User's Manual

16

RL78/G1G Group

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

RENESAS MCU
RL78 Family / RL78/G1X Series

Renesas Electronics
www.renesas.com

Rev. 1.00 Jan 2015

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anticrime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(2012.4)

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 RL78 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 RL78/G1G 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 RL78/G1G 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.	RSKRL78/G1G User's Manual	R20UT3022EG
Tutorial	Provides a guide to setting up RSK environment, running sample code and debugging programs.	RSKRL78/G1G Tutorial Manual	R20UT3023EG
Quick Start Guide	Provides simple instructions to setup the RSK and run the first sample.	RSKRL78/G1G Quick Start Guide	R20UT3024EG
Code Generator Tutorial	Provides a guide to code generation in the e^2 studio IDE.	RSKRL78/G1G Code Generator Tutorial Manual	R20UT3025EG
Schematics	Full detail circuit schematics of the RSK.	RSKRL78/G1G Schematics	R20UT3017EG
Hardware Manual	Provides technical details of the RL78RL78/G1X microcontroller.	RL78/G1G Group, User's Manual: Hardware	R01UH0499EJ

2. List of Abbreviations and Acronyms

Abbreviation	Full Form
ADC	Analog-to-Digital Converter
CPU	Central Processing Unit
DVD	Digital Versatile Disc
E1	On-chip Debugger
GUI	Graphical User Interface
IDE	Integrated Development Environment
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
RSK	Renesas Starter Kit
SAU	Serial Array Unit
SPI	Serial Peripheral Interface
TAU	Timer Array Unit
UART	Universal Asynchronous Receiver/Transmitter

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 RL78 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 RL78xxx. These are general screenshots and are applicable across the whole RL78 family. In this case, simply substitute RL78xxx for RL78/G1G

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 E1 emulator. Please refer to the relevant user manuals for more in-depth information.

3. Project Creation with e2 studio

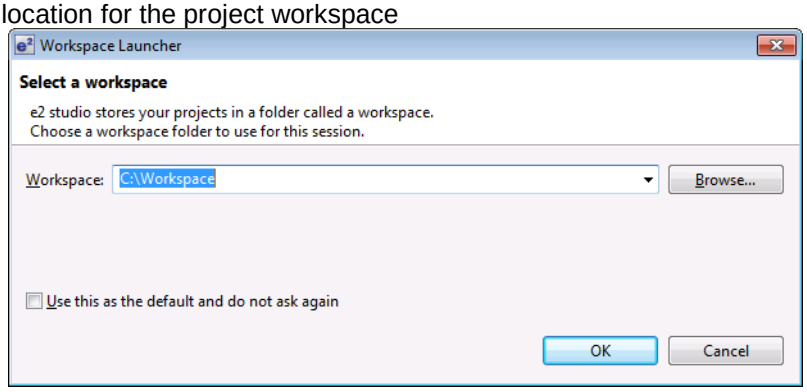
3.1 Introduction

In this section the user will be guided through the steps required to create a new 'C' project for the RL78/G1G microcontroller, ready to generate 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

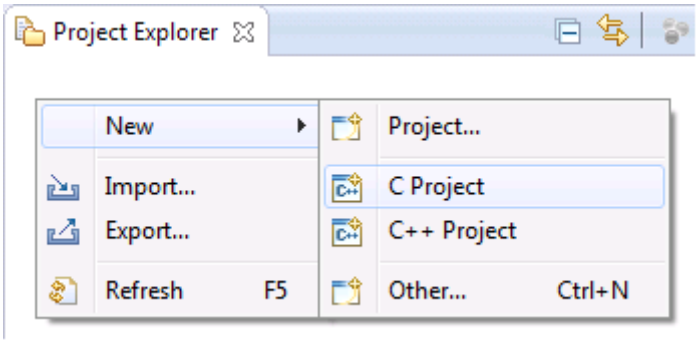
Start e2 studio and select a suitable location for the project workspace

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

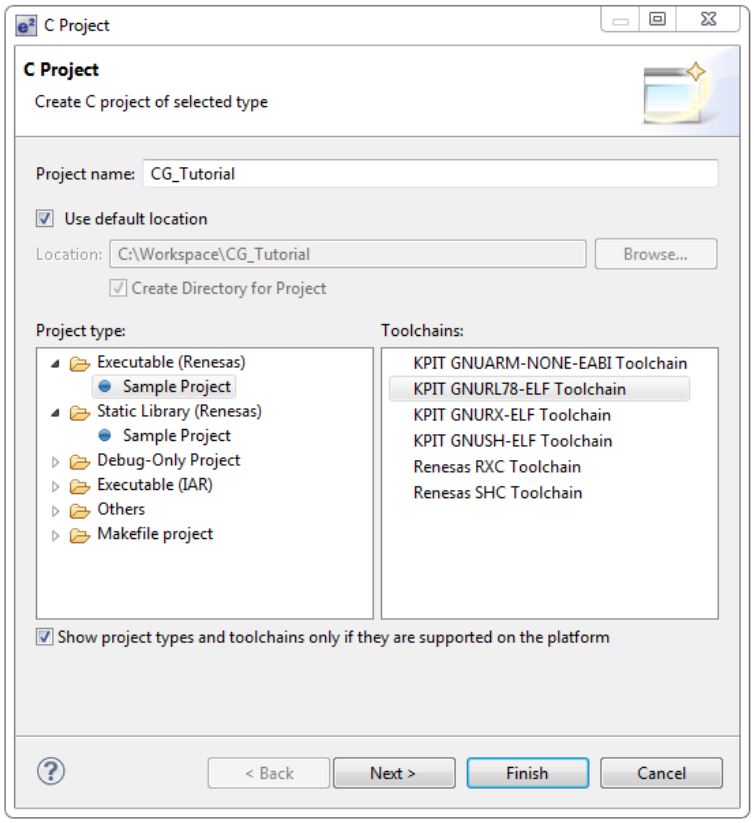


- In the Welcome page, click 'Go to the workbench'.

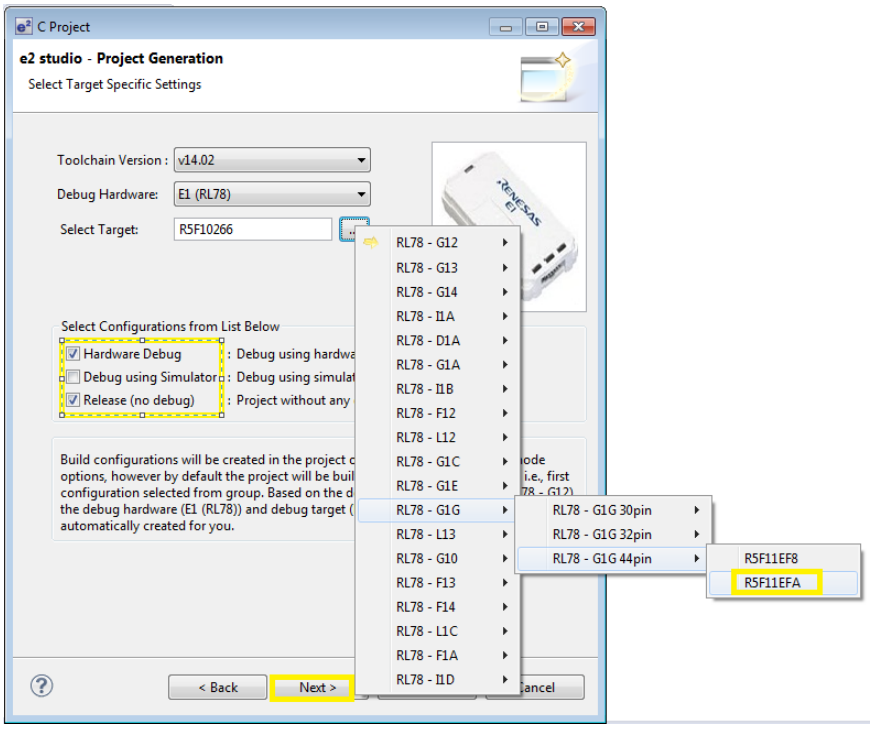
· Create a new C project by right-clicking in the Project Explorer pane and selecting ‘New -> C Project’ as shown. Alternatively, use the menu item ‘File -> New -> C Project’.



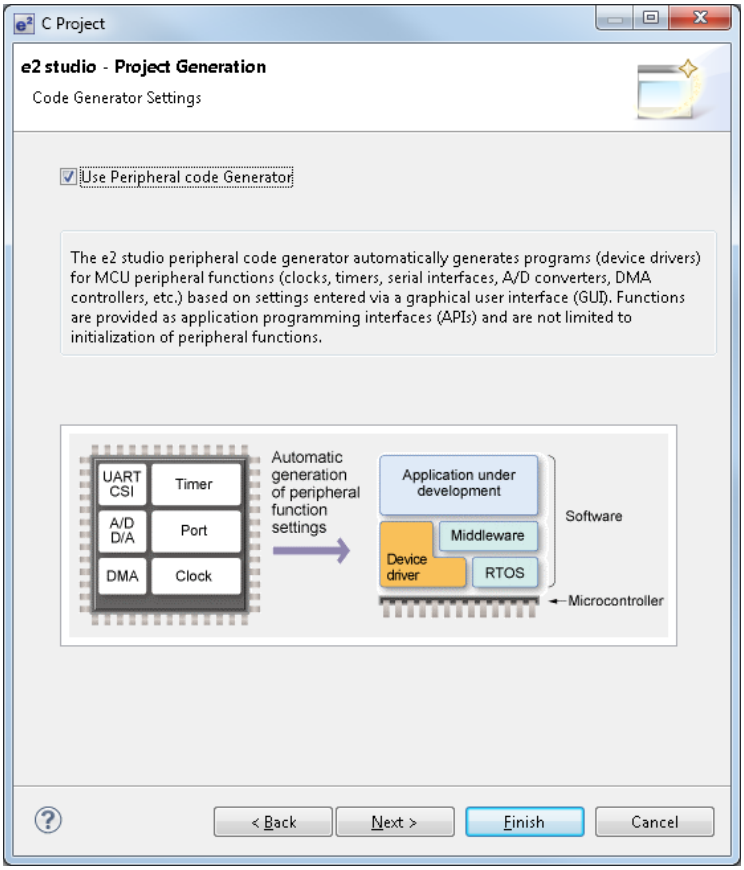
· Enter the project name ‘CG_Tutorial’. In ‘Project type:’ choose ‘Sample Project’. In ‘Toolchains’ choose ‘KPIT GNURL78-ELF Toolchain’. Click ‘Next’.



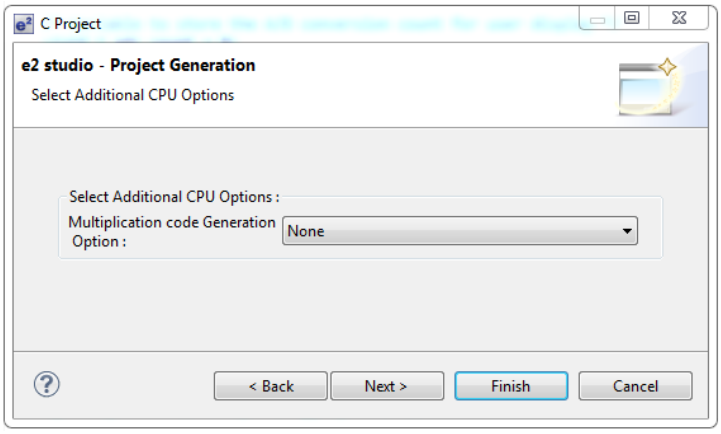
- In the 'Target Specific Settings' dialog, select the options as shown in the screenshot opposite.
- Click 'Next'.



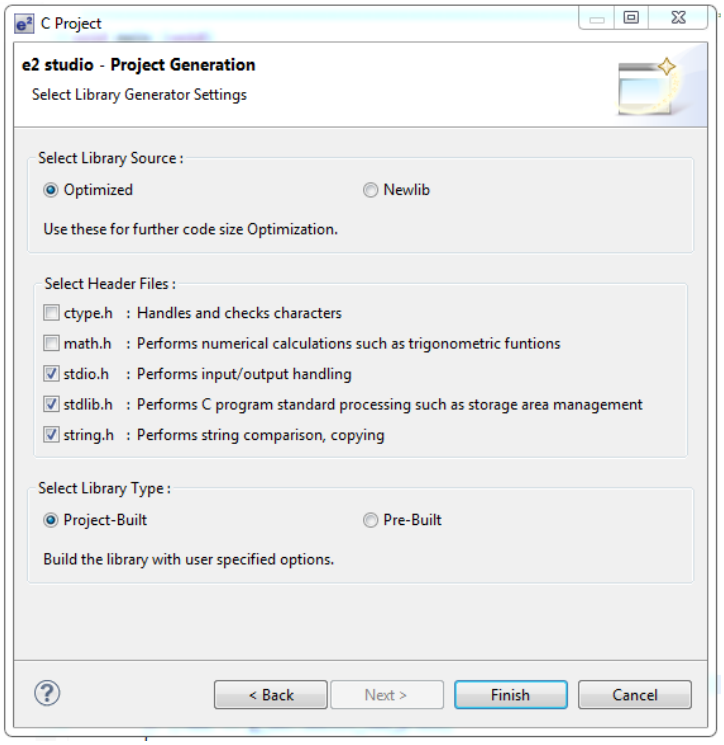
- In the 'Code Generator Settings' dialog, ensure the 'Use Peripheral code Generator' is checked.
- Click 'Next'.



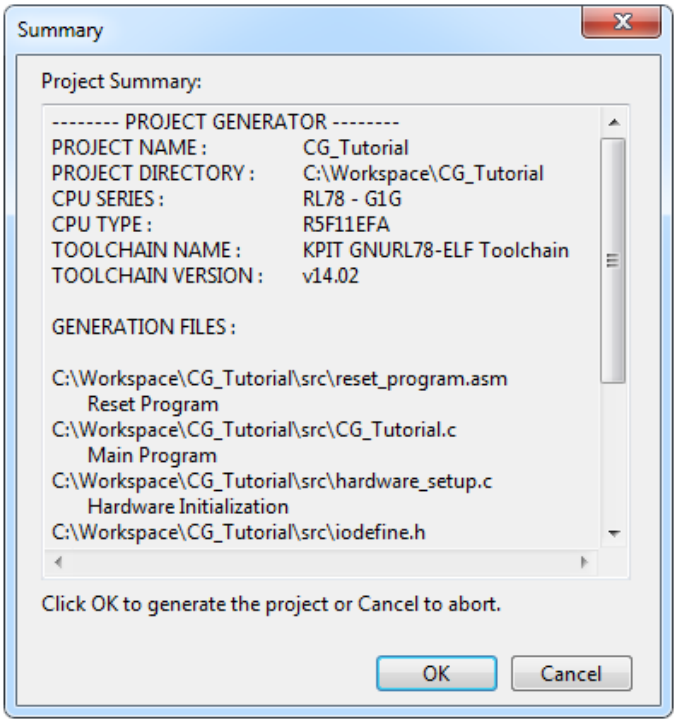
- In ‘Select Additional CPU Options’ leave everything at default values.
- Click ‘Next’.



- In the ‘Select Library Generation Settings’ dialog, leave all at defaults.
- Click ‘Finish’.



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



4. Code Generation Using the e2 studio plug in

4.1 Introduction

Code Generator is an e2 studio plug in GUI tool for generating template 'C' source code for the RL78/G1G. When using Code Generator, the user is able 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 e2 studio project called CG_Tutorial. A fully completed Tutorial project is contained on the DVD and may be imported into e2 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 e2 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 name '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 'adc'. 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 polls switch inputs and uses interrupts for the ADC module and the Serial Array Unit (SAU). These modules are used to perform A/D conversion and display the results via the UART in a terminal emulator running on the PC and also on the pmod LCD module connected to the CPU board. In addition a modulo 16 counter is maintained that counts the number of requested ADC conversions. The count results are displayed on the PC and they are also represented on LEDs 0 to 3.

Following a tour of the key user interface features of Code Generator in §4.2, the reader is guided through each of the peripheral function configuration dialogs in §4.3. In §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.

4.2 Code Generator Tour

This section presents a brief tour of Code Generator. For further details of the Code Generator paradigm and reference, refer to the Application Leading Tool Common Operations manual (r20ut2663ej0100). Application Leading Tool is the stand-alone version of Code Generator and this manual is applicable to the Code Generator.

From the e2 studio menus, select 'Window -> Open Perspective -> Other. In the 'Open Perspective' dialog shown in Figure 4-1, select 'Code Generator' and click 'OK'.

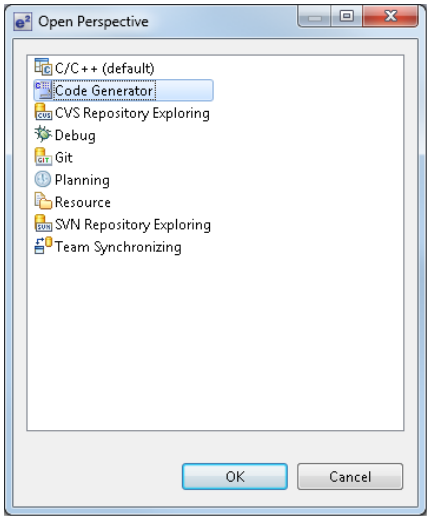


Figure 4-1 Open Perspective Dialog

In the Project Explorer pane, expand the 'Code Generator' and 'Peripheral Functions' node. The Code Generator initial view is displayed as illustrated in Figure 4-2.

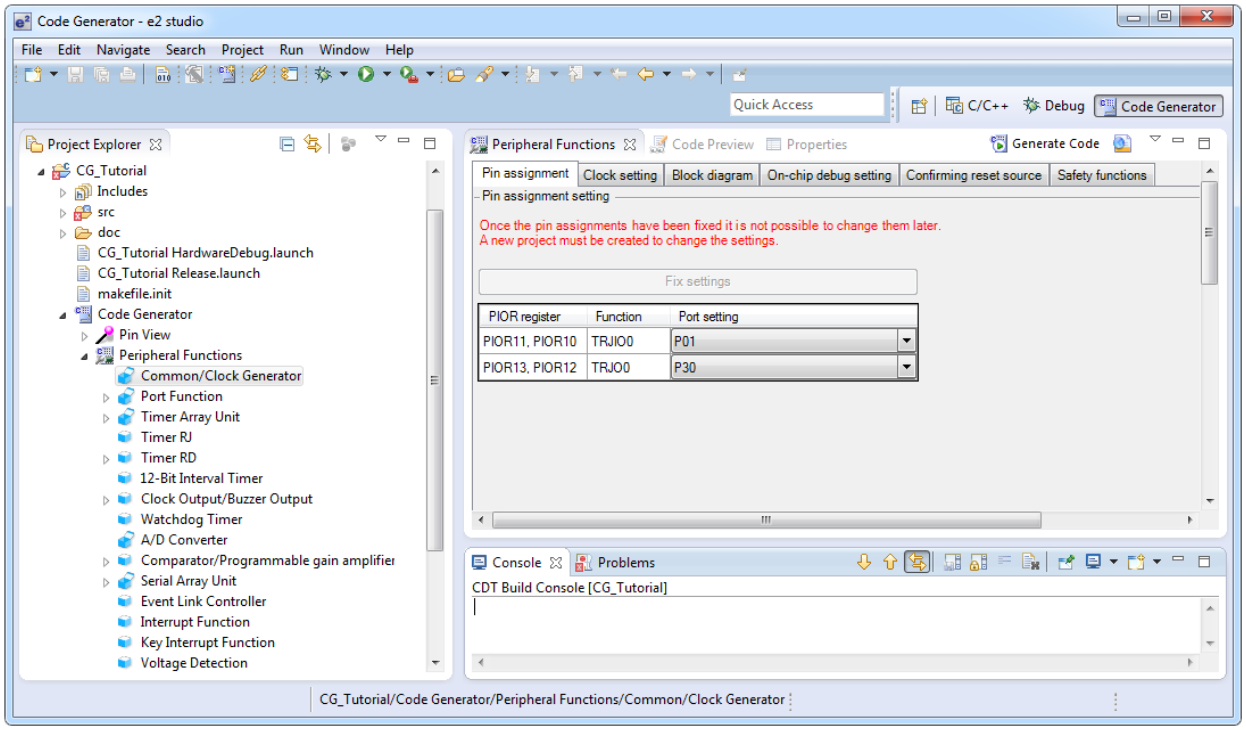


Figure 4-2 Initial View

Code Generator provides GUI features for configuration of MCU subsystems and peripherals. Once the user has configured all required MCU subsystems and peripherals, the user can click the 'Generate Code' button, resulting in a fully configured e2 studio project.

Navigation to the MCU peripheral configuration screens may be performed by double-clicking the required function in the Code Generator -> Peripheral Function on the left.

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 for a simple tutorial project containing ADC with external switch trigger, Serial Array Unit (SAU), Timer Array Unit (TAU) and LCD Output.

4.3.1 Common/Clock Generator

Certain MCU pins in the RL78/G1G are configurable for different peripheral functions. In order to proceed to setting up the MCU peripheral functions, the user must first fix these pin assignments using the 'Fix settings' button (see Figure 4-2). Once fixed, these pin assignments may not be changed and it will be necessary to create a new project if different pin assignments are required. For this RSK the default settings are applicable, click 'Fix settings' and the button will then be greyed out.

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

In this tutorial we are using the High-speed system clock with a 20 MHz crystal oscillator for the main clock source. The 'Block diagram' tab shows how clocks are distributed throughout the system.

Double click on the 'Clock Generator' entry in the Code Generator -> Peripheral Functions list.
Configure the Clock Generator options as shown in Figure 4-3.

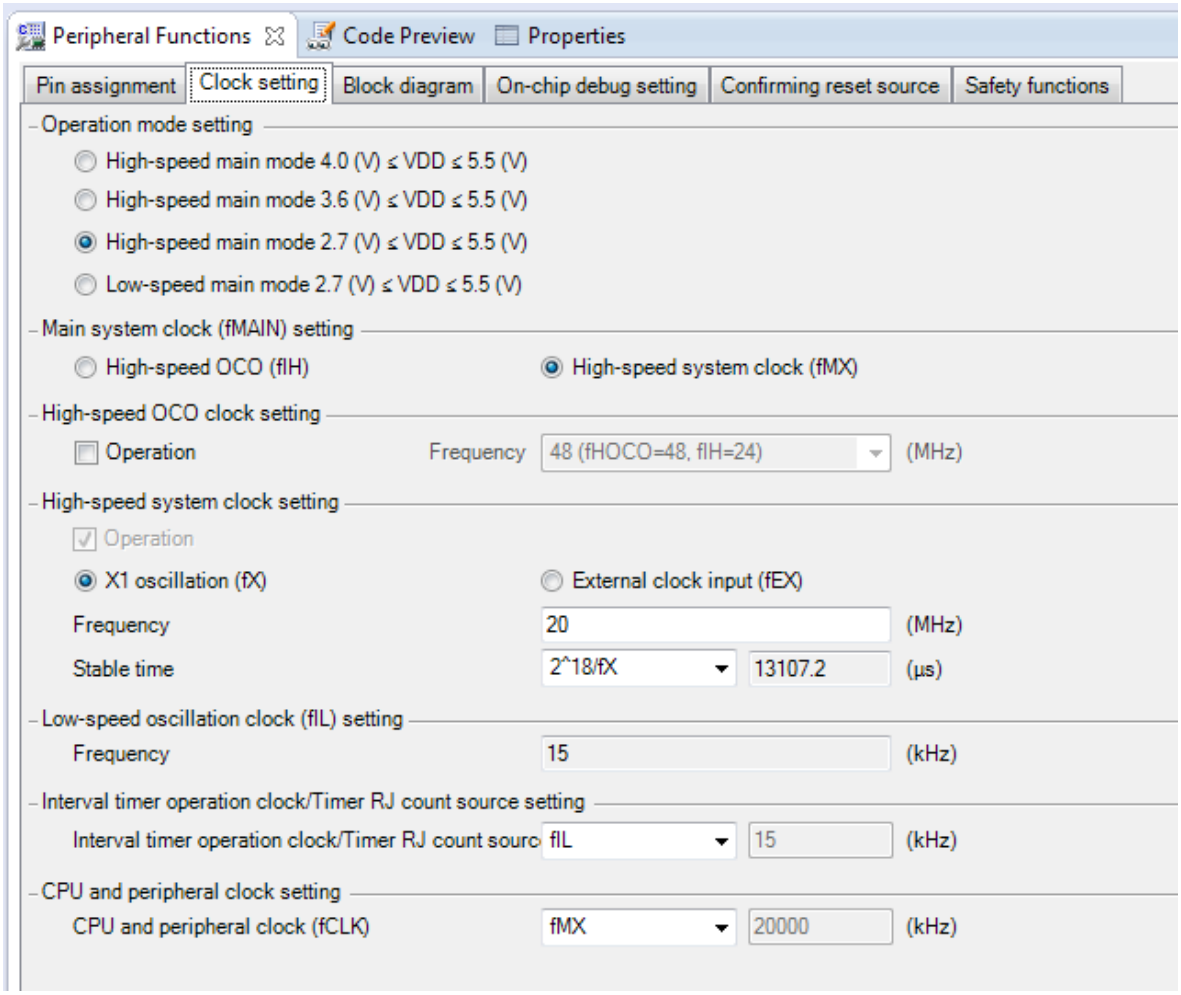


Figure 4-3 Clock setting tab

4.3.2 Port Function

This peripheral will be configured to assign output pins for user LEDs and input pins for user switches, with the exception of SW3 which is used as a trigger for the A/D Converter peripheral. Please refer to the RSK schematic for full details of the connectivity. A summary of those port settings is shown in Table 4-1

RSK component	Port	Configuration
SW1	P7.0	Input
SW2	P12.4	Input
SW3	P12.3	Input
LED0	P4.1	Output
LED1	P6.3	Output
LED2	P7.2	Output
LED3	P7.3	Output
PMOD	P6.1	Output
PMOD	P6.2	Output
PMOD	P7.1	Output

Table 4-1 RSK port configurations. The port number specifies a port and bit number of that port e.g. P7.0 indicates Port 7 bit 0.

Double click on the 'Port Function' entry in the Code Generator -> Peripheral Functions list.
All ports may be left with their default configurations except for ports 4, 6, 7 and 12. Select each of those port tabs and configure as shown in Figure 4-4 to Figure 4-8. Note that in order that the initial state of the LEDs is off (not illuminated) then the 'Output 1' tick box is selected for those ports connected to LEDs.

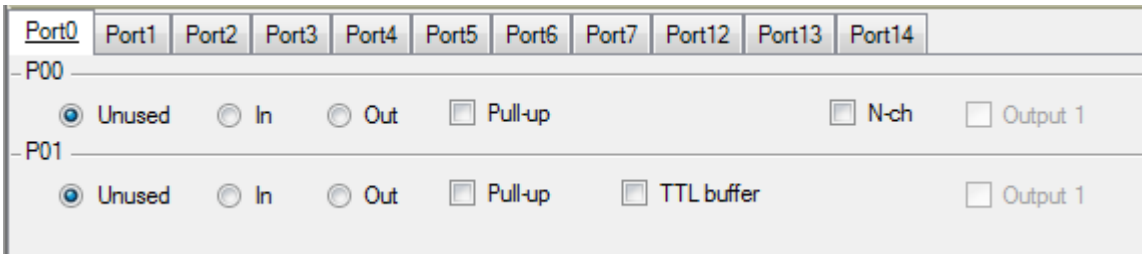


Figure 4-4 Port 0 Configuration.

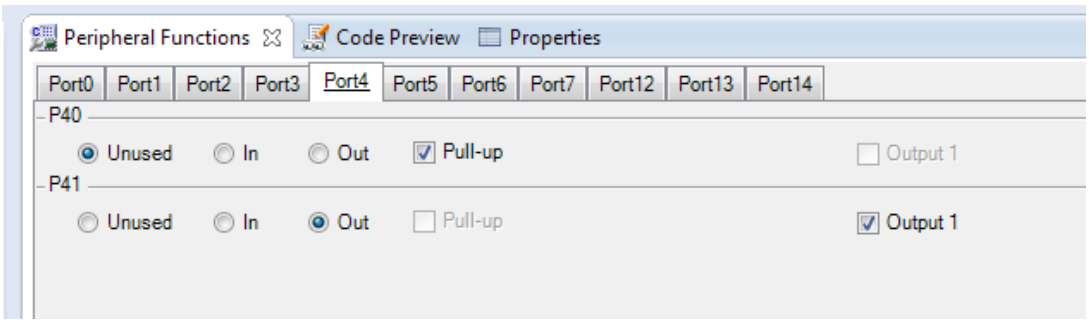


Figure 4-5 Port 4 Configuration

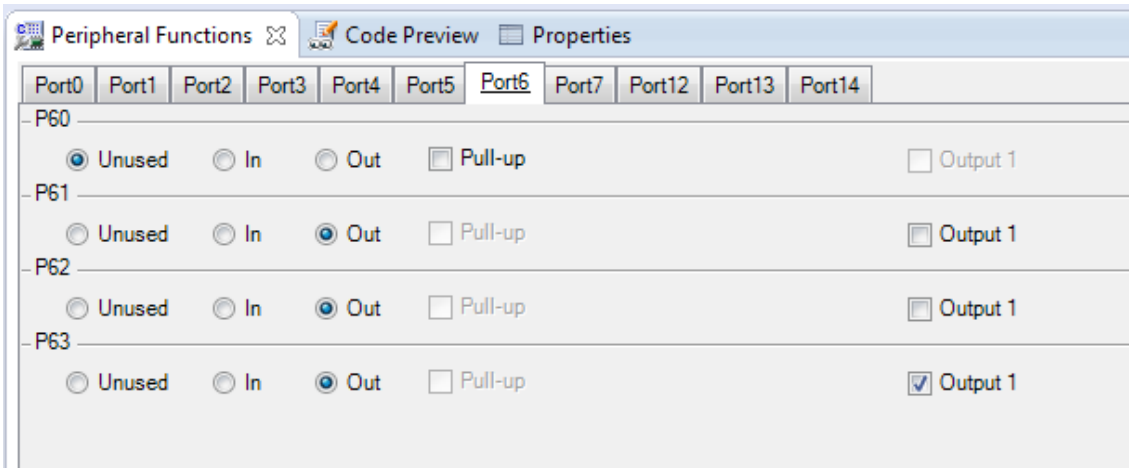


Figure 4-6 Port 6 Configuration

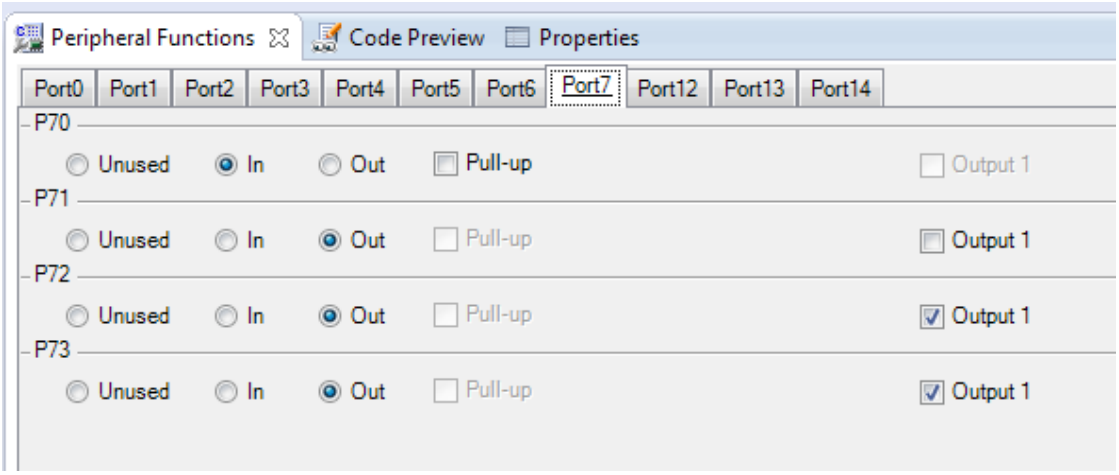


Figure 4-7 Port 7 Configuration

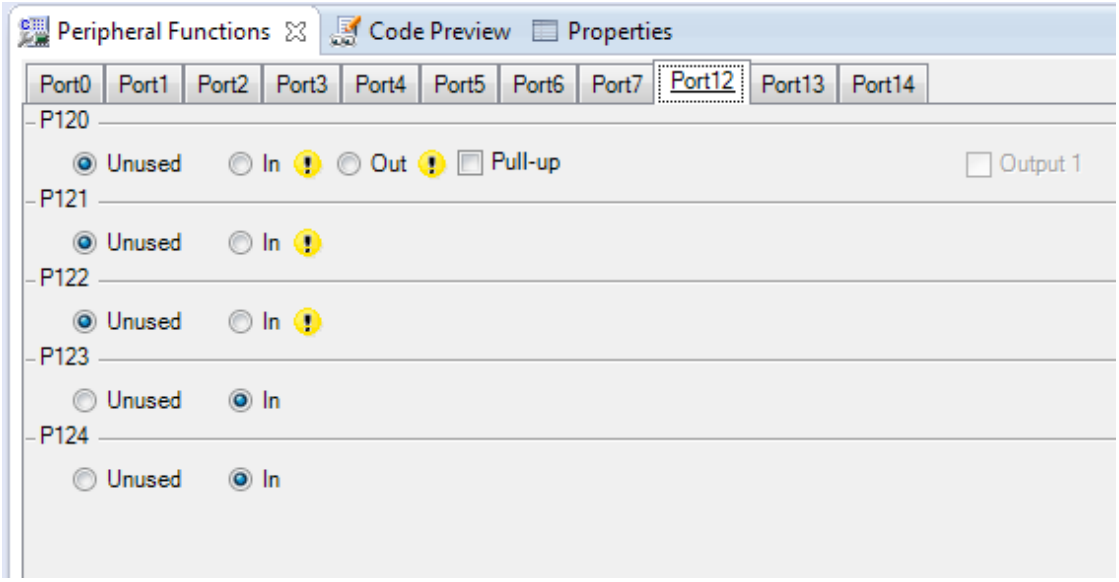


Figure 4-8 Port 12 Configuration. Note that e2 studio warns the user of any pin conflicts, all warnings may be ignored in this tutorial

4.3.3 Timer Array Unit

For this tutorial Channel 0 and Channel 2 are set up as 1ms interval timers. Double click on the 'Timer Array Unit' entry in the Code Generator -> Peripheral Functions list and configure as shown in Figure 4-9

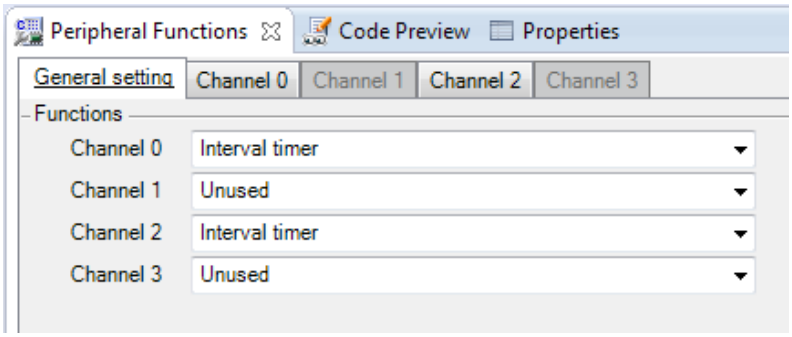


Figure 4-9 TAU channel 0 configured as a 1ms interval timer

Left click on the 'Channel 0' tab and configure as shown in Figure 4-10

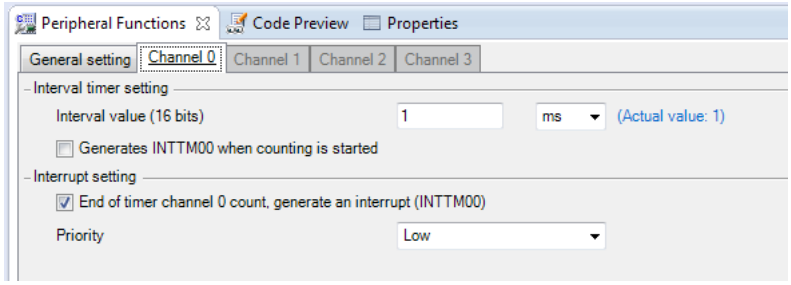


Figure 4-10 TAU Channel 0 Configuration

Left click on the 'Channel 2' tab and configure this in exactly the same way as Channel 0.

4.3.4 Watchdog Timer

The Watchdog Timer is enabled by default but it is not required in this project. Double click on 'Watchdog Timer' in the project tree and select 'Unused' for the Watchdog timer operation setting.

4.3.5 A/D Converter

For this tutorial the ADC is configured in 10-bit one shot mode on the ANI0 input, which is connected to the RV1 potentiometer output on the RSK.

Double click on the 'A/D Converter' entry in the Code Generator -> Peripheral Functions list and configure as shown in Figure 4-11

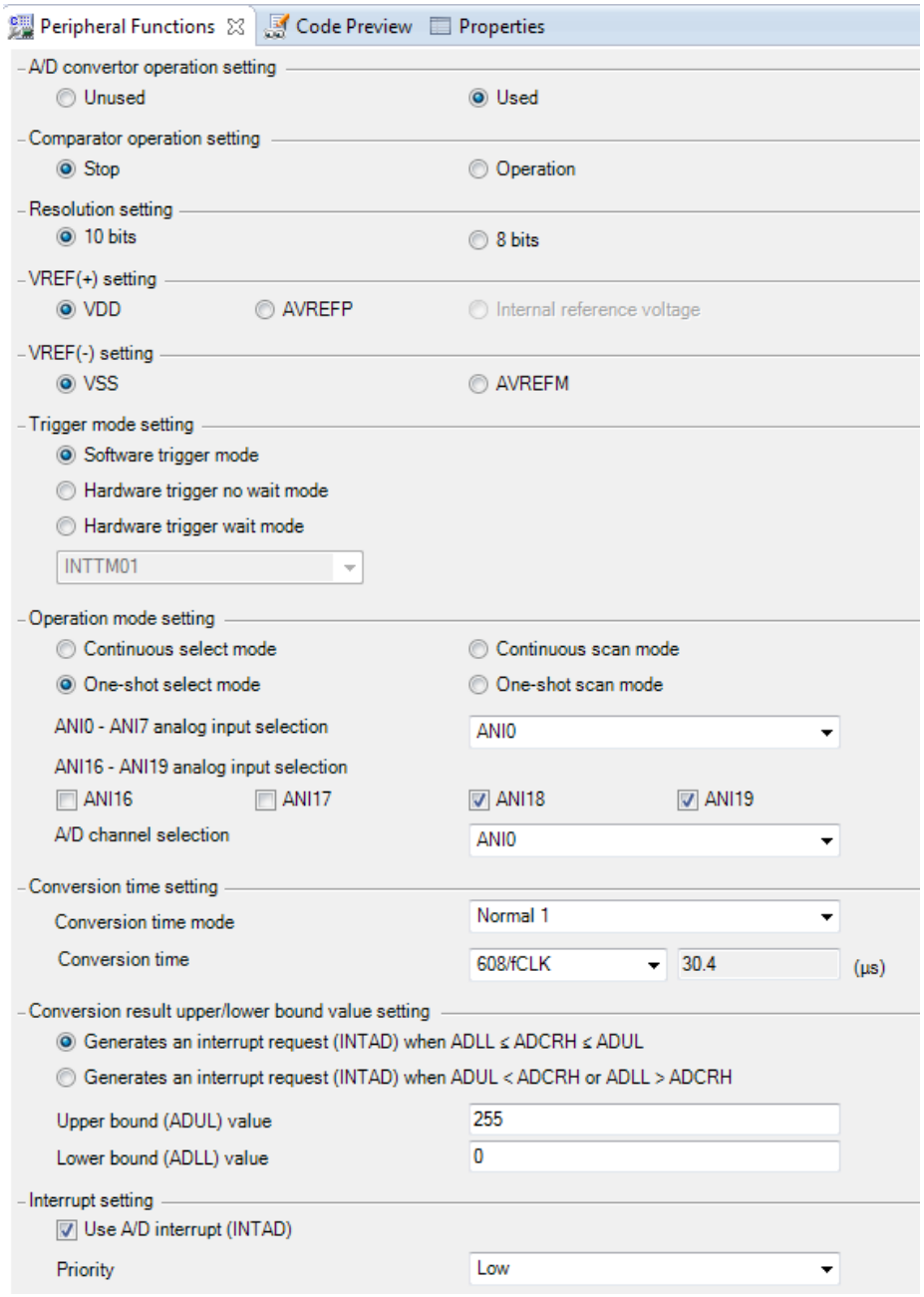


Figure 4-11 A/D Converter configuration

4.3.6 Serial Array Unit

The ‘Serial Array Unit’ (SAU) is used to communicate with both the pmod LCD module (via CSI00 on channel 0) and the PC (via UART1 on channel 2).

The UART1 lines TXD1 and RXD1 are connected to the RL78G1C, which is pre-configured as a serial to USB converter.

Double click on ‘Serial Array Unit’ in the project tree and configure the SAU channels as shown in Figure 4-12.

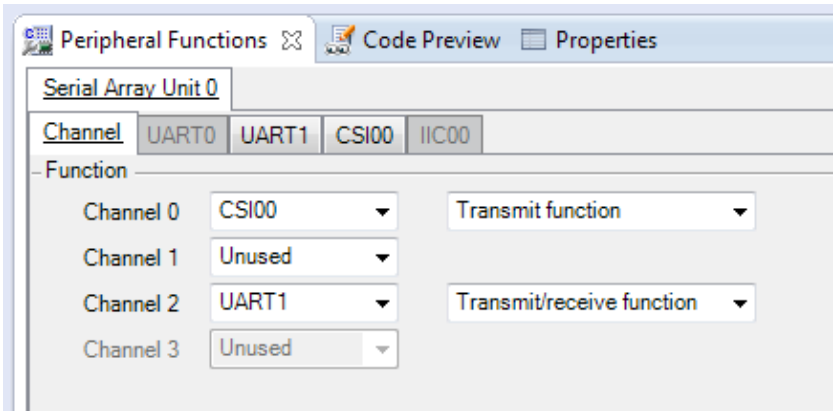


Figure 4-12 SAU channel configuration. Communications with the PMOD LCD module is via channel 0 (CSI00) and with the PC it is via channel 2 (UART1).

Left click on the CSI00 tab and configure as shown in Figure 4-13.

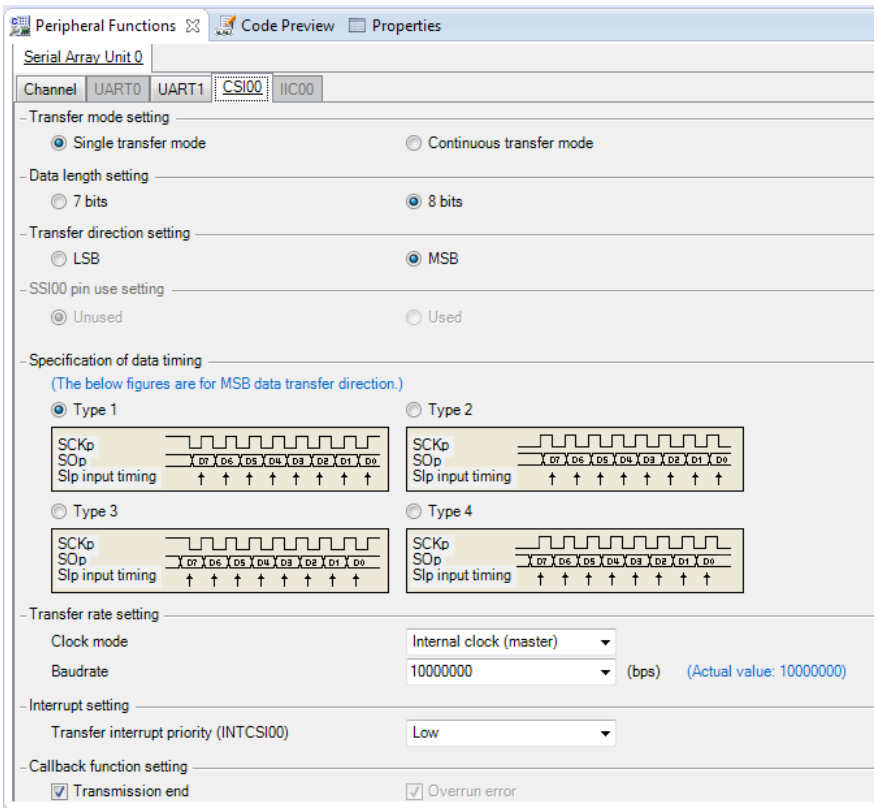


Figure 4-13 SAU CSI00 configuration.

Left click on the UART1 tab and configure as shown in Figure 4-14 and Figure 4-15.

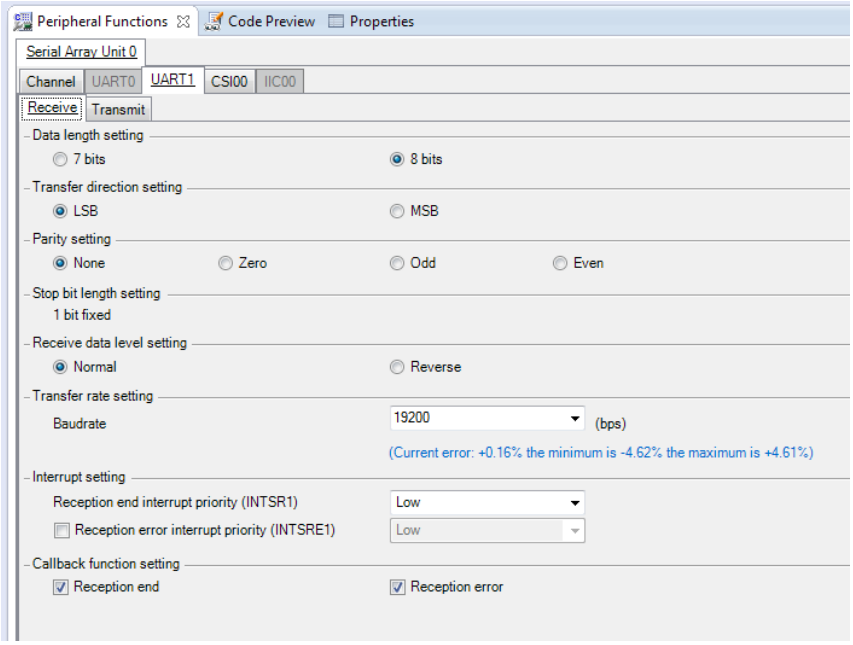


Figure 4-14 SAU UART1 Receive configuration (select Receive tab).

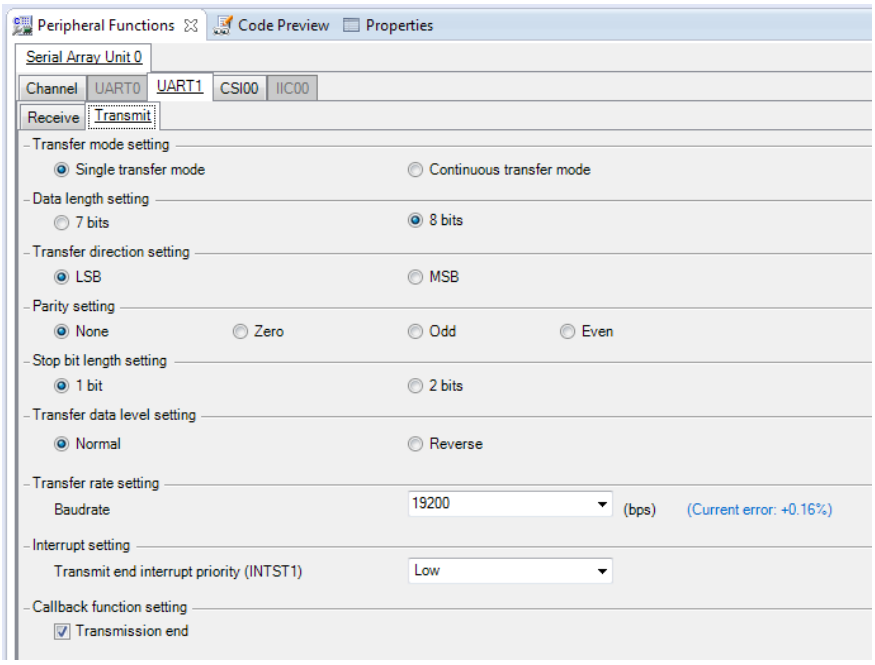


Figure 4-15 SAU UART1 Transmit configuration (select Transmit tab).

Code Generator configuration is now complete. Proceed to the next section to generate the code.

4.3.7 Generating the code

Peripheral function configuration is now complete. Click 'Generate Code' button located at the top right of the Peripheral Function tab. The Console pane should report 'The operation of generating file was successful', as shown Figure 4-16 below.

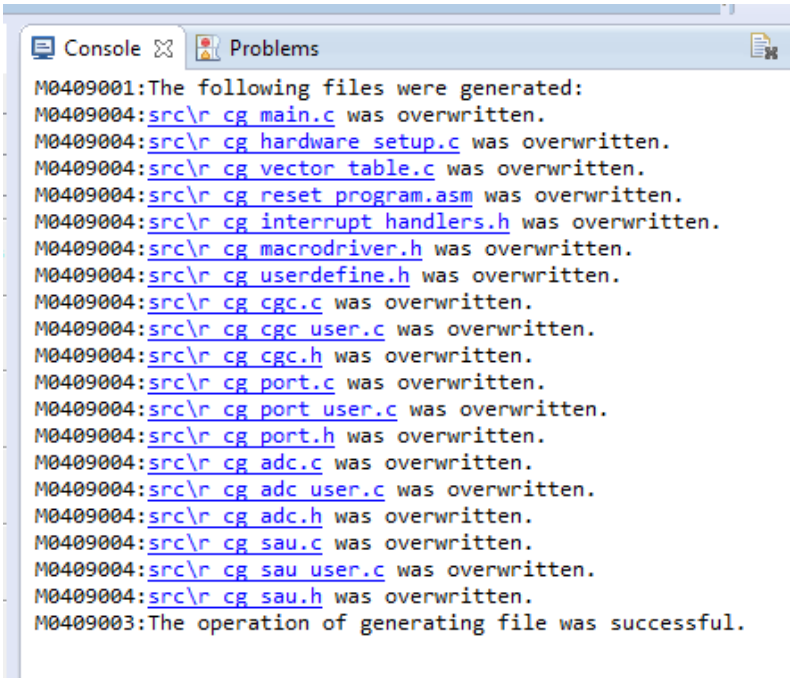


Figure 4-16 Code generator console

4.4 Building the Project

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

Seven files created by the New Project Wizard in §3.2 have been excluded from the build automatically as part of the code generation procedure as shown in Figure 4-17. This is because the main() function now resides in r_cg_main.c, type definitions and setting of sections has been handled by the Code Generator.

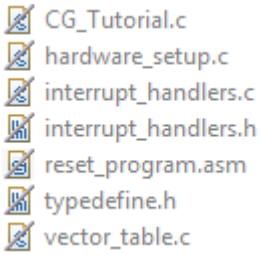


Figure 4-17 Files excluded from the build by Code Generator

Switch back to the 'C/C++' perspective using the 

button on the top right of the e2 studio workspace.

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

5. User Code Integration

At this stage of a typical project development the user would expand on the generated code to create the application required. As a demonstration this tutorial will include code lines and files from the complete 'Tutorial' project, supplied with the RSK.

The 'Tutorial' project is included as part of the RSK DVD installation process and can be found at the following location:

C:\Renesas\Workspace\RSK\RSKRL78G1G\Tutorial

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 refreshes the Code Generator-generated code.

5.1 Support file copying

RSK support and utility functions are provided in the following files:

r_ascii.h,
r_ascii.c,
r_lcd.h
r_lcd.c

Locate these files in the 'Tutorial' project and copy them in to the 'CG_Tutorial\src' folder. This will be located at the path specified during the project creation in section 3.2 The newly copied files should appear automatically in e^2 studio's Project Explorer window, if not then refresh the window as shown in Figure 5-1.

- In the e^2 studio Project Explorer, select the CG_Tutorial project. Right-click and select 'Refresh', or simply press F5. The files r_ascii.c, r_ascii.h, r_lcd.c and r_lcd.h will be added to the project in the 'src' folder.

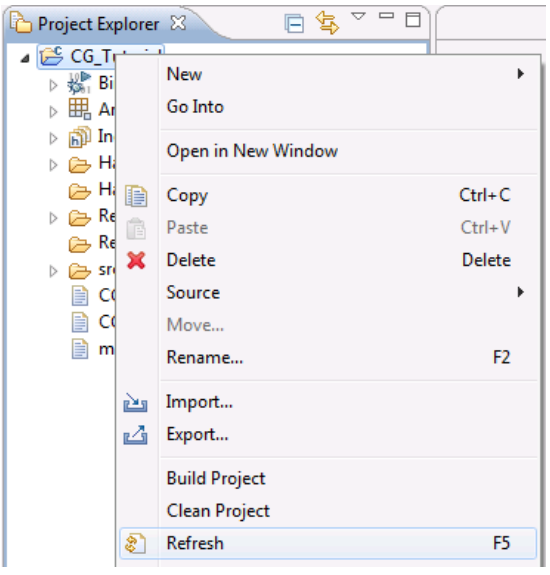


Figure 5-1 Refreshing Project Explorer files

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 Generated source file that needs to be opened by double clicking on the file name in e^2 studio's Project Tree window in the 'src tree.

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_main.c Code Insertion

Now it is time to modify the files that have been generated by CG.

Note that code must only be inserted into 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 if the code generator is run again. Note also that if the commented lines are modified, then that may also render the code susceptible to being overwritten.

In the e^2 studio Project Explorer, expand the 'src' folder and open the file 'r_cg_main.c' by double clicking on it. Modify the user code sections in the file so it matches the following; note that only the code between the 'start' and 'end' comments needs to be modified.

Insert the following between the user code delimiter comments as shown below in the file section designated Includes:

```
/* Start user code for include. Do not edit comment generated here */
#include <string.h>
#include "r_lcd.h"
/* End user code. Do not edit comment generated here */
```

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 global. Do not edit comment generated here */

/* Converts count to binary and displays on LEDs 0 to 3 */
static void led_display_count (const uint8_t count);

/* Read value from ADC. */
static uint16_t get_adc (void);

/* Read state of switches */
static void read_switch (volatile switch_t g_swn, unsigned char port_value);

/* Write to UART1 */
static void text_write (uint8_t * const msg_string);

/* Conversion to facilitate outputting to LCD module. */
static void uint16_to_string (uint8_t * const output_string, uint8_t pos, const uint16_t
input_number);

/* Prototype declaration for uart_display_adc */
static void uart_display_adc (uint8_t adc_count, uint16_t adc_result);

/* LCD module string buffer */
static uint8_t lcd_buf[10];

/* Variable for flagging user requested ADC conversion */
volatile uint8_t g_adc_trigger = FALSE;
```

```
/* Character received from PC terminal */
extern volatile uint8_t g_rx_char;

/* Commands to clear terminal window and set cursor to start of window */
const uint8_t g_cmd_clr_scr[] =
{ 27, '[', '2', 'J', 0 };
const uint8_t g_cmd_cur_home[] =
{ 27, '[', 'H', 0 };

/* ADC rx complete interrupt flag */
extern volatile uint8_t g_adc_rx_int;

/* UART1 serial transmission in progress */
extern volatile uint8_t g_uart1_tx_busy;

/* Debounce state */
extern volatile uint8_t g_debounce_ongoing;

/* Switches */
extern volatile switch_t g_sw3;

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

In the main function replace this code:

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

With this code:

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

    /* Variable to store the A/D conversion count for user display */
    uint8_t adc_count = 0;
    uint16_t adc_result;
    uint8_t initial_adc_meas = TRUE;

    /* Initialise the LCD display */
    init_lcd();

    /* Display test information */
    display_lcd(0, (uint8_t const *) "Renesas");
    display_lcd(1, (uint8_t const *) "RL78/G1G");
    display_lcd(3, (uint8_t const *) "Tutorial sample");
    display_lcd(4, (uint8_t const *) "Connect USB to PC");
    display_lcd(5, (uint8_t const *) "Serial configuration:");
    display_lcd(6, (uint8_t const *) "Baud Rate 19200");
    display_lcd(7, (uint8_t const *) "Data Bits 8");
    display_lcd(8, (uint8_t const *) "Stop Bits 1");
    display_lcd(9, (uint8_t const *) "Parity    None");
    display_lcd(10, (uint8_t const *) "Flow      None");

    /* Set up UART1 receive buffer and callback function */
    R_UART1_Receive((uint8_t * const) &g_rx_char, 1);

    /* Enable UART1 operations */
    R_UART1_Start();

    while (1U)
    {
        /* Read SW3. */
        read_switch(g_sw3, SW3_VALUE);

        /* If a new press of SW3 then request a new A/D conversion. */
        if (TRUE == g_sw3.switch_new_press)
        {
            g_sw3.switch_new_press = FALSE;

            /* set the flag indicating a user requested A/D conversion is required */
            g_adc_trigger = TRUE;
        }
```

```
        /* Wait for user requested A/D conversion flag to be set */
        if ((TRUE == g_adc_trigger) || (TRUE == initial_adc_meas))
        {
            /* Call the function to perform an A/D conversion  */
            adc_result = get_adc();

            /* Display the result on the LCD */
            uint16_to_string(lcd_buf, (uint8_t) 0, adc_result);
            display_lcd(12, (uint8_t const *) lcd_buf);

            /* Increment the adc_count and display using the LEDs if not the initial reading. */
            if (FALSE == initial_adc_meas)
            {
                if (16 == (++adc_count))
                {
                    adc_count = 0;
                }
            }
            led_display_count(adc_count);

            /* Send count and ADC result to the UART */
            uart_display_adc(adc_count, adc_result);

            /* Reset the flag */
            g_adc_trigger = FALSE;

            initial_adc_meas = FALSE;
        }
    }

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

Add the following code between the start and end user code comments at the end of the file.

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

/***********************************************************************************************
******************
 * Function Name : read_switch
 * Description   : If the switch state has changed then trigger the debounce timer, which will set
the debounced switch
 *                 state to pressed or released as appropriate. The calling program must set the new
press or new
 *                 released state to false once processed.
 * Argument      : none
 * Return value  : none

***********************************************************************************************
*****************/
static void read_switch (volatile switch_t g_sw, unsigned char port_value)
{
    /* Start TAU channel 0 timer (debounce timer) if switch state change detected. */
    if (((SWITCH_PRESSED == port_value) && (SWITCH_RELEASED == g_sw.current_switch_state))
            || ((SWITCH_RELEASED == port_value) && (SWITCH_PRESSED == g_sw.current_switch_state)))
    {
        /* TAU channel 0 only needs to be started if it has already been stopped */
        if (FALSE == g_debounce_ongoing)
        {
            g_debounce_ongoing = TRUE;

            /* Start TAU channel 0. which is configured as a periodic timer to aid switch debouncing.
*/
            R_TAU0_Channel0_Start();
        }
    }
}
/***********************************************************************************************
******************
 * End of function read_switch

***********************************************************************************************
*****************/

/***********************************************************************************************
******************
```

```
 * Function Name : get_adc
 * Description   : Reads the ADC result.
 * Argument      : none
 * Return value  : adc_result - Value of ADC conversion

*******************************************************************************************************
******************/
static uint16_t get_adc (void)
{
    uint16_t adc_result;

    /* Enable comparator operation */
    R_ADC_Set_OperationOn();

    /* Start a conversion */
    R_ADC_Start();

    /* Wait for the A/D conversion to complete */
    while (FALSE == g_adc_rx_int)
    {
        /* Wait */
    }
    g_adc_rx_int = FALSE;

    R_ADC_Get_Result(&adc_result);

    /* stops comparator operation */
    R_ADC_Set_OperationOff();

    /* stops the AD converter */
    R_ADC_Stop();

    return adc_result;
}
/*******************************************************************************************************
******************
 * End of function get_adc

*******************************************************************************************************
******************/

/*******************************************************************************************************
******************
 * Function Name : uart_display_adc
 * Description   : Converts adc result to a string and sends it to UART1.
 * Argument      : adc_count  - Number of ADC conversions (modulo 16)
 *                 adc result - Value of ADC conversion
 * Return value  : none

*******************************************************************************************************
******************/
static void uart_display_adc (uint8_t adc_count, uint16_t adc_result)
{
    uint8_t str1[50];

    /* Clear terminal window and set cursor to start of window */
    text_write((uint8_t *) &g_cmd_clr_scr);
    text_write((uint8_t *) &g_cmd_cur_home);

    strcpy((char *) str1, "ADC value =       \r\n");
    uint16_to_string(str1, (uint8_t) 12, (uint16_t) adc_result);
    text_write(str1);

    strcpy((char *) str1, "Number of ADC conversions (modulo 16) =      \r\n");
    uint16_to_string(str1, (uint8_t) 40, (uint16_t) adc_count);
    text_write(str1);
}
/*******************************************************************************************************
******************
 * End of function uart_display_adc

*******************************************************************************************************
******************/

/*******************************************************************************************************
******************
 * Function name : text_write
```

```
 * Description   : Transmits null-terminated string.
 * Argument      : msg_string - null terminated string
 * Argument      : None

***********************************************************************************************
*****************/
static void text_write (uint8_t * const msg_string)
{
    uint16_t i;

    for (i = 0; msg_string[i]; i++)
    {
        /* Send one byte and set UART transmit busy flag */
        R_UART1_Send(&msg_string[i], 1);
        g_uart1_tx_busy = TRUE;

        /* Wait until UART transfer is complete*/
        while (TRUE == g_uart1_tx_busy)
        {
            /* Wait */
        }
    }
}
/***********************************************************************************************
*******************
 * End of Function text_write

***********************************************************************************************
*****************/

/***********************************************************************************************
*******************
 * Function Name : led_display_count
 * Description   : Converts count to binary and displays on LEDs 0 to 3
 * Argument      : count - Number of ADC conversions (modulo 16)
 * Return value  : none
 ***********************************************************************************/
static void led_display_count (const uint8_t count)
{

    /* Set LEDs according to lower nibble of count parameter */
    LED0 = (count & 0x01) ? LED_ON : LED_OFF;
    LED1 = (count & 0x02) ? LED_ON : LED_OFF;
    LED2 = (count & 0x04) ? LED_ON : LED_OFF;
    LED3 = (count & 0x08) ? LED_ON : LED_OFF;
}
/***********************************************************************************************
*******************
 * End of function led_display_count

***********************************************************************************************
*****************/

/***********************************************************************************************
*******************
 * Function Name: uint16_to_string
 * Description  : Function converts a 16 bit integer into a character string, inserts it into the
array via the pointer
 *                  passed at execution.
 * Argument     :   output_string - Pointer to char array that will hold character string.
 *                          pos - uint8_t number, element number to begin inserting the character
string from (offset).
 *                   input_number - 16 bit integer to convert into a string.
 * Return value : none
 * Note         : No input validation is used, so output data can overflow the array passed.

***********************************************************************************************
*****************/
static void uint16_to_string (uint8_t * const output_string, uint8_t pos, const uint16_t
input_number)
{

    /* Declare temporary character storage variable, and bit_shift variable */
    uint8_t a = 0x00;
    uint8_t bit_shift = 12u;

    /* Declare 16bit mask variable */
```

```
    uint16_t mask = 0xF000;

    /* Loop through until each hex digit is converted to an ASCII character */
    while (bit_shift < 30u)
    {
        /* Mask and shift the hex digit, and store in temporary variable, a */
        a = (uint8_t) ((input_number & mask) >> bit_shift);

        /* Convert the hex digit into an ASCII character, and store in output
         string */
        output_string[pos] = (uint8_t) ((a < 0x0A) ? (a + 0x30) : (a + 0x37));

        /* Shift the bit mask 4 bits to the right, to convert the next digit */
        mask = (uint16_t) (mask >> 4u);

        /* Decrement the bit_shift counter by 4 (bits in a each digit) */
        bit_shift = (uint8_t) (bit_shift - 4u);

        /* Increment the output string location */
        pos++;
    }
}
/***********************************************************************************************
******************
 * End of function uint16_to_string

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

5.2.2 r_cg_adc_user.c Code Insertion

In the e^2 studio Project Explorer, expand the ‘src’ folder and open the file ‘r_cg_adc_user.c’ by double-leftclicking on it.

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

```
/***********************************************************************************************
******************
Global variables and functions
***********************************************************************************************
******************/
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_adc_rx_int = FALSE;

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

Insert the following in to the function r_adc_interrupt.

```
    /* Start user code. Do not edit comment generated here */
    g_adc_rx_int = TRUE;

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

5.2.3 r_cg_sau.h Code Insertion

In the e^2 studio Project Explorer, expand the ‘src’ folder and open the file ‘r_cg_sau.h’ by double-leftclicking on it.

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

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

void send_csi0 (uint8_t * const tx_buf, uint16_t const tx_num);
uint8_t csi0_tx_is_busy (void);

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

5.2.4 r_cg_sau.c Code Insertion

In the e^2 studio Project Explorer, expand the 'src' folder and open the file 'r_cg_sau.c' by double-leftclicking on it.

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 global. Do not edit comment generated here */

volatile uint8_t g_csi0_tx_in_process = FALSE;

/* 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: send_csi0
 * Description  : This function sends CSI00 data to slave device. Adds flagging around R_CSI00_Send
 * Arguments    : tx_buf -
 *                    transfer buffer pointer (Not used when transmit data handled by DTC)
 *                tx_num -
 *                    buffer size
 * Return Value : status -
 *                    MD_OK or MD_ARGERROR

***********************************************************************************************
*****************/
void send_csi0 (uint8_t * const tx_buf, uint16_t const tx_num)
{
    g_csi0_tx_in_process = TRUE;
    R_CSI00_Send(tx_buf, tx_num);
}
/***********************************************************************
 * End of function send_csi0
 ***********************************************************************/

/***********************************************************************
 * Function Name : csi0_tx_is_busy
 * Description   : reports if sci6 is transmitting
 * Argument      : none
 * Return value  : None
 ***********************************************************************/
uint8_t csi0_tx_is_busy (void)
{
    return (g_csi0_tx_in_process);
}
/***********************************************************************
 * End of function csi0_tx_is_busy
 ***********************************************************************/
/* End user code. Do not edit comment generated here */
```

5.2.5 r_cg_sau_user.c Code Insertion

In the e^2 studio Project Explorer, expand the 'src' folder and open the file 'r_cg_sau_user.c' by double-leftclicking on it.

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 global. Do not edit comment generated here */
extern volatile uint8_t g_csi0_tx_in_process;
extern volatile uint8_t g_adc_trigger;

/* UART1 serial transmission in progress */
volatile uint8_t g_uart1_tx_busy = FALSE;

/* Character received from PC terminal                    */
```

```
volatile uint8_t g_rx_char;

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

Insert the following in to the function r_uart1_callback_receiveend.

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

    /* Check the character received from the PC */
    if (('c' == g_rx_char) || ('C' == g_rx_char))
    {
        g_adc_trigger = TRUE;
    }

    /* Set up UART1 receive buffer and callback function again */
    R_UART1_Receive((uint8_t * const) &g_rx_char, 1);

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

Insert the following in to the function r_uart1_callback_sendend.

```
    /* Start user code. Do not edit comment generated here */
    /* UART1 serial transmission finished */
    g_uart1_tx_busy = FALSE;

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

Insert the following in to the function r_csi00_callback_sendend.

```
    /* Start user code. Do not edit comment generated here */
    g_csi0_tx_in_process = FALSE;

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

5.2.6 r_cg_userdefine.h Code Insertion

In the e^2 studio Project Explorer, expand the ‘src’ folder and open the file ‘r_cg_userdefine.h’ by double-left clicking on it.

Insert the following between the user code delimiter comments as shown below in the file section designated User definitions.

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

/* Switch port settings */
#define SW1             (1)
#define SW2             (2)
#define SW3             (3)
#define SW1_VALUE       (P7_bit.no0)
#define SW2_VALUE       (P12_bit.no4)
#define SW3_VALUE       (P12_bit.no3)
#define SWITCH_PRESSED  (0)
#define SWITCH_RELEASED (1)

/* Switch debounce settings */
#define PRESSED_DEBOUNCE_COUNT  (10)
#define RELEASED_DEBOUNCE_COUNT (20)

/* LED port settings */
#define LED0            (P4_bit.no1)
#define LED1            (P6_bit.no3)
#define LED2            (P7_bit.no2)
#define LED3            (P7_bit.no3)

/* LED lights.  */
#define LED_ON          (0)
#define LED_OFF         (1)

#define TRUE            (1)
#define FALSE           (0)

/* Switches */
typedef struct
```

```
{
    uint8_t current_switch_state;
    uint8_t switch_new_press;
    uint8_t switch_new_release;
    uint8_t debounce_counter;
} switch_t;

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

5.2.7 r_cg_tau_user.c Code Insertion

In the e^2 studio Project Explorer, expand the 'src' folder and open the file 'r_cg_tau_user.c' by double-leftclicking on it.

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 global. Do not edit comment generated here */

/* Debounce state */
volatile uint8_t g_debounce_ongoing = FALSE;

/* Switches */
volatile switch_t g_sw1 =
{ SWITCH_RELEASED, FALSE, FALSE, 0 };
volatile switch_t g_sw2 =
{ SWITCH_RELEASED, FALSE, FALSE, 0 };
volatile switch_t g_sw3 =
{ SWITCH_RELEASED, FALSE, FALSE, 0 };

/* TAU0 channel2 interrupt count */
volatile uint16_t g_tau_ch2_cnt = 0;

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

Insert the following in to the function r_tau0_channel0_interrupt.

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

    /* This ISR will debounce switches SW1, SW2 and SW3. The debounce algorithm will check that the
switch state (either
     * pressed or released is stable over a defined period, which can be modified at compile time.
The debounce time
     * for pressing and releasing can be independently configured. Once a switch is pressed or
released then the state
     * of the switch is sampled over the predefined time; a counter is incremented every time the
sampled signal is the
     * same as the previous state. On reaching the end of the debounce period, if it has been stable
for the whole
     * period then the change in switch state is deemed to be valid (debounced) and the new state is
updated. If the
     * sampled switch state is not the same as the previous one then the counter is reset and
counting recommences.
     * This timer will start when any of the switches have been pressed or released and will stop
only after all
     * switches have been debounced. */

    /* Check the last current stable state of SW1. */
    if (SWITCH_RELEASED == g_sw1.current_switch_state)
    {
        /* Switch is in the RELEASED state so it must have been pressed. Read switch input value,
clear debounce counter
         * if switch has bounced back to the release position (open), else increment debounce counter
and confirm new
         * state and new switch pressed once debounce count is reached. */
        if (SWITCH_RELEASED == SW1_VALUE)
        {
            g_sw1.debounce_counter = 0;
        }
        else
        {
            g_sw1.debounce_counter++;
```

```
            /* If at the end of the debounce period, then update the current state and indicate that
a new press has
             * been detected. */
            if (PRESSED_DEBOUNCE_COUNT == g_sw1.debounce_counter)
            {
                g_sw1.current_switch_state = SWITCH_PRESSED;
                g_sw1.switch_new_press = TRUE;
            }
        }
    }
    else
    {
        if (SWITCH_PRESSED == g_sw1.current_switch_state)
        {
            /* Switch is in the PRESSED state so it must have been released. Read switch input value,
clear debounce
             * counter if switch has bounced back to the pressed position (closed), else increment
debounce counter and
             * confirm new state and new switch released once debounce count is reached. */
            if (SWITCH_PRESSED == SW1_VALUE)
            {
                g_sw1.debounce_counter = 0;
            }
            else
            {
                g_sw1.debounce_counter++;

                /* If at the end of the debounce period, then update the current state and indicate
that a new release
                 * has been detected. */
                if (RELEASED_DEBOUNCE_COUNT == g_sw1.debounce_counter)
                {
                    g_sw1.current_switch_state = SWITCH_RELEASED;
                    g_sw1.switch_new_release = TRUE;
                }
            }
        }
    }

    /* Check the last current stable state of SW2. */
    if (SWITCH_RELEASED == g_sw2.current_switch_state)
    {
        /* Switch is in the RELEASED state so it must have been pressed. Read switch input value,
clear debounce counter
         * if switch has bounced back to the release position (open), else increment debounce counter
and confirm new
         * state and new switch pressed once debounce count is reached. */
        if (SWITCH_RELEASED == SW2_VALUE)
        {
            g_sw2.debounce_counter = 0;
        }
        else
        {
            g_sw2.debounce_counter++;

            /* If at the end of the debounce period, then update the current state and indicate that
a new press has
             * been detected. */
            if (PRESSED_DEBOUNCE_COUNT == g_sw2.debounce_counter)
            {
                g_sw2.current_switch_state = SWITCH_PRESSED;
                g_sw2.switch_new_press = TRUE;
            }
        }
    }
    else
    {
        if (SWITCH_PRESSED == g_sw2.current_switch_state)
        {
            /* Switch is in the PRESSED state so it must have been released. Read switch input value,
clear debounce
             * counter if switch has bounced back to the pressed position (closed), else increment
debounce counter and
             * confirm new state and new switch released once debounce count is reached. */
            if (SWITCH_PRESSED == SW2_VALUE)
            {
                g_sw2.debounce_counter =                    0;
```

```
            }
            else
            {
                g_sw2.debounce_counter++;

                /* If at the end of the debounce period, then update the current state and indicate
that a new release
                 * has been detected. */
                if (RELEASED_DEBOUNCE_COUNT == g_sw2.debounce_counter)
                {
                    g_sw2.current_switch_state = SWITCH_RELEASED;
                    g_sw2.switch_new_release = TRUE;
                }
            }
        }
    }


    /* Check the last current stable state of SW3. */
    if (SWITCH_RELEASED == g_sw3.current_switch_state)
    {
        /* Switch is in the RELEASED state so it must have been pressed. Read switch input value,
clear debounce counter
         * if switch has bounced back to the release position (open), else increment debounce counter
and confirm new
         * state and new switch pressed once debounce count is reached. */
        if (SWITCH_RELEASED == SW3_VALUE)
        {
            g_sw3.debounce_counter = 0;
        }
        else
        {
            g_sw3.debounce_counter++;

            /* If at the end of the debounce period, then update the current state and indicate that
a new press has
             * been detected. */
            if (PRESSED_DEBOUNCE_COUNT == g_sw3.debounce_counter)
            {
                g_sw3.current_switch_state = SWITCH_PRESSED;
                g_sw3.switch_new_press = TRUE;
            }
        }
    }
    else
    {
        if (SWITCH_PRESSED == g_sw3.current_switch_state)
        {
            /* Switch is in the PRESSED state so it must have been released. Read switch input value,
clear debounce
             * counter if switch has bounced back to the pressed position (closed), else increment
debounce counter and
             * confirm new state and new switch released once debounce count is reached. */
            if (SWITCH_PRESSED == SW3_VALUE)
            {
                g_sw3.debounce_counter = 0;
            }
            else
            {

                g_sw3.debounce_counter++;

                /* If at the end of the debounce period, then update the current state and indicate
that a new release
                 * has been detected. */
                if (RELEASED_DEBOUNCE_COUNT == g_sw3.debounce_counter)
                {
                    g_sw3.current_switch_state = SWITCH_RELEASED;
                    g_sw3.switch_new_release = TRUE;
                }
            }
        }
    }

    /* Stop TAU channel 0 timer if no switches are in the process of being debounced */
    if (((0 == g_sw1.debounce_counter) && (0 == g_sw2.debounce_counter)) && (0 ==
g_sw3.debounce_counter))
```

```
{
    g_debounce_ongoing = FALSE;
    R_TAU0_Channel0_Stop();
}

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

Insert the following in to the function r_tau0_channel2_interrupt.

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

/* TAU0 channel2 interrupt count */
g_tau_ch2_cnt++;

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

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 §6.

6.Debugging 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 6-1 will be displayed.

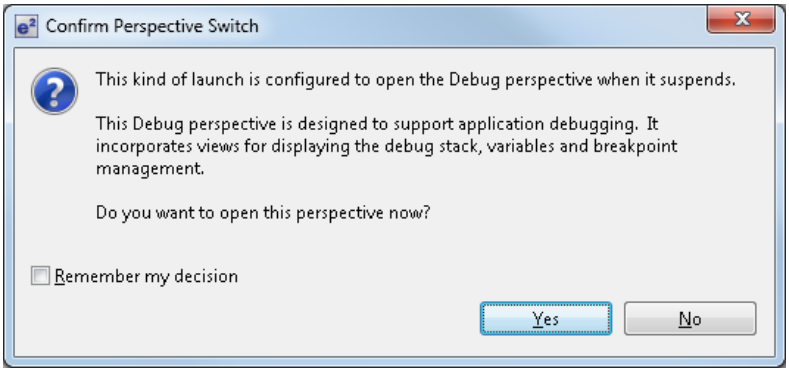


Figure 6-1 Perspective Switch Dialog

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 display the following on the pmod display:

Renesas
RL78/G1G

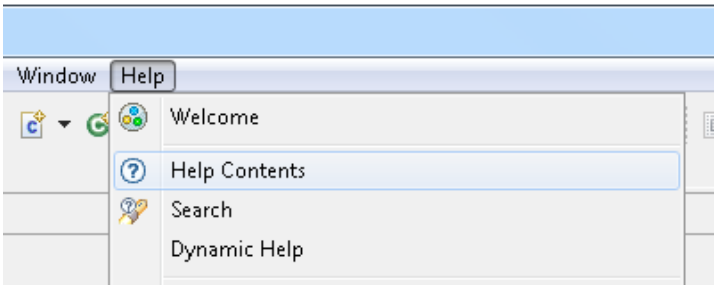
Tutorial sample
Connect USB to PC
Serial configuration:
Baud Rate 19200
Data Bits 8
Stop Bits 1
Parity None
Flow None

Pressing SW3 or entering the character ‘C’ or ‘c’ in the serial terminal window will trigger an ADC conversion and display the resulting value on the terminal window and the LCD. In addition a modulo 16 counter is maintained that counts the number of requested ADC conversions. The count results are displayed on the PC and they are also represented on LEDs 0 to 3.

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

7. 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 RL78/G1X group microcontroller refer to the RL78/G1X Group Hardware Manual.

For information about the RL78 assembly language, refer to the RL78 Series Software Manual.

Technical Contact Details

Please refer to the contact details listed in section 9 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

REVISION HISTORY	RSK RL78/G1G Code Generator Tutorial Manual (e2 studio)

Rev.	Date	Description	
		Page	Summary
1.00	Jan 09, 2015	¾	First Edition issued

Renesas Starter Kit Manual: Code Generator Tutorial Manual

Publication Date: Rev. 1.00 Jan 09, 2015

Published by: Renesas Electronics Corporation

Renesas Electronics Corporation

http://www.renesas.com

SALES OFFICES

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

RL78/G1G Group

Renesas Electronics Corporation

R20UT3025EG0100