Renesas RX Family

Tracealyzer® for FreeRTOS Debugging

Introduction

FreeRTOS is an RTOS from Amazon Web Services and is based on a high-performance embedded kernel.

Percepio Tracealyzer® (hereafter called Tracealyzer®) is a convenient solution for visual trace diagnostics for developers of RTOS- or Linux-based embedded software systems.
Tracealyzer® provides realtime monitoring in detail of the following information on the internal state of the RTOS.

1. Use of RTOS system calls
2. Per-task and total CPU loads
3. Per-task and total heap usage

This application note describes procedures for checking FreeRTOS thread and object states (referred to as resources) during the development of applications in the e^2 studio. The procedure for starting Tracealyzer® is also explained.

Target Device

RX65N Group (R5F565NEHDFB)

Operating Environment

Target Board	CK-RX65N
Integrated Development Environment (IDE)	e^2 studio version 2023-01
Trace Tool	Percepio Tracealyzer® v4.6.6
OS	FreeRTOS 10.4.3
Toolchain	CC-RX V3.05
USB-Serial Converter	Pmod USBUART module (from Digilent, Inc.)

Note: Please download the e^2 studio, CC-RX, and Tracealyzer® in advance with reference to the documents available at the following URLs.

- e^2 studio Integrated Development Environment 2021-04 and e^2 studio v7.8 User's Manual: Quick Start Guide site:

  e² studio 2021-04 and e² studio v7.8 User's Manual: Quick Start Guide
- RX Smart Configurator User's Guide: e^2 studio site:

  RX Smart Configurator User's Guide: e² studio
- Tracealyzer® for FreeRTOS User Manual site:

  Tracealyzer® for FreeRTOS - User Manual
- Percepio Tracealyzer® download site:

  Download Tracealyzer® - Percepio AB

Contents

1. Installing Tracealyzer® 3

2. Creating a Project in the e² studio 3

3. Debugging with Tracealyzer® (through a UART) 4
3.1 Embedding Tracealyzer® for FreeRTOS into the Project 4
3.1.1 Copying the Tracealyzer® for FreeRTOS Source Code under the Tracealyzer® Installation Folder 4
3.1.2 Removing Unnecessary Folders 4
3.1.3 Creating Files for UART Communications 5
3.2 Settings of the Project 13
3.2.1 Setting the UART to Output Data Monitored by Tracealyzer® 13
3.3 Settings of the Compiler 17
3.3.1 Adding the Include Paths Required by Tracealyzer® through Compiler Settings 17
3.4 Settings of FreeRTOS 19
3.4.1 Modifying "portmacro.h" of the FreeRTOS Kernel 19
3.4.2 Modifying the Hook Function to be Executed before the Startup of the FreeRTOS Kernel 20
3.4.3 Adding the Code for Starting Tracealyzer® to the main Task 21
3.4.4 Building the Project 22
3.5 Connecting the Host PC and CK-RX65N Board 23
3.6 Using the [RTOS Resources] View 25
3.6.1 Displaying the [RTOS Resources] View 25
3.6.2 Context Menu 25
3.6.3 Stack Setting 26
3.6.4 Tabbed Pages 28
3.7 Starting Debugging of a Project with Tracealyzer® 29
3.7.1 Launching the Debugger on the e² studio 29
3.7.2 Launching Tracealyzer® 30
3.7.3 Executing Software 32
3.7.4 Display of Trace Information 33

1. Installing Tracealyzer®

Install Tracealyzer® with reference to the Tracealyzer® for FreeRTOS User Manual.

2. Creating a Project in the e² studio

A project generation wizard is available in the e² studio to ease the generation of an RX project.

Install the e² studio and CC-RX with reference to the e² studio 2021-04 and e² studio v7.8 User's Manual: Quick Start Guide.

Create a CC-RX RTOS project with reference to section 2.2, Create an RTOS project, in the RX Smart Configurator User's Guide: e² studio.

Enter the RTOS and target device information as follows in the wizard.

— RTOS: FreeRTOS (kernel only) *
— RTOS Version: **10.4.3-rx-1.0.5**
— Target Board: **CK-RX65N**

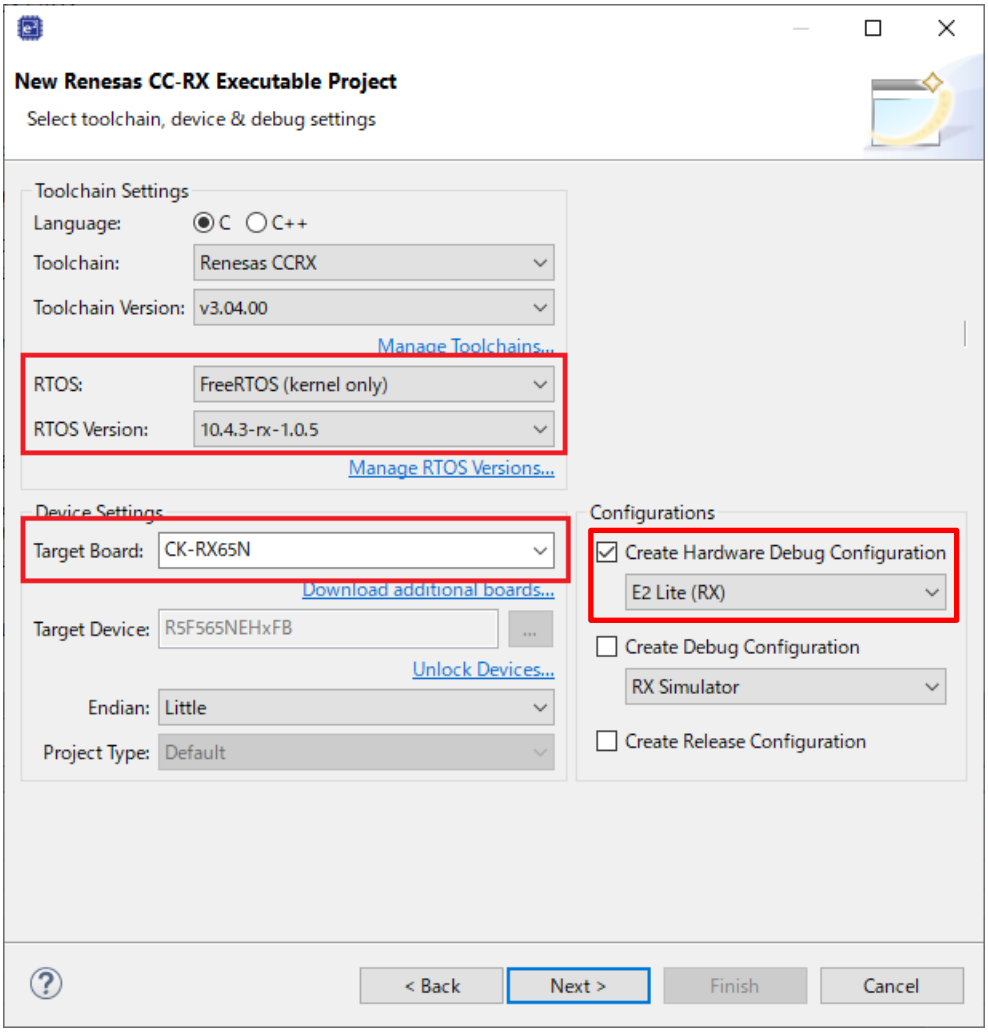

Figure 2-1 Selecting RTOS and the Target Device

Note: * Selection of RTOS:
When "FreeRTOS (with IoT Libraries)" is selected, monitored data can be superimposed on a network communications path (Ethernet). The usage of this method will be described in future versions of this application note.
When "FreeRTOS (kernel only)" is used, one SCI channel (UART mode) is occupied for the transmission of monitored data.

3. Debugging with Tracealyzer® (through a UART)

This section describes how to use Tracealyzer® through a UART.

3.1 Embedding Tracealyzer® for FreeRTOS into the Project

3.1.1 Copying the Tracealyzer® for FreeRTOS Source Code under the Tracealyzer® Installation Folder

Copy the Program Files\Percepio\Tracealyzer 4\FreeRTOS\TraceRecorder folder to the workspace folder "src" by using the File Explorer of Windows.

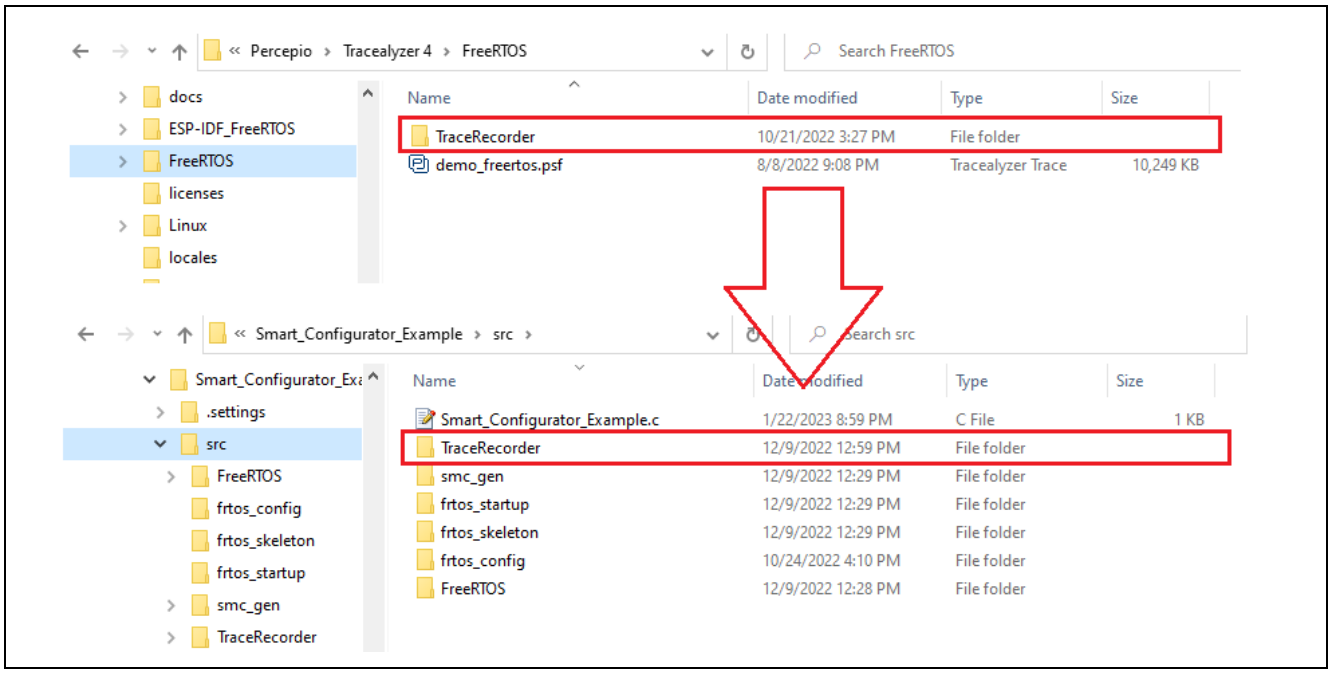

Figure 3-1 Copying the Folder

3.1.2 Removing Unnecessary Folders

Remove all sub-folders in the workspace folder "src/TraceRecorder/streamports".

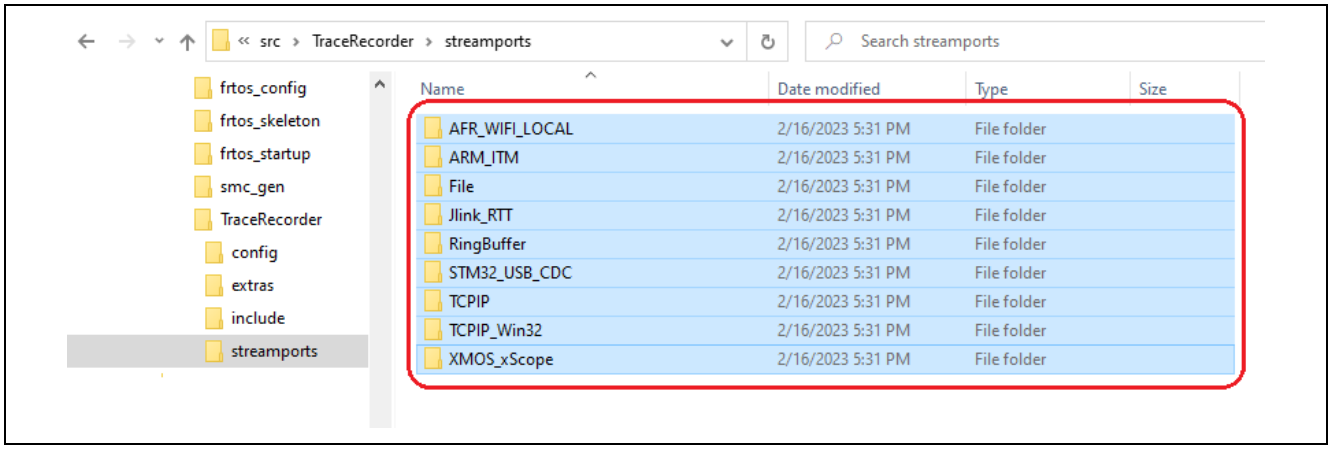

Figure 3-2 Folders for Removal

3.1.3 Creating Files for UART Communications

Use the Project Explorer of the e² studio to create the "Renesas_RX_UART" folder under the "streamports" folder and then create the "config" and "include" folders in the "Renesas_RX_UART" folder. In these folders, create empty files with the names "trcStreamPort.c", "trcStreamPort.h", "trcStreamPortConfig.h", and "Readme-Streamport.txt" as shown below.

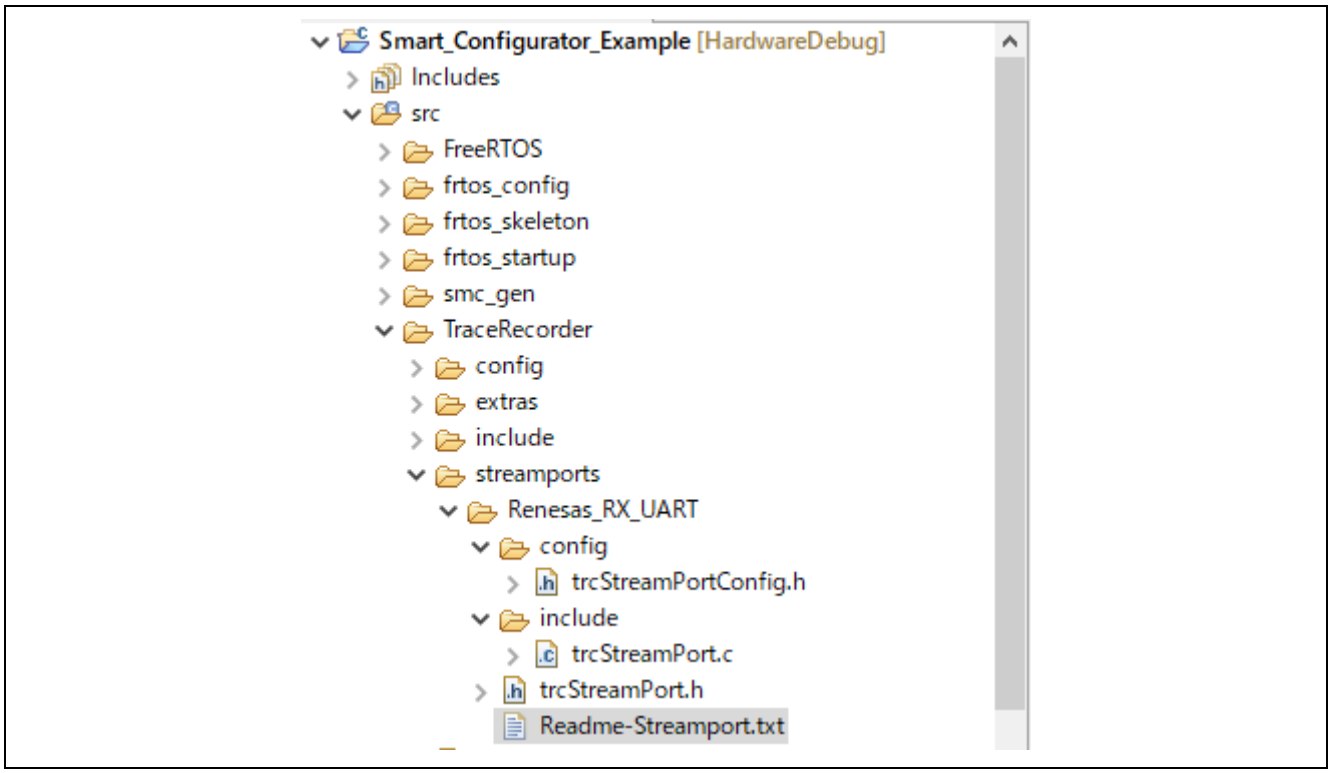

Figure 3-3 Creating Folders and Files

Copy the following code to "trcStreamPort.c".

```
#include <string.h>

#include "trcRecorder.h"
#include "r_sci_rx_if.h"
#include "r_sci_rx_pinset.h"

#if (TRC_CFG_RECORDER_MODE == TRC_RECORDER_MODE_STREAMING)
#if (TRC_USE_TRACEALYZER_RECORDER == 1)

static uint8_t string[1024];
static uint8_t sci_buffer[1024];
static uint32_t sci_current_received_size = 0;
static volatile uint32_t wait_sending = 0;

extern sci_hdl_t sci_handle_tracealyzer;

void sci_callback_tracealyzer(void *arg);

traceResult xTraceStreamPortInitialize(TraceStreamPortBuffer_t* pxBuffer)
{
  TRC_ASSERT_EQUAL_SIZE(TraceStreamPortBuffer_t,
TraceStreamPortUSBBuffers_t);
```

```
  if (pxBuffer == 0)
  {
   return TRC_FAIL;
  }

  return xTraceInternalEventBufferInitialize(pxBuffer->buffer,
sizeof(pxBuffer->buffer));
}

traceResult prvTraceUARTTransmit(void* pvData, uint32_t uiSize, int32_t*
piBytesSent)
{
  int32_t error_code = -1;

  while(1)
  {
   if(wait_sending)
   {
      xTraceKernelPortDelay(1);
   }
   else
   {
      break;
   }
  }

  if(uiSize < sizeof(string))
  {
   memcpy(string, pvData, uiSize);
   if(SCI_SUCCESS == R_SCI_Send(sci_handle_tracealyzer, string, uiSize))
   {
      wait_sending = 1;
      *piBytesSent = uiSize;
      error_code = 0;
   }
  }
  return error_code;
}

traceResult prvTraceUARTReceive(void* data, uint32_t uiSize, int32_t*
piBytesReceived)
{
  if(sci_current_received_size == uiSize)
  {
   memcpy(data, sci_buffer, sci_current_received_size);
   *piBytesReceived = sci_current_received_size;
   sci_current_received_size = 0;
  }
  return 0;
}

void sci_callback_tracealyzer(void *arg)
{
  sci_cb_args_t   *p_args;
  p_args = (sci_cb_args_t *)arg;

  if (SCI_EVT_RX_CHAR == p_args->event)
  {
```

```
    R_SCI_Receive(p_args->hdl, &sci_buffer[sci_current_received_size], 1);
    if(sci_current_received_size == (sizeof(sci_buffer) - 1)) /* -1 means
string terminator after "\n" */
    {
       sci_current_received_size = 0;
    }
    else
    {
       sci_current_received_size++;
    }
  }
  else if(SCI_EVT_TEI == p_args->event)
  {
    wait_sending = 0;
  }
}

#endif
#endif
```

Copy the following code to "trcStreamPortConfig.h".

```
#ifndef TRC_STREAM_PORT_CONFIG_H
#define TRC_STREAM_PORT_CONFIG_H

#ifdef __cplusplus
extern "C" {
#endif

/*************************************************************************
******
* Configuration Macro: TRC_CFG_STREAM_PORT_INTERNAL_BUFFER_SIZE
*
* Specifies the size of the internal buffer.
*************************************************************************
****/
#define TRC_CFG_STREAM_PORT_INTERNAL_BUFFER_SIZE 1024

#ifdef __cplusplus
}
#endif

#endif /* TRC_STREAM_PORT_CONFIG_H */
```

Copy the following code to " trcStreamPort.h".

```
#ifndef TRC_STREAM_PORT_H
#define TRC_STREAM_PORT_H

#include <trcTypes.h>
#include <trcStreamPortConfig.h>

#ifdef __cplusplus
extern "C" {
#endif

typedef struct TraceStreamPortBuffer
{
  uint8_t buffer[(TRC_CFG_STREAM_PORT_INTERNAL_BUFFER_SIZE) +
sizeof(TraceUnsignedBaseType_t)];
} TraceStreamPortBuffer_t;

traceResult prvTraceUARTReceive(void* data, uint32_t uiSize, int32_t*
piBytesReceived);

traceResult prvTraceUARTTransmit(void* pvData, uint32_t uiSize, int32_t*
piBytesSent);

/**
 * @internal Stream port initialize callback.
 *
 * This function is called by the recorder as part of its initialization
phase.
 *
 * @param[in] pxBuffer Buffer
 *
 * @retval TRC_FAIL Initialization failed
 * @retval TRC_SUCCESS Success
 */
traceResult xTraceStreamPortInitialize(TraceStreamPortBuffer_t* pxBuffer);

/**
 * @brief Allocates data from the stream port.
 *
 * @param[in] uiSize Allocation size
 * @param[out] ppvData Allocation data pointer
 *
 * @retval TRC_FAIL Allocate failed
 * @retval TRC_SUCCESS Success
 */
#define xTraceStreamPortAllocate(uiSize, ppvData) ((void)uiSize,
xTraceStaticBufferGet(ppvData))

/**
 * @brief Commits data to the stream port, depending on the
implementation/configuration of the
 * stream port this data might be directly written to the stream port
interface, buffered, or
 * something else.
 *
 * @param[in] pvData Data to commit
 * @param[in] uiSize Data to commit size
 * @param[out] piBytesCommitted Bytes committed
```

```
 *
 * @retval TRC_FAIL Commit failed
 * @retval TRC_SUCCESS Success
 */
#define xTraceStreamPortCommit xTraceInternalEventBufferPush

/**
 * @brief Writes data through the stream port interface.
 *
 * @param[in] pvData Data to write
 * @param[in] uiSize Data to write size
 * @param[out] piBytesWritten Bytes written
 *
 * @retval TRC_FAIL Write failed
 * @retval TRC_SUCCESS Success
 */
#define xTraceStreamPortWriteData prvTraceUARTTransmit

/**
 * @brief Reads data through the stream port interface.
 *
 * @param[in] pvData Destination data buffer
 * @param[in] uiSize Destination data buffer size
 * @param[out] piBytesRead Bytes read
 *
 * @retval TRC_FAIL Read failed
 * @retval TRC_SUCCESS Success
 */
#define xTraceStreamPortReadData prvTraceUARTReceive

#define xTraceStreamPortOnEnable(uiStartOption) ((void)(uiStartOption),
TRC_SUCCESS)

#define xTraceStreamPortOnDisable() (TRC_SUCCESS)

#define xTraceStreamPortOnTraceBegin() (TRC_SUCCESS)

#define xTraceStreamPortOnTraceEnd() (TRC_SUCCESS)

#ifdef __cplusplus
}
#endif

#endif /* TRC_STREAM_PORT_H */
```

Nothing need be written to "Readme-Streamport.txt".

Add the following statement at the end of "FreeRTOSConfig.h".

```
#include "trcRecorder.h"
```

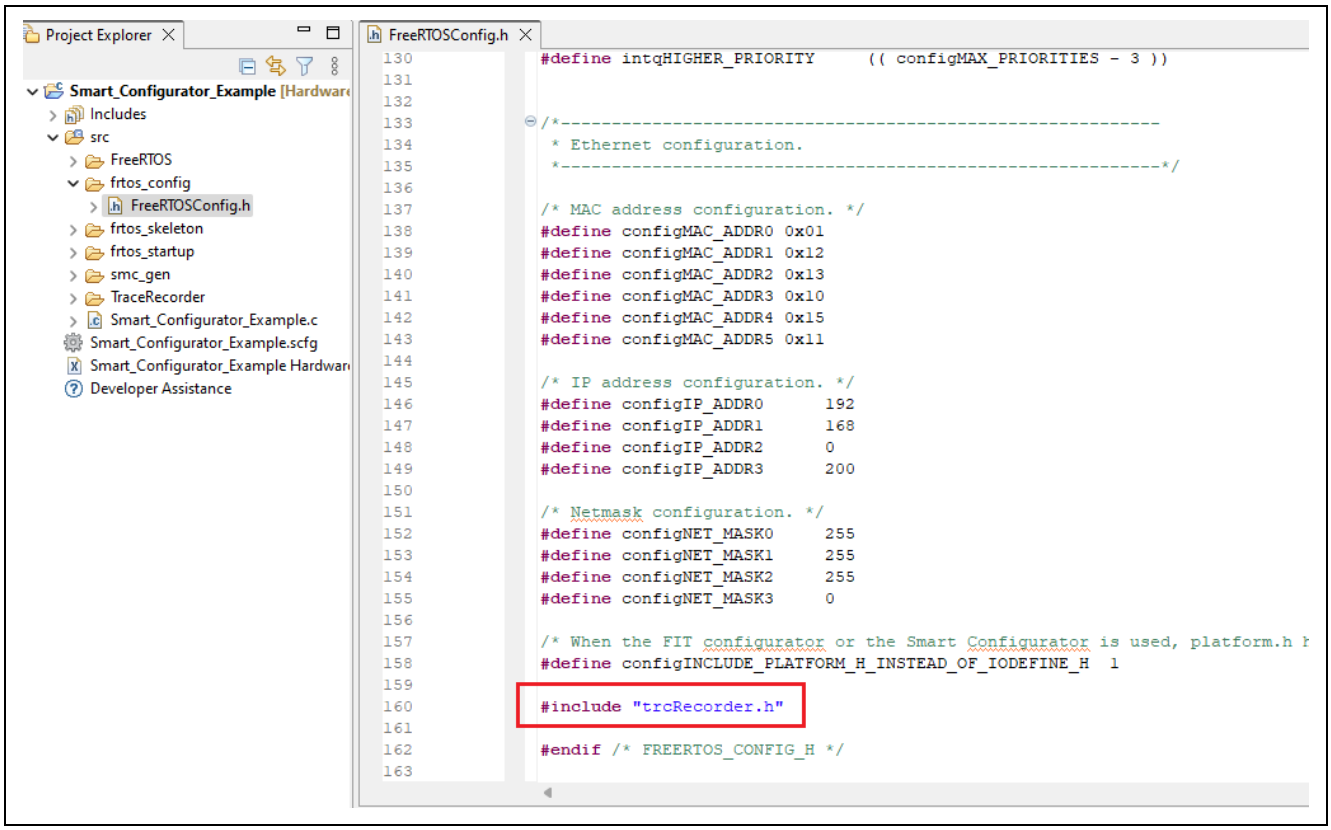

Figure 3-4 Adding a Statement to "FreeRTOSConfig.h"

Modify "trcConfig.h" as follows.

- Comment out the line starting with "#error ... ".
- Specify "TRC_HARDWARE_PORT_Renesas_RX600" for TRC_CFG_HARDWARE_PORT.

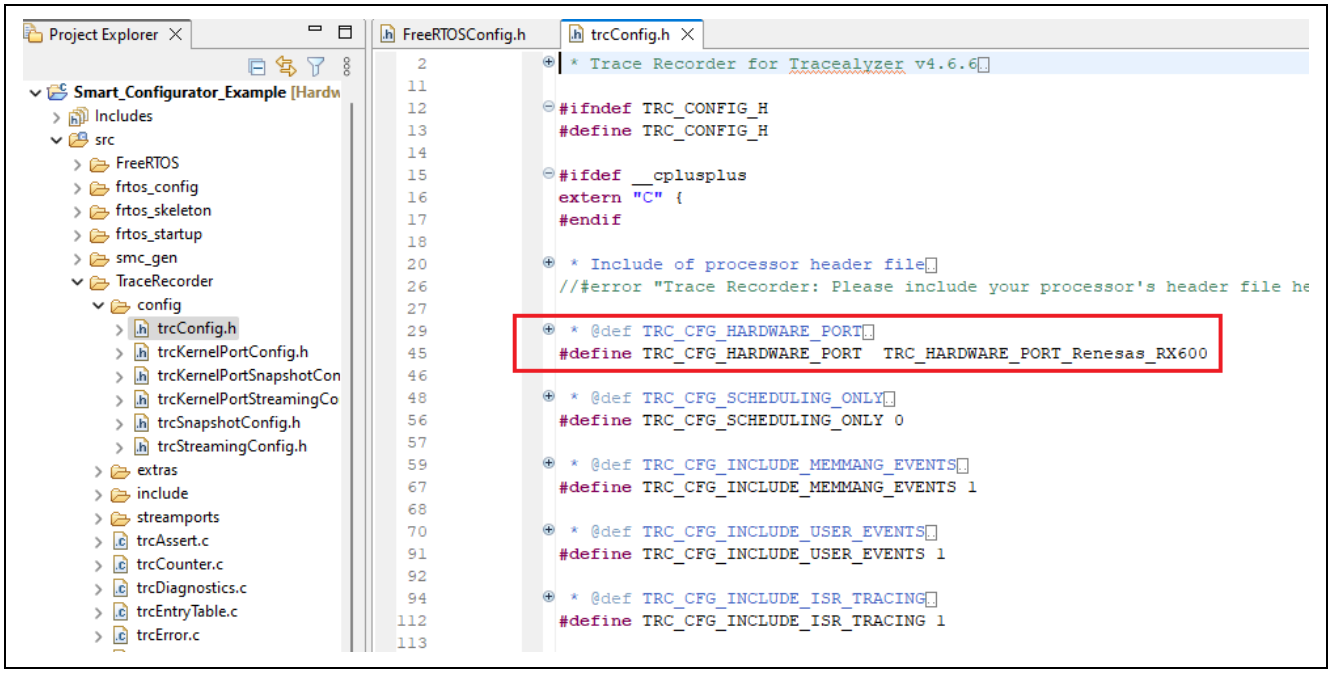

Figure 3-5 Modifying "trcConfig.h"

Modify "trcKernelPortConfig.h" as follows.

- Specify "TRC_RECORDER_MODE_STREAMING" for TRC_CFG_RECORDER_MODE.
- Specify "TRC_FREERTOS_VERSION_10_4_1" for TRC_CFG_FREERTOS_VERSION.

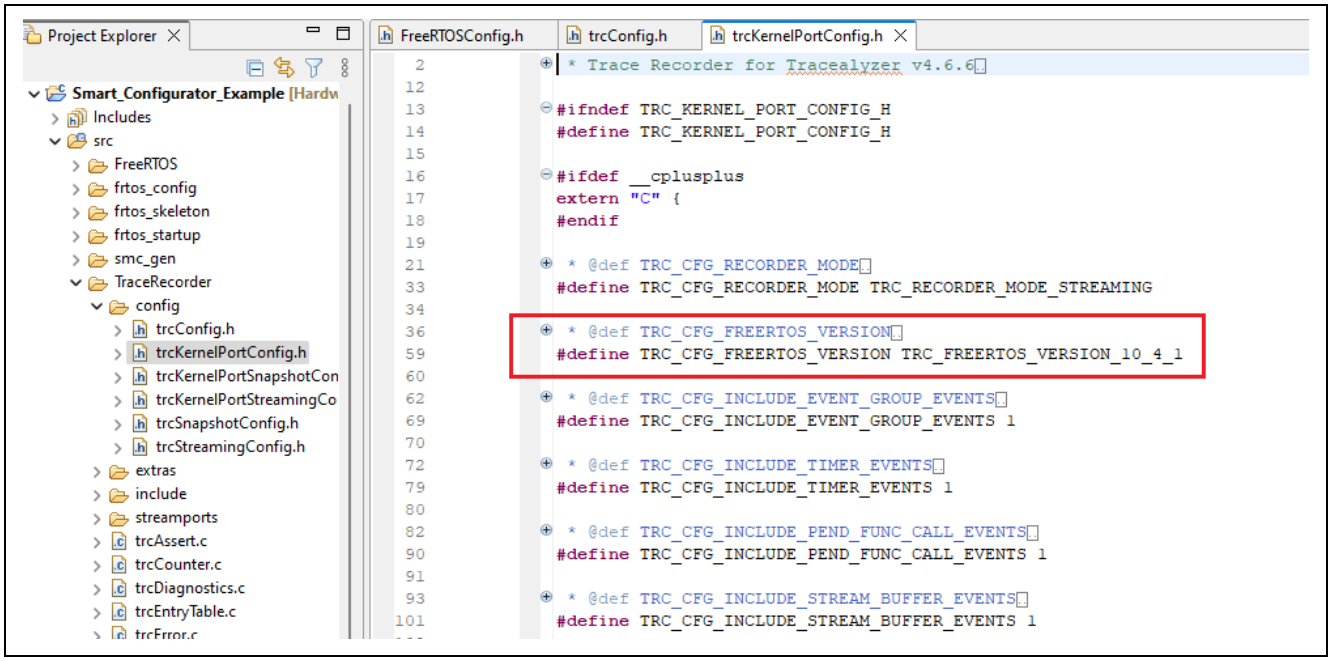

Figure 3-6 Modifying "trcKernelPortConfig.h"

3.2 Settings of the Project

3.2.1 Setting the UART to Output Data Monitored by Tracealyzer®

Use the Smart Configurator to add the FIT module for the SCI.

Open the [Software Component Selection] dialog box, enter "SCI" in the [Filter] textbox, select [SCI Driver], and click on [Finish] as shown in Figure 3-7.

If the FIT module for the SCI does not appear even after "SCI" has been entered for [Filter], select [Download the latest FIT drivers and middleware] and add the SCI module after having downloaded the module.

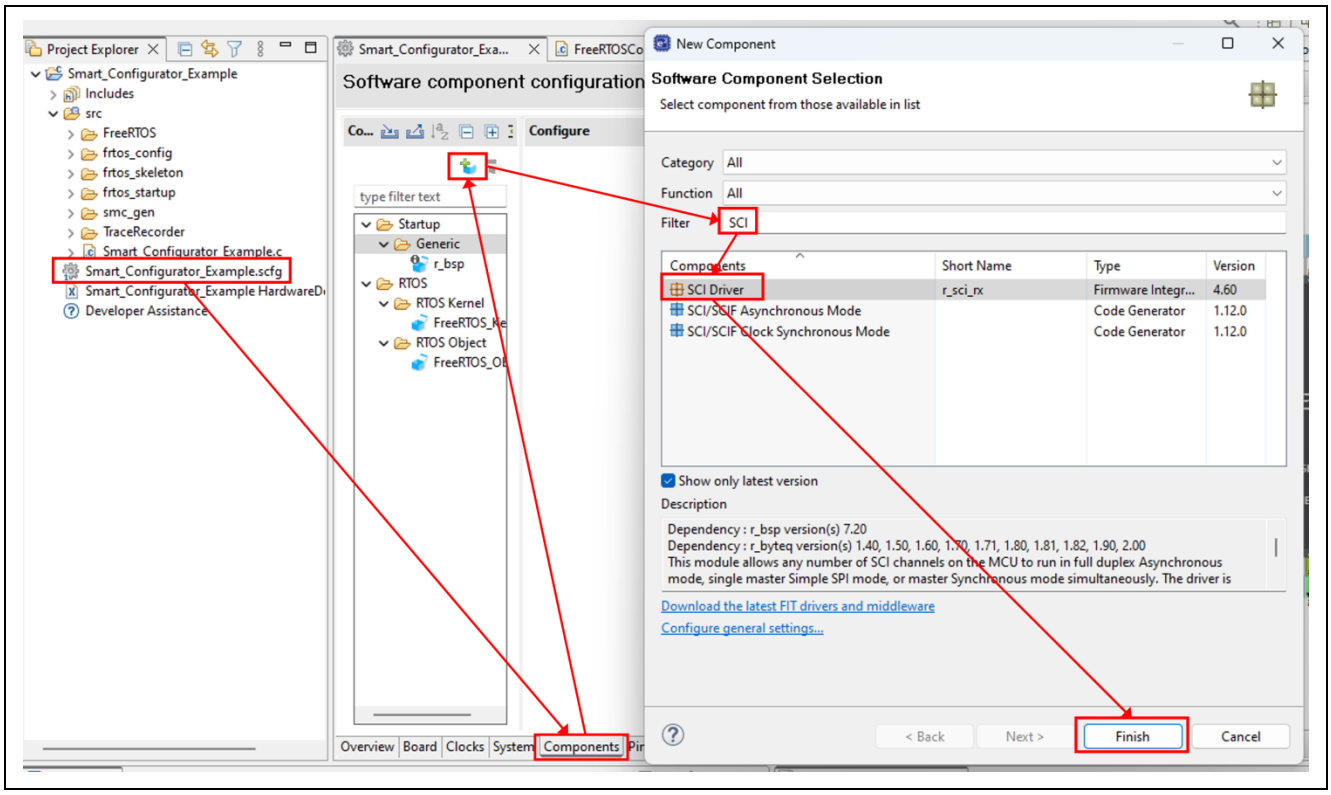

Figure 3-7 Adding the FIT Module for the SCI

To use PMOD1 of the CK-RX65N board, set up SCI channel 6.
Select [r_sci_rx] on the [Components] tabbed page and set [Include] for SCI channel 6.

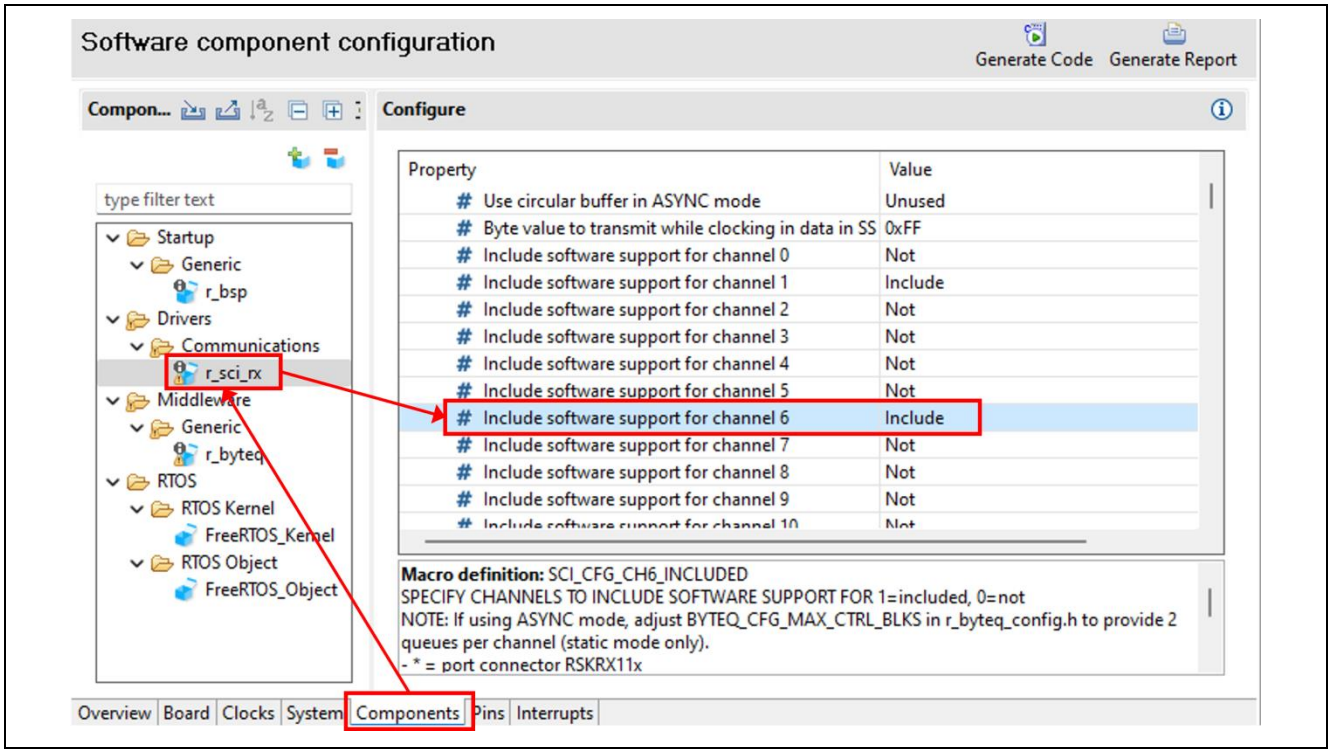

Figure 3-8 Setting SCI Channel 6

Change the size of the transmission buffer for channel 6 from 80 bytes to 1024 bytes.

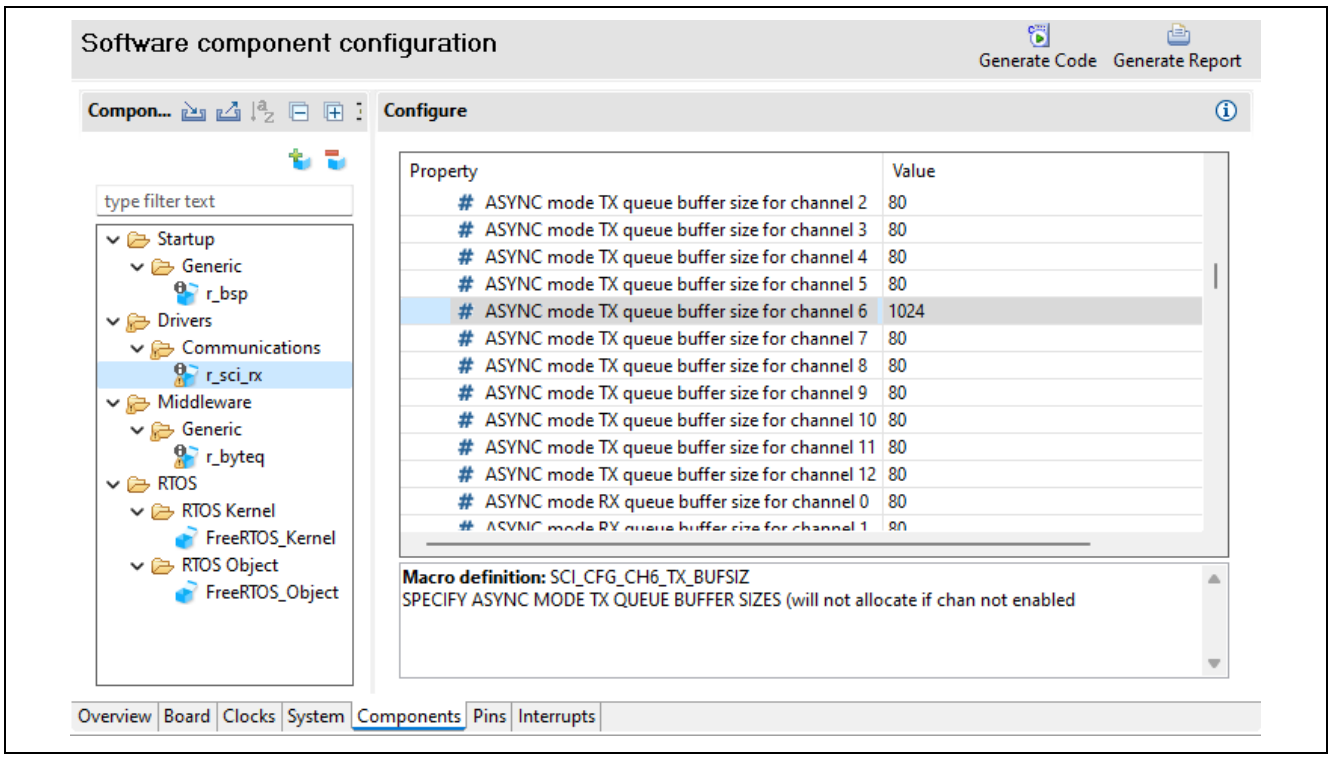

Figure 3-9 Modifying the Transmission Buffer for SCI Channel 6

Enable transmission buffer empty interrupts from channel 6.

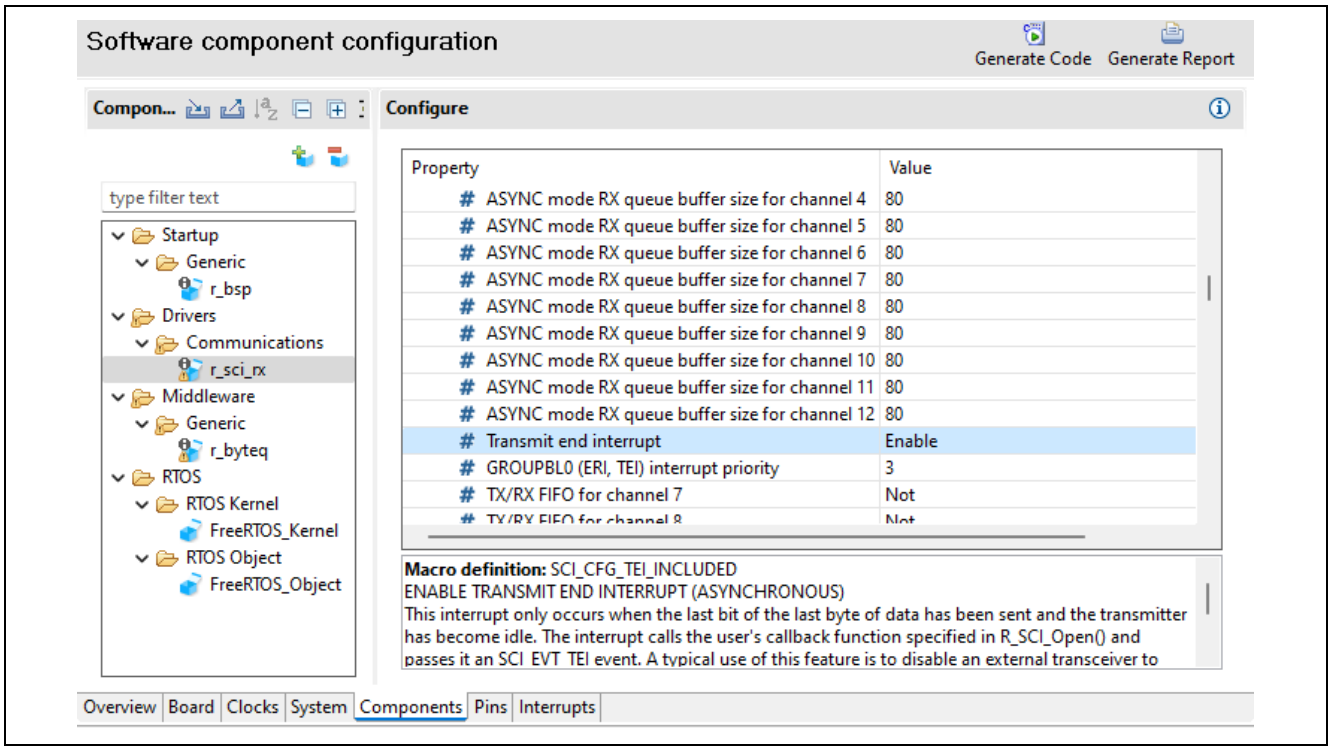

Figure 3-10 Setting Interrupts from SCI Channel 6

To use channel 6 as a UART without flow control, disable the flow control pins (RTS and CTS) and only enable the transmission and reception pins (TxD and RxD).

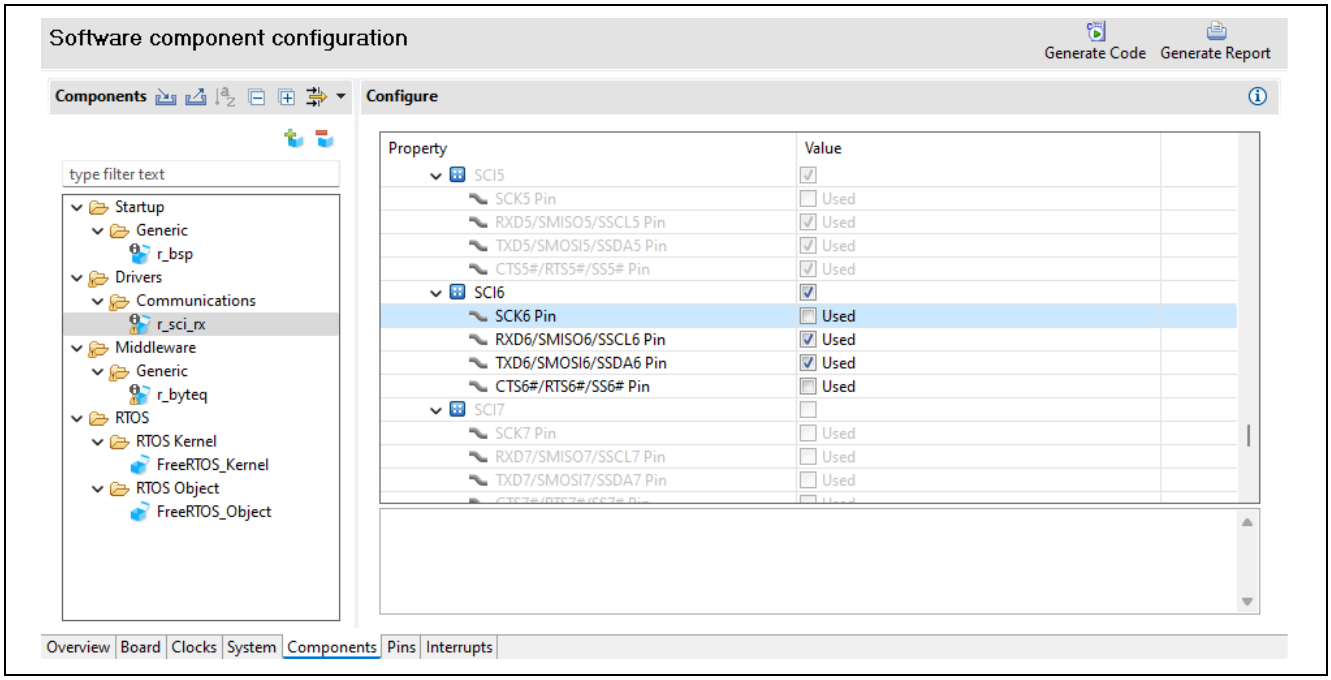

Figure 3-11 Setting the Pins for SCI Channel 6

Set the pin functions for SCI channel 6 on the [Pins] tabbed page.

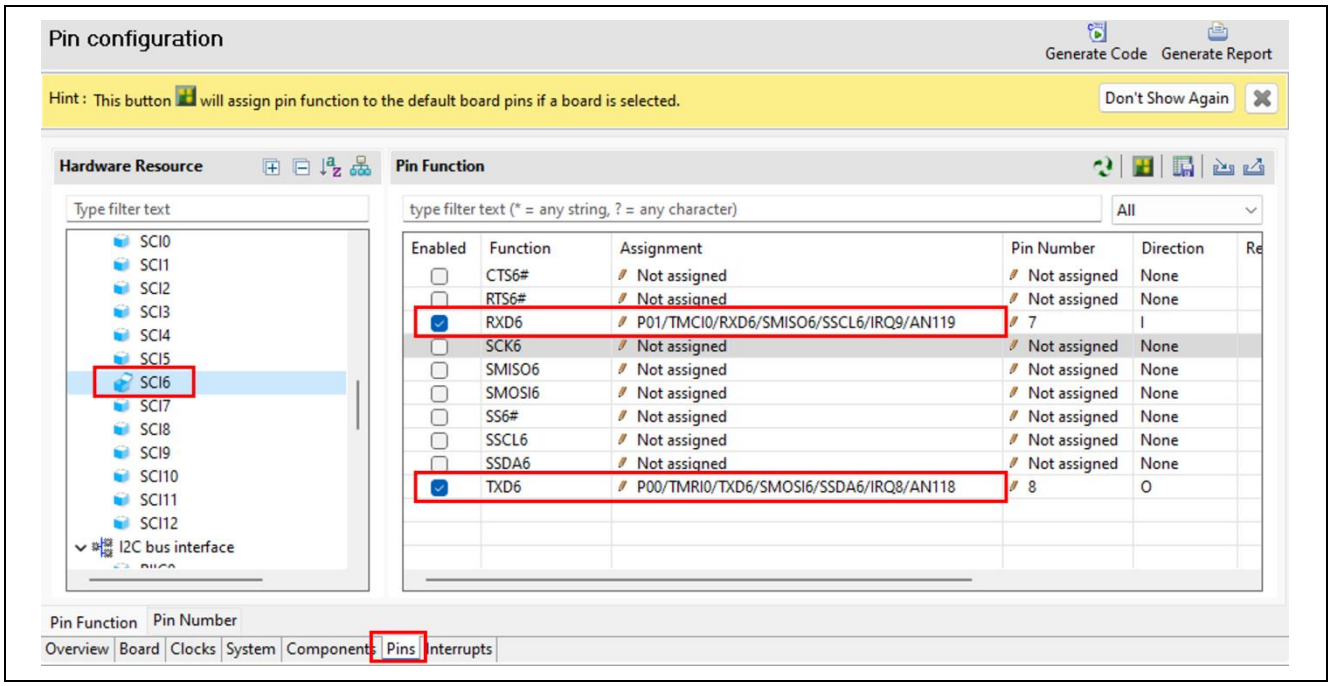

Figure 3-12 Setting the Pin Functions for SCI Channel 6

3.3 Settings of the Compiler

3.3.1 Adding the Include Paths Required by Tracealyzer® through Compiler Settings

Right-click on the project name in the Project Explorer and select [Properties].

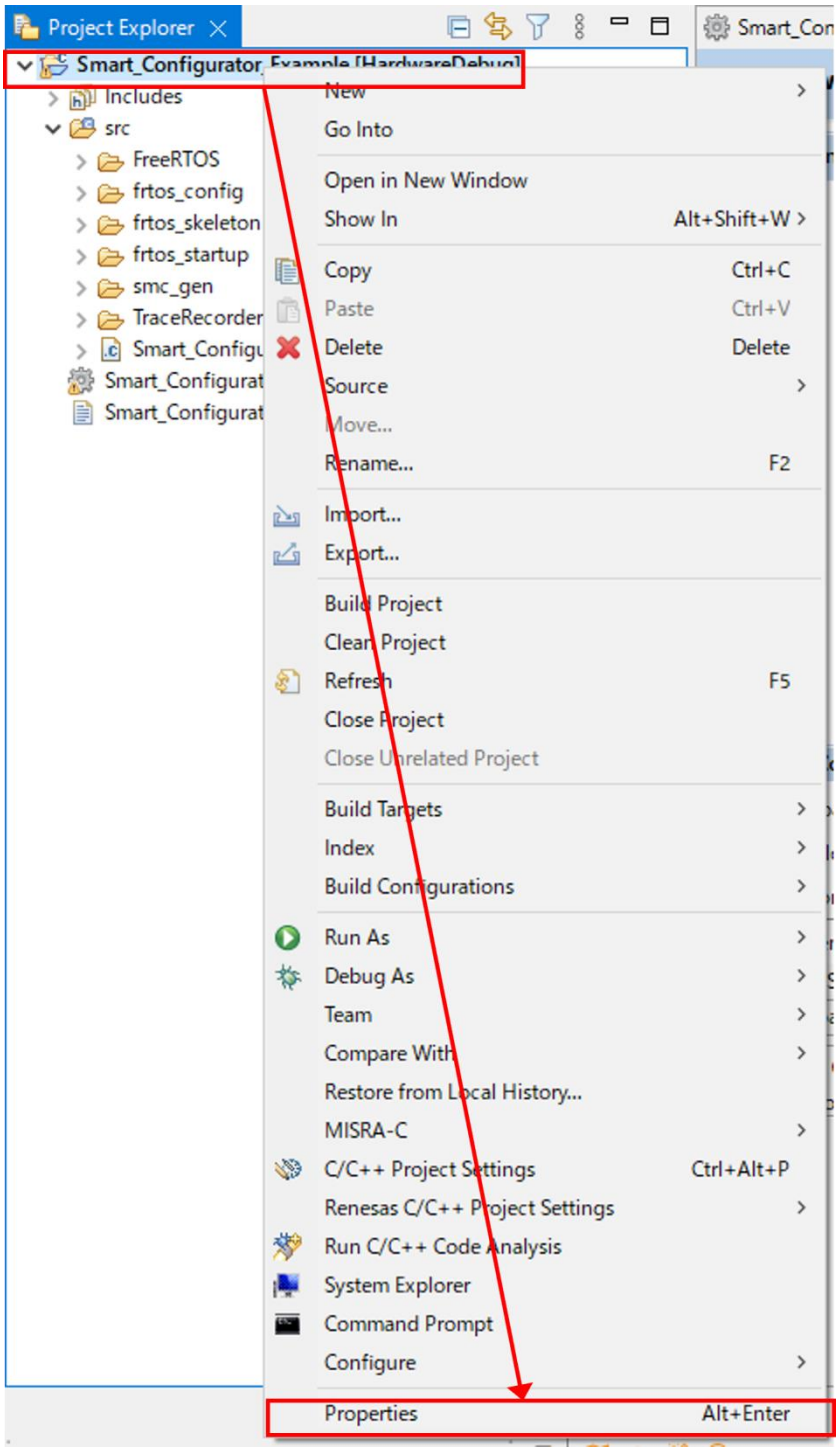

Figure 3-13 Project Properties

Select [C/C++ Build] → [Settings] → [Tool Settings] → [Compiler] → [Source] and click on the [Add] button.

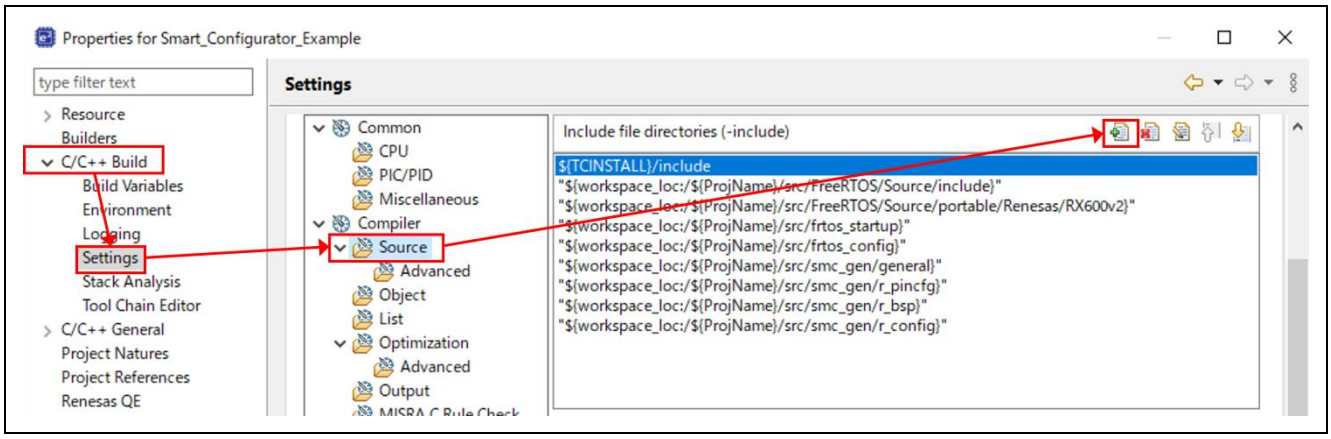

Figure 3-14 Adding Paths

Add the following five paths.

"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp/mcu/rx65n/register_access/ccrx}"

"${workspace_loc:/${ProjName}/src/TraceRecorder/config}"

"${workspace_loc:/${ProjName}/src/TraceRecorder/include}"

"${workspace_loc:/${ProjName}/src/TraceRecorder/streamports/Renesas_RX_UART/config}"

"${workspace_loc:/${ProjName}/src/TraceRecorder/streamports/Renesas_RX_UART/include}"

Note:
The e^2 studio deletes "${workspace_loc:/${ProjName}/src/smc_gen/r_bsp/mcu/rx65n/register_access/ccrx}" every time the Smart Configurator generates code. Be sure to specify the same path again after every round of code generation.

3.4 Settings of FreeRTOS

3.4.1 Modifying "portmacro.h" of the FreeRTOS Kernel

Modify "portmacro.h" of the FreeRTOS kernel to support calls from Tracealyzer®.

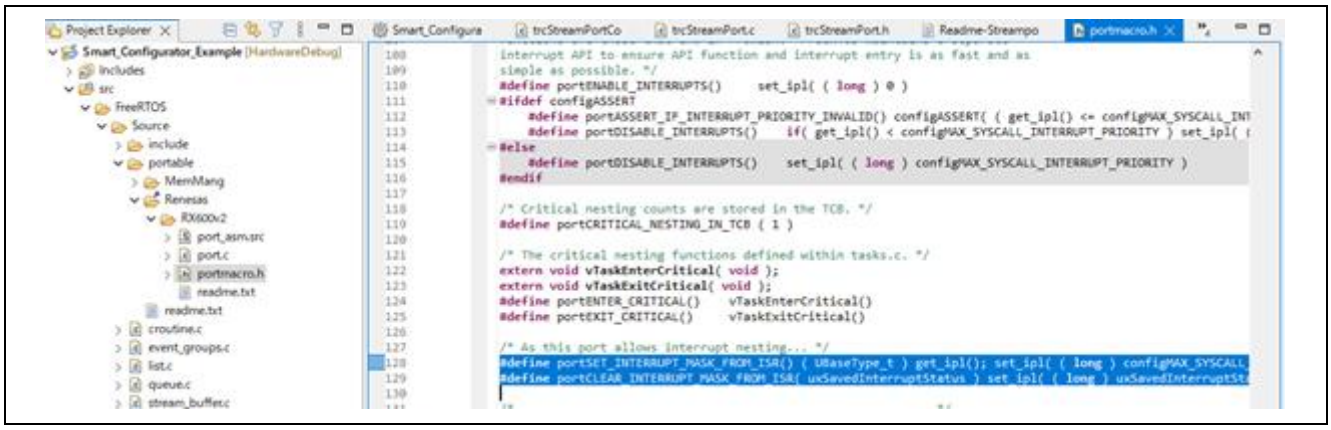

Figure 3-15 portmacro.h

Modify the code under /* As this port allows interrupt nesting... */ as follows.

```
/* As this port allows interrupt nesting... */
       static int32_t set_interrupt_mask_from_isr( void );
       static int32_t set_interrupt_mask_from_isr( void )
       {
          int32_t tmp = __get_ipl();
          __set_ipl( ( long ) configMAX_SYSCALL_INTERRUPT_PRIORITY );
          return tmp;
       }
       #define portSET_INTERRUPT_MASK_FROM_ISR()
set_interrupt_mask_from_isr()
       #define portCLEAR_INTERRUPT_MASK_FROM_ISR( uxSavedInterruptStatus )
set_ipl( ( long ) uxSavedInterruptStatus )
```

3.4.2 Modifying the Hook Function to be Executed before the Startup of the FreeRTOS Kernel

Add the code for initializing Tracealyzer® and the SCI to the hook function (Processing_Before_Start_Kernel() in "freertos_start.c") to be executed before the FreeRTOS kernel is started.

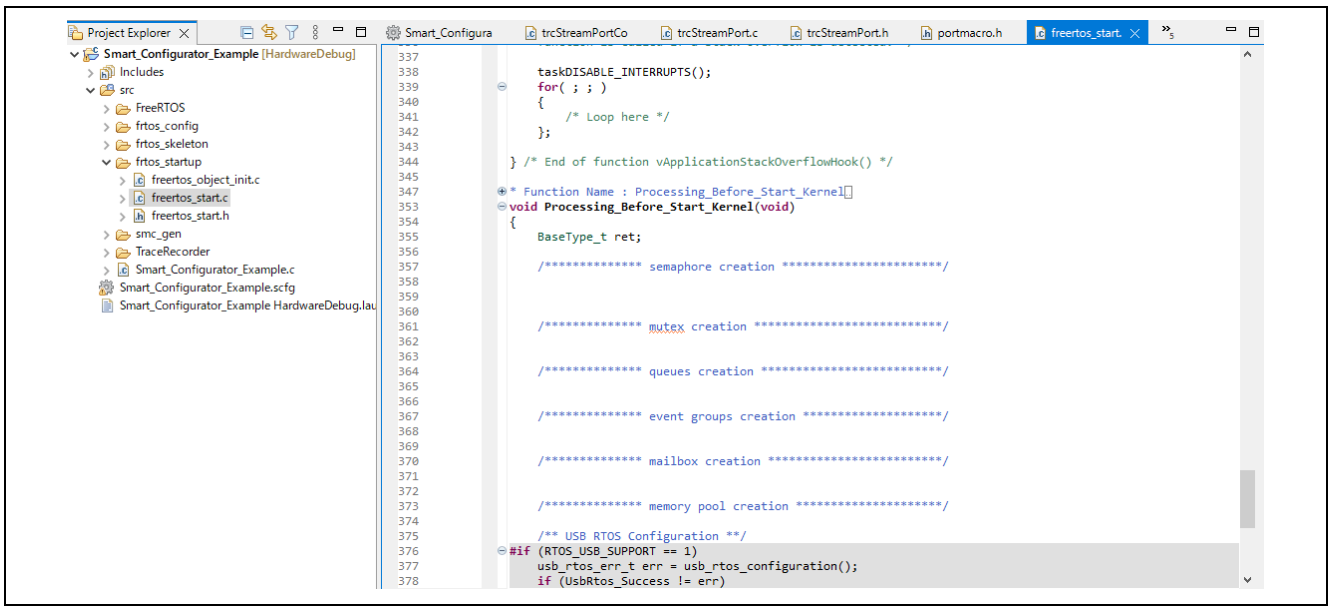

Figure 3-16 freertos_start.c

```
#include "r_sci_rx_if.h"
#include "r_sci_rx_pinset.h"

static sci_cfg_t my_sci_config;
sci_hdl_t sci_handle_tracealyzer;

extern void sci_callback_tracealyzer(void *arg);

void Processing_Before_Start_Kernel(void)
{
    BaseType_t ret;

    /* Create all other application tasks here */
    /* Set up the configuration data structure for asynchronous (UART)
operation. */
    my_sci_config.async.baud_rate    = 921600;
    my_sci_config.async.clk_src     = SCI_CLK_INT;
    my_sci_config.async.data_size    = SCI_DATA_8BIT;
    my_sci_config.async.parity_en    = SCI_PARITY_OFF;
    my_sci_config.async.parity_type  = SCI_EVEN_PARITY;
    my_sci_config.async.stop_bits    = SCI_STOPBITS_1;
    my_sci_config.async.int_priority = 15; /* disable 0 - low 1 - 15 high */

    R_SCI_Open(SCI_CH6, SCI_MODE_ASYNC, &my_sci_config,
sci_callback_tracealyzer, &sci_handle_tracealyzer);
    R_SCI_PinSet_SCI6();

    xTraceInitialize();
```

3.4.3 Adding the Code for Starting Tracealyzer® to the main Task

Add the line "xTraceEnable(TRC_START);" for starting Tracealyzer® to the main task (Smart_Configurator_Example.c).

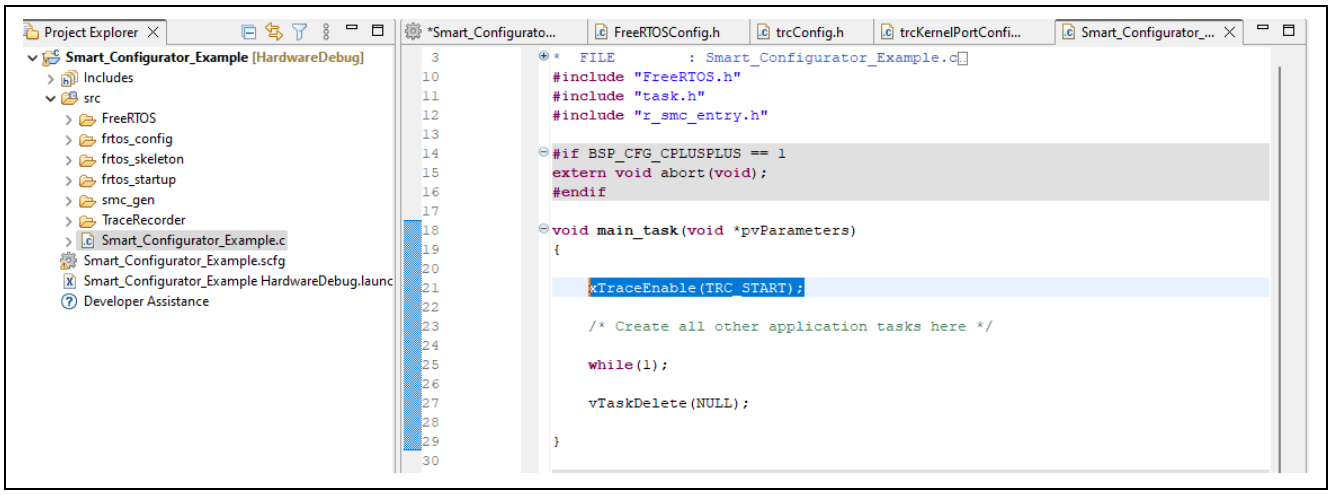

Figure 3-17 main Task

Setting the heap size of the project to at least 128 Kbytes is recommended when Tracealyzer® is to be used.

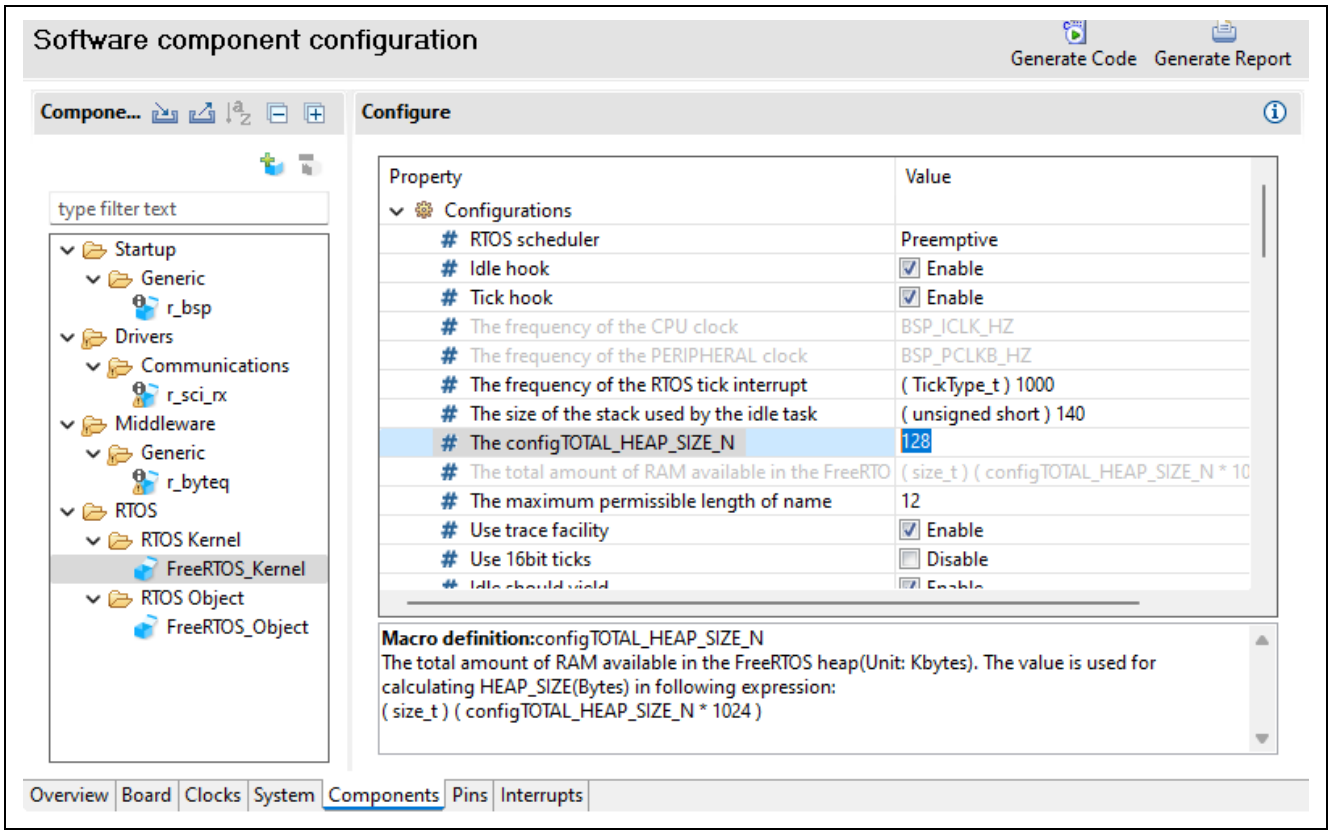

Figure 3-18 Modifying the Heap Size

3.4.4 Building the Project

Right-click on the project and select [Build Project]. On completion of the build process, check that no errors have occurred.

3.5 Connecting the Host PC and CK-RX65N Board

Connect the host PC and CK-RX65N board through a Pmod USBUART module (from Digilent, Inc.).

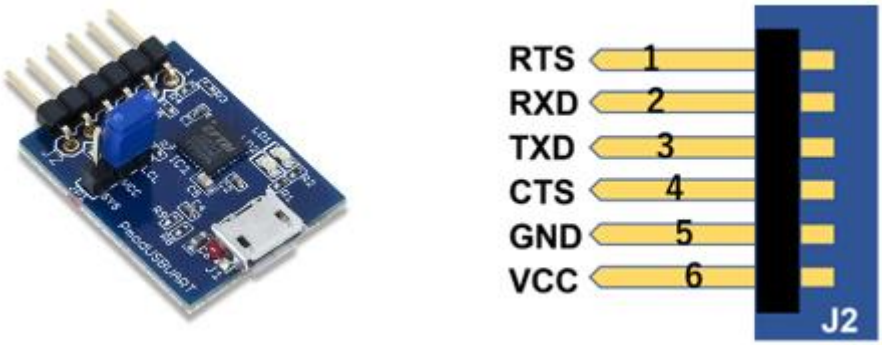

Figure 3-19 External Appearance and Pin Arrangement of the Pmod USBUART Module (from Digilent, Inc.)

Connect pins 1 to 6 of the Pmod USBUART module to pins 1 to 6 in the top row of the Pmod1 connector on the CK-RX65N board.

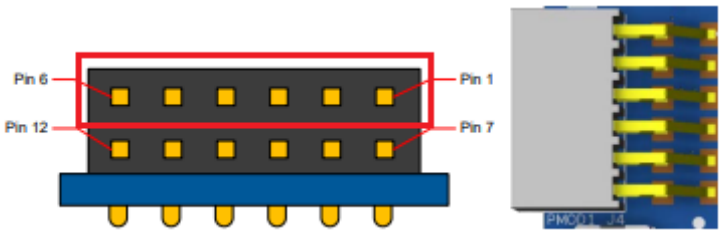

Figure 3-20 Pmod1 Connector on the CK-RX65N Board

The following table lists the hardware settings for debugging.

Table 3-1 Jumper Settings

Jumper	Setting	Function
J15	Open	E2OB normal debugging mode
J16	Close pins 1 and 2	Debugger is enabled.

Connect the USB connector on the Pmod module and the USB connector for use in debugging on the CK-RX65N board to the host PC.

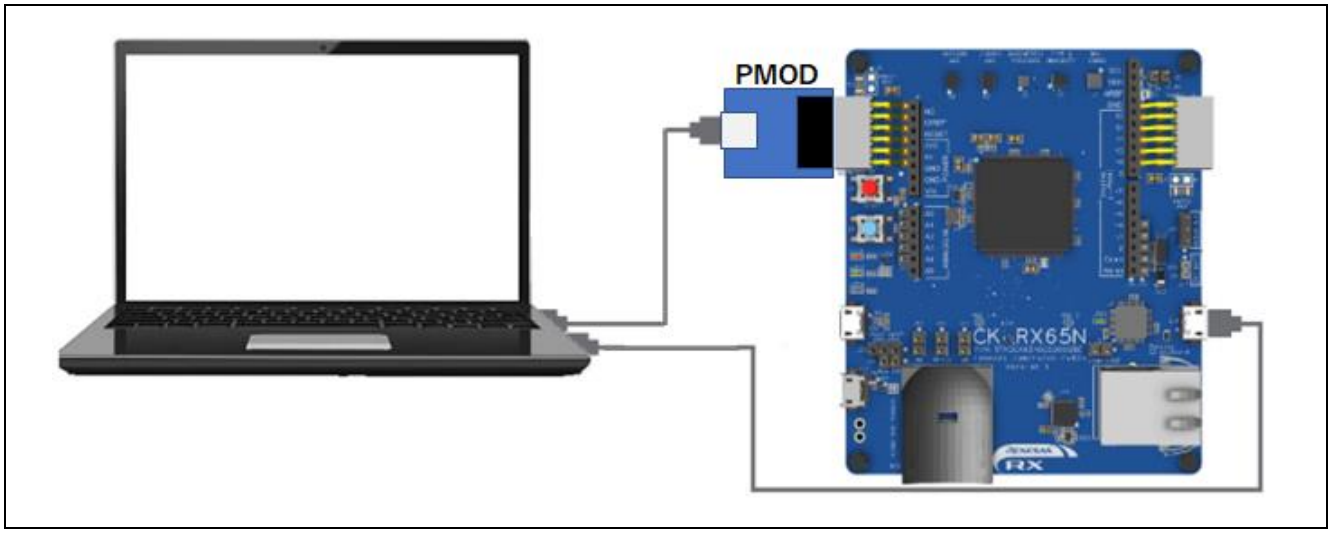

Figure 3-21 Connections between the Host PC and CK-RX65N Board

Supplementary Information:

This system occupies the RX65N MCU's SCI channel 6, which is connected to the PMOD module.

Configuration of this system is based on "Custom Streaming" as described at the destination of the link below to the Tracealyzer® document.

According to the Tracealyzer® document, the rate of monitoring data generation in the MCU and output of the data by Tracealyzer® is from 20 to 200 Kbytes/s.

The Pmod USBUART module introduced in this application note can support a bit rate of 921600 bps (= 112.5 Kbytes/s), so the monitored data may be incomplete when the system is using a complex configuration of tasks.
In such a case, consider a faster interface (such as Ethernet) for output of the monitored data.

Refer to the following Tracealyzer® document as well as this application note.

Percepio Tracealyzer® Documentation

3.6 Using the [RTOS Resources] View

The e^2 studio has an RTOS resource view function that displays the state of FreeRTOS resources. The following is a description of the procedure for using the [RTOS Resources] view.

3.6.1 Displaying the [RTOS Resources] View

The [RTOS Resources] view is only available while the debugger is running. Start the debugger and then select [Renesas Views] → [Partner OS] → [RTOS Resources]. After the [Select OS] dialog box is displayed, select "FreeRTOS" as shown in Figure 3-22. The [RTOS Resources] view will appear as shown in Figure 3-23.

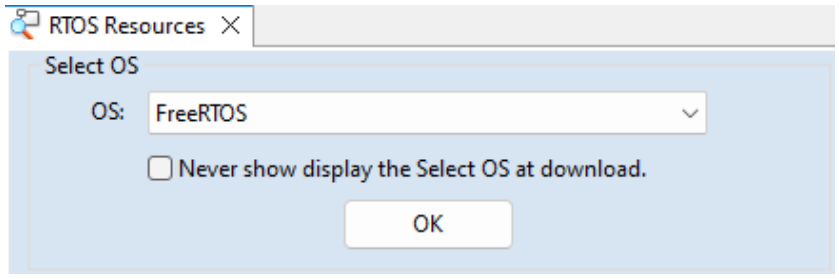

Figure 3-22 Selecting the OS

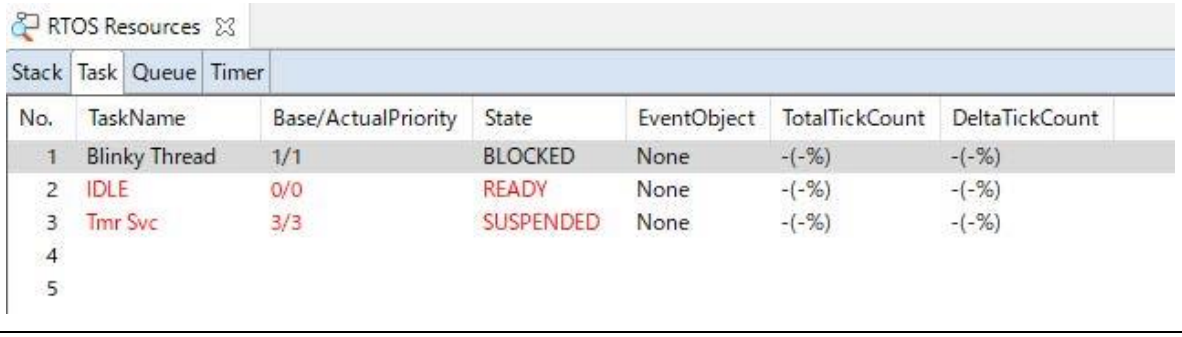

Figure 3-23 [RTOS Resources] View

3.6.2 Context Menu

Display the context menu by right-clicking on the mouse with the cursor in the [RTOS Resources] view.

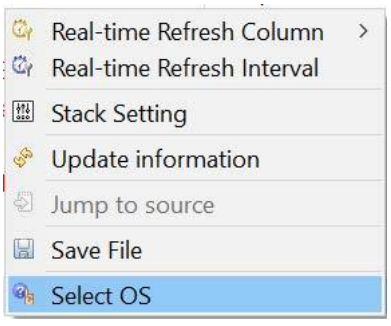

Figure 3-24 Context Menu

Explanation:

- **Real-time Refresh Column:**
  Enables or disables real-time updating of information displayed in the individual columns (tabbed pages).
  This is grayed out and not selectable while the program is running.
- **Real-time Refresh Interval:**
  Specifies the interval for real-time updating of the display. The specifiable values are in the range from 500 ms to 10000 ms.
  This is grayed out and not selectable while the program is running.
- **Stack Setting:**
  Enables or disables loading of the stack data and specifies the threshold for the stack warning function.
  This is grayed out and not selectable while the program is running.
- **Update information:**
  Updates the displayed information.
- **Jump to source:**
  Opens an editor view displaying the source code of the task/thread or handler. Double-clicking on a task/thread or a handler also opens an editor view.
  This is grayed out and not selectable while the program is running.
- **Save File:**
  Saves the data on the currently selected tabbed page in a text file (*.txt).
  This is grayed out and not selectable while the program is running.
- **Select OS:**
  Opens the [Select OS] dialog box.
  This is grayed out and not selectable while the program is running.

3.6.3 Stack Setting

This is for enabling the loading of stack data and setting the stack threshold.

1. Open the context menu and select [Stack Setting].
2. To load stack data to the [RTOS Resources] view, check the [Enable loading Stack data] checkbox in the [Stack Setting] dialog box. If this option is not enabled, stack data will not be loaded in the next debugging session.

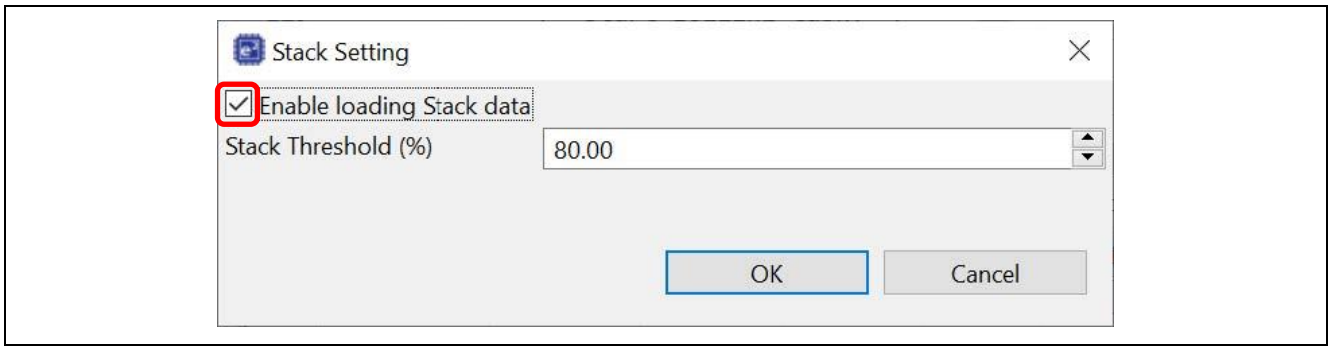

Figure 3-25 Enabling Loading of Stack Data

3. A desired threshold value can be set in the [Stack Threshold (%)] textbox. Click on [OK] to save the setting.

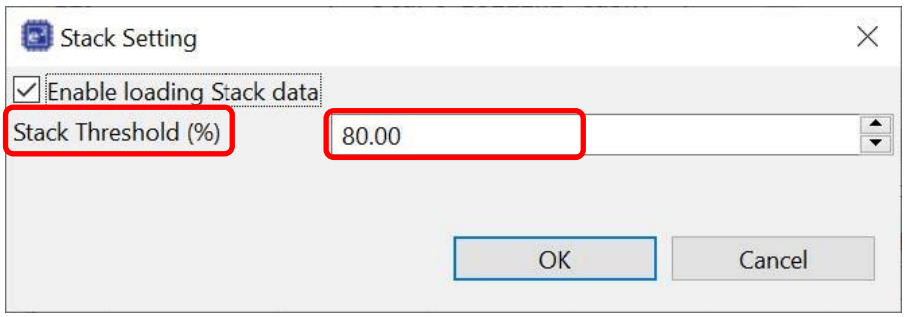

Figure 3-26 Setting the Stack Threshold

4. Run the target project and then suspend it to load the stack data. The stack threshold warning will pop up if the set threshold is reached.

5. There are two types of popup warning: [Stack Threshold Warning] (with a list of the threads that have used stack space up to the specified threshold) and [Stack Overflow Warning] (with a list of threads that have used 100% of the stack).

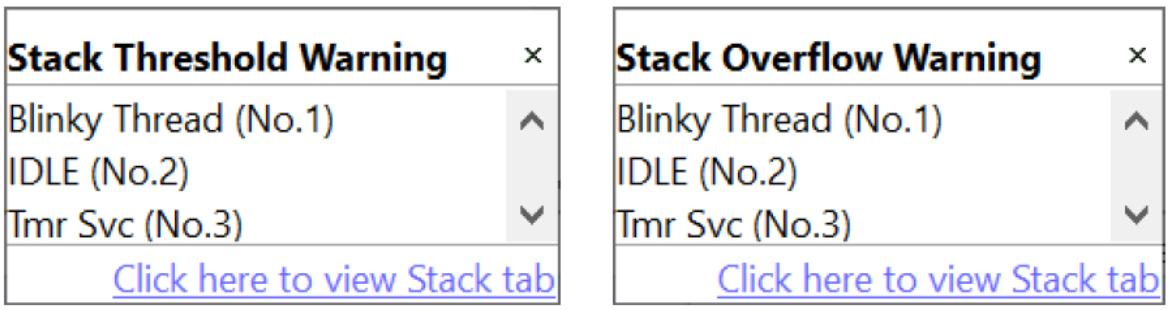

Figure 3-27 [Stack Threshold Warning] Popup (Left) and [Stack Overflow Warning] Popup (Right)

3.6.4 Tabbed Pages

Table 3-2 lists the items displayed on the individual tabbed pages.

Table 3-2 Contents of Individual Tabbed Pages

Name of the Tabbed Page in the [RTOS Resources] View	Name of Displayed Information and Selection	Information to be Displayed
Stack	No.	Row index
	TaskName	Name assigned to the task upon creation
	StartOfStack	Address of the beginning of the stack
	EndOfStack	Address of the end of the stack
	TopOfStack	Address of the top of the stack area when the contents of the stack were saved; that is, the address of the last location to which writing had proceeded
	StackSize(bytes)	Total stack size
	StackUsageSize	Maximum amount of stack usage in bytes
	StackUsageRatio	Percentage of maximum usage relative to the total stack size
Task	No.	Row index
	TaskName	Name assigned to the task upon creation
	Base/ActualPriority	Base priority used by the priority inheritance mechanism and actual priority used by the task
	State	State of the task: “RUNNING”, “READY”, “BLOCKED”, or “SUSPENDED”
	EventObject	Name of the queue that has caused blocking of the task
	TotalTickCount	Total number of ticks until the task becomes active
	DeltaTickCount	Number of ticks until the task becomes active after a previous suspension event
Queue	No.	Row index
	Name(Type)	Name assigned to the queue upon registration and its type (Queue, Semaphore, or Mutex)
	Address	Address of the queue handle
	MaxLength	Maximum number of items that can be stored in the queue
	ItemSize	Size per item in the queue (in bytes)
	CurrentLength	Number of items currently stored in the queue
	#WaitingTx	Number of tasks blocked while waiting for transmission to the queue
	#WaitingRx	Number of tasks blocked while waiting for reception from the queue
Timer	No.	Row index
	Name	Current period of the timer (in system ticks)
	Period	Enabling or disabling of automatic reloading. On: Automatic reloading is enabled. The timer is reset each time the timer period expires. Off: Automatic reloading is disabled. The timer does nothing when the timer period expires.
	CallbackFn	Address and name of the callback function to be executed each time the timer period expires.
	TimerID	Numeric ID (in hexadecimal) assigned to the timer when it was created

3.7 Starting Debugging of a Project with Tracealyzer®

3.7.1 Launching the Debugger on the e² studio

Select the [Run] menu → [Debug Configurations] → [Debugger] tabbed page → [Connection Settings] tabbed page and set [Power Target From The Emulator] to "No".

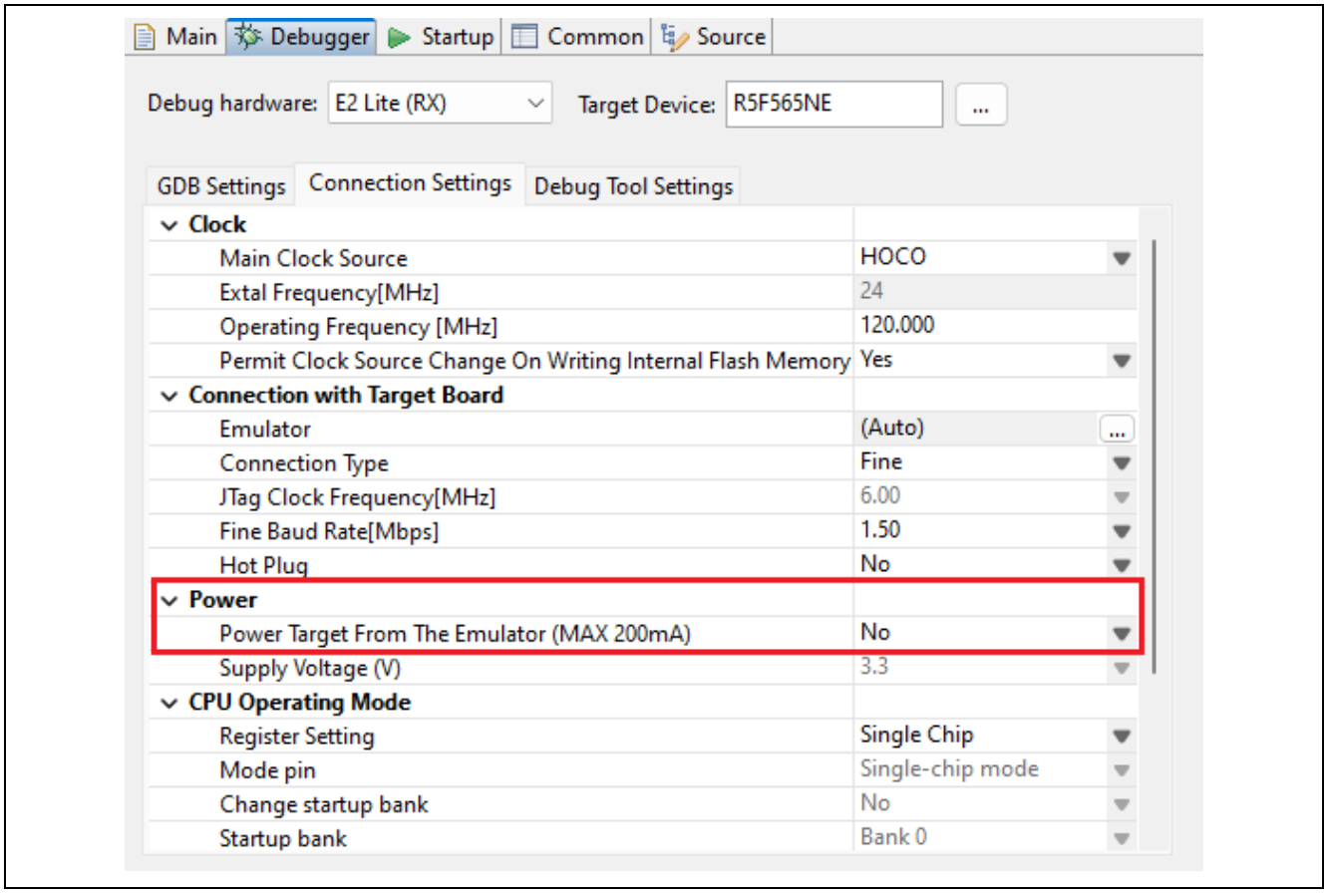

Figure 3-28 Modifying the Connection Settings

Select [Debug] from the [Run] menu to launch the debugger.

3.7.2 Launching Tracealyzer®

Launch Tracealyzer®.

Click on [Recording Settings] in the Tracealyzer® window, select [PSF Streaming Settings], and make the settings listed following the figure below.

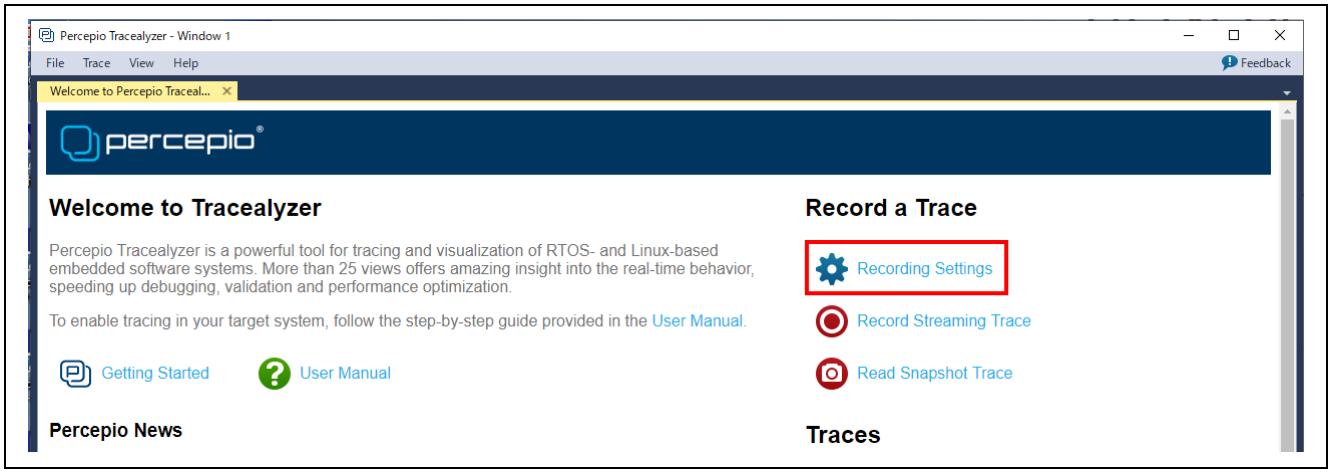

Figure 3-29 Recording Settings

- Device: (User PC system port)
- Data bits: **8**
- Data rate: **921600**
- Handshake: **None**
- Parity: **None**
- Stop bits: **One**

Specify the COM port number that corresponds to the USB-serial converter chip on the Pmod module connected to the CK-RX65N board.

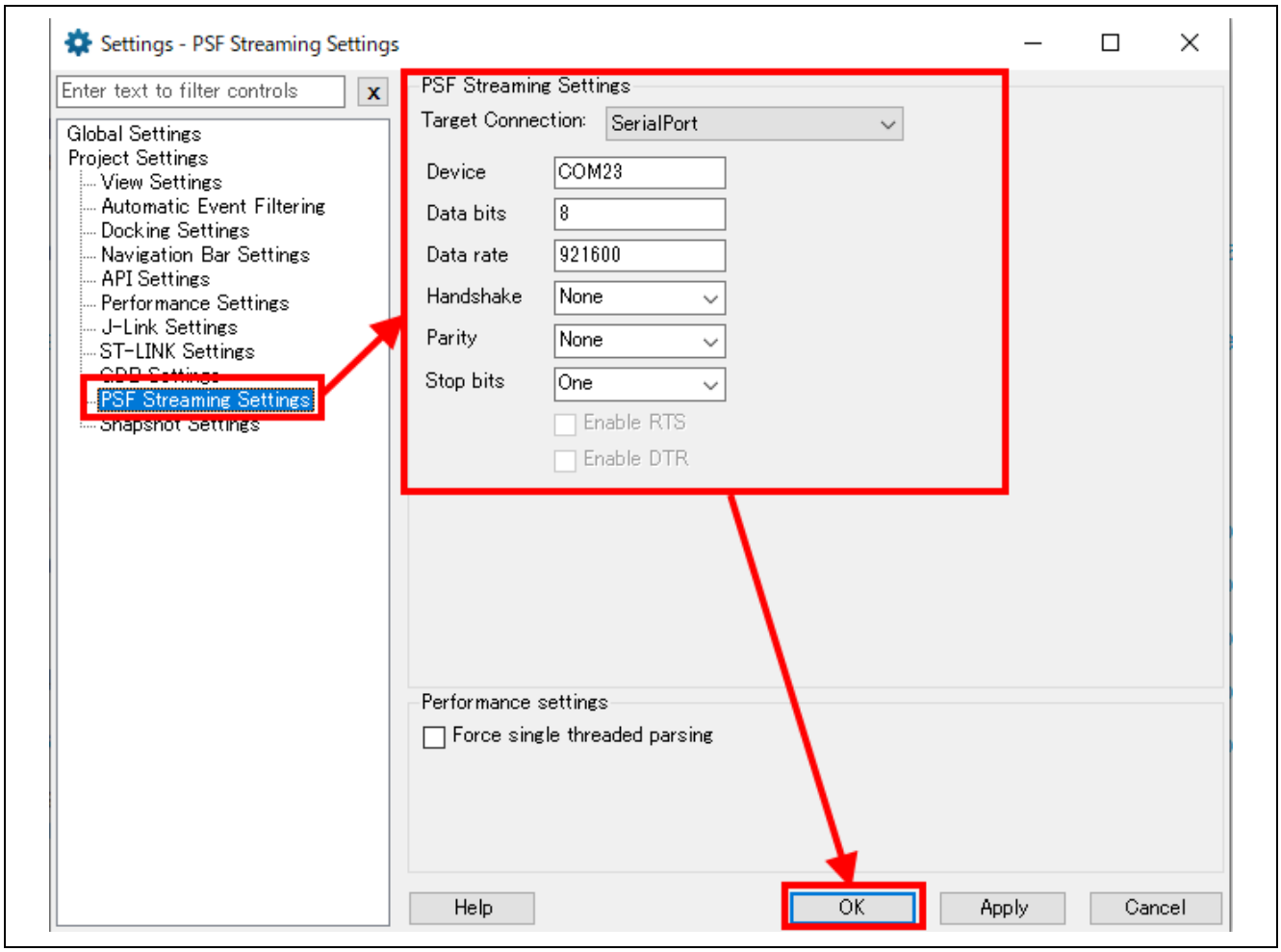

Figure 3-30 Settings for [PSF Streaming Settings]

Next, select [Record Streaming Trace].

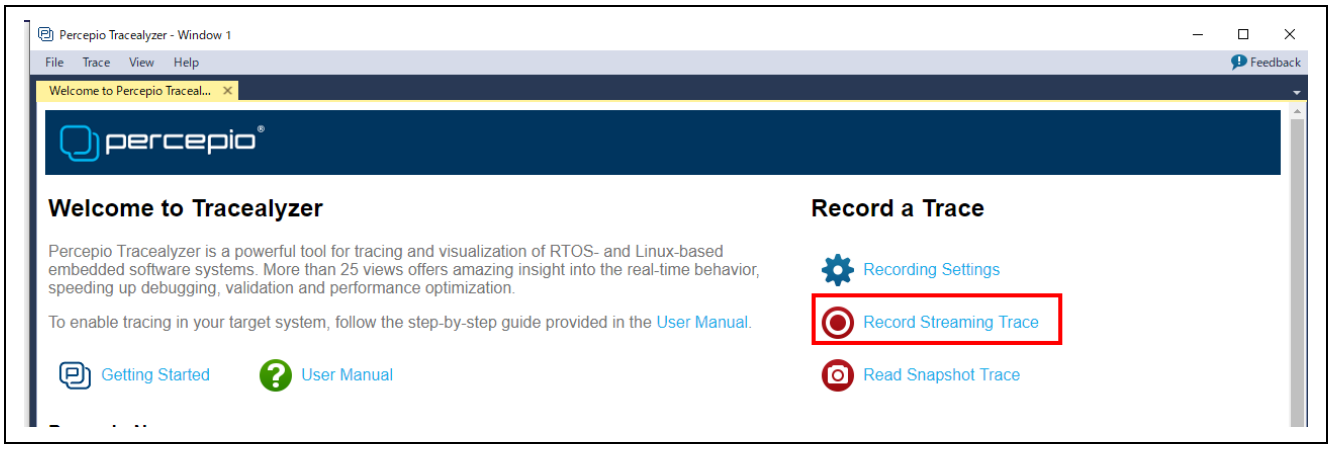

Figure 3-31 Record Streaming Trace

Select [Reconnect] and then [Start Session] to place Tracealyzer® in a state of waiting.

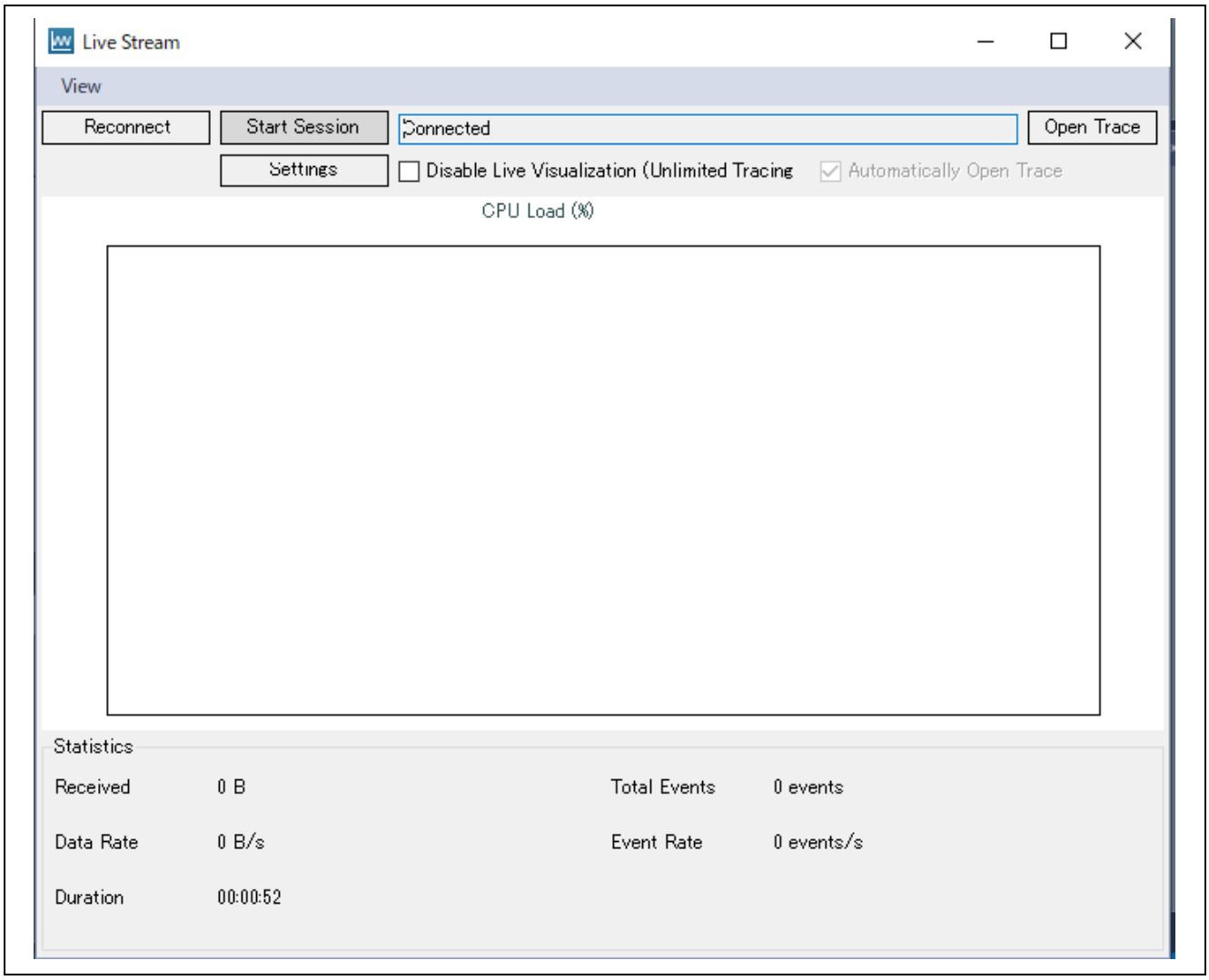

Figure 3-32 Placing Tracealyzer® in a State of Waiting

3.7.3 Executing Software

Select [Resume] from the [Run] menu of the e^2 studio to run the software. Communications between the CK-RX65N board and host PC (running Tracealyzer®) will begin and the Tracealyzer® window will display the internal state of FreeRTOS.

3.7.4 Display of Trace Information

Various modes of analysis are provided. For more information, see the [Help] tabbed page.

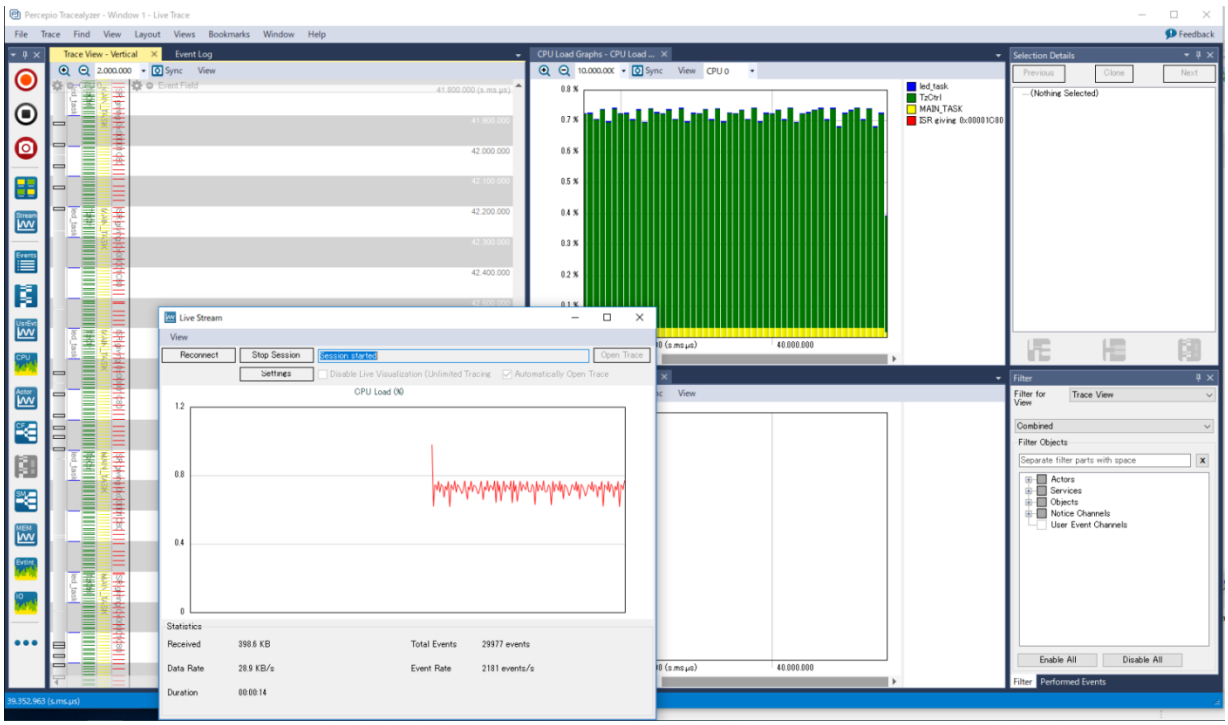

Figure 3-33 Display of Trace Information

Website and Support

Visit the following URLs to learn about key elements of the RX family, download components, and related documentation, and get support.

RX Family Product Information	www.renesas.com/rx
RX Family Product Support Forum	www.renesas.com/rx/forum
Renesas Support	www.renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar.25.23	-	First release document
1.01	May.19.23	5,6,7	Modifying code in trcStreamPort.c

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.

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

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

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/.