SIEMENS

Microcontrollers

ApNote AP1644

☑ additional file
AP164401.EXE available

The Bootstrap Loader Tool

The Bootstrap Loader Tool can download a program (in Intel Hex Format) from the PC via the bootstrap loader of the microcontroller in any location of the internal or external RAM.

Author: Frank Riemenschneider, Siemens Mircoelectronics, San Jose

AP1644 ApNote - Revision History		
Actual Revision : 02.98	Previous Revison: none (Original Version)	
Page of actual Rel.	Page of prev. Rel.	Subjects changes since last release

1. Introduction: Some Facts about the Bootstrap Loader

A built-in bootstrap loader which is implemented in all microcontrollers of the C166-family (except in the C163 and the C161V) provides a mechanism to load a startup program via the serial interface. In this case no external (ROM) memory or an internal ROM is required. The bootstrap loader moves code/data into the internal RAM, but it is also possible to transfer data via the serial interface into an external RAM using a second level loader routine. ROM memory (internal or external) is not necessary.

The Bootstrap Loader may be used to load the complete application software into ROMless systems or it may load temporary software into complete systems for testing or calibration.

It is useful for standard system startup as well as only for special occasions like system maintenance (firmware update) or end-of-line programming or testing.

1.1 Entering the Bootstrap Loader

The microcontroller enters the BootStrap Loader (BSL) mode, if pin P0L.4 is sampled low at the end of a hardware reset. In this case the built-in bootstrap loader is activated independent of the selected bus mode. The bootstrap loader code is stored in a special Boot-ROM, no part of the standard mask ROM or Flash memory area is required for this.
After entering BSL mode and the respective initialization the microcontroller scans the RxD line to receive a zero byte, i.e. one start bit, eight ‘0’ data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the asynchronous serial channel (ASC) accordingly and switches pin TxD to output. Using this baudrate, an identification byte is returned to the host that provides the loaded data.

This identification byte identifies the device to be booted. The following codes are defined:

8xC166:	55 H	
C161, C165:	B5 H	
C167:	C5 H	(previous versions returned A5 H).

When the microcontroller has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are **marked**):

Watchdog Timer: **Disabled**	Register SYSCON: 0E00 H
Context Pointer CP: FA00 H	Register STKUN: FA40 H
Stack Pointer SP: FA40 H	Register STKOV: FA0C H 0<->C
Register S0CON: **8011 H**	Register BUSCON0: acc. to startup config.
Register S0BG: acc. to ‘00’ byte	P3.10 / TXD0: ‘ **1**’
	DP3.10: ‘ **1**’

Other than after a normal reset the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Pin TxD is configured as output, so the microcontroller can return the identification byte.

Note: Even if the internal ROM is enabled, no code can be executed out of it.

The hardware that activates the BSL during reset may be a simple pull-down resistor on P0L.4 for systems that use this feature upon every hardware reset. You may want to use a switchable solution (via jumper or an external signal) for systems that only temporarily use the bootstrap loader.
After sending the identification byte the ASC receiver is enabled and is ready to receive the initial 32 bytes from the host. A half duplex connection is therefore sufficient to feed the BSL.

1.2 Memory Configuration after Reset

The configuration (ie. the accessibility) of the microcontroller's memory areas after reset in Bootstrap-Loader mode differs from the standard case. Pin EA is not evaluated when BSL mode is selected, and accesses to the internal ROM area are **partly redirected**, while the microcontroller is in BSL mode. All code fetches are made from the special Boot-ROM, while data accesses read from the internal user ROM. Data accesses will return undefined values on ROMless devices.

Note: User software should not try to execute code from the internal ROM area while the BSL mode is still active, as these fetches will be redirected to the Boot-ROM. The Boot-ROM will also "move" to segment 1, when the internal ROM area is mapped to segment 1. Therefore, the external access to a RAM mapped in the internal ROM area in segment 0 is only possible if the bit "ROMS1" of register SYSCON is set.

1.3 Loading the Startup Code

After sending the identification byte the BSL enters a loop to receive 32 bytes via ASC. These bytes are stored sequentially into locations 00'FA40 H through 00'FA5F H of the internal RAM. So up to 16 instructions may be placed into the RAM area.
To execute the loaded code the BSL then jumps to location 00'FA40 H , i.e. the first loaded instruction. The bootstrap loading sequence is now terminated, the microcontroller remains in BSL mode, however.
Most probably the initially loaded routine will load additional code or data, as an average application is likely to require substantially more than 16 instructions. This second receive loop may directly use the pre-initialized interface ASC to receive data and store it to arbitrary user-defined locations.
This second level of loaded code may be the final application code. It may also be another, more sophisticated, loader routine that adds a transmission protocol to enhance the integrity of the loaded code or data. It may also contain a code sequence to change the system configuration and enable the bus interface to store the received data into external memory.

In all cases the microcontroller will still run in BSL mode, i.e. with the watchdog timer disabled and limited access to the internal ROM area.

1.4 Exiting Bootstrap Loader Mode

In order to execute a program in normal mode, the BSL mode must be terminated first. The microcontroller exits BSL mode upon a software reset (ignores the level on P0L.4) or a hardware reset (P0L.4 must be high then!). After a reset the microcontroller will start executing from location 00'0000 H of the internal ROM or the external memory, as programmed via pin EA.

2. The Features of the Bootstrap Loader Tool

♦ The Bootstrap Loader Tool running on a PC communicates with the microcontroller via the asynchronous serial interface.

♦ With the Bootstrap Loader Tool it is possible to download:
- the initial 32 bytes that are expected by the µC (mode 1)
- program code in the Internal RAM (mode 2)
- program code in the External RAM at any location (mode 3).

♦ The program code in all three download modes must be in Intel Hex-File Format.

♦ After downloading the code it is possible to select between:
- an Absolute Inter-Segment Jump to any (start-)address.
- a software reset to exit the BSL mode.

♦ The Bootstrap Loader Tool is independent of any hardware. Registers which are important for the External Bus Configuration (i.e. BUSCONx, SYSCON,...) can be modified easily according to any user hardware.

♦ It is possible to save the settings of these registers that enable the external memory access and to reload them before starting the download process.

♦The Bootstrap Loader Tool is a 32bit application that runs under Windows95.

3. Software Overview and Installation

The software packet contains:
- the Installation Software
- assembly and C-code examples (source code, Intel Hex-Files, Linker Files)
- configuration files for the evaluation boards KitCON-161, KitCON-163/165, KitCON-164 and KitCON-167 from Phytec
- a special startup file for KEIL C-Compiler C-166 (“Botstart.a86”)

Please read before installation:

To extract the compressed installation files double-click on the executable file. Once you have 'unzipped' the files double-click on "setup.exe" to start the installation and then follow the instructions given by the setup program.

At the beginning of the setup process you can modify the directory where the Bootstrap Loader Tool will be installed (default path: 'C:\Program Files\Bootstrap Loader Tool').
After the installation you will find in this directory also
- the Manual (PDF-Format),
- the KitCON Configuration Files (*.cfg)
- the assembly and C-code example programs (Intel Hex-Files, Linker Files and source code for KEIL C166).

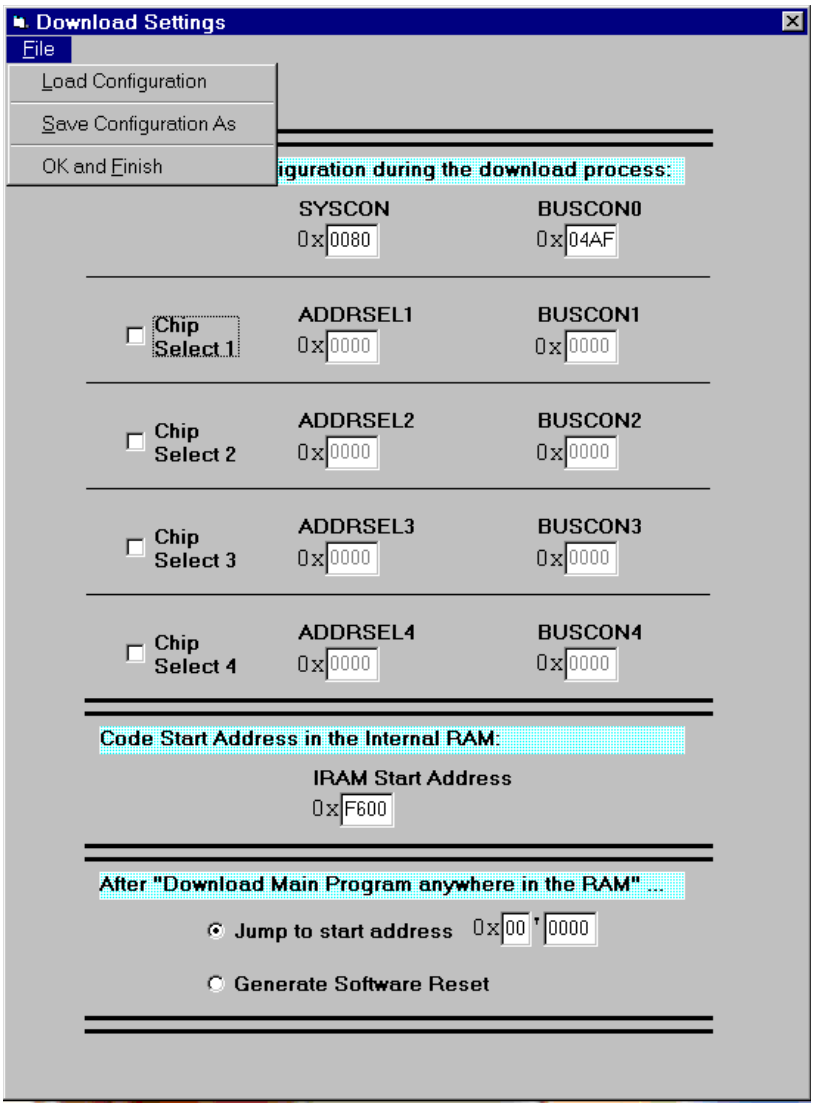
4. How to use the Bootstrap Loader Tool

In this chapter you will find a step by step description for using the Bootstrap Loader Tool. To start the Bootstrap Loader Tool click on the button “Bootstrap Loader Tool” that you can find in the list of your program files.

4.1 The Download-Settings window

The first window that appears after starting the Bootstrap Loader Tool will be the Download-Settings window. Here you can specify different settings concerning the download process. In the following paragraphs you will get an explanation of all the settings that are important in the three download modes. More explanations of the different modes you will find in the next paragraph “The main window of the Bootstrap Loader Tool”.

The Download-Settings window:



4.1.1 Mode 1: Proper Bootstrap File (32 bytes)

In this case you don't have to care about this window. *None* of the settings have an influence on the download process in this mode. If you only want to send the 32 bytes to the microcontroller then you can click directly on "OK and Finish" to leave this window and to go to the main window of the Bootstrap Loader Tool.

4.1.2 Mode 2: Main Program in Internal RAM

For this download mode only the *"Code Start Address in the Internal RAM"* is of importance. Beginning at this address the Bootstrap Loader Tool will store each instruction after the other independently of the addresses indicated in the Intel Hex-File! (use this mode only to download short assembly programs!). At the end of the download process a jump to this address will be executed.

4.1.3 Mode 3: Main Program anywhere in the RAM

In this download mode all settings will influence the download process. After sending a 32 byte long first level loader routine to the microcontroller, a second level loader routine will be downloaded. This second routine will receive the main program code and store it at the addresses indicated in the Intel Hex-File.

4.1.3.1 External Bus Configuration during the download process

At the beginning of the second level loader routine the values of the registers SYSCON, BUSCONx and ADDRSELx are loaded with the values that you can specify in this Download-Settings window.

• *Register SYSCON*
The most important bits of this register that concern the download process are ROMS1 (=SYSCON.12), BYTDIS (=SYSCON.9) and WRCFG (=SYSCON.7). In BSL mode this register is automatically set to 0E00 H. That means that the Internal ROM area is mapped to segment 0 (in this case it is impossible to access an External RAM mapped to the internal ROM area in segment 0 !), the pin BHE is disabled and WR retain their normal function (this depends on your hardware !).
If you want to store your code in an external RAM, mapped to the internal ROM area in segment 0, make sure that you set the bit ROMS1. (The Internal RAM can be accessed even if bit ROMS1 is set.) And you have to modify the bits BYTDIS and WRCFG in accordance with your hardware. All the other bits of register SYSCON (Stack Size, Segmentation,...) can be modified according to your application but have no direct influence on the download process.
The default value is 1080 H (= Internal ROM area in segment 1, Pin WR acts as WRL, pin BHE acts as WRH).

• *Chip Select x options*
If your code shall be stored in an External RAM that needs to be selected by one of the 4 CS lines on Port 6, you have to enable the corresponding Chip Select option. After selecting the option you are able to specify the contents of the registers ADDRSELx and BUSCONx.

• *Registers ADDRSELx*
Here you can specify the address windows and the individual bus characteristics within these areas. Note that after a reset these registers are set to 0000 H. You must choose the right value for this register to generate correctly the Chip Select signal, if your External RAM use it. Please refer to the user manual of the microcontroller to get detailed information. The default value of these registers is 0000 H.

• *Registers BUSCONx*
The properties of a bus cycle are controlled via registers BUSCONx. Four of these registers (BUSCON1...BUSCON4) have an address select register (ADDRSEL1...ADDRSEL4) associated with them. All accesses that are not covered by the address select registers are then controlled via BUSCON0.
You must choose the right values for the BUSCONx registers to access the External RAM in the correct bus mode.
The default value of register BUSCON0 is 0680 H (= External 16-bit Demultiplexed Bus enabled, slowest Time Control,...). The default value of the other registers BUSCONx is 0000 H (= External Bus disabled,...).
Please refer to the user manual of the microcontroller to get detailed information about these registers.

4.1.3.2 Code Start Address in the Internal RAM

The “Code Start Address in the Internal RAM” indicates the address in the Internal RAM where the second level loader routine will be located. This second level loader routine consists of 170 (=0AAh) bytes. It will receive the main program code and store it at the addresses indicated in the Intel Hex-File. You must choose a start address of an Internal RAM area where these 170 bytes won’t overwrite any other data (i.e. Stack, GPR,...). The default value is 00’F600 H.

4.1.3.3 After “Download Main Program anywhere in the RAM”...

You can select here the instruction that shall be executed after the code has been downloaded. Either an Absolute Inter-Segment Jump to the specified address or a software reset can be executed.
Note: After a software reset the External Bus Configuration is determined by the configuration of port 0 and the first instruction will be fetched at address 00h. In addition, all SFR registers will be set to their reset values (especially the registers SYSCON, BUSCON and ADDRSEL will change their contents). If your program shall be executed after reset you have to make sure that the first instruction will be fetched from the External RAM where you have downloaded your code (by pulling down the corresponding bits of

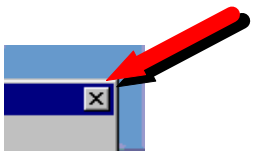
port 0). A software reset will turn off the BSL mode. (The content of the external or internal RAM is not modified after reset.)

4.1.4 Save and Load your configuration

It is possible to save the settings that you have selected in this window: click on “Save Configuration As” in the menu File. The configuration will be saved in a (text-)file with the extension (*.cfg).
To gain time you can load these settings the next time, when you start the Bootstrap Loader Tool: click on “Load Configuration” in the File menu. Configuration files for the evaluation boards KitCON-161, KitCON-163/165, KitCON-164 and KitCON-167 from Phytec are included in the software packet.

4.1.5 Exiting the Download-Settings window

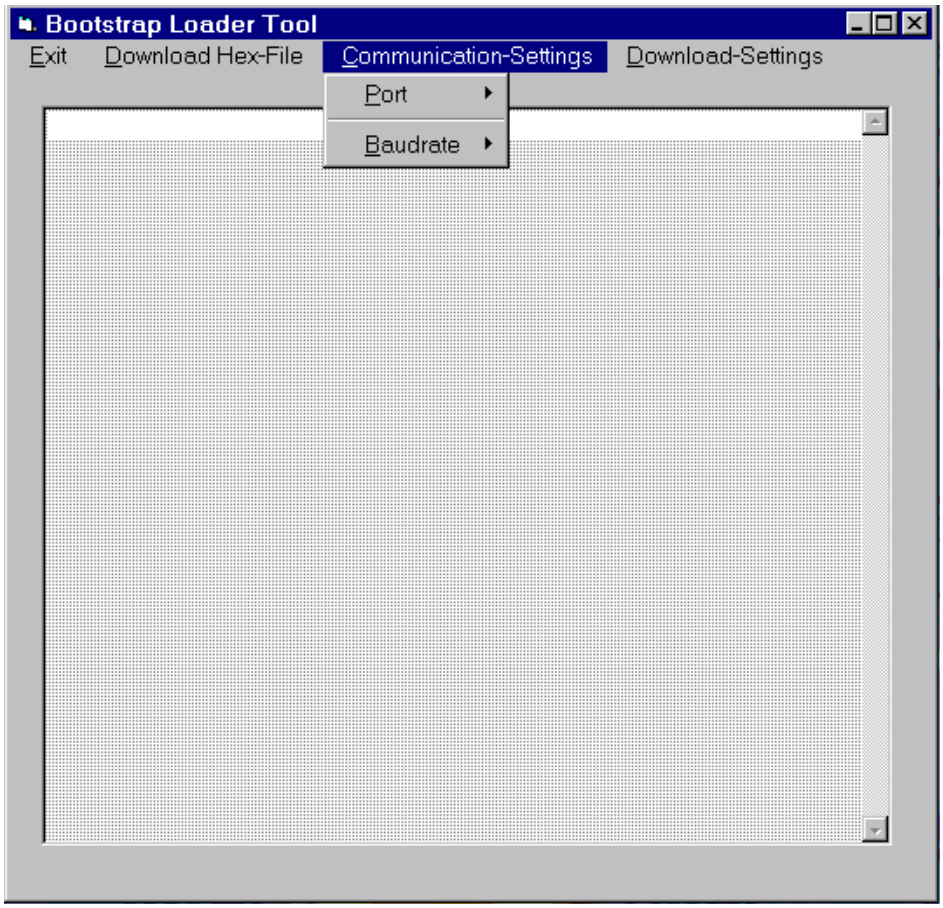
After you have finished with all necessary settings you can exit the Download-Settings window by clicking on “OK and Finish”. If you have modified anything but you didn’t save this modification a warning will appear. If you don’t want to save the modification you must confirm this by clicking once again on “OK and Finish”.
To cancel and to leave the window without any modification you on the button with the cross on the top in the right corner:



4.2 The Main window of the Bootstrap Loader Tool

4.2.1 The Communication-Settings menu

The first thing you have to do is to select a serial port (COM1 or COM2) and a baudrate (2400, 4800, 9600, 14400 or 19600 bauds) for the communication between the PC and the microcontroller. The default settings are 9600 bauds and port COM 1.

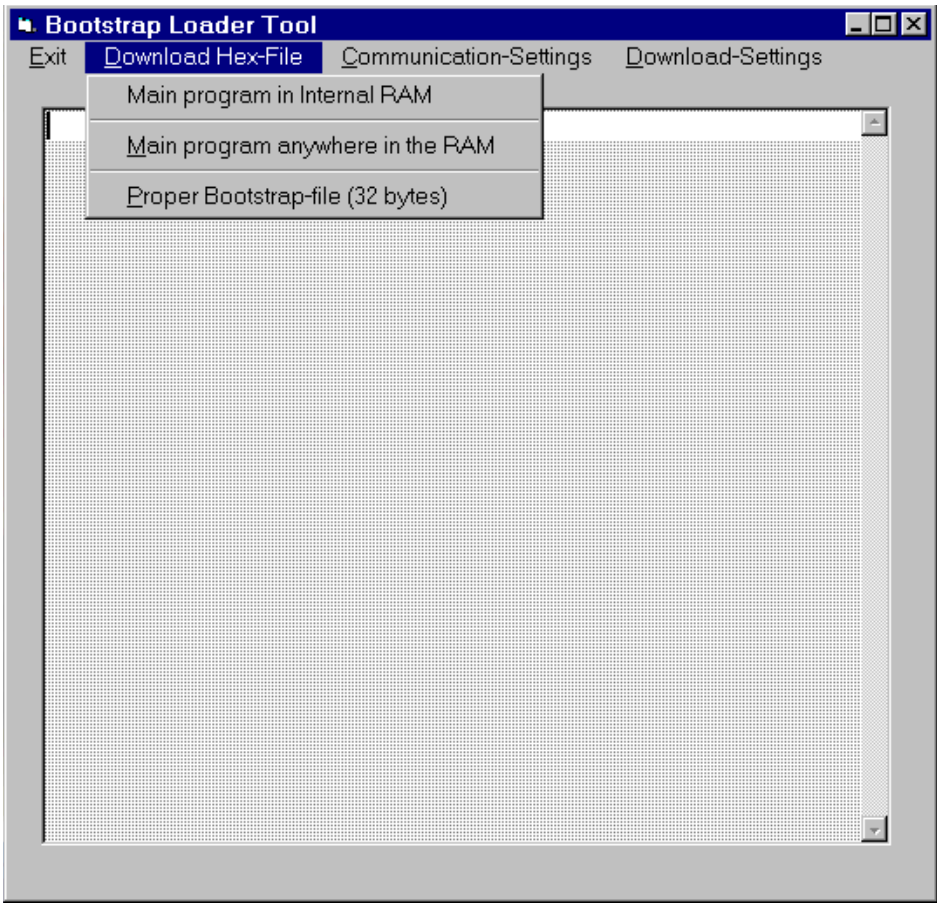


4.2.2 The Download-Settings menu

If you want to change any download settings you can open the Download-Settings window again by clicking there.

4.2.3 The Download Hex-File menu

In this menu you can start the download process. There are three different download modes. In every mode the first thing you will have to do is to select a Intel Hex-File of your program which shall be send to the microcontroller.



4.2.3.1 Mode 1: Proper Bootstrap-file (32 bytes)

Choose this mode if you only want to send the 32 bytes that the microcontroller expects after he has sent the identification byte. The Bootstrap Loader Tool will send the first 32 data bytes of the Intel Hex-File that you have selected via the serial interface. All address information that the Intel Hex-File contains will be ignored. There will be no transmission error verification when sending the 32 bytes to the microcontroller. The code will be executed immediately after all 32 bytes have been received. Use only an Assembler to generate the Hex-File.

4.2.3.2 Mode 2: Main program in Internal RAM

If you want to download a program into the Internal RAM at the address specified in the Download-Settings window (IRAM Start Address) then you can choose this mode. The

first level loader routine (32 bytes) will be automatically downloaded and executed. This first level loader routine then receives all data bytes of the Intel Hex-File via the serial interface and stores them in the Internal RAM beginning at the IRAM Start Address.
The address information that contains the Intel Hex-File is not evaluated. All address information that the Intel Hex-File contains will be ignored. Therefore, in generally, this mode cannot be used to download a Hex-File generated by a C-code Compiler. It is only possible to download relocatable assembly code (only relative addresses).
Make sure that there is enough space at this address in the Internal RAM so that your code won't overwrite any data or will be overwritten by data (i.e. Stack, GPRs,...). There will be no transmission error verification. After all data bytes have been received by the microcontroller a jump to the 'IRAM Start Address' (defined in the Download-Settings window) will be executed.

4.2.3.3 Mode 3: Main program anywhere in the RAM

If you want to store your program either in the Internal or External RAM then you can use this mode. The first level loader routine (32 bytes) will receive a second level, more sophisticated, loader routine (AA H = 170 bytes) which will be located at the 'IRAM Start Address' that you defined in the Download-Settings window.
This second level loader routine receives now the Intel Hex-File line by line and stores the data bytes at the addresses as indicated in the Intel Hex-File. In this mode you determine the location of the code already when you link your program after compilation or assembling. Code examples for the KEIL Assembler and Compiler can be found in the next chapter.
In addition, after the transmission of every line of the Hex-File, the Bootstrap Loader Tool will execute a checksum error verification to be sure that no transmission error has occurred. At the end of a successful download process either a software reset or an Absolute Inter-Segment Jump to the address indicated in the Download-Settings window will be executed.

5. Assembly-code Examples

The following assembly code examples are written for the KEIL-Assembler C166. They can be used to test the Bootstrap Loader Tool. Every example can be downloaded into a certain memory area depending on the selected download mode. All examples send continuously bytes over the asynchronous serial interface once the microcontroller has received any byte.
The pre-initialized (in BSL mode!) asynchronous serial interface is used to send the bytes with the same baudrate as the program has been downloaded. These transmitted bytes from the microcontroller can be visualized by using the Windows Terminal or HyperTerminal Program. You have to press any key to start this transmission.
You will find the source codes and the Intel Hex-Files for every example in your software packet.

5.1.1 Proper Bootstrap File Example (32 bytes)

This assembler code consists of exactly 32 bytes. Once a byte has been received, the microcontroller will send '32' all the time via the ASC.
This example can be used to test the following download modes:

- Proper Bootstrap File (32 bytes)
- Main program in Internal RAM
- Main program anywhere in the RAM

```
$ MOD167
$ INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE AT 0FA40H

_32BYTES  PROC NEAR

          BCLR S0RIR              ;clear transmitter interrupt request flag
          NOP
   WAIT_0:    JNB S0RIR, WAIT_0       ;wait until any byte has been received
                              ;(this is necessary to synchronize the
                                          ;terminal program)

Loop:
          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#33h         ;send a '3'
        WAIT_1:     JNB S0TIR, WAIT_1       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#32h         ;send a '2'
        WAIT_2:     JNB S0TIR, WAIT_2       ;wait until data has been transmitted

          JMPR CC_UC, Loop        ;jump back to loop


_32BYTES  ENDP

BOOTSTRAP ENDS

          END
```

• *Proper Bootstrap File (32 bytes)*
The 32 bytes can be transmitted to the microcontroller (in BSL mode) after receiving the identification byte. They will automatically be stored beginning at the address 00'FA40 H. independently of the addresses in the Hex-File. After the 32nd bit has been received a jump to address 00'FA40 H will execute the program.

• *Main program in Internal RAM*
When this download mode is used, the first level loader routine (32 bytes) is downloaded before the code will be stored in the Internal RAM beginning at address "IRAM Start Address" (specified in the Download-Settings window). The addresses indicated in the Intel Hex-File are ignored (they start at 0FA40 H !). At the end of the downloading process a jump to the IRAM Start Address is executed.

• *Main program anywhere in the RAM*
In this mode the first level loader routine (= 32 bytes located from 00'FA40 H to 00'FA5F H). will download a second level loader routine (= 170 bytes located from "IRAM Start Address" to "IRAM Start Address"+170-1). This second level loader routine then stores the code at the addresses as indicated in the Intel Hex-File (here at 00'FA40 H !) This will overwrite the first level loader routine (it isn't used anymore) that has been located there. To execute the code you should have specified in the Download-Settings window in the section "After Download Main Program anywhere in the RAM" a Jump to start address 0x00'0FA40.

5.1.2 Internal RAM at address 00'F600 h

Once a byte has been received, the microcontroller will send 'F600 ' all the time via the ASC.
This example can be used to test the following download modes:

- Main program in Internal RAM
- Main program anywhere in the RAM

Length: 60 bytes (3CH)

```
$ MOD167
$INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE AT 0F600H

IRAM          PROC NEAR

          BCLR S0RIR              ;clear transmitter interrupt request flag
          NOP
   WAIT_0:    JNB S0RIR, WAIT_0       ;wait until any byte has been received
                              ;(this is necessary to synchronize the
                                        ;terminal program)

Loop:
          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#46h         ;send a 'F'
   WAIT_1:    JNB S0TIR, WAIT_1       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#36h         ;send a '6'
   WAIT_2:    JNB S0TIR, WAIT_2       ;wait until data has been transmitted

           BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_3:    JNB S0TIR, WAIT_3       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_4:    JNB S0TIR, WAIT_4       ;wait until data has been transmitted

           BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#20h         ;send a ' ' (space)
   WAIT_5:    JNB S0TIR, WAIT_5       ;wait until data has been transmitted

          JMPR CC_UC, Loop        ;jump back to loop


IRAM      ENDP

BOOTSTRAP ENDS

          END
```

• *Main program in Internal RAM*
When this download mode is used, the first level loader routine (32 bytes) is downloaded before the code will be stored (by this first level loader routine) in the Internal RAM beginning at address "IRAM Start Address" (specified in the Download-Settings window). The addresses indicated in the Intel Hex-File are ignored (they start at 0F600 H !). At the end of the downloading process a jump to the IRAM Start Address is executed.

• *Main program anywhere in the RAM*
In this mode the first level loader routine (= 32 bytes located from 00'FA40 H to 00'FA5F H). will download a second level loader routine (= 170 bytes located from "IRAM Start Address" to "IRAM Start Address"+170-1). This second level loader routine then stores the code at the addresses as indicated in the Intel Hex-File (here at 00'F600 H !). To execute the code you should have specified in the Download-Settings window in the section "After Download Main Program anywhere in the RAM" a Jump to start address 0x00'0F600.

Note that if you choose IRAM Start Address = 00'F600 H one part of the second level loader routine will be overwritten !
(In this case it doesn't matter because the first 52 bytes of the second level loader routine are already executed and won't be used anymore. If you want really overwrite a part of the second level loader routine that is currently executed then you should download the following example "Internal RAM at address 00'F700 h" and set "IRAM Start Address" = 0xF680, for example. Now you will see that the download process will be interrupted, because a part of the second level loader routine has been overwritten by the code and can't work anymore.)

5.1.3 Internal RAM at address 00’F700 h

Once a byte has been received, the microcontroller will send 'F700 ' all the time via the ASC.
This example can be used to test the following download modes:

- Main program in Internal RAM
- Main program anywhere in the RAM

Length: 60 bytes (3CH)

```
$ MOD167
$INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE AT 0F700H

IRAM      PROC NEAR

          BCLR S0RIR              ;clear transmitter interrupt request flag
          NOP
   WAIT_0:    JNB S0RIR, WAIT_0       ;wait until any byte has been received
                              ;(this is necessary to synchronize the
                                        ;terminal program)

Loop:
          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#46h         ;send a 'F'
   WAIT_1:    JNB S0TIR, WAIT_1       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#37h         ;send a '7'
   WAIT_2:    JNB S0TIR, WAIT_2       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_3:    JNB S0TIR, WAIT_3       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_4:    JNB S0TIR, WAIT_4       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#20h         ;send a ' ' (space)
   WAIT_5:    JNB S0TIR, WAIT_5       ;wait until data has been transmitted

          JMPR CC_UC, Loop        ;jump back to loop


IRAM      ENDP

BOOTSTRAP ENDS

          END
```

5.1.4 Internal RAM at address 00’FD00 h

Once a byte has been received, the microcontroller will send 'FD00 ' all the time via the ASC.
This example can be used to test the following download modes with microcontrollers that have only an internal RAM of 1 kByte which starts at address 00’FA00 H (i.e. C-163):

- Main program in Internal RAM
- Main program anywhere in the RAM

Length: 60 bytes (3CH)

```
$ MOD167
$INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE AT 0FD00H

IRAM      PROC NEAR

          BCLR S0RIR              ;clear transmitter interrupt request flag
          NOP
   WAIT_0:    JNB S0RIR, WAIT_0       ;wait until any byte has been received
                              ;(this is necessary to synchronize the
                                        ;terminal program)

Loop:
          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#46h         ;send a 'F'
   WAIT_1:    JNB S0TIR, WAIT_1       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#44h         ;send a 'D'
   WAIT_2:    JNB S0TIR, WAIT_2       ;wait until data has been transmitted

           BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_3:    JNB S0TIR, WAIT_3       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_4:    JNB S0TIR, WAIT_4       ;wait until data has been transmitted

           BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#20h         ;send a ' ' (space)
   WAIT_5:    JNB S0TIR, WAIT_5       ;wait until data has been transmitted

          JMPR CC_UC, Loop        ;jump back to loop


IRAM      ENDP

BOOTSTRAP ENDS

          END
```

5.1.5 External RAM at address 00’0000 h

Once a byte has been received, the microcontroller will send '0000H ' all the time via the serial interface.
This example can be used to test the following download modes:

- Main program in Internal RAM
- Main program anywhere in the RAM

Length: 70 bytes (46H)

```
$ MOD167
$ INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE AT 00H        ;The following code will be stored at address
                                             ;00H

EX_RAM_00 PROC NEAR

          BCLR S0RIR              ;clear transmitter interrupt request flag
          NOP
   WAIT_0:    JNB S0RIR, WAIT_0       ;wait until any byte has been received
                            ;(this is necessary to synchronize the
                                             ;terminal program)

Loop:
          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_1:    JNB S0TIR, WAIT_1       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_2:    JNB S0TIR, WAIT_2       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_3:    JNB S0TIR, WAIT_3       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_4:    JNB S0TIR, WAIT_4       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#48h         ;send a 'H'
   WAIT_5:    JNB S0TIR, WAIT_5       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#20h         ;send a ' '  (space)
   WAIT_6:    JNB S0TIR, WAIT_6       ;wait until data has been transmitted

          JMPR CC_UC, Loop        ;jump back to loop


EX_RAM_00     ENDP

BOOTSTRAP ENDS

          END
```

- *Main program in Internal RAM*

Note that it is possible to execute this program in the Internal RAM because this example doesn’t use any interrupts. Interrupt vectors are located in the internal ROM area in segment 0.

- *Main program anywhere in the RAM*

In this case it is important that you have configured the external bus correctly.
Make sure that you have mapped the internal ROM to segment 1 (set bit ROMS1). Otherwise the access to an external RAM in that area will be denied and the downloading process won't be possible. (There will be no error message!).
Normally after "Download Main Program (anywhere) in the RAM" you could either jump to address 00'0000 H or execute a software reset to start running the code. But in this case here, if a reset is executed the asynchronous serial channel is no longer initialized (it is pre-initialized in BSL mode). Therefore, you must not select the software reset after download. The reason why there is no example which initializes the ASC and which could be run after a reset is that this initialization of the ASC (i.e. baudrate) depends on the oscillator frequency of the microcontroller.

For your own real application, please note:

The software reset will not modify the content of the External RAM. The code located in the External RAM at address 00h will only be executed if the first instruction after reset is fetched in the External RAM. That means that the external bus configuration in this address space must be determined by the BUSCON0 register. Certain bits of this register are set according to pin EA and port 0 during reset.

5.1.6 External RAM at address 04’0000 h

Once a byte has been received, the microcontroller will send '04’0000 ' all the time via the serial interface.
This example can be used to test the following download modes:

- Main program in Internal RAM
- Main program anywhere in the RAM

Length: 90 bytes (5AH)

```
$ SEGMENTED
$ MOD167
$ INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE AT 040000H        ;The following code will be stored at address
                                            ;04'0000H
EX_RAM_04 PROC NEAR

          BCLR S0RIR              ;clear transmitter interrupt request flag
          NOP
   WAIT_0:    JNB S0RIR, WAIT_0       ;wait until any byte has been received
                            ;(this is necessary to synchronize the
                                        ;terminal program)

Loop:
          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_1:    JNB S0TIR, WAIT_1       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#34h         ;send a '4'
   WAIT_2:    JNB S0TIR, WAIT_2       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#27h         ;send a '''
   WAIT_3:    JNB S0TIR, WAIT_3       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_4:    JNB S0TIR, WAIT_4       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_5:    JNB S0TIR, WAIT_5       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_6:    JNB S0TIR, WAIT_6       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_7:    JNB S0TIR, WAIT_7       ;wait until data has been transmitted

               BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#20h         ;send a ' '  (space)
   WAIT_8:    JNB S0TIR, WAIT_8       ;wait until data has been transmitted

          JMPR CC_UC, Loop        ;jump back to loop

EX_RAM_04 ENDP

BOOTSTRAP ENDS
          END
```

• *Main program anywhere in the RAM*
Only an (Absolute Inter-Segment) Jump to start address 0x04’0000 can execute this code after being downloaded.

5.1.7 External RAM at address 08’0000 h

Once a byte has been received, the microcontroller will send '10’0000 ' all the time via the serial interface.
When using the KEIL Assembler the Output File Format "Intel Hex-386" in Options -> OH166 Object-Hex Converter... must be selected if not no Hex-File will be created.
This example can be used to test the following download modes:

- Main program in Internal RAM
- Main program anywhere in the RAM

Length: 90 bytes (5AH)

```
$ SEGMENTED
$ MOD167
$ INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE AT 080000H         ;The following code will be stored at address
                                             ;04'0000H

EX_RAM_10 PROC NEAR

          BCLR S0RIR              ;clear transmitter interrupt request flag
          NOP
   WAIT_0:    JNB S0RIR, WAIT_0       ;wait until any byte has been received
                              ;(this is necessary to synchronize the
                                        ;terminal program)

Loop:
          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_1:    JNB S0TIR, WAIT_1       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#38h         ;send a '8'
   WAIT_2:    JNB S0TIR, WAIT_2       ;wait until data has been transmitted

               BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#27h         ;send a '''
   WAIT_3:    JNB S0TIR, WAIT_3       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_4:    JNB S0TIR, WAIT_4       ;wait until data has been transmitted

               BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_5:    JNB S0TIR, WAIT_5       ;wait until data has been transmitted

               BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_6:    JNB S0TIR, WAIT_6       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_7:    JNB S0TIR, WAIT_7       ;wait until data has been transmitted

               BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#20h         ;send a ' '  (space)
   WAIT_8:    JNB S0TIR, WAIT_8       ;wait until data has been transmitted

          JMPR CC_UC, Loop        ;jump back to loop

EX_RAM_10 ENDP

BOOTSTRAP ENDS

          END
```

5.1.8 External RAM at address 10’0000 h

Once a byte has been received, the microcontroller will send '10’0000 ' all the time via the serial interface.
When using the KEIL Assembler the Output File Format "Intel Hex-386" in Options -> OH166 Object-Hex Converter... must be selected if not no Hex-File will be created.
This example can be used to test the following download modes:

- Main program in Internal RAM
- Main program anywhere in the RAM

Length: 90 bytes (5AH)

```
$ SEGMENTED
$ MOD167
$ INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE AT 100000H         ;The following code will be stored at address
                                             ;04'0000H

EX_RAM_10 PROC NEAR

          BCLR S0RIR              ;clear transmitter interrupt request flag
          NOP
   WAIT_0:    JNB S0RIR, WAIT_0       ;wait until any byte has been received
                              ;(this is necessary to synchronize the
                                        ;terminal program)

Loop:
          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#31h         ;send a '1'
   WAIT_1:    JNB S0TIR, WAIT_1       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_2:    JNB S0TIR, WAIT_2       ;wait until data has been transmitted

           BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#27h         ;send a '''
   WAIT_3:    JNB S0TIR, WAIT_3       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_4:    JNB S0TIR, WAIT_4       ;wait until data has been transmitted

           BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_5:    JNB S0TIR, WAIT_5       ;wait until data has been transmitted

           BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_6:    JNB S0TIR, WAIT_6       ;wait until data has been transmitted

          BCLR S0TIR              ;clear transmitter interrupt request flag
          MOV S0TBUF,#30h         ;send a '0'
   WAIT_7:    JNB S0TIR, WAIT_7       ;wait until data has been transmitted

           BCLR S0TIR             ;clear transmitter interrupt request flag
          MOV S0TBUF,#20h         ;send a ' '  (space)
   WAIT_8:    JNB S0TIR, WAIT_8       ;wait until data has been transmitted

          JMPR CC_UC, Loop        ;jump back to loop

EX_RAM_10 ENDP

BOOTSTRAP ENDS

          END
```

6. C-code Examples

The following C-code examples are written for the KEIL-Assembler C166. They can be used to test the Bootstrap Loader Tool.
It is *only* possible to download these programs using the download mode 3: “Main Program anywhere in the RAM”. The C-Compiler produces several sections located at different addresses and therefore it is necessary to store the instructions according to the address information of the Intel Hex-File.
There are several examples which don’t use interrupts and which can be allocated therefore at any address areas and there is one that uses an interrupt service routine and that has to be allocated at address 00H.
All examples send continuously a string over the asynchronous serial interface using the pre-initialized (in BSL mode!) asynchronous serial interface. Therefore, you must not select the software reset after download. The transmission is done in the same baudrate as the code has been downloaded by the Bootstrap Loader Tool. The reason why there is no example which initializes the ASC and which could be run after a reset is that this initialization of the ASC (i.e. baudrate) depends on the oscillator frequency of the microcontroller.
The transmitted bytes can be visualized by using the Windows Terminal or HyperTerminal Program. You have to press any key to start this transmission. (The microcontroller waits for any byte to allow the Terminal program to synchronize itself. You will find the source codes, the Intel Hex-Files and the Linker Files for every example in your software package.

There are some additional things to note when using the KEIL C-Compiler:

- The compiler adds a “startup” code to your proper code. This startup code modifies some registers like SYSCON, BUSCONx and ADDRSELx. In many cases (especially when the code cannot not be executed after a software reset and should be executed by a jump to the start address) these registers *must not* be changed. In addition, when you want to jump to the beginning of your code, you have to know the start address.
To avoid that the compiler adds this code and changes these registers and to specify a start address, you have to add your own startup. A startup file which allows to use the Bootstrap Loader Tool without any limits you can find in this software packet (“Bootstart.a86”). code You simply have to specify some few settings in the startup file (make every time a copy before you modify it) and to include the “Bootstart.a86” in your project. See the following examples and KEIL documentation for further information.

- Since the mode 3 “Main Program anywhere in the RAM” must be used for downloading, you determine the location where the Bootstrap Loader Tool will store the code already when you link your program. To tell the Linker where to allocate the different sections of the code you should use a Linker File (same name as project name and with the extension (*.lin)). In “Options” -> “L166 Linker...” -> “Files” you must select “Use command File” and specify the File Name of the Linker File.

6.1 Examples without any interrupts

6.1.1 Internal RAM at 00'F700 h

This example writes all the time the string 'RAM F700 '. Although it shall be downloaded into the internal RAM the mode3 "Main Program anywhere in the RAM" must be used (and not mode 2!) to store the code in the different sections at the right place (see memory map below).

```
#pragma MOD167

#define uword unsigned int
#define ASC_bTxBufFree  S0TBIR
#define ASC_bRxDataReady S0RIR

#include <reg167.h>

void ASC_vSendData(uword Data)
{
  S0TBIR = 0;           // reset transmit buffer interrupt request flag
  S0TBUF = Data;        // load transmit buffer register
}

void main(void)
{
  while (!ASC_bRxDataReady) {};       //wait until any byte will be received by the uC

  S0TBIR = 1;               // indicate that the transmit buffer register is empty

  while (1)
{                       // endless loop that writes 'RAM F700 ' all the time
  while (!ASC_bTxBufFree) {};             //wait until transmission buffer is free
  ASC_vSendData('R');                       //send data
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('A');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('M');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData(' ');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('F');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('7');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};             //wait until transmission buffer is free
  ASC_vSendData(' ');                       //send space

};
}
```

To determine where the Linker shall allocate the different sections the following Linker File has been used:

```
NOVT //no vector table necessary

// Fix classes for SMALL model in free RAM
CLASSES(NCODE(00F700H),
     NDATA(00F700H))
```

Since no interrupts are used, no vector table is necessary. NDATA corresponds to the User stack and NCODE corresponds to the example code. The special startup code has

been allocated at address 00’F700. Note that therefore it can’t be used for microcontroller with less than 2 kBytes. The allocation was not done in the Linker File but in the startup code itself. The first thing you have to do when using this special startup code is to modify the following lines:

```
;
$SET (ABSOLUTE_ = 1)
STARTUP_START EQU  00F700H
;------------------------------------------------------------------------------
```

If you set (ABSOLUTE_ = 1) the startup code will be allocated at the address that you specify in the next line (here 00F700h). If (ABSOLUTE_ = 0) you let the Linker allocate the code.
In the present case the code should be downloaded into the Internal RAM. After the downloading process a jump to this address must be executed. In the ‘Downloading Settings window’ you have to specify that jump address. This address must be the same as you specify it here in the startup file (00’F700h).
If you let the Linker allocate the startup file then you should jump to the reset vector (in general at address 00’0000 H). There is located an absolute jump to the startup code. Of course, this is always possible if you allocate the startup file by yourself or not.

This is the result of the linking process:

```
MEMORY MAP OF MODULE:


START     STOP      LENGTH    TYPE  ALIGN  TGR  GRP  COMB  CLASS   SECTION NAME
=============================================================================
00F700H   00F71FH   000020H   CODE  AT..   ---  ---  PRIV  ICODE   ?C_STARTUP_CODE
00F720H   00F72FH   000010H   DATA  WORD   ---    1  PUBL  NDATA   ?C_USERSTACK
00F730H   00F7AFH   000080H   CODE  WORD   ---    2  PUBL  NCODE   ?PR?NOINT_IR
00FA00H   00FBFFH   000200H   ---   ---    ---  ---  ---   * SYSTEM STACK *
00FC00H   00FC1FH   000020H   DATA  WORD   ---  ---  ---   *REG*   ?C_MAINREGISTERS
```

6.1.2 External RAM at 00’0000 h

This example writes all the time the string ‘0x000000 ’.

```
#pragma MOD167

#define uword unsigned int
#define ASC_bTxBufFree  S0TBIR
#define ASC_bRxDataReady S0RIR

#include <reg167.h>

void ASC_vSendData(uword Data)
{
  S0TBIR = 0;           // reset transmit buffer interrupt request flag
  S0TBUF = Data;        // load transmit buffer register
}



void main(void)
{
  while (!ASC_bRxDataReady) {};       //wait until any byte will be received by the uC

  S0TBIR = 1;         // indicate that the transmit buffer register is empty

  while (1)
{                       // endless loop that writes '0x000000 ' all the time
  while (!ASC_bTxBufFree) {};         // wait until transmission buffer is free
  ASC_vSendData('0');                // send data
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('x');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};         // wait until transmission buffer is free
  ASC_vSendData(' ');                // send space

};
}
```

To determine where the Linker shall allocate the different sections the following Linker File has been used:

```
NOVT //no vector table necessary

// Fix classes for SMALL model in free RAM
CLASSES(NCODE(00000H),
     NDATA(00000H))
```

Since no interrupts are used, no vector table is necessary. NDATA corresponds to the User stack and NCODE corresponds to the example code. The special startup code has been allocated at address 00’0000. This was not done in the Linker File but in the startup code itself. The first thing you have to do when using this special startup code is to modify the following lines:

```
$SET (ABSOLUTE_ = 1)
STARTUP_START EQU  00H
;------------------------------------------------------------------------------
```

After the downloading process a jump to address 00h must be executed. In the ‘Downloading Settings window’ you have to specify that jump address. This address must be the same as you specify it here in the startup file (00’0000h).

This is the result of the linking process:

```
MEMORY MAP OF MODULE:


START     STOP      LENGTH    TYPE  ALIGN  TGR  GRP  COMB  CLASS   SECTION NAME
==============================================================================
000000H   00001FH   000020H   CODE  AT..   ---  ---  PRIV  ---     ?C_STARTUP_CODE
000020H   00002FH   000010H   DATA  WORD   ---    1  PUBL  NDATA   ?C_USERSTACK
000030H   0000AFH   000080H   CODE  WORD   ---    2  PUBL  NCODE   ?PR?NOINT_0
00FA00H   00FBFFH   000200H   ---   ---    ---  ---  ---   * SYSTEM STACK *
00FC00H   00FC1FH   000020H   DATA  WORD   ---  ---  ---   *REG*   ?C_MAINREGISTERS
```

6.1.3 External RAM at 04’0000 h

This example writes all the time the string ‘0x040000 ’.

```
#pragma MOD167

#define uword unsigned int
#define ASC_bTxBufFree  S0TBIR
#define ASC_bRxDataReady S0RIR

#include <reg167.h>

void ASC_vSendData(uword Data)
{
  S0TBIR = 0;           // reset transmit buffer interrupt request flag
  S0TBUF = Data;        // load transmit buffer register
}



void main(void)
{
  while (!ASC_bRxDataReady) {};       //wait until any byte will be received by the uC

  S0TBIR = 1;               // indicate that the transmit buffer register is empty

  while (1)
{                       // endless loop that writes all the time '0x04000 '
  while (!ASC_bTxBufFree) {};         // wait until transmission buffer is free
  ASC_vSendData('0');                // send data
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('x');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('4');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};         // wait until transmission buffer is free
  ASC_vSendData(' ');                // send space

};
}
```

To determine where the Linker shall allocate the different sections the following Linker File has been used:

```
NOVT //no vector table necessary

// Fix classes for SMALL model in free RAM
CLASSES(NCODE(040000H),
     NDATA(040000H))
```

Since no interrupts are used, no vector table is necessary. NDATA corresponds to the User stack and NCODE corresponds to the example code. The special startup code has been allocated at address 04’0000. This was not done in the Linker File but in the startup code itself. The first thing you have to do when using this special startup code is to modify the following lines:

```
;
```

```
$SET (ABSOLUTE_ = 1)
STARTUP_START EQU  040000H
;------------------------------------------------------------------------------
```

After the downloading process a jump to address 04’0000h must be executed. In the ‘Downloading Settings window’ you have to specify that jump address. This address must be the same as you specify it here in the startup file (04’0000h).

This is the result of the linking process:

```
MEMORY MAP OF MODULE:


START     STOP      LENGTH    TYPE  ALIGN  TGR  GRP  COMB  CLASS   SECTION NAME
================================================================================
00FA00H   00FBFFH   000200H   ---   ---    ---  ---  ---   * SYSTEM STACK *
00FC00H   00FC1FH   000020H   DATA  WORD   ---  ---  ---   *REG*   ?C_MAINREGISTERS
040000H   04001FH   000020H   CODE  AT..   ---  ---  PRIV  ---     ?C_STARTUP_CODE
040020H   04009FH   000080H   CODE  WORD   ---    1  PUBL  NCODE   ?PR?NOINT_4
0400A0H   0400AFH   000010H   DATA  WORD   ---    2  PUBL  NDATA   ?C_USERSTACK
```

6.1.4 External RAM at 10’0000 h

This example writes all the time the string ‘0x100000 ’. When using the KEIL Compiler the Output File Format "Intel Hex-386" in Options -> OH166 Object-Hex Converter... must be selected if not, no Hex-File will be created.

```
#pragma MOD167

#define uword unsigned int
#define ASC_bTxBufFree  S0TBIR
#define ASC_bRxDataReady S0RIR

#include <reg167.h>

void ASC_vSendData(uword Data)
{
  S0TBIR = 0;           // reset transmit buffer interrupt request flag
  S0TBUF = Data;        // load transmit buffer register
}



void main(void)
{
  while (!ASC_bRxDataReady) {};            //wait until any byte will be received by the uC

  S0TBIR = 1;         // indicate that the transmit buffer register is empty

  while (1)
{                      // endless loop that writes '0x100000 ' all the time
  while (!ASC_bTxBufFree) {};              //wait until transmission buffer is free
  ASC_vSendData('0');                           //send data
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('x');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('1');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('0');
  while (!ASC_bTxBufFree) {};              //wait until transmission buffer is free
  ASC_vSendData(' ');                           //send space

};
}
```

To determine where the Linker shall allocate the different sections the following Linker File has been used:

```
NOVT //no vector tabel necessary

// Fix classes for SMALL model in free RAM
CLASSES(NCODE(100000H),
     NDATA(100000H))
```

Since no interrupts are used, no vector table is necessary. NDATA corresponds to the User stack and NCODE corresponds to the example code. The special startup code has been allocated at address 10’0000. This was not done in the Linker File but in the startup

code itself. The first thing you have to do when using this special startup code is to modify the following lines:

```
;
$SET (ABSOLUTE_ = 1)
STARTUP_START EQU  100000H
;------------------------------------------------------------------------------
```

After the downloading process a jump to address 10’0000h must be executed. In the ‘Downloading Settings window’ you have to specify that jump address. This address must be the same as you specify it here in the startup file (10’0000h).

This is the result of the linking process:

```
MEMORY MAP OF MODULE:


START     STOP      LENGTH    TYPE  ALIGN  TGR  GRP  COMB  CLASS   SECTION NAME
================================================================================
00FA00H   00FBFFH   000200H   ---   ---    ---  ---  ---   * SYSTEM STACK *
00FC00H   00FC1FH   000020H   DATA  WORD   ---  ---  ---   *REG*   ?C_MAINREGISTERS
100000H   10001FH   000020H   CODE  AT..   ---  ---  PRIV  ICODE   ?C_STARTUP_CODE
100020H   10002FH   000010H   DATA  WORD   ---    1  PUBL  NDATA   ?C_USERSTACK
100030H   1000AFH   000080H   CODE  WORD   ---    2  PUBL  NCODE   ?PR?NOINT_10
```

6.2 Example using an interrupt routine

If your code contains interrupt routines, a vector table must be generated and this vector table has to be located at the bottom of segment 0. Therefore, you don't have any choice where to store this vector table. But it is up to you to locate the interrupt routines anywhere else using a Linker File.

6.2.1 External RAM at 00'0000 h

Here, in this example the interrupt routine also was located in segment 0.

```
#define ASC_bTxBufFree  S0TBIR
#define ASC_bRxDataReady S0RIR
#define T2INT   0x22

#include <reg167.h>

void ASC_vSendData(uword Data)
{
  S0TBIR = 0;           // reset transmit buffer interrupt request flag
  S0TBUF = Data;        // load transmit buffer register
}



void GT1_viIsrTmr2(void) interrupt T2INT     // timer 2 interrupt routine
{
  while (!ASC_bTxBufFree) {};              // wait until transmission buffer is free
  ASC_vSendData('i');                        // send data
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('n');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('t');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('e');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('r');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('r');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('u');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('p');
  while (!ASC_bTxBufFree) {};
  ASC_vSendData('t');
  while (!ASC_bTxBufFree) {};              // wait until transmission buffer is free
  ASC_vSendData(' ');                        // send space

}


void main(void)
{
  while (!ASC_bRxDataReady) {};            //wait until any byte will be received by the uC

  S0TBIR = 1;               // indicate that the transmit buffer register is empty

 /// ---------- Timer 2 Control Register ----------
  ///  timer 2 works in timer mode
  ///  prescaler factor is 1024
  ///  timer 2 run bit is reset
  ///  up/down control bit is reset
  ///  external up/down control is disabled
  T2CON = 0x0005;
  T2    = 0x0000;  //  load timer 2 register

  ///  enable timer 2 interrupt
  ///  timer 2 interrupt priority level(ILVL) = 9
  ///  timer 2 interrupt group level (GLVL) = 2
  T2IC = 0x0066;
```

```
 // globally enable interrupts
  IEN = 1;
  T2R = 1;         //   set timer 2 run bit

  while (1) {};                       // endless loop
}
```

To determine where the Linker shall allocate the different sections the following Linker File has been used:

```
     // vector table now necessary

// Fix classes for SMALL model in free RAM
CLASSES(NCODE(000000H),
     NDATA(000000H),
     IDATA(000000H))
```

NDATA corresponds to the User stack and NCODE corresponds to the example code. The special startup code (IDATA) has been allocated this time by the Linker. This is possible because we don't have to know the start address. After the download process we can just jump to address 00h because at this location there is the reset interrupt vector of the vector table.
In the special startup code we have to select now (ABSOLUTE = 0) to allow the Linker where to allocate the startup code.

```
;
$SET (ABSOLUTE_ = 0)
STARTUP_START EQU  9999H              ;This value is not important if (ABSOUTE = 0)
;-------------------------------------------------------------------------------
```

After the downloading process a jump to address 00h must be executed. In the 'Downloading Settings window' you have to specify that jump address. Now you don't care about the address where is located the startup code.

This is the result of the linking process:

```
MEMORY MAP OF MODULE:


START     STOP      LENGTH    TYPE  ALIGN  TGR  GRP  COMB  CLASS   SECTION NAME
================================================================================
000000H   000003H   000004H   ---   ---    ---  ---  ---   * INTVECTOR TABLE *
000004H   000013H   000010H   DATA  WORD   ---    2  PUBL  NDATA   ?C_USERSTACK
000014H   000033H   000020H   CODE  WORD   ---  ---  PRIV  ICODE   ?C_STARTUP_CODE
000088H   00008BH   000004H   ---   ---    ---  ---  ---   * INTVECTOR TABLE *
00008CH   000133H   0000A8H   CODE  WORD   ---    1  PUBL  NCODE   ?PR?INTERUPT
00FA00H   00FBFFH   000200H   ---   ---    ---  ---  ---   * SYSTEM STACK *
00FC00H   00FC1FH   000020H   DATA  WORD   ---  ---  ---   *REG*   ?C_MAINREGISTERS
```

7. Getting Started with Phytec Evaluation Boards

Configuration Files (that contain the Download-Settings) have been prepared for four evaluation boards from Phytec: KitCON-161, KitCON-163/165, KitCON-164 and KitCON-167. If you have one of these evaluation boards you can start immediately and try to download the examples in the possible memory areas.
But even if you don't have these evaluation boards this chapter will give you an example of how to select the right Download-Settings for different hardware.

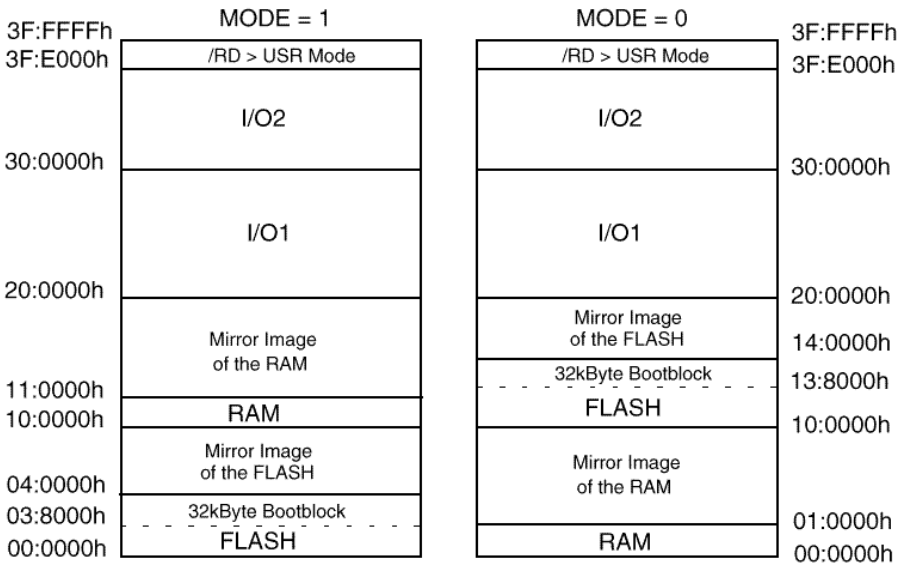
7.1 KitCON-161

The KitCON-161 has TWO Bootstrap Loaders! In addition to the on-chip bootstrap loader (which is not used!) another one has been implemented on this board. The code for this second bootstrap loader is located in the 32kByte Bootblock. But this software implemented bootstrap loader behaves completely equivalent to the on-chip bootstrap loader so that this fact can be ignored.

7.1.1 Memory Models

The KitCON-161 evaluation board is equipped with 64 kByte RAM. You can locate RAM memory, starting from address 0 (to 00'FFFFh) or starting from address 10'0000h (to 10'FFFFh). This can be selected using jumper settings JP4. RAM memory at address 0 has the advantage, that you can use the interrupt vector table for your application. Two GAL devices on the evaluation board enable address decoding. The Chip Select signals are not used.

Here you can see the *memory map* of the C161 evaluation board:



		JP9 (bootstrap loader)
		1+2
JP4 (mapping)	open	flash programming Bootstrap Mode: ON MODE 1 RAM: 10:0000h - 10:FFFFh FLASH: 00:0000h - 03:7FFFh
	closed	debugging with monitor which is downloaded via bootstrap loader Bootstrap Mode: ON MODE 0 RAM: 00:0000h - 00:FFFFh FLASH: 10:0000h - 13:7FFFh

You always have to select the bootstrap loader (JP9 1+2) but you can switch between two different memory models (MODE0 and MODE1, JP4) All other jumpers should be set as indicated in 'Default Settings'.

7.1.2 The Download Settings

- Mode 0
BUSCON0 = 04AFh for 55ns memory devices (0 Waitstate, RW-Delay, no Tri-state, short ALE, 16-Bit Demultiplexed-Bus)
SYSCON = 1080h to select: 'Pin /WR acts as /WRL, pin /BHE acts as /WRH' and to map the Internal ROM area to segment 1. If you don't do this, accesses to the Internal ROM area at segment 0 will be partly redirected to the Boot-ROM when the microcontroller is in BSL mode.

- Mode 1
BUSCON0 = 04AFh for 55ns memory devices (0 Waitstate, RW-Delay, no Tri-state, short ALE, 16-Bit Demultiplexed-Bus)
SYSCON = 0080h to select: 'Pin /WR acts as /WRL, pin /BHE acts as /WRH'.

7.2 KitCON-163/165 and KitCON-167

7.2.1 Memory Models

The KitCON-163/165 and KitCON-167 evaluation boards are equipped with 64 kBytes of RAM and 256 kBytes of flash memory. Memory banks are accessed with C167's Chip-Select signals. /CS0 selects the 256 kBytes flash memory, /CS1 selects 64 kByte RAM memory.

After a reset the default memory configuration is enabled. In this configuration 256 kBytes of flash memory is allocated in address range from 0h to 3'FFFFh and addressed via /CS0. In order to access the 64 kBytes RAM memory, /CS1 has to be activated via configuration of the according ADDRSEL1 and BUSCON1 register.

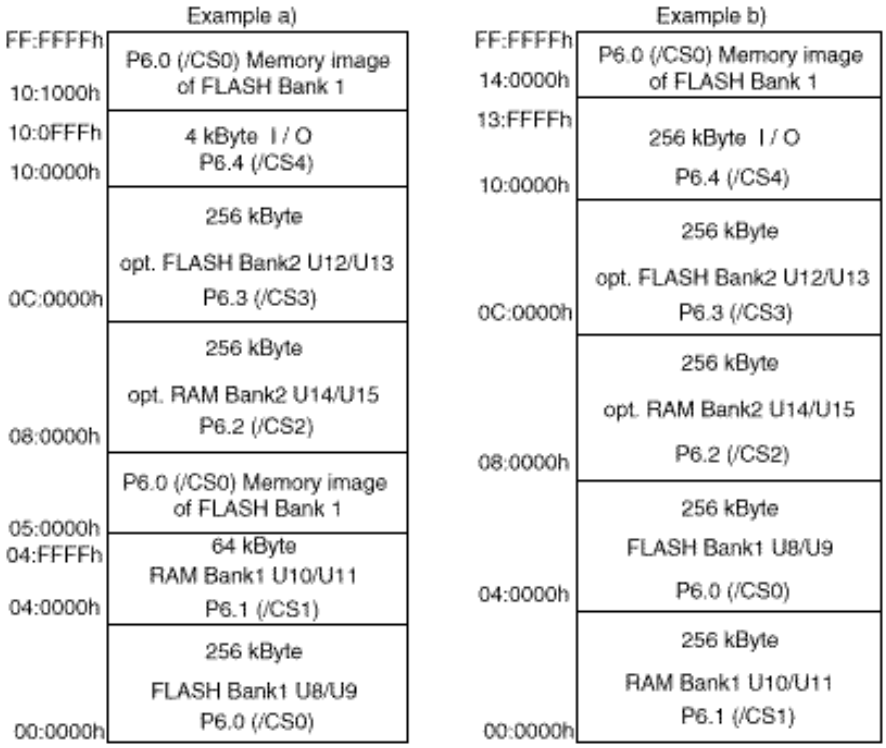


Figure 9: Memory Model Examples

7.2.2 The Download Settings

- Example a)
ADDRESEL1 = 0404h. Selects the memory area 04'0000h - 04'FFFFh via /CS1 (64 KBytes RAM Bank 1 on U10/11)
BUSCON0 = BUSCON1 = 04AFh for 55ns memory devices (0 Waitstate, RW-Delay, no Tri-state, short ALE, 16-Bit Demultiplexed-Bus)
SYSCON = 0080h to select: 'Pin /WR acts as /WRL, pin /BHE acts as /WRH'.

- Example b)
ADDRESEL1 = 0006h. Selects the memory area 00'0000h - 03'FFFFh via /CS1 (256 KBytes RAM Bank 1 on U10/11)
BUSCON0 = BUSCON1 = 04AFh for 55ns memory devices (0 Waitstate, RW-Delay, no Tri-state, short ALE, 16-Bit Demultiplexed-Bus)
SYSCON = 1080h to select: 'Pin /WR acts as /WRL, pin /BHE acts as /WRH' and to map the Internal ROM area to segment 1. If you don't do this, accesses to the Internal ROM area at segment 0 will be partly redirected to the Boot-ROM when the microcontroller is in BSL mode.

7.2.3 How to enter Bootstrap Loader Mode

Please refer also to your Hardware Manual for additional information !

- KitCON-163/165
Close Jumper JP1 (1+2), open Jumper J2 (1+2) and close Jumper JP3 (2+3).

- KitCON-167
Close Jumper JP2 (1+2).

7.3 KitCON-164

7.3.1 Memory Models

The KitCON-164 evaluation boards is equipped with 64 kBytes of RAM and 256 kBytes of flash memory. Memory banks are accessed with an address decoding that generates the memory's Chip-Select signals according to figure 9b.

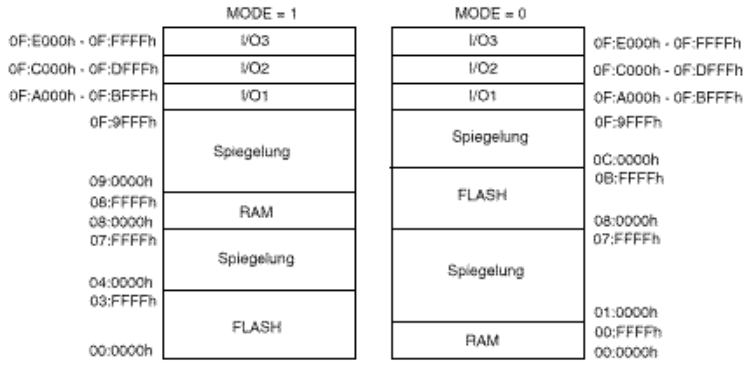


Figure 9b: Memory Model at Standard Memory (64kByte SRAM, 256kByte Flash)

Jumper JP4 selects between the two memory models. Open JP4 to work with mode 1 and close JP4 to get mode 0.

7.3.2 The Download Settings

- Mode 0
BUSCON0 = 04AFh for 55ns memory devices (0 Waitstate, RW-Delay, no Tri-state, short ALE, 16-Bit **Multiplexed**-Bus)
SYSCON = 1080h to select: 'Pin /WR acts as /WRL, pin /BHE acts as /WRH' and to map the Internal ROM area to segment 1. If you don't do this, accesses to the Internal ROM area at segment 0 will be partly redirected to the Boot-ROM when the microcontroller is in BSL mode.

- Mode 1
BUSCON0 = 04AFh for 55ns memory devices (0 Waitstate, RW-Delay, no Tri-state, short ALE, 16-Bit **Multiplexed**-Bus)
SYSCON = 0080h to select: 'Pin /WR acts as /WRL, pin /BHE acts as /WRH'.

7.3.3 How to enter Bootstrap Loader Mode

Please refer also to your Hardware Manual for additional information !
Close Jumper JP8 (1+2) to enable the bootstrap loader mode.

8. Appendix

8.1 First level loader routine

Here you can find the assembly code of the First level loader routine (32 bytes). The KEIL Assembler has been used to generate the Hex-File.

```
$INCLUDE (REG167.INC)

BOOTSTRAP  SECTION CODE   AT 0FA40H

BYTE_COUNTER  EQU  R1
ADDRESS       EQU  R2
NUMBER_BYTES  EQU  ??H                    ;depending on the program size
TRANS_START   EQU  22H
STARTADDRESS  EQU  ??H                    ;depending on settings
                                        ;IRAM Start Address

;*******************************************************************************
BOOT      PROC NEAR

START:         MOV S0TBUF, #TRANS_START             ;send start byte
          MOV BYTE_COUNTER, #NUMBER_BYTES
          MOV ADDRESS, #STARTADDRESS

LOOP:
READ:          BCLR S0RIR
     WAIT:     JNB S0RIR, WAIT
          MOVB [ADDRESS], S0RBUF               ;store byte at ADDRESS
               ADD ADDRESS, #1                 ;inc ADDRESS
               CMPD1 BYTE_COUNTER, #1
          JMPR CC_UGE, LOOP

          JMPA CC_UC, STARTADDRESS             ;jump to IRAM Start Address


BOOT      ENDP

BOOTSTRAP ENDS

          END
```

8.2 Second level loader routine

Here you can find the assembly code of the Second level loader routine (AA H = 170 bytes). The KEIL Assembler has been used to generate the Hex-File.

```
$SEGMENTED
$INCLUDE (REG167.INC)
ASSUME DPP3: SYSTEM

BOOTSTRAP  SECTION CODE AT 0F600H

READ_BYTE EQU  RL0
READ_WORD EQU  R0

ADDRESS        EQU  R1
ADDRESS_LOW    EQU  RL1
ADDRESS_HIGH   EQU  RH1

STARTADDRESS   EQU  R2
NUMBER_DATA    EQU  R3
_256_          EQU  R4
SEGMENT_NR     EQU  R5
TEMP_VAR  EQU  RL6
CHECKSUM  EQU  R7

END_OF_FILE    EQU  PSW.6
TRANS_START    EQU  55H


BOOT      PROC NEAR

START:    MOV SEGMENT_NR, #00H

          MOV SYSCON,  #1234H              ;register initialization
          MOV BUSCON0, #1234H

          MOV ADDRSEL1,#1234H
               MOV BUSCON1, #1234H

          MOV ADDRSEL2,#1234H
               MOV BUSCON2, #1234H

          MOV ADDRSEL3,#1234H
               MOV BUSCON3, #1234H

          MOV ADDRSEL4,#1234H
               MOV BUSCON4, #1234H


          CALLR INIT
               MOV S0TBUF, #TRANS_START        ;send start byte

  CONTINUE:   CALLR READ_LINE             ;read one line of the Hex-File
          CALLR CHECK_SUM             ;verify the checksum byte

          JNB END_OF_FILE, CONTINUE

      wait0:  JNB S0TIR, wait0

FINISH:       SRST                    ;or JMPS ..,..

;*******************************************************************************
INIT:         BCLR END_OF_FILE            ;some initializations
          MOV _256_, #256
          BCLR S0RIR
          RET
;*******************************************************************************
READ:                                 ;sub routine that receives one byte
     WAIT1:    JNB S0RIR, WAIT1
          MOVB READ_BYTE, S0RBUF
          BCLR S0RIR
          ADD CHECKSUM, READ_WORD
          RET
```

```
;*******************************************************************************************
READ_LINE:     MOV CHECKSUM, #00H               ;sub routine that receives one line
               MOV READ_WORD, #00H

          CALLR READ
          CMPB READ_BYTE, #0
          JMPR CC_EQ, DATA_RECORD
               CMPB READ_BYTE, #1
          JMPR CC_EQ, EOF
               CMPB READ_BYTE, #2
          JMPR CC_EQ, SEGMENT_ADDR_RECORD
               CMPB READ_BYTE, #4
          JMPR CC_EQ, SEGMENT_ADDR_RECORD
               RET
;-------------------------------------------------------------------------------------------
DATA_RECORD:   CALLR READ                  ;data record line
          MOV NUMBER_DATA, READ_WORD

          CALLR READ
          MOVB ADDRESS_HIGH, READ_BYTE
          CALLR READ
          MOVB ADDRESS_LOW, READ_BYTE

     LOOP:     CALLR READ
          EXTS SEGMENT_NR, #1
          MOVB [ADDRESS], READ_BYTE        ;store data byte
          ADD ADDRESS, #1
          CMPD1 NUMBER_DATA, #01H
          JMPR CC_NE, LOOP
          RET
;-------------------------------------------------------------------------------------------
SEGMENT_ADDR_RECORD:                       ;segment address record line
          CALLR READ
          MOV SEGMENT_NR, READ_WORD
          RET
;-------------------------------------------------------------------------------------------
EOF:                              ;end of file line
          BSET END_OF_FILE
          RET
;*******************************************************************************************
CHECK_SUM:     MOV MDL, CHECKSUM           ;calculation of the checksum byte
          NOP
          DIVU _256_
          NOP
          MOVB TEMP_VAR, MDH
          NEG TEMP_VAR
          MOV S0TBUF,TEMP_VAR
          BCLR S0TIR
          RET
;*******************************************************************************************
BOOT      ENDP

BOOTSTRAP ENDS

          END
```

8.3 C-code Startup File (Extract)

Here you find the special Startup File “Bootstart.a86” that should be added to any C-code when you want to start the code by jumping to the beginning and not by a software reset. In most parts it is the same as the startup file from KEIL (“start167.a86”). The main difference is the part that is **marked**. For further information see the KEIL-Compiler manual.

```
$MOD167                    ; Define C167 mode
;
;
;    M O D I F I E D    S T A R T U P     F I L E
;
;
;------------------------------------------------------------------------------
;  (This file is part of the C166 Compiler package
;  Copyright KEIL ELEKTRONIK GmbH 1996)        has been modified !
;------------------------------------------------------------------------------
;  START167.A66:  This code is executed after processor reset and provides
;                 the startup sequence for the C167/C165/C163 processor.
;
;  This file should be included when you want to download C-code programs
;  into the External RAM. If you don't include it, the compiler will auto-
;  matically add a startup file and will probably overwrite the contents
;  of the registers SYSCON and BUSCON1..4 that have been specified in
;  the Bootstrap-Loader program.
;
;
;    It is still possible to change the stack size and enable or
;    disable some initialization options like in the original
;    start up file from KEIL
;
;
;
;
;
;------------------------------------------------------------------------------
; Setup model-dependent Assembler controls
$CASE
$IF NOT TINY
$SEGMENTED
$ENDIF
;------------------------------------------------------------------------------
; First you have to determine how the startup code should be located. Either you locate
; the startup code at an absolute address (ABSOLUTE_ = 1) or you let the linker allocate
; your code (ABSOLUTE_ = 0).
; If you select an absolute address then you can specify this address by modifying the
; constant "STARTUP_START".
; This address will be the address of the first instruction of your code. If you want to
; start your code after the downloading process by a jump instruction then you have to
; select an absolute address and then to jump to this address!
;
$SET (ABSOLUTE_ = 1)
STARTUP_START EQU  00F700H
;------------------------------------------------------------------------------
;
; STKSZ: Maximum System Stack Size selection  (SYSCON.13 .. SYSCON.15)
;  Defines the system stack space which is used by CALL/RET and PUSH/POP
;  instructions.  The system stack space must be adjusted according the
;  actual requirements of the application.
$SET (STK_SIZE = 0)
;     System stack sizes:
;       0 = 256 words (Reset Value)
;       1 = 128 words
;       2 =  64 words
;       3 =  32 words
;       4 = 512 words
;       5 = not implemented
;       6 = not implemented
;       7 = no wrapping (entire internal RAM use as STACK, set size with SYSSZ)
; If you have selected 7 for STK_SIZE, you can set the actual system stack size
; with the following SSTSZ statement.
```

```
SSTSZ    EQU  200H ; set System Stack Size to 200H Bytes
;
;
; USTSZ: User Stack Size Definition
;  Defines the user stack space available for automatics.  This stack space is
;  accessed by R0.  The user stack space must be adjusted according the actual
;  requirements of the application.
USTSZ    EQU  10H  ; set User Stack Size to 10H Bytes.
;
;
; WATCHDOG: Disable Hardware Watchdog
; --- Set WATCHDOG = 1 to enable the Hardware watchdog
$SET (WATCHDOG = 0)
;
;
; CLR_MEMORY: Disable Memory Zero Initialization of RAM area
; --- Set CLR_MEMORY = 0 to disable memory zero initilization
$SET (CLR_MEMORY = 0)
;
; INIT_VARS: Disable Variable Initialization
; --- Set INIT_VARS = 0 to disable variable initilization
$SET (INIT_VARS = 0)
;
; DPPUSE:  Re-assign DPP registers
; --- Set DPPUSE = 0 to reduce the code size of the startup code, if you
;                    are not using the L166 DPPUSE directive.
$SET (DPPUSE = 0)
;
;------------------------------------------------------------------------------
```