User's Manual

32

RX660 Group

Renesas Starter Kit for RX660
Smart Configurator Tutorial Manual
For e^2 studio

RENESAS 32-Bit MCU
RX Family / RX600 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

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/.

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:

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 RX660 microcontroller may be found in 'RX660 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 RX660 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 RX660 User's Manual	R20UT5017EG
Tutorial Manual	Provides a guide to setting up RSK environment, running sample code and debugging programs.	Renesas Starter Kit for RX660 Tutorial Manual	R20UT5021EG
Quick Start Guide	Provides simple instructions to setup the RSK and run the first sample.	Renesas Starter Kit for RX660 Quick Start Guide	R20UT5022EG
Smart Configurator Tutorial	Provides a guide to code generation and importing into the e^2 studio IDE.	Renesas Starter Kit for RX660 Smart Configurator Tutorial Manual	R20UT5023EG
Schematics	Full detail circuit schematics of the RSK.	Renesas Starter Kit for RX660 Schematics	R20UT5016EG
Hardware Manual	Provides technical details of the RX660 microcontroller.	RX660 Group User's Manual: Hardware	R01UH0937EJ

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

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.

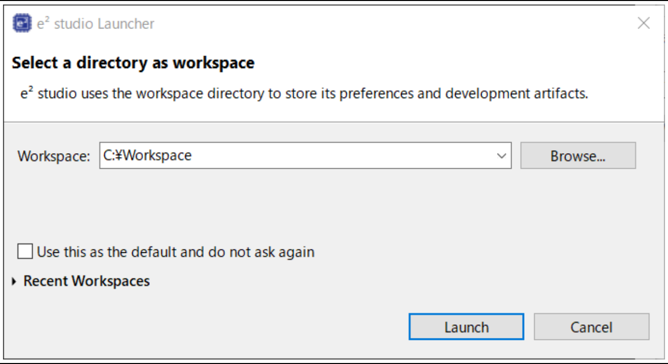
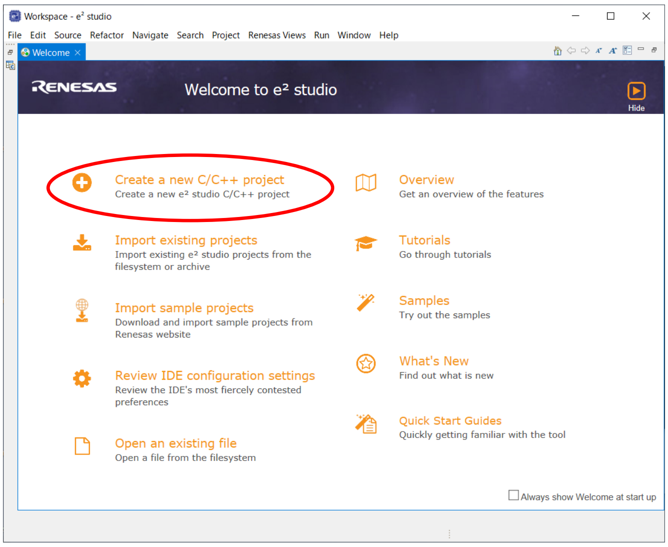
> These tutorials are designed to show you how to use the RSK and are not intended as a comprehensive introduction to the e^2 studio debugger, compiler toolchains or the E2 emulator Lite. Please refer to the relevant user manuals for more in-depth information.

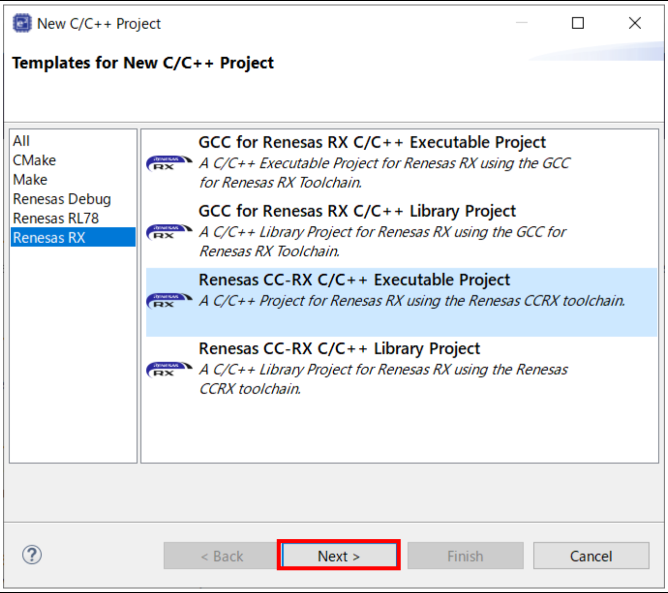
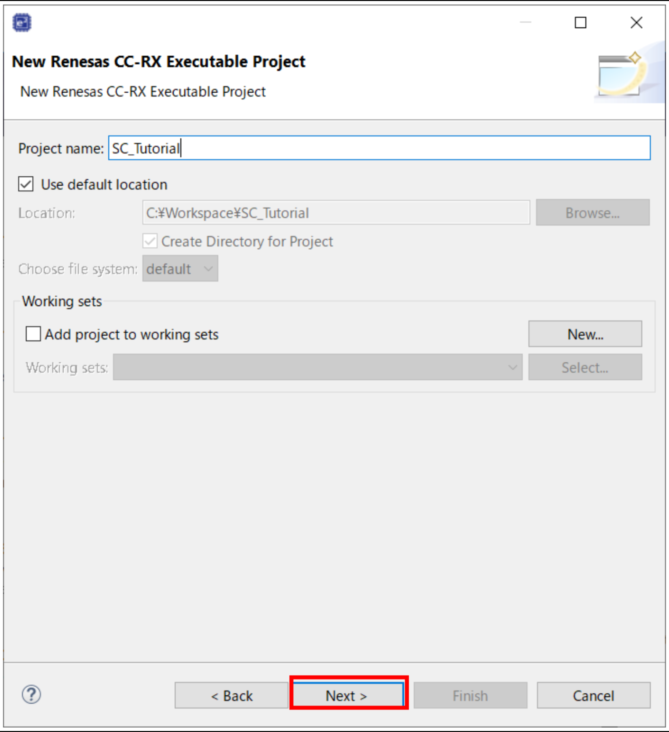
3. Project Creation with e² studio

3.1 Introduction

In this section, the user will be guided through the steps required to create a new C project for the RX660 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.

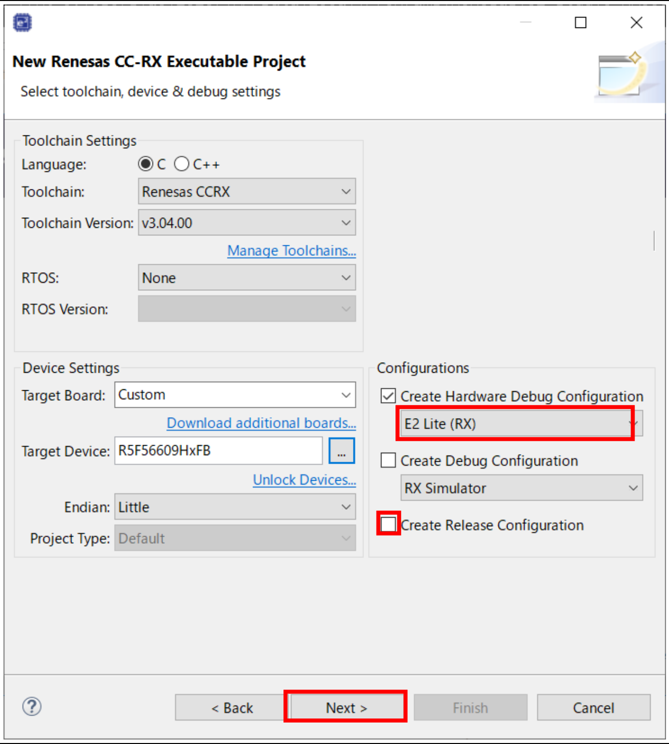
3.2 Creating the Project

• Start e² studio and select a suitable location for the project workspace.	
• In the Welcome page, click 'Create a new C/C++ project'. (The Welcome page can also be opened from 'Help'-> 'Welcome'.)	

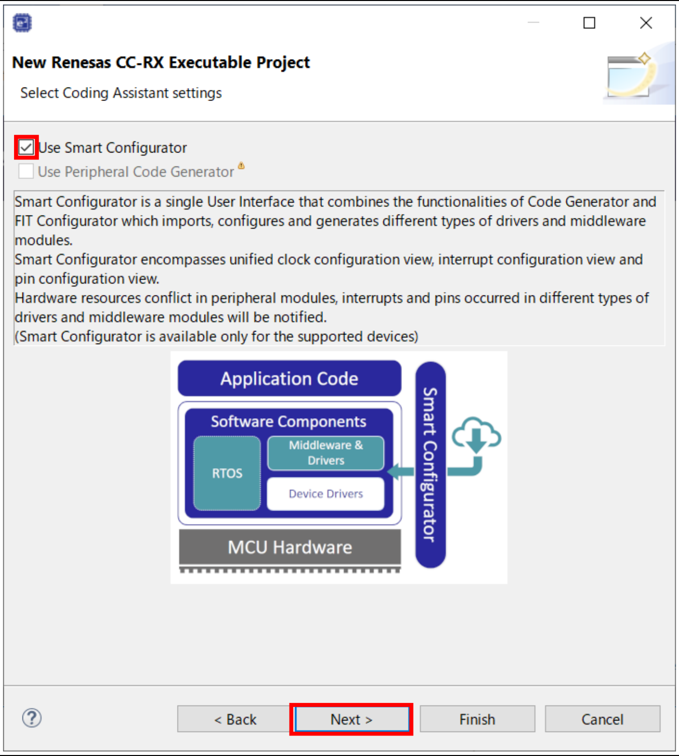
• In the ‘Templates for New C/C++ Project’ dialog, selecting ‘Renesas RX’ -> ‘Renesas CC-RX C/C++ Executable Project’. • Click ‘Next’.	
• Enter the project name ‘SC_Tutorial’. Click ‘Next’.	

- In the ‘Select toolchain, device & debug settings’ dialog, select the options as shown in the screenshot opposite.
- In ‘Toolchains’ choose ‘Renesas CCRX’.
- The R5F56609HxFB MCU is found under RX600 -> RX660 ->

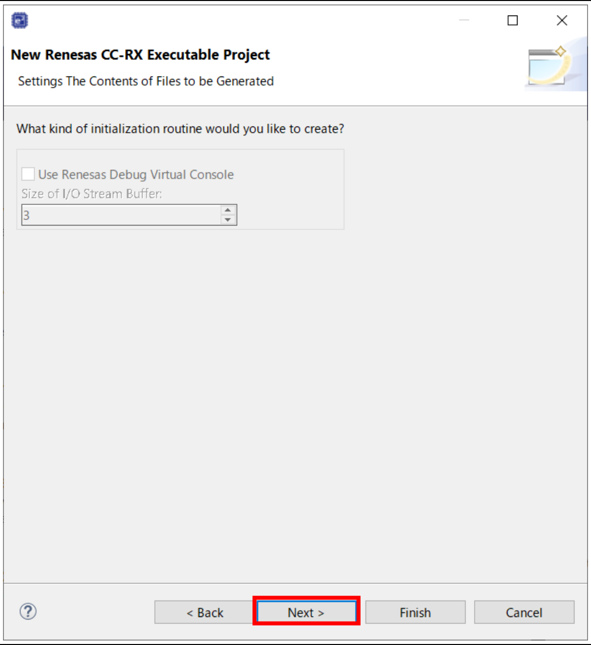
  RX660 - 144 pin.
- Select 'E2 Lite (RX)' from the pulldown and check 'Create Release Configuration' check box.
- Click ‘Next’.



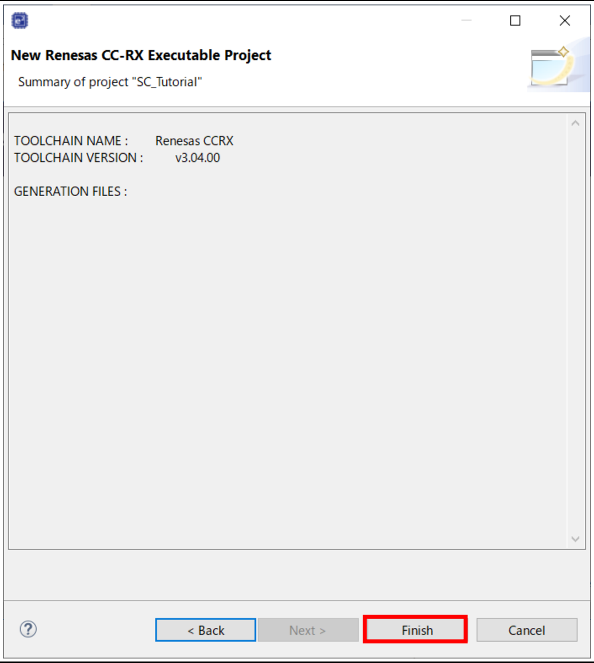
- In the ‘Select Coding Assistant settings’ dialog, select ‘Smart Configurator’.
- Click ‘Next’.



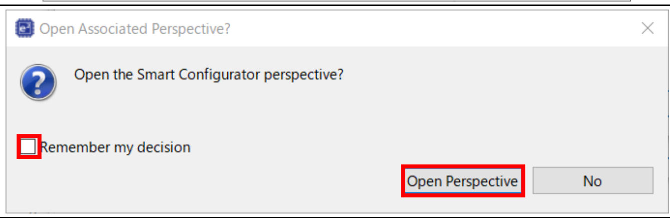
- Click ‘Next’.



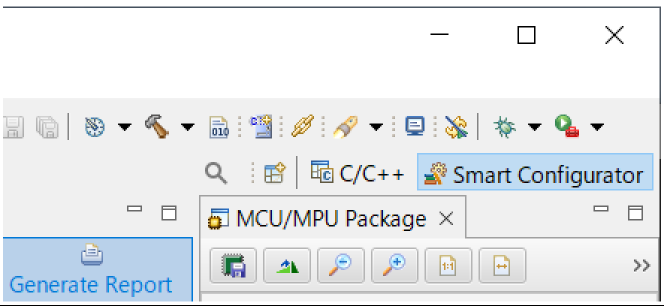
- A summary dialog will appear, click ‘Finish’ to complete the project generation.



- In future, to skip the pop-up message on the right, check the 'Remember my decision' check box and click on 'Open Perspective'.



- The perspective changes automatically when the Smart Configurator starts up.



4. Smart Configurator Using the e^2 studio

4.1 Introduction

The Smart Configurator plug-in for the RX660 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 RX660. 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/rskrx660/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**.

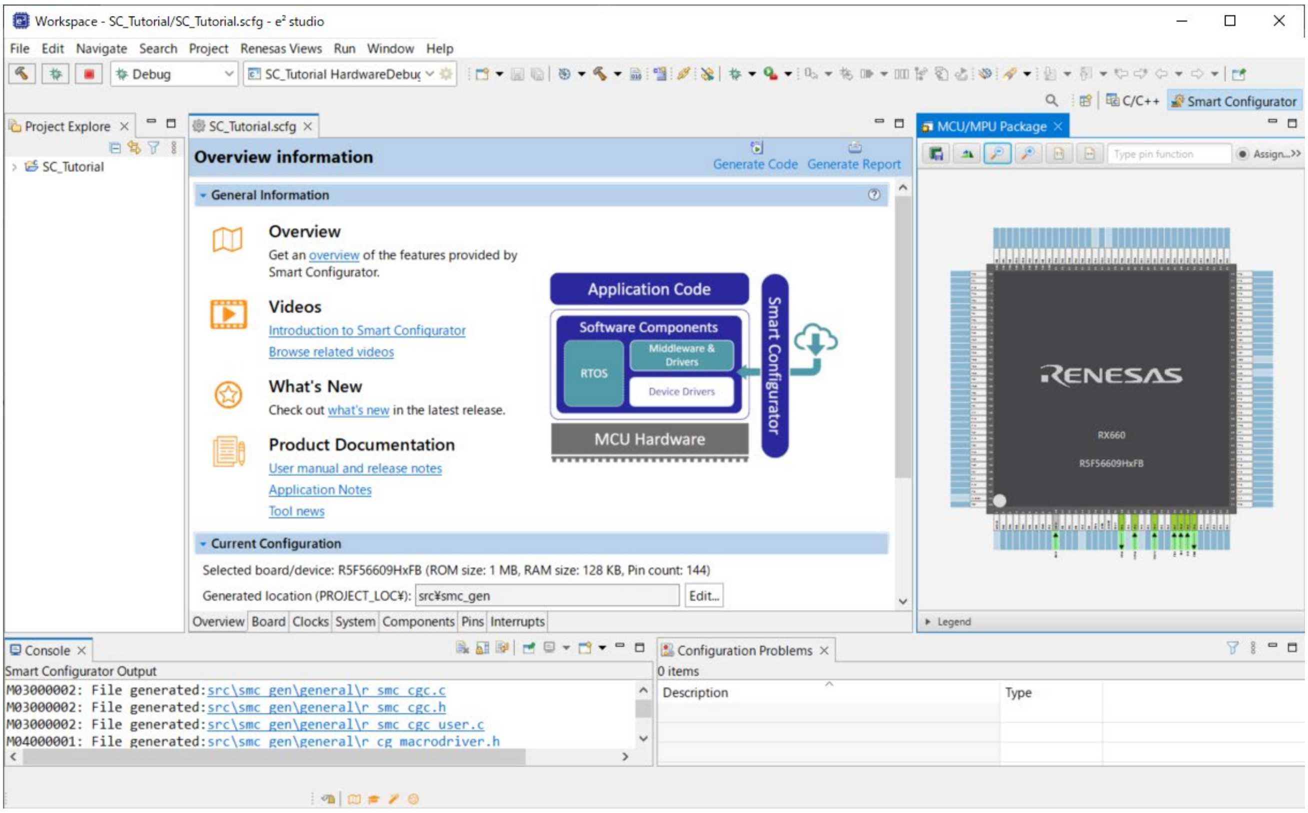


Figure 4-1 Overview page

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**.

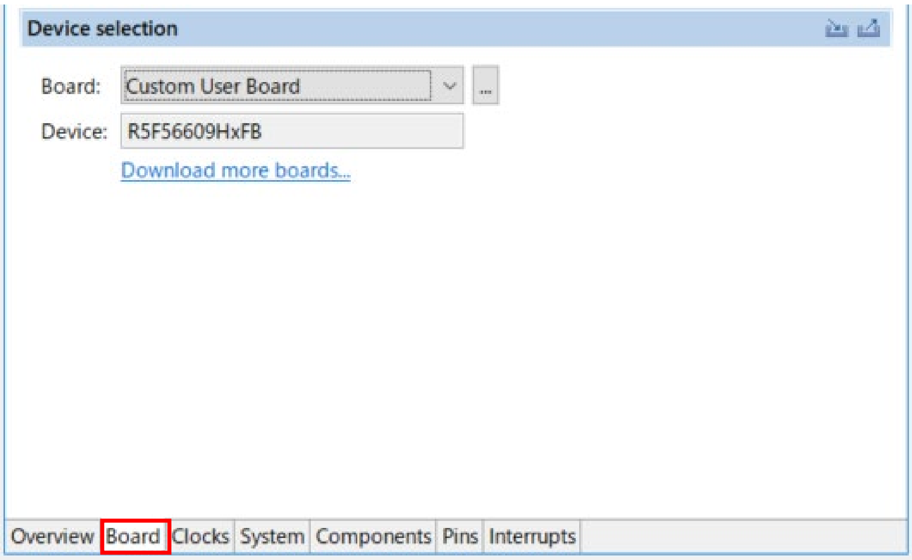


Figure 4-2 Board configuration page

4.3.1 Board configuration page

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

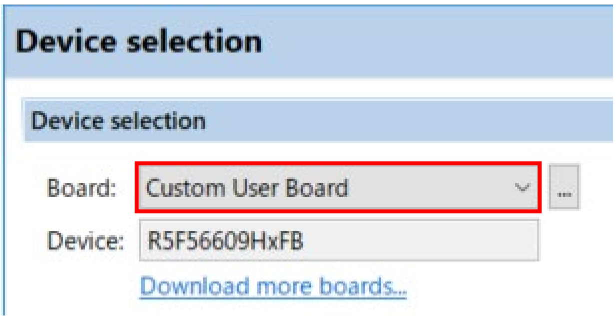


Figure 4-3 Select board

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 24 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**.

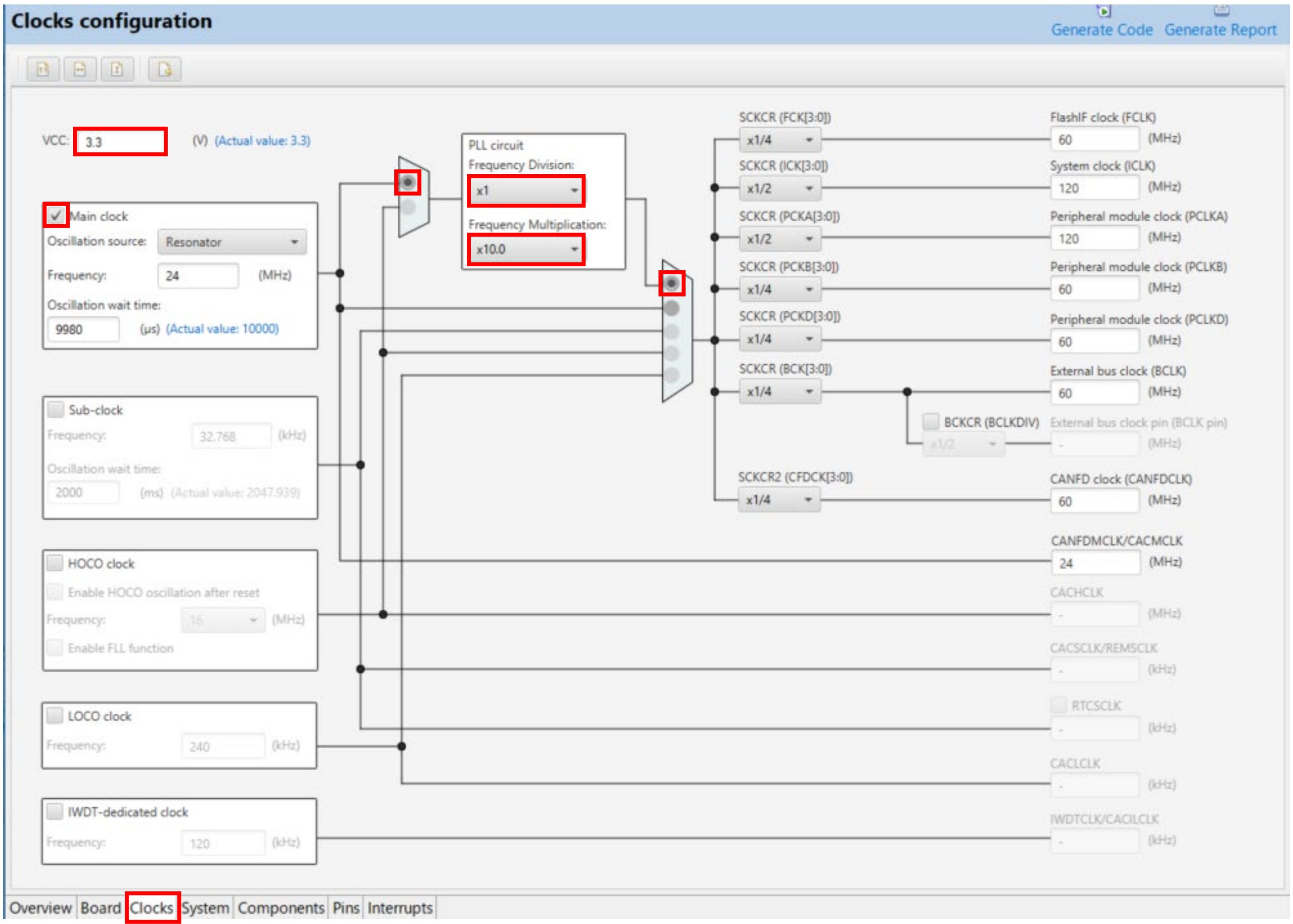


Figure 4-4 The ‘Clocks’ tabbed page

4.5 The 'System' tabbed page

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

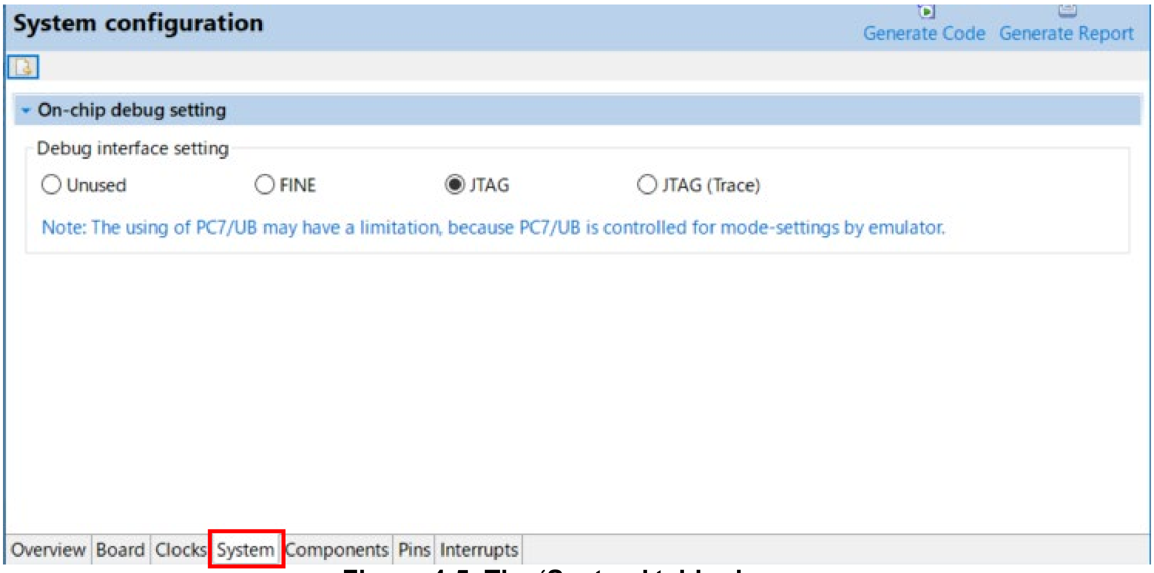


Figure 4-5 The 'System' tabbed page

4.5.1 On-chip debug setting

The On-chip debug settings set the interface used for debugging. For the RSKRX660 CPU board, select as shown in **Figure 4-6**.

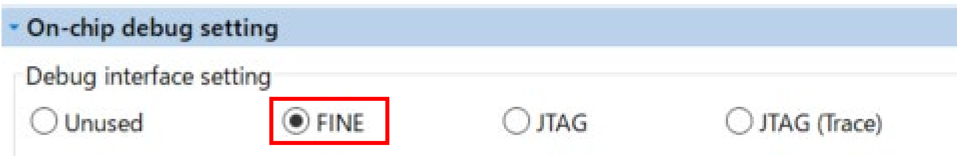


Figure 4-6 Debug interface setting

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.

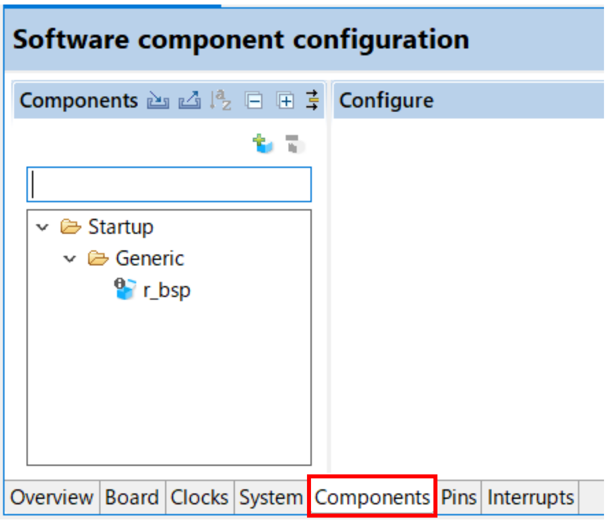


Figure 4-7 Components page

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.

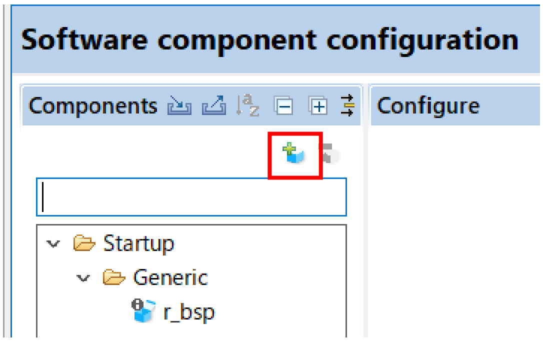


Figure 4-8 Add a Software component (1)

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

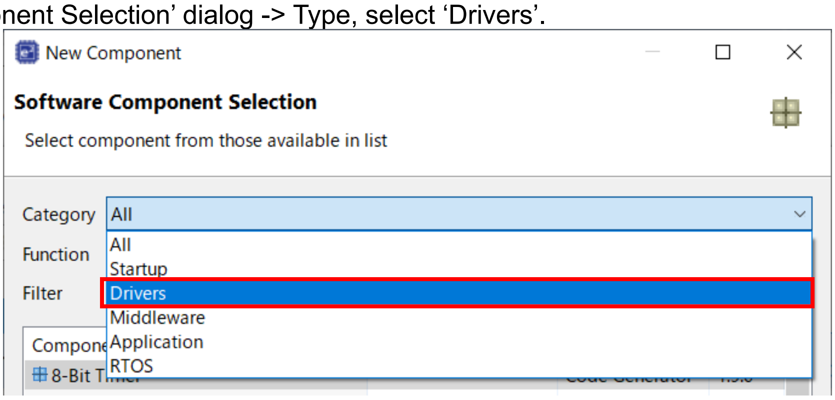


Figure 4-9 Add a Software component (2)

4.6.2 Compare Match Timer

CMT0 will be used as an interval timer for generation of accurate delays. Select ‘Compare Match Timer’ as shown in **Figure 4-10** below then click ‘Next’.

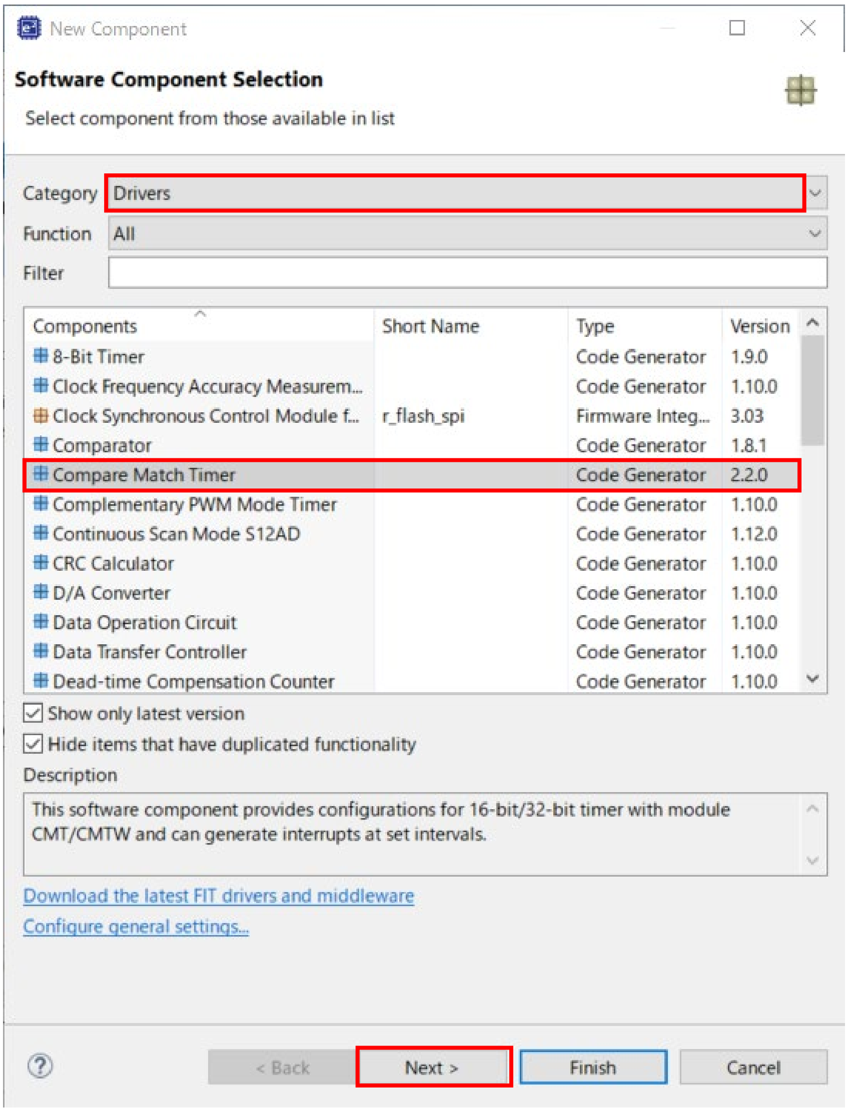


Figure 4-10 Compare Match Timer

In ‘Add new configuration for selected component’ dialog -> Resource, select ‘CMT0’ as shown in **Figure 4-11** below.

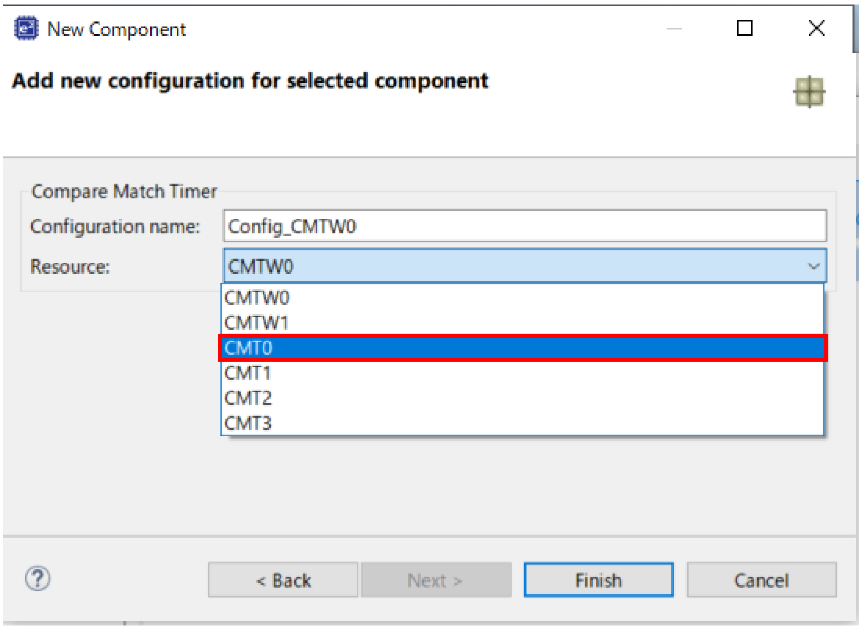


Figure 4-11 Select Resource - CMT0

Ensure that the ‘Configuration name’ updates to ‘Config_CMT0’ as shown in **Figure 4-12** below then click ‘Finish’.

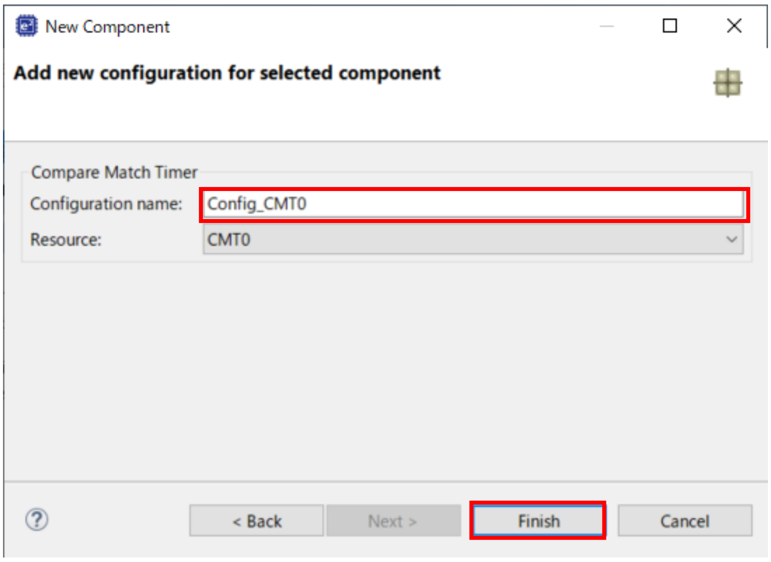


Figure 4-12 Ensure Configuration name - CMT0

In ‘Config_CMT0’, configure CMT0 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.

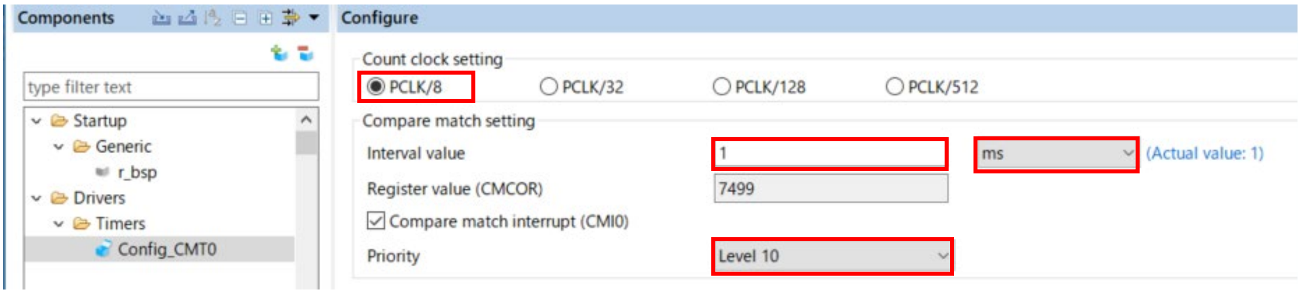


Figure 4-13 Config_CMT0 setting

CMT1 will be used as an interval timer for generation of accurate delays. CMT1 and CMT2 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’.

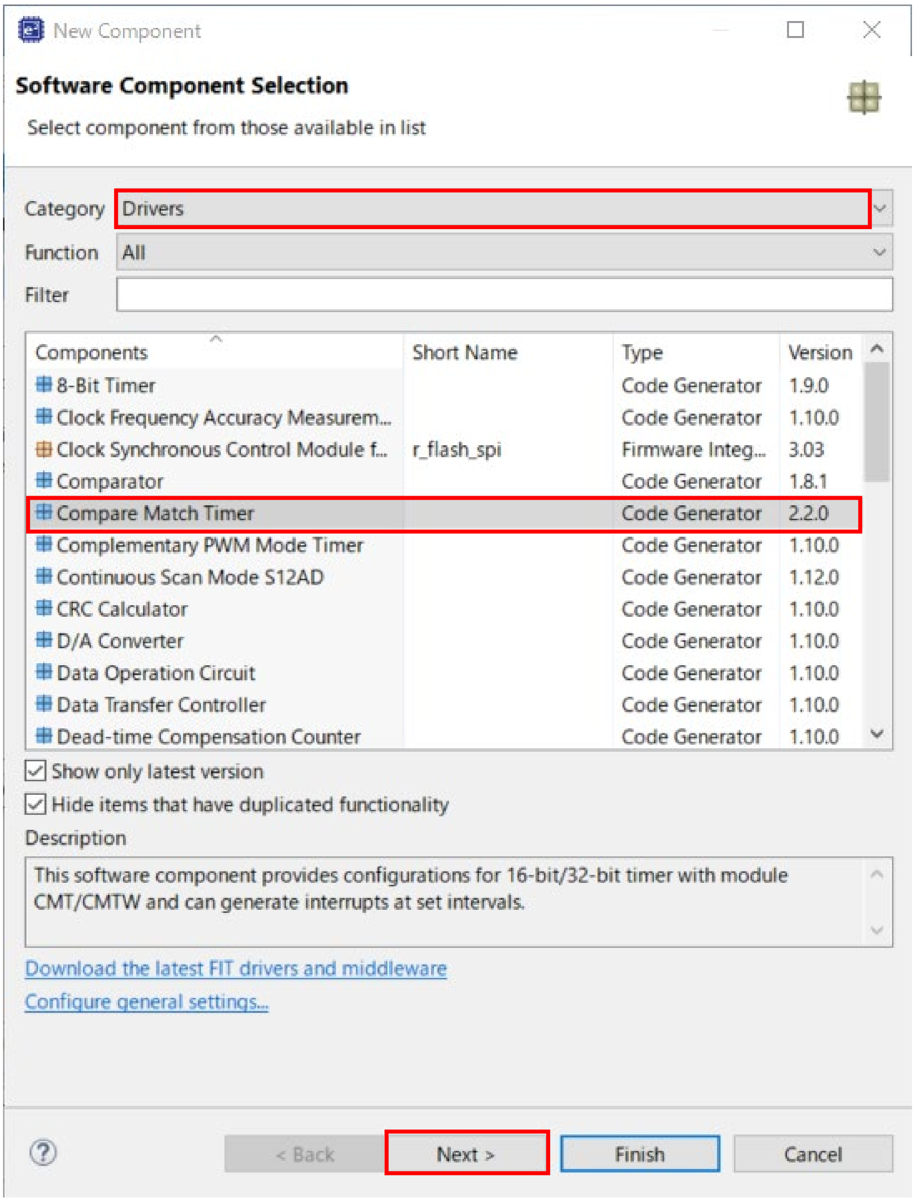


Figure 4-14 Select Compare Match Timer

In 'Add new configuration for selected component' dialog -> Resource, select 'CMT1' as shown in **Figure 4-15** below.

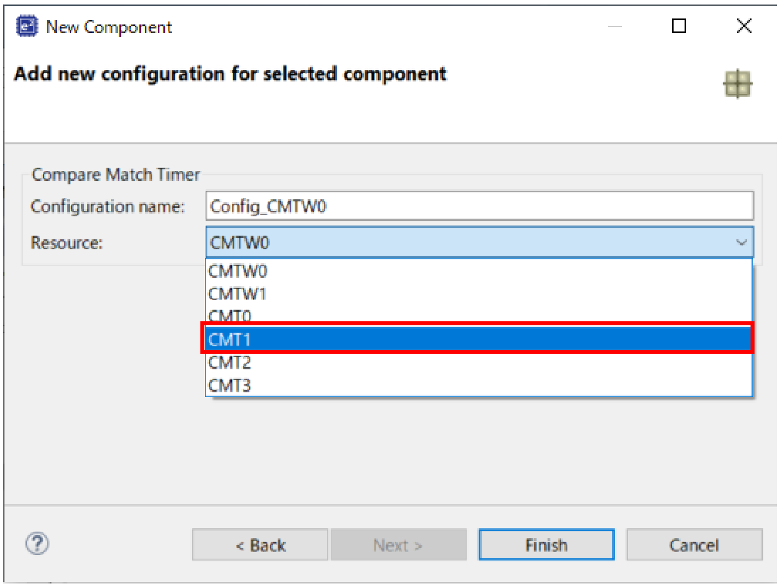


Figure 4-15 Select Resource - CMT1

Ensure that the 'Configuration name' updates to 'Config_CMT1' as shown in **Figure 4-16** below then click 'Finish'.

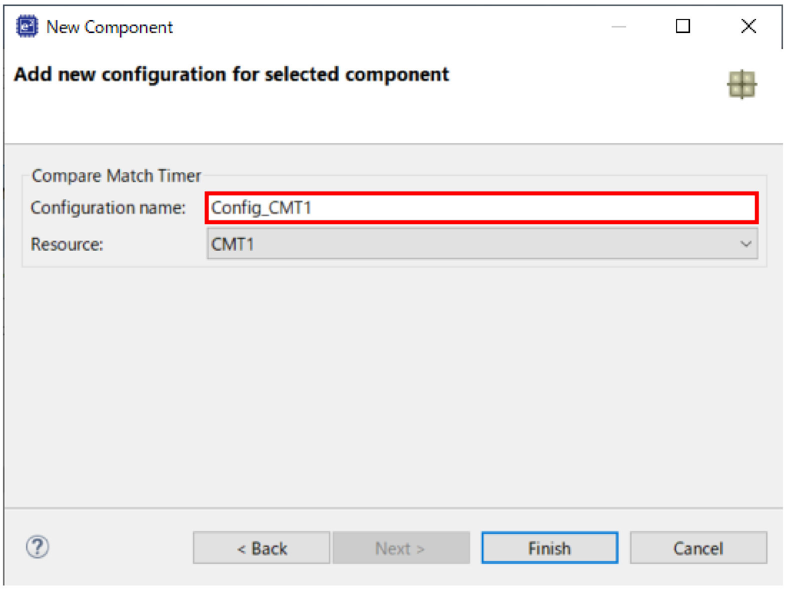


Figure 4-16 Ensure Configuration name - CMT1

In 'Config_CMT1', configure CMT1 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.

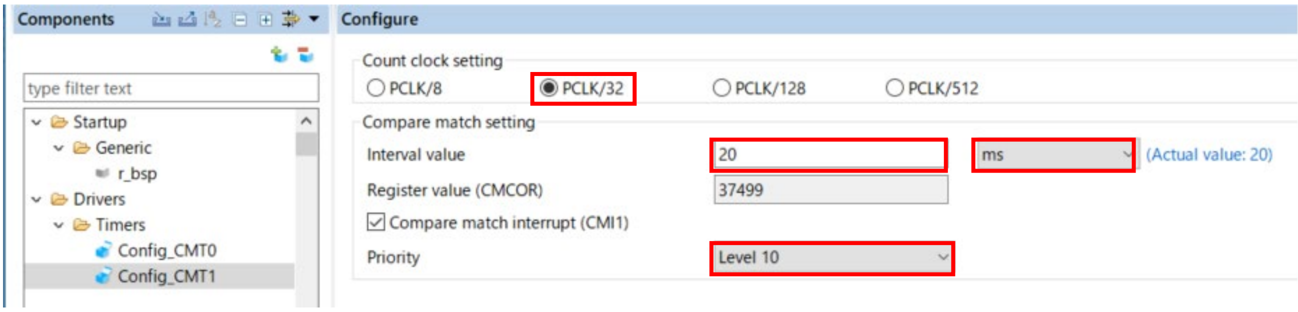


Figure 4-17 Config_CMT1 setting

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 'CMT2' as shown in **Figure 4-18** below.

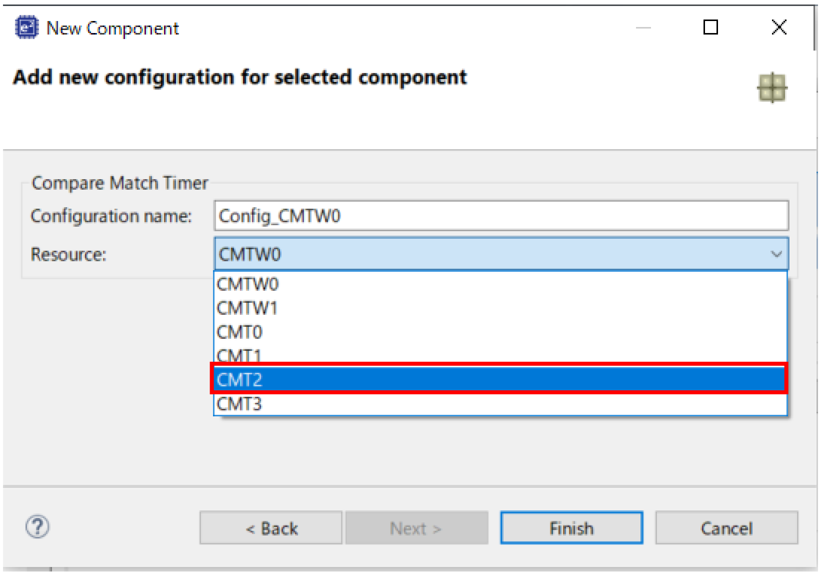


Figure 4-18 Select Resource – CMT2

Ensure that the 'Configuration name' updates to 'Config_CMT2' as shown in **Figure 4-19** below then click 'Finish'.

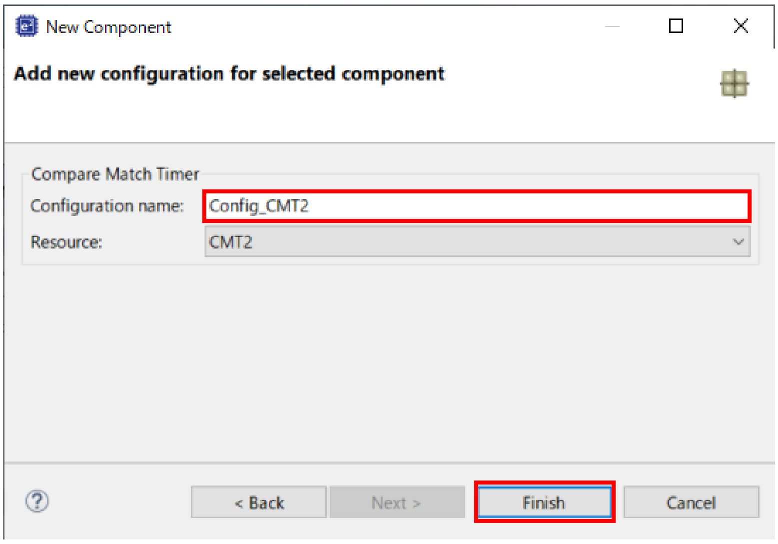


Figure 4-19 Ensure Configuration name – CMT2

Navigate to the 'Config_CMT2' and configure CMT2 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.

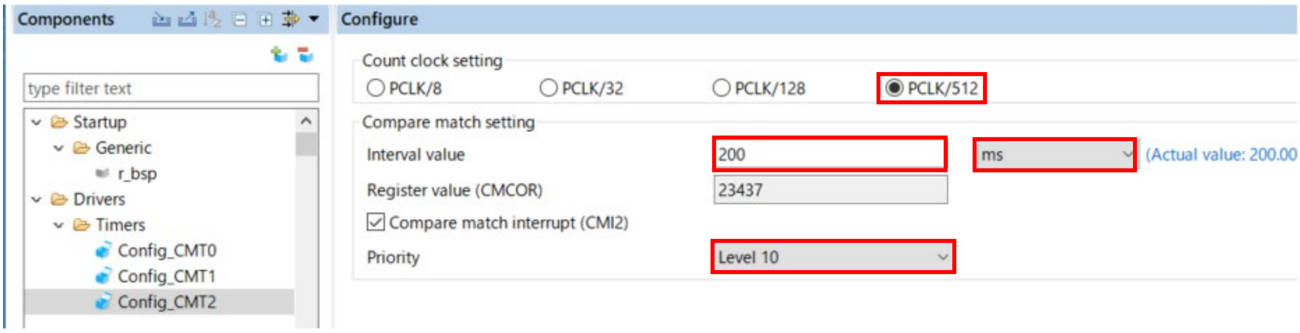


Figure 4-20 Config_CMT2 setting

4.6.3 Interrupt Controller

Referring to the RSK schematic, SW1 is connected to IRQ9(P91) and SW2 is connected to IRQ10(P92). SW3 is connected to ADTRG0n (P07). This tutorial uses ADTRG0n, which will be configured later in section 4.6.7.

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’.

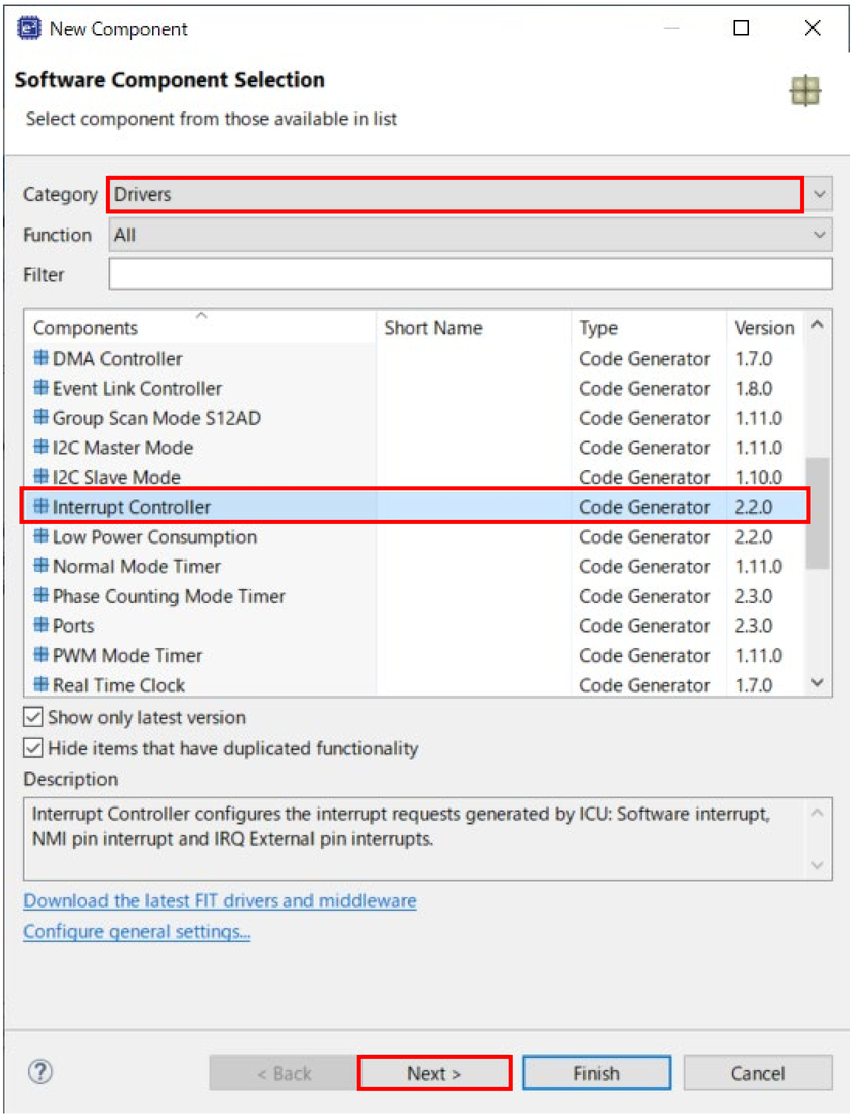


Figure 4-21 Select Interrupt Controller

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

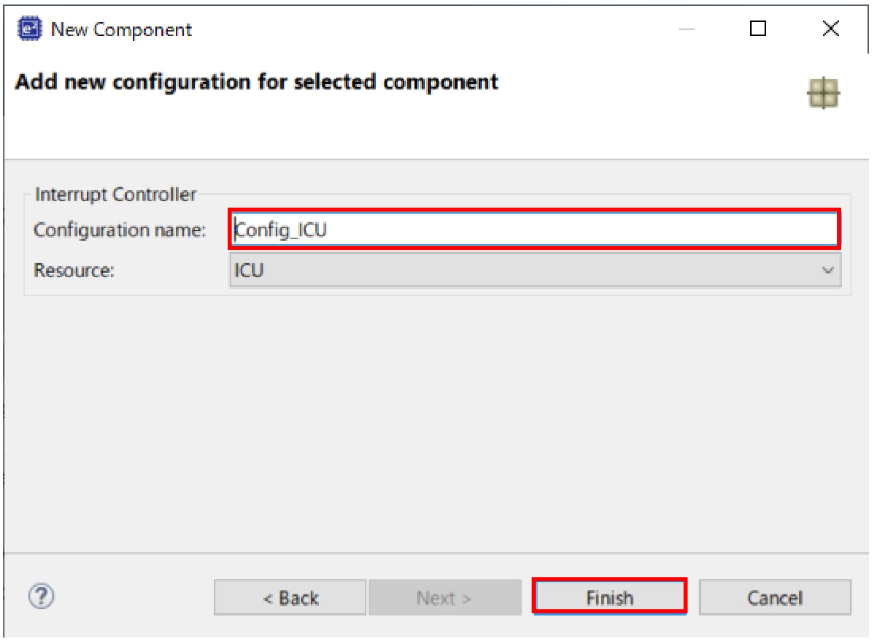


Figure 4-22 Select Resource – ICU

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

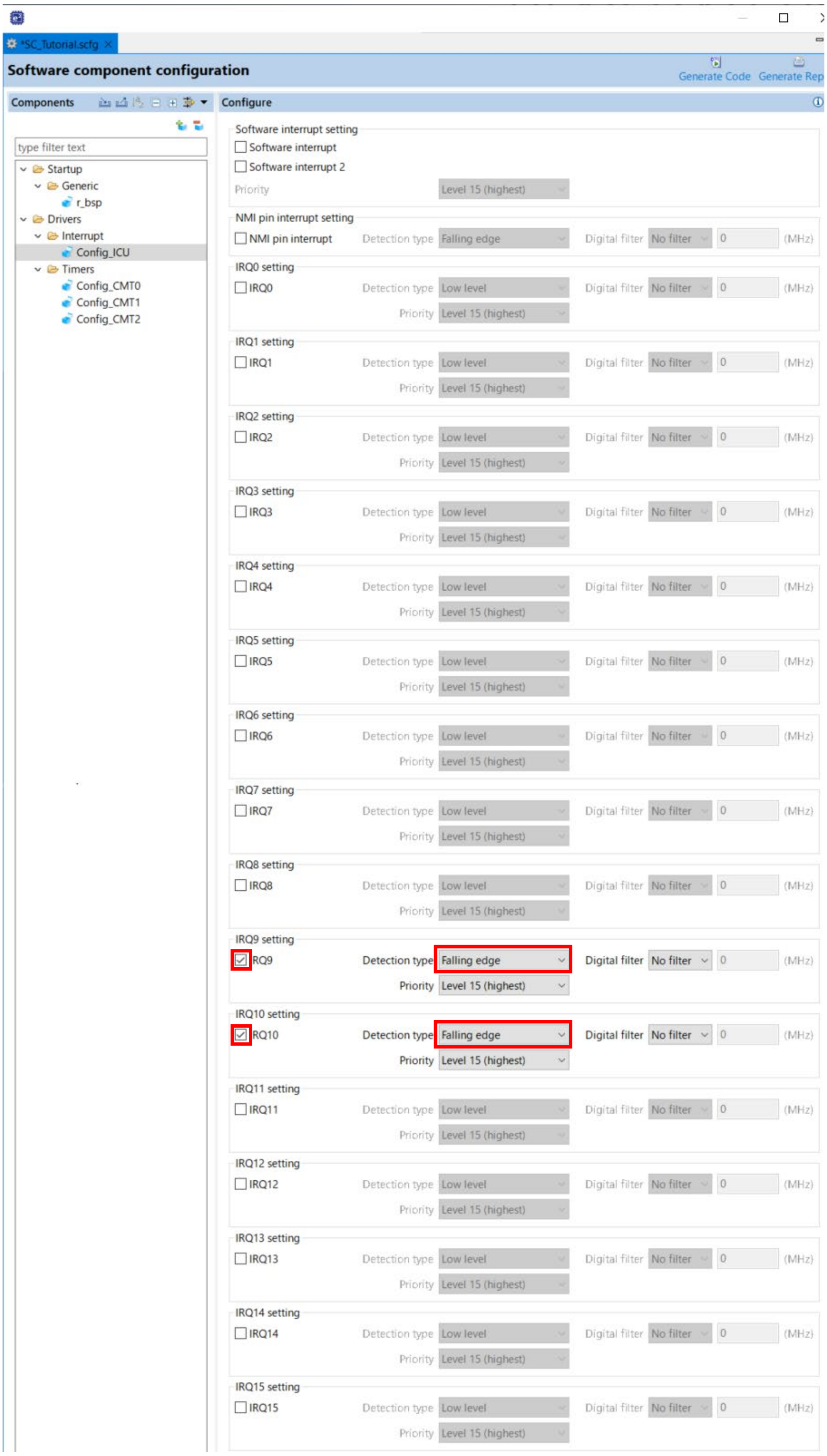


Figure 4-23 Config_ICU setting

4.6.4 Ports

Referring to the RSK schematic, LED0 is connected to P17, LED1 is connected to PF5, LED2 is connected to P04 and LED3 is connected to P06. PJ3 is used as one of the LCD control lines, together with PL0, P71 and P72.

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

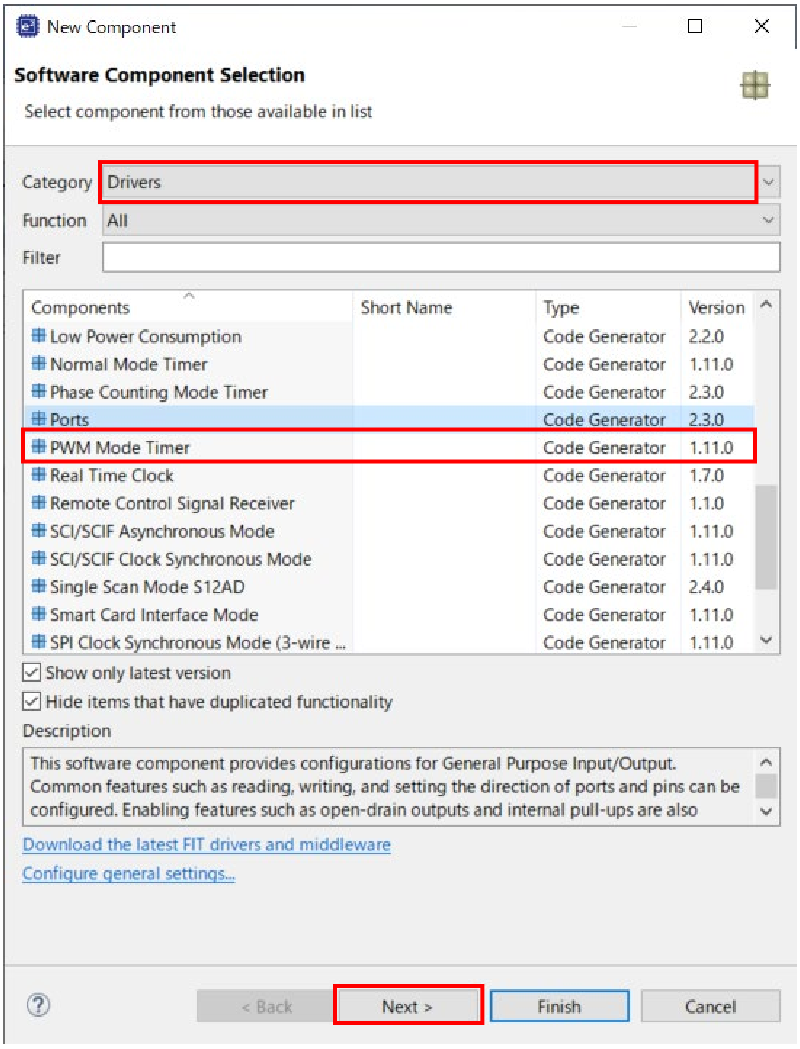


Figure 4-24 Select Ports

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

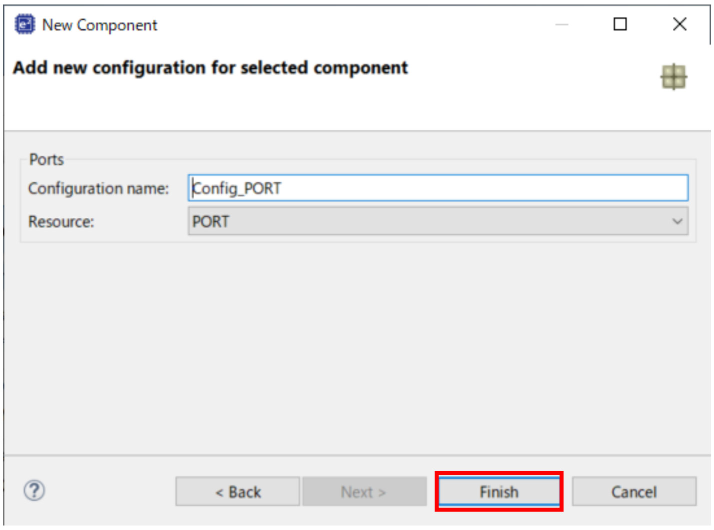


Figure 4-25 Select Resource – PORT

Tick the tickboxes for ‘PORT0’, ‘PORT1’, ‘PORT7’, ‘PORTF’ , ‘PORTJ’ and ‘PORTL’ as shown in **Figure 4-26** below.

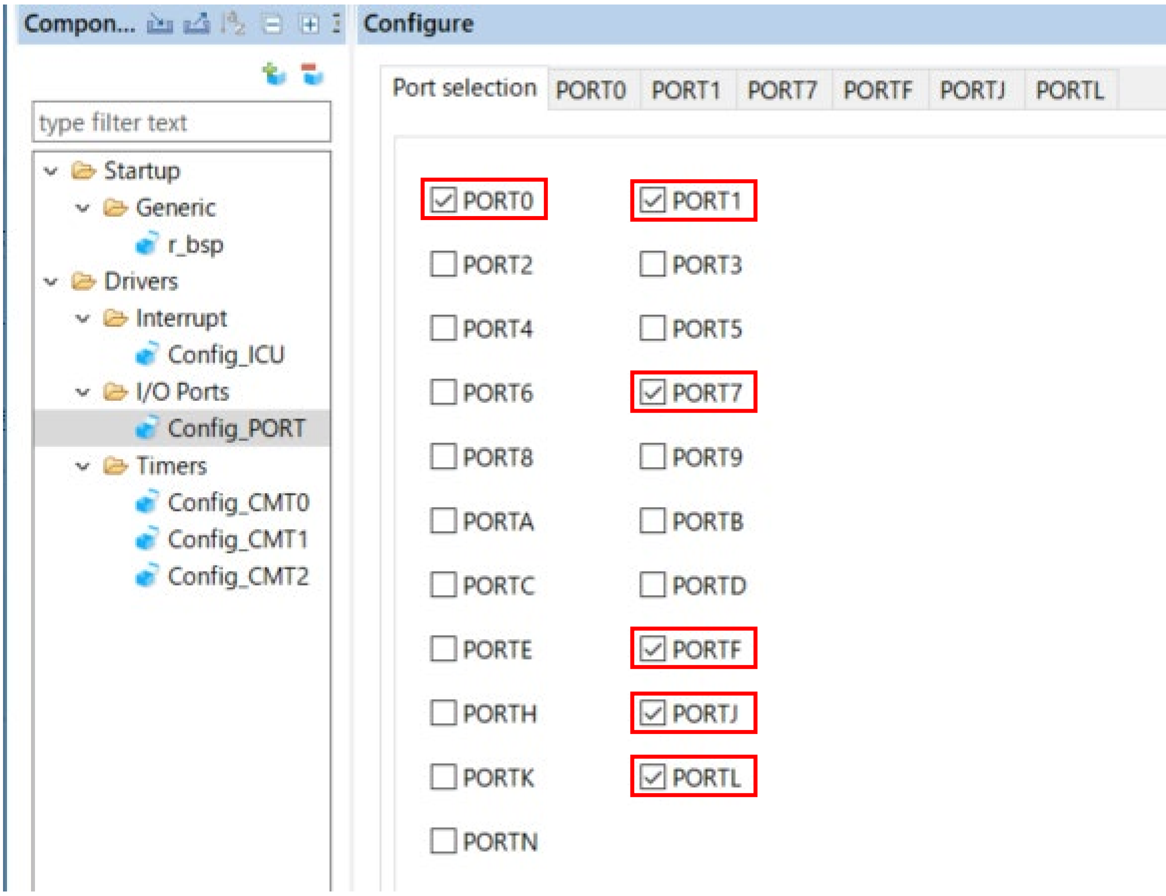


Figure 4-26 Select Port selection

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**, **Figure 4-31** and **Figure 4-32** below. Tick the tickboxes for 'Out' and tick 'Output 1' the tickboxes except for P72 under the 'PORT7' tab. Start with the 'PORT0' tab.

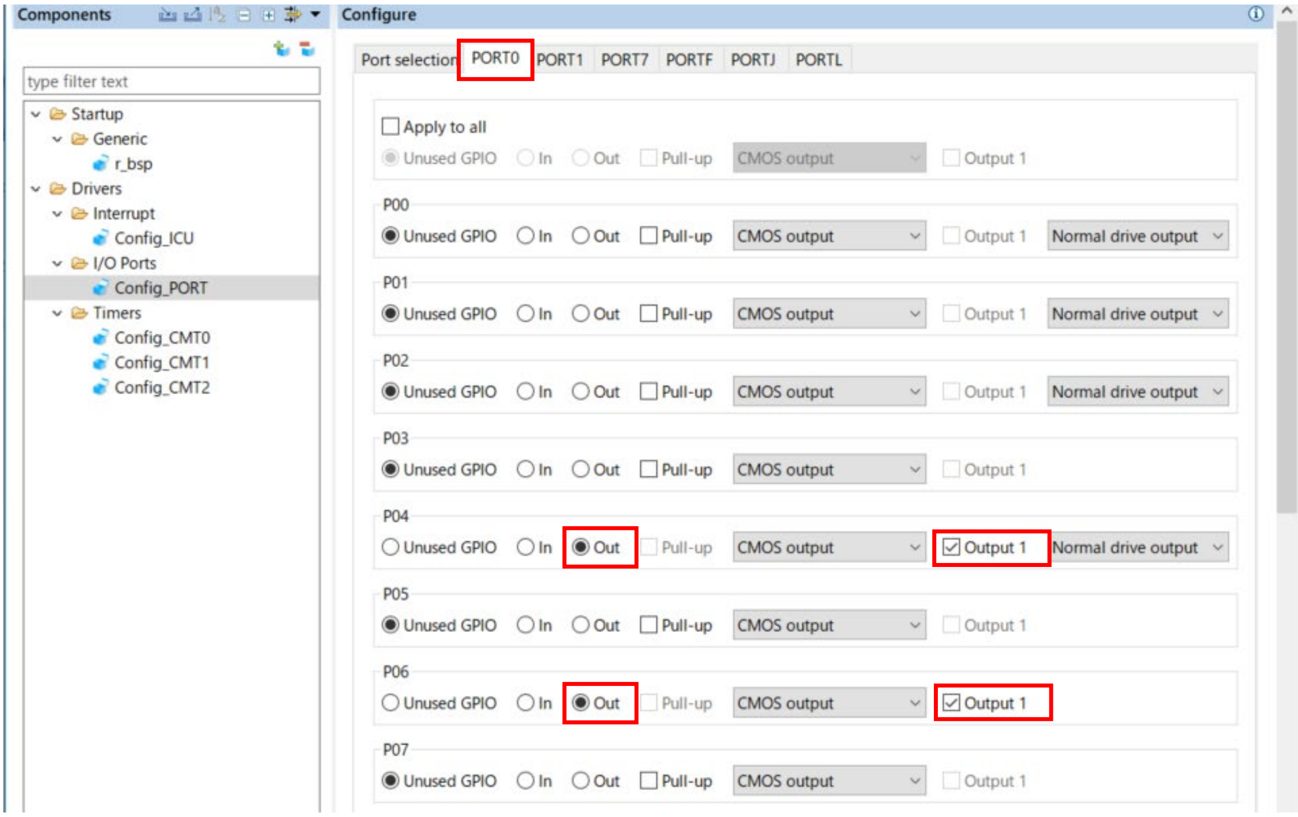


Figure 4-27 Select PORT0 tab

Select 'PORT1' tab.

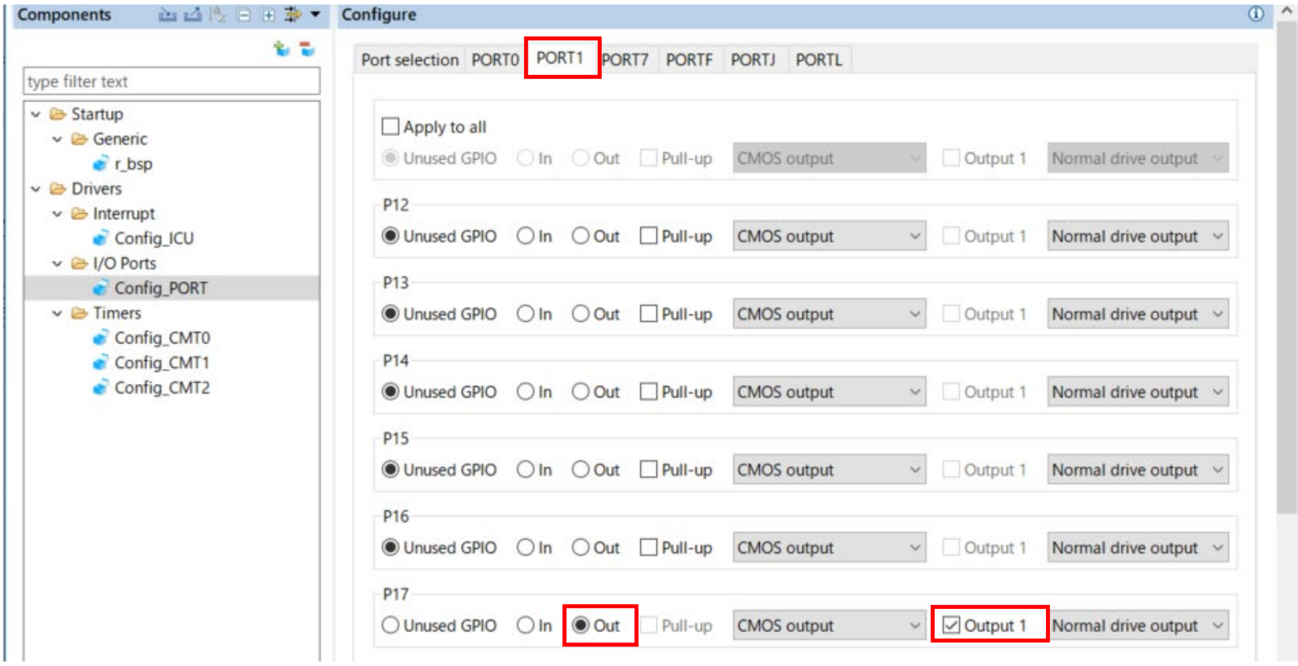


Figure 4-28 Select PORT1 tab

Select ‘PORT7’ tab.

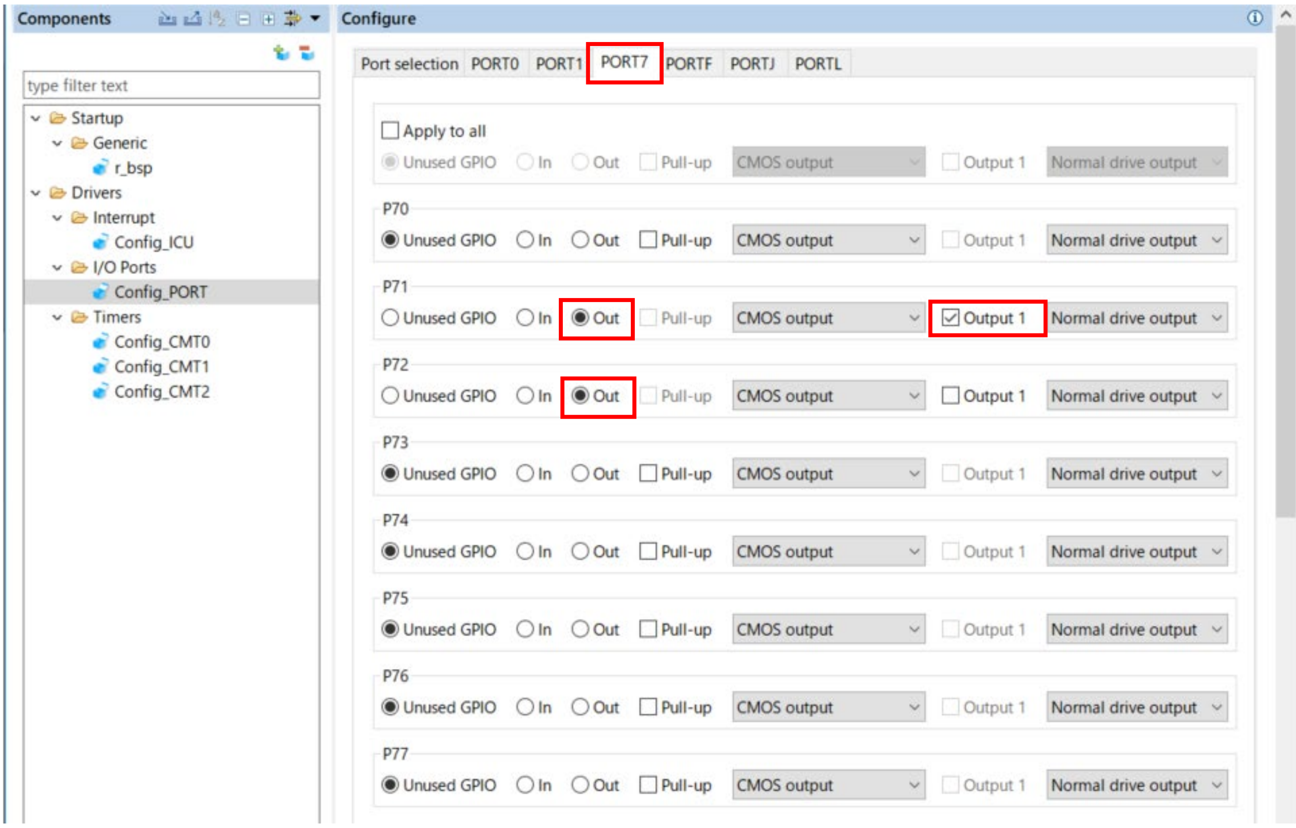


Figure 4-29 Select PORT7 tab

Select ‘PORTF’ tab.

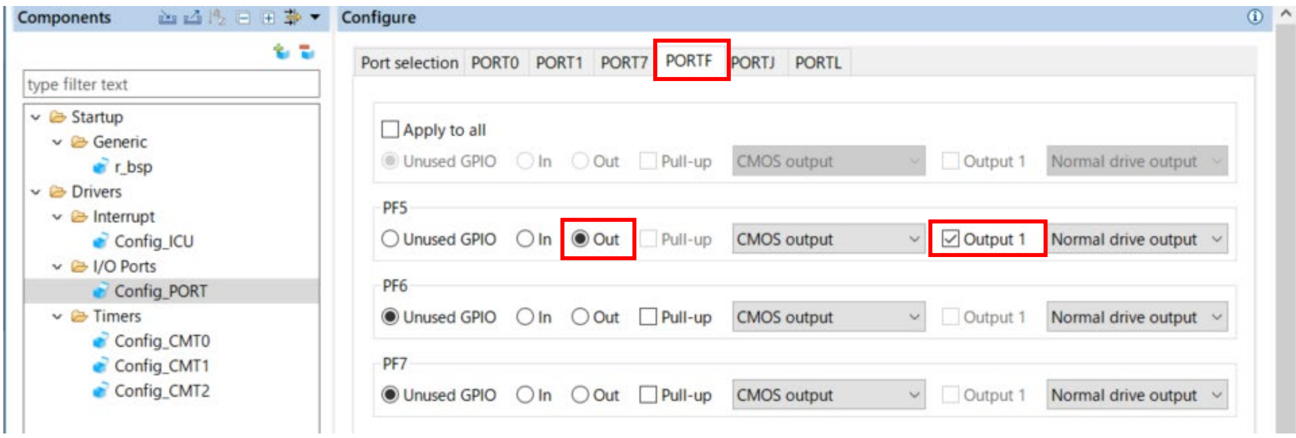


Figure 4-30 Select PORTF tab

Select 'PORTJ' tab.

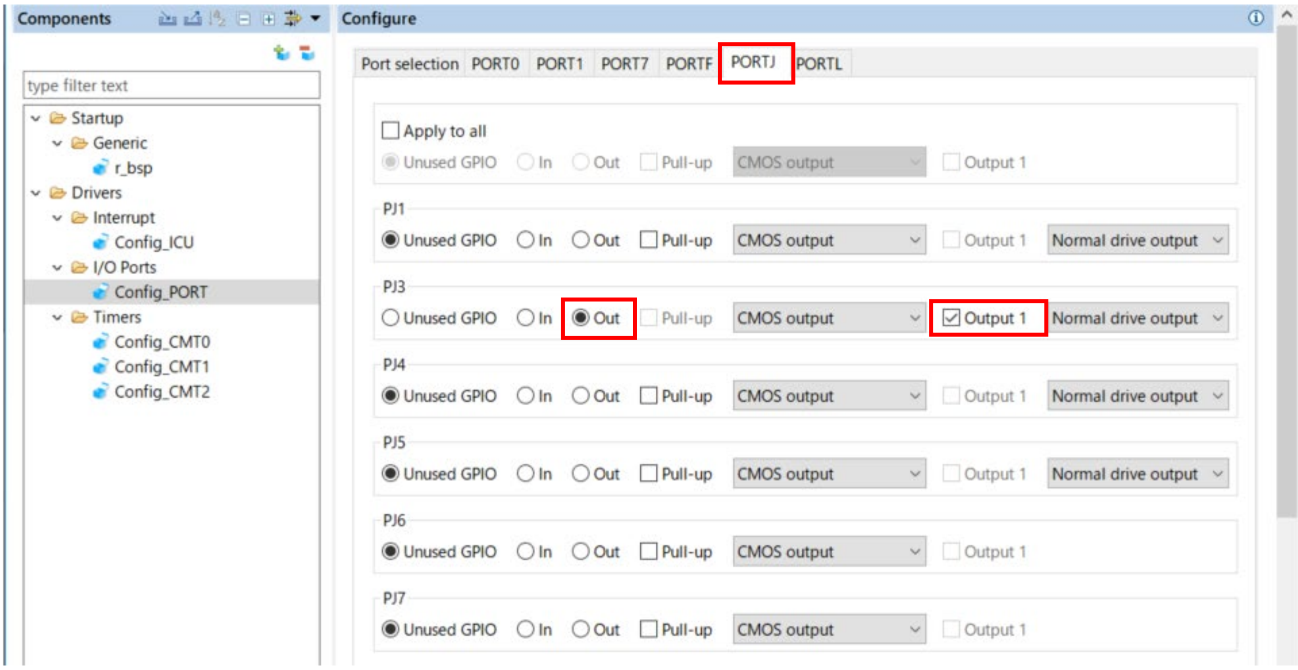


Figure 4-31 Select PORTJ tab

Select 'PORTL' tab.

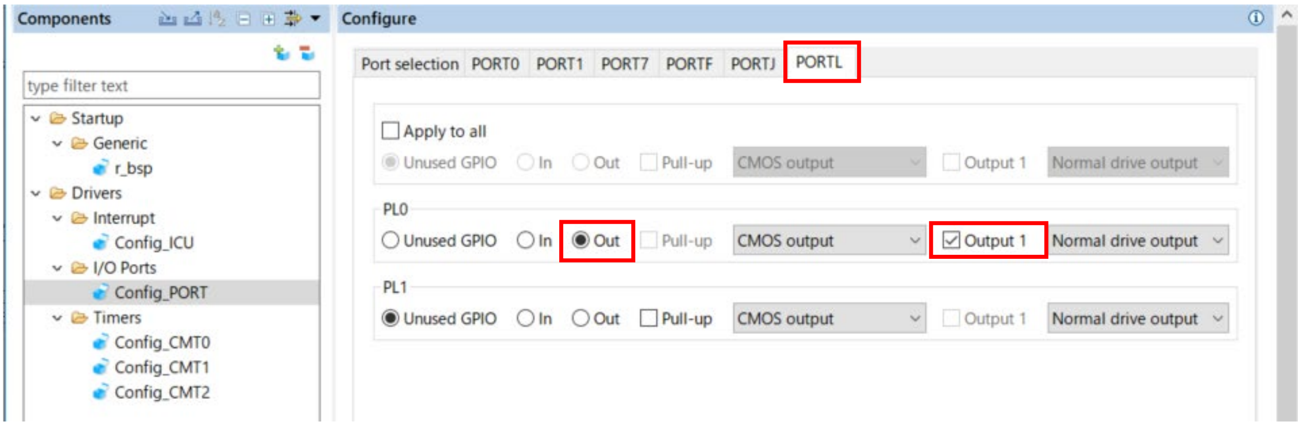


Figure 4-32 Select PORTL tab

4.6.5 SCI/SCIF Asynchronous Mode

In the RSKRX660, SCI10 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 4-33** then click 'Next'.

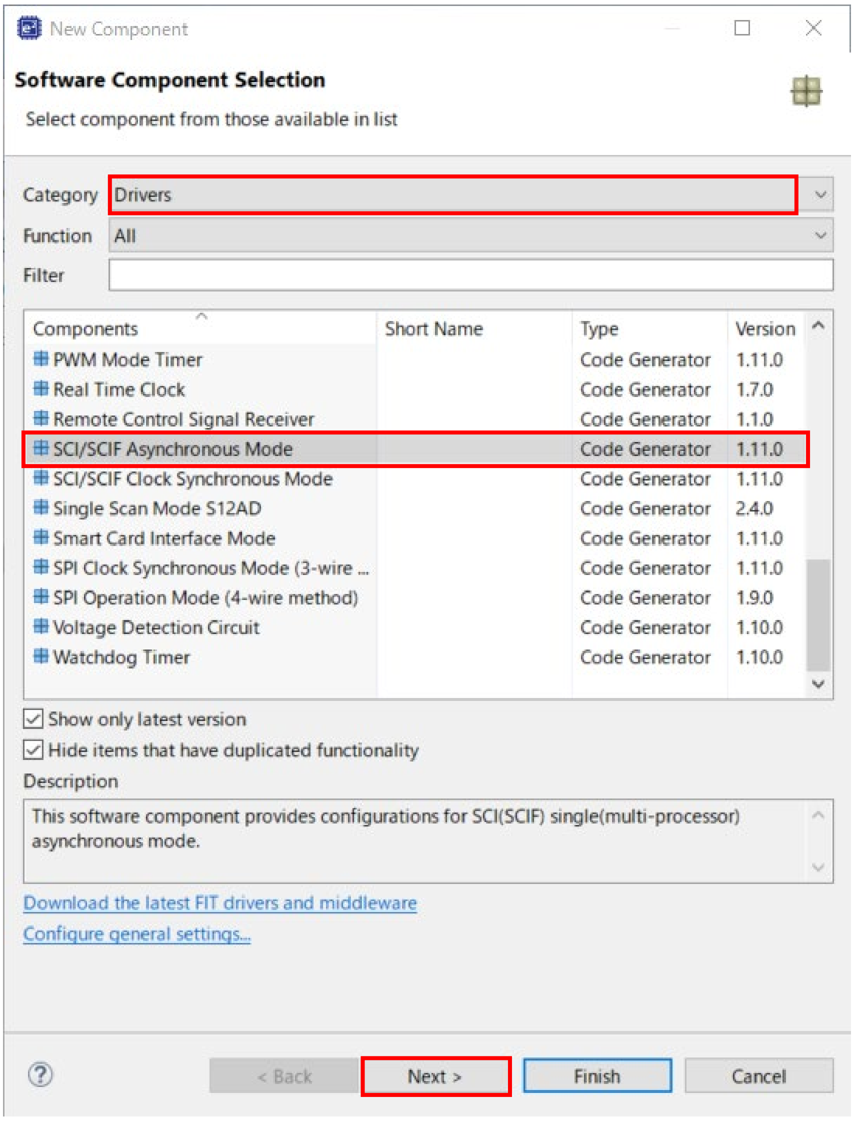


Figure 4-33 Select SCI/SCIF Asynchronous Mode

In 'Add new configuration for selected component' dialog -> Work mode, select 'Transmission/Reception' as shown in **Figure 4-34** below.

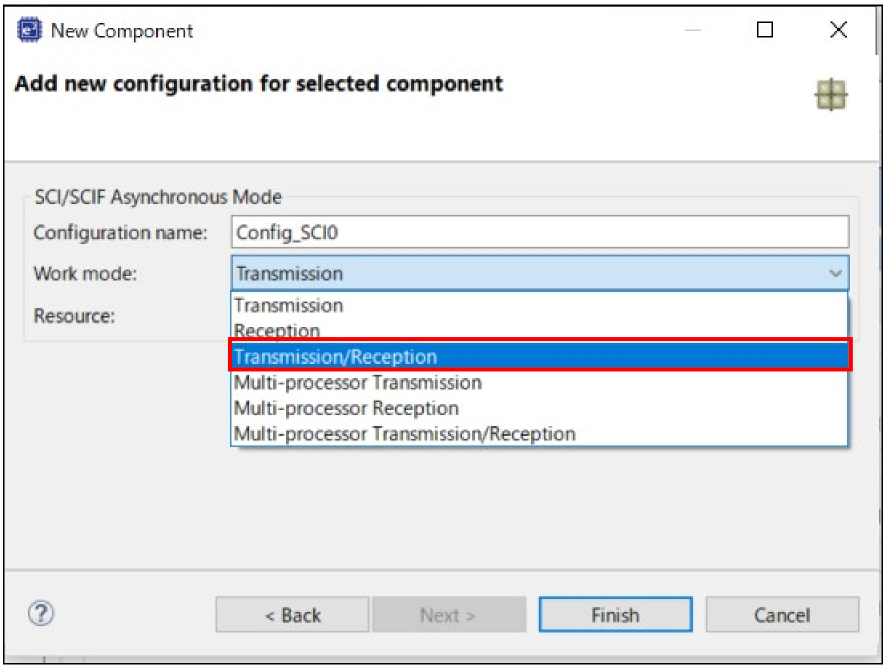


Figure 4-34 Select Work mode – Transmission/Reception

In 'Resource', select 'SCI10' as shown in **Figure 4-35** below.

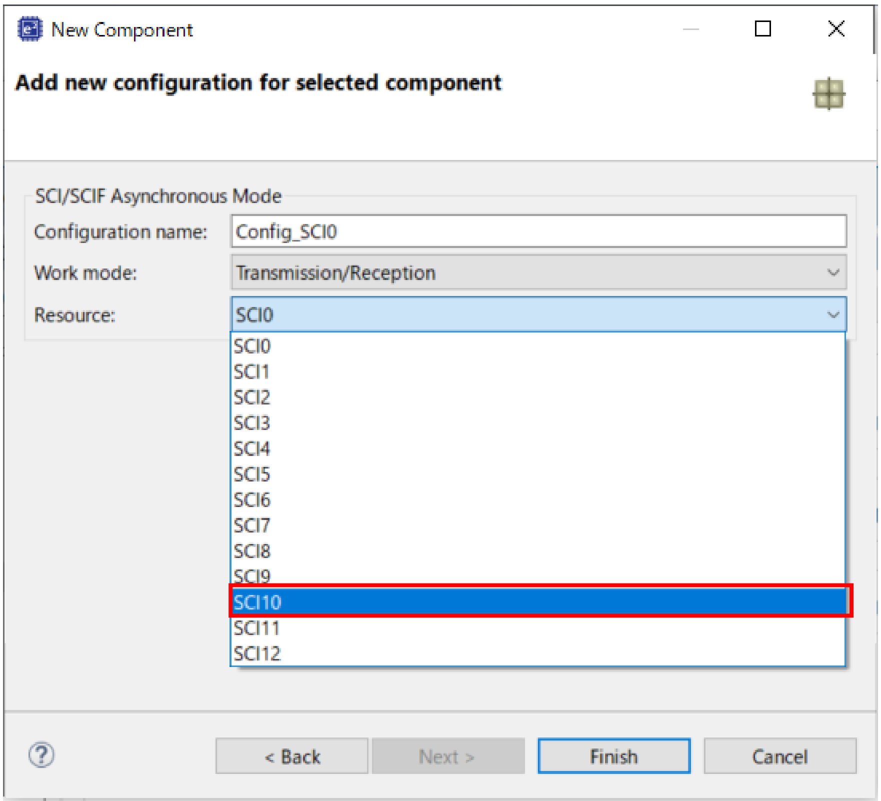


Figure 4-35 Select Resource – SCI10

Ensure that the 'Configuration name' updates to 'Config_SCI10' as shown in **Figure 4-36** below then click 'Finish'.

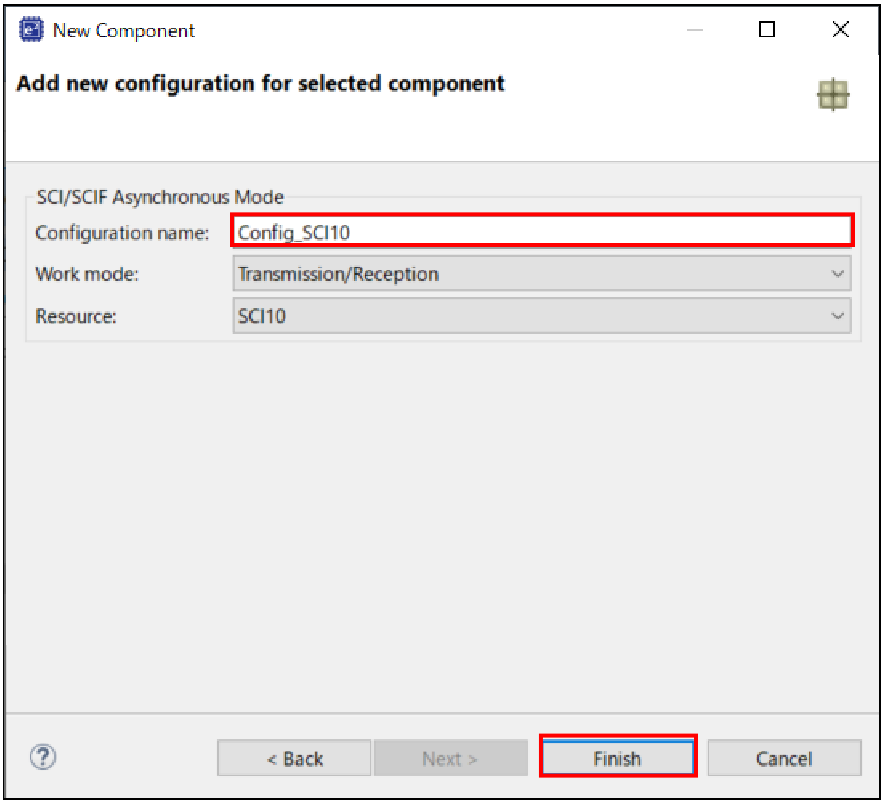


Figure 4-36 Ensure Configuration name - Config_SCI10

Configure SCI10 as shown in **Figure 4-37**. 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.

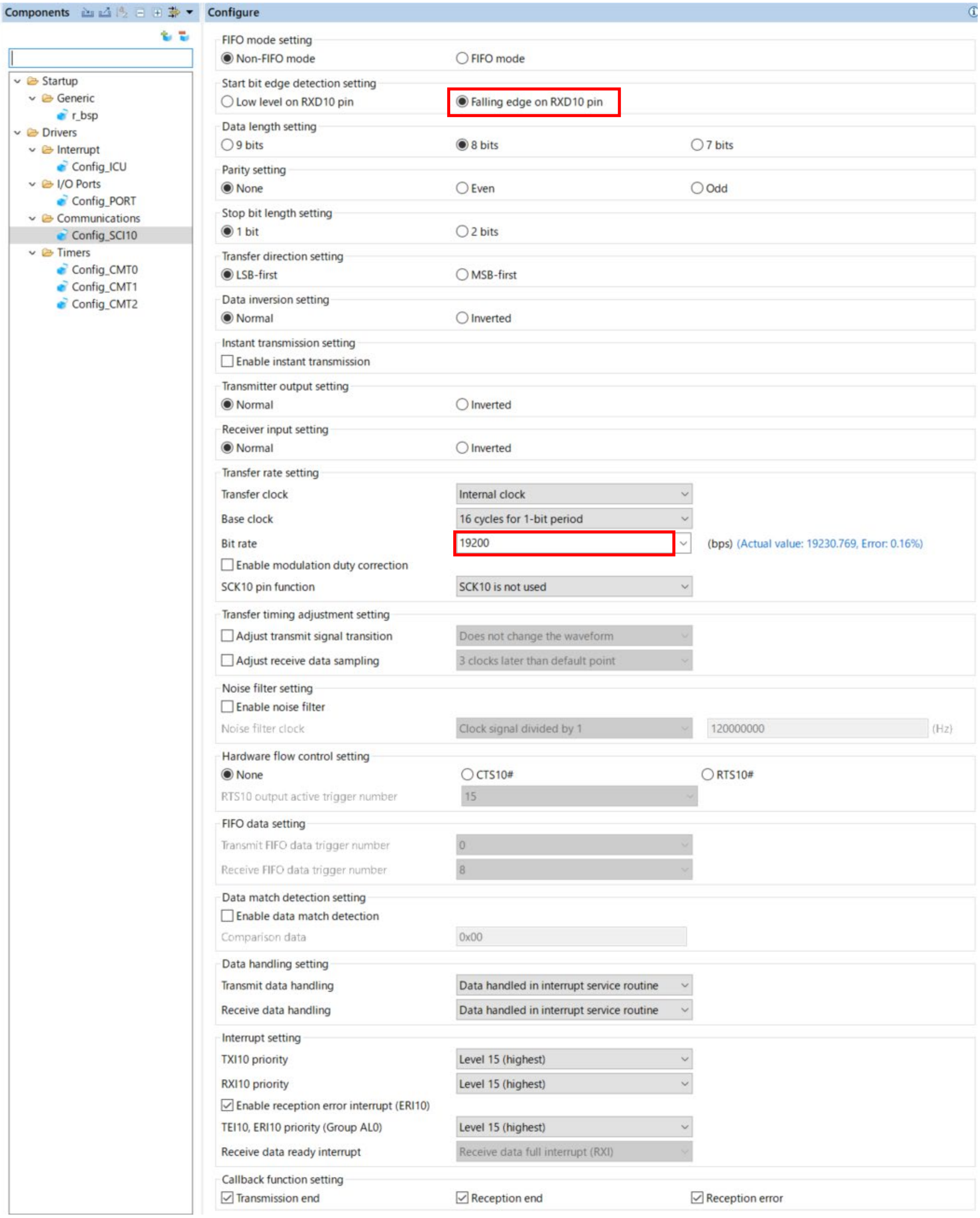


Figure 4-37 Config_SCI10 setting

4.6.6 SPI Clock Synchronous Mode

In the RSKRX660, 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 4-38** then click ‘Next’.

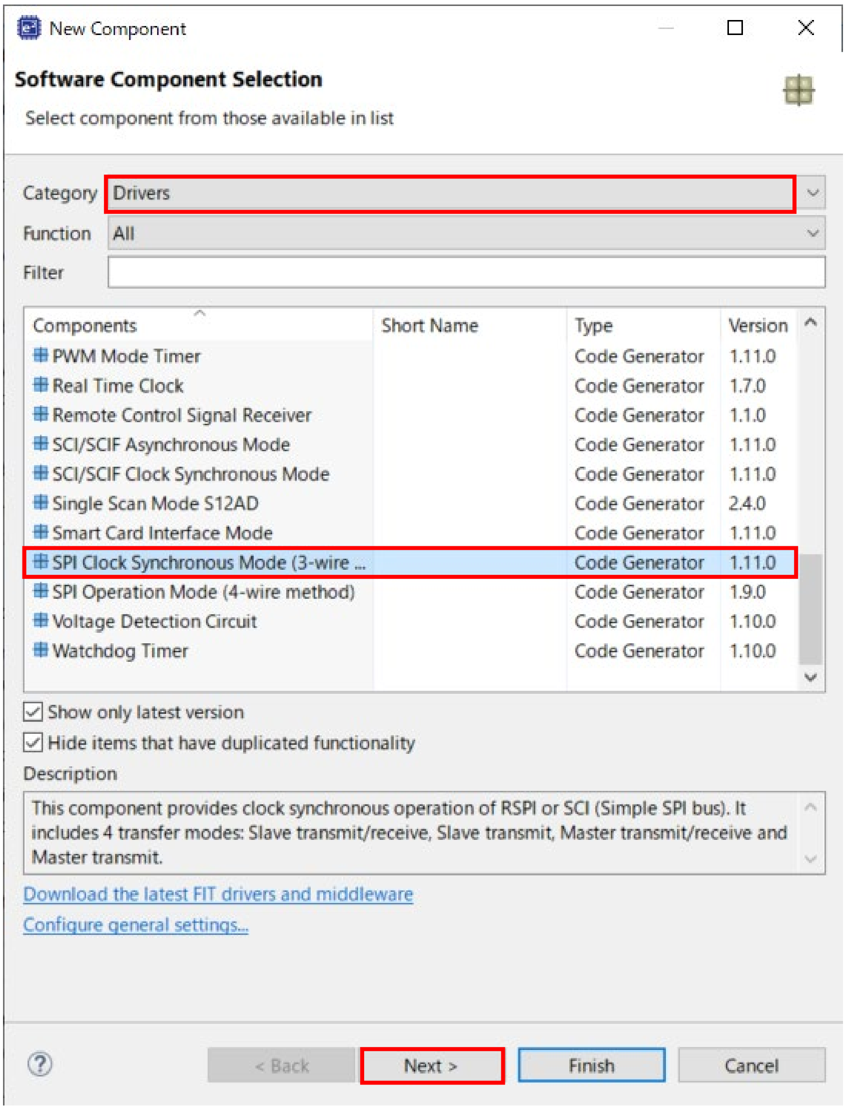


Figure 4-38 Select SPI Clock Synchronous Mode

In 'Add new configuration for selected component' dialog -> Operation, select 'Master transmit only' as shown in **Figure 4-39** below.

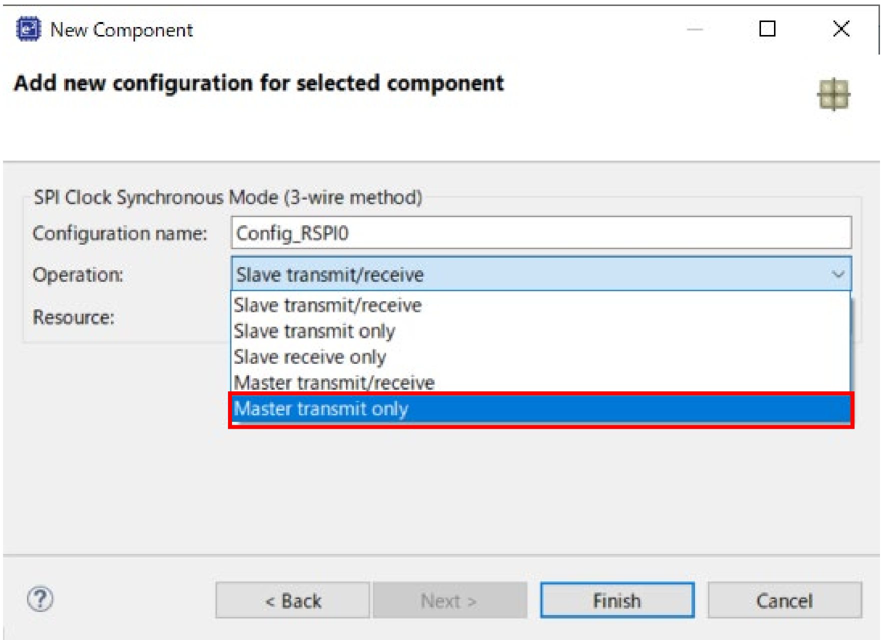


Figure 4-39 Select Operation – Master transmit only

In 'Resource', select 'SCI6' as shown in **Figure 4-40** below.

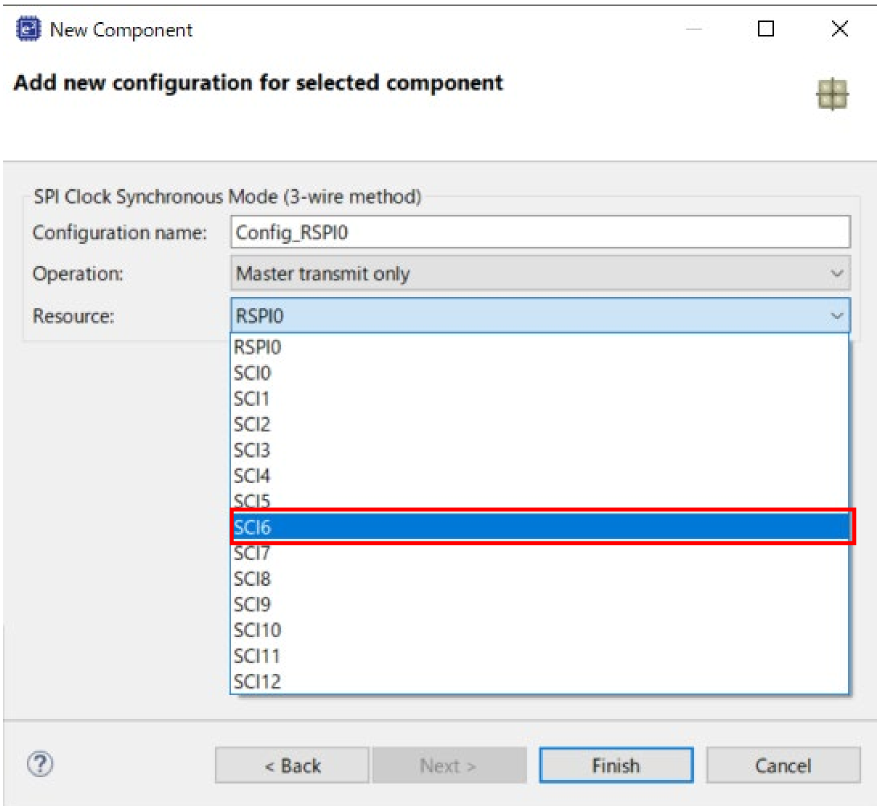


Figure 4-40 Select Resource – SCI6

Ensure that the 'Configuration name' updates to 'Config_SCI6' as shown in **Figure 4-41** below then click 'Finish'.

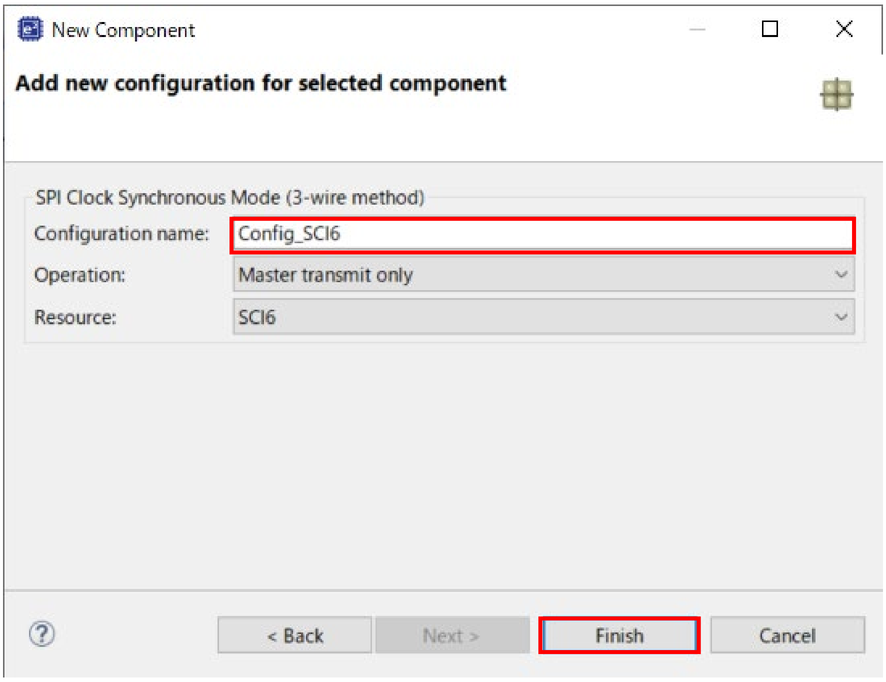


Figure 4-41 Ensure Configuration name - Config_SCI6

Configure SCI6 as shown in **Figure 4-42**. 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.

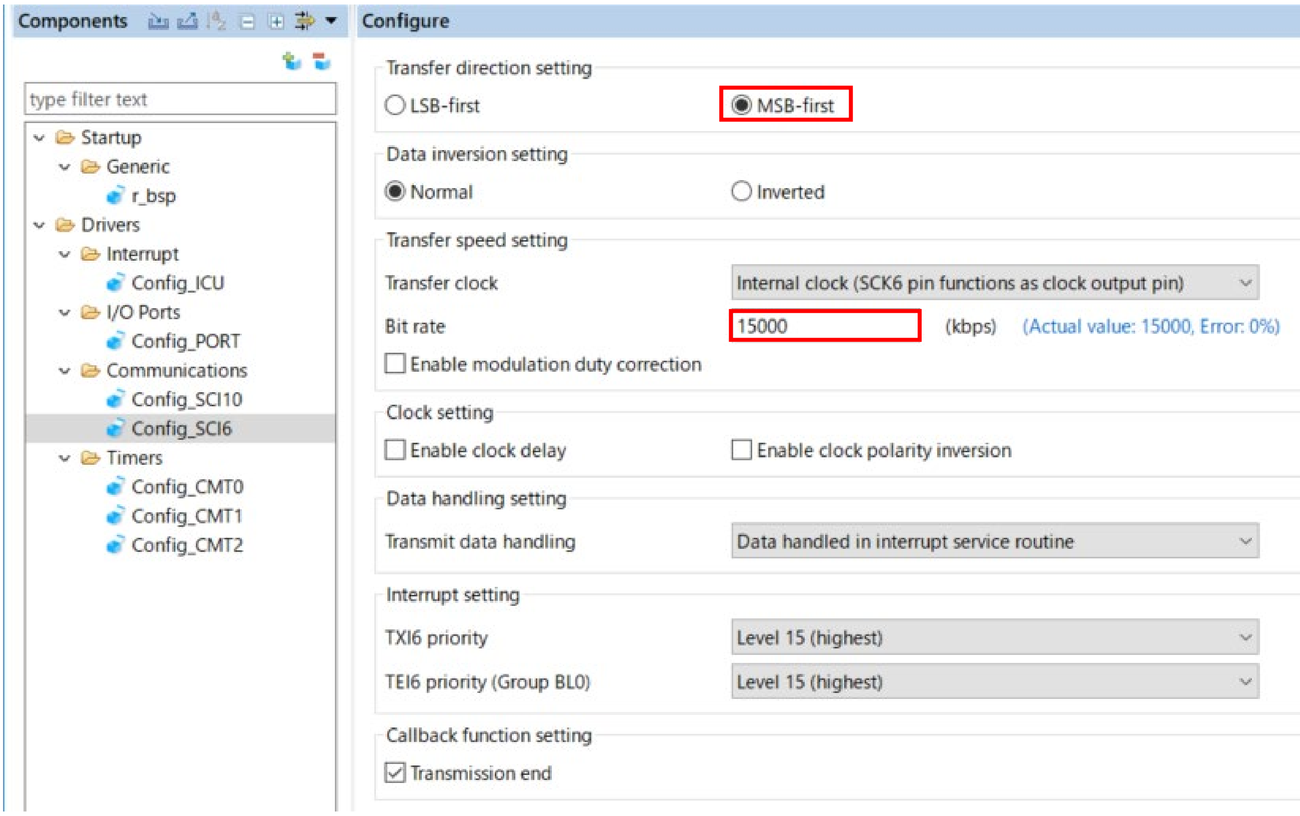


Figure 4-42 Config_SCI6 setting

4.6.7 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 4-43** then click ‘Next’.

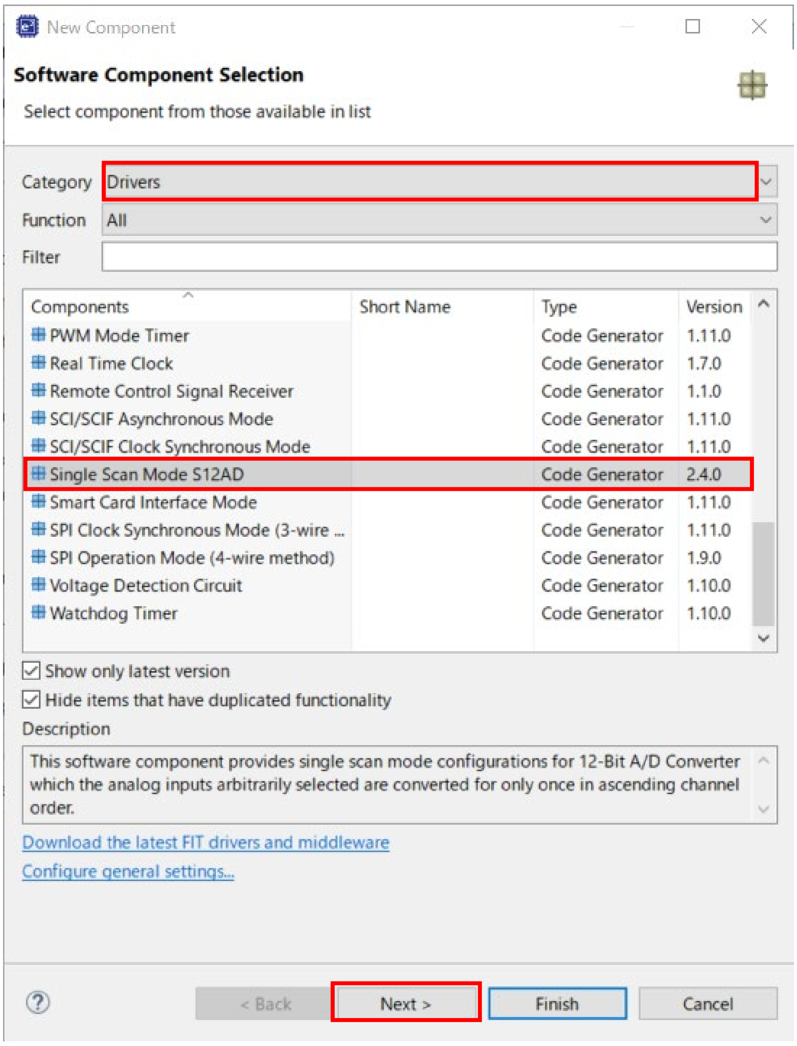


Figure 4-43 Select Single Scan Mode S12AD

Ensure that the 'Configuration name' is'Config_S12AD0' as shown in **Figure 4-44** below then click ‘Finish’.

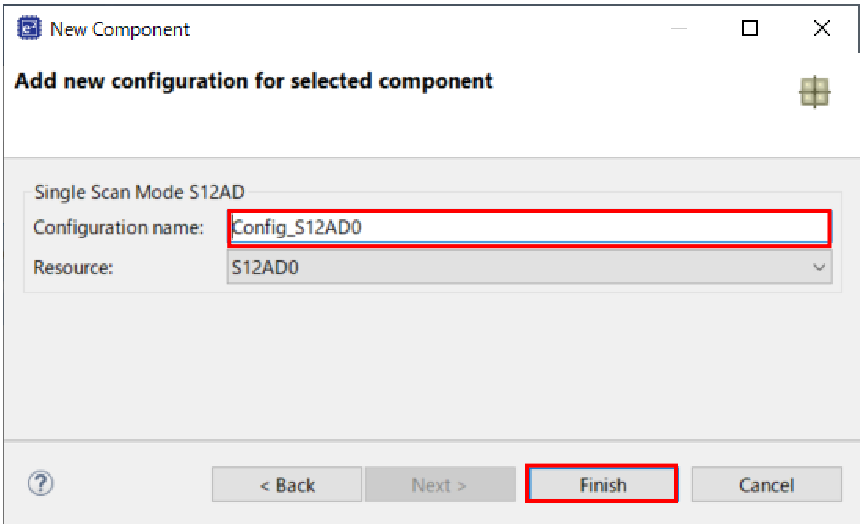


Figure 4-44 Ensure Configuration name - S12AD0

Configure S12AD0 as shown in **Figure 4-45** and **Figure 4-47** and **Figure 4-46**. 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.

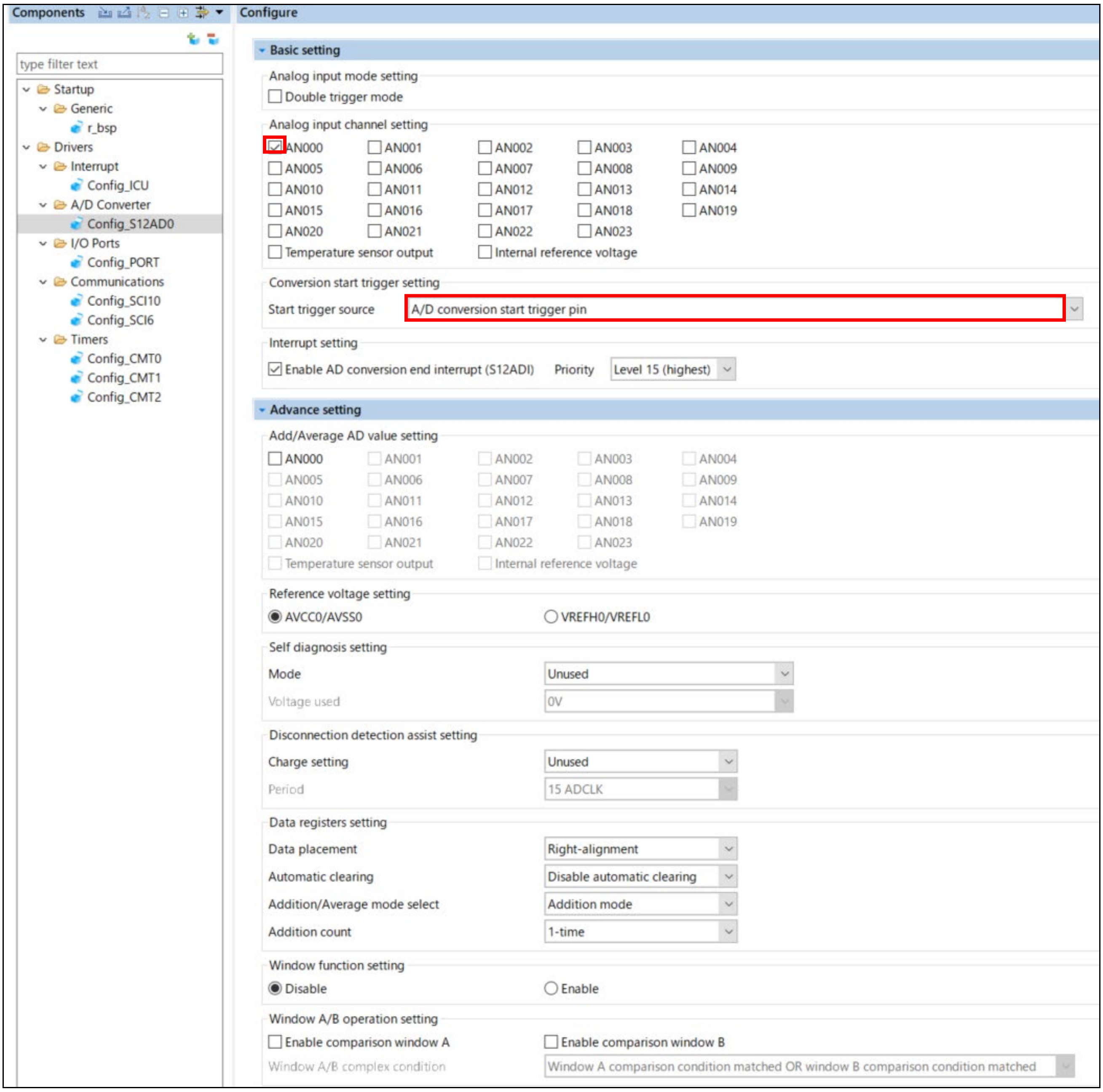


Figure 4-45 Config_S12AD0 setting (1)

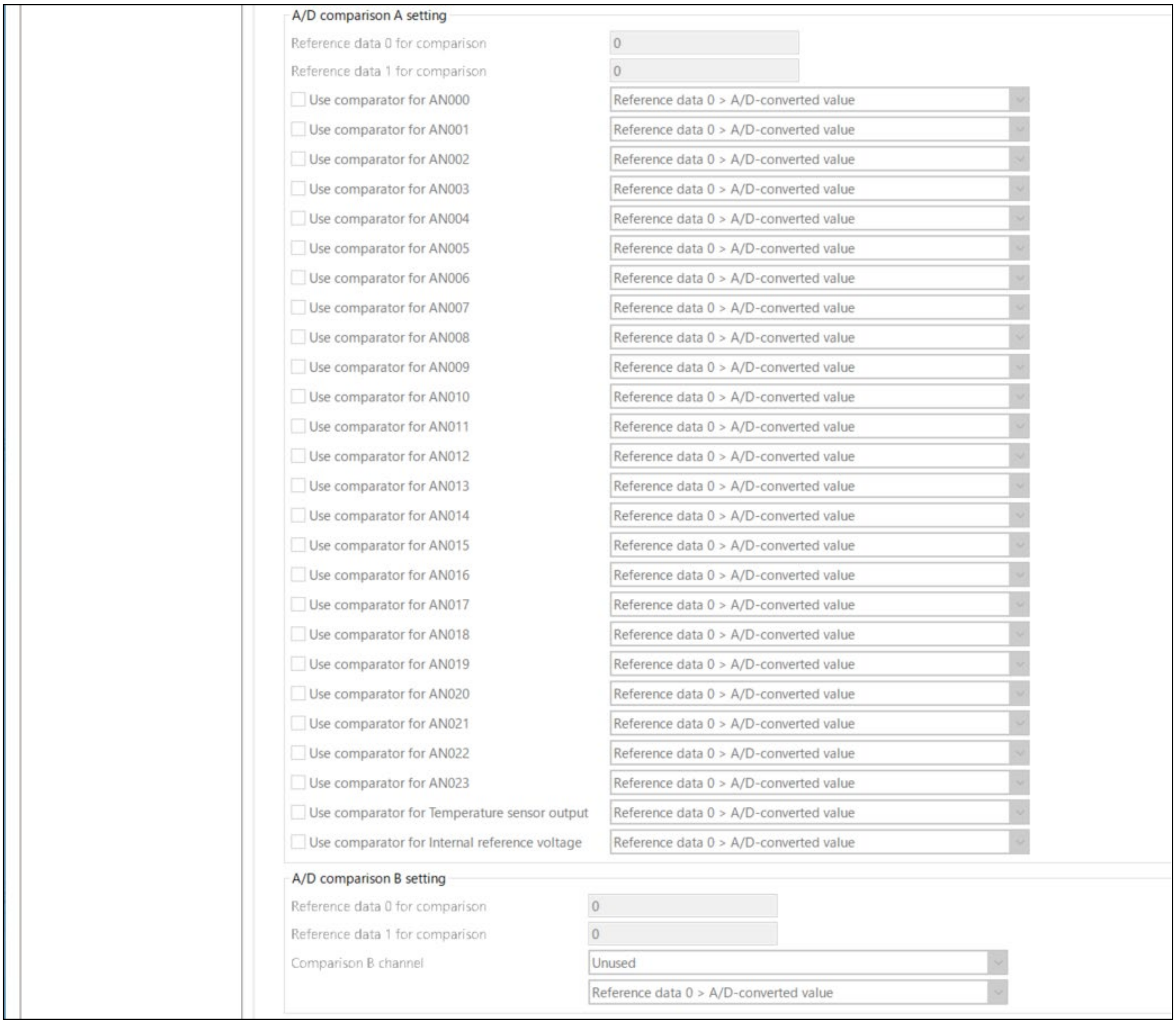


Figure 4-46 Config_S12AD0 setting (2)

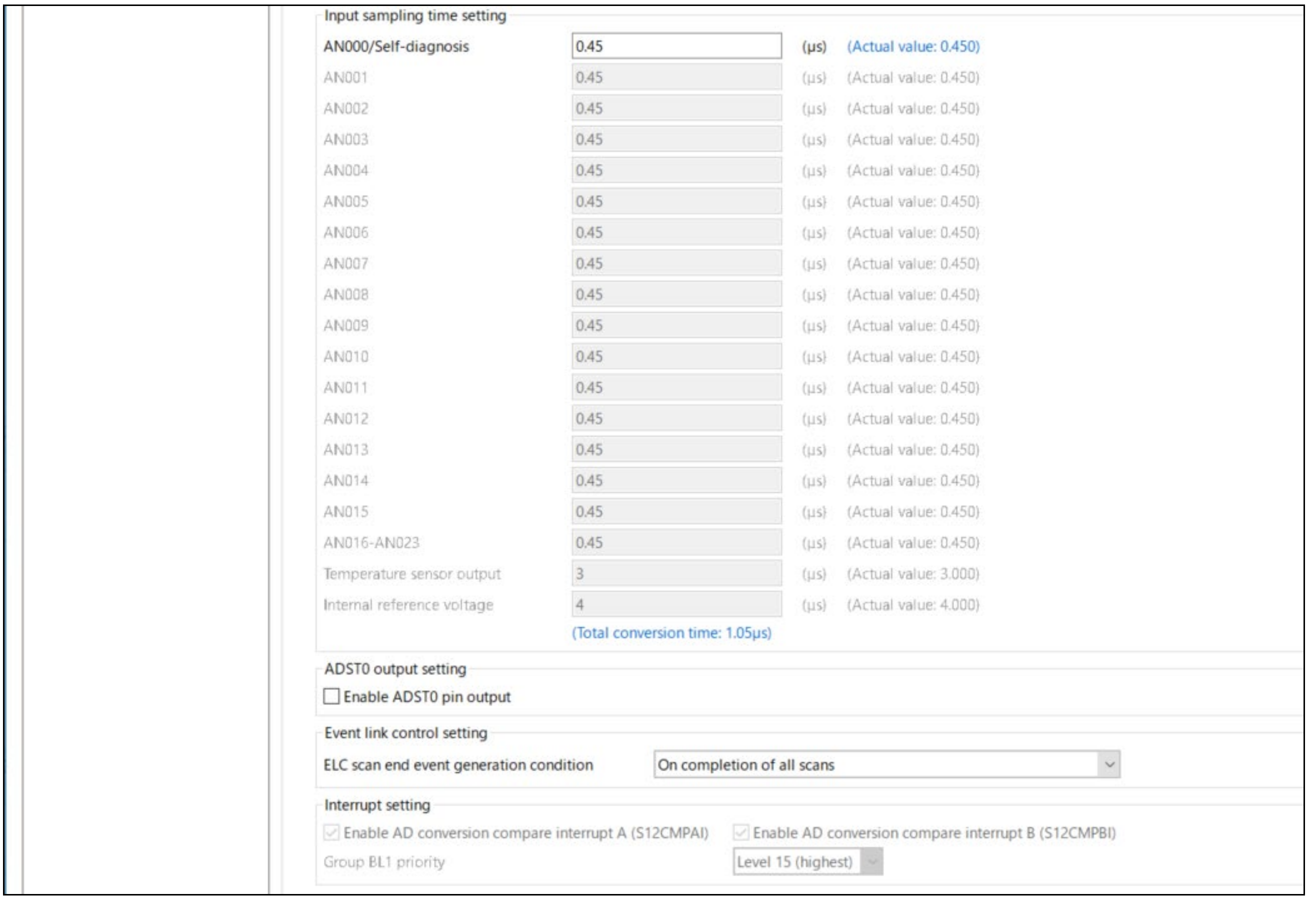


Figure 4-47 Config_S12AD0 setting (3)

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.

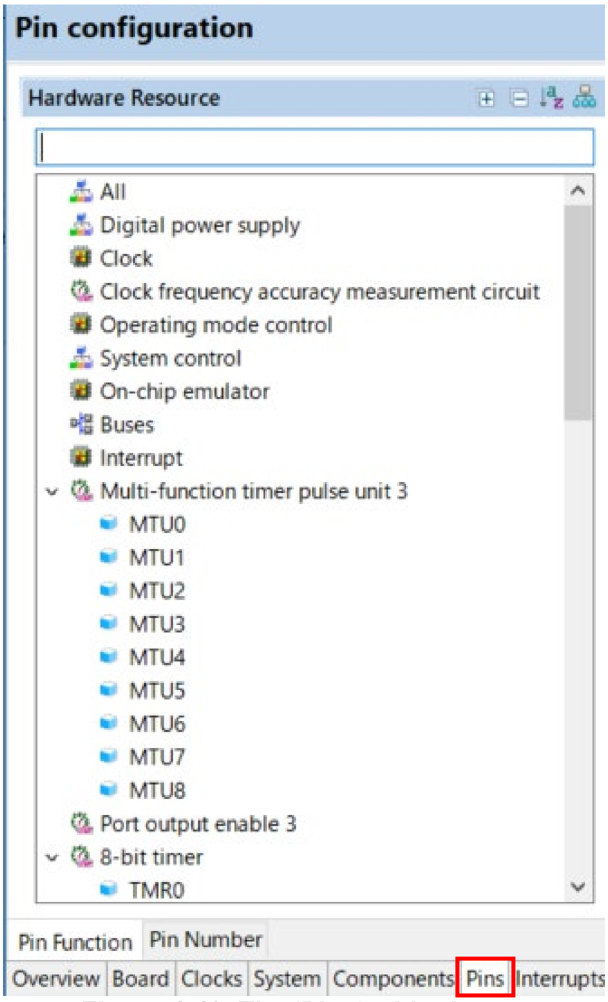


Figure 4-48 The 'Pins' tabbed page

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  to change view to show by Software Components.

Figure 4-49 Change view to show by Hardware Resource

Select the Config_ICU of Software Components. In the Pin Function list -> Assignment column, change the pin assignment IRQ9 to P91, IRQ10 to P92. Ensure the 'Enable' tick box of IRQ9 and IRQ10 are checked, as shown in **Figure 4-50**.

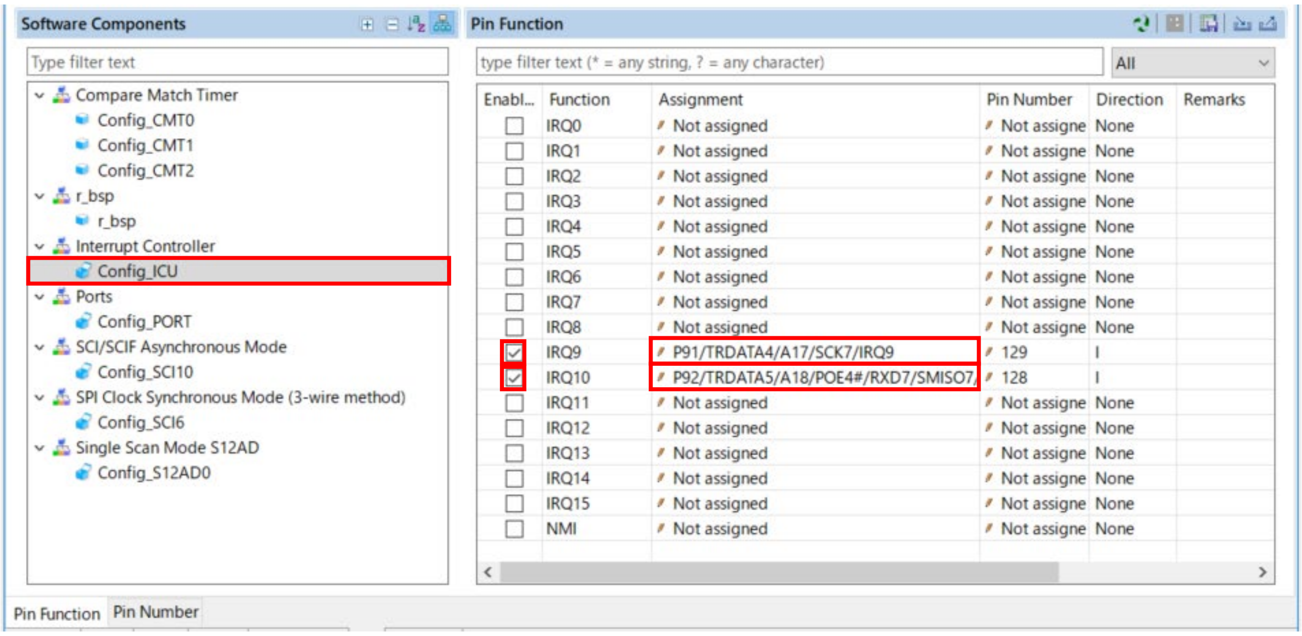


Figure 4-50 Configure pin assignment - Config_ICU

Select the Config_SCI10 of Software Components. In the Pin Function list -> Assignment column, Ensure the 'Enable' tick box of RXD10 and TXD10 are checked and Assignment column of RXD10 is P86 and TXD10 is P87 as shown in **Figure 4-51**.

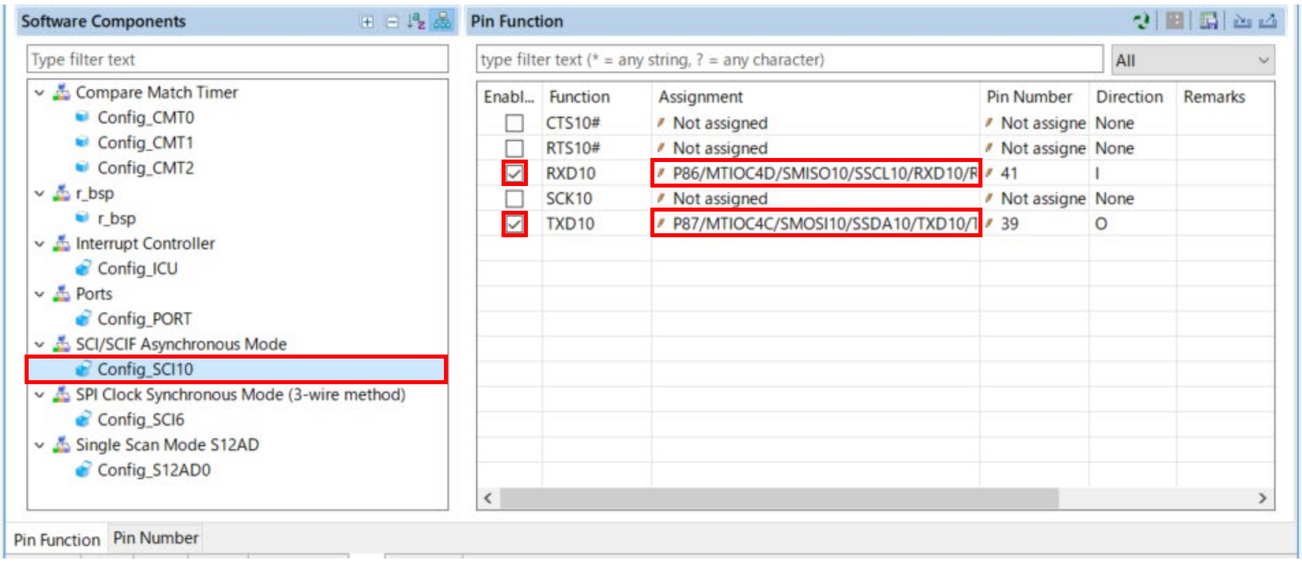


Figure 4-51 Configure pin assignment - Config_SCI10

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 P02, SMOSI6 is P00 as shown in **Figure 4-52**.

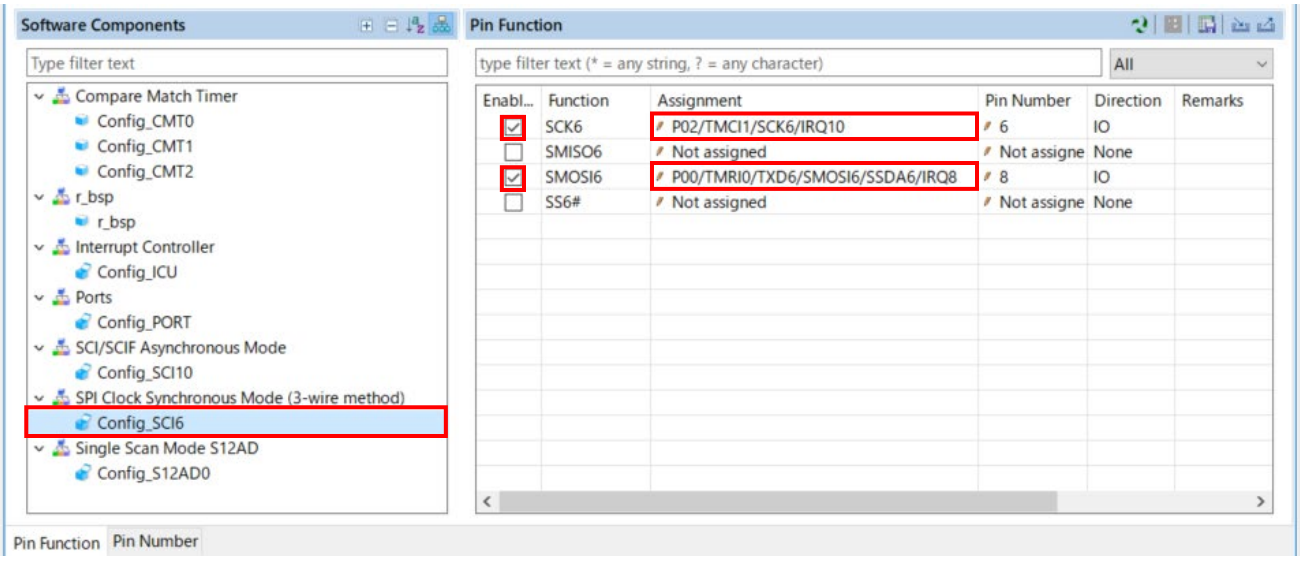


Figure 4-52 Configure pin assignment - Config_SCI6

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 P07 as shown in **Figure 4-53**.

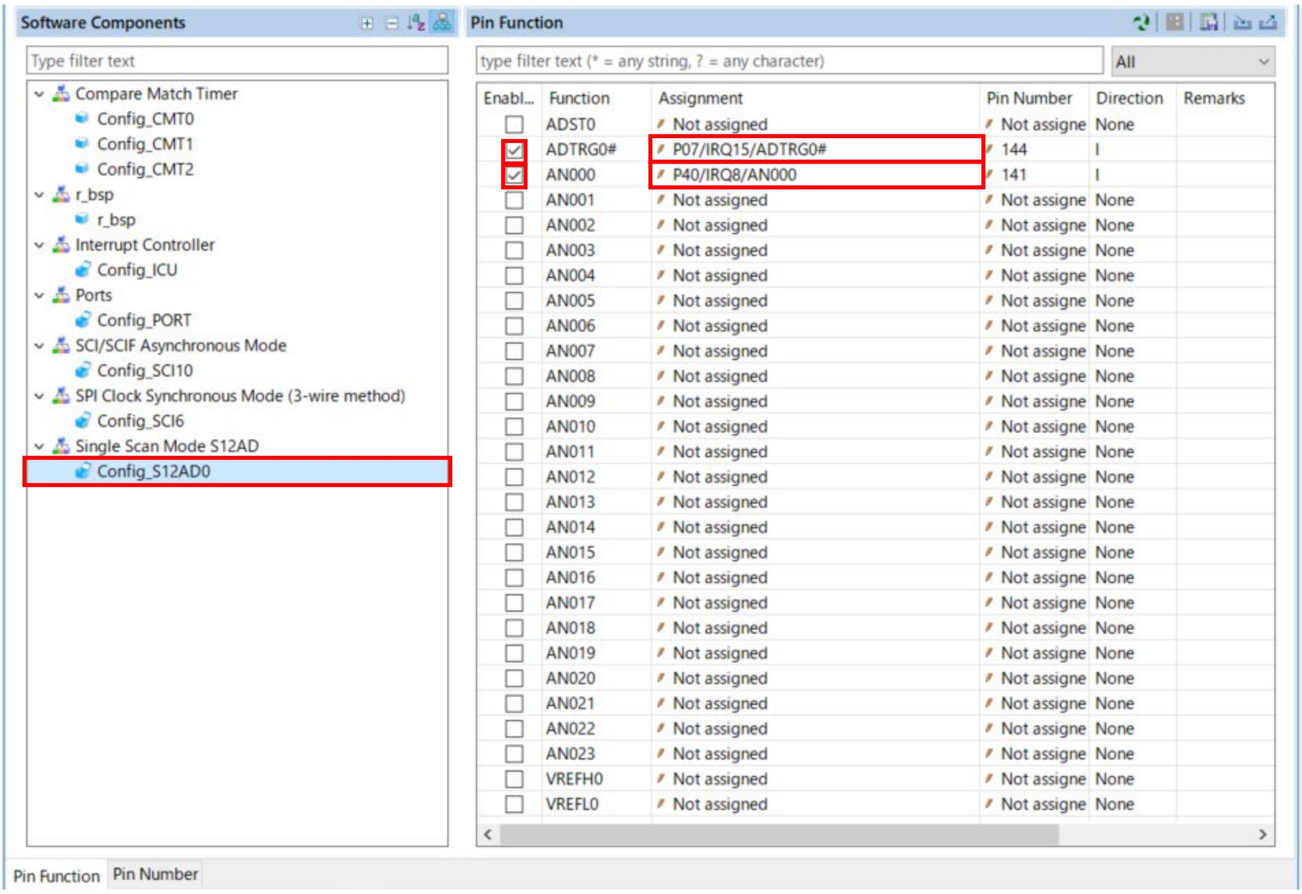


Figure 4-53 Configure pin assignment - Config_S12AD0

Peripheral function configuration is now complete. Save the project using the File -> Save, then click 'Generate Code' at location of **Figure 4-54**.

Figure 4-54 Generate Code Button

The Console pane should report 'Code generation is successful', as shown **Figure 4-55** below.

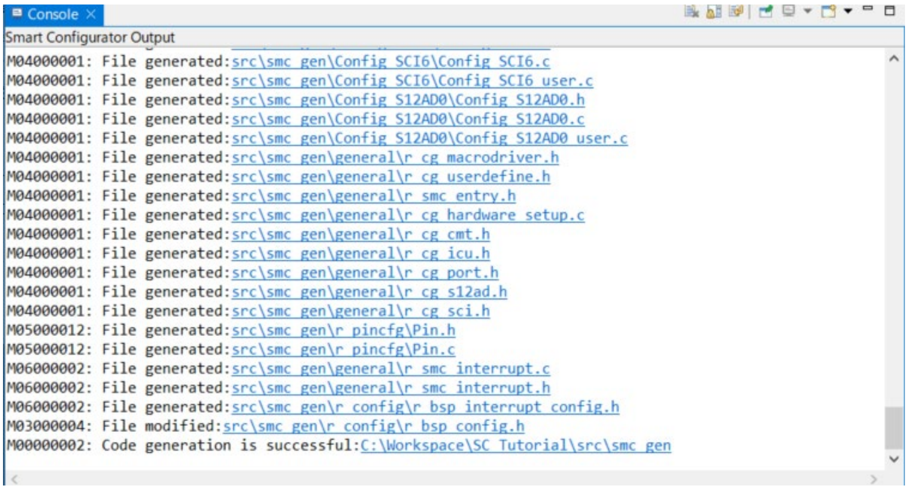


Figure 4-55 Smart Configurator console

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.

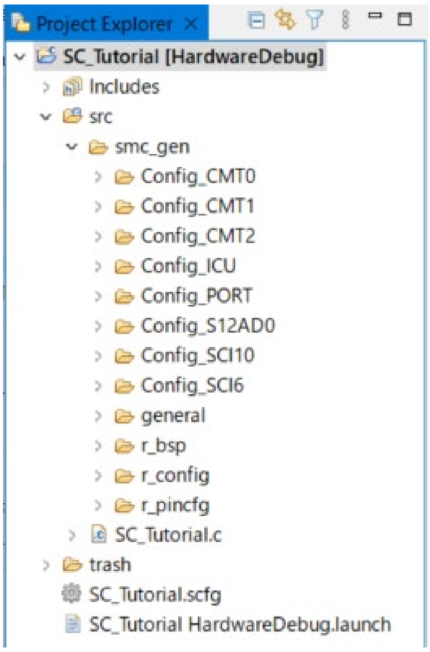


Figure 4-56 Generated folder structure

Switch back to the 'C/C++' perspective using the button on the top right of the e2 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.

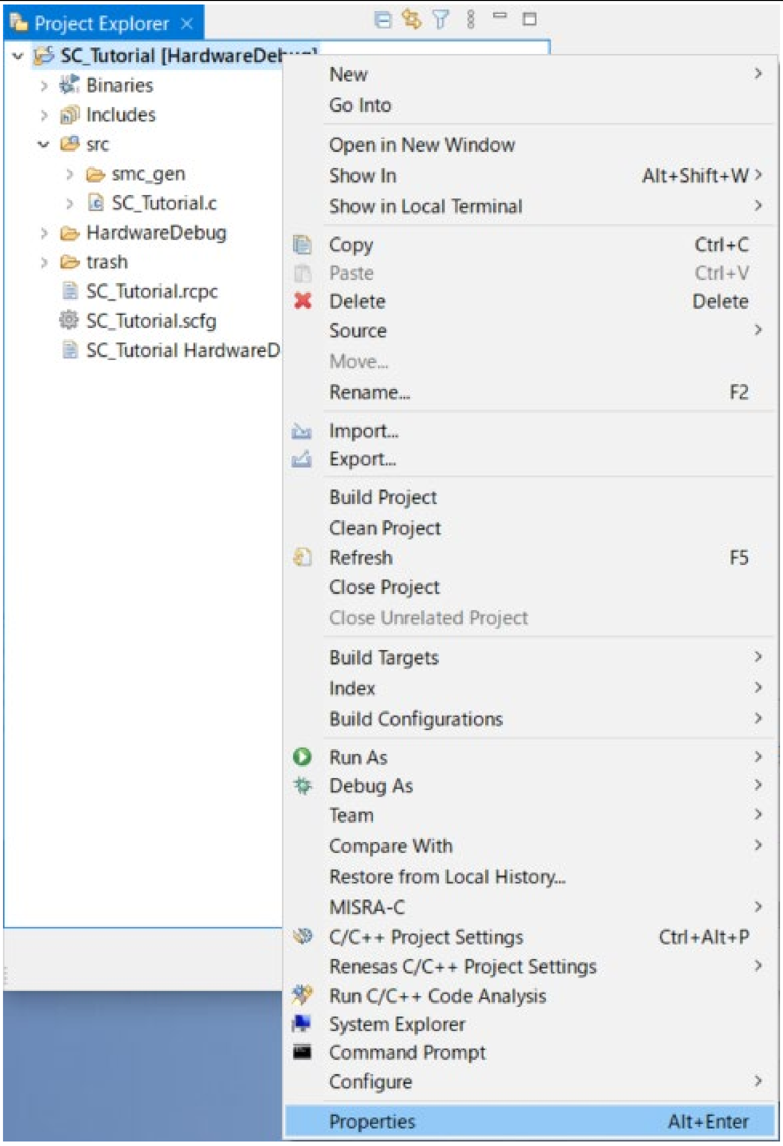
Code must be inserted into the user code area within many Smart Configurator-generated files in this project, these user code areas are delimited by comments as follows:

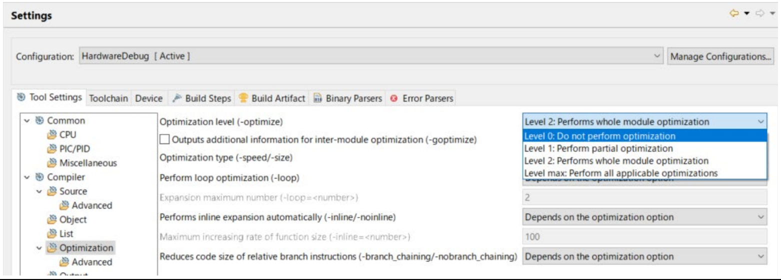
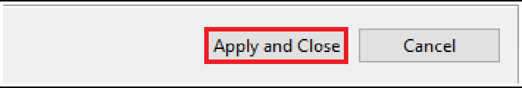
```
/* Start user code for _xxxxx_. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
```

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.	
• Press the 'Apply and Close' button to close Properties window.	

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**.

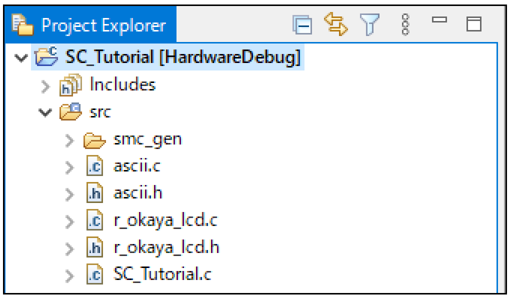


Figure 5-1 Adding files to the project

In the e² 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 e² 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 *)" RSKRX660 ");
    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 4.6.6. 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 CMT 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_CMT0\Config_CMT0.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_CMT_MsDelay(const uint16_t millisec);

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

Open the file 'Config_CMT0_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_CMT0_cmi0_interrupt function and insert the following line in the user code area:

```
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_cmi0_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_CMT_MsDelay
* Description  : Uses CMT0 to wait for a specified number of milliseconds
* Arguments    : uint16_t millisecs, number of milliseconds to wait
* Return Value : None
*******************************************************************************/
void R_CMT_MsDelay (const uint16_t millisec)
{
    uint16_t ms_count = 0;

    do
    {
        R_Config_CMT0_Start();
        while (FALSE == s_one_ms_delay_complete)
        {
            /* Wait */
        }
        R_Config_CMT0_Stop();
        s_one_ms_delay_complete = FALSE;
        ms_count++;
    } while (ms_count < millisec);
}
/*******************************************************************************
End of function R_CMT_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**.

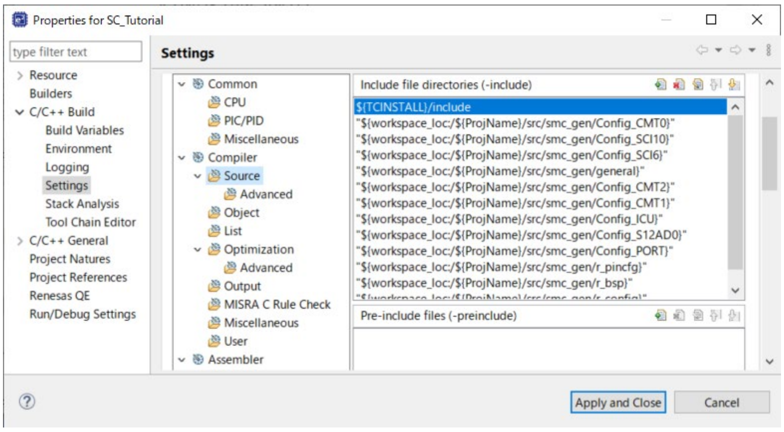


Figure 5-2 Adding additional search paths

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.

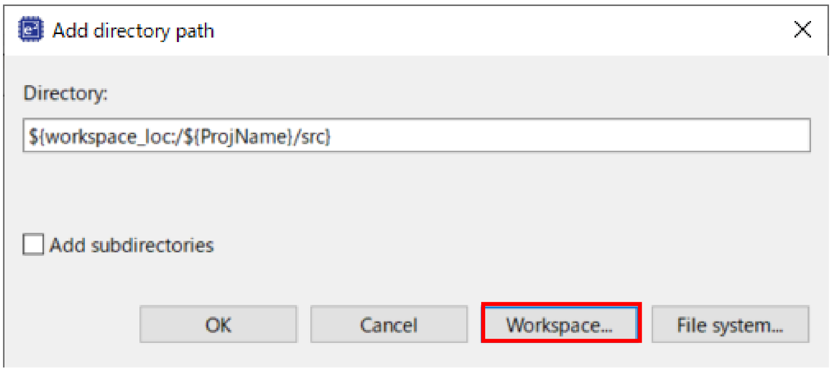


Figure 5-3 Adding workspace search path

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.

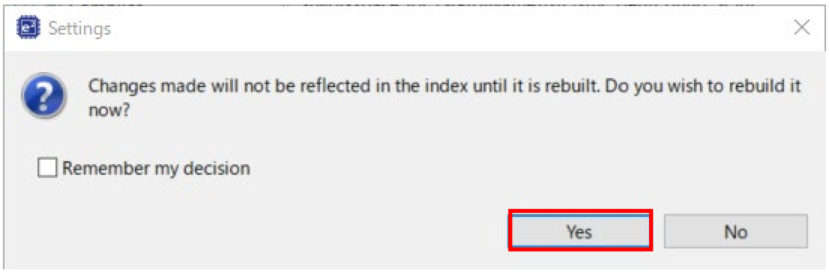


Figure 5-4 Settings dialog

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 'RSKRX660 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:

- rskrx660def.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_CMT1.h, Config_CMT1.c, Config_CMT1_user.c, Config_CMT2.h, Config_CMT2.c and Config_CMT2_user.c as described in section 4.6.2. and section 4.6.3 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_irq9_interrupt:

```
    /* Start user code for r_Config_ICU_irq9_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_irq10_interrupt:

```
    /* Start user code for r_Config_ICU_irq10_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 e² studio Project Tree, expand the 'src\smc_gen\Config_CMT1' folder and open the '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 Config_CMT1_user.c' 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 */
```

In the e² studio Project Tree, expand the 'src\smc_gen\Config_CMT2' folder and open the file 'Config_CMT2_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_CMT2_cmi2_interrupt:

```
/* Start user code for r_Config_CMT2_cmi2_interrupt. Do not edit comment generated here */

/* Stop this timer - we start it again in the de-bounce routines */
R_Config_CMT2_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 4.6.7 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 *)" RSKRX660 ");
    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 e² 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(S12ADC0, S12ADI0) = 0U;
    IEN(S12ADC0, 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(S12ADC0, S12ADI0) = 0U;
    IR(S12ADC0, 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_SCI10_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_SCI10' folder and open the file 'Config_SCI10.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_SCI10_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_SCI10_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_SCI10_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_SCI10_callback_transmitend function:

```
static void r_Config_SCI10_callback_transmitend (void)
{
    /* Start user code for r_Config_SCI10_callback_transmitend. Do not edit comment generated here */

    s_SCI10_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_SCI10_callback_receiveend function:

```
static void r_Config_SCI10_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI10_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 SCI10 receive buffer and callback function again */
    R_Config_SCI10_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_SCI10_AsyncTransmit
* Description  : This function sends SCI10 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_SCI10_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_SCI10_txdone = FALSE;

    /* Send the data using the API */
    status = R_Config_SCI10_Serial_Send(tx_buf, tx_num);

    /* Wait for the transmit end flag */
    while (FALSE == s_SCI10_txdone)
    {
        /* Wait */
    }
    return (status);
}

/******************************************************************************
* End of function R_SCI10_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_SCI10.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 *)" RSKRX660 ");
    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 SCI10 receive buffer and callback function */
    R_Config_SCI10_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI10 operations */
    R_Config_SCI10_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 SCI10 (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 SCI10.

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_SCI10.h"
#include "rskrx660def.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 *)" RSKRX660 ");
    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 SCI10 receive buffer and callback function */
    R_Config_SCI10_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI10 operations */
    R_Config_SCI10_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'.

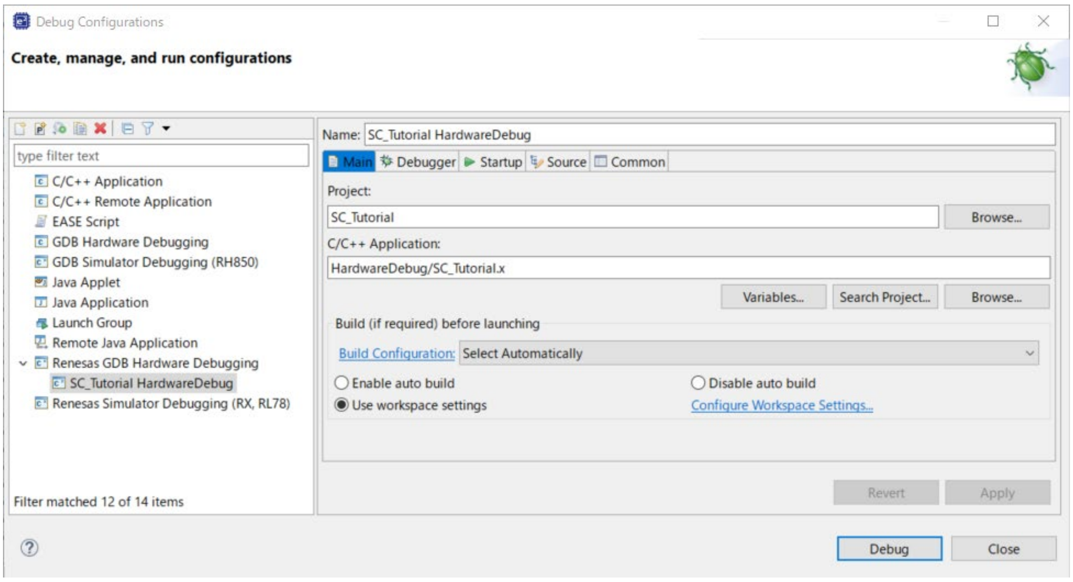


Figure 6-1 Debug Configurations

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 RSKRX660 set the EXTAL Frequency: 24.0000, Operating Frequency: 120.000).
For more information on powering the RSKRX660 please refer to the User's Manual.

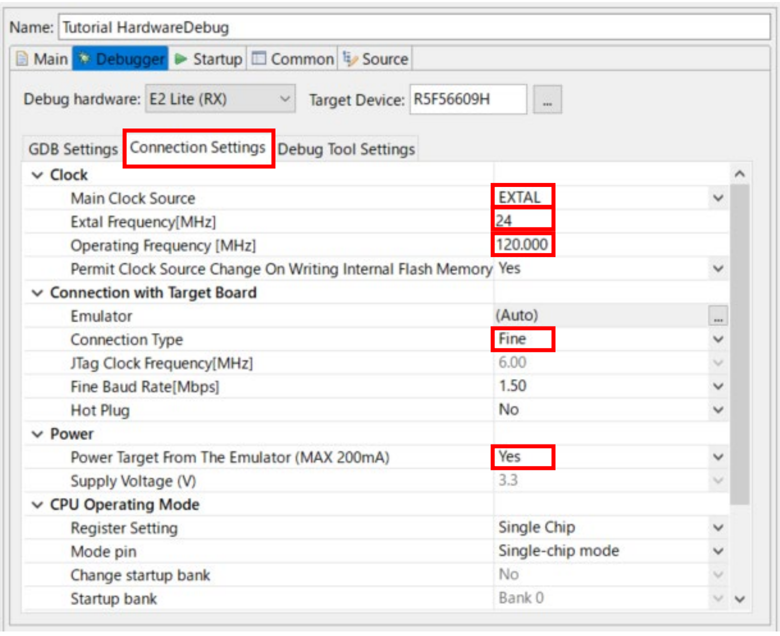


Figure 6-2 Connection Settings

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.

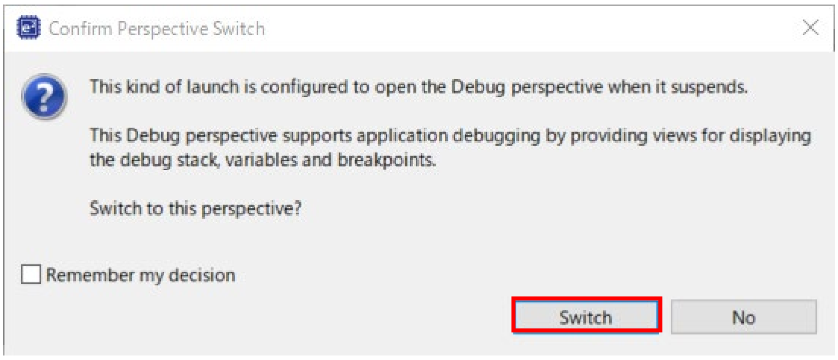


Figure 6-3 Perspective Switch Dialog

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**.

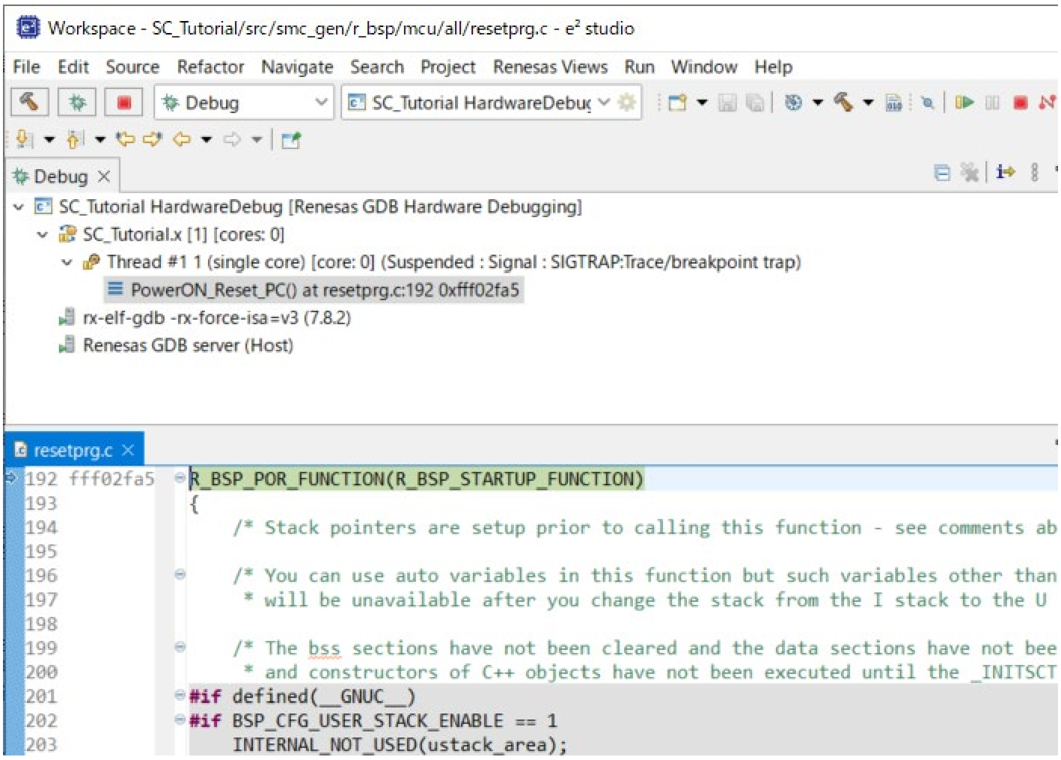
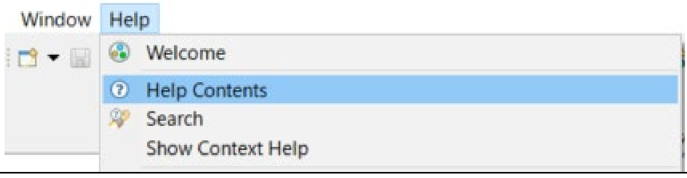


Figure 6-4 Debugger start up screen

For more information on the e² 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.	

For information about the RX660 group microcontroller refer to 'RX660 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/rskrx660

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

REVISION HISTORY	RX660 Group Renesas Starter Kit for RX660 Smart Configurator Tutorial Manual For e^2 studio

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

RX660 Group
Renesas Starter Kit for RX660
Smart Configurator Tutorial Manual For e^2 studio

Publication Date: Rev. 1.00 Jan.17.2022

Published by: Renesas Electronics Corporation

RX660 Group

Renesas Electronics Corporation

R20UT5023EG0100