SIEMENS

Bootstrap Loader

C165 / C167

Microcomputer Components

C165/C167
SAB 80C166 Family ApNotes

Application Note of the
On-chip Bootstrap Loader

Advance Information 1.94

SIEMENS

Edition 1.94 (Advance Information)
This edition was realized using the software system FrameMaker®.

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

Contents

1 Introduction

In the C165/C167, an on-chip bootstrap loader (BSL) is implemented and introduced with the C165: AA-step and C167: AD-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 C165/C167 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 C165/C167.

2 Purpose of a bootstrap loader

As already mentioned, it is possible to load a program to the internal or external memory of the C165/C167 via the BSL. Fundamentally there are 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 already includes 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 P0L.4 is sampled at low level. The BSL is entered regardless of the state of the $\overline{\text{EA}}$ pin, the bus type, chip select, and segmet address configuration pins. In this bootstrap loading mode, the C165/C167 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 is sent back to the host. The members of the SAB 8xC166 family send different identification bytes (C165: <$B5_H$>, 8xC166: <55_H>, C167: <$A5_H$>). After sending the identification byte, the BSL goes into a receive loop, expecting to receive exactly 32 data bytes from a host. If less than 32 bytes are received the C165/C167 waits forever, no timeout condition is detected. The received bytes are stored sequentially into the internal RAM, beginning at address $00'FA40_H$, and ending at address $00'FA5F_H$. After the reception of the 32 bytes, the BSL automatically performs a jump to location $00'FA40_H$, 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), the pins of PORT0 are switched to input and held high via internal pullup devices. After termination of the internal reset sequence, the pullup devices are turned off.

4.1 How to activate the bootstrap loader

The C165/C167 offers two different ways to start the bootstrap loader routine. The BSL entry via P0L.4 is recommended for all applications. The BSL entry via P0L.3 and $\overline{\text{NMI}}$ is only implemented for compatibility with the 8xC166.

4.1.1 BSL entry via P0L.4

The C165/C167 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 C165/C167 to the internal Boot-ROM and the second step is to start the BSL routine from the internal Boot-ROM. This can be achieved by connecting an external device which has to be strong enough to force the P0L.4 pin to a logic low level, and executing a hardware reset. A typical value of a pulldown resistor in normal systems is 8KΩ.

Note that the port pins P0L.2, P0L.3 and P0L.5 must be at high level, and stable during and at the end of reset otherwise the BSL routine is not started. The state at the $\overline{\text{EA}}$ input pin has to be stable during and at the end of reset, too, to avoid unexpected effects.

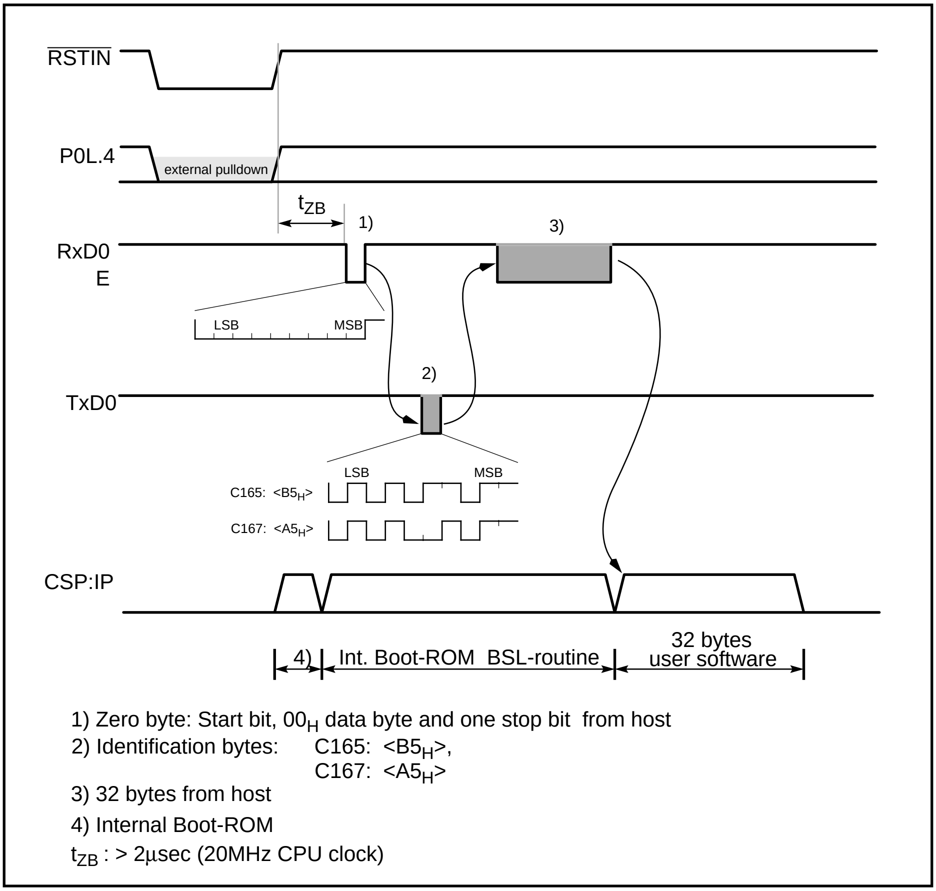

$\overline{\text{RSTIN}}$
P0L.4
external pulldown
t_{ZB}
1)
3)
RxD0
E
LSB
MSB
2)
TxD0
LSB
MSB
C165: <B5$_H$>
C167: <A5$_H$>
CSP:IP
4)
Int. Boot-ROM BSL-routine
32 bytes user software
1) Zero byte: Start bit, 00$_H$ data byte and one stop bit from host
2) Identification bytes: C165: <B5$_H$>, C167: <A5$_H$>
3) 32 bytes from host
4) Internal Boot-ROM
t_{ZB} : > 2µsec (20MHz CPU clock)

Figure 1: Bootstrap loader sequence

Special care has to be taken when designing the pulldown device. External devices connected to P0L.4 (e.g. an address latch) 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 a pullup device). The pulldown must be strong enough to force the voltage at the P0L.4 pin during reset below the upper V_{IH} limit of $0.2V_{CC} + 0.9V$. On the other hand, it must be weak enough to allow the P0L.4 output to drive a high level after reset.

4.1.2 BSL entry via P0L.3 and $\overline{\text{NMI}}$

Because of compatibility with 8xC166 systems the BSL entry via $\overline{\text{NMI}}$ is also supported by the C165/C167, but with a little difference to the 8xC166.
The first step, which switches to the internal Boot-ROM, is done with a pulldown device at P0L.3 (P0L.2, P0L.4 and P0L.5 are at high level). The considerations which are necessary for the calculation of the pulldown resistor, are the same as already described for P0L.4.

The second step which starts the BSL routine from the internal Boot-ROM, can be achieved by activating the $\overline{\text{NMI}}$ pin directly after $\overline{\text{RSTIN}}$ is inactive. For detailed information see SAB 80C166 Family ApNotes 'Bootstrap Loader 8xC166'.

Note that it is not recommended to use BSL entry via P0L.3 and $\overline{\text{NMI}}$ because of the hardware effort, which is necessary to start and exit the BSL. Besides the support of BSL entry via P0L.3 and $\overline{\text{NMI}}$ for C165/C167 is not guaranteed for the future. Therefore the further description of the BSL in this application note only refers to BSL entry via P0L.4.

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.
In the second circuit the connection of P0L.4 is switchable. The hardware switch allows to select between normal system start or start via BSL.
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 P0L.4. 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.

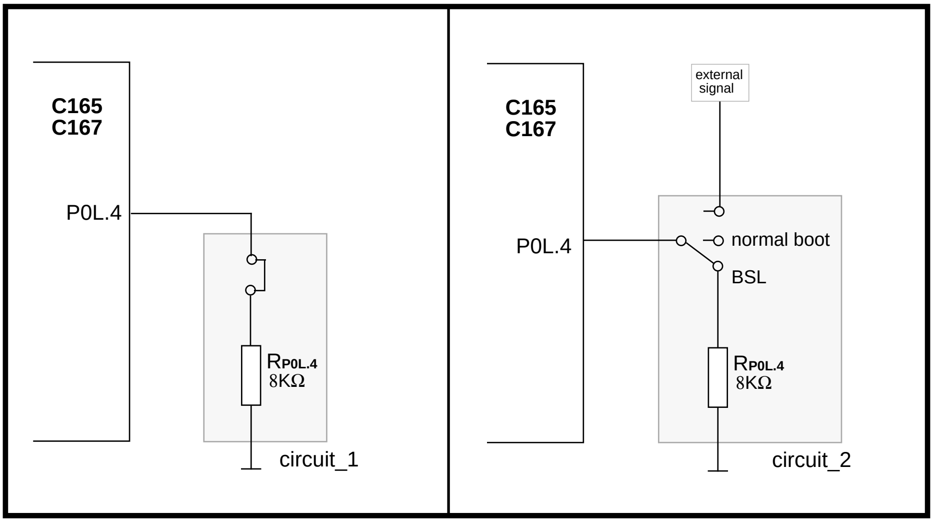


Figure 2: Two exemplary circuit diagrams for BSL entry via plug or switchable circuit

4.3 Special characteristics of the C165/C167 in bootstrap loader mode

If the BSL routine in the internal Boot-ROM is started, there is no automatic timeout or other termination of the BSL mode, if no proper connection to a host is installed. The C165/C167 will remain in BSL mode until a software reset is performed or until a hardware reset (without BSL selection) is performed.

Starting the C165/C167 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	:	$FA40_H$	SYSCON Register	:	$0E00_H$
STKOV Register	:	$FA0C_H$	BUSCON0 Register	:	according to the system startup configuration
S0BG Register	:	value calculated from received <00> byte			

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 reset without BSL activation or a software reset.

If the C165/C167 is in BSL mode, all code fetches from the internal address space 00’0000$_H$ through 00’7FFF$_H$ (if mapped, 01’0000$_H$-01’7FFF$_H$) 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 C165/C167. 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.

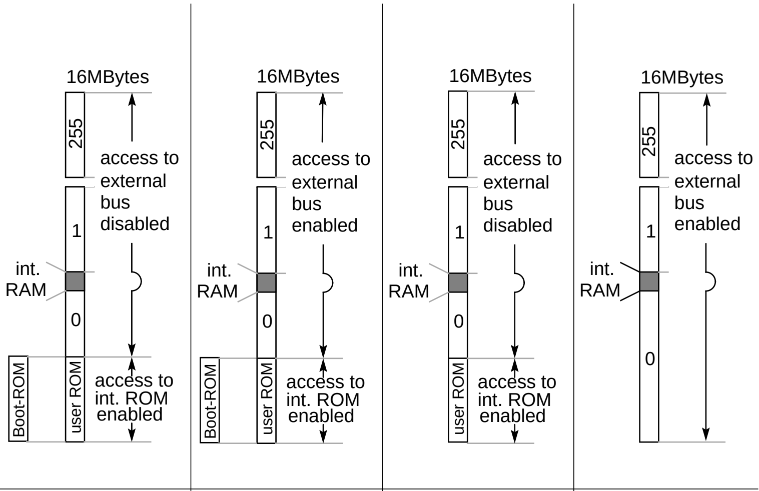
Note that it is strongly recommended not to use half-duplex communication, or else unexpected results will occur. Because the identification byte will be the first byte of the 32-byte data block to be received.

4.4 Reset configuration and memory access

The lines of PORT0 are used during reset to configure the system, such that the state of the PORT0 pins read during reset determines the system startup configuration. The PORT0 pins contain weak pullup devices which are switched on during the entire time that a reset sequence is active and pull the pins to a high level. The state of these pins is read during reset.

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

Configurations I) and II) show the reservation of the first 32 KBytes for internal accesses. The reservation is independent from the $\overline{\text{EA}}$ pin. As already mentioned the accesses to internal address space (00’0000$_H$-00’7FFF$_H$) are different for data or code. All code fetches are done from the internal Boot-ROM which is necessary to start and run the BSL routine after the hardware reset. After the 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.


16MBytes
255
1
0
int.
RAM
Boot-ROM
user ROM
access to external bus disabled
access to external bus enabled
access to int. ROM enabled

BSL mode active	yes	yes	no	no
P0L.4 pin	low	low	high	high
$\overline{EA}$ 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)

1) All accesses to the internal address space ($00'0000_H$ - $00'7FFF_H$) of a ROMless version of the C165/C167 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 C165/C167.

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

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

The upper limit of the baudrate is specified by the maximum deviation of the baudrates between host and C165/C167 which is allowed for a correct serial data transfer. A detailed explanation how to calculate the baudrate deviations between host and ASC0 is shown below.

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%.

Via the contents of the baudrate generator reload value (S0BRL), the baudrate generator of the C165/C167 allows baudrates with the following values:

$$\boxed{B_{165\backslash 167} = \frac{f_{CPU}}{32 \cdot (S0BRL + 1)}}$$

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 division T6/72 of the equation S0BRL is an integer calculation. 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 maximum deviation between the internal initialized baudrate for ASC0 and the baudrate of the host during BSL is active (8 data bits, one stop bit, no parity) has to be lower than or equal to 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:

$$\boxed{F_B = \left| \frac{B_{165\backslash167} - B_{Host}}{B_{165\backslash167}} \right| \cdot 100\ \% \quad , \qquad F_B \le 2.5\ \%}$$

This baudrate deviation is a nonlinear 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 a C165/C167 at 20MHz CPU clock.

Lower limit of the baudrate:

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

Upper limit of the baudrate:

The real maximum baudrate (B_{Max}) which is supported by the baudrate generator of the C165/C167 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 C165/C167 leaves the allowed tolerance of ±2.5%. See figure 4.

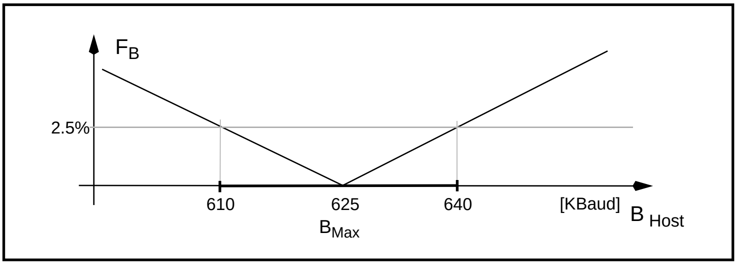


Figure 4: Real maximum baudrate for the BSL

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.

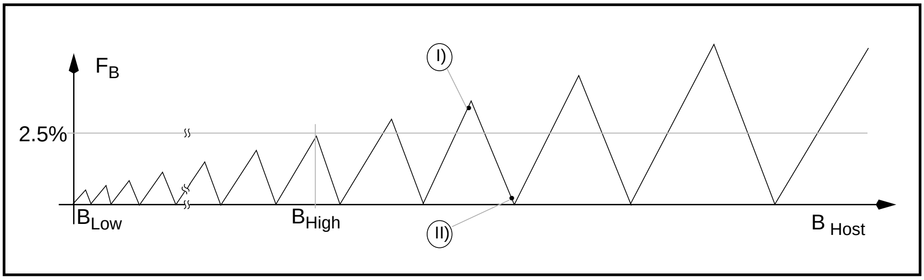


Figure 5: Principle baudrate deviation between host and C165/C167 when the BSL is active

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 C165/C167 is calculated in the following steps:

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

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

$$B_{165\backslash 167} = \frac{f_{CPU}}{32 \cdot (S0BRL + 1)} \quad , \quad B_{165\backslash 167} = \frac{20 \cdot 10^6}{32 \cdot (14 + 1)} \quad , \quad B_{165\backslash 167} = 41666 Baud$$

$$F_{B-I)} = \left| \frac{B_{165\backslash 167} - B_{Host-I)}}{B_{165\backslash 167}} \right| \cdot 100 \quad \% \quad , \quad F_{B-I)} = \left| \frac{41666 - 40300}{41666} \right| \cdot 100 \quad \%$$

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

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 C165/C167 in bootstrap loading mode, a 32 KByte address range is reserved for internal accesses. The default address range for this 32 KByte block is 00’0000$_H$ through 00’7FFF$_H$. 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,#xxxY xZxx xxxx xxxxb    ; Y: (ROMS1=1): maps 32 KByte block to seg. 1
                                             ; Z: (ROMEN=0): disables internal ROM
      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 results 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 ignored.

3.) 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.

4.) 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.

5.) 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.

6.) After mapping, the 32 KByte address range (reserved for internal accesses) to memory location 01’0000$_H$ through 01’7FFF$_H$ or after disabling the 32 KByte block, in both cases, the external memory can now be accessed in the address range 00’0000h through 00’7FFF$_H$.

5.2 Program examples how to program the BSL

The following programs show how to load the user software to the internal or external memory with the BSL and run it. This is done in three examples. Example 1 checks the communication between the C165/C167 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 physical stack size of 256 words but the maximum stack size which is allowed in the program examples is only 32 words. The location of the used stack is shown in the memory maps for the user software in figure 6 and figure 8. 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 C165/C167 and host via ASC0 will be tested. See example program 67USRSW1.ASM. The program ‘user_software_1’ is loaded via BSL to start address 00’FA40$_H$ and end address 00’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 C165/C167 is still in BSL mode after the test routine has been executed.

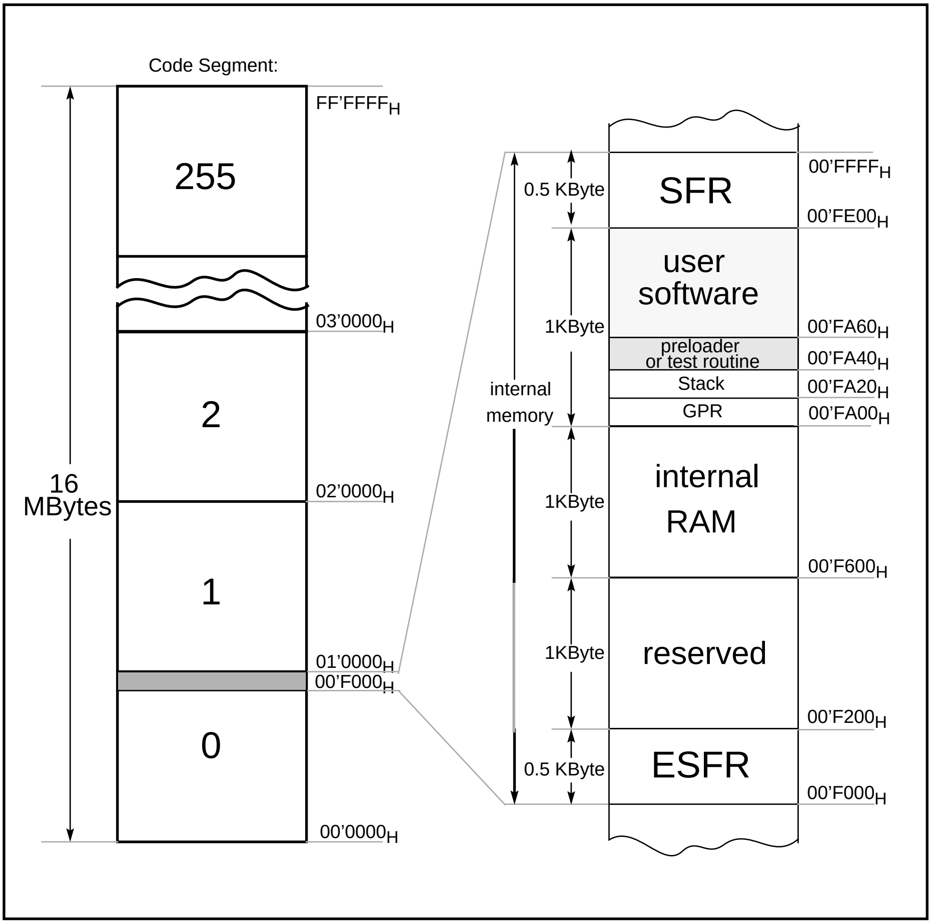


Figure 6: Memory map for user software loaded to the internal RAM

5.2.2 Example 2: Load user software to internal memory

An example how to do this is shown in the two programs 67PRELD1.ASM and 67USRSW2.ASM. See appendix. The preloader is loaded via the BSL and located from memory address 00'FA40$_H$ to end address 00'FA5F$_H$. The preloader 67PRELD1. ASM loads the user software to the internal RAM (start address 00'FA60$_H$) of the C165/C167. When using only the internal RAM, the largest possible user program loaded by the preloader is 1952 bytes. In this case the user program has to be seperated into two parts. The memory range of part one is from address 00’FA60$_H$ up to 00’FDFF$_H$ and part two from address 00’F600$_H$ up to 00’F9FF$_H$. 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 00’FA60$_H$. The program 67USRSW2.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 C165/C167 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 C165/C167 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.

Now an explanation of example 3 in detail:

After performing a hardware reset including the external circuit_1 or circuit_2 for BSL entry, the BSL loads the preloader (67PRELD1.ASM, same program as in example 2) via ASC0 to the internal RAM of the C165/C167 from memory address 00’FA40$_H$ to end address 00’FA5F$_H$.

After the BSL has started the preloader, that loads the external loader (67EXTLD1.ASM) via ASC0 to the internal RAM storing from memory address 00’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.

2) The user software is loaded via ASC0 and stored from external memory address 00’1000$_H$, this location is user defined.

3) A jump instruction to the user program start address 00’1000$_H$ is stored to the reset vector location 00’0000$_H$.

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

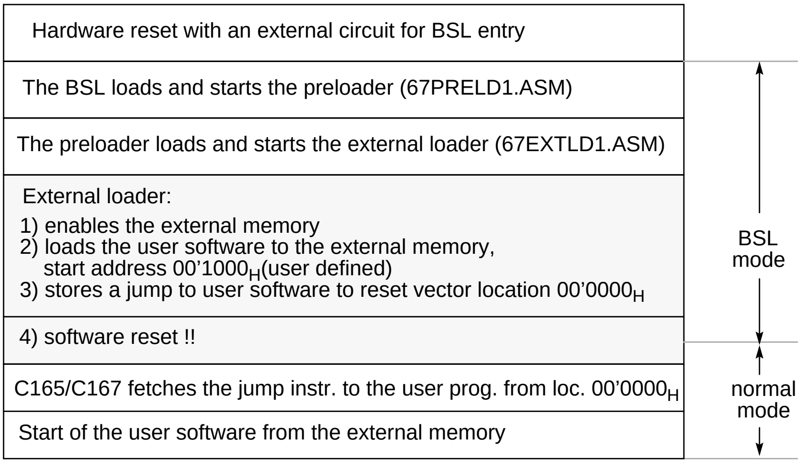

Hardware reset with an external circuit for BSL entry
The BSL loads and starts the preloader (67PRELD1.ASM)
The preloader loads and starts the external loader (67EXTLD1.ASM)
External loader:
1) enables the external memory
2) loads the user software to the external memory, start address 00'1000$_H$(user defined)
3) stores a jump to user software to reset vector location 00'0000$_H$
4) software reset !!
C165/C167 fetches the jump instr. to the user prog. from loc. 00'0000$_H$
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 at location 00'0000$_H$.

This location now contains a jump to the start address 00'1000$_H$ of the user program.

The user program (67USRSW3.ASM) sends back acknowledge bytes <07> in an endless loop via ASC0. Because of the software reset the C165/C167 has to be initialized. This is done in the program 67USRSW3.ASM, it disables the Watchdog Timer, sets the Context Pointer, the Stack Pointer and initializes ASC0 (Baudrate 9600 Baud).

Note that the C165/C167 leaves the BSL mode after the software reset. The user software is performed in normal mode.

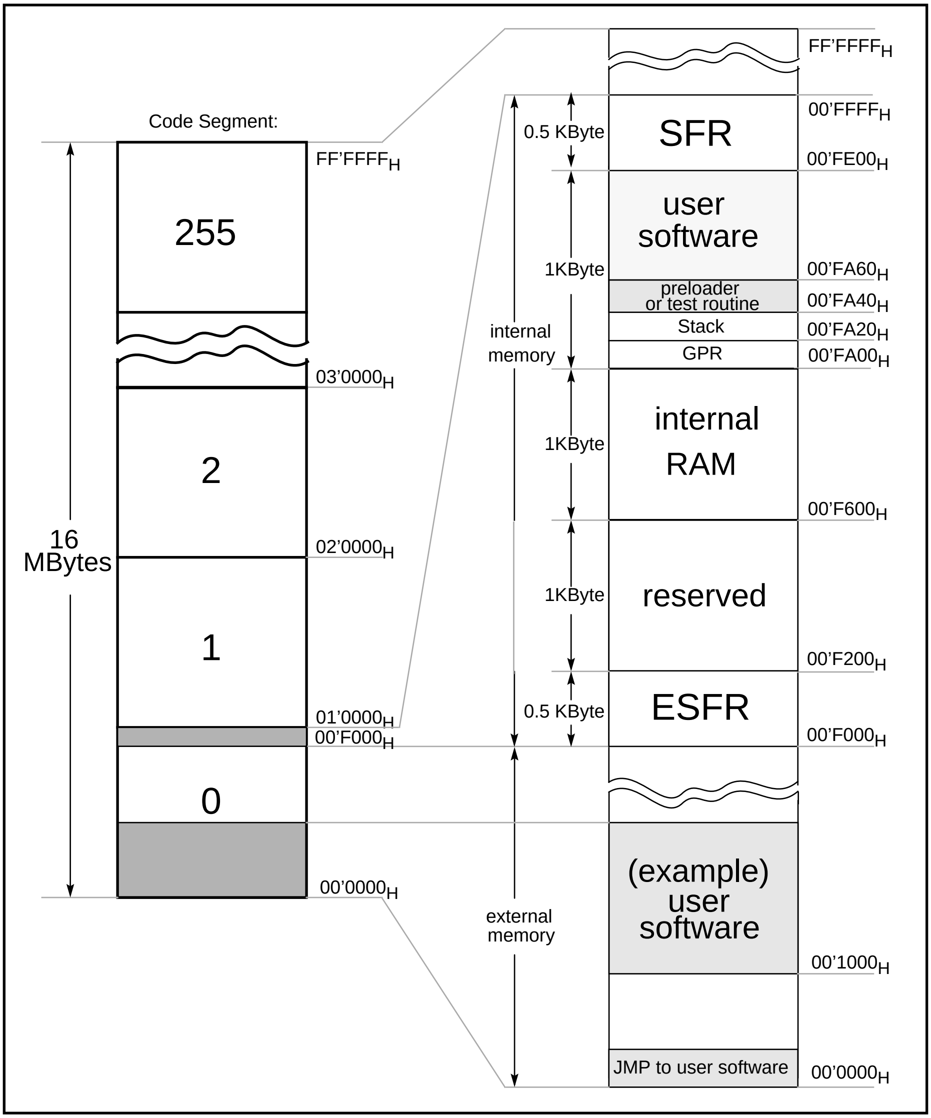


Figure 8: Memory map for user software loaded to the external memory

6 Appendix

6.1 USER_SOFTWARE_1

```
$LISTALL

$DEBUG

;+----------------------------------------------------------------------+
;| Program Name  : 67USRSW1.ASM                        Rev. : 1.1     |
;| Description   : is loaded via the BSL and sends back an            |
;|                 acknowledge byte <03>.                             |
;|                 Program start address: 0FA40h                      |
;+----------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0     |
;| Author        : Mariutti                          Date :  21.1.94 |
;+----------------------------------------------------------------------+
;| Revision History: Rev.: 1.0 ==> 1.1: stack size                    |
;|                                                                    |
;+----------------------------------------------------------------------+
NAME   USER_SOFTWARE_1

       SSKDEF 0                ; system stack size 256 words
       REGDEF R0

SEC1   SECTION CODE AT 0FA40h  ; program start address 0FA40h
MAIN   PROC  TASK  INTNO=0
;+-----------------------------------------------------------------------+
;|     initialization    | BSL RESET values: Stack Pointer   = 0FA40h   |
;|                       | physic. stack size: only 32 words are allowed|
;|                       |                  Context Pointer = 0FA00h   |
;+-----------------------------------------------------------------------+
       MOV   STKUN ,#0FA40h    ; set Stack Underflow Pointer Register
       MOV   STKOV ,#0FA20h    ; set Stack Overflow  Pointer Register
       MOV   SYSCON,#00000h    ; max. physical stack size: 256 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. 2 PRELOADER_1

```
$LISTALL
$DEBUG
;+----------------------------------------------------------------------+
;| Program Name  : 67PRELD1.ASM                      Rev. : 1.1         |
;| Description   : is loaded via the BSL. 67PRELD1 loads a prog. to     |
;|                 the C165/C167 with a length of 928 bytes via ASC0.|
;|                 Program start address: 0FA40h                        |
;+----------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0        |
;| Author        : Mariutti                          Date :  21.1.94    |
;+----------------------------------------------------------------------+
;| Revision History: Rev.: 1.0 ==> 1.1: stack size                      |
;+----------------------------------------------------------------------+
NAME   PRELOADER1
       SSKDEF 0               ; system stack size 256 words
       REGDEF R0
SEC1   SECTION CODE AT 0FA40h ; program start address 0FA40h
MAIN   PROC  TASK  INTNO=0
;+------------------------------------------------------------------------+
;|     initialization  | BSL RESET values:  Stack Pointer   = 0FA40h   |
;|                     | physic. stack size: only 32 words are allowed|
;|                     |                  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)
USRSW:                        ; start address 0FA60h
SEC2   ENDS
       END
```

6.3 USER_SOFTWARE_2

```
$LISTALL
$DEBUG
;+--------------------------------------------------------------------+
;| Program Name  : 67USRSW2.ASM                          Rev. : 1.1   |
;| Description   : is loaded via 67PRELD1 and sends back an           |
;|                 acknowledge byte <05>.                             |
;|                 Program start address: 0FA60h                      |
;+--------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0      |
;| Author        : Mariutti                          Date :  21.1.94 |
;+--------------------------------------------------------------------+
;| Revision History: Rev.: 1.0 ==> 1.1: stack size                    |
;|                                                                    |
;+--------------------------------------------------------------------+

NAME   USER_SOFTWARE_2

       SSKDEF 0               ; system stack size 256 words
       REGDEF R0

SEC1   SECTION CODE AT 0FA60h  ; program start address 0FA60h
MAIN   PROC TASK  INTNO=0
;+----------------------------------------------------------------------+
;|     initialization    | BSL RESET values:  Stack Pointer    = 0FA40h |
;|                       | physic. stack size: only 32 words are allowed|
;|                       |                   Context Pointer  = 0FA00h |
;+----------------------------------------------------------------------+
       MOV   STKUN ,#0FA40h    ; set Stack Underflow Pointer Register
       MOV   STKOV ,#0FA20h    ; set Stack Overflow  Pointer Register
       MOV   SYSCON,#00000h    ; max. physical stack: 256 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  : 67EXTLD1.ASM                       Rev. : 1.1      |
;| Description   : stores a user program to the external memory       |
;|                 installs a jump to the user program of a C165/C167|
;|                 system and the NMI# trap routine. Performs a sw   |
;|                 RESET. Program start address: 0FA60h               |
;+-------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0      |
;| Author        : Mariutti                          Date :  21.1.94 |
;+-------------------------------------------------------------------+
;| Revision History: Rev.: 1.0 ==> 1.1: stack size                    |
;|                                                                    |
;+-------------------------------------------------------------------+
NAME   EXTERNAL_LOADER_1

       SSKDEF 0                ; system stack size 256 words
       REGDEF R0-R5

sec1   SECTION CODE AT 0FA60h
main   PROC  TASK  INTNO=0
;+---------------------------------------------------------------------+
;| enable ext. memory, dummy jump segment and reload all Data Page Pointer|
;+---------------------------------------------------------------------+
       MOV   SYSCON,#00000h    ; disable int. ROM (enable ext. memory)
                               ; max. physical stack size: 256 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 and init. ext. bus to system configuration                  |
;+---------------------------------------------------------------------+
       MOV   BUSCON0,#006BFh   ; enable external bus, 16bit nonmux
;+---------------------------------------------------------------------+
;| 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
;+---------------------------------------------------------------------+
;| leave the BSL mode via a software reset                            |
;+---------------------------------------------------------------------+
       SRST                    ; software reset

sec1   ENDS
       END
```

6.5 USER_SOFTWARE_3

```
$LISTALL
$DEBUG
;+---------------------------------------------------------------------+
;| Program Name  : 67USRSW3.ASM                          Rev. : 1.1    |
;| Description   : initializes the system after a software reset,      |
;|                 baudrate: 9600Baud and sends back an acknowledge    |
;|                 byte <07> via ASC0. Program start address: 01000h |
;+---------------------------------------------------------------------+
;| Assembler     : BSO/Tasking 80166 Assembler       Rev. :  3.0      |
;| Author        : Mariutti                          Date :  21.1.94 |
;| Rev. History  : Rev.: 1.0 ==> 1.1: stack size                      |
;+---------------------------------------------------------------------+
NAME     USER_SOFTWARE_3
         SSKDEF 0                ; system stack size 256 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 register bank
         MOV   SP     ,#0FA40h   ; set stack pointer
         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
         MOV   SYSCON ,#00000h   ; max. physical stack size: 256 words,
                                 ; enable external bus
;+----------------------------------------------------------------------+
;|       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
```