1. The bootstrap-loader

The SAB-C509-L/-16F includes a special bootstrap-mode, which is activated by setting the PRGEN-Pin at a logic high level during the rising edge of the external RESET# or HWPD# signal (→ PRGEN1 = 1). In this mode the bootstrap-loader, located at the addresses 0000H to 01FFH in the boot-ROM will be executed. Its purpose is to allow the easy and quick programming of the internal XRAM (F400H to FFFFH) via serial interface while the MCU is in-circuit. In this way the customer has a comfortable tool to transfer custom routines to the XRAM, which will program or reprogram the 128 KByte FLASH memory. The serial routines of the bootstrap-loader may be replaced by own custom software or even can be blocked to prevent unauthorized persons from reading out or writing to the FLASH memory. Therefore the bootstrap-loader checks the FLASH memory for existing custom software and executes it.
The bootstrap-loader consists of three functional parts which represent three phases as described below:

- Phase **I**: Check for existing custom software in the FLASH memory and execute it
- Phase **II**: Establish a serial connection and automatically synchronize to the transfer speed (baud rate) of the serial communication partner (host)
- Phase **III**: Perform the serial communication to the host. The host controls the bootstrap-loader by sending special header informations, which select one of four working modes. These modes are:
  - **Mode 0**: Transfer a custom program from the host to the XRAM (F400H - FFFFH). This mode returns to the beginning of phase **III**
  - **Mode 1**: Execute a custom program in the XRAM at any start address from F400H to FFFFH.
  - **Mode 2**: Check the contents of any area of the 128 KByte FLASH memory by calculating a special checksum. This mode returns to the beginning of phase **III**
  - **Mode 3**: Execute a custom program in the FLASH memory at any start address beyond 0200H (at addresses 0000H to 01FFH the boot-ROM is active)

The phases and their connections are illustrated in the figure below:

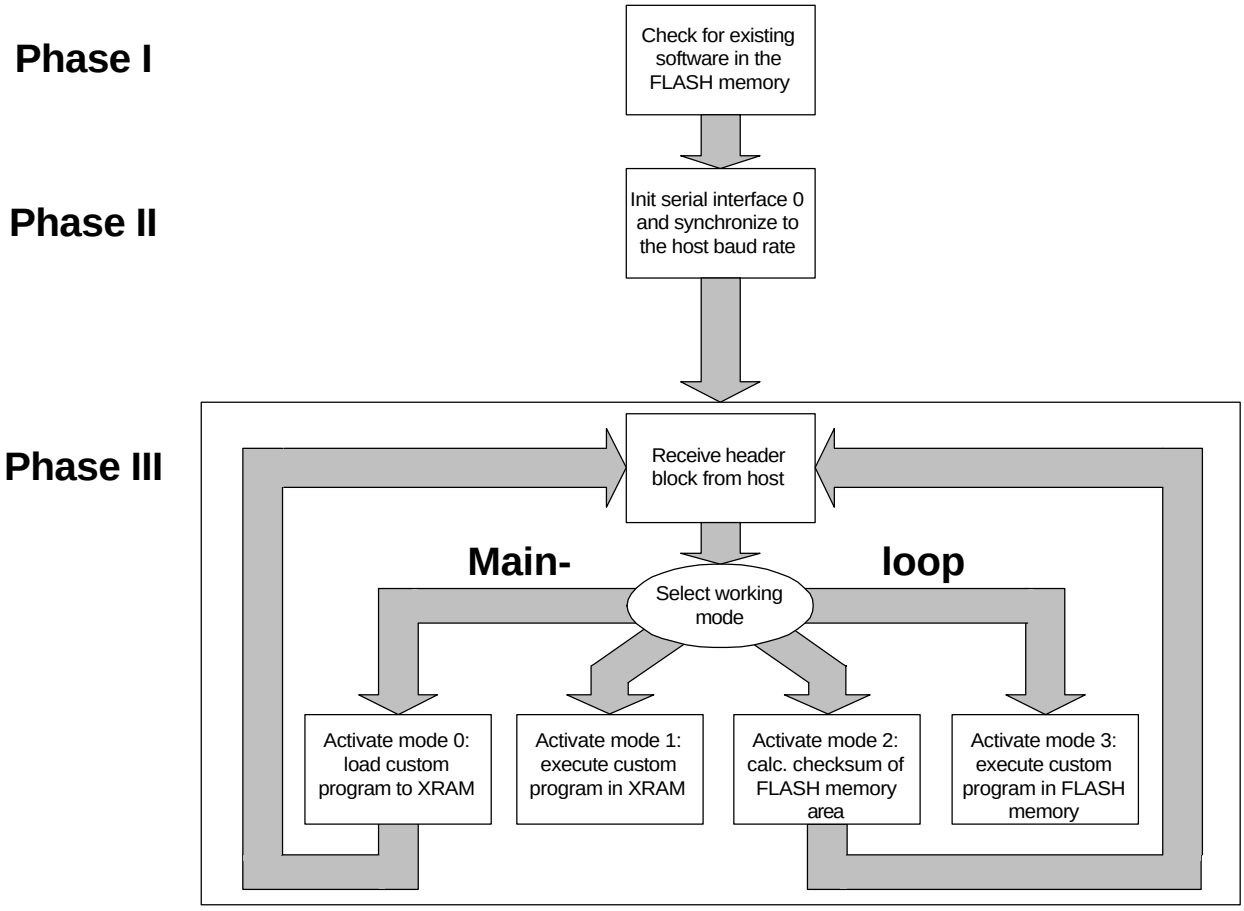

Figure 1: The three phases of the bootstrap-loader

The serial communication, which is activated in phase **II** is performed with the integrated serial interface 0 of the SAB-C509-L/-16F. Using a full- or half-duplex serial cable (RS232) the MCU must be connected to the serial port of the host as shown in the figure below:

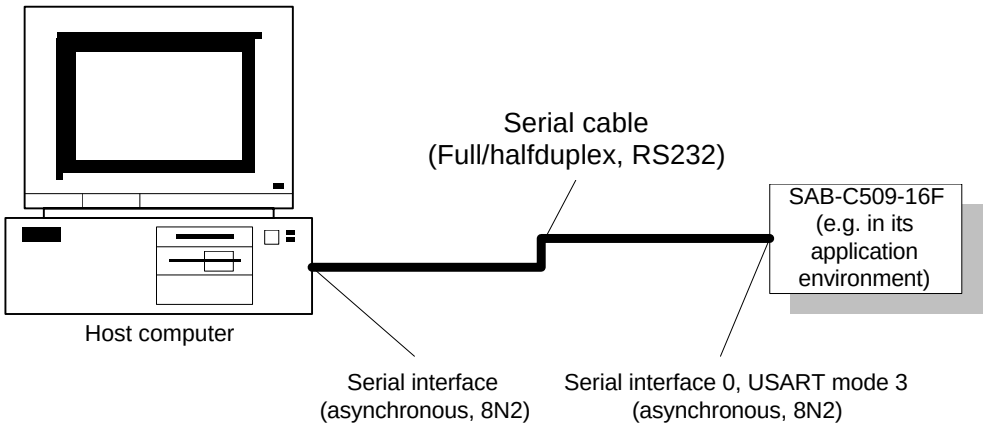

Figure 2: The control of the bootstrap-loader is attained by a host computer connected to the MCU via serial interface

The serial transfer is working in asynchronous mode with the serial parameters 8N2 (eight data bits, no parity and two stop bits). The baud rate however can be varied by the host in a wide range, because the bootstrap-loader does an automatic synchronisation in phase **II**.
The following sections give detailed informations of all three bootstrap-phases.

1.1. Supported interrupts

The bootstrap-loader itself does not use any of the implemented interrupts of the MCU for its work. But for further custom program execution in the bootstrap-mode two interrupts are supported. These are the interrupts for the timer 0 and for the serial interface 0. As the interrupt vectors are located in the address area of the bootstrap-loader (0000H - 01FFH), the appropriate interrupt vector addresses are routed by the bootstrap-loader via the command LJMP to a reserved XRAM area at F400H - F41FH as shown below:

Timer 0 (TIMER 0) usually at 000BH	is routed to F400H in XRAM
Serial interface 0 (SINT 0) usually at 0023H	is routed to F410H in XRAM

At these addresses (F400H - F40FH and F410H - F41FH) the user specified interrupt routines can be loaded to handle the interrupts in a user defined way. To avoid unexpected software behaviour when these interrupts are used, the reserved memory area should be used only by interrupt handling routines. Notice: In this case the XRAM for custom programs reduces to F420H - FFFFH.
If there is no need of these interrupts, the reserved memory area can be programmed with normal custom software.
It is recommended not to activate other interrupts than TIMER 0 and SINT 0, because this could lead to uncontrolled software execution in the interrupt vector area (0000H - 0100H).

1.2. Phase I: Check for existing custom software in the FLASH memory

The first action of the bootstrap-loader is to check two defined 8 KByte blocks in the 128 KByte FLASH memory for functional custom software. If the check is successful, the custom software is started directly, if no software is found, phase **II** is entered, to establish a serial communication with a connected host. This feature can be used to protect the FLASH memory contents against unauthorized reading and writing in the bootstrap-mode. This security check can be used if needed, but can be skipped as well, as described in the next section.

1.2.1. Custom software check by a special info block

For the above mentioned custom software two 8 KByte memory sectors (sector A: E000H to FFFFH, sector B: C000H to DFFFH) of the FLASH memory are assigned. To activate the FLASH memory check, a special info-block is needed, which has to be written at the beginning of the corresponding FLASH memory sector. It has the following structure:

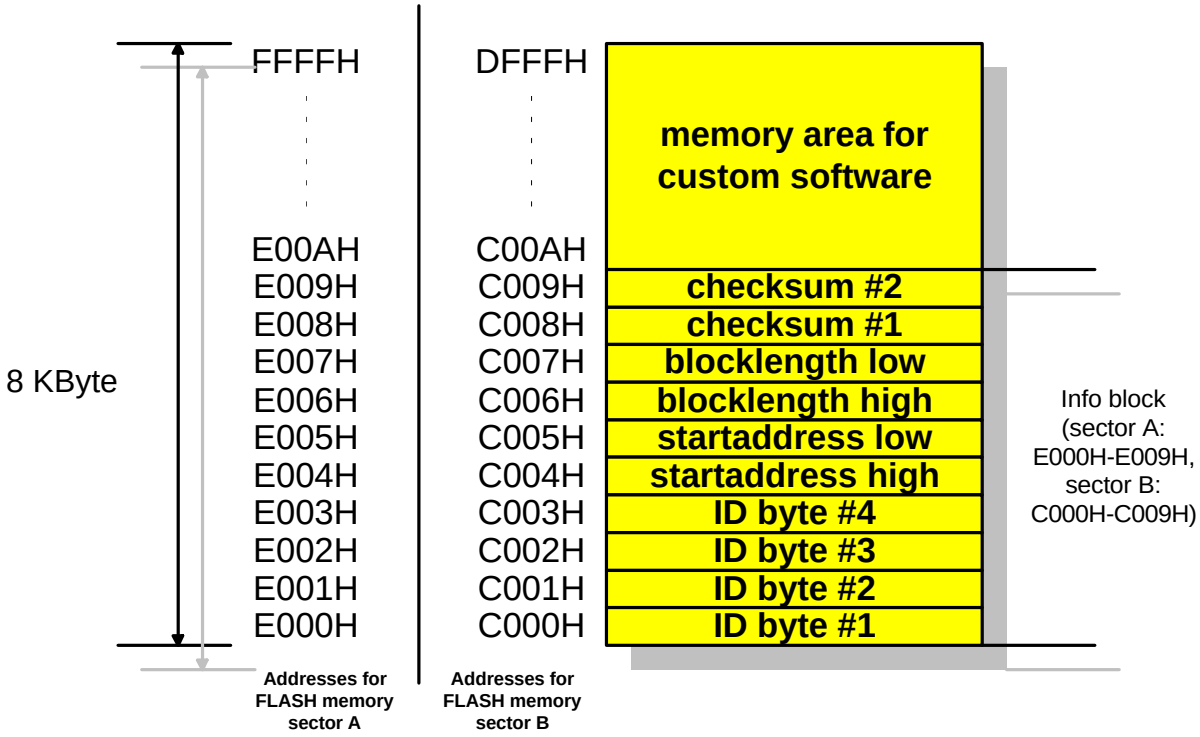

Figure 3: The structure of a info block in the FLASH memory (E000H to FFFFH, C000H to DFFFH)

Description:

ID byte #1 - #4 ..... Four ID bytes which mark the FLASH memory sector as programmed

startaddress (high, low) ..... The start address, where the custom program starts

blocklength (high, low) ..... The length of the memory block, which is filled with the custom program code

checksum #1, #2 ..... The two checksums for the FLASH memory block

memory area for custom software ..... The area for the custom program starting at E00AH in sector A and C00AH in sector B

The info-block starts with a fixed sequence of four identification bytes (**ID bytes**). They mark the corresponding memory sector as custom programmed. If the ID bytes are not present at the beginning of the info block, the bootstrap-loader assumes, that the corresponding sector is not custom programmed.
The ID-bytes must be absolutely definite, to prevent the bootstrap-loader from recognizing normal program code as ID bytes. Therefore the four bytes represent a not senseful command sequence in 8051-code, which should be never occur in normal programs. The bootstrap-loader uses the following ID byte sequence:

ID byte	**Value**	**8051-code**
#4	13H	RRC A
#3	33H	RLC A
#2	03H	RR A
#1	23H	RL A

To check, if the custom software in the relevant sector is functional, the bootstrap-loader calculates a special checksum consisting of two independent checksum bytes over the appropriate memory area. This area begins at the address in **startaddress** and has a length of **blocklength**. The two generated checksums are compared with **checksum #1** and **#2** in the info block. If the checksums are equal, the bootstrap-loader starts the custom routine in this sector at the address **startaddress**.
If either the ID bytes or the checksums are not correct, the checking procedure is done with sector B of the FLASH memory. The check of two sectors (A and B) in the FLASH memory is necessary for a maximum of security, e.g. if the custom software in one sector is not functional. This can happen, if the programming of the corresponding FLASH memory sector suddenly is interrupted by the cause of a power failure.
When the check fails in both sectors, the bootstrap-loader leaves phase **I** and enters phase **II** to establish the serial communication to a connected host via serial interface 0.
If the customer wants not to use this FLASH sector check, he can override it by programming other values than the four ID bytes in the corresponding FLASH memory addresses in sector A and B. If this is done, the remaining bytes of the two FLASH memory sectors may be used for normal program execution as well.

1.2.2. The flowchart of phase I

The following flowchart shows the detailed action of the bootstrap-loader in phase **I**.

Phase I:

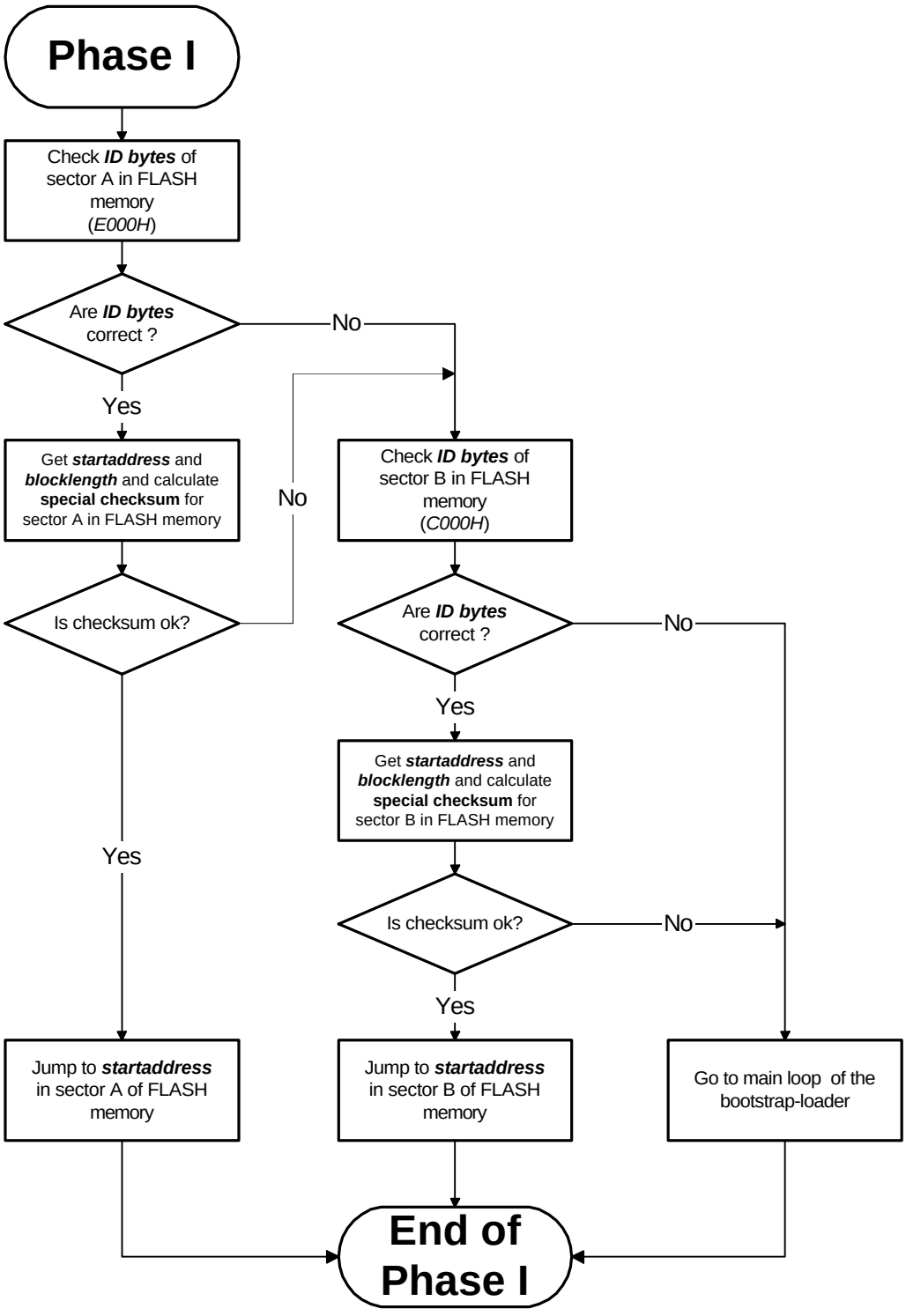

Figure 4: Flowchart of phase I actions

1.2.3. The special checksum calculation

The FLASH memory check in phase **I** uses a special checksum algorithm, which is based on a continuous addition and 8-bit-left-rotation of all relevant data bytes. The data in the corresponding FLASH memory area is split into two parts, on which an extra checksum is built. This is done to reach a maximum of data security. The following figure describes, how the data is split.

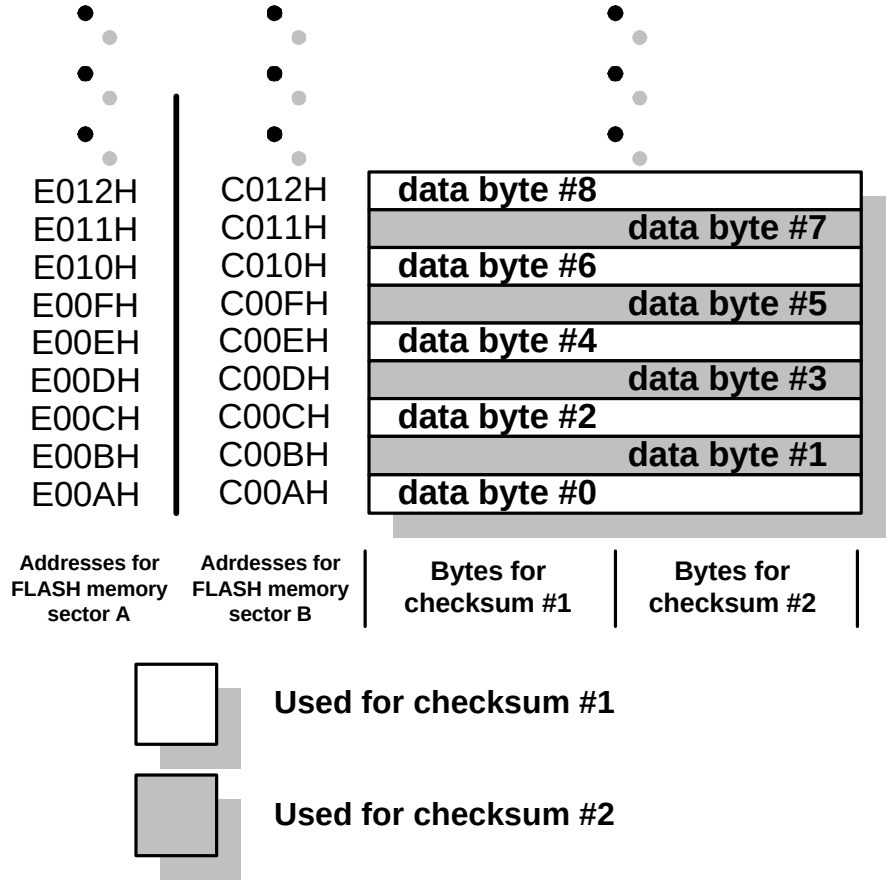

Figure 5: The FLASH memory block is divided into two parts to calculate two independent checksums

The special checksum calculation itself is described in the following flowchart:

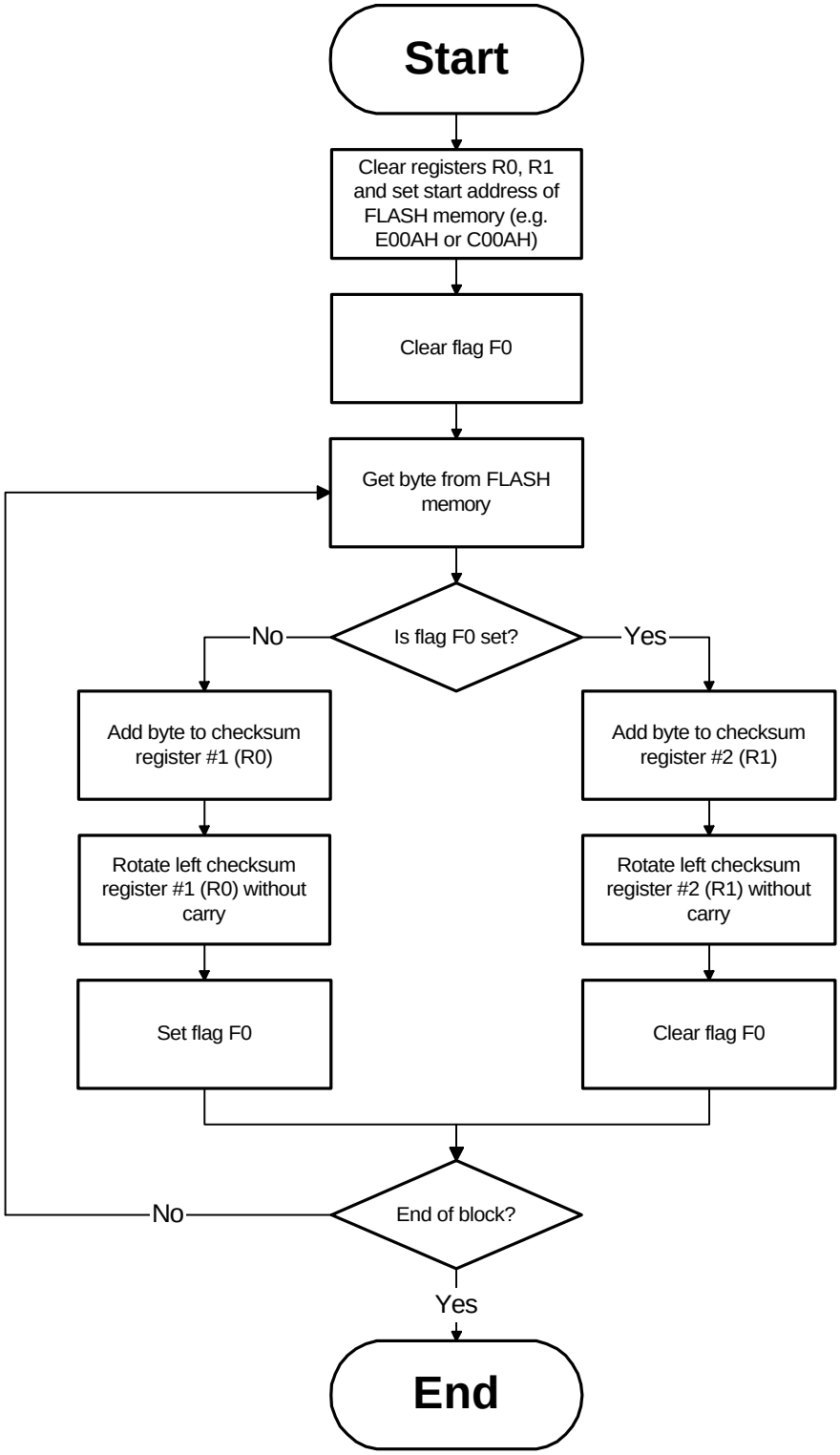

Figure 6: Flowchart of the special checksum calculation for the FLASH memory

1.3. Phase II: Automatic serial synchronisation to the host

When the bootstrap-loader leaves phase **I** and enters phase **II,** the synchronisation procedure between MCU and host will be started. It has to be done as shown in the block diagram below:

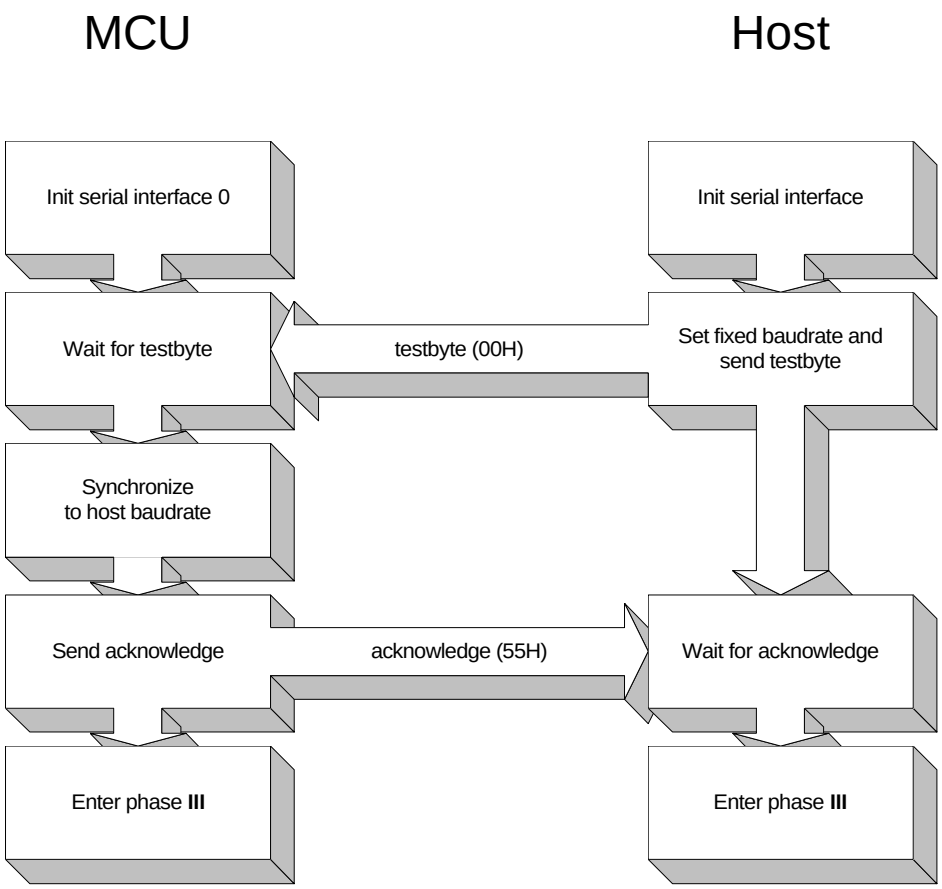

Figure 7: The synchronisation between MCU and host

After receiving the test byte from the host, the bootstrap-loader calculates the actual baud rate and activates the baud rate generator of the serial interface 0. When the synchronisation is accomplished, the MCU sends back the acknowledge code (55H). The baud rate calculation works correctly only in a specific range of baud rates (see section 1.3.3.). If the synchronisation fails, the baud rates between MCU and host are different, and the acknowledge code from the MCU can't be received properly by the host. In this case, the host software may give a message to the customer, e.g. that he has to repeat the synchronisation procedure. Attention: the bootstrap-loader doesn't recognize, if the synchronisation was correct. It always enters phase **III** after sending the acknowledge code. Therefore, if synchronisation fails, a reset of the MCU has to be invoked, to restart the bootstrap-loader for a new synchronisation attempt.

1.3.1. Detailed information for the automatic synchronisation

In phase **II** the bootstrap-loader starts the serial communication with the connected host via serial interface 0. The interface of the MCU is set to mode 3, which means asynchronous transmission with the data format 8N2 (eight data bits, no parity, two stop bits). The host has to use the same serial parameters.
For the baud rate synchronisation of the MCU to the fixed baud rate of the host, the bootstrap-loader waits for a test byte (zero byte, 00H), which has to be sent by the host. By polling the receive port of the serial interface 0 (P3.0) the bootstrap-loader measures the receiving time of the test byte by using timer 0 as shown in the figure below.

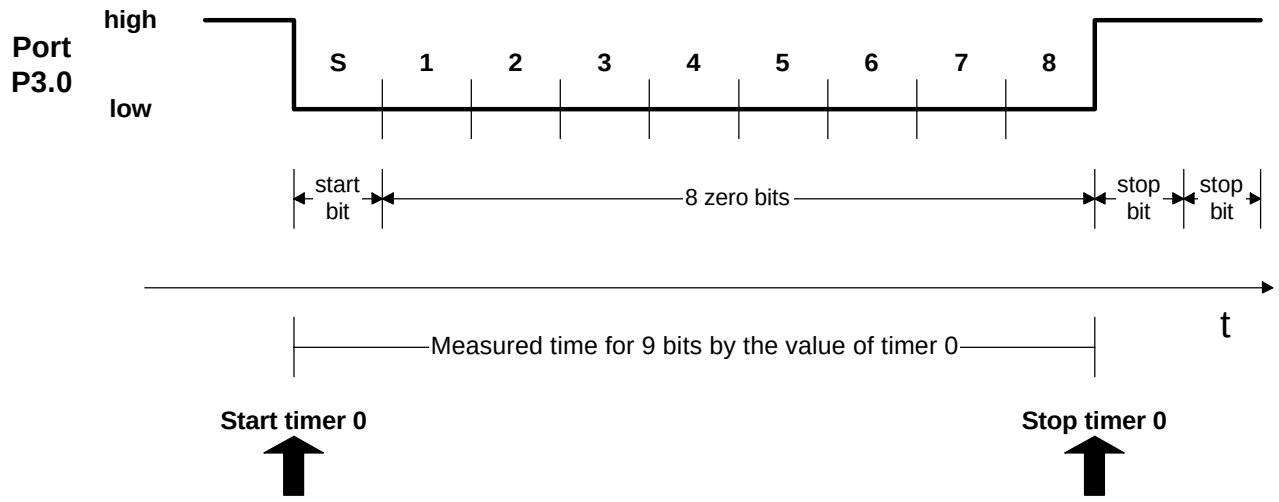

Figure 8: Measuring the receive time of a zero byte by using Timer 0

The resulting timer value is used to calculate the reload value for the 10 bit baud rate generator of the serial interface 0 (S0REL). This calculation needs two formulas: the correlation between the baud rate (Bd) and the reload value (S0REL) depending on the oscillator frequency of the MCU (f_{osc})

$$Bd = \frac{fosc}{32 \bullet (1024 - S0REL)} \qquad (1)$$

and the relation between the baud rate (Bd) and the value of timer 0 (T0) depending on the oscillator frequency (f_{osc}) and the number of received bits (Nb)

$$Bd = \frac{fosc \bullet Nb}{T0 \bullet 12} \qquad (2)$$

Equating **(1)** to **(2)** and resolving the result to S0REL leads to the formula

$$\mathbf{S0REL = 1024 - \frac{T0 \bullet 12}{32 \bullet Nb}}$$

which is independent from the oscillator frequency of the MCU (f_{OSC}).
The value of Nb is nine, because one start bit plus eight data bits are measured. The resulting formula then is

$$\mathbf{S0REL = 1024 - \frac{T0 \bullet 12}{32 \bullet 9}}$$

This equation contains the constant factor

$$\mathbf{\frac{12}{32 \bullet 9} = 0.0417}$$

so the formula can be written as

$$\mathbf{S0REL = 1024 - 0.0417 \bullet T0}$$

To avoid complicated float point arithmetic the factor 0.0417 is scaled by multiplying it with 4096 (result is 171) and then performing an integer multiplication with T0. In the next step the product is rescaled by a integer division with 4096, which can be simply achieved by a twelve bit right-shift operation. The final formula for calculating the reload value S0REL including scaling and rescaling is

$$\mathbf{S0REL = 1024 - \frac{171 \bullet T0}{4096}} \qquad \mathbf{(3)}$$

Additionally, the result of the division is rounded by a simple bit comparison of the last right shifted bit. After setting S0REL to the calculated value and activating the baud rate generator of the serial interface 0, the bootstrap-loader sends an acknowledge byte (55H) to the host. If this byte is received correctly, it will be assured, that both serial interfaces are working with the same baud rate.

1.3.2. The calculation of S0REL in detail

The following flowchart shows the detailed calculation of the reload value S0REL for the baud rate generator of the serial interface 0.

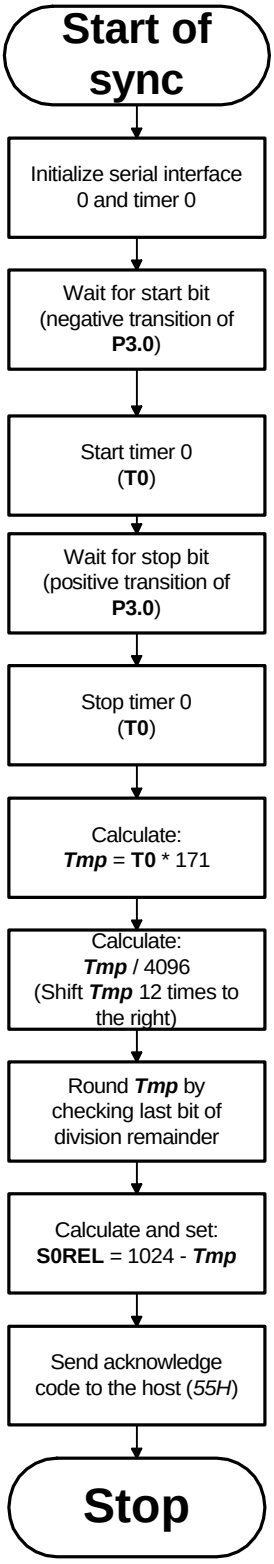

Figure 9: The detailed calculation of S0REL

1.3.3. Functional baud rates for a correct synchronisation

The automatic baud rate synchronisation will work correctly only in a specific range of baud rates, which depends on the oscillator frequency (f_{osc}) of the SAB-C509-L/-16F and the resolution of the timer 0 (T0).
The minimum baud rate (Bd_{min}) results in the possible underflow of SFR S0REL, when the value of T0 gets bigger than 24556D. In this case the value of SFR S0REL**,** corresponding formula **(3),** gets below zero, which leads to a underflow of the S0REL-register and therefore to a incorrect baud rate of the baud rate generator. The formula for calculating this underflow margin is derived from formula **(1)** and is reduced to

$$\mathbf{Bd_{min} = \frac{f_{osc}}{32768}}$$

The theoretic maximum baud rate (Bd_{high}) can be attained, if S0REL is set to its maximum value of 1023**.** In this case formula **(1)** reduces to

$$\mathbf{Bd_{high} = \frac{f_{osc}}{32}}$$

The real maximum baud rate (Bd_{max}) is smaller, because of the decreasing resolution of S0REL and T0 at higher baud rates, This causes an increasing deviation between the host baud rate and the MCU baud rate. To perform a correct transfer between the MCU and the host without transmission errors, the deviation of the host baud rate to the MCU baud rate may not exceed 2.5 %

$$\mathbf{F_b = 0.025 \leq \frac{Bd_{host} - Bd_{MCU}}{Bd_{host}}}$$

The deviation (F_b) depending on the host baud rate for the standard oscillator frequency f_{osc} = 16 MHz is shown in the diagram below.

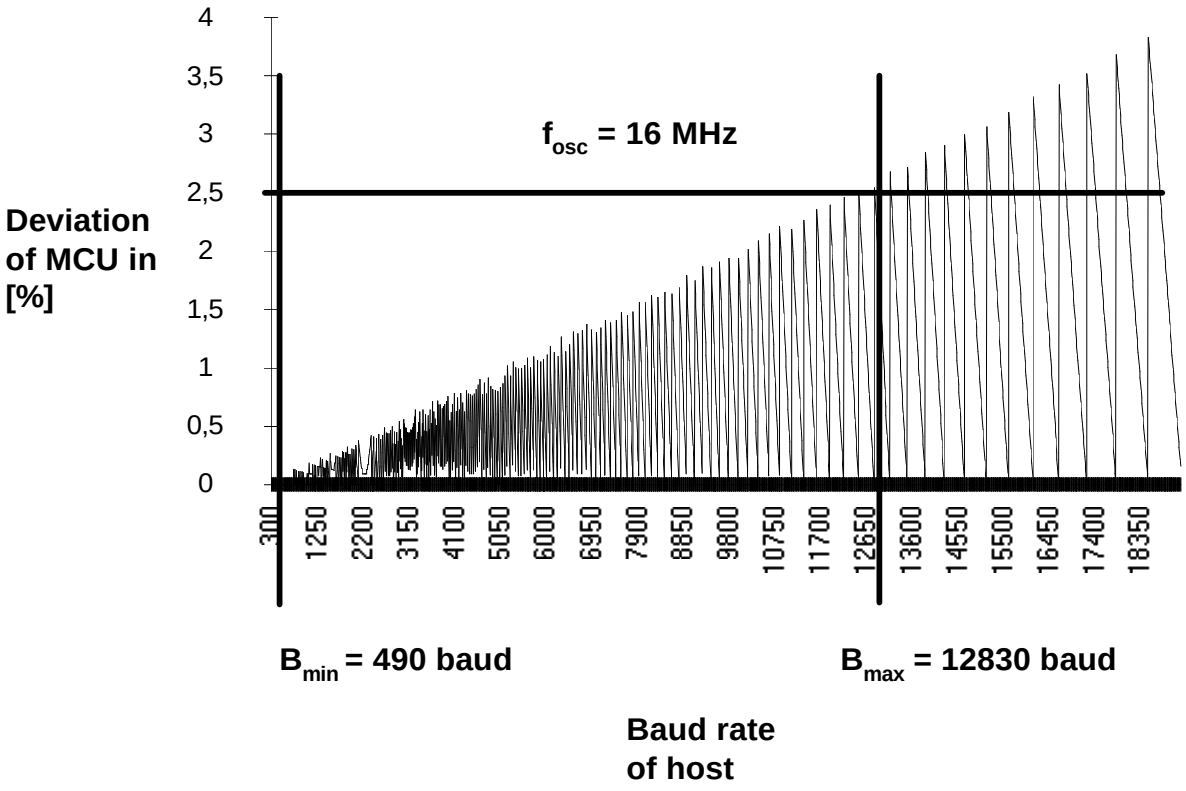

Figure 10: The deviation of the MCU baud rate depending on the host baud rate

Between Bd_{min} and Bd_{max} every host baud rate can be synchronized successfully by the bootstrap-loader. Above Bd_{max} only discreet baud rates with a deviation of less than 2.5 % may be used. The following table shows the guaranteed range of baud rates Bd_{min} to Bd_{max} and typical functional host baud rates above Bd_{max} for some MCU clock rates f_{osc}.

MCU clock rate f_{osc}	Minimum baud rate Bd_{min}	Maximum baud rate Bd_{max} ($F_b \leq 2{,}5\%$)	Functional higher baud rates
8 MHz	250 baud	6580 baud	9600, 19200 baud
12 MHz	370 baud	9380 baud	9600, 19200 baud
16 MHz	490 baud	12830 baud	9600, 19200, 38400 baud

1.4. Phase III: Serial communication and selecting the working modes

After the successful synchronisation to the host the bootstrap-loader enters phase III, in which it communicates to the host to select the desired working modes.The global communication protocol is explained now.

1.4.1. The block transfer protocol

The communication between the host and the bootstrap-loader is done by a simple transfer protocol, which includes a specified block structure. The communication is nearly unidirectional, that means, that the host is sending several data blocks and the bootstrap-loader is just confirming them by sending back single acknowledge or error bytes. The MCU itself does not send any data blocks
The general format of a transfer block is shown in the following figure.

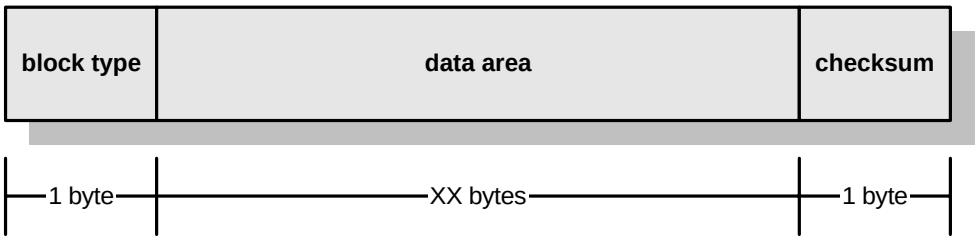

Figure 11: The structure of a transfer block sent by the host

Description:

block type ..... the type of block, which determines how the data in the **data area** has to be interpreted.
Implemented block types are:
00H type '**HEADER**'
01H type '**DATA**'
02H type '**END OF TRANSMISSION**' (**EOT**)

data area ..... a list of bytes, which represents the data of the block. The number of bytes in the data area may range between **00H** and **7FH** (**0D** and **127D**).

checksum ..... for safety purposes a checksum of **block type** and **data area** is built by the host and sent with the transfer block.

A transfer block is built by the host depending on the data (header or program data) it contains. For safety purposes the host calculates a simple **checksum** of the whole block (**block type** and **data area**) to attach it at the end of the block. The checksum must be generated by EXOR-ing all bytes of the transfer block with themselves. Every time the bootstrap-loader receives a transfer block, it recalculates the checksum of the received bytes (**block type** and **data area**) and compares it with the attached checksum. If the comparison fails, the bootstrap-loader is rejecting the transfer block by sending back a checksum error code (FEH) to the host. Another possible error is a wrong block type. In this case the bootstrap-loader sends back a block error code (FFH) to the host. In both error cases the bootstrap-loader awaits the actual block from the host again. If a block is received correctly, an acknowledge code (55H) is sent.

Receive status	**transmitted code to host**
Acknowledge	55H
block error	FFH
checksum error	FEH

Three types of transfer blocks depending on the value of **block type** are implemented in the transfer protocol. teh following table shows an overview of these block types. The detailed structures of the blocks are described in the corresponding sections later.

block name	**block type**	**Description**
header block	00H (HEADER)	This block always has a length of 8 bytes (including the attached checksum) and contains special information in the **data area**, which selects the working mode of the bootstrap-loader in phase III.
data block	01H (DATA)	This block is used in working mode 0 to transfer a portion of normal data in the **data area** (e.g. program code) from the host to the XRAM of the MCU. The length of this block depends on a special information given in the header block before.
EOT block	02H (EOT)	This block is used to indicate the end of a data transmission in working mode 0. It contains the last bytes of the transferred data. The length of this block depends on a special information given in a header block before.

1.4.2. The selection of the working modes

When the bootstrap-loader enters phase III, it first waits for a eight byte long header block from the host, which will be confirmed with an acknowledge code (55H). The header block contains the information for the selection of the working mode. Depending on this data, the bootstrap-loader selects and activates the desired working mode. This action is shown in the block diagram below:

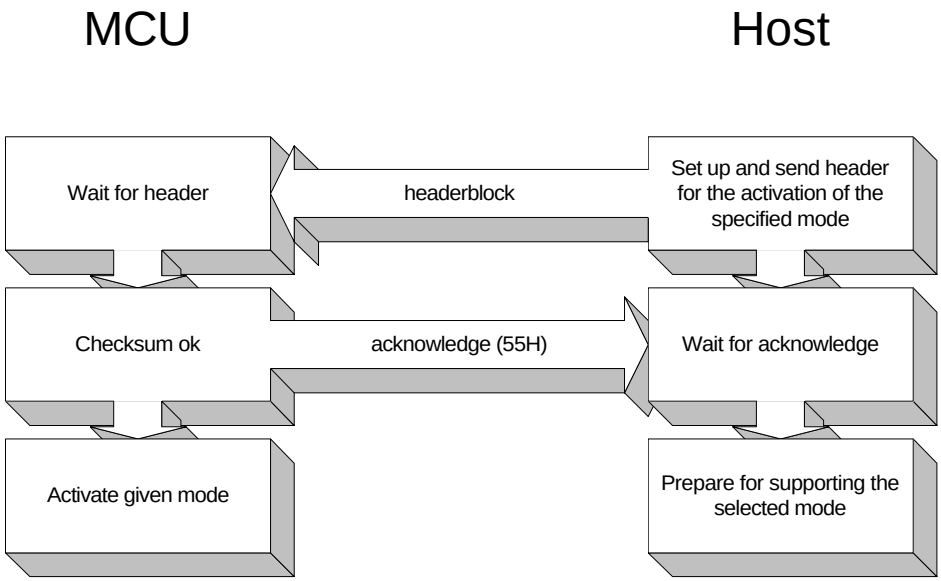

Figure 12: The working-mode is selected by sending a special header block to the MCU

If the MCU receives an incorrect header block, e.g. because of a bad serial transmission, the bootstrap-loader sends, instead of an acknowledge-code, a checksum- or block-error code to the host and awaits the header block again. In this case the host may react by resending the header block or by releasing a message to the customer.

The following flowchart shows the general structure of phase III:

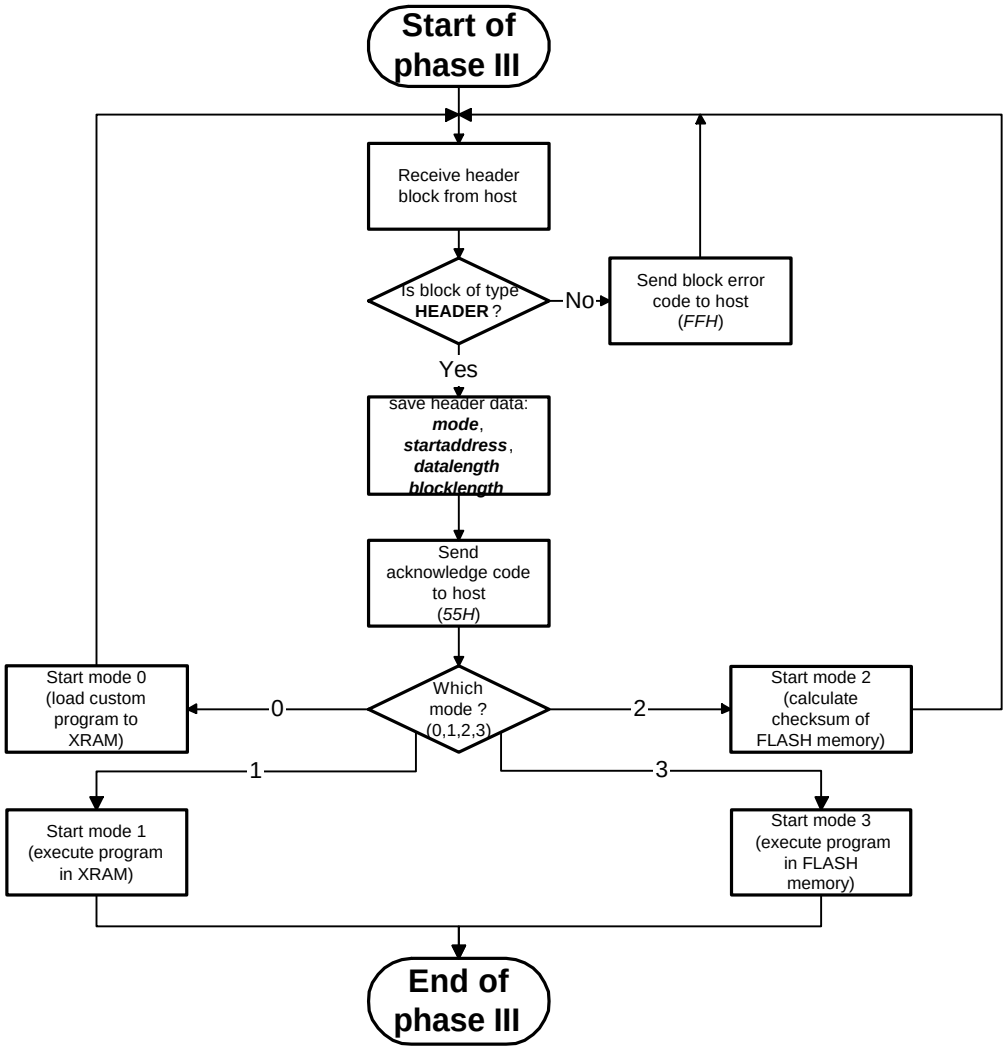

Figure 13: The flowchart of phase III

1.4.2.1. Receiving the header block

The header block from the host, which contains the mode number and additional data (**mode data**) to start this mode, is a normal transfer block with the special block type HEADER (**block type** = 00H) and a fixed length of eight bytes (including the attached checksum of the transfer block). The general structure of this header block is shown below.

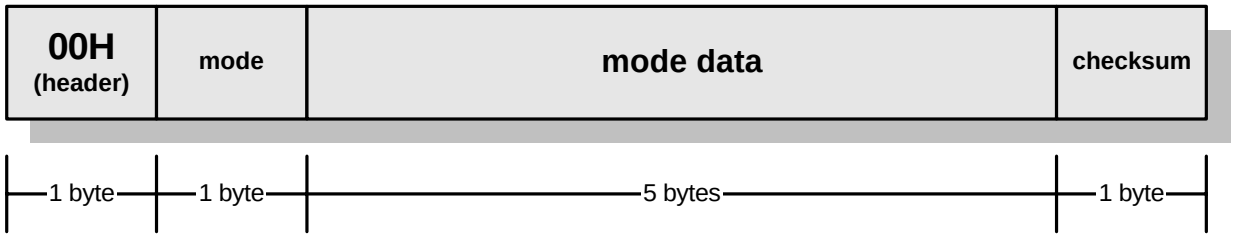

Figure 14: The structure of the header block

Description:

00H (header) ..... The byte, which marks the block as **HEADER**

mode ..... The data value for the working mode, which has to be activated by the bootstrap-loader. Possible modes are:

- **mode 0** (**00H**) - Transfer a custom program to the XRAM
- **mode 1** (**01H**) - Jump to a specified start address in the XRAM
- **mode 2** (**02H**) - Calculate a special checksum for a specified block of the FLASH memory
- **mode 3** (**03H**) - Jump to a specified start address in the FLASH memory

mode data ..... Five bytes of special data, which are neccessary to activate the corresponding working mode

checksum ..... The checksum of the header block

If the block type and the checksum of the received header block are correct, the bootstrap-loader sends an acknowledge byte to the host (55H). In case of a wrong block type, the bootstrap-loader rejects by sending the block error code (FFH). A wrong checksum leads to the checksum error code (FEH).

1.4.2.2. The activation of working mode 0

Mode 0 is used to transfer a program from the host to the XRAM of the MCU via serial interface.The special header block, which has to be prepared and sent by the host for the activation of working mode 0 has the following structure:

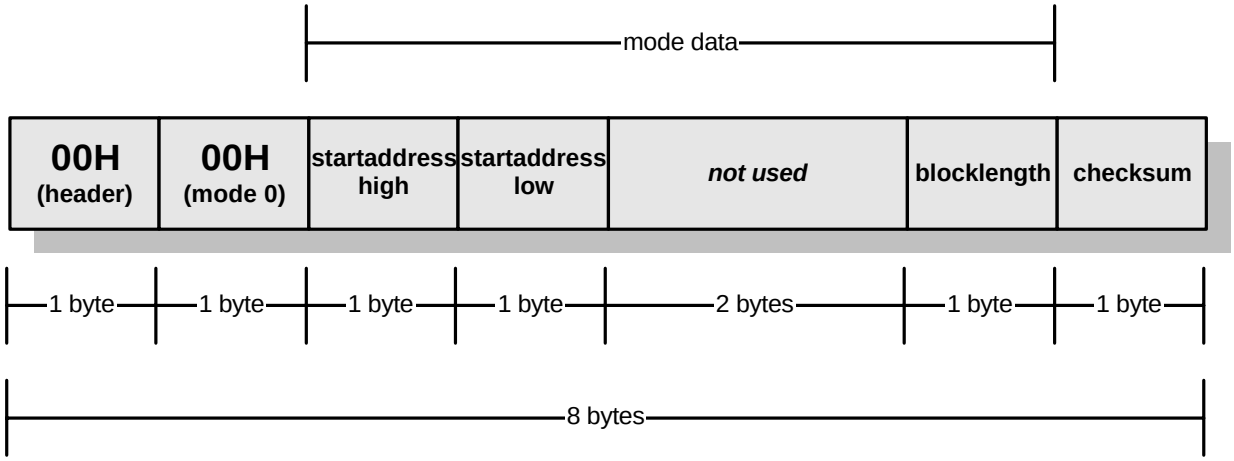

Figure 15: The header block for activating mode **0**

Mode data description:

startaddress high, low ..... 16 bit start address, which determines, where to copy the received data in the XRAM.

blocklength ..... The length of the following data bytes in the data and EOT blocks

not used ..... These bytes are not used and can be set to any value. They will be ignored in mode 0

After confirming the received header block, the bootstrap-loader enters mode 0, which transmits the desired data from the host to the XRAM of the MCU using two transfer blocks of the types **DATA** and **EOT**, which are described below:

The data block (DATA):

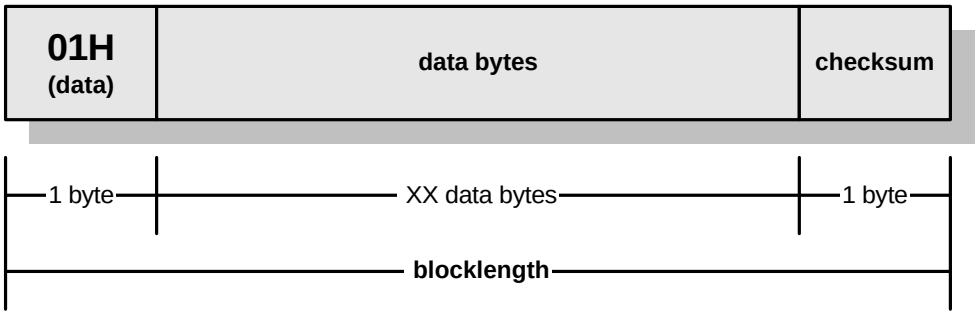

Figure 16: The structure of a data block

The data block contains a bulk of data, which has to be copied to the XRAM beginning at **startaddress** in the header block. The number of the **data bytes** is specified by **blocklength** in the header block.

The EOT block (End **O**f **T**ransmission**):**

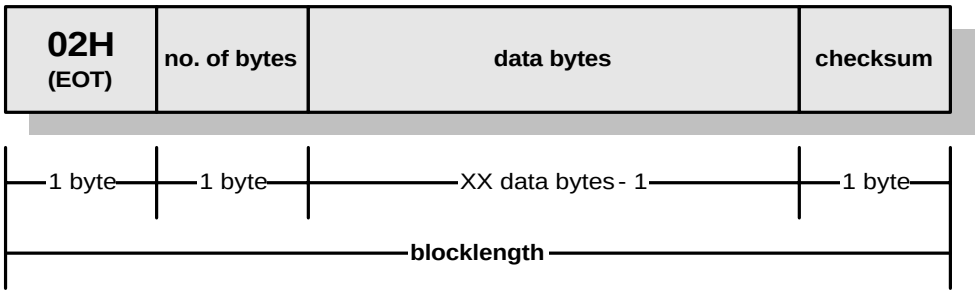

Figure 17: The structure of an EOT block

The EOT block contains the last bytes of a data transmission to the XRAM. The number of the remaining relevant bytes in the EOT block is given in **no. of bytes**.
The complete communication for mode 0 (including entering mode 0) between the host and the MCU after the synchronisation is shown in the block diagram on the next page.

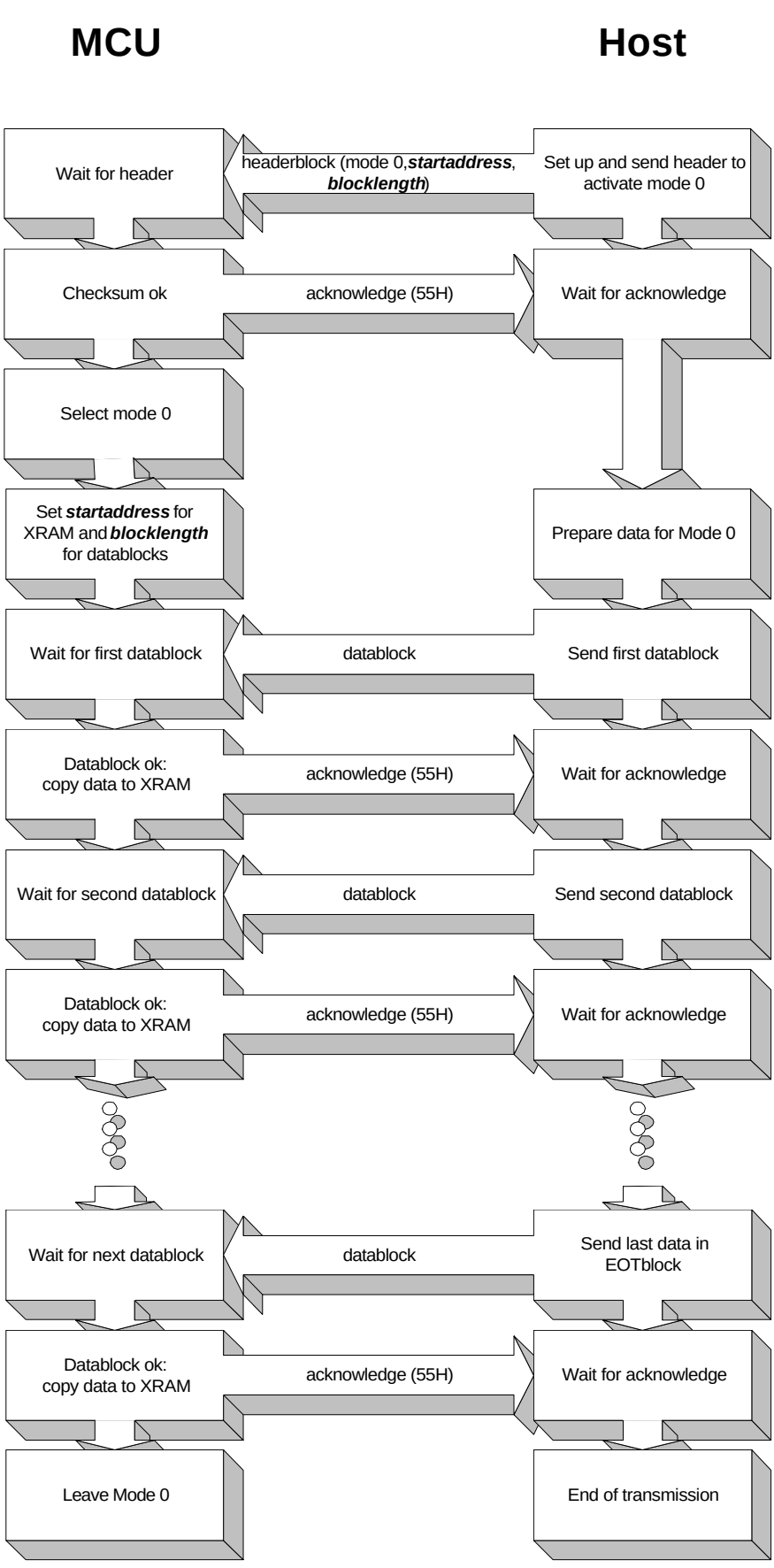

Figure 18: The error free transmission of blocks in mode 0

If an error occurs while transmitting data blocks, the host software has to react on error codes, sent by the bootstrap-loader when rejecting a block. This case is illustrated in the next block diagram:

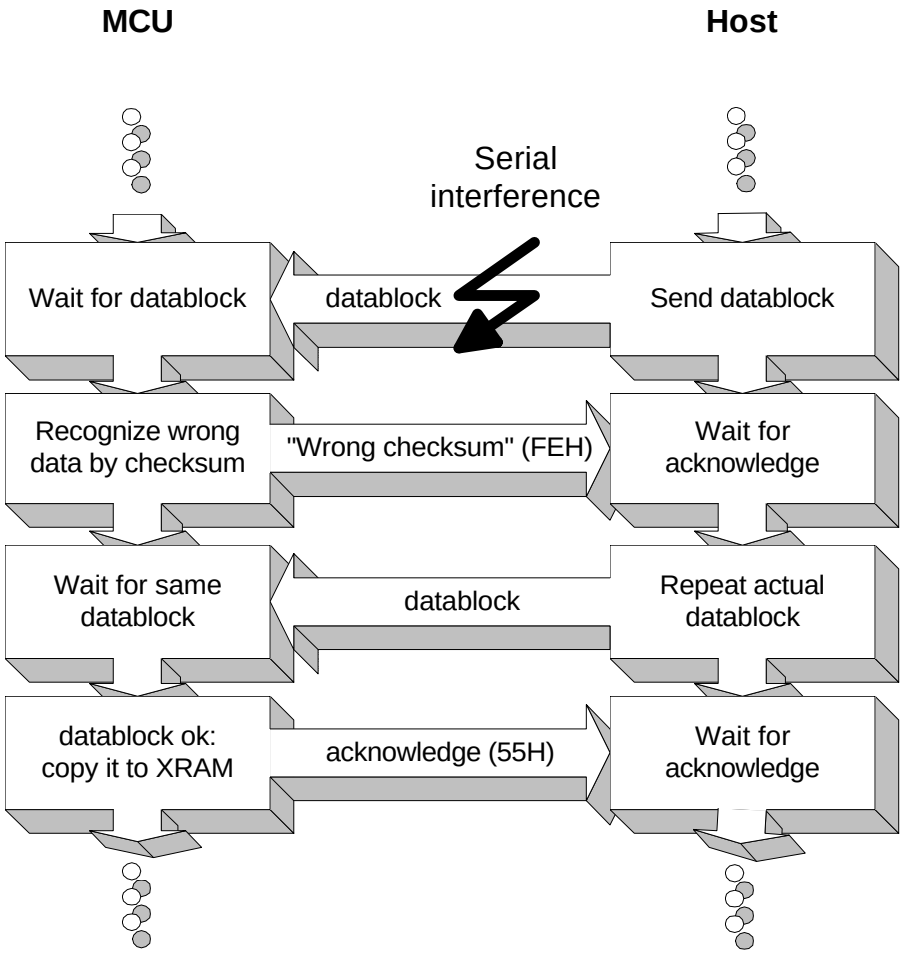

Figure 19: The block diagram for handling serial transmission errors in mode 0

If the host erroneously sends a header block or a block that is not implemented in the protocol (a block type number higher than 02H), the bootstrap-loader reacts in a similar way as described in the figure above, with the exception, that now a block error code (FFH) is sent to the host. It is up to the host software to handle this error properly.

The flowchart of the complete transfer protocol is shown below.

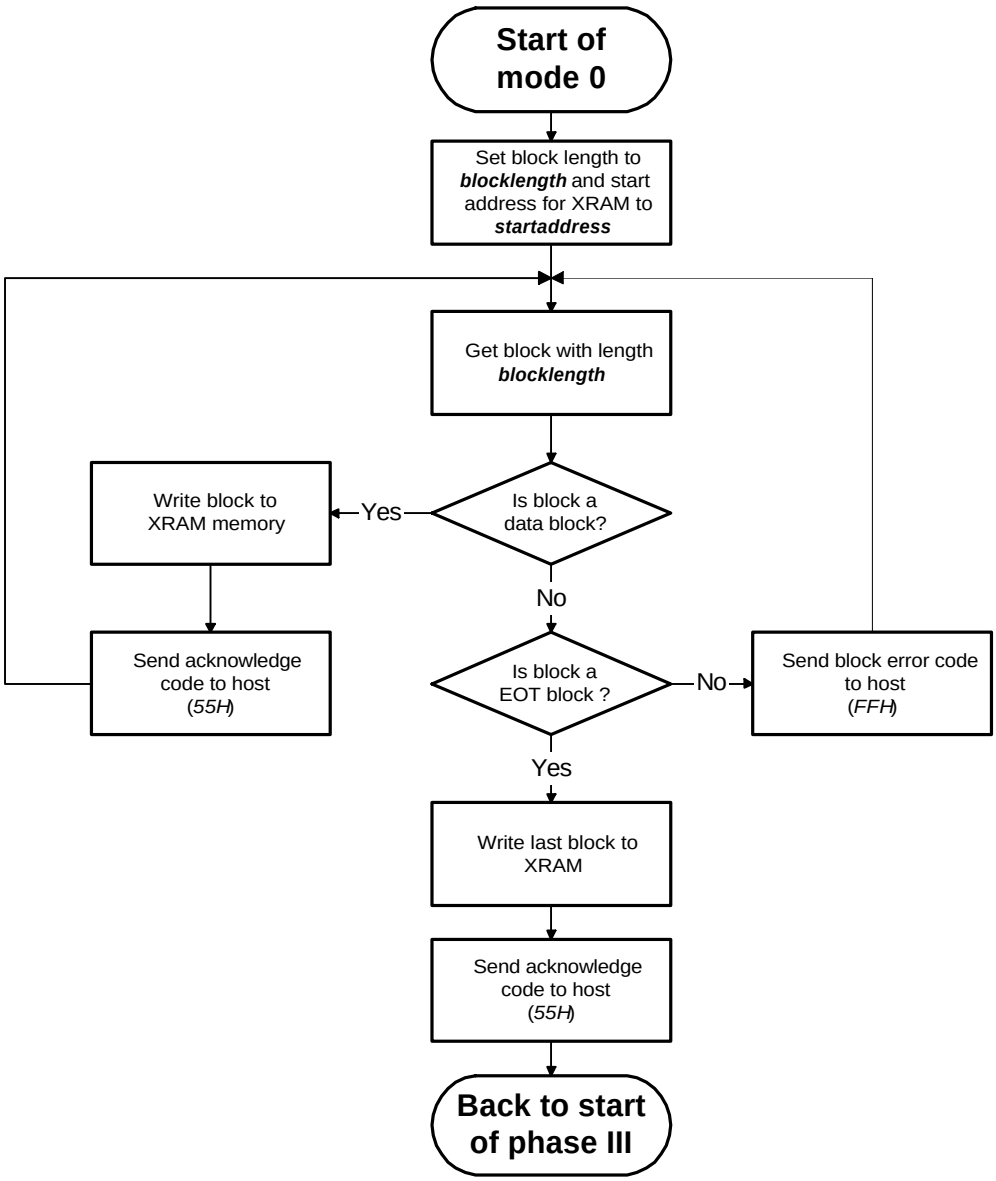

Figure 20: The transfer protocol of mode 0

1.4.2.3. The activation of working mode 1

Mode 1 is used to execute a custom program in the XRAM of the MCU at a given start address. The special header block, which has to be prepared and sent by the host for the activation of working mode 1 has the following structure:

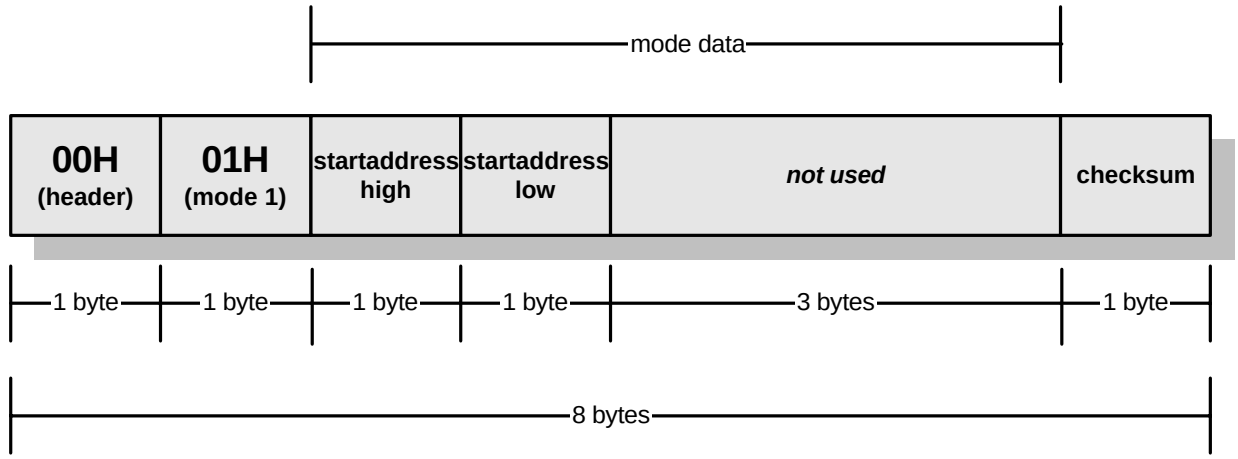

Figure 21: The header block for activating mode **1**

Mode data description:

startaddress high, low ..... 16 bit start address for program execution in the XRAM (F400H - FFFFH, if the interrupts TIMER 0 and SINT 0 are not used; F420H - FFFFH if the interrupts TIMER 0 and SINT 0 are used.

not used ..... These bytes are not used and can be set to any value. They will be ignored in mode 1.

After sending the appropriate header block for mode 1, no further serial communication is necessary. The block diagram for the selection of mode 1 is shown below:

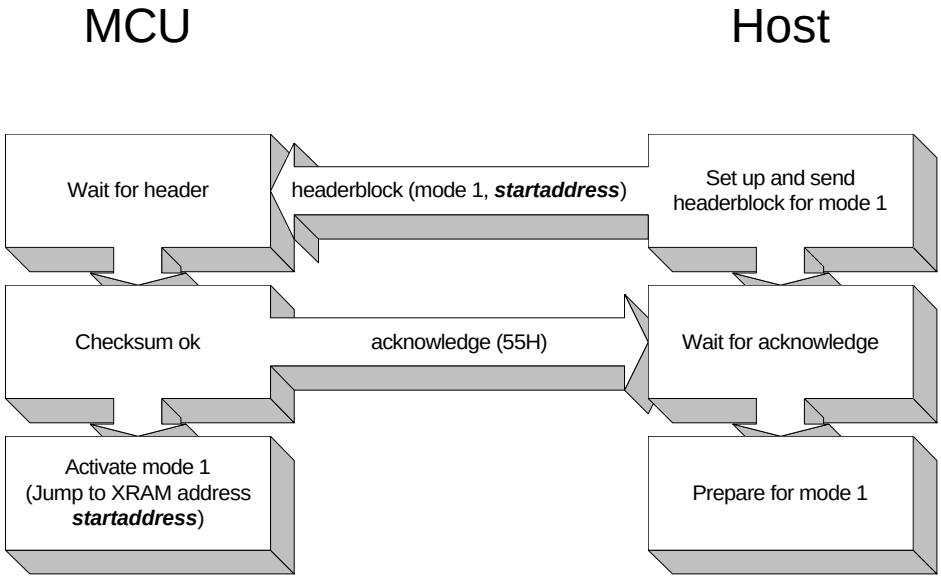

Figure 22: The block diagram for the activation and the handling of mode 1

Mode 1 swaps the XRAM to code memory by setting the corresponding swap-bit in SFR SYSCON1 and starts the program execution in the XRAM at any start address given in the header block as **startaddress**. Notice: if the supported interrupts TIMER 0 and SINT 0 are used as described in section 1.1., the XRAM area from F400H to F41FH is reserved for interrupt handling routines. Therefore **startaddress** has to be greater than or equal to F41FH.
The corresponding flowchart is shown below:

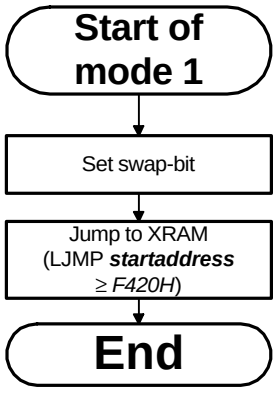

Figure 23: Flowchart of working mode 1

1.4.3. The activation of mode 2

Mode 2 is used to calculate a special checksum of a FLASH memory sector beginning at a start address and with a specified length. The special header block, which has to be prepared and sent by the host for the activation of working mode 2 has the following structure:

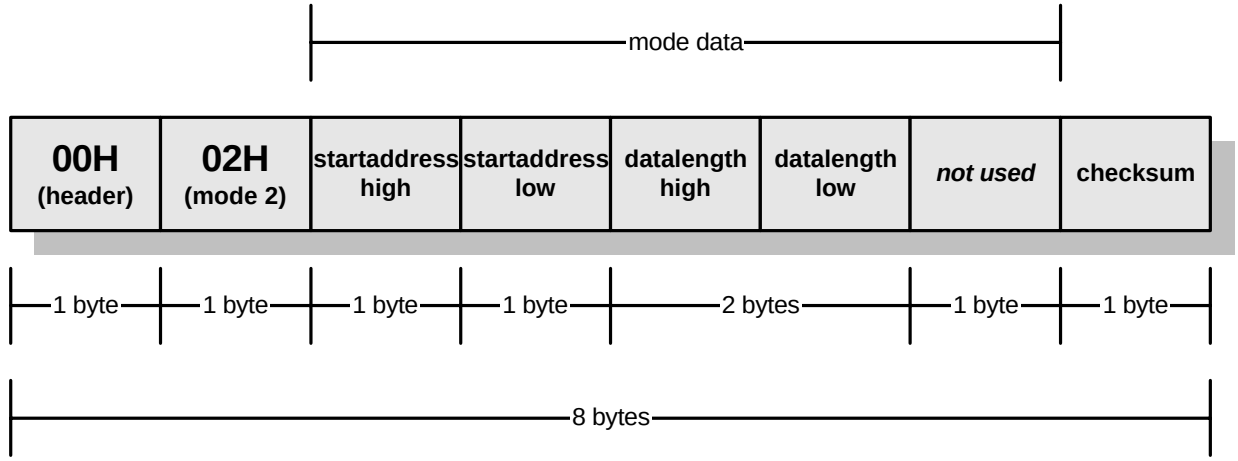

Figure 24: The header block for activating mode **2**

Mode data description:

startaddress high, low ..... The 16-bit start address of the FLASH memory block, which has to be checked

datalength high, low ..... The number of bytes, which have to be checked beginning at **startaddress**. Allowed is any value between 0000H and FFFFH.

not used ..... This byte is not used and can be set to any value. It will be ignored in mode 2

Mode 2 calculates a special checksum of any area of the FLASH memory starting at **startaddress** with a length of **datalength**. The end address of the corresponding memory block is **startaddress + datalength**. The two calculated checksum bytes are compared with two fixed checksum values which must be placed at the end of the checked FLASH memory block, that is on addresses **startaddress + datalength + 1** and **startaddress + datalength + 2**. If these checksum values are equal to the calculated checksums, an acknowledge byte (55H) is sent to the host. Otherwise the checksum error code (FEH) is transmitted. The block diagram below illustrates this process:

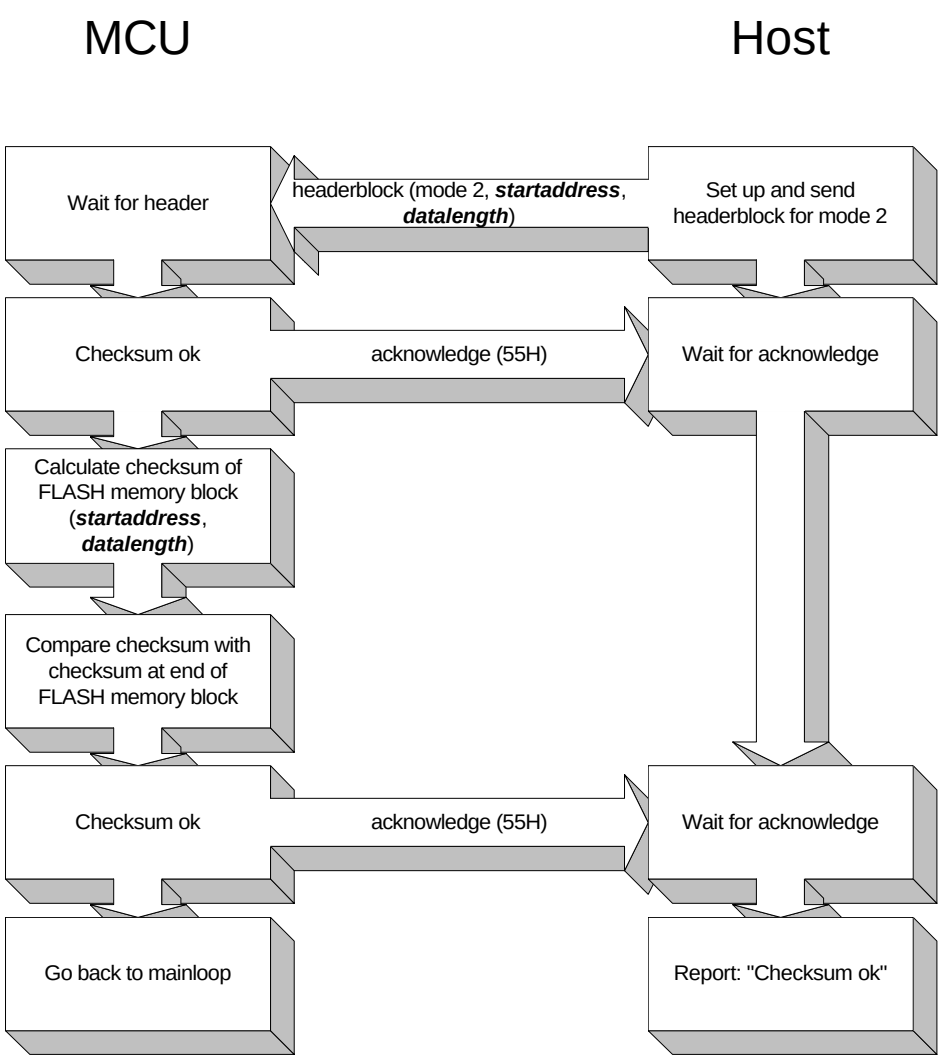

Figure 25: The block diagram for the activation of mode 1

The following figure shows the function of mode 2 as flowchart.

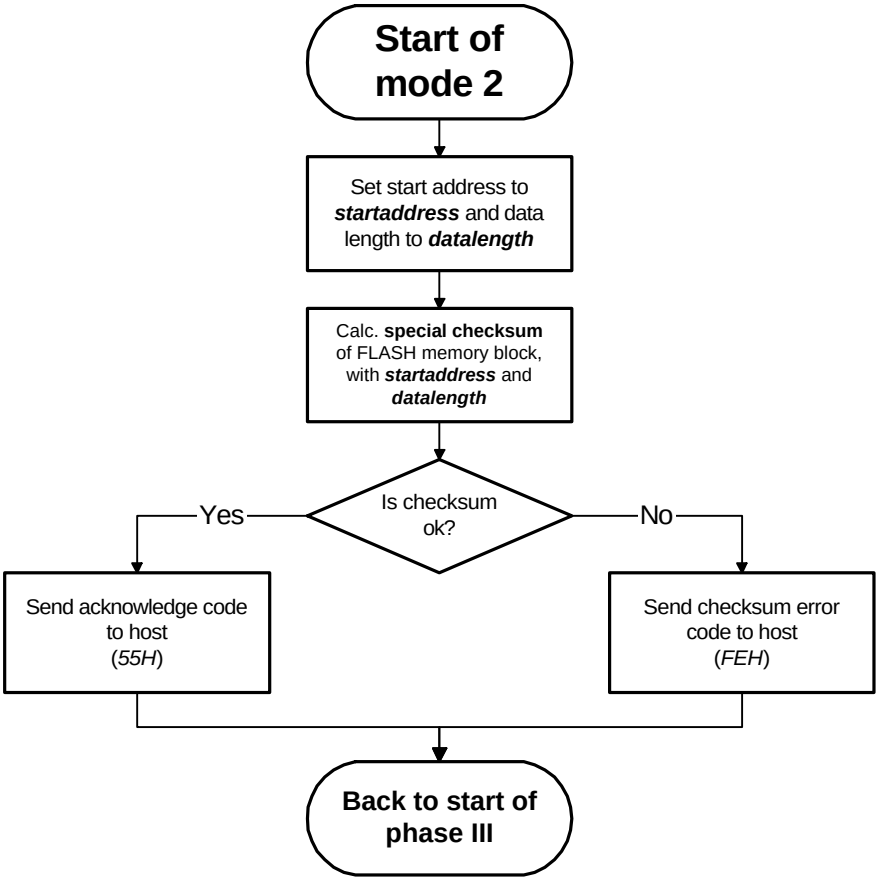

Figure 26: Flowchart of the working mode 2

1.4.4. The activation of working mode 3

Mode 3 is used to execute a custom software in the FLASH memory at any start address. The special header block, which has to be prepared and sent by the host for the activation of working mode 3 has the following structure:

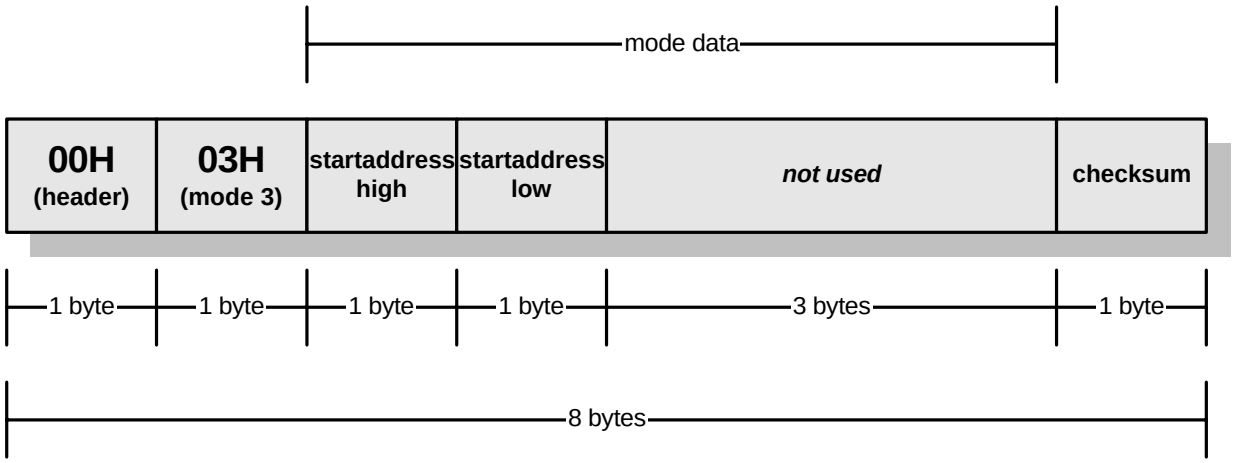

Figure 27: The header block for activating mode **3**

Mode data description:

startaddress high, low ..... The 16-bit-start address for program execution in the FLASH memory

not used ..... These bytes are not used and can be set to any value. They will be ignored in mode 3

The working-mode 3 performs a direct program execution at a given start address, which is sent in the header block. After sending the appropriate header block no further serial communication is necessary for this mode. The block diagram for the selection of mode 3 is shown below.

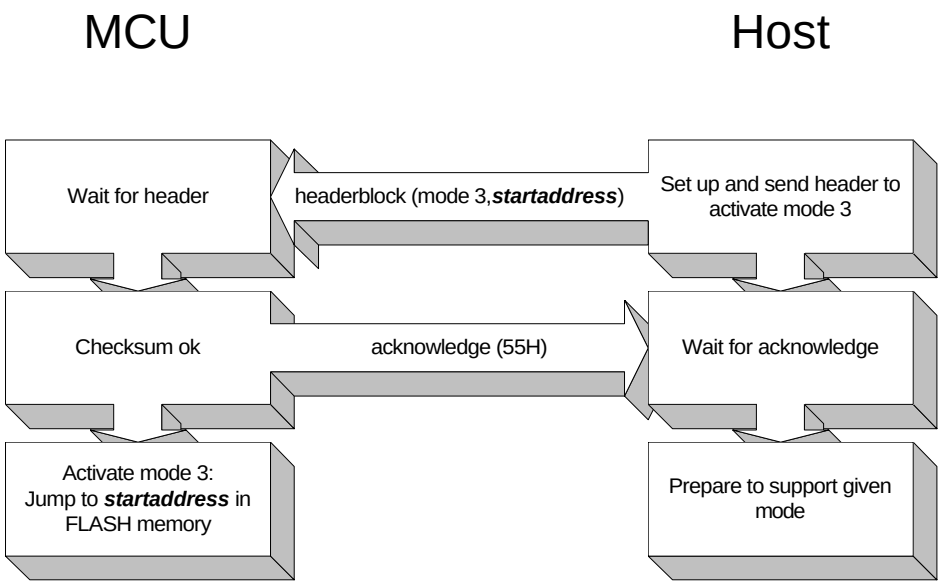

Figure 28: The block diagram for the activation of mode 3

Attention: The start address **startaddress** has to be greater than 200H, because in the bootstrap-mode the bootstrap-loader overlaps the code address area of the FLASH memory from 0000H to 01FFH.
The following figure shows thee function of working mode 3 as flowchart:

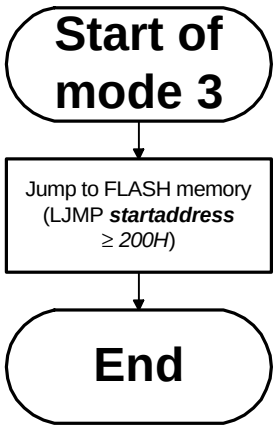

Figure 29: Flowchart of the bootstrap-loader - Mode 3

Appendix A

A.1. Technical Reference

This section describes the software of the bootstrap-loader with its most important subroutines and start addresses, which can be used by the customer for own purposes. This technical reference is only relevant for the version **1.6** of the bootstrap-loader.

A.1.1 Description of the subroutines

The following table contains all subroutines of the bootstrap-loader which can be used by the customer, when executing custom programs.

Address	Function	Registers	Description
000EH	SendByte	In: **A** - Byte to send Out: None	Send a byte to the serial interface 0
0016H	SendAckn	In: None Out: None	Send an acknowledge code (55H) to the serial interface 0
001BH	SendBlockErr	In: None Out: None	Send a block error code (FFH) to the serial interface 0
0074H	CalcBaudRate	In: **TH0**/**TL0** - measured value for test byte in **T0** Out: **R1**/**R2** - value for S0REL	Calculate the value for the 10 bit baud rate generator reload register S0REL of the serial interface 0 depending on the value of T0 (**TH0**/**TL0**) for receiving the test byte (00H)
00A6H	CheckBaud	In: None Out: None	The complete baud rate synchronisation: 1. Prepare the timer 0 for measurement 2. Wait for the test byte from host 3. Measure the time between the start bit and the stop bit 4. Calculate the baudrate from the value of timer 0 (T0) 5. Initialize the serial interface 0 by setting the baud rate and the serial parameters (8N2) 6. Send an acknowledge code (55H) to the host

00C7H	GetBlock	In: **R7** - block length Out: None	Get an amount (in **R7**) of data bytes from the host and save it to a temporary buffer starting at address 70H and ending at address FFH. The last received byte contains the checksum of the data block. Examine the sent checksum and send back a checksum error code, if it is incorrect.
00E9H	CalcChks	In: **R0** - start address of the temporary buffer **R1** - length of the buffer Out: **A** - calculated checksum	Calculate the checksum of the data block starting at address in **R0** with a length of **R1**. The checksum is saved in **A**.
00F0H	CalcChksFLASH	In: **DPTR** - start address of the area in the FLASH memory **R4**/**R5** - length of the FLASH memory area Out: **R0** - calculated checksum #1 **R1** - calculated checksum #2	Calculate the special checksum of an area in the FLASH memory starting at address in **DPTR** with a length of **R4**/**R5** (high, low). The special checksums #1 and #2 are saved in **R0** and **R1**.
0119H	CheckFLASH	In: **R2**/**R3** - start address of corresponding FLASH memory block Out: **C** - carry flag	Check if the FLASH memory block starting at address **R2**/**R3** contains a functional custom program. 1. Check the ID bytes of the FLASH memory block 2. Calculate the special checksum of the given memory block 3. Set the carry flag if no custom program exists
0161H	Block2XRAM	In: **R0** - start address of the temporary buffer **R1** - length of the buffer **DPTR** - actual address in the XRAM Out: **DPTR** - actual address in the XRAM	Copy the contents of the temporary buffer starting at address in **R0** with a length of **R1** to the actual address (**DPTR**) in the XRAM . The data pointer **DPTR** is incremented automatically.

0168H	CheckHeader	In: **R0** - start address of the temporary buffer Out: **R1** - working mode **R2**/**R3, DPTR** - header data **startaddress** **R4**/**R5** - header data **datalength** **R7** - header data **blocklength** **C** - Carry flag	Analyze the received header and save the header data into the corresponding registers. If the checked block is not of type **HEADER,** a block error code (FFH) is sent to the host and the carry flag **C** is set.
018AH	SendCheckErr	In: None Out: None	Send a checksum error code to the serial interface 0
018FH	Mode0	In: **DPTR** - start address of XRAM to copy a custom program **R7** - length of the data blocks received via serial interface 0 Out: **DPTR** - actual address in the XRAM	Activate working mode 0. The header data must be received or the registers **DPTR** and **R7** must be set manually, before this routine can work correctly.
01B1H	Mode1	In: **DPTR** - address to start a custom program in the XRAM Out: None	Activate working mode 1, that is to start custom program in the XRAM at address in **DPTR**. Attention: This routine does not return, when finished.
01BBH	Mode2	In: **DPTR** - start address of the area in the FLASH memory **R4**/**R5** - length of the FLASH memory area Out: None	Activate working mode 2, that is to calculate a special checksum of a FLASH memory area given by the start address in **DPTR** and the area length in **R4**/**R5**.
01D6H	Mode3	In: **DPTR** - address to start a custom program in the FLASH memory Out: None	Activate working mode 3, that is to start a custom program in the FLASH memory at address in **DPTR**. Attention: This routine does not return, when finished.

A.1.2. Stepping into any bootstrap-loader phase

The bootstrap-loader is sequentially working through the three phases. The following addresses allow the customer to step into the bootstrap-loader in several phases.

Address	Function	Description
0000H	Main	Start the complete bootstrap-loader
0026H	LookFLASH1	Phase **I**: Search for a functional custom program in the FLASH memory sectors A and B
0035H	LookFLASH2	Phase **I**: Search for a functional custom program only in the FLASH memory sector B
0044H	PrepSerial	Phase **II**: Initialize the serial interface 0 and synchronize it to the host baud rate
004AH	WaitHeader	Phase **III**: Wait for the header to select the working mode
004FH	Header	Phase **III**: Check the header block information and save it in specific registers
0054H	Jump2Mode	Phase **III**: Select and activate the working mode given in register **R1**