User's Manual

32

RZ/T1 Group

Renesas Starter Kit Code Generator Tutorial Manual For e^2 studio

RENESAS MCU
RZ Family / RZ/T1 Series

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corporation without notice. Please review the latest information published by Renesas Electronics Corporation through various means, including the Renesas Electronics Corporation website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev. 1.00 Mar 2015

Disclaimer

By using this Renesas Starter Kit (RSK), the user accepts the following terms:

The RSK is not guaranteed to be error free, and the entire risk as to the results and performance of the RSK is assumed by the User. The RSK is provided by Renesas on an “as is” basis without warranty of any kind whether express or implied, including but not limited to the implied warranties of satisfactory quality, fitness for a particular purpose, title and non-infringement of intellectual property rights with regard to the RSK. Renesas expressly disclaims all such warranties. Renesas or its affiliates shall in no event be liable for any loss of profit, loss of data, loss of contract, loss of business, damage to reputation or goodwill, any economic loss, any reprogramming or recall costs (whether the foregoing losses are direct or indirect) nor shall Renesas or its affiliates be liable for any other direct or indirect special, incidental or consequential damages arising out of or in relation to the use of this RSK, even if Renesas or its affiliates have been advised of the possibility of such damages.

Precautions

The following precautions should be observed when operating any RSK product:

This Renesas Starter Kit is only intended for use in a laboratory environment under ambient temperature and humidity conditions. A safe separation distance should be used between this and any sensitive equipment. Its use outside the laboratory, classroom, study area or similar such area invalidates conformity with the protection requirements of the Electromagnetic Compatibility Directive and could lead to prosecution.

The product generates, uses, and can radiate radio frequency energy and may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment causes harmful interference to radio or television reception, which can be determined by turning the equipment off or on, you are encouraged to try to correct the interference by one or more of the following measures;

- ensure attached cables do not lie across the equipment
- reorient the receiving antenna
- increase the distance between the equipment and the receiver
- connect the equipment into an outlet on a circuit different from that which the receiver is connected
- power down the equipment when not in use
- consult the dealer or an experienced radio/TV technician for help NOTE: It is recommended that wherever possible shielded interface cables are used.

The product is potentially susceptible to certain EMC phenomena. To mitigate against them it is recommended that the following measures be undertaken;

- The user is advised that mobile phones should not be used within 10m of the product when in use.
- The user is advised to take ESD precautions when handling the equipment.

The Renesas Starter Kit does not represent an ideal reference design for an end product and does not fulfil the regulatory standards for an end product.

How to Use This Manual

1. Purpose and Target Readers

This manual is designed to provide the user with an understanding of how to use Code Generator for RZ together with the e^2 studio IDE to create a working project for the RSK platform. It is intended for users designing sample code on the RSK platform, using the many different incorporated peripheral devices.

The manual comprises of step-by-step instructions to generate code and import it into e^2 studio, but does not intend to be a complete guide to software development on the RSK platform. Further details regarding operating the RZT1 microcontroller may be found in the Hardware Manual and within the provided sample code.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.

The following documents apply to the RZT1 Group. Make sure to refer to the latest versions of these documents. The newest versions of the documents listed may be obtained from the Renesas Electronics Web site.

Document Type	Description	Document Title	Document No.
User's Manual	Describes the technical details of the RSK hardware.	RSK+RZT1 User's Manual	R20UT3242EG
Tutorial	Provides a guide to setting up RSK environment, running sample code and debugging programs.	RSK+RZT1 Tutorial Manual	R20UT3243EG
Quick Start Guide	Provides simple instructions to setup the RSK and run the first sample.	RSK+RZT1 Quick Start Guide	R20UT3244EG
Code Generator Tutorial	Provides a guide to the standalone code generation tool.	RSK+RZT1 Code Generator Tutorial Manual	R20UT3281EG
Schematics	Full detail circuit schematics of the RSK.	RSK+RZT1 Schematics	R20UT3241EG
Hardware Manual	Provides technical details of the RZ/T1 microcontroller.	RZT1 Group, User's Manual: Hardware	R01UH0483EJ
NOR Boot Loader Application Note	Describes operational details of the NOR Boot Loader Program.	RZT1 NOR Boot Loader Application Note	R01AN2470EG
QSPI Boot Loader Application Note	Describes operational details of the QSPI Boot Loader Program.	RZT1 QSPI Boot Loader Application Note	R01AN2471EG

2. List of Abbreviations and Acronyms

Abbreviation	Full Form
ADC	Analog-to-Digital Converter
API	Application Programming Interface
COM	COMmunications port referring to PC serial port
CPU	Central Processing Unit
DVD	Digital Versatile Disc
E1	Renesas On-chip Debugger
GUI	Graphical User Interface
IDE	Integrated Development Environment
IRQ	Interrupt Request line
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MCU	Micro-controller Unit
PC	Personal Computer
Pmod™	Digilent Pmod™ Compatible connector. PmodTM is registered to Digilent Inc. Digilent-Pmod_Interface_Specification
PLL	Phase-locked Loop
QSPI	Quad Serial Peripheral Interface
RSK	Renesas Starter Kit
SCI	Serial Communications Interface
SPI	Serial Peripheral Interface

Table of Contents

1. Overview

1.1 Purpose

This RSK is an evaluation tool for Renesas microcontrollers. This manual describes how to use the e^2 studio IDE Code Generator plug in to create a working project for the RSK platform.

1.2 Features

This RSK tutorial guides the user through creating a project to evaluate the following features:

- Project creation with e^2 studio,
- Code Generation using the Code Generator plug in,
- User circuitry such as switches, LEDs and a potentiometer.

The RSK board contains all the circuitry required for microcontroller operation.

2. Introduction

This manual is designed to answer, in tutorial form, how to use the Code Generator plug in for the RZ family together with the e^2 studio IDE to create a working project for the RSK platform. The tutorials help explain the following:

- Project generation using the e^2 studio,
- Detailed use of the Code Generator plug in for e^2 studio,
- Integration with custom code,
- Building and running the project e^2 studio.

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

- 'HardwareDebug' is a project built with the debugger support included. Optimisation is set to zero.
- 'Release' is a project with optimised compile options, producing code suitable for release in a product.

Some of the illustrative screenshots in this document will show text in the form RZxx. These are general screenshots and are applicable across the whole RZ family. In this case, simply substitute RZxx for RZT1

These tutorials are designed to show you how to use the RSK and are not intended as a comprehensive introduction to the e^2 studio debugger, compiler toolchains or the J-Link emulator. Please refer to the relevant user manuals for more in-depth information.

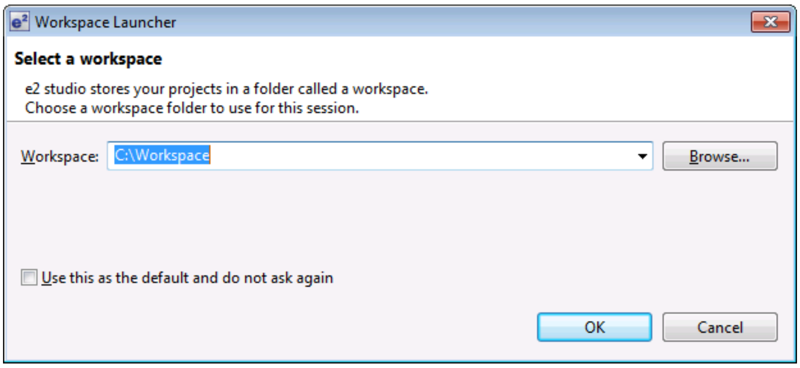
3. Project Creation with e2 studio

3.1 Introduction

In this section the user will be guided through the steps required to create a new 'C' project for the RZT1 microcontroller, ready to generate and add peripheral driver code using Code Generator. This project generation step is necessary to create the MCU-specific source, project and debug files.

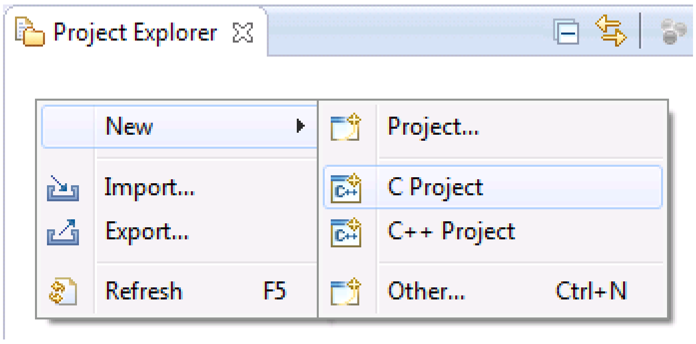
3.2 Creating the Project

- Start e2 studio and select a suitable location for the project workspace. Start e2 studio and select a suitable location for the project workspace.

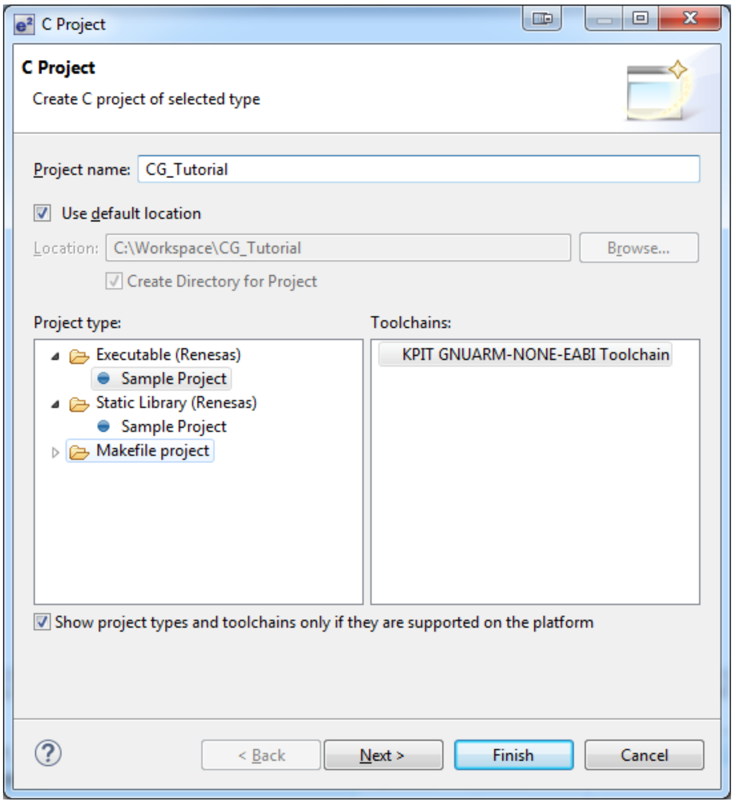


- In the Welcome page, click 'Go to the workbench'.

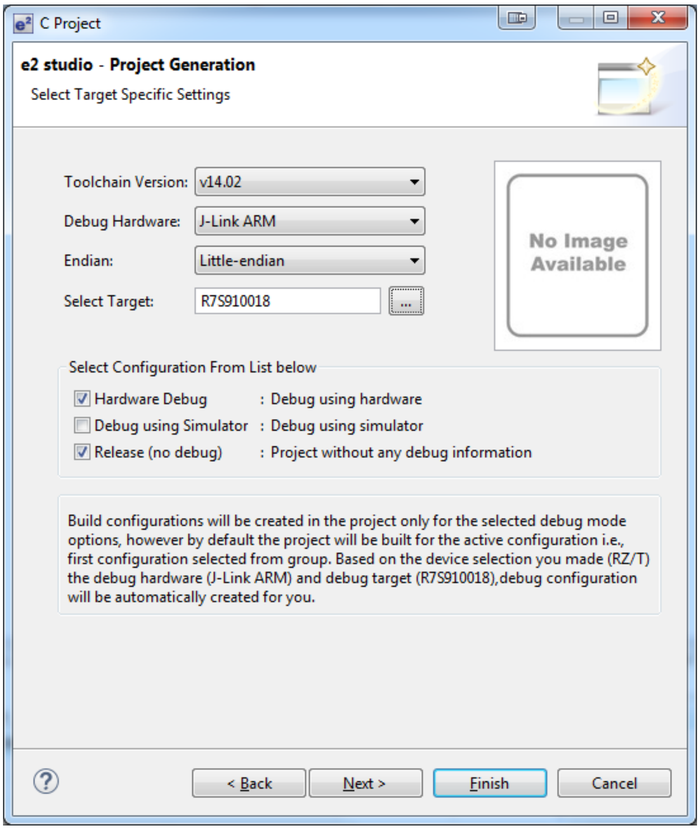
- Create a new C project by right-clicking in the Project Explorer pane and selecting ‘New -> C Project’ as shown. Alternatively, use the menu item ‘File -> New -> C Project’.



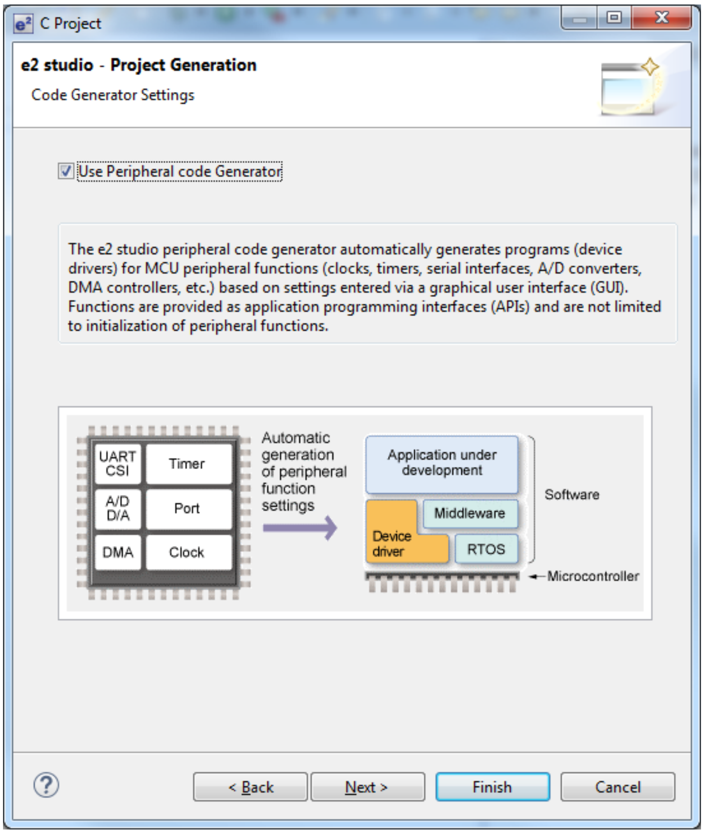
- Enter the project name ‘CG_Tutorial’. In ‘Project type:’ choose ‘Sample Project’. In ‘Toolchains’ choose ‘KPIT GNUARM-NONE-EABI Toolchain’. Click ‘Next’.



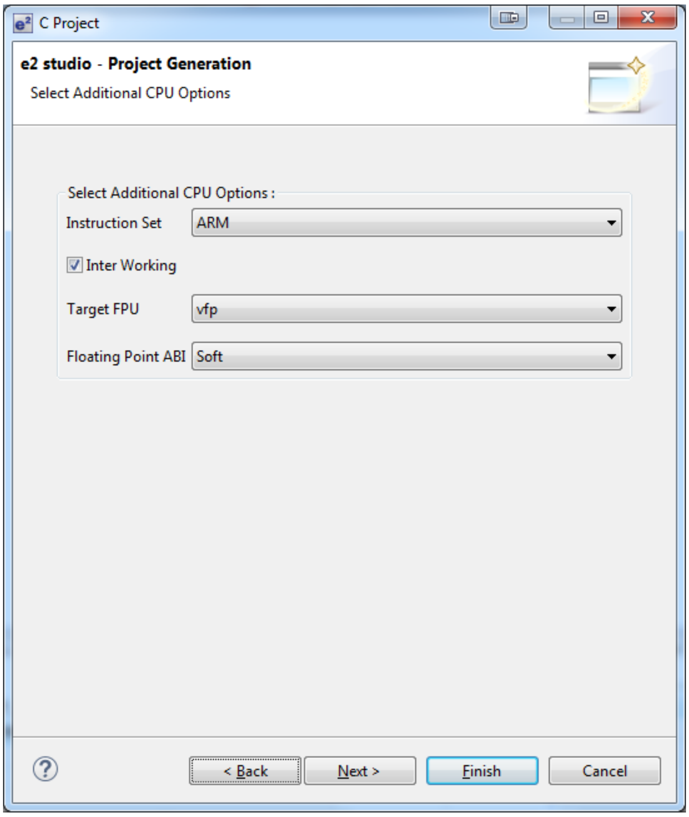
- In the ‘Target Specific Settings’ dialog, select the options as shown in the screenshot opposite.
- The R7S910018 MCU is found under RZ/T -> RZ/T1 -> RZ/T1 - 320 pin -> R7S910018.
- Click ‘Next’.



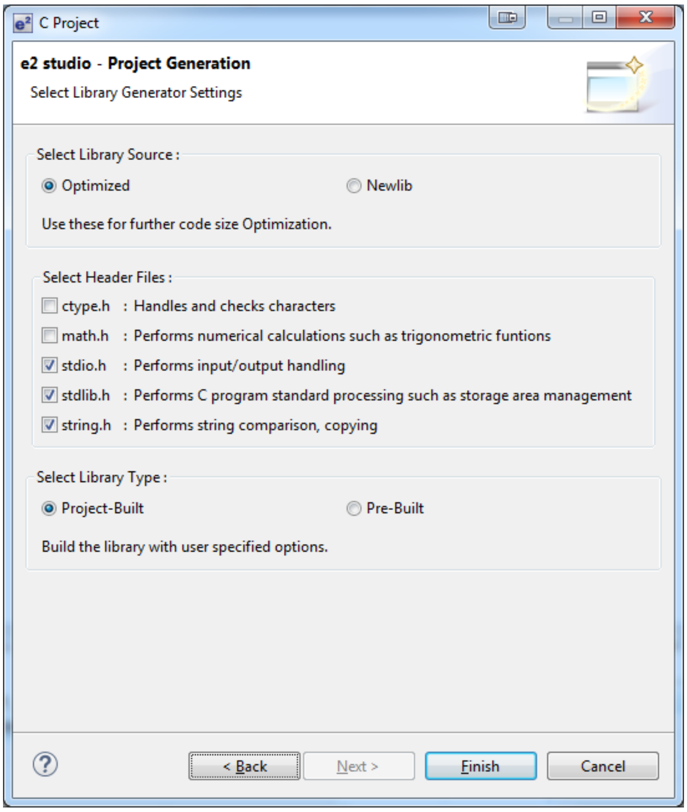
- Select ‘Use Peripheral code Generator’.
- Click ‘Next’.



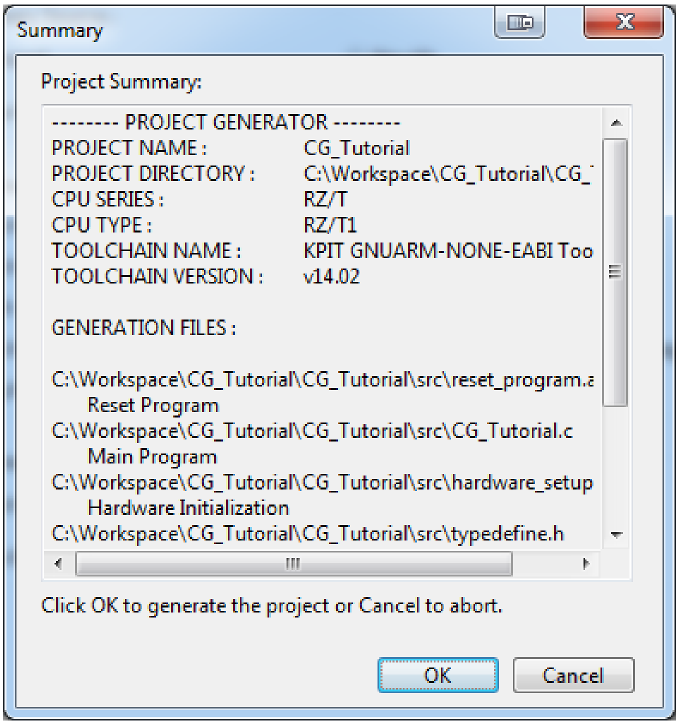
- In 'Select Additional CPU Options' leave everything at default values.
- Click 'Next'.



- In the 'Library Generator Settings' leave everything at default values.
- Click 'Finish'.



- A summary dialog will appear, click 'OK' to complete the project generation.



The generated sample is a fully functional sample that can be built and executed, however, for the purpose of this tutorial, the sample's functionality will not be tested.

Note: the sample toggles LED0 on the RSK+RZT1 board. The toggling rate changes with variations of the potentiometer (RV1). Pressing SW3 enables/disables the LED toggling. This manual does not focus on the functionality of the sample.

Use 'Build Project' from the 'Project' menu or the  button to build the tutorial. The project will build with no errors.

4. Using the Code Generator

4.1 Introduction

Code Generator is a GUI tool for generating template 'C' source code. This tool comes in two versions, either as an integrated plugin in e^2 studio or as a standalone application. The Code Generator tool distributed with the RSK+RZT1 is the plugin version. Code Generator enables the user is to configure various MCU features and operating parameters using intuitive GUI controls, bypassing the need, in most cases, to refer to sections of the Hardware Manual.

By following the steps detailed in this tutorial, the user will generate an e^2 studio project called CG_Tutorial. A fully completed CG_Tutorial project is contained on the DVD and may be imported into e^2 studio by following the steps in the Quick Start Guide. This tutorial is intended as a learning exercise for users who wish to use the Code Generator to generate their own custom projects for e^2 studio.

Once the user has configured the project, the 'Generate Code' function is used to generate three code modules for each specific MCU feature selected. These code modules are named 'r_cg_xxx.h', 'r_cg_xxx.c', and 'r_cg_xxx_user.c', where 'xxx' is a three letter acronym for the relevant MCU feature, for example 'scifa'. Within these code modules, the user is free to add custom code to meet their specific requirement. Custom code should be added between the following comment delimiters:

```
/* Start user code for adding. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
```

Code Generator will locate these comment delimiters, and preserve any custom code inside the delimiters on subsequent code generation operations. Any code outside of these comment delimiters will be overwritten on subsequent code generation sessions.

The CG_Tutorial project uses the ADC module with external trigger, Interrupt Controller Unit (ICU) and Comapare Match Timer (CMT) and an LED (I/O Port). As a demonstration this tutorial performs the following actions:

- Configure an LED to be toggled.
- Configure an A/D channel for setting the toggling period.
- Configure a timer channel to generate the toggling period.
- Configure a switch used to enable or disable toggling of the LED.

Following a tour of the key user interface features of Code Generator in Section 4.2, the reader is guided through each of the peripheral function configuration dialogs in Section 4.3.1. In Section 5, the reader is familiarised with the structure of the template code, as well as how to add custom code in the areas provided by the Code Generator and any other changes required to be made in the project generated in Section 3.2.

4.2 Code Generator Tour

This section presents a brief tour of Code Generator. AP4 is the stand-alone version of Code Generator and this manual is applicable to the Code Generator.

From the e^2 studio menus, select 'Window -> Open Perspective -> Other. In the 'Open Perspective' dialog shown in Figure 4-1, select 'Code Generator' and click 'OK'.

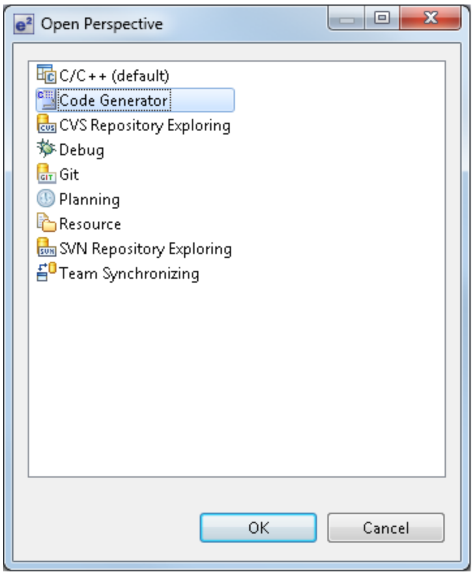


Figure 4-1: Changing Perspectives

A Code Generator project file with extension '.cpg' exists in the CG_Tutorial project's .settings/CodeGenerator directory. Code Generator also creates a folder named 'cg_src' in the 'src' folder to store generated source and header files. The user is encouraged to add non-CodeGenerator files to the 'src' folder.

The Code Generator's Peripheral View is displayed as illustrated in Figure 4-2.

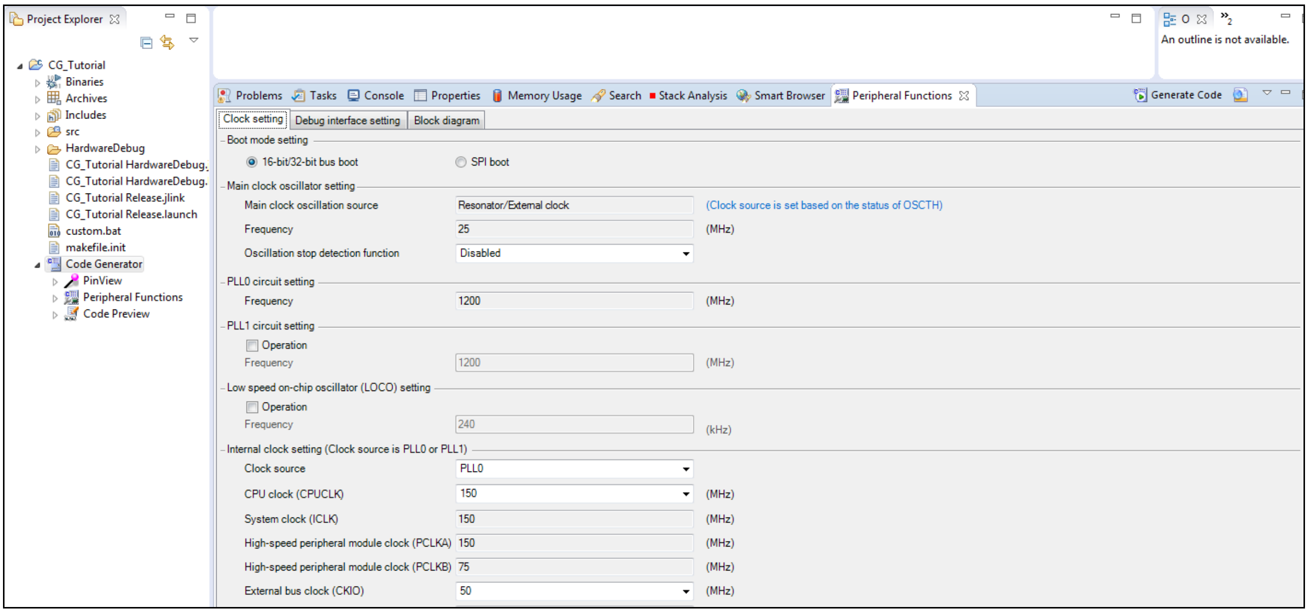


Figure 4-2: Peripheral View

Code Generator provides GUI features for configuration of MCU sub-systems and peripherals. Once the user has configured all required MCU sub-systems and peripherals, the user can click the 'Generate Code' button, resulting in the creation of a number of source and header files in the e^2 studio project's 'src' folder. A few more steps will need to be carried out before the project is fully configured and ready for use.

Navigation to the MCU peripheral configuration screens may be performed by double-clicking the required function in the Code Generator -> Peripheral Function under the Project View pane.

It is also possible to see a preview of the code that will be generated for the current peripheral function settings by double-clicking the required function in the Code Generator -> Code Preview on the left.

4.3 Code Generation

In the following sections, the reader is guided through the steps to configure the MCU peripherals required.

Note: Configuration options that are not shown should be left with the default settings.

4.3.1 Peripheral Function Configuration

4.3.1.1 Clock Generator

Figure 4-3 shows a screenshot of Code Generator with the Clock Generator function open.

In this tutorial we are using the 25 MHz crystal resonator for the main clock source with the PLL circuit used as a multiplier. Some peripherals can be configured to use the main clock or PLL circuitry sources to generate their clock.

Double click on the ‘Clock Generation Circuit’ entry in Code Generator -> Peripheral Functions list in the Project Tree.

Configure the Clock Generation Circuit options under the ‘Clock Settings’ tab as shown in Figure 4-3.

A block diagram of the Clock Generation Circuit is provided under ‘Block Diagram’ tab, shown on the next page. This helps to see the different clock configurations available for the system and peripheral clocks.

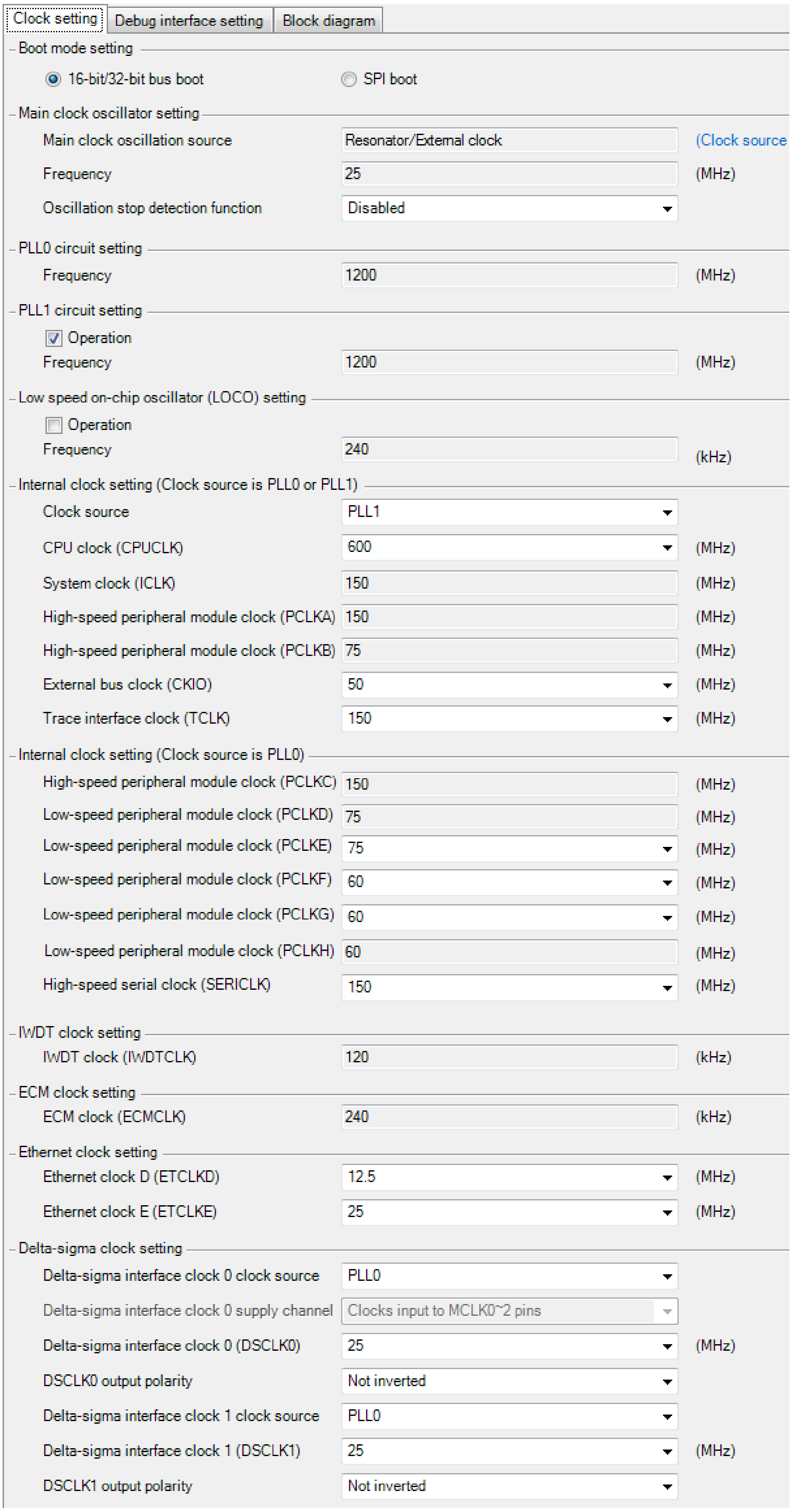


Figure 4-3: Clock setting tab

4.3.1.2 I/O Ports

This peripheral will be configured to assign output pins for user LEDs. The CG_Tutorial only makes use of LED0. User LED connectivity port pins on the schematic are as shown in Table 4-1: I/O Ports Connectivity.

Function	MCU Pin	I/O Port	Note
LED0	A5	PF7	
LED1	E3	P56	Not used
LED2	K16	P77	Not used
LED3	J18	PA0	Not used

Table 4-1: I/O Ports Connectivity

Please refer to the RSK schematic for full details of the connectivity.

Double click on the 'I/O Ports' entry in Project Tree -> Peripheral Functions -> I/O Ports. Expand the list. Configuration is required for the port pins listed in Table 4-1: I/O Ports Connectivity.

Configure the port as shown in Figure 4-4: LED Port Pin Configuration.

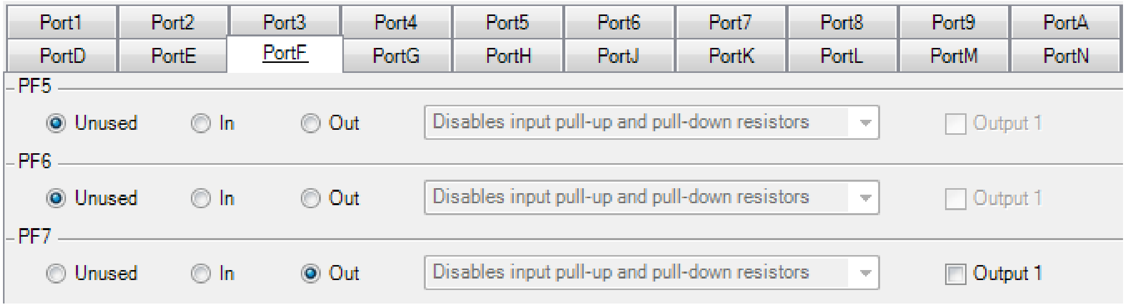


Figure 4-4: LED Port Pin Configuration

4.3.1.3 Compare Match Timer (CMT)

This peripheral is configured to generate regular intervals used to flash LED0.

Double click on the 'Compare Match Timer' entry in Project Explorer -> CG_Tutorial -> Code Generator -> Peripheral Functions.

Configure the CMT channel as shown in Figure 4-5.

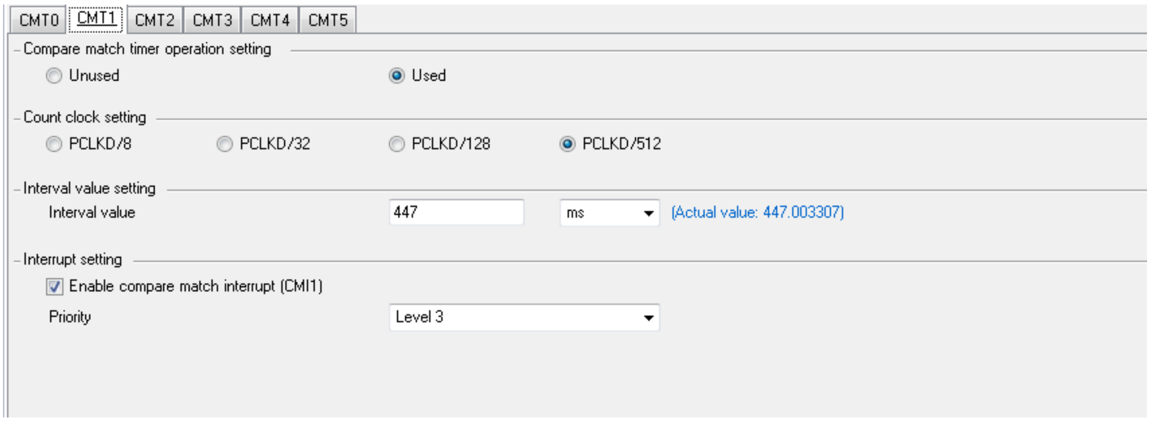


Figure 4-5: CMT Setting Tab

4.3.1.4 A/D Converter

This peripheral is configured to sample the analogue output value of the RV1 potentiometer. The A/D Converter is set to perform a sample when the user presses SW3, which is connected to the **AN007** pin of the microcontroller.

Double click on the 'S12AD0' entry in Project Tree -> Peripheral Functions -> 12-Bit A/D Converter.

Configure the 'Setting 1' sub-tab as shown in the following figures:

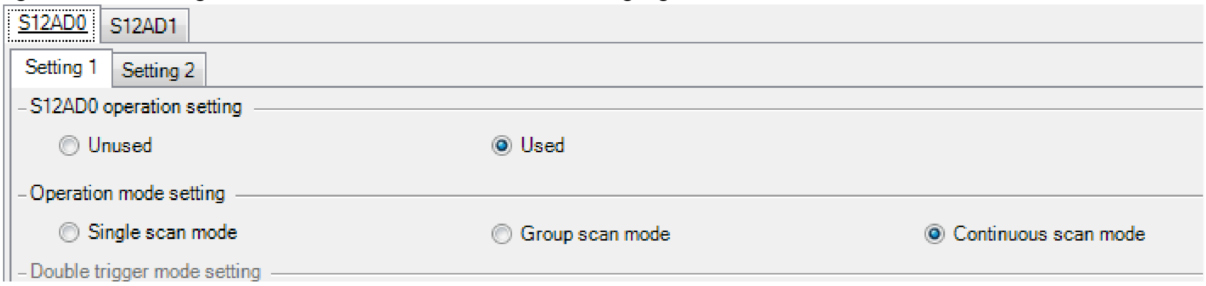


Figure 4-6: A/D Converter Setting tab

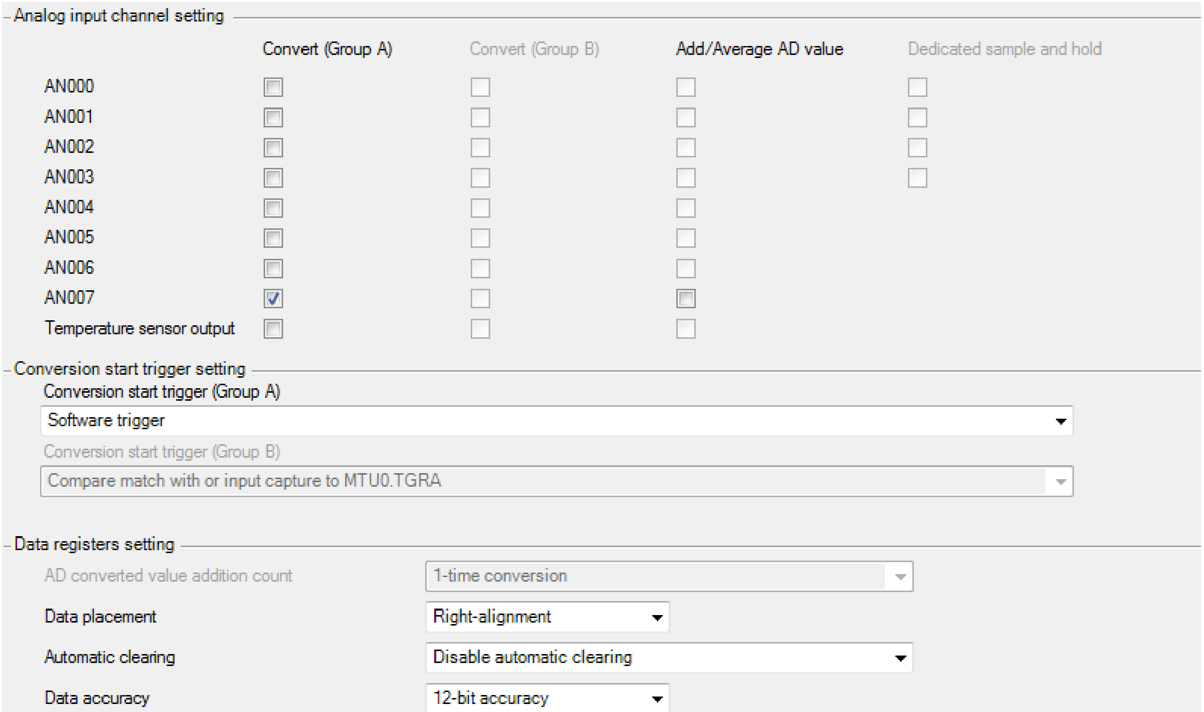


Figure 4-7: A/D Converter Setting tab (2)

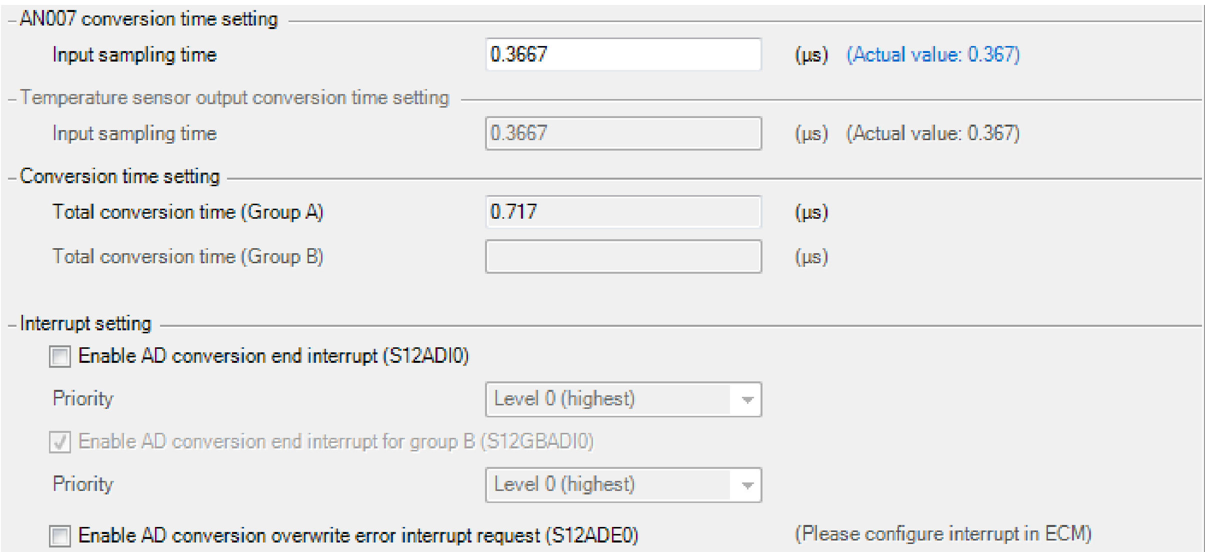


Figure 4-8 A/D Converter Setting tab (3)

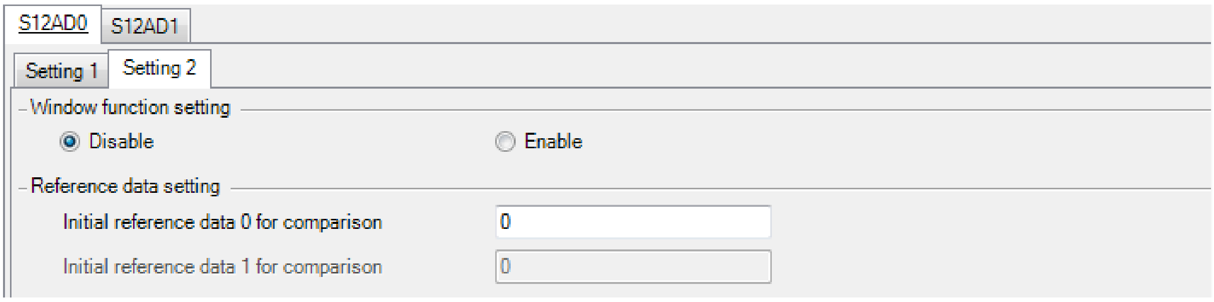


Figure 4-9 A/D Converter Setting tab (4)

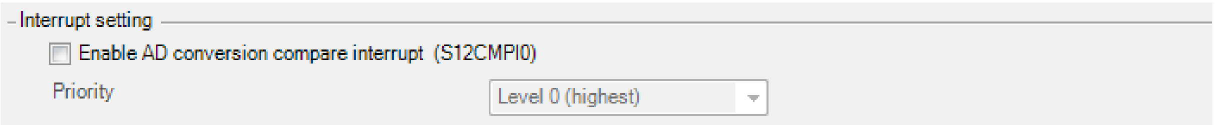


Figure 4-10: A/D Converter Setting tab (5)

4.3.1.5 Interrupt Controller Unit (ICU)

This peripheral is used to configure external interrupts input pins connected to user switches. The CG_Tutorial only makes use of switch SW3. User switch connectivity on the schematic are shown in Table 4-2: ICU Connectivity

Function	MCU Pin	I/O Port	Note
NMI	H3	P35	Not used.
IRQ5	W3	PN5	Not used.
IRQ12	W15	P44	SW3

Table 4-2: ICU Connectivity

Please refer to the RSK schematic for full details of the connectivity.

Double click on the ‘Interrupt Controller’ entry under the Project Tree -> Peripheral Functions list.

This is to configure switch SW3 to trigger IRQ12 interrupts.

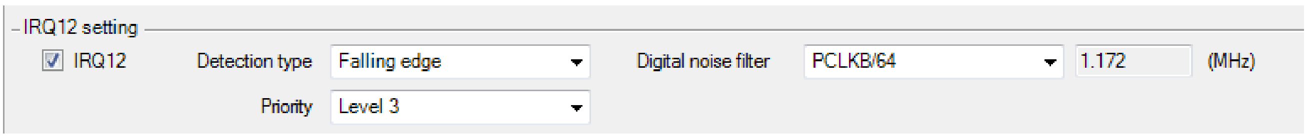


Figure 4-11: ICU Setting tab

4.3.1.6 Multi-Function Pin Controller (MPC)

This peripheral is used to select and map port/peripheral functionalities of the MCU pins. By default, mapping of functionalities to pins is done during peripheral configuration as shown in the setup of the I/O Ports, A/D and ICU modules. The Multi-Function Controller is used to re-assign the default functionalities mapping if required.

Double click on the ‘Device List View’ entry under the Project Tree -> Pin View.

Please ensure to verify the port pin functions for each configured peripheral by viewing the ‘Pin Number’ and ‘Pin Function’ tabs as shown in Figure 4-12: Pin Number Device List View tab and Figure 4-13: Pin Function Device List View tab

Pin Number	Pin Name	Selected Function	Pin Direction	Pin Remarks
A1	VSS	VSS	-	
A2	PC2/ ETH0_TXC/ ETH1_RXD2/ CATI2CDATA/ SDA0	Not assigned	-	
A3	PJ3/ IRQ11/ ETH0_TXD0/ ADTRG0	Not assigned	-	
A4	PJ1/ ETH0_TXD2/ CATLEDSTER/ RSPCK3	Not assigned	-	
A5	PF7/ IRQ7/ A25/ ETH0_TXER/ RTS3#/ SSL30	PF7	Out	
A6	PB4/ A24/ ETH1_COL/ ETH0_RXER/ CATSYNC0/ CATL...	Not assigned	-	
A7	PB0/ ETH1_RXDV/ MTCLKB/ TCLKD/ TIC3	Not assigned	-	
A8	PC0/ WAIT#/ ETH1_RXD2/ GTETRG/ SCL1/ MDAT3	Not assigned	-	
A9	PF6/ ETH1_RXD0/ MTIOC3D/ GTIOC0B/ TOC2	Not assigned	-	
A10	VCCQ33	VCCQ33	-	
A11	P54/ CLKOUT25M1/ MOSI2	Not assigned	-	
A12	VSS	VSS	-	
A13	AN007	AN007	In	
A14	AN005	Not assigned	-	
A15	AN002	Not assigned	-	
A16	AVCC0	Not assigned	-	
A17	AVCC1	Not assigned	-	
A18	VREFH1	Not assigned	-	
A19	P17/ CS5#/ ETH1_TXER/ PHYRESETOUT#/ ADTRG0	Not assigned	-	
A20	VSS	VSS	-	
B1	PJ5/ ETH0_RXD1/ TIOCD0/ RXD3	Not assigned	-	
B2	PJ4/ ETH0_RXD0/ TXD3	Not assigned	-	
B3	PC3/ ETH0_RXC/ ETH0_RXDV/ CATI2CCLK/ RXD4/ SC...	Not assigned	-	

Pin Number Pin Function

Figure 4-12: Pin Number Device List View tab

- Clock Generation Circuit
- Interrupt Controller
- Bus state controller
- DMA Controller
- I/O Ports
- Multi-Function Timer Pulse
- Port Output Enable 3
- General PWM Timer
- 16-Bit Timer Pulse Unit
- Programmable Pulse Gener
- Compare Match Timer W
- Serial Communications Inte
- I2C Bus Interface
- Serial Peripheral Interface
- SPI Multi I/O Bus Controller
- ΔΣ Interface
- Error Control Module
- 12-Bit A/D Converter
- Gigabit Ethernet MAC
- EtherCAT Slave Controller
- USB 2.0 HS Host/Function
- CAN Interface
- Serial Sound Interface
- Others

Pin Name	Pin Assignment	Pin Number	Pin Direction	Pin Rem
NMI	Not assigned	Not assigned	In	
IRQ0	Not assigned	Not assigned	In	
IRQ1	Not assigned	Not assigned	In	
IRQ2	Not assigned	Not assigned	In	
IRQ3	Not assigned	Not assigned	In	
IRQ4	Not assigned	Not assigned	In	
IRQ5	Not assigned	Not assigned	In	
IRQ6	Not assigned	Not assigned	In	
IRQ7	Not assigned	Not assigned	In	
IRQ8	Not assigned	Not assigned	In	
IRQ9	Not assigned	Not assigned	In	
IRQ10	Not assigned	Not assigned	In	
IRQ11	Not assigned	Not assigned	In	
IRQ12	PN4/ IRQ12/ MTIOC6C/ TIOCC6/ SSL11	V4	In	
IRQ13	Not assigned	Not assigned	In	
IRQ14	Not assigned	Not assigned	In	
IRQ15	Not assigned	Not assigned	In	
ETH0_INT	Not assigned	Not assigned	In	
ETH1_INT	Not assigned	Not assigned	In	
ETH2_INT	Not assigned	Not assigned	In	

Pin Number **Pin Function**

Figure 4-13: Pin Function Device List View tab

Figure 4-13 shows IRQ12 (Pin Name) function assigned to MCU pin number V4 but on the RSK+RZT1 schematic, V4 is connected to I/O port pin P44. The IRQ12 function needs to be re-assigned to P44 (MCU pin number W15).

To assign/re-assign a pin, click on the pin to reveal a drop-down menu button. Click on the button to reveal a list of available pins, then select W15 as shown in Figure 4-14.

IRQ11	Not assigned	Not assigned	In
IRQ12	PN4/ IRQ12/ MTIOC6C/ TIOCC6/ SSL11	V4	In
IRQ13	Not assigned	Not assigned V4	In
IRQ14	Not assigned	W13	In
IRQ15	Not assigned	W15	In

Figure 4-14: Pin Function Assignment/Re-assignement

The MPC assigns pins in the order in which the peripherals were configured. Once a pin has been assigned to a function, configuring another peripheral that uses the same function will result in the assignment of that function to an alternate pin. **In addition, the MPC does not automatically map functions to pins based on the connectivity on the RSK+RZT1**. This is why it is important to verify the pin functions in the Device List View.

A pin name will be shown in red if configuration clashes exist.

A remark will be shown on the Pin Remark column of the Device List View if a pin function is assigned without configuring the peripheral that uses the function.

4.3.2 Generating the Code

Peripheral function configuration is now complete. Click ‘Generate Code’ button located at the top right of the Peripheral Function tab. The Output pane should report ‘The operation of generating file was successful’, as shown Figure 4-15 below.

```
M0409001:The following files were generated:
M0409004:src\cg_src\r_cg_main.c was overwritten.
M0409004:src\cg_src\r_cg_mpc.c was overwritten.
M0409004:src\cg_src\r_cg_mpc.h was overwritten.
M0409004:src\cg_src\r_cg_interrupthandlers.h was overwritten.
M0409004:src\cg_src\r_cg_intprg.c was overwritten.
M0409004:src\cg_src\r_cg_systeminit.c was overwritten.
M0409004:src\cg_src\r_cg_macrodriver.h was overwritten.
M0409004:src\cg_src\r_cg_userdefine.h was overwritten.
M0409004:src\cg_src\r_cg_cgc.c was overwritten.
M0409004:src\cg_src\r_cg_cgc_user.c was overwritten.
M0409004:src\cg_src\r_cg_cgc.h was overwritten.
M0409004:src\cg_src\r_cg_icu.c was overwritten.
M0409004:src\cg_src\r_cg_icu_user.c was overwritten.
M0409004:src\cg_src\r_cg_icu.h was overwritten.
M0409004:src\cg_src\r_cg_port.c was overwritten.
M0409004:src\cg_src\r_cg_port_user.c was overwritten.
M0409004:src\cg_src\r_cg_port.h was overwritten.
M0409004:src\cg_src\r_cg_cmt.c was overwritten.
M0409004:src\cg_src\r_cg_cmt_user.c was overwritten.
M0409004:src\cg_src\r_cg_cmt.h was overwritten.
M0409004:src\cg_src\r_cg_s12ad.c was overwritten.
M0409004:src\cg_src\r_cg_s12ad_user.c was overwritten.
M0409004:src\cg_src\r_cg_s12ad.h was overwritten.
M0409003:The operation of generating file was successful.
```

Figure 4-15: Code Generator’s Output pane

5. Adding Code to Generated Files

At this stage of a typical project development the user would expand on the generated code to create the application required.

When inserting code in Code Generator created files, it must be placed in the areas delimited by comments as follows:

```
/* Start user code for _xxxxx_. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
```

Where _xxxx_ depends on the particular area of code, i.e. 'function' for insertion of user functions and prototypes, 'global' for insertion of user global variable declarations, or 'include' for insertion of pre-processor include directives. User code inserted inside these comment delimiters is protected from being overwritten by Code Generator, if the user regenerates the Code Generator code.

5.1 Excluding Files

All sample code can only have one main file. The init_main.c file generated in Section 3.2 and the r_cg_main.c file generated by Code Generator both include a main function. The init_main.c file is automatically excluded following code generation. To exclude a file from the project following these steps:

1. Locate the source file in the 'Project Explorer' view.
2. Right click on the file and select 'Exclude from build…'.
3. Click on 'Select All' to make change on all available build configurations.
4. Click 'OK'.

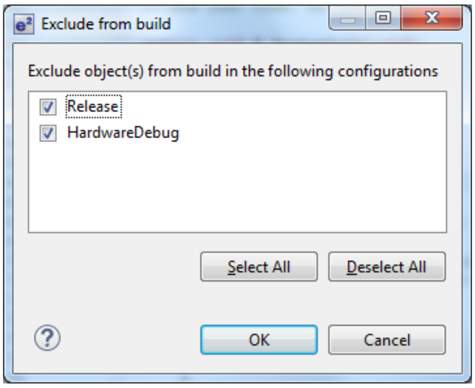


Figure 5-1: Excluding files from the project

Multiple files can be excluded in on step by selecting the desired files during step 1. Exclude the following files:

loader_param.c
r_ecm.c
r_ecm.h
r_icu_init.c
r_icu_init.h
r_system.h
typdefine.h

To re-include a file, repeat the above steps and click on 'Deselect All' then click 'OK'.

5.2 Adding Code to Generated Files

This section covers inserting code in to the newly created Code Generator files.

Each subsection is a Code Generator generated source file that needs to be opened by double clicking on the file name in e^2 studio's Project Tree window under the 'src' folder.

The code from each section should be copied from this document and pasted in to the relevant file at the location indicated.

5.2.1 r_cg_userdefine.h Code Insertion

Open this file by double clicking on the file name in e^2 studio's Project Tree window.

Insert the following at the end of the file between the user code delimiter comments as shown below.

```
/* Start user code for function. Do not edit comment generated here */

#define LED0        (PORTF.PODR.BIT.B7)

/* End user code. Do not edit comment generated here */
```

5.2.2 r_cg_icu_user.c Code Insertion

Open this file by double clicking on the file name in e^2 studio's Project Tree window.

Insert the followings between the specific user code insertion delimiter comments as shown below.

```
/* Start user code for global. Do not edit comment generated here */

volatile uint8_t g_switch_press = 0;

/* End user code. Do not edit comment generated here */
```

```
/* Start user code. Do not edit comment generated here */

/* Invert the flag */
g_switch_press = (~g_switch_press);

/* End user code. Do not edit comment generated here */
```

5.2.3 r_cg_icu.h Code Insertion

Open this file by double clicking on the file name in e^2 studio's Project Tree window.

Insert the following at the end of the file between the user code delimiter comments as shown below.

```
/* Start user code for function. Do not edit comment generated here */

extern volatile uint8_t g_switch_press;

/* End user code. Do not edit comment generated here */
```

5.2.4 r_cg_cmt_user.c Code Insertion

Open this file by double clicking on the file name in e^2 studio's Project Tree window.

Insert the following between the user code delimiter comments as shown below in the file section designated Global variables and functions:

```
/* Start user code for include. Do not edit comment generated here */
#include "r_cg_icu.h"
#include "r_cg_cmt.h"
#include "r_cg_s12ad.h"
/* End user code. Do not edit comment generated here */
```

```
/* Start user code for global. Do not edit comment generated here */

/* Function prototype for scaling a value */
static uint32_t scale_value (const uint32_t value, const uint32_t in_max, const uint32_t out_max);

/* End user code. Do not edit comment generated here */
```

Insert the following in to the function `void r_cmt_cmi1_interrupt(void)`

```
    /* Start user code. Do not edit comment generated here */

    /* Update the period based on the flag set in the switch handler interrupt */
    if (0 == g_switch_press)
    {
        /* scale the ADC value. ADC range: 0-4095, CMT range: 0 - 65478 */
        CMT1.CMCOR = scale_value ((uint32_t)(S12ADC0.ADDR7), 4095, 65478);
    }
    else
    {
        /* Do not update the CMT period */
    }

    LED0 = (~LED0);

    /* End user code. Do not edit comment generated here */
```

Insert the following between the user code delimiter comments at the end of the file:

```
/* Start user code for adding. Do not edit comment generated here */

/******************************************************************************************
* Function Name: scale_value
* Description  : This function is CMI1 interrupt service routine.
*                The formula used
*                output = 1 + (value - 0) * (out_max - 0) / (in_max - 0)
*
*                Note - The actual and desired ranges' minimum value is assumed to be 0.
* Arguments    : uint32_t value   - value to scale
*                                  uint32_t in_max  - maximum range of value to scale
*                                  uint32_t out_max - maximum range of desired scale
* Return Value : None
******************************************************************************************/
static uint32_t scale_value (const uint32_t value, const uint32_t in_max, const uint32_t out_max)
{
        uint32_t output;

        output = (out_max - 0) / (in_max - 0);
        output = (value - 0) * output;
        output = (1 + output);

        return output;
}

/******************************************************************************************
* End of function scale_value
******************************************************************************************/

/* End user code. Do not edit comment generated here */
```

5.2.5 r_cg_main.c Code Insertion

Insert the following in to the function `void main (void)`.

```
    /* Start user code. Do not edit comment generated here */

    /* The rest of the code is executed in interrupt handlers */

    while (1U)
    {
        asm("nop");
    }
    /* End user code. Do not edit comment generated here */
```

Insert the following in to the function **void R_MAIN_UserInit (void):**

```
/* Start user code. Do not edit comment generated here */

uint32_t delay = 0x3FFFF;

/* Clear the switches' interrupt flags before enabling the interrupts */
VIC.PIC0.LONG = 0x00000200UL;
VIC.PIC0.LONG = 0x00010000UL;

/* Enable the switch interrupts */
R_ICU_IRQ12_Start();

/* Enabling interrupts can cause generation of an interrupt which should
   be ignored. Allow some delay to catch the interrupt should it occur. */
while ((0 == g_switch_press) && (delay--))
{
    asm("nop");
}

/* Ensure the switch pressed flag is cleared to enable timer period updates */
g_switch_press = 0;

/* Enable continuous A/D conversions */
R_S12AD0_Start();

/* Enable the timer's count */
R_CMT1_Start();

/* End user code. Do not edit comment generated here */
```

5.3 Additional include paths

Before the project can be built the compiler needs some additional include paths added. Select the CG_Tutorial project in the Project Explorer pane. Use the button in the toolbar to open the project settings. Navigate to 'C/C++ Build -> Settings ->Compiler -> Source and click the button as shown in below in Figure 5-2.

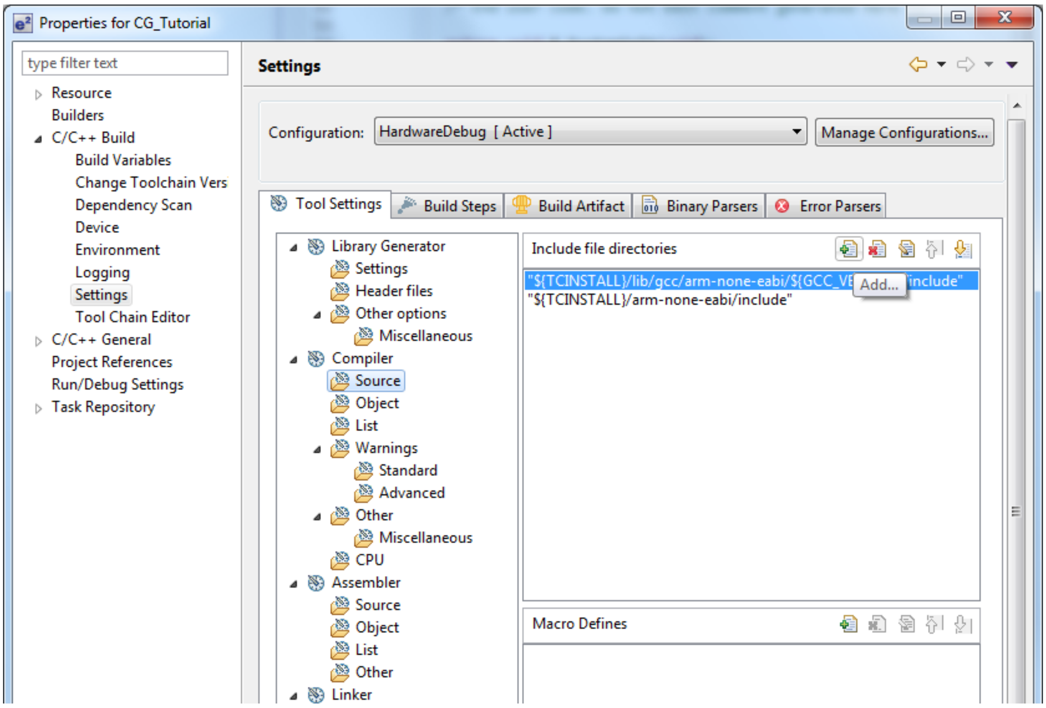


Figure 5-2: Adding additional search paths

In the ‘Add directory path’ dialog, click the ‘Workspace’ button and in the ‘Folder selection’ dialog browse to the ‘CG_Tutorial/src’ folder and click ‘OK’. e^2 studio formats the path as shown in Figure 5-3 below.

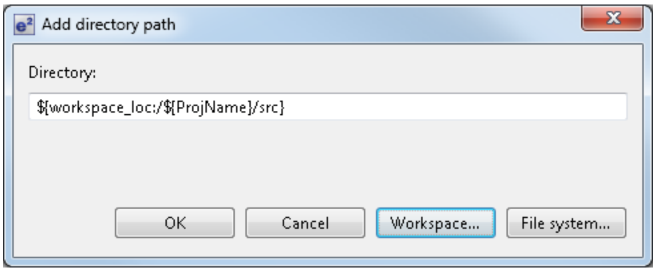


Figure 5-3 Adding workspace search path

Repeat the above steps to add the workspace search paths and press OK to exit the Properties dialog.

Select ‘Build Project’ from the ‘Project’ menu, or use the  button. e^2 studio will build the project with no errors. The project may now be run using the debugger as described in §7.

5.4 Release Build Section Map

Code Generator makes changes to Linker Section addresses while generating code. These changes are only performed on the build configuration currently selected.

The steps followed above will create a working ‘HardwareDebug’ build configuration. Follow the steps below to create a working ‘Release’ build configuration. For details of the differences in these build configurations please see Section 2.

Select the ‘Release’ Build configuration by clicking:
Project > Build Configurations > Set Active > Release (Release – No Debug).

In the Project Explorer tree, right click the entry ‘Code Generator’ and select ‘Generate Code’. This will run an update for the generated code and make the required changes to Linker Section addresses.

Open the Release configuration from:
Debug Configurations > Renesas GDB Hardware Debugging . CG_Tutorial Release

Select the ‘Startup’ tab. Ensure the ‘Set breakpoint at:’ option is unticked.

Note:
Section 6 will need to be done for both

6. External Linker File

e[2] studio allows specifying a different linker file to be used by the linker. The default linker map declaration can be found in:

Project properties > C/C++ Build > Settings > Linker > Sections

The CG_Tutorial code does not make use of the default linker mapping declaration. The loader_init.asm file includes variables declared in the default linker map. These variables are used for storing specific addresses in the linker file. Open the loader_init.asm and change #if 1 to #if 0

6.1 Linker File Over-ride

The following steps are used to create a new linker file, define the linker sections of the RZ/T1 device and set the GNU Linker to use the file created.

- Create a new file in the project
- Right click on the 'src' source folder.
- Select New > File
- Specify the name as: linker_file.ld
- Open the file by double-clicking on it.
- Copy and paste the following text:

```
/**************************************Start copying after this line*****************************************/

OUTPUT_FORMAT("elf32-littlearm", "elf32-bigarm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_PowerON_Reset)

MEMORY
{
  /* Internal RAM address range H'2000_0000 to H'2001_FFFF is configured as data retention RAM */
  /* Write access to this address range has to be enabled by writing to registers SYSCR1 and SYSCR2 */
  ATCM       (rwx)   : ORIGIN = 0x00000000, LENGTH = 0x00080000  /* (512KB)  H'00000000 to H'0007FFFF */
  BTCM       (rwx)   : ORIGIN = 0x00800000, LENGTH = 0x00800000  /* (32KB)   H'00800000 to H'00807FFF */
  BUFFER_RAM (rwx)   : ORIGIN = 0x08000000, LENGTH = 0x10000000  /* (128MB) H'08000000 to H'10000000  */
  DATA_RAM   (rwx)   : ORIGIN = 0x20000000, LENGTH = 0x00080000  /* (512KB)  H'20000000 to H'2007FFFF */

  /* Mapped memory type */
  SPI_ROM    (rw)    : ORIGIN = 0x30000000, LENGTH = 0x04000000
  CS0_ROM    (rw)    : ORIGIN = 0x40000000, LENGTH = 0x04000000
  CS1_ROM    (rw)    : ORIGIN = 0x44000000, LENGTH = 0x04000000
  SDRAM0_EXT (rw)    : ORIGIN = 0x48000000, LENGTH = 0x04000000
  SDRAM1_EXT (rw)    : ORIGIN = 0x4C000000, LENGTH = 0x04000000
}

SYS_STACK_SIZE     = 0x200;     /* Application stack size       */
SVC_STACK_SIZE     = 0x200;     /* SVC mode stack               */
IRQ_STACK_SIZE     = 0x100;     /* IRQ mode stack               */
FIQ_STACK_SIZE     = 0x100;     /* FRQ mode stack               */
UND_STACK_SIZE     = 0x100;     /* SVC mode stack               */
ABT_STACK_SIZE     = 0x100;     /* ABT mode stack               */
HEAP_STACK_SIZE    = 0x1000;    /* Heap stack size              */

ATCM_BASE          = 0x00000000;  /* User application located here          */
BTCM_BASE          = 0x00800000;  /* BTCM base address                      */
USER_EXEC_BASE     = 0x00000000;  /* Application loads and runs from here   */
USER_RAM           = 0x20000000;  /* Application's RAM base                 */
STACK_BASE         = 0x00807800;  /* Stacks located in BTCM                 */

SECTIONS
{
  .loader_text USER_EXEC_BASE :
  {
    reset_start = .;
    *(.loader_text);
    .   = ALIGN(0x4);
    reset_end = .;
  } > ATCM
```

```
.text :
{
    text_start = .;
    *(.text)
    *(.text.startup)
    text_end = .;
} > ATCM

.rodata :
{
    rodata_start = .;
    _start_data_ROM = .;
    *(.rodata)
    *(.rodata.*)
    .   = ALIGN(0x8);
    *(.data)
    *(.data.*)
    _end_data_ROM = .;
    *(.got.plt)
    *(.got)
    .   = ALIGN(0x8);
    rodata_end = .;
    PROVIDE(end = .);
} > ATCM

_ram_data_size = (_end_data_ROM - _start_data_ROM);

.data USER_RAM :
{
    _start_data_RAM = .;
    data_start = .;
    start_data_RAM = .;
    . += _ram_data_size;
    data_end = .;
}

.bss data_end :
{
     bss_start = .;
     PROVIDE(__bss_start__ = .);
     *(.bss)
     *(.bss.**)
     *(COMMON)
     . = ALIGN(0x4);
     PROVIDE(__bss_end__ = .);
     ebss_end = .;
     _end = .;
     PROVIDE(end = .);
}

.heap :
{
    heap_start = .;
    .   = ALIGN(0x8);
    *(.heap_stack)
    . += HEAP_STACK_SIZE;
    heap_end = .;
} > ATCM

.sys_stack 0x807800 : AT (0x807800)
{
    sys_stack_start = .;
    .   = ALIGN(0x8);
    *(.sys_stack)
    . += SYS_STACK_SIZE;
    sys_stack_end = .;
    _sys_stack = .;
} > BTCM

.svc_stack 0x807A00 : AT (0x807A00)
{
    svc_stack_start = .;
    .   = ALIGN(0x8);
    *(.svc_stack)
    . += SVC_STACK_SIZE;
    svc_stack_end = .;
```

```
        _svc_stack = .;
    } > BTCM

    .irq_stack 0x807C00 : AT (0x807C00)
    {
        irq_stack_start = .;
        .   = ALIGN(0x8);
        *(.irq_stack)
        . += IRQ_STACK_SIZE;
        irq_stack_end = .;
        _irq_stack = .;
    } > BTCM

    .fiq_stack 0x807D00 : AT (0x807D00)
    {
        fiq_stack_start = .;
        .   = ALIGN(0x8);
        *(.fiq_stack)
        . += FIQ_STACK_SIZE;
        fiq_stack_end = .;
        _fiq_stack = .;
    } > BTCM

    .und_stack 0x807E00 : AT (0x807E00)
    {
        und_stack_start = .;
        .   = ALIGN(0x8);
        *(.und_stack)
        . += UND_STACK_SIZE;
        und_stack_end = .;
        _und_stack = .;
    } > BTCM

    .abt_stack 0x807F00 : AT (0x807F00)
    {
        abt_stack_start = .;
        .   = ALIGN(0x8);
        *(.abt_stack)
        . += ABT_STACK_SIZE;
        abt_stack_end = .;
        _abt_stack = .;
    } > BTCM
}
```

/***Stop copying on the above line**/

Click File > Save

- Open Project properties > C/C++ Build > Settings > Linker Other
- Change the ‘Command file overide’ option to ‘External Linker script(-T)’.
- Add the following to the ‘File’ entry (including the speech marks): "${workspace_loc:/${ProjName}/src/linker_file.ld}"
- Click ‘Apply’.
- Navigate to Project properties > C/C++ Build > Settings > Linker Other > Miscellaneous
- Ensure to untick the ‘Enable garbage collection of unused input sections(-gc-sections) if it is ticked.
- Click ‘OK’.

The above 7 steps needs to be done for the HardwareDebug and Release configurations.

6.2 Building the Project

The project template created by Code Generator can now be built. In the Project Explorer pane expand the ‘src’ folder.

Use ‘Build Project’ from the ‘Project’ menu or the  button to build the CG_Tutorial project. The project will build with no errors.

7. Executing the Project

In the Project Explorer pane, ensure that the ‘CG_Tutorial’ project is selected. To debug the project, click the button. The dialog shown in Figure 7-1 will be displayed.

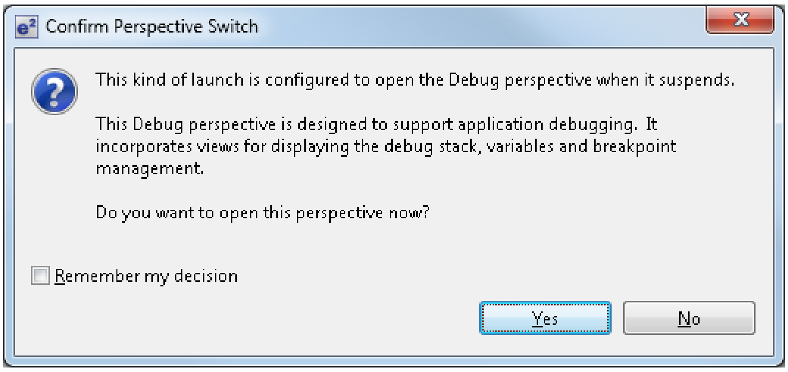


Figure 7-1: Perspective Switch Dialog

Click ‘OK’ to confirm that the debug window perspective will be used.

The debugger will start up and the e^2 studio will show the Code Generator function ‘PowerOn_Reset’.

Click the ‘Resume’ button. The debugger will stop again at the beginning of the main() function.
Press again to run the code.

The program will toggle LED0 at a rate set by the position of RV1. Slowly rotate RV1 fully clockwise then counter-clockwise and observe the change in the rate at which the LED toggles. Press SW3 to keep the rate at the position of RV1 when the SW3 was pressed. Rotating RV1 will not change the toggling rate. Press SW3 to re-enable the variations to the toggling.

For more information on the e^2 studio debugger refer to the Tutorial manual.

8. Usage Notes

8.1 iodefine.h File

Location of the iodefine.h file.
By default, the r_cg_macrodriver.h header file which includes the iodefine.h file expects the iodefine.h file to be located in the 'src' folder.

8.2 RIIC Module

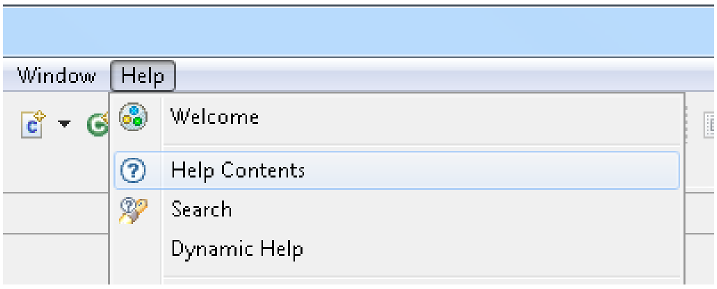
The RIIC peripheral contains an error in one of its interrupt handler functions. In the r_cg_riic_user.c file, in the function `void r_riic0_error_interrupt(void)`, replace the existing `else if` condition and the encapsulated lines of code with the following:

```
else if (_IIC_MASTER_RECEIVE == g_riic0_mode_flag)
{
    if ((_IIC_MASTER_SENDS_ADR_7_R == g_riic0_state) || (_IIC_MASTER_SENDS_ADR_10A_W == g_riic0_state))
    {
        RIIC0.ICSR2.BIT.START = 0U;
        RIIC0.ICIER.BIT.STIE = 0U;
        RIIC0.ICIER.BIT.SPIE = 1U; /* Enable stop condition detection to prepare for the next receive */

        /* Enable the TXI0 interrupt */
        VIC.IEN3.LONG |= 0x08000000UL;

        /* Enable the RXI0 interrupt */
        VIC.IEN3.LONG |= 0x04000000UL;
    }
    else if (_IIC_MASTER_RECEIVES_RESTART == g_riic0_state)
    {
        RIIC0.ICSR2.BIT.START = 0U;
        RIIC0.ICIER.BIT.STIE = 0U;
        g_riic0_state = _IIC_MASTER_SENDS_ADR_10A_R;
    }
    else if (_IIC_MASTER_RECEIVES_STOP == g_riic0_state)
    {
        RIIC0.ICMR3.BIT.RDRFS = 0U;
        RIIC0.ICMR3.BIT.ACKWP = 1U;
        RIIC0.ICMR3.BIT.ACKBT = 0U;
        RIIC0.ICSR2.BIT.NACKF = 0U;
        RIIC0.ICSR2.BIT.STOP = 0U;
        RIIC0.ICIER.BIT.SPIE = 0U;
        RIIC0.ICIER.BIT.STIE = 1U;  /* Enable start condition detection to prepare for the next receive */

        /* Clear TXI0 interrupt flag */
        VIC.PIC3.LONG = 0x08000000UL;
        /* Disable TXI0 interrupt */
        VIC.IEC3.LONG = 0x08000000UL;

        /* Clear RXI0 interrupt flag */
        VIC.PIC3.LONG = 0x04000000UL;
        /* Disable RXI0 interrupt */
        VIC.IEC3.LONG = 0x04000000UL;

        r_riic0_callback_receiveend();
    }
}
```

9. Additional Information

Technical Support

For details on how to use e[2] studio, refer to the help file by opening e[2] studio, then selecting Help > Help Contents from the menu bar.



For information about the RZ/T1 group microcontroller refer to the RZ/T1 Group Hardware Manual.

For information about the RZ assembly language, refer to the RZ Series Software Manual.

Technical Contact Details

Please refer to the contact details listed in section 11 of the "Quick Start Guide"

General information on Renesas microcontrollers can be found on the Renesas website at:
http://www.renesas.com/

Trademarks

All brand or product names used in this manual are trademarks or registered trademarks of their respective companies or organisations.

Copyright

This document may be, wholly or partially, subject to change without notice. All rights reserved. Duplication of this document, either in whole or part is prohibited without the written permission of Renesas Electronics Europe Limited.

REVISION HISTORY	RSK RZT1 Code Generator Tutorial Manual (e^2 studio)

Rev.	Date	Description	
		Page	Summary
1.00	Mar 21, 2015	¾	First Edition issued

Renesas Starter Kit Manual: Code Generator Tutorial Manual

Publication Date: Rev. 1.00 Mar 21, 2015

Published by: Renesas Electronics Corporation

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0

RZT1 Group

Renesas Electronics Corporation

R20UT3281EG0100