SIEMENS

Microcomputer Components

8-Bit CMOS Microcontroller

SAB-C512A-H / SAB-C513A-H

Evaluation Programming System

Application Note **2’95**

Contents Page

1 General . . . 3

2 Features of the Evaluation Programming System . . . 4

3 System Description . . . 4

4 User Interface Program . . . 6
4.1 Commands of the User Interface Program . . . 7
4.2 Further Features of the User Interface Program . . . 9

5 Host System Interface Formats . . . 10
5.1 Read Data Records . . . 11
5.1.1 Read Request Data Record . . . 11
5.1.2 Read Data Record . . . 11
5.2 Write Data Record . . . 12
5.3 Status Records . . . 13
5.4 Program Device Record . . . 16
5.5 Verify Device Record . . . 17
5.6 Transfer Device-to-RAM Record . . . 18
5.7 Check Device Record . . . 19
5.8 Erase Record . . . 20
5.9 Reset Record . . . 21

6 Program-Listings . . . 22

1 General

The SAB-C512A-H/SAB-C513A-H microcontroller are programmable versions of the SAB-C512A/ SAB-C513A microcontrollers, in which the mask programmable 12 KB ROM has been replaced by an EEPROM memory of the same size. For programming of the devices several commercially available programmers can be used. For evaluation purposes in addition a simple SAB-C512A-H/ SAB-C513A-H evaluation programming system (figure 1) which is described in this application note is available. Further information about device operation and programming capabilities are described in the user manuals.

The evaluation programming system is build up by a host controller system with a SAB-C501 microcontroller, which controls the programming of the SAB-C512A-H/SAB-C513A-H device in the programming socket. The host system itself is connected to a PC via a RS232 interface. A special user interface program, which communicates with the host system, handles all actions of the evaluation programming system. It especially also reads absolute data files (standard Intel formats), which are generated by compilers and assemblers for the 8051 microcontroller family, and transfers its data to the host controller.

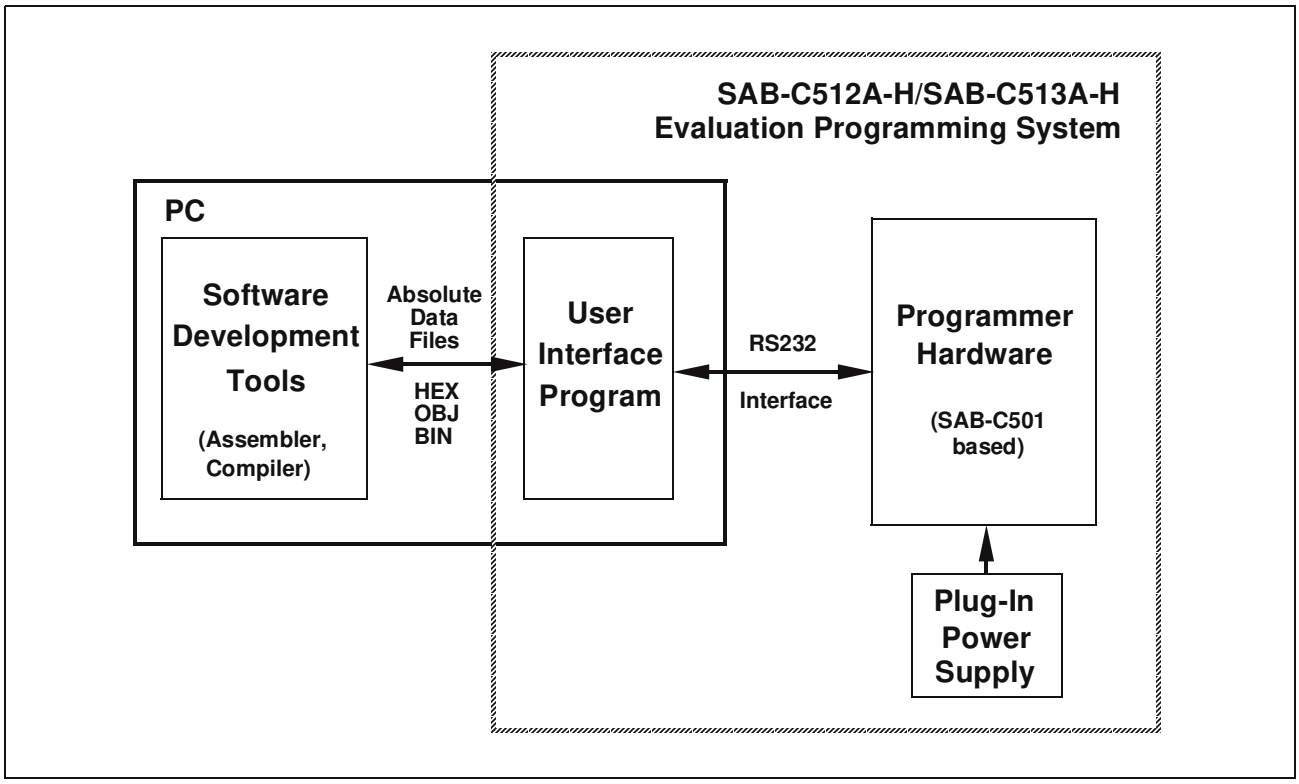


Figure 1 : SAB-C512A-H/SAB-C513A-H Evaluation Programming System Configuration

The programming hardware, which contains the programming socket, is powered by a plug-in power supply.

2 Features of the Evaluation Programming System

The SAB-C512A-H/SAB-C513A-H evaluation programming system provides the following features:

- o Programming System with a RS232 interface for connection to a PC via serial interface
- o One programming sockets for SAB-C512A-H or SAB-C513A-H devices
- o PC user interface program provides
  - Load Intel HEX/OBJ and BIN data into programming RAM
  - Display / change content of programming RAM
  - Program and verify EEPROM with programming RAM content
  - Read EEPROM(s) content into programming RAM
  - Save programming RAM content as BIN data file
  - Erase EEPROM
  - Check programmability of EEPROM

The evaluation programming system features all functions typically implemented in a programmer system. The absolute data file formats which can be handled by the evaluation programming system are the “Intel Absolute Object Format” (referenced as OMF-51 or OBJ) and the “Intel Absolute Hex File Format” (referenced as HEX). Additionally a BIN file format (bytewise storage of the RAM/EEPROM content without any address information) can be handled (read and written).

3 System Description

Figure 2 shows the block diagram of the evaluation programming system. The host system is built up with a SAB-C501 MCU standard system, which includes the MCU, the address latch, a 32K EPROM and a 32K SRAM. The EPROM holds the host controller firmware and the SRAM is used for the 12K programming RAM. The host controller system operates at 8 MHz clock rate. The interface to the PC is build up with a MAX232 device which transforms the CMOS levels to/from the V24 voltage levels. A reset switch and an 8 MHz clock generator completes the basic microcontroller system.

The power supply for the programming socket can be switched to the 5V VCC via a relay. This allows to insert/extract the programming device without power applied to it. A red LED indicates the “voltage on programming socket“ condition. If the red LED is off and the green LED is on, the programming device can be inserted/extracted.

The clock and the control signals to the programming sockets and the bidirectional data bus is decoupled from the host system by 74HC241 and 74HC245 driver circuits. Pullup resistors connected to the switched VCC power supply avoid wrong signal levels at the control signals to the programming sockets. The driver circuits are controlled by some gates. To avoid data bus contention at programming RAM accesses, the data bus driver is enabled only at read/write accesses (MOVX) in address range 8000H to FFFFH. The control signal driver 74HC241 is enabled by P3.3=0.

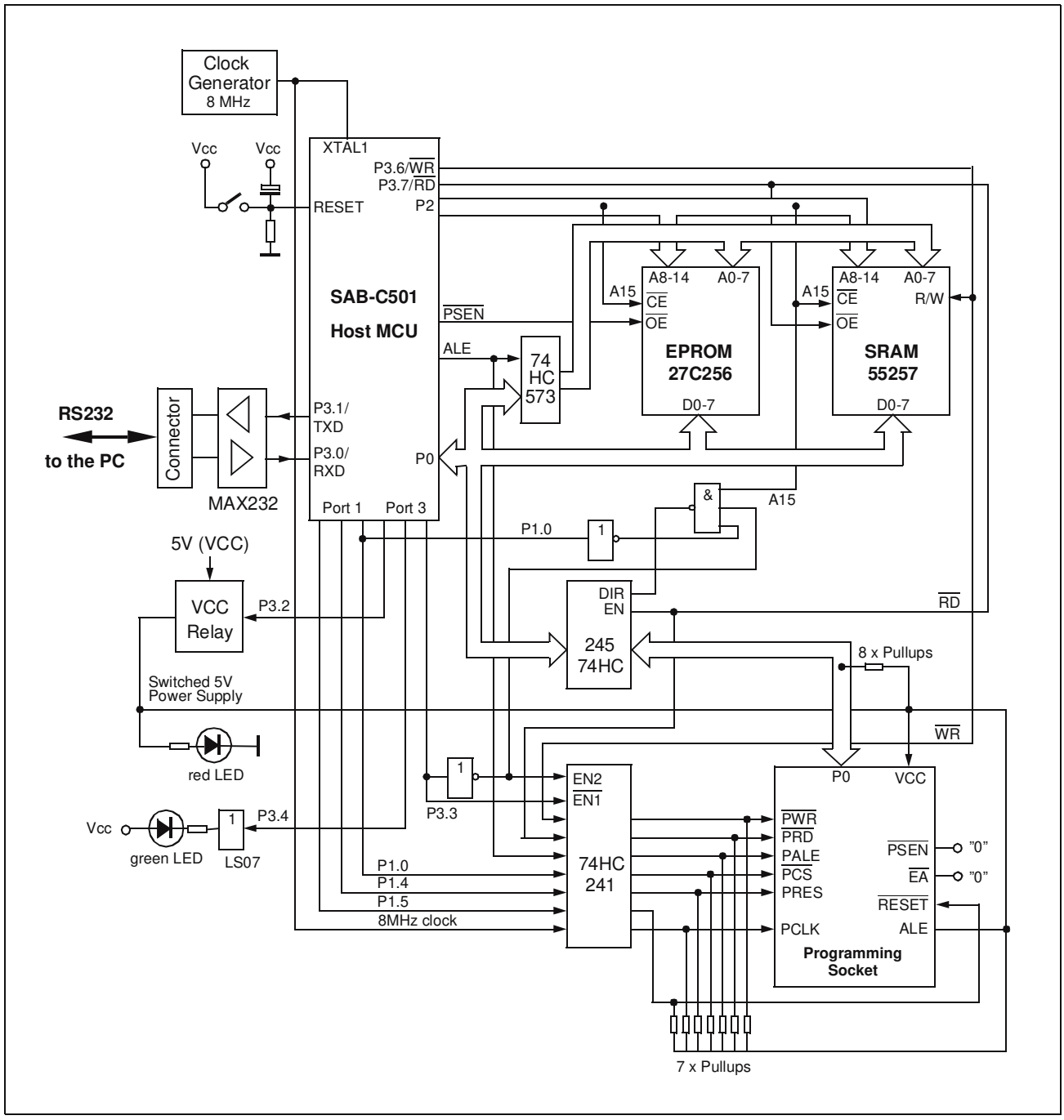


Figure 2 : Evaluation Programming System Diagram

Table 1 shows which function of the evaluation programming system is assigned to the specific SAB-C501 host controller port pins.

Table 1 : Functions of the MCU Port Pins

C501 Pins	Function	Comment
P1.0	Select programming socket	A “0“ selects the device inserted in the corresponding socket. With “1“ the device are deselected.
P1.4	PRES signal	A “1“ at this line resets the programming interface of the programming device.
P1.5	RESET polarity control	0 : SAB-C512A-H selected 1 : SAB-C513A-H selected
P3.2	VCC control of the programming sockets	With a “0“, the 5V VCC power supply of the programming socket is switched on.
P3.3	Enable control/data bus drivers of the programming socket	With a “0“, the bus drivers to the programming sockets are enabled.
P3.4	Green LED control	0: LED on, 1: LED off

4 User Interface Program

The SAB-C512A-H/SAB-C513A-H evaluation programming system includes two software parts : the host controller firmware for the host controller hardware and the MS-DOS user interface program (PRG51X.EXE), which is the human interface to the evaluation programming system. Both software parts communicate together via a RS232 standard asynchronous serial data interface. Therefore, a well defined software protocol must be handled between both software parts (detailed definition see section 5).

The user interface program operates command line oriented and provides typical programmer commands. It requires two parameters for operation : the number of display lines (25 or 43/50 lines) and the number of the PC COM port (1 or 2) used for communication. These two parameters are stored in a small configuration file. This configuration file (PRG51X.CFG) is created by the user interface program itself if it does not exist in the current directory.

For proper operation of PRG51X.EXE the serial interface of the PC (especially its interrupts) must be correctly configured. The user interface program uses the interrupt feature of the COM ports for the reception of data bytes from the host controller hardware. At the PC, COM1 must be connected to IRQ4 or COM2 must be connected to IRQ3.

4.1 Commands of the User Interface Program

The commands of the user interface program are listed in table 2.

The **"Display Programming RAM"** command allows to display the content of the programming RAM on the screen of the PC. A single RAM byte or a specified RAM address range can be displayed. The screen output can be stopped by pressing the Ctrl-S key and proceeded again by the Ctrl-Q key.

The **"Substitute/Change Programming RAM"** command allows to alter the programming RAM content bytewise with incrementing addresses or blockwise with a specific data value.

The **"Load Programming RAM with HEX/OBJ/BIN File"** command is used to transfer absolute located data files, which are typically created by the 8051 family compiler, assembler, and locater programs, into the programming RAM. In case of HEX or OBJ file format, the user interface program scans the specified data files for absolute code information and transfers it into the programming RAM. In case of BIN file format selection, the bytes of the specified data file are transferred sequentially into the programming RAM starting at the address which is specified in the command line.

The **"Save Programming RAM as BIN File"** command allows to save the content of a specifc area of the programming RAM in a BIN file. This file represents the content of the specified area byte by byte and can be loaded again as a BIN file into the programming by the "Load programming RAM" command.

The **"Program EEPROM from RAM"** command allows to program a specified area of the SAB-C512A-H/SAB-C513A-H EEPROM or the with the content of the programming RAM. An erase EEPROM operation may optionally preceed the programming operation. A verify EEPROM operation may optionally follow the programming operation if selected in the command line.

The **"Verify EEPROM to RAM"** command is used to verify the content of a specified area of the SAB-C512A-H/SAB-C513A-H device with the content of the programming RAM. If a verify error is detected, an error message is displayed and the verify operation can be continued or aborted.

The **"Read EEPROM into RAM"** command is used to transfer (read) a specified area of the EEPROM of the SAB-C512A-H/SAB-C513A-H device into the programming RAM.

The **"Erase EEPROM"** command allows to erase the complete EEPROM.

The **"Check EEPROM Device"** command allows to check the complete EEPROM for programmability. This check includes an erase, a write 00H data, a write 55H data, and a write AAH data operation. After the completion of the command a pass/fail message is displayed and the EEPROM is erased.

The **"Reset programmer"** command puts the evaluation programming system into an initilal state (software reset operation). This command also allows to select the device type - SAB-C512A-H or SAB-C513A-H.

Table 2 : Command Survey of the User Interface Program

Command Type	Command Synax	Description
Display Programming RAM	<D> <adr> <D> <adr1> <T> <adr2>	Display single RAM address Display RAM from “adr1” to “adr2”
Substitute/Change Programming RAM	<S> <adr> <S> <adr1> <T> <adr2> <W> <val>	Sequential change of RAM content beginning at address “adr”; a “,” selects next location, incrementing the address. Substitute RAM from “adr1” to “adr2” with “val”
Load Programming RAM with HEX/OBJ/BIN File	<L> <H> <filename> <L> <O> <filename> <L> <B> <filename> <S> <adr>	Load data of HEX-File “filename” into RAM Load data of OBJ-File “filename” into RAM Load data of BIN-File “filename” into RAM at start address “adr”
Save Programming RAM as BIN File	<T> <adr1> <T> <adr2> <I> <filename>	Save RAM data from “adr1” to “adr2” in BIN-File “filename”
Program EEPROM from RAM	<P> <adr1> <T> <adr2> {E} {V} <P> {E} {}V}	Program EEPROM with RAM content from address <adr1> to address <adr2> with optional erase or verify Program total EEPROM with optional erase or verify
Verify EEPROM to RAM	<V> <adr1> <T> <adr2> <V>	Verify EEPROM with RAM content from address <adr1> to address <adr2> Verify total EEPROM with RAM content
Read EEPROM into RAM	<R> <adr1> <T> <adr2> <R>	Read EEPROM content from address <adr1> to address <adr2> into RAM Read total EEPROM content
Erase EEPROM	<E>	Erase EEPROM
Reset Programmer	<X>	Software reset

4.2 Further Features of the User Interface Program

During the programming operation the Host generates a 16-bit (word) modulo-2 checksum (CS) over the data bytes which are read from the programming RAM. This checksum is displayed when the program command is terminated. During a verify operation, the Host also generates a CS checksum over the data bytes which are read from the programming RAM. So, if a verify after programming is selected, programming and verify CS checksum should be identical.
This feature allows to check whether data in the programming RAM is disturbed or changed during programming.

The last command, which was executed, is stored and can be executed again by pressing the “Cursor-up” key.

For debugging purposes, it is possible to display the bytes which are transfered between Host and PC on the screen. This debugging mode is invoking the User Interface Program by <PRG51X /D>. The yellow written bytes are the bytes which are transferref from the Host to the PC. The cyan written bytes are the bytes which are transfered from the PC to the Host.

The command survey shown in table 2 can be read online by the help feature of the user interface program.

5 Host System Interface Formats

The communication between the host system and the PC is fully controlled by the user interface program. This program controls as a master system the data transfer on the serial interface interconnection between PC and the host system. The host system always operates as a slave system.

The data transfer between host and PC has a well defined format which is described in this section. This host system interface format allows to use the SAB-C512A-H/SAB-C513A-H evaluation programming system also with other software programs which are written other hardware platforms than a standard PC.

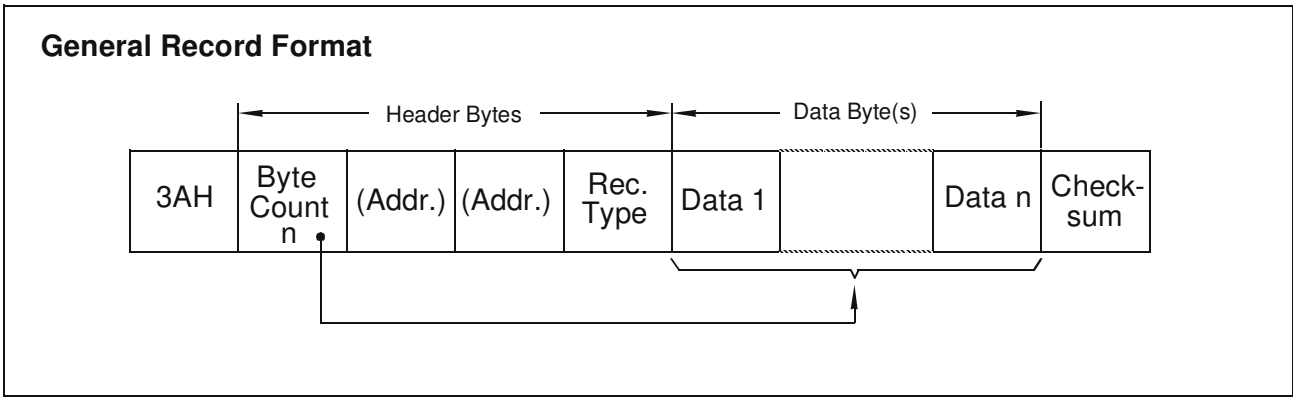


Figure 3 : General Record Format

The basic format of all records is build up as shown in figure 3. All records have the same construction. The records layout is related to the standard HEX file format. The header and data bytes are not coded as two ASCII characters as in a HEX file but as a single byte. Each record starts with a 3AH (ASCII ":") sign. The second byte of a record is the first byte of a four byte header block. This byte always defines number of data bytes which are transmitted in the data block. Depending on the specific record type, the second and third header byte contain a 16-bit address value (first MSB and then LSB) or a data information. The 4th header byte indicates the record type. After the header bytes, data byte(s) are transmitted in a data block. The number of data bytes can reach from 0 up to 255 as defined in the byte count field of the header block. The last byte of a record is the checksum byte. The checksum is build in the following way : adding all header and data bytes, truncating it to an 8 bit value, and subtracting it from 100H. For a check of the correct checksum, the 8-bit sum of all bytes of a record except the first byte (3AH) will give a 8-bit value of 0.

In the following description the host system is referenced as "Host" and the user interface program is abrivated as "PC".

5.1 Read Data Records

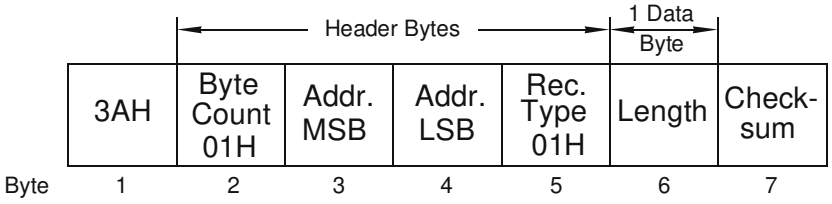
The Read Data Records are used to control a operation, at which the contents of the programming RAM are transferred to the PC. A read data operation is always executed in the two following steps :

1. The PC sends a Read Request Data Record to the Host.
2. The Host transfers with one Read Data Record as an answer back to the PC. If a checksum error was detected by the Host at the Read Request Data Record which was sent before, a Status Record with checksum error indication is sent back instead of the Read Data Record.

A Read Data Record may transfer up to 255 (FFH) data bytes in one record. If more than 255 bytes have to be read, multiple Read Request Data / Read Data Record transactions must be initiated by the PC.

5.1.1 Read Request Data Record

This record is sent by the PC to the Host. It contains a RAM start address and a length byte which informs the host to transfer a Read Data Record, defined by RAM start address and length, back to the PC. The format of the Read Request Data Record is shown in figure 4.



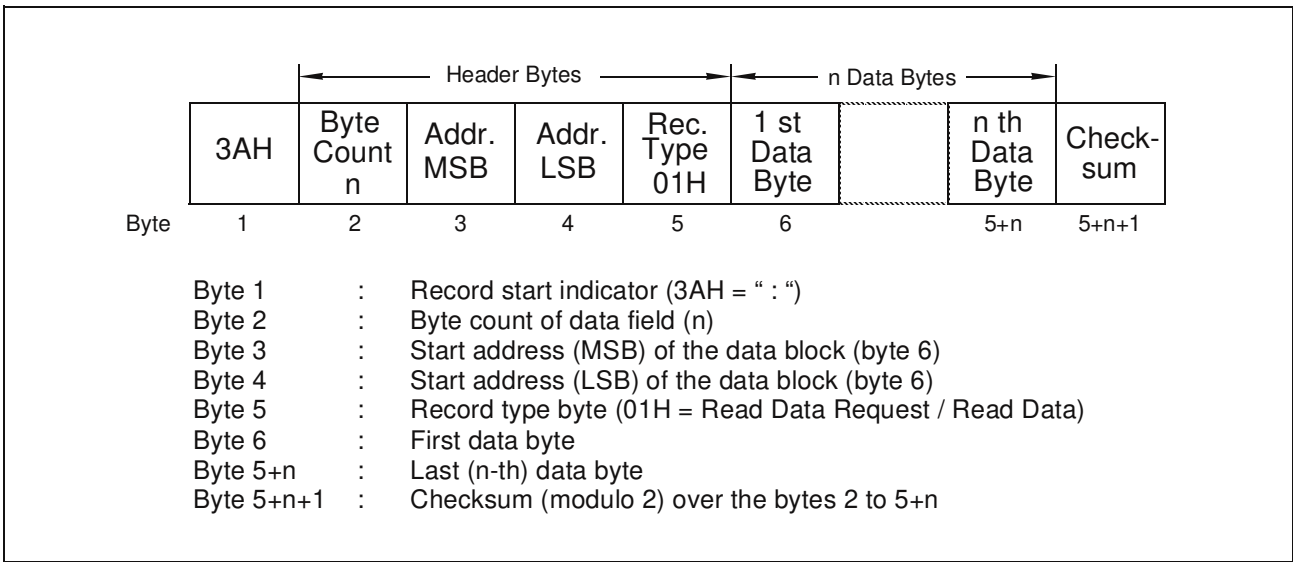
Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (01H; byte 6 is the single byte of the data field)
Byte 3 : Start address (MSB) of the requested data block
Byte 4 : Start address (LSB) of the requested data block
Byte 5 : Record type byte (01H = Read Data Request / Read Data)
Byte 6 : Number of bytes requested (01H to FFH)
Byte 7 : Checksum (modulo 2) over the bytes 2 to 6

Figure 4 : Read Data Request Record

The Read Request Data Record has the same record type (01H) as the Read Data Record.

5.1.2 Read Data Record

The Read Data Record is sent from the Host to the PC and transmits data of the programming RAM. It is the answer of the Host to a Read Request Data Record without checksum error. The header bytes contain the programming RAM start address and a byte count field which defines the number (n) of the data bytes which are transmitted within the Read Data Record. Up to 255 data bytes can be transmitted within one Read Data Record. The format of the Read Data Record is shown in the figure 5.

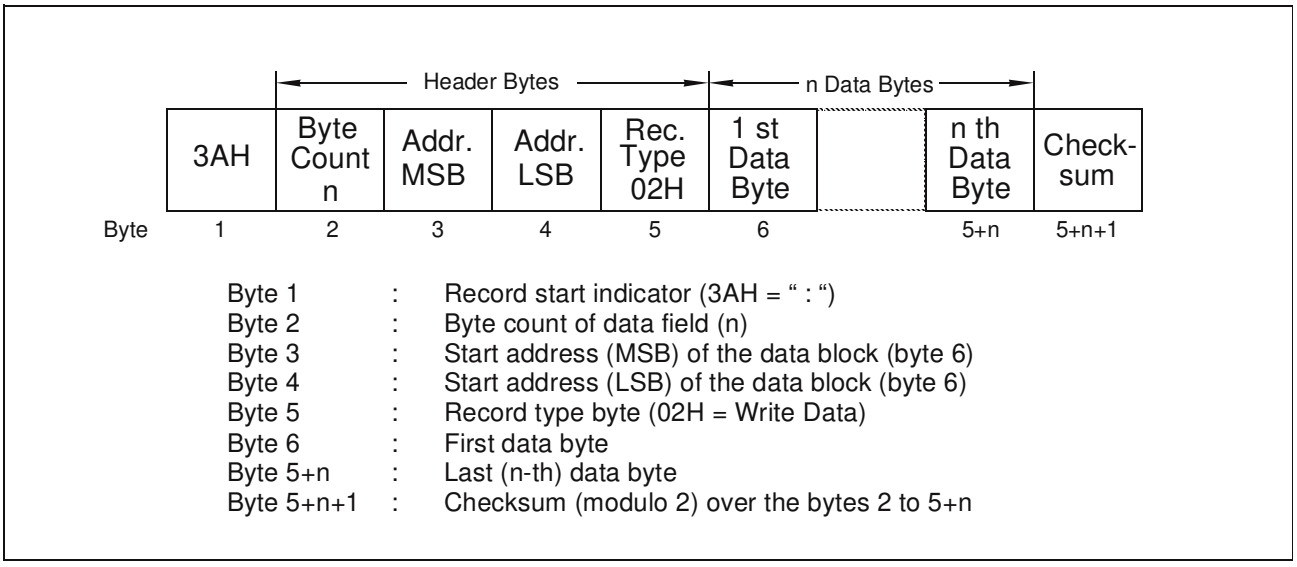


Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (n)
Byte 3 : Start address (MSB) of the data block (byte 6)
Byte 4 : Start address (LSB) of the data block (byte 6)
Byte 5 : Record type byte (01H = Read Data Request / Read Data)
Byte 6 : First data byte
Byte 5+n : Last (n-th) data byte
Byte 5+n+1 : Checksum (modulo 2) over the bytes 2 to 5+n

Figure 5 : Read Data Record

5.2 Write Data Record

The Write Data Record is sent from the PC to the Host. It informs the Host to place the transmitted data into the programming RAM beginning at the data block start address. The header bytes contain this data block start address and a byte count field which defines the number (n) of the data bytes which are transmitted. Up to 255 data bytes can be transmitted within one Write Data Record. The format of the Write Data Record is shown in figure 6.



Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (n)
Byte 3 : Start address (MSB) of the data block (byte 6)
Byte 4 : Start address (LSB) of the data block (byte 6)
Byte 5 : Record type byte (02H = Write Data)
Byte 6 : First data byte
Byte 5+n : Last (n-th) data byte
Byte 5+n+1 : Checksum (modulo 2) over the bytes 2 to 5+n

Figure 6 : Write Data Record

After the correct reception of a Write Data Record a status record of type "action terminated" is sent to the Host. If a checksum error was detected at the Write Data Record, a Status Record with checksum error indication is sent back.

If the programming RAM must be filled with constant data, a Fill Data Record is available. This record type allows a fast RAM write operation with constant data. Start and end address as well as the fill data value are transmitted within the Fill Data Record. When the Host has executed the fill data operation it sends a status record with type “action terminated“ back to the PC. If a checksum error was detected at the Fill Data Record, a Status Record with checksum error indication is sent back.The format of the Fill Data Record is shown in figure 7.

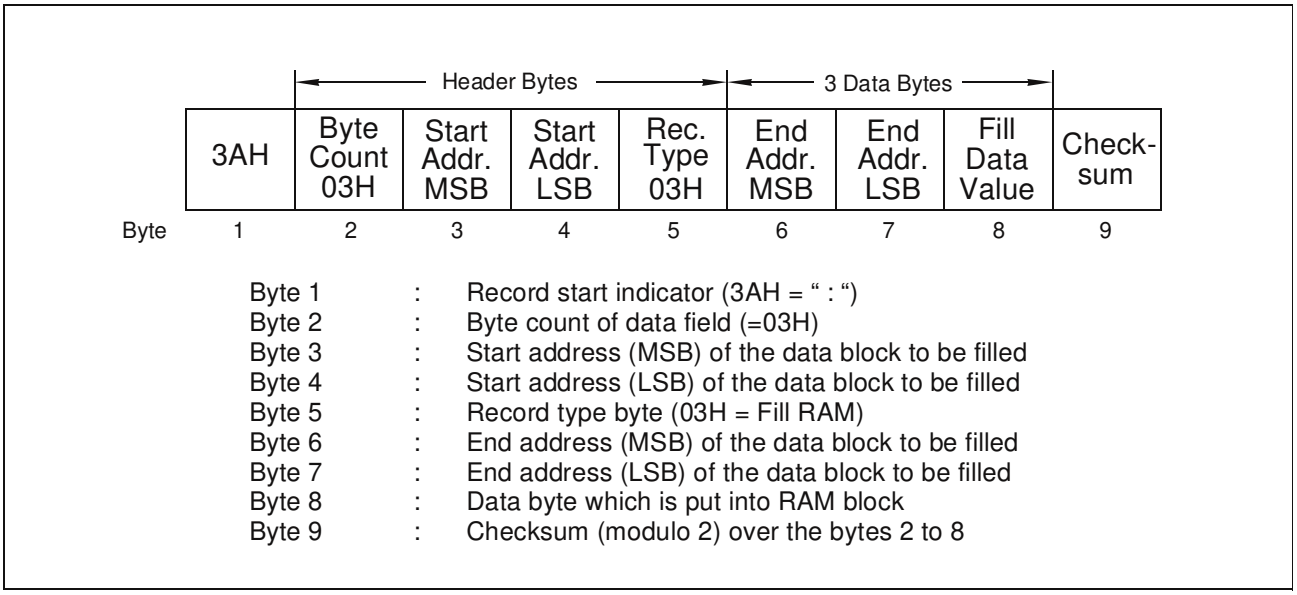
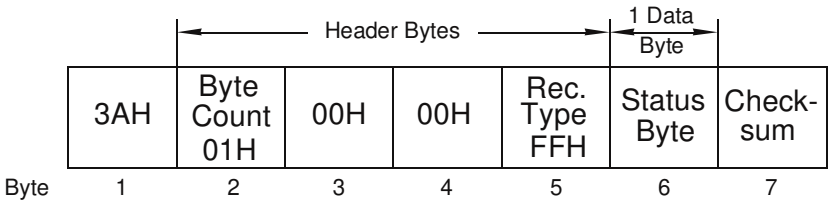


Figure 7 : Fill Data Record

5.3 Status Records

Status Records transfer specific status information between Host and PC. A Status Record is always indicated by the record type FFH. The status type is decoded in the data byte following the record type byte. For the verify and the check device operation an extended status record is available. The meaning of the status bytes is shown in the table of figure 8.

Status Record (PC --> Host and Host --> PC)



Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (01H; byte 6 is the single byte of the data field)
Byte 3 : Marker byte 00H (at status byte 10H : contains MSB of a CS checksum)
Byte 4 : Marker byte 00H (at status byte 10H : contains LSB of a CS checksum)
Byte 5 : Record type byte (FFH = Status)
Byte 6 : Status byte (definition see table below)
Byte 7 : Checksum (modulo 2) over the bytes 2 to 6

Status Byte	Direction	Description
80H	Host --> PC	Checksum error; sent by the Host if a checksum error of a record, which was sent by the PC is detected
10H	PC --> Host Host --> PC	Terminate action/task; sent by the PC to terminate a operation in the Host (e.g. abort verify operation) Action/task terminated; sent by the Host if a operation is terminated (e.g. Fill RAM operation); at the check device operation an extended status record with status byte 10H is used (see figure 13).
20H	PC --> Host	Request programmer status
30H 31H	Host --> PC	Programmer status, answer of the host to a request programmer status record 30H : no device in the socket 31H : device in the socket
40H	Host --> PC	256 bytes programmed/verified; sent by the host if 256 bytes have been programmed or verified
50H	PC --> Host	Succeed action/task; sent by the PC to succeed a operation in the host (e.g. resume verify operation)
60H	Host‹--> PC	Verify Error (further details see figure 9)
70H	PC --> Host	Switch off sockets from power supply

Figure 8 : Status Records

After the execution of a program EEPROM or verify EEPROM operation, the Host transmits a 10H status records to the PC which includes a CS checksum word in byte 3 (MSB) and byte 4 (LSB) instead of 00H. This checksum is the modulo-2 checksum of the data bytes which are processed during the programming (RAM content) or verify (EEPROM content) operation.

A special status record is the Verify Error Status Record. This type of status record contains 4 more data bytes than the “normal” status records. It is sent from the Host to the PC if an error is detected during a verify operation. The PC must respond to the reception of a Verify Error Status Record with a Status Record with either 10H (terminate action) or 50H (succeed action) as status byte.

The Verify Error Status Record (see figure 9) sends the address of the error location, the data which was read from the EEPROM of a SAB-C512A-H/SAB-C513A-H and the expected data from the programming RAM.

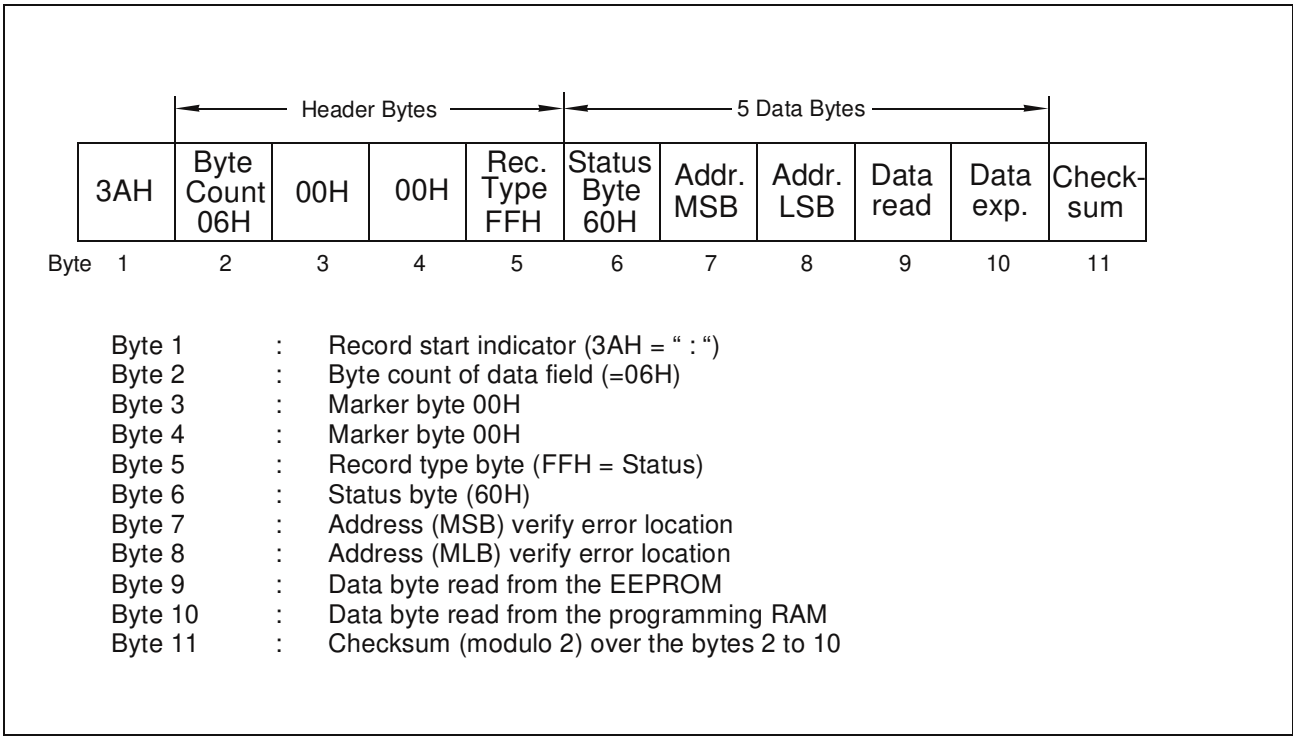


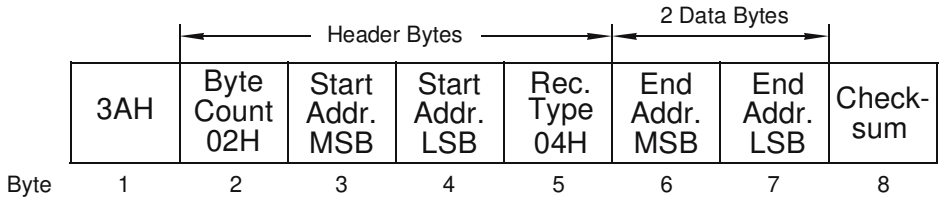
Figure 9 : Verify Error Status Record

5.4 Program Device Record

The Program Device Record is sent from the PC to the Host. It informs the Host about the RAM area which has to be programmed. Before issuing a Program Device Record, the PC should check the programmer status by sending a status record with status byte 20H (request programmer status) to the Host. The PC will get back from the Host a programmer status record that informs the PC whether the socket is equipped with a device or not. The Program Device Record should then only be issued if a device is inserted in the socket.

After sending a Program Device Record to the Host and during the programming operation, the PC will receive the following status records form the Host :

– 40H status record(s) - 256 bytes programmed; not sent if byte count to be programmed < 256
– Action terminated (status byte = 10H) at the end of the programming cycle; byte 3 and 4 of this status record contain a checksum word.



Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (=02H)
Byte 3 : Start address (MSB) of the data block to be programmed
Byte 4 : Start address (LSB) of the data block to be programmed
Byte 5 : Record type byte (04H)
Byte 6 : End address (MSB) of the data block to be programmed
Byte 7 : End address (LSB) of the data block to be programmed
Byte 8 : Checksum (modulo 2) over the bytes 2 to 2

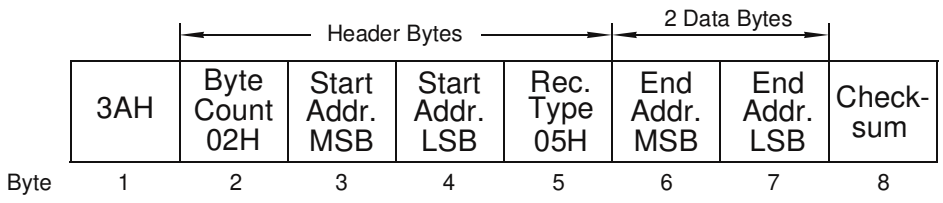
Figure 10 : Program Device Record

5.5 Verify Device Record

The Verify Device Record is sent from the PC to the Host. It informs the Host about the data block which has to be verified. Before issuing a Verify Device Record, the PC should check the programmer status by sending a status record with status byte 20H (request programmer status) to the Host. The PC will get back from the Host a programmer status record that informs the PC whether the socket is equipped with a device or not. The Verify Device Record should then only be issued if a device is inserted in the socket.

After sending a Verify Device Record to the Host and during the verify operation, the PC is informed by the Host about the verify procedure with the following status records :

– 40H status record(s) - 256 bytes verified; not sent if byte count to be verified < 256
– Action terminated (status byte = 10H) at the end of the verify operation; at both status records, byte 3 and 4 contain a checksum word.



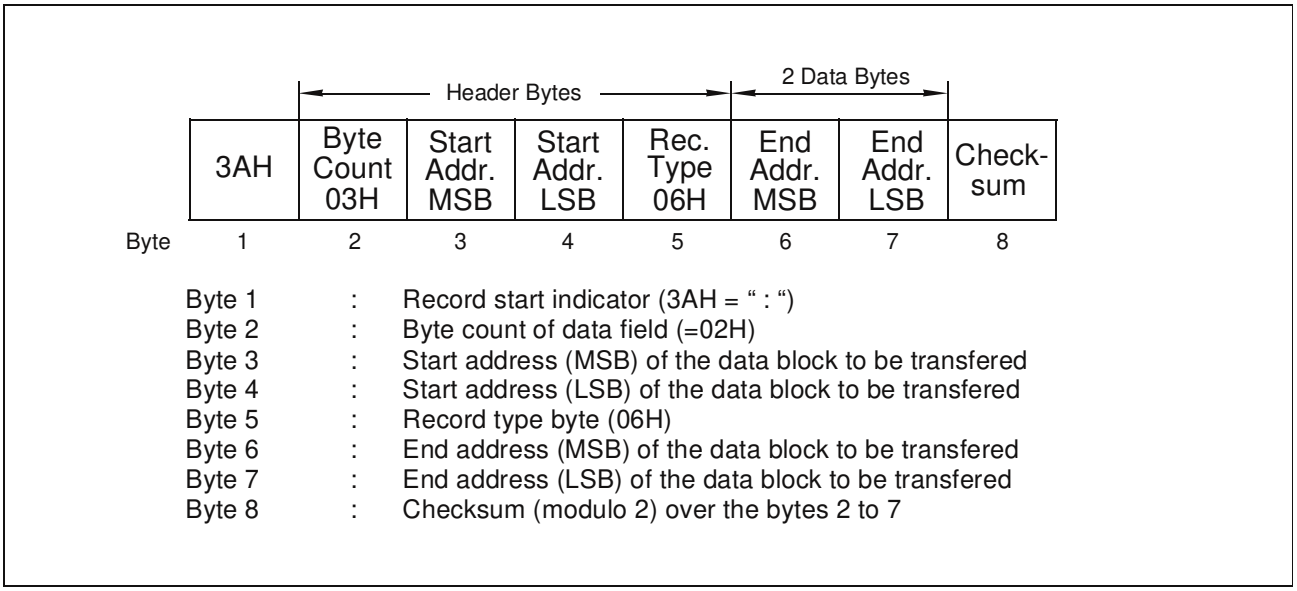
Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (=02H)
Byte 3 : Start address (MSB) of the data block to be verified
Byte 4 : Start address (LSB) of the data block to be verified
Byte 5 : Record type byte (05H)
Byte 6 : End address (MSB) of the data block to be verified
Byte 7 : End address (LSB) of the data block to be verified
Byte 8 : Checksum (modulo 2) over the bytes 2 to 7

Figure 11 : Verify Device Record

5.6 Transfer Device-to-RAM Record

The Transfer Device-to-RAM Record is sent from the PC to the Host. It informs the Host to read (transfer) the EEPROM contents of the specified data block from the device into the programming RAM. Before issuing a Transfer Device-to-RAM Record, the host should check the programmer status by sending a status record with status byte 20H (request programmer status). The PC will get back from the Host a programmer status record that informs the PC that informs the PC whether the socket is equipped with a device or not. The Transfer Device-to-RAM Record should then only be issued if a device is inserted in the socket.

When the Host has executed the read data operation it send a status record (record type 10H - action terninated) back to the PC.



Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (=02H)
Byte 3 : Start address (MSB) of the data block to be transfered
Byte 4 : Start address (LSB) of the data block to be transfered
Byte 5 : Record type byte (06H)
Byte 6 : End address (MSB) of the data block to be transfered
Byte 7 : End address (LSB) of the data block to be transfered
Byte 8 : Checksum (modulo 2) over the bytes 2 to 7

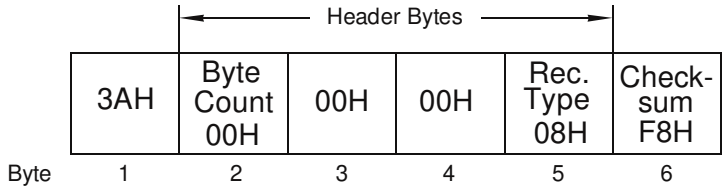
Figure 12 : Transfer Device-to-RAM Record

5.7 Check Device Record

The Check Device Record is sent from the PC to the Host. It informs the Host to check the programmability of the EEPROM of the SAB-C512A-H/SAB-C513A-H device. Before issuing a Check Device Record, the PC should check the programmer status by sending a status record with status byte 20H (request programmer status) to the Host. The PC will get back from the Host a programmer status record that informs the PC whether the socket is equipped with a device or not. The Check Device Record should then only be issued if a device is inserted in the socket.

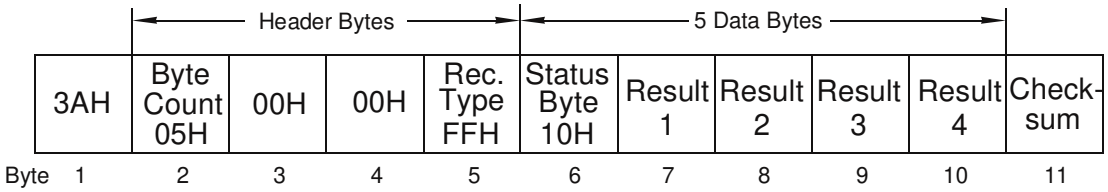
When the Host has executed the EEPROM check operation it send an extended status record (record type 10H - action terninated) back to the PC. This record includes the test result of the following tests : clear test (content=FFH), write 00H test, write 55H test, write AAH test.

Check Device Record (PC --> Host)



Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (=00H)
Byte 3 : Marker byte 00H
Byte 4 : Marker byte 00H
Byte 5 : Record type byte (08H = Erase)
Byte 6 : Checksum (modulo 2) over the bytes 2 to 5 (=F8H)

Extended Status Record (Host --> PC)
(only after Check Device)



Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (=05H)
Byte 3 : Marker byte 00H
Byte 4 : Marker byte 00H
Byte 5 : Record type byte (FFH = Status Record)
Byte 6 : Status Byte (10H = action terminated)
Byte 7 : Result Byte 1 : Erase test (00H=pass, FFH=fail)
Byte 8 : Result Byte 2 : Write 00H test (00H=pass, FFH=fail)
Byte 9 : Result Byte 3 : Write 55H test (00H=pass, FFH=fail)
Byte 10 : Result Byte 4 : Write AAH test (00H=pass, FFH=fail)
Byte 11 : Checksum (modulo 2) over the bytes 2 to 10

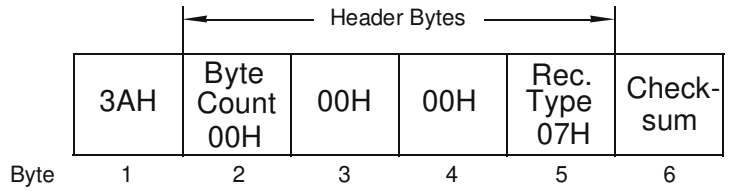
Figure 13 : Check Device Record

5.8 Erase Record

The Erase Record is sent from the PC to the Host. It informs the Host to erase the EEPROM of the SAB-C512A-HSAB-C513A-H device. Before issuing a Erase Record, the PC should check the programmer status by sending a status record with status byte 20H (request programmer status) to the Host. The PC will get back from the Host a programmer status record that informs the PC whether the socket is equipped with a device or not. The Erase Record should then only be issued if a device is inserted in the socket.

When the Host has executed the erase EEPROM operation it send a status record (record type 10H - action terninated) back to the PC.

Erase Record (PC ‹Host)



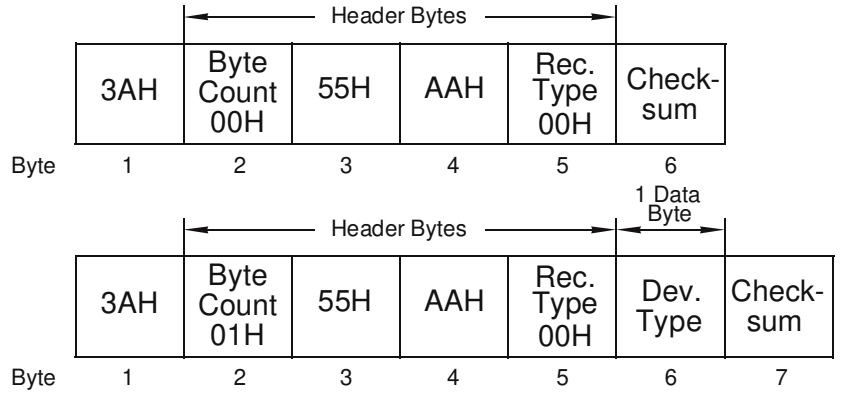
Byte 1	:	Record start indicator (3AH = “ : “)
Byte 2	:	Byte count of data field (=00H)
Byte 3	:	Marker byte 00H
Byte 4	:	Marker byte 00H
Byte 5	:	Record type byte (07H = Erase)
Byte 6	:	Checksum (modulo 2) over the bytes 2 to 5

Figure 14 : Erase Record

5.9 Reset Record

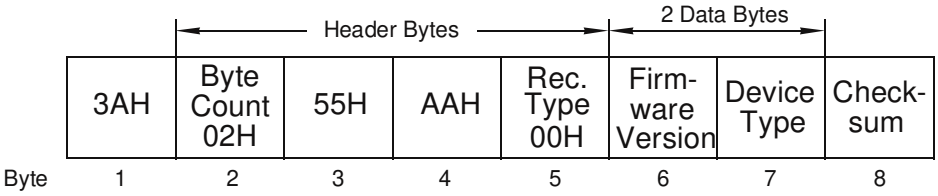
The Reset Record (type C) is sent to the PC if a hardware reset is executed at the Host. The PC may also reset the Host system by sending a Reset Record of type A or B to the host which is again responded by the Host with a Reset Record of type C.

Reset Records (PC --> Host)



Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (=00H or 01H)
Byte 3 : Marker byte 55H
Byte 4 : Marker byte AAH
Byte 5 : Record type byte (00H = Reset)
Byte 6 : Optional device type select byte : 00H = SAB-C512A-H
01H = SAB-C513A-H
Byte 6,7 : Checksum (modulo 2) over the bytes 2 to 5, 6

Reset Record (Host --> PC)



Byte 1 : Record start indicator (3AH = " : ")
Byte 2 : Byte count of data field (=02H)
Byte 3 : Marker byte 55H
Byte 4 : Marker byte AAH
Byte 5 : Record type byte (00H = Reset)
Byte 6 : Firmware Rrevision byte (e.g. 13H = Version 1.3)
Byte 7 : Selected device type : 00H = SAB-C512A-H
01H = SAB-C513A-H
Byte 8 : Checksum (modulo 2) over the bytes 2 to 7

Figure 15 : Reset Records

6 Program-Listings

This chapter shows the listings of the host control program (firmware) of the SAB-C512A-H/SAB-C513A-H evaluation programming system. The host control program is written in C51 (Keil). Some routines are implemented in 8051 assembler code.

```
/*===========================================================================*/
/*                                                                           */
/*     C51 - SAB-C512A-H/C513A-H  Programmer System HOST Control Program     */
/*                                                                           */
/*     Program controls the C512/C513 evaluation programming system. It      */
/*     handles the serial interface to the PC and controls the EEPROM        */
/*     accesses.                                                             */
/*                                                                           */
/*     (c) SIEMENS AG    HL MCB PD/AT 1     1995                             */
/*     Compiler : Keil C-Compiler Rev 3.2                                    */
/*                                                                           */
/*===========================================================================*/

#pragma code symbols debug oe

/*      Declaration of the required SFR-Symbols                              */

sfr SCON  = 0x98;
sfr SBUF  = 0x99;
sfr T2CON = 0xC8;
sfr RC2L  = 0xCA;
sfr RC2H  = 0xCB;
sfr TL2   = 0xCC;
sfr TH2   = 0xCD;

/*      Declaration of HW Bits                                               */

sbit TR2  = 0xCA;
sbit RI   = 0x98;
sbit TI   = 0x99;
sbit vcc_control   = 0xb2;    /*  0: Vcc on,  1: Vcc off                     */
sbit enable_driver = 0xb3;    /*  0: driver enabled, 1: driver disabled      */
sbit pres          = 0x94;    /*  0: inaktiv, 1: aktiv                       */
sbit pcs           = 0x90;    /*  0: aktiv, 1:inaktiv                        */
sbit led           = 0xb4;    /*  0: LED on, 1: LED off                      */
sbit reset         = 0x95;    /*  0: C512A-H   1: C513A-H                    */

/*      Declarations of variables                                            */

unsigned char ch, checksum;
unsigned char reclength, adrh, adrl, type;
unsigned char lenl, lenh, val;
unsigned char xdata buf[0x3000];
unsigned int adr, len;

/*      Definitions of literals                                              */

#define checksum_error    0x80
#define action_terminated 0x10
#define request_status    0x20
#define bytes256_handled  0x40

/*      Definitions of C512A-H/C513A-H programming register addresses        */

#define pcntrl 0x00
#define pmsel  0x01
#define pdata  0x02
#define paddrl 0x03
#define paddrh 0x04

/*      Definitions of external assembler routines                           */

extern void waitxms (char ms);                    /* External delay for x ms */
extern void wait20us ();                          /* External delay of 20 us */
extern char read_eeprom (char adr);               /* Read EEPROM register    */
extern void write_eeprom (char adr, char val);    /* Write EEPROM register   */

/*===========================================================================*/
/*      Procedures                                                           */
/*===========================================================================*/
```

```
void out_char (char c)                          /* Serial port output routine */
{
    while (!TI);                                /* wait for tranamitter empty */
    waitxms (1);                                /* wait 1 ms (for slow PC)    */
    TI = 0;                                     /* clear TI                   */
    SBUF = c;                                   /* write character to SBUF    */
}

unsigned char in_char ()                        /* Serial port input routine  */
{
    while (!RI);                                /* wait for receiver readz    */
    RI = 0;                                     /* clear RI                   */
    return (SBUF);                              /* read SBUF and return value */
}

/*============================================================================*/

void send_reset_frame ()             /* routine sends a reset record to host */
{
    out_char (':');
    out_char (0x02);
    out_char (0x55);
    out_char (0xaa);
    out_char (0x00);
    out_char (0x10);                /* revision code 1.0                      */
    if (reset)
    {   out_char (0x01);            /* SAB-C513A-H selected                   */
        out_char (0xee);
    }
    else
    {   out_char (0x00);            /* SAB-C512A-H selected                   */
        out_char (0xef);
    }
}

void send_status_frame (unsigned int w, char s)
{                                   /* routine sends a status frame to host */
    unsigned char cs;               /* w: address, s: status byte           */
    out_char (':');
    out_char (0x01);
    out_char (w >> 8);
    out_char (w);
    out_char (0xff);
    out_char (s);
    cs = (w >> 8) + w + s;
    out_char (0x100 - cs);
}

void activate_socket ()
{                                       /* routine enables programming socket */
    if (vcc_control==1)                 /* VCC is switched on and drivers for */
    {   vcc_control = 0;                /* the progr. device are enabled      */
        waitxms (10);
        enable_driver = 0;
        waitxms (10);
    }
    pres = 1;                           /* programming interface is reset     */
    waitxms (10);
    pres = 0;
    led = 1;                            /* turn green LED off                 */
}

void release_socket ()
{                                       /* routine disables prog. socket      */
    enable_driver = 1;                  /* disable drivers                    */
    vcc_control = 1;                    /* switch off socket VCC              */
    waitxms (10);
    led = 0;                            /* turn green LED on                  */
}
unsigned char detect_socket ()          /* routine checks whether a device is */
{                                       /* in the socket or not               */
    unsigned char erg;
    activate_socket ();
    write_eeprom (paddrl,0x55);         /* write test byte in paddrl register */
    erg = read_eeprom (paddrl);         /* read back test byte                */
    if (erg != 0x55) return (0x30);
    else return (0x31);
}
```

```
void send_verify_error_frame (unsigned int adr,unsigned char expval, actval)
{                                       /* routine outputs an extended status */
    out_char (':');                     /* byte record for verify error       */
    out_char (0x05);                    /* adr: address; expval: expected val;*/
    out_char (0x00);                    /* actval: value in EEPROM            */
    out_char (0x00);
    out_char (0xff);
    out_char (0x60);
    out_char (adr >> 8);
    out_char (adr);
    out_char (expval);
    out_char (actval);
    checksum = 0x64 + (adr >> 8) + adr + expval + actval;
    out_char (0x100 - checksum);
}

unsigned char get_status_frame ()
{                                       /* routine receives a complete status */
    unsigned char l,a1,a2;              /* record and returns the status byte */
    l  = in_char ();
    l  = in_char ();
    a1 = in_char ();
    a2 = in_char ();
    checksum = l + a1 + a2;
    a1 = in_char ();
    l  = in_char ();
    a2 = in_char ();
    checksum = checksum + a1 + l + a2;
    return (l);
}

void erase_eeprom ()
{                                       /* routine erases EEPROM content      */
    write_eeprom (pcntrl,0x00);
    write_eeprom (pmsel ,0x19);
    write_eeprom (pcntrl,0x03);
    waitxms (10);
    write_eeprom (pcntrl,0x02);
    wait20us ();
    write_eeprom (pcntrl,0x00);
}

void write_block (unsigned char pms,dat, unsigned int adr)
{                                          /* routine writes the complete     */
    write_eeprom (paddrl,adr);             /* EEPROM with data                */
    write_eeprom (paddrh,(adr >> 8));      /* pms: PMSEL value; dat: data to  */
    write_eeprom (pcntrl,0x00);            /* be written; adr: address value  */
    write_eeprom (pmsel,pms);
    write_eeprom (pdata,dat);
    write_eeprom (pcntrl,0x05);
    waitxms (10);
    write_eeprom (pcntrl,0x04);
    wait20us ();
    write_eeprom (pcntrl,0x00);
}

unsigned char check_device (unsigned char low,high)
{                                       /* routine checks the complete EEPROM  */
    unsigned char l,erg;                /* for a value at even and odd address */
    unsigned int k;                     /* low: exp.value at all even addresses*/
    erg = 0;                            /* high:exp.value at all odd addresses */
    write_eeprom (paddrh,0);            /* erg=0: check ok; erg=1:check failed */
    write_eeprom (paddrl,0);
    write_eeprom (pcntrl,0x88);
    k = 0;
    while (k < 0x3000)
    {   l = read_eeprom (pdata);
        if ((k & 0x01 == 0) & (l != low)) erg = 0xff;
        if ((k & 0x01 == 1) & (l != high)) erg = 0xff;
        k++;
    }
    return (erg);
}


/*=========================================================================*/
```

```
void soft_reset ()                      /* routine is executed after reception*/
{                                       /* of a reset record                  */
    unsigned char tt;
    if (reclength == 0x01)
    {   tt = in_char ();
        checksum = checksum + tt;
    }
    checksum = checksum + in_char ();
    if (checksum == 0)
    {   if (reclength == 0x01)
        {   if (tt == 0x00) reset = 0;
            if (tt == 0x01) reset = 1;
        }
        send_reset_frame ();
    }
    elsesend_status_frame (0,checksum_error);
}

void read_ram ()                        /* read RAM routine : reads RAM and   */
{                                       /* transfers data to serial port      */
    unsigned char i,j,len;
    len = in_char ();
    j = in_char ();
    checksum = checksum + len + j;
    if (checksum != 0)
        send_status_frame (0,checksum_error);
    else
    {   out_char (':');
        out_char (len);
        out_char (adrh);
        out_char (adrl);
        out_char (0x01);
        checksum = len + adrh + adrl + 0x01;
        for (i = 0; i < len; i++)
        {   j = buf [adr + i];
            out_char (j);
            checksum = checksum + j;
        }
        out_char (0x100 - checksum);
        ch = 0;
    }
}

void write_ram ()                       /* write RAM routine : writes received*/
{                                       /* data into RAM                      */
    unsigned char j;
    unsigned int b, q;
    b = 0;
    q = reclength;
    for (b = 0; b != q; b++)
    {   j = in_char ();
        buf [adr + b] = j;
        checksum = checksum + j;
    }
    j = in_char ();
    checksum = checksum + j;
    if (checksum != 0) send_status_frame (0,checksum_error);
    else send_status_frame (0,action_terminated);
}

void fill_ram ()                        /* fills RAM with constant data       */
{
    unsigned char j;
    unsigned int w;
    lenh = in_char ();
    lenl = in_char ();
    val  = in_char ();
    len  = lenh;
    len  = (lenh << 8) + lenl;
    j = in_char ();
    checksum = checksum + lenl + lenh + val + j;
    if (checksum != 0) send_status_frame (0,checksum_error);
    else for (w=adr; w <= len; w++) buf [w] = val;
    send_status_frame (0,action_terminated);
}
```

```
void program_device ()                  /* writes EEPROM with data from RAM   */
{
    unsigned char c;
    unsigned int k,psum;
    lenh = in_char ();
    lenl = in_char ();
    len  = lenh;
    len  = (lenh << 8) + lenl;
    checksum = checksum + lenl + lenh + in_char ();

    if (checksum != 0) send_status_frame (0,checksum_error);
    else
    {   activate_socket ();
        c = 0;
        psum = 0;
        for (k = adr; k <= len; k++)
        {   write_eeprom (paddrh,(k >> 8));
            write_eeprom (paddrl,k);
            write_eeprom (pmsel,0);
            write_eeprom (pdata,buf[k]);
            psum = psum + buf[k];
            write_eeprom (pcntrl,0x05);
            waitxms (10);
            write_eeprom (pcntrl,0x04);
            wait20us ();
            write_eeprom (pcntrl,0x00);
            c++;
            if (c == 0) send_status_frame (0,bytes256_handled);
        }
        send_status_frame (psum,action_terminated);
    }

}

void verify_device ()                   /* verifies the data in RAM with the  */
{                                       /* data in the EEPROM                 */
    unsigned char b,c,result;
    unsigned int k,psum;
    lenh = in_char ();
    lenl = in_char ();
    len  = lenh;
    len  = (lenh << 8) + lenl;
    checksum = checksum + lenl + lenh + in_char ();
    if (checksum != 0) send_status_frame (0,checksum_error);
    else
    {   activate_socket ();
        c = 0;
        psum = 0;
        k = adr;
        while (k <= len)
        {   write_eeprom (paddrh,(k >> 8));
            write_eeprom (paddrl,k);
            write_eeprom (pcntrl,0x88);
            result = read_eeprom (pdata);
            psum = psum + result;
            c++;
            if (buf[k] != result)           /* error detected*/
            {   send_verify_error_frame (k,result,buf[k]);
                b = get_status_frame ();
                if (b == 0x10) k = len;     /* abort verify  */
                else
                if (b == 0x50)              /* succed verify */
                {   if (c == 0)
                    send_status_frame(0,bytes256_handled);
                }
            }
            else
            {   if (c == 0)
                send_status_frame(0,bytes256_handled);
            }
            k++;
        }
        send_status_frame (psum,action_terminated);
    }
}
```

```
void transfer_device ()                 /* reads content of EEPROM into the   */
{                                       /* programming RAM                    */
    unsigned int k;
    lenh = in_char ();
    lenl = in_char ();
    len  = lenh;
    len  = (lenh << 8) + lenl;
    checksum = checksum + lenh + lenl + in_char ();
    if (checksum != 0) send_status_frame (0,checksum_error);
    else
    {   activate_socket ();
        write_eeprom (paddrh,adrh);
        write_eeprom (paddrl,adrl);
        write_eeprom (pcntrl,0x88);
        for (k = adr; k <= len; k++) buf[k] = read_eeprom (pdata);
        send_status_frame (0,action_terminated);
    }
}

void erase_device ()                    /* erases the complete EEPROM         */
{
    checksum = checksum + in_char ();
    if (checksum != 0) send_status_frame (0,checksum_error);
    else
    {   activate_socket ();
        erase_eeprom ();
        send_status_frame (0,action_terminated);
    }
}

void test_device ()                     /* tests the EEPROM for programabilty  */
{
    unsigned char r1,r2,r3,r4;
    r1 = in_char ();
    checksum = checksum + r1;
    if (checksum != 0) send_status_frame (0,checksum_error);
    else
    {   activate_socket ();
        erase_eeprom ();
        r1 = check_device (0xff,0xff);              /* check EEPROM for FFH  */
        write_block (0x19,0x00,0);
        r2 = check_device (0x00,0x00);              /* check EEPROM for 00H  */
        erase_eeprom ();
        write_block (0x19,0x55,0);
        r3 = check_device (0x55,0x55);              /* check EEPROM for 55H  */
        erase_eeprom ();
        write_block (0x19,0xaa,0);
        r4 = check_device (0xaa,0xaa);              /* check EEPROM for AAH  */
        erase_eeprom ();
        out_char (':');                             /* output extended status*/
        out_char (0x05);                            /* record with result    */
        out_char (0x00);
        out_char (0x00);
        out_char (0xff);
        out_char (0x10);
        out_char (r1);
        out_char (r2);
        out_char (r3);
        out_char (r4);
        checksum = 0x14 + r1 + r2 + r3 + r4;
        out_char (0x100 - checksum);
    }
}

void handle_status ()                   /* handles a status byte from the PC */
{
    unsigned char status;
    status = in_char ();
    checksum = checksum + status + in_char ();
    if (checksum != 0) send_status_frame (0,checksum_error);
    else
    {   if (status == 0x20) send_status_frame (0,detect_socket());
        else
        if (status == 0x70) release_socket ();
    }
}
```

```
void init_led ()                        /* routines toggles the green LED    */
{
    led = 0; waitxms (100);
    led = 1; waitxms (100);
    led = 0; waitxms (100);
    led = 1; waitxms (100);
    led = 0; waitxms (100);
    led = 1; waitxms (100);
    led = 0;
}

/*=========================================================================*/
/*      Main programm                                                      */
/*=========================================================================*/

main ()
{
    SCON  = 0x5F;                       /* Mode 1, 8-bit, variable baud rate     */
    T2CON = 0x30;                       /* Timer 2 as baud rate generator        */
    RC2L  = 0xE6;                       /* Reload value : -1AH = 26D = 9600 Baud */
    RC2H  = 0xFF;
    TL2   = 0xE6;
    TH2   = 0xFF;
    TR2   = 1;                          /* start timer 2                         */

    init_led ();                        /* toggle green LED                      */
    send_reset_frame ();
    ch = 0;
    while (1)                           /* endless loop with record type select  */
    {
        while (ch != ':') ch = in_char ();
        reclength = in_char ();
        adrh = in_char ();
        adrl = in_char ();
        adr = adrh;
        adr = (adr << 8) + adrl;
        type = in_char ();
        checksum = reclength + adrh + adrl + type;
        switch (type)
        {
            case 0x00: soft_reset (); break;
            case 0x01: read_ram (); break;
            case 0x02: write_ram (); break;
            case 0x03: fill_ram (); break;
            case 0x04: program_device (); break;
            case 0x05: verify_device (); break;
            case 0x06: transfer_device (); break;
            case 0x07: erase_device (); break;
            case 0x08: test_device (); break;
            case 0xff: handle_status (); break;
        }
        ch = 0;
    }
}
```

```
$SYMBOLS
$DEBUG
;=============================================================================
;       SAB-C512/C513A-H Prototype evaluation programming system
;       Assemblerroutines : Write C512/C513, Read C512/C513, variable 1ms and
;       10us delay
;       Author : R.Schmid / Siemens AG
;       Assembler : Keil A51 Rev.4.4
;       Rev.   : 1.0 / 19.1.95
;=============================================================================
;
PUBLIC _waitxms             ;external declarations
PUBLIC wait20us
PUBLIC _write_eeprom
PUBLIC _read_eeprom
;
;=============================================================================
;       PROGRAM CODE
;=============================================================================

        HOST_ASSEMBLER_DATASEGMENT DATA
        RSEG   HOST_ASSEMBLER_DATA
        PARA1: DS     1
        PARA2: DS     1

        HOST_ASSEMBLER_CODESEGMENTCODE
        RSEG   HOST_ASSEMBLER_CODE

_read_eeprom:
        MOV    P2,#0F0H             ;load address into P2/R0
        MOV    A,R7
        MOV    R0,A
        CLR    P1.0                 ;clear P1.0 (PCS)
        MOVX   A,@R0                ;read C512
        SETB   P1.0                 ;set P1.0 (PCS)
        MOV    R7,A
        RET

_write_eeprom:
        MOV    P2,#0F0H             ;load address into P2/R0
        MOV    A,R7
        MOV    R0,A
        MOV    A,R5                 ;load ACC with write value
        CLR    P1.0                 ;clear P1.0 (PCS)
        MOVX   @R0,A                ;write C512
        SETB   P1.0                 ;set P1.0 (PCS)
        RET

_waitxms:
D1:     MOV    PARA1,#03H           ;outer loop = 6 + 2   = 8   cycles
D2:     MOV    PARA2,#6CH           ;inner loop = 216 + 2 = 218 cycles (x3)
        DJNZ   PARA2,$              ;              total :  662 cycles
        DJNZ   PARA1,D2      ;                     = 1002 us @ 8MHz
        NOP                         ;2 cycles correction
        NOP
        DJNZ   R7,D1
        NOP
        RET

wait20us:
        NOP                         ;4 (entry/exit) + 9 = 13 cycles =
        NOP                         ;19.5 us @ 8 MHz
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        RET                         ;20 usec

;=============================================================================
;       Identification String for EPROM Code
;
        DB     'SIEMENS AG - HL MCB PD/AT 1  C512A-H/C513A-H Firmware V1.0'
        DB     '  19.1.1995'

END
```