# BCU 1

## Introduction



## Reference



## Troubleshooting

# Introduction

The bus-coupling-unit is an essential component of the EIBus. It is the standard interface for all bus-devices. Due to its universal usage the bus-coupling-unit may contain the application-program or just serves as bus-interface. Concerning reconfiguration the bus-coupling-unit can be programmed as well via the bus as via the PEI.

## Controller

| | |
|---|---|
| - CPU: | MC68HC05B6 |
| - Operating Frequency: | 2,0 MHz(Crystal Frequency of 4MHz) |
| - On-Chip RAM: | 176 Bytes, (18 Bytes available for user ) |
| - On-Chip EEPROM: | 256 Bytes, (230 Bytes available for user) |

- 8-Bit A/D-Converter (5 Channels available for user)
- 8-Bit Pulse Length Modulator (PLM)
- Serial Asynchronous Communication-Interface
- Serial Synchronous Communication-Interface
- Watch Dog

## Language support

The user interface of the operating system of BCU is defined on assembler level. It is possible to implement any language support for any cross assembler and any cross compiler.
With our templates and project generation we support IAR products.

| | |
|---|---|
| Micro Series 6805 Assembler V1.80/MD2 | (c) Copyright IAR Systems 1985 |
| Micro Series Universal Linker V4.47D/DXT | (c) Copyright IAR Systems 1995 |

The supported language (by the IDE) for implementing user modules is assembler only. There are enviromental adjustments made for easier supporting the products from IAR Systems.

Segment definitions:

| | | |
|---|---|---|
| EEPRM | (100h-115h) | Sytem EEPROM |
| CODE | (116h-1ffh) | EEPROM for User Tables and Code |
| ZDATA | (CEh-DFh) | Zeropage RAMData |

# Compatibility

## Compatibility

**Never** use any **undocumented feature** of the BCU !

Such an application-software will not run on any future version of the BCU.

## Warning

**Never** write complete bytes on port C of the BCU.

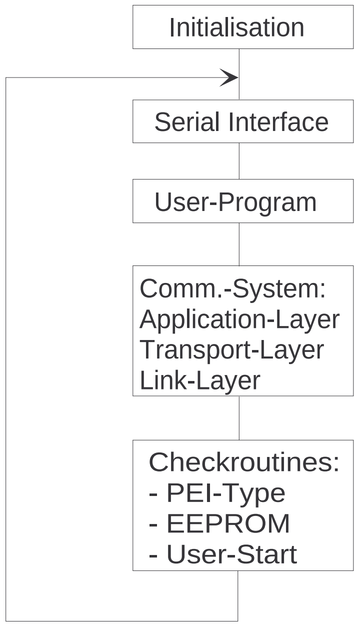
The following commands are **forbidden** :

```
sta PORTC        ; addr. 02H
stx PORTC
```

and so on...

# Structure of the BCU 1 system software

The system software consists of two parts: a sequential part and an interrupt-driven part.
The interrupt-driven part is responsible for receiving messages from the bus and for the start of the user-save-routine.
The sequential part is a large loop which is shown below.




# Addressing

There are two types of addresses : physical addresses and group-addresses.

Physical addresses are unique system-wide. They incorporate the overall system-topology. I.e. : they include functional-area- and branch-number.

The physical address is an unique number for a physical access-point, the BCU, to the bus.

The physical addressing is reserved for system-management functions.

The group-addresses are the communication-addresses for communication between the communication-objects (with their corresponding functions).

That means from the user's point of view, that the group-addresses are just codes for application functions.

For example : The lights (all) in an office can be switched on and off via group-address 100.

Hence for a user, the group-address 100 is just a code for "office lighting".

From the system's point of view, the group-address 100 is the communication address for the communication-objects belonging to the "office lighting".

No programming of BCUs is possible using group-addresses.

# Parametrization-Mode (for setting phys. addr.)

The press-button-switch (toggle-function) on the BCU is used to select or deselect the parametrization-mode of the BCU.

Parametrization-mode means, that in this mode the physical address of a BCU can be set or read via broadcast. I.e. this can be done without knowledge of the physical address of the BCU.

In parametrisation mode, the LED on the BCU is switched on, otherwise off.

# PEI Type Detection

The current PEI type is checked cyclic. The PEI is each time set according to the measured PEI type.

# User-Program-Start

The user-program is only started if the EEPROM and the communication-object- and association-table-data are ok (i.e. the corresponding error-flags are not set (low active)) and the current PEI type matches the required PEI type in EEPROM.

If one of the above conditions is not fulfilled, then the user-program is stopped.

If later on, the above conditions are fulfilled, then the user-program is restarted.

Start or restart of the user-program always means, that first the user-initialization-routine is called and the User-RAM is cleared once and then the user-routine is called periodically.
Each time before the user-routine is called, the registers (RegB - RegN) are cleared.

# Check-Routines

The EEPROM is periodically checked. In the case of an error, the corresponding runtime-error-flag is set.

The runtime-error-flags can be read and reset via the bus.

In addition to a correct checksum, the following conditions must be fulfilled to assume a valid EEPROM-contents :

1. Length of address table < 116
2. PEI-type in EEPROM < 20
3 Pointer to RAM-flags-table in communication- objects-table points to User RAM area.

There are some other internal consistency checks which may cause a restart of the BCU in the case of error (e.g. stack overflow).

# Watchdog

The Watchdog-System is automatically started by the system-software. Taking into account the time which may be consumed by interrupt-routines, e.g. bit receive, the watchdog must by triggered about every 1.5ms.

Concerning the user-program, the watchdog is triggered just before calling it and just after returning from it. In the User-Program itself, the application program writer is responsible for appropriately triggering the watchdog.

# Tool Interface

# Device Model

In this chapter the device model is decribed from the tools point of view. This means that many aspects of the device model which are not relevant for a tool are skipped.

From the tools point of view the BCU consists of five major parts. These are:

1. System Parameters
2. Address Table
3. Association Table
4. Communication Object Table
5. Application Program

The location and interpretation of the system parameters are described in the EIB-Handbook. Only the error flags (10DH), the length of the address table, the physical address, and the pointers to the association and the communication object tables are important for the tool. The other parameters are just set as specified by the developer.

In the address table the tool may only change the length and the group addresses. The physical address should usually not be changed directly. The maximum size of the address table is defined by the developer.

The association table is completely configured by the tool. The maximum size and location is defined by the developer.

In the communication object table the tool may change only the flags for the different communication objects, **except the value memory type flag**. The size and location is defined by the developer and must not be changed..

The application program and its parameters are defined be the developer. The parameters may also lie in the code, which means that a parameter selection may be a patch of the application program.

The address table and the association table can be resized by the tool provided the developer has designed the memory map appropriately. I.e. the association table is directly behind the address table. In such a case the pointer to the association table has also to be modified.

# Memory Access

The memory is directly accessed by the MemoryRead- and MemoryWrite-services. This access is based on the physical memory addresses.

# Calculating / Checking Checksums

The developer selects a memory area which is supervised by a checksum. The area starts at 0108H and ends between 0108H and 01FEH as defined by the developer.

Each time a value is written to this area the checksum is updated.

The checksum is periodically checked. If there is an error then the corresponding bit in the EEPROM error flags (10DH) is set. One consequence of that is that the application program is stopped and the communication objects do not work any longer.

# Application Program Control

The application program can be stopped by setting the EEPROM error flags (10DH) to 00H.

If the EEPROM error flags (10DH) are cleared (FFH) then the application program is automatically started as soon as the correct PEI type is detected.

# Device Identification

The only information one can get about a BCU using the old tool interface is the so called mask type and mask version number. This information can be read using the MaskRead-service.

The mask type identifies a device class, e.g. a BCU (00H) or line coupler.

The mask version identifies the version of that device, e.g. version 1.2 of an BCU.
For the BCU versions 1.0 to (at least) 1.2 the mask version also identifies the API version. The compatibility rules for the application programs are set up in the following way. All application programs for BCUs with lower version numbers can be loaded in those with higher version numbers which are in the compatibility list of the tool. Therefore the tool must keep a list of compatible versions.

Note: To know that an application program may be loaded in a certain BCU version does not mean that the application really works. For proper operation the right application module or compact device is required, too.

# Load Procedure

# Programming the Physical Address

The physical address is programmed in the following way:

Check for Existence

1. Try to connect (T_CONNECT) to a device with the specified physical address.

Check for selection

2. Check using the PhysAddrRead-service via broadcast, whether the programming button of a device was pressed, inform the user, and wait.
   In the case of multiple selection abort the procedure with an error message.
   In the case that a device with the specified physical address was found but is not selected abort the procedure with an error message.

Programming the Physical Address

3. Program the physical address using the PhysAddrWrite-service via broadcast.

Verify Programming

4. Set up a communication connection to the specified physical address.
5. Use the Reset-service to reset the device.
   Note that by this operation first the programming LED is switch off, and second the communication connection breaks down.

The procedure for programming the physical address via the PEI is different for the different tool interfaces. Using the old tool interface the physical address must be changed by direct memory access. Using the new tool interface the same procedure, with some additional initialization must be used as via the bus.

# Loading Applications

The old load procedure is very simple. It is based completely on direct memory access.

Connecting

1. Connect via bus or serial PEI.

Verifying BCU version

2. Check for the correct mask type and mask version.

Verifying Hardware type

3. Check whether the port A direction bit setting in the application program (10CH) matches that in the BCU.

Preparing for Download

4. Set the right entry points in the application program according to the actual BCU version.
5. Generate the memory image of the BCU including all tables, parameters, and the application program.

Loading the Data

6. Set all error flags in the BCU (10DH = 00H).
7. Set the length of the address table to 1.
8. Load the data from 100H to 100H by direct memory access.
9. Load the data from 104H to 10CH by direct memory access.
10. Load the data from 10EH to 115H by direct memory access.
11. Load the data from 119H to 4??H by direct memory access.
12. Erase the user RAM (0CEH to 0DFH). This is required for BCUs 1.x only.
13. Set the length of the address table.
14. Reset all error flags in the BCU (10DH = FFH).

Disconnecting

15. Disconnect via bus or serial PEI

# Modifying the Address and Association Tables

The address and association table can be easily modified (without change of maximum size) by writing new values to it. In this case it must be assured that no malfunctions occur during the modification process. This can be done by setting the length of the address table to one and the length of the association table to zero during the modification. When the modification is finished then the length can be set to the desired value.

The address and association table can be easily modified (without change of maximum size) by writing new values to it. In this case it must be assured that no malfunctions occur during the modification process. This can be done by setting the length of the address table to one and the length of the association table to zero during the modification. When the modification is finished then the length can be set to the desired value.

To change the maximum size of the tables is more complicated. In any case the memory layout defined by the developer must not be corrupted.

The change requires that also the pointer to the association table is updated. To do the whole operation it is necessary to set the BCU in a safe state for these chages. The procedure is as follows.

Connecting

1. Connect via bus or serial PEI.

Preparing for Modification

2. Check for the correct operating system and application version.
3. Set the error flags 10DH=00H.
4. Set length of Address Table to 1.

Modifying the Address Table

5. Change the data by direct memory access (absolute code + data)

Modifying the Association Table

6. Set the pointer to the Association Table, only if required!
7. Set length of Association Table to 0.
8. Change the data by direct memory access (absolute code + data)
9. Set length of Association Table to desired value.

Disconnecting

10. Set length of Address Table to desired value.
11. Reset the error flags 10DH=FFH.
12. Disconnect via bus or serial PEI

# Diagnostics

This chapter describes how one do diagnostics via the bus.

In this section the common diagnostics features are described. The features which are different for the old and new tool interface are then described in the corresponding chapters.

The most important diagnostic information can be received from the system status byte at memory address 60H. Via this byte the status of the BCU can also be changed.

| Parity | DwnMod | UsrEn | SerEn | ALEn | TLEn | LLMod | ProgMod |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

ProgMod — Program mode bit
- 0 BCU is in normal mode (LED off)
- 1 BCU is in program mode (LED on)

LLMod — Link Layer mode bit
- 0 Link Layer is in monitoring mode

| | | |
|---|---|---|
| | 1 | Link Layer is in normal operation mode |
| TLEn | Transport Layer enable bit | |
| | 0 | Transport Layer is disabled |
| | 1 | Transport Layer is enabled |
| ALEn | Application Layer enable bit | |
| | 0 | Application Layer is disabled |
| | 1 | Application Layer is enabled |
| SerEn | Serial PEI enable bit (for message transfer viy PEI only) | |
| | 0 | Serial PEI is disabled |
| | 1 | Serial PEI is enabled |
| UsrEn | Application program enable bit | |
| | 0 | Application program is disabled |
| | 1 | Application program is enabled |
| DwnMod | Download mode bit | |
| | 0 | Download mode is disabled |
| | 1 | Download mode is enabled |
| Parity | Parity bit for this byte (even) | |

The actual PEI-type can be read from the AD-converter channel 4. To read the value use the AdcRead-service. The value read can be converted to the actual PEI-type by using the following formula:

$$PEI\_Type = \frac{10 \cdot ADC\_Value + 60}{128}$$

**Warning:** The following procedure may not work with all devices!

A coarse estimation for the current bus voltage can be obtained from the AD-converter channel 1. To read the value use the AdcRead-service. The value read can be converted to an voltage value by using the following formula:

$$Voltage = ADC\_Value \cdot 0.15V$$

Additional useful information can be read from the following memory cells:

| Address | Contents | Remark |
|---|---|---|
| 10DH | Error Flags | see Memory Map |
| 109H | Required PEI Type | see Memory Map |

# BCU 1 API

## Communication-Object-Manipulation



## EEPROM-Manipulation-Support-Functions



## Application-Support-Functions



## PEI-Support-Functions



## Timer-Support-Functions



## Message-Handling-Functions



## Arithmetic Functions



## Miscellaneous Functions



## Tables



# API

The following information is valid for the standard BCU-system-software version 1.0 and 1.1 only.

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.

# Communication-Object-Manipulation-Functions

## Reading RAM-Flags

Symbol : **U_flagsGet**

Call Address :
Mask 1.0: 0C8CH
Mask 1.1 / 1.2: 0C9DH
Mask 2.0: 505AH »Page 62

Description : gets the RAM-flags of the specified communication-object.

Inputs : A = communication-object-number

Outputs : RegB = RAM-flags

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Meaning | Undefined | | | | update flag | data-request flag | transmission status | |

Effects : changed Registers : A, X, RegC, RegJ

## Writing RAM-Flags

Symbol : **U_flagsSet**

Call Address :
Mask 1.0: 0C94H
Mask 1.1: 0CB3H
Mask 2.0: 505HD »Page 62

Description : sets the RAM-flags of the specified communication-object.

Inputs : A = communication-object-number
RegB = RAM-flags

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Meaning | Undefined | | | | update flag | data-request flag | transmission status | |

Outputs : none

Effects : changed Registers : A, X, RegB, RegC, RegJ

## Testing the RAM-Flags

Symbol : **U_testObj**

Call Address :

| | |
|---|---|
| Mask 1.0: | not available |
| Mask 1.1 / 1.2: | 0CA5H |
| Mask 2.0: | 507BH »Page 62 |

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

Inputs : A = communication-object-number

Outputs : RegC = RAM-flags (right adjusted)
Zero-flag = **NOT** Update-flag

Effects : changed Registers : A, X, RegB, RegC, RegJ

## Setting Transmit-Request

Symbol : **U_transRequest**

Call Address :

| | |
|---|---|
| Mask 1.0: | 0D91H |
| Mask 1.1 / 1.2: | 0DB9H |
| Mask 2.0: | 507EH »Page 62 |

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

Inputs : A = communication-object-number

Outputs : Carry
- 0 = OK
- 1 = transmit-request not set

Effects : changed Registers : A, X, RegB, RegC, RegJ

# EEPROM-Manipulation-Support-Functions

## Writing to EEPROM

| | | |
|---|---|---|
| Symbol : | **EEwrite** | |
| Call Address : | Mask 1.0: | 0C2DH |
| | Mask 1.1 / 1.2: | 0C38H |
| | Mask 2.0: | 503FH |

Description : writes a byte 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

Outputs : none

Effects : changed Registers: RegB, RegC, RegH

## Updating Checksum

| | | |
|---|---|---|
| Symbol : | **EEsetChecksum** | |
| Call Address : | Mask 1.0: | 0C5DH |
| | Mask 1.1 / 1.2: | 0C68H |
| | Mask 2.0: | 503CH »Page 63 |

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

Inputs : none

Outputs : none

Effects : changed Registers : A, X, RegC, RegH

# Application-Support-Functions

## Debouncing

Symbol : **U_debounce**

Call Address :
Mask 1.0: 0C64H
Mask 1.1 / 1.2: 0C75H
Mask 2.0: 5051H

Description : debounces a complete byte. As long as the debounce time is not yet expired or the value is changing the latest debounced value is returned.
UserRAM = last input value
UserRAM+1 = debounced value

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

Inputs :
A = value (byte) for debouncing
X = debounce-time in 0.5ms-units

Outputs : A = debounced value (byte)

Effects :
uses software-timer 2
changed RAM-locations: UserRAM, UserRAM+1
changed Registers : X, RegB, RegC, RegD, RegE, RegF, RegG

Symbol : **U_deb10**

Call Address :
Mask 1.0: not available
Mask 1.1 / 1.2: 0C73H
Mask 2.0: 504BH

Description : This function is the same as U_debounce except that a fixed debounce time of 10ms is used.

Symbol : **U_deb30**

Call Address :
Mask 1.0: not available
Mask 1.1 / 1.2: 0C6FH
Mask 2.0: 504EH

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

## Ignoring Messages to the User

| | | |
|---|---|---|
| Symbol : | **U_delMsgs** | |
| Call Address : | Mask 1.0: | 0C82H |
| | Mask 1.1 / 1.2: | 0C93H |
| | Mask 2.0: | 5057H |

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

Outputs : none

Effects : changed Registers : A, X, RegB

## Characterisation-Function

| | | |
|---|---|---|
| Symbol : | **U_map** | |
| Call Address : | Mask 1.0: | 0C9BH |
| | Mask 1.1 /1.2: | 0CBAH |
| | Mask 2.0: | 5069H |

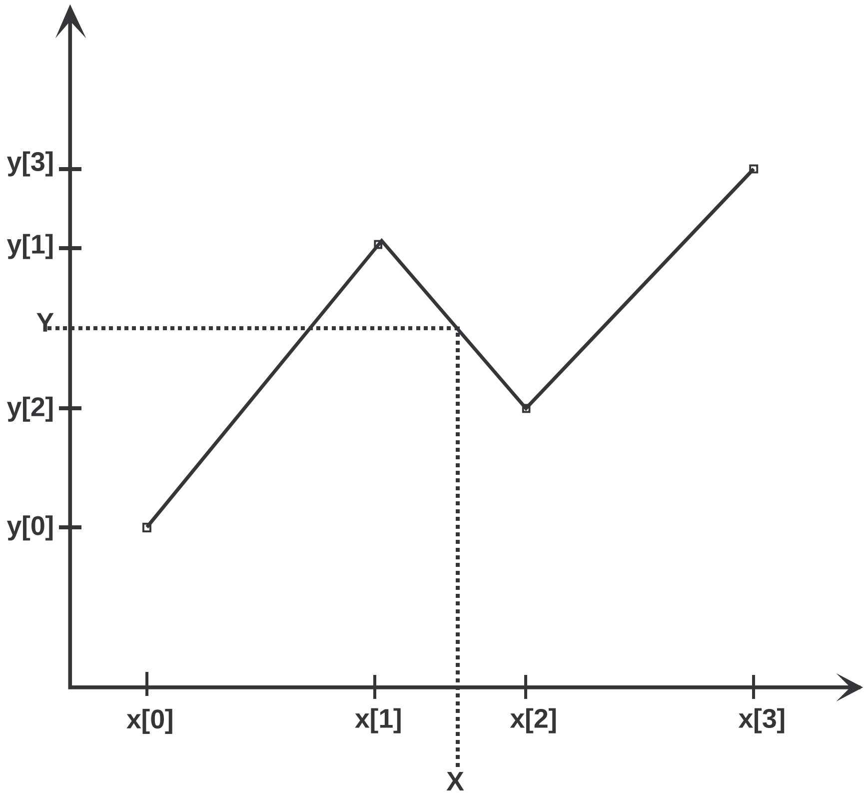
Description : maps the input value by the use of a conversion table. Input value and result are 16-bit signed integer numbers. For the conversion a table of x-y-valuepairs is used. The input value must be inside the x-value-range. For values in between two x-y-value-pairs linear interpolation is used.

The interpolation-formula used is :

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

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

Starting with mask-version 1.2 of the system-software, this function can operate in all 4 quadrants.




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

Length (x-y-pair count)
(1 byte)

| low word | high word |
| --- | --- |
| X0 | Y0 |
| X1 | Y1 |
| ... | ... |

Inputs : RegB:RegC = value to be mapped (signed integer)
X = pointer to conversion table (EEPROM-offset to Address 100h)

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

Effects : changed Registers : A, X, RegD, RegE, RegF, RegG, RegH, RegI

# Doing AD-Conversion

| | | |
|---|---|---|
| Symbol : | **U_readAD** | |
| Call Address : | Mask 1.0: | 0D35H |
| | Mask 1.1 / 1.2: | 0D54H |
| | Mask 2.0: | 506CH |
| Description : | starts AD-conversion for the specified 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. | |
| Inputs : | A = AD-port number<br>X = number of read operations | |
| Outputs : | RegD:RegE = sum of read values | |
| Effects : | changed Registers : A, X | |

# PEI-Support-Functions

## Binary-Port-Access

Symbol : **U_ioAST**

Call Address :

| | |
|---|---|
| Mask 1.0: | 0DA7H |
| Mask 1.1 / 1.2: | 0DCFH |
| Mask 2.0: | 5066H |

Description : Binary i/o is handled via the PEI-port.
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 |

Via i/o-flags reading or writing can be selected for each bit/pin.

Inputs : A = I/O-flags and values

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Meaning | I/O flags 0 = read ,1= write | | | | Bit values (for Writing only) | | | |
| I/O Bit # | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |

Outputs : RegB = read bit-values

| 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, all writeselected bits are set to 0) | | | |
| I/O Bit # | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |

Effects : Changed Registers : A, X, RegB, RegC, RegD

## Data-Block-Exchange via Serial PEI

Symbol : **S_AstShift** / **S_LAstShift**

Call Address : Mask 1.0: 1103H (S_AstShift)
Mask 1.1 /1.2: 1117H
Mask 2.0: 5042H

Call Address : Mask 1.0: 1101H (S_LAstShift)
Mask 1.1 / 1.2: 1115H
Mask 2.0: 5045H

Description : The specified data block is exchanged via the serial PEI-interface. The function "S_AstShift" uses a ca. 130ms- time-out and the function "S_LAstShift" uses a ca. 1s-time-out. I.e. the data-block-exchange must be completed within the time-out-time.
The data format is shown below :

| Message length in bytes (1 byte) | Data Max 24 byte |
| --- | --- |

Inputs : X = pointer to data block

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

Effects : changed Registers : A, RegB, RegC, RegD, RegE, RegF, RegG, RegI

## Byte-Exchange via Serial PEI

Symbol : **U_SerialShift**

Call Address : Mask 1.0: not available
Mask 1.1: not available
Mask 1.2: 0C90H
Mask 2.0: 5048H

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

Inputs: A = Byte to be transferred
Outputs: A = received byte
Carry = 0 : communication O.K.
1 : communication failed

Effects: changed registers: A, RegB, RegC, RegD, RegE, RegF, RegG,RegI

**Important Remarks to these 3 functions:**

- THIS FUNCTIONS MAY ONLY BE USED IN COMBINATION WITH PEI-TYPE 14.
- LONG WAIT-TIMES MAY CAUSE SERIOUS PROBLEMS ON THE BUS (BUSY-ACKNOWLEDGES).

# Timer-Functions

## BCU1 System-Timer

There are 4 software-timers : 0..3.

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 40ms.

| Range | Time-Unit |
|---|---|
| 1 | ca. 0.5 ms |
| 2 | ca. 8.0 ms |
| 3 | ca. 130 ms |
| 4 | ca. 2.1 s |
| 5 | ca. 33 s |

***Note : A not-initialized timer has the status of an expired timer.***

## Starting Timer

Symbol : **TM_Load**

Call Address : Mask 1.0: 0E0CH
Mask 1.1 / 1.2: 0E2BH
Mask 2.0: 5039H

Description : Starting Systemtimer
see BCU1 Systemtimer »Page 23
and
see BCU2 Systemtimer »Page 61

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)

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|

| **Meaning** | must be 0 | timer # | operation mode | timer range |
|---|---|---|---|---|

X = run-time (operation mode 0 only)

Outputs : none

Effects : changed Registers : A, X, RegB, RegC, RegD, RegE, RegF

## Reading Timer-Status

Symbol : **TM_GetFlg**

Call Address :

| | |
|---|---|
| Mask 1.0: | 0E2AH |
| Mask 1.1 / 1.2: | 0E49H |
| Mask 2.0: | 5036 |

Description :

Reading Systemtimer
see BCU1 Systemtimer »Page 23
and
see BCU2 Systemtimer »Page 61
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

Outputs :

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

Effects : changed Register : A, X, RegB, RegC, RegD, RegE

## User-Timer

There is another set of support routines which can be used to define additional User-Timers.

Each User-Timer 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 (bytes) | Meaning |
|---|---|---|
| 0 | 1 | Pointer to RAM-data |
| 1 | 1 | Time base 0 and 1;<br>timer 0 = low nibble;<br>timer 1 = high nibble |
| 2 | 1 | Time base 2 and 3 |
| .. | | … |

| Time Base Number | Minimum Time Resolution |
|---|---|
| 0 | ca. 130ms |
| 1 | ca. 260ms |
| 2 | ca. 520ms |
| 3 | ca. 1.0s |
| 4 | ca. 2.1s |
| 5 | ca. 4.2s |
| 6 | ca. 8.4s |
| 7 | ca. 17s |
| 8 | ca. 34s |
| 9 | ca. 1.1min |
| 10 | ca. 2.2min |
| 11 | ca. 4.5min |
| 12 | ca. 9.0min |
| 13 | ca. 18min |
| 14 | ca. 35min |
| 15 | ca. 1.2h |

For each User-Timer, a byte must be reserved in RAM. All of these bytes must by allocated in one block. The first byte in this block corresponds with the first User-Timer, the second byte corresponds with the second User-Timer and so on. The bit 7 in each byte is reserved, the bits 0-6 is used for the timer-value (0-127). 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.

.

## Setting User-Timers

Symbol : **U_SetTM**

Call Address : Mask 1.0: 0D8AH
Mask 1.1 / 1.2: 0DB3H
Mask 2.0: 506FH

Description : loads a User-Timer »Page 24

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

Outputs : none

Effects : changed Registers : A, X, RegB, RegC, RegD

Symbol : **U_SetTMx**

Call Address : Mask 1.0: not available
Mask 1.1 / 1.2: 0DAFH
Mask 2.0: 5072H »Page 64

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

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 immediate expired, because the timer-tolerance is one timer-tick. If you load the timer with a value >127, the bit7 will be ignored.

.

## Reading User-Timer-Status

Symbol : **U_GetTM**

Call Address : Mask 1.0: 0D4DH
Mask 1.1 / 1.2: 0D71H
Mask 2.0 5060H

Description : reads a User-Timer »Page 24
gets the status of a User-Timer. This function is the only function which updates this (specified) timer.

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

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

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

Effects : changed Registers : A, X, RegB, RegC, RegD

Symbol : **U_GetTMx**

Call Address : Mask 1.0: not available
Mask 1.1 / 1.2: 0D6CH
Mask 2.0: 5063H »Page 64

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

## Example

This example shows how a delay can be implemented using the user-timers.

```
RAM-Data:
UsrTmr0     rmb     1
EEPROM-Data:
UsrTmr      fcb     UsrTmr
            fcb     0                   ;time base = 130ms
Code:
... other code ...

; start of timer

        lda     #10
        sta     RegE    ;time = 1.3 sec
        lda     #0      ;user-timer 0
        ldx     #.Low.(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     #.Low.(UsrTmr);ptr to description
        jsr     U_GetTM

        beq     ...     ;branch if timer
;                        expired
... other code ...
```

## Delay

| | | |
|---|---|---|
| Symbol : | **U_Delay** | |
| Call Address : | Mask 1.0: | 0DDBH |
| | Mask 1.1 / 1.2: | 0DFAH |
| | Mask 2.0: | 5054H »Page 65 |

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

***Attention : No delay times above 15ms should be used.***

Inputs : A = delay time in 0.5ms

Outputs : none

Effects : changed Registers : A, X, RegB

# Message-Handling-Functions

## Buffer-Allocation

| | | |
|---|---|---|
| Symbol : | **AllocBuf** | |
| Call Address : | Mask 1.0: | 116AH |
| | Mask 1.1 / 1.2: | 117EH |
| | Mask 2.0: | 5000H |

Description : allocates a message buffer.

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

Inputs : Carry = buffer type (1 = long)

Outputs : X = pointer to buffer
Carry = 1 buffer allocated
0 no buffer allocated (X invalid)

Effects : changed Registers : A

## Buffer Release

| | | |
|---|---|---|
| Symbol : | **FreeBuf** | |
| Call Address : | Mask 1.0: | 118CH |
| | Mask 1.1 / 1.2: | 11A0H |
| | Mask 2.0: | 5006H |

Description : releases a previously allocated buffer.

Inputs : X = pointer to buffer

Outputs : none

Effects : changed Registers: A, X, RegB

# Message Request

Symbol : **PopBuf**

Call Address :

| | |
|---|---|
| Mask 1.0: | 11ACH |
| Mask 1.1 / 1.2: | 11C0H |
| Mask 2.0: | 5003H »Page 66 |

Description : searches for a certain message type.

Inputs : A = message and buffer type

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Meaning | Buffer type (1=long) | Layer address | | | must be 0 | | | |

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

Effects : changed Registers : A, RegB

# Arithmetic Functions

## Unsigned Integer Multiply

| | | |
|---|---|---|
| Symbol : | **multDE_FG** | |
| Call Address : | Mask 1.0: | 0B3CH |
| | Mask 1.1 / 1.2: | 0B4BH |
| | Mask 2.0: | 5033H |
| Description : | multiplies the unsigned integer values in the registers RegD:RegE and RegF:RegG. | |
| Inputs : | RegD:RegE = factor<br>RegF:RegG = factor | |
| Outputs : | RegB:RegC = product<br>Carry = 0 ok<br>1 overflow | |
| Effects : | changed Registers : A, X | |

## Unsigned Integer Divide

| | | |
|---|---|---|
| Symbol : | **divDE_BC** | |
| Call Address : | Mask 1.0: | 0AFCH |
| | Mask 1.1 /1.2: | 0B0BH |
| | Mask 2.0: | 5030H |
| 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<br>RegB:RegC = divisor | |
| Outputs : | RegF:RegG = quotient<br>RegD:RegE = remainder<br>Carry = 0 ok<br>1 divide by zero | |
| Effects : | changed Registers : A, X, RegB, RegC | |

# Miscellaneous Functions

## Accu Shift Left

Symbol : **shlA*n***

| | | | |
|---|---|---|---|
| Call Address : | Mask 1.0: | 0B9AH | (shlA4) |
| | | 0B99H | (shlA5) |
| | | 0B98H | (shlA6) |
| | | 0B97H | (shlA7) |
| | Mask 1.1 /1.2: | 0BA9H | (shlA4) |
| | | 0BA8H | (shlA5) |
| | | 0BA7H | (shlA6) |
| | | 0BA6H | (shlA7) |
| | Mask 2.0: | 5018H | (shlA4) |
| | | 501BH | (shlA5) |
| | | 501EH | (shlA6) |
| | | 5021H | (shlA7) |

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

Inputs : A = value

Outputs : A = shifted value

Effects : changed Registers : none

## Accu Shift Right

Symbol : **shrA*n***

| | | | |
|---|---|---|---|
| Call Address : | Mask 1.0: | 0BDAH | (shrA4) |
| | | 0BD9H | (shrA5) |
| | | 0BD8H | (shrA6) |
| | | 0BD7H | (shrA7) |
| | Mask 1.1/1.2: | 0BE9H | (shrA4) |
| | | 0BE8H | (shrA5) |
| | | 0BE7H | (shrA6) |
| | | 0BE6H | (shrA7) |
| | Mask 2.0: | 5024H | (shrA4) |
| | | 5027H | (shrA5) |
| | | 502AH | (shrA6) |
| | | 502DH | (shrA7) |

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

Inputs : A = value

Outputs : A = shifted value

Effects : changed Registers : none

## Accu_Rotate_Left

Symbol: **rolA*n***

| | | |
|---|---|---|
| Call Address: | Mask 1.0: | not implemented |
| | Mask 1.1: | not implemented |
| | Mask 1.2: | 0AF4H (rolA1) |
| | | 0AF2H (rolA2) |
| | | 0AF0H (rolA3) |
| | | 0AEEH (rolA4) |
| | | 0AECH (rolA7) |
| | Mask 2.0: | 5009H (rolA1) |
| | | 500CH (rolA2) |
| | | 500FH (rolA3) |
| | | 5012H (rolA4) |
| | | 5015H (rolA7) |

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

Inputs : A = value

Outputs : A = rotated value

Effects : changed Registers : none

## Bit Write

Symbol : **U_SetBit**

| | | |
|---|---|---|
| Call Address : | Mask 1.0: | 0DF9H |
| | Mask 1.1: | 0E18H |
| | Mask 1.2: | 0E18H |
| | Mask 2.0: | 5078H |

Description : sets the specified bit in register RegH.

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

Outputs : A = RegH = modified byte

Effects : changed Registers : RegB

# Bit Read

| | | |
|---|---|---|
| Symbol : | **U_GetBit** | |
| Call Address : | Mask 1.0: | 0DEDH |
| | Mask 1.1: | 0E0CH |
| | Mask 1.2: | 0E0CH |
| | Mask 2.0: | 5075H |
| Description : | reads the specified bit in register RegH. | |
| Inputs : | A = bit number<br>RegH = byte to be read from | |
| Outputs : | Zero-flag = 0 if bit set<br>1 if bit clear<br><br>RegB = value of OR_TAB | |
| Effects : | changed Registers: A | |

# Tables

| | | |
|---|---|---|
| Symbol : | **OR_TAB** | |
| Address : | Mask 1.0: | 0020H |
| | Mask 1.1: | 0020H |
| | Mask 1.2: | 0020H |
| | Mask 2.0: | 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 |

| | | |
|---|---|---|
| Symbol : | **AND_TAB** | |
| Address : | Mask 1.0: | 0028H |
| | Mask 1.1: | 0028H |
| | Mask 1.2: | 0028H |
| | Mask 2.0: | 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 helpful for **MASK**-operation. If you need a value of these tables, you need only a one-byte-offset. This is helpful for loops.

Example:

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

# BCU 1.x Macro Reference

Macros.inc Version 1.00

## Reference

**%ComTabPtr** <Value>

<Value> Value for Communication Table pointer to Offset 100h

**Description** CommsTabPtr »Page 49 is a pointer to the table of communication objects.

---

**%UsrInit** <Value>

<Value> Value for Userinitpointer

**Description** UsrInitPtr is a pointer to the entry point of the USER-initialization-routine in EEPROM.The initialization-routine starts at 100H+[UsrInitPtr].The initialization-routine is called once at user-startup-time.The initialization-routine must be written as a subroutine, i.e. it must be terminated by "rts".

---

**%UsrPrg** <Value>

<Value> Value for userprogrammpointer

**Description** UsrPrgPtr is a pointer to the entry point of the USER-program in EEPROM.The USER-program starts at 100H+[UsrPrgPtr].The USER-program is called periodically if the BCU is in normal operation mode.The USER-program must be written as a subroutine, i.e. it must be terminated by "rts".

---

**%UsrSave** <Value>

<Value> Value for Usersavepointer

**Description** UsrSavPtr is a pointer to the entry point of the USER-Save-program in EEPROM.The USER-Save-program starts at 100H+[UsrSavPtr].The USER-Save-program is called if the save-signal is **generated due to supply-power-breakdown** and the user is active at the same time.After calling the USER-Save-program the BCU is reset.The USER-Save-program must be written as a subroutine, i.e. it must be terminated by "rts".

---

| | |
|---|---|
| **%OptionReg** | <Value> |
| <Value> | Value for OptionRegister |
| **Description** | option register always FFh<br>( see 68HC05B6 specification for more details) |

---

| | |
|---|---|
| **%SyncRate** | <Value> |
| <Value> | Syncrate »Page 54 |
| **Description** | Defines the Syncrate »Page 54 for synchronous PEI (PEI-Type 12h and 14h) at 10Ah |

---

| | |
|---|---|
| **%PortCDDR** | <Value> |
| <Value> | Directionbits forPORTC »Page 45 |
| **Description** | Defines Port C Direction Bits Settings at 10Bh |

---

| | |
|---|---|
| **%PortADDR** | <Value> |
| <Value> | Directionbits forPORTA »Page 46 |
| **Description** | Defines Port A Direction Bits Settings at 10Ch |

---

**%RouteCnt** <Value>

| | | |
|---|---|---|
| <Value> | bits 3-0 | always 0 |
| | bits 4-6 | routing count |
| | bit 7 | U_delMsgs-call 0 = disabled<br>1 = enabled |

**Description** Defines the Value including Routing Count Start Value and U_delMsgs-switch at 10Eh

---

**%MxRstCnt** Value

| | | |
|---|---|---|
| <Value> | bits 2-0 | NAK-Restart-Limit |
| | bits 4-3 | always 0 |
| | bits 7-5 | BUSY-Restart-Limit |

**Description** Defines the values for NAK / BUSY Retrys at 10Fh

---

**%ConfigDes** Value

<Value>

bit 0 = PLMA (automatic) clear
- 0 = enabled
- 1 = disabled

bit 1 = CPHA ( see 68HC05BE12 specification for more details)
bit 2 = CPOL ( see 68HC05BE12 specification for more details)
bit 3 = telegram rate limitation
- 0 = enabled (limit @UsrInitPtr-1)
- 1 = disabled

bit 4 = allways 0
bit 5 = application processor
- 0 = exists
- 1 = does not exist

bit 6 = A_EVENT-message-generation
- 0 = enabled
- 1 = disabled

bit 7 = PLMB-frequency
- 0 = fast mode
- 1 = slow mode

**Description** Defines special functions for the BCU at 110h

---

**%AppID** Value

<Value> 1 Byte Manufacturer
2 Byte Application ID

**Description** Defines Application ID-Number and Manufacturer ID

---

**%AppVersion** <Value>

<Value> 1 Byte Applicationversion

**Description** Defines The Application Version

---

**%PeiType** <Value>

<Value> 1 Byte PeiType

**Description** Defines the requested PEI-Type of the application
This is the PEI-type which is required by the software in EEPROM. If this PEI-type matches not the one read from the PEI, then the USER-program is stopped.

---

**%IMPLEMENT_ADDRESSTABLE**

**Description** Marks the start of the Addresstable

---

**%END_ADDRESSTABLE**

**Description** Marks the end of the Addresstable

## %IMPLEMENT_ASSOCTABLE

**Description** Marks the start of the Associationtabletable

## %END_ASSOCTABLE

**Description** Marks the end of the Associationtabletable

## %COM_OBJECT

**%COM_OBJECT** <Pointer>, <Type>, <Flags> [,CSegPtr1]

<Pointer> 1 Byte Pointer to Object Value

<Type> 1 Byte Communication Object-Type
UINT1
UINT2
UINT3
UINT4
UINT5
UINT6
UINT7
UINT8
UINT16
TIME_DATE
FLOAT
DATA6
DOUBLE
DATA10
MAXDATA

<Flags>

| | |
|---|---|
| Bit0/1 | :Transmission Priority |
| Bit 2 | :Communication Enable (1=Enable) |
| Bit 3 | :Read Enable (1=Enable) |
| Bit 4 | :Write Enable (1=Enable) |
| Bit 5 | :Value Offset(0=0000h, 1=0100h) |
| Bit 6 | :Transmission Enable (1=Enable) |
| Bit 7 | :Always 1 |

**Description** Defines a communication Object-Description

# EEPROM

## Memory Map

| memory address | name | length | comment |
|---|---|---|---|
| $0100 | OptionReg »Page 44 | 1 | EEPROM Option Register |
| $0101-$0102 | ManData »Page 44 | 2 | BCU-Manufacturing Data |
| $0103-$0104 | Manufact »Page 44 | 2 | EEPROM-Software Manufacturer |
| $0105-$0106 | DevTyp »Page 44 | 2 | Device Type Number |
| $0107 | Version »Page 44 | 1 | Software-Version Number |
| $0108 | CheckLim »Page 45 | 1 | EEPROM check limit |
| $0109 | PEI_Type expected »Page 45 | 1 | required PEI-Type by Software |
| $010A | SyncRate »Page 54 | 1 | baud rate used for serial synchronous PEI |
| $010B | PortCDDR »Page 45 | 1 | Port C Direction Bit Setting for PEI-Type 17 |
| $010C | PortADDR »Page 46 | 1 | Port A Direction Bit Setting |
| $010D | RunError »Page 46 | 1 | Run Time Error Flags |
| $010E | RouteCnt »Page 47 | 1 | Routing-count constant |
| $010F | MxRstCnt »Page 47 | 1 | INAK-Retransmit-Limit BUSY-Retransmit-Limit |
| $0110 | ConfigDes »Page 47 | 1 | Configuration Descriptor |
| $0111 | AssocTabPtr »Page 48 | 1 | Pointer to Association Table |
| $0112 | CommsTabPtr »Page 49 | 1 | Pointer to Communication Object Table |
| $0113 | UsrInitPtr »Page 51 | 1 | Pointer to USER Initialization Routine |
| $0114 | UsrPrgPtr »Page 51 | 1 | Pointer to USER Program |
| $0115 | UsrSavPtr »Page 52 | 1 | Pointer to USER Save Program |
| $0116-$01fe | UsrEEPROM | 233 | User EEPROM Start with address table |
| $01ff | EE_EXOR | 1 | Checksum |

## OptionReg

Bits 0 and 1 of this byte are used by the processor-hardware (see data-sheet).

| **Bit #** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | must all be 1 | | | | | | EE1P (MC68HC05-specific)<br>1 = EEPROM 120H-1FFH writable<br>0 = EEPROM 120H-1FFH protected | SEC (MC68HC05-specific) |

## ManData

Manufacturing data from manufacturer of BCU.

**This data must not be changed !**

## Manufact

Manufacturer code of the manufacturer providing the EEPROM-software.

Manufacturer code table : (see appendix A)

## DevTyp

Each manufacturer may have its own device code table. But unique numbering within EIBA is preferable.

Device code table (hex) :

0000 reserved (esc)
00001 ... (defined by each manufacturer)

## Version

Version of the EEPROM-software.

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | Main Version Number (0-F) | | | | Sub Version Number (0-F) | | | |

## CheckLim

This byte defines the area of EEPROM which is secured by the checksum.

Check range : 108 to (CheckLim-1)
Legal values : 09H to FFH

If an object value is in EEPROM, then it must be outside the check-range.

## PEI-Type

This is the PEI-type which is required by the software in EEPROM.

If this PEI-type matches not the one read from the PEI, then the USER-program is stopped.

## SyncRate

This byte specifies the baud rate used for the serial synchronous PEI.

For the data format and interpretation see MC68HC05B6-data-sheet: baud rate register.

The crystal-frequency of the BCU is 4.0MHz.

## PortCDDR

This byte contains the direction bit setting for the port C. (Bit 0-7= PortC 0-7, 0 = Input , 1 = Output

This value is used only if the PEI-Type 17 or 10 is used and the EEPROM is assumed to be ok.

***Attention : This feature is a potential source of incompatibility!***
***Try to avoid to use this feature !***

# PortADDR

This byte contains the direction bit setting for the port A. (Bit 0-7= PortA 0-7, 0 = Input , 1 = Output)

This value is used only if the EEPROM is assumed to be ok. Otherwise the port A is set to input.

This byte will no longer be programmed by future ETS versions. So take care to set it to the correct value during your manufacturing process. The ETS will use it to verify whether the application program to be loaded matches the hardware environment.

# RunError

In this set of flags, runtime-errors are stored for error analysis purposes.

**A flag is set if the corresponding bit = 0.**

The flags are only set by the system.
They must be cleared explicitly by some management-tool.

| **Bit #** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | <reserved> must be 1 | SYS3_ERR | SYS2_ERR | OBJ_ERR | STK_OVL | EEPROM_ERR | SYS1_ERR | SYS0_ERR |

| | |
|---|---|
| **SYS0_ERR** | Internal system error. |
| **SYS1_ERR** | Internal system error. |
| **EEPROM_ERR** | The EEPROM-check detected an error. The EEPROM-data are probably corrupted.<br>This error inhibits user-execution. |
| **STK_OVL** | A stack overflow was detected. |
| **OBJ_ERR** | The AL detected an error in the communication-object- or association-table.<br>Probably due to inconsistent EEPROM-data.<br>This error inhibits user-execution. |
| **SYS2_ERR** | Internal system error. |
| **SYS3_ERR** | Internal system error.<br>Probably due to inconsistent EEPROM-data. |

## RouteCnt

This byte contains the constant start value for the routing counter.

The range of legal values is 0-7.

| **Bit #** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | must be 0 (*) | | route count | | must be 0 | must be 0 | must be 0 | must be 0 |

(*) Starting with BCU version 1.2 the automatic call of the function U_DELMSG can be enabled by setting this bit to 1. This function is called just before the user main routine.

## MxRstCnt

This byte contains the repeat limits for both the retransmissions due to transmission errors (INAK) and due to busy devices (BUSY).

| **Bit #** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | | BUSY-Retransmit-Limit | | must be 0 | must be 0 | | INAK-Retransmit-Limit | |

## ConfigDes

Some optional features of the system-software can be selected in this byte.

| Bit # | 7 | 6 | 5 | 4 |
|---|---|---|---|---|
| Meaning | must be 1 | A_EVENT-message-generation 0 = enabled<br>1 = disabled | must be 1 | must be 0 |
| Bit # | 3 | 2 | 1 | 0 |
| Meaning | telegram rate limitation(limit at 100H+[UsrInitPtr]-1) 0 = enabled<br>1 = disabled | CPOL clockphase for serial synchronous interface(see MC68HC05B6 data sheet) | CPHA clockphase for serial synchronous interface(see MC68HC05B6 data sheet) | auto. PLMA clear<br>0 = enabled<br>1 = disabled |

***Notes :***

***If the telegram-rate-limitation is enabled, the application-layer generates not more than the specified number (1-127) of group telegrams in about 17 seconds. A side effect of this feature is, that no group telegrams are generated within the first 17 seconds after restart.***

# AssocTabPtr

AssocTabPtr is a pointer to the association table which starts at 100H+[AssocTabPtr].
This table associates the connection numbers with the communication objects.

The first entries (as many as there are communication objects) in the association table may be used for sending. In such a case, they must be sorted in such a way that the communication object number is equal the used association number.

**Association Table**

| Association # | | low memory |
|---|---|---|
| | Length (1 byte) | number of associations in the table |
| 0 | Association | |
| 1 | Association | |
| 2 | Association | |
| ... | ... | high memory |

**Association**

| Connection # (1 byte) | Communication Object # (1 byte) |
|---|---|
| low memory | high memory |

# CommsTabPtr

CommsTabPtr is a pointer to the table of communication objects.
The table starts at 100H+[CommsTabPtr].

**Communication Objects Table**

| Comms # | | |
|---|---|---|
| | Object Count (1 byte) | (low memory) |
| | RAM-Flag-Table-Ptr (1 byte) | |
| 0 | Object-Descriptor (3 bytes) | |
| 1 | Object-Descriptor (3 bytes) | |
| ... | ... | (high memory) |

**Object-Descriptor**

| | | |
|---|---|---|
| Byte 0 | Data-Pointer (offset to memory segment) H-DFH) or the EEPROM (21H-FEH). | (low memory) |
| Byte 1 | Config-Byte | |
| Byte 2 | Type-Byte | (high memory) |

**Config-Byte**

| **Bit #** | **7** | **6** | **5** | **4** |
|---|---|---|---|---|
| **Meaning** | must be 1 | transmit enable<br>1 = enable<br>0 = disable | memory segment select<br>0 = 0 (RAM)<br>1 = 100h (EEPROM) | write enable<br>1 = enabled<br>0 = disabled |
| **Bit #** | **3** | **2** | **1** | **0** |
| **Meaning** | read enable<br>1 = enabled<br>0 = disabled | comm. enable<br>1 = enabled<br>0 = disabled | transmission priority<br>00 reserved (system priority)<br>01 alarm priority<br>10 high operational priority<br>11 low operational priority | |

**Type-Byte**

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | must be 0 | must be 0 | value field type | | | | | |

**Value field types**

| Code | Symbol | value-size |
|---|---|---|
| 0 | UINT1 | 1 bit |
| 1 | UINT2 | 2 bit |
| 2 | UINT3 | 3 bit |
| 3 | UINT4 | 4 bit |
| 4 | UINT5 | 5 bit |
| 5 | UINT6 | 6 bit |
| 6 | UINT7 | 7 bit |
| 7 | UINT8 | 1 byte |
| 8 | UINT16 | 2 byte |
| 9 | BYTE3 | 3 byte |
| 10 | FLOAT | 4 byte |
| 11 | DATA6 | 6 byte |
| 12 | DOUBLE | 8 byte |
| 13 | DATA10 | 10 byte |
| 14 | MAXDATA | 14 byte |
| 15 | VARDATA | 1-14 bytes |

Note: The value field size VARDATA(15) may only be used if values of variable length must be transmitted. This may be the case if larger amounts of data must be transmitted via group addresses and a compensation for varying transmission rates, e.g. due to varying bus loads, is required.

The data should contain a length indicator. The size of the value field must be sufficient to hold the maximum value of 14 bytes.

The value field size VARDATA is not supported by the BCU versions 1.2 and below.

**RAM-Flag-Table**

| | | | |
|---|---|---|---|
| Byte 0 | Comms-Flag 1 | Comms-Flag 0 | (low memory) |
| Byte 1 | Comms-Flag 3 | Comms-Flag 2 | |
| ... | ... | ... | (high memory) |

**Comms-Flags**

| **Bit #** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|
| **Meaning** | update-flag (external)<br>0 = not updated<br>1 = updated | data-request-flag<br>0 = idle/response (arrived)<br>1 = data-request | transmission status<br>00 idle/OK<br>01 idle/error<br>10 transmitting<br>11 transmit-request | |

The communication object number is implicitly given by its index in the table.

The connection number (for sending) is also given implicitly by the association with the same number. E.g. communication object 5 uses by default association 5.

The data-pointer points to the beginning of the value (MSB), which always starts at a byte-boundary. The value is right adjusted and not used bits must be set to zero.

## UsrInitPtr

UsrInitPtr is a pointer to the entry point of the USER-initialization-routine in EEPROM.
The initialization-routine starts at 100H+[UsrInitPtr].

The initialization-routine is called once at user-startup-time.

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

## UsrPrgPtr

UsrPrgPtr is a pointer to the entry point of the USER-program in EEPROM.
The USER-program starts at 100H+[UsrPrgPtr].

The USER-program is called periodically if the BCU is in normal operation mode.

The USER-program must be written as a subroutine, i.e. it must be terminated by "rts".

## UsrSavPtr

UsrSavPtr is a pointer to the entry point of the USER-Save-program in EEPROM.

The USER-Save-program starts at 100H+[UsrSavPtr].

The USER-Save-program is called if the save-signal is **generated due to supply-power-breakdown** and the user is active at the same time.

After calling the USER-Save-program the BCU is reset.

The USER-Save-program must be written as a subroutine, i.e. it must be terminated by "rts".

## AdrTab

AdrTab is the start address of the address table which is specified below. The address table contains as first entry the physical address of the device. The following addresses are the different group addresses used by the device.

The search algorithm in the BCU requires a sorted list of group addresses. The group addresses must be filled in **in ascending order**.

**Address Table**

| Connection # | | |
|---|---|---|
| | Length<br>(1 byte) | (low memory) |
| 0 | physical address<br>(2 bytes) | |
| 1 | group address 1<br>(2 bytes) | |
| 2 | group address 2<br>(2 bytes) | |
| ... | ... | (high memory) |

Length" is the number of entries in the address table.

If there are no group addresses (physical address only), then the length is 1.

If the length is 0, then all group-addresses are accepted.

In the case of EEPROM- or communication-object-errors, the length is set to 1.

**Physical Address**

| **Bit #** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | Area Address | | | | Branch Address | | | |
| **Bit #** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Meaning** | Device Address | | | | | | | |

**Group Address**H is reserved as the broadcast address.

| **Bit #** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Meaning** | Group Address | | | | | | | | | | | | | | | |

The implementation of group addresses ex factory is explained in chapter Interworking.

## EE_EXOR

This byte contains the checksum over the specified part of the EEPROM.

# SyncRate

This byte specifies the baud rate used for the serial synchronous PEI type 12h and 14h

| SyncRate | BCU 1.x | BCU 2 |
|---|---|---|
| 00h | 125 000 | 153 000 |
| 40h | 41 666 | 51 200 |
| 02h | - | 38 400 |
| 03h | - | 19 200 |
| C0h | 9 600 | 9 600 |
| C9h | 4 800 | 4 800 |
| D2h | 2 400 | 2 400 |
| Dbh | 1 200 | 1 200 |
| E4h | 600 | 600 |
| EDh | 300 | 300 |

# RAM

# Memory Map

| RAM Address | Name | Length (bytes) | Commet |
|---|---|---|---|
| 0x0050 | RegB | 1 | Register B |
| 0x0051 | RegC | 1 | Register C |
| 0x0052 | RegD | 1 | Register D |
| 0x0053 | RegE | 1 | Register E |
| 0x0054 | RegF | 1 | Register F |
| 0x0055 | RegG | 1 | Register G |
| 0x0056 | RegH | 1 | Register H |
| 0x0057 | RegI | 1 | Register I |
| 0x0058 | RegJ | 1 | Register J |
| 0x0059 | RegK | 1 | Register K |
| 0x005A | RegL | 1 | Register L |
| 0x005B | RegM | 1 | Register M |
| 0x005C | RegN | 1 | Register N |
| 0x005D<br>0x005F | reserved | 3 | System Software |
| 0x0060 | SystemState<br>»Page 56 | 1 | State for each layer |
| 0x0061<br>0x00CD | reserved | 113 | System Software |
| 0x00CE<br>0x00DF | UserRAM | 18 | RAM-Area for Comms-Object- and Application-Data |
| 0x00E0<br>0x00FF | reserved | 32 | System Software and Stack Space |

All variables called "register" can be used as temporary RAM-storage.

***These variables may also be used by ROM-functions and for parameter passing or as temporary variables. See function descriptions !***

A variable at address 60h (BCU RAM) is used to specify the operation mode of the BCU :

## System State

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| PARIT Y | DM | UE | SE | ALE | TLE | LLM | PROG |

**PROG**

If this bit is set, the BCU is in programming-mode else it is in normal operation-mode.

**LLM**

If this bit is set, the link-layer is in normal operation-mode else it is in busmonitor-mode.

**TLE**

If this bit is set, the transport-layer is enabled else it is disabled.

**ALE**

If this bit is set, the application-layer is enabled else it is disabled.

**SE**

If this bit is set, the serial PEI-interface (message-protocol) is enabled else it is disabled.

**UE**

If this bit is set, the user program is enabled, else it is disabled.

**DM**

If this bit is set, the BCU is in download-mode else it is in normal operation-mode

**PARITY**

This bit is the parity-bit for this byte. Even parity is used.

To activate a layer, a user must write at address 60h in the BCU's RAM the legal value :

| **Mode** | **Value** |
|---|---|
| Busmonitor | 90h |
| Link-Layer | 12h |
| Transport-Layer | 96h |
| Application-Layer | 1Eh |
| Reset | COh |

After a reset, the Application-Layer is always activated.

# Physical External Interface

The electrical and logical properties of the PEI connector lines depend on the PEI type given by the periphery module. For the various applications, different functional types of physical external interfaces are available. The PEI lines for 24V (pin 8), 5V (pin 5), ground (pin 1 and pin 10) and PEI type selection (pin 6) are the same for all PEI types.

The following figure shows the logical specification of the other PEI lines, dependent on the PEI type supported by System BCU 1.

| PEI-Type | Functional description | PEI pin 2<br>I/O 2<br>RxD | PEI pin 3<br>I/O 1<br>SCLK<br>(syn) | PEI pin 4<br>I/O 3<br>TxD | PEI pin 5a<br>PWM2<br>- | PEI pin 6a<br>I/O6<br>- | PEI pin 7<br>I/O 4<br>CTS | PEI pin 9<br>I/O 5<br>RTS |
|---|---|---|---|---|---|---|---|---|
| 0 | No adaptor | | | | | | | |
| 1 | illegal adaptor | | | | | | | |
| 2 | 4 inputs, 1 output (LED) | INPUT | INPUT | INPUT | OUTPUT | OUTPUT | INPUT | OUTPUT |
| 3 | reserved | | | | | | | |
| 4 | 2 inputs & 2 outputs +<br>1 output (LED) | OUTPUT | OUTPUT | INPUT | OUTPUT | OUTPUT | INPUT | OUTPUT |
| 5 | reserved | | | | | | | |
| 6 | 3 inputs & 1 output<br>+1 output (LED) | INPUT | OUTPUT | INPUT | OUTPUT | OUTPUT | INPUT | OUTPUT |
| 7 | reserved | | | | | | | |
| 8 | 5 inputs | INPUT | INPUT | INPUT | OUTPUT | OUTPUT | INPUT | INPUT |
| 9 | reserved | | | | | | | |
| 10 | reserved | | | | | | | |
| 11 | reserved | | | | | | | |
| 12 | serial synchronous<br>interface<br>message protocol | ser. input<br>RDI | output:<br>SCLK | ser. output:<br>TDO | OUTPUT | OUTPUT | CTS | RTS |
| 13 | reserved | | | | | | | |
| 14 | Serial synchronous<br>interface<br>data block protocol | ser.input:<br>RDI | output:<br>SCLK | ser. output:<br>TDO | OUTPUT | OUTPUT | CTS | RTS |
| 15 | reserved | | | | | | | |
| 16 | Serial asynchronous<br>interface,<br>message protocol | ser.input:<br>RxD | OUTPUT | ser. output:<br>TxD | OUTPUT | OUTPUT | CTS | RTS |
| 17 | programmable I/O | def. by<br>user | def. by<br>user | def. by<br>user | OUTPUT | OUTPUT | def. by<br>user | def. by<br>user |
| 18 | reserved | | | | | | | |
| 19 | 4 outputs, 1 output(LED) | OUTPUT | OUTPUT | OUTPUT | OUTPUT | OUTPUT | OUTPUT | OUTPUT |
| 20 | Download | ser. input:<br>RxD | OUTPUT | ser. output:<br>TxD | OUTPUT | OUTPUT | CTS | RTS |

For serial PEI See Desricption of External Message Interface »Page 58

# External Message Interface

For internal and external purposes each service primitive corresponds to a unique message code. The following table summarizes the names and values of the internal constants at the BCU 1.

| **Service primitive name** | **name of the correspon-ding constant** | **message code** | **desti-nation (layer)** |
|---|---|---|---|
| Link Layer | | | |
| L_DATA.req | L_DATA_requ | 11 | LL |
| L_DATA.ind | L_DATA_ind | 49 | TL |
| L_DATA.con | L_DATA_conf | 4E | TL |
| L_BUSMON.ind | L_BUSMON_ind | 49 | PEI |
| Transport Layer | | | |
| T_GROUPDATA.req | T_GROUP_DATA_requ | 22 | TL grp |
| T_GROUPDATA.ind | T_GROUP_DATA_ind | 4A | AL grp |
| T_GROUPDATA.con | T_GROUP_DATA_conf | 4E | AL grp |
| T_DATA.req | T_DATA_requ | 21 | TL co |
| T_DATA.ind | T_DATA_ind | 49 | AL co |
| T_CONNECT.req | T_CONNECT_requ | 23 | TL co |
| T_CONNECT.ind | T_CONNECT_ind | 43 | PEI |
| T_DISCONNECT.req | T_DISCONNECT_requ | 24 | TL co |
| T_DISCONNECT.ind | T_DISCONNECT_ind | 44 | PEI |
| Application Layer | | | |
| A_USER_DATA.req | M_USER_DATA_requ | 31 | AL co |
| A_USER_DATA.ind | M_USER_DATA_ind | 49 | User |
| User Layer | | | |
| U_VALUE_READ.req | A_VALUE_READ_req | 35 | User |
| U_VALUE_READ.con | A_VALUE_READ_con | 45 | User |
| U_VALUE_WRITE.req | A_VALUE_WRITE_req | 36 | User |
| U_EVENT.ind | A_EVENT_ind | 4D | User |
| U_FLAGS_READ.req | A_FLAGS_READ_req | 37 | User |
| U_FLAGS_READ.con | A_FLAGS_READ_con | 47 | User |
| PEI-Services | | | |
| PC_SET_VAL.req | PC_SET_VAL_req | | PEI |
| PC_GET_VAL.req | PC_GET_VAL_req | | PEI |
| PC_GET_VAL.con | PC_GET_VAL_con | | User |

***Table B-2: BCU 1 service primitives and message codes***

Key for the destination layer column of table B-2:

| | | |
|---|---|---|
| LL, TL, AL, PEI, User: | layers, respectively the user application | |
| | grp | = message-oriented application layer |
| | cl | = connectionless application layer |
| | co | = connection-oriented application layer |

*Notes: The contents of Row B has yet to be defined.*

*Line coupler services are not shown. They should be regarded as summarized by the T_Data service.*

*The A_USER_DATA service is a solely a peer-to-peer service.*

*Message codes 0F0h to 0FFh are reserved for OEM purposes, i.e. for private communication via the PEI between the external application(s) and the internal application.*

The following table B-3 shows all the message codes in a two-dimensional table. The message codes consist of 2 parts. The first part (high nibble) is the number of the destination module, the second part (low nibble) is a independent number.

Known Problems

???....

For BCU 2 Timer System see Helpfile of BCU 2

# BCU 1 Communication Objects

**Use this functions only if System BCU 1 Communication Objects are used (standard callback is active).**

# BCU 1 Checksum

**Only used if old load procedure is used.**

# BCU 1 Usertimer pointer

**If new load procedure is used the pointer to timer description Block is located at UserPrgPtr-1. Possible not supported in newer versions of BCU 2.**

# BCU 1 Delay

**The delay is not in 0,5 ms steps it is 0,416 ms.**

# BCU 1 PopBuf

**With new load procedure use External message interface of BCU 2 else use EMI of BCU 1.**

**Namensnennung — Nicht-kommerziell — Keine Bearbeitung 2.0**

CREATIVE COMMONS IST KEINE RECHTSANWALTSGESELLSCHAFT UND LEISTET KEINE RECHTSBERATUNG. DIE WEITERGABE DIESES LIZENZENTWURFES FÜHRT ZU KEINEM MANDATSVERHÄLTNIS. CREATIVE COMMONS ERBRINGT DIESE INFORMATIONEN OHNE GEWÄHR. CREATIVE COMMONS ÜBERNIMMT KEINE GEWÄHRLEISTUNG FÜR DIE GELIEFERTEN INFORMATIONEN UND SCHLIEßT DIE HAFTUNG FÜR SCHÄDEN AUS, DIE SICH AUS IHREM GEBRAUCH ERGEBEN.

*Lizenzvertrag*

DAS URHEBERRECHTLICH GESCHÜTZTE WERK ODER DER SONSTIGE SCHUTZGEGENSTAND (WIE UNTEN BESCHRIEBEN) WIRD UNTER DEN BEDINGUNGEN DIESER CREATIVE COMMONS PUBLIC LICENSE („CCPL“ ODER „LIZENZVERTRAG“) ZUR VERFÜGUNG GESTELLT. DER SCHUTZGEGENSTAND IST DURCH DAS URHEBERRECHT UND/ODER EINSCHLÄGIGE GESETZE GESCHÜTZT.

DURCH DIE AUSÜBUNG EINES DURCH DIESEN LIZENZVERTRAG GEWÄHRTEN RECHTS AN DEM SCHUTZGEGENSTAND ERKLÄREN SIE SICH MIT DEN LIZENZBEDINGUNGEN RECHTSVERBINDLICH EINVERSTANDEN. DER LIZENZGEBER RÄUMT IHNEN DIE HIER BESCHRIEBENEN RECHTE UNTER DER VORAUSSETZUNGEIN, DASS SIE SICH MIT DIESEN VERTRAGSBEDINGUNGEN EINVERSTANDEN ERKLÄREN.

**1. Definitionen**

a. Unter einer **„Bearbeitung“** wird eine Übersetzung oder andere Bearbeitung des Werkes verstanden, die Ihre persönliche geistige Schöpfung ist. Eine freie Benutzung des Werkes wird nicht als Bearbeitung angesehen.

b. Unter den **„Lizenzelementen“** werden die folgenden Lizenzcharakteristika verstanden, die vom Lizenzgeber ausgewählt und in der Bezeichnung der Lizenz genannt werden: „Namensnennung“, „Nicht-kommerziell“, „Weitergabe unter gleichen Bedingungen“.

c. Unter dem **„Lizenzgeber“** wird die natürliche oder juristische Person verstanden, die den Schutzgegenstand unter den Bedingungen dieser Lizenz anbietet.

d. Unter einem **„Sammelwerk“** wird eine Sammlung von Werken, Daten oder anderen unabhängigen Elementen verstanden, die aufgrund der Auswahl oder Anordnung der Elemente eine persönliche geistige Schöpfung ist. Darunter fallen auch solche Sammelwerke, deren Elemente systematisch oder methodisch angeordnet und einzeln mit Hilfe elektronischer Mittel oder auf andere Weise zugänglich sind (Datenbankwerke). Ein Sammelwerk wird im Zusammenhang mit dieser Lizenz nicht als Bearbeitung (wie oben beschrieben) angesehen.

e. Mit **„SIE“** und **„Ihnen“** ist die natürliche oder juristische Person gemeint, die die durch diese Lizenz gewährten Nutzungsrechte ausübt und die zuvor die Bedingungen dieser Lizenz im Hinblick auf das Werk nicht verletzt hat, oder die die ausdrückliche Erlaubnis des Lizenzgebers erhalten hat, die durch diese Lizenz gewährten Nutzungsrechte trotz einer vorherigen Verletzung auszuüben.

f. Unter dem **„Schutzgegenstand“**wird das Werk oder Sammelwerk oder das Schutzobjekt eines verwandten Schutzrechts, das Ihnen unter den Bedingungen dieser Lizenz angeboten wird, verstanden

g. Unter dem **„Urheber“** wird die natürliche Person verstanden, die das Werk geschaffen hat.

h. Unter einem **„verwandten Schutzrecht“** wird das Recht an einem anderen urheberrechtlichen Schutzgegenstand als einem Werk verstanden, zum Beispiel einer wissenschaftlichen Ausgabe, einem nachgelassenen Werk, einem Lichtbild, einer Datenbank, einem Tonträger, einer Funksendung, einem Laufbild oder einer Darbietung eines ausübenden Künstlers.

i. Unter dem **„Werk“** wird eine persönliche geistige Schöpfung verstanden, die Ihnen unter den Bedingungen dieser Lizenz angeboten wird.

**2. Schranken des Urheberrechts.** Diese Lizenz lässt sämtliche Befugnisse unberührt, die sich aus den Schranken des Urheberrechts,aus dem Erschöpfungsgrundsatz oder anderen Beschränkungen der Ausschließlichkeitsrechte des Rechtsinhabers ergeben.

**3. Lizenzierung.** Unter den Bedingungen dieses Lizenzvertrages räumt Ihnen der Lizenzgeber ein lizenzgebührenfreies, räumlich und zeitlich (für die Dauer des Urheberrechts oder verwandten Schutzrechts) unbeschränktes einfaches Nutzungsrecht ein, den Schutzgegenstand in der folgenden Art und Weise zu nutzen:

a. den Schutzgegenstand in körperlicher Form zu verwerten, insbesondere zu vervielfältigen, zu verbreiten und auszustellen;

b. den Schutzgegenstand in unkörperlicher Form öffentlich wiederzugeben, insbesondere vorzutragen, aufzuführen und vorzuführen, öffentlich zugänglich zu machen, zu senden, durch Bild- und Tonträger wiederzugeben sowie Funksendungen und öffentliche Zugänglichmachungen wiederzugeben;

c. den Schutzgegenstand auf Bild- oder Tonträger aufzunehmen, Lichtbilder davon herzustellen, weiterzusenden und in dem in a. und b. genannten Umfang zu verwerten;

Die genannten Nutzungsrechte können für alle bekannten Nutzungsarten ausgeübt werden. Die genannten Nutzungsrechte beinhalten das Recht, solche Veränderungen an dem Werk vorzunehmen, die technisch erforderlich sind, um die Nutzungsrechte für alle Nutzungsarten wahrzunehmen. Insbesondere sind davon die Anpassung an andere Medien und auf andere Dateiformate umfasst.

**4. Beschränkungen.** Die Einräumung der Nutzungsrechte gemäß Ziffer 3 erfolgt ausdrücklich nur unter den folgenden Bedingungen:

a. Sie dürfen den Schutzgegenstand ausschließlich unter den Bedingungen dieser Lizenz vervielfältigen, verbreiten oder öffentlich wiedergeben, und Sie müssen stets eine Kopie oder die vollständige Internetadresse in Form des Uniform-Resource-Identifier (URI) dieser Lizenz beifügen, wenn Sie den Schutzgegenstandvervielfältigen, verbreiten oder öffentlich wiedergeben. Sie dürfen keine Vertragsbedingungen anbieten oder fordern, die die Bedingungen dieser Lizenz oder die durch sie gewährten Rechte ändern oder beschränken. Sie dürfen den Schutzgegenstand nicht unterlizenzieren. Sie müssen alle Hinweise unverändert lassen, die auf diese Lizenz und den Haftungsausschluss hinweisen. Sie dürfen den Schutzgegenstand mit keinen technischen Schutzmaßnahmen versehen, die den Zugang oder den Gebrauch des Schutzgegenstandes in einer Weise kontrollieren, die mit den Bedingungen dieser Lizenz im Widerspruch stehen. Die genannten Beschränkungen gelten auch für den Fall, dass der Schutzgegenstand einen Bestandteil eines Sammelwerkes bildet; sie verlangen aber nicht, dass das Sammelwerk insgesamt zum Gegenstand dieser Lizenz gemacht wird. Wenn Sie ein Sammelwerk erstellen, müssen Sie - soweit dies praktikabel ist - auf die Mitteilung eines Lizenzgebers oder Urhebers hin aus dem Sammelwerk jeglichen Hinweis auf diesen Lizenzgeber oder diesen Urheber entfernen. Wenn Sie den Schutzgegenstand bearbeiten, müssen Sie - soweit dies praktikabel ist- auf die Aufforderung eines Rechtsinhabers hin von der Bearbeitung jeglichen Hinweis auf diesen Rechtsinhaber entfernen.

b. Sie dürfen die in Ziffer 3 gewährten Nutzungsrechte in keiner Weise verwenden, die hauptsächlich auf einen geschäftlichen Vorteil oder eine vertraglich geschuldete geldwerte Vergütung abzielt oder darauf gerichtet ist. Erhalten Sie im Zusammenhang mit der Einräumung der Nutzungsrechte ebenfalls einen Schutzgegenstand, ohne dass eine vertragliche Verpflichtung hierzu besteht, so wird dies nicht als geschäftlicher Vorteil oder vertraglich geschuldete geldwerte Vergütung angesehen, wenn keine Zahlung oder geldwerte Vergütung in Verbindung mit dem Austausch der Schutzgegenstände geleistet wird (z.B. File-Sharing).

c. Wenn Sie den Schutzgegenstand oder ein Sammelwerk vervielfältigen, verbreiten oder öffentlich wiedergeben, müssen Sie alle Urhebervermerke für den Schutzgegenstand unverändert lassen und die Urheberschaft oder Rechtsinhaberschaft in einer der von Ihnen vorgenommenen Nutzung angemessenen Form anerkennen, indem Sie den Namen (oder das Pseudonym, falls ein solches verwendet wird) des Urhebers oder Rechteinhabers nennen, wenn dieser angegeben ist. Dies gilt auch für den Titel des Schutzgegenstandes, wenn dieser angeben ist, sowie - in einem vernünftigerweise durchführbaren Umfang - für die mit dem Schutzgegenstand zu verbindende Internetadresse in Form des Uniform-Resource-Identifier (URI), wie sie der Lizenzgeber angegeben hat, sofern dies geschehen ist, es sei denn, diese Internetadresse verweist nicht auf den Urhebervermerk oder die Lizenzinformationen zu dem Schutzgegenstand. Ein solcher Hinweis kann in jeder angemessenen Weise erfolgen, wobei jedoch bei einer Datenbank oder einem Sammelwerk der Hinweis zumindest an gleicher Stelle und in ebenso auffälliger Weise zu erfolgen hat wie vergleichbare Hinweise auf andere Rechtsinhaber.

d. Obwohl die gemäss Ziffer 3 gewährten Nutzungsrechte in umfassender Weise ausgeübt werden dürfen, findet diese Erlaubnis ihre gesetzliche Grenze in den Persönlichkeitsrechten der Urheber und ausübenden Künstler, deren berechtigte geistige und persönliche Interessen bzw. deren Ansehen oder Ruf nicht dadurch gefährdet werden dürfen, dass ein Schutzgegenstand über das gesetzlich zulässige Maß hinaus beeinträchtigt wird.

**5. Gewährleistung.** Sofern dies von den Vertragsparteien nicht anderweitig schriftlich vereinbart,, bietet der Lizenzgeber keine Gewährleistung für die erteilten Rechte, außer für den Fall, dass Mängel arglistig verschwiegen wurden. Für Mängel anderer Art, insbesondere bei der mangelhaften Lieferung von Verkörperungen des Schutzgegenstandes, richtet sich die Gewährleistung nach der Regelung, die die Person, die Ihnen den Schutzgegenstand zur Verfügung stellt, mit Ihnen außerhalb dieser Lizenz vereinbart, oder - wenn eine solche Regelung nicht getroffen wurde - nach den gesetzlichen Vorschriften.

**6. Haftung.** Über die in Ziffer 5 genannte Gewährleistung hinaus haftet Ihnen der Lizenzgeber nur für Vorsatz und grobe Fahrlässigkeit.

**7. Vertragsende**

a. Dieser Lizenzvertrag und die durch ihn eingeräumten Nutzungsrechte enden automatisch bei jeder Verletzung der Vertragsbedingungen durch Sie. Für natürliche und juristische Personen, die von Ihnen eine Datenbank oder ein Sammelwerk unter diesen Lizenzbedingungen erhalten haben, gilt die Lizenz jedoch weiter, vorausgesetzt, diese natürlichen oder juristischen Personen erfüllen sämtliche Vertragsbedingungen. Die Ziffern 1, 2, 5, 6, 7 und 8 gelten bei einer Vertragsbeendigung fort.

b. Unter den oben genannten Bedingungen erfolgt die Lizenz auf unbegrenzte Zeit (für die Dauer des Schutzrechts). Dennoch behält sich der Lizenzgeber das Recht vor, den Schutzgegenstand unter anderen Lizenzbedingungen zu nutzen oder die eigene Weitergabe des Schutzgegenstandes jederzeit zu beenden, vorausgesetzt, dass solche

Handlungen nicht dem Widerruf dieser Lizenz dienen (oder jeder anderen Lizenzierung, die auf Grundlage dieser Lizenz erfolgt ist oder erfolgen muss) und diese Lizenz wirksam bleibt, bis Sie unter den oben genannten Voraussetzungen endet.

**8. Schlussbestimmungen**

a. Jedes Mal, wenn Sie den Schutzgegenstand vervielfältigen, verbreiten oder öffentlich wiedergeben, bietet der Lizenzgeber dem Erwerber eine Lizenz für den Schutzgegenstand unter denselben Vertragsbedingungen an, unter denen er Ihnen die Lizenz eingeräumt hat.

b. Sollte eine Bestimmung dieses Lizenzvertrages unwirksam sein, so wird die Wirksamkeit der übrigen Lizenzbestimmungen dadurch nicht berührt, und an die Stelle der unwirksamen Bestimmung tritt eine Ersatzregelung, die dem mit der unwirksamen Bestimmung angestrebten Zweck am nächsten kommt.

c. Nichts soll dahingehend ausgelegt werden, dass auf eine Bestimmung dieses Lizenzvertrages verzichtet oder einer Vertragsverletzung zugestimmt wird, so lange ein solcher Verzicht oder eine solche Zustimmung nicht schriftlich vorliegen und von der verzichtenden oder zustimmenden Vertragspartei unterschrieben sind

d. Dieser Lizenzvertrag stellt die vollständige Vereinbarung zwischen den Vertragsparteien hinsichtlich des Schutzgegenstandes dar. Es gibt keine weiteren ergänzenden Vereinbarungen oder mündlichen Abreden im Hinblick auf den Schutzgegenstand. Der Lizenzgeber ist an keine zusätzlichen Abreden gebunden, die aus irgendeiner Absprache mit Ihnen entstehen könnten. Der Lizenzvertrag kann nicht ohne eine übereinstimmende schriftliche Vereinbarung zwischen dem Lizenzgeber und Ihnen abgeändert werden.

e. Auf diesen Lizenzvertrag findet das Recht der Bundesrepublik Deutschland Anwendung.

CREATIVE COMMONS IST KEINE VERTRAGSPARTEI DIESES LIZENZVERTRAGES UND ÜBERNIMMT KEINERLEI GEWÄHRLEISTUNG FÜR DAS WERK. CREATIVE COMMONS IST IHNEN ODER DRITTEN GEGENÜBER NICHT HAFTBAR FÜR SCHÄDEN JEDWEDER ART. UNGEACHTET DER VORSTEHENDEN ZWEI (2) SÄTZE HAT CREATIVE COMMONS ALL RECHTE UND PFLICHTEN EINES LIZENSGEBERS, WENN SICH CREATIVE COMMONS AUSDRÜCKLICH ALS LIZENZGEBER BEZEICHNET.

AUSSER FÜR DEN BESCHRÄNKTEN ZWECK EINES HINWEISES AN DIE ÖFFENTLICHKEIT, DASS DAS WERK UNTER DER CCPL LIZENSIERT WIRD, DARF KENIE VERTRAGSPARTEI DIE MARKE “CREATIVE COMMONS” ODER EINE ÄHNLICHE MARKE ODER DAS LOGO VON CREATIVE COMMONS OHNE VORHERIGE GENEHMIGUNG VON CREATIVE COMMONS NUTZEN. JEDE GESTATTETE NUTZUNG HAT IN ÜBREEINSTIMMUNG MIT DEN JEWEILS GÜLTIGEN NUTZUNGSBEDINGUNGEN FÜR MARKEN VON CREATIVE COMMONS ZU ERFOLGEN, WIE SIE AUF DER WEBSITE ODER IN ANDERER WEISE AUF ANFRAGE VON ZEIT ZU ZEIT ZUGÄNGLICH GEMACHT WERDEN.

CREATIVE COMMONS KANN UNTER http://creativecommons.org KONTAKTIERT WERDEN.