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.

AN204781

F²MC-16FX Family, Softune Workbench

This is a short documentation to give an introduction how to start a project with the Softune Workbench and describes how to set up a new project for the MB96(F)34x series, but the settings for other 16FX MCU series are quite similar.

Contents

1 Introduction

The Softune Workbench is a software development environment to develop programs for the Cypress F²MC-16FX microcontroller families. It is a combination of a development manager, emulator debugger, simulator and an integrated development environment for efficient development.

This is a short documentation to give an introduction how to start a project with the Softune Workbench.

This document describes how to set up a new project for the MB96(F)34x series, but the settings for other 16FX MCU series are quite similar.

2 Getting Started

How to Begin with a New Project

2.1 Creating a new Project

If you want to create a new project it is strongly recommended to use the **template project** as a base of operations. This template project includes all necessary files and settings for a built- and debug-able project. It includes the following files:

- *Main.c* Main C file, which includes void main(void) as project main function
- *Start.asm* Start-up code, which includes user settings for the project
- *vectors.c* Interrupt vector definition, initialization and default handler C file
- *mb96xxx.asm* Assembly file of MCU register address definition
- *mb96xxx.h* MCU register header file for including in C files
- *readme.txt* Further information of the template project and version history

These files are described in detail in chapter 3.3.

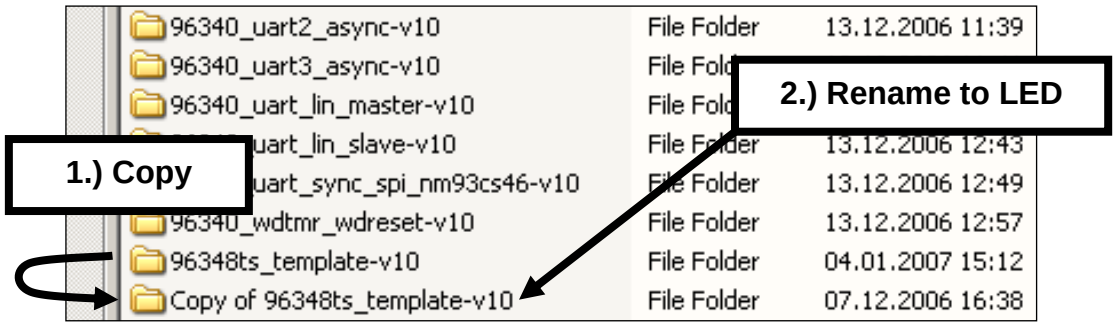
The template project can be found on Micros DVD (since Version 5.0) “Development Tools” or for the latest version (recommended) on our webpage:

http://www.cypress.com/16lx

Then choose the 16FX-MCU family you use and download the template-ZIP archive.

To start, copy the “9634*x*_template-v*x*” directory with all its sub folders.

As an example, a project is created, which toggles the LEDs on the Flash-Can-100-340 Starter Kit. For that, rename the copy of the “template” folder to e. g. “LED”.



2.2 Adjusting the project files

First click into the new “LED” folder and rename the following files:

9634x_template-vx.prj → LED.prj

9634x_template-vx.wsp → LED.wsp

Now open the file *LED.prj* with your favorite text editor.

You will find an entry like this:

```
[MEMBER-Debug]
F1=0 m 1 ABS\96348ts_template.abs
```

Please change the `abs` file name to `LED.abs`.

Save and close the file. Attention: If you use MS Word, please be sure to save the file as a normal text file, *not* as a word document!

Now open the file *Led.wsp* also with a text editor.

```
[PrjFile]
Count=1
FILE-0=96348ts_tempalte          →    FILE-0=LED.prj
ActivePrj=96348ts_tempalte.prj   →    ActivePrj=LED.prj

[SubPrj-96348ts_tempalte.prj]    →    [SubPrj-LED.prj]
```

Now save and close the file also as a normal text file.

The new project environment is now adjusted.

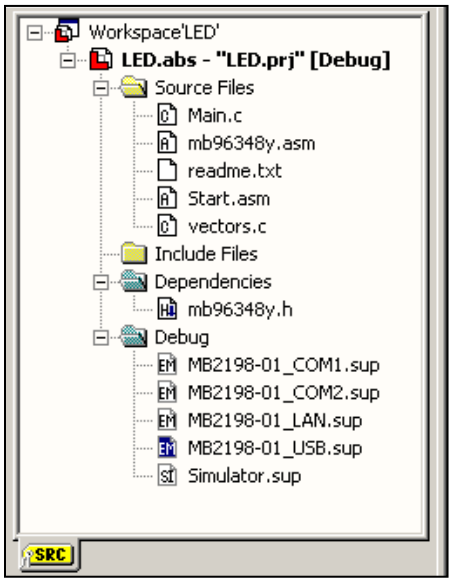
Note: “LED” is only an example name. You can choose any other.

3 The Workspace

What the Template Project Contains

3.1 Template Project

It is recommended to use the latest version of the template project, which can be found on our website >http://www.cypress.com/16lx Use then the template for the used MCU type.



3.2 The Workspace

If you open the renamed template project with the Softune Workbench with *File >Open Workspace*, browsing to the file *LED.wsp* and double-clicking it, you will see a window on the left side of the screen like in the left illustration.

This is the workspace window, which shows all members of the project.

The most important member files are the following.

3.3 Members of the Project

The members of the project are listed alphabetically in their folders.

3.3.1 Main.c

This is the main C program and the entry point of your project. Your program code will start here in the function `void main(void)`.

3.3.2 mb9634*x*.asm

In this assembly file all addresses of the resource registers are defined. The functions of these registers are described in the corresponding Hardware Manual.

Note: The name of this assembly file depends on the used MCU family.

3.3.3 readme.txt

This member file contain the version information of the template project and some useful other information. This file can be erased if not needed.

3.3.4 Start.asm

This very important file contains all software and pre-processor adjustable basic project settings and the start up code. This code is executed after Reset and before entering the `void main(void)` function.

See chapter 4 for details.

3.3.5 vectors.c

In this C file the interrupt levels are initialized and it contains the interrupt vector definition and the default interrupt handler. Corresponding interrupt control registers might be initialized as well. The user should add his own handler (or handler prototypes) here.

See chapter 5 for details.

3.3.6 mb9634*x*.h

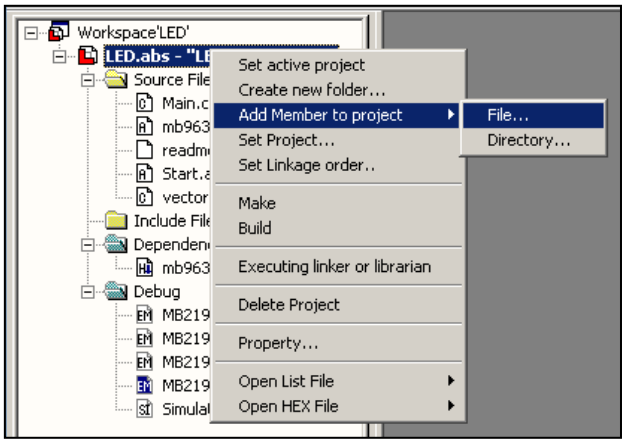
In this header file together with the mb9634*x*.asm file the bit names of the resource registers are declared and defined. They correspond also to the bit names in the Hardware Manual.

Note: The name of this header file depends on the used MCU type.

3.4 Edit the members List of the Project

Add Member to project

To add a new member to the project just click with the right mouse button on the target file name of the project (*LED.abs*) in the workspace window. A drop down menu appears to add the new member. Just browse to the corresponding location and add the module to the project.



3.4.1 Property...

Use the property tab to check the location of the *.abs* file and the location of the modules which are used in the project. With the "*Set Linkage order*" tab it is possible to set the order of the modules for the linker.

3.4.2 Set Linkage order...

The modules are displayed in the workspace in alphabetical order, but an individual linkage order can be set if it is necessary.

3.4.3 Open List file, open HEX file

The project tab can also be used to have a look at the generated List Files (generated by the Assembler, C-Compiler, Linker) and the Hex output files generated by the converter. After a build of a project a look at the mapping file is highly recommended to check the segment , sections, locations, Stack, Vector table and so on.

3.4.4 Open Source files

The source files can be opened with the editor be a double click on the source name. To create a new source file use the *File >New >Text file*. The editor supports syntax highlighting, line numbers, different font selection and some more features which can be configured by the user. To customize the editor view just open the source file and use the right mouse button to drop down the context menu.

3.5 Check Project Setup

The project setup contains the overall data, which defines the MCU environment, linker, compiler and other settings.

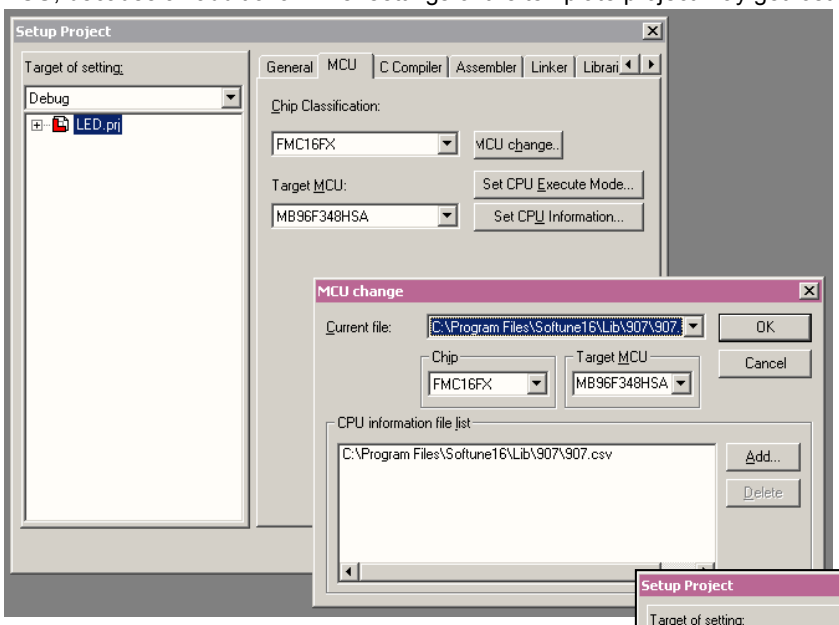
To check and modify the project setup use the *Project >Setup Project* menu.

3.5.1 General

The General settings contain the project overall data types and project paths.

3.5.2 MCU

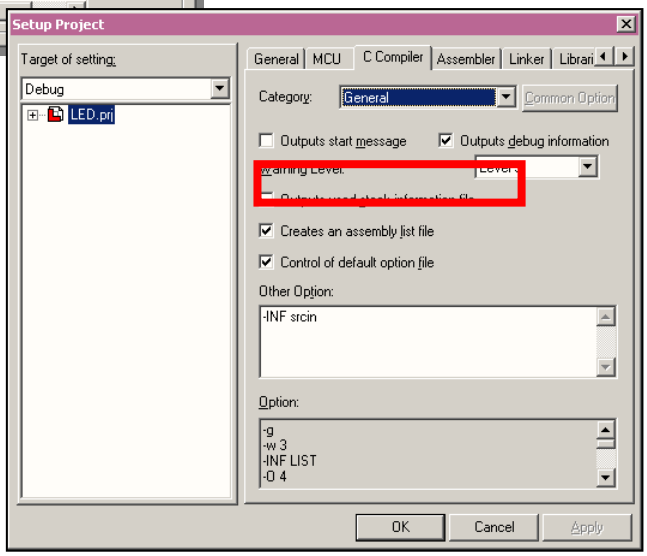
Here the MCU family, the target CPU and the detailed CPU Information can be configured. Be careful changing the MCU, because all additional linker settings of the template project may get lost and the initial setting is set.



When the “I0101W: Changed target MCU. CPU information changed to default value” window occurs, please choose Yes and when the “I0101W: The target MCU was changed. The memory map must be changed.” Window occurs choose No to save the current linker settings of the template project.

3.5.3 C Compiler

In the compiler tab of the project settings, several categories can be selected, which allows several sub settings.



Category: General

Here are several adjustments possible to control the compiler output log files, and further options can be set in textual form.

Also the generation of debug information can be set here, which is important for simulation and emulation mode.

Category: Define Macro

Preprocessor macros inclusive values can be set here.

Category: Include path

e.g. *c:\Softune\lib\907\include\sample* sets the default include path. It is possible to set further paths where header includes can be found. Be careful, that each path string *must not* end with a backslash!

Category: Optimize

Several optimization methods (e. g. for speed or size) and levels can be set. Please refer to application note *AN-900092* (Compiler Optimization) for more details.

Category: Language

Several C language settings can be adjusted here, e. g. how the `char` type is treated (`signed` or `unsigned`).

Category: Target dependent

Select the memory model which should be used (Small, Medium, Compact or Large Model). Corresponding to this setting 16Bit or 24Bit addresses will be generated for code and data addresses. Please refer to the C-Compiler Manual for more detailed information on that topic. Small memory model should be used for devices with less than 64K ROM/Flash memory only. Medium model should be used for bigger projects (code size > 64Kbyte).

Memory models:

	Data Access	Code Access	
Small	16 bit	16 bit	
Compact	24 bit	16 bit	
Medium	16 bit	24 bit	Recommended
Large	24 bit	24 bit	

RAM Constants:

Now define whether Constants should be used in RAM or not. If Constant in RAM is selected all “variables” which are declared by const in the C-source file, will be accessed in RAM. For that purpose a section `CINIT` is generated which is located by default in the RAM area. To have the correct values available in the `CINIT` section, it is necessary to copy the initial values from ROM to the `CINIT` which is done in the *Start.asm* file. To use the copy routines it is also necessary to set `RAMCONST` in the *Start.asm* file! Otherwise no copy of the initial values to the RAM area will be done!

ROM Constants:

If `ROMCONST` is selected in the *Start.asm* file, do not enable the compiler option *Const variable in RAM*! Otherwise the constants are located into the `CINIT` section. But `CINIT` is not initialized because the copy routines are not executed if `ROMCONST` is selected. If compiler is set to `ROMCONST` and the startup file (*Start.asm*) is set to `RAMCONST`, the startup file will only generate an empty section.

The code, which copies from `CONST` to `CINIT` will not have any effect then.

Note: It is recommended to set the compiler to `ROMCONST` for single-chip mode or internal ROM + external bus.

3.5.4 Assembler

Category: General

Here are several adjustments possible to control the assembler output log files, and further options can be set in textual form.

Also the generation of debug information can be set here, which is important for simulation and emulation mode.

Category: Define Macro

Preprocessor macros inclusive values can be set here.

Category: Include path

e.g. *C:\Softune\lib\907\include\sample* sets the default include path. It is possible to set further paths where header includes can be found. Be careful, that each path string *must not* end with a backslash!

Note: With the above mentioned include path, it is referred to the default header files supplied by Cypress Limited. Cypress Microelectronics Europe (Germany) also offers some header files, which are compatible to the previous Softune Versions. Therefore please check our website http://www.cypress.com/16lx for updates.

Category: Output list

Here are special adjustments possible to control the assembler output log files.

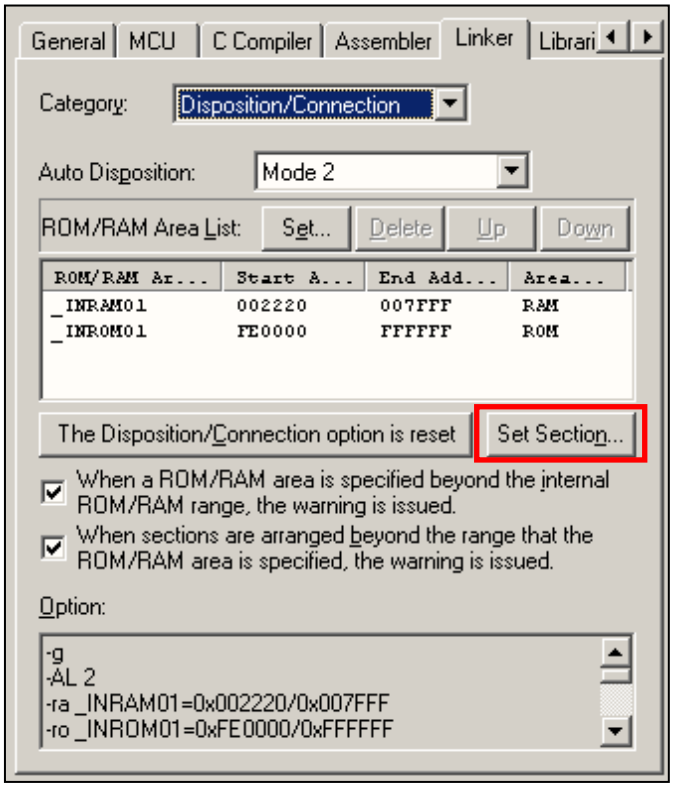
3.5.5 Linker

Category: General

Here are several adjustments possible to control the linker output log files, and further options can be set in textual form.

Also the generation of debug information can be set here, which is important for simulation and emulation mode.

Category: Disposition/Connection



One of the basic settings are the linker settings to define the RAM and ROM areas with the adequate location settings for the used segments.

Disposition “Mode 2”. In this mode the linker will place the sections and segments corresponding to the attribute type into the RAM or ROM area automatically. Now the RAM and ROM areas must be defined.

ROM/RAM Area List (e. g. for MB96348) :

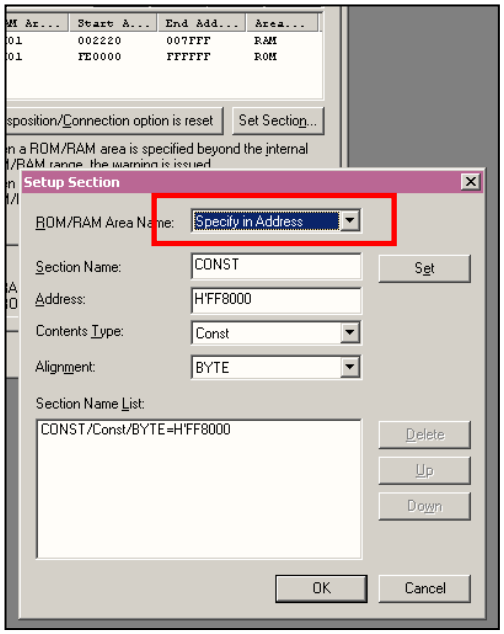
RAM
Area Attribute: `RAM`
Start Address: `0x002220`
End Address: `0x007FFF`

ROM
Area Attribute: `ROM`
Start Address: `H'FE0000`
End Address: `H'FFFFFF`

Also, the `const` section for the constants has to be set in order to be compliant to the ROM Mirror function (if you want to use small or medium memory model, which uses 16-Bit-Addressing):

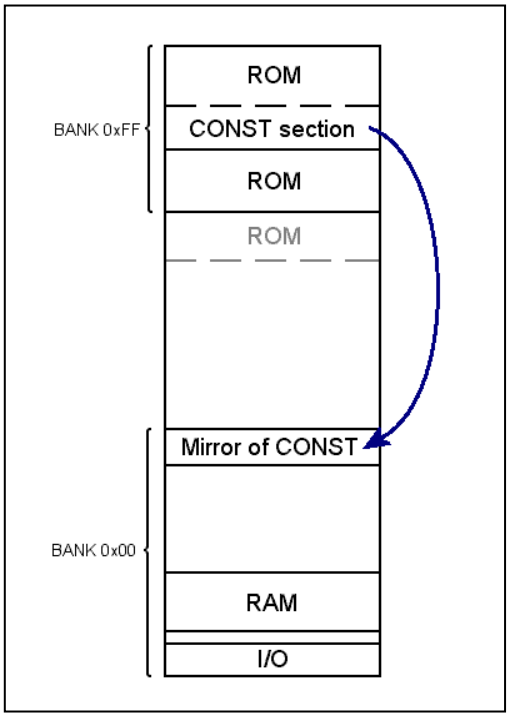
Therefore use *Set Section …>ROM/RAM>Area Name: Specify in Address*
Section Name: `CONST`
Address: `FF8000`(see: ROM-Mirror)
Content Type: `Const`
Alignment: `BYTE`

Click on *Set*, then *OK*.



ROM-Mirror

The ROM-Mirror-Function allows 16-Bit-Addressing to the ROM constants although they are placed in the `0xFF`-Bank.



The `const`-section is mirrored to an address space in the `0x00`-Bank, which is usually the data bank, because it contains the RAM area.

The ROM-Mirror area depends on the device. Please see the hardware manual for the correct address.

The illustration on the right side illustrates the ROM-Mirror-Function.

Please note, that for 16FX devices the ROM mirror size is adjustable and the mirror area can be shared with external memory space. Please refer to the application note *mcu-an-300212-e-16fx_rom_mirror* for detail information.

Category: Register Banks

This linker setting reserves RAM memory corresponding to the selected CPU registers bank(s). The general register bank can be selected in the *Start.asm* code (see General Register Bank). The register bank can be selected in the project with the `#pragma register n` and *#pragma noregister* directives. Please refer to the *CM42-00327-E Embedded C Programming Manual* for details.

Category: Define Symbols

This setting is used for defining special linker symbols.

Category: Output List

Here are special adjustments possible to control the linker output log files. The warning message during project built, that libraries do not contain debug information can be suppressed here.

Category: Absolute Assembly List

Here settings for creating absolute assembly lists are possible. This feature is very useful for debugging. The compiled and linked results for selected modules can be output. After linking these files have the extension `.ALS` and contain all used symbols, source code lines and compiled mnemonic and assembly code including addresses. The standard output for these files is the *LST* directory of the project.

Category: Control Library

Linker library settings can be done in this category.

3.5.6 Librarian

Category: General

Here are special adjustments possible to control the librarian output log files.

Category: Output list

Here are further adjustments possible to control the librarian output log files.

3.5.7 Converter

Here the format of the converted object code can be adjusted.

Cypress recommends *Motorola S Format* for output data to stay compatible with the Flash Programming Tool supplied by Cypress.

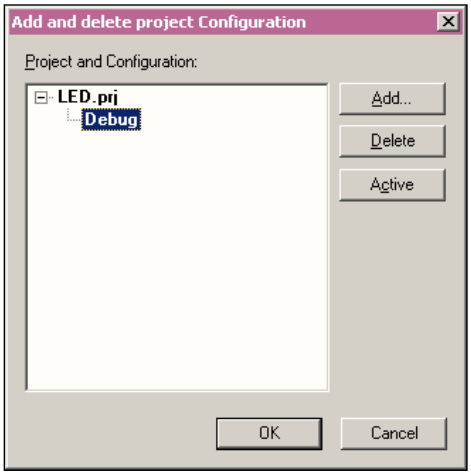
3.5.8 Debug

Debug information file adjustment and simulator/emulator setup can be administered here.

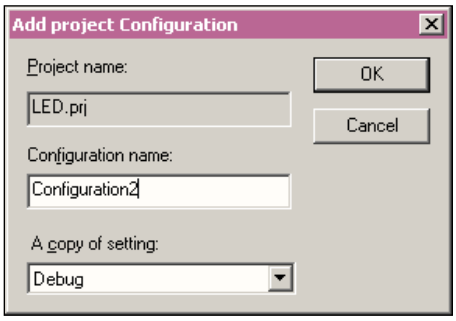
3.6 Project Configuration

Sometimes it is necessary to have more than one project configuration (i. e. linker and compiler setting, etc.). With the menu *Project>Configuration>Add and Delete…* the user can manage several project settings.

The default setting is *Debug*:



With *Add* the user can set a new configuration. The project settings of the last setting will be over taken. New settings will be assigned to the actual configuration name.

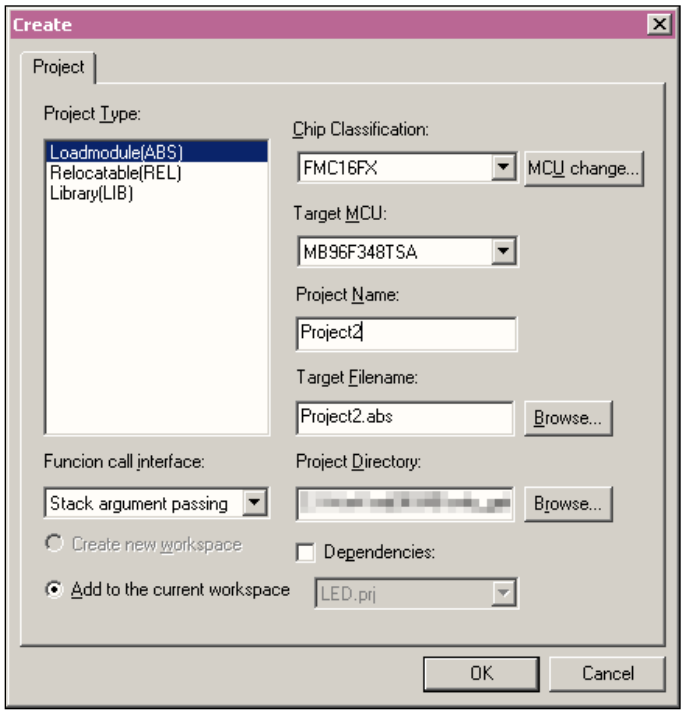


Afterwards it is possible the switch between the different configurations by the pull down menus in the tool bar.

The actual configuration is displayed in the workspace window behind the project *abs* file name. In this case it would be [Debug] or [Configuration2].

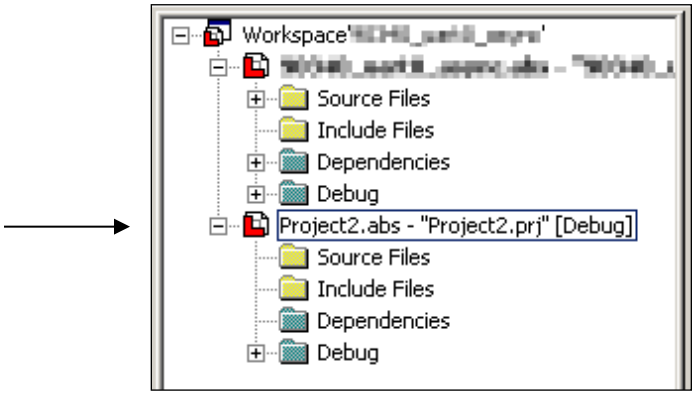
3.7 Adding Projects

It is possible to have more than one project in the Softune Workbench. With *Project>Add Project>New* a new project can be added. It is also possible to add an existing project to the workbench.



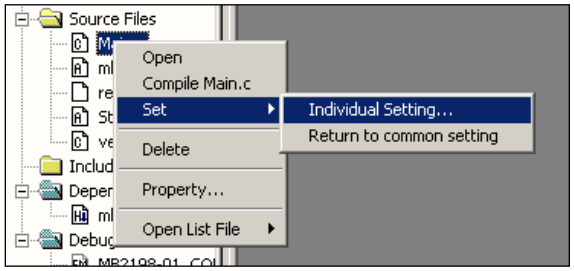
If choosing the menu *New* a completely empty project will be created. Therefore it is recommended to add a new template project for the desired MCU derivative and choose the menu *Add*.

After including a new or existing project the workspace window will be updated with the new project.

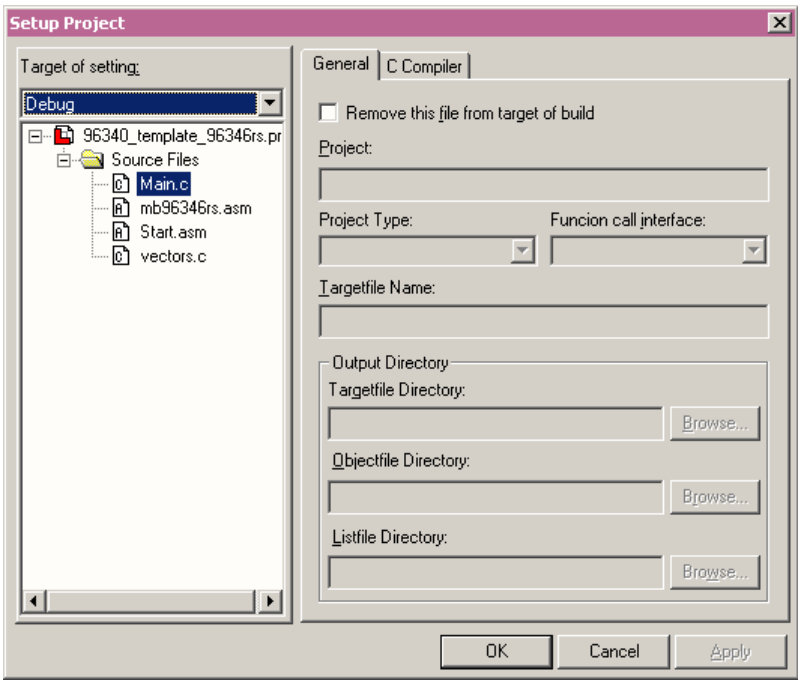


3.8 Individual Settings

For each member (module) of a project individual settings can be defined. For this right-click on the member in the workspace window and choose *Set> Individual Setting*….



Then the individual setting dialog opens.



For *c* modules individual compiler settings (e.g. optimization level) can be defined, and for *asm* modules the same applies to the assembler.

4 The Start-Up File

What the Start-Up Assembly File Contains

4.1 General

The start-up file *Start.asm* is the entry point just after a Reset is performed.

It contains several setting for the compiler and the device operation. It initializes the stack and the variable initial values. Afterwards the *void main(void)* function is called.

The user can change settings, which contain the symbols "<<<" in the comment lines.

4.2 Start.asm

The following settings can be done in the *Start.asm* file.

4.2.1 Settings

Controller Series and Device

Here the general 16FX-MCU series and the device have to be selected. In our example case it is:

```
#set       SERIES     MB96340
#set       DEVICE     MB96348YB
```

Please take care for MB96340 series. When a device of this series is used, please also specify correct `DEVICE` definition. The numbering of this definition *must* not be changed, because it is used for correct baud rate calculation for background debugging and clock setting.

Memory Model

In this definition the user selects whether the *start.asm* code has been compiled for small, medium, compact, or large memory model. Corresponding to this setting, the `main`() function of *main.c* will be called at the end of the *Start.asm* program code by a 16-Bit-Address `CALL` or 24-Bit-Address `CALLP` instruction. Note, that this option does not influence the compiler settings.

It is recommended to use `MEMMODELAUTOMODEL`, which will use a `CALLP` instruction, which always works.

Register Passing Setting

If register function argument passing instead of stack argument passing is used, this option sets the assembler pseudo code. The regarding compiler switch can be found in *Project>Setup Project…*: *Function call interface*. In the assembler settings the macro __REG_PASS__ has to be added (*Project>Setup Project…>Assembler*: *Define Macro*). A Value is not needed.

Please note, if the pseudo op-code `.REG_PASS` is missing in the *MB96xxxxx.asm* module, please add it for register function argument passing option.

Constant Data Setting

Here the user can choose RAM or ROM constant handling. It is recommended to use `CONSTDATAAUTOCONST` handling. Then this mode can be changed in the compiler settings if needed (see chapter Category: Target dependent and Category: Disposition/Connection).

If RAM constants are used please refer to Category: Target dependent for further details.

Stack Type and Size

In this configuration the user can choose whether to use the user or the system stack. It is recommended to use the system stack: `STACKUSE SYSSTACK`. Note: Interrupts will *always* switch to system stack.

Also the stack size can be set. We recommend a stack size (`STACK_SYS_SIZE` / `STACK_USR_SIZE`) of 768 bytes. The other stack (if not used) should get the size value 2.

Please note, that some library functions (e. g. `sprintf()`) needs a big stack size. Please see the corresponding *stk* file of the *.\lib\907* directory of the Softune installation path.

With `STACK_FILL ON` the stack will be filled by the start-up code with the word pattern given in `STACK_PATTERN`.

General Register Bank

It is recommended to use register bank `0` for `REGBANK`. If this is modified, the corresponding register bank memory space must also be reserved in the linker options (see Category: Register Banks).

Low Level Library Interface

The `CLIBINIT` option has only to be set, if stream-IO/standard-IO function of the C-library have to be used (`printf()`, `fopen()` ...). This also requires low-level functions to be defined by the application software. For other library functions like (e.g. `sprintf()`) all this is not necessary. However, several functions consume a large amount of stack.

(Please see also: CM81-00204 SoftuneC Compiler Manual – Library Incorporation)

Clock Settings

Here the crystal frequency (CRYSTAL) and the MCU clock speeds (CLOCK_SPEED) can be set. Several different settings for the clock domains in the 16FX-MCU can be chosen. Please refer to the *Start.asm* code itself for details. The listed clock settings are suggestions for several clock speeds. If the user wants a customized setting please always refer to the corresponding datasheet for maximum clock frequencies and other restrictions.

Please note that some devices need special clock settings due to restrictions. Therefore please set correct device series and, if needed, the device itself. Please see chapter Controller Series and Device.

Clock Stabilization Time

The clock stabilization time (MC_STAB_TIME) after power on can be set here if CLOCKWAIT is set to ON. The stabilization time is needed, if a stable clock is needed when the main function is called to have e. g. stable baud rates for CAN communication or fix frequencies for PWM signals.

External Bus Interface

In this setting configuration for the external bus interface (if available) can be done. It contains the settings for the chip select signals, address pins and other bus signals. For the chip select signals an own memory area can be selected. The default value is BUSMODE SINGLE_CHIP, which means, that the external bus interface is switched off.

ROM Mirror

The ROM mirror function and its memory size can be selected here (see chapter Category: Disposition/Connection). When using this function ROMMIRROR has to be set to ON.

Flash Security

Here the Flash Security can be set (MAIN_SECURITY_ENABLE, SATELLITE_SECURITY_ENABLE). The user can set own unlock keys here (MAIN_UNLOCK_*n*, SATELLITE_UNLOCK_*n*).

Flash Write Protection

Each sector of the Flash memory can be prevented from accidental erase or overwrite in this setting (MAIN_FLASH_WRITE_PROTECT ON and individual sector setting, e. g. PROTECT_SECTOR_SA0).

Please note that these registers will be set by the Boot ROM and are write once register. This means, if a sector is protected via these settings, it can't be unlocked by the application anymore.

Boot Vector

This option sets the Boot Vector to the start code of this start up file. The boot vector can be set by vector (address 0xFFFFDC), fixed (pointing to address 0xDF0080) or off (for e.g. boot loaders).

UART scanning

If this setting is selected, the Flash programming UARTs are scanning for (serial programming) communication even in RUN mode (internal vector mode). The scanning time is typically about 250 ms (maximum about 500 ms).

Note: This setting is active the clock stabilization time may be chosen smaller (chapter 0.0.0)

RAM code copying

This section declares if a special RAM code section should be copied to the RAM memory area or not. RAM code is needed, if the MCU does not provide Satellite or Dual Operation Flash memory and the Main Flash contents have to be modified. Please specify COPY_RAMCODE ON. Note, that in this case a special section named RAMCODE has to be used. Please see Flash programming example project and application note *mcu-an-300218-e-16fx_flash* for details.

Version Stamp

Here the user can set a version stamp string (VERSION_STAMP). It get an own section (VERSIONS).

Background Debugging Settings

In this section the background debugging mode can be set. Please specify the debugging USART channel and the used baud rate (please consider notes in chapter 0.0.0). For further setting please refer to *Start.asm* code notes and application note *mcu-an-300235-e-16fx_using_euroscope*.

At this point the user settings are complete.

4.2.2 Start Procedure

The following code in the assembly file performs the oscillation stabilization and the setting from above the user has done. Additionally all uninitialized global variables are set to 0 and constants are copied to RAM (in case of `RAMCONST` selected), the stack is initialized and filled with a pattern if selected.

If selected, the section `@RAMCODE` is copied to RAM from ROM section `RAMCODE`.

After this the function `voidmain(void)` of the module *main.c* is called.

5 Interrupt Vectors

How Interrupt Vectors are Defined and Initialized

5.1 Vectors.c

5.1.1 Interrupt Levels

In the module *vectors.c* first the interrupt level function `void InitIrqLevels(void)` is defined:

```
#define MIN_ICR  12
#define MAX_ICR  88

#define DEFAULT_ILM_MASK 7

void InitIrqLevels(void)
{
volatile int irq;

for (irq = MIN_ICR; irq <= MAX_ICR; irq++)
{
ICR = (irq << 8) | DEFAULT_ILM_MASK;
}

    ICR = (17 << 8) | 2;  // Example: External Interrupt (17) with priority 2
}
```

This function initializes the interrupt control register, which e. g. defines the interrupt levels for each interrupt channel with a common level (`DEFAULT_ILM_MASK`). 7 means lowest priority (i.e. disabled), 0 the highest. The user has to put levels for his own service routines here (see example above for interrupt 17).

5.1.2 Interrupt Handler Function Prototypes

After the level initialization, the function prototype of the interrupt handler has to be declared:

```
__interrupt void DefaultIRQHandler (void);
__interrupt void My_IRQ_Handler (void); // <-- Example for user handler
       . . .
```

Note: Interrupt function qualifier has to begin with two underscores: "__".

5.1.3 Vector Definition

Here the vector numbers are linked with the interrupt handler functions. For non-used interrupts the default interrupt handler is used:

```
#pragma intvect DefaultIRQHandler 12   /* Delayed Interrupt            */
#pragma intvect DefaultIRQHandler 13   /* RC Timer                     */
#pragma intvect My_IRQ_Handler    14   /* Main Clock Timer             */
#pragma intvect DefaultIRQHandler 15   /* Sub Clock Timer              */

   . . .
```

Enter your interrupt handler function(s) here.

5.1.4 Default Interrupt Handler

At the end of the *vectors.c* module the default interrupt handler function is listed. This function performs a system halt with an endless loop. For debugging purposes it is quite useful to set a breakpoint here in order to detect missing interrupt handler functions.

```
__interrupt void DefaultIRQHandler (void)
{
    __DI();              // disable interrupts
while(1)
    {
        __wait_nop();    // halt system
    }
}
```

6 Main Program

Main.c

6.1 The Main.c Program

The module *main.c* contains the main function `void main(void)` which is called by the S*tart.asm* file.

6.2 Example Code

For the LED example project type in the necessary lines, so that your *main.c* file looks like this:

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.          */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                              */
/*----------------------------------------------------------------------------
  MAIN.C
  - description
  - See README.TXT for project description and disclaimer.

  ----------------------------------------------------------------------------*/

#include "mb96348ys.h"

unsigned long cnt;

void main(void)
{
    InitIrqLevels();
    __set_il(7);              // allow all levels
    __EI();                   // globally enable interrupts

    DDR0 = 0xFF;              // set parallel port direction register : output
    PDR0 = 0xFF;              // switch off all LEDs

while(1)
    {
for(cnt = 0; cnt < 60000; cnt++);     // wait loop
PDR0++;                             // counter...
    }
}
```

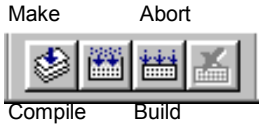
This small example code toggles all port LEDs of the FLASH-CAN-100P-340 board or the right 7-segment LED of the SK-16FX-100PMC board respectively.

7 Build a Project

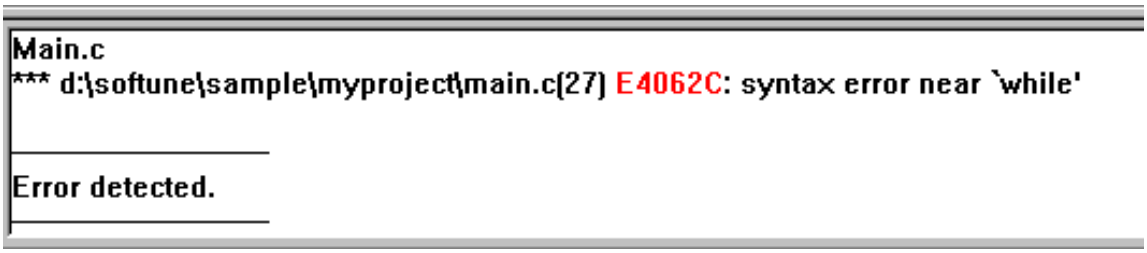
How to Build a Project

7.1 Building a Project

After the basic settings and entering the program code a Compile, Make or Build command can be executed. Therefore the special buttons can be used:



If there are errors you will get warning or error messages in the output window corresponding to the selected warning level (Tool option settings).



To jump to the source code line which invoked this error message just double click the error/warning message or use F1 to get the adequate help information for this message.

7.2 Fine Tuning and Error Checking

After Linking it is always worthwhile to have a look at the linkage map and the output list files which were generated. In the Linker Mapping file (**.mp1*) it can be checked whether any section or segments are overlapping. The location of the sections or segments itself can be checked (e.g. Stack area) and the I/O register addresses can be checked. Right at the beginning also the linkage order can be seen. The other generated list files (if enabled in the Tool options) contains further detailed information which are very useful in case of debugging and error search.

The following Table shows the section listing, which is an extract from the Linker mapping file. You can view it by clicking with the right mouse button to the entry .abs in the workspace window and choosing *Open List File>LED.mp1*.

Among other things you can find the following list:

- F2MC-16 Family SOFTUNE Linker Mapping List

```
S_Addr. -E_Addr.   Size     Section Type Al    Sec.(Top 81)
00000000-000000E6  000000E7  IO    N RW-- 00 ABS  IOBASE
00000100-0000012F  00000030  DATA  N RW-- 00 ABS  DMADESCRIPTOR
00000180-0000018F  00000010  DATA  N RW-- 01 ABS  Register Bank No. 00
00000380-000008CE  0000054F  DATA  N RW-- 00 ABS  IOXTND
00004240-........  00000000  DATA  P RW-- 02 REL  DATA
00004240-........  00000000  DATA  P RW-- 02 REL  DATA2
00004240-........  00000000  DIR   P RW-- 02 REL  DIRDATA
00004240-........  00000000  DATA  P RW-- 02 REL  LIBDATA
00004240-........  00000000  DATA  P RW-- 02 REL  INIT
00004240-........  00000000  DATA  P RW-- 02 REL  INIT2
00004240-........  00000000  DIR   P RW-- 02 REL  DIRINIT
00004240-........  00000000  DATA  P RW-- 02 REL  LIBINIT
00004240-........  00000000  DATA  P RW-- 02 REL  CINIT
00004240-........  00000000  DATA  P RW-- 02 REL  CINIT2
00004240-0000453F  00000300  STACK P RW-- 02 REL  SSTACK
00004540-00004541  00000002  STACK P RW-- 02 REL  USTACK
00DF0000-00DF001B  0000001C  CONST N R--I 00 ABS  MAIN_SECURITY
00DF001C-00DF002F  00000014  CONST N R--I 00 ABS  MAIN_PROTECT
00DF0030-00DF0033  00000004  CONST N R--I 00 ABS  BOOT_SELECT
00DF0034-00DF003F  0000000C  CONST N R--I 00 ABS  UART_SCAN_SELECT
00DF0040-00DF007F  00000040  CONST N R--I 00 ABS  BDM_CONFIG
00FC0000-00FC00B8  000000B9  CODE  P R-XI 01 REL  CODE_START
00FC00B9-00FC00C3  0000000B  CODE  P R-XI 01 REL  CODE_main
00FC00C4-00FC00EA  00000027  CODE  P R-XI 01 REL  CODE_vectors
00FC00EC-........  00000000  CONST P R--I 02 REL  DCONST
00FC00EC-........  00000000  CONST P R--I 02 REL  LIBDCONST
00FC00EC-........  00000000  CONST P R--I 02 REL  CONST2
00FC00EC-00FC00FD  00000012  CONST P R--I 02 REL  DCLEAR
00FC00FE-00FC012F  00000032  CONST P R--I 02 REL  DTRANS
00FC0130-........  00000000  DIRC  P R--I 02 REL  DIRCONST
00FF8000-........  00000000  CONST P R--I 02 REL  CONST
00FFFE7C-00FFFFD3  00000158  CONST N R--I 00 ABS  INTVECT
00FFFFDC-00FFFFDE  00000003  CONST N R--I 00 ABS  RESVECT
```

Please note that all sections of the modules can be found with the prefix `CODE_`.

In the example LED project *Main.c* creates the section `CODE_main`, *Start.asm* creates `CODE_start`, and *vectors.c*`CODE_vectors`.

For detailed information of this mapping list please refer to application note: *mcu-an-390108-e-default_data_sections*.

8 Download Program to Flash MCU

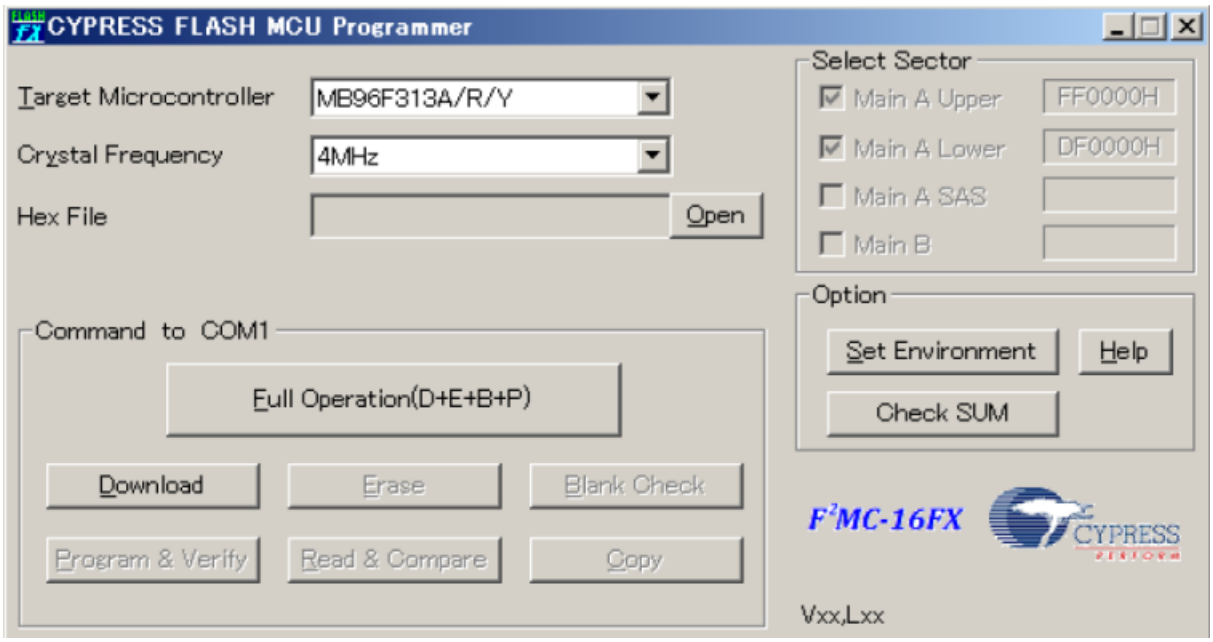
How to Download the Object Code to a Flash MCU

8.1 Programming a Flash MCU

To program a Flash MCU first install the Cypress Flash MCU Programmer Software. The newest free-of-charge version can be downloaded at:

http://www.cypress.com/products/16fx

After installing open the Programmer Software:



8.1.1 Settings

- Target Microcontroller: Choose the MCU you usehere
- Crystal Frequency: Choose here the crystal/quartz frequency you use on your target system
- Hex File: Browse to your project and open the compiled code (e. g. *…\LED\Abs\LED.mhx*)
- Set Environment: Choose here the PC-COM Port, which is connected to your target system.

8.1.2 Download the compiled Code

You either can choose Full Operation or the single steps: Download, Erase, Blank Check, Program & Verify.

Be sure, that the mode pins of your MCU to program are set to asynchronous programming mode. Please refer to the corresponding hardware manual for details.

9 Simulator

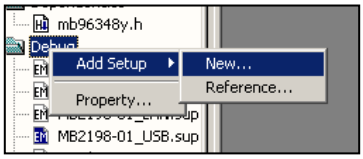
Using the Softune Simulator

9.1 Core Simulator

The simulator is only a CPU core simulator, with no simulation of the peripherals except I/O access and interrupts.

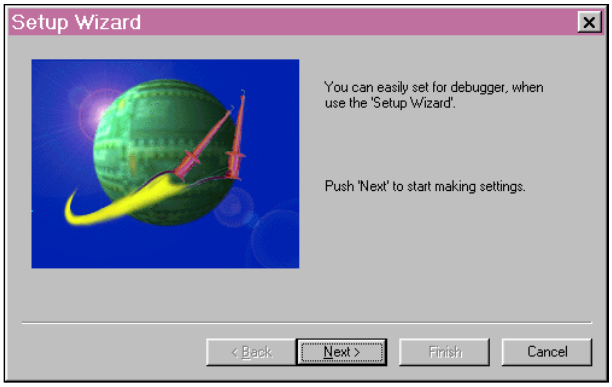
9.2 Simulator Set up

This paragraph can be skipped, if the simulator is already configured (using *simulator.sup* of the template project).

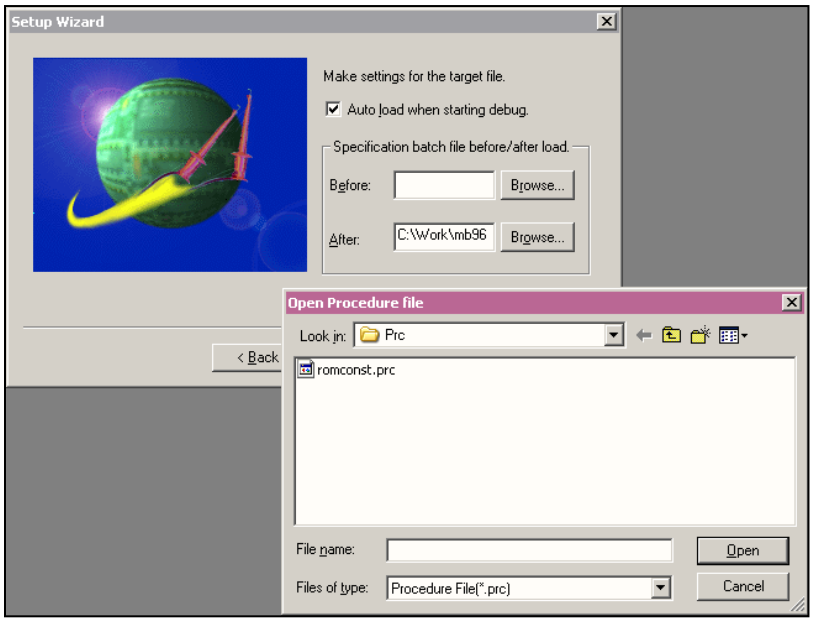


To set up the simulator, click with the right mouse button to the *Debug* entry in the workspace window. Then choose *Add Setup>New…* . You will be asked for a setup name. Type in e. g. *Simulator*.

The Setup wizard will open:



Click on *Next* >. Then click on *Simulator Debugger* and on *Next* > again. Next you are asked for a *Batch file*. Skip this item by clicking on *Next* > again.



The next item makes settings for the target file. Click on the check box *Auto load when starting debug*. At the entry *Specification batch file* ***after*** *load* choose the file *romconst.prc* which is located in the subfolder *prc* of the template project. It is recommended to check the correct file path. This procedure file ensures, that the ROM mirror memory space is copied to the corresponding RAM area and the reset vector entry is performed correctly.

Click on *Next* >.

Choose *Select all* in the next item and click on *Next* > again. Click on *Finish* at the last item window.

9.3 Using the Simulator

To enter the simulator debugging mode, double-click on the simulator entry **.sup* in the Workspace window or choose *Debug>Start Debug*.

In the command window the comments of the procedure file for the ROM constants/mirror is displayed:

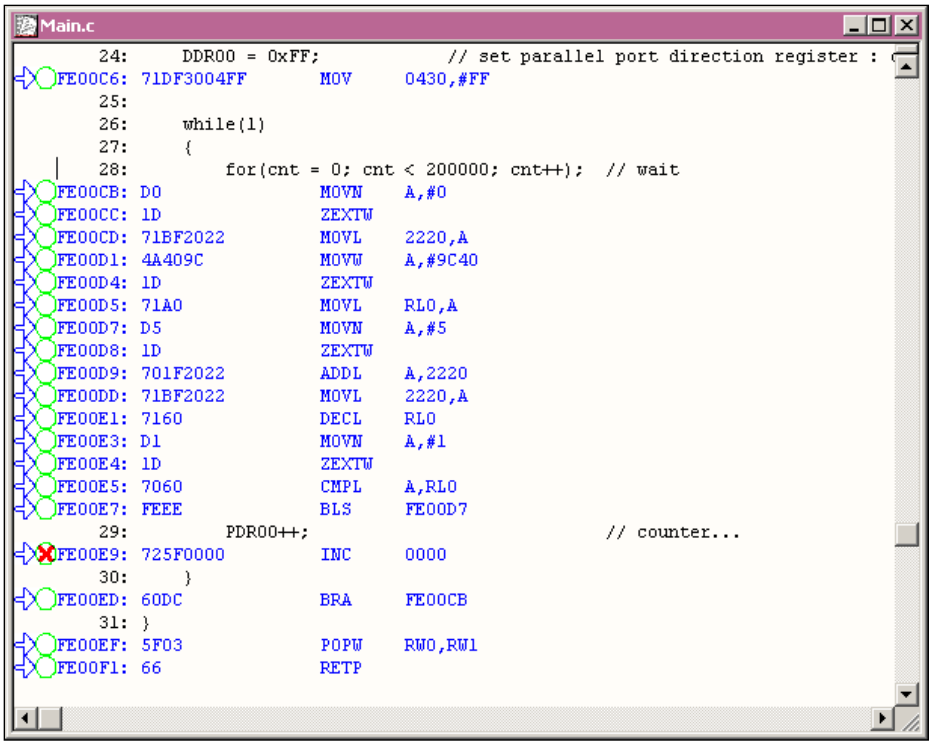
```
>Set ROM-mirror memory map...
address          attribute
000000 .. 000BFF read write
000C00 .. 000FFF undefined
001000 .. 001501 read write
001502 .. 0021FF code read write
002200 .. 002525 read write
002526 .. 007FFF code read write
008000 .. 00FFFF read
010000 .. 01FFFF code read write
020000 .. 0EFFFF undefined
0F0000 .. 0F0FFF read write
0F1000 .. 0FDFFF undefined
0FE000 .. 0FFFFF code read
100000 .. DDFFFF undefined
DE0000 .. DE00AA code read
DE00AB .. DEFFFF undefined
DF0000 .. DF0001 code read
DF0002 .. DF001B undefined
DF001C .. DF001F code read
DF0020 .. DF002F undefined
DF0030 .. DF0033 code read
DF0034 .. FDFFFF undefined
FE0000 .. FFFFFF code read
>Copy ROMCONST for simulation...OK
------------------------------------------------------------
Use command "batch prc\romconst.prc" after each download
------------------------------------------------------------
```

This procedure file copies the `const` ROM area from bank `0xFF` to the ROM mirror address area in bank `0x00` in order to the ROM Mirror Function with the Simulator and set read/write access to the different memory areas.

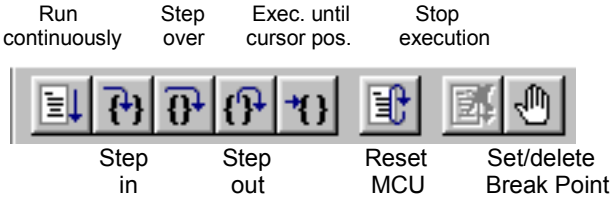
9.4 Source and Assembly View

To start debug, open the main program *main.c* by double-clicking on its entry in the workspace window.

To observe the source code together with the compiled assembly code, click with the right mouse button into the *main.c* window and choose *Mix display*.



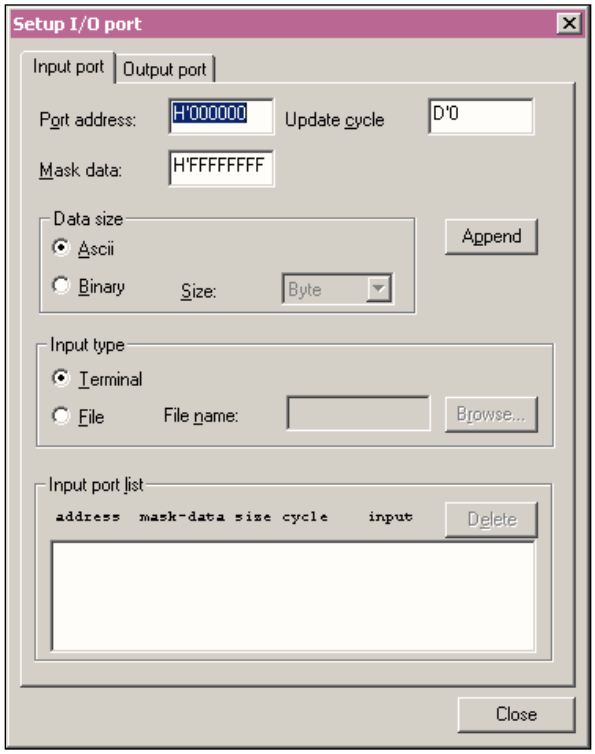
You can use the following buttons for debugging:



A break point is visible as a red cross within the green circle at the beginning of each instruction line. Setting and deleting a break point is possible by the button with the hand or clicking into the green circle.

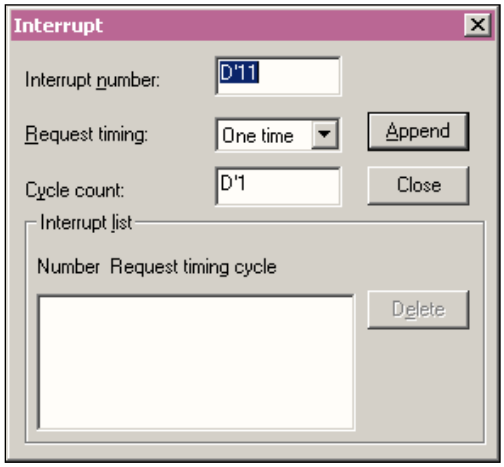
9.5 I/O-Port Simulator Settings

Because the Simulator is a core simulator only, it is not possible to simulate the resources as e.g. USART or timers. But it is possible to simulate interrupts and I/O ports. I/O ports can be simulated using a file to input or output data whenever an access is made to a special I/O address. To simulate Interrupts or I/O port functionality use the *Setup>Debug Environment >IOPort*, after the debugger is started.



9.6 Interrupt Simulator Settings

Via *Setup>Debug Environment>Interrupts* it is possible to generate interrupts at a dedicated cycle time. This can be used to debug interrupt handler routines.



9.7 Debugging with Emulation System

If you use an emulation system, please refer to the application note:

AN205547: F2MC-16FX Family, Emulator System MB2198-01 Installation Guide.

10 Related Application Notes and Contact

Overview of Related Application Notes and Contact to Cypress

10.1 Manuals and Application Notes

- AN205547: F2MC-16FX Family, Emulator System MB2198-01 Installation Guide
  Installing the 16FX emulation system
- AN205555: F2MC-16FX Family, Emulator System MB2198
  How to use the emulator MB2198-01 together with the Softune Workbench

11 Document History

Document Title: AN204781 - F²MC-16FX Family, Softune Workbench

Document Number: 002-04781

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	01/05/2007 10/02/2007	Initial Release Updated, actual Start.asm code used
*A	5060908	MKEA	12/22/2015	Migrated Spansion Application Note from MCU-AN-300233-E-V11 to Cypress format
*B	5836321	AESATMP8	07/28/2017	Updated logo and Copyright.
*C	6037021	NOFL	01/19/2018	Updated logo. Updated links. Updated Section 8.1 Programming a Flash MCU and Section 10.1 Manuals and Applications Notes.

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

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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