User’ s Manual

32

RX140 Group

Renesas Starter Kit for RX140 Smart Configurator Tutorial Manual For e^2 studio

RENESAS 32-Bit MCU
RX Family / RX100 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 Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev. 1.00 Jan 2022

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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. 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 shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended 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; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

1.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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 Smart Configurator for RX 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 RX140 microcontroller may be found in 'RX140 Group User's Manual: Hardware' and within the provided sample code. The setup procedure for the RSK Web installer is described in the Quick Start Guide.

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.

In this manual, the display may differ slightly from screen shots. There is no problem in reading this manual.

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 RX140 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.	Renesas Starter Kit for RX140 User's Manual	R20UT5026EG
Tutorial Manual	Provides a guide to setting up RSK environment, running sample code and debugging programs.	Renesas Starter Kit for RX140 Tutorial Manual	R20UT5030EG
Quick Start Guide	Provides simple instructions to setup the RSK and run the first sample.	Renesas Starter Kit for RX140 Quick Start Guide	R20UT5031EG
Smart Configurator Tutorial	Provides a guide to code generation and importing into the e^2 studio IDE.	Renesas Starter Kit for RX140 Smart Configurator Tutorial Manual	R20UT5032EG
Schematics	Full detail circuit schematics of the RSK.	Renesas Starter Kit for RX140 Schematics	R20UT5025EG
Hardware Manual	Provides technical details of the RX140 microcontroller.	RX140 Group User's Manual: Hardware	R01UH0905EJ

2. List of Abbreviations and Acronyms

Abbreviation	Full Form
ADC	Analog-to-Digital Converter
API	Application Programming Interface
bps	bits per second
CMT	Compare Match Timer
COM	COMmunications port referring to PC serial port
CPU	Central Processing Unit
E1 / E2 Lite	Renesas On-chip Debugging Emulator
GUI	Graphical User Interface
IDE	Integrated Development Environment
IRQ	Interrupt Request
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LSB	Least Significant Bit
LVD	Low Voltage Detect
MCU	Micro-controller Unit
MSB	Most Significant Bit
PC	Personal Computer
PLL	Phase-locked Loop
Pmod™	This is a Digilent Pmod™ Compatible connector. Pmod™ is registered to Digilent Inc. Digilent-Pmod_Interface_Specification
PSU	Power Supply Unit
RAM	Random Access Memory
ROM	Read Only Memory
RSK	Renesas Starter Kit
RTC	Real Time Clock
SCI	Serial Communications Interface
SPI	Serial Peripheral Interface
TFT	Thin Film Transistor
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WDT	Watchdog Timer

All trademarks and registered trademarks are the property of their respective owners.

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 Smart Configurator plug-in to create a working project for the RSK platform.

1.2 Features

This RSK provides an evaluation of the following features:

- Project Creation with e^2 studio.
- Code generation using the Smart Configurator 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 Smart Configurator plug-in for the RX 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 e^2 studio
- Detailed use of the Smart Configurator plug-in for e^2 studio
- Integration with custom code
- Building the project in 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 (level two) and ‘Outputs debugging information’ option not selected, producing code suitable for release in a product.

The tutorial examples in this manual assume that installation procedures described in the RSK Quick Start Guide have been completed. Please refer to the Quick Start Guide for details of preparing the configuration.

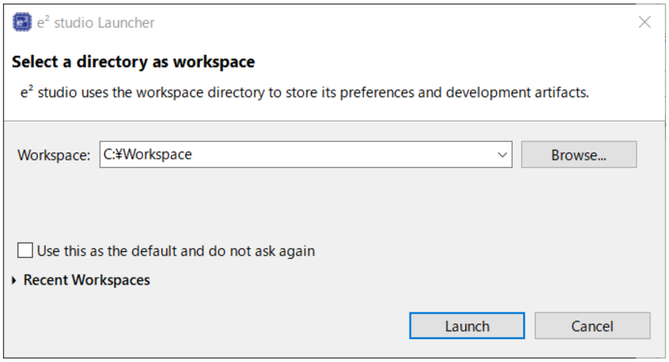
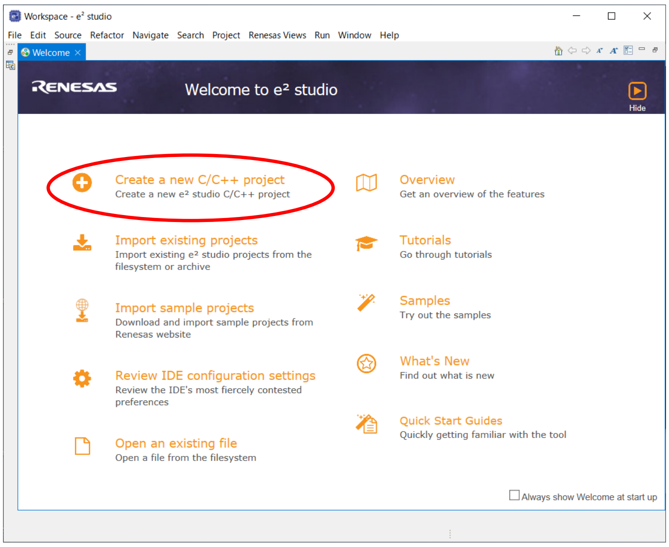
These tutorials are designed to show you how to use the RSK and are not intended as a comprehensive introduction to the e^2 studio debugger, compiler toolchains or the E2 emulator Lite. Please refer to the relevant user manuals for more in-depth information.

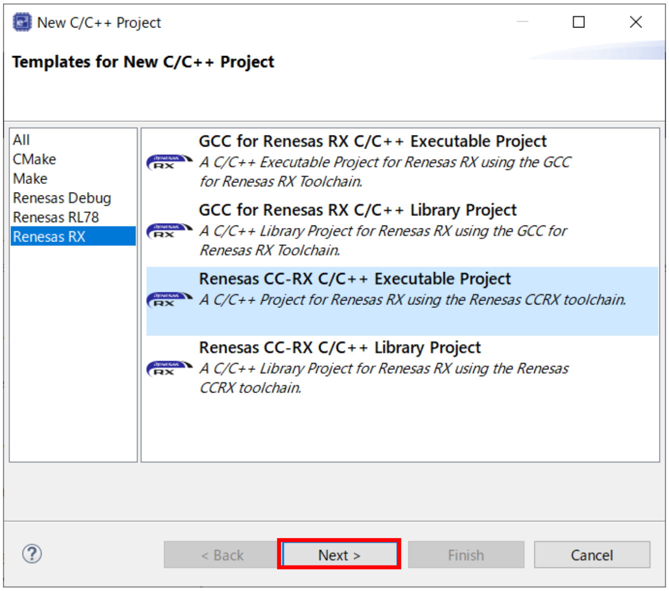
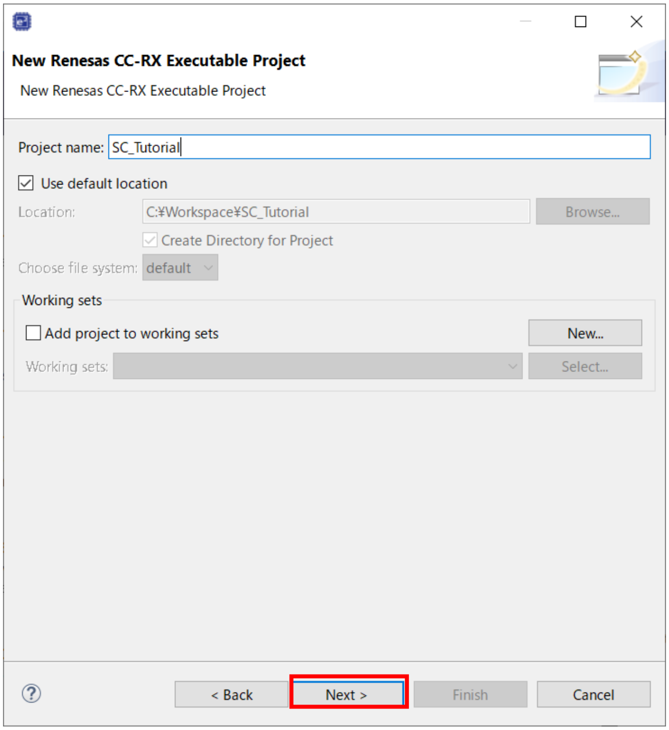
3. Project Creation with e^2 studio

3.1 Introduction

In this section, the user will be guided through the steps required to create a new C project for the RX140 MCU, ready to generate peripheral driver code using Smart Configurator. This project generation step is necessary to create the MCU-specific source, project and debug files.

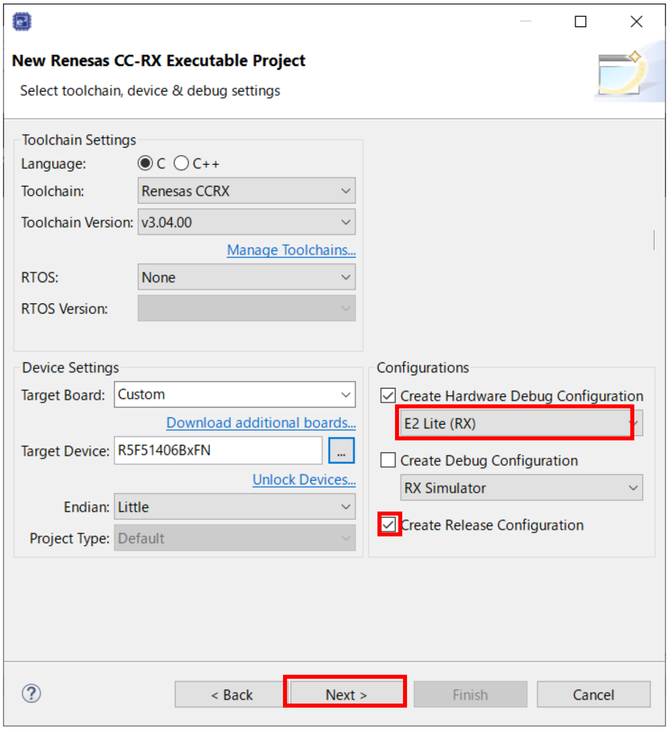
3.2 Creating the Project

• Start e^2 studio and select a suitable location for the project workspace.	 e² studio Launcher Recent Workspaces Launch Cancel Browse... Select a directory as workspace e² studio uses the workspace directory to store its preferences and development artifacts. Workspace: C:¥Workspace □ Use this as the default and do not ask again
• In the Welcome page, click 'Create a new C/C++ project'. (The Welcome page can also be opened from 'Help'-> 'Welcome'.)	 Workspace - e² studio File Edit Source Refactor Navigate Search Project Renesas Views Run Window Help Welcome RENESAS Welcome to e² studio Hide Create a new C/C++ project Create a new e² studio C/C++ project Import existing projects Import existing e² studio projects from the filesystem or archive Import sample projects Download and import sample projects from Renesas website Review IDE configuration settings Review the IDE's most fiercely contested preferences Open an existing file Open a file from the filesystem Overview Get an overview of the features Tutorials Go through tutorials Samples Try out the samples What's New Find out what is new Quick Start Guides Quickly getting familiar with the tool □ Always show Welcome at start up

- In the 'Templates for New C/C++ Project' dialog, selecting 'Renesas RX' -> 'Renesas CC-RX C/C++ Executable Project'. - Click 'Next'.	 New C/C++ Project **Templates for New C/C++ Project** All CMake Make Renesas Debug Renesas RL78 Renesas RX **GCC for Renesas RX C/C++ Executable Project** *A C/C++ Executable Project for Renesas RX using the GCC for Renesas RX Toolchain.* **GCC for Renesas RX C/C++ Library Project** *A C/C++ Library Project for Renesas RX using the GCC for Renesas RX Toolchain.* **Renesas CC-RX C/C++ Executable Project** *A C/C++ Project for Renesas RX using the Renesas CCRX toolchain.* **Renesas CC-RX C/C++ Library Project** *A C/C++ Library Project for Renesas RX using the Renesas CCRX toolchain.*
- Enter the project name 'SC_Tutorial'. Click 'Next'.	 **New Renesas CC-RX Executable Project** New Renesas CC-RX Executable Project Project name: SC_Tutorial ☑ Use default location Location: C:¥Workspace¥SC_Tutorial ☑ Create Directory for Project Choose file system: default Working sets ☐ Add project to working sets Working sets: Browse... New... Select... < Back Next > Finish Cancel

- In the ‘Select toolchain, device & debug settings’ dialog, select the options as shown in the screenshot opposite.
- In ‘Toolchains’ choose ‘Renesas CCRX’.
- The R5F51406BxFN MCU is found under RX100 -> RX140 ->

  RX140 - 144 pin.
- Select 'E2 Lite (RX)' from the pulldown and check 'Create Release Configuration' check box.
- Click ‘Next’.



New Renesas CC-RX Executable Project

Select toolchain, device & debug settings

Toolchain Settings

Language: ● C ○ C++

Toolchain: Renesas CCRX

Toolchain Version: v3.04.00

Manage Toolchains...

RTOS: None

RTOS Version:

Device Settings

Target Board: Custom

Download additional boards...

Target Device: R5F51406BxFN

Unlock Devices...

Endian: Little

Project Type: Default

Configurations

☑ Create Hardware Debug Configuration

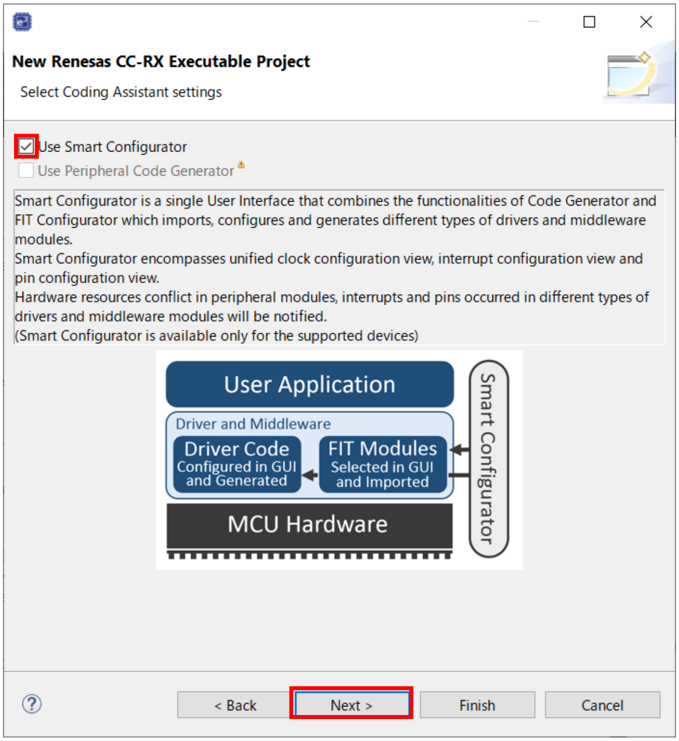
E2 Lite (RX)

□ Create Debug Configuration

RX Simulator

☑ Create Release Configuration

- In the ‘Select Coding Assistant settings’ dialog, select ‘Smart Configurator’.
- Click ‘Next’.



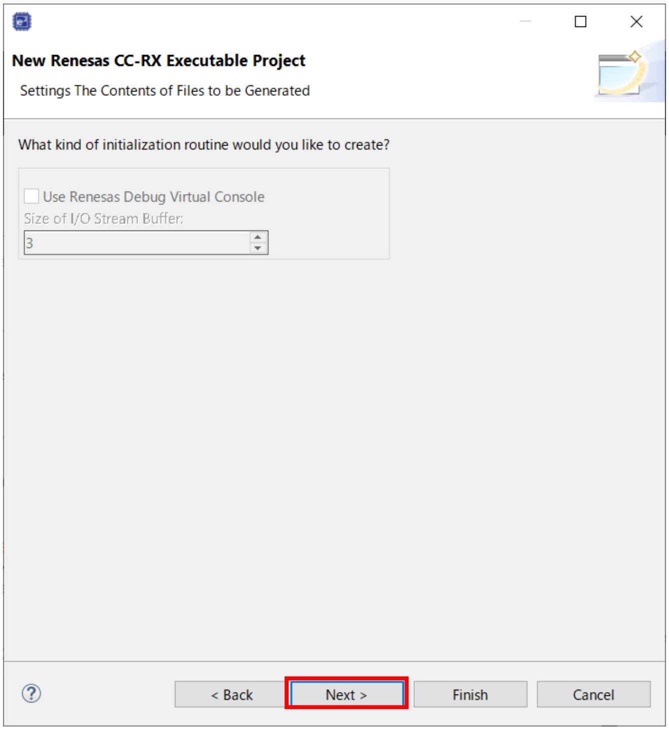
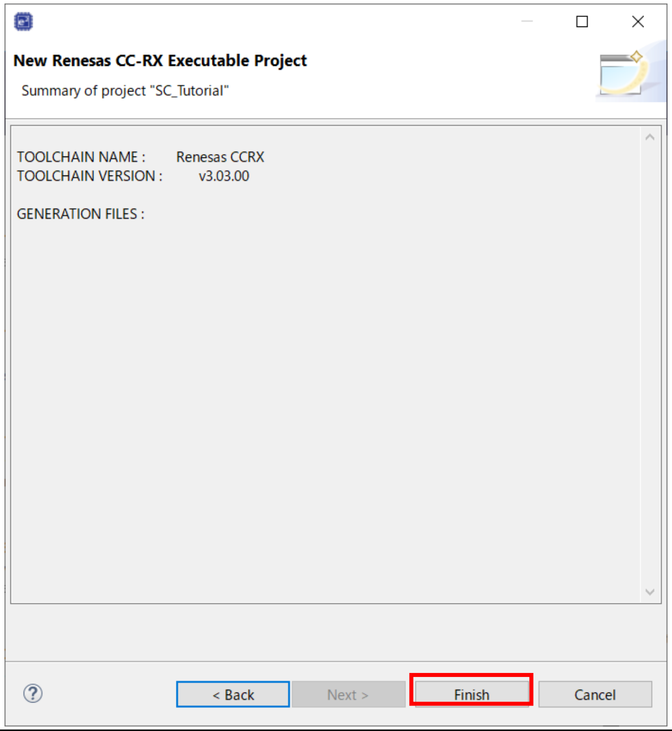
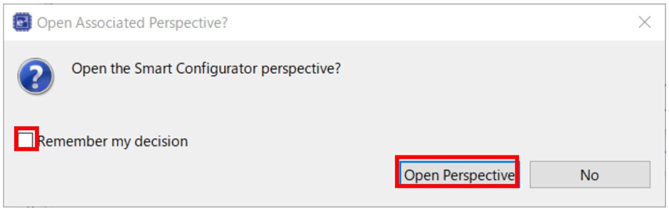
New Renesas CC-RX Executable Project

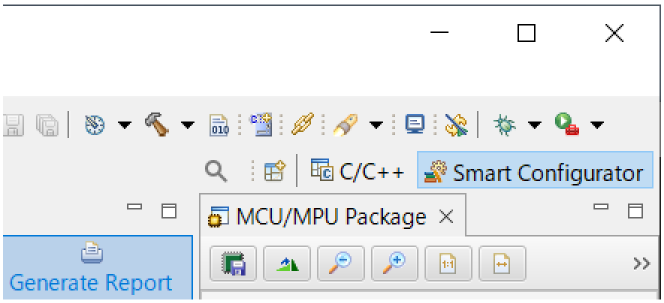
Select Coding Assistant settings

☑ Use Smart Configurator

□ Use Peripheral Code Generator

Smart Configurator is a single User Interface that combines the functionalities of Code Generator and FIT Configurator which imports, configures and generates different types of drivers and middleware modules.
Smart Configurator encompasses unified clock configuration view, interrupt configuration view and pin configuration view.
Hardware resources conflict in peripheral modules, interrupts and pins occurred in different types of drivers and middleware modules will be notified.
(Smart Configurator is available only for the supported devices)

• Click ‘Next’.	 **New Renesas CC-RX Executable Project** Settings The Contents of Files to be Generated What kind of initialization routine would you like to create? □ Use Renesas Debug Virtual Console Size of I/O Stream Buffer: 3
• A summary dialog will appear, click ‘Finish’ to complete the project generation.	 **New Renesas CC-RX Executable Project** Summary of project "SC_Tutorial" TOOLCHAIN NAME : Renesas CCRX TOOLCHAIN VERSION : v3.03.00 GENERATION FILES :
• In future, to skip the pop-up message on the right, check the 'Remember my decision' check box and click on 'Open Perspective'.	 Open Associated Perspective? Open the Smart Configurator perspective? □ Remember my decision

- The perspective changes automatically when the Smart Configurator starts up.	 C/C++ Smart Configurator MCU/MPU Package Generate Report

4. Smart Configurator Using the e^2 studio

4.1 Introduction

The Smart Configurator plug-in for the RX140 has been used to generate the sample code discussed in this document. Smart Configurator for e^2 studio is a plug-in tool for generating template 'C' source code and project settings for the RX140. When using Smart Configurator, it provides the user with a visual way of configuring the target device, clocks, software components, hardware resources and interrupts for the project; thereby bypassing the need, in most cases, to refer to sections of the Hardware Manual.

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 'Config_xxx.h', 'Config_xxx.c', and 'Config_xxx_user.c', where 'xxx' is an acronym for the relevant MCU feature, for example 'S12AD'. Within these code modules, the user is then free to add custom code to meet their specific requirement. However, these files require custom code to 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 */
```

Smart Configurator will locate these comment delimiters, and preserve any custom code inside the delimiters on subsequent code generation operations. This is useful if, after adding custom code, the user needs to re-visit Smart Configurator to change any MCU operating parameters.
Note: If code is added outside the above user code area, it will be lost if code generation is executed again with Smart Configurator.

By following the steps detailed in this Tutorial, the user will generate an e^2 studio project called SC_Tutorial. The fully completed Tutorial project is contained in the RSK Web Installer (https://www.renesas.com/rskrx140/install/e2) 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 Smart Configurator to generate their own custom projects for e^2 studio.

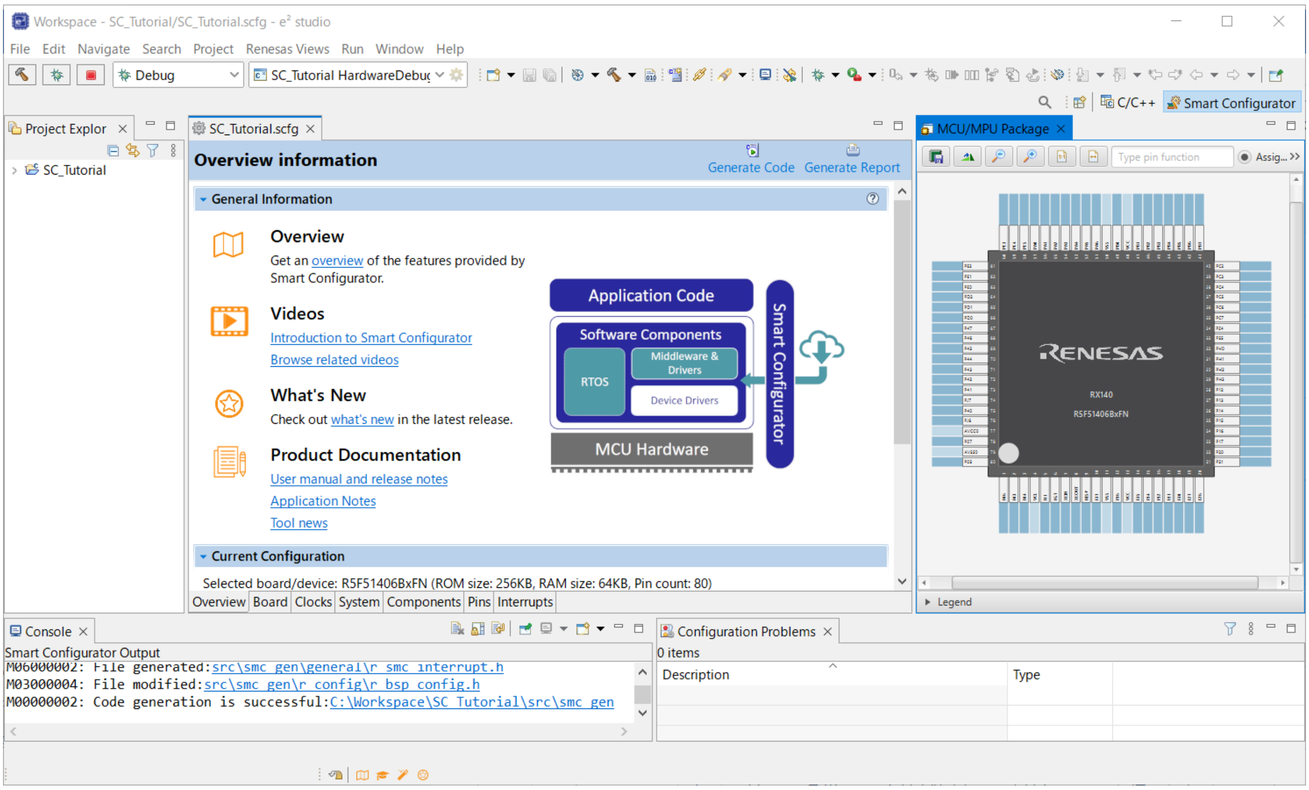
The SC_Tutorial project uses interrupts for switch inputs, the ADC module, the Compare Match Timer (CMT), the Serial Communications Interface (SCI) and uses these modules to perform A/D conversion. Results are displayed via the virtual COM port in a terminal program and also on the PMOD display connected to the RSK.

Following a tour of the key user interface features of Smart Configurator in the tabbed pages (board, clocks, components and pins), as well as a demonstration of building a project, the reader is guided through each of the peripheral function configuration pages and familiarised with the structure of the template code, including the process of adding their own code to the user code areas provided by the Smart Configurator

4.2 Project Configuration using Smart Configurator

In this section, a brief tour of Smart Configurator is presented. For further details of the Smart Configurator paradigm and reference, refer to the RX Smart Configurator User's Guide: e^2 studio.
You can download the latest document from: https://www.renesas.com/smart-configurator.

The Smart Configurator initial view is displayed as illustrated in **Figure 4-1**.



Selected board/device: R5F51406BxFN (ROM size: 256KB, RAM size: 64KB, Pin count: 80)

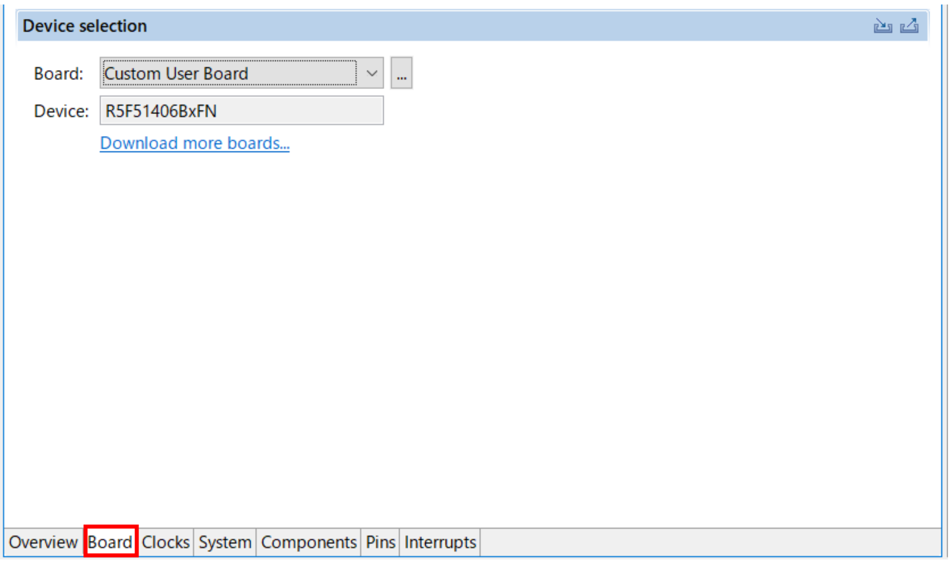
```
M06000002: File generated:src\smc_gen\general\r_smc_interrupt.h
M03000004: File modified:src\smc_gen\r_config\r_bsp_config.h
M00000002: Code generation is successful:C:\Workspace\SC_Tutorial\src\smc_gen
```

Figure 4-1 Overview page

Smart Configurator provides GUI features for configuration of MCU sub systems. Once the user has configured all required MCU sub systems and peripherals, the user can click the ‘Generate Code’ button, resulting in a fully configured e^2 studio project that builds and runs without error.

4.3 The ‘Board’ tabbed page

On the ‘Board’ tabbed page, set the board type and device type.
Click the 'Board' tab and it will be displayed as shown in **Figure 4-2**.



Device selection

Board: Custom User Board

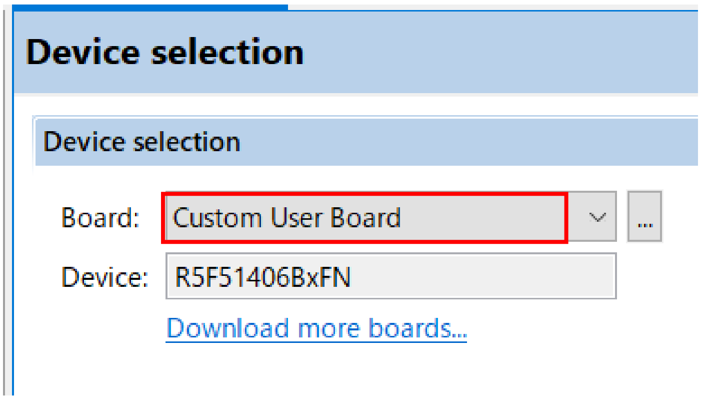
Device: R5F51406BxFN

Download more boards...

Figure 4-2 Board configuration page

4.3.1 Board configuration page

Make sure that ‘Custom User Board’ is selected for the ‘board:’.



Device selection

Device selection

Board: Custom User Board

Device: R5F51406BxFN

Download more boards...

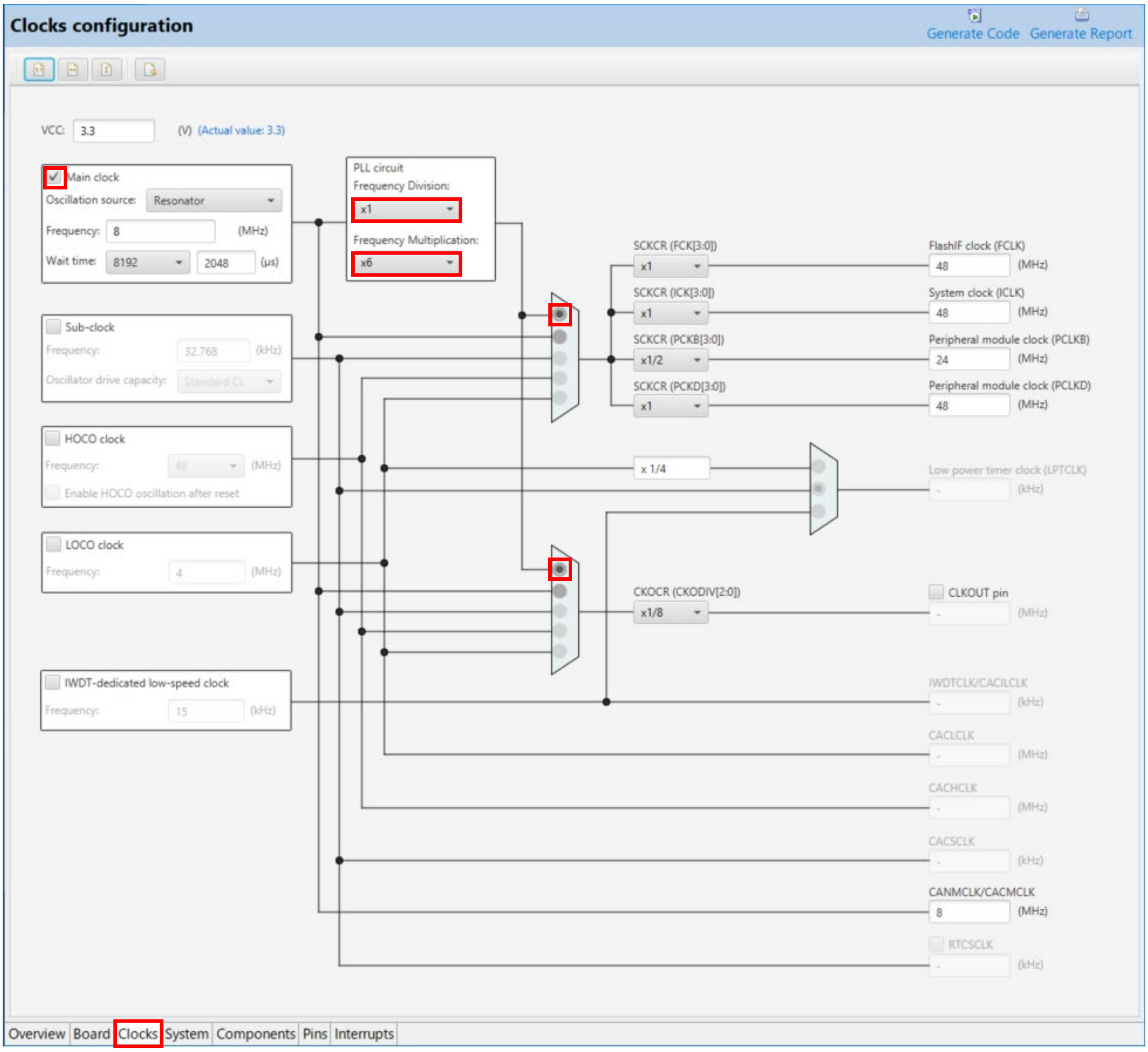
Figure 4-3 Select board

4.4 The ‘Clocks’ tabbed page

The ‘Clocks’ tabbed page configures clocks of the device selected. Clock source, frequency, PLL settings and clock divider settings can be configured for the output clocks. Clock configurations will be reflected in the r_bsp_config.h file in \src\smc_gen\r_config.

4.4.1 Clocks configuration

Figure 4-4 shows a screenshot of Smart Configurator with the Clocks configurations. Click on the ‘Clocks’ tab. Configure the system clocks as shown in the figure. In this tutorial, we are using the on board 8 MHz crystal resonator for our main clock oscillation source and the PLL circuit is in operation. The PLL output is used as the main system clock and the divisors should be set as shown in **Figure 4-4**.



Clocks configuration

VCC: 3.3 (V) (Actual value: 3.3)

☑ Main clock
Oscillation source: Resonator
Frequency: 8 (MHz)
Wait time: 8192 2048 (µs)

□ Sub-clock
Frequency: 32.768 (kHz)
Oscillator drive capacity: Standard CL

□ HOCO clock
Frequency: 48 (MHz)
□ Enable HOCO oscillation after reset

□ LOCO clock
Frequency: 4 (MHz)

□ IWDT-dedicated low-speed clock
Frequency: 15 (kHz)

PLL circuit
Frequency Division: x1
Frequency Multiplication: x6

SCKCR (FCK[3:0]): x1 — FlashIF clock (FCLK): 48 (MHz)
SCKCR (ICK[3:0]): x1 — System clock (ICLK): 48 (MHz)
SCKCR (PCKB[3:0]): x1/2 — Peripheral module clock (PCLKB): 24 (MHz)
SCKCR (PCKD[3:0]): x1 — Peripheral module clock (PCLKD): 48 (MHz)

x 1/4 — Low power timer clock (LPTCLK): - (kHz)

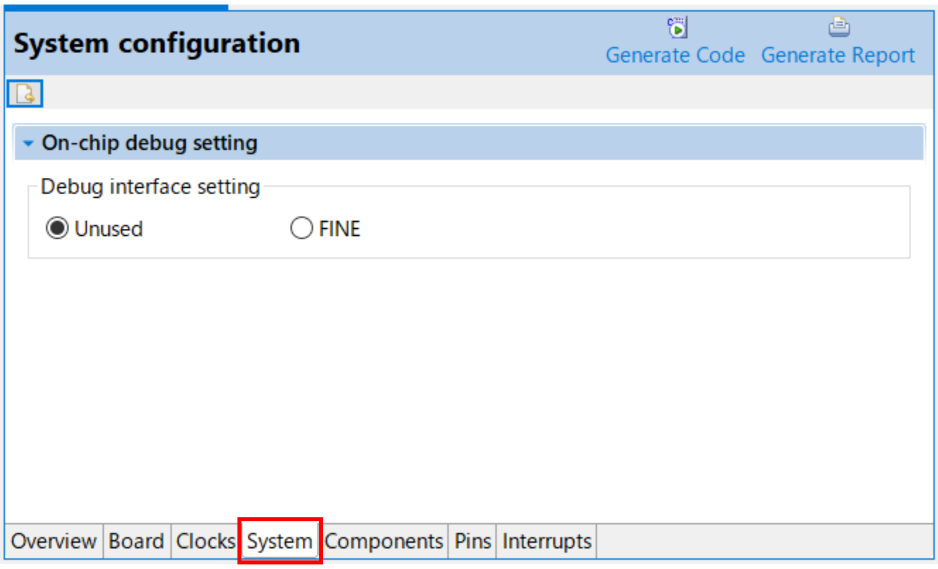
CKOCR (CKODIV[2:0]): x1/8 — □ CLKOUT pin: - (MHz)

IWDTCLK/CACILCLK: - (kHz)
CACLCLK: - (MHz)
CACHCLK: - (MHz)
CACSCLK: - (kHz)
CANMCLK/CACMCLK: 8 (MHz)
□ RTCSCLK: - (kHz)

Figure 4-4 The ‘Clocks’ tabbed page

4.5 The ‘System’ tabbed page

Set the On-chip debug setting mode on the ‘System’ tabbed page.



On-chip debug setting

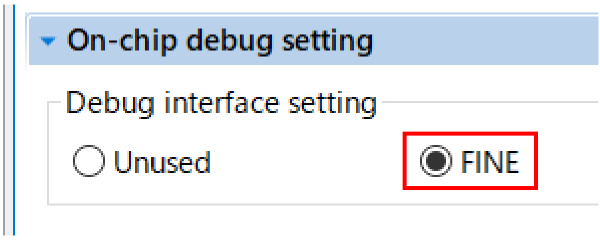
Debug interface setting

● Unused ○ FINE

Figure 4-5 The ‘System’ tabbed page

4.5.1 On-chip debug

The On-chip debug settings set the interface used for debugging. For the RSKRX140 CPU board, select JTAG as shown in **Figure 4-6**.



On-chip debug setting

Debug interface setting

○ Unused ● FINE

Figure 4-6 Debug interface setting

4.6 The ‘Components’ tabbed page

Drivers and middleware are handled as software components in Smart Configurator. The ‘Components’ page allows the user to select and configure software components.

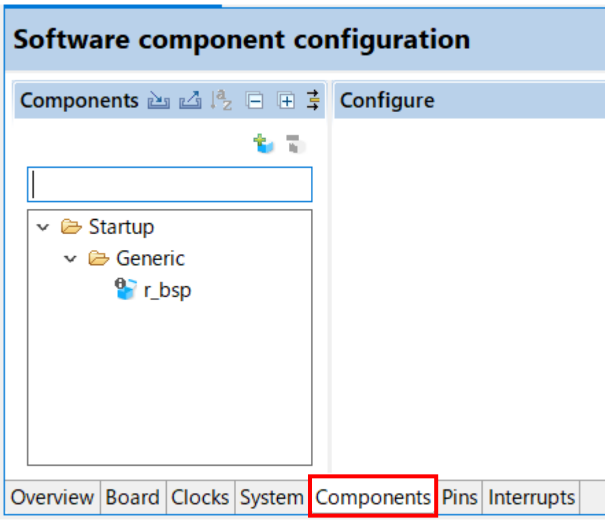


Figure 4-7 Components page

4.6.1 Add a software component into the project

Smart Configurator supports five types of software components: Startup, Drivers, Middleware, Application and RTOS. In the following sub-sections, the reader is guided through the steps to configure the MCU for a simple project containing interrupts for switch inputs, timers, ADC and a SCI by component of Drivers.

Click the ‘Add component’ icon.

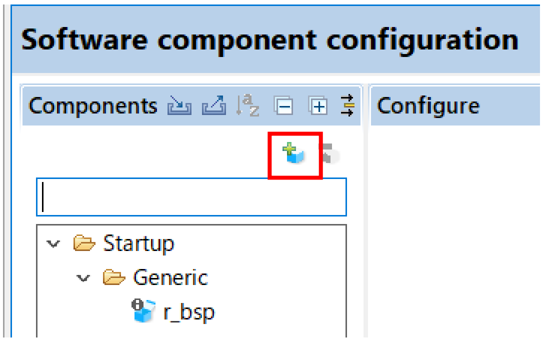
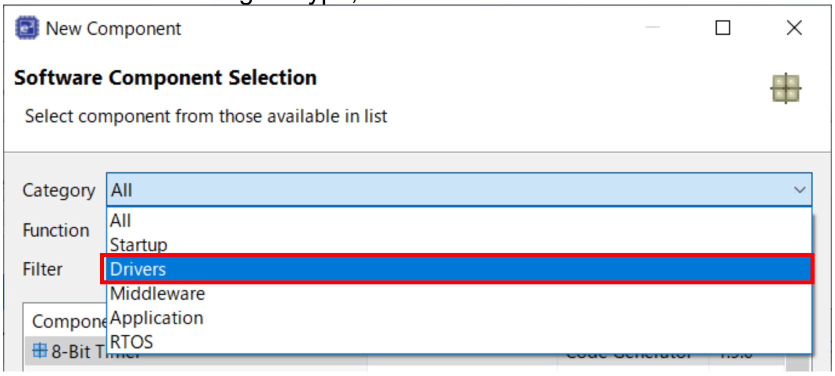


Figure 4-8 Add a Software component (1)

In ‘Software Component Selection’ dialog -> Type, select ‘Drivers’.



New Component

Software Component Selection

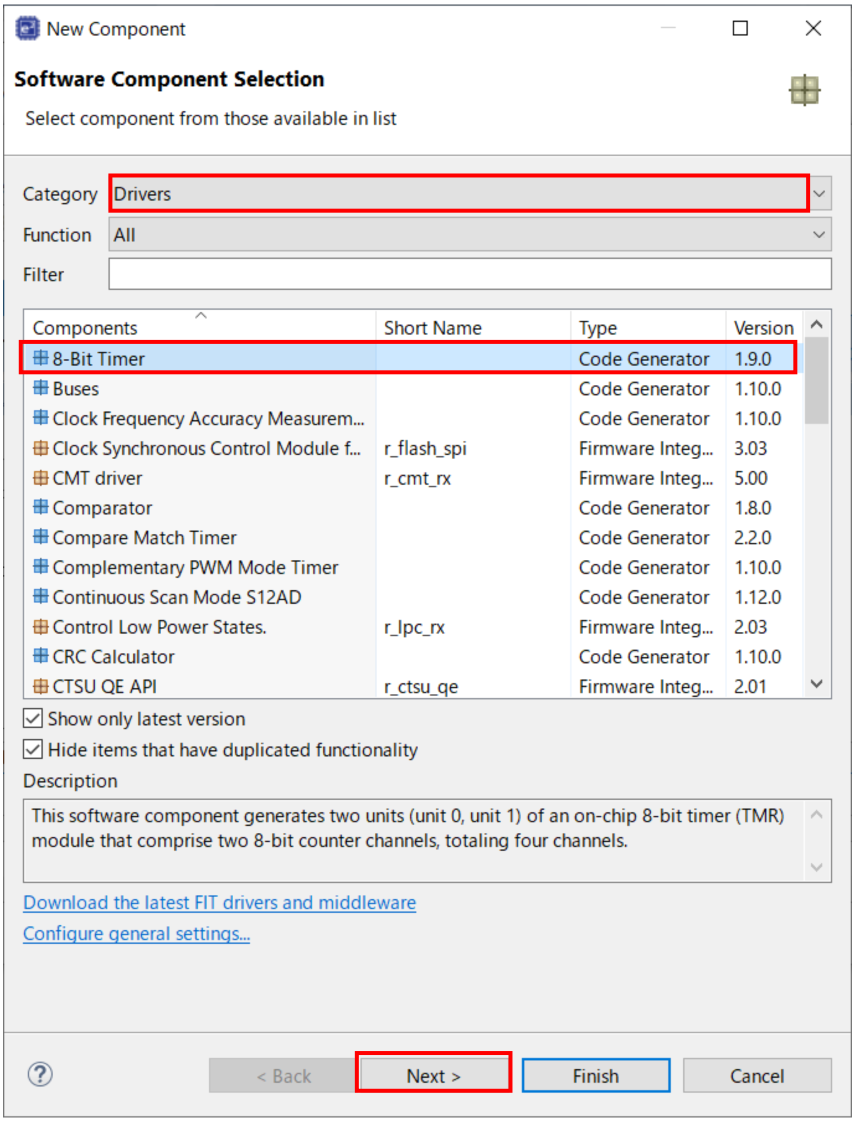
Select component from those available in list

Category All

Figure 4-9 Add a Software component (2)

4.6.2 8-Bit Timer

TMR0 will be used as an interval timer for generation of accurate delays. Select ‘8-Bit Timer’ as shown in **Figure 4-10** below then click ‘Next’.



New Component

Software Component Selection

Select component from those available in list

Category: Drivers

Function: All

Filter:

Components	Short Name	Type	Version
8-Bit Timer		Code Generator	1.9.0
Buses		Code Generator	1.10.0
Clock Frequency Accuracy Measurem...		Code Generator	1.10.0
Clock Synchronous Control Module f...	r_flash_spi	Firmware Integ...	3.03
CMT driver	r_cmt_rx	Firmware Integ...	5.00
Comparator		Code Generator	1.8.0
Compare Match Timer		Code Generator	2.2.0
Complementary PWM Mode Timer		Code Generator	1.10.0
Continuous Scan Mode S12AD		Code Generator	1.12.0
Control Low Power States.	r_lpc_rx	Firmware Integ...	2.03
CRC Calculator		Code Generator	1.10.0
CTSU QE API	r_ctsu_qe	Firmware Integ...	2.01

☑ Show only latest version

☑ Hide items that have duplicated functionality

Description

This software component generates two units (unit 0, unit 1) of an on-chip 8-bit timer (TMR) module that comprise two 8-bit counter channels, totaling four channels.

Download the latest FIT drivers and middleware

Configure general settings...

Figure 4-10 Select 8-Bit Timer

In ‘Add new configuration for selected component’ dialog -> Resource, select ‘TMR0’ as shown in **Figure 4-11** below.

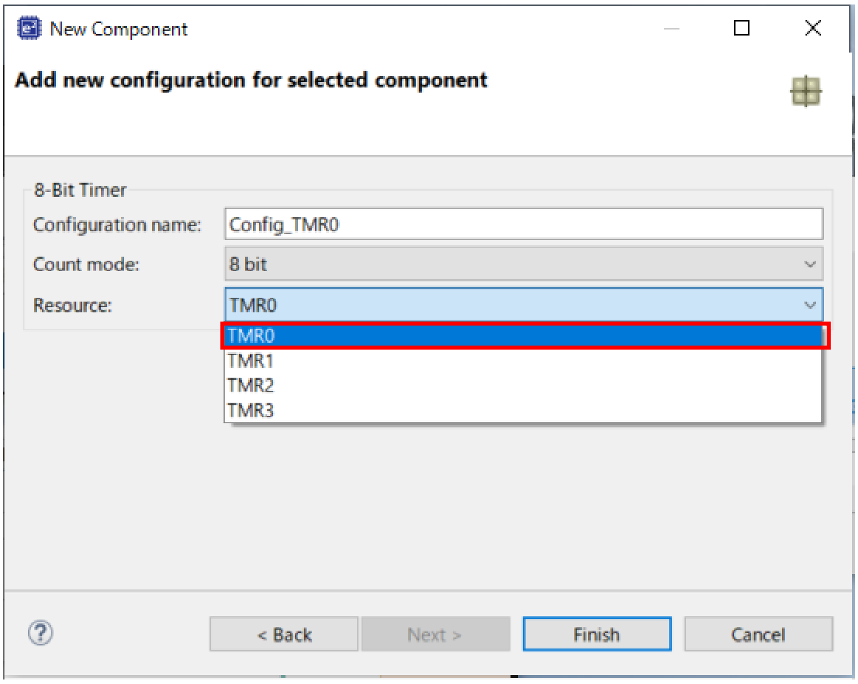


Figure 4-11 Select Resource - TMR0

Ensure that the ‘Configuration name’ updates to ‘Config_TMR0’ as shown in **Figure 4-12** below then click ‘Finish’.

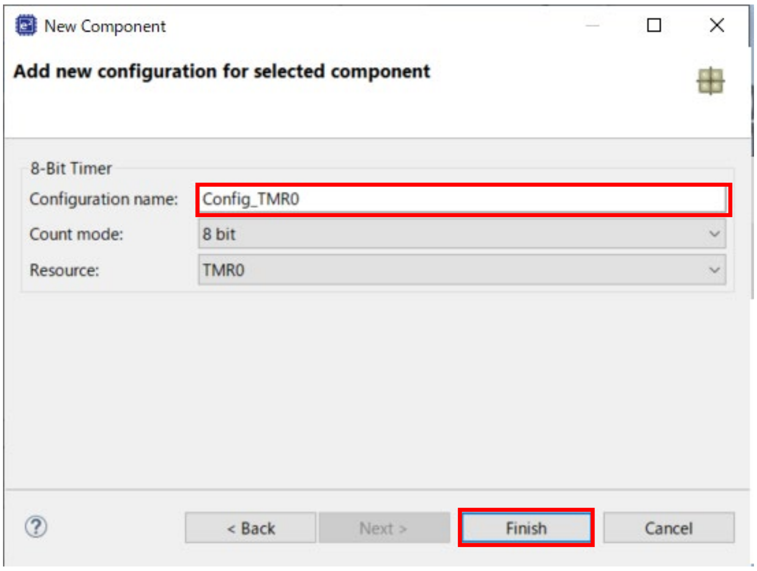
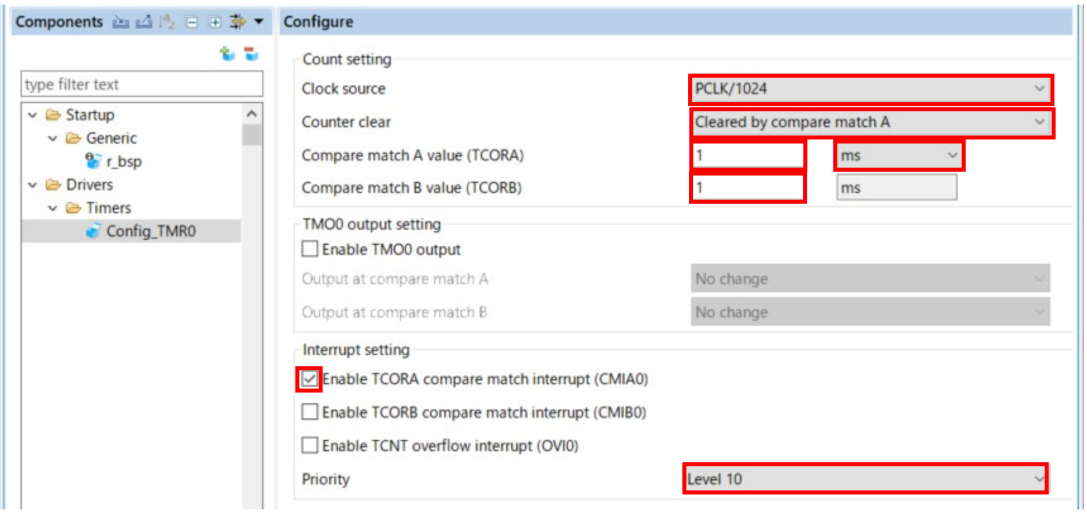


Figure 4-12 Ensure Configuration name - TMR0

In ‘Config_TMR0’, configure TMR0 as shown in **Figure 4-13**. This timer is configured to generate a high priority interrupt every 1ms. We will use this interrupt later in the tutorial to provide an API for generating high accuracy delays required in our application.



Count setting
Clock source: PCLK/1024
Counter clear: Cleared by compare match A
Compare match A value (TCORA): 1 ms
Compare match B value (TCORB): 1 ms

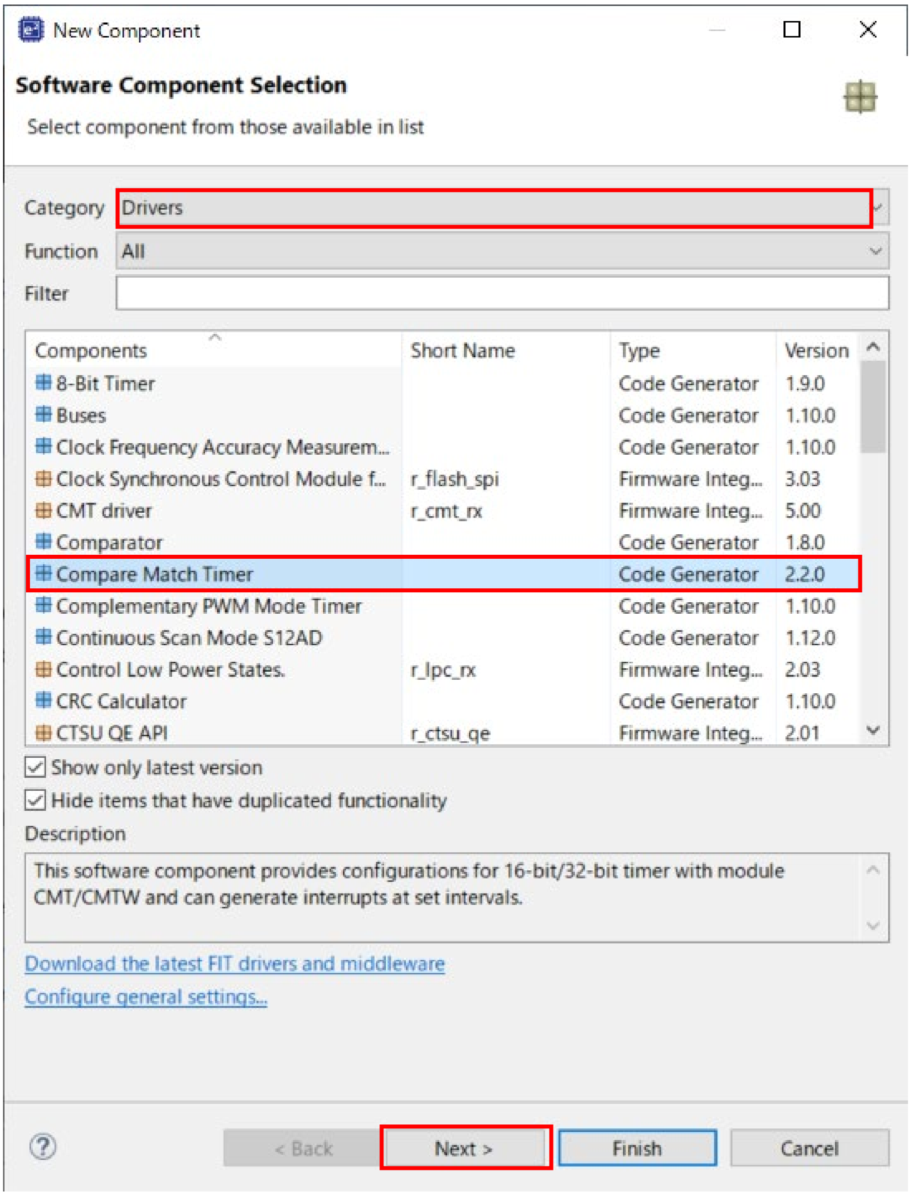
TMO0 output setting
□ Enable TMO0 output
Output at compare match A: No change
Output at compare match B: No change

Interrupt setting
☑ Enable TCORA compare match interrupt (CMIA0)
□ Enable TCORB compare match interrupt (CMIB0)
□ Enable TCNT overflow interrupt (OVI0)
Priority: Level 10

Figure 4-13 Config_TMR0 setting

4.6.3 Compare Match Timer

CMT0 will be used as an interval timer for generation of accurate delays. CMT0 and CMT1 will be used as timers in de-bouncing of switch interrupts.
Select ‘Compare Match Timer’ as shown in **Figure 4-14** below then click ‘Next’.



New Component

Software Component Selection

Select component from those available in list

Category: Drivers

Function: All

Filter:

Components	Short Name	Type	Version
8-Bit Timer		Code Generator	1.9.0
Buses		Code Generator	1.10.0
Clock Frequency Accuracy Measurem...		Code Generator	1.10.0
Clock Synchronous Control Module f...	r_flash_spi	Firmware Integ...	3.03
CMT driver	r_cmt_rx	Firmware Integ...	5.00
Comparator		Code Generator	1.8.0
Compare Match Timer		Code Generator	2.2.0
Complementary PWM Mode Timer		Code Generator	1.10.0
Continuous Scan Mode S12AD		Code Generator	1.12.0
Control Low Power States.	r_lpc_rx	Firmware Integ...	2.03
CRC Calculator		Code Generator	1.10.0
CTSU QE API	r_ctsu_qe	Firmware Integ...	2.01

☑ Show only latest version

☑ Hide items that have duplicated functionality

Description

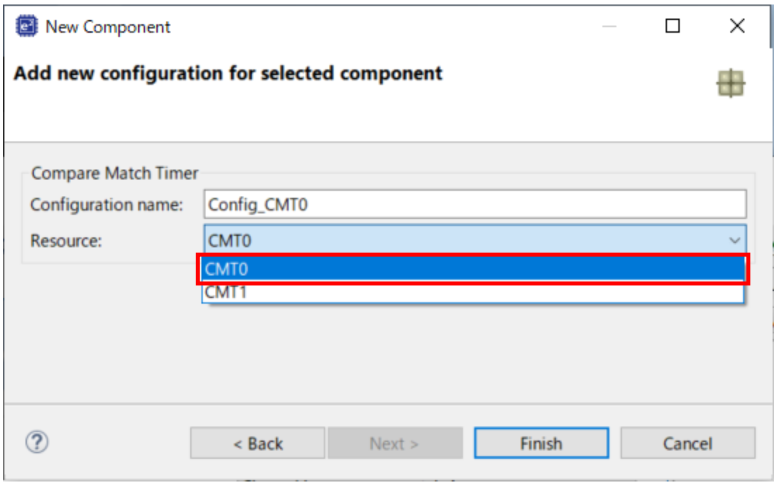
This software component provides configurations for 16-bit/32-bit timer with module CMT/CMTW and can generate interrupts at set intervals.

Download the latest FIT drivers and middleware

Configure general settings...

Figure 4-14 Select Compare Match Timer

In ‘Add new configuration for selected component’ dialog -> Resource, select ‘CMT0’ as shown in **Figure 4-15** below.


New Component

Add new configuration for selected component

Compare Match Timer

Configuration name: Config_CMT0

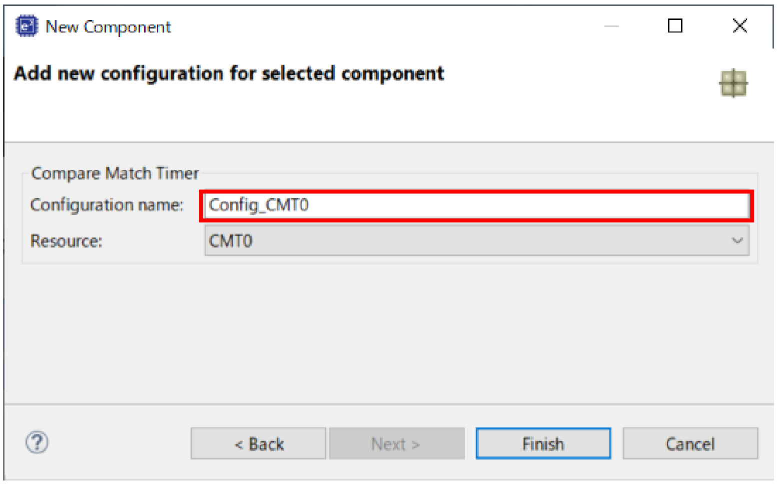
Resource: CMT0

CMT0

CMT1

Figure 4-15 Select Resource - CMT0

Ensure that the ‘Configuration name’ updates to ‘Config_CMT0’ as shown in **Figure 4-16** below then click ‘Finish’.


New Component

Add new configuration for selected component

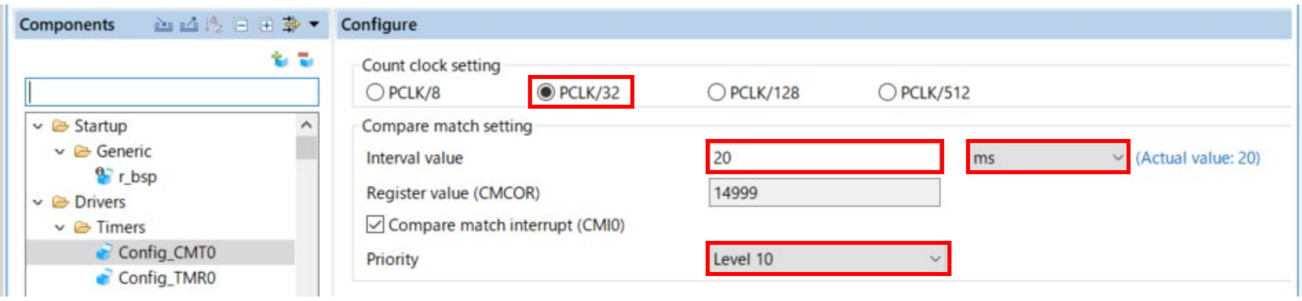
Compare Match Timer

Configuration name: Config_CMT0

Resource: CMT0

Figure 4-16 Ensure Configuration name - CMT0

In ‘Config_CMT0’, configure CMT0 as shown in **Figure 4-17**. This timer is configured to generate a high priority interrupt every 20ms. We will use this interrupt later in the tutorial to provide an API for generating high accuracy delays required in our application.



Count clock setting

○ PCLK/8 ● PCLK/32 ○ PCLK/128 ○ PCLK/512

Compare match setting

Interval value: 20 ms (Actual value: 20)

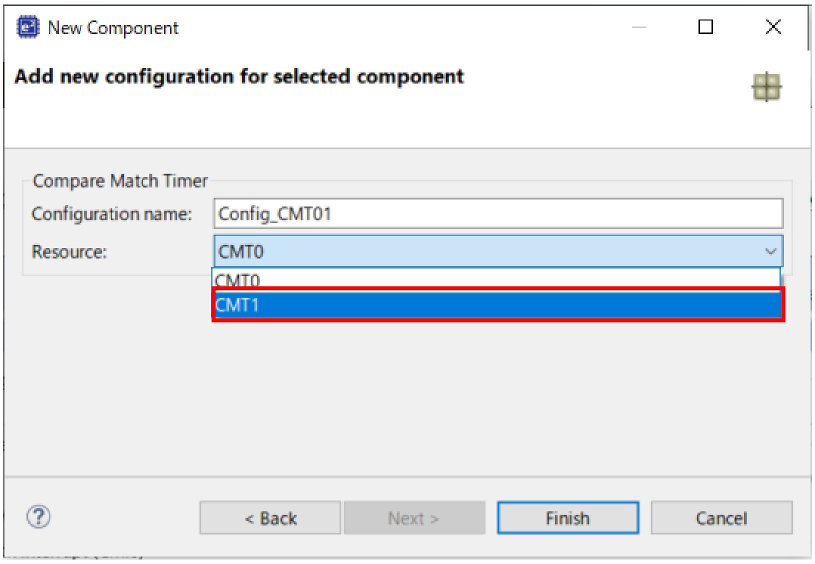
Register value (CMCOR): 14999

☑ Compare match interrupt (CMI0)

Priority: Level 10

Figure 4-17 Config_CMT0 setting

Click the ‘Add component’ icon. In ‘Software Component Selection’ dialog -> Type, select ‘Drivers’ . Select ‘Compare Match Timer’ then click ‘Next’. In ‘Add new configuration for selected component’ dialog -> Resource, select ‘CMT1’ as shown in **Figure 4-18** below.



New Component

Add new configuration for selected component

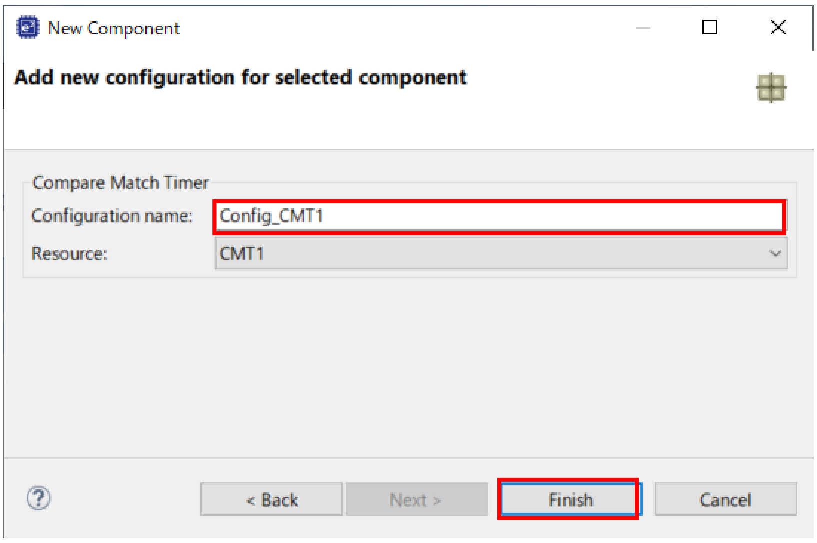
Compare Match Timer

Configuration name: Config_CMT01

Resource: CMT0

Figure 4-18 Select Resource – CMT1

Ensure that the ‘Configuration name’ updates to ‘Config_CMT1’ as shown in **Figure 4-19** below then click ‘Finish’.



New Component

Add new configuration for selected component

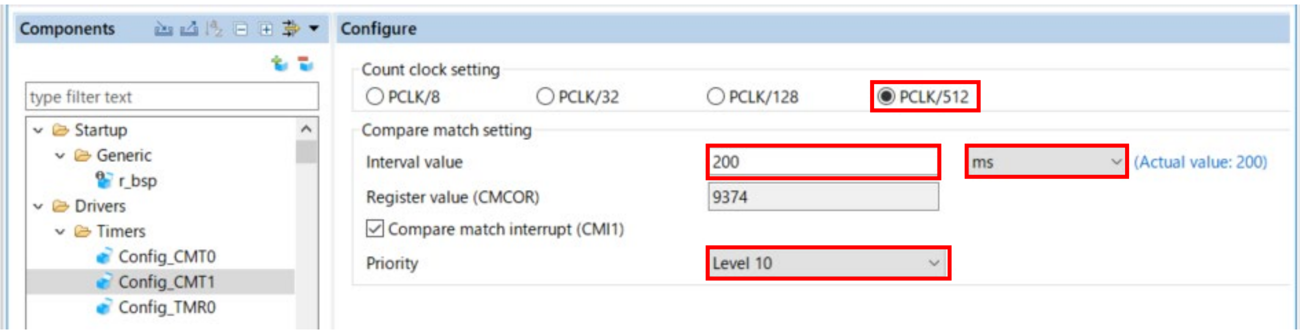
Compare Match Timer

Configuration name: Config_CMT1

Resource: CMT1

Figure 4-19 Ensure Configuration name – CMT1

Navigate to the 'Config_CMT1' and configure CMT1 as shown in **Figure 4-20**. This timer is configured to generate a high priority interrupt after 200ms. This timer is used as our short switch de-bounce timer later in this tutorial.



Count clock setting

○ PCLK/8 ○ PCLK/32 ○ PCLK/128 ● PCLK/512

Compare match setting

Interval value 200 ms (Actual value: 200)

Register value (CMCOR) 9374

☑ Compare match interrupt (CMI1)

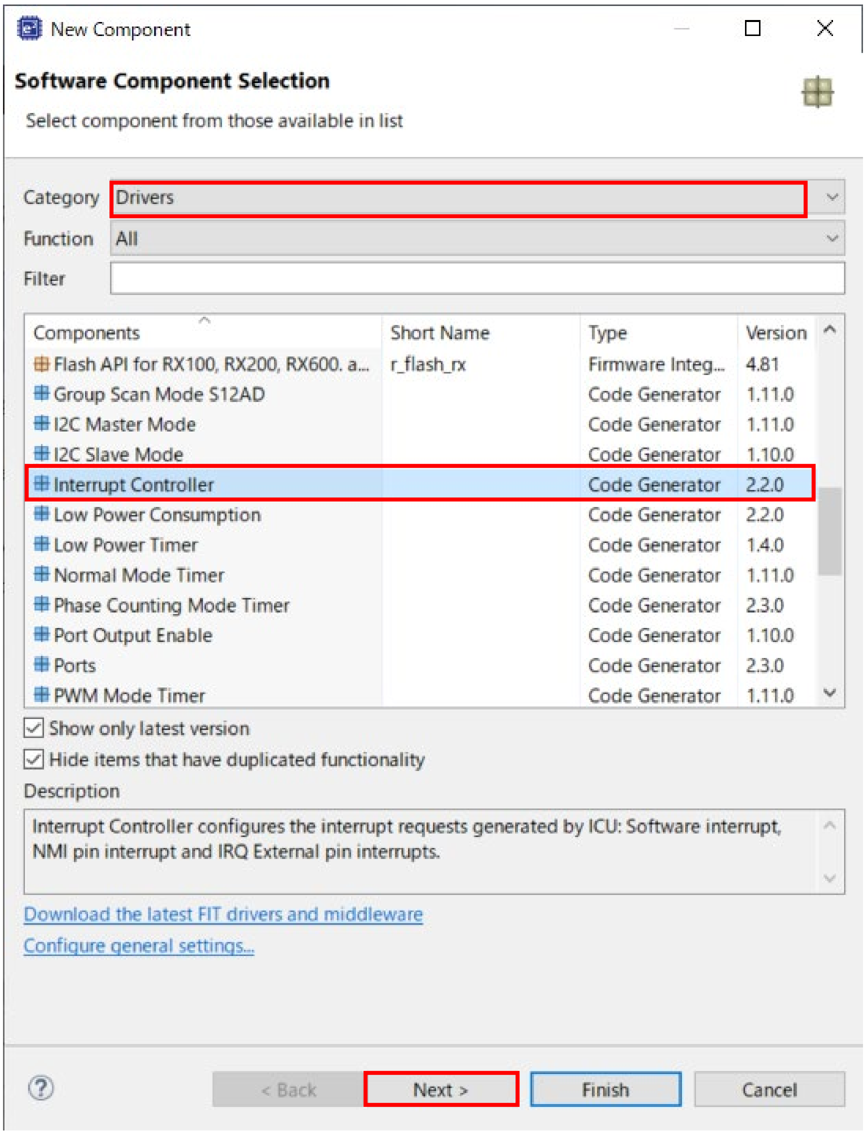
Priority Level 10

Figure 4-20 Config_CMT1 setting

4.6.4 Interrupt Controller

Referring to the RSK schematic, SW1 is connected to IRQ1(P31) and SW2 is connected to IRQ2(P32). SW3 is connected to IRQ6(P16) and ADTRG0n. This tutorial uses ADTRG0n, which will be configured later in section 4.6.8.

Click the 'Add component' icon. In 'Software Component Selection' dialog -> Type, select 'Drivers'. Select 'Interrupt Controller' as shown in **Figure 4-21** then click 'Next'.



New Component

Software Component Selection

Select component from those available in list

Category: Drivers

Function: All

Filter:

Components	Short Name	Type	Version
Flash API for RX100, RX200, RX600. a...	r_flash_rx	Firmware Integ...	4.81
Group Scan Mode S12AD		Code Generator	1.11.0
I2C Master Mode		Code Generator	1.11.0
I2C Slave Mode		Code Generator	1.10.0
Interrupt Controller		Code Generator	2.2.0
Low Power Consumption		Code Generator	2.2.0
Low Power Timer		Code Generator	1.4.0
Normal Mode Timer		Code Generator	1.11.0
Phase Counting Mode Timer		Code Generator	2.3.0
Port Output Enable		Code Generator	1.10.0
Ports		Code Generator	2.3.0
PWM Mode Timer		Code Generator	1.11.0

☑ Show only latest version

☑ Hide items that have duplicated functionality

Description

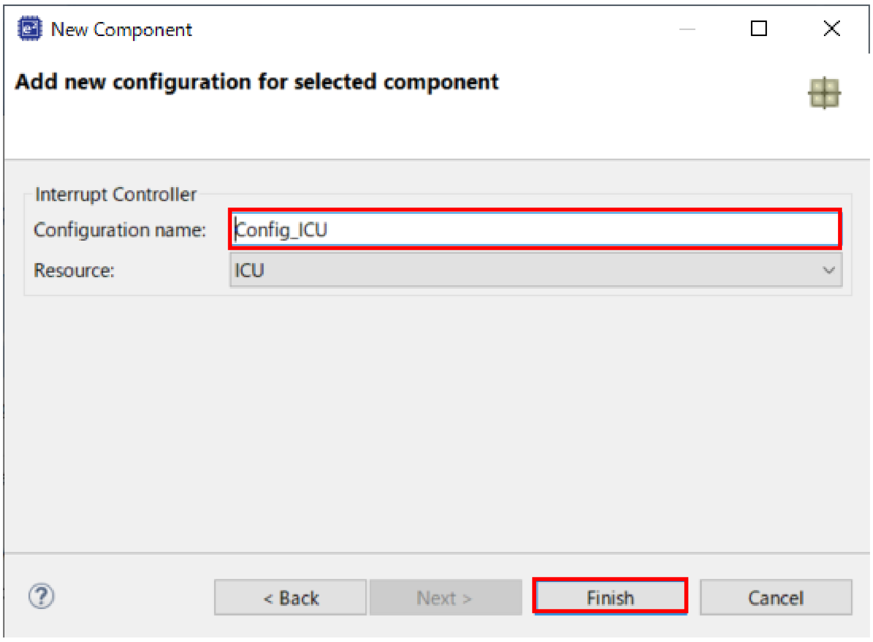
Interrupt Controller configures the interrupt requests generated by ICU: Software interrupt, NMI pin interrupt and IRQ External pin interrupts.

Download the latest FIT drivers and middleware

Configure general settings...

Figure 4-21 Select Interrupt Controller

In 'Add new configuration for selected component' dialog -> Resource, select 'ICU' as shown in **Figure 4-22** below then click 'Finish'.



New Component

Add new configuration for selected component

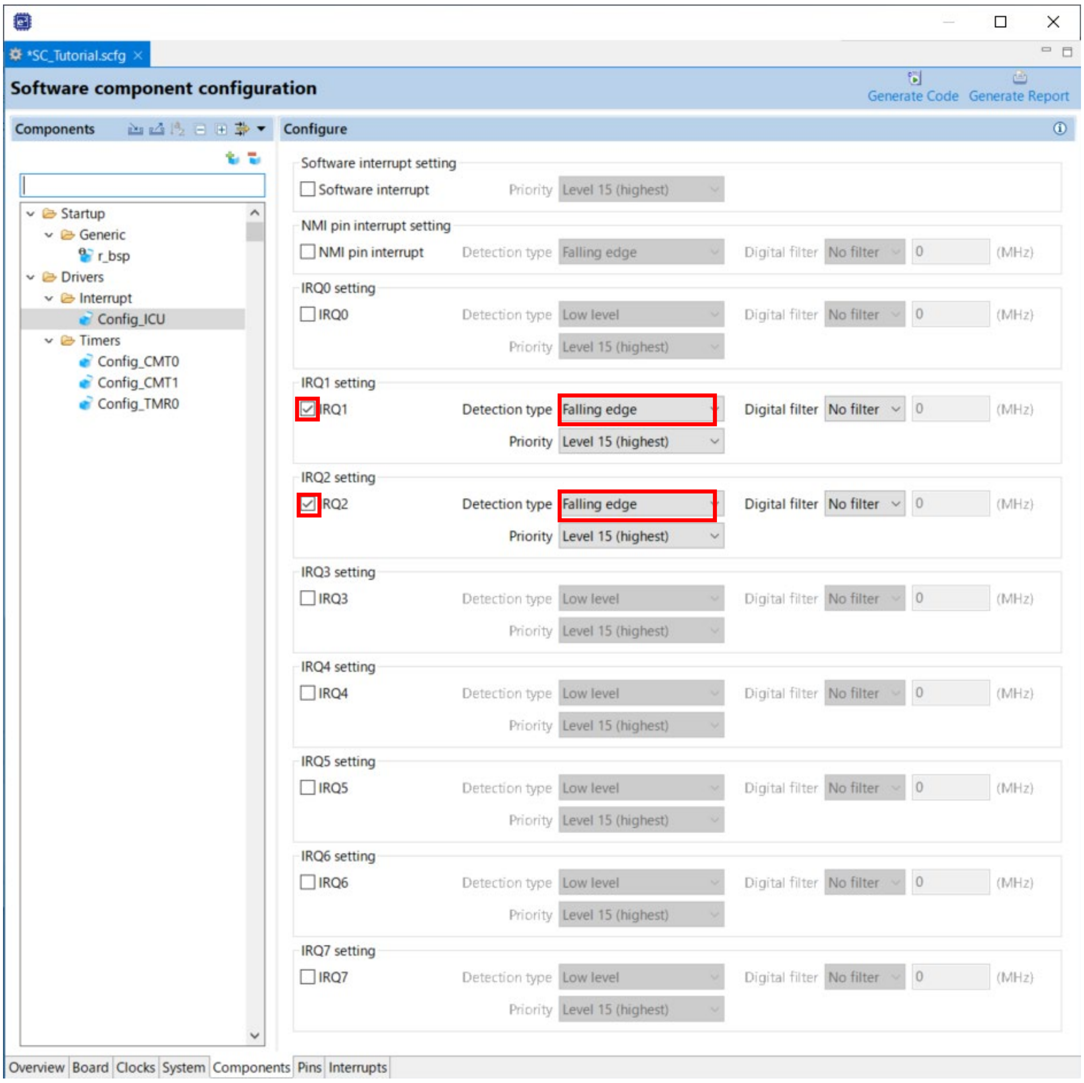
Interrupt Controller

Configuration name: Config_ICU

Resource: ICU

Figure 4-22 Select Resource – ICU

Navigate to the ‘Config_ICU’, configure these two interrupts as falling edge triggered as shown in **Figure 4-23** below.



Software interrupt setting
□ Software interrupt   Priority: Level 15 (highest)

NMI pin interrupt setting
□ NMI pin interrupt   Detection type: Falling edge   Digital filter: No filter   0 (MHz)

IRQ0 setting
□ IRQ0   Detection type: Low level   Digital filter: No filter   0 (MHz)
Priority: Level 15 (highest)

IRQ1 setting
☑ IRQ1   Detection type: Falling edge   Digital filter: No filter   0 (MHz)
Priority: Level 15 (highest)

IRQ2 setting
☑ IRQ2   Detection type: Falling edge   Digital filter: No filter   0 (MHz)
Priority: Level 15 (highest)

IRQ3 setting
□ IRQ3   Detection type: Low level   Digital filter: No filter   0 (MHz)
Priority: Level 15 (highest)

IRQ4 setting
□ IRQ4   Detection type: Low level   Digital filter: No filter   0 (MHz)
Priority: Level 15 (highest)

IRQ5 setting
□ IRQ5   Detection type: Low level   Digital filter: No filter   0 (MHz)
Priority: Level 15 (highest)

IRQ6 setting
□ IRQ6   Detection type: Low level   Digital filter: No filter   0 (MHz)
Priority: Level 15 (highest)

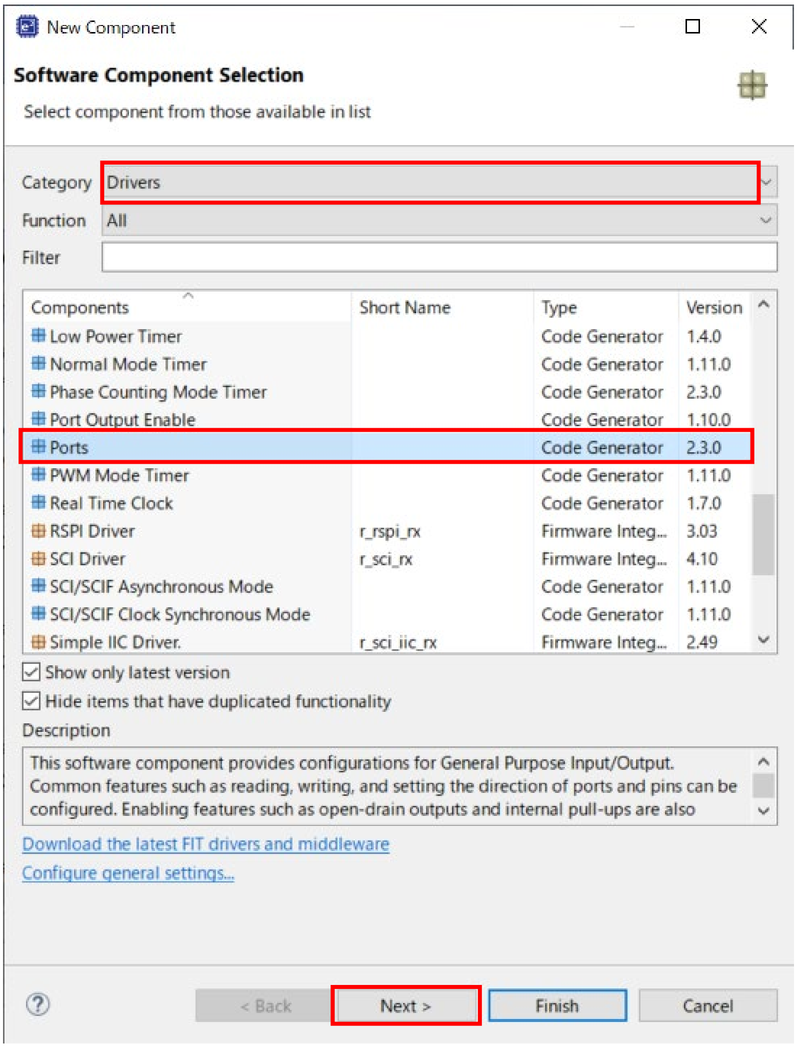
IRQ7 setting
□ IRQ7   Detection type: Low level   Digital filter: No filter   0 (MHz)
Priority: Level 15 (highest)

Figure 4-23 Config_ICU setting

4.6.5 Ports

Referring to the RSK schematic, LED0 is connected to P21, LED1 is connected to P04, LED2 is connected to P06 and LED3 is connected to P07. PB2 is used as one of the LCD control lines, together with PE4, PC7 and PC6.

Click the 'Add component' icon. In 'Software Component Selection' dialog -> Type, select 'Drivers'. Select 'Ports' as shown in **Figure 4-24** then click 'Next'.


New Component

Software Component Selection

Select component from those available in list

Category: Drivers

Function: All

Filter:

Components	Short Name	Type	Version
Low Power Timer		Code Generator	1.4.0
Normal Mode Timer		Code Generator	1.11.0
Phase Counting Mode Timer		Code Generator	2.3.0
Port Output Enable		Code Generator	1.10.0
Ports		Code Generator	2.3.0
PWM Mode Timer		Code Generator	1.11.0
Real Time Clock		Code Generator	1.7.0
RSPI Driver	r_rspi_rx	Firmware Integ...	3.03
SCI Driver	r_sci_rx	Firmware Integ...	4.10
SCI/SCIF Asynchronous Mode		Code Generator	1.11.0
SCI/SCIF Clock Synchronous Mode		Code Generator	1.11.0
Simple IIC Driver.	r_sci_iic_rx	Firmware Integ...	2.49

☑ Show only latest version

☑ Hide items that have duplicated functionality

Description

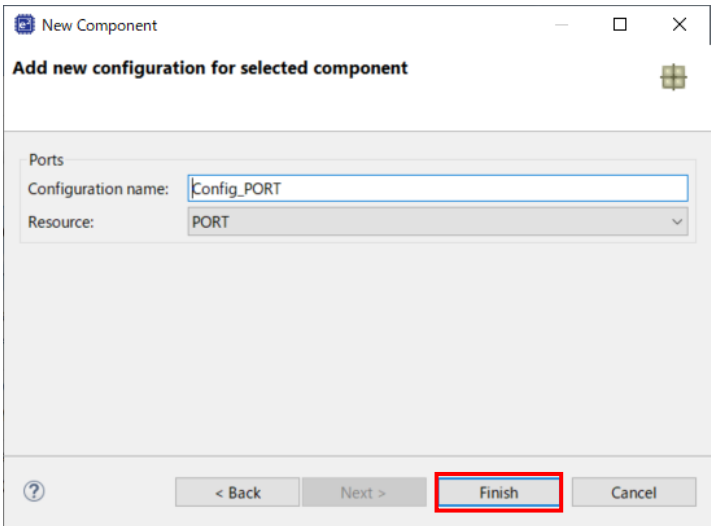
This software component provides configurations for General Purpose Input/Output. Common features such as reading, writing, and setting the direction of ports and pins can be configured. Enabling features such as open-drain outputs and internal pull-ups are also

Download the latest FIT drivers and middleware

Configure general settings...

Figure 4-24 Select Ports

In ‘Add new configuration for selected component’ dialog -> Resource, select ‘PORT’ as shown in **Figure 4-25** below then click ‘Finish’.


New Component

Add new configuration for selected component

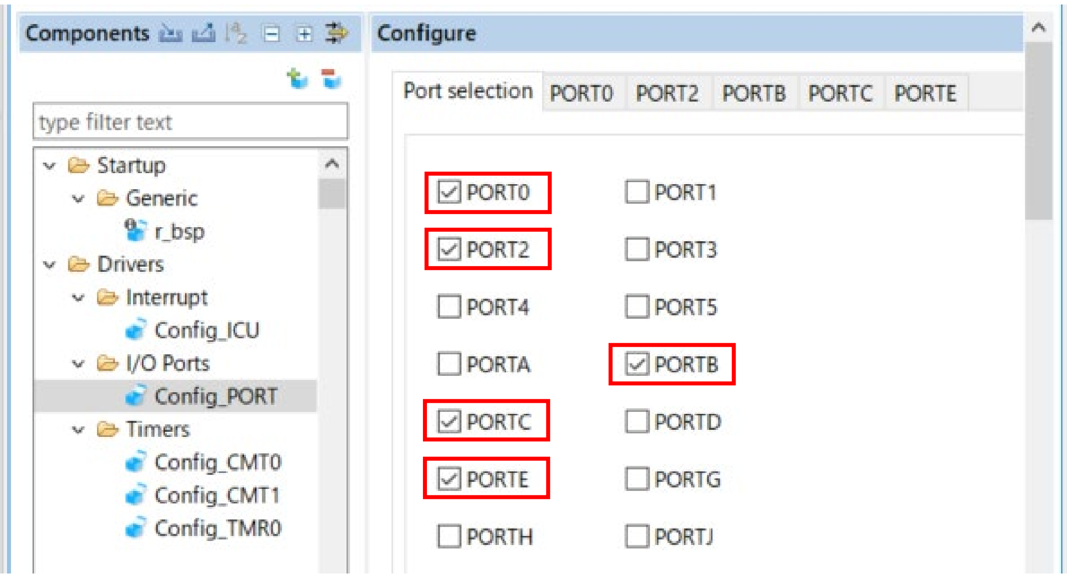
Ports

Configuration name: Config_PORT

Resource: PORT

Figure 4-25 Select Resource – PORT

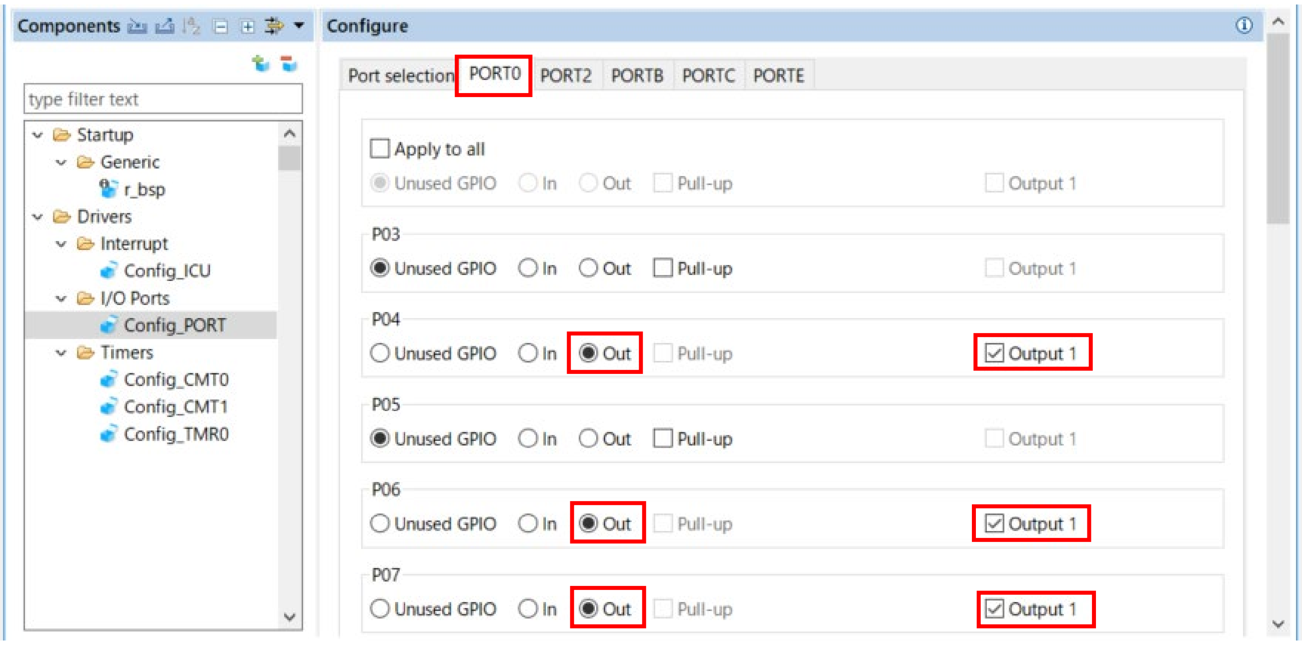
Tick the tickboxes for ‘PORT0’, ‘PORT2’, ‘PORTB’, ‘PORTC’ and ‘PORTE’ as shown in **Figure 4-26** below.



☑ PORT0 □ PORT1
☑ PORT2 □ PORT3
□ PORT4 □ PORT5
□ PORTA ☑ PORTB
☑ PORTC □ PORTD
☑ PORTE □ PORTG
□ PORTH □ PORTJ

Figure 4-26 Select Port selection

Navigate through each of the 'PORTx' tabs, configuring these four I/O lines and LCD control lines as shown in **Figure 4-27**, **Figure 4-28**, **Figure 4-29**, **Figure 4-30** and **Figure 4-31** below. Tick the tickboxes for ‘Out’ and tick ‘Output 1’ the tickboxes except for PC6 under the ‘PORTC’ tab. Start with the 'PORT0' tab.



□ Apply to all
● Unused GPIO ○ In ○ Out □ Pull-up □ Output 1

P03
● Unused GPIO ○ In ○ Out □ Pull-up □ Output 1

P04
○ Unused GPIO ○ In ● Out □ Pull-up ☑ Output 1

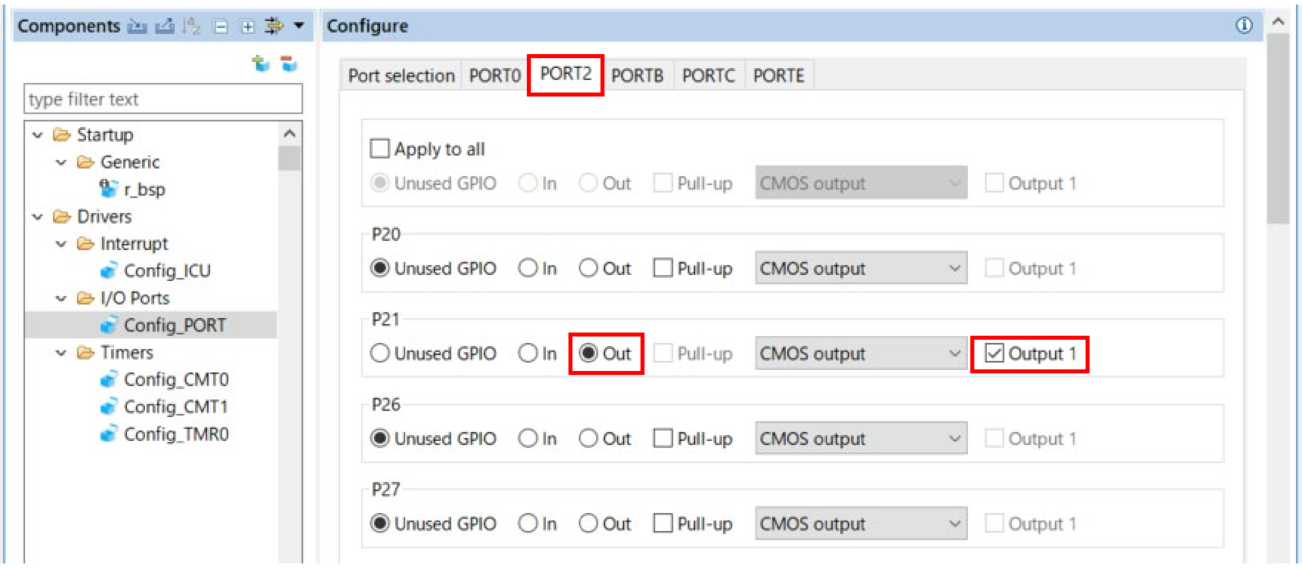
P05
● Unused GPIO ○ In ○ Out □ Pull-up □ Output 1

P06
○ Unused GPIO ○ In ● Out □ Pull-up ☑ Output 1

P07
○ Unused GPIO ○ In ● Out □ Pull-up ☑ Output 1

Figure 4-27 Select PORT0 tab

Select ‘PORT2’ tab.



□ Apply to all
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

P20
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

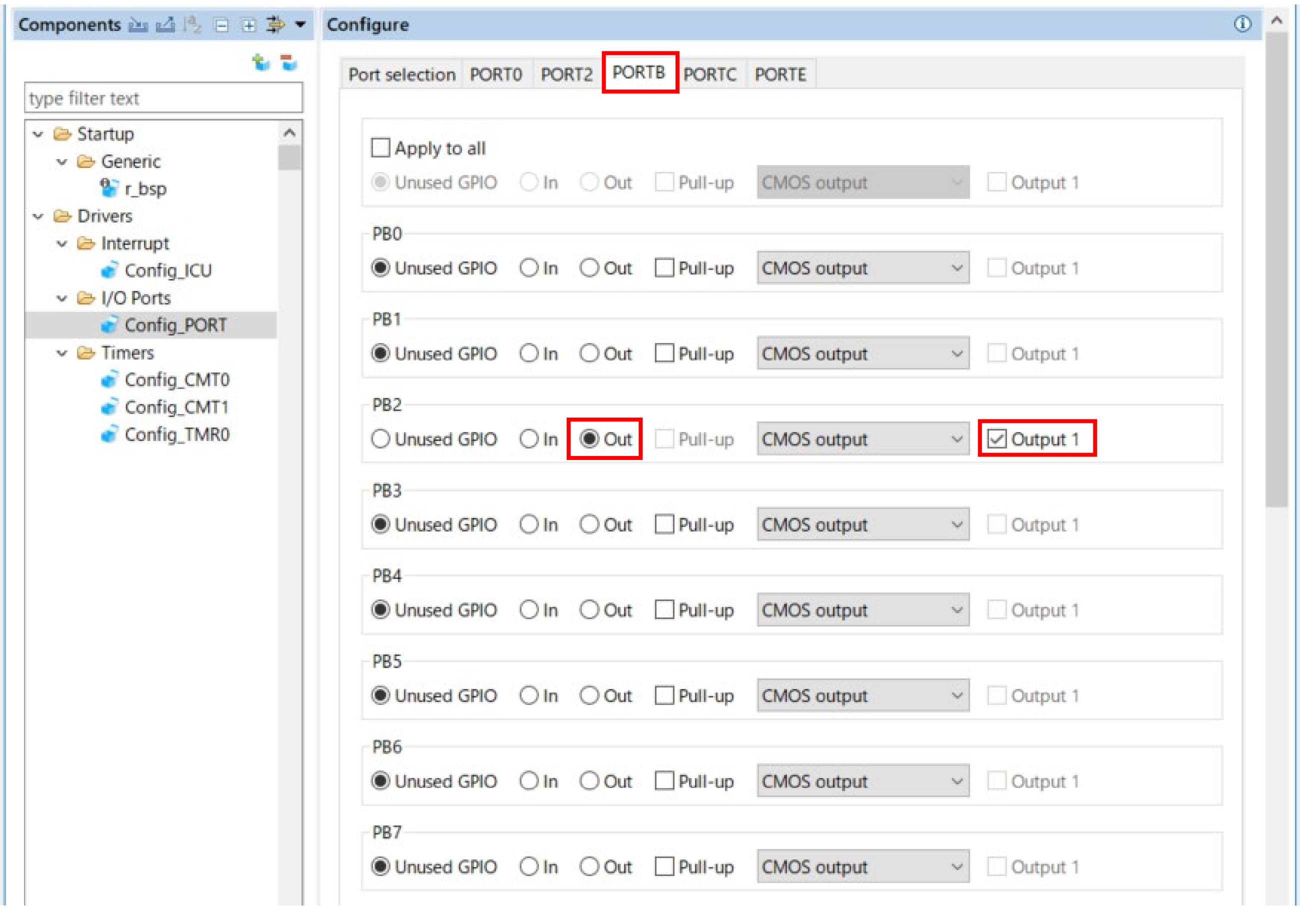
P21
○ Unused GPIO ○ In ● Out □ Pull-up CMOS output ☑ Output 1

P26
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

P27
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

Figure 4-28 Select PORT2 tab

Select ‘PORTB’ tab.



□ Apply to all
○ Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PB0
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PB1
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PB2
○ Unused GPIO ○ In ● Out □ Pull-up CMOS output ☑ Output 1

PB3
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PB4
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

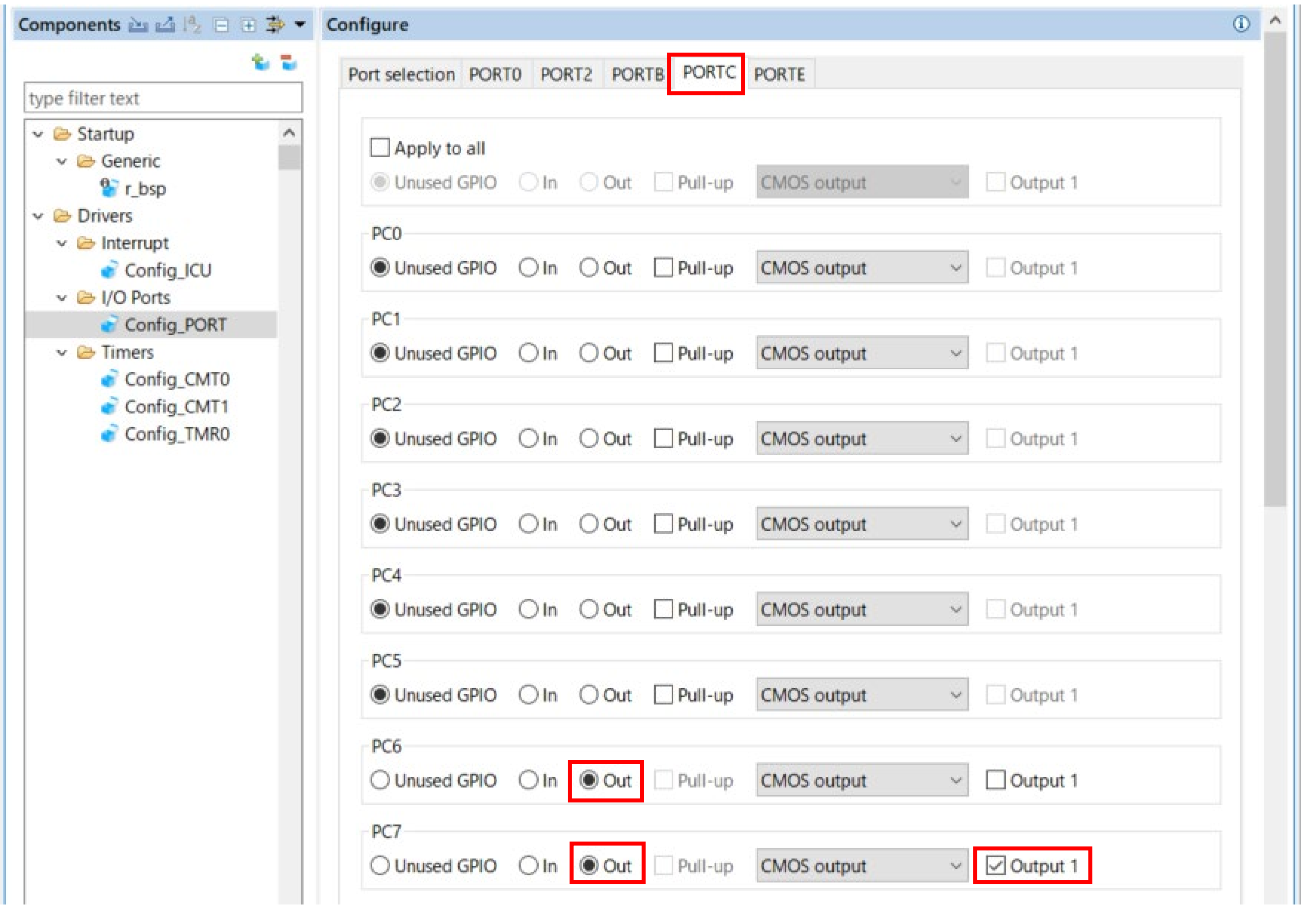
PB5
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PB6
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PB7
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

Figure 4-29 Select PORTB tab

Select ‘PORTC’ tab.



□ Apply to all
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PC0
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PC1
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PC2
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PC3
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PC4
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

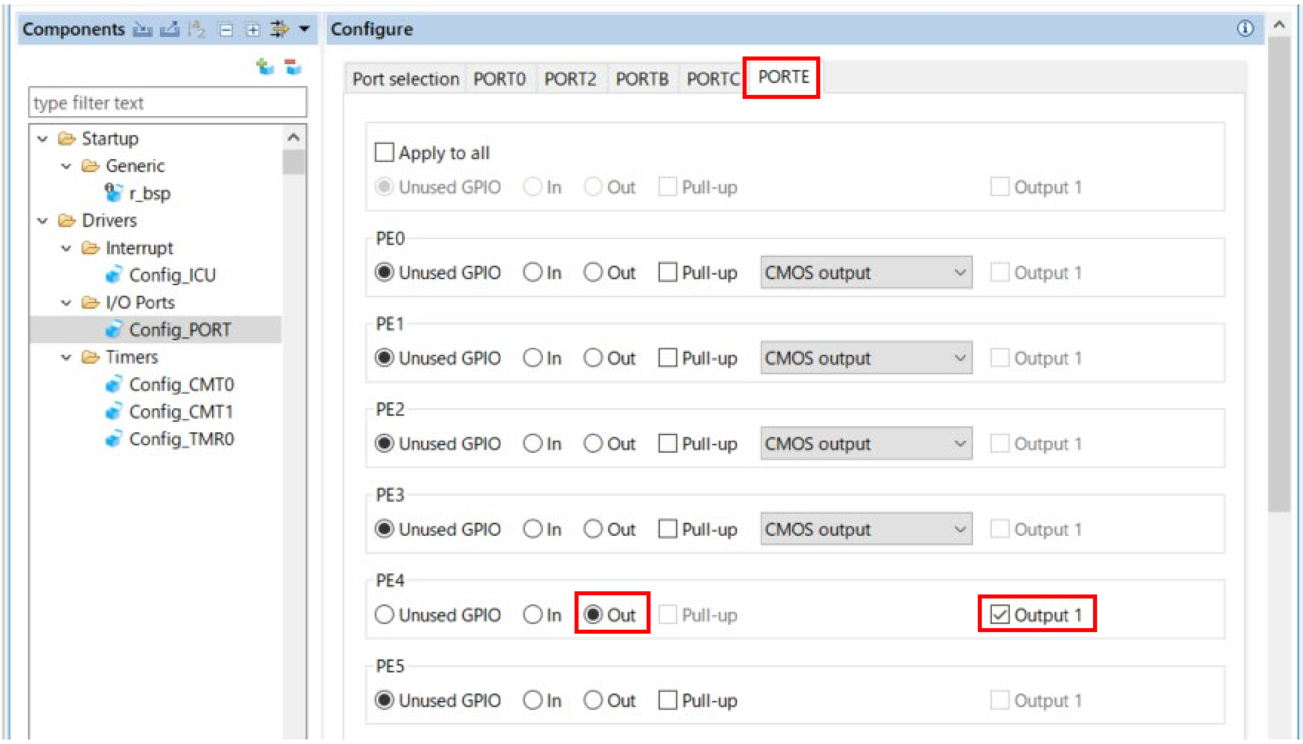
PC5
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PC6
○ Unused GPIO ○ In ● Out □ Pull-up CMOS output □ Output 1

PC7
○ Unused GPIO ○ In ● Out □ Pull-up CMOS output ☑ Output 1

Figure 4-30 Select PORTC tab

Select ‘PORTE’ tab.



□ Apply to all
● Unused GPIO ○ In ○ Out □ Pull-up □ Output 1

PE0
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PE1
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PE2
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PE3
● Unused GPIO ○ In ○ Out □ Pull-up CMOS output □ Output 1

PE4
○ Unused GPIO ○ In ● Out □ Pull-up ☑ Output 1

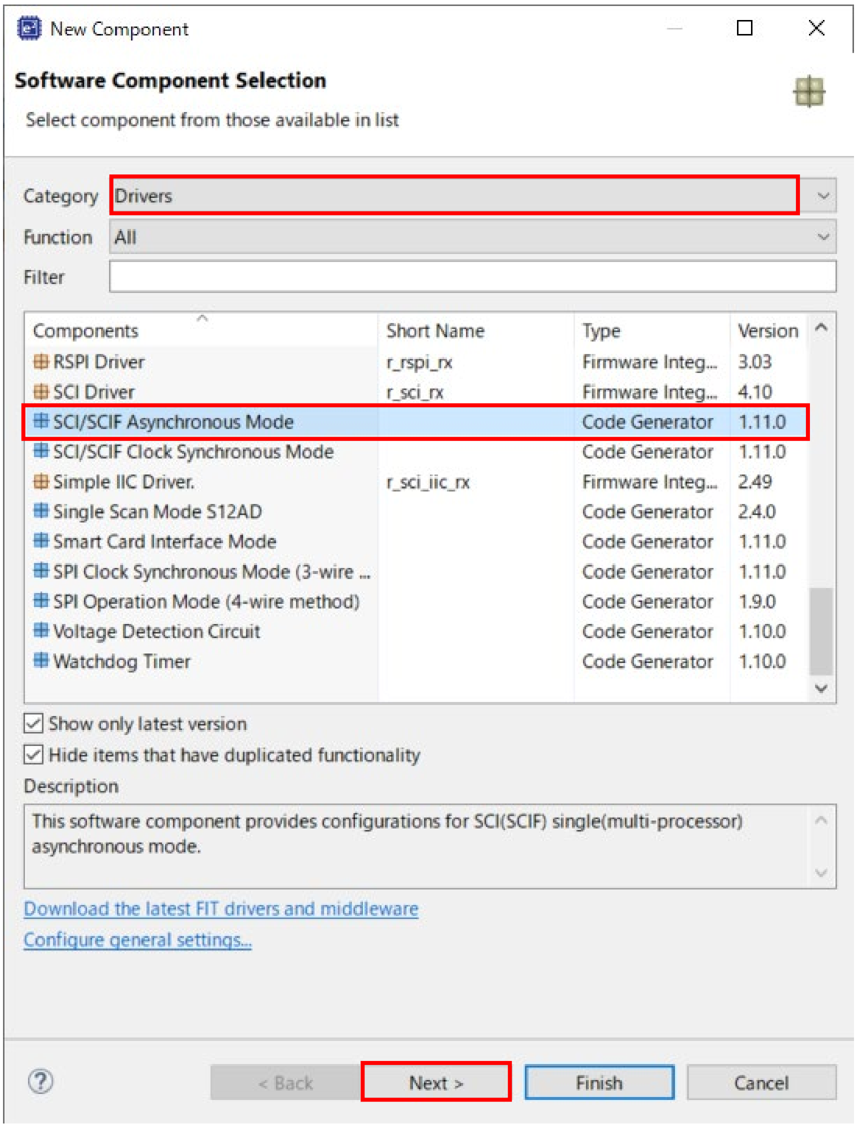
PE5
● Unused GPIO ○ In ○ Out □ Pull-up □ Output 1

Figure 4-31 Select PORTE tab

4.6.6 SCI/SCIF Asynchronous Mode

In the RSKRX140, SCI1 is connected via a Renesas RL78/G1C to provide a USB virtual COM port as shown in the schematic.

Click the 'Add component' icon. In 'Software Component Selection' dialog -> Type, select 'Drivers'. Select 'SCI/SCIF Asynchronous Mode' as shown in **Figure 4-32** then click 'Next'.



Software Component Selection

Select component from those available in list

Category: Drivers

Function: All

Filter:

Components	Short Name	Type	Version
RSPI Driver	r_rspi_rx	Firmware Integ...	3.03
SCI Driver	r_sci_rx	Firmware Integ...	4.10
SCI/SCIF Asynchronous Mode		Code Generator	1.11.0
SCI/SCIF Clock Synchronous Mode		Code Generator	1.11.0
Simple IIC Driver.	r_sci_iic_rx	Firmware Integ...	2.49
Single Scan Mode S12AD		Code Generator	2.4.0
Smart Card Interface Mode		Code Generator	1.11.0
SPI Clock Synchronous Mode (3-wire ...		Code Generator	1.11.0
SPI Operation Mode (4-wire method)		Code Generator	1.9.0
Voltage Detection Circuit		Code Generator	1.10.0
Watchdog Timer		Code Generator	1.10.0

☑ Show only latest version

☑ Hide items that have duplicated functionality

Description

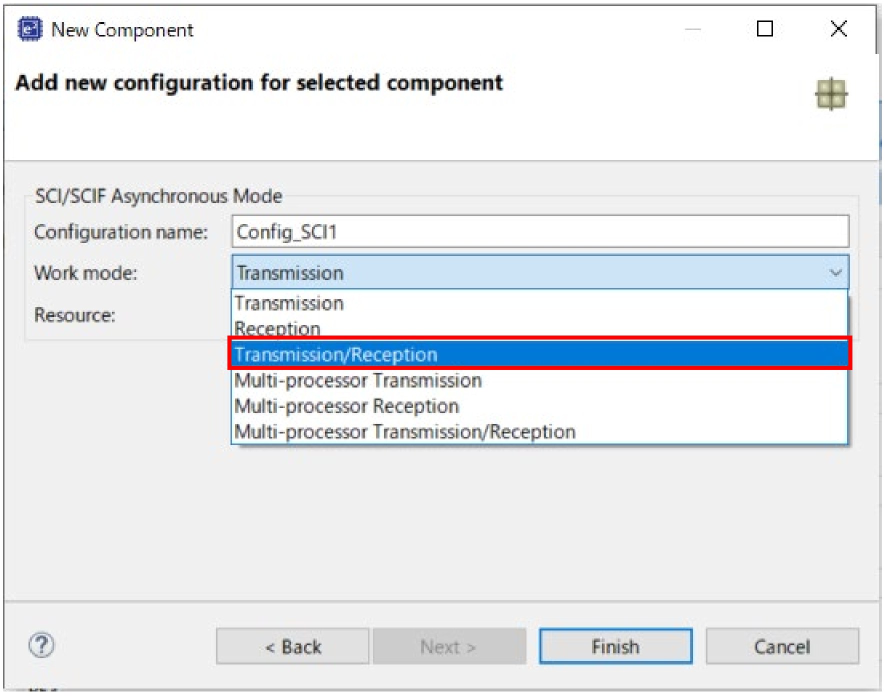
This software component provides configurations for SCI(SCIF) single(multi-processor) asynchronous mode.

Download the latest FIT drivers and middleware

Configure general settings...

Figure 4-32 Select SCI/SCIF Asynchronous Mode

In 'Add new configuration for selected component' dialog -> Work mode, select 'Transmission/Reception' as shown in **Figure 4-33** below.



New Component

Add new configuration for selected component

SCI/SCIF Asynchronous Mode

Configuration name: Config_SCI1

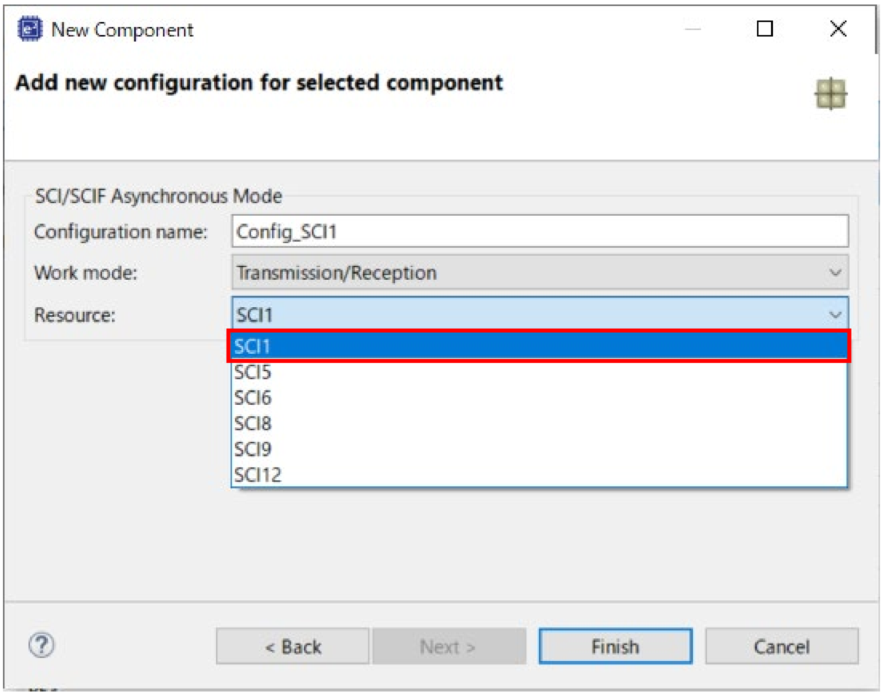
Work mode: Transmission

Resource:

Transmission
Reception
Transmission/Reception
Multi-processor Transmission
Multi-processor Reception
Multi-processor Transmission/Reception
< Back
Next >
Finish
Cancel

Figure 4-33 Select Work mode – Transmission/Reception

In 'Resource', select 'SCI1' as shown in **Figure 4-34** below.



New Component

Add new configuration for selected component

SCI/SCIF Asynchronous Mode

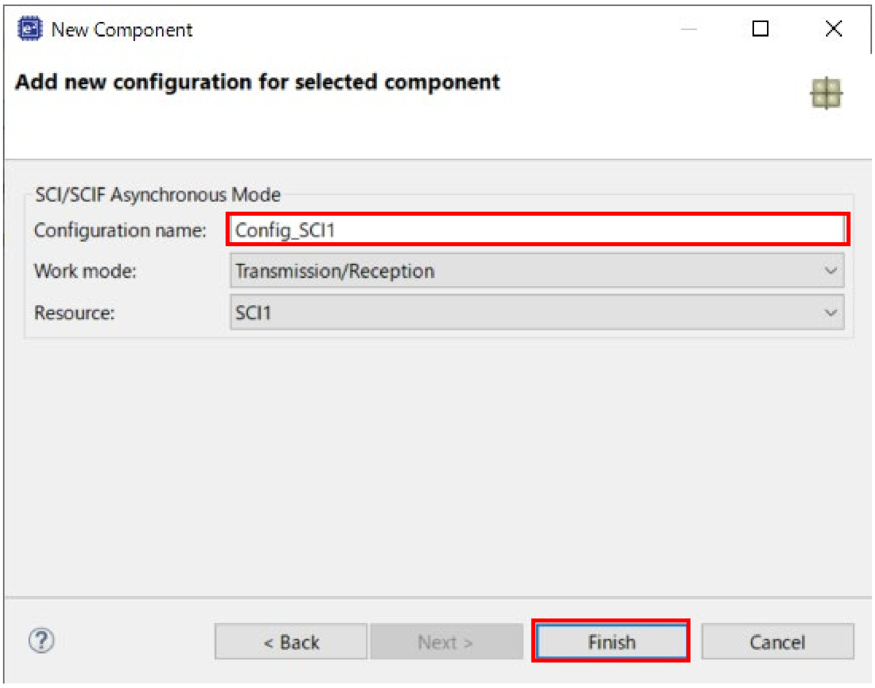
Configuration name: Config_SCI1

Work mode: Transmission/Reception

Resource: SCI1

Figure 4-34 Select Resource – SCI1

Ensure that the ‘Configuration name’ updates to ‘Config_SCI1’ as shown in **Figure 4-35** below then click ‘Finish’.



New Component

Add new configuration for selected component

SCI/SCIF Asynchronous Mode

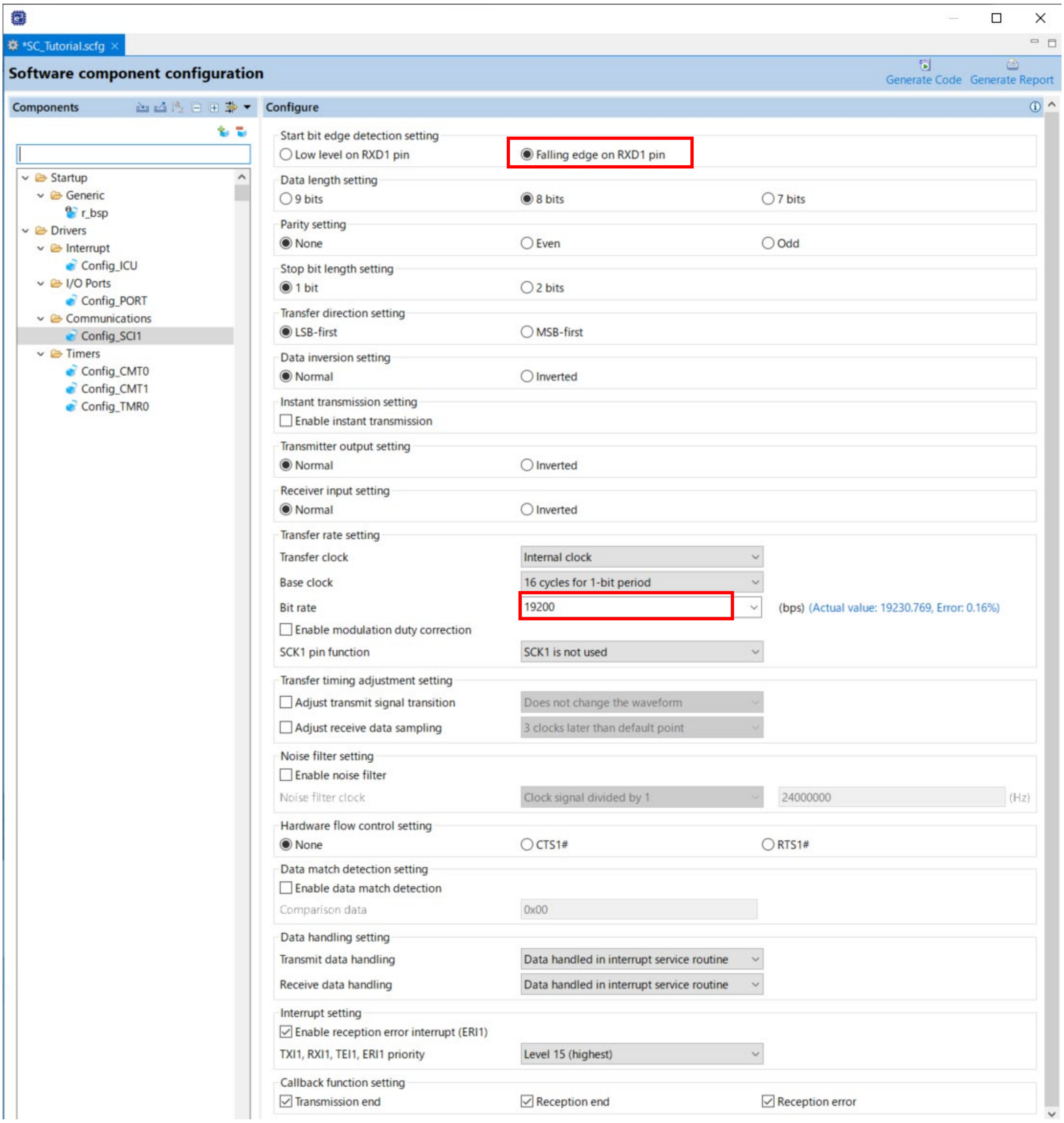
Configuration name: Config_SCI1

Work mode: Transmission/Reception

Resource: SCI1

Figure 4-35 Ensure Configuration name - Config_SCI1

Configure SCI1 as shown in **Figure 4-36**. Ensure the ‘Start bit edge detection’ is set as ‘Falling edge on RXD1 pin’ and the ‘Bit rate’ is set to 19200 bps. All other settings remain at their defaults.



Start bit edge detection setting
○ Low level on RXD1 pin ● Falling edge on RXD1 pin

Data length setting
○ 9 bits ● 8 bits ○ 7 bits

Parity setting
● None ○ Even ○ Odd

Stop bit length setting
● 1 bit ○ 2 bits

Transfer direction setting
● LSB-first ○ MSB-first

Data inversion setting
● Normal ○ Inverted

Instant transmission setting
□ Enable instant transmission

Transmitter output setting
● Normal ○ Inverted

Receiver input setting
● Normal ○ Inverted

Transfer rate setting
Transfer clock: Internal clock
Base clock: 16 cycles for 1-bit period
Bit rate: 19200 (bps) (Actual value: 19230.769, Error: 0.16%)
□ Enable modulation duty correction
SCK1 pin function: SCK1 is not used

Transfer timing adjustment setting
□ Adjust transmit signal transition: Does not change the waveform
□ Adjust receive data sampling: 3 clocks later than default point

Noise filter setting
□ Enable noise filter
Noise filter clock: Clock signal divided by 1 24000000 (Hz)

Hardware flow control setting
● None ○ CTS1# ○ RTS1#

Data match detection setting
□ Enable data match detection
Comparison data: 0x00

Data handling setting
Transmit data handling: Data handled in interrupt service routine
Receive data handling: Data handled in interrupt service routine

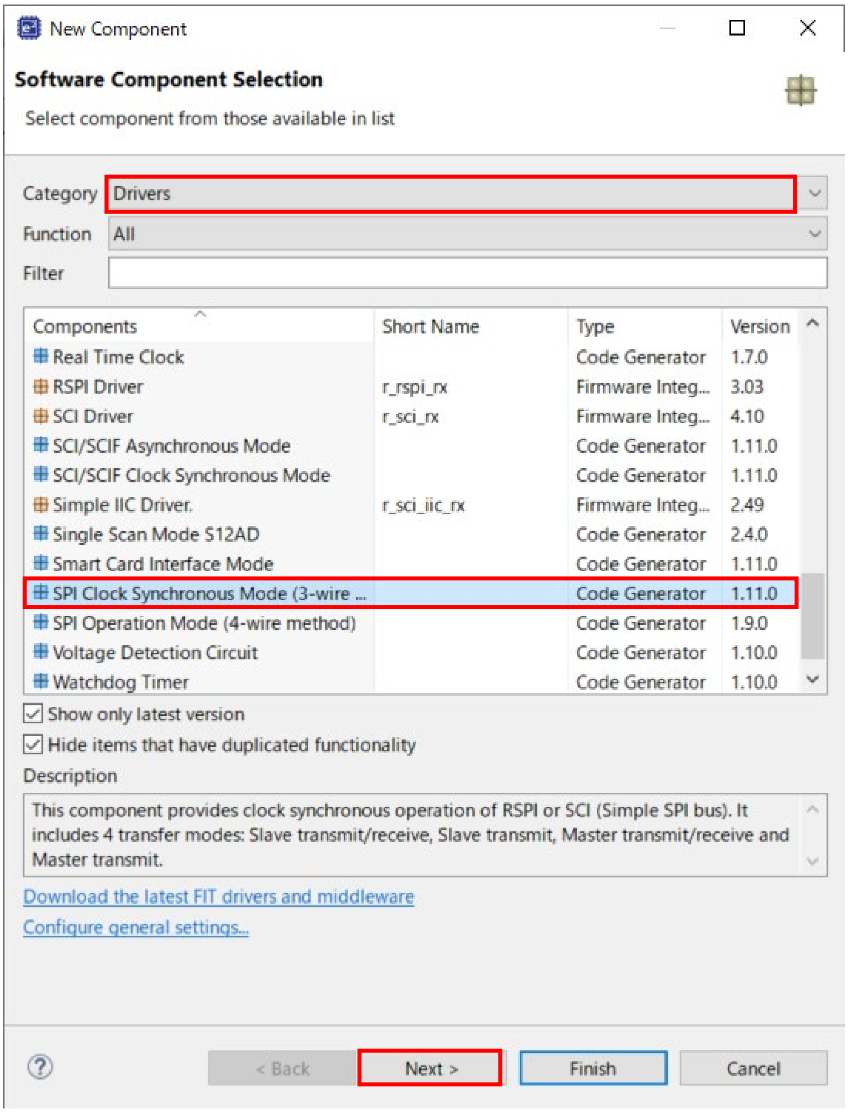
Interrupt setting
☑ Enable reception error interrupt (ERI1)
TXI1, RXI1, TEI1, ERI1 priority: Level 15 (highest)

Callback function setting
☑ Transmission end ☑ Reception end ☑ Reception error

Figure 4-36 Config_SCI1 setting

4.6.7 SPI Clock Synchronous Mode

In the RSKRX140, SCI6 is used as an SPI master for the Pmod LCD on the PMOD1 connector as shown in the schematic. Click the ‘Add component’ icon. In ‘Software Component Selection’ dialog -> Type, select ‘Drivers’. Select ‘SPI Clock Synchronous Mode’ as shown in **Figure 4-37** then click ‘Next’.



Software Component Selection

Select component from those available in list

Category: Drivers

Function: All

Filter:

Components	Short Name	Type	Version
Real Time Clock		Code Generator	1.7.0
RSPI Driver	r_rspi_rx	Firmware Integ...	3.03
SCI Driver	r_sci_rx	Firmware Integ...	4.10
SCI/SCIF Asynchronous Mode		Code Generator	1.11.0
SCI/SCIF Clock Synchronous Mode		Code Generator	1.11.0
Simple IIC Driver.	r_sci_iic_rx	Firmware Integ...	2.49
Single Scan Mode S12AD		Code Generator	2.4.0
Smart Card Interface Mode		Code Generator	1.11.0
SPI Clock Synchronous Mode (3-wire ...		Code Generator	1.11.0
SPI Operation Mode (4-wire method)		Code Generator	1.9.0
Voltage Detection Circuit		Code Generator	1.10.0
Watchdog Timer		Code Generator	1.10.0

☑ Show only latest version

☑ Hide items that have duplicated functionality

Description

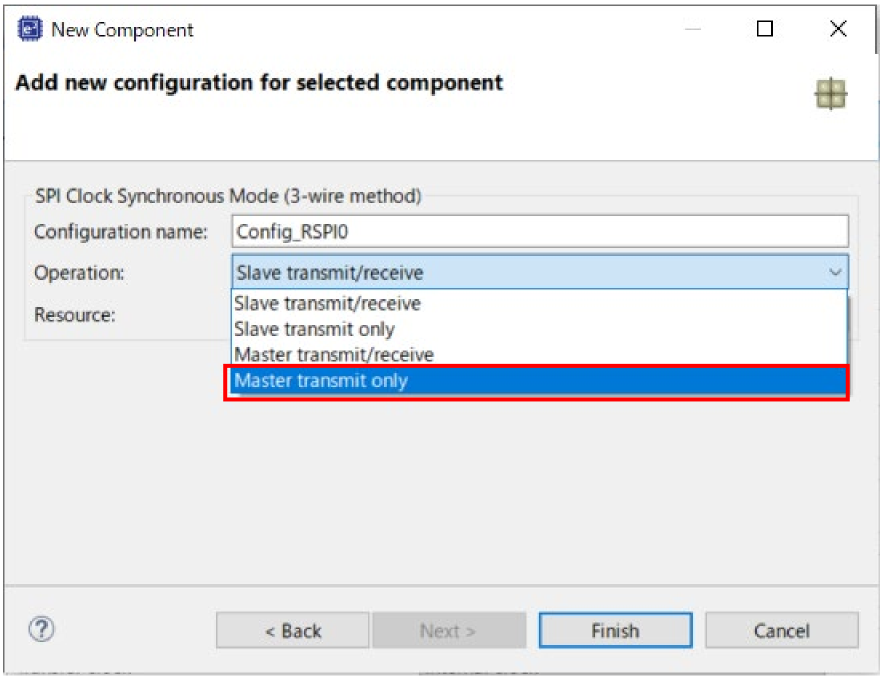
This component provides clock synchronous operation of RSPI or SCI (Simple SPI bus). It includes 4 transfer modes: Slave transmit/receive, Slave transmit, Master transmit/receive and Master transmit.

Download the latest FIT drivers and middleware

Configure general settings...

Figure 4-37 Select SPI Clock Synchronous Mode

In ‘Add new configuration for selected component’ dialog -> Operation, select ‘Master transmit only’ as shown in **Figure 4-38** below.



New Component

Add new configuration for selected component

SPI Clock Synchronous Mode (3-wire method)

Configuration name: Config_RSPI0

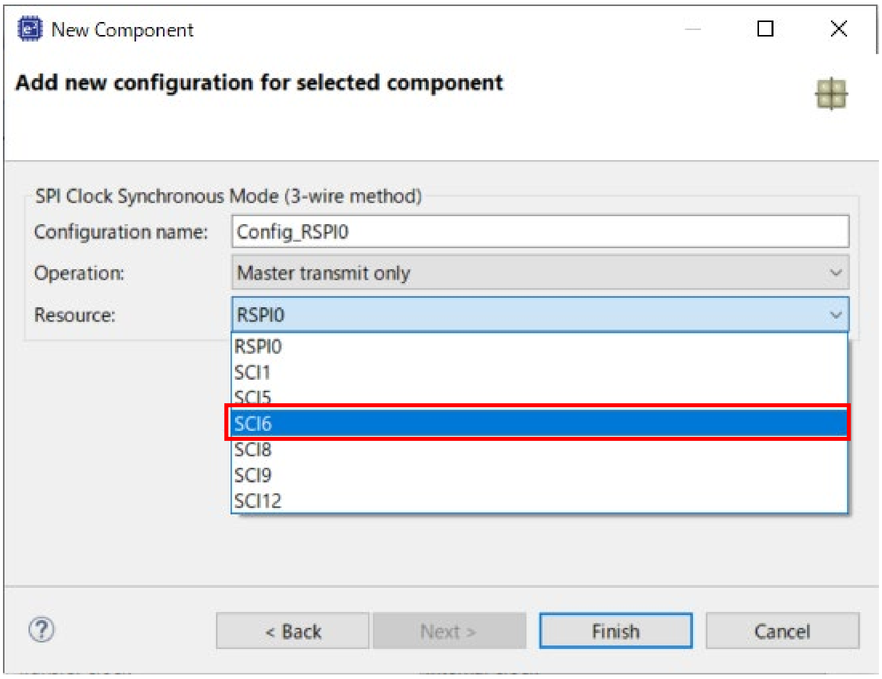
Operation: Slave transmit/receive

Resource:

- Slave transmit/receive
- Slave transmit only
- Master transmit/receive
- Master transmit only

Figure 4-38 Select Operation – Master transmit only

In ‘Resource’, select ‘SCI6’ as shown in **Figure 4-39** below.



New Component

Add new configuration for selected component

SPI Clock Synchronous Mode (3-wire method)

Configuration name: Config_RSPI0

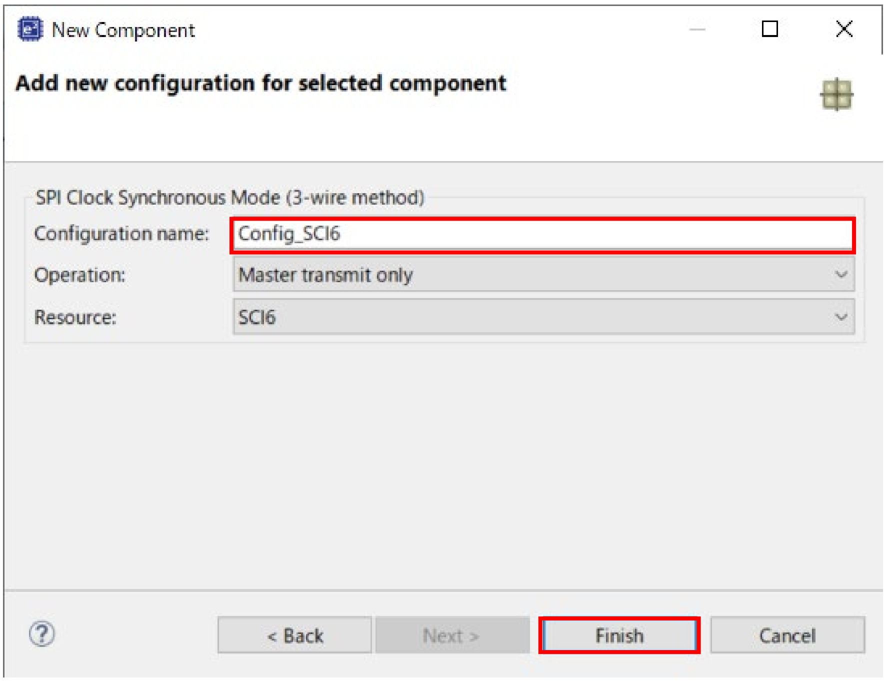
Operation: Master transmit only

Resource: RSPI0

- RSPI0
- SCI1
- SCI5
- SCI6
- SCI8
- SCI9
- SCI12

Figure 4-39 Select Resource – SCI6

Ensure that the ‘Configuration name’ updates to ‘Config_SCI6’ as shown in **Figure 4-40** below then click ‘Finish’.



New Component

Add new configuration for selected component

SPI Clock Synchronous Mode (3-wire method)

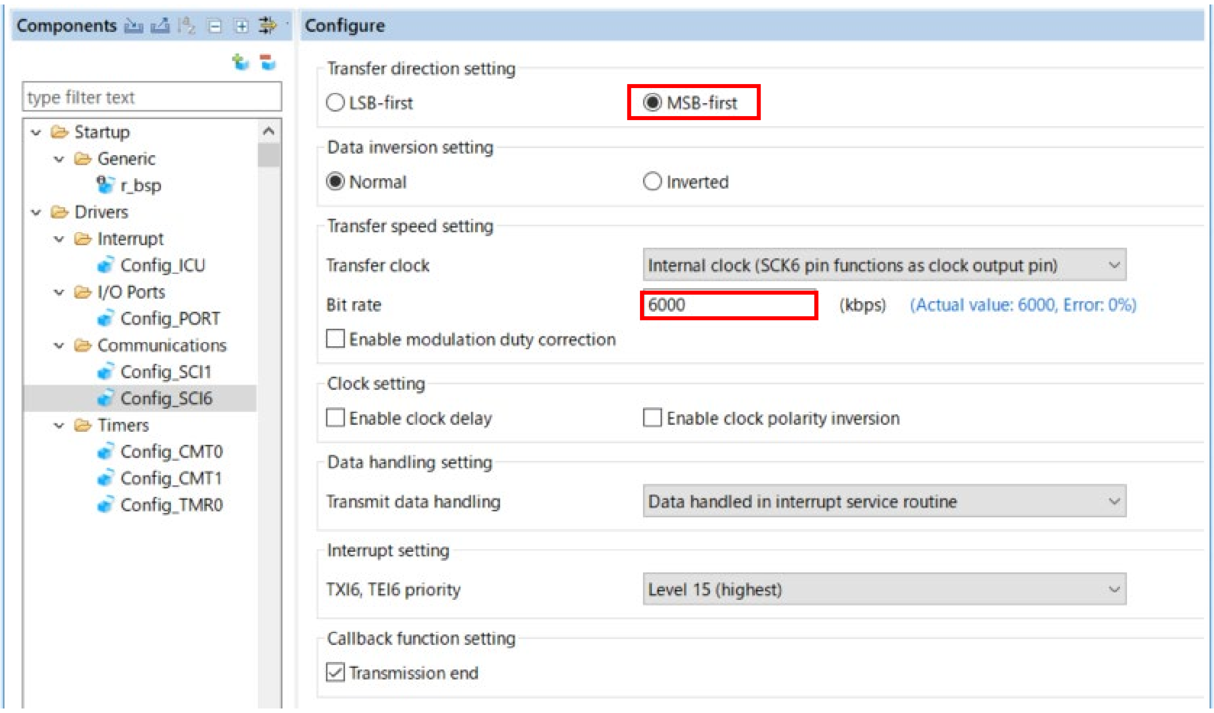
Configuration name: Config_SCI6

Operation: Master transmit only

Resource: SCI6

Figure 4-40 Ensure Configuration name - Config_SCI6

Configure SCI6 as shown in **Figure 4-41**. Ensure the ‘Transfer direction’ is set as ‘MSB-first’ and the ‘Bit rate’ is set to 6000 kbps. All other settings remain at their defaults.



Transfer direction setting

○ LSB-first ● MSB-first

Data inversion setting

● Normal ○ Inverted

Transfer speed setting

Transfer clock: Internal clock (SCK6 pin functions as clock output pin)

Bit rate: 6000 (kbps) (Actual value: 6000, Error: 0%)

□ Enable modulation duty correction

Clock setting

□ Enable clock delay □ Enable clock polarity inversion

Data handling setting

Transmit data handling: Data handled in interrupt service routine

Interrupt setting

TXI6, TEI6 priority: Level 15 (highest)

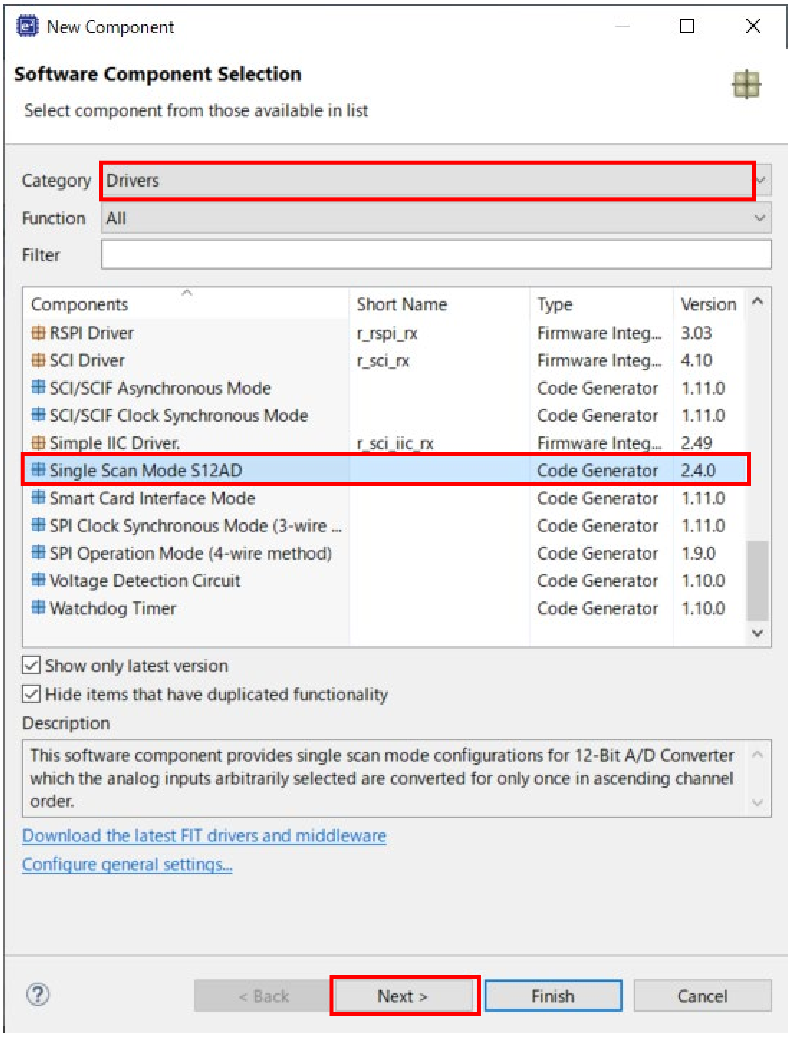
Callback function setting

☑ Transmission end

Figure 4-41 Config_SCI6 setting

4.6.8 Single Scan Mode S12AD

We will be using the S12AD in Single Scan Mode on the AN000 input, which is connected to the RV1 potentiometer output on the RSK. The conversion start trigger will be via the pin connected to SW3. Click the 'Add component' icon. In 'Software Component Selection' dialog -> Type, select 'Drivers'. Select 'Single Scan Mode S12AD' as shown in **Figure 4-42** then click 'Next'.



New Component

Software Component Selection

Select component from those available in list

Category: Drivers

Function: All

Filter:

Components	Short Name	Type	Version
RSPI Driver	r_rspi_rx	Firmware Integ...	3.03
SCI Driver	r_sci_rx	Firmware Integ...	4.10
SCI/SCIF Asynchronous Mode		Code Generator	1.11.0
SCI/SCIF Clock Synchronous Mode		Code Generator	1.11.0
Simple IIC Driver.	r_sci_iic_rx	Firmware Integ...	2.49
Single Scan Mode S12AD		Code Generator	2.4.0
Smart Card Interface Mode		Code Generator	1.11.0
SPI Clock Synchronous Mode (3-wire ...		Code Generator	1.11.0
SPI Operation Mode (4-wire method)		Code Generator	1.9.0
Voltage Detection Circuit		Code Generator	1.10.0
Watchdog Timer		Code Generator	1.10.0

☑ Show only latest version

☑ Hide items that have duplicated functionality

Description

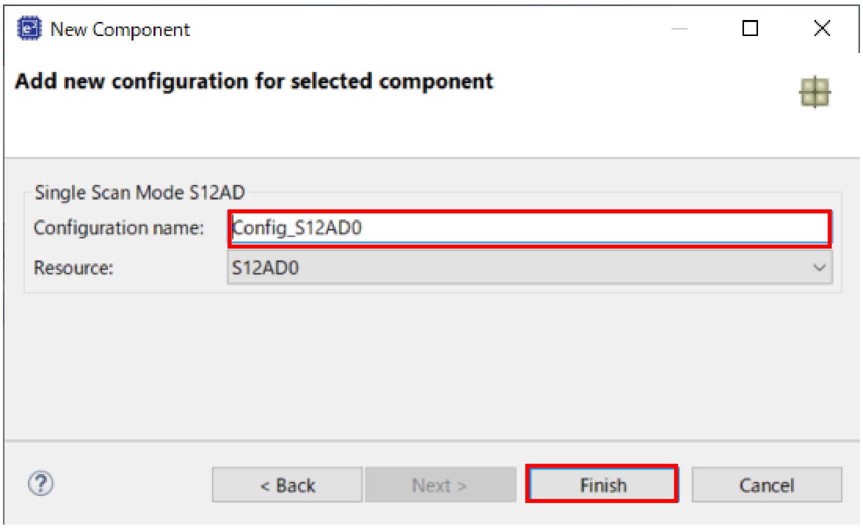
This software component provides single scan mode configurations for 12-Bit A/D Converter which the analog inputs arbitrarily selected are converted for only once in ascending channel order.

Download the latest FIT drivers and middleware

Configure general settings...

Figure 4-42 Select Single Scan Mode S12AD

Ensure that the 'Configuration name' is'Config_S12AD0' as shown in **Figure 4-43** below then click ‘Finish’.



New Component

Add new configuration for selected component

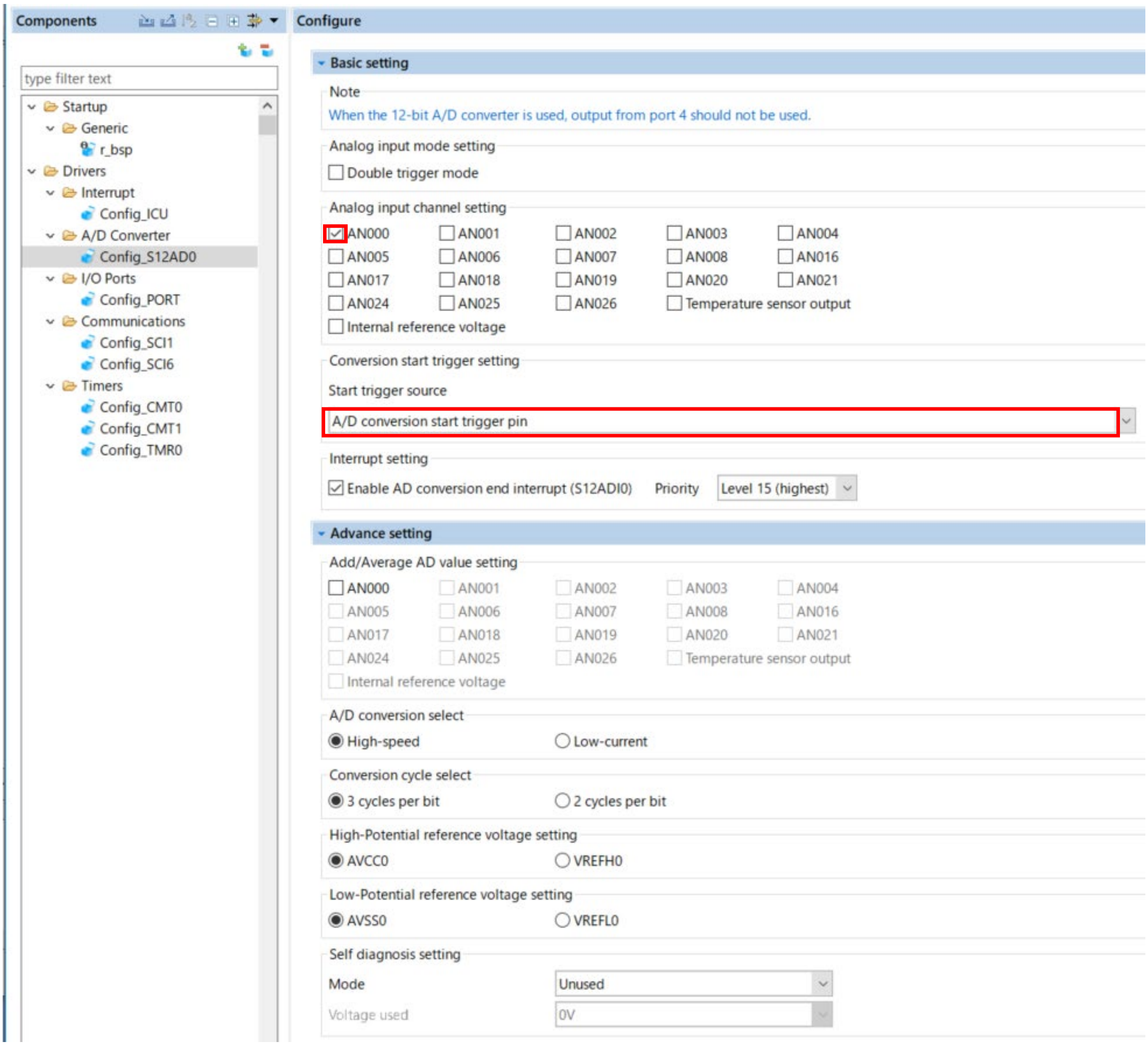
Single Scan Mode S12AD

Configuration name: Config_S12AD0

Resource: S12AD0

Figure 4-43 Ensure Configuration name - S12AD0

Configure S12AD0 as shown in **Figure 4-44** and **Figure 4-46** and **Figure 4-45**. Ensure the ‘Analog input channel’ tick box for AN000 is checked and the ‘Start trigger source’ is set to ‘A/D conversion start trigger pin’. All other settings remain at their defaults.



Components

type filter text

- Startup
  - Generic
    - r_bsp
- Drivers
  - Interrupt
    - Config_ICU
  - A/D Converter
    - Config_S12AD0
  - I/O Ports
    - Config_PORT
  - Communications
    - Config_SCI1
    - Config_SCI6
  - Timers
    - Config_CMT0
    - Config_CMT1
    - Config_TMR0

Configure

Basic setting

Note

When the 12-bit A/D converter is used, output from port 4 should not be used.

Analog input mode setting

□ Double trigger mode

Analog input channel setting

☑ AN000 □ AN001 □ AN002 □ AN003 □ AN004
□ AN005 □ AN006 □ AN007 □ AN008 □ AN016
□ AN017 □ AN018 □ AN019 □ AN020 □ AN021
□ AN024 □ AN025 □ AN026 □ Temperature sensor output
□ Internal reference voltage

Conversion start trigger setting

Start trigger source: A/D conversion start trigger pin

Interrupt setting

☑ Enable AD conversion end interrupt (S12ADI0) Priority: Level 15 (highest)

Advance setting

Add/Average AD value setting

□ AN000 □ AN001 □ AN002 □ AN003 □ AN004
□ AN005 □ AN006 □ AN007 □ AN008 □ AN016
□ AN017 □ AN018 □ AN019 □ AN020 □ AN021
□ AN024 □ AN025 □ AN026 □ Temperature sensor output
□ Internal reference voltage

A/D conversion select

● High-speed ○ Low-current

Conversion cycle select

● 3 cycles per bit ○ 2 cycles per bit

High-Potential reference voltage setting

● AVCC0 ○ VREFH0

Low-Potential reference voltage setting

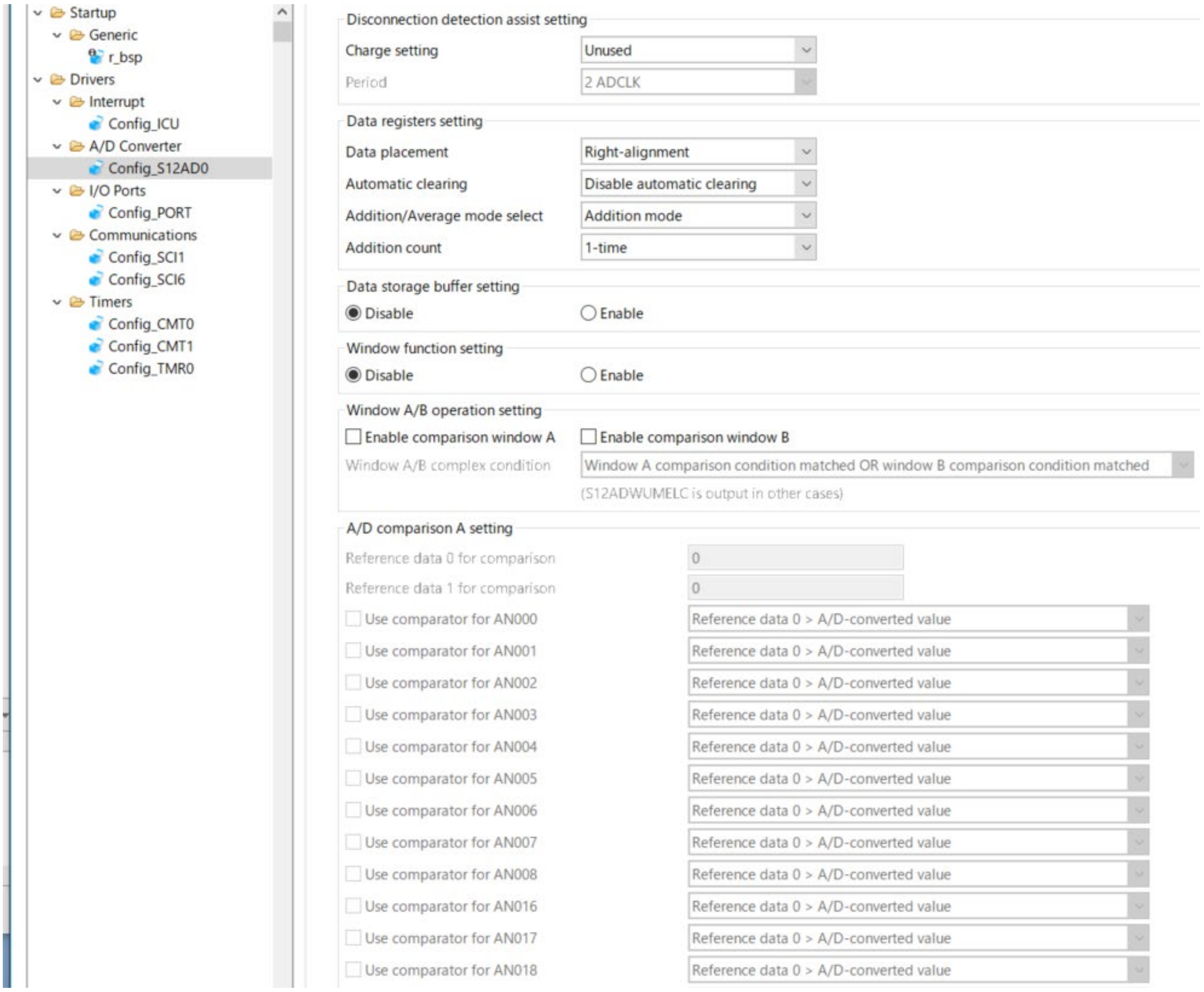
● AVSS0 ○ VREFL0

Self diagnosis setting

Mode: Unused

Voltage used: 0V

Figure 4-44 Config_S12AD0 setting (1)



- Startup
  - Generic
    - r_bsp
- Drivers
  - Interrupt
    - Config_ICU
  - A/D Converter
    - Config_S12AD0
  - I/O Ports
    - Config_PORT
  - Communications
    - Config_SCI1
    - Config_SCI6
  - Timers
    - Config_CMT0
    - Config_CMT1
    - Config_TMR0

Disconnection detection assist setting

Charge setting	Unused
Period	2 ADCLK

Data registers setting

Data placement	Right-alignment
Automatic clearing	Disable automatic clearing
Addition/Average mode select	Addition mode
Addition count	1-time

Data storage buffer setting

● Disable ○ Enable

Window function setting

● Disable ○ Enable

Window A/B operation setting

□ Enable comparison window A □ Enable comparison window B

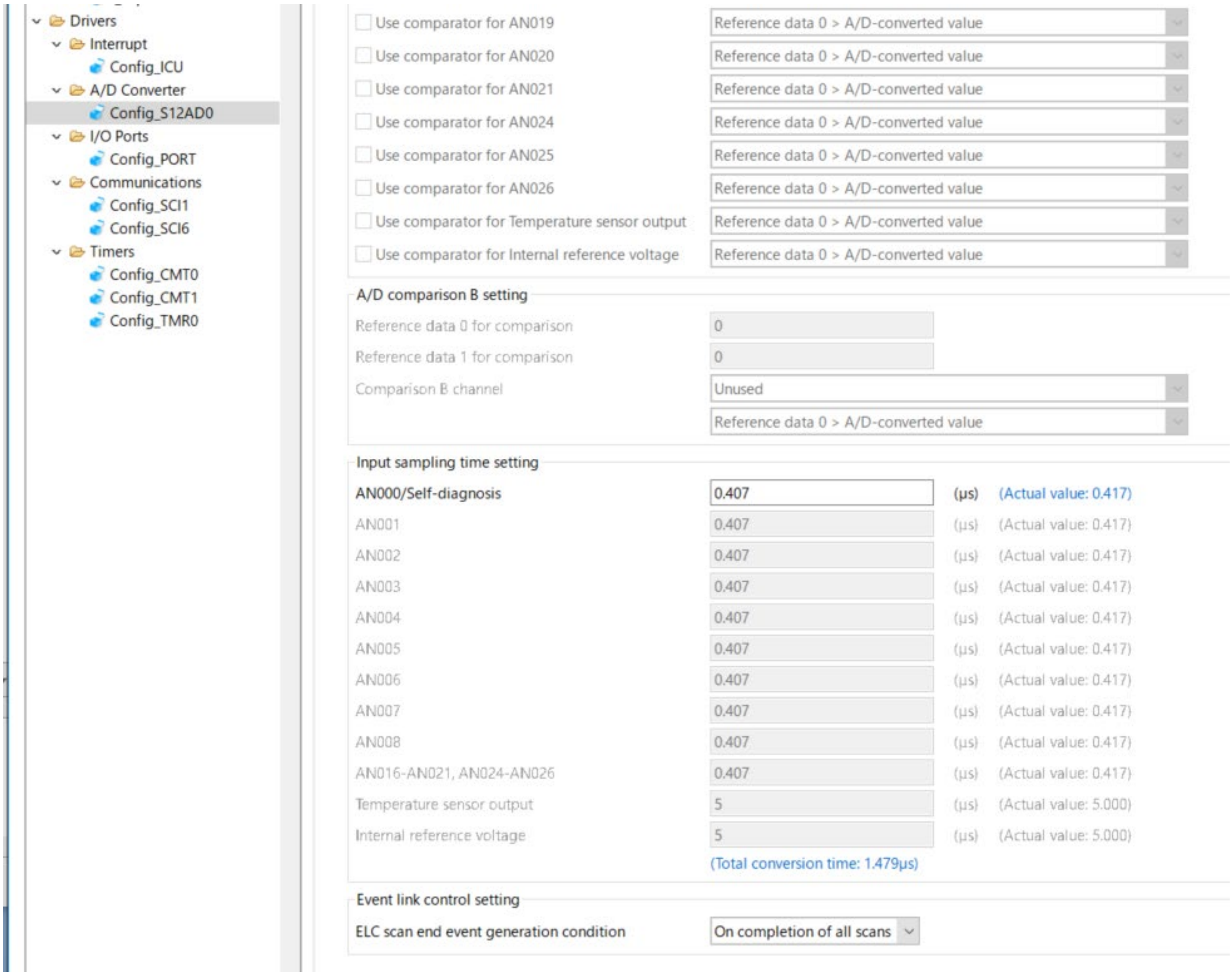
Window A/B complex condition: Window A comparison condition matched OR window B comparison condition matched

(S12ADWUMELC is output in other cases)

A/D comparison A setting

Reference data 0 for comparison	0
Reference data 1 for comparison	0
□ Use comparator for AN000	Reference data 0 > A/D-converted value
□ Use comparator for AN001	Reference data 0 > A/D-converted value
□ Use comparator for AN002	Reference data 0 > A/D-converted value
□ Use comparator for AN003	Reference data 0 > A/D-converted value
□ Use comparator for AN004	Reference data 0 > A/D-converted value
□ Use comparator for AN005	Reference data 0 > A/D-converted value
□ Use comparator for AN006	Reference data 0 > A/D-converted value
□ Use comparator for AN007	Reference data 0 > A/D-converted value
□ Use comparator for AN008	Reference data 0 > A/D-converted value
□ Use comparator for AN016	Reference data 0 > A/D-converted value
□ Use comparator for AN017	Reference data 0 > A/D-converted value
□ Use comparator for AN018	Reference data 0 > A/D-converted value

Figure 4-45 Config_S12AD0 setting (2)



□ Use comparator for AN019 — Reference data 0 > A/D-converted value

□ Use comparator for AN020 — Reference data 0 > A/D-converted value

□ Use comparator for AN021 — Reference data 0 > A/D-converted value

□ Use comparator for AN024 — Reference data 0 > A/D-converted value

□ Use comparator for AN025 — Reference data 0 > A/D-converted value

□ Use comparator for AN026 — Reference data 0 > A/D-converted value

□ Use comparator for Temperature sensor output — Reference data 0 > A/D-converted value

□ Use comparator for Internal reference voltage — Reference data 0 > A/D-converted value

A/D comparison B setting

Reference data 0 for comparison	0
Reference data 1 for comparison	0
Comparison B channel	Unused
	Reference data 0 > A/D-converted value

Input sampling time setting

AN000/Self-diagnosis	0.407	(μs)	(Actual value: 0.417)
AN001	0.407	(μs)	(Actual value: 0.417)
AN002	0.407	(μs)	(Actual value: 0.417)
AN003	0.407	(μs)	(Actual value: 0.417)
AN004	0.407	(μs)	(Actual value: 0.417)
AN005	0.407	(μs)	(Actual value: 0.417)
AN006	0.407	(μs)	(Actual value: 0.417)
AN007	0.407	(μs)	(Actual value: 0.417)
AN008	0.407	(μs)	(Actual value: 0.417)
AN016-AN021, AN024-AN026	0.407	(μs)	(Actual value: 0.417)
Temperature sensor output	5	(μs)	(Actual value: 5.000)
Internal reference voltage	5	(μs)	(Actual value: 5.000)

(Total conversion time: 1.479μs)

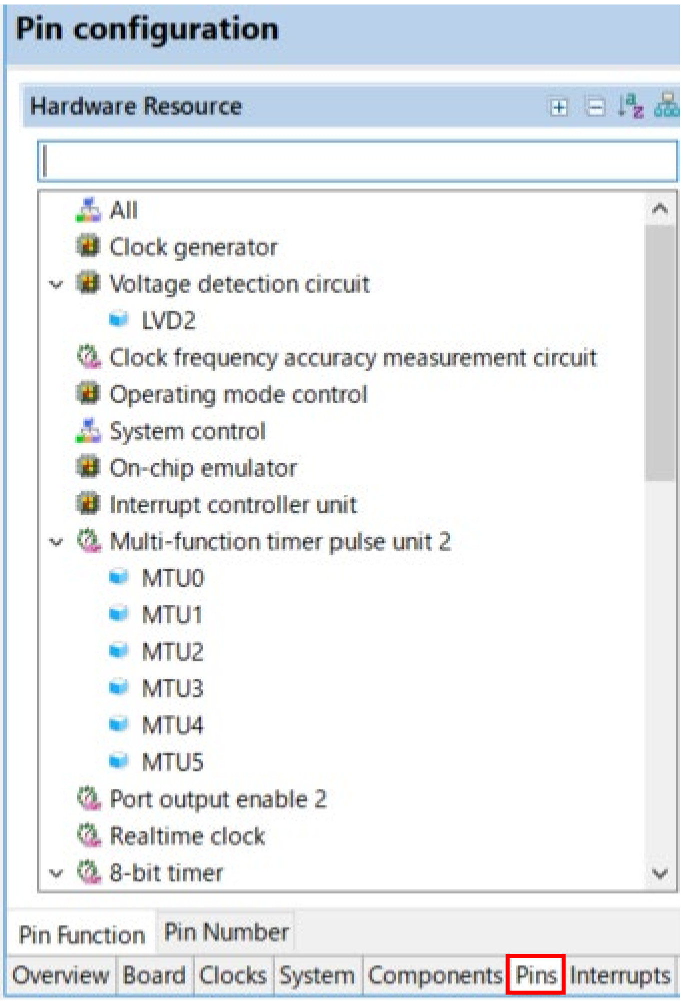
Event link control setting

ELC scan end event generation condition: On completion of all scans

Figure 4-46 Config_S12AD0 setting (3)

4.7 The ‘Pins’ tabbed page

Smart Configurator assigns pins to the software components that are added to the project. Assignment of the pins can be changed using the Pins page.

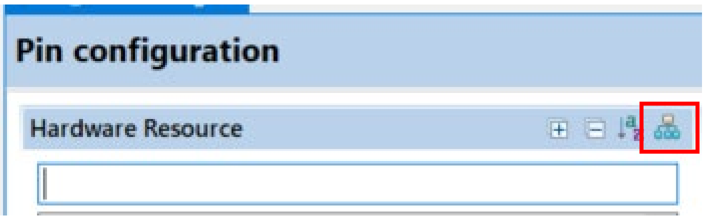

Pin configuration

Hardware Resource

Figure 4-47 The ‘Pins’ tabbed page

4.7.1 Change pin assignment of a software component

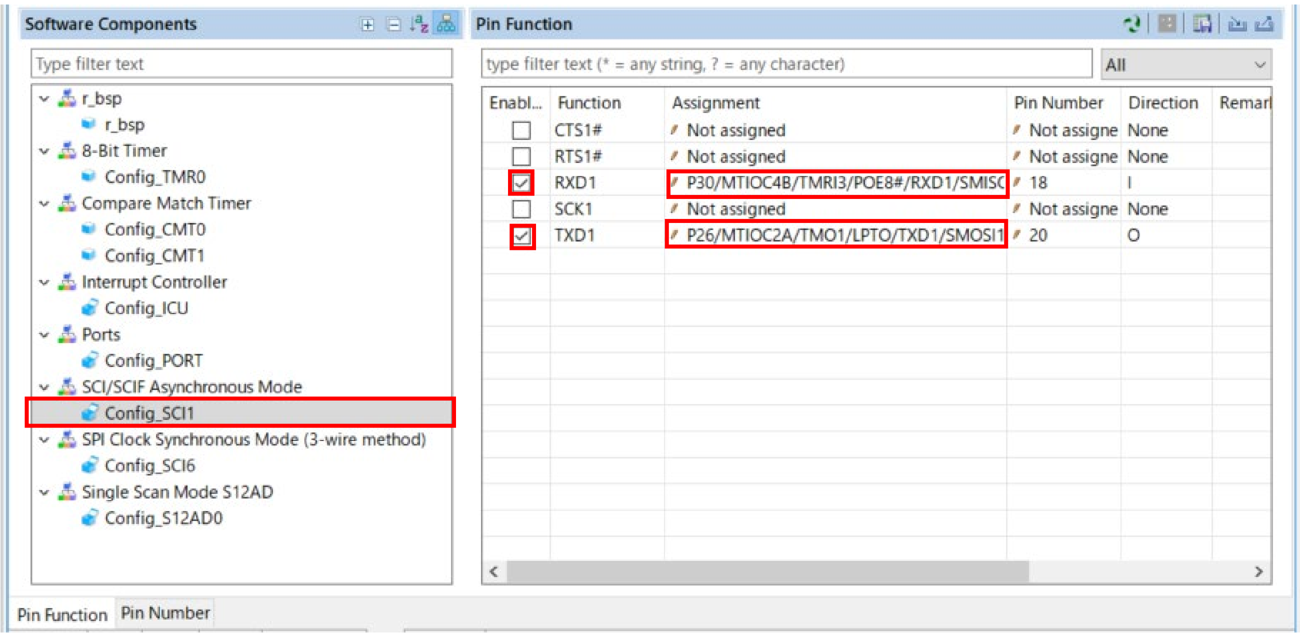
To change the pin assignment of a software component in the Pin Function list, click 品 to change view to show by Software Components.


Pin configuration

Hardware Resource

Figure 4-48 Change view to show by Hardware Resource

Select the Config_SCI1 of Software Components. In the Pin Function list -> Assignment column, Ensure the ‘Enable’ tick box of RXD1 and TXD1 are checked and Assignment column of RXD1 is P30 and TXD1 is P26 as shown in **Figure 4-49**.

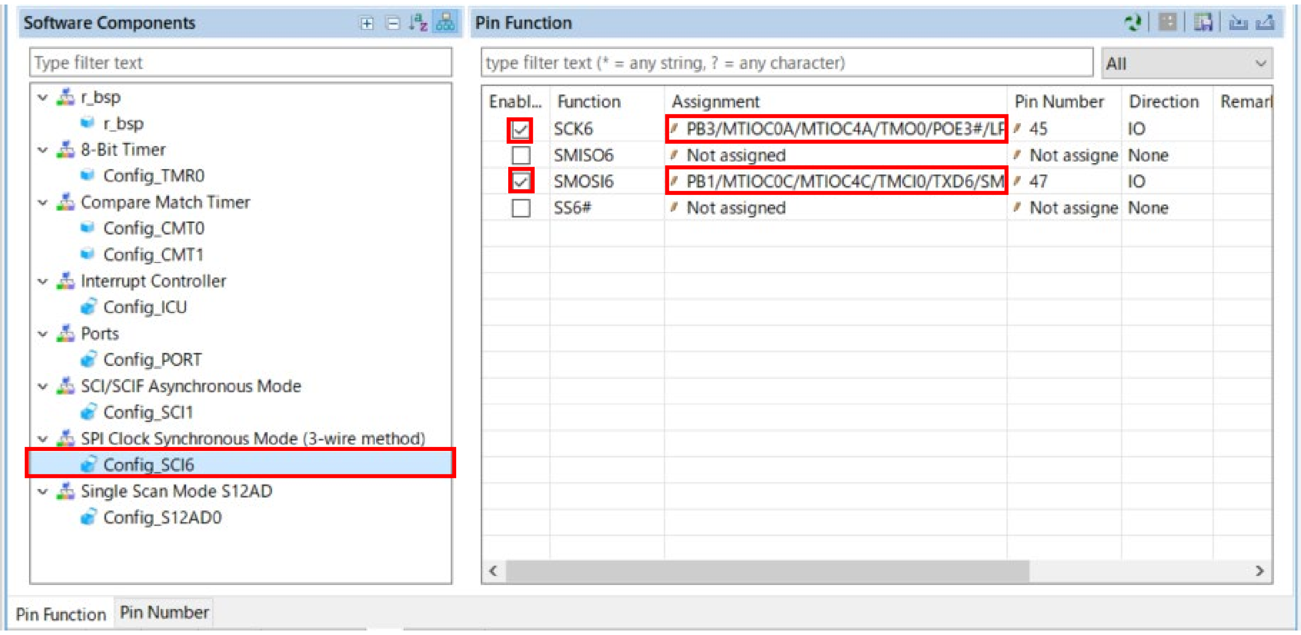


- r_bsp
  - r_bsp
- 8-Bit Timer
  - Config_TMR0
- Compare Match Timer
  - Config_CMT0
  - Config_CMT1
- Interrupt Controller
  - Config_ICU
- Ports
  - Config_PORT
- SCI/SCIF Asynchronous Mode
  - Config_SCI1
- SPI Clock Synchronous Mode (3-wire method)
  - Config_SCI6
- Single Scan Mode S12AD
  - Config_S12AD0

Enabl...	Function	Assignment	Pin Number	Direction	Remar
□	CTS1#	Not assigned	Not assigne	None	
□	RTS1#	Not assigned	Not assigne	None	
☑	RXD1	P30/MTIOC4B/TMRI3/POE8#/RXD1/SMISC	18	I	
□	SCK1	Not assigned	Not assigne	None	
☑	TXD1	P26/MTIOC2A/TMO1/LPTO/TXD1/SMOSI1	20	O	

Figure 4-49 Configure pin assignment - Config_SCI1

Select the Config_SCI6 of Software Components. In the Pin Function list -> Assignment column, Ensure the ‘Enable’ tick box of SCK6 and SMOSI6 are checked and Assignment column of SCK6 is PB3, SMOSI6 is PB1 as shown in **Figure 4-50**.

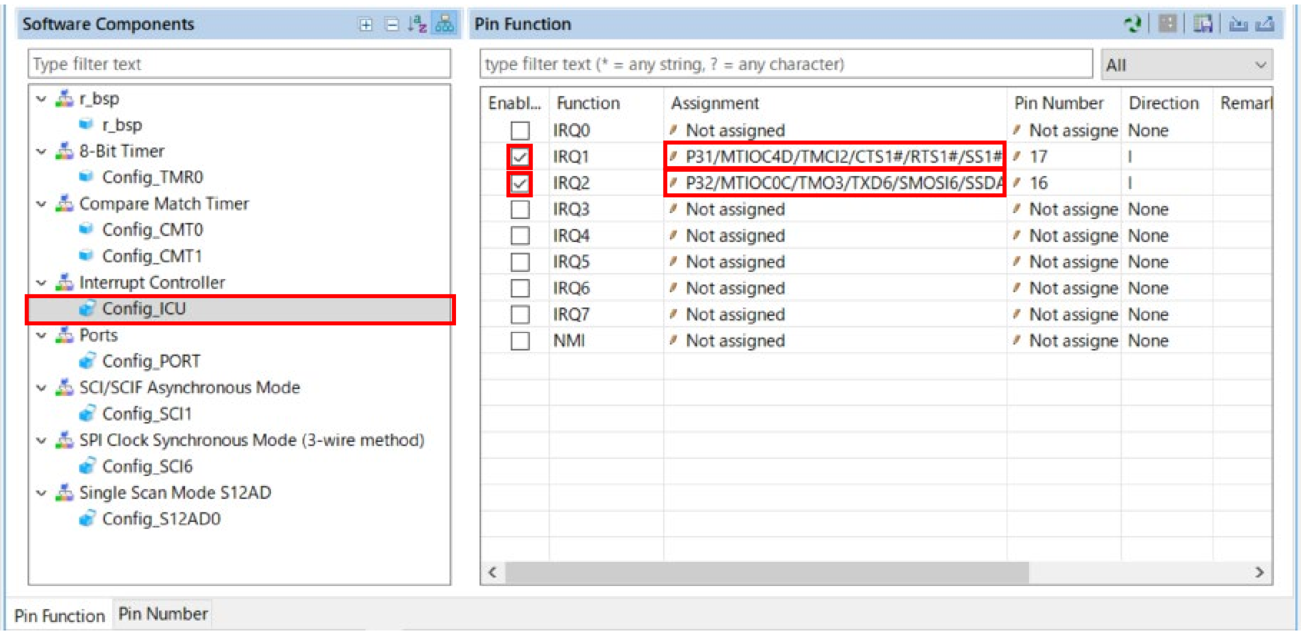


- r_bsp
  - r_bsp
- 8-Bit Timer
  - Config_TMR0
- Compare Match Timer
  - Config_CMT0
  - Config_CMT1
- Interrupt Controller
  - Config_ICU
- Ports
  - Config_PORT
- SCI/SCIF Asynchronous Mode
  - Config_SCI1
- SPI Clock Synchronous Mode (3-wire method)
  - Config_SCI6
- Single Scan Mode S12AD
  - Config_S12AD0

Enabl...	Function	Assignment	Pin Number	Direction	Remar
☑	SCK6	PB3/MTIOC0A/MTIOC4A/TMO0/POE3#/LF	45	IO	
□	SMISO6	Not assigned	Not assigne	None	
☑	SMOSI6	PB1/MTIOC0C/MTIOC4C/TMCI0/TXD6/SM	47	IO	
□	SS6#	Not assigned	Not assigne	None	

Figure 4-50 Configure pin assignment - Config_SCI6

Select the Config_ICU of Software Components. In the Pin Function list -> Assignment column, change the pin assignment IRQ1 to P31, IRQ2 to P32. Ensure the ‘Enable’ tick box of IRQ1 and IRQ2 are checked, as shown in **Figure 4-51**.



Software Components

Type filter text

- r_bsp
  - r_bsp
- 8-Bit Timer
  - Config_TMR0
- Compare Match Timer
  - Config_CMT0
  - Config_CMT1
- Interrupt Controller
  - Config_ICU
- Ports
  - Config_PORT
- SCI/SCIF Asynchronous Mode
  - Config_SCI1
- SPI Clock Synchronous Mode (3-wire method)
  - Config_SCI6
- Single Scan Mode S12AD
  - Config_S12AD0

Pin Function

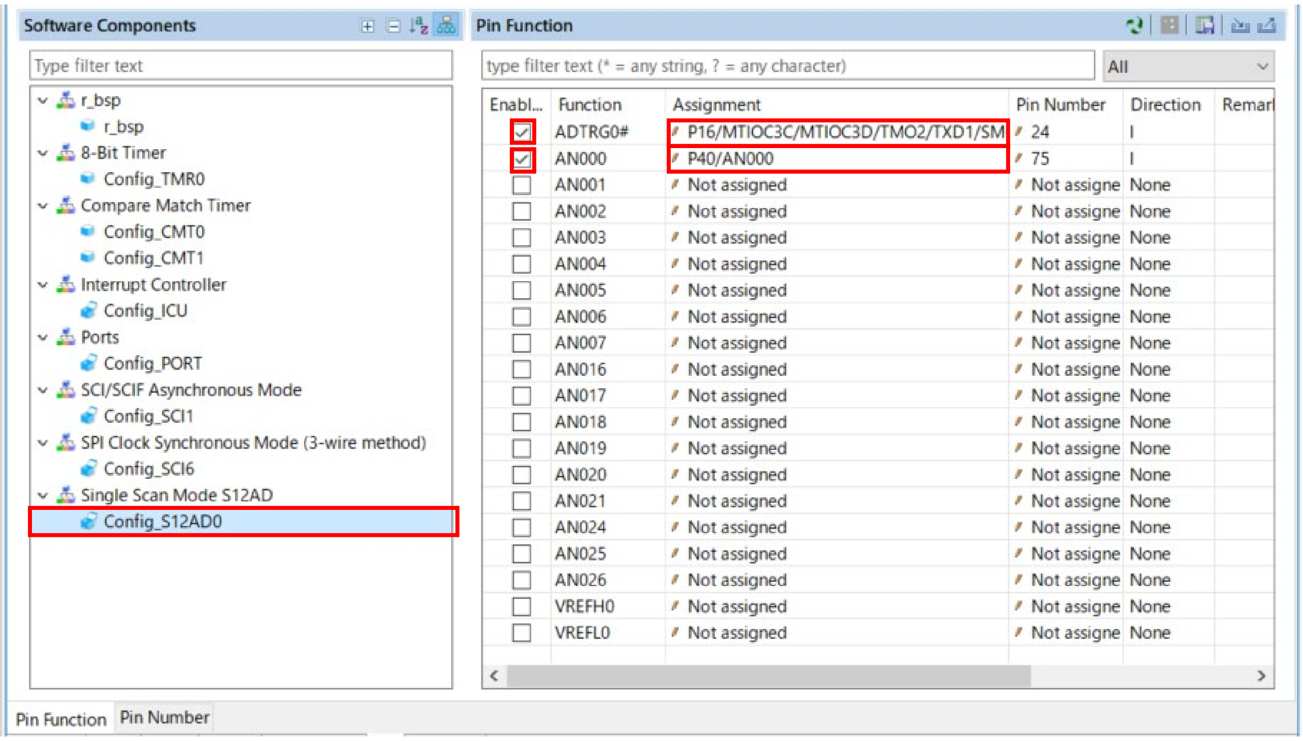
type filter text (* = any string, ? = any character) All

Enabl...	Function	Assignment	Pin Number	Direction	Remarl
□	IRQ0	Not assigned	Not assigne	None	
☑	IRQ1	P31/MTIOC4D/TMCI2/CTS1#/RTS1#/SS1#	17	I	
☑	IRQ2	P32/MTIOC0C/TMO3/TXD6/SMOSI6/SSDA	16	I	
□	IRQ3	Not assigned	Not assigne	None	
□	IRQ4	Not assigned	Not assigne	None	
□	IRQ5	Not assigned	Not assigne	None	
□	IRQ6	Not assigned	Not assigne	None	
□	IRQ7	Not assigned	Not assigne	None	
□	NMI	Not assigned	Not assigne	None	

Pin Function | Pin Number

Figure 4-51 Configure pin assignment - Config_ICU

Select the Config_S12AD0 of software components. In the Pin Function list -> Assignment column, Ensure the ‘Enable’ tick box of ADTRG0#, AN000 are checked and Assignment column of AN000 is P40, ADTRG0# is P16 as shown in **Figure 4-52**.



Software Components

Type filter text

- r_bsp
  - r_bsp
- 8-Bit Timer
  - Config_TMR0
- Compare Match Timer
  - Config_CMT0
  - Config_CMT1
- Interrupt Controller
  - Config_ICU
- Ports
  - Config_PORT
- SCI/SCIF Asynchronous Mode
  - Config_SCI1
- SPI Clock Synchronous Mode (3-wire method)
  - Config_SCI6
- Single Scan Mode S12AD
  - Config_S12AD0

Pin Function

type filter text (* = any string, ? = any character) All

Enabl...	Function	Assignment	Pin Number	Direction	Remarl
☑	ADTRG0#	P16/MTIOC3C/MTIOC3D/TMO2/TXD1/SM	24	I	
☑	AN000	P40/AN000	75	I	
□	AN001	Not assigned	Not assigne	None	
□	AN002	Not assigned	Not assigne	None	
□	AN003	Not assigned	Not assigne	None	
□	AN004	Not assigned	Not assigne	None	
□	AN005	Not assigned	Not assigne	None	
□	AN006	Not assigned	Not assigne	None	
□	AN007	Not assigned	Not assigne	None	
□	AN016	Not assigned	Not assigne	None	
□	AN017	Not assigned	Not assigne	None	
□	AN018	Not assigned	Not assigne	None	
□	AN019	Not assigned	Not assigne	None	
□	AN020	Not assigned	Not assigne	None	
□	AN021	Not assigned	Not assigne	None	
□	AN024	Not assigned	Not assigne	None	
□	AN025	Not assigned	Not assigne	None	
□	AN026	Not assigned	Not assigne	None	
□	VREFH0	Not assigned	Not assigne	None	
□	VREFL0	Not assigned	Not assigne	None	

Pin Function | Pin Number

Figure 4-52 Configure pin assignment - Config_S12AD0

Peripheral function configuration is now complete. Save the project using the File -> Save, then click ‘Generate Code’ at location of **Figure 4-53**.

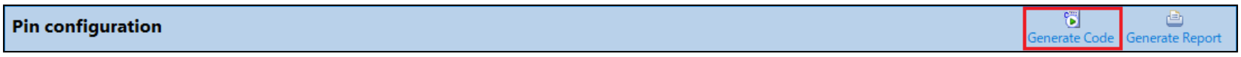
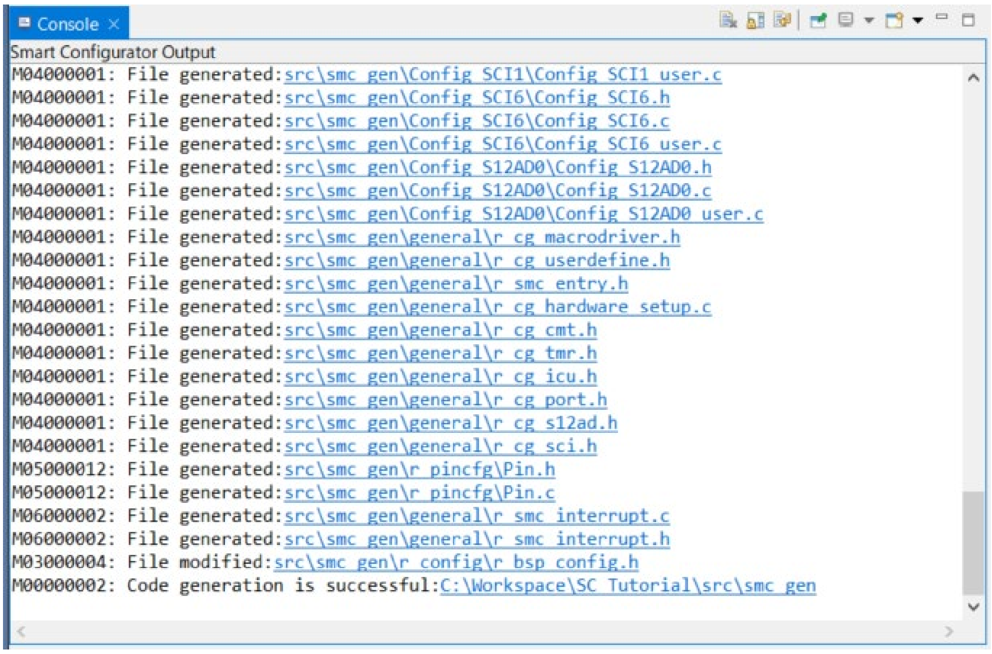


Figure 4-53 Generate Code Button

The Console pane should report ‘Code generation is successful’, as shown **Figure 4-54** below.

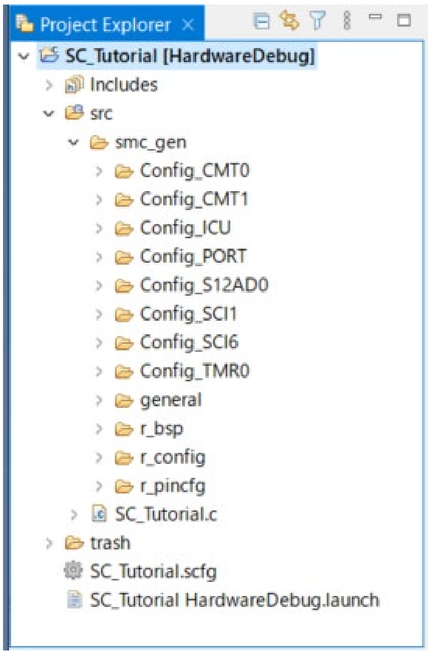


```
Smart Configurator Output
M04000001: File generated:src\smc_gen\Config_SCI1\Config_SCI1_user.c
M04000001: File generated:src\smc_gen\Config_SCI6\Config_SCI6.h
M04000001: File generated:src\smc_gen\Config_SCI6\Config_SCI6.c
M04000001: File generated:src\smc_gen\Config_SCI6\Config_SCI6_user.c
M04000001: File generated:src\smc_gen\Config_S12AD0\Config_S12AD0.h
M04000001: File generated:src\smc_gen\Config_S12AD0\Config_S12AD0.c
M04000001: File generated:src\smc_gen\Config_S12AD0\Config_S12AD0_user.c
M04000001: File generated:src\smc_gen\general\r_cg_macrodriver.h
M04000001: File generated:src\smc_gen\general\r_cg_userdefine.h
M04000001: File generated:src\smc_gen\general\r_smc_entry.h
M04000001: File generated:src\smc_gen\general\r_cg_hardware_setup.c
M04000001: File generated:src\smc_gen\general\r_cg_cmt.h
M04000001: File generated:src\smc_gen\general\r_cg_tmr.h
M04000001: File generated:src\smc_gen\general\r_cg_icu.h
M04000001: File generated:src\smc_gen\general\r_cg_port.h
M04000001: File generated:src\smc_gen\general\r_cg_s12ad.h
M04000001: File generated:src\smc_gen\general\r_cg_sci.h
M05000012: File generated:src\smc_gen\r_pincfg\Pin.h
M05000012: File generated:src\smc_gen\r_pincfg\Pin.c
M06000002: File generated:src\smc_gen\general\r_smc_interrupt.c
M06000002: File generated:src\smc_gen\general\r_smc_interrupt.h
M03000004: File modified:src\smc_gen\r_config\r_bsp_config.h
M00000002: Code generation is successful:C:\Workspace\SC_Tutorial\src\smc_gen
```

Figure 4-54 Smart Configurator console

4.8 Building the Project

The project template created by Smart Configurator can now be built. In the Project Explorer pane expand the ‘src’ folder then smc_gen folder.



```
Project Explorer
SC_Tutorial [HardwareDebug]
  Includes
  src
    smc_gen
      Config_CMT0
      Config_CMT1
      Config_ICU
      Config_PORT
      Config_S12AD0
      Config_SCI1
      Config_SCI6
      Config_TMR0
      general
      r_bsp
      r_config
      r_pincfg
    SC_Tutorial.c
  trash
  SC_Tutorial.scfg
  SC_Tutorial HardwareDebug.launch
```

Figure 4-55 Generated folder structure

Switch back to the ‘C/C++’ perspective using the button on the top right of the e^2 studio workspace. Select SC_Tutorial in the Project Explorer pane, then 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

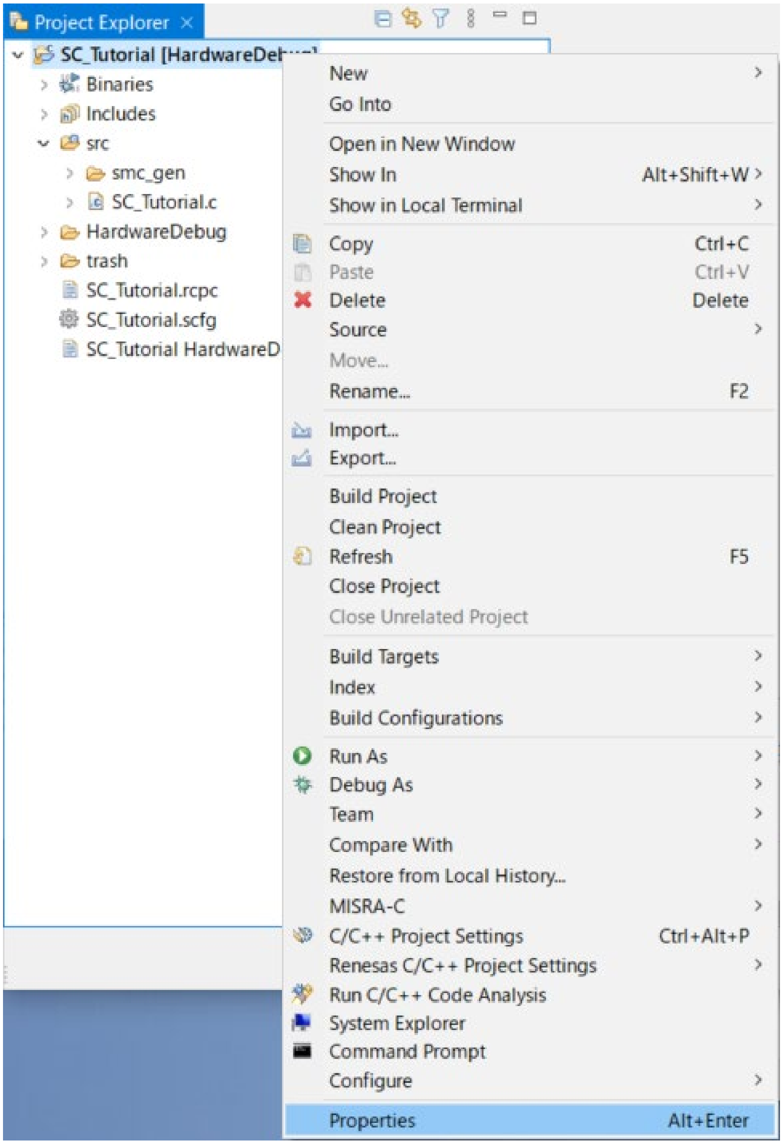
In this section, the remaining application code is added to the project. Source files found in the RSK Web Installer are copied into the workspace and the user is directed to add code in the user areas of the code generator files.

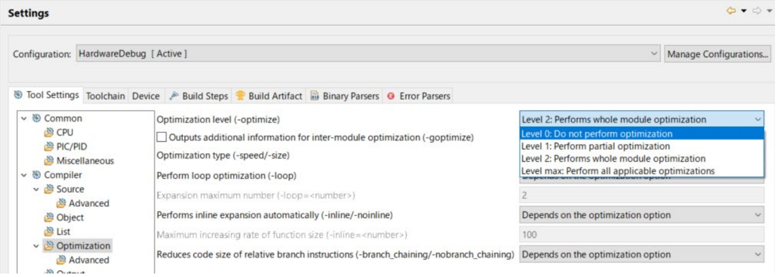
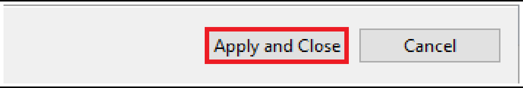
Code must be inserted into the user code area within many Smart Configurator-generated files in this project, these user code areas are delimited by comments as follows:

```
/* 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 Smart Configurator, if the user subsequently needs to use Smart Configurator to regenerate any of the Smart Configurator-generated code.

5.1 Project Settings

- Change the optimization level of the build configuration 'HardwareDebug' before building the project. With the SC_Tutorial project selected, right-click and select [Properties], or use the shortcut keys [Alt] + [Enter] to open the Properties window.	

• Navigate to ‘C/C++ Build -> Settings ->Compiler -> Optimization. • Select 'Level 0: Do not perform optimization' from the Optimization level pull-down.	 Settings Configuration: HardwareDebug [Active] Manage Configurations... Tool Settings Toolchain Device Build Steps Build Artifact Binary Parsers Error Parsers Common CPU PIC/PID Miscellaneous Compiler Source Advanced Object List Optimization Advanced Optimization level (-optimize): Level 2: Performs whole module optimization Level 0: Do not perform optimization Level 1: Perform partial optimization Level 2: Performs whole module optimization Level max: Perform all applicable optimizations □ Outputs additional information for inter-module optimization (-goptimize) Optimization type (-speed/-size) Perform loop optimization (-loop) Expansion maximum number (-loop=<number>): 2 Performs inline expansion automatically (-inline/-noinline): Depends on the optimization option Maximum increasing rate of function size (-inline=<number>): 100 Reduces code size of relative branch instructions (-branch_chaining/-nobranch_chaining): Depends on the optimization option
• Press the ‘Apply and Close’ button to close Properties window.	 Apply and Close Cancel

5.2 LCD Code Integration

API functions for the Okaya LCD display are provided with the RSK. Refer to the Tutorial project folder created according to the Quick Start Guide procedure. Check that the following files are in the src folder:

- ascii.c
- ascii.h
- r_okaya_lcd.c
- r_okaya_lcd.h

Copy these files in to the src folder below the workspace. These files will be automatically added to the project as shown in **Figure 5-1**.

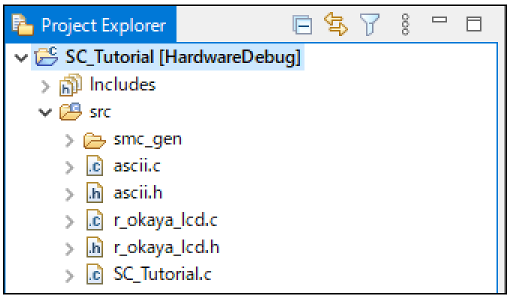


Figure 5-1 Adding files to the project

In the e^2 studio Project Tree, expand the 'src\smc_gen\general' folder and open the file 'r_cg_userdefine.h' by double-clicking on it. Insert the following #defines in between the user code delimiter comments as shown below.

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

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

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

In the same file insert the following code in the user code area for include near the top of the file:

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

In the same file insert the following code in the user code area inside the type define.

```
/* Start user code for type define. Do not edit comment generated here */
typedef char char_t;
/* End user code. Do not edit comment generated here */
```

In the e^2 studio Project Tree, expand the 'src' folder and open the file 'SC_Tutorial.c' by double-clicking on it. Add header files near the declaration '#include r_smc_entry.h'.

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_cg_userdefine.h"
```

Scroll down to the ‘main’ function and insert the highlighted code as shown below into the beginning of the ‘main’ function:

```
void main(void)
{
    /* Initialize the debug LCD */
    R_LCD_Init();

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX140 ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");
    while (1U)
    {
        ;
    }
}
```

Indentation is lost when the code described in this manual is pasted into the e^2 studio source file. Also check that the pasted code is correct.

5.2.1 SPI Code

The Okaya LCD display is driven by the SPI Master that was configured using Smart Configurator in section 4.6.7. In the e^2 studio Project Tree, expand the 'src\smc_gen\Config_SCI6' folder and open the file 'Config_SCI6.h' by double-clicking on it. Insert the following code in the user code area at the end of the file:

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

/* Exported functions used to transmit a number of bytes and wait for completion */
MD_STATUS R_SCI6_SPIMasterTransmit(uint8_t * const tx_buf, const uint16_t tx_num);

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

Now, open the Config_SCI6_user.c file and insert the following code in the user area for global:

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

/* Flag used locally to detect transmission complete */
static volatile uint8_t s_sci6_txdone;

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

Insert the following code in the transmit end call-back function for SCI6:

```
static void r_Config_SCI6_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI6_callback_transmitend. Do not edit comment generated here */

    s_sci6_txdone = TRUE;

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

Now insert the following function in the user code area at the end of the file:

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

/*******************************************************************************
* Function Name: R_SCI6_SPIMasterTransmit
* Description  : This function sends SPI6 data to slave device.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
*******************************************************************************/
MD_STATUS R_SCI6_SPIMasterTransmit (uint8_t * const tx_buf, const uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    /* Clear the flag before initiating a new transmission */
    s_sci6_txdone = FALSE;

    /* Send the data using the API */
    status = R_Config_SCI6_SPI_Master_Send(tx_buf, tx_num);

    /* Wait for the transmit end flag */
    while (FALSE == s_sci6_txdone)
    {
        /* Wait */
    }

    return (status);
}

/*******************************************************************************
* End of function R_SCI6_SPIMasterTransmit
*******************************************************************************/
```

This function uses the transmit end callback function to perform flow control on the SPI transmission to the LCD and is used as the main API call in the LCD code module.

5.2.2 TMR Code

The LCD code needs to insert delays to meet the timing requirements of the display module. This is achieved using the dedicated timer which was configured using Smart Configurator in section 4.6.2. Open the file 'src\smc_gen\Config_TMR0\Config_TMR0.h' and insert the following code in the user area for function at the end of the file:

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

void R_TMR_MsDelay(const uint16_t millisec);

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

Open the file 'Config_TMR0_user.c' and insert the following code in the user area for global at the beginning of the file:

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

static volatile uint8_t s_one_ms_delay_complete = FALSE;

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

Scroll down to the r_Config_TMR0_cmia0_interrupt function and insert the following line in the user code area:

```
static void r_Config_TMR0_cmia0_interrupt(void)
{
    /* Start user code for r_Config_TMR0_cmia0_interrupt. Do not edit comment generated here */

    s_one_ms_delay_complete = TRUE;

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

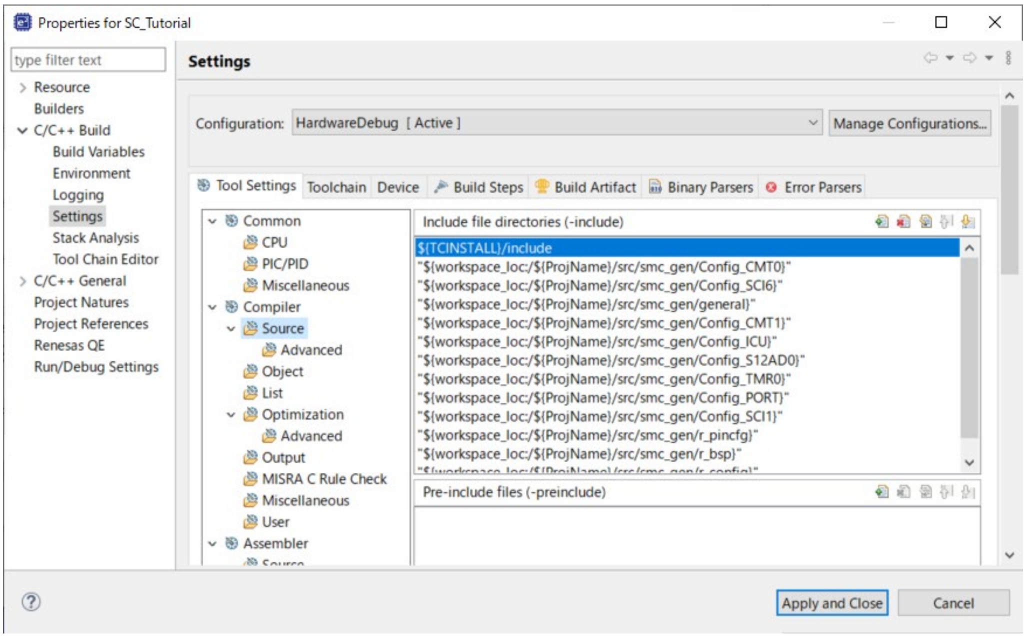
Then insert the following function in the user code area at the end of the file:

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

```
/*******************************************************************************
* Function Name: R_TMR_MsDelay
* Description  : Uses TMR0 to wait for a specified number of milliseconds
* Arguments    : uint16_t millisecs, number of milliseconds to wait
* Return Value : None
*******************************************************************************/
void R_TMR_MsDelay (const uint16_t millisec)
{
    uint16_t ms_count = 0;

    do
    {
        R_Config_TMR0_Start();
        while (FALSE == s_one_ms_delay_complete)
        {
            /* Wait */
        }
        R_Config_TMR0_Stop();
        s_one_ms_delay_complete = FALSE;
        ms_count++;
    } while (ms_count < millisec);
}
/*******************************************************************************
End of function R_TMR_MsDelay
*******************************************************************************/
```

5.3 Additional include paths

Before the project can be built the compiler needs some additional include paths added. Select the SC_Tutorial project in the Project Explorer pane. Right click in the Project Explorer window and select 'Properties'. Navigate to ‘C/C++ Build -> Settings ->Compiler -> Source and click the button as shown in **Figure 5-2**.



Configuration: HardwareDebug [Active]

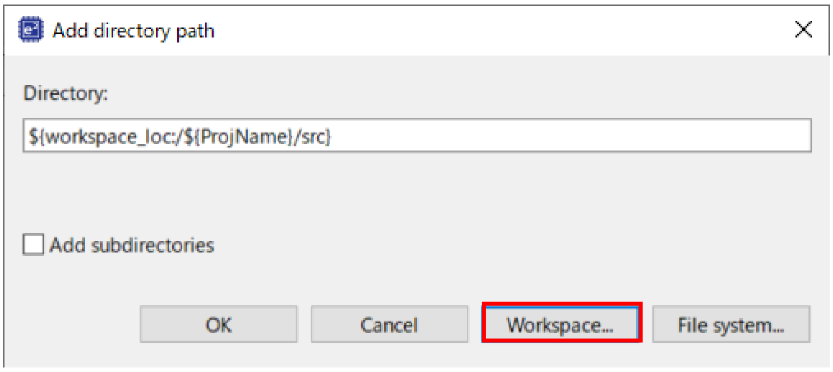
Include file directories (-include)

```
${TCINSTALL}/include
"${workspace_loc:/${ProjName}/src/smc_gen/Config_CMT0}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_SCI6}"
"${workspace_loc:/${ProjName}/src/smc_gen/general}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_CMT1}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_ICU}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_S12AD0}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_TMR0}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_PORT}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_SCI1}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_pincfg}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp}"
```

Pre-include files (-preinclude)

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 ‘SC_Tutorial/src’ folder and click ‘OK’. e^2 studio formats the path as shown in **Figure 5-3** below.



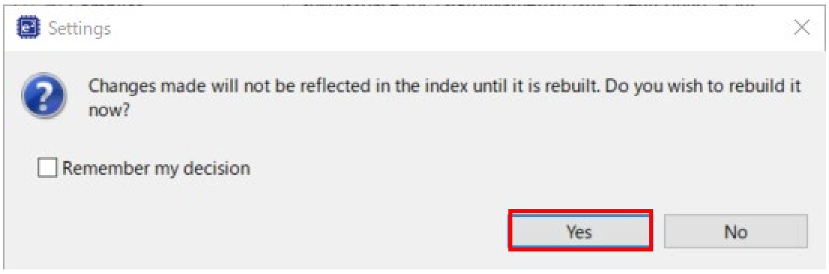
Directory:

${workspace_loc:/${ProjName}/src}

□ Add subdirectories

Figure 5-3 Adding workspace search path

Close the property by clicking the 'Apply and Close' button shown in **Figure 5-2**, and when the 'Settings' dialog shown in **Figure 5-4** is appeared, click 'Yes' to finish the setting.



Changes made will not be reflected in the index until it is rebuilt. Do you wish to rebuild it now?

□ Remember my decision

Figure 5-4 Settings 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 section 6. The program will display ‘RSKRX140 Tutorial Press Any Switch’ on three lines in the LCD display.

5.4 Switch Code Integration

API functions for user switch control are provided with the RSK. Refer to the Tutorial project folder created according to the Quick Start Guide procedure. Check that the following files are in the src folder:

- rskrx140def.h
- r_rsk_switch.c
- r_rsk_switch.h

Copy these files in to the src folder below the workspace.

The switch code uses interrupt code in the files Config_ICU.h, Config_ICU.c and Config_ICU_user.c and timer code in the files Config_CMT0.h, Config_CMT0.c, Config_CMT0_user.c, Config_CMT1.h, Config_CMT1.c and Config_CMT1_user.c as described in section 4.6.2. and section 4.6.4 It is necessary to provide additional user code in these files to implement the switch press/release detection and de-bouncing required by the API functions in r_rsk_switch.c.

5.4.1 Interrupt Code

In the e^2 studio Project Tree, expand the 'src\smc_gen\Config_ICU' folder and open the file 'Config_ICU.h' by double-clicking on it. Insert the following code in the user code area at the end of the file:

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

/* Function prototypes for detecting and setting the edge trigger of ICU_IRQ */
uint8_t R_ICU_IRQIsFallingEdge(const uint8_t irq_no);
void R_ICU_IRQSetFallingEdge(const uint8_t irq_no, const uint8_t set_f_edge);
void R_ICU_IRQSetRisingEdge(const uint8_t irq_no, const uint8_t set_r_edge);

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

Now, open the Config_ICU.c file and insert the following code in the user code area at the end of the file:

```
/* Start user code for adding. Do not edit comment generated here */
/*******************************************************************************
* Function Name: R_ICU_IRQIsFallingEdge
* Description  : This function returns 1 if the specified ICU_IRQ is set to
*                falling edge triggered, otherwise 0.
* Arguments    : uint8_t irq_no
* Return Value : 1 if falling edge triggered, 0 if not
*******************************************************************************/
uint8_t R_ICU_IRQIsFallingEdge (const uint8_t irq_no)
{
    uint8_t falling_edge_trig = 0x0;

    if (ICU.IRQCR[irq_no].BYTE & _04_ICU_IRQ_EDGE_FALLING)
    {
        falling_edge_trig = 1;
    }

    return (falling_edge_trig);

}

/*******************************************************************************
* End of function R_ICU_IRQIsFallingEdge
*******************************************************************************/

/*******************************************************************************
* Function Name: R_ICU_IRQSetFallingEdge
* Description  : This function sets/clears the falling edge trigger for the
*                specified ICU_IRQ.
* Arguments    : uint8_t irq_no
*                uint8_t set_f_edge, 1 if setting falling edge triggered, 0 if
*                clearing
* Return Value : None
*******************************************************************************/
void R_ICU_IRQSetFallingEdge (const uint8_t irq_no, const uint8_t set_f_edge)
{
    if (1 == set_f_edge)
    {
        ICU.IRQCR[irq_no].BYTE |= _04_ICU_IRQ_EDGE_FALLING;
    }
    else
    {
        ICU.IRQCR[irq_no].BYTE &= (uint8_t) ~_04_ICU_IRQ_EDGE_FALLING;
    }
}

/******************************************************************************
* End of function R_ICU_IRQSetFallingEdge
*******************************************************************************/

/*******************************************************************************
* Function Name: R_ICU_IRQSetRisingEdge
* Description  : This function sets/clear the rising edge trigger for the
*                specified ICU_IRQ.
* Arguments    : uint8_t irq_no
*                uint8_t set_r_edge, 1 if setting rising edge triggered, 0 if
*                clearing
* Return Value : None
*******************************************************************************/
void R_ICU_IRQSetRisingEdge (const uint8_t irq_no, const uint8_t set_r_edge)
{
    if (1 == set_r_edge)
    {
        ICU.IRQCR[irq_no].BYTE |= _08_ICU_IRQ_EDGE_RISING;
    }
    else
    {
        ICU.IRQCR[irq_no].BYTE &= (uint8_t) ~_08_ICU_IRQ_EDGE_RISING;
    }
}

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

Open the Config_ICU_user.c file and insert the following code in the user code area for include near the top of the file:

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

/* Defines switch callback functions required by interrupt handlers */
#include "r_rsk_switch.h"

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

In the same file insert the following code in the user code area inside the function r_Config_ICU_irq1_interrupt:

```
    /* Start user code for r_Config_ICU_irq1_interrupt. Do not edit comment generated here */

    /* Switch 1 callback handler */
    R_SWITCH_IsrCallback1();

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

In the same file insert the following code in the user code area inside the function r_Config_ICU_irq2_interrupt:

```
    /* Start user code for r_Config_ICU_irq2_interrupt. Do not edit comment generated here */

    /* Switch 2 callback handler */
    R_SWITCH_IsrCallback2();

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

5.4.2 De-bounce Timer Code

In the e^2 studio Project Tree, expand the 'src\smc_gen\Config_CMT0' folder and open the 'Config_CMT0_user.c' file and insert the following code in the user code area for include near the top of the file:

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

/* Defines switch callback functions required by interrupt handlers */
#include "r_rsk_switch.h"

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

In the Config_CMT0_user.c' file, insert the following code in the user code area inside the function r_Config_CMT0_cmi1_interrupt:

```
    /* Start user code for r_Config_CMT0_cmi0_interrupt. Do not edit comment generated here */

    /* Stop this timer - we start it again in the de-bounce routines */
    R_Config_CMT0_Stop();

    /* Call the de-bounce call back routine */
    R_SWITCH_DebounceIsrCallback();

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

In the e^2 studio Project Tree, expand the 'src\smc_gen\Config_CMT1' folder and open the file 'Config_CMT1_user.c' file and insert the following code in the user code area for include near the top of the file:

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

/* Defines switch callback functions required by interrupt handlers */
#include "r_rsk_switch.h"

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

In the same file insert the following code in the user code area inside the function r_Config_CMT1_cmi1_interrupt:

```
    /* Start user code for r_Config_CMT1_cmi1_interrupt. Do not edit comment generated here */

    /* Stop this timer - we start it again in the de-bounce routines */
    R_Config_CMT1_Stop();

    /* Call the de-bounce call back routine */
    R_SWITCH_DebounceIsrCallback();

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

5.4.3 Main Switch and ADC Code

In this part of the tutorial we add the code to act on the switch presses to activate A/D conversions and display the result on the LCD. In section 4.6.8 we configured the ADC to be triggered from the ADTRG0# pin, SW3. In this code, we also perform software triggered A/D conversion from the user switches SW1 and SW2, by reconfiguring the ADC trigger source on-the-fly once an SW1 or SW2 press is detected.

In the e2 studio Project Tree, expand the ‘src\smc_gen\general’ folder and open the file ‘r_cg_userdefine.h’. Insert the following code the user code area, resulting in the code shown below:

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

extern volatile uint8_t g_adc_trigger;

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

In the e2 studio Project Tree, expand the ‘src’ folder and Open the file ‘SC_Tutorial.c’ and add the highlighted code, resulting in the code shown below:

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_cg_userdefine.h"
#include "Config_S12AD0.h"
#include "r_rsk_switch.h"

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

/* Prototype declaration for cb_switch_press */
static void cb_switch_press (void);

/* Prototype declaration for get_adc */
static uint16_t get_adc(void);

/* Prototype declaration for lcd_display_adc */
static void lcd_display_adc (const uint16_t adc_result);
```

Next add the highlighted code below in the main function and the code inside the while loop, resulting in the code shown below:

```
void main(void)
{
    /* Initialize the switch module */
    R_SWITCH_Init();

    /* Set the call back function when SW1 or SW2 is pressed */
    R_SWITCH_SetPressCallback(cb_switch_press);

    /* Initialize the debug LCD */
    R_LCD_Init ();

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX140 ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");

    /* Start the A/D converter */
    R_Config_S12AD0_Start();

    while (1U)
    {
        uint16_t adc_result;

        /* Wait for user requested A/D conversion flag to be set (SW1 or SW2) */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an A/D conversion */
            adc_result = get_adc();

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

            /* Reset the flag */
            g_adc_trigger = FALSE;
        }
        /* SW3 is directly wired into the ADTRG0n pin so will
           cause the interrupt to fire */
        else if (TRUE == g_adc_complete)
        {
            /* Get the result of the A/D conversion */
            R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
}
```

Then add the definition for the switch call-back, get_adc and lcd_display_adc functions below the main function, as shown below:

```
/*******************************************************************************
* Function Name : cb_switch_press
* Description   : Switch press callback function. Sets g_adc_trigger flag.
* Argument      : none
* Return value  : none
*******************************************************************************/
static void cb_switch_press (void)
{
    /* Check if switch 1 or 2 was pressed */
    if (g_switch_flag & (SWITCHPRESS_1 | SWITCHPRESS_2))
    {

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

        /* Clear flag */
        g_switch_flag = 0x0;
    }
}
/*******************************************************************************
* End of function cb_switch_press
*******************************************************************************/

/*******************************************************************************
* Function Name : get_adc
* Description   : Reads the ADC result, converts it to a string and displays
*                 it on the LCD panel.
* Argument      : none
* Return value  : uint16_t adc value
*******************************************************************************/
static uint16_t get_adc (void)
{
    /* A variable to retrieve the adc result */
    uint16_t adc_result;

    /* Stop the A/D converter being triggered from the pin ADTRG0n */
    R_Config_S12AD0_Stop();

    /* Start a conversion */
    R_S12AD0_SWTriggerStart();

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

    /* Stop conversion */
    R_S12AD0_SWTriggerStop();

    /* Clear ADC flag */
    g_adc_complete = FALSE;

    R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

    /* Set AD conversion start trigger source back to ADTRG0n pin */
    R_Config_S12AD0_Start();


    return (adc_result);
}
/*******************************************************************************
* End of function get_adc
*******************************************************************************/
```

```
/******************************************************************************
* Function Name : lcd_display_adc
* Description   : Converts adc result to a string and displays
*                 it on the LCD panel.
* Argument      : uint16_t adc result
* Return value  : none
******************************************************************************/
static void lcd_display_adc (const uint16_t adc_result)
{
    /* Declare a temporary variable */
    char_t tmp;

    /* Declare temporary character string */
    char_t lcd_buffer[11] = " ADC: XXXH";

    /* Convert ADC result into a character string, and store in the local.
       Casting to ensure use of correct data type. */
    tmp           = (char_t)((adc_result & 0x0F00) >> 8);
    lcd_buffer[6] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp           = (char_t)((adc_result & 0x00F0) >> 4);
    lcd_buffer[7] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp           = (char_t)(adc_result & 0x000F);
    lcd_buffer[8] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);

    /* Display the contents of the local string lcd_buffer */
    R_LCD_Display(3, (uint8_t *)lcd_buffer);
}

/******************************************************************************
* End of function lcd_display_adc
******************************************************************************/
```

In the e^2 studio Project Tree, expand the 'src\smc_gen\Config_S12AD0' folder and open the file 'Config_S12AD0.h' by double-clicking on it. Insert the following code in the user code area for function, resulting in the code shown below:

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

/* Flag indicates when A/D conversion is complete */
extern volatile uint8_t g_adc_complete;

/* Functions for starting and stopping software triggered A/D conversion */
void R_S12AD0_SWTriggerStart(void);
void R_S12AD0_SWTriggerStop(void);

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

Open the file ‘Config_S12AD0.c’ by double-clicking on it. Insert the following code in the user code area for adding at the end of the file, as shown below:

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

/*******************************************************************************
* Function Name: R_S12AD0_SWTriggerStart
* Description  : This function starts the AD0 converter.
* Arguments    : None
* Return Value : None
*******************************************************************************/
void R_S12AD0_SWTriggerStart(void)
{
    IR(S12AD, S12ADI0) = 0U;
    IEN(S12AD, S12ADI0) = 1U;
    S12AD.ADCSR.BIT.ADST = 1U;
}

/*******************************************************************************
End of function R_S12AD0_SWTriggerStart
*******************************************************************************/

/*******************************************************************************
* Function Name: R_S12AD0_SWTriggerStop
* Description  : This function stops the AD0 converter.
* Arguments    : None
* Return Value : None
*******************************************************************************/
void R_S12AD0_SWTriggerStop(void)
{
    S12AD.ADCSR.BIT.ADST = 0U;
    IEN(S12AD, S12ADI0) = 0U;
    IR(S12AD, S12ADI0) = 0U;
}

/*******************************************************************************
End of function R_S12AD0_SWTriggerStop
*******************************************************************************/

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

Open the file Config_S12AD0_user.c and insert the following code in the user code area for global, resulting in the code shown below:

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

/* Flag indicates when A/D conversion is complete */
volatile uint8_t g_adc_complete;

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

Insert the following code in the user code area of the r_Config_S12AD0_interrupt function, resulting in the code shown below:

```
static void r_Config_S12AD0_interrupt(void)
{
    /* Start user code for r_Config_S12AD0_interrupt. Do not edit comment generated here */

    g_adc_complete = TRUE;

    /* 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 section 6. When any switch is pressed, the program will perform an A/D conversion of the voltage level on the RV1 potentiometer line and display the result on the LCD panel. Return to this point in the Tutorial to add the UART user code.

5.5 Debug Code Integration

API functions for trace debugging via the RSK serial port are provided with the RSK. Refer to the Tutorial project folder created according to the Quick Start Guide procedure. Check that the following files are in the src folder:

- r_rsk_debug.c
- r_rsk_debug.h

Copy these files in to the src folder below the workspace.

In the r_rsk_debug.h file, ensure the following macro definition is included:

```
/* Macro for definition of serial debug transmit function - user edits this */
#define SERIAL_DEBUG_WRITE (R_SCI1_AsyncTransmit)
```

This macro is referenced in the r_rsk_debug.c file and allows easy re-direction of debug output if a different debug interface is used.

5.6 UART Code Integration

5.6.1 SCI Code

In the e^{2} studio Project Tree, expand the ‘src\smc_gen\Config_SCI1’ folder and open the file ‘Config_SCI1.h’ by double-clicking on it. Insert the following code in the user code area at the end of the file:

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

/* Exported functions used to transmit a number of bytes and wait for completion */
MD_STATUS R_SCI1_AsyncTransmit(uint8_t * const tx_buf, const uint16_t tx_num);

/* Character is used to receive key presses from PC terminal */
extern uint8_t g_rx_char;

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

Open the file ‘Config_SCI1_user.c’. Insert the following code in the user area for global near the beginning of the file:

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

/* Global used to receive a character from the PC terminal */
uint8_t g_rx_char;

/* Flag used locally to detect transmission complete */
static volatile uint8_t s_sci1_txdone;

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

In the same file, insert the following code in the user code area inside the r_Config_SCI1_callback_transmitend function:

```
static void r_Config_SCI1_callback_transmitend (void)
{
    /* Start user code for r_Config_SCI1_callback_transmitend. Do not edit comment generated here */

    s_sci1_txdone = TRUE;

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

In the same file, insert the following code in the user code area inside the r_Config_SCI1_callback_receiveend function:

```
static void r_Config_SCI1_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI1_callback_receiveend. Do not edit comment generated here */

    /* Check the contents of g_rx_char */
    if (('c' == g_rx_char) || ('C' == g_rx_char))
    {
        g_adc_trigger = TRUE;
    }

    /* Set up SCI1 receive buffer and callback function again */
    R_Config_SCI1_Serial_Receive((uint8_t *)&g_rx_char, 1);

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

At the end of the file, in the user code area for adding, add the following function definition:

```
/*******************************************************************************
* Function Name: R_SCI1_AsyncTransmit
* Description  : This function sends SCI1 data and waits for the transmit end flag.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
*******************************************************************************/
MD_STATUS R_SCI1_AsyncTransmit(uint8_t * const tx_buf, const uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    /* Clear the flag before initiating a new transmission */
    s_sci1_txdone = FALSE;

    /* Send the data using the API */
    status = R_Config_SCI1_Serial_Send(tx_buf, tx_num);

    /* Wait for the transmit end flag */
    while (FALSE == s_sci1_txdone)
    {
        /* Wait */
    }
    return (status);
}

/*******************************************************************************
* End of function R_SCI1_AsyncTransmit
*******************************************************************************/
```

5.6.2 Main UART code

Open the file ‘SC_Tutorial.c’. Add the following declaration to near the top of the file:

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_cg_userdefine.h"
#include "Config_S12AD0.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "Config_SCI1.h"

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

/* Prototype declaration for cb_switch_press */
static void cb_switch_press (void);

/* Prototype declaration for get_adc */
static uint16_t get_adc(void);
```

```
/* Prototype declaration for lcd_display_adc */
static void lcd_display_adc (const uint16_t adc_result);

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

/* Variable to store the A/D conversion count for user display */
static uint8_t s_adc_count = 0;
```

Add the following highlighted code in the main function:

```
void main(void)
{
    /* Initialize the switch module */
    R_SWITCH_Init();

    /* Set the call back function when SW1 or SW2 is pressed */
    R_SWITCH_SetPressCallback(cb_switch_press);

    /* Initialize the debug LCD */
    R_LCD_Init();

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX140 ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");

    /* Start the A/D converter */
    R_Config_S12AD0_Start();

    /* Set up SCI1 receive buffer and callback function */
    R_Config_SCI1_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI1 operations */
    R_Config_SCI1_Start();

    while (1U)
    {
        uint16_t adc_result;

        /* Wait for user requested A/D conversion flag to be set (SW1 or SW2) */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an A/D conversion */
            adc_result = get_adc();

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

            /* Increment the s_adc_count */
            if (16 == (++s_adc_count))
            {
                s_adc_count = 0;
            }

            /* Send the result to the UART */
            uart_display_adc(s_adc_count, adc_result);

            /* Reset the flag */
            g_adc_trigger = FALSE;
        }
        /* SW3 is directly wired into the ADTRG0n pin so will
           cause the interrupt to fire */
        else if (TRUE == g_adc_complete)
        {
            /* Get the result of the A/D conversion */
            R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

            /* Increment the s_adc_count */
            if (16 == (++s_adc_count))
            {
                s_adc_count = 0;
            }

            /* Send the result to the UART */
            uart_display_adc(s_adc_count, adc_result);
            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
}
```

Then, add the following function definition in the end of the file:

```
/******************************************************************************
* Function Name : uart_display_adc
* Description   : Converts adc result to a string and sends it to the UART.
* Argument      : uint8_t : adc_count
*                 uint16_t: adc result
* Return value  : none
******************************************************************************/
static void uart_display_adc (const uint8_t adc_count, const uint16_t adc_result)
{
    /* Declare a temporary variable */
    char_t tmp;

    /* Declare temporary character string */
    char_t uart_buffer[] = "ADC xH Value: xxxH\r\n";

    /* Convert ADC result into a character string, and store in the local.
       Casting to ensure use of correct data type. */
    tmp            = (char_t)(adc_count & 0x000F);
    uart_buffer[4] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp            = (char_t)((adc_result & 0x0F00) >> 8);
    uart_buffer[14] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp             = (char_t)((adc_result & 0x00F0) >> 4);
    uart_buffer[15] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp             = (char_t)(adc_result & 0x000F);
    uart_buffer[16] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);

    /* Send the string to the UART */
    r_debug_print(uart_buffer);
}

/******************************************************************************
* End of function uart_display_adc
******************************************************************************/
```

Select ‘Build Project’ from the ‘Project’ menu. e^2 studio will build the project with no errors.

The project may now be run using the debugger as described in section 6. Connect the RSK G1CUSB0 port to a USB port on a PC. If this is the first time the RSK has been connected to the PC then a device driver will be installed automatically. Open Device Manager, the virtual COM port will be appeared under 'Port (COM & LPT)' as 'RSK USB Serial Port (COMx)', where x is a number.

Open a terminal program, such as HyperTerminal, on the PC with the same settings as for SCI1 (Baudrate: 19200, Data Length: 8, Parity Bit: None, Stop Bit: 1, Flow Control: None).
When any switch is pressed, or when ‘c’ is sent via the COM port, the program will perform an A/D conversion of the voltage level on the RV1 potentiometer line and display the result on the LCD panel and send the result to the PC terminal program via the SCI1.

5.7 LED Code Integration

Open the file 'SC_Tutorial.c'. Add the following declaration to the near the top of the file:

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_cg_userdefine.h"
#include "Config_S12AD0.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "Config_SCI1.h"
#include "rskrx140def.h"


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

/* Prototype declaration for cb_switch_press */
static void cb_switch_press (void);

/* Prototype declaration for get_adc */
static uint16_t get_adc(void);

/* Prototype declaration for lcd_display_adc */
static void lcd_display_adc (const uint16_t adc_result);

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

/* Variable to store the A/D conversion count for user display */
static uint8_t s_adc_count = 0;

/* Prototype declaration for led_display_count */
static void led_display_count(const uint8_t count);
```

Add the following highlighted code in the main function:

```
void main(void)
{
    /* Initialize the switch module */
    R_SWITCH_Init();

    /* Set the call back function when SW1 or SW2 is pressed */
    R_SWITCH_SetPressCallback(cb_switch_press);

    /* Initialize the debug LCD */
    R_LCD_Init();

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX140 ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");

    /* Start the A/D converter */
    R_Config_S12AD0_Start();

    /* Set up SCI1 receive buffer and callback function */
    R_Config_SCI1_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI1 operations */
    R_Config_SCI1_Start();

    while (1U)
    {
        uint16_t adc_result;

        /* Wait for user requested A/D conversion flag to be set (SW1 or SW2) */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an A/D conversion */
            adc_result = get_adc();

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);
            /* Increment the s_adc_count and display using the LEDs */
            if (16 == (++s_adc_count))
            {
                s_adc_count = 0;
            }
            led_display_count(s_adc_count);

            /* Send the result to the UART */
            uart_display_adc(s_adc_count, adc_result);
            /* Reset the flag */
            g_adc_trigger = FALSE;
        }
        /* SW3 is directly wired into the ADTRG0n pin so will
           cause the interrupt to fire */
        else if (TRUE == g_adc_complete)
        {
            /* Get the result of the A/D conversion */
            R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

            /* Increment the s_adc_count and display using the LEDs */
            if (16 == (++s_adc_count))
            {
                s_adc_count = 0;
            }
            led_display_count(s_adc_count);

            /* Send the result to the UART */
            uart_display_adc(s_adc_count, adc_result);
            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
}
```

Then, add the following function definition at the end of the file:

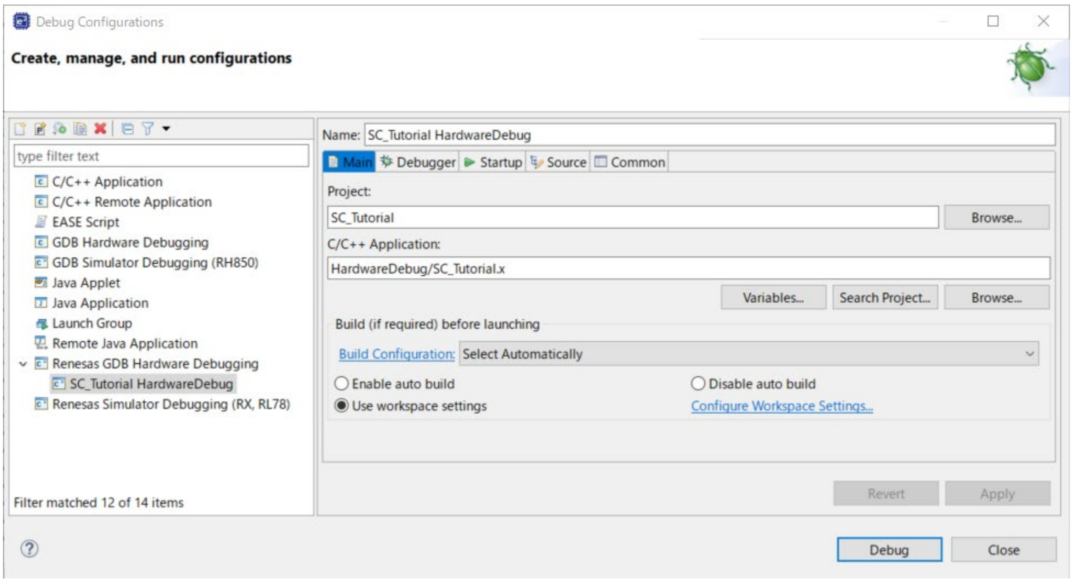
```
/*******************************************************************************
* Function Name : led_display_count
* Description   : Converts count to binary and displays on 4 LEDS0-3
* Argument      : uint8_t count
* Return value  : none
*******************************************************************************/
static void led_display_count (const uint8_t count)
{
    /* Set LEDs according to lower nibble of count parameter */
    LED0 = (uint8_t)((count & 0x01) ? LED_ON : LED_OFF);
    LED1 = (uint8_t)((count & 0x02) ? LED_ON : LED_OFF);
    LED2 = (uint8_t)((count & 0x04) ? LED_ON : LED_OFF);
    LED3 = (uint8_t)((count & 0x08) ? LED_ON : LED_OFF);
}
/*******************************************************************************
* End of function led_display_count
*******************************************************************************/
```

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 section 6. The code will perform the same but now the LEDs will display the s_adc_count in binary form.

6. Debugging the Project

In the Project Explorer pane, ensure that the ‘SC_Tutorial’ project is selected. To enter the configurations, click upon the arrow next to the debug button ▾ and select ‘Debug Configuration’.



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB Simulator Debugging (RH850)
- Java Applet
- Java Application
- Launch Group
- Remote Java Application
- Renesas GDB Hardware Debugging
  - SC_Tutorial HardwareDebug
- Renesas Simulator Debugging (RX, RL78)

Filter matched 12 of 14 items

Name: SC_Tutorial HardwareDebug

Main | Debugger | Startup | Source | Common

Project: SC_Tutorial [Browse...]

C/C++ Application: HardwareDebug/SC_Tutorial.x [Variables...] [Search Project...] [Browse...]

Build (if required) before launching

Build Configuration: Select Automatically

○ Enable auto build ○ Disable auto build

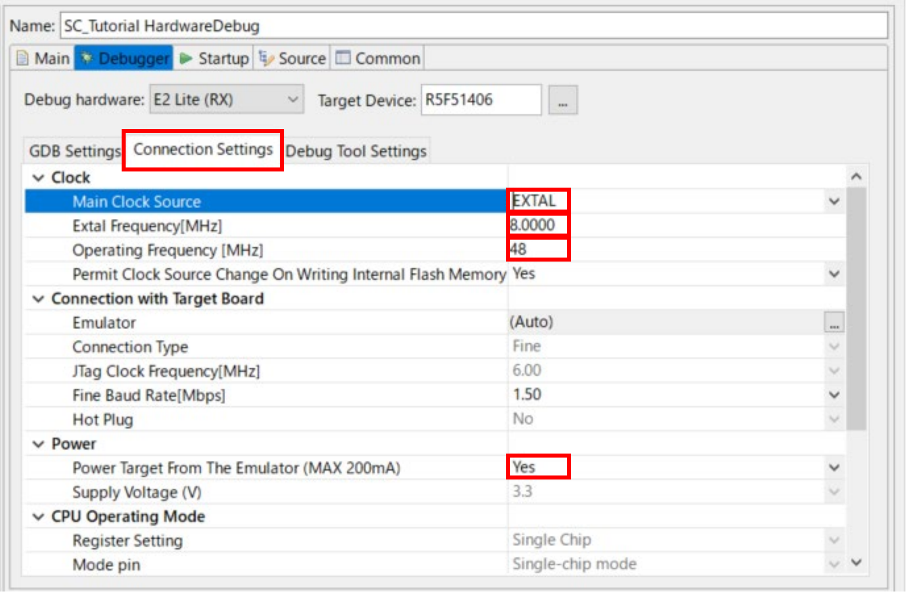
● Use workspace settings Configure Workspace Settings...

[Revert] [Apply]

[Debug] [Close]

Figure 6-1 Debug Configurations

In order to execute the project, it is necessary to change the following settings in ‘Renesas GDB Hardware Debugging’ -> ‘SC_Tutorial HardwareDebug’ -> ‘Debugger’ -> ‘Connection Settings’.
Set ‘Power Target From The Emulator (MAX 200mA)’ to ‘Yes’, set ‘Extal Frequency [MHz]’ and 'Operating Frequency [MHz]' to the correct frequency. (They should not use the 'Enter' key after typing in values.)
These can be found from the device schematics (in the case of RSKRX140 set the EXTAL Frequency: 8.0000, Operating Frequency: 48.000).
For more information on powering the RSKRX140 please refer to the User’s Manual.



Name: SC_Tutorial HardwareDebug

Main | Debugger | Startup | Source | Common

Debug hardware: E2 Lite (RX) Target Device: R5F51406

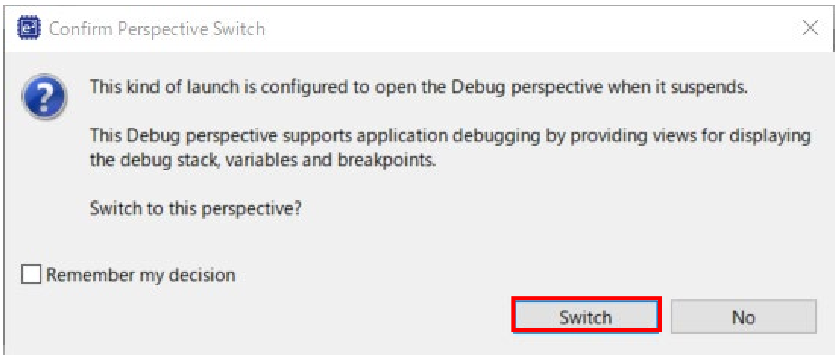
GDB Settings | Connection Settings | Debug Tool Settings

Clock	
Main Clock Source	EXTAL
Extal Frequency[MHz]	8.0000
Operating Frequency [MHz]	48
Permit Clock Source Change On Writing Internal Flash Memory	Yes
Connection with Target Board	
Emulator	(Auto)
Connection Type	Fine
JTag Clock Frequency[MHz]	6.00
Fine Baud Rate[Mbps]	1.50
Hot Plug	No
Power	
Power Target From The Emulator (MAX 200mA)	Yes
Supply Voltage (V)	3.3
CPU Operating Mode	
Register Setting	Single Chip
Mode pin	Single-chip mode

Figure 6-2 Connection Settings

When the setting is complete, press the 'Apply' button followed by the "Close" button to close the debug configuration window.

Connect the E2 Lite to the PC and the RSK E2 Lite connector. Connect the Pmod LCD to the PMOD1 connector. Connect the center positive +5V PSU to the PWR connector on the RSK and apply power.
In the Project Explorer pane, ensure that the 'SC_Tutorial' project is selected. To debug the project, click the button. The dialog shown in **Figure 6-3** will be displayed.


Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

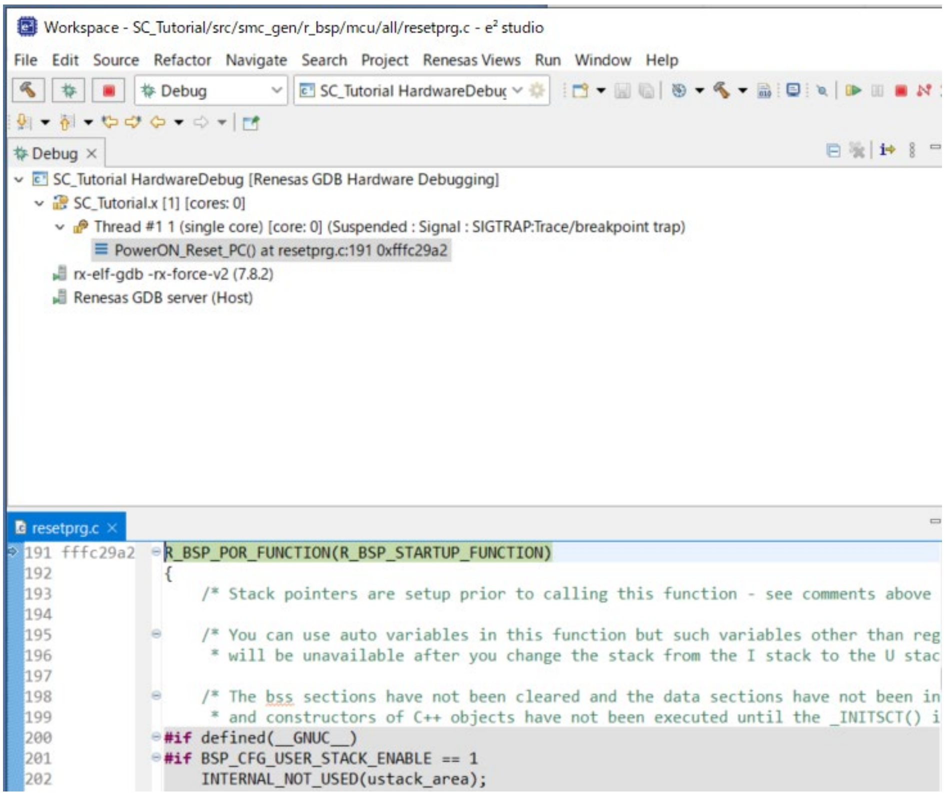
This Debug perspective supports application debugging by providing views for displaying the debug stack, variables and breakpoints.

Switch to this perspective?

□ Remember my decision

Figure 6-3 Perspective Switch Dialog

Click 'Remember my decision' to skip this dialog later. Click 'Switch' to confirm that the debug window perspective will be used. The debugger will start up and the code will stop at the Smart Configurator function 'PowerOn_Reset_PC' as shown in **Figure 6-4**.


Workspace - SC_Tutorial/src/smc_gen/r_bsp/mcu/all/resetprg.c - e² studio

File Edit Source Refactor Navigate Search Project Renesas Views Run Window Help
Debug
SC_Tutorial HardwareDebug
Debug
resetprg.c

SC_Tutorial HardwareDebug [Renesas GDB Hardware Debugging]
SC_Tutorial.x [1] [cores: 0]
Thread #1 1 (single core) [core: 0] (Suspended : Signal : SIGTRAP:Trace/breakpoint trap)
PowerON_Reset_PC() at resetprg.c:191 0xfffc29a2
rx-elf-gdb -rx-force-v2 (7.8.2)
Renesas GDB server (Host)

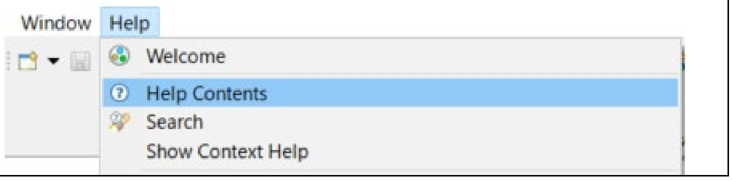
```
fffc29a2  R_BSP_POR_FUNCTION(R_BSP_STARTUP_FUNCTION)
          {
              /* Stack pointers are setup prior to calling this function - see comments above

              /* You can use auto variables in this function but such variables other than reg
               * will be unavailable after you change the stack from the I stack to the U stac

              /* The bss sections have not been cleared and the data sections have not been in
               * and constructors of C++ objects have not been executed until the _INITSCT() i
          #if defined(__GNUC__)
          #if BSP_CFG_USER_STACK_ENABLE == 1
              INTERNAL_NOT_USED(ustack_area);
```

Figure 6-4 Debugger start up screen

For more information on the e^2 studio debugger refer to the Tutorial manual. To run the code click the ▷ button. The debugger will stop again at the beginning of the main function. Press ▷ again to run the code.

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.	 Window Help Welcome Help Contents Search Show Context Help

For information about the RX140 group microcontroller refer to ‘RX140 Group User’s Manual: Hardware’.

For information about the RX assembly language, refer to ‘RX Family User’s Manual: Software’.

Technical Contact Details

America: techsupport.america@renesas.com
Europe: https://www.renesas.com/eu/en/support/contact.html
Global & Japan: https://www.renesas.com/support/contact.html

General information on this product can be found on the Renesas website at:
https://www.renesas.com/rskrx140

General information on Renesas microcontrollers can be found on the Renesas website at:
https://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 GmbH.

REVISION HISTORY	RX140 Group Renesas Starter Kit for RX140 Smart Configurator Tutorial Manual For e^2 studio

Rev.	Date	Description	
		Page	Summary
1.00	Jan.17.2022	—	First Edition issued

RX140 Group

Renesas Starter Kit for RX140

Smart Configurator Tutorial Manual For e^2 studio

Publication Date: Rev. 1.00 Jan.17.2022

Published by: Renesas Electronics Corporation

RX140 Group

Renesas Electronics Corporation

R20UT5032EG0100