System Specifications

3

Standardised Interfaces

6

Application Programmer's Interface

1

Summary

This document defines the standardised Application Programmer's Interfaces (API's) for standardised bus devices.

Version 01.01.01 is a KNX Approved Standard.

This document is part of the KNX Specifications v2.1.

Document updates

Version	Date	Modifications
0.3	2001.05.22	Inclusion from comments from RfV.
1.0	2002.02.28	Preparation of the Approved Standard. Inclusion of mask 0021, added specification for functions U_EE_WriteHI and U_Char_Out.
1.1	2008.10.27	• **AN015 "API CU"** integration started
1.1	2009.06.24	• Preparation for inclusion in the KNX Specifications v2.0.
01.01.01	2013.10.28	Editorial updates for the publication of KNX Specifications 2.1.

References

[01] Chapter 3/6/3 "External Message Interface"
[02] Part 3/7 "Interworking"
[03] Part 9/4 "BCUs and BIMs"

Filename: 03_06_01 Application Programmers Interface v01.01.01 AS.docx
Version: 01.01.01
Status: Approved Standard
Savedate: 2013.10.28
Number of pages: 53

Contents

1 Characteristics of BCU RAM and EEPROM Variables 6
1.1 API RAM Variables and RAM Space 6
1.1.1 Registers B to N 6
1.1.2 Stack Space available for the Internal Application 6
1.1.3 Internal Application (i.e. "User") RAM 6
1.2 API EEPROM Variables and EEPROM Space 6
1.2.1 Group Object Table pointer "CommsTabPtr" and "CommsTabPtr2" 6
1.2.2 User Initialisation Routine pointer "UsrInitPtr" and "UsrIntiPtr2" 6
1.2.3 User Program Pointer "UsrPrgPtr" and "UsrPrgPtr2" 6
1.2.4 User Save Routine Pointer "UsrSavPtr" and "UsrSavPtr2" 7
1.2.5 Internal User Application (i.e. "user") EEPROM space 7
1.3 API Tables 7
1.3.1 OR_TAB 7
1.3.2 AND_TAB 7
1.4 System Timer 8
1.5 User Timer 8

2 API Function Reference 10
2.1 Group Object Manipulation Functions 10
2.1.1 U_flagsGet Reading RAM Flags 10
2.1.2 U_flagsSet Writing RAM flags 10
2.1.3 U_testObj Testing RAM flags 11
2.1.4 U_transRequest Setting Transmit Request 11
2.1.5 Standard Callback 12
2.2 EEPROM Manipulation Support Functions 13
2.2.1 Eewrite Writing to EEPROM 13
2.2.2 EEsetChecksum Updating the checksum 13
2.2.3 U_EE_WriteBlock Write Block to EEPROM 14
2.2.4 U_EE_WriteHI 14
2.3 Application Support Functions 14
2.3.1 U_debounce Debouncing 14
2.3.2 U_delMsgs Ignoring messages to the user 16
2.3.3 U_readAD Doing AD conversion 16
2.3.4 U_map Characterisation function 16
2.3.5 U_GetAccess Get actual Access Level 18
2.3.6 U_SetPollingRsp Set Polling Response 18
2.4 PEI Support Functions 18
2.4.1 U_ioAST Binary Port Access 18
2.4.2 S_AstShift / S_LastShift Data-Block Exchange via Serial Synchronous PEI (PEI-Type 14 only) 20
2.4.3 U_SerialShift Octet exchange via Serial PEI 20
2.4.4 U_Char_Out SPI/SCI Interface 21
2.5 Timer Functions 21
2.5.1 TM_Load Starting the timer 21
2.5.2 TM_GetFlg Reading the Timer Status 22
2.5.3 U_SetTM Setting a User Timer 22
2.5.4 U_GetTM Reading the User Timer Status 24
2.5.5 U_Delay 25
2.5.6 BCU 2 Timer System 26

2.5.6.1 U_TS_Set Set a BCU 2 Timer 26
2.5.6.2 U_TS_Del Delete BCU 2 Timer 27
2.5.6.3 U_TS_Seti Set Timer Interrupt Event 27
2.6 Message Handling Functions 28
2.6.1 AllocBuf Buffer Allocation 28
2.6.2 FreeBuf Buffer Release 28
2.6.3 PopBuf Message request 29
2.6.4 U_MS_Post Post Messages 30
2.6.5 U_MS_Switch Message redirection 30
2.7 Arithmetic Functions 31
2.7.1 multDE_FG Unsigned Integer Multiplication 31
2.7.2 divDE_BC Unsigned Integer Division 32
2.7.3 FP_Flt2Int Float to Integer 32
2.7.4 FP_Int2Flt Integer to Float 33
2.8 Miscellaneous Functions 33
2.8.1 shlA*n* Accu shift left n 33
2.8.2 shrA*n* Accu shift right n 34
2.8.3 rolAn Accu rotate Left 34
2.8.4 U_SetBit Bit write 35
2.8.5 U_GetBit 35
2.9 Loadable PEI-Support 36
2.9.1 U_FT12_Reset Reset FT 1.2 Protocol 36
2.9.2 U_FT12_GetStatus Get FT 1.2 Protocol Status 36
2.9.3 U_SCI_Init Initialize Serial Communication Interface 37
2.9.4 U_SPI_Init Initialize Serial Peripal Interface 37

3 API for BIM M112 38
3.1 Overview 38
3.2 Description of API functions 39
3.2.1 API Function: _AL_isSapLinked 39
3.2.2 API Function: _check 39
3.2.3 API Function: _copyMem 40
3.2.4 API Function: _getAPIVersion 40
3.2.5 API Function: _installCallback 41
3.2.6 API Function: _modulMode 41
3.2.7 API Function: _MSG_allocBuffer 42
3.2.8 API Function: _MSG_freeMessage 42
3.2.9 API Function: _MSG_getMessage 43
3.2.10 API Function: _MSG_postMessage 43
3.2.11 API Function: _MSG_staticRedirection 44
3.2.12 API Function: _PEI_addEvent 44
3.2.13 API Function: _PEI_analogueIn 44
3.2.14 API Function: _PEI_binaryInOut 45
3.2.15 API Function: _PEI_getActualPeiType 45
3.2.16 API Function: _PEI_getEvent 45
3.2.17 API Function: _PEI_getFreeBufferSpace 46
3.2.18 API Function: _PEI_init 46
3.2.19 API Function: _PEI_initBuffer 47
3.2.20 API Function: _PEI_initEventQueue 47
3.2.21 API Function: _PEI_rcvBufferRead 48

3.2.22 API Function: _PEI_rcvBufferWrite 48
3.2.23 API Function: _PEI_sendBufferRead 49
3.2.24 API Function: _PEI_sendBufferWrite 49
3.2.25 API Function: _PEI_sendByteSerial 49
3.2.26 API Function: _PEI_setTimer 50
3.2.27 API Function: _runRomModule 50
3.2.28 API Function: _setParam 51
3.2.29 API Function: _taskSwitch 52
3.2.30 API Function: _TM_getSystemTime 52
3.2.31 API Function: _TM_getUserTicks 53

1 Characteristics of BCU RAM and EEPROM Variables

1.1 API RAM Variables and RAM Space

1.1.1 Registers B to N

The RAM area of 50h to 5Ch can be used as register memory space or as temporary RAM storage by the application programmer.

Attention: ***"registers" B to N may also be used by ROM functions or for parameter passing or as temporary variables. See description of API functions!***

Registers can be used to store parameters for functions to be called.

Attention: Before the next complete execution of the user program all registers are cleared by the system firmware.

1.1.2 Stack Space available for the Internal Application

Internal applications can use the microprocessors' stack for own purposes. Applications shall take care not to use more than the maximimum indicated stack space for the implementation. If an application uses more stack space, this can result in severe conflicts with the stack needs by the BCU firmware. The stack space is for the various implementations defined in [03].

1.1.3 Internal Application (i.e. "User") RAM

The internal application gets next to the processor's registers also processor RAM memory available. The proper size and organization of this RAM can be found in in [03].

1.2 API EEPROM Variables and EEPROM Space

1.2.1 Group Object Table pointer "CommsTabPtr" and "CommsTabPtr2"

CommsTabPtr is a one octet EEPROM variable. CommsTabPtr contains the octet offset to the location of the Group Object Table in EEPROM, if it is not zero. If CommsTabPtr is not zero then the Group Object Table starts at address 100h+[CommsTabPtr].

If CommsTabPtr is zero then the EEPROM variable CommsTabPtr2 exists which contains a Big Endian absolute pointer to the Group Object Table location.

1.2.2 User Initialisation Routine pointer "UsrInitPtr" and "UsrIntiPtr2"

If CommsTabPtr is not zero then UsrInitPtr exists and contains the octet offset to the start address of the user initialisation routine in EEPROM. The initialisation routine starts at 100h+[UsrInitPtr].

If CommsTabPtr is zero then the EEPROM variable UsrInitPtr2 exists which contains a Big Endian absolute pointer to the start address of the user initialisation routine.

The initialisation routine is called once at the startup time of the internal user application. It must be written as a subroutine, i.e. it must be terminated by "rts".

1.2.3 User Program Pointer "UsrPrgPtr" and "UsrPrgPtr2"

If CommsTabPtr is not zero then UsrPrgPtr exists and contains the octet offset to the start address of the internal user application program in EEPROM. Then the internal user application program starts at 100h+[UsrPrgPtr].

If CommsTabPtr is zero then the EEPROM variable UsrPrgPtr2 exists which contains a Big Endian absolute pointer to the start address of the internal user application program.

This routine is called periodically if the BCU is in normal operation mode and the PEI type expected is equal to the PEI adaptor's PEI type measured by the A/D converter of the BCU.

This routine must be written as a subroutine, i.e. it must be terminated by "rts".

1.2.4 User Save Routine Pointer "UsrSavPtr" and "UsrSavPtr2"

If CommsTabPtr is not zero then UsrSavPtr exists and contains the octet offset to the start address of the user save routine in EEPROM. The user save routine starts at 100h+[UsrSavPtr].

If CommsTabPtr is zero then the EEPROM variable UsrSavPtr2 exists which contains a Big Endian absolute pointer to the start address of the user save routine.

This routine is called if a bus power loss is detected at a bus-powered device and the internal user application program is not inactivated.

After calling this routine the BCU is reset. Nonetheless this routine must be written as a subroutine, i.e. it must be terminated by "rts".

1.2.5 Internal User Application (i.e. "user") EEPROM space

This is the EEPROM space usable by the internal user application program and the optional application programmer-written PEI driver for PEI 10.

Size: see [03].

1.3 API Tables

1.3.1 OR_TAB

Symbol: **OR_TAB**

Address: Mask 0010h: 0020h
Mask 0011h: 0020h
Mask 0020h: 0020h
Mask 1013h: 0020h

OR_TAB

Addr. (hex)	Value (binary)
20	0000 0001
21	0000 0010
22	0000 0100
23	0000 1000
24	0001 0000
25	0010 0000
26	0100 0000
27	1000 0000

1.3.2 AND_TAB

Symbol: **AND_TAB**

Address: Mask 0010h: 0028h
Mask 0011h: 0028h
Mask 0020h: 0028h
Mask 1013h: 0028h

AND_TAB

Addr.(hex)	Value (binary)
28	1111 1110
29	1111 1101

2A	1111 1011
2B	1111 0111
2C	1110 1111
2D	1101 1111
2E	1011 1111
2F	0111 1111

Both tables are helpfull for masking-operation. If you need a value of these tables, you need only a one-octet-offset. This is helpfull for loops.

Example

```
        ldx     #AND_TAB        ;ptr to AND_TAB
loop    lda
        and     ,X              ;MASK-operation
        sta
:
        incx
        bra     loop
```

1.4 System Timer

There are 4 software-timers: 0 to3.

The timers 0 and 1 are reserved for the system software.

The timers 2 and 3 are available to the user.

The timer 2 is also used by the debounce-function. Do not use it a second time, if debouncing is used.

A timer can be used in one of two operation modes.

In operation mode, 0 a timer is initialized with a run-time. If this time is expired, then this is flagged. The timer may be restarted during operation.

In operation mode 1, the timer calculates the time since the last call to the timer function. The user must take care to avoid range overflows.

A timer has a resolution of 8 bits. It can be operated in five different time-ranges. E.g. a timer-value of 5 in time-range 2 means about 40 ms.

Range	Time-Unit
1	± 0,55 ms
2	± 8,0 ms
3	± 130 ms
4	± 2,1 s
5	± 33 s

NOTE A not-initialized timer has the status of an expired timer.

1.5 User Timer

There is another set of support routines that can be used to define additional User timers.

Each pair of User-Timers can have a different time base. But each timer must be updated at least one time per timer-tick in its own time base.

A description-block in EEPROM is used to describe the User timers. Each User timer function needs a pointer to this description block.

The structure of the EEPROM description block is:

Offset in block	Length (octets)	Meaning
0	1	Pointer to RAM-data
1	1	Time base [1] 0 and 1
2	1	Time base 2 and 3
…		...

Time bases:

Number	Minimum Time Resolution	
0	ca. 130	ms
1	ca. 260	ms
2	ca. 520	ms
3	ca. 1,0	s
4	ca. 2,1	s
5	ca. 4,2	s
6	ca. 8,4	s
7	ca. 17	s
8	ca. 34	s
9	ca. 1,1	min
10	ca. 2,2	min
11	ca. 4,5	min
12	ca. 9,0	min
13	ca. 18	min
14	ca. 35	min
15	ca. 1,2	h

For each user timer, one octet must be reserved in RAM. All of these octets must by allocated in one block. The first octet in this block corresponds with the first user timer, the second octet corresponds with the second user timer and so on. The bit 7 in each octet is reserved, the bits 0-6 is used for the timer value (0 to 127). This value will be decremented only if you use the (update function) U_GetTM. You can set the user timer if you use the function U_SetTM, or you can load the corresponding RAM directly with the value. The bit7 must be 0.

[1] In the mask-version 1.0 only the low nibble is evaluated. It serves as the time base for both timers.

2 API Function Reference

The API function reference lists all the API functions that support the task of the programmer of the internal user application. The interface to the functions is a pure assembler interface. If you want to interface at e.g. C language level then you must write your own interface adaptations (header files).

The estimated value given for "Watchdog-time" is an indication of how much watchdog-time this routine needs for processing. I.e. processing-time since latest triggering of watchdog (inside that routine).

This value is necessary to estimate when the watchdog must be triggered.

2.1 Group Object Manipulation Functions

2.1.1 U_flagsGet Reading RAM Flags

Symbol: **U_flagsGet**

Description: gets the RAM-flags of the specified Group Object.

Inputs: A = Group Object number

	Call Address	Outputs	Changed Registers
Mask 0010h	0C8Ch	See below	A, X, RegC, RegJ
Mask 0011h	0C9Dh	See below	A, X, RegC, RegJ
Mask 0012h	0C9Dh	See below	A, X, RegC, RegJ
Mask 0020h	505Ah	See below	A, X, RegC, RegJ
Mask 1013h	3D14h	See below	A, X, RegJ RegA = RAM-flags

Watchdog-time: (to be completed) µs

Outputs: RegB

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Meaning	undefined Mask 1013h: zero				update flag	data request flag	transmission status	

2.1.2 U_flagsSet Writing RAM flags

Symbol: **U_flagsSet**

Description: sets the RAM-flags of the specified Group Object.

Inputs: A = Group Object number

RegB = RAM-flags

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Meaning	undefined				update flag	data request flag	transmission status	

	Call Address	Outputs	Changed Registers
Mask 0010h	0C94h	None	A, X, RegB, RegC, RegJ
Mask 0011h	0CB3h	None	A, X, RegB, RegC, RegJ
Mask 0012h	0CB3h	None	A, X, RegB, RegC, RegJ
Mask 0020h	505Dh	None	A, X, RegB, RegC, RegJ
Mask 1013h	3D11h	None	A, X, RegB, RegJ

Watchdog-time: (to be completed) µs

2.1.3 U_testObj Testing RAM flags

Symbol: **U_testObj**

Description: fetches the RAM-flags from the specified object and resets the Update-flag.

Inputs: A = Group Object number

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	0CA5h		A, X, RegB, RegC, RegJ
Mask 0012h	0CA5h		A, X, RegB, RegC, RegJ
Mask 0020h	507Bh		A, X, RegB, RegC, RegJ
Mask 1013h	3D17h		A, X, RegJ

Watchdog-time: (to be completed) µs

Outputs: RegC = RAM-flags (right adjusted)

Zero-flag = **NOT** Update-flag

2.1.4 U_transRequest Setting Transmit Request

Symbol: **U_transRequest**

Description: sets a transmit-request in the specified Group Object. If a telegram is being currently transmitted for this object then the transmit-request is not set.

Inputs: A = Group Object number

	Call Address	Outputs	Changed Registers
Mask 0010h	0D91h		A, X, RegB, RegC, RegJ
Mask 0011h	0DB9h		A, X, RegB, RegC, RegJ
Mask 0012h	0DB9h		A, X, RegB, RegC, RegJ
Mask 0020h	507Eh		A, X, RegB, RegC, RegJ
Mask 1013h	3D0Eh		A, X, RegJ

Watchdog-time: (to be completed) µs

Outputs: Carry

0 = OK

1 = transmit-request not set

Note

- Mask 1013h: Due to a slower bus-transmission speed its strongly recommended to evaluate the return value of this function. Changing the object value while transmission-request is set may cause unpredictable telegrams on the bus.

2.1.5 Standard Callback

Symbol: **AL_SAPcallback**

Description: Complete handling of Group Objects and Application Interface Objects

Inputs: X Index to message

Carry 0 : Polling mode: call one time in a cycle

1 : Message mode: call if a message for an object is received either a Group Object or a Interface Object (SAP = FFh)

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	5081h	See below	A, X, RegB -RegN
Mask 0021h	5081h	See below	A, X, RegB -RegN
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

Outputs: X Index to message

A state
- 0 Ok (Return messagebuffer to system)
- 1 break and free buffer
- 2 hold buffer and break
- 3 send message and break

Example

```
KNXMSG_ASAP   EQU  24                ; the offset of the ASAP in the message buffer
AppCallBack
              bcc  Nomsg
* Message received
              lda  KNXMSG_ASAP,x     Load SAP number
              cmp  #SPECIAL_SAP
              bne  STDcallback
*handle special SAP
              ...
              ...

STDcallback
              sec                    ;mark message
Nomsg         jmp  AL_SAPcallback    ;Use standard callback
```

2.2 EEPROM Manipulation Support Functions

2.2.1 Eewrite Writing to EEPROM

Symbol: **EEwrite**

Description: writes an octet to the specified location in memory.
The write operation takes up to 20ms.
Attention: Update checksum if necessary!

Inputs: A = value to write
X = offset in EEPROM

	Call Address	Outputs	Changed Registers
Mask 0010h	0C2Dh	None	RegB, RegC, RegH
Mask 0011h	0C38h	None	RegB, RegC, RegH
Mask 0012h	0C38h	None	RegB, RegC, RegH
Mask 0020h	503Fh	None	RegB, RegC, RegH
Mask 1013h	3D08h	None	RegB, RegC, RegH

Watchdog-time: internally set by function

2.2.2 EEsetChecksum Updating the checksum

Symbol: **EEsetChecksum**

Description: updates the checksum octet of the EEPROM. This function must be called if any octet inside the check range is modified by the user.

Inputs: None

	Call Address	Outputs	Changed Registers
Mask 0010h	0C5Dh	None	A, X, RegB, RegC, RegH
Mask 0011h	0C68h	None	A, X, RegB, RegC, RegH
Mask 0012h	0C68h	None	A, X, RegB, RegC, RegH
Mask 0020h	503Ch	None	A, X, RegB, RegC, RegH
Mask 1013h	3D05h	None	A, X, RegB, RegC, RegH

Watchdog-time: internally set by function

2.2.3 U_EE_WriteBlock Write Block to EEPROM

Symbol: **U_EE_WriteBlock**

Description : Write a block of 4 octets to the EEPROM

Inputs RegH:RegI Memory Address
RegK,RegL,RegM,RegN Memory Block to write

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	5099h	None	A, X, RegB, RegC, RegJ
Mask 0021h	5099h	None	A, X, RegB, RegC, RegJ
Mask 1013h	-	-	-

Watchdog-time: internally set by function µs

2.2.4 U_EE_WriteHI

Symbol: **U_EE_WriteHI**

Description: Write one octet to the EEPROM.

Inputs RegH:RegI Memory Address
A Octet to write

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	-	-	-
Mask 0021h	50B2h	None	A, X, RegB, RegG, RegJ
Mask 1013h	-	-	-

Watchdog-time: internally set by function µs

2.3 Application Support Functions

2.3.1 U_debounce Debouncing

Symbol: **U_debounce**

Description: debounces a complete octet. As long as the debounce-time is not yet expired or the value is changing the latest debounced value is returned.

- Mask 00xx: UserRAM = last input value
  UserRAM+1 = debounced value

  Note: It is not necessary to initialize the software timer 2.

- Mask 10xx: UserRAM1 = last input value
  UserRAM1+1 = debounced value

  Note: It is not necessary to initialize the system-timer #2.

Inputs: A = value (octet) to debounce
X = debounce-time in 0.5 ms-units

	Call Address	Outputs	Changed Registers
Mask 0010h	0C64h	See below	X, RegB, RegC, RegD, RegE, RegF, RegG
Mask 0011h	0C75h	See below	X, RegB, RegC, RegD, RegE, RegF, RegG
Mask 0012h	0C75h	See below	X, RegB, RegC, RegD, RegE, RegF, RegG
Mask 0020h	C5051h	See below	X, RegB, RegC, RegD, RegE, RegF, RegG
Mask 1013h	3D26h	See below	X, RegB, RegC, RegD, RegE, RegF, RegG

Watchdog-time: (to be completed) µs

Outputs: A = debounced value (octet)

Effects:

- Mask 00xx uses software-timer 2
  changed RAM-locations: UserRAM, UserRAM + 1
- Mask 10xx uses system-timer #2
  changed RAM-locations: UserRAM1, UserRAM1 + 1

Symbol: **U_deb10**

Description: This function is the same as U_debounce except that a fixed debounce-time of 10 ms is used. Therefore the X register has not to be loaded.

Inputs: A = value (octet) to debounce

	Call Address	Outputs	Changed Registers
Mask 0010h	-		
Mask 0011h	0C73h	See U_debounce	See U_debounce
Mask 0012h	0C73h	See U_debounce	See U_debounce
Mask 0020h	504Bh	See U_debounce	See U_debounce
Mask 1013h	3D23h	See U_debounce	See U_debounce

Symbol: **U_deb30**

Description: This function is the same as U_debounce except that a fixed debounce-time of 30 ms is used.

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	0C6Fh	See U_debounce	See U_debounce
Mask 0012h	0C6Fh	See U_debounce	See U_debounce
Mask 0020h	504Eh	See U_debounce	See U_debounce
Mask 1013h	3D20h	See U_debounce	See U_debounce

Watchdog-time: (to be completed) µs

2.3.2 U_delMsgs Ignoring messages to the user

Symbol: **U_delMsgs**[2]

Description: removes any messages addressed to the user program. Call this function in the USER main routine, if you do not expect any messages. Otherwise the buffer resources in the BCU may become exhausted due to external faults.

Inputs: None

	Call Address	Outputs	Changed Registers
Mask 0010h	0C82h	None	A, X, RegB
Mask 0011h	0C93h	None	A, X, RegB
Mask 0012h	0C93h	None	A, X, RegB
Mask 0020h	5057h	None	A, X, RegB
Mask 1013h	3D89h	None	A, X, RegB

Watchdog-time: (to be completed) µs

2.3.3 U_readAD Doing AD conversion

Symbol: **U_readAD**

Description: starts A/D conversion for the specified port (not necessarily a PEI I/O port!) and reads the result. This operation is repeated the specified number of times. The sum of the values of all read operations is returned.

Note: This function is valid for A/D conversion of any analog port values in the bus access module, no matter whether the port line is connected to a PEI input line.

Inputs: A = channel number of analog port
X = number of read operations

	Call Address	Outputs	Changed Registers
Mask 0010h	0D35h	See below	A, X
Mask 0011h	0D54h	See below	A, X
Mask 0012h	0D54h	See below	A, X
Mask 0020h	506Ch	See below	A, X
Mask 1013h	3D38h	See below A = value of last AD-conversion	A, X

Watchdog-time: (to be completed) µs

Outputs: RegD:RegE = sum of read values. Precision is implementation-specific.

Note:

- Mask 1013h: If only one AD-conversion was requested the contents of the Accu is identical with the contents of RegD:RegE

2.3.4 U_map Characterisation function

Symbol: **U_map**

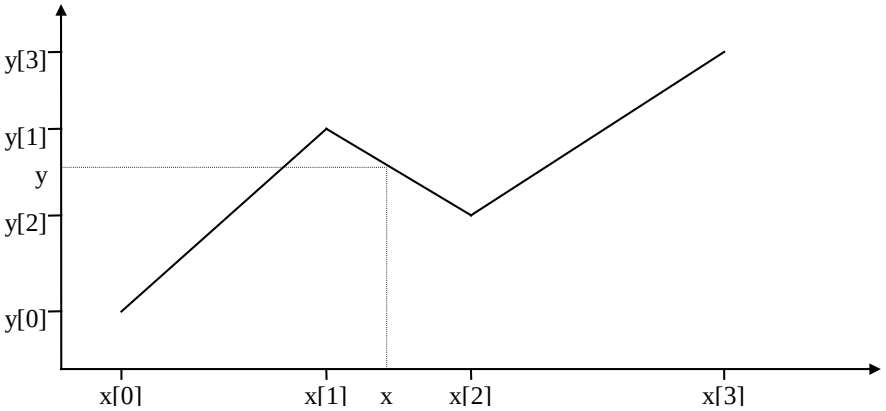
[2] The same effect as calling this function can be achieved, for the mask-version 0011h and upward, and the masks 0021h and 0022h, if bit 7 in the EEPROM-parameter RouteCnt is set.

Description: maps the input value by the use of a conversion table. Input value and result are 16-bit signed integer numbers and cover all 4 quadrants. For the conversion a table of x-y-value pairs is used. The input value must be inside the x-value range. For values in between two x-y-value pairs linear interpolation is used.

The interpolation-formula used is:

$$Y = [(X-X1)*(Y2-Y1)]/(X2-X1) + Y1$$

All intermediate results must not exceed the signed integer value range.



The conversion table is assumed to be in EEPROM and has the following format: (x-values in ascending order, LSOctet is the high octet).

Length (x-y-pair count) (1 octet)	
X0	Y0
X1	Y1
...	...
low word	high word

Inputs: RegB:RegC = value to be mapped (signed integer)
X = pointer to conversion table (EEPROM-offset)

	Call Address	Outputs	Changed Registers
Mask 0010h	0C9Bh	See below	A, X, RegD, RegE, RegF, RegG, RegH, RegI
Mask 0011h	0CBAh	See below	A, X, RegD, RegE, RegF, RegG, RegH, RegI
Mask 0012h	0CBAh	See below	A, X, RegD, RegE, RegF, RegG, RegH, RegI
Mask 0020h	5069h	See below	A, X, RegD, RegE, RegF, RegG, RegH, RegI
Mask 1013h	3D86h	See below	A, X, RegD, RegE, RegF, RegG, RegH, RegI

Watchdog-time: (to be completed) µs

Outputs: RegB:RegC = result (signed integer)
Carry = 0 OK
1 conversion error, result not valid

Note:

- Masks 0010h, 0011h: These mask versions only operate in the first quadrant!

2.3.5 U_GetAccess Get actual Access Level

Symbol: **U_GetAccess**

Description : gets the actual accesslevel of the system

Inputs None

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	5096h	A = Accesslevel	None
Mask 0021h	5096h	A = Accesslevel	None
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

2.3.6 U_SetPollingRsp Set Polling Response

Symbol: **U_SetPollingRsp**

Description : Set the value to response at poll request.

Inputs A Value to response

Reserved Values

FFh	No user running / not set from user
F0h-Fdh	Reserved
FEh	Fill character no slave responding
00h	Set at Userstartup

Bit 7 1: Systemstate
0: Userstate

Bit 6 Error

Bit 5 Alarm

Bit 4 Reseverd

Bit 0-3: Usercode

For more informations please refer to [02].

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	50AEh	None	None
Mask 0021h	50AEh	None	None
Mask 1013h	-	-	-

2.4 PEI Support Functions

2.4.1 U_ioAST Binary Port Access

Symbol: **U_ioAST**

Description: Digital ports can be accessed by the function U_ioAST that allows reading and/or writing digital I/O pins 1 to 4. Reading or writing can be selected for each PEI I/O line via I/O flags. Accumulator and RegB serve as parameters for U_ioAST.

Via I/O-flags reading or writing can be selected for each bit/pin. For both input and output the bits are mapped to the PEI-port as follows:

I/O-Bit #	Input # or Output #	PEI-Pin
0	1	3
1	2	2
2	3	4
3	4	7

Encoding of read / write bits: 0 means off, i.e. 0 V

1 means on, i.e. 5 VDC

Inputs: Accumulator: for I/O flags and values

bit #	7	6	5	4	3	2	1	0
meaning	I/0-flags 0 = read 1 = write				write bit values (valid only if write selected for that bit)			
I/O bit #	3	2	1	0	3	2	1	0

	Call Address	Outputs	Changed Registers
Mask 0010h	0DA7h	See below	A, X, RegB, RegC, RegD
Mask 0011h	0DCFh	See below	A, X, RegB, RegC, RegD
Mask 0012h	0DCFh	See below	A, X, RegB, RegC, RegD
Mask 0020h	5066h	See below	A, X, RegB, RegC, RegD
Mask 1013h	3D35h	See below	A, X, RegB, RegC

Watchdog-time: (to be completed) µs

Outputs: RegB: for bit values to be read

bit #	7	6	5	4	3	2	1	0
meaning	bit values of A before read/write				read bit values (valid only if read selected for that bit)			
I/O bit #	3	2	1	0	3	2	1	0

Note: Digital Input Value Sampling

Digital input value sampling must be done in the internal user application program with the help of several calls of U_ioAST. This means, that the sampling rate per time unit and debouncing is left to the user application programmer.

2.4.2 S_AstShift / S_LastShift Data-Block Exchange via Serial Synchronous PEI (PEI-Type 14 only)

Symbol: **S_AstShift / S_LastShift**

Description: The specified data block is exchanged via the serial PEI-interface. The synchronous protocol, that runs at PEI type 14, and the data block format are described in [01] of these KNX Specifications.

The function "S_AstShift" uses a ca. 130 ms timeout and the function "S_LAstShift" uses a ca. 1 s timeout. I.e. the data block exchange must be completed within these timeout times.

The data format is shown below:

Message length in octets (1 octet)	data

Inputs: X = pointer to data block

	Call Address		Outputs	Changed Registers
	S_AstShift	S_LAstShift		
Mask 0010h	1103h	1101h	See below	A, RegB, RegC, RegD, RegE, RegF, RegG, RegH, RegI
Mask 0011h	1117h	1115h	See below	A, RegB, RegC, RegD, RegE, RegF, RegG, RegH, RegI
Mask 0012h	1117h	1115h	See below	A, RegB, RegC, RegD, RegE, RegF, RegG, RegH, RegI
Mask 0020h	5042h	5045h	See below	A, RegB, RegC, RegD, RegE, RegF, RegG, RegH, RegI
Mask 1013h	3D3Eh	3D3Bh	See below	A, RegB, RegC, RegD, RegE, RegF, RegG, RegH, RegI

Outputs: X = pointer to data block (contains response)
Carry: 0 = communication Ok
1 = communication failed

Note: When using the serial interface, it is very important to change the "own" handshake signal not before the end of the stop bit. Otherwise, the transmission will fail.

2.4.3 U_SerialShift Octet exchange via Serial PEI

Symbol: **U_SerialShift**

Description : The specified octet is exchanged via the serial PEI-interface. The function U_SerialShift" uses a ca. 130 ms-timeout. I.e. the octet-exchange must be completed within the timeout-time.

Inputs: A = Octet to be transferred

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	0C90h	See below	A, RegB, RegC, RegD, RegE, RegF, RegG,RegI
Mask 0020h	5048h	See below	A, RegB, RegC, RegD, RegE, RegF, RegG,RegI
Mask 1013h	3D41h	See below	A, RegB, RegC, RegD, RegE, RegF, RegG,RegH

Watchdog-time: (to be completed) µs

Outputs: A = received octet
Carry = 0 : communication O.K.
1 : communication failed

Note: **Important Remarks to the functions S_AstShift, S_LastShift and U_SerialShift**

1. These functions may only be used in combination with PEI-type 14.
2. Long wait times may cause serious problems on the bus (busy-acknowledges).

2.4.4 U_Char_Out SPI/SCI Interface

Symbol: **U_Char_Out**

Description: The specified octet is exchanged via the serial SPI/SCI PEI interface (depends on PEI-type).

- no timeout
- no flow control activ (RTS/CTS handling)
- use %SyncRate for SPI/U_SCI_Init for SCI baudrate (if not, 9600 baud is preselected on the SCI by the system initialisation)
- 8 and 9 databits are possible

Inputs: A = octet to be transferred

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	508Ah	-	X (if 9 databits are used)
Mask 0021h	508Ah	-	X (if 9 databits are used)
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

2.5 Timer Functions

2.5.1 TM_Load Starting the timer

Symbol: **TM_Load**

Description: The specified timer is initialized with operation mode and time-range.
In addition, in operation mode 0 the run-time is set and the timer is started.

Inputs: A = (see below)

X = run-time (operation mode 0 only)

Input in A:

- Mask 0010h, 0011h, 0012h, 0020h:

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Meaning	must be 0		timer #		operation mode	timer range		

- Mask 1013h:

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Meaning	timer #				operation mode	timer range		

	Call Address	Outputs	Changed Registers
Mask 0010h	0E0Ch	None	A, X, RegB, RegC, RegD, RegE, RegF
Mask 0011h	0E2Bh	None	A, X, RegB, RegC, RegD, RegE, RegF
Mask 0012h	0E2Bh	None	A, X, RegB, RegC, RegD, RegE, RegF
Mask 0020h	5039h	None	A, X, RegB, RegC, RegD, RegE, RegF
Mask 1013h	3D1Ah	None	A, X, RegB, RegC, RegD, RegE, RegF

Watchdog-time: (to be completed) µs

2.5.2 TM_GetFlg Reading the Timer Status

Symbol: **TM_GetFlg**

Description: The timer-status is returned.
In operation mode 0, it returns if the run-time is expired.
In operation mode 1, the time since the last call to this function is returned.

Inputs: A = timer number

	Call Address	Outputs	Changed Registers
Mask 0010h	0E2Ah		A, X, RegB, RegC, RegD, RegE
Mask 0011h	0E49h		A, X, RegB, RegC, RegD, RegE
Mask 0012h	0E49h		A, X, RegB, RegC, RegD, RegE
Mask 0020h	5036h		A, X, RegB, RegC, RegD, RegE
Mask 1013h	3D1Dh		A, X, RegB, RegC, RegD, RegE

Watchdog-time: (to be completed) µs

Outputs:

Operation Mode 0	Operation Mode 1
Carry = 0 time not yet expired 1 time expired	A = time since the last call

2.5.3 U_SetTM Setting a User Timer

Symbol: **U_SetTM**

Description: loads a User-Timer.

Inputs:

- Mask 0010h, 0011h, 0012h, 0020h:

  A = user-timer number
  X = pointer to EEPROM description block
  RegE = time to be set

- Mask 1013h:

  A = See table below

Bit #	7	6	5	4	3	2	1	0
Meaning	Location of timer-RAM-block 0 = UserRAM1 1 = UserRAM2	User-Timer number						

  X = pointer to EEPROM-description-block

  RegE = (see below)

Bit #	7	6	5	4	3	2	1	0
Meaning	not evaluated	time to be set						

	Call Address	Outputs	Changed Registers
Mask 0010h	0D8Ah	None	A, X, RegB, RegC, RegD
Mask 0011h	0DB3h	None	A, X, RegB, RegC, RegD
Mask 0012h	0DB3h	None	A, X, RegB, RegC, RegD
Mask 0020h	506Fh	None	A, X, RegB, RegC, RegD
Mask 1013h	3D32h	None	A, X RegE if Bit #7 is set

Watchdog-time: (to be completed) µs

Symbol: **U_SetTMx**

Description: This function is the same as U_SetTM except that the pointer to the pointer to EEPROM description block is fetched from the octet directly before the user main program.

Inputs: See U_SetTM

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	0DAFh	See U_SetTM	See U_SetTM
Mask 0012h	0DAFh	See U_SetTM	See U_SetTM
Mask 0020h	5072h	See U_SetTM	See U_SetTM
Mask 1013h	3D2Fh	See U_SetTM	See U_SetTM

Watchdog-time: (to be completed) µs

Note: If you load the timer with the timer value 0, the timer is always expired. If you load the timer with 1, the timer can be immediatelly expired, because the timer tolerance is one timer tick. If you load the timer with a value >127, the bit 7 will be ignored.

2.5.4 U_GetTM Reading the User Timer Status

Symbol: **U_GetTM**

Description: gets the status of a user timer. This function is the only function that updates this (specified) timer.

Inputs:

- Mask 0010h, 0011h, 0012h, 0020h:

  A = user-timer number
  X = pointer to EEPROM-description-block

- Mask 1013h:

  A = See table below
  X = pointer to EEPROM-description-block

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Meaning	Location of timer-RAM-block 0 = UserRAM1 1 = UserRAM2	User-Timer number						

	Call Address	Outputs	Changed Registers
Mask 0010h	0D4Dh	See below	A, X, RegB, RegC, RegD
Mask 0011h	0D71h	See below	A, X, RegB, RegC, RegD
Mask 0012h	0D71h	See below	A, X, RegB, RegC, RegD
Mask 0020h	5060h	See below	A, X, RegB, RegC, RegD
Mask 1013h	3D2Ch	See below	A, X, RegB, RegC, RegD

Watchdog-time: (to be completed) µs

Outputs: Zero-Flag = 0 timer not yet expired
1 timer expired

Note: Periodically update the User-Timer. Otherwise some "timer-ticks" may be lost.

Symbol: **U_GetTMx**

Description: This function is the same as U_GetTM, except that the pointer to the pointer to EEPROM description block is fetched from the octet directly before the user main program.

Inputs: See U_GetTM

	Call Address	Outputs	Changed Registers
Mask 0010h	-		-
Mask 0011h	0D6Ch	See U_GetTM	See U_GetTM
Mask 0012h	0D6Ch	See U_GetTM	See U_GetTM
Mask 0020h	5063h	See U_GetTM	See U_GetTM
Mask 1013h	3D29h	See U_GetTM	See U_GetTM

Watchdog-time: (to be completed) µs

2.5.5 U_Delay

Symbol: **U_Delay** **Delay**

Description: waits the specified amount of time.
The delay is based upon the internal hardware-timer.

Inputs: A = delay time in 0.5 ms

Note: No delay times above 15 ms should be used.

	Call Address	Outputs	Changed Registers
Mask 0010h	0DDBh	None	A, X, RegB
Mask 0011h	0DFAh	None	A, X, RegB
Mask 0012h	0DFAh	None	A, X, RegB
Mask 0020h	5054h	None	A, X, RegB
Mask 1013h	3D02h	None	A, X, RegB

Watchdog-time: triggers watchdog internally

Th example below shows how a delay can be implemented using the user timers.

RAM-Data:

```
UsrTmr0     DS      1
```

EEPROM-Data:

```
UsrTmr      DB      LowOctet(UsrTmr0)
            DB      0               ;time base = 130ms
```

Code:

```
... other code ...
                                    ; start of timer
        lda     #10
        sta     RegE                ;time = 1.3 sec
        lda#0   ;user-timer 0
        ldx     #LowOctet(UsrTmr) ;ptr to description
        jsr     U_SetTM
... other code ...


                                    ; or other way to start a timer
        lda     #10
        sta     UsrTmr0
                                    ; check of timer status
        lda     #0                  ; user-timer 0
        ldx     #LowOctet(UsrTmr) ; ptr to description
        jsr     U_GetTM
        beq                         ; branch if timer
                                    ; expired
... other code ...      Message handling functions
```

2.5.6 BCU 2 Timer System

For the interpretation of the following 3 API-functions U_TS_Set, U_TS_Del and U_TS_Seti, please refer to the system implementation descriptions in [03].

The BCU 2 has a maximum of 3 system timers (1 to 3) available for the user. If a serial protocol is active it will use timer 1. If the U_debounce function is used, it will use timer 2.

The following modes and scales are possible:

- Modes

TS_TYPE_EVT	020h	Event only used with TS_Seti fixed scale of 6.6ms generates an event in PEI-driver event queue.
TS_TYPE_MSG	030h	Message generates a timerindication message
TS_TYPE_MSG_CYC	040h	Message cyclically generates a timerindication message
TS_TYPE_DOWN	050h	Down counter reserved for compatible function used with TM_Load
TS_TYPE_DIFF	060h	Difference counter reserved for compatible function used with TM_Load

- Scales

TS_SCALE041MS	00h	0.416	ms
TS_SCALE6_6MS	02h	6.6	ms
TS_SCALE106S	04h	106	ms
TS_SCALE1_6S	06h	1,6	sec
TS_SCALE26S	08h	26	sec

- Cyclic Scales (only for type messagecyclic)

TS_100MS	04h	100	ms
TS_1SEC	08h	1	sec
TS_1MIN	0ch	1	min

2.5.6.1 U_TS_Set Set a BCU 2 Timer

Symbol: **U_TS_Set**

Description : Set an BCU 2 Timer

Inputs

X	Timernumber
A	Timermode
RegB	Timer Scale / Cyclic Scale
RegC	TimerValue (not for Mode TS_TYPE_MSG_CYC)
RegD	TimerParameter (only Mode TS_TYPE_MSG)

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	5090h	None	A, X, RegB-RegF
Mask 0021h	5090h	None	A, X, RegB-RegF
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

2.5.6.2 U_TS_Del Delete BCU 2 Timer

Symbol: **U_TS_Del**

Description: Delete Timer

Inputs: X Timer number

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	5093h	None	None
Mask 0021h	5093h	None	None
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

2.5.6.3 U_TS_Seti Set Timer Interrupt Event

Symbol: **U_TS_Seti**

Description: Set BCU 2 Timer for Event Generation (only use in Interrupts)

Inputs:

X	Timernumber
iRegD	message queue ID or TS_TYPE_EVT
iRegC	Timervalue
A	Timer Parameter max. 6 bit

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	50ABh	None	None
Mask 0021h	50ABh	None	None
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

2.6 Message Handling Functions

2.6.1 AllocBuf Buffer Allocation

Symbol: **AllocBuf**

Description: allocates a message buffer.

Note: There is a distinction between long and short buffers. But non-system-software requires only long buffers.

Inputs: Carry: = buffer type
0 = short buffer
1 = long buffer

	Call Address	Outputs	Changed Registers
Mask 0010h	116Ah	See below	A
Mask 0011h	117Eh	See below	A
Mask 0012h	117Eh	See below	A
Mask 0020h	5000h	See below	A
Mask 1013h	3D8Fh	See below	A

Watchdog-time: (to be completed) µs

Outputs: X = pointer to buffer
Carry = 1 buffer allocated
0 no buffer allocated (all buffers in use) (X invalid)

2.6.2 FreeBuf Buffer Release

Symbol: **FreeBuf**

Description: releases a previously allocated buffer.

Inputs: X = pointer to buffer

	Call Address	Outputs	Changed Registers
Mask 0010h	118Ch	None	A, X, RegB
Mask 0011h	11A0h	None	A, X, RegB
Mask 0012h	11A0h	None	A, X, RegB
Mask 0020h	5006h	None	A, X, RegB
Mask 1013h	3D8C	None	A, X, RegB

Watchdog-time: (to be completed) µs

2.6.3 PopBuf Message request

Symbol: **PopBuf**

Description: searches for a certain message type.

Inputs:

- Mask 0010h, 0011h, 0012h, 0020h:

  A = message and buffer type

Bit #	7	6	5	4	3	2	1	0
Meaning	buffer type 0 = short 1 = long	layer address			must be 0			

- Mask 1013h:

  A = message and buffer type

Bit #	7	6	5	4	3	2	1	0
Meaning	buffer type 0 = short 1 = long	layer address			requested service or 0			

	Call Address	Outputs	Changed Registers
Mask 0010h	11ACh	See below	A, RegB
Mask 0011h	11C0h	See below	A, RegB
Mask 0012h	11C0h	See below	A, RegB
Mask 0020h	5003h	See below	A, RegB
Mask 1013h	3D92h	See below	A, RegB

Watchdog-time: (to be completed) µs

Outputs: X = pointer to buffer
Carry = 1 = message found
0 = no such message (X invalid)

2.6.4 U_MS_Post Post Messages

Symbol: **U_MS_Post**

Description: Post the message pointed by X to the message queue A

Inputs:

X pointer to message

A message queue ID

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	508Dh	None	A, X, RegB, RegL, RegM, RegN
Mask 0021h	508Dh	None	A, X, RegB, RegL, RegM, RegN
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

Example

```
...
        jsr     PopBuf          ; get message from user queue
        bcc     no_message      ; jump out if no message in queue
        lda     #TASK_PM_ID     ; load queue/task id from PEI
        jsr     U_MS_Post
... other code…
no_message
        ; continued here if no messages to post
```

2.6.5 U_MS_Switch Message redirection

Symbol: **U_MS_Switch**

Description: redirects messages from one task to an other ,all messages posted to queue ID X now posted to queue ID A.

Inputs:

X message queue ID to redirect

A new destination for queue X

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	508Ah	None	A, X, RegC
Mask 0021h	508Ah	None	A, X, RegC
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

Example

```
ldx     #TASK_AL_ID     ;load queue/task id from applicationlayer
lda     #TASK_US_ID     ;load queue/task id from user
jsr     U_MS_Switch     ;now all messages for the applicationlayer
                        ;are send to the user
                        ;receive messages
                        ;handle messages for application layer
clrx
clra
jsr     U_MS_Switch     ;reset all redirections
```

2.7 Arithmetic Functions

2.7.1 multDE_FG Unsigned Integer Multiplication

Symbol: **multDE_FG**

Description: multiplies the unsigned integer values in the registers RegD:RegE and RegF:RegG.

Inputs: RegD:RegE = factor
RegF:RegG = factor

	Call Address	Outputs	Changed Registers
Mask 0010h	0B3Ch	See below	A, X
Mask 0011h	0B4Bh	See below	A, X
Mask 0012h	0B4Bh	See below	A, X
Mask 0020h	5033h	See below	A, X
Mask 1013h	3D80h	See below	A, X

Watchdog-time: (to be completed) µs

Outputs: RegB:RegC = product
Carry = 0 ok
1 overflow

Note:

- Mask 1013h: In case of overflow the product is RegB:RegC = FFFFh.

2.7.2 divDE_BC Unsigned Integer Division

Symbol: **divDE_BC**

Description: divides the unsigned integer value in the registers RegD:RegE by the unsigned integer value in the registers RegB:RegC.

Inputs: RegD:RegE = dividend
RegB:RegC = divisor

	Call Address	Outputs	Changed Registers
Mask 0010h	0AFCh	See below	A, X, RegB, RegC
Mask 0011h	0B0Bh	See below	A, X, RegB, RegC
Mask 0012h	0B0Bh	See below	A, X, RegB, RegC
Mask 0020h	5030h	See below	A, X, RegB, RegC
Mask 1013h	3D83h	See below	A, X, RegB, RegC

Watchdog-time: (to be completed) µs

Outputs: RegF:RegG = quotient
RegD:RegE = remainder
Carry = 0 ok
1 divide by zero

2.7.3 FP_Flt2Int Float to Integer

Symbol: **FP_Flt2Int**

Description : Converts a 4 octet IEEE floatingpoint to 2 octet integer

Inputs:

X pointer to 32 bit float
RegF offset to exponent

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	5087h	See below	A, X, RegD, RegE
Mask 0021h	5087h	See below	A, X, RegD, RegE
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

Outputs RegB:RegC 2 octet integer

2.7.4 FP_Int2Flt Integer to Float

Symbol: **FP_Int2Flt**

Description : Converts an 2 octet integer to 4 octet IEEE floatingpoint

Inputs:

RegB:RegC	2 octet integer
X	pointer to 32 bit float
RegF	offset to exponent

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	5084h	See below	A, X, RegD
Mask 0021h	5084h	See below	A, X, RegD
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

Outputs 4 octet IEEE float at address X

2.8 Miscellaneous Functions

2.8.1 shlA*n* Accu shift left n

Symbol: **shlA*n***

Description: shifts the accu left by *n* bits. The values possible for *n* are 4, 5, 6, and 7. The new bits are set to zero.

Inputs: A = value

	Call Address				Outputs	Changed Registers
	shlA4	shlA5	shlA6	shlA7		
Mask 0010h	0B9Ah	0B99h	0B98h	0B97h	A = shifted value	None
Mask 0011h	0BA9h	0BA8h	0BA7h	0BA6h	A = shifted value	None
Mask 0012h	0BA9h	0BA8h	0BA7h	0BA6h	A = shifted value	None
Mask 0020h	5018h	501Bh	501Eh	5021h	A = shifted value	None
Mask 1013h	3D59h	3D5Ch	3D5Fh	3D62h	A = shifted value	None

Watchdog-time: (to be completed) µs

2.8.2 shrA*n* Accu shift right n

Symbol: **shrA*n***

Description: shifts the accu right by *n* bits. The values possible for *n* are 4, 5, 6, and 7. The new bits are set to zero.

Inputs: A = value

	Call Address				Outputs	Changed Registers
	shrA4	shrA5	shrA6	shrA7		
Mask 0010h	0BDAh	0BD9h	0BD8h	0BD7h	A = shifted value	None
Mask 0011h	0BE9h	0BE8h	0BE7h	0BE6h	A = shifted value	None
Mask 0012h	0BE9h	0BE8h	0BE7h	0BE6h	A = shifted value	None
Mask 0020h	5024h	5027h	502Ah	502Dh	A = shifted value	None
Mask 1013h	3D4Dh	3D50h	3D53h	3D56h	A = shifted value	None

Watchdog-time: (to be completed) µs

2.8.3 rolAn Accu rotate Left

Symbol: **rolA*n***

Description: rotates the accu left by *n* bits. The values possible for *n* are 1, 2, 3, 4 and 7.

Inputs: A = value

	Call Address					Outputs	Changed Registers
	rolA1	rolA2	rolA3	rolA4	rolA7		
Mask 0010h	-	-	-	-	-	-	-
Mask 0011h	-	-	-	-	-	-	-
Mask 0012h	0AF4h	0AF2h	0AF0h	0AEEh	0AECh	A = rotated value	None
Mask 0020h	5009h	500Ch	500Fh	5012h	5015h	A = rotated value	None
Mask 1013h	3D77h	3D74h	3D71h	3D6Eh	3D6Bh	A = rotated value	None

Watchdog-time: (to be completed) µs

2.8.4 U_SetBit Bit write

Symbol: **U_SetBit**

Description: sets the specified bit in register RegH.

Inputs: A = bit number
RegH = octet to modify
Carry = bit value to set

	Call Address	Outputs	Changed Registers
Mask 0010h	0DF9h	A = RegH = modified octet	RegB
Mask 0011h	0E18h	A = RegH = modified octet	RegB
Mask 0012h	0E18h	A = RegH = modified octet	RegB
Mask 0020h	5078h	A = RegH = modified octet	RegB
Mask 1013h	3D47h	A = RegH = modified octet	RegB

Watchdog-time: (to be completed) µs

2.8.5 U_GetBit

Symbol: **U_GetBit**

Description: reads the specified bit in register RegH.

Inputs: A = bit number
RegH = octet to be read from

	Call Address	Outputs	Changed Registers
Mask 0010h	0DEDh	See below	A, X, RegB
Mask 0011h	0E0Ch	See below	A, X, RegB
Mask 0012h	0E0Ch	See below	A, X, RegB
Mask 0020h	5075h	See below	A, X, RegB
Mask 1013h	3D44h	See below	A, RegB

Watchdog-time: (to be completed) µs

Outputs: Zero-flag = 0 if bit set
1 if bit clear

2.9 Loadable PEI-Support

2.9.1 U_FT12_Reset Reset FT 1.2 Protocol

Symbol: **U_FT12_Reset**

Description: Reset FT 1.2 protocol with specified baudrate

Inputs: A Baudrate

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	509Ch	None	A, X, RegB-RegN
Mask 0021h	509Ch	None	A, X, RegB-RegN
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

2.9.2 U_FT12_GetStatus Get FT 1.2 Protocol Status

Symbol: **U_FT12_GetStatus**

Description: Gets the FT 1.2 Protocollstatus

Inputs: Carry 0 get status
1 force status request

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	509Fh	See below	A
Mask 0021h	509Fh	See below	A
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

Outputs: Carry = 0 no new State
Carry = 1 new state
A = 0 State OK
≠ 0 State not OK

2.9.3 U_SCI_Init Initialize Serial Communication Interface

Symbol: **U_SCI_Init**

Description: Initialize the SCI of the 68HC05BE12
(for Serial Asynchronous Communication)
(enable SCI-System , disable SPI-System , 8 databits, receiver enable, transmitter enable configure PCMUX, set Baudrate)

Inputs A Baudrate

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	50A2h	None	A, X
Mask 0021h	50A2h	None	A, X
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

2.9.4 U_SPI_Init Initialize Serial Peripal Interface

Symbol: **U_SPI_Init**

Description: Initialize the SPI of the 68HC05BE12
(for Serial Synchronous Communication)
(enable SPI-System, disable SCI-System, configure PCMUX, set baudrate from EE_SyncRate, CPOL and CPHA from EE_ConfigDes)

Inputs None

	Call Address	Outputs	Changed Registers
Mask 0010h	-	-	-
Mask 0011h	-	-	-
Mask 0012h	-	-	-
Mask 0020h	50A5h	None	A, X
Mask 0021h	50A5h	None	A, X
Mask 1013h	-	-	-

Watchdog-time: (to be completed) µs

3 API for BIM M112

3.1 Overview

Function	Description	Clause
Message handling		
• _MSG_allocBuffer	allocates a message buffer	clause 3.2.7
• _MSG_freeMessage	releases a message buffer	clause 3.2.8
• _MSG_getMessage	retrieves a message from the message queue	clause 3.2.9
• _MSG_postMessage	posts an message to specified module	clause 3.2.10
• _MSG_staticRedirection	sets and returns a static redirection	clause 3.2.11
Miscellaneous		
• _AL_isSapLinked	returns if a AL-SAP is linked via the Association Table	clause 3.2.1
• _check	checks the system (or a part of it)	clause 3.2.2
• _copyMem	copies a block of memory	clause 3.2.3
• _getAPIVersion	returns the version of the Application Programming Interface	clause 3.2.4
• _runRomModule	executes a ROM-system module referenced by the module ID	clause 3.2.27
• _setParam	sets the initial values of the maximum restart counters or the routing counter	clause 3.2.28
PEI-IO		
• _PEI_analogueIn	reads an analogue value from the PEI	clause 3.2.13
• _PEI_binaryInOut	reads from and writes to the binary PEI-ports	clause 3.2.14
• _PEI_sendByteSerial	sends one byte via the serial interface	clause 3.2.25
PEI event queue		
• _PEI_addEvent	adds one event to the PEI event queue	clause 3.2.12
• _PEI_getEvent	gets the oldest event	clause 3.2.16
• _PEI_initEventQueue	initialises the event queue of the PEI.	clause 3.2.20
PEI buffer		
• _PEI_getFreeBufferSpace	return the free space of the PEI buffers	clause 3.2.17
• _PEI_initBuffer	initialises the PEI receive and send buffers	clause 3.2.19
• _PEI_rcvBufferRead	reads byte(s) from the PEI-receive buffer	clause 3.2.21
• _PEI_rcvBufferWrite	writes byte(s) to the PEI-receive buffer	clause 3.2.22
• _PEI_sendBufferRead	reads byte(s) from the PEI-send buffer	clause 3.2.23
• _PEI_sendBufferWrite	writes byte(s) to the PEI-send buffer	clause 3.2.24
PEI (miscellaneous)		
• _PEI_getActualPeiType	returns the value of the actual PEI-type	clause 3.2.15

• _PEI_init	initialises the PEI	clause 3.2.18
• _PEI_setTimer	start/stop the PEI-timer	clause 3.2.26
Timer		
• _TM_getSystemTime	returns the system counter	clause 3.2.30
• _TM_getUserTicks	returns the 25ms-time	clause 3.2.31
Scheduler		
• _installCallback	installs the call-back function	clause 3.2.5
• _modulMode	changes the state of a module and returns the actual state	clause 3.2.6
• _taskSwitch	enables or disables task-switching	clause 3.2.29

3.2 Description of API functions

3.2.1 API Function: _AL_isSapLinked

Description: returns if a AL-SAP is linked via the Association Table. The function searches for the first entry in the Association Table with a matching ASAP number.

Inputs: B ASAP number

Outputs: Carry 0 link exists
1 not linked

Changed Registers: CCR, A, Y

Call Address: FFA7h

There is no example for this topic.

May be used in: □ call-backs □ interrupts

Task switch: □ must be disabled

3.2.2 API Function: _check

Description: checks the CRCs of all memory blocks of the given owner.

Inputs: B owner number
0 system
1 address table
2 Association Table
3 application program (user)
4 PEI program

Outputs: Carry 0 OK
1 CRC error

Changed Registers: CCR, A, B, X, Y

Call Address: FFC8h

May be used in: □ call-backs □ interrupts

Task switch: □ must be disabled

3.2.3 API Function: _copyMem

Description:	copies a block of data from the source address to the destination address. This function copies also to EEPROM.	
Inputs:	B	number of bytes
	X	destination address
	Y	source address
Outputs:	none	
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FFBFh	
May be used in:	☐ call-backs ☐ interrupts	
Task switch:	☒ must be disabled	

3.2.4 API Function: _getAPIVersion

Description:	returns the version of the Application Programming Interface	
Inputs:	none	
Outputs:	A,B	version number
Changed Registers:	CCR, A, B	
Call Address:	FFD1h	
May be used in:	☒ call-backs ☒ interrupts	
Task switch:	☐ must be disabled	

3.2.5 API Function: _installCallback

Description: installs call-back functions or starts user threads.

Inputs:
- B type of function (PEI state machine, user timer, save routine, ASAP, user thread 1, thread 2)
- Y pointer to call-back function or user start address
- X pointer to the highest memory location of the stack (stack top for user threads only)

Outputs: none

Changed Registers: CCR, A, B, X, Y

Call Address: FFC5h

Note: The call-back function for the ASAP (Application Layer Service Access Point) has several parameters:

Inputs:
accu A: command A = 0 ⇒ polling SAP's
A = 1 ⇒ message for specific SAP
accu B: SAP-number
reg X: pointer to message (if command in A = 1)

Outputs:
carry: CY = 0 ⇒ access without error was possible
if unknown command in A or illegal service then
CY = 0
CY = 1 ⇒ SAP referred to in accu B not available

reg X: is only changed if command indicates polling !!!
X = 0 ⇒ no message for actual SAP generated
X = 1 ⇒ message for polled SAP generated

May be used in: □ call-backs □ interrupts

Task switch: □ must be disabled

3.2.6 API Function: _modulMode

Description: This function changes the state of a module and returns the actual state of the module. No other than the specified module IDs are allowed. If you user other IDs then this may cause malfunctions.

Inputs:
- A mode of module
  FFh no change of module state
- B module ID

Outputs: A actual mode of module in the bits 0 and 1.

Changed Registers: CCR, A, X

Call Address: FFC2h

May be used in: □ call-backs □ interrupts

Task switch: □ must be disabled

3.2.7 API Function: _MSG_allocBuffer

Description:	With this function an empty buffer can be allocated from the system. When the function is successfully, the X-register contains the pointer to the buffer.	
	Attention:	The task switch must be disabled until the buffer is released (freed or posted) again.
Inputs:	B	module ID
Outputs:	carry	indicates, if the function was successfully 0 = success 1 = error
	X	pointer to allocated buffer (only valid when carry = 0)
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FFB0h	
May be used in:	⊠ call-backs ⊠ interrupts	
Task switch:	⊠ must be disabled	

3.2.8 API Function: _MSG_freeMessage

Description:	With this function a previous allocated buffer can be released.	
Inputs:	X	pointer to the buffer, which should be released
Outputs:	carry	indicates, if the function was successfully 0 = always successfully
Changed Registers:	CCR, A, X	
Call Address:	FFB9h	
May be used in:	⊠ call-backs ⊠ interrupts	
Task switch:	⊠ must be disabled	

3.2.9 API Function: _MSG_getMessage

Description: With this function a module can get a message from a message queue. A message filter can be used to specify, which message (service) should be returned. If this filter is set to 0, the module gets the oldest message in the specified message-queue. If the filter has another value, the function searches for this service. The messages are received in the same order, as they were posted to the message queue.

Attention: The task switch must be disabled until the buffer is released (freed or posted) again.

Inputs:
- B: module ID
- A: service filter
  - 0 no filter used (get oldest message)
  - <>0 get oldest message with the service indicated by the service filter

Outputs:
- carry: indicates, if the function was successfully
  - 0 = success (message found)
  - 1 = error
- X: pointer to message-buffer (only valid when carry = 0)

Changed Registers: CCR, A, X

Call Address: FFB6h

May be used in: □ call-backs □ interrupts

Task switch: ⊠ must be disabled

3.2.10 API Function: _MSG_postMessage

Description: With this function a message can be sent to another module. Only message buffers which are received from MSG_allocBuffer or MSG_getMessage may be posted. There are three modes of this function:

- post a message and execute the active redirections
- post a message without executing a redirection
- post a message and set a temporary redirection

Use only one message with a temporary redirection at one time!

Inputs:
- A: temporary redirection (if requested)
- B: target module ID
- X: pointer to the buffer, which should be sent
- Y: mode of MSG_postMessage
  - 0 = MSG_normalPost normal posting
  - 1 = MSG_noRedir post without executing any redirection
  - 2 = MSG_tempRedir post and set a temporary redirection

Outputs:
- carry: indicates, if the function was successfully
  - 0 = success (always successfully)
  - 1 = error

Changed Registers: CCR, A, B, X, Y

Call Address: FFB3h

May be used in: ⊠ call-backs ⊠ interrupts

Task switch: ⊠ must be disabled

3.2.11 API Function: _MSG_staticRedirection

Description:	Sets a static redirection for a specific module and returns the current redirection (after set). In register A the module ID, to which module all messages for the module referenced in B will be sent, is specified. If register A is FFh, the redirection is not changed.	
Inputs:	A	Redirection (module ID)
	B	module ID
Outputs:	A	actual Redirection (module ID) (carry indicates, if the function was successfully 0 = success 1 = error)
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FFBCh	
May be used in:	☐ call-backs ☐ interrupts	
Task switch:	☐ must be disabled	

3.2.12 API Function: _PEI_addEvent

Description:	Adds one event to the PEI event queue and executes the PEI state machine (if not running). For internally generated events no data can be transferred.	
Inputs:	Y	event ID
Outputs:	none	
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FF98h	
May be used in:	☐ call-backs ☒ interrupts	
Task switch:	☒ must be disabled	

3.2.13 API Function: _PEI_analogueIn

Description:	This function reads an analogue value from the PEI. This function can only read one channel at a time.	
Inputs:	A	number of PEI channel
	B	number of read operations > 0
Outputs:	D	sum of read values
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FF8Fh	
	There is no example for this topic.	
May be used in:	☒ call-backs ☒ interrupts	
Task switch:	☒ must be disabled	

3.2.14 API Function: _PEI_binaryInOut

Description: This function reads from and writes to the binary PEI-ports. All ports are read, also the ports, which are set to output. But only those ports are written which are specified in the output mask.
For both the output data as well as the write mask the following bit definitions apply:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	I/O 5	I/O 4	I/O 3	I/O 2	I/O 1

Inputs:
A output data
B write mask, one bit per port (0 = read only, 1 = write)

Outputs: B data read from the PEI- ports after the write to the ports.

Changed Registers: CCR, A, B, X, Y

Call Address: FF8Ch

There is no example for this topic.

May be used in: ☒ call-backs ☒ interrupts

Task switch: ☒ must be disabled

3.2.15 API Function: _PEI_getActualPeiType

Description: returns the value of the current PEI-type.

Inputs: none

Outputs: A current PEI-type

Changed Registers: CCR, A, B, X, Y

Call Address: FFA4h

May be used in: ☐ call-backs ☐ interrupts

Task switch: ☒ must be disabled

3.2.16 API Function: _PEI_getEvent

Description: Gets the oldest event from the PEI event queue.

Inputs: none

Outputs:
Zero flag 0 event found
1 no event in event queue
D, Y event data (see description of event queue)
D (high byte) event ID

Changed Registers: CCR, A, B, X, Y

Call Address: FF9Bh

May be used in: ☐ call-backs ☒ interrupts

Task switch: ☒ must be disabled

3.2.17 API Function: _PEI_getFreeBufferSpace

Description: returns the free space of the PEI buffers.

Inputs: none

Outputs:
- Y free space in receive buffer in bytes
- X free space in send buffer in bytes

Changed Registers: CCR, A, B, X, Y

Call Address: FF7Dh

May be used in: □ call-backs □ interrupts

Task switch: ☒ must be disabled

3.2.18 API Function: _PEI_init

Description: initialises the serial interface and sets the port directions.
The operations which are executed are specified in the B-register.
You must call this function before you can use the serial interface.

Inputs:

B
- Bit 0:
  - 0 disable function
  - 1 execute initialisation of serial interface
- Bit 1:
  - 0 disable function
  - 1 set CTS usage according bit 2
- Bit 2:
  - 0 no CTS interrupt
  - 1 CTS interrupt
- Bit 3:
  - 0 disable function (Y then not needed)
  - 1 set PEI_SerConfig & PEI_SerBaud
- Bit 4:
  - 0 disable function (X then not needed)
  - 1 set PEI direction register

Y
- Bit 0 - 7: PEI_SerBaud
  - 0 = 2 400
  - 1 = 4 800
  - 2 = 9 600
  - 3 = 19 200
  - 4 = 38 400
  - 5 = 76 800
- Bit 8 - 15: PEI_SerConfig
- Bit 8:
  - 0 disable asynchron interface
  - 1 enable asynchron interface
- Bit 9:
  - 0 enable TC transmit complete interrupt
  - 1 enable TDRE transmit data register empty interrupt
- Bit 10:0 must be 0
- Bit 11:0 must be 0
- Bit 12:0 must be 0
- Bit 13:
  - 0 BCU like
  - 1 PC like
- Bit 14:0 must be 0
- Bit 15:
  - 0 8 data bits
  - 1 9 data bits

	X	PEI direction register (0=input, 1=output) Bit 0: I/O 1 Bit 1: I/O 2 Bit 2: I/O 3 Bit 3: I/O 4 Bit 4: I/O 5 Bit 5 must be 0 Bit 6 must be 0 Bit 7 must be 0

Outputs: none

Changed Registers: CCR, A, B, X, Y

Call Address: FFA1h

May be used in: □ call-backs □ interrupts

Task switch: ⊠ must be disabled

3.2.19 API Function: _PEI_initBuffer

Description: This function initialises the PEI receive and send buffers. You must call this function before any use of these buffers.
If you only want to clear a buffer, you must remove all the data by read operations.

For buffer description details see: PEI Buffer Description

Inputs: X pointer to the buffer description

Outputs: none

Changed Registers: CCR, A, B, X, Y

Call Address: FF7Ah

May be used in: □ call-backs □ interrupts

Task switch: ⊠ must be disabled

See also: Structure of the PEI Software

3.2.20 API Function: _PEI_initEventQueue

Description: initialises the event queue of the PEI. All events currently in the queue are deleted.

Inputs: none

Outputs: none

Changed Registers: CCR

Call Address: FF95h

May be used in: □ call-backs □ interrupts

Task switch: ⊠ must be disabled

3.2.21 API Function: _PEI_rcvBufferRead

Description:	Reads data from the PEI-receive buffer. If the X-register is set to '0' the function reads one byte only. If the X-register is non-zero, it must point to a memory block.	
Inputs:	X	0 byte-mode <> 0 block-mode: pointer to memory block
	B	length of data block (only in block mode) excluding length-byte
Outputs:	carry	indicates if bytes are read 0 bytes read 1 no bytes read
	B	byte-mode: data byte, read from buffer block-mode: number of bytes read
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FF80h	
May be used in:	☐ call-backs ☒ interrupts	
Task switch:	☒ must be disabled	

3.2.22 API Function: _PEI_rcvBufferWrite

Description:	writes data to the PEI-receive buffer. If the X-register is set to '0' the function writes one byte. If the X-register is non-zero, then it must point to a memory block, which contains the data. The B-register contains the number of bytes. In block-mode the complete block including a leading length byte is sent.	
Inputs:	X	0 byte-mode <> 0 block-mode: pointer to memory block
	B	byte-mode: byte block-mode: length of data block, excluding length-byte
Outputs:	carry	indicates if the data are written successfully 0: bytes written 1: error
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FF83h	
May be used in:	☐ call-backs ☒ interrupts	
Task switch:	☒ must be disabled	

3.2.23 API Function: _PEI_sendBufferRead

Description:	reads data from the PEI-send buffer. If the X-register is set to '0' the function reads one byte only. If the X-register is non-zero, it must point to a memory block.	
Inputs:	X	0 byte-mode <> 0 block-mode: pointer to memory block
	B	length of data block (only in block mode) excluding length-byte
Outputs:	carry	indicates if bytes are read 0 bytes read 1 no bytes read
	B	byte-mode: data byte, read from buffer block-mode: number of bytes read
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FF86h	
May be used in:	□ call-backs ⊠ interrupts	
Task switch:	⊠ must be disabled	

3.2.24 API Function: _PEI_sendBufferWrite

Description:	writes data to the PEI-send buffer. If the X-register is set to '0' the function writes one byte. If the X-register is non-zero, then it must point to a memory block, which contains the data. The B-register contains the number of bytes. In block-mode the complete block including a leading length byte is sent.	
Inputs:	X	0 byte-mode <> 0 block-mode: pointer to memory block
	B	byte-mode: byte block-mode: length of data block, excluding length-byte
Outputs:	carry	indicates if the data are written successfully 0: bytes written 1: error
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FF86h	
May be used in:	□ call-backs □ interrupts	
Task switch:	⊠ must be disabled	

3.2.25 API Function: _PEI_sendByteSerial

Description:	sends one byte via the serial interface.	
Inputs:	A	byte to send
	B	MSB = 9th data bit or parity bit
Outputs:	none	
Changed Registers:	CCR, A, B	
Call Address:	FF92h	
May be used in:	⊠ call-backs □ interrupts	
Task switch:	⊠ must be disabled	

Known BUG in the 0700 MASK version:

- The parity bit is not set correctly due to a mistake in the handling of the parity bit.

Work around:

- set this: bclr 02ch,#040h

immediately before calling _PEI_sendByteSerial if you use parity!

3.2.26 API Function: _PEI_setTimer

Description:	starts and stops a timer for the PEI-state-machine. There are two timer available. The time-base of the timer is 0,407 µs. The maximum value is 2^31. If a timer expires then a timer event is generated in PEI event queue. is stop then also pending timer events are changed to NOP-events.	
Inputs:	A	timer number (0, 1)
	X:Y	timer value (0 = stop)
Outputs:	none	
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FF9Eh	
	There is no example for this topic.	
May be used in:	☐ call-backs ☒ interrupts	
Task switch:	☒ must be disabled	

3.2.27 API Function: _runRomModule

Description:	executes a ROM-system module referenced by the module ID.	
Inputs:	A	0 normal call 1 initialisation
	B	module ID
Outputs:	none	
Changed Registers:	CCR, A, B, X, Y	
Call Address:	FFCBh	
May be used in:	☐ call-backs ☐ interrupts	
Task switch:	☐ must be disabled	

3.2.28 API Function: _setParam

Description: sets the initial values of the maximum restart counters or the routing counter

Inputs:

A
- 0 set restart counters
- 1 set routing counter
- 2 set the runtime for user task 1 in system ticks
- 3 set the runtime for user task 2 in system ticks
- 4 set the runtime for user task 3 in system ticks

B value to set:

- Setting the restart counters:

  The high nibble defines the maximum number of repetitions, when busy-acknowledgements are received, the low nibbles defines the maximum number of repetitions, when not-acknowledgements or no acknowledgements are received.

- Setting the routing counter :

  Values from 0 to 7 are allowed.

- Setting the runtime for a user task:

  Values from 1 (2,5 ms) to 10 (25 ms) are allowed. The default value is 2 (5 ms). A value bigger then 15 does not guarantee retriggering of the watch dog .

Outputs: none

Changed Registers: CCR, A, B

Call Address: FF77h

May be used in: ☒ call-backs ☒ interrupts

Task switch: ☐ must be disabled

3.2.29 API Function: _taskSwitch

Description: Enables or disables the task switch. Each time this function is called a counter is incremented (disable) or decremented (enable). Only when the counter is zero a task switch of a user task may be executed.
In addition a user task can use to release the CPU immediately.

Inputs: A = 0 disable task switch
1 enable task switch
FFh release CPU

Warning: If you disable the task switch in a user application for longer time (>100ms) a COP (watch dog) reset may occur. In normal operation the system restarts the watchdog. For manually restarting the watchdog execute this sequence:

```
ldaa    #055h
staa    03Ah
coma
staa    03Ah
```

Outputs: none

Changed Registers: CCR, A

Call Address: FFCEh

May be used in: □ call-backs □ interrupts

Task switch: □ must be disabled

3.2.30 API Function: _TM_getSystemTime

For the 0700 and 0701 MASK version only !

Description: Returns the current system time.
The system time is the value of the internal free running 16-bit counter extended to 32-bits. The time is directly derived from the crystal.
For a crystal frequency of 9,830 4 MHz the time unit for this counter is 406,901 042 ns.

Inputs: none

Outputs: X low counter word
D high counter word

Changed Registers: CCR, D, X

Call Address: FFAAh

May be used in: □ call-backs □ interrupts

Task switch: □ must be disabled

3.2.31 API Function: _TM_getUserTicks

Description:	Returns the user time. The user time is the value of a free running 3 octet counter. The time unit for this counter is 25 ms.	
Inputs:	none	
Outputs:	X B	low counter word most significant counter byte
Changed Registers:	CCR, B, X	
Call Address:	FFADh	
May be used in:	□ call-backs □ interrupts	
	Task switch: ⊠ must be disabled	