RX Family

PTX NFC Control Module Using Firmware Integration Technology

Introduction

This application note describes the usage of the PTX NFC control module, which conforms to the Firmware Integration Technology (FIT) standard.

In the following pages, the PTX NFC control module software is referred to collectively as “the PTX FIT module” or “the FIT module.”

The FIT module supports the following NFC PMOD module:

- PTX105R (PTX105RQC)

In the following pages, the PTX105R is referred to as “the NFC module”.

Target Device

- RX Family
- RX200 Series
- RX261 Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family

For details of the confirmed operation contents of each compiler, refer to “5.1 Confirmed Operation Environment”.

Related Documents

[1] RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
[2] RX Smart Configurator User’s Guide: e² studio (R20AN0451)
[3] RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
[4] RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)
[5] RX Family CMT Module Using Firmware Integration Technology (R01AN1856)
[6] RX Family GPIO Module Using Firmware Integration Technology (R01AN1721)
[7] RX Family IRQ Module Using Firmware Integration Technology (R01AN1668)
[8] EK-RX261 v1 – User's Manual (R20UT5351)

Contents

1. Overview 4
1.1 PTX NFC FIT Module 4
1.2 Overview of the PTX NFC FIT Module 4
1.2.1 Connection with the PTX NFC Module 4
1.2.2 Hardware Configuration 5
1.2.3 Software Configuration 6
1.3 API Overview 7
1.4 Status Transitions 8

2. API Information 9
2.1. Hardware Requirements 9
2.2. Software Requirements 9
2.3. Supported Toolchain 9
2.4. Interrupt Vector 9
2.5. Header Files 9
2.6. Integer Types 9
2.7. Compile Settings 10
2.8. Code Size 12
2.9. Parameters 13
2.10. Return Values 14
2.11. Adding the FIT Module to Your Project 15
2.12. “for”, “while” and “do while” statements 16

3. API Functions 17
3.1. R_NFC_PTX_Open() 17
3.2. R_NFC_PTX_StartDiscovery() 18
3.3. R_NFC_PTX_GetStatus() 19
3.4. R_NFC_PTX_GetCardRegistry() 20
3.5. R_NFC_PTX_ActivateCard() 21
3.6. R_NFC_PTX_DataExchange() 22
3.7. R_NFC_PTX_ReaderDeactivation() 23
3.8. R_NFC_PTX_SWReset() 24
3.9. R_NFC_PTX_Close() 25

4. Demo Projects 26
4.1. NFC PTX105RQC Get Card Information Demo Project 26
4.1.1 Prerequisites 26
4.1.2 Import the Demo Project 26
4.1.3 Hardware Setup 26
4.1.4 How to Run the Demo 27
4.1.5 Understanding NFC Data Fields in the Example 32

4.2. Adding a Demo to a Workspace 34
4.3. Downloading Demo Projects 34

5. Appendices 35
5.1 Confirmed Operation Environment 35
5.2 Troubleshooting 36

6. Reference Documents 37

Revision History 38

1. Overview

1.1 PTX NFC FIT Module

The PTX NFC FIT module can be used by being implemented in a project as an API. See section

2.11 Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the PTX NFC FIT Module

The PTX NFC module supplies an implementation for the NFC IoT Reader on the PTX module. The NFC functionalities are provided via a modular NFC Soft Controller (NCS) architecture, and this FIT module will include the NSC (known as the NFC SDK). Functionality is provided in a split-stack architecture, where time-critical operations are running on the on-chip MCU, and the rest of the NFC logic is handled by the host controller to carry out applications such as the IoT Reader. The driver may also be adapted to further support next-generation PTX devices in future (which have not yet been released).

- The PTX NFC FIT Module supplies these features:

- Initializes the IOTRD (IoT Reader) API and NSC stack
- Initializes the PTX chip
- Discovers cards according to NFC Forum
- Selects a specific card in case multiple cards/protocols were discovered
- Retrieves card details like technical/activation parameters
- Exchanges RF data and bitstreams
- Stops RF communication

1.2.1 Connection with the PTX NFC Module

An example connection to the PTX NFC module is shown below.

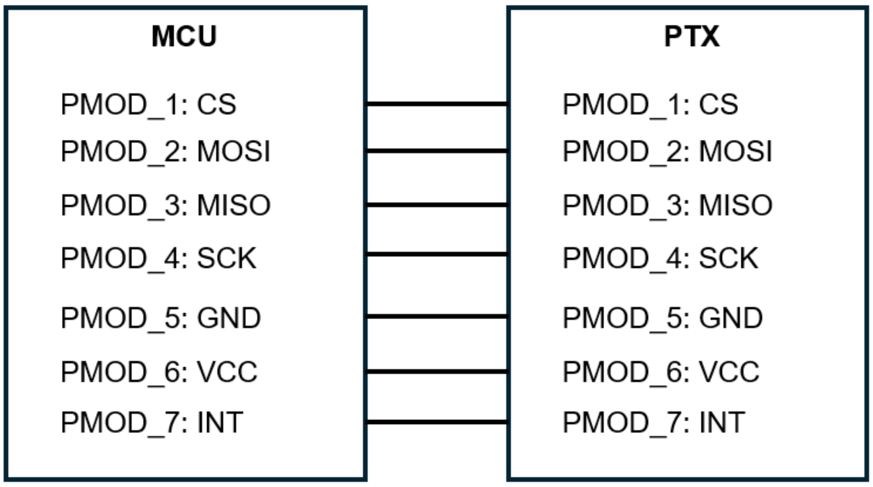


Figure 1.1 Example Connection to the PTX NFC Module

1.2.2 Hardware Configuration

The hardware configuration for MCU host and the PTX105R NFC PMOD module is shown below.

Table 1.1 MCU Host Hardware Configuration

Item	Content	Description
EK-RX261 Evaluation Kit [*1]	Target board for EK-RX261 Part number: RTK5EK2610S00001BE	Please see detail at: https://www.renesas.com/rx/ek-rx261

Note 1: The EK-RX261 Evaluation Kit only supports PMOD1 for the NFC PMOD module

Table 1.2 PMOD Module Hardware Configuration

Item	Content	Description
PTX105R NFC PMOD module	NFC IoT Reader connection	This PMOD is used with MCU host for NFC connection. Please see detail at: PTX105RQC - PTX105R PMOD™ Board for IoT | Renesas

1.2.3 Software Configuration

Figure 1.2 shows the software configuration.

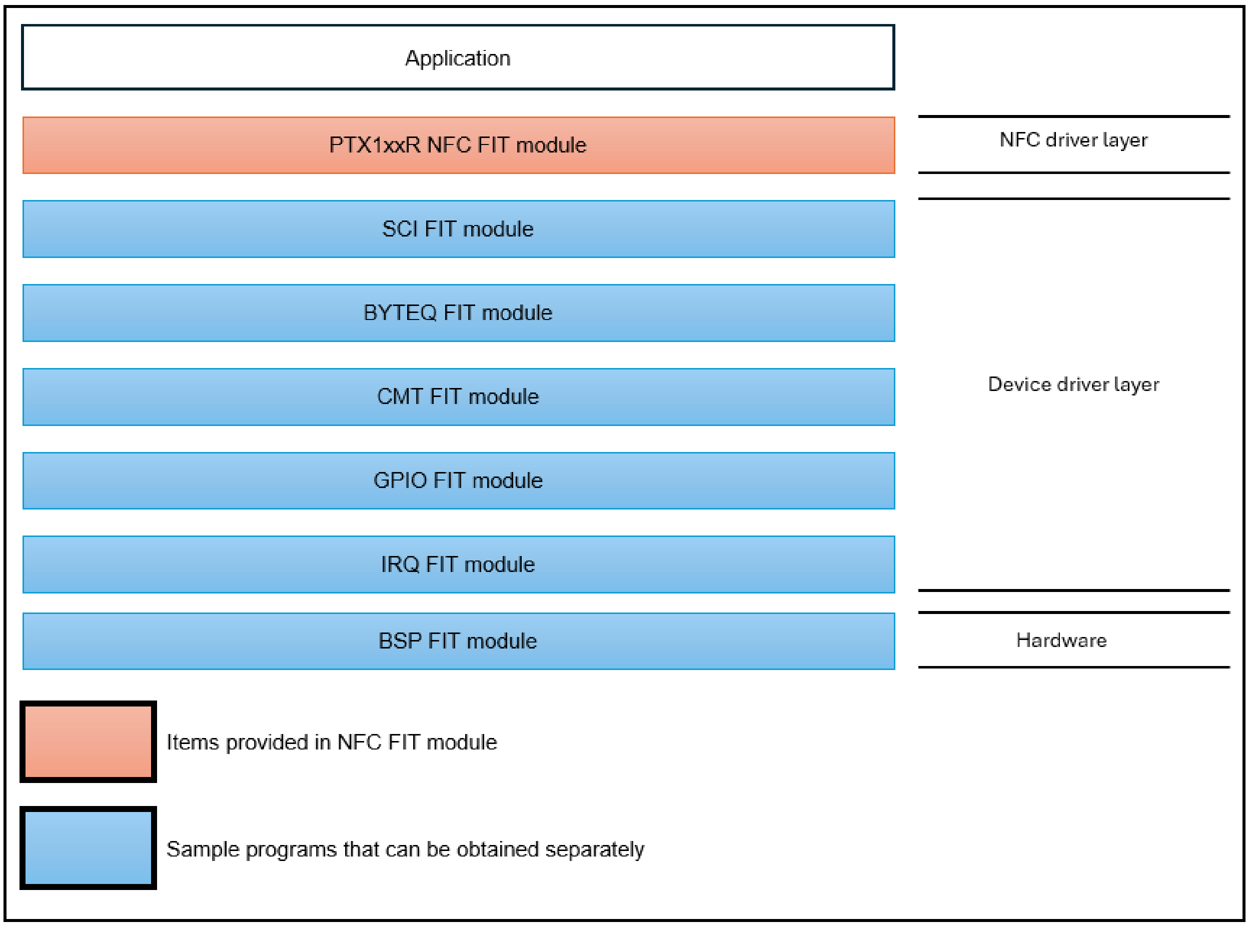


Figure 1.2 Software Configuration Diagram

1. PTX NFC FIT module
   The FIT module. This software is used to control the NFC module.
2. SCI FIT module
   Implements communication between the NFC module and the MCU. A sample program is available.
   Refer to “Related Documents” on page 1 and obtain the software.
3. BYTEQ FIT module
   Implements circular buffers used by the SCI FIT module. A sample program is available.
   Refer to “Related Documents” on page 1 and obtain the software.
4. CMT FIT module
   Implements compare match timer used by the PTX NFC FIT module. A sample program is available.
   Refer to “Related Documents” on page 1 and obtain the software.
5. GPIO FIT module
   Implements pin direction set function used by the PTX NFC FIT module. A sample program is available.
   Refer to “Related Documents” on page 1 and obtain the software.
6. IRQ FIT module
   Implements external IRQ source used by the PTX NFC FIT module. A sample program is available.
   Refer to “Related Documents” on page 1 and obtain the software.
7. BSP FIT module
   The Board Support Package module. A sample program is available.
   Refer to “Related Documents” on page 1 and obtain the software.

1.3 API Overview

Table 1.3 lists the API functions included in the FIT module. The required memory sizes are lists in 2.8 Code Size

Table 1.3 API Functions

Function	Function Description
R_NFC_PTX_Open()	Initialize the NFC module
R_NFC_PTX_StartDiscovery()	Initiates discovery of NFC tags
R_NFC_PTX_GetStatus()	Gets the status information on the provided status identifier (system, discovery, etc.)
R_NFC_PTX_GetCardRegistry()	Gets the internal card registry
R_NFC_PTX_ActivateCard()	Gets the details of the discovered card and connects to it
R_NFC_PTX_DataExchange()	Sends and receives data to/from the activated card
R_NFC_PTX_ReaderDeactivation()	Deactivates the activated card to return to the desired state (idle, discovery, etc.).
R_NFC_PTX_SWReset()	Performs a soft-reset of the PTX chip.
R_NFC_PTX_Close()	Closes the NFC module and resets all variables

1.4 Status Transitions

Figure 1.3 shows the status transitions of the FIT module.

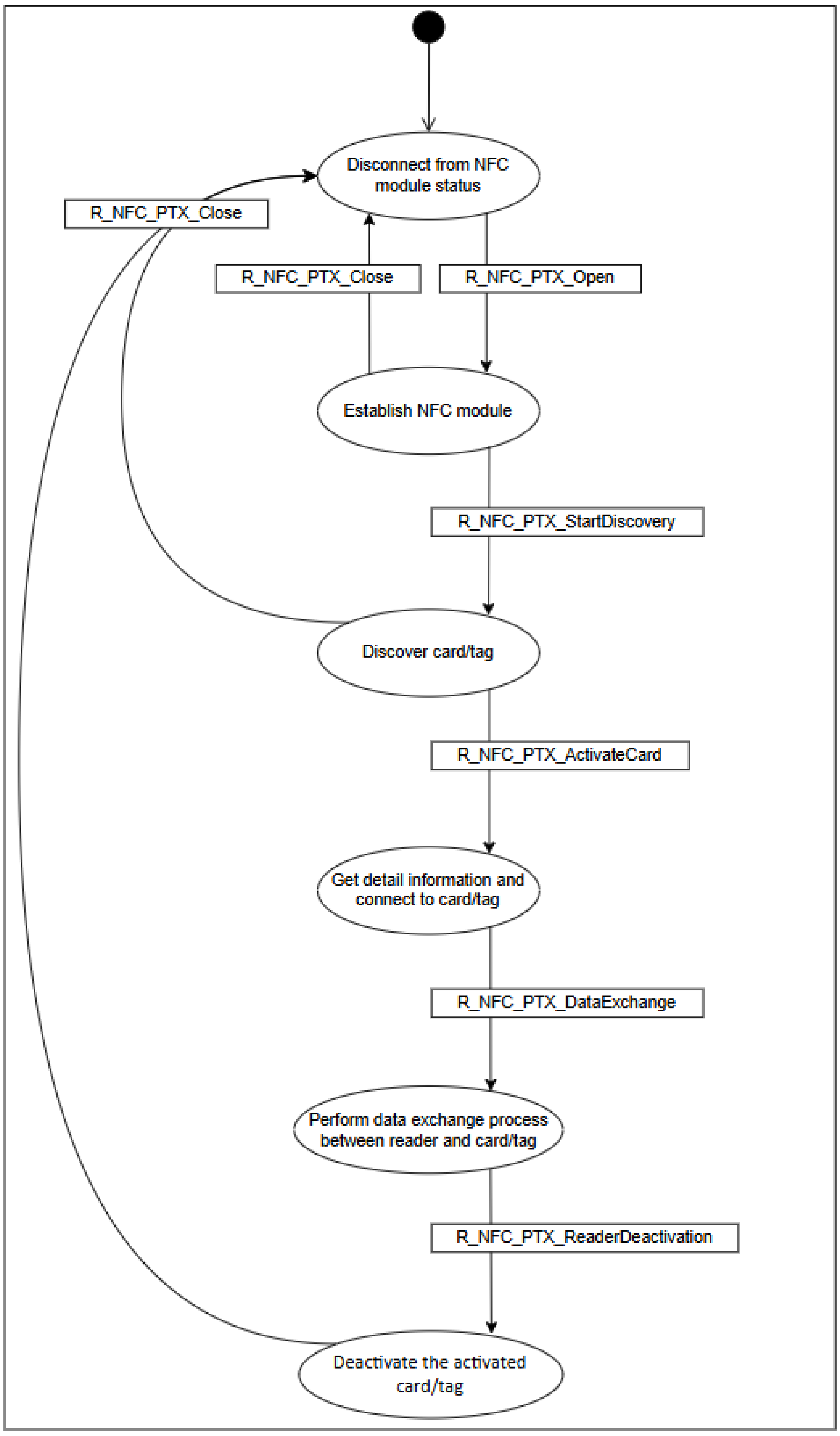

Disconnect from NFC module status
R_NFC_PTX_Close
R_NFC_PTX_Close
R_NFC_PTX_Open
Establish NFC module
R_NFC_PTX_StartDiscovery
Discover card/tag
R_NFC_PTX_ActivateCard
Get detail information and connect to card/tag
R_NFC_PTX_DataExchange
Perform data exchange process between reader and card/tag
R_NFC_PTX_ReaderDeactivation
Deactivate the activated card/tag

Figure 1.3 Status Transitions

(**Note**: can check detail state machine in PTX1xxR NFC IoT-Reader API Non-OS Stack Integration User Manual).

2. API Information

This FIT module has been confirmed to operate under the following conditions.

2.1. Hardware Requirements

The MCU used must support the following functions:

- Serial communication
- I/O ports

2.2. Software Requirements

The driver is dependent upon the following FIT modules:

- r_bsp
- r_sci_rx
- r_byteq_rx
- r_cmt_rx
- r_gpio_rx
- r_irq_rx

2.3. Supported Toolchain

The FIT module has been confirmed to work with the toolchain listed in 5.1 Confirmed Operation Environment.

2.4. Interrupt Vector

None

2.5. Header Files

All API calls and their supporting interface definitions are located in r_nfc_ptx_if.h.

2.6. Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.7. Compile Settings

The configuration option settings of the FIT module are contained in r_nfc_ptx_rx_config.h.
The names of the options and their setting values are listed in the table below.

Table 2.1 Configuration Options (r_nfc_ptx_rx_config.h)

Configuration Options in r_nfc_ptx_rx_config.h	
NFC_CFG_EXT_IRQ_PORT Default: “D”	Configure the general port that controls IRQ port triggered by PTX to indicate to host that data is available. Set this option to match the port to be controlled.
NFC_CFG_EXT_IRQ_PIN Default: “5”	Configure the general pin that controls IRQ pin triggered by PTX to indicate to host that data is available. Set this option to match the pin to be controlled
NFC_CFG_IRQ_NUM Default: “5”	IRQ number that connected to IRQ pin of PTX Set this option to match the IRQ number to be controlled.
NFC_CFG_SCI_CHANNEL Default: “5”	SCI channel for serial communication with PTX. Set this option to match the SCI port to be controlled.
NFC_CFG_SCI_SSPI_SS_PORT Default: “C”	Configure the general port that controls SS port (for SPI interface only). Set this option to match the port to be controlled.
NFC_CFG_SCI_SSPI_SS_PIN Default: “0”	Configure the general pin that controls SS pin (for SPI interface only). Set this option to match the pin to be controlled.
NFC_CFG_SCI_MODE Default: “0”	Configure SCI mode for communication 0 = Simple SPI
NFC_CFG_SCI_SSPI_SPEED Default: “1,000,000”	Configure SCI SSPI bitrate Set this value in range from 1,000,000 to 10,000,000
NFC_SCI_SPI_MSB_FIRST Default: “1”	Configure SCI data transfer mode 0: LSB first transfer, 1: MSB first transfer
NFC_SCI_SPI_INVERT_DATA Default: “0”	Set to 1 if need to invert logic level of data transmission 0: Disable, 1: Enable
NFC_CFG_POLL_TYPE_A Default: “1”	Select to discover NFC Type-A tags 0: Disable, 1: Enable
NFC_CFG_POLL_TYPE_B Default: “1	Select to discover NFC Type-B tags 0: Disable, 1: Enable
NFC_CFG_POLL_TYPE_F Default: “1”	Select to discover NFC Type-F tags 0: Disable, 1: Enable
NFC_CFG_POLL_TYPE_V Default: “1”	Select to discover NFC Type-V tags 0: Disable, 1: Enable
NFC_CFG_TEMP_SENSOR_SHUTDOWN	Set the expected thermal shutdown threshold value (Celsius). Set this value in range from 1 to 222
NFC_CFG_TEMP_SENSOR_AMBIENT	Set the ambient temperature value at which calibration takes place (Celsius). Set this value in range from 1 to 84
NFC_CFG_DEVICE_LIMIT	Select the max number of tags to discover Set this value in range from 1 to 49
NFC_CFG_IDLE_TIME_MS	Select idle time between polling cycles (ms) Set this value in range from 1 to 65534

Table 2.2 Configuration Options (r_sci_rx_config.h)

Configuration Options in r_ sci_rx_config.h	
#define SCI_CFG_SSPI_INCLUDED Note: The default is 0.	Specifies whether to include code for SSPI mode. This option needs to be set to 1 in this v1.00 (only support SPI).
#define SCI_CFG_CHx_INCLUDED Notes: 1. CHx = CH0 to CH12 2. The default values are as follows: CH0 CH2 to CH12: 0, CH1: 1	Each channel has resources such as transmit and receive buffers, counters, interrupts, other programs, and RAM. Setting this option to 1 assigns related resources to the specified channel.
#define SCI_CFG_TEI_INCLUDED Note: The default is 0.	Enables the transmit end interrupt for serial transmissions. This option should be set to 1.

2.8. Code Size

Typical code sizes associated with this module are listed below.
The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7 Compile Settings. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.3 Supported Toolchain. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

The values in the table below are confirmed under the following conditions.

Module Revision: r_nfc_ptx_rx rev1.00.
Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.07.00
(The option of "-lang=c99" is added to the default settings of the integrated development environment.)
Configuration Options: Default settings.

Table 2.3 Memory Sizes for RX261

Device	Memory Type	Category	Memory usage
			Renesas Compiler (CCRX)
RX261	With platform modules	ROM	62576 bytes
		RAM	2806 bytes
	Without platform modules	ROM	59253 bytes
		RAM	2694 bytes

(**Note**: platform modules are low-level modules: SCI, GPIO, IRQ, CMT).

2.9. Parameters

This section describes the parameter structure used by the API functions in this module. The structure is located in r_nfc_ptx_if.h as are the prototype declarations of API functions.

Table 2.4 Member in Structure for Data Exchange Packet (nfc_reader_ptx_data_info_t)

Type	Name	Description
uint8_t *	p_tx_buf	Pointer to buffer holding data to send
uint32_t	tx_length	Length of data to send
uint8_t *	p_rx_buf	Pointer to buffer for data to receive
uint32_t	rx_length	Length of data to receive

Table 2.5 Member in Structure for NFC Configuration Block (nfc_reader_ptx_cfg_t)

Type	Name	Description
bool	poll_type_a	Flag to indicate enabling discovery for Type-A tags
bool	poll_type_b	Flag to indicate enabling discovery for Type-B tags
bool	poll_type_f	Flag to indicate enabling discovery for Type-F tags
bool	poll_type_v	Flag to indicate enabling discovery for Type-V tags
uint32_t	idle_time_ms	Idle time between polling cycles in milliseconds
uint8_t	temp_sensor_calibrate	Flag to enable/disable temperature sensor calibration
uint8_t	temp_sensor_shutdown	Expected thermal shutdown threshold value
uint8_t	temp_sensor_ambient	Ambient temperature at which temperature sensor calibration takes place
ptxIoTRd_t *	iot_reader_context	Instance of the NFC SDK IoT Reader main structure

Table 2.6 Member in Structure for NFC State Machine Status (nfc_reader_ptx_state_t)

Value	Name	Description
1	NFC_PTX_IDLE	Idle state before starting discovery
2	NFC_PTX_POLLING	Polling state for discovering tags
3	NFC_PTX_DISCOVERED	Discovered state for tags that were found
4	NFC_PTX_ACTIVATED	Activated state for activating a discovered tag
5	NFC_PTX_ERROR_TEMP	Error state for temperature sensor
6	NFC_PTX_ERROR_OVERCURRENT	Error state for overcurrent
7	NFC_PTX_ERROR_RF	Error state for RF error

Table 2.7 Member in Structure for NFC Control Block (nfc_reader_ptx_ctrl_t)

Type	Name	Description
uint32_t	open	Flag to indicate if NFC has been opened.
nfc_reader_ptx_state_t	state_flag	Flag to track Idle/discovered/activated.
nfc_reader_ptx_cfg_t const *	p_cfg	Pointer to configuration block for NFC.

2.10. Return Values

This section describes return values of API functions. This enumeration is located in r_nfc_ptx_if.h as are the prototype declarations of API functions.

Table 2.9 API Error Codes (nfc_err_t)

Value	Error code	Description
0	NFC_SUCCESS	OK, no error
1	NFC_ERR_ASSERTION	Parameters assertion
2	NFC_ERR_INVALID_DATA	Input data input is invalid or incomplete
3	NFC_ERR_NOT_OPEN	NFC module has not been opened
4	NFC_ERR_ALREADY_OPEN	NFC module had been initialized
5	NFC_ERR_INVALID_STATE	Invalid status state
6	NFC_ERR_INVALID_ARGUMENT	Invalid input argument

2.11. Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) or (5) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e^2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: CS+ (R20AN0470)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

(5) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: IAREW (R20AN0535)” for details.

2.12. “for”, “while” and “do while” statements

In FIT module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

This FIT module does not have any WAIT_LOOP. But others might have. Please take care for this WAIT_LOOP.

3. API Functions

3.1. R_NFC_PTX_Open()

This function initializes the NFC IoT Reader.

Format

```
nfc_err_t R_NFC_PTX_Open(
      nfc_reader_ptx_ctrl_t * const p_ctrl,
      nfc_reader_ptx_cfg_t const * const p_cfg
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure
p_cfg	Pointer to NFC IoT Reader configuration structure

Return values

NFC_SUCCESS	Function completed successfully
NFC_ERR_ASSERTION	Assert error occurred
NFC_ERR_INVALID_DATA	NFC SDK initialization data was incomplete

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Initialize NFC IoT Reader.

Reentrant

No

Example

```
  nfc_reader_ptx_ctrl_t        p_ctrl;
  const nfc_reader_ptx_cfg_t   p_cfg;
  err = R_NFC_PTX_Open(&p_ctrl, &p_cfg);
```

Special Notes:

None

3.2. R_NFC_PTX_StartDiscovery()

This function initiates discovery of NFC tags.

Format

```
nfc_err_t R_NFC_PTX_StartDiscovery(
      nfc_reader_ptx_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure

Return values

NFC_SUCCESS	Function completed successfully
NFC_ERR_ASSERTION	Assertion error occurred
NFC_ERR_NOT_OPEN	Module is not opened yet
NFC_ERR_INVALID_STATE	NFC module is not in the idle state
NFC_ERR_INVALID_DATA	NFC Discovery parameters were invalid or discovery was not started

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Initiates discovery of NFC tags.

Reentrant

No

Example

```
nfc_reader_ptx_ctrl_t        p_ctrl;
nfc_err_t                    err;
err = R_NFC_PTX_StartDiscovery(&p_ctrl);
```

Special Notes:

None

3.3. R_NFC_PTX_GetStatus()

This function gets the status information on the provided status identifier (system, discovery, etc.). Used during polling to confirm a card was discovered

Format

```
nfc_err_t R_NFC_PTX_GetStatus(
      nfc_reader_ptx_ctrl_t   const p_ctrl,
      ptxIoTRd_StatusType_t   status_type,
      uint8_t * const   p_status
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure
status_type	Status identifier for target info
p_status	Pointer to location to store NFC chip status flag

Return values

NFC_SUCCESS	Function completed successfully
NFC_ERR_ASSERTION	Assertion error occurred
NFC_ERR_NOT_OPEN	Module is not opened yet
NFC_ERR_INVALID_STATE	NFC module is not in the polling state
NFC_ERR_INVALID_ARGUMENT	Invalid input parameters to NFC Status function

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Gets the status information on the provided status identifier.

Uses during polling to confirm a card was discovered.

Reentrant

No

Example

```
nfc_reader_ptx_ctrl_t        p_ctrl;
ptxIoTRd_StatusType_t        status_type = StatusType_Discover;
uint8_t                      p_status = PTX_SYSTEM_STATUS_OK;
nfc_err_t                    err;
err = R_NFC_PTX_GetStatus(&p_ctrl, status_type, &p_status);
```

Special Notes:

None

3.4. R_NFC_PTX_GetCardRegistry()

This function gets the internal card registry.

Format

```
nfc_err_t R_NFC_PTX_GetCardRegistry(
      nfc_reader_ptx_ctrl_t * const p_ctrl,
      ptxIoTRd_CardRegistry_t       pp_card_registry
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure
pp_card_registry	Pointer to card registry for discovered cards

Return values

NFC_SUCCESS	Function completed successfully
NFC_ERR_ASSERTION	Assertion error occurred
NFC_ERR_INVALID_ARGUMENT	Invalid input parameters to NFC registry function

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Gets the status information on the provided status identifier.

Uses during polling to confirm a card was discovered.

Reentrant

No

Example

```
nfc_reader_ptx_ctrl_t        p_ctrl;
ptxIoTRd_CardRegistry_t      *card_registry = NULL;
nfc_err_t                    err;
err = R_NFC_PTX_GetCardRegistry(&p_ctrl, &card_registry);
```

Special Notes:

None

3.5. R_NFC_PTX_ActivateCard()

This function gets the details of the discovered card and connects to it.

Format

```
nfc_err_t R_NFC_PTX_ActivateCard(
      nfc_reader_ptx_ctrl_t * const p_ctrl,
      ptxIoTRd_CardParams_t * const p_card_params,
      ptxIoTRd_CardProtocol_t       protocol
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure
p_card_params	Parameters of the card to be activated
protocol	The NFC protocol of the card to be activated

Return value

NFC_SUCCESS	Function completed successfully
NFC_ERR_ASSERTION	Assertion error occurred

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Connects and gets the details of the discovered card

Reentrant

No

Example

```
nfc_reader_ptx_ctrl_t        p_ctrl;
ptxIoTRd_CardRegistry_t      *card_registry    = NULL;
ptxIoTRd_CardProtocol_t      prot = Prot_T2T;
nfc_err_t                    err;
err = R_NFC_PTX_ActivateCard(&p_ctrl,&card_registry->Cards[0],prot);
```

Special Notes:

None

3.6. R_NFC_PTX_DataExchange()

This function sends and receives data to/from the activated card

Format

```
nfc_err_t R_NFC_PTX_DataExchange(
      nfc_reader_ptx_ctrl_t * const       p_ctrl,
      nfc_reader_ptx_data_info_t * const  p_data_info
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure
p_data_info	Pointer to the NFC TX/RX data

Return values

NFC_SUCCESS	Function completed successfully
NFC_ERR_ASSERTION	Assertion error occurred

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Sends and receives data to/from the activated card.

Reentrant

No

Example

```
nfc_reader_ptx_ctrl_t       p_ctrl;
uint8_t q_nfc_tx_buf[100] = {0};
uint8_t q_nfc_rx_buf[100] = {0};
nfc_reader_ptx_data_info_t nfc_buf =
{
    .p_tx_buf  = nfc_tx_buf,
    .tx_length = sizeof(nfc_tx_buf),
    .p_rx_buf  = nfc_rx_buf,
    .rx_length = sizeof(nfc_rx_buf),
};
err = R_NFC_PTX_DataExchange(&p_ctrl, &nfc_buf);
```

Special Notes:

None

3.7. R_NFC_PTX_ReaderDeactivation()

This function deactivates the activated card to return to the desired state (idle, discovery, etc.).

Format

```
nfc_err_t R_NFC_PTX_ReaderDeactivation(
      nfc_reader_ptx_ctrl_t * const       p_ctrl,
      nfc_reader_ptx_return_state_t       return_state
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure
return_state	Expectation for end state of NFC state machine

Return values

NFC_SUCCESS	Function completed successfully
NFC_ERR_ASSERTION	Assertion error occurred

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Deactivates the activated card to return to the desired state.

Reentrant

No

Example

```
nfc_reader_ptx_ctrl_t               p_ctrl;
nfc_reader_ptx_return_state_t      return_state = NFC_PTX_RETURN_IDLE;
err = R_NFC_PTX_ReaderDeactivation(&p_ctrl, return_state);
```

Special Notes:

None

3.8. R_NFC_PTX_SWReset()

This function resets system state, card registry and IoT Reader instance.

Format

```
nfc_err_t R_NFC_PTX_SWReset(
      nfc_reader_ptx_ctrl_t * const        p_ctrl
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure

Return values

NFC_SUCCESS	Function completed successfully
NFC_ERR_INVALID_ARGUMENT	Invalid input parameters to NFC Status function

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Performs a soft-reset of the PTX chip.

Reentrant

No

Example

```
 nfc_reader_ptx_ctrl_t        p_ctrl;
 err = R_NFC_PTX_SWReset(&p_ctrl);
```

Special Notes:

None

3.9. R_NFC_PTX_Close()

This function closes the NFC module and resets all variables.

Format

```
nfc_err_t R_NFC_PTX_Close(
      nfc_reader_ptx_ctrl_t * const       p_ctrl
)
```

Parameters

p_ctrl	Pointer to NFC IoT Reader control structure

Return values

NFC_SUCCESS	Function completed successfully
NFC_ERR_INVALID_ARGUMENT	Invalid input parameters to NFC Status function

Properties

Prototype declarations are contained in r_nfc_ptx_if.h.

Description

Closes the FSP NFC module and resets all variables.

Reentrant

No

Example

```
nfc_reader_ptx_ctrl_t       p_ctrl;
err = R_NFC_PTX_Close(&p_ctrl);
```

Special Notes:

None

4. Demo Projects

Demo projects include function main() that utilizes the FIT module and its dependent modules (e.g. r_bsp). This FIT module includes the following demo projects.

4.1. NFC PTX105RQC Get Card Information Demo Project

4.1.1 Prerequisites

- Hardware requirements:
  - EK-RX261: Renesas Evaluation Kit for RX261 v1 (Product no.: RTK5EK2610S00001BE)
  - PTX105R: PTX105R PMOD™ Board for IoT as NFC module
  - PC running Windows 10.
  - Micro-USB cables for Debugging and the serial log output (included as part of the kit. See EK-RX261 v1 – User's Manual at “Related Documents” on page 1).
- Software requirements for Windows 10 PC
  - IDE: e2 studio 2025-04 or later.
  - Compiler: Renesas Electronics C/C++ Compiler for RX Family V3.07.00.
  - Tera Term v4.106 or later.

4.1.2 Import the Demo Project

User can import the demo project by adding the demo to their e^2 studio workspace (see section 4.2 Adding a Demo to a Workspace or by downloading the demo project (see section 4.3 Downloading Demo Projects).

4.1.3 Hardware Setup

- Connect the NFC PTX105RQC PMOD module to the EK-RX261 **PMOD1** connector.
- Connect a micro-USB cable from the PC to the EK-RX261 micro-USB connector (**J26**) for power supply.
- Connect a micro-USB cable from the PC to the EK-RX261 micro-USB connector (**J16**) for serial log output.
- Open **J24** to enable E2 on-board debugger circuit.
- Short **pins 1-2** of **J31** to enable SCK pin of PMOD SPI.
- Short **pins 1-2** of **J32** to enable MISO pin of PMOD SPI.

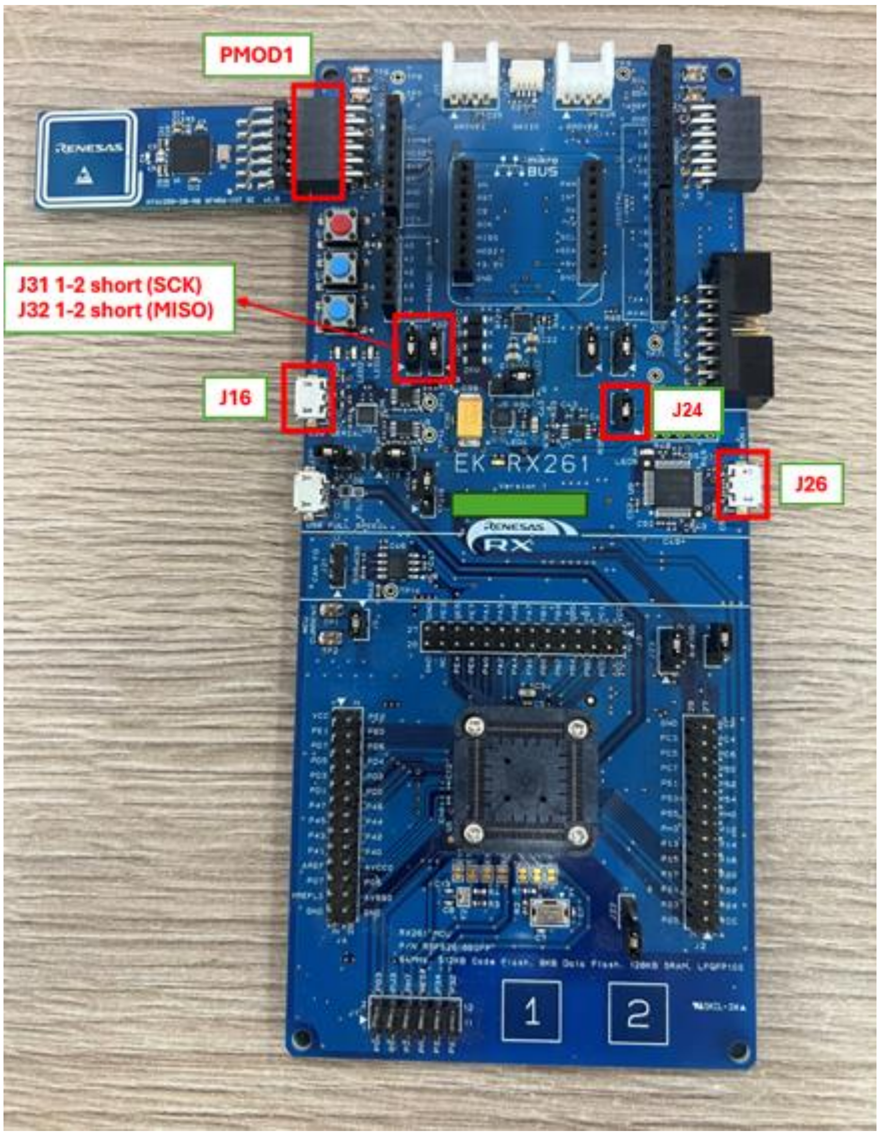
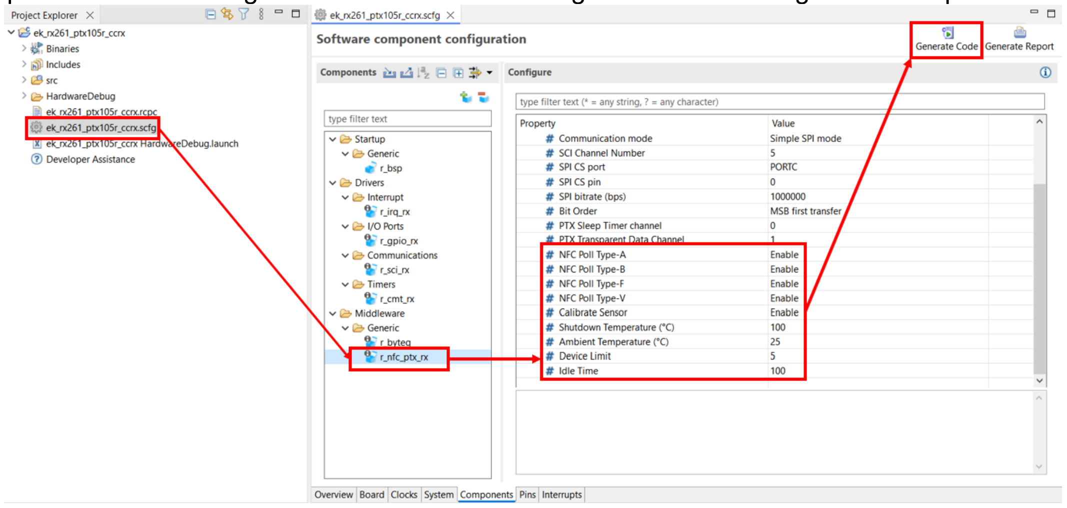


Figure 4.1 Hardware Setup

4.1.4 How to Run the Demo

a) Configure Basic Settings required to discover, activate, and exchange data with an NFC tag. Open the Smart Configurator as shown in the image below and change essential parameters.



Property	Value
Communication mode	Simple SPI mode
SCI Channel Number	5
SPI CS port	PORTC
SPI CS pin	0
SPI bitrate (bps)	1000000
Bit Order	MSB first transfer
PTX Sleep Timer channel	0
PTX Transparent Data Channel	1
NFC Poll Type-A	Enable
NFC Poll Type-B	Enable
NFC Poll Type-F	Enable
NFC Poll Type-V	Enable
Calibrate Sensor	Enable
Shutdown Temperature (°C)	100
Ambient Temperature (°C)	25
Device Limit	5
Idle Time	100

Figure 4.1 NFC Settings

- "*NFC_CFG_POLL_TYPE_A*": Select to discover NFC Type-A tags
- "*NFC_CFG_POLL_TYPE_B*": Select to discover NFC Type-B tags
- "*NFC_CFG_POLL_TYPE_F*": Select to discover NFC Type-F tags
- "*NFC_CFG_POLL_TYPE_V*": Select to discover NFC Type-V tags

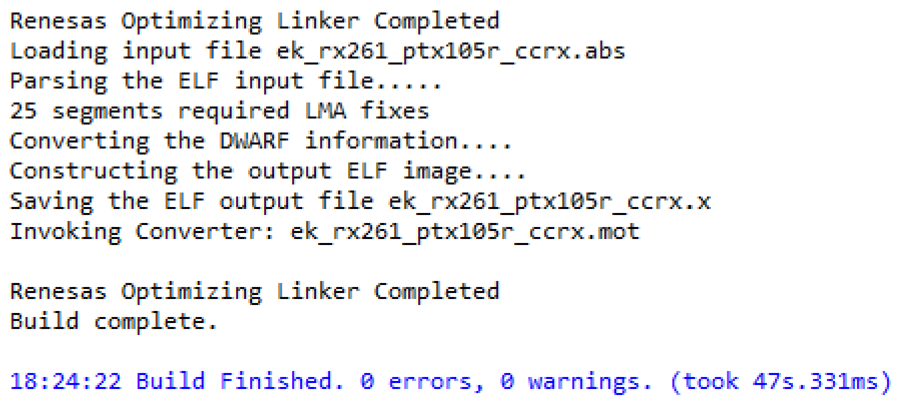
  (**Note**: Please select at least one polling type for proper card)
- "*NFC_CFG_TEMP_SENSOR_SHUTDOWN*": Expected thermal shutdown threshold value

  (unit: Celsius).
- "*NFC_CFG_TEMP_SENSOR_AMBIENT*": Set the ambient temperature value at which calibration takes place (unit: Celsius).
- "*NFC_CFG_DEVICE_LIMIT*": Select the max number of tags to discover (maximum number is 49).
- "*NFC_CFG_IDLE_TIME_MS*": Select idle time between polling cycles for RF-Discovery loop

  (unit: ms)

b) Building the Demo Project

Build the project and confirm no build errors occur.

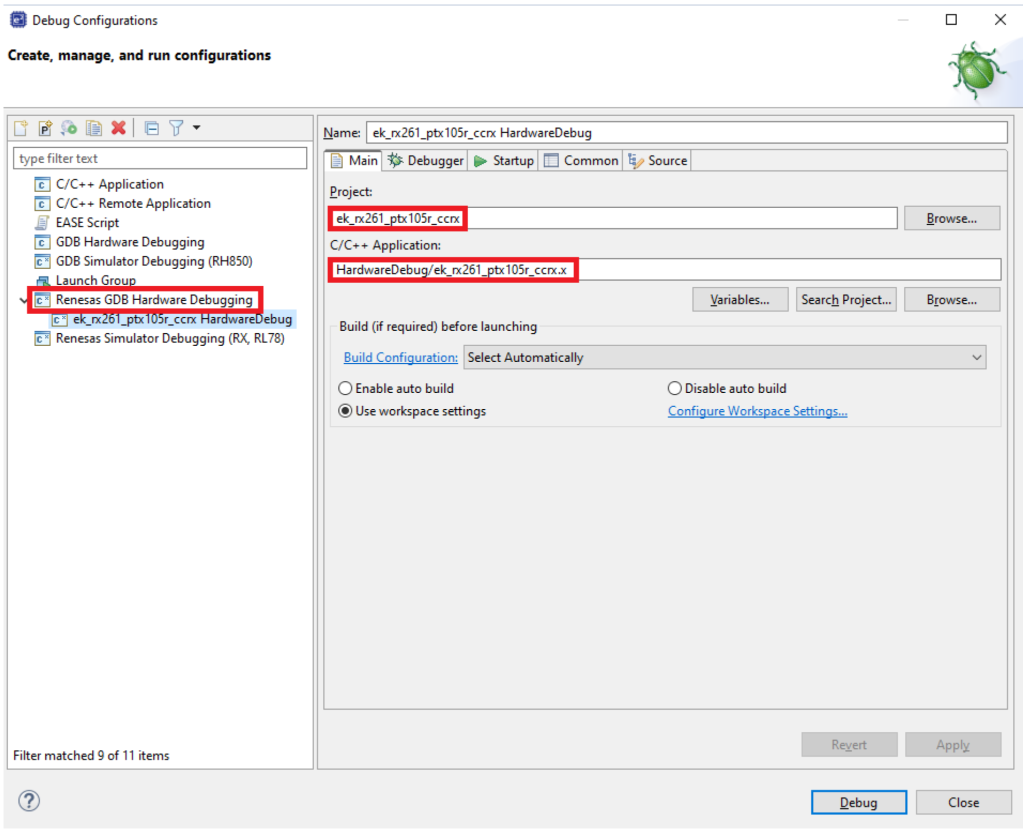


```
Renesas Optimizing Linker Completed
Loading input file ek_rx261_ptx105r_ccrx.abs
Parsing the ELF input file.....
25 segments required LMA fixes
Converting the DWARF information....
Constructing the output ELF image....
Saving the ELF output file ek_rx261_ptx105r_ccrx.x
Invoking Converter: ek_rx261_ptx105r_ccrx.mot

Renesas Optimizing Linker Completed
Build complete.

18:24:22 Build Finished. 0 errors, 0 warnings. (took 47s.331ms)
```

Figure 4.3 Confirm the Demo Project Build

c) Go to **Run ⟶ Debug Configurations…**, double-click on **Renesas GDB Hardware Debugging**, and set up the debugger.

d) Check whether exact **.x** file is selected (**.x** file is generated in HardwareDebug folder after build succeeded) same as figure below


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB Simulator Debugging (RH850)
- Launch Group
- Renesas GDB Hardware Debugging
  - ek_rx261_ptx105r_ccrx HardwareDebug
- Renesas Simulator Debugging (RX, RL78)

Filter matched 9 of 11 items

Name: ek_rx261_ptx105r_ccrx HardwareDebug

Main | Debugger | Startup | Common | Source

Project: ek_rx261_ptx105r_ccrx

C/C++ Application: HardwareDebug/ek_rx261_ptx105r_ccrx.x

Variables... | Search Project... | Browse...

Build (if required) before launching

Build Configuration: Select Automatically

( ) Enable auto build ( ) Disable auto build

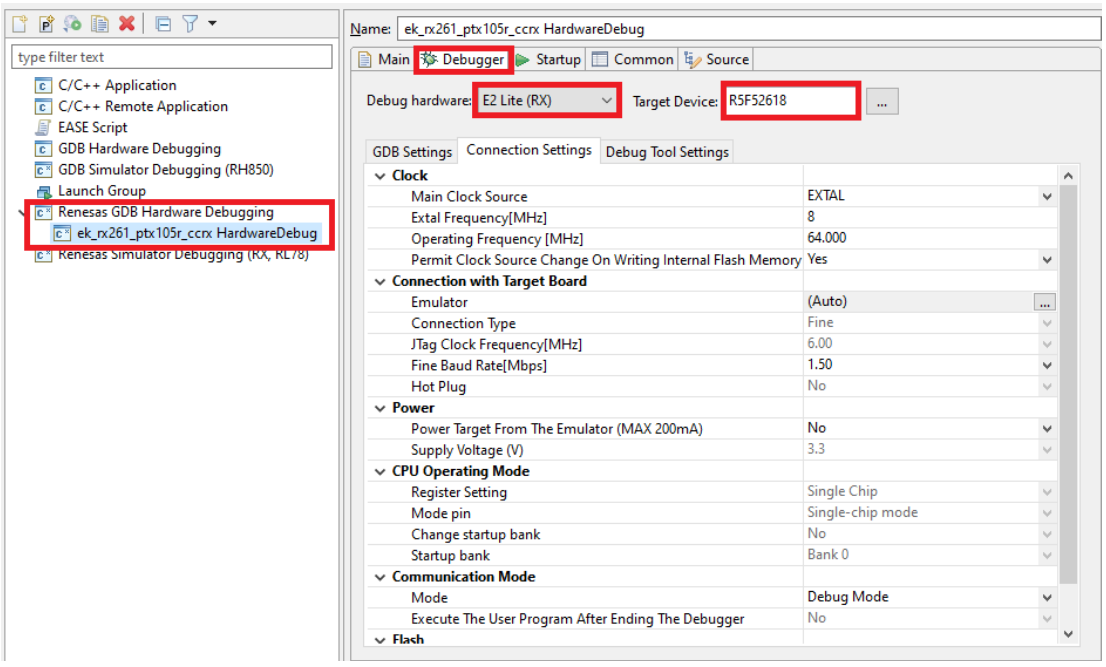
(●) Use workspace settings Configure Workspace Settings...

Revert | Apply

Debug | Close

Figure 4.4 Set Up the Debug File

e) Check “Debug hardware” and “Target Device”, ensure them are same as below figure by going to Renesas GDB Hardware Debugging->Debugger


type filter text

- C/C++ Application
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB Simulator Debugging (RH850)
- Launch Group
- Renesas GDB Hardware Debugging
  - ek_rx261_ptx105r_ccrx HardwareDebug
- Renesas Simulator Debugging (RX, RL78)

Name: ek_rx261_ptx105r_ccrx HardwareDebug

Main | Debugger | Startup | Common | Source

Debug hardware: E2 Lite (RX) Target Device: R5F52618

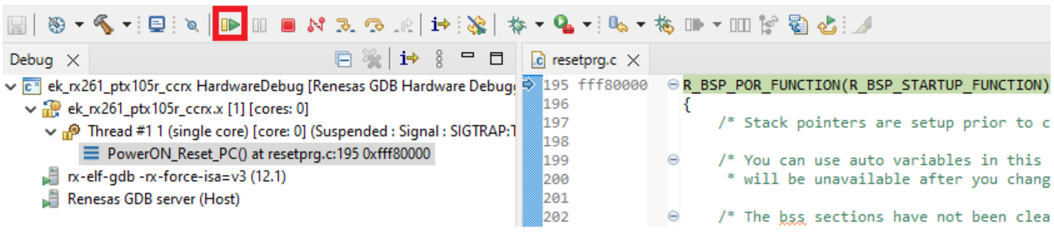
GDB Settings | Connection Settings | Debug Tool Settings

Clock	
Main Clock Source	EXTAL
Extal Frequency[MHz]	8
Operating Frequency [MHz]	64.000
Permit Clock Source Change On Writing Internal Flash Memory	Yes
Connection with Target Board	
Emulator	(Auto)
Connection Type	Fine
JTag Clock Frequency[MHz]	6.00
Fine Baud Rate[Mbps]	1.50
Hot Plug	No
Power	
Power Target From The Emulator (MAX 200mA)	No
Supply Voltage (V)	3.3
CPU Operating Mode	
Register Setting	Single Chip
Mode pin	Single-chip mode
Change startup bank	No
Startup bank	Bank 0
Communication Mode	
Mode	Debug Mode
Execute The User Program After Ending The Debugger	No
Flash	

Figure 4.5 Set Up the Debugger

f) Click **Debug** to flash the application on the board.

g) When completed, click **Resume** twice to start the application.



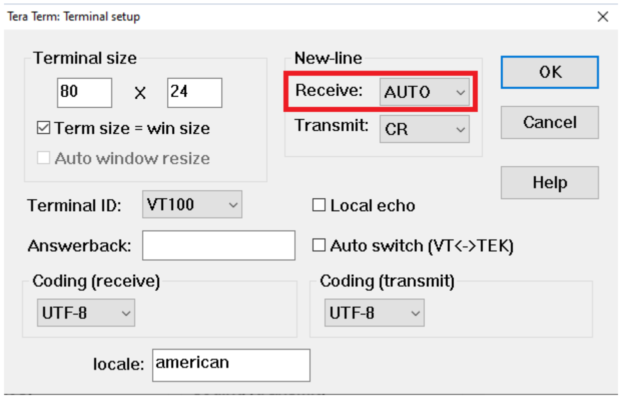
```
ek_rx261_ptx105r_ccrx HardwareDebug [Renesas GDB Hardware Debug
  ek_rx261_ptx105r_ccrx.x [1] [cores: 0]
    Thread #1 1 (single core) [core: 0] (Suspended : Signal : SIGTRAP:T
      PowerON_Reset_PC() at resetprg.c:195 0xfff80000
  rx-elf-gdb -rx-force-isa=v3 (12.1)
  Renesas GDB server (Host)
```

```
fff80000  R_BSP_POR_FUNCTION(R_BSP_STARTUP_FUNCTION)
          {
              /* Stack pointers are setup prior to c

              /* You can use auto variables in this
               * will be unavailable after you chang

              /* The bss sections have not been clea
```

Figure 4.6 Start the Application

To view the application output, open the Tera Term set up below settings

h) Click **Setup > Terminal**…, select "**New line: Receive**" as **AUTO**.

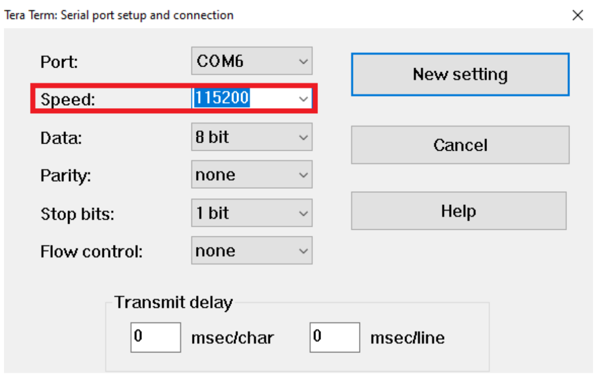


Tera Term: Terminal setup

Terminal size: 80 x 24
☑ Term size = win size
☐ Auto window resize
New-line
Receive: AUTO
Transmit: CR
Terminal ID: VT100
☐ Local echo
Answerback:
☐ Auto switch (VT<->TEK)
Coding (receive): UTF-8
Coding (transmit): UTF-8
locale: american

Figure 4.7 Terminal Setup

i) Click **Setup > Serial port**… and ensure that the speed is set to **115200**.



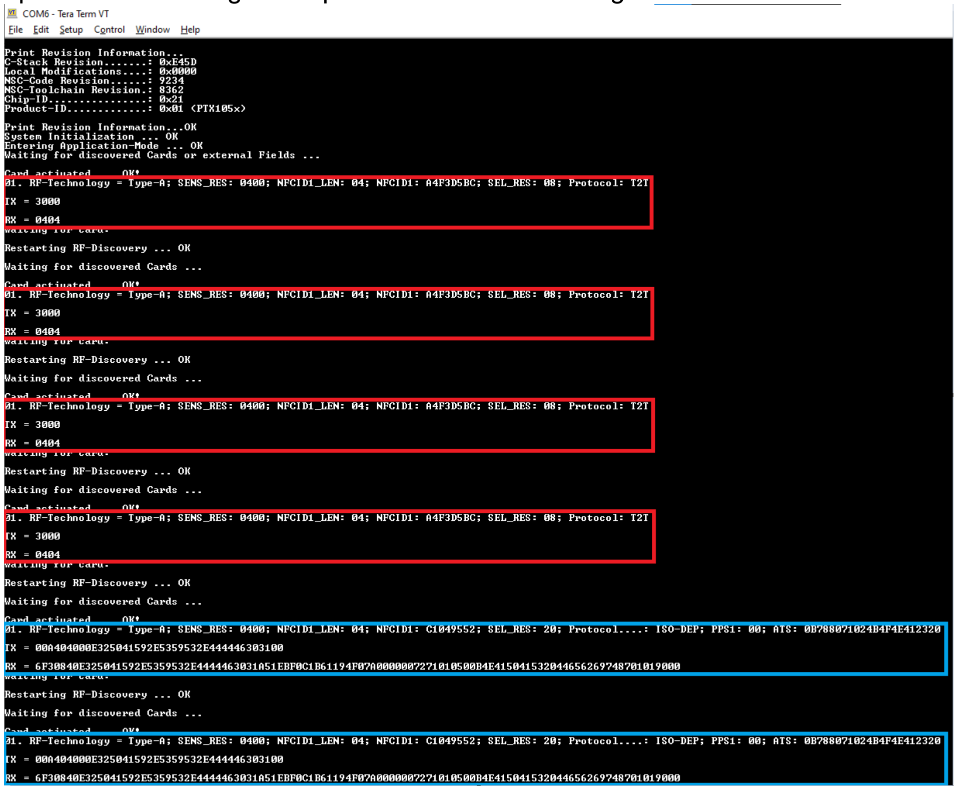
Tera Term: Serial port setup and connection

Port: COM6
Speed: 115200
Data: 8 bit
Parity: none
Stop bits: 1 bit
Flow control: none
Transmit delay
0 msec/char 0 msec/line

Figure 4.8 Terminal Setup for USB Serial Port

a) Once the serial terminal shows “**Waiting for discovered Cards or external Fields …**”, the user can place a card in the field and view the data displayed on the terminal.

Note:
1. This demo only supports 3 tag types: T2T, T3T and T4T.
2. Tera Term output data will depend on the type of NFC tag being used. As shown in the below figure, the data in the red box is printed when using a sample e-parking card and the other one in the blue box is printed when using a sample bank card as NFC tag devices.

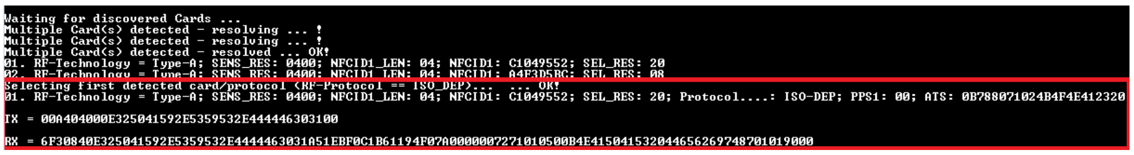


```
Print Revision Information...
C-Stack Revision.......: 0xE45D
Local Modifications....: 0x0000
NSC-Code Revision......: 9234
NSC-Toolchain Revision.: 8362
Chip-ID................: 0x21
Product-ID.............: 0x01 (PTX105x)

Print Revision Information...OK
System Initialization .... OK
Entering Application-Mode ... OK
Waiting for discovered Cards or external Fields ...

Card activated     OK!
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: A4F3D5BC; SEL_RES: 08; Protocol: T2T

TX = 3000

RX = 0404
Waiting for card.

Restarting RF-Discovery ... OK

Waiting for discovered Cards ...

Card activated     OK!
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: A4F3D5BC; SEL_RES: 08; Protocol: T2T

TX = 3000

RX = 0404
Waiting for card.

Restarting RF-Discovery ... OK

Waiting for discovered Cards ...

Card activated     OK!
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: A4F3D5BC; SEL_RES: 08; Protocol: T2T

TX = 3000

RX = 0404
Waiting for card.

Restarting RF-Discovery ... OK

Waiting for discovered Cards ...

Card activated     OK!
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: A4F3D5BC; SEL_RES: 08; Protocol: T2T

TX = 3000

RX = 0404
Waiting for card.

Restarting RF-Discovery ... OK

Waiting for discovered Cards ...

Card activated     OK!
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: C1049552; SEL_RES: 20; Protocol....: ISO-DEP; PPS1: 00; ATS: 0B788071024B4F4E412320

TX = 00A404000E325041592E5359532E444446303100

RX = 6F30840E325041592E5359532E4444463031A51EBF0C1B61194F07A0000007271010500B4E41504153204465626974870101 9000
Waiting for card.

Restarting RF-Discovery ... OK

Waiting for discovered Cards ...

Card activated     OK!
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: C1049552; SEL_RES: 20; Protocol....: ISO-DEP; PPS1: 00; ATS: 0B788071024B4F4E412320

TX = 00A404000E325041592E5359532E444446303100

RX = 6F30840E325041592E5359532E4444463031A51EBF0C1B61194F07A0000007271010500B4E41504153204465626974870101 9000
```

Figure 4.9 Example of Data Exchange

(9) The demo also supports selection from multiple cards by placing multiple cards over the PTX.

The program will print out generic information for each card and only perform data exchange for the card that was first detected.



```
Waiting for discovered Cards ...
Multiple Card(s) detected - resolving ... !
Multiple Card(s) detected - resolving ... !
Multiple Card(s) detected - resolved ... OK!
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: C1049552; SEL_RES: 20
02. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: A4F3D5BC; SEL_RES: 08
Selecting first detected card/protocol (RF-Protocol == ISO_DEP)...   ... OK!
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: C1049552; SEL_RES: 20; Protocol....: ISO-DEP; PPS1: 00; ATS: 0B788071024B4F4E412320

TX = 00A404000E325041592E5359532E444446303100

RX = 6F30840E325041592E5359532E4444463031A51EBF0C1B61194F07A0000007271010500B4E41504153204465626974870101 9000
```

Figure 4.10 Multiple Card Selection Handling

4.1.5 Understanding NFC Data Fields in the Example

In this section, the data exchange example shown in Figure 4.9 will be explained.

Sample NFC data

```
01. RF-Technology = Type-A; SENS_RES: 0400; NFCID1_LEN: 04; NFCID1: C1049552; SEL_RES: 20; Protocol....: ISO-DEP; PPS1: 00; ATS:
0B788071024B4F4E412320
TX = 00A404000E325041592E5359532E444446303100
RX = 6F30840E325041592E5359532E4444463031A51EBF0C1B61194F07A0000007271010500B4E41504153204465626974870101 9000
```

The above data exchange example is generated when a bank card using RF Technology Type A with the ISO-DEP protocol is placed in the field.

Polling/activation fields

Field	Value (Hex)	Description
RF Technology	Type-A	Indicates this is an ISO/IEC 14443 Type A card
SENS_RES (ATQA)	0x0400	ATQA (Answer to Request) to REQA (Request command of reader). Indicates card's UID (unique identifier) type is Single-length UID (4-byte UID)
NFCID1_LEN	0x04	Length of NFCID1 (UID) 0x04 means the UID is 4 bytes long.
NFCID1	0x C1049552	4-byte unique identifier (UID) of card
SEL_RES (SAK)	0x20	Select Response (also known as SAK – Select Acknowledge). This 1-byte response comes after the reader selects the card's UID. Confirms the card is now in an active state. A value of "0x20" means: ISO/IEC 14443-4 compliant card (supports ISO-DEP).
Protocol	ISO-DEP	Indicates the high-level protocol used is ISO-DEP (ISO Data Exchange Protocol).
PPS1	0x00	Protocol and Parameter Selection (PPS) byte. After the ATS (see below), the reader can send a PPS to adjust communication parameters (such as bitrate). 0x00 means no parameter changes were requested.
ATS	0x0B788071024B4F4E412320	The Answer to Select (ATS) from the card. The ATS provides the card's communication capabilities for the ISO-DEP layer (such as: ATS bytes length, maximum frame size, bitrate, timeout, manufacturer info, etc).

Table 4.11 General Card Parameters

APDU Command/Response (TX and RX) Explained

TX (Transmit)

00A404000E325041592E5359532E444446303100

This is an APDU command used to select an application (AID - Application Identifier) on a smart card.

Byte(s)	Meaning
00	CLA (Class Byte) – Standard ISO 7816 command
A4	INS (Instruction) – A4 means SELECT an application or file on the card
04	P1 (Parameter 1) – Select by name (AID)
00	P2 (Parameter 2) – First occurrence
0E	Lc (Length of AID) – 14 bytes
325041592E5359532E4444463031	Data (AID) – "2PAY.SYS.DDF01" (directory name of payment cards, often called PSE – Payment System Environment)
00	Indicates not specifying an expected length for the RX response

Table 4.12 Select Proximity Payment System Environment (PPSE)

RX (Response)

6F30840E325041592E5359532E4444463031A51EBF0C1B61194F07A0000007271010500B4E415041532044656269748701019000

Decoding the response message:

Byte(s)	Meaning
6F 30	FCI Template (TLV structure, 48 bytes)
84 0E 325041592E5359532E4444463031	Dedicated File (DF) Name – "2PAY.SYS.DDF01"
A5 1E	Proprietary Information
BF0C 1B	FCI Issuer Discretionary Data (27 bytes)
61 19	Application Template (25 bytes)
4F 07 A0000007271010	Application Identifier (AID) – A0000007271010
50 0B 4E415041532044656269 74	Application Label – "NAPAS Debit"
87 01 01	Application Priority Indicator (Priority = 1)
9000	Status Word – Success

Table 4.13 Select Proximity Payment System Environment (PPSE)

4.2. Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo project to a workspace, select *File* >> *Import* >> *General* >> *Existing Projects into Workspace*, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

4.3. Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the *Smart Brower* >> *Application Notes* tab.

5. Appendices

5.1 Confirmed Operation Environment

This section describes confirmed operation environment for the PTX NFC FIT module.

Table 5.5.1 Confirmed Operation Environment (Ver. 1.00)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2025.04
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Big endian / little endian
Revision of the module	Rev.1.00
Board used	Renesas EK-RX261 Evaluation Kit (Product no.: RTK5EK2610S00001BE)

5.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file "platform.h".

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"

- Using e^2 studio:

  Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note "Board Support Package Module Using Firmware Integration Technology (R01AN1685)".

(2) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file "r_nfc_ptx_rx_config.h" may be wrong. Check the file "r_nfc_ptx_rx_config.h". If there is a wrong setting, set the correct value for that. Refer to 2.7 Configuration Overview for details.

6. Reference Documents

User’s Manual: Hardware
The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User’s Manual: Development Tools
RX Family CC-RX Compiler User’s Manual (R20UT3248)
The latest versions can be downloaded from the Renesas Electronics website.

Website and Support
Renesas Electronics Website
http://www.renesas.com/
Inquiries
http://www.renesas.com/contact/

Revision History

Rev.	Date	Revision History	
		Page	Summary
1.00	Jun. 27, 2025	-	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.