

RL78 Family

PMBus Slave Module Software Integration System

Introduction

This application note describes the PMBus Slave module.

Target Device

RL78/G24

Related Documents

- RL78/G24 User's Manual: Hardware (R01UH0961J)
- PMBus Specification Rev. 1.4 Part I
- PMBus Specification Rev. 1.4 Part II
- System Management Bus (SMBus) Specification Version 3.2

Contents

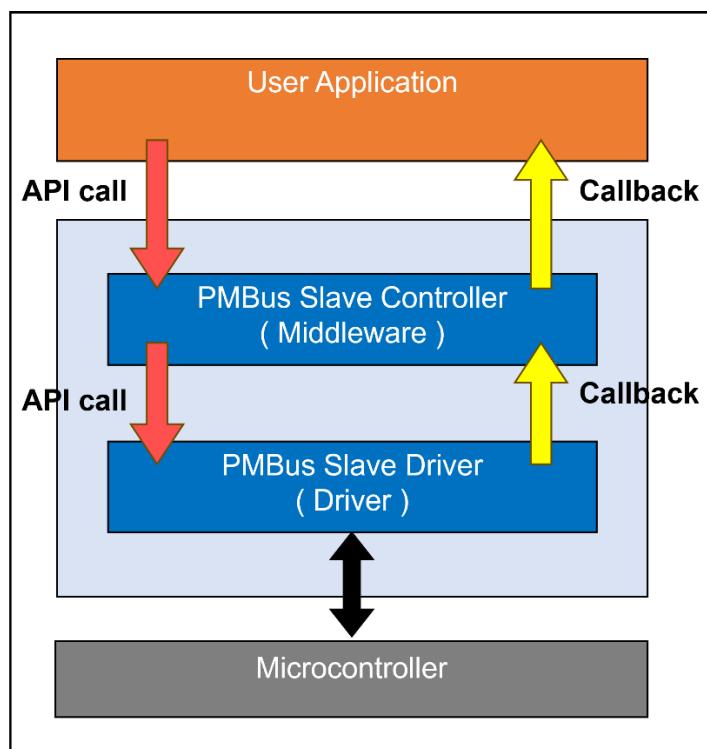
1. Overview	4
1.1 PMBus Slave Driver (PMBSDRV) Features.....	4
1.1.1 Bus Communication Features	4
1.1.2 Communication Error Detection	6
1.1.3 Optional Pin Features.....	6
1.2 PMBus Slave Controller (PMBSCTL) Features	7
1.2.1 Communication Formats	7
1.2.2 User Notification of Received Commands and Data.....	10
1.2.3 Fault Detection and User Notification.....	10
1.2.4 Reporting to the Controller	11
1.2.5 H/W Signal Information	11
2. API Information.....	12
2.1 Hardware Requirements	12
2.2 Software Requirements	12
2.3 Supported Tool Chains.....	13
2.4 Header files	13
2.5 Integer Type	13
2.6 Code Size	13
3. Configuration Specifications	14
4. API Specification	16
4.1 API Typedef Definitions (PMBus Slave Controller)	16
4.1.1 pmbectl_cbk_ret_t	16
4.1.2 pmbectl_fault_t	17
4.1.3 pmbectl_polarity_t	18
4.1.1 pmbectl_data_t	18
4.1.1 pmbectl_callback_t	19
4.1.2 pmbectl_hw_signal_t	20
4.2 API Function Specifications (PMBus Slave Controller).....	21
4.2.1 RM_PMBSCTL_Open	21
4.2.2 RM_PMBSCTL_Close.....	22
4.2.3 RM_PMBSCTL_GetHWSignal	23
4.2.4 RM_PMBSCTL_SetPolarityControlPin	24
4.2.5 RM_PMBSCTL_SetSMBAlert	25
4.2.6 RM_PMBSCTL_SetHostNotify.....	26
4.3 API Typedef Definitions (PMBus Slave Driver)	27
4.4 API Function Specifications (PMBus Slave Driver).....	28
5. Operation Sequence.....	30
5.1.1 Send Byte	30
5.1.2 Write Byte/Word, Write 32 protocol, Write 64 protocol.....	31
5.1.3 Read Byte/Word, Read 32 protocol, Read 64 protocol.....	32
5.1.4 Block Write	33

5.1.5	Block Read	34
5.1.6	Block Write-Block Read Process Call	35
5.1.7	SMBALERT#	36
5.1.8	SMBus Host Notify	37
6.	Website and Support	38

1. Overview

This module consists of a driver layer (PMBus Slave Driver) and a middleware layer (PMBus Slave Controller) and provides an interface to send and receive data via PMBus (Power Management Bus) communication.

Figure 1-1 Module construction



1.1 PMBus Slave Driver (PMBSDRV) Features

The PMBus Slave Driver is intended to be accessed from the middleware layer (PMBus Slave Controller).

The PMBus Slave Driver provides the following features as the driver layer of the PMBus Slave module.

1.1.1 Bus Communication Features

PMBus slave send and receive operations are performed using the serial interface IICA.

(1) Address setting

The physical address of the slave device can be set by one of the following methods

- Set an any fixed address value with the Smart Configurator
- Select any input pin (7bit) in the Smart Configurator and set the address value based on the input level of that pin

(2) Bus speed

100 kHz, 400 kHz, and 1MHz can be set with the Smart Configurator.

(3) Event Notification

When the following events are detected in serial bus communication, notification is sent to upper layers.

- Start Condition
- Stop Condition
- Receive data addressed to own station address (Notification is made to the upper layer in Byte units)
- Send to SMBus Alert Response Address ^(Note1)
- Communication Error Detection (Details are explained in the next chapter)

Note1. Notification to upper layers is made only when the SMBALERT# signal is actively output.

1.1.2 Communication Error Detection

The PMBus Slave module's driver layer detects the following communication errors and notifies the upper layers.

(1) Response data length error

If more data than expected is requested when responding to the master (ACK is detected in spite of the timing for sending the last data), it is judged as a send error. If NACK is detected at the timing of sending less data than expected, it is judged as a send error as well.

(2) Protocol errors

The PMBus specification stipulates that the R/W bit of the Address Byte should be sent as the W setting at the start of a communication transaction. (*Except when sending to SMBus Alert Response Address)

If an Address Byte with the R setting is sent to the own station address at the transaction start timing, this is not accepted and is judged as a protocol error.

(3) Communication timeout

Byte data reception timing is monitored during a communication transaction, and a communication timeout is determined when the reception interval is 25 ms or longer. Byte data reception timing monitoring is achieved by detecting INTIICA0 interrupts. This enables detection of a Low fixation state on the SCLA0 line.

1.1.3 Optional Pin Features

The following optional terminal ports can be assigned in the Smart Configurator.

(1) Control Signal (CONTROL)

The CONTROL pin is an optional input pin; it is used in conjunction with the PMBus command to turn the device on or off. The active level of the pin can also be set by the PMBus command.

(2) Write Protect (WP)

The WP pin is an optional input pin. When the input level is High, updating of the internal memory must be inhibited.

(3) SMBALERT#

The SMBALERT# pin is an optional output pin. It is used as an interrupt line from the slave to the master.

*Connect the pull-up resistor when using the SMBALERT# pin.

Note:

The electrical characteristics of these pins follow those described in the "RL78/G24 User's Manual: Hardware".



1.2 PMBus Slave Controller (PMBCTL) Features

The PMBus Slave Controller is intended to be accessed by user applications.
As the middleware layer of the PMBus Slave module, it provides the following features

1.2.1 Communication Formats

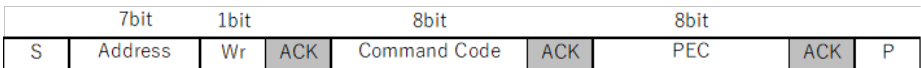
Upon receiving notification from the PMBus Slave Driver, the module constructs packet data and interprets the communication format. The communication formats supported by this module and their protocol diagrams are described below.

Protocol diagram legend:

- S : Start Condition
- Sr : Repeat Start Condition
- Rd : Read (1)
- Wr : Write (0)
- ACK : Acknowledge
- NACK : Not Acknowledge
- PEC : Packet Error Code *PEC support is optional
- P : Stop Condition
-  : Master → Slave
-  : Slave → Master

(1) Send Byte
Command code is sent to any target address.

Figure 1-2 Communication Format Send Byte



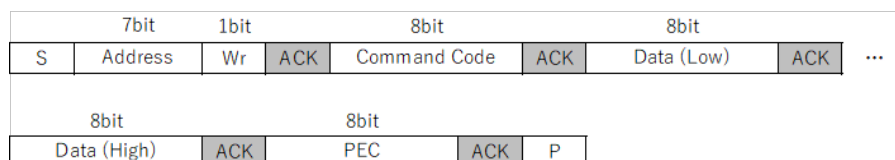
(2) Write Byte/Word, Write 32 protocol, Write 64 protocol

Command code and corresponding write data are sent to any target address.

Figure 1-3 Communication Format Write (Single byte)



Figure 1-4 Communication Format Write (Multi byte)



(3) Read Byte/Word, Read 32 protocol, Read 64 protocol

Command code is sent to any target address. Next, the response data corresponding to the command code is read from the device.

Figure 1-5 Communication Format Read (Single byte)

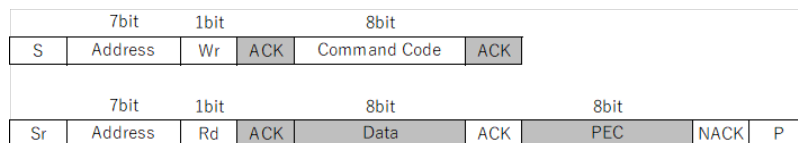
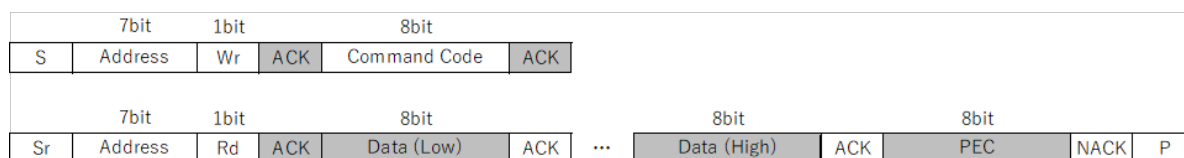


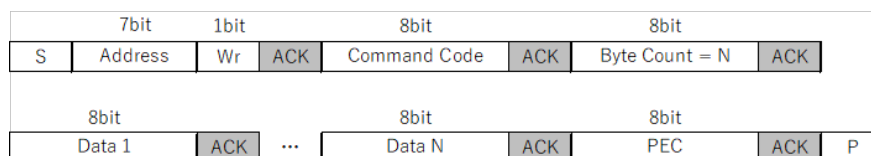
Figure 1-6 Communication Format Read (Multi byte)



(4) Block Write

Command code and corresponding write data are sent to any target address. Before sending write data, data (Byte Count) indicating the data length is sent.

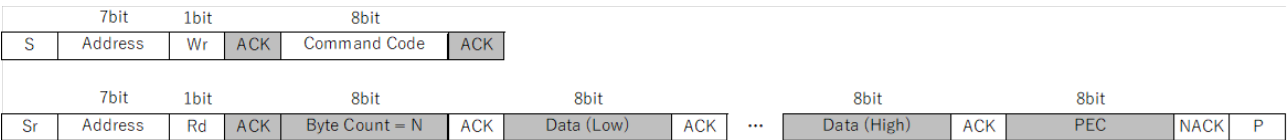
Figure 1-7 Communication Format Block Write



(5) Block Read

Command code is sent to any target address. Next, the response data corresponding to the command code is read from the device. Before sending the response data, the device sends data (Byte Count) indicating the data length.

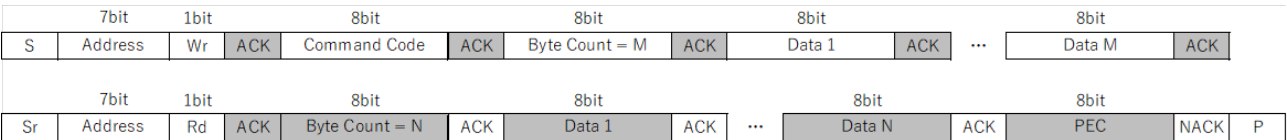
Figure 1-8 Communication Format Block Read



(6) Block Write-Block Read Process Call

The Block Write and Block Read described above are executed in the same transaction. Write data length (Byte Count M) and response data length (Byte Count N) may differ.

Figure 1-9 Communication Format Block Write-Block Read Process Call



1.2.2 User Notification of Received Commands and Data

When packet data consisting of the communication format described above is received, the received command and data are notified to the user. The flow from command reception to user notification is described below.

(1) Command support check (judgment in module)

The command support is determined at the timing when the command code is sent to the home station address. The command support is determined using the "gs_pmbstcl_command_list" table variable in the "rm_pmbstcl.c" file. In the "gs_pmbstcl_command_list" table variable, each command defined in the PMBus specification is registered as the default setting.

The following commands are manufacturer-specific. Set the command information in the "gs_pmbstcl_command_list" table variable to support these commands. If not, the command will be judged as unsupported.

- MFR_SPECIFIC_C4 - MFR_SPECIFIC_FD
- MFR_SPECIFIC_COMMAND_EXT
- PMBUS_COMMAND_EXT

(2) Command support check (user determination)

Next, the received command code is notified to the user in a callback. The user must return a return value to the callback function indicating whether the notified command is supported in the application. (Alternatively, the "gs_pmbstcl_command_list" table variable could be populated with only those commands supported by the user application, and the callback function would always return a value indicating support.)

(3) Communication format judgment

After checking that the received command is supported by the above-mentioned method, the communication format is determined. The PMBus specification defines the communication format to be applied to each command, and based on this, it judges that packet data was received in the correct communication format. The correspondence between commands and communication formats is defined in the "gs_pmbstcl_command_list" table variable.

(4) Command reception notification

When packet data is received in the correct communication format, the received command code and received data are notified to the user via callback.

1.2.3 Fault Detection and User Notification

This feature detects each fault that occurs during data communication and notifies the user via callback. For details, please refer to chapter 4.1.2 pmbstcl_fault_t.

1.2.4 Reporting to the Controller

PMBus devices must notify the controller (master) of warnings and fault conditions during the aforementioned Fault notification or when a device abnormality is detected. This module provides the following two notification features; the PMBus specification specifies that at least one of these two methods must be supported.

(1) SMBALERT# Notification by Signal

The user can generate an alert signal from the SMBALERT# pin by calling the API function "RM_PMBUSCTL_SetSMBAlert". The controller that detects the alert signal makes a read request to the SMBus Alert Response Address in order to identify the device that is the source of the notification. The device responds to this request with its own station address value.

Figure 1-10 Communication Format Device responds to an ARA

	7bit	1bit		8bit		8bit		
S	Alert Response Address	Rd	ACK	Device Address	ACK	PEC	NACK	P

(2) Notification by SMBus Host Notify Protocol

The user can call the API function "RM_PMBUSCTL_SetHostNotify" to notify device addresses and error information (STATUS_WORD) via the SMBus Host Notify protocol. The SMBus Host Notify protocol can be called.

Figure 1-11 Communication Format Host Notify

	7bit	1bit		8bit		8bit		8bit		
S	SMBus Host Address	Wr	ACK	Device Address	ACK	STATUS_WORD(Low)	ACK	STATUS_WORD(High)	ACK	P

1.2.5 H/W Signal Information

The user can call the API function "RM_PMBUSCTL_GetHWSignal" to obtain the signal levels of the following optional pins.

- Control Signal (CONTROL)
- Write Protect (WP)
- SMBALERT#

2. API Information

This section describes the API information for this module.

2.1 Hardware Requirements

The MCU to be used must support the following pins

- SCLA0 (P60)
- SDAA0 (P61)

Target products: 30, 32, 40, 44, 48, 52, 64-pin products

2.2 Software Requirements


This driver depends on the following modules

- Board Support Package (r_bsp) v1.60 or later

In addition, the following API functions of r_bsp must be enabled, which can be configured from the Software Component Settings screen on the Smart Configurator.

- R_BSP_GetFclkFreqHz
(BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE = 0)

Figure 2-1 Smart Configurator BSP Setting

▼  Configurations	
# Start up select	Enable (use BSP startup)
# Control of illicit memory access detection(IAWEN)	Disable
# Protected area in the RAM(GRAM0-1)	Disabled
# Protection of the port control registers(GPORT)	Disabled
# Protection of the interrupt control registers(GINT)	Disabled
# Protection of the clock, voltage detector, and RAM parity error detection control regi	Disabled
# Data flash memory area/extra area access control(DFLEN)	Disables
# Initialization of peripheral functions by Code Generator/Smart Configurator	Enable
# API functions disable(R_BSP_StartClock, R_BSP_StopClock)	Disable
# API functions disable(R_BSP_GetFclkFreqHz)	Enable
# API functions disable(R_BSP_SetClockSource)	Disable

2.3 Supported Tool Chains

This module has been tested with the following toolchains.

- Renesas CC-RL Toolchain v1.12.01
- IAR Embedded Workbench for Renesas RL78 v5.10.2

2.4 Header files

API calls and I/F definitions used are described in "rm_pmbstcl_api.h" and "rm_pmbdrv_api.h".

2.5 Integer Type

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

2.6 Code Size

ROM and RAM sizes increase or decrease depending on the settings on the Smart Configurator and compiler option settings. Here, the sizes are given for reference when the settings on the Smart Configurator are the default settings and the compile options on the CC-RL compiler are set to the default settings.

ROM : 3,891 [byte]

RAM : 1,693 [byte]

3. Configuration Specifications

A list of configuration items that can be set in the Smart Configurator is shown below.

Table 3.1 PMBus Slave Driver setting items list (1/2)

Item	Possible values	Description
Bus Speed	100kHz, 400kHz, 1MHz	Specify the bus speed.
SDAA and SCLA signal falling times (tF)[ns]	0 to 300	Set the fall time of the SDAA and SCLA signals. (Note1)
SDAA and SCLA signal rising times (tR)[ns]	0 to 1000	Set the rise time of the SDAA and SCLA signals. (Note1)
Digital filter	ON, OFF	Specify whether the digital filter is enabled or not. (Note2)
Interrupt level for INTIICA0	Level 0(Highest), Level 1, Level 2, Level 3(Lowest)	Set the interrupt priority level for INTIICA0.
How to set the device address	Setting by Pin, Setting by GUI	Specify how to set the device address.
Device address	0x00 to 0xFE	Set the device address.
Pin for setting device address 1	P00 to P147	Select the device address (bit1) pin.
Pin for setting device address 2	P00 to P147	Select the device address (bit2) pin.
Pin for setting device address 3	P00 to P147	Select the device address (bit3) pin.
Pin for setting device address 4	P00 to P147	Select the device address (bit4) pin.
Pin for setting device address 5	P00 to P147	Select the device address (bit5) pin.
Pin for setting device address 6	P00 to P147	Select the device address (bit6) pin.
Pin for setting device address 7	P00 to P147	Select the device address (bit7) pin.

Note1. tF and tR should be set according to the user's hardware environment.

Note2. When f_{CLK} is 48 MHz, the filter width of the digital filter is 41.67 ns. To obtain a filter width of 50 ns or more, set f_{CLK} to 32 MHz or lower or use an external noise filter.

Table 3.2 PMBus Slave Driver setting items list (2/2)

Item	Possible values	Description
Pin for Control Signal	Unused, P00 to P147	Select the Control Signal pin.
Pin for Write Protect	Unused, P00 to P147	Select the Write Protect pin.
Pin for SMBALERT#	Unused, P00, P02 to P04, P10 to P15, P17, P30,P50, P51, P55, P71 to P74	Select the SMBALERT# pin.
Timer resource for device timeout measurement	TAU0_0, TAU0_1, TAU0_2, TAU0_3	Select the timer source for measuring device timeout.

Table 3.3 PMBus Slave Controller setting items list

Item	Possible values	Description
Support the SMBus Packet Error Checking (PEC)	Supported, Not Supported	Select whether to support PEC.

4. API Specification

4.1 API Typedef Definitions (PMBus Slave Controller)

This section describes the Typedef definition provided by the middleware layer of this module.

4.1.1 pmbectl_cbk_ret_t

This typedef defines the return value of the user callback function.

```
typedef enum
{
    PMBSCTL_CBK_RTN_OK,
    PMBSCTL_CBK_RTN_CMD_NOT_SUPPORTED,
    PMBSCTL_CBK_RTN_DATA_NG,
    PMBSCTL_CBK_RTN_DEVICE_BUSY,
} pmbectl_cbk_ret_t;
```

Description

Use as the return value of the user callback function.

- (a) PMBSCTL_CBK_RTN_OK
If the notified receive command or data is acceptable
- (b) PMBSCTL_CBK_RTN_CMD_NOT_SUPPORTED
If the notified receive command is an unsupported command
- (c) PMBSCTL_CBK_RTN_DATA_NG
If the received data notified is out of range, an invalid value
Or, if data update is requested during Write Protect.
- (d) PMBSCTL_CBK_RTN_DEVICE_BUSY
If the device is busy and cannot return a response

4.1.2 pmbctl_fault_t

This typedef defines the fault information to be notified by the user callback function "FaultNotification".

```
typedef enum
{
    PMBSCTL_FAULT_CML_PEC,
    PMBSCTL_FAULT_CML_INVALID_DATA,
    PMBSCTL_FAULT_CML_OTHER,
    PMBSCTL_FAULT_BUSY,
} pmbctl_fault_t;
```

Description

It is used as fault information notified by the FaultNotification callback function.

Fault information is determined as follows based on the PMBus specification.

- (a) PMBSCTL_FAULT_CML_PEC
Notified when a PEC error is detected.
- (b) PMBSCTL_FAULT_CML_INVALID_DATA
Notified when any of the following is detected
 - The number of bytes of data received from the master is larger than expected.
 - If the user returns a return value of "PMBSCTL_CBK_RTN_CMD_NOT_SUPPORTED" or "PMBSCTL_CBK_RTN_DATA_NG" is received.
- (c) PMBSCTL_FAULT_CML_OTHER
Notified when any of the following is detected.
 - When the number of bytes of data received from the master is smaller than expected.
 - When NACK is detected before all data is sent in response to the master.
 - When ACK is continued even though all data has been sent when responding to the master.
 - When a protocol error is detected.
- (d) PMBSCTL_FAULT_BUSY
Notified when a callback receives a command or data reception with the return value "PMBSCTL_CBK_RTN_DEVICE_BUSY" is notified if received.

4.1.3 pmbctl_polarity_t

This typedef defines the active level of the Control pin.

```
typedef enum
{
    PMBSCTL_POLARITY_ACTIVE_LO,
    PMBSCTL_POLARITY_ACTIVE_HI,
} pmbctl_polarity_t;
```

Description

Use the API function "RM_PMBCTL_SetPolarityControlPin" to set the active level of the Control pin.

4.1.1 pmbctl_data_t

This typedef defines the data structures that are sent and received via PMBus communication.

```
typedef struct
{
    uint8_t    data_length;
    uint8_t *  p_data;
} pmbctl_data_t;
```

Description

Use when setting received data notification and response data by user callback function.

- (a) data_length
Indicates data length [byte].
- (b) p_data
Indicates the initial address of the array variable in which the received or response data is stored.

4.1.1 pmbstl_callback_t

This typedef defines a user callback function structure.

```
typedef struct
{
    pmbstl_cbk_ret_t (* CheckCommandSupported)(uint8_t command);
    pmbstl_cbk_ret_t (* SendCommandReceived)(uint8_t command);
    pmbstl_cbk_ret_t (* WriteDataReceived)(uint8_t command, pmbstl_data_t wdata);
    pmbstl_cbk_ret_t (* ReadDataReceived)(uint8_t command, pmbstl_data_t * p_rdata);
    pmbstl_cbk_ret_t (* WriteReadDataReceived)(uint8_t command, pmbstl_data_t wdata,
                                              pmbstl_data_t * p_rdata);
    void (* FaultNotification)(pmbstl_fault_t fault);
} pmbstl_callback_t;
```

Description

Register the callback function by storing the user function pointer in this structure and passing it as an argument when calling the API function "RM_PMBSTL_Open". The callback function will notify the user at each event timing.

(a) CheckCommandSupported

Notification is made when a command code is received from the master. The user performs support judgment on the command code passed as an argument. The judgment result should be returned as a return value.

(b) SendCommandReceived

Notification is made when reception is detected in Send Byte communication format. The received command code is passed as an argument. The processing result should be returned in the return value.

(c) WriteDataReceived

Notification is given at the timing when reception in Write Byte/Word, Write 32 protocol, Write 64 protocol communication format or Block Write communication format is detected. The received command code and data are passed as arguments. The processing result should be returned in the return value.

(d) ReadDataReceived

Notification is made at the timing when reception in Read Byte/Word, Read 32 protocol, Read 64 protocol communication format or Block Read communication format is detected. The received command code and a pointer variable for setting response data are passed as arguments. Set the response data and return the processing result in the return value.

(e) WriteReadDataReceived

Notification is made at the timing when reception is detected in the Block Write-Block Read Process Call communication format. The received command code and data and a pointer variable for setting response data are passed as arguments. Set the response data and return the processing result in the return value.

(f) FaultNotification

Notification is made at the timing of fault detection. The detected fault information is passed as an argument; refer to 4.1.2 pmbstl_fault_t for details on fault information.

4.1.2 pmbctl_hw_signal_t

This Typedef defines the H/W signal structure.

```
typedef struct
{
    bool control;
    bool write_protect;
    bool smbalert;
} pmbctl_hw_signal_t;
```

Description

Use as the return value of the API function "RM_PMBCTL_GetHWSignal". Each H/W signal information is defined as a member of the structure.

- (a) control
 - true : device ON
 - false : device OFF
- (b) write_protect
 - true : internal memory update prohibited
 - false : internal memory update permitted
- (c) smbalert
 - true : Asserting SMBALERT# signal
 - false : Negating SMBALERT# signal

4.2 API Function Specifications (PMBus Slave Controller)

This section describes the API function specifications provided by the middleware layer of this module.

4.2.1 RM_PMBUSCTL_Open

This function initializes the module and starts the PMBus communication feature.

Format

```
void RM_PMBUSCTL_Open (const pmbusctl_callback_t * p_callback_set)
```

Parameters

p_callback_set

Pointer to user callback function structure

Return Values

None

Properties

Prototype declared in rm_pmbusctl_api.h.

Description

Initialize the driver layer and middleware layer and start the PMBus communication feature. It also registers the user callback function passed as an argument.

Example

```
/** User function */
static pmbusctl_cbk_ret_t r_cbk_check_command_supported(uint8_t command);
static pmbusctl_cbk_ret_t r_cbk_send_command_received(uint8_t command);
static pmbusctl_cbk_ret_t r_cbk_write_data_received(uint8_t command, pmbusctl_data_t wdata);
static pmbusctl_cbk_ret_t r_cbk_read_data_received(uint8_t command, pmbusctl_data_t * p_rdata);
static pmbusctl_cbk_ret_t r_cbk_write_read_data_received(uint8_t command, pmbusctl_data_t wdata,
                                                         pmbusctl_data_t * p_rdata);

static void r_cbk_fault_notification(pmbusctl_fault_t fault);

/** Callback function set */
static pmbusctl_callback_t gs_pmbusctl_cbk;
. . .

/** User Init */
gs_pmbusctl_cbk.CheckCommandSupported = r_cbk_check_command_supported;
gs_pmbusctl_cbk.SendCommandReceived = r_cbk_send_command_received;
gs_pmbusctl_cbk.WriteDataReceived = r_cbk_write_data_received;
gs_pmbusctl_cbk.ReadDataReceived = r_cbk_read_data_received;
gs_pmbusctl_cbk.WriteReadDataReceived = r_cbk_write_read_data_received;
gs_pmbusctl_cbk.FaultNotification = r_cbk_fault_notification;

RM_PMBUSCTL_Open(gs_pmbusctl_cbk);
```

4.2.2 RM_PMBCTL_Close

This function performs the module shutdown process and terminates the PMBus communication feature.

Format

```
void RM_PMBCTL_Open (void)
```

Parameters

None

Return Values

None

Properties

Prototype declared in rm_pmbctl_api.h.

Description

Stop the driver layer and middleware layer, and terminate the PMBus communication feature.

Example

```
/** Terminate PMBus communication */  
RM_PMBCTL_Close();
```

4.2.3 RM_PMBCTL_GetHWSignal

This function obtains H/W signal information.

Format

```
pmbctl_hw_signal_t RM_PMBCTL_GetHWSignal (void)
```

Parameters

None

Return Values

H/W Signal Information

Properties

Prototype declared in rm_pmbctl_api.h.

Description

Obtain the following H/W signal information.

- Control Signal input status
- Write Protect input status
- SMBALERT# output status

Example

```
static pmbctl_hw_signal_t gs_hw_signal;

/* get H/W signal */
gs_hw_signal = RM_PMBCTL_GetHWSignal();

if (gs_hw_signal.control == true)
{
    . . .
}
```

4.2.4 RM_PMBCTL_SetPolarityControlPin

This function sets the polarity of the Control Signal input pin.

Format

```
void RM_PMBCTL_SetPolarityControlPin (pmbctl_polarity_t active_level)
```

Parameters

active_level

Control Signal pin active level

Return Values

None

Properties

Prototype declared in rm_pmbctl_api.h.

Description

Set the polarity of the Control Signal input terminal, corresponding to the terminal polarity setting by the ON_OFF_CONFIG command of the PMBus specification.

Example

```
/* Set active level of the Control Signal pin */  
RM_PMBCTL_SetPolarityControlPin(PMBCTL_POLARITY_ACTIVE_HI);
```

4.2.5 RM_PMBCTL_SetSMBAlert

This function sets the output level of the SMBALERT# pin.

Format

```
void RM_PMBCTL_SetSMBAlert (bool alert_level)
```

Parameters

alert_level

true : SMBALERT#信号をアサート

false : SMBALERT#信号をネゲート

Return Values

None

Properties

Prototype declared in rm_pmbctl_api.h.

Description

Set the output level of the SMBALERT# pin. When the output level is set to assert, it is automatically switched to the negate setting when the response of the own station address to the SMBus Alert Response Address is completed.

Example

```
/** Fault Notification user callback */
static void r_cbk_fault_notification(pmbctl_fault_t fault)
{
    . . .

    RM_PMBCTL_SetSMBAlert(true);
}
```

4.2.6 RM_PMBSCTL_SetHostNotify

This function uses the HostNotify protocol to notify device failure and warning conditions.

Format

```
void RM_PMBSCTL_SetHostNotify (uint16_t status_word)
```

Parameters

status_word

STATUS_WORD Information

Return Values

None

Properties

Prototype declared in rm_pmbectl_api.h.

Description

Send device failure and warning status (STATUS_WORD information) using the SMBus Host Notify protocol.

Example

```
static uint16_t gs_status_word;

/** Fault Notification user callback */
static void r_cbk_fault_notification(pmbectl_fault_t fault)
{
    . . .

    /** Update status_word */
    gs_status_word = . . .

    /** Set Host Notify */
    RM_PMBSCTL_SetHostNotify(status_word);
}
```

4.3 API Typedef Definitions (PMBus Slave Driver)

This section describes the Typedef definitions provided by the driver layer of this module. Since the driver layer is basically assumed to be accessed from the PMBus Slave Controller (middleware layer), users do not need to be particularly aware of it, but it is provided here as reference information.

Table 4.1 PMBSDRV callback return value enumeration type (pmbdrv_cbk_ret_t)

Macro	Macro Value	Description
PMBSDRV_CBK_RTN_OK	0	Normal end
PMBSDRV_CBK_RTN_NG	1	Abnormal end

Table 4.2 PMBSDRV port level enumerated type (pmbdrv_port_lv_t)

Macro	Macro Value	Description
PMBSDRV_PORT_LO	0	Port Lo level
PMBSDRV_PORT_HI	1	Port Hi level

Table 4.3 PMBSDRV error notification enumeration type (pmbdrv_err_t)

Macro	Macro Value	Description
PMBSDRV_ERR_RESPONSE_DATA_LENGTH	0	Response data length error detection
PMBSDRV_ERR_PROTOCOL_VIOLATION	1	Protocol error detection
PMBSDRV_ERR_TIMEOUT	2	Timeout detection

Table 4.4 PMBSDRV callback structure (pmbdrv_callback_t)

Type	Function Pointer Name	Arguments	Description
void	ReceiveStartCondition	uint8_t address_byte	start condition receipt notification
pmbdrv_cbk_ret_t	ReceiveByteData	uint8_t data	Notification of receipt of data
void	ReceiveAlertResponse	void	Alert Response Address receipt notification
void	ReceiveStopCondition	void	stop condition receipt notification
void	ErrorNotification	pmbdrv_err_t error	Error detection notification

4.4 API Function Specifications (PMBus Slave Driver)

This section describes the API function specifications provided by the driver layer of this module. Since the driver layer is basically assumed to be accessed from PMBus Slave Controller (middleware layer), users do not need to be particularly aware of it, but it is provided here as reference information.

Table 4.5 R_PMBSDRV_Open

Function name	R_PMBSDRV_Open	Initialize I2C features and start PMBus communication.
Arguments	const pmbsdrv_callback_t * p_callback_set	Pointer to register callback function to upper layer module
Return value	-	-

Table 4.6 R_PMBSDRV_Close

Function name	R_PMBSDRV_Close	Stop the I2C feature and terminate PMBus communication.
Arguments	-	-
Return value	-	-

Table 4.7 R_PMBSDRV_GetDeviceAddress

Function name	R_PMBSDRV_GetDeviceAddress	Get the physical address information of the device.
Arguments	uint8_t device_address	Device Address Information
Return value	-	-

Table 4.8 R_PMBSDRV_GetControlLevel

Function name	R_PMBSDRV_GetControlLevel	Get the input level of the Control pin.
Arguments	-	-
Return value	pmbsdrv_port_lv_t level	Port Level

Table 4.9 R_PMBSDRV_GetWriteProtectLevel

Function name	R_PMBSDRV_GetWriteProtectLevel	Get the input level of the Write Protect pin.
Arguments	-	-
Return value	pmbsdrv_port_lv_t level	Port Level

Table 4.10 R_PMBSDRV_GetSMBALertLevel

Function name	R_PMBSDRV_GetSMBALertLevel	Get the output level of the SMBALERT# pin.
Arguments	-	-
Return value	pmbstdrv_port_lv_t level	Port Level

Table 4.11 R_PMBSDRV_SetSMBALertLevel

Function name	R_PMBSDRV_SetSMBALertLevel	Set the output level of the SMBALERT# pin.
Arguments	pmbstdrv_port_lv_t level	Port Level
Return value	-	-

Table 4.12 R_PMBSDRV_SetResponseData

Function name	R_PMBSDRV_SetResponseData	Set the response data to the master.
Arguments	uint8_t data_length	Send data length
Arguments	uint8_t *p_data	Pointer to sent data
Return value	-	-

Table 4.13 R_PMBSDRV_SendData

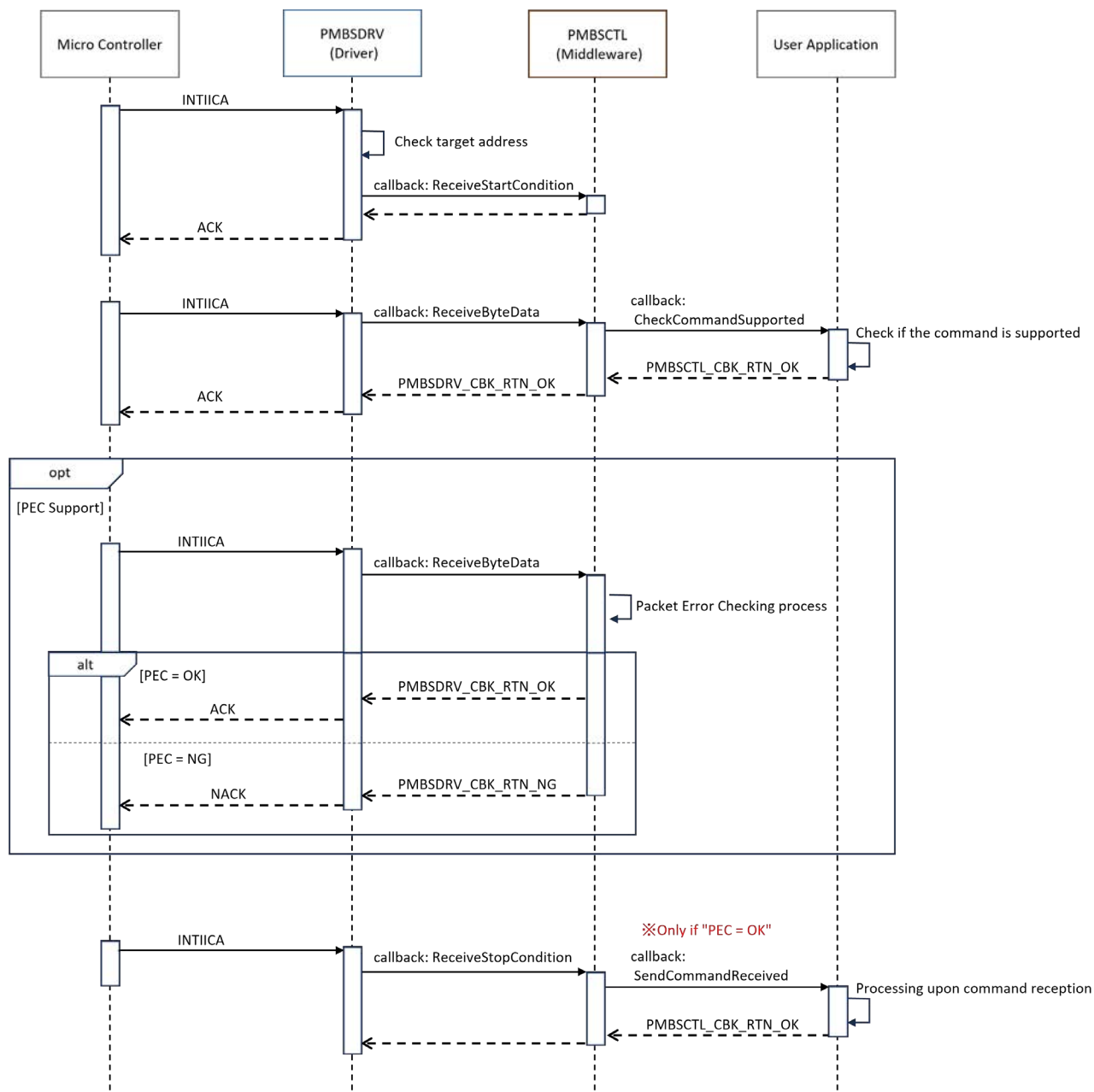
Function name	R_PMBSDRV_SendData	Send any data. (send master)
Arguments	uint8_t dest	Destination address
Arguments	uint8_t data_length	Send data length
Arguments	uint8_t *p_data	Pointer to sent data
Return value	-	-

5. Operation Sequence

Sequence diagrams are shown below for the operation upon command reception for each communication format and the notification operation to the controller (master) after fault detection.

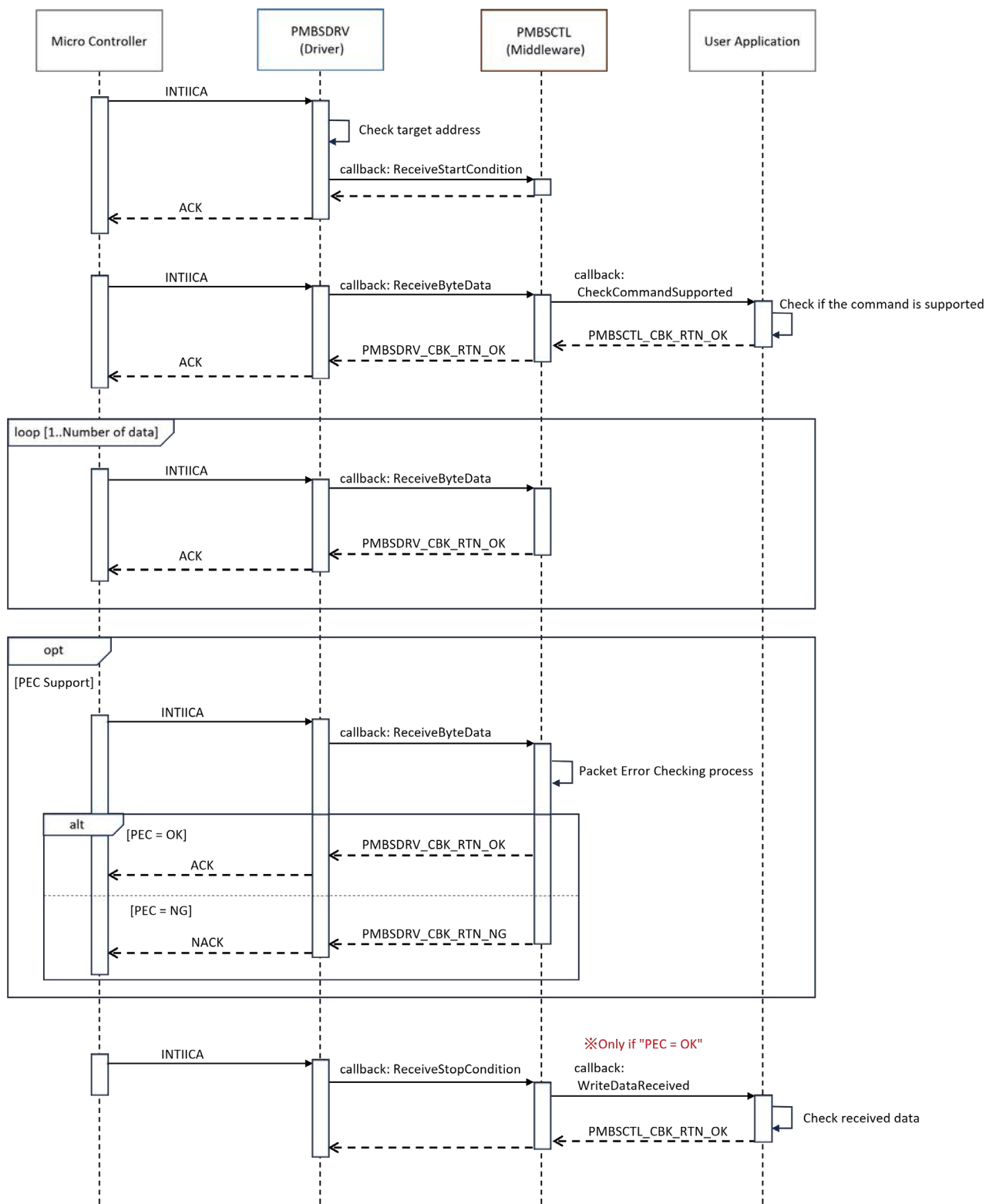
5.1.1 Send Byte

Figure 5-1 Send Byte sequence



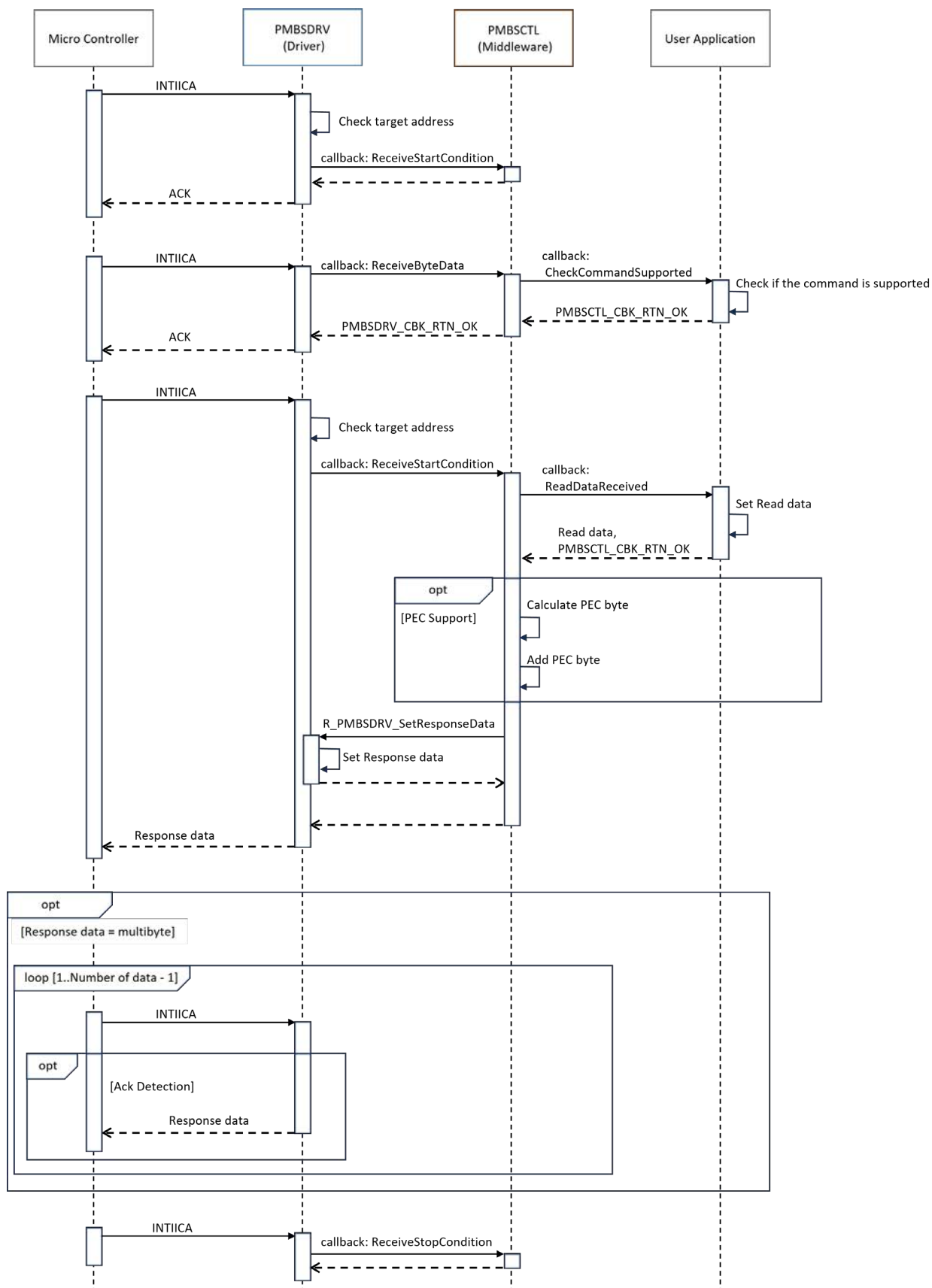
5.1.2 Write Byte/Word, Write 32 protocol, Write 64 protocol

Figure 5-2 Write sequence



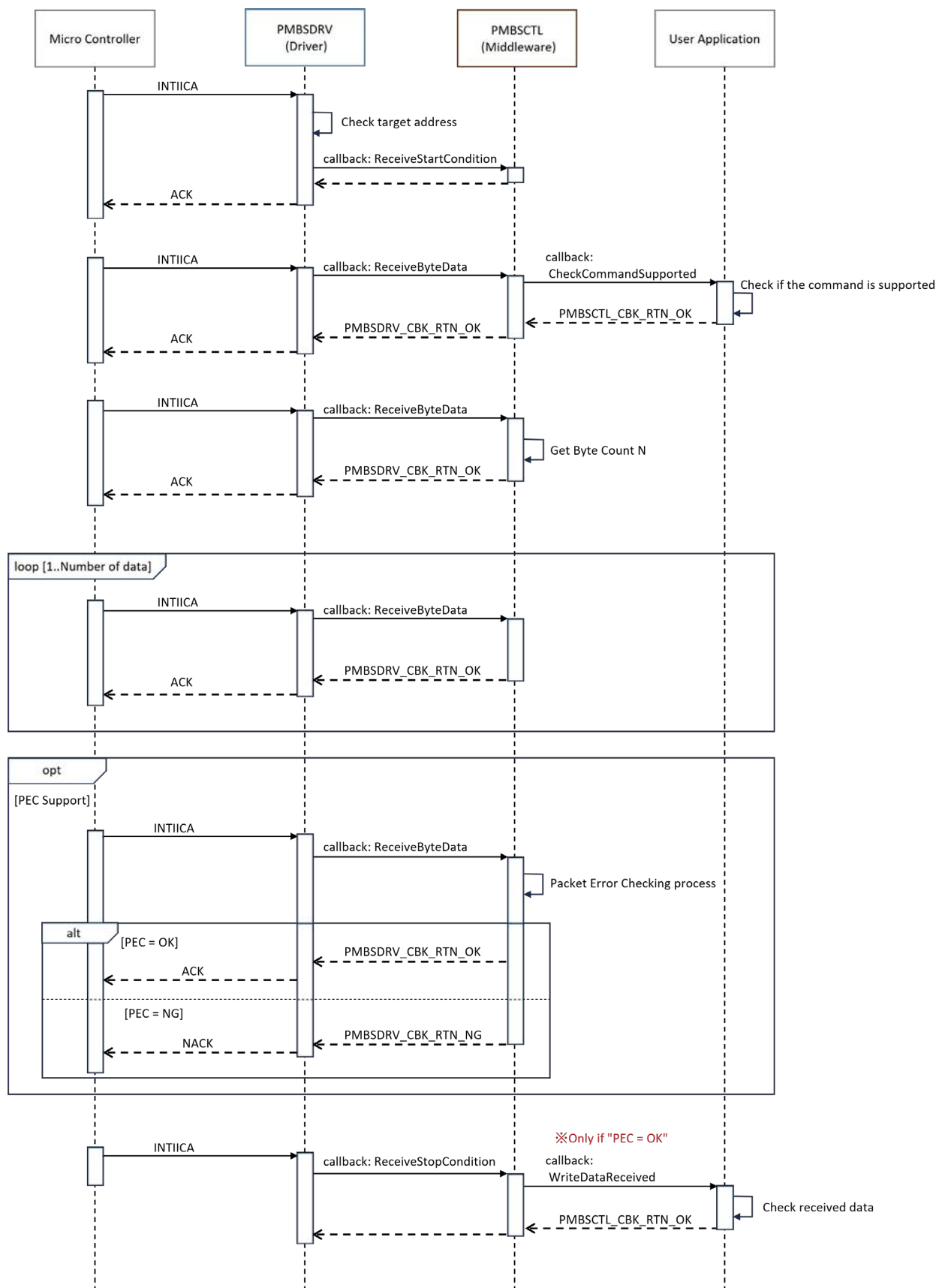
5.1.3 Read Byte/Word, Read 32 protocol, Read 64 protocol

Figure 5-3 Read sequence



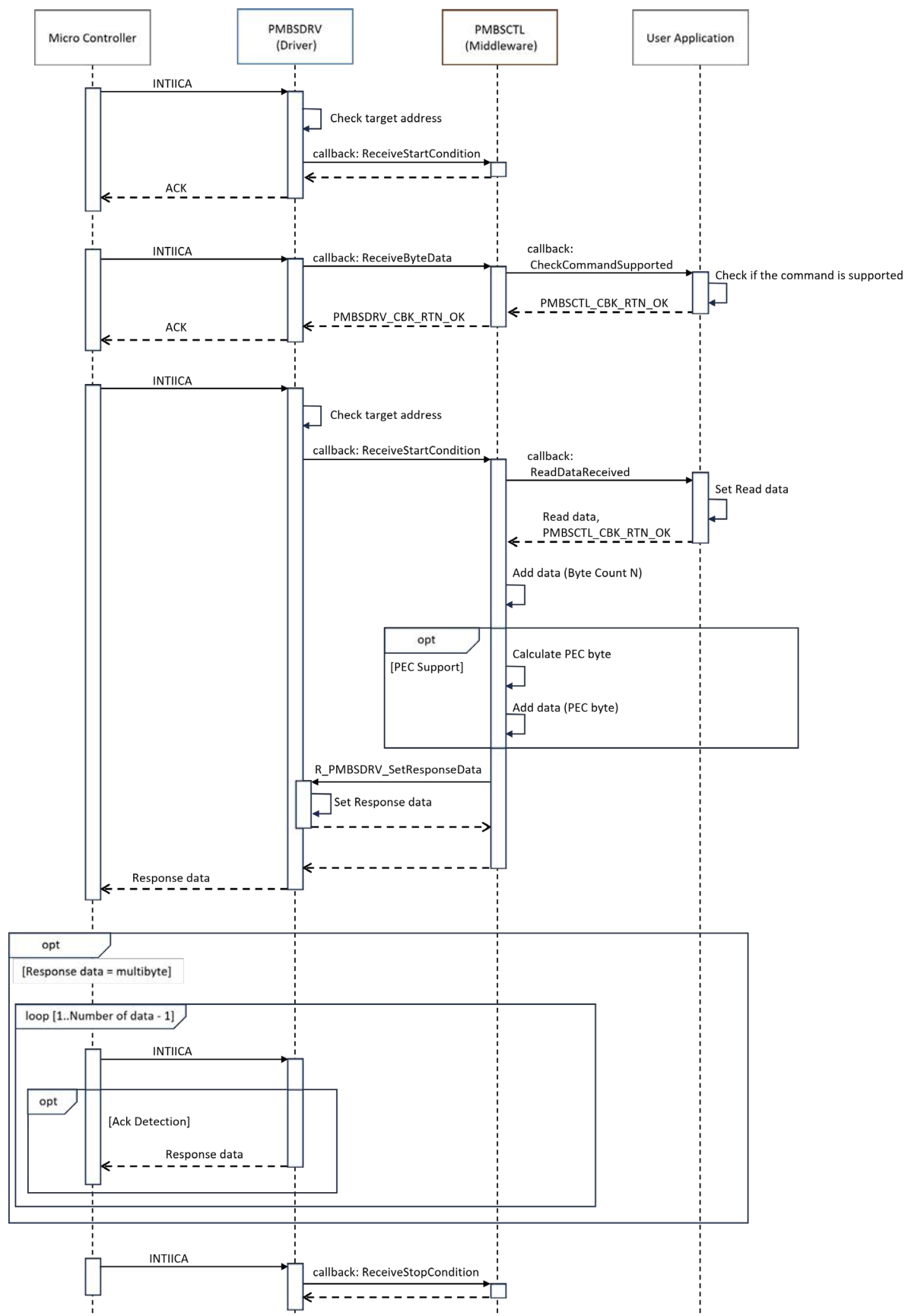
5.1.4 Block Write

Figure 5-4 Block Write sequence



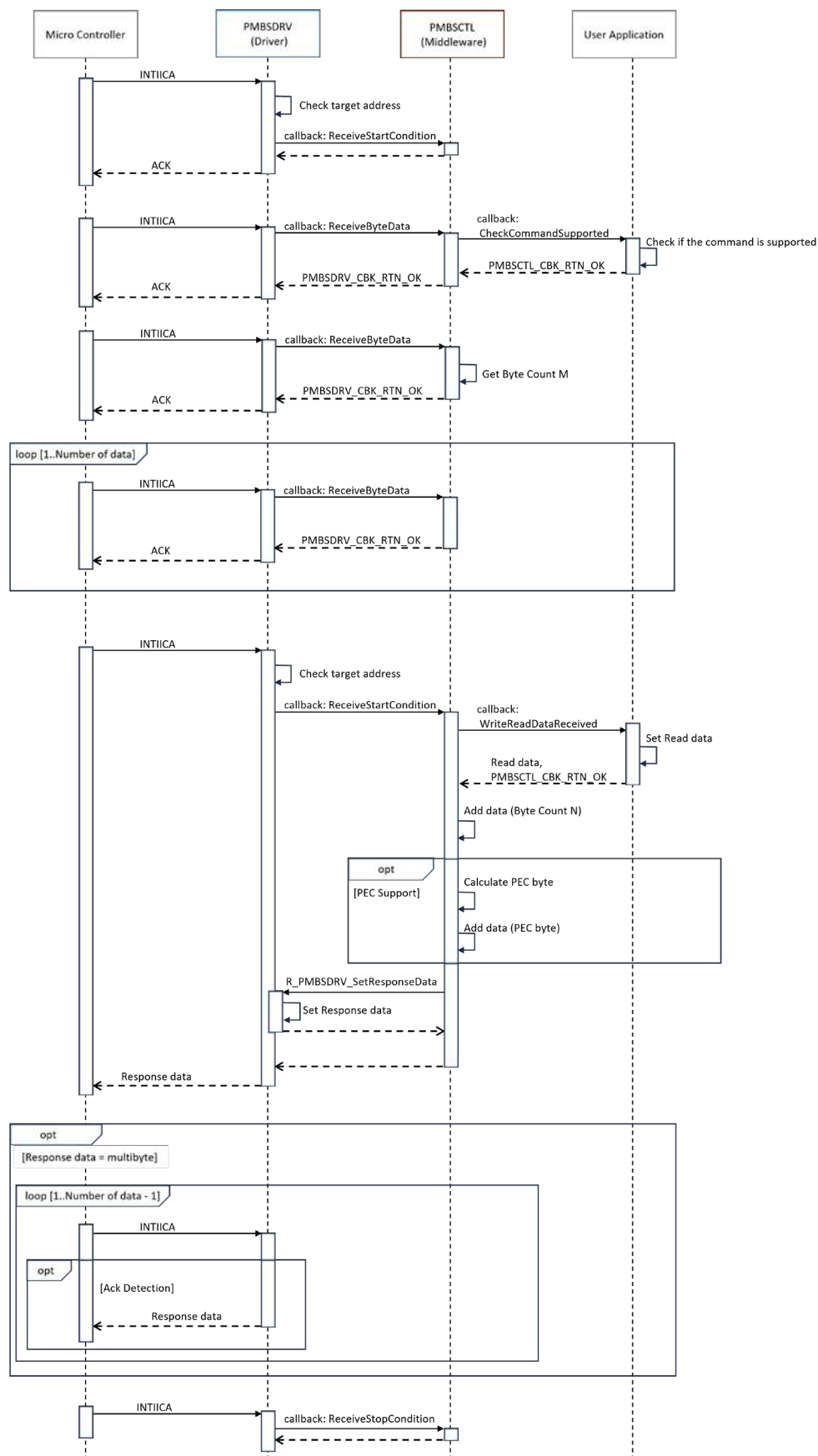
5.1.5 Block Read

Figure 5-5 Block Read sequence



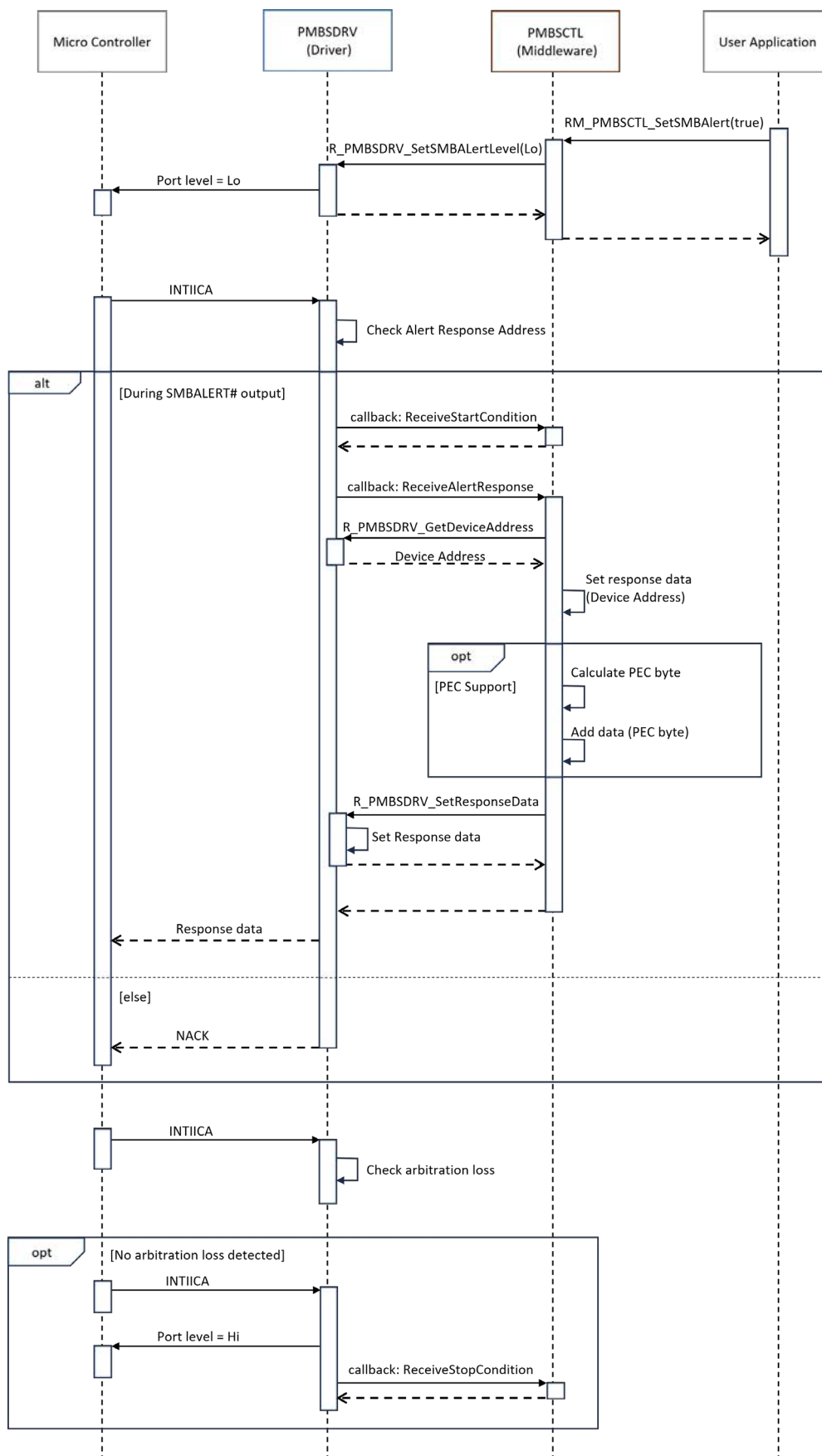
5.1.6 Block Write-Block Read Process Call

Figure 5-6 Block Write-Block Read Process sequence



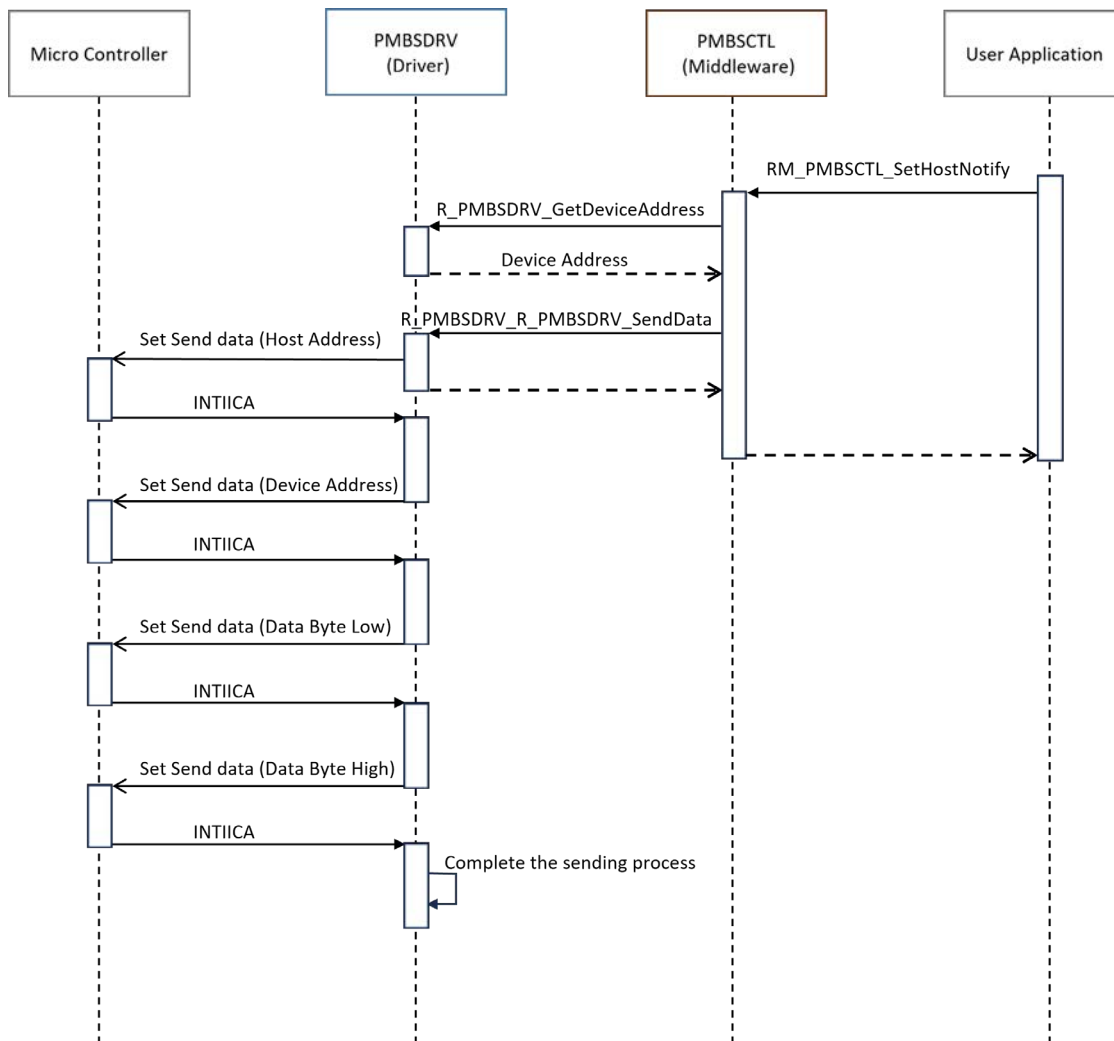
5.1.7 SMBALERT#

Figure 5-7 SMBALERT# sequence



5.1.8 SMBus Host Notify

Figure 5-8 SMBus Host Notify sequence



6. Website and Support

Renesas Electronics Website

<http://www.renesas.com/>

Contact information

<http://www.renesas.com/contact/>

All trademarks and registered trademarks are the property of their respective owners.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug.28, 2023	-	First edition issued
1.01	Apr.18, 2024	-	Supports Bus Speed 1MHz.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

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

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:

www.renesas.com/contact/.