User's Manual

Renesas Starter Kit for RX610

Tutorial Manual

Renesas Single-Chip Microcomputer
RX Family/RX600 Series

Renesas Electronics
www.renesas.com

Rev.1.02 2010.03

Table of Contents

Chapter 1. Preface

Cautions

Trademarks

Copyright

Website: http://www.renesas.com/

Glossary

ADC	Analog to Digital Converter	LED	Light Emitting Diode
API	Application Programming Interface	PC	Program Counter
CD	Compact Disk	ROM	Read Only Memory
CPU	Central Processing Unit	RPDL	Renesas Peripheral Driver Library
E1	'E1 for Starter Kit' Emulator	RSK	Renesas Starter Kit
E20	'E20 for Starter Kit' Emulator	SDRAM	Synchronous dynamic random access memory
HEW	High performance Embedded Workshop	USB	Universal Serial Bus
LCD	Liquid Crystal Display		

Chapter 2. Introduction

This manual is designed to answer, in tutorial form, the most common questions asked about using a Renesas Starter Kit (RSK): The tutorials help explain the following:

- How do I compile, link, download, and run a simple program on the RSK?
- How do I build an embedded application?
- How do I use Renesas' tools?

The project generator will create a tutorial project with two selectable build configurations

- 'Debug' is a project built with the debugger support included.
- 'Release' build demonstrating code suitable for release in a product.

Files referred to in this manual are installed using the project generator as you work through the tutorials. The tutorial examples in this manual assume that installation procedures described in the RSK Quick Start Guide have been completed. Please refer to the Quick Start Guide for details of preparing the configuration.

NOTE: These tutorials are designed to show you how to use the RSK and are not intended as a comprehensive introduction to the High performance Embedded Workshop (HEW) debugger or the compiler toolchains or E1 Emulator – please consult the relevant user manuals for more in-depth information.

Chapter 3. Tutorial Project Workspace

The workspace includes all of the files for two build configurations. The tutorial code is common to both the Debug and the Release build configurations. The tutorial is designed to show how code can be written, debugged then downloaded without the debug monitor in a 'Release' situation.

The build configuration menu in High-performance Embedded Workshop (HEW) allows the project to be configured such that certain files may be excluded from each of the build configurations. This allows the inclusion of the debug monitor within the Debug build, and its exclusion in the Release build. Contents of common C files are controlled with defines set up in the build configuration options and `#ifdef` statements within the source files.

Maintaining only one set of project files means that projects are more controllable.

Chapter 4. Project Workspace

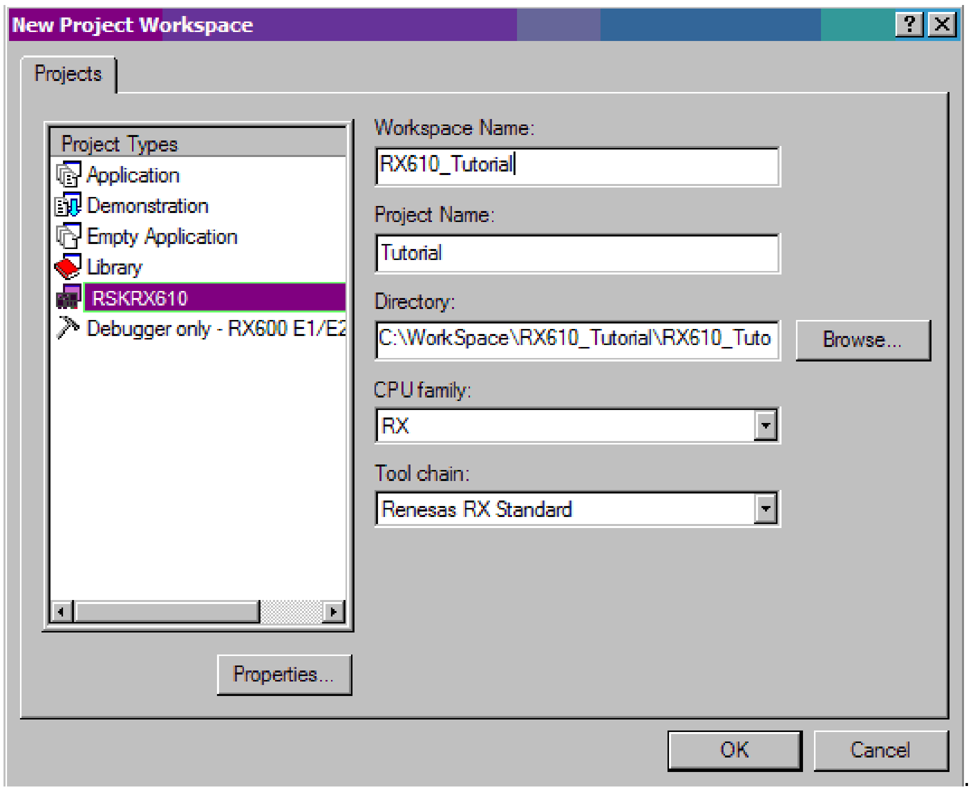
4.1. Introduction

High-performance Embedded Workshop is an integrated development tool that allows the user to write, compile, program and debug a software project on any of the Renesas Microcontrollers. High-performance Embedded Workshop will have been installed during the installation of the software support for the Renesas Starter Kit product. This manual will describe the stages required to create and debug the supplied tutorial code.

4.2. Starting HEW and Connecting the E1 Debugger

To look at the program, start High-performance Embedded Workshop from the Windows Start Menu.

Open a new tutorial workspace from the [File -> New Workspace…] menu or select 'Create a new project workspace' when presented with the 'Welcome!' dialog



Workspace Name: RX610_Tutorial

Project Name: Tutorial

Directory: C:\WorkSpace\RX610_Tutorial\RX610_Tuto

CPU family: RX

Tool chain: Renesas RX Standard

The example above shows the New Project Workspace dialog with the RSKRX610 selected.

- Select the RX CPU family and 'Renesas RX Standard' toolchain.
- Select the 'RSKRX610' project type from the left-hand projects list.
- Enter a name for the workspace – all your files will be stored under a directory with this name.
- The project name field will be pre-filled to match the workspace name above, but this name may be changed manually.

  Note: High-performance Embedded Workshop allows you to add multiple projects to a workspace. You may add the sample code projects later so you may wish to choose a suitable name for the tutorial project now.
- Click <OK> to start the Renesas Starter Kit Project Generator wizard.

The next dialog presents the three types of example project available:

1. **Tutorial:** this is the one of interest at this time – the code is explained later in this manual.
2. Sample Code: This provides examples for using various peripherals. If you select this and click <Next> it will open a new dialog, allowing the selection of many code examples for the peripheral modules of the device.
3. Application: where the debugger is configured but there is no program code. This project is suitable for the user to add code without having to configure the debugger.

The project generator wizard will display a confirmation dialog. Press <OK> to create the project and insert the necessary files.

A tree showing all the files in this project will appear in High-performance Embedded Workshop.

- To view the file 'main.c', double click on the file in the Workspace window. A new window will open showing the code.

4.3. Build Configurations and Debug Sessions

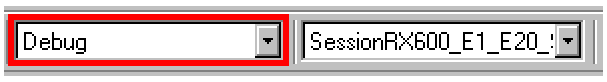
The workspace that has been created contains two build configurations and two debug sessions. The Build Configuration allows the same project to be built but with different compiler options. The options available to the user are described fully in the High-performance Embedded Workshop Manual.

4.3.1. Build Configuration

The build configurations are selected from the left hand drop down list on the tool bar. The options available are Debug and Release. The debug build is configured for use with the debugger. The Release build is configured for final ROM-able code.

A common difference between the two builds may be the optimisation settings. With Optimisation turned on the Debugger may seem to execute code in an unexpected order. To assist in debugging it is often helpful to turn optimisation off on the code being debugged.

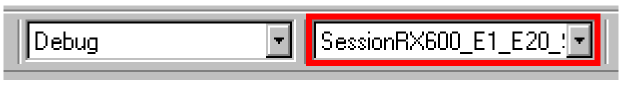
- Select the 'Debug' Build Configuration.



4.3.2. Debug Session

The debug sessions are selected from the right hand drop down list on the tool bar. The options vary between Renesas Starter Kit types however one will always start Debug and include the type of debug interface. The alternate selection will be 'SessionRX600_E1_E20_SYSTEM'. The purpose of the debug sessions is to allow the use of different debugger tools or different debugger settings on the same project.

- Select the session 'SessionRX600_E1_E20_SYSTEM'



Chapter 5. Building the Tutorial Program

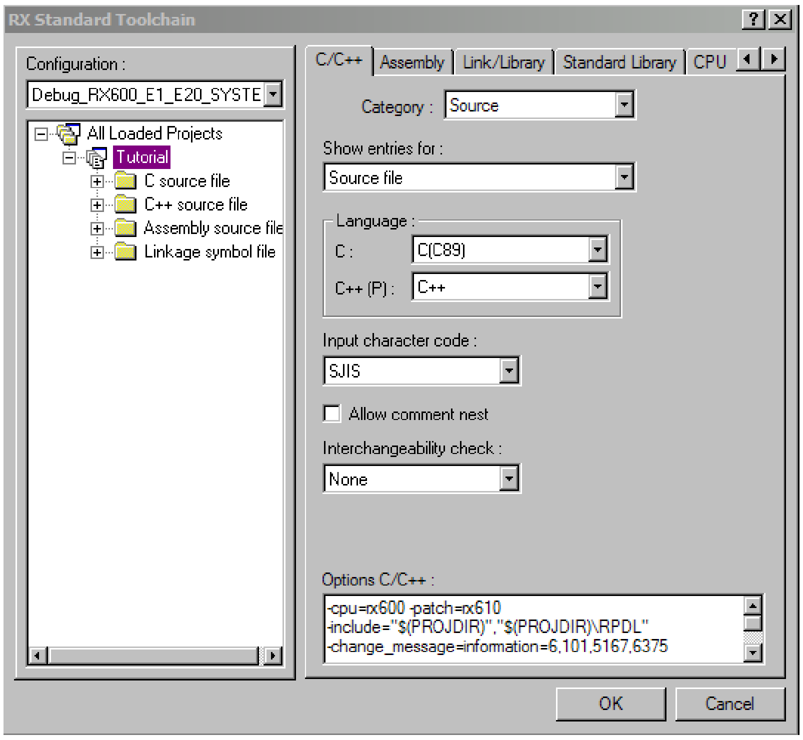
The tutorial project build settings have been pre-configured in the tool-chain options. To view the tool chain options select the 'Build' menu item and the relevant tool-chain. This should be the first option(s) on the drop down menu.

The dialog that is displayed will be specific to the tool-chain selected.

The Configuration pane on the left hand side will exist on all the tool-chain options. It is important when changing any setting to be aware of the current configuration that is being modified. If you wish to modify multiple or all build configurations this is possible by selecting 'All' or 'Multiple' from the 'Configuration' drop down list.

- Review the options on each of the tabs and 'Category' drop down lists to be aware of the options available. Note: For the purposes of the tutorial, leave all options at default.

When complete close the dialog box by clicking <OK>.



Configuration : Debug_RX600_E1_E20_SYSTE

All Loaded Projects
- Tutorial
  - C source file
  - C++ source file
  - Assembly source file
  - Linkage symbol file

Category : Source

Show entries for : Source file

Language :
C : C(C89)
C++ (P) : C++

Input character code : SJIS

☐ Allow comment nest

Interchangeability check : None

Options C/C++ :

```
-cpu=rx600 -patch=rx610
-include="$(PROJDIR)","$(PROJDIR)\RPDL"
-change_message=information=6,101,5167,6375
```

5.1. Building Code

There is a choice of three shortcuts available for building the project.

1. Selecting the 'Build All' tool bar button.

   This will build everything in the project that has not been excluded from the build. This includes the standard library.

2. Selecting the 'Build' tool bar button.

   This will build all files that have changed since the last build. The standard library will not be built unless an option has been changed.

3. Pressing <F7>.

   This is equivalent to pressing the 'Build' button described above.

- Build the project now by pressing <F7> or pressing one of the build icons as shown above.
  During the build each stage will be reported in the Output Window.
  The build will complete with an indication of any errors and warnings encountered during the build.

5.2. Connecting the debugger

For this tutorial it is not necessary to provide an external power supply to the board. The power will be obtained from the USB port. Please be aware that if you have too many devices connected to your USB port it may be shut down by Windows. If this happens remove some devices and try again. Alternatively provide an external power source taking care to ensure the correct polarity and voltage.

The Quick Start Guide provided with the Renesas Starter Kit board gives detailed instructions on how to connect the E1 to the host computer. The following assumes that the steps in the Quick Start Guide have been followed and the E1 drivers have been installed.

- Fit the LCD module to LCD connector on the board, so it lies above J4. Ensure all the pins of the connector are correctly inserted in the socket.
- Connect the E1 debugger to a free USB port on your computer.
- Connect the E1 Debugger to the target hardware ensuring that it is plugged into the connector marked ‘E1’.
- If supplying external power to the board this can be turned on now.

5.3. Connecting to the target with the E1

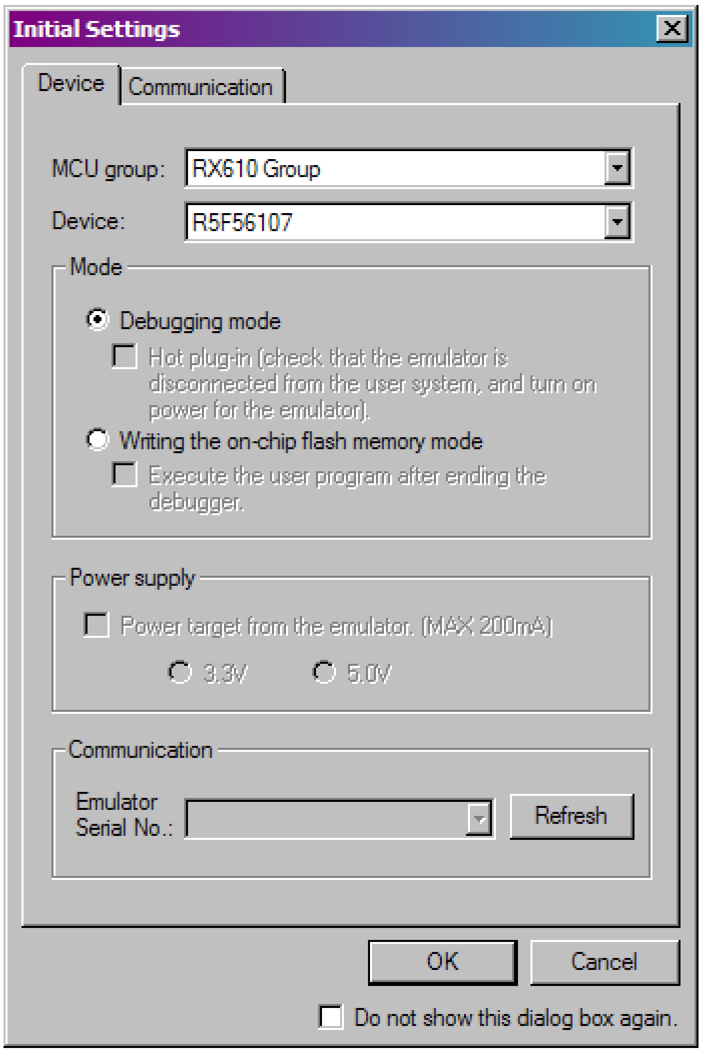
This section will take you through the process of connecting to the device, programming the Flash and executing the code.

- Select the ‘SessionRX600_E1_E20_SYSTEM’ debug session.
- Click the <Connect> button on the debug toolbar.

Please note that the “Emulator Mode” wizard shown here will only appear the FIRST time you connect to the target within a project. On subsequent connections the “Emulator Setting” dialog will appear, please choose the same options to connect.

- Select the correct MCU Group type (RX610 Group illustrated).
- Select the correct device type (R5F56107 illustrated).
- Select “Debugging mode”.
- If the E1 is to provide power to the CPU board, select “Power Target from Emulator” and choose the “3.3V” option.

Otherwise connect a 5V centre positive supply.


Initial Settings

Device | Communication

MCU group: RX610 Group

Device: R5F56107

Mode

[✓ Debugging mode]

☐ Hot plug-in (check that the emulator is disconnected from the user system, and turn on power for the emulator).

Writing the on-chip flash memory mode

☐ Execute the user program after ending the debugger.

Power supply

☐ Power target from the emulator. (MAX 200mA)

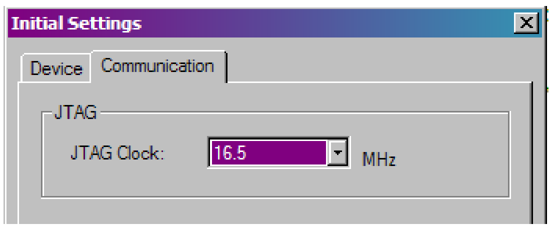
3.3V 5.0V

Communication

Emulator Serial No.:

☐ Do not show this dialog box again.

- Click the ‘Communication’ tab and ensure the JTAG Clock is set to 16.5MHz. Once these settings have been confirmed, click the <OK> button to continue.

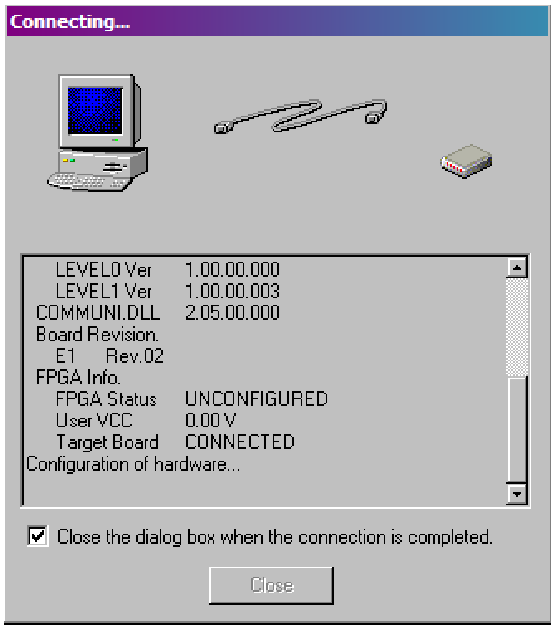


Initial Settings

JTAG

JTAG Clock: 16.5 MHz

- The Flash Memory write program will downloaded to the target.



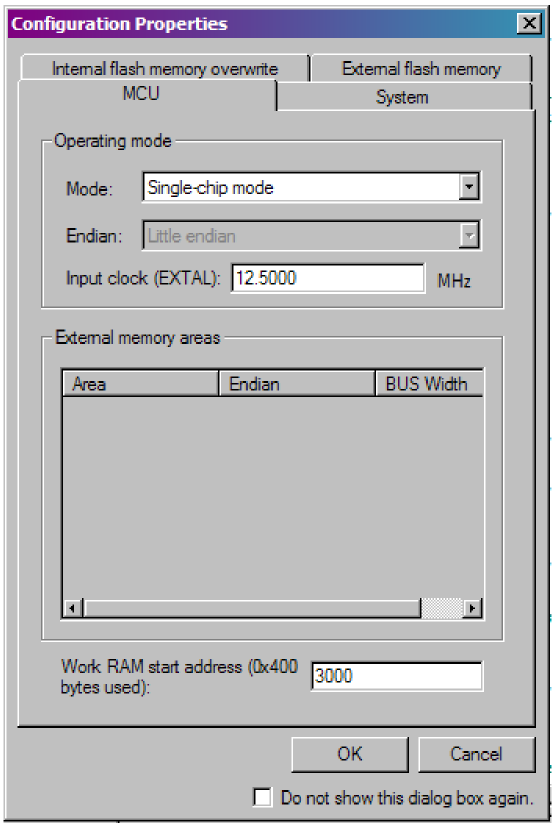
Connecting...

```
    LEVEL0 Ver      1.00.00.000
    LEVEL1 Ver      1.00.00.003
  COMMUNI.DLL       2.05.00.000
  Board Revision.
    E1      Rev.02
  FPGA Info.
    FPGA Status     UNCONFIGURED
    User VCC        0.00 V
    Target Board    CONNECTED
Configuration of hardware...
```

☑ Close the dialog box when the connection is completed.

- Once the debugger has connected, the configuration properties dialog will appear. Keep the default settings, and click <OK>.
- The output window in High-performance Workshop will show ‘Connected’.

Note: The connection to the target will active the debugger buttons on the High-performance Embedded Workshop toolbar.



Configuration Properties

Operating mode

Mode: Single-chip mode

Endian: Little endian

Input clock (EXTAL): 12.5000 MHz

External memory areas

Area	Endian	BUS Width

Work RAM start address (0x400 bytes used): 3000

☐ Do not show this dialog box again.

Now is a good time to save the High-performance Embedded Workshop session.

- Select ‘File’ | ‘Save Session’.

If you have changed any workspace settings now is a good time to save the workspace.

- Select ‘File’ | ‘Save Workspace’.

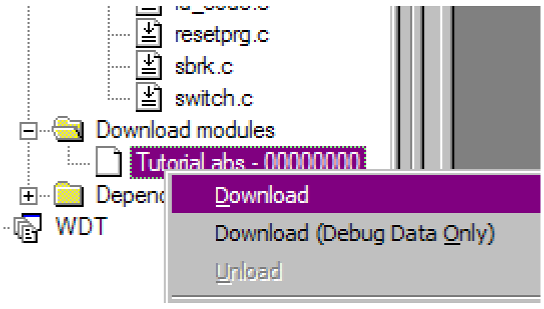
Chapter 6. Downloading and Running the Tutorial

6.1. Downloading the Program Code

Now the code has been built in HEW it needs to be downloaded to the RSK.

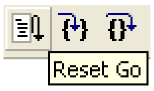
Now that you are connected to the target you should see an additional category in the workspace view called 'Download Modules'

- Right click on the download module listed and select 'Download'
- On completion the debugger and code are ready to be executed



6.2. Running the Tutorial

Once the program has been downloaded onto the RSK device, the program can be executed. Click the 'Reset Go' button to begin the program. It is recommended that you run through the program once first, and then continue to the review section.



Chapter 7. Reviewing the Tutorial Program

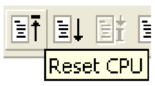
This section will look at each section of the tutorial code, how it works, and how it could be altered to be implemented into more complex code.

It is recommended that a copy of the RX610 API Manual is made available, as the tutorial program uses RPDL and it is outside the scope of this manual to fully document the API system.

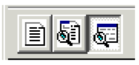
7.1. Program Initialisation

Before the main program can run, the Microcontroller must be configured. The following parts of the tutorial program are used exclusively for initialising the RSK device so that the main function can execute correctly. The initialisation code is run every time the device is reset via the reset switch or from a power reboot.

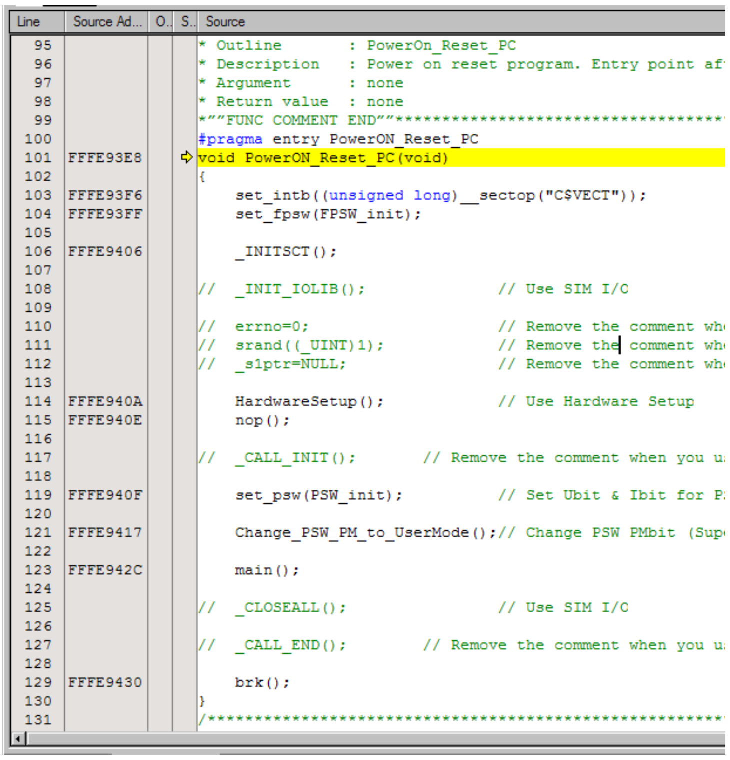
Ensuring the tutorial program has been downloaded onto the RX610; press the ‘Reset CPU’ button on the Debug Tool Bar.



- The File window will open the Tutorial code at the entry point. An arrow and a yellow highlight marks the current position of the program counter.
- Use these buttons to switch between ‘source, disassembly and mixed modes’.

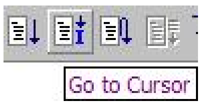


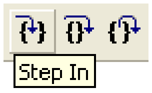
Ensure the view is switched to ‘source’ before continuing.

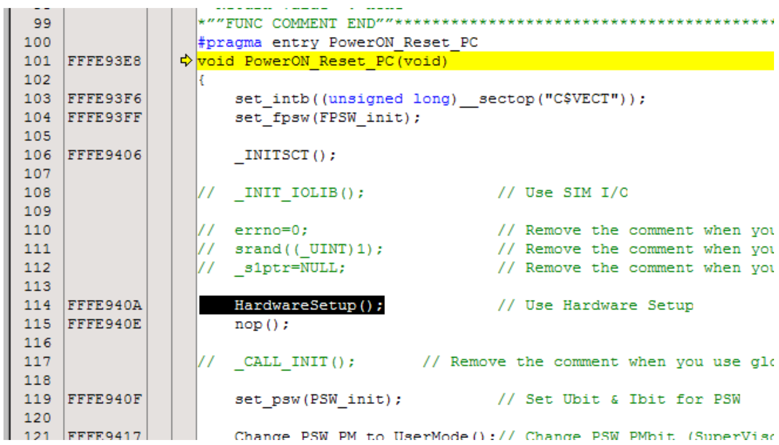


```
          * Outline       : PowerOn_Reset_PC
          * Description   : Power on reset program. Entry point af
          * Argument      : none
          * Return value  : none
          *""FUNC COMMENT END""*************************************
          #pragma entry PowerON_Reset_PC
FFFE93E8  void PowerON_Reset_PC(void)
          {
FFFE93F6      set_intb((unsigned long)__sectop("C$VECT"));
FFFE93FF      set_fpsw(FPSW_init);

FFFE9406      _INITSCT();

          //  _INIT_IOLIB();                 // Use SIM I/O

          //  errno=0;                       // Remove the comment wh
          //  srand((_UINT)1);               // Remove the comment wh
          //  _s1ptr=NULL;                   // Remove the comment wh

FFFE940A      HardwareSetup();               // Use Hardware Setup
FFFE940E      nop();

          //  _CALL_INIT();         // Remove the comment when you u

FFFE940F      set_psw(PSW_init);             // Set Ubit & Ibit for P

FFFE9417      Change_PSW_PM_to_UserMode();// Change PSW PMbit (Sup

FFFE942C      main();

          //  _CLOSEALL();                   // Use SIM I/O

          //  _CALL_END();          // Remove the comment when you u

FFFE9430      brk();
          }
          /*********************************************************
```

- Highlight the 'HardwareSetup()' function call by left clicking to the right of the text, and holding the left mouse button and dragging over to the left of it and releasing the left mouse button.
- Click the 'Go to Cursor' button to run the program up to this point. Then click 'Step In'

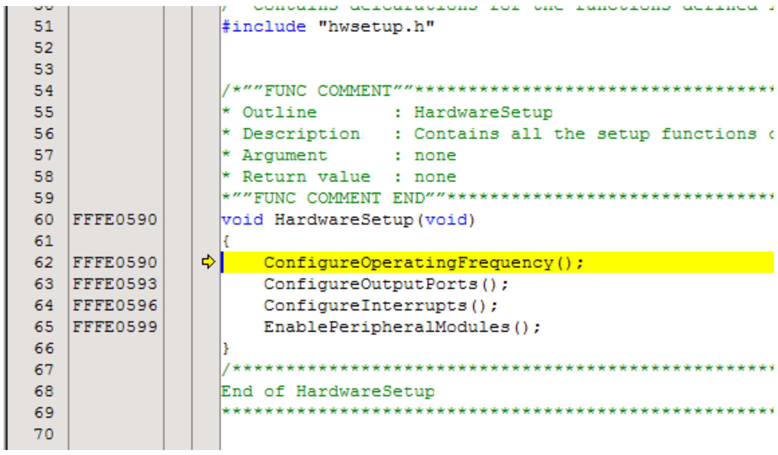






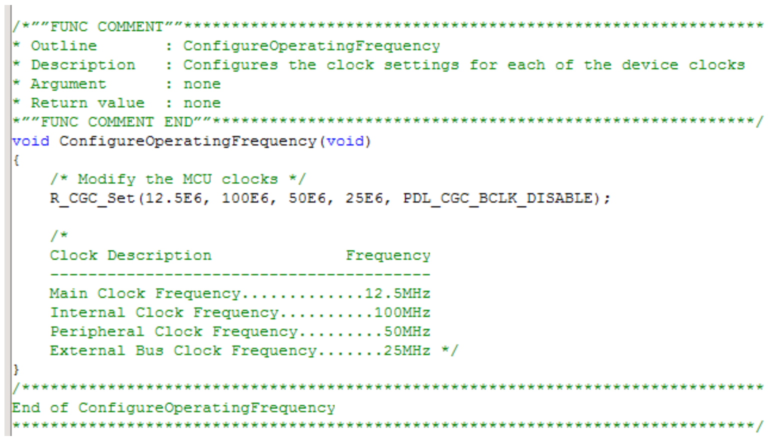
```
          *""FUNC COMMENT END""****************************************
          #pragma entry PowerON_Reset_PC
FFFE93E8  void PowerON_Reset_PC(void)
          {
FFFE93F6      set_intb((unsigned long)__sectop("C$VECT"));
FFFE93FF      set_fpsw(FPSW_init);

FFFE9406      _INITSCT();

          //  _INIT_IOLIB();                // Use SIM I/O

          //  errno=0;                      // Remove the comment when yo
          //  srand((_UINT)1);              // Remove the comment when yo
          //  _s1ptr=NULL;                  // Remove the comment when yo

FFFE940A      HardwareSetup();              // Use Hardware Setup
FFFE940E      nop();

          //  _CALL_INIT();         // Remove the comment when you use gl

FFFE940F      set_psw(PSW_init);            // Set Ubit & Ibit for PSW

FFFE9417      Change_PSW_PM_to_UserMode();// Change PSW PMbit (SuperVis
```

- The program counter should now move to the HardwareSetup function definition. This function groups together several key functions that are used to ensure the device is setup correctly before the main program is executed.



```
          #include "hwsetup.h"


          /*""FUNC COMMENT""******************************************
          * Outline        : HardwareSetup
          * Description    : Contains all the setup functions c
          * Argument       : none
          * Return value   : none
          *""FUNC COMMENT END""**************************************
FFFE0590  void HardwareSetup(void)
          {
FFFE0590      ConfigureOperatingFrequency();
FFFE0593      ConfigureOutputPorts();
FFFE0596      ConfigureInterrupts();
FFFE0599      EnablePeripheralModules();
          }
          /*********************************************************
          End of HardwareSetup
          **********************************************************
```

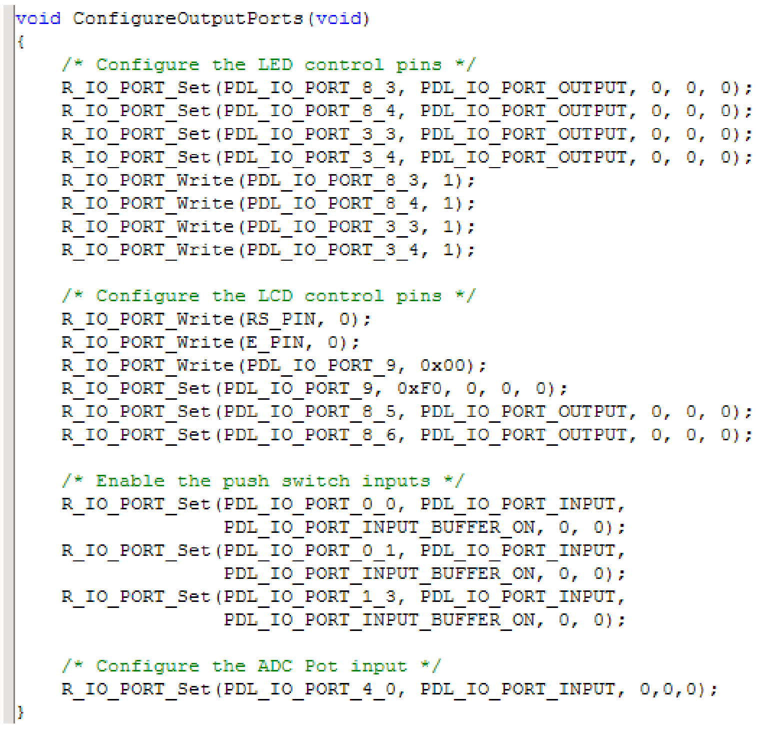
- You should be viewing the hwsetup.c file currently. Scroll down and review the different setup functions used.
- The ConfigureOperatingFrequency function is greatly simplified here, with the use of the 'R_CGC_Set' API function, which configures all the device's clocks in a single function.



```
/*""FUNC COMMENT""*****************************************************************
* Outline        : ConfigureOperatingFrequency
* Description    : Configures the clock settings for each of the device clocks
* Argument       : none
* Return value   : none
*""FUNC COMMENT END""*************************************************************/
void ConfigureOperatingFrequency(void)
{
    /* Modify the MCU clocks */
    R_CGC_Set(12.5E6, 100E6, 50E6, 25E6, PDL_CGC_BCLK_DISABLE);

    /*
    Clock Description                   Frequency
    ----------------------------------------
    Main Clock Frequency.............12.5MHz
    Internal Clock Frequency..........100MHz
    Peripheral Clock Frequency.........50MHz
    External Bus Clock Frequency.......25MHz */
}
/*********************************************************************************
End of ConfigureOperatingFrequency
*********************************************************************************/
```

- The ‘ConfigureOutputPorts’ function uses the ‘R_IO_PORT_Set’ API function to simplifies setting the data direction and port control registers. The first set of commands set all pins to be outputs. This is to ensure each pin is in a defined state before the main program runs, since after Reset the data direction register is undefined.

- The next set of commands individually sets different pins to be inputs. This section can be changed as required, and since a large number of the pins on the RX610 are multiplexed with many other functions – different port directions will be required for different programs. The current settings are to set default input intended pins as inputs, for correct operation of the tutorial program.

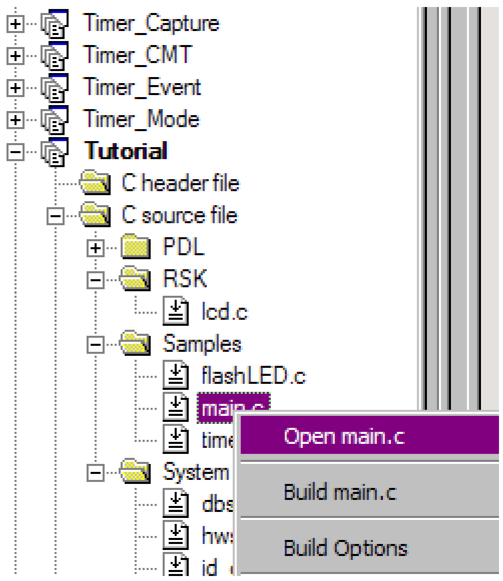


```
void ConfigureOutputPorts(void)
{
    /* Configure the LED control pins */
    R_IO_PORT_Set(PDL_IO_PORT_8_3, PDL_IO_PORT_OUTPUT, 0, 0, 0);
    R_IO_PORT_Set(PDL_IO_PORT_8_4, PDL_IO_PORT_OUTPUT, 0, 0, 0);
    R_IO_PORT_Set(PDL_IO_PORT_3_3, PDL_IO_PORT_OUTPUT, 0, 0, 0);
    R_IO_PORT_Set(PDL_IO_PORT_3_4, PDL_IO_PORT_OUTPUT, 0, 0, 0);
    R_IO_PORT_Write(PDL_IO_PORT_8_3, 1);
    R_IO_PORT_Write(PDL_IO_PORT_8_4, 1);
    R_IO_PORT_Write(PDL_IO_PORT_3_3, 1);
    R_IO_PORT_Write(PDL_IO_PORT_3_4, 1);

    /* Configure the LCD control pins */
    R_IO_PORT_Write(RS_PIN, 0);
    R_IO_PORT_Write(E_PIN, 0);
    R_IO_PORT_Write(PDL_IO_PORT_9, 0x00);
    R_IO_PORT_Set(PDL_IO_PORT_9, 0xF0, 0, 0, 0);
    R_IO_PORT_Set(PDL_IO_PORT_8_5, PDL_IO_PORT_OUTPUT, 0, 0, 0);
    R_IO_PORT_Set(PDL_IO_PORT_8_6, PDL_IO_PORT_OUTPUT, 0, 0, 0);

    /* Enable the push switch inputs */
    R_IO_PORT_Set(PDL_IO_PORT_0_0, PDL_IO_PORT_INPUT,
                  PDL_IO_PORT_INPUT_BUFFER_ON, 0, 0);
    R_IO_PORT_Set(PDL_IO_PORT_0_1, PDL_IO_PORT_INPUT,
                  PDL_IO_PORT_INPUT_BUFFER_ON, 0, 0);
    R_IO_PORT_Set(PDL_IO_PORT_1_3, PDL_IO_PORT_INPUT,
                  PDL_IO_PORT_INPUT_BUFFER_ON, 0, 0);

    /* Configure the ADC Pot input */
    R_IO_PORT_Set(PDL_IO_PORT_4_0, PDL_IO_PORT_INPUT, 0,0,0);
}
```

7.2. Main Functions

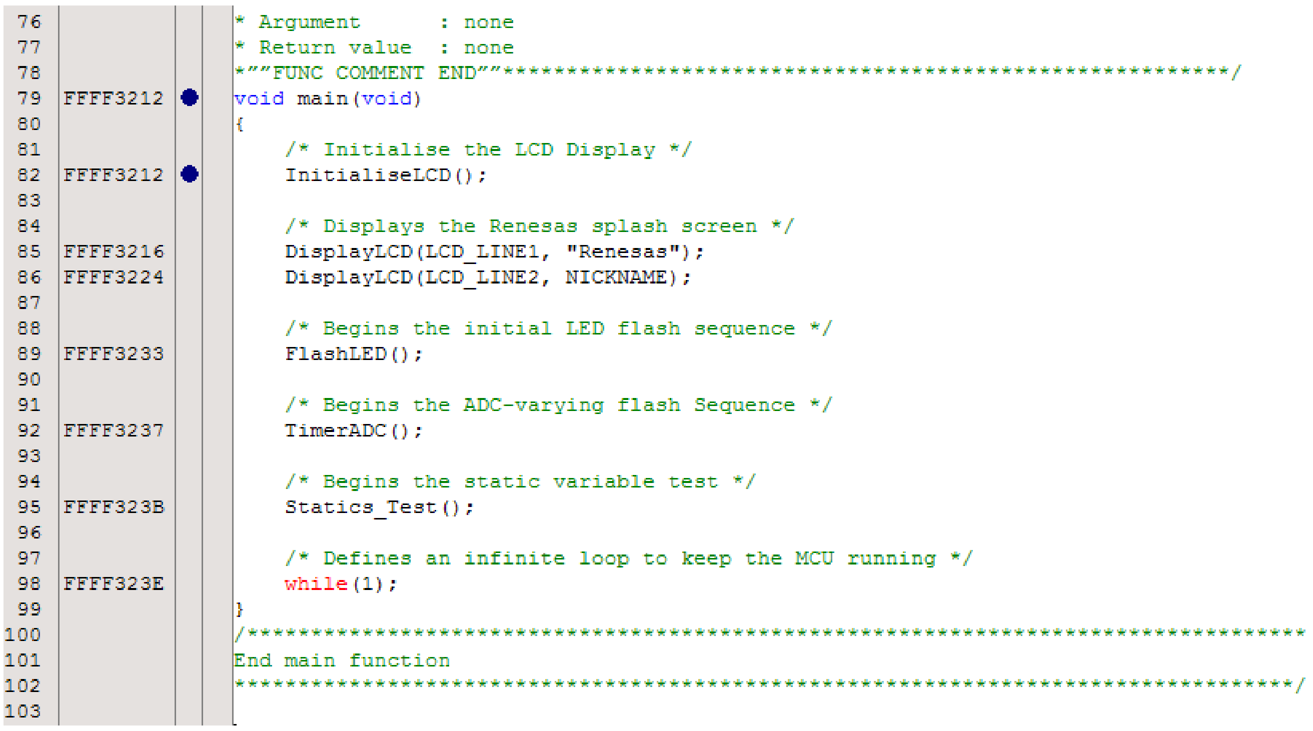
This section will look at the program code called from with the main() function, and how it works.

- Find the main.c file from the file tree on the left hand side, then right click it and select 'Open main.c'



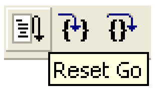
- Place an event at the call to main(); by double clicking in the **On-Chip Breakpoint** column next to the line to stop at. (NB. Two event points will appear because they share the same source address).

Note: The E1 emulator features advanced logic-based event point trigger system, and full instruction on its use is outside the scope of this tutorial. For further details, please refer to the *RX Family E1/E20 Emulator User's Manual*

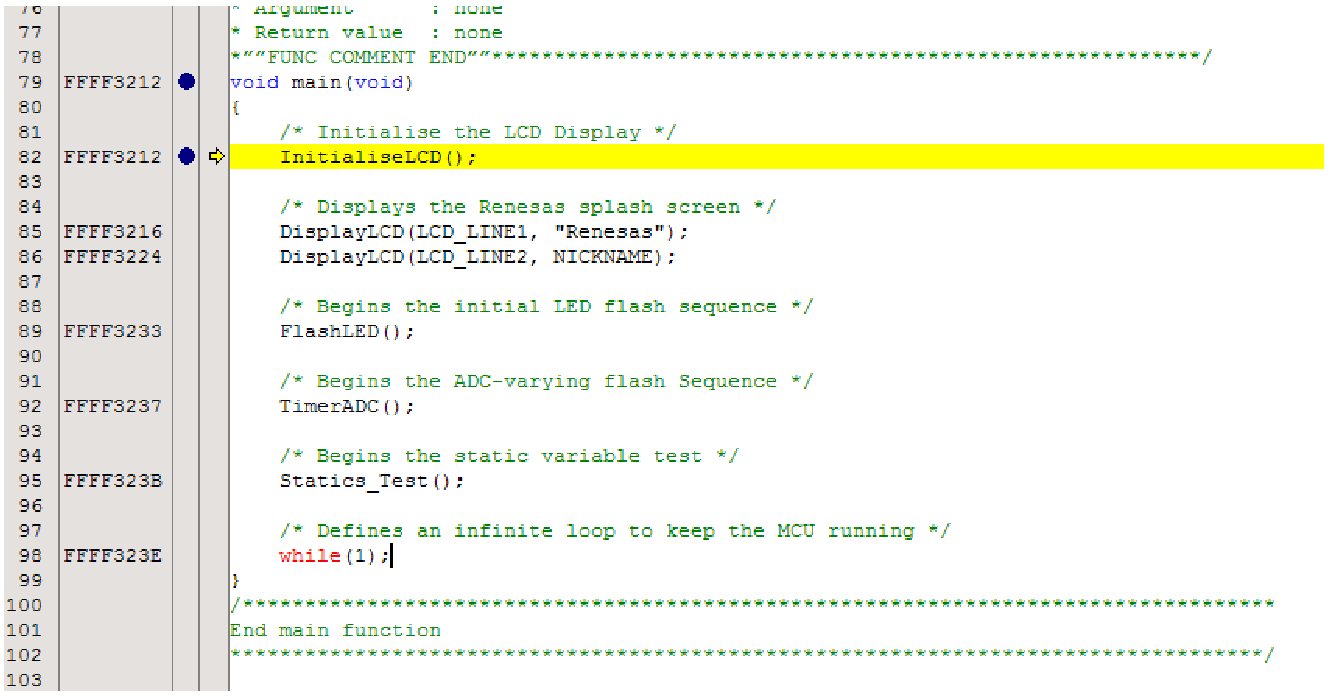


```
                 * Argument      : none
                 * Return value  : none
                 *""FUNC COMMENT END""*****************************************************************/
FFFF3212  ●      void main(void)
                 {
                     /* Initialise the LCD Display */
FFFF3212  ●          InitialiseLCD();

                     /* Displays the Renesas splash screen */
FFFF3216             DisplayLCD(LCD_LINE1, "Renesas");
FFFF3224             DisplayLCD(LCD_LINE2, NICKNAME);

                     /* Begins the initial LED flash sequence */
FFFF3233             FlashLED();

                     /* Begins the ADC-varying flash Sequence */
FFFF3237             TimerADC();

                     /* Begins the static variable test */
FFFF323B             Statics_Test();

                     /* Defines an infinite loop to keep the MCU running */
FFFF323E             while(1);
                 }
                 /*******************************************************************************
                 End main function
                 *******************************************************************************/
```

- Press 'Reset Go' on the Debug Tool Bar.



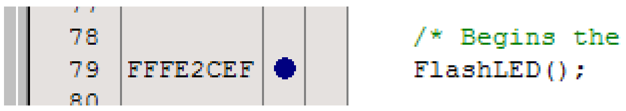
The code will execute to the event point. At this point all the device initialisation will have been completed. The code window will open 'main.c' and show the new position of the program counter.



```
         * Argument       : none
         * Return value   : none
         *""FUNC COMMENT END""*******************************************************/
FFFF3212 ● void main(void)
           {
               /* Initialise the LCD Display */
FFFF3212 ● ⇨   InitialiseLCD();

               /* Displays the Renesas splash screen */
FFFF3216       DisplayLCD(LCD_LINE1, "Renesas");
FFFF3224       DisplayLCD(LCD_LINE2, NICKNAME);

               /* Begins the initial LED flash sequence */
FFFF3233       FlashLED();

               /* Begins the ADC-varying flash Sequence */
FFFF3237       TimerADC();

               /* Begins the static variable test */
FFFF323B       Statics_Test();

               /* Defines an infinite loop to keep the MCU running */
FFFF323E       while(1);
           }
           /***************************************************************************
           End main function
           ***************************************************************************/
```

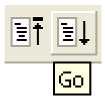
Support for the LCD display is included in the tutorial code. We do not need to be concerned about the details of the LCD interface – except that the interface is write-only and so is not affected if the LCD display is attached or not.

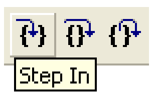
- Insert a event on the 'FlashLED();' function call in the file main.c.



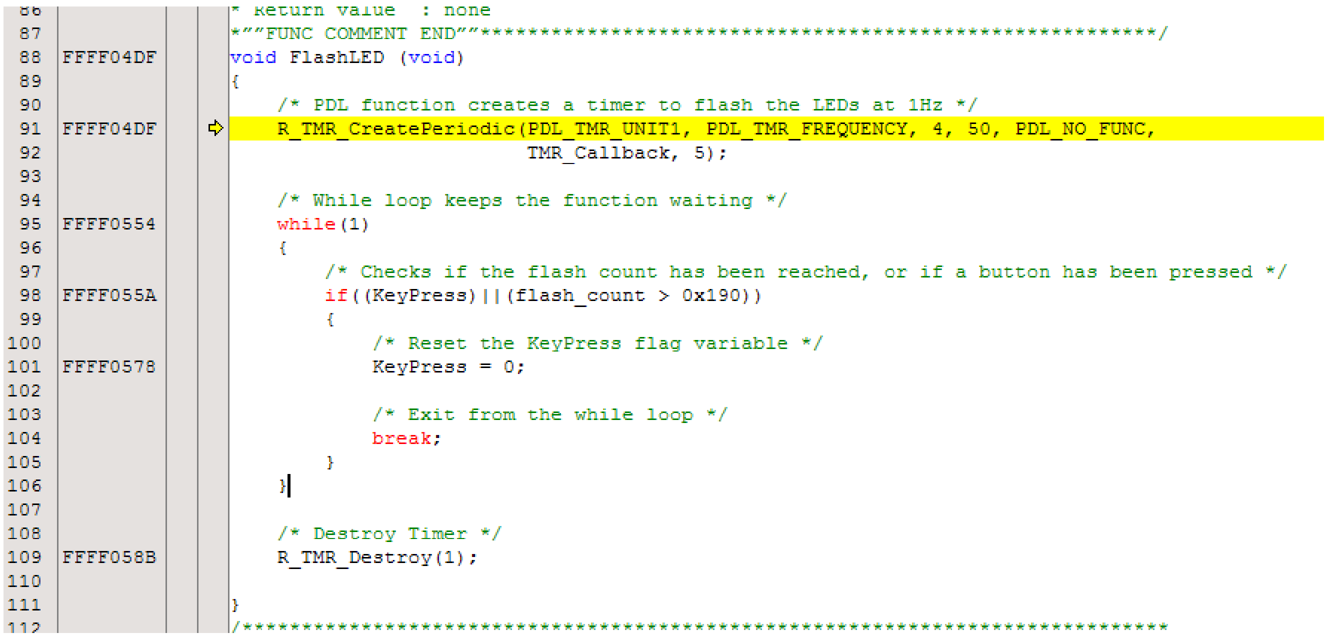
```
FFFE2CEF ●     /* Begins the
               FlashLED();
```

- Press 'Go' to run the program up to the event, then press 'Step In', to move the program counter to the beginning of the FlashLED() function definition.



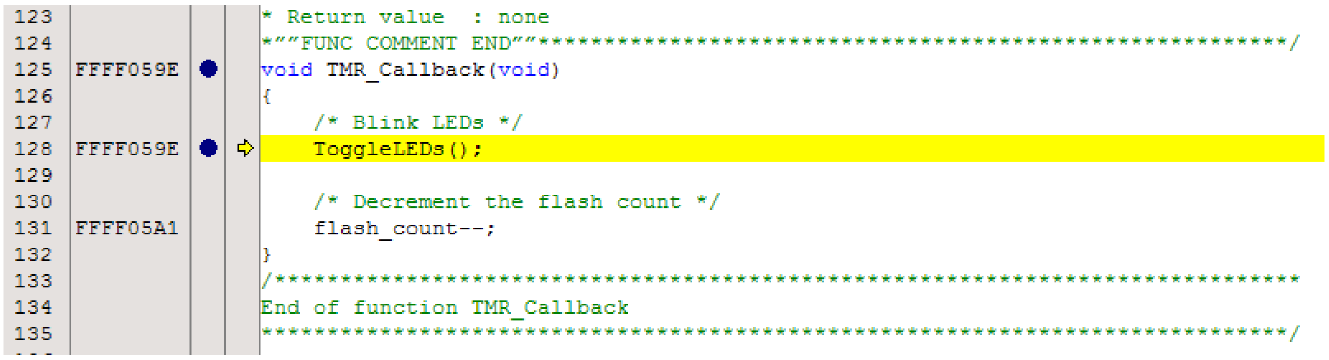


- Review the FlashLED function. The function uses the 'R_TMR_CreatePeriodic' API function to set a timer with a callback function. Every period duration of the timer will call the callback function 'TMR_callback' at interrupt level.



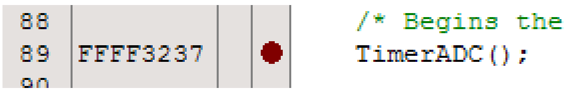
```
* Return value  : none
*""FUNC COMMENT END""*****************************************************************/
void FlashLED (void)
{
    /* PDL function creates a timer to flash the LEDs at 1Hz */
    R_TMR_CreatePeriodic(PDL_TMR_UNIT1, PDL_TMR_FREQUENCY, 4, 50, PDL_NO_FUNC,
                         TMR_Callback, 5);

    /* While loop keeps the function waiting */
    while(1)
    {
        /* Checks if the flash count has been reached, or if a button has been pressed */
        if((KeyPress)||(flash_count > 0x190))
        {
            /* Reset the KeyPress flag variable */
            KeyPress = 0;

            /* Exit from the while loop */
            break;
        }
    }

    /* Destroy Timer */
    R_TMR_Destroy(1);

}
/*******************************************************************************
```

- Insert an event point on the TMR_Callback(void) function call, in the FlashLED.c file, then press ‘Go’ to resume the software. When the timer reaches the end of the period, an interrupt will call the TMR_callback function and the program counter will stop at the event point.
- The TMR_Callback function executes the ‘ToggleLEDs’ function, which simply inverts the output of the LED pins. The callback function also decrements the ‘flash_count’ variable, which is used to ensure the FlashLED function only flashes 200 times.
- Remove the event point before continuing, by double clicking the blue dot symbol.



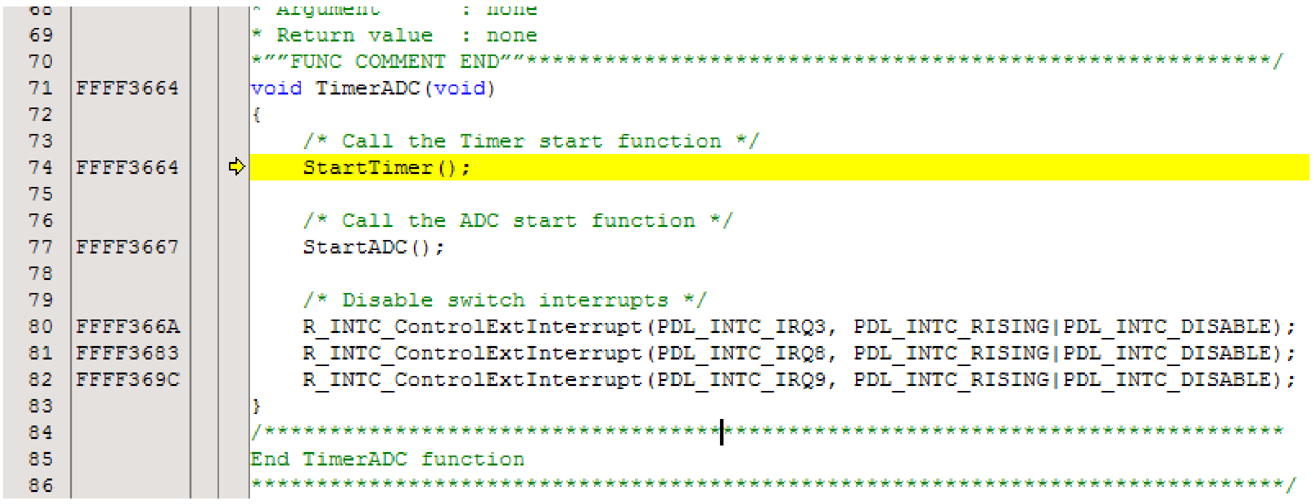
```
* Return value  : none
*""FUNC COMMENT END""*****************************************************************/
void TMR_Callback(void)
{
    /* Blink LEDs */
    ToggleLEDs();

    /* Decrement the flash count */
    flash_count--;
}
/*******************************************************************************
End of function TMR_Callback
*******************************************************************************/
```

- Return to the main.c file, and insert a software break point at the TimerADC() function call, by double clicking in the S/W **Breakpoints** column, in line with the function.



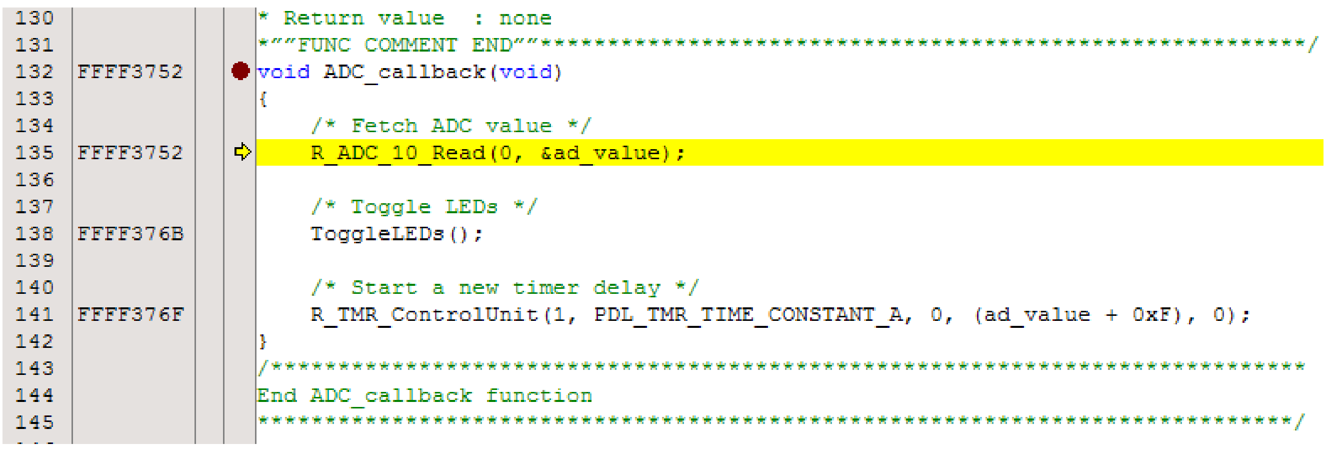
```
/* Begins the
TimerADC();
```

- Click ‘Go’ or press F5 to resume the program, then push switch 1 to proceed. The program should halt at the breakpoint set on the TimerADC() function call. Press ‘Step In’ to single-step into the TimerADC function.
- The TimerADC function configures both the timer and the ADC peripheral, so that every timer period triggers an AD conversion, and AD conversion completion triggers the callback function ADC_callback().



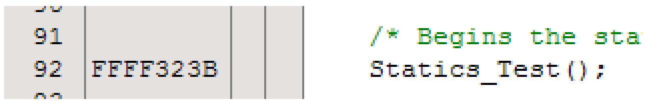
```
          * Argument       : none
          * Return value   : none
          *""FUNC COMMENT END""*****************************************************************/
FFFF3664  void TimerADC(void)
          {
              /* Call the Timer start function */
FFFF3664      StartTimer();

              /* Call the ADC start function */
FFFF3667      StartADC();

              /* Disable switch interrupts */
FFFF366A      R_INTC_ControlExtInterrupt(PDL_INTC_IRQ3, PDL_INTC_RISING|PDL_INTC_DISABLE);
FFFF3683      R_INTC_ControlExtInterrupt(PDL_INTC_IRQ8, PDL_INTC_RISING|PDL_INTC_DISABLE);
FFFF369C      R_INTC_ControlExtInterrupt(PDL_INTC_IRQ9, PDL_INTC_RISING|PDL_INTC_DISABLE);
          }
          /*******************************************************************************
          End TimerADC function
          *******************************************************************************/
```

- Scroll down to find the ADC_callback function, and insert a breakpoint at the start of the function. Right click the variable ‘ad_value’ in the function ‘R_ADC_10_Read’, and select instant watch, then click add to the dialog box that follows.
- Click ‘Go’ or press F5 to resume the program. Once the timer period ends, the function will be called and the program counter will halt at the breakpoint.
- The ADC_callback function first fetches the ADC value, and then calls the ToggleLEDs function to toggle the output to the LEDs. The callback function then reconfigures the timer period by the value of the ADC, thus changing the frequency of the flashing lights.
- Press F5 again to allow the program to complete a loop. Check the value of the watched variable, ad_value.
- Remove the breakpoint at the ADC_callback function before continuing.



```
          * Return value   : none
          *""FUNC COMMENT END""*****************************************************************/
FFFF3752  void ADC_callback(void)
          {
              /* Fetch ADC value */
FFFF3752      R_ADC_10_Read(0, &ad_value);

              /* Toggle LEDs */
FFFF376B      ToggleLEDs();

              /* Start a new timer delay */
FFFF376F      R_TMR_ControlUnit(1, PDL_TMR_TIME_CONSTANT_A, 0, (ad_value + 0xF), 0);
          }
          /*******************************************************************************
          End ADC_callback function
          *******************************************************************************/
```

- Return to the main.c file, and review the main() function.
- Whilst the TimerADC functions vary the LED flash rate at interrupt level, the Statics_Test function runs at non-interrupt level.



```
                  /* Begins the sta
FFFF323B          Statics_Test();
```

- The Statics_Test function initalises a character string with the contents of a static variable; then gradually replaces it, letter by letter, with another static string.
- Click ‘Go’ or press F5 to resume the program code. You should observe the word ‘STATIC’ appear on the second LCD line, to be gradually replaced with the string ‘TESTTEST’. The program then reverts the LCD back to the original message of ‘RX610’.
- The program then enters the infinite while loop, and continues to operate the TimerADC flash interrupt service routine (ISR).
- This is the extent of the tutorial code. For further information on the RPDL function calls used in the tutorial sample, please refer to *Renesas Peripheral Driver Library User’s Manual.*

Chapter 8. Additional Information

For details on how to use High-performance Embedded Workshop (HEW), refer to the HEW manual available on the CD or from the web site.

Further information available for this product can be found on the Renesas website at: http://www.renesas.com/renesas_starter_kits

General information on Renesas Microcontrollers can be found at the following websites.

Global:	http://www.renesas.com/

Regional (English language) sites can be accessed from the Global site, or directly by going to:

Europe:	http://renesas.eu
Americas:	http://america.renesas.com
Asia:	http://sg.renesas.com

Renesas Starter Kit for RX610

Tutorial Manual

Publication Date	Rev 1.02 29.Mar.2010
Published by:	Renesas Technology Europe Ltd.
	Duke’s Meadow, Millboard Road, Bourne End
	Buckinghamshire SL8 5FH, United Kingdom

Renesas Starter Kit for RX610
Tutorial Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J1918-0102