User's Manual

32

RX140 Group

Renesas Starter Kit for RX140
Smart Configurator Tutorial Manual
For CS+

RENESAS 32-Bit MCU
RX Family / RX100 Series

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

Trademarks

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:

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 Application Leading Tool (Smart Configurator) for RX together with the CS+ 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 CS+, 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 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	R20UT5027EG
Quick Start Guide	Provides simple instructions to setup the RSK and run the first sample.	Renesas Starter Kit for RX140 Quick Start Guide	R20UT5028EG
Smart Configurator Tutorial Manual	Provides a guide to code generation and importing into the CS+ IDE.	Renesas Starter Kit for RX140 Smart Configurator Tutorial Manual	R20UT5029EG
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

Table of Contents

1. Overview

1.1 Purpose

This RSK is an evaluation tool for Renesas microcontrollers. This manual describes how to use the CS+ IDE Smart Configurator to create a working project for the RSK platform.

1.2 Features

This RSK provides an evaluation of the following features:

- Project Creation with CS+
- Code generation using the Smart Configurator.
- 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 for the RX family together with the CS+ IDE to create a working project for the RSK platform. The tutorials help explain the following:

- Project generation using the CS+
- Detailed use of the Smart Configurator for CS+
- Integration with custom code
- Building the project CS+

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

- 'DefaultBuild' is a project with debug support and optimisation level set to two.
- 'Debug' 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 no 'Outputs debugging information' options 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 CS+ debugger, compiler toolchains or the E2 emulator Lite. Please refer to the relevant user manuals for more in-depth information.

3. Project Creation with CS+

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 project and debug files.

3.2 Creating the Project

To use the program, start CS+:

Windows™ 8.1: From Apps View ⓥ, click 'CS+ for CC (RL78,RX,RH850)' icon
Windows™ 10 / 11: Start Menu > All Apps > Renesas Electronics CS+ > CS+ for CC (RL78,RX,RH850)

<table>
<tr>
<td>

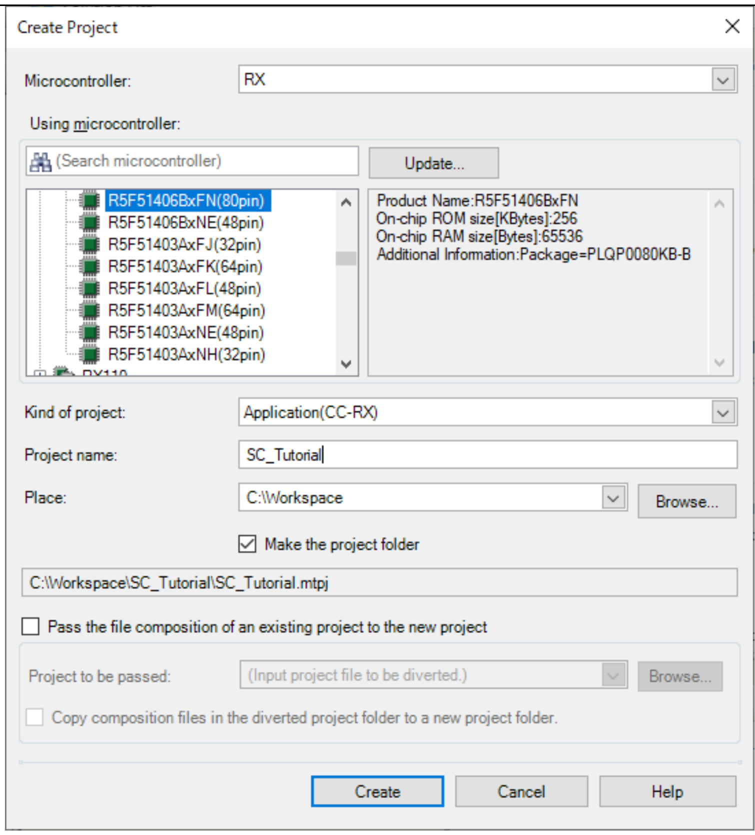
- CS+ will show the Start Page. Use the 'GO' button to Create a New Project.

</td>
<td>


</td>
</tr>
<tr>
<td>

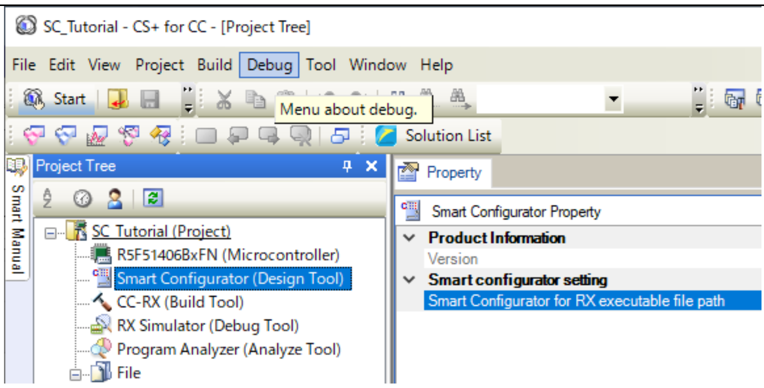
- In the 'Create Project' dialog, select 'RX' from the 'Microcontroller' pull-down.
- In the 'Using Microcontroller' list control, scroll down to 'RX140' and expand the tree control by clicking '+'. Select 'R5F514006BxFN (80pin)'.
- Ensure that in the 'Kind of project' pull-down, 'Application(CC-RX)' is selected.
- Choose an appropriate name and location for the project, then click 'Create'.

  Note: this tutorial assumes the project is named and located at the place shown opposite.
- If the folder entered cannot be found a 'Question' dialog will be displayed; click 'Yes'.

</td>
<td>


</td>
</tr>
<tr>
<td>

- CS+ will create the blank project with the standard project tree. A 'Smart Configurator' node may also be shown, if previously enabled.

</td>
<td>


</td>
</tr>
</table>

4. Smart Configurator Using the CS+

4.1 Introduction

The Smart Configurator for the RX140 has been used to generate the sample code discussed in this document. Smart Configurator for CS+ is a 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.

By following the steps detailed in this tutorial, the user will generate a CS+ project called SC_Tutorial. A fully completed Tutorial project is contained in the RSK Web Installer (https://www.renesas.com/rskrx140/install/cs) and may be imported into CS+ 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 CS+.

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.

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 moddules 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: CS+.
You can download the latest document from: https://www.renesas.com/smart-configurator.

Smart Configurator will start up by double clicking on “Smart Configurator (Design Tool)” in the project tree.
RTOS configuration dialog will only appear once, on the first instance that smart configurator is opened. It will appear as illustrated below in **Figure 4-1**. This project does not use RTOS, so press the ‘Finish’ button.

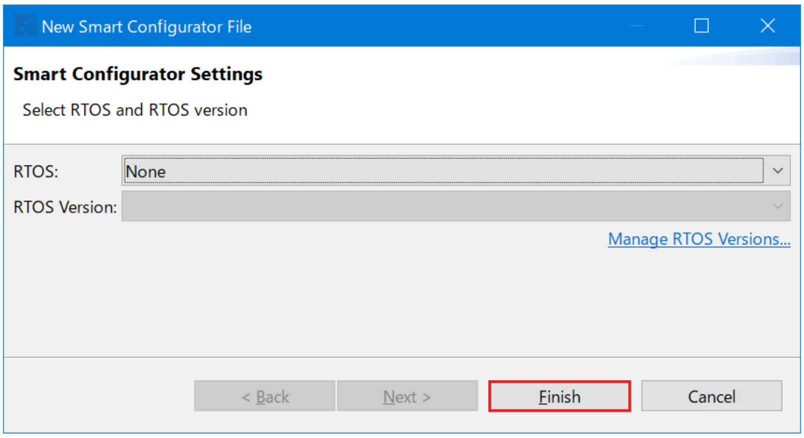


Figure 4-1 RTOS setting dialog

The Smart Configurator initial view is displayed as illustrated in **Figure 4-2**.

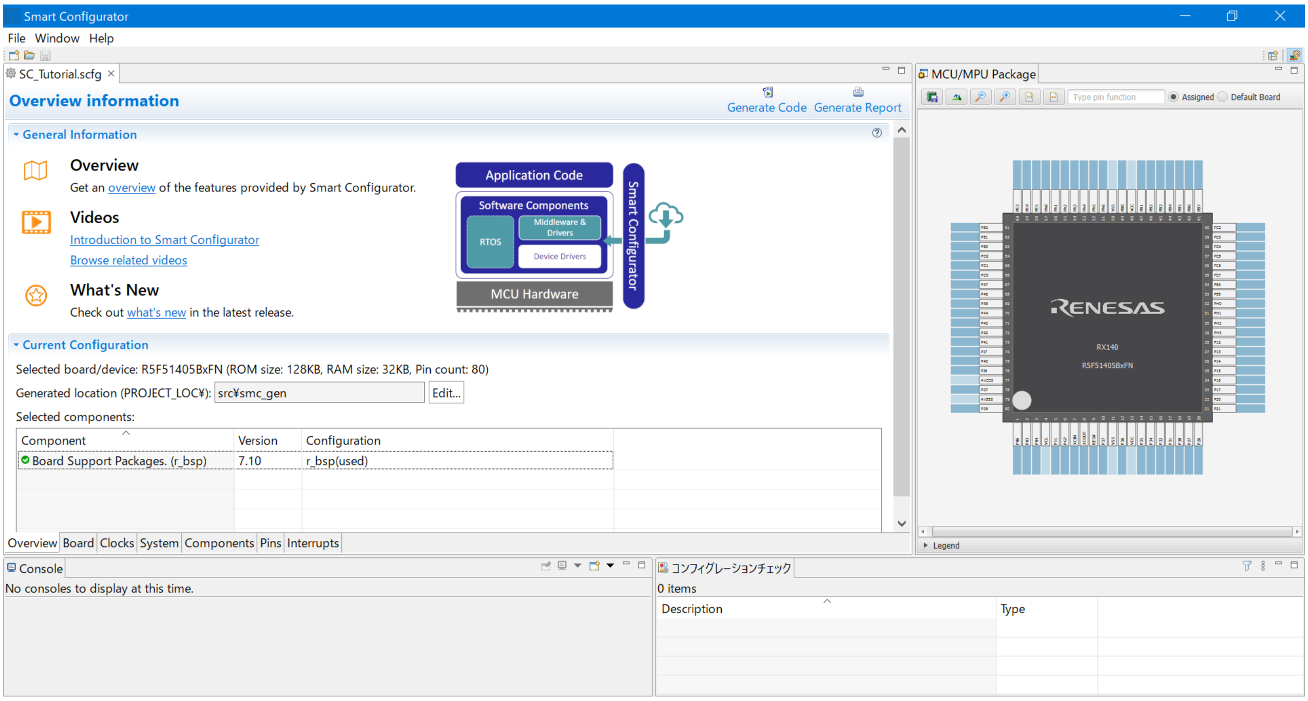


Figure 4-2 Overview page

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 CS+ 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 4-3**.

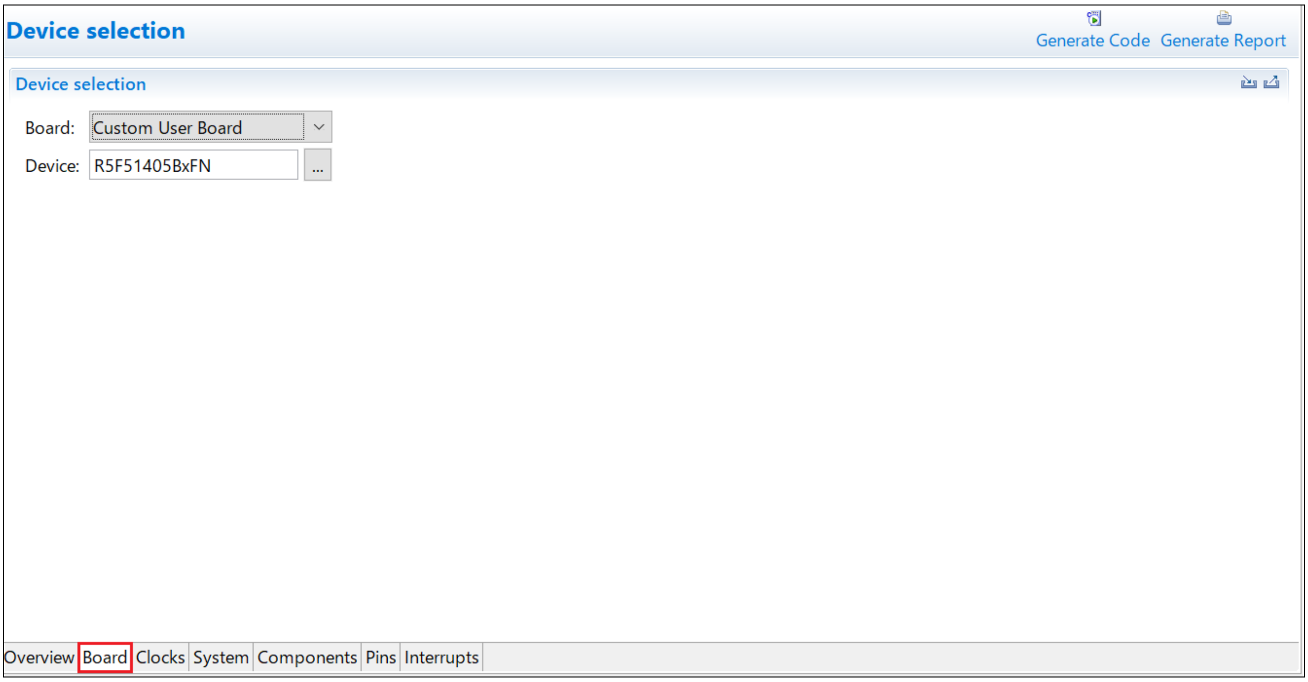


Figure 4-3 The 'Board' tabbed page

4.3.1 Board configuration

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

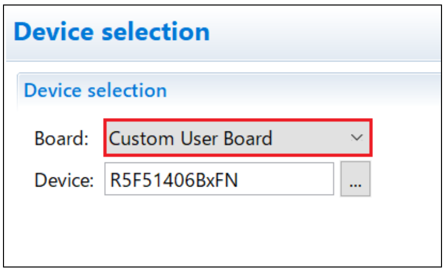


Figure 4-4 Select board

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 to ‘r_bsp_config.h’ file in ‘Smart Configurator\r_config’ of project tree.

4.4.1 Clocks configuration

Figure 4-5 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 4-5**.

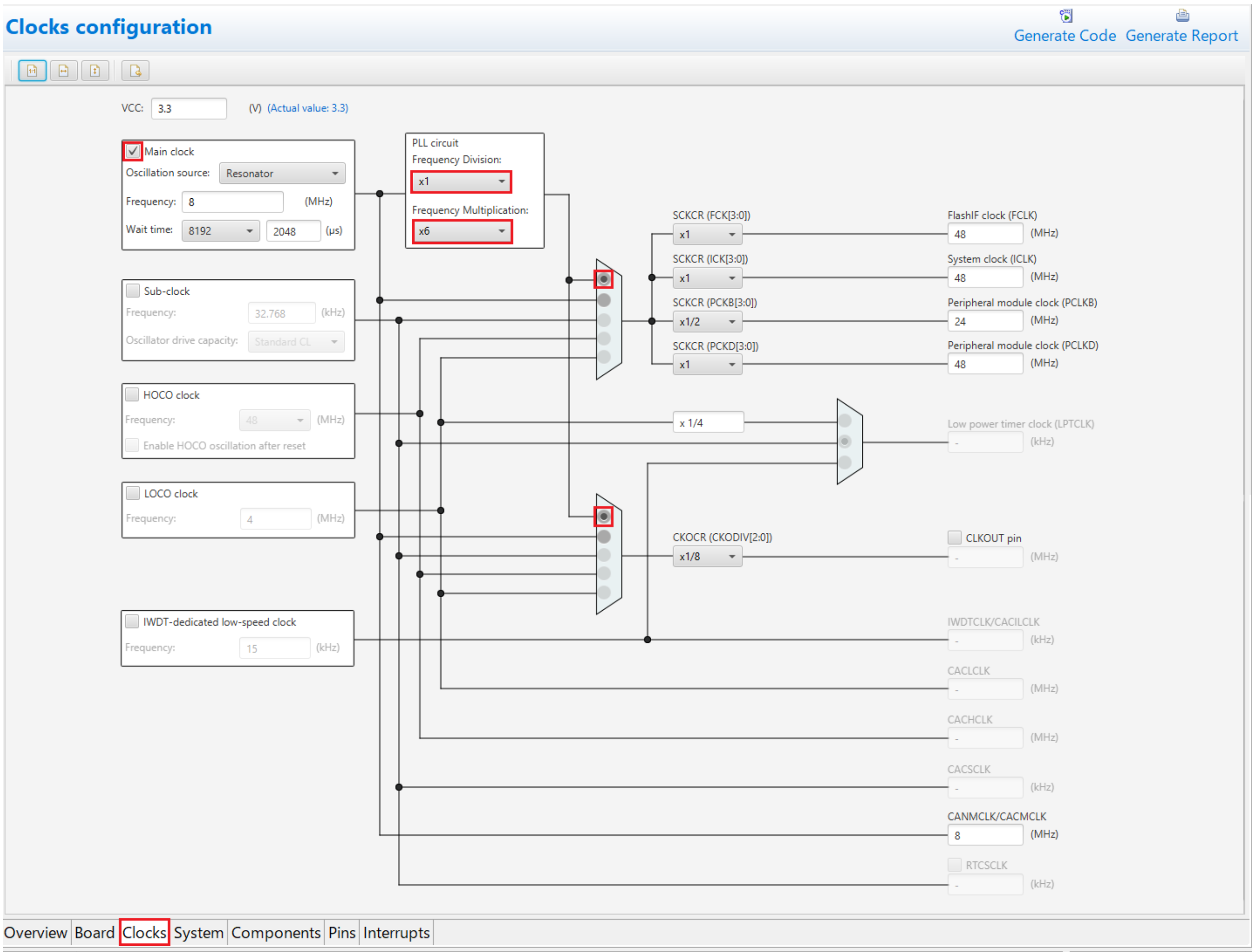


Figure 4-5 The ‘Clocks’ tabbed page

4.5 The ‘System’ tabbed page

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

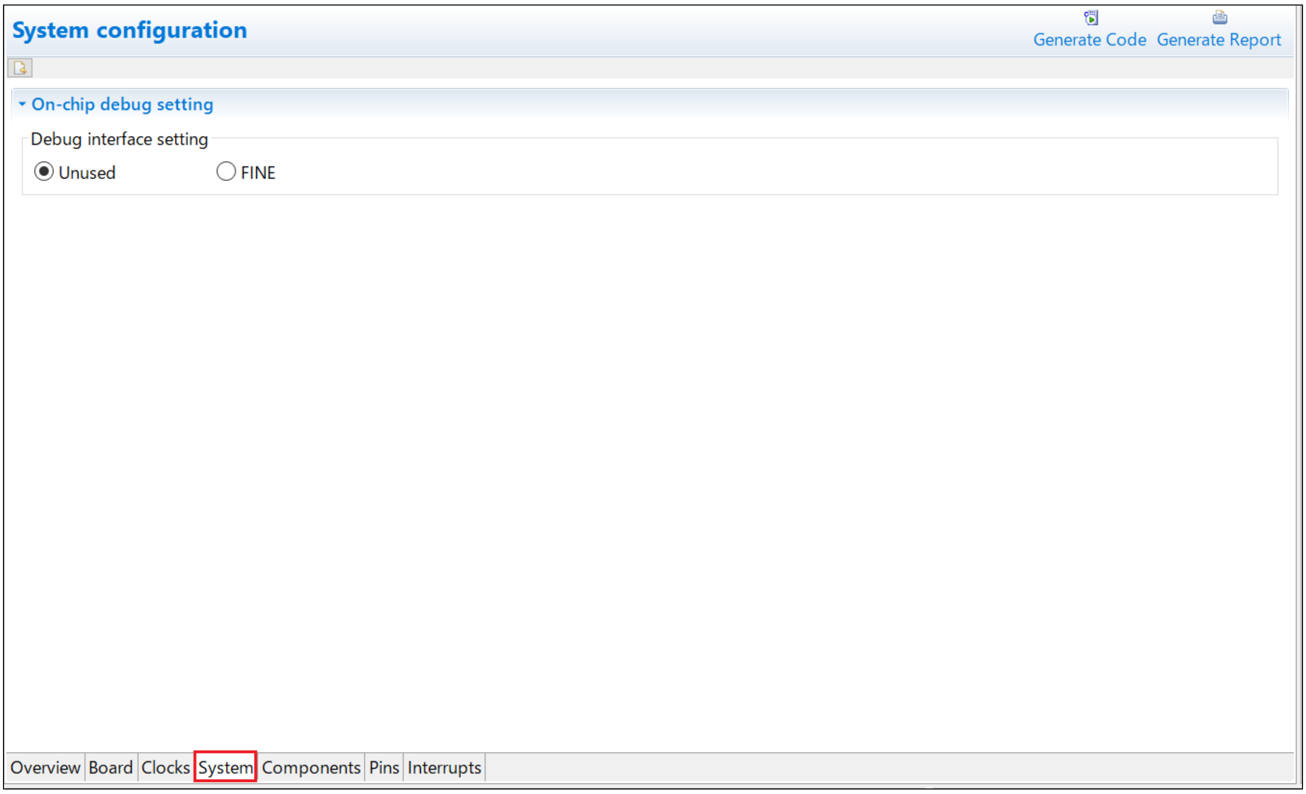


Figure 4-6 The ‘System’ tabbed page

4.5.1 On-chip debug setting

The On-chip debug settings set the interface used for debugging. For the RSKRX140 CPU board, select FINE as shown in **Figure 4-7**.

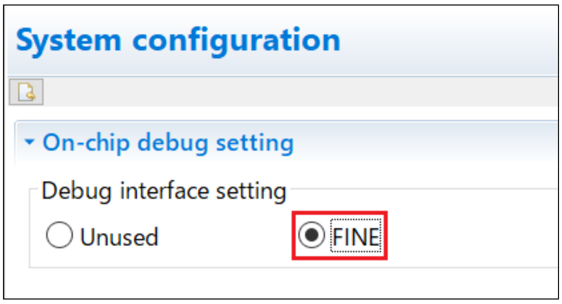


Figure 4-7 Debug interface setting

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.

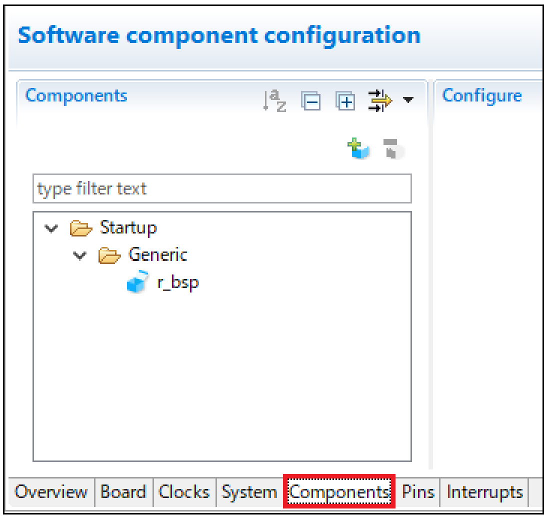


Figure 4-8 Components page

4.6.1 Add a software component into the project

Smart Configurator supports four types of software components: Startup, Drivers, Middleware and Application. 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.

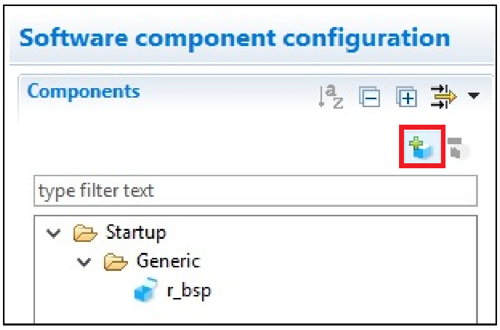


Figure 4-9 Add a Software component (1)

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

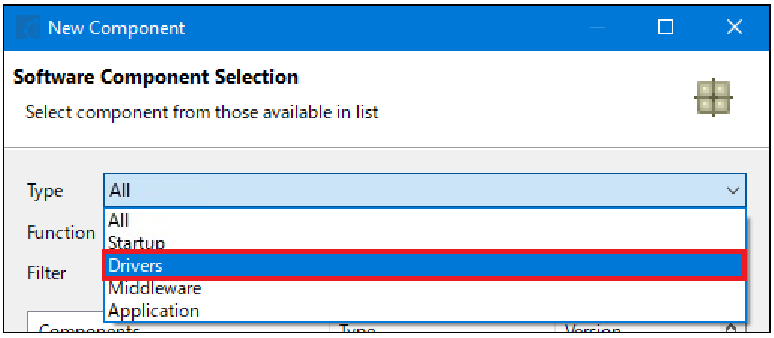


Figure 4-10 Add a Software component (2)

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 4-11** below then click ‘Next’.

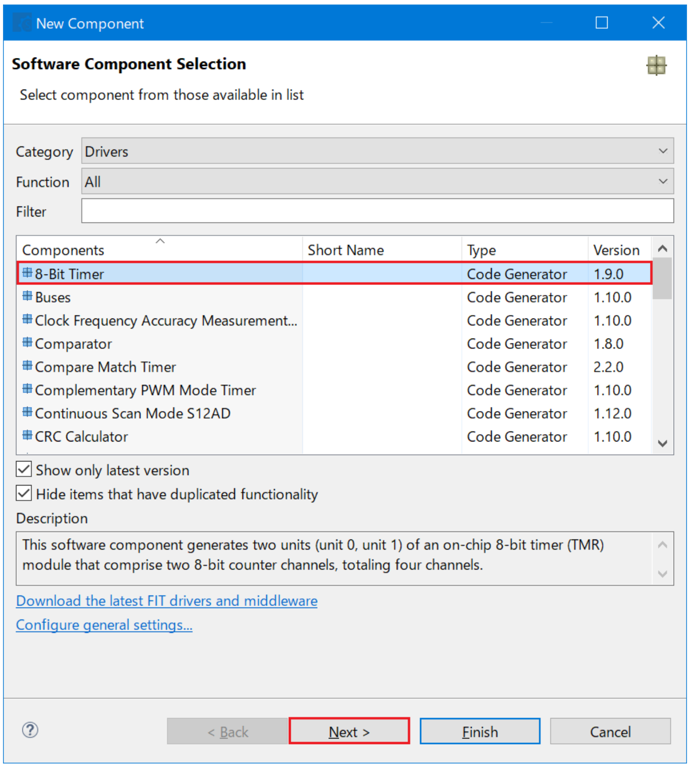


Figure 4-11 Select 8-Bit Timer

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

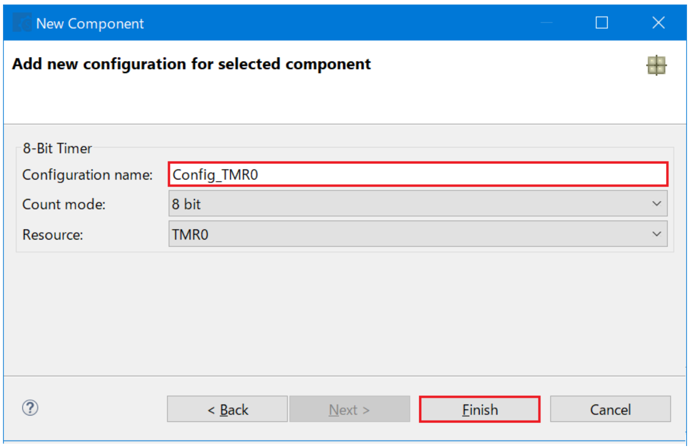


Figure 4-12 Ensure Configuration name - TMR0

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.

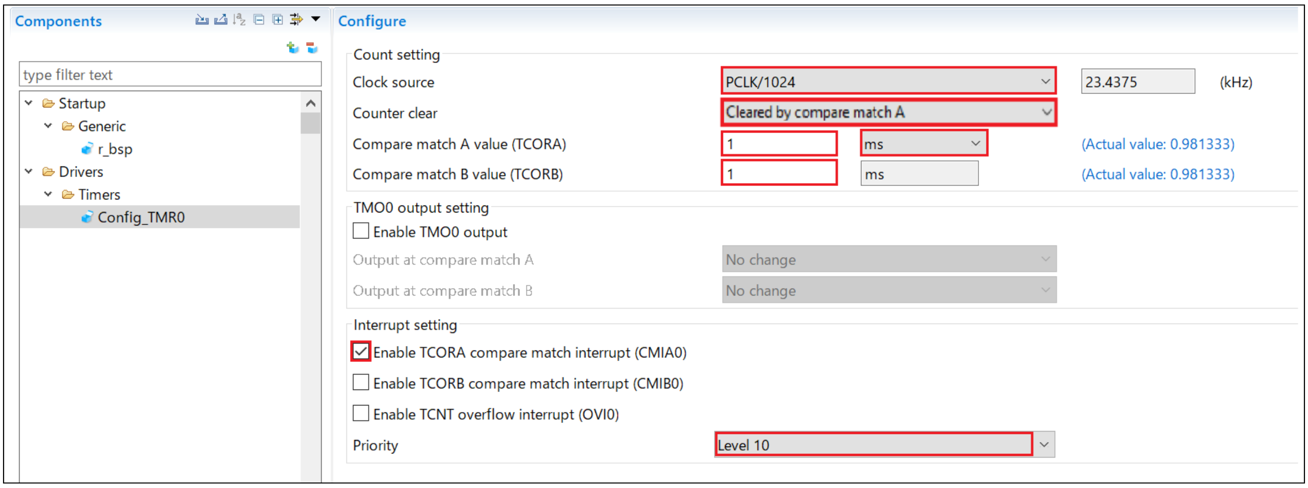


Figure 4-13 Config_TMR0 setting

4.6.3 Compare Match Timer

CMT0 and CMT1 will be used as timers in de-bouncing of switch interrupts.

Click the ‘Add component’ icon. In ‘Software Component Selection’ dialog -> Type, select ‘Drivers’. Select ‘Compare Match Timer’ as shown in **Figure 4-14** below then click ‘Next’.

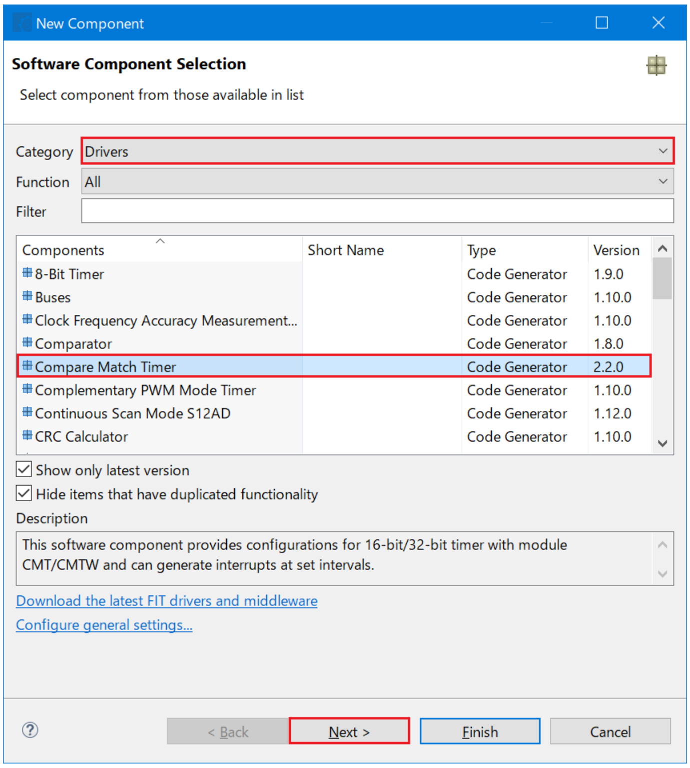


Figure 4-14 Select Compare Match Timer

Ensure that the 'Configuration name' updates to 'Config_CMT0' as shown in **Figure 4-15** below then click 'Finish'.

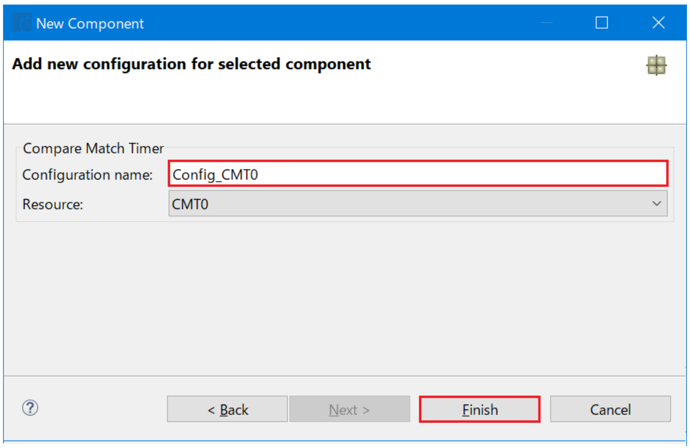


Figure 4-15 Ensure Configuration name – CMT0

Navigate to the 'Config_CMT0' and configure CMT0 as shown in **Figure 4-16**. This timer is configured to generate a high priority interrupt after 20ms. This timer is used as our short switch de-bounce timer later in this tutorial.

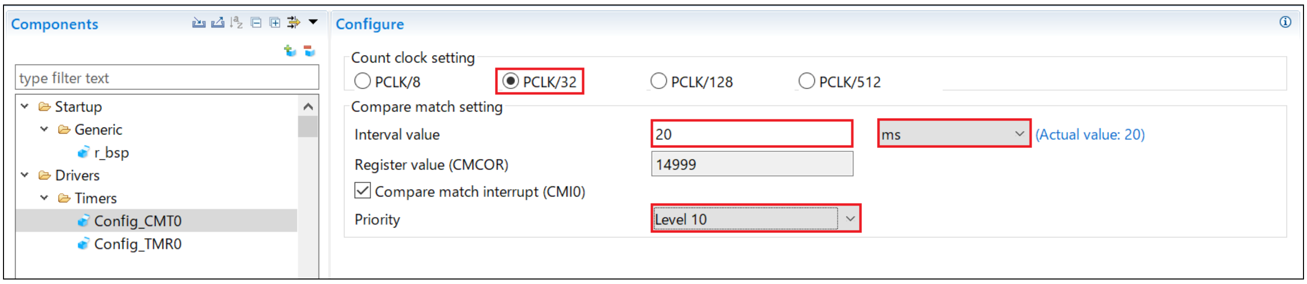


Figure 4-16 Config_CMT0 setting

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 4-17** below.

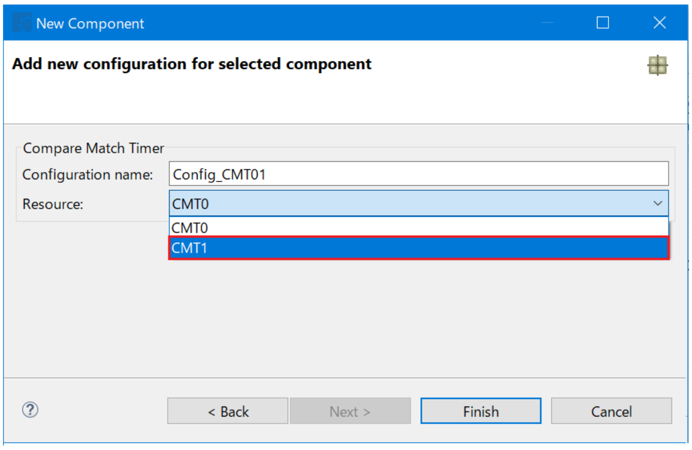


Figure 4-17 Select Resource – CMT1

Ensure that the 'Configuration name' updates to 'Config_CMT1' as shown in **Figure 4-18** below then click 'Finish'.

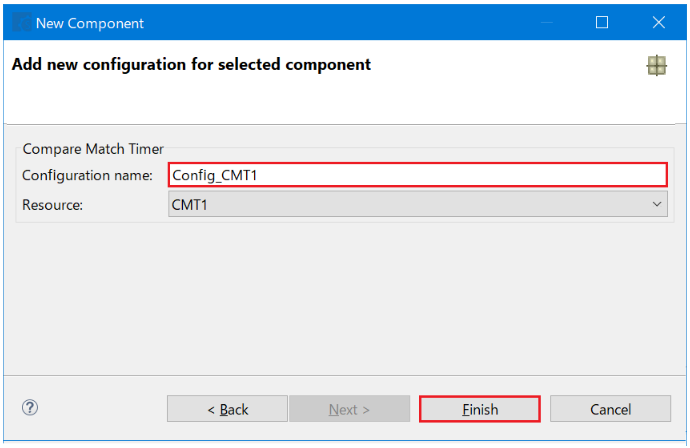


Figure 4-18 Ensure Configuration name – CMT1

Navigate to the 'Config_CMT1' and configure CMT1 as shown in **Figure 4-19**. This timer is configured to generate a high priority interrupt after 200ms. This timer is used as our long switch de-bounce timer later in this tutorial.

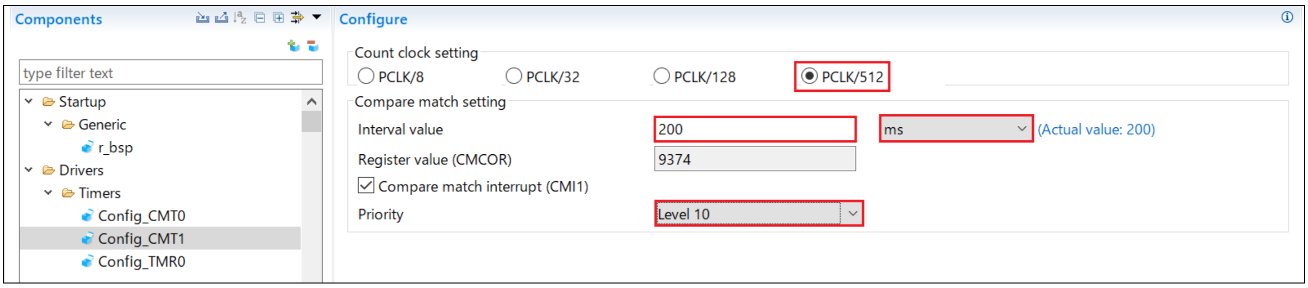


Figure 4-19 Config_CMT1 setting

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 IRQ4(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 4-20** then click 'Next'.

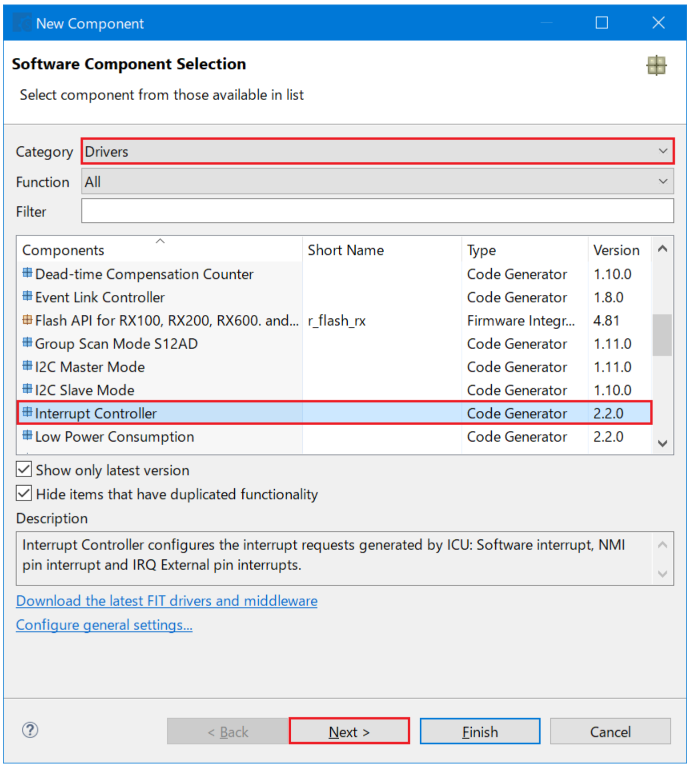


Figure 4-20 Select Interrupt Controller

In 'Add new configuration for selected component' dialog -> Resource, select 'ICU' as shown in **Figure 4-21** below then click 'Finish'.

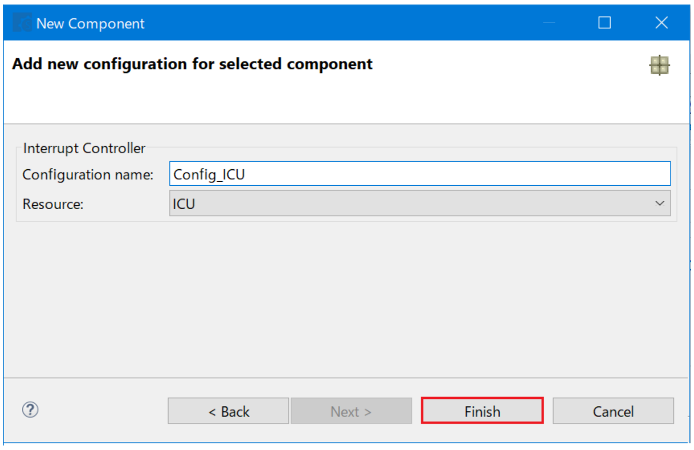


Figure 4-21 Select resource – ICU

Navigate to the 'Config_ICU', configure these two interrupts as falling edge triggered as shown in **Figure 4-22** below.

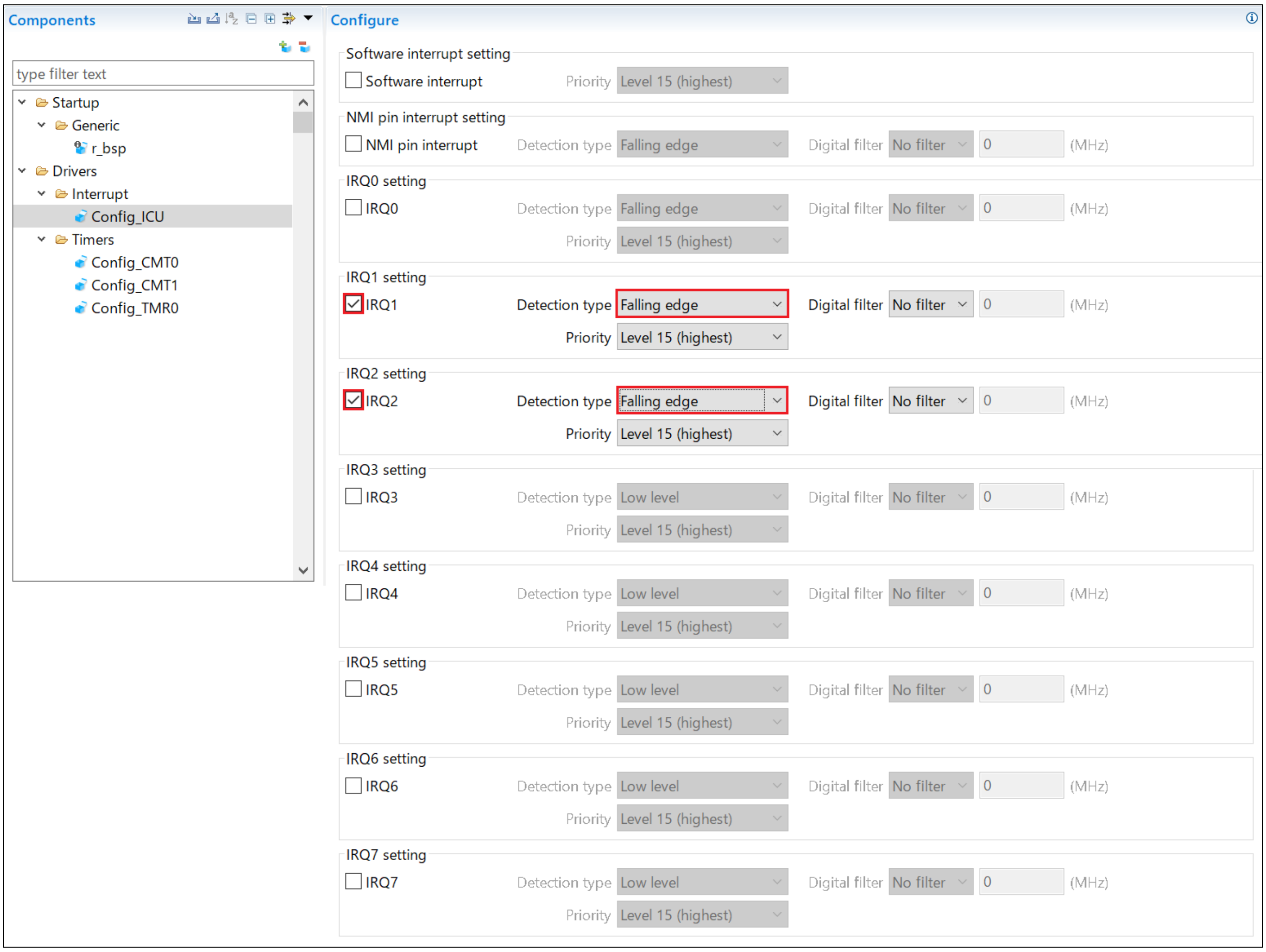


Figure 4-22 Config_ICU setting

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 4-23** then click 'Next'.

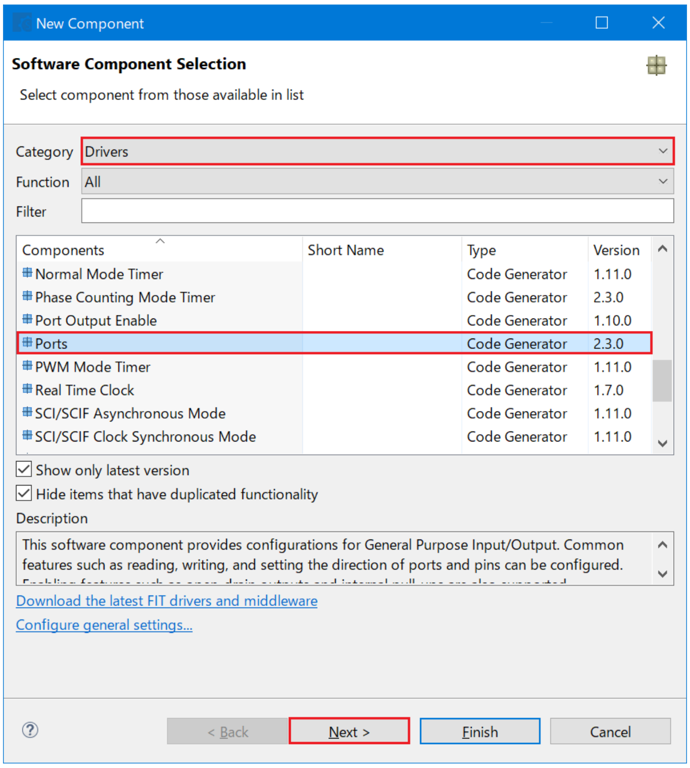


Figure 4-23 Select Ports

In 'Add new configuration for selected component' dialog -> Resource, select 'PORT' as shown in **Figure 4-24** below then click 'Finish'.

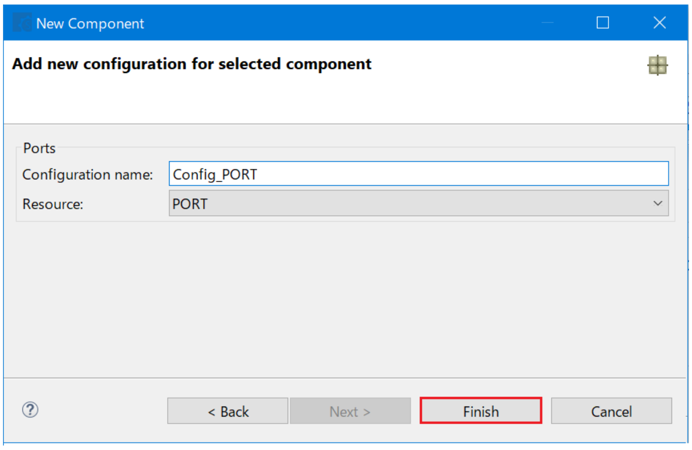


Figure 4-24 Select resource – PORT

Tick the tickboxes for ‘PORT0’, ‘PORT2’, PORTB’, ‘PORTC’ and ‘PORTE’ as shown in **Figure 4-25** below.

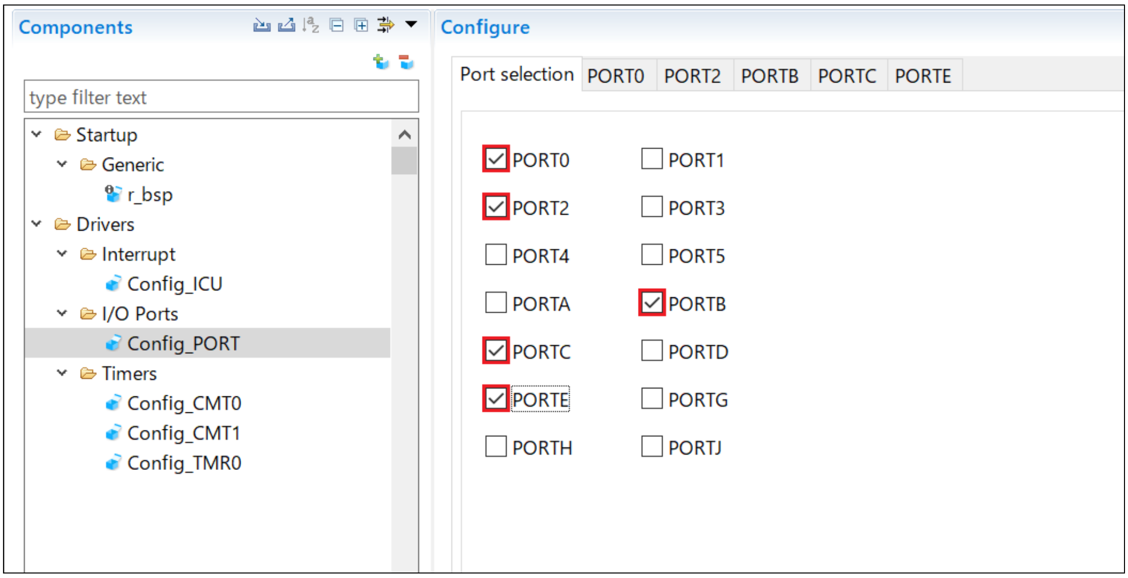


Figure 4-25 Select Port selection

Navigate through each of the ‘PORTx’ tabs, configuring these four I/O lines and LCD control lines as shown in **Figure 4-26**, **Figure 4-27**, **Figure 4-28**, **Figure 4-29** and **Figure 4-30** below. Tick the tickboxes for ‘Out’ and tick ‘Output 1’ the tickboxes except for PC6 under the ‘PORTC’ tab. Start with the 'PORT0' tab.

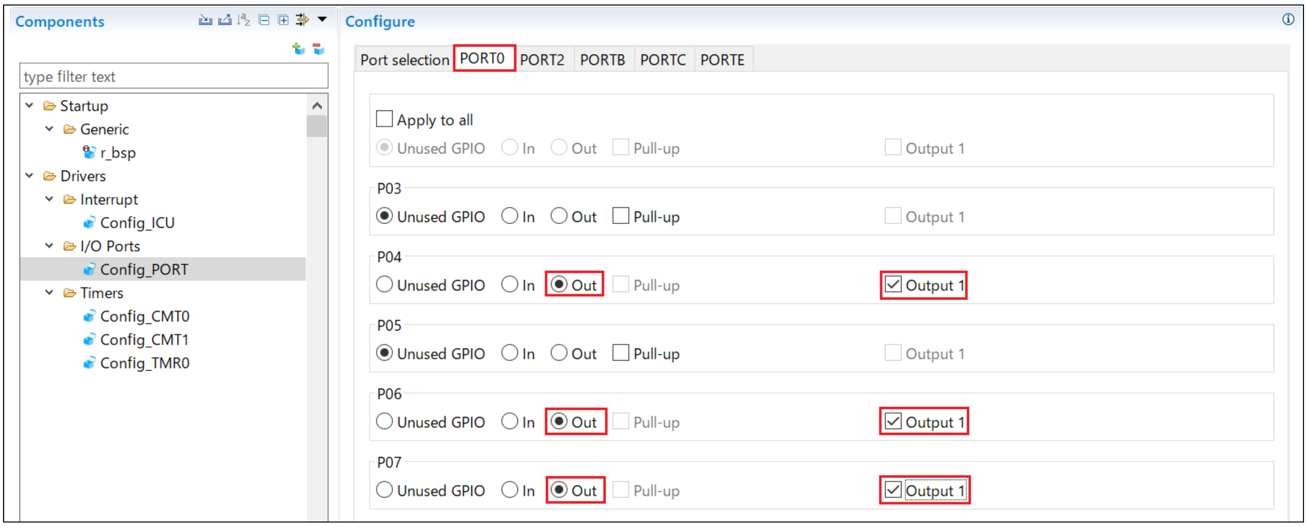


Figure 4-26 Select PORT0 tab

Select ‘PORT2’ tab.

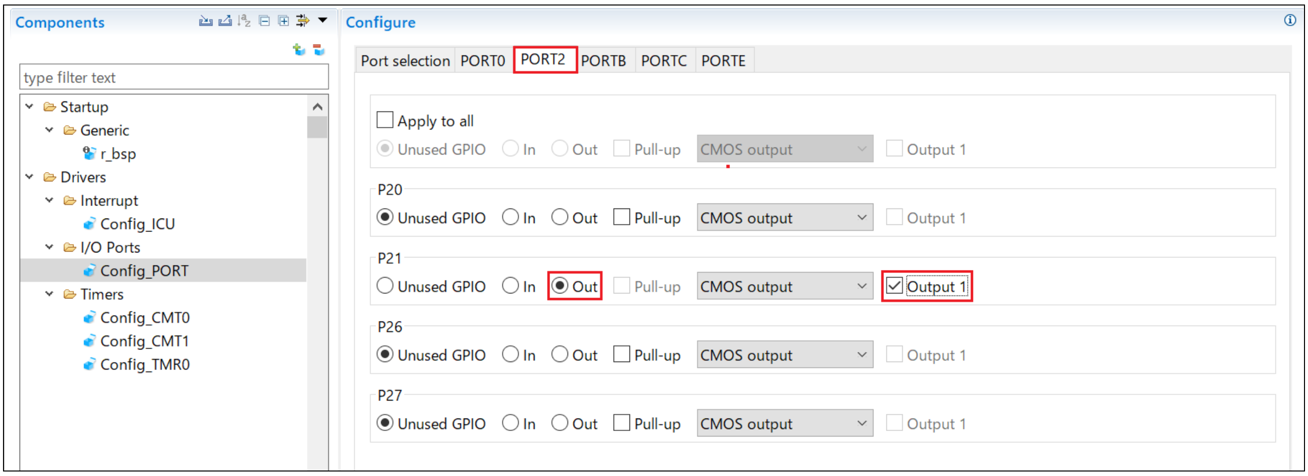


Figure 4-27 Select PORT2 tab

Select ‘PORTB tab.

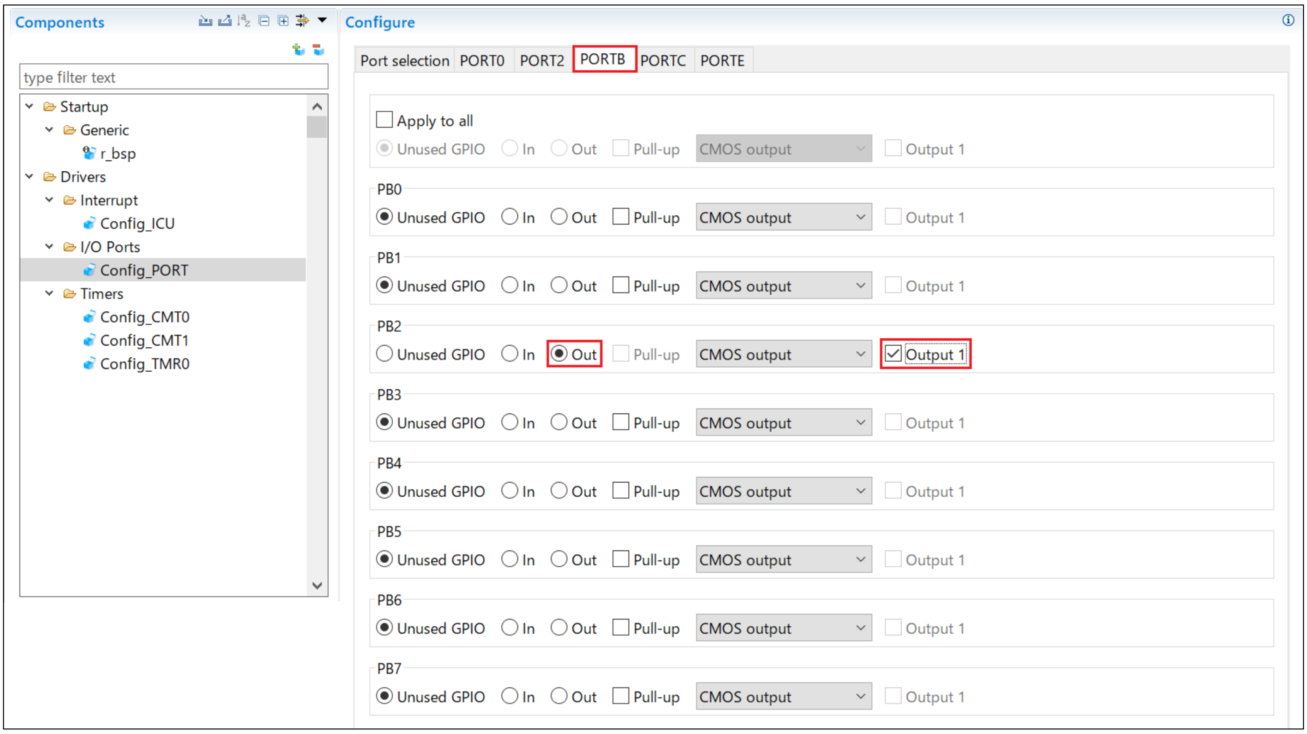


Figure 4-28 Select PORTB tab

Select 'PORTC' tab.

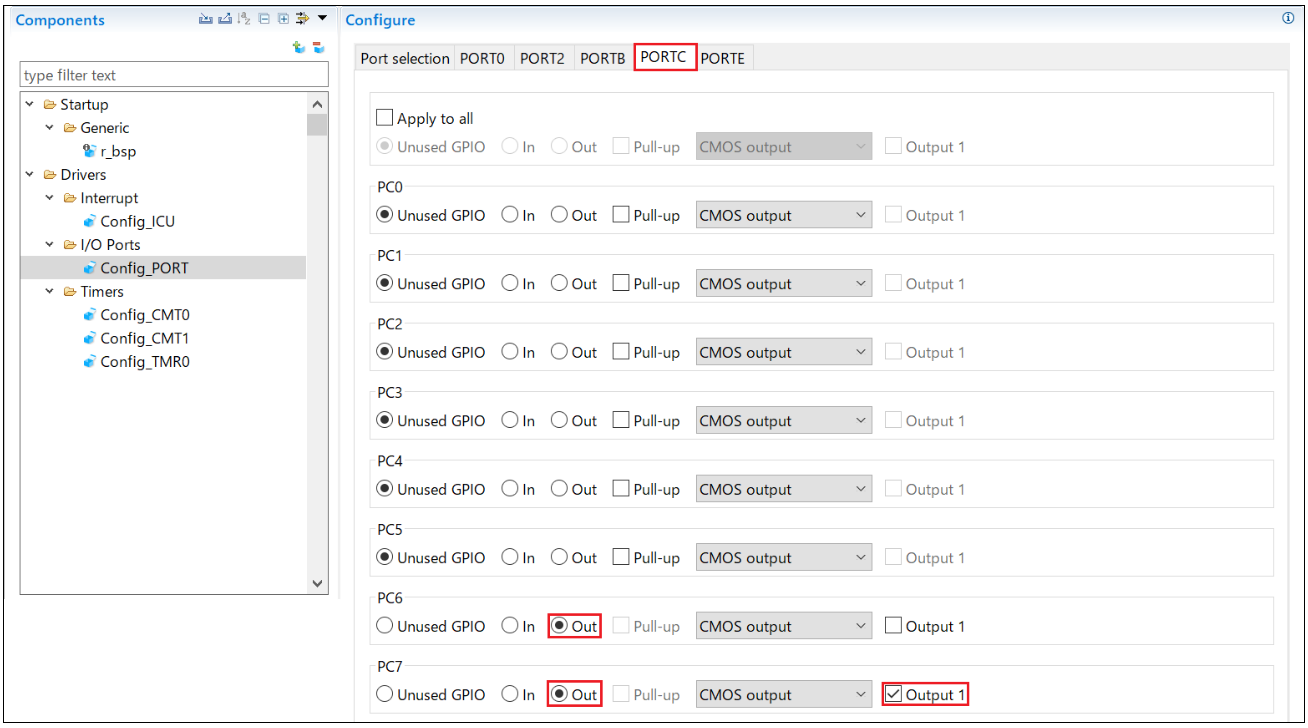


Figure 4-29 Select PORTC tab

Select 'PORTE tab.

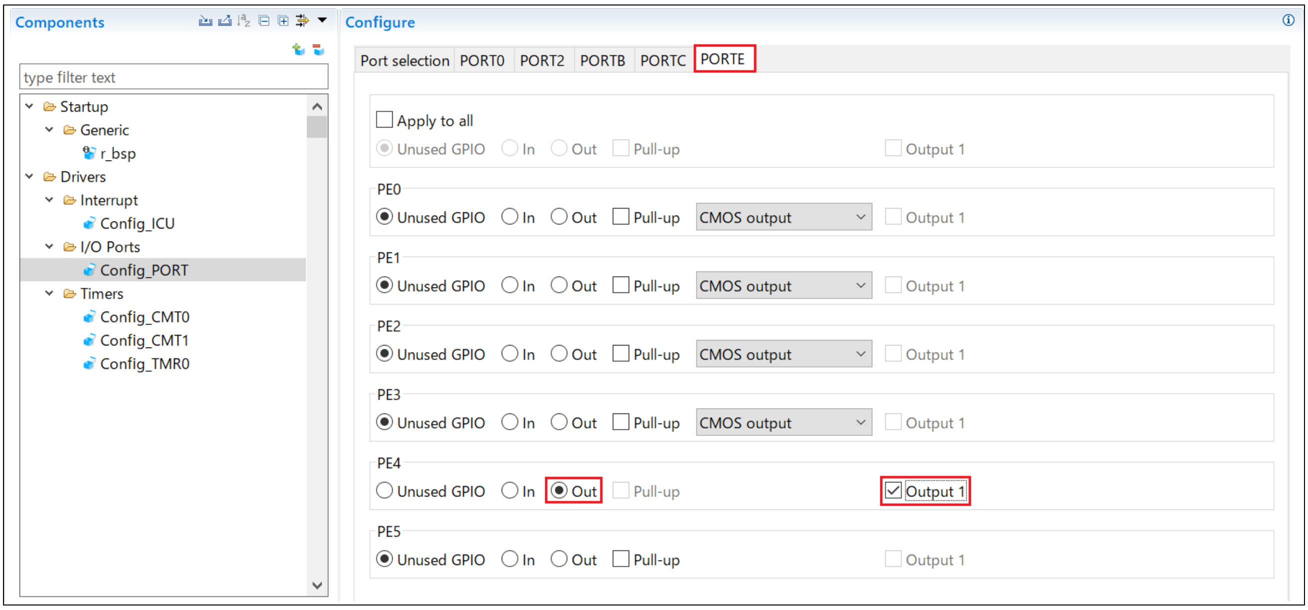


Figure 4-30 Select PORTE tab

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 4-31** then click ‘Next’.

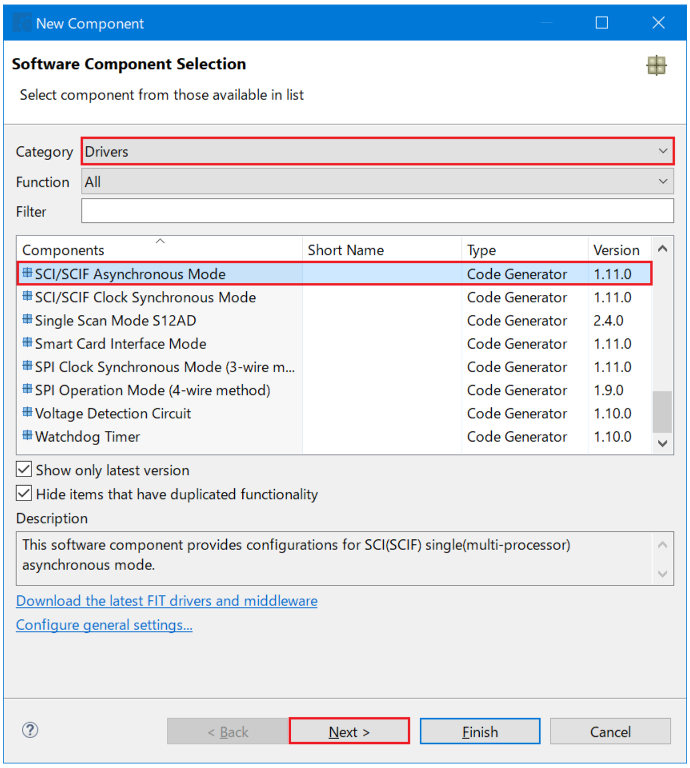


Figure 4-31 Select SCI/SCIF Asynchronous Mode

In ‘Add new configuration for selected component’ dialog -> Work mode, select ‘Transmission/Reception’ as shown in **Figure 4-32** below.

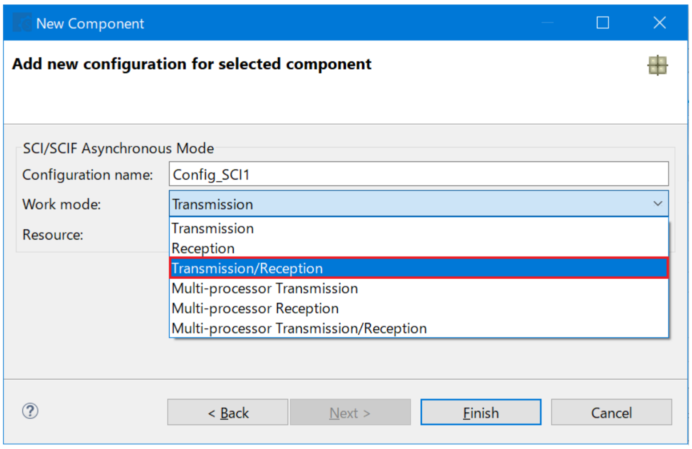


Figure 4-32 Select Work mode – Transmission/Reception

Ensure that the ‘Configuration name’ is set to ‘Config_SCI1’ as shown in **Figure 4-33** below then click ‘Finish’.

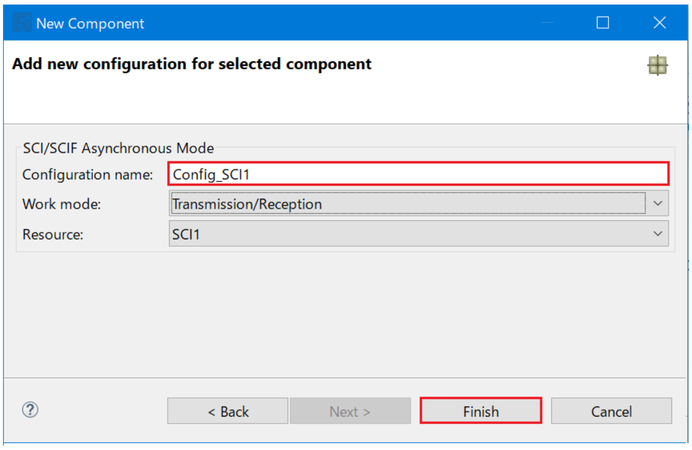


Figure 4-33 Ensure Configuration name - Config_SCI1

Configure SCI1 as shown in **Figure 4-34**. 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.

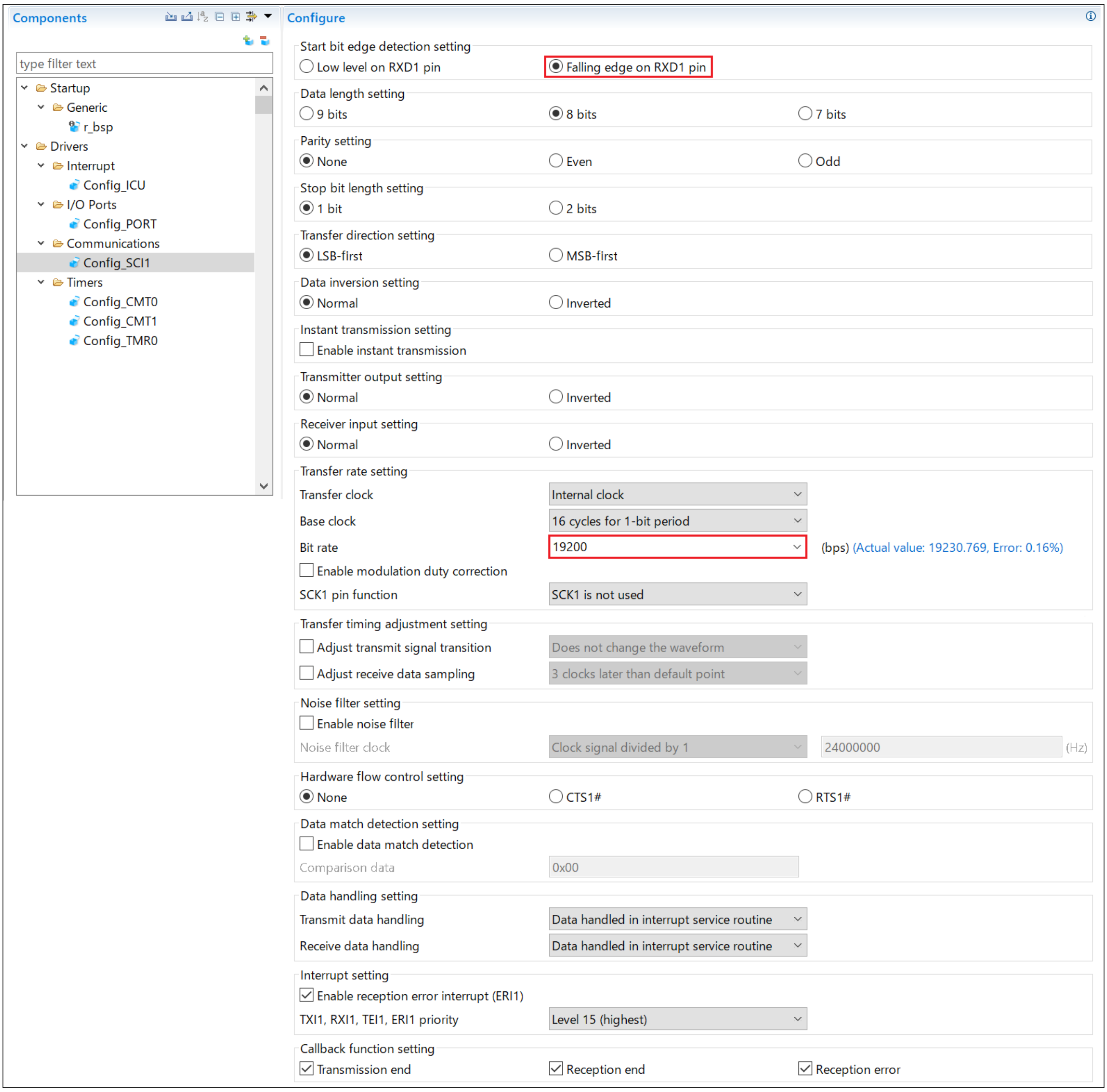


Figure 4-34 Config_SCI1 setting

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 4-35** then click 'Next'.

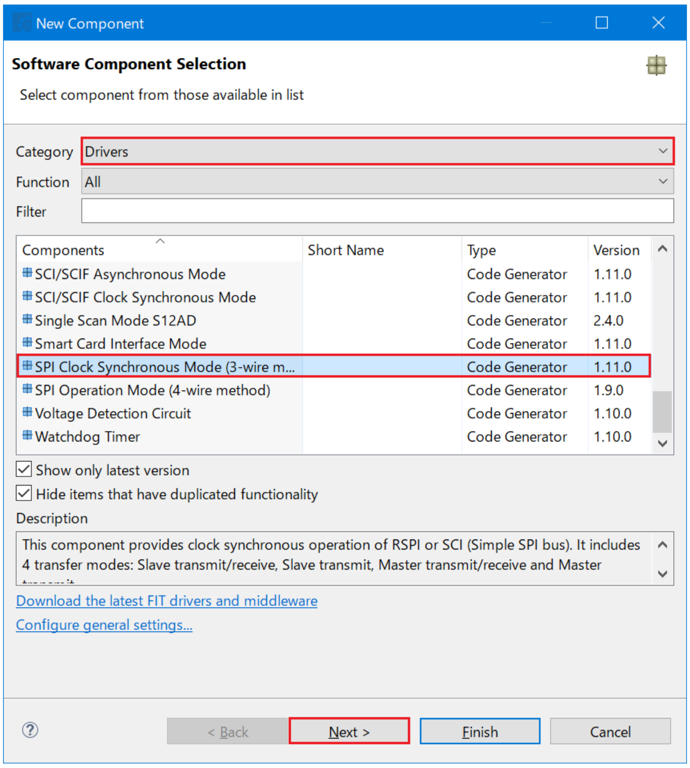


Figure 4-35 Select SPI Clock Synchronous Mode

In 'Add new configuration for selected component' dialog -> Operation, select 'Master transmit only' as shown in **Figure 4-36** below.

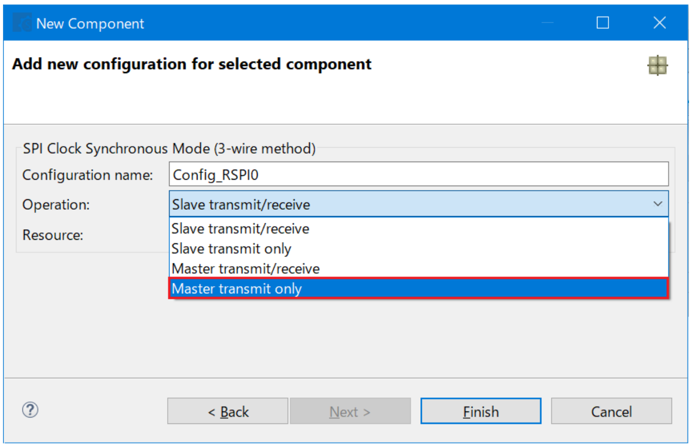


Figure 4-36 Select Operation – Master Transmit

In 'Resource', select 'SCI6' as shown in **Figure 4-37** below.

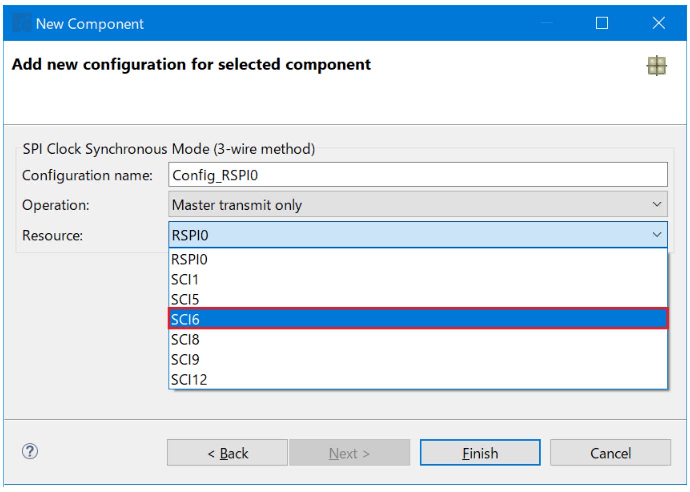


Figure 4-37 Select Resource – SCI6

Ensure that the 'Configuration name' is set to 'Config_SCI6' as shown in **Figure 4-38** below then click 'Finish'

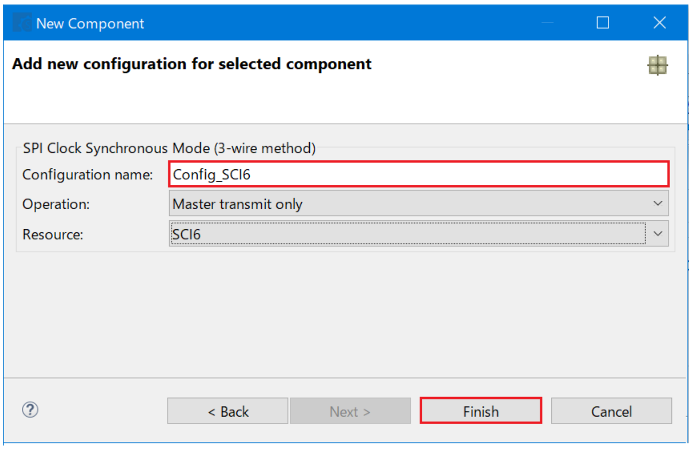


Figure 4-38 Ensure Configuration name - Config_SCI6

Configure SCI6 as shown in **Figure 4-39**. 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.

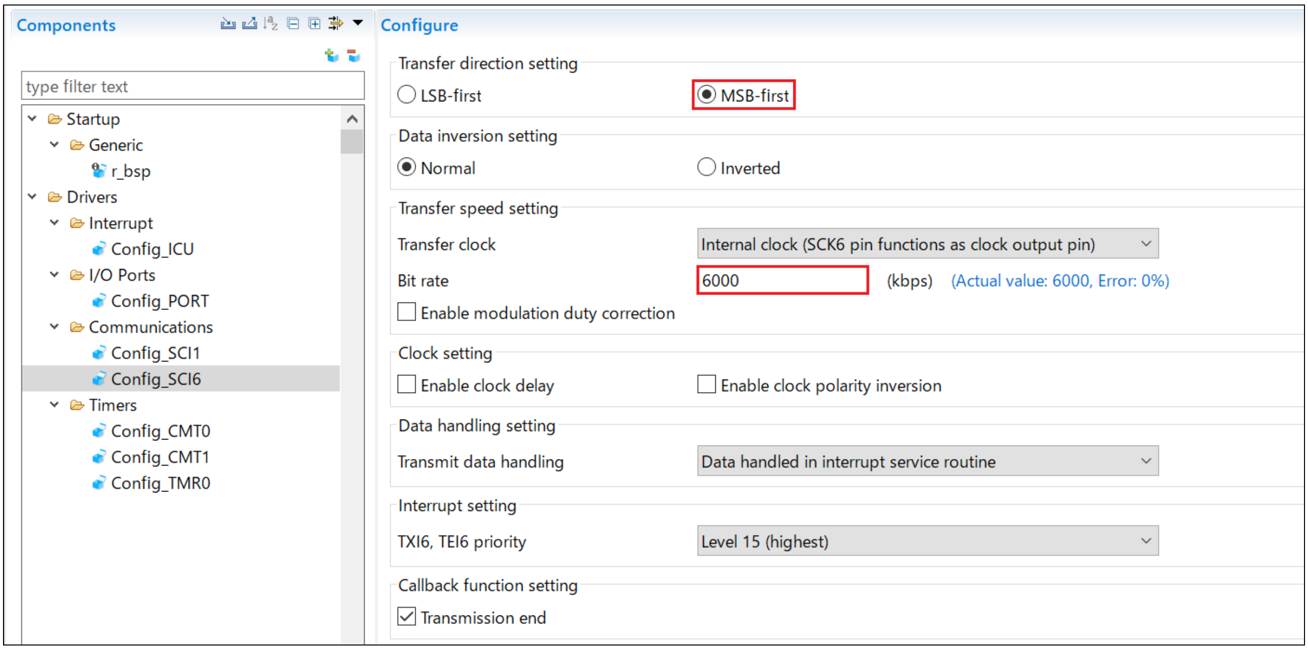


Figure 4-39 Config_SCI6 setting

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 4-40** then click 'Next'.

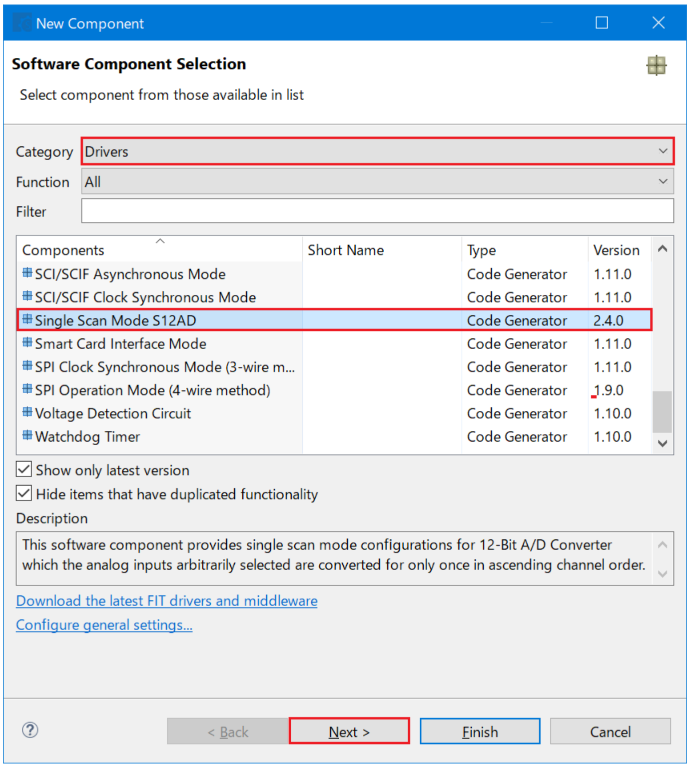


Figure 4-40 Select Single Scan Mode S12AD

Ensure that the 'Configuration name' is'Config_S12AD0' as shown in **Figure 4-41** below then click 'Finish'.

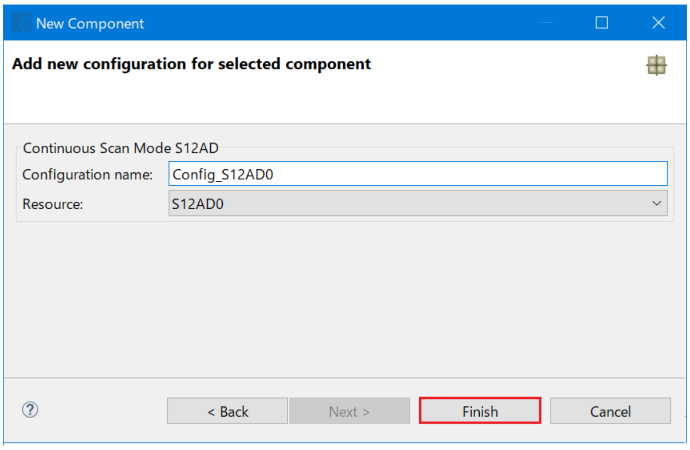


Figure 4-41 Ensure Configuration name - S12AD0

Configure S12AD0 as shown in **Figure 4-42** and **Figure 4-43**. 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.

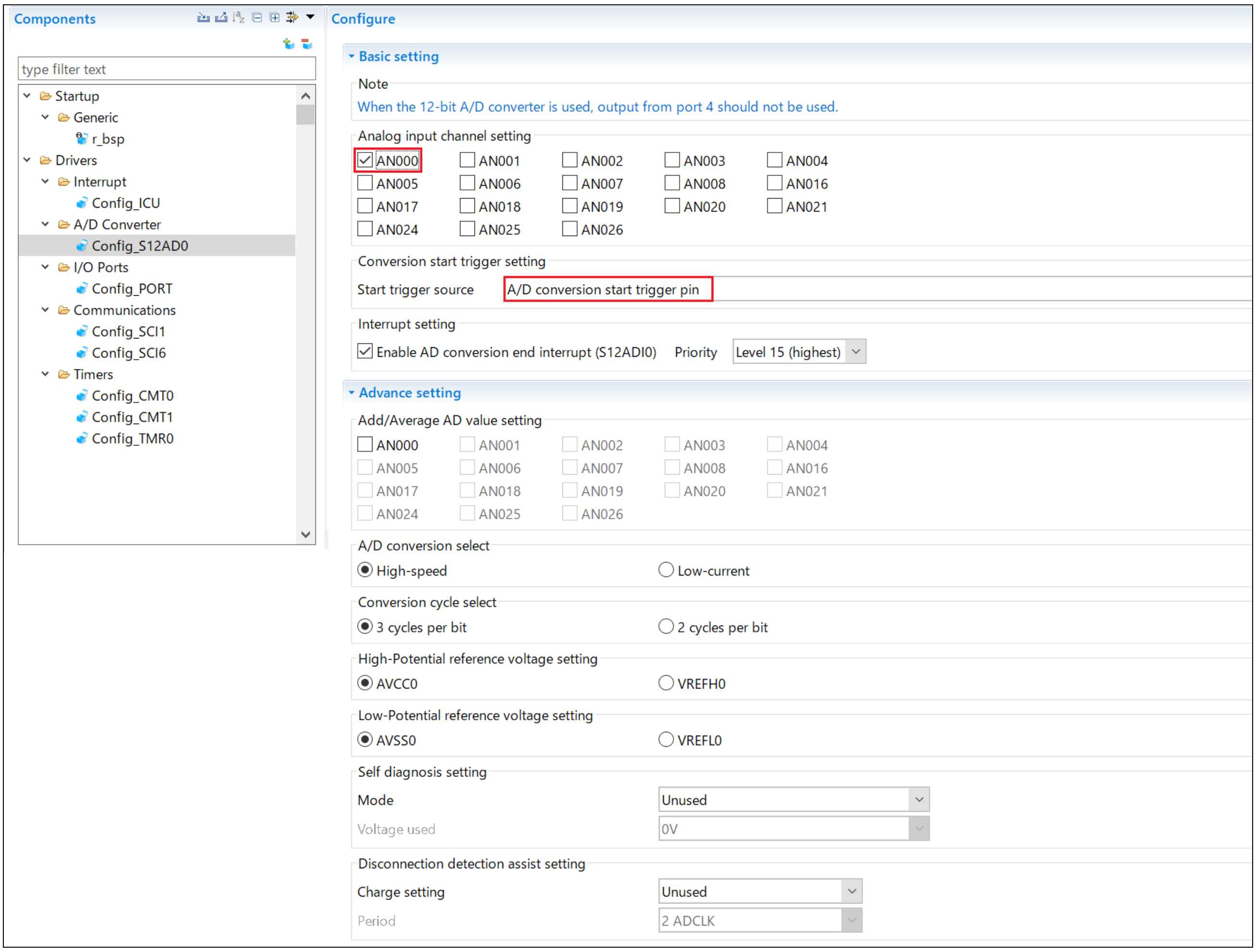


Figure 4-42 Config_S12AD0 setting (1)

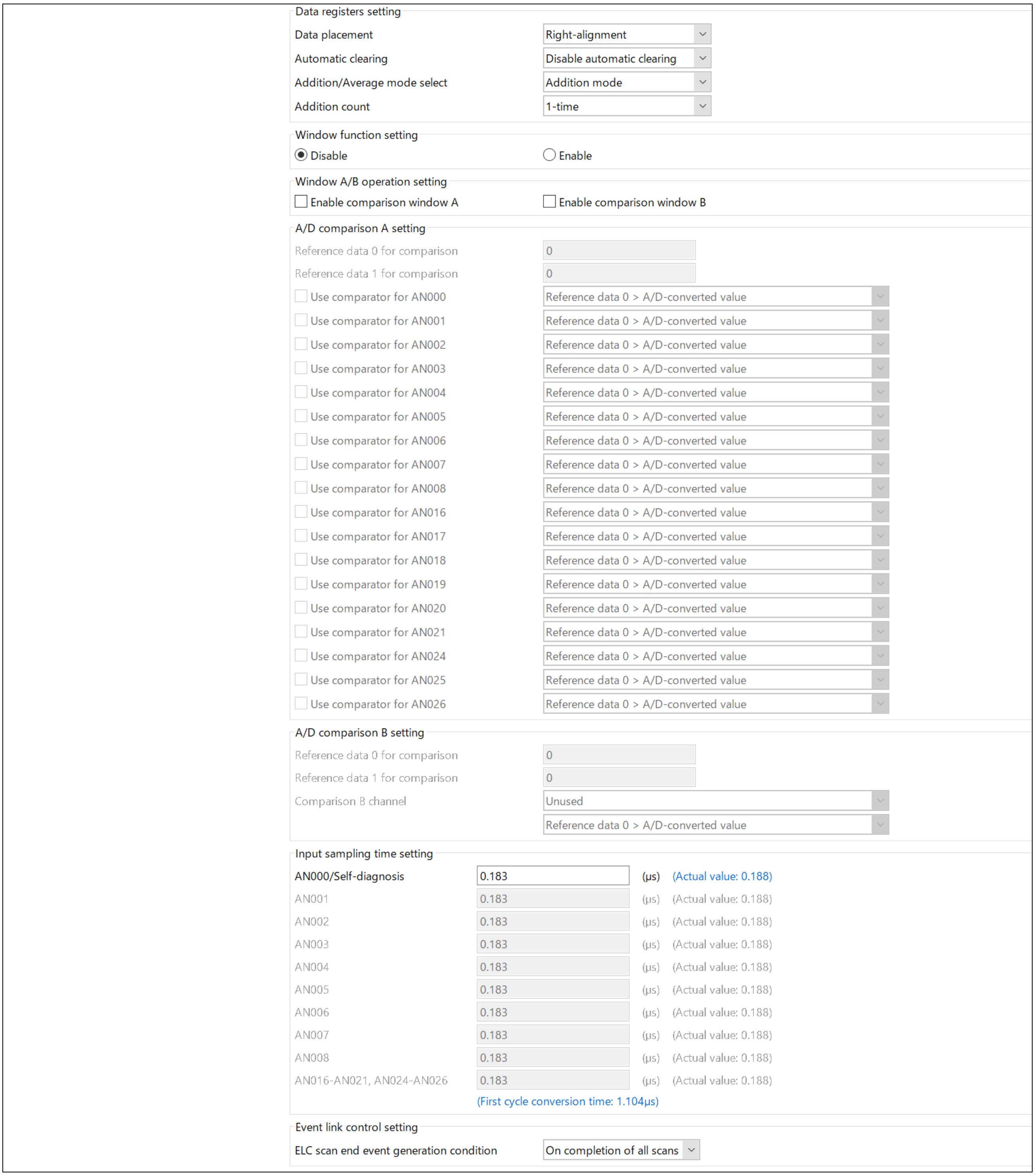


Figure 4-43 Config_S12AD0 setting (2)

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 at Pins page.

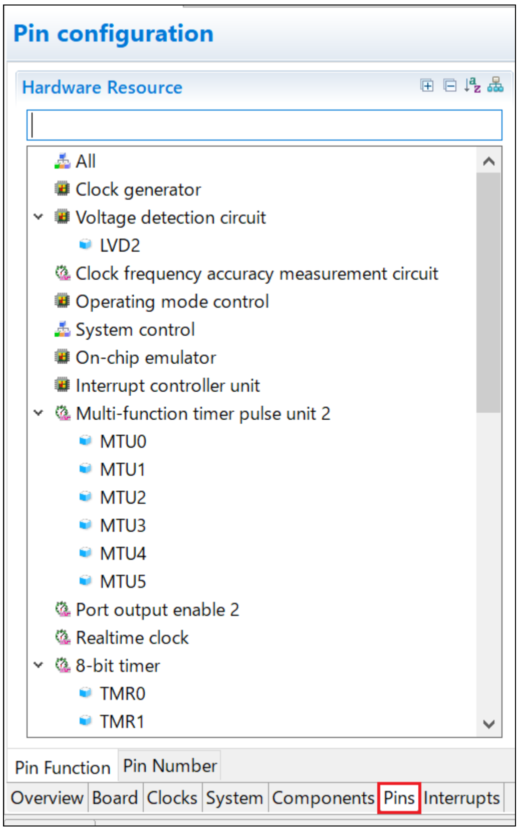


Figure 4-44 The 'Pins' tabbed page

4.7.1 Change pin assignment of a software component

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

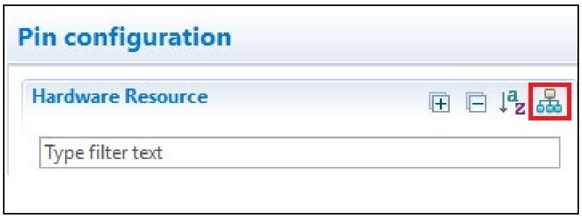


Figure 4-45 Change view to show by Software Components

Select the Config_SCI1 of software component. 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 4-46**.

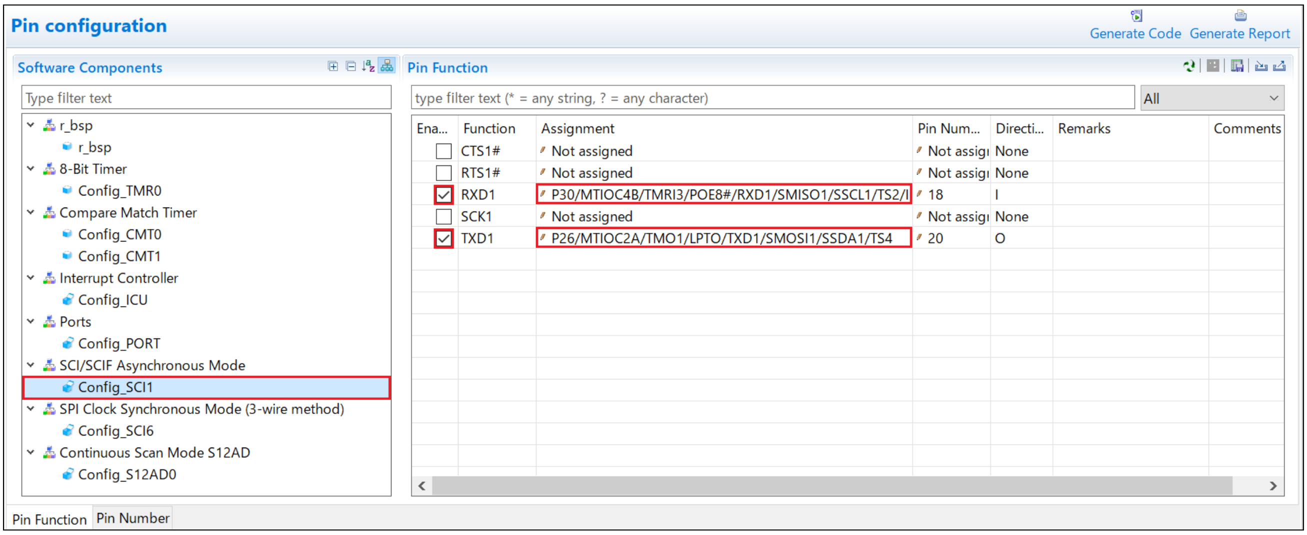


Figure 4-46 Configure pin assignment - Config_SCI1

Select the Config_SCI6 of software component. 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 4-47**.

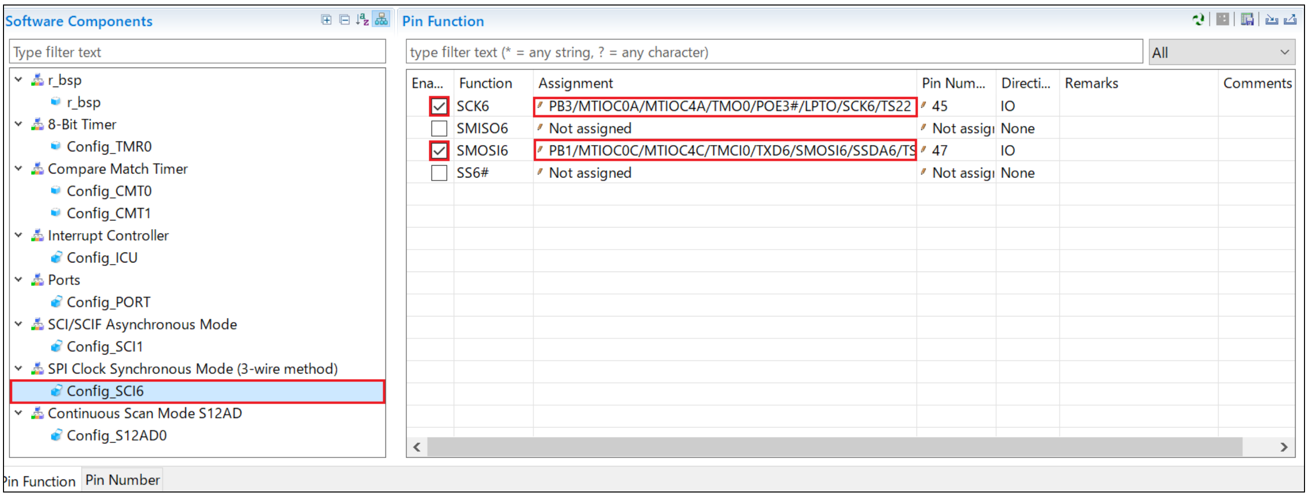


Figure 4-47 Configure pin assignment - Config_SCI6

Select the Config_ICU of software component. 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 4-48**.

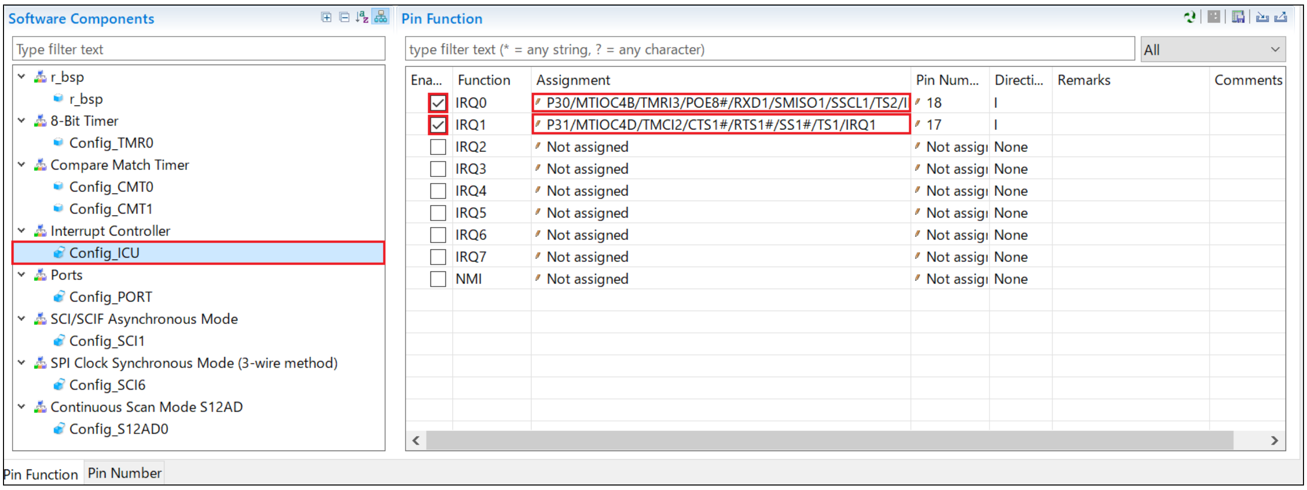


Figure 4-48 Configure pin assignment - Config_ICU

Select the Config_S12AD0 of software components. In the Pin Function list -> Assignment column, Ensure the 'Enable' tick box of ADTRG0# and AN000 are checked and Assignment column of AN000 is P40, ADTRG0# is P16 as shown in **Figure 4-49**.

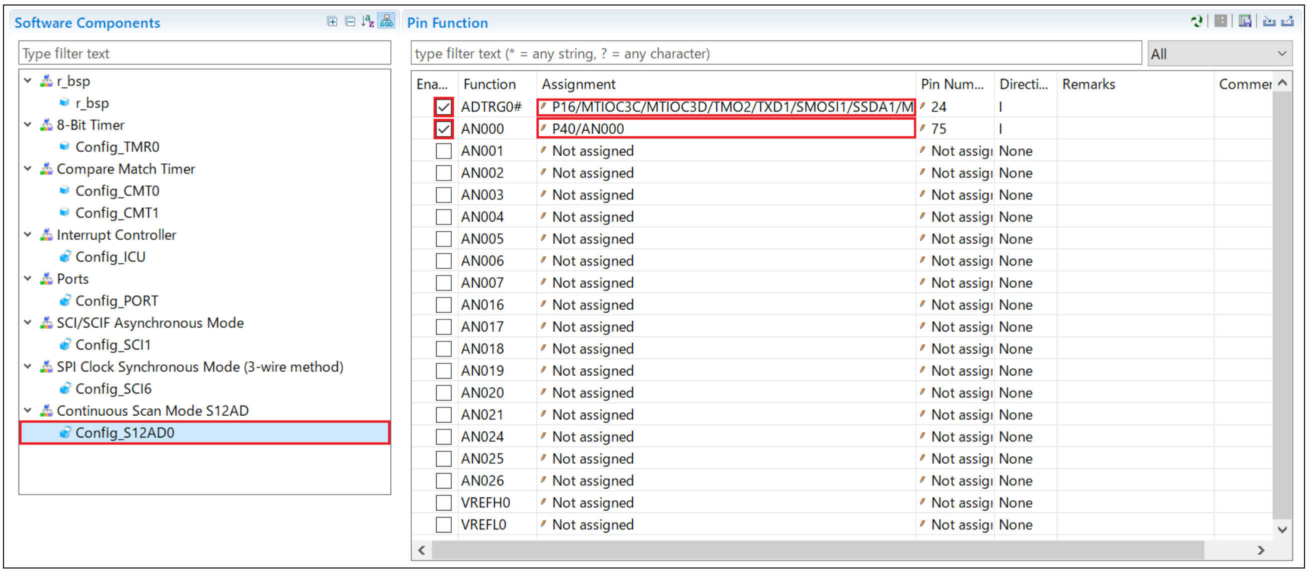


Figure 4-49 Configure pin assignment - Config_S12AD0

Peripheral function configuration is now complete. Save the project using the File -> Save, then click 'Generate Code' at location of **Figure 4-50**.

Figure 4-50 Generate Code Button

If the Section Setting Dialog is displayed as shown in the **Figure 4-51**, Please check the box and click “Yes”.

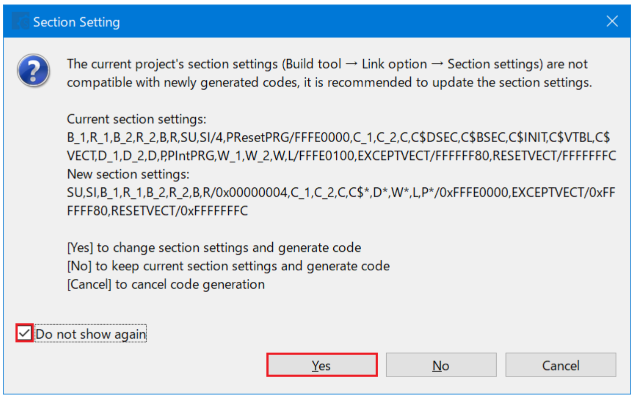


Figure 4-51 Section Setting Dialog

The Console pane should report ‘Code generation is successful’, as shown **Figure 4-52** below.
After execution, close Smart Configurator and return to CS +.

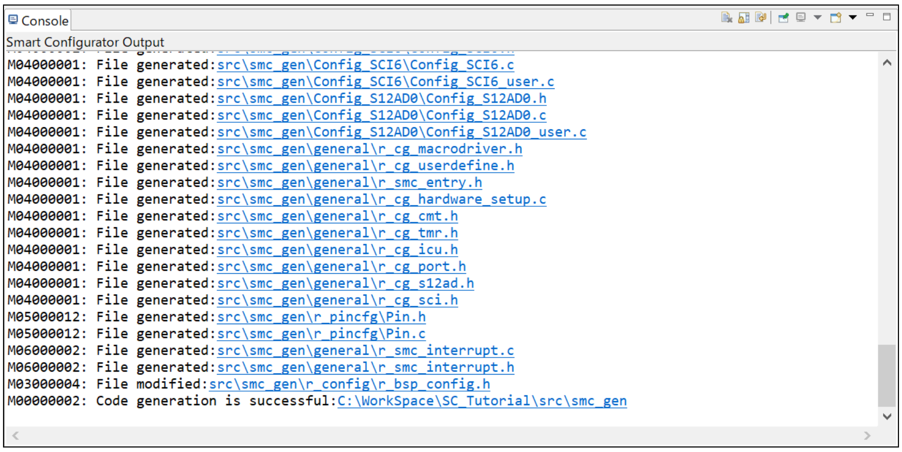


Figure 4-52 Smart Configurator console

When code generation is executed, the startup files generated at the time of CS+ project creation are replaced with those generated by Smart Configurator. **Figure 4-53** the project tree after code generation. In the next chapter, user code is added to these files, and SC_Tutorial is completed by adding a new source file to the project.

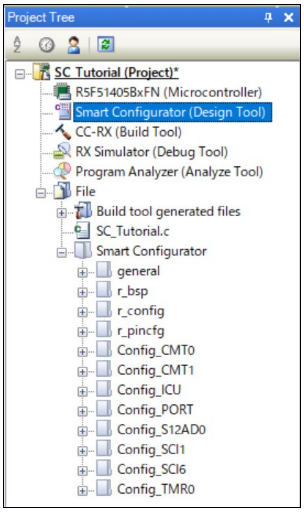
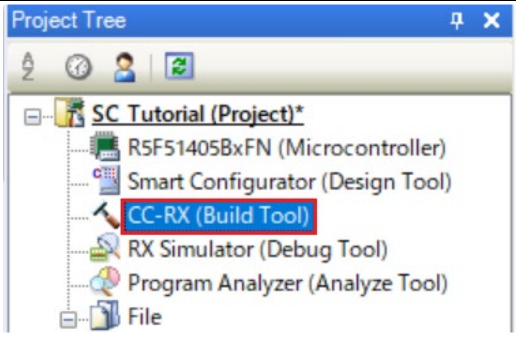


Figure 4-53 Smart Configurator folder structure

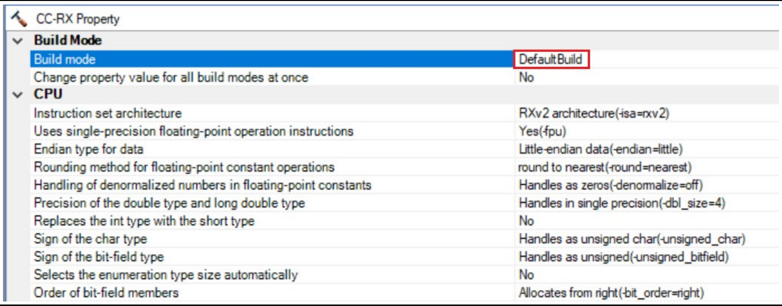
5. Completing the Tutorial Project

5.1 Project Settings

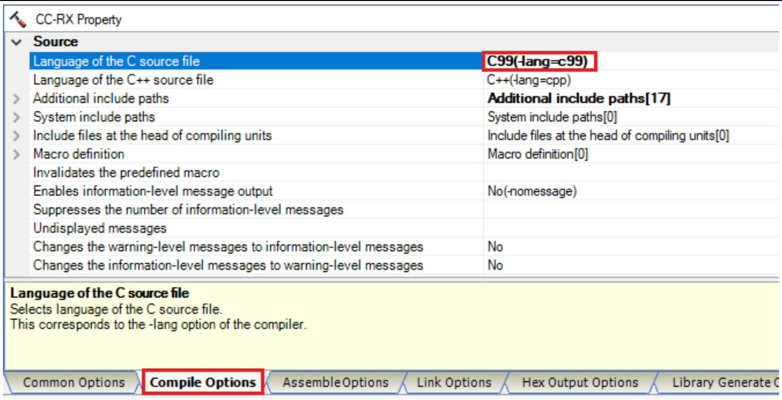
- In the 'Project Tree' pane, select 'CC-RX (Build Tool)'. The build properties will appear in the main window.



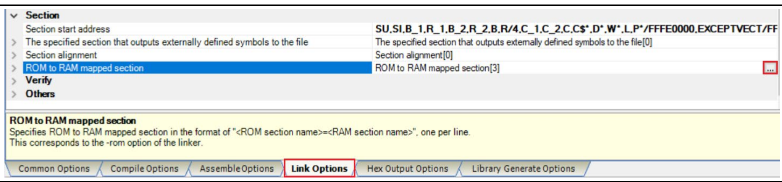
- CS+ creates a single build configuration called 'Default Build' for the project. This has standard code optimisation turned on by default.



- Select the 'Compile Options' tab at the bottom of the properties window pane. Under 'Language of the C source file' select 'C99(-lang=c99)' as shown opposite.



- Select the 'Link Options' tab at the bottom of the properties window pane. Under 'Section -> ROM to RAM mapped section', add the three mappings as shown opposite.



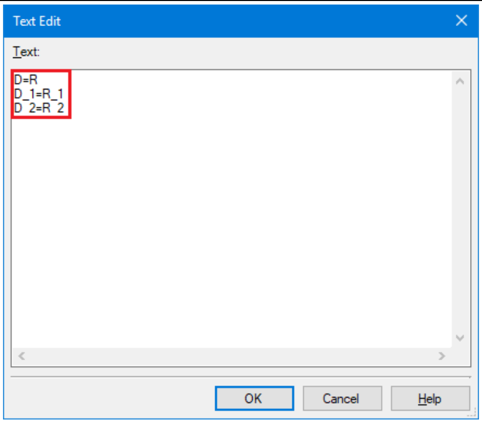
- These settings are easily added by clicking the button '…' and pasting the following text into the dialog:

D=R

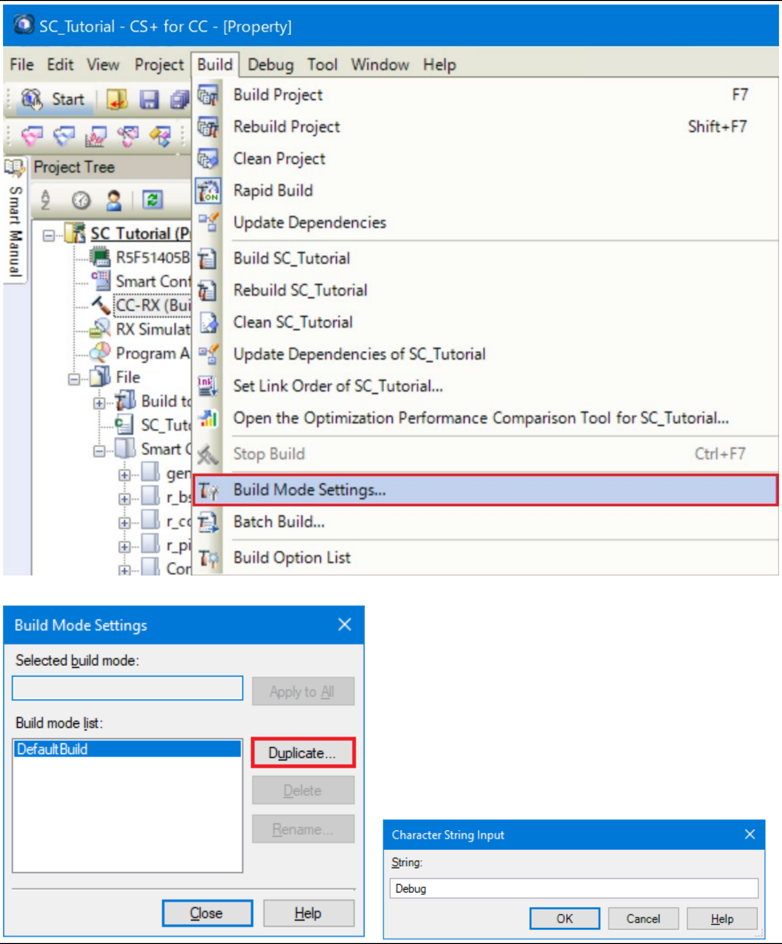
D_1=R_1

D_2=R_2

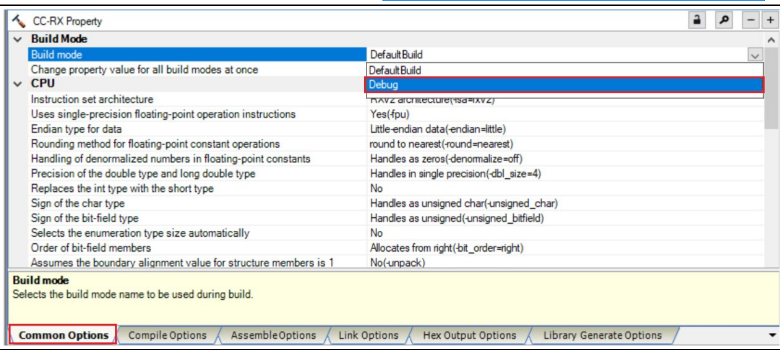
- This ensures that the linker assigns RAM rather than ROM addresses to C variables. Click 'OK'



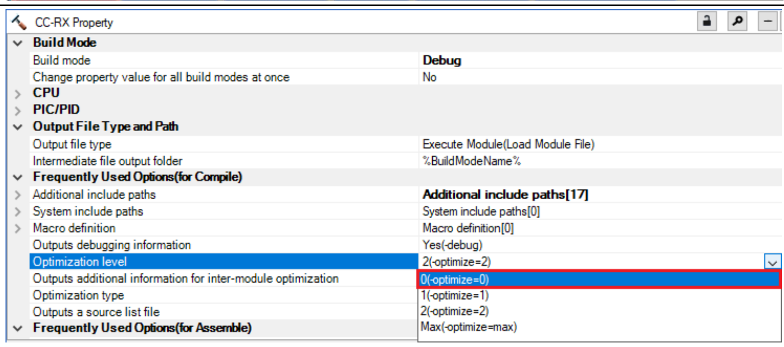
- From the 'Build' menu, select 'Build Mode Settings…'. Click 'Duplicate' and in the resulting 'Character String Input' dialog, enter 'Debug' for the name of the duplicate Build Mode.



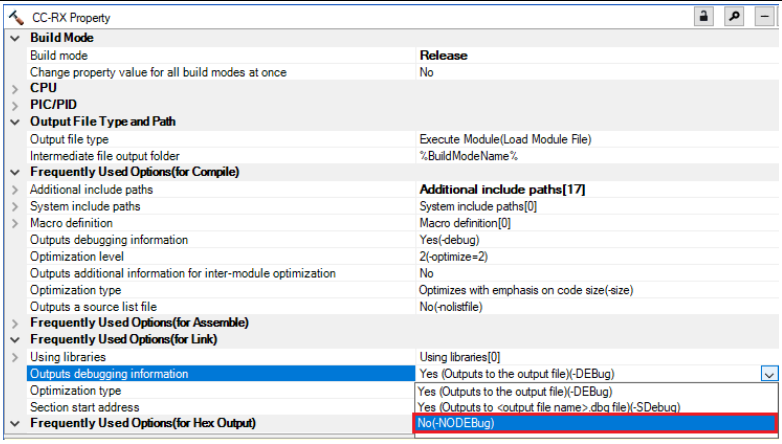
- The new 'Debug' Build Mode will be added to the Build Mode list. Click 'Close'. Now, in the main CC-RX Property window, under the 'Common Options' tab, click on the line containing 'Build Mode', click the pull-down arrow and select 'Debug' from the pull-down.



- In the 'Frequently Used Options (for Compile)' group, select the 'Optimization Level' option and select '0' from the pull-down. We have now created a 'Debug' Build Mode with no code optimisation and will be using the Build Mode to create and debug the project.

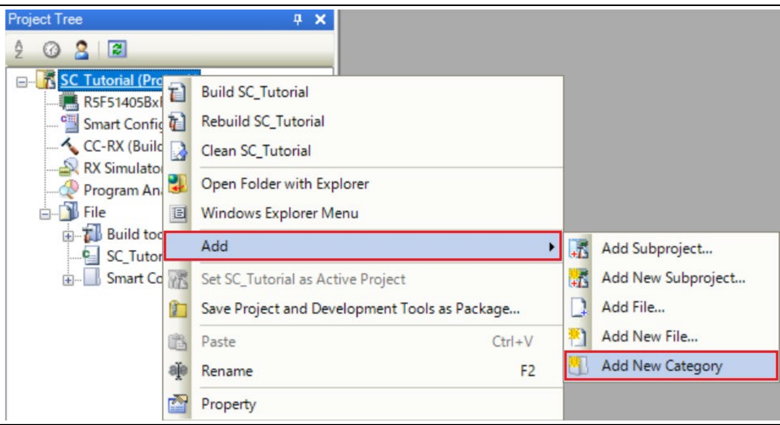


- All of the sample code projects contained in this RSK are configured with three Build Modes; 'DefaultBuild', 'Debug' and 'Release'. 'Release' is created in the same way as above; by duplicating 'Default Build'. 'Release' Build Mode leaves code optimisation turned on and removes debug information from the output file.
- To remove debug information from the 'Release' Build Mode, in the 'CC-RX Property' window, select the 'Common Options' tab at the bottom of the window pane. For the 'Outputs debugging information' option, select 'No(-nodebug).
- Reset the Build Mode back to 'Debug' using the 'Build Mode' pull-down control.
- From the menus, select 'File -> Save All' to save all project settings.

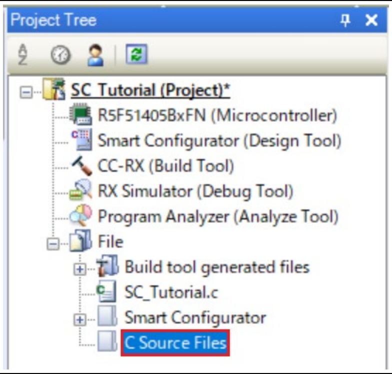


5.2 Additional Folders

- Before new source files are added to the project, we will create two additional folders in the CS+ Project Tree.
- In the Project Tree pane, right-click the SC_Tutorial project and select 'Add -> Add New Category'.



- Rename the newly-created 'New Category' folder to 'C Source Files'. Repeat these steps to create a new category folder for 'Dependencies'.

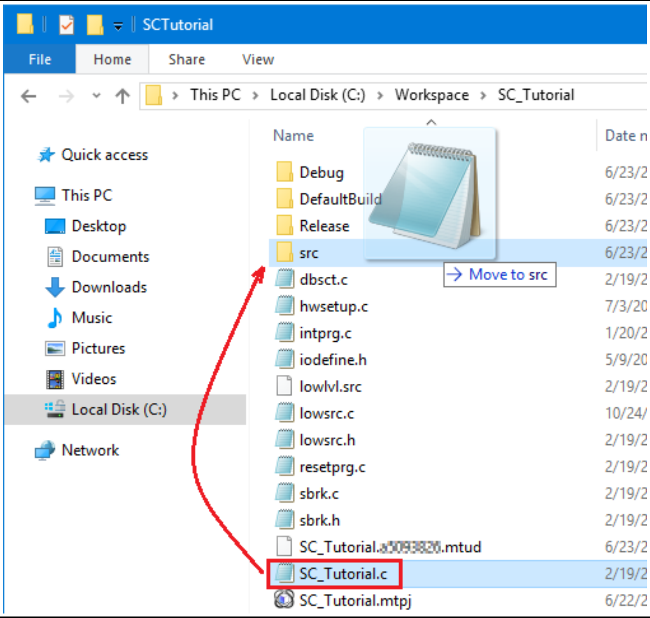
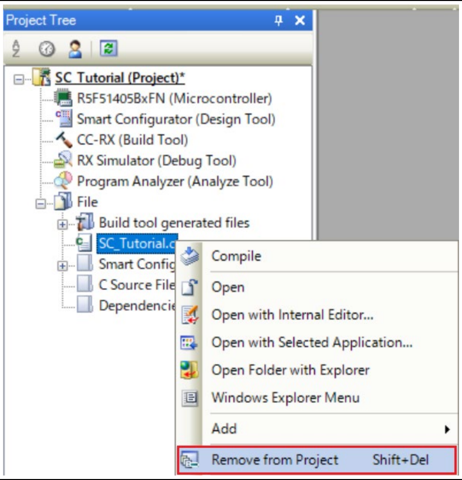
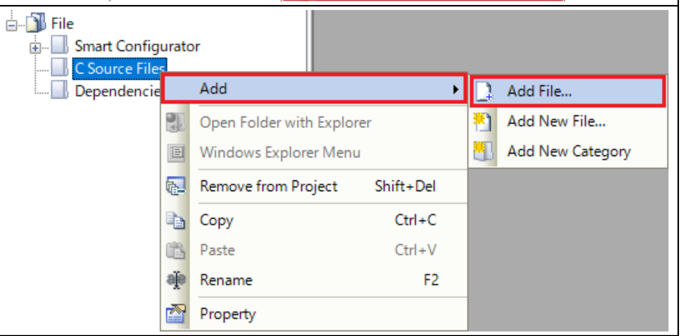


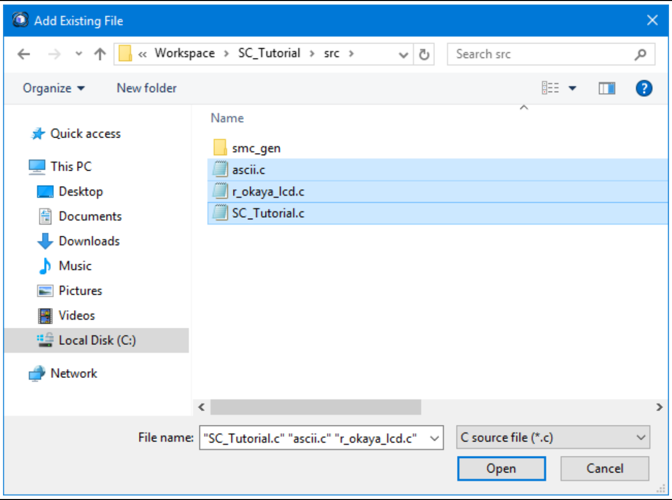
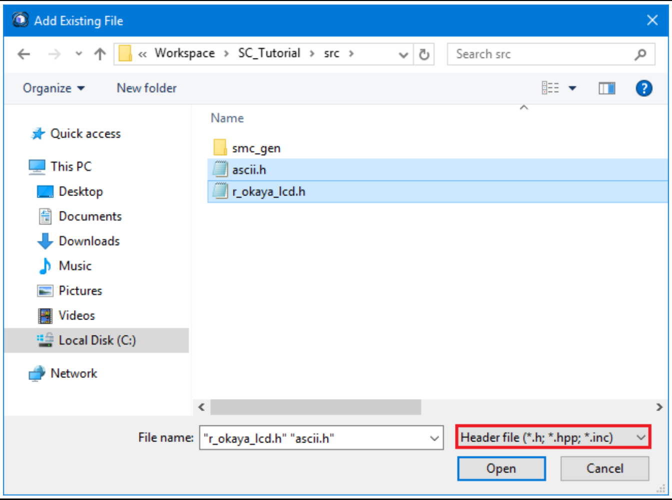
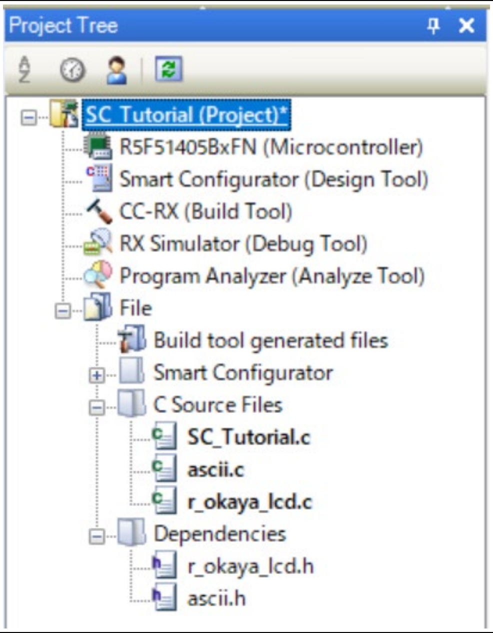
5.3 LCD Code Integration

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 and then follow the steps below.

• Move the 'SC_Tutorial.c' file from 'C:\Workspace\SC_Tutorial' to 'C:\Workspace\SC_Tutorial\src'.	
• Select SC_Tutorial.c on the project tree, Right-click and select 'Remove from Project' to exclude it from the project.	
• Right-click on the 'C Source Files' folder and select 'Add' -> 'Add File...'.	

• Select the files to be added (ascii.c, r_okaya_lcd.c, SC_Tutorial) from C:\Workspace\SC_Tutorial\src.	
• Similarly, add 'ascii.h' and 'r_okaya_lcd.h' to the 'Dependencies' folder. Note: Select the Header file (* .h; * hpp; * .inc).	
• Make sure the project tree is the same as the screen shot.	

Code must be inserted in to the user code area in many files in this project, in the areas delimited by comments as follows:

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

Where _xxxx_ depends on the particular area of code, i.e. 'function' for insertion of user functions and prototypes, 'global' for insertion of user global variable declarations, or 'include' for insertion of pre-processor include directives. User code inserted inside these comment delimiters is protected from being overwritten by Smart Configurator, if the user subsequently needs to use Smart Configurator to regenerate any of the Smart Configurator-generated code.

In the CS+ Project Tree, expand the 'Smart Configurator\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 CS+ Project Tree, expand the 'C Source Files' folder and open the file 'SC_Tutorial.c' by double-clicking on it. Add header files above the 'main' function as shown below.

```
#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 CS+ source file. Also check that the pasted code is correct.

5.3.1 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 CS+ Project Tree, expand the 'Smart Configurator/Config_SCI6' 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.

5.3.2 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. In the CS+ Project Tree, expand the 'Smart Configurator\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
******************************************************************************/
```

Select 'Build Project' from the 'Build' menu, or press F7. CS+ 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 3 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. Add these files into the project in the same way as the LCD files as in section 5.3.

The switch code uses interrupt code in the files Config_ICU.c, Config_ICU_user.c and Config_ICU.h and timer code in the files Config_ICU.c, Config_ICU_user.c, 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 4.6.3 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 CS+ Project Tree, expand the ‘Smart Configurator/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 Project Tree, expand the 'Smart Configurator\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_cmi0_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 Project Tree, expand the 'Smart Configurator\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 same file and 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 CS+ Project Tree, expand the ‘Smart Configurator\general’ folder and open the file ‘r_cg_userdefine.h’ by double-clicking on it. 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 Project Tree, expand the ‘C Source Files’ 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 adding at the below of 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 Project Tree, expand the 'Smart Configurator\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, resulting in the code 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 'Build' menu, or press F7. CS+ 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 SC_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. Add these files into the project in the same way as the LCD files as in section 5.3.

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 CS+ Project Tree, expand the 'Smart Configurator\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

In the Project Tree, expand the 'C Source Files' folder and open the file 'SC_Tutorial.c'. Add the following declaration to above the 'main' function:

```
#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 to 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 UART1.
* 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 ‘Build’ menu, or press F7. CS+ 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 (Baud Rate: 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

In the Project Tree, expand the 'C Source Files' folder and open the file 'SC_Tutorial.c'. Add the following declaration to the above the 'main' function:

```
#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 to 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 ‘Build’ menu, or press F7. CS+ will build the project with no errors.

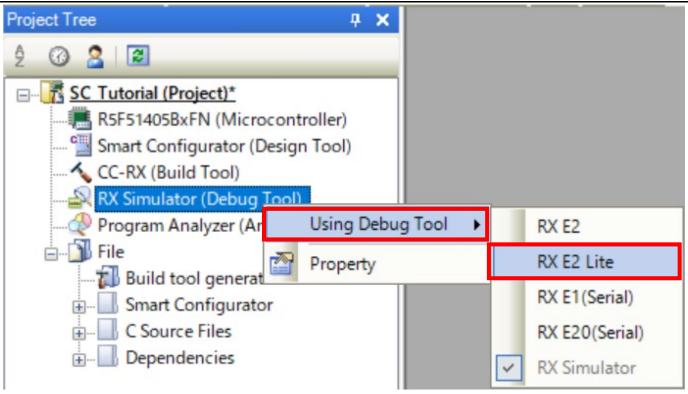
The project may now be run using the debugger as described in section 6. The code will perform the same but now the LEDs will display the s_adc_count in binary form.

6. Debugging the Project

<table>
<tr>
<td>

- In the ‘Project Tree’ pane, right-click the ‘RX Simulator (Debug Tool)’. Select ‘Using Debug Tool -> RX E2 Lite’.

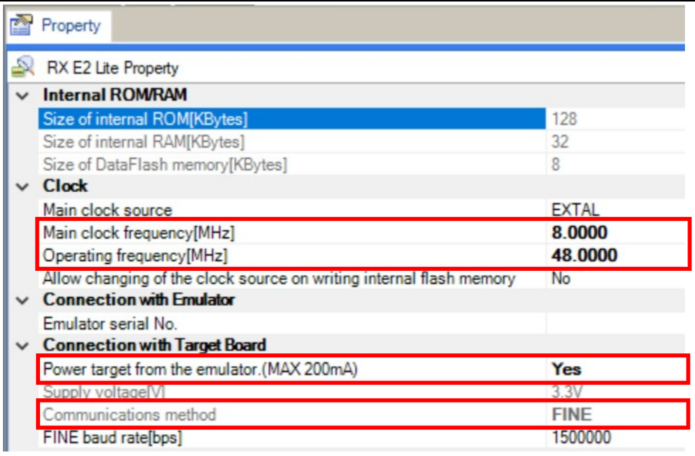
</td>
<td>



</td>
</tr>
<tr>
<td>

- Double-click ‘RX E2 Lite (Debug Tool)’ to display the debugger tool properties. Under ‘Clock’, change the main clock frequency to 8MHz, Communications method ‘FINE’ and operating frequency to 48MHz.’
- Under ‘Connection with Target Board’, change ‘Power target from the emulator. (MAX 200mA)’ to ‘Yes’.
- All other settings can remain at their defaults.

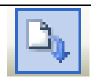
</td>
<td>



</td>
</tr>
<tr>
<td>

- Connect the E2 Lite to the PC and the RSK E1/E2 Lite connector. Connect the Pmod LCD to the PMOD1 connector.
- From the ‘Debug’ menu select ‘Download’ to start the debug session and download code to the target.

</td>
<td>



</td>
</tr>
</table>

7. Running the Smart Configurator Tutorial

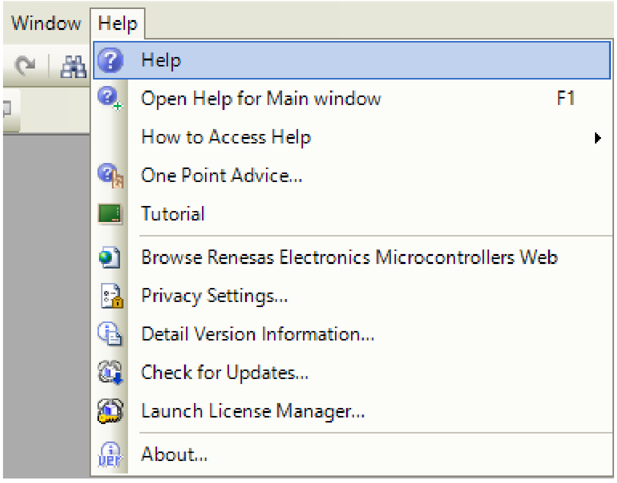
7.1 Running the Tutorial

Once the program has been downloaded onto the RSK device, the program can be executed. Click the ‘Go’ button or press F5 to begin the program from the current program counter position. It is recommended that you run through the program once first, and then continue to the Tutorial manual to review the code.	

8. Additional Information

Technical Support

For details on how to use CS+, refer to the help file by opening CS+, then selecting Help > Help Contents from the rx140 menu bar.



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.

REVISION HISTORY	RX140 Group Renesas Starter Kit for RX140 Smart Configurator Tutorial Manual For CS+

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

RX140 Group
Renesas Starter Kit for RX140
Manual: Smart Configurator Tutorial Manual For CS+

Publication Date: Rev. 1.00 Jan.17.2022

Published by: Renesas Electronics Corporation

RX140 Group

Renesas Electronics Corporation

R20UT5029EG0100