The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN205379

FR, MB91F460, ACCEMIC MDE 2008 and Boot ROM Type II

The goal of this application note is to show how to integrate ACCEMIC MDE 2008 using Boot ROM type 2. In order to reach this aim, it is explained in detail how the Boot ROM type 2 supports BDM (Background Debug Mode) and the configuration requirements of ACCEMIC MDE 2008 to be integrated into an application

Contents

1 Introduction....1
2 Features of Boot ROM Type 2 and ACCEMIC MDE 2008 1
2.1 Description of Boot ROM Type 2 1
2.2 Description of ACCEMIC MDE 2008....6
3 Integration of ACCEMIC MDE 2008 to an Application 6
3.1 General Configuration Requirements for Integration of ACCEMIC MDE 2008 into an Application 6
3.2 Interrupt Vectors Configuration Requirements for Integration of ACCEMIC MDE 2008 into an Application 7
4 Adaptation of a template to use ACCEMIC MDE 2008 8
4.1 Renaming of the Cypress Template Project....8
4.2 Necessary source files, libraries and header files 8
4.3 Necessary configurations in SOFTUNE Workbench....8
4.4 Serial Flashing and Basic Configuration of ACCEMIC MDE 2008 14
5 Using ACCEMIC MDE 2008 on demand....20
5.1 Preparing Application for later ACCEMIC MDE 2008 usage 20
5.2 Preparing ACCEMIC MDE2008 project....20
5.3 Example preparation 20
6 Related Documents and Contact....28
6.3 Manuals, Application Notes and Customer Information 28
6.4 Abbreviations 28
6.5 Contact to Cypress 28
6.6 Contact to Accemic 28
Document History....29

1 Introduction

The goal of this application note is to show how to integrate ACCEMIC MDE 2008 using Boot ROM type 2.

In order to reach this aim, it is explained in detail how the Boot ROM type 2 supports BDM (Background Debug Mode) and the configuration requirements of ACCEMIC MDE 2008 to be integrated into an application.

Finally, it is shown how to adapt a template supplied by Cypress Semiconductor GmbH (from now on just Cypress) to use ACCEMIC MDE 2008.

2 Features of Boot ROM Type 2 and ACCEMIC MDE 2008

Interaction of boot ROM Type 2 and ACCEMIC MDE 2008

2.1 Description of Boot ROM Type 2

Starting from the following devices, all flash devices designed with FME's boot ROM support boot ROM type 2:

- MB91F467DA – ES Revision 4
- MB91F469GA – ES Revision 3
- MB91F464AA – ES Revision 3
- MB91F465KA – ES Revision 3

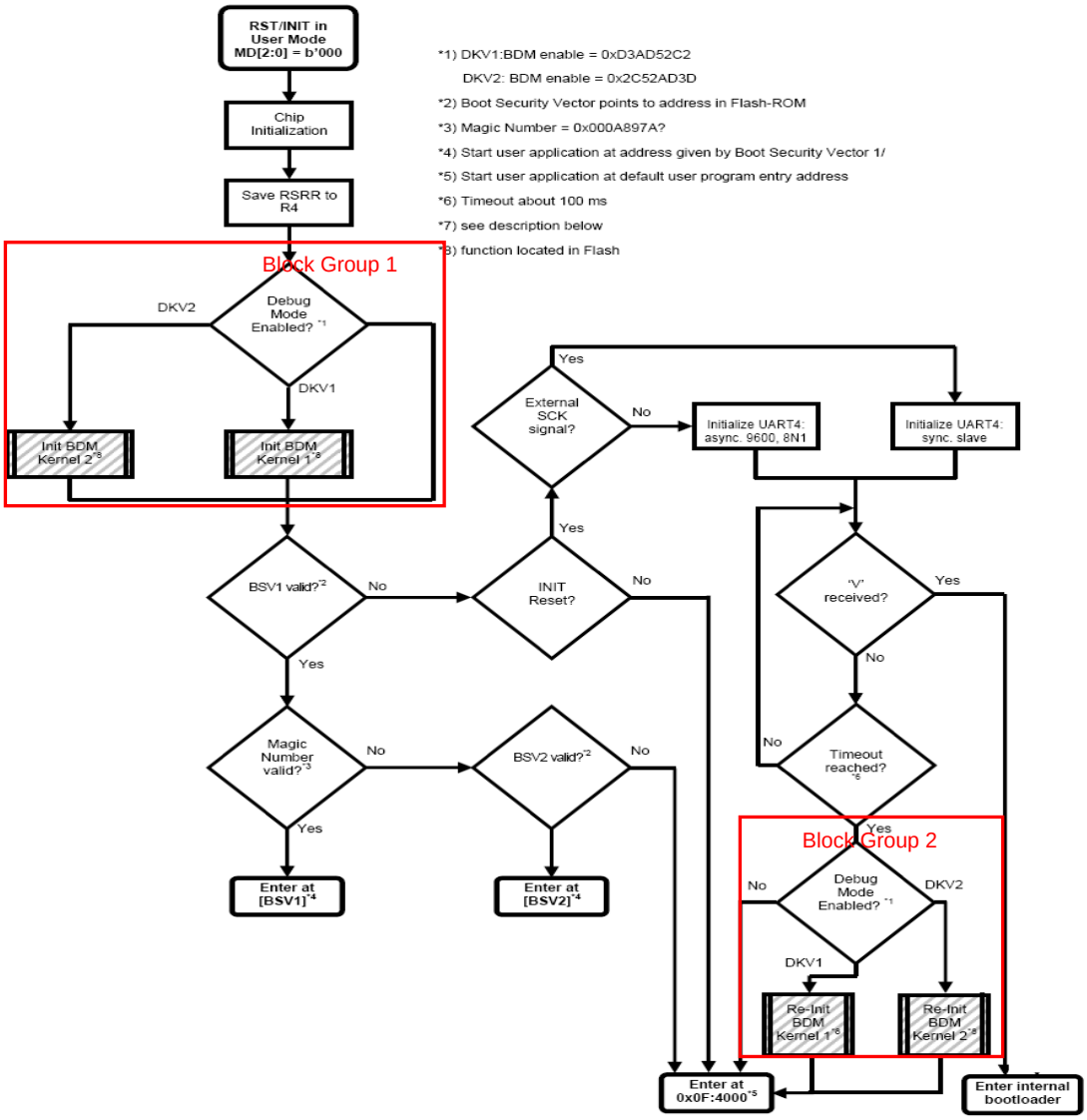
- MB91F467BA – ES Revision 2
- MB91F465DA
- MB91F465XA
- MB91F467CA
- MB91F467MA
- MB91F465BA.

The microcontroller MB91F467CA is selected from this list for this application sample.

The Boot ROM type 2 supports BDM (Background Debug Mode).

The flow chart of the Boot ROM type 2 is shown below.

Figure 1. Flow Chart of Boot ROM Type 2



In comparison with the Boot ROM Type 1, the Boot ROM type 2 has additionally the Block Group 1 and 2. These block groups are specifically intended for supporting Background Debug Mode (BDM).

An initialization function (generally called Init_Debug ()) is located in the BDM memory area. The debugger is initialized by this function.

In order to configure the program flow through the Boot ROM, the DCB (Debug Configuration Block) is implemented.

The Debug Configuration Block is a fixed 16kB area in the main Flash ROM which enables the user to configure the behavior of the debug function.

Below the DCB is shown.

Figure 2. Memory Area of Debug Configuration Block (DCB) for MB91F46xx

14:8020h		BDME *(DKV1: 0xD3AD52C2 DKV2: 0x2C52AD3D None: all other)*	Background Debug Mode Enable	erased (disabled)
14:8018h		DKV2	Debug Kernel Vector 2	erased (not set)
14:8010h		DKV1	Debug Kernel Vector 1	erased (not set)
14:8008h	FSV2	BSV2	Flash / Boot Security Vector 2	
14:8000h	FSV1	BSV1	Flash / Boot Security Vector 1	

In order to obtain the program flow shown in Figure 3 and Figure 4:

- The DKV1 vector is selected. Therefore, the BDME content has to be equal to 0xD3AD52C2.
- The DKV1 vector contains the first address of the memory area where the init_debug() function is located. According to the DCB, a possible address is 0x00148100.

Figure 3. Block Group 1

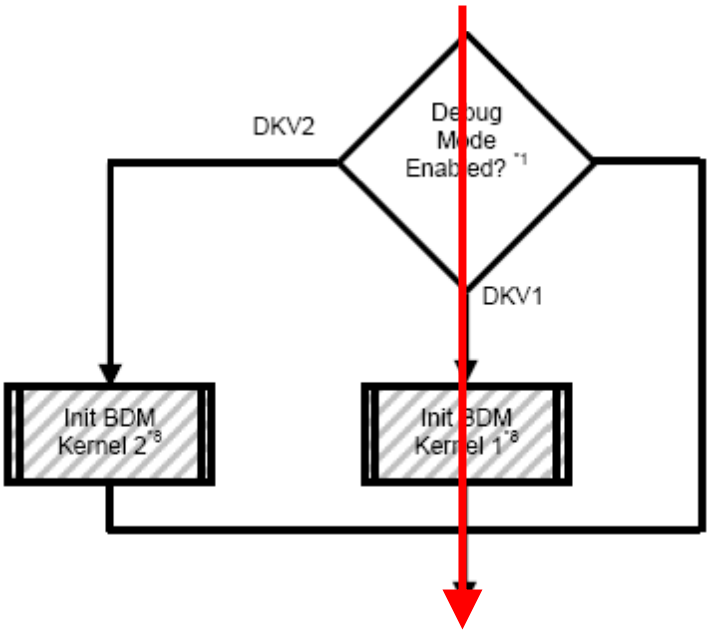
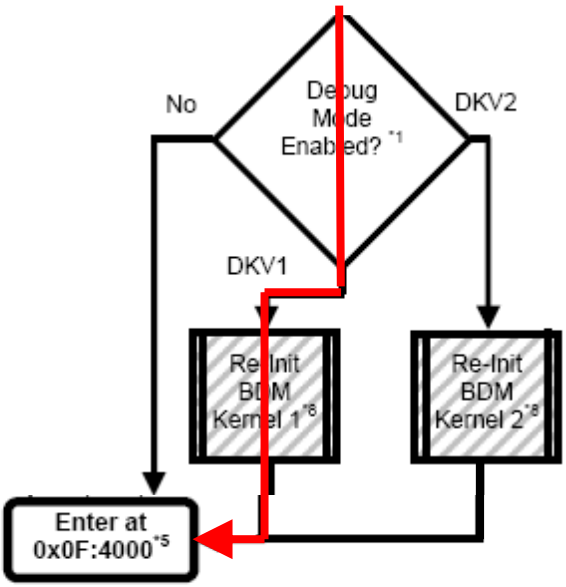


Figure 4. Block Group 2



In Figure 3 and Figure 4 it is possible to see, that the program flow takes place through the BDM kernel. Once this code has been executed, the program returns back to the Boot ROM program flow.

In ACCEMIC MDE 2008, the `init_debug()` function is called `acc_InitKernel()`.

In the *acc_wrapper.c* file, the `acc_InitKernel()` function is called from the code located in the memory area which first address is pointed by DKV1. Therefore, the function `accemic_init_wrapper()` (instead of `acc_InitKernel()`) is located at the first address of the BDM memory area after the stack initialization. This is due to using this function, it is possible to perform any actions before the `init_debug()` function is called.

In this extra function the PORT26 is configured in order to control the program flow through the `accemic_init_wrapper()` function. Therefore, depending on the input value of the PORT26, the `acc_InitKernel()` function is called or not. The PORT 27 is used to show the resulting program flow.

In order to fulfil what is mentioned above, the *acc_wrapper.c* source file is implemented. Below the *acc_wrapper.c* code is shown.

Figure 5. acc_wrapper.c source code

```
#include "mb91467c.h"
 extern void acc_InitKernel(void);
 void accemic_init_wrapper(void);
 #pragma section CODE=WRAPPERCODE,attr=CODE,locate=0x148100
 #pragma asm
         .IMPORT _userstack_top
         .IMPORT _systemstack_top
         ; setup stack
         ORCCR    #0x20
         LDI      #__userstack_top, SP    ; initialize SP
         ANDCCR   #0xDF
         LDI      #__systemstack_top, SP  ; initialize SP
         ; save return pointer
         ST       RP,@-R15
                  ; save register R12
         STM1     (R12)
                  ; call debugger init wrapper
         LDI:32   #_accemic_init_wrapper, R12
         CALL     @R12
         ; restore register R12
         LDM1     (R12)
         ; restore return pointer
         LD       @R15+,RP
         RET
 #pragma endasm
 void accemic_init_wrapper()
 {
     PORTEN = 0x3;
     // Port 27: Debug Port
     DDR27 = 0xFF;
     // Port 26: MOD Port (Bits: 0, 1 and 2)
     DDR26 = 0x00;
     // MOD Patern = 5;
     if ((PDR26 & 0x07) == 0x05) {
               acc_InitKernel();
               PDR27 = 0x05;
      } else PDR27 = 0x01;
     return;
 }
 #pragma asm
        .section        BDME_VECTOR, code, locate = 0x148024
        .data.w 0xD3AD52C2    ; DKV1
        ;.data.w 0x434AB5CB   ; DKV2
           .section        DKV1_VECTOR, code, locate = 0x148014
        .data.w 0x00148100    ; WRAPPERCODE, ACC_LIB_CODE, ACC_ LIB_CONST
        .section        DKV2_VECTOR, code, locate = 0x14801C
              .data.w 0xFFFFFFFF    ; not used
 #pragma endasm
```

Note: In this example, the `WRAPPERCODE`, `ACC_LIB_CODE` and `ACC_LIB_CONST` are placed in the address range from `0x148100` to `0x14BFFF`.

`ACC_LIB_CODE` is the memory area where the code of the ACCEMIC MDE 2008 kernel debug is located.

`ACC_LIB_CONST` is the memory area where the constants of the ACCEMIC MDE 2008 kernel debug are located

2.2 Description of ACCEMIC MDE 2008

The following BDM functionalities are supported by ACCEMIC MDE 2008:

- Debug kernel
- Boot/flash loader
- Communication support package for UART0/4

These functionalities are implemented in the following files and libraries:

- MDE_MB91V460.c
- MonitorCom_MB91V460_U0.lib
- MonitorCom_MB91V460_U4.lib
- MonitorCore.lib

The MDE_MB91V460.c source file contains functions related to the communication and kernel configuration.

ACCEMIC MDE 2008 offers the possibility of using either the UART4 or the UART0. In the present application sample, the UART4 is selected as the communication interface. Therefore, only the MonitorCom_MB91V460_U4.lib library will be used.

MonitorCore.lib is the debugger core itself.

3 Integration of ACCEMIC MDE 2008 to an Application

How to integrate ACCEMIC MDE 2008 to an own application

3.1 General Configuration Requirements for Integration of ACCEMIC MDE 2008 into an Application

The requirements to integrate ACCEMIC MDE 2008 with UART0/4 as debug communication interface to the project are listed below:

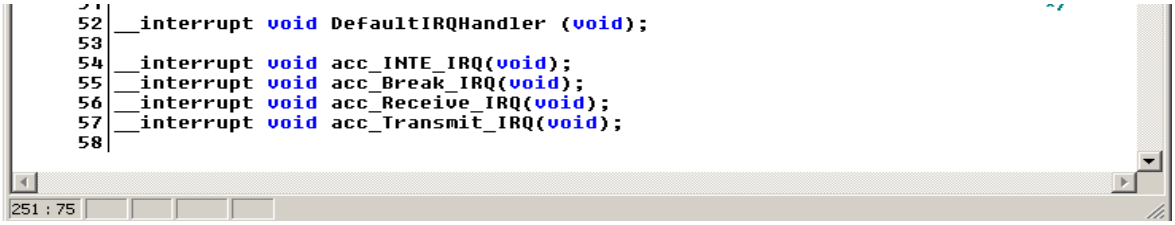
- Addition of the following files supplied by ACCEMIC MDE 2008 to the application
  - MDE_MB91V460.c
  - Either MonitorCom_MB91V460_U4.lib or MonitorCom_MB91V460_U0.lib
  - MonitorCore.lib
- Enable of the BDM writing the number of the corresponding vector into BDME (`0x00148028`):
  - 0xD3AD52C2 for DKV1
  - 0x2C52AD3D for DKV2
- Configuration of the selected BDM vector to point to the `acc_InitKernel` function. This procedure was illustrated in the previous section using the *wrapper.c* file.
- Modification of the *vector.c* file for including the interrupts that the ACCEMIC BDM kernel requires to communicate using the debug communication interface according to the description in the next subsection.
- Reservation of memory areas for the ACCEMIC BDM kernel in the desired memory area range pointed by the selected vector for:
  - Code
  - Const
  - Data
  - Stack

3.2 Interrupt Vectors Configuration Requirements for Integration of ACCEMIC MDE 2008 into an Application

ACCEMIC MDE 2008 uses the selected UART to communicate to the debug interface using interrupts. Additionally, the debugging functionalities are also supported by using interrupts. Consequently the *vectors.c* source file has to be adapted to this.

Therefore it is necessary to exchange the *vectors.c* file supplied by CYPRESS template projects for the *vectors.c* file supplied by ACCEMIC MDE 2008.

Figure 6. vectors.c file supplied by ACCEMIC MDE 2008



Relevant code lines of the *vectors.c* file supplied by ACCEMIC MDE 2008 are shown below.

Figure 7. Excerpt of vectors.c file

```
. . .

#pragma intvect acc_Break_IRQ    10   /* Instruction break exception
   */
#pragma intvect acc_Break_IRQ    11   /* Operand break trap
   */

. . .

#pragma intvect acc_Receive_IRQ  66   /* USART (LIN, FIFO) 4 RX
*/
#pragma intvect acc_Transmit_IRQ 67   /* USART (LIN, FIFO) 4 TX
*/

. . .
```

Note that it is necessary to define globally the macros `MDE_DEBBUGER` and `UART4`.

Also note: The *cpu.h* file is also supplied by ACCEMIC MDE 2008. But this file does not required extra explanation because it can be easily adapted to every application.

Normally the starting point of every application is the Softune's template project provided by CYPRESS. Therefore, in the following section, it is shown how to adapt a template project to use ACCEMIC MDE 2008.

4 Adaptation of a template to use ACCEMIC MDE 2008

Example of integration of ACCEMIC MDE 2008 to an application

In order to illustrate the integration process of ACCEMIC the adaptation of a template project to use ACCEMIC MDE 2008 is shown in this subsection step by step.

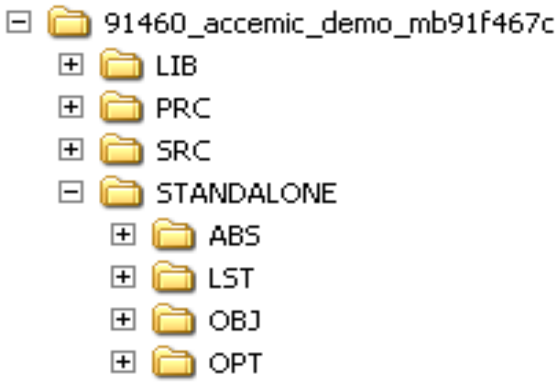
The starting point is the *91460_template_91467c* template project supplied by CYPRESS.

4.1 Renaming of the CYPRESS's Template Project

- Select the CYPRESS's template: *91460_template_91467c*
- Rename folder
  91460_template_91467c-vxx >> 91460_accemic_demo_mb91f467c
- Rename files
  91460_template_91467c.wsp >> 91460_accemic_demo_mb91f467c.wsp
  91460_template_91467c.prj >> 91460_accemic_demo_mb91f467c.prj
  91460_template_91467c.dat >> 91460_accemic_demo_mb91f467c.dat

4.2 Necessary source files, libraries and header files

Figure 8. Template Folders



- Put into the *SRC* folder the following file:
  - acc_wrapper.c
- Put into the *SRC* folder the followings ACCEMIC's files:
  - MDE_MB91V460.c
  - cpu.h
- Exchange the *vectors.c* file supplied by Cypress for ACCEMIC's *vectors.c* file
- Create a new folder: *LIB*
- Save in the *LIB* folder the followings ACCEMIC's libraries:
  - MonitorCom_MB91V460_U4.lib
  - MonitorCore.lib

4.3 Necessary configurations in SOFTUNE Workbench

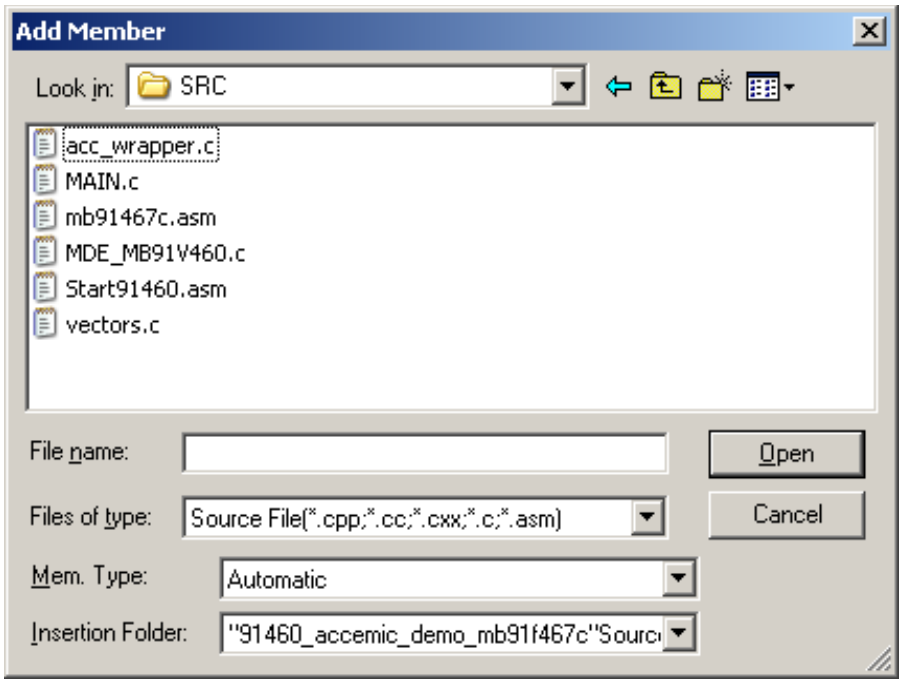
- Open SOFTUNE Workbench
- Open *the 91460_accemic_demo_mb91f467c.wsp* workspace
- Add members from the *SRC* folder as follows:
  Source Files >> Add Member to folder >> file

Select:

acc_wrapper.c

MDE_MB91V460.c

Figure 9. Member Addition



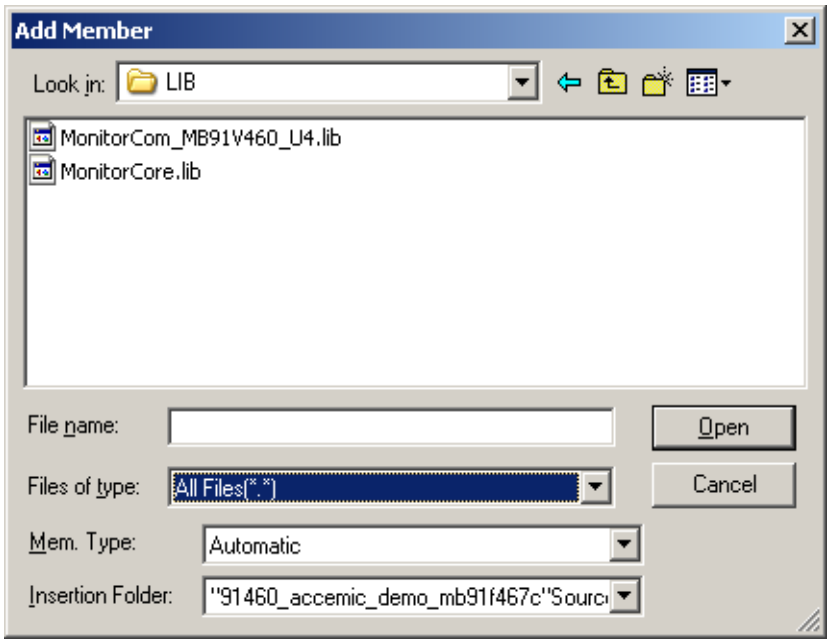
- Add member from the *LIB* folder as follows:
  Source Files >> Add Member to folder >> file

  Select from All Files [*.*]

  MonitorCom_MB91V460_U4.lib

  MonitorCore.lib

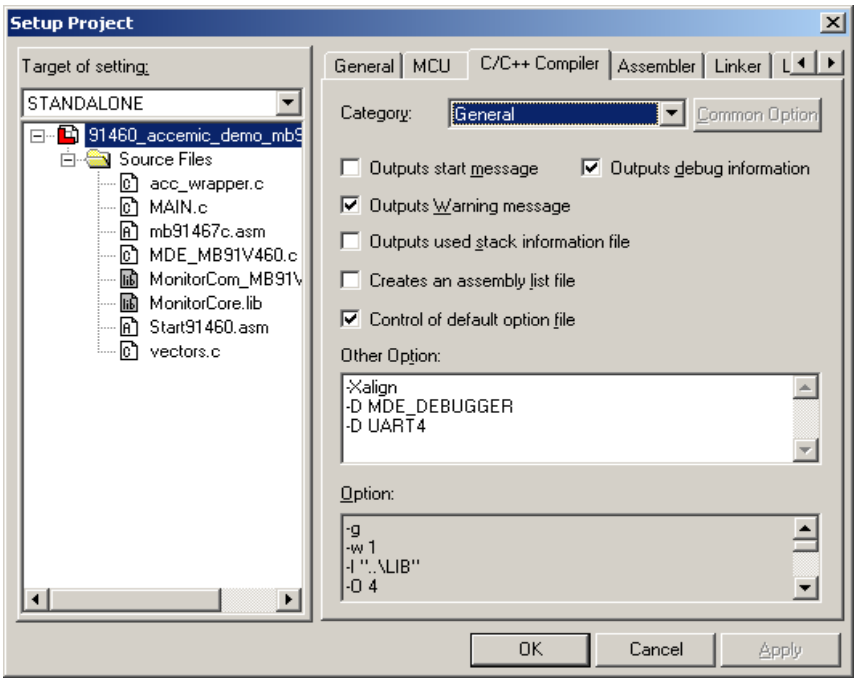
Figure 10. Member Addition



- Define macros: `MDE_DEBUGGER` and `UART4`
  Project >> Setup Project >> C/C++ Compiler >>

  −D MDE_DEBUGGER
  - D UART4

Figure 11. Setup Project (Define Macros)



- Setup memory areas, as follows:
  Project >> Setup Project >> Linker >>

Figure 12. Setup Project (Memory Areas)

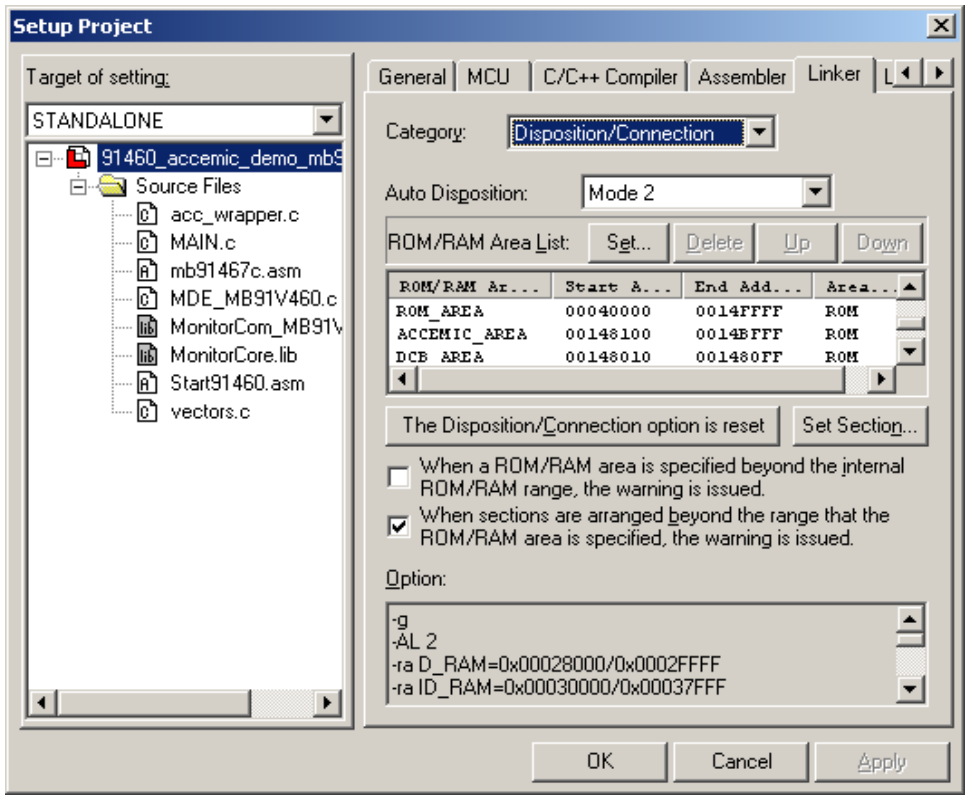


Figure 13. Setup Project (Memory Area)

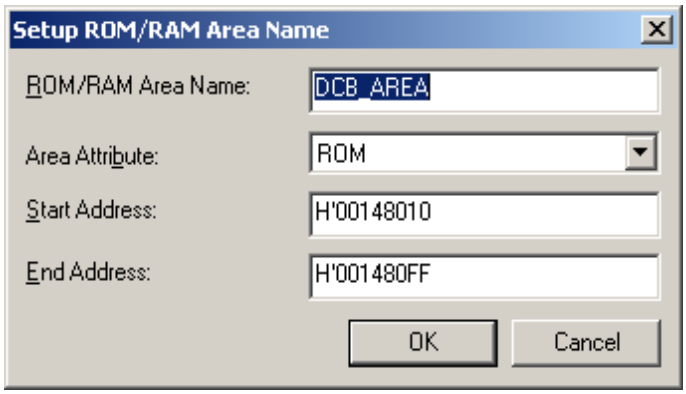


Figure 14. Setup ROM (ACCEMIC AREA

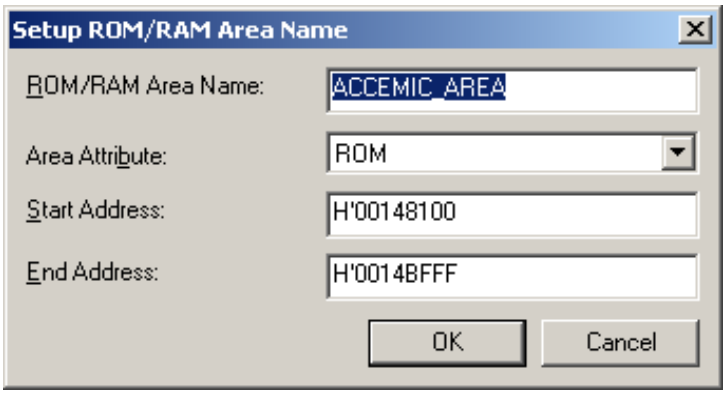
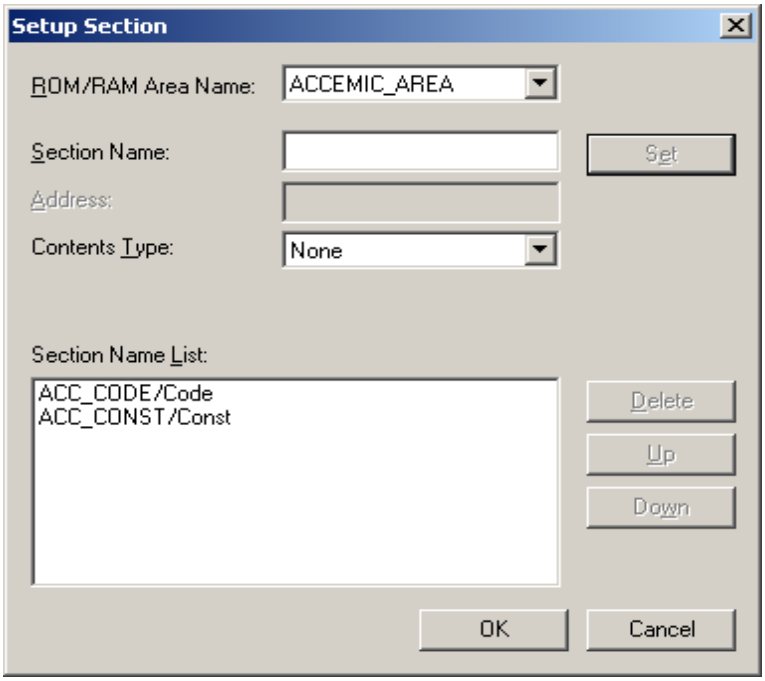


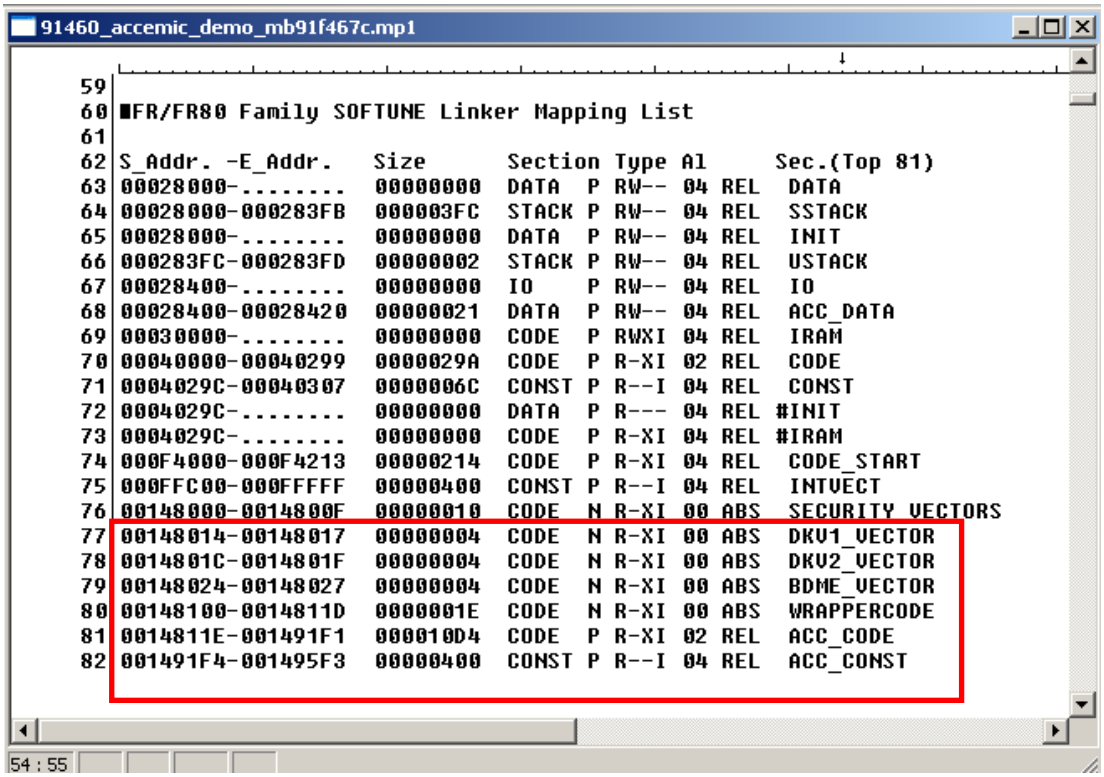
Figure 15. Setup Section (ACCEMIC CODE and CONST)



- Click “build all source files” 

The generated linker mapping list is shown below.

Figure 16. Linker Mapping List

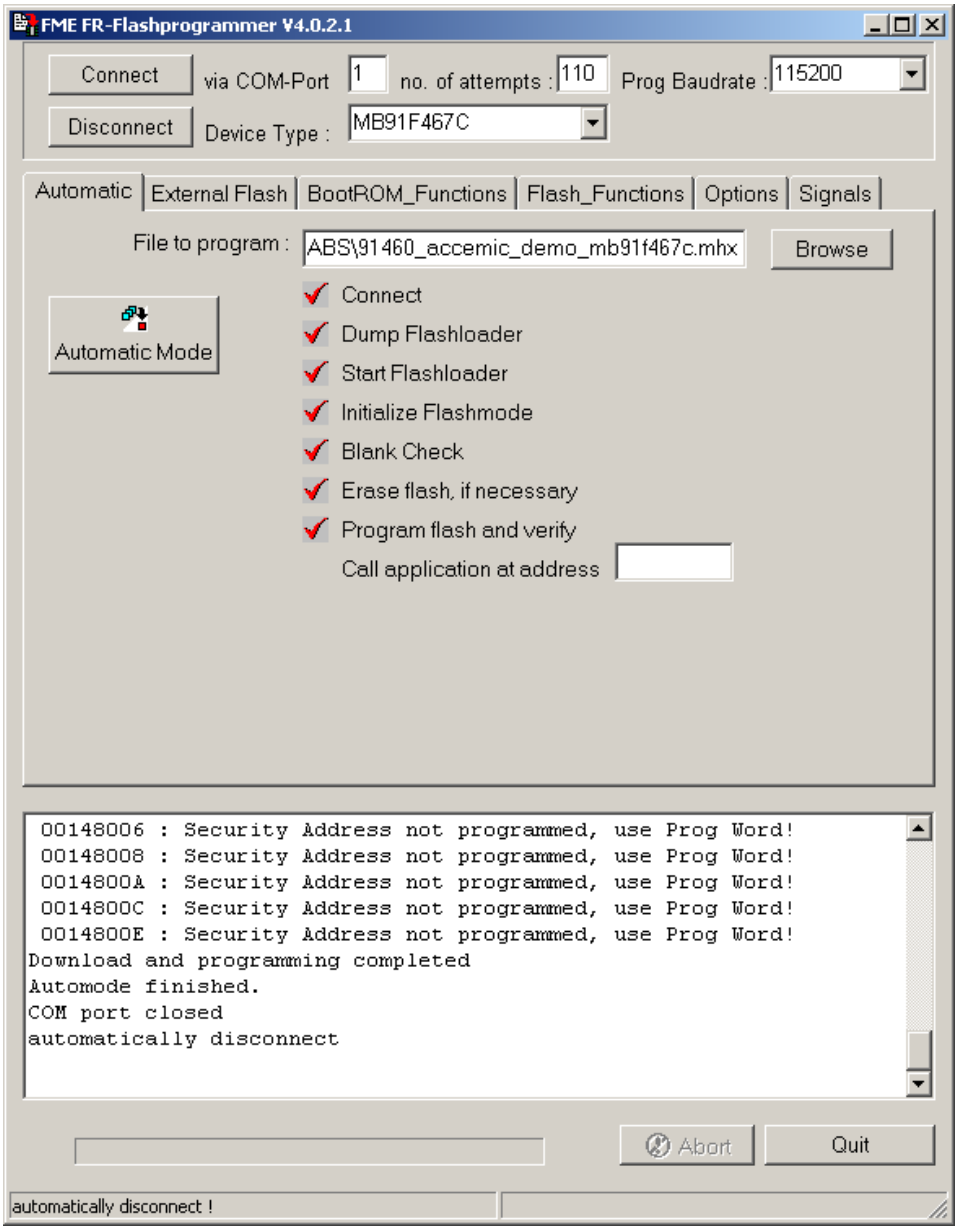


In this linker mapping list it is possible to observe where is located the wrapper code (`WRAPPERCODE`), the ACCEMIC's data, code and const (`ACC_DATA`, `ACC_CODE` and `ACC_CONST`), the vectors (`BDME`, `DVK1` and `DVK2`), etc.

4.4 Serial Flashing and Basic Configuration of ACCEMIC MDE 2008

- Flash the microcontroller using the Serial Flash Programmer:

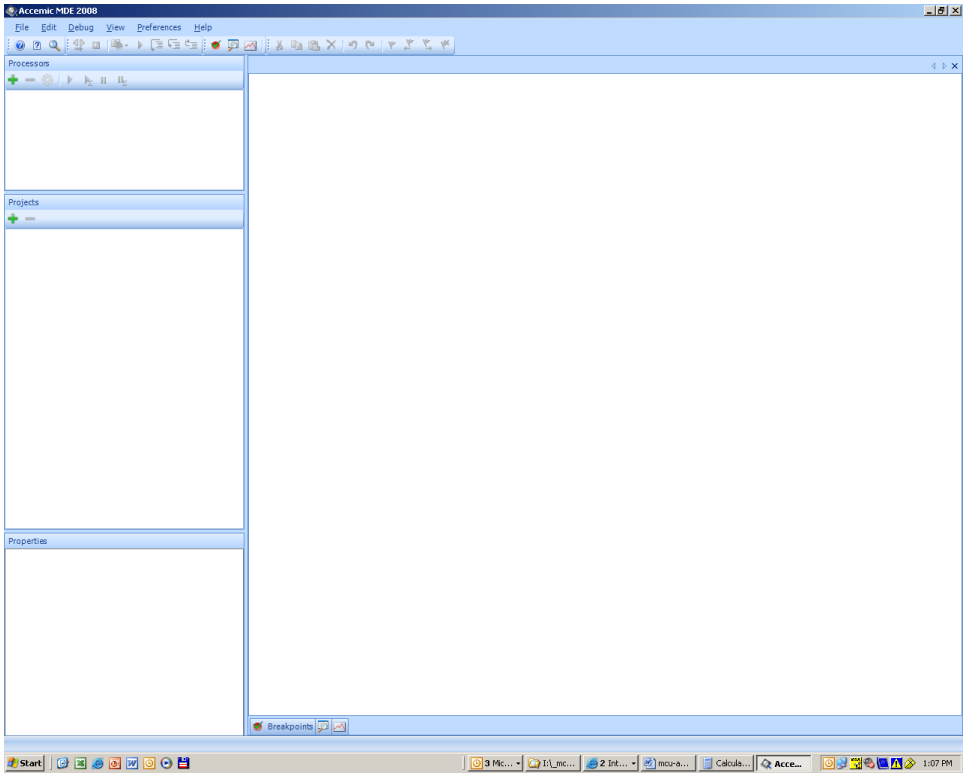
Figure 17. Flash Serial Programmer



Note: it is also possible to flash the MCU using “download application” functionality of ACCEMIC MDE 2008.

- ACCEMIC MDE 2008
  - Copy the license file into the corresponding ACCEMIC's folder and connect the key-dongle to an USB connector
  - Open ACCEMIC MDE 2008

Figure 18. ACCEMIC MDE 2008



  - Create a new workspace (*.awf* file extension)

Figure 19. Creation of a New Workspace

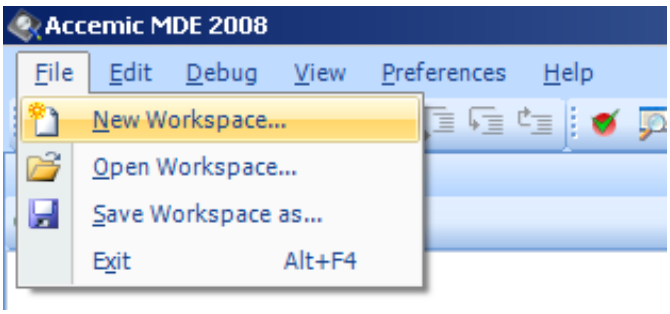
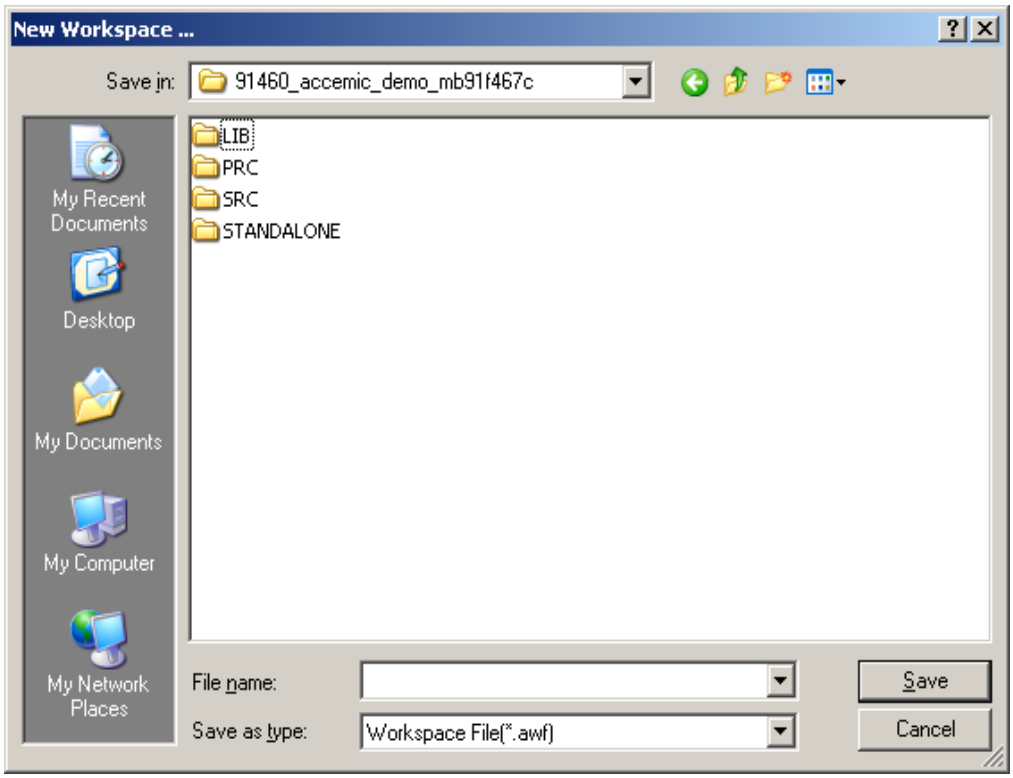
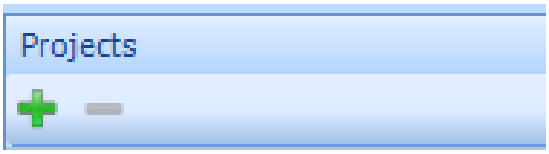


Figure 20. New Workspace…



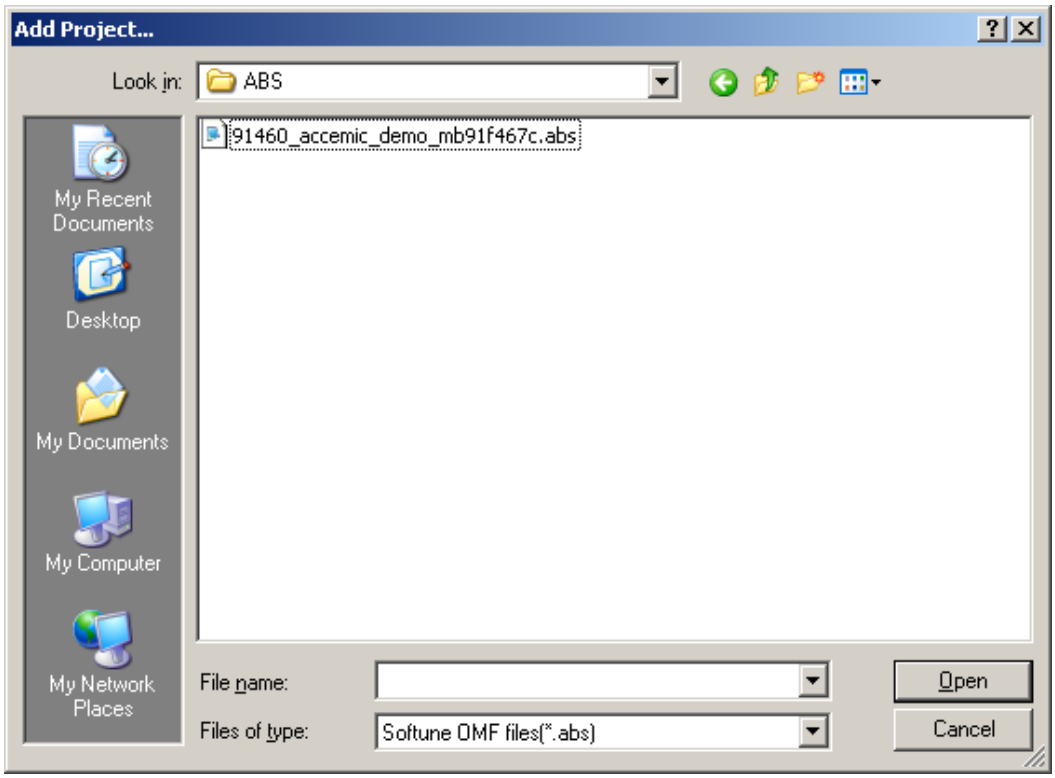
- Add Project: 91460_accemic_demo_mb91f467c.abs
  a) In order to add an existing project, press on the green icon

Figure 21. Projects Window



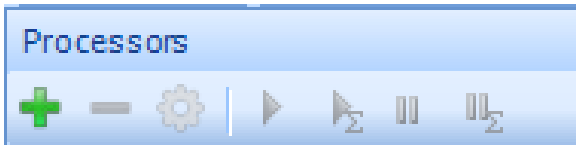
- Open the desired project file (*.abs* file extension)

Figure 22. Add Project…



- Add the corresponding processor (in our example: MB91F467CA) and configure the serial port (COM) for the communication
  a) In order to select the processor and to configure the serial port (COM), press on the green icon

Figure 23. Processors Windows



- After clicking on the green icon, the device can be selected and the COM can be configured.

Figure 24. Processor Setup…

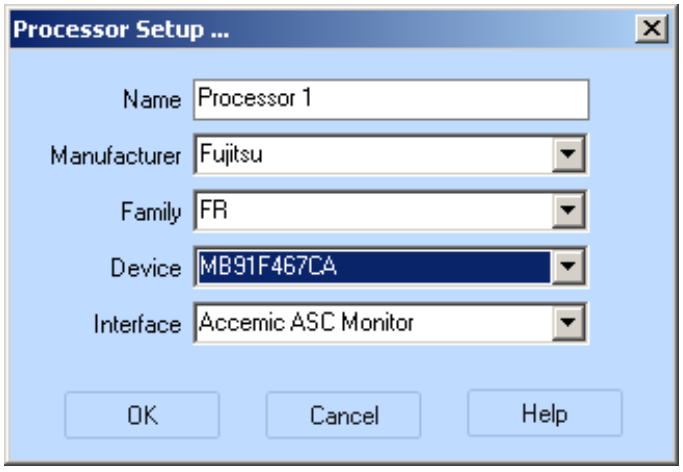
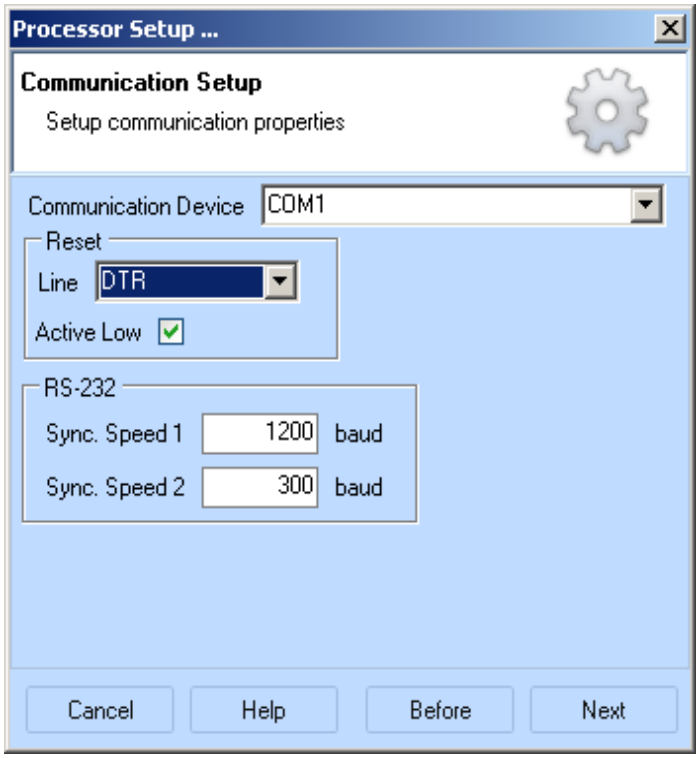
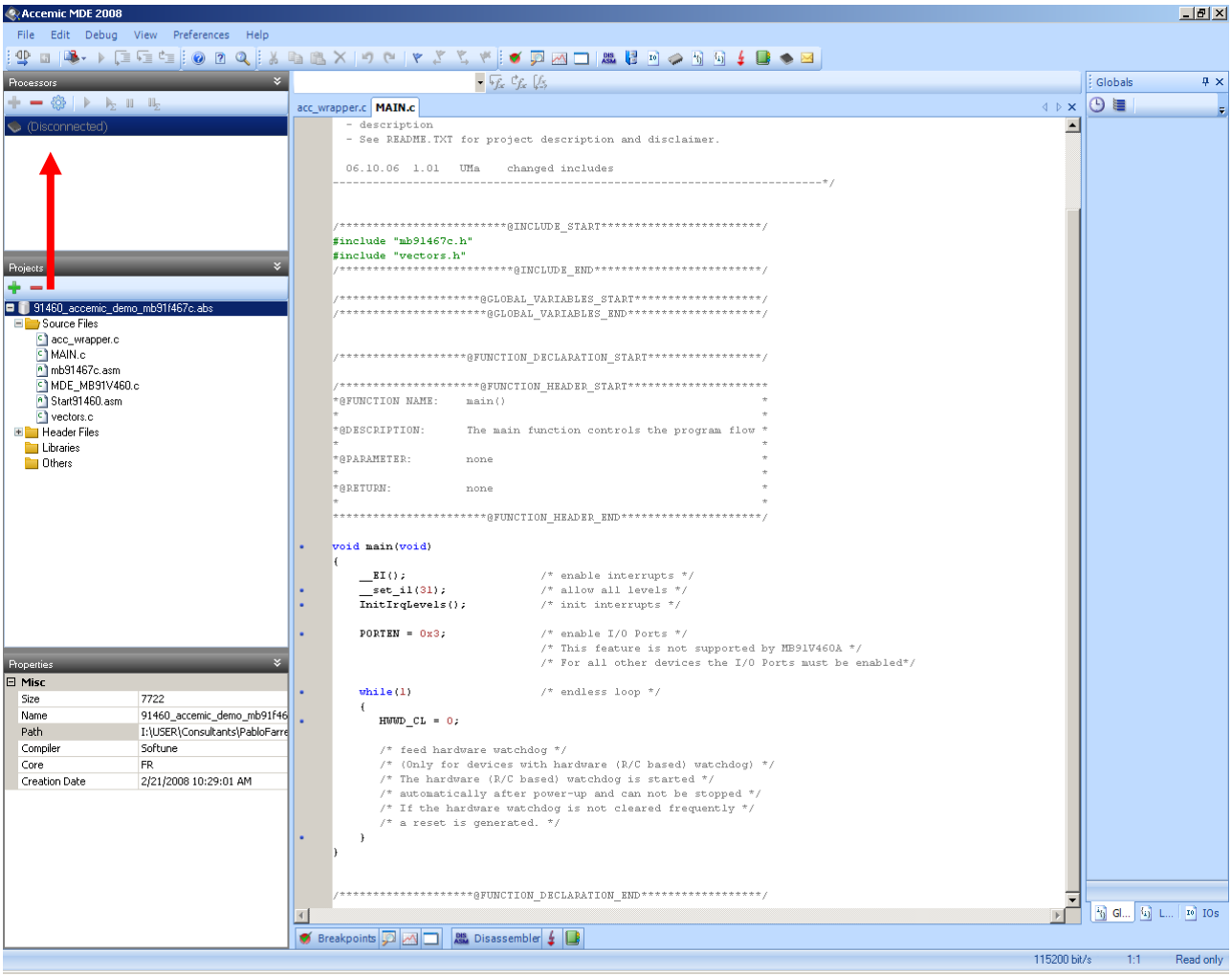


Figure 25. Processor Setup…



- Drag and drop *91460_accemic_demo_mb91f467c.abs* file into Processor Window.

Figure 26. ACCEMIC MDE 2008



- Click “connect” 
- Congratulations, you know how to configure these tools!

5 Using ACCEMIC MDE 2008 on demand

Preparing an Application to be used with ACCEMIC MDE2008 on demand

Instead of integration the Accemic MDE into the Application, it is also possible to program the ACCEMIC MDE 2008 into the Flash Memory just in case of usage.

E.g. when entering pre-series or series production, just in case a debugging of the Application is required the ACCEMIC MDE 2008 is programmed into the internal Flash memory. The Application needs to be prepared accordingly.

5.1 Preparing Application for later ACCEMIC MDE 2008 usage

To use the Accemic MDE 2008 the Application has to be prepared.

- The communication with the PC is done via UART interfaces, either UART0 or UART4 can be used with the default delivery, depending which UART are available in the used MB91460 series.
- Interrupt number 10 is used for break conditions.
- A memory area in Flash and RAM has to be reserved for the Accemic MDE

5.1.1 Function calls in ISR to select Application or ACCEMIC MDE

In the Interrupt Service Routines (ISR) of UART RX and TX as well as the debug interface a conditional jump needs to be added. When entering the ISR first a check of a value in a dedicated memory area should be done. Depending on the result the Application ISR or Accemic ISR is called. As check condition the BDME value could be used (e.g. if Value at BDME is not 0xFFFFFFFF jump to Accemic ISR, in case the option to use two debug kernels the BDME value can be check if DKV1 or DKV2 value is used).

As the Accemic MDE is not linked in the Application code, the Application needs to know the dedicated addresses of the ISR. A change of these addresses requires also a change in the Application Code!

5.1.2 Reserve memory area in Application for Accemic MDE

In order to program the Accemic MDE into the internal Flash of the MCU the Application has to reserve a memory area. The size of the memory space depends if Accemic Flash routines shall also be available to re-program the Application into the MCU memory, if required. Also RAM for STACK and Variables used by Accemic MDE needs to be reserved.

5.2 Preparing ACCEMIC MDE2008 project

To program the Accemic MDE into the MCU memory, the MDE have to setup in a project according to the required demands.

Following MDE files need to be used in the project:

- MDE_MB91V460.c
- cpu.h
- MonitorCom_MB91V460_U0.lib or MonitorCom_MB91V460_U4.lib
- MonitorCore.lib

In case that Flash programming should also be possible with Accemic MDE following files are required in addition:

- FlasherCom_MB91V460_U0.lib or FlasherCom_MB91V460_U4.lib
- FlasherCore.lib
- Flash_Type1.lib or Flash_Type2.lib (depends on used series, see Accemic help)

The project RAM and ROM size needs to be set according to the Application reserved areas. The start addresses of the ISR called in Application can easily checked in the linker mapping file *.mp1.

An addition file is required which includes a function which calls the acc_InitKernel() function. This function must have the start address of the project, where the DKVx vector is pointing to.

5.3 Example preparation

Following describing the necessary steps to prepare Application and built an Accemic MDE kernel.

The following example is based on MB91F467S series.

5.3.1 Preparing the Application

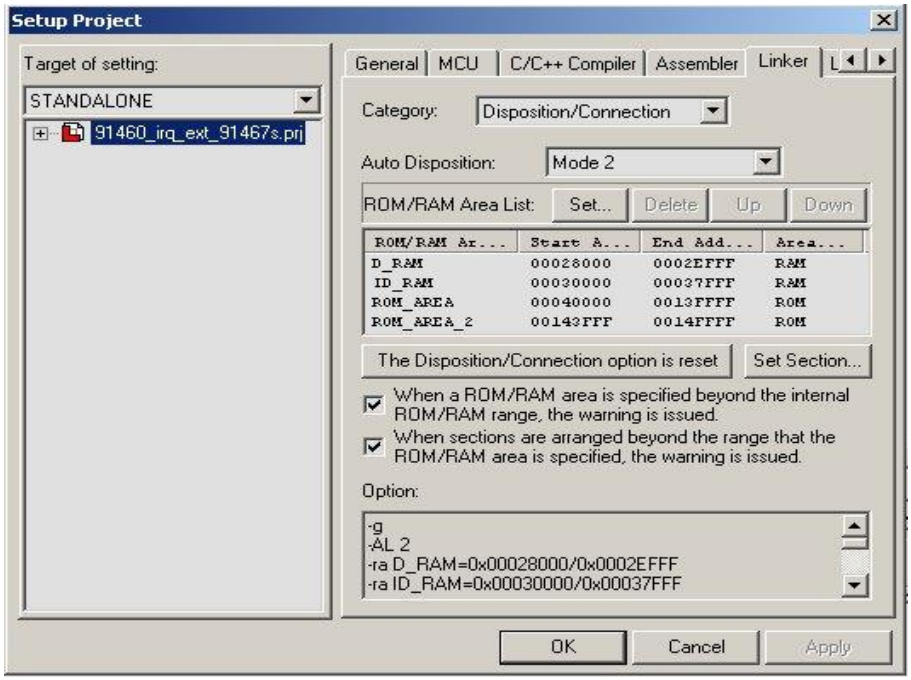
The software example 91460_acc_irq_ext_91467s-v10 is used to demonstrate the integration of Accemic MDE. The application itself is enabling the external interrupt 1, which toggles Port 15 in the isr. In case of SK-91460S-176M07 board usage, LEDs are toggled when pressing the INT1 button.

In case DKV1 pattern is set active in BDME entry, the Accemic MDE2008 is initialized. It is possible to connect to the running application for debug purposes.

5.3.1.1 Memory settings

Accemic MDE requires some RAM and ROM memory space. In this example RAM area 0x2F000 - 0x2FFFF are not used by Application. Also the ROM area 0x14000 – 0x143FFF are not used.

Figure 27. Application memory space settings



5.3.1.2 Shared Interrupt usage

To use Accemic with MB91F467S the delivery includes communication drivers for USART4 interface. In addition the instruction break interrupt is used by Accemic to issue break requests.

In MB91F467S following interrupts needs to be prepared:

- intvect no. 10 /* Instruction Break */
- intvect no. 66 /* USART (LIN, FIFO) 4 RX */
- intvect no. 67 /* USART (LIN, FIFO) 4 TX */

In FME examples the interrupt vectors are defined in vectors.c file. Here all required resource vectors are setup. The instruction break entries need to be insert into vectors.c.

As the USART interrupts shall be available for the application a wrapper is used which checks if the condition for debugging is set and calling the dedicated interrupt service routines. In case of debugging is active the interrupt of the usart4 interface are assigned for Accemic MDE usage and cannot be used by the application. Also the CR register for UART4 interrupts shall not be set when Accemic MDE is used as the debugger is setting up this register.

The address location of the Accemic MDE interrupts needs to be known. This can easily be checked via the linker mapping file (*.mp1) of the Accemic Kernel project.

Figure 28. Application vectors.c file snippet

```
#include "mb91467s.h"
#include "global.h"
#include "acc_check.h"
void InitIrqLevels(void)
{
…
    if (*BDME != 0xD3AD52C2)  /* if Accemic debugger do not set USART4 Level */
    ICR25 = 31;     /* USART (LIN, FIFO) 4 RX       */
                    /* USART (LIN, FIFO) 4 TX       */
…
}
/*---------------------------------------------------------------------------
  Prototypes

  Add your own prototypes here. Each vector definition needs is proto-
  type. Either do it here or include a header file containing them.
---------------------------------------------------------------------------*/
__interrupt void DefaultIRQHandler (void);
__interrupt void IRQHandler1 (void);
__interrupt void usart_4_rx_irq (void);
__interrupt void usart_4_tx_irq (void);
__interrupt void instr_break_irq (void);
/*---------------------------------------------------------------------------
   Vector definiton

   Use following statements to define vectors. All resource related
   vectors are predefined. Remaining software interrupts can be added here
   as well.
---------------------------------------------------------------------------*/
#pragma intvect 0xBFF8            0     /* (fixed) reset vector         */
#pragma intvect 0x06000000        1     /* (fixed) Mode Byte            */

#pragma intvect instr_break_irq   10    /* Instruction Break            */
…
#pragma intvect usart_4_rx_irq    66    /* USART (LIN, FIFO) 4 RX       */
#pragma intvect usart_4_tx_irq    67    /* USART (LIN, FIFO) 4 TX       */
…
```

The interrupt wrapper is written in Assembler as this allows a better control of the used CPU register. In case a usart4 are instruction break routine is called by hw, the wrapper checks if the Accemic condition is set and calls accordingly the dedicated isr. Within the used CPU registers (R12 and R13) are saved to stack in beginning and restored before jumping to the interrupt routines. To ensure correct register content when executing the isr.

This is done in the acc_irq_wrapper.asm file of the example. The acc_irq_wrapper_def.h includes the addresses for Accemic isr location and condition check register.

Figure 29. USART 4 isr in acc_irq_wrapper.asm file

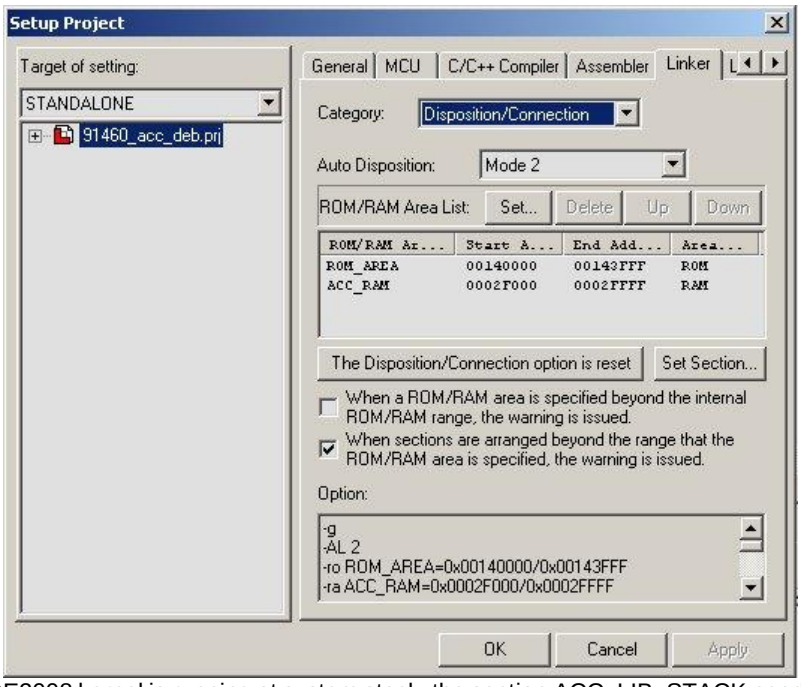
5.3.2 Preparing Accemic Kernel

```
#include acc_irq_wrapper_def.h

  .section        acc_irq_wrapper, code, align=4
/*-------------------------------------------------------------------------
   Usart_4_rx_irq()
-------------------------------------------------------------------------*/
 .import _DefaultIRQHandler  ; import application irq handler
 .export _usart_4_rx_irq     ; export usart rx wrapper for other modules
   _usart_4_rx_irq:
   ST R12,@-R15              ; save register to be restored afterwards
   ST R13,@-R15              ; save register to be restored afterwards
   LDI:32  #0x148024,R12     ; load Flash address to check if acc is available
   LD      @R12,R13          ; write value from address R12 to R13
   LDI:32  #acc_cmp_val,R12  ; load compare value
   CMP R12,R13               ; check if accemic is available
   BNE no_acc_rx             ; if not available jmp to start application irq
   LD @R15+,R13              ; restore R13 content
   LDI:32  #acc_Receive_IRQ,R12 ; load irq address
   jmp:d   @R12              ; jump delayed to acc irq
   LD @R15+,R12              ; restore R12 befor isr jump
   BRA acc_rx_end
 no_acc_rx:
  LDI:32 #_DefaultIRQHandler,R12  ; load application irq address
   jmp:d   @R12              ; jump delayed to application irq
   LD @R15+,R12              ; restore R12 befor isr jump
 acc_rx_end:
  RETI   ; only required if application irq routine is not an interrupt function
…
```

A template project can be used as start point. The ROM and RAM memory area needs to be changed according to the area which is reserved in the application. This can be done via the linker settings of the Softune Workbench. This example is using the RAM area 0x2F00-0x2FFF and the ROM area 0x14.0000 – 0x143FFF.

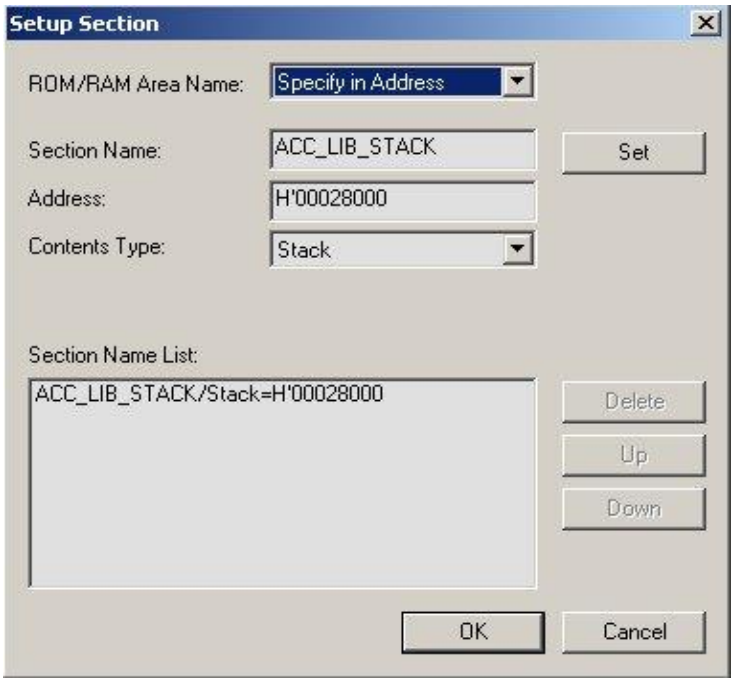
Figure 30. Accemic kernel project linker settings



The Accemic MDE2008 kernel is running at system stack, the section ACC_LIB_STACK needs to be linked to the same memory area as the system stack of your application.

In the example, the Application system Stack is linked to start address 0x28000. The locate of section can be done via Softune Workbench setup section menu (project -> setup project -> tab Linker -> category disposition/connection -> button set section).

Figure 31. ACC_LIB_STACK locate



5.3.2.1 Required project files

Following files are required to build the Accemic kernel mhx file.

- Accemic files
  - MDE_MB91V460.c
  - cpu.h
  - MonitorCom_MB91V460_U4.lib
  - MonitorCore.lib
- Start-up file
  - start_acc.asm
- MB91460S header files
  - mb91467s.asm
  - mb91467s.h

5.3.2.2 Start-up file

Instead of the start91460.asm another start-up file is required. The new file does not require to setup all clocks etc. as it is done in the standard asm file.

This start_acc.asm file is setting up the System Stack and calls the initialization routine of Accemic MDE. It is the start code which is executed when the debug vector is set.

The debug vector itself is not set in this project. They need to be programmed separately via the Flash Programming tool or application boot loader.

Figure 32. start_acc.asm file snippet

```
;=============================================================================================
; Define system stack size
;=============================================================================================
   .SECTION  ACC_LIB_STACK, STACK, ALIGN=4
    __ACC_LIB_STACK:
        .RES.B          200
    __ACC_LIB_STACK_top:
;=============================================================================================
;  Startup code for kernel
;=============================================================================================
        .import _acc_InitKernel
        .import _acc_CallKernel
        .import _acc_CheckWatchdog
        .SECTION RESET_ENTRY,CODE,locate=0x140000
KernelInit:
        LDI             #0x4C7,R7            ; clear RC watchdog
        BANDL           #0x7,@R7
         ANDCCR          #0xDF
         LDI             #__ACC_LIB_STACK_top,SP  ; initialize SSP
         ST       RP,@-R15   ; save return pointer
         STM1     (R12) ; save register R12
         LDI                          #_acc_InitKernel,R0    ; Init Kernel
         CALL                  @R0
         LDM1     (R12)    ; restore register R12
         LD       @R15+,RP ; restore return pointer
        LDI             #0x4C7,R7                    ; clear RC watchdog
        BANDL           #0x7,@R7
 RET
```

5.3.3 Programming Accemic kernel and activating the debug vector

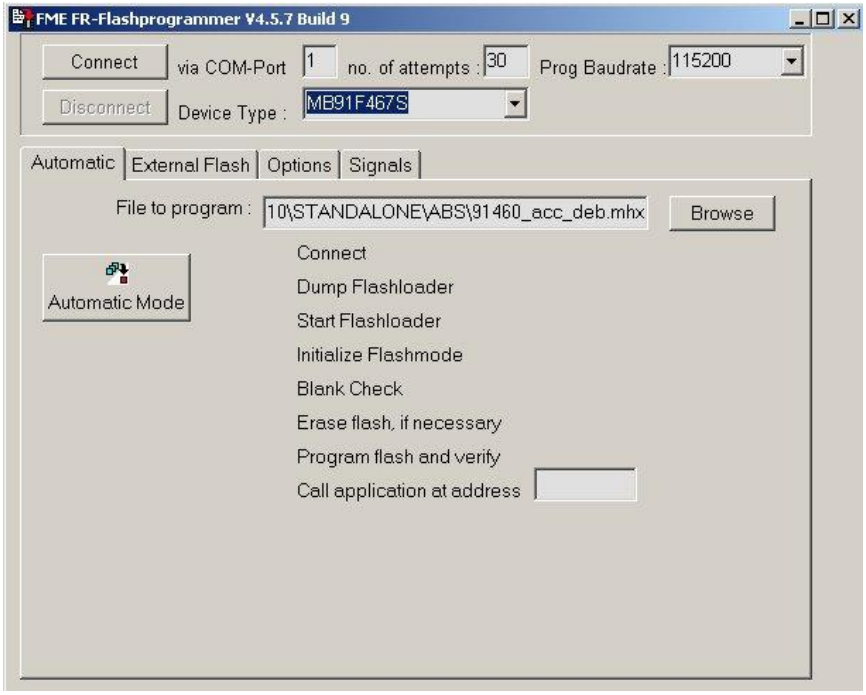
After building both projects a mhx file for application and for Accemic debugger is generated.

The Application can be programmed into the flash memory. In case of debugging is required the Accemic MDE kernel needs to be programmed in addition. Even both mhx files are programmed the Accemic MDE activated when programming the Debug kernel vectors. In this example debug kernel vector 1 (DKV1) is used.

5.3.3.1 Programming the Accemic MDE via FME Flash Programmer

Start the FME Flash Programmer, select MB91F467S as target MCU, used COM port and baud rate. Connect to the target system, e.g. SK-91460S-176M06. Select via browse button the 91460_acc_deb.mhx file. Via Automatic Mode button the Flash Programming is started.

Figure 33. Programming 91460_acc_deb.mhx file



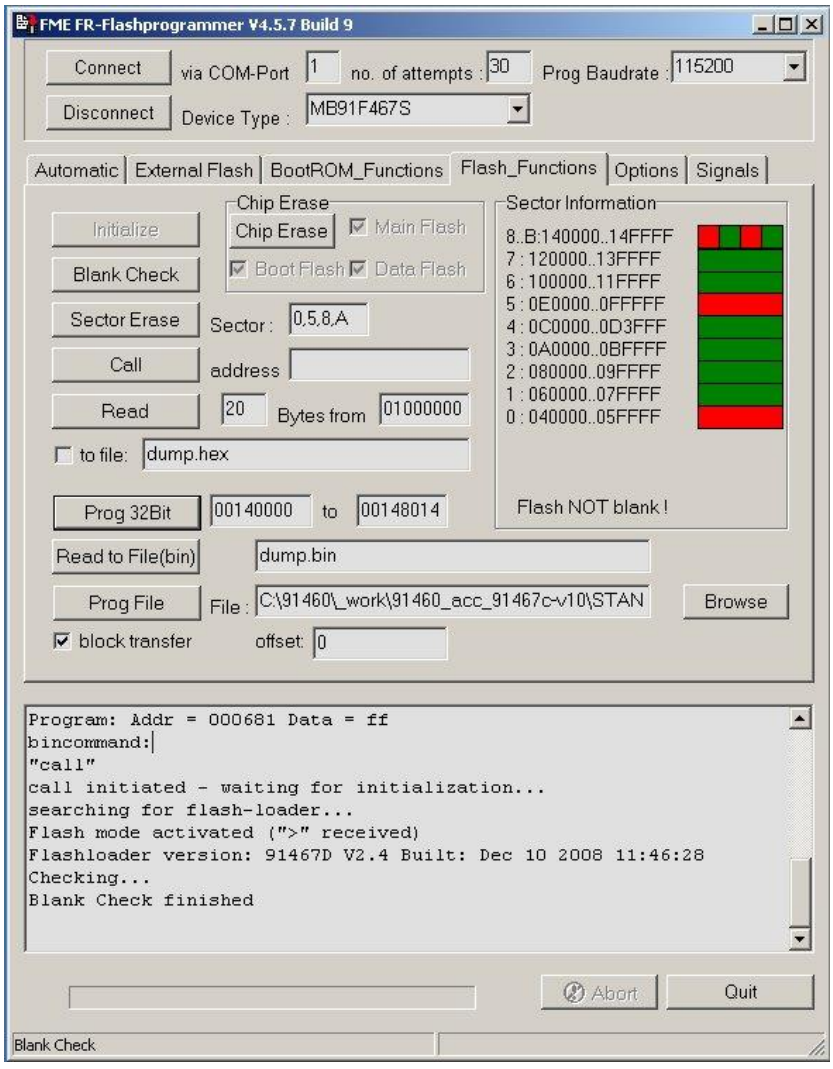
The Accemic debugger kernel is now programmed into the flash memory. However, it is not started. No communication via the Accemic MDE PC software is possible. The debug kernel vectors need to be programmed.

5.3.3.2 Programming the debug kernel vectors via FME Flash programmer

In this example debug kernel vector 1 is used.
The debug vector kernel 1 is located at address 0x14.8014. The 32-bit Word at this address needs to contain the start address of the debugger code. In this example, the start address is 0x0014.0000. To activate the debug kernel vector 1 the BDME value has to be set correctly.
The BDME is located at address 0x148024. In case of DKV1 usage the 32-bit Word data needs to be set to 0xD3AD.52C2. If this is not set correctly the DKV1 entry is ignored.
To program these two 32-bit words into flash memory the manual mode of FME flash Programmer is used.

Select MB91F467S as target MCU, used COM port and baud rate. Connect the target system UASRT 4 (e.g. SK-91460S-176M06) to PC.
Click on connect button to connect to the MCU. (press reset button at the target system to enter boot loader part)
If connected, go to tab 'BootROM_Functions' and press button 'Write file', after successful download of the file pres 'Write Byte' button and afterwards the 'Call' button.
The tab is changed to 'Flash_Functions' tab.
Enter in 'Prog 32Bit' field the 32-bit data and the address. Via the button the 32-bit data is programmed to the given address.

Figure 34. Write DKV vector



Program first 0x00140000 to address 0x148014.

Afterwards program 0xD3AD52c2 to address 0x148024.

Click on button ‘Disconnect’ to close the RS232 connection.

Assert reset at the target system to restart the application.

The Accemic Kernel is also initialized. When connecting to the target via ACCEMIC MDE debugger software it is possible to stop the running application and debug the application

5.3.4 Using application boot loader to program Accemic kernel

It is common that today’s applications include also a flash programming part, which allows update the internal Flash memory.

When using a so-called application boot loader to program the Accemic MDE and the debug kernel vector check following items.

- The boot loader must be able to program single 32-bit words into flash memory for debug kernel vector programming.
- If CRC values needs also be calculated for programming check, ensure that CRC are calculated for Accemic MDE project.

6 Related Documents and Contact

Overview of related documents and contact to CYPRESS

5.4 Manuals, Application Notes and Customer Information

- AccemicMDE2008Help.chm

5.5 Abbreviations:

- FME Cypress Microelectronics Europe GmbH
- MCU Microcontroller

5.6 Contact to Cypress

Websites:

http://www.cypress.com/cypress-microcontrollers

5.7 Contact to Accemic

Website:

http://www.accemic.com/

Contact:

http://www.accemic.com/company-profile/

Document History

Document Title: AN205379 - FR, MB91F460, ACCEMIC MDE 2008 and Boot ROM type II

Document Number:002-05379

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	02/20/2008	Initial and Preliminary Version; PFa; V0.1
			06/02/2008	Updates; PFa; V0.2
			06/06/2008	First Release, MWi; V1.0
			08/12/2008	Step-by-step setup added, MWi; V1.1
			03/26/2009	Figure 2 exchanged, added chapter 0, MSt; V1.2
*A	5135504	NOFL	02/12/2016	Converted Spansion Application Note “MCU-AN-300091-E-V12” to Cypress format
*B	5846632	AESATP12	08/24/2017	Updated logo and copyright.
*C	6070279	NOFL	02/18/2018	Sunset Review Migrated to new template Updated links

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2008-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.
TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.
Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.