RX Family/RA Family

PMBus Master-Slave communication using I2C bus interface (RIIC/I3C)

Summary

This application note is intended to explain how to use RX family RX26T group and RA family RA6T3 group to communicate and control PMBus. PMBus communication and control software includes PMBus Master software and PMBus Slave software. PMBus Master software executes in RX26T group, PMBus Slave software Execute in RX26T group, and RA6T3 group, and Execute PMBus communication between Master-Slave devices.

These sample programs are only to be used as reference and Renesas Electronics Corporation does not guarantee operations. Please use them after carrying out a thorough evaluation in a suitable environment.

■ Operation Check Device

The operation of the sample program is checked with the following devices.

- PMBus Master
  RX family RX26T Group (R5F526TFCDFP)

- PMBus Slave
  RX family RX26T Group (R5F526TFCDFP)
  RA family RA6T3 Group (R7FA6T3BB3CFM)

It is also applicable to RX/RA family that has the resources described in this application note or equivalent peripheral functions. (RX72T, RX66T, RX24T, RX23T, RX13T, RX72M, RX72N, RX66N, RA6T1, RA6T2, RA4T1 etc.)

■ Target Sample Program

The sample program for this application note is shown below.

- PMBus Master
  RX26T_MCBA_PMBUS_MASTER_E2S_V100 (IDE : e²studio)
  RX26T_MCBA_PMBUS_MASTER_CSP_V100 (IDE : CS+)

- PMBus Slave
  RX26T_MCBA_PMBUS_SLAVE_E2S_V100 (IDE : e²studio)
  RX26T_MCBA_PMBUS_SLAVE_CSP_V100 (IDE : CS+)
  RA6T3_MCILV1_PMBUS_SLAVE_E2S_V100 (IDE : e²studio)

■ Reference materials

- RX26T Group User's Manual Hardware (R01UH0979)
- RA6T3 Group User's Manual Hardware (R01UH0998)
- MCK-RX26T User's Manual (R12UZ0111)
- MCK-RA6T3 User's Manual (R12UZ0114)
- RX Famiky Sensorless Vector Control of a Permanent Magnet Synchronous Motor - For MCK (R01AN6858)
- Sensorless vector control for permanent magnetic synchronous motor For Renesas Flexible Motor Control (R01AN6839)

Contents

1. Overview

This application note is intended to provide PMBus communication and control methods using I2C bus interface (RIIC/I3C) *1 installed in RX family and RA family. Connect RX26T group to PMBus Master and RX26T group or RA6T3 group to PMBus Slave for PMBus transmission and reception between both devices in Master-Slave. In this application note, the RX26T group of the PMBus master analyzes and processes PMBus commands from PC. Then, it controls and monitors a permanent-magnet synchronous motor connected from PMBus Master to PMBus Slave at 100kbps communication rate in a vector control method.

Although PMBus is a communication method widely used in power supply systems, this application note uses the Flexible Motor Control Kit for the motor system shown below as an alternative to a power supply system.

*1: RIIC/I3C supports communication compliant with SMBus (Ver.2.0).

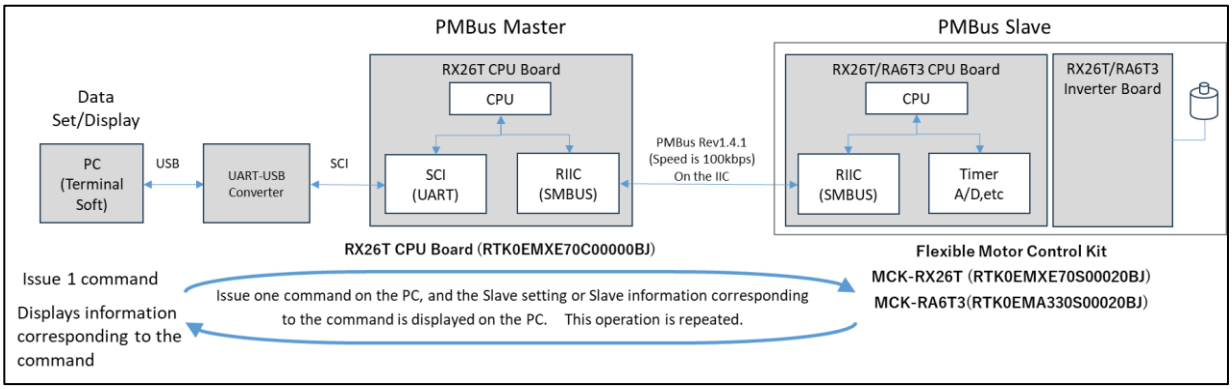


Figure 1 System Configuration and Operation Overview of This Application Note

1.1 Development Environment

Table 1 shows the hardware development environment for this application note. Table 2 shows the development environment for software for this application note.

Table 1 Hardware Development Environment

Master/Slave	MCU	Board name	Model
Master	RX26T (R5F526TFCDFP)	RX26T CPU Board	RTK0EMXE70C00000B
	-	USB-UART Conversion Module	Pmod-USBUART (Made by DIGILENT)
Slave	RX26T (R5F526TFCDFP)	MCK-RX26T Renesas Flexible Motor Control Kit for RX26T MCU Group (Include RX26T CPU Board)	RTK0EMXE70S00020BJ
	RA6T3 (R7FA6T3BB3CFM)	MCK-RA6T3 Renesas Flexible Motor Control Kit for RA6T3 MCU Group (Include RA6T3 CPU Board)	RTK0EMA330S00020BJ
	-	Motor (Included with RTK0EMXE70S00020BJ or RTK0EMXE70S00020BJ)	R42BLD30L3 (Made by MOONS')

Table 2 Software Development Environment

Device	IDE version	RX Smart Configurator	FSP	Toolchain version[*1]
RX26T (R5F526TFCDFP)	CS+:V8.12.00	Version 2.22.0	-	CC-RX: V3.06.00
	e²studio:2024-07	e²studio plug-in version		
RA6T3 (R7FA6T3BB3CFM)	e²studio:2024-07	-	V4.4.0	GCC ARM Embedded:13.2.1.arm-13-7

Note If the same version as the toolchain (C compiler) specified in the project does not exist in the import destination, the toolchain is not selected and an error occurs. Check the toolchain selection status in the project settings screen.
Refer to FAQ 3000404 for the selection procedure.
(https://en-support.renesas.com/knowledgeBase/18398339)

2. PMBus Outline

PMBus(Power Management Bus) is a general communication standard for power converters. It is based on SMBus communication standard for protocols derived from I2C. PMBus is used in servers, data sensors, and communication devices to monitor the power supply and set the power supply. PMBus is characterized by its ability to communicate with several slaves by synchronous communication of two-wire type (two-wire type of clock/data) based on SMBus, and is adopted mainly for industrial equipment. PMBus simplifies power-system component-to-component communication, enabling component configuration, control, and monitoring. Figure 2 shows a sample system configuration using PMBus.

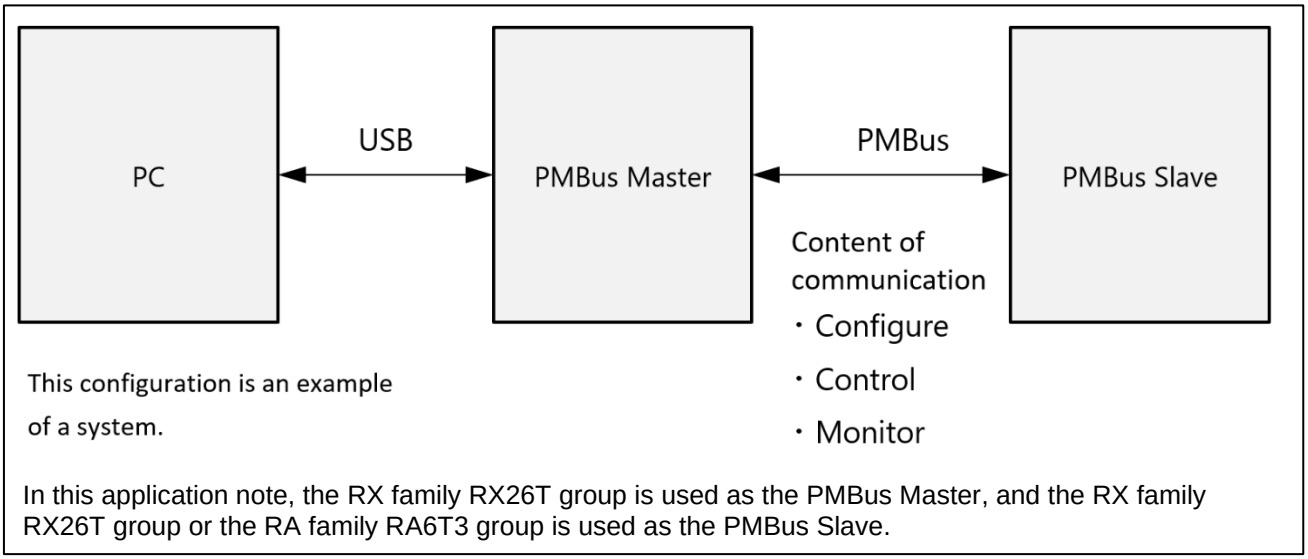


Figure 2 Structure of PMBus

2.1 PMBus protocols

The protocol format of PMBus uses a SMBus compliant transaction protocol. The transaction protocol consists of Send Byte, Write Byte, Write Word, Block Write, Receive Byte, Read Byte, Read Word, Block Read. The transaction protocols are shown in Figure 3 to Figure 6. PMBus commands are classified into standard command protocol, group command protocol, zone command protocol, extended command protocol, and bus master protocol. Both are command protocols that consist of SMBus based transactions. Of the five command protocols classified, the Group Command Protocol, the Zone Command Protocol, and the Extended Command Protocol are protocols dedicated to PMBus that extend SMBus protocol. This application note controls and monitors slave equipment using standard commands. For others, only protocol process is implemented as a reference, so please confirm according to your purpose of use. Figure 7 shows the standard command protocol format. Figure 8 shows the group command protocol, Figure 9 and Figure 10 shows the extended command protocol, and Figure 11 shows the zone command protocol.

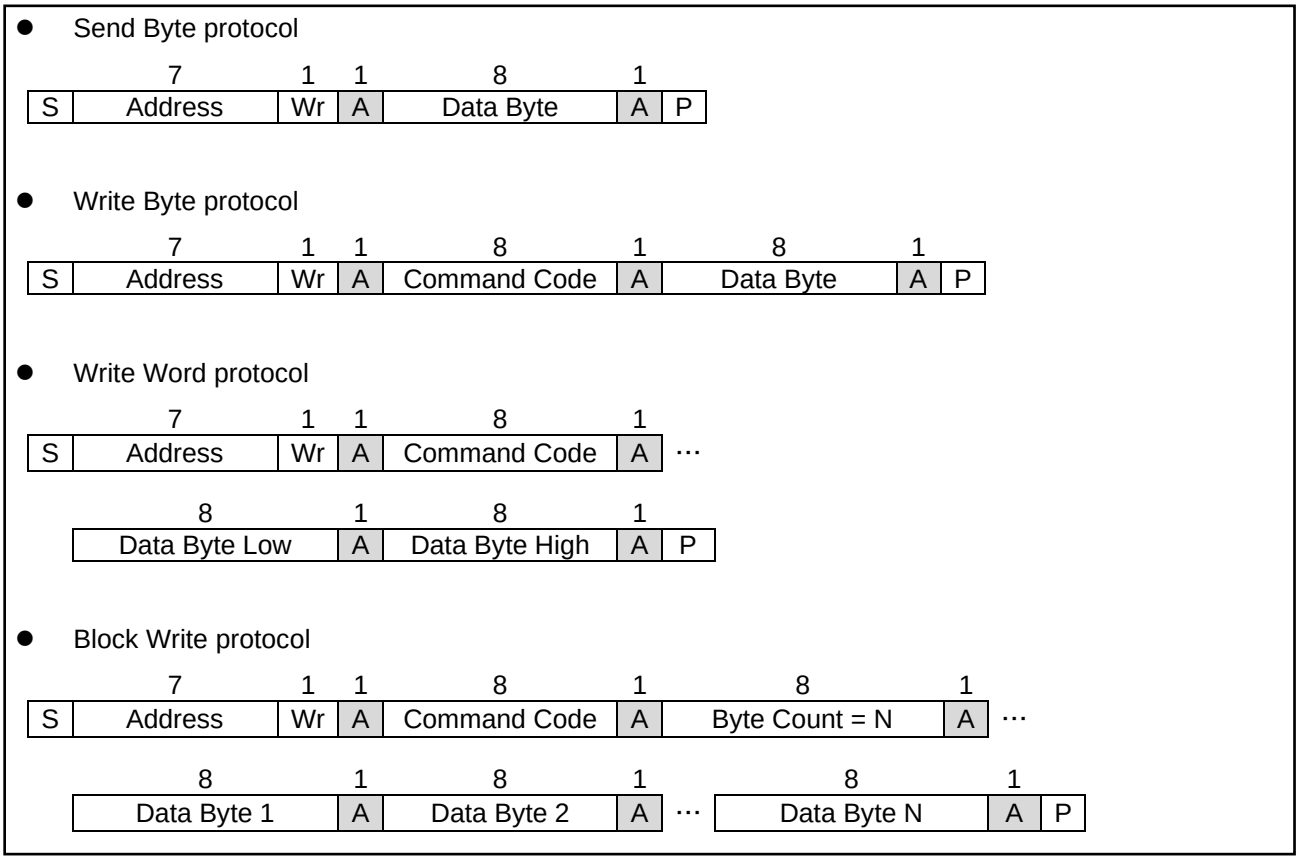
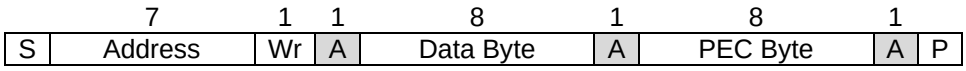
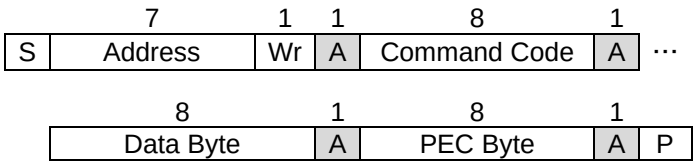


Figure 3 Transactional of Write protocol

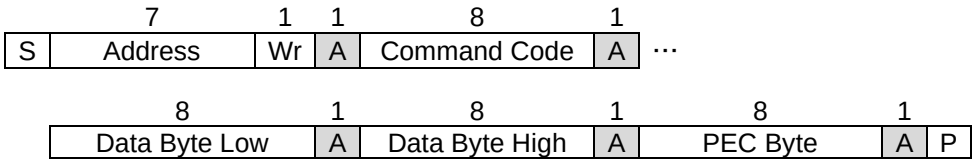
- Send Byte protocol with PEC



- Write Byte protocol with PEC



- Write Word protocol with PEC



- Block Write protocol with PEC

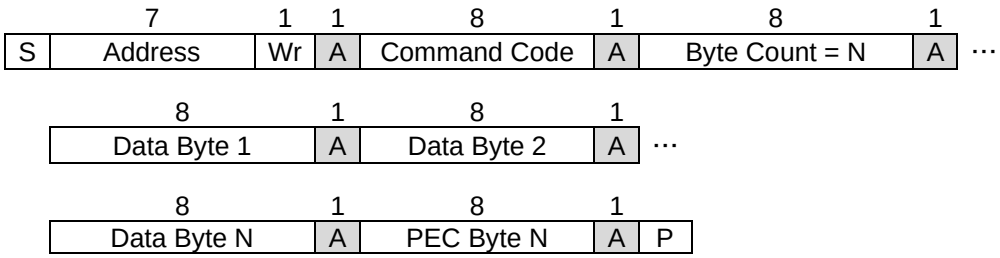
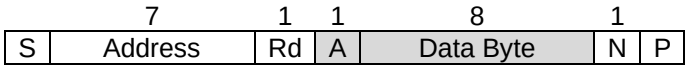
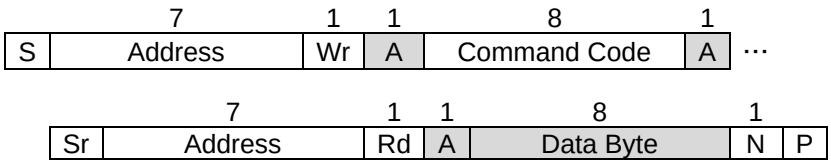


Figure 4 Transactional of Write protocol with PEC

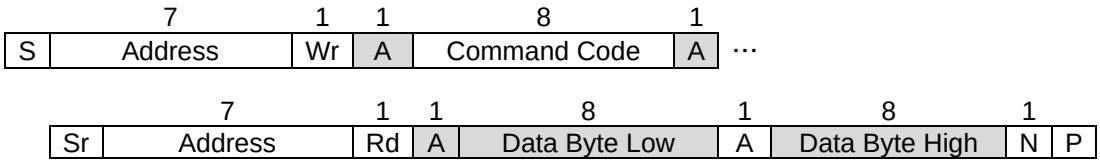
- Receive Byte protocol



- Read Byte protocol



- Read Word protocol



- Block Read protocol

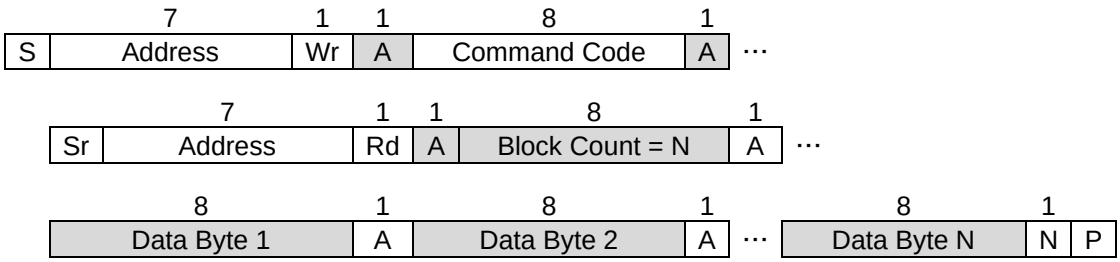


Figure 5 Transactional of Read protocol

- Receive Byte protocol with PEC

7	1	1	8	1	8	1
S	Address	Rd	A	Data Byte	A	PEC Byte	N	P

- Read Byte protocol with PEC

7	1	1	8	1	
S	Address	Wr	A	Command Code	A	…

7	1	1	8	1	8	1	
Sr	Address	Rd	A	Data Byte	A	PEC Byte	N	P

- Read Word protocol with PEC

7	1	1	8	1	
S	Address	Wr	A	Command Code	A	…

7	1	1	8	1	
Sr	Address	Rd	A	Data Byte Low	A	…

8	1	8	1	
Data Byte High	A	PEC Byte	N	P

- Block Read protocol with PEC

7	1	1	8	1	
S	Address	Wr	A	Command Code	A	…

7	1	1	8	1	8	1	
Sr	Address	Rd	A	Block Count = N	A	Data Byte 1	A	…

8	1		8	1	8	1	
Data Byte 2	A	…	Data Byte N	A	PEC Byte	N	P

Figure 6 Transactional of Read protocol with PEC

- 7-bit address format

	n(n=1～)						
1	7	1	1	8	1	1	1
S	SLA(Slave Address)	R/ W#	A	DATA(Command Code & Data)	A	A/ A#	P

n: Number of transfer data bytes

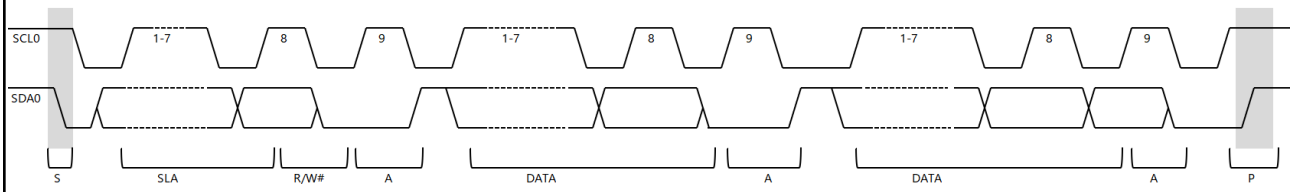
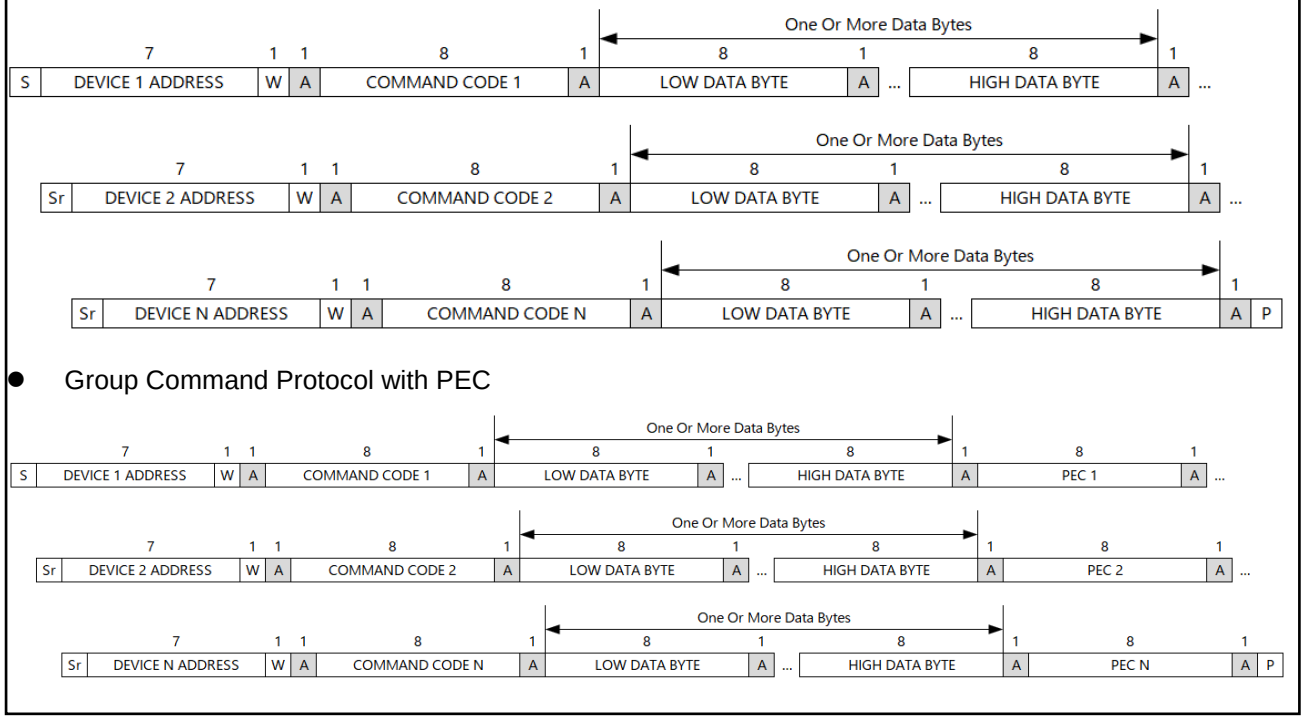


Figure 7 Standard Command Protocol

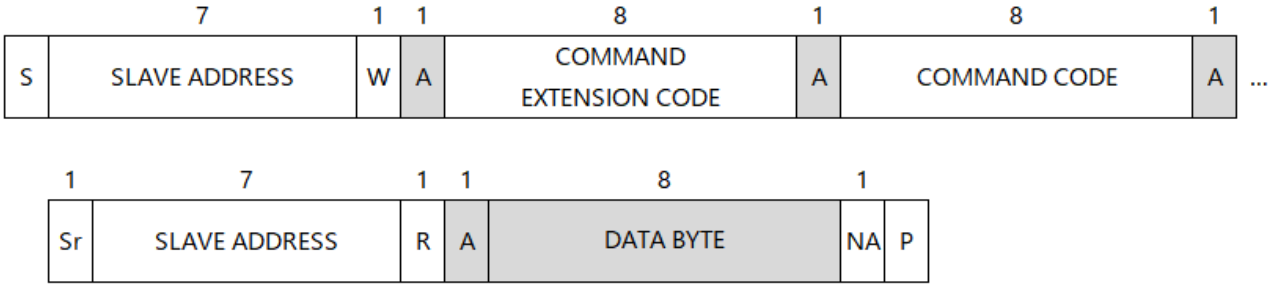
- Group command protocol



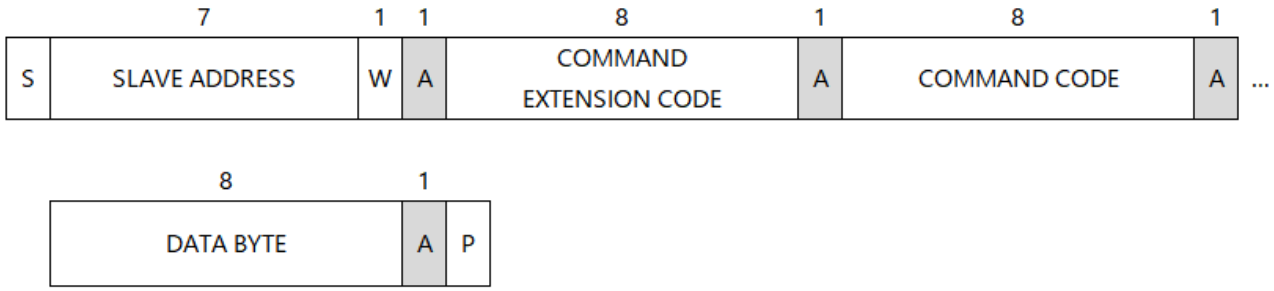
- Group Command Protocol with PEC

Figure 8 Group Command Protocol

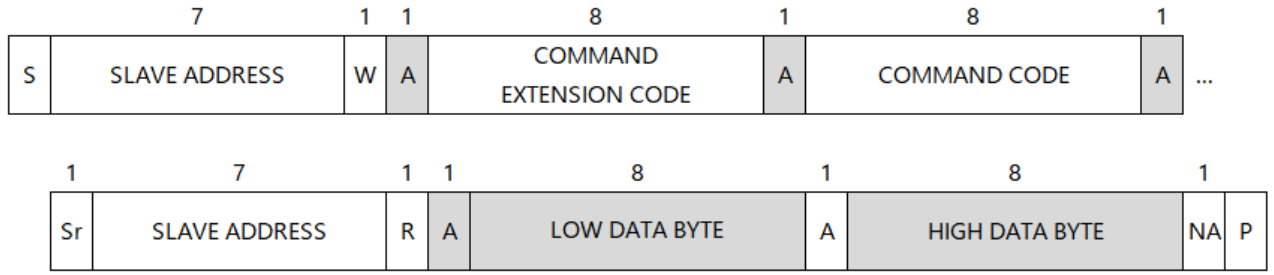
- Extended command Read Byte protocol



- Extended command Write Byte protocol



- Extended command Read Word protocol



- Extended command Write Word protocol

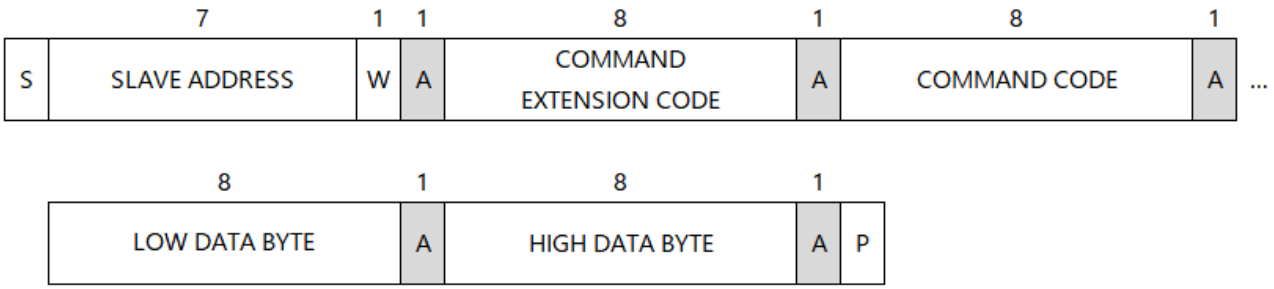
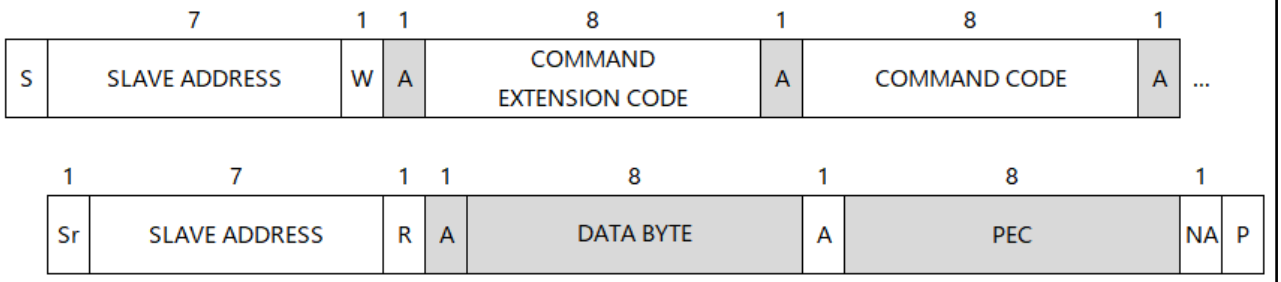
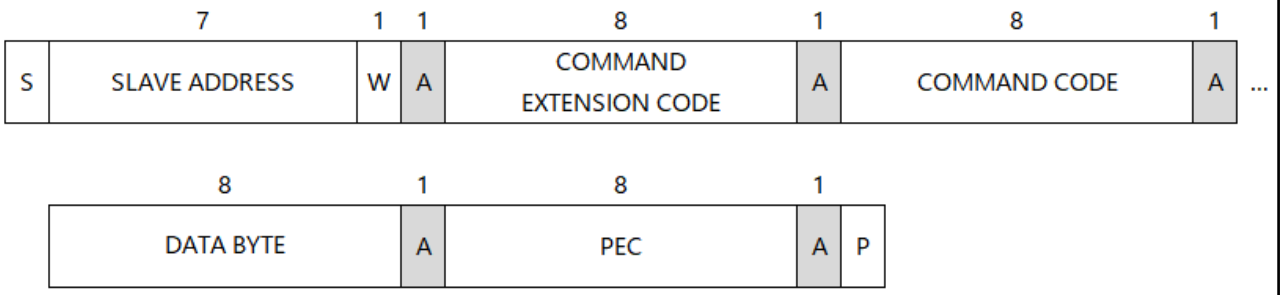


Figure 9 Extended command protocol

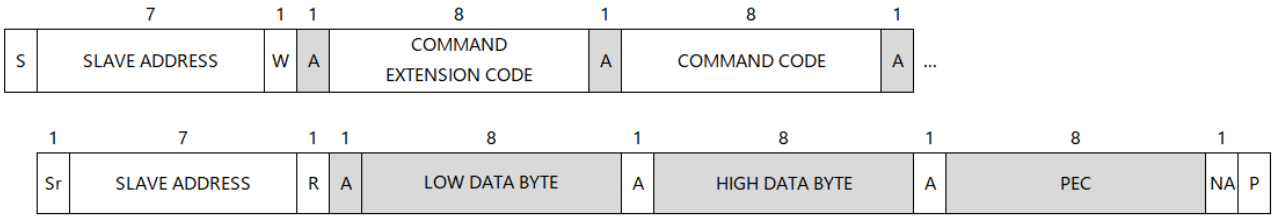
- Extended command Read Byte protocol with PEC



- Extended command Write Byte protocol with PEC



- Extended command Read Word protocol with PEC



- Extended command Write Word protocol with PEC

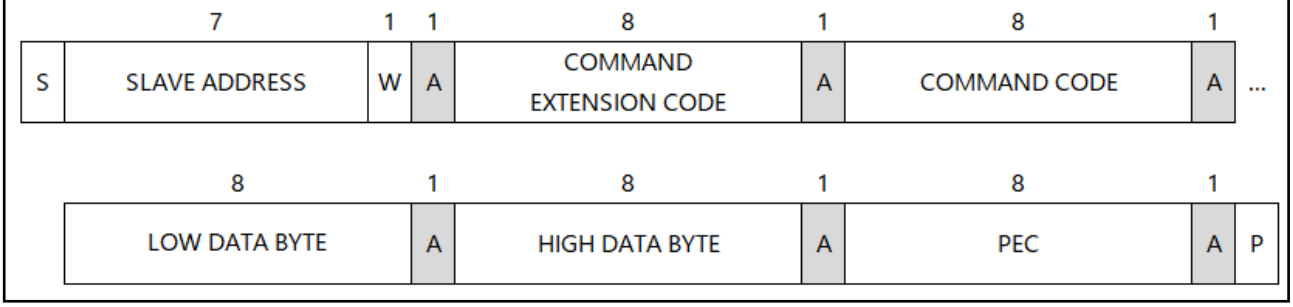
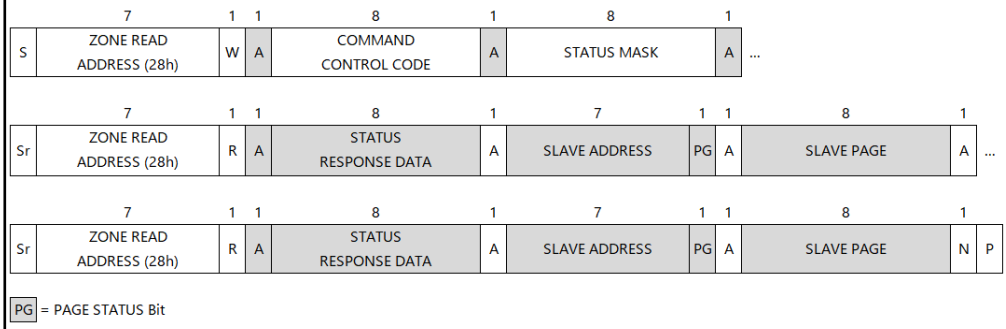
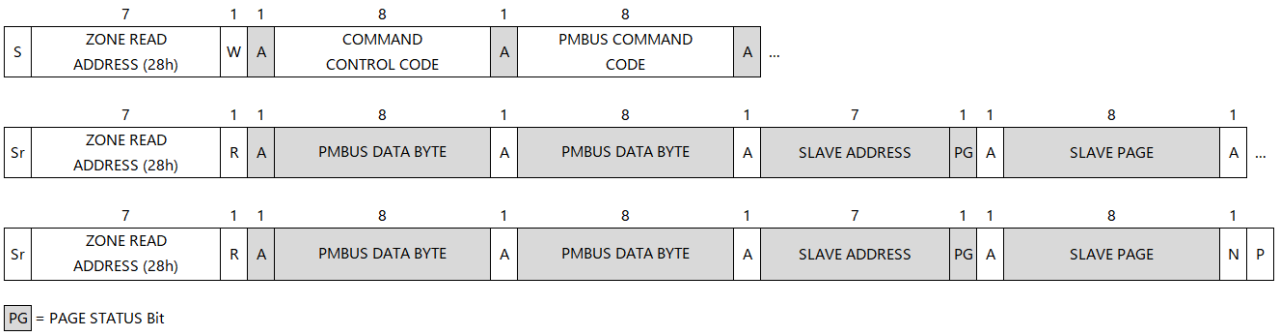


Figure 10 Extended command protocol with PEC

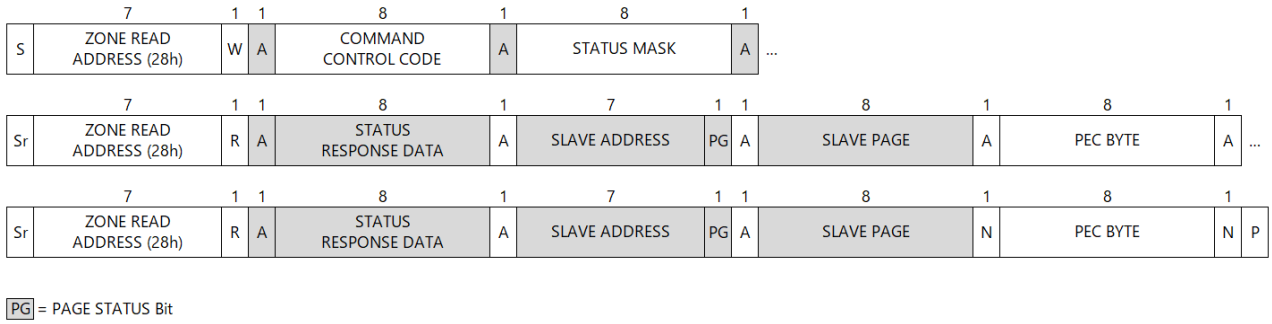
- Zone Read Command Protocol with Status Response Data



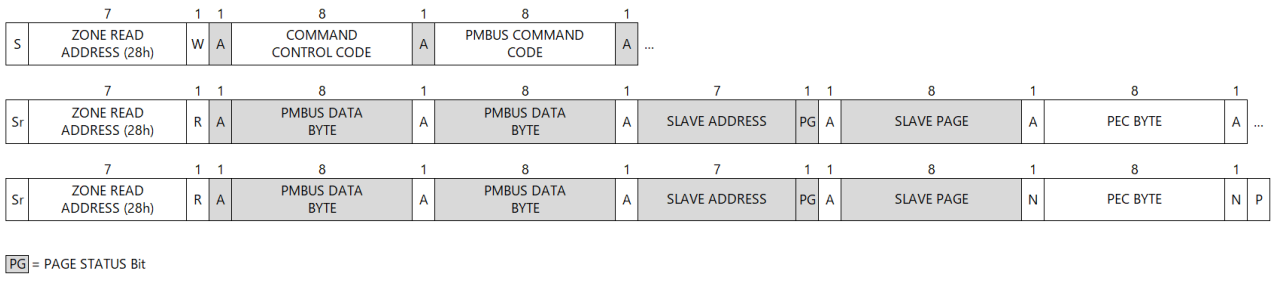
- Zone Read Command Protocol with PMBus Command Code



- Zone read command protocol with PEC and status response data



- Zone Read Command Protocol with PEC, PMBus Command Code



- 2-byte data, zone write command protocol with PMBus command code

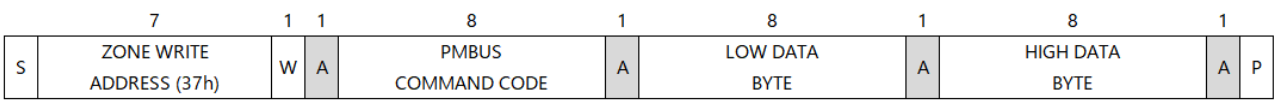


Figure 11 Zone Command Protocol

2.2 PMBus Command

PMBus has a predefined command to facilitate interconnection between devices. Commands are also prepared for future extensions and user-defined commands for easy customization. Table 3 lists PMBus commands and the transaction types associated with the commands.
This application note uses the command code 01h, 02h, 03h, 81h, 8Bh, 8Ch, 90h, 95h among the commands shown in Table 3.

Table 3 PMBus Commands List

: Commands Used in this Application Note

Command Code	Command Name	Transaction Type: Writing Data	Transaction Type: Reading Data	Number Of Data Bytes
00h	PAGE	Write Byte	Read Byte	1
01h	OPERATION	Write Byte	Read Byte	1
02h	ON_OFF_CONFIG	Write Byte	Read Byte	1
03h	CLEAR_FAULTS	Send Byte	N/A	0
04h	PHASE	Write Byte	Read Byte	1
05h	PAGE_PLUS_WRITE	Block Write	N/A	Variable
06h	PAGE_PLUS_READ	N/A	Block Write – Block Read Process Call	Variable
07h	ZONE_CONFIG	Write Word	Read Word	2
08h	ZONE_ACTIVE	Write Word	Read Word	2
09h	Reserved			
0Ah	Reserved			
0Bh	Reserved			
0Ch	Reserved			
0Dh	Reserved			
0Eh	Reserved			
0Fh	Reserved			
10h	WRITE_PROTECT	Write Byte	Read Byte	1
11h	STORE_DEFAULT_ALL	Send Byte	N/A	0
12h	RESTORE_DEFAULT_ALL	Send Byte	N/A	0
13h	STORE_DEFAULT_CODE	Write Byte	N/A	1
14h	RESTORE_DEFAULT_CODE	Write Byte	N/A	1
15h	STORE_USER_ALL	Send Byte	N/A	0
16h	RESTORE_USER_ALL	Send Byte	N/A	0
17h	STORE_USER_CODE	Write Byte	N/A	1
18h	RESTORE_USER_CODE	Write Byte	N/A	1
19h	CAPABILITY	N/A	Read Byte	1
1Ah	QUERY	N/A	Block Write- Block Read Process Call	1
1Bh	SMBALERT_MASK	Write Word	Block Write- Block Read Process Call	2

1Ch	Reserved			
1Dh	Reserved			
1Eh	Reserved			
1Fh	Reserved			
20h	VOUT_MODE	Write Byte	Read Byte	1
21h	VOUT_COMMAND	Write Word	Read Word	2
22h	VOUT_TRIM	Write Word	Read Word	2
23h	VOUT_CAL_OFFSET	Write Word	Read Word	2
24h	VOUT_MAX	Write Word	Read Word	2
25h	VOUT_MARGIN_HIGH	Write Word	Read Word	2
26h	VOUT_MARGIN_LOW	Write Word	Read Word	2
27h	VOUT_TRANSITION_RATE	Write Word	Read Word	2
28h	VOUT_DROOP	Write Word	Read Word	2
29h	VOUT_SCALE_LOOP	Write Word	Read Word	2
2Ah	VOUT_SCALE_MONITOR	Write Word	Read Word	2
2Bh	VOUT_MIN	Write Word	Read Word	2
2Ch	Reserved			
2Dh	Reserved			
2Eh	Reserved			
2Fh	Reserved			
30h	COEFFICIENTS	N/A	Block Write- Block Read Process Call	5
31h	POUT_MAX	Write Word	Read Word	2
32h	MAX_DUTY	Write Word	Read Word	2
33h	FREQUENCY_SWITCH	Write Word	Read Word	2
34h	POWER_MODE	Write Byte	Read Byte	1
35h	VIN_ON	Write Word	Read Word	2
36h	VIN_OFF	Write Word	Read Word	2
37h	INTERLEAVE	Write Word	Read Word	2
38h	IOUT_CAL_GAIN	Write Word	Read Word	2
39h	IOUT_CAL_OFFSET	Write Word	Read Word	2
3Ah	FAN_CONFIG_1_2	Write Byte	Read Byte	1
3Bh	FAN_COMMAND_1	Write Word	Read Word	2
3Ch	FAN_COMMAND_2	Write Word	Read Word	2
3Dh	FAN_CONFIG_3_4	Write Byte	Read Byte	1
3Eh	FAN_COMMAND_3	Write Word	Read Word	2
3Fh	FAN_COMMAND_4	Write Word	Read Word	2
40h	VOUT_OV_FAULT_LIMIT	Write Word	Read Word	2

41h	VOUT_OV_FAULT_RESPONSE	Write Byte	Read Byte	1
42h	VOUT_OV_WARN_LIMIT	Write Word	Read Word	2
43h	VOUT_UV_WARN_LIMIT	Write Word	Read Word	2
44h	VOUT_UV_FAULT_LIMIT	Write Word	Read Word	2
45h	VOUT_UV_FAULT_RESPONSE	Write Byte	Read Byte	1
46h	IOUT_OC_FAULT_LIMIT	Write Word	Read Word	2
47h	IOUT_OC_FAULT_RESPONSE	Write Byte	Read Byte	1
48h	IOUT_OC_LV_FAULT_LIMIT	Write Word	Read Word	2
49h	IOUT_OC_LV_FAULT_RESPONSE	Write Byte	Read Byte	1
4Ah	IOUT_OC_WARN_LIMIT	Write Word	Read Word	2
4Bh	IOUT_UC_FAULT_LIMIT	Write Word	Read Word	2
4Ch	IOUT_UC_FAULT_RESPONSE	Write Byte	Read Byte	1
4Dh	Reserved			
4Eh	Reserved			
4Fh	OT_FAULT_LIMIT	Write Word	Read Word	2
50h	OT_FAULT_RESPONSE	Write Byte	Read Byte	1
51h	OT_WARN_LIMIT	Write Word	Read Word	2
52h	UT_WARN_LIMIT	Write Word	Read Word	2
53h	UT_FAULT_LIMIT	Write Word	Read Word	2
54h	UT_FAULT_RESPONSE	Write Byte	Read Byte	1
55h	VIN_OV_FAULT_LIMIT	Write Word	Read Word	2
56h	VIN_OV_FAULT_RESPONSE	Write Byte	Read Byte	1
57h	VIN_OV_WARN_LIMIT	Write Word	Read Word	2
58h	VIN_UV_WARN_LIMIT	Write Word	Read Word	2
59h	VIN_UV_FAULT_LIMIT	Write Word	Read Word	2
5Ah	VIN_UV_FAULT_RESPONSE	Write Byte	Read Byte	1
5Bh	IIN_OC_FAULT_LIMIT	Write Word	Read Word	2
5Ch	IIN_OC_FAULT_RESPONSE	Write Byte	Read Byte	1
5Dh	IIN_OC_WARN_LIMIT	Write Word	Read Word	2
5Eh	POWER_GOOD_ON	Write Word	Read Word	2
5Fh	POWER_GOOD_OFF	Write Word	Read Word	2
60h	TON_DELAY	Write Word	Read Word	2
61h	TON_RISE	Write Word	Read Word	2
62h	TON_MAX_FAULT_LIMIT	Write Word	Read Word	2
63h	TON_MAX_FAULT_RESPONSE	Write Byte	Read Byte	1
64h	TOFF_DELAY	Write Word	Read Word	2
65h	TOFF_FALL	Write Word	Read Word	2

66h	TOFF_MAX_WARN_LIMIT	Write Word	Read Word	2
67h	Reserved (Was Used In Revision 1.0)			
68h	POUT_OP_FAULT_LIMIT	Write Word	Read Word	2
69h	POUT_OP_FAULT_RESPONSE	Write Byte	Read Byte	1
6Ah	POUT_OP_WARN_LIMIT	Write Word	Read Word	2
6Bh	PIN_OP_WARN_LIMIT	Write Word	Read Word	2
6Ch	Reserved			
6Dh	Reserved			
6Eh	Reserved			
6Fh	Reserved			
70h	Reserved (Test Input Fuse A)			
71h	Reserved (Test Input Fuse B)			
72h	Reserved (Test Input OR-ing A)			
73h	Reserved (Test Input OR-ing B)			
74h	Reserved (Test Output OR-ing)			
75h	Reserved			
76h	Reserved			
77h	Reserved			
78h	STATUS_BYTE	Write Byte	Read Byte	1
79h	STATUS_WORD	Write Word	Read Word	2
7Ah	STATUS_VOUT	Write Byte	Read Byte	1
7Bh	STATUS_IOUT	Write Byte	Read Byte	1
7Ch	STATUS_INPUT	Write Byte	Read Byte	1
7Dh	STATUS_TEMPERATURE	Write Byte	Read Byte	1
7Eh	STATUS_CML	Write Byte	Read Byte	1
7Fh	STATUS_OTHER	Write Byte	Read Byte	1
80h	STATUS_MFR_SPECIFIC	Write Byte	Read Byte	1
81h	STATUS_FANS_1_2	Write Byte	Read Byte	1
82h	STATUS_FANS_3_4	Write Byte	Read Byte	1
83h	READ_KWH_IN	N/A	Read 32	4
84h	READ_KWH_OUT	N/A	Read 32	4
85h	READ_KWH_CONFIG	Write Word	Read Word	2
86h	READ_EIN	N/A	Block Read	5
87h	READ_EOUT	N/A	Block Read	5
88h	READ_VIN	N/A	Read Word	2
89h	READ_IIN	N/A	Read Word	2
8Ah	READ_VCAP	N/A	Read Word	2

8Bh	READ_VOUT	N/A	Read Word	2
8Ch	READ_IOUT	N/A	Read Word	2
8Dh	READ_TEMPERATURE_1	N/A	Read Word	2
8Eh	READ_TEMPERATURE_2	N/A	Read Word	2
8Fh	READ_TEMPERATURE_3	N/A	Read Word	2
90h	READ_FAN_SPEED_1	N/A	Read Word	2
91h	READ_FAN_SPEED_2	N/A	Read Word	2
92h	READ_FAN_SPEED_3	N/A	Read Word	2
93h	READ_FAN_SPEED_4	N/A	Read Word	2
94h	READ_DUTY_CYCLE	N/A	Read Word	2
95h	READ_FREQUENCY	N/A	Read Word	2
96h	READ_POUT	N/A	Read Word	2
97h	READ_PIN	N/A	Read Word	2
98h	PMBUS_REVISION	N/A	Read Byte	1
99h	MFR_ID	Block Write	Block Read	Variable
9Ah	MFR_MODEL	Block Write	Block Read	Variable
9Bh	MFR_REVISION	Block Write	Block Read	Variable
9Ch	MFR_LOCATION	Block Write	Block Read	Variable
9Dh	MFR_DATE	Block Write	Block Read	Variable
9Eh	MFR_SERIAL	Block Write	Block Read	Variable
9Fh	APP_PROFILE_SUPPORT	N/A	Block Read	Variable
A0h	MFR_VIN_MIN	N/A	Read Word	2
A1h	MFR_VIN_MAX	N/A	Read Word	2
A2h	MFR_IIN_MAX	N/A	Read Word	2
A3h	MFR_PIN_MAX	N/A	Read Word	2
A4h	MFR_VOUT_MIN	N/A	Read Word	2
A5h	MFR_VOUT_MAX	N/A	Read Word	2
A6h	MFR_IOUT_MAX	N/A	Read Word	2
A7h	MFR_POUT_MAX	N/A	Read Word	2
A8h	MFR_TAMBIENT_MAX	N/A	Read Word	2
A9h	MFR_TAMBIENT_MIN	N/A	Read Word	2
AAh	MFR_EFFICIENCY_LL	N/A	Block Read	14
ABh	MFR_EFFICIENCY_HL	N/A	Block Read	14
ACh	MFR_PIN_ACCURACY	N/A	Read Byte	1
ADh	IC_DEVICE_ID	N/A	Block Read	Variable
AEh	IC_DEVICE_REV	N/A	Block Read	Variable
AFh	Reserved			

B0h	USER_DATA_00	Block Write	Block Read	Variable
B1h	USER_DATA_01	Block Write	Block Read	Variable
B2h	USER_DATA_02	Block Write	Block Read	Variable
B3h	USER_DATA_03	Block Write	Block Read	Variable
B4h	USER_DATA_04	Block Write	Block Read	Variable
B5h	USER_DATA_05	Block Write	Block Read	Variable
B6h	USER_DATA_06	Block Write	Block Read	Variable
B7h	USER_DATA_07	Block Write	Block Read	Variable
B8h	USER_DATA_08	Block Write	Block Read	Variable
B9h	USER_DATA_09	Block Write	Block Read	Variable
BAh	USER_DATA_10	Block Write	Block Read	Variable
BBh	USER_DATA_11	Block Write	Block Read	Variable
BCh	USER_DATA_12	Block Write	Block Read	Variable
BDh	USER_DATA_13	Block Write	Block Read	Variable
BEh	USER_DATA_14	Block Write	Block Read	Variable
BFh	USER_DATA_15	Block Write	Block Read	Variable
C0h	MFR_MAX_TEMP_1	Write Word	Read Word	2
C1h	MFR_MAX_TEMP_2	Write Word	Read Word	2
C2h	MFR_MAX_TEMP_3	Write Word	Read Word	2
C3h	Reserved			
C4h	MFR_SPECIFIC_C4	Mfr. Defined	Mfr. Defined	Mfr. Defined
C5h	MFR_SPECIFIC_C5	Mfr. Defined	Mfr. Defined	Mfr. Defined
C6h	MFR_SPECIFIC_C6	Mfr. Defined	Mfr. Defined	Mfr. Defined
C7h	MFR_SPECIFIC_C7	Mfr. Defined	Mfr. Defined	Mfr. Defined
C8h	MFR_SPECIFIC_C8	Mfr. Defined	Mfr. Defined	Mfr. Defined
C9h	MFR_SPECIFIC_C9	Mfr. Defined	Mfr. Defined	Mfr. Defined
CAh	MFR_SPECIFIC_CA	Mfr. Defined	Mfr. Defined	Mfr. Defined
CBh	MFR_SPECIFIC_CB	Mfr. Defined	Mfr. Defined	Mfr. Defined
CCh	MFR_SPECIFIC_CC	Mfr. Defined	Mfr. Defined	Mfr. Defined
CDh	MFR_SPECIFIC_CD	Mfr. Defined	Mfr. Defined	Mfr. Defined
CEh	MFR_SPECIFIC_CE	Mfr. Defined	Mfr. Defined	Mfr. Defined
CFh	MFR_SPECIFIC_CF	Mfr. Defined	Mfr. Defined	Mfr. Defined
D0h	MFR_SPECIFIC_D0	Mfr. Defined	Mfr. Defined	Mfr. Defined
D1h	MFR_SPECIFIC_D1	Mfr. Defined	Mfr. Defined	Mfr. Defined

D2h	MFR_SPECIFIC_D2	Mfr. Defined	Mfr. Defined	Mfr. Defined
D3h	MFR_SPECIFIC_D3	Mfr. Defined	Mfr. Defined	Mfr. Defined
D4h	MFR_SPECIFIC_D4	Mfr. Defined	Mfr. Defined	Mfr. Defined
D5h	MFR_SPECIFIC_D5	Mfr. Defined	Mfr. Defined	Mfr. Defined
D6h	MFR_SPECIFIC_D6	Mfr. Defined	Mfr. Defined	Mfr. Defined
D7h	MFR_SPECIFIC_D7	Mfr. Defined	Mfr. Defined	Mfr. Defined
D8h	MFR_SPECIFIC_D8	Mfr. Defined	Mfr. Defined	Mfr. Defined
D9h	MFR_SPECIFIC_D9	Mfr. Defined	Mfr. Defined	Mfr. Defined
DAh	MFR_SPECIFIC_DA	Mfr. Defined	Mfr. Defined	Mfr. Defined
DBh	MFR_SPECIFIC_DB	Mfr. Defined	Mfr. Defined	Mfr. Defined
DCh	MFR_SPECIFIC_DC	Mfr. Defined	Mfr. Defined	Mfr. Defined
DDh	MFR_SPECIFIC_DD	Mfr. Defined	Mfr. Defined	Mfr. Defined
DEh	MFR_SPECIFIC_DE	Mfr. Defined	Mfr. Defined	Mfr. Defined
DFh	MFR_SPECIFIC_DF	Mfr. Defined	Mfr. Defined	Mfr. Defined
E0h	MFR_SPECIFIC_E0	Mfr. Defined	Mfr. Defined	Mfr. Defined
E1h	MFR_SPECIFIC_E1	Mfr. Defined	Mfr. Defined	Mfr. Defined
E2h	MFR_SPECIFIC_E2	Mfr. Defined	Mfr. Defined	Mfr. Defined
E3h	MFR_SPECIFIC_E3	Mfr. Defined	Mfr. Defined	Mfr. Defined
E4h	MFR_SPECIFIC_E4	Mfr. Defined	Mfr. Defined	Mfr. Defined
E5h	MFR_SPECIFIC_E5	Mfr. Defined	Mfr. Defined	Mfr. Defined
E6h	MFR_SPECIFIC_E6	Mfr. Defined	Mfr. Defined	Mfr. Defined
E7h	MFR_SPECIFIC_E7	Mfr. Defined	Mfr. Defined	Mfr. Defined
E8h	MFR_SPECIFIC_E8	Mfr. Defined	Mfr. Defined	Mfr. Defined
E9h	MFR_SPECIFIC_E9	Mfr. Defined	Mfr. Defined	Mfr. Defined
EAh	MFR_SPECIFIC_EA	Mfr. Defined	Mfr. Defined	Mfr. Defined
EBh	MFR_SPECIFIC_EB	Mfr. Defined	Mfr. Defined	Mfr. Defined
ECh	MFR_SPECIFIC_EC	Mfr. Defined	Mfr. Defined	Mfr. Defined
EDh	MFR_SPECIFIC_ED	Mfr. Defined	Mfr. Defined	Mfr. Defined
EEh	MFR_SPECIFIC_EE	Mfr. Defined	Mfr. Defined	Mfr. Defined
EFh	MFR_SPECIFIC_EF	Mfr. Defined	Mfr. Defined	Mfr. Defined

F0h	MFR_SPECIFIC_F0	Mfr. Defined	Mfr. Defined	Mfr. Defined
F1h	MFR_SPECIFIC_F1	Mfr. Defined	Mfr. Defined	Mfr. Defined
F2h	MFR_SPECIFIC_F2	Mfr. Defined	Mfr. Defined	Mfr. Defined
F3h	MFR_SPECIFIC_F3	Mfr. Defined	Mfr. Defined	Mfr. Defined
F4h	MFR_SPECIFIC_F4	Mfr. Defined	Mfr. Defined	Mfr. Defined
F5h	MFR_SPECIFIC_F5	Mfr. Defined	Mfr. Defined	Mfr. Defined
F6h	MFR_SPECIFIC_F6	Mfr. Defined	Mfr. Defined	Mfr. Defined
F7h	MFR_SPECIFIC_F7	Mfr. Defined	Mfr. Defined	Mfr. Defined
F8h	MFR_SPECIFIC_F8	Mfr. Defined	Mfr. Defined	Mfr. Defined
F9h	MFR_SPECIFIC_F9	Mfr. Defined	Mfr. Defined	Mfr. Defined
FAh	MFR_SPECIFIC_FA	Mfr. Defined	Mfr. Defined	Mfr. Defined
FBh	MFR_SPECIFIC_FB	Mfr. Defined	Mfr. Defined	Mfr. Defined
FCh	MFR_SPECIFIC_FC	Mfr. Defined	Mfr. Defined	Mfr. Defined
FDh	MFR_SPECIFIC_FD	Mfr. Defined	Mfr. Defined	Mfr. Defined
FEh	MFR_SPECIFIC_COMMAND EXT	Extended Command	Extended Command	Mfr Defined
FFh	PMBUS_COMMAND_EXT	Extended Command	Extended Command	Mfr Defined

3. Hardware Description

The boards and parts used in this application note are shown in Table 4.

Table 4 Boards Used and Parts List

No.	Board and Part Name	Model	Master/ Slave	Remarks
1	CPU Board	RTK0EMXE70C00000BJ	Master & Slave	Renesas CPU evaluation board with R5F526TFCDFP included in MCK-RX26T(RTK0EMXE70S00020BJ)
		RTK0EMA330C00000BJ	Slave	Renesas CPU evaluation board with R7FA6T3BB3CFM included in MCK-RA6T3(RTK0EMA330S00020BJ)
2	Inverter Board	RTK0EM0000B12020BJ	Slave	Renesas Motor drive evaluation inverter board Included in MCK-RX26T(RTK0EMXE70S00020BJ), OR MCK-RA6T3(RTK0EMA330S00020BJ)
3	USB-UART Conversion board	Pmod-USBUART	Master	DIGILENT's USB-UART converter that connects USB of PC to SCI of MCU
4	Motor	R42BLD30L3	Slave	Renesas MOONS'-made brushless DC motor (rated 36V,1.67A) included in MCK-RX26T(RTK0EMXE70S00020BJ), OR MCK-RA6T3(RTK0EMA330S00020BJ)

3.1 Hardware configuration

Figure 12 shows the hardware configuration used in this application note. Figure 13 to Figure 16 show the external views and schematic specifications of each board. The board information is written in abbreviated form. For details such as the latest specifications, refer to the user's manuals to various boards in the reference material of summary.

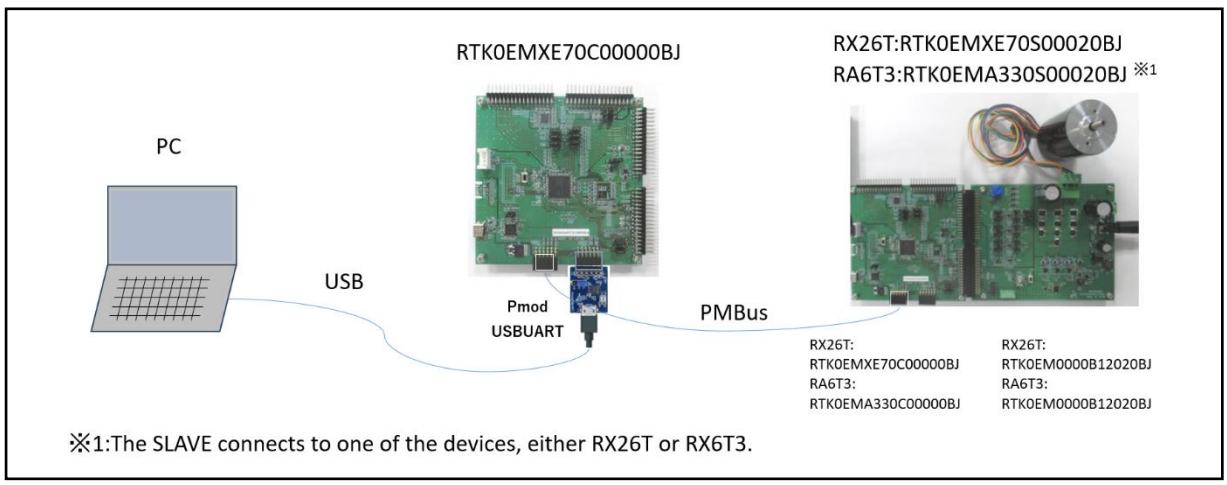


Figure 12 Hardware configuration diagram

item		Specification
Product name		CPU Board
Board part No.		RTK0EMXE70C00000BJ
Compatible inverter board		RTK0EM0000B12020BJ
External view		 Note: The actual product may differ from this photo.
Mounted MCU	Product group	RX26T group
	Product No.	R5F526TFCDFP
	CPU maximum operating frequency	120MHz
	Bit count	32 bit
	Package / Pin count	LFQFP / 100 pin
	ROM	512KB
MCU input clock		10MHz (Generate with external crystal oscillator)
Power supply		DC 5V,3.3V (selectable with jumper switch) Select one way automatically from the below • Power is supplied from compatible inverter board • Power is supplied from USB connector
Debugger		E2OB (Onboard debugger circuit)
Connector		• Inverter board connector • USB connector for E2 OB • SCI connector for Renesas Motor Workbench communication • Through hole for CAN communication • Through hole for SPI communication • PMOD connectors
Switch		MCU reset switch
LED		User-controllable LED x4, Power LED x1

Figure 13 RX26T CPU Board used in PMBus Master

Item	Specification	
Kit product name	MCK-RX26T	
Kit product No.	RTK0EMXE70S00020BJ	
Kit configuration	Inverter Board	RTK0EM0000B12020BJ
	CPU Board	RTK0EMXE70C00000BJ
	Communication board	RTK0EMXC90Z00000BJ
	Brushless DC Motor	R42BLD30L3 (MOONS') Rated voltage : 36[V] Rated current : 1.67[A]
Isolation	Inverter board - CPU board : Non-isolated Communication board – CPU board : isolated (up to $1kV_{RMS}$)	
External view	 Note: The actual product may differ from this photo.	
Board size	Inverter board : 133 mm (W) x 109 mm (L) CPU board : 109 mm (W) x 109 mm (L) Comunication board : 89mm(W) × 52mm(L)	
Operating temperature	Room temperature	
Operating humidity	No condensation allowed	
EMC Directive	EN61326-1:2021 EMI：Class A EMS：Basic Electromagnetic environment	

Figure 14 Flexible Motor Control Kit for RX26T Groups used in PMBus Slave

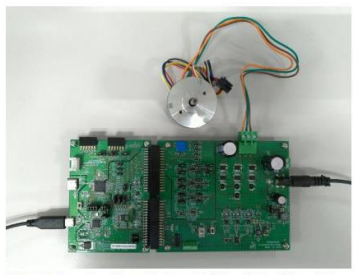
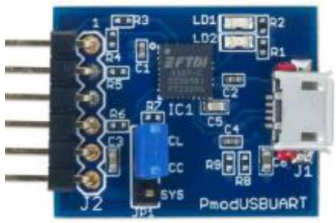
Item	Specification	
Kit product name	MCK-RA6T3	
Kit product No.	RTK0EMA330S00020BJ	
Kit configuration	Inverter Board	RTK0EM0000B12020BJ
	CPU Board	RTK0EMA330C00000BJ
	Brushless DC Motor	R42BLD30L3 (MOONS') Rated voltage : 36[V] Rated current : 1.67[A]
Isolation	Inverter board - CPU board : Non-isolated	
External view	 Note: The actual product may differ from this photo.	
Board size	Inverter board : 133 mm (W) x 109 mm (L) CPU board : 109 mm (W) x 109 mm (L)	
Operating temperature	Room temperature	
Operating humidity	No condensation allowed	
EMC Directive	EN61326-1:2021 EMI：Class A EMS：Basic Electromagnetic environment	

Figure 15 Flexible Motor Control Kit for RA6T3 Groups used in PMBus Slave



Pmod USBUART

USB to UART Interface

Features

- USB to serial UART interface
- Micro USB connector
- Option to power the system board through the FTDI chip
- 6-pin Pmod connector with UART interface
- Follows the Digilent Pmod Interface Specification Type 4

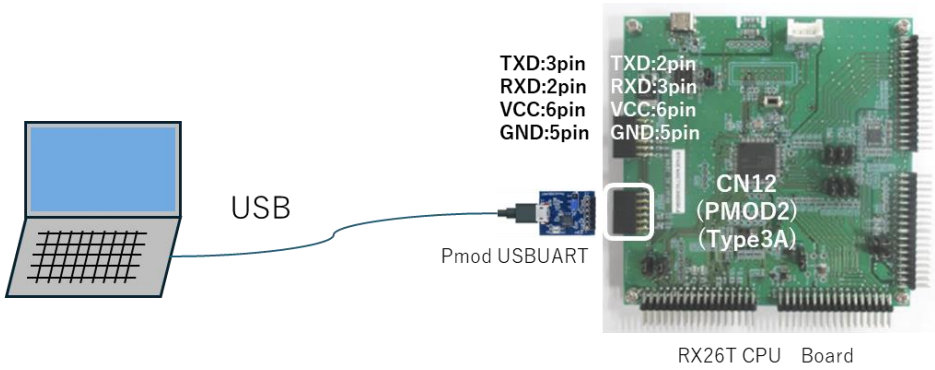
Electrical	
Bus	UART
Specification Version	1.2.0
Logic Level	3.3V
Physical	
Width	1.0 in (2.54 cm)
Length	0.80 in (2.03 cm)

Figure 16 DIGILENT's Pmod USBUART

3.2 Hardware Setup

Connect RX26T CPU Board of PMBus Master and PC, and RX26T CPU Board of PMBus Master and RX26T CPU Board or RA6T3 CPU Board of PMBus Slave as shown in Figure 12 Hardware Configuration Diagram. Figure 17 shows the board terminal connections. For details on how to handle the flexible motor control kit, such as connection with motor, refer to the user's manual to the flexible motor control kit described in the reference material of summary.

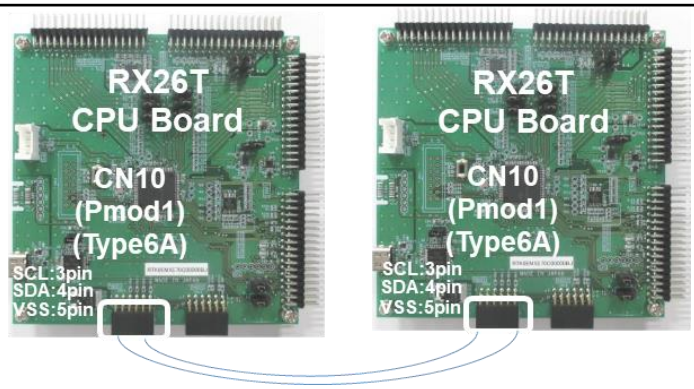
① Connecting PC to PMBus Master's RX26T CPU Board



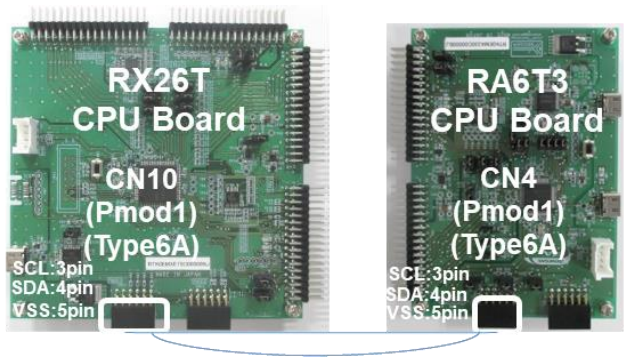
Signal Name	Pmod USBUART	RX26T CPU board
TXD	3pin	CN12-2pin
RXD	2pin	CN12-3pin
VCC	6pin	CN12-6pin
GND	5pin	CN12-5pin

② Connecting PMBus Master's RX26T to PMBus Slave's RX26T or RA6T3

Connect the RX26T CPU board with Pmod1 (Type6A).
SCL : Pmod1 3pin
SDA : Pmod1 4pin
VSS : Pmod1 5pin



Connect the RX26T CPU board and the RA6T3 CPU board with Pmod1 (Type6A).
SCL : Pmod1 3pin
SDA : Pmod1 4pin
VSS : Pmod1 5pin



【Note】Set the JP of RA6T3 to be PMOD (Type 6a).

Figure 17 Device Connection diagram

3.3 Configuration of MCU Function

The configuration for connecting MCU function is shown in Figure 18.

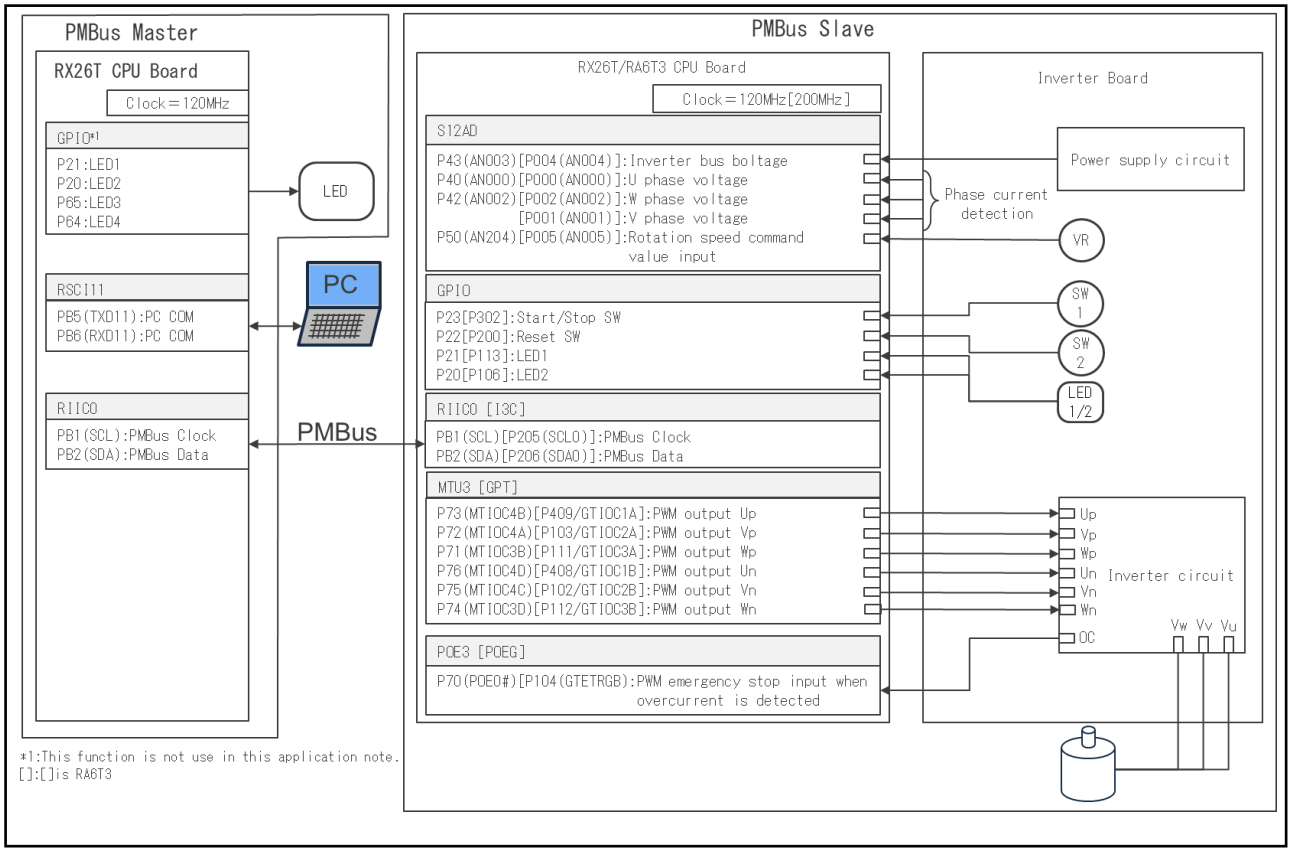


Figure 18 MCU Function Connection Configuration Diagram

3.4 MCU peripheral function

Table 5 lists shows the peripheral functions of RX26T group of PMBus Master, RX26T group of PMBus Slave, and RA6T3 group of PMBus Slave used in this system.

Table 5 Peripheral Functions List

Device	Peripheral functions	Usage
RX26T group (PMBus Master)	RSCI11	Use for communication with PC Terminal Soft.
	RIIC0	Use as the master for PMBus communication.
	TMR	Use unit0 for PMBus communication time-out monitoring.
RX26T group (PMBus Slave)	RIIC0	Use as a slave for PMBus communication.
	TMR	Use unit0 for PMBus communication time-out monitoring.
	S12AD	Use for the following functions: ・ Inverter bus voltage measurement ・ Rotation speed command value input (analog value) ・ U-phase current measurement ・ W-phase current measurement
	MTU3	Use ch3,4 for the following functions: ・ U-phase PWM output （ U_p / U_n ） ・ V-phase PWM output （ V_p / V_n ） ・ W-phase PWM output （ W_p / W_n ）
	POE3	Use for PWM emergency-stop when overcurrent is detected.
	CMT	Speed control interval timer
	IWDT	Independent Watchdog Timer
RA6T3 group (PMBus Slave)	I3C	Use as slave for PMBus communication.
	GPT	Use ch5 for PMBus communication time-out monitoring.
	ADC12	Use for the following functions: ・ Inverter bus voltage measurement ・ Rotation speed command value input (analog value) ・ U-phase current measurement ・ V-phase current measurement ・ W-phase current measurement
	GPT	Use ch1,2,3 for the following functions: ・ U-phase PWM output （ U_p / U_n ） ・ V-phase PWM output （ V_p / V_n ） ・ W-phase PWM output （ W_p / W_n ）
	POEG	Use for PWM emergency-stop when overcurrent is detected.
	AGTW	Speed control interval timer

3.5 Port interface

Table 6 lists port interfaces of RX26T group of PMBus Master, RX26T group and RA6T3 group of PMBus Slave.

Table 6 PMBus Master RX26T, PMBus Slave RX26T, RA6T3 port interfaces

Device	Peripheral functions	Port name	Usage
RX26T group (PMBus Master)	RSCI11	PB5_TXD11	Use for communication with PC Terminal Soft.
		PB6_RXD11	
	RIIC0	PB1_SCL	Use as the master for PMBus communication.
		PB2_SDA	
	GPIO	P21	LED1 control
		P20	LED2 control
		P65	LED3 control
		P64	LED4 control
RX26T group (PMBus Slave)	RIIC0	PB1_SCL	Use as slave for PMBus communication.
		PB2_SDA	
	S12AD	P43/ANI003	Inverter bus voltage measurement
		P50/ANI204	Rotation speed command value input (VR In, analog value)
		P40/ANI000	U-phase current measurement
		P42/ANI002	W-phase current measurement
	MTU3	P73/MTIOC4B	PWM output (U_p) / "High" active
		P72/MTIOC4A	PWM output (V_p) / "High" active
		P71/MTIOC3B	PWM output (W_p) / "High" active
		P76/MTIOC4D	PWM output (U_n) / "High" active
		P75/MTIOC4C	PWM output (V_n) / "High" active
		P74/MTIOC3D	PWM output (W_n) / "High" active
	POE3	P70/POE0#	Use for PWM emergency-stop when overcurrent detected.
	GPIO	P23	START/STOP toggle switch
		P22	ERROR RESET push-switch
		P21	LED1 control
		P20	LED2 control
RA6T3 group (PMBus Slave)	I3C	P205_SCL0	Use as slave for PMBus communication.
		P206_SDA0_C	
	ADC12	P000/ANI000	U-phase current measurement
		P001/ANI001	V-phase current measurement
		P002/ANI002	W-phase current measurement
		P004/ANI004	Inverter bus voltage measurement
		P005/ANI005	Rotation speed command value input (VR In, analog value)
	GPT	P409/GTIOC1A	PWM output (U_p) / "High" active
		P103/GTIOC2A	PWM output (V_p) / "High" active
		P111/GTIOC3A	PWM output (W_p) / "High" active
		P408/GTIOC1B	PWM output (U_n) / "High" active
		P102/GTIOC2B	PWM output (V_n) / "High" active
		P112/GTIOC3B	PWM output (W_n) / "High" active
	POEG	P104/GTERGB	Use for PWM emergency-stop when overcurrent detection

	GPIO	P302	START/STOP toggle switch
		P200	ERROR RESET push-switch
		P113	LED1 control
		P106	LED2 control

4. Operation procedure

As shown in Section 1, the system in this application note uses PC terminal software to issue PMBus commands to control the motor. Operates motor module samplewith the setting to operate in Board UI. After connecting the board and turning on the power as shown in Section 3, follow the procedure below.

1. Turn ON SW1 (toggle SW) of the inverter board.

2. Rotate VR1 on the inverter-board in CW or CCW orientation.

3. Use the terminal software to enter PMBus commands in the order shown in Figure 19.

Table 7 lists the command specifications supported by the demo system.

When starting motor rotation, start rotation by sending the setting shown in Table 7 with ON_OFF_CONFIG command and OPERATION command of PMBUS under the condition of the above-mentioned steps 1 and 2. When stopping the rotation of the motor, issue a Pmbus command with ON_OFF_CONFIG and Operation, turn off SW1 (Togur SW), or set VR1 to the center (the lowest motor rotation). When the motor is to be rotated again, SW1 (toggle SW) must be rotated to ON position and VR1 must be rotated to a position other than the center position, and then OPERATION command-based rotational start-request must be received again.

- PC Terminal Software Settings for Write,Write/Read Commands

1st line of transmission	Slave address (8bit)⏎
2nd lin of transmission	R or W(UTF-8) *2⏎
3rd line of transmission	Command code(8bit)⏎
4th line of transmission	write data (8bit)*1⏎
	:
n line of transmission	write data(8bit))*1⏎
1st line received	Slave return value(8bit))*1 ⏎
	:
n line received	Slave return value(8bit))*1 ⏎
API return value	API return value(8bit) ⏎
Packet result	Packet result(8bit)⏎

Ex: For OPERATION command (WRITE)
0x0A
W
0x01
0x80

Return code:0x00
Packet result:0x00

- PC Terminal Software Settings for Read Commands

1st line of transmission	Slave address (8bit)⏎
2nd lin of transmission	R (UTF-8) *2⏎
3rd line of transmission	Command code(8bit)⏎
1st line received	Slave return value(8bit))*1 ⏎
	:
n line received	Slave return value(8bit))*1 ⏎
API return value	API return value(8bit) ⏎
Packet result	Packet result(8bit)⏎

Ex: For OPERATION command (READ)
0x0A
R
0x01
>>PMBUS_RESPONSE_START
0x08

Return code:0x00
Packet result:0x00

*1: Byte: 1 line, word:2 lines, BLOCK: n lines. Some commands display the value in decimal.
Refer **Table 7** for details.
*2: "R" means Read, and "W" means Write specification.

*: Enter 8bit in HEX notation, for example "0xAB".
*: In this application, the slave address is fixed to 0x0A.

Figure 19 PC terminal software to Issue PMBus Command

Table 7 Commands used in this demonstration system

Command name (Command code)	TRANSACTION CODE	Functional Description
OPERATION (0x01)	WRITE_BYTE	Only when bit3 is set to "1" in ON_OFF_CONFIG, the following operation is executed according to bit7. Bit7 = 1: Start rotating the motor. Bit7 = 0: Stop rotating the motor
	READ_BYTE	Return the value set in WRITE_BYTE.
ON_OFF_CONFIG (0x02)	WRITE_BYTE	Used in conjunction with OPERATION command. This demonstration system uses only bit3. For more information, refer to the explanation of OPERATION command-related functions.
	READ_BYTE	Return the value set in WRAITE_BYTE.
CLEAR_FAULTS (0x03)	SEND_BYTE	Clear the error information of the motor. (For RX26T, the motor operation is also reset.)
STATUS_FANS_1_2 (0x81)	WRITE_BYTE	Clear the status corresponding to the bit set to “1” in the FAN status.
	READ_BYTE	Return FAN status.
READ_VOUT (0x8b)	READ_WORD	Return the measured output voltage (V). In PC terminal software, it is displayed as and integer decimal number.
READ_IOUT (0x8c)	READ_WORD	Return the measured output current (mA). In PC terminal software, it is displayed as a signed integer decimal number.
READ_FAN_SPEED_1 (0x90)	READ_WORD	Return FAN velocity (rad/s). In PC terminal software, it is displayed as a signed integer decimal number.
READ_FREQENCY (0x95)	READ_WORD	Return the main power converter switching frequency (μ sec). In PC terminal software, it is displayed as and integer decimal number.

5. Software Description

The software process of this application note is divided into driver sections that control MCU peripheral functions, PMBus middleware sections that control PMBus, and user applications that operate PMBus middleware. In addition, SCI drivers connected to PC on PMBus Master and the motor control middleware for controlling the motor on PMBus Slave are operated by the user application. Figure 20 shows the module configuration of PMBus Master software, and Figure 21 and Figure 22 show the module configuration of PMBus Slave software. For details on PMBus Master state transitions and function operations, see Section 5.1. For details on PMBus Slave state transitions and function operations, see Section 5.2.

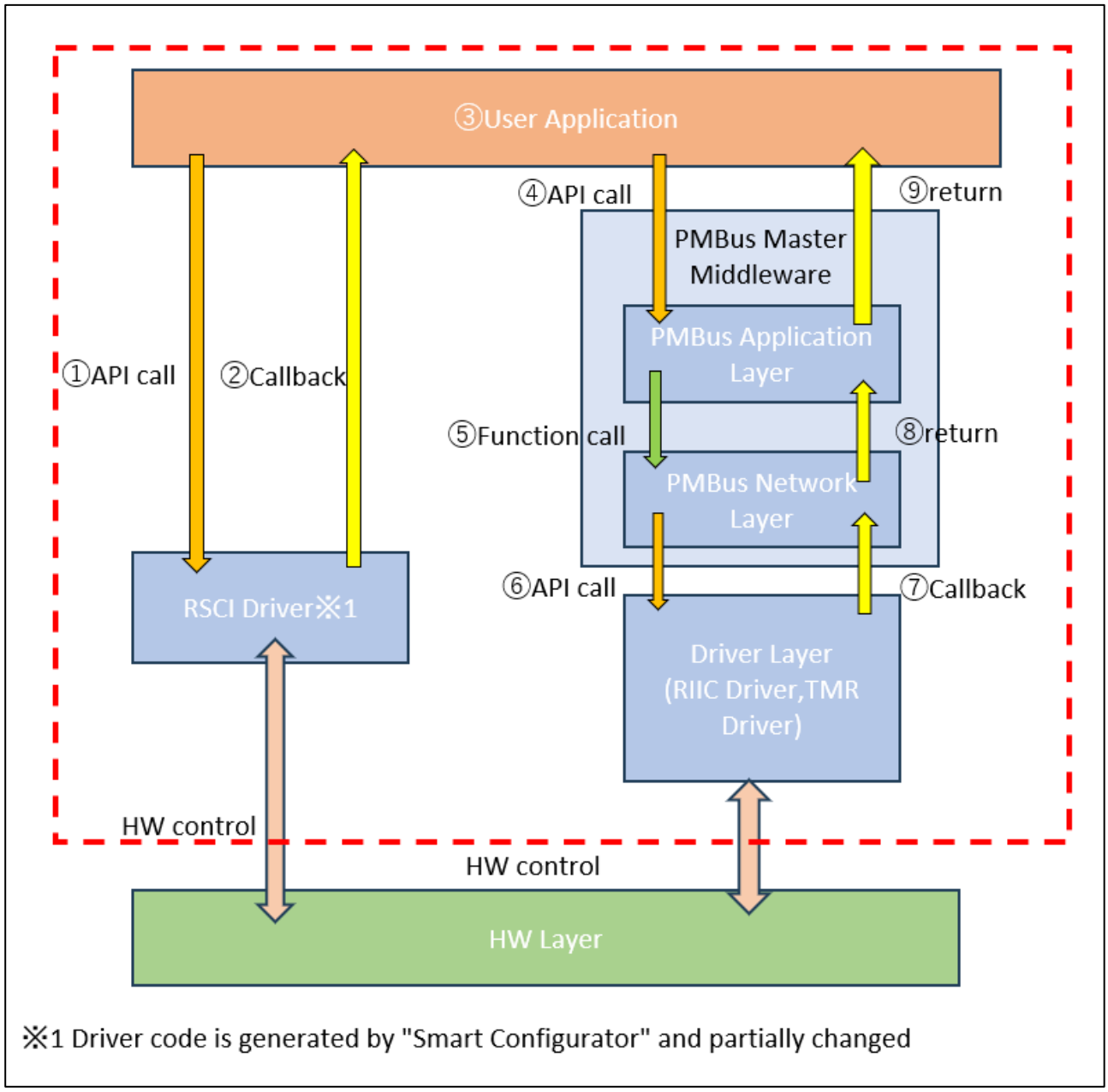


Figure 20 PMBus Master Software Module Configuration (RX26T Group)

Table 8 List of Global Variables Used on PMBus master Software Interfaces

IF No indicates the number of the interfaces in Figure 20 PMBus Master Software Module Configuration (RX26T Groups).

IF No	R_PMBUS_Master_Open	R_PMBUS_Master_Write	R_PMBUS_Master_Read
①	-	s_u1_uart_rx_relay_buf, s_u1_uart_tx_buf	s_u1_uart_rx_relay_buf, s_u1_uart_tx_buf
②	-	s_u1_uart_rx_relay_buf	s_u1_uart_rx_relay_buf
③	s_e_packet_result, s_u1_pmbus_ret, s_user_pmbus_cfg	s_u1_uart_rx_buf, s_u1_uart_tx_buf, s_u2_uart_rx_index, s_u2_rx_r_index, s_u2_rx_w_index, s_e_main_status, s_u2_seq_index, s_st_pmbus_data, s_e_packet_result, s_u1_pmbus_ret	s_u1_uart_rx_buf, s_u1_uart_tx_buf, s_u2_uart_rx_index, s_u2_rx_r_index, s_u2_rx_w_index, s_e_main_status, s_u2_seq_index, s_u2_rx_size, s_u1_pmbus_temp_rx_buf, s_e_packet_result, s_u1_pmbus_ret
④	s_user_pmbus_cfg, s_u1_pmbus_tx_buf, s_u1_pmbus_rx_buf, s_e_packet_result	s_st_pmbus_data, s_e_packet_result	s_st_pmbus_data, s_u1_pmbus_temp_rx_buf s_u2_rx_size, s_e_packet_result
⑤	g_st_pmbus_ctrl	g_st_pmbus_ctrl, s_st_pmbus_data	g_st_pmbus_ctrl
⑥	-	g_st_pmbus_ctrl	g_st_pmbus_ctrl
⑦	-	g_st_pmbus_ctrl	g_st_pmbus_ctrl
⑧	-	g_st_pmbus_ctrl	g_st_pmbus_ctrl
⑨	-	s_e_packet_result	s_u1_pmbus_temp_rx_buf s_u2_rx_size, s_e_packet_result

【NOTE】Only PMBUS Master API supported by this application are listed in this table.
For more information on global-variables, see 5.1.6 PMBus Master global variables List.

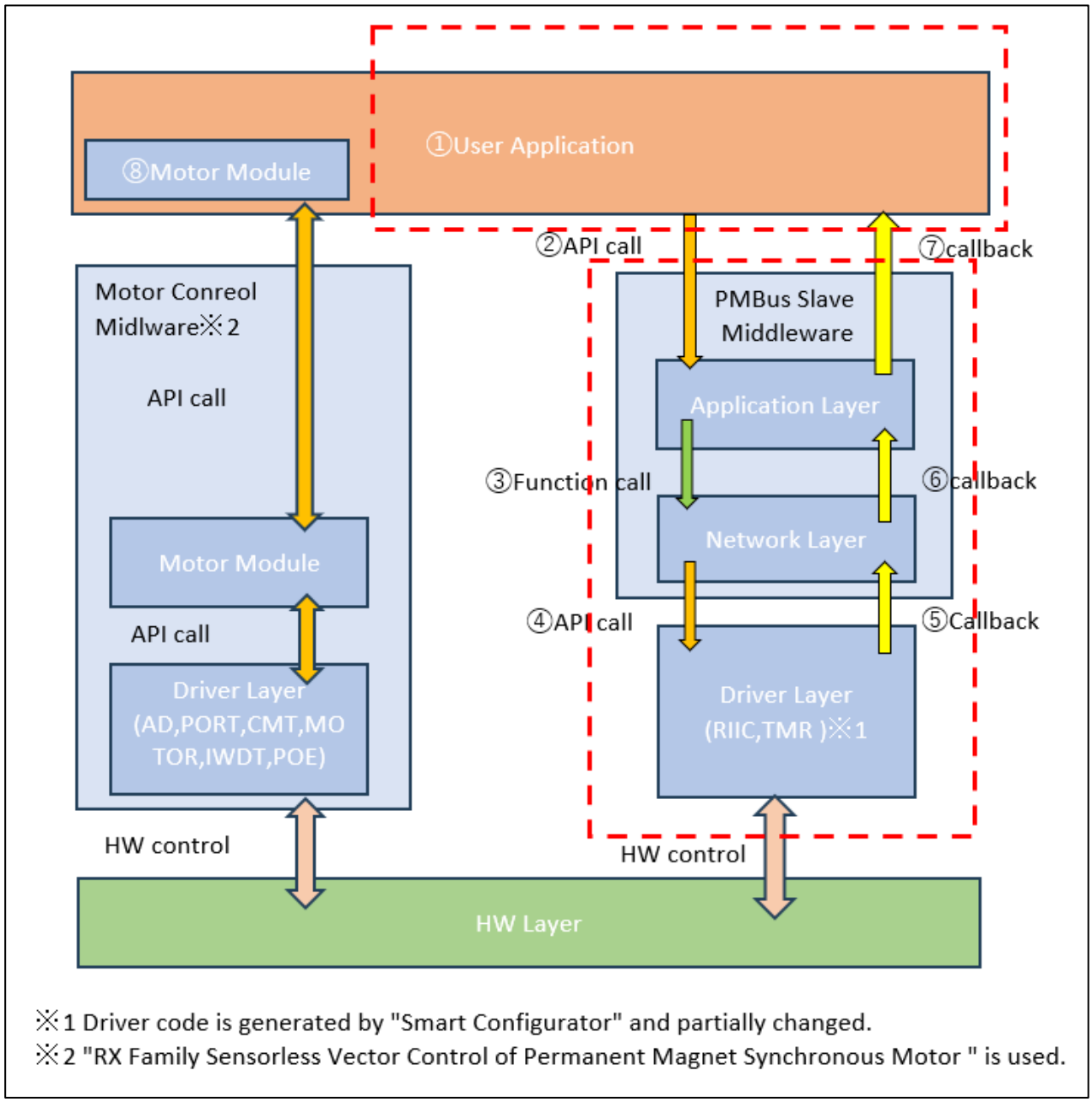


Figure 21 PMBus Slave Software Module Configuration (RX26T Group)

Table 9 List of Global Variables Used on PMBus slave Software Interfaces for RX26T Group

IF No indicates the number of the interfaces in Figure 21 PMBus Slave Software Module Configuration (RX26T Groups).

IF No	R_PMBUS_Slave_Open	Write Byte protocol	Read Byte Protocol
①	s_st_pmbus_cfg	s_u1_pmbus_rx_buf	s_u1_pmbus_rx_buf, s_u1_pmbus_tx_buf
②	s_st_pmbus_cfg, s_u1_pmbus_tx_buf, s_u1_pmbus_rx_buf	-	-
③	g_st_pmbus_ctrl, g_riic0_user_slave_addr	g_st_pmbus_ctrl	g_st_pmbus_ctrl
④	g_riic0_user_slave_addr	g_st_pmbus_ctrl	g_st_pmbus_ctrl
⑤	g_st_pmbus_ctrl	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf
⑥	g_st_pmbus_ctrl	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf
⑦	g_st_pmbus_ctrl	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf, s_u1_pmbus_tx_buf
⑧	-	s_u1_pmbus_config_data, s_u1_pmbus_operation_data *1	s_u1_pmbus_config_data, s_u1_pmbus_operation_data *1

【NOTE】Since the timing at which slave operation is started is an interrupt notification from Driver Layer, this table lists the global-variables to be used when operating under typical protocols and during Open API.
For more information on global-variables, see 5.2.6 PMBus Slave Global variables List.

*1. This global-variable is used only when OPERATION command is received or when ON_OFF_CONFIG command is received.

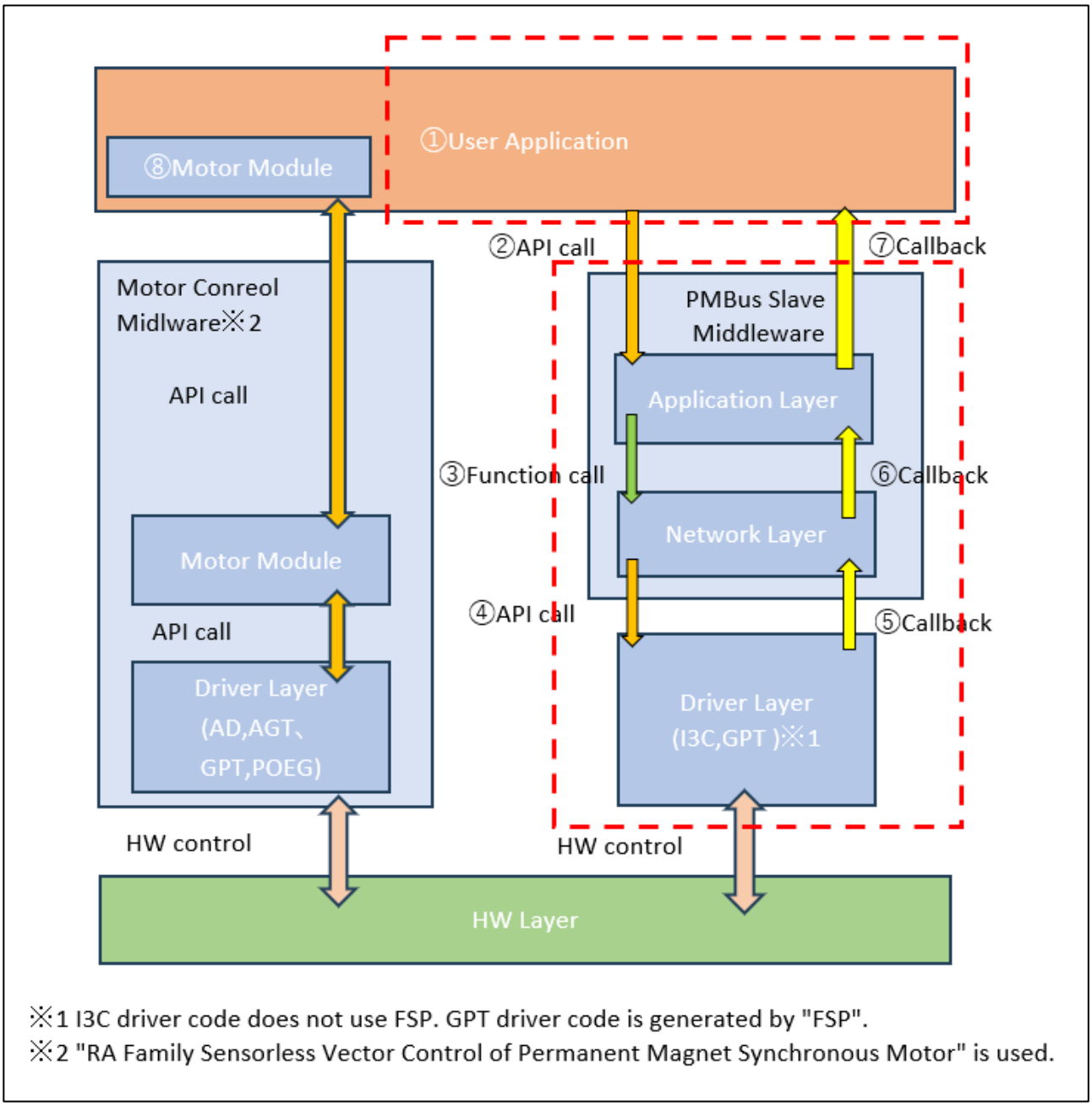


Figure 22 PMBus Slave Software Module Configuration (RA6T3 Group)

Table 10 List of Global Variables Used on PMBus slave Software Interfaces for RA6T3 Group

IF No indicates the number of the interfaces in

Figure 22 PMBus Slave Software Module Configuration (RA6T3 Groups).

IF No	R_PMBUS_Slave_Open	Write Byte protocol	Read Byte Protocol
①	s_st_pmbus_cfg	s_u1_pmbus_rx_buf	s_u1_pmbus_rx_buf, s_u1_pmbus_tx_buf
②	s_st_pmbus_cfg, s_u1_pmbus_tx_buf, s_u1_pmbus_rx_buf	-	-
③	g_st_pmbus_ctrl	g_st_pmbus_ctrl	g_st_pmbus_ctrl
④	s_st_gpt_ctrl, s_st_smbus_ctrl, s_st_smbus_slave_cfg, g_smbus_slave0	g_st_pmbus_ctrl, st_smbus_ctrl	g_st_pmbus_ctrl, st_smbus_ctrl
⑤	g_st_pmbus_ctrl	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf
⑥	g_st_pmbus_ctrl	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf
⑦	g_st_pmbus_ctrl	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf	g_st_pmbus_ctrl, s_u1_pmbus_rx_buf, s_u1_pmbus_tx_buf
⑧	-	s_u1_pmbus_config_data, s_u1_pmbus_operation_data*1	s_u1_pmbus_config_data, s_u1_pmbus_operation_data*1

【NOTE】Since the timing at which slave operation is started is an interrupt notification from Driver Layer, this table lists the global-variables to be used when operating under typical protocols and during Open API.
For more information on global-variables, see 5.2.6 PMBus Slave Global variables List.

*1. This global-variable is used only when OPERATION command is received or when ON_OFF_CONFIG command is received.

5.1 PMBus Master softwares

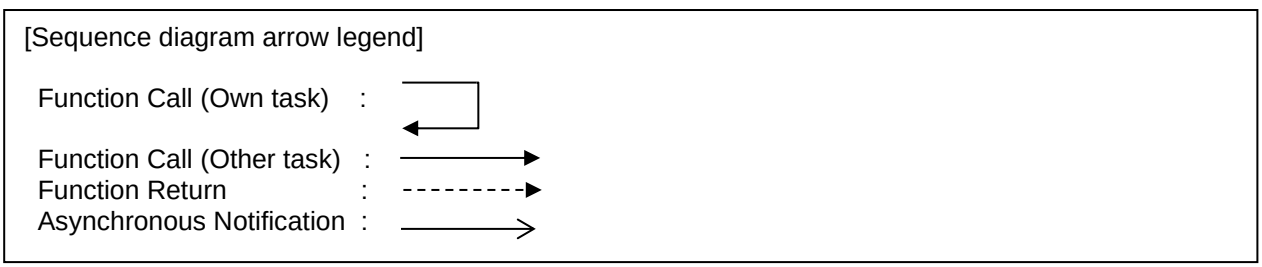
PMBus Master software is classified into the user application part, middleware part, and driver part as shown in Figure 20 PMBus Master software module configuration. The driver section is modified or added to Execute some PMBus Master operations using the software generated by the smart configurator. Please refer to Section 5.1.4 for details of changes and additions. Table 11 shows the folder and file structure of each software.

Table 11 PMBus Master RX26T Group Folder/File Configuration

Folder name	File name	Outline
app	r_pmbus_demo_master.c	The main program of PMBus demonstration system. (User Applications)
	r_pmbus_demo_master.h	The header file to use for the main program of PMBus demonstration system.
pmbus_master	r_pmbus_app_master.c	The application-layer of PMBus Middleware.
	r_pmbus_app_master.h	The header file to use for the application-layer of PMBus Middleware.
	r_pmbus_nwk_master.c	The network-layer of PMBus Middleware.
	r_pmbus_nwk_master.h	The header file to use for the network-layer of PMBus Middleware.
src\smc_gen\Config_RIIC0	Config_RIIC0.c	The driver-layer of PMBus Middleware. Generate by the smart configurator.
	Config_RIIC0.h	The driver-layer of PMBus Middleware. Generate by the smart configurator.
	Config_RIIC0_user.c	The driver-layer of PMBus Middleware. Generate by the smart configurator.
src\smc_gen\Config_RSCI11	Config_RSCI11.c	The driver-layer of PMBus user application. Generate by the smart configurator.
	Config_RSCI11.h	The driver-layer of PMBus user application. Generate by the smart configurator.
	Config_RSCI11_user.c	The driver-layer of PMBus user application. Generate by the smart configurator.
src\smc_gen\Config_TMR0	Config_TMR0.c	The driver-layer of PMBus Middleware. Generate by the smart configurator.
	Config_TMR0.h	The driver-layer of PMBus Middleware. Generate by the smart configurator.
	Config_TMR0_user.c	The driver-layer of PMBus Middleware. Generate by the smart configurator.

5.1.1 PMBus Master Operation Sequence

Figure 23 shows the sequence of operations from the issuance of PMBus command to the completion of the operation. Figure 24 and Figure 25 shows Write operation sequence, and Figure 26 and Figure 27 show Read operation sequence and Write/Read operation sequence according to PMBus command. For the functions used in each operation, refer to PMBus Master Function List in Section 5.1.3.



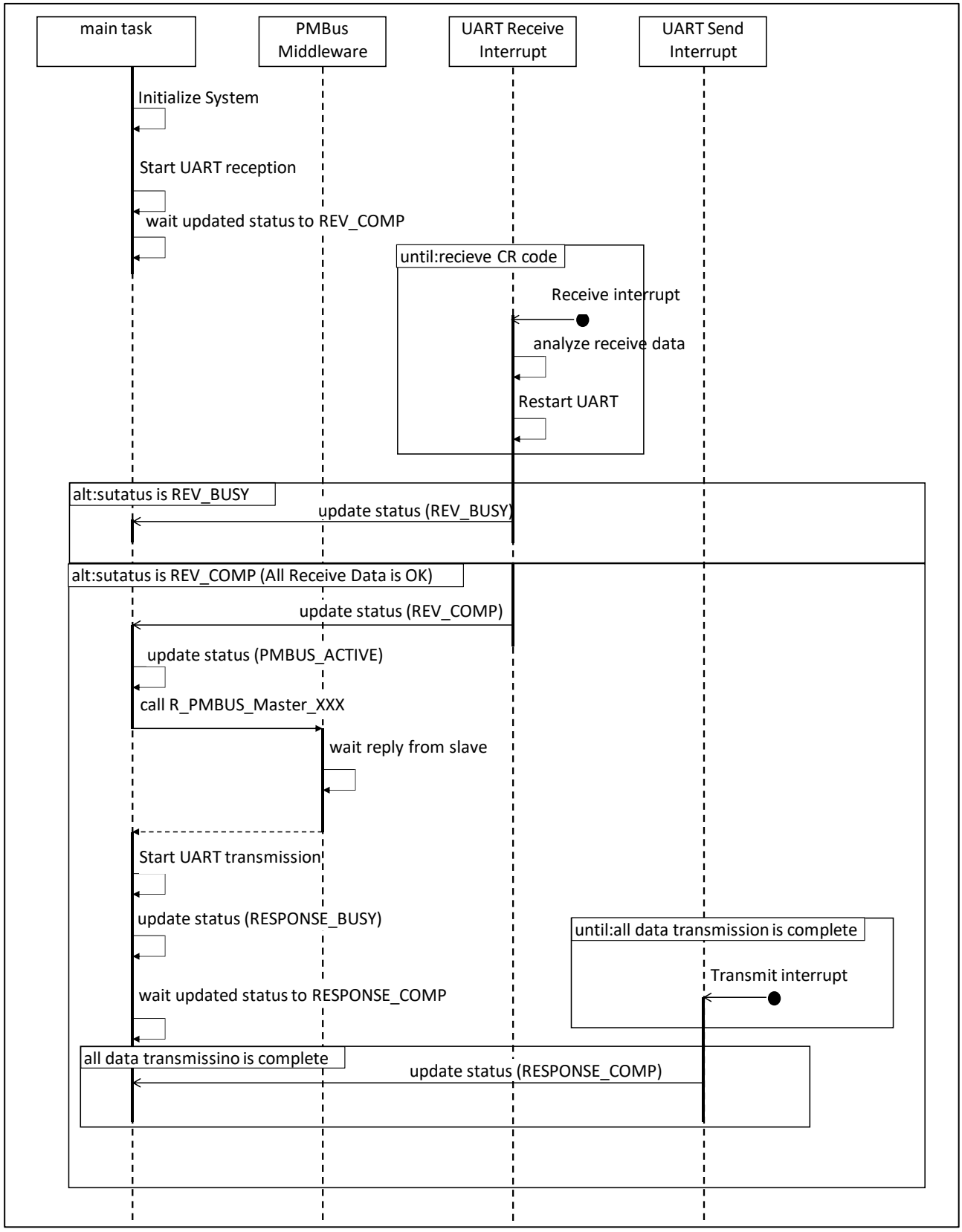


Figure 23 PMBus Master operation Sequence diagram

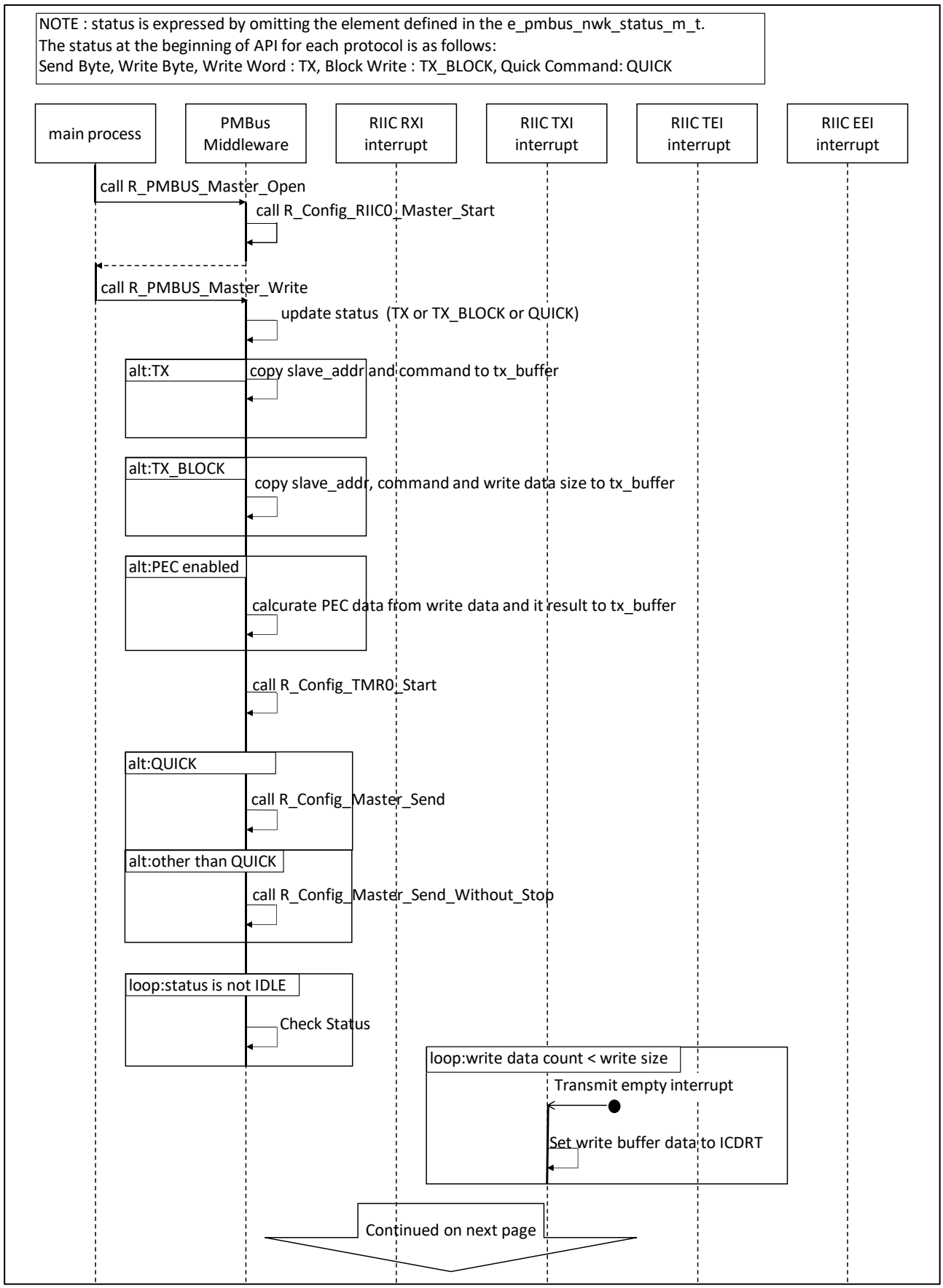


Figure 24 Write Operation (R_PMBUS_Master_Write) Sequence Diagram (1/2)

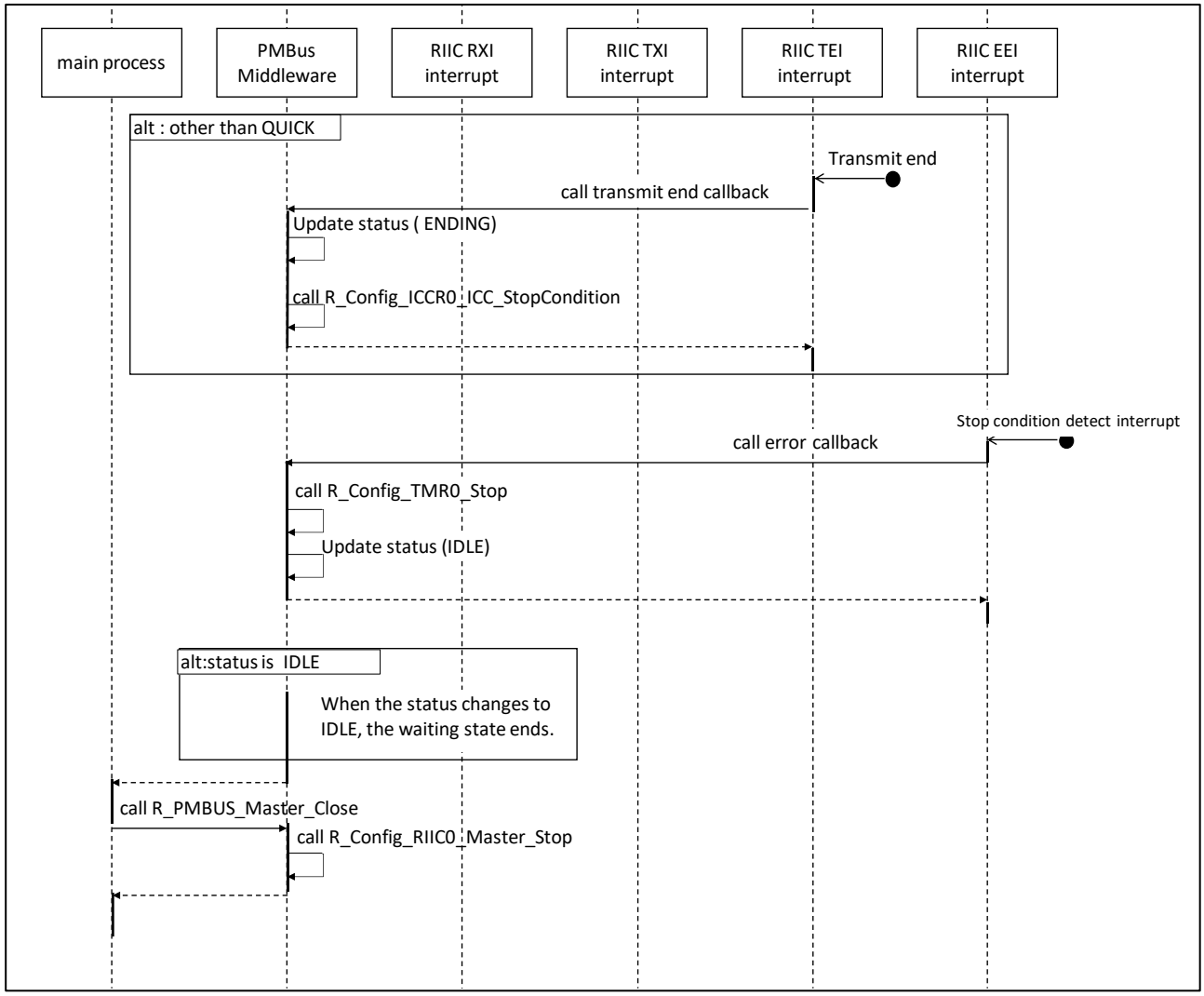


Figure 25 Write Operation (R_PMBUS_Master_Write) Sequence Diagram (2/2)

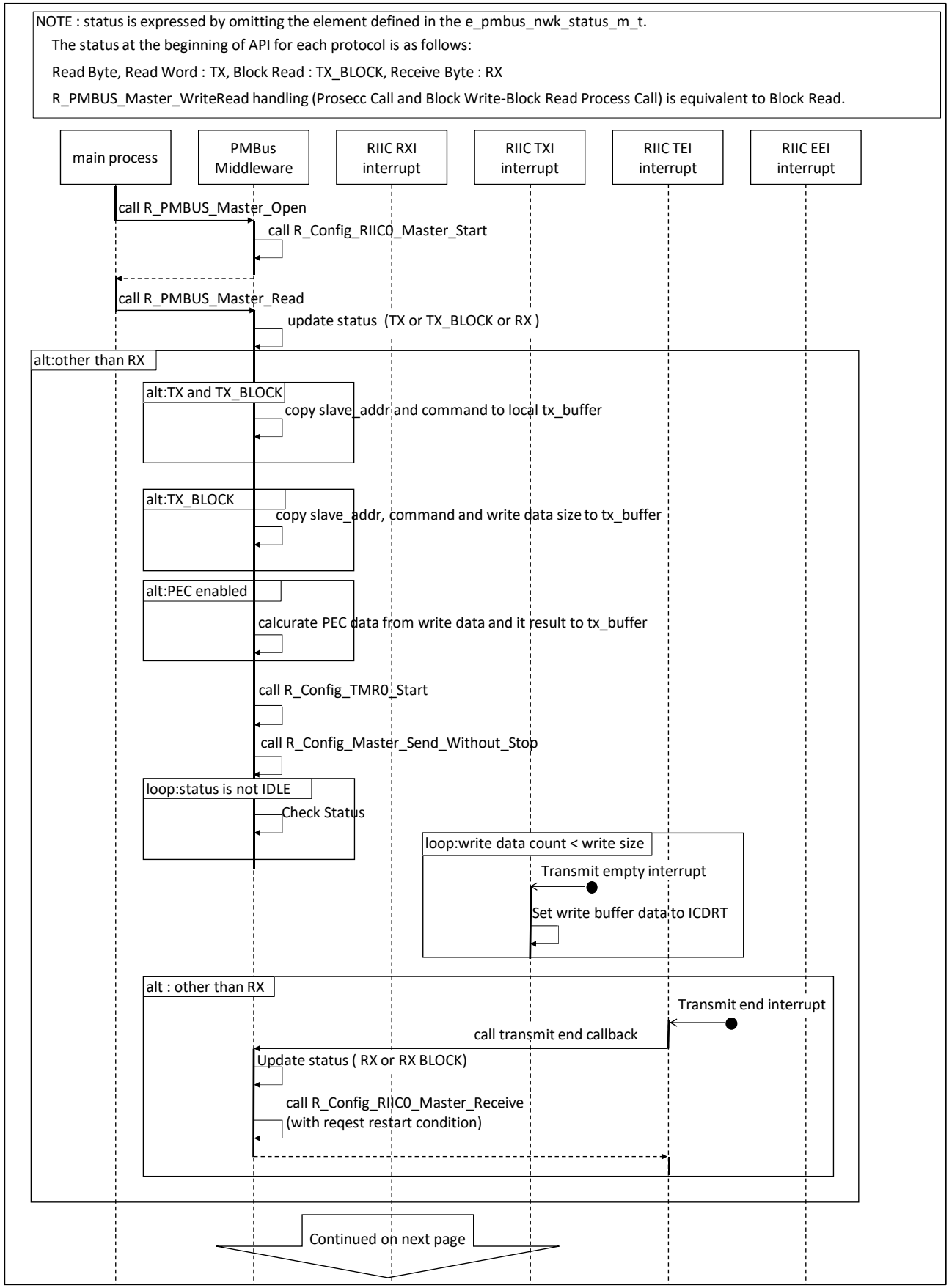


Figure 26 Read operation (R_PMBUS_Master_Read) and Write/Read operation (R_PMBUS_Master_WriteRead) Sequence Diagram (1/2)

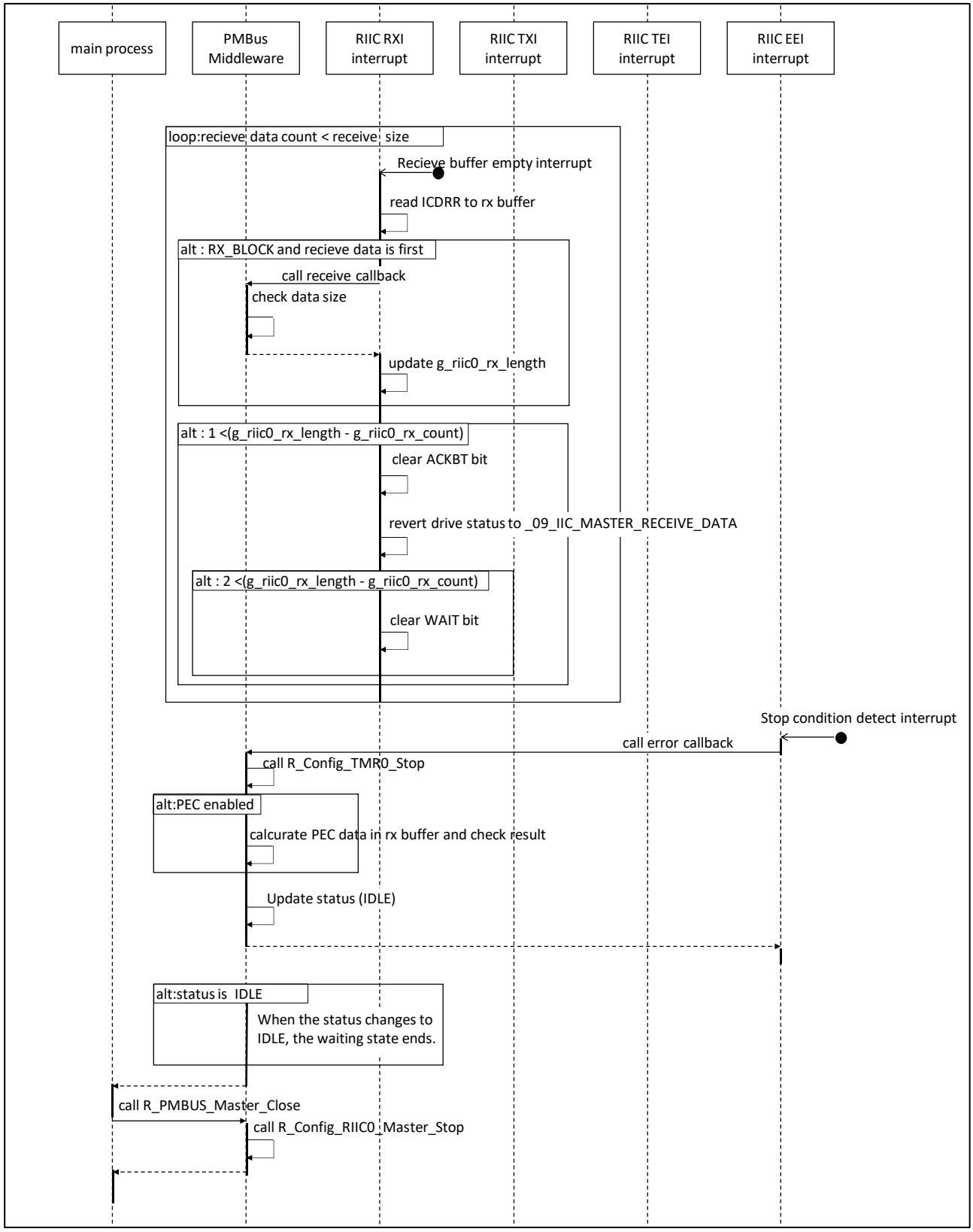


Figure 27 Read operation (R_PMBUS_Master_Read) and Write/Read operation (R_PMBUS_Master_WriteRead) Sequence Diagram (2/2)

5.1.2 PMBus Master status transitions

The status of PMBus Master middleware is managed by each of Application Layer and Driver Layer sections. Application Layer manages protocol-status transitions, and Driver Layer manages data-transmission/reception counts. Section 5.1.2.1 shows the status transitions of Application Layer part, and Section 5.1.2.2 shows the status transitions of Driver Layer part.

5.1.2.1 PMBus Master Middleware Application Layer status transitions

The status of Application Layer of PMBus Master manages the status of send, receive, Quick command, Block command, and error-handling in accordance with the command code. Figure 28, Figure 29, and Table 12 show below.

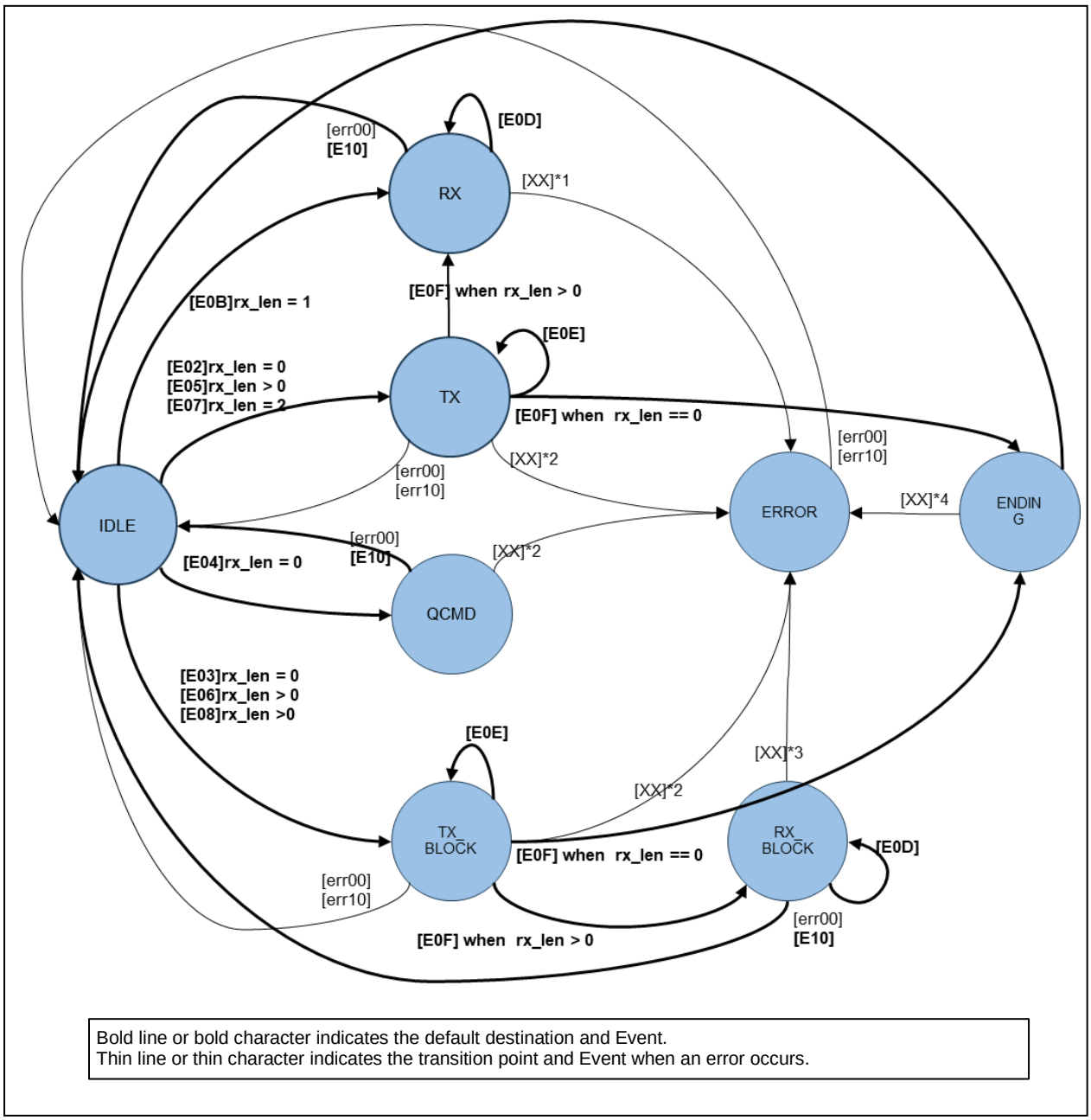


Figure 28 PMBus Master Application Layer status transition diagram

[Event list] * [EXX] means normal event, and [errXX] means Event in the case of abnormal occurrence.

E00/err00_Open ,E01_Close ,E02_Write
E03_Write (Block) ,E04_Write (Quick) ,E05_Read
E06_Read (Block) ,E07_WriteRead ,E08_WriteRead (BLOCK)
E09_EnablePEC ,E0A_DisablePEC ,E0B_ReceiveARA
E0C_Interrupt(Start Condition detect) ,E0D/err0D_Interrupt(Receive Buffer Full)
E0E/err0E_Interrupt(Transmit Interrupt) ,E0F/err0F_Interrupt(Transmit End Interrupt)
E10/err10_Interrupt(Stop Condition detect) ,err11_Interrupt(Arbitlation Lost)
err12_Interrupt(NACK detect) ,err13_Interrupt(Timeout Detect)

*1. conditions are as follows.
[err0D] If (rx_index over rx_len) ,[err0F] ,[err10] If (pec is enabled and pec data is error)
[err11] ,[err12] ,[err13]

*2. conditions are as follows.
[err0D] ,[err11] ,[err12] ,[err13]

*3. conditions are as follows.
[err0D]if (rx_index over rx_len) or first receive data size is Out of range) ,[err0F]
[err10]if (pec is enabled and pec data is error) ,[err11]
[err12] ,[err13]

*4. conditions are as follows.
[err0D] ,[err0F] .「err11」 ,[err13]

Figure 29 PMBus Master Application Layer status transition diagram(Supplement to Figure 28)

Table 12 PMBus Master Application Layer status transition table

The table can be read as follows.
- The event consists of an abbreviation of the API name and an interrupt factor.
- status" is an abbreviation of "e_pmbus_nwk_status_m_t".
- [If (<Condition>)] means that transitions are conditional.
- [→ <state>] means a transition to a state.
- [ERROR (<error_name>)] means the return value of API.
- [PACKET RESULT (<error name>)] means the error information stored in API parameter p_e_packet_result.
- [-] means that the state does not transition.
- Light blue indicates the state transitions in Figure 28

	IDLE	RX	TX	QCMD	TX_BLOCK	RX_BLOCK	ENDING	ERROR
E00/err00_Open	→ IDLE*2	→ IDLE*2	→ IDLE*2	→ IDLE*2	→ IDLE*2	→ IDLE*2	→ IDLE*2	→ IDLE*2
E01_Close	-	-	-	-	-	-	-	-
E02_Write	→ TX rx_len = 0	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)
E03_Write (Block)	→ TX_BLOCK rx_len = 0	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)
E04_Write (Quick)	→ QCMD rx_len = 0	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)
E05_Read	→ TX rx_len = not 0	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)
E06_Read (Block)	→ TX_BLOCK rx_len = not 0	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)
E07_Write Read	→ TX rx_len = 2	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)
E08_Write Read (Block)	→ TX_BLOCK rx_len = not 0	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)
E09_EnablePEC	-	-	-	-	-	-	-	-

E0A_DisablePEC	-	-	-	-	-	-	-	-
E0B_ReceiveARA	→ RX rx_len = 1	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)
E0C_Interrupt (Start Condition detect)	-	-	-	-	-	-	-	-
E0D/err0D_Interrupt (Receive Buffer Full)	-	(read receive data from ICDRR) if (rx_index over rx_len) PACKET RESULT (DATA_SIZE) → ERROR*1	PACKET RESULT (INTERNAL) → ERROR*1	PACKET RESULT (INTERNAL) → ERROR*1	PACKET RESULT (INTERNAL) → ERROR*1	(read receive data from ICDRR) if (rx_index over rx_len) or (first receive data size is Out of range) PACKET RESULT (DATA_SIZE) → ERROR*1 if (first receive data size is in range) Update rx_len	PACKET RESULT (INTERNAL) → ERROR*1	PACKET RESULT (INTERNAL) → ERROR*1
E0E/err0E_Interrupt (Transmit Interrupt)	-	-	(set transmit data to ICDRT)	-	(set transmit data to ICDRT)	-	-	-
E0F/err0F_Interrupt (Transmit End Interrupt)	-	PACKET RESULT (INTERNAL) → ERROR*1	if (rx_len > 0): → RX if (rx_len == 0): → ENDING*1	-	if (rx_len > 0): → RX_BLOCK if (rx_len == 0): → ENDING*1	PACKET RESULT (INTERNAL) → ERROR*1	PACKET RESULT (INTERNAL) → ERROR*1	PACKET RESULT (INTERNAL) → ERROR*1
E10/err10_Interrupt (Stop Condition detect)	-	If (pec is enabled and pec data is error) PACKET_ERROR（PEC） → IDLE	→ IDLE	→ IDLE	→ IDLE	If (pec is enabled and pec data is error) PACKET RESULT（PEC） → IDLE	→ IDLE	→ IDLE
err11_Interrupt (Arbitlation Lost)	-	PACKET RESULT (ARB_LOST) → ERROR	PACKET RESULT (ARB_LOST) → ERROR	PACKET RESULT (ARB_LOST) → ERROR	PACKET RESULT (ARB_LOST) → ERROR	PACKET RESULT (ARB_LOST) → ERROR	PACKET RESULT (ARB_LOST) → ERROR	PACKET RESULT (ARB_LOST) → ERROR
err12_Interrupt (NACK detect)	-	PACKET RESULT (NACK) → ERROR*1	PACKET RESULT (NACK) → ERROR*1	PACKET RESULT (NACK) → ERROR*1	PACKET RESULT (NACK) → ERROR*1	PACKET RESULT (NACK) → ERROR*1	PACKET RESULT (OK)	PACKET RESULT (NACK) → ERROR*1
err13_Interrupt (Timeout Detect)	-	PACKET RESULT (TIMEOUT) → ERROR*1	PACKET RESULT (TIMEOUT) → ERROR*1	PACKET RESULT (TIMEOUT) → ERROR*1	PACKET RESULT (TIMEOUT) → ERROR*1	PACKET RESULT (TIMEOUT) → ERROR*1	PACKET RESULT (TIMEOUT) → ERROR*1	PACKET RESULT (TIMEOUT) → ERROR*1

*1. Issue Stop Condition.
*2. Switches to Idle only when Open occurs after Close.

5.1.2.2 PMBus Master Driver Layer status transitions

The state transition of the PMBus Master Driver Layer is divided into the transmit operation part and the receive operation part to PMBus Slave, and the specified data is sent/received by the specified number of bytes. PMBus Master Driver status transitions are shown in Figure 30, Figure 31, and Table 13, Table 14.

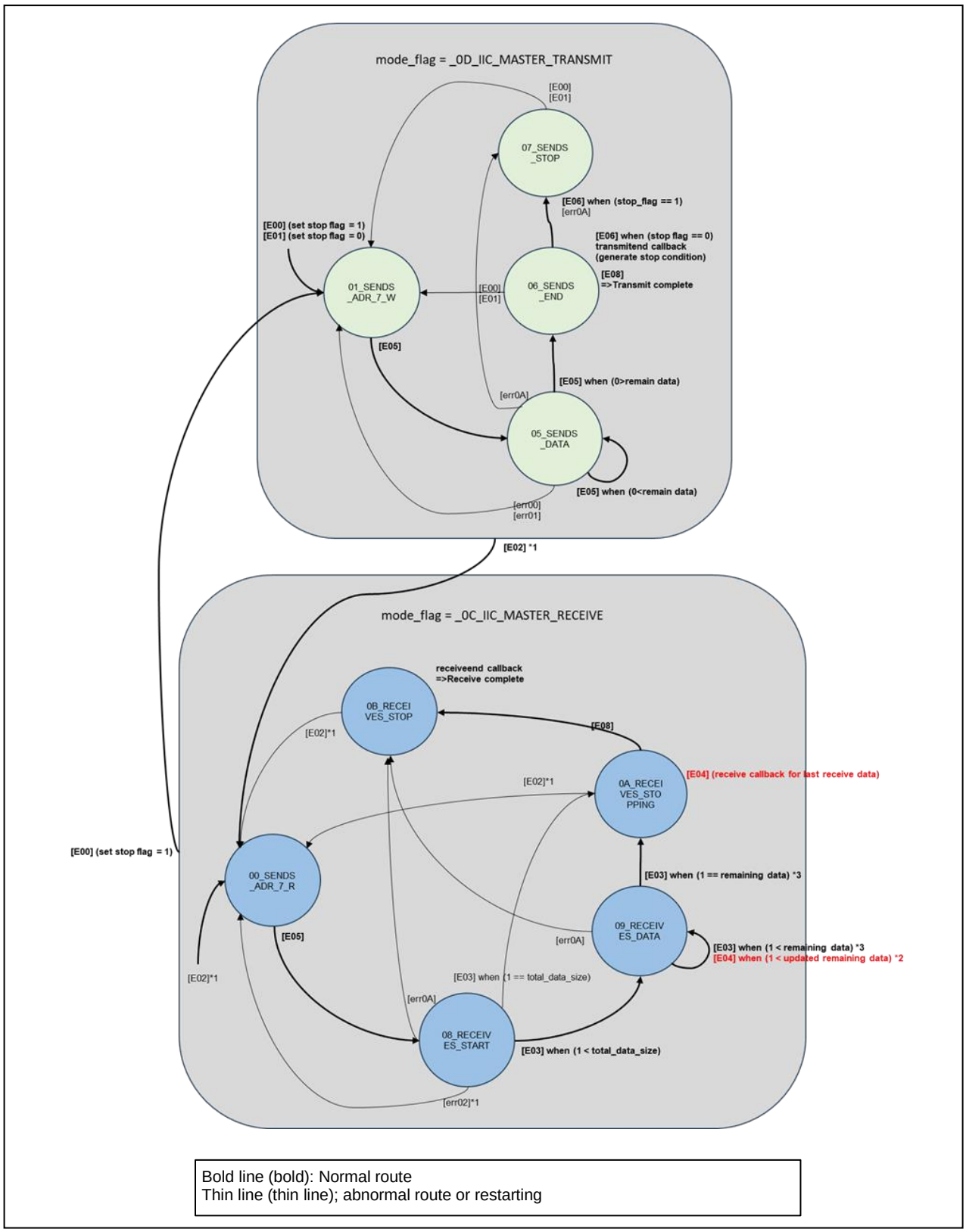


Figure 30 PMBus Master Driver Layer status transitions

[Event List]
* [Exx] indicates normal Event, and [errxx] indicates Event when an error occurs.
An error Event not shown in the list occurs in any status, and an error is notified by callback error.
E00/err00_Send (stop_flag = 1)
E01/err01_Send_Without_Stop (stop_flag = 0)
E02/err02_Rreceive
E03_Interrupt (Receive Buffer Full)
E04_callback receive *1
E05_Interrupt (Transmit)
E06_Interrupt (Transmit End)
E07/err07_Interrupt (Start Condition Detect)
E08/err08_Interrupt (Stop Condition Detect)
err09_Interrupt (Arbitlation Lost)
err0A_Interrupt (NACK detect)

*1. The default receive data count (total_data_size) specified. Block Read and Block Write-BlockRead Process Call is 3 (Data Size(1) + Data(1) + PEC(1)), which is the minimum number of received data.
Otherwise, specify the number of data according to each protocol. However, "0" cannot be specified.
*2. Check the received data. callback receive, and if Block read, refresh the expected data count.
As a result, if the remaining number of received data is 2 bytes or more, the status is returned to 09.
Set ACKBT=0 when the number of remaining received data is 2 bytes or more, and set WAIT=0 when the number of remaining received data is 3 bytes or more.
*3. Set WAIT=1 when the remaining number of received data is 2 bytes. Set ACKBT=1 when the remaining number of received data is 1-byte.

Figure 31 PMBus Master Driver Layer status transition diagram (Supplement to Figure 30)

Table 13 PMBus Master Driver Layer status transition table（when transmit）

This table shows the state transition table when mode_flag is _0D_IIC_MASTER_TRANSMIT.
Explanations of the annotations are summarized in Table 14.

This table and Table 14. should be interpreted as follows.
- Since slave-address 10-bit is not used in PMBus, status control is omitted.
- The status control is omitted because RIIC0 timeout detection interrupt is not used in PMBus.
- [→<number>] is the number of the transition destination. (<number>) indicates the number of the transition destination of mode_flag.
- [-] means no state transition.
- [Callback <xxx>] refers to executing a callback. Callback error (<number>) means the error-information passed to callback error.
- [If (<Condition>)] refers to conditional operation.
- Red text indicates the process changed for PMBus middleware.
- Light green indicates state transitions when mode_flag=0D in Figure 30.
- Light blue indicates state transitions when mode_flag=0C in Figure 30.

	_01_IIC_MASTER_SENDS_ADR_7_W	**_05_IIC_MASTER_SENDS_DATA**	**_06_IIC_MASTER_SENDS_END**	**_07_IIC_MASTER_SENDS_STOP**
E00/err00_Send (stop_flag = 1)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)
E01/err01_Send_Without_Stop (stop_flag = 0)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)
E02/err02_Rreceive *2	→ 00 (0C)*1	→ 00 (0C)*1	→ 00 (0C)*1	→ 00 (0C)*1
E03_Interrupt (Receive Buffer Full)	-	-	-	-
E04_callback receive*3	-	-	-	-
E05_Interrupt (Transmit)	→ 05	if (0>remain_data) → 06	-	-
E06_Interrupt (Transmit End)	-	-	If (stop_flag == 1) → 07 If (stop_flag == 0) callback transmitend	-
E07/err07_Interrupt (Start Condition Detect)*6	-	callback error (MD_ERROR4)	callback error (MD_ERROR4)	callback error (MD_ERROR4)

E08/err08_Interrupt (Stop Condition Detect)	-	callback error (MD_ERROR4)	callback error (MD_ERROR4)	callback transmitend
err09_Interrupt (Arbitlation Lost)	callback error (MD_ERROR1)	callback error (MD_ERROR1)	callback error (MD_ERROR1)	callback error (MD_ERROR1)
err0A_Interrupt (NACK detect)	→ 07 callback error (MD_ERROR3)	→ 07 callback error (MD_ERROR3)	→ 07 callback error (MD_ERROR3)	→ 07 callback error (MD_ERROR3)

Table 14 PMBus Master Driver Layer status transition table（When receive）

This table shows the state transition when mode_flag is _0C_IIC_MASTER_RECEIVE.

	_00_IIC_MASTER_SENDS_ADR_7_R	**_08_IIC_MASTER_RECEIVE_START**	**_09_IIC_MASTER_RECEIVE_DATA**	**_0A_IIC_MASTER_RECEIVE_STOPPING**	**_0B_IIC_MASTER_RECEIVE_STOP**
E00/err00_Send (stop_flag = 1)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)
E01/err01_Send_Without_Stop (stop_flag = 0)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)	→ 01 (0D)
E02/err02_Rreceive *2	→ 00 (0C)*1	→ 00 (0C)*1	→ 00 (0C)*1	→ 00 (0C)*1	→ 00 (0C)*1
E03_Interrupt (Receive Buffer Full)	-	*4 (1 == total_data_size) → 0A If (1<total_data_size) →09 callback receive	*4 (1 == total_data_size) → 0A If (1 == remain data) → 0A callback receive If (1<total_data_size) or (1 < remain_data) callback receive	*5 → 0B callback receive	-
E04_callback receive*3	-	-	If (1 <updated remain data) → 09 *6	(last receive data process)	-
E05_Interrupt (Transmit)	→ 08	-	-	-	-
E06_Interrupt (Transmit End)	-	-	-	-	-
E07/err07_Interrupt (Start Condition Detect)*6	-	callback error (MD_ERROR4)	callback error (MD_ERROR4)	callback error (MD_ERROR4)	callback error (MD_ERROR4)
E08/err08_Interrupt (Stop Condition Detect)	-	callback error (MD_ERROR4)	callback error (MD_ERROR4)	callback error (MD_ERROR4)	callback receiveend
err09_Interrupt (Arbitlation Lost)	-	callback error (MD_ERROR1)	callback error (MD_ERROR1)	callback error (MD_ERROR1)	callback error (MD_ERROR1)
err0A_Interrupt (NACK detect)	-	→ 0B callback error (MD_ERROR3)	→ 0B callback error (MD_ERROR3)	→ 0B callback error (MD_ERROR3)	→ 0B callback error (MD_ERROR3)

*1. When ICCR2.MST is 1, set ICCR2.RS to 1 and issue a restart condition.
*2. The default receive data count (total_data_size) specified at Block Read and Block Write-BlockRead Process Call is 3 (Data Size(1) + Data(1) + PEC(1)), which is the minimum number of received data. For other protocols, specify the number of data according to each protocol. "0" cannot be specified.
*3. Status transitions are also executed inside callback receive.
*4. Set WAIT=1 if (2 == tatal_data_size) or (1 = tatal_data_size). If (1 = tatal_data_size), set ACKBT=1.
*5. Set WAIT = 0.
*6. (2 <updated remain data): Set WAIT=0. (1 <updated remain data): Set ACKBT=0.
*7. Start condition interrupt is not used.

5.1.3 PMBus Master Function List

PMBus Master functions are divided into Application functions in Table 15, API functions in Table 16, Middleware functions in Table 17, and driver functions generated by the Smart Configurator in Table 18. The driver-function is partially changed according to PMBus Master process. Refer to Customizing 5.1.4 PMBus Master Driver section for details.

Table 15 PMBus Master Application function list

File name	Function name	Function
r_pmbus_app_master.c	main	The main process of the application.
	pmbus_demo_main	The main process for transmission and reception by RSCI11 and control PMBUS.
	pmbus_demo_init	Application initialization process.
	pmbus_demo_updata_status	Update the internal state of the application.
	pmbus_demo_notify_result	Send PMBUS process to the terminal software via RSCI11.
	pmbus_demo_response	Send PMBUS received data to the terminal software via RSCI11.
	pmbus_demo_pmbus_execute	Execute PMBUS API according to the receive command.
	pmbus_demo_uart_recv_callback	Execute RSCI11 receive interrupt process.
	pmbus_demo_notify_event	Send application process information to the terminal software via RSCI11.
	pmbus_demo_update_sequence	The main process for manage internal state transitions within an application.
	pmbus_demo_seq_idle	Analyzes the received data when the first character string is received from the terminal software.
	pmbus_demo_seq_slave_addr	Analyzes the received data when the slave address and R/W data is received from the terminal software.
	pmbus_demo_seq_rw	Analyzes the transaction when a command is received from the terminal software.
	pmbus_demo_seq_cmd	Analyzes the data when data for sending PMBUS is received from the terminal software.
	r_uart_ctrl_check_CmdFormData	Check whether the number of data received from the terminal software matches the format of the received command.
	r_uart_ctrl_check_CmdFormCmd	Check whether the command received from the terminal software is supported.
	r_uart_ctrl_conv_StrToDec	Converts hexadecimal ASCII characters received from the terminal software to hexadecimal digits.
	r_uart_ctrl_is_read_write	Converts W/R data received from the terminal software to bool data.
	r_uart_ctrl_conv_DecToStrCont	To return a response to the terminal software, convert the response data to hexadecimal ASCII character.
	r_uart_ctrl_conv_DecToStr2Byte	To return a response to the terminal software, convert the response data to decimal ASCII character.
	pmbus_demo_system_init	Initialize the driver used by the application.
	pmbus_demo_system_deinit	Stop the driver used by the application.
	pmbus_demo_uart_ctrl_init	Initialize RSCI11 communication function.
	pmbus_demo_uart_ctrl_start	Start RSCI11 communication with the terminal software.

	pmbus_demo_uart_ctrl_stop	Stop RSCI11 communication with the terminal software.
	pmbus_demo_uart_ctrl_read_buf	Copy data received by RSCI11 to the internal buffer.
	pmbus_demo_uart_ctrl_clear_buf	Clear buffer data received by RSCI11.

Table 16 PMBus Master API function list

File name	Function name	Function
r_pmbus_app_master.c	R_PMBUS_Master_Open	Open PMBus Middleware.
	R_PMBUS_Master_Close	Close PMBus Middleware.
	R_PMBUS_Master_Write	Execute PMBUS command protocol for the sending.
	R_PMBUS_Master_Read	Execute PMBUS command protocol for the receiving.
	R_PMBUS_Master_WriteRead	Execute PMBUS command protocol for the Process Call system.
	R_PMBUS_Master_EnablePEC	Enable sending and receiving packets with PEC.
	R_PMBUS_Master_DisablePEC	Disable sending and receiving packets with PEC.
	R_PMBUS_Master_ReceiveARA	Execute alert response.

Table 17 PMBus Master Middleware function list

File name	Function name	Function
r_pmbus_app_master.c	r_pmbus_app_InitCtrl	Initialize PMBus Middleware parameters.
	r_pmbus_app_SendByte	Execute Send Byte protocol.
	r_pmbus_app_WriteByte	Execute Write Byte protocol.
	r_pmbus_app_WriteWord	Execute Write Word protocol.
	r_pmbus_app_BlockWrite	Execute Block Write protocol.
	r_pmbus_app_QuickWrite	Execute Quick Command (Write) protocol.
	r_pmbus_app_ReceiveByte	Execute Receive Byte protocol.
	r_pmbus_app_ReadByte	Execute Read Byte protocol.
	r_pmbus_app_ReadWord	Execute Read Word protocol.
	r_pmbus_app_BlockRead	Execute Block Read protocol.
	r_pmbus_app_ProcessCall	Execute Process Call protocol.
	r_pmbus_app_BlockProcessCall	Execute Block Write-Block Read protocol.
	r_pmbus_app_StartendByte	Start Send Byte protocol.
	r_pmbus_app_StartWriteByteWord	Start Write protocol.
	r_pmbus_app_StartBlockWrite	Start Block Write protocol.
	r_pmbus_app_StartQuickCmd	Start Quick Command protocol.
	r_pmbus_app_StartReceiveByte	Start Receive Byte protocol.
	r_pmbus_app_StartReadByteWord	Start Read protocol.
	r_pmbus_app_StartBlockRead	Start Block Read protocol.
	r_pmbus_app_StartProcessCall	Start Process Call protocol.
	r_pmbus_app_StartBlockProcess	Start Block Write-Block Read Process Call protocol.
	r_pmbus_app_WaitProcessEnd	Wait for the protocol to finish executing.
	r_pmbus_app_SetTxBuf	Copy the transmission data specified by the argument to the transmission buffer.
	r_pmbus_app_SetBlockTxBuf	Copy the transmission Block data specified by the argument to the transmission buffer.

	r_pmbus_app_GetRxBuf	Copy the received data from the receive buffer to the return buffer.
	r_pmbus_app_int_ReceiveEnd	Execute receive end callback process.
	r_pmbus_app_int_Receive	Execute receive callback process.
	r_pmbus_app_int_TransmitEnd	Execute transmit end callback process.
	r_pmbus_app_int_Notify	Execute error detection callback process.
r_pmbus_nwk_master.c	r_pmbus_nwk_StartTx	Execute PMBus transmission start process.
	r_pmbus_nwk_StartRx	Execute PMBus reception start process.
	r_pmbus_nwk_ProcessTx	Execute PMBus transmission process.
	r_pmbus_nwk_ProcessRx	Execute PMBus reception process.
	r_pmbus_nwk_ProcessStop	Execute PMBus stopping process.
	r_pmbus_nwk_ProcessErrorStop	Execute the stopping process at PMBus error timing.
	r_pmbus_nwk_StartMaster	Start PMBus physical-layer operation.
	r_pmbus_nwk_StopMaster	Stop PMBus physical-layer operation.
	r_pmbus_nwk_ResetMaster	Initialize the physical layer of PMBus.
	r_pmbus_nwk_ProcessTimeout	Execute the timeout detection process.
	r_pmbus_nwk_ProcessNACK	Execute a NACK discovery operation.
	r_pmbus_nwk_ProcessArbLost	Execute arbitration-lost detection process.
	r_pmbus_nwk_GetRxPayloadSize	Get the received data size.
	r_pmbus_nwk_AddCrc8	Execute a CRC operation on a single file.
	r_pmbus_nwk_CalculatePEC	Execute a CRC operation on more than one datum.

Table 18 Smart Configurator Function list

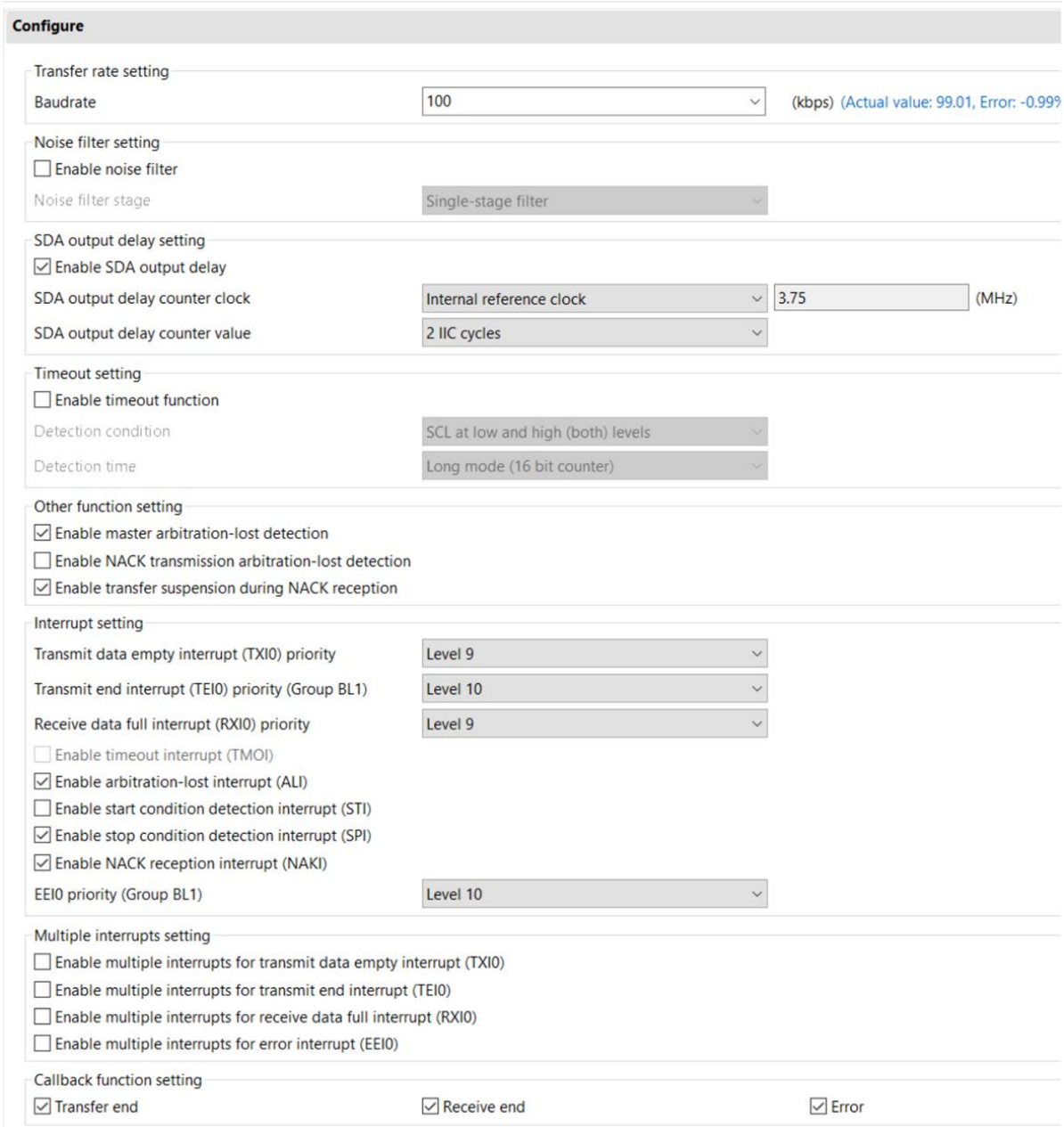
File name	Function name	Function	Change from diversion source*
Config_RIIC0_user.c	r_Config_RIIC0_transmit_interrupt	RIIC0 transmit buffer empty interrupt process.	Yes
	r_Config_RIIC0_receive_interrupt	RIIC0 receive buffer full interrupt process.	Yes
	r_Config_RIIC0_callback_transmitend	RIIC0 transmit end callback.	Yes
	r_Config_RIIC0_callback_receiveend	RIIC0 reception end callback.	Yes
	r_Config_RIIC0_callback_error	RIIC0 error-detection callback.	Yes
	r_User_RIIC0_callback_receive	RIIC0 receive callback.	New
Config_TMR0.c	R_Config_TMR0_Start	Execute TMR after the counter clear.	Yes
Config_TMR0_User.c	r_Config_TMR0_cmia0_interrupt	TMR compare match A interrupt process.	Yes
Config_RSCI11_user.c	r_Config_RSCI11_callback_transmitend	RSCI11 transmit end interrupt callback.	Yes
	r_Config_RSCI11_callback_receiveend	RSCI11 receive end interrupt callback.	Yes
	r_Config_RSCI11_callback_receiveerror	Error interrupt callback for RSCI11.	Yes

*: Refer to Customizing 5.1.4 PMBus Master Driver section for details.

5.1.4 Customizing PMBus Master Driver section

PMBus Master driver code (RIIC0,TMR,RSCI11) is generated by the Smart Configurator. For RIIC0 and TMR, some processes have been changed and added using the user code protection function of the Smart Configurator. The settings and changes for each smart configurator are shown below.

- Setting Smart Configurator RIIC0



- List of Changed Parts of RIIC0 Driver Codes Generated by Smart Configurator

Function Name	r_User_RIIC0_callback_receive()
File name	Config_RIIC0_user.c
Change Details	This is the callback function for the receive buffer full interrupt added by PMBus Middleware. r_pmbus_app_int_Receive() is executed to check and update PMBus Middleware status each time a receive buffer full interrupt is generated. Specify "g_riic0_rx_count" (number of received data) and g_riic0_rx_length as parameters. When the g_riic0_rx_count is "1" and block read process is in progress, the g_riic0_rx_length is updated to the receive data size. If the remaining receive data count becomes larger than 1, the g_riic0_state is reset to _09_IIC_MASTER_RECEIVE_DATA after AKBT bit of ICMR3 is cleared once. If the remaining receive data size is greater than 2, clear this register to ICMR3's WAIT setting.

Before change	After change
None.	

```
/* Start user code for adding. Do not edit comment generated here */
static void r_User_RIIC0_callback_receive(void)
{
    uint16_t u2_current_rx_count = g_riic0_rx_count;
    uint16_t u2_current_rx_length = g_riic0_rx_length;

    r_pmbus_app_int_Receive(u2_current_rx_count, &u2_current_rx_length);

    if (u2_current_rx_length != g_riic0_rx_length)
    {
        g_riic0_rx_length = u2_current_rx_length;

        if (1 < (g_riic0_rx_length - g_riic0_rx_count))
        {
            /* If remain receive data over 1, reverse ACK bit setting */
            RIIC0.ICMR3.BIT.ACKWP = 1U;
            RIIC0.ICMR3.BIT.ACKBT = 0U;

            /* If remain receive data over 1, reverse status to _09_IIC_MASTER_RECEIVES_DATA */
            g_riic0_state = _09_IIC_MASTER_RECEIVES_DATA;

            if (2 < (g_riic0_rx_length - g_riic0_rx_count))
            {
                /* If remain receive data over 2, reverse WAIT bit setting ACK bit setting */
                RIIC0.ICMR3.BIT.WAIT = 0U;
            }
        }
    }

}
/* End user code. Do not edit comment generated here */
```

Function Name	r_Config_RIIC0_receive_interrupt()
File name	Config_RIIC0_user.c
Change Details	Receive buffer full interrupt handler. Execute r_User_RIIC0_callback_receive() when _09_IIC_MASTER_RECEIVE_DATA == g_riic0_state and there is more data to receive or_0A_IIC_MASTER_RECEIVE_STOPPING = g_riic0_state.

Before change

```
    else if (_09_IIC_MASTER_RECEIVES_DATA == g_riic0_state)
    {
        if (g_riic0_rx_count < g_riic0_rx_length)
        {
            if (g_riic0_rx_count == (g_riic0_rx_length - 3))
            {
                RIIC0.ICMR3.BIT.WAIT = 1U;

                *gp_riic0_rx_address = RIIC0.ICDRR;
                gp_riic0_rx_address++;
                g_riic0_rx_count++;
            }
            else if (g_riic0_rx_count == (g_riic0_rx_length - 2))
            {
                RIIC0.ICMR3.BIT.ACKWP = 1U;
                RIIC0.ICMR3.BIT.ACKBT = 1U;

                *gp_riic0_rx_address = RIIC0.ICDRR;
                gp_riic0_rx_address++;
                g_riic0_rx_count++;

                g_riic0_state = _0A_IIC_MASTER_RECEIVES_STOPPING;
            }
            else
            {
                *gp_riic0_rx_address = RIIC0.ICDRR;
                gp_riic0_rx_address++;
                g_riic0_rx_count++;
            }
        }
    }
    else if (_0A_IIC_MASTER_RECEIVES_STOPPING == g_riic0_state)
    {
        RIIC0.ICSR2.BIT.STOP = 0U;
        RIIC0.ICCR2.BIT.SP = 1U;

        *gp_riic0_rx_address = RIIC0.ICDRR;
        gp_riic0_rx_address++;
        g_riic0_rx_count++;

        RIIC0.ICMR3.BIT.WAIT = 0U;
        g_riic0_state = _0B_IIC_MASTER_RECEIVES_STOP;
    }
    else
```

After change

```
    else if (_09_IIC_MASTER_RECEIVES_DATA == g_riic0_state)
    {
        if (g_riic0_rx_count < g_riic0_rx_length)
        {
            if (g_riic0_rx_count == (g_riic0_rx_length - 3))
            {
                RIIC0.ICMR3.BIT.WAIT = 1U;

                *gp_riic0_rx_address = RIIC0.ICDRR;
                gp_riic0_rx_address++;
                g_riic0_rx_count++;
            }
            else if (g_riic0_rx_count == (g_riic0_rx_length - 2))
            {
                RIIC0.ICMR3.BIT.ACKWP = 1U;
                RIIC0.ICMR3.BIT.ACKBT = 1U;

                *gp_riic0_rx_address = RIIC0.ICDRR;
                gp_riic0_rx_address++;
                g_riic0_rx_count++;

                g_riic0_state = _0A_IIC_MASTER_RECEIVES_STOPPING;
            }
            else
            {
                *gp_riic0_rx_address = RIIC0.ICDRR;
                gp_riic0_rx_address++;
                g_riic0_rx_count++;
            }
            /* Start user code */
            /* Add callback function for PMBus demo. */
            r_User_RIIC0_callback_receive();

            /* End user code */
        }
    }
    else if (_0A_IIC_MASTER_RECEIVES_STOPPING == g_riic0_state)
    {
        RIIC0.ICSR2.BIT.STOP = 0U;
        RIIC0.ICCR2.BIT.SP = 1U;

        *gp_riic0_rx_address = RIIC0.ICDRR;
        gp_riic0_rx_address++;
        g_riic0_rx_count++;

        RIIC0.ICMR3.BIT.WAIT = 0U;
        g_riic0_state = _0B_IIC_MASTER_RECEIVES_STOP;
        /* Start user code */
        /* Add callback function for PMBus demo. */
        r_User_RIIC0_callback_receive();

        /* End user code */
    }
```

Function Name	r_Config_RIIC0_transmi_interrupt()
File name	Config_RIIC0_user.c
Change Details	Transmit buffer empty interrupt handler. After adding the g_riic0_tx_length to the globals, After transmitting the slave address with _0D_IIC_MASTER_TRANSMIT == g_riic0_mode_flag, save the g_riic0_tx_count in the g_riic0_tx_length.

Before change

```
/* Start user code for global. Do not edit comment generated here */

/* End user code. Do not edit comment generated here */
```

```
        if (_01_IIC_MASTER_SENDS_ADR_7_W == g_riic0_state)
        {
            RIIC0.ICDRT = (uint8_t)(g_riic0_slave_address << 1U);
            g_riic0_state = _05_IIC_MASTER_SENDS_DATA;

        }
```

After change

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_riic0_tx_length;                    /* RIIC0 transmit data length */

/* End user code. Do not edit comment generated here */
```

```
        if (_01_IIC_MASTER_SENDS_ADR_7_W == g_riic0_state)
        {
            RIIC0.ICDRT = (uint8_t)(g_riic0_slave_address << 1U);
            g_riic0_state = _05_IIC_MASTER_SENDS_DATA;
            /* Start user code */
            g_riic0_tx_length = g_riic0_tx_count;

            /* End user code */
        }
```

Function Name	r_Config_RIIC0_callback_transmitend()
File name	Config_RIIC0_user.c
Change Details	This is a callback function for transmission completion interrupts. Execute r_pmbus_app_int_TransmitEnd() to check the status of PMBus Middleware and to refresh the status after the transmit end interrupt. Specify "g_riic0_tx_length" (total number of data to be sent) as the parameter. 0 Execute Quick Command process.

Before change

```
385 static void r_Config_RIIC0_callback_transmitend(void)
386 {
387     /* Start user code for r_Config_RIIC0_callback_transmitend. Do not edit comment generated here */
388     /* End user code. Do not edit comment generated here */
389 }
```

After change

```
402 static void r_Config_RIIC0_callback_transmitend(void)
403 {
404     /* Start user code for r_Config_RIIC0_callback_transmitend. Do not edit comment generated here */
405     r_pmbus_app_int_TransmitEnd(g_riic0_tx_length);
406
407     /* End user code. Do not edit comment generated here */
408 }
```

Function Name	r_Config_RIIC0_callback_receiveend ()
File name	Config_RIIC0_user.c
Change Details	Callback function upon receive end. Execute r_pmbus_app_int_ReceiveEnd() to check and update the status of PMBus Middleware after Stop condition interrupt after receive end of all the data.

Before change

```
398 static void r_Config_RIIC0_callback_receiveend(void)
399 {
400     /* Start user code for r_Config_RIIC0_callback_receiveend. Do not edit comment generated here */
401     /* End user code. Do not edit comment generated here */
402 }
```

After change

```
417 static void r_Config_RIIC0_callback_receiveend(void)
418 {
419     /* Start user code for r_Config_RIIC0_callback_receiveend. Do not edit comment generated here */
420     r_pmbus_app_int_ReceiveEnd();
421
422     /* End user code. Do not edit comment generated here */
423 }
```

Function Name	r_Config_RIIC0_callback_error()
File name	Config_RIIC0_user.c
Change Details	This is a callback function for error detection interrupts. Execute the following process for each parameter status. For "MD_ERROR1" (Arbitration Lost): Execute the r_pmbus_app_int_Notify (E_PMBUS_INT_EVENT_ARB_LOST) to refresh PMBus Middleware status. When "MD_ERROR3" (NACK detected): Execute the r_pmbus_app_int_Notify (E_PMBUS_INT_EVENT_NACK) to refresh PMBus Middleware status. When "MD_ERROR4" (a Stop condition outside the driver-sequence is detected): If STOP of ICSR2 is 1, If the g_riic0_mode_flag is_0C_IIC_MASTER_RECEIVE to refresh PMBus Middleware status, set RIIC0.ICMR3.BIT.WAIT to 0, and then execute r_pmbus_app_int_Notify (E_PMBUS_INT_EVENT_STOP_ERROR). If the g_riic0_mode_flag is other than the above, execute r_pmbus_app_int_Notify (E_PMBUS_INT_EVENT_STOP). *MD_ERROR2 (TimeoutError detect) is not used in PMBus Middleware.

Before change

```
        case MD_ERROR1:
        {
            /* Start user code for arbitration-lost error. Do not edit comment generated here
*/

            /* End user code. Do not edit comment generated here */
            RIIC0.ICSR2.BIT.AL = 0U;
            break;
        }
```

```
        case MD_ERROR3:
        {
            /* Start user code for NACK signal. Do not edit comment generated here */

            /* End user code. Do not edit comment generated here */
            break;
        }
```

```
        case MD_ERROR4:
        {
            /* Start user code for communication sequence error. Do not edit comment
generated here */

            /* End user code. Do not edit comment generated here */
            break;
        }
```

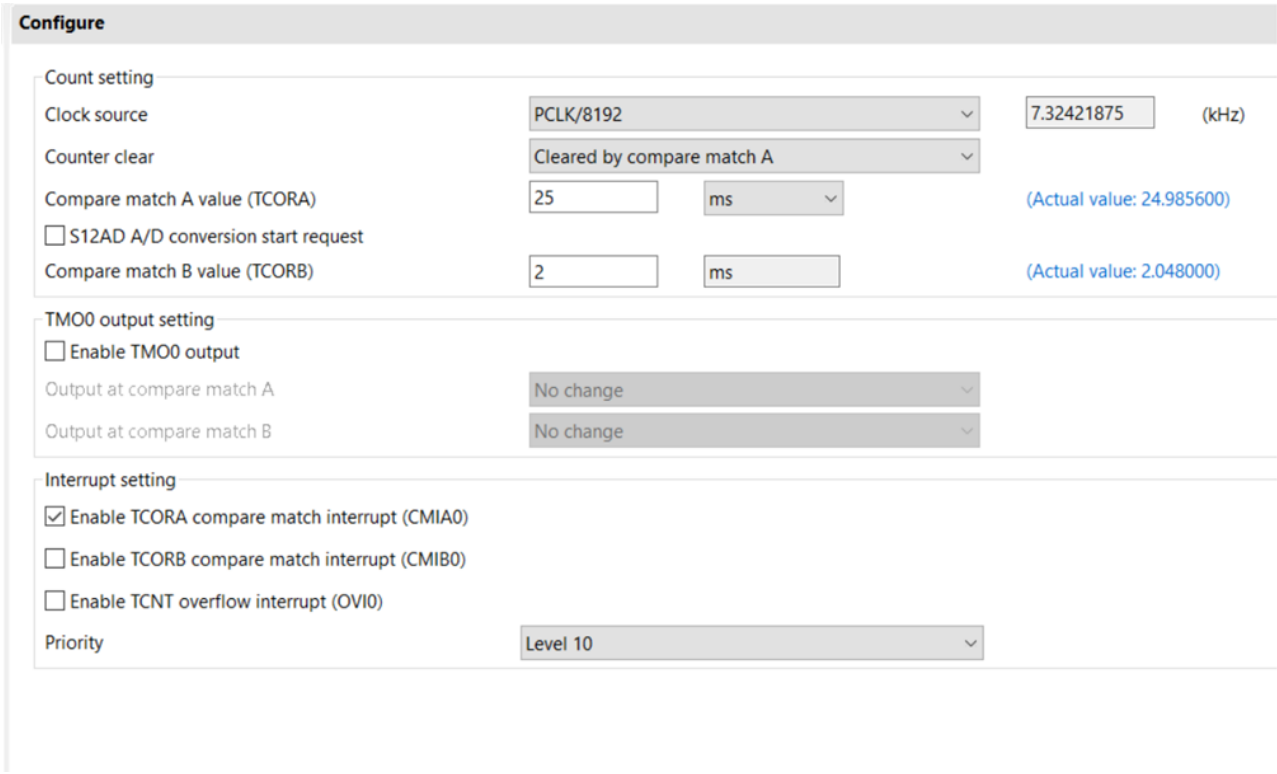
After change

```
        case MD_ERROR1:
        {
            /* Start user code for arbitration-lost error. Do not edit comment generated here
*/
            r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_ARB_LOST);

            /* End user code. Do not edit comment generated here */

            RIIC0.ICSR2.BIT.AL = 0U;
            break;
        }
```

```
        case MD_ERROR3:
        {
            /* Start user code for NACK signal. Do not edit comment generated here */
            r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_NACK);

            /* End user code. Do not edit comment generated here */
            break;
        }
```

```
        case MD_ERROR4:
        {
            /* Start user code for communication sequence error. Do not edit comment generated
here */
            if (1 == RIIC0.ICSR2.BIT.STOP)
            {
                RIIC0.ICSR2.BIT.NACKF = 0U;
                RIIC0.ICSR2.BIT.STOP = 0U;

                if (_0C_IIC_MASTER_RECEIVE == g_riic0_mode_flag)
                {
                    /* If STOP conodition detect before last data received is illegal. */
                    RIIC0.ICMR3.BIT.WAIT = 0U;
                    r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_STOP_ERROR);
                }
                else
                {
                    r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_STOP);
                }
            }

            /* End user code. Do not edit comment generated here */
            break;
        }
```

- Setting smart configurator TMR



- List of Changed Parts of TMR Driver Codes Generated by Smart Configurator

Function Name	R_Config_TMR0_Start()
File name	Config_TMR0.c
Change Details	Add a process for clearing TMR0.TCNT.

Before change

```
void R_Config_TMR0_Start(void)
{


    /* Enable TMR0 interrupt */
    IR(TMR0, CMIA0) = 0U;
    IEN(TMR0, CMIA0) = 1U;

    /* Start counting */
    TMR0.TCCR.BYTE = _08_TMR_CLK_SRC_PCLK | _06_TMR_PCLK_DIV_8192;
}
```

After change

```
void R_Config_TMR0_Start(void)
{
    /* Start user code */
    TMR0.TCNT = 0U;

    /* End user code */
    /* Enable TMR0 interrupt */
    IR(TMR0, CMIA0) = 0U;
    IEN(TMR0, CMIA0) = 1U;

    /* Start counting */
    TMR0.TCCR.BYTE = _08_TMR_CLK_SRC_PCLK | _06_TMR_PCLK_DIV_8192;
}
```

Function Name	r_Config_TMR0_cmia0_interrupt()
File name	Config_TMR0_user.c
Change Details	This is a callback function for compare match interrupts. Execute the r_pmbus_app_int_Notify (E_PMBUS_INT_EVENT_TIMEOUT) to refresh PMBus Middleware status.

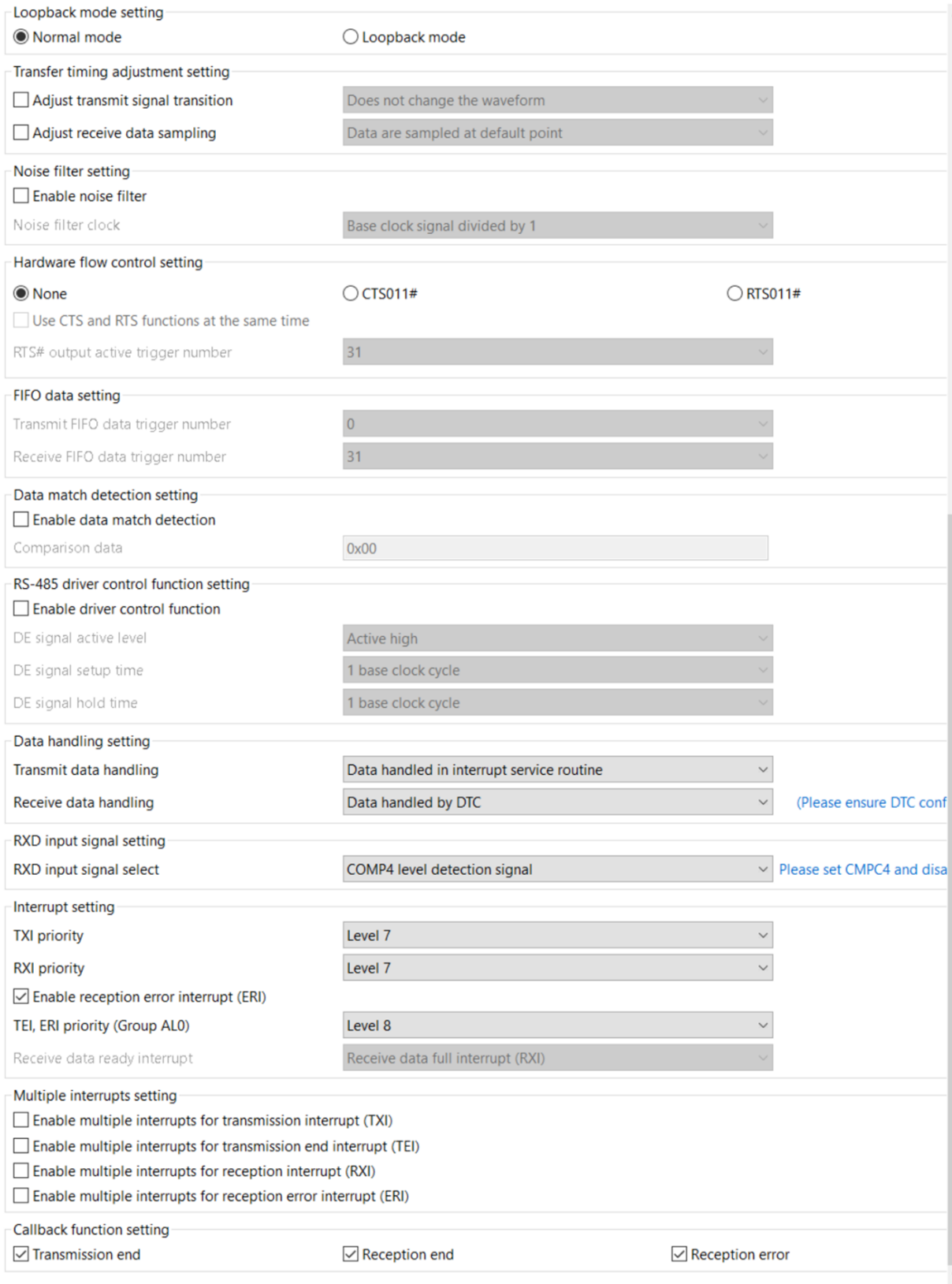
Before change

```
static void r_Config_TMR0_cmia0_interrupt(void)
{
    /* Start user code for r_Config_TMR0_cmia0_interrupt. Do not edit comment generated here */

    /* End user code. Do not edit comment generated here */
}
```

After change

```
static void r_Config_TMR0_cmia0_interrupt(void)
{
    /* Start user code for r_Config_TMR0_cmia0_interrupt. Do not edit comment generated here */
    r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_TIMEOUT);
    /* End user code. Do not edit comment generated here */
}
```

- Setting smart configurator RSCI11



- List of Changed Parts of RSCI11 Driver Codes Generated by Smart Configurator

Function Name	r_Config_RSCI11_callback_transmitend
File name	Config_RSCI11_user.c
Change Details	Execute the pmbus_demo_update_status (E_MAIN_STATUS_RESPONSE_COMP).

Before change

```
168 static void r_Config_RSCI11_callback_transmitend(void)
167 {
168     /* Start user code for r_Config_RSCI11_callback_transmitend. Do not edit comment
    generated here */

169     /* End user code. Do not edit comment generated here */
170 }
```

After change

```
167 static void r_Config_RSCI11_callback_transmitend(void)
168 {
169     /* Start user code for r_Config_RSCI11_callback_transmitend. Do not edit comment generated
    here */
170     pmbus_demo_update_status(E_MAIN_STATUS_RESPONSE_COMP);
171
172     /* End user code. Do not edit comment generated here */
173 }
```

Function Name	r_Config_RSCI11_callback_receiveend
File name	Config_RSCI11_user.c
Change Details	Execute pmbus_demo_uart_recv_callback().

Before change

```
179 static void r_Config_RSCI11_callback_receiveend(void)
180 {
181     /* Start user code for r_Config_RSCI11_callback_receiveend. Do not edit comment generated
    here */

182     /* End user code. Do not edit comment generated here */
183 }
```

After change

```
182 static void r_Config_RSCI11_callback_receiveend(void)
183 {
184     /* Start user code for r_Config_RSCI11_callback_receiveend. Do not edit comment generated
    here */
185     pmbus_demo_uart_recv_callback();
186
187     /* End user code. Do not edit comment generated here */
188 }
```

Function Name	r_Config_RSCI11_callback_receiveerror
File name	Config_RSCI11_user.c
Change Details	Execute the pmbus_demo_notify_event (E_UART_EVENT_RECEIVE_ERROR).

Before change

```
192 static void r_Config_RSCI11_callback_receiveerror(void)
193 {
194     /* Start user code for r_Config_RSCI11_callback_receiveerror. Do not edit comment
    generated here */

195     /* End user code. Do not edit comment generated here */
196 }
```

After change

```
197 static void r_Config_RSCI11_callback_receiveerror(void)
198 {
199     /* Start user code for r_Config_RSCI11_callback_receiveerror. Do not edit comment
    generated here */
200     pmbus_demo_notify_event(E_UART_EVENT_RECEIVE_ERROR);
201
202     /* End user code. Do not edit comment generated here */
203 }
```

5.1.5 PMBus Master Data Types and Structure list

The following table lists the Data Types and Structure used in this control program.

- Data Types and Structure for PMBus Master Application Functions

e_main_status_t

Enumeration Name	e_main_status_t	
Description	This enumeration is used to indicate the main status of the PMBus Master Application.	
Declared header file	r_pmbus_demo_master.h	
Remarks	-	
Element name	**Description**	**Value**
E_MAIN_STATUS_IDLE	IDLE state. Waits for UART data to be received.	0
E_MAIN_STATUS_REV_BUSY	Receiving UART data.	1
E_MAIN_STATUS_REV_COMP	Completed receiving UART data required for command process.	2
E_MAIN_STATUS_PMBUS_BUSY	PMBUS API is being executed. (Reserve)	3
E_MAIN_STATUS_PMBUS_COMP	PMBUS API operation has ended.	4
E_MAIN_STATUS_PMBUS_ACTIVE	PMBUS API is being executed.	5
E_MAIN_STATUS_RESPONSE_BUSY	Command process results are being sent via UART.	6
E_MAIN_STATUS_RESPONSE_COMP	UART data transmission completed.	7
E_MAIN_STATUS_ERROR_BUSY	Error process. (Reserve)	8
E_MAIN_STATUS_ERROR_COMP	Error process completed. (Reserve)	9
E_MAIN_STATUS_MAX	The maximum value of the main status.	10

e_uart_event_t

Enumeration Name	e_uart_event_t	
Description	This enumeration is used to specify the message number to respond by UART.	
Declared header file	r_pmbus_demo_master.h	
Remarks	-	
Element name	**Description**	**Value**
E_UART_EVENT_COMMAND_END	Specify "PMBUS COMMAND END" as the transmission string.	0
E_UART_EVENT_RESP_START	Specify ">>PMBUS_RESPONSE_START" as the transmission string.	1
E_UART_RECEIVE_FULL	Specify ">>ERROR:UART RECEIVE BUFFER IS FULL" as the transmission string.	2
E_UART_RECEIVE_ERROR	Specify "ERROR:UART RECEIVE ERROR" as the transmission string.	3
E_UART_EVENT_FORMAT_ERROR	Specify "ERROR:PMBUS FORMAT ERROR" as the transmission string.	4
E_UART_EVENT_COMMAND_INVALIDATE	Specify "ERROR:PMBUS COMMAND NOT SUPPORT" as the transmission string.	5
E_UART_EVENT_COMMAND_FAIL	Specify "ERROR:PMBUS API RETURN ERROR" as the transmission string.	6
E_UART_EVENT_MAX	The maximum value of the message number.	7

e_response_type_t

Enumeration Name	e_response_type_t	
Description	This enumeration type is used to specify the notation type of the numeric expression that is responded to via UART. It is used to specify the notation type for each PMBUS command registered in s_st_pmbus_command_check_table.	
Declared header file	r_pmbus_demo_master.h	
Remarks	-	
Element name	**Description**	**Value**
E_RESPONSE_TYPE_HEX	Specifies hexadecimal notation.	0
E_RESPONSE_TYPE_DECIMAL	Specifies signed decimal notation.	1
E_RESPONSE_TYPE_DECIMAL_UNSIGNED	Specifies unsigned decimal notation.	2
E_RESPONSE_TYPE_MAX	The maximum value of notation type.	3

e_app_varidate_info_t

Enumeration Name	e_app_varidate_info_t	
Description	This enumeration type is used to indicate whether the PMBUS command received via UART is valid or invalid. It is used to specify whether each PMBUS command registered in s_st_pmbus_command_check_table is valid or invalid.	
Declared header file	r_pmbus_demo_master.h	
Remarks	-	
Element name	**Description**	**Value**
E_COMMAND_INVALIDATE	Specifies an invalid command.	0
E_COMMAND_VALIDATE	Specifies a valid command.	1

st_app_command_format_t

Structure Name	st_app_command_format_t
Description	This structure manages the format information of PMBUS commands processed by the PMBus Master Application. It is used to check whether the data received via UART matches the format of the corresponding PMBUS command, and to determine the format to respond to via UART.
Declared header file	r_pmbus_demo_master.h
Remarks	-
Member name	**Description**
uint8_t u1_validate	Command valid/invalid information. (E_COMMAND_INVALIDATE or E_COMMAND_VALIDATE)
uint8_t u1_command	PMBUS command code. (0x00 to 0xFF)
uint8_t u1_transaction	Transaction type of PMBUS command. (Combination of transaction code (PMBUS_TRANS_XXX))
uint16_t u2_tx_data_size	The amount of data sent by the PMBUS command. (0x00 to 0xFF)
uint16_t u2_rx_data_size	The amount of data received by the PMBUS command. (0x00 to 0xFF)
e_response_type_t e_resp_type	Specifies the numeric representation when returning received data of PMBUS command via UART. (E_RESPONSE_TYPE_HEX, E_RESPONSE_TYPE_DECIMAL or E_RESPONSE_TYPE_DECIMAL_UNSIGNED)

st_pmbus_data_t

Structure Name	st_pmbus_data_t
Description	This structure manages PMBUS commands information received from UART.
Declared header file	r_pmbus_demo_master.h
Remarks	-
Member name	**Description**
uint8_t u1_command	The command code of the PMBUS command to execute.
uint8_t u1_transaction	Transaction code of the PMBUS command to be executed. (Combination of transaction codes (PMBUS_TRANS_XXX))
uint8_t u1_slave_addr	The slave address to which the PMBUS command is to be communicated.
uint16_t u2_data_size	Total number of transmission data for PMBUS command to be executed. (Reserve)
uint16_t u2_tx_index	The current number of transmission data being processed by the PMBUS command.
uint16_t u2_command_index	The index of the s_st_pmbus_command_check_table corresponding to the PMBUS command to be executed. (0 to (COMMAND_TABLE_SIZE – 1))
uint8_t u1_tx_buf[PMBUS_TX_BUF_SIZE]	A buffer that stores the transmission data to be processed by PMBUS commands.
bool b_direction	The communication direction to be handled by the PMBUS command. (false: receive protocol or true: transmit or transmit/receive protocol)

st_app_uart_rx_t

Structure Name	st_app_uart_rx_t
Description	This structure manages PMBUS commands information received from UART.
Declared header file	r_pmbus_demo_master.h
Remarks	-
Member name	**Description**
uint8_t u1_recv_buf[UART_RX_BUF_SIZE]	Buffer for storing received UART data.
bool b_recv_flag	The receive end status of one line of data. (false:(receive is not end) or true:(receive end (set when line feed code "0x0A" is received)))

- Data Types and Structure for PMBus Master API

e_pmbus_packet_result_m_t

Enumeration Name	e_pmbus_packet_result_m_t	
Description	This enumeration type is used to indicate the execution result of PMBus communication (master). It is used as the argument type of each PMBus Master API. It indicates the details of the error cause when the PMBus API return value is PMBUS_RET_ERROR.	
Declared header file	r_pmbus_app_master.h	
Remarks	-	
Element name	**Description**	**Value**
E_PMBUS_PACKET_M_OK	Normal operation.	0
E_PMBUS_PACKET_M_DATA_SIZE_ERROR	Detects packet size error.	1
E_PMBUS_PACKET_M_PEC_ERROR	Detects error in PEC operation.	2
E_PMBUS_PACKET_M_TIMEOUT	Detects timeout error. ($T_{TIMEOUT}$ error detected)	3
E_PMBUS_PACKET_M_ARB_LOST	Detects arbitration-lost error.	4
E_PMBUS_PACKET_M_NACK	Detects NACK receive.	5
E_PMBUS_PACKET_M_INTERNAL_ERROR	Detects internal error.	6

st_pmbus_cfg_m_t

Structure Name	st_pmbus_cfg_m_t
Description	This is a structure for PMBus Master configuration data. It is used as the argument type for R_PMBUS_Master_Open. It is used to register the configuration data to the internal global variables of the PMBus Master Middleware.
Declared header file	r_pmbus_app_master.h
Remarks	-
Member name	**Description**
uint16_t u2_rx_size	*The dimensions of the p_u1_rx_buf. (1 to 35)
uint8_t *p_u1_rx_buf	Pointer to the receive data storage buffer. The data received by R_PMBUS_Master_Read or R_PMBUS_Master_WriteRead is stored in this buffer. The data is multiplied by the *p_u1_rx_buf and *u2_rx_len arguments, and then returned to API caller.
uint16_t u2_tx_size	*The dimensions of the p_u1_tx_buf. (1 to 35)
uint8_t *p_u1_tx_buf	Pointer to the transmit data storage buffer. The data of the u1_command, u2_tx_len, and *p_u1_tx_buf specified by R_PMBUS_Master_Write or R_PMBUS_Master_WriteRead parameter is stored in this buffer according to the protocol corresponding to the command code. Then, the data is sent.

- Data Types and Structure for PMBus Master Middleware function

e_pmbus_nwk_status_m_t

Enumeration Name	e_pmbus_nwk_status_m_t	
Description	This enumeration type indicates the internal state of the PMBus network layer (master). It is used to manage the internal state of the PMBus Master Middleware.	
Declared header file	r_pmbus_app_master.h	
Remarks	-	
Element name	**Description**	**Value**
E_PMBUS_NWK_STATUS_M_IDLE	Waiting for new packet receive.	0
E_PMBUS_NWK_STATUS_M_RX	Receiving packet.	1
E_PMBUS_NWK_STATUS_M_TX	Transmitting packet.	2
E_PMBUS_NWK_STATUS_M_TX_QUICK	Transmitting quick command.	3
E_PMBUS_NWK_STATUS_M_TX_BLOCK	Transmitting block.	4
E_PMBUS_NWK_STATUS_M_RX_BLOCK	Receiving block.	5
E_PMBUS_NWK_STATUS_M_ENDING	End of packet.	6
E_PMBUS_NWK_STATUS_M_ERROR	Detects packet error.	7

e_pmbus_int_event_m_t

Enumeration Name	e_pmbus_int_event_m_t	
Description	This enumeration type indicates the cause of an I2C error detection interrupt. It is used to execute process for each interrupt cause in the internal process of the PMBus Master Middleware.	
Declared header file	r_pmbus_app_master.h	
Remarks	-	
Element name	**Description**	**Value**
E_PMBUS_INT_EVENT_M_NONE	Interrupt not detected	0
E_PMBUS_INT_EVENT_M_ARB_LOST	Detects arbitration lost.	1
E_PMBUS_INT_EVENT_M_TIMEOUT	Detects timeout.	2
E_PMBUS_INT_EVENT_M_NACK	Detects NACK receive.	3
E_PMBUS_INT_EVENT_M_START	Detects start condition.	4
E_PMBUS_INT_EVENT_M_STOP	Detects stop condition.	5
E_PMBUS_INT_EVENT_M_STOP_ERROR	Detects unexpected Stop condition.	6

st_pmbus_nwk_ctrl_m_t

Enumeration Name	st_pmbus_nwk_ctrl_m_t
Description	This is a structure that manages PMBus network layer (master) parameters. It is used to manage the communication status inside the PMBus Master Middleware.
Declared header file	r_pmbus_app_master.h
Remarks	-
Member name	**Description**
volatile e_pmbus_nwk_status_m_t e_m_status	Network layer status
uint8_t u1_current_addr	Currently executing slave address
uint8_t u1_current_cmd	Currently executing command
uint16_t u2_rx_index	Current number of received data bytes
uint16_t u2_rx_len	Number of data bytes to be received
uint16_t u2_rx_size	*p_u1_rx_buf Size
uint8_t *p_u1_rx_buf	Pointer to receive data storage buffer
uint16_t u2_tx_index	Current number of transmission data bytes
uint16_t u2_tx_len	Number of data bytes to be transmitted
uint16_t u2_tx_size	*p_u1_tx_buf Size
uint8_t *p_u1_tx_buf	Pointer to transmit data storage buffer
uint8_t u1_pec	Present PEC calculation

st_pmbus_ctrl_m_t

Structure Name	st_pmbus_ctrl_m_t
Description	This is the control data structure of the PMBus Middleware (master). It is used to manage the PMBus Master Middleware setting information and communication status.
Declared header file	r_pmbus_app_master.h
Remarks	-
Member name	**Description**
st_pmbus_nwk_ctrl_m_t st_nwk_ctrl_m	Parameter-managed struct of PMBus network Layer (master)
volatile e_pmbus_packet_result_m_t e_pmbus_result_m	Executing PMBus communication (master)
bool b_open_flag	OPEN status (No false: Open or true: Open).
bool b_pec_flag	PEC enable/disable information (false: disable, true: enable)
uint8_t u1_own_slave_addr	Its own slave address. (Not used in demonstration systems.)

5.1.6 PMBus Master global variables List

The following table lists the global variables used in this control program.

Table 19 PMBus Master Application global variables list

File name	Global Variables	Usage
r_pmbus_demo_master.c	static const st_app_command_format_t s_st_pmbus_command_check_table[COMMAND_TABLE_SIZE]	A const table used to check the format of PMBUS commands received via UART.
	static const char * sp_pmbus_message_return_res	A const value that stores the string "return code:" that is added when storing the PMBUS execution result in the UART transmit buffer.
	static const char * sp_pmbus_message_packet_res	A const value that stores the string "packet result:" that is added when storing the PMBUS execution result in the UART transmit buffer.
	static st_app_uart_rx_t s_st_uart_rx_data	Structure variable that manages UART receive data information. Analyzes the receive data stored in u1_recv_buf, a member of this structure, and controls PMBUS process.
	static uint8_t s_u1_uart_tx_buf[UART_TX_BUF_SIZE]	Buffers for storing UART transmit data.
	static uint8_t s_u1_uart_rx_buf[UART_RX_BUF_SIZE]	A buffer that stores UART received data. The received data stored in this buffer is analyzed to detect the line feed code that separates the received data. This buffer is used as a ring buffer.
	static uint16_t s_u2_uart_rx_index;	Variable that manages how many data items are currently received in UART.
	static uint16_t s_u2_rx_r_index	This parameter indicates the most recent data position at which the receive data buffer (s_u1_uart_rx_buf) was read.
	static uint16_t s_u2_rx_w_index	Indicates the most recent data position stored in the receive data buffer (s_u1_uart_rx_buf).
	static uint8_t s_u1_uart_rx_relay_buf[UART_RELAY_BUF_SIZE]	Buffer for storing UART receive data. After the UART receive interrupt handler stores the received data in this buffer, the main process Copy the data to s_u1_uart_rx_buf.
	volatile static e_main_status_t s_e_main_status	Variables for managing the internal process state of an application.
	volatile static uint16_t s_u2_seq_index	Variables for managing the UART command receive process status. This variable determines how much data required for PMBUS communication has been received.
	static st_pmbus_data_t s_st_pmbus_data	Structural variables that manage the data for PMBUS commands. The data stored in this structure variable is used as an argument to be passed to the PMBUS API.
	static uint8_t s_u1_pmbus_tx_buf[PMBUS_TX_BUF_SIZE]	Buffers for storing the transmitted data for PMBUS. This buffer is set to the member *p_u1_tx_buf of s_user_pmbus_cfg and used inside the PMBUS Middleware.
	static uint8_t s_u1_pmbus_rx_buf[PMBUS_RX_BUF_SIZE]	Buffers to store received data for PMBUS. This buffer is set to the member *p_u1_rx_buf of s_user_pmbus_cfg and used inside the PMBUS Middleware.

	static uint16_t s_u2_pmbus_rx_size	Number of data received by PMBUS. Specify this as an argument to R_PMBUS_Master_Read and R_PMBUS_Master_WriteRead to store the amount of data received via PMBUS.
	static uint8_t s_u1_pmbus_temp_rx_buf[PMBUS_RX_BUF_SIZE]	Buffer for storing PMBUS receive data specified when executing the PMBUS API. Specify as an argument to R_PMBUS_Master_Read and R_PMBUS_Master_WriteRead to store data received via PMBUS.
	static e_pmbus_packet_result_m_t s_e_packet_result	Variable that stores the execution result (packet result) of the PMBUS API. Specify as an argument for each PMBUS API to store the execution result (packet result) of the PMBUS API.
	static uint8_t s_u1_pmbus_ret	Variable that stores the result of executing PMBUS API (return code).
	static st_pmbus_cfg_m_t s_user_pmbus_cfg	Variable to store the configuration data to be registered in R_PMBUS_Master_Open.

Table 20 Global variables for PMBus Master Middleware

File name	Global Variables	Usage
r_pmbus_master_app.c	static st_pmbus_ctrl_m_t g_st_pmbus_ctrl	Global variable that manages the control information of PMBUS Master Middleware. It is used only within PMBUS Master Middleware.

5.1.7 PMBus Master macro Definition List

The following table lists the macro definitions used in this control program.

Table 21 PMBus Master Application macro definition list

File name	Macro name	Usage	Defined Value
r_pmbus_demo_master.h	PMBUS_APP_CMD_xxx	・Define the command code for PMBUS. (Refer to below for macro names)	-
		PMBUS_APP_CMD_OPERAION : OPERAION command	0x01
		PMBUS_APP_CMD_ON_OFF_CONFIG : ON_OFF_CONFIG command	0x02
		PMBUS_APP_CMD_CLEAR_FAULTS : CLEAR FAULTS command	0x03
		PMBUS_APP_CMD_STATUS_FAN_1_2 : STATUS_FAN_1_2 command	0x81
		PMBUS_APP_CMD_READ_VOUT : READ_VOUT command	0x8B
		PMBUS_APP_CMD_READ_IOUT : READ_IOUT command	0x8C
		PMBUS_APP_CMD_READ_FAN_SPEED_1 : REA_DFAN_SPEED_1 command	0x90
		PMBUS_APP_CMD_READ_FREQUENCY : READ_FREQUENCY command	0x95
		PMBUS_APP_CMD_RESERVED : RESERVED command	0x09
		PMBUS_APP_CMD_PMBUS_REVISION : PMBUS_REVISION command	0x98
		PMBUS_APP_CMD_STORE_DEFAULT_CODE : STORE_DEFAULT_CODE command	0x13
		PMBUS_APP_CMD_FAN_COMMAND_1 : FAN_COMMAND_1 command	0x38
		PMBUS_APP_CMD_READ_EOUT : READ_EOUT command	0x87
		PMBUS_APP_CMD_PAGE_PLUS_WRITE : PAGE_PLUS_WRITE command	0x05

	UART_TX_BUF_SIZE	Max. of buffers for storing data to be sent by UART.	256
	UART_RX_BUF_SIZE	Max. of buffers for storing data to be received by UART.	256
	UART_RELAY_BUF_SIZE	Max. buffers to temporarily store data received by UART.	3
	PMBUS_TX_BUF_SIZE	Max. value of buffers for storing data to be sent by PMBUS.	40
	PMBUS_RX_BUF_SIZE	Max. value of buffers for storing data to be received by PMBUS.	40
	COMMAND_TABLE_SIZE	The total number of commands to be registered in the command check table (s_st_pmbus_command_check_table).	14
	SEQ_INDEX_xxx	・Define the UART command receive process state. (see below for macro names)	-
		SEQ_INDEX_IDLE : Waiting for UART receive.	0
		SEQ_INDEX_SLAVE_ADDR : Receive end up to slave address.	1
		SEQ_INDEX_RX : READ/WRITE information has been received.	2
		SEQ_INDEX_COMMAND : Command code has been received.	3
		SEQ_INDEX_WRITE_DATA : PMBUS transmission data is being received or has been received.	4
		SEQ_INDEX_MAX : The maximum number of PMBUS transmission data that can be processed by this application.	256
	UART_RESPONSE_TIMEOUT_TIME	Soft timer count value that waits until UART communication is completed when PMBUS receive is returned in UART	0x50000
	RET_OK	Return value of the application internal function. Normal end.	0
	RET_ERROR	Return value of the application internal function. Abnormal end.	1

Table 22 PMBus Master API macro definition list

File name	Macro name	Usage	Defined Value
r_pmbus_app_master.h	PMBUS_RET_xxx	・Error code returned from PMBus middleware API. (See below for macro names)	-
		PMBUS_RET_OK : Normal end.	0
		PMBUS_RET_ERROR : Abnormal end. See the st_pmbus_cfg_m_t.e_pmbus_result_m for more information about the source.	1
		PMBUS_RET_PARAM : Specified argument is invalid.	2
		PMBUS_RET_NOT_OPENED : No OPEN.	3
		PMBUS_RET_OPENED : Already OPEN.	4

Table 23 PMBus Master Middleware function macro definition list

File name	Macro name	Usage	Defined Value
r_pmbus_app_master.h	PMBUS_TRANS_xxx	・Defines the transaction code used to determine the protocol supported by each command code. (See below for macro names)	-
		PMBUS_TRANS_RESERVED : Command code is RESERVED.	0x00

		PMBUS_TRANS_READ_BYTE : Command Code Supports READ BYTE Transactions.	0x01
		PMBUS_TRANS_READ_WORD : Command Code Supports READ WORD Transactions.	0x02
		PMBUS_TRANS_BLOCK_READ : Command Code Supports BLOCK READ Transactions.	0x03
		PMBUS_TRANS_SEND_BYTE : Command Code Supports SEND BYTE Transactions.	0x10
		PMBUS_TRANS_WRITE_BYTE : Command Code Supports WRITE BYTE Transactions	0x20
		PMBUS_TRANS_WRITE_WORD : Command Code Supports WRITE WORD Transactions.	0x30
		PMBUS_TRANS_BLOCK_WRITE : Command Code Supports BLOCK WRITE Transactions.	0x40
		PMBUS_TRANS_WRITE_QUICK : Command Code Supports Write Quick Command Transactions.	0x50
		PMBUS_TRANS_PROCESS_CALL : Command Code Supports PROCESS CALL Transactions.	0x60
		PMBUS_TRANS_BLOCK_PROCESS_CALL : Command Code Supports Block Write-Block Read Process Call Transactions.	0x70
	PMBUS_COMMAND_CODE_MAX	The maximum number of commands supported by the PMBus middleware.	256
	PMBUS_BLOCK_SIZE_MIN	Min. amount of data that can be sent/received by Block command will.	1
	PMBUS_BLOCK_SIZE_MAX	Max. amount of data that can be sent/received by Block command will.	32
	PMBUS_BUF_SIZE_MIN	Min. of buffer size that can be registered in PMBUS Middleware during Open.	1
	PMBUS_BUF_SIZE_MAX	Max. of buffer size that can be registered in PMBUS Middleware during Open. (Max. number of data to write during Block Read/Block Write (32) + Command code (1) + Number of data to write/Number of data to read (1) + PEC (1))	PMBUS_BLOCK_SIZE_MAX + 3
	PMBUS_CRC8_USE_IP	Defined value that specifies the PEC calculation method. Set to "1 (uses the calculator built into the MCU)" or "0 (specifies whether to use a table for calculation)." If you want to use the calculator built into the MCU, you must implement code that uses a CRC calculator in the r_pmbus_nwk_AddCrc8() function and then change the setting to "1 (uses the calculator built into the MCU)."	0
	PMBUS_ALERT_RESPONSE_ADDR	The slave address (ARA) of the communication destination for receiving ALERT information specified when R_PMBUS_Master_ReceiveARA is executed. The value defined is compliant with the SMBus specifications.	0x0C
	PMBUS_SLAVE_ADDR_MAX	Max. number of slave address that can be specified in the PMBUS master API arguments.	0x80

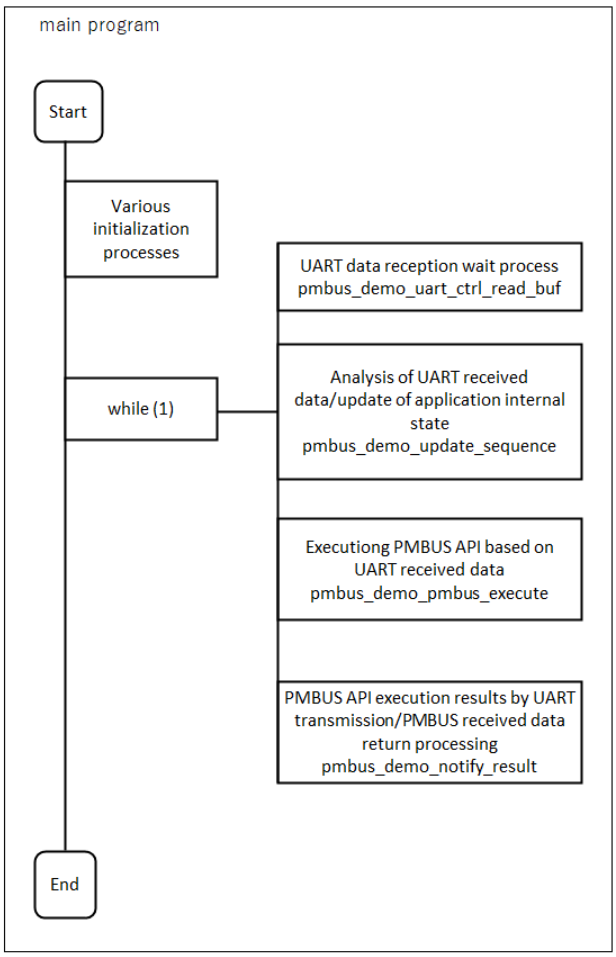
5.1.8 PMBus Master Control Flowchart

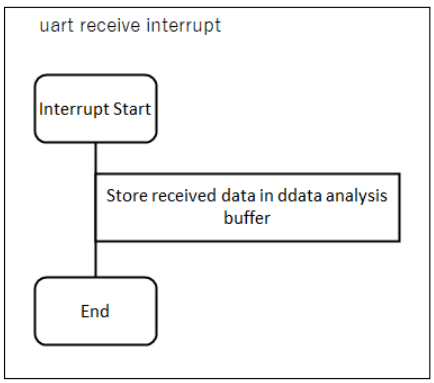
The flow of PMBus Master Application section is shown in section 5.1.8.1, the flow of PMBus Master API section is shown in Section 5.1.8.2, and the flow of PMBus Master drivers section is shown in Section 5.1.8.3. In addition, Please refer to the project code for PMBus Master Middleware section.

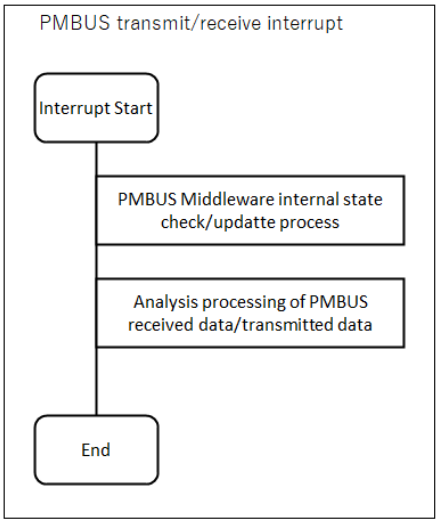
5.1.8.1 PMBus Master Application Flowchart

PMBus Master Application part communicates with PC and calls API that control PMBus Master Middleware part. The flowchart of PMBus Master Application part is shown below. Refer to the Project Code for the process flow of the intermediate functions in PMBus Master Application section. The red-framed areas in the PAD diagram related to RSCI11 indicate the changes made to the code generated by the Smart Configurator.

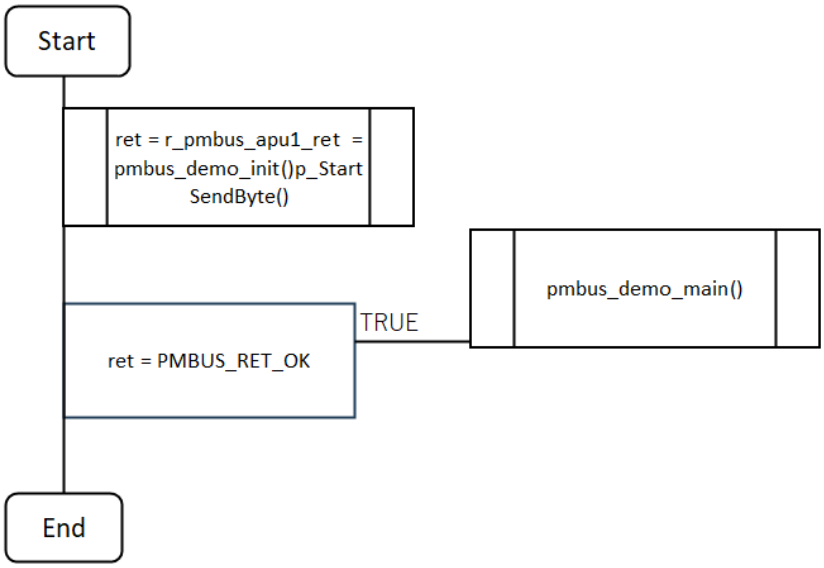
- PMBus master overall outline flow



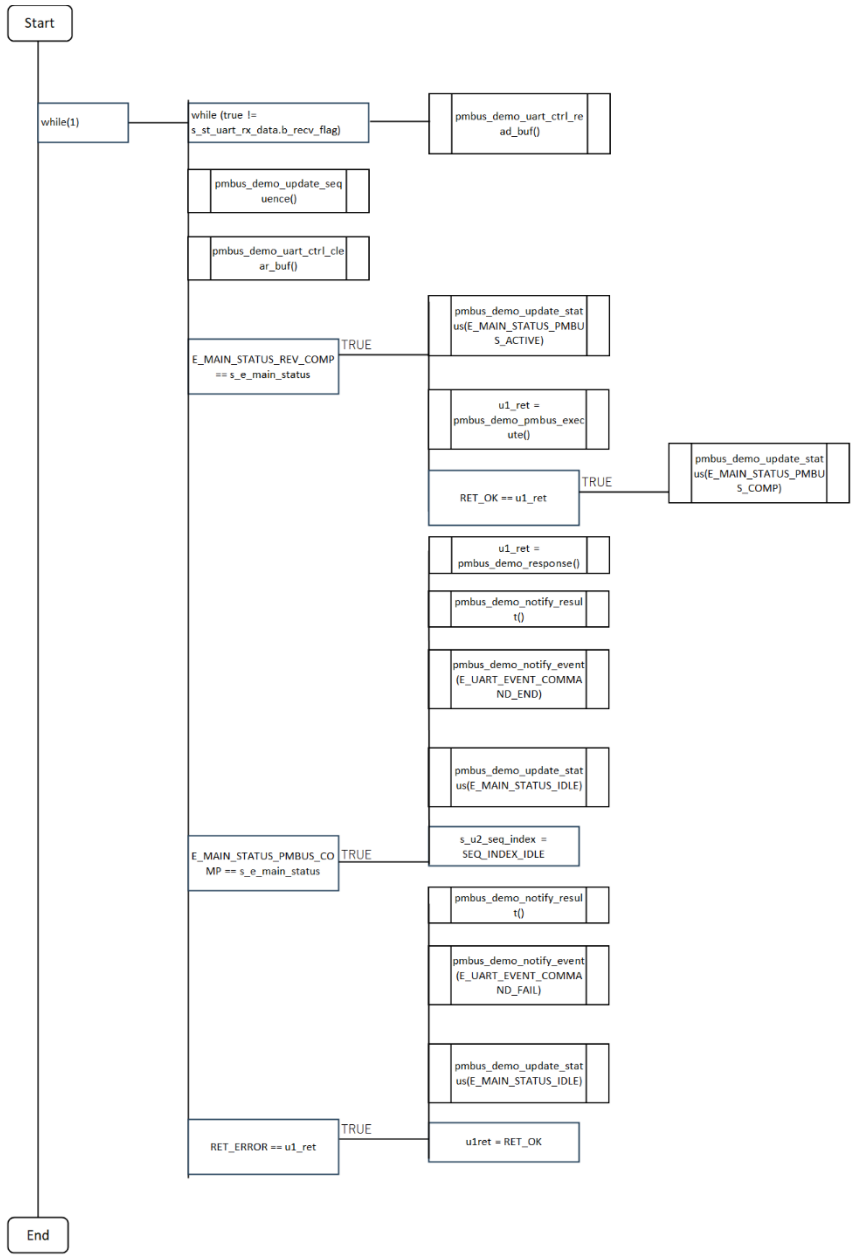




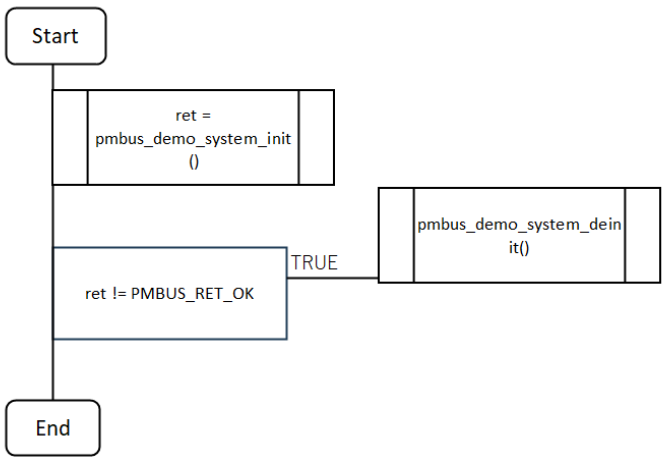
- PMBus master function detailed flow
  - main



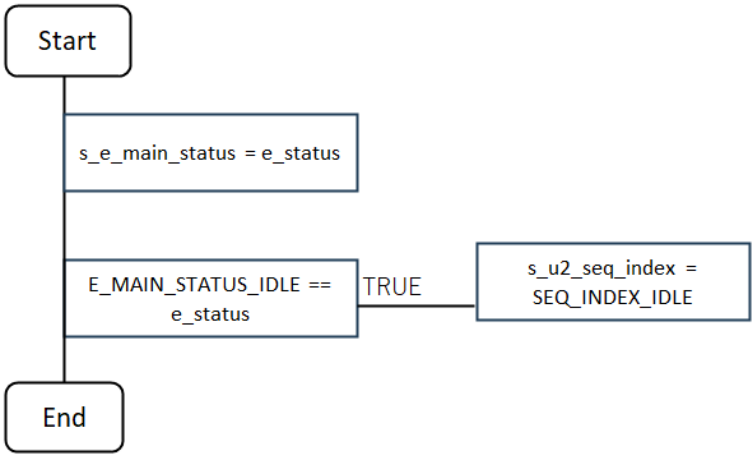
- pmbus_demo_main



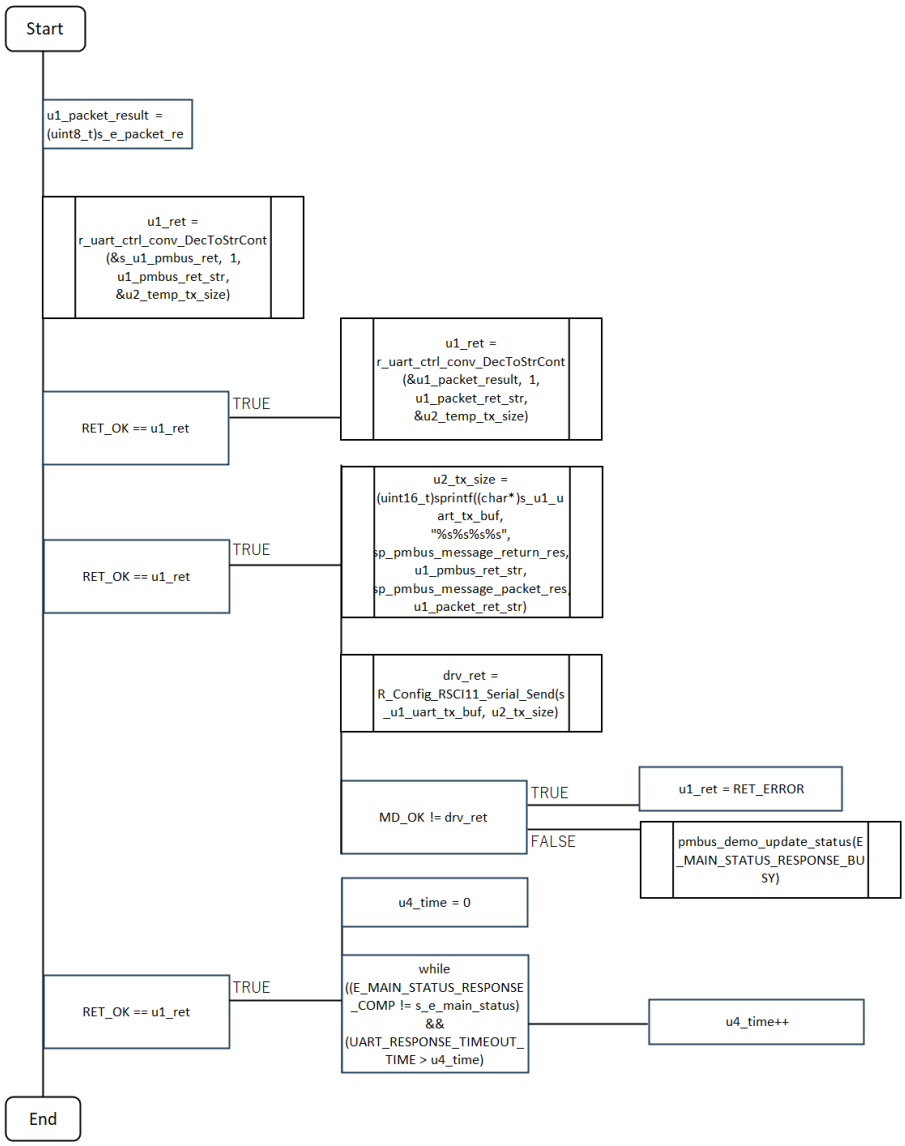
- pmbus_demo_init



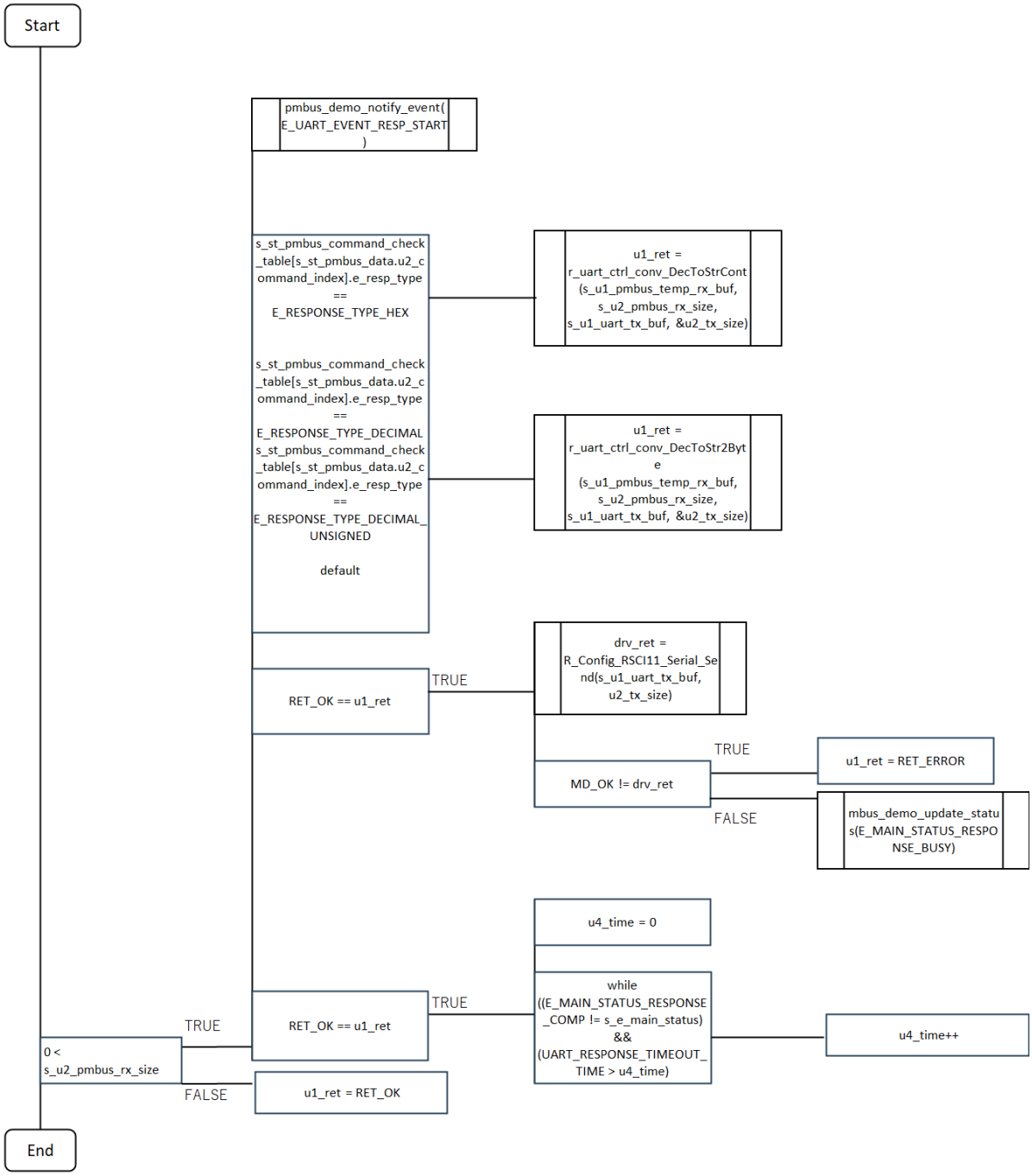
- pmbus_demo_update_status



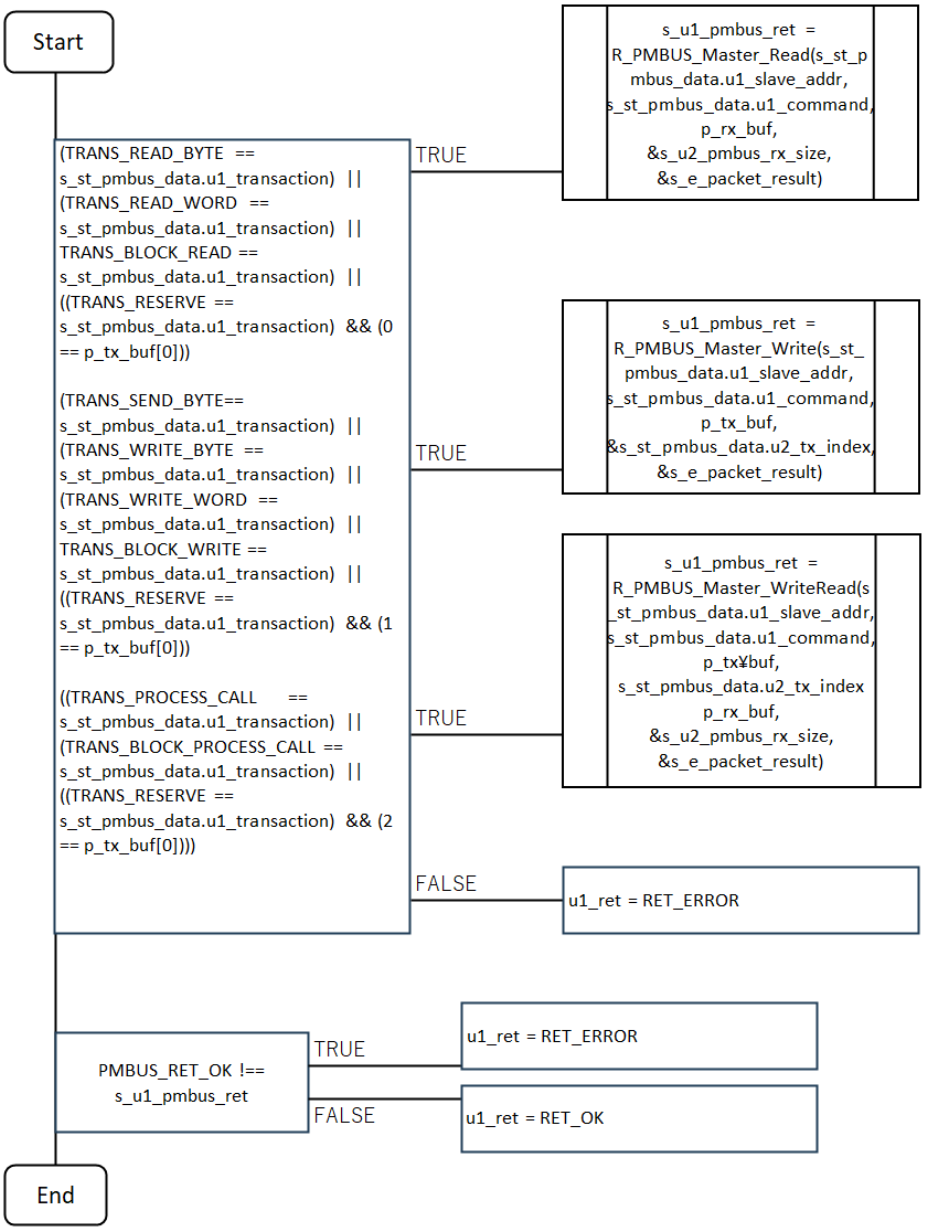
- pmbus_Demo_notify_result



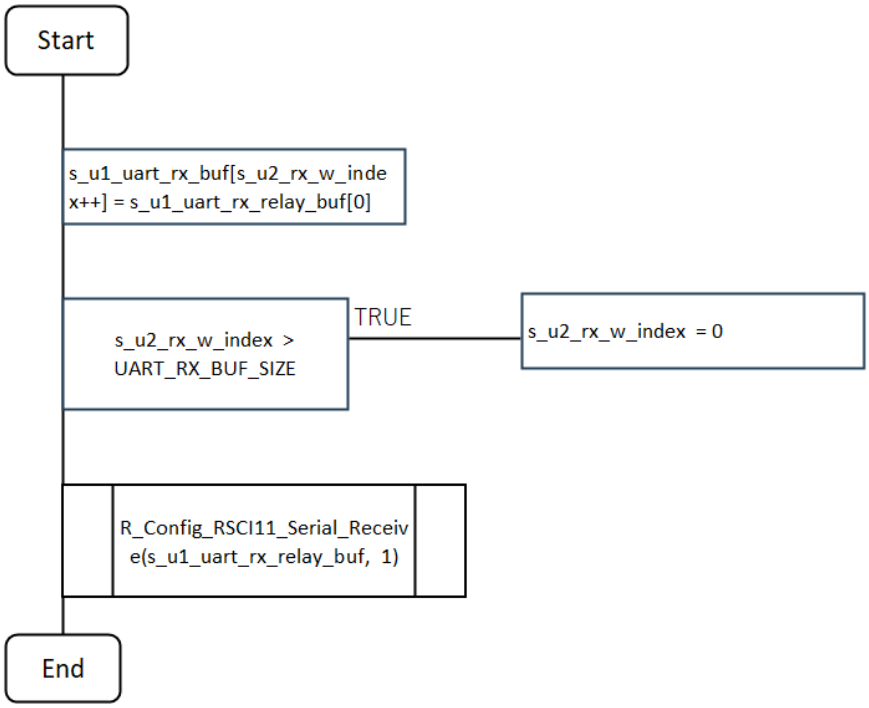
- pmbus_demo_response



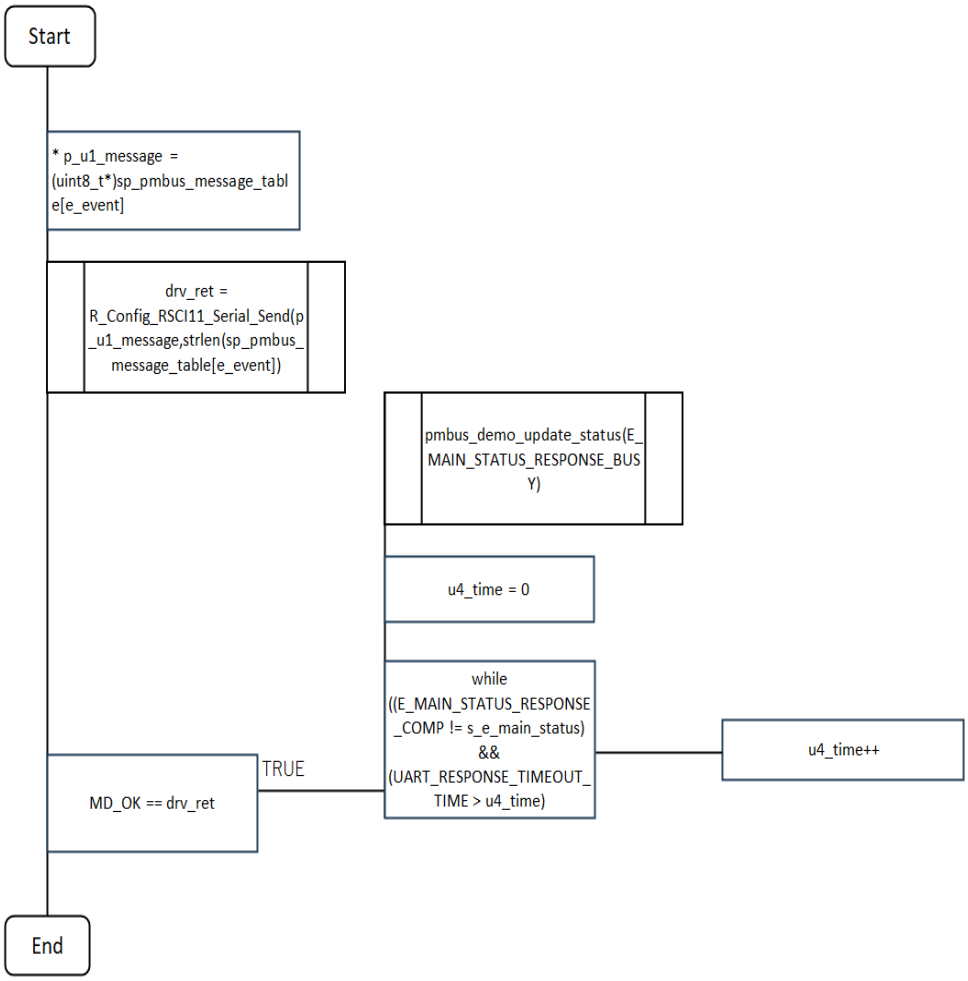
- pmbus_demo_pmbus_execute



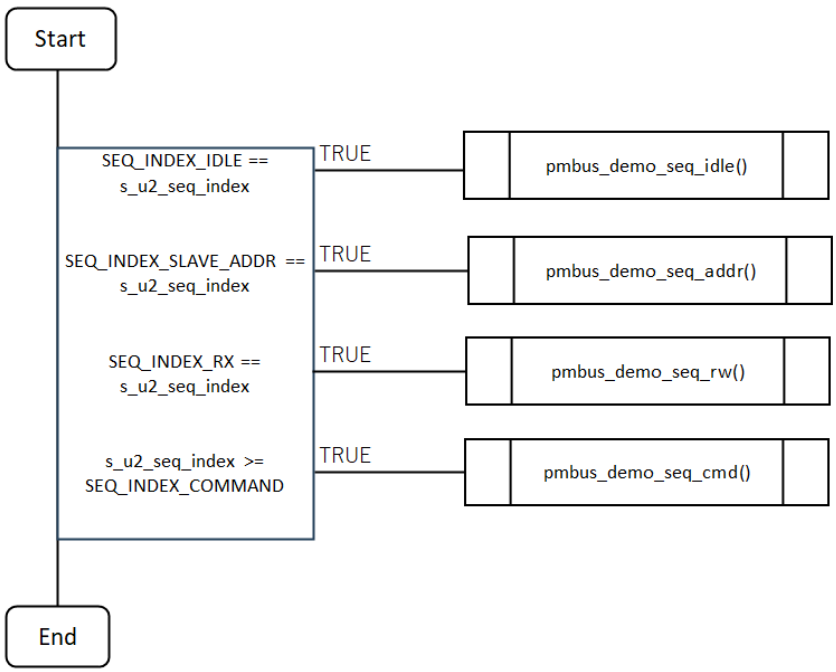
- pmbus_demo_uart_rev_callback



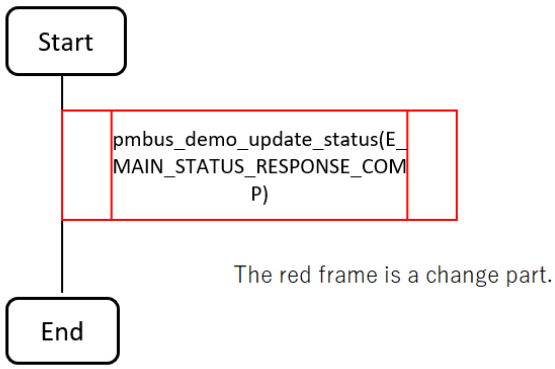
- pmbus_demo_notify_event



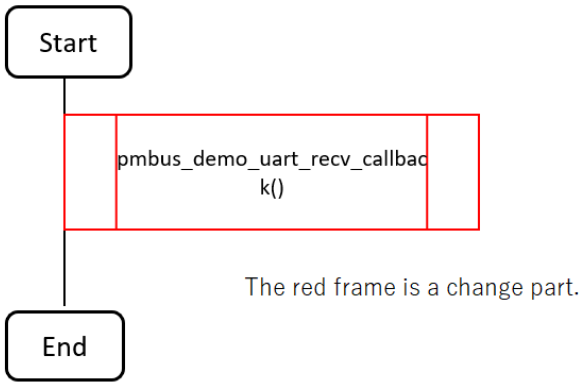
- pmbus_demo_update_sequence



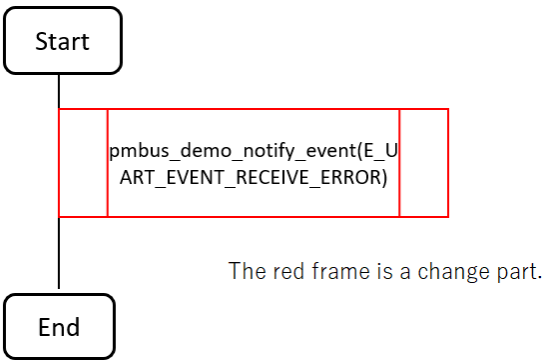
- r_Config_RSCI11_callback_transmitend



- r_Config_RSCI11_callback_receiveend



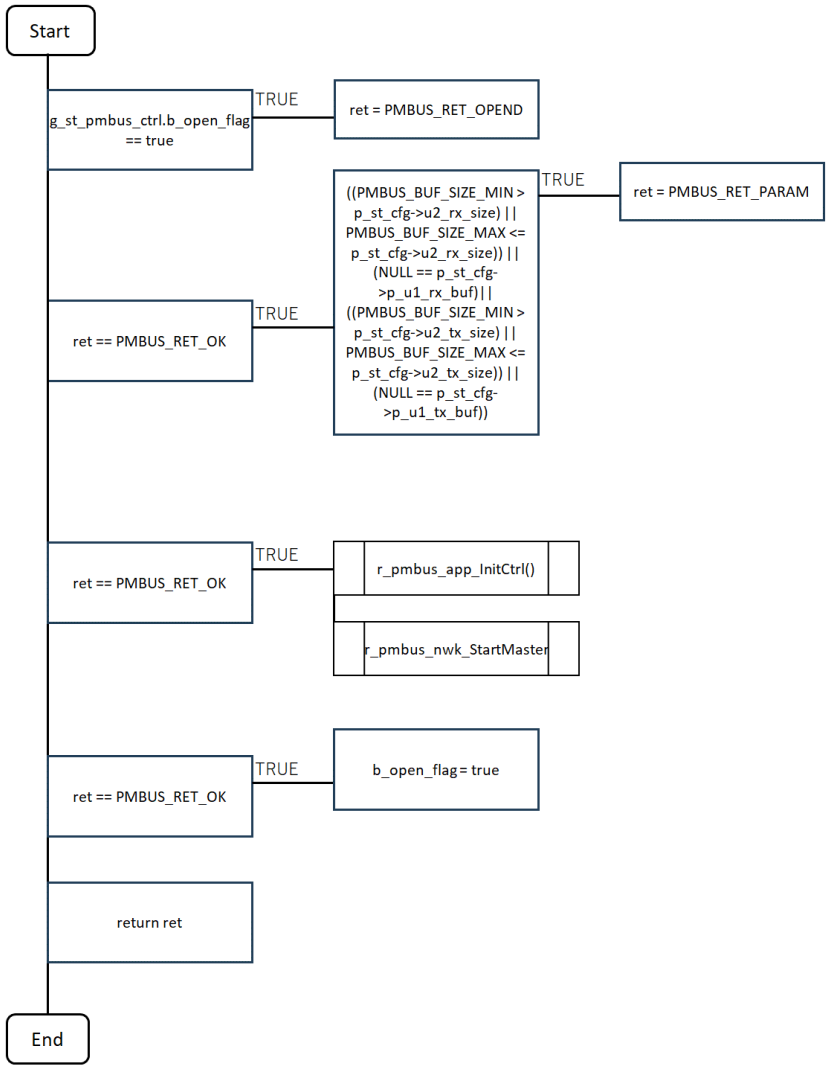
- r_Config_RSCI11_callback_receiveerror



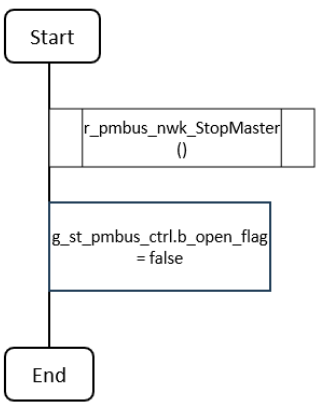
5.1.8.2 PMBus Master API flowchart

PMBus Master API part controls PMBus Master Middleware part. The flowchart for PMBus Master API part is shown below.

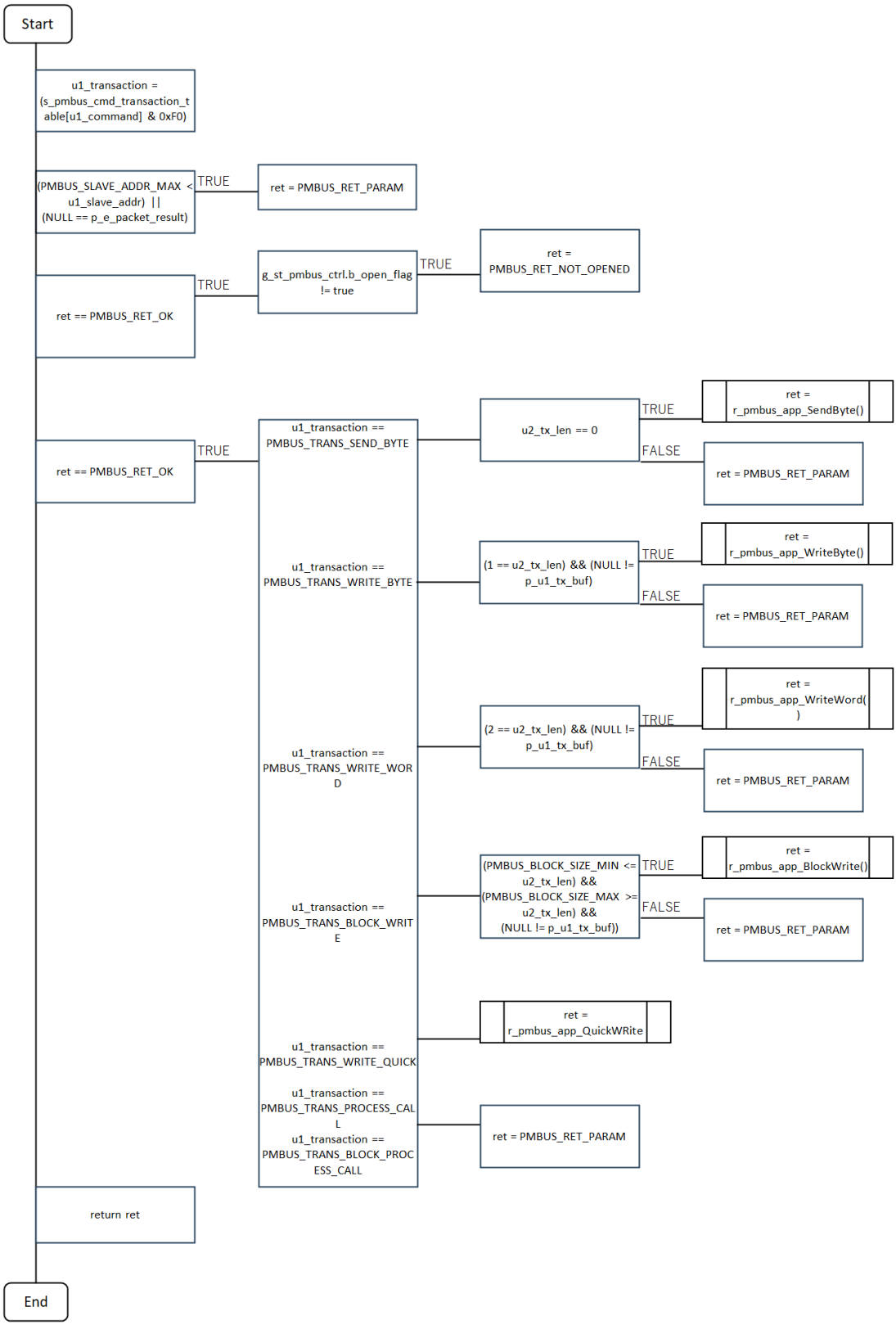
- R_PMBUS_Master_Open



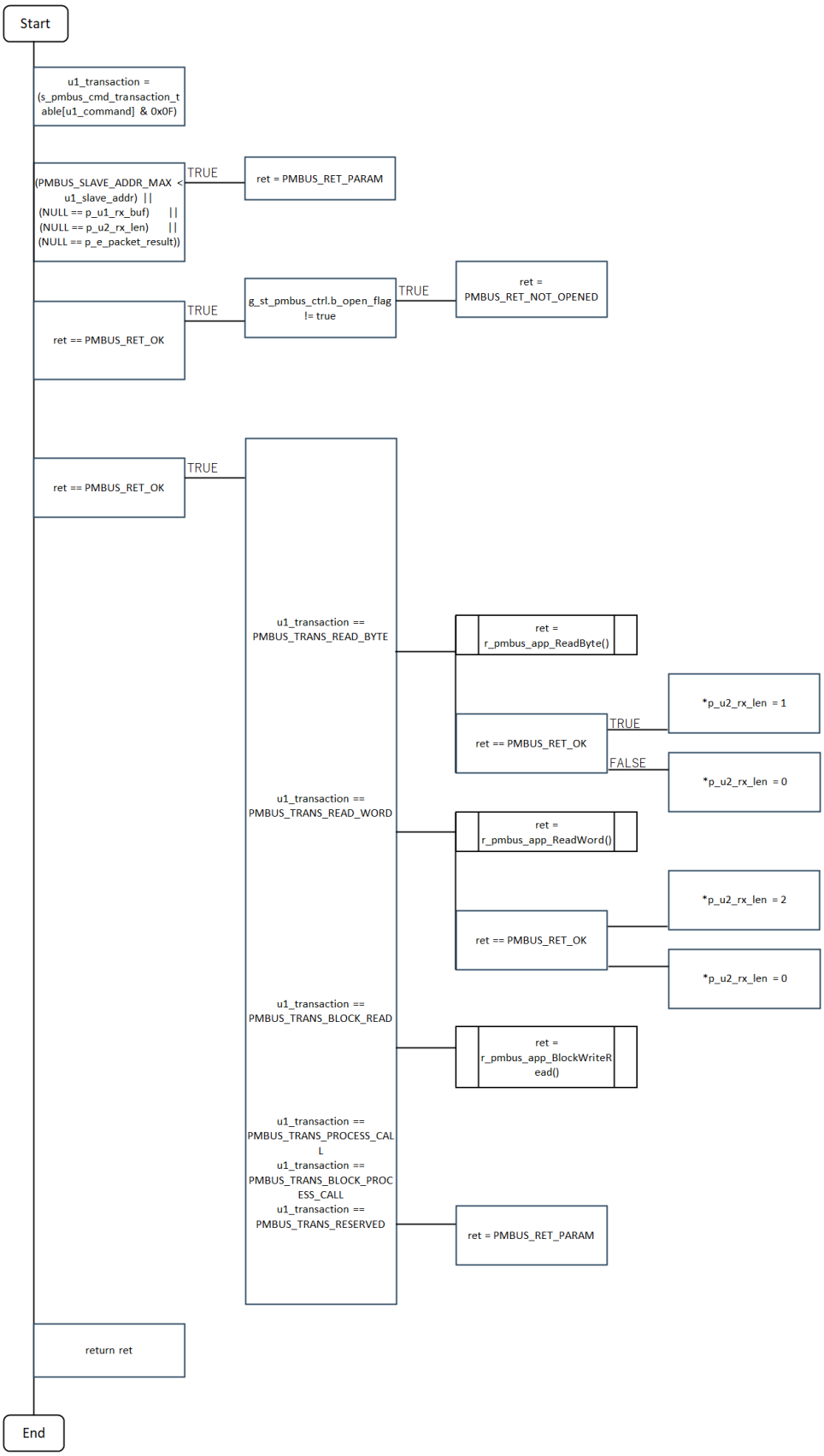
- R_PMBUS_Master_Close



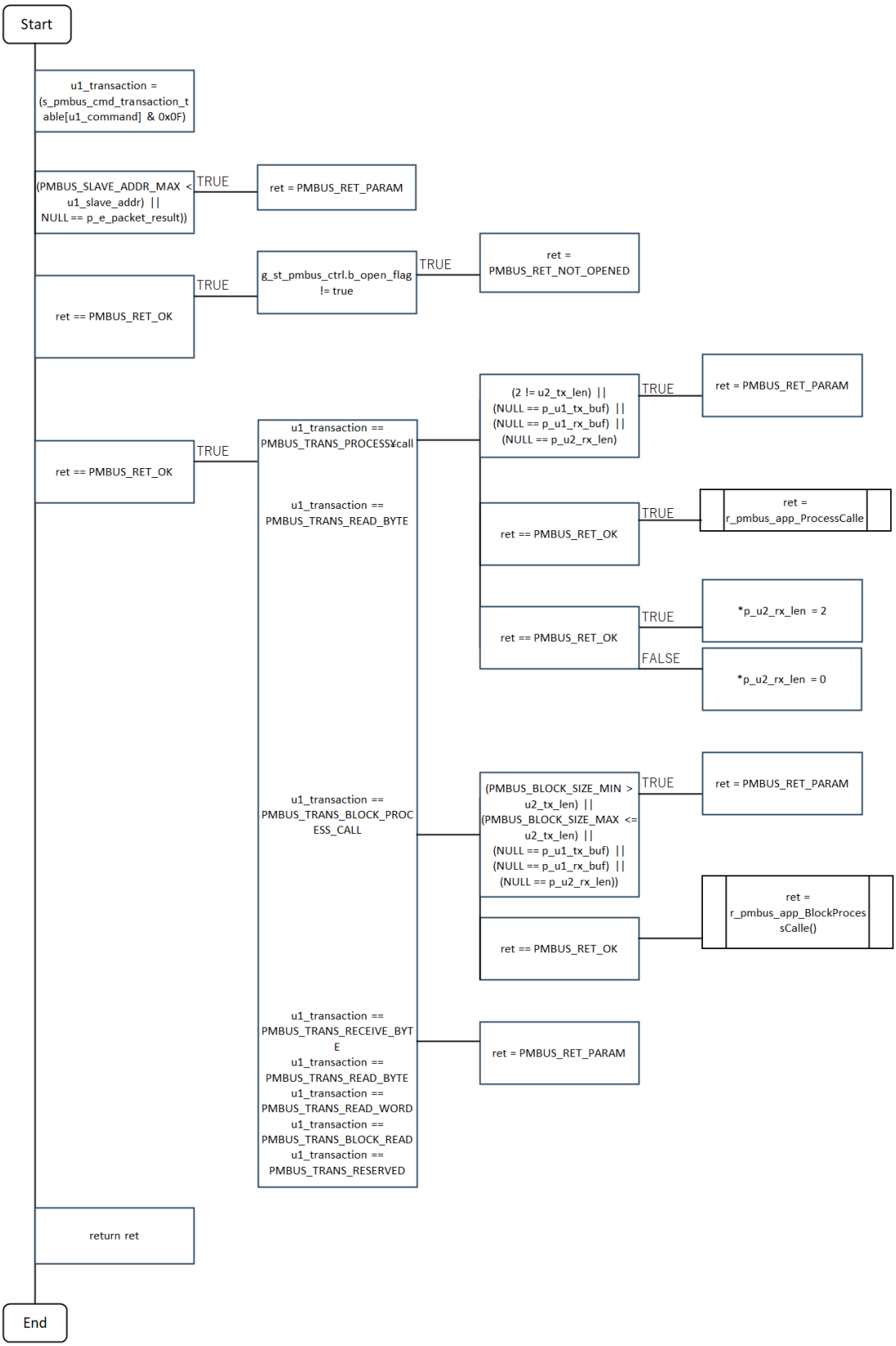
■ R_PMBUS_Master_Write



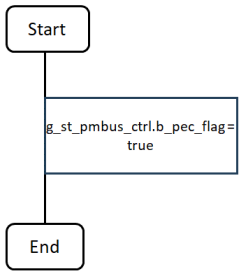
- R_PMBUS_Master_Read



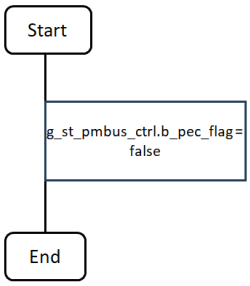
- R_PMBUS_Master_WriteRead



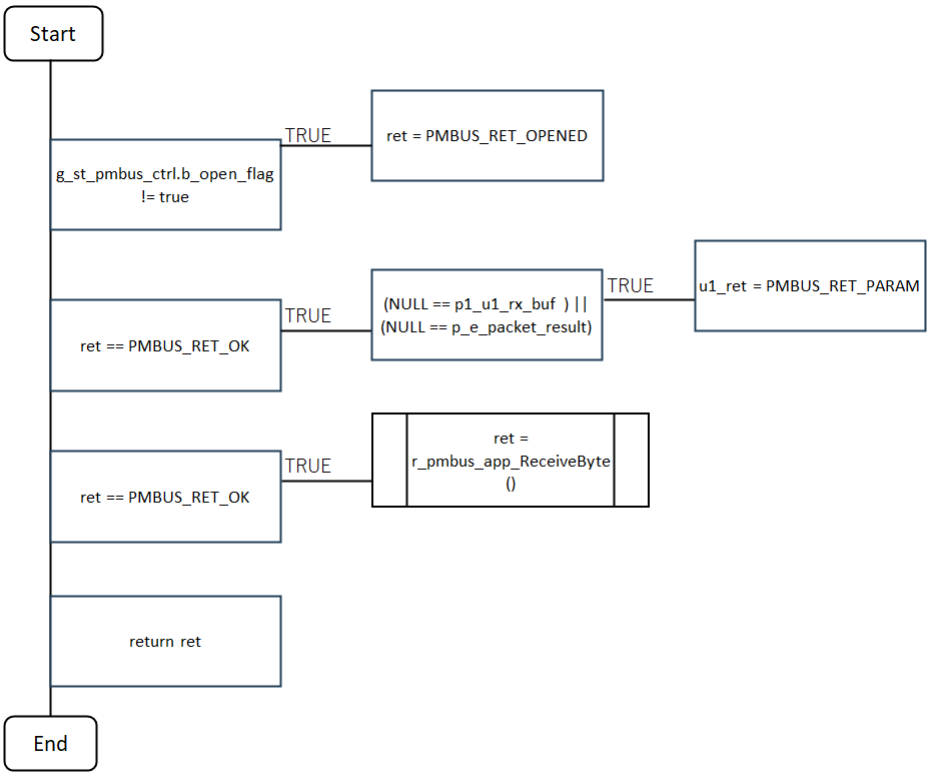
- R_PMBUS_Master_EnablePEC



- R_PMBUS_Master_DisablePEC



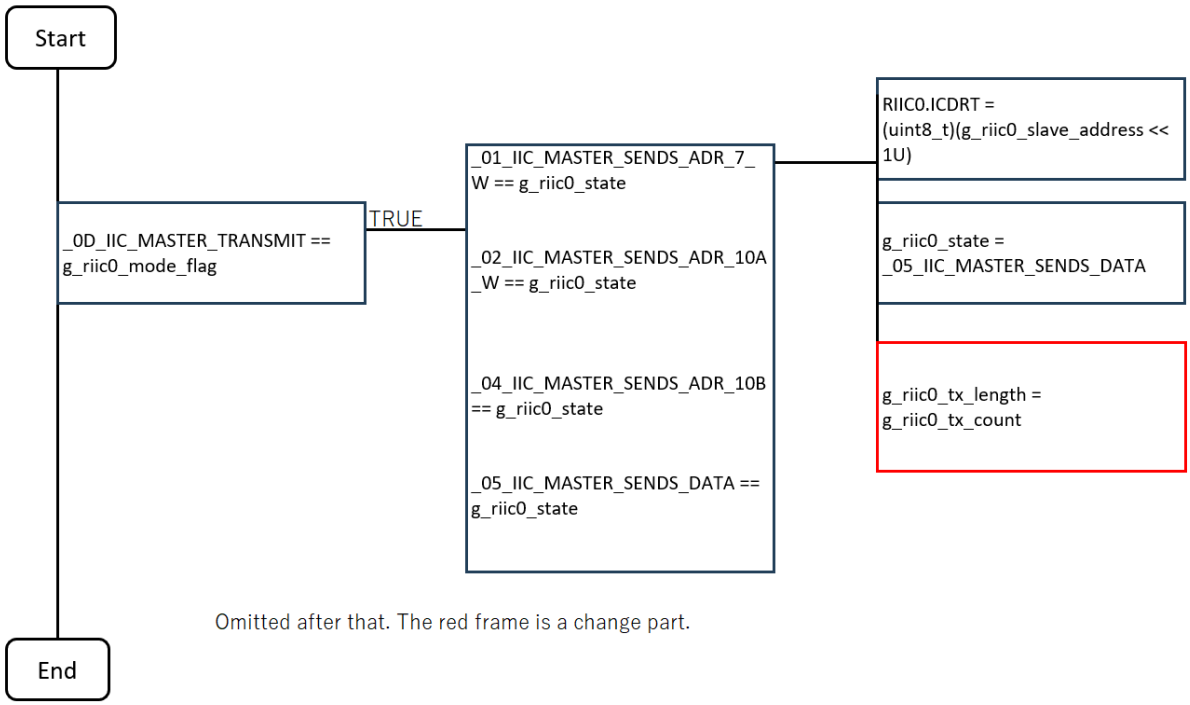
- R_PMBUS_Master_ReceiveARA



5.1.8.3 PMBus Master Drivers Flowchart

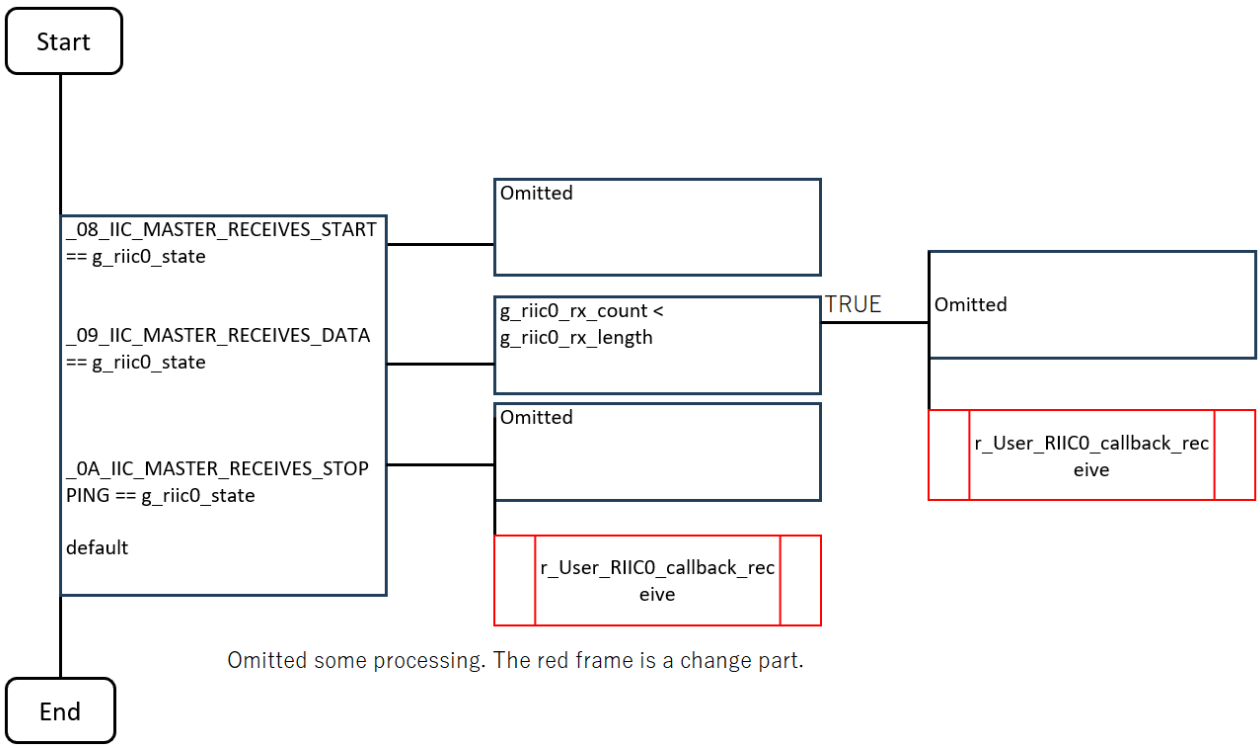
In RX26T, the drivers are partially changed according to PMBus Master process from the code generated by the smart configurator. Refer to Customizing 5.1.4 PMBus Master Driver section for details. The deficit in each pad diagram is the correction part. The areas framed in red in each PAD diagram are the areas to be corrected.

- r_Config_RIIC0_transmit_interrupt



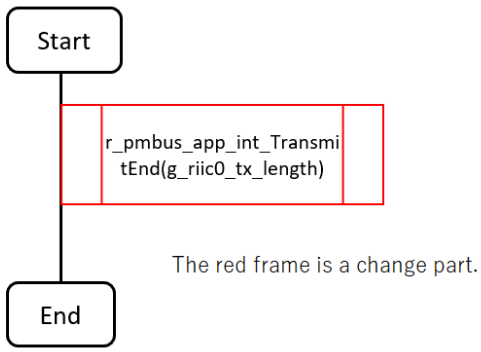
Omitted after that. The red frame is a change part.

- r_Config_RIIC0_receive_interrupt

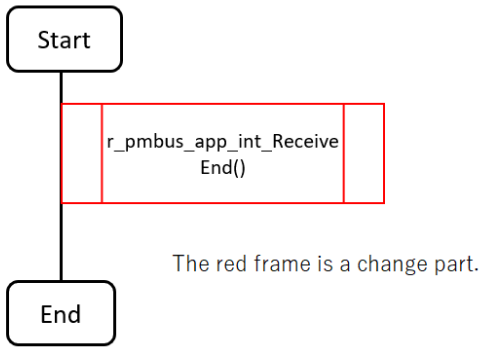


Omitted some processing. The red frame is a change part.

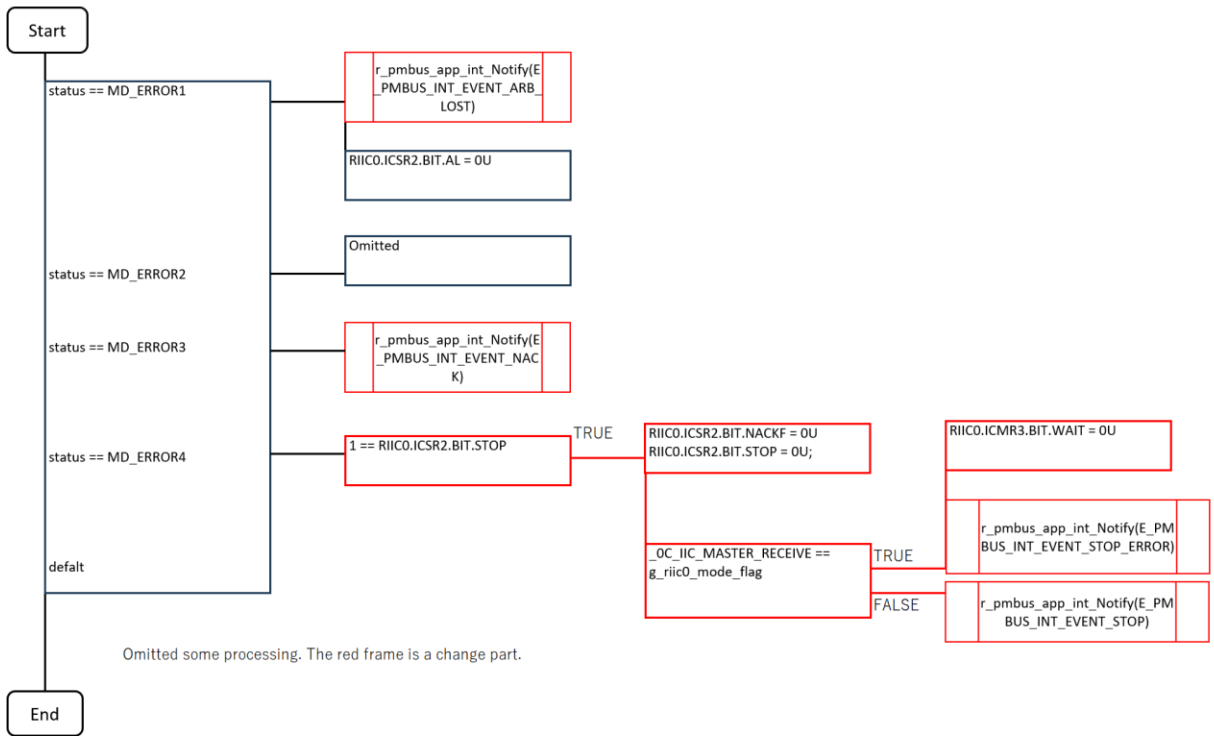
- r_Config_RIIC0_callback_transmitend



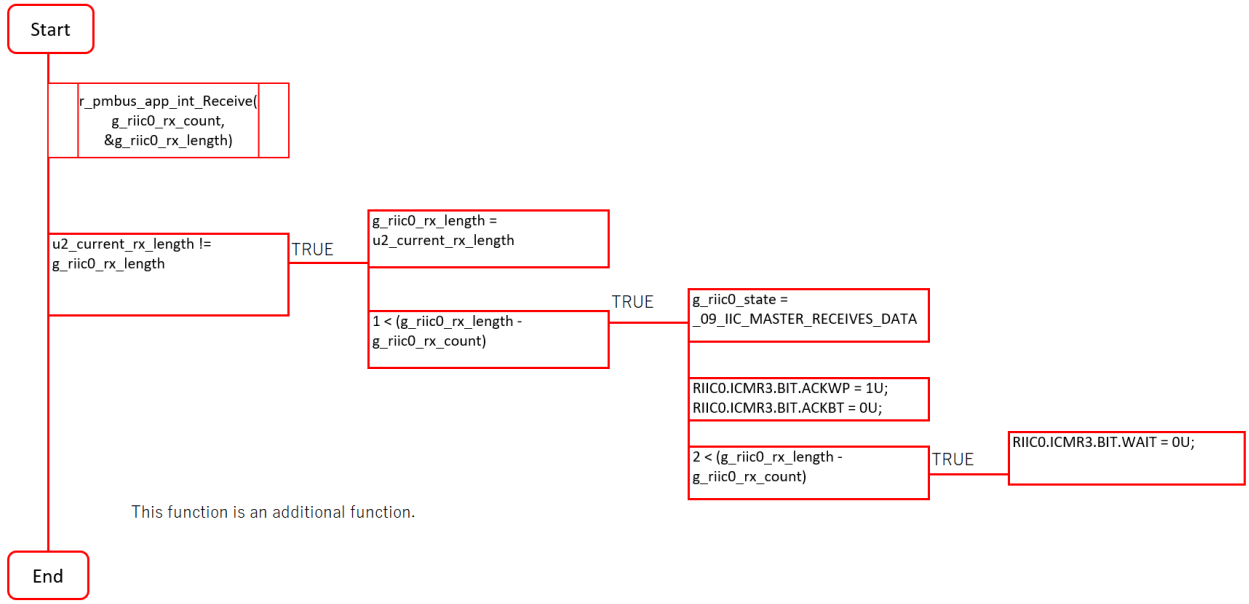
- r_Config_RIIC0_callback_receiveend



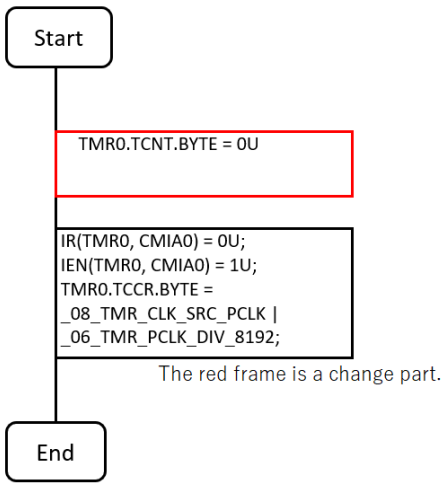
- r_Config_RIIC0_callback_error



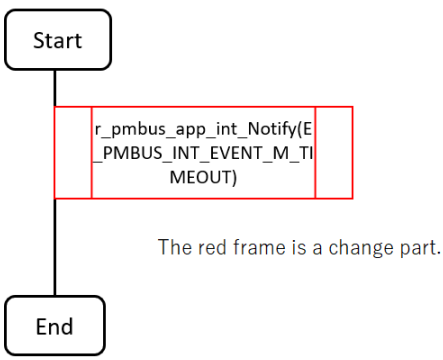
- r_User_RIIC0_callback_receive



- R_Config_TMR0_Start



- r_Config_TMR0_cmia0_interrupt



5.2 PMBus Salve softwares

PMBus Slave software is classified into the user application part, middleware part, and driver part as shown in Figure 21 PMBus Slave Software Module Configuration (RX26T Group) and Figure 22 PMBus Slave Software Module Configuration (RA6T3 Group). The driver for the RX26T group uses software generated by the Smart Configurator, and for the RA6T3 group uses software generated by FSP, with some modifications made to Execute PMBus Slave operations. Table 24 shows RX26T folder/file configurations for each software, and Table 25 shows RA6T3 groups.

Table 24 PMBus Slave RX26T Folder/File Configuration

Folder name	File name	Outline
pmbus_app\	r_app_main.c	The main program of PMBus demonstration system. User Applications PMBus user application process is added to main file of the motor sample.
pmbus_app\	r_app_board_ui.c	The program for controlling the motor to monitor UI of the motor sample board. Commented motor revolution starting control is used when SW1 is ON for the motor sample.
pmbus_app\	r_app_main.h	The header file to use for the main program of PMBus demonstration system. PMBus user application process is added to main file of the motor sample.
pmbus_slave\	r_pmbus_app_slave.c	The application-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_app_slave.h	The header file to use for application-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_nwk_slave.c	The network-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_nwk_slave.h	The header file to use for the network-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_wrapper_slave.c	Use in the wrapper function which absorbs the difference between RX26T and RA6T3 driver API in the driver-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_wrapper_slave.h	The header file to use for wrapper function in the driver-layer of PMBus Middleware.
src\smc_gen\Config_RIIC0\	Config_RIIC0.c	The driver-layer of PMBus Middleware. Generate by the smart configurator.
src\smc_gen\Config_RIIC0\	Config_RIIC0.h	The header file to use for the driver-layer of PMBus Middleware. Generate by the smart configurator.
src\smc_gen\Config_RIIC0\	Config_RIIC0_user.c	The driver-layer of PMBus Middleware. Generate by the smart configurator.
src\smc_gen\Config_TMR0\	Config_TMR0.c	The driver-layer of PMBus Middleware. Generate by the smart configurator.
src\smc_gen\Config_TMR0\	Config_TMR0.h	The header file to use for the driver-layer of PMBus Middleware. Generate by the smart configurator.
src\smc_gen\Config_TMR0\	Config_TMR0_user.c	The driver-layer of PMBus Middleware. Generate by the smart configurator.
src\smc_gen\Config_CMT0\	-	The driver-layer used in motor sample. Generate by the smart configurator.
src\smc_gen\Config_IWDT\	-	The driver-layer used in motor sample. Generate by the smart configurator.
src\smc_gen\Config_MOTOR\	-	The driver-layer used in motor sample. Generate by the smart configurator.

src\smc_gen\Config_POE\	-	The driver-layer used in motor sample. Generate by the smart configurator.
src\smc_gen\Config_PORT\	-	The driver-layer used in motor sample. Generate by the smart configurator.
src\smc_gen\Config_S12AD2\	-	The driver-layer used in motor sample. Generate by the smart configurator.
motor_module\	-	Middleware parts of motor sample.
app\	-	Main application parts of motor sample. The demonstration system excludes files in main folder and r_app_board_ui.c in the board_ui folder from being built.
app\cfg\	r_app_control_cfg.h	File that defines the configuration information of motor sample. Change "APP_CFG_USE_UI" to "MAIN_UI_BOARD" to control the motor by Board UI.

Table 25 PMBus Slave RA6T3 Folder/File Configuration

Folder name	File name	Outline
pmbus_app\	r_app_pmbus_main.c	The main program of PMBus demonstration system. User Applications PMBus user application process is added to main file of motor sample.
pmbus_app\	r_app_pmbus_main.h	The header file to use for the main program of PMBus demonstration system. PMBus user application process is added to main file of motor sample.
pmbus_app\	r_app_control_parameter.h	The header file to use for the main program of PMBus demonstration system. Renamed only, the file r_mtr_control_parameter.h of motor sample.
pmbus_app\	r_app_motor_parameter.h	The header file to use for the main program of PMBus demonstration system. Renamed only, the file r_mtr_moter_parameter.h of motor sample.
pmbus_slave\	r_pmbus_app_slave.c	The application-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_app_slave.h	The header file to use for the application-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_nwk_slave.c	The network-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_nwk_slave.h	The header file to use for the network-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_wrapper_slave.c	Use in the wrapper function which absorbs the difference between RX26T and RA6T3 driver API in the driver-layer of PMBus Middleware.
pmbus_slave\	r_pmbus_wrapper_slave.h	The header file for wrapper function in the driver-layer of PMBus Middleware.
pmbus_slave\	r_smbus_slave.c	The SMBus driver-layer of PMBus Middleware. Corresponds to the r_iic_b_slave.c of FSP.
pmbus_slave\	r_smbus_slave.h	The header file to use for the SMBus driver-layer of PMBus Middleware. Corresponds to the r_iic_b_slave.h of FSP.

pmbus_slave\	r_smbus_slave_api.h	The header file to use for SMBus driver-layer of PMBus Middleware. Corresponds to the r_i2c_slave_api.h of FSP.
pmbus_slave\	r_smbus_slave_cfg.h	The header file to use for SMBus driver-layer of PMBus Middleware. Corresponds to the r_iic_b_slave_cfg.h of FSP.
ra\fsp\src\r_gpt\	r_gpt.c	PMBus Middleware generated by FSP and the drivers used by the motor samples.
ra\fsp\inc\api\	r_tiemr_api.h	PMBus Middleware generated by FSP and the header file of the driver layer used in the motor sample.
ra\fsp\inc\instance\	r_gpt.h	PMBus Middleware generated by FSP and the header file of the driver layer used in the motor sample.
ra\	-	Various files including the drivers used in the motor samples generated by FSP.
ra_cfg\	-	The driver-layer of PMBus Middleware. Generate by the smart configurator.
ra_gen\	-	The driver-layer of PMBus Middleware. Generate by the smart configurator.
motor_module\	-	Middleware of the motor sample
src\	-	Main application parts of motor sample. The demonstration system excludes files in src \ application \ main folder from being built.

5.2.1 PMBus Slave operation Sequence

To PMBus for Write operation, Read operation, Write/Read operation, and Alert Response operation according to the command-sequence. the Write operation sequence is shown in Figure 32 and Figure 33, the Read operation sequence and Write/Read operation sequence are shown in Figure 34 and Figure 35, and the Alert Response operation is shown in Figure 36 and Figure 37 For API functions used in each operation, refer to PMBus Slave Function List in Section 5.2.3.

[Sequence diagram arrow legend]

Function Call (Own task) :

Function Call (Other task) :
Function Return :
Asynchronous Notification :

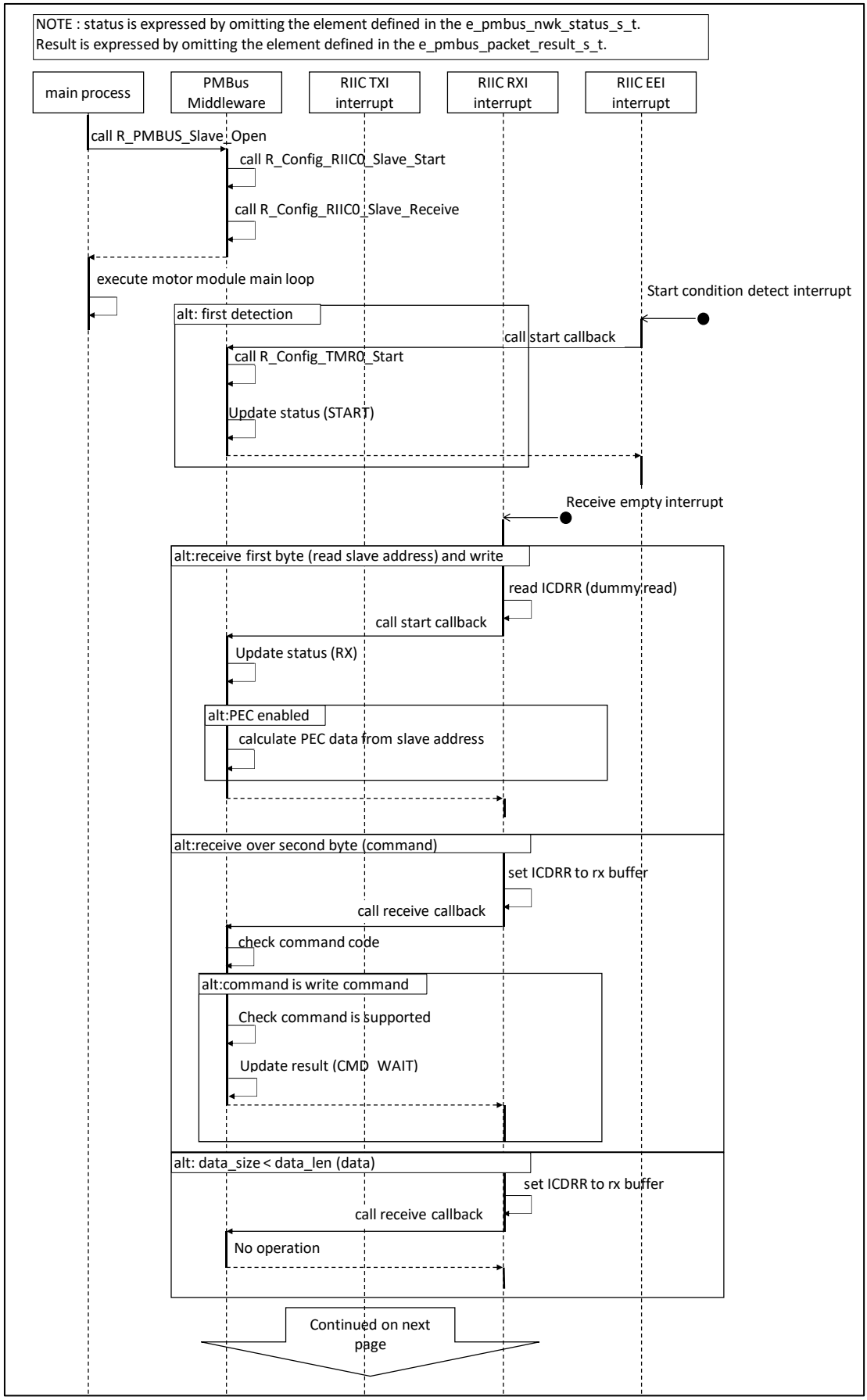


Figure 32 PMBus Slave Write protocol Sequence Diagram (1/2)

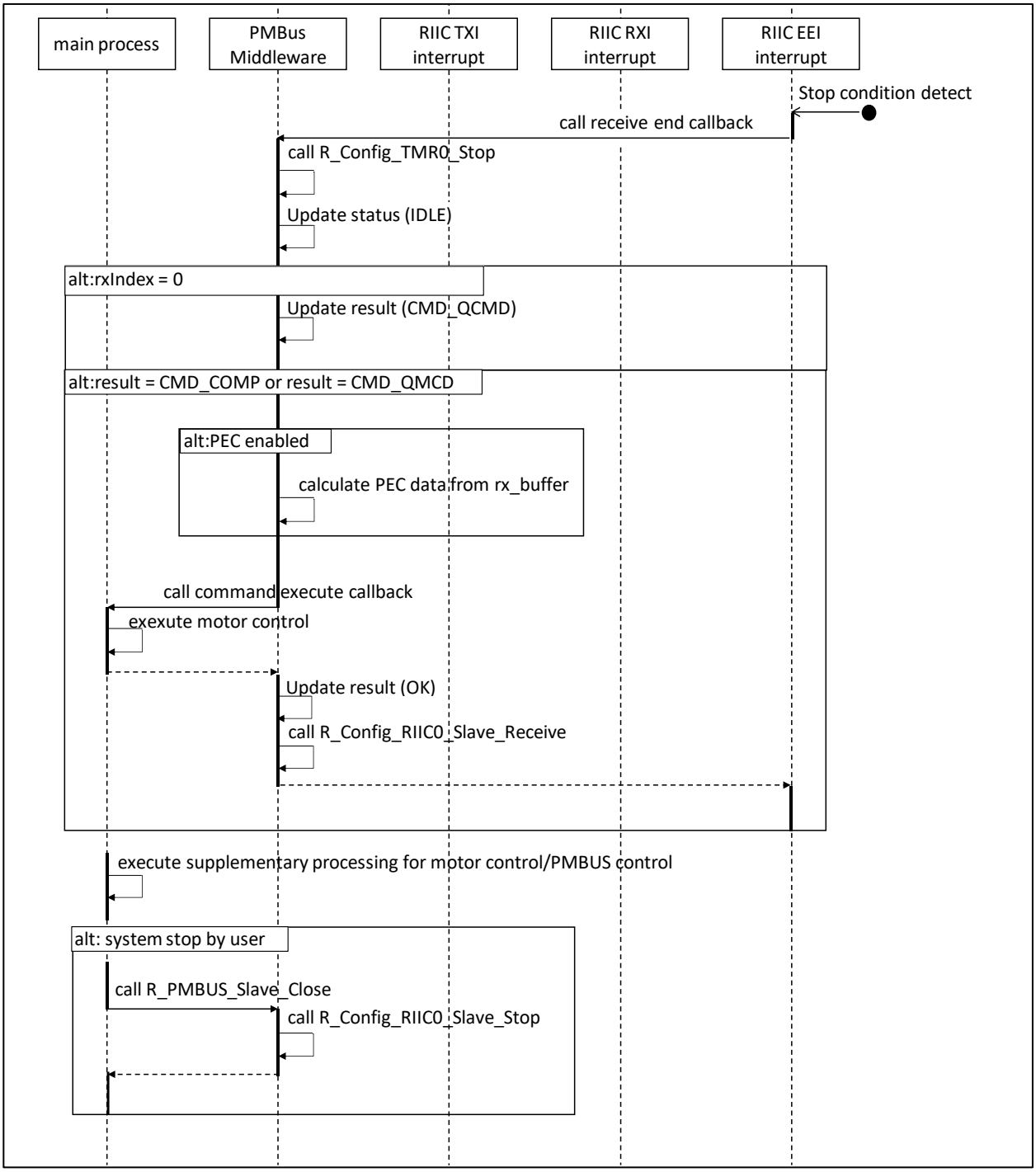


Figure 33 PMBus Slave Write protocol Sequence Diagram (2/2）

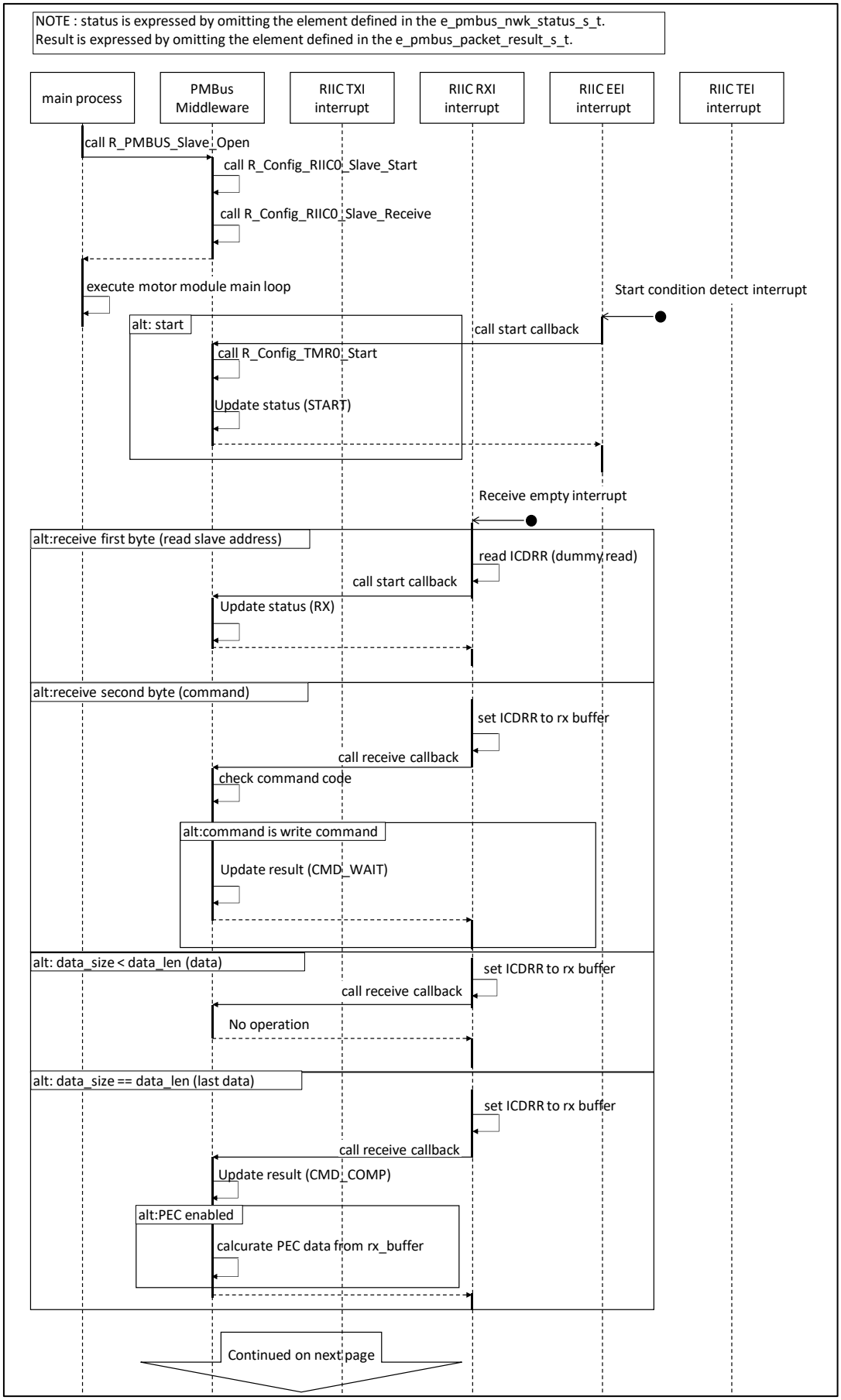


Figure 34 PMBus Slave Read and Write Read protocol Sequence Diagram (1/2)

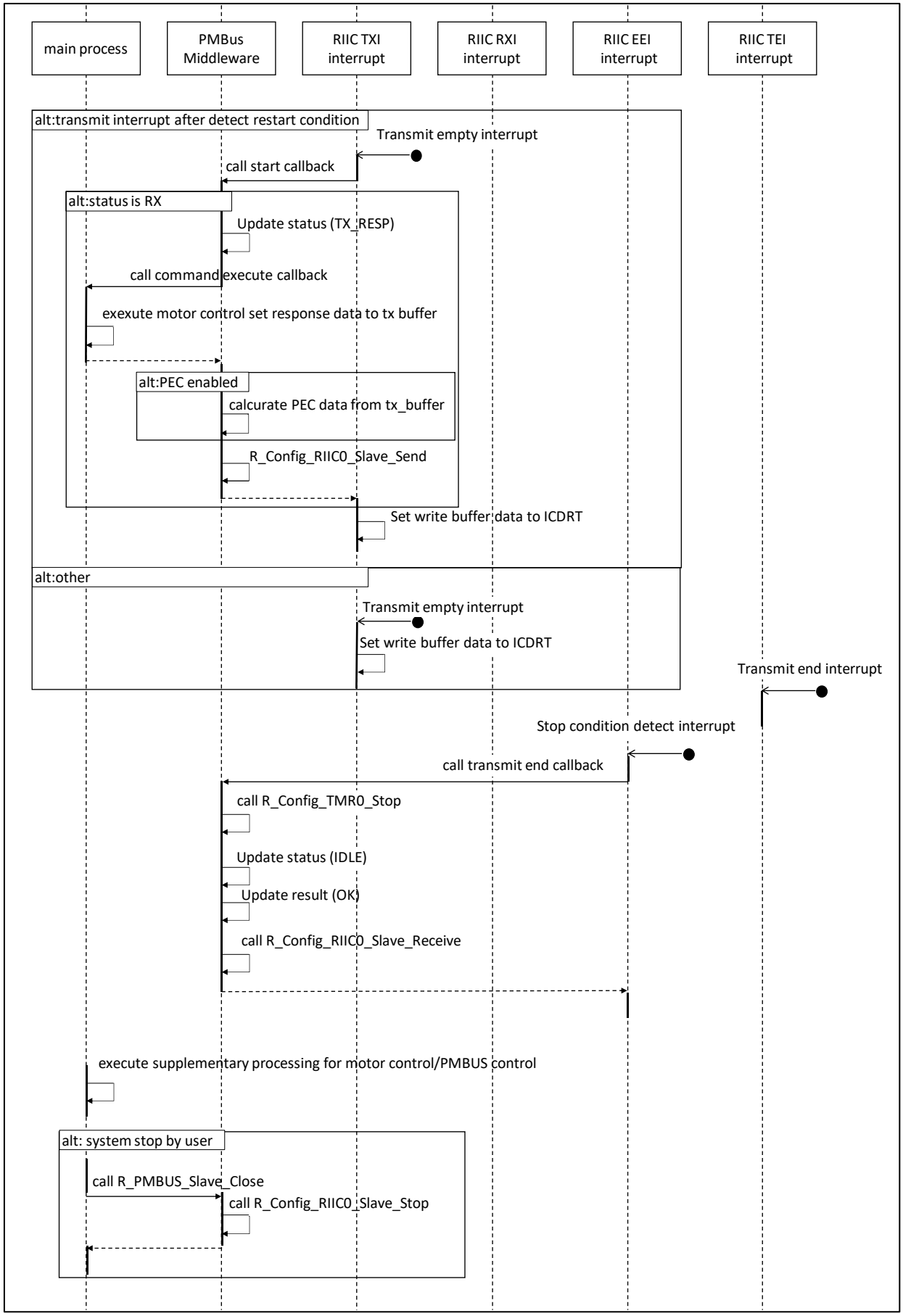


Figure 35 PMBus Slave Read and Write Read protocol Sequence Diagram (2/2)

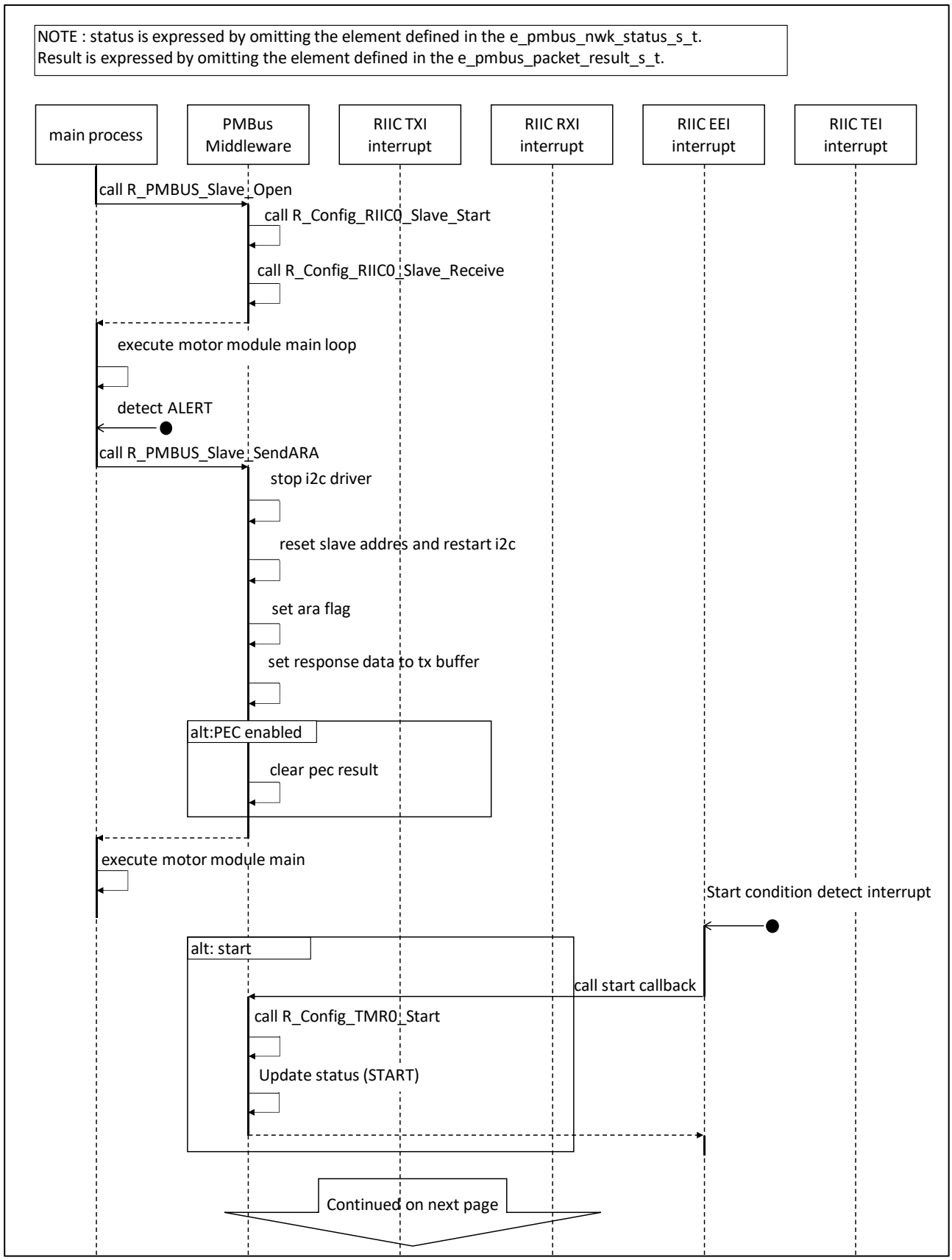


Figure 36 PMBus Slave Alert Response Address protocol Sequence Diagram（1/2）

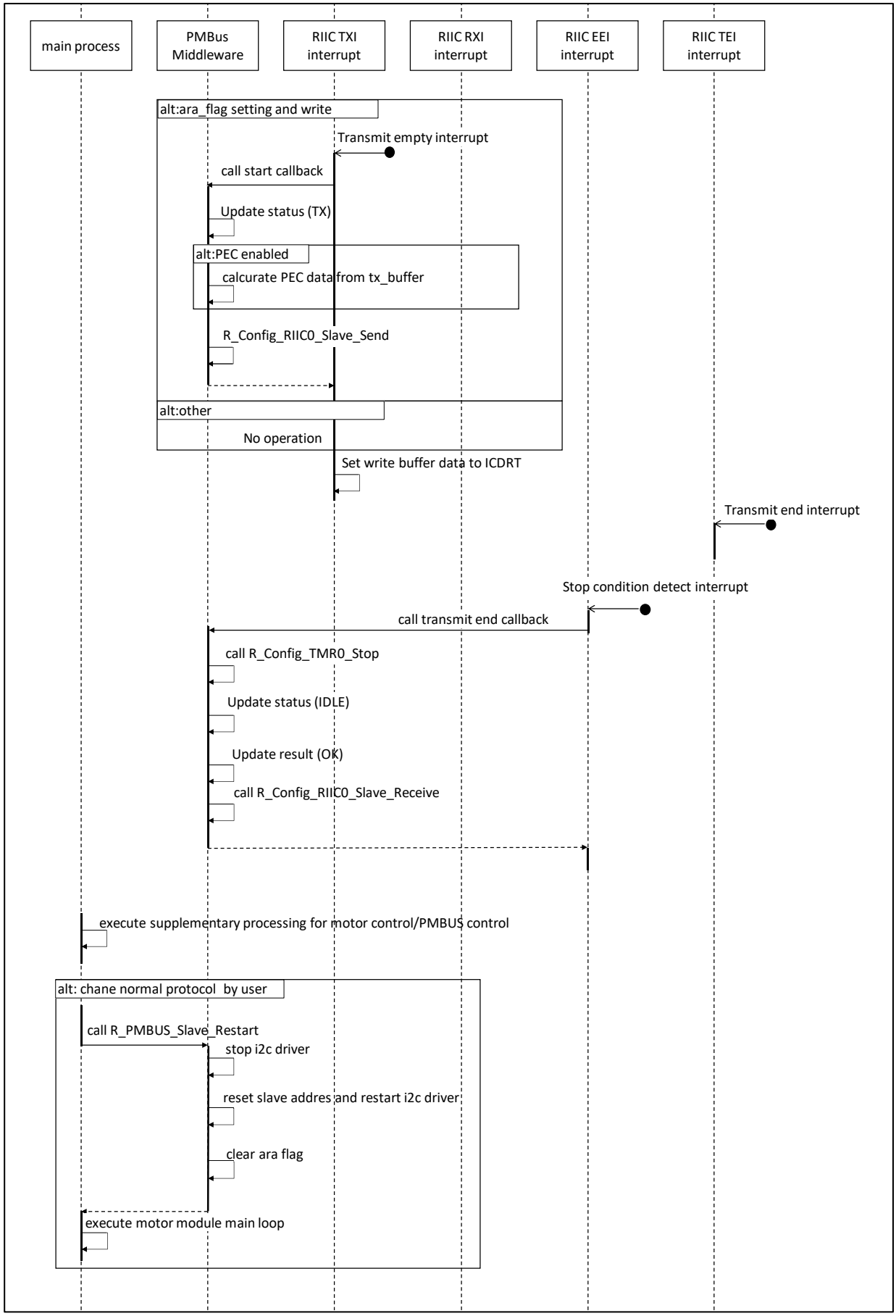


Figure 37 PMBus Slave Alert Response Address protocol Sequence Diagram（2/2）

5.2.2 PMBus Slave status transitions

The status of PMBus Slave middleware is managed by each of Application Layer and Driver Layer. API part manages protocol status transitions, and the driver part manages data transmission/reception numbers. Section 5.2.2.1 shows the status transitions of API part and Section 5.2.2.2 shows the status transitions of the driver part.

5.2.2.1 PMBus Slave Middleware Application Layer status transitions

Middleware Application Layer status transitions of PMBus Slave are waiting in IDLE status in preparation for reception from PMBus Master. When a command is subsequently sent from PMBus Master, the status of transmission, reception, and error handling is managed in accordance with the command code. Application Layer status transitions of the following PMBus Slave are shown in Figure 38, Figure 39, Figure 40, Figure 41, Table 26, Table 27 and Table 28.

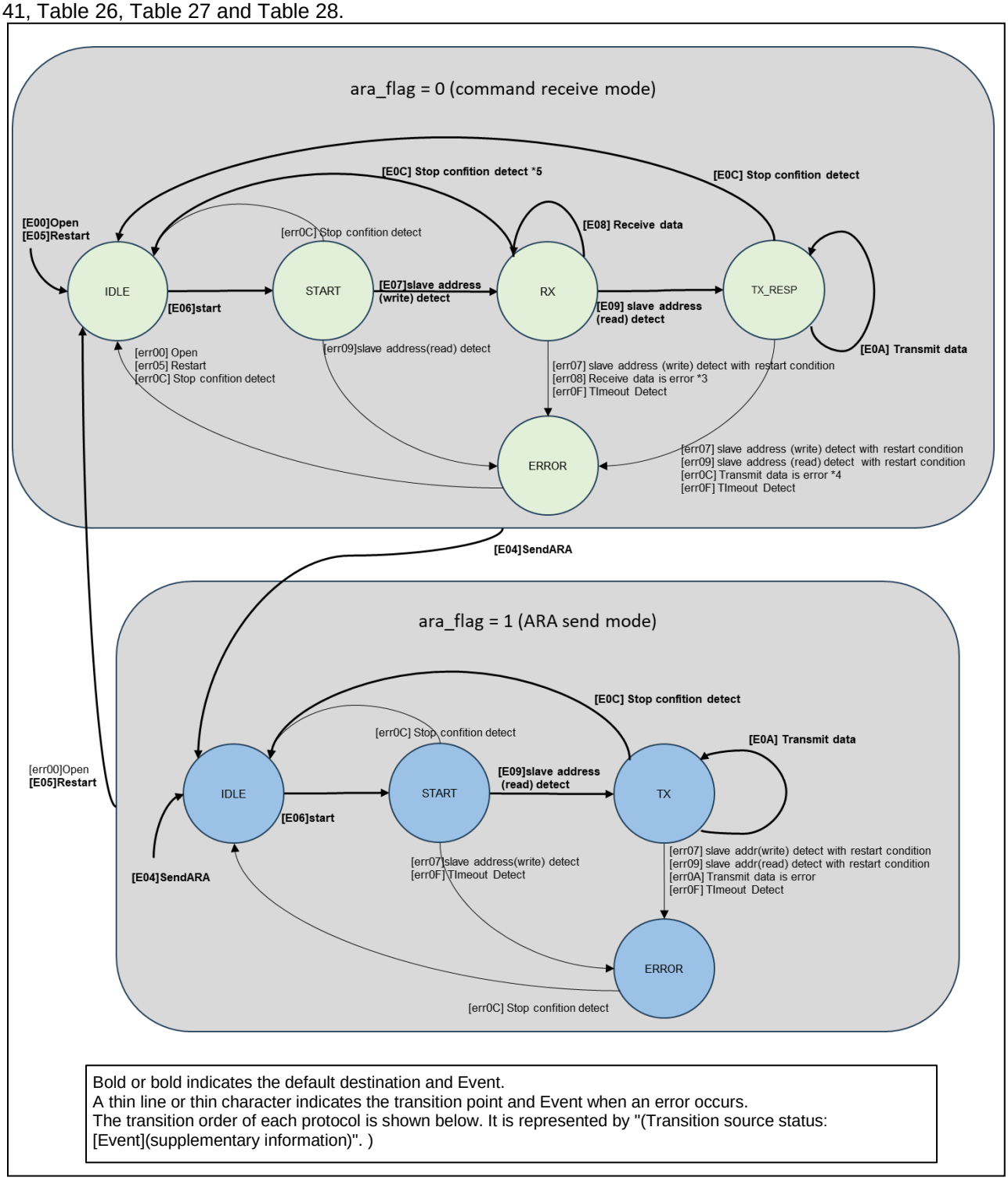


Figure 38 PMBus Slave Middleware Application Layer status transitions diagram

[Event List]
*[Exx] indicates normal Event, and [errxx] indicates Event when an error occurs.

E00/err0_Open ,E01_Close *1 ,E02_EnablePEC *1
E03_DisablePEC *1 ,E04_SendARA ,E05/err05_Restart
E06_Interrupt (Start Condition detect) ,
E07/err07_Interrupt (Receive buffer full when the RW bit of the slave address is set to write) *2,
E08/err08_Interrupt (Receive Buffer Full),
E09/err09_Interrupt (Transmit buffer empty when the RW bit of the slave address is read) *2,
E0A/err0A_Interrupt (Transmit Interrupt) ,E0B_Interrupt (Transmit End Interrupt) *1,
E0C/err0C_Interrupt (Stop Condition detect),
err0D_Interrupt (Arbitlation Lost) *1 ,err0E_Interrupt (NACK detect) *1 ,err0F_Interrupt (Timeout Detect)

*1. A Event in which no status transitions occur.
*2. Whether a Receive Buffer full interrupt or a Transmit Buffer empty interrupt occurs when a slave address is received,
The hardware is determined by RW of the slave address.
*3. Set an error code in PACKET RESULT and enter ERROR in the following cases.
- The number of received data exceeds the receive buffer size. PACKET RESULT=DATA_SIZE
- When the receive buffer is not registered. PACKET RESULT=NOT_READY
- The first data (command) received is unsupported. PACKET RESULT=CMD_NOT_SUPPORT
*4. Set an error code in PACKET RESULT and enter ERROR in the following cases.
- The transmit buffer has not been registered. PACKET RESULT = NOT_READY
*5. Check PACKET RESULT and execute the callback. Refer to the status diagram of PACKET RESULT for the term to execute the callback.

Figure 39 PMBus Slave Middleware Application Layer status transitions diagram Supplement to Figure 38

- PACKET RESULT status transitions

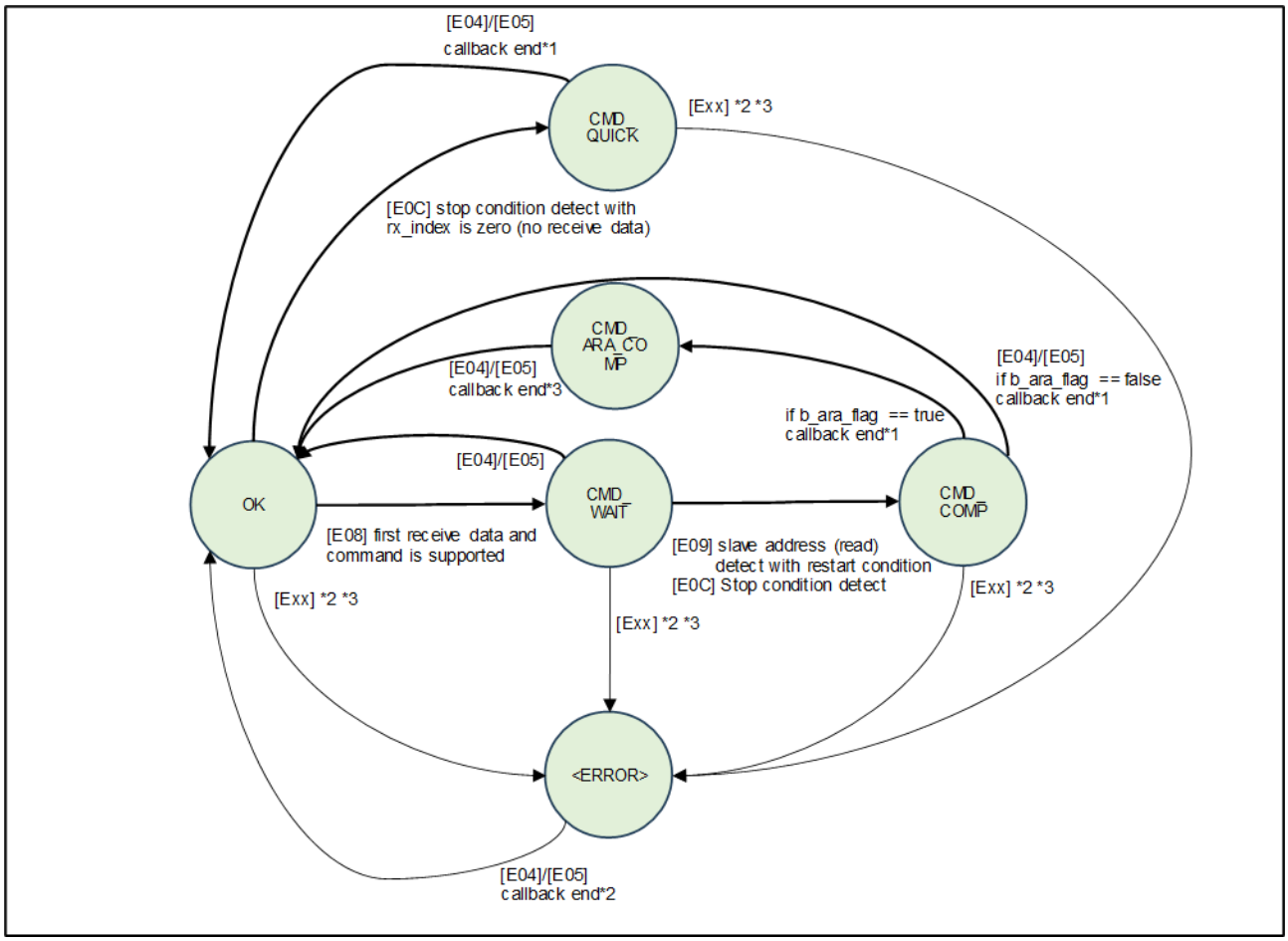


Figure 40 PMBus Slave Middleware Application Layer status transitions diagram for PACKET RESULT

*1. Calls a callback function and executes command processing.
*2. [Exx] stands for one of the following events:。
[E07/err07]
[E08/err08]
[E09/err09]
[E0A/err0A]
[err0F]
*3. Call the callback function and Executes error notification.
<ERROR> means one of the following.
DATA_SIZE_ERROR
PEC_ERROR
TIMEOUT
INTERNAL_ERROR
NOT_READY
CMD_NOT_SUPPORT
*4. Call the callback function and execute ARA reply completion process.

Figure 41 PMBus Slave Middleware Application Layer status transitions diagram for PACKET RESULT Supplement to Figure 40

Table 26 PMBus Slave Application Layer status transitions table (ara_flag = 0)

This table shows the state transition table when ara_flag is 0.
Explanations of the annotations are summarized in Figure 27

This table and.Figure 27 should be interpreted as follows.

- event consists of an abbreviation for the API name and the interrupt cause.
- status consists of an abbreviation for the "e_pmbus_nwk_status_s_t" element name.
- "If (<condition>)" means a conditional transition.
- [→<state>] means a transition to a state.
- "ERROR (<error name>)" means the API return value.
- "PACKET RESULT(<error name>)" means the error information stored in the API argument p_e_packet_result).
- [-] means no state transition.
- Light green indicates state transitions when ara_flag=0 in Figure 38.
- Light blue indicates state transitions when ara_flag=1 in Figure 38.

	IDLE	START	RX	TX_RESP	ERROR *5
E00/err00_Open	→ IDLE*7	→ IDLE*7	→ IDLE*7	→ IDLE*7	→ IDLE*7
E01_Close	-	-	-	-	-
E02_EnablePEC	-	-	-	-	-
E03_DisablePEC	-	-	-	-	-
E04_SendARA	→ ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1
E05/err05_Restart	→ ara_flag = 0	→ IDLE PACKET RESULT (OK) ara_flag = 0	→ IDLE PACKET RESULT (OK) ara_flag = 0	→ IDLE PACKET RESULT (OK) ara_flag = 0	→ IDLE PACKET RESULT (OK) ara_flag = 0
E06_Interrupt (Start Condition detect)	→ START	-	-	-	-
E07/err07_Interrupt (Receive Buffer Full with slave address is write)	-	if (detect start condition) → RX	PACKET RESULT (PEC_ERROR) → ERROR	PACKET RESULT (PEC_ERROR) → ERROR	-
E08/err08_Interrupt (Receive Buffer Full)	-	-	read receive data from ICDRR if (rx_index over buffer size) PACKET RESULT (DATA_SIZE) → ERROR if (buffer size is NULL) PACKET RESULT (NOT_READY) → ERROR if (rx_index ==0) & (command supported) PACKET RESULT (CMD_WAIT) if (rx_index ==0) & (command not support) PACKET RESULT (CMD_NOT_SUPPORT) → ERROR	PACKET RESULT (INTERNAL) → ERROR	-
E09/err09_Interrupt (Transmit Interrupt with slave address is read)	-	PACKET RESULT (INTERNAL) → ERROR	if (PACKET RESULT == CMD_WAIT) & (detect restart condition) PACKET RESULT (CMD_COMP) → TX_RESP*1* if (PACKET RESULT != CMD_WAIT) PACKET RESULT	-	-

			(PEC_ERROR) → ERROR		
E0A/err0A_Interrupt (Transmit Interrupt)	-	-	-	set transmit data to ICDRT	-
E0B_Interrupt (Transmit End Interrupt)	-	-	-	-	-
E0C/err0C_Interrupt (Stop Condition detect)	-	→ IDLE	if (PACKET RESULT == CMD_WAIT) PACKET RESULT (CMD_COMP)*3 → IDLE if(rx_index==0) PACKET RESULT (CMD_QUICK)*3 → IDLE	if (tx bufer size is NULL) PACKET RESULT (NOT_READY) → ERROR if (PACKET RESULT == CMD_WAIT) PACKET RESULT (CMD_COMP)*3 → IDLE if (b_ara_flag == true) PACKET RESULT (CMD_ARA_COMP) *3 → IDLE	→ IDLE
err0D_Interrupt (Arbitlation Lost)	-	-	-	-	-
err0E_Interrupt (NACK detect)	-	-	-	-	-
err0F_Interrupt (Timeout Detect)	-	PACKET RESULT (TIMEOUT) → ERROR → IDLE*6	PACKET RESULT (TIMEOUT) → ERROR → IDLE*6	PACKET RESULT (TIMEOUT) → ERROR → IDLE*6	-

Table 27 PMBus Slave Application Layer status transitions table (ara_flag = 1)

This table shows the state transition table when ara_flag is 1.

	IDLE	**START**	**TX**	**ERROR *5**
E00/err00_Open	→ IDLE*7	→ IDLE*7	→ IDLE*7	→ IDLE*7
E01_Close	-	-	-	-
E02_EnablePEC	-	-	-	-
E03_DisablePEC	-	-	-	-
E04_SendARA	ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1
E05/err05_Restart	ara_flag = 0	→ IDLE PACKET RESULT (OK) ara_flag = 0	→ IDLE PACKET RESULT (OK) ara_flag = 0	→ IDLE PACKET RESULT (OK) ara_flag = 0
E06_Interrupt (Start Condition detects)	→ START	-	-	-
E07/err07_Interrupt (Receive Buffer Full with slave address is write)	-	PACKET RESULT (PEC ERROR) → →ERROR	PACKET RESULT (INTERNAL) → ERROR	-

E08/err08_Interrupt (Receive Buffer Full)	-	-	-	-
E09/err09_Interrupt (Transmit Interrupt with slave address is read)	-	if (detect start condition) → TX	PACKET RESULT (PEC_ERROR) → ERROR	-
E0A/err0A_Interrupt (Transmit Interrupt)	-	-	set transmit data to ICDRT if (buffer size is NULL) PACKET RESULT (NOT_READY) → ERROR	-
E0B_Interrupt (Transmit End Interrupt)	-	-	-	-
E0C/err0C_Interrupt (Stop Condition detect)	-	→ IDLE	→ IDLE	→ IDLE
err0D_Interrupt (Arbitlation Lost)	-	-	-	-
err0E_Interrupt (NACK detect)	-	-	-	-
err0F_Interrupt (Timeout Detect)	-	-	PACKET RESULT (TIMEOUT) → ERROR → IDLE*6	-
E00/err00_Open	→ IDLE*7	→ IDLE*7	→ IDLE*7	→ IDLE*7
E01_Close	-	-	-	-
E02_EnablePEC	-	-	-	-
E03_DisablePEC	-	-	-	-
E04_SendARA	ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1	→ IDLE PACKET RESULT (OK) ara_flag = 1

[NOTE]
*1. Execute user callback after state transition.
*2. Start sending response data.
*3. Start receiving the next command after executing the callback.
*4. Execute a callback to notify the user that the command is not supported.
*5. Execute a callback after ERROR status is changed, and notifies the user of the occurrence of the error.
*6. The callback is executed after ERROR state transition, and Restart process is executed immediately, and IDLE state is entered.
*7. Switches to Idle only when Open occurs after Close.

Table 28 PACKET_RESULT State Transition Table

This table shows the state transition table for PACKET_RESULT corresponding to Figure 40.

This table should be interpreted as follows.

- event consists of an abbreviation for the API name and the interrupt cause.
- status consists of an abbreviation for the "e_pmbus_nwk_status_s_t" element name.
- "If (<condition>)" means a conditional transition.
- [→<state>] means a transition to a state.
- [-] means no state transition.
- Light green indicates state transitions in Figure 40.

	OK	CMD_WAIT	CMD_COMP	CMD_QUICK	CMD_ARA_COMP	<ERROR> *
E00/err00_Open	-	-	-	-	-	-
E01_Close	-	-	-	-	-	-
E02_EnablePEC	-	-	-	-	-	-
E03_DisablePEC	-	-	-	-	-	-
E04_SendARA	-	→ OK	→ OK	→ OK	→ OK	→ OK
E05/err05_Restart	-	→ OK	→ OK	→ OK	→ OK	→ OK
E06_Interrupt (Start Condition detect)	-	-	-	-	-	-
E07/err07_Interrupt (Receive Buffer Full with slave address is write)	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	-
E08/err08_Interrupt (Receive Buffer Full)	if (first receive data and command is supported) → CMD_WAIT else → <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	-
E09/err09_Interrupt (Transmit Interrupt with slave address is read)	→ <ERROR>	if (slave address (read) detect with restart condition) → CMD_CMP else → <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	-
E0A/err0A_Interrupt (Transmit Interrupt)	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	
E0B_Interrupt (Transmit End Interrupt)	-	-	-	-	-	-
E0C/err0C_Interrupt (Stop Condition detect)	if (rx_index is zero (no receive data)) → CMD_QUICK	→ CMD_CMP	-	-	-	-

err0D_Interrupt (Arbitlation Lost)	-	-	-	-	-	-
err0E_Interrupt (NACK detect)	-	-	-	-	-	-
err0F_Interrupt (Timeout Detect)	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	-
callback end	-	-	if (ara_flag == false) → OK if (ara_flag == true) → CMD_ARA_COMP	→ OK	→ OK	→ OK

* For detailed conditions under which an <ERROR> occurs, see the PMBus Slave Application Layer state transition tables shown in Table 26 and Table 27.

5.2.2.2 PMBus Slave Driver Layer status transitions

The state transition of the PMBus Slave Driver Layer is divided into the transmit operation part and the receive operation part to PMBus Master, and the specified data is transmitted and received by the specified number of bytes. PMBus Slave Driver Layer status transitions are shown in Figure 42, Figure 43, Table 29 and Table 30.

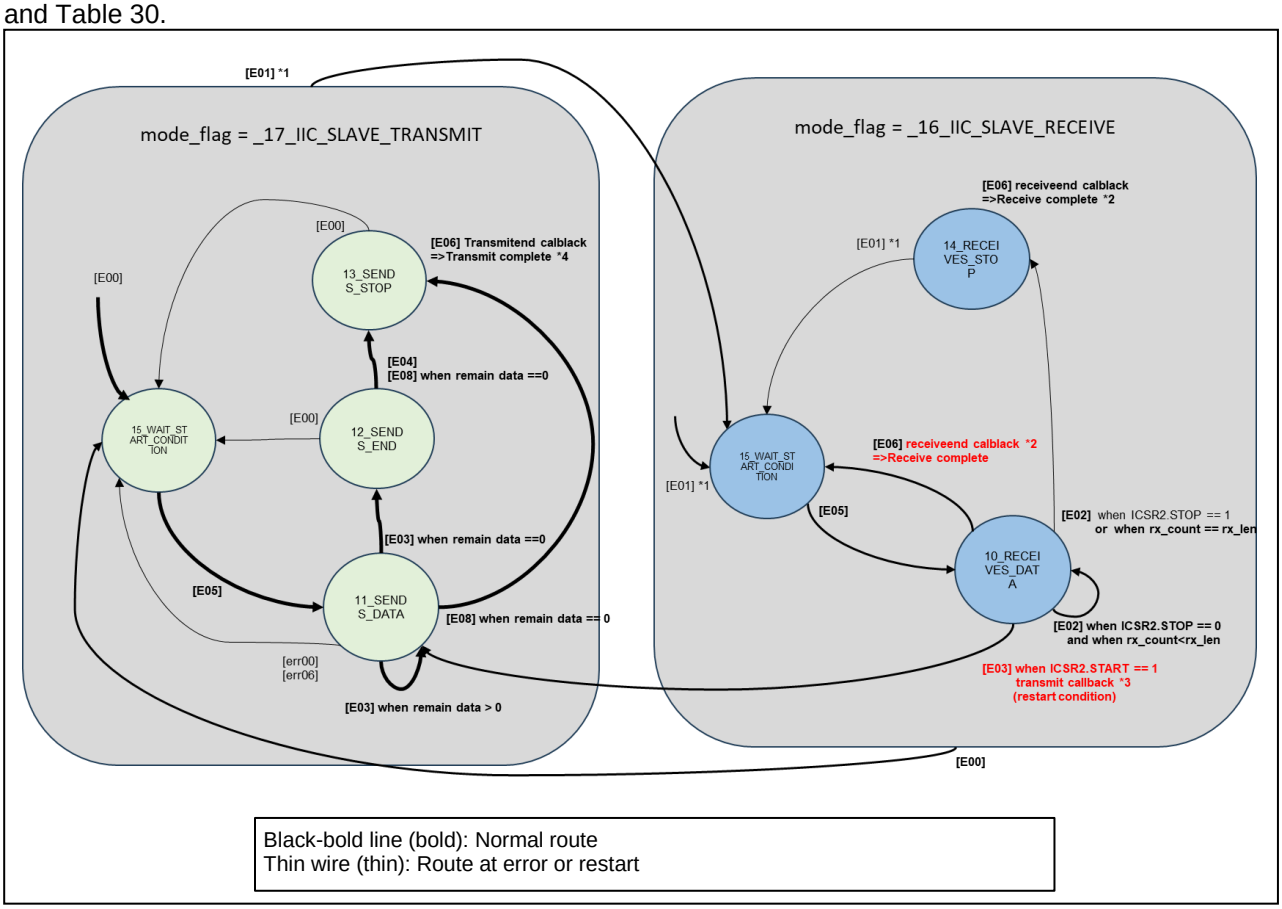


Figure 42 PMBus Slave Driver Layer status transitions

[Event List]
*[Exx] indicates normal Event, and [errxx] indicates Event when an error occurs.
Abnormal Event that are not shown in the list occur in any status, and Notify an error by callback error.
E00/err00_Send
E01/err01_Rreceive
E02_Interrupt (Receive Buffer Full)
E03_Interrupt (Transmit)
E04_Interrupt (Transmit End)
E05/err05_Interrupt (Start Condition Detect)
E06/err06_Interrupt (Stop Condition Detect)
err07_Interrupt (Arbitlation Lost)
E08/err08_Interrupt (NACK detect)

*1. PMBus demonstration system normally executes a Receive and waits for a read from master.
The receive data count initial value (total_data_size) specified in Receive API specifies the maximum number of Block Write protocols (35(command (1) + Data Size (1) + Data(32) + PEC (1)) so that the receive data count does not change to STOP condition reception wait during reception.
*2. If a Stop condition is detected prior to transition to 14, a Receive Event[E01 is generated in callback receivend to initialize the receive buffer settings, since reception operation is not completed.
Even when the maximum number of receive data is received, Receive Event[E01 is generated in callback receiveend and the receive buffer setting is initialized.
*3. Send Event[E00] is raised in callback transmit, mode_flag is updated to 17, and status is updated to 11. (Corresponds to detection of restart condition.)
*4. In transmitend callback, if Receive Event[E01] (PMBUS middleware is command receive mode for the next communication, or if Send Event[E00] (PMBus middleware is ARA send mode).

Figure 43 PMBus Master Driver status transition diagram (Supplement to Figure 42)

Table 29 PMBus Slave Driver Layer status transition table（when transmit）

This table shows the state transition when mode_flag is _17_IIC_SLAVE_TRANSMIT.
Explanations of the annotations are summarized in Table 30.

This table and Table 30 should be interpreted as follows.

- Since slave-address 10-bit is not used in PMBus, status control is omitted.
- The status control is omitted because RIIC0 timeout detection interrupt is not used in PMBus.
- [→<number>] is the number of the transition destination. (<number>) indicates the number of the transition destination of mode_flag.
- [-] means no state transition.
- [Callback <xxx>] refers to executing a callback. Callback error (<number>) means the error-information passed to callback error.
- [If (<Condition>)] refers to conditional operation.
- Red text indicates the process changed for PMBus middleware.
- Light green indicates state transitions when mode_flag=17 in Figure 42.
- Light blue indicates state transitions when mode_flag=16 in Figure 42.

	_15_IIC_SLAVE_WAIT_START_CONDITION	**_11_IIC_SLAVE_SENDS_DATA**	**_12_IIC_SLAVE_SENDS_END**	**13_IIC_SLAVE_SENDS_STOP**
E00/err00_Send	→ 15 (17)	→ 15 (17)	→ 15 (17)	→ 15 (17)
E01/err01_Rreceive	→ 15 (16)	→ 15 (16)	→ 15 (16)	→ 15 (16)
E02_Interrupt (Receive Buffer Full)	-	-	-	-
E03_Interrupt (Transmit)	-	if (0==remain data) → 12	-	-
E04_Interrupt (Transmit End)	-	-	→ 13	-
E05/err05_Interrupt (Start Condition Detect)	→ 11 callback start	-	callback error (MD_ERROR4)	-
E06/err06_Interrupt (Stop Condition Detect)	-	→ 15	callback error (MD_ERROR4)	callback transmitend *3
err07_Interrupt (Arbitlation Lost)	-	callback error (MD_ERROR1)	callback error (MD_ERROR1)	callback error (MD_ERROR1)
E08/err08_Interrupt (NACK detect)	-	if (0 == remain data) → 13 if (0 < remain data) callback transmitend *3 callback error (MD_ERROR3)	if (0 == remain data) → 13 if (0 < remain data) callback transmitend *3 callback error (MD_ERROR3)	if (0 == remain data) → 13 if (0 < remain data) callback transmitend *3 callback error (MD_ERROR3)

Table 30 PMBus Slave Driver Layer status transition table（when receive）

This table shows the state transition when mode_flag is _16_IIC_SLAVE_RECEIVE.

	_15_IIC_SLAVE_WAIT_START_CONDITION	_10_IIC_SLAVE_RECEIVE_DATA	_14_IIC_SLAVE_RECEIVE_STOP
E00/err00_Send	→ 15 (17)	→ 15 (17)	→ 15 (17)
E01/err01_Rreceive	→ 15 (16)	→ 15 (16)	→ 15 (16)
E02_Interrupt (Receive Buffer Full)	-	if (dummy_count < 1) callback receive if (ICSR2.STOP == 1) || (rx_count == rx_len) → 14 callback receive	-
E03_Interrupt (Transmit)	-	if (ICSR2.START == 1) callback transmit*1 → 11 (15)	-
E04_Interrupt (Transmit End)	-	*1 → 15 (17)	-
E05/err05_Interrupt (Start Condition Detect)	→ 10 callback start	-	-
E06/err06_Interrupt (Stop Condition Detect)	-	→ 15 callback receiveend*2	callback receiveend *2
err07_Interrupt (Arbitlation Lost)	-	callback error (MD_ERROR1)	callback error (MD_ERROR1)
E08/err08_Interrupt (NACK detect)	-	callback error (MD_ERROR3)	callback error (MD_ERROR3)

[NOTE]

*1. Raises an Send Event in callback transmit, updates mode_flag to 17, and then updates the status to 11. (Corresponds to detection of restart condition.)

*2. If a Stop condition is detected prior to transition to 14, a Receive Event[E01 is generated in callback receivend to initialize the receive buffer settings, since the receive operation is not completed. Even when the maximum number of receive data is received, Receive Event[E01 is generated in callback receiveend and the receive buffer setting is initialized.

*3. In transmit end callback, if Receive Event[E01] (PMBUS middleware is command receive mode for the next communication, or if Send Event[E00] (PMBus middleware is ARA send mode).

5.2.3 PMBus Slave Function List

PMBus Slave functions are divided into Table 31 and Table 32 of Application functions, API functions in Table 33, Middleware functions in Table 34, and driver functions in Table 35 and Table 36 generated by the Smart Configurator and FSP. Some of the driver functions have been changed according to PMBus Slave process. Refer to Customizing 5.2.4 PMBus Slave Driver section for details.

Table 31 PMBus Slave Application RX26T Function List

File Name	Function Name	Function
pmbus_app\ r_app_pmbus_main.c	main	The main process for the application in which the initialization process of PMBus Middleware is added to the existing motor sample.
	r_app_main_start_pmbus_ctrl	Open PMBUS middleware.
	pmbus_ctrl	Execute the process corresponding to PMBUS command received in the main process.
	r_pmbus_callback	Callback to register with PMBus Middleware. Execute process corresponding to the received command code.
	r_app_pmbus_exe_write_command	Execute the process corresponding to the write transaction code command. Called from the callback function.
	r_app_pmbus_exe_read_command	Execute the process corresponding to the read transaction code command. Called from the callback function.
pmbus_app\r_app_board_ui.c	r_app_board_ui_mainloop	This process comment the motor rotation start control when SW1 is ON in the motor sample.

Table 32 PMBus Slave Application RA6T3 Function List

File Name	Function Name	Function
src\hal_entry.c	hal_entry	The main process for the application in which the initialization process of PMBus Middleware is added to the existing motor sample.
pmbus_app\main\ mtr_main.c	r_app_main_start_pmbus_ctrl	Open PMBus Middleware.
	mtr_main	The main process for the application in which the initialization process of PMBus Middleware is added to the existing motor sample.
	board_ui	This process monitors the status of the board UI of the motor sample and controls the motor.
	pmbus_ctrl	Execute the process corresponding to PMBUS command received in the main process.
	r_pmbus_callback	Callback to register with PMBus Middleware. Execute process corresponding to the received command code.
	r_app_pmbus_exe_write_command	Execute the process corresponding to the write transaction code command. Called from the callback function.
	r_app_pmbus_exe_read_command	Execute the process corresponding to the read transaction code command. Called from the callback function.

Table 33 PMBus Slave API Function List

File Name	Function Name	Function
r_pmbus_app_slave.c	R_PMBUS_Slave_Open	Open PMBus Middleware and wait for a command from the master.
	R_PMBUS_Slave_Close	Close PMBus Middleware.
	R_PMBUS_Slave_EnablePEC	Enable sending and receiving packets to which PEC has been added.
	R_PMBUS_Slave_DisablePEC	Disable sending and receiving packets with PEC.
	R_PMBUS_Slave_SendARA	Change to wait for Alert Response protocol-response.
	R_PMBUS_Slave_Restart	Return from Alert Response protocol-response wait state to the command-reception wait state.

Table 34 PMBus Slave Middleware Function List

File Name	Function Name	Function
r_pmbus_app_slave.c	r_pmbus_app_InitCtrl	Initialize PMBus Middleware parameters.
	r_pmbus_app_int_TransmitEnd	Execute transmit end callback process.
	r_pmbus_app_int_ReceiveEnd	Execute receive end callback process.
	r_pmbus_app_int_Receive	Execute receive callback process.
	r_pmbus_app_int_Transmit	Execute transmit callback process.
	r_pmbus_app_int_Notify	Execute error detection callback process.
	r_pmbus_app_CheckCommandSupport	Check whether the received command is supported by PMBus spec.
r_pmbus_nwk_slave.c	r_pmbus_nwk_Startlave	Start PMBus slave-receive operation.
	r_pmbus_nwk_Stoplave	Stop PMBus slave-operation.
	r_pmbus_nwk_ResetSlave	Reset PMBus slave-operation.
	r_pmbus_nwk_StartendARA	Start PMBus ARA protocol-response operation.
	r_pmbus_nwk_ReStartlave	Restart PMBus slave-receive process.
	r_pmbus_nwk_ProcessStart	Execute PMBus start condition detecting process.
	r_pmbus_nwk_ProcessRx	Execute PMBus slave-receive process.
	r_pmbus_nwk_ProcessTx	Execute PMBus slave-transmit process.
	r_pmbus_nwk_ProcessStop	Execute PMBus stop condition detecting process.
	r_pmbus_nwk_ProcessErrorNotice	Execute the process when an interrupt is detected at an unexpected timing of PMBus.
	r_pmbus_nwk_ProcessStartRead	Execute preprocess when the slave receive mode is detected.
	r_pmbus_nwk_ProcessStartWrite	Execute preprocess when slave transmit mode is detected.
	r_pmbus_nwk_ProcessAfterStop	Execute post-processing after PMBus stop condition detection.
	r_pmbus_nwk_ProcessCallback	Execute a user-registered callback function.
	r_pmbus_nwk_AddCrc8	Execute a CRC operation on a single file.
	r_pmbus_nwk_CaluculatePECAtSlave	Execute a CRC operation on more than one data. (Slave address is not included.)
r_pmbus_wrapper.c	r_pmbus_wrapper_I2cStart	Start the I2C Driver. This function is a wrapper function that absorbs the differences in driver code between RX26T and RA6T3.

	r_pmbus_wrapper_I2cStop	Stop the I2C Driver. This function is a wrapper function that absorbs the differences in driver code between RX26T and RA6T3.
	r_pmbus_wrapper_I2cReceive	Start the I2C driver slave receive operation. This function is a wrapper function that absorbs the differences in driver code between RX26T and RA6T3.
	r_pmbus_wrapper_I2cSend	Start the I2C driver slave transmit operation. This function is a wrapper function that absorbs the differences in driver code between RX26T and RA6T3.
	r_pmbus_wrapper_TimerOpen	Start the timer driver. This function is a wrapper function that absorbs the differences in driver code between RX26T and RA6T3.
	r_pmbus_wrapper_TimerClose	Stop the timer driver. This function is a wrapper function that absorbs the differences in driver code between RX26T and RA6T3.
	r_pmbus_wrapper_TimerStart	Start the timer driver count operation. This function is a wrapper function that absorbs the differences in driver code between RX26T and RA6T3.
	r_pmbus_wrapper_TimerStop	Stop the timer driver count operation. This function is a wrapper function that absorbs the differences in driver code between RX26T and RA6T3.
	r_smbus_handle_callback	Callback function for I3C interrupt. This function is only used RA6T3.
	g_gpt5_i2c_timeout_callback	Callback function for the GPT interrupt of the channel that monitors communication timeout. This function is only used RA6T3.

Table 35 Smart Configurator Function List (RX26T)

File Name	Function Name	Function	Changes from diversion source
Config_RIIC0.c	R_Config_RIIC0_Create	Initialize RIIC0 drivers that can specify slave addresses.	Yes
	R_Config_RIIC0_Stop	Stop the operation of RIIC0 in slave mode, including the slave address detection disable setting.	Yes
	R_Config_RIIC0_SLave_Receive	Start the slave receive operation.	Yes
Config_TMR0.c	r_Config_TMR0_Start	Start the timer count.	Yes
Config_TMR0_User.c	r_Config_TMR0_cmia0_interrupt	Interrupt process for the compare match interrupt A.	Yes
Config_RIIC0_user.c	r_Config_RIIC0_transmit_interrupt	Interrupt process for the transmit buffer empty interrupt.	Yes
	r_Config_RIIC0_receive_interrupt	Interrupt process for the receive buffer full interrupt.	Yes
	r_Config_RIIC0_error_interrupt	Interrupt process for error detection interrupt.	Yes
	r_Config_RIIC0_callback_transmitend	Transmit end callback function.	Yes
	r_Config_RIIC0_callback_receiveend	Receive end callback function.	Yes
	r_Config_RIIC0_callback_error	Error callback function when error is detected by various interrupts of RIIC0.	Yes
	r_User_RIIC0_callback_start	Start condition detection callback function.	New
	r_User_RIIC0_callback_transmit	Transmit callback function.	New
	r_User_RIIC0_callback_receive	Receive callback function.	New

Table 36 FSP Function List (RA6T3)

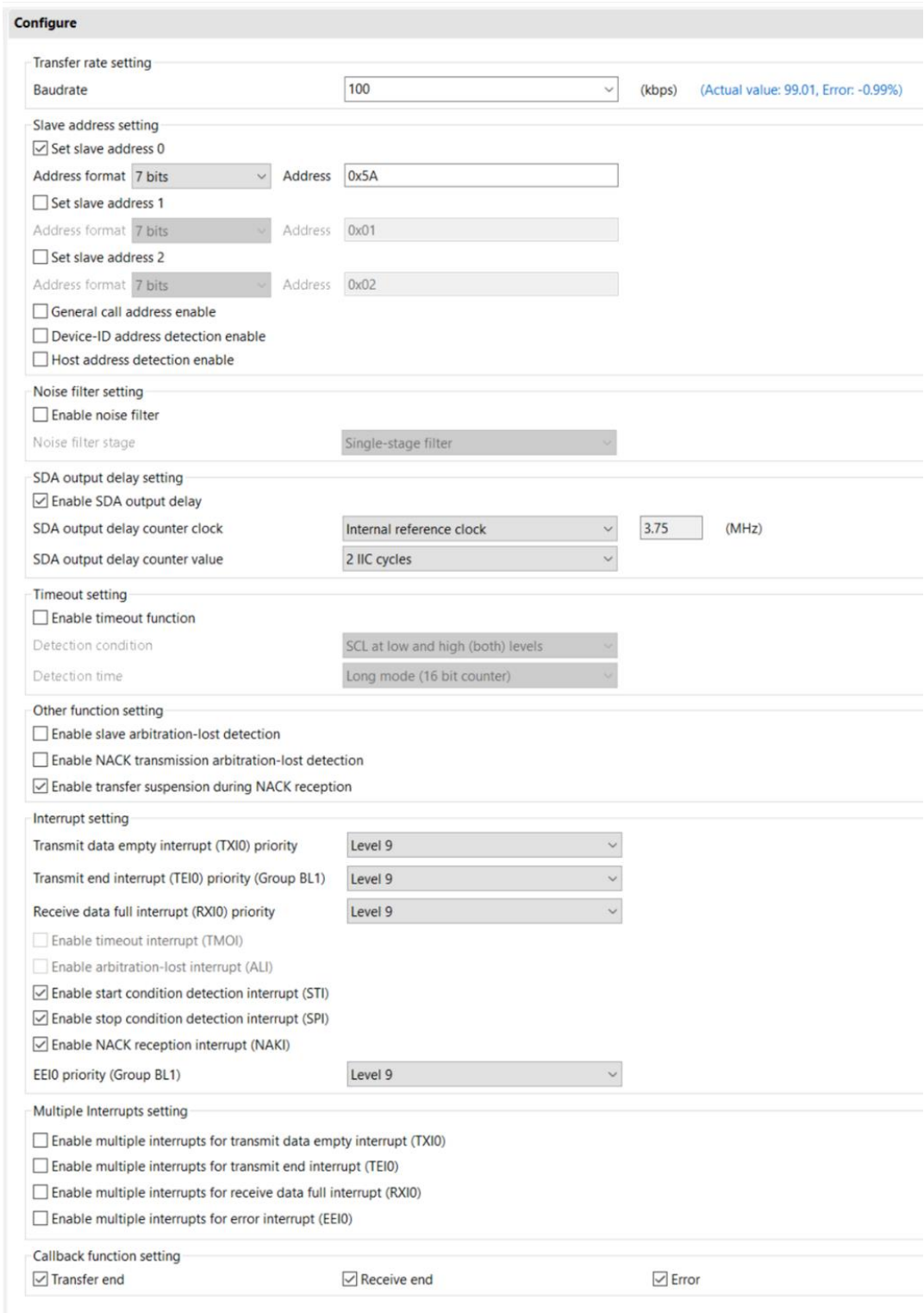
File Name	Function Name	Function	Changes from diversion source
\pmbus_slave\r_smbus_slave.c	R_SMBUS_SLAVE_Open	Execute the open process of SMBUS drivers.	New
	r_smbus_slave_read_write	Execute the transmission start/reception start process of SMBUS drivers.	New
	r_smbus_slave_notify	Calls the post-process and callback when the error interrupt of SMBUS driver is detected.	New
	r_smbus_slave_callback_request	Calls the callback function when an interrupt occurs in SMBUS drivers.	New
	r_smbus_open_hw_slave	Initialize I3C registers.	New
	r_smbus_slave_call_callback	Call the callback function of SMBUS driver.	New
	r_smbus_rxi_check_illegal_start	Check Start condition detection status when the receive buffer full interrupt of SMBUS drivers is generated.	New
	r_smbus_rxi_slave	Receive buffer full interrupt process of SMBUS drivers.	New
	r_smbus_txi_slave	Transmit buffer empty interrupt process of SMBUS drivers.	New
	r_smbus_tei_slave	Transmit completion interrupt process of SMBUS drivers.	New
	r_smbus_err_slave	Error-detection interrupt process of SMBUS drivers.	New

5.2.4 Customizing PMBus Slave Drivers

PMBus Slave driver code (RIIC0, TMR) is generated by the smart configurator of RX26T and FSP of RA6T3. RX26T modifies and adds some of RIIC0, and TMR processes by protecting the user code in the Smart Configurator. RA6T3 registers the code generated by FSP as a separate function. The driver is dedicated to PMBus Slave. The following shows RX26T smart configurator settings, FSP settings for RA6T3 to change the generated driver code, and the changes for the generated driver.

(1) Customizing Smart Configurator (RX26T)

- Setting Smart Configurator RIIC0 (RX26T)



- List of Changes in RIIC0 Driver Codes Generated by Smart Configurator (RX26T)

Function Name	r_Config_RIIC0_Slave_Create()
File Name	Config_RIIC0.c
Change Details	RIIC0's initialfunction. Adds the process of overwriting SARL0 by adding a global-variable g_riic0_user_slave_addr so that the user-specified slaveaddress can be set to SARL0 register.

Before change

```
/* Start user code for global. Do not edit comment generated here */

/* End user code. Do not edit comment generated here */
```

```
    RIIC0.SARL0.BYTE = 0xB4U;

    /* Set ICSER */
```

After change

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_riic0_user_slave_addr = 0;          /* User specified slave address */
```

```
    RIIC0.SARL0.BYTE = 0xB4U;

    /* Start user code */
    /* Reset slave address for user specification */
    RIIC0.SARL0.BYTE = g_riic0_user_slave_addr << 1;

    /* End user code */
    /* Set ICSER */
```

Function Name	R_Config_RIIC0_Stop()
File Name	Config_RIIC0.c
Change Details	This function disable the function by disabling the interrupt in RIIC0. Adds clearing of ISCER register to disable slave address detection when RIIC0 function is stopped.

Before change

```
    EN(RIIC0, EEIO) = 0U;
```

After change

```
    EN(RIIC0, EEIO) = 0U;
    /* Start user code */
    /* Clear slave address detection */
    RIIC0.ICSER.BYTE = 0;

    /* End user code */
```

Function Name	R_Config_RIIC0_Slave_Receive()
File Name	Config_RIIC0.c
Change Details	This function Start the reception operation of RIIC0 (slaves). Add the g_riic0_start_detect_at_receive initialization process.

Before change

```
    g_riic0_dummy_read_count = 0U;

    g_riic0_mode_flag = _16_IIC_SLAVE_RECEIVE;
```

After change

```
    g_riic0_dummy_read_count = 0U;

    /* Start user code */
    g_riic0_start_detect_at_receive = 0;

    /* End user code */
    g_riic0_mode_flag = _16_IIC_SLAVE_RECEIVE;
```

Function Name	r_User_RIIC0_callback_Start()
File Name	Config_RIIC0_user.c
Change Details	This is a new callback function added to execute when a Start condition detection interrupt occurs in Slave. Execute the r_pmbus_app_int_Notify (E_PMBUS_INT_EVENT_S_START) to execute the process of starting slave transmission/reception.

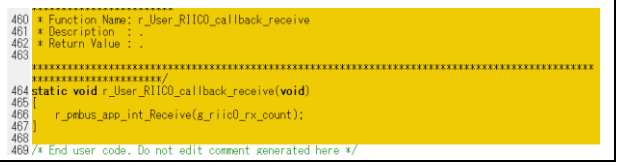
Before change

None.

After change

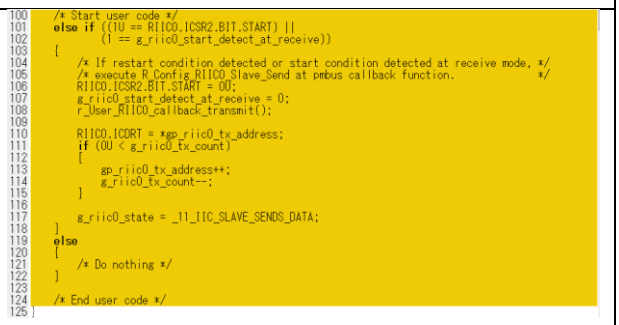
```
/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************************************************************
 * Function Name: r_User_RIIC0_callback_start
 * Description  : .
 * Return Value : .
 ***********************************************************************************************************************/
static void r_User_RIIC0_callback_start(void)
{
    r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_START);
}
```

Function Name	r_User_RIIC0_callback_Receive()
File Name	Config_RIIC0_user.c
Change Details	This is the callback function for the receive buffer full interrupt added by PMBus Middleware. r_pmbus_app_int_Receive() is executed to check and update PMBus Middleware status each time a receive buffer full interrupt is generated. Within r_pmbus_app_int_Receive(), If the "g_riic0_rx_count" (currently received data count) is 0, the r_pmbus_nwk_ProcessStart() is internally executed, and the reception Start after the "p_pmbusSlaveCmdCallback" command-process is executed. If "g_riic0_rx_count" is not 0, the r_pmbus_nwk_ProcessRx is executed internally and reception is executed.

Before change	After change
None.	

```
/*****************************************************************************
* Function Name: r_User_RIIC0_callback_receive
* Description  : .
* Return Value : .
******************************************************************************
******************************************************************************/
static void r_User_RIIC0_callback_receive(void)
{
    r_pmbus_app_int_Receive(g_riic0_rx_count);
}
/* End user code. Do not edit comment generated here */
```

Function Name	r_Config_RIIC0_transmit_interrupt ()
File Name	Config_RIIC0_user.c
Change Details	Transmit buffer empty interrupt handler. In response to a start condition (restart condition) detected by RIIC0 driver in transmit mode, the following process is executed when ICSR2.START is 1 or the g_riic0_start_detect_at_receive is 1. -Clear ICSR2.START. -The g_riic0_start_detect_at_receive is cleared. -Execute r_User_RIIC0_callback_transmit() and Execute PMBus Middleware process when a restart condition is detected. -Set the *gp_riic0_tx_address in ICDRT. -If the g_riic9_tx_count is greater than 0, increment the gp_riic0_tx_address and decrement the g_riic0_tx_count.

Before change	After change
	

```
    /* Start user code */
    else if ((1U == RIIC0.ICSR2.BIT.START) ||
             (1 == g_riic0_start_detect_at_receive))
    {
        /* If restart condition detected or start condition detected at receive mode, */
        /* execute R_Config_RIIC0_Slave_Send at pmbus callback function.              */
        RIIC0.ICSR2.BIT.START = 0U;
        g_riic0_start_detect_at_receive = 0;
        r_User_RIIC0_callback_transmit();

        RIIC0.ICDRT = *gp_riic0_tx_address;
        if (0U < g_riic0_tx_count)
        {
            gp_riic0_tx_address++;
            g_riic0_tx_count--;
        }

        g_riic0_state = _11_IIC_SLAVE_SENDS_DATA;
    }
    else
    {
        /* Do nothing */
    }

    /* End user code */
}
```

Function Name	r_Config_RIIC0_receive_interrupt ()
File Name	Config_RIIC0_user.c
Change Details	Receive buffer full interrupt handler. When the g_riic0_state is _10_IIC_SLAVE_RECEIVE_DATA, add the following process. -When ICSR2.START is 1, ICDRR is dummy-read and "g_riic0_dummy_read_count" is incremented to execute the r_Config_RIIC9_callback_error (MD_ERROR4). -Adds PMBus Middleware operation by executing r_User_RIIC0_callback_receive() to the operation when the g_riic0_dummy_read_count is greater than 1. -After reading RIIC0.ICDRR, add PMBus Middleware action by executing r_User_RIIC0_callback_receive(). When the g_riic0_state is _10_IIC_SLAVE_SEND_DATA, PMBus Middleware process by executing r_User_RIIC0_callback_receive() is added.

Before change

```
135     if (_10_IIC_SLAVE_RECEIVES_DATA == g_riic0_state)
136     {
137         if (1U > g_riic0_dummy_read_count)
138         {
139             dummy = RIIC0.ICDRR;
140             g_riic0_dummy_read_count++;
141             return;
142         }
148         g_riic0_rx_count++;
149
150         if (1U == RIIC0.ICSR2.BIT.STOP)
```

After change

```
163     if (_10_IIC_SLAVE_RECEIVES_DATA == g_riic0_state)
164     {
165         /* Start user code */
166         /* If an unexpected restart condition is detected, error processing is performed. */
167         if (1U == RIIC0.ICSR2.BIT.START)
168         {
169             dummy = RIIC0.ICDRR;
170             g_riic0_dummy_read_count++;
171             r_Config_RIIC0_callback_error(MD_ERROR4);
172
173             return;
174         }
175
176         /* End user code */
177         if (1U > g_riic0_dummy_read_count)
178         {
179             dummy = RIIC0.ICDRR;
180             g_riic0_dummy_read_count++;
181             /* Start user code */
182             r_User_RIIC0_callback_receive();
183
184             /* End user code */
185             return;
186         }
192         g_riic0_rx_count++;
193
194         /* Start user code */
195         r_User_RIIC0_callback_receive();
196
197         /* End user code */
198         if (1U == RIIC0.ICSR2.BIT.STOP)
```

Function Name	r_Config_RIIC0_error_interrupt ()
File Name	Config_RIIC0_user.c
Change Details	This is a callback function for error detection interrupts. To enable PMbus communication startup process when a Start condition is detected, Execute r_User_RIIC0_callback_start() when "g_riic0_state" is "_15_IIC_SLAVE_WAIT_START_CONDITION". (2 places) In addition, initialize the g_riic0_start_detect_at_receive in reception mode (_16_IIC_SLAVE_RECEIVE = g_riic0_mode_flag). In reception mode (_16_IIC_SLAVE_RECEIVE = g_riic0_mode_flag), if "g_riic0_state" is "_18_IIC_SLAVE_WAIT_RESTART_CONDITION", add a branch that does not disable Start condition interrupt by "RIIC0.ICIER.BIT.SPIE".

Before change

```
230         else if (_15_IIC_SLAVE_WAIT_START_CONDITION == g_riic0_state)
231         {
232             RIIC0.ICSR2.BIT.START = 0U;
233             RIIC0.ICIER.BIT.STIE = 0U;
234             RIIC0.ICIER.BIT.SPIE = 1U;
235             g_riic0_state = _10_IIC_SLAVE_RECEIVES_DATA;
236         }
237         else if (_10_IIC_SLAVE_RECEIVES_DATA == g_riic0_state)
238         {
239             RIIC0.ICSR2.BIT.STOP = 0U;
240             RIIC0.ICIER.BIT.SPIE = 0U;
241             RIIC0.ICIER.BIT.STIE = 1U;
242             g_riic0_state = _15_IIC_SLAVE_WAIT_START_CONDITION;
243         }
244         else
259         else if (_15_IIC_SLAVE_WAIT_START_CONDITION == g_riic0_state)
260         {
261             RIIC0.ICSR2.BIT.START = 0U;
262             RIIC0.ICIER.BIT.STIE = 0U;
263             RIIC0.ICIER.BIT.SPIE = 1U;
264             g_riic0_state = _11_IIC_SLAVE_SENDS_DATA;
265         }
```

After change

```
282         else if (_15_IIC_SLAVE_WAIT_START_CONDITION == g_riic0_state)
283         {
284             RIIC0.ICSR2.BIT.START = 0U;
285             RIIC0.ICIER.BIT.STIE = 0U;
286             RIIC0.ICIER.BIT.SPIE = 1U;
287             g_riic0_state = _10_IIC_SLAVE_RECEIVES_DATA;
288             /* Start user code */
289             /* set start condition detection flag. */
290             g_riic0_start_detect_at_receive = 1;
291             r_User_RIIC0_callback_start();
292
293             /* End user code */
295         else if (_10_IIC_SLAVE_RECEIVES_DATA == g_riic0_state)
296         {
297             RIIC0.ICSR2.BIT.STOP = 0U;
298             RIIC0.ICIER.BIT.SPIE = 0U;
299             RIIC0.ICIER.BIT.STIE = 1U;
300             g_riic0_state = _15_IIC_SLAVE_WAIT_START_CONDITION;
301             /* Start user code */
302             /* If detect stop condition before receive buffer is full, execute pmbus command,
*/
303             /* or error notification from master.
*/
304             r_Config_RIIC0_callback_receiveend();
305
306             /* End user code */
307         }
308         else
323         else if (_15_IIC_SLAVE_WAIT_START_CONDITION == g_riic0_state)
324         {
325             RIIC0.ICSR2.BIT.START = 0U;
326             RIIC0.ICIER.BIT.STIE = 0U;
327             RIIC0.ICIER.BIT.SPIE = 1U;
328             g_riic0_state = _11_IIC_SLAVE_SENDS_DATA;
329             /* Start user code */
330             r_User_RIIC0_callback_start();
331
332             /* End user code */
333         }
```

Function Name	r_Config_RIIC0_callback_transmitend ()
File Name	Config_RIIC0_user.c
Change Details	This is the callback function for Stop condition detection interrupt during transmit operation. Execute r_pmbus_app_int_TransmitEnd() to execute PMBus Middleware transmit completion process after Stop condition detected interrupt.

Before change

```
291 static void r_Config_RIIC0_callback_transmitend(void)
292 {
293     /* Start user code for r_Config_RIIC0_callback_transmitend. Do not edit comment generated
    here */
294     /* End user code. Do not edit comment generated here */
295 }
```

After change

```
359 static void r_Config_RIIC0_callback_transmitend(void)
360 {
361     /* Start user code for r_Config_RIIC0_callback_transmitend. Do not edit comment generated
    here */
362     r_pmbus_app_int_TransmitEnd();
363
364     /* End user code. Do not edit comment generated here */
365 }
```

Function Name	r_Config_RIIC0_callback_receiveend ()
File Name	Config_RIIC0_user.c
Change Details	This is the callback function for Stop condition detection interrupt during receive operation. After Stop condition is detected, execute r_pmbus_app_int_ReceiveEnd() to execute PMBus Middleware reception completion process and "r_pmbusSlaveCmdCallback".

Before change

```
291 static void r_Config_RIIC0_callback_transmitend(void)
292 {
293     /* Start user code for r_Config_RIIC0_callback_transmitend. Do not edit comment generated here */
294     /* End user code. Do not edit comment generated here */
295 }
```

After change

```
359 static void r_Config_RIIC0_callback_transmitend(void)
360 {
361     /* Start user code for r_Config_RIIC0_callback_transmitend. Do not edit comment generated here */
362     r_pmbus_app_int_TransmitEnd();
363
364     /* End user code. Do not edit comment generated here */
365 }
```

Function Name	r_Config_RIIC0_callback_error ()
File Name	Config_RIIC0_user.c
Change Details	This is a callback function to be executed when the interrupt source is an error when an error is detected. For "MD_ERROR4" (Start condition detected outside driver sequence): If the START bit in ICSR2 is 1, clear the START bit in ICSR2 and then execute r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_START_UNEXPECTED) to update the state of PMBus Middleware.

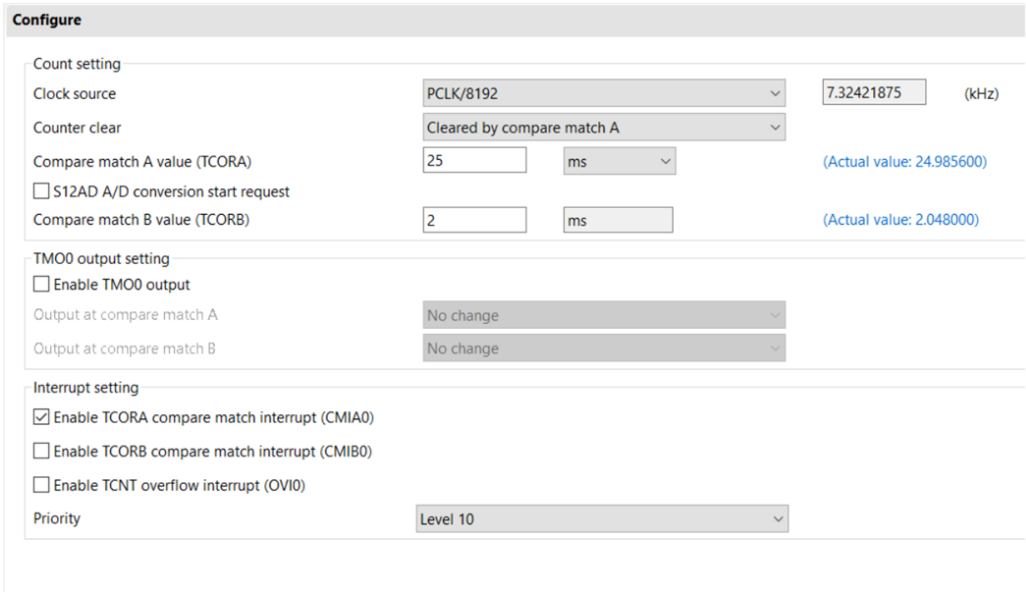
Before change

```
344         case MD_ERROR4:
345         {
346             /* Start user code for communication sequence error. Do not edit comment generated here */
347             /* End user code. Do not edit comment generated here */
348             break;
349         }
```

After change

```
416         case MD_ERROR4:
417         {
418             /* Start user code for communication sequence error. Do not edit comment generated here */
419             if (1U == RIIC0.ICSR2.BIT.START)
420             {
421                 RIIC0.ICSR2.BIT.START = 0;
422                 r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_START_UNEXPECTED);
423             }
424
425             /* End user code. Do not edit comment generated here */
426             break;
427         }
```

- Setting smart configurator TMR (RX26T)



- List of Changes in TMR Driver Codes Generated by Smart Configurator (RX26T)

Function Name	r_Config_TMR0_cmia0_interrupt ()
File Name	Config_TMR0_User.c
Change Details	This is a callback function to be executed when the interrupt source is an error when an error is detected. Execute r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_TIMEOUT) to update the status of PMBus Middleware.

Before change

```
static void r_Config_TMR0_cmia0_interrupt(void)
{
    /* Start user code for r_Config_TMR0_cmia0_interrupt. Do not edit comment generated here */

    /* End user code. Do not edit comment generated here */
}
```

After change

```
static void r_Config_TMR0_cmia0_interrupt(void)
{
    /* Start user code for r_Config_TMR0_cmia0_interrupt. Do not edit comment generated here */
    r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_TIMEOUT);

    /* End user code. Do not edit comment generated here */
}
```

Function Name	r_Config_TMR0_Start()
File Name	Config_TMR0.c
Change Details	This API Start counting. Add process to initialize the counter register (TCNT) at the start of the function.

Before change

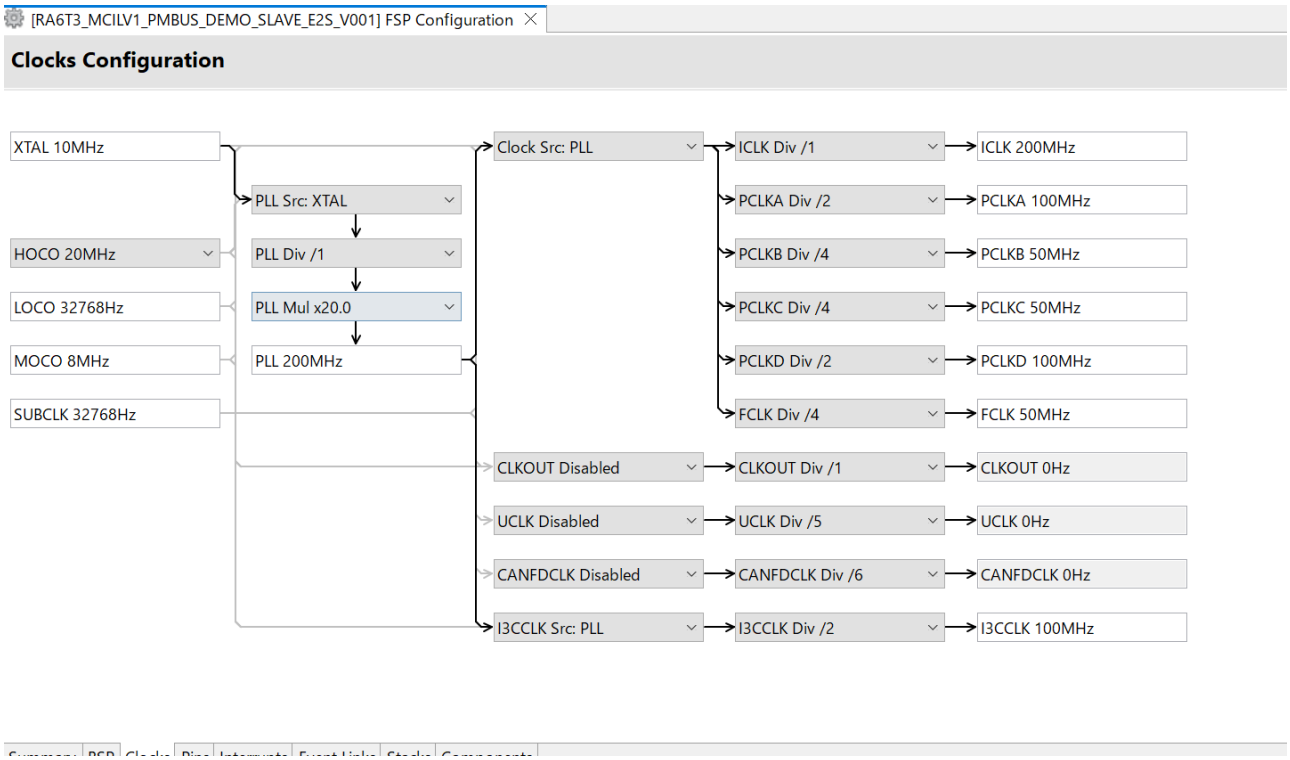
```
{

    /* Enable TMR0 interrupt */
    IR(TMR0, CMIA0) = 0U;
```

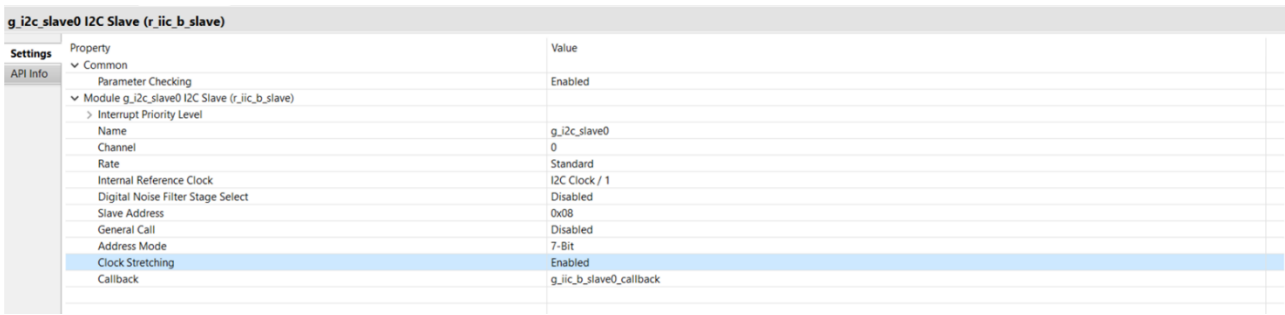
After change

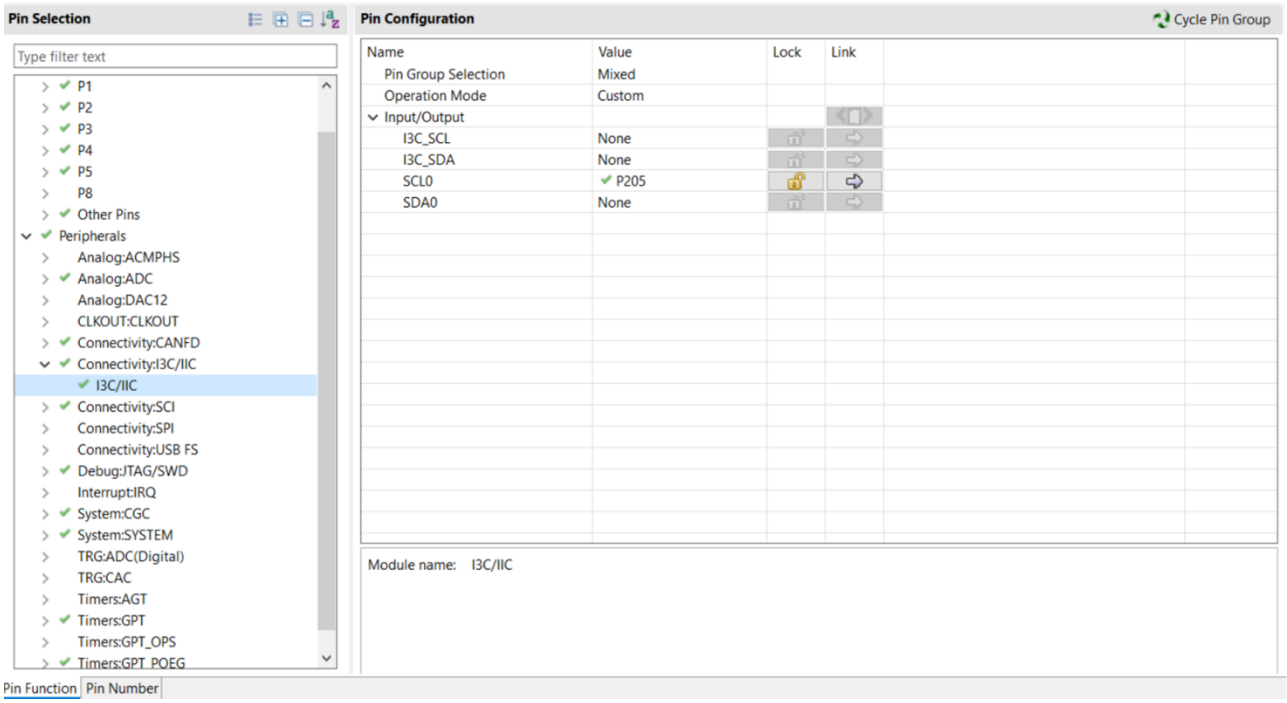
```
    /* Start user code */
    /* Clear Timer count*/
    TMR0.TCNT = 0U;

    /* End user code */
```

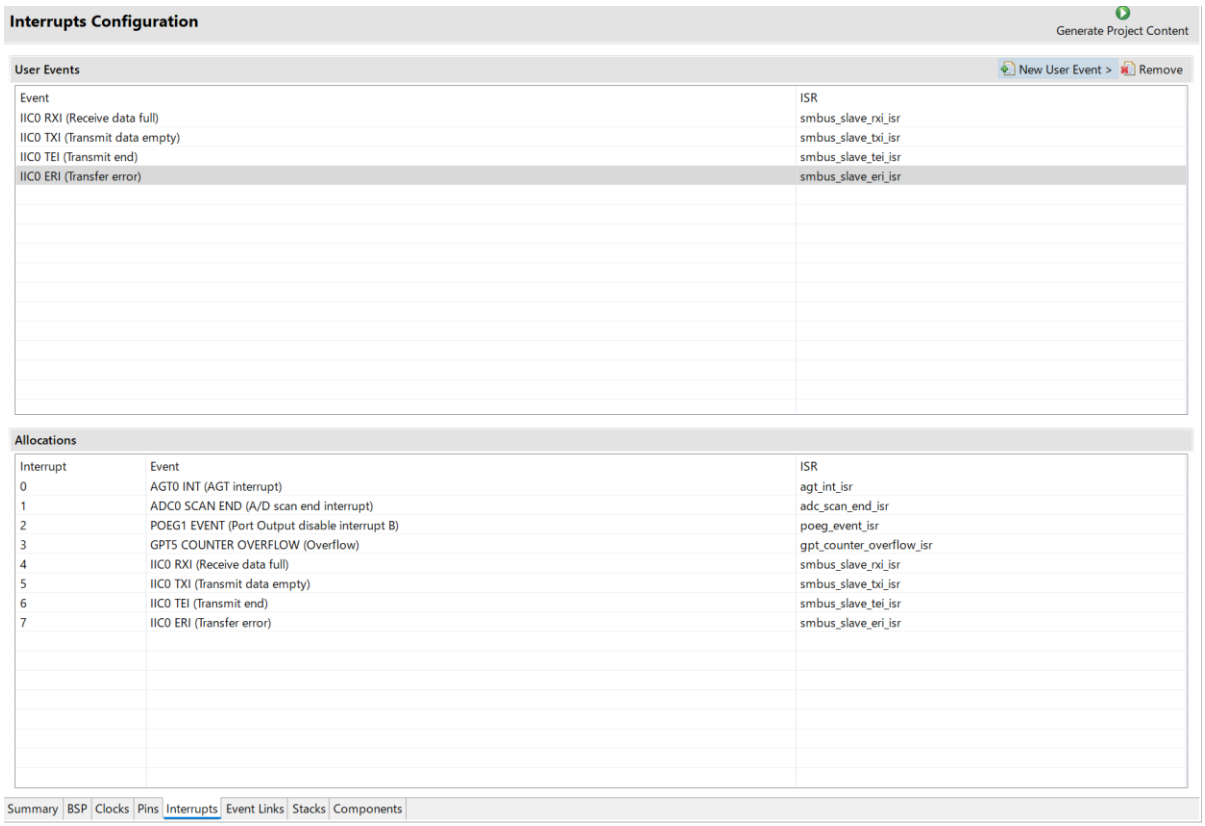
(2) Customizing FSP (RA6T3)

- Setting FSP I3C (RA6T3)






Pin Selection
Type filter text
P1
P2
P3
P4
P5
P8
Other Pins
Peripherals
Analog:ACMPHS
Analog:ADC
Analog:DAC12
CLKOUT:CLKOUT
Connectivity:CANFD
Connectivity:I3C/IIC
I3C/IIC
Connectivity:SCI
Connectivity:SPI
Connectivity:USB FS
Debug:JTAG/SWD
Interrupt:IRQ
System:CGC
System:SYSTEM
TRG:ADC(Digital)
TRG:CAC
Timers:AGT
Timers:GPT
Timers:GPT_OPS
Timers:GPT_POEG
Pin Configuration
Cycle Pin Group
Name
Value
Lock
Link
Pin Group Selection
Mixed
Operation Mode
Custom
Input/Output
I3C_SCL
None
I3C_SDA
None
SCL0
P205
SDA0
None
Module name: I3C/IIC
Pin Function
Pin Number


Interrupts Configuration
Generate Project Content
User Events
New User Event >
Remove
Event
ISR
IIC0 RXI (Receive data full)
smbus_slave_rxi_isr
IIC0 TXI (Transmit data empty)
smbus_slave_txi_isr
IIC0 TEI (Transmit end)
smbus_slave_tei_isr
IIC0 ERI (Transfer error)
smbus_slave_eri_isr
Allocations
Interrupt
Event
ISR
0
AGT0 INT (AGT interrupt)
agt_int_isr
1
ADC0 SCAN END (A/D scan end interrupt)
adc_scan_end_isr
2
POEG1 EVENT (Port Output disable interrupt B)
poeg_event_isr
3
GPT5 COUNTER OVERFLOW (Overflow)
gpt_counter_overflow_isr
4
IIC0 RXI (Receive data full)
smbus_slave_rxi_isr
5
IIC0 TXI (Transmit data empty)
smbus_slave_txi_isr
6
IIC0 TEI (Transmit end)
smbus_slave_tei_isr
7
IIC0 ERI (Transfer error)
smbus_slave_eri_isr
Summary
BSP
Clocks
Pins
Interrupts
Event Links
Stacks
Components

- List of Changes to I3C Driver Codes Generated by FSP (RA6T3)

【NOTE】The symbolic name of the code generated by FSP is replaced in PMBus Middleware for:
Symbol name replacement is not described in detail in this table.
- I2C_SLAVE -> SMBUS_SLAVE
- i2c_slave -> smbus_slave
- IIC_B_SLAVE -> SMBUS_SLAVE
- Iic_b_slave -> smbus_slave

Data Type	i2c_slave_event_t
File Name	r_i2c_slave_api.h
Change Details	-Added EVENT_START_REQUEST. -Added EVENT_STOP_REQUEST. -Added EVENT_ARB_LOST. -Added EVENT_TIMEOUT. -Added EVENT_NACK. -Added EVENT_START_ERR.

Before change

```
typedef enum e_i2c_slave_event

    I2C_SLAVE_EVENT_ABORTED         = 1, ///< A transfer was aborted
    I2C_SLAVE_EVENT_RX_COMPLETE     = 2, ///< A receive operation was completed successfully
    I2C_SLAVE_EVENT_TX_COMPLETE     = 3, ///< A transmit operation was completed successfully
    I2C_SLAVE_EVENT_RX_REQUEST      = 4, ///< A read operation expected from slave. Detected a write from master
    I2C_SLAVE_EVENT_TX_REQUEST      = 5, ///< A write operation expected from slave. Detected a read from master
    I2C_SLAVE_EVENT_RX_MORE_REQUEST = 6, ///< A read operation expected from slave. Master sends out more data than configured to be read in slave.
    I2C_SLAVE_EVENT_TX_MORE_REQUEST = 7, ///< A write operation expected from slave. Master requests more data than configured to be written by slave.
    I2C_SLAVE_EVENT_GENERAL_CALL    = 8, ///< General Call address received from Master. Detected a write from master
} i2c_slave_event_t;
```

After change

```
typedef enum e_smbus_slave_event

    SMBUS_SLAVE_EVENT_ABORTED         = 1, ///< A transfer was aborted
    SMBUS_SLAVE_EVENT_RX_COMPLETE     = 2, ///< A receive operation was completed successfully
    SMBUS_SLAVE_EVENT_TX_COMPLETE     = 3, ///< A transmit operation was completed successfully
    SMBUS_SLAVE_EVENT_RX_REQUEST      = 4, ///< A read operation expected from slave. Detected a write from master
    SMBUS_SLAVE_EVENT_TX_REQUEST      = 5, ///< A write operation expected from slave. Detected a read from master
    SMBUS_SLAVE_EVENT_RX_MORE_REQUEST = 6, ///< A read operation expected from slave. Master sends out more data than configured to be read in slave.
    SMBUS_SLAVE_EVENT_TX_MORE_REQUEST = 7, ///< A write operation expected from slave. Master requests more data than configured to be written by slave.
    SMBUS_SLAVE_EVENT_GENERAL_CALL    = 8, ///< General Call address received from Master. Detected a write from master
    SMBUS_SLAVE_EVENT_START_REQUEST   = 9, ///< Detection of the start condition
    SMBUS_SLAVE_EVENT_STOP_REQUEST    = 10, ///< Detection of the stop condition
    SMBUS_SLAVE_EVENT_ARB_LOST        = 11, ///< Detection of the arbitration lost
    SMBUS_SLAVE_EVENT_TIMEOUT         = 12, ///< Detection of the SCL timeout
    SMBUS_SLAVE_EVENT_NACK            = 13, ///< Detection of the NACK
    SMBUS_SLAVE_EVENT_START_ERR       = 14, ///< Detection of the unexpected start condition >
} smbus_slave_event_t;
```

Data Type	iic_b__slave_instance_ctrl_t
File Name	r_iic_b_slave.h
Change Details	Add volatile bool start_detect.

Before change

```
    volatile bool transaction_completed;          // Tracks whether previous transaction restarted

    /* Pointer to callback and optional working memory */
    void (* p_callback)(i2c_slave_callback_args_t *);
    i2c_slave_callback_args_t * p_callback_memory;

    /* Pointer to context to be passed into callback function */
    void const * p_context;
} iic_b_slave_instance_ctrl_t;
```

After change

```
    volatile bool transaction_completed;          // Tracks whether previous transaction restarted
    volatile bool start_detect;                   // Tracks whether previous transaction restarted

    /* Pointer to callback and optional working memory */
    void (* p_callback)(smbus_slave_callback_args_t *);
    smbus_slave_callback_args_t * p_callback_memory;

    /* Pointer to context to be passed into callback function */
    void const * p_context;
} smbus_slave_instance_ctrl_t;
```

Macro Name	IIC_B_SLAVE_PRV_BIE_INIT_MASK
File Name	r_iic_b_slave.c
Change Details	Removed R_I3C0_BIE_ALIE_Mask and R_I3C0_BIE_TODIE_Msk.

Before change	After change

```
#define IIC_B_SLAVE_PRV_BIE_INIT_MASK          (R_I3C0_BIE_NACKDIE_Msk | R_I3C0_BIE_ALIE_Msk
 | R_I3C0_BIE_TODIE_Msk)
```

```
#define SMBUS_SLAVE_PRV_BIE_INIT_MASK          (R_I3C0_BIE_NACKDIE_Msk)
```

Macro Name	IIC_B _SLAVE_PRV_BIE_INIT_MASK
File Name	r_iic_b_slave.c
Change Details	Removed R_I3C0_BSTE_ALE_Mask and R_I3C0_BSTE_TODE_Msk.

Before change	After change

```
#define IIC_B_SLAVE_PRV_BSTE_INIT_MASK          (R_I3C0_BSTE_NACKDE_Msk | R_I3C0_BSTE_ALE_Msk
 | R_I3C0_BSTE_TODE_Msk | ¥
                                        R_I3C0_BSTE_TENDE_Msk)
```

```
#define SMBUS_SLAVE_PRV_BSTE_INIT_MASK          (R_I3C0_BSTE_NACKDE_Msk |R_I3C0_BSTE_TENDE_Msk)
```

Table Name	g_iic_b_slave0_extend
File Name	r_iic_b_slave.c
Change Details	Move global tables generated in hal.data.c.

Before change	After change
None	

```
const smbus_slave_extended_cfg_t g_smbus_slave0_extend =
{
/* Actual delay: 250 ns. */.clock_settings.brl_value = 25,
    .clock_settings.digital_filter_stages = 0, .clock_settings.cks_value = 0, };
```

Table Name	g_iic_b_slave0_cfg
File Name	r_iic_b_slave.c
Change Details	Move global tables generated in hal.data.c.

Before change	After change
None	

```
const smbus_slave_cfg_t g_smbus_slave0_cfg =
{ .channel = 0, .rate = SMBUS_SLAVE_RATE_STANDARD, .slave = 0x5A, .general_call_enable =
false,
    .addr_mode = SMBUS_SLAVE_ADDR_MODE_7BIT,
    .p_callback = r_smbus_handle_callback, .p_context = NULL,
    .rxi_irq                = VECTOR_NUMBER_IICO_RXI,
    .txi_irq                = VECTOR_NUMBER_IICO_TXI,
    .tei_irq                = VECTOR_NUMBER_IICO_TEI,
    .eri_irq                = VECTOR_NUMBER_IICO_ERI,
    .ipl = (4),
    .eri_ipl = (3), .clock_stretching_enable = false, .p_extend = &g_smbus_slave0_extend, };
```

Table Name	g_iic_slave0
File Name	r_iic_b_slave.c
Change Details	Move global tables generated in hal.data.c.

Before change	After change
None	

```
smbus_slave_instance_t g_smbus_slave0 =
{   .p_ctrl = &g_smbus_slave0_ctrl, .p_cfg = &g_smbus_slave0_cfg, .p_api =
&g_smbus_slave_on_smbus };
```

Function Name	R_IICB_B_SLAVE_Open
File Name	r_iic_b_slave.c
Change Details	p_ctrl->Added start_detect initialization.

Before change

```
198     p_ctrl->notify_request     = false;
199     p_ctrl->transaction_count = 0U;
200
201     return FSP_SUCCESS;
202 }
```

After change

```
227     p_ctrl->transaction_count = 0U;
228     p_ctrl->start_detect      = false;
229
230     return FSP_SUCCESS;
231 }
```

Function Name	iic_b_slave_read_write
File Name	r_iic_b_slave.c
Change Details	-Bytes is cast to uint16_t when internally assigned. -p_ctrl->Removed direction checking. -p_ctrl-> Added a process to initialize the p_ctrl-> start_detect when direction is "MASTER_WRITE_SLAVE_READ".

Before change

```
380     FSP_ERROR_RETURN(IIC_B_SLAVE_OPEN == p_ctrl->open, FSP_ERR_NOT_OPEN);
381
382     /* Fail if there is already a transfer in progress */
383     FSP_ERROR_RETURN(IIC_B_SLAVE_TRANSFER_DIR_NOT_ESTABLISHED == p_ctrl->direction,
FSP_ERR_IN_USE);
384
385     FSP_ASSERT(((iic_b_slave_instance_ctrl_t *) p_api_ctrl)->p_callback != NULL);
```

```
389     p_ctrl->p_buff     = p_buffer;
390     p_ctrl->total      = bytes;
391     p_ctrl->remain     = bytes;
392     p_ctrl->direction = direction;
```

```
403     p_ctrl->transaction_completed = false;
404
405     return FSP_SUCCESS;
406 }
```

After change

```
407     FSP_ERROR_RETURN(SMBUS_SLAVE_OPEN == p_ctrl->open, FSP_ERR_NOT_OPEN);
408
409     FSP_ASSERT(((smbus_slave_instance_ctrl_t *) p_api_ctrl)->p_callback != NULL);
```

```
413     p_ctrl->p_buff     = p_buffer;
414
415     /* In order to reuse the base driver code, cast uint16_t from uint32_t. */
416     p_ctrl->total      = (uint16_t)bytes;
417
418     /* In order to reuse the base driver code, cast uint16_t from uint32_t. */
419     p_ctrl->remain     = (uint16_t)bytes;
420     p_ctrl->direction = direction;
```

```
433     /* The flag is cleared in preparation for detecting a start condition for transmission
processing */
434     /* while waiting for reception.
           */
435     if (SMBUS_SLAVE_TRANSFER_DIR_MASTER_WRITE_SLAVE_READ == direction)
436     {
437         p_ctrl->start_detect = false;
438     }
439
440     return FSP_SUCCESS;
```

Function Name	iic_b_slave_notify
File Name	r_iic_b_slave.c
Change Details	-The setting to set STCNDDE bit and SPCNDDE bit to 1 was added to BSTE setting. -Change the type of the local transaction_count to uint16_t.

Before change

```
421     p_ctrl->p_reg->BSTE = IIC_B_SLAVE_PRV_BSTE_INIT_MASK & ~R_I3C0_BSTE_TODE_Msk;
422     p_ctrl->p_reg->NTIE = IIC_B_SLAVE_PRV_NTIE_INIT_MASK;
423
424     /* Reset the status flags */
```

```
449     /* Save transaction count */
450     uint32_t transaction_count = p_ctrl->transaction_count;
451
452     /* Reset the transaction count here */
```

After change

```
456     p_ctrl->p_reg->BSTE = (SMBUS_SLAVE_PRV_BSTE_INIT_MASK | R_I3C0_BSTE_STCNDDE_Msk |
R_I3C0_BSTE_SPCNDDE_Msk) &
457                           (~R_I3C0_BSTE_TODE_Msk);
458     p_ctrl->p_reg->NTIE = SMBUS_SLAVE_PRV_NTIE_INIT_MASK;
459
460     /* Reset the status flags */
```

```
485     /* Save transaction count */
486     uint16_t transaction_count = p_ctrl->transaction_count;
487
488     /* Reset the transaction count here */
```

Function Name	iic_b_slave_callback_request
File Name	r_iic_b_slave.c
Change Details	-Removed the setting of BSTE.TODE bit and BIE.TODIE bit before and after iic_b_slave_call_callback().

Before change

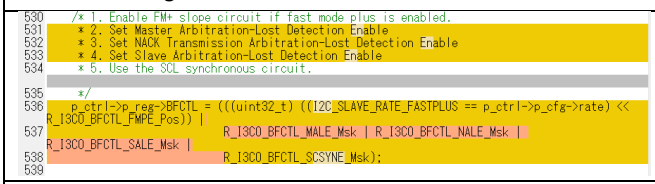
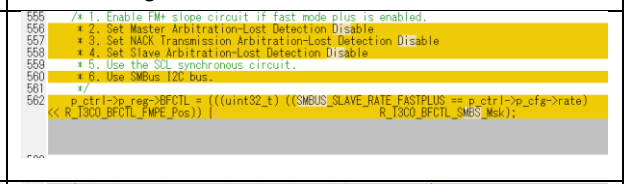
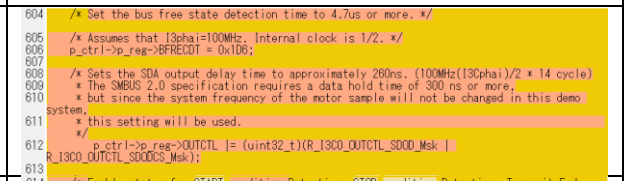
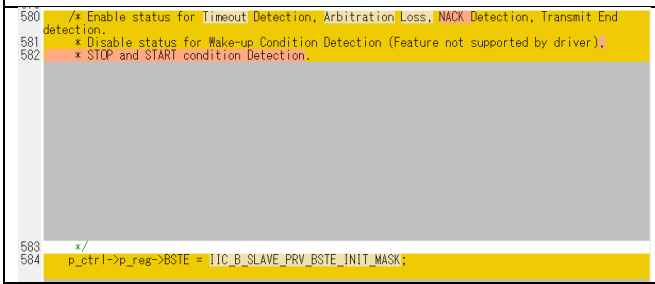
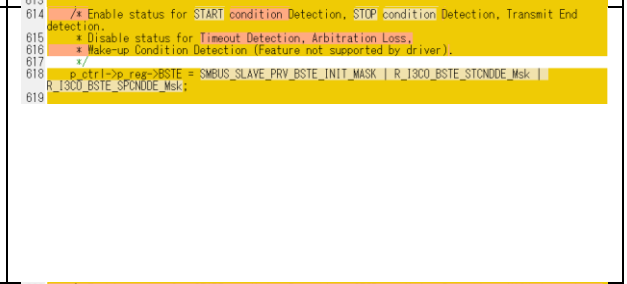
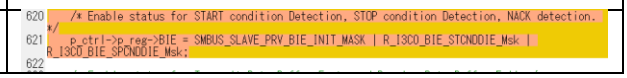
```
469     p_ctrl->direction = IIC_B_SLAVE_TRANSFER_DIR_NOT_ESTABLISHED;
470
471     /* Disable timeout function */
472     p_ctrl->p_reg->BSTE_b.TODE = 0U;
473     p_ctrl->p_reg->BIE_b.TODIE = 0U;
474
475     /* Invoke the callback to notify the read request.
```

```
479     /* Allow timeouts to be generated on the low value of SCL using long count mode */
480     p_ctrl->p_reg->TMOCTL = R_IIC0_TMOCTL_TOLCTL_Msk;
481
482     /* Enable timeout function */
483     p_ctrl->p_reg->BSTE_b.TODE = 1U;
484     p_ctrl->p_reg->BIE_b.TODIE = 1U;
```

After change

```
505     /* Invoke the callback to notify the read request.
```

Function Name	iic_b_open_hw_slave
File Name	r_iic_b_slave.c
Change Details	-Added the setting of BFCTL.SMBS=1 (select SMBus). -Change the setting to BFCTL.SALE=0 (slave arbitration-lost detection is disabled). -Change the setting to BFCTL.NALE=0 (disable NACK transmit arbitration-lost detection). -OUTCTL.SDOD[2:0] =111b(13 or 14 I3Cφ cycles (I3Cφ/2 when OUTCTL.SDODCS=1)), OUTCTL.SDODCS=1 (internal reference clock divided by 2 is selected as the clock source for SDA output delay counter.) -TMOCTL.TOLCTL=0 (SCL L-period timeout detection disabled), TMOCTL.TOHCTL=0 (SCL H-period timeout detection disabled) -Set BIE.NACKDIE to 1 (NACK detect interrupt enabled). -Set BIE.STCNDIE=1 (Start condition detected interrupt is enabled). -Set BIE.NACKDIE to 1 (Stop condition detection interrupt enabled). -Addition of BFRECDT.FRECYC[8:0] (bus-free interval) setting. (1/I3Cφ(100MHz)*4.7μs=470(0x1D6hex)

Before change	After change
 530 /* 1. Enable FM+ slope circuit if fast mode plus is enabled. 531 * 2. Set Master Arbitration-Lost Detection Enable 532 * 3. Set NACK Transmission Arbitration-Lost Detection Enable 533 * 4. Set Slave Arbitration-Lost Detection Enable 534 * 5. Use the SCL synchronous circuit. 535 */ 536 p_ctrl->p_reg->BFCTL = (((uint32_t) ((I2C_SLAVE_RATE_FASTPLUS == p_ctrl->p_cfg->rate) << R_I3C0_BFCTL_FMPE_Pos)) | 537 R_I3C0_BFCTL_MALE_Msk | R_I3C0_BFCTL_NALE_Msk | R_I3C0_BFCTL_SALE_Msk | 538 R_I3C0_BFCTL_SCSYNE_Msk); 539	 555 /* 1. Enable FM+ slope circuit if fast mode plus is enabled. 556 * 2. Set Master Arbitration-Lost Detection Disable 557 * 3. Set NACK Transmission Arbitration-Lost Detection Disable 558 * 4. Set Slave Arbitration-Lost Detection Disable 559 * 5. Use the SCL synchronous circuit. 560 * 6. Use SMBus I2C bus. 561 */ 562 p_ctrl->p_reg->BFCTL = (((uint32_t) ((SMBUS_SLAVE_RATE_FASTPLUS == p_ctrl->p_cfg->rate) << R_I3C0_BFCTL_FMPE_Pos)) | R_I3C0_BFCTL_SMBS_Msk);
None.	 604 /* Set the bus free state detection time to 4.7us or more. */ 605 /* Assumes that I3Cphai=100MHz. Internal clock is 1/2. */ 606 p_ctrl->p_reg->BFRECDT = 0x1D6; 607 608 /* Sets the SDA output delay time to approximately 260ns. (100MHz(I3Cphai)/2 * 14 cycle) 609 * The SMBUS 2.0 specification requires a data hold time of 300 ns or more, 610 * but since the system frequency of the motor sample will not be changed in this demo system, 611 * this setting will be used. 612 */ p_ctrl->p_reg->OUTCTL |= (uint32_t)(R_I3C0_OUTCTL_SDOD_Msk | R_I3C0_OUTCTL_SDODCS_Msk); 613
 580 /* Enable status for Timeout Detection, Arbitration Loss, NACK Detection, Transmit End detection. 581 * Disable status for Wake-up Condition Detection (Feature not supported by driver). 582 * STOP and START condition Detection. 583 */ 584 p_ctrl->p_reg->BSTE = IIC_B_SLAVE_PRV_BSTE_INIT_MASK;	 614 /* Enable status for START condition Detection, STOP condition Detection, Transmit End detection. 615 * Disable status for Timeout Detection, Arbitration Loss, 616 * Wake-up Condition Detection (Feature not supported by driver). 617 */ 618 p_ctrl->p_reg->BSTE = SMBUS_SLAVE_PRV_BSTE_INIT_MASK | R_I3C0_BSTE_STCNDDE_Msk | R_I3C0_BSTE_SPCNDDE_Msk; 619
None.	 620 /* Enable status for START condition Detection, STOP condition Detection, NACK detection. */ 621 p_ctrl->p_reg->BIE = SMBUS_SLAVE_PRV_BIE_INIT_MASK | R_I3C0_BIE_STCNDDIE_Msk | R_I3C0_BIE_SPCNDDIE_Msk; 622

Function Name	iic_b_slave_initiate_transaction
File Name	r_iic_b_slave.c
Change Details	This function is deleted because the calling function is replaced with iic_b_slave_callback_request.

Before change	After change
	Delete

```
 *
 * @param[in]     p_ctrl          Pointer to the control structure.
 * @param[in]  slave_event        Slave event to be reported via callback.
 **********************************************************************************************************************/
static void iic_b_slave_initiate_transaction (iic_b_slave_instance_ctrl_t * p_ctrl, i2c_slave_event_t slave_event)
{
    /* Set the status flag to ensure this conditional clause execution only once */
    p_ctrl->notify_request = true;

    /* Enable Interrupts: Timeout Detection, Arbitration Loss, NACK Detection.
     * Disable Interrupt: Transmit End, Start, Stop
     */
    p_ctrl->p_reg->BIE = IIC_B_SLAVE_PRV_BIE_INIT_MASK;

    /* Enable Interrupts: Transmit Data Buffer Empty and Receive Data Buffer Full */
    p_ctrl->p_reg->NTIE = IIC_B_SLAVE_PRV_NTIE_INIT_MASK;

    /* Invoke callback for the user to call a valid API. */
    iic_b_slave_callback_request(p_ctrl, slave_event);

    /* Check if correct API is called here. Check direction (API called) against slave event requested (ISR invoked) */
    if (((((IIC_B_SLAVE_TRANSFER_DIR_MASTER_WRITE_SLAVE_READ ==
            p_ctrl->direction) &&
           ((I2C_SLAVE_EVENT_RX_REQUEST == slave_event) || (I2C_SLAVE_EVENT_GENERAL_CALL == slave_event))) ||
          ((IIC_B_SLAVE_TRANSFER_DIR_MASTER_READ_SLAVE_WRITE ==
            p_ctrl->direction) && (I2C_SLAVE_EVENT_TX_REQUEST == slave_event))))
    {
        /* In case MasterWriteSlaveRead API is NOT called to service Master write operation a NACK is
         * issued from the RXI ISR (which is fired once) and the bus is released.
         *
         * In case MasterWriteSlaveRead API is NOT called to service Master General Call operation a NACK is
         * issued from the RXI ISR (which is fired once) and the bus is released.
         *
         * In case MasterReadSlaveWrite API is NOT called to service Master read operation the TXI will fire once,
         * no data will be written to ICDRT and the master will read 0xFF for every byte it tries to read.
         *
         * For both the cases above the slave callback is invoked with I2C_SLAVE_EVENT_ABORTED
         * event to notify the user application.
         *
         */
    }
    else
    {
        /* Enable the Start condition detection to trigger ERI ISR */

        /* Since address match is detected, enable STOP and RESTART detection for Master Read Slave Write.
         * This must be done conditionally only for Master Read Slave Write to prevent clearing the start bit
         * in case a restart occurred (and got captured) while we were in the user callback.
         * This capturing is made possible in the 'iic_b_rxi_slave' after the dummy read.
         * In case of Master Write Slave Read for 10-bit, this is done here as the address issues a
         * restart which we don't want to detect.
         */
        if ((IIC_B_SLAVE_TRANSFER_DIR_MASTER_READ_SLAVE_WRITE == p_ctrl->direction) ||
            ((IIC_B_SLAVE_TRANSFER_DIR_MASTER_WRITE_SLAVE_READ == p_ctrl->direction) &&
             (p_ctrl->p_cfg->addr_mode == I2C_SLAVE_ADDR_MODE_10BIT)))
        {
            /* Clear the Start and Stop condition flag for Slave Read/Write operation */
            p_ctrl->p_reg->BST &= ((uint32_t) ~(R_IIC0_BST_STCNDDF_Msk | R_IIC0_BST_SPCNDDF_Msk));
        }

        /* Enable the Start and Stop condition detection interrupt */

        /* Enable Interrupts: Timeout Detection, Arbitration Loss, NACK, Start, Stop Detection.
         * Disable Interrupt: Transmit End
         */
        p_ctrl->p_reg->BSTE = IIC_B_SLAVE_PRV_BSTE_INIT_MASK | R_IIC0_BIE_STCNDDIE_Msk | R_IIC0_BIE_SPCNDDIE_Msk;
        p_ctrl->p_reg->BIE  = IIC_B_SLAVE_PRV_BIE_INIT_MASK | R_IIC0_BIE_STCNDDIE_Msk | R_IIC0_BIE_SPCNDDIE_Msk;

        /* Enable Interrupts: Transmit Data Buffer Empty and Receive Data Buffer Full */
        p_ctrl->p_reg->NTIE = IIC_B_SLAVE_PRV_NTIE_INIT_MASK;
    }
}

/***********************************************************************************************************************/
 * Calls user callback
```

Function Name	iic_b_rxi_slave
File Name	r_iic_b_slave.c
Change Details	-Callback calls changed to use iic_b_slave_callback_request(). -Add a r_smbus_rxi_check_illegal_start to check whether a Start condition is detected, and Execute the following: -When STCNDDF bit of BST is "1", STCNDDF bit of BST is cleared after NTDTBP0 register is dummy read, and iic_b_slave_callback_request() is executed by specifying EVENT_START_ERR. -- If p_ctrl->direction is "MASTER_READ_SLAVE_WRITE", NTDTBP0 register is dummy-read, and then iic_b_slave_callback_request() is executed with EVENT_RX_REQUEST specified. -- If other than the above, the next process of the reception buffer empty interrupt process is executed. -The handling of unexpected reception interrupts is changed to ACKCTL.ACKT bit-based NACK response. Instead, dummy read of NTDTBP0 register and callback function of iic_b_slave_callback_request() are executed. -When setting BIE register when dummy read p_ctrl->do_dummy_read is 0, STCNDDIE bit is not set, and NACKDIE bit and SPCNDDIE bit are set.

Before change	After change
None.	

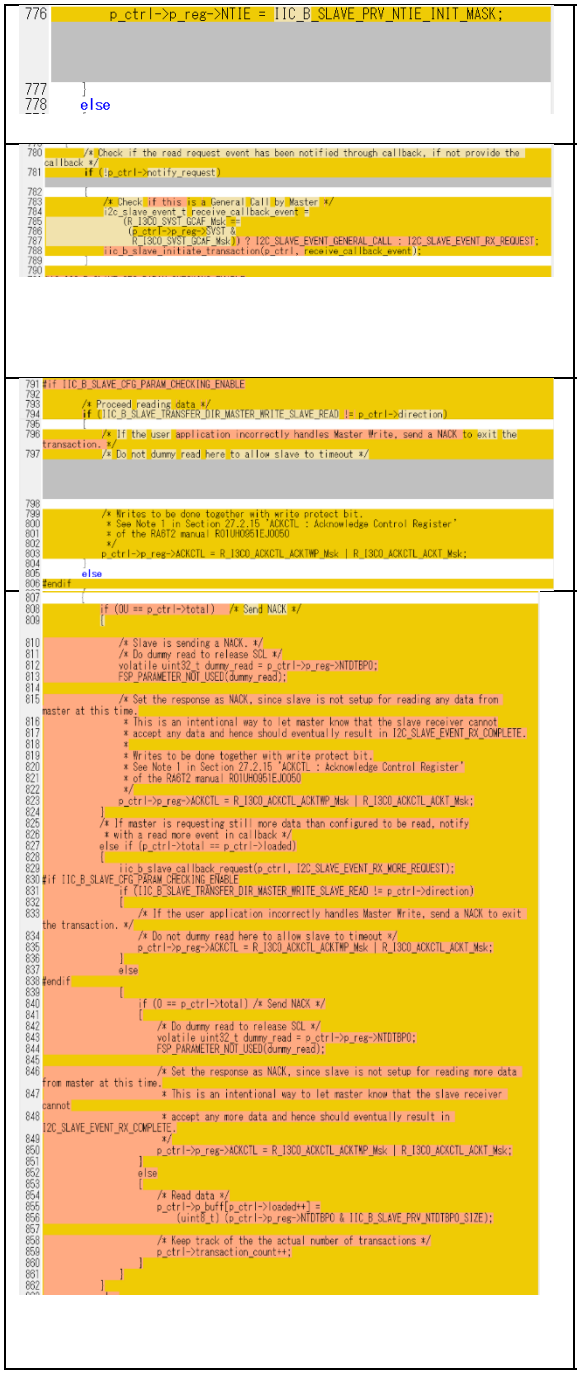
```
 * Check start detection when receive data full interrupt operating as a slave.
 *
 * @param[in]       p_ctrl     The target IIC block's control block.
 **********************************************************************************************************************/
static bool r_smbus_rxi_check_illegal_start(smbus_slave_instance_ctrl_t * p_ctrl)
{
    bool check_result;

    /* If a start condition is detected during reception mode, abnormality processing is performed. */
    if (1 == p_ctrl->p_reg->BST_b.STCNDDF)
    {
        /* dummy read */
        volatile uint8_t dummy_read = (uint8_t) (p_ctrl->p_reg->NTDTBP0 & SMBUS_SLAVE_PRV_NTDTBP0_SIZE);
        FSP_PARAMETER_NOT_USED(dummy_read);

        /* clear start condition detection flag */
        p_ctrl->p_reg->BST &= ((uint32_t) ~(R_I3C0_BST_STCNDDF_Msk));

        r_smbus_slave_callback_request(p_ctrl, SMBUS_SLAVE_EVENT_START_ERR);

        check_result = true;
    }
    /* Perform dummy read after an address match detection. */
    else if (SMBUS_SLAVE_TRANSFER_DIR_MASTER_READ_SLAVE_WRITE == p_ctrl->direction)
    {
        /* dummy read */
        volatile uint8_t dummy_read = (uint8_t) (p_ctrl->p_reg->NTDTBP0 & SMBUS_SLAVE_PRV_NTDTBP0_SIZE);
        FSP_PARAMETER_NOT_USED(dummy_read);

        r_smbus_slave_callback_request(p_ctrl, SMBUS_SLAVE_EVENT_RX_REQUEST);

        check_result = true;
    }
    else
    {
        check_result = false;
    }

    return check_result;
}

/***********************************************************************************************************************//**
```

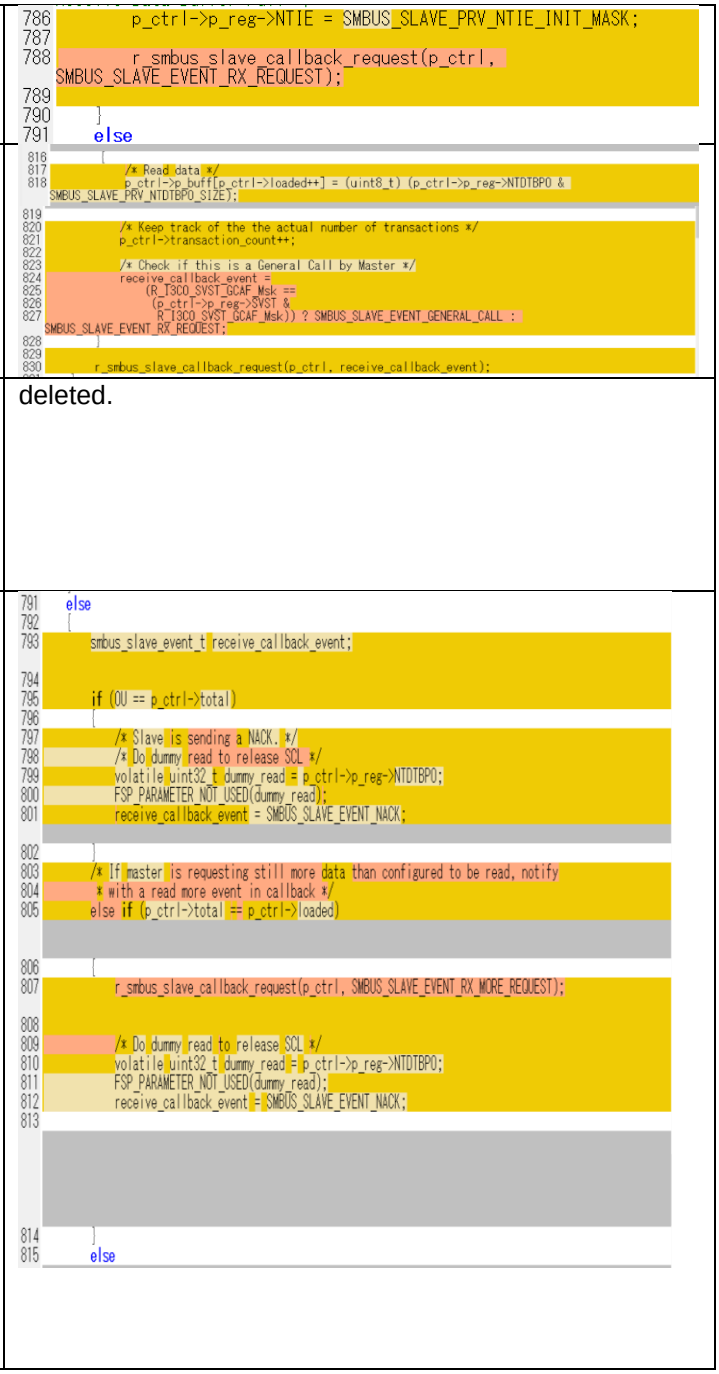
```
static void iic_b_rxi_slave (iic_b_slave_instance_ctrl_t * p_ctrl)
{

    /* Perform dummy read after an address match detection. */
    if (!p_ctrl->do_dummy_read)
```

```
static void r_smbus_rxi_slave (smbus_slave_instance_ctrl_t * p_ctrl)
{

    if (r_smbus_rxi_check_illegal_start(p_ctrl))
    {
        /* If illegal start condition detected, operate in r_smbus_rxi_check_illegal_start */
        ;
    }
    /* Perform dummy read after an address match detection. */
    else if (!p_ctrl->do_dummy_read)
```

```
        * Enable Interrupts: Start, Stop Detection.
        * Disable Interrupt: Transmit End, Timeout Detection, Arbitration Loss, NACK
        */
       p_ctrl->p_reg->BIE = R_I3C0_BIE_STCNDDIE_Msk | R_I3C0_BIE_SPCNDDIE_Msk;
```

```
        * Enable Interrupts: Transmit End, Timeout Detection, Arbitration Loss, NACK, Stop Detection.
        * Disable Interrupt: Start

        */
       p_ctrl->p_reg->BIE = SMBUS_SLAVE_PRV_BIE_INIT_MASK | R_I3C0_BIE_SPCNDDIE_Msk;
```





deleted.

Function Name	iic_b_txi_slave
File Name	r_iic_b_slave.c
Change Details	-Callback calls changed to use iic_b_slave_callback_request(). -The term for calling iic_b_slave_callback_request() is changed when BST.STCNDDF=1 (Start condition detected) or p_ctrl->start_detect = true (start condition detected in receive mode). In this case, the process of initializing p_ctrl->start_detect to false is added.

Before change

```
883     if (!p_ctrl->notify_request)
884     {
885         iic_b_slave_initiate_transaction(p_ctrl, I2C_SLAVE_EVENT_TX_REQUEST);
886     }
```

After change

```
842     if ((1 == p_ctrl->p_reg->BST_b.STCNDDF) ||
843         (true == p_ctrl->start_detect))
844     {
845         p_ctrl->p_reg->BST_b.STCNDDF = 0;
846         p_ctrl->start_detect = false;
847         r_smbus_slave_callback_request(p_ctrl, SMBUS_SLAVE_EVENT_TX_REQUEST);
848     }
849
850     /* If MasterReadSlaveWrite API is invoked, proceed writing data */
```

Function Name	iic_b_tei_slave
File Name	r_iic_b_slave.c
Change Details	The data to be written to NTDTBP0 when ACKCTL.ACKR=1 (ACK detected) is changed to dummy data.

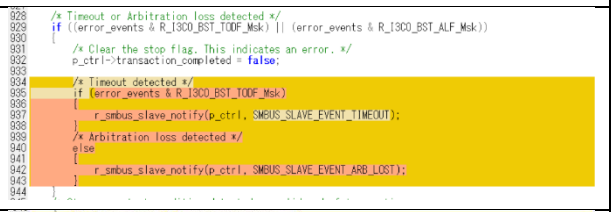
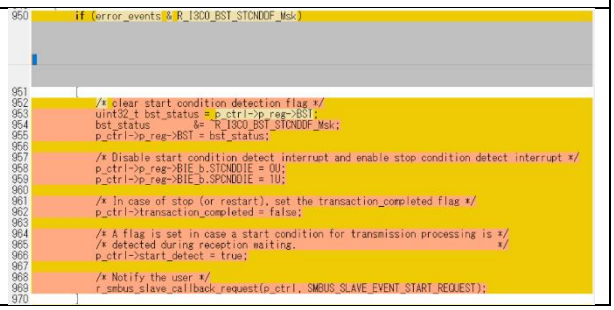
Before change

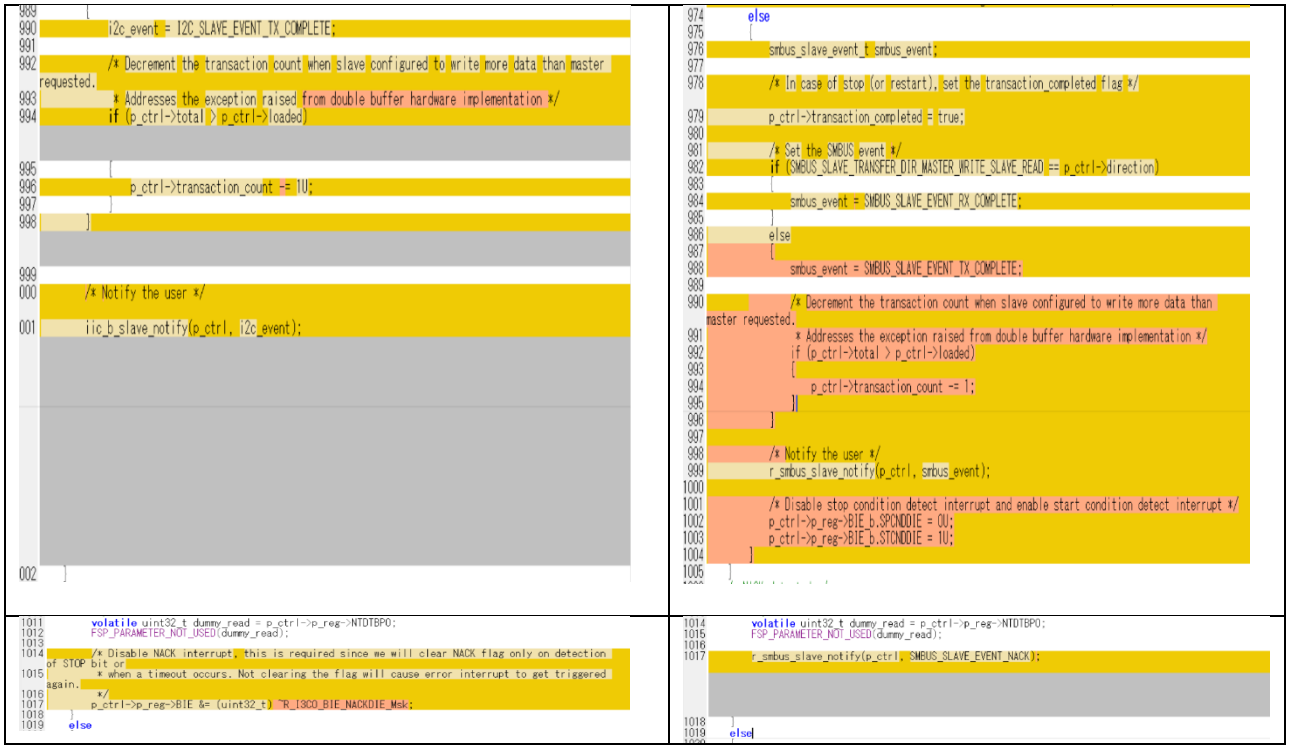
```
940         else
941         {
942             p_ctrl->p_reg->NTDTBP0 = (uint32_t) p_ctrl->p_buff[p_ctrl->loaded];
943             p_ctrl->loaded++;
944             p_ctrl->transaction_count++;
945         }
```

After change

```
902         else
903         {
904             volatile uint8_t dummy_data = 0xFF;
905
906             /* Write the dummy data, this will also release SCL */
907             p_ctrl->p_reg->NTDTBP0 = (uint32_t) dummy_data;
908         }
```

Function Name	iic_b_err_slave
File Name	r_iic_b_slave.c
Change Details	-Change the branching of error events so that BST.TODF=1 (time-out detection), BST.ALFDF=1 (arbitration-lost detection), BST.STCNDDF=1 (Start condition detection), BST.SPCDDF=1 (Stop condition detection), and BST.NACKDF=1 (NACK detection). -When BST.TODF=1, the event set to iic_b_slave_notify() is changed to "EVENT_TIMEOUT". -When BST.ALFDF=1, the event set to iic_b_slave_notify() is changed to "EVENT_ARB_LOST". -When BST.STCNDDF=1, set BIE.STCNDDIE=0 (Start condition detection interrupt is disabled) and BIE.SPCNDDIE=1 (Stop condition detection interrupt is enabled), and then change to notify "EVENT_START_REQUEST" by iic_b_slave_callback_request. -When BST.SPCNDDF=1, set BIE.SPCNDDIE=0 (Stop condition detection interrupt is disabled), BIE.STCNDDIE=1 (Start condition detection interrupt is enabled), set true to p_ctrl->start_detect, and change to notify "EVENT_RX_COMPLETE" or "EVENT_TX_COMPLETE" in iic_b_slave_callback_request. -When BST.NACKDF=1, the setting of BIE.NACKDIE=0 (NACK detected interrupt disabled) is deleted, Change iic_b_slave_notify() to set the event "EVENT_NACK".

Before change	After change
	
None.	



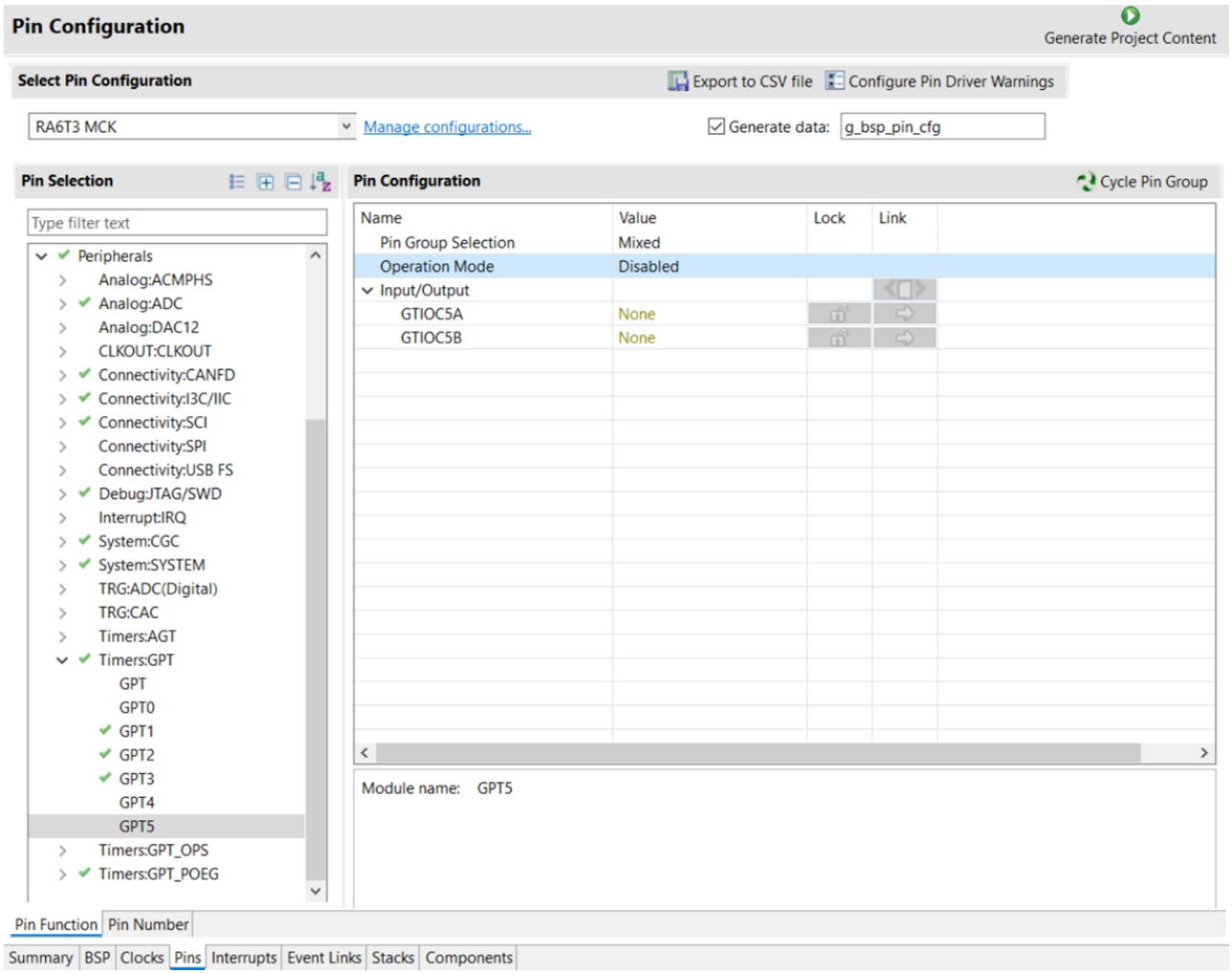
```
            i2c_event = I2C_SLAVE_EVENT_TX_COMPLETE;

            /* Decrement the transaction count when slave configured to write more data than master requested.
             * Addresses the exception raised from double buffer hardware implementation */
            if (p_ctrl->total > p_ctrl->loaded)
            {
                p_ctrl->transaction_count -= 1U;
            }
        }

        /* Notify the user */
        iic_b_slave_notify(p_ctrl, i2c_event);
    }
```

```
        else
        {
            smbus_slave_event_t smbus_event;

            /* In case of stop (or restart), set the transaction_completed flag */
            p_ctrl->transaction_completed = true;

            /* Set the SMBUS event */
            if (SMBUS_SLAVE_TRANSFER_DIR_MASTER_WRITE_SLAVE_READ == p_ctrl->direction)
            {
                smbus_event = SMBUS_SLAVE_EVENT_RX_COMPLETE;
            }
            else
            {
                smbus_event = SMBUS_SLAVE_EVENT_TX_COMPLETE;

                /* Decrement the transaction count when slave configured to write more data than master requested.
                 * Addresses the exception raised from double buffer hardware implementation */
                if (p_ctrl->total > p_ctrl->loaded)
                {
                    p_ctrl->transaction_count -= 1;
                }
            }

            /* Notify the user */
            r_smbus_slave_notify(p_ctrl, smbus_event);

            /* Disable stop condition detect interrupt and enable start condition detect interrupt */
            p_ctrl->p_reg->BIE_b.SPONDIE = 0U;
            p_ctrl->p_reg->BIE_b.STONDIE = 1U;
        }
    }
```

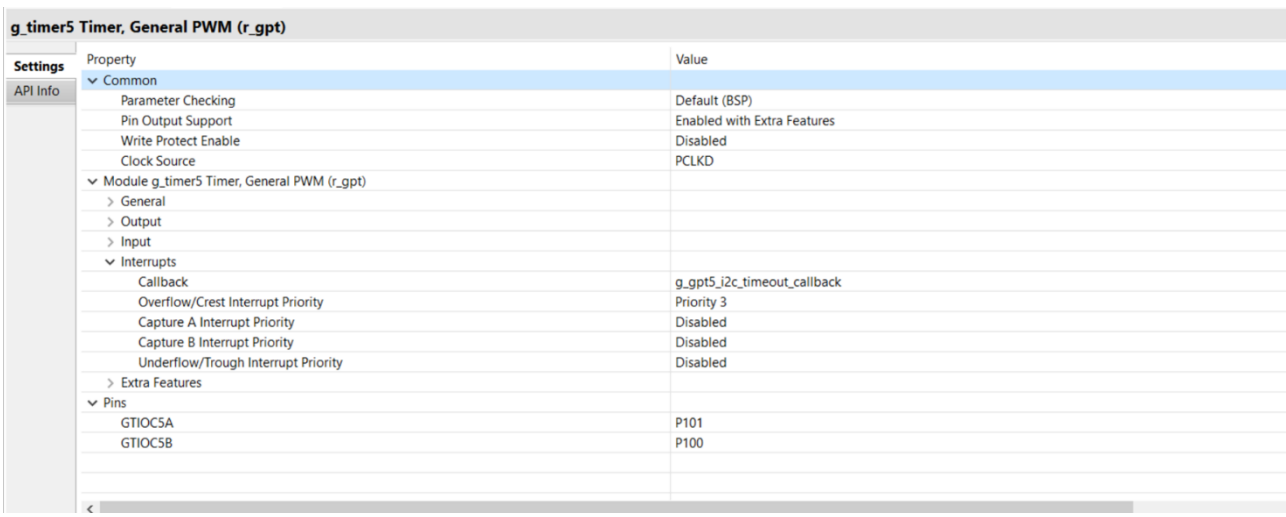
```
        volatile uint32_t dummy_read = p_ctrl->p_reg->NTDTBP0;
        FSP_PARAMETER_NOT_USED(dummy_read);

        /* Disable NACK interrupt, this is required since we will clear NACK flag only on detection of STOP bit or
         * when a timeout occurs. Not clearing the flag will cause error interrupt to get triggered again.
         */
        p_ctrl->p_reg->BIE &= (uint32_t) ~R_I3C0_BIE_NACKDIE_Msk;
    }
    else
```

```
        volatile uint32_t dummy_read = p_ctrl->p_reg->NTDTBP0;
        FSP_PARAMETER_NOT_USED(dummy_read);

        r_smbus_slave_notify(p_ctrl, SMBUS_SLAVE_EVENT_NACK);
    }
    else
```

- Setting GPT of FSP (RA6T3)



-



- List of Changes to GPT Driver Codes Generated by FSP (RA6T3)

Function Name	p_callback	
File Name	r_pmbus_wrapper_slave.c	
Change Details	A callback function executed by an overflow interrupt, which is registered as a parameter during R_GPT_Open. The actual state of this function is placed in the r_pmbus_wrapper_slave.c in PMBus Middleware. Execute r_pmbus_nwk_slave_process_Timeout() to refresh PMBus Middleware status.	
Before change		After change
None.		``` /********** * Function Name: g_gpt5_i2c_timeout_callback * Description : . * Argument : p_args * Return Value : . **********/ void g_gpt5_i2c_timeout_callback(timer_callback_args_t *p_args) { if (NULL != p_args) { switch (p_args->event) { case TIMER_EVENT_CYCLE_END: r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_TIMEOUT); break; default: break; } } } /********** * End of function g_gpt5_i2c_timeout_callback ```

- Comparing RX26T and RA6T3 Drivers API

Peripheral functions	API name of RX26T RIIC0	API name of RA6T3 I3C	Function overview
I2C (I3C)	void R_Config_RIIC0_Create(void)	fsp_err_t R_IIC_B_SLAVE_Open (i2c_slave_ctrl_t * const p_api_ctrl, i2c_slave_cfg_t const * const p_cfg)	Initialization/communication start.
	void R_Config_RIIC0_Create_UserInit(void)		
	void R_Config_RIIC0_Start(void)		
	MD_STATUS R_User_RIIC0_Slave_Receive (uit8_t * const rx_buf, uint16_t rx_num)*1	fsp_err_t R_IIC_B_SLAVE_Read (i2c_slave_ctrl_t * const p_api_ctrl, uint8_t * const p_dest, uint32_t const bytes)	Start reception.
	MD_STATUS R_Conig_RIIC0_Slave_Send (uint8_t * const txbuf, uint16_t tx_num)	fsp_err_t R_IIC_B_SLAVE_Write (i2c_slave_ctrl_t * const p_api_ctrl, uint8_t * const p_src, uint32_t const bytes)	Start transmission.
	void R_Config_RIIC0_Stop(void)	fsp_err_t R_IIC_B_SLAVE_Close (i2c_slave_ctrl_t * const p_api_ctrl)	Stop communication.
	void R_Config_RIIC0_IIC_StartCondition(void)	-	Start condition issuance.
	void R_Config_RIIC0_IIC_StopCondition(void)	-	Stop condition issuance.
	-	R_IIC_B_SLAVE_CallbackSet (i2c_slave_ctrl_t * const p_api_ctrl, void (* p_callback)(i2c_slave_callback_args_t *), void const * const p_context, i2c_slave_callback_args_t * const p_callback_memory)	Modify a callback function.
Timer	void R_Config_TMR0_Create(void)	fsp_err_t R_GPT_Open (timer_ctrl_t * const p_ctrl, timer_cfg_t const * const p_cfg)	Initialization.
	void R_Config_TMR0_Create_UserInit(void)		
	void R_User_TMR0_Start(void) *1	fsp_err_t R_GPT_Start (timer_ctrl_t * const p_ctrl)	Timer start.
		fsp_err_t R_GPT_Reset (timer_ctrl_t * const p_ctrl)	Counter reset.
	void R_Config_TMR0_Stop(void)	fsp_err_t R_GPT_Stop (timer_ctrl_t * const p_ctrl)	Timer stop.
		fsp_err_t R_GPT_Close (timer_ctrl_t * const p_ctrl)	Timer End.

5.2.5 PMBus Slave Data Types and Structure List

The following table lists the Data Types and Structure used in this control program.

- Data Types and Structure for PMBus Slave Application Functions

e_app_req_event_t

Enumeration Name	e_app_req_event_t	
Description	This enumeration type is used to notify the main process of the cause of a PMBUS callback or the occurrence of other events.	
Declared header file	r_app_main.h	
Remarks	-	
Element name	**Description**	**Value**
E_PMBUS_REQ_EVENT_NONE	No event occurred.	0
E_PMBUS_REQ_EVENT_ERROR	This is used when I2C error interrupt occurs and the callback will notify the main process of the error. Depending on the needs of the user system, the main process should Execute post-error process such as calling R_PMBUS_Slave_Restart, or R_PMBUS_Slave_Close/R_PMBUS_Slave_Open.	1
E_PMBUS_REQ_EVENT_ARA_START	This event is an optional event for checking the ALERT RESPONSE response operation. If you want to check the operation of the ALERT RESPONSE process by detecting an ALERT signal, etc., you can use it by changing the PMBus Slave Application. When this event is detected, execute R_PMBUS_Slave_SendARA in the main process.	2
E_PMBUS_REQ_EVENT_ARA_STOP	This event is an optional event to check the return from ALERT RESPONSE response operation to normal operation. If you want to return from ALERT RESPONSE process operation to normal operation, the user should change the PMBus Slave Application to use this event. When this event is detected, execute R_PMBUS_Slave_Restart in the main process.	3
E_PMBUS_REQ_EVENT_PEC_ENA	This event is an optional event for checking communication with PEC. If you want to communicate with PEC without resetting the slave during slave operation, you should change the PMBus slave application and use it. When this event is detected, execute R_PMBUS_Slave_PEC_Enable in the main process.	4
E_PMBUS_REQ_EVENT_PEC_DIS	This event is an optional event for returning to normal operation from an operating state of communication with PEC. If you want to return to normal communication from communication with PEC without resetting the slave during the slave's operation, the user should change the PMBus slave application and use this event. When this event is detected, execute R_PMBUS_Slave_PEC_Disavle in the main process.	5

- Data Types and Structure for PMBus Slave API

e_pmbus_packet_result_s_t

Enumeration Name	e_pmbus_packet_result_s_t	
Description	This enumeration type is used to indicate the execution result of PMBus communication (slave). It is used as the argument type for each PMBus Slave API. It indicates the details of the error cause when the PMBus API return value is PMBUS_RET_ERROR.	
Declared header file	r_pmbus_app_slave.h	
Remarks	-	
Element name	**Description**	**Value**
E_PMBUS_PACKET_S_OK	Normal operation.	0
E_PMBUS_PACKET_S_DATA_SIZE_ERROR	A packet size error was detected.	1
E_PMBUS_PACKET_S PEC_ERROR	An error was detected in PEC operation.	2
E_PMBUS_PACKET_S_TIMEOUT	A timeout error was detected. ($T_{TIMEOUT}$ error detected)	3
E_PMBUS_PACKET_S_CMD_WAIT	Command code and received data are being received.	4
E_PMBUS_PACKET_S_CMD_COMP	The command code and received data have been received.	5
E_PMBUS_PACKET_S_CMD_QUICK	Quick Command is being taken.	6
E_PMBUS_PACKET_S_INTERNAL_ERROR	An internal error was detected.	7
E_PMBUS_PACKET_S_NOT_READY	Command reception is not ready.	8
E_PMBUS_PACKET_S_CMD_NOT_SUPPORT	A command not supported by PMBUS spec is received.	9
E_PMBUS_PACKET_S_ARA_COMP	SendARA process completed.	10

p_pmbusSlaveCmdCallback

Callback Function Type Name	void (* p_pmbusSlaveCmdCallback)(st_pmbus_nwk_ctrl_s_t *p_st_nwk_ctrl, e_pmbus_packet_result_s_t e_pmbus_result)
Description	Callback calls from PMBus interrupt handlers. It is specified when you R_PMBUS_Slave_Open(). The user must use this callback function to implement the execution process corresponding to the command code.
Declared header file	r_pmbus_app_slave.h
Remarks	-
Element name	**Description**
st_pmbud_nwk_ctrl_s_t *p_st_pmbus_nwk_ctrl	Pointer to the storage of PMBus Middleware's network-control information.
e_pmbus_packet_result_s e_pmbus_result	Pointer to store PMBus Middleware packet-information.

st_pmbus_cfg_s_t

Structure Name	r_pmbus_cfg_s_t
Description	This is a structure for PMBus Slave configuration data. It is used as the argument type for R_PMBUS_Slave_Open. It is used to register the configuration data to the internal global variables of the PMBus Slave Middleware.
Declared header file	r_pmbus_app_slave.h
Remarks	-
Member name	**Description**
uint16_t u2_rx_size	*The dimensions of the p_u1_rx_buf. (1～35)
uint8_t *p_u1_rx_buf	Pointer to the receive data storage buffer. The data received from master is stored in this buffer.
uint16_t u2_tx_size	*The dimensions of the p_u1_tx_buf. (1～35)
uint8_t *p_u1_tx_buf	Pointer to the transmit data storage buffer. Stores the data to be sent to master.
uint8_t u1_slave_addr	Own slave address. (0x01～0x0F)
void (* p_pmbusSlaveCmdCallback)(st_pmbus_nwk_ctrl_s_t * const p_st_nwk_ctrl, e_pmbus_packet_result_s_t * const p_e_pmbus_result)	Callback calls from PMBus interrupt handlers. The user must execute the control process of motor module corresponding to the command in this function.

- Data Types and Structure for PMBus Slave Middleware functions

e_pmbus_nwk_status_s_t

Enumeration Name	e_pmbus_nwk_status_s_t	
Description	This enumeration type indicates the internal state of the PMBus network layer (slave). It is used to manage the internal state of the PMBus Slave Middleware.	
Declared header file	r_pmbus_app_slave.h	
Remarks	-	
Element name	**Description**	**Value**
E_PMBUS_NWK_STATUS_S_IDLE	Waiting for new packet reception.	0
E_PMBUS_NWK_STATUS_S_START	Detects start condition.	1
E_PMBUS_NWK_STATUS_S_RX	Receiving packet.	2
E_PMBUS_NWK_STATUS_S_TX	Transmitting Receive Byte packet.	3
E_PMBUS_NWK_STATUS_S_TX_RESP	Transmitting packet.	4
E_PMBUS_NWK_STATUS_S_ERROR	Detects Packet error.	5

e_pmbus_int_event_s_t

Enumeration Name	e_pmbus_int_event_s_t	
Description	This enumeration type indicates the cause of an I2C error detection interrupt. It is used to execute process for each interrupt cause in the internal process of the PMBus Slave Middleware.	
Declared header file	r_pmbus_app_slave.h	
Remarks	-	
Element name	**Description**	**Value**
E_PMBUS_INT_EVENT_S_NONE	No interrupt detected	0
E_PMBUS_INT_EVENT_S_ARB_LOST	Arbitration lost is detected.	1
E_PMBUS_INT_EVENT_S_TIMEOUT	Timeout detected.	2
E_PMBUS_INT_EVENT_S_NACK	NACK reception is detected.	3
E_PMBUS_INT_EVENT_S_START	A Start condition is detected.	4
E_PMBUS_INT_EVENT_S_STOP	A Stop condition is detected.	5
E_PMBUS_INT_EVENT_S_START_UNEXPECTED	Detects an unexpected timed Start condition.	6

st_pmbus_nwk_ctrl_s_t

Structure Name	st_pmbus_nwk_ctrl_s_t
Description	This is a structure that manages PMBus network layer (slave) parameters. It is used to manage communication status inside the PMBus Slave Middleware.
Declared header file	r_pmbus_app_slave.h
Remarks	-
Member name	**Description**
volatile e_pmbus_nwk_status_s_t e_status	Network layer status
uint8_t u1_current_addr_rw	Currently executing slave address (including RW)
uint8_t u1_current_cmd	Currently executing command
uint16_t u2_rx_index	Current number of received data bytes
uint16_t u2_rx_len	Number of data bytes to be received
uint16_t u2_rx_size	*p_u1_rx_buf Size
uint8_t *p_u1_rx_buf	Pointer to receive data storage buffer
uint16_t u2_tx_index	Current number of transmission data bytes
uint16_t u2_tx_len	Number of data bytes to be transmitted
uint16_t u2_tx_size	*p_u1_tx_buf Size
uint8_t *p_u1_tx_buf	Pointer to transmit data storage buffer
uint8_t u1_pec	Present PEC calculation

st_pmbus_ctrl_s_t

Structure Name	st_pmbus_ctrl_s_t
Description	This is the control data structure of the PMBus Middleware (slave). It is used to manage the PMBus Slave Middleware setting information and communication status.
Declared header file	r_pmbus_app_slave.h
Remarks	-
Member name	**Description**
st_pmbus_nwk_ctrl_s_t st_nwk_ctrl_s	Parameter-managed struct of PMBus network Layer (slave)
volatile e_pmbus_packet_result_s_t e_pmbus_result_s	Executing PMBus communication (slave)
bool b_open_flag	OPEN status (No false: Open or true: Open).
bool b_pec_flag	PEC enable/disable information (false: disable, true: enable)
uint8_t u1_own_slave_addr	Its own slave address.
void (* p_pmbusSlaveCmdCallback)(st_pmbus_nwk_ctrl_s_t * const p_st_nwk_ctrl, e_pmbus_packet_result_s_t * const p_e_pmbus_result)	Callback calls from PMBus interrupt handlers.

5.2.6 PMBus Slave Global variables List

The following table lists the global variables used in this control program.

Table 37 PMBus Slave Application global variable list

File Name	Global Variables	Usage
r_app_main.c	static uint8_t s_u1_pmbus_config_data	Global-variable that manages ON_OFF_CONFIG command-specification values. Only bit3 is valid in this demonstration.
	static uint8_t s_u1_pmbus_operation_data	Global-variable that manages OPERATION command-specification values. Only bit7 is valid in this demonstration.
	static uint8_t s_u1_pmbus_tx_buf[PMBUS_BUF_SIZE_MAX]	Buffer for storing transmit data for PMBUS. This buffer is set to the member *p_u1_tx_buf of s_user_pmbus_cfg and is used inside PMBus Middleware.
	static uint8_t s_u1_pmbus_rx_buf[PMBUS_BUF_SIZE_MAX]	Buffer for storing received data for PMBUS. This buffer is set to the member *p_u1_rx_buf of s_user_pmbus_cfg and is used inside PMBus Middleware.
	static volatile e_app_req_event_t s_e_req_status	Variable used to notify the main process of the cause of a PMBUS callback or the occurrence of other events.
	static st_pmbus_cfg_s_t s_st_pmbus_cfg	Variable that stores the configuration data to be registered by R_PMBUS_Slave_Open.

Table 38 PMBus Slave Middleware global variable list

File Name	Global Variables	Usage
r_app_main.c	static st_pmbus_ctrl_s_t g_st_pmbus_ctrl	Global variable that manages the control information of PMBUS Slave Middleware. It is used only within PMBUS Slave Middleware.
Config_RIIC0.c	volatile uint8_t g_riic0_user_slave_addr	In the RX26T, this is used to set the slave address specified by the user with R_PMBUS_Slave_Open to the SARL0 register.
	volatile uint8_t g_riic0_start_detect_at_receive	In the RX26T, this is a flag that manages the detection of a Start condition detection interrupt when the RIIC0 driver's g_riic0_mode_flag is in receive mode.
r_pmbus_wrapper_slave.c	static gpt_instance_ctrl_t s_st_gpt_ctrl	In the RA6T3, this variable is used as the control handler for the GPT driver.
	static smbus_slave_instance_ctrl_t s_st_smbus_ctrl	In the RA6T3, this variable is used as the control handler for the SMBUS driver.
	static smbus_slave_cfg_t s_st_smbus_cfg	In the RA6T3, this variable stores the configuration data set in the SMBUS driver by R_SMBUS_SLAVE_Oepn.
	smbus_slave_instance_t g_smbus_slave0	In the RA6T3, this instance used inside SMBUS drivers.

5.2.7 PMBus Slave macro-definition list

The following table lists the macro definitions used in this control program.

Table 39 PMBus Master Application macro definition list

File name	Macro name	Usage	Defined Value
r_app_main.h	PMBUS_APP_CMD_xxx	・Define the command code for PMBUS. (Refe to below for macro names)	-
		PMBUS_APP_CMD_OPERAION : OPERAION command	0x01
		PMBUS_APP_CMD_ON_OFF_CONFIG : ON_OFF_CONFIG command	0x02
		PMBUS_APP_CMD_CLEAR_FAULTS : CLEAR_FAULTS command	0x03
		PMBUS_APP_CMD_STATUS_FAN_1_2 : STATUS_FAN_1_2 command	0x81
		PMBUS_APP_CMD_READ_VOUT : READ_VOUT command	0x8B
		PMBUS_APP_CMD_READ_IOUT : READ_IOUT command	0x8C
		PMBUS_APP_CMD_READ_FAN_SPEED_1 : REA_DFAN_SPEED_1 command	0x90
		PMBUS_APP_CMD_READ_FREQUENCY : READ_FREQUENCY command	0x95
		PMBUS_APP_CMD_RESERVED : RESERVED command	0x09
		PMBUS_APP_CMD_PMBUS_REVISION : PMBUS_REVISION command	0x98
		PMBUS_APP_CMD_STORE_DEFAULT_CODE : STORE_DEFAULT_CODE command	0x13
		PMBUS_APP_CMD_FAN_COMMAND_1 : FAN_COMMAND_1 command	0x38
		PMBUS_APP_CMD_READ_EOUT : READ_EOUT command	0x87
		PMBUS_APP_CMD_PAGE_PLUS_WRITE : PAGE_PLUS_WRITE command	0x05
	SLAVE_ADDRESS	This defines the slave address to be set in the PMBus Slave Middleware.	0x0A
	OPERATION_OPE_START_BIT	Defined OPERATION command-motor control indication bit.	0x80
	ON_OFF_CONFIG_OPE_ENABLE_BIT	The data-definition used in ON_OFF_CONONFIG command. Specifies whether to use the motor control instruction setting of OPERATION command when controlling the motor.	0x08
	RET_OK	Return value of the application internal function. Normal end.	0
	RET_ERROR	Return value of the application internal function. Abnormal end.	1

Table 40 PMBus Slave API macro definition list

File name	Macro name	Usage	Defined Value
r_pmbus_app_slave.h	PMBUS_RET_xxx	・Error code returned from PMBus middleware API. (See below for macro names)	-
		PMBUS_RET_OK : Normal end.	0
		PMBUS_RET_ERROR : Abnormal end. See the st_pmbus_cfg_s_t.e_pmbus_result_m for more information about the source.	1
		PMBUS_RET_PARAM : Specified argument is invalid.	2
		PMBUS_RET_NOT_OPENED : No OPEN.	3
		PMBUS_RET_OPENED : Already OPEN.	4

Table 41 PMBus Slave Middleware function macro definition list

File name	Macro name	Usage	Defined Value
r_pmbus_app_slave.h	PMBUS_TRANS_xxx	・Defines the transaction code used to determine the protocol supported by each command code. (See below for macro names)	-
		PMBUS_TRANS_RESERVED : Command code is RESERVED.	0x00
		PMBUS_TRANS_READ_BYTE : Command Code Supports READ BYTE Transactions.	0x01
		PMBUS_TRANS_READ_WORD : Command Code Supports READ WORD Transactions.	0x02
		PMBUS_TRANS_BLOCK_READ : Command Code Supports BLOCK READ Transactions.	0x03
		PMBUS_TRANS_SEND_BYTE : Command Code Supports SEND BYTE Transactions.	0x10
		PMBUS_TRANS_WRITE_BYTE : Command Code Supports WRITE BYTE Transactions	0x20
		PMBUS_TRANS_WRITE_WORD : Command Code Supports WRITE WORD Transactions.	0x30
		PMBUS_TRANS_BLOCK_WRITE : Command Code Supports BLOCK WRITE Transactions.	0x40
		PMBUS_TRANS_WRITE_QUICK : Command Code Supports Write Quick Command Transactions.	0x50
		PMBUS_TRANS_PROCESS_CALL : Command Code Supports PROCESS CALL Transactions.	0x60
		PMBUS_TRANS_BLOCK_PROCESS_CALL : Command Code Supports Block Write-Block Read Process Call Transactions.	0x70
	PMBUS_COMMAND_CODE_MAX	The maximum number of commands supported by the PMBus middleware.	256
	PMBUS_BLOCK_SIZE_MIN	Min. amount of data that can be sent/received by Block commandI will.	1
	PMBUS_BLOCK_SIZE_MAX	Max. amount of data that can be sent/received by Block commandI will.	32
	PMBUS_BUF_SIZE_MIN	The smallest buffer size that can be registered in PMBUS Middleware during Open.	1
	PMBUS_BUF_SIZE_MAX	The largest buffer size that can be registered in PMBUS Middleware during Open. (Max. number of data to write during	PMBUS_BLOCK_

		Block Read/Block Write (32) + Command code (1) + Number of data to write/Number of data to read (1) + PEC (1))	SIZE_MAX + 3
	PMBUS_CRC8_USE_IP	Defined value that specifies the PEC calculation method. Set to "1 (uses the calculator built into the MCU)" or "0 (specifies whether to use a table for calculation)." If you want to use the calculator built into the MCU, you must implement code that uses a CRC calculator in the r_pmbus_nwk_AddCrc8() function and then change the setting to "1 (uses the calculator built into the MCU)."	0
	PMBUS_ALERT_RESPONSE_ADDR	Slave address (ARA) for responding to ALERT information set at execution time R_PMBUS_Slave_SendARA. Values conforming to the SMBus specifications are defined.	0x0C
	PMBUS_SLAVE_ADDR_MIN	Defines the highest slave address that can be specified as a parameter in the slave API of PMBUS.	0x01
	PMBUS_SLAVE_ADDR_MAX	Defines the highest slave address that can be specified as a parameter in the slave API of PMBUS.	0x10
r_pmbus_wrapper_slave.h	PMBUS_CFG_DEVICE_RX26T	Used when using the wrapper function of the driver provided in r_pmbus_wrapper_slave.c with the RX26T.	0
	PMBUS_CFG_DEVICE_RA6T3	Used when using the wrapper function of the driver provided in r_pmbus_wrapper_slave.c with the RA6T3.	1
	PMBUS_CFG_DEVICE	Definition of the MCU that executes the wrapper function of the driver provided in r_pmbus_wrapper_slave.c. Specify PMBUS_CFG_DEVICE_RX26T or PMBUS_CFG_DEVICE_RA6T3.	Follow MCU used.

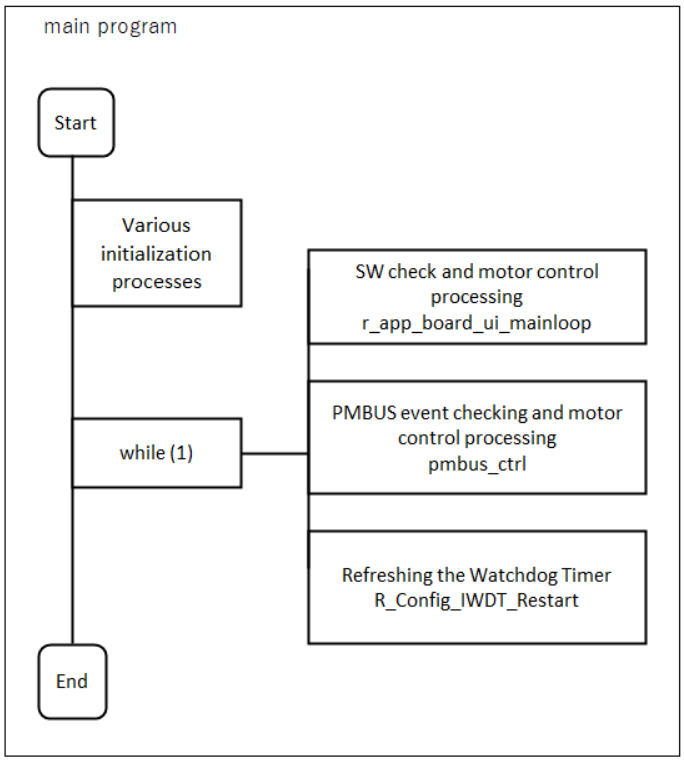
5.2.8 PMBus Slave Control Flowchart

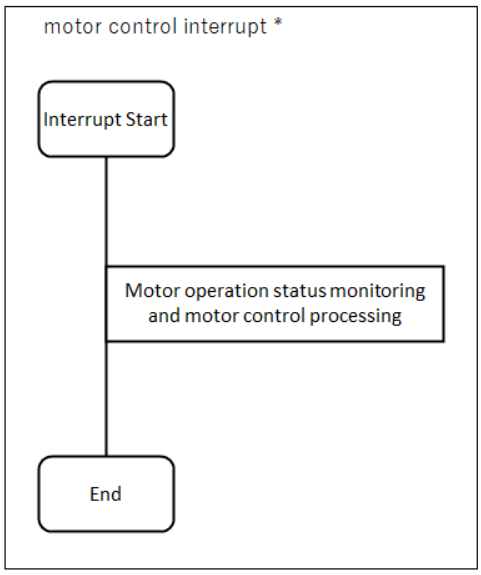
The flow of PMBus Slave Application part is shown in 5.2.8.1, the flow of PMBus Slave API part is shown in 5.2.8.2, and the flow of PMBus Slave driver part is shown in 5.2.8.3. Please refer to the project code for PMBus Slave Middleware part.

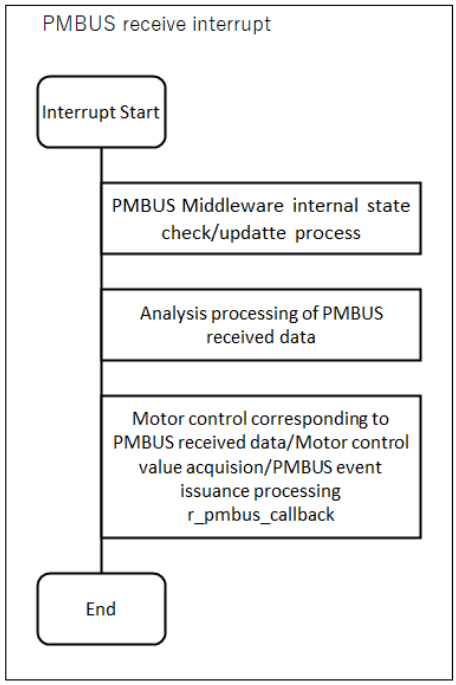
5.2.8.1 PMBus Slave Application section flowchart

PMBus Slave Application section controls the Motor Control Middleware and API calls to PMBus Slave Middleware section that communicates with PMBus Master. The flowchart of PMBus Slave Application part is shown below.

- PMBus slave overall outline flow
- PMBus slave overall outline flow (RX26T)

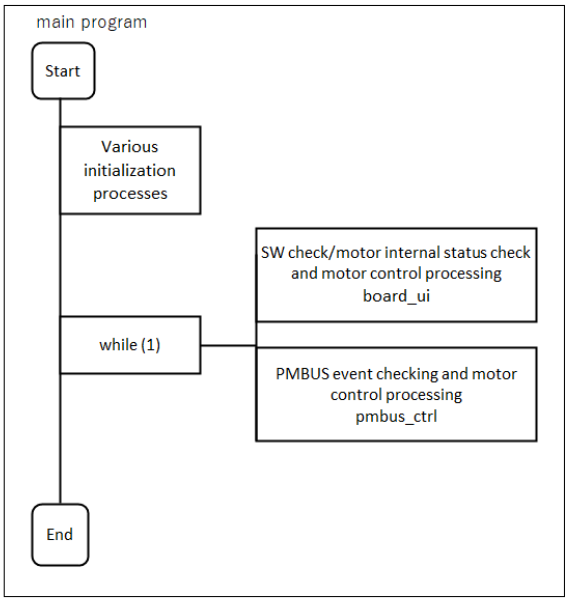


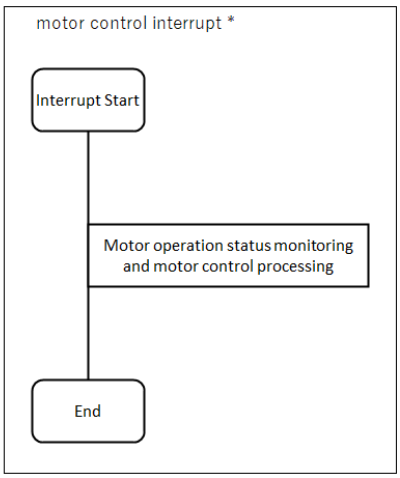


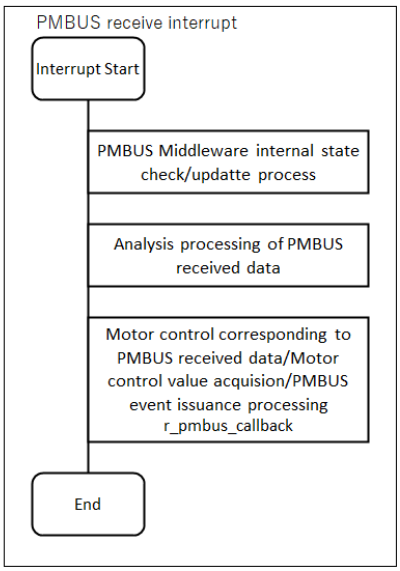


note: For details, refer to RX Famiky Sensorless Vector Control of a Permanent Magnet Synchronous Motor - For MCK (R01AN6858)

- PMBus slave overall outline flow (RA6T3)





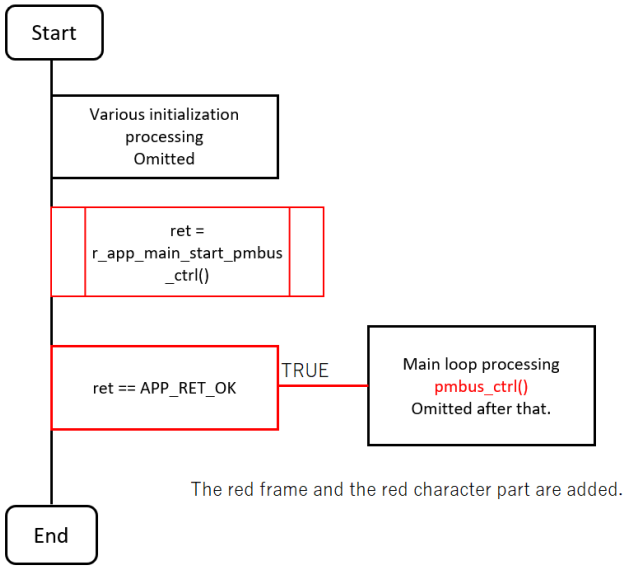


note: For details, refer to Sensorless vector control for permanent magnetic synchronous motor For Renesas Flexible Motor Control (R01AN6839)

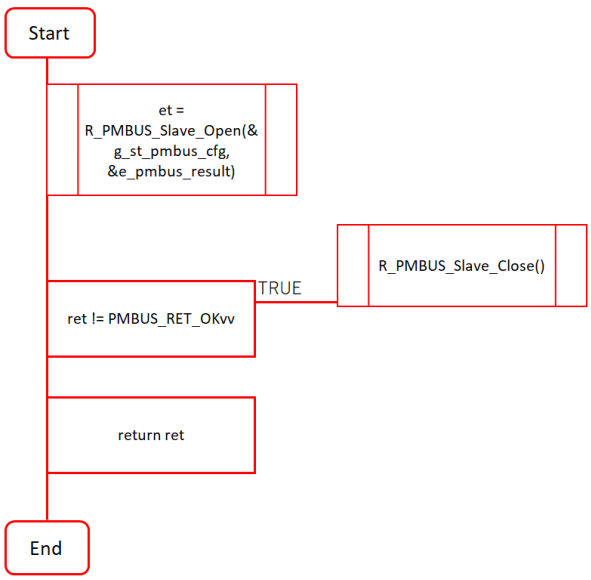
■ PMBus slave function detailed flow

● main (RX26T), hal_entry (RA6T3)

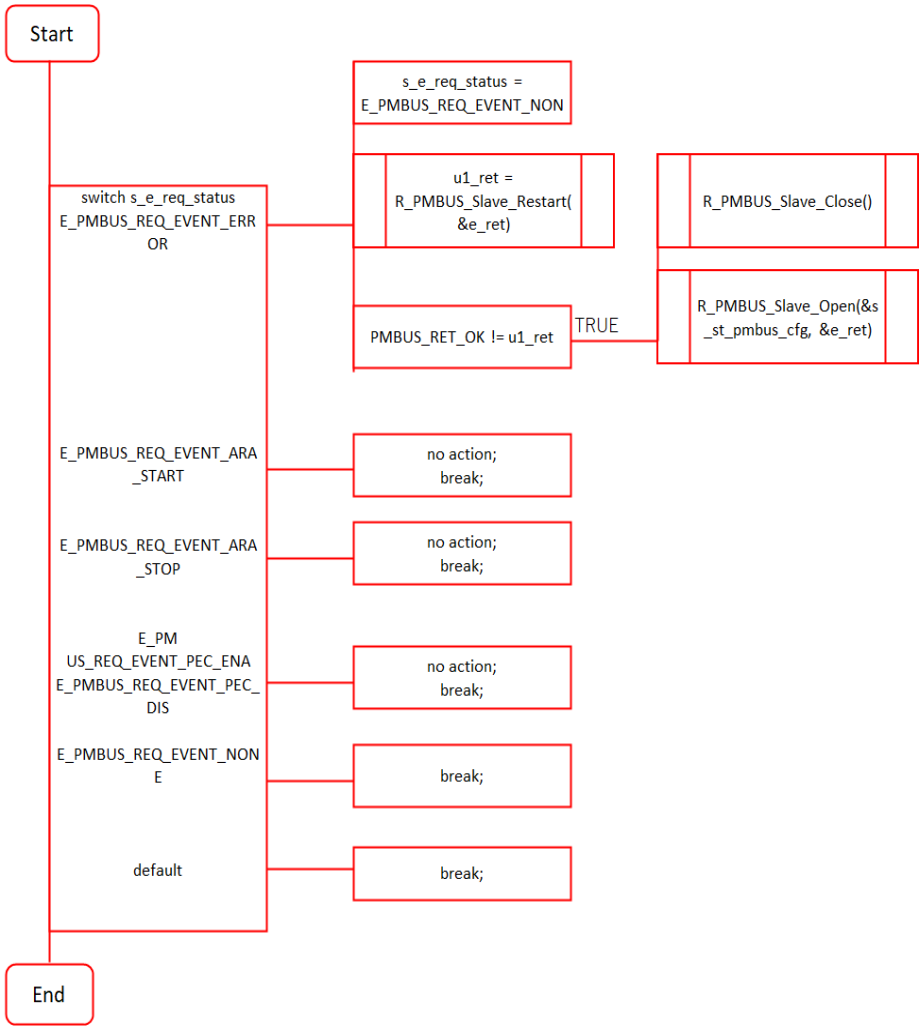
Application part of the motor control middleware is used for various initialization processes and main-loop processes. Refer to Motor Middle application notes for details. The red framed and red text parts of each PAD diagram are the parts to be corrected, and the blue framed and blue text parts are the parts to be deleted.



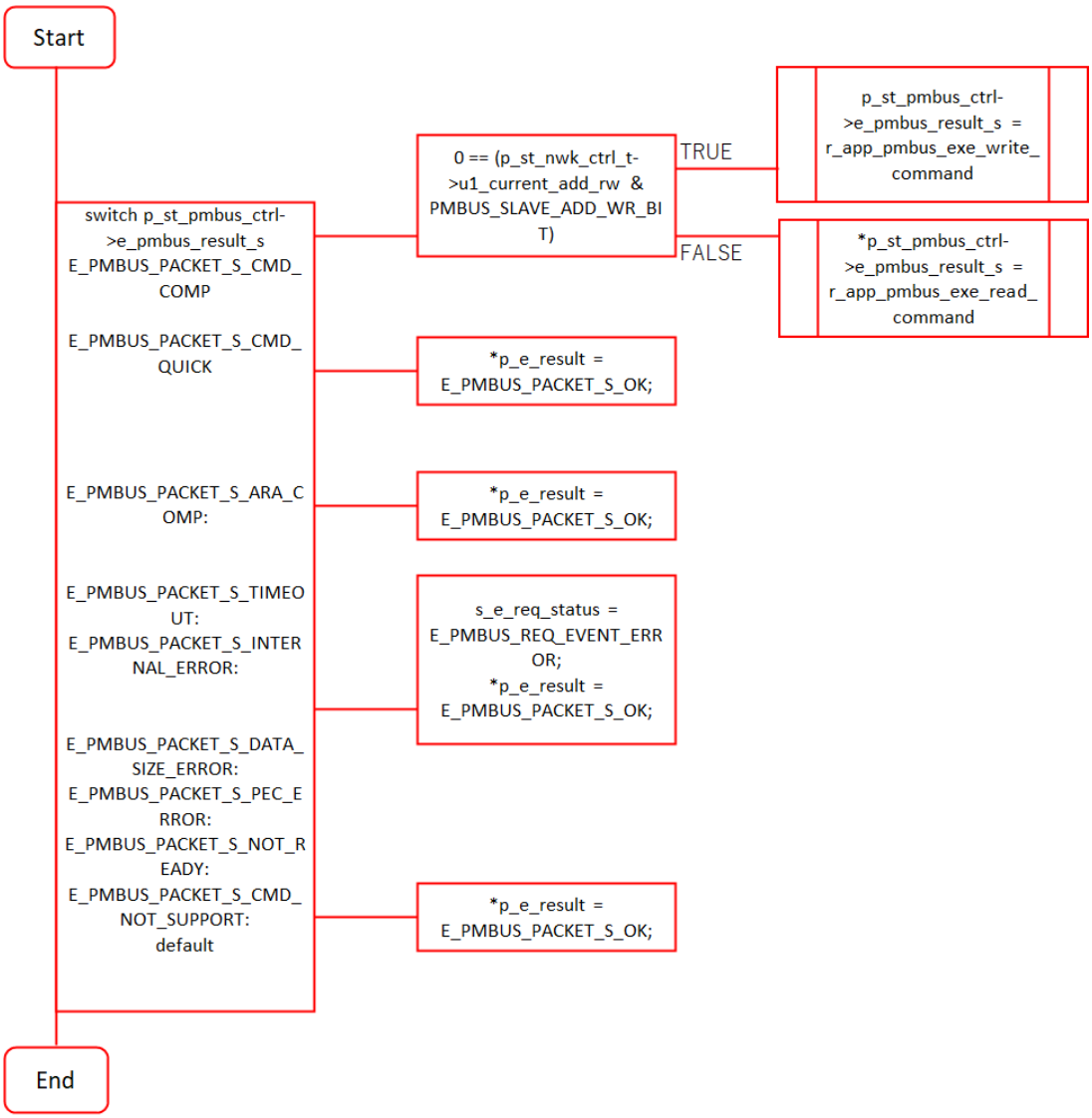
● r_app_main_start_pmbus_ctrl (RX26T, RA6T3)



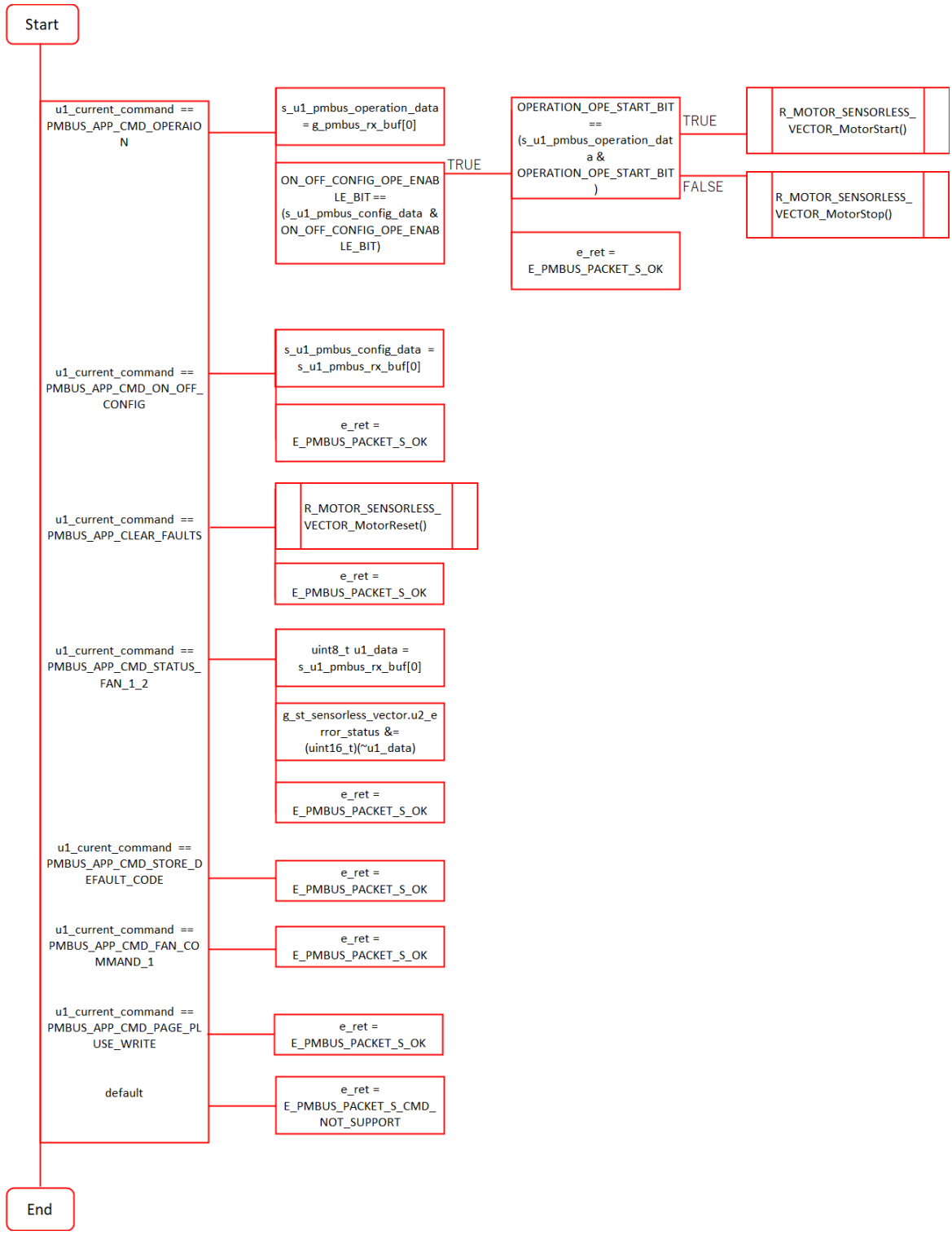
- pmbus_ctrl (RX26T, RA6T3)



- r_pmbus_callback (RX26T, RA6T3)

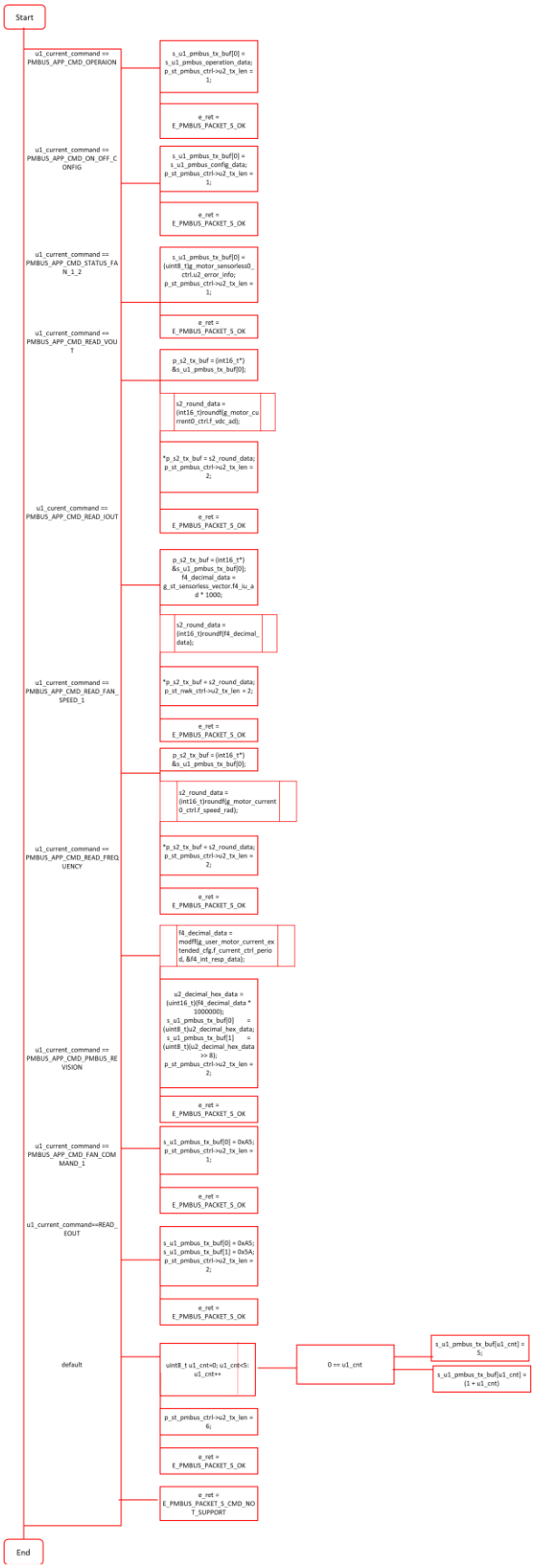


- r_app_pmbus_exe_write_command (RX26T, RA6T3)



[Note] The motor sample API names in the diagram are different for RA6T3. Please refer to the project for details.

- r_app_pmbus_exe_read_command (RX26T, RA6T3)

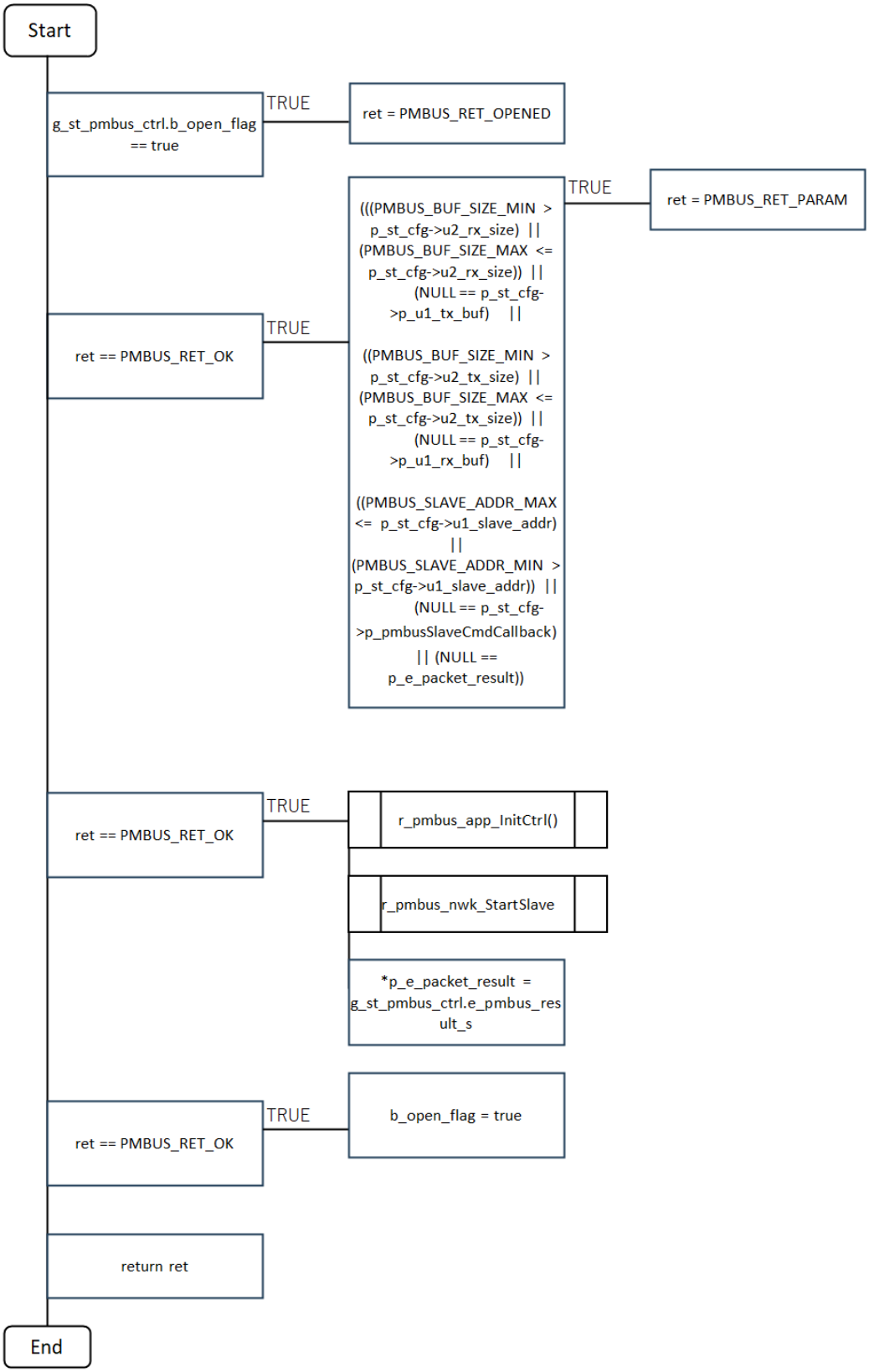


[Note] The motor sample API names in the diagram are different for RA6T3. Please refer to the project for details.

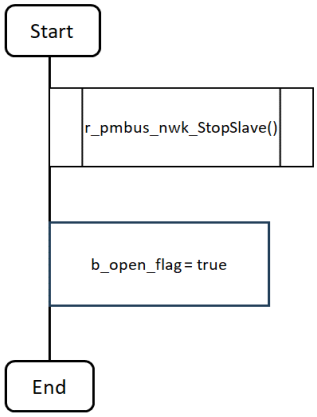
5.2.8.2 PMBus Slave API section flowchart

PMBus Slave API part controls PMBus Slave Middleware part. The flowchart for PMBus Slave API part is shown below.

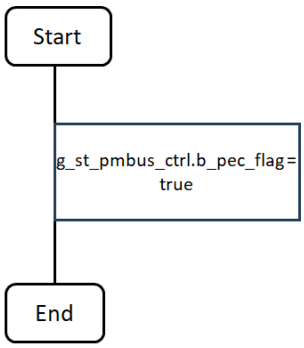
- R_PMBUS_Slave_Open



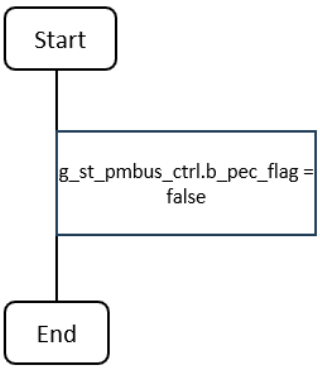
- R_PMBUS_Slave_Close



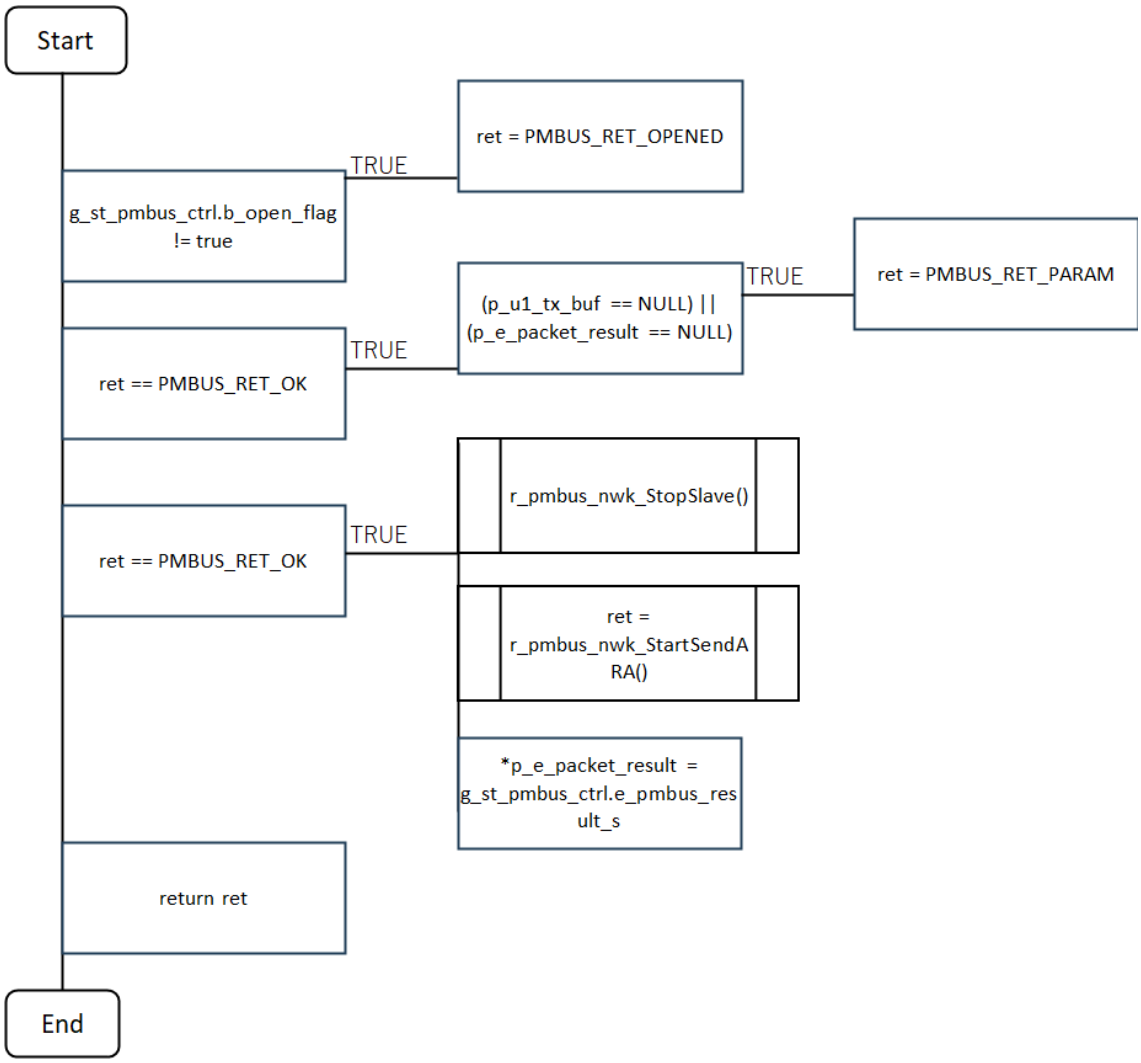
- R_PMBUS_Slave_EnablePEC



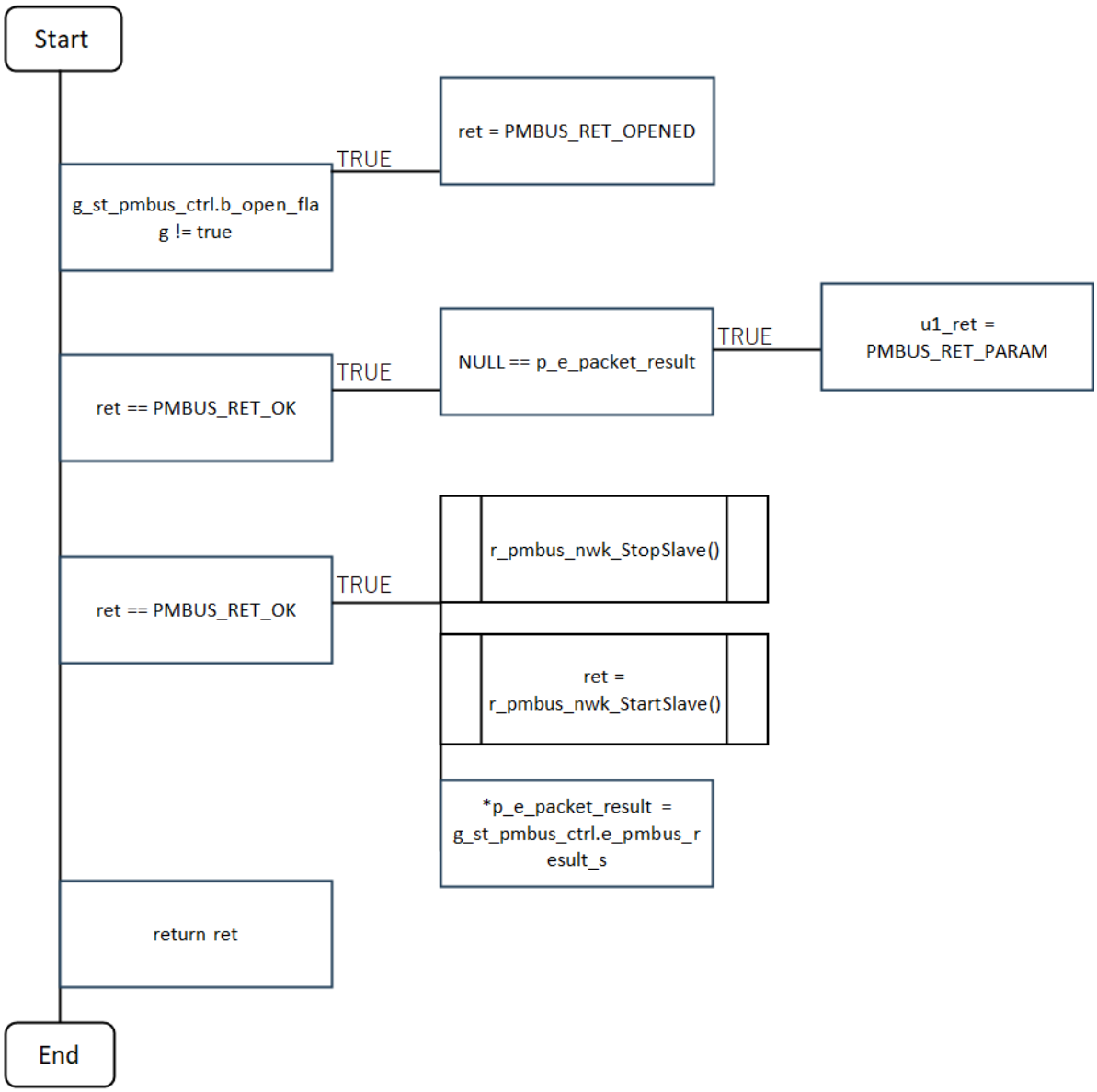
- R_PMBUS_Slave_DisablePEC



- R_PMBUS_Slave_SendARA



- R_PMBUS_Slave_Restart



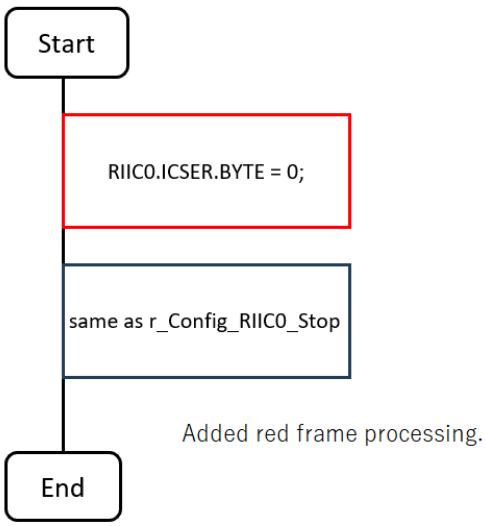
5.2.8.3 PMBus Slave Drivers Flowchart

PMBus Slave driver section is a smart configurator in RX26T, and in RA6T3, it is partially changed according to PMBus Slave process from the code generated in FSP. Refer to Customizing 5.2.4 PMBus Slave Drivers for details. The deficit in each pad diagram is the correction part. The red framed and red text parts of each PAD diagram are the parts to be corrected, and the blue framed and blue text parts are the parts to be deleted.

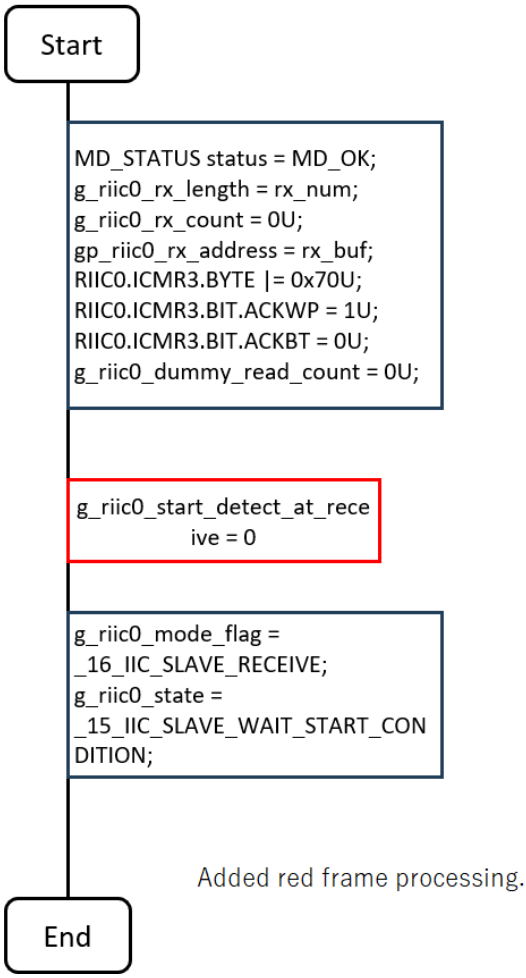
(1) Smart Configurator (RX26T)

- R_Config_RIIC0_Create

- R_Config_RIIC0_Stop

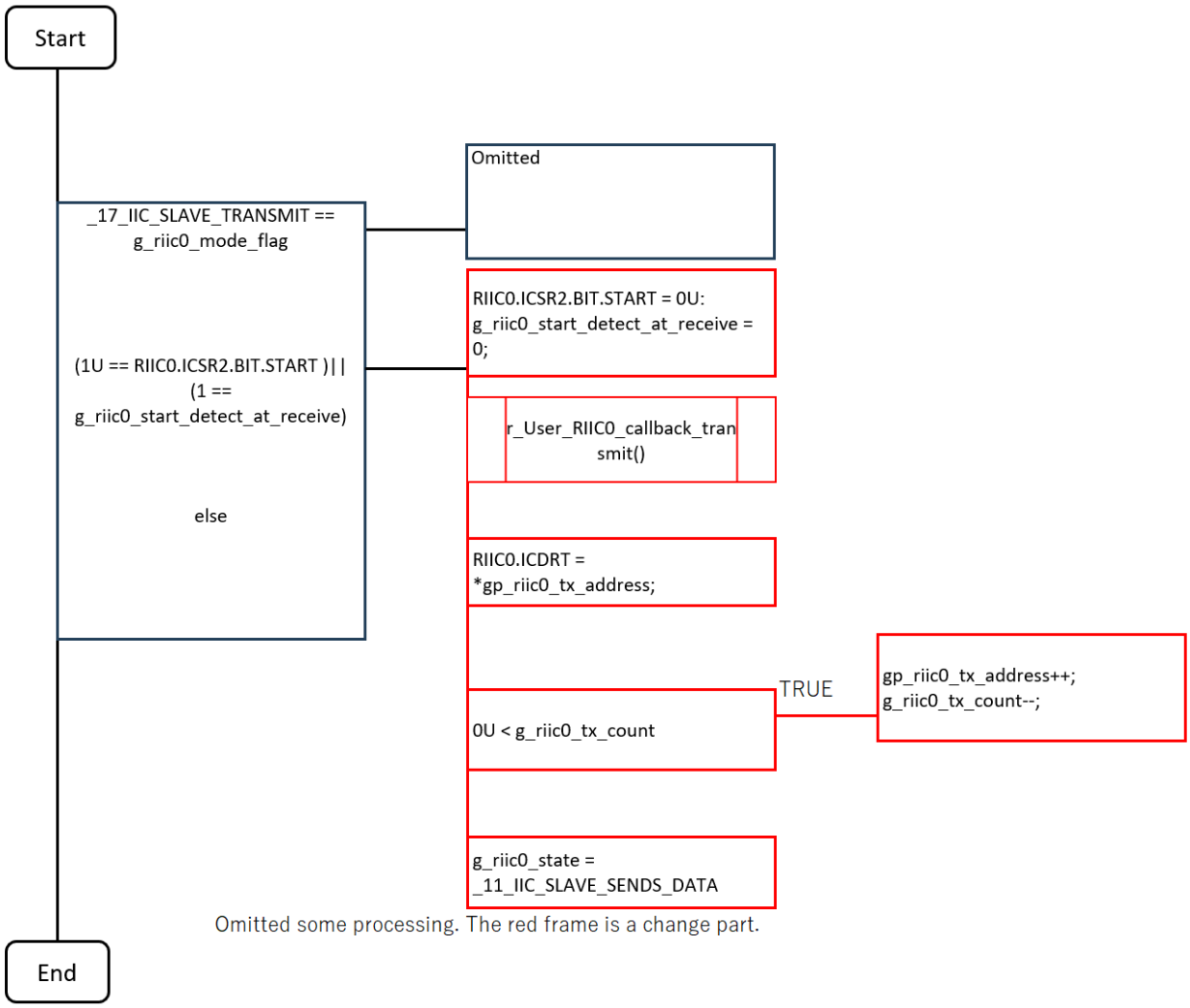


- r_Config_RIIC0_Slave_Receive



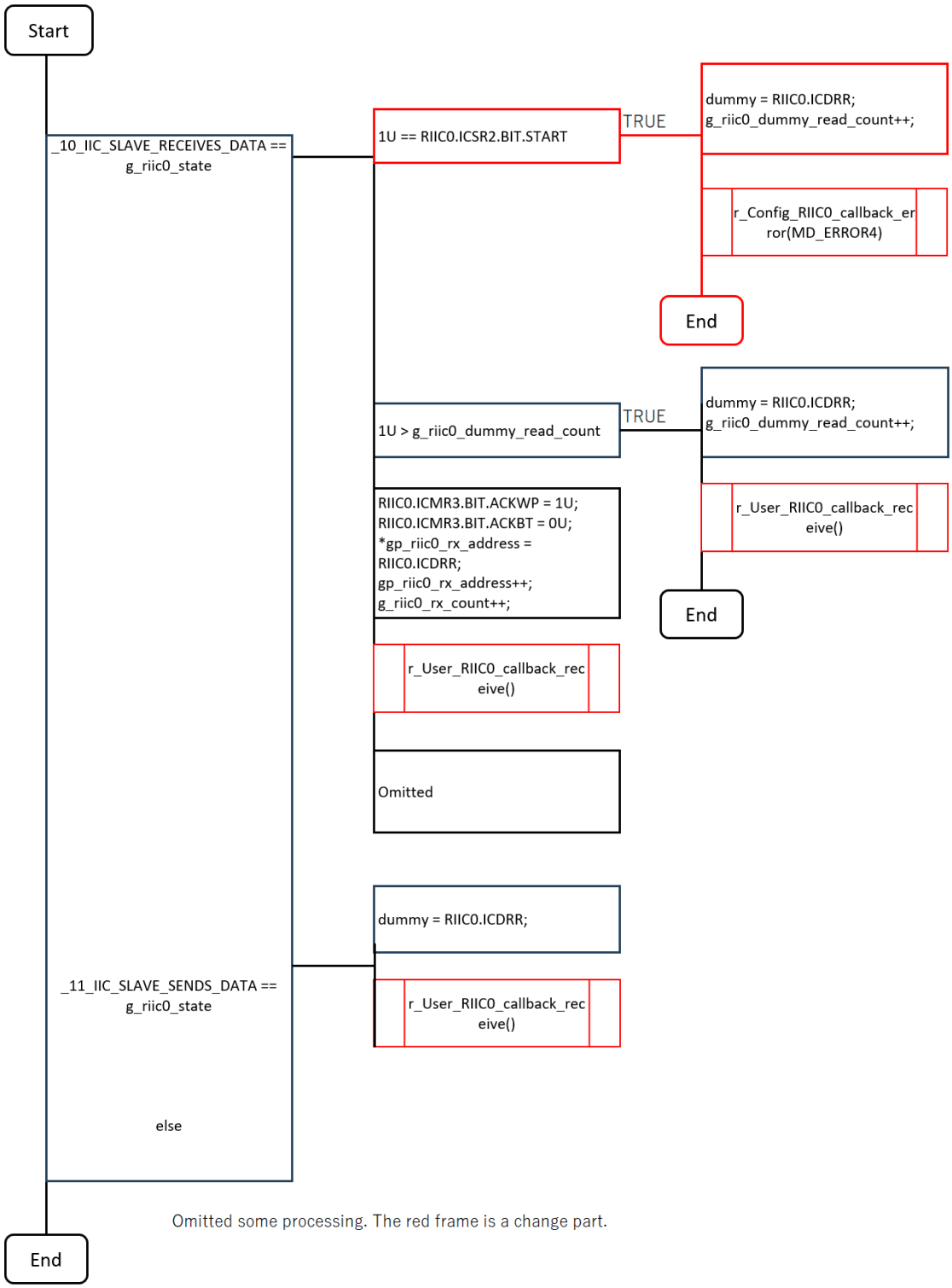
Added red frame processing.

- r_Config_RIIC0_transmit_interrupt



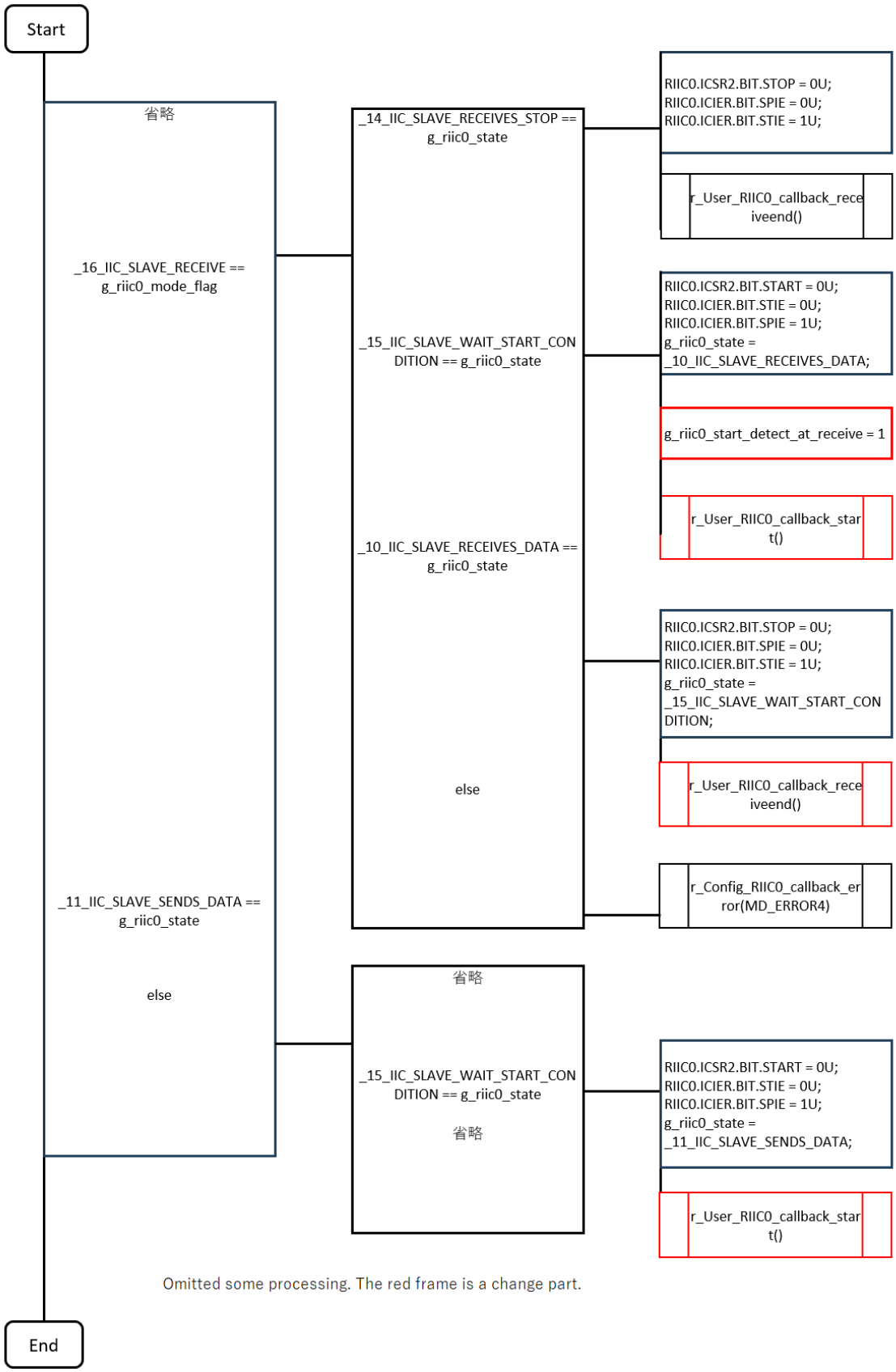
Omitted some processing. The red frame is a change part.

- r_Config_RIIC0_receive_interrupt



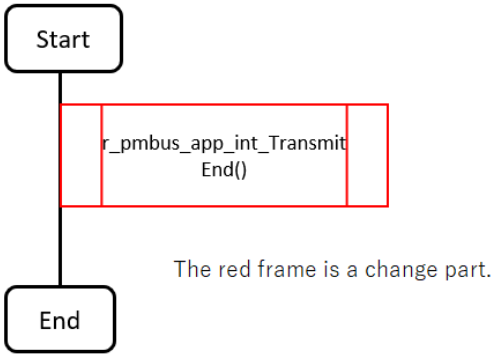
Omitted some processing. The red frame is a change part.

- r_Config_RIIC0_error_interrupt

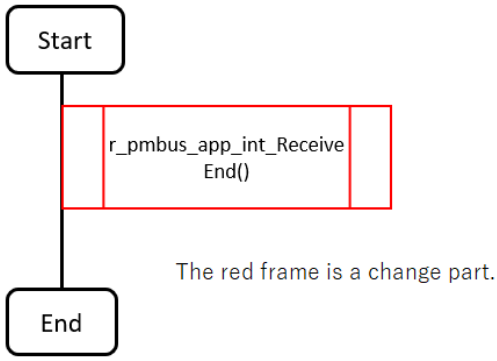


Omitted some processing. The red frame is a change part.

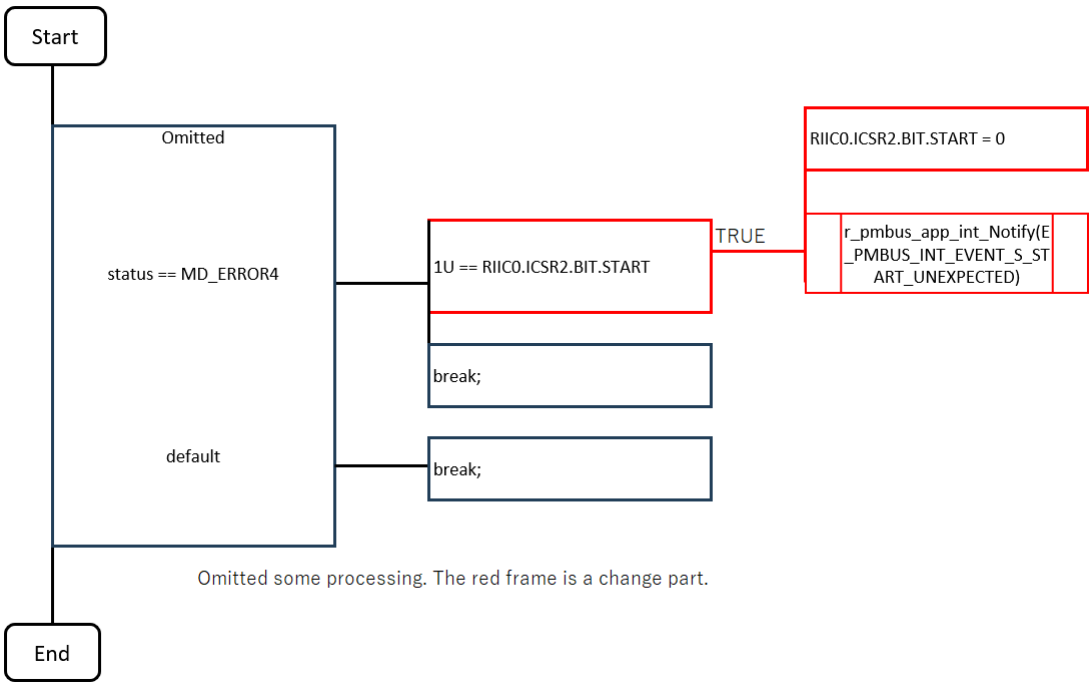
- r_Config_RIIC0_callback_transmitend



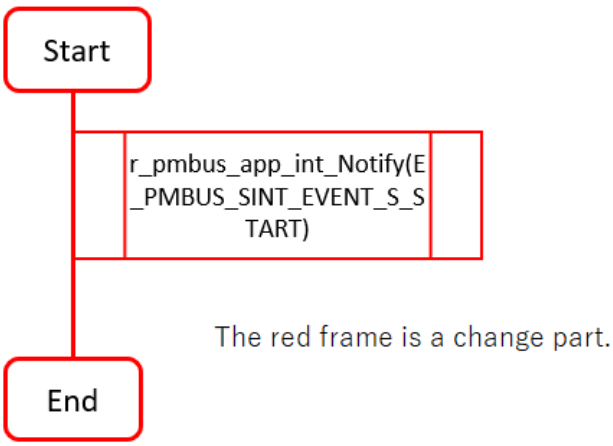
- r_Config_RIIC0_callback_receiveend



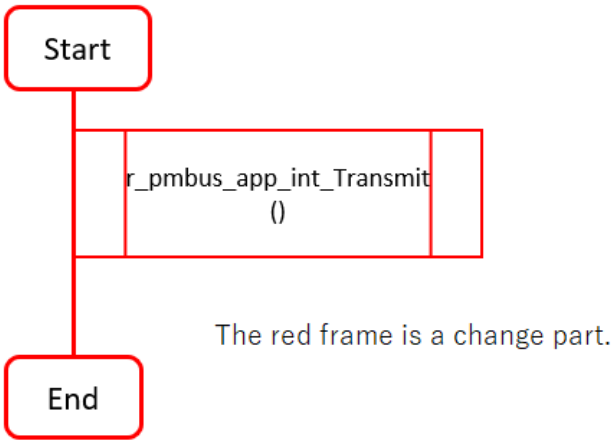
- r_Config_RIIC0_callback_error

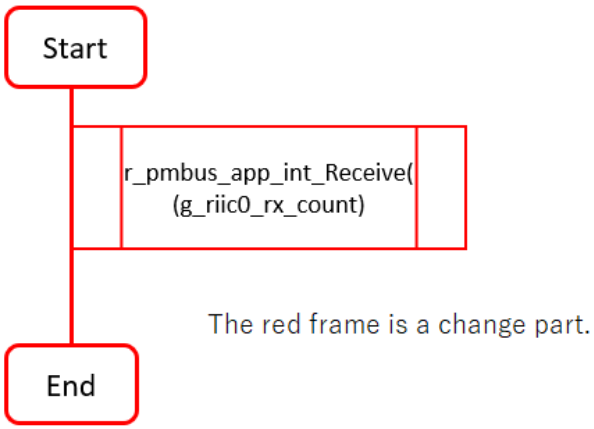


- r_User_RIIC0_callback_start



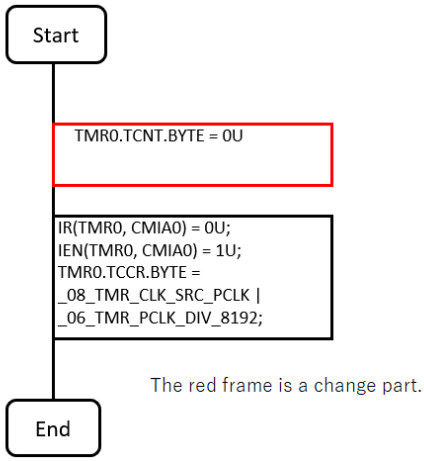
- r_User_RIIC0_callback_transmit





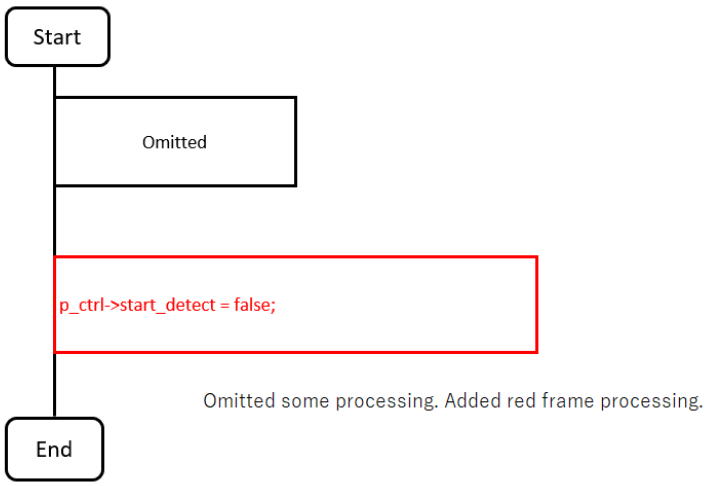
- r_User_RIIC0_callback_receive

- R_Config_TMR0_Start

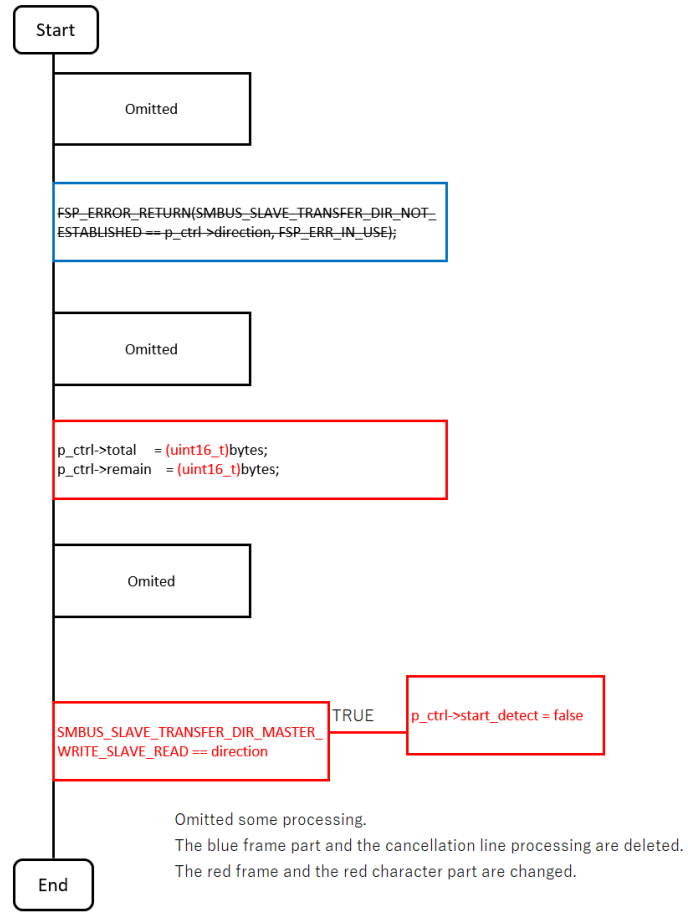


(2) FSP functional (RA6T3)

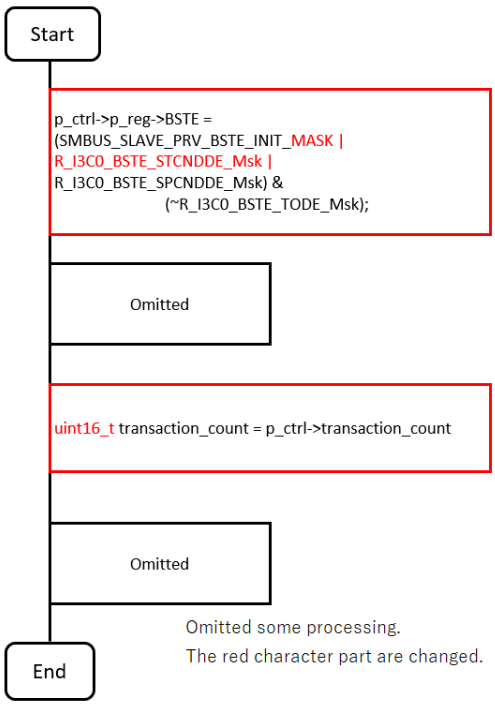
- R_SMBUS_SLAVE_Open



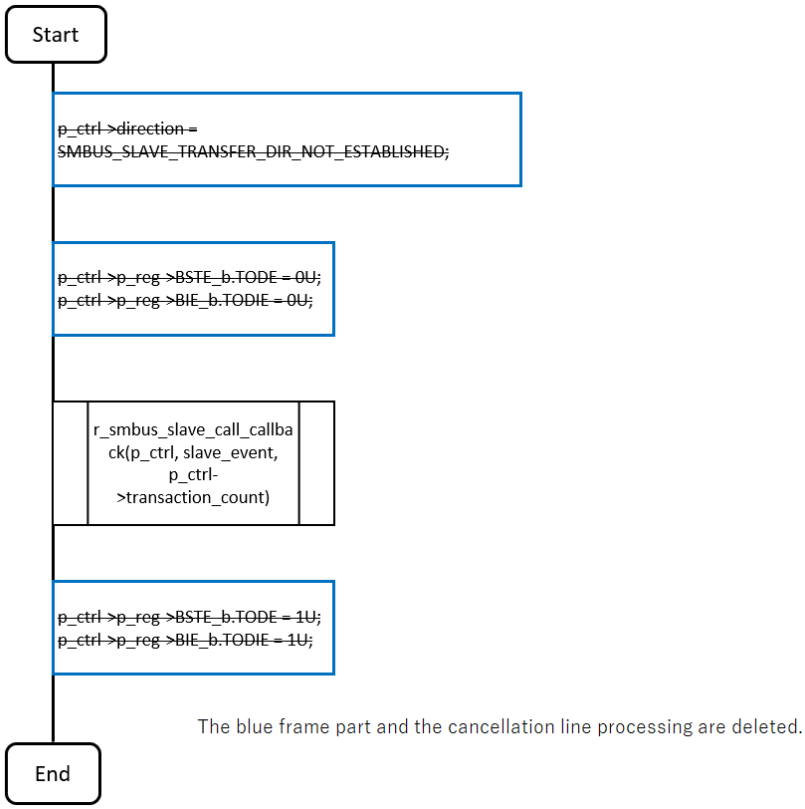
- r_smbus_slave_read_write



- r_smbus_slave_notify

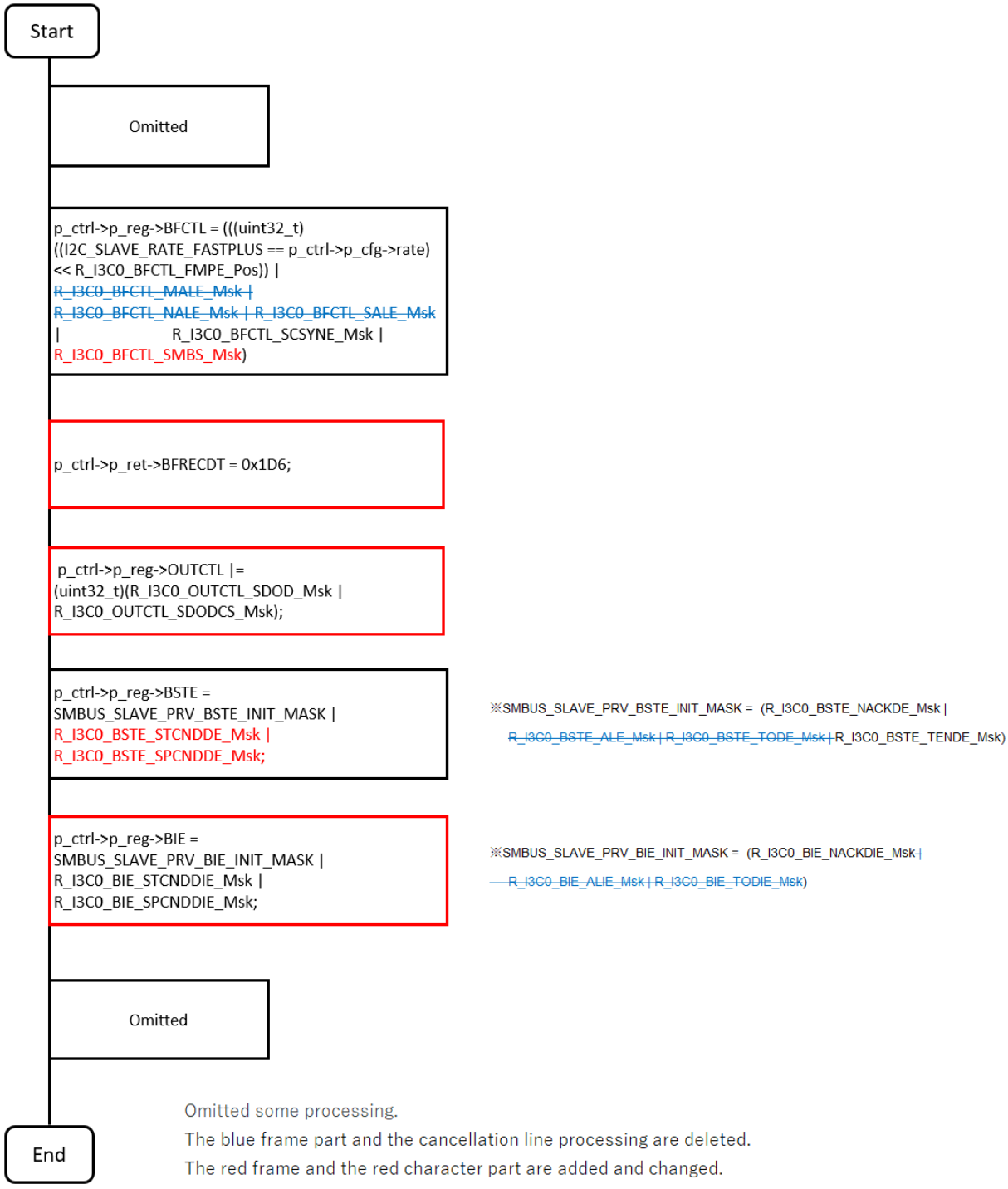


- r_smbus_slave_callback_request

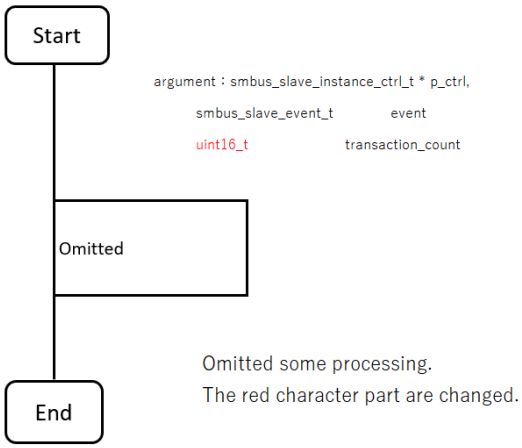


The blue frame part and the cancellation line processing are deleted.

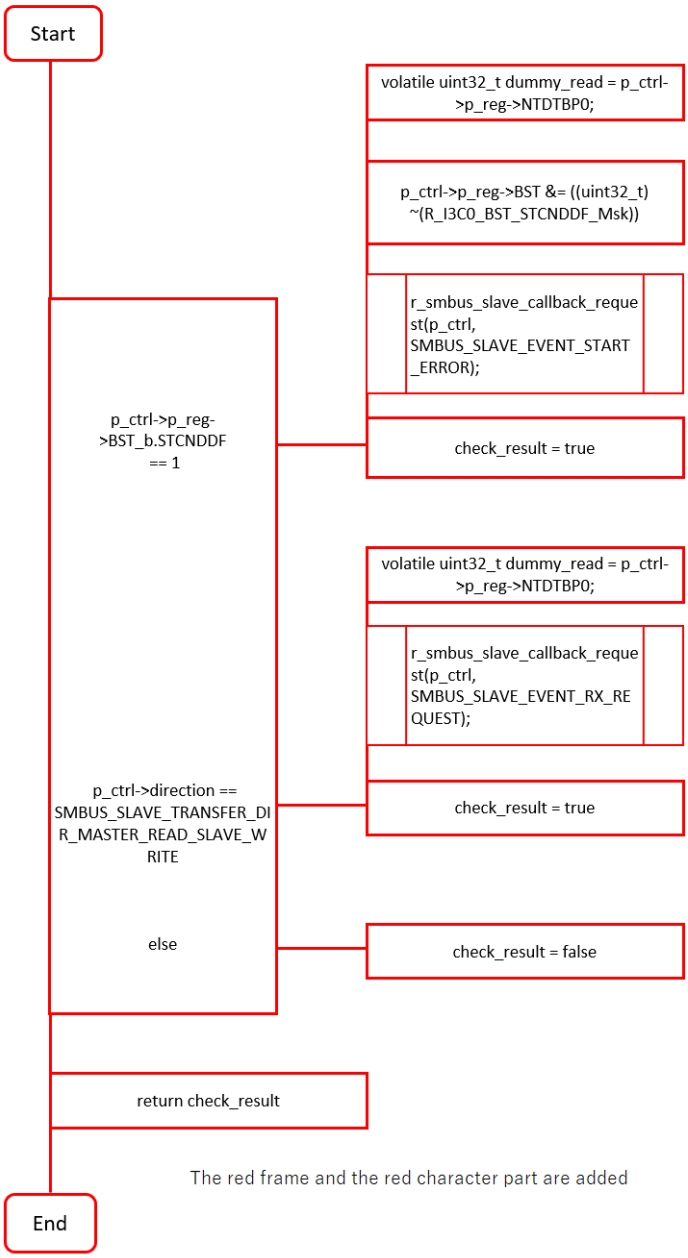
- r_smbus_open_hw_slave



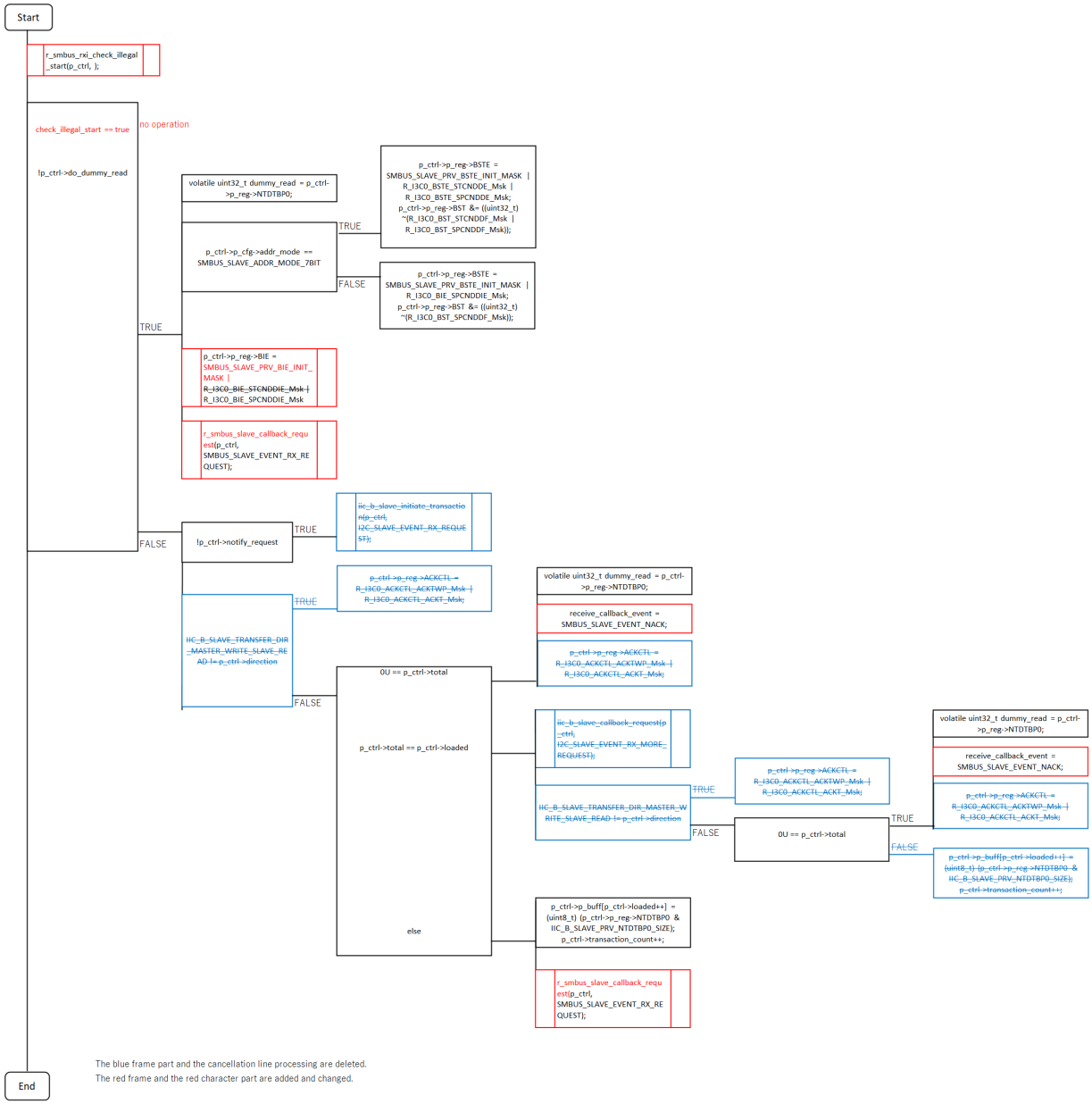
- r_smbus_slave_call_callback



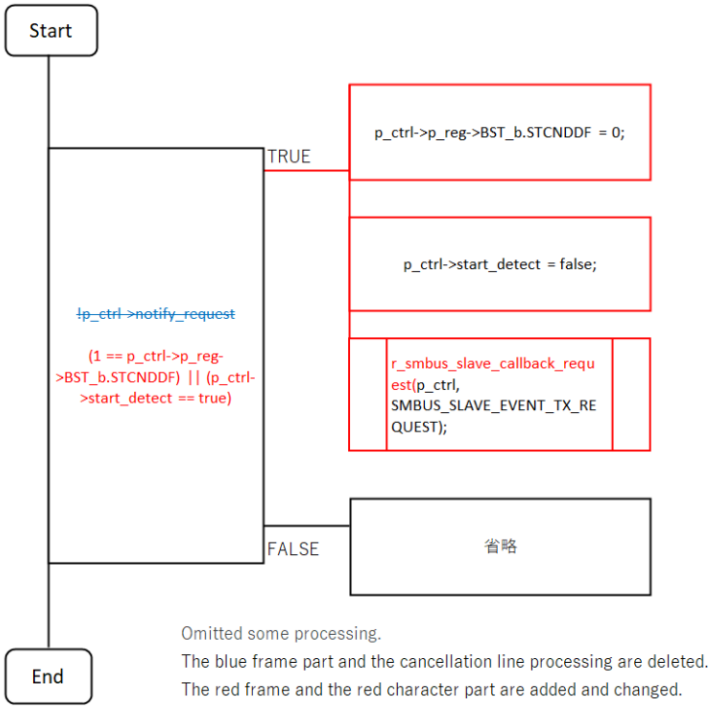
- r_smbus_rxi_check_illegal_start



- r_smbus_rxi_slave

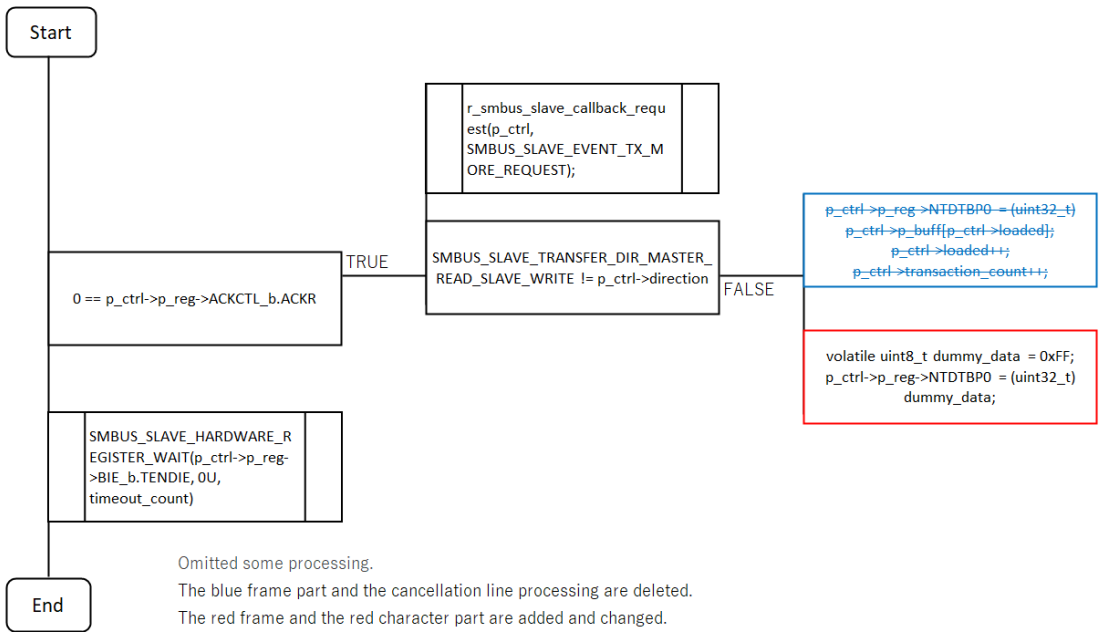


- r_smbus_txi_slave



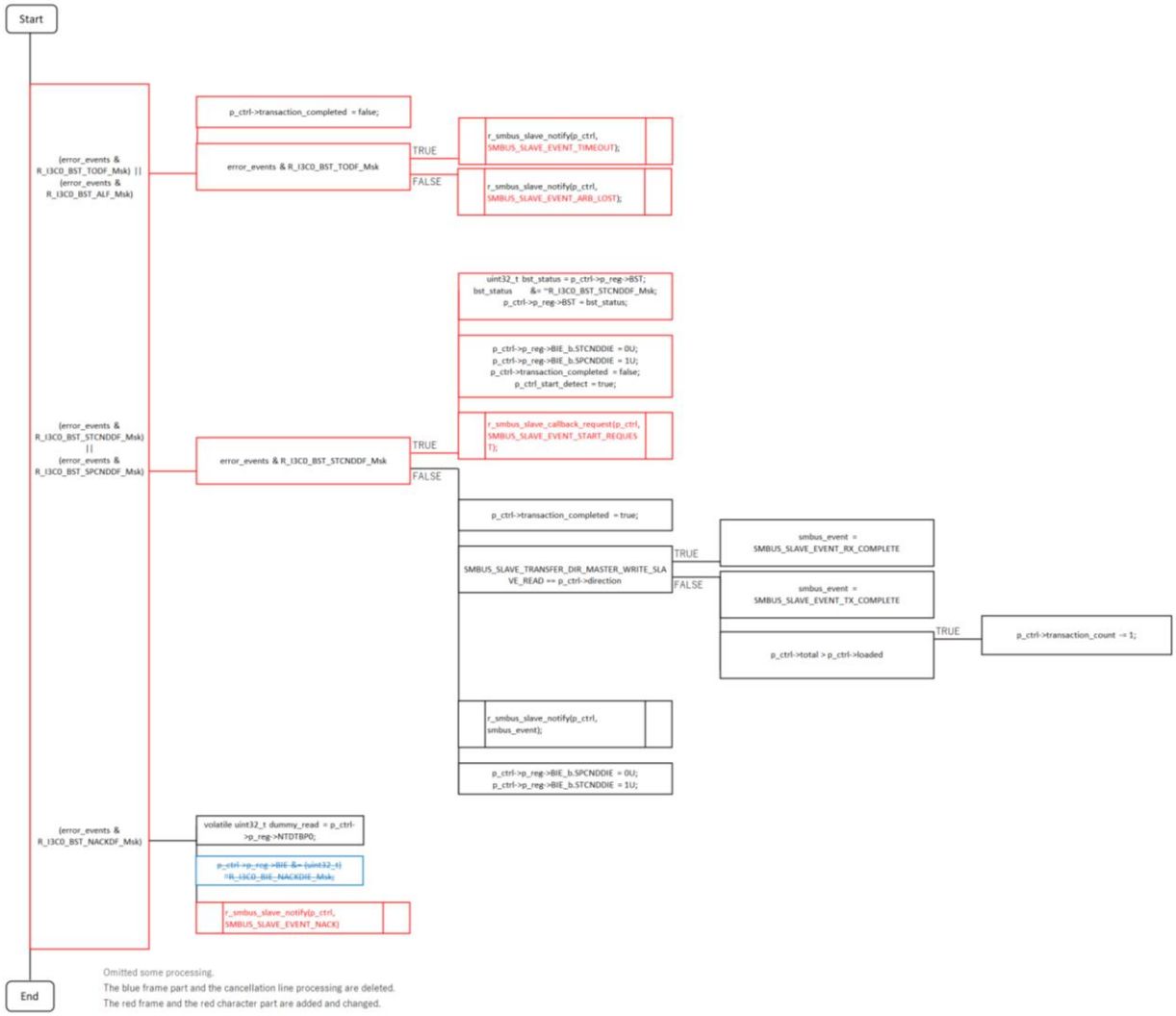
Omitted some processing.
The blue frame part and the cancellation line processing are deleted.
The red frame and the red character part are added and changed.

- r_smbus_tei_slave



Omitted some processing.
The blue frame part and the cancellation line processing are deleted.
The red frame and the red character part are added and changed.

- r_smbus_err_slave

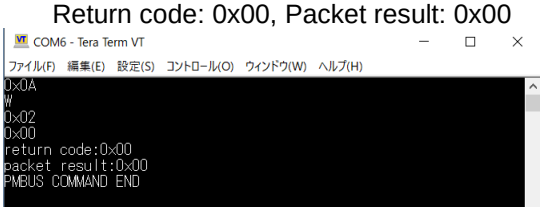


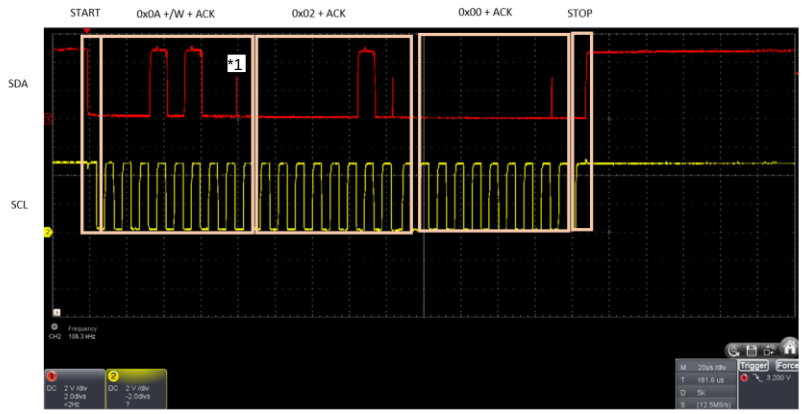
6. PMBus command-transmit/receive test-result

The following shows sample communication using tera term.

- ON_OFF_CONFIG (write) WRITE_BYTE protocol

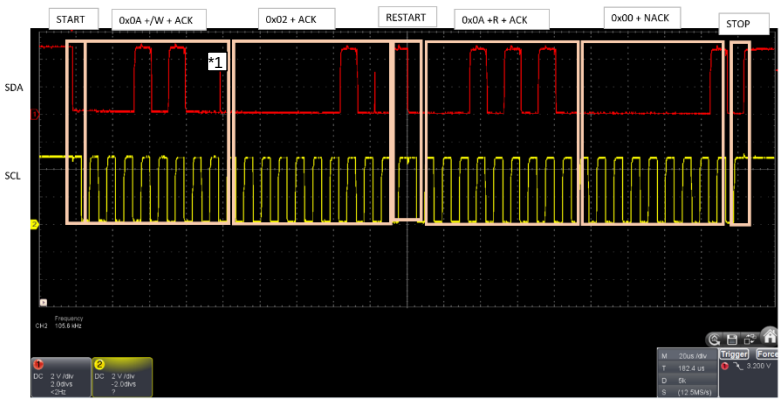
  Transmission :
  Slave address: 0x0A, READ/WRITE orientation: W, Command code: 0x02, Write data: 0x00
  Reception :
  Return code: 0x00, Packet result: 0x00





- ON_OFF_CONFIG (read) READ_BYTE protocol

  Transmission :
  Slave address: 0x0A, READ/WRITE orientation: R, Command code: 0x02
  Reception :
  Read data: 0x00, Return code: 0x00, Packet result: 0x00



*1: This is due to ACK timing and does not affect PMBus communication.

7. FAQ

■ Question1:

Are there any PMBUS specifications that cannot be checked by this application?

Answer :

The following specifications cannot be checked.

- Group command protocol
- Extended command protocol
- Zone Command Protocol
- Bus master protocol
- Quick Command (read) Protocol
- Address resolution protocol(ARP)
- Control of Control signals (including Alert signals)
- Failure Management and Reporting Features
- Multi-master operation
- Clock stretching function
- Suspend mode notification by SMBUS signal
- Slave bus master switching function
- Host communication
- Internal-memory protective function by Write Protect signal
- General Call Operation
- Functionality added in SMBUS 3.0.0 or later and PMBUS 1.3.0 or later.

You can also use the following specifications by adjusting your application and PMBUS middleware. This item should be evaluated by yourself.

- Commands Using Block Write
- Commands Using Block Read
- Commands Using Process Call
- Commands Using Block Write-Block Read Process Call
- Quick Command (write) Protocol
- Packet Error Check (PEC) Communication with
- Alert reply using Alert Response Address (ARA)

■ Question2 :

What is the setting of terminal software for communication between PC and master?

Answer :

Make the following settings.

Bit rate :115200 bps
Data length :8 bit
Parity: None
Stop bit: 1 bit
Data-transfer-direction: LSB first
Line feed: Receive: LF, Send: CR+LF

■ Question3 :

In this application, CRC calculation for communication with PEC uses a conversion table. Is it feasible to use CRC calculator installed in MCU?

Answer :

Yes Possible. When using CRC calculator in this application, change 4.1.6 PMBus Master macro definition list and 4.2.6 PMBus_Slave macro definition list to define "PMBUS_CRC8_USE_IP" to "1", and then change "r_pmbus_nwk_AddCrc8()" which is implemented as an empty function to operate using CRC of MCU. API for controlling CRC calculator of MCU can be generated by the smart configurator and FSP.

■ Question4 :

If the motor does not stop, what action should be taken?

Answer :

The motor can be stopped regardless of PMBUS command-receiving status by turning OFF SW1 (toggled SW). When restarting the motor rotation, eliminate the reason for the error, set SW1 (toggle SW) to the position where the motor can rotate, and then send the required ON_OFF_CONFIG command and OPERATION command of PMBUS to restart the motor rotation.
It may also stop if an error is detected in the motor sample used in this application. Refer also to the application notes for each motor sample for details.

■ Question5 :

When a user application on the slave side is customized and evaluated, a "packet result:0x03" (timeout detection) is returned from the terminal software. What are the possible causes?

Answer :

The problem may be caused by the high load of process added to the slave by the customer.
This application monitors the completion of the protocol-specified PMBUS(SMBUS within 25ms ($T_{TIMEOUT}$). For protocols that include Master receive /Slave transmissions, such as READ BYTE protocol, the callback process on the slave is executed prior to sending Slave, so a timeout occurs if the callback process is overloaded. If you want to Execute high-load process, consider executing high-load process in main process by sending an event notification to main loop process from the callback using the pmbus_ctrl() and e_pmbus_int_event_s_t that are executed periodically in the loop process.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan.14.25	-	First Edition

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/.