RX Family/RA Family
I2C バスインタフェース(RIIC/I3C)による PMBus Master - Slave 通信

要旨

本アプリケーションノートは RX ファミリ RX26T グループ、および RA ファミリ RA6T3 グループを使用した PMBus の通信・制御方法について説明することを目的としています。PMBus 通信・制御ソフトウェアは PMBus Master ソフトウェアと PMBus Slave ソフトウェアがあります。PMBus Master ソフトウェアは RX26T グループ、PMBus Slave ソフトウェアは RX26T グループ、および RA6T3 グループで動作し、Master - Slave の両デバイス間で PMBus 通信を行います。

本アプリケーションノート対象ソフトウェアはあくまで参考用途であり、弊社がこの動作を保証するものではありません。本アプリケーションノート対象ソフトウェアを使用する場合、適切な環境で十分な評価をしたうえで御使用ください。

■ 動作確認デバイス

本アプリケーションノート対象ソフトウェアの動作確認は下記のデバイスで行っています。

- PMBus Master
  RX family RX26T Group (R5F526TFCDFP)

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

なお、本アプリケーションノートで説明するリソースまたは同等の周辺機能を搭載する RX ファミリ、および RA ファミリにも適用できます。(RX72T、RX66T、RX24T、RX23T、RX13T、RX72M、RX72N、RX66N、RA6T1、RA6T2、RA4T1 etc.)

■ 対象サンプルプログラム

本アプリケーションノートの対象サンプルプログラムを下記に示します。

- 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)

● 参考資料

- RX26T グループ ユーザーズマニュアル ハードウェア編 (R01UH0979)
- RA6T3 グループ ユーザーズマニュアル ハードウェア編 (R01UH0998)
- MCK-RX26T ユーザーズマニュアル
- MCK-RA6T3 ユーザーズマニュアル
- RX ファミリ 永久磁石同期モータのセンサレスベクトル制御 - MCK 用
- 永久磁石同期モータのセンサレスベクトル制御 Renesas Flexible Motor Control シリーズ用

目次

1.　概要

本アプリケーションノートは RX ファミリ、および RA ファミリに搭載する I2C バスインターフェース(RIIC/I3C)*1 を使用した PMBus 通信・制御方法の提供を目的としています。PMBus Master 側に RX26T グループを、PMBus Slave 側に RX26T グループ、または RA6T3 グループを接続し、Master - Slave の両デバイス間で PMBus による送受信を行います。

本アプリケーションノートでは PC から発行する PMBus のコマンドを PMBus Master の RX26T グループで解析・処理した後、100kbps の通信速度で PMBus Master から PMBus Slave に接続する永久磁石同期モータをベクトル制御方式で制御・モニタリングします。

なお、PMBus は電源システムで幅広く活用される通信方式ですが、本アプリケーションノートでは電源システムの代替として以下に示すモータシステムの Flexible Motor Control Kit を使用しています。

*1:RIIC/I3C は SMBus (Ver.2.0) に準拠した通信をサポートしています。

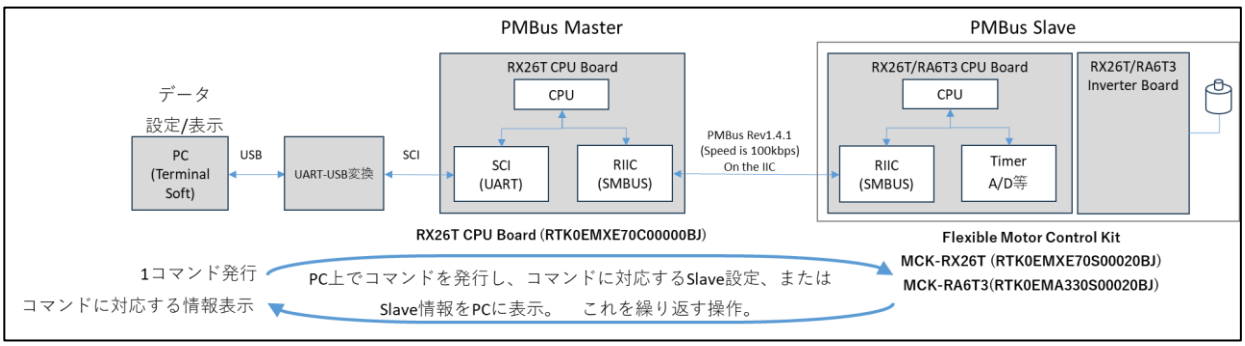


図 1 本アプリケーションノートのシステム構成と動作概要

1.1　開発環境

本アプリケーションノート対象のハードウェア開発環境を表 1 に、ソフトウェアの開発環境を表 2 に示します。

表 1 ハードウェア開発環境

Master/Slave	マイコン	ボード名	型式
Master	RX26T (R5F526TFCDFP)	RX26T CPU Board	RTK0EMXE70C00000B
	-	USB-UART 変換モジュール	Pmod-USBUART (DIGILENT 製)
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
	-	モータ (Included with RTK0EMXE70S00020BJ or RTK0EMXE70S00020BJ)	R42BLD30L3 (MOONS' 製)

表 2 ソフトウェア開発環境

マイコン	IDE バージョン	RX スマート・コンフィグレータ	FSP	ツールチェーンバージョン[注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

【注】 1. プロジェクトで指定するツールチェーン(C コンパイラ）と同一のバージョンがインポート指定先に存在しない場合は、ツールチェーンが選択されない状態になり、エラーが発生します。プロジェクトの設定画面でツールチェーンの選択状態を確認してください。
選択方法は、FAQ 3000404 をご参照ください。
(https://ja-support.renesas.com/knowledgeBase/18367361)

2.　PMBus 概要

PMBus(Power Management Bus)は電力変換装置向けの標準的な通信規格で、I2C から派生したプロトコルの SMBus 通信規格に基づく通信方式です。PMBus はサーバ、データセンサー、および通信機器などに用いられており、電源のモニタリング、電源設定などの目的に使用されています。PMBus は SMBus をベースにした 2 線式(クロックとデータの 2 線)の同期通信で複数のスレーブと通信出来る特徴があり、主に産業機器向けに採用が進んでいます。PMBus により電源システムのコンポーネント間通信がより簡単になり、コンポーネントの設定、制御、およびモニタリングが可能となります。図 2 に PMBus を使用したシステム構成例を示します。

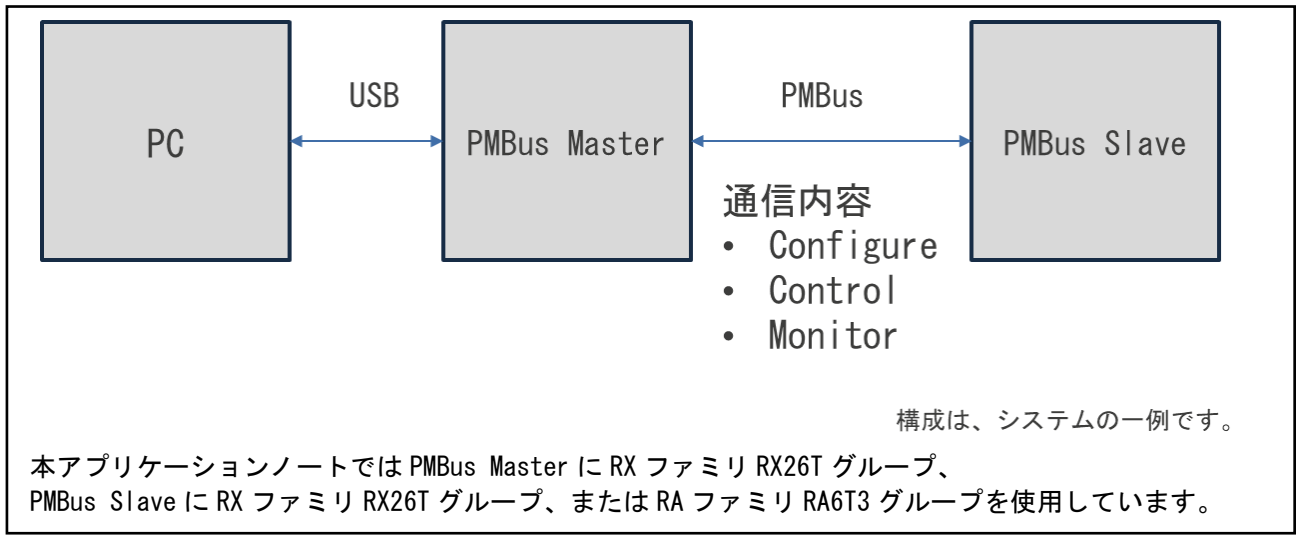


図 2 PMBus システムの構成例

2.1　PMBus プロトコル

PMBus のプロトコルフォーマットは SMBus に準拠したトランザクションプロトコルを使用しています。トランザクションプロトコルは Send Byte, Write Byte, Write Word, Block Write, Receive Byte, Read Byte, Read Word, Block Read で構成しています。各々トランザクションプロトコルを図 3～図 6 に示します。また、PMBus のコマンドは分類すると、標準コマンドプロトコル、グループコマンドプロトコル、ゾーンコマンドプロトコル、拡張コマンドプロトコル、およびバスマスタプロトコルに分類されます。何れも SMBus をベースにしたトランザクションからなるコマンドプロトコルです。分類した 5 つのコマンドプロトコルの内、グループコマンドプロトコル、ゾーンコマンドプロトコル、拡張コマンドプロトコルは SMBus プロトコルを拡張した PMBus 専用のプロトコルです。本アプリケーションノートでは標準コマンドによるスレーブ機器の制御、および監視をしています。その他は参考としてプロトコル処理のみ実装をしていますので、お客様の使用目的に合わせてご確認お願いいたします。
図 7 に標準コマンドプロトコルフォーマット、図 8 にグループコマンドプロトコル、図 9、図 10 に拡張コマンドプロトコル、図 11 にゾーンコマンドプロトコルを示します。

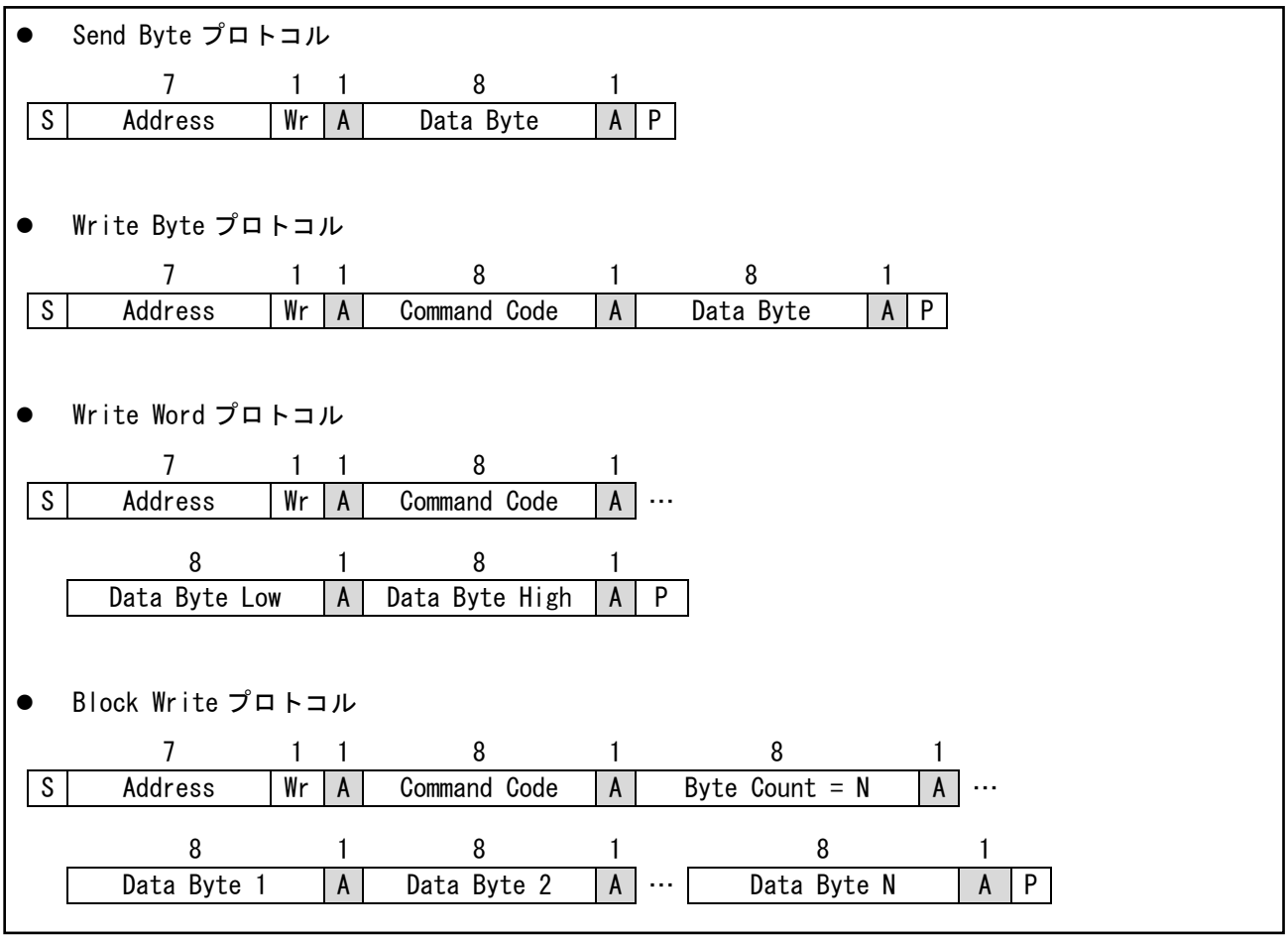
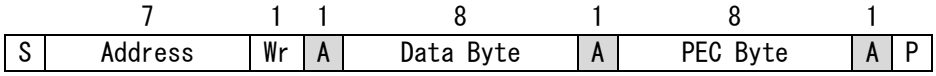
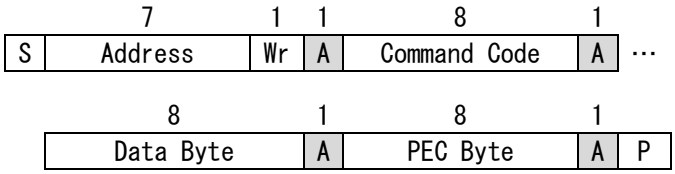


図 3 トランザクション Write 系プロトコル

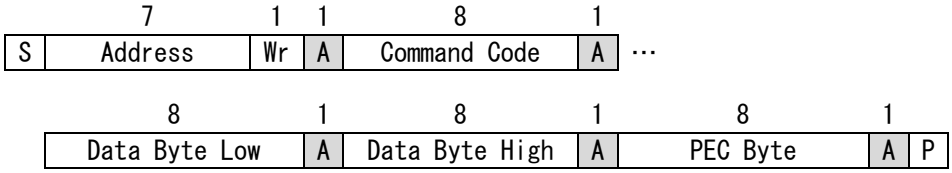
- PEC 付き Send Byte プロトコル



- PEC 付き Write Byte プロトコル



- PEC 付き Write Word プロトコル



- PEC 付き Block Write プロトコル

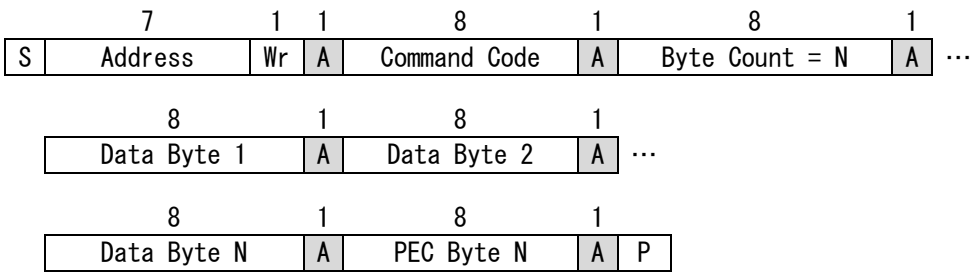
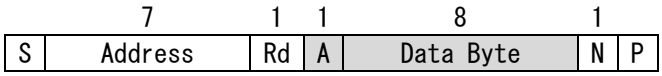
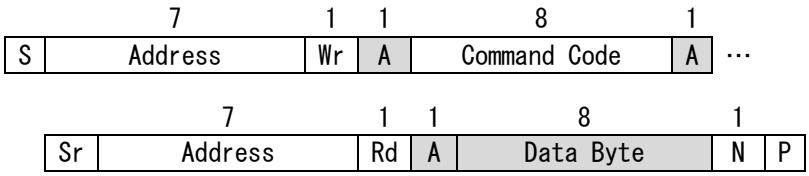


図 4 トランザクション PEC 付き Write 系プロトコル

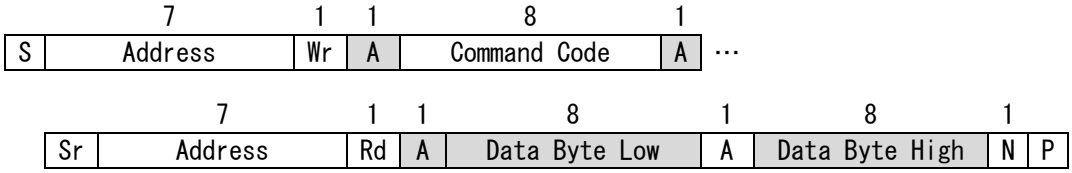
- Receive Byte プロトコル



- Read Byte プロトコル



- Read Word プロトコル



- Block Read プロトコル

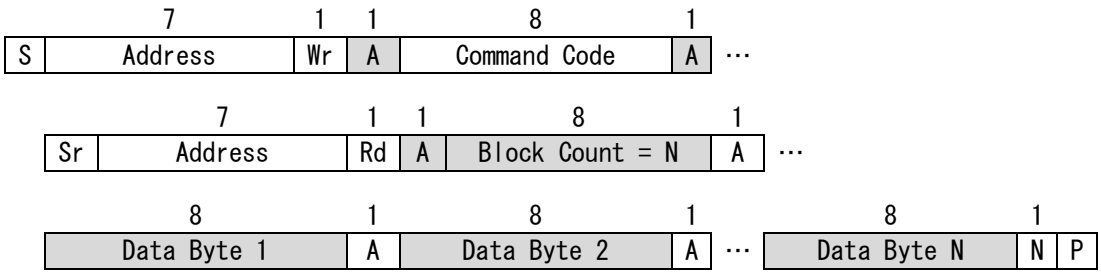
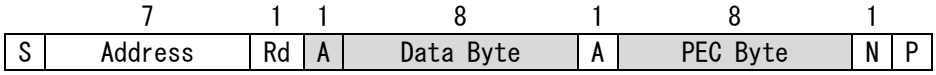
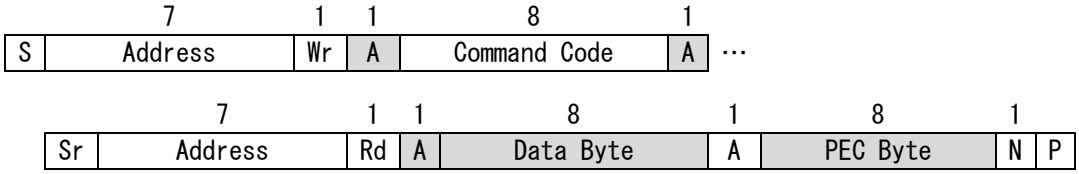


図 5 トランザクション Read 系プロトコル

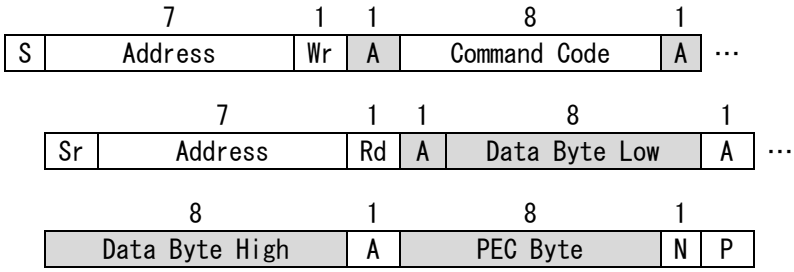
● PEC 付き Receive Byte プロトコル



● PEC 付き Read Byte プロトコル



● PEC 付き Read Word プロトコル



● PEC 付き Block Read プロトコル

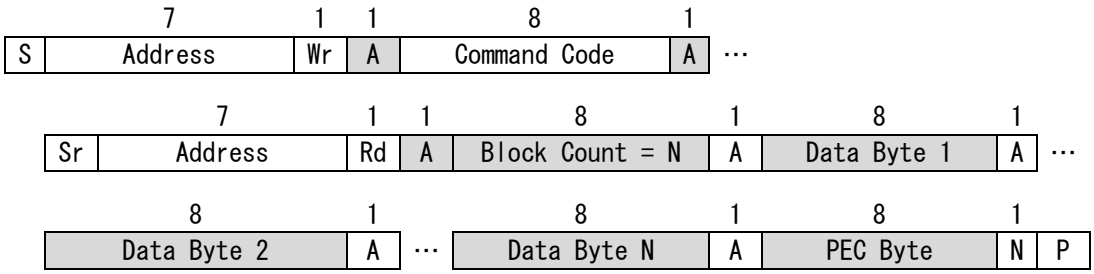
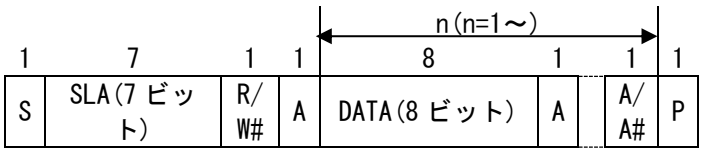


図 6 トランザクション PEC 付き Read 系プロトコル

● 7 ビットアドレスフォーマット



n:転送データバイト数

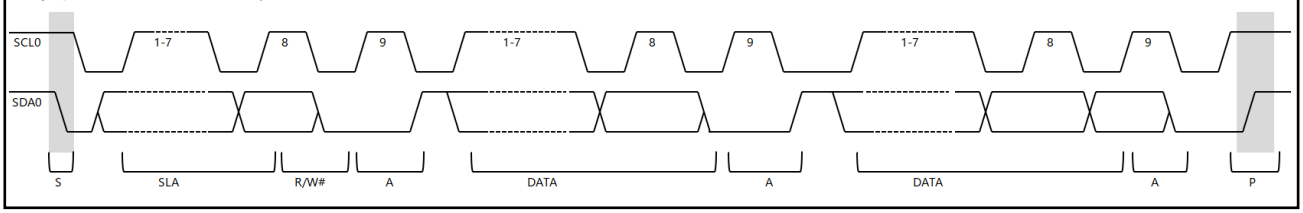
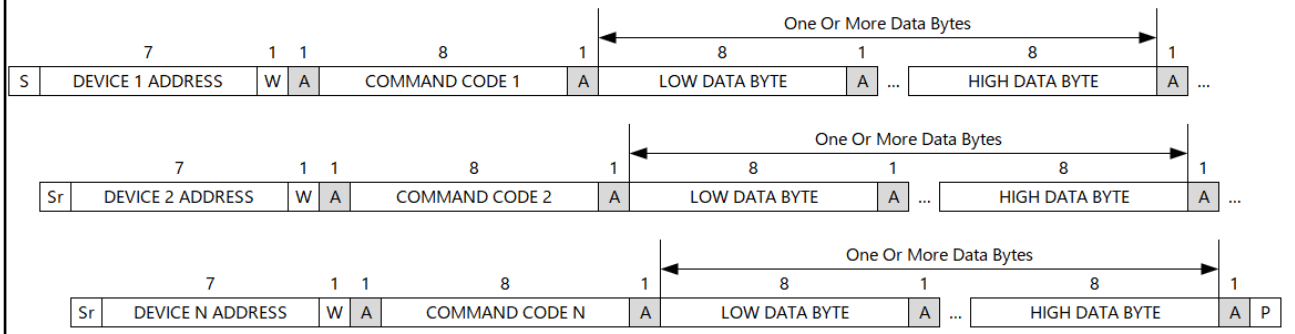


図 7 標準コマンドプロトコル

- グループコマンドプロトコル



- PEC 付きグループコマンドプロトコル

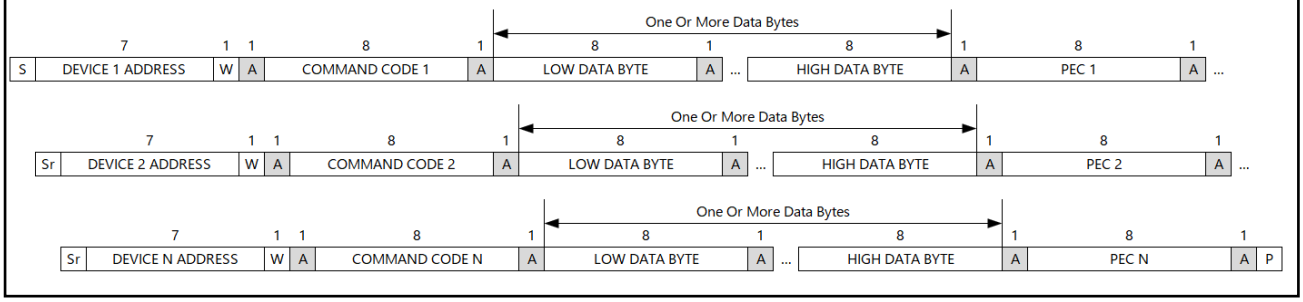
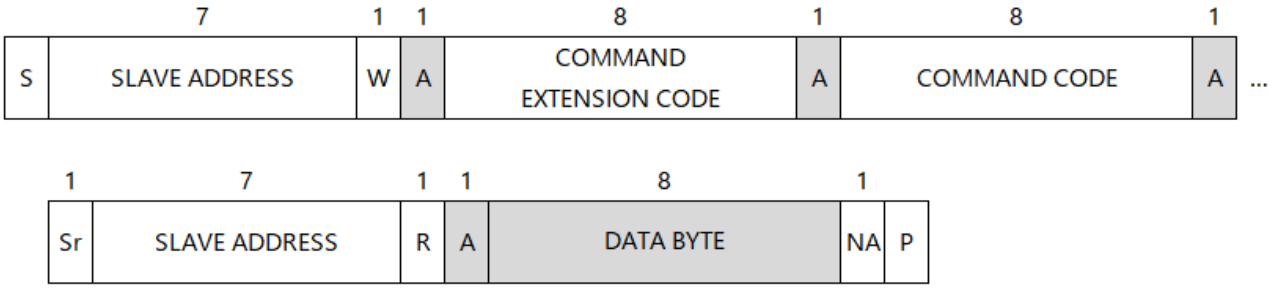
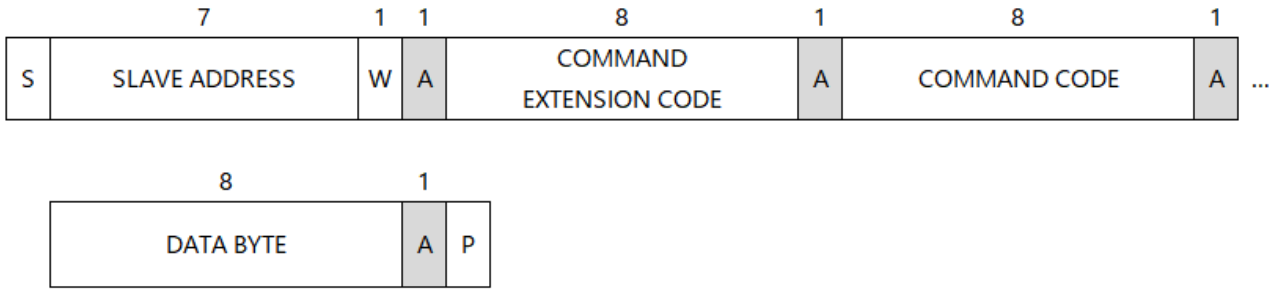


図 8 グループコマンドプロトコル

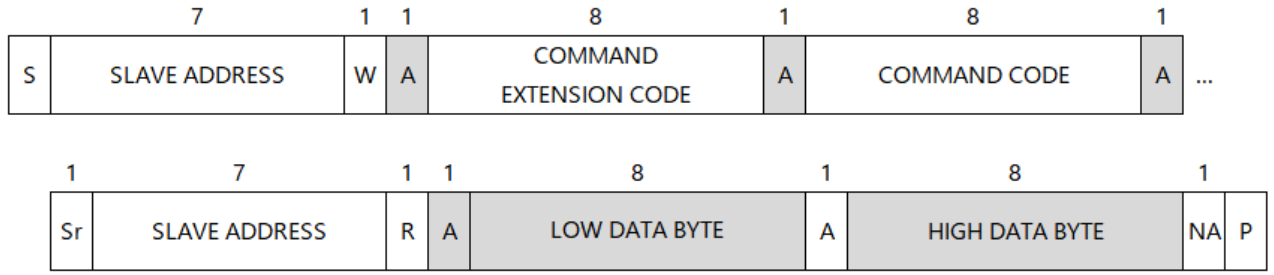
- 拡張コマンド Read Byte プロトコル



- 拡張コマンド Write Byte プロトコル



- 拡張コマンド Read Word プロトコル



- 拡張コマンド Write Word プロトコル

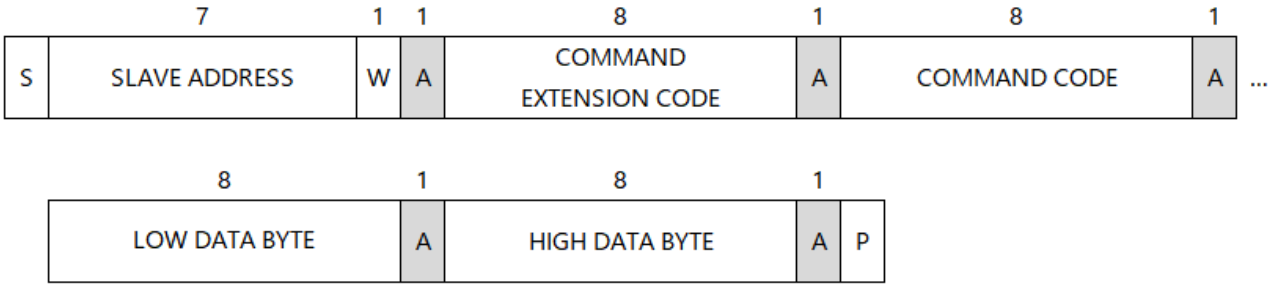
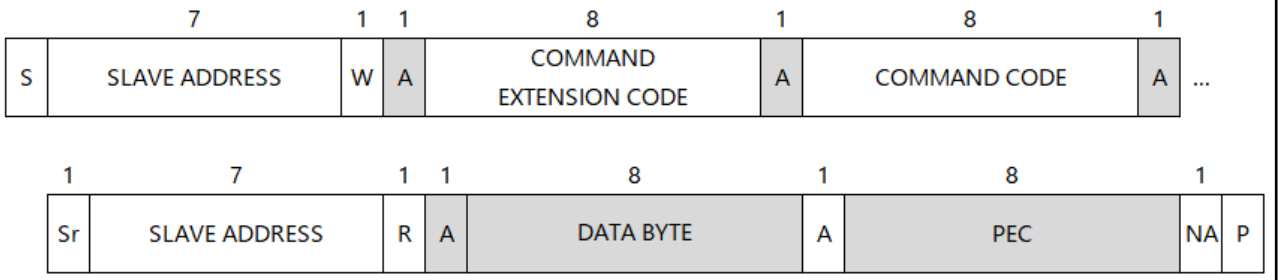
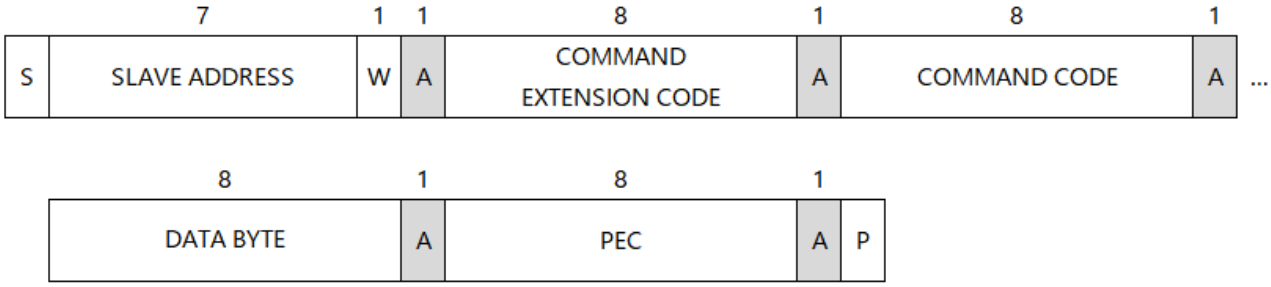


図 9 拡張コマンドプロトコル

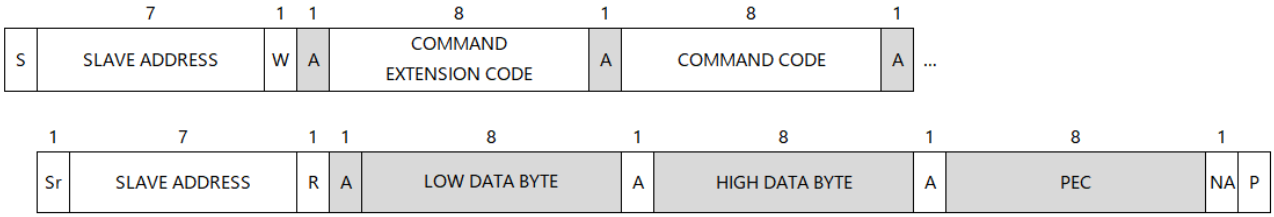
- PEC 付き拡張コマンド Read Byte プロトコル



- PEC 付き拡張コマンド Write Byte プロトコル



- PEC 付き拡張コマンド Read Word プロトコル



- PEC 付き拡張コマンド Write Word プロトコル

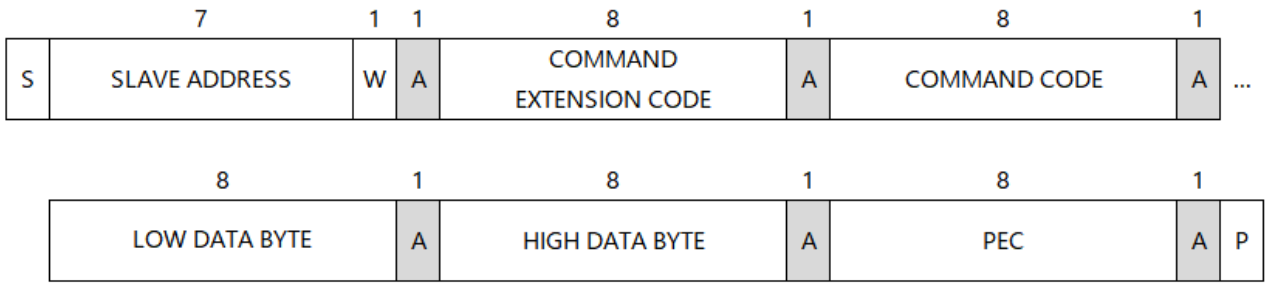
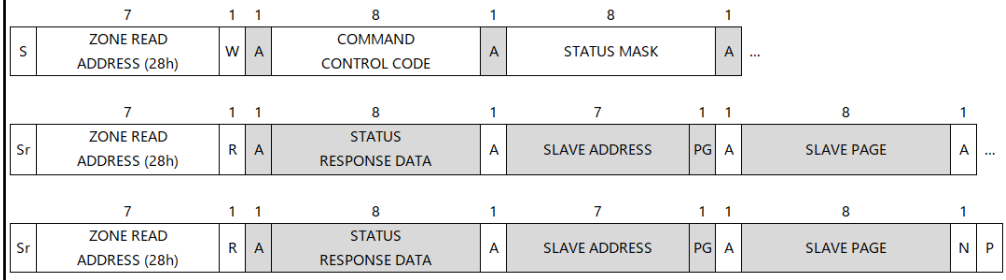
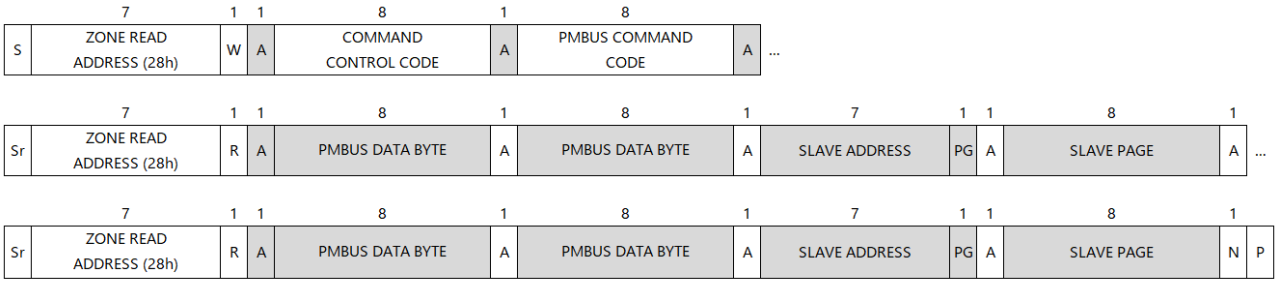


図 10 PEC 付き拡張コマンドプロトコル

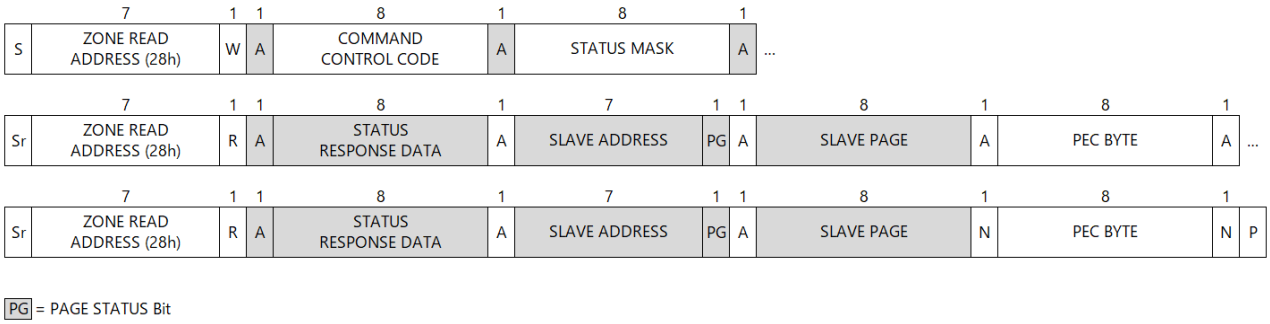
- ステータスレスポンスデータ付きゾーンリードコマンドプロトコル



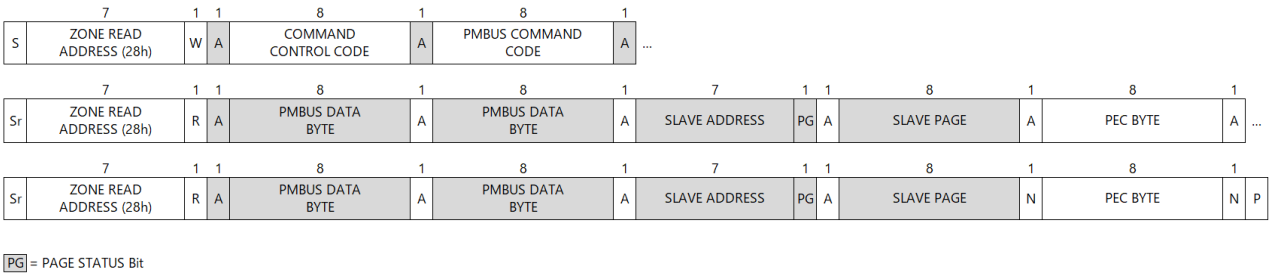
- PMBus コマンドコード付きゾーンリードコマンドプロトコル



- PEC、ステータスレスポンスデータ付きゾーンリードコマンドプロトコル



- PEC、PMBus コマンドコード付きゾーンリードコマンドプロトコル



- 2 バイトのデータ、PMBus コマンドコード付きゾーンライトコマンドプロトコル

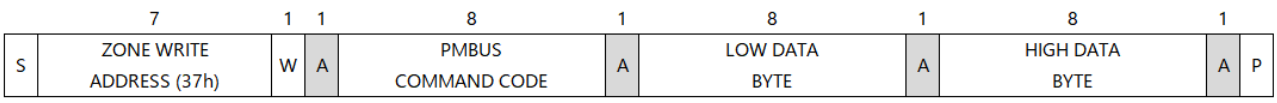


図 11 ゾーンコマンドプロトコル

2.2　PMBus コマンド

PMBus は機器間で相互接続を容易にする為に予め決められたコマンドが準備されています。コマンドには将来の拡張用、およびユーザ定義で使用するコマンドも準備されていますのでカスタマイズが容易になっています。表 3 に PMBus コマンド一覧とコマンドに対応するトランザクションタイプを示します。
本アプリケーションノートでは表 3 に示すコマンドの内、コマンドコード 01h、02h、03h、81h、8Bh、8Ch、90h、95h を使用しています。

表 3 PMBus コマンド一覧

：本アプリケーションノートで使用するコマンド

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.　ハードウェア説明

本アプリケーションノートで使用するボード、および部品を表 4 に示します。

表 4 使用ボード、および各種部品一覧

No.	ボード・部品名称	型名	Master/ Slave	備考
1	CPU Board	RTK0EMXE70C00000BJ	Master & Slave	ルネサス製 MCK-RX26T(RTK0EMXE70S00020BJ) に付属する R5F526TFCDFP 搭載 CPU 評価ボード
		RTK0EMA330C00000BJ	Slave	ルネサス製 MCK-RA6T3(RTK0EMA330S00020BJ) に付属する R7FA6T3BB3CFM 搭載 CPU 評価ボード
2	Inverter Board	RTK0EM0000B12020BJ	Slave	ルネサス製 MCK-RX26T(RTK0EMXE70S00020BJ)、 もしくは MCK-RA6T3(RTK0EMA330S00020BJ) に付属するモータ駆動評価用インバータボード
3	USB-UART 変換ボード	Pmod-USBUART	Master	PC の USB と MCU の SCI を接続する DIGILENT 製 USB-UART 変換ボード
4	Motor	R42BLD30L3	Slave	ルネサス製 MCK-RX26T(RTK0EMXE70S00020BJ)、 もしくは MCK-RA6T3(RTK0EMA330S00020BJ) に付属する MOONS’ 製ブラシレス DC モータ(定格 36V, 1.67A)

3.1　ハードウェア構成

本アプリケーションノートで使用するハードウェア構成を図 12 に、各々ボードの外観図と概略仕様を図 13～図 16 に示します。なお、ボード情報については、略式転記していますので、最新の仕様などの詳細については、要旨の参考資料に記載の各種ボードのユーザーズマニュアルをご参照ください。

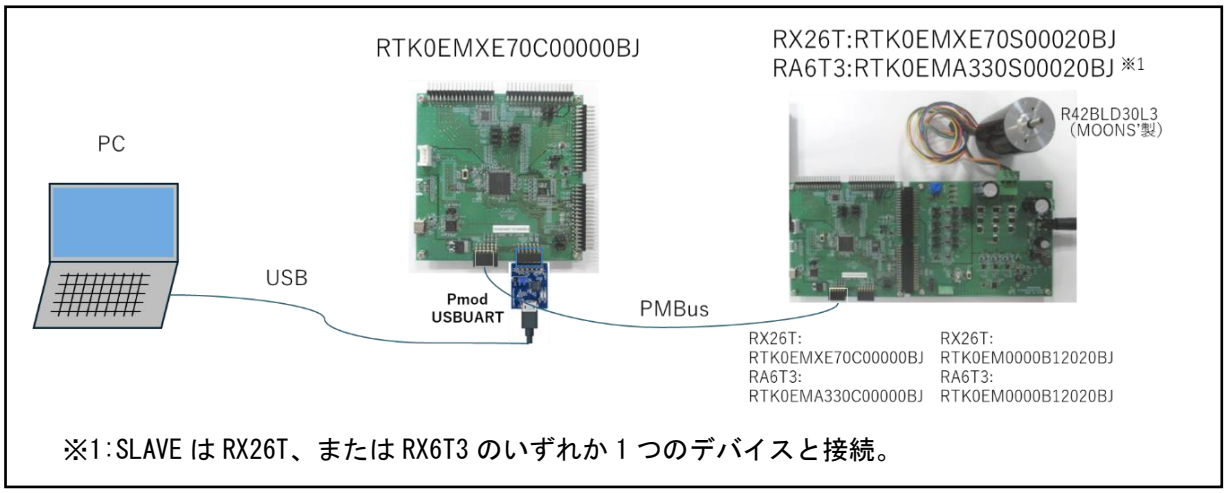


図 12 ハードウェア構成図

項目		仕様
品名		CPU ボード
基板型名		RTK0EMXE70C00000BJ
対応インバータボード		RTK0EM0000B12020BJ
外観		 【注】実物は写真と異なる場合があります。
搭載 MCU	製品グループ	RX26T グループ
	製品型名	R5F526TFCDFP
	CPU 最大動作周波数	120MHz
	ビット数	32 ビット
	パッケージ / ピン数	LFQFP / 100 ピン
	ROM	512K バイト
MCU 入力クロック		10MHz (外部水晶発振子で生成)
電源入力		DC 5V,3.3V(ジャンパーにより切り替え可能) 下記のどちらか一方を選択 • 対応インバータボードからの電源供給 • USB コネクタからの電源供給
デバッガ		E2OB (オンボードデバッガ回路)
コネクタ		• インバータボードコネクタ(2 組) • E2OB 用 USB コネクタ • Renesas Motor Workbench 通信用 SCI コネクタ • CAN 通信用スルーホール • SPI 通信用スルーホール • PMOD モジュール接続用コネクタ
スイッチ		MCU リセット用スイッチ
LED		ユーザ制御用 LED x4、電源 LED x1

図 13 PMBus Master で使用する RX26T CPU Board

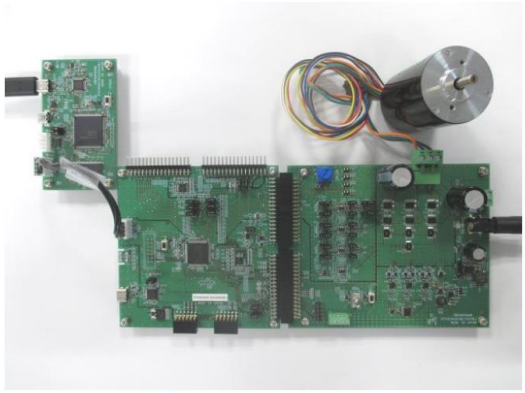
項目	仕様	
製品名	MCK-RX26T	
キット型名	RTK0EMXE70S00020BJ	
キット構成	インバータボード	RTK0EM0000B12020BJ
	CPU ボード	RTK0EMXE70C00000BJ
	通信ボード	RTK0EMXC90Z00000BJ
	ブラシレス DC モータ	R42BLD30L3（MOONS'製） 定格電圧：36[V] 定格電流：1.67[A]
絶縁	インバータボード-CPU ボード間 :非絶縁 通信ボード-CPU ボード間 :絶縁(1kVRMS 以上)	
外観	 【注】実物は写真と異なる場合があります。	
ボード寸法	インバータボード：133mm(幅)×109mm(長さ) CPU ボード： 109mm(幅)×109mm(長さ) 通信ボード：89mm(幅)×52mm(長さ)	
使用温度	常温	
使用湿度	結露なきこと	
EMC 規格	EN61326-1:2021 EMI：Class A EMS：Basic Electromagnetic environment	

図 14 PMBus Slave で使用する RX26T MCU グループ向けフレキシブルモータコントロールキット

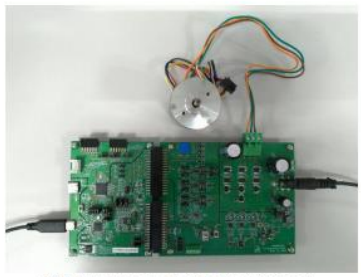
項目	仕様	
製品名	MCK-RA6T3	
キット型名	RTK0EMA330S00020BJ	
キット構成	インバータボード	RTK0EM0000B12020BJ
	CPU ボード	RTK0EMA330C00000BJ
	ブラシレス DC モータ	R42BLD30L3（MOONS'製） 定格電圧：36[V] 定格電流：1.67[A]
絶縁	インバータボード-CPU ボード間 :非絶縁	
外観	 【注】実物は写真と異なる場合があります。	
ボード寸法	インバータボード：133mm(幅)×109mm(長さ) CPU ボード： 85mm(幅)×109mm(長さ)	
使用温度	常温	
使用湿度	結露なきこと	
EMC 規格	EN61326-1:2021 EMI：Class A EMS：Basic Electromagnetic environment	

図 15 PMBus Slave で使用する RA6T3 MCU グループ向けフレキシブルモータコントロールキット

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)

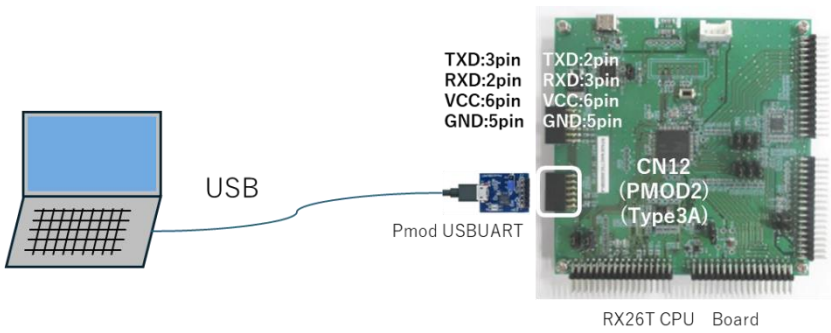
図 16 DIGILENT 社 Pmod USBUART

3.2　ハードウェアセットアップ

本アプリケーションノートの機器接続は図 12 ハードウェア構成図に示す通り、PC と PMBus Master の RX26T CPU Board、および PMBus Master の RX26T CPU Board と PMBus Slave の RX26T CPU Board、もしくは RA6T3 CPU Board を接続します。

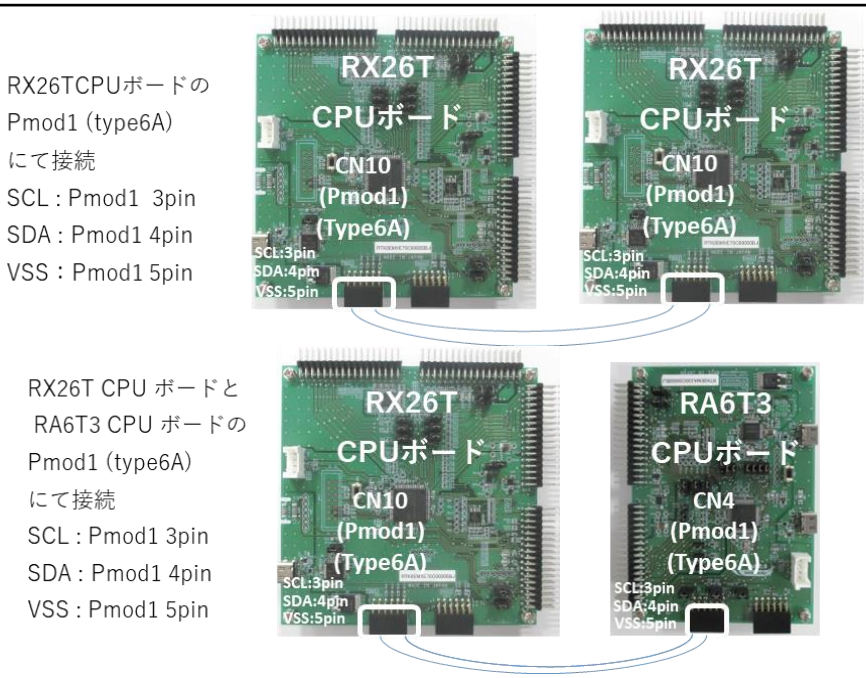
図 17 に機器接続内容を示します。その他モータとの接続など、フレキシブルモータコントロールキットの取り扱い方法については、要旨の参考資料に記載のフレキシブルモータコントロールキットのユーザーズマニュアルをご参照ください。

① PC と PMBus Master の RX26T CPU Board の接続



信号名	Pmod USBUART	RX26T CPU ボード
TXD	3pin	CN12-2pin
RXD	2pin	CN12-3pin
VCC	6pin	CN12-6pin
GND	5pin	CN12-5pin

② PMBus Master の RX26T と、PMBus Slave の RX26T、もしくは RA6T3 の接続



【注】RA6T3 の JP 設定は PMOD (Type 6A)となるように設定してください。

図 17 機器接続図

3.3 MCU 機能接続構成

MCU 機能接続構成を図 18 に示します。

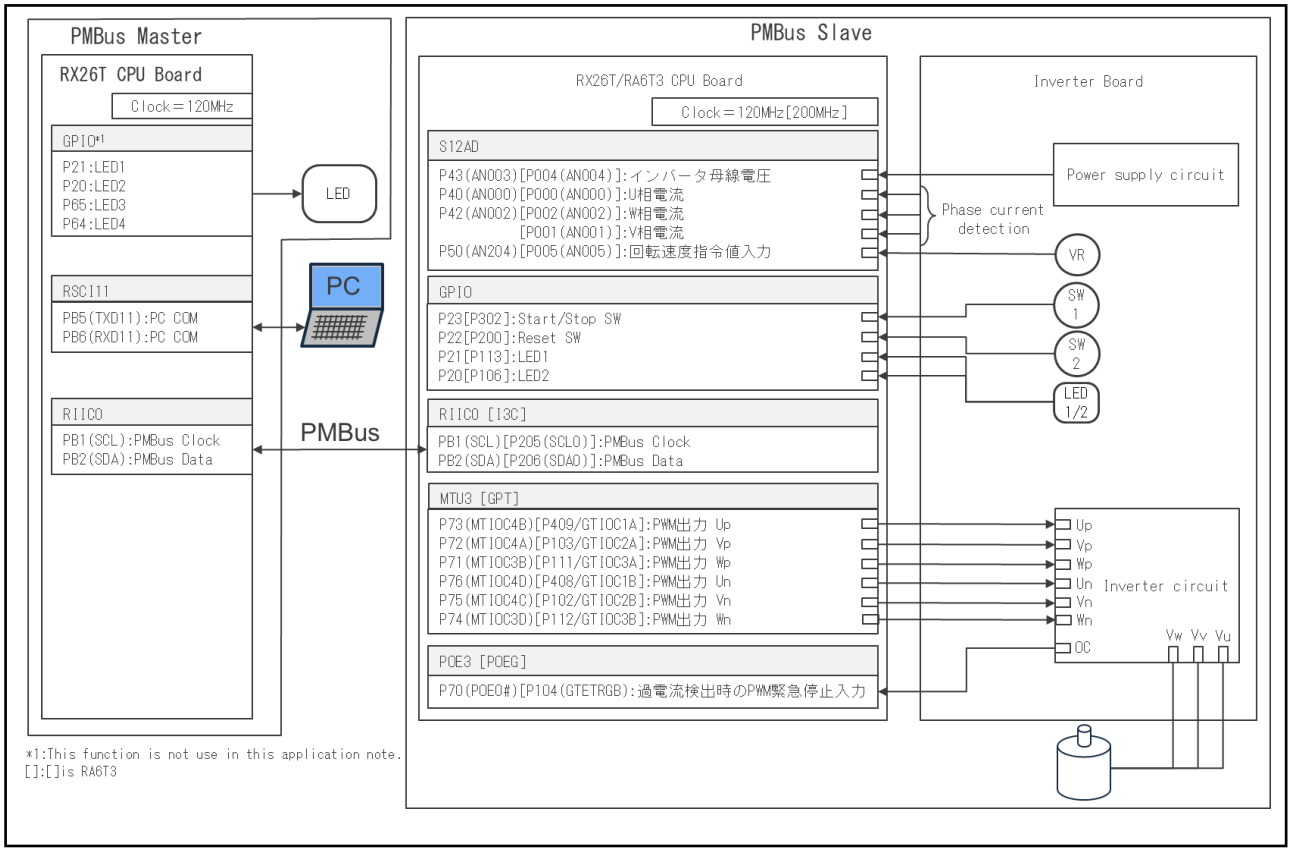


図 18 MCU 機能接続構成図

3.4　MCU 周辺機能

本システムで使用する PMBus Master の RX26T グループ、PMBus Slave の RX26T グループ、および RA6T3 グループの周辺機能一覧を表 5 に示します。

表 5 使用周辺機能一覧表

デバイス	周辺機能	用途
RX26T グループ (PMBus Master)	RSCI11	PC Terminal Soft との通信で使用します。
	RIIC0	PMBus 通信のマスタとして使用します。
	TMR	Unit0 を PMBus 通信のタイムアウト監視で使用します。
RX26T グループ (PMBus Slave)	RIIC0	PMBus 通信のスレーブとして使用します。
	TMR	Unit0 を PMBus 通信のタイムアウト監視で使用します。
	S12AD	以下の機能で使用します。 ・　インバータ母線電圧測定 ・　回転速度指令値入力(アナログ値) ・　U 相電流測定 ・　W 相電流測定
	MTU3	Ch3,4 を以下の機能で使用します。 ・　U 相 PWM 出力 (U_p / U_n) ・　V 相 PWM 出力 (V_p / V_n) ・　W 相 PWM 出力 (W_p / W_n)
	POE3	過電流検出時の PWM 緊急停止入力で使用します。
	CMT	速度制御インターバルタイマ
	IWDT	独立ウォッチドッグタイマ
RA6T3 グループ (PMBus Slave)	I3C	PMBus 通信のスレーブとして使用します。
	GPT	Ch5 を PMBus 通信のタイムアウト監視で使用します。
	ADC12	以下の機能で使用します。 ・　インバータ母線電圧測定 ・　回転速度指令値入力(アナログ値) ・　U 相電流測定 ・　V 相電流測定 ・　W 相電流測定
	GPT	Ch1,2,3 を以下の機能で使用します。 ・　U 相 PWM 出力 (U_p / U_n) ・　V 相 PWM 出力 (V_p / V_n) ・　W 相 PWM 出力 (W_p / W_n)
	POEG	過電流検出時の PWM 緊急停止入力で使用します。
	AGTW	速度制御インターバルタイマ

3.5　端子インタフェース

本システムで使用する PMBus Master の RX26T グループ、PMBus Slave の RX26T グループ、および RA6T3 グループの端子インタフェースを表 6 に示します。

表 6 PMBus Master RX26T, PMBus Slave RX26T,RA6T3 端子インタフェース

デバイス	周辺機能	ポート名	用途
RX26T グループ (PMBus Master)	RSCI11	PB5_TXD11	PC Terminal Soft との通信で使用します。
		PB6_RXD11	
	RIIC0	PB1_SCL	PMBus 通信のマスタとして使用します。
		PB2_SDA	
	GPIO	P21	LED1 と接続されています。
		P20	LED2 と接続されています。
		P65	LED3 と接続されています。
		P64	LED4 と接続されています。
RX26T グループ (PMBus Slave)	RIIC0	PB1_SCL	PMBus 通信のスレーブとして使用します。
		PB2_SDA	
	S12AD	P43/AN003	インバータ母線電圧測定
		P50/AN204	回転速度指令値入力(アナログ値)
		P40/AN000	U 相電流測定
		P42/AN002	W 相電流測定
	MTU3	P73/MTIOC4B	PWM 出力 (U_p) ／"High"アクティブ
		P72/MTIOC4A	PWM 出力 (V_p) ／"High"アクティブ
		P71/MTIOC3B	PWM 出力 (W_p) ／"High"アクティブ
		P76/MTIOC4D	PWM 出力 (U_n) ／"High"アクティブ
		P75/MTIOC4C	PWM 出力 (V_n) ／"High"アクティブ
		P74/MTIOC3D	PWM 出力 (W_n) ／"High"アクティブ
	POE3	P70/POE0#	過電流検出時の PWM 緊急停止入力で使用します。
	GPIO	P23	START/STOP トグルスイッチ
		P22	ERROR RESET プッシュスイッチ
		P21	LED1 制御
		P20	LED2 制御
RA6T3 グループ (PMBus Slave)	I3C	P205_SCL0	PMBus 通信のスレーブとして使用します。
		P206_SDA0_C	
	ADC12	P000/AN000	U 相電流測定
		P001/AN001	V 相電流測定
		P002/AN002	W相電流測定
		P004/AN004	インバータ母線電圧測定
		P005/AN005	回転速度指令値入力(VR 入力、アナログ値)
	GPT	P409/GTIOC1A	PWM 出力 (U_p) ／"High"アクティブ
		P103/GTIOC2A	PWM 出力 (V_p) ／"High"アクティブ
		P111/GTIOC3A	PWM 出力 (W_p) ／"High"アクティブ
		P408/GTIOC1B	PWM 出力 (U_n) ／"High"アクティブ
		P102/GTIOC2B	PWM 出力 (V_n) ／"High"アクティブ
		P112/GTIOC3B	PWM 出力 (W_n) ／"High"アクティブ
	POEG	P104/GTERGB	過電流検出時の PWM 緊急停止入力

	GPIO	P302	START/STOP トグルスイッチ
		P200	ERROR RESET プッシュスイッチ
		P113	LED1 制御
		P106	LED2 制御

4. 操作手順

本アプリケーションノートのシステムは 1 章に示す通り PC のターミナルソフトにより PMBus コマンドを発行しモータ制御を行うシステムです。motor module サンプルを Board UI で動作する設定で動作させます。3 章に示す通りボードを接続して電源を投入後、以下の手順で操作します。

1. インバータボードの SW1（トグル SW）を ON にします。

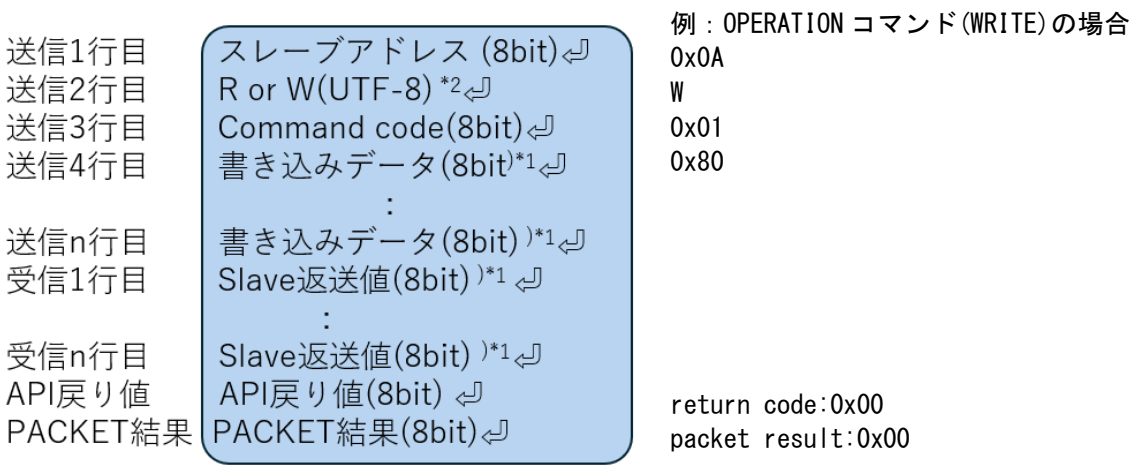
2. インバータボードの VR1 を CW 方向または CCW 方向に回転させます。

3. ターミナルソフトで図 19 に示す入力順序に従い PMBus コマンドのデータを入力します。

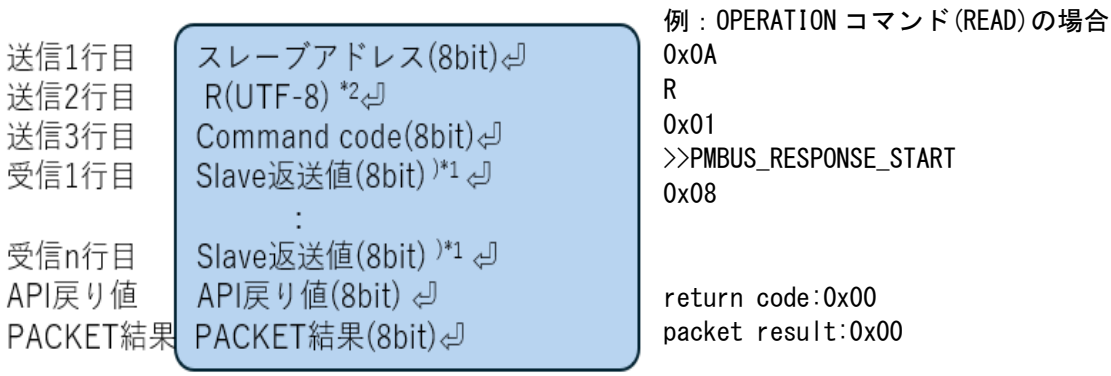
デモシステムでサポートするコマンドの仕様を表 7 に示します。

モータの回転を開始する場合は、上記 1,2 の手順の状態で PMBUS の ON_OFF_CONFIG コマンドと OPERATION コマンドで表 7 に示す設定を送信することで回転開始します。モータの回転を停止させる場合は、ON_OFF_CONFIG と OPERATION による PMBUS コマンドを発行するか、SW1（トグル SW）を OFF するか、または VR1 をセンター（モータ回転最低速）にします。モータを再回転させる場合は、SW1（トグル SW）を ON、および、VR1 をセンター以外の位置に回転させたのち、再度 OPERATION コマンドによる回転開始要求データを受信する必要があります。

- Write,Write/Read コマンド時の PC ターミナルソフトの設定



- Read コマンド時のターミナルソフトの設定



*1:Byte:1 行、word:2 行、BLOCK:n 行。一部のコマンドでは 10 進数で表示します。
　詳細は表 7 をご参照ください。
*2:”R”は Read、”W”は Write 指定を指す。

*:8bit データは"0xAB"のように HEX 表記で入力する。
*:本アプリケーションではスレーブアドレスを 0x0A に固定しています。

図 19 PC ターミナルソフトによるコマンド発行

表 7 デモシステムで使用するコマンド

コマンド名 (コマンドコード)	TRANSACTION CODE	機能説明
OPERATION (0x01)	WRITE_BYTE	ON_OFF_CONFIG で bit3 を"1"に設定している場合のみ、bit7 の値により以下の動作を行います。 bit7=1：モータの回転を開始します。 bit7=0：モータの回転を停止します。
	READ_BYTE	WRITE_BYTE で設定した値を返信します。
ON_OFF_CONFIG (0x02)	WRITE_BYTE	OPERATION コマンドと組み合わせて使用します。 本デモシステムでは bit3 のみを使用します。詳細は OPERATION コマンドの機能説明をご参照ください。
	READ_BYTE	WRAITE_BYTE で設定した値を返信します。
CLEAR_FAULTS (0x03)	SEND_BYTE	モータのエラー情報をクリアします。（RX26T の場合はモータ動作自体にもリセットをかけます。）
STATUS_FANS_1_2 (0x81)	WRITE_BYTE	FAN の状態の内"1"を指定したビットに該当する状態をクリアします。
	READ_BYTE	FAN の状態を返信します。
READ_VOUT (0x8b)	READ_WORD	測定された出力電圧（V）を返信します。PC ターミナルソフトでは整数 10 進数で表示されます。
READ_IOUT(0x8c)	READ_WORD	測定された出力電流（mA）を返信します。PC ターミナルソフトでは符号付整数 10 進数で表示されます。
READ_FAN_SPEED_1 (0x90)	READ_WORD	FAN の速度（rad/s）を返信します。PC ターミナルソフトでは符号付整数 10 進数で表示されます。
READ_FREQENCY (0x95)	READ_WORD	メイン電源コンバータのスイッチング周波数(μsec)を返信します。PC ターミナルソフトでは整数 10 進数で表示されます。

5.　ソフトウェア説明

本アプリケーションノートのソフトウェア処理は、MCU の周辺機能を制御するドライバ部、PMBus を制御する PMBus ミドルウェア部、その PMBus ミドルウェア部を操作するユーザアプリケーションに分かれます。その他、PMBus Master 側は PC と接続する SCI ドライバ部、PMBus Slave 側はモータを制御するモータ制御ミドルウェアがあり、各々ユーザアプリケーションで操作します。PMBus Master のソフトウェアのモジュール構成を図 20 に、PMBus Slave のソフトウェアのモジュール構成を図 21、図 22 に示します。また PMBus Master の状態遷移、および関数動作などの詳細ついては、5.1 節を、PMBus Slave の状態遷移、および関数動作などの詳細については、5.2 節をご参照ください。

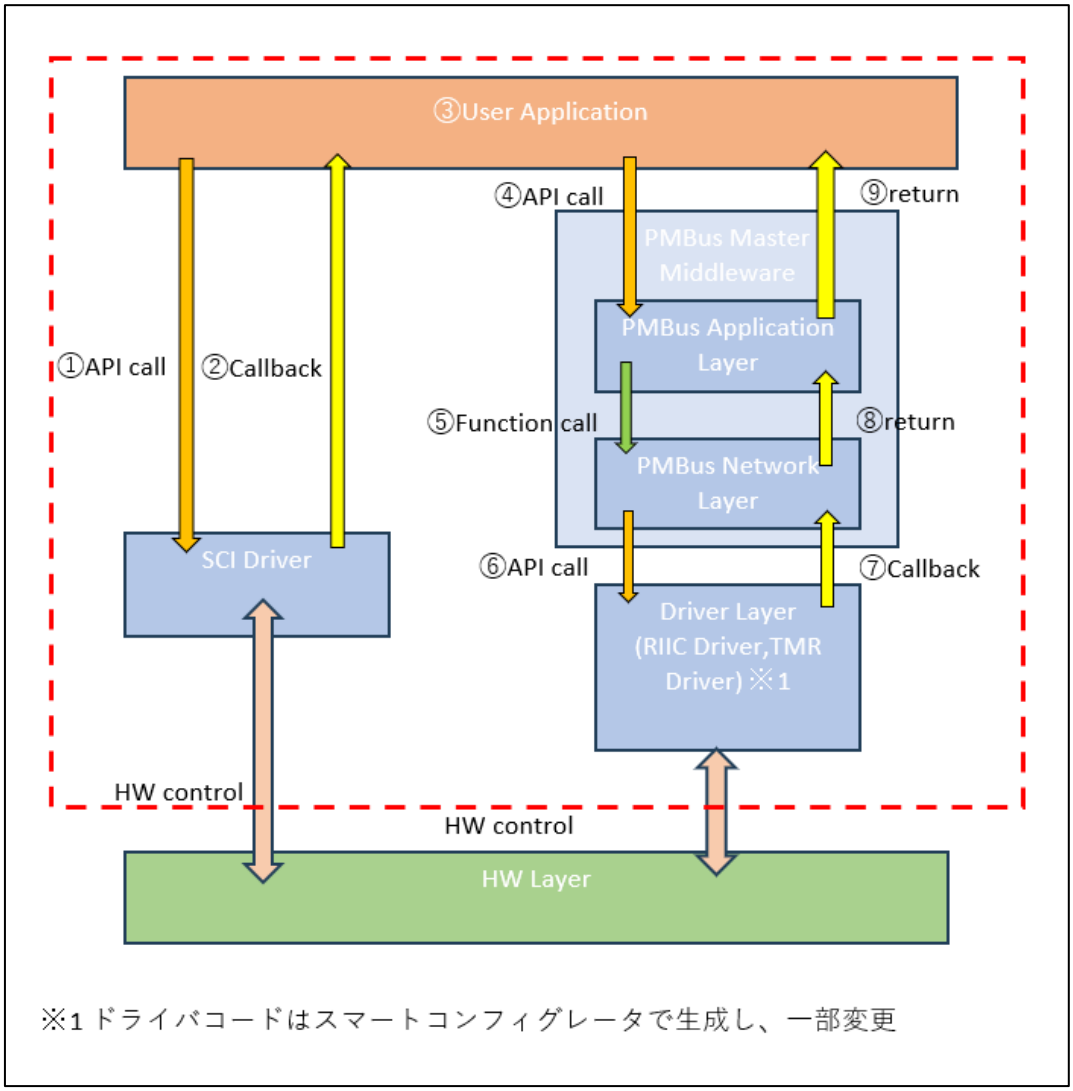


図 20 PMBus Master ソフトウェアモジュール構成　(RX26T グループ)

表 8 PMBus master ソフトウェアの各インタフェースで使用するグローバル変数一覧

IF No は「図 20 PMBus Master ソフトウェアモジュール構成　(RX26T グループ)」の各インタフェースの番号を意味します。

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

【注】本表では本アプリケーションでサポート対象としている PMBUS Master API のみ記載しています。
各グローバル変数の詳細は「5.1.6 PMBus Master グローバル変数一覧」をご参照ください。

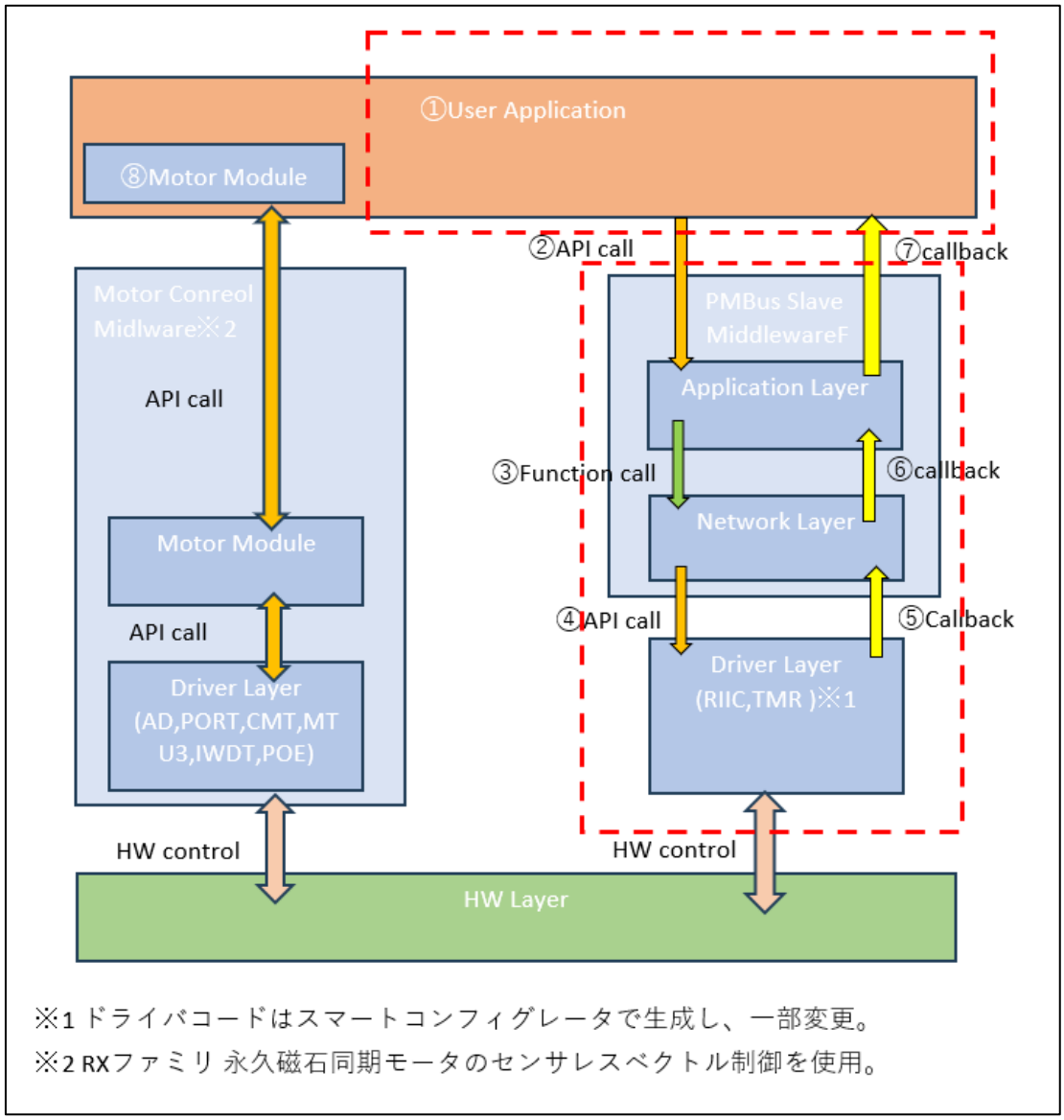


図 21 PMBus Slave ソフトウェアモジュール構成（RX26T グループ）

表 9 PMBus slave (RX26T)ソフトウェアの各インタフェースで使用するグローバル変数一覧

IF No は「図 21 PMBus Slave ソフトウェアモジュール構成　(RX26T グループ)」の各インタフェースの番号を意味します。

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

【注】スレーブの動作を開始するタイミングは Driver Layer からの割り込み通知となるため、本表では代表的なプロトコルで動作した場合と、Open API 時に使用するグローバル変数を記載しています。
各グローバル変数の詳細は「5.2.6 PMBus Slave グローバル変数一覧」をご参照ください。

*1. 本グローバル変数は OPERATION コマンド時、または、ON_OFF_CONFIG コマンド受信時のみ使用します。

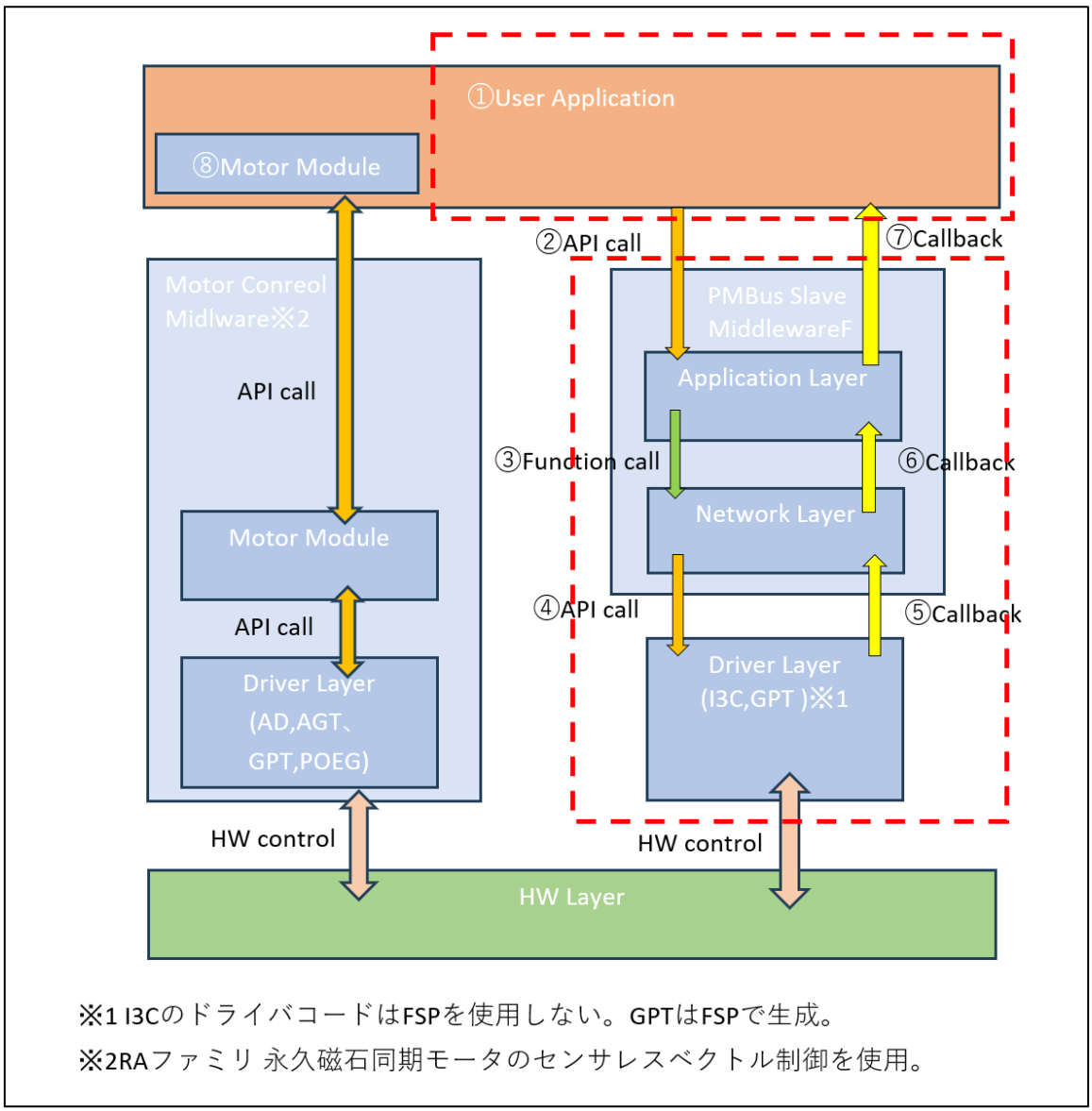


図 22 PMBus Slave ソフトウェアモジュール構成　(RA6T3 グループ)

表 10 PMBus slave (RA6T3)ソフトウェアの各インタフェースで使用するグローバル変数一覧

IF No は「図 22 PMBus Slave ソフトウェアモジュール構成　(RA6T3 グループ)」の各インタフェースの番号を意味します。

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

【注】スレーブの動作を開始するタイミングは Driver Layer からの割り込み通知となるため、本表では代表的なプロトコルで動作した場合と、Open API 時に使用するグローバル変数を記載しています。
各グローバル変数の詳細は「5.2.6 PMBus Slave グローバル変数一覧」をご参照ください。

*1. 本グローバル変数は OPERATION コマンド時、または、ON_OFF_CONFIG コマンド受信時のみ使用します。

5.1　PMBus Master ソフトウェア

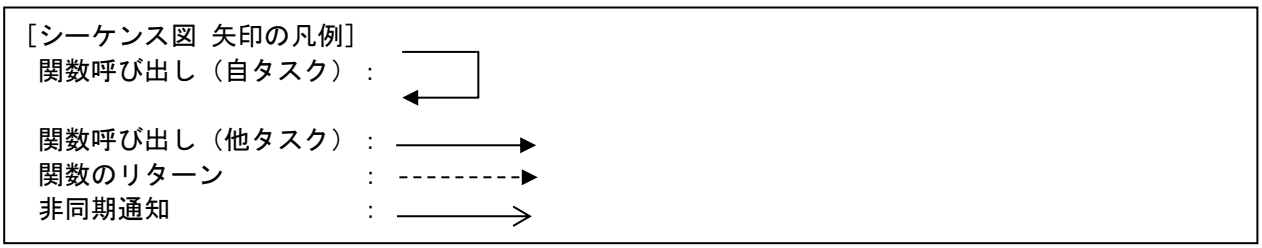
PMBus Master のソフトウェアは、図 20 PMBus Master ソフトウェアモジュール構成に示す通り、ユーザアプリケーション部、ミドルウェア部、ドライバ部に分類されます。ドライバ部はスマート・コンフィグレータで生成するソフトウェアを使用し一部 PMBus Master 動作を行う為の変更、追加をしています。変更、追加内容については、5.1.4 節をご参照ください。各々ソフトウェアのフォルダ・ファイル構成を表 11 に示します。

表 11 PMBus Master RX26T グループフォルダ・ファイル構成

フォルダ名	ファイル名	概要
app¥	r_pmbus_demo_master.c	PMBus デモシステムのメインプログラム。（ユーザアプリケーション）
	r_pmbus_demo_master.h	PMBus デモシステムのメインプログラムで使用するヘッダファイル。
pmbus_master¥	r_pmbus_app_master.c	PMBus Middleware のアプリケーション層。
	r_pmbus_app_master.h	PMBus Middleware のアプリケーション層で使用するヘッダファイル。
	r_pmbus_nwk_master.c	PMBus Middleware のネットワーク層。
	r_pmbus_nwk_master.h	PMBus Middleware のネットワーク層で使用するヘッダファイル。
src¥smc_gen¥Config_RIIC0¥	Config_RIIC0.c	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
	Config_RIIC0.h	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
	Config_RIIC0_user.c	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_RSCI11¥	Config_RSCI11.c	PMBus ユーザアプリケーションのドライバ層。スマート・コンフィグレータで生成する。
	Config_RSCI11.h	PMBus ユーザアプリケーションのドライバ層。スマート・コンフィグレータで生成する。
	Config_RSCI11_user.c	PMBus ユーザアプリケーションのドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_TMR0¥	Config_TMR0.c	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
	Config_TMR0.h	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
	Config_TMR0_user.c	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。

5.1.1　PMBus Master 動作シーケンス

PMBus コマンド発行から動作完了までの動作シーケンスを図 23 に示します。また PMBus コマンドに合わせた Write 動作、Read 動作、Write/Read 動作について、Write 動作シーケンスを図 24 と図 25 に、Read 動作シーケンスと Write/Read 動作シーケンスを図 26 と図 27 に示します。
各々動作で使用する関数は 5.1.3 節の PMBus Master 関数一覧をご参照ください。



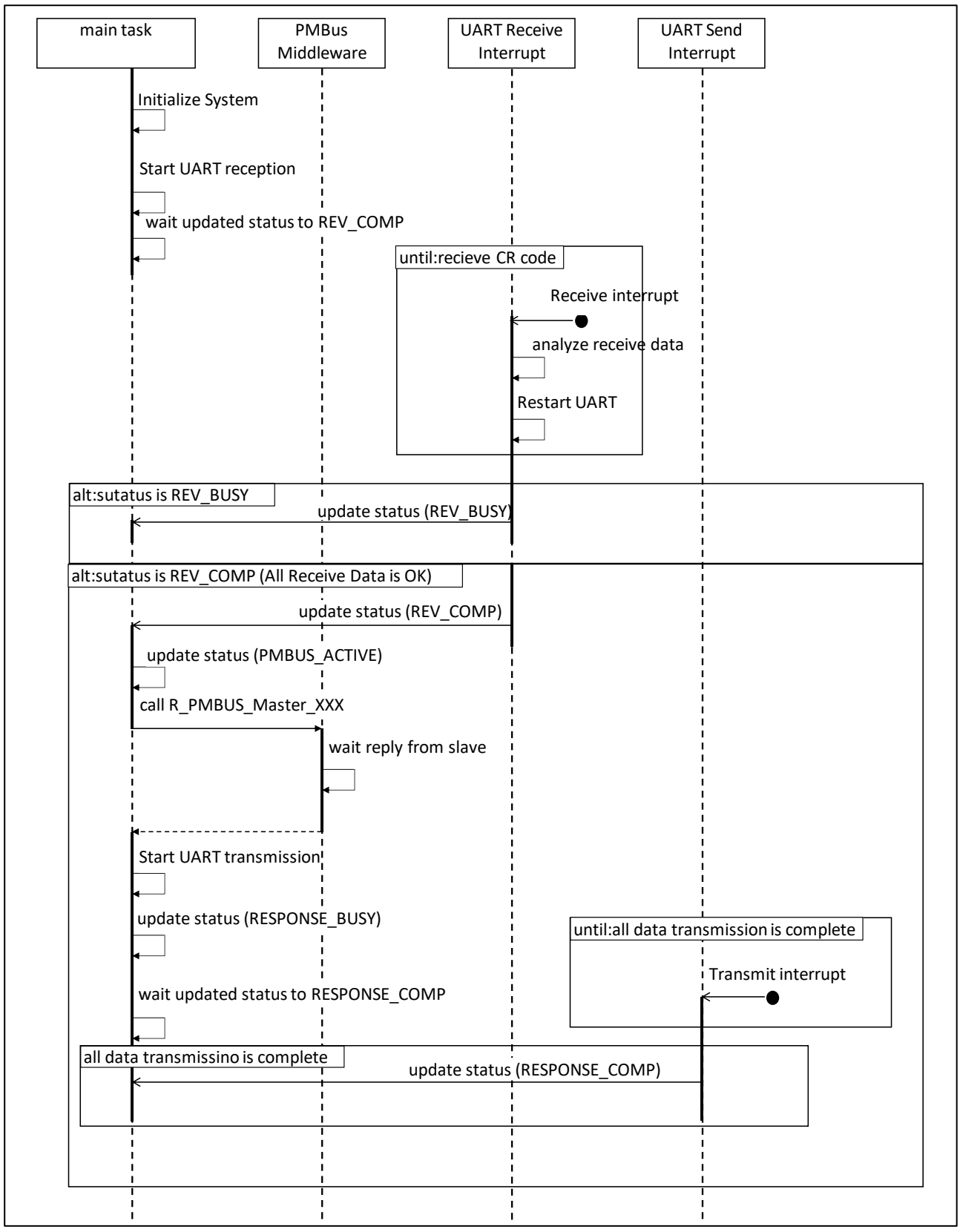


図 23 PMBus Master 動作シーケンス図

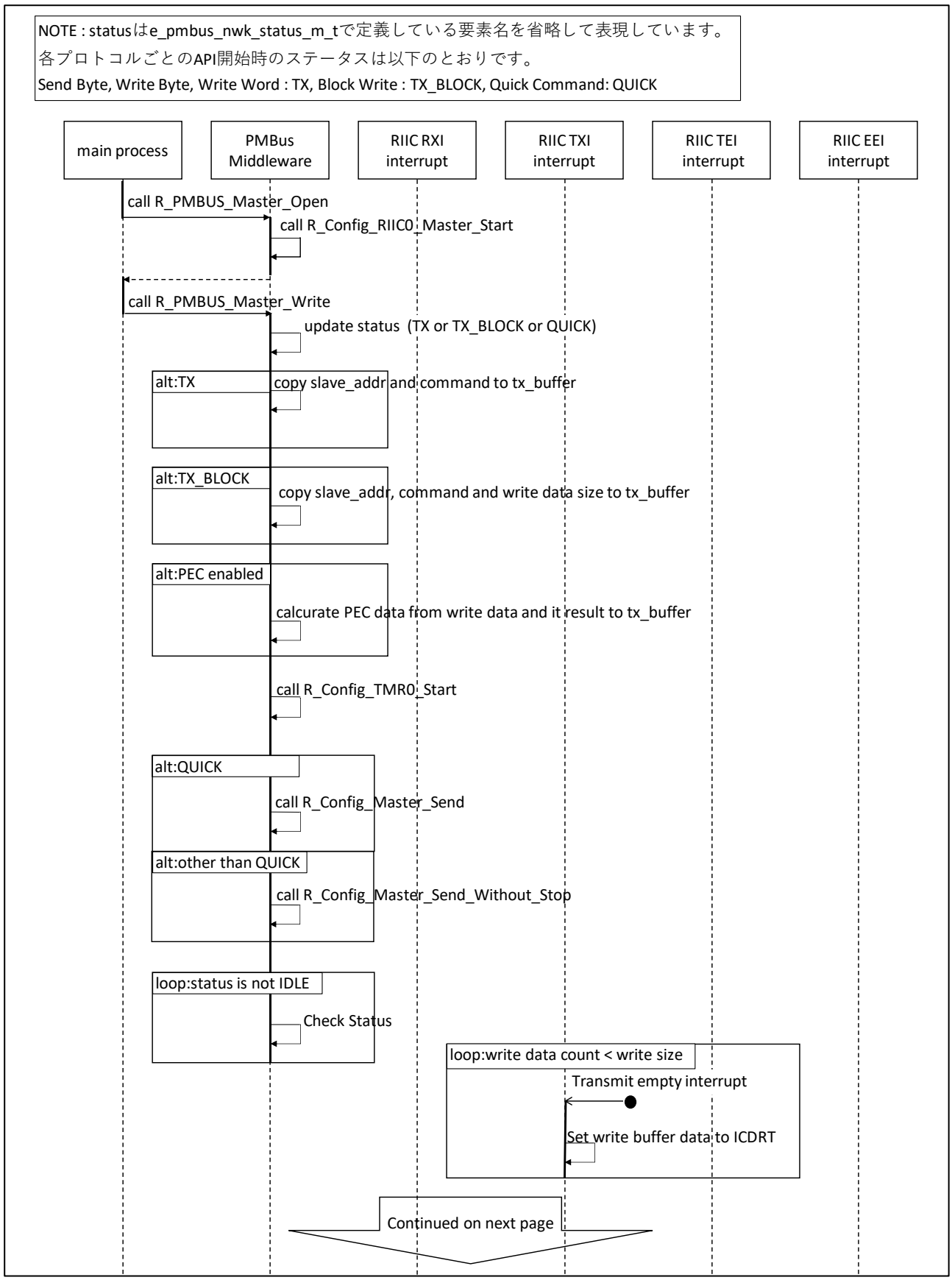


図 24 Write 動作(R_PMBUS_Master_Write)シーケンス図 (1/2)

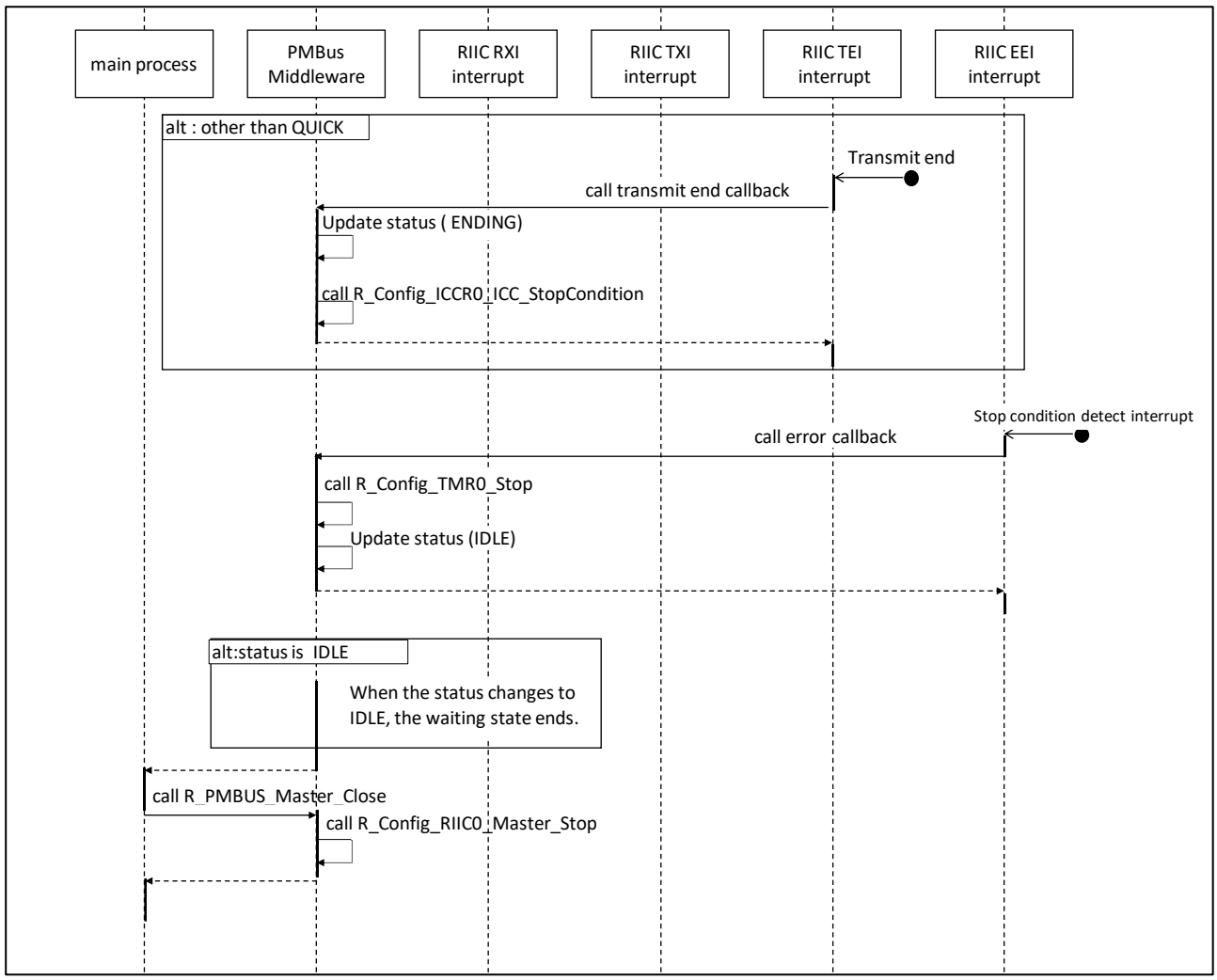


図 25 Write 動作(R_PMBUS_Master_Write)シーケンス図 (2/2)

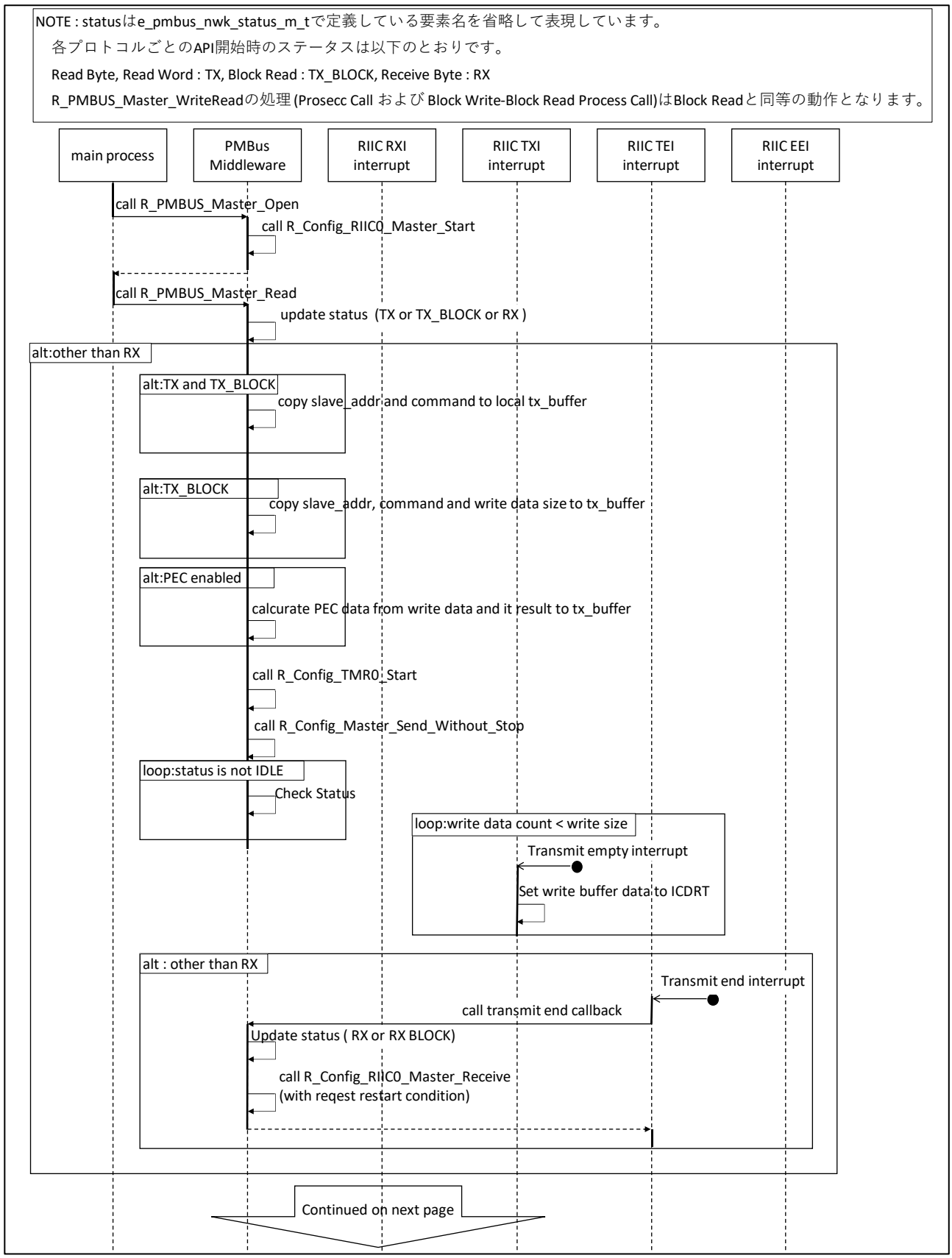


図 26 Read 動作(R_PMBUS_Master_Read),Write/Read 動作(R_PMBUS_Master_WriteRead)のシーケンス図
(1/2)

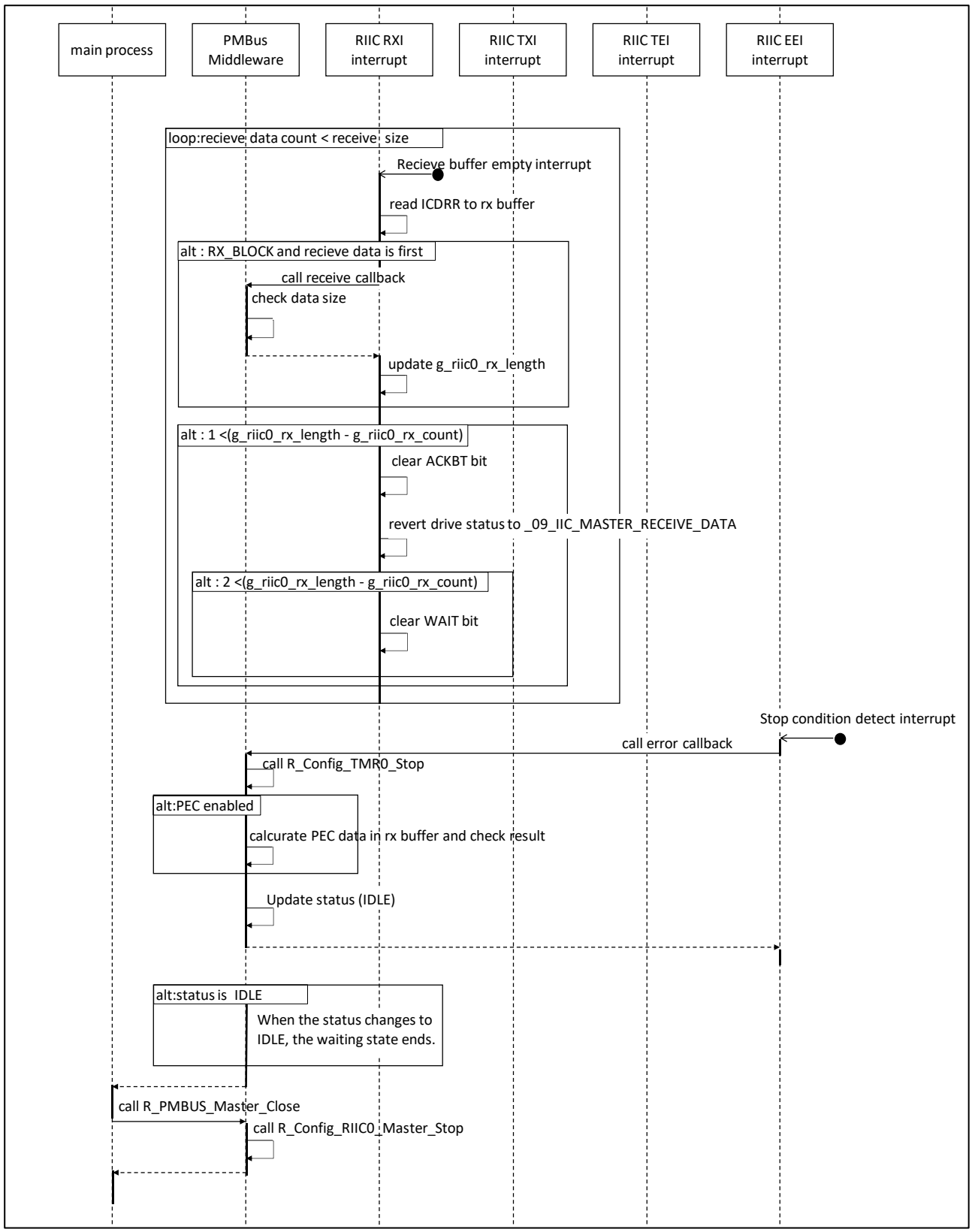


図 27 Read 動作(R_PMBUS_Master_Read), Write/Read 動作(R_PMBUS_Master_WriteRead)のシーケンス図 (2/2)

5.1.2　PMBus Master 状態遷移

PMBus Master ミドルウェアの状態管理は、Application Layer と Driver Layer の各々で状態を管理します。Application Layer はプロトコル状態遷移を、Driver Layer はデータ送受信数を管理します。
Application Layer の状態遷移を 5.1.2.1 項に、Driver Layer の状態遷移を 5.1.2.2 項に示します。

5.1.2.1 PMBus Master Middleware Application Layer 状態遷移

PMBus Master の Application Layer 状態遷移は、コマンドコードに合わせて、送信、受信、Quick コマンド、Block コマンド、およびエラー処理の状態を管理します。以下図 28、図 29、および表 12 に示します。

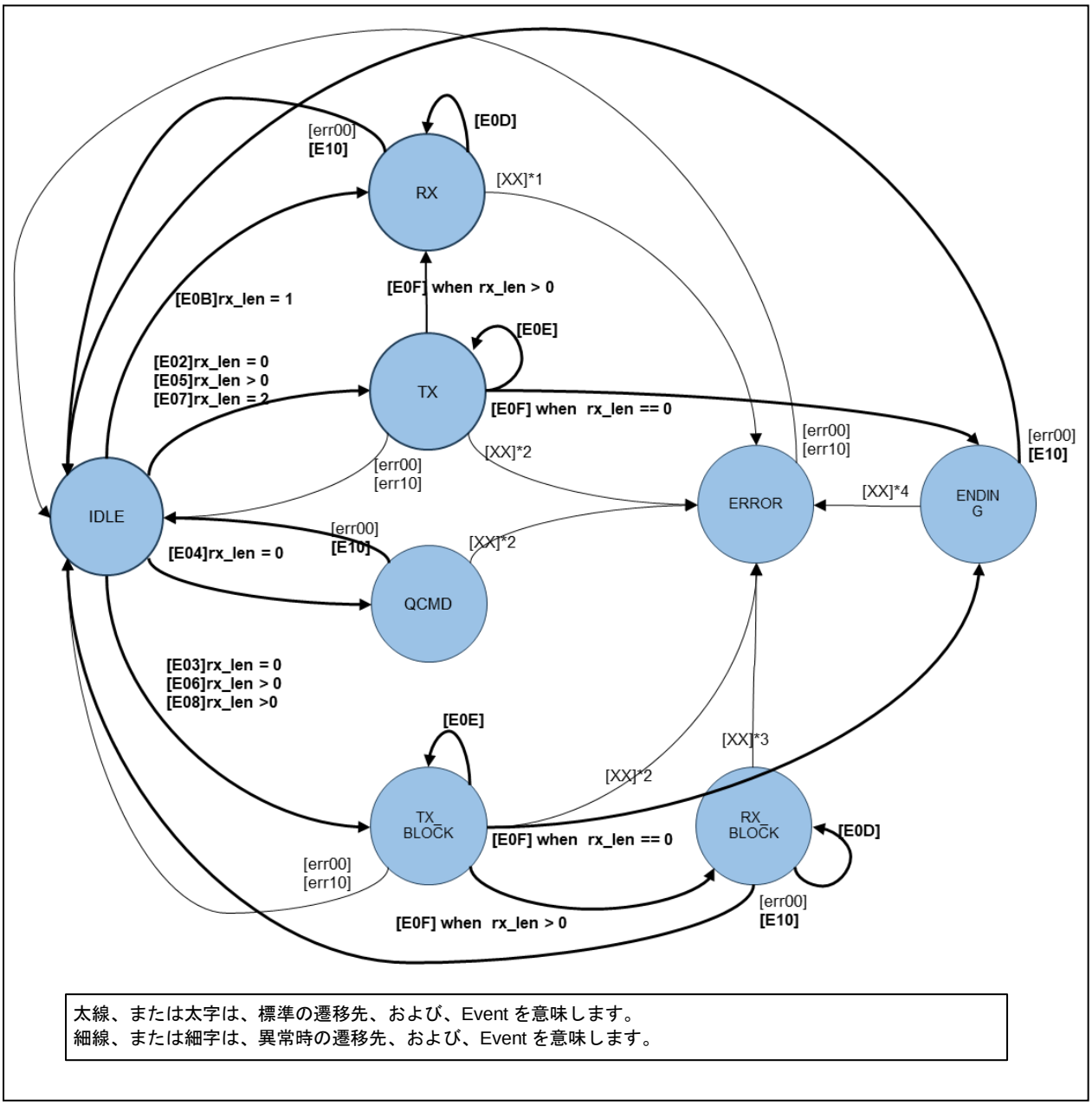


図 28 PMBus Master Application Layer 状態遷移図

[Event 一覧]
※[Exx]は通常の Event、[errXX]は異常発生の場合の Event を意味する。
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 の条件は以下の通り。
[err0D] If (rx_index over rx_len)　,[err0F]　,[err10] If (pec is enabled and pec data is error)
[err11]　,[err12]　,[err13]

*2 の条件は以下の通り。
[err0D]　,[err11]　,[err12]　,[err13]

*3 の条件は以下の通り。
[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 の条件は以下の通り。
[err0D]　,[err0F]　,「err11」　,[err13]

図 29 PMBus Master Application Layer 状態遷移図(図 28 の補足)

表 12 PMBus Master Application Layer の状態遷移表

表の見方は以下の通りとなります。
・event は API 名の省略表現と、割り込み要因で構成します。
・status は"e_pmbus_nwk_status_m_t"の要素名の省略表現で構成します。
・「If (<条件>)」は条件付きで遷移することを意味します。
・「→ <状態>」は状態に遷移することを意味します。
・「ERROR (<エラー名>)」は API の返却値を意味します。
・「PACKET RESULT(<エラー名>)」は、API の引数 p_e_packet_result)に格納するエラー情報を意味します。
・「-」は状態遷移しないことを意味します。
・薄青は図 28 における状態遷移を意味します。

	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_WriteRead	→ TX rx_len = 2	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)	ERROR (BUSY)

E08_WriteRead (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. Stop Condition を発行します。
*2. Close 後に Open した場合のみ IDLE に遷移します。

5.1.2.2 PMBus Master Driver Layer 状態遷移

PMBus Master Driver Layer の状態遷移は、PMBus Slave への送信動作部と受信動作部に分かれ、各々指定のデータを指定のバイト数だけ送受信します。PMBus Master Driver Layer の状態遷移を図 30、図 31、および表 13、 表 14 に示します。

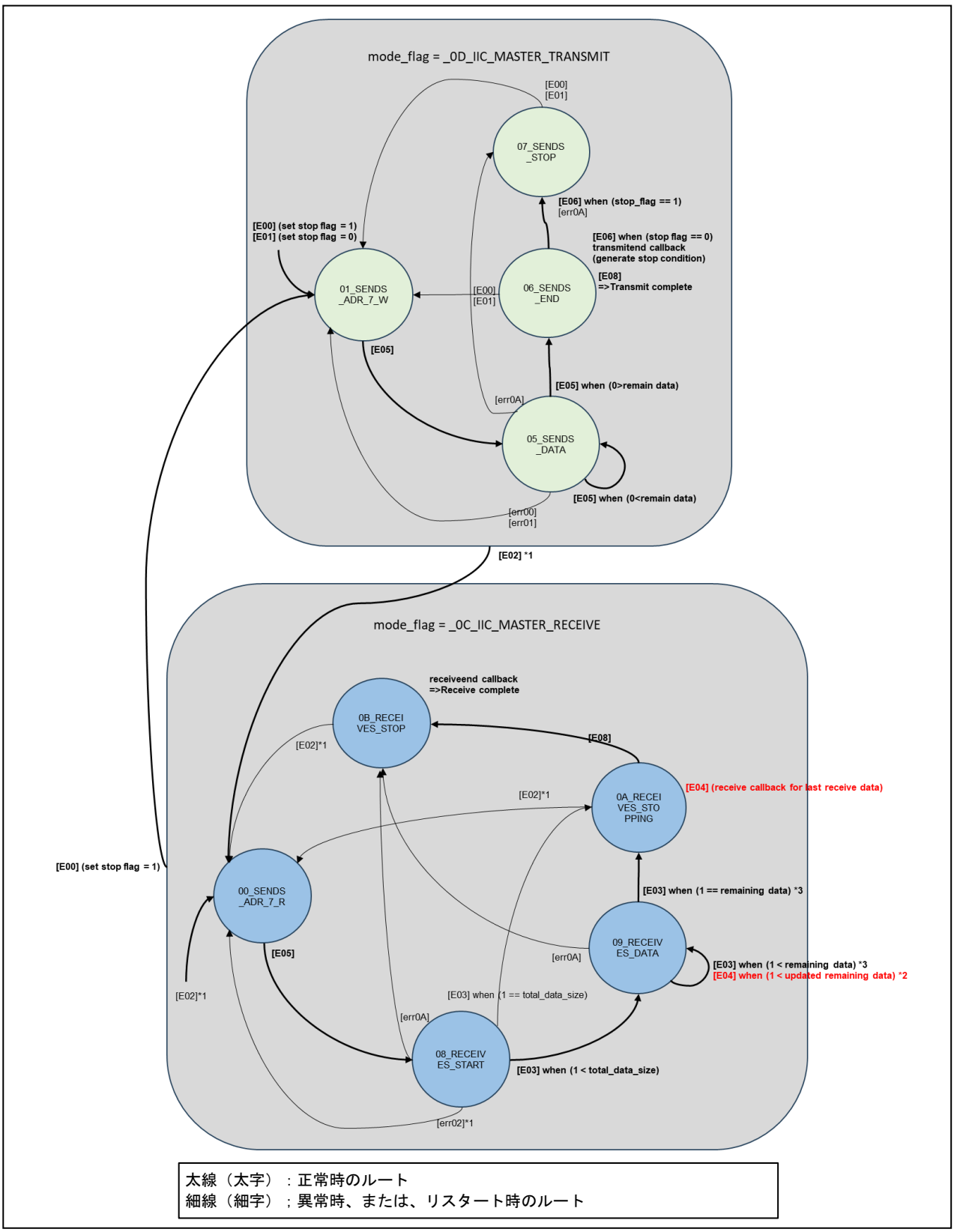


図 30 PMBus Master Driver Layer の状態遷移図

[Event 一覧]
※[Exx]は通常の Event、[errxx]は異常発生の場合の Event を意味する。
一覧にあり図中に無い異常 Event はどの状態でも発生し、callback error によりエラーを通知する。
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.Block Read および Block Write-BlockRead Process Call 時に指定する受信データ数の初期値(total_data_size)は,プロトコルの最小受信数である 3(Data Size(1)+Data(1)+PEC(1))を指定する。それ以外は各プロトコルに合わせてデータ数を指定する。ただし、"0"を指定できない。
*2. callback receive 内で受信データを確認し、Block read の場合は受信予定のデータ数を更新する。
その結果残りの受信データ数が 2 バイト以上となった場合は状態を 09 に戻す。
また、残り受信データ数が 2 バイト以上の場合は ACKBT=0 に設定、3 バイト以上の場合は WAIT=0 を設定。
*3.残り受信データ数が 2 バイトの場合 WAIT=1 を設定。残り受信データ数が 1 バイトの場合 ACKBT=1 を設定。

図 31 PMBus Master Driver Layer の状態遷移図(図 30 の補足)

表 13 PMBus Master Driver Layer の状態遷移表（送信時）

本表は mode_flag が_0D_IIC_MASTER_TRANSMIT の場合の状態遷移表を示します。
注釈の説明は表 14 にまとめて記載しています。

本表、および、表 14 の見方は以下の通りとなります。
・スレーブアドレス 10 ビットは PMBus では使用しないため、状態管理を省略しています。
・RIIC0 のタイムアウト検出割り込みは PMBus では使用しないため、状態管理を省略しています。
・「If (<条件>)」は条件付きで遷移することを意味します。
・「→ <番号>」は遷移先の番号。(<番号>は mode_flag の遷移先の番号を指します。)
・「-」は状態遷移しないことを意味します。
・callback <xxx>はコールバック関数を実行することを指します。callback error (<番号>)は callback error に渡すエラー情報を意味します。
・赤文字は PMBus 用に変更した処理を意味します。
・薄緑は図 30 における mode_flag=0D の場合の状態遷移を意味します。
・薄青は図 30 における mode_flag=0C の場合の状態遷移を意味します。

	_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)

表 14 PMBus Master Driver Layer の状態遷移表（受信時）

本表は mode_flag が_0C_IIC_MASTER_RECEIVE の場合の状態遷移表を示します。

	_00_IIC_MASTER_SENDS_ADR_7_R	_08_IIC_MASTER_RECEIVES_START	_09_IIC_MASTER_RECEIVES_DATA	_0A_IIC_MASTER_RECEIVES_STOPPING	_0B_IIC_MASTER_RECEIVES_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. ICCR2.MST が"1"の場合は、ICCR2.RS に"1"を設定しリスタートコンディションを発行。

*2. Block Read および Block Write-BlockRead Process Call 時に指定する受信データ数の初期値(total_data_size)は，プロトコルの最小受信数である 3(Data Size(1)+Data(1)+PEC(1))を指定する。それ以外のプロトコルでは各プロトコルに合わせてデータ数を指定する。また、"0"は指定できない。

*3. callback receive の内部でも、状態遷移を実行する。
*4. (2 == tatal_data_size)または (1 == tatal_data_size)の場合 WAIT=1 を設定。(1==tatal_data_size)の場合、ACKBT=1 を設定。
*5. WAIT=0 を設定。
*6. (2 <updated remain data)の場合 WAIT=0 を設定。(1 <updated remain data)の場合 ACKBT=0 を設定。
*7. Start コンディション割り込みの検出は使用しない。

5.1.3 PMBus Master 関数一覧

PMBus Master の関数は表 15 の Application 関数、表 16 の API 関数、表 17 の Middleware 関数、およびスマート・コンフィグレータで生成する表 18 のドライバ関数に分かれます。なおドライバ関数については PMBus Master 処理に合わせて一部変更しています。変更内容については 5.1.4 PMBus Master Driver 部のカスタマイズをご参照ください。

表 15 PMBus Master Application 関数一覧

File 名	Function 名	機能
r_pmbus_app_master.c	main	アプリケーションのメイン関数です。
	pmbus_demo_main	RSCI11 の送受信と PMBUS の制御を行うメインルーチンです。
	pmbus_demo_init	アプリケーションの初期化処理です。
	pmbus_demo_updata_status	アプリケーションの内部状態を更新します。
	pmbus_demo_notify_result	PMBUS の処理結果を RSCI11 でターミナルソフトへ送信します。
	pmbus_demo_response	PMBUS の受信データを RSCI11 でターミナルソフトへ送信します。
	pmbus_demo_pmbus_execute	受信したコマンドに従った PMBUS API を実行します。
	pmbus_demo_uart_recv_callback	RSCI11 受信割り込み処理を行います。
	pmbus_demo_notify_event	RSCI11 経由でアプリケーションの処理情報をターミナルソフトへ送信します。
	pmbus_demo_update_sequence	アプリケーション内の内部状態遷移を管理するメイン処理です。
	pmbus_demo_seq_idle	ターミナルソフトから最初の文字列を受信したときの受信データの解析を行います。
	pmbus_demo_seq_slave_addr	ターミナルソフトからスレーブアドレスと R/W 情報を受信した時の受信データの解析を行います。
	pmbus_demo_seq_rw	ターミナルソフトからコマンドを受信した時のトランザクションの解析を行います。
	pmbus_demo_seq_cmd	ターミナルソフトから PMBUS 送信用データを受信した時のデータ解析を行います。
	r_uart_ctrl_check_CmdFormData	ターミナルソフトから受信したデータ数が受信したコマンドのフォーマットに沿っているかのチェックを行います。
	r_uart_ctrl_check_CmdFormCmd	ターミナルソフトから受信したコマンドがサポート対象かのチェックを行います。
	r_uart_ctrl_conv_StrToDec	ターミナルソフトから受信した 16 進数の ASCII 文字を 16 進数字に変換します。
	r_uart_ctrl_is_read_write	ターミナルソフトから受信した W/R 情報を bool 値に変換します。
	r_uart_ctrl_conv_DecToStrCont	ターミナルソフトへ応答を返すために、応答用のデータを 16 進表記の ASCII 文字に変換します。
	r_uart_ctrl_conv_DecToStrr2Byte	ターミナルソフトへ応答するために、応答用のデータを 10 進数表記の ASCII 文字に変換します。

	pmbus_demo_system_init	アプリケーションで使用するドライバを初期化します。
	pmbus_demo_system_deinit	アプリケーションで使用するドライバを停止します。
	pmbus_demo_uart_ctrl_init	RSCI11 通信機能を初期化します。
	pmbus_demo_uart_ctrl_start	ターミナルソフトとの RSCI11 通信を開始します。
	pmbus_demo_uart_ctrl_stop	ターミナルソフトとの RSCI11 通信を停止します。
	pmbus_demo_uart_ctrl_read_buf	RSCI11 で受信したデータを内部バッファにコピーします。
	pmbus_demo_uart_ctrl_clear_buf	RSCI11 で受信したバッファ情報をクリアします。

表 16 PMBus Master API 関数一覧

File 名	Function 名	機能
r_pmbus_app_master.c	R_PMBUS_Master_Open	PMBus Middleware をオープンします。
	R_PMBUS_Master_Close	PMBus Middleware をクローズします。
	R_PMBUS_Master_Write	送信系の PMBUS コマンドのプロトコルを実行します。
	R_PMBUS_Master_Read	受信系の PMBUS コマンドのプロトコルを実行します。
	R_PMBUS_Master_WriteRead	Process Call 系の PMBUS コマンドのプロトコルを実行します。
	R_PMBUS_Master_EnablePEC	パケットに PEC を付与した送受信を有効にします。
	R_PMBUS_Master_DisablePEC	パケットに PEC を付与した送受信を無効にします。
	R_PMBUS_Master_ReceiveARA	Alert Response プロトコルを実行します。

表 17 PMBus Master Middleware 関数一覧

File 名	Function 名	機能
r_pmbus_app_master.c	r_pmbus_app_InitCtrl	PMBus Middleware のパラメータを初期化します。
	r_pmbus_app_SendByte	Send Byte プロトコルを実行します。
	r_pmbus_app_WriteByte	Write Byte プロトコルを実行します。
	r_pmbus_app_WriteWord	Write Word プロトコルを実行します。
	r_pmbus_app_BlockWrite	Block Write プロトコルを実行します。
	r_pmbus_app_QuickWrite	Quick Command (Write)プロトコルを実行します。
	r_pmbus_app_ReceiveByte	Receive Byte プロトコルを実行します。
	r_pmbus_app_ReadByte	Read Byte プロトコルを実行します。
	r_pmbus_app_ReadWord	Read Word プロトコルを実行します。
	r_pmbus_app_BlockRead	Block Read プロトコルを実行します。
	r_pmbus_app_ProcessCall	Process Call プロトコルを実行します。
	r_pmbus_app_BlockProcessCall	Block Write-Block Read プロトコルを実行します。
	r_pmbus_app_StartSendByte	Send Byte プロトコルを開始します。
	r_pmbus_app_StartWriteByteWord	Write プロトコルを開始します。
	r_pmbus_app_StartBlockWrite	Block Write プロトコルを開始します。
	r_pmbus_app_StartQuickCmd	Quick Command プロトコルを開始します。
	r_pmbus_app_StartReceiveByte	Receive Byte プロトコルを開始します。
	r_pmbus_app_StartReadByteWord	Read プロトコルを開始します。
	r_pmbus_app_StartBlockRead	Block Read プロトコルを開始します。

	r_pmbus_app_StartProcessCall	Process Call プロトコルを開始します。
	r_pmbus_app_StartBlockProcess	Block Write-Block Read Process Call を開始する。
	r_pmbus_app_WaitProcessEnd	プロトコルの実行が終了するまで待機します。
	r_pmbus_app_SetTxBuf	引数で指定された送信データを送信用バッファにコピーします。
	r_pmbus_app_SetBlockTxBuf	引数で指定された Block 送信用のデータを送信用バッファにコピーします。
	r_pmbus_app_GetRxBuf	受信データを受信用バッファから返却用バッファにコピーします。
	r_pmbus_app_int_ReceiveEnd	受信完了コールバック処理を実行します。
	r_pmbus_app_int_Receive	受信コールバック処理を実行します。
	r_pmbus_app_int_TransmitEnd	送信終了コールバック処理を実行します。
	r_pmbus_app_int_Notify	エラー検出コールバック処理を実行します。
r_pmbus_nwk_master.c	r_pmbus_nwk_StartTx	PMBus の送信開始プロセスを実行します。
	r_pmbus_nwk_StartRx	PMBus の受信開始プロセスを実行します。
	r_pmbus_nwk_ProcessTx	PMBus の送信処理を実行します。
	r_pmbus_nwk_ProcessRx	PMBus の受信処理を実行します。
	r_pmbus_nwk_ProcessStop	PMBus の停止処理を実行します。
	r_pmbus_nwk_ProcessErrorStop	PMBus の異常タイミングでの停止処理を実行します。
	r_pmbus_nwk_StartMaster	PMBus の物理層の動作を開始します。
	r_pmbus_nwk_StopMaster	PMBus の物理層の動作を停止します。
	r_pmbus_nwk_ResetMaster	PMBus の物理層を初期化します。
	r_pmbus_nwk_ProcessTimeout	タイムアウト検出処理を実行します。
	r_pmbus_nwk_ProcessNACK	NACK 検出処理を実行します。
	r_pmbus_nwk_ProcessArbLost	アービトレーションロスト検出処理を実行します。
	r_pmbus_nwk_GetRxPayloadSize	受信データサイズを取得します。
	r_pmbus_nwk_AddCrc8	1 つのデータの CRC 演算を実行します。
	r_pmbus_nwk_CalculatePEC	複数のデータの CRC 演算を実行します。

表 18 スマート・コンフィグレータ関数一覧

File 名	Function 名	機能	流用元からの変更有無*
Config_RIIC0_user.c	r_Config_RIIC0_transmit_interrupt	RIIC0 の送信バッファエンプティ割り込みルーチンです。	有
	r_Config_RIIC0_receive_interrupt	RIIC0 の受信バッファフル割り込みルーチンです。	有
	r_Config_RIIC0_callback_transmitend	RIIC0 の送信完了コールバックです。	有
	r_Config_RIIC0_callback_receiveend	RIIC0 の受信完了コールバックです。	有
	r_Config_RIIC0_callback_error	RIIC0 のエラー検出コールバックです。	有
	r_User_RIIC0_callback_receive	RIIC0 の受信コールバックです。	新規
Config_TMR0.c	R_Config_TMR0_Start	TMR をカウンタクリア後に実行します。	有
Config_TMR0_User.c	r_Config_TMR0_cmia0_interrupt	TMR のコンペアマッチ A 割り込みルーチンです。	有
Config_RSCI11_user.c	r_Config_RSCI11_callback_transmitend	RSCI11 の送信完了割り込みコールバックです。	有
	r_Config_RSCI11_callback_receiveend	RSCI11 の受信完了割り込みコールバックです。	有
	r_Config_RSCI11_callback_receiveerror	RSCI11 のエラー割り込みコールバックです。	有

*:変更内容については 5.1.4 **PMBus Master Driver 部のカスタマイズ**をご参照ください。

5.1.4　PMBus Master Driver 部のカスタマイズ

PMBus Master のドライバコード(RIIC0,TMR,RSCI11)はスマート・コンフィグレータにより生成します。RIIC0、および TMR についてはスマート・コンフィグレータのユーザコード保護機能により一部の処理を変更、追加しています。以下に各々のスマート・コンフィグレータの設定、および変更箇所を示します。

- スマート・コンフィグレータの RIIC0 の設定

- スマート・コンフィグレータで生成した RIIC0 ドライバコードの変更箇所一覧

関数名	r_User_RIIC0_callback_receive()
ファイル名	Config_RIIC0_user.c
変更内容	PMBus Middleware で追加した受信バッファフル割り込み時のコールバック関数です。受信バッファフル割り込みが発生ごとに PMBus Middleware の状態を確認と更新をするために、 r_pmbus_app_int_Receive()を実行します。引数には"g_riic0_rx_count"(現在の受信データ数)と g_riic0_rx_length のアドレスを指定してください。g_riic0_rx_count が"1"でブロックリード処理中の場合、g_riic0_rx_length を受信データサイズに更新されます。残りの受信データ数が 1 より大きくなった場合は、ICMR3 のＡKBT ビットを一度クリア後、g_riic0_state を_09_IIC_MASTER_RECEIVES_DATA に戻します。さらに、残りの受信データ数が 2 より大きい場合は、ICMR3 の WAIT ビットとクリアしてください。

変更前	変更後

無し。	/* 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 */

関数名	r_Config_RIIC0_receive_interrupt()
ファイル名	Config_RIIC0_user.c
変更内容	受信バッファフル割り込みハンドラです。 _09_IIC_MASTER_RECEIVES_DATA == g_riic0_state 時に受信データがまだある場合か、_0A_IIC_MASTER_RECEIVES_STOPPING == g_riic0_state の場合に r_User_RIIC0_callback_receive()を実行してください。

変更前

```
    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
```

変更後

```
    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 */
    }
```

関数名	r_Config_RIIC0_transmi_interrupt()
ファイル名	Config_RIIC0_user.c
変更内容	送信バッファエンプティ割り込みハンドラです。 グローバル変数に g_riic0_tx_length を追加した上で、 _0D_IIC_MASTER_TRANSMIT == g_riic0_mode_flag でスレーブアドレスを送信した後に、g_riic0_tx_length に g_riic0_tx_count を保存してください。

変更前

```
/* 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;
        }
```

変更後

```
/* 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 */
        }
```

関数名	r_Config_RIIC0_callback_transmitend()
ファイル名	Config_RIIC0_user.c
変更内容	送信完了割り込み時のコールバック関数です。 送信完了割り込み後に PMBus Middleware の状態を確認と更新をするために、r_pmbus_app_int_TransmitEnd ()を実行してください。引数には"g_riic0_tx_length"(送信予定のデータ総数)を指定してください。0 の場合は Quick Command の処理を実行します。

変更前

```
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 }
```

変更後

```
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 }
```

関数名	r_Config_RIIC0_callback_receiveend ()
ファイル名	Config_RIIC0_user.c
変更内容	受信完了時のコールバック関数です。 全データの受信完了後の Stop コンディション割り込み後に PMBus Middleware の状態を確認と更新をするために、 r_pmbus_app_int_ReceiveEnd ()を実行してください。

変更前

```
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 }
```

変更後

```
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 }
```

関数名	r_Config_RIIC0_callback_error()
ファイル名	Config_RIIC0_user.c
変更内容	エラー検出割り込み時のコールバック関数です。引数 status ごとに以下の処理を実行してください。 "MD_ERROR1"（Arbitration Lost）の場合： PMBus Middleware の状態を更新するために r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_ARB_LOST) を実行してください。 "MD_ERROR3"（NACK 検出）の場合： PMBus Middleware の状態を更新するために r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_NACK) を実行してください。 "MD_ERROR4"（ドライバシーケンス外での Stop コンディションを検出）の場合： ICSR2 の STOP ビットが 1 である場合は、 PMBus Middleware の状態を更新するために g_riic0_mode_flag が _OC_IIC_MASTER_RECEIVE の場合は、RIIC0.ICMR3.BIT.WAIT に 0 を設定した上で、r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_STOP_ERROR)、g_riic0_mode_flag が上記以外の場合は、r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_STOP) を実行してください。 ※MD_ERROR2"（TimeoutError 検出）は PMBus Middleware では使用しません。

変更前

```
418         case MD_ERROR1:
417         {
418             /* Start user code for arbitration-lost error. Do not edit comment generated here
    */
419             /* End user code. Do not edit comment generated here */
420
421             RIIC0.ICSR2.BIT.AL = 0U;
422             break;
423         }
```

```
444         case MD_ERROR3:
445         {
446             /* Start user code for NACK signal. Do not edit comment generated here */
447             /* End user code. Do not edit comment generated here */
448             break;
449         }
```

```
450         case MD_ERROR4:
451         {
452             /* Start user code for communication sequence error. Do not edit comment
    generated here */
453             /* End user code. Do not edit comment generated here */
454             break;
455         }
```

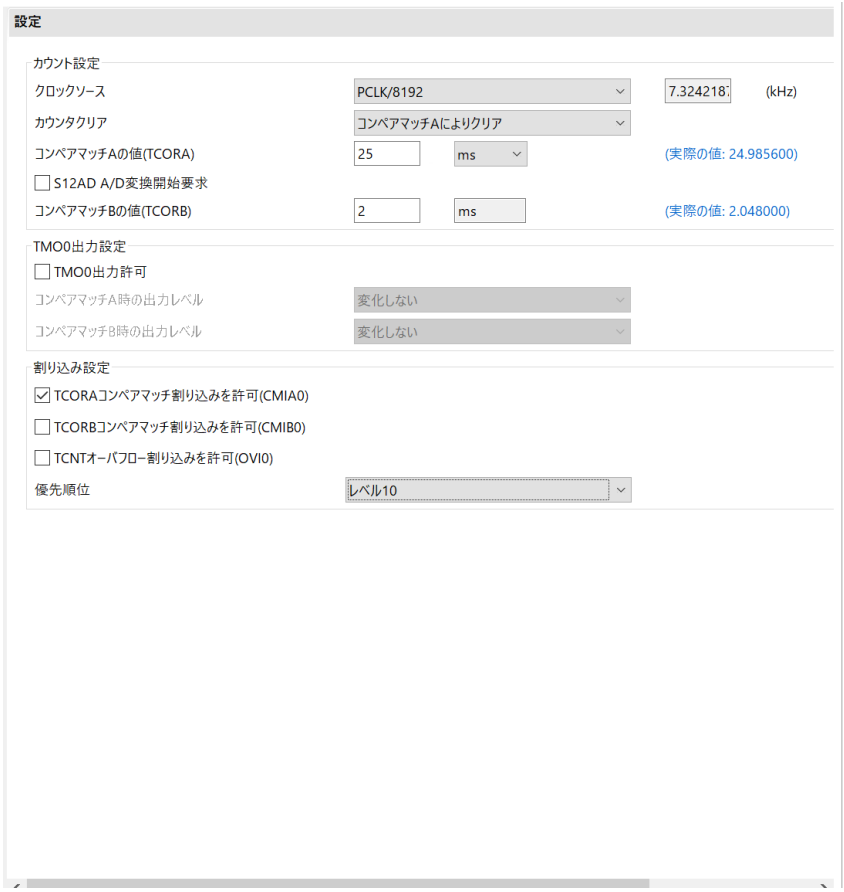
変更後

```
437         case MD_ERROR1:
438         {
439             /* Start user code for arbitration-lost error. Do not edit comment generated here
    */
440             r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_ARB_LOST);
441
442             /* End user code. Do not edit comment generated here */
443
444             RIIC0.ICSR2.BIT.AL = 0U;
445             break;
446         }
```

```
467         case MD_ERROR3:
468         {
469             /* Start user code for NACK signal. Do not edit comment generated here */
470             r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_NACK);
471
472             /* End user code. Do not edit comment generated here */
473             break;
474         }
```

```
475         case MD_ERROR4:
476         {
477             /* Start user code for communication sequence error. Do not edit comment generated
    here */
478             if (1 == RIIC0.ICSR2.BIT.STOP)
479             {
480                 RIIC0.ICSR2.BIT.NACKF = 0U;
481                 RIIC0.ICSR2.BIT.STOP = 0U;
482
483                 if (_OC_IIC_MASTER_RECEIVE == g_riic0_mode_flag)
484                 {
485                     /* If STOP conodition detect before last data received is illegal. */
486                     RIIC0.ICMR3.BIT.WAIT = 0U;
487                     r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_STOP_ERROR);
488                 }
489                 else
490                 {
491                     r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_M_STOP);
492                 }
493             }
494
495             /* End user code. Do not edit comment generated here */
496             break;
497         }
```

- スマート・コンフィグレータの TMR の設定



- スマート・コンフィグレータで生成した TMR ドライバコードの変更箇所一覧

関数名	R_Config_TMR0_Start()
ファイル名	Config_TMR0.c
変更内容	TMR0.TCNT をクリアする処理を追加してください。

変更前

```
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;
}
```

変更後

```
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;
}
```

関数名	r_Config_TMR0_cmia0_interrupt()
ファイル名	Config_TMR0_user.c
変更内容	コンペアマッチ割り込み時のコールバック関数です。 PMBus Middleware の状態を更新するために r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_TIMEOUT)を実行してください。

変更前

```
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 */
}
```

変更後

```
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 */
}
```

- スマート・コンフィグレータの RSCI11 の設定

- スマート・コンフィグレータで生成した RSCI11 ドライバコードの変更箇所一覧

関数名	r_Config_RSCI11_callback_transmitend
ファイル名	Config_RSCI11_user.c
変更内容	pmbus_demo_update_status(E_MAIN_STATUS_RESPONSE_COMP)を実行してください。

変更前

```
166 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 }
```

変更後

```
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 */
```

関数名	r_Config_RSCI11_callback_receiveend
ファイル名	Config_RSCI11_user.c
変更内容	pmbus_demo_uart_recv_callback()を実行してください。

変更前

```
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 */
```

変更後

```
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 }
```

関数名	r_Config_RSCI11_callback_receiveerror
ファイル名	Config_RSCI11_user.c
変更内容	pmbus_demo_notify_event(E_UART_EVENT_RECEIVE_ERROR)を実行してください。

変更前

```
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 }
```

変更後

```
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 データタイプ、構造体一覧

本制御プログラムで使用するデータタイプ、および、構造体を次に示します。

- PMBus Master Application 関数で使用するデータタイプ、構造体

e_main_status_t

列挙型名	e_main_status_t	
説明	PMBus Master Application のメインステータスを示すために使用する列挙型です。	
宣言しているヘッダファイル	r_pmbus_demo_master.h	
備考	-	
要素名	**説明**	**値**
E_MAIN_STATUS_IDLE	IDLE 状態。UART データ受信があるまで待機します。	0
E_MAIN_STATUS_REV_BUSY	UART データを受信中。	1
E_MAIN_STATUS_REV_COMP	コマンド処理に必要な UART データ受信を完了。	2
E_MAIN_STATUS_PMBUS_BUSY	PMBUS API を実行中。（予備）	3
E_MAIN_STATUS_PMBUS_COMP	PMBUS API の動作終了。	4
E_MAIN_STATUS_PMBUS_ACTIVE	PMBUS API を実行中。	5
E_MAIN_STATUS_RESPONSE_BUSY	コマンド処理結果を UART で送信中。	6
E_MAIN_STATUS_RESPONSE_COMP	UART データの送信を完了。	7
E_MAIN_STATUS_ERROR_BUSY	エラー処理中。（予備）	8
E_MAIN_STATUS_ERROR_COMP	エラー処理完了。（予備）	9
E_MAIN_STATUS_MAX	メインステータスの最大値。	10

e_uart_event_t

列挙型名	e_uart_event_t	
説明	UART で応答するメッセージの番号を指定するために使用する列挙型です。	
宣言しているヘッダファイル	r_pmbus_demo_master.h	
備考	-	
要素名	**説明**	**値**
E_UART_EVENT_COMMAND_END	送信文字列に" PMBUS COMMAND END"を指定。	0
E_UART_EVENT_RESP_START	送信文字列に">>PMBUS_RESPONSE_START"を指定。	1
E_UART_RECEIVE_FULL	送信文字列に">>ERROR:UART RECEIVE BUFFER IS FULL"を指定。	2
E_UART_RECEIVE_ERROR	送信文字列に"ERROR:UART RECEIVE ERROR"を指定。	3
E_UART_EVENT_FORMAT_ERROR	送信文字列に"ERROR:PMBUS FORMAT ERROR "を指定。	4
E_UART_EVENT_COMMAND_INVALIDATE	送信文字列に"ERROR:PMBUS COMMAND NOT SUPPORT"を指定。	5
E_UART_EVENT_COMMAND_FAIL	送信文字列に"ERROR:PMBUS API RETURN ERROR "を指定。	6
E_UART_EVENT_MAX	メッセージ番号の最大値。	7

_response_type_t

列挙型名	e_response_tyep_t	
説明	UART で応答する数字表現の表記タイプを指定するために使用する列挙型です。s_st_pmbus_command_check_table に登録している PMBUS コマンドごとの表記タイプの指定で使用します。	
宣言しているヘッダファイル	r_pmbus_demo_master.h	
備考	-	
要素名	**説明**	**値**
E_RESPONSE_TYPE_HEX	16 進数表記を指定。	0
E_RESPONSE_TYPE_DECIMAL	符号あり 10 進数表記を指定。	1
E_RESPONSE_TYPE_DECIMAL_UNSIGNED	符号なし 10 進数表記を指定。	2
E_RESPONSE_TYPE_MAX	数字表現の表記タイプの最大値。	3

e_app_varidate_info_t

列挙型名	e_app_varidate_info_t	
説明	UART で受信した PMBUS のコマンドの有効/無効情報を示すために使用する列挙型です。s_st_pmbus_command_check_table に登録している PMBUS コマンドごとの有効/無効の指定で使用します。	
宣言しているヘッダファイル	r_pmbus_demo_master.h	
備考	-	
要素名	**説明**	**値**
E_COMMAND_INVALIDATE	無効コマンドを指定。	0
E_COMMAND_VALIDATE	有効コマンドを指定。	1

st_app_command_format_t

構造体名	st_app_command_format_t
説明	PMBus Master Application で処理する PMBUS コマンドのフォーマット情報を管理する構造体です。 UART 経由で受信したデータが該当の PMBUS コマンドのフォーマットに一致するかの確認、および、UART 経由で応答するフォーマットを決定するために使用します。
宣言しているヘッダファイル	r_pmbus_demo_master.h
備考	-
メンバ名	**メンバ説明**
uint8_t u1_validate	コマンドの有効/無効情報。(E_COMMAND_INVALIDATE or E_COMMAND_VALIDATE)
uint8_t u1_command	PMBUS のコマンドコード。(0x00～0xFF)
uint8_t u1_transaction	PMBUS コマンドのトランザクションタイプ。(トランザクションコード(PMBUS_TRANS_XXX)の or の組み合わせ)
uint16_t u2_tx_data_size	PMBUS コマンドで処理する送信データ数。(0x00～0xFF)
uint16_t u2_rx_data_size	PMBUS コマンドで処理する受信データ数。(0x00～0xFF)
e_response_type_t e_resp_type	PMBUS コマンドの受信データを UART 経由で返信する際の数字表現の指定。 (E_RESPONSE_TYPE_HEX, E_RESPONSE_TYPE_DECIMAL または E_RESPONSE_TYPE_DECIMAL_UNSIGNED)

st_pmbus_data_t

構造体名	st_pmbus_data_t
体説明	UART から受信した PMBUS コマンド情報を管理するの構造体です。
宣言しているヘッダファイル	r_pmbus_demo_master.h
備考	-
メンバ名	メンバ説明
uint8_t u1_command	実行する PMBUS コマンドのコマンドコード。
uint8_t u1_transaction	実行する PMBUS コマンドのトランザクションコード。(トランザクションコード(PMBUS_TRANS_XXX)の or の組み合わせ)
uint8_t u1_slave_addr	PMBUS コマンドの通信先のスレーブアドレス。
uint16_t u2_data_size	実行する PMBUS コマンドの送信データ総数。（予備）
uint16_t u2_tx_index	PMBUS コマンドで処理する現在の送信データ数。
uint16_t u2_command_index	実行する PMBUS コマンドに対応した s_st_pmbus_command_check_table のインデックス。(0～(COMMAND_TABLE_SIZE - 1))
uint8_t u1_tx_buf[PMBUS_TX_BUF_SIZE]	PMBUS コマンドで処理する送信データを格納するバッファ。
bool b_direction	PMBUS コマンドで処理する通信方向。(false:受信プロトコル or true:送信または送受信プロトコル)

st_app_uart_rx_t

構造体名	st_app_uart_rx_t
説明	UART で受信したデータを管理する構造体です。
宣言しているヘッダファイル	r_pmbus_demo_master.h
備考	-
メンバ名	メンバ説明
uint8_t u1_recv_buf[UART_RX_BUF_SIZE]	UART の受信データを格納するバッファ。
bool b_recv_flag	1 行分のデータの受信完了状況。(false:受信未完了 or true:受信完了(改行コード"0x0A"受信時に設定))

- PMBus Master API で使用するデータタイプ、構造体

e_pmbus_packet_result_m_t

列挙型名	e_pmbus_packet_result_m_t	
説明	PMBus 通信(master)の実行結果を示すために使用する列挙型です。各 PMBus Master API の引数の型として使用します。PMBus API の Return 値が PMBUS_RET_ERROR の場合のエラー原因の詳細を示します。	
宣言しているヘッダファイル	r_pmbus_app_master.h	
備考	-	
要素名	**説明**	**値**
E_PMBUS_PACKET_M_OK	正常動作。	0
E_PMBUS_PACKET_M_DATA_SIZE_ERROR	パケットサイズ異常を検出。	1
E_PMBUS_PACKET_M_PEC_ERROR	PEC 演算結果の異常を検出。	2
E_PMBUS_PACKET_M_TIMEOUT	タイムアウトエラーを検出。（$T_{TIMEOUT}$ 異常の検出）	3
E_PMBUS_PACKET_M_ARB_LOST	アービトレーションロストエラーを検出。	4
E_PMBUS_PACKET_M_NACK	NACK 受信を検出。	5
E_PMBUS_PACKET_M_INTERNAL_ERROR	内部異常を検出。	6

st_pmbus_cfg_m_t

構造体名	st_pmbus_cfg_m_t
説明	PMBus Master のコンフィグレーションデータ用の構造体です。 R_PMBUS_Master_Open の引数の型として使用します。コンフィグレーションデータを PMBus Master Middleware の内部グローバル変数へ登録するために使用します。
宣言しているヘッダファイル	r_pmbus_app_master.h
備考	-
メンバ名	**メンバ説明**
uint16_t u2_rx_size	*p_u1_rx_buf のサイズ。(1～35)
uint8_t *p_u1_rx_buf	受信データ格納バッファへのポインタ。R_PMBUS_Master_Read、または、R_PMBUS_Master_WriteRead で受信したデータは本バッファへ格納し、引数*p_u1_rx_buf、および、*u2_rx_len に積みなおしてから API の呼び出し元へ返却します。
uint16_t u2_tx_size	*p_u1_tx_buf のサイズ。(1～35)
uint8_t *p_u1_tx_buf	送信データ格納バッファへのポインタ。R_PMBUS_Master_Write、または、R_PMBUS_Master_WriteRead の引数で指定した u1_command、u2_tx_len、*p_u1_tx_buf のデータをコマンドコードに対応したプロトコルに従ったデータ配置で本バッファへ積みなおしたのち送信します。

- PMBus Master Middleware 関数で使用するデータタイプ、構造体

e_pmbus_nwk_status_m_t

列挙型名	e_pmbus_nwk_status_m_t	
説明	PMBus network layer(master)の内部状態を示す列挙型です。PMBus Master Middleware の内部状態を管理するために使用します。	
宣言しているヘッダファイル	r_pmbus_app_master.h	
備考	-	
要素名	**説明**	**値**
E_PMBUS_NWK_STATUS_M_IDLE	新しいパケットの受信待ち。	0
E_PMBUS_NWK_STATUS_M_RX	パケットを受信中。	1
E_PMBUS_NWK_STATUS_M_TX	パケットを送信中。	2
E_PMBUS_NWK_STATUS_M_TX_QUICK	Quick コマンドを送信中。	3
E_PMBUS_NWK_STATUS_M_TX_BLOCK	ブロック送信中。	4
E_PMBUS_NWK_STATUS_M_RX_BLOCK	ブロック受信中。	5
E_PMBUS_NWK_STATUS_M_ENDING	パケット終了。	6
E_PMBUS_NWK_STATUS_M_ERROR	パケットエラーを検出。	7

e_pmbus_int_event_m_t

列挙型名	e_pmbus_int_event_m_t	
説明	I2C のエラー検出割り込みの発生要因を示す列挙型です。PMBus Master Middleware の内部処理で割り込み要因ごとの処理を実行するために使用します。	
宣言しているヘッダファイル	r_pmbus_app_master.h	
備考	-	
要素名	**説明**	**値**
E_PMBUS_INT_EVENT_M_NONE	割り込み未検出	0
E_PMBUS_INT_EVENT_M_ARB_LOST	アービトレーションロストを検出。	1
E_PMBUS_INT_EVENT_M_TIMEOUT	タイムアウトを検出。	2
E_PMBUS_INT_EVENT_M_NACK	NACK 受信を検出。	3
E_PMBUS_INT_EVENT_M_START	Start コンディションを検出。	4
E_PMBUS_INT_EVENT_M_STOP	Stop コンディションを検出。	5
E_PMBUS_INT_EVENT_M_STOP_ERROR	想定外の Stop コンディションを検出。	6

st_pmbus_nwk_ctrl_m_t

構造体名	st_pmbus_nwk_ctrl_m_t
説明	PMBus network Layer (master) パラメータを管理する構造体です。PMBus Master Middleware 内部の通信状態を管理するために使用します。
宣言しているヘッダファイル	r_pmbus_app_master.h
備考	-
メンバ名	メンバ説明
volatile e_pmbus_nwk_status_m_t e_m_status	Network layer のステータス
uint8_t u1_current_addr	現在実行中のスレーブアドレス
uint8_t u1_current_cmd	現在実行中のコマンド
uint16_t u2_rx_index	現在の受信データバイト数
uint16_t u2_rx_len	予定する受信データバイト数
uint16_t u2_rx_size	*p_u1_rx_buf のサイズ
uint8_t *p_u1_rx_buf	受信データ格納バッファへのポインタ
uint16_t u2_tx_index	現在の送信データバイト数
uint16_t u2_tx_len	予定する送信データバイト数
uint16_t u2_tx_size	*p_u1_tx_buf のサイズ
uint8_t *p_u1_tx_buf	送信データ格納バッファへのポインタ
uint8_t u1_pec	現在の PEC 演算結果

st_pmbus_ctrl_m_t

構造体名	r_pmbus_app_master.h
説明	PMBus Middleware (master)のコントロールデータ構造体です。PMBus Master Middleware の設定情報、および、通信状態を管理するために使用します。
宣言しているヘッダ	r_pmbus_app_master.h
備考	-
メンバ名	メンバ説明
st_pmbus_nwk_ctrl_m_t st_nwk_ctrl_m	PMBus network Layer (master) のパラメータ管理する構造体
volatile e_pmbus_packet_result_m_t e_pmbus_result_m	PMBus 通信(master)の実行結果
bool b_open_flag	OPEN 状態（false: Open していない、true: Open している。）
bool b_pec_flag	PEC の有効無効情報（false:無効、true:有効）
uint8_t u1_own_slave_addr	自身のスレーブアドレス。（デモシステムでは使用しない。）

5.1.6 PMBus Master グローバル変数一覧

本制御プログラムで使用するグローバル変数一覧を次に示します。

表 19 PMBus Master Application グローバル変数一覧

File 名	グローバル変数	用途
r_pmbus_demo_master.c	static const st_app_command_format_t s_st_pmbus_command_check_table[COMMAND_TABLE_SIZE]	UART で受信した PMBUS コマンドのフォーマットを確認するために使用する const table。
	static const char * sp_pmbus_message_return_res	PMBUS の実行結果を UART 送信バッファに格納する際に付加する文字列"return code:"を格納した const 値。
	static const char * sp_pmbus_message_packet_res	PMBUS の実行結果を UART 送信バッファに格納する際に付加する文字列"packet result:"を格納した const 値。
	static st_app_uart_rx_t s_st_uart_rx_data	UART の受信データ情報を管理する構造体変数。本構造体のメンバである u1_recv_buf に格納された受信データを解析し、PMBUS 処理を制御します。
	static uint8_t s_u1_uart_tx_buf[UART_TX_BUF_SIZE]	UART の送信データを格納するバッファ。
	static uint8_t s_u1_uart_rx_buf[UART_RX_BUF_SIZE]	UART の受信データを格納するバッファ。本バッファに格納された受信データを解析し、受信データの区切りである改行コードを検出します。本バッファはリングバッファとして使用します。
	static uint16_t s_u2_uart_rx_index;	現在の UART の受信データ数を管理する変数。
	static uint16_t s_u2_rx_r_index	受信データバッファ(s_u1_uart_rx_buf)をリードした最新のデータ位置を示す変数。
	static uint16_t s_u2_rx_w_index	受信データバッファ(s_u1_uart_rx_buf)に格納された最新のデータ位置を示す変数。
	static uint8_t s_u1_uart_rx_relay_buf[UART_RELAY_BUF_SIZE]	UART の受信データを格納するバッファ。UART 受信割り込みハンドラで本バッファに受信データを格納後、メイン処理で s_u1_uart_rx_buf へ受信データをコピーします。
	volatile static e_main_status_t s_e_main_status	アプリケーションの内部処理状態を管理するための変数。
	volatile static uint16_t s_u2_seq_index	UART コマンドの受信処理状態を管理するための変数。本変数により PMBUS 通信に必要なデータをどこまで受信しているかを判断します。
	static st_pmbus_data_t s_st_pmbus_data	PMBUS コマンド用のデータを管理する構造体変数。本構造体変数に格納されたデータを PMBUS API に渡す引数として使用します。
	static uint8_t s_u1_pmbus_tx_buf[PMBUS_TX_BUF_SIZE]	PMBUS 用の送信データを格納するバッファ。本バッファを s_user_pmbus_cfg のメンバ*p_u1_tx_buf に設定し、PMBUS Middleware 内部で使用します。
	static uint8_t s_u1_pmbus_rx_buf[PMBUS_RX_BUF_SIZE]	PMBUS 用の受信データを格納するバッファ。本バッファを s_user_pmbus_cfg のメンバ*p_u1_rx_buf に設定し、PMBUS Middleware 内部で使用します。

	static uint16_t s_u2_pmbus_rx_size	PMBUS で受信したデータ数。R_PMBUS_Master_Read、および、R_PMBUS_Master_WriteRead の引数に指定し、PMBUS で受信したデータ数を格納します。
	static uint8_t s_u1_pmbus_temp_rx_buf[PMBUS_RX_BUF_SIZE]	PMBUS API 実行時に指定する PMBUS の受信データを格納するバッファ。R_PMBUS_Master_Read、および、R_PMBUS_Master_WriteRead の引数に指定し、PMBUS で受信したデータを格納します。
	static e_pmbus_packet_result_m_t s_e_packet_result	PMBUS API の実行結果（パケット結果）を格納する変数。各 PMBUS API の引数に指定し、PMBUS API の実行結果（パケット結果）を格納します。
	static uint8_t s_u1_pmbus_ret	PMBUS API の実行結果（リターンコード）を格納する変数。
	static st_pmbus_cfg_m_t s_user_pmbus_cfg	R_PMBUS_Master_Open で登録するコンフィグレーションデータを格納する変数。

表 20 PMBus Master Middleware のグローバル変数

File 名	グローバル変数	用途
r_pmbus_master_app.c	static st_pmbus_ctrl_m_t g_st_pmbus_ctrl	PMBUS Master Middleware のコントロール情報を管理するグローバル変数です。PMBUS Master Middleware の内部でのみ使用します。

5.1.7 PMBus Master マクロ定義一覧

本制御プログラムで使用するマクロ定義一覧を次に示します。

表 21 PMBus Master Application マクロ定義一覧

File 名	マクロ名	用途	定義値
r_pmbus_demo_master.h	PMBUS_APP_CMD_xxx	・PMBUS のコマンドコードの定義。（マクロ名は以下参照）	-
		PMBUS_APP_CMD_OPERAION ： OPERAION コマンド	0x01
		PMBUS_APP_CMD_ON_OFF_CONFIG ： ON_OFF_CONFIG コマンド	0x02
		PMBUS_APP_CMD_CLEAR_FAULTS ： CLEAR FAULTS コマンド	0x03
		PMBUS_APP_CMD_STATUS_FAN_1_2 ： STATUS_FAN_1_2 コマンド	0x81
		PMBUS_APP_CMD_READ_VOUT ： READ_VOUT コマンド	0x8B
		PMBUS_APP_CMD_READ_IOUT ： READ_IOUT コマンド	0x8C
		PMBUS_APP_CMD_READ_FAN_SPEED_1 ： REA_DFAN_SPEED_1 コマンド	0x90
		PMBUS_APP_CMD_READ_FREQUENCY ： READ_FREQUENCY コマンド	0x95
		PMBUS_APP_CMD_RESERVED ： RESERVED コマンド	0x09
		PMBUS_APP_CMD_PMBUS_REVISION ： PMBUS_REVISION コマンド	0x98
		PMBUS_APP_CMD_STORE_DEFAULT_CODE ： STORE_DEFAULT_CODE コマンド	0x13
		PMBUS_APP_CMD_FAN_COMMAND_1 ： FAN_COMMAND_1 コマンド	0x38
		PMBUS_APP_CMD_READ_EOUT ： READ_EOUT コマンド	0x87
		PMBUS_APP_CMD_PAGE_PLUS_WRITE ： PAGE_PLUS_WRITE コマンド	0x05
	UART_TX_BUF_SIZE	UART で送信するデータを格納するバッファの最大値。	256
	UART_RX_BUF_SIZE	UART で受信するデータを格納するバッファの最大値。	256
	UART_RELAY_BUF_SIZE	UART で受信したデータを一時的に格納するバッファの最大値。	3
	PMBUS_TX_BUF_SIZE	PMBUS で送信するデータを格納するバッファの最大値。	40
	PMBUS_RX_BUF_SIZE	PMBUS で受信するデータを格納するバッファの最大値。	40
	COMMAND_TABLE_SIZE	コマンドチェックテーブル（s_st_pmbus_command_check_table）に登録するコマンド総数。	14
	SEQ_INDEX_xxx	・UART コマンドの受信処理状態の定義。（マクロ名は以下参照）	-
		SEQ_INDEX_IDLE：UART 受信データ待機中。	0
		SEQ_INDEX_SLAVE_ADDR：スレーブアドレスまで受信完了。	1
		SEQ_INDEX_RW： READ/WRITE 情報まで受信完了。	2
		SEQ_INDEX_COMMAND：コマンドコードまで受信完了。	3
		SEQ_INDEX_WRITE_DATA：PMBUS で送信する送信データの受信中、または、受信完了。	4
		SEQ_INDEX_MAX：本アプリケーションで処理できる PMBUS 送信データの最大数。	256
	UART_RESPONSE_TIMEOUT_TIME	PMBUS の受信結果を UART で返信する際の、UART 通信完了まで待機するソフトタイマカウント値。	0x50000
	RET_OK	アプリケーション内部関数の戻り値。正常終了。	0
	RET_ERROR	アプリケーション内部関数の戻り値。異常終了。	1

表 22 PMBus Master API マクロ定義一覧

File 名	マクロ名	用途	定義値
r_pmbus_app_master.h	PMBUS_RET_xxx	・PMBus middleware API から返却するエラーコード。（マクロ名は以下参照）	-
		PMBUS_RET_OK ： 正常終了	0
		PMBUS_RET_ERROR ： 異常終了。原因の詳細は st_pmbus_cfg_m_te_pmbus_result_m で確認してください	1
		PMBUS_RET_PARAM ： 指定された引数が異常。	2
		PMBUS_RET_NOT_OPENED ： OPEN されていません	3
		PMBUS_RET_OPENED ： すでに OPEN されています	4

表 23 PMBus Master Middleware 関数のマクロ定義一覧

File 名	マクロ名	用途	定義値
r_pmbus_app_master.h	PMBUS_TRANS_xxx	・各コマンドコードごとでサポートするプロトコルを判別する際に使用するトランザクションコードの定義。（マクロ名は以下参照）	-
		PMBUS_TRANS_RESERVED ： コマンドコードは RESERVED	0x00
		PMBUS_TRANS_READ_BYTE ： コマンドコードは READ BYTE トランザクションをサポート	0x01
		PMBUS_TRANS_READ_WORD ： コマンドコードは READ WORD トランザクションをサポート	0x02
		PMBUS_TRANS_BLOCK_READ ： コマンドコードは BLOCK READ トランザクションをサポート	0x03
		PMBUS_TRANS_SEND_BYTE ： コマンドコードは SEND BYTE トランザクションをサポート	0x10
		PMBUS_TRANS_WRITE_BYTE ： コマンドコードは WRITE BYTE トランザクションをサポート	0x20
		PMBUS_TRANS_WRITE_WORD ： コマンドコードは WRITE WORD トランザクションをサポート	0x30
		PMBUS_TRANS_BLOCK_WRITE ： コマンドコードは BLOCK WRITE トランザクションをサポート	0x40
		PMBUS_TRANS_WRITE_QUICK ： コマンドコードは Write Quick トランザクションをサポート	0x50
		PMBUS_TRANS_PROCESS_CALL ： コマンドコードは PROCESS CALL トランザクションをサポート	0x60
		PMBUS_TRANS_BLOCK_PROCESS_CALL ： コマンドコードは Block Write-Block Read Process Call トランザクションをサポート	0x70
	PMBUS_COMMAND_CODE_MAX	PMBus Middleware でサポートするコマンドの最大数。	256
	PMBUS_BLOCK_SIZE_MIN	Block コマンドで送受信できるデータ数の最小値。	1
	PMBUS_BLOCK_SIZE_MAX	Block コマンドで送受信できるデータ数の最大値。	32
	PMBUS_BUF_SIZE_MIN	Open 時に PMBUS Middleware に登録できるバッファサイズの最小値。	1
	PMBUS_BUF_SIZE_MAX	Open 時に PMBUS Middleware に登録できるバッファサイズの最大値。（Block Read/Block Write 時の書き込み最大データ数(32)+	PMBUS_BLOCK_SI

		コマンドコード(1)+書き込みデータ数/読み出しデータ数(1)+PEC(1))	ZE_MAX + 3
	PMBUS_CRC8_USE_IP	PEC の演算方法を指定する定義値。"1(MCU に搭載している演算器を使用する)"か、"0(テーブルを使用して演算するかを指定する)"かを設定する。MCU に搭載している演算器を使用する場合は、r_pmbus_nwk_AddCrc8()関数に CRC 演算器を使用したコードをユーザが実装した上で"1(MCU に搭載している演算器を使用する)"に変更してください。	0
	PMBUS_ALERT_RESPONSE_ADDR	R_PMBUS_Master_RecevieARA を実行時に指定する ALERT 情報受信用の通信先のスレーブアドレス(ARA)。SMBus の仕様に準拠した値を定義しています。	0x0C
	PMBUS_SLAVE_ADDR_MAX	PMBUS のマスタ API の引数で指定できるスレーブアドレスの最大値の定義です。	0x80

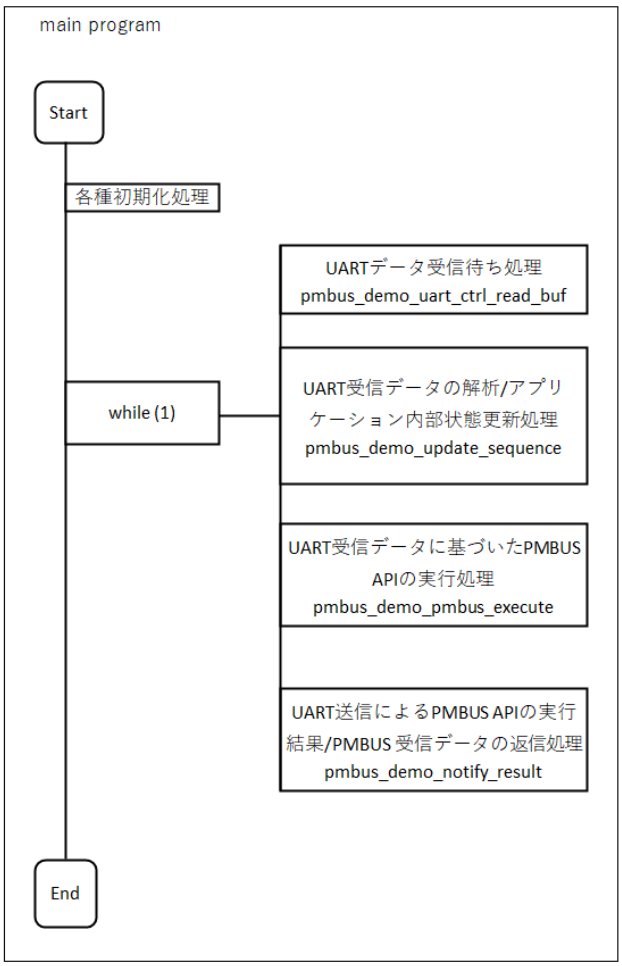
5.1.8　PMBus Master 制御フロー

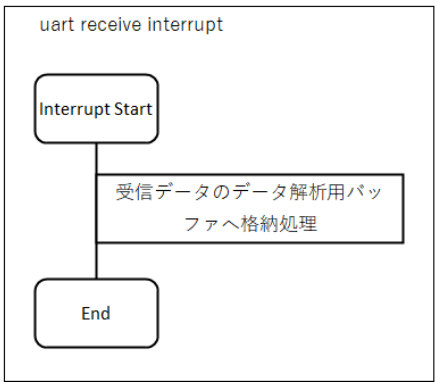
PMBus Master Application 部のフローを 5.1.8.1 項に、PMBus Master API 部のフローを 5.1.8.2 項に、PMBus Master ドライバ部のフローを 5.1.8.3 項に示します。なお PMBus Master Middleware 部についてはプロジェクトコードをご参照ください。

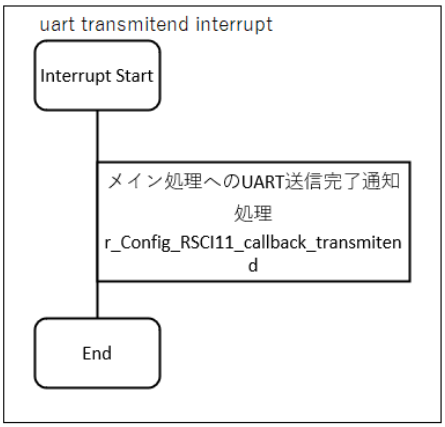
5.1.8.1 PMBus Master Application 部フロー

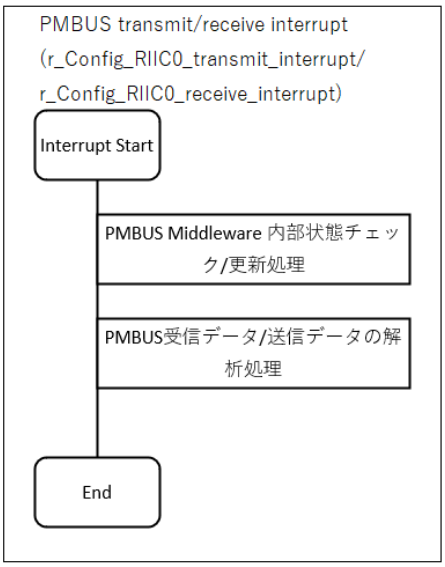
PMBus Master Application 部は PC との通信、および PMBus Master Middleware 部を制御する API のコールを行います。PMBus Master Application 部のフローを以下に示します。RSCI11 に関連する PAD 図の赤枠部分は、スマート・コンフィグレータで生成したコードからの変更箇所を意味します。
なお PMBus Master Application 部の中間関数の処理フローはプロジェクトコードをご参照ください。

- PMBus master 全体概略フロー



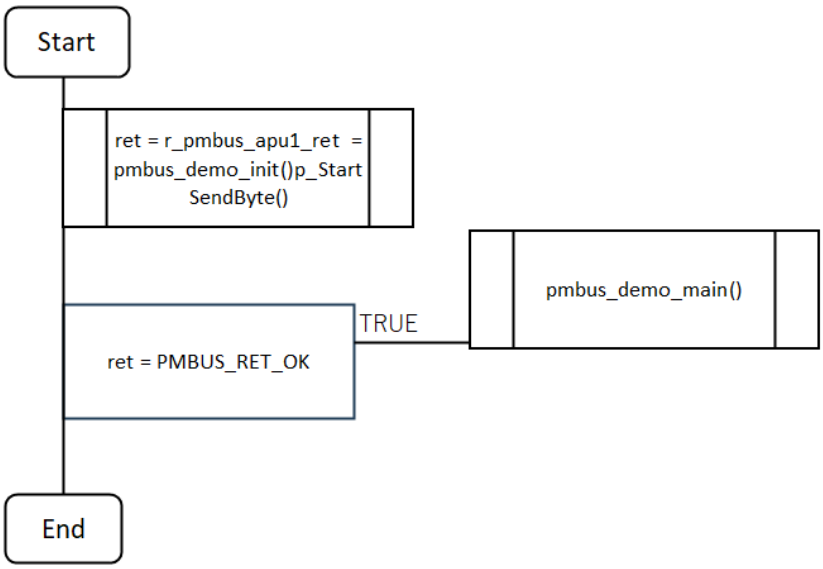




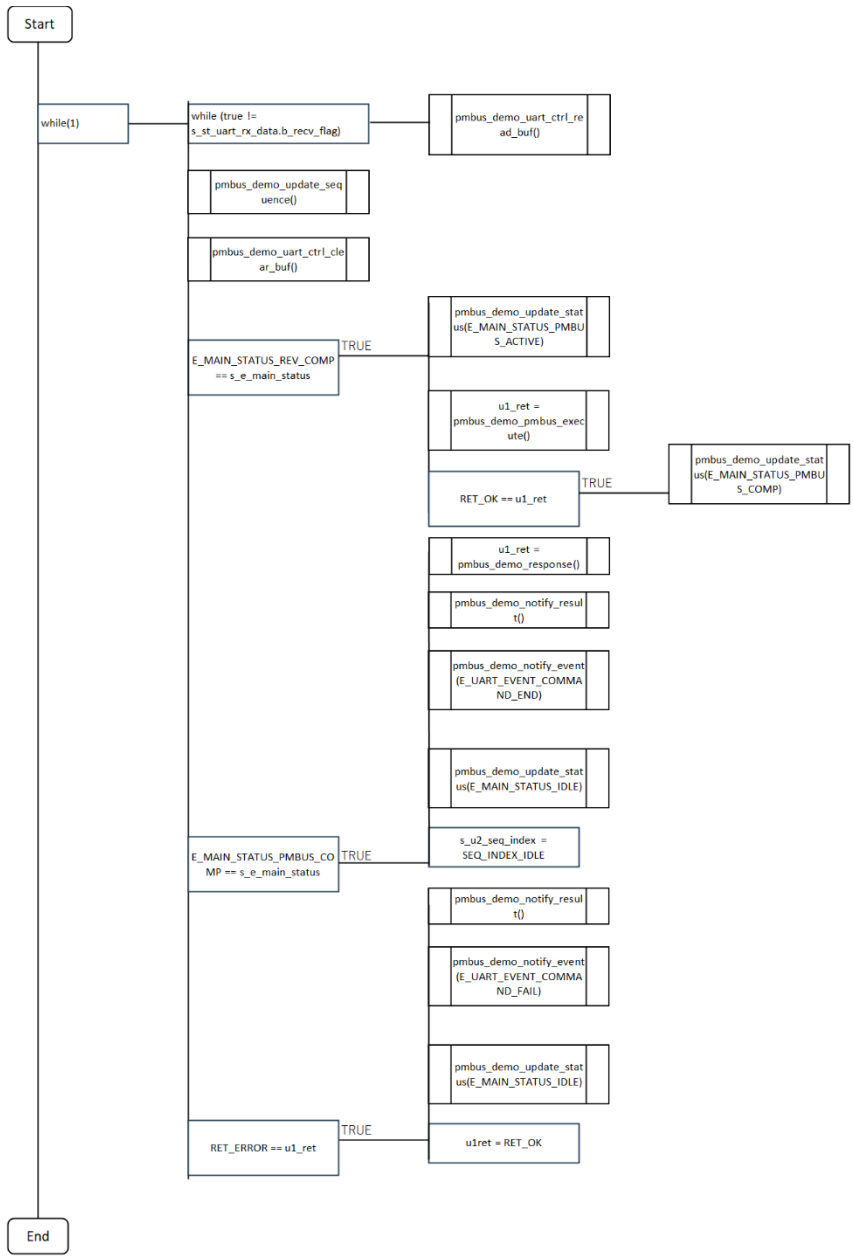


■ PMBus Master 関数 詳細フロー

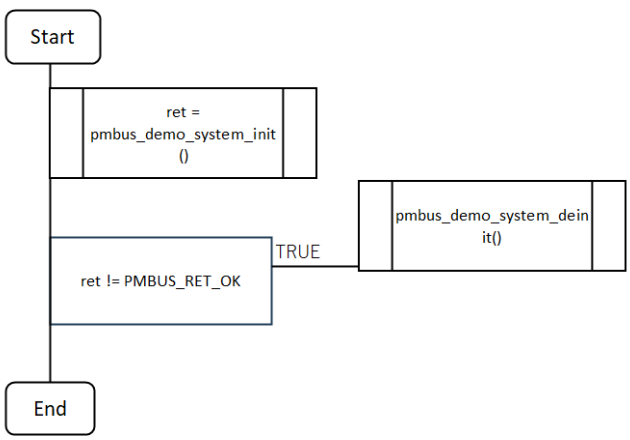
● main



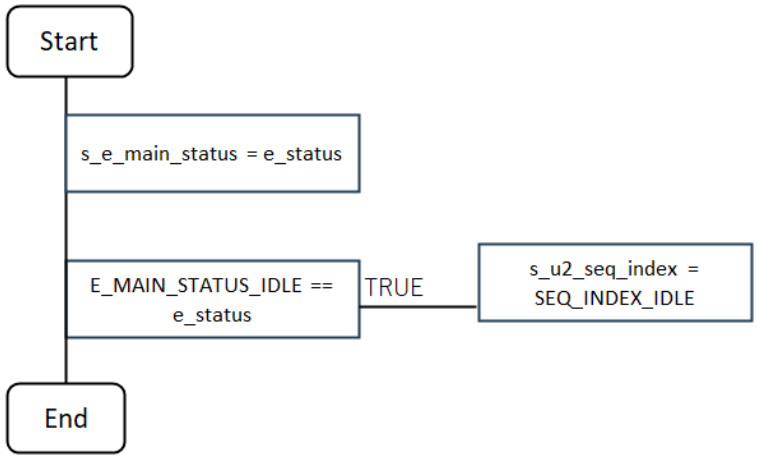
- pmbus_demo_main



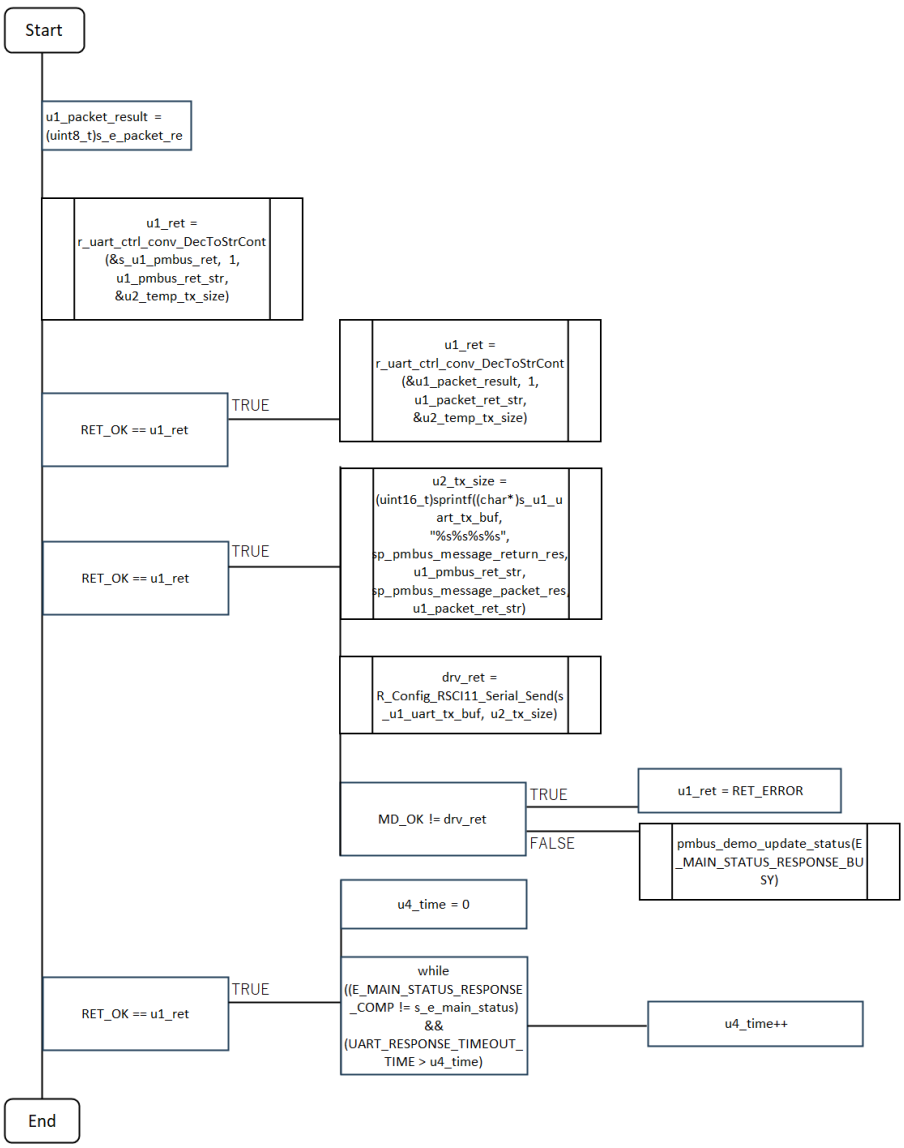
- pmbus_demo_init



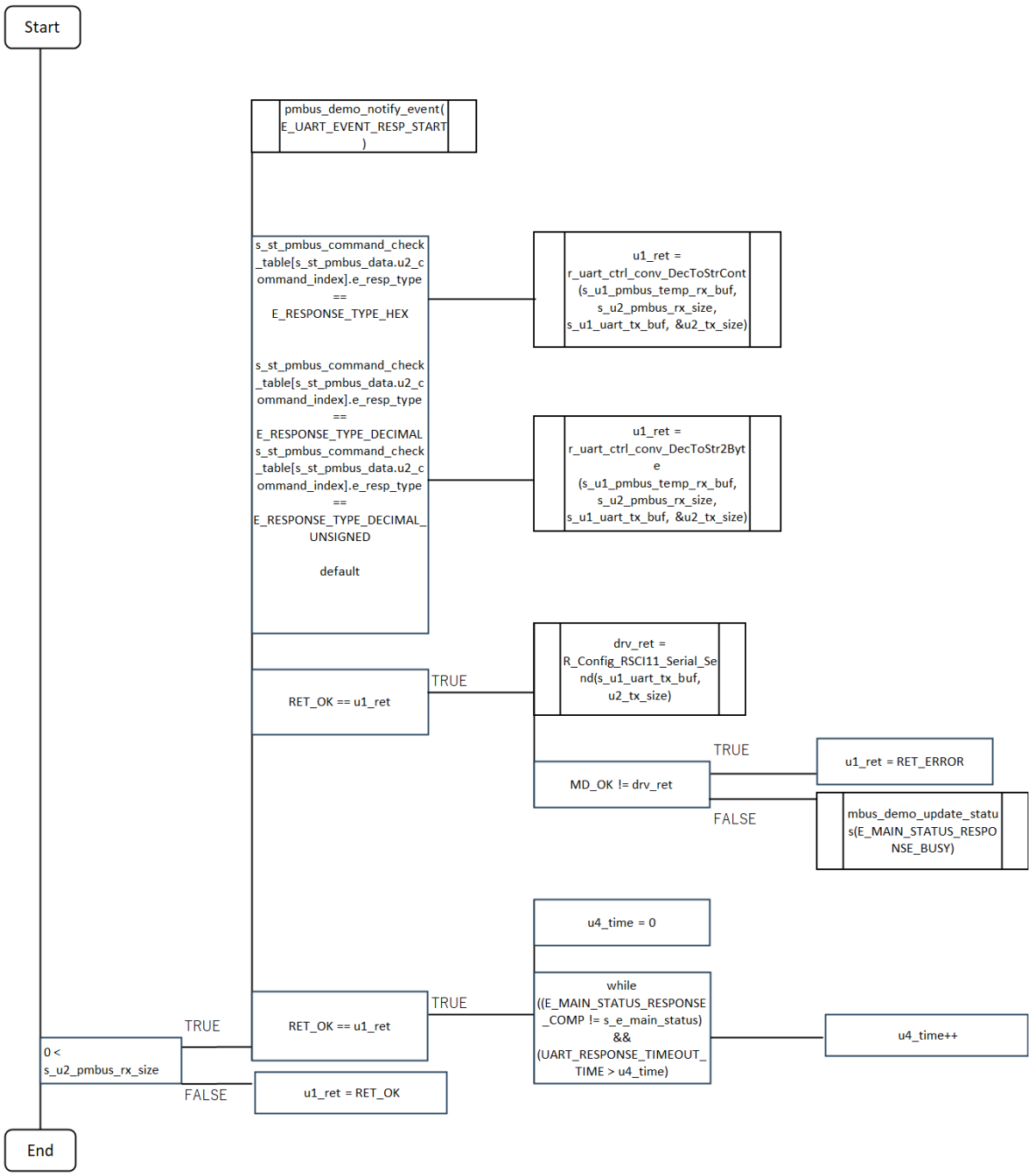
- pmbus_demo_update_status



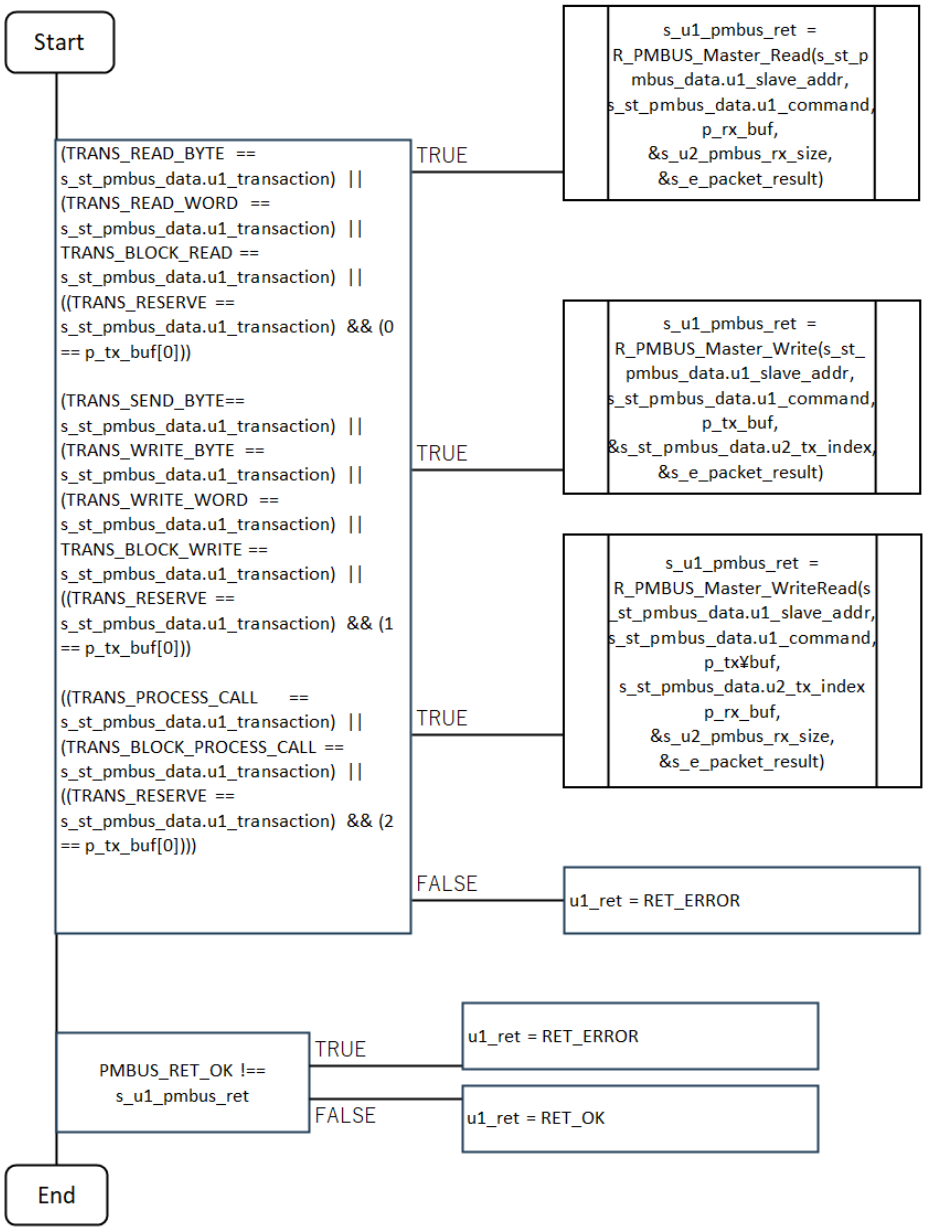
- pmbus_Demo_notify_result



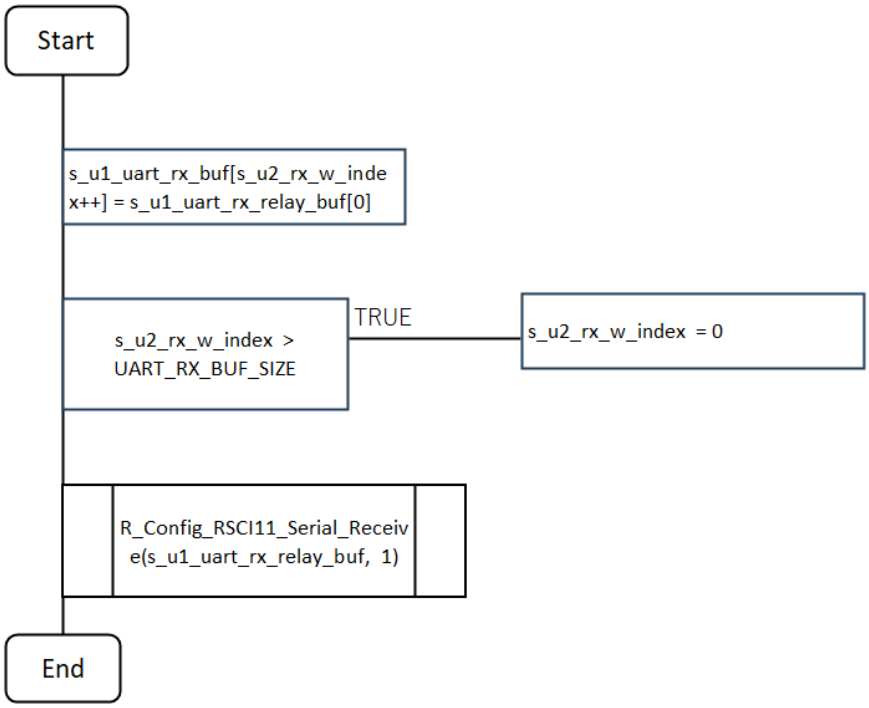
- pmbus_demo_response



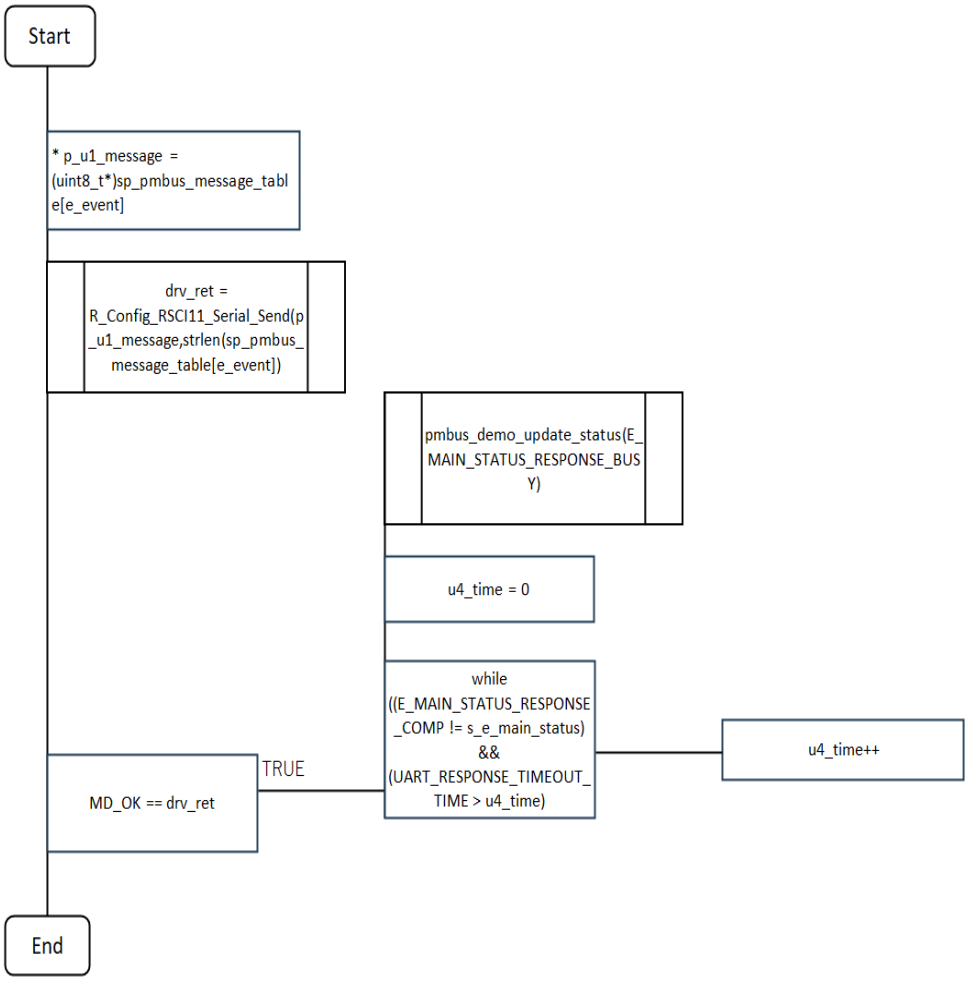
- pmbus_demo_pmbus_execute



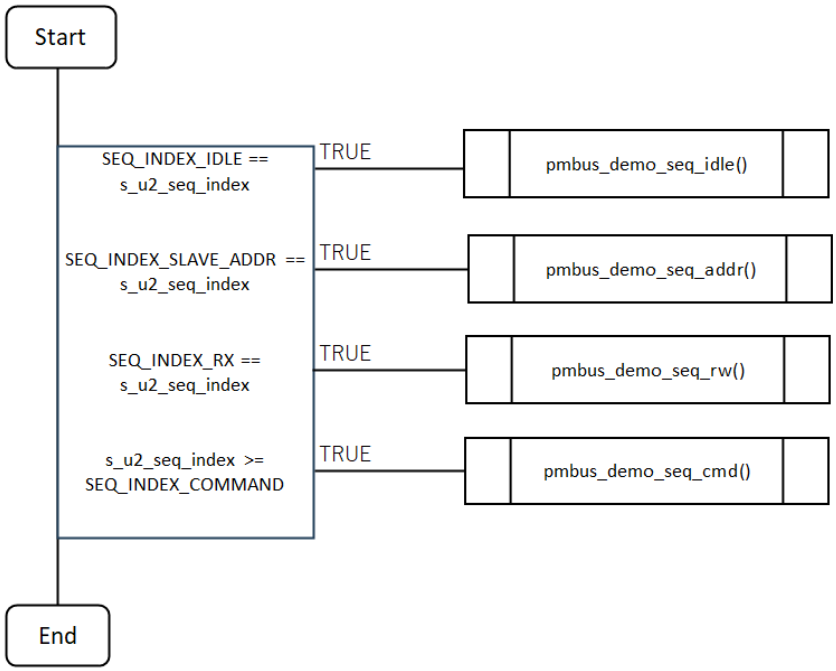
- pmbus_demo_uart_rev_callback



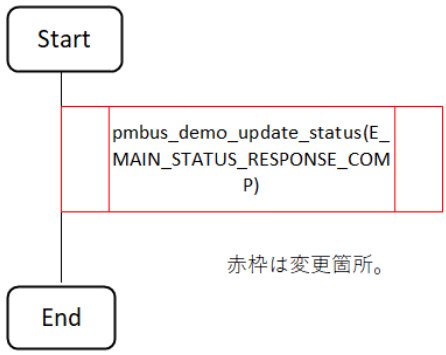
- pmbus_demo_notify_event



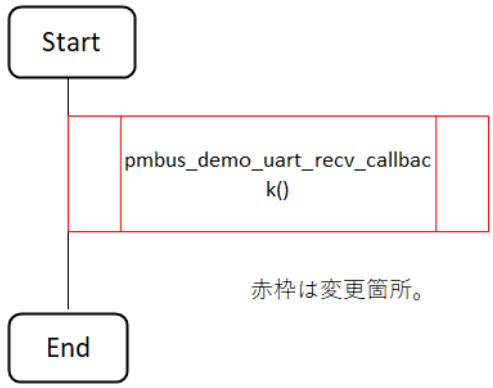
- pmbus_demo_update_sequence



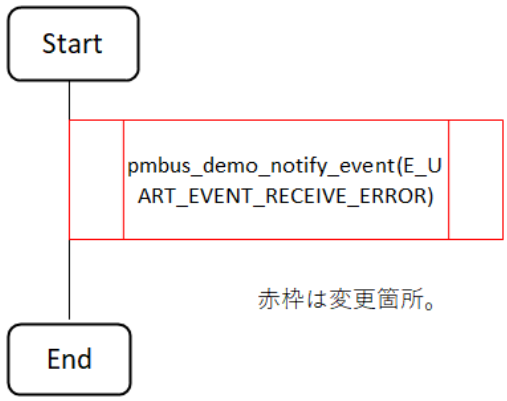
- r_Config_RSCI11_callback_transmitend



- r_Config_RSCI11_callback_receiveend



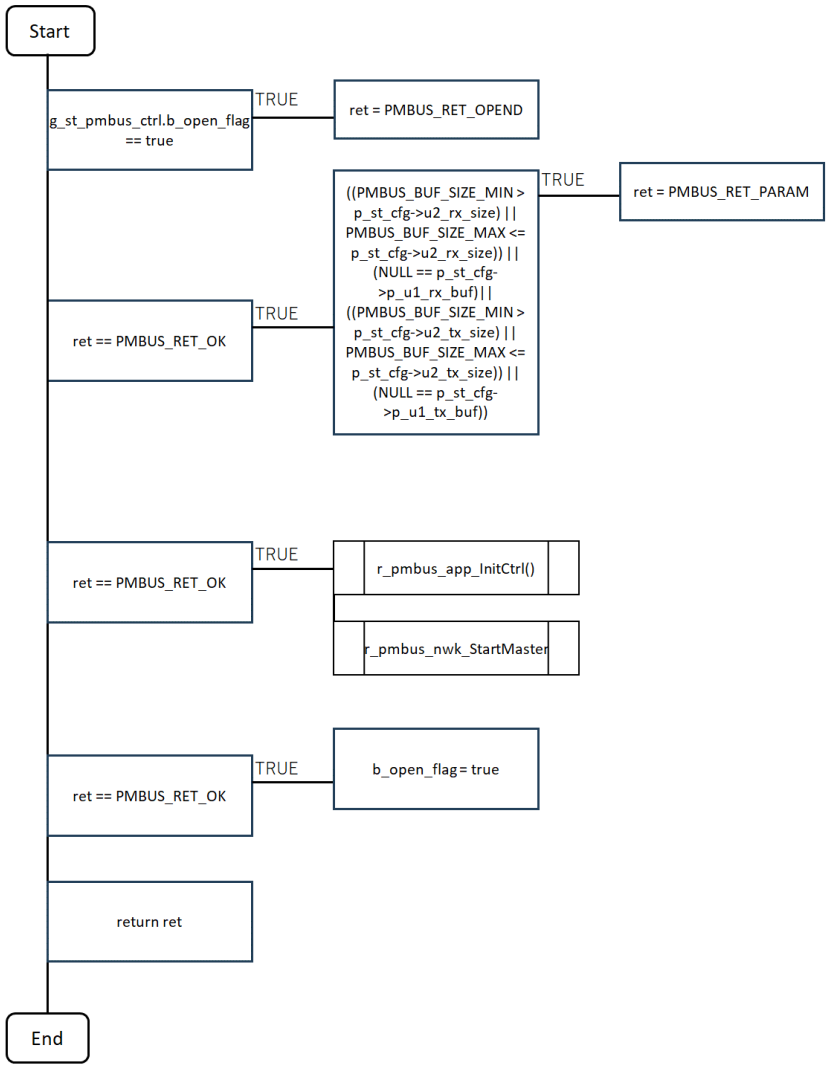
- r_Config_RSCI11_callback_receiveerror



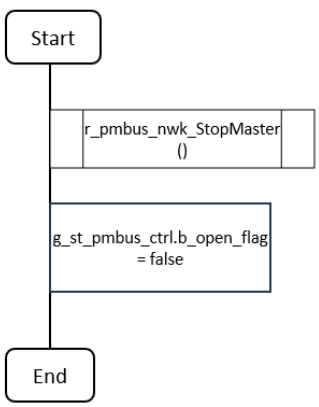
5.1.8.2 PMBus Master API 部フロー

PMBus Master API 部は PMBus Master Middleware 部の制御を行います。PMBus Master API 部のフローを以下に示します。

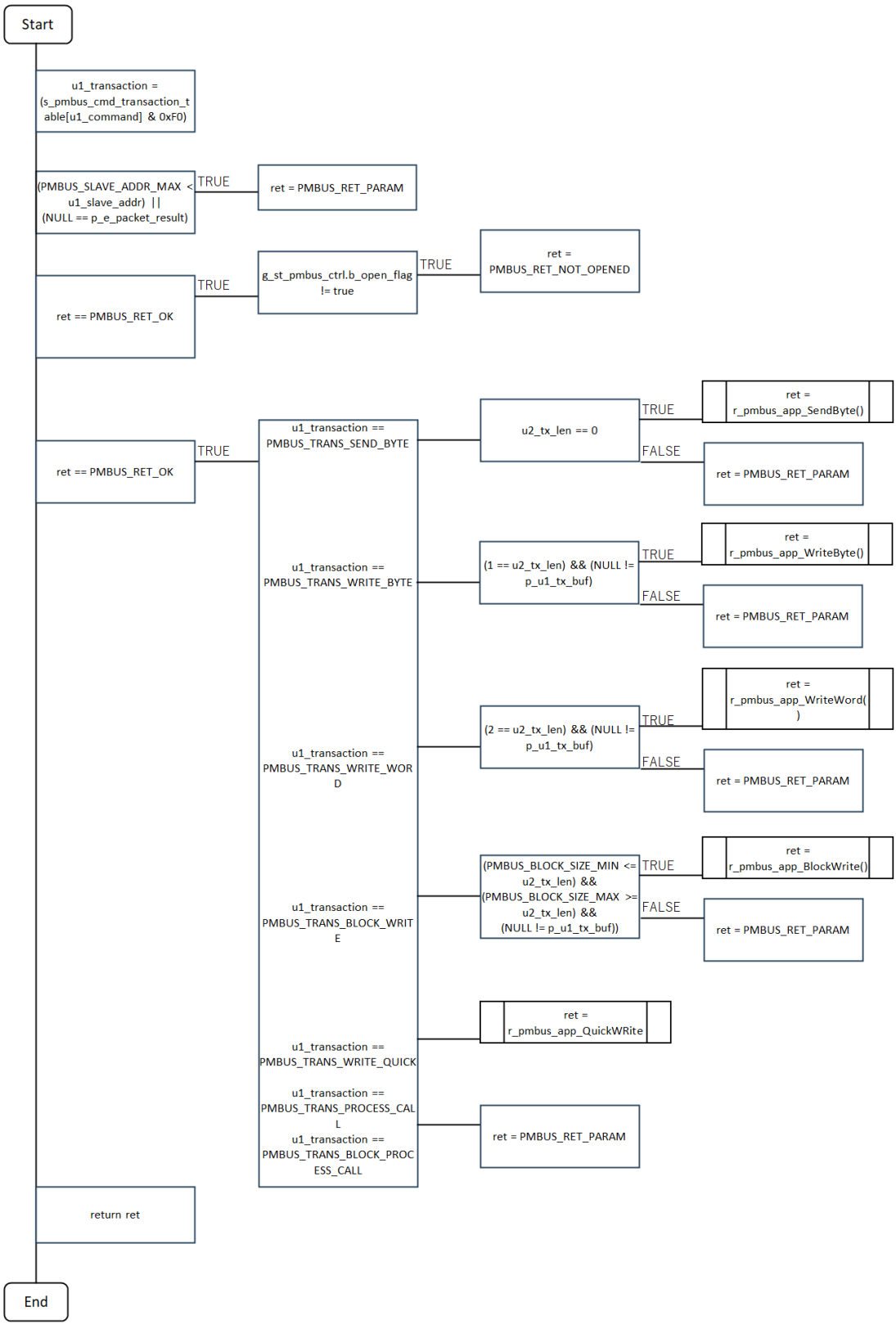
- R_PMBUS_Master_Open



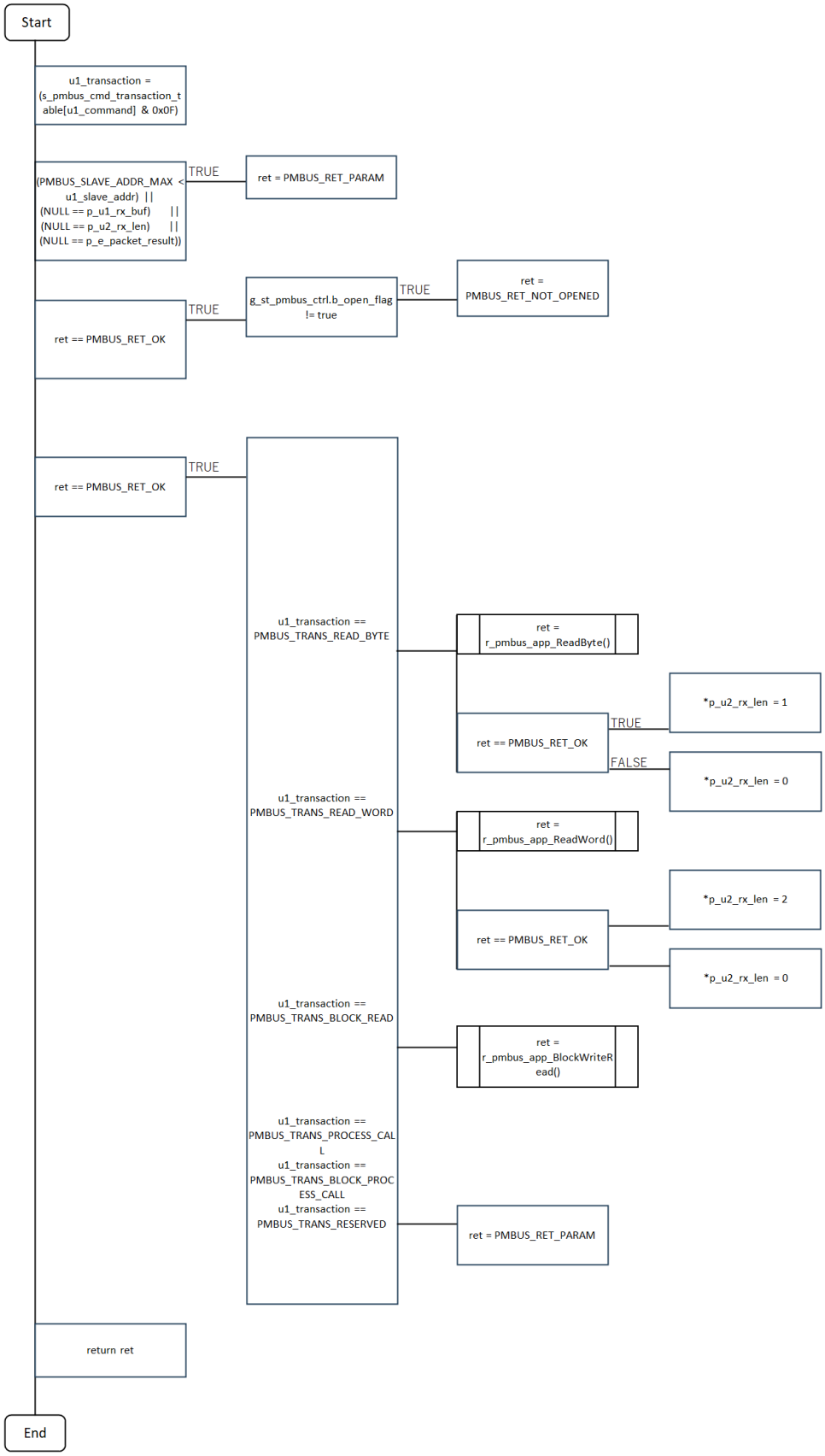
- R_PMBUS_Master_Close



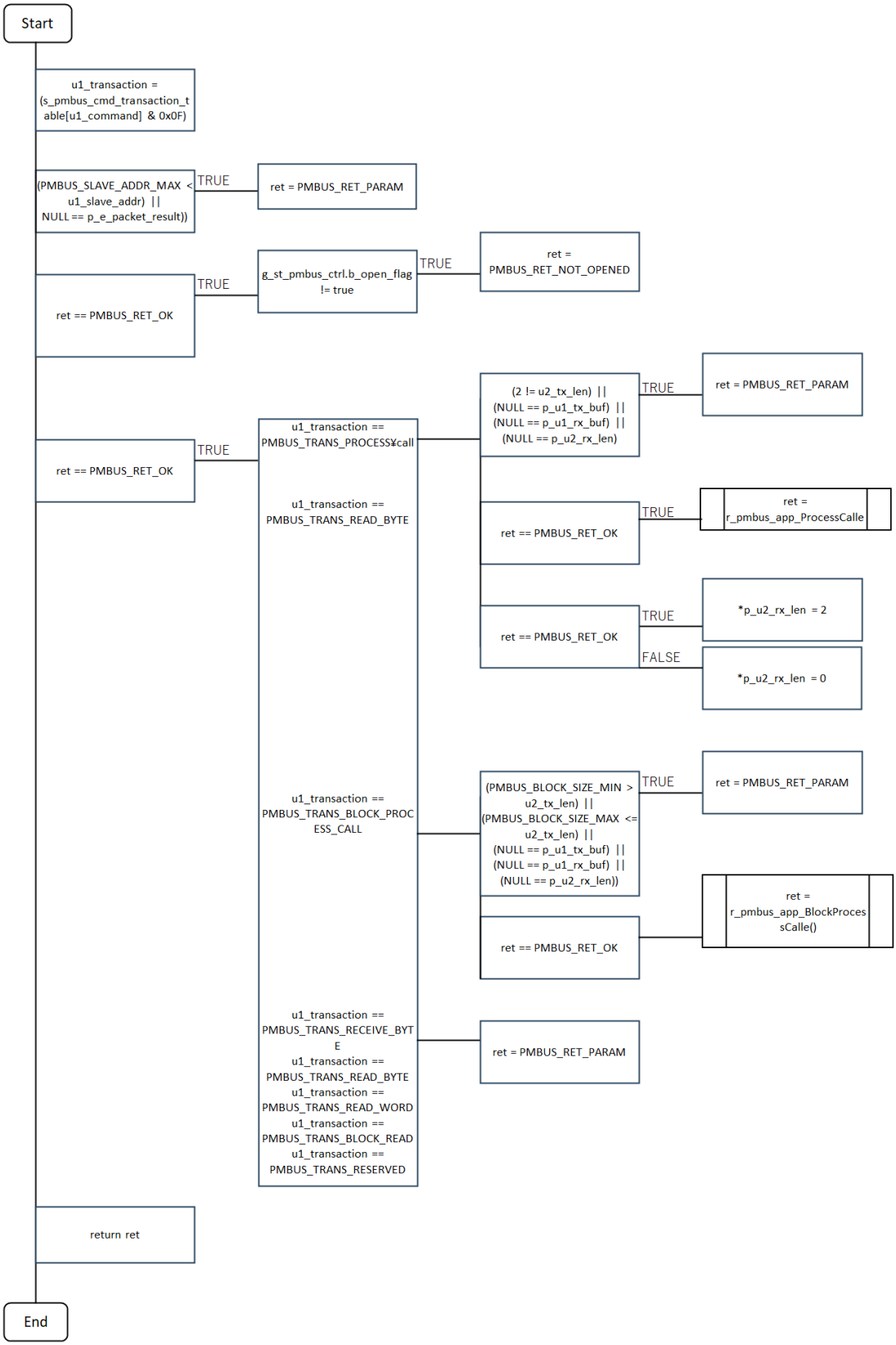
- R_PMBUS_Master_Write



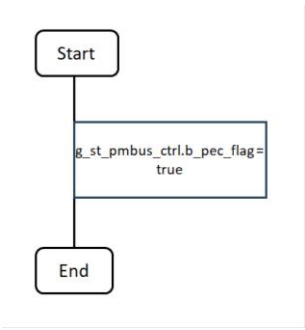
- R_PMBUS_Master_Read



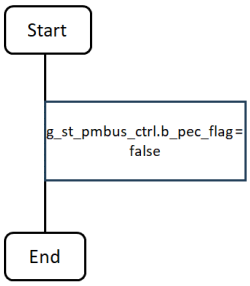
- R_PMBUS_Master_WriteRead



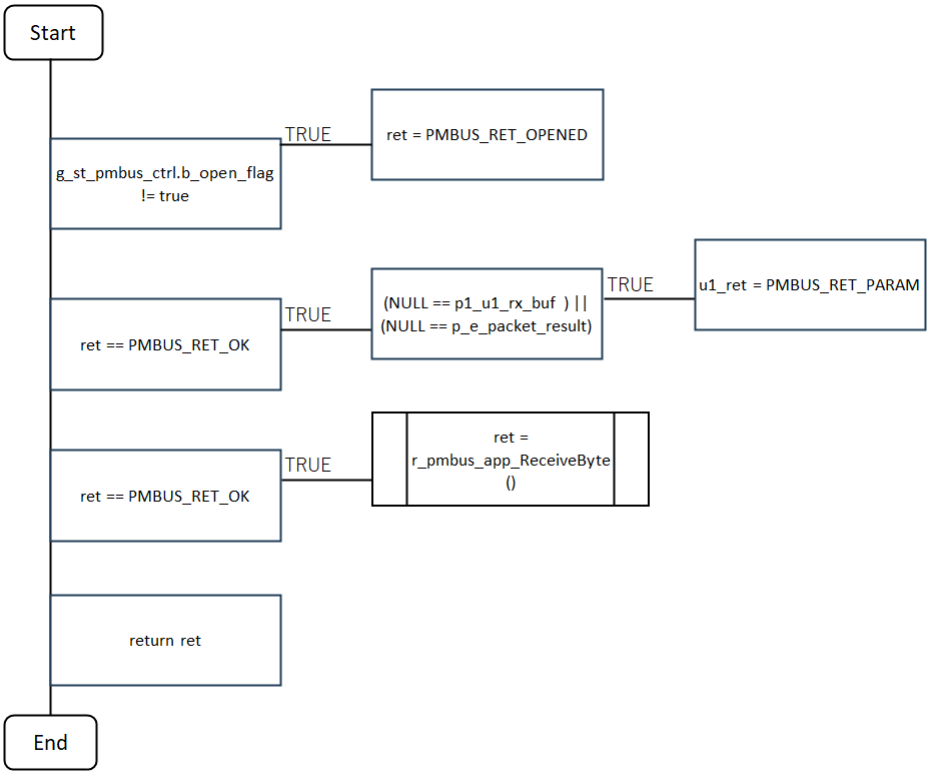
- R_PMBUS_Master_EnablePEC



- R_PMBUS_Master_DisablePEC



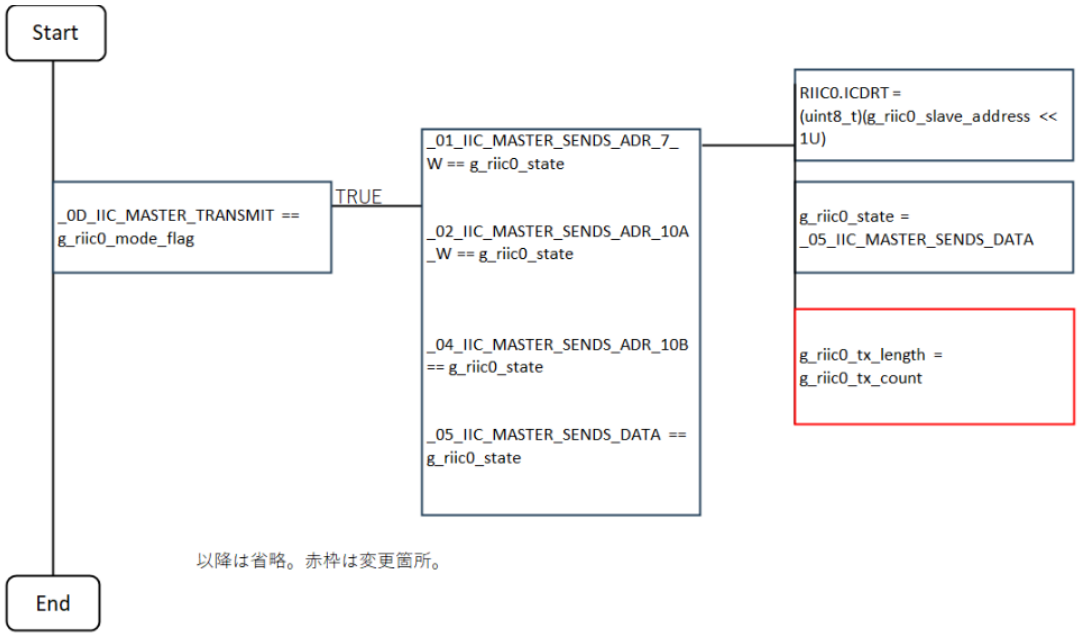
- R_PMBUS_Master_ReceiveARA



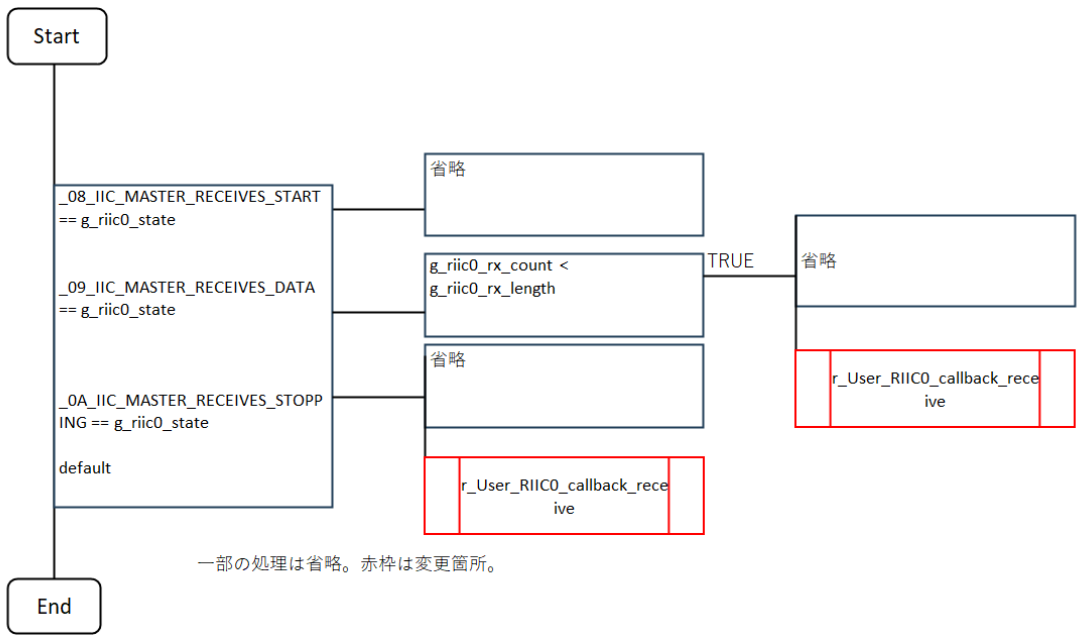
5.1.8.3 PMBus Master ドライバ部フロー

ドライバ部は RX26T ではスマート・コンフィグレータで生成するコードから PMBus Master 処理に合わせて一部変更しています。変更内容については 5.1.4 **PMBus Master Driver 部のカスタマイズ**をご参照ください。各 PAD 図の赤枠部分が修正箇所となります。

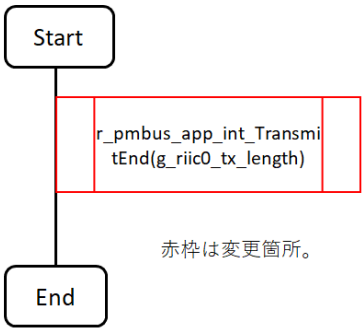
- r_Config_RIIC0_transmit_interrupt



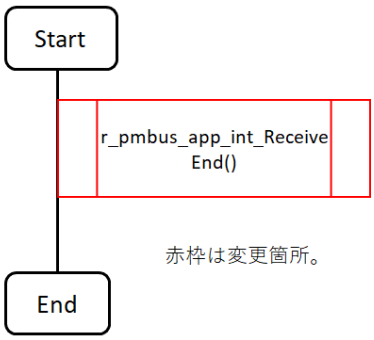
- r_Config_RIIC0_receive_interrupt



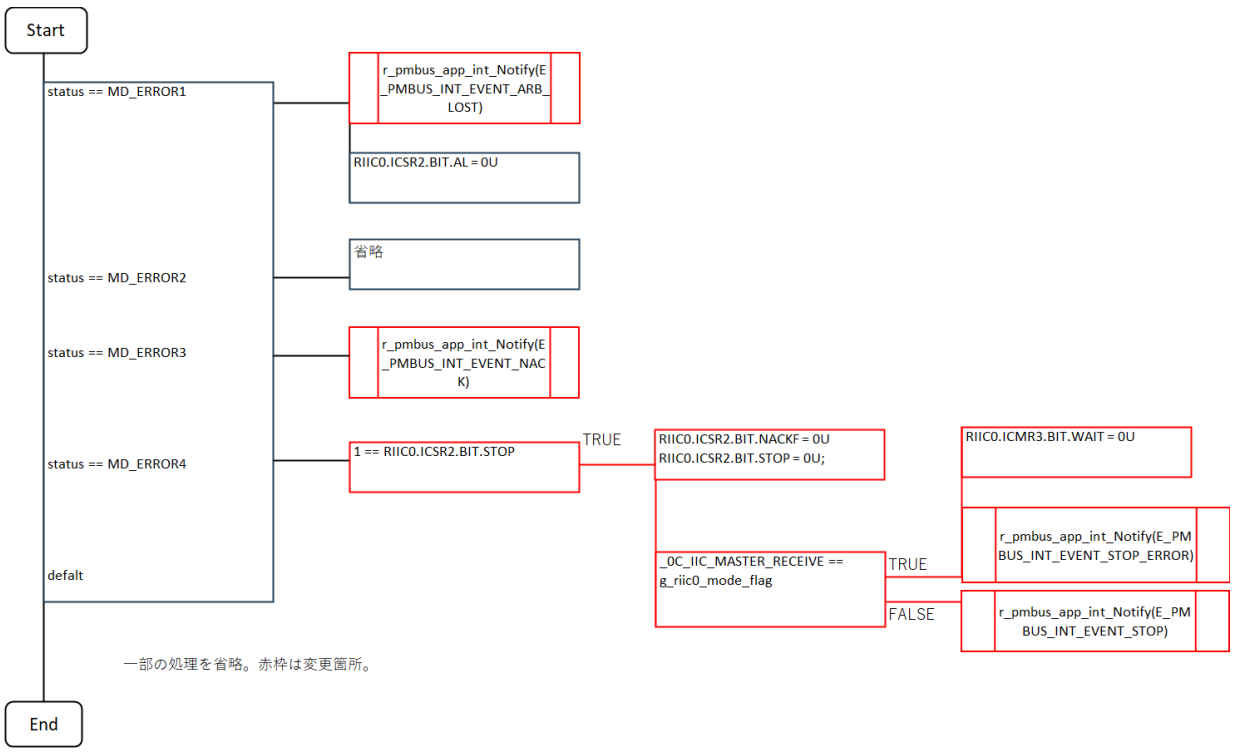
- r_Config_RIIC0_callback_transmitend



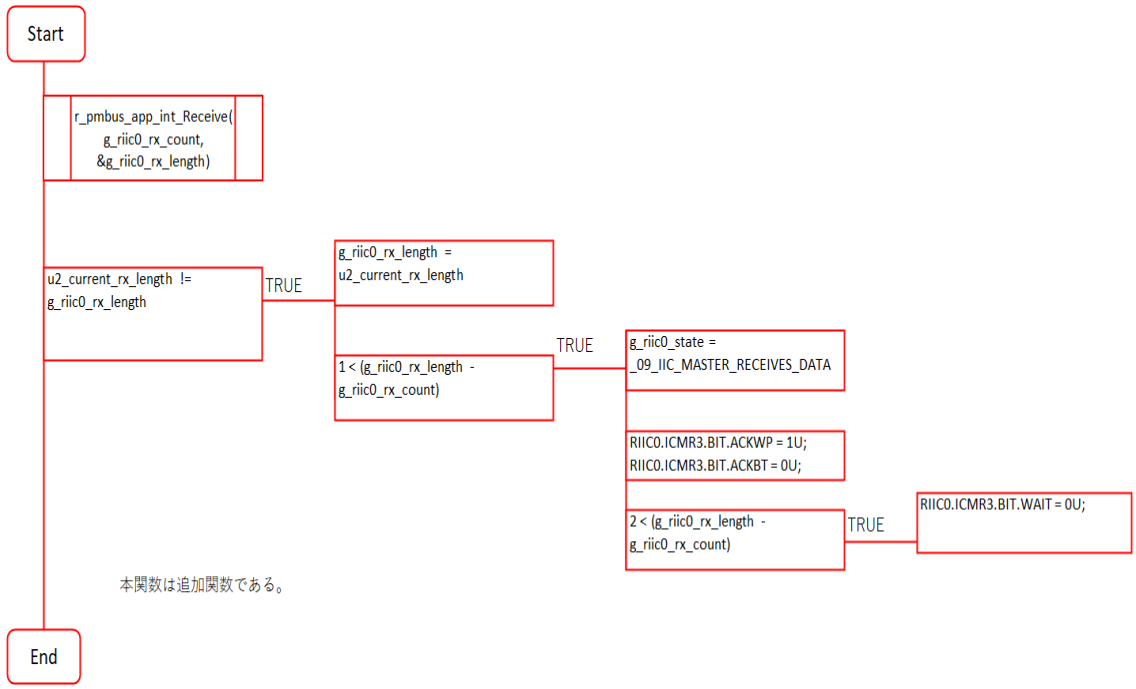
- r_Config_RIIC0_callback_receiveend



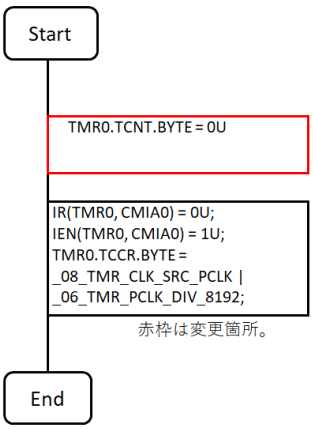
- r_Config_RIIC0_callback_error



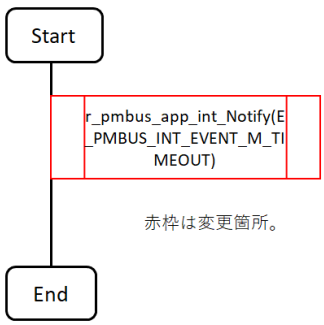
- r_User_RIIC0_callback_receive



- R_Config_TMR0_Start



- r_Config_TMR0_cmia0_interrupt



5.2　PMBus Salve ソフトウェア

PMBus Slave のソフトウェアは、図 21 PMBus Slave ソフトウェアモジュール構成 (RX26T グループ)、および図 22 PMBus Slave ソフトウェアモジュール構成　(RA6T3 グループ)に示す通り、ユーザアプリケーション部、ミドルウェア部、ドライバ部に分類されます。ドライバ部は RX26T グループではスマート・コンフィグレータで生成するソフトウェアを、RA6T3 グループでは FSP で生成するソフトウェアをそれぞれ使用し、一部は PMBus Slave 動作を行う為の修正をしています。RX26T グループのソフトウェアのフォルダ・ファイル構成を表 24 に、RA6T3 グループのソフトウェアのフォルダ・ファイル構成を表 25 に示します。

表 24 PMBus Slave RX26T フォルダ・ファイル構成

フォルダ名	ファイル名	概要
pmbus_app¥	r_app_main.c	PMBus デモシステムのメインプログラム。（ユーザアプリケーション） モータサンプルの main ファイルに PMBus のユーザアプリケーション処理を追加した構成となります。
pmbus_app¥	r_app_board_ui.c	モータサンプルのボード UI 部のモニタとモータ制御を行うプログラム。 既存のモータサンプルに SW1 が ON の場合のモータ回転開始制御をコメント化して使用します。
pmbus_app¥	r_app_main.h	PMBus デモシステムのメインプログラムで使用するヘッダファイル。 モータサンプルの main ファイルに PMBus のユーザアプリケーション処理を追加した構成となります。
pmbus_slave¥	r_pmbus_app_slave.c	PMBus Middleware のアプリケーション層。
pmbus_slave¥	r_pmbus_app_slave.h	PMBus Middleware のアプリケーション層で使用するヘッダファイル。
pmbus_slave¥	r_pmbus_nwk_slave.c	PMBus Middleware のネットワーク層。
pmbus_slave¥	r_pmbus_nwk_slave.h	PMBus Middleware のネットワーク層で使用するヘッダファイル。
pmbus_slave¥	r_pmbus_wrapper_slave.c	PMBus Middleware のドライバ層において、RX26T と RA6T3 のドライバ API の差分を吸収するラッパ関数で使用。
pmbus_slave¥	r_pmbus_wrapper_slave.h	PMBus Middleware のドライバ層のラッパ関数のヘッダファイル。
src¥smc_gen¥Config_RIIC0¥	Config_RIIC0.c	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_RIIC0¥	Config_RIIC0.h	PMBus Middleware のドライバ層使用するヘッダファイル。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_RIIC0¥	Config_RIIC0_user.c	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_TMR0¥	Config_TMR0.c	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_TMR0¥	Config_TMR0.h	PMBus Middleware のドライバ層使用するヘッダファイル。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_TMR0¥	Config_TMR0_user.c	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_CMT0¥	-	モータサンプルで使用するドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_IWDT¥	-	モータサンプルで使用するドライバ層。スマート・コンフィグレータで生成する。

src¥smc_gen¥Config_MOTOR¥	-	モータサンプルで使用するドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_POE¥	-	モータサンプルで使用するドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_PORT¥	-	モータサンプルで使用するドライバ層。スマート・コンフィグレータで生成する。
src¥smc_gen¥Config_S12AD2¥	-	モータサンプルで使用するドライバ層。スマート・コンフィグレータで生成する。
motor_module¥	-	モータサンプルのミドルウェア部
app¥	-	モータサンプルのメインアプリケーション部。 本デモシステムでは main フォルダのファイル、および、board_ui フォルダ内の r_app_board_ui.c はビルド対象から除外します。
app¥cfg¥	r_app_control_cfg.h	モータサンプルのコンフィグレーション情報を定義するファイル。 モータをボード UI で制御するために、"APP_CFG_USE_UI" の値を" MAIN_UI_BOARD"に変更してください。

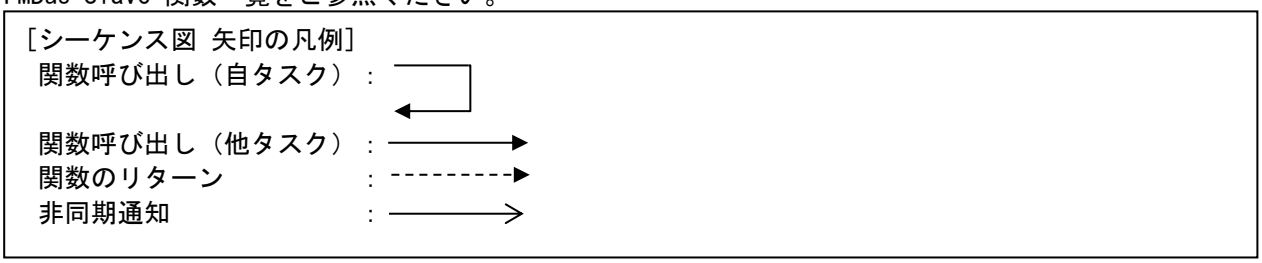
表 25 PMBus Slave RA6T3 フォルダ・ファイル構成

フォルダ名	ファイル名	概要
pmbus_app¥	r_app_pmbus_main.c	PMBus デモシステムのメインプログラム。（ユーザアプリケーション） モータサンプルの main ファイルに PMBus のユーザアプリケーション処理を追加した構成となります。
pmbus_app¥	r_app_pmbus_main.h	PMBus デモシステムのメインプログラムで使用するヘッダファイル。 モータサンプルの main ファイルに PMBus のユーザアプリケーション処理を追加した構成となります。
pmbus_app¥	r_app_control_parameter.h	PMBus デモシステムのメインプログラムで使用するヘッダファイル。 モータサンプルの r_mtr_control_parameter.h のファイル名を変更したのみのファイル。
pmbus_app¥	r_app_motor_parameter.h	PMBus デモシステムのメインプログラムで使用するヘッダファイル。 モータサンプルの r_mtr_moter_parameter.h のファイル名を変更したのみのファイル。
pmbus_slave¥	r_pmbus_app_slave.c	PMBus Middleware のアプリケーション層。
pmbus_slave¥	r_pmbus_app_slave.h	PMBus Middleware のアプリケーション層で使用するヘッダファイル。
pmbus_slave¥	r_pmbus_nwk_slave.c	PMBus Middleware のネットワーク層。
pmbus_slave¥	r_pmbus_nwk_slave.h	PMBus Middleware のネットワーク層で使用するヘッダファイル。
pmbus_slave¥	r_pmbus_wrapper_slave.c	PMBus Middleware のドライバ層において、RX26T と RA6T3 のドライバ API の差分を吸収するラッパ関数で使用。
pmbus_slave¥	r_pmbus_wrapper_slave.h	PMBus Middleware のドライバ層のラッパ関数のヘッダファイル。
pmbus_slave¥	r_smbus_slave.c	PMBus Middleware の SMBus ドライバ層。 FSP の r_iic_b_slave.c に相当します。

pmbus_slave¥	r_smbus_slave.h	PMBus Middleware の SMBus ドライバ層のヘッダファイル。 FSP の r_iic_b_slave.h に相当します。
pmbus_slave¥	r_smbus_slave_api.h	PMBus Middleware の SMBus ドライバ層のヘッダファイル。 FSP の r_i2c_slave_api.h に相当します。
pmbus_slave¥	r_smbus_slave_cfg.h	PMBus Middleware の SMBus ドライバ層のヘッダファイル。 FSP の r_iic_b_slave_cfg.h に相当します。
ra¥fsp¥src¥r_gpt¥	r_gpt.c	FSP で生成する PMBus Middleware および、モータサンプルで使用するドライバ。
ra¥fsp¥inc¥api¥	r_tiemr_api.h	FSP で生成する PMBus Middleware および、モータサンプルで使用するドライバ層のヘッダファイル。
ra¥fsp¥inc¥instance¥	r_gpt.h	FSP で生成する PMBus Middleware および、モータサンプルで使用するドライバ層のヘッダファイル。
ra¥	-	FSP で生成するモータサンプルで使用するドライバを含む各種ファイル。
ra_cfg¥	-	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
ra_gen¥	-	PMBus Middleware のドライバ層。スマート・コンフィグレータで生成する。
motor_module¥	-	モータサンプルのミドルウェア部
src¥	-	モータサンプルのメインアプリケーション部。 本デモシステムでは src¥application¥main フォルダのファイルはビルド対象から除外します。

5.2.1　PMBus Slave 動作シーケンス

PMBus のコマンドに合わせた Write 動作、Read 動作、Write/Read 動作、および Alert Response 動作について、Write 動作シーケンスを図 32 と図 33 に、Read 動作シーケンスと Write/Read 動作シーケンスを図 34 と図 35 に、Alert Response 動作を図 36 と図 37 に示します。各々動作で使用する API 関数は 5.2.3 節の PMBus Slave 関数一覧をご参照ください。



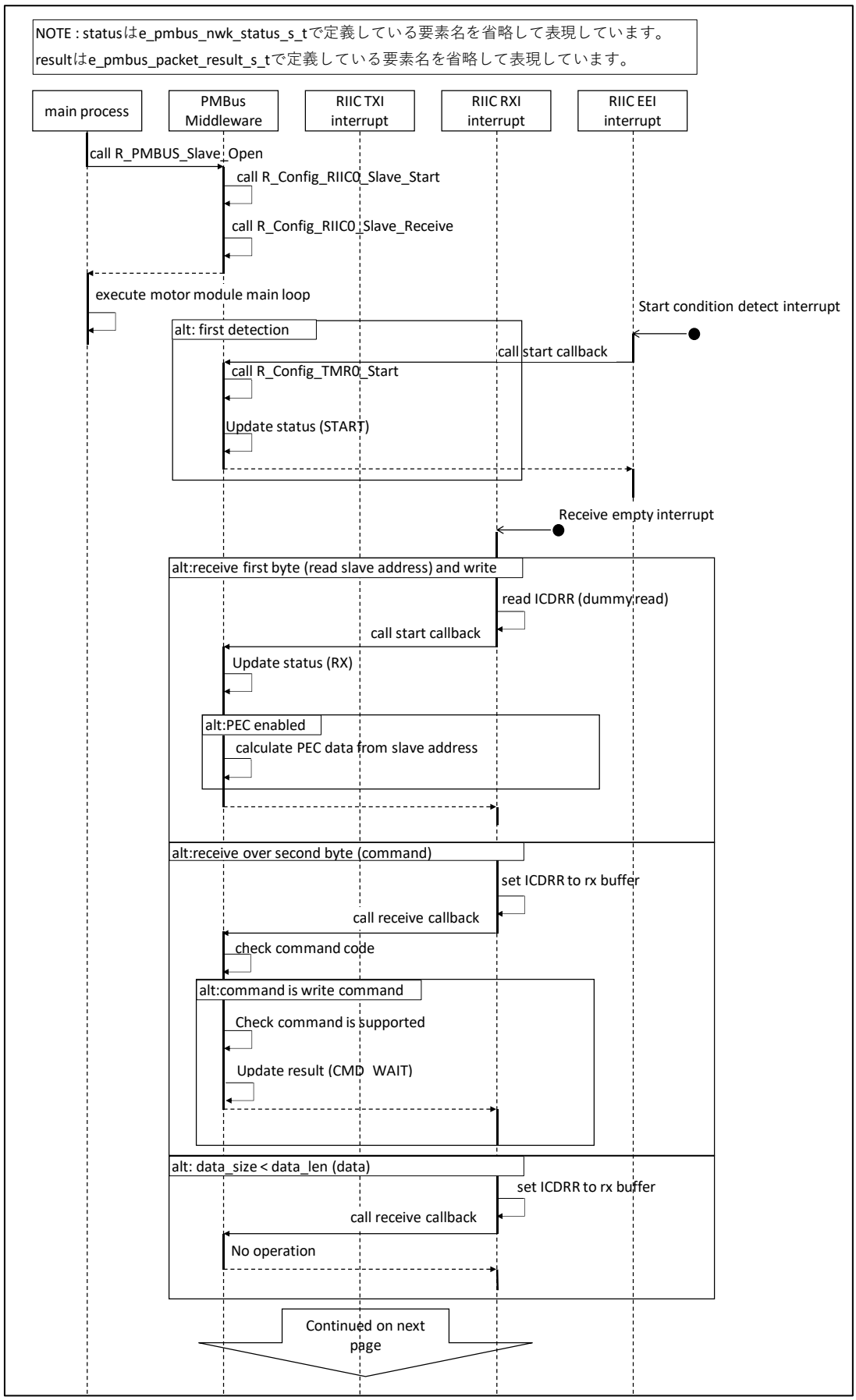


図 32 PMBus Slave Write 系プロトコルのシーケンス図 (1/2)

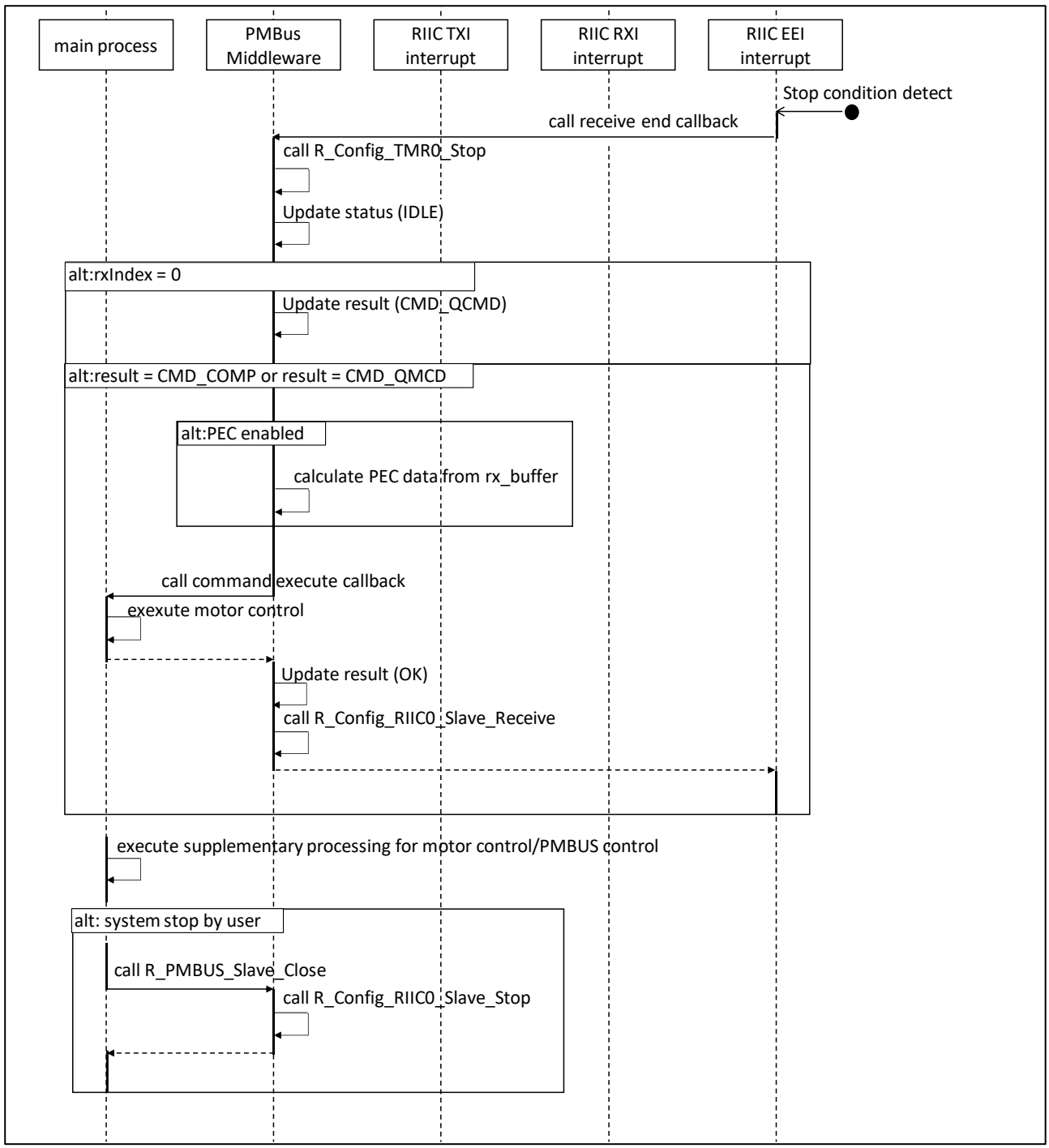


図 33 PMBus Slave Write 系プロトコルのシーケンス図 (2/2)

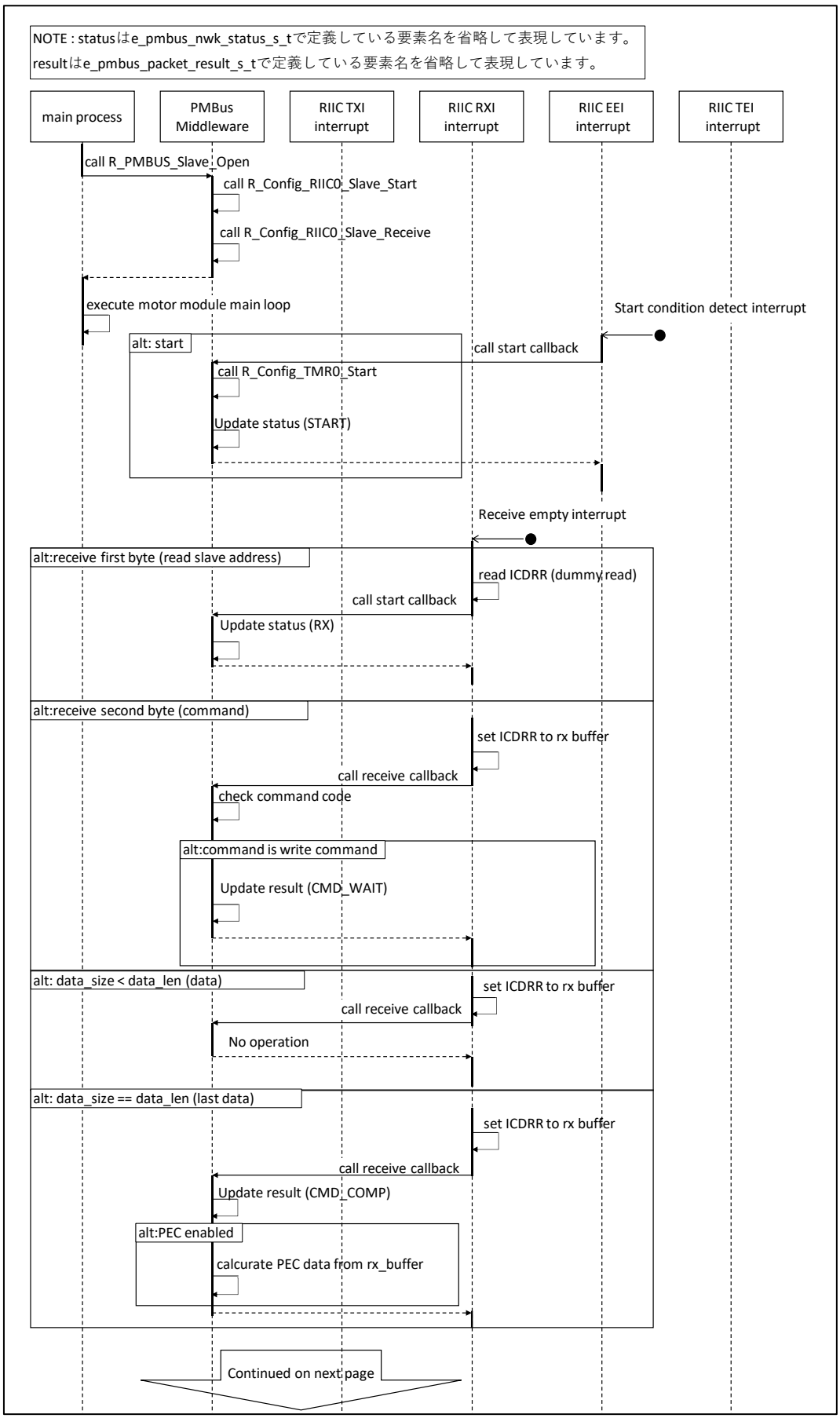


図 34 PMBus Slave Read 系および Write Read 系プロトコルのシーケンス図 (1/2)

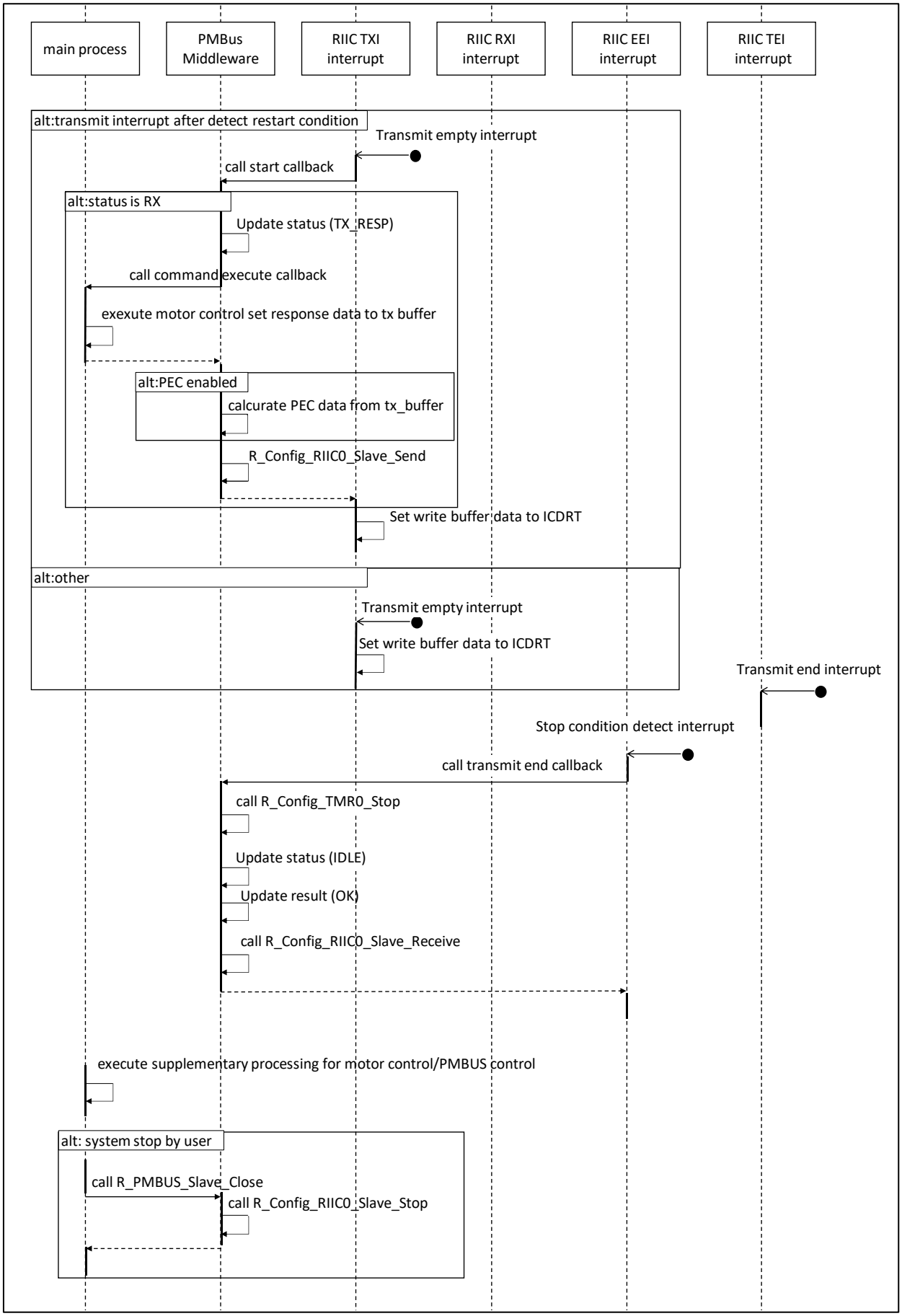


図 35 PMBus Slave Read 系および Write Read 系プロトコルのシーケンス図 (2/2)

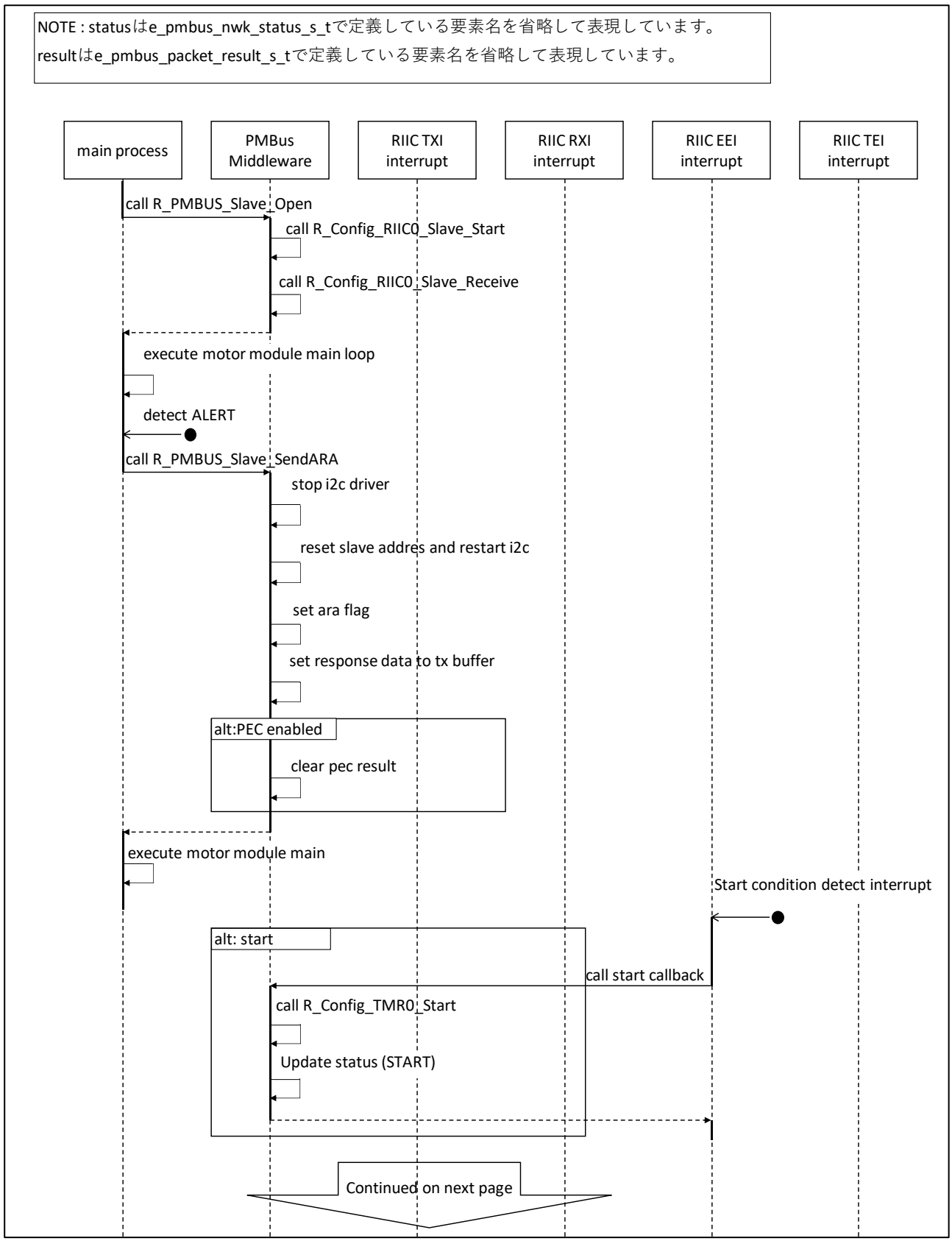


図 36 PMBus Slave Alert Response Address プロトコルのシーケンス図 (1/2)

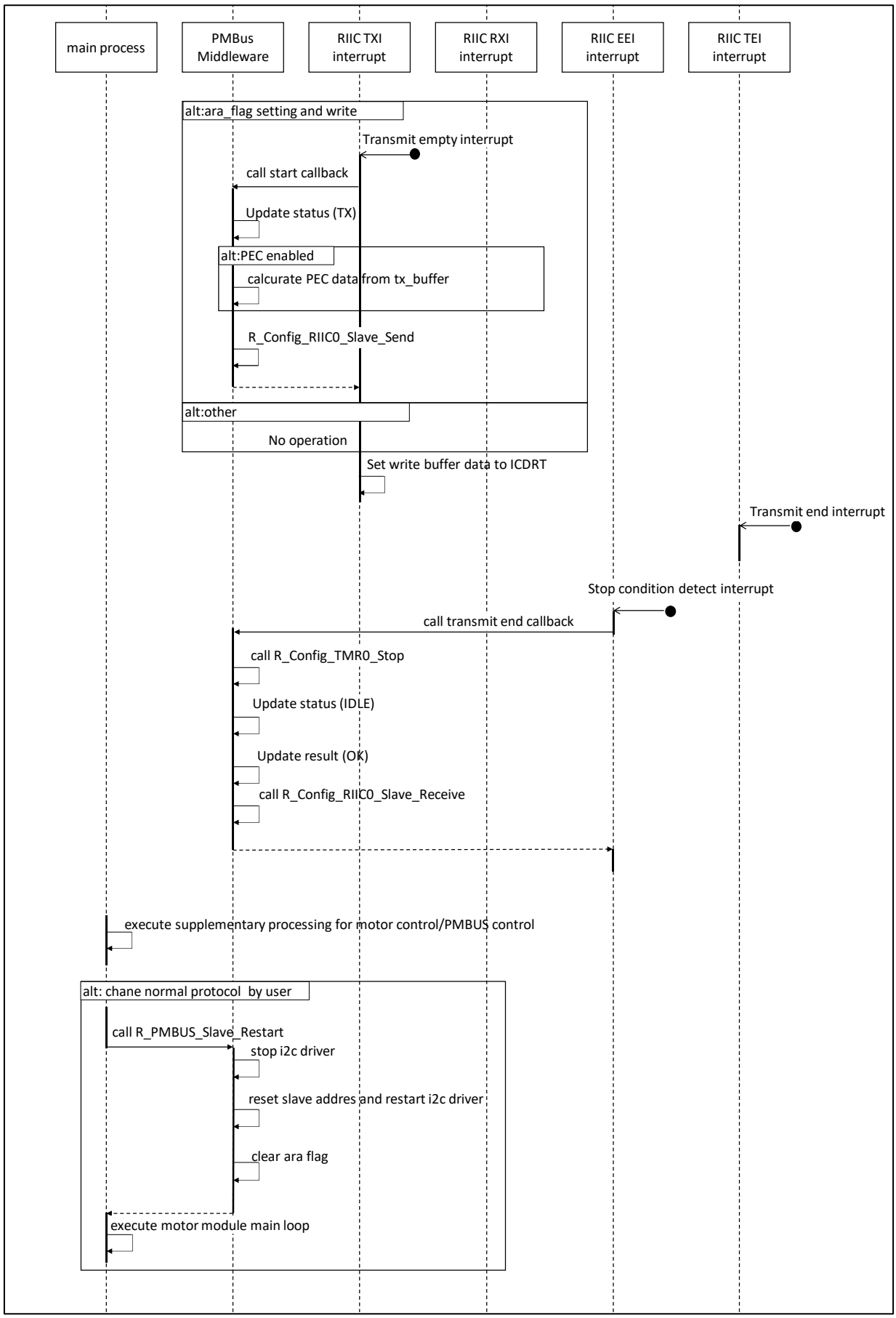


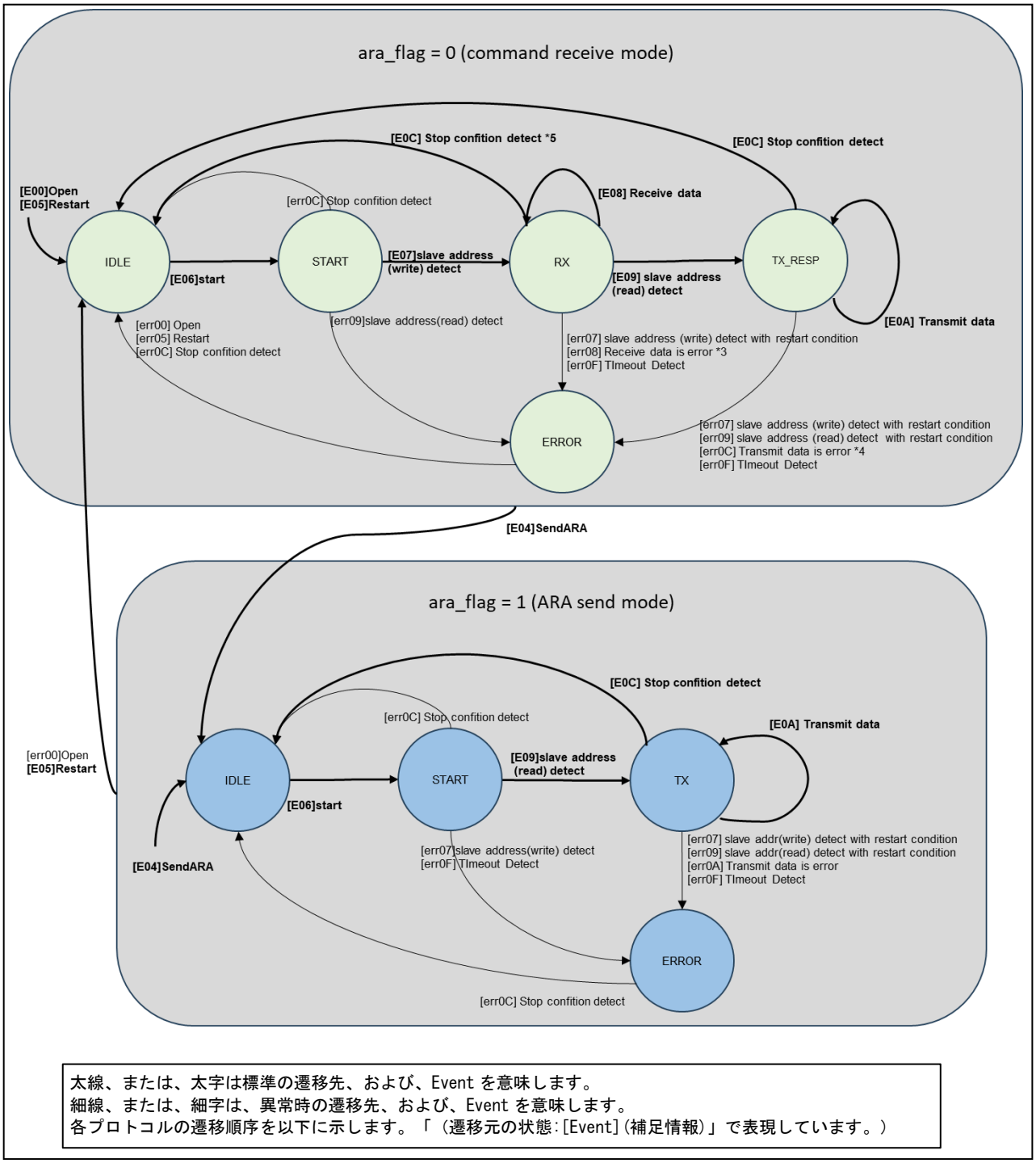
図 37 PMBus Slave Alert Response Address プロトコルのシーケンス図 (2/2)

5.2.2　PMBus Slave 状態遷移

PMBus Slave ミドルウェアの状態管理は、Application Layer と Driver Layer の各々で状態を管理しています。Application Layer はプロトコル状態の管理、Driver Layer はデータ送受信数の管理となります。Applicationi Layer の状態遷移を 5.2.2.1 節に、Driver Layer の状態遷移を 5.2.2.2 節に示します。

5.2.2.1 PMBus Slave Middleware Application Layer 状態遷移

PMBus Slave の Middleware Application Layer 状態遷移は、PMBus Master からの受信に備え、IDLE の状態で待機しています。その後 PMBus Master からコマンドが送付されるとコマンドコードに合わせて、送信、受信、およびエラー処理の状態を管理します。以下図 38、図 39、図 40、図 41、表 26、表 27、表 28 に PMBus Slave の Application Layer の状態遷移を示します。



太線、または、太字は標準の遷移先、および、Event を意味します。
細線、または、細字は、異常時の遷移先、および、Event を意味します。
各プロトコルの遷移順序を以下に示します。「（遷移元の状態:[Event](補足情報)」で表現しています。）

図 38 PMBus Slave Middleware Application Layer 状態遷移図

[Event 一覧]
　※[Exx]は通常の Event、[errxx]は異常発生の場合の Event を意味する。
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. 状態遷移は発生しない Event。
*2. スレーブアドレス受信時に、Receive Buffer full interrupt か Transmit Buffer empty interrupt が発生するかは、
　ハードウェアがスレーブアドレスの RW ビットににより決定する。
*3. 以下の場合は PACKET RESULT にエラーコードを設定し、ERROR へ遷移する。
　・受信データ数が受信バッファサイズを超えた場合。PACKET RESULT=DATA_SIZE
　・受信バッファが登録されていない場合。PACKET RESULT＝NOT_READY
　・1 回目に受信したデータ（コマンド）が未サポートの場合。PACKET RESULT=CMD_NOT_SUPPORT
*4. 以下の場合は PACKET RESULT にエラーコードを設定し、ERROR へ遷移する。
　・送信バッファが登録されていない。PACKET RESULT = NOT_READY
*5. PACKET RESULT を確認しコールバックを実行する。コールバックを実行する条件は PACKET RESULT の状態遷移図を参照。

図 39 PMBus Slave Application Layer 状態遷移図(図 38 の補足)

- PACKET RESULT の状態遷移

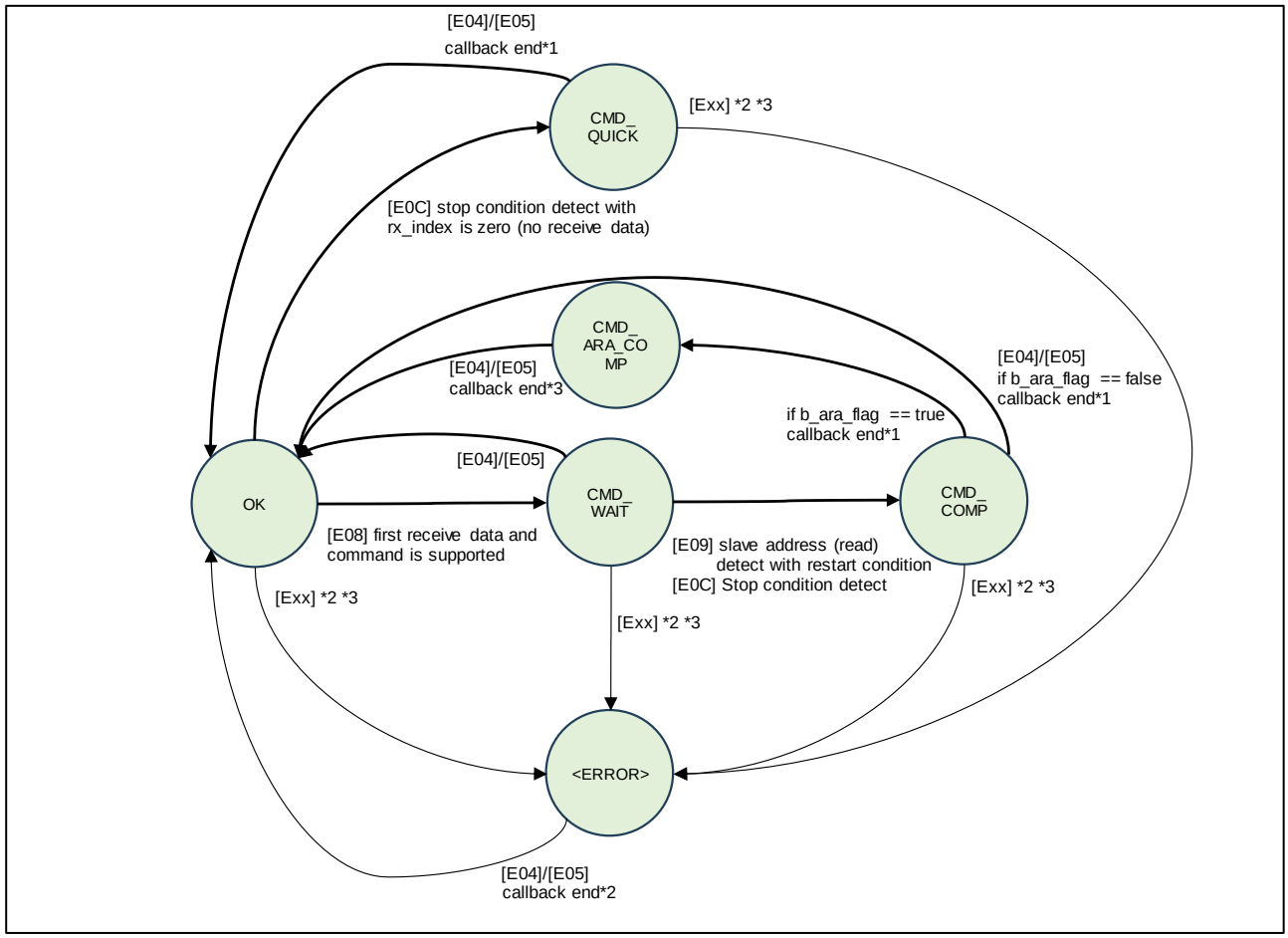


図 40 PMBus Slave Application Layer 状態遷移図 PACKET RESULT

*1. コールバック関数を呼び出しコマンド処理を実施する。
*2. [Exx]は以下のイベントのいずれかを意味する。
　[E07/err07]
　[E08/err08]
　[E09/err09]
　[E0A/err0A]
*3. コールバック関数を呼び出しエラー通知を実施する。
　<ERROR>は以下のいずれかを意味する。
　　DATA_SIZE_ERROR
　　PEC_ERROR
　　TIMEOUT
　　INTERNAL_ERROR
　　NOT_READY
　　CMD_NOT_SUPPORT
*4. コールバック関数を呼び出し ARA 応答完了処理を実施する。

図 41 PMBus Slave Application Layer 状態遷移図 PACKET RESULT（図 40 の補足）

表 26 PMBus Slave Application Layer の状態遷移表(ara_flag=0 時)

本表は ara_flag が 0 の場合の状態遷移表を示します。
注釈の説明は表 27 にまとめて記載しています。

本表および表 27 の見方は以下の通りとなります。
・event は API 名の省略表現と、割り込み要因で構成します。
・status は"e_pmbus_nwk_status_s_t"の要素名の省略表現で構成します。
・「If (<条件>)」は条件付きで遷移することを意味します。
・「→ <状態>」は状態に遷移することを意味します。
・「ERROR (<エラー名>)」は API の返却値を意味します。
・「PACKET RESULT(<エラー名>)」は、API の引数 p_e_packet_result)に格納するエラー情報を意味します。
・「-」は状態遷移しないことを意味します。
・薄緑は図 38 における ara_flag=0 の場合の状態遷移を意味します。
・薄青は図 38 における ara_flag=1 の場合の状態遷移を意味します。

	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	-

表 27 PMBus Slave Application Layer の状態遷移表(ara_flag=1 時)

本表は ara_flag が 1 の場合の状態遷移表を示す。

	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

*1. 状態遷移後、ユーザコールバックを実行。
*2. 応答データを送信開始。
*3. コールバックを実行後、次のコマンド受信開始。
*4. コールバックを実行し、ユーザに未サポートコマンドであることを通知する。
*5. ERROR 状態遷移時後にコールバックを実行し、ユーザにエラーの発生を通知する。
*6. ERROR 状態遷移後にコールバックを実行し、すぐに Restart 処理を実行し IDLE 状態に遷移する。
*7. Close 後に Open した場合のみ Idle に遷移する。

表 28 PACKET_RESULT の状態遷移表

本表は、図 40 に対応した PACKET_RESULT の状態遷移表をします。

表の見方は以下の通りとなります。
・event は API 名の省略表現と、割り込み要因で構成します。
・status は"e_pmbus_nwk_status_s_t"の要素名の省略表現で構成します。
・「If (<条件>)」は条件付きで遷移することを意味します。
・「→ <状態>」は状態に遷移することを意味します。
・「-」は状態遷移しないことを意味します。
・薄緑は図 40 における状態遷移を意味します。

	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>				
EOA/err0A_Interrupt (Transmit Interrupt)	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	→ <ERROR>	
EOB_Interrupt (Transmit End Interrupt)	-	-	-	-	-	-
EOC/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

*<ERROR>の詳細要因の発生条件は、表 26、表 27 に示す PMBus Slave Application Layer の状態遷移表を参照。

5.2.2.2 PMBus Slave Driver Layer 状態遷移

PMBus Slave Driver Layer の状態遷移は、PMBus Master への送信動作部と受信動作部に分かれ、各々指定のデータを指定のバイト数だけ送受信します。PMBus Slave Driver Layer の状態遷移を図 42、図 43、表 29、表 30 に示します。

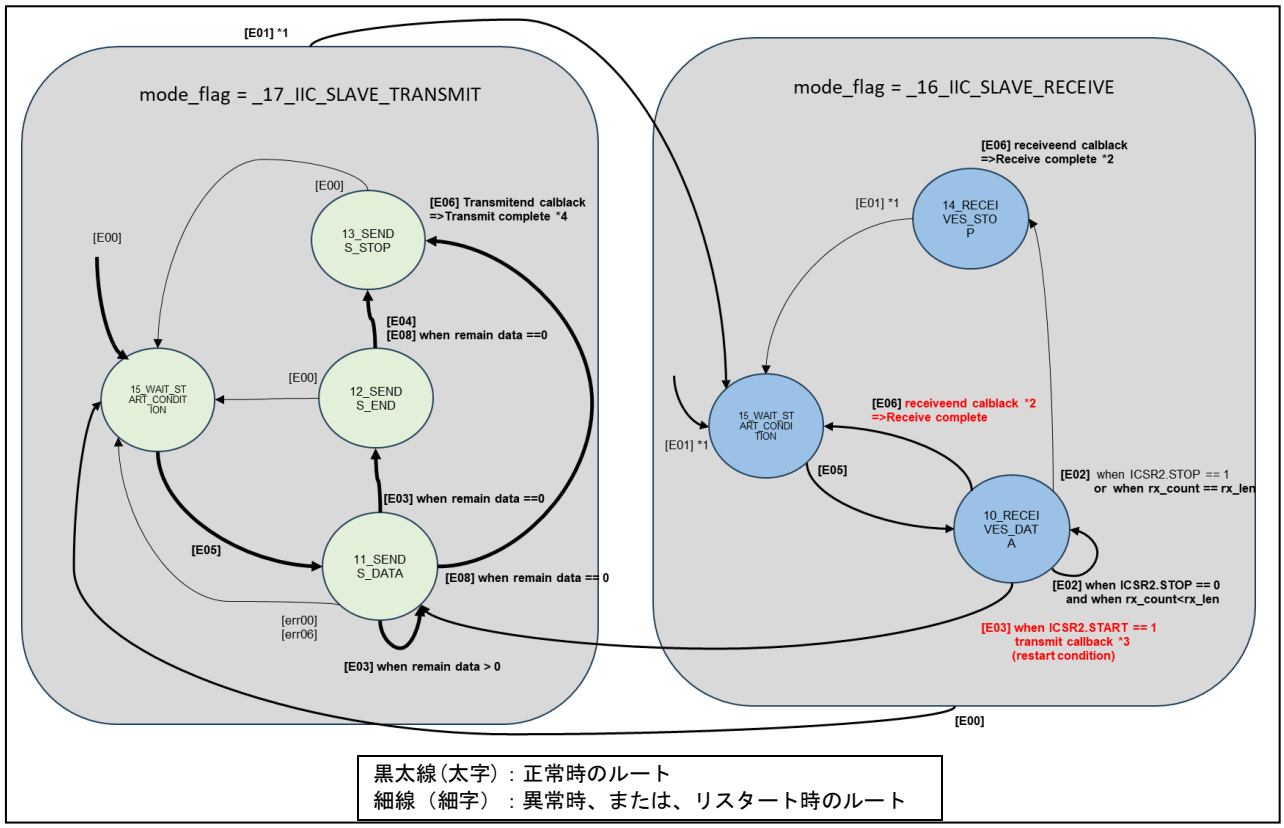


図 42 PMBus Slave Driver Layer 状態遷移図

[Event 一覧]
※[Exx]は通常の Event、[errxx]は異常発生の場合の Event を意味する。
一覧にあり図中に無い異常 Event はどの状態でも発生し、callback error によりエラーを通知する。
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 デモシステムとしては通常は Receive を実行し、master からの read 要求まで待機する。
Receive API に指定する受信データ数初期値(total_data_size)は,受信途中で STOP コンディション受信待ちに遷移しないように、Block Write プロトコルの最大数である 35(command(1)+Data Size(1)+Data(32)+PEC(1))を指定する。
*2. 14 に遷移する前に Stop コンディションを検出した場合は、受信動作未完了となるため、受信バッファの設定を初期化するために callback receivend 内で Receive Event[E01]を発生させる。
受信データ最大数まで受信した場合も、callback receiveend 内で Receive Event[E01]を発生させ、受信バッファの設定を初期化する。
*3. callback transmit 内で Send Event[E00]を発生させ、mode_flag を 17 に更新したのち、ステータスを 11 に更新する。（リスタートコンディションの検出に対応。）
*4. transmitend callback 内で、次の通信に備え Receive Event[E01] (PMBUS middleware が command receive mode の場合)、または、Send Event[E00] (PMBus middleware が ARA send mode の場合)を発生させる。

図 43 PMBus Slave Driver 部状態遷移図(図 42 の補足)

表 29 PMBus Slave Driver Layer の状態遷移表（送信時）

本表は mode_flag が_17_IIC_SLAVE_TRANSMIT の場合の状態遷移表を示します。
注釈の説明は表 30 にまとめて記載しています。

本表および表 30 の見方は以下の通りとなります。
・スレーブアドレス 10 ビットは PMBus では使用しないため、状態管理を省略しています。
・RIIC0 のタイムアウト検出割り込みは PMBus では使用しないため、状態管理を省略しています。
・→<番号>は遷移先の番号。(<番号>)は mode_flag の遷移先の番号を指します。
・「-」は状態遷移しないことを意味します。
・callback <xxx>はコールバック関数を実行することを指します。callback error (<番号>)は callback error に渡すエラー情報を意味します。
・If (<条件>)は条件付きの動作を指します。
・赤文字は PMBus 用に変更した処理を意味します。
・薄緑は図 42 における mode_flag=17 の場合の状態遷移を意味する。
・薄青は図 42 における mode_flag=16 の場合の状態遷移を意味する。

	_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)

表 30 PMBus Slave Driver Layer の状態遷移表（受信時）

本表は mode_flag が_16_IIC_SLAVE_RECEIVE の場合の状態遷移表を示します。

	_15_IIC_SLAVE_WAIT_START_CONDITION	_10_IIC_SLAVE_RECEIVES_DATA	_14_IIC_SLAVE_RECEIVES_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)

*1. callback transmit 内で Send Event を発生させ、mode_flag を 17 に更新したのち、ステータスを 11 に更新する。（リスタートコンディションの検出に対応。）

*2. 14 に遷移する前に Stop コンディションを検出した場合は、受信動作未完了となるため、受信バッファの設定を初期化するために callback receivend 内で Receive Event[E01]を発生させる。
受信データ最大数まで受信した場合も、callback receiveend 内で Receive Event[E01]を発生させ、受信バッファの設定を初期化する。

*3. transmitend callback 内で、次の通信に備え Receive Event[E01] (PMBUS middleware が comamnd receive mode の場合)、または、Send Event[E00] (PMBus middleware が ARA send mode の場合)を発生させる。

5.2.3　PMBus Slave 関数一覧

PMBus Slave の関数は表 31、表 32 の Application 関数、表 33 表 33 の API 関数、表 34 の Middleware 関数、およびスマート・コンフィグレータまたは FSP で生成する表 35、表 35、表 36 表 36 のドライバ関数に分かれます。なおドライバ関数については PMBus Slave 処理に合わせて一部変更しています。変更内容については 5.2.4 PMBus Slave ドライバ部のカスタマイズをご参照ください。

表 31 PMBus Slave Application RX26T 関数一覧

File 名	Function 名	機能
pmbus_app¥ r_app_pmbus_main.c	main	既存のモータサンプルに PMBus Middleware の初期化処理を追加したアプリケーションのメイン処理です。
	r_app_main_start_pmbus_ctrl	PMBus Middleware をオープンします。
	pmbus_ctrl	メインルーチンで受信した PMBUS コマンドに対応した処理を実行します。
	r_pmbus_callback	PMBus Middleware に登録するコールバック関数です。受信したコマンドコードに対応した処理を実行します。
	r_app_pmbus_exe_write_command	ライトトランザクションコードのコマンドに対応した処理を実行します。コールバック関数から呼び出されます。
	r_app_pmbus_exe_read_command	リードトランザクションコードのコマンドに対応した処理を実行します。コールバック関数から呼び出されます。
pmbus_app¥r_app_board_ui.c	r_app_board_ui_mainloop	既存のモータサンプルに SW1 が ON の場合のモータ回転開始制御をコメント化した処理です。

表 32 PMBus Slave Application RA6T3 関数一覧

File 名	Function 名	機能
src¥hal_entry.c	hal_entry	既存のモータサンプルに PMBus Middleware の初期化処理を追加したアプリケーションのメイン処理です。
pmbus_app¥main¥ mtr_main.c	r_app_main_start_pmbus_ctrl	PMBUS Middleware をオープンします。
	mtr_main	既存のモータサンプルに PMBus Middleware の初期化処理を追加したアプリケーションのメイン処理です。
	board_ui	既存のモータサンプルのボード UI の状態の監視とモータの制御を行う処理です。
	pmbus_ctrl	メインルーチンで受信した PMBUS コマンドに対応した処理を実行します。
	r_pmbus_callback	PMBus Middleware に登録するコールバック関数です。受信したコマンドコードに対応した処理を実行します。
	r_app_pmbus_exe_write_command	ライトトランザクションコードのコマンドに対応した処理を実行します。コールバック関数から呼び出されます。
	r_app_pmbus_exe_read_command	リードトランザクションコードのコマンドに対応した処理を実行します。コールバック関数から呼び出されます。

表 33 PMBus Slave API 関数一覧

File 名	Function 名	機能
r_pmbus_app_slave.c	R_PMBUS_Slave_Open	PMBus Middleware をオープンし、マスターからのコマンド受信待ち状態にします。
	R_PMBUS_Slave_Close	PMBus Middleware をクローズします。
	R_PMBUS_Slave_EnablePEC	パケットに PEC を付与した送受信を有効にします。
	R_PMBUS_Slave_DisablePEC	パケットに PEC を付与した送受信を無効にします。
	R_PMBUS_Slave_SendARA	Alert Response プロトコル応答待ちに変化させます。
	R_PMBUS_Slave_Restart	Alert Response プロトコル応答待ち状態から、コマンド受信待ちに状態に戻します。

表 34 PMBus Slave Middleware 関数一覧

File 名	Function 名	機能
r_pmbus_app_slave.c	r_pmbus_app_InitCtrl	PMBus Middleware のパラメータを初期化します。
	r_pmbus_app_int_TransmitEnd	送信完了コールバック処理を実行します。
	r_pmbus_app_int_ReceiveEnd	受信完了コールバック処理を実行します。
	r_pmbus_app_int_Receive	受信コールバック処理を実行します。
	r_pmbus_app_int_Transmit	送信コールバック処理を実行します。
	r_pmbus_app_int_Notify	エラー検出コールバック処理を実行します。
	r_pmbus_app_CheckCommandSupport	受信したコマンドが PMBus の仕様でサポートしているかのチェックを実施します。
r_pmbus_nwk_slave.c	r_pmbus_nwk_StartSlave	PMBus のスレーブ受信動作を開始します。
	r_pmbus_nwk_StopSlave	PMBus のスレーブ動作を停止します。
	r_pmbus_nwk_ResetSlave	PMBus のスレーブ動作をリセットします。
	r_pmbus_nwk_StartSendARA	PMBus の ARA プロトコル応答動作を開始します。
	r_pmbus_nwk_RestartSlave	PMBus のスレーブ受信動作を再開します。
	r_pmbus_nwk_ProcessStart	PMBus のスタートコンディション検出処理を実行します。
	r_pmbus_nwk_ProcessRx	PMBus のスレーブ受信処理を実行します。
	r_pmbus_nwk_ProcessTx	PMBus のスレーブ送信処理を実行します。
	r_pmbus_nwk_ProcessStop	PMBus のストップコンディション検出処理を実行します。
	r_pmbus_nwk_ProcessErrorNotice	PMBus の想定外のタイミングでの割り込み検出時の処理を実行します。
	r_pmbus_nwk_ProcessStartRead	スレーブ受信モード検出時の事前処理を実行します。
	r_pmbus_nwk_ProcessStartWrite	スレーブ送信モード検出時の事前処理を実行します。
	r_pmbus_nwk_ProcessAfterStop	PMBus のストップコンディション検出後の後処理を実行します。
	r_pmbus_nwk_ProcessCallback	ユーザが登録したコールバック関数を実行します。
	r_pmbus_nwk_AddCrc8	1 つのデータの CRC 演算を実行します。
	r_pmbus_nwk_CaluculatePECAtSlave	複数のデータの CRC 演算を実行します。（スレーブアドレスは含みません。）
r_pmbus_wrapper.c	r_pmbus_wrapper_I2cStart	I2C ドライバの動作を開始します。本関数は RX26T と RA6T3 のドライバコードの差分を吸収するラッパー関数です。
	r_pmbus_wrapper_I2cStop	I2C ドライバの動作を停止します。本関数は RX26T と RA6T3 のドライバコードの差分を吸収するラッパー関数です。
	r_pmbus_wrapper_I2cReceive	I2C ドライバのスレーブ受信動作を開始します。本関数は RX26T と RA6T3 のドライバコードの差分を吸収するラッパー関数です。

	r_pmbus_wrapper_I2cSend	I2C ドライバのスレーブ送信動作を開始します。本関数は RX26T と RA6T3 のドライバコードの差分を吸収するラッパー関数です。
	r_pmbus_wrapper_TimerOpen	タイマドライバの動作を開始します。本関数は RX26T と RA6T3 のドライバコードの差分を吸収するラッパー関数です。
	r_pmbus_wrapper_TimerClose	タイマドライバの動作を停止します。本関数は RX26T と RA6T3 のドライバコードの差分を吸収するラッパー関数です。
	r_pmbus_wrapper_TimerStart	タイマドライバのカウント動作を開始します。本関数は RX26T と RA6T3 のドライバコードの差分を吸収するラッパー関数です。
	r_pmbus_wrapper_TimerStop	タイマドライバのカウント動作を停止します。本関数は RX26T と RA6T3 のドライバコードの差分を吸収するラッパー関数です。
	r_smbus_handle_callback	I3C 割り込みのコールバック関数です。本関数は RA6T3 でのみ使用します。
	g_gpt5_i2c_timeout_callback	I3C 通信のタイムアウトを監視するチャネルの GPT 割り込みのコールバック関数です。本関数は RA6T3 でのみ使用します。

表 35 スマート・コンフィグレータ関数（RX26T）

File 名	Function 名	機能	流用元からの変更有無
Config_RIIC0.c	R_Config_RIIC0_Create	スレーブアドレスを指定可能な、RIIC0 ドライバの初期設定を行う関数です。	有
	R_Config_RIIC0_Stop	スレーブアドレス検出無効化設定を含んで RIIC0 のスレーブ動作停止設定を行う関数です。	有
	R_Config_RIIC0_SLave_Receive	スレーブの受信動作を開始する関数です。	有
Config_TMR0.c	r_Config_TMR0_Start	タイマのカウントを開始する関数です。	有
Config_TMR0_User.c	r_Config_TMR0_cmia0_interrupt	コンペアマッチ割り込み A の割り込みルーチンです。	有
Config_RIIC0_user.c	r_Config_RIIC0_transmit_interrupt	送信バッファエンプティ割り込みの割り込みルーチンです。	有
	r_Config_RIIC0_receive_interrupt	受信バッファフル割り込みの割り込みルーチンです。	有
	r_Config_RIIC0_error_interrupt	エラー検出割り込みの割り込みルーチンです。	有
	r_Config_RIIC0_callback_transmitend	送信完了コールバック関数です。	有
	r_Config_RIIC0_callback_receiveend	受信完了コールバック関数です。	有
	r_Config_RIIC0_callback_error	RIIC0 の各種割り込みでエラー検出時のエラーコールバック関数です。	有
	r_User_RIIC0_callback_start	スタートコンディション検出コールバック関数です。	新規
	r_User_RIIC0_callback_transmit	送信コールバック関数です。	新規
	r_User_RIIC0_callback_receive	受信コールバック関数です。	新規

表 36 FSP 関数 (RA6T3)

File 名	Function 名	機能	流用元からの変更有無
¥pmbus_slave¥r_smbus_slave.c	R_SMBUS_SLAVE_Open	SMBUS ドライバのオープン処理を実行します。	新規
	r_smbus_slave_read_write	SMBUS ドライバの送信開始/受信開始処理を実行します。	新規
	r_smbus_slave_notify	SMBUS ドライバのエラー割り込みを検出した場合の後処理とコールバック関数を呼び出します。	新規
	r_smbus_slave_callback_request	SMBUS ドライバで割り込み発生時のコールバック関数を呼び出します。	新規
	r_smbus_open_hw_slave	I3C のレジスタを初期設定します。	新規
	r_smbus_slave_call_callback	SMBUS ドライバのコールバック関数を呼び出します。	新規
	r_smbus_rxi_check_illegal_start	SMBUS ドライバの受信バッファフル割り込み発生時の Start コンディション検出状況を確認します。	新規
	r_smbus_rxi_slave	SMBUS ドライバの受信バッファフル割り込みルーチンです。	新規
	r_smbus_txi_slave	SMBUS ドライバの送信バッファエンプティ割り込みルーチンです。	新規
	r_smbus_tei_slave	SMBUS ドライバの送信完了割り込みルーチンです。	新規
	r_smbus_err_slave	SMBUS ドライバのエラー検出割り込みルーチンです。	新規

5.2.4　PMBus Slave ドライバ部のカスタマイズ

PMBus Slave のドライバコード(RIIC0, TMR)を RX26T はスマート・コンフィグレータにより生成、RA6T3 は FSP により生成しています。RX26T はスマート・コンフィグレータのユーザコード保護機能により RIIC0、および TMR の一部の処理を変更、追加しています。RA6T3 は FSP で生成したコードを別関数として登録し、PMBus Slave 専用のドライバにしています。以下に RX26T のスマート・コンフィグレータの設定、および生成したドライバコードの変更箇所を、RA6T3 の FSP の設定、および生成したドライバの変更箇所を示します。

(1) スマート・コンフィグレータのカスタマイズ (RX26T)

- スマート・コンフィグレータの RIIC0 の設定　(RX26T)

- スマート・コンフィグレータで生成した RIIC0 ドライバコードの変更箇所一覧（RX26T）

関数名	r_Config_RIIC0_Slave_Create()
ファイル名	Config_RIIC0.c
変更内容	RIIC0 の初期化関数です。 SARL0 レジスタへユーザが指定したスレーブアドレスを設定できるようにグローバル変数 g_riic0_user_slave_addr を追加し、SARL0 へ上書き設定する処理を追加します。

変更前

```
/* 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 */
```

変更後

```
/* 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 */
```

関数名	R_Config_RIIC0_Stop()
ファイル名	Config_RIIC0.c
変更内容	RIIC0 を割り込み禁止に設定し、機能を停止させる関数です。 RIIC0 の機能を停止時にスレーブアドレスの検出を禁止するための ICSER レジスタのクリア処理を追加します。

変更前

```
    EN(RIIC0, EEI0) = 0U;
```

変更後

```
    EN(RIIC0, EEI0) = 0U;
    /* Start user code */
    /* Clear slave address detection */
    RIIC0.ICSER.BYTE = 0;

    /* End user code */
```

関数名	R_Config_RIIC0_Slave_Receive()
ファイル名	Config_RIIC0.c
変更内容	RIIC0（スレーブ）の受信動作を開始する関数です。 g_riic0_start_detect_at_receive の初期化処理を追加します。

変更前

```
    g_riic0_dummy_read_count = 0U;

    g_riic0_mode_flag = _16_IIC_SLAVE_RECEIVE;
```

変更後

```
    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;
```

関数名	r_User_RIIC0_callback_Start()
ファイル名	Config_RIIC0_user.c
変更内容	Slave で Start コンディション検出割り込み発生時のに実行用に新規に追加したコールバック関数です。スレーブ送受信の開始処理を実行するために r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_START) を実行します。

変更前

なし。

変更後

```
/* 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);
}
```

関数名	r_User_RIIC0_callback_Receive()
ファイル名	Config_RIIC0_user.c
変更内容	PMBus Middleware で追加した受信バッファフル割り込み時のコールバック関数です。 受信バッファフル割り込みが発生ごとに PMBus Middleware の状態を確認と更新をするために、r_pmbus_app_int_Receive()を実行します。 r_pmbus_app_int_Receive()の内部では、 "g_riic0_rx_count"(現在の受信データ数)が 0 の場合は、内部で r_pmbus_nwk_ProcessStart()を実行し、"p_pmbusSlaveCmdCallback"によるコマンド処理を実行後に、受信開始処理を実行します。"g_riic0_rx_count"が 0 以外の場合は、内部で r_pmbus_nwk_ProcessRx を実行し、受信処理を実行します。

変更前	変更後
なし。	

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

関数名	r_Config_RIIC0_transmit_interrupt ()
ファイル名	Config_RIIC0_user.c
変更内容	送信バッファエンプティ割り込みハンドラです。 RIIC0 ドライバが送信モード中でスタートコンディション（リスタートコンディション）を検出した場合に対応するため、ICSR2.START が 1 の場合、または g_riic0_start_detect_at_receive が 1 の場合に、以下の処理を実行します。 ・ICSR2.START をクリアします。 ・g_riic0_start_detect_at_receive をクリアします。 ・r_User_RIIC0_callback_transmit()を実行し、リスタートコンディション検出時の PMBus Middleware の処理を実行します。 ・ICDRT に*gp_riic0_tx_address を設定します。 ・g_riic9_tx_count が 0 より大きい場合、gp_riic0_tx_address のインクリメントと g_riic0_tx_count のデクリメントを実行します。

変更前	変更後

```
}
```

```
    /* 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 */
}
```

関数名	r_Config_RIIC0_receive_interrupt ()
ファイル名	Config_RIIC0_user.c
変更内容	受信バッファフル割り込みハンドラです。 g_riic0_state が_10_IIC_SLAVE_RECEIVE_DATA の場合の処理において、以下の処理を追加します。 ・ICSR2.START が 1 の場合は、ICDRR をダミーリード後”g_riic0_dummy_read_count”をインクリメントして、r_Config_RIIC9_callback_error(MD_ERROR4)を実行します。 ・g_riic0_dummy_read_count が 1 より大きい場合の処理に r_User_RIIC0_callback_receive()の実行による、PMBus Middleware の処理を追加します。 ・RIIC0.ICDRR をリード後に、r_User_RIIC0_callback_receive()の実行による、PMBus Middleware の処理を追加します。 g_riic0_state が_10_IIC_SLAVE_SEND_DATA の場合の処理において、r_User_RIIC0_callback_receive()の実行による、PMBus Middleware の処理を追加します。

変更前

```
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)
```

変更後

```
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)
```

関数名	r_Config_RIIC0_error_interrupt ()
ファイル名	Config_RIIC0_user.c
変更内容	エラー検出割り込み時のコールバック関数です。 Start コンディションを検出した時の PMbus 通信開始処理ができるように、 "g_riic0_state"が"_15_IIC_SLAVE_WAIT_START_CONDITION"の時に r_User_RIIC0_callback_start()を実行してください。（2 か所） さらに、受信モード時（_16_IIC_SLAVE_RECEIVE == g_riic0_mode_flag）の場合は、上記に加え g_riic0_start_detect_at_receive を初期化してください。 また、受信モード時(_16_IIC_SLAVE_RECEIVE == g_riic0_mode_flag)場合、 "g_riic0_state"が"_18_IIC_SLAVE_WAIT_RESTART_CONDITION"の場合に、 "RIIC0.ICIER.BIT.SPIE"による Start コンディション割り込み禁止設定を行わない分岐を追加してください。

変更前

```
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         }
```

変更後

```
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         }
```

関数名	r_Config_RIIC0_callback_transmitend ()
ファイル名	Config_RIIC0_user.c
変更内容	送信動作時の Stop コンディション検出割り込み時のコールバック関数です。 Stop コンディション検出割り込み後に、PMBus Middleware の送信完了処理を実行するために、r_pmbus_app_int_TransmitEnd()を実行してください。

変更前

```
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 }
```

変更後

```
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 }
```

関数名	r_Config_RIIC0_callback_receiveend ()
ファイル名	Config_RIIC0_user.c
変更内容	受信動作時の Stop コンディション検出割り込み時のコールバック関数です。 Stop コンディション検出割り込み後に、PMBus Middleware の受信完了処理と、"r_pmbusSlaveCmdCallback"を実行するために、r_pmbus_app_int_ReceiveEnd()を実行してください。

変更前

```
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 }
```

変更後

```
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 }
```

関数名	r_Config_RIIC0_callback_error ()
ファイル名	Config_RIIC0_user.c
変更内容	エラー検出割り込み時に、割り込み要因がエラーの場合に実行するコールバック関数です。 "MD_ERROR4"（ドライバシーケンス外での Start コンディションを検出）の場合： ICSR2 の START ビットが 1 である場合は、ICSR2 の START ビットをクリア後に、PMBus Middleware の状態を更新するために r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_START_UNEXPECTED) を実行してください。

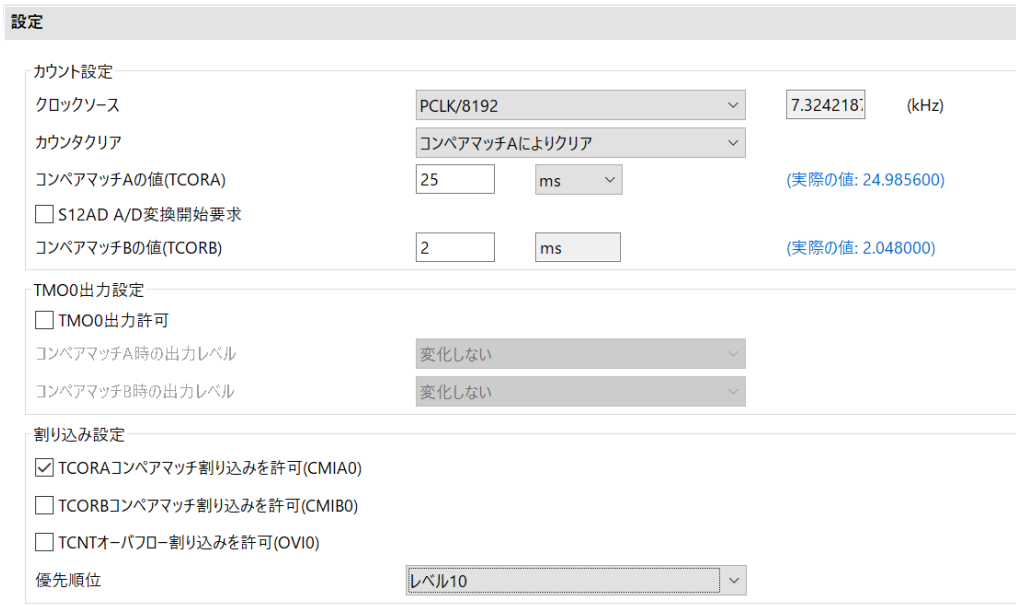
変更前

```
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         }
```

変更後

```
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         }
```

- スマート・コンフィグレータの TMR の設定（RX26T）



- スマート・コンフィグレータで生成した TMR ドライバコードの変更箇所一覧（RX26T）

関数名	r_Config_TMR0_cmia0_interrupt （）
ファイル名	Config_TMR0_User.c
変更内容	エラー検出割り込み時に、割り込み要因がエラーの場合に実行するコールバック関数です。 PMBus Middleware の状態を更新するために r_pmbus_app_int_Notify(E_PMBUS_INT_EVENT_S_TIMEOUT) を実行してください。

変更前

```
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 */
}
```

変更後

```
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 */
}
```

関数名	r_Config_TMR0_Start()
ファイル名	Config_TMR0.c
変更内容	カウントを開始する API です。 関数の先頭でカウンタレジスタ（TCNT）を初期化する処理を追加してください。

変更前

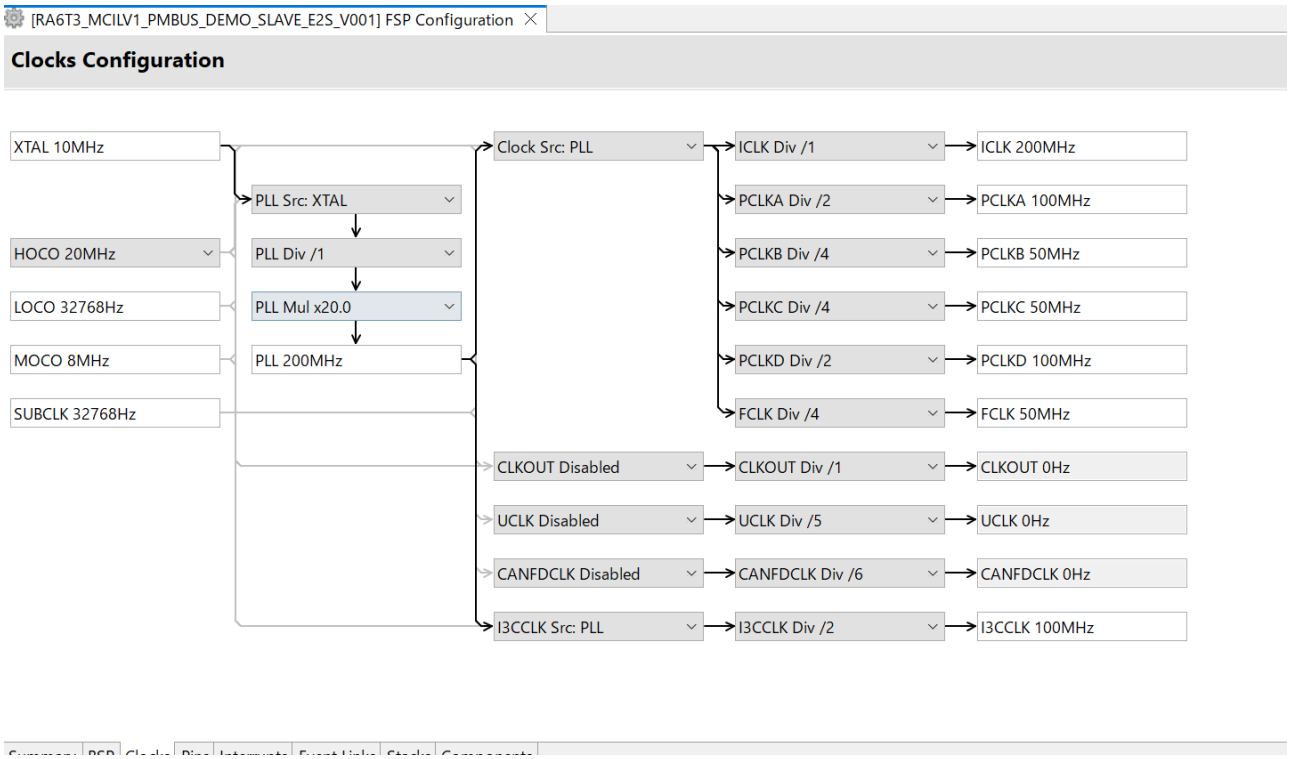
```
{

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

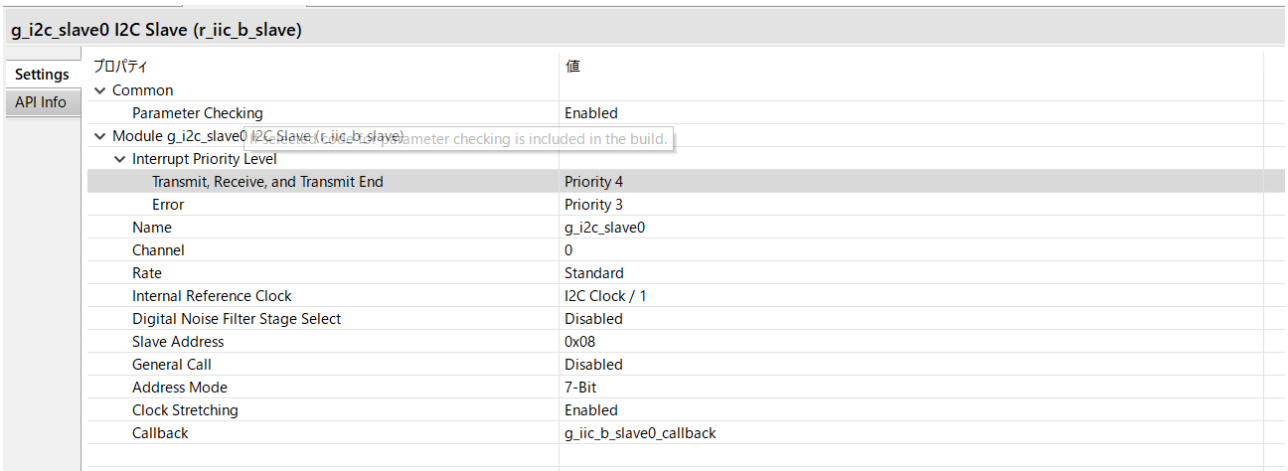
変更後

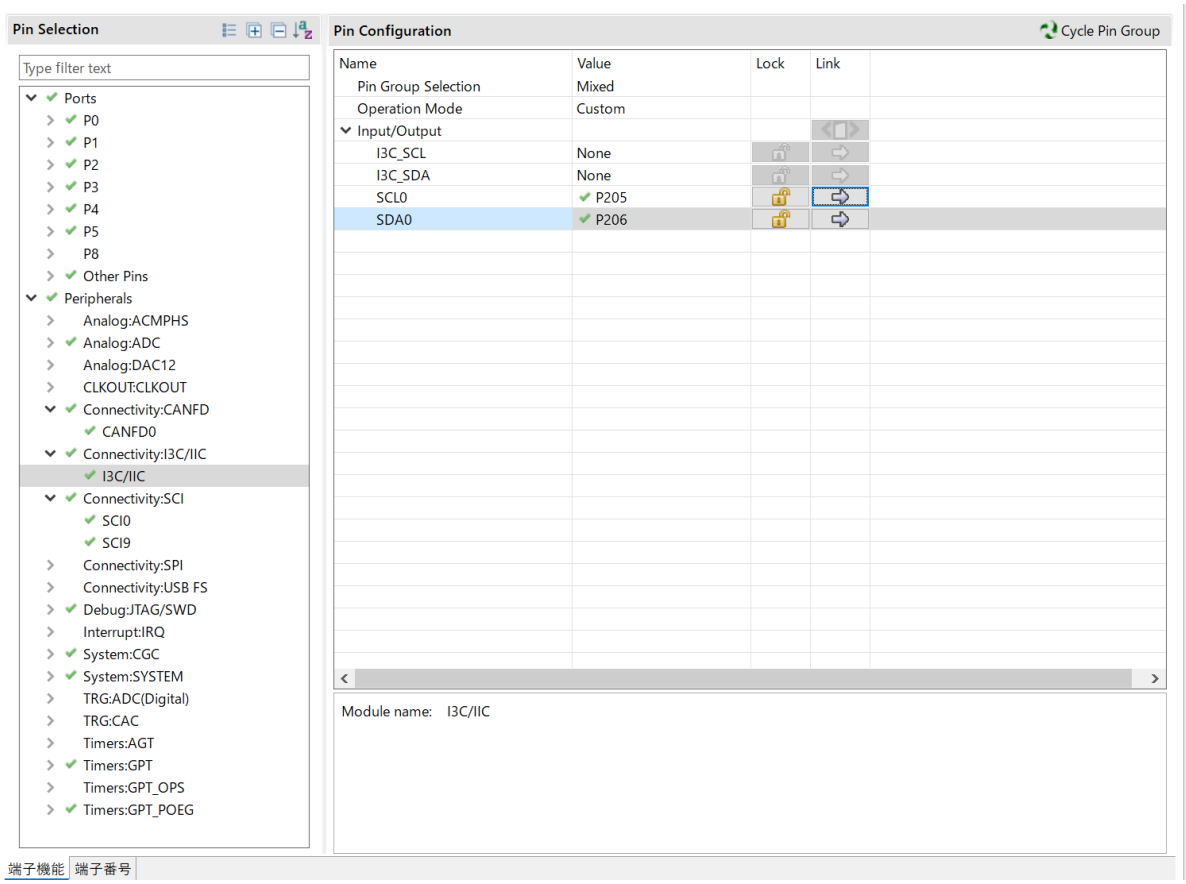
```
    /* Start user code */
    /* Clear Timer count*/
    TMR0.TCNT = 0U;

    /* End user code */
```

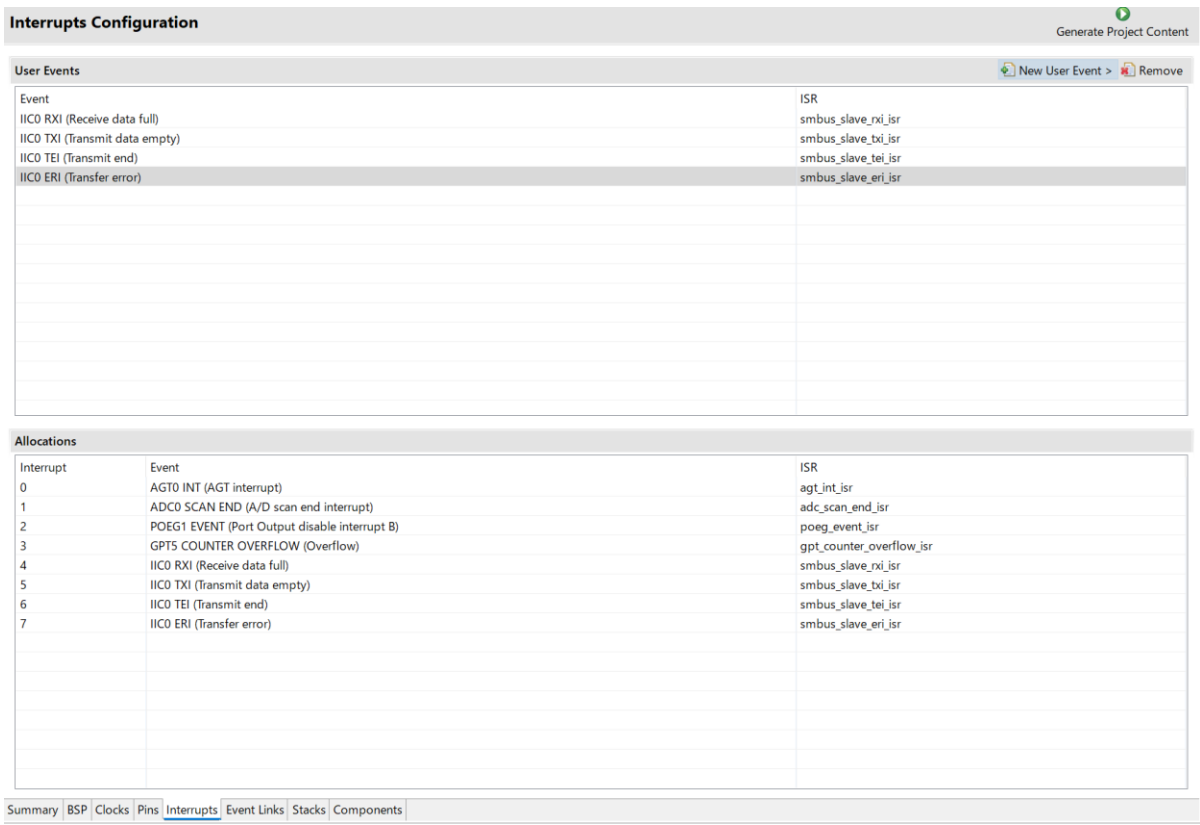
(2) FSP のカスタマイズ (RA6T3)

- FSP の I3C の設定 (RA6T3)






Pin Selection
Type filter text
Ports
P0
P1
P2
P3
P4
P5
P8
Other Pins
Peripherals
Analog:ACMPHS
Analog:ADC
Analog:DAC12
CLKOUT:CLKOUT
Connectivity:CANFD
CANFD0
Connectivity:I3C/IIC
I3C/IIC
Connectivity:SCI
SCI0
SCI9
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
P206
Module name:
I3C/IIC
端子機能
端子番号


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

- FSP で生成した I3C ドライバコードの変更箇所一覧（RA6T3）

【注】FSP で生成されるコードのシンボル名は PMBus Middleware においては、以下の用に置き換えています。

シンボル名の置き換えについては本一覧表で詳細を説明することは省略しています。

・I2C_SLAVE　-> SMBUS_SLAVE
・i2c_slave　-> smbus_slave
・IIC_B_SLAVE -> SMBUS_SLAVE
・iic_b_slave -> smbus_slave

データタイプ	i2c_slave_event_t
ファイル名	r_i2c_slave_api.h
変更内容	・EVENT_START_REQUEST を追加。 ・EVENT_STOP__REQUEST を追加。 ・EVENT_ARB_LOST を追加。 ・EVENT_TIMEOUT を追加。 ・EVENT_NACK を追加。 ・EVENT_START_ERR を追加。

変更前

```
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;
```

変更後

```
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;
```

データタイプ	iic_b_slave_instance_ctrl_t
ファイル名	r_iic_b_slave.h
変更内容	volatile bool start_detect; を追加。

変更前

```
    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;
```

変更後

```
    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;
```

マクロ名	IIC_B_SLAVE_PRV_BIE_INIT_MASK
ファイル名	r_iic_b_slave.c
変更内容	R_I3C0__BIE_ALIE_Mask、R_I3C0__BIE__TODIE_Msk を削除。

変更前

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

変更後

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

マクロ名	IIC_B _SLAVE_PRV_BIE_INIT_MASK
ファイル名	r_iic_b_slave.c
変更内容	R_I3C0__BSTE_ALE_Mask、R_I3C0__BSTE__TODE_Msk を削除。

変更前

```
#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)
```

変更後

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

テーブル名	g_iic_b_slave0_extend
ファイル名	r_iic_b_slave.c
変更内容	hal.data.c に生成されるグローバルテーブルを移動。

変更前

なし。

変更後

```
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, };
```

テーブル名	g_iic_b_slave0_cfg
ファイル名	r_iic_b_slave.c
変更内容	hal.data.c に生成されるグローバルテーブルを移動。

変更前

なし。

変更後

```
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, };
```

テーブル名	g_iic_slave0
ファイル名	r_iic_b_slave.c
変更内容	hal.data.c に生成されるグローバルテーブルを移動。

変更前

なし。

変更後

```
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 };
```

関数名	R_IICB_B_SLAVE_Open
ファイル名	r_iic_b_slave.c
変更内容	p_ctrl->start_detect の初期化を追加。

変更前

```
    p_ctrl->notify_request    = false;
    p_ctrl->transaction_count = 0U;

    return FSP_SUCCESS;
}
```

変更後

```
    p_ctrl->transaction_count = 0U;
    p_ctrl->start_detect      = false;

    return FSP_SUCCESS;
}
```

関数名	iic_b_slave_read_write
ファイル名	r_iic_b_slave.c
変更内容	・引数 bytes は内部で代入する際に uint16_t にキャストする。 ・p_ctrl->direction のチェック処理を削除。 ・p_ctrl->direction が" MASTER_WRITE_SLAVE_READ"の場合に、p_ctrl->start_detect を初期化する処理を追加。

変更前

```
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 }
```

変更後

```
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;
```

関数名	iic_b_slave_notify
ファイル名	r_iic_b_slave.c
変更内容	・BSTE の設定に STCNDDE ビットと SPCNDDE ビットを 1 にする設定を追加。 ・ローカル変数 transaction_count の型を uint16_t に変更。

変更前

```
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 */
```

変更後

```
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 */
```

関数名	iic_b_slave_callback_request
ファイル名	r_iic_b_slave.c
変更内容	iic_b_slave_call_callback()実行前後の BSTE.TODE ビット、および、BIE.TODIE ビットの設定を削除。

変更前

```
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_I3C0_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;
```

変更後

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

関数名	iic_b_open_hw_slave
ファイル名	r_iic_b_slave.c
変更内容	・BFCTL.SMBS=1(SMBus を選択)の設定を追加。 ・BFCTL.SALE=0(スレーブアービトレーションロスト検出は無効)の設定に変更。 ・BFCTL.NALE=0(NACK 送信アービトレーションロスト検出は無効)の設定に変更。 ・OUTCTL.SDOD[2:0]=111b(13 または 14 I3Cφ サイクル (OUTCTL.SDODCS = 1 のとき(I3Cφ/2)))、OUTCTL.SDODCS=1(SDA 出力遅延カウンタのクロックソースに内部基準クロックの 2 分周 (I3Cφ/2)を選択) ・TMOCTL.TOLCTL=0(SCL の L 期間タイムアウト検出無効)、TMOCTL.TOHCTL=0(SCL の H 期間タイムアウト検出無効) ・BIE.NACKDIE=1(NACK 検出割り込み有効)に設定。 ・BIE.STCNDIE=1(Start コンディション検出割り込み有効)に設定。 ・BIE.NACKDIE=1(Stop コンディション検出割り込み有効)に設定。 ・BFRECDT.FRECYC[8:0](バスフリー期間)の設定の追加。 (1/I3Cφ(100MHz)*4.7μs=470(0x1D6hex)

変更前	変更後

```
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);
```

なし。

```
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      */
613     p_ctrl->p_reg->OUTCTL |= (uint32_t)(R_I3C0_OUTCTL_SDOD_Msk |
    R_I3C0_OUTCTL_SDODCS_Msk);
```

```
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
```

なし。

```
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
```

関数名	iic_b_slave_initiate_transaction
ファイル名	r_iic_b_slave.c
変更内容	呼出し元の関数を iic_b_slave_callback_request に置き換えるため、本関数は削除。

変更前	変更後
	削除。

```
 *
 * @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->IER = IIC_B_SLAVE_PRV_IER_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
```

関数名	i iic_b_rxi_slave
ファイル名	r_iic_b_slave.c
変更内容	・コールバックの呼び出し関数を、iic_b_slave_callback_request()を使用するように変更。 ・Start コンディションの検出状況を確認する r_smbus_rxi_check_illegal_start 関数を追加し、以下の処理を実施します。 -BST の STCNDDF ビットが"1"の場合、NTDTBP0 レジスタをダミーリードしたのち、BST の STCNDDF ビットをクリアし、EVENT_START_ERR を指定して iic_b_slave_callback_request()を実行する。 - p_ctrl->direction が"MASTER_READ_SLAVE_WRITE"の場合、NTDTBP0 レジスタをダミーリードしたのち、EVENT_RX_REQUEST を指定して iic_b_slave_callback_request()を実行する。 - 上記以外の場合は受信バッファエンプティ割り込み処理の次の処理を実行する。 ・想定外での受信割り込みの発生となった場合の処理を、ACKCTL.ACKT ビットによる NACK 応答ではなく、NTDTBP0 レジスタのダミーリードと iic_b_slave_callback_request()のコールバック関数実行による、NACK 検出通知に変更。 ・ダミーリード p_ctrl->do_dummy_read が 0 の場合の BIE レジスタの設定時に、STCNDDIE ビットは設定せず、NACKDIE ビットと SPCNDDIE ビットを設定するように変更。

変更前

なし。

```
734 static void iic_b_rxi_slave (iic_b_slave_instance_ctrl_t * p_ctrl)
735 {
736     /* Perform dummy read after an address match detection. */
737     if (!p_ctrl->do_dummy_read)
```

変更後

```
692  * Check start detection when receive data full interrupt
     operating as a slave.
693  *
694  * @param[in]       p_ctrl     The target IIC block's control
     block.
695  
     ***********************************************************************
     ***********************************************************/
696 static bool
     r_smbus_rxi_check_illegal_start(smbus_slave_instance_ctrl_t *
     p_ctrl)
697 {
698     bool check_result;
699 
700     /* If a start condition is detected during reception mode,
     abnormality processing is performed. */
701     if (1 == p_ctrl->p_reg->BST_b.STCNDDF)
702     {
703         /* dummy read */
704         volatile uint8_t dummy_read = (uint8_t)
     (p_ctrl->p_reg->NTDTBP0 & SMBUS_SLAVE_PRV_NTDTBP0_SIZE);
705         FSP_PARAMETER_NOT_USED(dummy_read);
706 
707         /* clear start condition detection flag */
708         p_ctrl->p_reg->BST &= ((uint32_t)
     ~(R_I3C0_BST_STCNDDF_Msk));
709 
710         r_smbus_slave_callback_request(p_ctrl,
     SMBUS_SLAVE_EVENT_START_ERR);
711 
712         check_result = true;
713     }
714     /* Perform dummy read after an address match detection. */
715     else if (SMBUS_SLAVE_TRANSFER_DIR_MASTER_READ_SLAVE_WRITE ==
     p_ctrl->direction)
716     {
717         /* dummy read */
718         volatile uint8_t dummy_read = (uint8_t)
     (p_ctrl->p_reg->NTDTBP0 & SMBUS_SLAVE_PRV_NTDTBP0_SIZE);
719         FSP_PARAMETER_NOT_USED(dummy_read);
720 
721         r_smbus_slave_callback_request(p_ctrl,
     SMBUS_SLAVE_EVENT_RX_REQUEST);
722 
723         check_result = true;
724     }
725     else
726     {
727         check_result = false;
728     }
729 
730     return check_result;
731 }
732 
733 /***********************************************************************
     ***********************************************************//**
```

```
738 static void r_smbus_rxi_slave (smbus_slave_instance_ctrl_t *
    p_ctrl)
739 {
740 
741     if (r_smbus_rxi_check_illegal_start(p_ctrl))
742     {
743         /* If illegal start condition detected, operate in
    r_smbus_rxi_check_illegal_start */
744         ;
745     }
746     /* Perform dummy read after an address match detection. */
747     else if (!p_ctrl->do_dummy_read)
```

<table>
<tr>
<td>

```
            * 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;

```

</td>
<td>

```
            * 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;

```

</td>
</tr>
<tr>
<td>

```
           p_ctrl->p_reg->NTIE = IIC_B_SLAVE_PRV_NTIE_INIT_MASK;




       }
       else
```

</td>
<td>

```
           p_ctrl->p_reg->NTIE = SMBUS_SLAVE_PRV_NTIE_INIT_MASK;

           r_smbus_slave_callback_request(p_ctrl,
SMBUS_SLAVE_EVENT_RX_REQUEST);

       }
       else
```

</td>
</tr>
<tr>
<td>

```
           /* Check if the read request event has been notified through callback, if not provide the
callback */
           if (p_ctrl->notify_request)
           {
               /* Check if this is a General Call by Master */
               i2c_slave_event_t receive_callback_event =
                   (R_I3C0_SVST_GCAF_Msk ==
                    (p_ctrl->p_reg->SVST &
                     R_I3C0_SVST_GCAF_Msk)) ? I2C_SLAVE_EVENT_GENERAL_CALL : I2C_SLAVE_EVENT_RX_REQUEST;
               iic_b_slave_initiate_transaction(p_ctrl, receive_callback_event);

```

</td>
<td>

```
           {
               /* Read data */
               p_ctrl->p_buff[p_ctrl->loaded++] = (uint8_t) (p_ctrl->p_reg->NTDTBP0 &
SMBUS_SLAVE_PRV_NTDTBP0_SIZE);

               /* Keep track of the the actual number of transactions */
               p_ctrl->transaction_count++;

               /* Check if this is a General Call by Master */
               receive_callback_event =
                   (R_I3C0_SVST_GCAF_Msk ==
                    (p_ctrl->p_reg->SVST &
                     R_I3C0_SVST_GCAF_Msk)) ? SMBUS_SLAVE_EVENT_GENERAL_CALL :
SMBUS_SLAVE_EVENT_RX_REQUEST;
           }

           r_smbus_slave_callback_request(p_ctrl, receive_callback_event);

```

</td>
</tr>
<tr>
<td>

```
#if IIC_B_SLAVE_CFG_PARAM_CHECKING_ENABLE

       /* Proceed reading data */
       if (IIC_B_SLAVE_TRANSFER_DIR_MASTER_WRITE_SLAVE_READ != p_ctrl->direction)
       {
           /* If the user application incorrectly handles Master Write, send a NACK to exit the
transaction. */
           /* Do not dummy read here to allow slave to timeout */




           /* Writes to be done together with write protect bit.
            * See Note 1 in Section 27.2.15 "ACKCTL : Acknowledge Control Register"
            * of the RA6T2 manual R01UH0951EJ0050
            */
           p_ctrl->p_reg->ACKCTL = R_I3C0_ACKCTL_ACKTWP_Msk | R_I3C0_ACKCTL_ACKT_Msk;
       }
       else
#endif
```

</td>
<td>

削除。

</td>
</tr>
<tr>
<td>

```
       {
           if (OU == p_ctrl->total)  /* Send NACK */
           {
               /* Slave is sending a NACK. */
               /* Do dummy read to release SCL */
               volatile uint32_t dummy_read = p_ctrl->p_reg->NTDTBP0;
               FSP_PARAMETER_NOT_USED(dummy_read);

               /* Set the response as NACK, since slave is not setup for reading any data from
master at this time.
                * This is an intentional way to let master know that the slave receiver cannot
                * accept any data and hence should eventually result in I2C_SLAVE_EVENT_RX_COMPLETE.
                *
                * Writes to be done together with write protect bit.
                * See Note 1 in Section 27.2.15 "ACKCTL : Acknowledge Control Register"
                * of the RA6T2 manual R01UH0951EJ0050
                */
               p_ctrl->p_reg->ACKCTL = R_I3C0_ACKCTL_ACKTWP_Msk | R_I3C0_ACKCTL_ACKT_Msk;
           }
           /* If master is requesting still more data than configured to be read, notify
            * with a read more event in callback */
           else if (p_ctrl->total == p_ctrl->loaded)
           {
               iic_b_slave_callback_request(p_ctrl, I2C_SLAVE_EVENT_RX_MORE_REQUEST);
#if IIC_B_SLAVE_CFG_PARAM_CHECKING_ENABLE
               if (IIC_B_SLAVE_TRANSFER_DIR_MASTER_WRITE_SLAVE_READ != p_ctrl->direction)
               {
                   /* If the user application incorrectly handles Master Write, send a NACK to exit
the transaction. */
                   /* Do not dummy read here to allow slave to timeout */
                   p_ctrl->p_reg->ACKCTL = R_I3C0_ACKCTL_ACKTWP_Msk | R_I3C0_ACKCTL_ACKT_Msk;
               }
               else
#endif
               {
                   if (0 == p_ctrl->total) /* Send NACK */
                   {
                       /* Do dummy read to release SCL */
                       volatile uint32_t dummy_read = p_ctrl->p_reg->NTDTBP0;
                       FSP_PARAMETER_NOT_USED(dummy_read);

                       /* Set the response as NACK, since slave is not setup for reading more data
from master at this time.
                        * This is an intentional way to let master know that the slave receiver
cannot
                        * accept any more data and hence should eventually result in
I2C_SLAVE_EVENT_RX_COMPLETE.
                        */
                       p_ctrl->p_reg->ACKCTL = R_I3C0_ACKCTL_ACKTWP_Msk | R_I3C0_ACKCTL_ACKT_Msk;
                   }
                   else
                   {
                       /* Read data */
                       p_ctrl->p_buff[p_ctrl->loaded++] =
                           (uint8_t) (p_ctrl->p_reg->NTDTBP0 & IIC_B_SLAVE_PRV_NTDTBP0_SIZE);

                       /* Keep track of the the actual number of transactions */
                       p_ctrl->transaction_count++;
                   }
               }
           }

```

</td>
<td>

```
       else
       {
           smbus_slave_event_t receive_callback_event;


           if (OU == p_ctrl->total)
           {
               /* Slave is sending a NACK. */
               /* Do dummy read to release SCL */
               volatile uint32_t dummy_read = p_ctrl->p_reg->NTDTBP0;
               FSP_PARAMETER_NOT_USED(dummy_read);
               receive_callback_event = SMBUS_SLAVE_EVENT_NACK;

           }
           /* If master is requesting still more data than configured to be read, notify
            * with a read more event in callback */
           else if (p_ctrl->total == p_ctrl->loaded)
           {
               r_smbus_slave_callback_request(p_ctrl, SMBUS_SLAVE_EVENT_RX_MORE_REQUEST);

               /* Do dummy read to release SCL */
               volatile uint32_t dummy_read = p_ctrl->p_reg->NTDTBP0;
               FSP_PARAMETER_NOT_USED(dummy_read);
               receive_callback_event = SMBUS_SLAVE_EVENT_NACK;


           }
           else
```

</td>
</tr>
</table>

関数名	iic_b_txi_slave
ファイル名	r_iic_b_slave.c
変更内容	・コールバックの呼び出し関数を、iic_b_slave_callback_request()を使用するように変更。 ・iic_b_slave_callback_request()を呼び出す条件を、BST.STCNDDF=1(Start コンディション検出)の場合、または、p_ctrl->start_detect=true（受信モード中のスタートコンディション検出）の場合に変更し、この場合に p_ctrl->start_detect を false に初期化する処理を追加。

変更前

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

変更後

```
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 */
```

関数名	iic_b_tei_slave
ファイル名	r_iic_b_slave.c
変更内容	ACKCTL.ACKR=1(ACK 検出)時の NTDTBP0 へ書き込むデータをダミーデータとするように変更。

変更前

```
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     }
```

変更後

```
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     }
```

関数名	iic_b_err_slave
ファイル名	r_iic_b_slave.c
変更内容	・エラーイベントの分岐を、BST.TODF=1(タイムアウト検出)、BST.ALFDF=1(アービトレーションロスト検出)、BST.STCNDDF=1(Start コンディション検出)、BST.SPCDDF=1(Stop コンディション検出)、BST.NACKDF=1(NACK 検出)のそれぞれで行うように変更。 ・BST.TODF=1 の場合は iic_b_slave_notify()に設定するイベントは"EVENT_TIMEOUT"に変更。 ・BST.ALFDF=1 の場合は、iic_b_slave_notify()に設定するイベントは"EVENT_ARB_LOST"に変更。 ・BST.STCNDDF=1 の場合は、BIE.STCNDDIE=0(Start コンディション検出割り込み禁止)、BIE.SPCNDDIE=1(Stop コンディション検出割り込み許可)に設定後、iic_b_slave_callback_request で"EVENT_START_REQUEST"を通知するように変更。 ・BST.SPCNDDF=1 の場合は、BIE.SPCNDDIE=0(Stop コンディション検出割り込み禁止)、BIE.STCNDDIE=1(Start コンディション検出割り込み許可)に設定、p_ctrl->start_detect に true を設定後、iic_b_slave_callback_request で"EVENT_RX_COMPLETE"、または、"EVENT_TX_COMPLETE"を通知するように変更。 ・BST.NACKDF=1 の場合は、BIE.NACKDIE=0(NACK 検出割り込み禁止)の設定を削除し、iic_b_slave_notify()にイベント"EVENT_NACK"を設定するように変更。

変更前

```
    if ((error_events & R_I3C0_BST_TODF_Msk) || (error_events & R_I3C0_BST_ALF_Msk))
    {
        /* Clear the stop flag. This indicates an error. */
        p_ctrl->transaction_completed = false;

        iic_b_slave_notify(p_ctrl, I2C_SLAVE_EVENT_ABORTED);
    }
```

なし。

```
        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);
}
```

変更後

```
    /* Timeout or Arbitration loss detected */
    if ((error_events & R_I3C0_BST_TODF_Msk) || (error_events & R_I3C0_BST_ALF_Msk))
    {
        /* Clear the stop flag. This indicates an error. */
        p_ctrl->transaction_completed = false;

        /* Timeout detected */
        if (error_events & R_I3C0_BST_TODF_Msk)
        {
            r_smbus_slave_notify(p_ctrl, SMBUS_SLAVE_EVENT_TIMEOUT);
        }
        /* Arbitration loss detected */
        else
        {
            r_smbus_slave_notify(p_ctrl, SMBUS_SLAVE_EVENT_ARB_LOST);
        }
    }
```

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

        /* Disable start condition detect interrupt and enable stop condition detect interrupt */
        p_ctrl->p_reg->BIE_b.STCNDDIE = 0U;
        p_ctrl->p_reg->BIE_b.SPCNDDIE = 1U;

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

        /* A flag is set in case a start condition for transmission processing is */
        /* detected during reception waiting.                                     */
        p_ctrl->start_detect = true;

        /* Notify the user */
        r_smbus_slave_callback_request(p_ctrl, SMBUS_SLAVE_EVENT_START_REQUEST);
    }
    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.SPCNDDIE = 0U;
        p_ctrl->p_reg->BIE_b.STCNDDIE = 1U;
    }
```

```
1011        volatile uint32_t dummy_read = p_ctrl->p_reg->NTDTBP0;
1012        FSP_PARAMETER_NOT_USED(dummy_read);
1013
1014        /* Disable NACK interrupt, this is required since we will clear NACK flag only on detection
            of STOP bit or
1015         * when a timeout occurs. Not clearing the flag will cause error interrupt to get triggered
            again.
1016         */
1017        p_ctrl->p_reg->BIE &= (uint32_t) ~R_I3C0_BIE_NACKDIE_Msk;
1018    }
1019    else
```

```
1014        volatile uint32_t dummy_read = p_ctrl->p_reg->NTDTBP0;
1015        FSP_PARAMETER_NOT_USED(dummy_read);
1016
1017        r_smbus_slave_notify(p_ctrl, SMBUS_SLAVE_EVENT_NACK);
1018    }
1019    else
```

- FSP の GPT の設定 (RA6T3)

g_timer5 Timer, General PWM (r_gpt)

Settings / API Info

プロパティ	値
Common	
Parameter Checking	Enabled
Pin Output Support	Enabled with Extra Features
Write Protect Enable	Disabled
Clock Source	PCLKD
Module g_timer5 Timer, General PWM (r_gpt)	
General	
Output	
Input	
Interrupts	
Callback	g_gpt5_i2c_timeout_callback
Overflow/Crest Interrupt Priority	Priority 3
Capture A Interrupt Priority	Disabled
Capture B Interrupt Priority	Disabled
Underflow/Trough Interrupt Priority	Disabled
Extra Features	
Pins	
GTIOC5A	<unavailable>
GTIOC5B	<unavailable>

Pin Configuration

Generate Project Content

Select Pin Configuration　Export to CSV file　Configure Pin Driver Warnings

RA6T3 MCK　Manage configurations...

Generate data: g_bsp_pin_cfg

Pin Selection

Type filter text

- Peripherals
  - Analog:ACMPHS
  - Analog:ADC
  - Analog:DAC12
  - CLKOUT:CLKOUT
  - Connectivity:CANFD
  - Connectivity: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
    - GPT
    - GPT0
    - GPT1
    - GPT2
    - GPT3
    - GPT4
    - GPT5
  - Timers:GPT OPS

Pin Configuration　Cycle Pin Group

Name	Value	Lock	Link
Pin Group Selection	Mixed		
Operation Mode	Disabled		
Input/Output			
GTIOC5A	None		
GTIOC5B	None		

Module name: GPT5

端子機能　端子番号

Summary　BSP　Clocks　Pins　Interrupts　Event Links　Stacks　Components

- FSP で生成した GPT ドライバコードの変更箇所一覧 (RA6T3)

関数名	p_callback
ファイル名	r_pmbus_wrapper_slave.c
変更内容	オーバーフロー割り込みで実行されるコールバック関数で、R_GPT_Open 時に引数で登録します。本関数の実態は PMBus Middleware においては、r_pmbus_wrapper_slave.c に配置します。 PMBus Middleware の状態を更新するために r_pmbus_nwk_slave_process_Timeout() を実行してください。

変更前	変更後
なし。	

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

- RX26T と RA6T3 のドライバ API の比較

周辺機能	RX26T の API 名	RA6T3 の API 名	機能概要
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)	初期化/通信開始
	void R_Config_RIIC0_Create_UserInit(void)		
	void R_Config_RIIC0_Start(void)		
	MD_STATUSR_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)	受信開始
	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)	送信開始
	void R_Config_RIIC0_Stop(void)	fsp_err_t R_IIC_B_SLAVE_Close (i2c_slave_ctrl_t * const p_api_ctrl)	通信停止
	void R_Config_RIIC0_IIC_StartCondition(void)	-	Start コンディション発行
	void R_Config_RIIC0_IIC_StopCondition(void)	-	Stop コンディション発行
	-	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)	コールバック関数の変更。
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)	初期化
	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)	タイマ開始
		fsp_err_t R_GPT_Reset (timer_ctrl_t * const p_ctrl)	カウンタリセット
	void R_Config_TMR0_Stop(void)	fsp_err_t R_GPT_Stop (timer_ctrl_t * const p_ctrl)	タイマ停止
		fsp_err_t R_GPT_Close (timer_ctrl_t * const p_ctrl)	タイマ終了

5.2.5 PMBus Slave データタイプ、構造体一覧

本制御プログラムで使用するデータタイプ、および、構造体を次に示します。

- PMBus Slave Application 関数で使用するデータタイプ、構造体

e_app_req_event_t

列挙型名	e_app_req_event_t	
説明	PMBUS コールバックの発生要因、または、その他のイベントの発生を main 処理へ通知するために使用する列挙型です。	
宣言しているヘッダファイル	r_app_main.h	
備考	-	
要素名	**説明**	**値**
E_PMBUS_REQ_EVENT_NONE	イベント発生なし。	0
E_PMBUS_REQ_EVENT_ERROR	I2C のエラー割り込みが発生した場合にコールバックからメイン処理にエラーの発生を通知する場合に使用します。ユーザシステムの必要に応じて、メイン処理で R_PMBUS_Slave_Restart、または、R_PMBUS_Slave_Close/R_PMBUS_Slave_Open をコールするなどの、エラー後処理を実行してください。	1
E_PMBUS_REQ_EVENT_ARA_START	本イベントは ALERT RESPONSE 応答動作を確認するためのオプションで用意しているイベントです。ALERT シグナルの検出等により、ALERT RESPONSE 処理の動作を確認したい場合に、ユーザが PMBus Slave Application を変更して利用してください。本イベント検出時はメイン処理で R_PMBUS_Slave_SendARA を実行してください。	2
E_PMBUS_REQ_EVENT_ARA_STOP	本イベントは ALERT RESPONSE 応答動作から通常動作に戻す動作を確認するためのオプションで用意しているイベントです。ALERT RESPONSE 処理の動作をから通常動作に戻したい場合に、ユーザが PMBus Slave Application を変更して利用してください。本イベント検出時はメイン処理で R_PMBUS_Slave_Restart を実行してください。	3
E_PMBUS_REQ_EVENT_PEC_ENA	本イベントは PEC 付きの通信を確認するためのオプションで用意しているイベントです。Slave の動作の途中に Slave のリセットなどをせずに PEC 付きの通信を行いたい場合に、ユーザが PMBus Slave Application を変更して利用してください。本イベント検出時はメイン処理で R_PMBUS_Slave_PEC_Enable を実行してください。	4
E_PMBUS_REQ_EVENT_PEC_DIS	本イベントは PEC 付きの通信の動作状態から通常動作に戻すためのオプションで用意しているイベントです。Slave の動作の途中に Slave のリセットなどをせずに PEC 付きの通信から通常の通信に戻す場合に、ユーザが PMBus Slave Application を変更して利用してください。本イベント検出時はメイン処理で R_PMBUS_Slave_PEC_Disavle を実行してください。	5

- PMBus Slave API で使用するデータタイプ、構造体

e_pmbus_packet_result_s_t

列挙型名	e_pmbus_packet_result_s_t	
説明	PMBus 通信(slave)の実行結果を示すために使用する列挙型です。 各 PMBus Slave API の引数の型として使用します。PMBus API の Return 値が PMBUS_RET_ERROR の場合のエラー原因の詳細を示します。	
宣言しているヘッダファイル	r_pmbus_app_slave.h	
備考	-	
要素名	**説明**	**値**
E_PMBUS_PACKET_S_OK	正常動作。	0
E_PMBUS_PACKET_S_DATA_SIZE_ERROR	パケットサイズ異常を検出。	1
E_PMBUS_PACKET_S PEC_ERROR	PEC 演算結果の異常を検出。	2
E_PMBUS_PACKET_S_TIMEOUT	タイムアウトエラーを検出。（$T_{TIMEOUT}$ 異常の検出）	3
E_PMBUS_PACKET_S_CMD_WAIT	コマンドコードおよび受信データを受信中。	4
E_PMBUS_PACKET_S_CMD_COMP	コマンドコードおよび受信データを受信完了。	5
E_PMBUS_PACKET_S_CMD_QUICK	Quick Command 処置中。	6
E_PMBUS_PACKET_S_INTERNAL_ERROR	内部エラーを検出。	7
E_PMBUS_PACKET_S_NOT_READY	コマンド受信準備ができていない。	8
E_PMBUS_PACKET_S_CMD_NOT_SUPPORT	PMBUS 仕様でサポートしていないコマンドを受信。	9
E_PMBUS_PACKET_S_ARA_COMP	SendARA の処理完了。	10

p_pmbusSlaveCmdCallback

コールバック関数型名	void (* p_pmbusSlaveCmdCallback)(st_pmbus_nwk_ctrl_s_t *p_st_nwk_ctrl, e_pmbus_packet_result_s_t e_pmbus_result)
説明	PMBus の割り込みハンドラから呼び出すコールバック関数。ユーザが R_PMBUS_Slave_Open()時に指定します。 ユーザは本コールバック関数で、コマンドコードに対応した実行処理を実装してください。
宣言しているヘッダファイル	r_pmbus_app_slave.h
備考	-
引数名	**説明**
st_pmbud_nwk_ctrl_s_t *p_st_pmbus_nwk_ctrl	PMBus Middleware のネットワークコントロール情報の格納変数へのポインタ。
e_pmbus_packet_result_s e_pmbus_result	PMBus Middleware のパケット情報を格納変数へのポインタ。

st_pmbus_cfg_s_t

構造体名	st_pmbus_cfg_s_t
説明	PMBus Slave のコンフィグレーションデータ用の構造体です。 R_PMBUS_Slave_Open の引数の型として使用します。コンフィグレーションデータを PMBus Slave Middleware の内部グローバル変数へ登録するために使用します。
宣言しているヘッダファイル	r_pmbus_app_slave.h
備考	-
メンバ名	**説明**

uint16_t u2_rx_size	*p_u1_rx_buf のサイズ。（1～35）
uint8_t *p_u1_rx_buf	受信データ格納バッファへのポインタ。master から受信したデータは本バッファへ格納します。
uint16_t u2_tx_size	*p_u1_tx_buf のサイズ。（1～35）
uint8_t *p_u1_tx_buf	送信データ格納バッファへのポインタ。master へ送信するデータを格納します。
uint8_t u1_slave_addr	自スレーブアドレス。(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)	PMBus の割り込みハンドラから呼び出すコールバック関数。ユーザは本関数内でコマンドに対応した motor module の制御処理を実行してください。

- PMBus Slave Middleware 関数で使用するデータタイプ、構造体

e_pmbus_nwk_status_s_t

列挙型名	e_pmbus_nwk_status_s_t	
説明	PMBus network layer(slave)の内部状態を示す列挙型です。PMBus Slave Middleware の内部状態を管理するために使用します。	
宣言しているヘッダファイル	r_pmbus_app_slave.h	
備考	-	
要素名	**説明**	**値**
E_PMBUS_NWK_STATUS_S_IDLE	新しいパケットの受信待ち。	0
E_PMBUS_NWK_STATUS_S_START	スタートコンディションを検出。	1
E_PMBUS_NWK_STATUS_S_RX	パケットを受信中。	2
E_PMBUS_NWK_STATUS_S_TX	Receive Byte パケットを送信中。	3
E_PMBUS_NWK_STATUS_S_TX_RESP	パケットを送信中。	4
E_PMBUS_NWK_STATUS_S_ERROR	パケットエラーを検出。	5

e_pmbus_int_event_s_t

列挙型名	e_pmbus_int_event_s_t	
説明	I2C のエラー検出割り込みの発生要因を示す列挙型です。PMBus Slave Middleware の内部処理で割り込み要因ごとの処理を実行するために使用します。	
宣言しているヘッダファイル	r_pmbus_app_slave.h	
備考	-	
要素名	**説明**	**値**
E_PMBUS_INT_EVENT_S_NONE	割り込み未検出	0
E_PMBUS_INT_EVENT_S_ARB_LOST	アービトレーションロストを検出。	1
E_PMBUS_INT_EVENT_S_TIMEOUT	タイムアウトを検出。	2
E_PMBUS_INT_EVENT_S_NACK	NACK 受信を検出。	3
E_PMBUS_INT_EVENT_S_START	Start コンディションを検出。	4
E_PMBUS_INT_EVENT_S_STOP	Stop コンディションを検出。	5
E_PMBUS_INT_EVENT_S_START_UNEXPECTED	想定外のタイミングでの Start コンディションを検出	6

st_pmbus_nwk_ctrl_s_t

構造体名	st_pmbus_nwk_ctrl_s_t
説明	PMBus network Layer (slave) パラメータを管理する構造体です。PMBus Slave Middleware 内部の通信状態を管理するために使用します。
宣言しているヘッダファイル	r_pmbus_app_slave.h
備考	-
メンバ名	説明
volatile e_pmbus_nwk_status_s_t e_status	Network layer のステータス
uint8_t u1_current_addr_rw	現在実行中のスレーブアドレス（RW 情報を含む）
uint8_t u1_current_cmd	現在実行中のコマンド
uint16_t u2_rx_index	現在の受信データバイト数
uint16_t u2_rx_len	予定する受信データバイト数
uint16_t u2_rx_size	*p_u1_rx_buf のサイズ
uint8_t *p_u1_rx_buf	受信データ格納バッファへのポインタ
uint16_t u2_tx_index	現在の送信データバイト数
uint16_t u2_tx_len	予定する送信データバイト数
uint16_t u2_tx_size	*p_u1_tx_buf のサイズ
uint8_t *p_u1_tx_buf	送信データ格納バッファへのポインタ
uint8_t u1_pec	現在の PEC 演算結果

st_pmbsu_ctrl_s_t

構造体名	st_pmbus_ctrl_s_t
説明	PMBus Middleware (slave)のコントロールデータ構造体です。PMBus Slave Middleware の設定情報、および、通信状態を管理するために使用します。
宣言しているヘッダ	r_pmbus_app_slave.h
備考	-
メンバ名	説明
st_pmbus_nwk_ctrl_s_t st_nwk_ctrl_s	PMBus network Layer (slave) のパラメータ管理する構造体
volatile e_pmbus_packet_result_s_t e_pmbus_result_s	PMBus 通信 (slave) の実行結果
bool b_open_flag	OPEN 状態（false: Open していない、true: Open している。）
bool b_pec_flag	PEC の有効無効情報（false:無効、true:有効）
uint8_t u1_own_slave_addr	自身のスレーブアドレス。
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)	PMBus の割り込みハンドラから呼び出すコールバック関数。

5.2.6　PMBus Slave グローバル変数一覧

本制御プログラムで使用するグローバル変数一覧を次に示します。

表 37 PMBus Slave Application グローバル変数一覧

File 名	グローバル変数	用途
r_app_main.c	static uint8_t s_u1_pmbus_config_data	ON_OFF_CONFIG コマンドの指定値を管理するグローバル変数です。本デモ開発では、bit3 のみが有効です。
	static uint8_t s_u1_pmbus_operation_data	OPERATION コマンドの指定値を管理するグローバル変数です。本デモ開発では、bit7 のみが有効です。
	static uint8_t s_u1_pmbus_tx_buf[PMBUS_BUF_SIZE_MAX]	PMBUS 用の送信データを格納するバッファ。本バッファを s_user_pmbus_cfg のメンバ*p_u1_tx_buf に設定し、PMBus Middleware 内部で使用します。
	static uint8_t s_u1_pmbus_rx_buf[PMBUS_BUF_SIZE_MAX]	PMBUS 用の受信データを格納するバッファ。本バッファを s_user_pmbus_cfg のメンバ*p_u1_rx_buf に設定し、PMBus Middleware 内部で使用します。
	static volatile e_app_req_event_t s_e_req_status	PMBUS コールバックの発生要因、または、その他のイベントの発生を main 処理へ通知するために使用する変数。
	static st_pmbus_cfg_s_t s_st_pmbus_cfg	R_PMBUS_Slave_Open で登録するコンフィグレーションデータを格納する変数。

表 38 PMBus Slave Middleware グローバル変数一覧

File 名	グローバル変数	用途
r_pmbus_app_slave.c	static st_pmbus_ctrl_s_t g_st_pmbus_ctrl	PMBUS Slaver Middleware のコントロール情報を管理するグローバル変数です。PMBUS Slave Middleware の内部でのみ使用します。
Config_RIIC0.c	volatile uint8_t g_riic0_user_slave_addr	RX26T において、R_PMBUS_Slave_Open によりユーザが指定したスレーブアドレスを SARL0 レジスタに設定するために使用します。
	volatile uint8_t g_riic0_start_detect_at_receive	RX26T において、RIIC0 ドライバの g_riic0_mode_flag が受信モードの時に Start コンディション検出割り込みを検出したことを管理するフラグ。
r_pmbus_wrapper_slave.c	static gpt_instance_ctrl_t s_st_gpt_ctrl	RA6T3 において、GPT ドライバのコントロールハンドラとして使用する変数。
	static smbus_slave_instance_ctrl_t s_st_smbus_ctrl	RA6T3 において、SMBUS ドライバのコントロールハンドラとして使用する変数。
	static smbus_slave_cfg_t s_st_smbus_cfg	RA6T3 において、R_SMBUS_SLAVE_Oepn により SMBUS ドライバに設定するコンフィグレーションデータを格納する変数。
	smbus_slave_instance_t g_smbus_slave0	RA6T3 において、SMBUS ドライバの内部で使用するインスタンス。

5.2.7　PMBus Slave マクロ定義一覧

本制御プログラムで使用するマクロ定義一覧を次に示します。

表 39 PMBus Slave Application マクロ定義一覧

File 名	マクロ名	用途	定義値
r_app_main.h	PMBUS_APP_CMD_xxx	・PMBUS のコマンドコードの定義。（マクロ名は以下参照）	-
		PMBUS_APP_CMD_OPERAION ： OPERAION コマンド	0x01
		PMBUS_APP_CMD_ON_OFF_CONFIG ： ON_OFF_CONFIG コマンド	0x02
		PMBUS_APP_CMD_CLEAR_FAULTS ： CLEAR_FAULTS コマンド	0x03
		PMBUS_APP_CMD_STATUS_FAN_1_2 ： STATUS_FAN_1_2 コマンド	0x81
		PMBUS_APP_CMD_READ_VOUT ： READ_VOUT コマンド	0x8B
		PMBUS_APP_CMD_READ_IOUT ： READ_IOUT コマンド	0x8C
		PMBUS_APP_CMD_READ_FAN_SPEED_1 ： REA_DFAN_SPEED_1 コマンド	0x90
		PMBUS_APP_CMD_READ_FREQUENCY ： READ_FREQUENCY コマンド	0x95
		PMBUS_APP_CMD_RESERVED ： RESERVED コマンド	0x09
		PMBUS_APP_CMD_PMBUS_REVISION ： PMBUS_REVISION コマンド	0x98
		PMBUS_APP_CMD_STORE_DEFAULT_CODE ： STORE_DEFAULT_CODE コマンド	0x13
		PMBUS_APP_CMD_FAN_COMMAND_1 ： FAN_COMMAND_1 コマンド	0x38
		PMBUS_APP_CMD_READ_EOUT ： READ_EOUT コマンド	0x87
		PMBUS_APP_CMD_PAGE_PLUS_WRITE ： PAGE_PLUS_WRITE コマンド	0x05
	SLAVE_ADDRESS	PMBus Slave Middleware に設定するスレーブアドレスの定義です。	0x0A
	OPERATION_OPE_START_BIT	OPERATION コマンドのモータ制御指示 bit の定義。	0x80
	ON_OFF_CONFIG_OPE_ENABLE_BIT	ON_OFF_CONONFIG コマンドで使用するデータ定義。モータ制御の際に OPERATION コマンドのモータ制御指示ビットの設定を使用するかを指定します。	0x08
	RET_OK	アプリケーション内部関数の戻り値。正常終了。	0
	RET_ERROR	アプリケーション内部関数の戻り値。異常終了。	1

表 40 PMBus Slave API マクロ定義一覧

File 名	マクロ名	用途	定義値
r_pmbus_app_slave.h	PMBUS_RET_xxx	・PMBus middleware API から返却するエラーコード。（マクロ名は以下参照）	-
		PMBUS_RET_OK ： 正常終了	0
		PMBUS_RET_ERROR ： 異常終了。原因の詳細は st_pmbus_cfg_s_te_pmbus_result_s で確認してください	1
		PMBUS_RET_PARAM ： 指定された引数が異常。	2
		PMBUS_RET_NOT_OPENED ： OPEN されていません	3
		PMBUS_RET_OPENED ： すでに OPEN されています	4

表 41 PMBus Slave Middleware 関数のマクロ定義一覧

File 名	マクロ名	用途	定義値
r_pmbus_app_slave.h	PMBUS_TRANS_xxx	・各コマンドコードごとでサポートするプロトコルを判別する際に使用するトランザクションコードの定義。（マクロ名は以下参照）	-
		PMBUS_TRANS_RESERVED ： コマンドコードは RESERVED	0x00
		PMBUS_TRANS_READ_BYTE ： コマンドコードは READ BYTE トランザクションをサポート	0x01
		PMBUS_TRANS_READ_WORD ： コマンドコードは READ WORD トランザクションをサポート	0x02
		PMBUS_TRANS_BLOCK_READ ： コマンドコードは BLOCK READ トランザクションをサポート	0x03
		PMBUS_TRANS_SEND_BYTE ： コマンドコードは SEND BYTE トランザクションをサポート	0x10
		PMBUS_TRANS_WRITE_BYTE ： コマンドコードは WRITE BYTE トランザクションをサポート	0x20
		PMBUS_TRANS_WRITE_WORD ： コマンドコードは WRITE WORD トランザクションをサポート	0x30
		PMBUS_TRANS_BLOCK_WRITE ： コマンドコードは BLOCK WRITE トランザクションをサポート	0x40
		PMBUS_TRANS_WRITE_QUICK ： コマンドコードは Write Quick トランザクションをサポート	0x50
		PMBUS_TRANS_PROCESS_CALL ： コマンドコードは PROCESS CALL トランザクションをサポート	0x60
		PMBUS_TRANS_BLOCK_PROCESS_CALL ： コマンドコードは Block Write-Block Read Process Call トランザクションをサポート	0x70
	PMBUS_COMMAND_CODE_MAX	PMBus Middleware でサポートするコマンドの最大数。	256
	PMBUS_BLOCK_SIZE_MIN	Block コマンドで送受信できるデータ数の最小値。	1
	PMBUS_BLOCK_SIZE_MAX	Block コマンドで送受信できるデータ数の最大値。	32
	PMBUS_BUF_SIZE_MIN	Open 時に PMBUS Middleware に登録できるバッファサイズの最小値。	1
	PMBUS_BUF_SIZE_MAX	Open 時に PMBUS Middleware に登録できるバッファサイズの最大値。（Block Read/Block Write 時の書き込み最大データ数(32)+	PMBUS_BLOCK_SI

		コマンドコード(1)+書き込みデータ数/読み出しデータ数(1)+PEC(1))	ZE_MAX + 3
	PMBUS_CRC8_USE_IP	PEC の演算方法を指定する定義値。"1(MCU に搭載している演算器を使用する)"か、"0(テーブルを使用して演算するかを指定する)"かを設定する。MCU に搭載している演算器を使用する場合は、r_pmbus_nwk_AddCrc8()関数に CRC 演算器を使用したコードをユーザが実装した上で"1(MCU に搭載している演算器を使用する)"に変更してください。	0
	PMBUS_ALERT_RESPONSE_ADDR	R_PMBUS_Slave_SendARA を実行時に設定する ALERT 情報応答用のスレーブアドレス(ARA)。SMBus の仕様に準拠した値を定義しています。	0x0C
	PMBUS_SLAVE_ADDR_MAX	PMBUS のスレーブ API の引数で指定できるスレーブアドレスの最小値の定義です。	0x01
	PMBUS_SLAVE_ADDR_MAX	PMBUS のスレーブ API の引数で指定できるスレーブアドレスの最大値の定義です。	0x10
r_pmbus_wrapper_slave.h	PMBUS_CFG_DEVICE_RX26T	r_pmbus_wrapper_slave.c で用意しているドライバの wrapper 関数を RX26T で使用する場合に使用する。	0
	PMBUS_CFG_DEVICE_RA6T3	r_pmbus_wrapper_slave.c で用意しているドライバの wrapper 関数を RA6T3 で使用する場合に使用する。	1
	PMBUS_CFG_DEVICE	r_pmbus_wrapper_slave.c で用意しているドライバの wrapper 関数を動作させる MCU の定義。値は PMBUS_CFG_DEVICE_RX26T か PMBUS_CFG_DEVICE_RA6T3 を指定してください。	使用する MCU に従う。

5.2.8　PMBus Slave 制御フロー

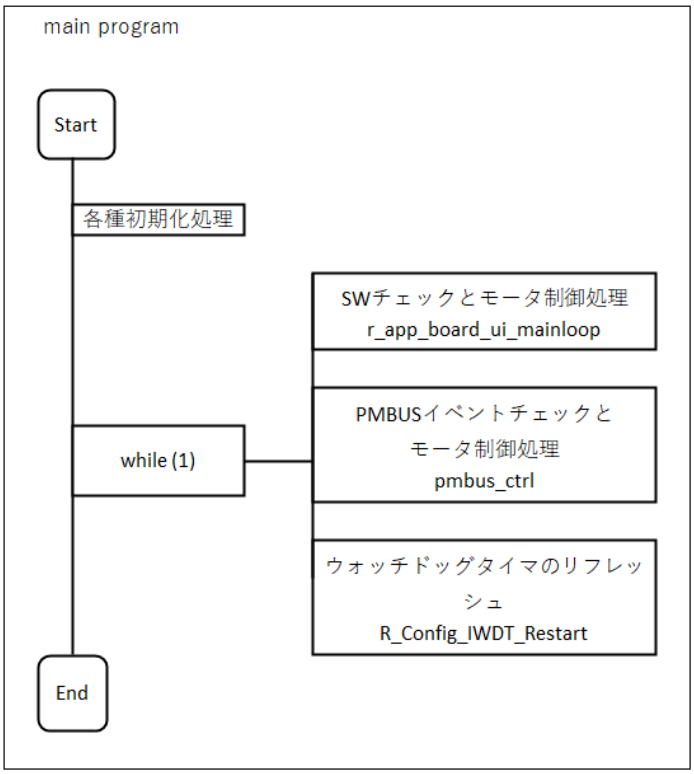
PMBus Slave Application 部のフローを 5.2.8.1 項に、PMBus Slave API 部のフローを 5.2.8.2 項に、PMBus Slave ドライバ部のフローを 5.2.8.3 項に示します。PMBus Slave Middleware 部についてはプロジェクトコードをご参照ください。

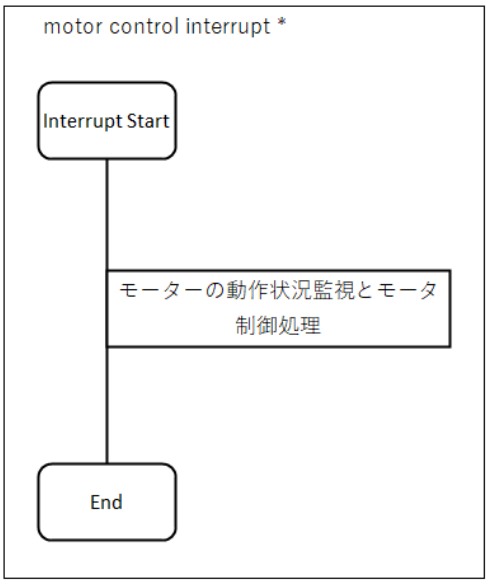
5.2.8.1　PMBus Slave Application 部フロー

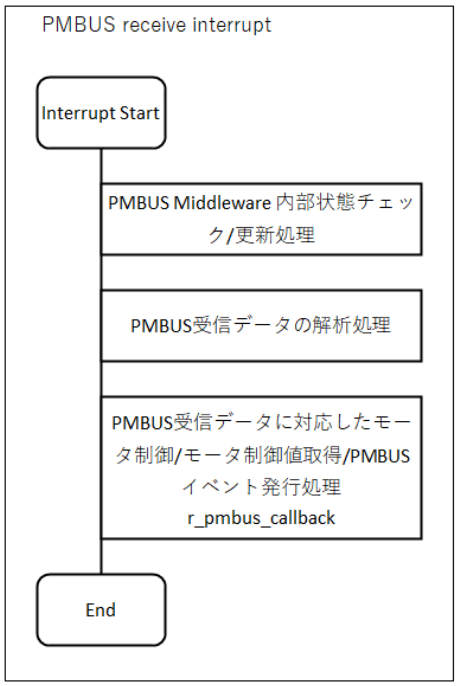
PMBus Slave Application 部はモータ制御ミドルウェアの制御、および PMBus Master と通信を行う PMBus Slave Middleware 部の API コールを行います。PMBus Slave Application 部のフローを以下に示します。

■　PMBus slave 全体概略フロー

●　PMBus slave 全体概略フロー (RX26T)

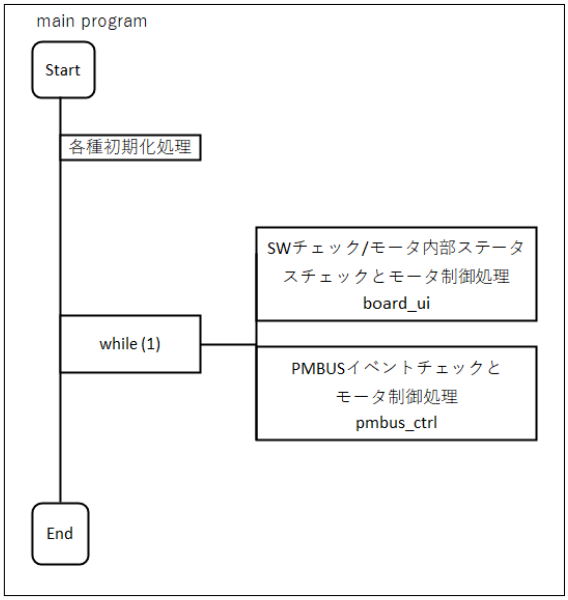


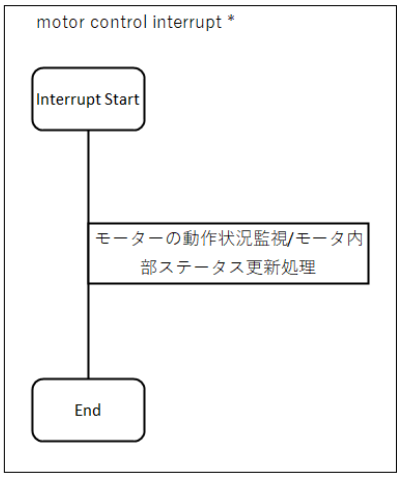


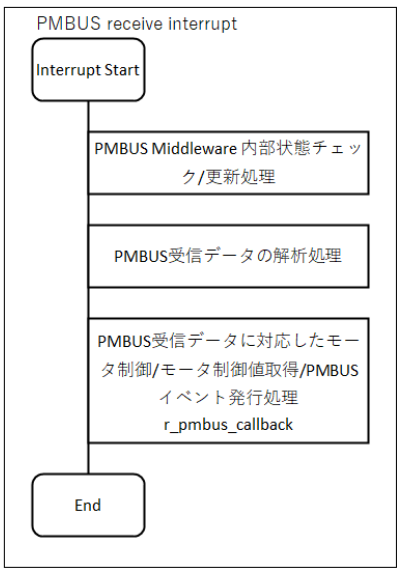


【注】詳細は RX ファミリ 永久磁石同期モータのセンサレスベクトル制御 - MCK 用をご参照ください。

- PMBus slave 全体概略フロー (RA6T3)





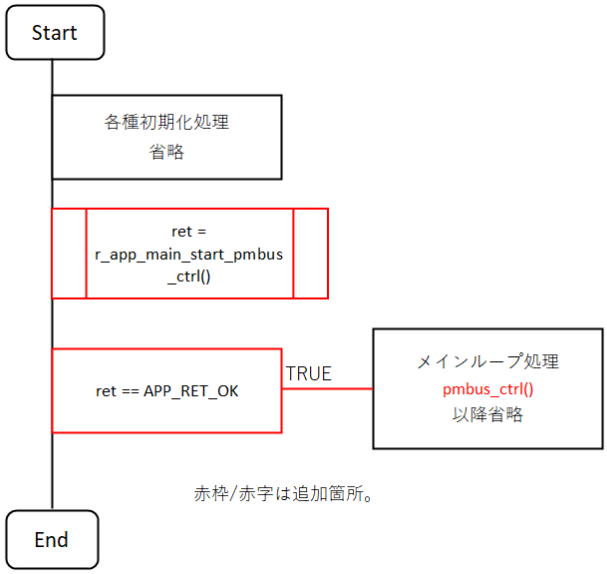


【注】詳細は 永久磁石同期モータのセンサレスベクトル制御 Renesas Flexible Motor Control シリーズ用をご参照ください。

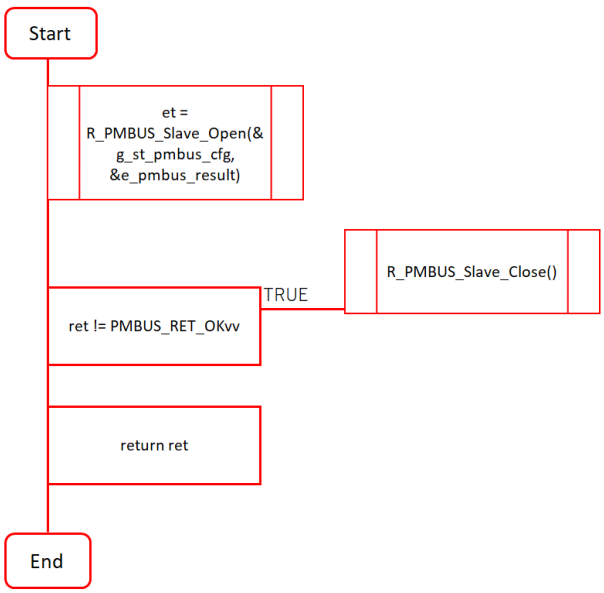
■　PMBus slave 関数 詳細フロー

- main (RX26T)、hal_entry (RA6T3)

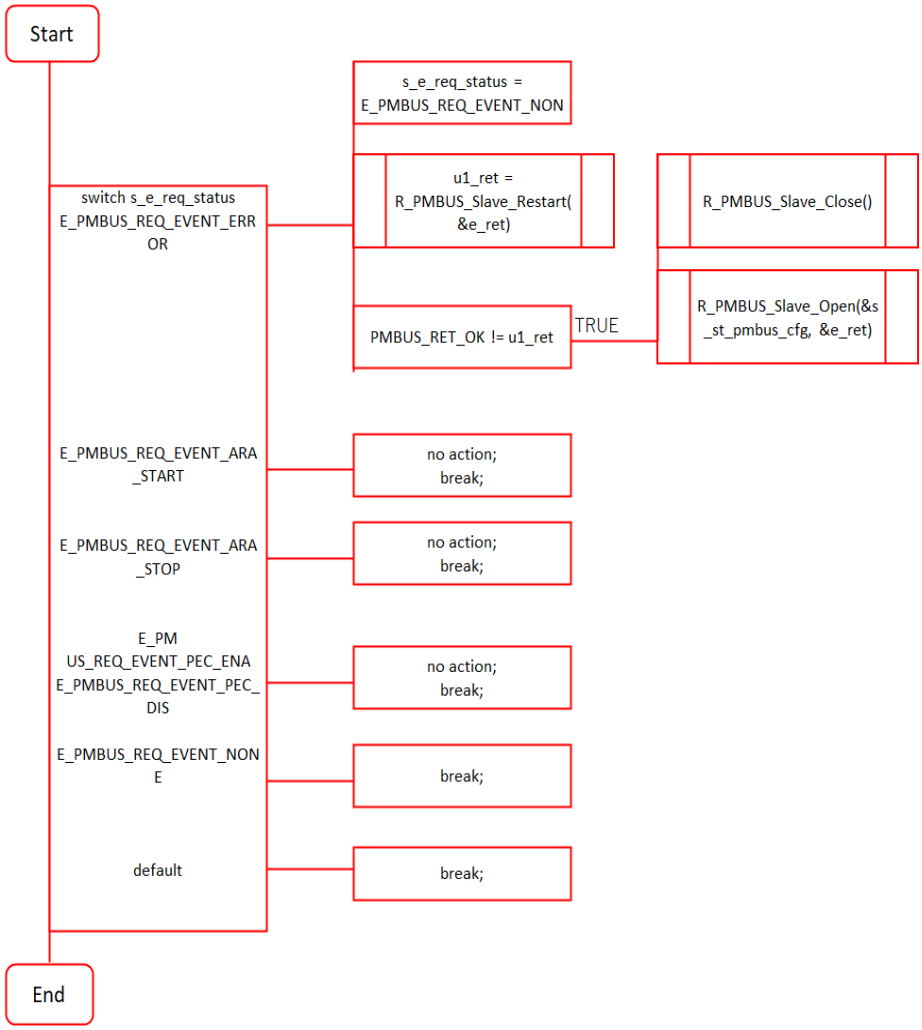
モータ制御ミドルウェアの Application 部は各種初期化処理、メインループ処理になります。詳細はモータミドルのアプリケーションノートをご参照ください。



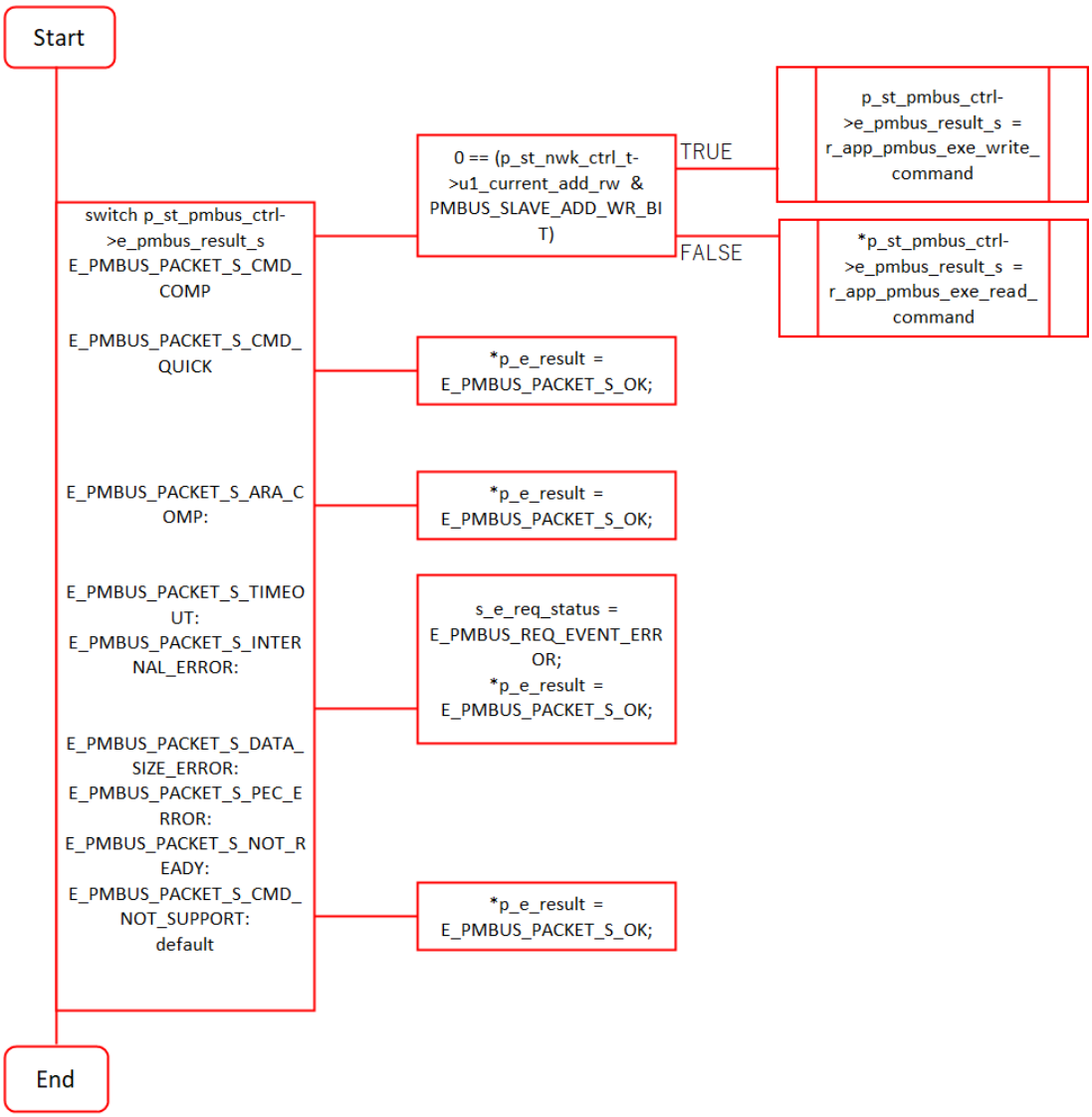
- r_app_main_start_pmbus_ctrl (RX26T、RA6T3)



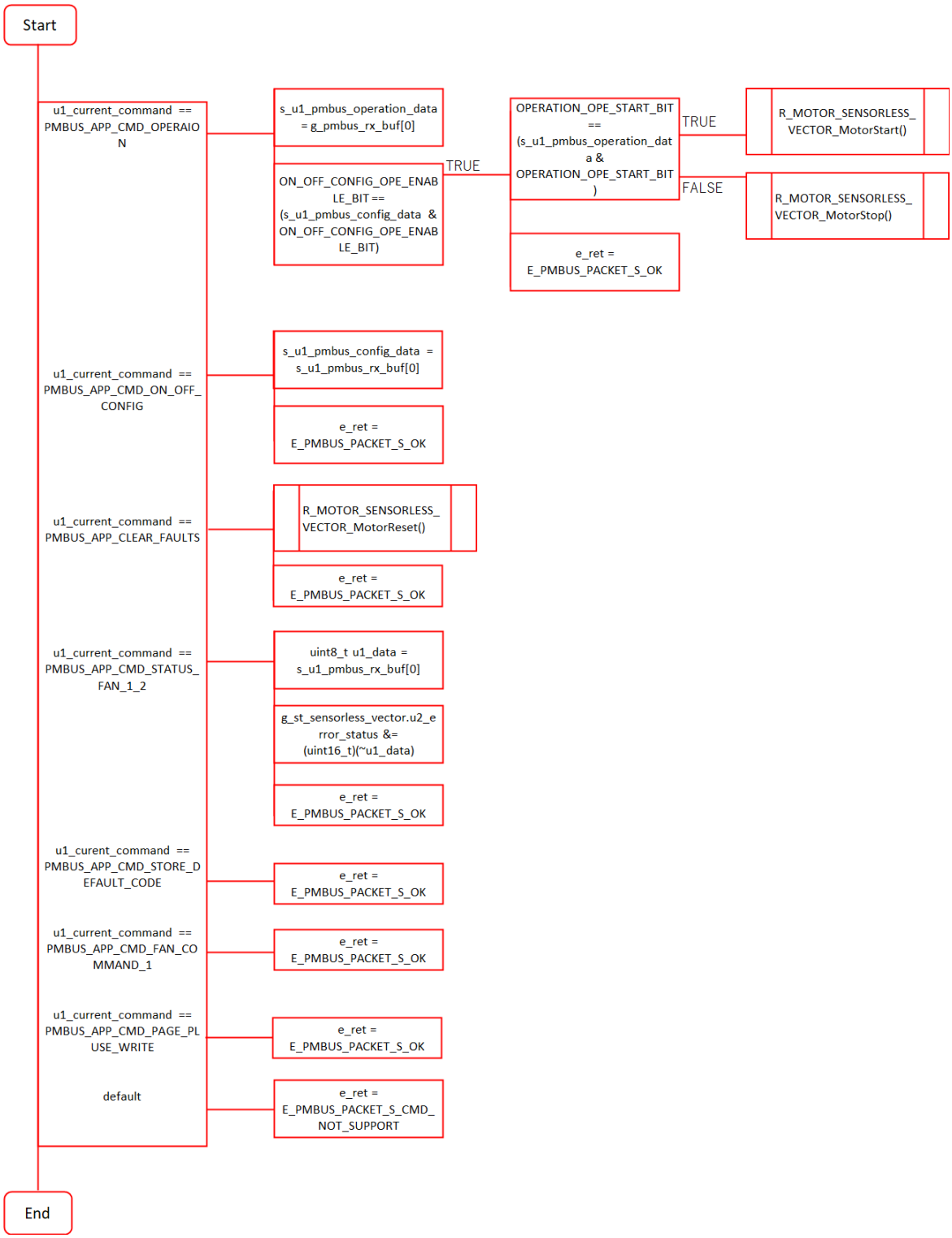
- pmbus_ctrl (RX26T、RA6T3)



- r_pmbus_callback（RX26T、RA6T3）

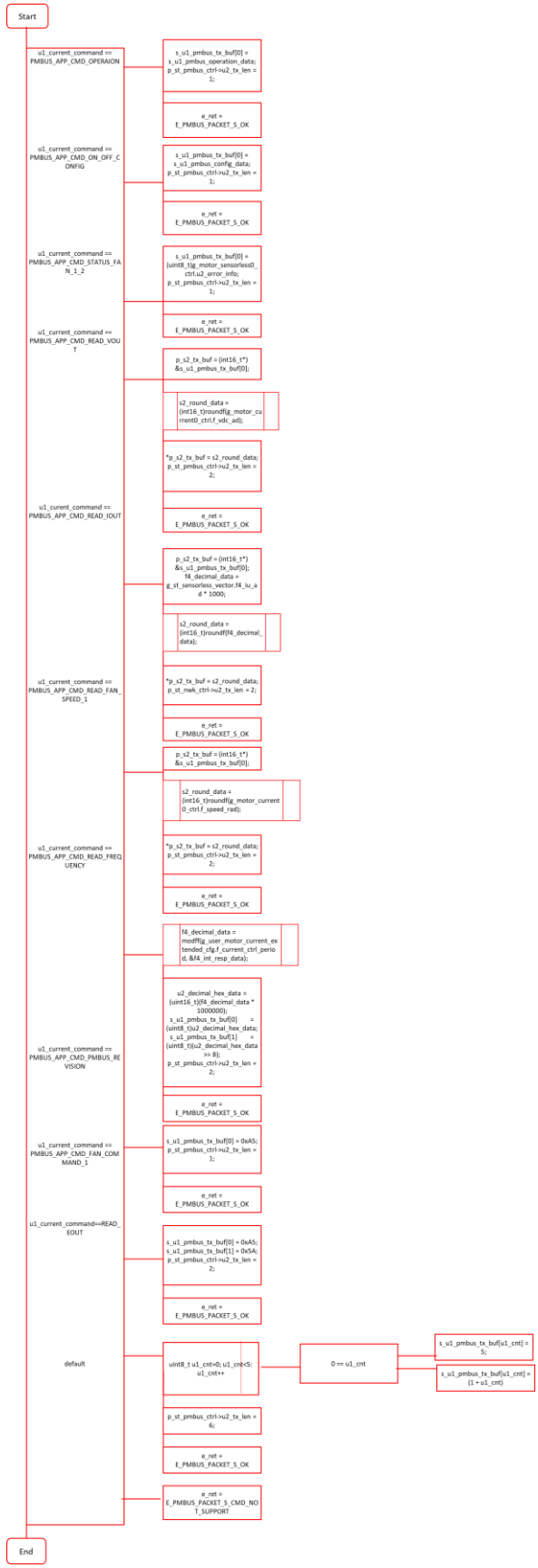


- r_app_pmbus_exe_write_command（RX26T、RA6T3）



【注】図中のモータサンプル API 名は RA6T3 の場合は異なります。詳細はプロジェクトをご参照ください。

- r_app_pmbus_exe_read_command（RX26T、RA6T3）

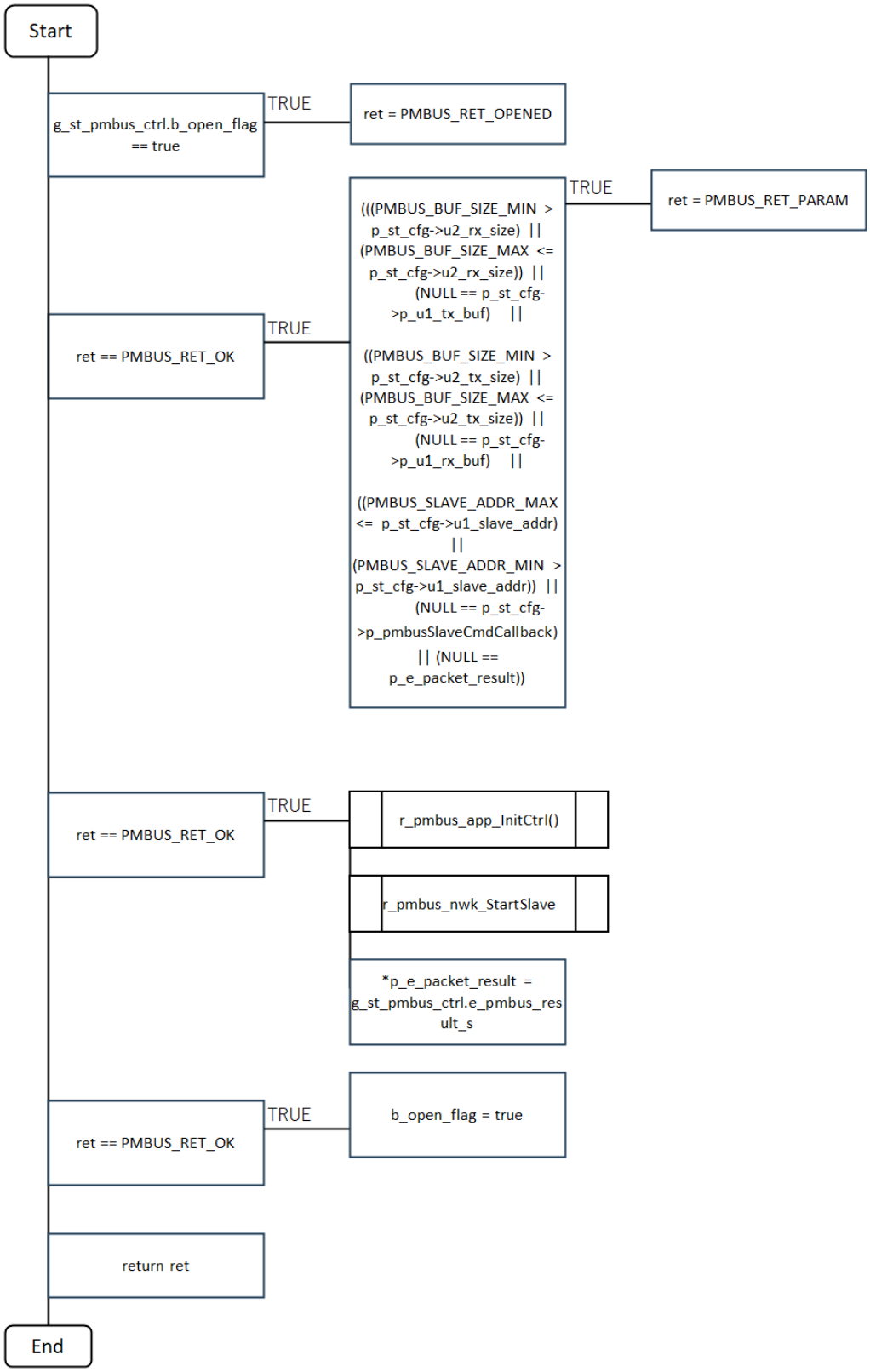


【注】図中のモータサンプル API 名は RA6T3 の場合は異なります。詳細はプロジェクトをご参照ください。

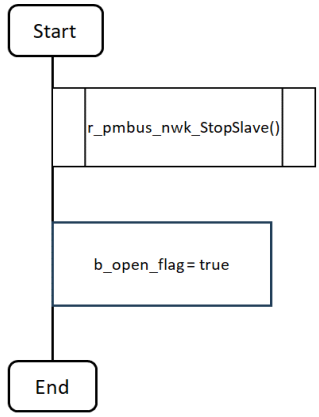
5.2.8.2 PMBus Slave API 部フロー

PMBus Slave API 部は PMBus Slave Middleware 部の制御を行います。PMBus Slave API 部のフローを以下に示します。

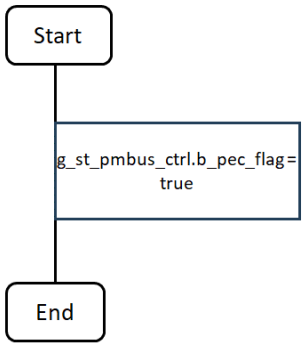
- R_PMBUS_Slave_Open



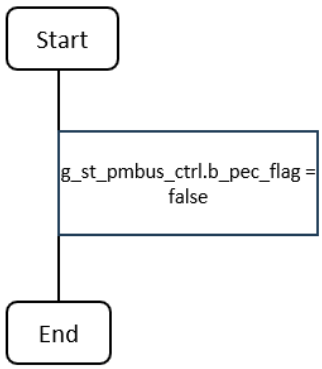
- R_PMBUS_Slave_Close



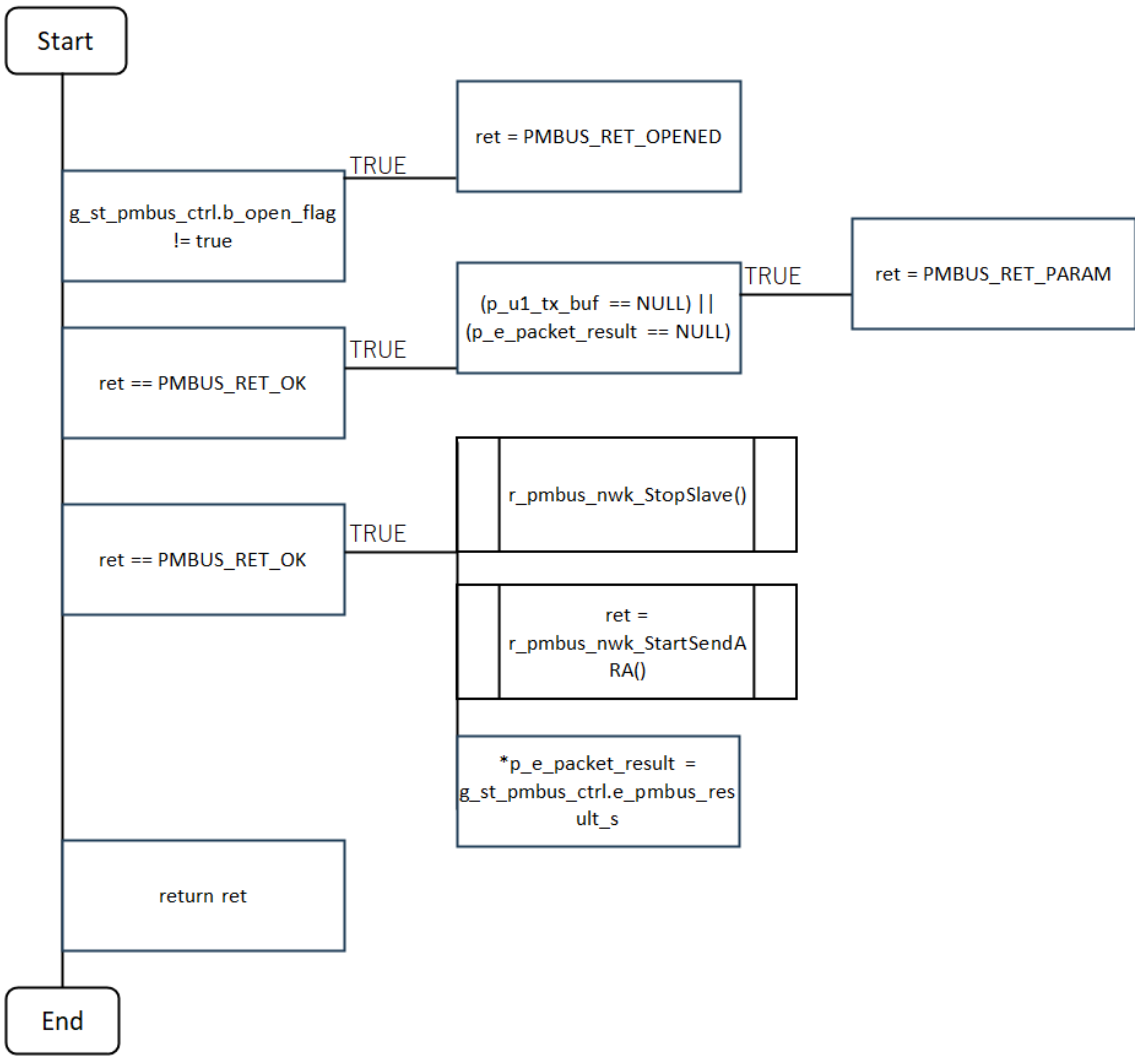
- R_PMBUS_Slave_EnablePEC



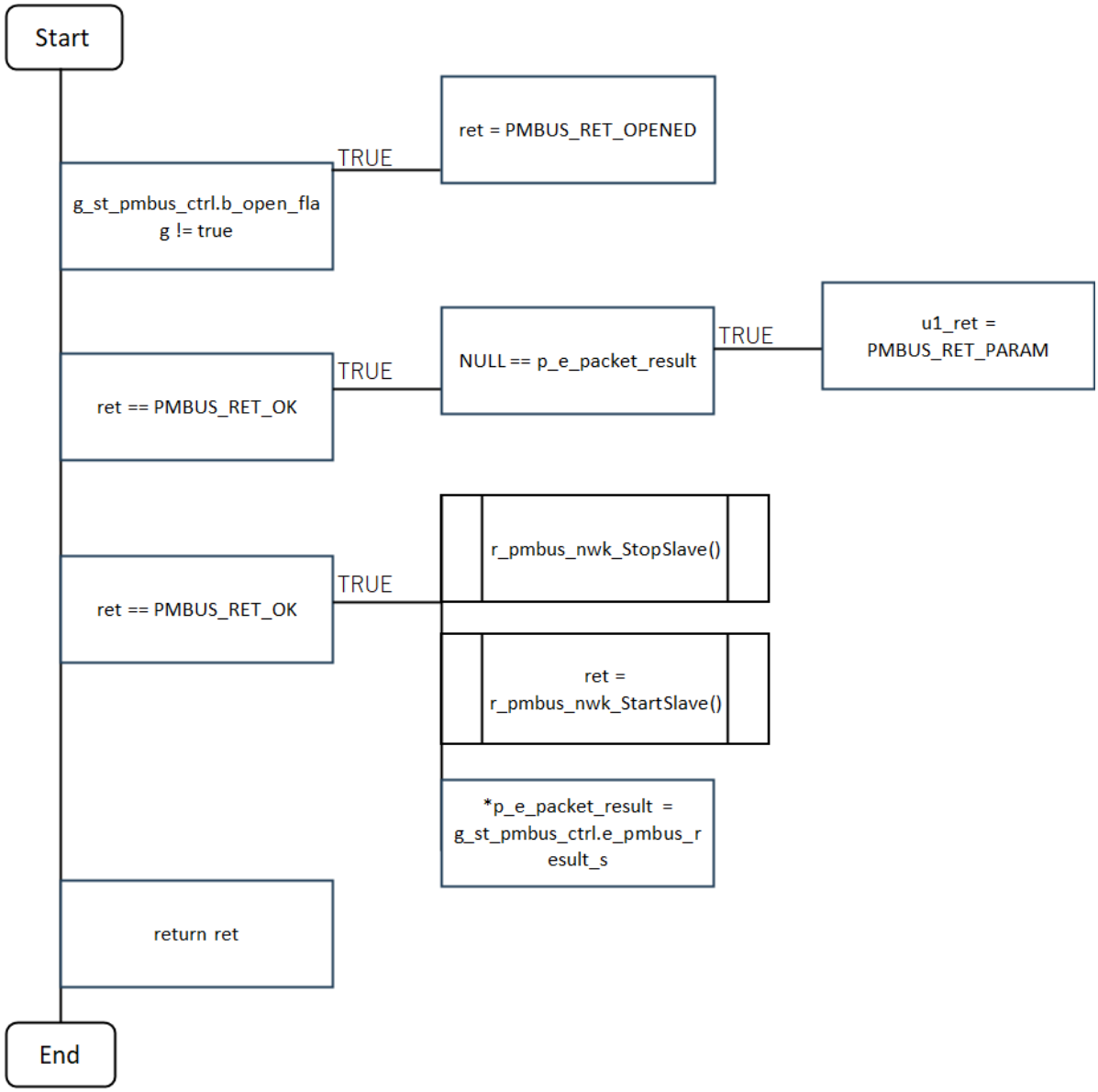
- R_PMBUS_Slave_DisablePEC



- R_PMBUS_Slave_SendARA



- R_PMBUS_Slave_Restart

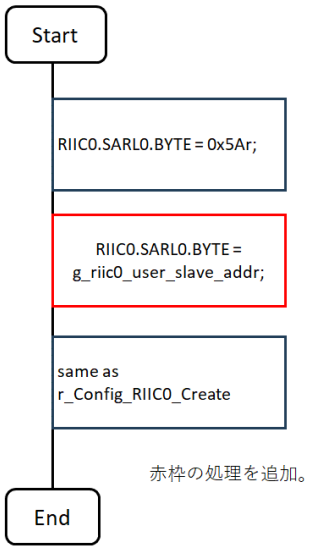


5.2.8.3 PMBus Slave ドライバ部フロー

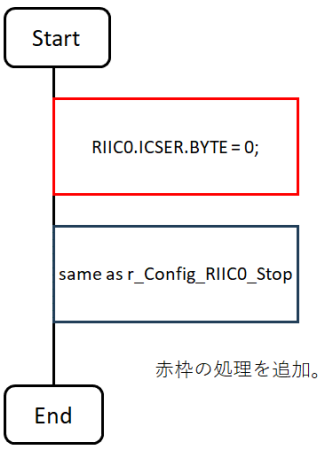
PMBus Slave ドライバ部は RX26T ではスマート・コンフィグレータで、RA6T3 では FSP で生成するコードから PMBus Slave 処理に合わせて一部変更しています。変更内容については 5.2.4 PMBus Slave ドライバ部のカスタマイズをご参照ください。各 PAD 図の赤枠部分および赤字部分が修正箇所、青枠部分および青文字部分が削除箇所となります。

(1) スマート・コンフィギュレータの関数（RX26T）

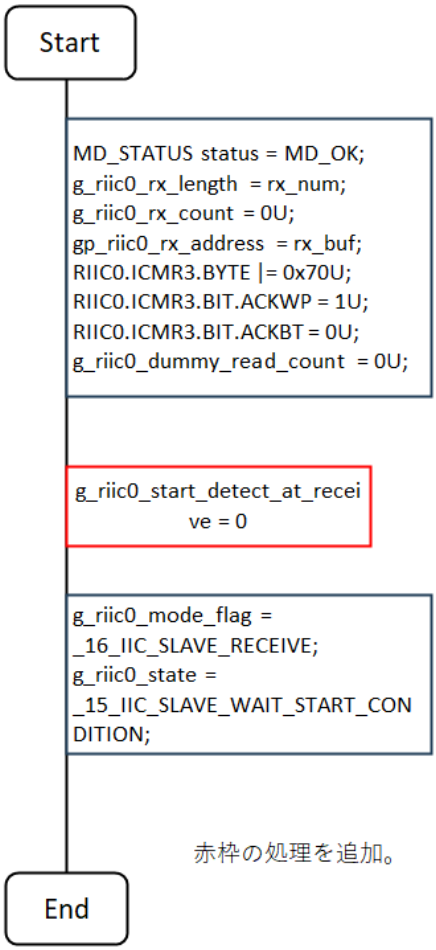
- R_Config_RIIC0_Create



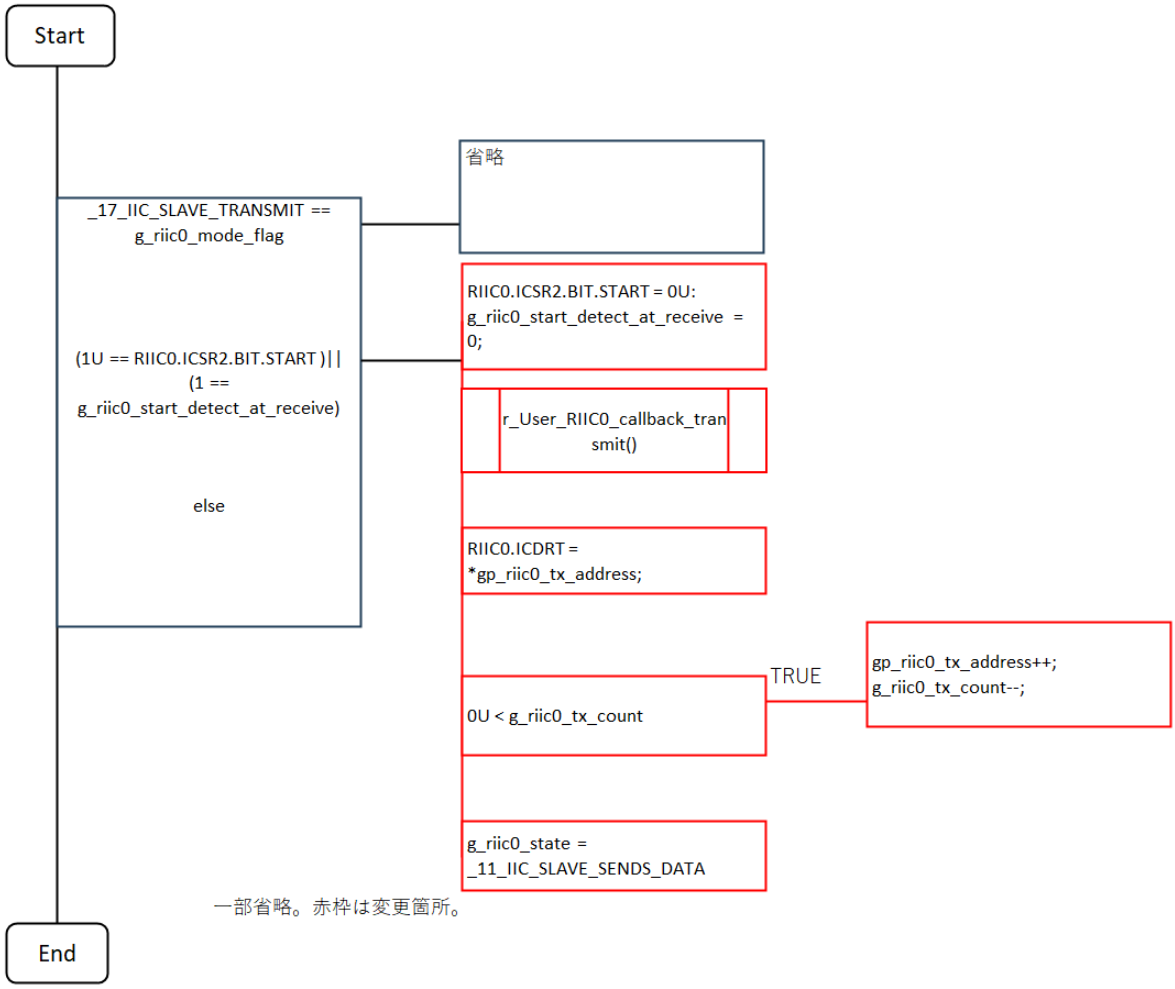
- R_Config_RIIC0_Stop



- r_Config_RIIC0_Slave_Receive

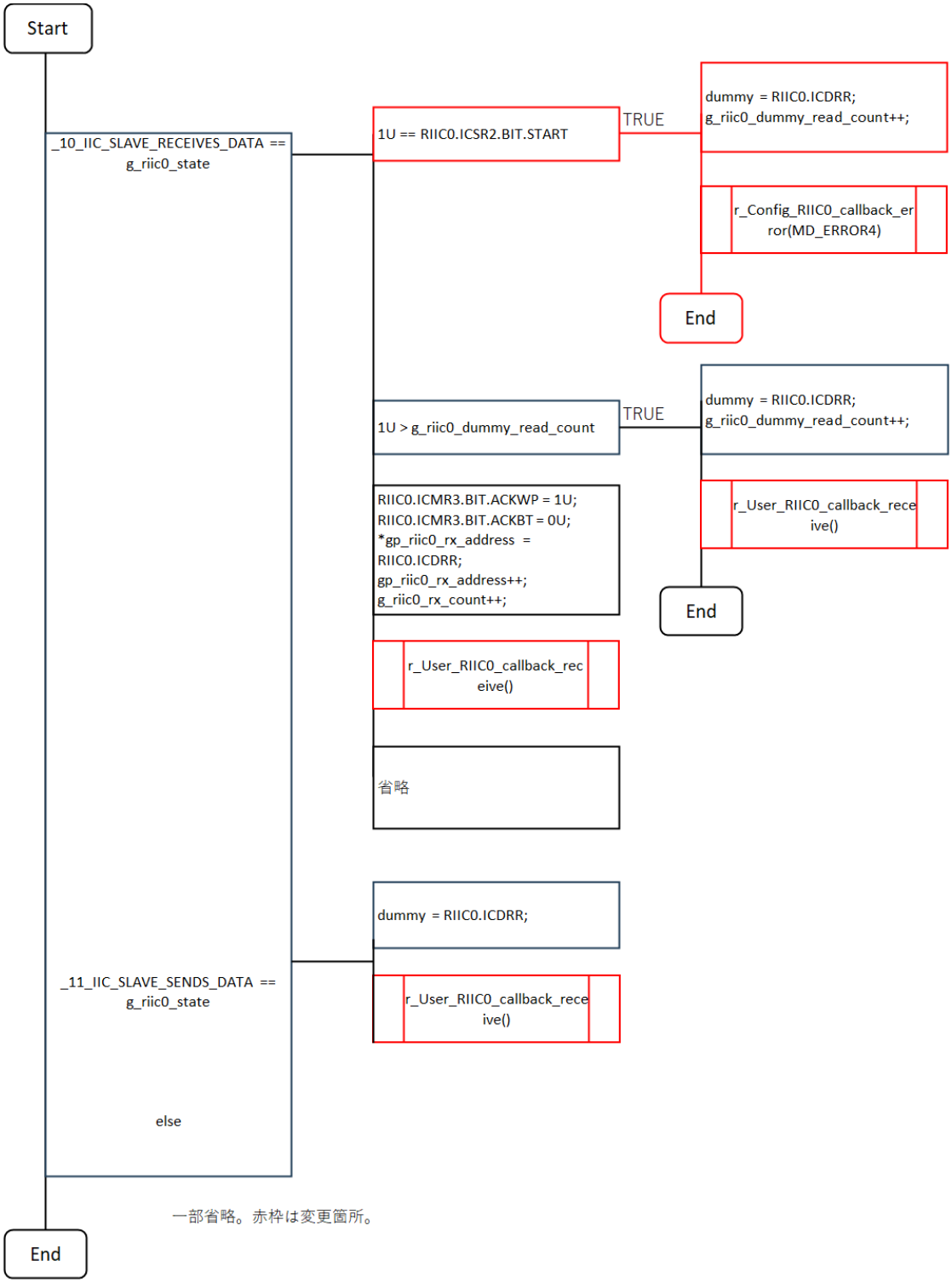


- r_Config_RIIC0_transmit_interrupt

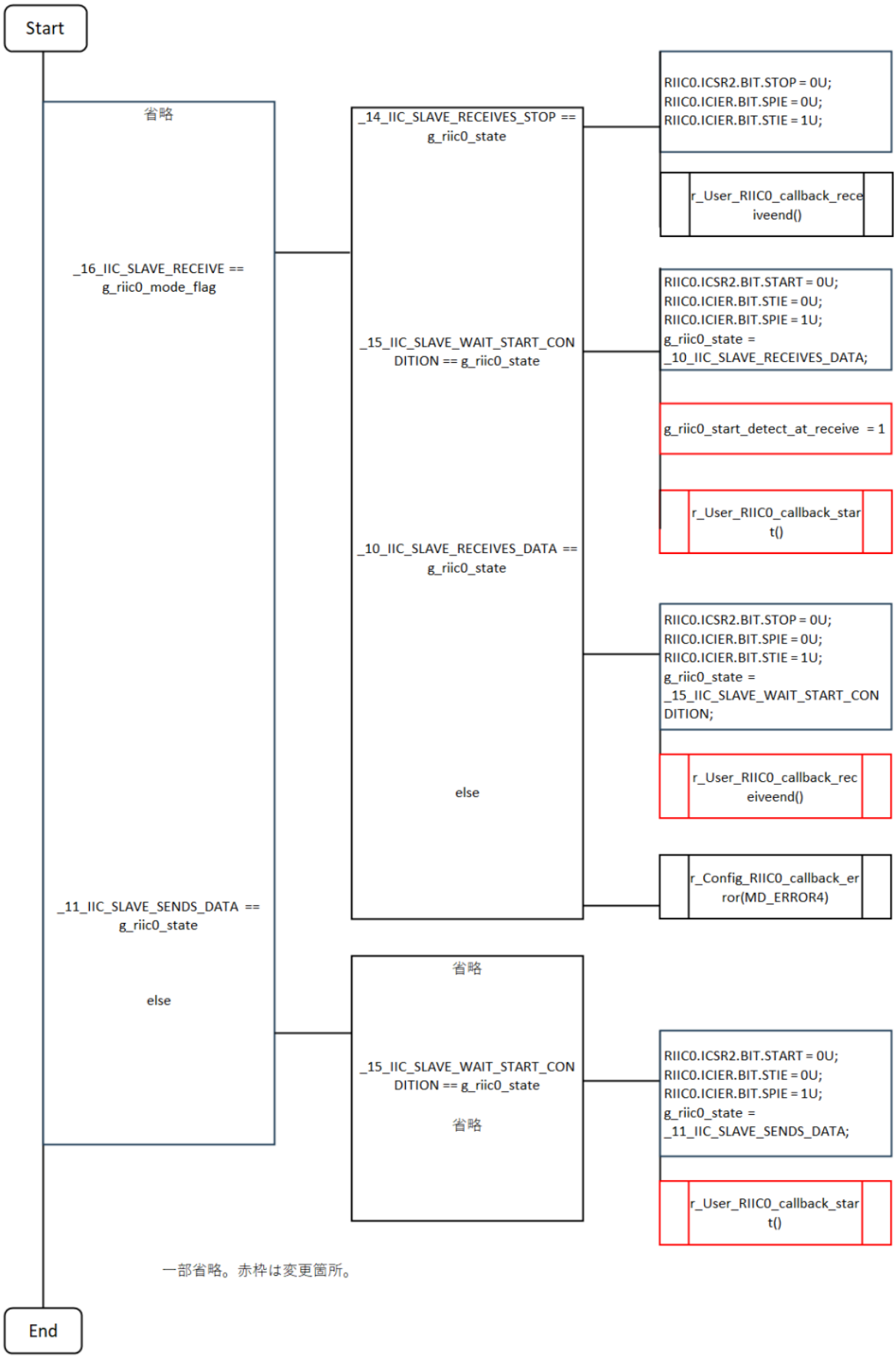


一部省略。赤枠は変更箇所。

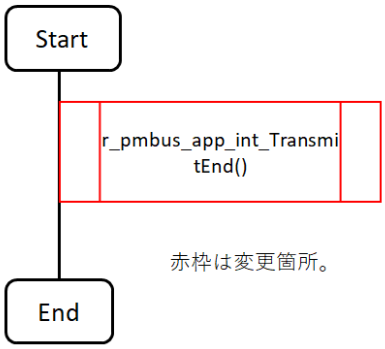
- r_Config_RIIC0_receive_interrupt



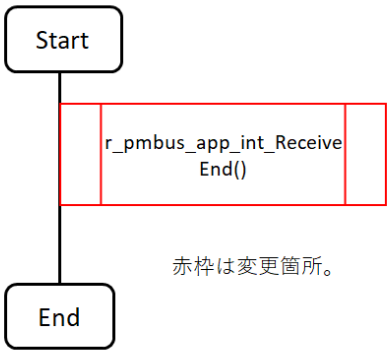
- r_Config_RIIC0_error_interrupt



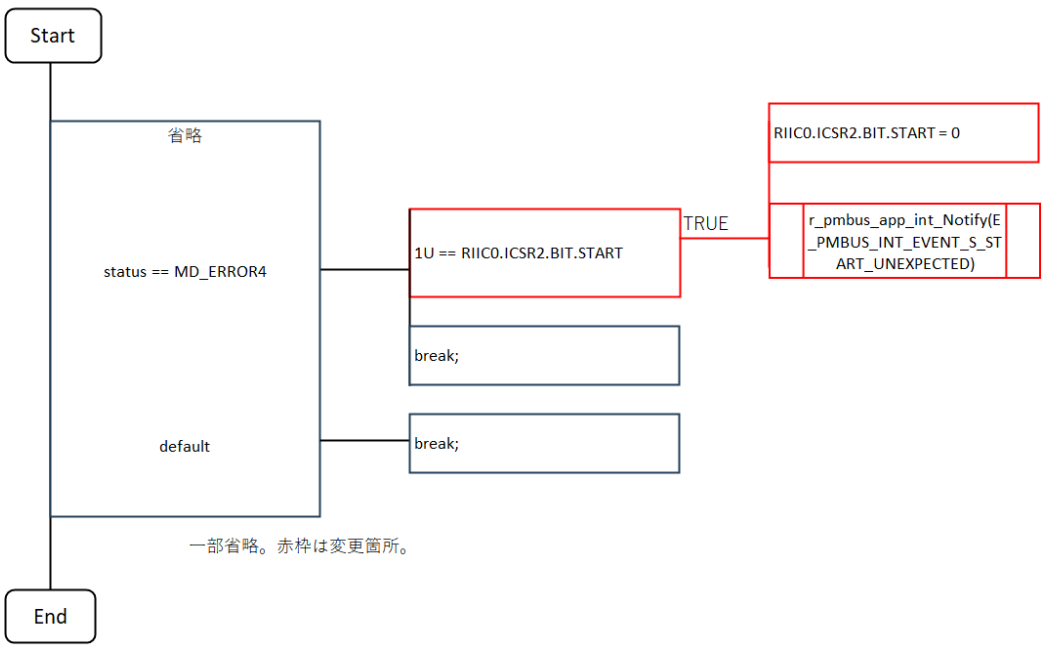
- r_Config_RIIC0_callback_transmitend



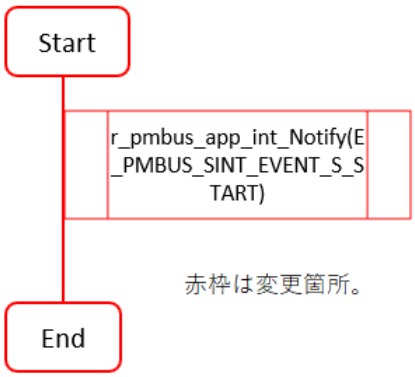
- r_Config_RIIC0_callback_receiveend



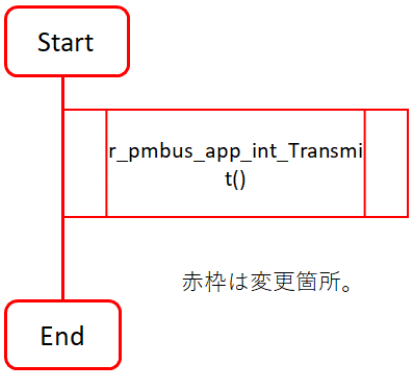
- r_Config_RIIC0_callback_error



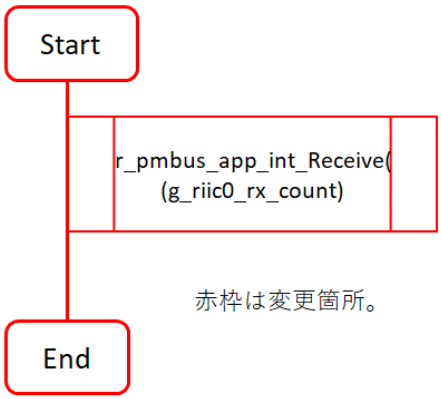
- r_User_RIIC0_callback_start



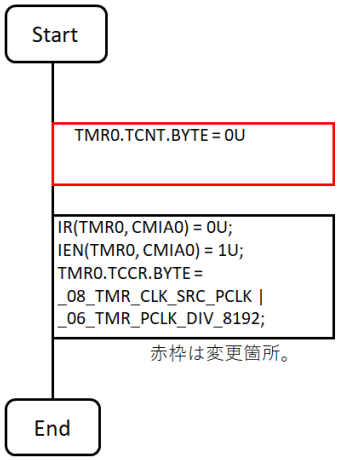
- r_User_RIIC0_callback_transmit



- r_User_RIIC0_callback_receive

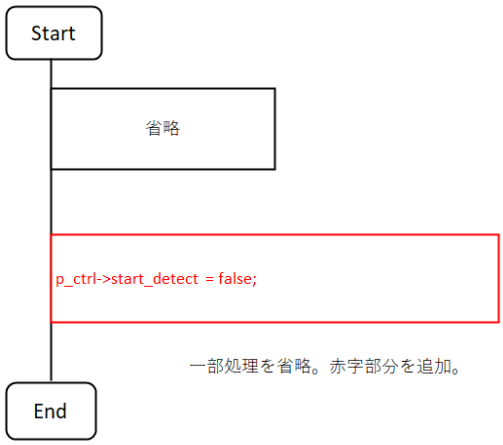


- R_Config_TMR0_Start

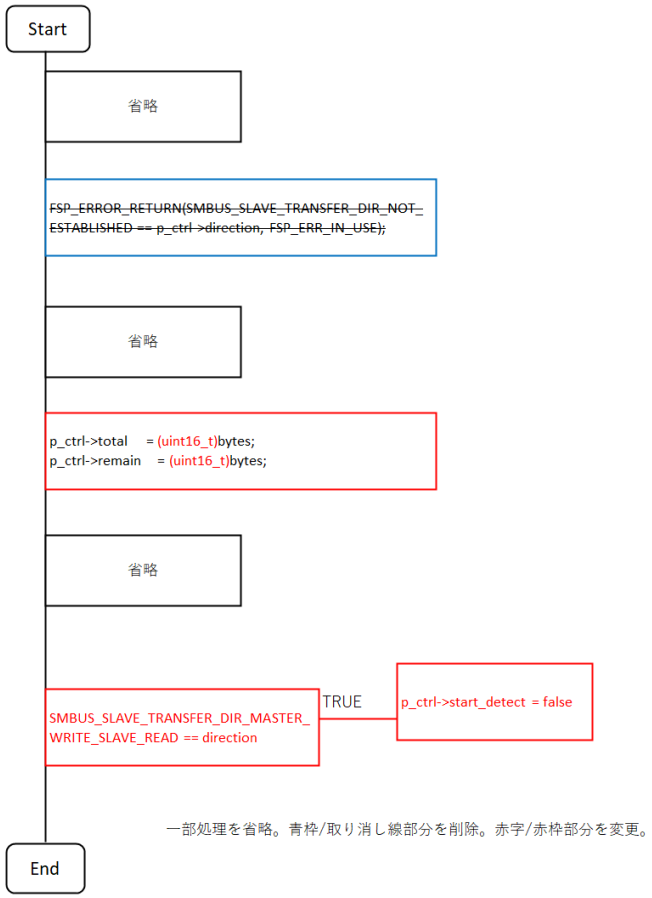


(2) FSP の関数（RA6T3）

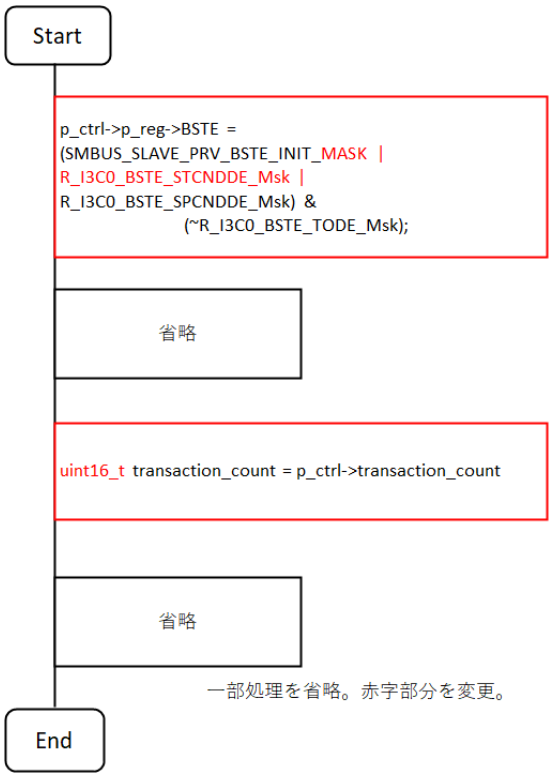
- R_SMBUS_SLAVE_Open



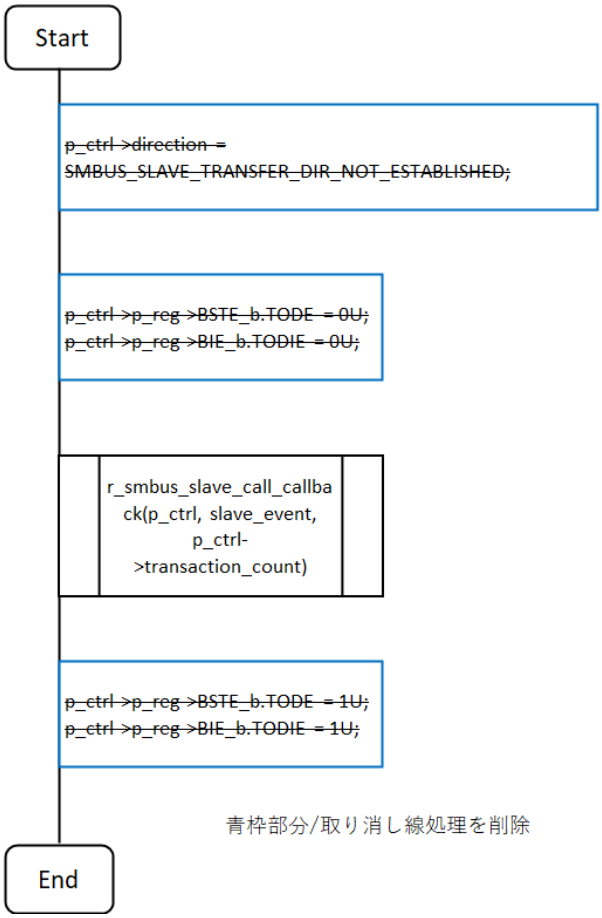
- r_smbus_slave_read_write



- r_smbus_slave_notify



- r_smbus_slave_callback_request

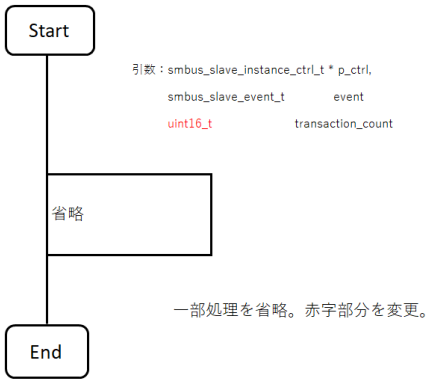


青枠部分/取り消し線処理を削除

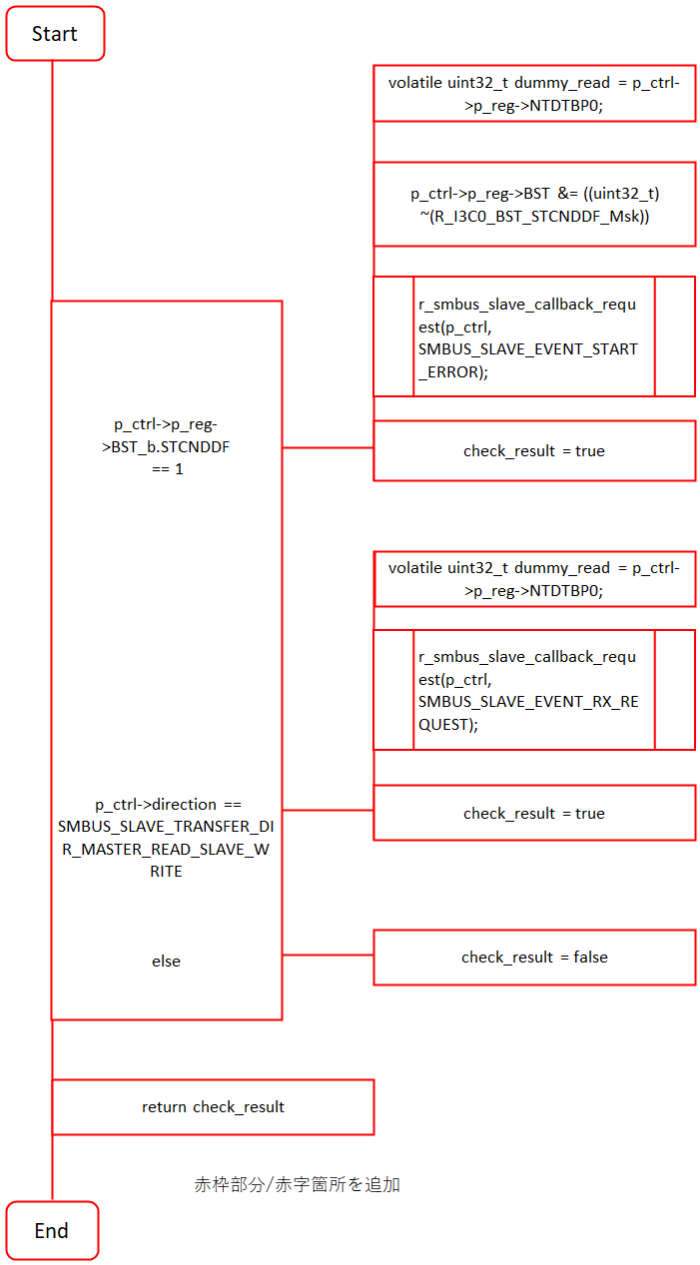
- r_smbus_open_hw_slave

一部処理を省略。青枠部分/取り消し線処理を削除。赤枠部分/赤字箇所を追加/変更。

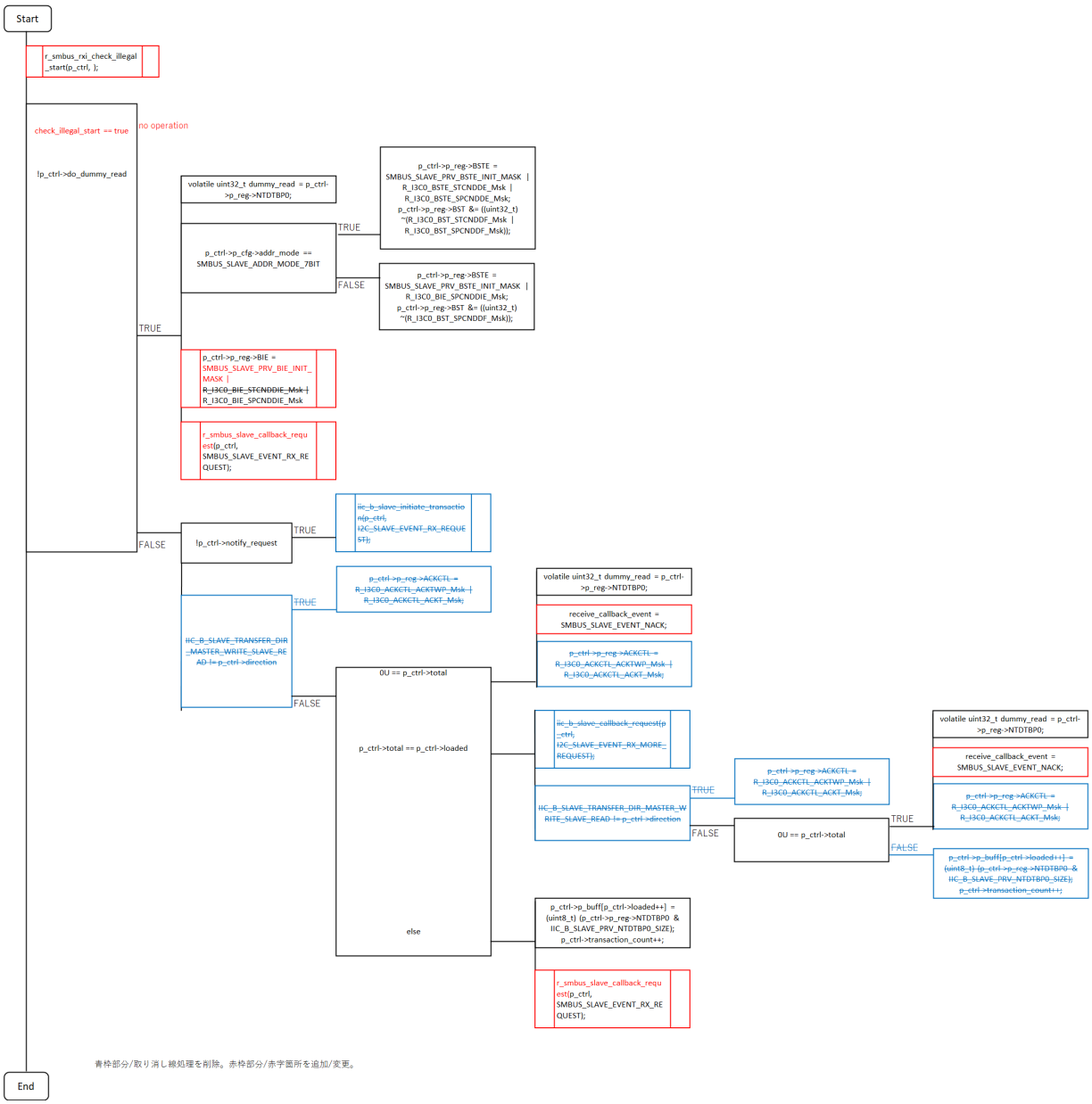
- r_smbus_slave_call_callback



- r_smbus_rxi_check_illegal_start

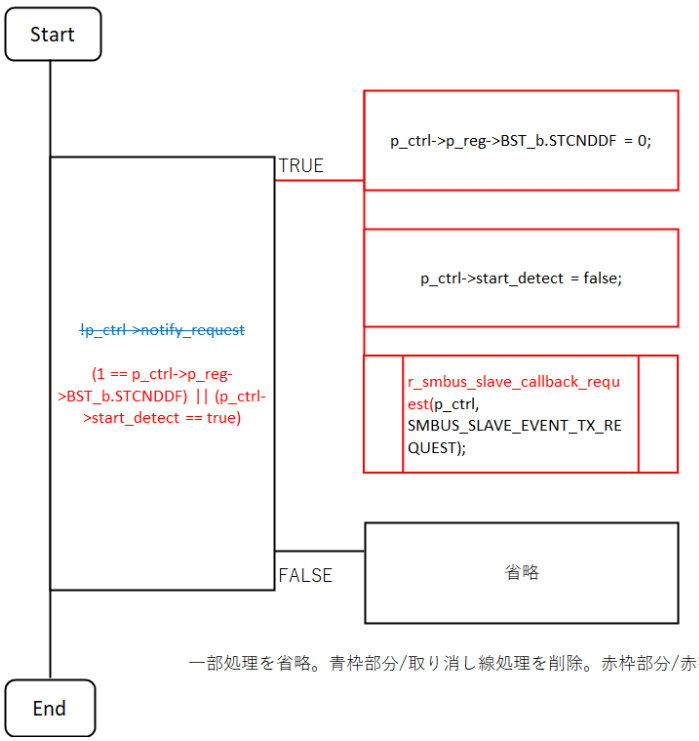


- r_smbus_rxi_slave



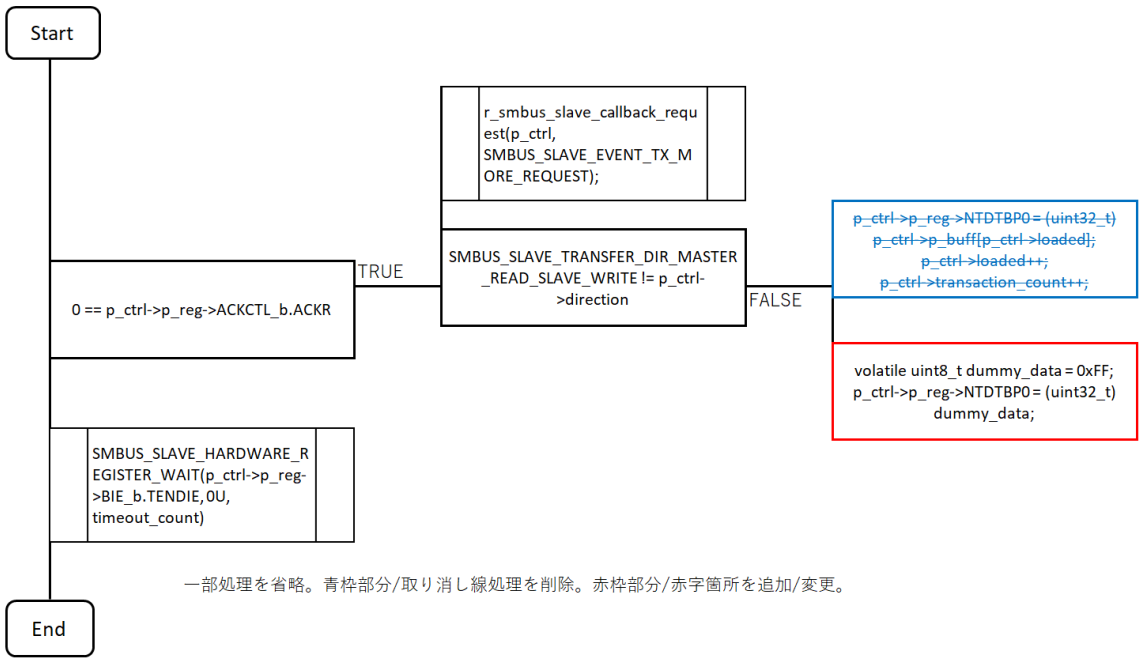
青枠部分/取り消し線処理を削除。赤枠部分/赤字箇所を追加/変更。

- r_smbus_txi_slave



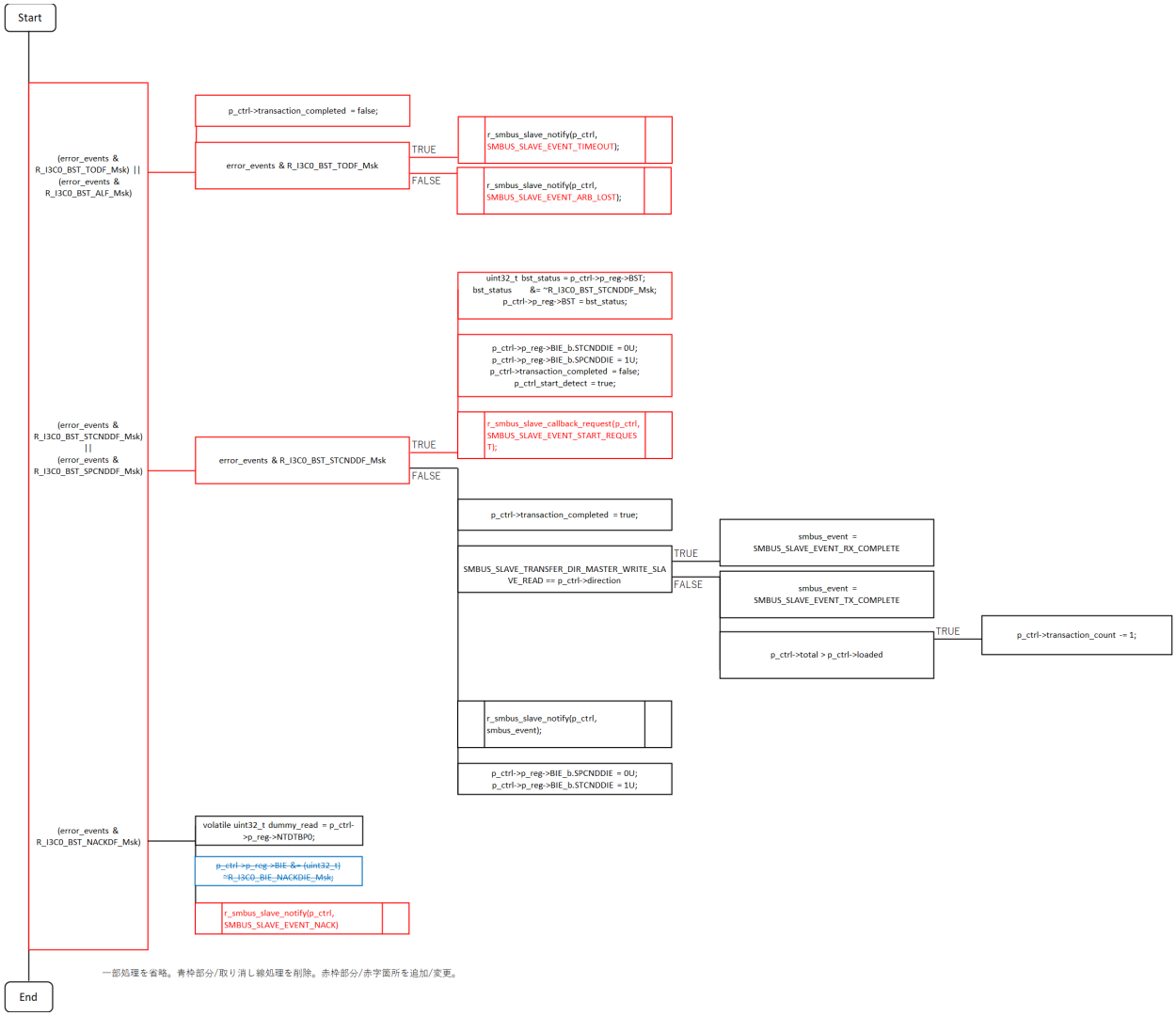
一部処理を省略。青枠部分/取り消し線処理を削除。赤枠部分/赤字箇所を追加/変更。

- r_smbus_tei_slave



一部処理を省略。青枠部分/取り消し線処理を削除。赤枠部分/赤字箇所を追加/変更。

- r_smbus_err_slave

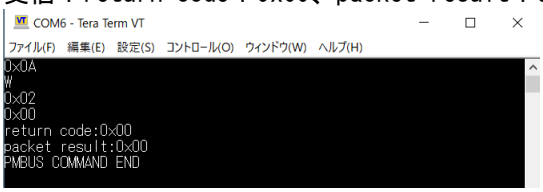


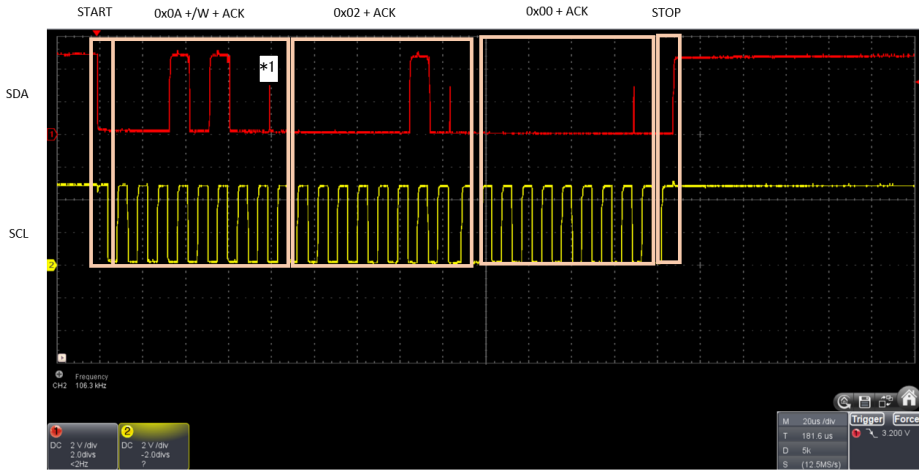
6.　PMBus コマンド送受信テスト結果

tera term を使用して送受信した通信結果の例を以下に示します。

- ON_OFF_CONFIG (write) WRITE_BYTE protocol

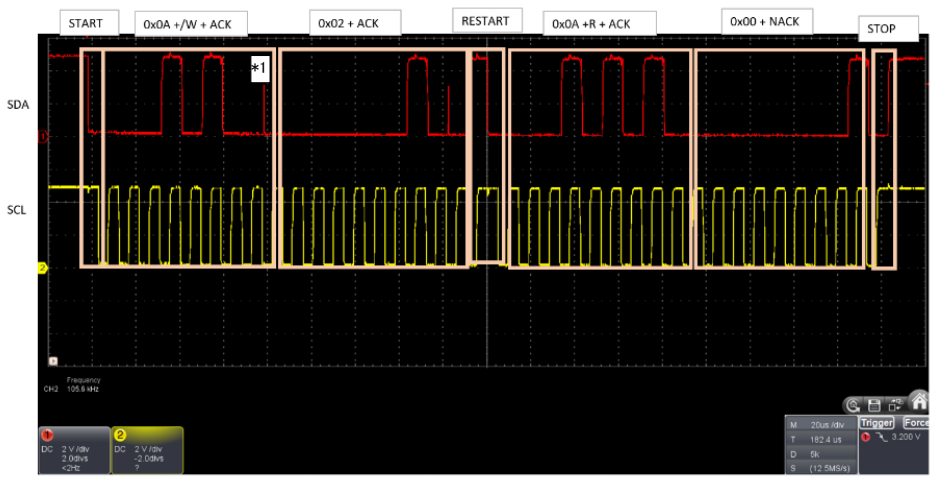
  送信：スレーブアドレス：0x0A、READ/WRITE 方向：W、コマンドコード：0x02、ライトデータ：0x00
  受信：return code：0x00、packet result：0x00





- ON_OFF_CONFIG (read) READ_BYTE protocol

  送信：スレーブアドレス：0x0A、READ/WRITE 方向：R、コマンドコード：0x02
  受信：リードデータ：0x00、return code：0x00、packet result：0x00



*1：ACK タイミングによるもので、PMBus 通信に影響はありません。

7. FAQ

Q1：本アプリケーションで確認することができない PMBUS の仕様はありますか。

A：以下の仕様は確認することができません。
- グループコマンドプロトコル
- 拡張コマンドプロトコル
- ゾーンコマンドプロトコル
- バスマスタプロトコル
- Quick Command (read)プロトコル
- Address resolution protocol(ARP)
- Control 信号の制御 (Alert 信号を含む)
- 障害管理とレポーティング機能
- マルチマスター動作
- クロックストレッチ機能
- SMBUS 信号によるサスペンドモード通知
- スレーブのバスマスター切り替え機能
- ホスト通信
- Write Protect シグナルによる内部メモリ保護機能
- General コール動作
- SMBUS 3.0.0 以降、および、PMBUS 1.3.0 以降で追加された機能。

また、以下の仕様は、ユーザアプリケーション、および、PMBUS ミドルウェアを調整頂くことで使用することができます。お客様の責任でご使用ください。
- Block Write プロトコルを使用するコマンド
- Block Read プロトコルを使用するコマンド
- Process Call プロトコルを使用するコマンド
- Block Write-Block Read Process Call プロトコルを使用するコマンド
- Quick Command (write) プロトコル
- Packet Error Check (PEC)付きの通信
- Alert Response Address (ARA)を使用した Alert 応答

Q2：PC と master の通信を行うための terminal soft はどのような設定とすればいいでしょうか。

A：以下のように設定してください。
ビットレート：115200bps
データ長：8bit
パリティ：無し
ストップビット：1 ビット
データ転送方向：LSB ファースト
改行コード：受信：LF、送信：CR+LF

Q3：本アプリケーションでは、PEC 付き通信をする場合の CRC 演算は変換テーブルを使用していますが、MCU に搭載している CRC 演算器を使用することは可能ですか。

A：はい。可能です。本アプリケーションで CRC 演算器を使用する場合は、5.1.5 PMBus Master マクロ定義一覧、および、5.2.6 PMBus_Slave マクロ定義一覧で定義している、"PMBUS_CRC8_USE_IP"を"1"に変更した上で、空関数として実装されている"r_pmbus_nwk_AddCrc8()"を MCU の CRC を使用して演算するように変更してください。なお、MCU の CRC 演算器を制御する API は、スマート・コンフィグレータ、および、FSP で生成することができます。

Q4：モータが停止しない場合は、どのような対応をすればいいですか。

A：SW1(トグル SW)を OFF することで、PMBUS コマンドの受信状態に関わらずモータを停止することが可能です。モータの回転を再開する場合は、異常が発生した原因を取り除いたうえで、SW1（トグル SW）を ON,VR1 のボリューム位置をモータが回転可能な位置にセットしたのちに、PMBUS の ON_OFF_CONFIG コマンドと OPERATION コマンドで再度モータ回転開始に必要なデータを送信してください。
なお、本アプリケーションで使用しているモータサンプルで異常を検出した場合にも停止する場合があります。詳細は各モータサンプルのアプリケーションノートもご参照ください。

Q5：スレーブ側のユーザアプリケーションをカスタマイズして評価したところ、ターミナルソフトから "packet result:0x03"（タイムアウト検出）が返却されます。
どのようなことが原因として考えられるでしょうか。

A：お客様でスレーブ側に追加した処理の負荷が高いことが原因として考えられます。
本アプリケーションでは、PMBUS（SMBUS）の仕様であるプロトコルの通信が 25ms($T_{TIMEOUT}$)以内で完了することを監視しています。READ BYTE プロトコルのように、Master 受信/Slave 送信を含むプロトコルの場合、スレーブ側のコールバック処理は Slave 送信前に行うため、コールバック処理の負荷が高いとタイムアウトが発生します。高負荷の処理を行わせたい場合は、main ループ処理で周期的に実行している pmbus_ctrl()と e_pmbus_int_event_s_t を利用し、コールバックから main ループ処理へイベント通知を行い、main 処理で高負荷処理を実行することをご検討ください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Jan.14.25	-	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策

   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置

   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号

   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1 2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

© 2025 Renesas Electronics Corporation. All rights reserved.