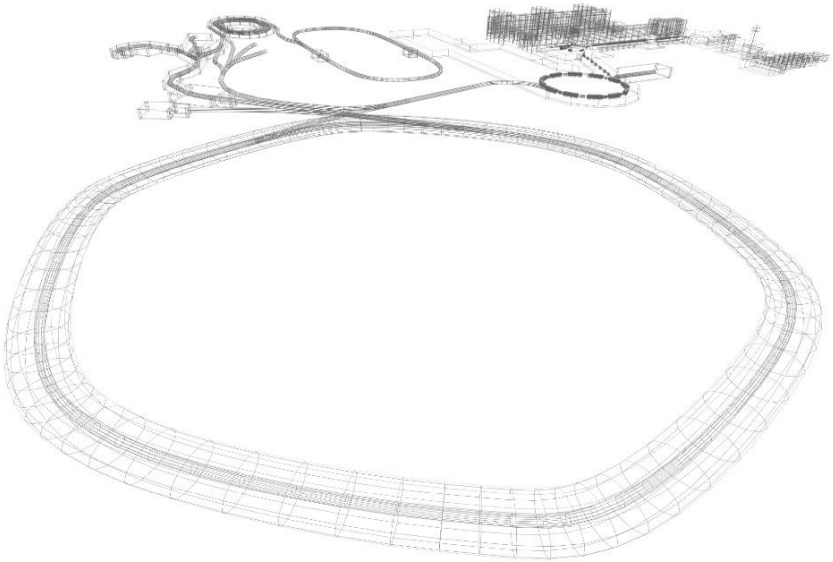
Electrical
Power
Engineering

GSI Darmstadt



Universal Serial Interface (USI 1.1) – Understanding USI

Derek Schupp

Version date: 11.09.2019, 07:47:00

Current versions can be found under:
https://wiki.gsi.de/foswiki/bin/view/EPS/ACUDocuments

Contents

1. Revision list

Date	Name	Comments
11.06.2008	D. Schupp	Document created
01.08.2008	D. Schupp	PC<-> Host communication
08.09.2008	D. Schupp	Clean up, chapter 9 driver structures is not valid
05.01.2009	D. Schupp	Several small changes, spell check
29.04.2009	D. Schupp	MDS MinorVersion changed from 'w' to 'l‘
22.06.2009	D. Schupp	Corrected HS to 4 Byte
26.01.2010	D. Schupp	Name change APC² in ACU and ACU in MFU
17.05.2010	D. Schupp	Chapter: “ The Strobe/Acknowledge Byte“ added Tables for details of the Strobe/Acknowledge Bytes
20.05.2010	D. Schupp	Checksum calculation in USI-Transmission, PID Module reset (RS) removed, chapter „Driver structures“ removed
25.05.2010	D. Schupp	Example for checksum
08.06.2010	D. Schupp	Formatting and Strobe/Acknowledge Byte for Tunnel mode
14.01.2011	D. Schupp	Document changed to USI specification and chapter “1. Stage of expansion” removed and as version 1.00 the comment ‚Draft’ deleted.
16.02.2011	D. Schupp	‚SerialNumber’ in MDS increased from 16 to 48 Bit
06.02.2011	D. Schupp	USI Module classes, Danfysik added.
29.05.2012	D. Schupp	USI 1.1 added to current version
10.06.2013	T. Keller	Translation into English
12.06.2014	D. Schupp	General revision
23.06.2014	T. Keller	Translation of the General revision 1.2 into English
03.09.2014	D. Schupp	Formatting and proofread
16.09.2014	D. Schupp	Minor syntax corrections
16.09.2015	D. Schupp	Print version
11.01.2017	D. Schupp	Add new module classes, MDS Attributes modified (Bit[7])
07.02.2019	D. Schupp	Maximum FSP depth redefined. In fact, only 4095 bytes, i.e. 4M-1 bytes are possible.
11.09.2019	D. Schupp	Switching to Office 2016, font "Universe" replaced by "Calibri" and the associated reformatting performed. USI Attributes supplemented by bit [0]. Baud replaced by bit. Document revised with respect to host / MFU.

2. Abbreviations

ACU	Adaptive Control Unit, composite of several power supply control modules, usually with a MFU as central unit.
ADC	Analogue to digital converter
DCCT	DC current transformer
FSP	FIFO Starting Point; sets a unique address for a module with a specific memory depth and function.
ICM	Interlock- and Control module, used to send control record information within the AC to the demand part of the PSU or to receive Interlocks (errors) from the PSU and shut it down if necessary.
MDS	Module Descriptor; present in every module of the FSP000 and contains the module description
MFU	Multi Function Unit, central Operating- and control unit within the ACU system
PC	Personal Computer
PSU	Power supply unit
SCU	Scalable Control Unit (Interface between ACU- and Control System)
USI	Universal serial interface, connects single power supply control modules through serial communication

3. USI (universal serial interface) description

3.1. Overview

The purpose of USI is the connection of sensors and actuators within an SVE.

USI is an asynchronous serial interface working in full-duplex which supports Baud rates between 115kBit/s and 20MBit/s, based on RS485. Furthermore, the supply voltage for the modules (24V / 500mA) might also be provided via USI. It also contains a hardware interlock wire, executed as WIRED OR that can force the power supply unit to shut down immediately. Connections use RJ45 cables and plugs. All of them are insulated. The supply voltage comes over a DC/DC converter to ensure the sensor- and actuator modules are potential-free.

3.2. Definition of Host

All modules must be connected to a host through which they can communicate with each other. It is usually not possible for modules to bypass the host and communicate directly.

In an ACU system, the MFU is usually the host. If this is connected via USB to a PC or via the backplane to a control system, the PC or the control system can form the host.

It is also possible to connect modules without an MFU directly to a PC via RS485/USB cable. This will happen mainly for debug and developer reasons. In this case, the PC is to be viewed as a host.

In practical operation, however, a MFU is normally in the system and forms the host.

Depending on whether modules are connected to a PC/control system via an MFU or directly via RS485/USB cable, the addressing of the module differs slightly. Information on this can be found in chapter 5.7. "USI Packages at Normal / Standard USI Transmission".

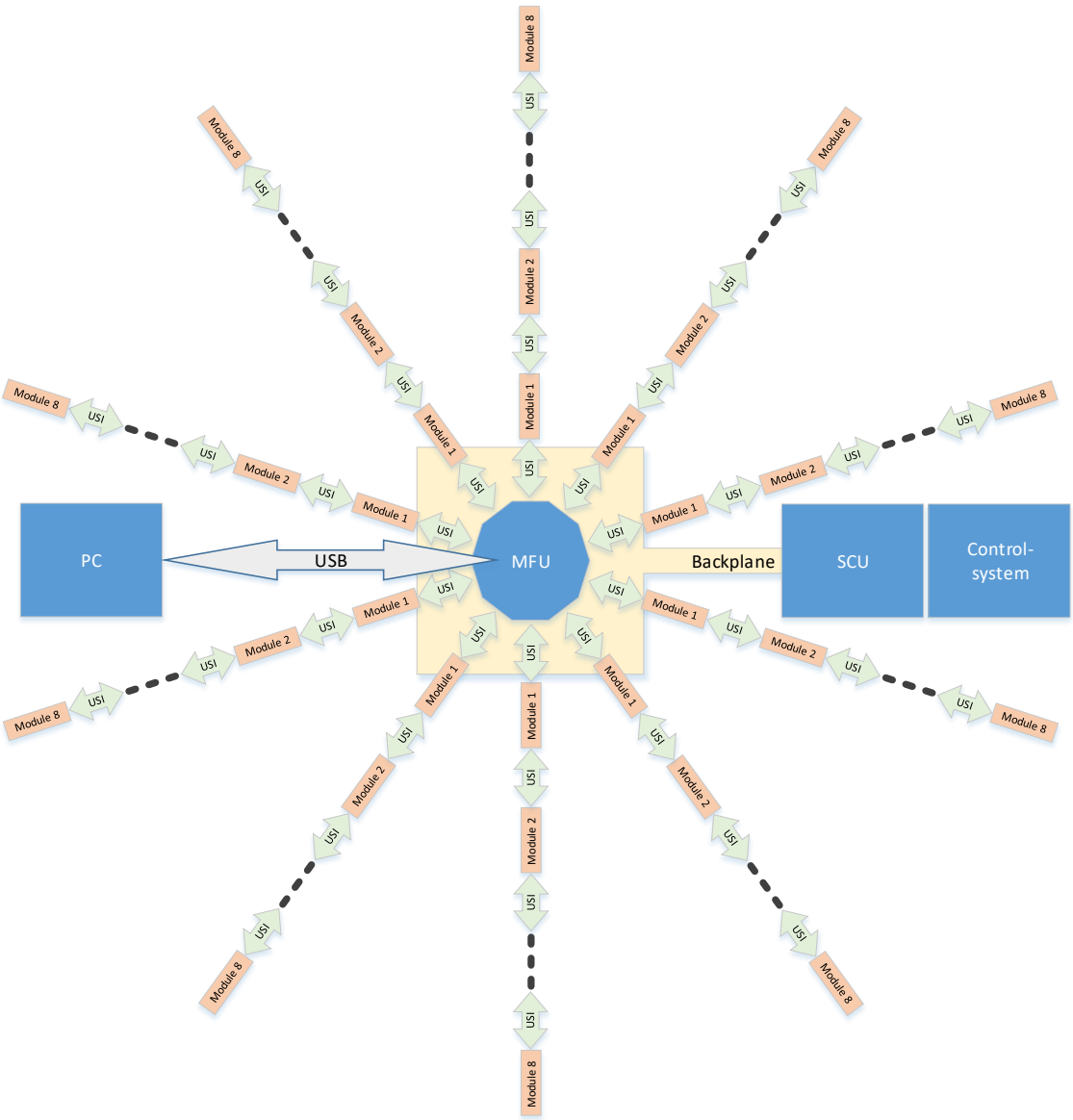


Figure 3.1: Connections via USI

With USI a star shaped architecture around the MFU is possible. The MFU supports up to 10 external USI channels at the same time.

3.3. Details

In the following, all different USI parts will be described.

Because modules in an ACU system are usually connected to an MFU as host, this topology is considered below.

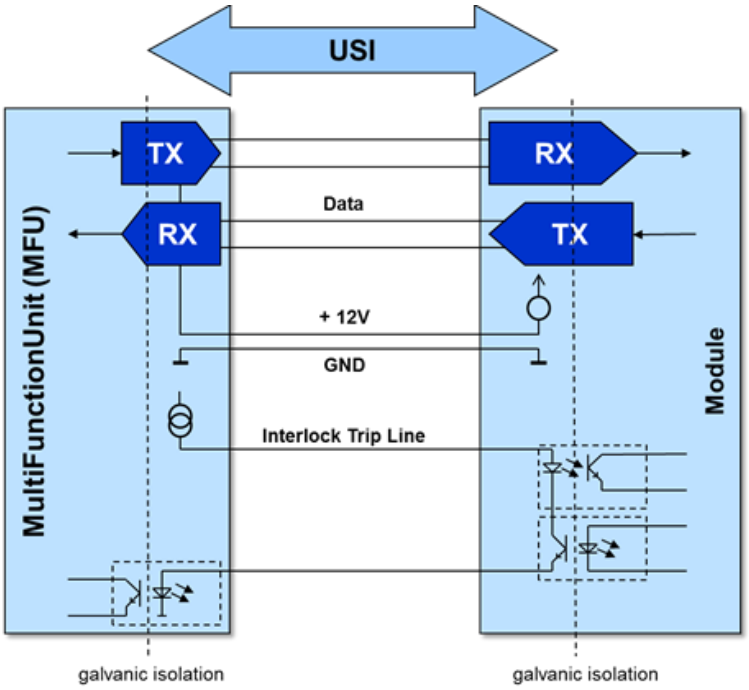


Figure 3.2: Detailed structure of USI between MFU and module

3.3.1. Power supply for secondary part of USI

Because of the galvanic separation of the USI, a power supply on both sides becomes necessary. The primary side of the USI interface on the MFU is being fed by the MFU power supply. The secondary side has to get its voltage from the module. For that, at least one module of the USI provides power through 2 wires of the network cable. The MFU backplane requires 3,3 Volt, which are provided via a series regulator. The voltage from the module needs to be high enough to compensate power losses and voltage drop at the series regulator. Figure 3.2 shows 12V for this voltage. If the wiring is short and efficient enough, 5V could also be enough.

Please note: In USI 1.0, these two wires have been used for the “PowerOverUSI” function. The MFU provides this voltage and supplies all connected modules. Therefore, from USI 1.1 on, the voltage direction is reversed.

3.3.2. Sum-Interlock-Wire

The Sum-Interlock-Wire gives all modules that are connected to the MFU, the ability to shut down the power supply unit. The wires are constructed as constant current supplies.

Once a module detects an error in the hardware, the connection interrupted via an optocoupler on the module to inform all participants of the respective system. The MFU recognises this interruption and informs the other modules by pulling the trigger lines of all the USIs. The actual turn off the PSU is usually done by the ICM.

Important note: This procedure has to work independently from the modules software.

All the Sum-Interlock-Wires are connected to each other on the MFU backplane, so it is also possible to detect broken cables for example.

3.3.3. Data supply

USI is a serial full-duplex interface that is used for the communication between modules and the host (usually the MFU).

All commands, data and status information are being sent and received with this interface. It is solely a Master/Slave interface, so the modules cannot request any information from the host.

The interface is technically an RS485 which, in the case of the MFU, electrically isolates the modules from it via iCoupler. The separation takes place on the module.

By default, the bit rate is set to 115.2kBit with 8 bits, 1 stop bit and no parity bit when the module is powered up. Next, the host requests the highest possible Baud rate from the module to ensure an ideal communication speed. The possible bit rates are between 115.2kBit and 20MBit.

3.3.4. Plug & Play on the MFU

USI is a Plug & Play interface, so it may be placed on any slot at the MFU. The MFU uses a system description to identify automatically the module. As a result, the internal data lines are configured according to the assignments implemented in the MFU's own configuration.

The system description is required, since multiple modules can be present that might be of the same type but serve several different purposes. This description is implemented in a permanent memory at the MFU.

To preserve this functionality, each module needs a standard register and standard commands (MSP, FSP).

3.3.5. Connection MFU ↔ Module

To connect to modules with the MFU, a standard CAT5 network cable is used.

4. Principle of transfer

4.1. *FIFO Start Points (FSP)*

A FIFO Start Point (FSP) is a memory (register) with a fixed data depth at a defined starting point. The FIFO depth is always set to full bytes, so at least its depth is 1 (Byte).

The starting points of the FSP start at address 0. At this point (FSP000) the Module Description Structure (MDS) can be found. Every module has to support MDS and the MDS is “read only”. All other FSPs are either, read-only, write-only and bidirectional. Information about the possible transfer directions can be found in the MDS in the related FSP-descriptor.

One module can have a maximum number of 256 FSPs. This means that the address of an FSP has a size of one Byte. FSPs 000 to 059 are universally valid and their functions are mostly generalised. Beginning with FSP060, the FSP are specific to each module and therefore mentioned in the module description. Please refer to document “ACU-FSPs mUSIC”. The FSPs of all present modules are described in it.

During addressing of a module via USI, a module number [1..8]and a certain FSP number is defined. That way, each FSP of a module can be addressed uniquely.

FSP Nubmer	Startpoint-Addresse	Depth	Direction
000 (MDS)	0x00	depends on the module	readable
001	0x01	3 Byte/24 Bit	readable
…			
010	0x0A	2 Byte/16 Bit	writable/readable

Table 4-1: FSP examples

Table 4-1 shows 3 examples for Module FSPs. FSP000 (the MDS) can be found at the starting point address 0x00. The depth of the FSP depends on the module that uses it (refer to chapter 6: “MDS (Module Descriptor Structure)”). This FSP is read-only. FSP001 is also read-only and is located at address 0x01. In this example its depth is 3byte/24bit. This means during a read process 3 Byte/24bit are sent. This data can contain status information about the muddle. FSP010 is located at address 0x0A, has a depth of 2Byte/16bit and is a read/write FSP. During a write access 2Byte/16bit have to be sent to this FSP. For example, such an FSP is used to configure a module.

4.2. *Important agreements*

If we look at an FSP, the direction of data transfer is always defined from the modules point of view. So, FSP outputs in modules have the transfer direction OUT and inputs IN. Bi-directional FSPs are INOUT.

Starting Point No	Direction of transfer	Starting Point Address	Description
FSP000*) [MDS]	OUT	0x00	This FSP is reserved as one directional, which means this FSP is read-only and contains the MDS (Module Description Structure).
FSP001*)	OUT	0x01	FSP001 is generally used for the module status and can only send data to the host.
FSP010*)	INOUT	0x0A	The FIFO that belongs to the FSP010 can be written and read and is used for transferring commands.
FSP064	OUT	0x40	FSP064 is specific to modules and is read-only.
…	…	…	…

Table 4-2: Example overview

*) FSP000 to FSP59 universal and therefore the direction of transfer is fixed.

4.3. FSP structure inside USI modules

In reference to table 4.2, the following shows the FSP structure of USI modules:

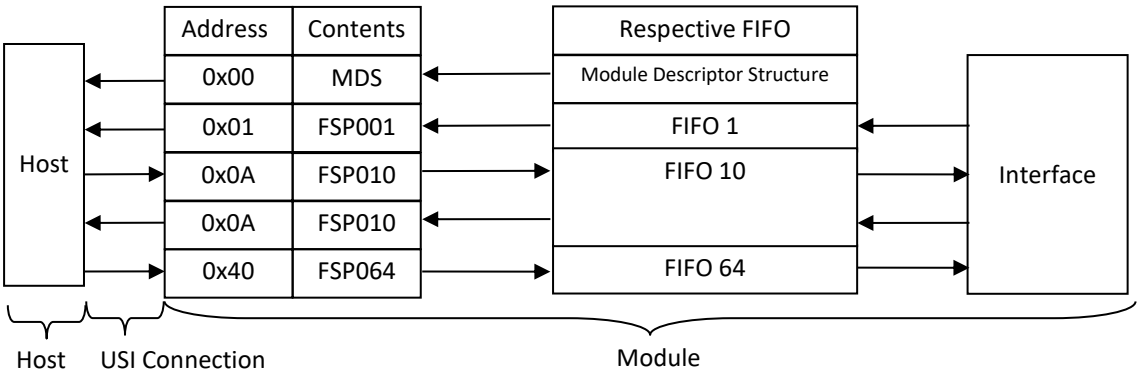


Figure 4.1: Memory structure of USI modules

This concept has the advantage that the memory depth behind an FSP can be varied, in case it isn't sufficient for later applications.

FSP000 contains the MDS (Module Descriptor Structure). The memory depth is variable and depends on the standard descriptors it contains.

FSP001 belongs to FIFO 1, is universal and has a memory depth of 3 Byte (or 24 bit). For example, it could belong to a status of an ADC module. This FSP is read-only to the host. It is important to note that when a host addresses FSP001 and wants to read it out, it is always done entirely. Therefore, all 3 Byte are being transmitted.

FSP010 belongs to FIFO10, is universal and its memory depth can be defined freely, e.g. 2 Byte. It contains module commands which can be read and written by the host. The same important note as for FSP001 applies here, so all, in this case 2, Byte are being sent and have to be received by the host.

4.4. Depth of FSP

The maximum depth of freely definable FSPs is 4095 Byte. The counting of the byte depth starts with 1 and ends with 4095 (2 ^ 12 bits), so a maximum of (4MByte - 1 Byte) is possible.

FSP000 to FSP059 are predefined regarding functionality. Their depths can be adjusted to the module requirements.

5. The USI Protocol

5.1. *Definition of the direction during data transfer*

In the following, the direction of data transfer is always defined from the modules point of view.

FSP for interfaces in modules that are only able to receive data are Receiver FSPs. An example would be a DAC module that receives data from the host to process them further. On side of the host this FSP is being served with an OUT while writing. In the module itself this FSP is INOUT, because all write-able FSPs in the modules can be read back. If the host reads back the module's FSP, it is done with an IN, which makes the transfer channel at the host finally INOUT, too.

If the host is supposed to receive data from a read-only module FSP, we have an OUT at the module side. The example for that could be an ADC, since it gathers data, converts it and forwards them to the host. So, on side of the host, we have an IN.

5.2. *USI protocols*

The USI protocol possesses 2 different formats. On one hand we have "Normal USI" or "Standard USI" and on the other hand "HighSpeed USI" transfers.

Before each data transfer all USI modules have to be identified clearly be the host. This means the host (usually the MFU) has to have all the necessary drivers and other information to communicate with the modules.

At this point we assume that all modules are recognized by the USI and are successfully integrated into the system.

Information about searching for modules can be found on page 30 in chapter 7, "Enumeration and Plug & Play".

5.3. *Data format*

All data is transmitted in ASCII code, so it is readable in plaintext. This should make any diagnosis in case of an error a lot easier. As a consequence of the ASCII coding, all data has to be transmitted as Byte.

5.3.1. *ASCII or Hex*

Internally, the module processes data in hexadecimal form. This means that an FSP with, for example 4 Byte depth, stores the information $0xCA38F8F6_H$ as follows:

4. Byte (MSB)				3. Byte				2. Byte				1. Byte (LSB)																			
C				A				3				8				F				8				F				6			
1	1	0	0	1	0	1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0	1	1	0
31			28	27			24	23			20	19			16	15			12	11			8	7			4	3			0

Figure 5.1: FSP internal data management

Still, via USI, this data is transmitted as ASCII symbols. For that, 4 Bit (1 Nibble) of FSP data is put together and converted into a single ASCII symbol.

These 4 Bit can take 16 different hexadecimal states and each of them stands of a hexadecimal ASCII symbol.

Nibble		ASCII	
Binary	Hex	Symbol	Hex
0000	0x0	0	0x30
0001	0x1	1	0x31
0010	0x2	2	0x32
0011	0x3	3	0x33
0100	0x4	4	0x34
0101	0x5	5	0x35
0110	0x6	6	0x36
0111	0x7	7	0x37
1000	0x8	8	0x38
1001	0x9	9	0x39
1010	0xA	A	0x41
1011	0xB	B	0x42
1100	0xC	C	0x43
1101	0xD	D	0x44
1110	0xE	E	0x45
1111	0xF	F	0x46

Table 5-1: Hexadecimal Nibble as hexadecimal ASCII symbol

In that case, the 32 Bit of hexadecimal data stored in the FSP will be transmitted via USI as hexadecimal ASCII symbols as follows:

C	A	3	8	F	8	F	6
0x43	0x41	0x33	0x38	0x46	0x38	0x46	0x36

Figure 5.2: Example of data transmission in ASCII

It is evident that the data volume is doubling due to the conversion from hexadecimal Nibble to hexadecimal ASCII symbols.

To minimize the decoding effort, all characters of the hexadecimal ASCII code have to be capital letters.

5.4. Control symbols

A USI package does not only contain ASCII data, but also ASCII control symbols. These indicate the start and end of a package.

ASCII Control Symbol	Type	hexadecimal	Description	USI mode
STX	Start of Text	0x02	Start of transmission	Normal/ HighSpeed
ETX	End of Text	0x03	End of transmission	
ENQ	Enquiry	0x05	Start of tunnel transmission	Tunnel marker in HS Package
EOT	End of Transmission	0x04	End of tunnel transmission	
ACK	Acknowledge	0x06	The module confirms successful data transmission	Normal/ HighSpeed
NAK	No Acknowledge	0x15	The module states the data transmission has failed, followed by an error code.	

Table 5-2: ASCII Control symbols

5.5. *Package Identifier*

A send or receive request to a USI module is on the host side always preceded by a Package-Identifier (PID). With that, the host signals whether it wants to send to or receive from the module.

A PID consists of a 16-Bit identification field, which contains the PID as an ASCII symbol in plaintext.

PID-Name	Type	PID shown as ASCII symbol	hexadecimal	Description
RD	Read	RD	0x52 0x44	Host expects to receive data from module
WR	Write	WR	0x57 0x52	Host sends data to module

Table 5-3: PID Overview

5.6. *Checksum calculation*

To verify a correct data transmission, all data being sent is followed by a checksum. This checksum is created with an exclusive Or (XOR). All ASCII data bytes are XOR-connected and the final result is given as 1 Byte hexadecimal value in 2 Byte hexadecimal ASCII form.

Table 5-4 shows an example of a checksum calculation for 8 Byte of ASCII data. Please notice that while the data is present as ASCII symbols, they are interpreted hexadecimal. By using the XOR conjunctions, we get the 1 Byte hexadecimal result that is send as 2 Byte hexadecimal ASCII symbol.

Important note: Only the reference data is used for the checksum calculation, so no PID, no USI-Module or FSP addresses can be found in the checksum.

Data		Checksum calculation in hexadecimal (Sum XOR next hex. ASCII value)							Checksum (ASCII)
Hex.	ASCII in Hex.								
1	0x31	0x03							
		XOR							
2	0x32		0x30						
			XOR						
3	0x33			0x04					
				XOR					
4	0x34				0x31				
					XOR				
5	0x35					0x37			
						XOR			
6	0x36						0x30		
							XOR		
7	0x37							0x08	
							XOR		
8	0x38								
								Result	0x30 0x38

Table 5-4: Example for calculating a Checksum

5.7. USI Package for Normal/Standard USI Transmission

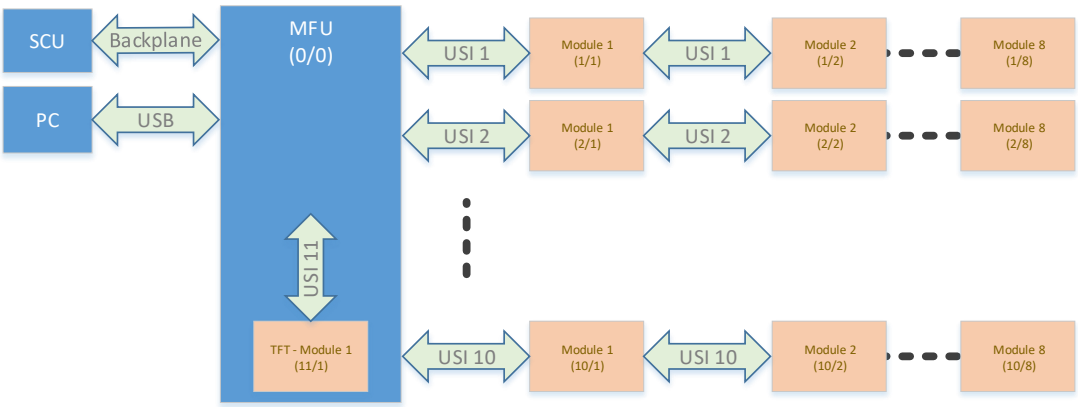


Figure 5.3: ACU Topology

The figure above shows a typical ACU topology.

The normal or standard USI transmission takes place between the MFU and the individual participants on the USI buses, but also between the PC and the MFU.

Figure 5.4: Direct connection of a PC with modules

It is also possible to connect a PC directly to a module/modules via USB/USI adapter cable. In this case, standard USI transmission is also commonly used.

5.7.1. Phases of a standard USI transaction between host and connected subscriber

In Figure 5.3 usually the MFU is the host. If a PC or a higher-level control system is connected by means of SCU and want to access modules via USI, these form the host. In this case, the MFU is to be regarded as a module at the USI.

In Figure 5.4, the PC is the host.

5.7.1.1. Direct data transfer between host and module

In the following, either the direct data transfer via USI between the MFU as host (Figure 5.3) or a PC↔Modul direct connection via a USB/USI cable with a PC as host (Figure 5.4), is considered. The data transfer from the PC via the MFU to a module is described in chapter 5.7.1.2.1: "Sending data from PC/Control System to the ".

Each data transfer via USI consists of several phases.

The host explicitly initiates the first phase, since it is the only part of the system that has the permission to start a transfer. This happens by sending a StartOfText (STX), followed by a WR/RD PID.

After that, the module address follows and number/address of the FSP which is to be addressed.

If the host wants to receive module data, it ends the first phase itself. Therefore, an EndOfText (ETX) is sent and the host waits for the answer of the module.

If the host wants to send data, in the second phase the data that should be written to the FSP follows. It is important that the data length is always equal to the depth of the FSP. 4 Byte depth for example requires 8 ASCII symbols. This is the second phase.

The second phase is completed by sending of the checksum and ETX.

If the host has sent data, the third phase is the confirmation of the module that it has received all the data correctly.

If the host wants to receive data, the second phase consists of the answer of the module. This second phase also starts with an STX, the module address follows as well as the number/address of the FSP. This is followed by the data of the FSP. This phase is completed by sending of the checksum and ETX.

5.7.1.1.1. Sending of data from the host to a module

Format: STX WR MA FSP DD... PP ETX

"STX" StartOfText, Package Start as 1 Byte ASCII

"WR" Command to tell the module that data will be written. PID as 2 Byte ASCII

"MA" Module address in hex as 1 Byte ASCII

"FSP" The desired FSP in Hex as 2 Byte ASCII

"DD..." At least 2 Bytes ASCII that are supposed to be written in the previously addressed FSP. If the memory depth is larger than 1 Byte, the following ASCII data is transmitted immediately.

"PP" Checksum, build from XOR conjunctions of the sent data, present in hex and transmitted as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync.	PID	MA	FSP	DD...	PP	Sync.
STX	WR	1	1	87654321	08	ETX
0x02	0x57 0x52	0x31	0x30 0x31	0x38 0x37 0x36 0x35 0x34 0x33 0x32 0x31	0x30, 0x38	0x03

Table 5-5: Sending data from the host to a module

Example: "STX WR 1 01 87654321 08 ETX"

If the transmission was fully correct, the module will answer with an “ACK”. Please refer to chapter 5.7.1.1.2 “Module confirms successful reception of data”.

If the module answers with a “NAK” (please refer to chapter 5.7.1.1.3 “Module has not successfully received data”) or gives no response at all, the host has to start another try to send after a specific waiting period.

5.7.1.1.2. Module confirms successful reception of data

Format: STX MA ACK FSP ETX

"STX" StartOfText, Package Start, as 1 Byte ASCII

"MA" Module address in Hex, as 1 Byte ASCII

"ACK" Control symbol that tells the host that the module has successfully received and processed the sent data, as 1 byte ASCII

"FSP" Desired FSP in Hex, as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync.	MA	ACK	FSP	Sync.
STX	1	6	1	ETX
0x02	0x31	0x06	0x30 0x31	0x03

Table 5-6: Module confirms successful reception of data

5.7.1.1.3. Module has not successfully received data

Format: STX NAK MA ERR ETX

"STX" StartOfText, Package Start, as 1 Byte ASCII

"MA" Module address in Hex, as 1 Byte ASCII

"NAK" Control symbol that tells the host that the module has not successfully received and processed the sent data, as 1 byte ASCII

"ERR" Error code in Hex, as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync.	MA	NAK	ERR	Sync.
STX	1	21	04	ETX
0x02	0x31	0x15	0x30 0x34	0x03

Table 5-7: Module has not successfully received data

Table 5-7 shows the response of a module that has not successfully received data and gives back error code 4. All available error number can be found in Table 5-15: “Error number” on page 20.

5.7.1.1.4. Host wants to read data from a module

Format: STX RD MA FSPx ETX

"STX" StartOfText, Package start, as 1 Byte ASCII

"RD" Command to let the module know that its data will be read by the host. PID as 2 Byte ASCII

"MA" Module address in hex, as 1 Byte ASCII

"FSP" Desired FSP in hex, as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync	PID	MA	FSP	Sync.
STX	RD	1	1	ETX
0x02	0x52 0x44	0x31	0x30 0x31	0x03

Table 5-8: Sending of host data to a module

Example "STX RD 1 01 ETX"

If the communication was fully correct, the module will respond with the requested data.

If the module responds with “NAK” (refer to chapter 5.7.1.1.3: “Module has not successfully received data”) or does not answer at all, the host has to send another request after a specific waiting period.

5.7.1.1.5. Sending data from a module to the host

It is required that the host has requested data from the module, as described in the last chapter 5.7.1.1.4: “Host wants to read data from a module”.

Format: STX MA FSP ZZ... PP ETX

"STX" StartOfText, Package start, as 1 Byte ASCII

"MA" Module address in hex, as 1 Byte ASCII

"FSP" Desired FSP in hex, as 2 Byte ASCII

"DD..." At least 2 Bytes ASCII, which are supposed to be written in the previously addressed FSP. If the memory depth is larger than 1 Byte, the following ASCII data is transmitted immediately.

"PP" Checksum, build from XOR conjunctions of the sent data, present in hex and transmitted as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync	MA	FSP	DD...	PP	Sync
STX	1	1	87654321	08	ETX
0x02	0x31	0x00, 0x31	0x38 0x37 0x36 0x35 0x34 0x33 0x32 0x31	0x30 0x38	0x03

Table 5-9: Sending of module data to the host

Example "STX 1 01 87654321 08 ETX"

5.7.1.2. Phases of a USI transaction between PC/Control System and the ACU System

The following describes the data transfer via USI between a PC or the Control System as host (Figure 5.3) and the ACU system. The direct data transfer between a host and a module is described in chapter 5.7.1.1: "Direct data transfer between host and module".

The data transfer between PC/Control System and host happens analogue to the one between host and module.

The only difference is that the MFU works as a gateway and forwards all data to both the addressed USI and respective module or keeps them, in case the host itself is the recipient.

For that reason, an additional ASCII symbol is added to the USI package. Either it addresses the USI or the MFU before a module is addressed.

The transfer phases look as follows:

During the first phase, the PC/Control System explicitly initiates the transfer. For that, a StartOfText (STX) is sent, along with a WR or RD PID.

After that, the gateway address (GW) follows, which is either an USI number (1_D..11_D, $0x1_H$..$0x9_H$, $0xA_H$, $0xB_H$) or 0_D ($0x0_H$), in case the MFU itself is the recipient.

Now the module address and the number/address of the addressed FSP are sent. Both gateway address (GW) and module address (MA) are 0 in case the MFU itself is the recipient.

If the PC/Control System wants to receive data, it ends the first phase with an ETX and waits for the ACU system to respond.

For response, the MFU forwards the request to the desired module, given that a USI was addressed, and then passes its answer on to the PC/Control System. Again, the MFU works as a gateway.

If the PC/Control System wants to send data, the second phase contains all data that are supposed to be written to the FSP at the target module. Again, the length of the data has to match the depth of the FSP. If the destination FSP is e.g. 4 bytes deep, now follow 8 ASCII characters. Each character contains a data nibble.

The second phase will be concluded by sending a test sum. With an adjoining ETX.

If the PC/Control System has sent data, the third phase is either an acknowledgement or an error code, coming from either the MFU or the module to which the request had previously been forwarded.

If the PC/Control System wants to receive data, the second phase consists of the response of the MFU or the respective module. It is started by sending an STX, followed by the USI number as GW (1_D..11_D, $0x1_D$..$0x9_H$, $0xA_H$, $0x0B_H$) or 0_D ($0x0_H$), if the host was addressed. Next is the module address (MA) and the number/address of the FSP, followed by the FSP data.

This phase is completed by sending a checksum and a final ETX.

5.7.1.2.1. Sending data from PC/Control System to the ACU System

Format: STX WR GW MA FSP ZZ... PP ETX

"STX" StartOfText, Package start, as 1 Byte ASCII

"WR" Command to tell the host that data is to be written. PID as 2 Byte ASCII

"GW" Gateway address, as 1Byte ASCII

"MA" Module address in hex, as 1 Byte ASCII

"FSP" Desired FIFO in hex, as 2 Byte ASCII

"DD..." At least 2 Bytes ASCII which are supposed to be written in the previously addressed FSP. If the memory depth is larger than 1 Byte, the following ASCII data is transmitted immediately.

"PP" Checksum, build from XOR conjunctions of the sent data, present in hex and transmitted as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync.	PID	GW	MA	FSP	DD...	PP	Sync.
STX	WR	1	1	01	87654321	08	ETX
0x02	0x5752	0x31	0x31	0x00, 0x31	0x38 0x37 0x36 0x35 0x34 0x33 0x32 0x31	0x30 0x38	0x03

Table 5-10: Sending of data to the ACU System

Example "STX WR 1 1 01 87654321 08 ETX"

If the transmission was fully correct, the MFU/module will answer with an “ACK”. Please refer to chapter 5.7.1.2.2: “MFU or module confirms successful reception of data”.

If the MFU or module answers with a “NAK” (please refer to chapter 5.7.1.2.3: “MFU or module has not successfully received data”) or gives no response at all, the PC/Control System has to start another try to send after a specific waiting period.

5.7.1.2.2. MFU or module confirms successful reception of data

Format: STX GW ACK MA FSP ETX

"STX" StartOfText, Package start, as 1 Byte ASCII

"GW" Gateway address, as 1Byte ASCII

"MA" Module address in hex, as 1 Byte ASCII

"ACK" Control symbol that tells the host that the module has successfully received and processed the sent data, as 1 byte ASCII

"FSP" Desired FSP in hex, as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync.	GW	MA	ACK	FSP	Sync.
STX	1	1	6	01	ETX
0x02	0x31	0x31	0x06	0x30 0x31	0x03

Table 5-11: MFU or module confirms successful reception of data

5.7.1.2.3. MFU or module has not successfully received data

Format: STX GW NAK MA ERR ETX

"STX" StartOfText, Package start, as 1 Byte ASCII

"GW" Gateway address, as 1Byte ASCII

"MA" Module address in hex, as 1 Byte ASCII

"NAK" Control symbol that tells the host that the module has successfully received and processed the sent data, as 1 byte ASCII

"ERR" Error code in hex, as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync.	GW	MA	NAK	ERR	Sync.
STX	1	1	21	04	ETX
0x02	0x31	0x31	0x15	0x30 0x34	0x03

Table 5-12: MFU or module has not successfully received data

Table 5-12 shows the response of the MFU or module that has not successfully received data and sends back the error code 4. All available error numbers can be found in Table 5-15.

5.7.1.2.4. PC/Control System wants to read data from the MFU or a module

Format: STX RD GW MA FSP ETX

"STX" StartOfText, Package start, as 1 Byte ASCII

"GW" Gateway address, as 1Byte ASCII

"RD" Command to tell the host or a module the data will be read. PID as 2 Byte ASCII

"MA" Module address in hex, as 1 Byte ASCII

"FSP" Desired FSP in hex, as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync	PID	GW	MA	FSP	Sync.
STX	RD	1	1	01	ETX
0x02	0x52 0x44	0x31	0x31	0x00 0x31	0x03

Table 5-13: Reding of data

Example "STX RD 1 1 01 ETX"

If the communication was fully correct, the MFU or module will respond with the requested data.

If the MFU or module responds with “NAK” (refer to chapter 5.7.1.2.3: “MFU or module has not successfully received data”) or does not answer at all, the PC/Control System has to send another request after a specific waiting period.

5.7.1.2.5. Sending data from the MFU or a module to the PC/Control System

It is required that the PC has requested data from the host or a module. Refer to chapter 5.7.1.2.4 “PC/Control System wants to read data from the MFU or a module”

Format: STX GW MA FSPx ZZ... PP ETX

"STX" StartOfText, Package start, as 1 Byte ASCII

"GW" Gateway address, as 1Byte ASCII

"MA" Module address in hex, as 1 Byte ASCII

"FSP" Desired FSP in hex, as 2 Byte ASCII

"DD..." At least 2 Byte ASCII which are supposed to be written in the previously addressed FSP. If the memory depth is larger than 1 Byte, the following ASCII data is transmitted immediately.

"PP" Checksum, build from XOR conjunctions of the sent data, present in hex and transmitted as 2 Byte ASCII

"ETX" EndOfText, Package End, as 1 Byte ASCII

Sync.	GW	MA	FSP	DD...	PP	Sync.
STX	1	1	1	87654321	08	ETX
0x02	0x31	0x31	0x00, 0x31	0x38 0x37 0x36 0x35 0x34 0x33 0x32 0x31	0x30 0x38	0x03

Table 5-14: Sending of data to PC/Control System

Example "STX 1 1 01 87654321 08 ETX"

5.7.2. *Error numbers*

If an USI transfer fails and the module sends a NACK, this response is followed by an error number.

Error number	ASCII	Comment
87_D / 57_H	0x35 0x37	FSP not present
85_D / 55_H	0x35 0x35	Data length does not match FSP depth
80_D / 50_H	0x35 0x30	No data allowed, FSP is supposed to be read, PID is “RD”
04_D / 04_H	0x30 0x34	PID not understood, MA is correct, but PID is unkown
75_D / 75_H	0x37 0x35	Error during ASCII to Hex conversion, data is not ASCII
83_D / 53_H	0x35 0x33	FSP is read-only, no write access allowed
12_D / 12_H	0x31 0x32	Error during CRC calculation

Table 5-15: Error number

5.8. *High-speed transfer of USI packages*

If data is supposed to be send very quickly from a module to the MFU (or vice versa), Highspeed USI packages can be used. This mode is limited to the communication between MFU and modules. Between PC/Control System and the ACU System, only the standard USI protocol is used.

5.8.1. *Form of HS USI packages*

A high-speed package can use up the complete USI bandwidth and has the following form:

Format: STX HSByte1 HSByte2 HSByte3 HSByte4 Tunnelbyte Strobe/Ack ETX

"STX" StartOfText, Package start, as 1 Byte ASCII

"HSByte1" The first of 4 hexadecimal data bytes. Is transmitted along with every package, as 2 Byte hexadecimal ASCII symbol.

"HSByte2" The second of 4 hexadecimal data bytes. Is transmitted along with every package, as 2 Byte hexadecimal ASCII symbol.

"HSByte3" The third of 4 hexadecimal data bytes. Is transmitted along with every package, as 2 Byte hexadecimal ASCII symbol.

"HSByte4" The fourth of 4 hexadecimal data bytes. Is transmitted along with every package, as 2 Byte hexadecimal ASCII symbol.

"Tunnelbyte" A data byte within a high-speed USI package. Is used to embed a standard USI within a HS USI communication. Sent as 1 byte hexadecimal ASCII symbol.

"Strobe/Ack" Contains Strobe or Acknowledgement information, which is used as handshake signals for the tunnel byte. Sent as 1 byte hexadecimal ASCII symbol. Two additional bit may be present.

"ETX" EndOfText, Package end, as 1 Byte ASCII

A high-speed package is made of the sum of 12 byte. Each byte begins with a start bit and ends with a stop bit. For 20Mbit/s transfer speed, this means that a complete package will be transmitted in 50ns * 10 Bit * 12 Byte = 6µs.

5.8.2. HSByte

The 4 hexadecimal HSByte (High Speed Byte) are transmitted with every package. At a closer look, we have in fact 8 Byte, since every hexadecimal byte is converted into 2 hexadecimal ASCII symbols for transmission. So, all of those hex ASCII symbols contains only one nibble of the original hex data byte.

Usually these bytes contain values that are needed in a fast sequence by the host or the slave for further processing. These are for example ADC values of a fast ADC module for acquisition of an actual value of electric currents or control logic of the host for IGBT controlling. By limiting to 4 Byte, each transfer can contain 32 bit of data, status and so on. 2 additional bit are available in the Strobe/ACK which can also be used as data bit. This increases the transfer capability to 34 bit.

5.8.3. The Tunnel Byte

The tunnel byte is used to ensure during high-speed mode that he MFU is still able to communicate with modules. This byte contains an ASCII byte of a regular standard USI request at a module FSP or an ASCII byte of a regular standard USI module response. This means a conventional standard USI communication is “tunneled” within a HS USI package.

Since the HS USI package is already started with STX and ended with ETX, the tunnel uses the ASCII symbols ENQ (0x05) instead of STX and EOT (0x04) at the end instead of ETX.

To ensure that a module understand all tunneled byte sent by the MFU correctly, the Strobe/Acknowledgement byte follows the tunnel byte. This is used for the handshake of the tunnel communication.

5.8.4. The Strobe/Acknowledge Byte

This byte is sent in each stream as a ASCII byte.

Once the sender has set a new value in the tunnel byte, this also sets the Strobe bit, which lets the receiver know that the tunnel byte has changed. The sender has to wait until the receiver sets the acknowledgement bit as handshake signal and thereby indicates that the new value has been fetched. Now the sender takes back the strobe bit, which leads to the receiver deleting the acknowledgement bit as well. Only then, the sender can change the tunnel byte and set the Strobe bit again.

This process slows down the communication in the tunnel in comparison to the rest of the stream, but it also ensures that both the sender and the receiver can also transmit standard USI packages during a high-speed communication.

An important advantage of this is the possibility of asynchronous communication within the HS USI package.

Since the Strobe/Acknowledgment byte is only a 1-byte hexadecimal ASCII symbol on the USI, meaning it can only represent a 4 bit Nibble, it is only possible to use these 4 bit within the modules.

The two “lowest” Bit[0] (Strobe) and [1]Acknowledge) are used for the Strobe/Acknowledgement bit. So this transmitted hex ASCII symbol can only be 0x30, 0x31 or 0x32, given that both editable bit (2) and (3) are not being used and therefore always ’0’. These bits, (2) and (3), can be used freely to add to the HSBytes.

The following Table 5-16 shows the Strobe/Acknowledgement Bytes as hexadecimal Nibble from the sender’s point of view.

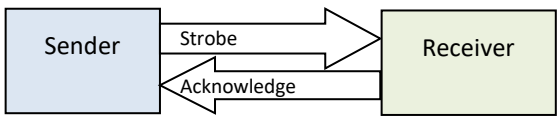


Figure 5.5: The Strobe/Acknowledge Byte from the sender’s point view

Bit	Description (sender's point of view)
3	Free to allocate
2	Free to allocate
1	If '1' → Acknowledge received, so the receiver has fetched a new data byte that was sent using a tunnel and confirms this to the sender.
0	If '1' → Send Strobe, meaning that the sender has created a new tunnel byte and lets the receiver know that this byte is ready.

Table 5-16: Details of Strobe/Acknowledge Bytes in hex from the host's point of view

The Strobe/Acknowledge Bytes behave analogue, as shown in Table 5-17.

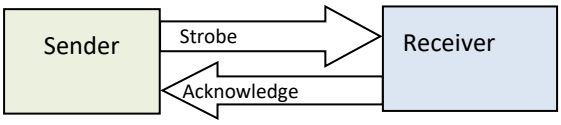


Figure 5.6: The Strobe/Acknowledge Byte from the receiver's point of view

Bit	Description (receiver's point of view)
3	Free to allocate
2	Free to allocate
1	If '1' → Send Acknowledge, meaning that the receiver has fetched the data byte that was announced by the sender using Strobe and confirms this to the sender.
0	If '1' → Strobe received, meaning that the receiver is told by the sender that a new tunnel byte is ready to be picked up.

Table 5-17: Details of Strobe/Acknowledge Bytes in hex from a module's point of view

The following Figure 5.7 shows a schematic that shows which bit are set in the handshake byte during USI high-Speed mode. Please note that each participant can sent and receive a handshake byte in parallel using full-duplex mode.

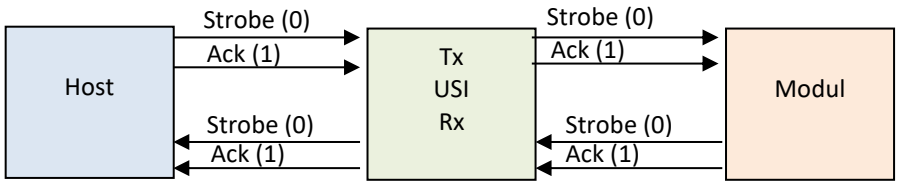


Figure 5.7: Full-duplex handshake during High-Speed USI (Bit)

5.8.4.1. Free bit within the Strobe/Acknowledge Byte

As described before, only the two lowest bit are used in the Strobe/Acknowledgement nibble. The two highest bit are free to allocate and can in principle be used as a 2 bit extension of the 4 HS bytes. Please note that the transmitted handshake ASCII byte can also take different values than 0x30, 0x31 and 0x32. If those additional bits will not be used, it is recommended to fix them on GND ('0').

5.8.5. *Example for a High-Speed- USI Package*

The data of the HSByte 1..4 can change in every stream and are therefore not specifically mentioned here.

#	Direc-tion	Sync	HSByte1	HSByte2	HSByte3	HSByte4	Tunnel Byte	Strobe/Ac knowl-edgement	Sync
			2 hexadecimal ASCII symbols each						
Within the HS USI package, the sender sends an ENQ to the receiver, which cannot respond at the same time during full-duplex mode and so the tunnel byte and Strobe/Acknowledgement are not set yet.									
1	S → R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	ENQ (0x05)	Strobe (0x31)	ETX (0x03)
	S ← R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	0x30	ETX (0x03)
The receiver confirms the reception of the tunnel request with an Acknowledgment.									
2	S → R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	ENQ (0x05)	Strobe (0x31)	ETX (0x03)
	S ← R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	Ack (0x32)	ETX (0x03)
The sender takes the Strobe back.									
3	S → R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	ENQ (0x05)	0x30	ETX (0x03)
	S ← R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	Ack (0x32)	ETX (0x03)
The sender sends the first byte of the Read PID.									
4	S → R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	R (0x52)	Strobe (0x31)	ETX (0x03)
	S ← R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	0x30	ETX (0x03)
The receiver responds with an Acknowledgment.									
5	S → R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	R (0x52)	Strobe (0x31)	ETX (0x03)
	S ← R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	Ack (0x32)	ETX (0x03)
The sender takes the Strobe back.									
6	S → R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	R (0x52)	0x30	ETX (0x03)
	S ← R	STX (0x02)	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	Ack (0x32)	ETX (0x03)
The sender sends the second byte of the Read PID.									
7	S → R	STX	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	D (0x44)	Strobe (0x31)	ETX (0x03)
	S ← R	STX	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	0x30	ETX (0x03)
The receiver responds with an Acknowledgement.									
8	S → R	STX	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	R (0x44)	Strobe (0x31)	ETX (0x03)
	S ← R	STX	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	Ack (0x32)	ETX (0x03)
And so on… The sender ends the request to the receiver using the EOT.									
..	S → R	STX	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	EOT	0x30	ETX
..	S ← R	STX	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0xYY, 0xYY	0x30	Ack	ETX
The receiver sends the requested data in the same way and the host confirms the correct reception of the respective byte using an Acknowledgement.									

Table 5-18: Example of an HS USI Package

6. MDS (Module Descriptor Structure)

To make sure that each module is distinctively identifiable by the host, the MDS is used. It is separated into chain of single Standard Descriptors. All together, they make up the MDS, which is always located at FSP000 (0_D or $0x00_H$ or 0x30, $0x30_{ASCII}$).

For the MFU as host: since further information about the purpose of each module has to be lodged in the MFU, the MDS serves only for basic information, so the MFU can decide whether the module is supported or not.

6.1. Agreement

To keep the following description simple and clearly arranged, we agree on all register values to be presented decimal or hexadecimal (0x..). Still, during transmission via USI those values are sent in ASCII. Therefore, the interface of the modules or the host has to change them from hex or dec to ASCII. So, 1_D or $0x1_H$ respectively is in ASCII one byte with the value 0x31, whereas 255_D or $0xFF_H$ respectively is two bytes with the values $0x46_A$, $0x46_A$.

If one byte hex value is finally transmitted as one or two ASCII byte, can be found in the respective descriptors in the column “Length”.

6.2. Standard Descriptors

A single Standard Descriptor of the MDS is made of a larger number of 8 bit fields in which, set by the USI specification, the properties of the modules are coded.

Now we distinguish 3 different Standard Descriptors:

Deskriptortyp	Code	Bemerkungen
Modul-Descriptor	0x1	One per module, beginning at the address 0x0000 within the MDS
FSP-Descriptor	0x2	Indicate the supported starting points
End Descriptor	0x9	Descriptor End

Table 6-1: Standard Descriptors

Top realize Plug & Play, the host has to be able to identify any module that is connected to USI. The MDS provides the short description of the physical and logical properties of a module. This MDS is prompted (command: GetMDS) by the host during enumeration.

A USI module has one specific Module Descriptor that contains the global information about it.

There has to be an FSP-descriptor for each FSP present in the module, except for FSP000 (the MDS). Each FSP-descriptor describes exactly one FSP. If a module has 25 FSPs (including FSP000), 24 FSPs are being described here.

The End Descriptor indicates the explicit end of the MDS.

6.3. Location of the MDS and its structure

The MDS is always located in FSP000.

If the host wants to read the MDS, it sends a read-request to the module with the starting point FSP000 (0x00). After that, the module sends the complete content of the MDS, which is interpreted by the host. It’s necessary that afterwards, all needed driver information is available to the host.

The module descriptor is always found at the first place. That means the first entry of the MDS always has to be 0x1, so the module descriptor is uniquely identifiable. If that value is not 0x1, usage of that module will be denied.

In the following entry, the offset until the next descriptor is found. This means, if the module descriptor has, for example, 62 byte of entries, an offset value of 62_D or $0x3E_H$ respectively is present.

Offset	Value	Description
0x0000	0x1	ID Modul-Descriptor
0x0001	0x3	MSB Offset until next Descriptor
0x0002	0xE [62]	LSB Offset until next Descriptor
...	...	
0x0010	0xy	last entry
0x0011	0x2	ID FSP-Descriptor
0x0012	0x0	MSB Offset until next Descriptor
0x0012	0x6	LSB Offset until next Descriptor
...	...	...
0x0018	0xy	last entry
0x0019	0xF	ID End-Descriptor

Table 6-2: Structure of the MDS

6.4. Interpretation cycle of the MDS for the host

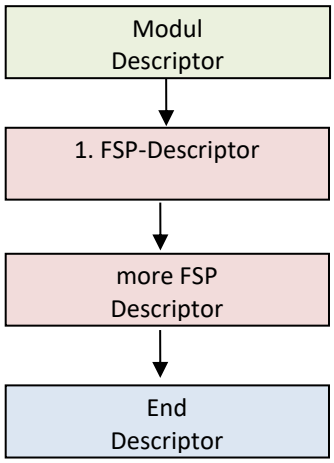


Figure 6.1: Time sequence for reading the MDS

6.5. Specification of Standard Descriptors

6.5.1. *Module Descriptor*

Each USI module contains exactly one module descriptor, which holds the basic information about that module, for example a Vendor-ID, Product- and Release-ID.

Offset	Field description	Length in Nibble	Description	Example in Nibble
0	bDescriptorType	1	Descriptor Type = Module Descriptor	0x1
1	wOffset	2	Offset until next Descriptor in Hex	0x46
3	wModuleClass	2	Module Classes-Code	0x01
5	wModuleSubClass	2	Module Sub-Classes-Code	0x00
7	sSerial	12	Serial-No of the module	0x123456789ABC
19	wVendorID	2	Vendor-ID of the manufacturer	0x01
21	lProductID	4	Product-ID of the manufacturer	0x0001
25	wHWMajorRelease	2	Major Release-No of the hardware (e.g. 2) Integer place	0x02
27	lHWMinorRelease	4	Minor Release-No of the hardware (e.g. 3) Decimal place	0x0003
31	sHWDate	6	Release date of the hardware (21.11.07) DDDDMMMMYYYY	0x230410
37	wFWMajorRelease	2	Major Release-No of the firmware (e.g. 2) Integer place	0x02
39	lFWMinorRelease	4	Minor and Sub-Release-No of the firmware (e.g. 3.1) Decimal place	0x0301
43	sFWDate	6	Release date of the firmware (21.11.07) DDDDMMMMYYYY	0x211117
49	wUSI	2	USI-Spec. vers. in BCD (e.g.: 1.0) MSB integer place LSB decimal place	0x11
51	bDeviceMaxSpeed	1	Max USI speed of the module (3 Bit)	0x1
52	wAttributes	2	Possibly more needed attributes of the modules	0x00
54	lMaxPower	4	Current usage of the module in 2mA units	0x0010
58	lNmbsOfFSPs	4	Number of supported FSPs, without FSP000 (MDS)	0x001E
62	sDescription	arbitrary	Module description as text string	

Table 6-3: Module Descriptor

There are the following Module Classes for *wVendorID*:

Vendor	Description
1	GSI
2	Danfysik LEM
3	WME
4	I-Tech
	...

Table 6-4: Manufacturer

There are the following different Classes for *wModuleClass*:

Class	Description
0	MFU
1	ADC Module
2	DAC Module
3	Digital Interlock Module
4	Analog Interlock Module
5	Interlock system
6	Load Switch – based on interlock systems
7	Climate Control System – based on interlock systems
8	TFT Display
9	Static Converter
10	Parallel Feeder Module
11	Water Interlock Module
12	ADC DAC IO Module
13	ADC Dan Physik
14	WME PSU Module
15	SER-Module
16	TS1MU1
17	Zero-Field-Control
20	ACU Testsystem
22	Analog Interlock Modul
23	DoseDeliverySystem Module (Therapie)
24	ACM Module (Ampulz Module)
25	ACU Monitoring Module
26	High Speed Data Channel (Dummy module)
27	ICM_SigmaPhi_QD
28	ICM_SigmaPhi_IngD
29	SR_Injection_Septa
30	FPGA_QuD
31	ADC 24Bit Class 1
...	
255	Special module

Table 6-5: Different Module classes

Further, sub classes ("*wModuleSubClass*") can be defined inside each class.

SubClass	Description
0	MFU LE/SE
...	
...	

Table 6-6: Module sub classes for MFU

SubClass	Description
0	keine SubKlasse
1	10 Bit
2	12 Bit
3	14 Bit
4	16 Bit
5	18 Bit
6	20 Bit
7	22 Bit
8	24 Bit
…	…

Table 6-7: Module sub classes for ADC, DAC Module

SubClass	Description
0	Slave USI Nr.1
1	Slave USI Nr.2
2	Slave USI Nr.3
3	Slave USI Nr.4

Table 6-8: Module sub classes for Static Converter Module

wDeviceMaxSpeed is transmitted as 3 bit. After a reset, a speed of 115,2kBit is chosen.

[2..0]	ASCII	Bit rate	
000	0x30	25MBit	
001	0x31	20MBit	
010	0x32	16,6MBit	
011	0x33	10MBit	
100	0x34	5MBit	
101	0x35	2MBit	
110	0x36	1MBit	
111	0x37	*115,2kBit*	*Reset*

Table 6-9: Supported Baud rates

wAttributes is interpreted as Bitmap and serves for more module specific information.

Bit position	Description
7	0-FactoryImage 1-ApplicationImage
6...2	n.u.
1	0-Module supports only normal USI mode 1-Module also supports the HighSpeed Tunnel mode
0	0-module expected NO commands via standard USI 1-module also expects commands via standard USI

Table 6-10: Attributes

6.5.2. *FSP descriptor*

The number of FSP descriptors equals the value of entry “INmbsOfFSPs”. Each descriptor contains all the information needed by the host to distinguish which of the FSP from the driver are really used by the module.

Offset	Field description	Length in Nibble	Description	Example in Nibble
0	bDescriptorType	1	Descriptor Type = FSP descriptor	0x2
1	wOffset	2	Offset until next Descriptor	0x08
3	lStartpointAttributes	8	Attribute of used FSP	0x0020A01

Table 6-11: FSP descriptor

Bit position	Description
31..28	FSP Number (ext.), not used at the moment
27..20	FSP Number (0..255)
19..16	FSP Depth (256..4095)
15..8	FSP depth (1..255)
7..1	n.u.
0	0 = FSP is readable and writeable 1 = FSP is ReadOnly

Table 6-12: FSP attributes

FSP Nr. (ext)				FSP Nr.								FSP Depth (ext)				FSP Depth								n.u.							
31			28	27							20	19			16	15							8	7						1	0

Figure 6.2: Structure of bit pattern

6.5.3. *End Descriptor*

This descriptor indicates the definite end of the MDS. It may be implemented only once and has to be placed at the end of the MDS.

Only the entries *bDescriptorType* and *wOffset* are mandatory, so the host can clearly recognize the end of MDS.

sEndOfDescriptor is optional, but it has to be set to ‘0’ in case it is not used.

Offset	Field description	Length in Nibble	Description	Example in Nibble
0	bDescriptorType	1	Descriptor Type = EndDescriptor	0x9
1	wOffset	2	Offset until the end	0x03
1	sEndOfDescriptor	3	Clear text string "EOD"	EOD

Table 6-13: End Descriptor

7. Enumeration and Plug & Play

Enumeration describes the identification and at times also the configuration of a module which is newly connected to the USI. Plugging in and out of modules is recognized by the host, which then does the enumeration.

Plugging in and out of a module cannot happen in a voltage free state.

Since the host usually realizes the enumeration of the modules after the voltage is connected, a restart of the system is reasonable after a change of configuration. Otherwise, the host can start a new scan for modules via the respective command.

7.1. General process

A USI module contains either a switch for coding which can be used to set a unique module address or an internally fixed module number. The enumeration of USI and module numbers start for every user with 1. Since up to 8 modules can be connected to a USI, the available module addresses are 0x01 … 0x8.

For HighSpeed modules with bandwidth are used up by a single USI, the module address (usually 0x01) can also be implemented permanently.

The user has to make sure the chosen USI part has no attached module with the same address yet, if a new one is supposed to be connected. If that happens, the host has to inform the user about this error.

To ensure Plug &Play functionality, the host has to check all USI interfaces upon system initialization and request the relevant module descriptors. For that, a read request as described in chapter 5.7.1.1.4. “Host wants to read data from a module” has to be sent.

The request looks as follows (with the MFU as host):

STX - RD - Module address – FSP000 - ETX

0x02 - 0x52 - 0x44 - 0x3y - 0x30 - 0x30 - 0x03

y = Module No 1...8

Since multiple modules can be connected to a USI, the unique module address “y” has to be included. This address is increased by 1 each cycle and starts over at zero once the highest one is reached.

A module has to respond with its descriptor upon request.

First the introducing header is sent:

STX - MA - FSP - MDS... - PP - ETX

0x02 - 0x3y - 0x30 - 0x30 ..... - PP - ETX

..... = Contents of the MDS.

The host evaluates the MDS and analyses if a driver is present. If that is true, the module is accepted.

All USI interfaces are checked in sequence for modules.