AN12871
SCI Bootloader for S08PB16

Rev. 0 — May 2020 Application Note

Contents

1 Introduction

The bootloader is a small flash resident code in the Microcontrollers (MCU) that is implemented to download the application code to on-chip non-volatile memory flash in S08PB series. Instead of using a dedicated debug interface, the user only uses the communication interface (for example, SCI) to upgrade the MCU firmware.

For this case, the bootloader requires the tool to download the user application code using the serial communication interface without P&E Multilink or CodeWarrior IDE.

This document introduces how to implement the bootloader with a SCI interface on the S08PB16-EVK board by using the PC tool, **win_hc08sprg.exe**, in the AN2295SW software package (available on www.nxp.com). The document also demonstrates an example of how to configure the bootloader code and application code.

The implementation of the SCI bootloader for S08PB16 is based on the CodeWarrior 11.1 development environment (you must install the service pack: CodeWarrior MCU 11.1 Service Pack for S08PB and S08PLS), SCI bootloader code, s19 file (application code), and the S08PB16-EVK board.

2 Overview

You can obtain details about development board S08PB16-EVK from www.nxp.com/S08PB16-EVK. Figure 1 shows the tools and codes that you can download from NXP website.

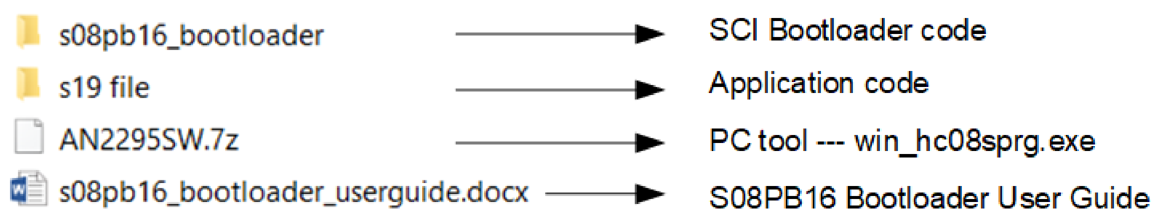

s08pb16_bootloader
SCI Bootloader code
s19 file
Application code
AN2295SW.7z
PC tool --- win_hc08sprg.exe
s08pb16_bootloader_userguide.docx
S08PB16 Bootloader User Guide

Figure 1. Tools and codes for bootloader

- PC tool: **win_hc08sprg.exe** is a host GUI software tool used in the PC. It is a free tool available in the AN2295SW software package. You can find this package on www.nxp.com. You can get the **win_hc08sprg.exe** tool in the folder *...\AN2295SW \masters\release*. The tool **win_hc08sprg.exe** is referred as PC tool in this document. The PC tool is used to decode the s19 file and transfer the application code to the target MCU via SCI interface, which is widely used in S08 products to implement bootloader for MCU. Refer to AN2295 application note for detailed development information and instructions on the PC tool.
- SCI bootloader code: SCI bootloader code is downloaded to MCU as a resident code, which is executed after MCU reset. The code can communicate with the PC tool to check if there is an application code that needs to be downloaded to flash memory.

- Application code: When using the bootloader method to download the application code, the user only needs to modify the linker file (Project.prm) to redistribute its memory, and then use PC tool to download its binary code (*mtim.abs.s19*).

The binary file, *mtim.abs.s19*, provided in the s19 file folder has been modified and can be used for test directly. The mtim project is also available in the s19 file folder for reference.

2.1 SCI bootloader process

The PC tool is compatible with FC protocol and communicates with the MCU via SCI interface. The protocol is called FC protocol because one significant character (acknowledge or ACK) 0xFC or 11111100b is used. The FC protocol communicates between the PC and MCU to reprogram the MCU. Refer to AN2295 application note for details about the FC protocol.

The bootloader commands, *ident/read/write/erase/quit*, are sent by the PC tool to the MCU to program the s19 file. Figure 2 shows the SCI bootloader process when using PC tool. The process uses the bootloader commands available in AN2295SW to program MCU.

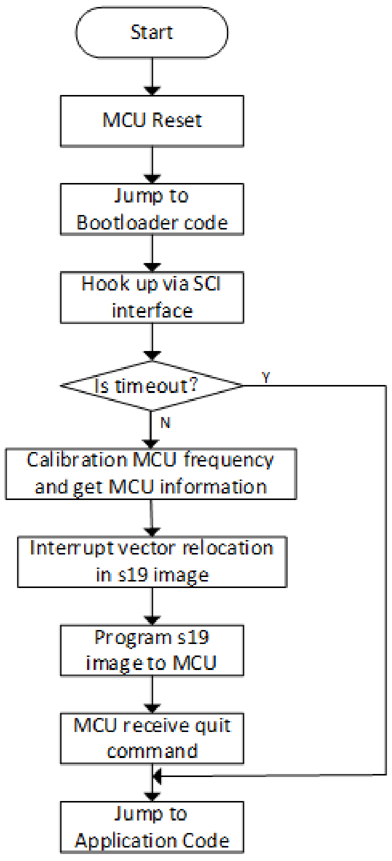


Figure 2. SCI bootloader process

Following is the SCI bootloader process:

1. The bootloader process begins when the MCU is reset. Then MCU jumps to the address which is loaded from vector 0 (0xFFFE:0xFFFF) to execute the SCI bootloader code.
2. The MCU sends the 0xFC to hook up with the PC tool. The hook up between MCU and PC tool is successful when PC tool receives the correct character 0xFC from the MCU. Then PC tool sends back 0xFC to MCU immediately.

   If the MCU and PC tool hook up fails in the specified time, the MCU bootloader process ends. The MCU jumps to the application code, and the next steps would not execute.

3. When MCU is calibrated to the correct clock or MCU is operating at the correct data rate, it sends the ACK character 0xFC to the PC tool to stop the calibration process. Then the PC tool sends IDENT command to the MCU, which sends back the information predefined in the bootloader code.
4. The PC tool copies the contents of the interrupt vector table in the s19 file to the new address. The contents in origin vector table are invalid. The interrupt vector table in s19 file is relocated.
5. The MCU receives the READ, ERASE, and WRITE commands from the PC tool to operate flash and download the s19 file.
6. The PC tool sends QUIT command to the MCU when s19 file downloads to MCU successfully. The MCU jumps to the address of s19 file relocate reset vector to execute application code, and the MCU bootloader process ends.

2.2 Memory allocation

The on-chip memory in S08PB16 consists of 1 KB RAM and 16 KB flash program, I/O and control/status registers. The RAM address space is 0x0040 - 0043F, and the flash address space is 0xC000 - 0xFFFF.

For flash allocation, 3 KB flash space 0xF400 - 0xFF9F is reserved for the SCI bootloader code. Before downloading application code to MCU, the PC tool decodes the s19 file (application code) and copies the content of the address space 0xFFB0 - 0xFFFF to the address 0xF3B0 - 0xF3FF. The content of the address space 0xFFB0 - 0xFFFF is invalid for the s19 file.

NOTE

In RAM address space, 3 bytes of RAM 0x43D - 0x43F are reserved for bootloader vector redirection in the SCI bootloader code, so they cannot be used by the application code. The bootloader code address and relocated interrupt vector table address cannot be assigned in the same sector. Otherwise, a part of the bootloader code will be erased.

For specific allocation and configuration of SCI bootloader code and application code memory, see Chapter 3.1.1 and Chapter 3.2.1 in this document.

Figure 3 below shows memory allocation for SCI bootloader code and application code.

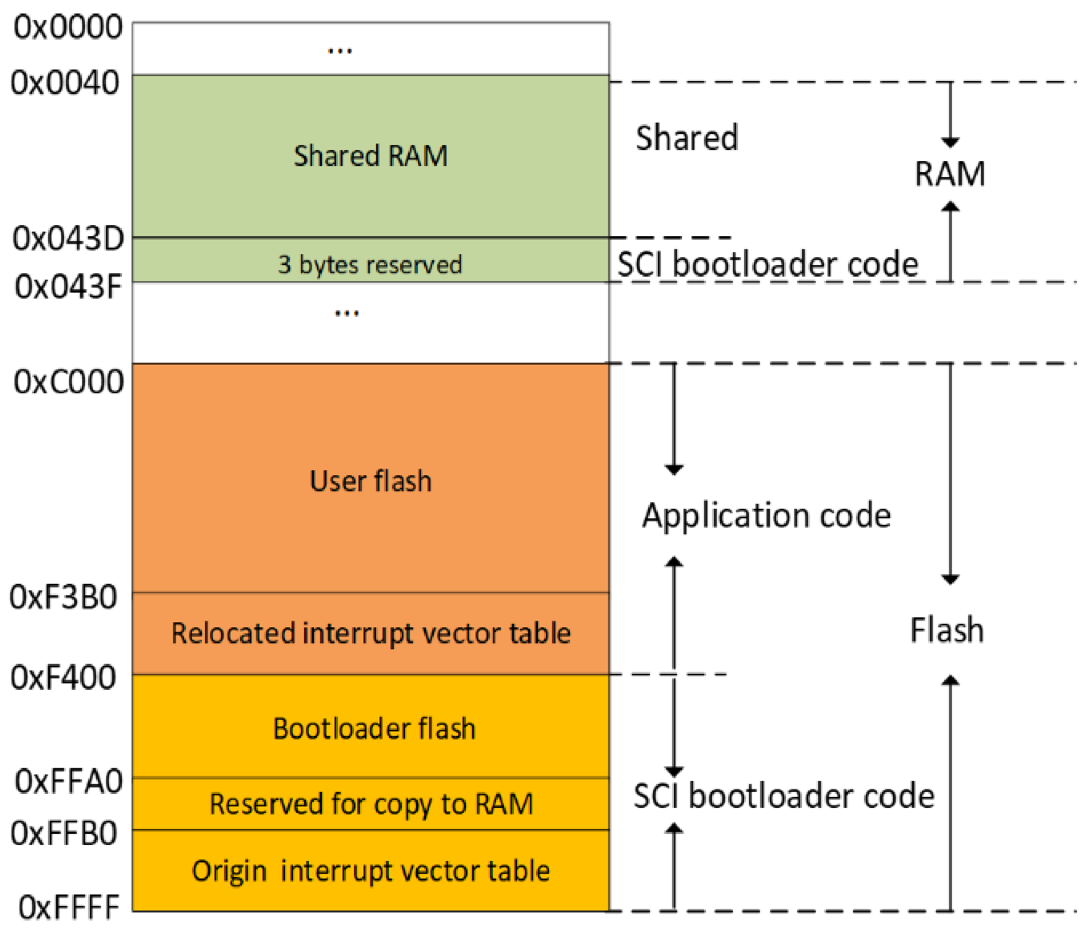


Figure 3. Memory allocation

2.3 Interrupt response process for application code

For the application code, *mtim.abs.s19* file, mtim0 generates an interrupt request every one second. The MCU interrupt response process is shown in Figure 4 after bootloader downloading of the *mtim.abs.s19* file to the MCU is successful.

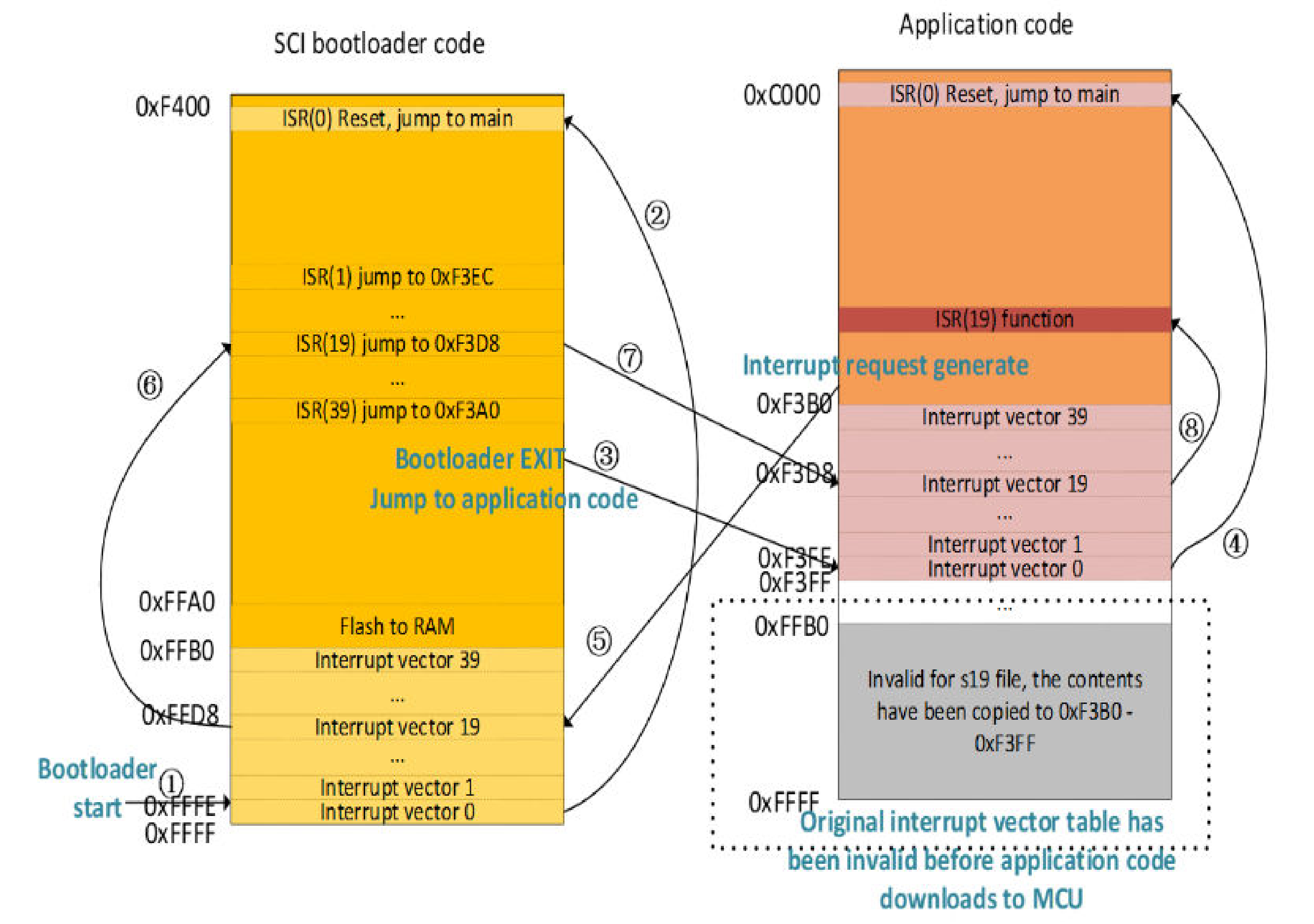


Figure 4. Interrupt response of application code

The process of application code interrupt response is summarized as follows:

Table 1. Summary of application code interrupt response process

1-2	When the MCU is reset, the program jumps to the reset vector (0xFFFE:0xFFFF), and the SCI bootloader code starts. According to the interrupt service routine (ISR) entry address stored in the reset interrupt vector 0, the reset ISR is executed.
2-3	The MCU enters main function of bootloader code from the reset ISR. The PC tool and MCU hook up is timeout, the bootloader process ends, and the program jumps to the application code.
3-4	The program jumps to the relocation reset vector (0xF3FE:0xF3FF) of the application code, then executes the reset ISR of the application code according to the ISR entry address stored in the new interrupt vector table.
4-5	The MCU enters main function of the application code from the reset ISR. At this time, the mtim0 module in the application code generates an interrupt request every one second.
5-6	When the mtim0 interrupt generates, the program jumps to the address which is loaded from mtim0 vector (0xFFD8:0xFFD9) and executes mtim0 ISR in the SCI bootloader code.

Table continues on the next page...

Table 1. Summary of application code interrupt response process (continued)

6-7	After executing the mtim0 ISR in the SCI bootloader code, the program counter points to the address loaded in the relocated mtim0 vector (0xF3D8: 0xF3D9).
7-8	The program executes mtim0 ISR in the application code according to the mtim0 ISR entry address stored in the relocated mtim0 vector (0xF3D8:0xF3D9).
8-power off	The mtim0 interrupt response completes in the application code. The application code continues to execute at this time. If the interrupt request generates again during the running of the program, the program repeats steps from 5 to 8.

3 SCI bootloader implementation

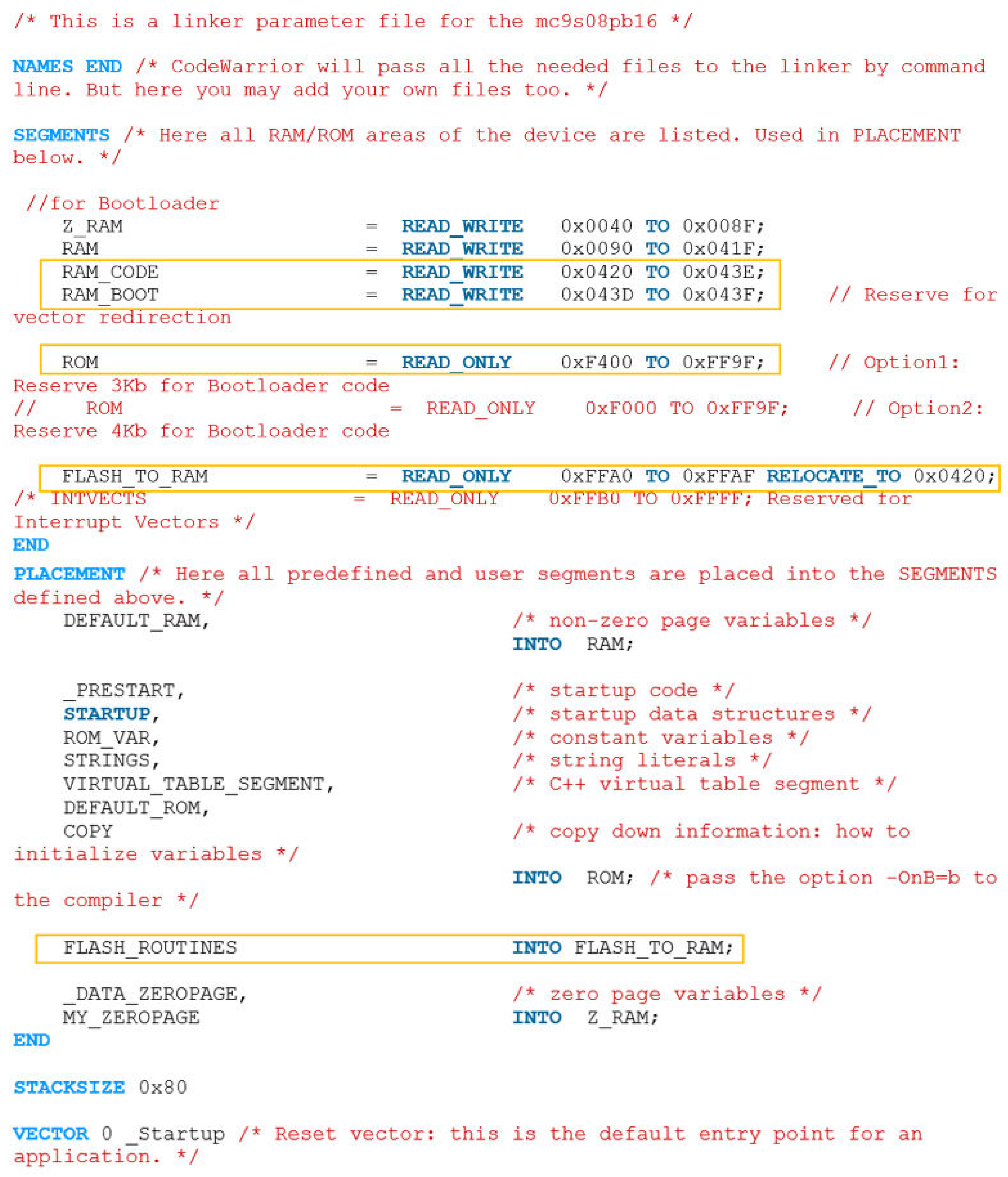
This section describes code implementation of the bootloader via SCI interface. The SCI bootloader code and application code (s19 file) need to reconfigure the linker file to redistribute memory space to prevent flash memory from being overwritten that might cause the bootloader to fail. In addition, the SCI bootloader code needs to include the FC protocol driver, FC_protocol.c/ FC_protocol.h, to implement the communication with the PC tool. The MCU information that the user needs to modify according to the requirements is defined in FC_protocol.h. The PC tool performs the corresponding operations according to the information obtained from MCU. Refer to the source code of PC tool for more details.

The following sections provide a brief introduction to the S08PB16 bootloader code and application code (mtim project) development.

3.1 Bootloader code

3.1.1 Example of prm file modification in bootloader code

The sample Project.prm file in bootloader code is shown in Figure 5 below.



```
/* This is a linker parameter file for the mc9s08pb16 */

NAMES END /* CodeWarrior will pass all the needed files to the linker by command
line. But here you may add your own files too. */

SEGMENTS /* Here all RAM/ROM areas of the device are listed. Used in PLACEMENT
below. */

 //for Bootloader
    Z_RAM                    =  READ_WRITE   0x0040 TO 0x008F;
    RAM                      =  READ_WRITE   0x0090 TO 0x041F;
    RAM_CODE                 =  READ_WRITE   0x0420 TO 0x043E;
    RAM_BOOT                 =  READ_WRITE   0x043D TO 0x043F;    // Reserve for
vector redirection

    ROM                      =  READ_ONLY    0xF400 TO 0xFF9F;    // Option1:
Reserve 3Kb for Bootloader code
//    ROM                      =  READ_ONLY    0xF000 TO 0xFF9F;    // Option2:
Reserve 4Kb for Bootloader code

    FLASH_TO_RAM             =  READ_ONLY    0xFFA0 TO 0xFFAF RELOCATE_TO 0x0420;
/* INTVECTS                =  READ_ONLY    0xFFB0 TO 0xFFFF; Reserved for
Interrupt Vectors */
END
PLACEMENT /* Here all predefined and user segments are placed into the SEGMENTS
defined above. */
    DEFAULT_RAM,                            /* non-zero page variables */
                                            INTO  RAM;

    _PRESTART,                              /* startup code */
    STARTUP,                                /* startup data structures */
    ROM_VAR,                                /* constant variables */
    STRINGS,                                /* string literals */
    VIRTUAL_TABLE_SEGMENT,                  /* C++ virtual table segment */
    DEFAULT_ROM,
    COPY                                    /* copy down information: how to
initialize variables */
                                            INTO  ROM; /* pass the option -OnB=b to
the compiler */

    FLASH_ROUTINES                          INTO FLASH_TO_RAM;

    _DATA_ZEROPAGE,                         /* zero page variables */
    MY_ZEROPAGE                             INTO  Z_RAM;
END

STACKSIZE 0x80

VECTOR 0 _Startup /* Reset vector: this is the default entry point for an
application. */
```

Figure 5. Project.prm in SCI bootloader code

As shown in Figure 5, the flash address 0xF400 - 0xFF9F is reserved to store the bootloader code, with RELOCATE_TO code in FLASH_ROUTINES that can be executed at a different address than it was allocated.

The code of flash launch command as shown in Figure 6 is programmed at 0xFFA0 - 0xFFAF address area (flash space), but references to functions in FLASH_ROUTINES use addresses in the 0x0420 - 0x042F area (RAM space).

```
#pragma CODE_SEG FLASH_ROUTINES
void RAM_Run_NVM_CMD(void)
{
        // Launch the command
        FTMRH_FSTAT |= 0x80;
        // Wait till command is completed
        while (!(FTMRH_FSTAT & FTMRH_FSTAT_CCIF_MASK));
}
#pragma CODE_SEG default
```

Figure 6. Flash launch command code in flash.c

The flash launch command code is copied to RAM by the *Flash_CopyInRAM()* function (Figure 7), so that the flash command code executes in the RAM space form 0x0420 and does not access flash when the flash is busy.

```
void  Flash_CopyInRAM(void)
{
  char *srcPtr, *dstPtr;
  uint16_t count;
  uint16_t sizeBytes = (uint16_t)Size_Copy_In_RAM;
  srcPtr = (char*)Start_Copy_In_RAM;
  dstPtr = (char*)&RAM_Run_NVM_CMD; // must be the start address of RAM_CODE
  for (count = 0; count < sizeBytes;  count++)
  {
    *dstPtr++ = *srcPtr++;
  }
}
```

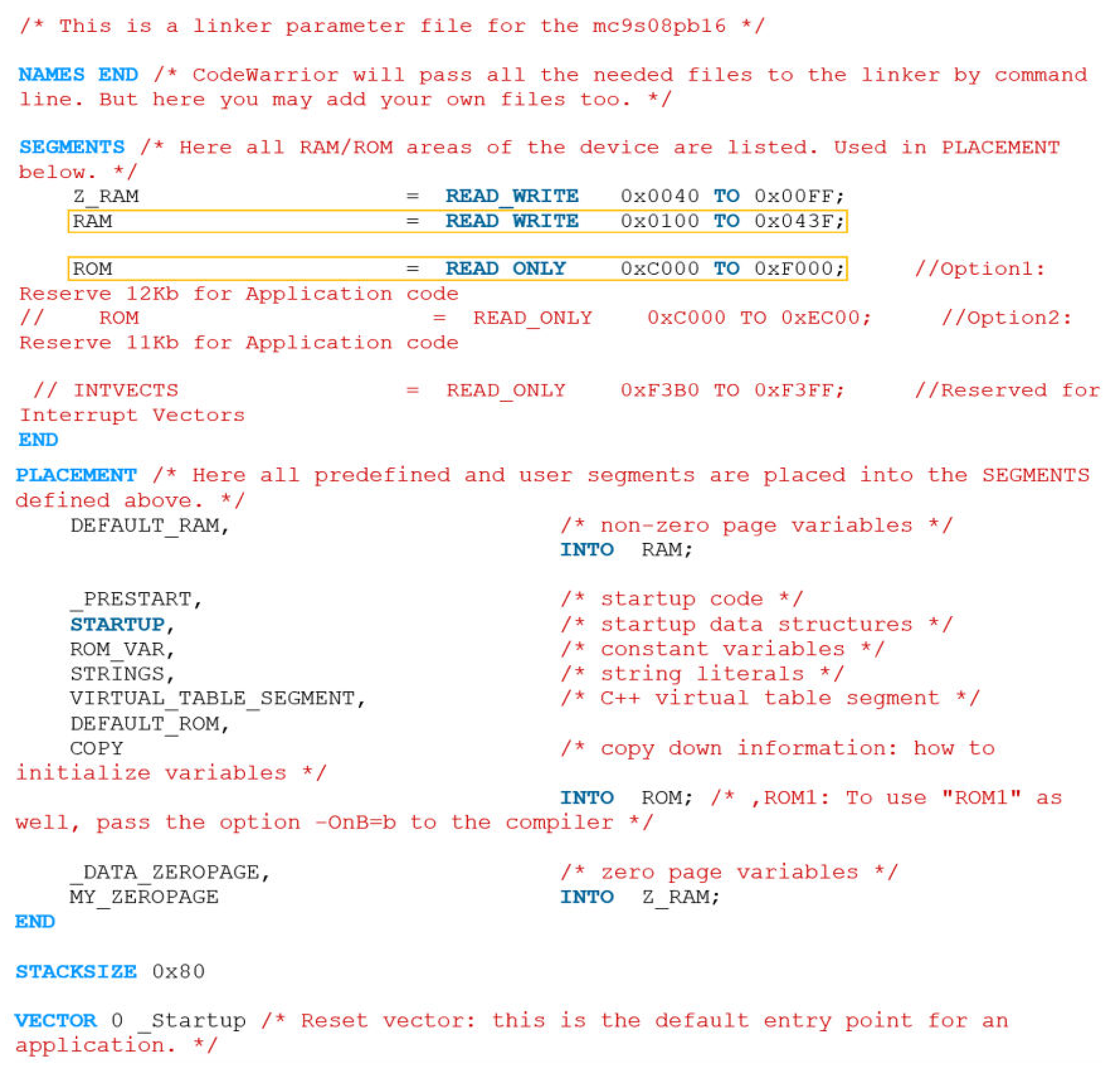
Figure 7. Flash_CopyInRAM() function in flash.c

3.2 Application code linker file

3.2.1 Example of prm file modification in application code

For the application code, the user only needs to modify the memory allocation in the linker file. The Project.prm file of the mtim project is shown in the following figure.

The available flash memory space 0xC000 - 0xF000 is reserved to store application code to avoid overwriting of memory space. The RAM space users can make appropriate assignments except for 3 bytes of RAM (0x43D - 0x43F) reserved for bootloader code.



```
/* This is a linker parameter file for the mc9s08pb16 */

NAMES END /* CodeWarrior will pass all the needed files to the linker by command
line. But here you may add your own files too. */

SEGMENTS /* Here all RAM/ROM areas of the device are listed. Used in PLACEMENT
below. */
    Z_RAM                    =  READ_WRITE   0x0040 TO 0x00FF;
    RAM                      =  READ_WRITE   0x0100 TO 0x043F;

    ROM                      =  READ_ONLY    0xC000 TO 0xF000;     //Option1:
Reserve 12Kb for Application code
//    ROM                      =  READ_ONLY    0xC000 TO 0xEC00;     //Option2:
Reserve 11Kb for Application code

 // INTVECTS                 =  READ_ONLY    0xF3B0 TO 0xF3FF;     //Reserved for
Interrupt Vectors
END

PLACEMENT /* Here all predefined and user segments are placed into the SEGMENTS
defined above. */
    DEFAULT_RAM,                        /* non-zero page variables */
                                        INTO  RAM;

    _PRESTART,                          /* startup code */
    STARTUP,                            /* startup data structures */
    ROM_VAR,                            /* constant variables */
    STRINGS,                            /* string literals */
    VIRTUAL_TABLE_SEGMENT,              /* C++ virtual table segment */
    DEFAULT_ROM,
    COPY                                /* copy down information: how to
initialize variables */
                                        INTO  ROM; /* ,ROM1: To use "ROM1" as
well, pass the option -OnB=b to the compiler */

    _DATA_ZEROPAGE,                     /* zero page variables */
    MY_ZEROPAGE                         INTO  Z_RAM;
END

STACKSIZE 0x80

VECTOR 0 _Startup /* Reset vector: this is the default entry point for an
application. */
```

Figure 8. Linker file configuration for application code

3.2.2 S-record file

After the Project.prm file memory allocation completes, rebuild the application code. The *mtim.abs.s19* file in the mtim project is available in the FLASH folder of the project, as shown in the following figure. The *mtim.abs.s19* file is downloaded to MCU through the PC tool.

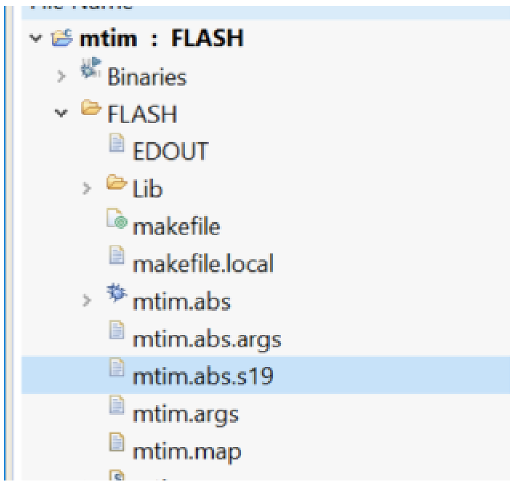
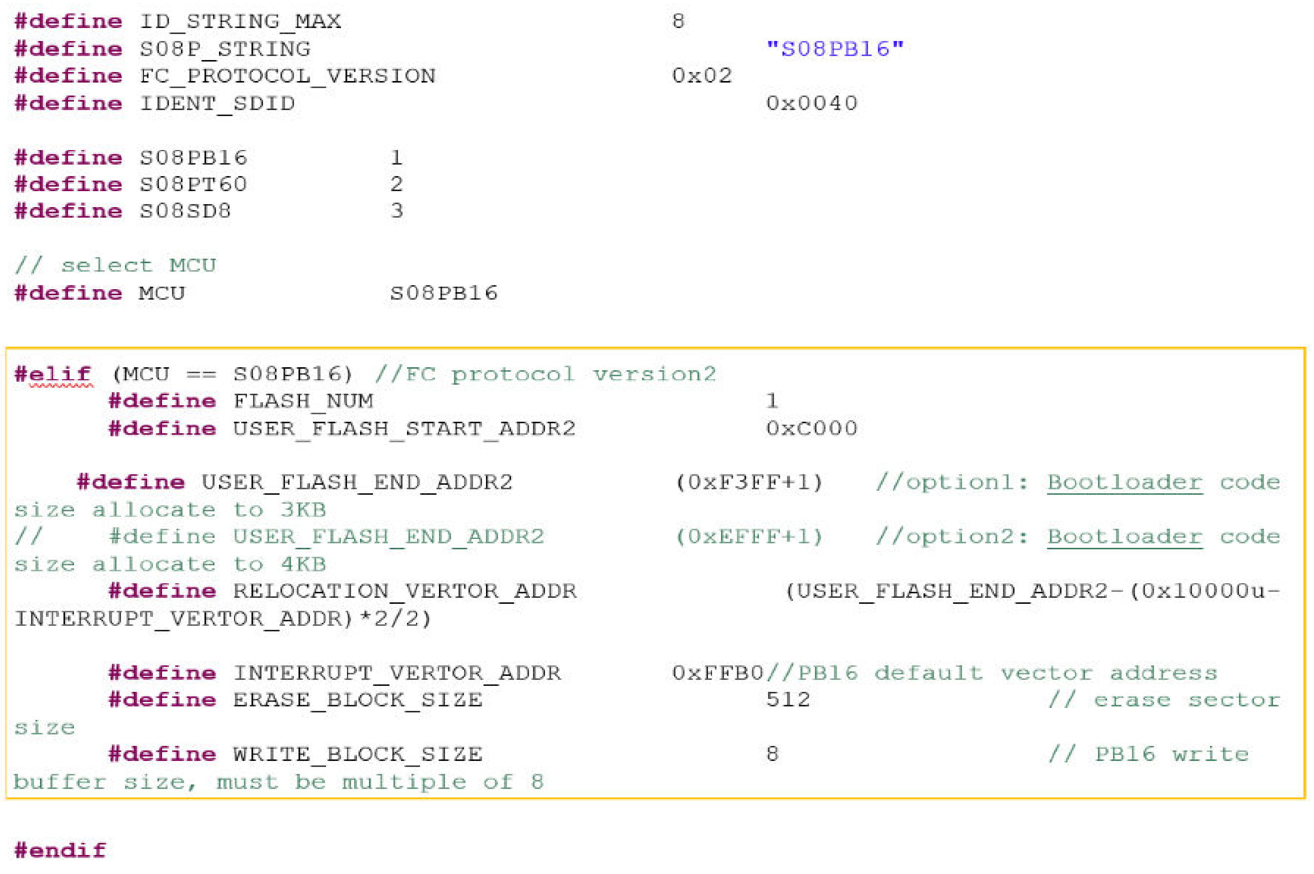


Figure 9. s19 file in application project

3.2.1 FC protocol driver

When using the PC tool to implement the MCU boot, the FC protocol driver (FC_protocol.c/FC_protocol.h) is required to include in the bootloader code. As shown in Figure 8, the required information for the MCU bootloader is predefined in FC_protocol.h. The pre-defined part of the content, such as the value of ‘FC_PROTOCOL_VERSION’, selects FC protocol version 2 to support HCS08 bootloader implement. The start address of relocated interrupt vector table in s19 file is defined as ‘RELOCATION_VERTOR_ADDR’.

When designing the bootloader code, the user needs to correctly define the MCU's information in FC_protocol.h, so that the MCU sends the information to the PC tool through the SCI interface, and the PC tool receives the correct response.



```
#define ID_STRING_MAX                    8
#define S08P_STRING                            "S08PB16"
#define FC_PROTOCOL_VERSION              0x02
#define IDENT_SDID                             0x0040

#define S08PB16          1
#define S08PT60          2
#define S08SD8           3

// select MCU
#define MCU              S08PB16

#elif (MCU == S08PB16) //FC protocol version2
      #define FLASH_NUM                        1
      #define USER_FLASH_START_ADDR2           0xC000

    #define USER_FLASH_END_ADDR2          (0xF3FF+1)    //option1: Bootloader code
size allocate to 3KB
//    #define USER_FLASH_END_ADDR2        (0xEFFF+1)    //option2: Bootloader code
size allocate to 4KB
      #define RELOCATION_VERTOR_ADDR             (USER_FLASH_END_ADDR2-(0x10000u-
INTERRUPT_VERTOR_ADDR)*2/2)

      #define INTERRUPT_VERTOR_ADDR        0xFFB0//PB16 default vector address
      #define ERASE_BLOCK_SIZE                 512              // erase sector
size
      #define WRITE_BLOCK_SIZE                 8                // PB16 write
buffer size, must be multiple of 8

#endif
```

Figure 10. FC_protocol.h configuration for SCI bootloader code

4 Conclusion

This application note explains how to implement the bootloader for S08PB16 via SCI interface by using the PC tool (see AN2295 application note to know details about PC tool). The bootloader is more convenient for the user to update the application code without using other programming tools, such as P&E Multilink. The bootloader utility (AN2295SW) and the S08PB16 bootloader code can be downloaded from www.nxp.com.

5 References

Following references are available on NXP website:

- Developer's Serial Bootloader (AN2295) - https://www.nxp.com/docs/en/application-note/AN2295.pdf
- Serial Bootloader for 56F82xx (AN2745) - https://www.nxp.com/docs/en/application-note/AN2745.pdf
- UART Boot Loader Design on Kinetis E Series (AN4767) - https://www.nxp.com/docs/en/application-note/AN4767.pdf
- HC(S)08 and HC(S)12 Build Tools Utilities Manual - https://www.nxp.com/docs/en/reference-manual/HCS-RS08-Build_Tools_Utilities.pdf
- AN2295SW

Date of release: May 2020
Document identifier: AN12871