SIEMENS

Bootstrap Loader

8xC166

Microcomputer Components

8xC166
SAB 80C166 Family ApNotes

Application Note of the
On-chip Bootstrap Loader

Advance Information 9.94

SIEMENS

Edition 9.94 (Advance Information)

This edition was realized using the software system FrameMaker®.

Published by Siemens AG, Bereich Halbleiter,
Balanstraße 73, D-8000 München 80

As far as patents or other rights of third parties are concerned, liability is only assumed for components per se, not for applications, processes and circuits implemented within components or assemblies.

The information describes the type of component and shall not be considered as assured characteristics.

Terms of delivery and rights to change design reserved.

For questions on technology, delivery, and prices please contact the Offices of Semiconductor Group in Germany or the Siemens Companies and Representatives worldwide.

Due to technical requirements components may contain dangerous substances. For information on the type in question please contact your nearest Siemens Office, Semiconductor Group.

Siemens AG is an approved CECC manufacturer.

Contents

1 Introduction

In the SAB 8xC166(W), an on-chip bootstrap loader (BSL) is implemented and introduced with the CB-step. The BSL code is stored in a special Boot-ROM. With the BSL it is possible to load a program of 32 bytes into the internal RAM of the SAB 8xC166 via the Serial Port 0 (ASC0), even if there is no internal or external program memory available. This short program can be used to load extensive user software to internal RAM or external memory.

Note that not all emulators support emulation of the BSL feature of the SAB 8xC166.

2 Purpose of a bootstrap loader

As already mentioned, it is possible to load a program to the internal or external memory of the SAB 8xC166 via the BSL. Fundamentally there is distinguished between three different kinds of software applications:

I) The BSL loads the basic software to the system
The system includes no software. The whole application software is loaded to the system via the BSL.
Typical application: > End of line programming
> (Monitor for the Ertec EVA board)

II) The BSL loads temporary software to the system
The system includes already the complete application software, but only for special tests an additional software is necessary. This ‘temporary’ test software is only needed for the duration of the test.
Typical application: > End of line testing
> Debugging, diagnostics, testing

III) The BSL loads additional software to the system
The basic system includes a standard application software which has to be adapted to the different versions of a product. This is done with an additional application software loaded via the BSL.
Typical application: > End of line programming

The different kind of software applications can be mixed with different system hardware concerning the memory available on the system.
The application software can be loaded via the BSL to the:

1) Internal RAM
2) External RAM
3) Internal Flash EPROM
4) External Flash EPROM

Note that the BSL is not a Flash EPROM programming algorithm. The BSL is a program which can be used to load the Flash EPROM programming algorithm. Using the BSL to load a Flash EPROM programming algorithm is only one of several ways:

> Programming with a programming board
> In-System programming with system memory:
the programming algorithm is executed from system memory
> In-System programming with on-chip BSL:
the programming algorithm is downloaded into the target system via the BSL

3 General operation of the bootstrap loader

The BSL is activated at the end of a hardware reset, if pin ALE is sampled at high level, and if $\overline{\text{NMI}}$ is activated directly after the internal reset sequence has been terminated. The BSL is entered regardless of the state of pins EBC0, EBC1 and $\overline{\text{BUSACT}}$. In this bootstrap loading mode, the SAB 8xC166 now expects the serial reception of a zero byte (one startbit, 00_H data, one stop bit, no parity) from a host at pin RxD0 (P3.11), from which it calculates the necessary factor for the serial port baudrate generator, taking into account the operating frequency of the CPU.

According to the calculated baudrate, the serial port ASC0 is initialized (one start bit, 8 data bits, one stop bit, no parity), and an identification byte, 055_H, is sent back to the host. The members of the SAB8xC16x family send different identification bytes. After sending the identification byte, the BSL goes into a receive loop, expecting to receive exactly 32 bytes data from a host. If less than 32 bytes are received the SAB 8xC166 waits forever, no timeout is performed. The received bytes are stored sequentially into the internal RAM, beginning at address 0'$FA40_H$, and ending at address 0'$FA5F_H$. After the reception of the 32 bytes, the BSL automatically performs a jump to location 0'$FA40_H$, and the loaded program is executed. See figure 1.

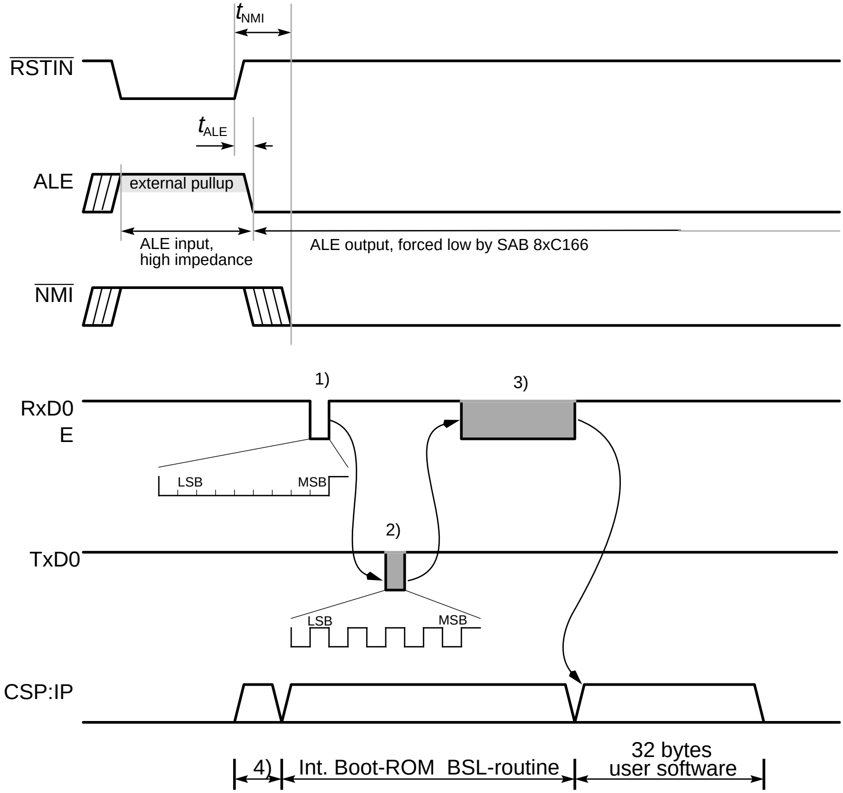
4 Hardware environment to use the bootstrap loader

During a normal reset (without activation of the internal Boot-ROM), ALE is switched to input and held low via an internal pulldown device. After termination of the internal reset sequence, the pulldown device is turned off, and ALE is switched to output mode, driving an active low level.

4.1 How to activate the bootstrap loader

The SAB 8xC166 which implements the BSL function is forced to enter the Boot-ROM and to execute the built-in bootstrap loader routine in two steps:

The **first** step is to switch the SAB 8xC166 to the internal Boot-ROM, therefore ALE must be high during $\overline{\text{RSTIN}}$ is low. This can be achieved by connecting an external device which has to be strong enough to pull the ALE pin to a logic high level. A typical value of a pullup resistor in normal systems is 2.2kΩ. Note also, that the state at the $\overline{\text{BUSACT}}$, EBC1 and EBC0 input pins have to be stable during and at the end of reset to avoid unexpected effects. $\overline{\text{NMI}}$ has also to be stable during reset.



1) Zero byte: Start bit, 00_H data byte and one stop bit from host

2) Identification byte from SAB 8xC166: 55_H

3) 32 bytes from host

4) Internal Boot-ROM

t_{ALE} : (0 - 0.25)µsec (20MHz CPU clock)

t_{NMI} : (0 - 1.2)msec (20MHz CPU clock)

$$t_{NMI(MAX)} \le \frac{25900}{f_{CPU}}$$

Figure 1: Bootstrap loader sequence

The **second** step is to start the BSL routine in the internal Boot-ROM. This can be achieved by activating the $\overline{\text{NMI}}$ pin directly after $\overline{\text{RSTIN}}$ is inactive. The timing window from $\overline{\text{RSTIN}}$ inactive to $\overline{\text{NMI}}$ active is 1.2msec, at 20MHz CPU clock. See figure 1, time t_{NMI}.

The following circuit is recommended for activation of the BSL in the two steps described above:

1) connect ALE with $\overline{\text{NMI}}$
2) connect an appropriate pullup device to the ALE/$\overline{\text{NMI}}$ line

In systems where the $\overline{\text{NMI}}$ pin is not used, a pullup device should be connected to $\overline{\text{NMI}}$ in any case to prevent erroneous NMI traps. Thus, only the (switchable) connection between ALE and $\overline{\text{NMI}}$ is required for entry into the BSL.

Special care has to be taken when designing the pullup device. External devices connected to ALE (e.g. an address latch) and $\overline{\text{NMI}}$ and their loads have to be taken into account (e.g. some TTL devices tend to pull a line connected to their inputs to high, acting as an additional pullup in parallel). The pullup must be strong enough to raise the voltage at the ALE pin during reset above the lower V_{IH} limit of $0.2V_{CC} + 0.9V$. On the other hand, it must be weak enough to allow the ALE output to drive a low level after reset (I_{OL} of 2.4mA at a V_{OL} of 0.4V).

4.2 Hardware example for bootstrap loader activation

Figure 2 shows two principle possibilities how to realize a circuit activating the BSL. The first circuit shows a simple way how to activate the BSL during a hardware reset. If it is desired to leave the BSL mode via a software reset, the connection between ALE and $\overline{\text{NMI}}$ has to be removed, before a software reset is executed and an access to the external memory is performed. Otherwise, if the interrupt system is already enabled, every ALE pulse would cause an $\overline{\text{NMI}}$. Before performing a hardware reset without activating the BSL, circuit_1 has to be removed, too.
In the second circuit the connection of ALE and $\overline{\text{NMI}}$ is switchable with the $\overline{\text{RSTOUT}}$ signal or with a hardware switch. The hardware switch allows to select between normal system start or start via BSL. In BSL mode the deactivation of $\overline{\text{RSTOUT}}$, which is done by the EINIT instruction, disconnects ALE and $\overline{\text{NMI}}$.

In order to return to normal operation, a hardware or software reset must be executed to terminate the BSL mode. The activation of the BSL is only performed with an external hardware reset ($\overline{\text{RSTIN}}$) and a BSL entry hardware circuit, while a software reset ignores the state of pin ALE. Care must be taken, however, that for a normal hardware reset the condition for entering the BSL is removed, otherwise the system starts in BSL mode. Immediately after a software reset was performed the connection between ALE and $\overline{\text{NMI}}$ has to be removed, too, because the code fetch, from the external memory address $0'0000_H$ causes an ALE pulse and thus activates $\overline{\text{NMI}}$.

Note also, that the configuration for the bus type is set appropriately for a hardware and software reset.

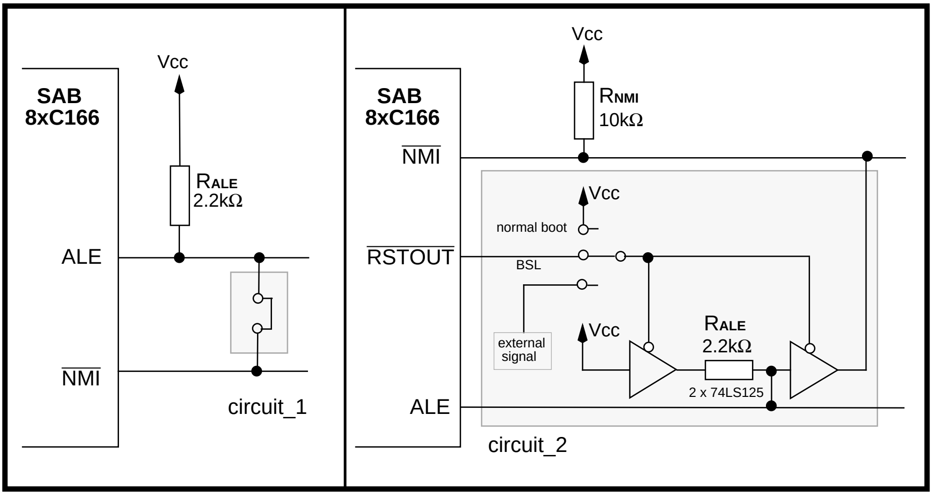


Figure 2: Two exemplary circuit diagrams for BSL entry via plug or switchable circuit

4.3 Special characteristics of the SAB 8xC166 in bootstrap loader mode

When the internal Boot-ROM is entered and no $\overline{\text{NMI}}$ occurs, an automatic timeout is performed after a maximum of 100msec (20MHz CPU clock). If the BSL routine in the internal Boot-ROM is started via an $\overline{\text{NMI}}$ there is no automatic timeout or other termination of the BSL mode, if no proper connection to a host is installed. The SAB 8xC166 will remain in BSL mode until a hardware reset is performed without the condition for entering the BSL.

Starting the SAB 8xC166 in BSL mode, the following system configuration is automatically programmed:

Watchdog Timer	:	disabled	S0CON Register	:	8011_H
Context Pointer	:	$FA00_H$	TxD/P3.10	:	1
Stack Pointer	:	$FA40_H$	DP3.10	:	1
STKUN Register	:	$FC00_H$	SYSCON Register	:	0000 0y00 xx00 0000b
STKOV Register	:	$FA00_H$			y (BUSACT), xx (BTYP)
S0BG Register	:	value calculated from received ‘00’ byte			according to the selected bus mode

Except for the Watchdog Timer, the system can be reprogrammed to the desired configuration after the bootstrap loading routine has been performed. The Watchdog Timer is only enabled after a hardware or software reset.

If the SAB 8xC166 is in BSL mode, all code fetches from the internal address space 0’0000$_H$ through 0’7FFF$_H$ (if mapped, 1’0000$_H$-1’7FFF$_H$) of the device are done from the internal Boot-ROM. Code fetches from this address area are not allowed to the user and would result in unexpected behavior. All data fetches are done from the internal user memory of the SAB 8xC166. See figure 3. Only after a software reset or a hardware reset (without BSL selection) was performed all code and data accesses to the internal memory are done from the internal user memory.

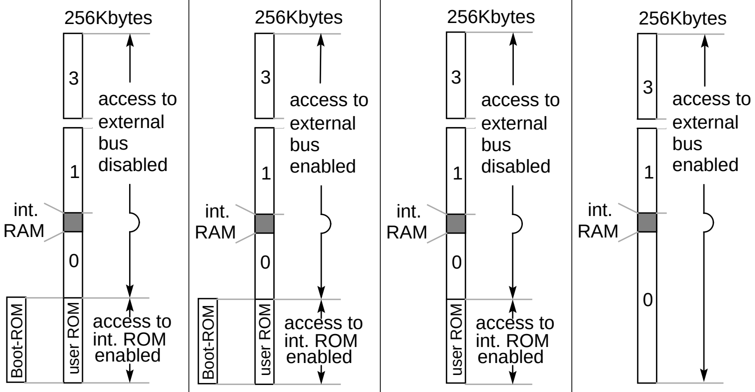
Note that due to the activation of the BSL via the $\overline{\text{NMI}}$, the system operates on the $\overline{\text{NMI}}$ trap level (trap priority II). Bit NMI in the Trap Flag Register (TFR.15) is set. All traps of class A and class B, and all interrupts or PECs are disabled, since they cannot interrupt the trap priority II.

Note that it is strongly recommended not to use half-duplex communication else unexpected results will occur.

4.4 Reset configuration and memory access

Figure 3 shows different possibilities how to configure the system memory via the external pins $\overline{\text{BUSACT}}$, EBC0 and EBC1 during reset, and the principle differences in memory access between a normal reset and a reset with BSL activation.

Configuration I) and II) show the reservation of the first 32 kbytes for internal accesses. The reservation is independent from the $\overline{\text{BUSACT}}$ pin. As already mentioned the accesses to internal address space (0’0000$_H$-0’7FFF$_H$) are distinguished between data or code. All code fetches are done from the internal Boot-ROM that is necessary to start and run the BSL routine after the hardware reset. After BSL is already started, code fetches from the internal Boot-ROM are not allowed and would cause undefined results. The data fetches are done from the internal user ROM. Only when the BSL mode is left by a software or hardware reset, the distinction between code, and data fetches is terminated. Configuration III) shows a normal system start where the internal ROM access is enabled, and in configuration IV) only the external memory is enabled.


256Kbytes
3
1
0
int. RAM
Boot-ROM
user ROM
access to external bus disabled
access to int. ROM enabled
256Kbytes
3
1
0
int. RAM
Boot-ROM
user ROM
access to external bus enabled
access to int. ROM enabled
256Kbytes
3
1
0
int. RAM
user ROM
access to external bus disabled
access to int. ROM enabled
256Kbytes
3
1
0
int. RAM
access to external bus enabled

BSL mode active	yes	yes	no	no
EBC0 pin	low	x)	low	x)
EBC1 pin	low	x)	low	x)
$\overline{\text{BUSACT}}$ pin	high	low	high	low
Code fetch int. ROM	Boot-ROM access[2)]	Boot-ROM access[2)]	user ROM access[1)]	----------
Data fetch int. ROM	user ROM access[1)]	user ROM access[1)]	user ROM access[1)]	----------
Mem. config. No.:	I)	II)	III)	IV)

x) depends on the external bus configuration

1) All accesses to the internal address space (0'0000$_H$ - 0'7FFF$_H$) of a ROMless version of the SAB 8xC166 result in undefined read values

2)Not allowed, unexpected result will occur

Figure 3: Configuration of system memory after reset

4.5 Which baudrates does the bootstrap loader accept

The BSL calculates the necessary reload value for the baudrate generator from the duration of the received zero byte which is sent from the host. Due to the resolution of Timer 6 and the resolution of the baudrate generator there are upper and lower limits concerning the baudrate for correct transfer.

4.5.1 Lower limit of the baudrate

The lower limit of the baudrate is specified by the longest zero byte which is measurable with the 16-bit Timer 6. This period depends on the CPU clock (f_{CPU}) of the SAB 8xC166.

$$\boxed{B_{Low} > \frac{9}{4} \cdot \frac{f_{CPU}}{2^{16}}}$$

For a system with a 20MHz CPU clock, the lowest allowed baudrate of the host is 687 Baud. See example. Therefore the lowest standard baudrate which can be used for correct serial communication with the BSL is 1200 Baud.

4.5.2 Upper limit of the baudrate

For a system with a 20MHz CPU clock a maximum standard baudrate for the host of 19200 Baud is recommended. Then the maximum deviation between the baudrate of the host and the ASC0 is lower than 1.6%. A detailed explanation how to calculate the baudrate deviations between host and ASC0 is shown below.

The upper limit of the baudrate is specified by the maximum deviation of the baudrates between host and SAB 8xC166 which is allowed for a correct serial data transfer.

Via the contents of the baudrate generator reload value (S0BRL), the baudrate generator of the SAB 8xC166 allows baudrates with the following values:

$$\boxed{B_{166} = \frac{f_{CPU}}{32 \cdot (S0BRL + 1)}}$$

That is why not all baudrates are adjustable with the baudrate generator.

From the duration of the zero byte which is sent from the host, the BSL calculates the corresponding reload value for the baudrate generator with respect to the current CPU clock (f_{CPU}). The calculation of the reload value is based on the contents of Timer 6 (T6). The contents of Timer 6 is calculated via the CPU clock and the baudrate of the host:

$$\boxed{S0BRL = \frac{T6}{72} - 1 \qquad , \qquad T6 = \frac{9}{4} \cdot \frac{f_{CPU}}{B_{Host}}}$$

The division T6/72 is an integer calculation and includes already a reduction of the error caused by the baudrate generator resolution.

The maximum deviation between the internal initialized baudrate for ASC0 and the baudrate of the host during BSL is active (8 data bits, one stopbit, no parity) has to be lower or equal than 2.5%, else a correct data transfer is not guaranteed. The deviation (F_B, in percent) between the baudrate of the host and the baudrate calculated and initialized by the BSL is:

$$F_B = \left| \frac{B_{166} - B_{Host}}{B_{166}} \right| \cdot 100 \ \% \quad , \qquad F_B \leq 2.5 \ \%$$

This baudrate deviation is a not linear function depending on CPU clock and the baudrate of the host. The maximums of the function (F_B) decrease during reduction of baudrate of the host. The values of the maximums depend on the CPU clock. The baudrates were the maximums and minimums are located depend on the baudrate values which are generated from the baudrate generator. The principle is shown in figure 4.

Note that the function (F_B) does not consider the tolerances of oscillators and other devices supporting the serial communication.

4.5.3 Example

The example is calculated for an SAB 8xC166 at 20MHz CPU clock.

Lower limit of the baudrate:

$$B_{Low} > \frac{9}{4} \cdot \frac{f_{CPU}}{2^{16}} \quad , \qquad B_{Low} > \frac{9}{4} \cdot \frac{(20 \cdot 10^6)}{2^{16}} \quad , \qquad \underline{\underline{B_{Low} > 687 \text{Baud}}}$$

Upper limit of the baudrate:

The real maximum baudrate (B_{Max}) which is supported by the baudrate generator of the SAB 8xC166 is 625kBaud. This maximum baudrate assumes that the baudrate of the host is in the range from 610KBaud up to 640KBaud, else the deviation between the baudrate of the host and the SAB 8xC166 leaves the allowed tolerance of ±2.5%. See figure 4.

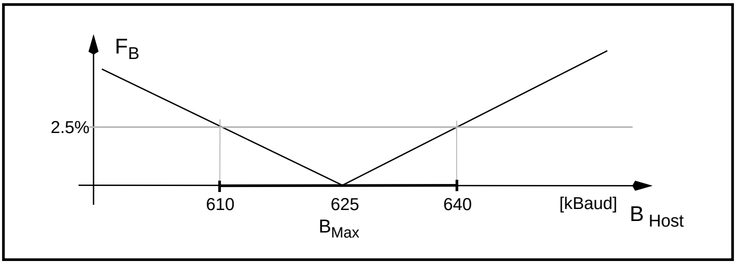


Figure 4: Real maximum baudrate for the BSL

The baudrate (B_{High}) recommended for the upper limit is specified by the maximums of the deviation function (F_B, figure 5).

If the upper limit (B_{High}) of the recommended baudrate is not exceeded the whole range of the baudrate from the lower limit to the recommended upper limit can be used for data transfer. If B_{High} is exceeded then special care has to be taken to the maximums of the baudrate deviation function F_B.

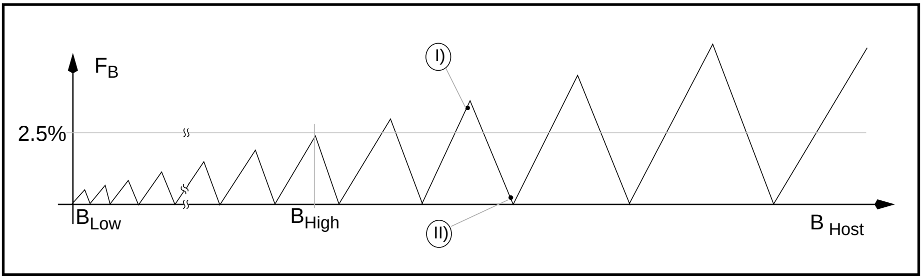


Figure 5: Principle baudrate deviation between host and SAB 8xC166 when the BSL is active

The marked points I) and II) of $F_{B\_I)}$ in figure 5 are now explained with a calculated example. The host baudrates are:

$$\text{I) : } B_{Host\_I)} = 40300\text{Baud}$$

$$\text{II): } B_{Host\_II)} = 41500\text{Baud}$$

The corresponding deviation F_B between host and SAB 8xC166 is calculated in the following steps:

$$T6 = \frac{9}{4} \cdot \frac{f_{CPU}}{B_{Host-I)}} \quad , \quad T6 = \frac{9}{4} \cdot \frac{20 \cdot 10^6}{40300} \qquad T6 = 1116$$

$$S0BRL = \frac{T6}{72} - 1 \quad , \quad S0BRL = \frac{1116}{72} - 1 \quad , \quad S0BRL = 14$$

$$B_{166} = \frac{f_{CPU}}{32 \cdot (S0BRL + 1)} \quad , \quad B_{166} = \frac{20 \cdot 10^6}{32 \cdot (14 + 1)} \quad , \quad B_{166} = 41666 \mathrm{Baud}$$

$$F_{B-I)} = \left| \frac{B_{166} - B_{Host-I)}}{B_{166}} \right| \cdot 100 \quad \% \quad , \quad F_{B-I)} = \left| \frac{41666 - 40300}{41666} \right| \cdot 100 \quad \%$$

$$\boxed{F_{B-I)} \approx 3.2 \ \%}$$

The result for the second baudrate value ($B_{Host\_II)}$ = 41500Baud) is:

$$\boxed{F_{B-II)} \approx 0.4 \ \%}$$

The first result $F_{B\_I)}$ is not inside of the permitted tolerance of 2.5% but the second one $F_{B\_II)}$ is allowed. The example shows the problem of F_B if the range of B_{Low} up to B_{High} is exceeded, then the baudrate of the host has to be carefully selected, else there is no correct data transfer guaranteed.

5 User software loaded with the bootstrap loader

Normally a program requires more than 32 bytes. Thus, to load larger routines, the 32 bytes program loaded via the BSL will in most cases be a preloader, now with user-defined start and end addresses. Since the serial port ASC0 is already initialized to the correct mode and baudrate, no special actions are necessary. This preloader loaded via the BSL may be used to load extensive user software to the internal RAM or external memory.

5.1 How to enable the external memory

After starting the SAB 8xC166 in bootstrap loading mode, a 32 kbyte address range is reserved for internal accesses. The default address range for this 32 kbyte block is 0’0000$_H$ through 0’7FFF$_H$. However, in order to access external memory in this range, disabling or mapping of this 32 kbyte block to segment 1 is possible. This disabling or mapping enables the access to the external memory and must be performed via the following instruction sequence:

```
      MOV    SYSCON,#xxxx x0xx YYxx xxxxb   ; BUSACT-bit = 0,
                                            ; YY: (BTYP=01): maps 32 kbyte block to seg. 1
                                            ; YY: (BTYP=10): disables 32 kbyte block
      JMPS   next                           ; dummy jump segment to next instruction
next: FAR
      MOV    DPP0, #m                       ; update data page pointers
      MOV    DPP1, #u
      MOV    DPP2, #v
      MOV    DPP3, #w
      ........
      ........
      EINIT                                 ; end of initialization
```

Notes:

1.) Special care must be taken regarding the address space for this instruction sequence. This sequence must not be executed within the lower 32 kbyte address space in both, segment 0 and segment 1. The address space for this instruction sequence has to be either in the internal RAM or in the external memory outside the scope of the 32 kbyte block before and after the mapping, otherwise unexpected result will occur.

2.) This instruction sequence has to be performed before the EINIT instruction! After the execution of EINIT, any operation to enable, disable or map the 32 kbyte block is cancelled.

3.) The ‘MOV SYSCON’ instruction should be used in any case, since the BUSACT bit have to be set to 0 with the same instruction that changes the BTYP bits. It is not possible to perform this operation with single bit or bitfield instructions!

4.) The selection of the external bus, reflected via the BTYP bits, is not changed via the MOV SYSCON instruction, if BUSACT is set to 0 with this instruction. Changes of the bus type before the EINIT instruction are only performed, if BUSACT is set to 1 with the same instruction.

5.) The initialization sequence ‘jump segment’ and reload all DPPs must be done in one block. After the whole initialization sequence is done the ROM disabling or mapping is active.

6.) An explicit JMPS mnemonic has to be used. A generic JMP mnemonic (without ‘FAR’) will be translated by the assembler to a normal JMPA or JMPR instruction.

7.) Do not change the sequence of the instructions. The SYSCON register has to be initialized first then the dummy jump segment and update of the Data Page Pointers must follow.

8.) After mapping, the 32 kbyte address range (reserved for internal accesses) to memory location 1’0000$_H$ through 1’7FFF$_H$ or after disabling the 32 kbyte block, in both cases, the external memory can now be accessed in the address range 00000h through 0’7FFF$_H$. But don’t forget to enable the $\overline{\text{WR}}$ pin!

5.2 Program examples how to program the BSL

The following programs show how to load and run the user software to the internal or external memory with the BSL. This is done in three examples. Example 1 checks the communication between the SAB 8xC166 and the host. Example 2 loads a user program to the internal memory and example 3 shows how to load a user program to the external memory.

Note that all examples are programmed with a minimum stack size. If extended application software is used care has to be taken to the stack size.

5.2.1 Example 1: The first test program loaded via the BSL to the internal memory

Because of a gradual development of the software it is recommended to start using the BSL not with a preloader routine but with a test routine which sends back only an acknowledge byte via the Serial Channel 0 (ASC0). Thus the basic function of the communication between SAB 8xC166 and host via ASC0 will be tested. See example program USRSW1.ASM. The program ‘user_software_1’ is loaded via BSL to start address 0’FA40$_H$ and end address 0’FA5F$_H$. This routine sends back an acknowledge byte ‘03’ to the host via Serial Channel 0. The corresponding memory map is shown in figure 6.

Note that the SAB 8xC166 is still in BSL mode after the test routine has been executed.

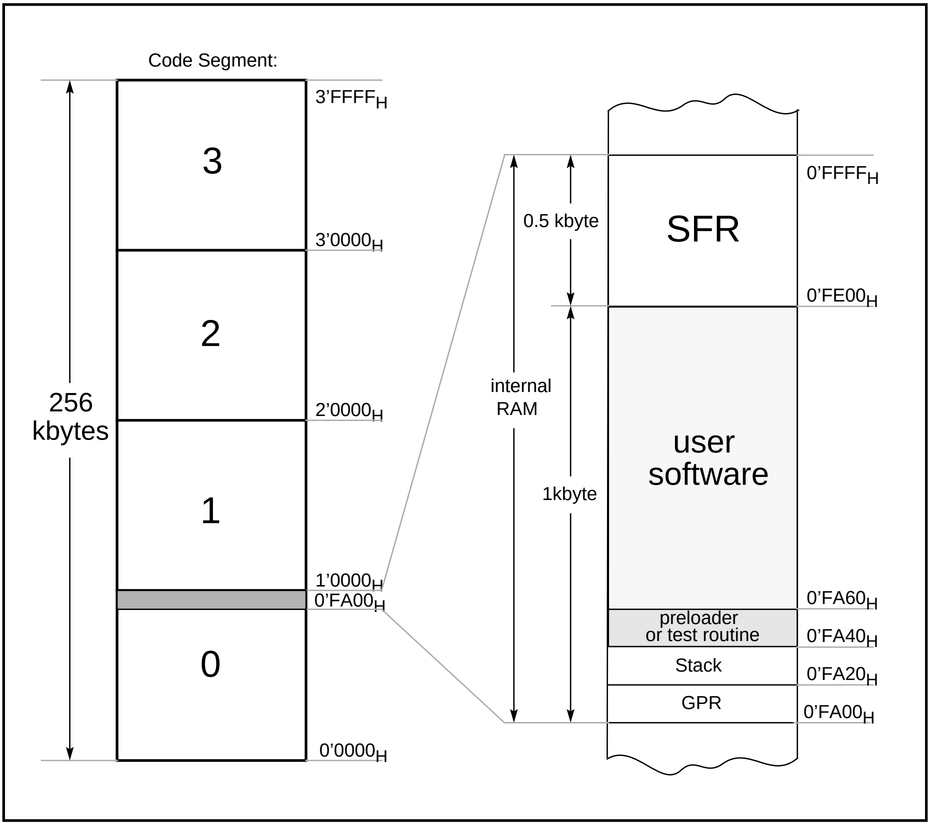


Figure 6: Memory map for user software loaded to the internal RAM

5.2.2 Example 2: Load user software to internal memory

An example how to do this is shown in the two programs PRELOAD1.ASM and USRSW2.ASM. See appendix. The preloader is loaded via the BSL and located from memory address 0'FA40$_H$ to end address 0'FA5F$_H$. The preloader PRELOAD1. ASM loads the user software to the internal RAM (start address 0'FA60$_H$) of the SAB 8xC166. When using only the internal RAM, the largest possible user program loaded by the preloader is 928 bytes. Start address is 0'FA60$_H$ and end address is 0'FDFF$_H$. See figure 6. The start of the user software is performed with a jump from the end of the preloader to the user program start address 0'FA60$_H$. The program USRSW2.ASM sends back an acknowledge byte '05' to the host via Serial Channel 0. The corresponding memory map is also shown in figure 6.

Note that the SAB 8xC166 is still in the BSL mode after the user software has been executed.

5.2.3 Example 3: Load user software to external memory

This is done in three steps. First a preloader is loaded via the BSL to the internal RAM. Then the preloader loads an external loader to the internal RAM of the SAB 8xC166 and finally the external loader stores the user software to the external memory. Figure 7 shows the action items in detail, how to load the user software to the external memory and figure 8 shows the according system memory map. Example 3 assumes that the BSL entry circuit_2 is used, otherwise little changes in the software are necessary.

Now an explanation of example 3 in detail:

After performing a hardware reset including the external circuit_2 for BSL entry, the BSL loads the preloader (PRELOAD1.ASM, same program as in example 2) via ASC0 to the internal RAM of the SAB 8xC166 from memory address 0’FA40$_H$ to end address 0’FA5F$_H$.

After the BSL has started the preloader, that loads the external loader (EXTLOAD1.ASM) via ASC0 to the internal RAM storing from memory address 0’FA60$_H$.

The external loader is started via the preloader and performs different actions:

1) The external bus is enabled allowing access to the external memory. For this the internal ROM access is disabled, a dummy jump segment is performed and the Data Page Pointers are reloaded. Further the $\overline{\text{WR}}$ pin is enabled.

2) The user software is loaded via ASC0 and stored from external memory address 0’1000$_H$, this location is user defined.

3) A jump instruction to the user program start address 0’1000$_H$ is stored to the reset vector location 0’0000$_H$.

4) A jump instruction to the $\overline{\text{NMI}}$ trap routine start address 0’0200$_H$ is stored to the $\overline{\text{NMI}}$ trap vector location 0’0008$_H$.

5) The $\overline{\text{NMI}}$ trap routine is stored from start address 0’0200$_H$, this location is user defined.

6) The last action of the external loader is to execute a software reset.

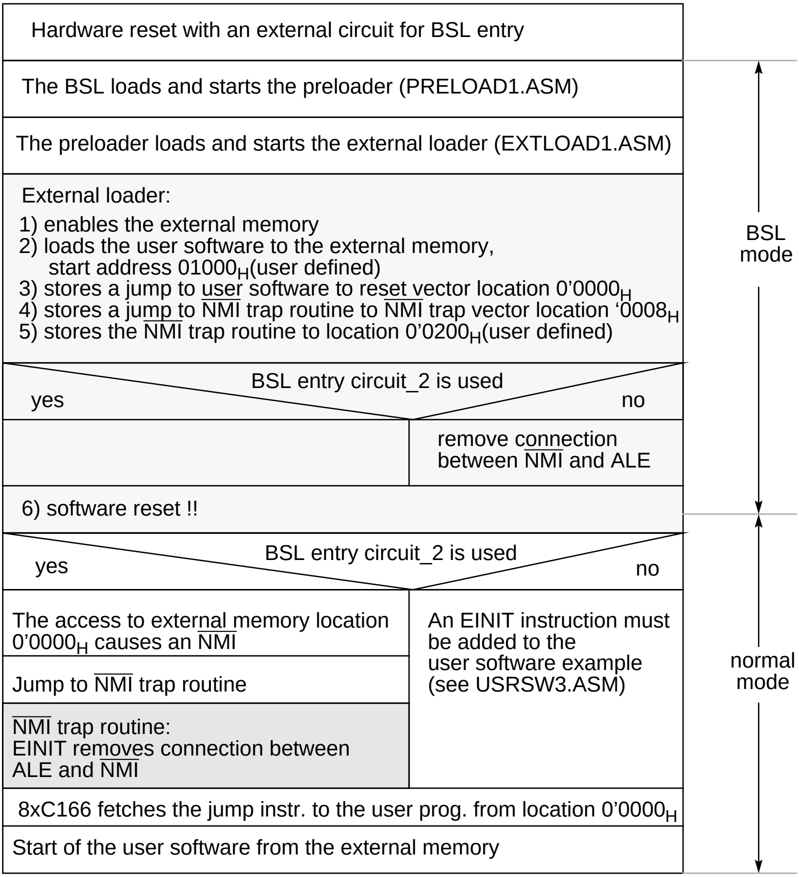

Hardware reset with an external circuit for BSL entry
The BSL loads and starts the preloader (PRELOAD1.ASM)
The preloader loads and starts the external loader (EXTLOAD1.ASM)
External loader:
1) enables the external memory
2) loads the user software to the external memory, start address 01000$_H$(user defined)
3) stores a jump to user software to reset vector location 0’0000$_H$
4) stores a jump to $\overline{\text{NMI}}$ trap routine to $\overline{\text{NMI}}$ trap vector location ‘0008$_H$
5) stores the $\overline{\text{NMI}}$ trap routine to location 0’0200$_H$(user defined)
BSL entry circuit_2 is used
yes
no
remove connection between $\overline{\text{NMI}}$ and ALE
6) software reset !!
BSL entry circuit_2 is used
yes
no
The access to external memory location 0’0000$_H$ causes an $\overline{\text{NMI}}$
Jump to $\overline{\text{NMI}}$ trap routine
$\overline{\text{NMI}}$ trap routine: EINIT removes connection between ALE and $\overline{\text{NMI}}$
An EINIT instruction must be added to the user software example (see USRSW3.ASM)
8xC166 fetches the jump instr. to the user prog. from location 0’0000$_H$
Start of the user software from the external memory
BSL mode
normal mode

Figure 7: Action items how to load user software to the external memory and leave the BSL mode

The software reset, performed by the external loader, causes a code fetch from the reset vector location 0’0000$_H$. Because of the BSL entry circuit_2 ($\overline{\text{NMI}}$ and ALE are connected) the ALE pulse of the first read access to the external address 0’0000$_H$ causes an $\overline{\text{NMI}}$ and therefore a jump to the $\overline{\text{NMI}}$ trap routine. The first instruction in the $\overline{\text{NMI}}$ trap routine is an EINIT. The EINIT instruction removes the connection between $\overline{\text{NMI}}$ and ALE via the BSL entry circuit_2. Further the Trap Flag Register is cleared and a return from interrupt (RETI) is performed.

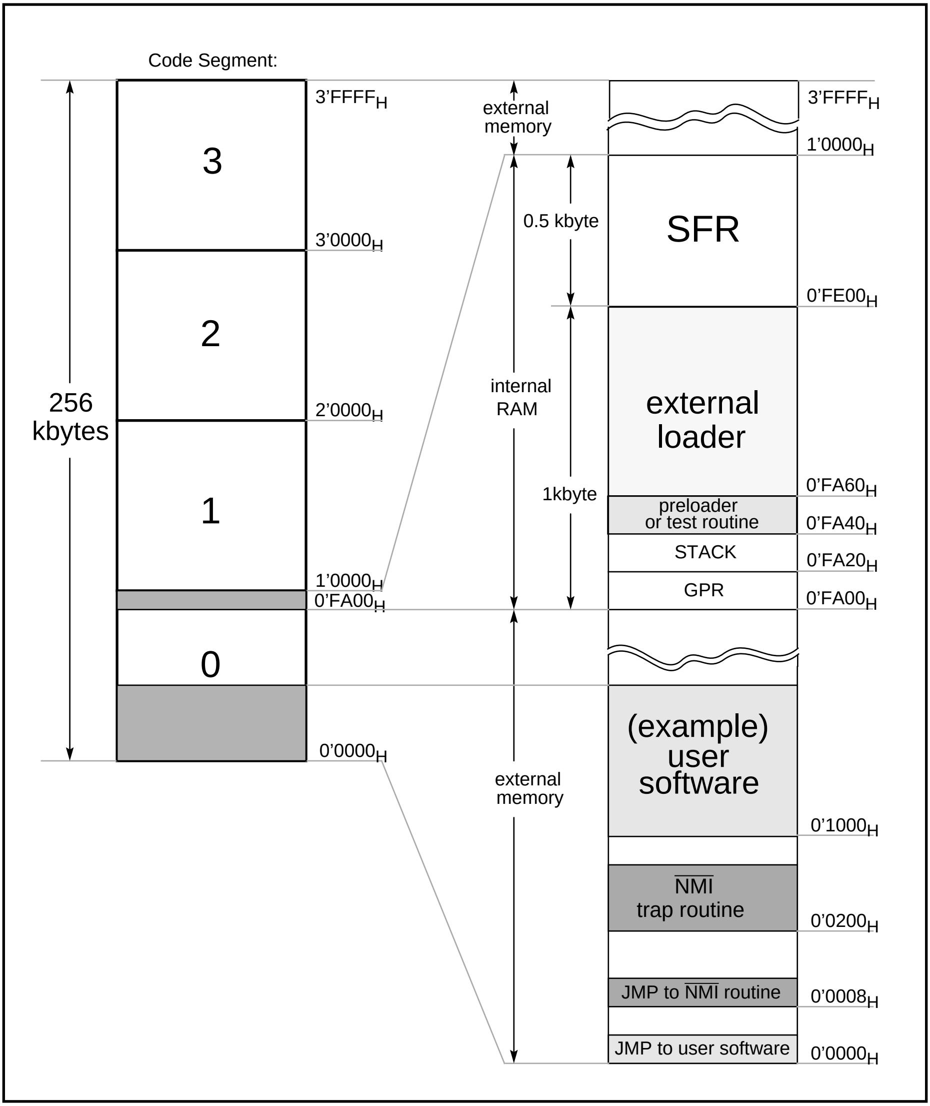


Figure 8: Memory map for user software loaded to the external memory

After the RETI instruction is executed the system performs a code fetch from address 0’0000$_H$ (now without $\overline{\text{NMI}}$) and then a jump to the start address 0’1000$_H$ of the user program.

The user program (USRSW3.ASM) sends back acknowledge bytes ‘07’ in an endless loop via ASC0. Because of the software reset the SAB 8xC166 has to be initialized. This is done in the program USRSW3.ASM, it disables the Watchdog Timer, sets the Context Pointer, the Stack Pointer and initializes ASC0.

Note that the SAB 8xC166 leaves the BSL mode after the software reset. The user software is performed in normal mode.

6 Appendix

6.1 USER_SOFTWARE_1

```
$LISTALL

$DEBUG

;+--------------------------------------------------------------------+
;| Program Name  : USRSW1.ASM                        Rev. : 1.0       |
;| Description   : is loaded via the BSL and sent back an             |
;|                 acknowledge byte <03>.                             |
;|                 Program start address: 0FA40h                      |
;+--------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0      |
;| Author        : Mariutti                          Date :  3.11.93  |
;+--------------------------------------------------------------------+
;| Revision History:                                                  |
;|                                                                    |
;+--------------------------------------------------------------------+
NAME   USER_SOFTWARE_1

       SSKDEF 03               ; stack size 32 words
       REGDEF R0

SEC1   SECTION CODE AT 0FA40h  ; program start address 0FA40h
MAIN   PROC  TASK  INTNO=0
;+---------------------------------------------------------------------+
;|     initialization  | BSL RESET values: Stack Pointer   = 0FA40h   |
;|                     |                   Context Pointer = 0FA00h   |
;+---------------------------------------------------------------------+
       MOV   STKUN ,#0FA40h    ; set Stack Underflow Pointer Register
       MOV   STKOV ,#0FA20h    ; set Stack Overflow  Pointer Register
       MOV   SYSCON,#062C0h    ; stack: 32 words, no external bus
;+---------------------------------------------------------------------+
;|      send back an acknowledge byte <03> via ASC0                    |
;+---------------------------------------------------------------------+
       MOV   S0TBUF, #0003h    ; write acknowledge byte <03> to
                               ; transmitbuffer of ASC0
Loop:  JMPR  LOOP              ; wait forever and ever.. .  .   .    .
       RETV
MAIN   ENDP
SEC1   ENDS
       END
```

6. 2PRELOADER_1

```
$LISTALL
$DEBUG
;+--------------------------------------------------------------------+
;| Program Name  : PRELOAD1.ASM                      Rev. : 1.0       |
;| Description   : is loaded via the BSL. PRELOAD1 loads a prog. to  |
;|                 the 8xC16x with a length of 928 bytes via ASC0.    |
;|                 Program start address: 0FA40h                      |
;+--------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0      |
;| Author        : Mariutti                          Date :  3.11.93 |
;+--------------------------------------------------------------------+
;| Revision History:                                                  |
;+--------------------------------------------------------------------+
NAME   PRELOADER1
       SSKDEF 03               ; stack size 32 words
       REGDEF R0
SEC1   SECTION CODE AT 0FA40h  ; program start address 0FA40h
MAIN   PROC  TASK  INTNO=0
;+----------------------------------------------------------------------+
;|     initialization  | BSL RESET values:  Stack Pointer   = 0FA40h    |
;|                     |                   Context Pointer  = 0FA00h    |
;+----------------------------------------------------------------------+
       MOV   STKUN ,#0FA40h    ; set Stack Underflow Pointer Register
       MOV   STKOV ,#0FA20h    ; set Stack Overflow  Pointer Register
;+----------------------------------------------------------------------+
;|     main program                                                     |
;+----------------------------------------------------------------------+
       MOV   R0,#0FA60h        ; mov start address to R0
LABEL1:JNB   S0RIC.7,LABEL1    ; wait until S0RBUF includes data
       MOVB  [R0],S0RBUF       ; mov data of S0BUF to start address
       BCLR  S0RIC.7           ; clear bit S0RIC.7 (S0RIR)
       CMPI1 R0,#0FDFFh        ; compare R0 with end address FDFFh
       JMPR  CC_NZ,LABEL1      ; jump to LABEL1 if R0 does not include
                               ; end address
       JMPA  USRSW             ; jump to user software (ext. loader)
                               ; start address
       RETV
MAIN   ENDP
SEC1   ENDS

SEC2   SECTION CODE AT 0FA60h  ; user software (external loader)
                               ; start address 0FA60h
USRSW:
SEC2   ENDS
       END
```

6.3 USER_SOFTWARE_2

```
$LISTALL
$DEBUG
;+-------------------------------------------------------------------+
;| Program Name  : USRSW2.ASM                        Rev. : 1.0     |
;| Description   : is loaded via PRELOAD1 and sent back an          |
;|                 acknowledge byte <05>.                           |
;|                 Program start address: 0FA60h                    |
;+-------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0     |
;| Author        : Mariutti                          Date :  3.11.93 |
;+-------------------------------------------------------------------+
;| Revision History:                                                 |
;|                                                                   |
;+-------------------------------------------------------------------+

NAME   USER_SOFTWARE_2

       SSKDEF 03 ; stack size 32 words
       REGDEF R0

SEC1   SECTION CODE AT 0FA60h  ; program start address 0FA60h
MAIN   PROC TASK  INTNO=0
;+---------------------------------------------------------------------+
;|     initialization    | BSL RESET values:  Stack Pointer   = 0FA40h |
;|                       |                   Context Pointer  = 0FA00h |
;+---------------------------------------------------------------------+
       MOV   STKUN ,#0FA40h    ; set Stack Underflow Pointer Register
       MOV   STKOV ,#0FA20h    ; set Stack Overflow  Pointer Register
       MOV   SYSCON,#062C0h    ; stack: 32 words, no external bus
;+---------------------------------------------------------------------+
;|      send back an acknowledge byte <05> via ASC0                           |
;+---------------------------------------------------------------------+
       MOV   S0TBUF, #0005h    ; write acknowledge byte <05> to
                               ; transmitbuffer of ASC0
LOOP:  JMPR  LOOP              ; wait forever and ever.. .  .   .
       RETV
MAIN   ENDP
SEC1   ENDS
       END
```

6.4 EXTERNAL_LOADER_1

```
$LISTALL
;+-------------------------------------------------------------------+
;| Program Name  : EXTLOAD1.ASM                      Rev. : 1.0      |
;| Description   : stores a user program to the external memory      |
;|                 installs a jump to the user program               |
;|                 and the NMI# trap routine. Performes a sw RESET   |
;|                 Program start address: 0FA60h                     |
;+-------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0     |
;| Author        : Mariutti                          Date :  3.11.93 |
;+-------------------------------------------------------------------+
;| Revision History:                                                 |
;|                                                                   |
;+-------------------------------------------------------------------+
NAME   EXTERNAL_LOADER_1
       SSKDEF 03               ; stack size 32 words
       REGDEF R0-R5
sec1   SECTION CODE AT 0FA60h
main   PROC  TASK  INTNO=0
;+---------------------------------------------------------------------+
;| disable int. ROM, dummy jump segment and reload all Data Page Pointer  |
;+---------------------------------------------------------------------+
       MOV   SYSCON,#06280h    ; disable internal ROM
                               ; stack size: 32 words
       JMPS  SEG sec1, next_line ;dummy jump seg. to next instructon
next_line:   far
       MOV   DPP0,#0h          ; update Data Page Pointer
       MOV   DPP1,#1h
       MOV   DPP2,#2h
       MOV   DPP3,#3h
;+---------------------------------------------------------------------+
;| enable WR# pin, enable and init. ext. bus to system configuration      |
;+---------------------------------------------------------------------+
       BSET  P3.13             ; set WR pin to HIGH level
       BSET  DP3.13            ; define WR pin to output
       MOV   SYSCON,#06640h    ; enable external bus, 8bit mux
;+---------------------------------------------------------------------+
;| load user software via ASC0 and store it to ext. mem. startadr: 1000h |
;+---------------------------------------------------------------------+
       MOV   R5,#01000h        ; mov addr 1000 to R5
label1: JNB  S0RIC.7,label1    ; wait until S0RBUF includes data
       MOVB  [R5],S0RBUF       ; store data of S0BUF
       BCLR  S0RIC.7           ; clear bit S0RIC.7 (S0RIR)
       CMPI1 R5,#0139fh        ; compare R5 with end address
       JMPR  cc_NZ,label1      ; jump to label1 if R5 does not include
                               ; end address
;+---------------------------------------------------------------------+
;| store a jump to user program start addr. to reset vector loc. 0000h |
;+---------------------------------------------------------------------+
       MOV   R4,#0000h         ; move (jmpa  #01000h) to addr 0000h
       MOV   R5,#00EAh
       MOV   [R4],R5
       MOV   R4,#0002h
       MOV   R5,#01000h
       MOV   [R4],R5
;+---------------------------------------------------------------------+
;| store a jump to NMI# trap routine to NMI# trap vector location 0008h   |
;+---------------------------------------------------------------------+
       MOV   R4,#0008h         ; move (jmpa  #00200h) to addr 0008h
       MOV   R5,#00EAh
       MOV   [R4],R5
       MOV   R4,#000Ah
       MOV   R5,#00200h
       MOV   [R4],R5
;+---------------------------------------------------------------------+
;|  store NMI# trap routine from start address 0200h                   |
;+---------------------------------------------------------------------+
       MOV   R4,#00200h        ; EINIT: the external BSL circuit_2
       MOV   R5,#04AB5h        ; removes the connection between
       MOV   [R4],R5           ; ALE and NMI#
       MOV   R4,#00202h
       MOV   R5,#0B5B5h
       MOV   [R4],R5

       MOV   R4,#00204h        ; MOV TFR, ZEROS
       MOV   R5,#0D6F2h
       MOV   [R4],R5
       MOV   R4,#00206h        ;
       MOV   R5,#0FF1Ch
       MOV   [R4],R5

       MOV   R4,#00208h        ; RETI
       MOV   R5,#088FBh
       MOV   [R4],R5
;+--------------------------------------------------------------------+
;| leave the BSL mode via a software reset                            |
;+--------------------------------------------------------------------+
       SRST                    ; software reset
sec1   ENDS
       END
```

6.5 USER_SOFTWARE_3

```
$LISTALL
$DEBUG
;+-------------------------------------------------------------------+
;| Program Name  : USRSW3.ASM                       Rev. : 1.0      |
;| Description   : initializes the system after a software reset     |
;|                 and sends back an acknowledge byte <07> via ASC0. |
;|                 Program start address: 01000h                     |
;+-------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0     |
;| Author        : Mariutti                          Date :  3.11.93 |
;+-------------------------------------------------------------------+
;| Revision History:                                                 |
;+-------------------------------------------------------------------+
NAME     USER_SOFTWARE_3
         SSKDEF 03              ; stack size 32 words
         REGDEF R0
SEC3     SECTION CODE AT 01000h
MAIN     PROC  TASK  INTNO=0
;+--------------------------------------------------------------------+
;|       initialization after software reset                          |
;+--------------------------------------------------------------------+
         DISWDT                 ; disable watchdog timer
         MOV   CP     ,#0FA00h  ; set registerbank
         MOV   SP     ,#0FA40h  ; set stackpointer
         MOV   STKUN  ,#0FA40h  ; set Stack Underflow Pointer Register
         MOV   STKOV  ,#0FA20h  ; set Stack Overflow  Pointer Register
         MOV   S0BG   ,#003Fh   ; Baud rate 9600 Baud
         BSET  P3.10            ; initialize TXD0 output
         BSET  DP3.10           ;
         MOV   S0CON  ,#8011h   ; initialize serial port 0:
         ; 8-bit data, no parity, one stopbit, receiver enabled
         BSET  P3.13            ; set WR pin to HIGH level
         BSET  DP3.13           ; define WR pin to output
         MOV   SYSCON ,#06640h  ; stack: 32 words,
                                ; enable external bus, 8bit mux
;+--------------------------------------------------------------------+
;|       send back acknowledge bytes <07> via ASC0                    |
;+--------------------------------------------------------------------+
LOOP:    MOV   S0TBUF ,#0007h
TRANSMIT:JNB   S0TIC.7, TRANSMIT
         BCLR  S0TIC.7
         JMPR  LOOP
         RETV
MAIN     ENDP
SEC3     ENDS
         END
```