

RX Family

CAN FD FIT to FSP Migration Guide

Introduction

This application note provides a comprehensive guide on migrating an existing application that uses Controller Area Network with Flexible Data Rate (CANFD) Firmware Integration Technology (FIT) module to the CAN FD FSP module (Renesas Flexible Software Package). It includes detailed instructions, migration steps, and demonstrations to assist users throughout the migration process.

Target Modules

- FIT Module: r_canfd_rx
- FSP Module: r_canfd

Contents

1. Overview	3
1.1 Stack Diagram	3
1.2 API Functions	4
2. Migrating Project FIT to FSP.....	5
2.1 Creating a new FSP Project	5
2.2 Migrating Configuration	6
2.3 Generate Code	11
2.4 Migrating Application Code	13
2.5 Build and Run the project	16
3. Reference Documents	18
4. Appendices.....	18
4.1 FIT & FSP Environment	18
4.2 Reference FSP Application Code.....	19
4.3 Comparison Reference	28
4.3.1 Compare Functions (Details).....	28
4.3.2 Compare Configuration (Details).....	31
Revision History	33

1. Overview

1.1 Stack Diagram

The figure below illustrates the stack diagram of the CAN FD FIT module and the CAN FD FSP module.

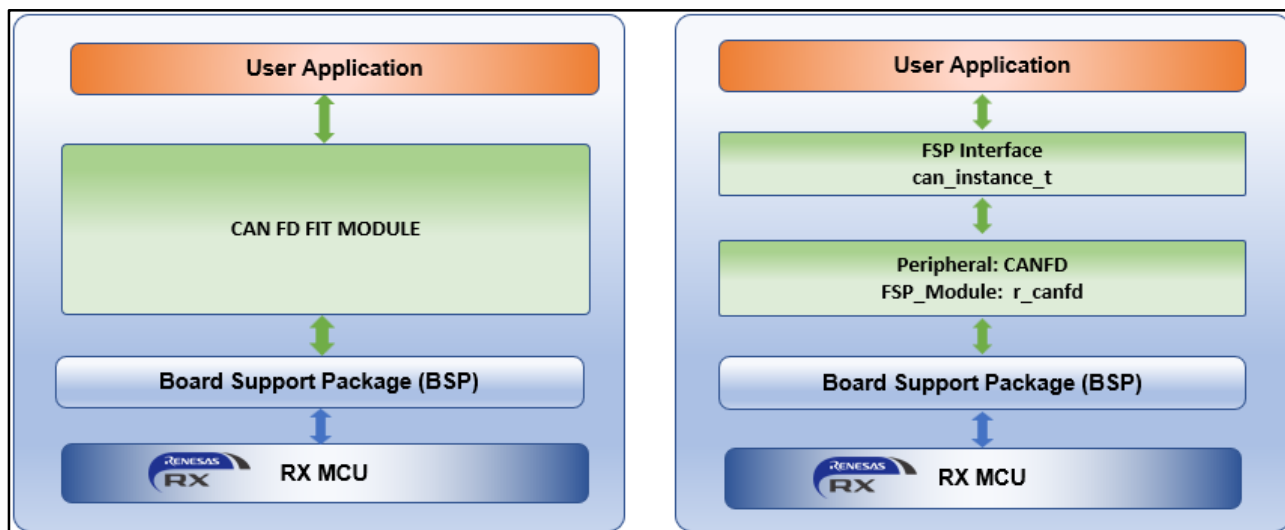


Figure 1-1 CAN FD FIT and CAN FD FSP stack diagram

In the CAN FD FIT, the user application directly accesses the CAN FD peripheral through the CAN FD FIT module, which is built on top of the Board Support Package (BSP).

In contrast, the CAN FD FSP stack introduces an additional FSP interface layer (`can_instance_t` interface) between the user application and the CAN FD peripheral. This interface provides a unified driver model that allows the application to interact with the CAN FD peripheral through a common API, improving portability and consistency across different peripherals and devices, while still depending on the BSP for hardware abstraction.

1.2 API Functions

This section summarizes the API Functions of the CAN FD FIT module and the CAN FD FSP module.

FIT	FSP
R_CANFD_Open	open
R_CANFD_Close	close
R_CANFD_Write	write
R_CANFD_Read	read
R_CANFD_GetVersion	Not supported in specific modules FSP has common function to get FSP pack version. R_FSP_VersionGet (in bsp_api.h)
R_CANFD_ModeTransition	modeTransition
Callback function set in R_CANFD_CallbackSet	callbackSet
R_CANFD_InfoGet	infoGet

For more details, please refer to the sections:

[Compare Functions \(Details\)](#) : comparision FIT and FSP.

[Migrate Application Code](#) : example of using FSP CAN FD API in an application.

2. Migrating Project FIT to FSP

2.1 Creating a new FSP Project

Refer to the reference document “RX FSP Migration Guide” for creating RX FSP project.

To create a new project in e² Studio, navigate to the top bar menu and select:

File > New > Renesas C/C++ Project > Renesas RX.

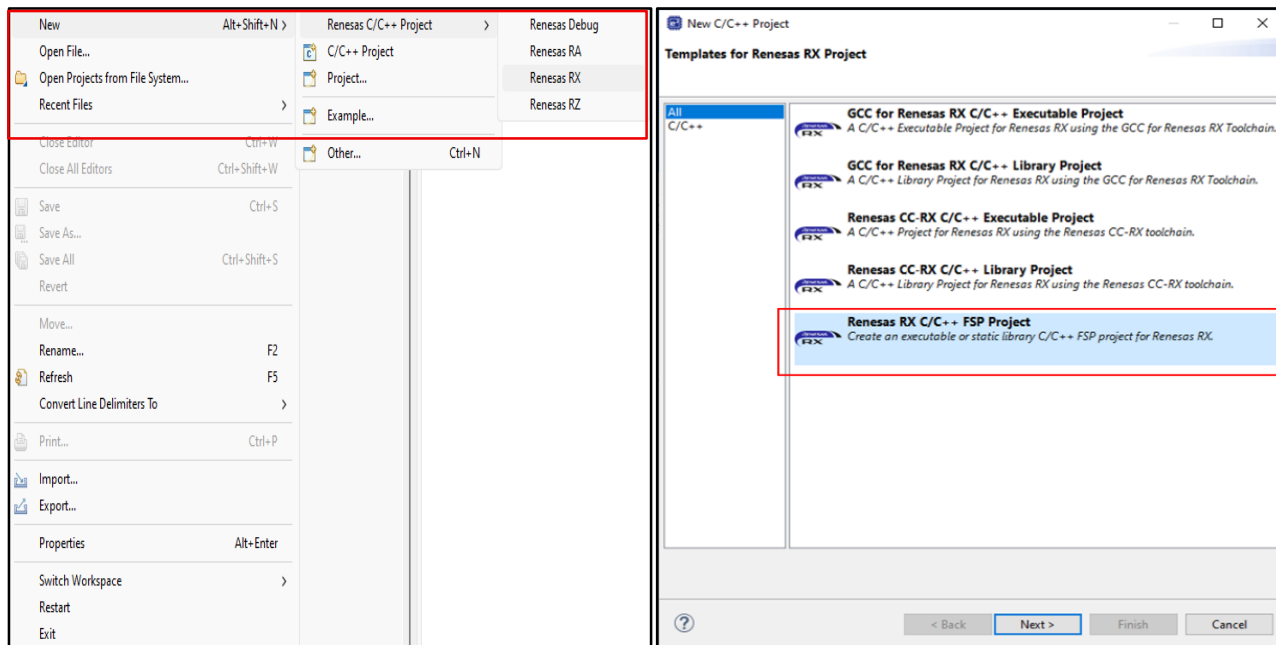


Figure 2-1 Create a new Renesas RX C/C++ FSP Project.

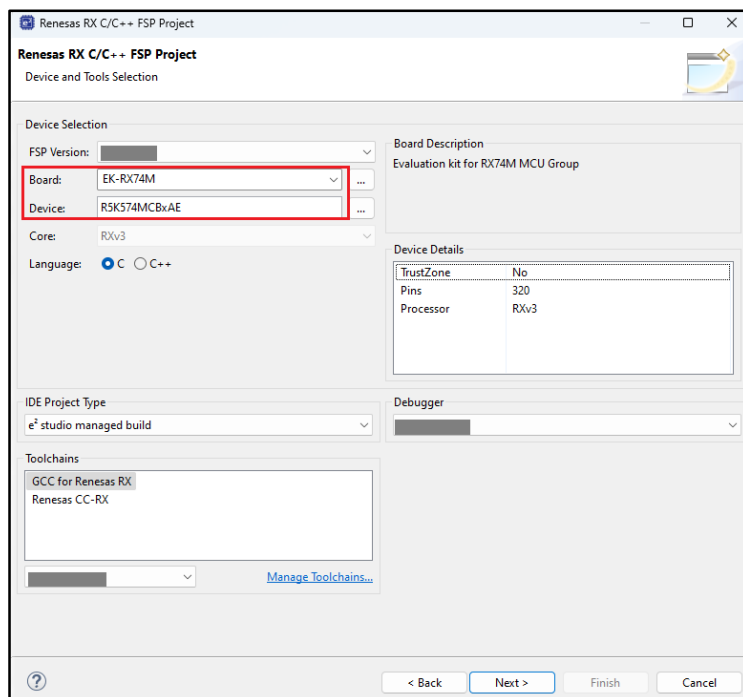


Figure 2-2 Select Board.

2.2 Migrating Configuration

This section describes migration configuration.

Configuration for CANFD in this application notes shown as table below:

Configuration	Value
Channel	0
Pin setting CRX0	P805
Pin Setting CTX0	P804
Bitrate	Nominal (bps): 500000 FD Data Rate (bps): 2000000 Sample Point (%): 75
Transmit Interrupts	TXMB 0
Filter List Array	p_canfd0_afl
Interrupts Priority	Priority 2
Callback	canfd0_callback (user's function name)

Note: Channel, Pins setting for CRX, CTX depend on User's board.

Steps of updating configuration on FSP Project.

On Configuration Stacks Tab

Click on New Stack -> Connectivity > CAN FD (r_canfd).

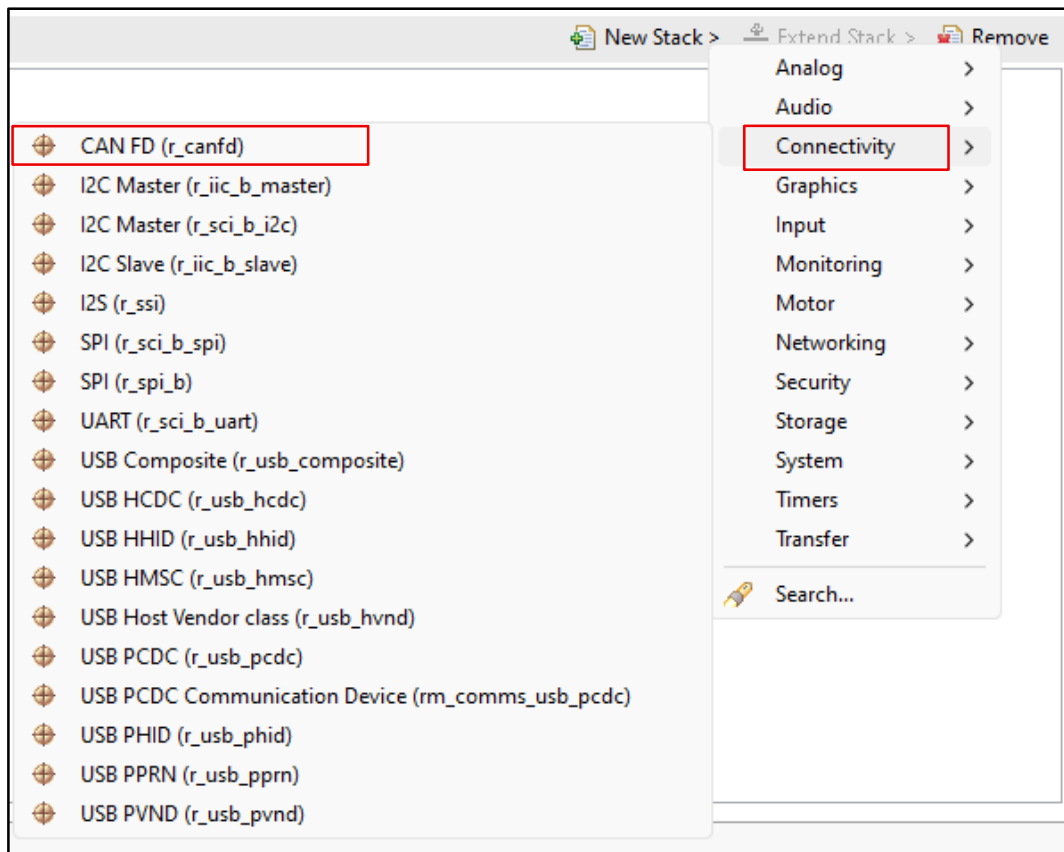


Figure 2-3 Add module CAN FD in Stack

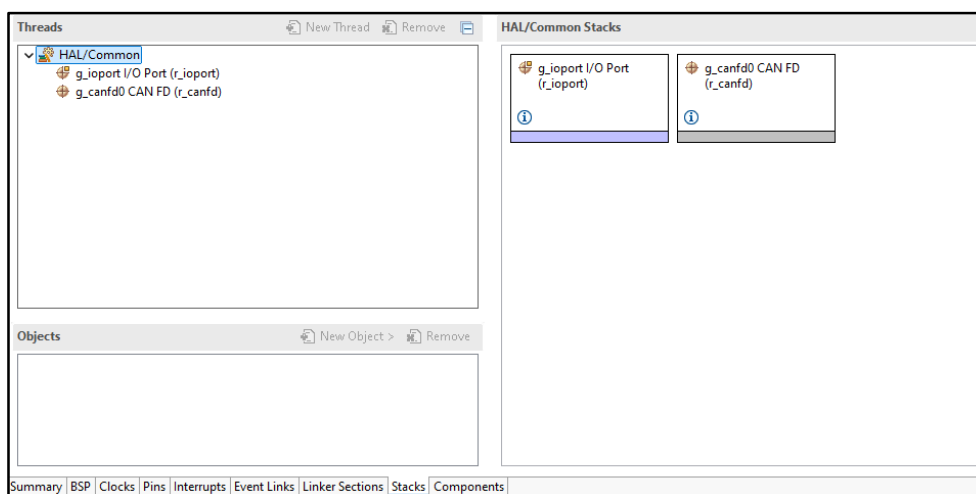


Figure 2-4 CAN FD in Stack

Click on the "g_canfd0 CAN FD (r_canfd)" module once it appears.

The "Properties" window will display its properties.

Configure CAN FD FSP according to configuration above.

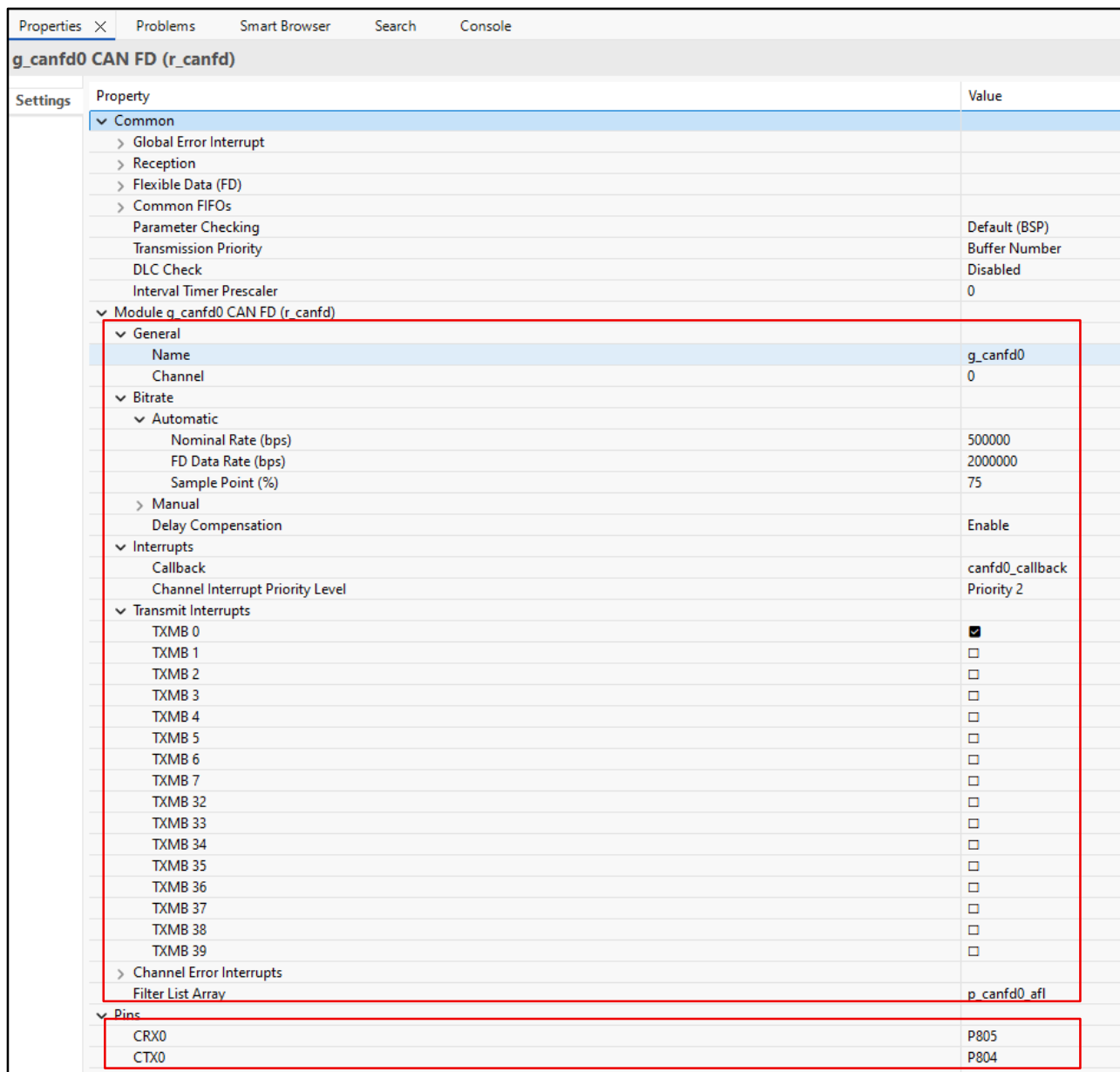


Figure 2-5 Configure CAN FD Properties

Requires CANFDCLK to be enabled on the Clocks tab.

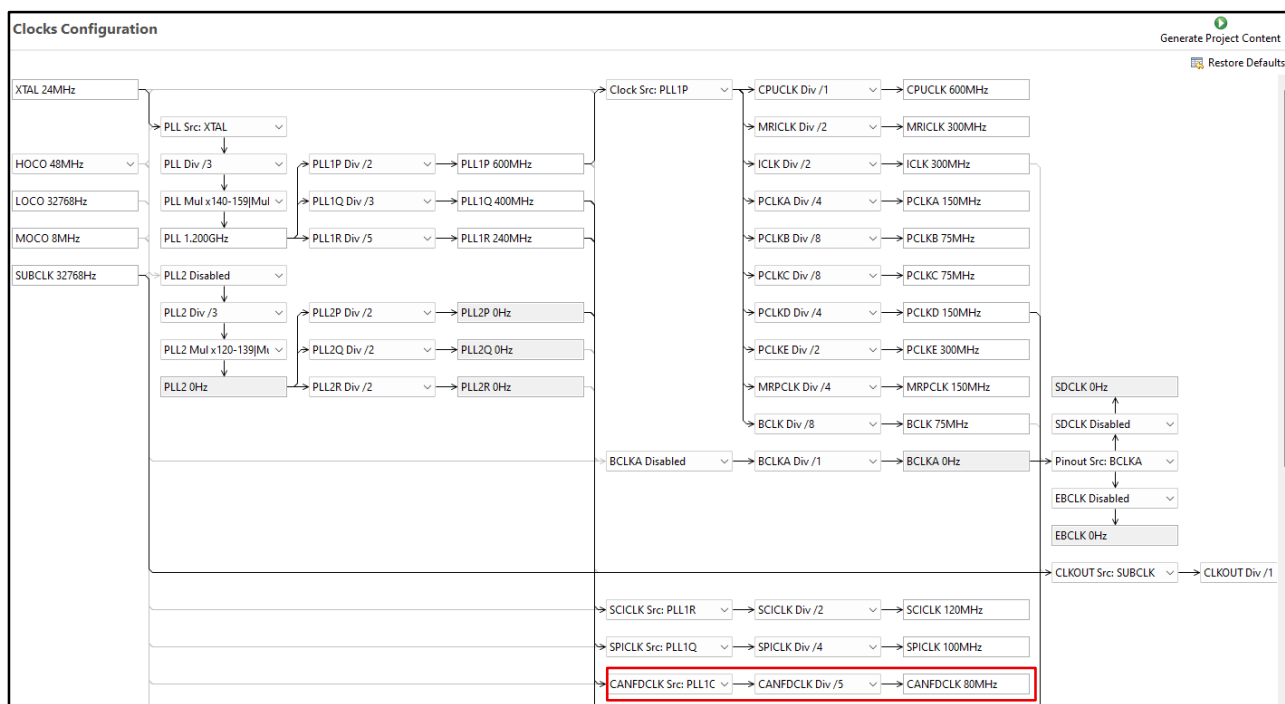


Figure 2-6 Clock Setting

Pin Configuration

Select Pin Configuration Export to CSV file Configure Pin Driver Warnings

EK-RX74M [Manage configurations...](#) Generate data: g_bsp_pin_cfg

Pin Selection

Type filter text

- > ✓ PE
- > ✓ PF
- > ✓ PG
- > ✓ Other Pins
- ✓ Peripherals
 - > ✓ Analog:ADC16H
 - > Analog:R12DA
 - ✓ Connectivity:CANFD
 - ✓ CANFD0
 - CANFD1
 - CANFD2
 - > Connectivity:DSMIF
 - > Connectivity:ESC
 - > Connectivity:ESWM
 - > ✓ Connectivity:ESWM_GMII/MII
 - > Connectivity:ESWM_GPTP
 - > Connectivity:ESWM_RGMII
 - > Connectivity:ESWM_RMII
 - > ✓ Connectivity:I3C
 - > Connectivity:IIC
 - > Connectivity:PCIF
 - > Connectivity:PDMIF
 - > ✓ Connectivity:SCI
 - > Connectivity:SDHI
 - > ✓ Connectivity:SPI

Pin Configuration

Name	Value	Lock	Link
Pin Group Selection	Mixed		
Operation Mode	Enabled		
✓ Input/Output			
CRX0	✓ P805		
CTX0	✓ P804		

Module name: CANFD0
Usage: CTX / CRX are recommended to use the same pin-group.

Pin Function | Pin Number

Summary | BSP | Clocks | Pins | Interrupts | Event Links | Linker Sections | Stacks | Components

The "Pin Configuration" tab shows the application's pin configuration.

2.3 Generate Code

After completing stacks and HAL configurations and making user selections, click Generate Project Content. This generates the configuration code for selected modules. You can find generated code using the Project Explorer window in e² studio. The configuration.xml file, containing all settings, can be edited in the GUI-based configuration editor for any necessary reconfigurations.

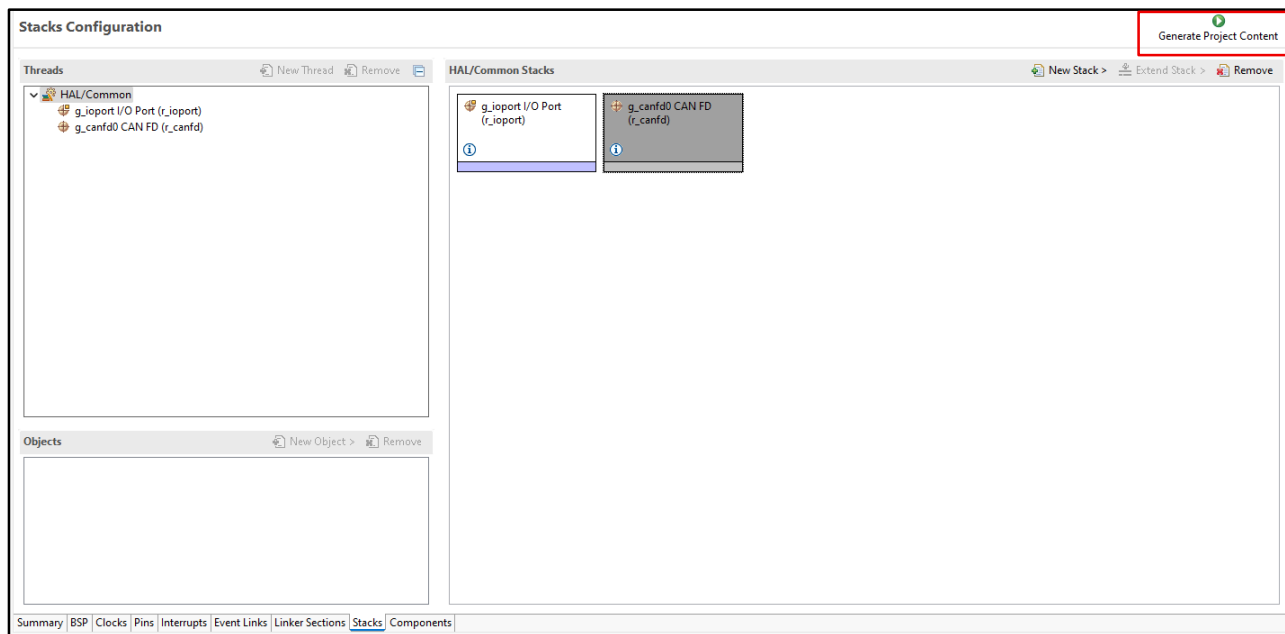


Figure 2-8 Generate Project Content

Once the module has been configured and the files generated, the CAN FD FSP module is ready to be used in an application.

Generated code in **rx_gen/hal_data.c**.

The following figure shows the generated code for CAN FD according to the configuration in the Properties.

```

1  /* generated HAL source file - do not edit */
2  #include "hal_data.h"
3  /* Nominal and Data bit timing configuration */
4
5  can_bit_timing_cfg_t g_canfd0_bit_timing_cfg =
6  {
7  /* Actual bitrate: 500000 Hz. Actual sample point: 75 %. */
8  .baud_rate_prescaler = 1,
9  .time_segment_1 = 119, .time_segment_2 = 40, .synchronization_jump_width = 4 };
10
11 #if BSP_FEATURE_CANFD_FD_SUPPORT
12 can_bit_timing_cfg_t g_canfd0_data_timing_cfg =
13 {
14 /* Actual bitrate: 2000000 Hz. Actual sample point: 75 %. */
15 .baud_rate_prescaler = 1,
16 .time_segment_1 = 29,
17 .time_segment_2 = 10,
18 .synchronization_jump_width = 4
19 };
20 #endif
21
22 extern const canfd_afl_entry_t p_canfd0_afl[CANFD_CFG_AFL_CH0_RULE_NUM];
23 #ifndef CANFD_PRV_GLOBAL_CFG
24 #define CANFD_PRV_GLOBAL_CFG
25
26 #ifdef FSP_NOT_DEFINED
27 #undef FSP_NOT_DEFINED
28 #endif
29 #define FSP_NOT_DEFINED (0)
30
31 /* Buffer RAM used: FSP_NOT_DEFINED bytes */
32 canfd_global_cfg_t g_canfd_global_cfg =
33 { .global_interruptions = CANFD_CFG_GLOBAL_ERR_SOURCES, .global_config = (CANFD_CFG_TX_PRIORITY | CANFD_CFG_DLC_CHECK
34 | (BSP_CFG_CANFDCLK_SOURCE == BSP_CLOCKS_SOURCE_CLOCK_MAIN_OSC ? R_CANFDG_GCFG_DLLCS_Msk : 0U)
35 | CANFD_CFG_FD_OVERFLOW | ((FSP_NOT_DEFINED) << R_CANFDG_GCFG_ITP_Pos)),
36 .rx_mb_config = (CANFD_CFG_RXMB_NUMBER | (CANFD_CFG_RXMB_SIZE << R_CANFDG_RMCR_PLS_Pos)), .global_err_ipl =
37 CANFD_CFG_GLOBAL_ERR_IPL,
38 .rx_fifo_ipl = CANFD_CFG_RX_FIFO_IPL, .rx_fifo_config =
39 { ((CANFD_CFG_RXFIFO0_INT_THRESHOLD << R_CANFDG_RFCR_RFITH_Pos) | (CANFD_CFG_RXFIFO0_DEPTH << R_CANFDG_RFCR_FDS_Pos)
40 | (CANFD_CFG_RXFIFO0_PAYLOAD << R_CANFDG_RFCR_PLS_Pos) | (CANFD_CFG_RXFIFO0_INT_MODE)
41 | (CANFD_CFG_RXFIFO0_ENABLE)),
42 ((CANFD_CFG_RXFIFO1_INT_THRESHOLD << R_CANFDG_RFCR_RFITH_Pos) | (CANFD_CFG_RXFIFO1_DEPTH << R_CANFDG_RFCR_FDS_Pos)
43 | (CANFD_CFG_RXFIFO1_PAYLOAD << R_CANFDG_RFCR_PLS_Pos) | (CANFD_CFG_RXFIFO1_INT_MODE)
44 | (CANFD_CFG_RXFIFO1_ENABLE)),
45 #if !BSP_FEATURE_CANFD_LITE

```

Figure 2-9 Generated file for CAN FD.

Generated code in file **pin_data.c**

The following figure shows the generated code according to the Pin Configuration.

```

234 { .pin = BSP_IO_PORT_08_PIN_02, .pin_cfg =
235 ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PORT_DIRECTION_INPUT) },
236 { .pin = BSP_IO_PORT_08_PIN_03, .pin_cfg =
237 ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PORT_DIRECTION_INPUT) },
238 { .pin = BSP_IO_PORT_08_PIN_04, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
239 | (uint32_t) IOPORT_CFG_PULLUP_ENABLE) },
240 { .pin = BSP_IO_PORT_08_PIN_05, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
241 | (uint32_t) IOPORT_CFG_PULLUP_ENABLE) },
242 { .pin = BSP_IO_PORT_08_PIN_06, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PORT_DIRECTION_INPUT
243 | (uint32_t) IOPORT_CFG_PULLUP_ENABLE) },

```

Figure 2-10 Generated files for Pin Configuration

2.4 Migrating Application Code

The typical steps for using the CAN FD FSP module in this application are as follows:

1. Configure the CAN FD module using FSP Configuration as described in section 2.2.
2. Initialize the CAN FD module by calling the open API.
3. Set the CAN FD operating mode by calling the modeTransition API.
4. Prepare the CAN FD transmit frame before transmission.
5. Transmit CAN FD data by calling the write API.
6. Receive CAN FD data using the callback and the read API.
7. Monitor communication status by calling the infoGet API.
8. Handle transmission, reception, and error events in the callback function.
9. Close the CAN FD module by calling the close API.

```
void hal_entry (void)
{
    /* Fill tx frame data that is to be sent in FD frame */
    for( uint16_t j = 0; j < SIZE_64; j++){}
    /* Fill rx frame data that is the same the transmitted FD frame */
    for(uint16_t j = 0; j < SIZE_64; j++){}

    /* Initializes pins for r_canfd_rx module */

    printf("\n- Starting demo for CAN FD module...\n");
    printf("\n- Call R_CANFD_Open() for Initialize the API...\n");

    fsp_err_t err;
    /* Initialize the API. */
    err = g_canfd0.p_api->open(&g_canfd0_ctrl, &g_canfd0_cfg);
    /* Error trap */
    if(FSP_SUCCESS != err)
    {
        printf("\nCANFD Open API failed\n");
        printf("Error: %d \n", err);
    }

    printf("\n- CANFD Open API success!\n");

    printf("\n- Go to the internal loop-back test mode!\n");
    /* Change to the loopback internal mode */
    g_canfd0.p_api->modeTransition(&g_canfd0_ctrl, (can_operation_mode_t)CAN_OPERATION_MODE_NORMAL, (can_test_mode_t)CAN_TEST_MODE_LOOPBACK_INTERNAL);
    /* Go to function canfd_operation in main.c */
    canfd_operation();

    /* Disable the loopback internal mode */
    g_canfd0.p_api->modeTransition(&g_canfd0_ctrl, (can_operation_mode_t)CAN_OPERATION_MODE_NORMAL, (can_test_mode_t)CAN_TEST_MODE_DISABLED);

    printf("\n- Go to Loop: \n");
    printf("\n + Check whether having the new message...\n");
    printf("\n + Or press sw1 to send a FD message in the internal loop-back test mode \n");
    printf("\n + Or press sw2 to send a FD message to the CAN FD analyzer/other board \n");
    printf("\n ***** \n");

    while(1)
    {
        /* Check whether having the new message... */
        can_read_operation();

        /* Read switches(sw1 or sw2 or sw3) */
        read_switches();
    }
}
```

Figure 2-11 Overall FSP CAN FD Application Code

Open

FIT

FSP

```

void main(void)
{
    g_canfd0_extended_cfg.p_af1 = p_canfd0_af1;
    g_canfd0_cfg.p_callback = canfd0_callback;

    /* Fill tx frame data that is to be sent in FD frame */
    for( uint16_t j = 0; j < SIZE_64; j++)
    {
        g_canfd_tx_frame.data[j] = (uint8_t) (j + 1);
    }
    /* Fill rx frame data that is the same the transmitted FD frame */
    for(uint16_t j = 0; j < SIZE_64; j++)
    {
        rx_fd_data[j] = (uint8_t) (j + 1);
    }

    /* Initializes pins for r_canfd_rx module */
    R_CANFD_PinSet_CANFD0();

    /* Configures the led & switch port and pin direction settings, and
    demo output ports configure();

#if BSP_CFG_IO_LIB_ENABLE
    printf("\n- Starting demo for CAN FD module...\n");
    printf("\n- Call R_CANFD_Open() for Initialize the API...\n");
#endif

    fsp_err_t err;
    /* Initialize the API. */
    err = R_CANFD_Open(&g_canfd0_ctrl1, &g_canfd0_cfg);
    /* Error trap */
    if(FSP_SUCCESS != err)
    {
        printf("\nCANFD Open API failed\n");
        printf("Error: %d \n", err);
    }
}

```

Figure 2-12 Open Code

FIT: Set configuration module in application code as figure above.

FSP: Set configuration module in FSP Configuration as shown in Figure 2-5 Configure for CAN FD FSP

Write

FIT

FSP

```

/* @brief This function is to transmit data frame[]
static void can_write_operation(can_frame_t can_transmit_frame)
{
    fsp_err_t err = FSP_SUCCESS;

    /* Transmit the data from mail box #0 with tx_frame */
    err = R_CANFD_Write(&g_canfd0_ctrl1, CAN_MAILBOX_NUMBER_0, &can_transmit_frame);
    /* Handle error */
    if (FSP_SUCCESS != err)
    {
        #if BSP_CFG_IO_LIB_ENABLE
            printf("\nCANFD Write API FAILED");
        #endif
        canfd_deinit();
    }

    uint32_t g_time_out = WAIT_TIME;
    /* Wait here for an event from callback */
    while ((true != b_canfd_tx_complete) && (g_time_out--));
    if (TIME_OUT_VALUE == g_time_out)
    {
        #if BSP_CFG_IO_LIB_ENABLE
            printf("\n Transmission failed due to timeout");
        #endif
        canfd_deinit();
        while(1);
    }
}

```

Figure 2-13 Write a message.

Read

FIT

FSP

```

/* Check if the data is received in Receive Message Buffers */
if(can_rx_info.rx_mb_status)
{
    /* Read the input frame received */
    err = R_CANFD_Read(&g_canfd0_ctrl, ZERO, &g_canfd_rx_frame);
    /* Handle error */
    if (FSP_SUCCESS != err)
    {
        printf("\nCAN Read API FAILED");
    }
}

```

Figure 2-14 Read a message

Get the status information

FIT

FSP

```

/* Get the status information for CAN transmission */
err = R_CANFD_InfoGet(&g_canfd0_ctrl, &can_rx_info);
/* Handle error */
if (FSP_SUCCESS != err)
{
    #if BSP_CFG_IO_LIB_ENABLE
        printf("\nCAN InfoGet API FAILED \n");
    #endif
    canfd_deinit();
}

```

Figure 2-15 InfoGet function code

Callback

FIT

FSP

```

/* @brief CAN FD callback function.
void canfd0_callback(can_callback_args_t *p_args)
{
    switch (p_args->event)
    {
        case CAN_EVENT_TX_COMPLETE:
        {
            b_canfd_tx_complete = true;    //set flag bit
            break;
        }
        case CAN_EVENT_RX_COMPLETE:
        {
            b_canfd_rx_complete = true;    //set flag bit
            for(uint16_t i = 0; i < 64; i++)
            {
                rx_fd_data_real[i] = p_args->frame.data[i];
            }
            break;
        }
        case CAN_EVENT_ERR_WARNING:
        case CAN_EVENT_ERR_PASSIVE:
        case CAN_EVENT_ERR_BUS_OFF:
        case CAN_EVENT_BUS_RECOVERY:
        case CAN_EVENT_MAILBOX_MESSAGE_LOST:
        case CAN_EVENT_ERR_BUS_LOCK:
        case CAN_EVENT_ERR_CHANNEL:
        case CAN_EVENT_TX_ABORTED:
        case CAN_EVENT_ERR_GLOBAL:
        {
            b_canfd_err_status = true;    //set flag bit
            break;
        }
    }
}

```

Figure 2-16 Callback Code

For more details about FSP sample code, please refer to this section [Reference FSP Application Code](#)

2.5 Build and Run the project

1. Build the sample application, download it to the board, and execute the application using a debugger.
2. Connect the analyzer to the board via the CAN interface. Connect the analyzer to the PC via USB. Open the CAN monitoring software on the PC to monitor and view CAN FD data frames.

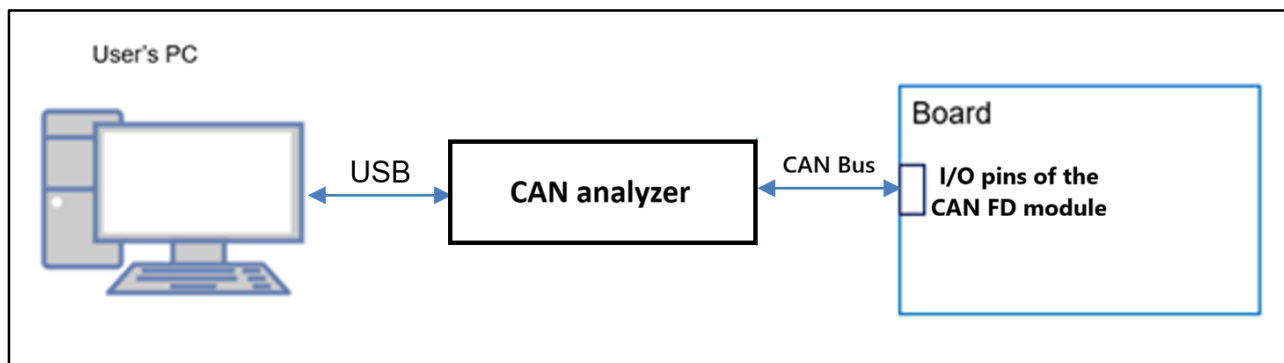


Figure 2-17 Board Connection Diagram

3. After running, the terminal will display logs such as:

- Start CAN FD demo
- CAN FD open API succeed
- Switch to internal loopback mode
- FD CAN frame transmitted successfully
- FD CAN frame received successfully

```

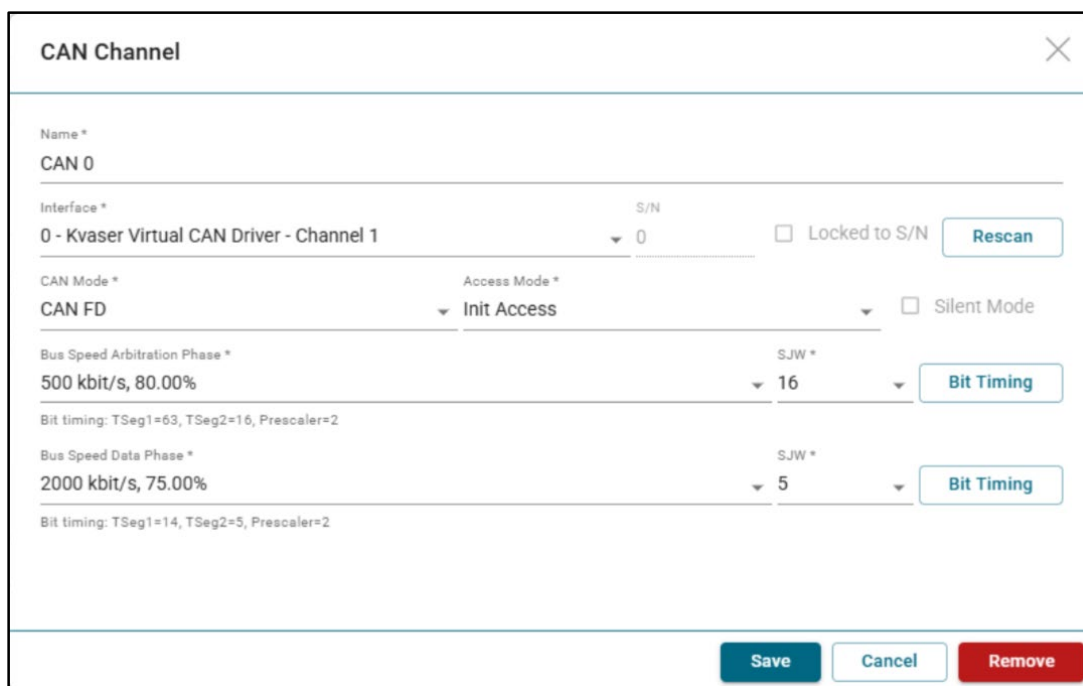
Registers Problems Smart Browser Debugger Console Renesas Debug Virtual Console X Memory

- Starting demo for CAN FD module...
- Call R_CANFD_Open() for Initialize the API...
- CANFD Open API success!
- Go to the internal loop-back test mode!
- Transmission:
  Transmission of data over FD CAN Frame
  The result of transmission: FD CAN transmission is successful!
- Go to Loop:
  + Check whether having the new message...
  + Or press sw1 to send a FD message in the internal loop-back test mode
  + Or press sw2 to send a FD message to the CAN FD analyzer/other board
  *****

****Sw2 pressed. Conduct send data to the CAN FD analyzer/other board***
- Transmission:
  Transmission of data over FD CAN Frame
  The result of transmission: FD CAN transmission is successful!

```

Figure 2-18 Output in Renesas Debug Virtual Console.



CAN Channel

Name *
CAN 0

Interface *
0 - Kvaser Virtual CAN Driver - Channel 1

S/N
0

☐ Locked to S/N Rescan

CAN Mode *
CAN FD

Access Mode *
Init Access

☐ Silent Mode

Bus Speed Arbitration Phase *
500 kbit/s, 80.00%

SJW *
16

Bit Timing

Bit timing: TSeg1=63, TSeg2=16, Prescaler=2

Bus Speed Data Phase *
2000 kbit/s, 75.00%

SJW *
5

Bit Timing

Bit timing: TSeg1=14, TSeg2=5, Prescaler=2

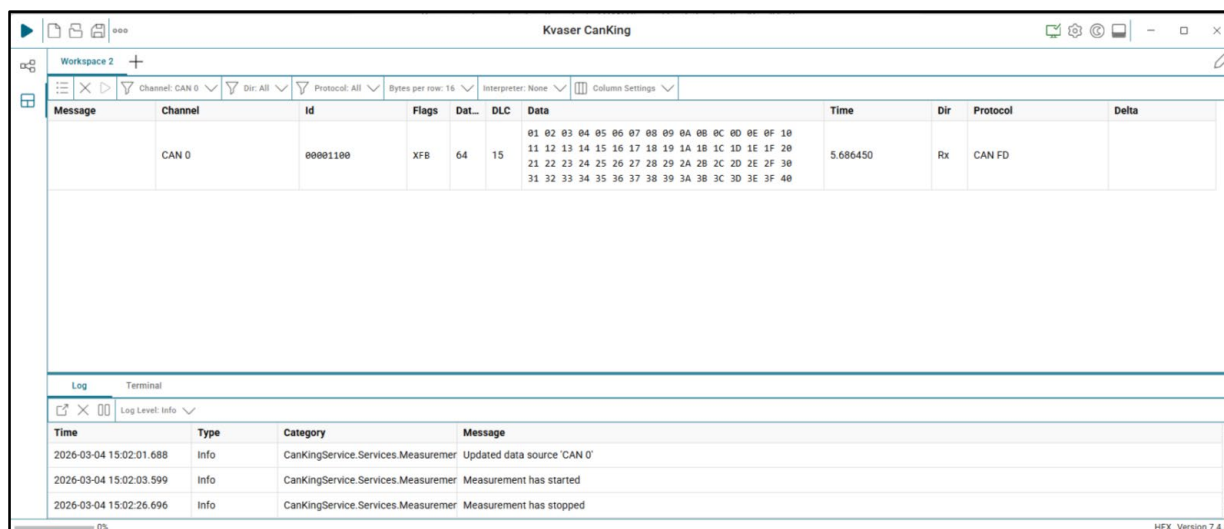
Save Cancel Remove

Figure 2-19 CAN Channel Configuration in CAN analyzer

The CAN analyzer interface is configured to operate in **CAN FD mode** with the following settings:

- **Interface:** Kvaser Virtual CAN Driver - Channel 1
- **CAN Mode:** CAN FD
- **Access Mode:** Init Access
- **Bus Speed Arbitration Phase:** 500 kbit/s
- **Bus Speed Data Phase:** 2000 kbit/s

These settings must match the CAN FD communication parameters used by the board in order to correctly transmit and receive CAN FD frames.



Kvaser CanKing

Workspace 2

Channel: CAN 0 | Dir: All | Protocol: All | Bytes per row: 16 | Interpreter: None | Column Settings

Message	Channel	Id	Flags	Dat...	DLC	Data	Time	Dir	Protocol	Delta
	CAN 0	00001100	XFB	64	15	01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40	5.686450	Rx	CAN FD	

Log | Terminal

Log Level: Info

Time	Type	Category	Message
2026-03-04 15:02:01.688	Info	CanKingService.Services.Measuremer	Updated data source 'CAN 0'
2026-03-04 15:02:03.599	Info	CanKingService.Services.Measuremer	Measurement has started
2026-03-04 15:02:26.696	Info	CanKingService.Services.Measuremer	Measurement has stopped

HEX Version 7.4.0

Figure 2-20 CAN FD Frame Output in CAN analyzer

The result confirms that the board successfully transmitted the CAN FD frame and that the data can be monitored through the external CAN analyzer.

3. Reference Documents

RX Family CAN FD API Using Firmware Integration Technology(R01AN6130)

Renesas Flexible Software Package (FSP)

RX FSP Migration Guide

Technical Update/Technical News

The latest information can be downloaded from the Renesas Electronics website.

4. Appendices

4.1 FIT & FSP Environment

This section describes FIT & FSP environment.

Item	Contents
FIT Module	r_canfd_rx v1.51
FSP Module	RX FSP v1.0.0 Module: r_canfd
Board used	Renesas EK-RX74M (product No.: RTKEK74Mxxxxxxxxx)
Integrated development environment	Renesas Electronics e ² studio
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family GCC for Renesas RX

4.2 Reference FSP Application Code

Add this FSP sample code to file "hal_entry.c"

```
#include "hal_data.h"
#include "ekrx74m.h"
#include "hal_entry.h"
#include <stdio.h>
void R_BSP_WarmStart(bsp_warm_start_event_t event);

#define CAN_MAILBOX_NUMBER_0      (0U)           //mail box number
#define CAN_CLASSIC_FRAME_DATA_BYTES (8U)       //Data Length code
for classic frame
#define CAN_FD_DATA_LENGTH_CODE   (64)          //Data Length code
for FD frame
#define CAN_ID                     (0x1100)      //ID to be updated in
transmit frame
/* Acceptance filter array parameters */
#define CANFD_FILTER_ID            (0x00001000)
#define MASK_ID                    (0x0FFFF000)
#define MASK_ID_MODE               (1)
#define ZERO                       (0U)          //Array Index value
#define WAIT_TIME                  (1000000U)     //Wait time value

/* Flag in the interrupt function */
bool b_canfd_tx_complete ;
bool b_canfd_rx_complete ;
bool b_canfd_err_status;

/* User defined functions */

void canfd_deinit(void);
static void demo_output_ports_configure(void);
static void can_write_operation(can_frame_t can_transmit_frame);
void canfd0_callback(can_callback_args_t *p_args);

/* checking switches */
extern void read_switches(void);

/* tx and rx data frame */
can_frame_t g_canfd_tx_frame;
can_frame_t g_canfd_rx_frame;

#define SIZE_64      (64u)
#define RESET_VALUE (0x00)
#define TIME_OUT_VALUE (4294967295)

/* Data to be loaded in Classic CAN and FD frames for transmission and
acknowledgement */
uint8_t tx_data[SIZE_64] = "TX_MESG";
uint8_t rx_data[SIZE_64] = "RX_MESG";
uint8_t tx_fd_data[SIZE_64];
uint8_t rx_fd_data[SIZE_64];
uint8_t rx_fd_data_real[SIZE_64];

/* Variable to store rx frame status info*/
can_info_t can_rx_info =
{
```

R01AN8210ES0100 Rev.1.00
Sep.30.26

```
/* Set time for led on/off */
#define LED_DELAY          (0x02000000)
#define Received_led       (0)
#define Transmitted_led    (1)
uint32 t j = LED_DELAY;
```

```

void hal_entry (void)
{
    /* Fill tx frame data that is to be sent in FD frame */
    for( uint16_t j = 0; j < SIZE_64; j++)
    {
        g_canfd_tx_frame.data[j] = (uint8_t) (j + 1);
    }
    /* Fill rx frame data that is the same the transmitted FD frame */
    for(uint16_t j = 0; j < SIZE_64; j++)
    {
        rx_fd_data[j] = (uint8_t) (j + 1);
    }

    /* Initializes pins for r_canfd_rx module */

    printf("\n- Starting demo for CAN FD module...\n");
    printf("\n- Call R_CANFD_Open() for Initialize the API...\n");

    fsp_err_t err;
    /* Initialize the API. */
    err = g_canfd0.p_api->open(&g_canfd0_ctrl, &g_canfd0_cfg);
    /* Error trap */
    if(FSP_SUCCESS != err)
    {

        printf("\nCANFD Open API failed\n");
        printf("Error: %d \n", err);

    }

    printf("\n- CANFD Open API success!\n");

    printf("\n- Go to the internal loop-back test mode!\n");
    /* Change to the loopback internal mode */
    g_canfd0.p_api->modeTransition(&g_canfd0_ctrl,
    (can_operation_mode_t)CAN_OPERATION_MODE_NORMAL,
    (can_test_mode_t)CAN_TEST_MODE_LOOPBACK_INTERNAL);
    /* Go to function canfd_operation in main.c */
    canfd_operation();

    /* Disable the loopback internal mode */
    g_canfd0.p_api->modeTransition(&g_canfd0_ctrl,
    (can_operation_mode_t)CAN_OPERATION_MODE_NORMAL,
    (can_test_mode_t)CAN_TEST_MODE_DISABLED);

    printf("\n- Go to Loop: \n");
    printf("\n + Check whether having the new message...\n");
    printf("\n + Or press sw1 to send a FD message in the internal loop-back
test mode \n");
    printf("\n + Or press sw2 to send a FD message to the CAN FD analyzer/other
board \n");
    printf("\n ***** \n");

    while(1)
    {
        /* Check whether having the new message... */
        can_read_operation();
    }
}

```

```

        /* Read switches(sw1 or sw2 or sw3) */
        read_switches();
    }
}

/*****
*****//**
 * @brief CAN FD callback function.
 * @param[in] p_args
 * @retval None
*****
*****/
void canfd0_callback(can_callback_args_t *p_args)
{
    switch (p_args->event)
    {
        case CAN_EVENT_TX_COMPLETE:
        {
            b_canfd_tx_complete = true;          //set flag bit
            break;
        }
        case CAN_EVENT_RX_COMPLETE:
        {
            b_canfd_rx_complete = true;          //set flag bit
            for(uint16_t i =0; i < 64; i++)
            {
                rx_fd_data_real[i] = p_args->frame.data[i];
            }
            break;
        }
        case CAN_EVENT_ERR_WARNING:              //error warning event
        case CAN_EVENT_ERR_PASSIVE:              //error passive event
        case CAN_EVENT_ERR_BUS_OFF:              //error Bus Off event
        case CAN_EVENT_BUS_RECOVERY:             //Bus recovery error event
        case CAN_EVENT_MAILBOX_MESSAGE_LOST:     //overwrite/overrun error event
        case CAN_EVENT_ERR_BUS_LOCK:             // Bus lock detected (32
consecutive dominant bits).
        case CAN_EVENT_ERR_CHANNEL:              // Channel error has occurred.
        case CAN_EVENT_TX_ABORTED:               // Transmit abort event.
        case CAN_EVENT_ERR_GLOBAL:              // Global error has occurred.
        {
            b_canfd_err_status = true;          //set flag bit
            break;
        }
    }
}

void canfd_operation(void)
{
    /* Update transmit frame parameters */
    g_canfd_tx_frame.id = CAN_ID;
    g_canfd_tx_frame.id_mode = CAN_ID_MODE_EXTENDED;
    g_canfd_tx_frame.type = CAN_FRAME_TYPE_DATA;

    /* FD CAN 64bytes*/
    g_canfd_tx_frame.data_length_code = CAN_FD_DATA_LENGTH_CODE;
    g_canfd_tx_frame.options = CANFD_FRAME_OPTION_FD | CANFD_FRAME_OPTION_BRS;

```

```

printf("\n- Transmission:");
printf("\n  Transmission of data over FD CAN Frame");

/* Transmission of data over FD CAN frame */
can_write_operation(g_canfd_tx_frame);
}

/*****
*****//**
* @brief      This function is to transmit data frame
* @param[in]  can_transmit_frame      Data frame to be transmitted
* @return     None
*****
*****/
static void can_write_operation(can_frame_t can_transmit_frame)
{
    fsp_err_t err = FSP_SUCCESS;

    /* Transmit the data from mail box #0 with tx_frame */
    g_canfd0.p_api->write(&g_canfd0_ctrl, CAN_MAILBOX_NUMBER_0,
&can_transmit_frame);
    /* Handle error */
    if (FSP_SUCCESS != err)
    {

        printf("\nCANFD Write API FAILED");

        canfd_deinit();
    }

    uint32_t g_time_out = WAIT_TIME;
    /* Wait here for an event from callback */
    while ((true != b_canfd_tx_complete) && (g_time_out--));
    if (TIME_OUT_VALUE == g_time_out)
    {

        printf("\n  Transmission failed due to timeout");

        canfd_deinit();
        while(1);
    }

    /* Check the data transmit successfully */
    if(true == b_canfd_tx_complete)
    {

        LED(0, LED_ON);
        while(j--)
        {

            ;
        }
        j = LED_DELAY;
        LED(0, LED_OFF);
    }
}

```

```

        printf("\n The result of transmission: FD CAN transmission is
successful!\n");

    }
    /* Reset flag bit */
    b_canfd_tx_complete = false;
}

/*****
*****//**
* @brief      This function is to de-initializes the CANFD module
* @param[in]  None
* @return     None
*****
*****/
void canfd_deinit(void)
{
    fsp_err_t err = FSP_SUCCESS;
    /* Close CANFD channel */
    err = g_canfd0.p_api->close(&g_canfd0_ctrl);
    if (FSP_SUCCESS != err)
    {

        printf("\n **CANFD Close API failed**");

    }
}

/*****
*****//**
* @brief      This function is to read Channel status info and read data.
* @param[in]  None
* @return     None
*****
*****/
void can_read_operation(void)
{
    fsp_err_t err = FSP_SUCCESS;

    /* Get the status information for CAN transmission */
    err = g_canfd0.p_api->infoGet(&g_canfd0_ctrl, &can_rx_info);
    /* Handle error */
    if (FSP_SUCCESS != err)
    {

        printf("\nCAN InfoGet API FAILED \n");

        canfd_deinit();
    }
    else
    {
        /* Do Nothing */
    }

    /* Check if the data is received in FIFO Buffers */
    if(true == b_canfd_rx_complete)
    {

```

```

    printf("\n- Receive: \n");
    printf("  The received data: ");
    for(uint16_t i =0; i < 30; i++)
    {
        printf("%d, ", rx_fd_data_real[i]);
    }
    printf("\n          ");
    for(uint16_t i =30; i < 64; i++)
    {
        printf("%d, ", rx_fd_data_real[i]);
    }
    printf("\n  The result of receive: FD CAN receive is successful!\n");
    while(j--)
    {
        ;
    }
    j = LED_DELAY;

    /* Reset flag bit */
    b_canfd_rx_complete = false;
}

/* Check if the data is received in Receive Message Buffers */
if(can_rx_info.rx_mb_status)
{
    /* Read the input frame received */
    err = g_canfd0.p_api->read(&g_canfd0_ctrl, ZERO, &g_canfd_rx_frame);
    /* Handle error */
    if (FSP_SUCCESS != err)
    {

        printf("\nCAN Read API FAILED");

    }
}

}

/*****
*****//**
* This function is called at various points during the startup process. This
implementation uses the event that is
* called right before main() to set up the pins.
*
* @param[in]  event      Where at in the start up process the code is currently
at
*****
*****//**
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0

        /* Enable reading from data flash. */
        R_FACI_LP->DFCTL = 1U;

```

```
        /* Would normally have to wait tDSTOP(6us) for data flash recovery.
Placing the enable here, before clock and
        * C runtime initialization, should negate the need for a delay since
the initialization will typically take more than 6us. */
#endif
    }

    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        IOPORT_CFG_OPEN(&IOPORT_CFG_CTRL, &IOPORT_CFG_NAME);
    }
}
```

4.3 Comparison Reference

4.3.1 Compare Functions (Details)

Compare API functions (details)

FIT		FSP	
FIT API (Public API)	FSP Interface API (Public API)	FSP Instance API (Private API)	
R_CANFD_Open	open	R_CANFD_Open	
R_CANFD_Close	close	R_CANFD_Close	
R_CANFD_Write	write	R_CANFD_Write	
R_CANFD_Read	read	R_CANFD_Read	
R_CANFD_GetVersion	Not supported in specific modules FSP has common function to get FSP pack version. R_FSP_VersionGet (in bsp_api.h)	Not supported in specific modules	
R_CANFD_ModeTransition	modeTransition	R_CANFD_ModeTransition	
Callback function set in R_CANFD_CallbackSet	callbackSet	R_CANFD_CallbackSet	
R_CANFD_InfoGet	infoGet	R_CANFD_InfoGet	

Compare functions in details

Open Function	
Description	Open and configure the CANFD channel for operation.
FIT	R_CANFD_Open (can_ctrl_t * const p_api_ctrl, can_cfg_t const * const p_cfg) Bitrate is configured manually only
FSP	open (can_ctrl_t * const p_api_ctrl, can_cfg_t const * const p_cfg) Bitrate can be configured either manually or Automatically

Callback Set Function	
Description	Updates the user callback with the option to provide memory for the callback argument structure.
FIT	The callback function is specified during the open operation. R_CANFD_CallbackSet (can_ctrl_t * const p_api_ctrl, void (* p_callback)(can_callback_args_t *), void const * const p_context, can_callback_args_t * const p_callback_memory)
FSP	callbackSet (can_ctrl_t * const p_api_ctrl, void (* p_callback)(can_callback_args_t *), void * const p_context, can_callback_args_t * const p_callback_memory) FSP is similar to FIT.

Close Function	
Description	Close operation.
FIT	R_CANFD_Close (can_ctrl_t * const p_api_ctrl)
FSP	close (can_ctrl_t * const p_api_ctrl) FSP is similar to FIT.

Receive Function	
Description	Read data from a CANFD Message Buffer or FIFO.
FIT	R_CANFD_Read (can_ctrl_t * const p_api_ctrl, uint32_t buffer, can_frame_t * const p_frame)
FSP	read (can_ctrl_t * const p_api_ctrl, uint32_t buffer, can_frame_t * const p_frame) FSP is similar to FIT.

Send Function	
Description	Write data to the CANFD channel.
FIT	R_CANFD_Write (can_ctrl_t * const p_api_ctrl, uint32_t buffer, can_frame_t * const p_frame)
FSP	write (can_ctrl_t * const p_api_ctrl, uint32_t buffer, can_frame_t * const p_frame) FSP is similar to FIT.

Change Mode Function	
Description	Switch to a different channel, global or test mode.
FIT	R_CANFD_ModeTransition (can_ctrl_t * const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)
FSP	modeTransition (can_ctrl_t * const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode) FSP is similar to FIT.

Get Information Function	
Description	Get CANFD state and status information for the channel.
FIT	R_CANFD_InfoGet (can_ctrl_t * const p_api_ctrl, can_info_t * const p_info)
FSP	infoGet (can_ctrl_t * const p_api_ctrl, can_info_t * const p_info) FSP is similar to FIT.

Additional Functions	
Description	List additional functions or specific function which only have in FIT but don't have in FSP and vice versa.
FIT	Get Version Function: R_CANFD_GetVersion (void) FIT provides a function to return the driver version number at runtime, not present in FSP.
FSP	None

4.3.2 Compare Configuration (Details)

Compare FIT and FSP Configuration:

FIT Configuration	FSP Configuration
Parameter Checking	Common> Parameter Checking
Number of acceptance filter list rules dedicated to Channel 0.	Common> Reception> Acceptance Filtering>Channel 0 Rule Count/Channel 1 Rule Count
Enable/Disable Flexible Data Protocol Exceptions	Common> Flexible Data (FD)> Protocol Exceptions
Global error interrupt	Common> Global error interrupt
Transmission Priority	Common> Transmission Priority
DLC Check	Common> DLC Check
Flexible Data Payload Overflow	Common> Flexible Data (FD)> Payload Overflow
CAN Clock Source	Clock Configuration
RX Message Buffers	Common> Reception> Message Buffers> Number of Buffers
Payload size for all RX Message Buffers.	Common> Reception> Message Buffers> Payload Size
Global Error Interrupt Priority	Common> Global Error Interrupt> Priority
Reception FIFOs Interrupt Priority	Common> Reception FIFOs Interrupt Priority
Reception FIFOs FIFO 0 Interrupt Threshold	Common> Reception FIFOs FIFO 0 Interrupt Threshold
Reception FIFOs FIFO 0 Depth	Common> Reception FIFOs FIFO 0 Depth
Reception FIFOs FIFO 0 Payload Size	Common> Reception FIFOs FIFO 0 Payload Size
Reception FIFOs FIFO 0 Interrupt Mode	Common> Reception FIFOs FIFO 0 Interrupt Mode
Reception FIFOs FIFO 0 Enable	Common> Reception FIFOs FIFO 0 Enable
Reception FIFOs FIFO 1 Interrupt Threshold	Common> Reception FIFOs FIFO 1 Interrupt Threshold
Reception FIFOs FIFO 1 Depth	Common> Reception FIFOs FIFO 1 Depth
Reception FIFOs FIFO 1 Payload Size	Common> Reception FIFOs FIFO 1 Payload Size
Reception FIFOs FIFO 1 Interrupt Mode	Common> Reception FIFOs FIFO 1 Interrupt Mode
Channel Error/Transmit interrupt priority	Channel Error/Transmit interrupt priority
Transmit Interrupts TX0	Transmit Interrupts TX0
Transmit Interrupts TX1	Transmit Interrupts TX1
Transmit Interrupts TX2	Transmit Interrupts TX2
Transmit Interrupts TX3	Transmit Interrupts TX3

FIT Configuration	FSP Configuration
Channel Error Interrupts Error Warning	Channel Error Interrupts Error Warning
Channel Error Interrupts Error Passive	Channel Error Interrupts Error Passive
Channel Error Interrupts Bus-Off Entry	Channel Error Interrupts Bus-Off Entry
Channel Error Interrupts Bus-Off Recovery	Channel Error Interrupts Bus-Off Recovery
Channel Error Interrupts Overload	Channel Error Interrupts Overload
Bitrate Manual Nominal - Prescaler (divisor) - Time Segment 1 (Tq) - Time Segment 2 (Tq) - Sync Jump Width (Tq)	Module> Bitrate Manual Nominal - Prescaler (divisor) - Time Segment 1 (Tq) - Time Segment 2 (Tq) - Sync Jump Width (Tq)
Bitrate Manual Data - Prescaler (divisor) - Time Segment 1 (Tq) - Time Segment 2 (Tq) - Sync Jump Width (Tq)	Module> Bitrate Manual Data - Prescaler (divisor) - Time Segment 1 (Tq) - Time Segment 2 (Tq) - Sync Jump Width (Tq)
Bitrate Delay Compensation	Module> Bitrate Delay Compensation

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep 30, 2026	-	First edition issued

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/.