CCE4511 Sample Code

This Application Note describes how to use the CCE4511_sample_app sample code to evaluate the CCE4511 on the CCE4511-EVAL-V1.

Contents

Figures

Tables

1. Required Tools and Hardware

To use the CCE4511_sample_app sample code, the following tools and hardware are required:

1. CCE4511-EVAL-V1 evaluation board
2. RZ/N2L-RSK Starter Kit+ for RZ/N2L
3. Flexible Software Package (FSP) for RZ/N2L release V.2.2.0 or higher (GitHub - renesas/rzn-fsp: Flexible Software Package (FSP) for Renesas RZ/N series)
4. Optional: Debug probe (for example. J-LINK)
5. 24 V Power Supply
6. Micro USB Cable (included in RZ/N2L-RSK Starter Kit+ for RZ/N2L)
7. Mini USB Type B Cable (included in RZ/N2L-RSK Starter Kit+ for RZ/N2L)
8. USB C Cable (included in RZ/N2L-RSK Starter Kit+ for RZ/N2L)

2. Import and Build the CCE4511_sample_app in E² Studio

Download the Flexible Software Package (FSP) for RZ/N2L release V.2.2.0 or higher and install it.

Copy the CCE4511_sample_app folder into your workspace.

To import the CCE4511_sample_app sample code, it is recommended to use the import wizard of E² Studio. The import wizard is available in the File menu:

File > Import… > General > Existing Projects into Workspace

After starting the import wizard:

1. Click “Next” and search for the CCE4511_sample_app folder in your workspace.
2. Select the CCE4511_sample_app and click “Finish”

The CCE4511_sample_app project is now visible in the Project Explorer.

To finish the import process, double-click on “configuration.xml”.

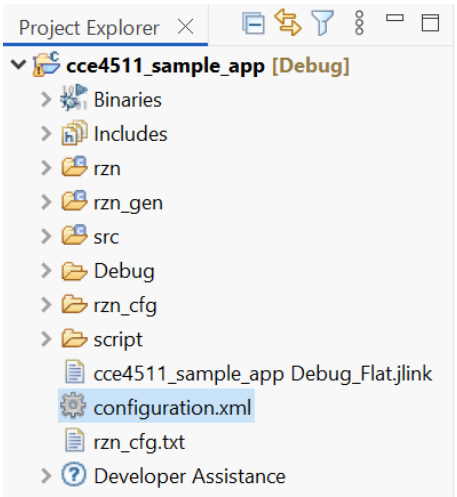

Figure 1. Project Explorer View

This opens the smart configurator view, where you can check if all necessary software components are available and updated.

Figure 2 shows the “Summary” tab when all software components are correctly installed.

Figure 2. Components Tab

Project Summary

Board:	RSK+RZN2L (RAM execution without flash memory)
Device:	R9A07G084M04GBG
Core:	CR52_0
Toolchain:	GCC for Renesas RZ
Toolchain Version:	13.3.1.arm-13-24
FSP Version:	2.2.0
Project Type:	Flat
Location:	C:/Software_Workspace/cce4511_sam...1_20250602/prj/cce4511_sample_app

Selected software components

Board Support Package Common Files	v2.2.0
Memory Config Checking	v2.2.0
I/O Port	v2.2.0
Arm CMSIS Version 5 - Core	v5.7.0+renesas.3.fsp.2.2.0
RSK+RZN2L Board Support Files (RAM execution without flash memory)	v2.2.0
Board support package for R9A07G084M04GBG	v2.2.0
Board support package for RZN2L	v2.2.0
Board support package for RZN2L - FSP Data	v2.2.0
Serial Peripheral Interface	v2.2.0
Compare Match Timer	v2.2.0
USB Basic	v2.2.0
USB Peripheral Communications Device Class	v2.2.0
SCI UART	v2.2.0

The CCE4511_sample_app code is now ready to build. Make sure to build the code again after adding changes.

3. Hardware Setup

3.1 RZ/N2L-RSK Starter Kit+

Follow the instructions in chapter 2.5.1 of the “RZ/T2, RZ/N2 Getting Started with Flexible Software Package” Document (R01AN6434EJ0116 Rev.1.16). Make sure that SW4 is configured according to chapter 2.5.1.1. to run on RAM without external flash memory. Additionally, make sure that J9 is open to use J-Link OB (connector J10).

Follow chapter 4 of the above document to start with the Blinky project to verify functionality of the RZ/N2L-RSK Starter Kit+ setup if needed.

3.2 CCE4511-EVAL-V1

It is recommended to use the CCE4511-EVA-V1 in the PMOD +3V3 Power Supply mode. This enables the boards 3.3 V power domain to be supplied by the RZ/N2L-RSK. To do so, set Jumper JP9 to “PMOD”.

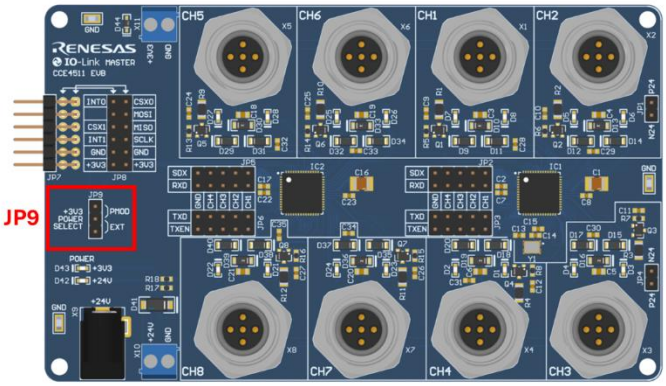

Figure 3. CCE4511-EVAL-V1 +3V3 Power Select

3.3 System Connections

Figure 4 shows how the RZ/N2L-RSK and the CCE4511-EVAL-V1 must be connected to a PC in order to use the CCE4511_sample_app.

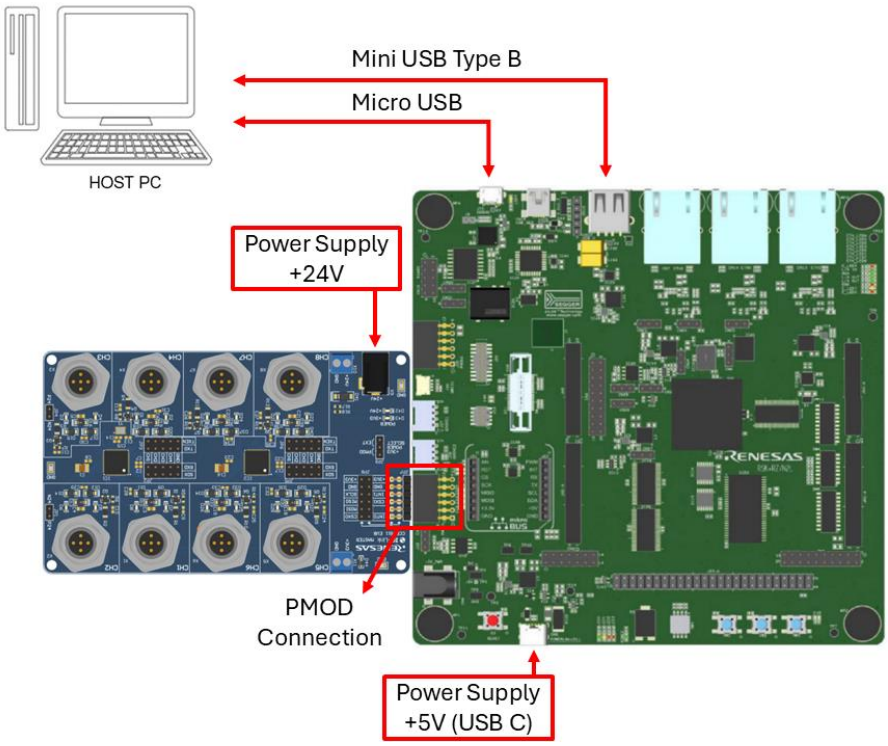

Figure 4. Hardware Connections

To use UART to control an IO-Link channel of the CCE4511, connect the selected channel to the RZ/N2L-RSK according to Table 1:

Table 1. UART Connection

UART Function (JP2, JP3, JP5, JP6)	RZ/N2L-RSK Pin
TXEN	J22.1
TXD	J22.4
RXD	J22.3

Only one UART channel is implemented for the RZ/N2L-RSK in the sample code. When using UART to control an IO-Link channel, please be aware that the SPI connection over the PMOD interface is needed for configuration.

4. Project Description

The CCE4511_sample_app is designed to easily evaluate the CCE4511 IO-Link Master Transceiver without the need to set up a new project. The user only needs to adapt the `sample_app.c` file, located in "*cce4511_sample_app* > *src* > *sample_app* > *src*" of the project, to control both CCE4511.

4.1 Functions Overview

Table 2. Functions Overview

Function	Description
`cce4511_spi_init`	Initializes the SPI interface to communicate with the CCE4511
`cce4511_register_read`	Reads the selected register of the CCE4511
`cce4511_register_write`	Writes to the selected register of the CCE4511
`cce4511_chip_reset`	Triggers and waits for chip reset
`cce4511_configure_sio_all_ch`	Configures the CCE4511 to be used in SIO mode (all channels)
`cce4511_configure_sio_pin_mode_all_ch`	Configures the CCE4511 to be used in SIO mode in pin-mode (all channels)
`cce4511_dump_register`	Reads the selected register and prints its value to the console
`cce4511_dump_cc_registers`	Reads all chip control registers and prints their values to the console
`cce4511_dump_ch_registers`	Reads the selected channels registers and prints their values to the console
`cce4511_dump_registers`	Reads all registers and prints their values to the console
`cce4511_pin_set`	Sets the selected pin high or low
`cce4511_wakeup_start`	Starts the wakeup procedure
`cce4511_wakeup_stop`	Stops the wakeup procedure
`cce4511_uart_init`	Initializes the UART interface to communicate with the CCE4511
`cce4511_uart_send`	Sends data over the UART interface
`printf`	Prints outputs to the console
`R_BSP_SoftwareDelay`	Delays the program

4.2 Description of Functions

4.2.1. cce4511_spi_init

Initializes the SPI interface to communicate with the CCE4511. This needs to be called once before starting communication.

Format

```
fsp_err_t cce4511_spi_init()
```

Parameters

None

4.2.2. cce4511_register_read

Reads the selected register of the CCE4511. All register definitions can be found in `cce4511.h`.

Format

```
fsp_err_t cce4511_register_read(E_CCE4511_REG_ADDR_t address,
                                uint8_t *reg_value,
                                uint16_t *p_status)
```

Parameters

address	Address of the register
reg_value	Value of the register
p_status	MISO status nibble

4.2.3. cce4511_register_write

Writes to the selected register of the CCE4511. All register definitions can be found in `cce4511.h`.

Format

```
fsp_err_t cce4511_register_write(E_CCE4511_REG_ADDR_t address,
                                 uint8 t *reg value,
                                 uint16_t *p_status)
```

Parameters

address	Address of the register
reg_value	Value of the register
p_status	MISO status nibble

4.2.4. cce4511_chip_reset

Triggers and waits for chip reset.

Format

```
fsp_err_t cce4511_chip_reset()
```

Parameters

None

4.2.5. cce4511_configure_sio_all_ch

Configures the CCE4511 for use in SIO mode. Applies to all channels.

Format

```
fsp_err_t cce4510_configure_sio_all_ch();
```

Parameters

None

4.2.6. cce4511_configure_sio_pin_mode_all_ch

Configures the CCE4511 for use in SIO mode and? in pin-mode (channel is controlled by TXD and TXEN pins). Applies to all channels.

Format

```
fsp_err_t cce4511_configure_sio_pin_mode_all_ch();
```

Parameters

None

4.2.7. cce4511_dump_register

Reads the selected register and prints its value to the console. The “name” parameter defines the printed name in the console.

Format

```
cce4511 dump register(const char *name,
                      E_CCE4511_REG_ADDR_t address);
```

Parameters

name	Name of the register
address	Address of the register

4.2.8. cce4511_dump_cc_registers

Reads all chip control registers and prints their values to the console

Format

```
cce4511_dump_cc_registers();
```

Parameters

None

4.2.9. cce4511_dump_ch_registers

Reads the selected channels registers and prints their values to the console

Format

```
cce4511_dump_ch_registers(E_CCE4511_CHANNEL_t channel);
```

Parameters

Channel	Selects the channel (Channel 1 – Channel 4)

4.2.10. cce4511_dump_registers

Reads all chip control registers and prints their values to the console

Format

```
cce4511_dump_cc_registers();
```

Parameters

None

4.2.11. cce4511_pin_set

Sets the selected pin high or low. Selectable pins are CQ1 – CQ4 and LP1 – LP4.

Format

```
cce4511_pin_set(E_CCE4511_CHANNEL_t channel,
                E_CCE4511_PIN_t pin,
                E_CCE4511_PIN_LEVEL_t level);
```

Parameters

channel	Selects the channel (Channel 1 – Channel 4)
pin	Selects the pin (CQ1 – CQ4, LP1 – LP4)
level	Expected level (high or low)

4.2.12. cce4511_wakeup_start

Starts the wakeup procedure for the selected channel.

Format

```
cce4511_wakeup_start(E_CCE4511_CHANNEL_t channel);
```

Parameters

channel	Selects the channel (Channel 1 – Channel 4)

4.2.13. cce4511_wakeup_stop

Stops the wakeup procedure for the selected channel.

Format

```
cce4511_wakeup_stop(E_CCE4511_CHANNEL_t channel);
```

Parameters

channel	Selects the channel (Channel 1 – Channel 4)

4.2.14. cce4511_uart_init

Initializes the UART interface to communicate with the CCE4511. This needs to be called once before starting communication.

Format

```
fsp_err_t cce4511_uart_init(void);
```

Parameters

None

4.2.15. cce4511_uart_send

Sends data over the UART interface

Format

```
fsp_err_t cce4511_uart_send(uint8_t *p_src,
                            uint16_t const length);
```

Parameters

*p_src	Buffer to send
length	Length of the buffer

4.2.16. printf

Prints outputs to the console. Standard stdio printf command.

Format

```
printf (const char *__restrict, ...)
```

Parameters

*__restrict	see standard stdio printf command description

4.2.17. R_BSP_SoftwareDelay

Static delay before executing the next command.

Format

```
R_BSP_SoftwareDelay (uint32_t delay,
                     bsp_delay_units_t units)
```

Parameters

delay	The number of 'units' to delay.
units	The 'base' for the units specified. Valid units are BSP_DELAY_UNITS_SECONDS, BSP_DELAY_UNITS_MILLISECONDS, BSP_DELAY_UNITS_MICROSECONDS

4.3 sample_app.c Overview

The `sample_app.c` file contains two use-case examples (internal macros), one for SPI controlled communication and one for UART controlled communication. The user can use and modify these examples as needed or completely develop their own example code in the `sample_app_main` function.

To choose between the two examples, the user needs to modify the `CCE4511_SAMPLE_APP_PHY_COMM_MODE` #define in the `sample_app_config.h` file (see Figure 5)

```
26          *** Global macros
29          * @brief CCE4510 PHY mode selection.
33          #define CCE4511_SAMPLE_APP_PHY_COMM_MODE          (CCE4511_SAMPLE_APP_PHY_COMM_MODE_UART)
34
35          #if CCE4511_SAMPLE_APP_PHY_COMM_MODE == CCE4511_SAMPLE_APP_PHY_COMM_MODE_UART
36
38          * @brief COM speed for transparent PHY mode.
42          #define CCE4511_SAMPLE_APP_UART_BAUD_RATE         (CCE4511_SAMPLE_APP_IOL_BR_COM1)
43
44          #endif
```

Figure 5. Comm Mode Selection

Changing line 33 from `CCE4511_SAMPLE_APP_PHY_COMM_MODE_UART` to `CCE4511_SAMPLE_APP_PHY_COMM_MODE_SPI` will switch the used example from UART to SPI.

Rebuild the project after changes are implemented.

The examples are called in the `sample_app_main` function, depending on what mode is chosen (see Figure 6)

```
188         void sample_app_main(void)
189         {
190
191         #if CCE4511_SAMPLE_APP_PHY_COMM_MODE == CCE4511_SAMPLE_APP_PHY_COMM_MODE_SPI
192
193             spi_main();
194
195         #elif CCE4511_SAMPLE_APP_PHY_COMM_MODE == CCE4511_SAMPLE_APP_PHY_COMM_MODE_UART
196
197             uart_main();
198
199         #else
200
201         #error "Unknown PHY communication mode selected!"
202
203         #endif
204         }
```

Figure 6. Sample_App_Main Function

4.3.1. SPI Slave Select Configuration

To switch between the SPI configuration of the two CCE4511 transceivers (IC1 and IC2), the SPI initialization configuration needs to be adapted.

The SPI initialization can be found in the `hal_data.c` file.

```
191             /** SPI extended configuration for SPI HAL driver */
192             const spi_extended_cfg_t g_spi2_ext_cfg = { .spi_clksyn = SPI_SSL_MODE_SPI,
193                     .spi_comm = SPI_COMMUNICATION_FULL_DUPLEX, .ssl_polarity = SPI_SSLP_LOW,
194                     .ssl_select = SPI_SSL_SELECT_SSL0, .mosi_idle =
195                             SPI_MOSI_IDLE_VALUE_FIXING_DISABLE, .parity =
196                             SPI_PARITY_MODE_DISABLE, .byte_swap = SPI_BYTE_SWAP_DISABLE,
```

Figure 7. SPI Initialization

Line 194 determines which SPI slave select line is used when communicating.

Table 3 shows the parameter values to switch between the different CCE4511 ICs.

Table 3. SPI SSL Assertion

Parameter for SSL Assertion	Selected CCE4511 IC
`SPI_SSL_SELECT_SSL0`	IC1
`SPI_SSL_SELECT_SSL1`	IC2

Some errors can occur after opening the hal_data.c file. These can be resolved with

Project > C/C++ Index > Rebuild

4.3.2. UART Example Code

The UART example code sends example data via the UART interface, while using the SPI interface for configuration of the CCE4511. After SPI and UART initialization, the CCE4511 is configured to SIO mode and the current register values are printed to the console.

The example data is then sent once every second.

Note: Additional cable connections are required when using the UART example code! See 3.3 System Connections

Figure 8 shows the UART example code.

```
161  static void uart_main(void)
162  {
163      uint8_t example_data[] = {0xCA, 0xFE, 0xCA, 0xFE, 0xCA, 0xFE};
164
165      /* SPI interface is needed for configuration. */
166      cce4511_spi_init();
167
168      cce4511_uart_init();
169
170      printf("Configuring PHY for SIO mode...\r\n");
171
172      cce4511_configure_sio_pin_mode_all_ch();
173
174      printf("Done\r\n");
175
176      cce4511_dump_registers();
177
178      while(1U)
179      {
180          printf("Sending example data...\r\n");
181          cce4511_uart_send(example_data, sizeof(example_data));
182          R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
183      }
184
185  }
186  #endif
```

Figure 8. UART Example Code

4.3.3. SPI Example Code

The SPI example code uses the SPI interface to toggle CQ and LP, and to trigger the automatic wake-up sequence of the CCE4511.

After SPI initialization, the CCE4511 is configured to SIO mode and the current register values will be printed to the console. CQ1 and CQ2 are toggled for ten times, LP1 and LP2 are toggled for ten times, followed by triggering the automatic wake-up sequence once every 5 seconds (see Figure 9, Figure 10 and Figure 11).

```
60          void spi_main(void)
61          {
62              uint16_t status;
63              uint8_t register_value;
64              uint8_t i;
65
66              cce4511_spi_init();
67
68              printf("Retrieving revision code from the PHY...\r\n");
69
70              cce4511_register_read(E_CCE4511_REG_ADDR_CC_REV, &register_value, &status);
71
72              printf("Revision: 0x%02X\r\n", register_value);
73
74              R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
75
76              printf("Configuring PHY for SIO mode...\r\n");
77
78              cce4511_configure_sio_all_ch();
79
80              printf("Done.\r\n");
81
82              R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
83
84              printf("Reading PHY registers current values...\r\n");
85
86              cce4511_dump_registers();
87
88              R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
```

Figure 9. SPI Example Code Initialization

```
90              printf("Toggling CQ pins...\r\n");
91
92              for(i = 0U; i < 10U; i++)
93              {
94                  printf("CQ --> LOW\r\n");
95                  cce4511_pin_set(E_CCE4511_CHANNEL_1, E_CCE4511_PIN_CQ, E_CCE4511_PIN_LEVEL_LOW);
96                  cce4511_pin_set(E_CCE4511_CHANNEL_2, E_CCE4511_PIN_CQ, E_CCE4511_PIN_LEVEL_LOW);
97                  cce4511_pin_set(E_CCE4511_CHANNEL_3, E_CCE4511_PIN_CQ, E_CCE4511_PIN_LEVEL_LOW);
98                  cce4511_pin_set(E_CCE4511_CHANNEL_4, E_CCE4511_PIN_CQ, E_CCE4511_PIN_LEVEL_LOW);
99
100                 R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
101
102                 printf("CQ --> HIGH\r\n");
103                 cce4511_pin_set(E_CCE4511_CHANNEL_1, E_CCE4511_PIN_CQ, E_CCE4511_PIN_LEVEL_HIGH);
104                 cce4511_pin_set(E_CCE4511_CHANNEL_2, E_CCE4511_PIN_CQ, E_CCE4511_PIN_LEVEL_HIGH);
105                 cce4511_pin_set(E_CCE4511_CHANNEL_3, E_CCE4511_PIN_CQ, E_CCE4511_PIN_LEVEL_HIGH);
106                 cce4511_pin_set(E_CCE4511_CHANNEL_4, E_CCE4511_PIN_CQ, E_CCE4511_PIN_LEVEL_HIGH);
107
108                 R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
109             }
110
111             printf("Toggling L+ pins...\r\n");
112
113             for(i = 0U; i < 10U; i++)
114             {
115                 printf("L+ --> LOW\r\n");
116                 cce4511_pin_set(E_CCE4511_CHANNEL_1, E_CCE4511_PIN_LP, E_CCE4511_PIN_LEVEL_LOW);
117                 cce4511_pin_set(E_CCE4511_CHANNEL_2, E_CCE4511_PIN_LP, E_CCE4511_PIN_LEVEL_LOW);
118                 cce4511_pin_set(E_CCE4511_CHANNEL_3, E_CCE4511_PIN_LP, E_CCE4511_PIN_LEVEL_LOW);
119                 cce4511_pin_set(E_CCE4511_CHANNEL_4, E_CCE4511_PIN_LP, E_CCE4511_PIN_LEVEL_LOW);
120
121                 R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
122
123                 printf("L+ --> HIGH\r\n");
124                 cce4511_pin_set(E_CCE4511_CHANNEL_1, E_CCE4511_PIN_LP, E_CCE4511_PIN_LEVEL_HIGH);
125                 cce4511_pin_set(E_CCE4511_CHANNEL_2, E_CCE4511_PIN_LP, E_CCE4511_PIN_LEVEL_HIGH);
126                 cce4511_pin_set(E_CCE4511_CHANNEL_3, E_CCE4511_PIN_LP, E_CCE4511_PIN_LEVEL_HIGH);
127                 cce4511_pin_set(E_CCE4511_CHANNEL_4, E_CCE4511_PIN_LP, E_CCE4511_PIN_LEVEL_HIGH);
128
129                 R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
130             }
```

Figure 10. SPI CQ and LP Toggle

```
132             printf("Reading PHY registers current values...\r\n");
133
134             cce4511_dump_registers();
135
136             while(1)
137             {
138                 printf("Triggering automatic wake-up sequence...\r\n");
139
140                 cce4511_wakeup_start(E_CCE4511_CHANNEL_1);
141                 cce4511_wakeup_start(E_CCE4511_CHANNEL_2);
142                 cce4511_wakeup_start(E_CCE4511_CHANNEL_3);
143                 cce4511_wakeup_start(E_CCE4511_CHANNEL_4);
144
145                 R_BSP_SoftwareDelay(5000, BSP_DELAY_UNITS_MILLISECONDS);
146
147                 cce4511_wakeup_stop(E_CCE4511_CHANNEL_1);
148                 cce4511_wakeup_stop(E_CCE4511_CHANNEL_2);
149                 cce4511_wakeup_stop(E_CCE4511_CHANNEL_3);
150                 cce4511_wakeup_stop(E_CCE4511_CHANNEL_4);
151
152                 cce4511_configure_sio_all_ch();
153
154                 R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
155             }
156         }
```

Figure 11. SPI Automatic Wake-Up

4.4 Debugging

To start debugging the CCE4511 sample code, perform the following steps:

1. Connect the Boards according to 3. Hardware Setup.
2. Optional: Connect UART interface to the selected channel.
3. Power on +24 V and +5 V power supply.
4. Click on “Run” > “Debug Configurations…” and double click on “Renesas GDB Hardware Debugging” to create a debug configuration.
5. In the “Debugger” tab of the created debug configuration, select “J-Link ARM” as Debug Hardware and “R9A07G084M04” as Target Device

   Note: This uses the on-board debugger of the RZ/N2L-RSK
6. In the “Debugger” tab, switch to the “Connections Settings” tab and set “Set CPSR(5bit) after download” to “yes”.
7. In the “Debugger” tab, switch to the “Debug Tool Settings” tab and set “Run Break Time Measurement” to “No”
8. Click “Apply”.
9. Build the project.
10. Start debugging. The sample code will stop at two break points. Click on “Resume” (F8) to continue until the host PC recognizes the USB Serial Device as COM port (e.g. COM5). The sample code pauses until a terminal is opened.
11. Open a terminal console in E² Studio (Ctrl+Alt+Shift+T) and choose “Serial Terminal” with a baud rate of 115200.

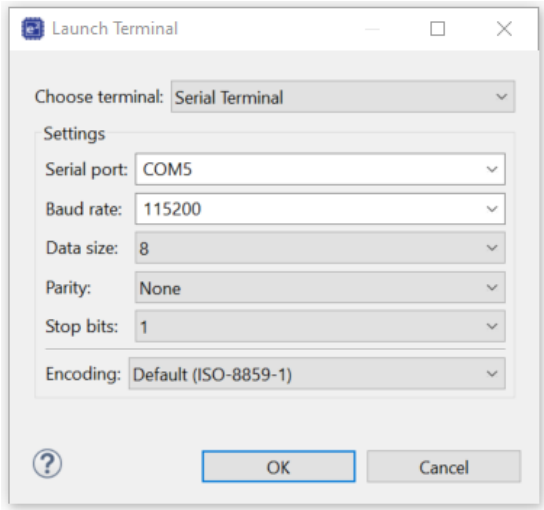

Figure 12. Project Explorer View

It is also possible to use any other suitable terminal emulator.

12. The sample code resumes automatically as soon as the terminal is started, and the chosen example is executed.

6. Revision History

Revision	Date	Description
0.01	July 25, 2025	Initial release.