack.Slot
SW	Software
TMO	Timeout
W	Write
WD	Watchdog
WDT	Watchdog time

Index of Figures

Index of Tables

MANUAL
ComUserTask
HI 801 521 E

For further information, please contact:

HIMA Paul Hildebrandt GmbH
Albert-Bassermann-Str. 28
68782 Brühl, Germany

Phone +49 6202 709-0
Fax +49 6202 709-107
E-mail info@hima.com

Learn more about HIMA solutions online:

www.hima.com/en/

www.hima.com