

RL78 Family

PTX NFC Control Module Using Software Integration System

Introduction

This application note describes the usage of the PTX NFC control module, which conforms to the Software Integration System (SIS) standard.

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

The NFC module supports the following NFC Pmod module:

- PTX105R (PTX105RQC)

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

Target Device

- RL78/G23 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 RL78 Family

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

Related Documents

- [1] RL78 Family Board Support Package Module Using Software Integration System (R01AN5522)
- [2] RL78 Smart Configurator User's Guide: e² studio (R20AN0579)
- [3] Smart Configurator User's Guide: RL78 API Reference (R20UT4852)
- [4] RL78/G23 Serial Array Unit (UART Communication) (R01AN6645)
- [5] RL78/G23-128p Fast Prototyping Board – User's Manual (R20UT4870)

Contents

1. Overview	4
1.1 PTX NFC SIS Module.....	4
1.2 Overview of the PTX NFC SIS 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 SIS Module to Your Project.....	15
2.12. “for”, “while” and “do while” statements	15
3. API Functions	16
3.1. R_NFC_PTX_Open()	16
3.2. R_NFC_PTX_StartDiscovery().....	17
3.3. R_NFC_PTX_GetStatus().....	18
3.4. R_NFC_PTX_GetCardRegistry()	19
3.5. R_NFC_PTX_ActivateCard()	20
3.6. R_NFC_PTX_DataExchange()	21
3.7. R_NFC_PTX_ReaderDeactivation()	22
3.8. R_NFC_PTX_SWReset().....	23
3.9. R_NFC_PTX_Close().....	24
4. Demo Projects.....	25
4.1. NFC PTX105RQC Get Card Information Demo Project	25
4.1.1 Prerequisites.....	25
4.1.2 Import the Demo Project	25
4.1.3 Hardware Setup.....	25
4.1.4 How to Run the Demo	26
4.1.5 Understanding NFC Data Fields in the Example	31

4.2.	Adding a Demo to a Workspace.....	33
4.3.	Downloading Demo Projects	33
5.	Appendices.....	34
5.1	Confirmed Operation Environment.....	34
5.2	NFC Dependent Module Changes	34
5.2.1	Timer	34
5.2.2	Interrupt	36
5.2.3	Communication.....	37
6.	Reference Documents	40
	Revision History	41

1. Overview

1.1 PTX NFC SIS Module

The PTX NFC SIS module can be used by being implemented in a project as an API. See section 2.11 Adding the SIS Module to Your Project for details on methods to implement this SIS module into a project.

1.2 Overview of the PTX NFC SIS 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 SIS 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 SIS 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.

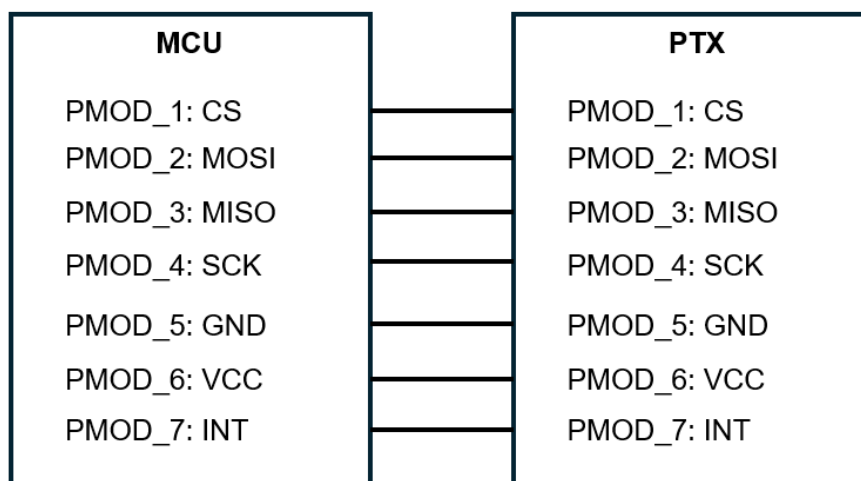


Figure 1.1 Example Connection to the PTX NFC Module

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
RL78/G23-128p Fast Prototyping Board	FPB for RL78/G23 Part number: RTK7RLG230CSN000BJ	Please see detail at: RTK7RLG230CSN000BJ - RL78/G23-128p Fast Prototyping Board (RL78/G23-128p FPB) Renesas

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.

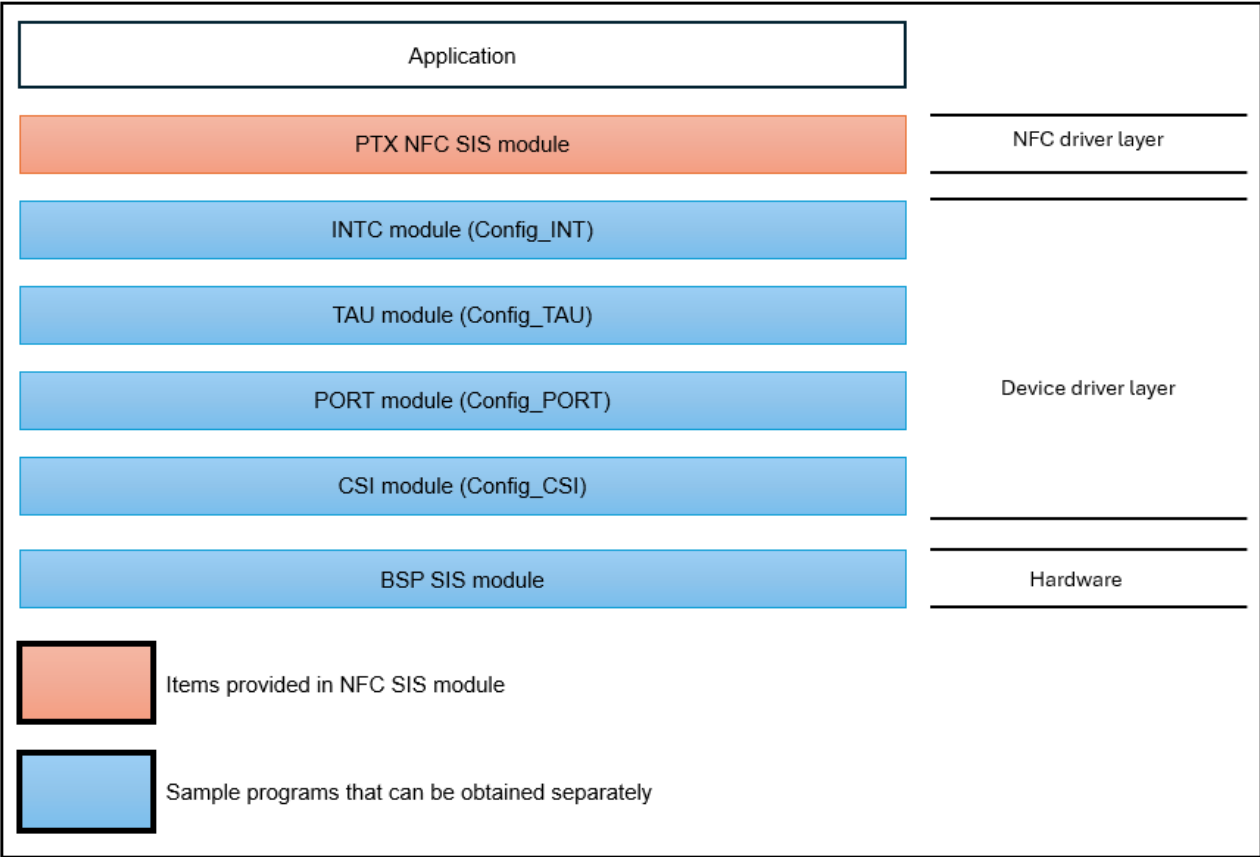


Figure 1.2 Software Configuration Diagram

1. PTX NFC SIS module
The SIS module. This software is used to control the NFC module.
2. BSP SIS 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 SIS 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 SIS module.

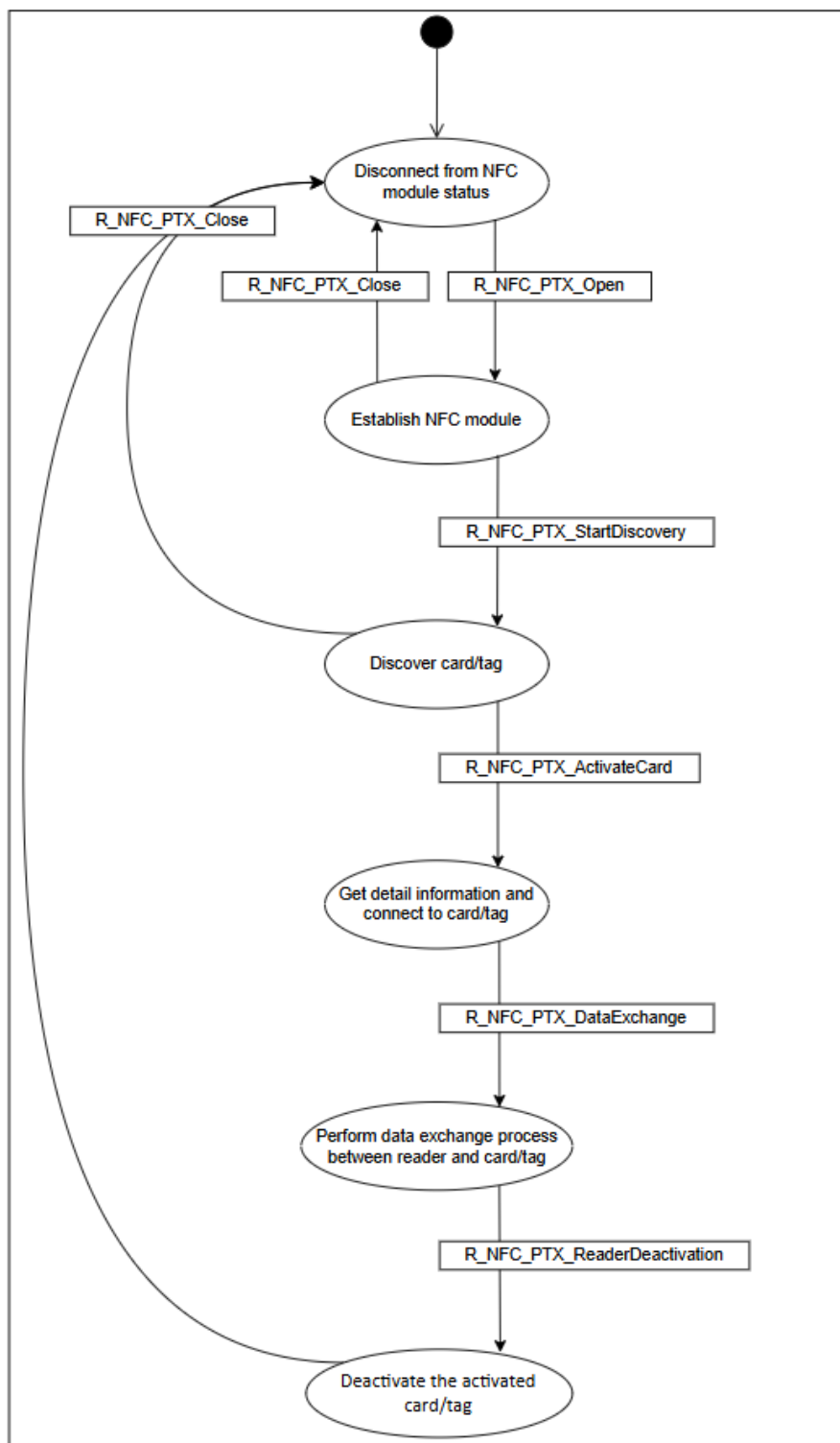


Figure 1.3 Status Transitions

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

2. API Information

This SIS 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 SIS modules:

- Board support package (r_bsp)
- INTC module (Config_INTC)
- TAU module (Config_TAU)
- PORT module (Config_PORT)
- CSI module (Config_CSI)

For using these dependencies with PTX NFC SIS module properly, please refer detail in 5.2 NFC Dependent Module Changes

2.3. Supported Toolchain

The SIS 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 SIS module are contained in `r_nfc_ptx_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_config.h`)

Configuration Options in <code>r_nfc_ptx_config.h</code>	
NFC_CFG_COMM_MODE Default: "0"	Configure host communication mode. 0 = SPI.
NFC_CFG_CSI_UNIT Default: "1"	Configure SPI unit (for SPI interface only) Set this option to match the SCI port to be controlled.
NFC_CFG_CSI_CHAN Default: "2"	Configure SPI channel (for SPI interface only) Set this option to match the SCI port to be controlled. IRQ number that connected to IRQ pin of PTX Set this option to match the IRQ number to be controlled.
NFC_CFG_CS_PORT Default: "14"	Configure the general port that controls SS port (for SPI interface only). Set this option to match the port to be controlled.
NFC_CFG_CS_PIN Default: "1"	Configure the general pin that controls SS pin (for SPI interface only). Set this option to match the pin to be controlled.
NFC_CFG_USER_SEND_RECV_FUNCTION_NAME Default: "UserCode_R_Config_CSI30_Send_Receive"	Specifies function name of the user send receive function for serial transaction. This option must not be NULL. Please refer detail 5.2 NFC Dependent Module Changes for how to implement this user function.
NFC_CFG_INT_NUM Default: "6"	IRQ number that connected to IRQ pin of PTX Set this option to match the IRQ number to be controlled.
NFC_CFG_EXT_INT_PORT Default: "14"	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_INT_PIN Default: "0"	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_TIMER_UNIT Default: "0"	Configure PTX module's sleep timer unit. Set this option to match the timer to be controlled
NFC_CFG_TIMER_CHAN Default: "1"	Configure PTX module's sleep timer channel. Set this option to match the timer to be controlled
NFC_CFG_USER_TMR_GET_FREQ_FUNCTION_NAME Default: "UserCode_R_Config_TAU0_1_GetTimerFrequency"	Specifies function name of the user get timer frequency function. This option must not be NULL. Please refer detail 5.2 NFC Dependent Module Changes for how to implement this user function.
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

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 rev1.00.

Compiler Version: Renesas Electronics C Compiler Package for RL78 Family V1.15.00

Configuration Options: Default settings.

Table 2.2 Memory Sizes for RL78/G23

Device	Memory Type	Category	Memory usage
			Renesas Compiler (CC-RL)
RL78/G23	With platform modules	ROM	72551 bytes
		RAM	2476 bytes
	Without platform modules	ROM	71612 bytes
		RAM	2469 bytes

(**Note:** platform modules are low-level modules: INTC, TAU, PORT, CSI).

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 2.3 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 2.4 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 2.5 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 2.6 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 SIS Module to Your Project

The SIS module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in below:

Adding the SIS module to your project using the Smart Configurator in e2 studio. By using the Smart Configurator in e2 studio, the SIS module is automatically added to your project. Refer to “RL78 Smart Configurator User’s Guide: e² studio (R20AN0579)” for details.

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

In SIS 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 SIS 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 SIS module and its dependent modules (e.g. r_bsp). This SIS module includes the following demo projects.

4.1. NFC PTX105RQC Get Card Information Demo Project

4.1.1 Prerequisites

- Hardware requirements:
 - RL78/G23-128p: RL78/G23-128p Fast Prototyping Board (RTK7RLG230CSN000BJ).
 - PTX105R: PTX105R PMOD™ Board for IoT as NFC module
 - PC running Windows 10.
 - Micro-USB cable for Power supply (included as part of the kit. See RL78/G23-128p Fast Prototyping Board – 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 Compiler for RL78 Family V1.15.00.
 - 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 e² 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 RL78/G23-128p FPB **PMOD1** connector.
- Connect a micro-USB cable from the PC to the RL78/G23-128p FPB micro-USB connector (**J12**) for power supply.
- Connect the USB to TTL cable from PC to the RL78/G23-128p FPB micro-USB connector (**UART2**) for serial log output.
 - Establish the UART2 connection on the RL78/G23-128p FPB board using the following pin mapping as table below:

USB-to-TTL	RL78/G23-128p FPB
TxD	Pin 19
RxD	Pin 18
GND	GND

Table 4.1 Pin mapping of UART2 connection with USB to TTL

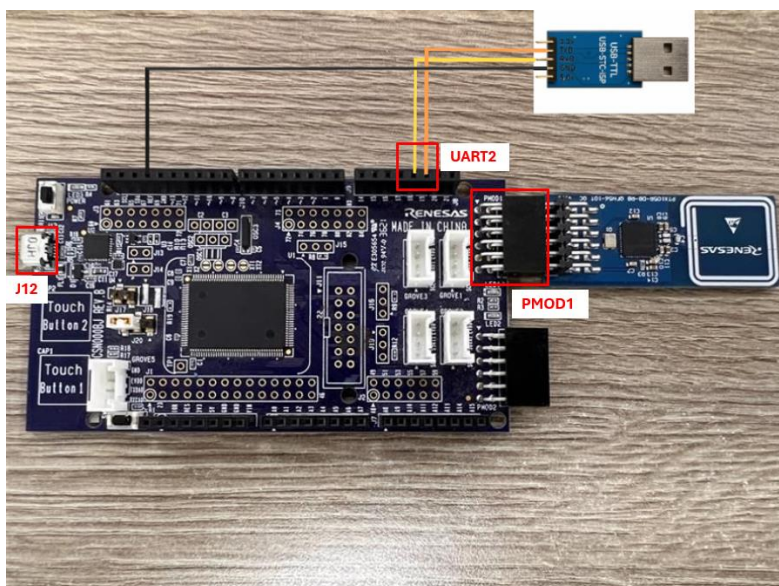


Figure 4.1 Hardware Setup

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.

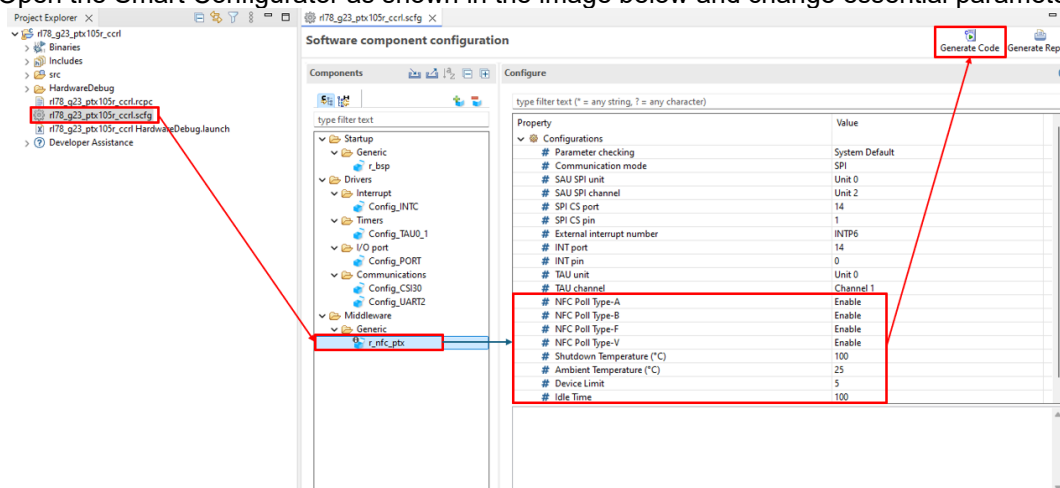


Figure 4.1 NFC Settings

- “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 rl78_g23_ptx105r_ccrl.abs
Parsing the ELF input file.....
18 segments required LMA fixes
Converting the DWARF information....
Constructing the output ELF image....
Saving the ELF output file rl78_g23_ptx105r_ccrl.x

Invoking Converter: rl78_g23_ptx105r_ccrl.mot
Renesas Optimizing Linker Completed
Build complete.

09:12:28 Build Finished. 0 errors, 0 warnings. (took 47s.211ms)
```

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

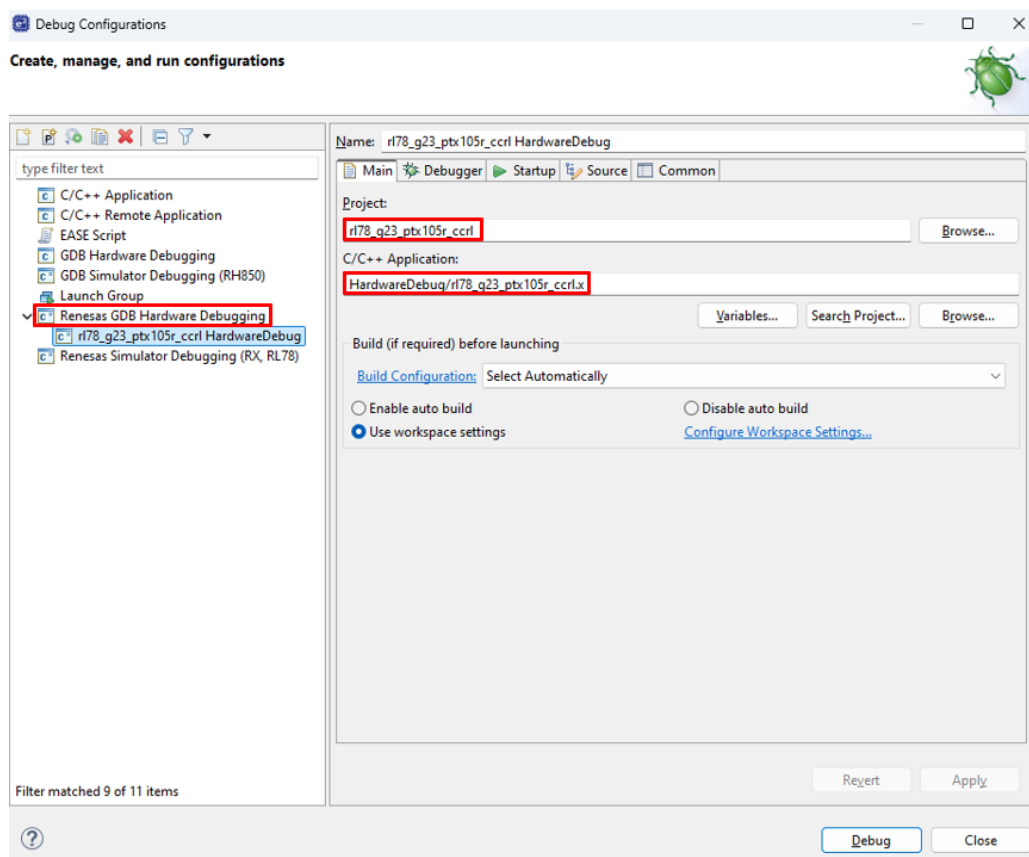


Figure 4.4 Set Up the Debug File

- e) Check “Debug hardware” and “Target Device”, ensure them are same as below figure by going to Renesas GDB Hardware Debugging->Debugger (please ensure to connect to correct debug port of RL78/G23 FPB)

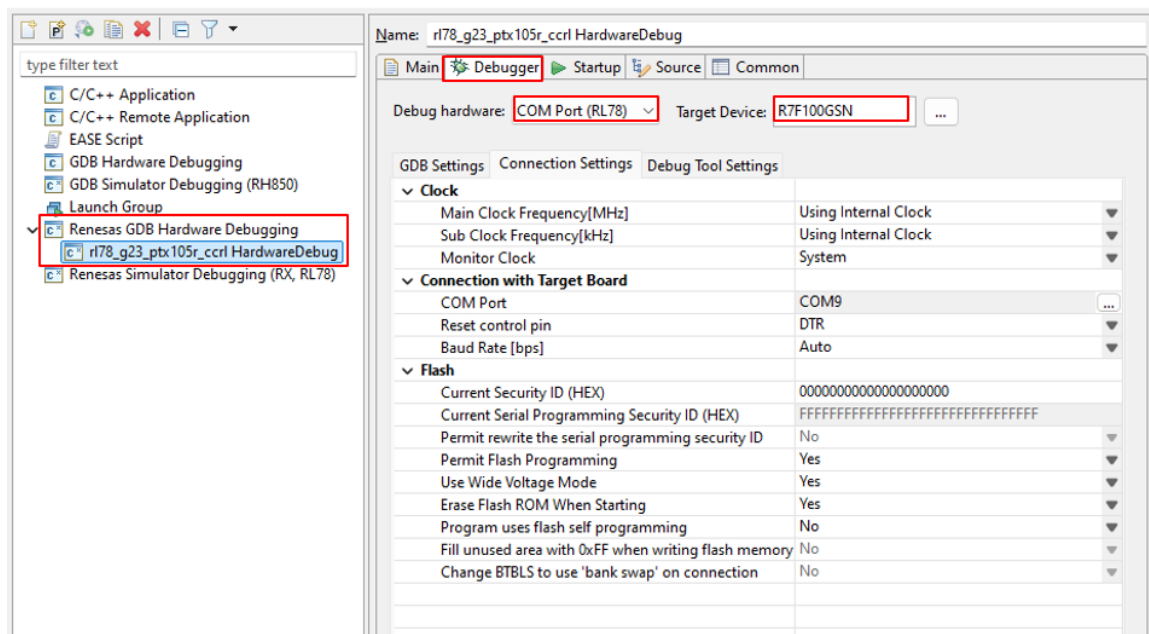
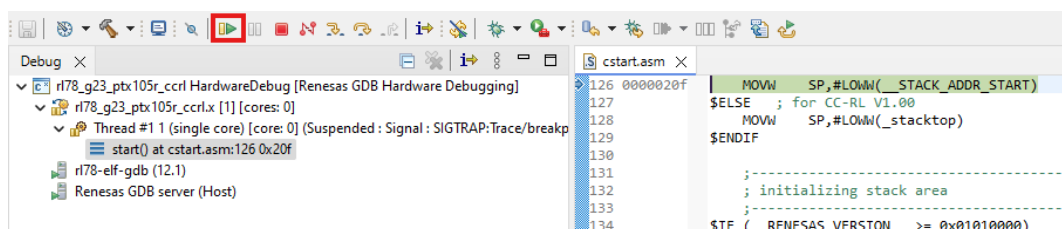


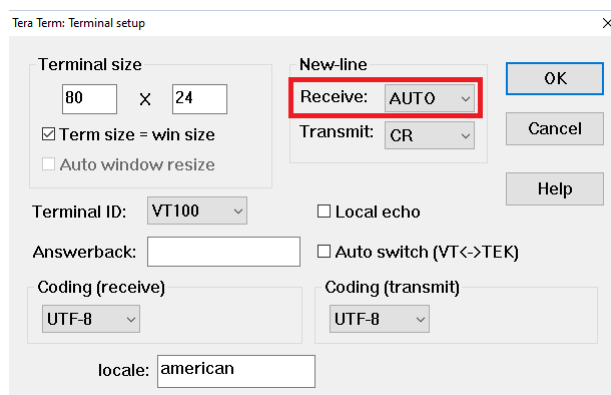
Figure 4.5 Set Up the Debugger

- f) Click **Debug** to flash the application on the board.
- g) When completed, click **Resume** twice to start the application.

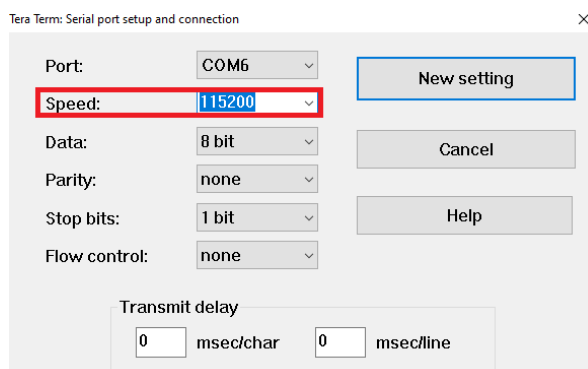
**Figure 4.6 Start the Application**

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

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

**Figure 4.7 Terminal Setup**

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

**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.

```

COM6 - Tera Term VT
File Edit Setup Control Window Help

Print Revision Information...
C-Stack Revision.....: 0x345D
Local Modifications....: 0x0000
NSC-Code Revision.....: 2224
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 = 6F30840E325041592E5359532E4444463031051EBF0C1B61194F07A0000007271010500B4E415041532044656269748701019000
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 = 6F30840E325041592E5359532E4444463031051EBF0C1B61194F07A0000007271010500B4E415041532044656269748701019000

```

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 = 6F30840E325041592E5359532E4444463031051EBF0C1B61194F07A0000007271010500B4E415041532044656269748701019000

```

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 = 6F30840E325041592E5359532E4444463031A51EBF0C1B61194F07A0000007271010500B4E415041532044656269748701019000
```

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 4E41504153204465626974	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 sample_code 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 sample_code subdirectory, select the desired demo zip file, then click "Finish".

4.3. Downloading Demo Projects

When using the demo project, the SIS module needs to be downloaded. To download the SIS module, right click on this application note and select "Sample Code (download)" from the context menu in the Smart Browser >> Application Notes tab.

5. Appendices

5.1 Confirmed Operation Environment

This section describes confirmed operation environment for the PTX NFC SIS 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 Compiler for RL78 Family V1.15.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Little endian
Revision of the module	Rev.1.00
Board used	RL78/G23-128p Fast Prototyping Board (RTK7RLG230CSN000BJ)

5.2 NFC Dependent Module Changes

The NFC module implements the PTX IoT Reader SDK, which includes platform files for communicating with low-level peripherals (Communications, Timers, and Interrupts). This section describes required changes to these dependent peripherals (SPI and Timer), as outlined in Section 2.2 Software Requirements, to work with the NFC module. The NFC module provides several properties for user-defined functions to work with these dependent peripherals. In this section, the additional steps needed for correct operation are explained.

Note: The following configurations and procedures are specific to the provided demo project (section 4.1.3 Hardware Setup). The user is responsible for adapting or implementing the required APIs if using new configurations or new devices.

5.2.1 Timer

The NFC module's Timer platform files require timer frequency input for calculating timer interval duration. Users are therefore required to define this user function in the NFC SIS module properties.

The procedure used for the provided demo project is explained in the following sections (sample resource: **TAU0_1**):

- **Smart Configuration**

User function name is defined by configuring NFC_CFG_USER_TMR_GET_FREQ_FUNCTION_NAME via Smart Configurator -> Generate Code

# TAU unit	Unit 0
# TAU channel	Channel 1
# NFC get timer frequency function name	UserCode_R_Config_TAU0_1_GetTimerFrequency

Figure 5.1 Get timer frequency function name configuration

- **Config_TAU0_1_user.c**

Define user function `NFC_Timer_Frequency_User_Function()` to compute the timer configured frequency

```

/*****
 * Function Name: NFC_Timer_Frequency_User_Function
 * Description  : Get the timer frequency.
 * Arguments   : None
 * Return Value : Timer frequency in Hz
 *****/
uint32_t NFC_Timer_Frequency_User_Function(void)
{
    /* Get the MSB 3 bits of 16 bits register TMR01 */
    /* 0 = CKSmn0, 1 = CKSmn1, 2 = CKSmn2, 3 = CKSmn3 */
    uint8_t ClockSourceSetting = (uint8_t)((uint16_t)(TMR01 & _C000_TAU_CLOCK_SELECT_CKM3) >> CLOCK_SOURCE_OFFSET);
    uint16_t ClockFrequencySetting = (uint16_t)TPS0;
    uint32_t timerFrequency = 0u;

    switch(ClockSourceSetting)
    {
        case CLOCK_SOURCE_CKM0:
        {
            ClockFrequencySetting &= CLOCK_000F_TAU_CKM0_CLEAR;
            ClockFrequencySetting = ClockFrequencySetting >> CLOCK_SOURCE_CKM0_OFFSET;
            timerFrequency = fCLK/(1uL << (uint8_t)ClockFrequencySetting); /* ckm0 - fCLK/2^ClockFrequencySetting */
            break;
        }
        case CLOCK_SOURCE_CKM1:
        {
            ClockFrequencySetting &= CLOCK_00F0_TAU_CKM1_CLEAR;
            ClockFrequencySetting = ClockFrequencySetting >> CLOCK_SOURCE_CKM1_OFFSET;
            timerFrequency = fCLK/(1uL << (uint8_t)ClockFrequencySetting); /* ckm1 - fCLK/2^ClockFrequencySetting */
            break;
        }
        case CLOCK_SOURCE_CKM2:
        {
            ClockFrequencySetting &= CLOCK_0300_TAU_CKM2_CLEAR;
            ClockFrequencySetting = ClockFrequencySetting >> CLOCK_SOURCE_CKM2_OFFSET;

            if( 0u != ClockFrequencySetting)
            {
                timerFrequency = fCLK/(1uL << (uint8_t)(ClockFrequencySetting * 2u)); /* ckm2 - fCLK/2^(ClockFrequencySetting*2) */
            }
            else
            {
                timerFrequency = fCLK/2uL; /* ckm2 - fCLK/2^1 */
            }
            break;
        }
        case CLOCK_SOURCE_CKM3:
        {
            ClockFrequencySetting &= CLOCK_3000_TAU_CKM3_CLEAR;
            ClockFrequencySetting = ClockFrequencySetting >> CLOCK_SOURCE_CKM3_OFFSET;
            timerFrequency = fCLK/(1uL << (uint8_t)((ClockFrequencySetting * 2) + 8u)); /* ckm3 - fCLK/2^ClockFrequencySetting */
            break;
        }
        default: /* timerFrequency already 0uL, which is invalid */
        {
            break;
        }
    }

    return timerFrequency;
}

```

Figure 5.2 Get timer frequency function definition

In this file, the timer interrupt is also handled. The **timeoutFlag** variable must be declared in this file for use in the Platform APIs of the NFC SDK.

```

/*****
Global variables and functions
*****/
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t timeoutFlag = 0u;
/* End user code. Do not edit comment generated here */

```

Figure 5.3 Timer flag declaration

The flag **timeoutFlag** is set when ISR is invoked. Ensure that **timeoutFlag** is correctly handled as below.

```

/*****
* Function Name: r_Config_TAU0_1_interrupt
* Description : This function is INTTM01 interrupt service routine.
* Arguments : None
* Return Value : None
*****/
static void __near r_Config_TAU0_1_interrupt(void)
{
    /* Start user code for r_Config_TAU0_1_interrupt. Do not edit comment generated here */
    timeoutFlag = 1;
    /* End user code. Do not edit comment generated here */
}

```

Figure 5.4 Timer ISR definition

5.2.2 Interrupt

The NFC module needs an external interrupt to trigger the Host controller (RL78/G23) from NFC chip (PTX) via using interrupt flag **int_flag**.

The demo project procedure is explained in the following sections (sample resource: **INTC_INTP6**):

- **Config_INTC_user.c**

In this file, the ISR is handled. The **int_flag** variable must be declared in this file to use in the Platform APIs of the NFC SDK

```

/*****
Global variables and functions
*****/
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t int_flag = 0u; /* INTP6 interrupt Flag */
/* End user code. Do not edit comment generated here */

```

Figure 5.5 Interrupt flag declaration

The flag **int_flag** is set when ISR is invoked. Ensure that **int_flag** is correctly handled as below

```

/*****
* Function Name: r_Config_INTC_intp6_interrupt
* Description : This function is INTP6 interrupt service routine
* Arguments : None
* Return Value : None
*****/
static void __near r_Config_INTC_intp6_interrupt(void)
{
    /* Start user code for r_Config_INTC_intp6_interrupt. Do not edit comment generated here */
    int_flag = 1u;
    /* End user code. Do not edit comment generated here */
}

```

Figure 5.6 Interrupt ISR definition

5.2.3 Communication

The serial communication platform files of the NFC SDK are used to transfer firmware code to the PTX chip and to exchange runtime commands. Since the firmware code is saved in the **.constf** memory section, some adaptation needs to be implemented for generated files.

The example procedure of demo project is explained in the following sections (sample resource: **CSI30**):

- **Smart Configuration**

User function name is defined by configuring **NFC_CFG_USER_SEND_RECV_FUNCTION_NAME** via Smart Configurator -> Generate Code

# Communication mode	SPI
# SAU SPI unit	Unit 1
# SAU SPI channel	Unit 2
# SPI CS port	14
# SPI CS pin	1
# NFC user-defined send_receive function name	UserCode_R_Config_CSI30_Send_Receive

Figure 5.7 Serial Communication send-receive function name configuration

- **Config_CSI30.c**

Define user function *NFC_Send_Receive_User_Function()* with modified TX buffer data type.

Because of changing data type of **tx_buf** input argument, need to declare new address pointer **gp_csi30_tx_address_usr** with **const** type accordingly (compared with original *R_Config_CSI30_Send_Receive()* generated by Smart Configurator).

```
/* Start user code for global. Do not edit comment generated here */
const volatile uint8_t * gp_csi30_tx_address_usr = NULL;
/* End user code. Do not edit comment generated here */
```

Figure 5.8 User -defined TX address pointer

```
MD_STATUS NFC_Send_Receive_User_Function(const uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf)
{
    MD_STATUS status = MD_OK;

    if (tx_num < 1U)
    {
        status = MD_ARGERROR;
    }
    else
    {
        g_csi30_send_length = tx_num; /* send data length */
        g_csi30_tx_count = tx_num; /* send data count */
        gp_csi30_tx_address_usr = tx_buf; /* send buffer pointer */
        gp_csi30_rx_address = rx_buf; /* receive buffer pointer */

        if (1U == tx_num)
        {
            SMR12 &= (uint16_t)~_0001_SAU_BUFFER_EMPTY;
        }
        else
        {
            SMR12 |= _0001_SAU_BUFFER_EMPTY;
        }
        CSIMK30 = 1U; /* disable INTCSI30 interrupt */

        if (0U != gp_csi30_tx_address_usr)
        {
            SIO30 = *gp_csi30_tx_address_usr; /* started by writing data to SDR12[7:0] */
            gp_csi30_tx_address_usr++;
        }
        else
        {
            SIO30 = 0xFFU;
        }

        g_csi30_tx_count--;
        CSIMK30 = 0U; /* enable INTCSI30 interrupt */
    }

    return (status);
}
```

Figure 5.9 Send-Receive function definition

Reason why TX buffer must be a constant pointer to a constant:

Firmware code to be sent to PTX chip is included in the NFC SDK saved in the **.constf** memory section. Since the TX buffer must point to the **.constf** location of the firmware code, the “const” modifier is required at the head of the data type. If not, the CC-RL compiler will automatically allocate an address into NEAR RAM and will no longer point to the **.constf**.

- Config_CSI30_user.c**

In this file, the SPI interrupts are handled. **txEndTransfer**, **rxEndTransfer** and **transferError** flags variable must be declared in this file to use in Platform APIs of the NFC SDK.

```
volatile uint8_t txEndTransfer = 0u;      /* Tx End Transfer Flag */
volatile uint8_t transferError = 0u;     /* Transfer Error Flag */
volatile uint8_t rxEndTransfer = 0u;     /* RxEndTransfer flag */
```

Figure 5.10 Communication flags declaration

In the event of regenerating, ensure that interrupt flags **txEndTransfer**, **transferError**, and **rxEndTransfer** are well handled as below.

```

/*****
* Function Name: r_Config_CSI30_callback_sendend
* Description  : This function is a callback function when CSI30 finishes transmission.
* Arguments    : None
* Return Value : None
*****/
static void r_Config_CSI30_callback_sendend(void)
{
    /* Start user code for r_Config_CSI30_callback_sendend. Do not edit comment generated here */
    SMR12 |= _0001_SAU_BUFFER_EMPTY;
    txEndTransfer = 1u;
    /* End user code. Do not edit comment generated here */
}

/*****
* Function Name: r_Config_CSI30_callback_receiveend
* Description  : This function is a callback function when CSI30 finishes reception.
* Arguments    : None
* Return Value : None
*****/
static void r_Config_CSI30_callback_receiveend(void)
{
    /* Start user code for r_Config_CSI30_callback_receiveend. Do not edit comment generated here */
    SMR12 |= _0001_SAU_BUFFER_EMPTY;
    rxEndTransfer = 1u;
    /* End user code. Do not edit comment generated here */
}

/*****
* Function Name: r_Config_CSI30_callback_error
* Description  : This function is a callback function when CSI30 reception error occurs.
* Arguments    : err_type -
                  error type value
* Return Value : None
*****/
static void r_Config_CSI30_callback_error(uint8_t err_type)
{
    /* Start user code for r_Config_CSI30_callback_error. Do not edit comment generated here */
    transferError = 1u;
    /* End user code. Do not edit comment generated here */
}

```

Figure 5.11 Communication callback functions definition

Along with addition of **gp_csi30_tx_address_usr** in **Config_CSI30.c**, this variable needs to be set as an extern variable in this file for the ISR to ensure correct implementation of firmware code transaction.

```
/* Start user code for global. Do not edit comment generated here */
extern const volatile uint8_t * gp_csi30_tx_address_usr;
```

Figure 5.12 User -defined TX address pointer extern usage

The ISR needs to be re-defined with a new **gp_csi30_tx_address_usr**, and implement **r_Config_CSI30_callback_sendend()** additionally in case the transfer end interrupt is triggered (compared with original **r_Config_CSI30_interrupt()** generated by Smart Configurator)

```
static void __near r_CSI30_isr(void)
{
    uint8_t err_type;

    err_type = (uint8_t)(SSR12 & _0001_SAU_OVERRUN_ERROR);
    SIR12 = (uint16_t)err_type;

    if (1U == err_type)
    {
        r_Config_CSI30_callback_error(err_type);    /* overrun error occurs */
    }
    else
    {
        if (g_csi30_tx_count > 0U)
        {
            if ((g_csi30_send_length - 1U) != g_csi30_tx_count)
            {
                *gp_csi30_rx_address = SIO30;
                gp_csi30_rx_address++;
            }

            SIO30 = *gp_csi30_tx_address_usr;
            gp_csi30_tx_address_usr++;
            g_csi30_tx_count--;
        }
        else
        {
            if (1U == (SMR12 & _0001_SAU_BUFFER_EMPTY))
            {
                r_Config_CSI30_callback_sendend();    /* complete send */
                *gp_csi30_rx_address = SIO30;
                gp_csi30_rx_address++;

                if (0U == (SSR12 & _0040_SAU_UNDER_EXECUTE))
                {
                    *gp_csi30_rx_address = SIO30;
                    r_Config_CSI30_callback_receiveend();    /* complete receive */
                }
                else
                {
                    SMR12 &= (uint16_t)~_0001_SAU_BUFFER_EMPTY;
                }
            }
            else
            {
                *gp_csi30_rx_address = SIO30;
                r_Config_CSI30_callback_sendend();    /* complete send */
                r_Config_CSI30_callback_receiveend();    /* complete receive */
            }
        }
    }
}
```

Figure 5.13 User-defined ISR for serial communication definition

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

RL78 Family's C Compiler CC-RL User's Manual (R20UT3123)

(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	Aug. 28, 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.

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 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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/