EC-1 シリーズ
ペリフェラルドライバマニュアル

R01AN3581JJ0120
Rev.1.20
2018.09.17

要旨

本書は、EC-1 シリーズにて使用しているドライバと、 サンプルソフトについて記載しています。

対象デバイス

EC-1

目次

1. 概説

ソフトウェア開発を速やかに進められるよう、EC-1 では各機能の使用例を示したサンプル・ソフトウェア（以降サンプルソフト）を準備しています。

本書では、EC-1 各機能のドライバの仕様、およびサンプルソフトの動作について記しています。

1.1 構成

サンプルソフトのレイヤ構成図を以下に示します。

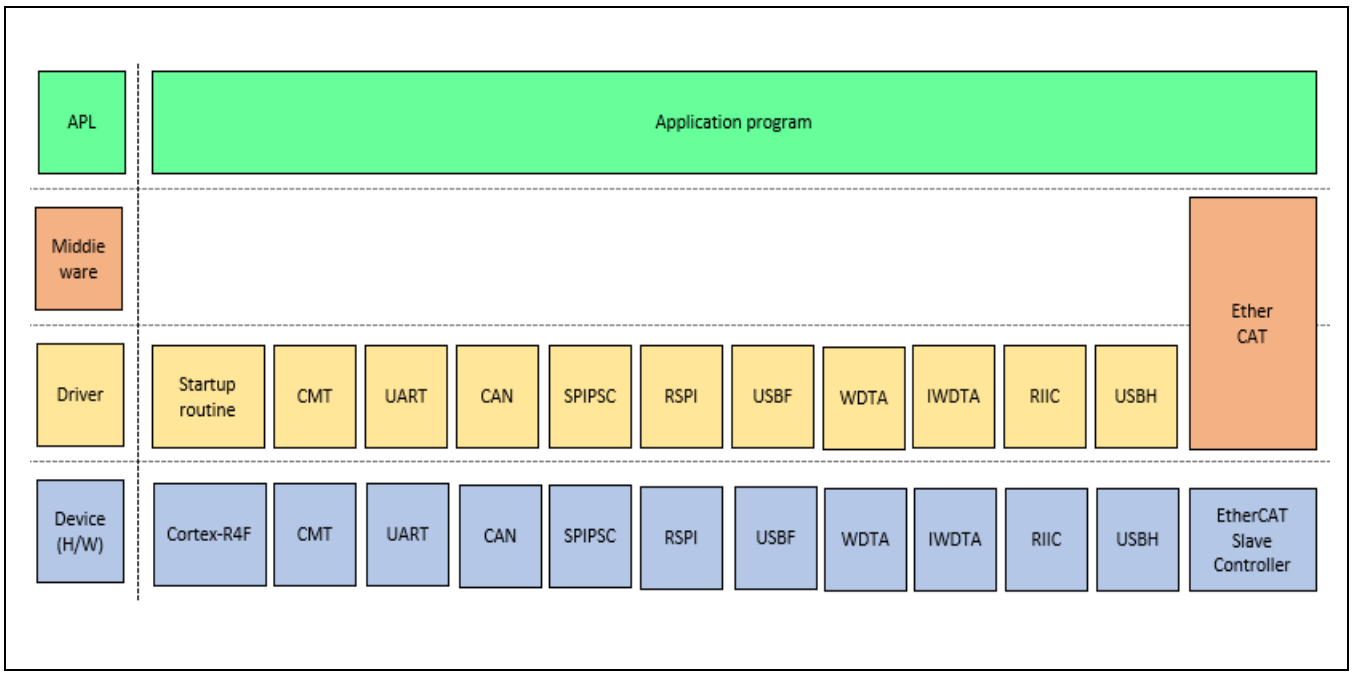


図 1-1 サンプルソフトのレイヤ構成図

1.2　開発環境

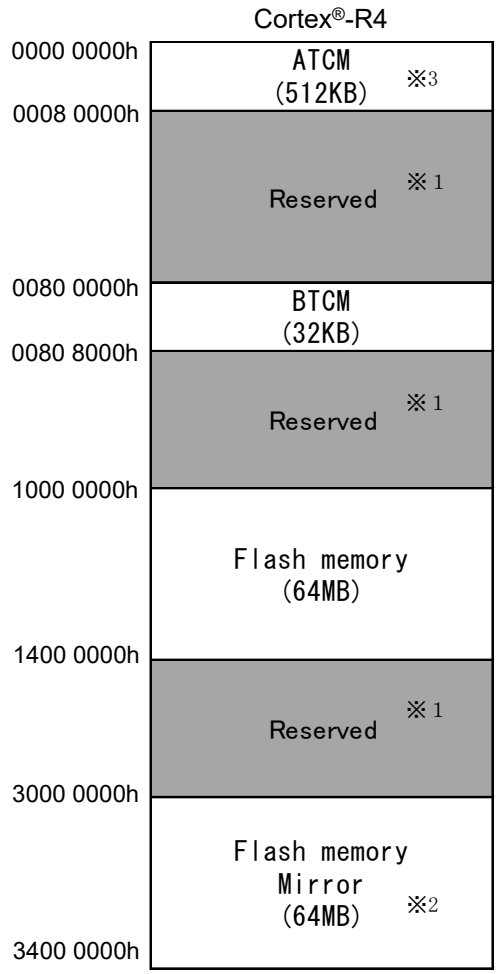
ソフトウェア開発ツールの説明を以下に示します。

本サンプルソフトでは、CMSIS の V2.10 を採用しております。詳細は CMSIS のドキュメントを参照してください。

表 1-1 ソフトウェア開発ツール一覧（ツールチェーン）

ツールチェーン	IDE	コンパイラ	デバッガ	ICE
IAR	Embedded Workbench for ARM V7.70.1 (IAR Systems)	Embedded Workbench for ARM V7.70.1 (IAR Systems)	Embedded Workbench for ARM V7.70.1 (IAR Systems)	I-jet JTAGjet-Trace-CM (IAR Systems)
GCC	Renesas e2studio	GNUARM-NONE v16.01-EABI	Renesas e2studio	Segger J-Link ARM

1.3　メモリ配置



※1　予約領域はアクセスしないでください。

※2　Cashe 対象（それ以外の領域は、MPU でキャッシュを有効にしないでください。）

※3　Cortex-R4 密結合メモリ空間

図 1-2 メモリマップ

1.4　プログラム配置例

シリアルフラッシュ ROM ブート時のプログラム配置例を以下に示します。

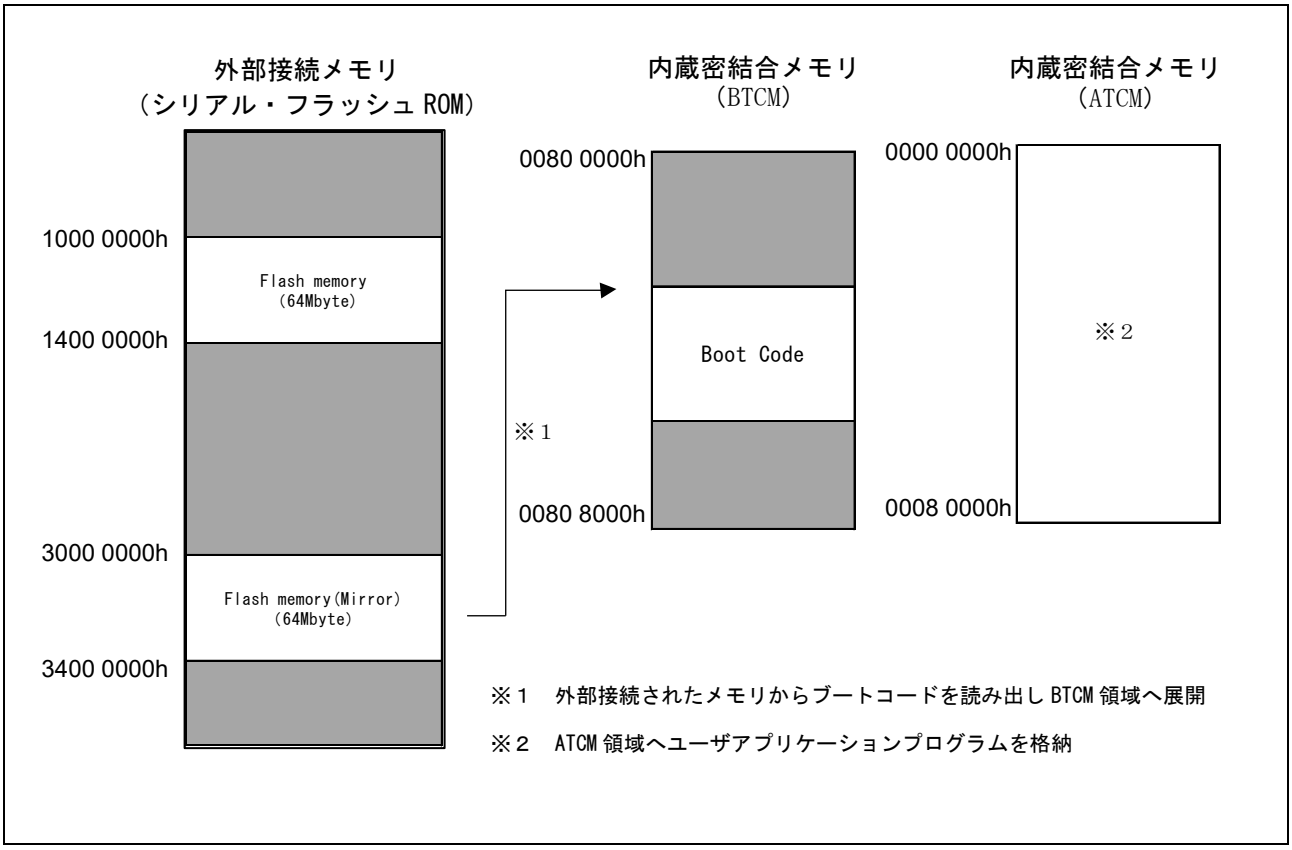


図 1-3 プログラム配置例

詳細は EC-1 ユーザーズマニュアル　ハードウェア編　3.4　動作モード　を参照して下さい。

2. ファイル構成

本章では、サンプルソフトのディレクトリ構成、ファイル構成を示します。

2.1 ディレクトリ構成

表 2-1 サンプルソフトのディレクトリ構成

ディレクトリ	内容
./	サンプルソフト格納ディレクトリ
./Include	インクルードファイル格納ディレクトリ
./Library	ライブラリ格納ディレクトリ
./Source	ソース格納ディレクトリ

2.2 ./Include：インクルードファイル

以下にインクルードファイルのファイル構成を示します。

表 2-2 インクルードファイルディレクトリのファイル構成

ディレクトリ	ファイル	内容
board/	board.h	EC-1 CPUボードの設定
can/	can_reg.h	CAN設定
	r_can_api.h	CANサンプルプログラム用APIのプロトタイプ宣言
	r_can_config.h	CANコンフィグレーション
cmt/	r_cmt.h	CMTドライバのプロトタイプ宣言
eth/	r_ether.h	ETHERドライバのプロトタイプ宣言
rspi/	r_rspi.h	RSPIドライバのプロトタイプ宣言
scifa/	r_scifa_api.h	CANサンプルプログラム用APIのプロトタイプ宣言
	r_scifa_uart.h	SCIFA_UARTドライバのプロトタイプ宣言
	sio_char.h	char型制御用シリアルI/ Oの設定
sflash/	r_sflash.h	シリアル・フラッシュROMドライバのプロトタイプ宣言
usbf/	r_usb_basic_config.h	USBf BASICユーザ定義
	r_usb_basic_if.h	USBf BASICインタフェース定義
	r_usb_cdefusbip.h	IP設定
	r_usb_pcdc_config.h	PCDCユーザ定義
	r_usb_pcdc_if.h	PCDCインタフェース定義
wdta/	r_wdt_config.h	WDTAユーザ定義
	r_wdt_if.h	WDTAインタフェース定義
iwdta/	r_iwdt_config.h	IWDTAユーザ定義
	r_iwdt_if.h	IWDTAインタフェース定義
riic/	r_riic_ec1_config.h	RIICユーザ定義
	r_riic_ec1_if.h	RIICインタフェース定義
usbh/	r_usb_basic_config.h	USBh BASICユーザ定義
	r_usb_basic_if.h	USBh BASICインタフェース定義
	r_usb_hatapi_define.h	USB共通の外部ヘッダー
	r_usb_hmsc_config.h	HMSCユーザ定義
	r_usb_hmsc_if.h	HMSCインタフェース定義

ディレクトリ	ファイル	内容
./	iodefine.h	レジスタ定義
	platform.h	MCUボード設定
	r_atcm_init.h	ATCMアクセスウェイト設定APIヘッダ
	r_bsc.h	BSC設定APIヘッダ
	r_cpg.h	CPG設定APIヘッダ
	r_ecm.h	ECM設定APIヘッダ
	r_icu_init.h	割り込みコントローラユニットの初期化
	r_mpc.h	MPC設定APIヘッダ
	r_port.h	ポート設定APIヘッダ
	r_reset.h	EC-1リセットAPI及び低電力APIヘッダ
	r_system.h	システム設定

2.3 ./Library：ライブラリ

このディレクトリにファイルはありません。

2.4 ./Source：ソース

以下にソースディレクトリの構成を示します。

表 2-3 ソースディレクトリの構成

ディレクトリ	内容
Driver	ドライバ関連
Project	サンプルアプリケーション
Templates	スタートアップ・ファイル等

2.4.1 ./Source/Driver：ドライバ関連

以下にドライバのソースファイルの構成を示します。

表 2-4 ドライバ関連ディレクトリのファイル構成

ディレクトリ	ファイル	内容
can/	r_can_api.c	CANドライバ
cmt/	cmt_userdef.c	CMTユーザー定義
	r_cmt.c	CMTドライバ
ether/	r_ether.c	ETHERドライバ
rspi/	r_rspi.c	RSPIドライバ
	r_rspi_user.c	RSPIユーザー定義
scifa_uart/	r_scifa_api.c	CANサンプルプログラム用API
	scifa_uart.c	SCIFA_UARTドライバ
	scifa_uart_userdef.c	SCIFA_UARTユーザー定義
	siochar.c	入出力API及びSCIFA初期化
	siorw.c	入出力制御API
sflash/	r_sflash.c	シリアル・フラッシュROMドライバ
usbf/pcdc/	r_usb_pcdc_api.c	USB PCDC API
	r_usb_pcdc_driver.c	USB PCDC ドライバ
	r_usb_pcdc_local.h	USB PCDC ヘッダ
usbf/pcdc/ utilities/	CDC_Demo.inf	CDCデバイスドライバ設定

ディレクトリ	ファイル	内容
usbf/basic/	r_usb_cdataio.c	USBペリフェラル lowレベルI/Oコード
	r_usb_cextern.h	USB共通外部ヘッダ
	r_usb_cintfifo.c	USBペリフェラル 割り込みコード
	r_usb_cinthandler_usbip0.c	USBペリフェラル 割り込みハンドラコード
	r_usb_clibusbip.c	USBペリフェラル lowレベルライブラリ
	r_usb_creg_abs.c	USBレジスタアクセス呼び出し
	r_usb_creg_access.c	USB IP レジスタアクセスコード
	r_usb_cusb_bitdefine.h	USB BIT定義
	r_usb_dma.c	DMA設定
	r_usb_dmac.h	DMA設定ヘッダ
	r_usb_pcontrolrw.c	USBペリフェラルコントロール転送APIコード
	r_usb_pdriver.c	USBペリフェラルドライバコード
	r_usb_pdriverapi.c	USBペリフェラルドライバAPIコード（PCD）
	r_usb_pintfifo.c	USBペリフェラルFIFOアクセスコード
	r_usb_preg_abs.c	USBペリフェラルレジスタアクセス呼び出し
	r_usb_preg_access.c	USBペリフェラルレジスタアクセスコード
	r_usb_psignal.c	USBペリフェラルシグナルコントロールコード
	r_usb_pstdfunction.c	USBペリフェラル標準機能コード
	r_usb_pstdrequest.c	USBペリフェラル標準要求コード
	r_usb_reg_access.h	USBペリフェラルレジスタアクセス機能
	ec1_mcu.c	EC-1用設定処理
wdta/	r_wdt.c	WDTAドライバ
iwdta/	r_iwdt.c	IWDTAドライバ
riic/	r_riic_ec1.c	RIICドライバ
	r_riic_ec1_private.h	RIICドライバヘッダ
usbh/basic/	r_usb_basic_local.h	USB BASICヘッダ
	r_usb_cscheduler.c	USBホストスケジューラ
	r_usb_hDriver.c	USBホストコントロールドライバ
	r_usb_hdriverapi.c	USBホストコントロールドライバAPI
	r_usb_hEhciDefUsr.h	EHCIユーザー定義ヘッダ
	r_usb_hEhciExtern.h	EHCI Externヘッダ
	r_usb_hEhciMain.c	EHCIメイン処理
	r_usb_hEhciMemory.c	EHCIメモリ制御コード
	r_usb_hEhciTransfer.c	EHCI転送コード
	r_usb_hEhciTypedef.h	EHCIタイプ定義ヘッダ
	r_usb_hHci.c	HCI共通コード
	r_usb_hHci.h	HCIヘッダ
	r_usb_hHciLocal.h	HCIローカル共通ヘッダ
	r_usb_hhubsys.c	USBホストハブシステムコード
	r_usb_hManager.c	USBホストコントロールマネージャ
	r_usb_hOhciDefUsr.h	OHCIユーザ定義ヘッダ
	r_usb_hOhciExtern.h	OHCI Externヘッダ
	r_usb_hOhciMain.c	OHCIメイン処理
	r_usb_hOhciMemory.c	OHCIメモリ制御コード
	r_usb_hOhciTransfer.c	OHCI転送コード
	r_usb_hOhciTypedef.h	OHCIタイプ定義ヘッダ

2.4.2 ./Source/Project：サンプルアプリケーション

以下にサンプルアプリケーションの構成を示します。

表 2-5 サンプルアプリケーションディレクトリのファイル構成

ディレクトリ	ファイル	内容
can_sample/	main.c	CANサンプルメイン処理ソースファイル
can_sample/IAR/	EC_1_can_serial_boot.eww	IARプロジェクト・ファイル
	EC_1_can_serial_boot.ewd	IARプロジェクト関連ファイル
	EC_1_can_serial_boot.ewp	IARプロジェクト関連ファイル
	EC_1_can_ram_debug.eww	IARプロジェクト・ファイル
	EC_1_can_ ram_debug.ewd	IARプロジェクト関連ファイル
	EC_1_can_ ram_debug.ewp	IARプロジェクト関連ファイル
can_sample/GCC/	EC-1_e2sws_serial_boot.bat	プロジェクト起動バッチファイル
	serial_boot_sample.zip	プロジェクトアーカイブファイル
rspi_sample/	main.c	RSPIサンプルメイン処理ソースファイル
	board_RenesaEva.c	ボード設定ファイル
rspi_sample/IAR/	EC-1_rspi_serial_boot.eww	IARプロジェクト・ファイル
	EC-1_rspi_serial_boot.ewd	IARプロジェクト関連ファイル
	EC-1_rspi_serial_boot.ewp	IARプロジェクト関連ファイル
	EC-1_rspi_ram_debug.eww	IARプロジェクト・ファイル
	EC-1_rspi_ram_debug.ewd	IARプロジェクト関連ファイル
	EC-1_rspi_ram_debug.ewp	IARプロジェクト関連ファイル
rspi_sample/GCC/	EC-1_e2sws_serial_boot.bat	プロジェクト起動バッチファイル
	serial_boot_sample.zip	プロジェクトアーカイブファイル
sflash_writer_sample/	main.c	SFLASH_WRITERサンプルメイン処理ソースファイル
	flash_writer.c	FLASH_WRITERソースファイル
	flash_writer.h	FLASH_WRITERヘッダファイル
sflash_writer_sample/IAR/	EC_1_sflash_writer_serial_boot.eww	IARプロジェクト・ファイル
	EC_1_sflash_writer_serial_boot.ewd	IARプロジェクト関連ファイル
	EC_1_sflash_writer_serial_boot.ewp	IARプロジェクト関連ファイル
	EC_1_sflash_writer_ram_debug.eww	IARプロジェクト・ファイル
	EC_1_sflash_writer_ram_debug.ewd	IARプロジェクト関連ファイル
	EC_1_sflash_writer_ram_debug.ewp	IARプロジェクト関連ファイル
sflash_writer_sample/GCC/	EC-1_e2sws_serial_boot.bat	プロジェクト起動バッチファイル
	serial_boot_sample.zip	プロジェクトアーカイブファイル
usbf_sample/	main.c	USBfサンプルメイン処理ソースファイル
	r_usb_pcdc_apl.c	PCDC API
	r_usb_pcdc_descriptor.c	PCDC 設定用データ
usbf_sample/IAR/	EC_1_usbf_serial_boot.eww	IARプロジェクト・ファイル
	EC_1_usbf_serial_boot.ewd	IARプロジェクト関連ファイル
	EC_1_usbf_serial_boot.ewp	IARプロジェクト関連ファイル
	EC_1_usbf_ram_debug.eww	IARプロジェクト・ファイル
	EC_1_usbf_ram_debug.ewd	IARプロジェクト関連ファイル
	EC_1_usbf_ram_debug.ewp	IARプロジェクト関連ファイル

ディレクトリ	ファイル	内容
usbf_sample/GCC/	EC-1_e2sws_serial_boot.bat	プロジェクト起動バッチファイル
	serial_boot_sample.zip	プロジェクトアーカイブファイル
wdta_sample/	main.c	WDTAサンプルメイン処理ソースファイル
wdta_sample/IAR/	EC_1_wdta_serial_boot.eww	IARプロジェクト・ファイル
	EC_1_wdta_serial_boot.ewd	IARプロジェクト関連ファイル
	EC_1_wdta_serial_boot.ewp	IARプロジェクト関連ファイル
	loader_init_sflash.c	EC-1ペリフェラル設定初期化(サンプル用)
wdta_sample/GCC/	EC-1_e2sws_serial_boot.bat	プロジェクト起動バッチファイル
	serial_boot_sample.zip	プロジェクトアーカイブファイル
	loader_init_sflash.c	EC-1ペリフェラル設定初期化(サンプル用)
iwdta_sample/	main.c	IWDTAサンプルメイン処理ソースファイル
iwdta_sample/IAR/	EC_1_iwdta_serial_boot.eww	IARプロジェクト・ファイル
	EC_1_iwdta_serial_boot.ewd	IARプロジェクト関連ファイル
	EC_1_iwdta_serial_boot.ewp	IARプロジェクト関連ファイル
	loader_init_sflash.c	EC-1ペリフェラル設定初期化(サンプル用)
iwdta_sample/GCC/	EC-1_e2sws_serial_boot.bat	プロジェクト起動バッチファイル
	serial_boot_sample.zip	プロジェクトアーカイブファイル
	loader_init_sflash.c	EC-1ペリフェラル設定初期化(サンプル用)
riic_sample/	main.c	RIICサンプルメイン処理ソースファイル
riic_sample/IAR/	EC_1_riic_serial_boot.eww	IARプロジェクト・ファイル
	EC_1_riic_serial_boot.ewd	IARプロジェクト関連ファイル
	EC_1_riic_serial_boot.ewp	IARプロジェクト関連ファイル
	EC_1_riic_ram_debug.eww	IARプロジェクト・ファイル
	EC_1_riic_ram_debug.ewd	IARプロジェクト関連ファイル
	EC_1_riic_ram_debug.ewp	IARプロジェクト関連ファイル
riic_sample/GCC/	EC-1_e2sws_serial_boot.bat	プロジェクト起動バッチファイル
	serial_boot_sample.zip	プロジェクトアーカイブファイル
usbh_sample/	main.c	USBh サンプルメイン処理ソースファイル
	r_usb_hmsc_api.c	USBh MSCアプリケーションコード
	r_usb_hmsc_api.h	USBh MSCアプリケションコードヘッダ
	r_usb_main.c	USBh メイン処理
usbh_sample/GCC/	EC-1_e2sws_serial_boot.bat	プロジェクト起動バッチファイル
	serial_boot_sample.zip	プロジェクトアーカイブファイル
	EC-1_init_serial_boot.gsi	USBh用リンカスクリプトファイル
usbh_samole/IAR/	EC_1_usbh_serial_boot.eww	IARプロジェクト・ファイル
	EC_1_usbh_serial_boot.ewd	IARプロジェクト関連ファイル
	EC_1_usbh_serial_boot.ewp	IARプロジェクト関連ファイル
	EC_1_usbh_ram_debug.eww	IARプロジェクト・ファイル
	EC_1_usbh_ram_debug.ewd	IARプロジェクト関連ファイル
	EC_1_usbh_ram_debug.ewp	IARプロジェクト関連ファイル
	EC-1_init_ram_debug.icf	マッピングファイル(サンプル用)
	EC-1_init_serial_boot.icf	マッピングファイル(サンプル用)

2.4.3 ./Source/Templates：スタートアップファイル等

以下にスタートアップ・ファイル等のソースファイルの構成を示します。

表 2-6 スタートアップ関連ディレクトリのファイル構成

ディレクトリ	ファイル	内容
Templates/	exit.c	終了シーケンス
	r_atcm_init.c	ATCMアクセスウェイト設定API
	r_cpg.c	CPG設定API
	r_ecm.c	ECM設定API
	r_icu_init.c	EC-1各機器設定初期化
	r_mpc.c	MPC設定API
	r_reset.c	EC-1リセットAPI及び低電力API
Templates/IAR/	loader_init.asm	EC-1用割り込みサービスルーチン
	vector.asm	ベクタテーブル設定
Templates/IAR/serial_boot/	bus_init_serial_boot.c	バス設定初期化
	EC-1_init_serial_boot.icf	マッピングファイル
	EC1_init_boot.mac	初期化マクロファイル
	loader_init_sflash.c	EC-1ペリフェラル設定初期化
	loader_param_serial_boot.c	SPIブートモード用パラメータ設定
Templates/IAR/ram_debug/	EC-1_init_ram_debug.icf	マッピングファイル
	EC1_init_ram_debug.mac	初期化マクロファイル
	loader_init_ram.c	EC-1ペリフェラル設定初期化
Templates/GCC/	loader_init.asm	EC-1用割り込みサービスルーチン
	vector.asm	ベクタテーブル設定
	usrheap.c	heapユーザ設定
Templates/GCC/ram_debug/	loader_init_ram.c	EC-1ペリフェラル設定初期化
Templates/GCC/serial_boot/	bus_init_serial_boot.c	バス設定初期化
	loader_init_sflash.c	EC-1ペリフェラル設定初期化
	loader_param_serial_boot.c	SPIブートモード用パラメータ設定
	EC-1_init_serial_boot.gsi	リンカスクリプトファイル

3. ドライバ

ドライバの構造体説明及び関数説明を示します。

3.1 構造体/共用体/列挙型一覧

ドライバにて定義している構造体/共用体/列挙型を示します。

3.1.1 CAN 制御

表 3-1 can_vector_t構造体

メンバ名	内容
uint8_t BYTE	フレーム情報
uint8_t CANGE :1	CAN global error interrupt
uint8_t CANIE0:1	CAN0 error interrupt
uint8_t CANIE1:1	CAN1 error interrupt
uint8_t CANRFI:1	CAN reception FIFO Interrupt
uint8_t CANFIR0:1	CAN0 transmission-and-reception FIFO interrupt
uint8_t CANTI0:1	CAN0 transmission interrupt
uint8_t CANFIR1:1	CAN1 transmission-and-reception FIFO interrupt
uint8_t CANTI1:1	CAN1 transmission interrupt

表 3-2 can_callback_t構造体

メンバ名	内容
void (*pintr_ge)(void);	Pointer to user callback function.
void (*pintr_ie0)(void);	Pointer to user callback function.
void (*pintr_ie1)(void);	Pointer to user callback function.
void (*pintr_rfi)(void);	Pointer to user callback function.
void (*pintr_fir0)(void);	Pointer to user callback function.
void (*pintr_ti0)(void);	Pointer to user callback function.
void (*pintr_fir1)(void);	Pointer to user callback function.
void (*pintr_ti1)(void);	Pointer to user callback function.

表 3-3 can_handle_t構造体

メンバ名	内容
bool	ch_opened
can_callback_t	can_callback

表 3-4 can_rx_rule_t構造体__受信ルール・テーブル構造体

メンバ名	内容
uint32_t buf_type;	Type of the buffer to use used Transmission:CAN_TX_BUFFER,CAN_TX_FIFO Reception:CAN_RX_BUFFER,CAN_RX_RX_FIFO,CAN_RX_FIFO
uint32_t rule_page;	Reception rule page number
uint32_t rule_table;	Reception rule table number
uint32_t rule_id;	Message ID
uint32_t rule_type;	Message type (data frame/remote frame)
uint32_t rule_format;	Message format (Standard ID/Extended ID)
uint32_t rule_label;	Message label
uint32_t rule_dlc_check;	DLC checking
uint32_t rule_mask;	Mask

表 3-5 udata_t構造体__4バイト長データ共用体

メンバ名	内容
uint32_t LONG;	-
uint8_t BYTE(4);	-

表 3-6 CAN_tx_message_t構造体__送信メッセージデータ構造体

メンバ名	内容
uint32_t id;	Message ID
uint32_t type;	0:Data frame / 1:Remote frame
uint32_t format;	0:Standard ID / 1:Extended ID
uint32_t length;	Message data length
udata_t data_h;	Message data
udata_t data_l;	Message data
uint32_t history_label;	Label
uint32_t buf_type;	type of the buffer to be used (buffer or FIFO)

表 3-7 CAN_rx_message_t構造体__受信メッセージデータ構造体

メンバ名	内容
uint32_t id;	Message ID
uint32_t format;	0:Standard ID / 1:Extended ID
uint32_t type;	0:Data frame / 1: Remote frame
uint16_t timestamp;	Timestamp data
uint16_t label;	Label information
uint32_t length;	Message length
udata_t data_h;	Message data
udata_t data_l;	Message data

表 3-8 CAN_rx_message_t構造体__グローバル・エラー発生要因カウント構造体

メンバ名	内容
uint32_t total_num;	Total count of the global errors
uint32_t dlc_error_num;	DLC error count
uint32_t fifo_message_lost_num;	FIFO message loss count
uint32_t history_overflow_num;	Transmission history overflow count

表 3-9 ch_err_source_t構造体__チャネル・エラー発生要因カウント構造体

メンバ名	内容
uint32_t total_num;	Total count of the channel errors
uint32_t bus_error_num;	Bus error count
uint32_t error_warning_num;	Error warning count
uint32_t error_passive_num;	Error passive count
uint32_t bus_off_start_num;	Bus-off start count
uint32_t bus_off_return_num;	Bus-off return count
uint32_t overload_num;	Overload count
uint32_t bus_lock_num;	Bus lock-up count
uint32_t arbitration_lost_num;	Arbitration loss count
uint32_t staff_error_num;	Staff error count
uint32_t form_error_num;	Form error count
uint32_t ack_error_num;	ACK error count
uint32_t crc_error_num;	CRC error count
uint32_t recessive_bit_error_num;	Recessive bit error count
uint32_t dominant_bit_error_num;	Dominant bit error count
uint32_t ack_delimiter_error_num;	ACK delimiter error count

表 3-10 ch_tx_source_t構造体__送信割り込み発生要因カウント構造体

メンバ名	内容
uint32_t total_num;	Total count of the transmission interrupts
uint32_t intr_sorce;	Interrupt source
uint32_t tx_buf_end_num;	Count of successful transmissions from the transmission buffer
uint32_t tx_fifo_end_num;	Count of successful transmissions from the FIFO buffer
uint32_t tx_abort_num;	Transmission abortion count
uint32_t tx_queue_num;	Transmission queue count
uint32_t tx_history_num;	Transmission history data count

表 3-11 CAN_intr_source_t構造体__割り込み要因情報構造体

メンバ名	内容
uint32_t rx_fifo_num;	Reception FIFO interrupt count
uint32_t ch_fifo_receive_num;	Transmission-and-reception FIFO receive interrupt count
ch_tx_source_t tx_source;	Transmission interrupt source
gl_err_source_t gl_err;	Global error interrupt source
ch_err_source_t ch_err;	Channel error interrupt source

表 3-12 can_used_buffer_t構造体__使用バッファ情報構造体

メンバ名	内容
uint32_t use_tx_buf_no;	送信バッファの番号
uint32_t use_rx_buf_no;	受信バッファの番号
uint32_t use_rx_fifo_no;	受信 FIFO バッファの番号
uint32 use_fifo_txmode_no;	送受信 FIFO バッファ(送信モード)の番号
uint32_t use_fifo_rxmode_no;	送受信 FIFO バッファ(受信モード)の番号
uint32_t use_fifo_link_buf_no;	FIFO バッファ・リンク先バッファの番号

表 3-13 can_tx_intr_sts_t構造体__送信割込み要求のステータス情報構造体

メンバ名		内容
uint8_t BYTE		フレーム情報
	uint8_t TMTRF:2;	送信バッファ送信結果
	uint8_t CFTXIF:1;	送受信 FIFO 送信割込み要求
	uint8_t TXQIF:1;	送信キュー割込み要求
	uint8_t THLIF:1;	送信履歴割込み要求
	uint8_t :3;	-

表 3-14 can_tx_history_t構造体__送信履歴情報構造体

メンバ名	内容
uint32_t buf_type;	Buffer type
uint32_t buf_no;	Buffer number
uint32_t label;	Label data

3.1.2　RSPI 制御

表 3-15 rspi_state_t構造体

メンバ名	内容		
uint8_t b_ovrf;	Overrun error flag	0: No error	1: Overrun error
uint8_t b_idlnf;	RSPI idle flag	0: Idle state	1: Not Idle state
uint8_t b_modf;	Mode fault error flag	0: No error	1: Mode fault error
uint8_t b_perf;	Parity error flag	0: No error	1: Parity error
uint8_t b_invalid;	invalid packet flag	0: No error	1: Invalid packet data
uint8_t b_rx_end;	Receive end flag	0: Not end	1: Recieve end
uint8_t b_tx_end;	Transmit end flag	0: Not end	1: Recieve end
uint8_t	not use		

3.1.3　SCIFA_UART 制御

表 3-16 scifa_callback_t構造体

メンバ名	内容
void (*pintr_ski)(void);	pointer to user callback function

表 3-17 scifa_handle_t構造体

メンバ名	内容
scifa_callback_t　scifa_callback;	scifa_callback

表 3-18 scifa_vector_para_t構造体

メンバ名	内容
uint16_t　vector_no;	vector no
uint32_t　priority;	priority
void (__irq __arm * pcallback_isr)(void);	pointer to user callback interrupt function

3.1.4　シリアル・フラッシュ ROM 制御

表 3-19 crr_sflash_infot構造体

メンバ名	内容
uint32_t　id;	Serial Flash ROM ID
uint32_t　device_size;	Serial Flash ROM Device size
uint32_t　erace_size;	Serial Flash ROM Erase size
uint32_t　page_size;	Serial Flash ROM Program size

3.1.5　USB ファンクション制御

表 3-20 USB_REQUEST_t構造体

USB_REQUEST_t は、スタンダードリクエスト以外の USB リクエストを格納するための構造体です。
USB_PCDREG_t 構造体の ctrltrans に登録されたコールバック関数の引数に使用されます。

メンバ名	内容
uint16_t　ReqType;	bmRequestType[D7-D0]の値が入ります (USBREQ レジスタ BMREQUESTTYPE の値が入ります) 使用時は USB_BMREQUESTTYPE(0x00FFu)でマスクして下さい
uint16_t　ReqTypeType;	bmRequestType の Type[D6-D5]の値が入ります (USBREQ レジスタ BMREQUESTTYPE の b6-5 値が入ります) 使用時は USB_BMREQUESTTYPETYPE(0x0060u)でマスクして下さい
uint16_t　ReqTypeRecip;	bmRequestType の Recipient[D4-D0]の値が入ります。 (USBREQ レジスタ BMREQUESTTYPE の b4-0 値が入ります。) 使用時は USB_BMREQUESTTYPERECIP(0x001Fu)でマスクして下さい
uint16_t　ReqRequest;	bRequest の値が入ります (USBREQ レジスタ BREQUEST の値が入ります) 使用時は USB_BREQUEST(0xFF00u)でマスクして下さい
uint16_t　ReqValue;	wValue の値が入ります (USBVAL レジスタの値が入ります)
uint16_t　ReqIndex;	wIndex の値が入ります (USBINDEX レジスタの値が入ります)
uint16_t　ReqLength;	wLength の値が入ります (USBLENG レジスタの値が入ります)

表 3-21 USB_UTR_t構造体

USB_UTR_t 構造体は、コントロール転送を除くデータ転送で使用する構造体です。

メンバ名	内容
uint16_t　keyword;	データ転送を行うパイプ番号を登録してください
USB_UTR_CB_t　complete;	データ転送終了時に起動する関数を登録してください
void　*tranadr;	データ転送用バッファのアドレスを登録してください
uint32_t　tranlen;	データ転送サイズを指定してください 転送サイズは、データ転送用バッファサイズ以下としてください
uint16_t　status;	ペリフェラルコントロールドライバ（PCD）によってデータ転送の結果が格納されます 「7.13.2 転送結果の通知」を参照してください
uint16_t　pipectr;	ペリフェラルコントロールドライバ（PCD）によって PIPECTR レジスタの値が格納されます

表 3-22 USB_PCDREG_t構造体

USB_PCDREG_t は、ペリフェラルデバイスクラスドライバ(PDCD)の情報を登録するための構造体です。
USB_PCDREG_t に登録したコールバック関数は、デバイスステートの変化などで実行されます。

メンバ名	内容
uint16_t **pipetbl;	パイプ情報テーブルのアドレス
uint8_t *devicetbl;	Device Descriptor テーブルのアドレス
uint8_t *qualitbl;	Device Qualifier Descriptor テーブルのアドレス
uint8_t **configtbl;	Configuration Descriptor テーブルのアドレス
uint8_t **othertbl;	Other speed Descriptor テーブルのアドレス
uint8_t **stringtbl;	String Descriptor テーブルのアドレス
USB_CB_t devdetach;	デフォルトステート遷移時に起動する関数を登録してください 登録した関数は USB リセット検出時に呼び出されます
USB_CB_t devconfig;	コンフィガード状態遷移時に起動する関数を登録してください 登録した関数は SET_CONFIGURATION リクエストのステータスステージで呼び出されます
USB_CB_t devdetach;	デタッチ状態遷移時に起動する関数を登録してください 登録した関数はデタッチ検出時に呼び出されます
USB_CB_t devsuspend;	サスペンド状態遷移時に起動する関数を登録してください 登録した関数はサスペンド検出時に呼び出されます
USB_CB_t devresume;	レジューム状態遷移時に起動する関数を登録してください 登録した関数はレジューム検出時に呼び出されます
USB_CB_t interface;	インタフェース変更時に起動する関数を登録してください 登録した関数は SET_INTERFACE リクエストのステータスステージで呼び出されます
USB_CB_TRN_t ctrltrans;	標準コマンド以外の USB リクエストを受信したときに起動する関数を登録してください

表 3-23 USB_SCI_SerialState_t構造体

メンバ名		内容
unsigned long WORD		-
	uint16_t bRxCarrier:1;	Data Carrier Detect：回線にキャリア検出
	uint16_t bTxCarrier:1;	Data Set Ready：回線が接続されて通信可能
	uint16_t bBreak:1;	ブレーク信号検出
	uint16_t bRingSignal:1;	着信（Ring signal）を感知した
	uint16_t bFraming:1;	フレーミングエラー検出
	uint16_t bParity:1;	パリティエラー検出
	uint16_t bOverRun:1;	オーバーランエラー検出
	uint16_t rsv:9;	予約

3.1.6 WDTA 制御

表 3-24 wdt_err_t列挙型

メンバ名	内容
WDT_SUCCESS=0	-
WDT_ERR_OPEN_IGNORED	The module has already been Open()ed
WDT_ERR_INVALID_ARG	Argument is not valid for parameter
WDT_ERR_NULL_PTR	Received null pointer or missing required argument
WDT_ERR_NOT_OPENED	Open function has not yet been called

表 3-25 wdt_ch_t列挙型

メンバ名	内容
WDT_CHANNEL_0=0	ch0
WDT_CHANNEL_1	ch1
WDT_CHANNEL_MAX	-

表 3-26 wdt_timeout_t列挙型

メンバ名	内容
WDT_TIMEOUT_1024 =0x0000u	1024 (cycles)
WDT_TIMEOUT_4096 =0x0001u	4096 (cycles)
WDT_TIMEOUT_8192 =0x0002u	8192 (cycles)
WDT_TIMEOUT_16384=0x0003u	16,384 (cycles)
WDT_NUM_TIMEOUTS	-

表 3-27 wdt_clock_div_t列挙型

メンバ名	内容
WDT_CLOCK_DIV_4 =0x0010u	WDTCLK/4
WDT_CLOCK_DIV_64 =0x0040u	WDTCLK/64
WDT_CLOCK_DIV_128 =0x00F0u	WDTCLK/128
WDT_CLOCK_DIV_512 =0x0060u	WDTCLK/512
WDT_CLOCK_DIV_2048=0x0070u	WDTCLK/2048
WDT_CLOCK_DIV_8192=0x0080u	WDTCLK/8192

表 3-28 wdt_window_end_t列挙型

メンバ名	内容
WDT_WINDOW_END_75=0x0000u	75%
WDT_WINDOW_END_50=0x0100u	50%
WDT_WINDOW_END_25=0x0200u	25%
WDT_WINDOW_END_0 =0x0300u	0% (window end position is not specified)

表 3-29 wdt_window_start_t列挙型

メンバ名	内容
WDT_WINDOW_START_25 =0x0000u	25%
WDT_WINDOW_START_50 =0x1000u	50%
WDT_WINDOW_START_75 =0x2000u	75%
WDT_WINDOW_START_100=0x3000u	100% (window start position is not specified)

表 3-30 wdt_timeout_control_t列挙型

メンバ名	内容
WDT_ERROR_ENABLE =0x00u	Error output is enabled
WDT_ERROR_DISABLE=0x80u	Error output is disabled

表 3-31 wdt_config_t構造体

メンバ名	内容
wdt_timeout_t timeout	Time-out period
wdt_clock_div_t wdtclk_div	WDT clock division ratio
wdt_window_start_t window_start	Window start position
wdt_window_end_t window_end	Window end position
wdt_timeout_control_t timeout_control	ERROR output when time-out

表 3-32 wdt_cmd_t列挙型

メンバ名	内容
WDT_CMD_GET_STATUS	Get WDT status
WDT_CMD_REFRESH_COUNTING	Refresh the counter
WDT_CMD_NO_ACTION	-

3.1.7 IWDTA 制御

表 3-33 iwdt_err_t列挙型

メンバ名	内容
IWDT_SUCCESS=0	-
IWDT_ERR_OPEN_IGNORED	The module has already been Open()ed
IWDT_ERR_INVALID_ARG	Argument is not valid for parameter
IWDT_ERR_NULL_PTR	Received null pointer or missing required argument
IWDT_ERR_NOT_OPENED	Open function has not yet been called

表 3-34 iwdt_timeout_t列挙型

メンバ名	内容
IWDT_TIMEOUT_1024 =0x0000u	1024 (cycles)
IWDT_TIMEOUT_4096 =0x0001u	4096 (cycles)
IWDT_TIMEOUT_8192 =0x0002u	8192 (cycles)
IWDT_TIMEOUT_16384=0x0003u	16,384 (cycles)
IWDT_NUM_TIMEOUTS	-

表 3-35 iwdt_clock_div_t列挙型

メンバ名	内容
IWDT_CLOCK_DIV_1 =0x0000u	WDTCLK/1
IWDT_CLOCK_DIV_16 =0x0020u	WDTCLK/16
IWDT_CLOCK_DIV_32 =0x0030u	WDTCLK/32
IWDT_CLOCK_DIV_64 =0x0040u	WDTCLK/64
IWDT_CLOCK_DIV_128=0x00F0u	WDTCLK/128
IWDT_CLOCK_DIV_256=0x0050u	WDTCLK/256

表 3-36 iwdt_window_end_t列挙型

メンバ名	内容
IWDT_WINDOW_END_75=0x0000u	75%
IWDT_WINDOW_END_50=0x0100u	50%
IWDT_WINDOW_END_25=0x0200u	25%
IWDT_WINDOW_END_0 =0x0300u	0% (window end position is not specified)

表 3-37 iwdt_window_start_t列挙型

メンバ名	内容
IWDT_WINDOW_START_25 =0x0000u	25%
IWDT_WINDOW_START_50 =0x1000u	50%
IWDT_WINDOW_START_75 =0x2000u	75%
IWDT_WINDOW_START_100=0x3000u	100% (window start position is not specified)

表 3-38 iwdt_timeout_control_t列挙型

メンバ名	内容
IWDT_ERROR_ENABLE =0x00u	Error output is enabled
IWDT_ERROR_DISABLE=0x80u	Error output is disabled

表 3-39 iwdt_config_t構造体

メンバ名	内容
iwdt_timeout_t timeout	Time-out period
iwdt_clock_div_t wdtclk_div	IWDT clock division ratio
iwdt_window_start_t window_start	Window start position
iwdt_window_end_t window_end	Window end position
iwdt_timeout_control_t timeout_control	ERROR output when time-out

表 3-40 iwdt_cmd_t列挙型

メンバ名	内容
IWDT_CMD_GET_STATUS	Get IWDT status
IWDT_CMD_REFRESH_COUNTING	Refresh the counter
IWDT_CMD_NO_ACTION	-

3.1.8 RIIC 制御

表 3-41 riic_return_t列挙型

メンバ名	内容
RIIC_SUCCESS = 0U	Successful operation
RIIC_ERR_LOCK_FUNC	Lock has already been acquired by another task.
RIIC_ERR_INVALID_CHAN	None existent channel number
RIIC_ERR_INVALID_ARG	Parameter error
RIIC_ERR_NO_INIT	Uninitialized state
RIIC_ERR_BUS_BUSY	Channel is on communication.
RIIC_ERR_AL	Arbitration lost error
RIIC_ERR_OTHER	Other error

表 3-42 riic_info_t構造体

メンバ名		内容
uint8_t	rsv2	reserved
uint8_t	rsv1	reserved
riic_ch_dev_status_t	dev_sts	Device status flag
uint8_t	ch_no	Channel No.
riic_callback	callbackfunc	Callback function
uint32_t	cnt2nd	2nd Data Counter
uint32_t	cnt1nd	1st Data Counter
uint8_t *	p_data2nd	Pointer for 2nd Data buffer
uint8_t *	p_data1st	Pointer for 1st Data buffer
uint8_t *	p_slv_adr	Pointer for Slave address buffer

表 3-43 riic_mcu_status_t構造体

メンバ名		内容
uint32_t LONG		-
uint32_t	rsv:12	reserve
uint32_t	AAS2:1	Slave2 address detection flag
uint32_t	AAS1:1	Slave1 address detection flag
uint32_t	AAS0:1	Slave0 address detection flag
uint32_t	GCA:1	Generalcall address detection flag
uint32_t	DID:1	DeviceID address detection flag
uint32_t	HOA:1	Host address detection flag
uint32_t	MST:1	Master mode / Slave mode flag
uint32_t	TMO:1	Time out flag
uint32_t	AL:1	Arbitration lost detection flag
uint32_t	SP:1	Stop condition detection flag
uint32_t	ST:1	Start condition detection flag
uint32_t	RBUF:1	Receive buffer status flag
uint32_t	SBUF:1	Send buffer status flag
uint32_t	SCLO:1	SCL pin output control status
uint32_t	SDAO:1	SDA pin output control status
uint32_t	SCLI:1	SCL pin level
uint32_t	SDAI:1	SDA pin level
uint32_t	NACK:1	NACK detection flag
uint32_t	TRS:1	Send mode / Receive mode flag
uint32_t	BSY:1	Bus status flag

3.1.9　USB ホスト制御

表 3-44 USB_HCDREG_t構造体

USB_HCDREG_t は、HDCD の情報を登録するための構造体です。
USB_HCDREG_t に登録したコールバック関数は、デバイスステートの変化などで実行されます。

メンバ名	内容
uint16_t　rootport	接続されているポート番号が登録されます。
uint16_t　devaddr	デバイスアドレスが登録されます。
uint16_t　devstate	デバイスの接続状態が登録されます。
uint16_t　ifclass	HDCD のインタフェースクラスコードを登録してください。
uint16_t　*tpl	HDCD が動作するターゲットペリフェラルリストを登録してください。 「11.8.6 ターゲットペリフェラルリスト（TPL）」を参照してください。
USB_CB_CHECK_t　classcheck	HDCD チェック時に起動する関数を登録してください。 TPL が一致した場合に呼び出します。
USB_CB_INFO_t　devconfig	Configured 状態遷移時に起動する関数を登録してください。 SET_CONFIGURATION リクエストが終了した場合に呼び出します。
USB_CB_INFO_t　devdetach	デタッチ状態遷移時に起動する関数を登録してください。
USB_CB_INFO_t　devsuspend	サスペンド状態遷移時に起動する関数を登録してください。
USB_CB_INFO_t　devresume	レジューム状態遷移時に起動する関数を登録してください。

表 3-45 USB_SETUP_t構造体

USB リクエストのデータ構造体です。

メンバ名	内容
uint16_t　type	bRequest[b15-b8], bmRequestType[b7-b0] を設定してください。
uint16_t　value	wValue を設定してください。
uint16_t　index	wIndex を設定してください。
uint16_t　length	wLength を設定してください。
uint16_t　devaddr	デバイスアドレス

表 3-46 USB_UTR_t構造体

USB 通信およびタスクメッセージ用の構造体です。
R_usb_hstd_TransferStart()の引数に設定することで、接続されたデバイスと USB 通信ができます。

メンバ名	内容
uint16_t msginfo	USB-BASIC-F/W のタスクが使用するメッセージ情報を設定してください。 USB 通信をする場合は設定不要です。
uint16_t keyword	通信するパイプ番号を設定してください。
uint16_t result	USB 通信結果が格納されます。 「7.13.2 転送結果の通知」を参照してください。
USB_UTR_CB_t complete	USB 通信が完了した場合に実行したい関数アドレスを設定してください。 「11.8.8 コールバック関数」を参照してください。
void *tranadr	USB 通信バッファアドレスを設定してください。 ・受信または ControlRead 転送 受信データを格納するバッファのアドレスを指定します。 ・送信または ControlWrite 転送 送信データが格納してあるバッファアドレスを指定します。 ・NoDataControl 転送 指定されても無視します。
uint32_t tranlen	USB 通信データ長を設定してください。 ・受信または ControlRead 転送 受信データ長を指定します。USB 通信終了後に、実際に受信したデータ長を引いた値に更新されます。 ・送信または ControlWrite 転送 送信データ長を指定します。 ・NoDataControl 転送 "0"を指定してください。
USB_SETUP_t *setup	コントロール転送時の Setup パケット情報を設定してください。 「表 3-45 USB_SETUP_t 構造体」および「11.8.9 USB 通信」の(6) コントロール転送を参照してください。

3.2　定数/エラーコード一覧

ドライバで用意している定数/エラーコードの一覧を示します。

3.2.1　CAN 制御

表 3-47 CANドライバで使用する定数

定数名	設定値	内容
CAN_NUM	2	CAN モジュールのチャネル数
CAN_CH_0	0	チャネル 0（CAN0）
CAN_CH_1	1	チャネル 1（CAN1）
CH_BUFFER_MAX	16	各チャネルで使用可能な送信バッファ数
CH_FIFO_BUFFER_MAX	3	各チャネルで使用可能な送受信 FIFO バッファ数
DATA_MAX	8	1 回の送信可能なメッセージデータ数
CAN_TX_BUFFER	0	状態フラグ（送信バッファ使用）
CAN_TX_FIFO	1	状態フラグ（送受信 FIFO(送信モード)バッファ使用）
CAN_TX_HISTORY	2	状態フラグ（送信履歴）
CAN_TX_QUEUE	3	状態フラグ（送信キュー）
CAN_RX_BUFFER	0	状態フラグ（受信バッファ使用）
CAN_RX_RX_FIFO	1	状態フラグ（受信 FIFO バッファ使用）
CAN_RX_FIFO	2	状態フラグ（送受信 FIFO(受信モード)バッファ使用）
CAN_MODULE_ON	0	ストップ状態解除
CAN_MODULE_OFF	1	ストップ状態に遷移
CAN_STANDARD	0	標準 ID
CAN_EXTENDED	1	拡張 ID
CAN_DATA_FRAME	0	データ・フレーム
CAN_REMOTE_FRAME	1	リモート・フレーム
CAN_RULE_PAGE_MAX	24	受信ルールページ最大数
CAN_RULE_TABLE_MAX	16	受信ルール・テーブル最大数
CAN_RX_FIFO_BUFFER_MAX	8	受信 FIFO バッファ最大数
CAN_RX_BUFFER_MAX	32	受信バッファ最大数
CAN_RULE_NUM_MAX	64	受信ルール最大数
CAN_RX_MODE	0	受信モード
CAN_TX_MODE	1	送信モード
CAN_GATEWAY_MODE	2	ゲートウェイモード
CAN_FIFO_MSG_0	0	送受信 FIFO バッファ段数（0 メッセージ）
CAN_FIFO_MSG_4	1	送受信 FIFO バッファ段数（4 メッセージ）
CAN_FIFO_MSG_8	2	送受信 FIFO バッファ段数（8 メッセージ）
CAN_FIFO_MSG_16	3	送受信 FIFO バッファ段数（16 メッセージ）
CAN_FIFO_MSG_32	4	送受信 FIFO バッファ段数（32 メッセージ）
CAN_FIFO_MSG_48	5	送受信 FIFO バッファ段数（48 メッセージ）
CAN_FIFO_MSG_64	6	送受信 FIFO バッファ段数（64 メッセージ）
GL_MODE_STOP	0	グローバル・ストップ・モード
GL_MODE_RESET	1	グローバル・リセット・モード
GL_MODE_TEST	2	グローバル・テスト・モード
GL_MODE_OPE	3	グローバル・動作・モード
CAN_GL_OPE	0	グローバル・動作・モードへ遷移
CAN_GL_RESET	1	グローバル・リセット・モードへ遷移
CAN_GL_TEST	2	グローバル・テスト・モードへ遷移

定数名	設定値	内容
CH_MODE_STOP	0	チャネル・ストップ・モード
CH_MODE_RESET	1	チャネル・リセット・モード
CH_MODE_WAIT	2	チャネル・待機・モード
CH_MODE_COMM	3	チャネル・通信・モード
CAN_CH_COMM	0	チャネル・通信・モードへ遷移
CAN_CH_RESET	1	チャネル・リセット・モードへ遷移
CAN_CH_WAIT	2	チャネル・待機・モードへ遷移
GL_TEST_RAMTEST	0	RAM テスト
GL_TEST_COMMTEST	1	チャネル間通信テスト
CH_TEST_STANDARD	0	標準テストモード
CH_TEST_LISTENONLY	1	リッスンオンリモード
CH_TEST_SELF0	2	セルフテストモード 0(外部ループバックモード)
CH_TEST_SELF1	3	セルフテストモード 1(内部ループバックモード)
CANCLKA_CLK	24000000u	CAN クロック（24MHz）
CANCLKB_CLK	25000000u	CAN クロック（25MHz）
CAN_INTR_DISABLE	0	割込み禁止
CAN_INTR_ENABLE	1	割込み許可
CAN_IR_PRIORITY_262_CANERR_GL	3	優先順位（CAN グローバル・エラー）
CAN_IR_PRIORITY_263_CANE RR_CH0	4	優先順位（CAN0 エラー）
CAN_IR_PRIORITY_264_CANE RR_CH1	4	優先順位（CAN1 エラー）
CAN_IR_PRIORITY_104_CANR FI	5	優先順位（CAN 受信 FIFO）
CAN_IR_PRIORITY_105_CANFI R0	5	優先順位（CAN0 送受信 FIFO 受信完了）
CAN_IR_PRIORITY_106_CANTI 0	5	優先順位（CAN0 送信）
CAN_IR_PRIORITY_107_CANFI R1	5	優先順位（CAN1 送受信 FIFO 受信完了）
CAN_IR_PRIORITY_108_CANTI 1	5	優先順位（CAN1 送信）
CAN_HVA_WRITE_DATA	0u	HVA 書き込みデータ
CAN_OK	0u	戻り値：正常
CAN_EMPTY	1u	戻り値：バッファエンプティ
CAN_NG	0xFFFFFFFFu	戻り値：エラー発生
CAN_INTR_TX_END	1	チャネル送信割込み要因：送信完了
CAN_INTR_ABORT_END	2	チャネル送信割込み要因：アボート送信完了
CAN_INTR_FIFO_REQ	3	チャネル送信割込み要因：送受信 FIFO（送信モード）送信
CAN_INTR_QUEUE_REQ	4	チャネル送信割込み要因：送信キュー要求
CAN_INTR_HISTORY_REQ	5	チャネル送信割込み要因：送信履歴要求
CAN_INTR_FIFO_EMPTY	1	受信 FIFO バッファエンプティ
CAN_INTR_FIFO_FULL	2	受信 FIFO バッファフル
CAN_INTR_FIFO_LOST	3	受信 FIFO バッファメッセージ・ロスト
CAN_INTR_FIFO_TX_MESSAG E	4	送受信 FIFO 送信割込み要求
CAN_INTR_FIFO_RX_MESSAG E	5	送受信 FIFO 受信割込み要求
CAN_BUS_ERR	1	エラー・フラグ（バス・エラー）
CAN_ERR_WARNING	2	エラー・フラグ（エラー・ワーニング）
CAN_ERR_PASSIVE	3	エラー・フラグ（エラー・パッシブ）
CAN_BUS_OFF_START	4	エラー・フラグ（バスオフ開始）
CAN_BUS_OFF_RETURN	5	エラー・フラグ（バスオフ復帰）
CAN_OVER_LOAD	6	エラー・フラグ（オーバロード）
CAN_BUS_LOCK	7	エラー・フラグ（チャネルバスロック）

定数名	設定値	内容
CAN_ARBITRATION_LOST	8	エラー・フラグ（アービトレーション・ロスト）
CAN_STAFF_ERR	9	エラー・フラグ（スタッフ・エラー）
CAN_FORM_ERR	10	エラー・フラグ（フォーム・エラー）
CAN_ACK_ERR	11	エラー・フラグ（ACK エラー）
CAN_CRC_ERR	12	エラー・フラグ（CRC エラー）
CAN_RECESSIVE_BIT_ERR	13	エラー・フラグ（レセシブビットエラー）
CAN_DOMINANT_BIT_ERR	14	エラー・フラグ（ドミナントビットエラー）
CAN_ACK_DELIMITER_ERR	15	エラー・フラグ（ACK デリミタエラー）
CAN_DLC_ERR	1	エラー・フラグ（DLC エラー）
CAN_FIFO_MSG_LOST_ERR	2	エラー・フラグ（FIFO メッセージ・ロスト）
CAN_HISTORY_OVERFLOW_E RR	3	エラー・フラグ（送信履歴バッファ・オーバフロー）
CAN0_CRXD0_P30_VAL	0x10	MPC：CAN0 CRXD0 設定値（未使用）
CAN0_CRXD0_PC6_VAL	0x10	MPC：CAN0 CRXD0 設定値
CAN0_CTXD0_P60_VAL	0x10	MPC：CAN0 CTXD0 設定値（未使用）
CAN0_CTXD0_P67_VAL	0x10	MPC：CAN0 CTXD0 設定値
CAN1_CRXD1_PC3_VAL	0x10	MPC：CAN1 CRXD1 設定値（未使用）
CAN1_CRXD1_PC7_VAL	0x10	MPC：CAN1 CRXD1 設定値
CAN1_CTXD1_P61_VAL	0x10	MPC：CAN1 CTXD1 設定値（未使用）
CAN1_CTXD1_P66_VAL	0x10	MPC：CAN1 CTXD1 設定値
CAN1_CTXD1_PB3_VAL	0x10	MPC：CAN1 CTXD1 設定値（未使用）
CAN_GL_STATUS_BIT	0x00000007u	RSCAN0GSTS　レジスタマスクビット
CAN_CH_STATUS_BIT	0x00000007u	RSCAN0CmSTS　レジスタマスクビット
GCFG_REG_INIT	0x00000013u	RSCAN0GCFG レジスタ初期値
TMIEC0_REG_DISABLE_LOW	0x0000FFFFu	RSCAN0TMIEC0 レジスタ TMIEp(p=15～0)マスクビット
TMIEC0_REG_DISABLE_HIGH	0xFFFF0000u	RSCAN0TMIEC0 レジスタ TMIEp(p=31～16)マスクビット
CAN_CH_STOP_MODE	0x00000004u	RSCAN0CmCTR レジスタチャネル・ストップ・モード
CAN_REL_CH_STOP_MODE	0xFFFFFFFBu	RSCAN0CmCTR レジスタチャネル・ストップ・モード解除
TIME_QUANTUM_MIN	8	（*1）ビット・タイミング設定範囲 SS + TSEG1 + TSEG2 = 8 ~ 25Tq の範囲で設定
TIME_QUANTUM_MAX	25	（*1）ビット・タイミング設定範囲 SS + TSEG1 + TSEG2 = 8 ~ 25Tq の範囲で設定
SAMPLE_POINT	0.666666667	（*1）サンプルポイント(%) 本サンプルでは、2/3 の値をサンプルポイントにしています。
FIFO_UPDATE	0x000000FFu	FIFO バッファポインタ制御設定値

（*1）EC-1 ユーザーズマニュアルハードウェア編の 27.9.1.2 ビットタイミングの設定 章を参照して下さい。

3.2.2　CMT 制御

表 3-48 CMTドライバで使用する定数

定数名	設定値	内容
CMT_SUCCESS	0	関数戻り値のための定数です。 関数実行が成功したことを意味します。
CMT_ERR	-1	関数戻り値のための定数です。 関数実行が失敗したことを意味します。
CMT_CH_TOTAL	4	CMT チャネル数
CMT_CH_0	0	CMT チャネル 0 を指定するための定数です。
CMT_CH_1	1	CMT チャネル 1 を指定するための定数です。
CMT_CH_2	2	CMT チャネル 2 を指定するための定数です。
CMT_CH_3	3	CMT チャネル 3 を指定するための定数です。
CMT_CKS_DIVISION_8	0	CMCNTn カウンタに入力するクロックを PCLKD/8 に設定するための定数です。
CMT_CKS_DIVISION_32	1	CMCNTn カウンタに入力するクロックを PCLKD/32 に設定するための定数です。
CMT_CKS_DIVISION_128	2	CMCNTn カウンタに入力するクロックを PCLKD/128 に設定するための定数です。
CMT_CKS_DIVISION_512	3	CMCNTn カウンタに入力するクロックを PCLKD/512 に設定するための定数です。
CMT_MODE_PERIODEC	0	CMT チャネルの動作モード: 周期イベント設定
CMT_MODE_ONESHOT	1	CMT チャネルの動作モード: ワンショットイベント設定
PCLKD_Hz	75000000	CMT クロック設定

3.2.3　ETHER 制御

表 3-49 ETHERドライバで使用する定数

定数名	設定値	内容
ETHER_SUCCESS	0	関数処理成功
ETHER_ERR	-1	関数処理失敗

3.2.4　RSPI 制御

表 3-50 RSPIドライバで使用する定数

定数名	設定値	内容
_RSPI_MSMODE_MASTER	0	RSPI チャネルの動作モード：マスターモード
_RSPI_MSMODE_SLAVE	1	RSPI チャネルの動作モード：スレーブモード
_RSPI_DIVISOR	0x05U	マスターモード時のビットレート設定
MD_STATUSBASE	0x00U	ステータスリストベースアドレス
MD_OK	MD_STATUSBASE + 0x00U	レジスタ設定完了
MD_SPT	MD_STATUSBASE + 0x01U	I2C 通信停止
MD_NACK	MD_STATUSBASE + 0x02U	I2C NOACK
MD_BUSY1	MD_STATUSBASE + 0x03U	ビジー1 ~ 2
MD_BUSY2	MD_STATUSBASE + 0x04U	-
MD_ERRORBASE	0x80U	エラーリストベースアドレス
MD_ERROR	MD_ERRORBASE + 0x00U	エラー
MD_ARGERROR	MD_ERRORBASE + 0x01U	エラー引数の入力エラー
MD_ERROR1	MD_ERRORBASE + 0x02U	エラー1 ~ 4
MD_ERROR2	MD_ERRORBASE + 0x03U	-
MD_ERROR3	MD_ERRORBASE + 0x04U	-
MD_ERROR4	MD_ERRORBASE + 0x05U	-
MD_ERROR5	MD_ERRORBASE + 0x06U	-
RSPI_STS_XFRCMP	0x60	送受信フラグステータス
RSPI_STS_ERR	0x1F	エラーフラグステータス
_RSPI_MODE_SPI	0x00U	SPCR レジスタ　RSPI モード選択ビット 0 : SPI 動作(4 線式)
_RSPI_MODE_CLOCK_SYNCHRONOUS	0x01U	SPCR レジスタ　RSPI モード選択ビット 1 : クロック同期式動作 SPI 動作(3 線式)
_RSPI_FULL_DUPLEX_SYNCHRONOUS	0x00U	SPCR レジスタ　通信動作モード選択ビット 0 : 全二重同期式シリアル通信
_RSPI_TRANSMIT_ONLY	0x02U	SPCR レジスタ　通信動作モード選択ビット 1 : 送信動作のみのシリアル通信
_RSPI_MODE_FAULT_DETECT_DISABLED	0x00U	SPCR レジスタ　モードフォルトエラー検出許可ビット 0 : モードフォルトエラー検出を禁止
_RSPI_MODE_FAULT_DETECT_ENABLED	0x04U	SPCR レジスタ　モードフォルトエラー検出許可ビット 1 : モードフォルトエラー検出を許可
_RSPI_SLAVE_MODE	0x00U	SPCR レジスタ　マスタ/スレーブモード選択ビット 0 : スレーブモード
_RSPI_MASTER_MODE	0x08U	SPCR レジスタ　マスタ/スレーブモード選択ビット 1 : マスタモード
_RSPI_ERROR_INTERRUPT_DISABLED	0x00U	SPCR レジスタ　エラー割り込み許可ビット 0 : エラー割り込み要求の発生を禁止
_RSPI_ERROR_INTERRUPT_ENABLED	0x10U	SPCR レジスタ　エラー割り込み許可ビット 1 : エラー割り込み要求の発生を許可
_RSPI_TRANSMIT_INTERRUPT_DISABLED	0x00U	SPCR レジスタ　送信バッファエンプティ割り込み許可ビット　0 : 送信バッファエンプティ割り込み要求の発生を禁止
_RSPI_TRANSMIT_INTERRUPT_ENABLED	0x20U	SPCR レジスタ　送信バッファエンプティ割り込み許可ビット　1 : 送信バッファエンプティ割り込み要求の発生を許可

定数名	設定値	内容
_RSPI_FUNCTION_DISABLED	0x00U	SPCR レジスタ　RSPI 機能許可ビット 0：RSPI 機能は無効
_RSPI_FUNCTION_ENABLED	0x40U	SPCR レジスタ　RSPI 機能許可ビット 1：RSPI 機能は有効
_RSPI_RECEIVE_INTERRUPT_DISABLED	0x00U	SPCR レジスタ　受信バッファフル割り込み許可ビット 0：受信バッファフル割り込み要求の発生を禁止
_RSPI_RECEIVE_INTERRUPT_ENABLED	0x80U	SPCR レジスタ　受信バッファフル割り込み許可ビット 0：受信バッファフル割り込み要求の発生を禁止
_RSPI_SSL0_POLARITY_LOW	0x00U	SSLP レジスタ　SSL0 信号極性設定ビット 0：SSLy0 信号はアクティブ Low
_RSPI_SSL0_POLARITY_HIGH	0x01U	SSLP レジスタ　SSL0 信号極性設定ビット 1：SSLy0 信号はアクティブ High
_RSPI_SSL1_POLARITY_LOW	0x00U	SSLP レジスタ　SSL1 信号極性設定ビット 0：SSLy1 信号はアクティブ Low
_RSPI_SSL1_POLARITY_HIGH	0x02U	SSLP レジスタ　SSL1 信号極性設定ビット 1：SSLy1 信号はアクティブ High
_RSPI_SSL2_POLARITY_LOW	0x00U	SSLP レジスタ　SSL2 信号極性設定ビット 0：SSLy2 信号はアクティブ Low
_RSPI_SSL2_POLARITY_HIGH	0x04U	SSLP レジスタ　SSL2 信号極性設定ビット 1：SSLy2 信号はアクティブ High
_RSPI_SSL3_POLARITY_LOW	0x00U	SSLP レジスタ　SSL3 信号極性設定ビット 0：SSLy3 信号はアクティブ Low
_RSPI_SSL3_POLARITY_HIGH	0x08U	SSLP レジスタ　SSL3 信号極性設定ビット 1：SSLy3 信号はアクティブ High
_RSPI_LOOPBACK_DISABLED	0x00U	SPPCR レジスタ　RSPI ループバックビット 0：通常モード
_RSPI_LOOPBACK_ENABLED	0x01U	SPPCR レジスタ　RSPI ループバックビット 1：ループバックモード（データを反転して送信）
_RSPI_LOOPBACK2_DISABLED	0x00U	SPPCR レジスタ　RSPI ループバック 2 ビット 0：通常モード
_RSPI_LOOPBACK2_ENABLED	0x02U	SPPCR レジスタ　RSPI ループバック 2 ビット 1：ループバックモード（データを反転せずに送信）
_RSPI_OUTPUT_PIN_CMOS	0x00U	SPPCR レジスタ　出力端子モードビット 0：CMOS 出力
_RSPI_OUTPUT_PIN_OPEN_DRAIN	0x04U	SPPCR レジスタ　出力端子モードビット 1：オープンドレイン出力
_RSPI_MOSI_LEVEL_LOW	0x00U	SPPCR レジスタ　MOSI アイドル固定値ビット 0：MOSI アイドル時の MOSIy 端子の出力値は Low
_RSPI_MOSI_LEVEL_HIGH	0x10U	SPPCR レジスタ　MOSI アイドル固定値ビット 1：MOSI アイドル時の MOSIy 端子の出力値は High
_RSPI_MOSI_FIXING_PREV_TRANSFER	0x00U	SPPCR レジスタ　MOSI アイドル値固定許可ビット 0：MOSI 出力値は前回転送の最終データ
_RSPI_MOSI_FIXING_MOIFV_BIT	0x20U	SPPCR レジスタ　MOSI アイドル値固定許可ビット 1：MOSI 出力値は MOIFV ビットの設定値
_RSPI_SEQUENCE_LENGTH_1 ~ 7	0x00U ~ 0x07U	SPSCR レジスタ　RSPI シーケンス長設定ビット 設定値にて参照する SPCMD0～7 レジスタの参照順を変更
_RSPI_FRAMES_1 ~ 4	0x00U ~ 0x03U	SPDCR レジスタ　フレーム数設定ビット フレーム数の指定
_RSPI_READ_SPDR_RX_BUFFER	0x00U	SPDCR レジスタ　RSPI 受信／送信データ選択ビット 0：SPDR は受信バッファを読み出す
_RSPI_READ_SPDR_TX_BUFFER	0x10U	SPDCR レジスタ　RSPI 受信／送信データ選択ビット 1：SPDR は送信バッファを読み出す
_RSPI_ACCESS_WORD	0x00U	SPDCR レジスタ　RSPI ロングワードアクセス／ワードアクセス設定ビット 0：SPDR レジスタへはワードアクセス
_RSPI_ACCESS_LONGWORD	0x20U	SPDCR レジスタ　RSPI ロングワードアクセス／ワードアクセス設定ビット 1：SPDR レジスタへはロングワードアクセス
_RSPI_RSPCK_DELAY_1 ~ 8	0x00U ~ 0x07U	SPCKD レジスタ　RSPCK 遅延設定ビット RSPCK 遅延の設定
_RSPI_SSL_NEGATION_DELAY_1 ~ 8	0x00U ~ 0x07U	SSLND レジスタ　SSL ネゲート遅延設定ビット SSL ネゲート遅延の設定

定数名	設定値	内容
_RSPI_NEXT_ACCESS_DELAY_1 ~ 8	0x00U ~ 0x07U	SPND レジスタ　RSPI 次アクセス遅延設定ビット 次アクセス遅延の設定
_RSPI_PARITY_DISABLE	0x00U	SPCR2 レジスタ　パリティ許可ビット 0：送信データパリティビットを付加しない
_RSPI_PARITY_ENABLE	0x01U	SPCR2 レジスタ　パリティ許可ビット 1：送信データにパリティビットを付加し、受信データのパリティチェックを行う（SPCR.TXMD = 0 のとき） 送信データにパリティビットを付加するが、受信データのパリティチェックは行わない（SPCR.TXMD = 1 のとき）
_RSPI_PARITY_EVEN	0x00U	SPCR2 レジスタ　パリティモードビット 0：偶数パリティで送受信
_RSPI_PARITY_ODD	0x02U	SPCR2 レジスタ　パリティモードビット 1：奇数パリティで送受信
_RSPI_IDLE_INTERRUPT_DISABLED	0x00U	SPCR2 レジスタ　RSPI アイドル割り込み許可ビット 0：アイドル割り込み要求の発生を禁止
_RSPI_IDLE_INTERRUPT_ENABLED	0x04U	SPCR2 レジスタ　RSPI アイドル割り込み許可ビット 1：アイドル割り込み要求の発生を許可
_RSPI_SELF_TEST_DISABLED	0x00U	SPCR2 レジスタ　パリティ自己判断ビット 0：パリティ回路自己診断機能は無効
_RSPI_SELF_TEST_ENABLED	0x08U	SPCR2 レジスタ　パリティ自己判断ビット 1：パリティ回路自己診断機能が有効
_RSPI_AUTO_STOP_DISABLED	0x00U	SPCR2 レジスタ　RSPCK 自動停止機能許可ビット 0：RSPCK 自動停止機能が無効
_RSPI_AUTO_STOP_ENABLED	0x10U	SPCR2 レジスタ　RSPCK 自動停止機能許可ビット 1：RSPCK 自動停止機能が有効
_RSPI_RSPCK_SAMPLING_ODD	0x0000U	SPCMD0~7 レジスタ　RSPCK 位相設定ビット 0：奇数エッジでデータサンプル、偶数エッジでデータ変化
_RSPI_RSPCK_SAMPLING_EVEN	0x0001U	SPCMD0~7 レジスタ　RSPCK 位相設定ビット 1：奇数エッジでデータ変化、偶数エッジでデータサンプル
_RSPI_RSPCK_POLARITY_LOW	0x0000U	SPCMD0~7 レジスタ　RSPCK 極性設定ビット 0：アイドル時の RSPCK が Low
_RSPI_RSPCK_POLARITY_HIGH	0x0002U	SPCMD0~7 レジスタ　RSPCK 極性設定ビット 1：アイドル時の RSPCK が High
_RSPI_BASE_BITRATE_1	0x0000U	SPCMD0~7 レジスタ　ビットレート分周設定ビット 0 0：ベースのビットレートを選択
_RSPI_BASE_BITRATE_2	0x0004U	SPCMD0~7 レジスタ　ビットレート分周設定ビット 0 1：ベースのビットレートの 2 分周を選択
_RSPI_BASE_BITRATE_4	0x0008U	SPCMD0~7 レジスタ　ビットレート分周設定ビット 1 0：ベースのビットレートの 4 分周を選択
_RSPI_BASE_BITRATE_8	0x000CU	SPCMD0~7 レジスタ　ビットレート分周設定ビット 1 1：ベースのビットレートの 8 分周を選択
_RSPI_SIGNAL_ASSERT_SSL0 ~ 3	0x0000U ~ 0x0030U	SPCMD0~7 レジスタ　SSL 信号アサート設定ビット SSLy0 ~ SSLy3 以外は設定禁止
_RSPI_SSL_KEEP_DISABLE	0x0000U	SPCMD0~7 レジスタ　SSL 信号レベル保持ビット 0：転送終了時に全 SSL 信号をネゲート
_RSPI_SSL_KEEP_ENABLE	0x0080U	SPCMD0~7 レジスタ　SSL 信号レベル保持ビット 1：転送終了後から次アクセス開始まで SSL 信号レベルを保持
_RSPI_DATA_LENGTH_BITS_8 ~ 32	0x0400U ~ 0x0200U	SPCMD0~7 レジスタ　RSPI データ長設定ビット
_RSPI_MSB_FIRST	0x0000U	SPCMD0~7 レジスタ　RSPI LSB ファーストビット 0：MSB ファースト
_RSPI_LSB_FIRST	0x1000U	SPCMD0~7 レジスタ　RSPI LSB ファーストビット 1：LSB ファースト
_RSPI_NEXT_ACCESS_DELAY_DISABLE	0x0000U	SPCMD0~7 レジスタ　RSPI 次アクセス遅延許可ビット 0：次アクセス遅延は 1RSPCK+2SERICLK
_RSPI_NEXT_ACCESS_DELAY_ENABLE	0x2000U	SPCMD0~7 レジスタ　RSPI 次アクセス遅延許可ビット 1：次アクセス遅延は RSPI 次アクセス遅延レジスタ SPND）の設定値
_RSPI_NEGATION_DELAY_DISABLE	0x0000U	SPCMD0~7 レジスタ　SSL ネゲート遅延設定許可ビット 0：SSL ネゲート遅延は 1RSPCK

定数名	設定値	内容
_RSPI_NEGATION_DELAY_ENABLE	0x4000U	SPCMD0~7 レジスタ　SSL ネゲート遅延設定許可ビット 1：SSL ネゲート遅延は RSPI スレーブセレクトネゲート遅延レジスタ（SSLND）の設定値
_RSPI_RSPCK_DELAY_DISABLE	0x0000U	SPCMD0~7 レジスタ　RSPCK 遅延設定許可ビット 0：RSPCK 遅延は 1RSPCK
_RSPI_RSPCK_DELAY_ENABLE	0x8000U	SPCMD0~7 レジスタ　RSPCK 遅延設定許可ビット 1：RSPCK 遅延は RSPI クロック遅延レジスタ（SPCKD）の設定値
_RSPI_PRIORITY_LEVEL0 ~ 15	0x00000000UL ~ 0x0000000FUL	PRLn レジスタ　割り込み優先レベル格納ビット 割り込み優先レベルは、0 が最も高く、15 が最も低い。

3.2.5　SCIFA_UART 制御

表 3-51 SCIFA_UARTドライバで使用する定数

定数名	設定値	内容
SCIFA_HVA_WRITE_DATA	0u	割り込みアドレスレジスタ初期化
SCIFA_UART_SUCCESS	0	成功
SCIFA_UART_ERR	-1	エラー
SCIFA_UART_ERR_RECEIVE	-2	受信エラー
SCIFA_UART_CH_TOTAL	5	SCIFA のチャネル数
SCIFA_UART_CH_0	0	SCIFA のチャネル 0 を指定するための定数です。
SCIFA_UART_CH_1	1	SCIFA のチャネル 1 を指定するための定数です。
SCIFA_UART_CH_2	2	SCIFA のチャネル 2 を指定するための定数です。
SCIFA_UART_CH_3	3	SCIFA のチャネル 3 を指定するための定数です。
SCIFA_UART_CH_4	4	SCIFA のチャネル 4 を指定するための定数です。
SCIFA_UART_MODE_R	1	SCIFA を受信モードで使用するための定数です。 SCIFA チャネル初期化関数および SCIFA チャネルオープン関数の引数に使用します。
SCIFA_UART_MODE_W	2	SCIFA を送信モードで使用するための定数です。 SCIFA チャネル初期化関数および SCIFA チャネルオープン関数の引数に使用します。
SCIFA_UART_MODE_RW	3	SCIFA を送受信モードで使用するための定数です。 SCIFA チャネル初期化関数および SCIFA チャネルオープン関数の引数に使用します。
SCIFA_UART_CKS_DIVISION_1	0	SCIFA のボーレートジェネレータのクロックソースを SERICLK クロックに設定するための定数です。SCIFA チャネル初期化関数の引数に使用します。
SCIFA_UART_CKS_DIVISION_4	1	SCIFA のボーレートジェネレータのクロックソースを SERICLK/4 クロックに設定するための定数です。SCIFA チャネル初期化関数の引数に使用します。
SCIFA_UART_CKS_DIVISION_16	2	SCIFA のボーレートジェネレータのクロックソースを ERICLK/16 クロックに設定するための定数です。SCIFA チャネル初期化関数の 引数に使用します。
SCIFA_UART_CKS_DIVISION_64	3	SCIFA のボーレートジェネレータのクロックソースを ERICLK/64 クロックに設定するための定数です。SCIFA チャネル初期化関数の 引数に使用します。

3.2.6　シリアル・フラッシュ ROM 制御

表 3-52 シリアル・フラッシュROMドライバで使用する定数

定数名	設定値	内容
SFLASH_BASE_ADDRESS	0x10000000	シリアル・フラッシュ ROM ベースアドレス
SFLASH_CHANNEL_MAX	1	チャネル数
SFLASH_DEVICE_SIZE	crr_sflash_info.device_size	デバイス情報構造体-device_size
SFLASH_ERASE_SIZE	crr_sflash_info.erase_size	デバイス情報構造体-erase_size
SFLASH_PROGRAM_SIZE	crr_sflash_info.page_size	デバイス情報構造体-page_size
SFLASH_ID	crr_sflash_info.id	デバイス情報構造体-id

3.2.7　USB ファンクション制御

表 3-53 USBファンクションドライバで使用する定数

定数名	設定値	内容
USB_DMA_USE_PP	1	DMA 転送使用
USB_DMA_NOT_USE_PP	0	DMA 転送未使用
USB_DMA_PP	USB_DMA_USE_PP	DMA 転送設定
USB_LPWR_USE_PP	1	Low Power Mode を使用する
USB_LPWR_NOT_USE_PP	0	Low Power Mode を使用しない
USB_CPU_LPW_PP	USB_LPWR_USE_PP	Low Power Mode 設定
USB_DEBUG_ON_PP	1	デバック情報表示有効
USB_DEBUG_OFF_PP	0	デバック情報表示無効
USB_DEBUG_OUTPUT_PP	USB_DEBUG_OFF_PP	デバッグ情報出力関数設定
USB_OK	USB_ER_t(0)	成功
USB_ERROR	USB_ER_t(-1L)	失敗
USB_QOVR	USB_ER_t(-43L)	指定したパイプがデータ転送処理中
USB_TRUE	1	リモートウェイクアップイネーブルフラグ：許可
USB_FALSE	0	リモートウェイクアップイネーブルフラグ：禁止
USB_NULL	0	NULL
USB_CS_SQER	0x0006u	Sequence error
USB_CS_WRND	0x0005u	Ctrl write nodata status stage
USB_CS_WRSS	0x0004u	Ctrl write status stage
USB_CS_WRDS	0x0003u	Ctrl write data stage
USB_CS_RDSS	0x0002u	Ctrl read status stage
USB_CS_RDDS	0x0001u	Ctrl read data stage
USB_CS_IDST	0x0000u	Idle or setup stage
USB_CUSE	0u	CFIFO に対し、CPU アクセスする
USB_D0DMA	2u	D0FIFO に対し、DMA アクセスする（サイクルスチルモード）
USB_D0DMA_C	6u	D0FIFOBn に対し、DMA アクセスする（32byte 連続アクセスモード）
USB_D1DMA_C	7u	D1FIFOBn に対し、DMA アクセスする（32byte 連続アクセスモード）
USB_FIFOERROR	0xFFFFu	FIFO アクセスエラー発生
USB_WRITEEND	0x0000u	データ書き込み終了（継続データなし/データ長"0"のパケット送信）
USB_WRITESHRT	0x0001u	データ書き込み終了（ショートパケットデータ書き込み）
USB_WRITING	0x0002u	データ書き込み中（継続データあり）
USB_PDTBLEND	0xFFFFu	パイプ情報テーブルの終端
USB_CTRL_END	0u	ステータスステージを正常に終了させる。
USB_DATA_NONE	1u	データ送信が正常終了した場合

定数名	設定値	内容
USB_DATA_OK	3u	データ受信が正常終了した場合
USB_DATA_SHT	4u	データ受信は正常終了したが指定されたデータ長未満で終了した場合
USB_DATA_OVR	5u	受信データがサイズオーバーした場合
USB_DATA_STALL	6u	STALL 応答もしくは MaxPacketSize エラーを検出した場合
USB_DATA_ERR	7u	ステータスステージでホストに STALL を返す。
USB_DATA_STOP	8u	データ転送を強制終了した場合
USB_DO_REMOTEWAKEUP	0x0155u	PCD へのリモートウェイクアップ実行要求を行います。
USB_DO_STALL	0x0164u	PCD へのストール応答実行要求を行います。
USB_PCDC_USE_PIPE_IN	USB_PIPE1	Bulk IN 転送パイプ番号
USB_PCDC_USE_PIPE_OUT	USB_PIPE2	Bulk OUT 転送パイプ番号
USB_PCDC_USE_PIPE_STATUS	USB_PIPE6	Interrupt IN 転送パイプ番号
USB_HSCONNECT	0x00C0u	動作するデバイススピード：High-Speed
USB_FSCONNECT	0x0080u	動作するデバイススピード：Full-Speed
USB_NOCONNECT	0x0000u	動作するデバイススピード：No connect
USB_PIPE0	0x0000u	使用するパイプ（USB_PIPE1～USB_PIPE9）
USB_PIPE1	0x0001u	-
USB_PIPE2	0x0002u	-
USB_PIPE3	0x0003u	-
USB_PIPE4	0x0004u	-
USB_PIPE5	0x0005u	-
USB_PIPE6	0x0006u	-
USB_PIPE7	0x0007u	-
USB_PIPE8	0x0008u	-
USB_PIPE9	0x0009u	-
USB_BULK	0x4000u	パイプ転送タイプ (USB_BULK/USB_INT/USB_ISO)
USB_INT	0x8000u	-
USB_ISO	0xC000u	-
USB_BFREOFF	0x0000u	BRDY 割り込み動作指定
USB_DBLBON	0x0200u	ダブルバッファモード
USB_DBLBOFF	0x0000u	-
USB_CNTMDON	0x0100u	連続転送モード
USB_CNTMDOFF	0x0000u	-
USB_SHTNAKON	0x0080u	SHTNAK 動作指定
USB_SHTNAKOFF	0x0000u	-
USB_DIR_P_OUT	0x0000u	転送方向
USB_DIR_P_IN	0x0010u	-
USB_EP1	0x0001u	パイプに対するエンドポイント番号(EP1～EP15)
USB_EP2	0x0002u	-
USB_EP3	0x0003u	-
USB_EP4	0x0004u	-
USB_EP5	0x0005u	-
USB_EP6	0x0006u	-
USB_EP7	0x0007u	-
USB_EP8	0x0008u	-
USB_EP9	0x0009u	-
USB_EP10	0x000Au	-
USB_EP11	0x000Bu	-
USB_EP12	0x000Cu	-
USB_EP13	0x000Du	-

定数名	設定値	内容
USB_EP14	0x000Eu	-
USB_EP15	0x000Fu	-

3.2.8 WDTA 制御

表 3-54 WDTAドライバで使用する定数

定数名	設定値	内容
WDT_STAT_REFRESH_ERR_MASK	(0x8000)	WDT ステータスからリフレッシュエラーフラグを取得するための定義です。
WDT_STAT_UNDERFLOW_ERR_MASK	(0x4000)	WDT ステータスからアンダーフローフラグを取得するための定義です。
WDT_STAT_ERROR_MASK	(0xC000)	WDT ステータスからリフレッシュエラーフラグ、アンダーフローフラグを取得するための定義です。
WDT_STAT_COUNTER_MASK	(0x3FFF)	WDT ステータスからカウンタ値を取得するための定義です。
WDT_CFG_PARAM_CHECKING_ENABLE	(1)	WDTA の API 関数でパラメータチェック許可の有効（1）／無効（0）を表します。

3.2.9 IWDTA 制御

表 3-55 IWDTAドライバで使用する定数

定数名	設定値	内容
IWDT_STAT_REFRESH_ERR_MASK	(0x8000)	IWDT ステータスからリフレッシュエラーフラグを取得するための定義です。
IWDT_STAT_UNDERFLOW_ERR_MASK	(0x4000)	IWDT ステータスからアンダーフローフラグを取得するための定義です。
IWDT_STAT_ERROR_MASK	(0xC000)	IWDT ステータスからリフレッシュエラーフラグ、アンダーフローフラグを取得するための定義です。
IWDT_STAT_COUNTER_MASK	(0x3FFF)	IWDT ステータスからカウンタ値を取得するための定義です。
IWDT_CFG_PARAM_CHECKING_ENABLE	(1)	IWDTA の API 関数でパラメータチェック許可の有効（1）／無効（0）を表します。

3.2.10　RIIC 制御

表 3-56 に RIIC ドライバで使用する定数、表 3-57 に RIIC ドライバで使用するエラーコードを示します。
表 3-58 , 表 3-59 にコンパイル時にコンフィグ可能な定数を示します。

表 3-56 RIICドライバで使用する定数

定数名	設定値	内容
RIIC_NO_INIT 注 1	0	未初期化状態
RIIC_IDLE 注 1	1	アイドル状態
RIIC_FINISH 注 1	2	アイドル状態
RIIC_NACK 注 1	3	アイドル状態
RIIC_COMMUNICATION 注 1	4	マスタ送受信状態／スレーブ送受信状態
RIIC_AL 注 1	5	アービトレーションロスト検出状態
RIIC_ERROR 注 1	6	エラー状態
RIIC_GEN_START_CON 注 2	(uint8_t)(0x01)	スタートコンディションの生成
RIIC_GEN_STOP_CON 注 2	(uint8_t)(0x02)	ストップコンディションの生成
RIIC_GEN_RESTART_CON 注 2	(uint8_t)(0x04)	リスタートコンディションの生成
RIIC_GEN_SDA_HI_Z 注 2	(uint8_t)(0x08)	SDA 端子を Hi-Z 出力
RIIC_GEN_SCL_ONESHOT 注 2	(uint8_t)(0x10)	SCL クロックのワンショット出力
RIIC_GEN_RESET 注 2	(uint8_t)(0x20)	RIIC のモジュールリセット
FIT_NO_PTR	(void *)0	FIT で定義されている NULL ポインタ

注 1. riic_ch_dev_status_t 型の値として使用
注 2. R_RIIC_Control()の出力パターンとして使用

表 3-57 RIICドライバで使用するエラーコード

定数名	設定値	内容
RIIC_SUCCESS	0U	関数コールが正常にできた場合
RIIC_ERR_LOCK_FUNC	1U	他のモジュールで RIIC が使用されている場合
RIIC_ERR_INVALID_CHAN	2U	存在しないチャネルを指定した場合
RIIC_ERR_INVALID_ARG	3U	不正な引数を設定した場合
RIIC_ERR_NO_INIT	4U	未初期化状態の場合
RIIC_ERR_BUS_BUSY	5U	バスビジーの場合
RIIC_ERR_AL	6U	アービトレーションロスト検出状態で関数コールした場合
RIIC_ERR_OTHER	7U	その他エラー

本モジュールのコンフィギュレーションオプションの設定は、r_riic_ec1_config.h で行います。
オプション名および設定値に関する説明を、下表に示します。

表 3-58 コンパイル時の設定（1/2）

定義	設定値
RIIC_CFG_PARAM_CHECKING_ENABLE 注. デフォルト値は“1”	パラメータチェック処理をコードに含めるか選択できます。 “0”を選択すると、パラメータチェック処理をコードから省略できるため、コードサイズが削減できます。 “0”の場合、パラメータチェック処理をコードから省略します。 “1”の場合、パラメータチェック処理をコードに含めます。
RIIC_CFG_PCLK_Hz 注. デフォルト値は“75000000”	RIIC1 モジュールに供給されるクロック（PCLK）の周波数を設定してください。 “RIIC_CFG_CH0_kBPS”と“RIIC_CFG_PCLK_Hz”の設定値からビットレートレジスタおよび内部基準クロック選択ビットへの設定値が算出されます。
RIIC_CFG_CH_NUMBER 注. デフォルト値は“1”	RIIC 通信にて使用するチャネルを設定してください。
RIIC_CFG_CH_kBPS 注. デフォルト値は“400”	RIIC1 の通信速度を設定できます。 “RIIC_CFG_CH_kBPS”と“RIIC_CFG_PCLK_Hz”の設定値からビットレートレジスタおよび内部基準クロック選択ビットへの設定値が算出されます。 “400”以下の値を設定してください。
RIIC_SCL_100K_UP_TIME 注. デフォルト値は“1000E-9”	RIIC1 の通信速度が 1-100[KBPS]時の SCL の立ち上がり時間[s]を設定してください。（double 型で指定）
RIIC_SCL_100K_DOWN_TIME 注. デフォルト値は“300E-9”	RIIC1 の通信速度が 1-100[KBPS]時の SCL の立ち下がり時間[s]を設定してください。（double 型で指定）
RIIC_SCL_400K_UP_TIME 注. デフォルト値は“175E-9”	RIIC1 の通信速度が 101-400[KBPS]時の SCL の立ち上がり時間[s]を設定してください。（double 型で指定）
RIIC_SCL_400K_DOWN_TIME 注. デフォルト値は“175E-9”	RIIC1 の通信速度が 101-400[KBPS]時の SCL の立ち下がり時間[s]を設定してください。（double 型で指定）
RIIC_CFG_CH_DIGITAL_FILTER 注. デフォルト値は“2”	ノイズフィルタの段数を選択できます。 “0”の場合、ノイズフィルタは無効となります。 “1”～“4”の場合、選択した段数のフィルタが有効になるようノイズフィルタ段数選択ビットおよびデジタルノイズフィルタ回路有効ビットの設定値が選択されます。
RIIC_CFG_CH_SCL 注. デフォルト値は“1”	RIIC1 の SCL の出力端子を選択できます。 選択した端子を SCL 端子とする処理をコードに含めます。 “0”の場合、SCL 端子の設定処理をコードから省略します。 “1”の場合、PC6 を SCL 端子として設定します。
RIIC_CFG_CH_SDA 注. デフォルト値は“1”	RIIC1 の SDA の出力端子を選択できます。 選択した端子を SDA 端子とする処理をコードに含めます。 “0”の場合、SDA 端子の設定処理をコードから省略します。 “1”の場合、PC7 を SDA 端子として設定します。
RIIC_CFG_CH_MASTER_MODE 注. デフォルト値は“1”	マスタアービトレーションロスト検出機能の有効／無効を選択できます。 マルチマスタで使用する場合は、“1”(有効)にしてください。 “0”の場合、マスタアービトレーションロスト検出を無効にします。 “1”の場合、マスタアービトレーションロスト検出を有効にします。
RIIC_CFG_CH_SLV_ADDR0_FORMAT 注 1 RIIC_CFG_CH_SLV_ADDR1_FORMAT 注 2 RIIC_CFG_CH_SLV_ADDR2_FORMAT 注 2 注 1. デフォルト値は“1” 注 2. デフォルト値は“0”	スレーブアドレスのフォーマットを 7 ビット／10 ビットから選択できます。 “0”の場合、スレーブアドレスを設定しません。 “1”の場合、7 ビットアドレスフォーマットに設定します。 “2”の場合、10 ビットアドレスフォーマットに設定します。
RIIC_CFG_CH_SLV_ADDR0 注 1 RIIC_CFG_CH_SLV_ADDR1 注 2 RIIC_CFG_CH_SLV_ADDR2 注 2 注 1. デフォルト値は“0x0025” 注 2. デフォルト値は“0x0000”	スレーブアドレスを設定します。 “RIIC_CFG_CH_SLV_ADDRi_FORMAT”の設定値によって、設定可能範囲が変わります。 “RIIC_CFG_CH_SLV_ADDRi_FORMAT”が“0”の場合は、設定値は無効となります。 “1”の場合は、設定値の下位 7 ビットが有効となります。 “2”の場合は、設定値の下位 10 ビットが有効となります。

表 3-59 コンパイル時の設定（2/2）

定義	設定値
RIIC_CFG_CH_SLV_GCA_ENABLE 注. デフォルト値は“0”	ゼネラルコールアドレスの有効／無効が選択できます。 “0”の場合、ゼネラルコールアドレスを無効にします。 “1”の場合、ゼネラルコールアドレスを有効にします。
RIIC_CFG_CH_INT_PRIORITY 注. デフォルト値は“1”	通信エラー／イベント発生割り込み(EEI)、受信データフル割り込み(RXI)、送信データエンプティ割り込み(TXI)、送信終了割り込み(TEI)の優先レベルを選択できます。 “1”～“15”の範囲で設定してください。
RIIC_CFG_BUS_CHECK_COUNTER 注. デフォルト値は“1000”	RIIC の API 関数のバスチェック処理時の、タイムアウトカウンタ(バス確認回数)を設定できます。 “0xFFFFFFFF”以下の値を設定してください。 バスチェック処理は、 ・スタートコンディション生成前 ・ストップコンディション検出後 ・RIIC 制御機能(R_RIIC_Control 関数)を使用した各コンディションおよび SCL ワンショットパルスの生成後に行います。 バスチェック処理では、バスビジー時、バスフリーになるまでタイムアウトカウンタをデクリメントします。“0”になるとタイムアウトと判断し、戻り値でエラー（Busy）を返します。 注. バスロックなどでロックされないようにするためのカウンタであるため、相手デバイスが SCL 端子を“L”ホールドする時間以上になるよう値を設定してください。 「タイムアウト時間(ns) ≒ (1 / ICLK(Hz)) * カウンタ値 * 10」

3.2.11　USB ホスト制御

共通ユーザ定義情報ファイル（r_usb_basic_config.h）を書き換えることにより、USB-BASIC-F/W の機能設定をします。
システムに合わせて、下記項目を変更してください。

表 3-60 ホストコントロールドライバ(HCD)で使用する定数

定数名	設定値	内容
USB_MAXDEVADDR	12u	デバイスアドレスの最大値（接続できるデバイスの最大数）を設定します。 設定値を大きくすると使用メモリが増加します。
USB_IDMAX	11u	スケジューラのタスク ID の最大数を設定します。 設定値を大きくすると使用メモリが増加します。
USB_PRIMAX	8u	スケジューラのプライオリティの最大値を設定します。 設定値を大きくすると使用メモリが増加します。
USB_BLKMAX	21u	スケジューラのメモリプールの最大数を設定します。 設定値を大きくすると使用メモリが増加します。
USB_TABLEMAX	USB_BLKMAX	スケジューラのメッセージプールの最大数を設定します。 設定値を大きくすると使用メモリが増加します。
USB_MAXPIPE	32u	パイプの最大数を設定します。 接続されたデバイスのエンドポイント毎に 1 つ使用します。 設定値を大きくすると使用メモリが増加します。
USB_HOST_COMPLIANCE_MODE	-	USB Embedded Host のコンプライアンステスト対応設定をします。 コンプライアンステストに対応する場合、本マクロを有効にしてください。
USB_COMPLIANCE_DISP(data1, data2)	usb_cstd_DummyFunction(data1, data2)	コンプライアンステスト対応時に表示機器（LCD 等）に情報を表示させるための関数を登録してください。
USB_OVERCURRENT(rootport)	usb_cstd_DummyFunction(rootport)	オーバーカレント検出時に呼び出される関数を登録してください。
USB_DEBUG_OUTPUT	-	デバッグ情報の出力設定をします。
USB_PRINTF0(FORM) ~ USB_PRINTF8(FORM,x1,x2,x3,x4,x5,x6,x7,x8)	printf(FORM) ~ printf(FORM,x1,x2,x3,x4,x5,x6,x7,x	UART および表示デバイスなどにデバッグ情報を出力するマクロです。シリアルドライバ、表示デバイス用ドライバが必要になります。 ユーザシステムに合わせて関数を登録してください。

表 3-61 EHCIユーザ定義ファイル(r_usb_hEhciDefUsr.h)の設定

定数名	設定値	内容
USB_EHCI_PFL_SIZE	256	EHCI ピリオディックフレームリストのサイズを 256、512、1024 の値から指定してください。 このサイズ指定によりピリオディック転送のスケジューリング幅が変更されます。
USB_EHCI_NUM_QH	16	EHCI キューヘッドデータ構造メモリの最大数を指定してください。 キューヘッドデータ構造は、コントロール転送、バルク転送、インターラプト転送のエンドポイントパイプ数分必要となります。 ユーザシステムに合わせて設定してください。
USB_EHCI_NUM_QTD	256	EHCI qTD(デバイスステータスレジスタキュー要素転送記述子)データ構造メモリの最大数を指定してください。 qTD は、コントロール転送、バルク転送、インターラプト転送において転送管理用の記述子としてキューヘッドとリンクして使用されます。 一つの qTD で転送可能な最大データサイズは 20480byte です。 また、コントロール転送の場合は、SETUP ステージ、DATA ステージ、STATUS ステージ毎に一つの qTD が必要となります。 ユーザシステムに合わせて設定してください。
USB_EHCI_NUM_ITD	4	EHCI iTD（ハイスピード用アイソクロナス転送記述子）データ構造メモリの最大数を指定してください。 iTD データ構造は、ハイスピードアイソクロナス転送のエンドポイントパイプ数分必要となります。 ユーザシステムに合わせて設定してください。
USB_EHCI_NUM_SITD	4	EHCI siTD（スプリットトランザクションアイソクロナス転送記述子）データ構造メモリの最大数を指定してください。 siTD データ構造は､スプリットトランザクションアイソクロナス転送のエンドポイントパイプ数分必要となります。 ユーザシステムに合わせて設定してください。
USB_EHCI_ITD_DATA_SIZE	512	EHCI iTD データ構造の最大データ転送サイズを指定してください。 この転送サイズは、ハイスピードアイソクロナス転送の 1 回分（1 トランザクション）の最大転送サイズを指定するものです。 設定可能な最大データサイズは 1024 です。 ユーザシステムに合わせて設定してください。
USB_EHCI_TIMEOUT	3000	EHCI タイムアウト時間を msec 単位で指定してください。

表 3-62 OHCIユーザ定義ファイル(r_usb_hOhciDefUsr.h)の設定

定数名	設定値	内容
USB_OHCI_NUM_ENDPOINT	16	OHCI エンドポイントデータ構造メモリの最大数を指定してください。 OHCI エンドポイントデータ構造は、コントロール転送、バルク転送、インターラプト転送、アイソクロナス転送のエンドポイントパイプ数分必要となります。 ユーザシステムに合わせて設定してください。
USB_OHCI_NUM_ED	64	OHCI エンドポイントディスクリプタデータ構造メモリの最大数を指定してください。 OHCI エンドポイントディスクリプタデータ構造は、コントロール転送、バルク転送、インターラプト転送、アイソクロナス転送のエンドポイントパイプ数＋インターラプト転送のスケジューリング用に 31 個必要となります。 ユーザシステムに合わせて設定してください。
USB_OHCI_NUM_TD	256	OHCI トランスファディスクリプタデータ構造メモリの最大数を指定してください。 OHCI トランスファディスクリプタデータ構造は、コントロール転送、バルク転送、インターラプト転送、アイソクロナス転送において転送管理用の記述子として使用されます。 一つのトランスファディスクリプタで転送可能な最大データサイズは 8192byte です。 ユーザシステムに合わせて設定してください。
USB_OHCI_ISO_MAXDEVICE	4	OHCI アイソクロナスデバイスの最大数を指定してください。 ユーザシステムに合わせて設定してください。
USB_OHCI_ISO_MAX_PACKET_SIZE	256	OHCI アイソクロナスデバイスの最大データ転送サイズを指定してください。 この転送サイズは、OHCI アイソクロナス転送の 1 回分（1 トランザクション）の最大転送サイズを指定するものです。 設定可能な最大データサイズは 1023 です。 ユーザシステムに合わせて設定してください。
USB_OHCI_ISO_MAX_FRAME	8	OHCI アイソクロナス転送の最大フレーム数を指定してください。 このフレーム数は、OHCI アイソクロナス転送時の最大転送フレーム数を指定するものです。 設定値は 2 のべき乗で、設定可能な最大フレーム数は 8 です。 ユーザシステムに合わせて設定してください。
USB_OHCI_TIMEOUT	3000	OHCI タイムアウト時間を msec 単位で指定してください。

3.3　関数一覧

ドライバで用意している関数の一覧を示します。

表 3-63 CANドライバ関数一覧

関数名	関数概要	ヘッダ
R_CAN_Open	CAN モジュールの起動	r_can_api.h
R_CAN_Close	CAN モジュールの停止	r_can_api.h
R_CAN_GlobalControl	グローバル・モードの遷移	r_can_api.h
R_CAN_ChannelControl	チャネル・モードの遷移	r_can_api.h
R_CAN_SetBitrate	通信速度の設定	r_can_api.h
R_CAN_UseBufferEntry	送信・受信で使用するバッファの登録	r_can_api.h
R_CAN_SetRxFifoBuffer	受信 FIFO バッファの設定	r_can_api.h
R_CAN_SetFifoBuffer	送受信 FIFO バッファの設定	r_can_api.h
R_CAN_ReleaseFifoBuffer	送受信 FIFO バッファの開放	r_can_api.h
R_CAN_ReleaseRxFifoBuffer	受信 FIFO バッファの開放	r_can_api.h
R_CAN_ReleaseBuffer	送信バッファ、受信バッファの開放	r_can_api.h
R_CAN_GetTxBufferStatus	送信バッファステータスの読み出し	r_can_api.h
R_CAN_WriteBuffer	送信メッセージを送信バッファに書き込む	r_can_api.h
R_CAN_GetFifoStatus	送受信 FIFO バッファステータスの読み出し	r_can_api.h
R_CAN_WriteFifo	送信メッセージを送受信 FIFO バッファに書き込む	r_can_api.h
R_CAN_Tx	送信開始	r_can_api.h
R_CAN_RxSet	受信設定	r_can_api.h
R_CAN_ReadBuff	受信バッファから受信メッセージの読み出し	r_can_api.h
R_CAN_ReadRxFifo	受信 FIFO バッファから受信メッセージの読み出し	r_can_api.h
R_CAN_ReadFifo	送受信 FIFO バッファから受信メッセージの読み出し	r_can_api.h
R_CAN_GetFifoMessageNum	送受信 FIFO バッファの未読メッセージ数の取得	r_can_api.h
R_CAN_GetRxFifoMessageNum	受信 FIFO バッファの未読メッセージ数の取得	r_can_api.h
R_CAN_SetCommTestMode	テスト設定	r_can_api.h
R_CAN_ResetTestMode	テスト設定を解除しチャネル・通信・モードへ遷移	r_can_api.h
R_CAN_SetInterruptHandler	割り込みハンドラの登録	r_can_api.h
R_CAN_SetInterruptEnableDisable	CAN モジュール割り込みベクタの割り込み許可・禁止	r_can_api.h
R_CAN_GetInterruptSource	割り込み要因の取得	r_can_api.h
R_CAN_ClearInterruptSource	割り込み要因のクリア	r_can_api.h

表 3-64 CMTドライバ関数一覧

関数名	関数概要	ヘッダ
R_CMT_Init	CMT チャネル初期化処理	r_cmt.h
R_CMT_CreatePeriodic	周期イベント設定処理	r_cmt.h
R_CMT_CreateOneShot	ワンショットイベント設定処理	r_cmt.h
R_CMT_Stop	CMT 動作停止処理	r_cmt.h
userdef_cmt_init	CMT 初期設定	r_cmt.h
userdef_cmt_create	CMT 周期設定	r_cmt.h
userdef_cmt_stop	CMT 動作停止	r_cmt.h
userdef_cmt_isr_cmi	CMI 割り込みハンドラ	r_cmt.h
cmt0_isr	CMI0 割り込みハンドラ	-
cmt1_isr	CMI1 割り込みハンドラ	-
cmt2_isr	CMI2 割り込みハンドラ	-
cmt3_isr	CMI3 割り込みハンドラ	-

表 3-65 ETHERドライバ関数一覧

関数名	関数概要	ヘッダ
EtherCAT_init	EtherCAT® 初期化処理	r_ether.h
R_ETHER_IRQ_Init	Ether 割り込み要求初期化	r_ether.h
ether_irq_init	Ether 割り込み要求初期化	-
ecat_sync0_isr	EtherCAT SYNC 信号出力端子０ 割り込みハンドラ	-
ecat_sync1_isr	EtherCAT SYNC 信号出力端子１ 割り込みハンドラ	-
ecat_isr	EtherCAT 割り込みハンドラ	-

表 3-66 RSPIドライバ関数一覧

RSPIm で使用している m はチャネル番号を表しています。

関数名	関数概要	ヘッダ
R_RSPIm_Create	RSPI 制御初期化処理	r_rspi.h
R_RSPIm_Pin_Init	RSPI ポート設定初期化処理	r_rspi.h
R_RSPIm_Start	RSPI 通信開始処理	r_rspi.h
R_RSPIm_Stop	RSPI 通信終了処理	r_rspi.h
R_RSPI_GetState	RSPI 通信ステータス取得処理	r_rspi.h
R_RSPIm_ClearState	RSPI 通信ステータス初期化処理	r_rspi.h
R_RSPIm_Send_Receive	RSPI 送受信開始処理	r_rspi.h
r_rspim_callback_transmitend	送信完了コールバック関数	r_rspi.h
r_rspim_callback_receiveend	受信完了コールバック関数	r_rspi.h
r_rspim_callback_error	エラーコールバック関数	r_rspi.h
r_rspim_transmit_interrupt	送信バッファエンプティ割り込みハンドラ	r_rspi.h
r_rspim_receive_interrupt	受信バッファフル割り込みハンドラ	r_rspi.h
r_rspim_error_interrupt	RSPI エラー割り込みハンドラ	r_rspi.h
r_rspim_idle_interrupt	RSPI アイドル割り込みハンドラ	r_rspi.h

表 3-67 SCFIA_UARTドライバ関数一覧

関数名	関数概要	ヘッダ
R_SCIFA_SetInterruptHandler	SCIFA 割り込みハンドラ登録	r_scifa_api.h
R_SCIFA_EnableItrReg	SCIFA 受信割り込み有効	r_scifa_api.h
R_SCIFA_ResetItrReg	SCIFA 受信割り込み無効	r_scifa_api.h
R_SCIFA_UART_Init	SCIFA 初期設定処理	r_scifa_uart.h
R_SCIFA_UART_Open	SCIFA 起動処理	r_scifa_uart.h
R_SCIFA_UART_Receive	SCIFA データ受信処理	r_scifa_uart.h
R_SCIFA_UART_Send	SCIFA データ送信処理	r_scifa_uart.h
userdef_scifa0_uart_init	SCIFA チャネル 0UART モード初期設定処理	r_scifa_uart.h
userdef_scifa0_uart_open	SCIFA チャネル 0UART モード起動処理	r_scifa_uart.h
userdef_scifa0_uart_receive	SCIFA チャネル 0UART モード受信処理	r_scifa_uart.h
userdef_scifa0_uart_send	SCIFA チャネル 0UART モード送信処理	r_scifa_uart.h
__write	文字列出力	siorw.c
__read	文字列入力	siorw.c
IoInitScifa0	入出力初期化(SCIFA チャネル 0 初期化)	siochar.c
IoGetchar	文字取得	siochar.c
IoPutchar	文字出力	siochar.c

表 3-68 シリアル・フラッシュROMドライバ関数一覧

関数名	関数概要	ヘッダ
R_SFLASH_init	シリアル・フラッシュ ROM 制御初期化	r_sflash.h
R_SFLASH_program	シリアル・フラッシュ ROM へデータ書き込み	r_sflash.h
R_SFLASH_erase	シリアル・フラッシュ ROM のデータ消去	r_sflash.h

表 3-69 USBファンクションドライバ関数一覧

関数名	関数概要	ヘッダ
R_USB_Open	USB モジュールの初期設定	r_usb_basic_if.h
R_usb_pstd_TransferStart	データ転送実行要求	r_usb_basic_if.h
R_usb_pstd_TransferEnd	データ転送強制終了要求	r_usb_basic_if.h
R_usb_pstd_ChangeDeviceState	USB デバイスステート遷移処理	r_usb_basic_if.h
R_usb_pstd_DriverRegistration	PDCD 情報登録	r_usb_basic_if.h
R_usb_cstd_SetBuf	指定したパイプの PID を BUF に設定	r_usb_basic_if.h
R_usb_pstd_SetPipeStall	指定したパイプの PID を STALL に設定	r_usb_basic_if.h
R_usb_pstd_ControlRead	コントロール IN 転送用データ転送実行要求	r_usb_basic_if.h
R_usb_pstd_ControlWrite	コントロール OUT 転送用データ転送実行要求	r_usb_basic_if.h
R_usb_pstd_ControlEnd	コントロール転送終了要求	r_usb_basic_if.h
R_usb_pstd_poll	USB 割り込み処理	r_usb_basic_if.h
R_usb_pcdc_SendData	USB 送信処理	r_usb_pcdc_if.h
R_usb_pcdc_ReceiveData	USB 受信処理	r_usb_pcdc_if.h
R_usb_pcdc_SerialStateNotification	クラスノーティフィケーション”SerialState”送信	r_usb_pcdc_if.h
R_usb_pcdc_ctrltrans	CDC 用コントロール転送処理	r_usb_pcdc_if.h

表 3-70 WDTAドライバ関数一覧

関数名	関数概要	ヘッダ
R_WDT_Open	WDT オープン	r_wdt_if.h
R_WDT_Control	WDT コントロール	r_wdt_if.h

表 3-71 IWDTAドライバ関数一覧

関数名	関数概要	ヘッダ
R_IWDT_Open	IWDT オープン	r_iwdt_if.h
R_IWDT_Control	IWDT コントロール	r_iwdt_if.h

表 3-72 RIICドライバ関数一覧

関数名	関数概要	ヘッダ
R_RIIC_Open	RIIC モジュールの起動	r_riic_ec1_if.h
R_RIIC_MasterSend	RIIC マスタ送信の開始	r_riic_ec1_if.h
R_RIIC_MasterReceive	RIIC マスタ受信の開始	r_riic_ec1_if.h
R_RIIC_SlaveTransfer	RIIC スレーブ送信・受信状態への遷移	r_riic_ec1_if.h
R_RIIC_GetStatus	RIIC モジュールの状態確認	r_riic_ec1_if.h
R_RIIC_Control	RIIC コントロール処理	r_riic_ec1_if.h
R_RIIC_Close	RIIC モジュールの停止	r_riic_ec1_if.h
R_RIIC_GetVersion	RIIC バージョン取得	r_riic_ec1_if.h

表 3-73 USBホスト USB-BASIC-F/W API関数一覧

関数名	関数概要	ヘッダ
R_usb_hstd_TransferStart	データ転送実行要求	r_usb_basic_if.h
R_usb_hstd_TransferEnd	データ転送強制終了要求	r_usb_basic_if.h
R_usb_hstd_DriverRegistration	HDCD 登録	r_usb_basic_if.h
R_usb_hstd_ReturnEnuMGR	クラスチェック完了通知	r_usb_basic_if.h
R_usb_hstd_ChangeDeviceState	接続デバイスの状態変更要求	r_usb_basic_if.h
R_usb_hstd_SetPipe	パイプ情報設定	r_usb_basic_if.h
R_usb_hstd_GetPipeID	パイプ番号取得	r_usb_basic_if.h
R_usb_hstd_ClearPipe	パイプ情報クリア	r_usb_basic_if.h
R_usb_hstd_MgrOpen	MGR タスク起動	r_usb_basic_if.h
R_usb_hstd_MgrTask	MGR タスク	r_usb_basic_if.h
R_usb_hhub_Registration	HUB Class Driver(HUBCD)登録	r_usb_basic_if.h
R_usb_hhub_Task	HUB タスク	r_usb_basic_if.h

表 3-74 USBホスト non-OSスケジューラ API関数一覧

関数名	関数概要	ヘッダ
R_usb_cstd_SndMsg	優先テーブルに処理要求を送信する。	r_usb_basic_if.h
R_usb_cstd_RecMsg	優先テーブルに処理要求があるか確認する。	r_usb_basic_if.h
R_usb_cstd_PgetBlk	処理要求を格納する領域を確保する。	r_usb_basic_if.h
R_usb_cstd_RelBlk	処理要求を格納する領域を解放する。	r_usb_basic_if.h
R_usb_cstd_Scheduler	各タスクからの処理要求を管理する。	r_usb_basic_if.h
R_usb_cstd_SetTaskPri	タスクの優先度を設定する。	r_usb_basic_if.h
R_usb_cstd_CheckSchedule	登録されたタスクに処理要求があるか確認する。	r_usb_basic_if.h

表 3-75 USBホスト HMSCD API関数一覧

関数名	関数概要	ヘッダ
R_usb_hmsc_Task	HMSCD タスク	r_usb_hmsc_if.h
R_usb_hmsc_driver_start	HMSC ドライバ起動	r_usb_hmsc_if.h
R_usb_hmsc_ClassCheck	ディスクリプタチェック処理	r_usb_hmsc_if.h
R_usb_hmsc_GetDevSts	HMSCD 動作状態の応答	r_usb_hmsc_if.h
R_usb_hmsc_Read10	READ10 コマンド発行	r_usb_hmsc_if.h
R_usb_hmsc_Write10	WRITE10 コマンド発行	r_usb_hmsc_if.h
R_usb_hmsc_GetMaxUnit	GetMaxLUN リクエスト発行	r_usb_hmsc_if.h
R_usb_hmsc_MassStorageReset	MassStorageReset リクエスト発行	r_usb_hmsc_if.h
R_usb_hmsc_alloc_drvno	ドライブ番号割り当て	r_usb_hmsc_if.h
R_usb_hmsc_free_drvno	ドライブ番号解放	r_usb_hmsc_if.h
R_usb_hmsc_ref_drvno	ドライブ番号参照	r_usb_hmsc_if.h

表 3-76 USBホスト HMSDD API関数一覧

関数名	関数概要	ヘッダ
R_usb_hmsc_StrgDriveTask	マスストレージドライバタスク	r_usb_hmsc_if.h
R_usb_hmsc_StrgDriveSearch	ドライブ情報取得	r_usb_hmsc_if.h
R_usb_hmsc_StrgDriveOpen	ドライブオープン	r_usb_hmsc_if.h
R_usb_hmsc_StrgDriveClose	ドライブクローズ	r_usb_hmsc_if.h
R_usb_hmsc_StrgReadSector	セクタ読み出し	r_usb_hmsc_if.h
R_usb_hmsc_StrgWriteSector	セクタ書き込み	r_usb_hmsc_if.h
R_usb_hmsc_StrgCheckEnd	読み出し／書き込み完了確認	r_usb_hmsc_if.h
R_usb_hmsc_StrgUserCommand	ストレージコマンド発行	r_usb_hmsc_if.h

表 3-77 USBホスト FSI API関数一覧

関数名	関数概要	ヘッダ
disk_status	デバイスの状態取得	diskio.h
disk_initialize	デバイスの初期化	diskio.h
disk_read	データの読み出し	diskio.h
disk_write	データの書き込み	diskio.h
disk_ioctl	その他のデバイス制御	diskio.h
get_fattime	日付と時刻の取得	ff.h

3.4　CAN 制御

CAN モジュール（RSCAN）を用いた通信を行うためのドライバを提供します。

3.4.1　CAN モジュールの起動

R_CAN_Open

(1)　概要

CAN モジュールを使用する際に最初に使用する関数です。

(2)　C 言語形式

void R_CAN_Open(uint32_t ch, uint32_t frequency);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	uint32_t	frequency	通信速度

(4)　機能

CAN 通信を開始するための初期設定をします。引数で、使用するチャネル、通信速度を設定します。
チャネルの状態が“未初期化状態の場合、次の処理を行います。

- API 使用する各種変数の初期化
- CAN のモジュールストップ状態の解除
- ポートの入出力設定
- CAN モジュールをグローバル・リセット・モードへ遷移
- 選択チャネルをチャネル・リセット・モードへ遷移
- CAN 通信で使用する CAN レジスタの初期化
- CAN 通信の通信速度の設定

(5)　戻り値

なし

3.4.2　CAN モジュールの停止

R_CAN_Close

(1)　概要

CAN の通信を終了し、CAN モジュールを開放する際に使用する関数です。

(2)　C 言語形式

void R_CAN_Close(uint32_t ch);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　ch	チャネル番号

(4)　機能

CAN 通信を終了するための設定をします。引数で指定したチャネルを無効にします。
本関数では次の処理を行います。

－ CAN のモジュールストップ状態への遷移
－ CAN 割り込みの禁止

再度通信を開始するには、R_CAN_Open ( ) (初期化関数)をコールする必要があります。通信中に強制的に停止した場合、その通信は保証しません。

(5)　戻り値

なし

3.4.3　グローバル・モードの遷移

R_CAN_GlobalControl

(1)　概要

RSCAN モジュール全体の制御を行います。

(2)　C 言語形式

void R_CAN_GlobalControl(uint32_t mode);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　mode	グローバル・モード GL_MODE_OPE：グローバル・動作・モードに遷移する GL_MODE_RESET：グローバル・リセット・モードに遷移する GL_MODE_STOP：グローバル・ストップ・モードに遷移する GL_MODE_TEST：グローバル・テスト・モードに遷移する

(4)　機能

RSCAN モジュール全体の状態を制御するグローバル・モードを引数で設定した モードに遷移させます。

- グローバル・ストップ・モード モジュール全体のクロックを停止させ、低消費電力を実現。
- グローバル・リセット・モード モジュール全体の初期設定を行うためのモード。
- グローバル・テスト・モード　テスト設定（RAM テスト、チャネル間通信テスト）の為のモード
- グローバル・動作・モード モジュール全体を動作可能にします。※通常は、このモードとなります。

(5)　戻り値

なし

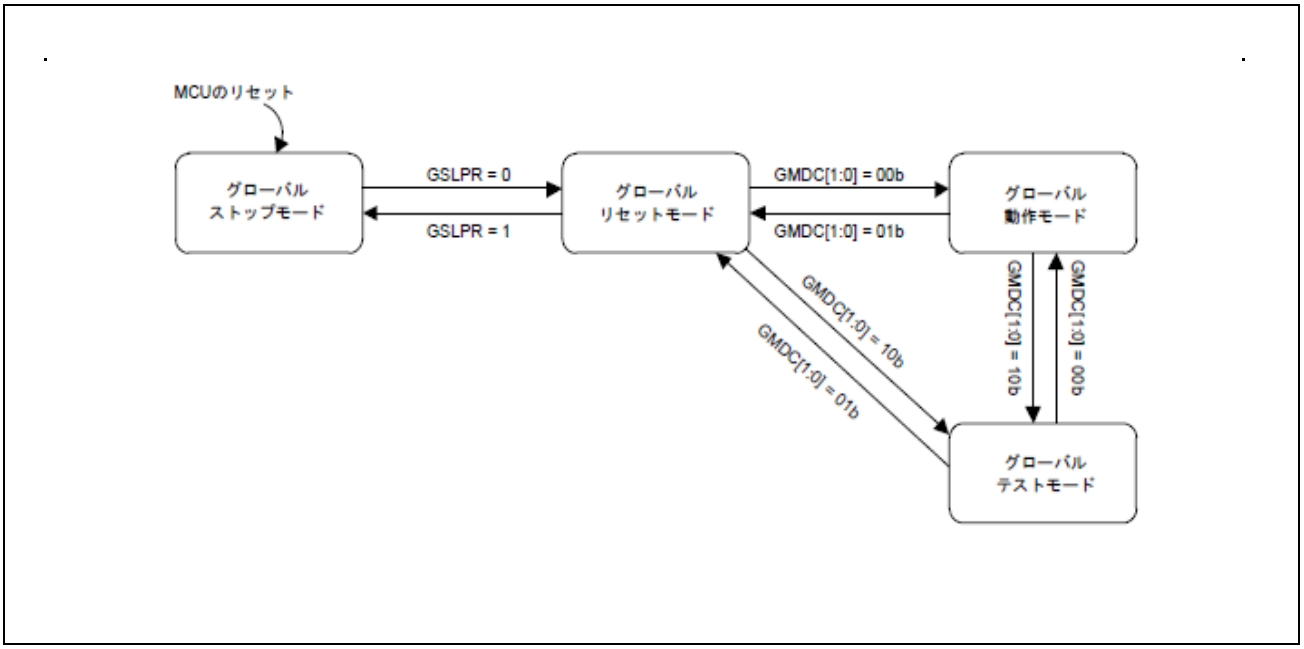


図 3-1 グローバル・モードの遷移図

3.4.4　チャネル・モードの遷移

R_CAN_ChannelControl

(1)　概要

CAN 通信にて使用しているチャネルを制御します。

(2)　C 言語形式

void R_CAN_ChannelControl(uint32_t ch, uint32_t mode);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	uint32_t	mode	チャネル・モード CH_MODE_STOP：チャネル・ストップ・モードへ遷移する CH_MODE_RESET：チャネル・リセット・モードへ遷移する CH_MODE_WAIT：チャネル・待機・モードへ遷移する CH_MODE_COMM：チャネル・通信・モードへ遷移する

(4)　機能

引数で指定したチャネルの状態を設定します。
- チャネル・ストップ・モード チャネルのクロックを停止させるモードです。
- チャネル・リセット・モード チャネルの初期設定を行う為のモードです。
- チャネル・待機・モード　CAN 通信を停止させ、チャネルのテストを許可するモードです。
- チャネル・通信・モード　CAN 通信を行うモードです。※通常は、このモードです。

(5)　戻り値

なし

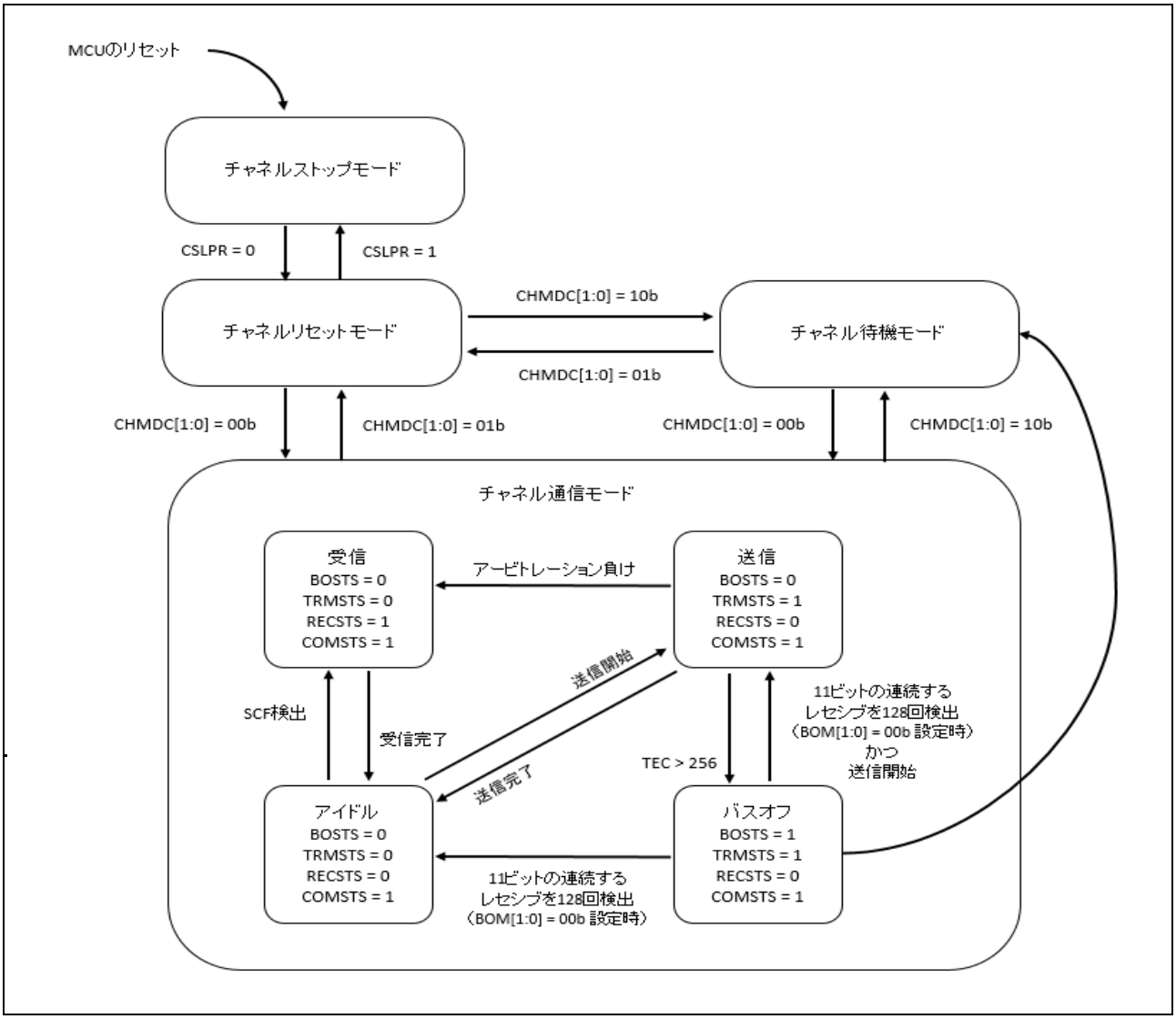


図 3-2 チャネル・モードの状態遷移図

3.4.5　通信速度の設定

R_CAN_SetBitrate

(1)　概要

CAN 通信の通信速度を設定します。

(2)　C 言語形式

void R_CAN_SetBitrate(uint32_t ch, uint32_t frequency);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	uint32_t	frequency	通信速度 BAUD_RATE_1MBPS：1Mbps BAUD_RATE_500KBPS：500Kbps BAUD_RATE_125KBPS：125Kbps

(4)　機能

引数で指定した値を用いて CAN 通信の通信速度を決定します。

EC-1 ユーザーズマニュアル　ハードウェア編 「27.9.1.2　ビットタイミングの設定」および「27.9.1.3 通信速度の設定」を参照して下さい。

(5)　戻り値

なし

3.4.6　送信・受信で使用するバッファの登録

R_CAN_UseBufferEntry

(1)　概要

CAN 通信で使用する各種バッファの情報を登録します。

(2)　C 言語形式

void　R_CAN_UseBufferEntry(can_used_buffer_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	can_used_buffer_t　　* obj	バッファ関連情報構造体のポインタ

(4)　機能

CAN 通信で使用する各種バッファの情報を登録します。

- 送信バッファの番号
- 受信バッファの番号
- 受信 FIFO バッファの番号
- 送受信 FIFO（送信モード）バッファの番号
- 送受信 FIFO（受信モード）バッファの番号
- 送受信 FIFO（送信モード）バッファにリンクさせる送信バッファの番号

(5)　戻り値

なし

3.4.7　受信 FIFO バッファの設定

R_CAN_SetRxFifoBuffer

(1)　概要

受信 FIFO バッファの使用を許可します。

(2)　C 言語形式

void R_CAN_SetRxFifoBuffer(uint32_t ch, can_rfcc_t * obj);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	can_rfcc_t	* obj	受信FIFO バッファ設定情報

(4)　機能

CAN 通信で、受信 FIFO バッファを使ったメッセージ受信を許可します。
メッセージを受信すると、CAN 受信 FIFO 割り込みが発生します。 この関数を呼び出す前に、R_CAN_UseBufferEntry()関数にて、事前に使用する受信 FIFO バッファの番号を登録しておいて下さい。
受信したメッセージは、R_CAN_ReadRxFifo()関数を使って読み出すことができます。

EC-1 ユーザーズマニュアル　ハードウェア編「27.5　受信機能」および「27.9.2.2 FIFO バッファの読み出し手順」を参照して下さい。

(5)　戻り値

なし

3.4.8　送受信 FIFO バッファの設定

R_CAN_SetFifoBuffer

(1)　概要

送受信 FIFO バッファの使用を許可します。

(2)　C 言語形式

void R_CAN_SetFifoBuffer(uint32_t ch, uint32_t mode, can_cfcc_t * obj);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	uint32_t	mode	モード CAN_TX_MODE：送信モード CAN_RX_MODE：受信モード
I	can_cfcc_t	* obj	送受信FIFO バッファ設定情報

(4)　機能

CAN 通信で、送受信 FIFO バッファを使ったメッセージ送信、メッセージ受信を許可します。

－ 送信モード

送受信 FIFO（送信モード）バッファに書き込んだメッセージの送信が完了すると、送信完了割り込み（割り込み要因：送受信 FIFO 送信完了）が発生します。

EC-1 ユーザーズマニュアル　ハードウェア編「27.6　送信機能」および「27.9.3.2 送受信 FIFO バッファからの送信手順」を参照して下さい。

－ 受信モード

送受信 FIFO（受信モード）バッファにメッセージを受信すると、送受信 FIFO 受信完了の割り込みが発生します。 この関数を呼び出す前に、R_CAN_UseBufferEntry()関数にて、事前に使用する送受信 FIFO バッファの番号を登録しておいて下さい。 受信したメッセージは、R_CAN_ReadFifo ()関数を使って読み出すことができます。

EC-1 ユーザーズマニュアル　ハードウェア編「27.5　受信機能」および「27.9.2.2 FIFO バッファの読み出し手順」を参照して下さい。

(5)　戻り値

なし

3.4.9　送受信 FIFO バッファの開放

R_CAN_ReleaseFifoBuffer

(1)　概要

CAN 通信で使用した送受信 FIFO バッファを開放します。

(2)　C 言語形式

void R_CAN_ReleaseFifoBuffer(uint32_t ch, uint32_t mode);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	uint32_t	mode	モード CAN_TX_MODE：送信モード CAN_RX_MODE：受信モード

(4)　機能

CAN 通信で使用した送受信FIFO バッファを開放します。

(5)　戻り値

なし

3.4.10　受信 FIFO バッファの開放

R_CAN_ReleaseRxFifoBuffer

(1)　概要

CAN 通信で使用した受信 FIFO バッファを開放します。

(2)　C 言語形式

void R_CAN_ReleaseRxFifoBuffer(uint32_t ch);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　ch	チャネル番号

(4)　機能

CAN 通信で使用した受信FIFO バッファを開放します。

(5)　戻り値

なし

3.4.11　送信バッファ、受信バッファの開放

R_CAN_ReleaseBuffer

(1)　概要

CAN 通信で使用したバッファを開放します。

(2)　C 言語形式

void R_CAN_ReleaseBuffer(uint32_t ch, uint32_t mode);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	uint32_t	mode	モード CAN_TX_MODE：送信モード CAN_RX_MODE：受信モード

(4)　機能

CAN 通信で使用したバッファを開放します。

(5)　戻り値

なし

3.4.12　送信バッファステータスの読み出し

R_CAN_GetTxBufferStatus

(1)　概要

送信バッファの状態を読み出します。

(2)　C 言語形式

uint32_t R_CAN_GetTxBufferStatus(uint32_t ch);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　ch	チャネル番号

(4)　機能

送信バッファの状態を読み出します。

(5)　戻り値

戻り値	意味
0	送信中で無い
1	送信中

3.4.13　送信メッセージを送信バッファに書き込む

R_CAN_WriteBuffer

(1)　概要

送信バッファにメッセージを書き込みます。

(2)　C 言語形式

void R_CAN_WriteBuffer(uint32_t ch, can_tx_message_t * msg);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	can_tx_message_t	* msg	送信メッセージ情報

(4)　機能

送信メッセージを送信バッファに書き込む処理です。

送信するメッセージの ID、データフォーマット、送信されるメッセージのデータ長、ラベル情報、送信データを can_tx_message_t 構造体に格納し、本関数の引数 に渡します。

EC-1 ユーザーズマニュアル　ハードウェア編「27.6　送信機能」および「27.9.3.1 送信バッファからの送信手順」を参照して下さい。

(5)　戻り値

なし

3.4.14　送受信 FIFO バッファステータスの読み出し

R_CAN_GetFifoStatus

(1)　概要

送受信 FIFO バッファの状態を読み出します。

(2)　C 言語形式

uint32_t R_CAN_GetFifoStatus(uint32_t ch, uint32_t mode);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	uint32_t	mode	モード CAN_TX_MODE：送信モード CAN_RX_MODE：受信モード

(4)　機能

送受信FIFO バッファの状態を読み出します。

(5)　戻り値

戻り値	意味
0	送受信 FIFO バッファフルではない
1	送受信 FIFO バッファフル

3.4.15　送信メッセージを送受信 FIFO バッファに書き込む

R_CAN_WriteFifo

(1)　概要

送受信FIFO（送信モード）バッファにメッセージを書き込みます。

(2)　C 言語形式

void R_CAN_WriteFifo(uint32_t ch, can_tx_message_t * msg);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　　ch	チャネル番号
I	can_tx_message_t　　* msg	送信メッセージ情報

(4)　機能

送信メッセージを送受信 FIFO（送信モード）バッファに書き込む処理です。

送信するメッセージの ID、データフォーマット、送信されるメッセージのデータ長、ラベル情報、送信データを can_tx_message_t 構造体に格納し、本関数の引数 に渡します。

EC-1 ユーザーズマニュアル　ハードウェア編「27.6　送信機能」および「27.9.3.2 送受信 FIFO バッファからの送信手順」を参照して下さい。

(5)　戻り値

なし

3.4.16　送信開始

R_CAN_Tx

(1)　概要

CAN 通信の送信を開始します。

(2)　C 言語形式

void R_CAN_Tx(uint32_t ch, can_tx_message_t * msg);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t ch	チャネル番号
I	can_tx_message_t * msg	送信メッセージ情報

(4)　機能

CAN 通信の送信を開始する処理です。
－ 送信バッファを用いた送信 送信バッファの送信要求ビットを”1”（送信を要求する）にします。
－ 送受信 FIFO（送信モード）バッファを用いた送信
送受信 FIFO バッファ許可ビットを”1”（送受信 FIFO バッファを使用する） にします。

(5)　戻り値

なし

3.4.17　受信設定

R_CAN_RxSet

(1)　概要

受信を許可します。

(2)　C 言語形式

void R_CAN_RxSet(uint32_t ch, can_rx_rule_t * rule);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t ch	チャネル番号
I	can_rx_rule_t * rule	受信ルール情報

(4)　機能

メッセージを受信するための受信ルールを設定する処理です。
受信ルール情報を can_rx_rule_t 構造体に格納し、本関数の引数に渡します。

EC-1 ユーザーズマニュアル　ハードウェア編「27.5.1　受信ルールテーブルを用いたデータ処理」および「27.9.1.4 受信ルールの設定」を参照して下さい。

(5)　戻り値

なし

3.4.18　受信バッファから受信メッセージの読み出し

R_CAN_ReadBuff

(1)　概要

受信したメッセージを受信バッファから読み出します。

(2)　C 言語形式

uint32_t R_CAN_ReadBuff(uint32_t ch, can_rx_message_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　ch	チャネル番号
I	can_rx_message_t　* obj	受信メッセージ格納領域へのポインタ

(4)　機能

受信メッセージを受信バッファから読み出す処理です。

EC-1 ユーザーズマニュアル　ハードウェア編「27.5　受信機能」および「27.9.2.1 受信バッファの読み出し手順」を参照して下さい。

(5)　戻り値

戻り値	意味
CAN_OK	受信バッファからの読み出し正常
CAN_EMPTY	受信バッファに新しいメッセージがない

3.4.19　受信 FIFO バッファから受信メッセージの読み出し

R_CAN_ReadRxFifo

(1)　概要

受信したメッセージを受信FIFO バッファから読み出します。

(2)　C 言語形式

uint32_t R_CAN_ReadRxFifo(can_rx_message_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	can_rx_message_t * obj	受信メッセージ格納領域へのポインタ

(4)　機能

受信メッセージを受信 FIFO バッファから読み出す処理です。

EC-1 ユーザーズマニュアル　ハードウェア編「27.5　受信機能」および「27.9.2.2 FIFO バッファの読み出し手順」を参照して下さい。

(5)　戻り値

戻り値	意味
CAN_OK	受信FIFO バッファからの読み出し正常
CAN_EMPTY	受信FIFO バッファに未読メッセージなし（バッファ空）
CAM_LOST	FIFO メッセージ・ロスト

3.4.20　送受信 FIFO バッファから受信メッセージの読み出し

R_CAN_ReadFifo

(1)　概要

受信したメッセージを送受信FIFO バッファから読み出します。

(2)　C 言語形式

uint32_t R_CAN_ReadFifo(uint32_t ch, can_rx_message_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t ch	チャネル番号
I	can_rx_message_t * obj	受信メッセージ格納領域へのポインタ

(4)　機能

受信メッセージを送受信FIFO バッファから読み出す処理です。

EC-1 ユーザーズマニュアル　ハードウェア編　27.5　受信機能　27.9.2.2 FIFO バッファの読み出し手順　を参照して下さい。

(5)　戻り値

戻り値	意味
CAN_OK	送受信FIFO バッファからの読み出し正常
CAN_EMPTY	送受信 FIFO バッファに未読メッセージなし（バッファ空）
CAM_LOST	FIFO メッセージ・ロスト

3.4.21　送受信 FIFO バッファの未読メッセージ数の取得

R_CAN_GetFifoMessageNum

(1)　概要

送受信 FIFO バッファの未読メッセージ数の取得。

(2)　C 言語形式

uint32_t R_CAN_GetFifoMessageNum(uint32_t ch);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　ch	チャネル番号

(4)　機能

送受信 FIFO バッファの未読メッセージ数を返します。

EC-1 ユーザーズマニュアル　ハードウェア編「27.5　受信機能」および「27.9.2.2 FIFO バッファの読み出し手順」を参照して下さい。

(5)　戻り値

戻り値	意味
	未読メッセージ数

3.4.22　受信 FIFO バッファの未読メッセージ数の取得

R_CAN_GetRxFifoMessageNum

(1)　概要

受信 FIFO バッファの未読メッセージ数取得。

(2)　C 言語形式

uint32_t R_CAN_GetRxFifoMessageNum(void);

(3)　パラメータ

なし

(4)　機能

受信 FIFO バッファの未読メッセージ数を取得します。

EC-1 ユーザーズマニュアル　ハードウェア編「27.5　受信機能」および「27.9.2.2 FIFO バッファの読み出し手順」を参照して下さい。

(5)　戻り値

戻り値	意味
	未読メッセージ数

3.4.23　テスト設定

R_CAN_SetCommTestMode

(1)　概要

通信テストモードを選択します。

(2)　C 言語形式

void R_CAN_SetCommTestMode(uint32_t ch, uint32_t mode);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	uint32_t	mode	テストモード CH_TEST_STANDARD：標準テストモード CH_TEST_LISTENONLY：リッスンオンリモード CH_TEST_SELF0：セルフテストモード 0（外部ループバックモード） CH_TEST_SELF1：セルフテストモード 1（内部ループ バックモード）

(4)　機能

通信テストモードを選択します。
通信テストモードで、以下の選択が可能です。
－ 標準テストモード
－ リッスンオンリモード
－ セルフテストモード0（外部ループバックモード）
－ セルフテストモード1（内部ループバックモード）

(5)　戻り値

なし

3.4.24　テスト設定を解除しチャネル・通信・モードへ遷移

R_CAN_ResetTestMode

(1)　概要

通信テストのクリア

(2)　C 言語形式

void R_CAN_ResetTestMode(uint32_t ch);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　ch	チャネル番号

(4)　機能

R_CAN_SetCommTestMode()関数で設定した通信テストモードをクリアします。

通信テスト終了後、必ずこの API 関数をコールして下さい。

(5)　戻り値

なし

3.4.25　割り込みハンドラの登録

R_CAN_SetInterruptHandler

(1)　概要

CAN 通信で使用する割り込みハンドラから呼び出されるコールバック関数の登録

(2)　C 言語形式

void R_CAN_SetInterruptHandler(uint32_t ch, can_callback_t * pcallback);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャネル番号
I	can_callback_t	* pcallback	コールバック関数情報

(4)　機能

CAN 通信で使用する割り込みハンドラから呼び出されるコールバック関数を登録します。
CAN 通信の割り込みハンドラは以下のとおりです。

- CAN グローバル・エラー
- CAN0 エラー
- CAN1 エラー
- CAN 受信 FIFO
- CAN0 送受信 FIFO 受信完了
- CAN0 送信
- CAN1 送受信 FIFO 受信完了
- CAN1 送信

* pcallback
コールバック関数情報構造体のポインタ。 この構造体のメンバにコールバック関数名を記述します。使用しない場合は、 NULL として下さい。

(5)　戻り値

なし

3.4.26　CAN モジュール割り込みベクタの割り込み許可・禁止

R_CAN_SetInterruptEnableDisable

(1)　概要

CAN 通信で使用する割り込みハンドラを許可／禁止

(2)　C 言語形式

void R_CAN_SetInterruptEnableDisable(uint32_t enable_disable);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　enable_disable	許可／禁止 CAN_INTR_DISABLE：禁止 CAN_INTR_ENABLE：許可

(4)　機能

CAN 通信で使用する割り込みハンドラの許可／禁止を行います。
CAN 通信の割り込みハンドラは以下のとおりです。

- CAN グローバル・エラー
- CAN0 エラー
- CAN1 エラー
- CAN 受信 FIFO
- CAN0 送受信 FIFO 受信完了
- CAN0 送信
- CAN1 送受信 FIFO 受信完了
- CAN1 送信

(5)　戻り値

なし

3.4.27　割り込み要因の取得

R_CAN_GetInterruptSource

(1)　概要

割り込み要因の取得

(2)　C 言語形式

void R_CAN_GetInterruptSource(can_intr_source_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	can_intr_source_t　　* obj	割り込み要因情報

(4)　機能

各割り込み発生時の割り込み要因情報を返します。
- 状態フラグの設定
- ポートの入出力設定
- I2C 出力ポートの割り当て
- RIIC のモジュールストップ状態の解除
- API で使用する変数の初期化
- RIIC 通信で使用する RIIC レジスタの初期化
- RIIC 割り込みの禁止

* obj
割り込み要因情報構造体のポインタ。 読み出した各割り込みの要因が格納されます。

(5)　戻り値

なし

3.4.28 割り込み要因のクリア

R_CAN_ClearInterruptSource

(1) 概要

割り込み要因情報のクリア

(2) C 言語形式

void R_CAN_ClearInterruptSource(void);

(3) パラメータ

なし

(4) 機能

各割り込みの割り込み要因をクリアします。

(5) 戻り値

なし

3.5 CMT 制御

コンペアマッチタイマ（CMT）機能を制御するためのドライバを提供します。

3.5.1 CMT チャネル初期化処理

R_CMT_Init

(1) 概要

CMT チャネル初期化処理

(2) C 言語形式

int32_t R_CMT_Init(uint32_t channel, uint16_t cks);

(3) パラメータ

I/O	パラメータ		説明
I	uint32_t	channel	初期化する CMT チャネルを指定します。 設定可能範囲（0 ～ 3）
I	uint16_t	cks	CMCNTn カウンタに入力するクロックを選択します。 CMT_CKS_DIVISION_8 : PCLKD / 8 CMT_CKS_DIVISION_32 : PCLKD / 32 CMT_CKS_DIVISION_128 : PCLKD / 128 CMT_CKS_DIVISION_512 : PCLKD / 512

(4) 機能

引数にて指定した CMT チャネルを初期化します。
引数 cks にて指定した値にて、CMCNTn カウンタに入力するクロックを選択します。

(5) 戻り値

戻り値	意味
CMT_SECCESS	成功
CMT_ERR	失敗

3.5.2　周期イベント設定処理

R_CMT_CreatePeriodic

(1)　概要

周期イベント設定処理

(2)　C 言語形式

int32_t R_CMT_CreatePeriodic(uint32_t channel, uint32_t frequency_hz, void (* callback)(uint32_t channnel));

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	channel	周期イベントを設定する CMT チャネルを指定します。 設定可能範囲（0 ～ 3）
I	uint32_t	frequency_hz	イベントを発生させる周期を指定します。（単位：Hz）
I	void	(* callback)(uint32_t channnel)	CMT チャネルのコンペアマッチ割り込み発生時に実行するコールバック関数へのポインタを指定します。

(4)　機能

引数にて指定した周期で CMT チャネルのコンペアマッチ割り込みを発生させます。
CMT チャネルのコンペアマッチタ割り込み発生時、引数 callback にて指定したコールバック関数を実行します。
CMT チャネルのコンペアマッチ割り込み発生後もカウント動作を継続します。
本関数にて開始した周期イベントを停止する場合は、周期イベント停止関数を実行してください。

(5)　戻り値

戻り値	意味
CMT_SECCESS	成功
CMT_ERR	失敗

備考　本関数実行前に、CMT チャネル初期化関数で CMT チャネルを初期化してください。

3.5.3　ワンショットイベント設定処理

R_CMT_CreateOneShot

(1)　概要

ワンショットイベント設定処理

(2)　C 言語形式

int32_t R_CMT_CreateOneShot(uint32_t channel, uint32_t period_us, void (* callback)(uint32_t channnel));

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	channel	周期イベントを設定する CMT チャネルを指定します。 設定可能範囲（0 ～ 3）
I	uint32_t	period_us	イベントを発生させる周期を指定します。（単位：μSecond）
I	void	(* callback)(uint32_t channnel)	CMT チャネルのコンペアマッチ割り込み発生時に実行するコールバック関数へのポインタを指定します。

(4)　機能

引数にて指定した周期で CMT チャネルのコンペアマッチ割り込みを発生させます。
CMT チャネルのコンペアマッチ割り込み発生時、引数 callback にて指定したコールバック関数を実行します。
CMT チャネルのコンペアマッチ割り込み発生後もカウント動作を継続します。
本関数にて開始した周期イベントを停止する場合は、周期イベント停止関数を実行してください。

(5)　戻り値

戻り値	意味
CMT_SECCESS	成功
CMT_ERR	失敗

備考　本関数実行前に、CMT チャネル初期化関数で CMT チャネルを初期化してください。

3.5.4　CMT 動作停止処理

R_CMT_Stop

(1)　概要

CMT 動作停止処理

(2)　C 言語形式

int32_t R_CMT_Stop(uint32_t channel);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　channel	停止する CMT チャネルを指定します。 設定可能範囲（0 ～ 3）

(4)　機能

周期イベント設定処理またはワンショットイベント設定処理にて動作を開始した CMT チャネルの動作を停止します。

(5)　戻り値

戻り値	意味
CMT_SECCESS	成功
CMT_ERR	失敗

備考　本関数実行前に、CMT チャネル初期化関数で CMT チャネルを初期化してください。

3.5.5　CMT 初期設定

userdef_cmt_init

(1)　概要

CMT 初期設定（ユーザ定義）

(2)　C 言語形式

int32_t userdef_cmt_init (uint32_t channel, uint16_t cks);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　channel	CMT チャネルを指定します。 設定可能範囲（0 ～ 3）
I	uint16_t　cks	CMCNTn カウンタに入力するクロックを選択します。 CMT_CKS_DIVISION_8 : PCLKD / 8 CMT_CKS_DIVISION_32 : PCLKD / 32 CMT_CKS_DIVISION_128 : PCLKD / 128 CMT_CKS_DIVISION_512 : PCLKD / 512

(4)　機能

引数にて指定した CMT チャネルを初期化します。
引数 cks にて指定した値にて、CMCNTn カウンタに入力するクロックを選択します。

(5)　戻り値

戻り値	意味
CMT_SECCESS	成功
CMT_ERR	失敗

3.5.6　CMT 周期設定

userdef_cmt_create

(1)　概要

CMT 周期設定（ユーザ定義）

(2)　C 言語形式

int32_t userdef_cmt_create (uint32_t channel, uint32_t frequency_hz, void (* callback)(uint32_t ch), uint32_t mode);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	channel	CMT チャネルを指定します。 設定可能範囲（0 ～ 3）
I	uint32_t	frequency_hz	イベントを発生させる周期を指定します。（単位：Hz）
I	void	(* callback)(uint32_t ch)	CMT チャネルのコンペアマッチ割り込み発生時に実行するコールバック関数へのポインタを指定します。
I	uint32_t	mode	CMTチャネルの動作モード CMT_MODE_PERIODIC：周期イベント設定 CMT_MODE_ONESHOT：ワンショットイベント設定

(4)　機能

チャネルに応じて、イベント周期、割り込み許可の設定を行い、割り込み発生時のコールバック関数を登録します。

(5)　戻り値

戻り値	意味
CMT_SECCESS	成功
CMT_ERR	失敗

備考　本関数実行前に、CMT チャネル初期化関数で CMT チャネルを初期化してください。

3.5.7　CMT 動作停止

userdef_cmt_stop

(1)　概要

CMT 動作停止（ユーザ定義）

(2)　C 言語形式

int32_t userdef_cmt_stop (uint32_t channel);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　channel	対象とする CMT チャネルを指定します。 設定可能範囲（0 ～ 3）

(4)　機能

周期イベント設定処理またはワンショットイベント設定処理にて動作を開始した CMT チャネルの動作を停止します。

(5)　戻り値

戻り値	意味
CMT_SECCESS	成功
CMT_ERR	失敗

備考　本関数実行前に、CMT チャネル初期化関数で CMT チャネルを初期化してください。

3.5.8　CMI 割り込みハンドラ

userdef_cmt_isr_cmi

(1)　概要

CMI 割り込みハンドラ（ユーザ定義）

(2)　C 言語形式

int32_t userdef_cmt_isr_cmi (uint32_t channel);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　channel	CMT チャネルを指定します。 設定可能範囲（0 ～ 3）

(4)　機能

指定の CMT チャネルについてコールバック関数が実行されます。

(5)　戻り値

戻り値	意味
CMT_SECCESS	成功
CMT_ERR	失敗

備考　本関数実行前に、CMT チャネル初期化関数で CMT チャネルを初期化してください。

3.5.9　CMI0 割り込みハンドラ

cmt0_isr

(1)　概要

CMI0 割り込みハンドラ

(2)　C 言語形式

void cmt0_isr (void);

(3)　パラメータ

なし

(4)　機能

チャネル 0 について割り込みが発生した場合の処理（割り込み要因：コンペアマッチ割り込み_ch0）を行います。

(5)　戻り値

なし

3.5.10　CMI1 割り込みハンドラ

cmt1_isr

(1)　概要

CMI1 割り込みハンドラ

(2)　C 言語形式

void cmt1_isr (void);

(3)　パラメータ

なし

(4)　機能

チャネル 1 について割り込みが発生した場合の処理（割り込み要因：コンペアマッチ割り込み_ch1）を行います。

(5)　戻り値

なし

3.5.11　CMI2 割り込みハンドラ

cmt2_isr

(1)　概要

CMI2 割り込みハンドラ

(2)　C 言語形式

void cmt2_isr (void);

(3)　パラメータ

なし

(4)　機能

チャネル 2 について割り込みが発生した場合の処理（割り込み要因：コンペアマッチ割り込み_ch0）を行います。

(5)　戻り値

なし

3.5.12　CMI3 割り込みハンドラ

cmt3_isr

(1)　概要

CMI3 割り込みハンドラ

(2)　C 言語形式

void cmt3_isr (void);

(3)　パラメータ

なし

(4)　機能

チャネル 3 について割り込みが発生した場合の処理（割り込み要因：コンペアマッチ割り込み_ch1）を行います。

(5)　戻り値

なし

3.6 ETHER 制御

EtherCAT の通信設定を行なうためのドライバを提供します。

3.6.1 EtherCAT 初期化処理

EtherCAT_init

(1) 概要

EtherCAT 初期化処理

(2) C 言語形式

void EtherCAT_init(void);

(3) パラメータ

なし

(4) 機能

EtherCAT 通信をする上で必要となる初期化処理を行います。

(5) 戻り値

なし

3.6.2　Ether 割り込み要求初期化

R_ETHER_IRQ_Init

(1)　概要

Ether 割り込み要求初期化

(2)　C 言語形式

int32_t R_ETHER_IRQ_Init(void);

(3)　パラメータ

なし

(4)　機能

Ether 割り込み要求の初期化処理を行います。
- EtherCAT Sync0 割り込み
- EtherCAT Sync1 割り込み
- EtherCAT 割り込み

(5)　戻り値

戻り値	意味
ETHER_SECCESS	成功
ETHER_ERR	失敗

3.6.3　Ether 割り込み要求初期化

ether_irq_init

(1)　概要

Ether 割り込み要求初期化

(2)　C 言語形式

int32_t ether_irq_init(void);

(3)　パラメータ

なし

(4)　機能

Ether 割り込み要求の初期化処理を行います。
- EtherCAT Sync0 割り込み
- EtherCAT Sync1 割り込み
- EtherCAT 割り込み

(5)　戻り値

戻り値	意味
ETHER_SECCESS	成功
ETHER_ERR	失敗

3.6.4　EtherCAT SYNC 信号出力端子 0 割り込みハンドラ

ecat_sync0_isr

(1) 概要

EtherCAT SYNC 信号出力端子 0 割り込みハンドラ

(2) C 言語形式

void ecat_sync0_isr(void);

(3) パラメータ

なし

(4) 機能

割り込み処理（割り込み要因：EtherCAT Sync0 割り込み）を行います。

(5) 戻り値

なし

3.6.5　EtherCAT SYNC 信号出力端子 1 割り込みハンドラ

ecat_sync1_isr

(1)　概要

EtherCAT SYNC 信号出力端子 1 割り込みハンドラ

(2)　C 言語形式

void ecat_sync1_isr(void);

(3)　パラメータ

なし

(4)　機能

割り込み処理（割り込み要因：EtherCAT Sync1 割り込み）を行います。

(5)　戻り値

なし

3.6.6　EtherCAT 割り込みハンドラ

ecat_isr

(1)　概要

EtherCAT 割り込みハンドラ

(2)　C 言語形式

```
void ecat_isr( void );
```

(3)　パラメータ

なし

(4)　機能

割り込み処理（割り込み要因：EtherCAT 割り込み）を行います。

(5)　戻り値

なし

3.7　RSPI 制御

シリアルペリフェラルインタフェース（RSPI）を用いた通信を行うためのドライバを提供します。
RSPIm で使用している m はチャネル番号を表しています。

3.7.1　RSPI 制御初期化処理

R_RSPIm_Create

(1)　概要

RSPI 制御初期化処理

(2)　C 言語形式

void R_RSPIm_Create(uint16_t mode);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　mode	RSPI チャネルmの動作モード _RSPI_MSMODE_MASTER：マスターモード _RSPI_MSMODE_SLAVE：スレーブモード

(4)　機能

チャネル m について RSPI 通信を制御する上で必要となる初期化処理を行います。

(5)　戻り値

なし

3.7.2　RSPI ポート設定初期化処理

R_RSPIm_Pin_Init

(1)　概要

RSPI ポート設定初期化処理

(2)　C 言語形式

void R_RSPIm_Pin_Init(void);

(3)　パラメータ

なし

(4)　機能

チャネル m にて使用するポート設定の初期化処理を行います。

(5)　戻り値

なし

3.7.3　RSPI 通信開始処理

R_RSPIm_Start

(1)　概要

RSPI 通信開始処理

(2)　C 言語形式

void R_RSPIm_Start (void);

(3)　パラメータ

なし

(4)　機能

チャネル m について RSPI 通信を開始します。

(5)　戻り値

なし

3.7.4　RSPI 通信終了処理

R_RSPIm_Stop

(1)　概要

RSPI 通信終了処理

(2)　C 言語形式

void R_RSPIm_Stop (void);

(3)　パラメータ

なし

(4)　機能

チャネル m について RSPI 通信を終了します。

(5)　戻り値

なし

3.7.5　RSPI 通信ステータス取得処理

R_RSPI_GetState

(1)　概要

RSPI 通信ステータス取得処理

(2)　C 言語形式

rspi_state_t R_RSPI_GetState (void);

(3)　パラメータ

なし

(4)　機能

RSPI 通信のステータスを返却します。

(5)　戻り値

戻り値	意味
OVRF：オーバーランエラーフラグ	0：オーバランエラーなし 1：オーバランエラー発生
IDLNF：RSPIアイドルフラグ	0：RSPIがアイドル状態 1：RSPIが転送状態
MODF：モードフォルトエラーフラグ	0：モードフォルトエラーなし 1：モードフォルトエラー発生
PERF：パリティエラーフラグ	0：パリティエラーなし 1：パリティエラー発生
RX_END：受信完了フラグ	0：受信完了していない 1：受信完了

3.7.6　RSPI 通信ステータス初期化処理

R_RSPIm_ClearState

(1)　概要

RSPI 通信ステータス初期化処理

(2)　C 言語形式

void R_RSPIm_ClearState (void);

(3)　パラメータ

なし

(4)　機能

RSPI 通信のステータスを初期化します。

(5)　戻り値

なし

3.7.7　RSPI 送受信開始処理

R_RSPIm_Send_Receive

(1)　概要

RSPI 送受信開始処理

(2)　C 言語形式

MD_STATUS　R_RSPIm_Send_Receive (const uint32_t * tx_buf, uint16_t tx_num, uint32_t * rx_buf);

(3)　パラメータ

I/O	パラメータ	説明
I	const uint32_t * tx_buf	送信するデータを格納したバッファへのポインタ
I	uint16_t tx_num	送受信するデータの総数
O	uint32_t * tx_buf	受信したデータを格納するバッファへのポインタ

(4)　機能

チャネル m にて RSPI 送受信を開始します。

(5)　戻り値

戻り値	意味
MD_OK	正常終了
MD_ARGERROR	引数*tx_num* の指定が不正

備考 1　本 API 関数では，RSPI 送信処理として，引数 tx_buf で指定されたバッファから 1 バイト単位の RSPI 送信を引数 tx_num で指定された回数だけ繰り返し行います。
備考 2　本 API 関数では，RSPI 受信処理として，1 バイト単位の RSPI 受信を引数 tx_num で指定された回数 だけ繰り返し行い，引数 rx_buf で指定されたバッファに格納します。
備考 3　RSPI 送受信を行う際には，本 API 関数の呼び出し以前に R_RSPIm_Start を呼び出す必要があります。

3.7.8　送信完了コールバック関数

r_rspim_callback_transmitend

(1)　概要

送信完了に伴う処理

(2)　C 言語形式

void　r_rspim_callback_transmitend (void);

(3)　パラメータ

なし

(4)　機能

チャネル m にて送信完了に伴う処理を行います。

(5)　戻り値

なし

備考 1　マスターモードのとき、本 API 関数は，r_rspim_idle_interrupt のコールバック・ルーチンとして呼び出されます。
備考 2　スレーブモードのとき、本 API 関数は，，r_rspim_transmit_interrupt で送信バッファに全送信データを格納したときのコールバック・ルーチンとして呼び出されます。

3.7.9　受信完了コールバック関数

r_rspim_callback_receiveend

(1)　概要

受信バッファフル割り込みの発生に伴う処理

(2)　C 言語形式

void　r_rspim_callback_receiveend (void);

(3)　パラメータ

なし

(4)　機能

チャネル m にて受信バッファフル割り込みの発生に伴う処理を行います.

(5)　戻り値

なし

備考　本 API 関数は，r_rspim_receive_interrupt のコールバック・ルーチンとして呼び出されます。

3.7.10　エラーコールバック関数

r_rspim_callback_error

(1)　概要

RSPI エラー割り込みの発生に伴う処理

(2)　C 言語形式

void　r_rspim_callback_error (uint8_t err_type);

(3)　パラメータ

I/O	パラメータ	説明
O	uint8_t　err_type	RSPI エラー割り込みの発生要因（x 部は不定） xxxx00x1B　：オーバランエラーの検出 xxxx01x0B　：モードフォルトエラーの検出 xxxx10x0B　：パリティエラーの検出

(4)　機能

チャネル m にて RSPI エラー割り込みの発生に伴う処理を行います。

(5)　戻り値

なし

備考　本 API 関数は，r_rspim_error_interrupt のコールバック・ルーチンとして呼び出されます。

3.7.11　送信バッファエンプティ割り込みハンドラ

r_rspim_transmit_interrupt

(1)　概要

送信バッファエンプティ割り込み発生に伴う処理

(2)　C 言語形式

void　r_rspim_transmit_interrupt (void);

(3)　パラメータ

なし

(4)　機能

チャネル m にて送信バッファエンプティ割り込みの発生に伴う処理を行います。

(5)　戻り値

なし

備考　本 API 関数は，送信バッファエンプティ割り込みに対応した割り込みハンドラとして呼び出されます。

3.7.12　受信バッファフル割り込みハンドラ

r_rspim_receive_interrupt

(1)　概要

受信バッファフル割り込みの発生に伴う処理

(2)　C 言語形式

void　r_rspim_receive_interrupt (void);

(3)　パラメータ

なし

(4)　機能

チャネル m にて受信バッファフル割り込みの発生に伴う処理を行います。

(5)　戻り値

なし

備考　本 API 関数は，受信バッファ・フル割り込みに対応した割り込みハンドラとして呼び出されます。

3.7.13　RSPI エラー割り込みハンドラ

r_rspim_error_interrupt

(1)　概要

RSPI エラー割り込みの発生に伴う処理

(2)　C 言語形式

void　r_rspim_receive_interrupt (void);

(3)　パラメータ

なし

(4)　機能

チャネル m にて RSPI エラー割り込みの発生に伴う処理を行います。

(5)　戻り値

なし

備考　本 API 関数は，RSPI エラー割り込みに対応した割り込みハンドラとして呼び出されます。

3.7.14　RSPI アイドル割り込みハンドラ

r_rspim_idle_interrupt

(1)　概要

RSPI アイドル割り込みの発生に伴う処理

(2)　C 言語形式

void　r_rspim_idle_interrupt (void);

(3)　パラメータ

なし

(4)　機能

チャネル m にて RSPI アイドル割り込みの発生に伴う処理を行います。

(5)　戻り値

なし

備考　本 API 関数は，RSPI アイドル割り込みに対応した割り込みハンドラとして呼び出されます。

3.8　SCIFA_UART 制御

FIFO 内蔵シリアルコミュニケーションインタフェース（SCIFA）の UART 機能を制御するためのドライバを提供します。

3.8.1　SCIFA 割り込みハンドラ登録

R_SCIFA_SetInterruptHandler

(1)　概要

SCIFA 割り込みハンドラ登録

(2)　C 言語形式

void R_SCIFA_SetInterruptHandler(scifa_callback_t * pcallback);

(3)　パラメータ

I/O	パラメータ	説明
I	scifa_callback_t　　* pcallback	コールバック関数

(4)　機能

SCIFA 割り込みハンドラの登録（割り込み要因：受信 FIFO データフル（RFD））を行います。

(5)　戻り値

なし

3.8.2 SCIFA 受信割り込み有効

R_SCIFA_EnableItrReg

(1) 概要

SCIFA 受信割り込み有効

(2) C 言語形式

void R_SCIFA_EnableItrReg(void);

(3) パラメータ

なし

(4) 機能

SCIFA 受信時の割り込みハンドラを有効とします。

(5) 戻り値

なし

3.8.3 SCIFA 受信割り込み無効

R_SCIFA_ResetItrReg

(1) 概要

SCIFA 受信割り込み無効

(2) C 言語形式

void R_SCIFA_ResetItrReg(void);

(3) パラメータ

なし

(4) 機能

SCIFA 受信時の割り込みハンドラを無効とします。

(5) 戻り値

なし

3.8.4　SCIFA チャネル UART モード初期設定処理

R_SCIFA_UART_Init

(1)　概要

SCIFA 初期設定処理

(2)　C 言語形式

int32_t R_SCIFA_UART_Init(uint32_t channel, uint32_t mode, uint16_t cks, uint8_t brr);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　channel	初期化する SCIFA チャネルを指定します。 設定可能範囲（0 ～ 4）
I	uint32_t　mode	SCIFA の動作モードを指定します。 SCIFA_UART_MODE_R：受信モード SCIFA_UART_MODE_W：送信モード SCIFA_UART_MODE_RW：送受信モード
I	uint16_t　cks	SCIFA のボーレートジェネレータのクロックソースを選択します。 SCIFA_UART_CKS_DIVISION_1：SERICLK SCIFA_UART_CKS_DIVISION_4：SERICLK / 4 SCIFA_UART_CKS_DIVISION_16：SERICLK / 16 SCIFA_UART_CKS_DIVISION_64：SERICLK / 64
I	uint8_t　brr	SCIFA ビットレートレジスタ（BRR）に設定する値を指定します。 設定可能範囲（ハードウェアマニュアルを参照）

(4)　機能

SCIFA を調歩同期式通信モードに初期化し、SCIFA で使用するポート設定を行います。

(5)　戻り値

戻り値	意味
SCIFA_UART_SUCCESS	初期化成功
SCIFA_UART_ERR	引数エラー

備考 1　SCIFA チャネル 0 のみ実装しています。チャネル 1 ～ 4 は未実装です。
備考 2　SCIFA チャネル 0 で使用する端子は以下の通りです。
TXD0：P23
RXD0：P42

3.8.5　SCIFA チャネル UART モード起動処理

R_SCIFA_UART_Open

(1)　概要

SCIFA チャネル UART モード起動処理

(2)　C 言語形式

int32_t R_SCIFA_UART_Open(uint32_t channel, uint32_t mode);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　channel	SCIFA チャネルを指定します。 設定可能範囲（0 ～ 4）
I	uint32_t　mode	SCIFA の動作モードを指定します。 SCIFA_UART_MODE_R：受信モード SCIFA_UART_MODE_W：送信モード SCIFA_UART_MODE_RW：送受信モード

(4)　機能

引数にて指定された動作モード（送信／受信／送受信）で SCIFA を起動し、調歩同期式通信を開始します。

(5)　戻り値

戻り値	意味
SCIFA_UART_SUCCESS	起動成功
SCIFA_UART_ERR	引数エラー

備考　SCIFA チャネル 0 のみ実装しています。チャネル 1 ～ 4 は未実装です。

3.8.6　SCIFA チャネル UART モードデータ受信処理

R_SCIFA_UART_Receive

(1)　概要

データ受信処理

(2)　C 言語形式

int32_t R_SCIFA_UART_Receive(uint32_t channel, uint8_t *data);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　channel	SCIFA チャネルを指定します。 設定可能範囲（0 ～ 4）
I	uint8_t　*data	受信したデータを格納する領域を指定します。

(4)　機能

RS-232C インタフェースによる COM ポート通信で、1 バイトのデータを受信します。

(5)　戻り値

戻り値	意味
SCIFA_UART_SUCCESS	受信成功
SCIFA_UART_ERR	引数エラー
SCIFA_UART_ERR_RECEIVE	受信エラー

備考　SCIFA チャネル 0 のみ実装しています。チャネル 1 ～ 4 は未実装です。

3.8.7　SCIFA チャネル UART モードデータ送信処理

R_SCIFA_UART_Send

(1)　概要

データ送信処理

(2)　C 言語形式

int32_t R_SCIFA_UART_Send(uint32_t channel, uint8_t data);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　channel	SCIFA チャネルを指定します。 設定可能範囲（0 ～ 4）
I	uint8_t　data	送信データを指定します。

(4)　機能

RS-232C インタフェースによる COM ポート通信で、1 バイトのデータを送信します。

(5)　戻り値

戻り値	意味
SCIFA_UART_SUCCESS	送信成功
SCIFA_UART_ERR	引数エラー

備考　SCIFA チャネル 0 のみ実装しています。チャネル 1 ～ 4 は未実装です。

3.8.8　SCIFA チャネル 0UART モード初期設定処理

userdef_scifa0_uart_init

(1)　概要

SCIFA チャネル 0 UART モード初期設定処理（ユーザ定義）

(2)　C 言語形式

void userdef_scifa0_uart_init(uint32_t mode, uint16_t cks, uint8_t brr);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　mode	SCIFA の動作モードを指定します。 SCIFA_UART_MODE_R：受信モード SCIFA_UART_MODE_W：送信モード SCIFA_UART_MODE_RW：送受信モード
I	uint16_t　cks	SCIFA のボーレートジェネレータのクロックソースを選択します。 SCIFA_UART_CKS_DIVISION_1：SERICLK SCIFA_UART_CKS_DIVISION_4：SERICLK / 4 SCIFA_UART_CKS_DIVISION_16：SERICLK / 16 SCIFA_UART_CKS_DIVISION_64：SERICLK / 64
I	uint8_t　brr	SCIFA ビットレートレジスタ（BRR）に設定する値を指定します。 設定可能範囲（ハードウェアマニュアルを参照）

(4)　機能

SCIFA チャネル 0 の UART 機能初期化の設定を行います。

(5)　戻り値

なし

3.8.9　SCIFA チャネル 0UART モード起動処理

userdef_scifa0_uart_open

(1)　概要

SCIFA チャネル 0 UART モード起動処理（ユーザ定義）

(2)　C 言語形式

void userdef_scifa0_uart_open(uint32_t mode);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　mode	SCIFA の動作モードを指定します。 SCIFA_UART_MODE_R：受信モード SCIFA_UART_MODE_W：送信モード SCIFA_UART_MODE_RW：送受信モード

(4)　機能

SCIFA チャネル 0 の UART 機能を開きます。
引数 mode にて送信、受信、または送信/受信モードを指定します。

(5)　戻り値

なし

3.8.10　SCIFA チャネル 0UART モード受信処理

userdef_scifa0_uart_receive

(1)　概要

SCIFA チャネル 0 データ受信処理（ユーザ定義）

(2)　C 言語形式

int32_t userdef_scifa0_uart_receive(uint8_t *data);

(3)　パラメータ

I/O	パラメータ	説明
I	uint8_t　*data	受信したデータを格納する領域を指定します。

(4)　機能

SCIFA チャネル 0 の UART 機能からデータを読み込みます。
受信エラーのとき、受信エラー状態をクリアし、戻り値に受信エラーを設定します。
受信データを取得したとき、引数で指定した領域に受信データを格納します。

(5)　戻り値

戻り値	意味
SCIFA_UART_SUCCESS	受信成功
SCIFA_UART_ERR_RECEIVE	受信エラー

3.8.11　SCIFA チャネル 0UART モード送信処理

userdef_scifa0_uart_send

(1)　概要

SCIFA チャネル 0 データ送信処理（ユーザ定義）

(2)　C 言語形式

int32_t userdef_scifa0_uart_send(uint8_t *data);

(3)　パラメータ

I/O	パラメータ	説明
I	uint8_t　data	送信データを指定します。

(4)　機能

SCIFA チャネル 0 の UART 機能にデータを書き込みます。
引数で指定したデータを送信します。

(5)　戻り値

なし

3.8.12　文字列出力

__write

(1)　概要

文字列出力

(2)　C 言語形式

size_t __write(int handle, const unsigned char * buffer, size_t size);

(3)　パラメータ

I/O	パラメータ		説明
I	int	handle	読み取りの対象となるファイル番号 STDIN　(0) STDOUT (1) STDERR (2)
O	const unsigned char	* buffer	読み出しデータが格納されている領域へのポインタ
I	size_t	size	読み取りバイト数

(4)　機能

UART によるシリアル通信を用いて、引数にて指定されたバッファより 1 バイトずつ読み出されたデータを送信します。

(5)　戻り値

戻り値	意味
>=0	送信文字数
-1	ファイルナンバーエラー

3.8.13　文字列入力

__read

(1)　概要

文字列入力

(2)　C 言語形式

size_t __read(int handle, unsigned char * buffer, size_t size);

(3)　パラメータ

I/O	パラメータ		説明
I	int	handle	読み取りの対象となるファイル番号 STDIN　(0) STDOUT (1) STDERR (2)
O	unsigned char	* buffer	読み出しデータが格納されている領域へのポインタ
I	size_t_t	size	読み込みバイト数

(4)　機能

UART によるシリアル通信を用いて、引数にて指定されたバッファへ 1 バイトずつデータを受信し、読み出されたデータの出力を行います。

(5)　戻り値

戻り値	意味
>0	受信文字数
-1	ファイルナンバー、受信データエラー

3.8.14 入出力初期化(SCIFA チャネル 0 初期化)

IoInitScifa0

(1) 概要

入出力初期化(SCIFA チャネル 0 初期化)

(2) C 言語形式

void IoInitScifa0 (void);

(3) パラメータ

なし

(4) 機能

標準入出力として UART を使用するために SCIFA チャネル 0 の初期化処理を行います。

(5) 戻り値

なし

3.8.15　文字取得

ioGetchar

(1)　概要

文字取得

(2)　C 言語形式

int32_t ioGetchar (void);

(3)　パラメータ

なし

(4)　機能

SCIFA チャネル 0 より 1 文字の受信を行い、データを返します。
文字は、unsigned char 型として取り込まれ、int 型に変換され戻り値となります。
データを受信するまで本 API は待機します。

(5)　戻り値

戻り値	意味
>0	受信バイト数
-1	エラー

3.8.16　文字出力

IoPutchar

(1)　概要

文字出力

(2)　C 言語形式

void IoPutchar (int32_t buffer)

(3)　パラメータ

I/O	パラメータ	説明
I	int32_t　buffer	出力文字

(4)　機能

引数にて指定された文字を unsigned char 型に変換し SCIFA チャネル 0 へ出力します。
送信可能となる状態になるまで本 API は待機し続けます。

(5)　戻り値

なし

3.9　シリアル・フラッシュ ROM 制御

シリアル・フラッシュ ROM を制御するためのドライバを提供します。

3.9.1　シリアル・フラッシュ ROM 制御初期化

R_SFLASH_init

(1)　概要

シリアル・フラッシュ ROM 制御初期化

(2)　C 言語形式

int32_t R_SFLASH_init(void)

(3)　パラメータ

なし

(4)　機能

シリアル・フラッシュ ROM 制御の初期化を行います。

(5)　戻り値

戻り値	意味
SFLASH_ER_OK	成功
SFLASH_ER_PARAM	パラメータエラー

3.9.2　シリアル・フラッシュ ROM のデータ書き込み

R_SFLASH_program

(1)　概要

シリアル・フラッシュ ROM のデータ書き込み

(2)　C 言語形式

int32_t R_SFLASH_program(uint8_t* buf, uint32_t addr, uint32_t size);

(3)　パラメータ

I/O	パラメータ	説明
I	uint8_t * buf	書き込みデータ格納バッファ・ポインタ
I	uint32_t addr	書き込み開始アドレス
I	uint32_t size	書き込みデータサイズ

(4)　機能

バッファ内のデータをシリアル･フラッシュ ROM の addr 引数にて指定された領域へ size 分のデータ書き込みを行います。

(5)　戻り値

戻り値	意味
SFLASH_ER_OK	成功
SFLASH_ER_PARAM	パラメータエラー

3.9.3　シリアル・フラッシュ ROM のデータ消去

R_SFLASH_erase

(1)　概要

シリアル・フラッシュ ROM のデータ消去

(2)　C 言語形式

int32_t R_SFLASH_erase(uint32_t addr, uint32_t size)

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　addr	データ消去開始アドレス
I	uint32_t　size	データ消去データサイズ

(4)　機能

バッファ内のデータをシリアル･フラッシュ ROM の addr 引数にて指定された領域から size 分の領域を含む消去可能サイズ単位でデータを消去します。

(5)　戻り値

戻り値	意味
SFLASH_ER_OK	成功
SFLASH_ER_PARAM	パラメータエラー

3.10　USB ファンクション制御

USB ファンクションを制御するためのドライバを提供します。

3.10.1　USB モジュールの初期設定

R_USB_Open

(1)　概要

USB モジュール初期設定

(2)　C 言語形式

void R_USB_Open(void)

(3)　パラメータ

なし

(4)　機能

USB モジュールが使用する端子設定と USB モジュールの初期設定を行います。
本 API を呼び出すことで、USB 機能の使用が可能となります。

(5)　戻り値

なし

備考 1　本 API の呼び出しは、R_usb_pstd_DriveRegistration()呼び出し後に行ってください。
備考 2　本 API は初期設定時にユーザアプリケーションで一度だけ呼び出してください。

使用例

```
void usb_smp_task()
{
        USB_PCDREG_t driver;
                :
        R_usb_pstd_DriverRegistration(&driver);
        R_USB_Open();    /* Start USB module */
                :
}
```

3.10.2　データ転送実行要求

R_usb_pstd_TransferStart

(1)　概要

データ転送実行要求

(2)　C 言語形式

USB_ER_t R_usb_pstd_TransferStart(USB_UTR_t *ptr)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_UTR_T　*ptr	USB 通信構造体

(4)　機能

本 API は、ペリフェラルコントロールドライバ（PCD）にデータ転送の要求を行ない、要求を受けた PCD は、引数 ptr(USB_UTR_t 構造体)に設定されている転送情報をもとにデータ転送処理を行います。

USB_UTR_t 構造体には以下の情報を設定してください。

uint16_t keyword パイプ番号
USB_UTR_CB_t complete コールバック関数
void* tranadr データ格納バッファの先頭ポインタ
uint32_t tranlen データ転送サイズ
uint16_t status ステータス
uint16_t pipectr 不使用

データ転送終了（指定データサイズ、ショートパケット受信エラー発生）時にコールバック関数が呼び出されます。このコールバック関数の引数（ptr）には、送受信の残りデータ長、ステータス、及び転送終了の情報が設定されています。このコールバック関数については、“7.8.5 PCD コールバック関数”を参照してください。

(5)　戻り値

戻り値	意味
USB_OK	成功
USB_ERROR	失敗
USB_QOVR	指定したパイプがデータ転送処理中

備考 1　ユーザアプリケーションプログラムまたはクラスドライバで本 API を呼び出してください。
備考 2　受信したデータがマックスパケットサイズの n 倍、かつ受信予定サイズに満たない場合は、データ転送の途中であると判断しコールバックが発生しません。

使用例

```
USB_UTR_t usb_smpl_trn_utr[USB_MAXPIPE_NUM + 1];

USB_ER_t usb_smp_task(uint16_t pipe, uint32_t size,uint8_t *table, USB_UTR_CB_t complete)
{
        /* 転送情報設定 */
        usb_smpl_trn_utr[pipe].keyword = pipe;                  /* パイプ番号 */
        usb_smpl_trn_utr[pipe].tranadr = table;                 /* データ格納バッファの先頭ポインタ */
        usb_smpl_trn_utr[pipe].tranlen = size;                  /* データ転送サイズ */
        usb_smpl_trn_utr[pipe].complete = complete;             /* コールバック関数 */
        usb_smpl_trn_utr[pipe].segment = USB_TRAN_END;          /* ステータス */

        /* 転送開始要求 */
        err = R_usb_pstd_TransferStart(&usb_smpl_trn_utr[pipe]);

        return err;
}
```

3.10.3　データ転送強制終了要求

R_usb_pstd_TransferEnd

(1)　概要

データ転送強制終了要求

(2)　C 言語形式

USB_ER_t R_usb_pstd_TransferEnd(uint16_t pipe)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　pipe	パイプ番号

(4)　機能

本 API は、ペリフェラルコントロールドライバ（PCD） にデータ転送の強制終了要求を行ない、要求を受けた PCD は、データ転送強制終了処理を行います。

データ転送強制終了時に、データ転送要求時(R_usb_pstd_TransferStart)に設定したコールバック関数が呼び出されます。

このコールバック関数の引数（ptr）には、送受信の残りデータ長、ステータス及び強制終了の情報が設定されています。

(5)　戻り値

戻り値	意味
USB_OK	成功
USB_ERROR	失敗

備考 1　ユーザアプリケーションプログラムまたはクラスドライバで本 API を呼び出してください。

使用例

```
void usb_smp_task()
{
                :
    /* 転送終了要求 */
    err = R_usb_pstd_TransferEnd(pipe);
                :
}
```

3.10.4　USB デバイスステート変更要求

R_usb_pstd_ChangeDeviceState

(1)　概要

USB デバイスステート変更要求

(2)　C 言語形式

void R_usb_pstd_ChangeDeviceState(uint16_t state, uint16_t keyword, USB_UTR_CB_t complete)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	state	遷移させたいデバイスステート
I	uint16_t	keyword	パイプ番号
I	USB_UTR_CB_t	complete	コールバック関数

(4)　機能

引数 state に以下の値を設定し、本 API をコールすることで USB デバイスステートの変更をペリフェラルコントロールドライバ（PCD） に要求します。

・USB_DO_REMOTEWAKEUP

PCD へのリモートウェイクアップ実行要求を行います。

・USB_DP_ENABLE

PCD への D+ラインのプルアップ要求を行います。

・USB_DP_DISABLE

PCD への D+ラインのプルアップ解除要求を行います。

・USB_DO_STALL

PCD へのストール応答実行要求を行います。

(5)　戻り値

なし

備考 1　ユーザアプリケーションプログラムまたはクラスドライバで本 API を呼び出してください。
備考 2　接続/切断を割り込みで検出した場合、D+ラインのプルアップ/解除は F/W で自動的に行います。
備考 3　パイプ番号、コールバック関数は、ストール応答時（USB_DO_STALL）のみ使用します。

使用例

```
void usb_smp_task()
{
                :
    /* ストール応答実行要求 */
    R_usb_pstd_ChangeDeviceState(USB_DO_STALL, USB_PIPE1, &usb_smp_dummy)
                :
}
```

3.10.5　ペリフェラルデバイスクラスドライバ(PDCD)登録

R_usb_pstd_DriverRegistration

(1)　概要

ペリフェラルデバイスクラスドライバ(PDCD)登録

(2)　C 言語形式

void R_usb_pstd_DriverRegistration(USB_PCDREG_t *registinfo)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_PCDREG_t *registinfo	クラスドライバ構造体

(4)　機能

クラスドライバ構造体に登録したペリフェラルデバイスクラスドライバ（PDCD）の情報をペリフェラルコントロールドライバ（PCD）に登録します。

(5)　戻り値

なし

備考 1　本 API は初期設定時にユーザアプリケーションで一度だけ呼び出してください。
備考 2　登録可能なデバイスは 1 つです。登録する情報は、「表 3-22 USB_PCDREG_t 構造体」を参照してください。

使用例

```
void usb_smp_registration()
{
        USB_PCDREG_t driver;
                :
        /* Pipe Define Table address */
        driver.pipetbl = (uint16_t**)&usb_gpvendor_smpl_epptr;
        /* Device descriptor Table address */
        driver.devicetbl = (uint8_t*)&usb_gpvendor_smpl_DeviceDescriptor;
        /* Qualifier descriptor Table address */
        driver.qualitbl = (uint8_t*)&usb_gpvendor_smpl_QualifierDescriptor;
        /* Configuration descriptor Table address */
        driver.configtbl = (uint8_t**)&usb_gpvendor_smpl_ConPtr;
        /* Other configuration descriptor Table address */
        driver.othertbl = (uint8_t**)&usb_gpvendor_smpl_ConPtrOther;
        /* String descriptor Table address */
        driver.stringtbl = (uint8_t**)&usb_gpvendor_str_ptr;
        /* Device default */
        driver.devdefault = &usb_smpl_devdefault;
        /* Device configured */
        driver.devconfig = &usb_smpl_devconfig;
        /* Device detach */
        driver.devdetach = &usb_smpl_detach;
        /* Device suspend */
        driver.devsuspend = &usb_smpl_dummy_cb;
        /* Device resume */
        driver.devresume = &usb_smpl_dummy_cb;
        /* Interfaced change */
        driver.interface = &usb_smpl_dummy_cb;
        /* Control Transfer */
        driver.ctrltrans = (USB_CB_TRN_t)&usb_smpl_dummy_cb;
        /* Driver registration */
        R_usb_pstd_DriverRegistration(&driver);
}
```

3.10.6　指定したパイプの PID を BUF に設定

R_usb_cstd_SetBuf

(1)　概要

指定したパイプの PID を BUF に設定

(2)　C 言語形式

void R_usb_cstd_SetBuf(uint16_t pipe)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　pipe	パイプ番号（USB_PIPE0 のみ設定可能）

(4)　機能

指定したパイプの PID を BUF に設定します。

(5)　戻り値

なし

備考 1　コントロール転送のセットアップステージ処理で呼び出してください。

使用例

```
void usb_smp_task()
{
                :
    R_usb_cstd_SetBuf(USB_PIPE0);
                :
}
```

3.10.7　指定したパイプの PID を STALL に設定

R_usb_pstd_SetPipeStall

(1)　概要

指定したパイプの PID を STALL に設定

(2)　C 言語形式

void R_usb_pstd_SetPipeStall(uint16_t pipe)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　pipe	パイプ番号（USB_PIPE0 のみ設定可能）

(4)　機能

引数で指定したパイプ番号の PID を STALL に設定します。

(5)　戻り値

なし

備考 1　コントロール転送のセットアップステージ/ステータスステージ処理で呼び出してください。

使用例

```
void usb_smp_task()
{
                :
        R_usb_pstd_SetPipeStall(pipe);
                :
}
```

3.10.8　コントロール IN 転送用データ転送実行要求

R_usb_pstd_ControlRead

(1)　概要

コントロール IN 転送用データ転送実行要求

(2)　C 言語形式

uint16_t R_usb_pstd_ControlRead(uint32_t bsize, uint8_t *table)

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　bsize	送信データバッファのサイズ
I	uint8_t　*table	送信データバッファのアドレス

(4)　機能

コントロール IN 転送のデータステージで本 API を呼び出すことで、
ペリフェラルコントロールドライバ（PCD）に対しデータ転送の要求を行ないます。
PCD は引数（*table）が示す領域からデータを読み出した後、H/W の FIFO バッファに書き込み、戻り値（end_flag）に以下の結果を通知します。
・USB_WRITESHRT
データ書き込み終了（ショートパケットデータ書き込み）
・USB_WRITEEND
データ書き込み終了（継続データなし/データ長"0"のパケット送信）
・USB_WRITING
データ書き込み中（継続データあり）
・USB_FIFOERROR
FIFO アクセスエラー発生

(5)　戻り値

戻り値	意味
end_flag	データ転送結果

備考 1　本 API は、コントロール IN 転送のデータステージで呼び出してください。「7.9.2 ペリフェラルコントロール転送」を参照してください。

使用例

```
uint8_t usb_smp_buf;

void usb_smp_task()
{
                    :
        /* usb_smp_buf にデータを 1byte 読み出す */
        R_usb_pstd_ControlRead((uint32_t)1, (uint8_t*)(&usb_smp_buf));
                    :
}
```

3.10.9　コントロール OUT 転送用データ転送実行要求

R_usb_pstd_ControlWrite

(1)　概要

コントロール OUT 転送用データ転送実行要求

(2)　C 言語形式

void R_usb_pstd_ControlWrite(uint32_t bsize, uint8_t *table)

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　bsize	バッファに書き込むデータのサイズ
I	uint8_t　*table	データを書き込むバッファのアドレス

(4)　機能

コントロール OUT 転送のデータステージで本 API を呼び出すことで、ペリフェラルコントロールドライバ（PCD）に対しデータ転送の要求を行ないます。
PCD は H/W の FIFO バッファからデータを読み出し、引数（*table）が示す領域に書き込みます。

(5)　戻り値

なし

備考 1　本 API は、コントロール OUT 転送のデータステージで呼び出してください。「7.9.2 ペリフェラルコントロール転送」を参照してください。

使用例

```
uint8_t usb_smp_buf;

void usb_smp_task()
{
                :
    /* usb_smp_buf にデータを 1byte 読み出す */
    R_usb_pstd_ControlWrite((uint32_t)1, (uint8_t*)(&usb_smp_buf));
                :
}
```

3.10.10　コントロール転送終了要求

R_usb_pstd_ControlEnd

(1)　概要

コントロール転送終了要求

(2)　C 言語形式

void R_usb_pstd_ControlEnd(uint16_t status)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　status	ステータス

(4)　機能

コントロール転送のステータスステージで本 API を呼び出すことで、ペリフェラルコントロールドライバ（PCD）に対しステータスステージの実行を要求します。
引数（status）に以下のいずれかの値を設定してください。
・USB_CTRL_END
ステータスステージを正常に終了させる。
・USB_DATA_ERR
ステータスステージでホストに STALL を返す。

(5)　戻り値

なし

備考 1　本 API は、コントロールイン転送のデータステージで呼び出してください。「7.9.2 ペリフェラルコントロール転送」を参照してください。

使用例

```
uint8_t usb_smp_buf;

void usb_smp_task()
{
                :
    /* usb_smp_buf にデータを 1byte 読み出す */
    R_usb_pstd_ControlEnd(USB_CTRL_END);
                :
}
```

3.10.11　USB 割り込み処理

R_usb_pstd_poll

(1)　概要

USB 割り込み処理

(2)　C 言語形式

void R_usb_pstd_poll(void)

(3)　パラメータ

なし

(4)　機能

USB 割り込み発生の有無を判別し、割り込みが発生していた場合、H/W の制御を行います。

(5)　戻り値

なし

備考 1　R_USB_Open()呼び出し後、本 API を定期的に呼び出してください。

使用例

```
uint8_t usb_smp_buf;

void usb_smp_task()
{
                :
        while(1)
        {
                R_usb_pstd_poll();
        }
                :
}
```

3.10.12　USB 送信処理

R_usb_pcdc_SendData

(1)　概要

USB 送信処理

(2)　C 言語形式

void R_usb_pcdc_SendData(uint8_t *table, uint32_t size, USB_UTR_CB_t complete)

(3)　パラメータ

I/O	パラメータ		説明
I	uint8_t	table	転送データアドレス
I	uint32_t	size	転送サイズ
I	USB_UTR_CB_t	complete	処理完了通利コールバック関数

(4)　機能

指定されたアドレスから転送サイズ分のデータを送信します。
送信完了後、コールバック関数 complete が呼出されます。

(5)　戻り値

なし

備考 1　USB 送信処理結果はコールバック関数の引数で得られます。「表 3-21 USB_UTR_t 構造体表 3-22 USB_PCDREG_t 構造体」を参照してください。

使用例

```
void usb_apl_task( void )
{
        uint8_t send_data[] = {0x01,0x02,0x03,0x04,0x05};          /* USB 送信データ */
        uint32_t size = 5;                                         /* USB 送信データ数 */

        R_usb_pcdc_SendData(send_data, size, &usb_complete);
}

/* USB 送信完了通知用コールバック関数 */
void usb_complete(USB_UTR_t *mess)
{
        /* USB 送信完了時の処理を記述して下さい。 */
}
```

3.10.13　USB 受信処理

R_usb_pcdc_ ReceiveData

(1)　概要

USB 受信処理

(2)　C 言語形式

void R_usb_pcdc_ReceiveData (uint8_t *table, uint32_t size, USB_UTR_CB_t complete)

(3)　パラメータ

I/O	パラメータ		説明
I	uint8_t	table	転送データアドレス
I	uint32_t	size	転送サイズ
I	USB_UTR_CB_t	complete	処理完了通利コールバック関数

(4)　機能

HCD に USB 受信要求を行います。
指定した転送サイズ分のデータ受信完了、またはマックスパケットサイズ未満のデータを受信した場合、コールバック関数 complete が呼出されます。
USB 受信データは、転送データアドレスで指定された領域に格納されます。

(5)　戻り値

なし

備考 1　USB 送信処理結果はコールバック関数の引数で得られます。「表 3-21 USB_UTR_t 構造体表 3-22 USB_PCDREG_t 構造体」を参照してください。

使用例

```
void usb_apl_task( void )
{
        uint8_t receive_data[64];                                   /* USB 受信データ格納領域 */
        uint32_t size = 64;                                         /* USB 受信要求サイズ */

        R_usb_pcdc_ReceiveData(receive_data, size, &usb_complete);
}

/* USB 受信完了通知用コールバック関数 */
void usb_complete(USB_UTR_t *mess)
{
        /* USB 受信完了時の処理を記述して下さい。 */
}
```

3.10.14　クラスノーティフィケーション”SerialState”を送信

R_usb_pcdc_SerialStateNotification

(1)　概要

クラスノーティフィケーション”SerialState”を送信

(2)　C 言語形式

void R_usb_pcdc_SerialStateNotification(USB_SCI_SerialState_t serial_state, USB_UTR_CB_t complete)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_SCI_SerialState_t　serial_state	シリアルステータス
I	USB_UTR_CB_t　complete	処理完了通知コールバック関数

(4)　機能

CDC クラスノーティフィケーション”SerialState”を USB ホストに送信します。
この送信はインタラプト IN 転送を使用します。
送信完了後、コールバック関数 complete が呼出されます。

(5)　戻り値

なし

備考 1　USB_SCI_SerialState_t については「表 3-23 USB_SCI_SerialState_t 構造体」を参照してください。
備考 2　USB 送信処理結果はコールバック関数の引数で得られます。「表 3-21 USB_UTR_t 構造体表 3-22 USB_PCDREG_t 構造体」を参照してください。

使用例

```
void usb_apl_task( void )
{
        USB_SCI_serialState_t serial_state; /* シリアルステータス */

        serial_state.WORD = 0x0000;
        serial_state.bParity = 0x0020; /* D5:パリティエラー */
        R_usb_pcdc_SerialStateNotification(serial_state, &usb_complete);
}

/* シリアルステート送信完了通知用コールバック関数 */
void usb_complete(USB_UTR_t *mess)
{
        /* シリアルステート送信完了時の処理を記述して下さい。 */
}
```

3.10.15　CDC 用コントロール転送処理

R_usb_pcdc_ctrltrans

(1)　概要

CDC 用コントロール転送処理

(2)　C 言語形式

void R_usb_pcdc_ctrltrans (USB_REQUEST_t *preq, uint16_t ctsq)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_REQUEST_t *preq	クラスリクエストメッセージへのポインタ
I	uint16_t ctsq	コントロール転送ステージ情報 USB_CS_IDST：Idle or setup stage USB_CS_RDDS：Control read data stage USB_CS_WRDS：Control write data stage USB_CS_WRND：Control write nodata status stage USB_CS_RDSS：Control read status stage USB_CS_WRSS：Control write status stage USB_CS_SQER：Sequence error

(4)　機能

本 API は USB_PCDREG_t 構造体のメンバ ctrltrans にコールバック関数として登録してください。
リクエストタイプが CDC クラスリクエストの場合、コントロール転送ステージに対応した処理を呼び出します。

(5)　戻り値

なし

使用例

```
void pcdc_init(void)
{
        USB_PCDREG_t driver;

        driver.ctrltrans = &R_usb_pcdc_ctrltrans;
        R_usb_pstd_DriverRegistration(&driver);
}
```

3.11　WDTA 制御

WDTA を制御するためのドライバを提供します。

3.11.1　WDT オープン

R_WDT_Open

(1)　概要

WDT オープン

(2)　C 言語形式

wdt_err_t R_WDT_Open (uint16_t channel, void * const p_cfg)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　channel	WDT チャネルを指定します。 設定可能範囲（0 のみ）
I	void * const　p_cfg	WDTA 関連レジスタに設定するデータ群を格納したポインタです。 タイムアウト時間 WDT_TIMEOUT_1024 WDT_TIMEOUT_4096 WDT_TIMEOUT_8192 WDT_TIMEOUT_16384 クロック分周比 WDT_CLOCK __ DIV_4 WDT_CLOCK __ DIV_64 WDT_CLOCK __ DIV_128 WDT_CLOCK __ DIV_512 WDT_CLOCK __ DIV_2048 WDT_CLOCK __ DIV_8192 ウィンドウ終了 WDT_WINDOW_END_75 WDT_WINDOW_END_50 WDT_WINDOW_END_25 WDT_WINDOW_END_0 ウィンドウ開始 WDT_WINDOW_START_25 WDT_WINDOW_START_50 WDT_WINDOW_START_75 WDT_WINDOW_START_100 ECM エラー通知 WDT_ERROR_ENABLE WDT_ERROR_DISABLE

(4)　機能

指定チャネルの WDTA 関連レジスタを初期化し、WDT カウンタのオプションを設定します。
「図 8-3 WDT オープン処理」にフローチャートを記載しています。

(5)　戻り値

戻り値	意味
WDT_SUCCESS	WDT が初期化されました
WDT_ERR_OPEN_IGNORED	モジュールはすでに開かれています
WDT_ERR_INVALID_ARG	p_cfg 構造体の要素に無効な値が含まれています
WDT_ERR_NULL_PTR	p_cfg ポインタがNULL です

備考 1　r_wdt_config.h で定義される WDT_CFG_PARAM_CHECKING_ENABLE を 1 とすることで、引数パラメータのチェック処理を有効にします。

3.11.2　WDT コントロール

R_WDT_Control

(1)　概要

WDT コントロール

(2)　C 言語形式

wdt_err_t R_WDT_Control(uint16_t channel, wdt_cmd_t const cmd, uint16_t * p_status)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	channel	WDT チャネルを指定します。 設定可能範囲（0 のみ）
I	wdt_cmd_t const	cmd	実行されるコマンドを指定します。 WDT_CMD_GET_STATUS WDT_CMD_REFRESH_COUNTING
I	uint16_t *	p_status	カウンタとステータスフラグの格納位置へのポインタ

(4)　機能

指定したチャネルの WDT の状態読み出し、WDT のダウンカウンタをリフレッシュします。
「図 8-4 WDT コントロール処理」にフローチャートを記載しています。

(5)　戻り値

戻り値	意味
WDT_SUCCESS	コマンドは正常に完了しました
WDT_ERR_INVALID_ARG	引数の値が無効です
WDT_ERR_NULL_PTR	p_status がNULLです
WDT_ERR_NOT_OPEND	Open が読み出されていません

備考 1　r_wdt_config.h で定義される WDT_CFG_PARAM_CHECKING_ENABLE を 1 とすることで、引数パラメータのチェック処理を有効にします。

3.12　IWDTA 制御

IWDTA を制御するためのドライバを提供します。

3.12.1　IWDT オープン

R_IWDT_Open

(1)　概要

IWDT オープン

(2)　C 言語形式

iwdt_err_t R_IWDT_Open (void * const p_cfg)

(3)　パラメータ

I/O	パラメータ	説明
I	void * const　p_cfg	IWDTA 関連レジスタに設定するデータ群を格納したポインタです。 タイムアウト時間 IWDT_TIMEOUT_1024 IWDT_TIMEOUT_4096 IWDT_TIMEOUT_8192 IWDT_TIMEOUT_16384 クロック分周比 IWDT_CLOCK __ DIV_1 IWDT_CLOCK __ DIV_16 IWDT_CLOCK __ DIV_32 IWDT_CLOCK __ DIV_64 IWDT_CLOCK_DIV_128 IWDT_CLOCK_DIV_256 ウィンドウ終了 IWDT_WINDOW_END_75 IWDT_WINDOW_END_50 IWDT_WINDOW_END_25 IWDT_WINDOW_END_0 ウィンドウ開始 IWDT_WINDOW_START_25 IWDT_WINDOW_START_50 IWDT_WINDOW_START_75 IWDT_WINDOW_START_100 ECM エラー通知 IWDT_ERROR_ENABLE IWDT_ERROR_DISABLE

(4)　機能

IWDTA 関連レジスタを初期化し、IWDT カウンタのオプションを設定します。
「図 9-3 IWDT オープン処理」にフローチャートを記載しています。

(5)　戻り値

戻り値	意味
IWDT_SUCCESS	IWDT が初期化されました
IWDT_ERR_OPEN_IGNORED	モジュールはすでに開かれています
IWDT_ERR_INVALID_ARG	p_cfg 構造体の要素に無効な値が含まれています
IWDT_ERR_NULL_PTR	p_cfg ポインタがNULL です

備考 1　r_iwdt_config.h で定義される IWDT_CFG_PARAM_CHECKING_ENABLE を 1 とすることで、引数パラメータのチェック処理を有効にします。

3.12.2　IWDT コントロール

R_IWDT_Control

(1)　概要

IWDT コントロール

(2)　C 言語形式

iwdt_err_t R_IWDT_Control(iwdt_cmd_t const cmd, uint16_t * p_status)

(3)　パラメータ

I/O	パラメータ	説明
I	iwdt_cmd_t　cmd const	実行されるコマンドを指定します。 IWDT_CMD_GET_STATUS IWDT_CMD_REFRESH_COUNTING
I	uint16_t *　p_status	カウンタとステータスフラグの格納位置へのポインタ

(4)　機能

指定したチャネルの IWDT の状態読み出し、IWDT のダウンカウンタをリフレッシュします。
「図 9-4 IWDT コントロール処理」にフローチャートを記載しています。

(5)　戻り値

戻り値	意味
IWDT_SUCCESS	コマンドは正常に完了しました
IWDT_ERR_INVALID_ARG	引数の値が無効です
IWDT_ERR_NULL_PTR	p_status がNULLです
IWDT_ERR_NOT_OPEND	Open が読み出されていません

備考 1　r_iwdt_config.h で定義される IWDT_CFG_PARAM_CHECKING_ENABLE を 1 とすることで、引数パラメータのチェック処理を有効にします。

3.13　RIIC 制御

RIIC を制御するためのドライバを提供します。

3.13.1　RIIC モジュールの起動

R_RIIC_Open

(1)　概要

本モジュールを使用する際に最初に使用する関数です。

(2)　C 言語形式

riic_return_t R_RIIC_Open(riic_info_t * p_riic_info)

(3)　パラメータ

I/O	パラメータ	説明
I	riic_info_t *　p_riic_info	RIIC 通信情報構造体のポインタ。 この構造体のうち、本関数で使用するメンバのみを以下に示します。 riic_ch_dev_status_t dev_sts;　ドライバ呼び出しにより更新あり 詳細は「図 10-6」参照 uint8_t ch_no;

(4)　機能

RIIC の通信を開始するための初期設定をします。引数で指定した RIIC のチャネルを設定します。
チャネルの状態が” 未初期化状態（RIIC_NO_INIT）” の場合、次の処理を行います。
- 状態フラグの設定
- ポートの入出力設定
- I2C 出力ポートの割り当て
- RIIC のモジュールストップ状態の解除
- API で使用する変数の初期化
- RIIC 通信で使用する RIIC レジスタの初期化
- RIIC 割り込みの禁止

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_LOCK_FUNC	他のタスクがAPI をロックしている場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合
RIIC_ERR_OTHER	現状態に該当しない不正なイベントが発生した場合

使用例

```
volatile riic_return_t ret;
riic_info_t iic_info_m;

iic_info_m.dev_sts = 0;
iic_info_m.ch_no = 0;

ret = R_RIIC_Open(&iic_info_m);
```

3.13.2　RIIC マスタ送信の開始

R_RIIC_MasterSend

(1)　概要

マスタデバイスとして、送信を開始する際に使用する関数です。

(2)　C 言語形式

riic_return_t R_RIIC_MasterSend(riic_info_t * p_riic_info)

(3)　パラメータ

I/O	パラメータ	説明
I	riic_info_t *　p_riic_info	RIIC 通信情報構造体のポインタ。 この構造体のうち、本関数で使用するメンバのみを以下に示します。 riic_ch_dev_status_t dev_sts;　ドライバ呼び出しにより更新あり 詳細は「図 10-6」参照 uint8_t ch_no; riic_callback callbackfunc; uint32_t cnt2nd;　送信パターン 1、2 のみ実行時、更新あり uint32_t cnt1st;　送信パターン 1 のみ実行時、更新あり uint8_t * p_data2nd; uint8_t * p_data1st; uint8_t * p_slv_adr;

(4)　機能

RIIC のマスタ送信を開始します。引数で指定した RIIC のチャネル、送信パターンで送信します。

チャネルの状態が“ アイドル状態”（RIIC_IDLE、RIIC_FINISH、RIIC_NACK）の場合、次の処理を行います。

- 状態フラグの設定
- API で使用する変数の初期化
- RIIC 割り込みの許可
- スタートコンディションの生成

引数によって、送信パターン（4 パターンあります）を変更できます。

各送信パターンの指定方法および引数の設定可能範囲は、表 3-78 を参照ください。

また、送信パターンの波形のイメージは図 3-3 ～図 3-6 を参照ください。

スレーブアドレスを設定する際、1 ビット左シフトせずに格納してください。

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合
RIIC_ERR_NO_INIT	初期設定ができていない場合（未初期化状態）
RIIC_ERR_BUS_BUSY	バスビジーの場合
RIIC_ERR_AL	アービトレーションエラーが発生した場合
RIIC_ERR_OTHER	現状態に該当しない不正なイベントが発生した場合

使用例

```
/* for MasterSend(Pattern 1) */
#include "r_riic_rx_if.h"

void CallbackMaster(void);
void main(void);

void main(void)
{
	volatile riic_return_t ret;
	riic_info_t iic_info_m;
	uint8_t addr_eeprom[1]={0x50};
	uint8_t access_addr1[1]={0x00};
	uint8_t mst_send_data[5]={0x81,0x82,0x83,0x84,0x85};

	/* Sets IIC Information for sending pattern 1. */
	iic_info_m.dev_sts = 0;
	iic_info_m.ch_no = 0;
	iic_info_m.callbackfunc = &CallbackMaster;
	iic_info_m.cnt2nd = 3;
	iic_info_m.cnt1st = 1;
	iic_info_m.p_data2nd = mst_send_data;
	iic_info_m.p_data1st = access_addr1;
	iic_info_m.p_slv_adr = addr_eeprom;

	/* RIIC open */
	ret = R_RIIC_Open(&iic_info_m);

	/* RIIC send start */
	ret = R_RIIC_MasterSend(&iic_info_m);
	while(1);
}

void CallbackMaster(void)
{
	/* callback process */
}
```

特記事項

送信パターンごとの引数の設定可能範囲は、下表を参照してください。

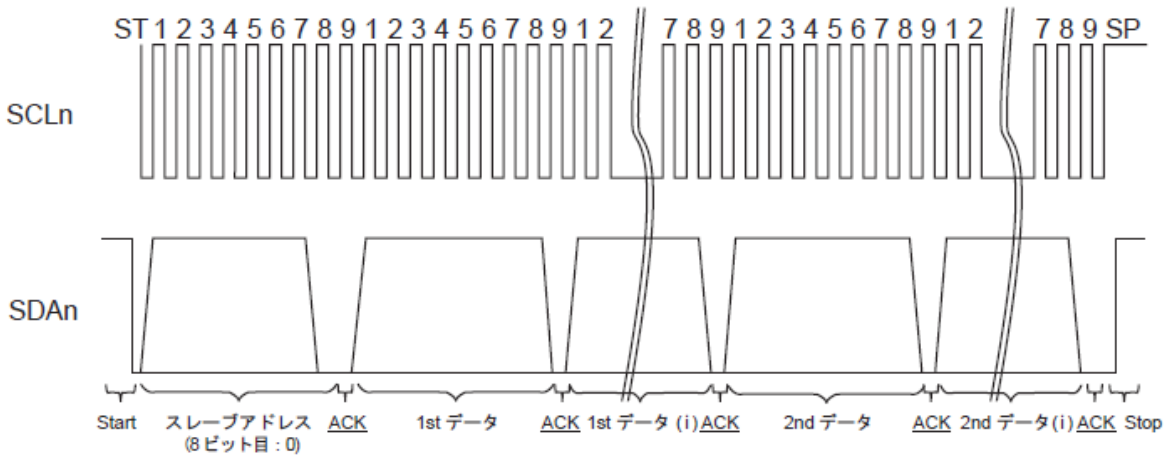
表 3-78 送信パターンごとの引数の設定可能範囲

構造体メンバ	ユーザ設定可能範囲			
	マスタ送信 パターン1	マスタ送信 パターン2	マスタ送信 パターン3	マスタ送信 パターン4
*p_slv_adr	スレーブアドレス バッファポインタ	スレーブアドレス バッファポインタ	スレーブアドレス バッファポインタ	FIT_NO_PTR （注 1）
*p_data1st	[送信用]1st データ バッファポインタ	FIT_NO_PTR 注 1	FIT_NO_PTR 注 1	FIT_NO_PTR 注 1
*p_data2nd	[送信用]2nd データ バッファポインタ	[送信用]2nd データ バッファポインタ	FIT_NO_PTR 注 1	FIT_NO_PTR 注 1
cnt1st	0000 0001h ～ FFFF FFFFh 注 2	0	0	0
cnt2nd	0000 0001h ～ FFFF FFFFh 注 2	0000 0001h ～ FFFF FFFFh 注 2	0	0
callbackfunc	使用する関数名を指 定してください。	使用する関数名を指 定してください。	使用する関数名を指 定してください。	使用する関数名を指 定してください。
ch_no	00h ～ FFh	00h ～ FFh	00h ～FFh	00h ～FFh
dev_sts	デバイス状態 フラグ	デバイス状態 フラグ	デバイス状態 フラグ	デバイス状態 フラグ
rsv1,rsv2	予約領域 （設定無効）	予約領域 （設定無効）	予約領域 （設定無効）	予約領域 （設定無効）

注1.　パターン2、パターン3、パターン4 を使用する場合は、上表のとおり、該当の構造体メンバに“FIT_NO_PTR”を入れてください。
注2.　“0”は設定禁止です。

(a)　パターン 1

マスタデバイスとして、2 つのバッファのデータ（1st データと 2nd データ）をスレーブデバイスへ送信する機能です。
初めにスタートコンディション（ST）を生成し、次にスレーブデバイスのアドレスを送信します。このとき、8 ビット目は転送方向指定ビットになりますので、データ送信時には“0”（Write）を送信します。次に 1st データを送信します。1st データとは、データ送信を行う前に、事前に送信したいデータがある場合に使用します。例えばスレーブデバイスが EEPROM の場合、EEPROM 内部のアドレスを送信することができます。次に 2nd データを送信します。2nd データがスレーブデバイスへ書き込むデータになります。データ送信を開始し、全データの送信が完了すると、ストップコンディション（SP）を生成してバスを解放します。



凡例 :
n: チャネル番号
ST: スタートコンディション生成
SP: ストップコンディション生成
ACK:Acknowledge“0”

注 .　下線のある信号は、スレーブからマスタへのデータ送信です

図 3-3 マスタ送信（パターン1）信号図

(b)　パターン 2

マスタデバイスとして、1 つのバッファのデータ（2nd データ）をスレーブデバイスへ送信する機能です。スタートコンディション（ST）の生成からスレーブデバイスのアドレスを送信まではパターン 1 と同様に動作します。次に 1st データを送信せず、2nd データを送信します。全データの送信が完了すると、ストップコンディション（SP）を生成してバスを解放します。

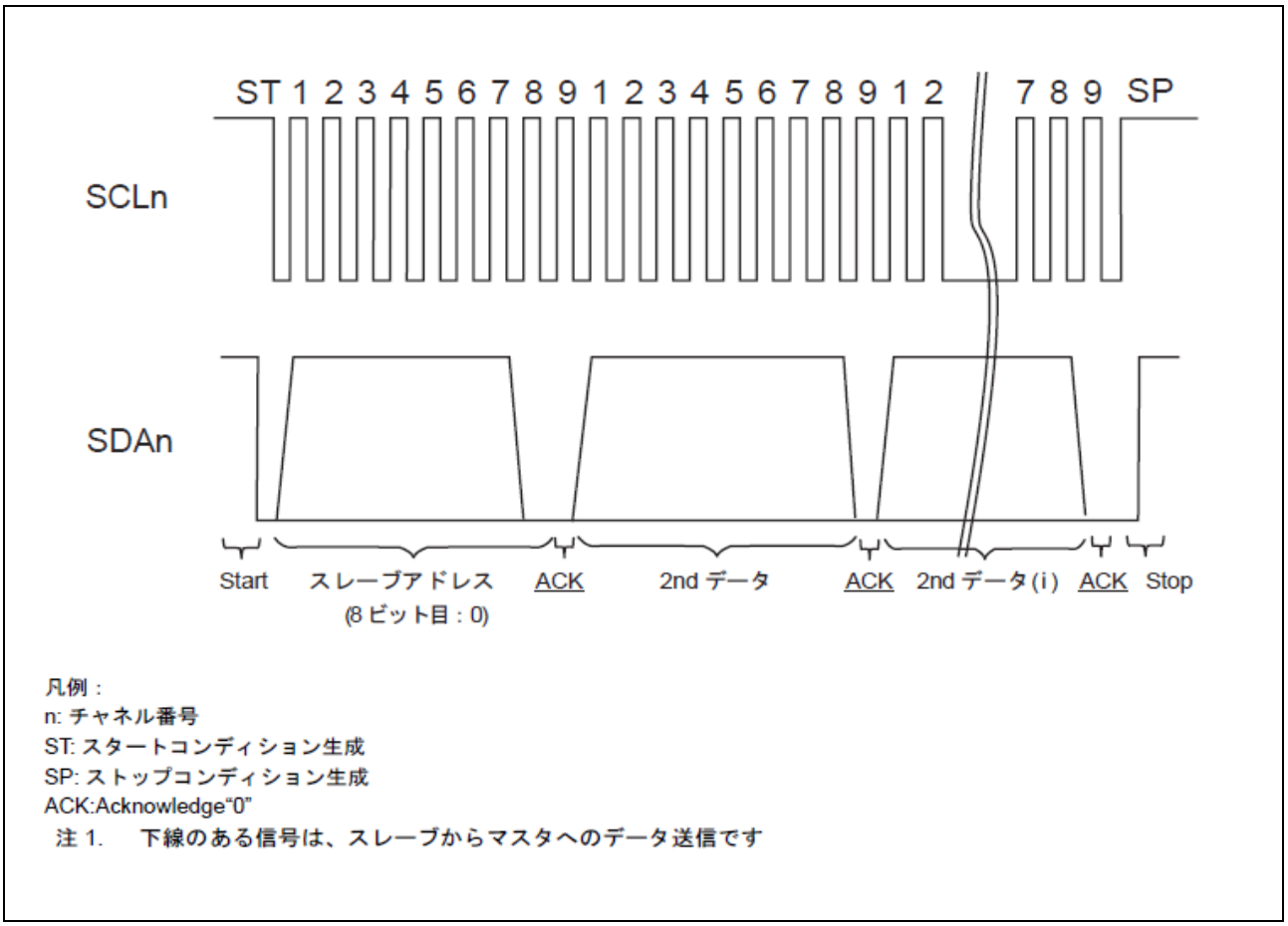
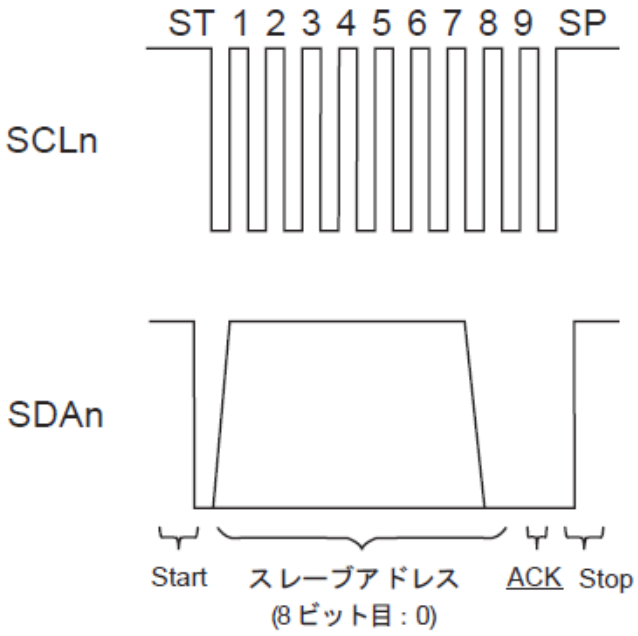


図 3-4 マスタ送信（パターン2）信号図

(c)　パターン 3

マスタデバイスとして、スレーブアドレスのみをスレーブデバイスへ送信する機能です。
スタートコンディション（ST）を生成から、スレーブアドレス送信まではパターン 1 と同様に動作します。
スレーブアドレス送信後、1st データ／ 2nd データを設定していない場合、データ送信は行わず、ストップコンディション（SP）を生成してバスを解放します。
接続されているデバイスを検索する場合や、EEPROM 書き換え状態を確認する Acknowledge Polling を行う際に有効な処理です。



凡例 :
n: チャネル番号
ST: スタートコンディション生成
SP: ストップコンディション生成
ACK:Acknowledge“0”
注 1.　下線のある信号は、スレーブからマスタへのデータ送信です

図 3-5 マスタ送信（パターン3）信号図

(d)　パターン 4

マスタデバイスとして、スタートコンディションとストップコンディションのみをスレーブデバイスへ送信する機能です。
スタートコンディション（ST）を生成後、スレーブアドレスと 1st データ／ 2nd データを設定していない場合、スレーブアドレス送信とデータの送信は行わず、ストップコンディション（SP）を生成してバスを解放します。バス解放のみを行いたい場合に有効な処理です。

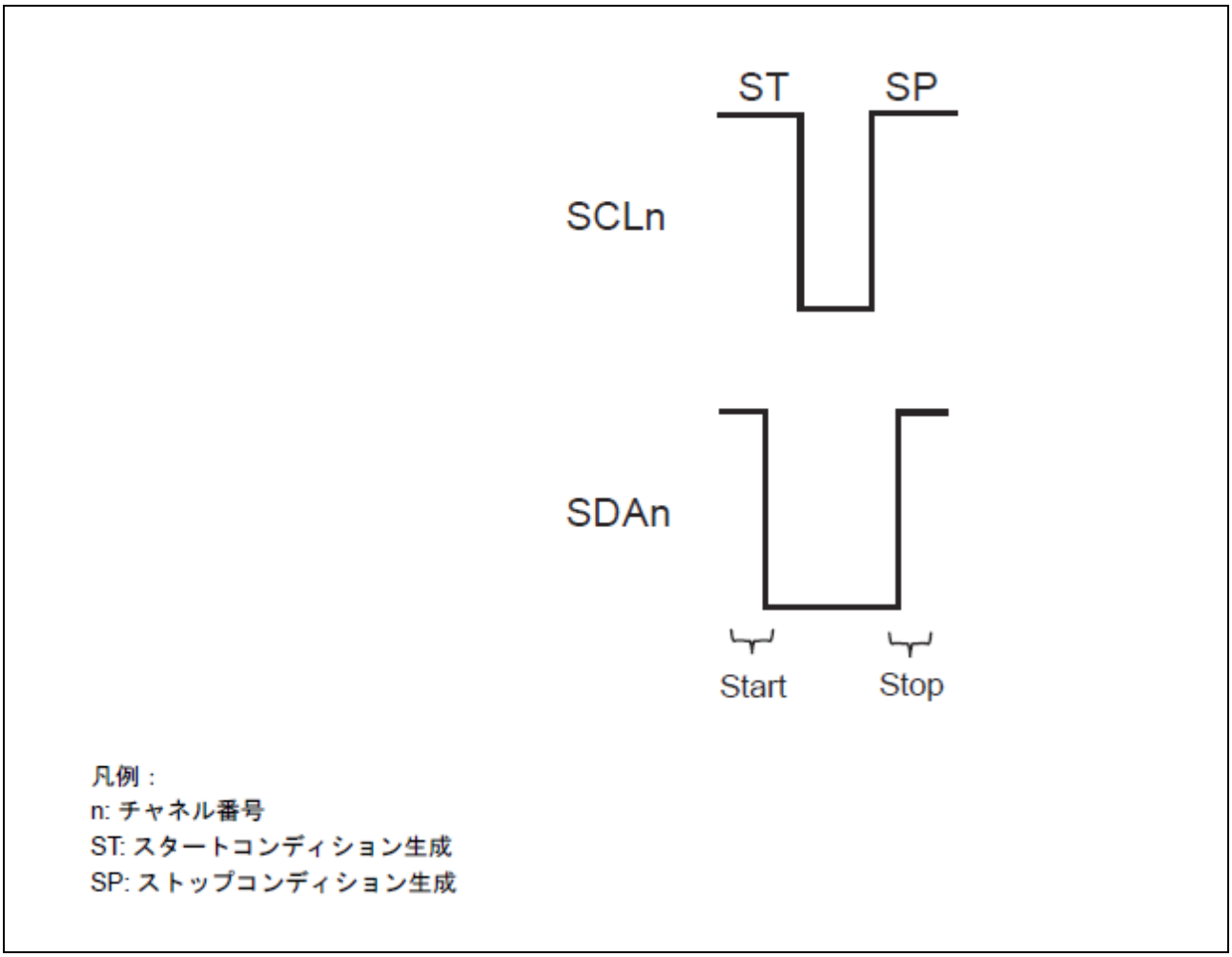


図 3-6 マスタ送信（パターン4）信号図

3.13.3　RIIC マスタ受信の開始

R_RIIC_MasterReceive

(1)　概要

マスタデバイスとして、受信を開始する際に使用する関数です。

(2)　C 言語形式

riic_return_t R_RIIC_MasterReceive(riic_info_t * p_riic_info)

(3)　パラメータ

I/O	パラメータ	説明
I	riic_info_t *　p_riic_info	RIIC 通信情報構造体のポインタ。 この構造体のうち、本関数で使用するメンバのみを以下に示します。 riic_ch_dev_status_t dev_sts;　ドライバ呼び出しにより更新あり 詳細は「図 10-6」参照 uint8_t ch_no; riic_callback callbackfunc;　実行時、更新あり uint32_t cnt2nd;　マスタ複合のみ実行時、更新あり uint32_t cnt1st; uint8_t * p_data2nd; uint8_t * p_data1st; uint8_t * p_slv_adr;

(4)　機能

RIIC のマスタ受信を開始します。引数で指定した RIIC のチャネル、受信パターンで受信します。

チャネルの状態が“ アイドル状態”(RIIC_IDLE、RIIC_FINISH、RIIC_NACK) の場合、次の処理を行います。

- 状態フラグの設定
- API で使用する変数の初期化
- RIIC 割り込みの許可
- スタートコンディションの生成

引数の設定によって、マスタ受信かマスタ複合を選択できます。

マスタ受信およびマスタ複合の指定方法と引数の設定可能範囲は表 3-79 を参照ください。

また、受信パターンの波形イメージは図 3-7、図 3-8 を参照ください。

スレーブアドレスを設定する際、1 ビット左シフトせずに格納してください。

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合
RIIC_ERR_NO_INIT	初期設定ができていない場合（未初期化状態）
RIIC_ERR_BUS_BUSY	バスビジーの場合
RIIC_ERR_AL	アービトレーションエラーが発生した場合
RIIC_ERR_OTHER	現状態に該当しない不正なイベントが発生した場合

使用例

```
/* for MasterReceive(combination mode) */
#include "r_riic_rx_if.h"

void CallbackMaster(void);
void main(void);

void main(void)
{
	volatile riic_return_t ret;
	riic_info_t iic_info_m;
	uint8_t addr_eeprom[1]={0x50};
	uint8_t access_addr1[1]={0x00};
	uint8_t mst_store_area[5]={0xFF,0xFF,0xFF,0xFF,0xFF};

	/* Sets IIC Information. */
	iic_info_m.dev_sts = 0;
	iic_info_m.ch_no = 0;
	iic_info_m.callbackfunc = &CallbackMaster;
	iic_info_m.cnt2nd = 3;
	iic_info_m.cnt1st = 1;
	iic_info_m.p_data2nd = mst_store_area;
	iic_info_m.p_data1st = access_addr1;
	iic_info_m.p_slv_adr = addr_eeprom;

	/* RIIC open */
	ret = R_RIIC_Open(&iic_info_m);

	/* RIIC receive start */
	ret = R_RIIC_MasterReceive(&iic_info_m);

	while(1);
}

void CallbackMaster(void)
{
	/* callback process */
}
```

特記事項

表 3-79 受信パターンごとの引数の設定可能範囲

構造体メンバ	ユーザ設定可能範囲	
	マスタ受信	マスタ複合
*p_slv_adr	スレーブアドレスバッファポインタ	スレーブアドレスバッファポインタ
*p_data1st	未使用（設定無効）	[送信用]1st データバッファポインタ
*p_data2nd	[受信用]2nd データバッファポインタ	[受信用]2nd データバッファポインタ
dev_sts	デバイス状態フラグ	デバイス状態フラグ
cnt1st 注 1	0	0000 0001h ～ FFFF FFFFh 注 2
cnt2nd	0000 0001h ～ FFFF FFFFh 注 2	0000 0001h ～ FFFF FFFFh 注 2
callbackfunc	使用する関数名を指定してください。	使用する関数名を指定してください。
ch_no	00h ～ FFh	00h ～ FFh
rsv1,rsv2	予約領域（設定無効）	予約領域（設定無効）

注1.　1st データが“0”か“0”以外かで受信パターンが決まります。

注2.　“0”は設定禁止です。

(a)　マスタ受信

マスタデバイスとして、スレーブデバイスからデータを受信する機能です。
初めにスタートコンディション（ST）を生成し、次にスレーブデバイスのアドレスを送信します。このとき、8 ビット目は転送方向指定ビットになりますので、データ受信時には“1”（Read）を送信します。次にデータ受信を開始します。受信中は、1 バイト受信するごとに ACK を送信しますが、最終データ時のみ NACK を送信し、スレーブデバイスへ受信処理が完了したことを通知します。全データの受信が完了すると、ストップコンディション（SP）を生成してバスを解放します。

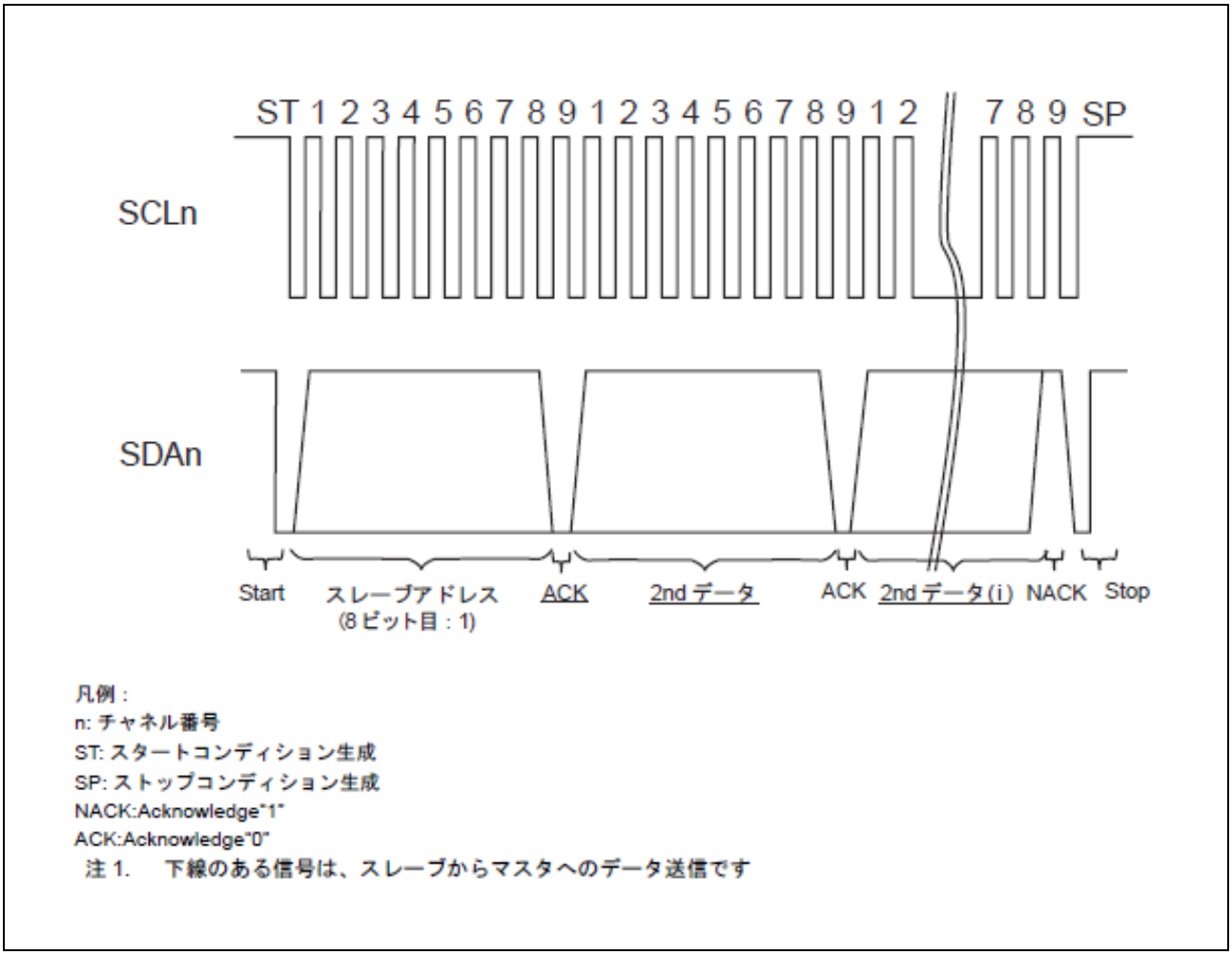
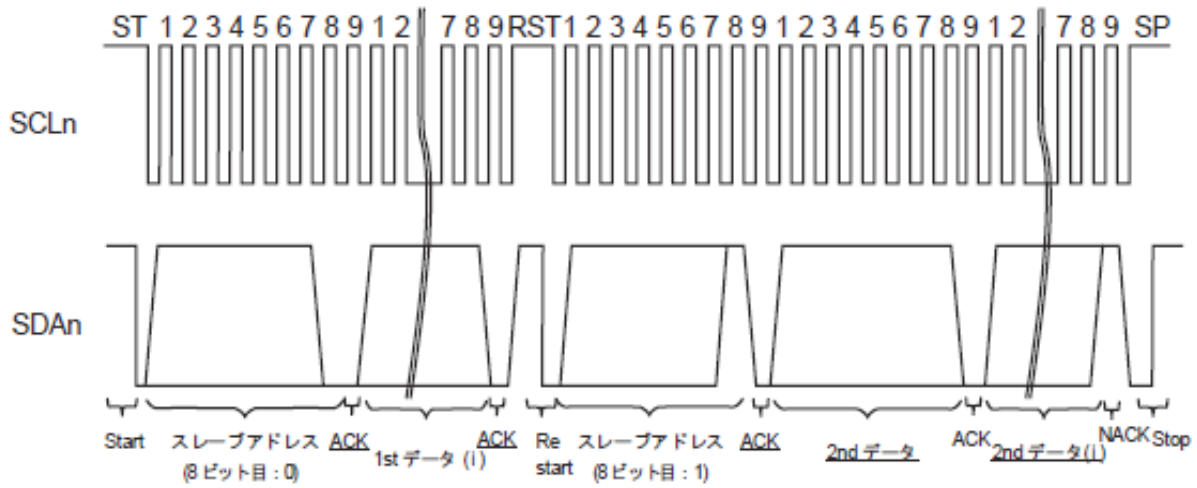


図 3-7 マスタ受信 信号図

(b)　マスタ複合

マスタデバイスとして、スレーブデバイスへデータを送信します。送信完了後、リスタートコンディションを生成し、スレーブデバイスからデータを受信する機能です。
初めにスタートコンディション（ST）を生成し、次にスレーブデバイスのアドレスを送信します。このとき、8 ビット目の転送方向指定ビットには、“0”（Write）を送信します。次に 1st データを送信します。データの送信が完了すると、リスタートコンディション（RST）を生成し、スレーブアドレスを送信します。このとき、転送方向指定ビットには、“1”（Read）を送信します。次にデータ受信を開始します。受信中は、1 バイト受信するごとに ACK を送信しますが、最終データ時のみ NACK を送信し、スレーブデバイスへ受信処理が完了したことを通知します。全データの受信が完了すると、ストップコンディション（SP）を生成してバスを解放します。



凡例 :
n: チャネル番号
ST: スタートコンディション生成
SP: ストップコンディション生成
RST: リスタートコンディション生成
NACK:Acknowledge"1"
ACK:Acknowledge"0"
注 1.　下線のある信号は、スレーブからマスタへのデータ送信です

図 3-8 マスタ複合 信号図

3.13.4　RIIC スレーブ送信・受信状態への遷移

R_RIIC_SlaveTransfer

(1)　概要

スレーブデバイスとして、送信および受信できる状態にする関数です。

(2)　C 言語形式

riic_return_t R_RIIC_SlaveTransfer(riic_info_t * p_riic_info)

(3)　パラメータ

I/O	パラメータ	説明
I	riic_info_t *　p_riic_info	RIIC 通信情報構造体のポインタ。 この構造体のうち、本関数で使用するメンバのみを以下に示します。 riic_ch_dev_status_t dev_sts;　ドライバ呼び出しにより更新あり 詳細は「図 10-6」参照 uint8_t ch_no; riic_callback callbackfunc;　スレーブ受信時のみ更新あり uint32_t cnt2nd;　スレーブ送信時のみ更新あり uint32_t cnt1st; uint8_t * p_data2nd; uint8_t * p_data1st; uint8_t * p_slv_adr;

(4)　機能

RIIC のスレーブ送信、またはスレーブ受信できる状態にします。マスタ通信中に本関数を呼び出した場合は、エラーとなります。引数で指定した RIIC のチャネルを設定します。

チャネルの状態が“ アイドル状態(RIIC_IDLE、RIIC_FINISH、RIIC_NACK)” の場合、次の処理を行います。

- 状態フラグの設定
- API で使用する変数の初期化
- RIIC 通信で使用する RIIC レジスタの初期化
- RIIC 割り込みの許可
- スレーブアドレスの設定、スレーブアドレス一致割り込みの許可

引数の設定によって、スレーブ受信許可状態かスレーブ送信許可状態、またはその両方を選択できます。
引数の設定可能範囲は表 3-80 を参照ください。
また、受信パターンの波形イメージは図 3-9 を、送信パターンの波形イメージは図 3-10 を参照ください。

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合
RIIC_ERR_NO_INIT	初期設定ができていない場合（未初期化状態）
RIIC_ERR_BUS_BUSY	バスビジーの場合
RIIC_ERR_AL	アービトレーションエラーが発生した場合
RIIC_ERR_OTHER	現状態に該当しない不正なイベントが発生した場合

使用例

```
/* for MasterReceive(combination mode) */
#include "r_riic_rx_if.h"

void CallbackMaster(void);
void CallbackSlave(void);
void main(void);

void main(void)
{
   volatile riic_return_t ret;
   riic_info_t iic_info_m;
   riic_info_t iic_info_s;
   uint8_t addr_eeprom[1]={0x50};
   uint8_t access_addr1[1]={0x00};
   uint8_t mst_send_data[5]={0x81,0x82,0x83,0x84,0x85};
   uint8_t slv_send_data[5]={0x71,0x72,0x73,0x74,0x75};
   uint8_t mst_store_area[5]={0xFF,0xFF,0xFF,0xFF,0xFF};
   uint8_t slv_store_area[5]={0xFF,0xFF,0xFF,0xFF,0xFF};

   /* Sets IIC Information for Master Send. */
   iic_info_m.dev_sts = 0;
   iic_info_m.ch_no = 0;
   iic_info_m.callbackfunc = &CallbackMaster;
   iic_info_m.cnt2nd = 3;
   iic_info_m.cnt1st = 1;
   iic_info_m.p_data2nd = mst_store_area;
   iic_info_m.p_data1st = access_addr1;
   iic_info_m.p_slv_adr = addr_eeprom;

   /* Sets IIC Information for Slave Transfer. */
   iic_info_s.dev_sts = 0;
   iic_info_s.ch_no = 0;
   iic_info_s.callbackfunc = &CallbackSlave;
   iic_info_s.cnt2nd = 3;
   iic_info_s.cnt1st = 3;
   iic_info_s.p_data2nd = slv_store_area;
   iic_info_s.p_data1st = slv_send_data;
   iic_info_s.p_slv_adr = (uint8_t*)FIT_NO_PTR;

   /* RIIC open */
   ret = R_RIIC_Open(&iic_info_m);

   /* RIIC slave transfer enable */
   ret = R_RIIC_SlaveTransfer(&iic_info_s);

   /* RIIC master send start */
   ret = R_RIIC_MasterSend(&iic_info_m);
   while(1);
}
void CallbackMaster(void)
{
   /* callback process (master)*/
}
void CallbackSlave(void)
{
   /* callback process (slave)
}
```

特記事項

スレーブ動作パターンごとの引数の設定可能範囲は、下表を参照してください。

表 3-80 スレーブ動作パターンごとの引数の設定可能範囲

構造体メンバ	ユーザ設定可能範囲	
	スレーブ受信	スレーブ送信
*p_slv_adr	未使用（設定無効）	未使用（設定無効）
*p_data1st	（スレーブ送信用）	[送信用]1st データバッファポインタ注
*p_data2nd	[受信用]2nd データバッファポインタ注	（スレーブ受信用）
dev_sts	デバイス状態フラグ	デバイス状態フラグ
cnt1st	（スレーブ送信用）	0000 0001h ～FFFF FFFFh
cnt2nd	0000 0001h ～FFFF FFFFh	（スレーブ受信用）
callbackfunc	使用する関数名を指定してください。	使用する関数名を指定してください。
ch_no	00h ～ FFh	00h ～ FFh
rsv1,rsv2	予約領域（設定無効）	予約領域（設定無効）

注1. スレーブ送信を使用する場合、設定してください。
スレーブ送信を使用しない場合、“FIT_NO_PTR” を設定してください。

注2. スレーブ受信をする場合、設定してください。
スレーブ受信を使用しない場合、“FIT_NO_PTR” を設定してください。

(a)　スレーブ受信

スレーブデバイスとして、マスタデバイスからのデータを受信する機能です。
マスタデバイスが指定するスレーブアドレスが、「r_riic_config_if.h」で設定したスレーブデバイスのスレーブアドレスと一致したとき、スレーブ送受信を開始します。スレーブアドレスの 8 ビット目（転送方向 指定ビット）によって、本モジュールが自動的にスレーブ受信かスレーブ送信かを判断して処理を行います。
マスタデバイスが生成したスタートコンディション（ST）を検出した後に、受信したスレーブアドレス が、自アドレスと一致し、かつスレーブアドレスの 8 ビット目（転送方向指定ビット）が “0”（Write）のとき、スレーブデバイスとして受信動作を開始します。最終データ（RIIC 通信情報構造体に設定された受信 データ数）を受信時は、NACK を返すことでマスタデバイスに必要なデータをすべて受信したことを通知します。

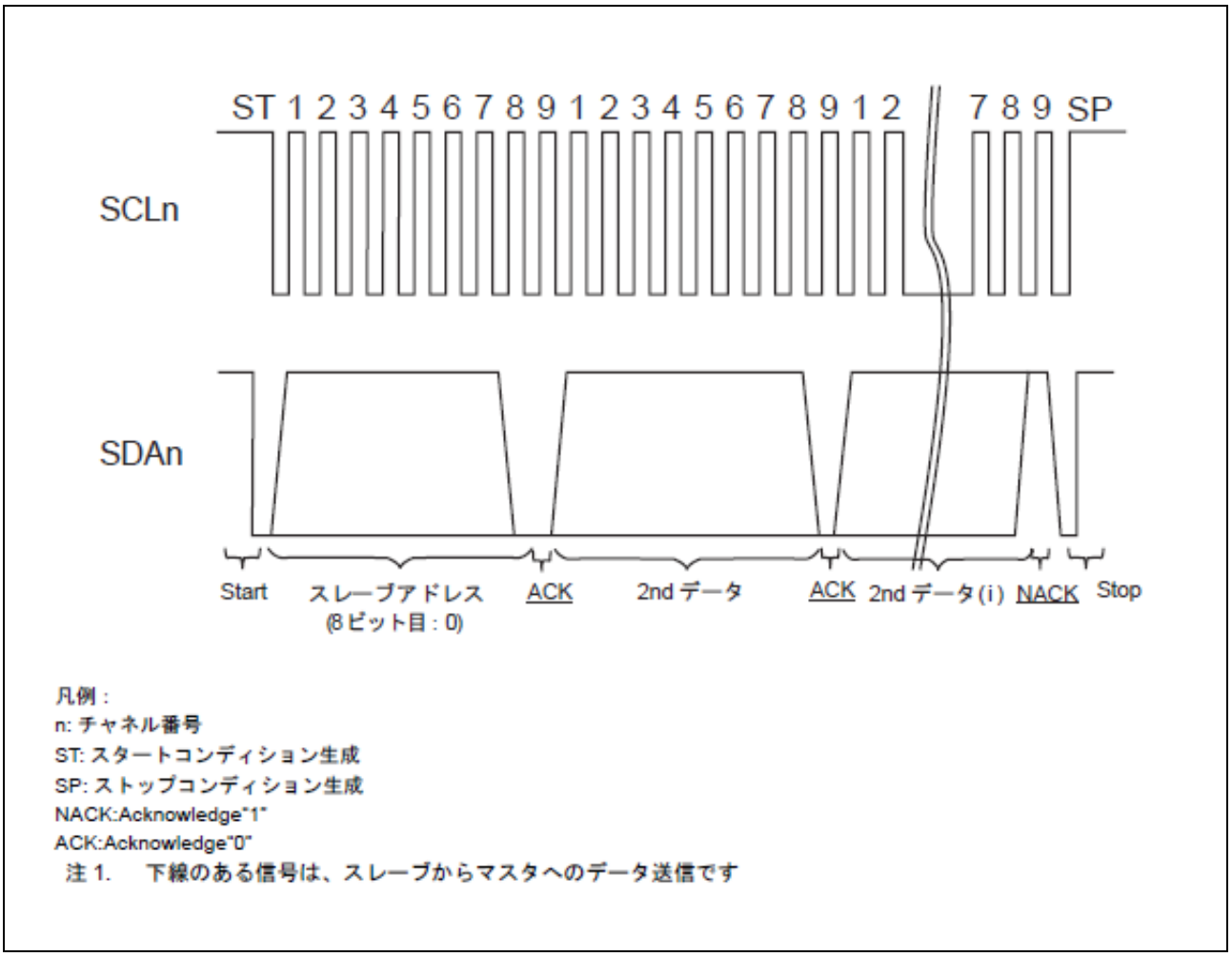


図 3-9 スレーブ受信 信号図

(b)　スレーブ送信

スレーブデバイスとして、マスタデバイスへデータを送信する機能です。 マスタデバイスが指定するスレーブアドレスが、「r_riic_config_if.h」で設定したスレーブデバイスのスレーブアドレスと一致したとき、スレーブ送受信を開始します。スレーブアドレスの 8 ビット目（転送方向 指定ビット）によって、本モジュールが自動的にスレーブ受信かスレーブ送信かを判断して処理を行います。
マスタデバイスからのスタートコンディション（ST）検出後に、受信したスレーブアドレスが、自アドレ スと一致し、かつスレーブアドレスの 8 ビット目（転送方向指定ビット）が “1”（Read）のとき、スレーブ デバイスとして送信動作を開始します。設定したデータ数（RIIC 通信情報構造体に設定した送信データ数） を超える送信データの要求があった場合、“0xFF” をデータとして送信します。ストップコンディション（SP）を検出するまでデータを送信します。

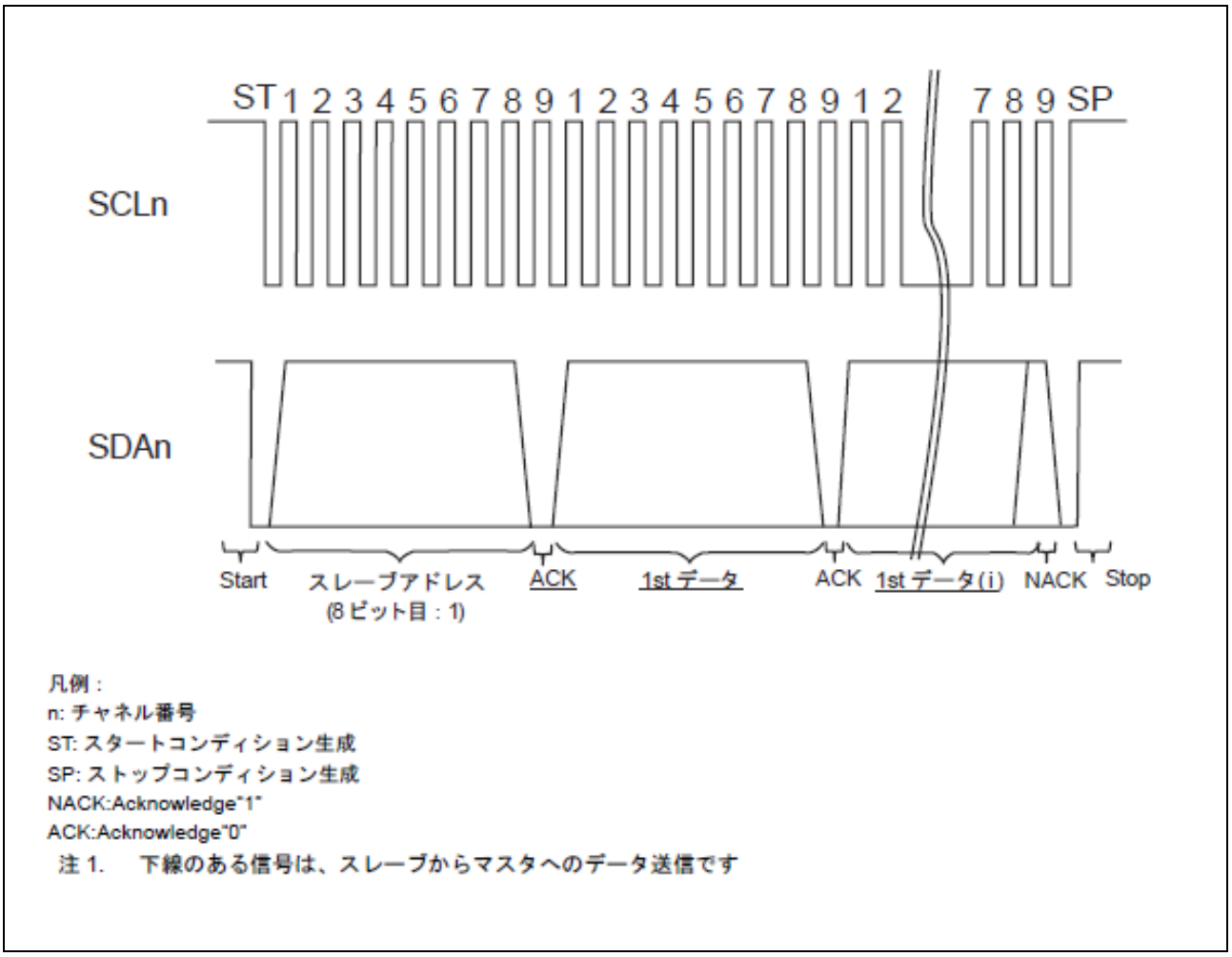


図 3-10 スレーブ送信 信号図

3.13.5　RIIC モジュールの状態確認

R_RIIC_GetStatus

(1)　概要

本モジュールの状態を確認する際に使用する関数です。

(2)　C 言語形式

riic_sts_flg_t R_RIIC_GetStatus(riic_info_t * p_riic_info, riic_mcu_status_t * p_riic_status)

(3)　パラメータ

I/O	パラメータ	説明
I	riic_info_t *　p_riic_info	RIIC 通信情報構造体のポインタ。 この構造体のうち、本関数で使用するメンバのみを以下に示します。 riic_ch_dev_status_t dev_sts;　ドライバ呼び出しにより更新あり 詳細は「図 10-6」参照 uint8_t ch_no;
I	riic_mcu_stat　p_riic_status us_t *	RIIC のステータスの格納位置へのポインタ

(4)　機能

本モジュールの状態を返します。
引数で指定した RIIC のチャネルの状態を、レジスタの読み出し、端子レベルの読み出し、
変数の読み出しなどにより取得し、32 ビットの構造体で戻り値として返します。
本関数のコールで、RIIC のアービトレーションロストフラグ、および NACK フラグを
“0” にクリアします。ステータスが“RIIC_ AL” の場合、“RIIC_FINISH” に更新します。

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合

使用例

```
volatile riic_return_t ret;
riic_info_t iic_info_m;
riic_mcu_status_t riic_status;

iic_info_m.ch_no = 0;

ret = R_RIIC_GetStatus(&iic_info_m, &riic_status);
```

3.13.6　RIIC コントロール処理

R_RIIC_Control

(1)　概要

主に通信エラー時に使用する関数です。
各コンディション出力、SDA 端子の Hi-Z 出力、SCL クロックのワンショット出力、
および RIIC のモジュールリセットすることができます。

(2)　C 言語形式

riic_return_t R_RIIC_Control(r_riic_info_t *p_riic_info, uint8_t ctrl_ptn)

(3)　パラメータ

I/O	パラメータ	説明
I	riic_info_t　p_riic_info *	RIIC 通信情報構造体のポインタ。 この構造体のうち、本関数で使用するメンバのみを以下に示します。 riic_ch_dev_status_t dev_sts;　出力パターン"RIIC_GEN_RESET"指定時、フラグ更新あり。詳細は「図 10-6」参照 uint8_t ch_no;
I	uint8_t　ctrl_ptn	出力パターンを設定します。

(4)　機能

RIIC の制御信号を出力します。引数で指定した各コンディション出力、SDA 端子 Hi-Z 出力、SCL クロックのワンショット出力、および RIIC のモジュールリセットを行います。

出力パターンに指定可能な値は表 3-56 を参照してください。
- 次の出力パターンは、同時指定が可能です。同時指定する場合は、“|” を用いてください。
・ “RIIC_GEN_START_CON” と “RIIC_GEN_STOP_CON” と “RIIC_GEN_RESTART_CON” の、3 つ、または、いずれか 2 つの組み合わせで同時指定可能です。
・ “RIIC_GEN_SDA_HI_Z” と “RIIC_GEN_SCL_ONESHOT” の 2 つを同時指定可能です。

```
RIIC_GEN_START_CON      0x01 /* スタートコンディションの生成*/
RIIC_GEN_STOP_CON       0x02 /* ストップコンディションの生成*/
RIIC_GEN_RESTART_CON    0x04 /* リスタートコンディションの生成*/
RIIC_GEN_SDA_HI_Z       0x08 /* SDA 端子を Hi-Z 出力 */
RIIC_GEN_SCL_ONESHOT    0x10 /* SCL クロックのワンショット出力 */
RIIC_GEN_RESET          0x20 /* RIIC のモジュールリセット */
```

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合
RIIC_ERR_NO_INIT	初期設定ができていない場合（未初期化状態）
RIIC_ERR_BUS_BUSY	バスビジーの場合
RIIC_ERR_AL	アービトレーションエラーが発生した場合
RIIC_ERR_OTHER	現状態に該当しない不正なイベントが発生した場合

使用例

```
/* Outputs an extra SCL clock cycle after changes the SDA pin in a high-impedance state*/
volatile riic_return_t ret;
riic_info_t iic_info_m;

iic_info_m.ch_no = 0;

ret = R_RIIC_Control(&iic_info_m, RIIC_GEN_SDA_HI_Z | RIIC_GEN_SCL_ONESHOT);
```

3.13.7　RIIC モジュールの停止

R_RIIC_Close

(1)　概要

RIIC の通信を終了し、使用していた RIIC を開放する際に使用する関数です。

(2)　C 言語形式

riic_return_t R_RIIC_Close(riic_info_t *p_riic_info)

(3)　パラメータ

I/O	パラメータ	説明
I	riic_info_t *　p_riic_info	RIIC 通信情報構造体のポインタ。 この構造体のうち、本関数で使用するメンバのみを以下に示します。 riic_ch_dev_status_t dev_sts;　ドライバ呼び出しにより更新あり 詳細は「図 10-6」参照 uint8_t ch_no;

(4)　機能

RIIC 通信を終了するための設定をします。引数で指定した RIIC のチャネルを無効にします。
本関数では次の処理を行います。
- RIIC のモジュールストップ状態への遷移
- RIIC 出力ポートの開放（SCL0, SDA0 をポートモード（入力）に変更）
- RIIC 割り込みの禁止

再度通信を開始するには、R_RIIC_Open ( ) (初期化関数) をコールする必要があります。通信中に強制的に停止した場合、その通信は保証しません。

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合

使用例

```
volatile riic_return_t ret;
riic_info_t iic_info_m;
iic_info_m.ch_no = 0;
ret = R_RIIC_Close(&iic_info_m);
```

3.13.8　RIIC バージョン取得

R_RIIC_GetVersion

(1)　概要

API のバージョンを返す関数です。

(2)　C 言語形式

uint32_t R_RIIC_GetVersion(void)

(3)　パラメータ

なし

(4)　機能

本 API のバージョン番号を返します。

(5)　戻り値

戻り値	意味
-	バージョン番号

使用例

```
uint32_t version;

version = R_RIIC_GetVersion();
```

3.14　USB ホスト制御

USB ホストを制御するためのドライバを提供します。

3.14.1　データ転送実行要求

R_usb_hstd_TransferStart

(1)　概要

データ転送実行要求

(2)　C 言語形式

USB_ER_t R_usb_hstd_TransferStart(USB_UTR_t * utr)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_ER_t　*utr	USB 通信構造体

(4)　機能

本 API は、utr に格納されている転送情報をもとにデータ転送要求をします。
データ転送終了（指定データサイズ、ショートパケット受信、エラー発生）時にコールバック関数が呼び出されます。このコールバック関数の引数には、送受信の残りデータ長、ステータス、転送終了の情報が設定されています。

(5)　戻り値

戻り値	意味
USB_OK	成功
USB_ERROR	失敗

備考 1　本 API はユーザアプリケーションまたはクラスドライバから呼び出してください。
備考 2　引数である構造体のメンバには、パイプ番号、転送データの先頭アドレス、転送データ長、終了時のコールバック関数を設定してください。
備考 3　エニュメレーション中のデバイスが存在する場合、そのデバイスに対するエニュメレーションが完了するまで本 API の戻り値は USB_ERROR となります。

使用例

```
USB_UTR_t usb_smp_trn_Msg[USB_MAXPIPE_NUM + 1];

USB_ER_t usb_smp_task(USB_UTR_CB_t complete, uint16_t pipe, uint32_t size, uint8_t*table)
{
    :
   /* 転送情報設定 */
   trn_msg[pipe].keyword = pipe; /* パイプ番号*/
   trn_msg[pipe].tranadr = table; /* データ格納バッファの先頭ポインタ */
   trn_msg[pipe].tranlen = size; /* データ転送サイズ */
   trn_msg[pipe].complete = complete; /* コールバック関数 */

   /* 転送開始要求 */
   err = R_usb_hstd_TransferStart(&trn_msg[pipe]);

   return err;
    :
}
```

3.14.2　データ転送強制終了要求

R_usb_hstd_TransferEnd

(1)　概要

データ転送強制終了要求

(2)　C 言語形式

USB_ER_t R_usb_hstd_TransferEnd(uint16_t pipe_id)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_ER_t　pipe_id	パイプ番号

(4)　機能

本 API は、指定したパイプのデータ転送の強制終了要求を行ないます。

(5)　戻り値

戻り値	意味
USB_OK	成功
USB_ERROR	失敗

備考 1　本 API はユーザアプリケーションまたはクラスドライバから呼び出してください。

使用例

```
void usb_smp_task(void)
{
        uint16_t pipe;
         :
        pipe = USB_PIPEx

        /*  転送終了要求  */
        err = R_usb_hstd_TransferEnd(pipe);
        return err;
         :
}
```

3.14.3　Host device class driver (HDCD) 登録

R_usb_hstd_DriverRegistration

(1)　概要

Host device class driver (HDCD) 登録

(2)　C 言語形式

void R_usb_hstd_DriverRegistration(USB_HCDREG_t *callback)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_HCDREG_t　*callback	クラスドライバ構造体

(4)　機能

クラスドライバ構造体に登録した HDCD の情報を HCD に登録します。HCD が管理する登録ドライバ数を更新し、新しい領域に HDCD の登録をします。登録完了後、初期化コールバック関数が実行されます。

(5)　戻り値

なし

備考 1　初期設定時、本 API をユーザアプリケーションプログラムまたはクラスドライバから呼び出してください。
備考 2　登録する情報は、「表 3-44 USB_HCDREG_t 構造体」を参照してください。

使用例

```
void usb_smp_registration(void)
{
    USB_HCDREG_t driver;
    /* Interface Class */
    driver.ifclass = (uint16_t)USB_IFCLS_VEN;
    /* Target peripheral list */
    driver.tpl = (uint16_t*)&usb_gapl_devicetpl;
    /* Driver check */
    driver.classcheck = &usb_hvendor_class_check;
    /* Device configured */
    driver.devconfig = &usb_hvendor_apl_init;
    /* Device detach */
    driver.devdetach = &usb_hvendor_apl_close;
    /* Device suspend */
    driver.devsuspend = &usb_hvendor_apl_suspend;
    /* Device resume */
    driver.devresume = &usb_hvendor_apl_resume;
    /* Driver registration */
    R_usb_hstd_DriverRegistration(&driver);
}
```

3.14.4　クラスチェック完了通知

R_usb_hstd_ReturnEnuMGR

(1)　概要

クラスチェック完了通知

(2)　C 言語形式

void R_usb_hstd_ReturnEnuMGR(uint16_t cls_result)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　cls_result	クラスチェック結果

(4)　機能

MGR にエニュメレーション処理の継続を要求します。MGR は本関数の引数を解析し、接続された USB デバイスを Configured ステートに遷移させるか判断します。
cls_result には、クラスチェック結果(USB_OK または USB_ERROR)を指定してください。

(5)　戻り値

なし

備考 1　本 API は、クラスチェック処理時に呼び出してください。

使用例

```
void usb_smp_task(void)
{
    :
    （エニュメレーション処理）
    :
  R_usb_hstd_ReturnEnuMGR(USB_OK);
}
```

3.14.5　接続されているデバイスの状態変更要求

R_usb_hstd_ChangeDeviceState

(1)　概要

接続されているデバイスの状態変更要求

(2)　C 言語形式

USB_ER_t R_usb_hstd_ChangeDeviceState(USB_UTR_CB_t complete, int16_t msginfo, uint16_t member)

(3)　パラメータ

I/O	パラメータ		説明
I	USB_UTR_CB_t	complete	コールバック関数
		msginfo	メッセージメッインフォメーション
		member	デバイスアドレス

(4)　機能

msginfo に次の値を設定し、この API を呼び出すと、接続された USB デバイスの状態を変更する要求が HCD に送信されます。

msginfo	Outline
USB_DO_GLOBAL_SUSPEND	リモートウェークアップを有効にしてサスペンド状態への移行要求（ハブからダウンストリームのデバイスに対してデバイス固有のサスペンド状態を要求する）
USB_DO_GLOBAL_RESUME	グローバル再開の実行要求（ポートから下流に接続されたデバイスの動作を再開する）
USB_DO_CLR_STALL	STALL クリアリクエスト

(5)　戻り値

戻り値	意味
USB_OK	成功
USB_ERROR	失敗

備考 1　本 API は、ユーザーアプリケーションプログラムまたはクラスドライバから呼び出してください。
備考 2　本 API は、引数 msginfo に USB_DO_CLR_STALL を設定する場合は、パイプ番号を引数メンバに設定してください。

使用例

```
void usb_smp_task(void)
{
   R_usb_hstd_ChangeDeviceState(&usb_smp_callback, USB_DO_GROBAL_SUSPEND, devaddr);
}
```

3.14.6　パイプ情報設定

R_usb_hstd_SetPipe

(1)　概要

パイプ情報設定

(2)　C 言語形式

USB_ER_t R_usb_hstd_SetPipe(uint16_t **table)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　　**table	クラスチェック情報テーブルポインタ（ 表 11-8 参照）

(4)　機能

受信したディスクリプタを解析してパイプ情報を設定します。

(5)　戻り値

戻り値	意味
USB_OK	成功
USB_ERROR	失敗

備考 1　本 API は、クラスチェック処理時に呼び出してください。
備考 2　パイプ情報領域に空きが無い場合は、USB_ERROR を返します。

使用例

```
void usb_smp_classcheck(uint16_t **table)
{
        R_usb_hstd_SetPipe(table);
}
```

3.14.7　パイプ番号取得

R_usb_hstd_GetPipeID

(1)　概要

パイプ番号取得

(2)　C 言語形式

uint16_t R_usb_hstd_GetPipeID(uint16_t devaddr, uint8_t type, uint8_t direction, uint8_t ifnum)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	devaddr	デバイスアドレス
I	uint8_t	type	エンドポイント属性 (USB_EP_ISO / USB_EP_BULK / USB_EP_INT)
I	uint8_t	direction	エンドポイント方向(USB_EP_IN / USB_EP_OUT)
I	uint8_t	ifnum	インターフェース番号

(4)　機能

引数で指定した条件に合致したパイプ番号を返却します。

(5)　戻り値

戻り値	意味
uint16_t	パイプ番号

備考 1　本 API は、ユーザアプリケーションプログラムまたはクラスドライバから呼び出してください。
備考 2　R_usb_hstd_SetPipe()でパイプ情報を設定した後に呼び出してください。
備考 3　インターフェース番号に 0xFF を指定すると、インターフェース番号を無視して検索します。

使用例

```
void usb_smp_configured(uint16_t devaddr)
{
        usb_gsmp_pipe = R_usb_hstd_GetPipeID(devaddr, USB_EP_BULK, USB_EP_IN, 0);
}
```

3.14.8　パイプ情報クリア

R_usb_hstd_ClearPipe

(1)　概要

パイプ情報クリア

(2)　C 言語形式

USB_ER_t R_usb_hstd_ClearPipe(uint16_t devaddr)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　devaddr	デバイスアドレス

(4)　機能

引数で指定したデバイスアドレスのパイプ設定をクリアして、領域を開放します。

(5)　戻り値

戻り値	意味
USB_OK	成功

備考 1　本 API は、ユーザアプリケーションプログラムまたはクラスドライバから呼び出してください。

使用例

```
void usb_smp_detach(uint16_t devaddr)
{
        R_usb_hstd_ClearPipe(devaddr);
}
```

3.14.9　MGR タスク起動

R_usb_hstd_MgrOpen

(1)　概要

MGR タスク起動

(2)　C 言語形式

USB_ER_t R_usb_hstd_MgrOpen(void)

(3)　パラメータ

なし

(4)　機能

MGR の登録状態を初期化します。
戻り値は常に USB_OK になります。

(5)　戻り値

戻り値	意味
USB_OK	成功

備考 1　初期設定時、ユーザアプリケーションプログラムで本 API を呼び出してください。
備考 2　MGR タスク起動後、本 API を呼び出さないでください。

使用例

```
void usb_smp_task(void)
{
        R_usb_hstd_MgrOpen();
}
```

3.14.10　MGR タスク

R_usb_hstd_MgrTask

(1)　概要

MGR タスク

(2)　C 言語形式

void R_usb_hstd_MgrTask(void)

(3)　パラメータ

なし

(4)　機能

ホスト機能選択時、HCD と HDCD 間の機能を補完します。

(5)　戻り値

なし

備考 1　スケジューラ処理をするループ内で呼び出してください。

使用例

```
void usb_smp_mainloop(void)
{
  while(1)
  {
    /* スケジューラ起動 */
    R_usb_cstd_Scheduler();
    /* フラグチェック */
    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
        R_usb_hstd_MgrTask();
    }
  }
}
```

3.14.11　HUB class driver(HUBCD)登録

R_usb_hhub_Registration

(1)　概要

HUB class driver(HUBCD)登録

(2)　C 言語形式

void R_usb_hhub_Registration (USB_HCDREG_t *callback)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_HCDREG_t　*callback	USB_HCDREG_t 構造体へのポインタ

(4)　機能

Hub class driver(HUBCD)の情報を登録します。

(5)　戻り値

なし

備考 1　初期設定時、ユーザアプリケーションプログラムで本 API を呼び出してください。
備考 2　USB_HCDREG_t 構造体のメンバ tpl に TPL 領域を設定した後、本 API をコールしてください。

使用例

```
extern uint16_t usb_ghhub_TPL[];

void usb_smp_registration(void)
{
   USB_HCDREG_t   driver;
    :
   driver.tpl = usb_ghhub_TPL;
   R_usb_hhub_Registration(&driver);
    :
}
```

3.14.12　HUB タスク

R_usb_hhub_Task

(1)　概要

HUB class driver(HUBCD)登録

(2)　C 言語形式

void R_usb_hhub_Task(void)

(3)　パラメータ

なし

(4)　機能

接続された USB ハブのダウンポートの状態を管理し、HCD と HDCD 間の機能を補完します。

(5)　戻り値

なし

備考 1　スケジューラ処理をするループ内で本 API を呼び出してください。

使用例

```
void usb_smp_mainloop(void)
{
  while(1)
  {
    /* スケジューラ起動 */
    R_usb_cstd_Scheduler();
    /* フラグチェック */
    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
     :
    R_usb_hhub_Task();
     :
  }
}
```

3.14.13　処理要求を送信

R_usb_cstd_SndMsg

(1)　概要

優先テーブルに処理要求を送信する。

(2)　C 言語形式

USB_ER_t R_usb_cstd_SndMsg(uint8_t id, USB_MSG_t *mess)

(3)　パラメータ

I/O	パラメータ		説明
I	uint8_t	id	メッセージを送信するタスク ID
I	USB_MSG_t	*mess	送信するメッセージ

(4)　機能

優先テーブルに処理要求を発信します。本 API は R_USB_SND_MSG マクロで定義されています。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	タスクID が未設定 タスク優先度が未設定 優先テーブルに空きがなく通知不可

備考 1　割り込み関数以外のユーザアプリケーションプログラムまたはクラスドライバで、本 API が登録されているスケジューラマクロ(R_USB_SND_MSG)を呼び出してください。
備考 2　送信するメッセージには、メモリバッファの先頭アドレスを指定してください。
備考 3　サンプルプログラムでは、R_PGET_BLK マクロで取得したメモリブロックの先頭アドレスを送信メッセージとして指定しています。

使用例

```
void usb_smp_task()
{
      USB_UTR_t *p_blf;
      USB_ER_t err;
       :

      /*メッセージ格納領域確保*/
      err = R_USB_PGET_BLK(USB_SMP_MPL, &p_blf);

      if(err == USB_OK)
      {
            /* メッセージ設定 */
            p_blf->msginfo = USB_SMP_REQ;
            p_blf->keyword = keyword;
            p_blf->complete = complete;
      }

      /* メッセージ送信 */
      err = R_USB_SND_MSG(USB_SMP_MBX, (USB_MSG_t*)p_blf);
      if(err != USB_OK)
      {
            /* エラー処理 */
      }
       :
}
```

3.14.14　処理要求を確認

R_usb_cstd_RecMsg

(1)　概要

優先テーブルに処理要求があるか確認する。

(2)　C 言語形式

USB_ER_t R_usb_cstd_RecMsg(uint8_t id, USB_MSG_t mess)**

(3)　パラメータ

I/O	パラメータ		説明
I	uint8_t	id	メッセージを受信するタスク ID
I	USB_MSG_t	*mess	受信するメッセージ

(4)　機能

優先テーブルにタスク ID の処理要求があるか確認します。本関数は R_USB_RCV_MSG/に登録されています。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	タスクID が未設定

備考 1　ユーザアプリケーションプログラムまたはクラスドライバで、本 API が登録されているスケジューラマクロを呼び出してください。

備考 2　R_usb_cstd_CheckSchedule ()関数の戻り値が USB_FLGCLR の場合には、本 API を実行しないでください。

使用例

```
void usb_smp_task()
{
    USB_UTR_t *mess;
    USB_ER_t err;
     :
    /*メッセージ確認*/
    err = R_USB_RCV_MSG(USB_SMP_MBX, (USB_MSG_t**)&mess);
    if(err == USB_OK)
    {
       /*  メッセージに対応した処理  */
    }
     :
}
```

3.14.15　処理要求を格納する領域を確保する

R_usb_cstd_PgetBlk

(1)　概要

処理要求を格納する領域を確保する。

(2)　C 言語形式

USB_ER_t R_usb_cstd_PgetBlk(uint8_t id, USB_UTR_t blk)**

(3)　パラメータ

I/O	パラメータ		説明
I	uint8_t	id	領域を確保するタスク ID
I	USB_MSG_t	**blk	確保する領域のポインタ

(4)　機能

メッセージを格納する領域を確保します。本 API では、1 ブロックあたり 40 バイト確保します。
本関数は R_USB_PGET_BLK に登録されています。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	タスクID が未設定 領域確保失敗

備考 1　ユーザアプリケーションプログラムまたはクラスドライバで、本 API が登録されているスケジューラマクロを呼び出してください。

使用例

```
void usb_smp_task()
{
   USB_UTR_t *p_blf;
   USB_ER_t err;
    :
   /*メッセージ格納領域確保*/
   err = R_USB_PGET_BLK(USB_SMP_MPL, &p_blf);
   if(err == USB_OK)
   {
      /* メッセージ設定 */
      p_blf->msginfo = USB_SMP_REQ;
      p_blf->keyword = keyword;
      p_blf->complete = complete;
   }

   /* メッセージ送信 */
   err = R_USB_SND_MSG(USB_SMP_MBX, (USB_MSG_t*)p_blf);
   if(err != USB_OK)
   {
         /* エラー処理 */
   }
    :
}
```

3.14.16　処理要求を格納する領域を解放

R_usb_cstd_RelBlk

(1)　概要

処理要求を格納する領域を解放する。

(2)　C 言語形式

USB_ER_t R_usb_cstd_RelBlk(uint8_t id,　USB_UTR_t* blk)

(3)　パラメータ

I/O	パラメータ		説明
I	uint8_t	id	領域を解放するタスク ID
I	USB_MSG_t	*blk	解放する領域のポインタ

(4)　機能

メッセージを格納する領域を開放します。本関数は R_USB_REL_BLK に登録されています。

(5)　戻り値

戻り値	意味
USB_OK	領域解放完了
USB_ERROR	タスクID が未設定 領域解放失敗

備考 1　ユーザアプリケーションプログラムまたはクラスドライバで、本 API が登録されているスケジューラマクロを呼び出してください。

使用例

```
void usb_smp_task()
{
   USB_ER_t err;
   USB_UTR_t *mess;
    :
   /*メッセージ格納領域開放*/
   err = R_USB_REL_BLK(USB_SMP_MPL, mess);
   if(err != USB_OK)
   {
        /*  エラー処理  */
   }
    :
}
```

3.14.17　処理要求を管理

R_usb_cstd_Scheduler

(1)　概要

各タスクからの処理要求を管理する。

(2)　C 言語形式

void R_usb_cstd_Scheduler(void)

(3)　パラメータ

なし

(4)　機能

各タスクからの処理要求を管理します。

(5)　戻り値

なし

備考 1　non-OS 時、スケジューラ処理をするループ内で呼び出してください。

使用例

```
void usb_smp_mainloop(void)
{
   while(1)
   {
   /* フラグチェック */
   if(USB_FLGSET == R_usb_cstd_CheckSchedule())
   {
       :
     /* タスク処理 */
       :
   }
   /* スケジューラ起動 */
   R_usb_cstd_Scheduler();
    :
}
```

3.14.18　タスクの優先度を設定

R_usb_cstd_SetTaskPri

(1)　概要

タスクの優先度を設定する。

(2)　C 言語形式

void R_usb_cstd_SetTaskPri(uint8_t tasknum, uint8_t pri)

(3)　パラメータ

I/O	パラメータ		説明
I	uint8_t	tasknum,	タスク ID
I	uint8_t	pri	タスク優先度

(4)　機能

タスクの優先度を設定します。第一引数(tasknum)で指定したタスクに、第二引数で指定した優先度(pri)を設定します。

(5)　戻り値

なし

備考 1　本 API は、初期設定時にユーザアプリケーションプログラムで呼び出してください。タスク ID の優先度が設定されることで、メッセージの送受信が可能になります。

使用例

```
void usb_smp_driver_start(void)
{
  /* サンプルタスクの優先度を 4 に設定 */
  R_usb_cstd_SetTaskPri(USB_SMP_TSK, USB_PRI_4);
}
```

3.14.19　処理要求があるか確認

R_usb_cstd_CheckSchedule

(1)　概要

登録されたタスクに処理要求があるか確認する。

(2)　C 言語形式

uint8_t R_usb_cstd_CheckSchedule(void)

(3)　パラメータ

なし

(4)　機能

登録されたタスクに受信メッセージがあるか確認します。

(5)　戻り値

戻り値	意味
USB_FLGSET	処理要求あり
USB_FLGCLR	処理要求なし

備考 1　non-OS 時、スケジューラ処理をするループ内で本 API を呼び出してください。

使用例

```
void usb_smp_mainloop(void)
{
   while(1)
   {
   /* フラグチェック */
   if(USB_FLGSET == R_usb_cstd_CheckSchedule())
   {
    :
   /* タスク処理 */
    :
   }
   /* スケジューラ起動 */
   R_usb_cstd_Scheduler();
    :
}
```

3.14.20　HMSCD タスク

R_usb_hmsc_Task

(1)　概要

HMSCD タスク

(2)　C 言語形式

void R_usb_hmsc_Task(void)

(3)　パラメータ

なし

(4)　機能

HMSCD のタスクです。
BOT の制御を行います。

(5)　戻り値

なし

備考 1　スケジューラ処理をするループ内で本 API を呼び出してください。

使用例

```
void usb_smp_mainloop(void)
{
  while(1)
  {
  /* スケジューラ起動 */
  R_usb_cstd_Scheduler();
  /* フラグチェック */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
    R_usb_hstd_MgrTask();
    R_usb_hhub_Task();
    R_usb_hmsc_Task();
  }
}
```

3.14.21　HMSC ドライバ起動

R_usb_hmsc_driver_start

(1)　概要

HMSC ドライバ起動

(2)　C 言語形式

void R_usb_hmsc_driver_start(void)

(3)　パラメータ

なし

(4)　機能

HMSC ドライバタスクの優先度を設定します。

(5)　戻り値

なし

備考 1　初期設定時にユーザアプリケーションで本 API を呼び出してください。

使用例

```
void usb_hstd_task_start( void )
{
   R_usb_hmsc_driver_start();     /* Host Class Driver Task Start Setting */
}
```

3.14.22　ディスクリプタチェック処理

R_usb_hmsc_ClassCheck

(1)　概要

ディスクリプタチェック処理

(2)　C 言語形式

void R_usb_hmsc_ClassCheck(uint16_t **table)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t **table	デバイス情報テーブル [0]：デバイスディスクリプタ [1]：コンフィグレーションディスクリプタ [2]：インタフェースディスクリプタ [3]：ディスクリプタチェック結果 [4]：HUB 種別 [5]：ポート番号 [6]：通信速度 [7]：デバイスアドレス

(4)　機能

本 API はクラスドライバレジストレーション関数です。この関数は、スタートアップ時の HMSC 登録時にドライバレジストレーション構造体メンバ classcheck にコールバック関数として登録され、エニュメレーション動作のコンフィグレーションディスクリプタ受信時に呼出されます。
ペリフェラルデバイスのコンフィグレーションディスクリプタからエンドポイントディスクリプタを参照し、パイプ情報テーブルの登録をします。

(5)　戻り値

なし

使用例

```
void usb_hapl_registration(USB_UTR_t *ptr)
{
   USB_HCDREG_t driver;
   /* Driver check */
   driver.classcheck = &R_usb_hmsc_ClassCheck;
}
```

3.14.23　HMSCD 動作状態の応答

R_usb_hmsc_GetDevSts

(1)　概要

HMSCD 動作状態の応答

(2)　C 言語形式

uint16_t R_usb_hmsc_GetDevSts(uint16_t side)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　side	ドライブ番号

(4)　機能

HMSCD の動作状態を取得します。

(5)　戻り値

戻り値	意味
usb_ghmsc_AttSts　SB_HMSC_DEV_ATT	接続
usb_ghmsc_AttSts　SB_HMSC_DEV_DET	切断

備考 1　引数 side には、R_usb_hmsc_alloc_drvno()によって割り当てられたドライブ番号を指定してください。

使用例

```
void usb_smp_task(USB_UTR_t *ptr)
{
   /* デバイス状態確認*/
   if(R_usb_hmsc_GetDevSts(drvno) == USB_HMSC_DEV_DET)
   {
   /* 切断処理 */
   }
}
```

3.14.24　READ10 コマンド発行

R_usb_hmsc_Read10

(1)　概要

READ10 コマンド発行

(2)　C 言語形式

uint16_t R_usb_hmsc_Read10(uint16_t side, uint8_t *buff, uint32_t secno,
uint16_t seccnt, uint32_t trans_byte)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	side	ドライブ番号
I	uint8_t	*buff	読み出しデータエリア
I	uint32_t	secno	セクタ番号
I	uint16_t	seccnt	セクタ数
I	uint32_t	trans_byte	転送データ長

(4)　機能

USB デバイスに READ10 コマンドを発行します。
コマンドエラーの場合は REQUEST_SENSE コマンドを発行しエラー情報を取得します。

(5)　戻り値

戻り値	意味
-	エラーコード

使用例

```
void usb_smp_task(void)
{
    uint16_t result;

    /* READ10 発行 */
    result = R_usb_hmsc_Read10(side, buff, secno, seccnt, trans_byte);
    if(result != USB_HMSC_OK)
    {
        /* エラー処理 */
    }
}
```

3.14.25　WRITE10 コマンド発行

R_usb_hmsc_Write10

(1)　概要

WRITE10 コマンド発行

(2)　C 言語形式

uint16_t R_usb_hmsc_Write10(uint16_t side, const uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	side	ドライブ番号
I	uint8_t	*buff	読み出しデータエリア
I	uint32_t	secno	セクタ番号
I	uint16_t	seccnt	セクタ数
I	uint32_t	trans_byte	転送データ長

(4)　機能

USB デバイスに WRITE10 コマンドを発行します。
コマンドエラーの場合は REQUEST_SENSE コマンドを発行しエラー情報を取得します。

(5)　戻り値

戻り値	意味
-	エラーコード

使用例

```
void usb_smp_task(void)
{
    uint16_t result;

    /* WRITE10 発行 */
    result = R_usb_hmsc_Write10(side, buff, secno, seccnt, trans_byte);
    if(result != USB_HMSC_OK)
    {
        /* エラー処理 */
    }
}
```

3.14.26　GetMaxLUN リクエスト発行

R_usb_hmsc_GetMaxUnit

(1)　概要

GetMaxLUN リクエスト発行

(2)　C 言語形式

USB_ER_t R_usb_hmsc_GetMaxUnit(uint16_t addr, USB_CB_t complete)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	addr	デバイスアドレス
I	USB_CB_t	complete	コールバック関数

(4)　機能

GET_MAX_LUN リクエストを発行して、最大ユニット数を取得します。リクエストが完了すると引数 complete に指定したコールバック関数がコールされます。

(5)　戻り値

戻り値	意味
USB_OK	GET_MAX_LUN 発行
USB_ERROR	GET_MAX_LUN 未発行

使用例

```
void usb_smp_task(void)
{
    USB_ER_t err;
    /* 最大ユニット数取得*/
    err = R_usb_hmsc_GetMaxUnit(devadr, usb_hmsc_StrgCheckResult);
    if(err == USB_ERROR)
    {
        /* エラー処理 */
    }
}
```

3.14.27　MassStorageReset リクエスト発行

R_usb_hmsc_MassStorageReset

(1)　概要

MassStorageReset リクエスト発行

(2)　C 言語形式

USB_ER_t R_usb_hmsc_MassStorageReset(uint16_t addr, USB_CB_t complete)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	addr	デバイスアドレス
I	USB_CB_t	complete	コールバック関数

(4)　機能

MASS_STORAGE_RESET リクエストを発行して、プロトコルエラーを解除します。リクエストが完了すると引数 comlete に指定したコールバック関数がコールされます。

(5)　戻り値

戻り値	意味
USB_OK	MASS_STORAGE_RESET 発行
USB_ERROR	MASS_STORAGE_RESET 未発行

使用例

```
void usb_smp_task(void)
{
   USB_ER_t err;

   /* プロトコルエラー解除*/
   err = R_usb_hmsc_MassStorageReset(devadr, usb_hmsc_CheckResult);
   if(err == USB_ERROR)
   {
          /* エラー処理 */
   }
}
```

3.14.28　ドライブ番号割り当て

R_usb_hmsc_alloc_drvno

(1)　概要

ドライブ番号割り当て

(2)　C 言語形式

uint16_t R_usb_hmsc_alloc_drvno(uint16_t *side, uint16_t devadr)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	*side	ドライブ番号ポインタ
I	uint16_t	devadr	MSC デバイスのデバイスアドレス

(4)　機能

接続された MSC デバイスに対しドライブ番号を割り当て、引数*side に格納します。
ドライブ番号は 0 から順に割り当てられます。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

使用例

```
void usb_smp_task(void)
{
  /* ドライブ番号割り当て */
  R_usb_hmsc_alloc_drvno(&drvno, devadr);
}
```

3.14.29　ドライブ番号解放

R_usb_hmsc_free_drvno

(1)　概要

ドライブ番号解放

(2)　C 言語形式

uint16_t R_usb_hmsc_free_drvno(uint16_t side)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　side	ドライブ番号

(4)　機能

引数で指定されたドライブ番号を解放します。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

使用例

```
void usb_smp_task(void)
{
  /* ドライブ番号解放 */
  R_usb_hmsc_free_drvno(drvno);
}
```

3.14.30　ドライブ番号参照

R_usb_hmsc_ref_drvno

(1)　概要

ドライブ番号参照

(2)　C 言語形式

uint16_t R_usb_hmsc_ref_drvno(uint16_t *side, uint16_t devadr)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	*side	ドライブ番号ポインタ
I	uint16_t	devadr	MSC デバイスのデバイスアドレス

(4)　機能

引数で指定されたデバイスアドレスのドライブ番号を参照して、引数*side に格納します。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

使用例

```
void usb_smp_task(void)
{
  /* ドライブ番号参照 */
  R_usb_hmsc_ref_drvno(&drvno, devadr);
}
```

3.14.31　マスストレージドライブタスク

R_usb_hmsc_StrgDriveTask

(1)　概要

マスストレージドライブタスク

(2)　C 言語形式

void R_usb_hmsc_StrgDriveTask(void)

(3)　パラメータ

なし

(4)　機能

ストレージコマンドを送信し、接続されたストレージ機器の情報を取得する処理を行います。

(5)　戻り値

なし

備考 1　スケジューラ処理を行うループ内で本 API を呼び出してください。

使用例

```
void usb_hapl_mainloop(void)
{
   while(1)
   {
      R_usb_cstd_Scheduler();
      if(USB_FLGSET == R_usb_cstd_CheckSchedule())
      {
         R_usb_hstd_MgrTask();
         R_usb_hhub_Task();
         R_usb_hmsc_Task();
         R_usb_hmsc_StrgDriveTask();
      }
   }
}
```

3.14.32　ドライブ情報取得

R_usb_hmsc_StrgDriveSearch

(1)　概要

ドライブ情報取得

(2)　C 言語形式

uint16_t R_usb_hmsc_StrgDriveSearch(uint16_t addr, USB_UTR_CB_t complete)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	addr	デバイスアドレス
I	USB_UTR_CB_t	complete	コールバック関数

(4)　機能

引数 addr で指定された USB デバイスに対し、情報取得処理をします。
処理が完了すると引数 complete に設定したコールバック関数がコールされます。
詳細は「11.9.4　USB ストレージ機器の情報取得」を参照してください。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

備考 1　本 API をコールしてからコールバック関数がコールされるまでは、スケジューラを動作させてください。また、他の USB 関連処理をしないでください。

使用例

```
/* Callback function */
void usb_hmsc_StrgCommandResult( USB_UTR_t *mess )
{
   :
}

void usb_hmsc_SampleAplTask(void)
{
   :
   R_usb_hmsc_StrgDriveSearch(addr, &usb_hmsc_StrgCommandResult);
   :
}
```

3.14.33　ドライブオープン

R_usb_hmsc_StrgDriveOpen

(1)　概要

ドライブオープン

(2)　C 言語形式

uint16_t R_usb_hmsc_StrgDriveOpen(uint16_t *side, uint16_t addr)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	*side	ドライブ番号ポインタ
I	uint16_t	addr	デバイスアドレス

(4)　機能

引数で指定されたアドレスをオープンします。
エニュメレーション完了後にコールしてください。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

備考 1　関数内で R_usb_hmsc_alloc_drvno() を使用してドライブ番号を取得します。
備考 2　関数内で R_usb_hstd_GetPipeID() を使用してパイプ番号を設定します。

使用例

```
void msc_configured(uint16_t devadr)
{
   R_usb_hmsc_StrgDriveOpen(&drvno, devadr);
}
```

3.14.34　ドライブクローズ

R_usb_hmsc_StrgDriveClose

(1)　概要

ドライブクローズ

(2)　C 言語形式

uint16_t R_usb_hmsc_StrgDriveClose(uint16_t side)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　side	ドライブ番号

(4)　機能

引数で指定されたドライブをクローズします。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

備考 1　関数内で R_usb_hmsc_free_drvno() を使用してドライブ番号を開放します。
備考 2　関数内で R_usb_hstd_ClearPipe() を使用してパイプ情報をクリアします。

使用例

```
void msc_detach(uint16_t devadr)
{
  R_usb_hmsc_StrgDriveClose(drv_no);
}
```

3.14.35　セクタ読み出し

R_usb_hmsc_StrgReadSector

(1)　概要

セクタ読み出し

(2)　C 言語形式

uint16_t R_usb_hmsc_StrgReadSector(uint16_t side, uint8_t *buff
, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	side	ドライブ番号
I	uint8_t	*buff	読み出しデータ格納領域
I	uint32_t	secno	セクタ番号
I	uint16_t	seccnt	セクタ数
I	uint32_t	trans_byte	転送データ長

(4)　機能

引数で指定されたドライブのセクタ情報を読み出します。
本 API は、ストレージ機器からドライブ情報が正しく読み込めなかった場合にエラーを返します。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

備考 1　本 API は FSI で呼び出してください。
備考 2　関数内で R_usb_hmsc_Read10() を使用してセクタ情報を読み出します。
備考 3　関数内で R_usb_hmsc_GetDevSts() を使用して接続状況を確認しており、切断状態であれば、読み出し前にエラー終了します。

使用例

```
DRESULT disk_read(BYTE pdrv, BYTE* buff, DWORD sector, UINT count)
{
   uint32_t err;
   uint32_t tran_byte;

   /* set transfer length */
   tran_byte = count * _MIN_SS;

   /* read function */
   err = R_usb_hmsc_StrgReadSector(pdrv, buff, sector, (uint16_t)count, tran_byte);
   if (err != USB_OK)
   {
   return RES_ERROR;
   }
}
```

3.14.36　セクタ書き込み

R_usb_hmsc_StrgWriteSector

(1)　概要

セクタ書き込み

(2)　C 言語形式

uint16_t R_usb_hmsc_StrgWriteSector(uint16_t side, const uint8_t *buff
, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	side	ドライブ番号
I	const uint8_t	*buff	読み出しデータ格納領域
I	uint32_t	secno	セクタ番号
I	uint16_t	seccnt	セクタ数
I	uint32_t	trans_byte	転送データ長

(4)　機能

引数で指定されたドライブのセクタ情報を書き込みます
本 API は、ストレージ機器からドライブ情報が正しく読み込めなかった場合にエラーを返します。

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

備考 1　本 API は FSI で呼び出してください。
備考 2　関数内で R_usb_hmsc_Write10() を使用してセクタ情報を書き込みます。
備考 3　関数内で R_usb_hmsc_GetDevSts() を使用して接続状況を確認しており、切断状態であれば、書き込み前にエラー終了します。

使用例

```
DRESULT disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)
{
   uint32_t err;
   uint32_t tran_byte;

   /* set transfer length */
   tran_byte = count * _MIN_SS;

   /* write function */
   err = R_usb_hmsc_StrgWriteSector(pdrv, buff, sector, (uint16_t)count, tran_byte);
   if (err != USB_OK)
   {
            return RES_ERROR;
   }
}
```

3.14.37　読み出し／書き込み完了確認

R_usb_hmsc_StrgCheckEnd

(1)　概要

読み出し／書き込み完了確認

(2)　C 言語形式

uint16_t R_usb_hmsc_StrgCheckEnd(void)

(3)　パラメータ

なし

(4)　機能

読み出し／書き込みシーケンスの状態を返却します。

(5)　戻り値

戻り値	意味
USB_FALSE	読み出し／書き込み中
USB_TRUE	読み出し／書き込み完了
USB_ERROR	読み出し／書き込みエラー

備考 1　本 API は、R_usb_hmsc_StrgReadSector()または R_usb_hmsc_StrgWriteSector()をコールした後に定期的に呼び出してください。

使用例

```
DRESULT disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)
{
   uint32_t err;

   /* write function */
   R_usb_hmsc_StrgWriteSector(pdrv, buff, sector, (uint16_t)count, tran_byte);

   /* Wait USB write sequence(WRITE10) */
   do
   {
      R_usb_cstd_Scheduler();
      if (R_usb_cstd_CheckSchedule() == USB_FLGSET)
      {
         R_usb_hstd_MgrTask(); /* MGR task */
         R_usb_hhub_Task(); /* HUB task */
         R_usb_hmsc_task(); /* HMSC Task */
         R_usb_hmsc_StrgDriveTask(); /* HSTRG Task */
      }
      err = R_usb_hmsc_StrgCheckEnd();
   }
   while (err == USB_FALSE);

   /* Set transfer result */
   if (err != USB_TRUE)
   {
      return RES_ERROR;
   }
   else
   {
      return RES_OK;
   }
}
```

3.14.38　ストレージコマンド発行

R_usb_hmsc_StrgUserCommand

(1)　概要

ストレージコマンド発行

(2)　C 言語形式

uint16_t R_usb_hmsc_StrgUserCommand(uint16_t side, uint16_t command , uint8_t *buff, USB_UTR_CB_t complete)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	side	ドライブ番号
I	uint16_t	command	発行するコマンド
I	uint8_t	*buff	データポインタ
I	USB_CB_t	complete	コールバック関数

(4)　機能

引数で指定されたドライブへストレージコマンドを発行します。コマンドが完了すると引数 complete に指定したコールバック関数がコールされます。
以下に、本 API が対応するストレージコマンドを示します。

ストレージコマンド	概要
USB_ATAPI_TEST_UNIT_READY	ペリフェラル機器の状態確認
USB_ATAPI_REQUEST_SENCE	ペリフェラル機器の状態取得
USB_ATAPI_INQUIRY	論理ユニットのパラメータ情報取得
USB_ATAPI_MODE_SELECT6	パラメータ指定
USB_ATAPI_PREVENT_ALLOW	メディアの取り出し許可/禁止
USB_ATAPI_READ_FORMAT_CAPACITY	フォーマット可能な容量取得
USB_ATAPI_READ_CAPACITY	論理ユニットの容量情報取得
USB_ATAPI_MODE_SENSE10	論理ユニットのパラメータ取得

(5)　戻り値

戻り値	意味
USB_OK	正常終了
USB_ERROR	エラー終了

備考 1　関数内で HMSCD の API 関数を使用してストレージコマンドを発行します。
備考 2　関数内で R_usb_hmsc_GetDevSts() を使用して接続状況を確認しており、切断状態であれば、ストレージコマンド発行前にエラー終了します。

使用例

```
/* Callback function */
void strgcommand_complete(USB_UTR_t *mess)
{
   :
}

void usb_smp_task(void)
{
    :
   /* TEST_UNIT_READY */
   err = R_usb_hmsc_StrgUserCommand(side, USB_ATAPI_TEST_UNIT_READY, buf,
   strgcommand_complete);
    :
}
```

3.14.39　デバイスの状態取得

disk_status

(1)　概要

デバイスの状態取得

(2)　C 言語形式

DSTATUS disk_status(BYTE pdrv)

(3)　パラメータ

I/O	パラメータ	説明
I	BYTE pdrv	物理ドライブ番号(0-9)

(4)　機能

サンプルアプリケーションの変数 R_usb_disk_status[pdrv] の値を返却します。
pdrv に 10 以上が設定された場合は、STA_NODISK | STA_NOINIT を返却します。

(5)　戻り値

現在のストレージデバイスの状態を次のフラグの組み合わせ値で返します。

戻り値	意味
STA_NOINIT	デバイスが初期化されていないことを示すフラグ
STA_NODISK	メディアが存在しないことを示すフラグ
STA_PROTECT	メディアがライトプロテクトされていることを示すフラグ (サンプルでは未使用)

3.14.40　デバイスの初期化

disk_initialize

(1)　概要

デバイスの初期化

(2)　C 言語形式

DSTATUS disk_initialize(BYTE pdrv)

(3)　パラメータ

I/O	パラメータ	説明
I	BYTE pdrv	物理ドライブ番号(0-9)

(4)　機能

R_usb_disk_status[pdrv] の値を返却します。

(5)　戻り値

現在のストレージデバイスの状態を次のフラグの組み合わせ値で返します。

戻り値	意味
STA_NOINIT	デバイスが初期化されていないことを示すフラグ
STA_NODISK	メディアが存在しないことを示すフラグ
STA_PROTECT	メディアがライトプロテクトされていることを示すフラグ (サンプルでは未使用)

備考 1　この関数は FatFs の管理下にあり、自動マウント動作により必要に応じて呼び出されます。
アプリケーションからの呼び出しは禁止です。
備考 2　再初期化が必要なときは、FatFs の API 関数（f_mount()）を使用してください。

3.14.41　デバイスの読み出し

disk_read

(1)　概要

デバイスの読み出し

(2)　C 言語形式

DRESULT disk_read(BYTE pdrv, BYTE* buff, DWORD sector, UINT count)

(3)　パラメータ

I/O	パラメータ		説明
I	BYTE	pdrv	物理ドライブ番号(0-9)
O	BYTE	*buff	読み出しバッファへのポインタ
I	DWORD	sector	読み出し開始セクタ番号
I	UINT	count	読み出すセクタ数(1-128)

(4)　機能

HMSDD の API 関数 R_usb_hmsc_StrgReadSector() をコールします。
R_usb_hmsc_StrgReadSector()の引数設定は以下の通りです。

引数	設定値
uint16_t side	pdrv
uint8_t *buff	buff
uint32_t secno	sector
uint16_t seccnt	(uint16_t)count
uint32_t trans_byte	count * _MIN_SS

USB 読み出しシーケンスが完了するまで、本関数内でループしてスケジューラを動作します。
途中で USB 切断を検出した場合は、RES_ERROR を返却します。

(5)　戻り値

戻り値	意味
RES_OK	正常終了
RES_ERROR	読み出し中にエラーが発生
RES_PARERR	コマンドが不正（サンプルでは未使用）
RES_NOTRDY	ストレージデバイスが動作可能状態ではない (サンプルでは未使用)

3.14.42　デバイスの書き込み

disk_write

(1)　概要

デバイスの書き込み

(2)　C 言語形式

DRESULT disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)

(3)　パラメータ

I/O	パラメータ		説明
I	BYTE	pdrv	物理ドライブ番号(0-9)
I	BYTE	*buff	書き込むデータへのポインタ
I	DWORD	sector	書き込み開始セクタ番号
I	UINT	count	書き込むセクタ数(1-128)

(4)　機能

HMSDD の API 関数 R_usb_hmsc_StrgWriteSector()をコールします。
R_usb_hmsc_StrgWriteSector()の引数設定は以下の通りです。

引数	設定値
uint16_t side	pdrv
const uint8_t *buff	buff
uint32_t secno	sector
uint16_t seccnt	(uint16_t)count
uint32_t trans_byte	count * _MIN_SS

USB 書き込みシーケンスが完了するまで、本関数内でループしてスケジューラを動作します。
途中で USB 切断を検出した場合は、RES_ERROR を返却します。

(5)　戻り値

戻り値	意味
RES_OK	正常終了
RES_ERROR	書き込み中にエラーが発生
RES_PARERR	コマンドが不正（サンプルでは未使用）
RES_NOTRDY	ストレージデバイスが動作可能状態ではない (サンプルでは未使用)

3.14.43　その他のデバイス制御

disk_ioctl

(1)　概要

その他のデバイス制御

(2)　C 言語形式

DRESULT disk_ioctl(BYTE pdrv, BYTE cmd, void* buff)

(3)　パラメータ

I/O	パラメータ		説明
I	BYTE	pdrv	物理ドライブ番号(0-9)
I	BYTE	cmd	制御コマンド
I/O	uint8_t	*buff	データ受け渡しバッファ

(4)　機能

全てのコマンドに対して処理をせず、RES_OK を返却します。

(5)　戻り値

戻り値	意味
RES_OK	正常終了
RES_ERROR	エラーが発生（サンプルでは未使用）
RES_PARERR	コマンドが不正（サンプルでは未使用）
RES_NOTRDY	ストレージデバイスが動作可能状態ではない（サンプルでは未使用）

3.14.44　日付と時刻の取得

get_fattime

(1)　概要

日付と時刻の取得

(2)　C 言語形式

DWORD get_fattime(void)

(3)　パラメータ

なし

(4)　機能

日付と時刻情報は設定せず、0x00000000 を返却します。

(5)　戻り値

現在のローカルタイムを DWORD 値にパックして返します。ビットフィールドは下記の通りです。

戻り値	意味
bit 31:25	1980 年を起点とした年を 0..127 でセット
bit 24:21	月を 1..12 の値でセット
bit 20:16	日を 1..31 の値でセット
bit 15:11	時を 0..23 の値でセット
bit 10:5	分を 0..59 の値でセット
bit 4:0	秒/2 を 0..29 の値でセット

4. CAN サンプルソフト

4.1 概要

本章では、評価ボード上に実装される CAN コントローラ 1 チャネル(CAN1)を制御する CAN ドライバを使用した通信を行うサンプルプログラムについて説明します。

CAN インターフェースサンプルプログラムの特長を以下に示します。
PC と接続し、ターミナルソフトを用いることで、CAN 通信を用いたテストを行うことができます。

・送信機能
　送信バッファを使用したメッセージ送信
　送受信 FIFO(送信モード)バッファを使用したメッセージ送信
・受信機能
　受信バッファを使用したメッセージ受信
　受信 FIFO バッファを使用したメッセージ受信
　送受信 FIFO(受信モード)バッファを使用したメッセージ受信
・テスト機能
　セルフテストモード 0（外部ループバックモード）
　　送信バッファ → 受信バッファ
　　送信バッファ → 受信 FIFO バッファ
　　送信バッファ → 送受信 FIFO(受信モード)バッファ
　　送受信 FIFO(送信モード)バッファ → 送受信 FIFO(受信モード)バッファ
　セルフテストモード 1（内部ループバックモード）
　　送信バッファ → 受信バッファ
　　送信バッファ → 受信 FIFO バッファ
　　送信バッファ → 送受信 FIFO(受信モード)バッファ
　　送受信 FIFO(送信モード)バッファ → 送受信 FIFO(受信モード)バッファ
・通信速度
　1Mbps、500Kbps、125Kbps に対応（メニューより選択可）

制限事項

本サンプルプログラムには以下の制限事項があります。
(1) 使用チャネル固定。（CAN1）
(2) メッセージフォーマット固定。（標準 ID(0x120)、データ・フレーム）
(3) 受信ルール固定。
　受信ルール・テーブルページ番号：0
　受信ルール数：1（テーブル番号：0）
　受信ルール ID：標準 ID(0x120)、データ・フレーム
(4) 使用バッファ固定。
　送信バッファ番号：0
　受信バッファ番号：1
　受信 FIFO バッファ番号：0

送受信 FIFO(送信モード)バッファ番号：0
送受信 FIFO(送信モード)バッファにリンクさせる送信バッファ番号：2
送受信 FIFO(受信モード)バッファ番号：1

(5) その他

以下の機能については対応していません。

送信アボート、送信キューによる送信、送信履歴機能、ゲートウェイ機能、テスト機能(標準テストモード、リッスンオンリモード、RAM テスト、チャネル間通信テスト)、RSCAN RAM のエラー検出／訂正

対象デバイス

EC-1

本サンプルを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

4.2 構造体/共用体/列挙型一覧

以下に、サンプルコードで使用する構造体/共用体/列挙型を示します。

表 4-1 tx_data_t構造体

メンバ名	内容
uint32_t num;	number
uint8_t * data;	data pointer
uint32_t send_num;	number of send message

表 4-2 rx_data_t構造体

メンバ名	内容
uint32_t num;	number
uint8_t * data;	data pointer

4.3　関数一覧

「表 4-3　関数一覧」を以下に示します。

表 4-3　関数一覧

関数名	概要	スコープ	定義ファイル
main	メイン処理	global	main.c
can_main_init	CAN サンプル初期化処理	local	main.c
ecm_init	ECM 設定初期化	local	main.c
icu_init	割り込み設定	local	main.c
port_init	ポート設定初期化	local	main.c
soft_wait	ソフトウェイト	local	main.c
scifa_interrupt_source	SCIFA 割り込み処理有効	local	main.c
key_handler_callback	コールバック関数（SCIFA 受信 FIFO データフル割り込み）	local	main.c
menu	CAN テストメニュー	local	main.c
self_menu	CAN セルフテストメニュー	local	main.c
test_end	CAN テスト終了	local	main.c
can1_open	CAN モジュールオープン	local	main.c
user_callback_entry	コールバック関数情報構造体登録	local	main.c
select_interrupt_source	割り込み、コールバック関数の登録	local	main.c
interrupt_disable	割り込み処理を無効	local	main.c
creat_message_header	サンプルメッセージヘッダ作成	local	main.c
user_gl_err_callback	コールバック関数（CAN グローバル・エラー）	local	main.c
user_ch1_err_callback	コールバック関数（CAN1 エラー）	local	main.c
user_rx_fifo_callback	コールバック関数（CAN 受信 FIFO 割り込み）	local	main.c
user_ch1_tx_callback	コールバック関数（CAN1 送信割り込み）	local	main.c
user_ch1_rx_fifo_callback	コールバック関数(CAN1 送受信 FIFO 受信完了割り込み)	local	main.c
tx_demo_buffer	送信バッファを使ったメッセージ送信処理	local	main.c
tx_demo_fifo	送受信 FIFO バッファ（送信モード）を使ったメッセージ送信処理	local	main.c
rx_demo_buffer	受信バッファを使ったメッセージ受信処理	local	main.c
rx_demo_fifo	送受信 FIFO バッファ（受信モード）を使ったメッセージ受信処理	local	main.c
rx_demo_rx_fifo	受信 FIFO バッファを使ったメッセージ受信処理	local	main.c
trx_demo_fifo	メッセージを受信しながらメッセージを送信するテスト	local	main.c
selftest_buf_to_buf	送信バッファを使ってメッセージを送信し、受信バッファでメッセージを受信するテスト	local	main.c
selftest_buf_to_rx_fifo	送信バッファを使ってメッセージを送信し、受信 FIFO バッファでメッセージを受信するテスト	local	main.c
selftest_buf_to_fifo	送信バッファを使ってメッセージを送信し、送受信 FIFO バッファ（受信モード）でメッセージを受信するテスト	local	main.c
selftest_fifo_to_fifo	送受信 FIFO バッファ（送信モード）を使ってメッセージを送信し、送受信 FIFO バッファ（受信モード）でメッセージを受信するテスト	local	main.c
demo_result	処理結果を出力	local	main.c

関数名	概要	スコープ	定義ファイル
demo_init	使用する変数等の初期化	local	main.c
demo_end	使用したバッファ情報をクリア	local	main.c
rx_rule	CAN 受信ルール設定	local	main.c
send_termination_code	メッセージ終了コードの送信	local	main.c
output_tx_msg_format	送信メッセージの形式を出力	local	main.c
output_tx_msg_data	送信メッセージを出力	local	main.c
output_rx_msg_data	受信メッセージを出力	local	main.c
output_rx_msg_format	受信メッセージの形式を出力	local	main.c
clear_status	サンプルプログラムで使用しているフラグをクリア	local	main.c
write_buffer	送信バッファ関連レジスタに送信メッセージを書き込む処理	local	main.c
write_fifo	送受信 FIFO バッファ関連レジスタに送信メッセージを書き込む処理	local	main.c
read_buffer	受信バッファから受信メッセージを読み出す処理	local	main.c
read_rx_fifo	受信 FIFO バッファから受信メッセージを読み出す処理	local	main.c
read_fifo	送受信 FIFO バッファから受信メッセージを読み出す処理	local	main.c
set_fifo_buffer	送受信 FIFO バッファの登録	local	main.c
rx_fifo_mode	受信 FIFO バッファの登録	local	main.c
get_error_status	割り込み要因取得	local	main.c

4.4　関数詳細

4.4.1　main

(1)　概要

サンプルプログラムメイン処理

(2)　C 言語形式

int main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。
処理内容は　4.5.1 メイン処理　を参照してください。

(5)　戻り値

戻り値	意味
0	CANサンプルプログラム終了

4.4.2 can_main_init

(1) 概要

CAN サンプル初期化処理

(2) C 言語形式

static void can_main_init (void);

(3) パラメータ

なし

(4) 機能

変数の初期化を行います。

(5) 戻り値

なし

4.4.3 ecm_init

(1) 概要

ECM 設定初期化

(2) C 言語形式

static void ecm_init (void);

(3) パラメータ

なし

(4) 機能

ECM 設定の初期化を行います。

(5) 戻り値

なし

4.4.4 icu_init

(1) 概要

割り込み設定

(2) C 言語形式

static void icu_init (void);

(3) パラメータ

なし

(4) 機能

割り込み処理を有効にします。

(5) 戻り値

なし

4.4.5 port_init

(1) 概要

ポート設定初期化

(2) C 言語形式

static void port_init(void);

(3) パラメータ

なし

(4) 機能

ポート設定の初期化を行います。

(5) 戻り値

なし

4.4.6 soft_wait

(1) 概要

ソフトウェアウェイト処理

(2) C 言語形式

static void soft_wait(void);

(3) パラメータ

なし

(4) 機能

NOP を使用したソフトウェアウェイト処理を行います。

(5) 戻り値

なし

4.4.7　scifa_interrupt_source

(1)　概要

SCIFA 割り込み処理の有効

(2)　C 言語形式

static void scifa_interrupt_source(void);

(3)　パラメータ

なし

(4)　機能

SCIFA 割り込み処理（受信 FIFO データフル（RDF））を有効にします。

(5)　戻り値

なし

4.4.8 key_handler_callback

(1) 概要

コールバック関数（SCIFA 受信 FIFO データフル割り込み）

(2) C 言語形式

static void key_handler_callback(void);

(3) パラメータ

なし

(4) 機能

SCIFA 受信 FIFO データフル割り込み時に実行されるコールバック関数です。

(5) 戻り値

なし

4.4.9 menu

(1) 概要

CAN テストメニュー

(2) C 言語形式

static void menu(void);

(3) パラメータ

なし

(4) 機能

CAN テストメニューを表示します。
表示されるメニュー内容については 4.6.5 サンプルプログラムの機能 を参照ください.

(5) 戻り値

なし

4.4.10　self_menu

(1)　概要

CAN セルフテストメニュー

(2)　C 言語形式

static void self_menu(void);

(3)　パラメータ

なし

(4)　機能

メインメニューにて[7]を選択した際のセルフテストメニューを表示します。
表示されるメニュー内容については 4.6.5 サンプルプログラムの機能 を参照ください.

(5)　戻り値

なし

4.4.11 test_end

(1) 概要

CAN テスト終了

(2) C 言語形式

static void test_end(void);

(3) パラメータ

なし

(4) 機能

CAN モジュールをクローズし、終了メッセージを表示します。

(5) 戻り値

なし

4.4.12 can1_open

(1) 概要

CAN モジュールオープン

(2) C 言語形式

static void can1_open(void);

(3) パラメータ

なし

(4) 機能

CAN モジュールをオープンします。

(5) 戻り値

なし

4.4.13　user_callback_entry

(1)　概要

コールバック関数情報構造体登録

(2)　C 言語形式

static void user_callback_entry (void);

(3)　パラメータ

なし

(4)　機能

can_callback_t 構造体（コールバック関数情報構造体）のメンバに使用するコールバック関数名を登録します。
使用しない場合は NULL を設定ください。

(5)　戻り値

なし

4.4.14 select_interrupt_source

(1) 概要

コールバック関数登録

(2) C 言語形式

static void select_interrupt_source(void);

(3) パラメータ

なし

(4) 機能

user_callback_entry()にて指定されたコールバック関数の登録を行います。

(5) 戻り値

なし

4.4.15 interrupt_disable

(1) 概要

割り込み処理無効

(2) C 言語形式

static void interrupt_disable (void);

(3) パラメータ

なし

(4) 機能

割り込み処理を無効にします。

(5) 戻り値

なし

4.4.16 creat_message_header

(1) 概要

サンプルメッセージヘッダ作成

(2) C 言語形式

static void creat_message_header (void);

(3) パラメータ

なし

(4) 機能

サンプルメッセージヘッダを作成します。

(5) 戻り値

なし

4.4.17　user_gl_err_callback

(1)　概要

コールバック関数（CAN グローバル・エラー）

(2)　C 言語形式

static void user_gl_err_callback (void);

(3)　パラメータ

なし

(4)　機能

CAN グローバル・エラー発生時に呼ばれるコールバック関数です。
処理については図 4-24 サンプルコードの CAN グローバル・エラー発生時に呼ばれるコールバック関数処理を参照ください。

(5)　戻り値

なし

4.4.18　user_ch1_err_callback

(1)　概要

コールバック関数（CAN1 エラー）

(2)　C 言語形式

static void user_ch1_err_callback (void);

(3)　パラメータ

なし

(4)　機能

CAN1 エラー発生時に呼ばれるコールバック関数です。
処理については図 4-25 サンプルコードの CAN1 エラー発生時に呼ばれるコールバック関数処理を参照ください。

(5)　戻り値

なし

4.4.19　user_rx_fifo_callback

(1)　概要

コールバック関数（CAN 受信 FIFO 割り込み）

(2)　C 言語形式

static void user_rx_fifo_callback (void);

(3)　パラメータ

なし

(4)　機能

CAN 受信 FIFO 割り込み発生時に呼ばれるコールバック関数です。
処理については図 4-26 サンプルコードの CAN 受信 FIFO 割込み発生時に呼ばれるコールバック関数処理を参照ください。

(5)　戻り値

なし

4.4.20　user_ch1_tx_callback

(1)　概要

コールバック関数（CAN1 送信割り込み）

(2)　C 言語形式

static void user_ch1_tx_callback (void);

(3)　パラメータ

なし

(4)　機能

CAN1 送信割り込み発生時に呼ばれるコールバック関数です。
処理については図 4-27 サンプルコードの CAN1 送信割込み発生時に呼ばれるコールバック関数処理を参照ください。

(5)　戻り値

なし

4.4.21　user_ch1_rx_fifo_callback

(1)　概要

コールバック関数（CAN1 送受信 FIFO 受信完了割り込み）

(2)　C 言語形式

static void user_ch1_rx_fifo_callback (void);

(3)　パラメータ

なし

(4)　機能

CAN1 送受信 FIFO 割り込み受信完了割り込み発生時に呼ばれるコールバック関数です。
処理については図 4-28 サンプルコードの CAN1 送受信 FIFO 受信完了割込み発生時に呼ばれるコールバック関数処理を参照ください。

(5)　戻り値

なし

4.4.22 tx_demo_buffer

(1) 概要

送信バッファを使ったメッセージ送信処理

(2) C 言語形式

static void tx_demo_buffer (void);

(3) パラメータ

なし

(4) 機能

送信バッファを使ったメッセージ送信処理です。
処理については図 4-2 サンプルコードの送信バッファを使ったメッセージ送信処理を参照ください。

(5) 戻り値

なし

4.4.23　tx_demo_fifo

(1)　概要

送受信 FIFO バッファ（送信モード）を使ったメッセージ送信処理

(2)　C 言語形式

static void tx_demo_fifo (void);

(3)　パラメータ

なし

(4)　機能

送受信 FIFO バッファ（送信モード）を使ったメッセージ送信処理です。
処理については図 4-3 サンプルコードの送受信 FIFO バッファ(送信モード)を使ったメッセージ送信処理を参照ください。

(5)　戻り値

なし

4.4.24 rx_demo_buffer

(1) 概要

受信バッファを使ったメッセージ受信処理

(2) C 言語形式

static void rx_demo_buffer (void);

(3) パラメータ

なし

(4) 機能

受信バッファを使ったメッセージ受信処理です。
処理については図 4-6 サンプルコードの受信バッファを使ったメッセージ受信処理を参照ください。

(5) 戻り値

なし

4.4.25　rx_demo_fifo

(1)　概要

送受信 FIFO バッファ（受信モード）を使ったメッセージ受信処理

(2)　C 言語形式

static void rx_demo_fifo (void);

(3)　パラメータ

なし

(4)　機能

送受信 FIFO バッファ（受信モード）を使ったメッセージ受信処理です。
処理については図 4-8 サンプルコードの送受信 FIFO バッファ(受信モード)を使ったメッセージ受信処理を参照ください。

(5)　戻り値

なし

4.4.26　rx_demo_rx_fifo

(1)　概要

受信 FIFO バッファを使ったメッセージ受信処理

(2)　C 言語形式

static void rx_demo_rx_fifo (void);

(3)　パラメータ

なし

(4)　機能

受信 FIFO バッファを使ったメッセージ受信処理です。
処理については図 4-7 サンプルコードの受信 FIFO バッファを使ったメッセージ受信処理を参照ください。

(5)　戻り値

なし

4.4.27　trx_demo_fifo

(1)　概要

メッセージを受信しながらメッセージを送信するテスト

(2)　C 言語形式

static void trx_demo_fifo (void);

(3)　パラメータ

なし

(4)　機能

メッセージを受信しながらメッセージを送信するテストです。
処理については図 4-12 サンプルコードのメッセージを受信しながらメッセージを送信するテストを参照ください。

(5)　戻り値

なし

4.4.28　selftest_buf_to_buf

(1)　概要

送信バッファを使ってメッセージを送信し、受信バッファでメッセージを受信するテスト

(2)　C 言語形式

static void selftest_buf_to_buf (void);

(3)　パラメータ

なし

図 4-12 サンプルコードのメッセージを受信しながらメッセージを送信するテスト（1/2）

(4)　機能

送信バッファを使ってメッセージを送信し、受信バッファでメッセージを受信するテストです。

処理については図 4-16、図 4-17 サンプルコードの送信バッファを使ってメッセージを送信し、受信バッファでメッセージを受信するテストを参照ください。

(5)　戻り値

なし

4.4.29　selftest_buf_to_rx_fifo

(1)　概要

送信バッファを使ってメッセージを送信し、受信 FIFO バッファでメッセージを受信するテスト

(2)　C 言語形式

static void selftest_buf_to_rx_fifo (void);

(3)　パラメータ

なし

(4)　機能

送信バッファを使ってメッセージを送信し、受信 FIFO バッファでメッセージを受信するテストです。
処理については図 4-18、図 4-19 サンプルコードの送信バッファを使ってメッセージを送信し、受信 FIFO バッファでメッセージを受信するテストを参照ください。

(5)　戻り値

なし

4.4.30　selftest_buf_to_fifo

(1)　概要

送信バッファを使ってメッセージを送信し、送受信 FIFO バッファ（受信モード）でメッセージを受信するテスト

(2)　C 言語形式

static void selftest_buf_to_fifo (void);

(3)　パラメータ

なし

(4)　機能

送信バッファを使ってメッセージを送信し、送受信 FIFO バッファ（受信モード）でメッセージを受信するテストです。

処理については図 4-20、図 4-21 サンプルコードの送信バッファを使ってメッセージを送信し、送受信 FIFO バッファ(受信モード)でメッセージを受信するテスト（2/2）を参照ください。

(5)　戻り値

なし

4.4.31　selftest_fifo_to_fifo

(1)　概要

送受信 FIFO バッファ（送信モード）を使ってメッセージを送信し、送受信 FIFO バッファ（受信モード）でメッセージを受信するテスト

(2)　C 言語形式

static void selftest_fifo_to_fifo (void);

(3)　パラメータ

なし

(4)　機能

送受信 FIFO バッファ（送信モード）を使ってメッセージを送信し、送受信 FIFO バッファ（受信モード）でメッセージを受信するテストです。

処理については図 4-22、図 4-23 サンプルコードの送受信 FIFO バッファ(送信モード)を使ってメッセージを送信し、送受信 FIFO バッファ(受信モード)でメッセージを受信するテストを参照ください。

(5)　戻り値

なし

4.4.32　demo_result

(1)　概要

処理結果を出力。

(2)　C 言語形式

static void demo_result (void);

(3)　パラメータ

なし

(4)　機能

テスト結果を表示します。

(5)　戻り値

なし

4.4.33 demo_init

(1) 概要

使用する変数等の初期化

(2) C 言語形式

static void demo_init (void);

(3) パラメータ

なし

(4) 機能

使用する変数等の初期化を行います。

(5) 戻り値

なし

4.4.34 demo_end

(1) 概要

使用したバッファ情報をクリア

(2) C 言語形式

static void demo_end(void);

(3) パラメータ

なし

(4) 機能

使用したバッファ情報のクリアを行います。

(5) 戻り値

なし

4.4.35　rx_rule

(1)　概要

CAN 受信ルール設定

(2)　C 言語形式

static void rx_rule(uint32_t buf_type);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	buf_type	バッファの形式

(4)　機能

CAN 受信ルールを設定する処理です。
受信ルール情報を can_rx_rule_t 構造体に格納し、R_CAN_RxSet()にて設定を行います。

EC-1 ユーザーズマニュアル　ハードウェア編　27.5.1　受信ルールテーブルを用いたデータ処理
27.9.1.4 受信ルールの設定　を参照して下さい。

(5)　戻り値

なし

4.4.36　send_termination_code

(1)　概要

メッセージ終了コードの送信

(2)　C 言語形式

static void send_termination_code(void);

(3)　パラメータ

なし

(4)　機能

メッセージ終了コードの送信を行います。

(5)　戻り値

なし

4.4.37 output_tx_msg_format

(1) 概要

送信メッセージの形式を出力

(2) C 言語形式

static void output_tx_msg_format(void);

(3) パラメータ

なし

(4) 機能

送信メッセージの形式を表示します。

(5) 戻り値

なし

4.4.38　output_tx_msg_data

(1)　概要

送信メッセージを出力

(2)　C 言語形式

static void output_tx_msg_data(uint32_t offset, uint32_t num);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	offset	メッセージバッファを送信オフセット
I	uint32_t	num	送信メッセージ数

(4)　機能

引数にて指定されたメッセージが送信されます

(5)　戻り値

なし

4.4.39　output_rx_msg_data

(1)　概要

受信メッセージを出力

(2)　C 言語形式

static void output_rx_msg_data(uint32_t offset, uint32_t num)

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	offset	メッセージバッファを受信オフセット
I	uint32_t	num	受信メッセージ数

(4)　機能

引数にて指定された受信メッセージを出力します。

(5)　戻り値

なし

4.4.40 output_rx_msg_format

(1) 概要

受信メッセージの形式を出力

(2) C 言語形式

static void output_rx_msg_format (void)

(3) パラメータ

なし

(4) 機能

受信メッセージの形式を出力します。

(5) 戻り値

なし

4.4.41 clear_status

(1) 概要

サンプルプログラムで使用しているフラグをクリア

(2) C 言語形式

static void clear_status(void);

(3) パラメータ

なし

(4) 機能

サンプルプログラムで使用している下記フラグをクリアします。

- グローバル・エラーフラグ
- チャネルエラーフラグ
- 送信完了フラグ
- 送受信 FIFO フラグ
- 受信 FIFO フラグ

また、受信メッセージ格納バッファの初期化を行います。

(5) 戻り値

なし

4.4.42　write_buffer

(1)　概要

送信バッファ関連レジスタに送信メッセージを書き込む処理

(2)　C 言語形式

static uint32_t write_buffer(uint32_t msg_type, tx_data_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　msg_type	メッセージタイプ
I	tx_data_t　* obj	送信情報のポインタ

(4)　機能

送信バッファ関連レジスタに送信メッセージを書き込む処理を行います。
処理については図 4-4 サンプルコードの送信バッファ関連レジスタに送信メッセージを書き込む処理を参照してください。

(5)　戻り値

戻り値	意味
RET_OK	送信完了
RET_ERROR	エラー
RET_BUSY	送信バッファ使用中

4.4.43　write_fifo

(1)　概要

送受信 FIFO バッファ関連レジスタに送信メッセージを書き込む処理

(2)　C 言語形式

static uint32_t write_fifo(uint32_t msg_type, tx_data_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　msg_type	メッセージタイプ
I	tx_data_t　* obj	送信情報のポインタ

(4)　機能

送受信 FIFO バッファ関連レジスタに送信メッセージを書き込む処理を行います。
処理については図 4-5 サンプルコードの送受信 FIFO バッファ関連レジスタに送信メッセージを書き込む処理を参照してください。

(5)　戻り値

戻り値	意味
RET_OK	送信完了
RET_ERROR	エラー
RET_BUSY	送信バッファ使用中

4.4.44　read_buffer

(1)　概要

受信バッファから受信メッセージを読み出す処理

(2)　C 言語形式

static uint32_t read_buffer(rx_data_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	rx_data_t　　* obj	受信データ格納情報のポインタ

(4)　機能

受信バッファから受信メッセージを読み出す処理を行います。
処理については図 4-9 サンプルコードの受信バッファから受信メッセージを読み出す処理を参照してください。

(5)　戻り値

戻り値	意味
MSG_NONE	未受信
MSG_BODY	受信メッセージ本体
MSG_END	デリミタコード

4.4.45　read_rx_fifo

(1)　概要

受信 FIFO バッファから受信メッセージを読み出す処理

(2)　C 言語形式

static void read_rx_fifo(rx_data_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	rx_data_t　　* obj	受信データ格納情報のポインタ

(4)　機能

受信 FIFO バッファから受信メッセージを読み出す処理を行います。

処理については図 4-10 サンプルコードの受信 FIFO バッファから受信メッセージを読み出す処理を参照してください。

(5)　戻り値

なし

4.4.46　read_fifo

(1)　概要

送受信 FIFO バッファから受信メッセージを読み出す処理

(2)　C 言語形式

static void read_fifo(uint32_t ch, rx_data_t * obj);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t　ch	チャンネル
I	rx_data_t　* obj	受信データ格納情報のポインタ

(4)　機能

受信 FIFO バッファから受信メッセージを読み出す処理を行います。
処理については図 4-11 サンプルコードの送受信 FIFO バッファから受信メッセージを読み出す処理を参照してください。

(5)　戻り値

なし

4.4.47　set_fifo_buffer

(1)　概要

送受信 FIFO バッファの登録

(2)　C 言語形式

static void set_fifo_buffer(uint32_t ch, uint32_t mode, uint32_t condition);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャンネル
I	uint32_t	mode	FIFOバッファモード
I	uint32_t	condition	トリガ条件

(4)　機能

送受信 FIFO バッファの登録を行います。

(5)　戻り値

なし

4.4.48　rx_fifo_mode

(1)　概要

受信 FIFO バッファの登録

(2)　C 言語形式

static void rx_fifo_mode(uint32_t ch, uint32_t condition);

(3)　パラメータ

I/O	パラメータ		説明
I	uint32_t	ch	チャンネル
I	uint32_t	condition	トリガ条件

(4)　機能

受信 FIFO バッファの登録を行います。

(5)　戻り値

なし

4.4.49 get_error_status

(1) 概要

割り込み要因取得

(2) C 言語形式

static void get_error_status(void);

(3) パラメータ

なし

(4) 機能

R_CAN_GetInterruptSource ()関数を使い、各割り込み要因を取得します。

(5) 戻り値

なし

4.5　フローチャート

4.5.1　メイン処理

本サンプルプログラムは、メニューから確認項目を選択する方式をとっています。
メニューの内容は、4.6 チュートリアル 4.6.5 サンプルプログラムの機能を参照して下さい。
図 4-1 にサンプルコードのメイン処理のフローチャートを示します。

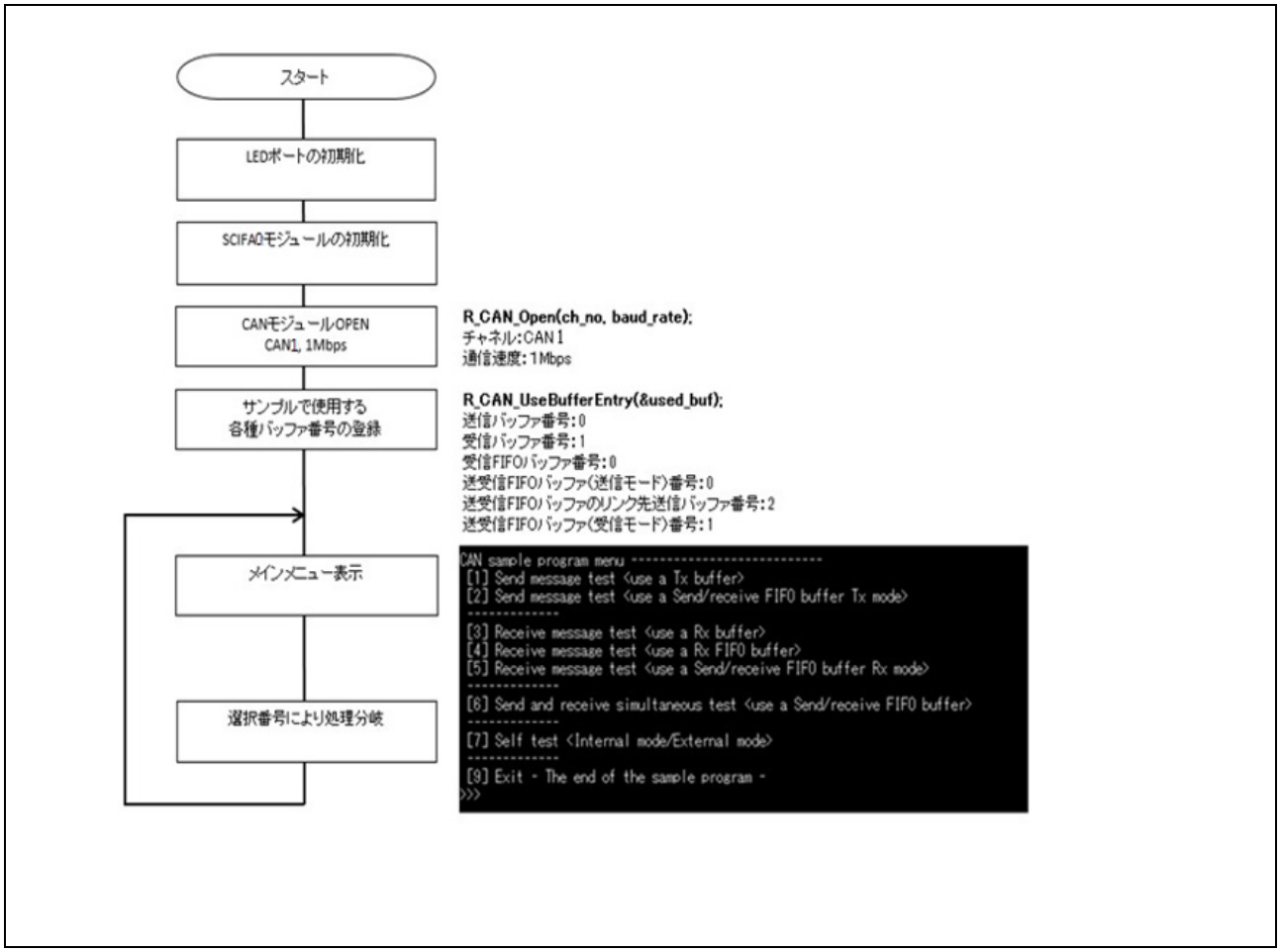


図 4-1 サンプルコードのメイン処理

4.5.2　送信テスト

送信テストは、「送信バッファを使ったメッセージ送信」と「送受信FIFO バッファ(送信モード)を使ったメッセージ送信」の2 通りがあり、メニューから選択することができます。

メニューの内容は、4.6 チュートリアル 4.6.5 サンプルプログラムの機能を参照して下さい。

各テストを行うには、以下の関数を使用します。

- void tx_demo_buffer(void)

  送信バッファを使ったメッセージ送信処理

- void tx_demo_fifo(void)

  送受信FIFO バッファ(送信モード)を使ったメッセージ送信処理

- uint32_t write_buffer(uint32_t msg_type, tx_data_t * obj)

  送信バッファ関連レジスタに送信メッセージを書き込む処理

- uint32_t write_fifo(uint32_t msg_type, tx_data_t * obj)

  送受信FIFO バッファ関連レジスタに送信メッセージを書き込む処理

図 4-2 ～ 図 4-5 にそれぞれの機能のフローチャートを示します。

- 送信バッファを使ったメッセージ送信処理

  関数名：void tx_demo_buffer(void)

  以下に送信バッファを使ったメッセージ送信処理のフローチャートを示します。

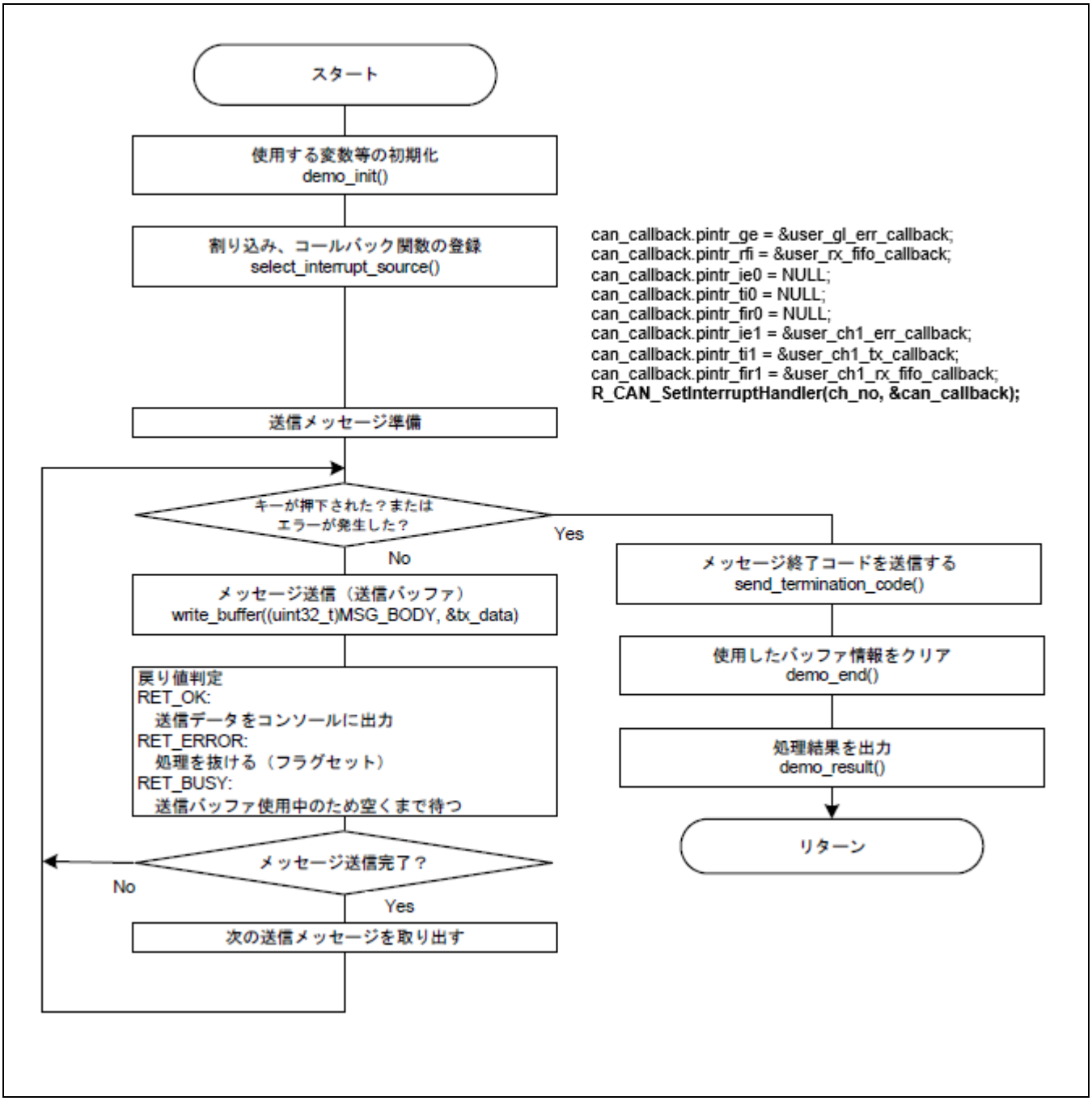


図 4-2 サンプルコードの送信バッファを使ったメッセージ送信処理

- 送受信 FIFO バッファ(送信モード)を使ったメッセージ送信処理

  関数名：void tx_demo_fifo(void)

  以下に、送受信 FIFO バッファ(送信モード)を使ったメッセージ送信処理のフローチャートを示します。

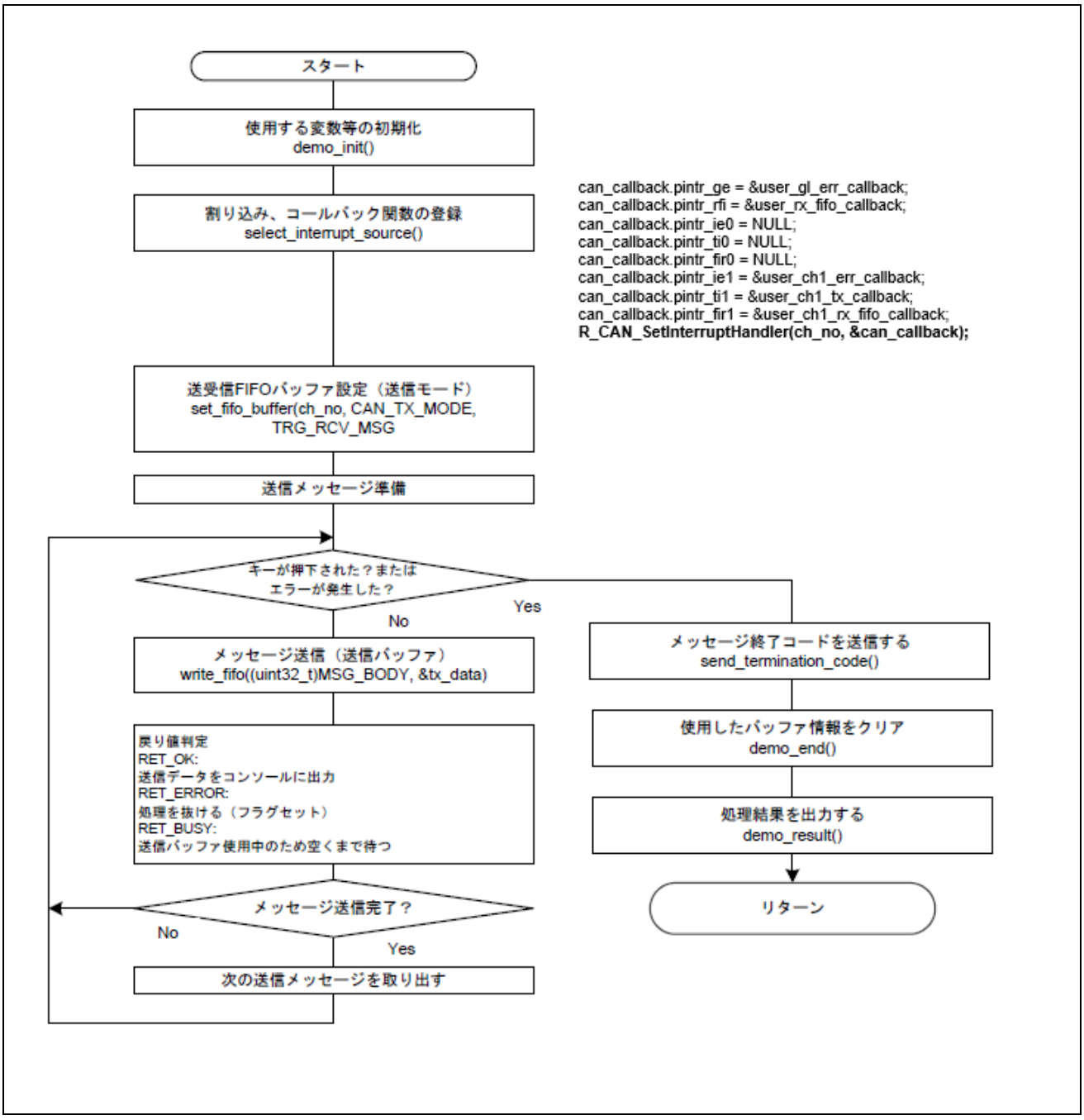


図 4-3 サンプルコードの送受信FIFOバッファ(送信モード)を使ったメッセージ送信処理

- 送信バッファ関連レジスタに送信メッセージを書き込む処理

  関数名：uint32_t write_buffer(uint32_t msg_type, tx_data_t * obj)

  以下に、送信バッファ関連レジスタに送信メッセージを書き込む処理のフローチャートを示します

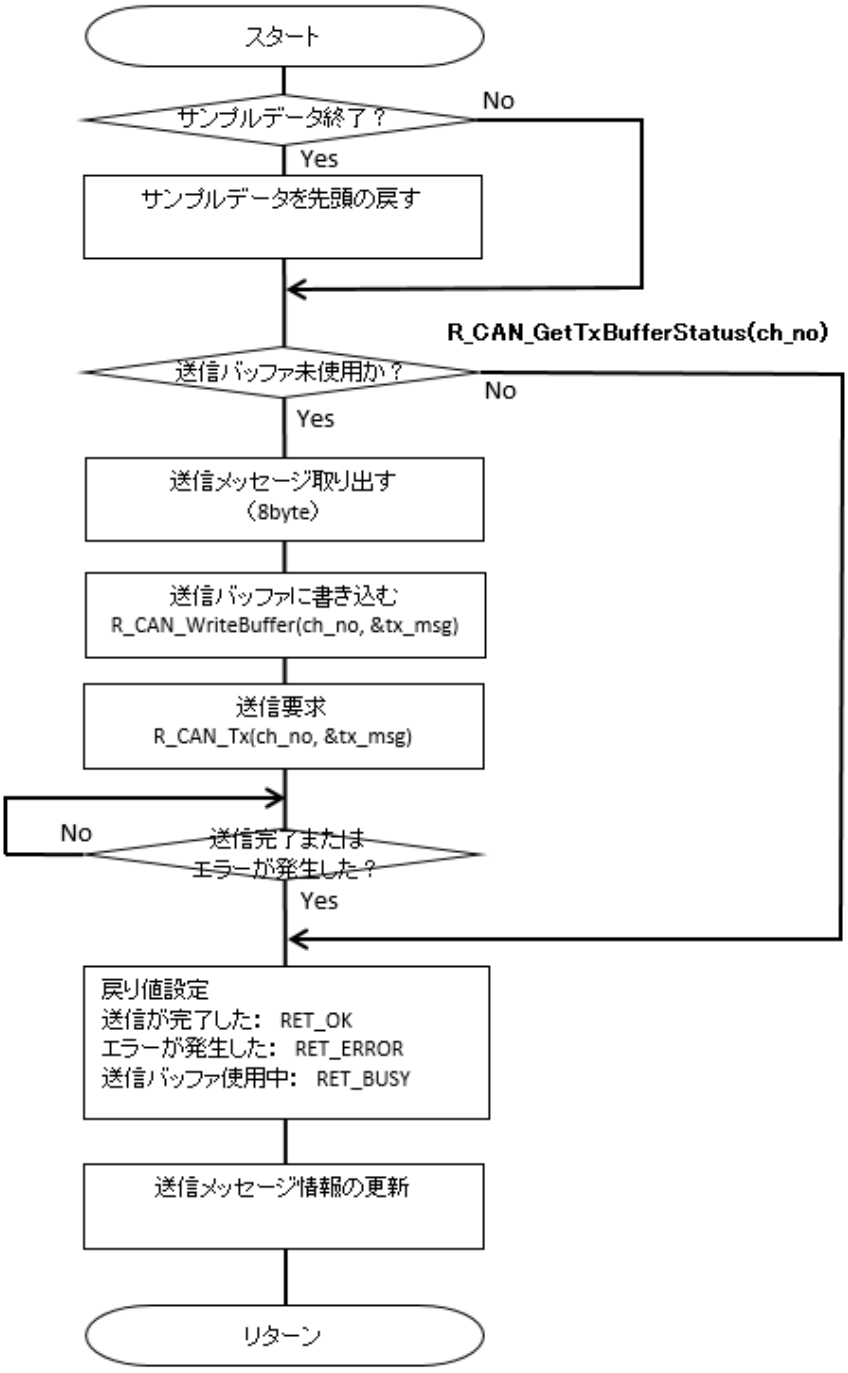


図 4-4 サンプルコードの送信バッファ関連レジスタに送信メッセージを書き込む処理

- 送受信 FIFO バッファ関連レジスタに送信メッセージを書き込む処理

  関数名：uint32_t write_fifo(uint32_t msg_type, tx_data_t * obj)

  以下に、送受信 FIFO バッファ関連レジスタに送信メッセージを書き込む処理のフローチャートを示します。

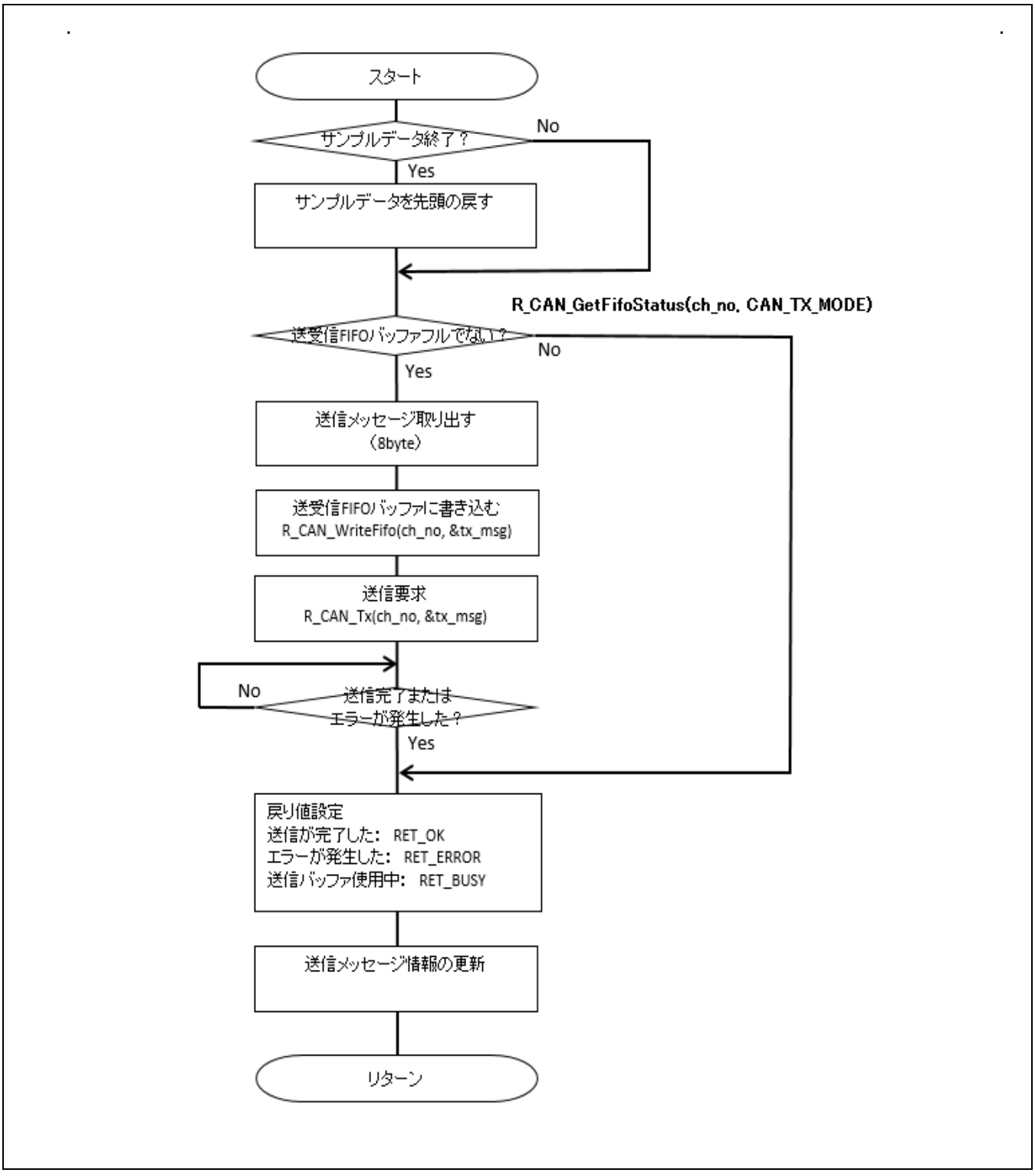


図 4-5 サンプルコードの送受信 FIFO バッファ関連レジスタに送信メッセージを書き込む処理

4.5.3　受信テスト

受信テストは、「受信バッファを使ったメッセージ受信」、「受信FIFO バッファを使ったメッセージ受信」および「送受信 FIFO バッファ(受信モード)を使ったメッセージ受信」をメニューから選択することができます。

メニューの内容は、4.6 チュートリアル 4.6.5 サンプルプログラムの機能を参照して下さい。

各テストを行うには以下の関数を使用します。

- void rx_demo_buffer(void)

  受信バッファを使ったメッセージ受信処理

- void rx_demo_rx_fifo(void)

  受信FIFO バッファを使ったメッセージ受信処理

- void rx_demo_fifo(void)

  送受信FIFO バッファ(受信モード)を使ったメッセージ受信処理

- uint32_t read_buffer(rx_data_t * obj)

  受信バッファから受信メッセージを読み出す処理

- void read_rx_fifo(rx_data_t * obj)

  受信FIFO バッファから受信メッセージを読み出す処理

- void read_fifo(uint32_t ch, rx_data_t * obj)

  送受信FIFO バッファから受信メッセージを読み出す処理

図 4-6 ～ 図 4-11 にそれぞれの機能のフローチャートを示します。

- 受信バッファを使ったメッセージ受信処理

  関数名：void rx_demo_buffer(void)

  以下に、受信バッファを使ったメッセージ受信処理のフローチャートを示します。

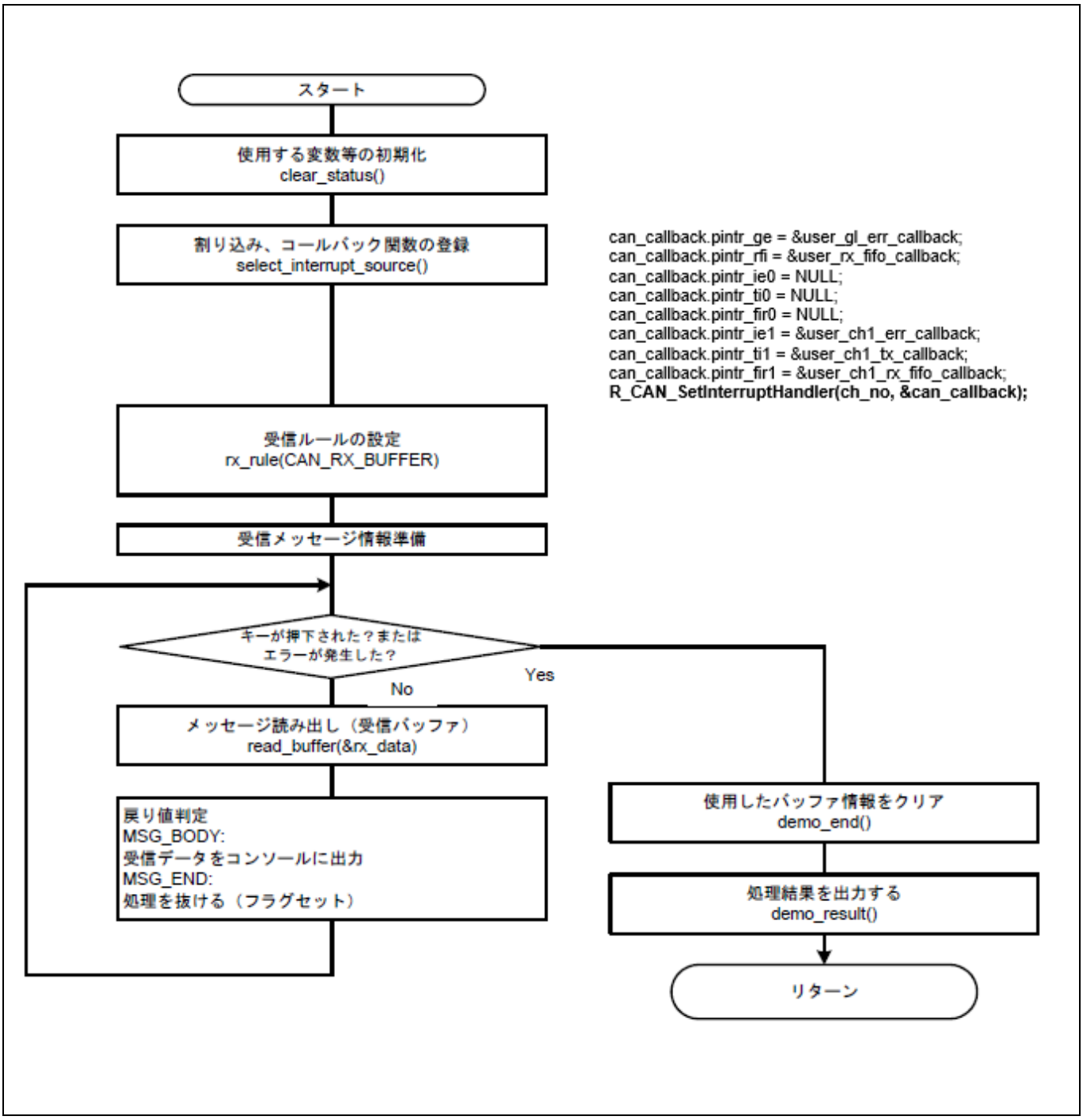


図 4-6 サンプルコードの受信バッファを使ったメッセージ受信処理

- 受信 FIFO バッファを使ったメッセージ受信処理

  関数名：void rx_demo_rx_fifo(void)

  以下に、受信FIFO バッファを使ったメッセージ受信処理のフローチャートを示します

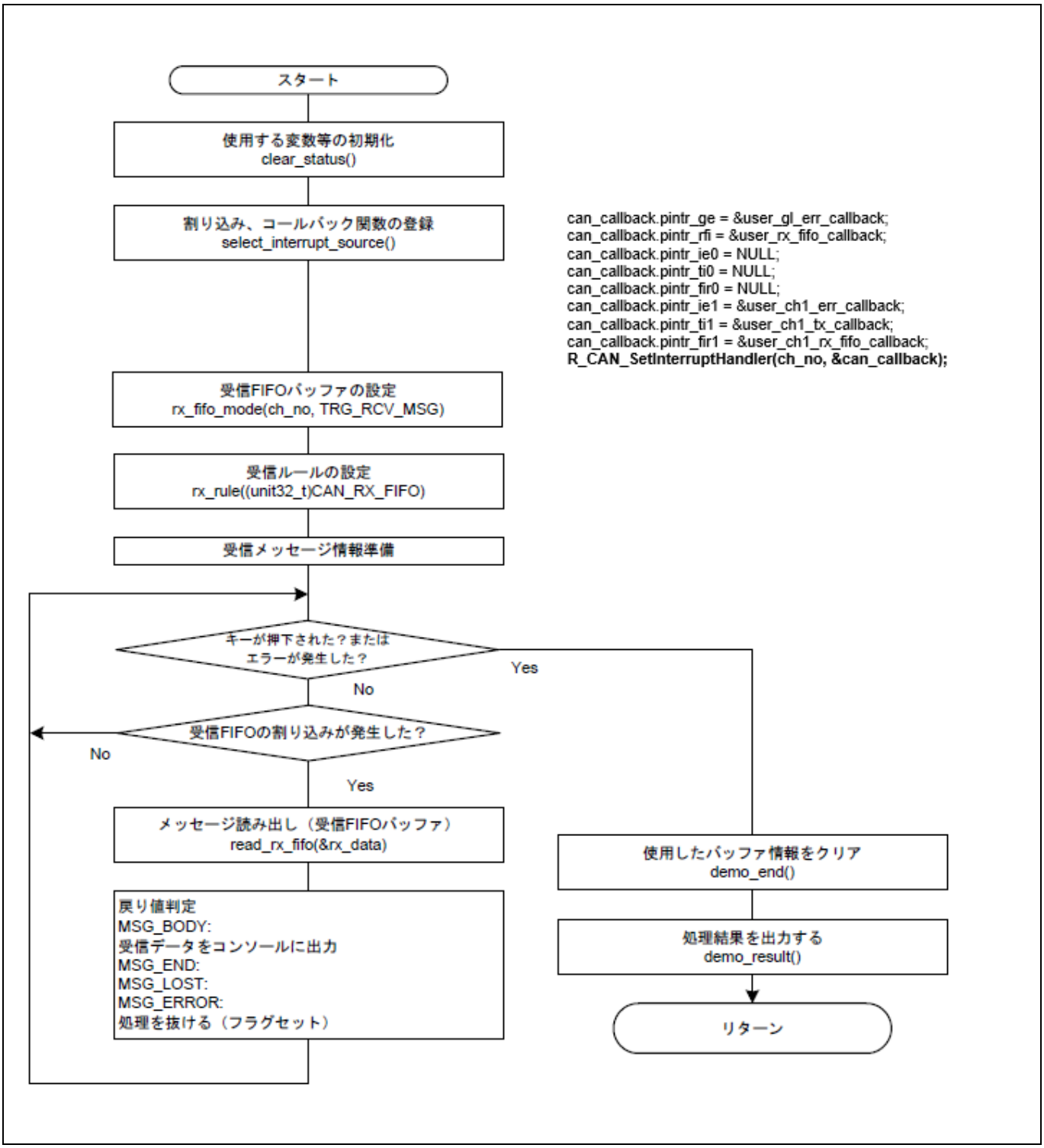


図 4-7 サンプルコードの受信 FIFO バッファを使ったメッセージ受信処理

- 送受信 FIFO バッファ（受信モード）を使ったメッセージ受信処理

  関数名：void rx_demo_fifo(void)

  以下に、送受信FIFO バッファ(受信モード)を使ったメッセージ受信処理のフローチャートを示します。

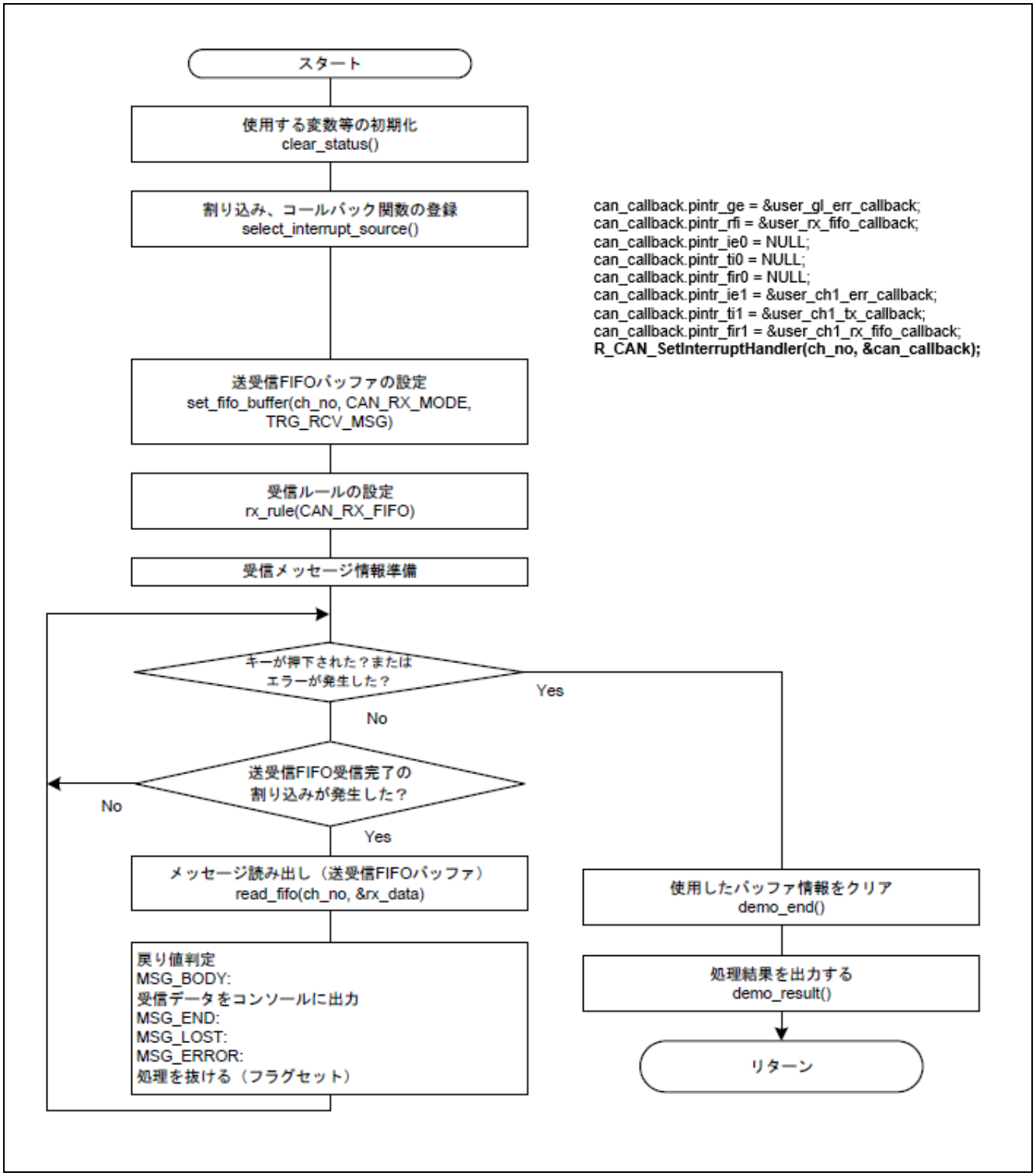


図 4-8 サンプルコードの送受信 FIFO バッファ(受信モード)を使ったメッセージ受信処理

- 受信バッファから受信メッセージを読み出す処理

  関数名：uint32_t read_buffer(rx_data_t * obj)

  以下に、受信バッファから受信メッセージを読み出す処理のフローチャートを示します。

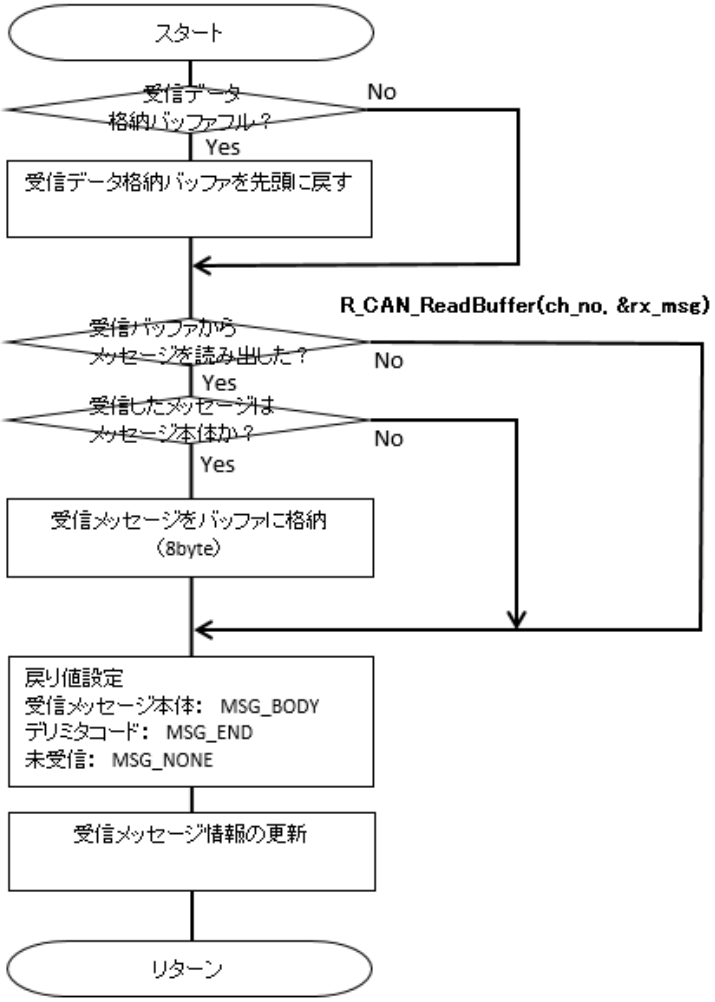


図 4-9 サンプルコードの受信バッファから受信メッセージを読み出す処理

- 受信 FIFO バッファから受信メッセージを読み出す処理

  関数名：void read_rx_fifo(rx_data_t * obj)

  以下に、受信FIFO バッファから受信メッセージを読み出す処理のフローチャートを示します

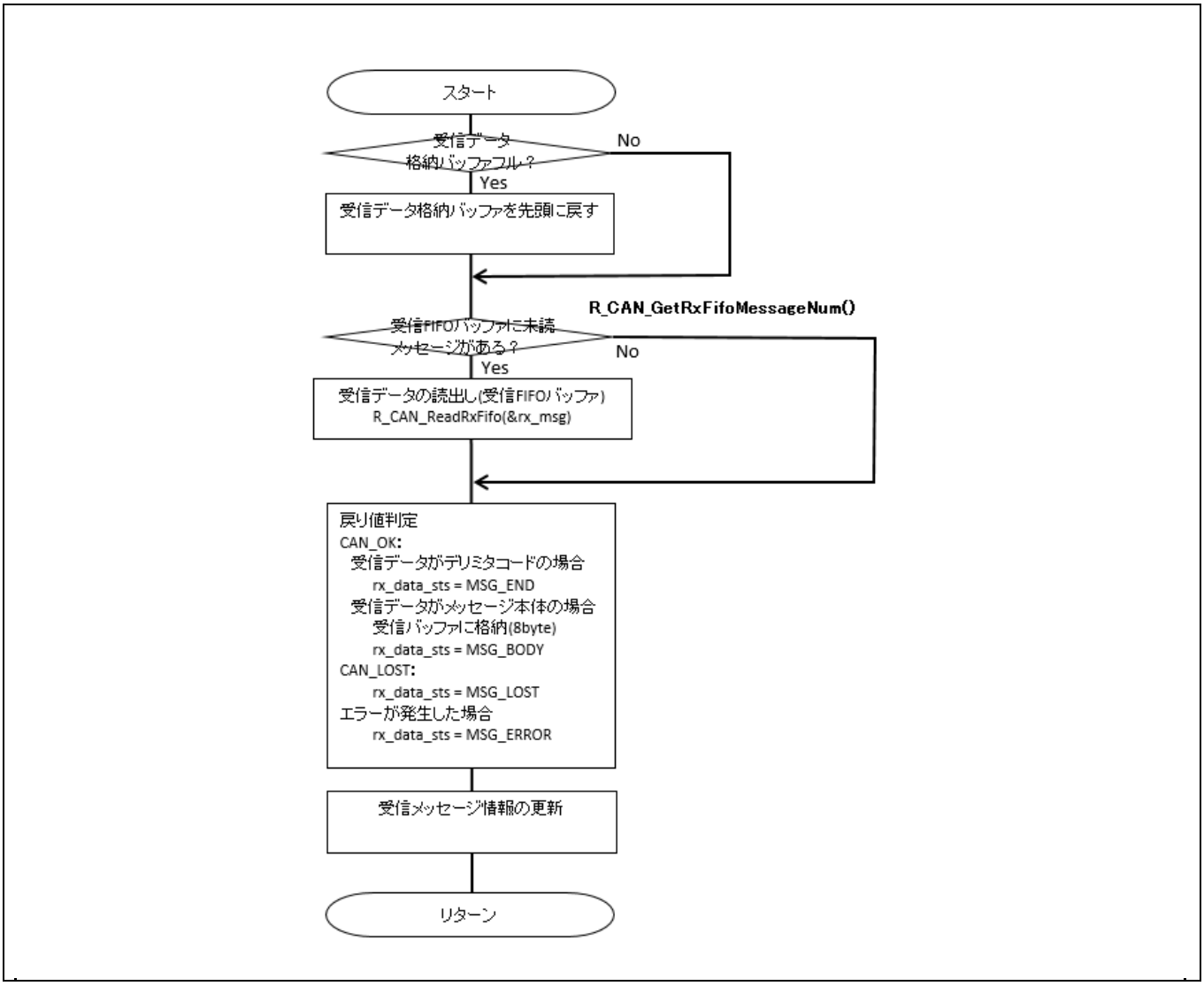


図 4-10 サンプルコードの受信 FIFO バッファから受信メッセージを読み出す処理

- 送受信 FIFO バッファから受信メッセージを読み出す処理

  関数名：void read_fifo(uint32_t ch, rx_data_t * obj)

  以下に、送受信FIFO バッファから受信メッセージを読み出す処理のフローチャートを示します。

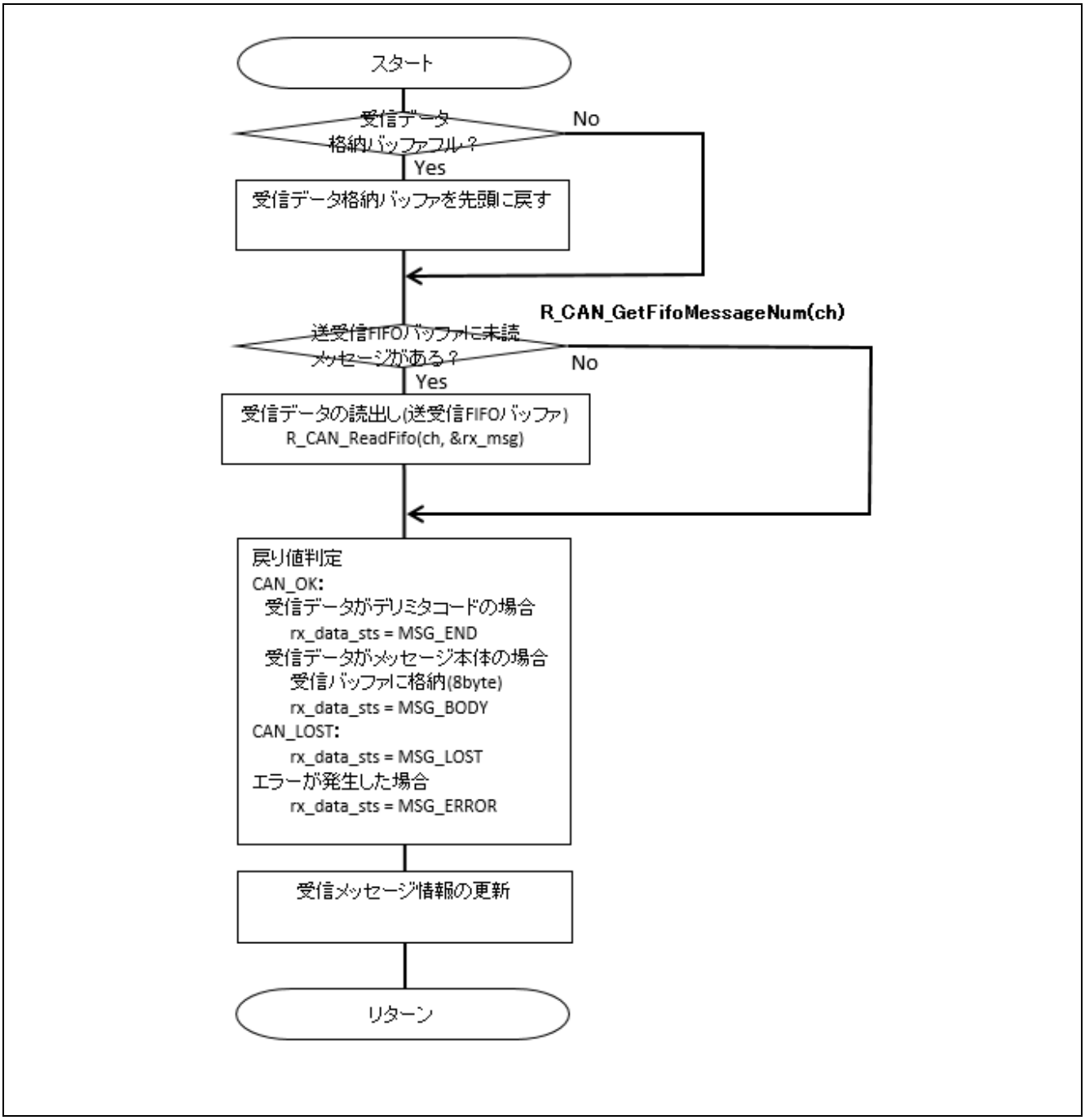


図 4-11 サンプルコードの送受信 FIFO バッファから受信メッセージを読み出す処理

4.5.4　受信を受け付けながらメッセージ送信を行うテスト

メッセージを受信しながらメッセージを送信するテストです。

本サンプルでは、送受信 FIFO バッファ(受信モード)を使ってメッセージを受信し、送受信 FIFO バッファ(送信モード)を使ってメッセージを送信します。

メニューの内容は、4.6 チュートリアル 4.6.5 サンプルプログラムの機能を参照して下さい。

本テストを行うには以下の関数を使用します。

- void trx_demo_fifo (void)

  メッセージを受信しながらメッセージを送信するテスト

図 4-12 にサンプルコードのメッセージを受信しながらメッセージを送信するテストのフローチャートを示します。

- メッセージを受信しながらメッセージを送信するテスト送受信 FIFO バッファから受信メッセージを読み出す処理

  関数名：void trx_demo_fifo (void)

  以下に、メッセージを受信しながらメッセージを送信するテストのフローチャートを示します。

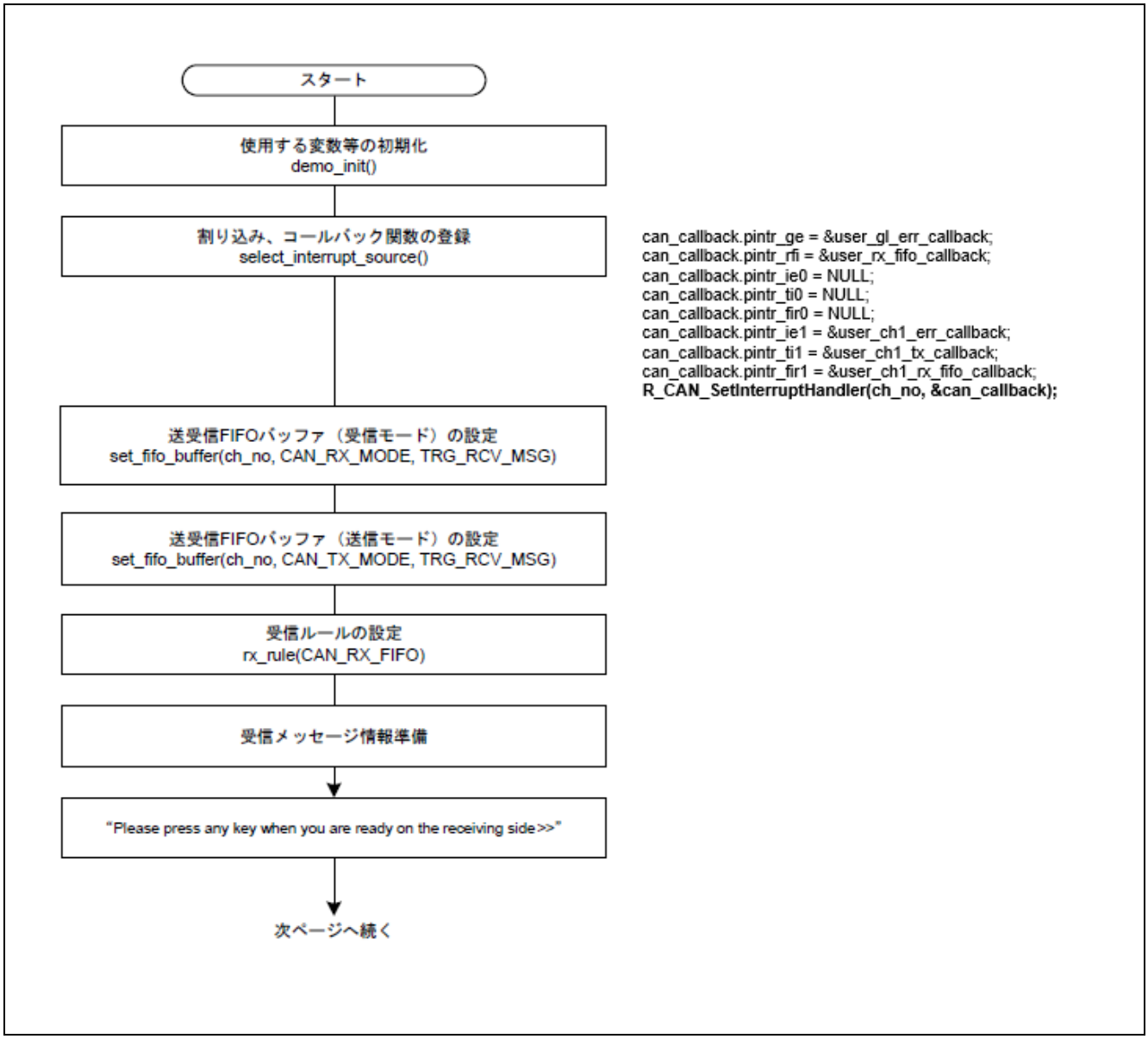


図 4-12 サンプルコードのメッセージを受信しながらメッセージを送信するテスト（1/2）

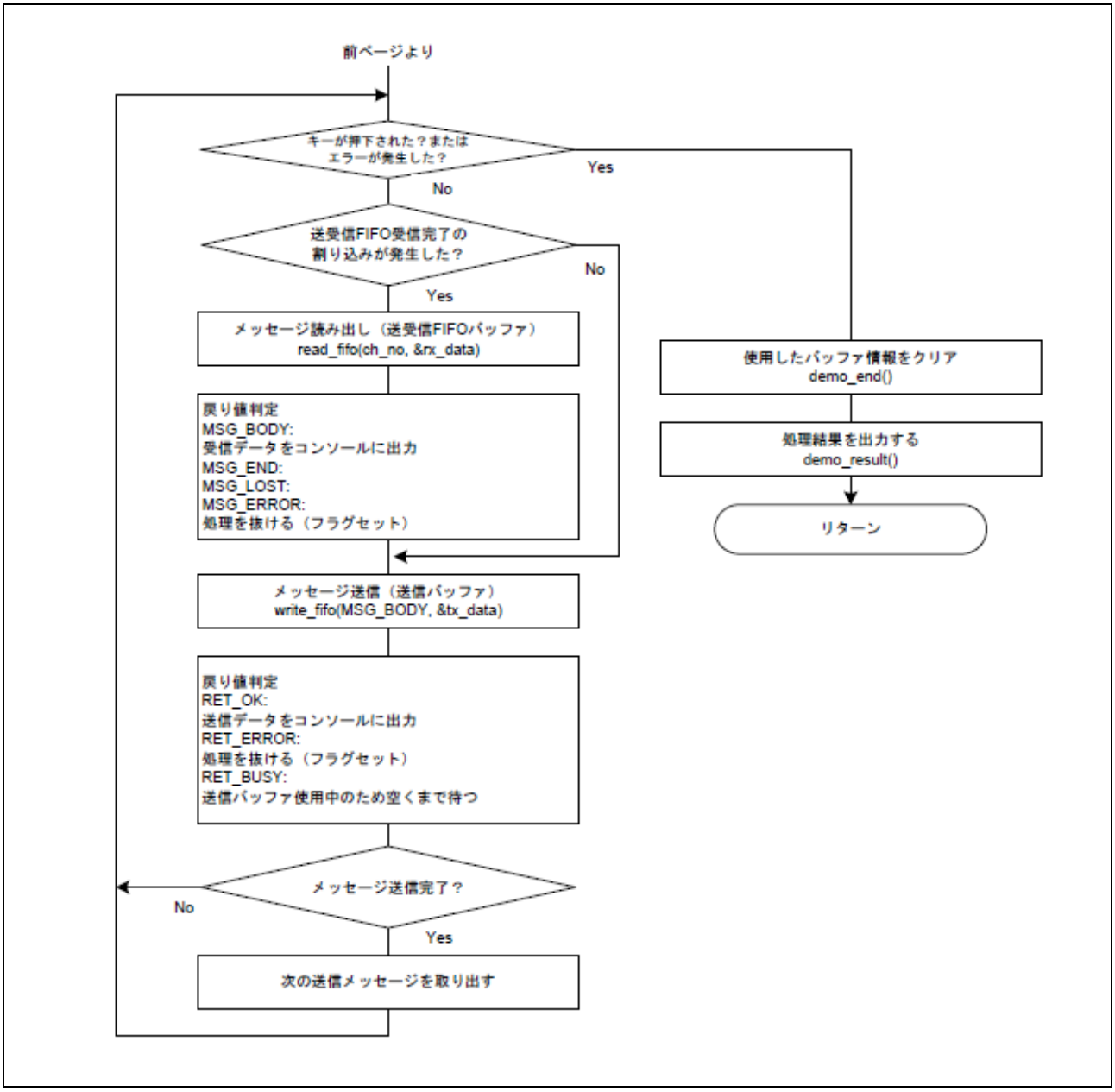


図 4-13 サンプルコードのメッセージを受信しながらメッセージを送信するテスト（2/2）

4.5.5　セルフテスト

開発環境に接続している評価ボードのみを使用して CAN 通信テストを行うことができます。

本サンプルプログラムでは、メッセージの送信・受信を行うセルフテストモードを用意しています。

また、セルフテストモード 0（外部ループバックモード）とセルフテストモード 1（内部ループバックモード）の切り替えが可能です。メニューから選択して下さい。

メニューの内容は、4.6 チュートリアル 4.6.5 サンプルプログラムの機能を参照して下さい。

- セルフテストモード 0（外部ループバックモード）

  CAN トランシーバを含めたチャネルのループバックテストを行うモードです。

  図 4-14 にセルフテストモード 0 選択時の接続を示します.

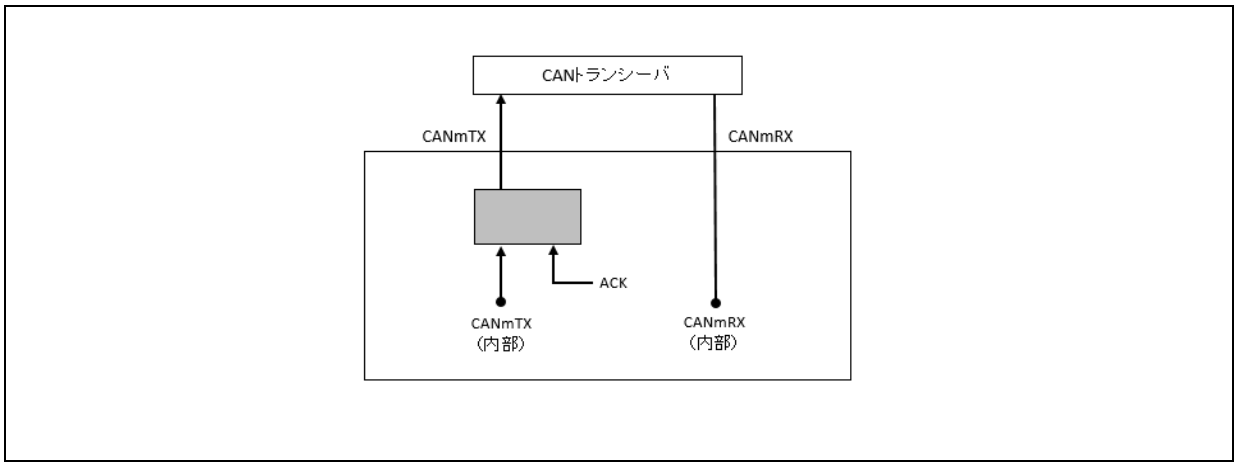


図 4-14 セルフテストモード 0 選択時の接続

- セルフテストモード 1（内部ループバックモード）

  送信したメッセージを受信したメッセージとして取り扱い、送信したメッセージをバッファに格納するモードです。

  図 4-15 にセルフテストモード 1 選択時の接続を示します。

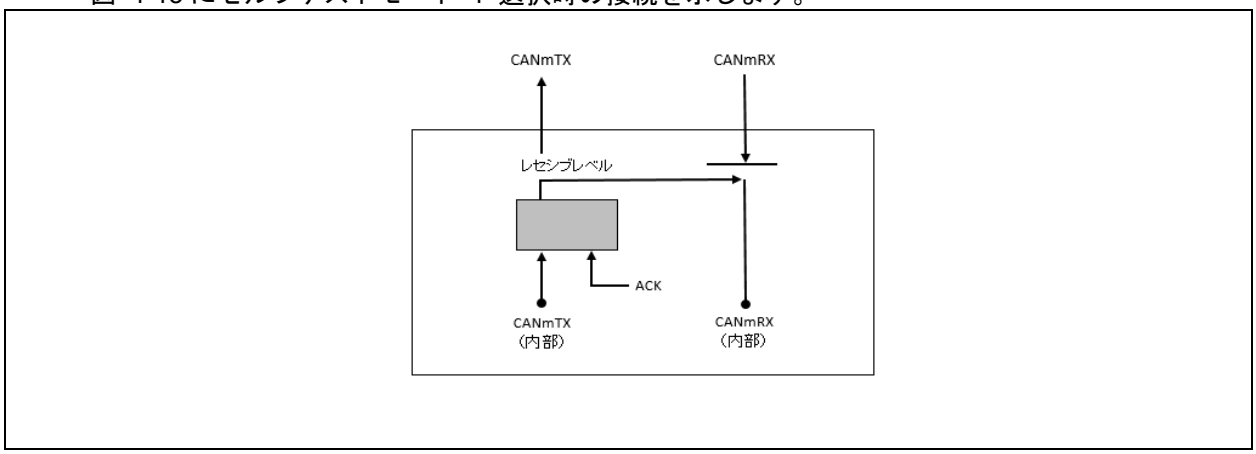


図 4-15 セルフテストモード 1 選択時の接続

セルフテストとして、下記を用意しております。

- 送信バッファを使ってメッセージを送信し、受信バッファでメッセージを受信するテスト
- 送信バッファを使ってメッセージを送信し、受信FIFO バッファでメッセージを受信するテスト
- 送信バッファを使ってメッセージを送信し、送受信FIFO バッファ(受信モード)でメッセージを受信するテスト
- 送受信FIFO バッファ(送信モード)を使ってメッセージを送信し、送受信FIFO バッファ(受信モード)でメッセージを受信するテスト

送信側の送信方法は

□ 送信バッファを使用時

1 メッセージ送信を繰り返します。

□ 送受信FIFO バッファ(送信モード)を使用時

1 メッセージ送信を繰り返します。

受信側の受信方法は

□ 受信FIFO バッファで受け取る場合

受信FIFO バッファフルで割込み発生（受信FIFO バッファ段数：4 メッセージ）

□ 送受信FIFO バッファ(受信モード)で受け取る場合

送受信FIFO バッファフルで割込み発生（送受信FIFO バッファ段数：4 メッセージ）

各テストを行うには以下の関数を使用します。

- void selftest_buf_to_buf(void)
  送信バッファを使ってメッセージを送信し、受信バッファでメッセージを受信するテスト
- void selftest_buf_to_rx_fifo(void)
  送信バッファを使ってメッセージを送信し、受信FIFO バッファでメッセージを受信するテスト
- void selftest_buf_to_fifo(void)
  送信バッファを使ってメッセージを送信し、送受信FIFO バッファ(受信モード)でメッセージを受信するテスト
- void selftest_fifo_to_fifo(void)
  送受信FIFO バッファ(送信モード)を使ってメッセージを送信し、送受信FIFO バッファ(受信モード) でメッセージを受信するテスト

図 4-16 ～ 図 4-22 にそれぞれの機能のフローチャートを示します。

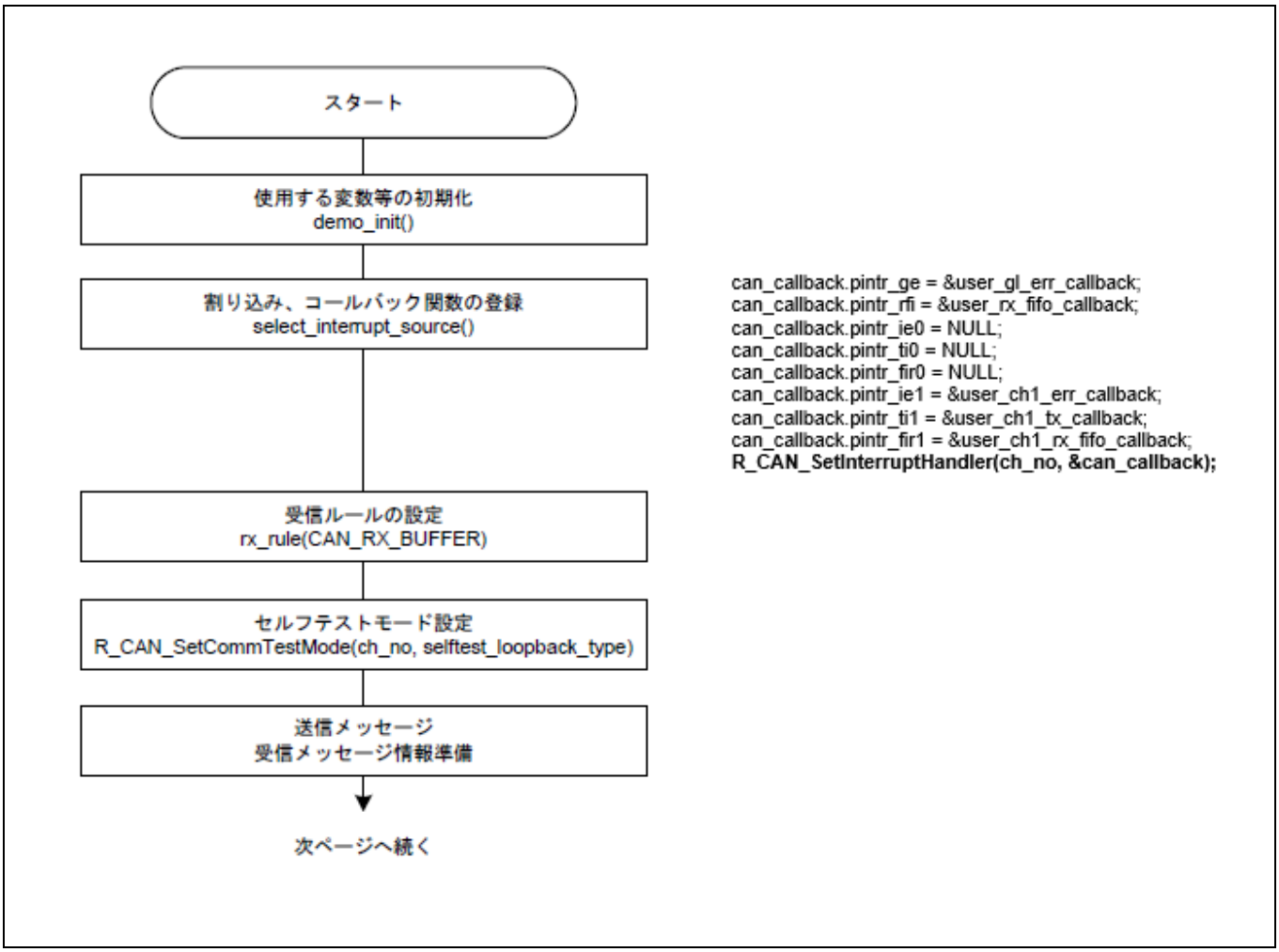


図 4-16 サンプルコードの送信バッファを使ってメッセージを送信し、受信バッファでメッセージを受信するテスト（1/2）

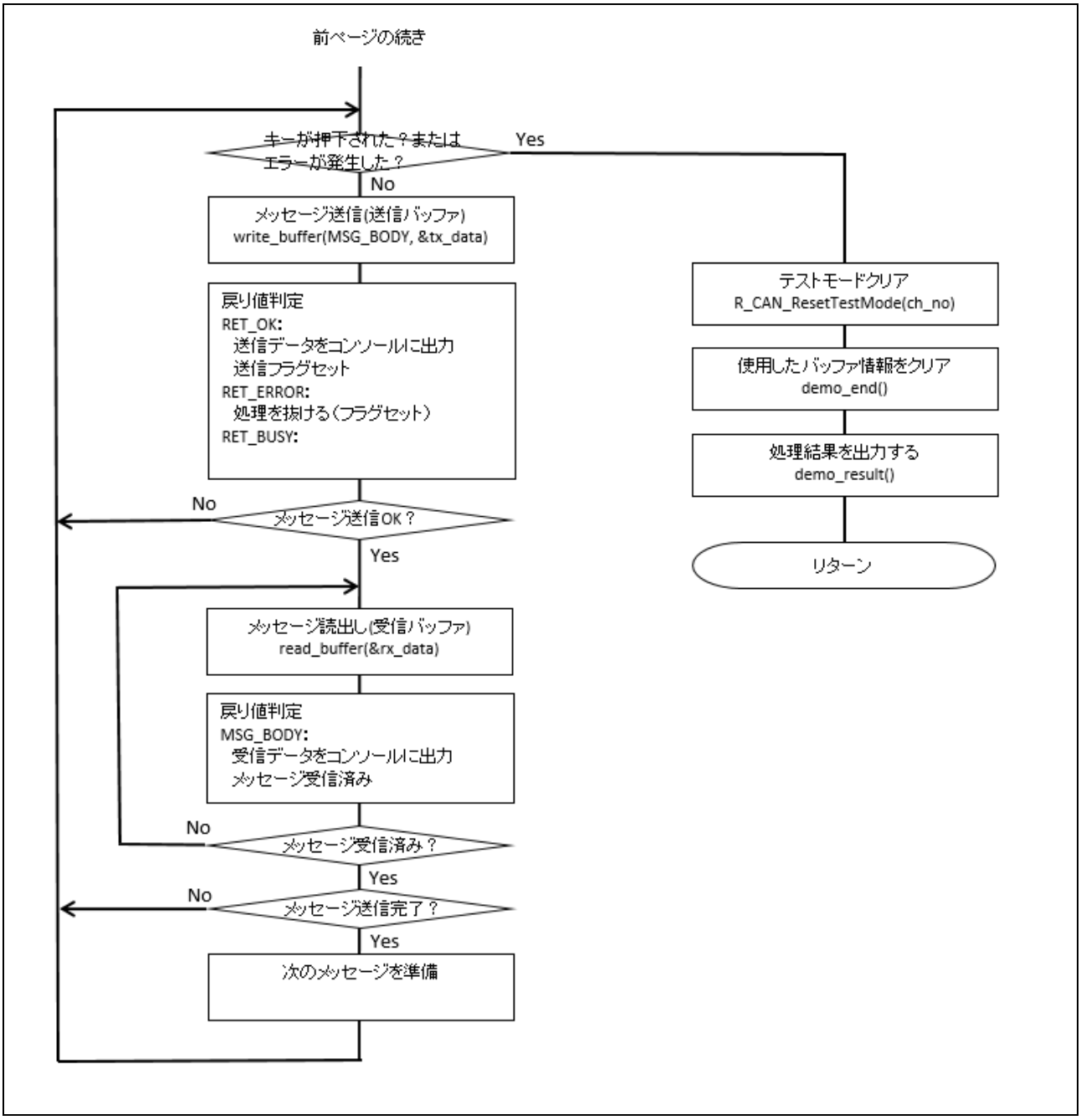


図 4-17 サンプルコードの送信バッファを使ってメッセージを送信し、受信バッファでメッセージを受信するテスト（2/2）

- 送信バッファを使ってメッセージを送信し、受信 FIFO バッファでメッセージを受信するテスト

  関数名：void selftest_buf_to_rx_fifo(void)

  以下に、送信バッファを使ってメッセージを送信し、受信FIFO バッファでメッセージを受信するテストのフローチャートを示します.

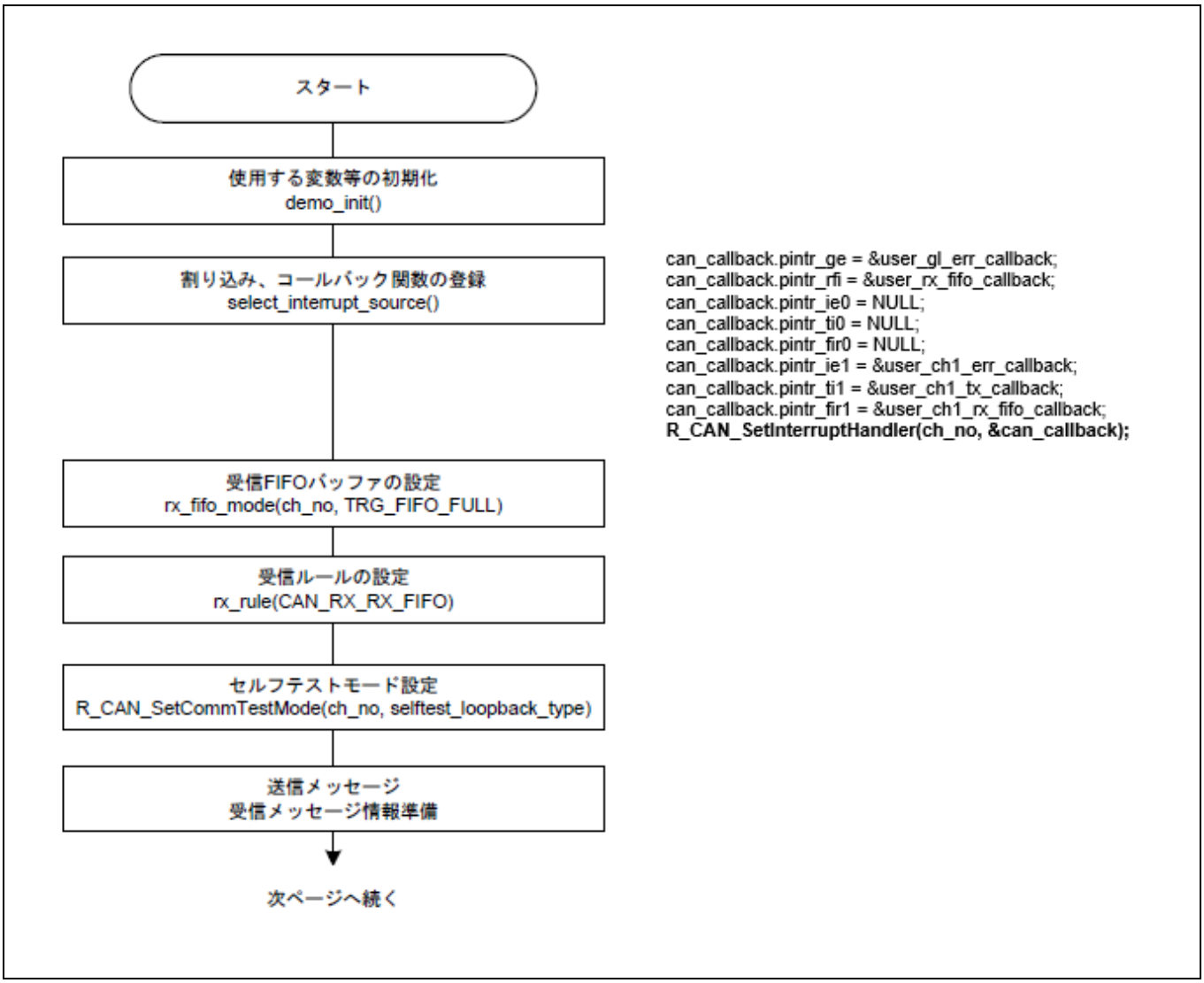


図 4-18 サンプルコードの送信バッファを使ってメッセージを送信し、受信 FIFO バッファでメッセージを受信するテスト（1/2）

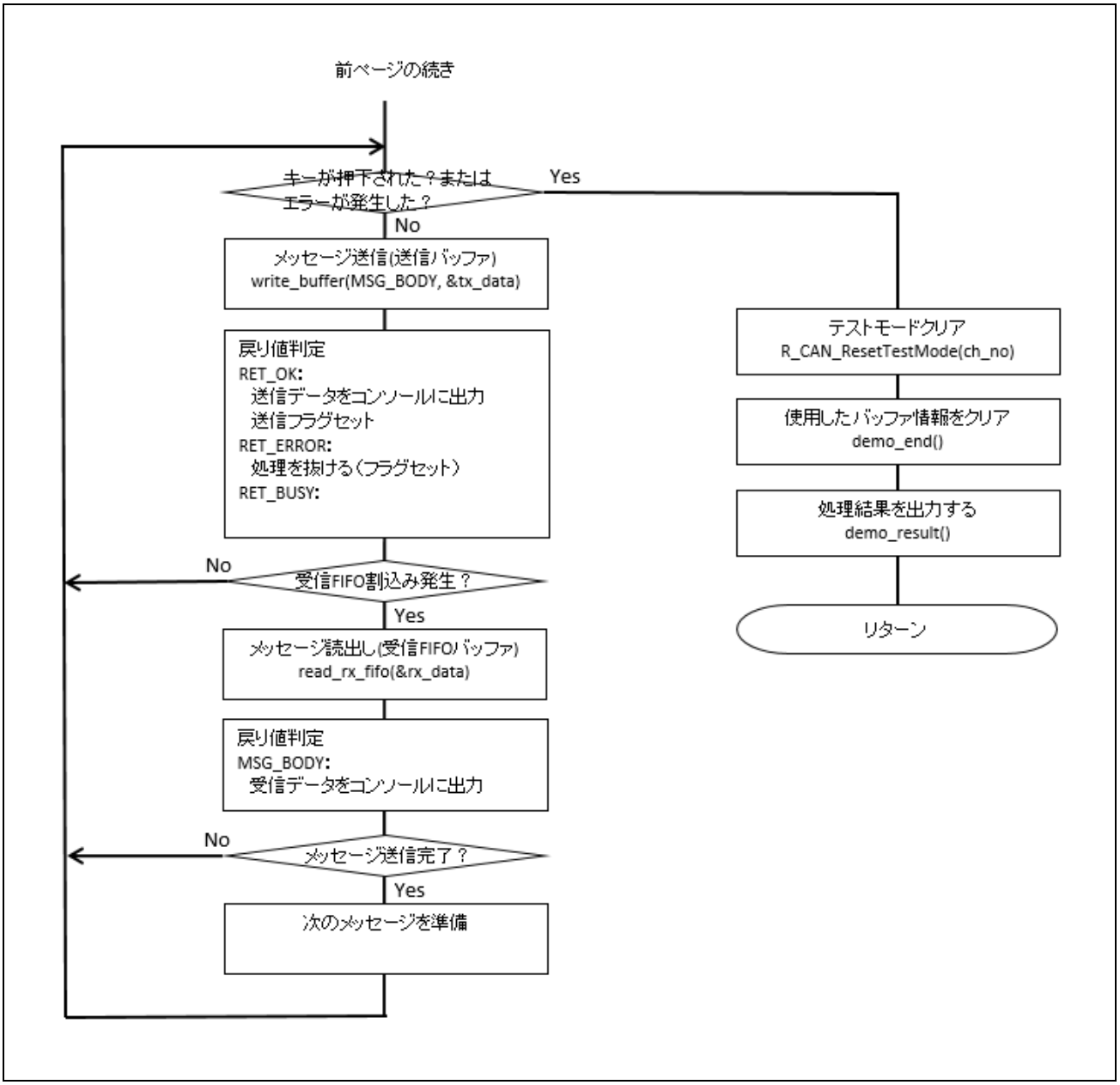


図 4-19 サンプルコードの送信バッファを使ってメッセージを送信し、受信 FIFO バッファでメッセージを受信するテスト（2/2）

- 送信バッファを使ってメッセージを送信し、送受信 FIFO バッファ（受信モード）でメッセージを受信するテスト

  関数名：void selftest_buf_to_fifo(void)

  以下に、送信バッファを使ってメッセージを送信し、送受信FIFO バッファ(受信モード)でメッセージを受信するテストのフローチャートを示します

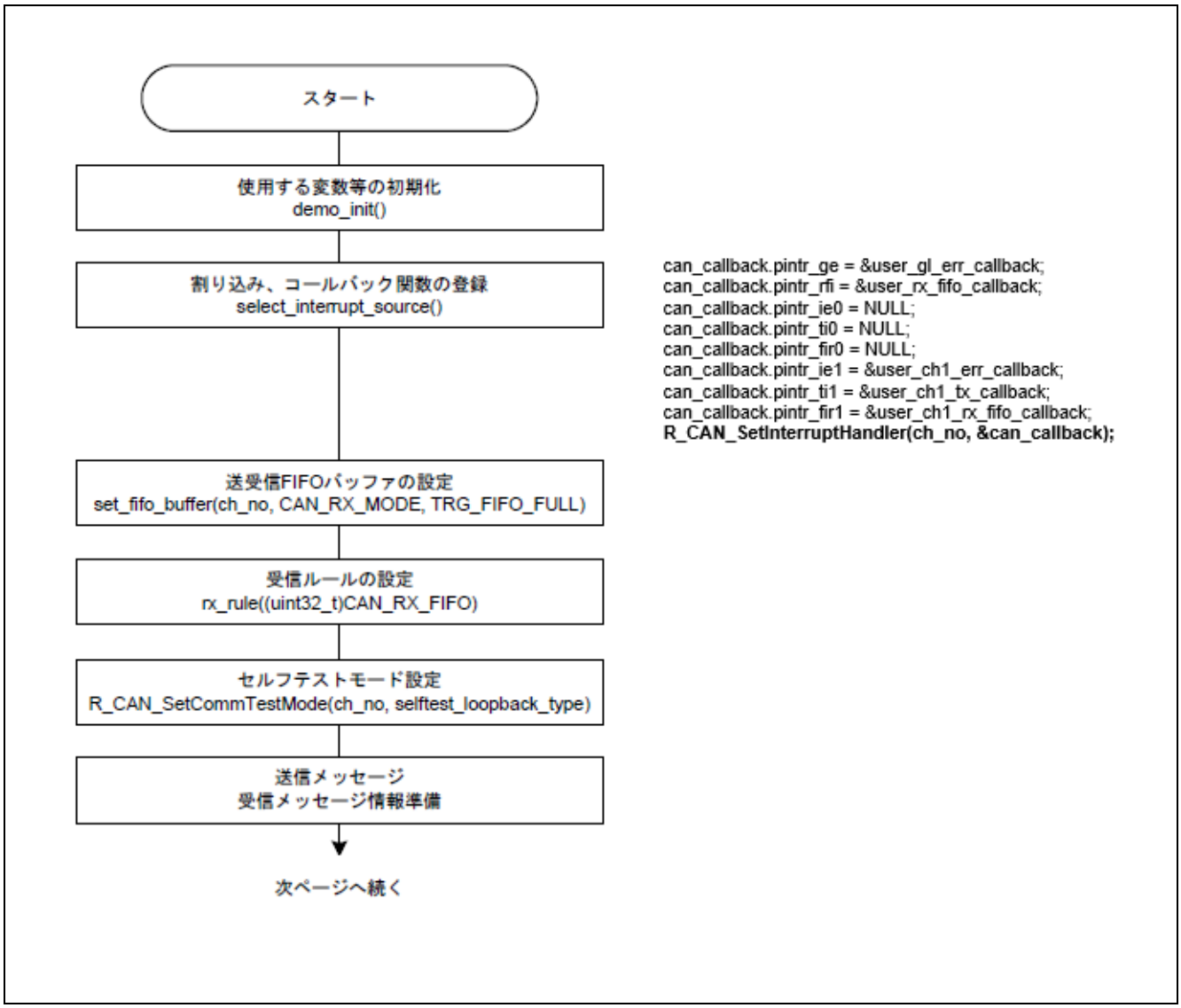


図 4-20 サンプルコードの送信バッファを使ってメッセージを送信し、送受信 FIFO バッファ(受信モード)でメッセージを受信するテスト（1/2）

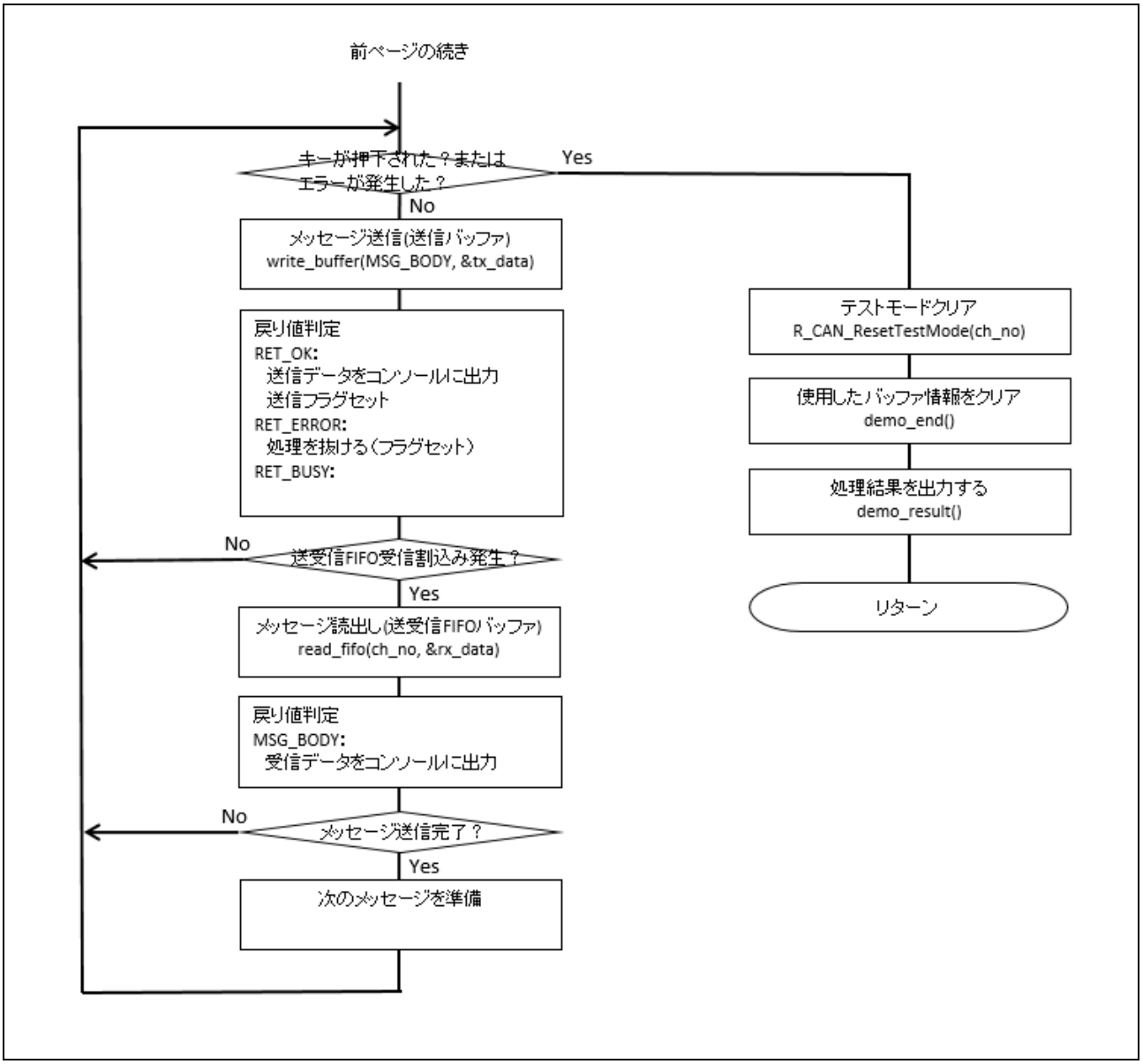


図 4-21 サンプルコードの送信バッファを使ってメッセージを送信し、送受信 FIFO バッファ(受信モード)でメッセージを受信するテスト（2/2）

- 送受信 FIFO バッファ（送信モード）を使ってメッセージを送信し、送受信 FIFO バッファ（受信モード）でメッセージを受信するテスト

  関数名：void selftest_fifo_to_fifo(void)

  以下に、送受信FIFO バッファ(送信モード)を使ってメッセージを送信し、送受信FIFO バッファ(受信モード)でメッセージを受信するテストのフローチャートを示します。

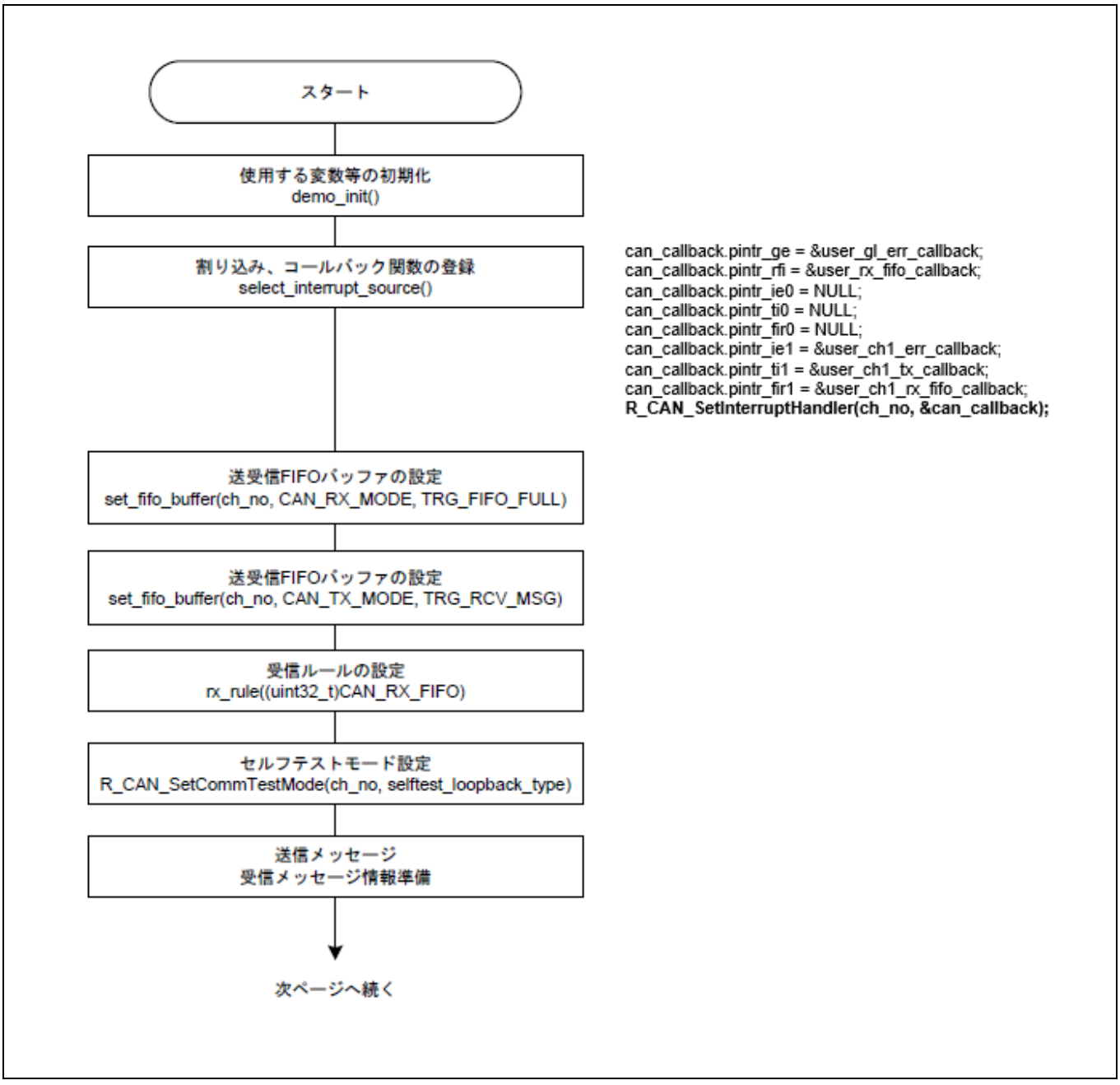


図 4-22 サンプルコードの送受信 FIFO バッファ(送信モード)を使ってメッセージを送信し、送受信 FIFO バッファ(受信モード)でメッセージを受信するテスト（1/2）

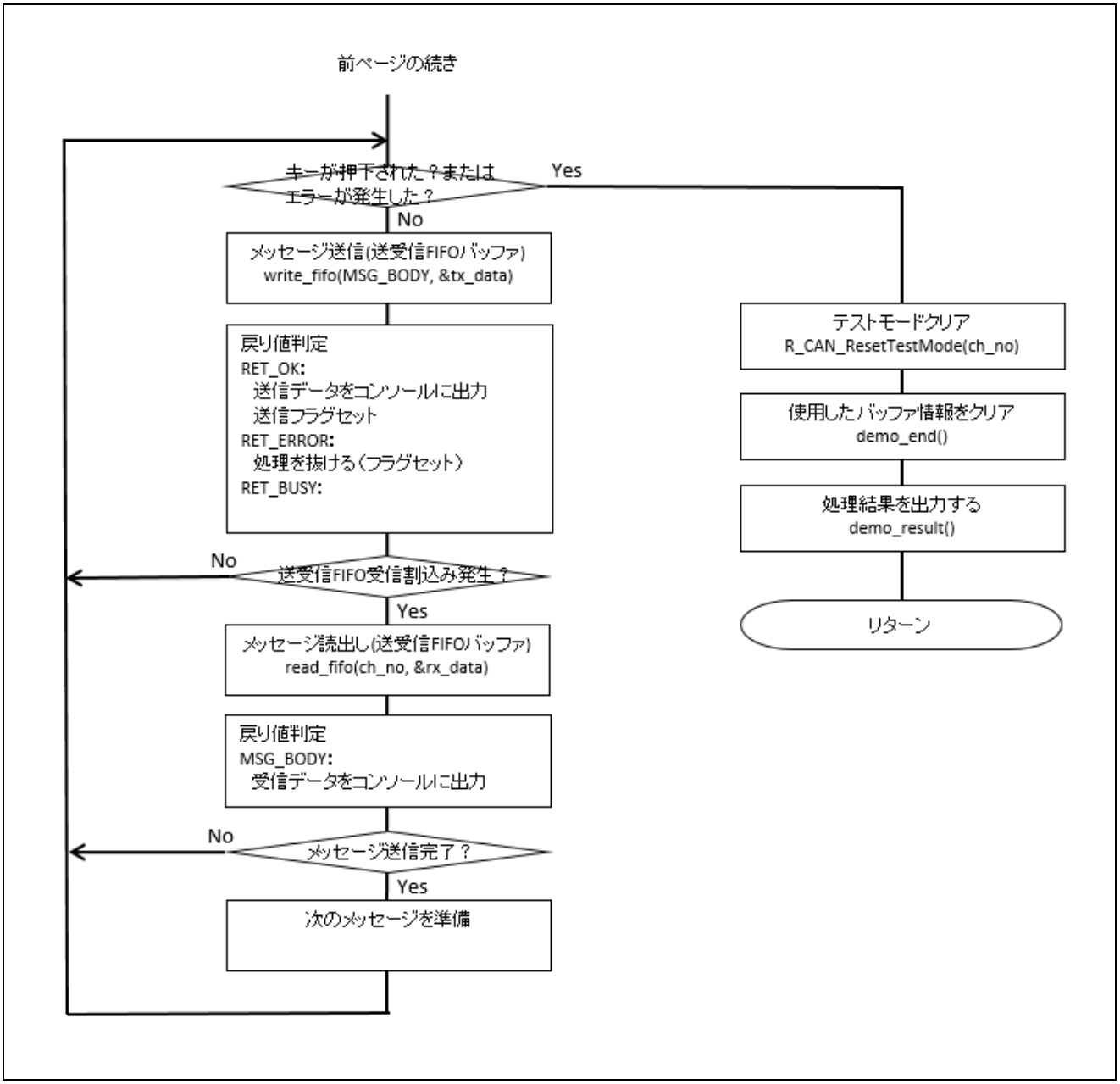


図 4-23 サンプルコードの送受信 FIFO バッファ(送信モード)を使ってメッセージを送信し、送受信FIFOバッファ(受信モード)でメッセージを受信するテスト（2/2）

4.5.6　コールバック処理

本サンプルプログラムは、割込み発生時に呼ばれる割込みハンドラと、割込みハンドラから呼ばれるコールバック関数を実装しています。

- RSCAN:CANGE　CAN グローバル・エラー

  割込みハンドラ：void user_gl_err_isr(void)

  コールバック関数：void user_gl_err_callback(void)

- RSCAN:CANIE1　CAN1 エラー

  割込みハンドラ：void user_ch1_err_isr(void)

  コールバック関数：void user_ch1_err_callback(void)

- RSCAN:CANRFI　CAN 受信FIFO 割込み

  割込みハンドラ：void user_rx_fifo_isr(void)

  コールバック関数：void user_rx_fifo_callback(void)

- RSCAN:CANTI1　CAN1 送信割込み

  割込みハンドラ：void user_ch1_tx_isr(void)

  コールバック関数：void user_ch1_tx_callback(void)

- RSCAN:CANFIR1　CAN1 送受信FIFO 受信完了割込み

  割込みハンドラ：void user_ch1_rx_fifo_isr(void)

  コールバック関数：void user_ch1_rx_fifo_callback(void)

- SCIFA:RXIF2　SCIFA 受信 FIFO データフル割込み

  割込みハンドラ：void scifa_key_input_isr(void)

  コールバック関数：void key_handler_callback(void)

- 割り込みハンドラの登録方法

  割り込みハンドラは、以下のAPI 関数を用いて行って下さい。

  void R_ICU_Regist(uint32_t vec_num, uint32_t type, uint32_t priority, uint32_t isr_addr);

  uint32_t vec_num：ベクタ番号

  uint32_t type：割込み検出タイプ

  uint32_t priority：割込み優先レベル

  uint32_t isr_addr：割込みハンドラ関数のアドレス

  また、割込みの許可／禁止は、以下のAPI 関数を用いて下さい。

  void R_ICU_Disable(uint32_t vec_num);

  void R_ICU_Enable(uint32_t vec_num);

  uint32_t vec_num：ベクタ番号

- コールバック関数の登録方法

  can_handle_t gb_can_handles[CAN_NUM];

```
typedef struct {
    void (*pintr_ge)(void);      /* pointer to user callback function. */
    void (*pintr_ie0)(void);     /* pointer to user callback function. */
    void (*pintr_ie1)(void);     /* pointer to user callback function. */
    void (*pintr_rfi)(void);     /* pointer to user callback function. */
    void (*pintr_fir0)(void);    /* pointer to user callback function. */
    void (*pintr_ti0)(void);     /* pointer to user callback function. */
    void (*pintr_fir1)(void);    /* pointer to user callback function. */
    void (*pintr_ti1)(void);     /* pointer to user callback function. */
} can_callback_t;

  typedef struct {
      bool
      can_callback_t
  } can_handle_t;
```

```
ch_opened;
can_callback;
```

図 4-24 ～ 図 4-29 にそれぞれの機能のフローチャートを示します。

- （RSCAN:CANGE）CAN グローバル・エラー発生時に呼ばれるコールバック関数

  割込みハンドラ：void user_gl_err_isr(void)

  コールバック関数：void user_gl_err_callback(void)

  CAN モジュール全体で、エラーが発生した場合に割込みハンドラ処理から呼び出されます。

  以下に、グローバル・エラー発生時に呼ばれるハンドラ処理とその中から呼ばれるコールバック処理のフローチャートを示します。

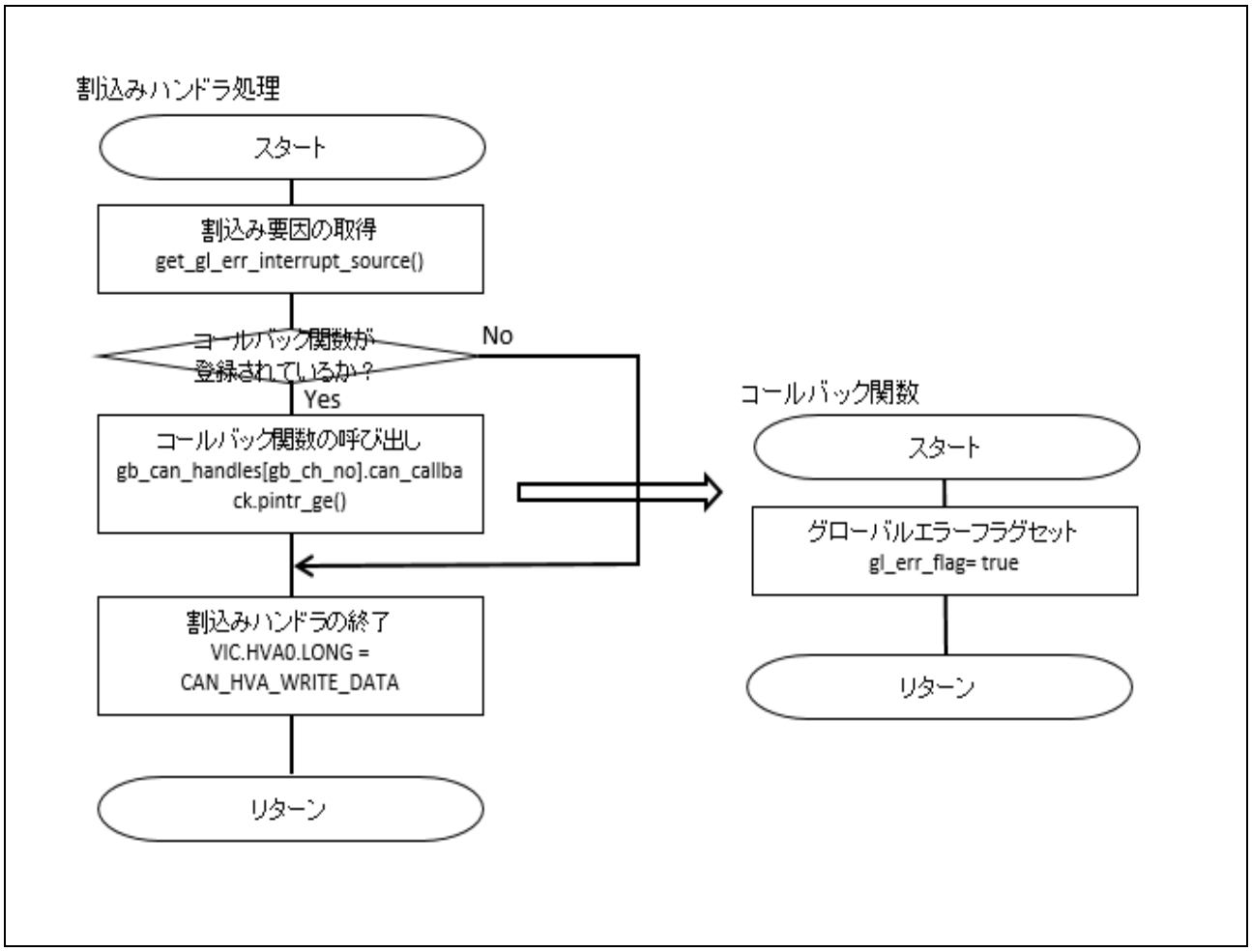


図 4-24 サンプルコードの CAN グローバル・エラー発生時に呼ばれるコールバック関数処理

- （RSCAN:CANIE1）CAN1 エラー発生時に呼ばれるコールバック関数

  割込みハンドラ： void user_ch1_err_isr(void)

  コールバック関数：void user_ch1_err_callback(void)

  CAN モジュールのチャネル（CAN1）でエラーが発生した場合に割込みハンドラ処理から呼び出されます。

  以下に、チャネル・エラー発生時に呼ばれるハンドラ処理とその中から呼ばれるコールバック処理のフローチャートを示します。

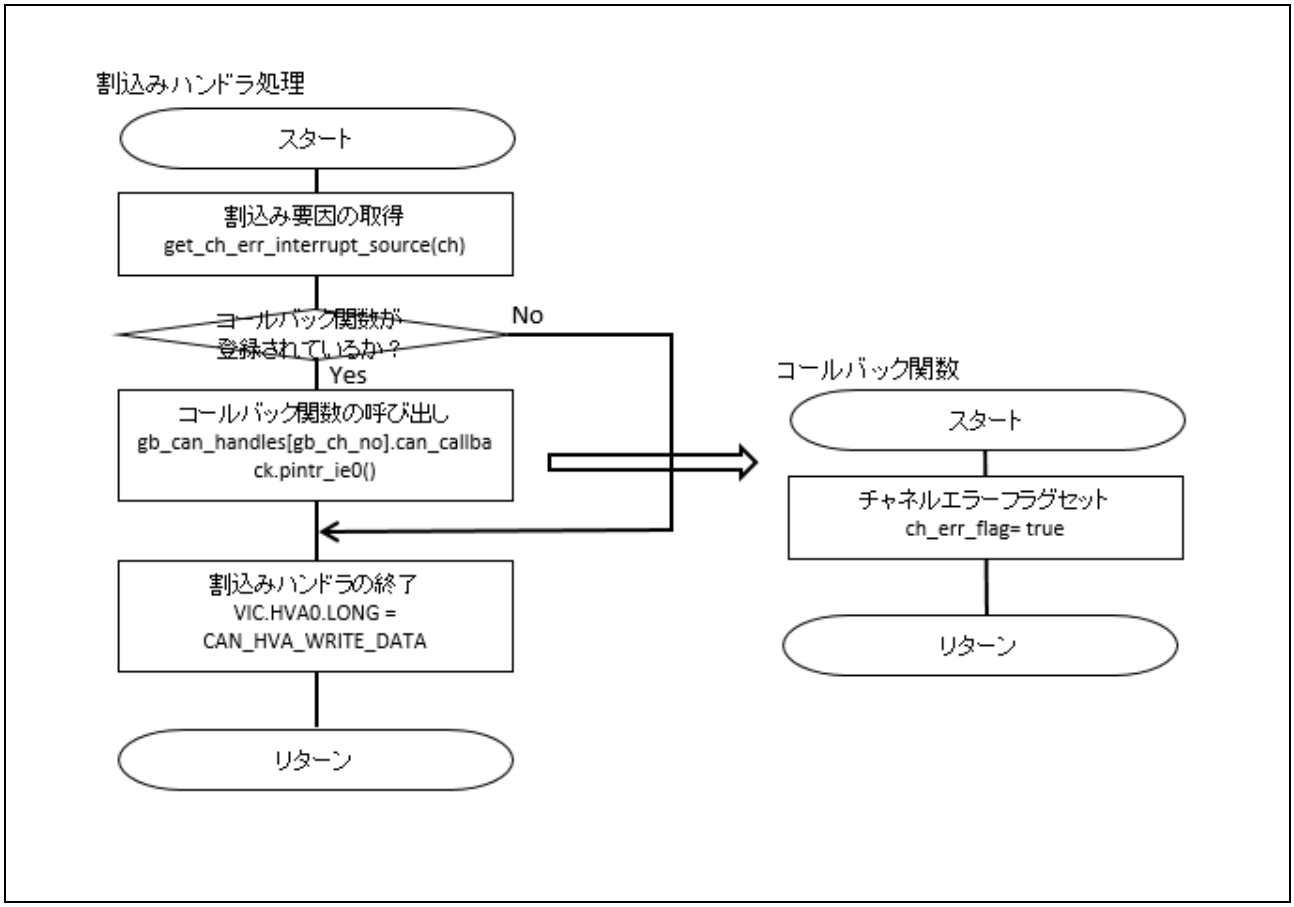


図 4-25 サンプルコードの CAN1 エラー発生時に呼ばれるコールバック関数処理

- （RSCAN:CANRFI）CAN 受信 FIFO 割込み発生時に呼ばれるコールバック関数

  割込みハンドラ：void user_rx_fifo_isr(void)

  コールバック関数：void user_rx_fifo_callback(void)

  受信 FIFO バッファを使って受信する設定にしておくと、受信 FIFO バッファにメッセージが溜まると割込みハンドラ処理から呼び出されます。

  以下に、受信 FIFO 割込み発生時に呼ばれるハンドラ処理とその中から呼ばれるコールバック処理のフローチャートを示します。

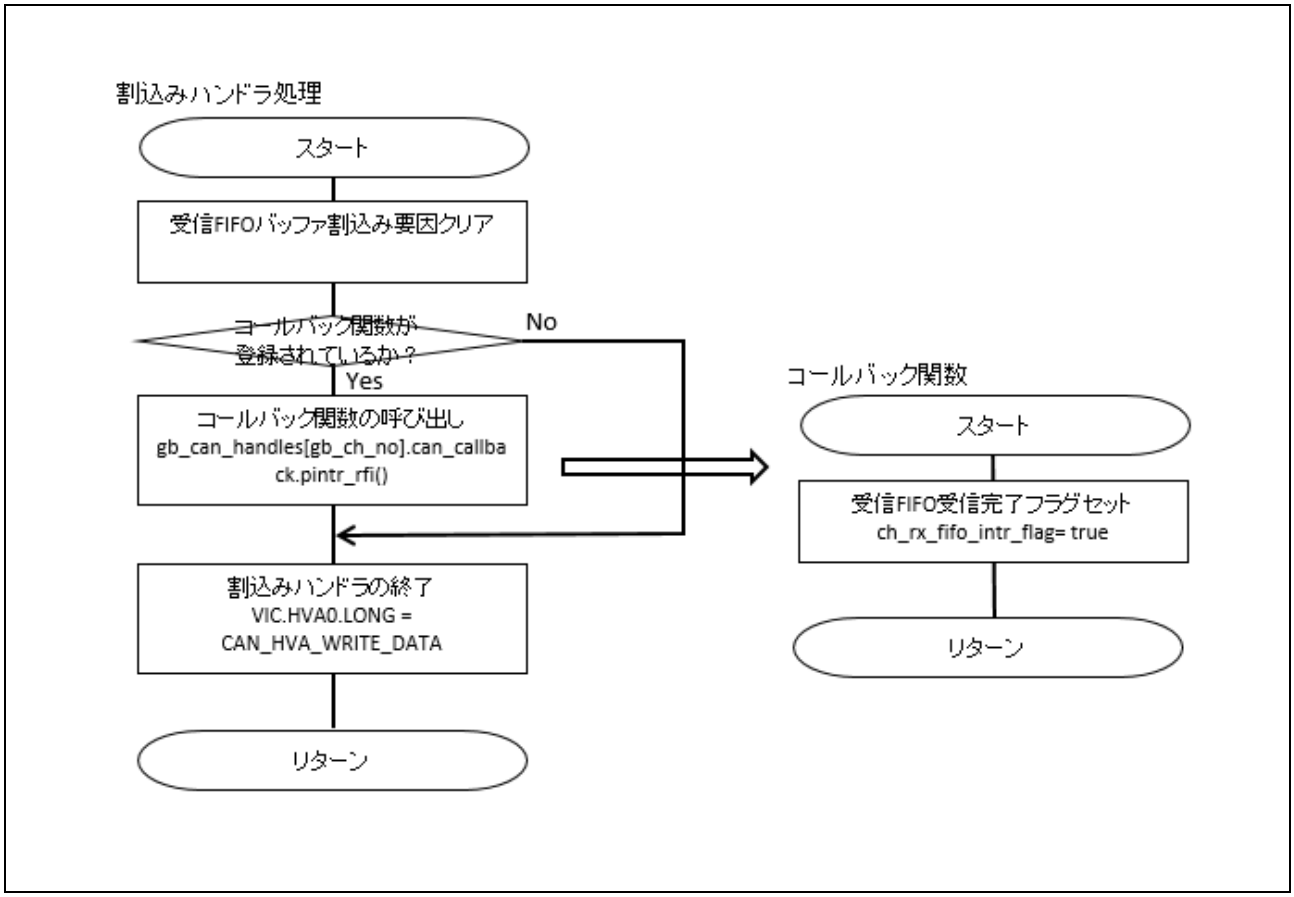


図 4-26 サンプルコードの CAN 受信 FIFO 割込み発生時に呼ばれるコールバック関数処理

- （RSCAN:CANTI1）CAN1 送信割込み発生時に呼ばれるコールバック関数

  割り込みハンドラ：void user_ch1_tx_isr(void)

  コールバック関数：void user_ch1_tx_callback(void)

  メッセージの送信が完了すると割込みハンドラ処理から呼び出されます。

  以下に、送信完了割込み発生時に呼ばれるハンドラ処理とその中から呼ばれるコールバック処理のフローチャートを示します。

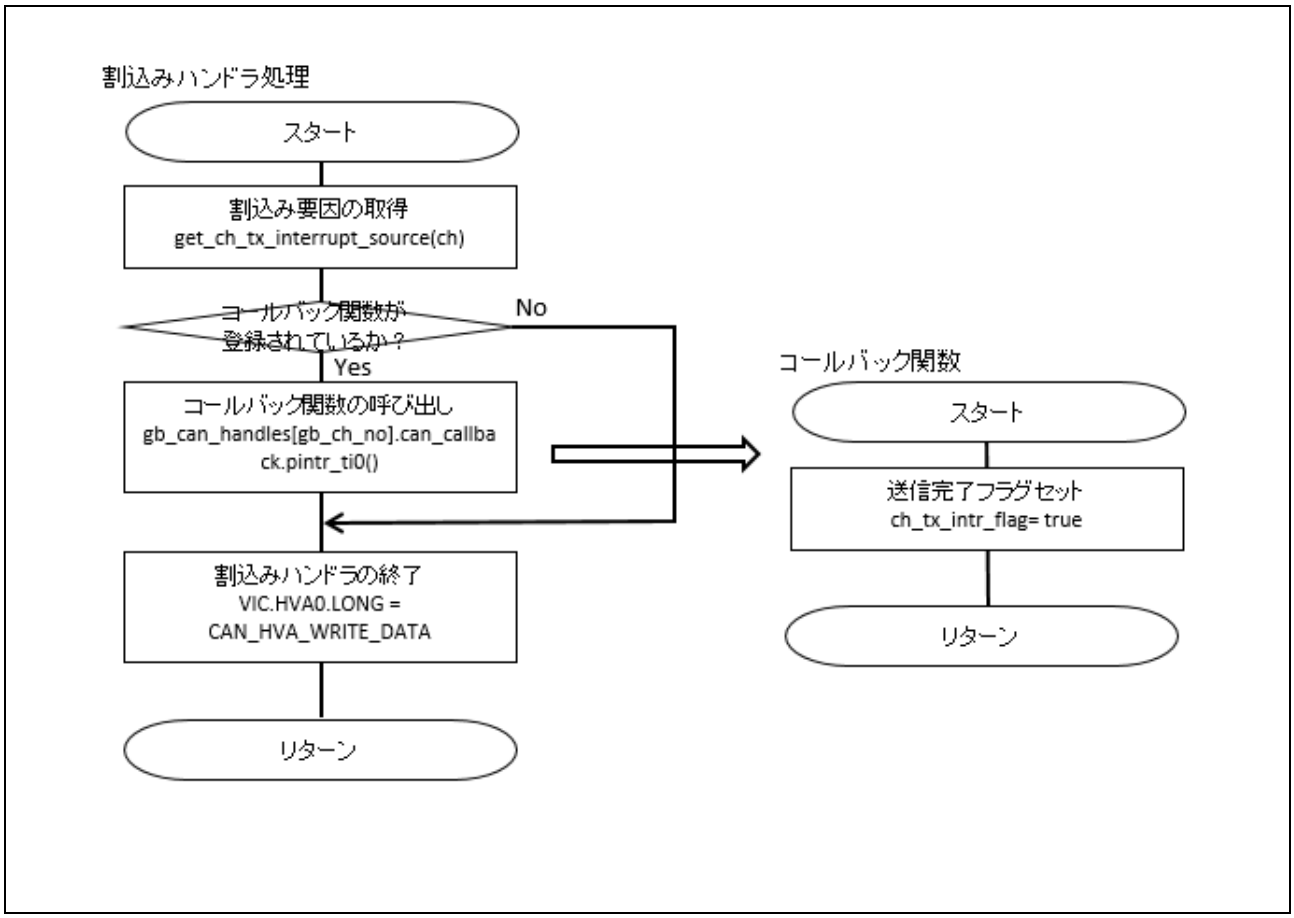


図 4-27 サンプルコードの CAN1 送信割込み発生時に呼ばれるコールバック関数処理

- （RSCAN:CANFIR1）CAN1 送受信 FIFO 受信完了割込み発生時に呼ばれるコールバック関数

  割り込みハンドラ：void user_ch1_rx_fifo_isr(void)

  コールバック関数：void user_ch1_rx_fifo_callback(void)

  送受信 FIFO バッファを使って受信する設定にしておくと、送受信 FIFO バッファにメッセージが溜まると割込みハンドラ処理から呼び出されます。

  以下に、送受信 FIFO 受信完了割込み発生時に呼ばれるハンドラ処理とその中から呼ばれるコールバック処理のフローチャートを示します。

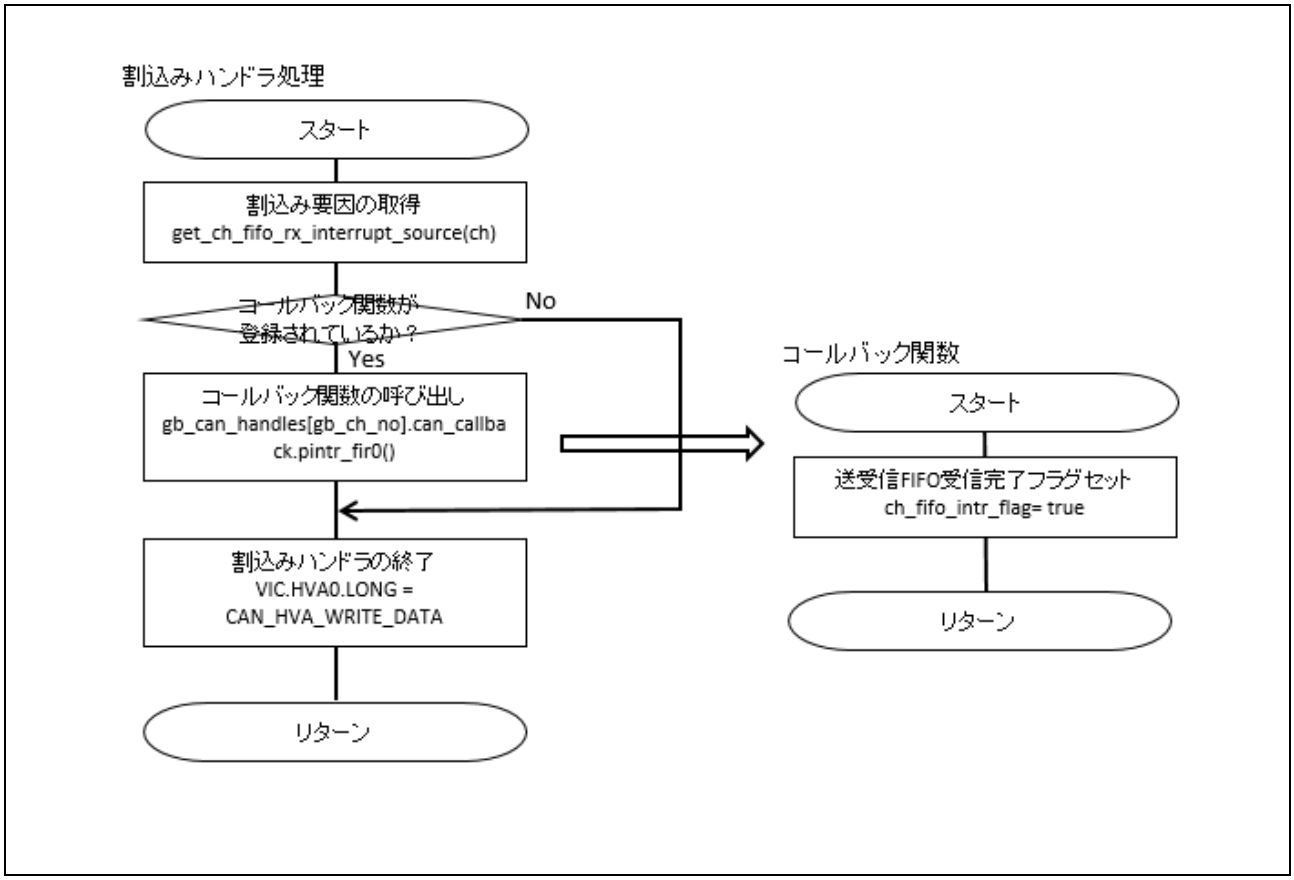


図 4-28 サンプルコードの CAN1 送受信 FIFO 受信完了割込み発生時に呼ばれるコールバック関数処理

- （SCIFA:RXIF2）受信 FIFO データフル割込み発生時に呼ばれるコールバック関数

  割り込みハンドラ：void scifa_key_input_isr(void)

  コールバック関数：void key_handler_callback(void)

  SCIFA のキー入力の割込み発生時にハンドラから呼ばれます。

  本サンプルプログラムは、送信時、メッセージを繰り返し受信側に送信つづけます。

  途中でメッセージの送信を中止させるために以下のコールバック関数を用意しています。

  送信中に何かキーが押下されるとキー押下フラグがセットされます。

  以下に、キー入力の割込み発生時に呼ばれるハンドラ処理とその中から呼ばれるコールバック処理のフローチャートを示します。

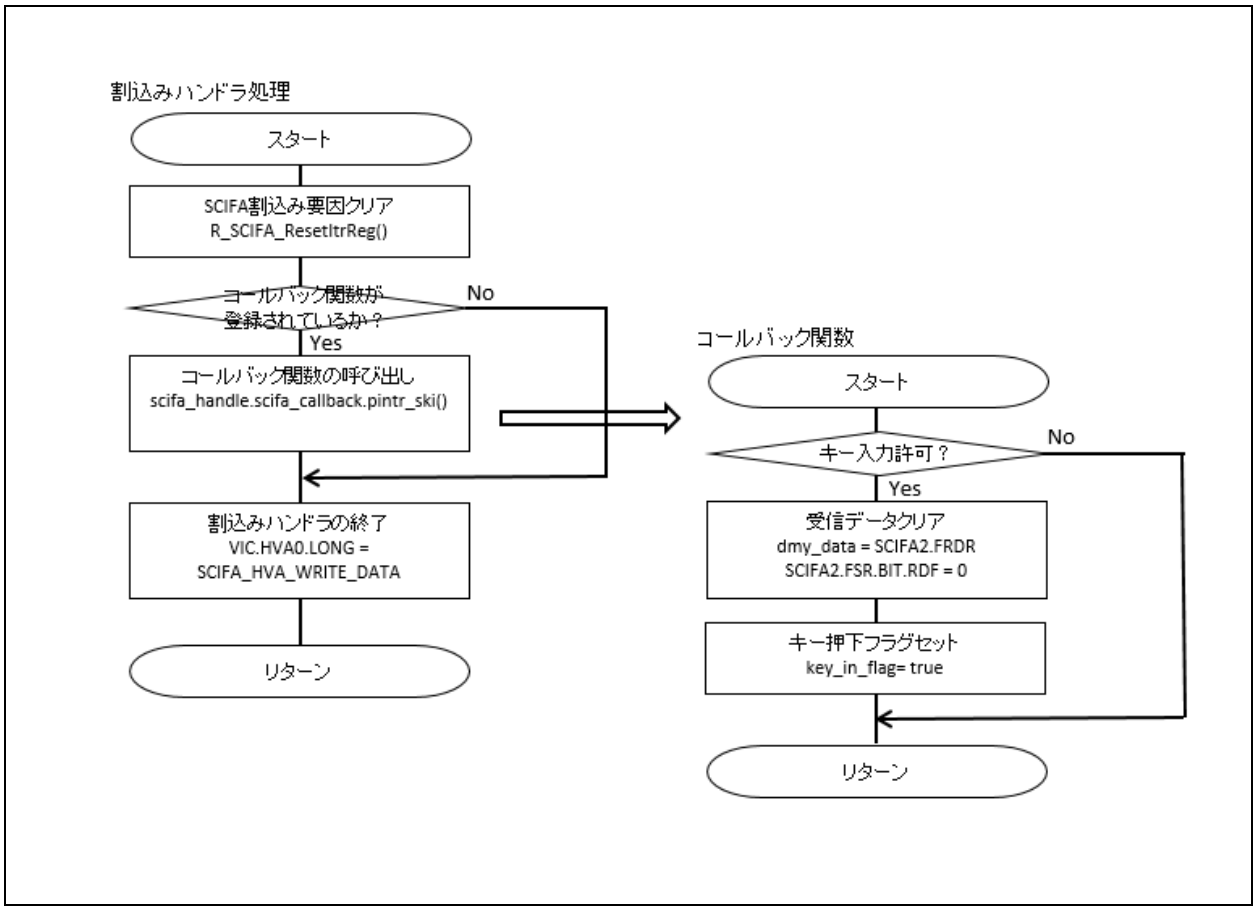


図 4-29 サンプルコードの SCIFA の受信 FIFO データフル割込み発生時に呼ばれるコールバック関数処理

4.6　チュートリアル

4.6.1　動作概要

RSCAN サンプルプログラムの機能概要を表 4-4 動作概要に示します。また、図 4-30 にシステムブロック図を示します。

表 4-4 動作概要

機能	概要
兼用端子設定	・PC7：CAN1 CRXD1　・P66：CAN1 CTXD1
CAN の通信チャネル	・チャネル1（CAN1）を使用
割り込み要因（割り込み優先度）	・CAN グローバル・エラー（5） ・CAN1 エラー（5） ・CAN 受信FIFO（5） ・CAN1 送受信FIFO 受信完了（5） ・CAN1 送信（5）
転送速度設定	・1M[bps]
動作モード	・送信モード（評価ボード2 台接続時の送信側） 送信バッファを使ったメッセージ送信 送受信FIFO（送信モード）バッファを使ったメッセージ送信 ・受信モード（評価ボード2 台接続時の受信側） 受信バッファを使ったメッセージ受信 受信FIFO バッファを使ったメッセージ受信 送受信FIFO（受信モード）バッファを使ったメッセージ受信 ・受信を受け付けながらの送信テスト ・テストモード（評価ボード1 台のみでのセルフテスト） セルフテストモード 0（外部ループバックモード） セルフテストモード1（内部ループバックモード）
動作概要	メニューより動作モードを選択
動作結果表示	コンソールに結果出力

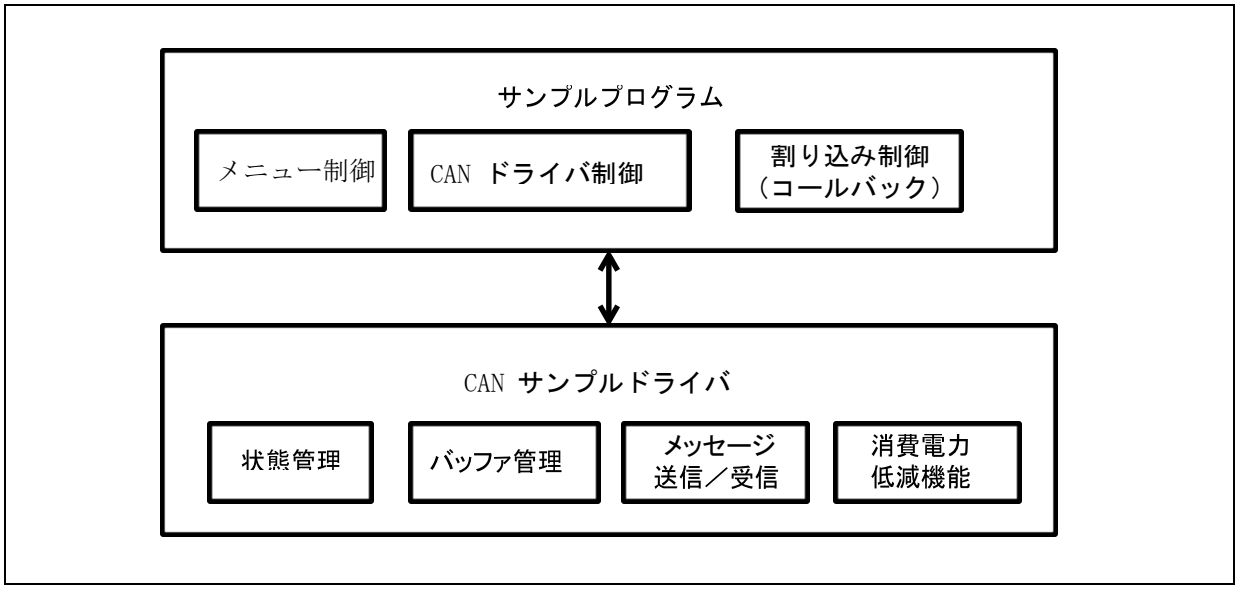


図 4-30 システムブロック図

4.6.2　使用準備（セルフテスト）

本サンプルプログラムを使用して、CAN 通信のセルフテストを行うことができます。 開発環境に接続している評価ボード単体で確認ができます。

4.6.3　使用準備（送信テスト・受信テスト）

開発環境に接続している評価ボード（評価ボード(A)）と、別の PC にセットアップされた開発環境に接続 している評価ボード（評価ボード(B)）の２台が必要です。
評価ボード(A)と評価ボード(B)を CAN ケーブル(*1)で接続します。（お互いの CAN1 コネクタに接続）

(*1)
　ボード(A)CAN コネクタ1(J4)のCAN-H 端子とボード(B)CAN コネクタ1(J4)のCAN-H 端子をケーブルで接続。
　ボード(A)CAN コネクタ1(J4)のCAN-L 端子とボード(B)CAN コネクタ1(J4)のCAN-L 端子をケーブルで接続。

4.6.4　ターミナルソフト（Tera Term）

本サンプルプログラムは、FIFO 内蔵シリアルコミュニケーションインターフェース（SCIFA）の調歩同期式通信を用い、ホストPC とRS-232C インターフェースのCOM ポート通信を行います。

ホストPC にてターミナルソフト（Tera Term）を起動してシリアルポートの設定（ボーレート：115200）を行って下さい。また、送受信時の改行コードは“CR”に設定します。

- 転送速度：　115200 bps
- キャラクタ長：　8　ビット
- ストップビット長：　1　ビット
- パリティ機能：　なし
- ハードウェアフロー制御：　なし

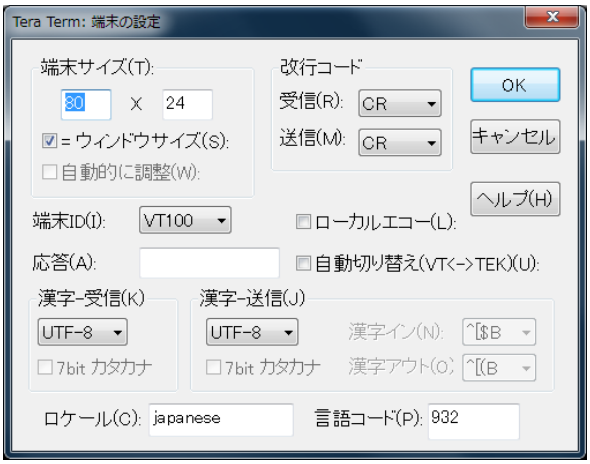


図 4-31 Tera Term：端末の設定

下記の例は、シリアルポートを“COM4”を使用した場合。

図 4-32 Tera Term：シリアルポートの設定

4.6.5　サンプルプログラムの機能

ターミナルソフト（Tera Term）を起動しておき、本サンプルプログラムを起動します。
コンソールのメニューよりサンプルプログラムの機能を選択して下さい。

- メインメニュー

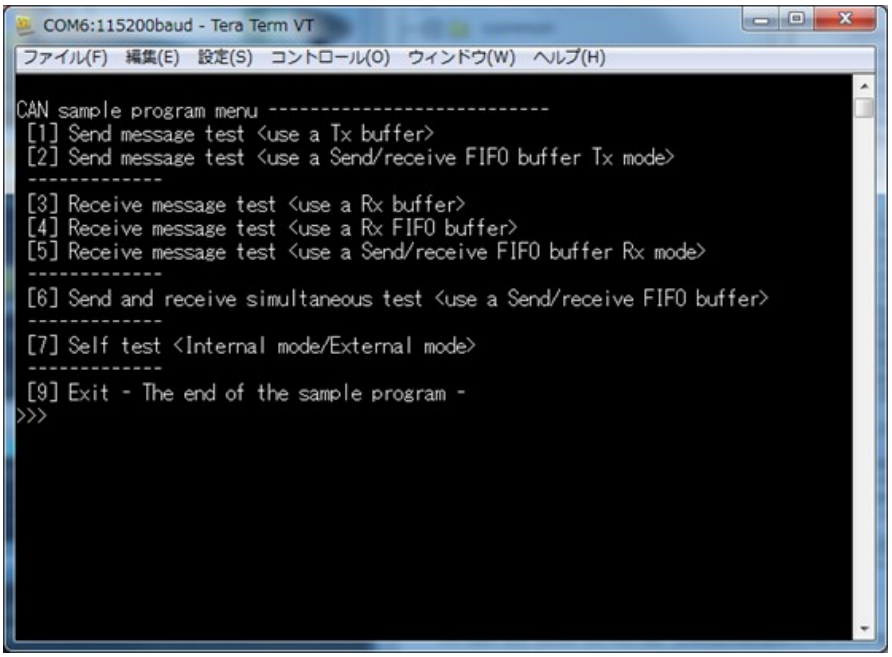


図 4-33 サンプルプログラムのメインメニュー

メニューの各項目の内容を以下に説明します。

- [1] Send message test <uses a Tx buffer>

  送信バッファからメッセージを送信するテスト。

- [2] Send message test <uses a Send/receive FIFO buffer Tx mode>

  送受信FIFO バッファ(送信モード)からメッセージを送信するテスト。

- [3] Receive message test <uses a Rx buffer>

  受信バッファでメッセージを受信するテスト。

- [4] Receive message test <uses a Rx FIFO buffer>

  受信FIFO バッファでメッセージを受信するテスト。

- [5] Receive message test <uses a Send/receive FIFO buffer Rx mode>

  送受信FIFO バッファ(受信モード)でメッセージを受信するテスト。

- [6] Send and receive simultaneous test < uses a Send/receive FIFO buffer >

  受信を受け付けながらの送信テスト。

- [7] Self test <Internal mode/External mode>

  セルフテスト選択メニュー。

- [9] Exit – The end of the sample program –

  サンプルプログラム終了。

- セルフテストメニュー

  本メニューを使って、評価ボード(A)単体でのセルフテスト（外部ループバック、内部ループバック）の動作確認を行うことができます。

  外部ループバックモード、内部ループバックモードはメニューから切り替え可能です。

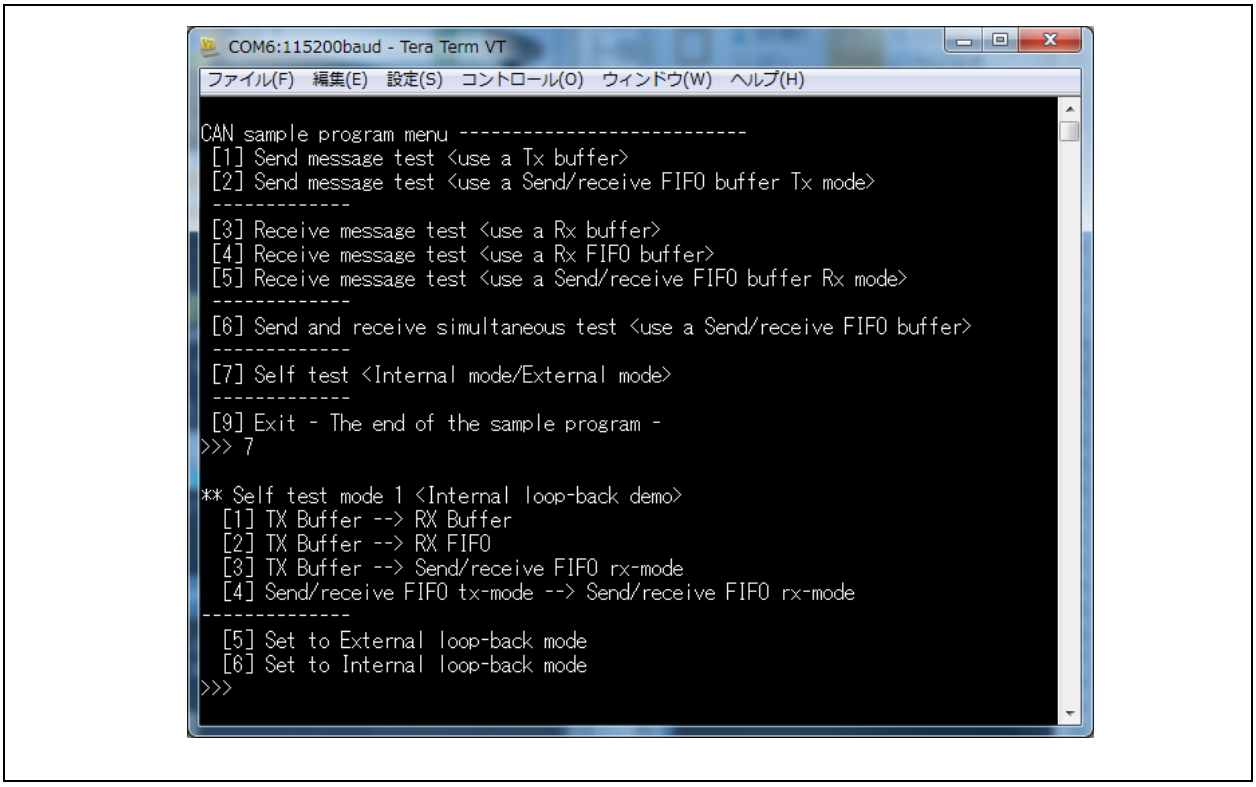


図 4-34 [7] Self test <Internal/Externa>選択時のセルフテスト選択メニュー

各セルフテストの内容を以下に説明します。

- 1: Tx Buffer → Rx Buffer

  送信バッファからメッセージを送信し、受信バッファでメッセージを受信するテスト。

- 2: Tx Buffer → Rx FIFO buffer

  送信バッファからメッセージを送信し、受信FIFO バッファでメッセージを受信するテスト。

- 3: Tx Buffer → Send/receive FIFO buffer Rx mode

  送信バッファからメッセージを送信し、送受信FIFO バッファ(受信モード)でメッセージを受信するテスト。

- 4: Send/receive FIFO buffer Tx mode → Send/receive FIFO buffer Rx mode

  送受信FIFO バッファ(送信モード)からメッセージを送信し、送受信FIFO バッファ(受信モード) でメッセージを受信するテスト。

- 5: Set to External loop back mode

  セルフテストモード0（外部ループバックモード）の選択。

- 6: Set to Internal loop back mode

  セルフテストモード1（内部ループバックモード）の選択。

4.6.6　サンプルプログラム設定値

本サンプルプログラムは、以下の設定値にて動作します。

- チャネル：CAN1（固定）
- 通信速度： 1 Mbps（固定）
- 送信メッセージ：1 メッセージ（8 byte）の繰り返し
- 受信ルール：1 ルール（メッセージID：0x120 のみ受信可）（固定）
- 送信バッファ番号：0（固定）
- 受信バッファ番号：1（固定）
- 受信FIFO バッファ番号：0（固定）
- 送受信FIFO バッファ番号(送信モード)：0（固定）
- 送受信FIFO バッファ番号(受信モード)：1（固定）
- 送受信FIFO バッファのリンク先送信バッファ番号：2（固定）

サンプルデータ

- メッセージID：0x120（固定）
- メッセージタイプ：標準ID（固定）
- データフォーマット：データ・フレーム（固定）
- メッセージデータ：00h ～ FFh（8byte を 1 個のメッセージとして送信）

受信バッファ

- バッファサイズ：1024byte（サイズを超えての受信時、先頭から上書き）

4.6.7　送信テスト

- 準備

  開発環境に接続している評価ボード(A)と、別のPC にセットアップされた開発環境に接続している評価ボード(B)をケーブルで接続して下さい。4.6.3 使用準備（送信テスト・受信テスト）項を参照して下さい。

- 操作

  最初に、受信側（別の PC にセットアップされた開発環境に接続している評価ボード(B)）を受信モードで待たせます。

  （評価ボード(B)で、メインメニューの[3]、[4]、[5]の何れかを選択）

  次に送信側（評価ボード(A)）で、メインメニューの[1] または [2] を選択して下さい。

  送信側（評価ボード(A)）からメッセージを送信し続けます。

  送信テストを終了する場合は、送信側で何かキーを押下して下さい。

  送信テストが終了します。また、受信側も受信モードから抜けます。

- 送信データ

  メッセージ ID：　0x120

  メッセージタイプ：標準 ID（値 0）

  データフォーマット：　データ・フレーム（値 0）

  メッセージデータ：　一回の送信メッセージは、8byte で構成する

  （サンプルでは、0x00 ～ 0xFF までの連続するデータを使用しています）

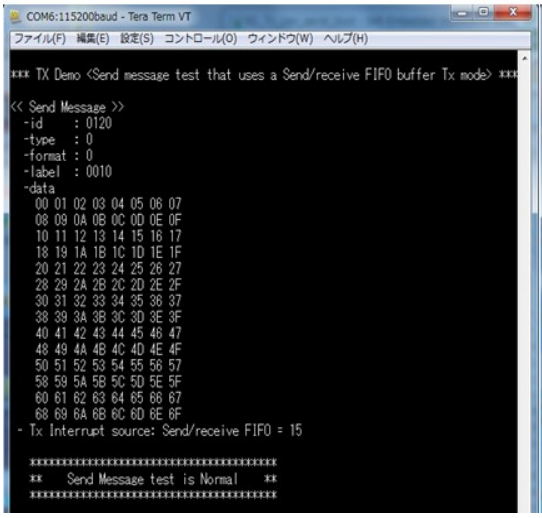
  デリミタコード：　メッセージ終了コード

  （サンプルでは、0x00 を 8byte 続けて送信することでメッセージの終了を判断しています）

- 結果

  下記に送受信 FIFO バッファ(送信モード)を使ったメッセージ送信テスト結果を示します。

送信側のログ出力結果



受信側のログ出力結果

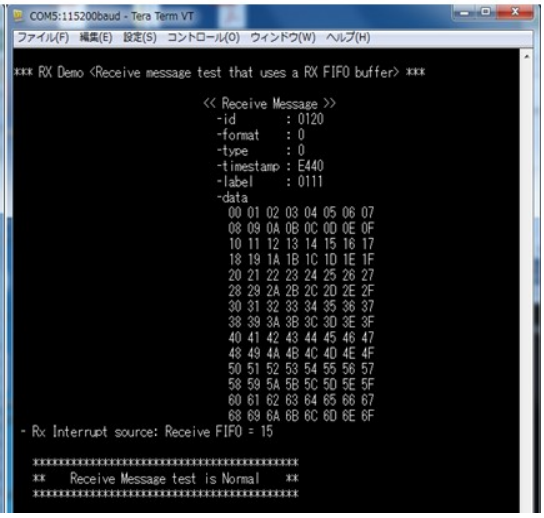


図 4-35 サンプルプログラムの送信テスト結果例

- 注意事項

  上記の操作項において、受信側（別のPC にセットアップされた開発環境に接続している評価ボード (B)）が受信モードになっていない状態で、送信側（評価ボード(A)）からメッセージを送信した場合、下記のエラーとなります。

  受信側（評価ボード(B)）の準備を終えてから、送信側（評価ボード(A)）よりメッセージを送信して下さい。

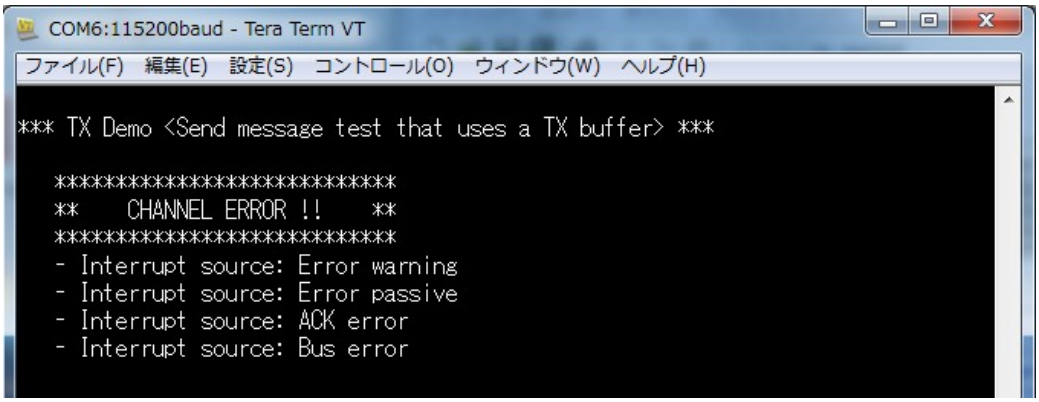


図 4-36 サンプルプログラムの送信テストエラー結果例

4.6.8　受信テスト

- 準備

  開発環境に接続している評価ボード(A)と、別のPC にセットアップされた開発環境に接続している評価ボード(B)をケーブルで接続して下さい。4.6.3 使用準備（送信テスト・受信テスト）項を参照して下さい。

- 操作

  最初に、評価ボード(A)のメインメニューで[3]、[4]、[5]の何れかを選択し、受信モードにして下さい。

  次に、別のPC にセットアップされた開発環境に接続している評価ボード(B)のメインメニューで[1] または[2]を選択して下さい。

  送信側（評価ボード(B)）からメッセージを送信し続けます。

  受信側（評価ボード(A)）の受信テストを終了する場合は、送信側（評価ボード(B)）で何かキーを押下して下さい。

  送信側（評価ボード(B)）は、送信テストを終了し、受信側（評価ボード(A)）も受信モードから抜けます。

- 結果

  下記に送受信FIFO バッファ(受信モード)でメッセージを受信するテスト結果を示します。

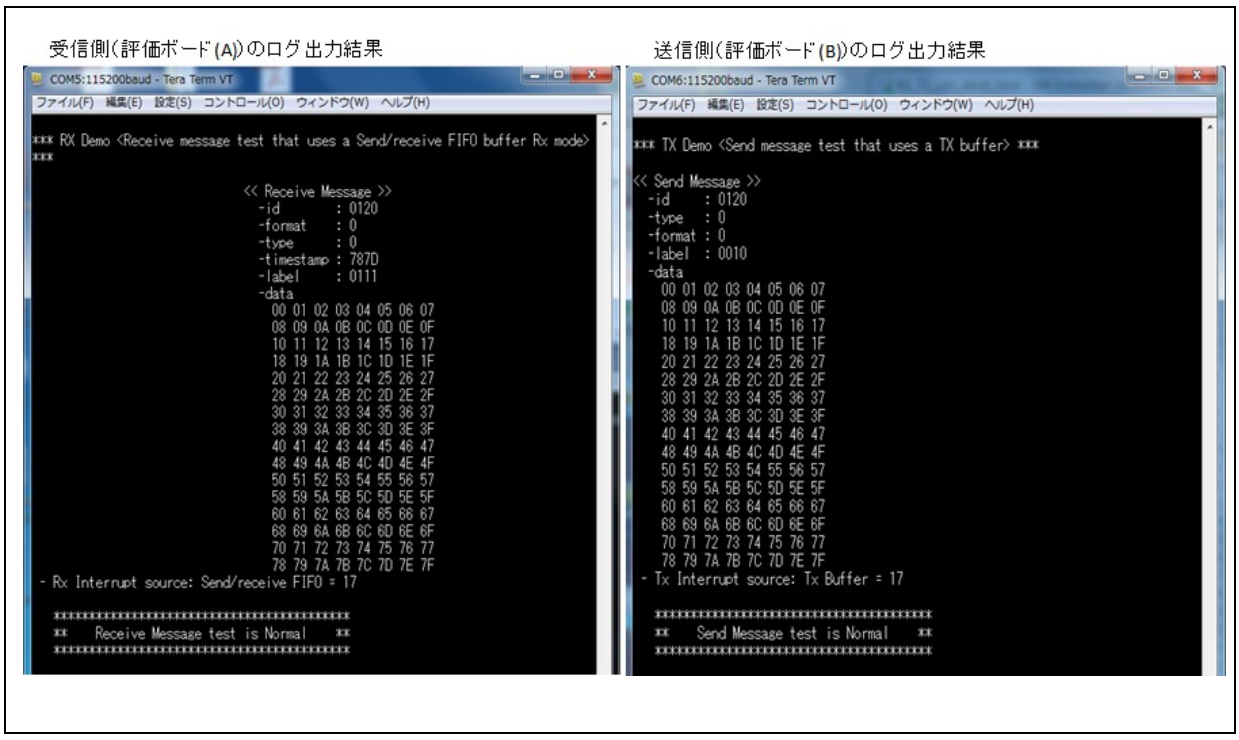


図 4-37 サンプルプログラムの受信テスト結果例

4.6.9　受信を受け付けながらメッセージ送信テスト

- 準備

  開発環境に接続している評価ボード(A)と、別のPC にセットアップされた開発環境に接続している評価ボード(B)をケーブルで接続して下さい。4.6.3 使用準備（送信テスト・受信テスト）項を参照して下さい。

- 操作

  1. 評価ボード(A)側のメインメニューで[6]を選択します。
  2. キー押下を促すガイダンスが出力されます。
  3. 別の PC にセットアップされた開発環境に接続している評価ボード(B)側のメインメニューで同じく[6]を選択して下さい。
  4. キー押下を促すガイダンスが出力されます。
  5. 両方にキー押下を促すガイダンスが出力されたことを確認後、評価ボード(B)側で何かキーを押下して下さい。

  1～5 の操作を行うと、評価ボード(A)側、評価ボード(B)側から共にメッセージを送信し続けます。

  テストを終了する場合は、評価ボード(A)側、評価ボード(B)側、何れかで何かキーを押下して下さい。

  評価ボード(A)側、評価ボード(B)側、共に受信を受け付けながらメッセージ送信テストを終了します。

- 結果

下記に受信を受け付けながらメッセージ送信するテストのテスト結果を示します。

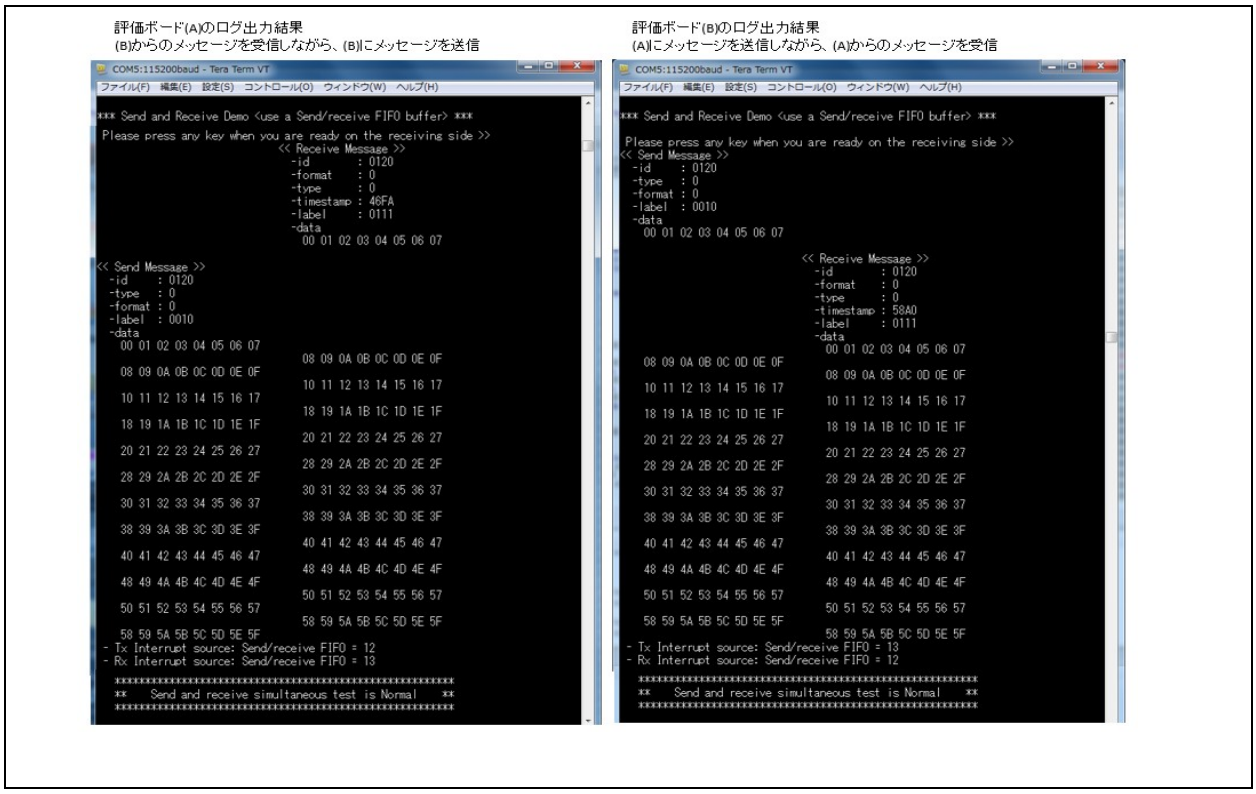


図 4-38 サンプルプログラムの受信を受け付けながらメッセージ送信テスト結果例

4.6.10　セルフテスト

- 準備

  開発環境に接続している評価ボード(A)のみを使用します。4.6.2 使用準備（セルフテスト）項を参照して下さい。

- 操作

  評価ボード(A)のメインメニューから[7]を選択し、セルフテストメニューを表示させます。

  セルフテストメニューから、テストを行いたい項目を選択して下さい。

  セルフテストは、外部ループバックモードと内部ループバックモードでのテストができます。

  セルフテストメニューの[5]または[6]で選択して下さい。（デフォルトは、内部ループバックモードです）

- 結果

  下記に送信バッファからメッセージを送信し、送受信FIFO バッファ(受信モード)でメッセージを受信するセルフテストの結果を示します。

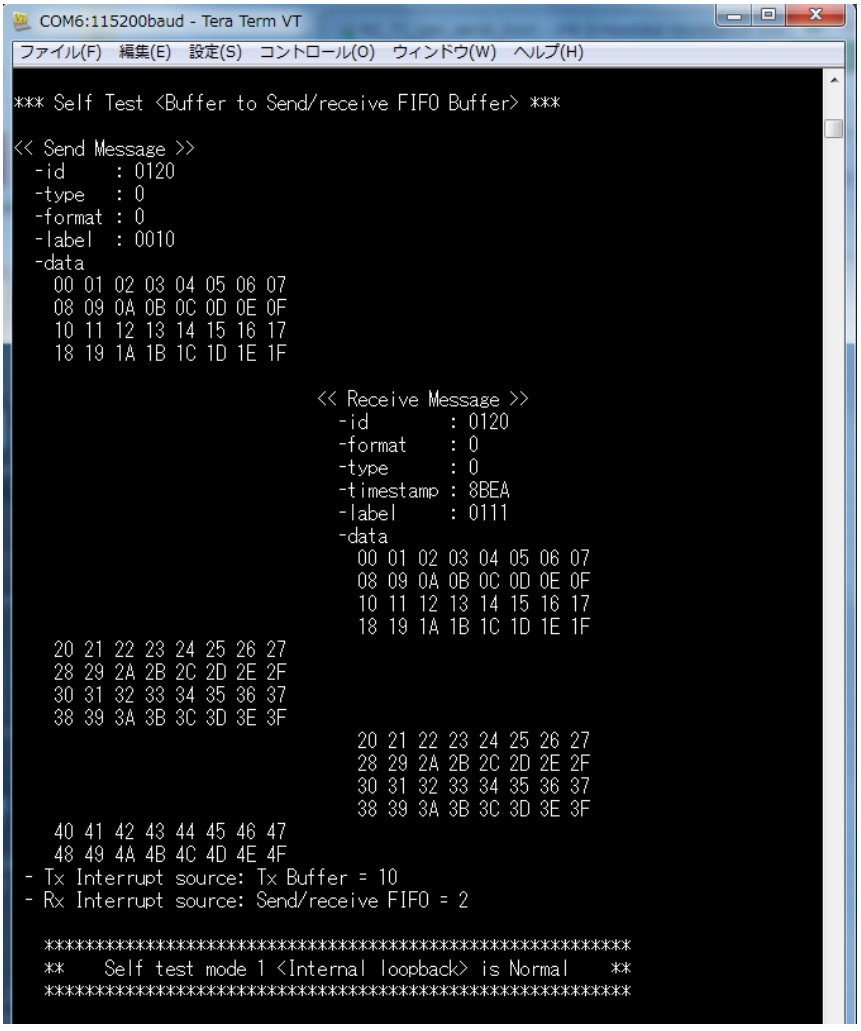


図 4-39 サンプルプログラムのセルフテスト結果例

5. RSPI サンプルソフト

5.1 概要

本章では、評価ボード上に実装される RSPI コントローラ 0 チャネル(RSPI0)を制御する RSPI ドライバを使用したサンプルプログラムについて説明します。

RSPI サンプルプログラムの特長を以下に示します。
PC と接続し、ターミナルソフトを用いることで、RSPI 通信を用いたマスタ~スレーブ間のデータの送受信結果を画面上より確認できます。

対象デバイス
EC-1

本サンプルを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

5.2　関数一覧

「表 5-1　関数一覧」を以下に示します。

表 5-1　関数一覧

関数名	概要	スコープ	定義ファイル
main	メイン処理	global	main.c
cmt_standby	CMT モジュール停止処理	local	main.c
get_sw1	スイッチ１ステータス取得	local	main.c
get_sw8	スイッチ８ステータス取得	local	main.c
board_init	ボード設定初期化	global	board_RenesaEva.c
board_output	ボード出力データ格納処理	global	board_RenesaEva.c
board_input	入力ポート状態取得処理	global	board_RenesaEva.c
board_input_dipsw	dip スイッチ状態取得処理	global	board_RenesaEva.c
board_rspi_init	RSPI ポート初期化処理	global	board_RenesaEva.c

5.3　関数詳細

5.3.1　main

(1)　概要

サンプルプログラムメイン処理

(2)　C 言語形式

int main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。
処理内容は 5.4.1 メイン処理 を参照してください。

(5)　戻り値

なし

5.3.2　cmt_standby

(1)　概要

CMT モジュール停止処理

(2)　C 言語形式

static void cmt_standby (void);

(3)　パラメータ

なし

(4)　機能

CMT モジュールを停止します。
リセットに関連するレジスタと低消費電力機能への書き込みを無効にします。

(5)　戻り値

なし

5.3.3　get_sw1

(1)　概要

スイッチ１ステータス取得

(2)　C 言語形式

static uint8_t get_sw1(void);

(3)　パラメータ

なし

(4)　機能

スイッチ１の状態を取得します。

(5)　戻り値

戻り値	意味
tmp_port	スイッチの状態

5.3.4　get_sw8

(1)　概要

スイッチ８ステータス取得

(2)　C 言語形式

static uint8_t get_sw8(void);

(3)　パラメータ

なし

(4)　機能

スイッチ 8 の状態を取得します。

(5)　戻り値

戻り値	意味
tmp_port	スイッチの状態

5.3.5　board_init

(1)　概要

ボード設定初期化

(2)　C 言語形式

void board_init(void);

(3)　パラメータ

なし

(4)　機能

ボード設定の初期化を行います。

(5)　戻り値

なし

5.3.6　board_output

(1)　概要

ボード出力データ格納処理

(2)　C 言語形式

void board_output(uint8_t output);

(3)　パラメータ

I/O	パラメータ	説明
I	uint8_t　output	出力データ値

(4)　機能

ポート出力データレジスタへ出力データの書き込みを行います。

(5)　戻り値

なし

5.3.7 board_input

(1) 概要

入力ポート状態取得処理

(2) C 言語形式

uint8_t board_input(void);

(3) パラメータ

なし

(4) 機能

入力ポート端子の状態を取得します。

(5) 戻り値

戻り値	意味
uint8_t	入力ポートの状態

5.3.8 board_input_dipsw

(1) 概要

dip スイッチ状態取得処理

(2) C 言語形式

uint8_t board_input_dipsw (void);

(3) パラメータ

なし

(4) 機能

dip スイッチの状態を取得します。

(5) 戻り値

戻り値	意味
uint8_t	dipスイッチの状態

5.3.9 board_rspi_init

(1) 概要

RSPI ポート初期化処理

(2) C 言語形式

void board_rspi_init(void);

(3) パラメータ

なし

(4) 機能

RSPI ポートの設定を初期化します。

(5) 戻り値

なし

5.4　フローチャート

5.4.1　メイン処理

サンプルコードのメイン処理のフローチャートを示します。

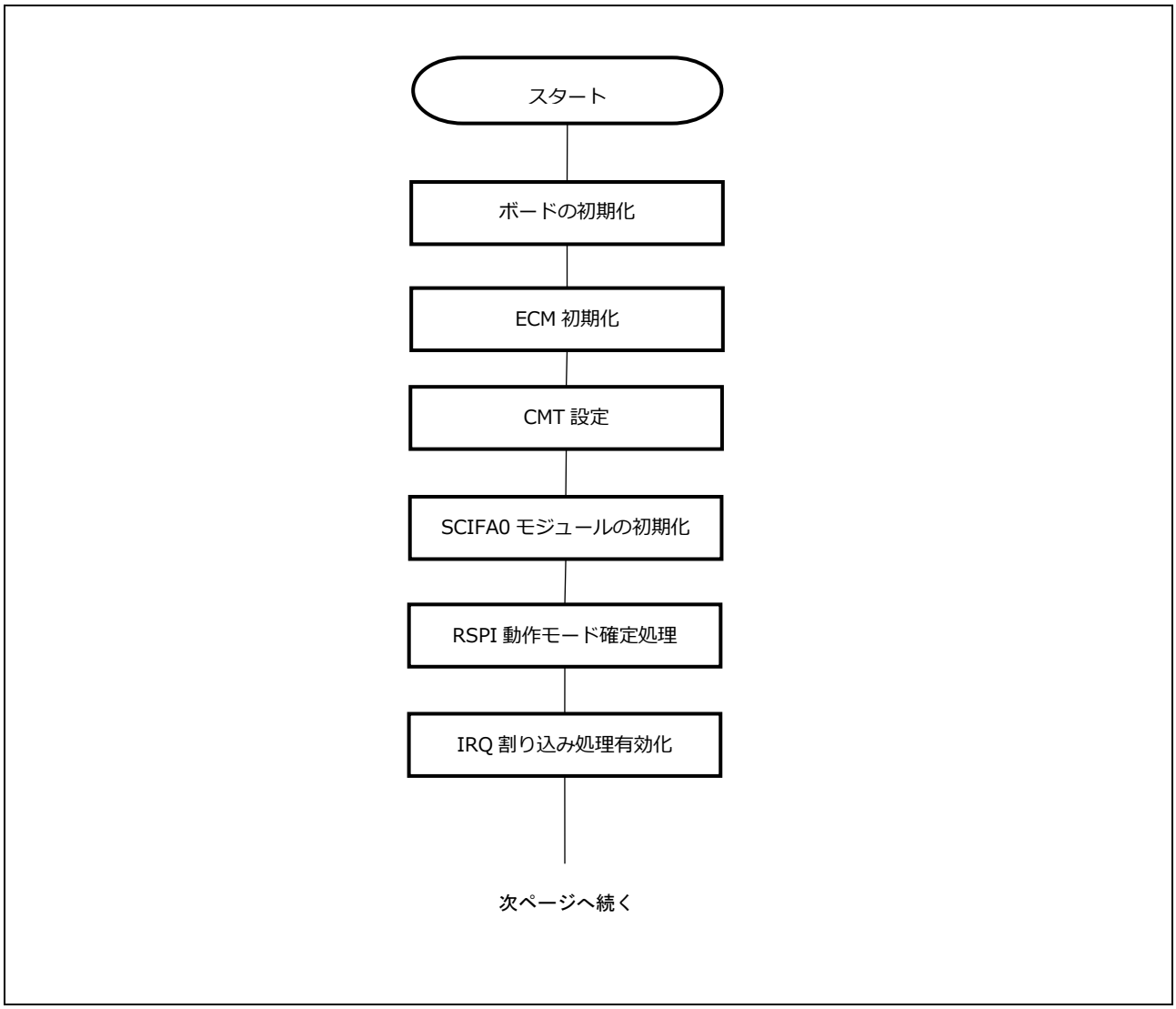


図 5-1 サンプルコードのメイン処理（1/2）

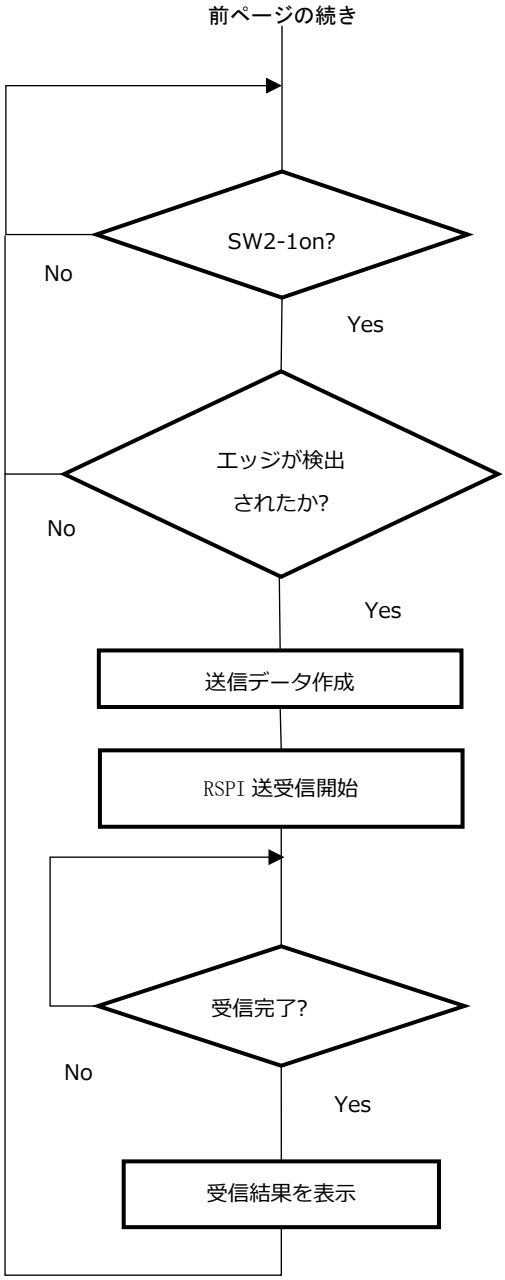


図 5-2 サンプルコードのメイン処理（2/2）

5.5　チュートリアル

5.5.1　動作概要

RSPI サンプルプログラムの機能概要を表 5-2 に示します。また、図 5-3 にシステムブロック図を示します。

表 5-2 動作概要

機能	概要
通信チャネル	・チャネル0 を使用
動作モード	・SPI 動作：3 線式 ・通信モード：マスタモード（通信のみ） ・データ長：16bit ・ビットレート：6.25[Mbps]スレーブ時
動作概要	dip スイッチにより動作モードを切り替え
動作結果表示	コンソールに結果出力

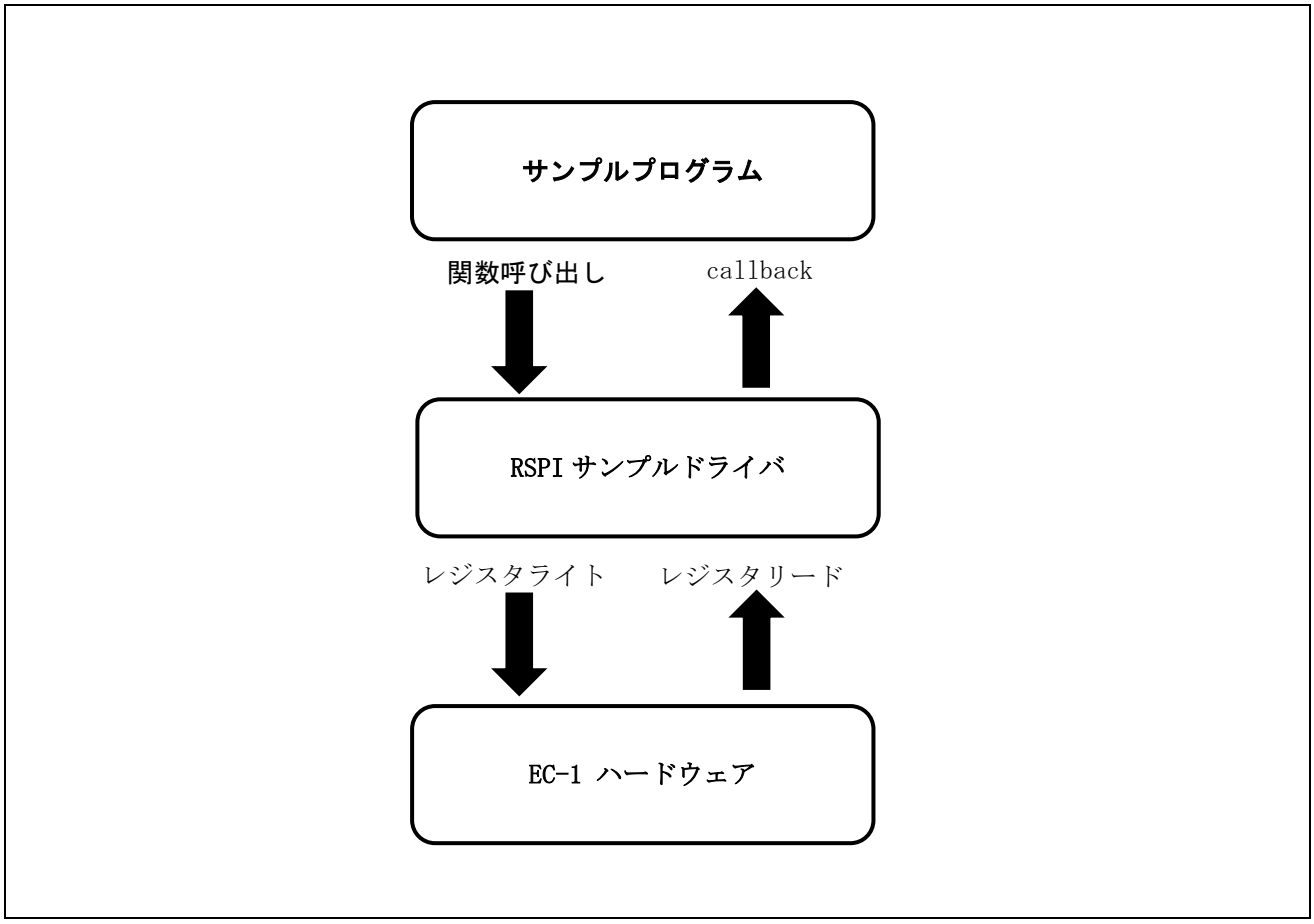


図 5-3 システムブロック図

5.5.2　使用準備

本サンプルプログラム実行の際は図 5-4 の通り基盤を 2 台（マスタ用/スレーブ用）用意し接続ください。

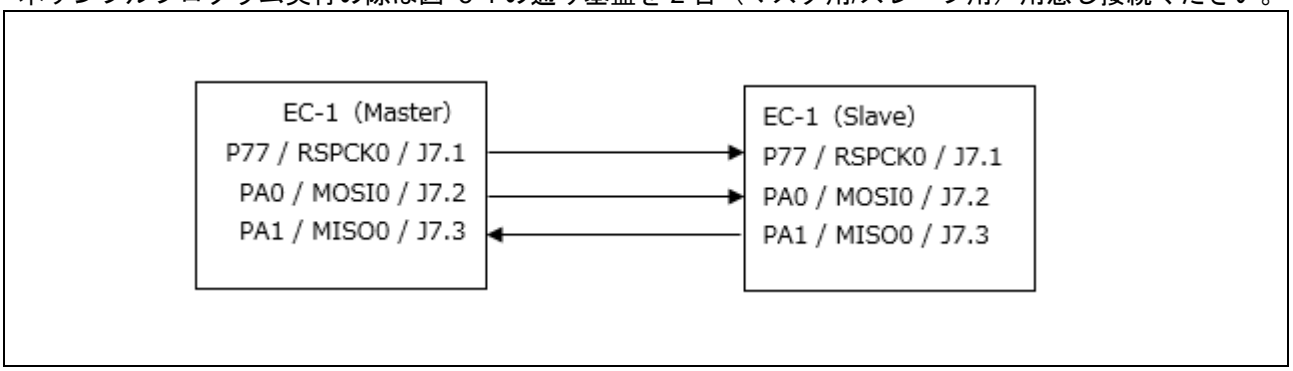


図 5-4 接続方法

5.5.3　ターミナルソフト（Tera Term）

本サンプルプログラムは SCIFA の調歩同期式通信を用い、ホスト PC と RS-232C インタフェースの COM ポート通信を行います。ホスト PC のターミナルソフトの設定は下記のとおりです。

・転送速度：115200bps
・キャラクタ長：8 ビット
・ストップビット長：1 ビット
・パリティ機能：なし
・ハードウェアフロー制御：なし

5.5.4　サンプルプログラムの機能

・起動時の dip スイッチ(SW2-8)の状態によりマスター/スレーブ切り替え
・dip スイッチ(SW2-1)操作にてスレーブ受信、マスター送信を実行
・送信データの一部と受信データを UART 表示

5.5.5　サンプルプログラム実行例

(1)基板の SW2-8 をマスタは ON、スレーブは OFF にした状態とします。
(2)基板の USB 仮想 COM ポート(CN4)とホスト PC を接続し、ターミナルソフトを起動の上、サンプルプログラムを実行します。
(3)スレーブの SW2-1 を ON にして受信データ待ち状態にします。
(4)次にマスタの SW2-1 を ON にするとマスタから送信を行います。
　マスタ~スレーブ間でデータの送受信が行われ結果がログとして表示されます。
(5)データ転送を続ける場合は、マスタ/スレーブの SW2-1 をいったん OFF にしてから(3)、(4)を実行します。

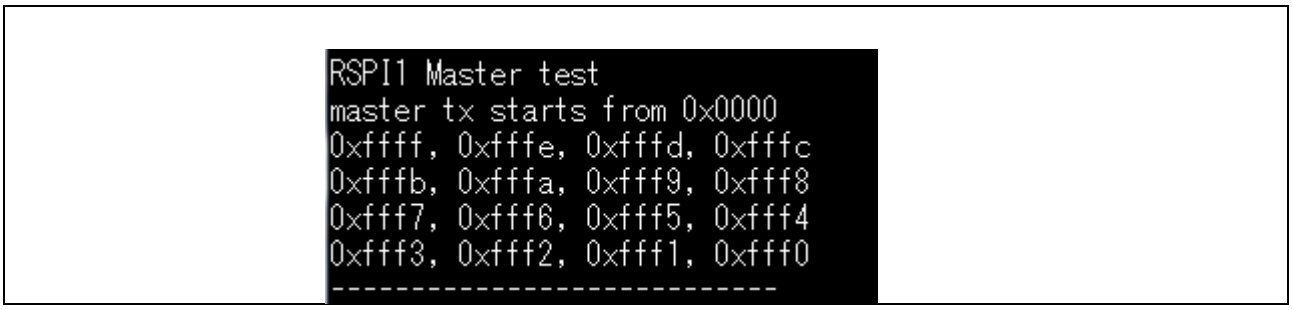


図 5-5 マスタ実行結果例

「master tx starts from 0x0000」はマスタが 0x0000 から始まる 16 ワードインクリメントデータを送信したことを示します。
次の 16 ワード(0xffff~0xfff0)はスレーブからの送信データを受信した結果を表示しています。

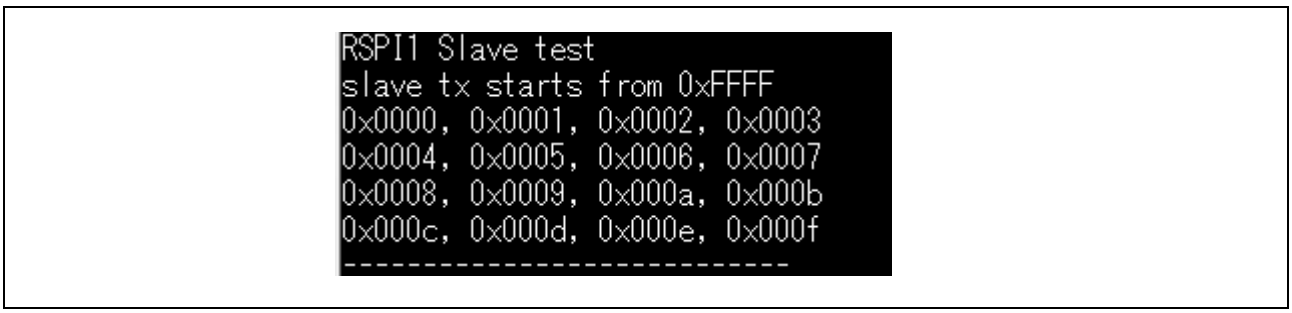


図 5-6 スレーブ実行結果例

「slave tx starts from 0xFFFF」はスレーブが 0xFFFF から始まる 16 ワードデクリメントデータを送信したことを示します。
次の 16 ワード(0x0000~0x000f)はマスタからの送信データを受信した結果を表示しています。

6. SFLASH_WRITER サンプルソフト

6.1 概要

本章では、評価ボード上に実装される シリアル・フラッシュ ROM を制御するシリアル・フラッシュ ROM ドライバを使用したサンプルプログラムについて説明します。

SFLASH_WRITER サンプルプログラムの特長を以下に示します。

PC と接続し、ターミナルソフトを用いることで、シリアル・フラッシュ ROM への Read／Write／Erace を行うことができます。

対象デバイス

EC-1

本サンプルを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

6.2 定数一覧

表 6-1 にサンプルコードで使用する定数を示します。

表 6-1 サンプルコードで使用されている定数

定数名	設定値	内容
CMDBUF_MAX_SIZE	256	コマンドバッファサイズ
RECIEVEBUF_MAX_SIZE	256*1024	受信バッファ最大サイズ
EXT_RAM_ADDR	&recv_buf[0][0]	データ退避アドレス
IMG_SRC_AADR	(uint32_t)EXT_RAM_ADDR	データ退避アドレス
IMG_SRC_SIZE	RECIEVEBUF_MAX_SIZE	受信バッファ最大サイズ
ER_OK	(ER_RET)0	Normal end (no error)
ER_NG	(ER_RET)-1	Abnormal end (error)
ER_SYS	(ER_RET)(2 * ER_NG)	Undefined error
ER_PARAM	(ER_RET)(3 * ER_NG)	Invalid parameter
ER_NOTYET	(ER_RET)(4 * ER_NG)	Incomplete processing
ER_NOMEM	(ER_RET)(5 * ER_NG)	Out of memory
ER_BUSY	(ER_RET)(6 * ER_NG)	Busy
ER_INVAL	(ER_RET)(7 * ER_NG)	Invalid state
ER_TIMEOUT	(ER_RET)(8 * ER_NG)	Timeout occurs

6.3 構造体/共用体/列挙型一覧

以下に、サンプルコードで使用する構造体/共用体/列挙型を示します。

表 6-2 writer_command_info_t構造体

メンバ名	内容
uint8_t* name;	command name
ER_RET (*func)(int32_t, uint8_t**);	command function
uint8_t* help;	command help

6.4　関数一覧

「表 6-3　関数一覧」を以下に示します。

表 6-3　関数一覧

関数名	概要	スコープ	定義ファイル
main	メイン処理	global	main.c
port_init	ポート設定初期化	local	main.c
flash_writer	flash_writer メイン処理	global	flash_writer.c
exec_getline	入力文字列の読み込み	local	flash_writer.c
exec_command	コマンドの実行	local	flash_writer.c
exec_cmd_help	help コマンド処理	local	flash_writer.c
exec_cmd_sfw	sfw コマンド処理	local	flash_writer.c
exec_cmd_sfr	sfr コマンド処理	local	flash_writer.c
exec_cmd_sfe	sfe コマンド処理	local	flash_writer.c
exec_cmd_flash_info	flash_info コマンド処理	local	flash_writer.c
exec_flash_write	フラッシュ書き込み処理	local	flash_writer.c
exec_flash_read	フラッシュ読み込み処理	local	flash_writer.c
exec_flash_erase	フラッシュ削除処理	local	flash_writer.c
btld_flash_write	フラッシュ書き込み処理	local	flash_writer.c
btld_flash_erase	フラッシュ削除処理	local	flash_writer.c
hex2dec	10 進数⇔16 進数　変換処理	local	flash_writer.c
*dmac_memcpy	dmac_memcpy 関数	local	flash_writer.c

6.5　関数詳細

6.5.1　main

(1)　概要

サンプルプログラムメイン処理

(2)　C 言語形式

void main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。「6.6.1 メイン処理」にフローチャートを記載しています。

- ECM 初期化
- ポート設定初期化
- SCIFA0 設定
- flash_writer()プログラムの呼び出し

(5)　戻り値

なし

6.5.2　port_init

(1)　概要

ポート設定初期化

(2)　C 言語形式

static void port_init (void);

(3)　パラメータ

なし

(4)　機能

ポート設定の初期化を行います。

(5)　戻り値

なし

6.5.3　flash_writer

(1)　概要

flash_writer 処理

(2)　C 言語形式

void flash_writer(void);

(3)　パラメータ

なし

(4)　機能

main()処理にて呼び出される flash_writer 処理となります。
ターミナルソフトより入力を受付し、入力されたコマンドに応じて処理を行います。
図 6-2　にフローチャートを記載しています。

(5)　戻り値

なし

6.5.4　exec_getline

(1)　概要

入力文字列の読み込み

(2)　C 言語形式

static ER_RET exec_getline(uint8_t* str, uint32_t echo);

(3)　パラメータ

I/O	パラメータ	説明
O	uint8_t* str	バッファポインタ
I	uint32_t echo	0：エコーバック 以外：エコーバック

(4)　機能

入力の待ち受けを行います。
引数にて指定された文字列を区切り文字か文字列の終端まで読み込みます。

(5)　戻り値

戻り値	意味
ER_NG	バッファオーバーフロー
ER_NOTYET	文字列の終端を非検出
ER_OK	文字列の終端を検出

6.5.5　exec_command

(1)　概要

コマンドの実行

(2)　C 言語形式

static ER_RET exec_command(uint8_t* str);

(3)　パラメータ

I/O	パラメータ	説明
I	uint8_t* str	コマンドバッファのポインタ

(4)　機能

引数で渡されたスペース区切りの文字列の先頭がサポートコマンドと一致した場合、対応した処理を実行する。

(5)　戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.6　exec_help

(1)　概要

help コマンド処理

(2)　C 言語形式

static ER_RET exec_cmd_help(int32_t argc, uint8_t* argv[]);

(3)　パラメータ

I/O	パラメータ	説明
I	int32_t argc	引数の数
I	uint8_t* argv[]	引数文字列（コマンド名）

(4)　機能

ターミナルソフトにて「help」と入力された際の処理です。
コマンドリストを表示します。
図 6-6 ヘルプコマンド実行結果例を参照ください。

(5)　戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.7　exec_cmd_sfw

(1)　概要

sfw コマンド処理

(2)　C 言語形式

static ER_RET exec_cmd_sfw(int32_t argc, uint8_t* argv[]);

(3)　パラメータ

I/O	パラメータ	説明
I	int32_t argc	引数の数
I	uint8_t* argv[]	引数文字列（コマンド名、アドレス、データ）

(4)　機能

ターミナルソフトにて「sfw」と入力された際の処理です。
exec_flash_write()処理を呼び出しフラッシュへの書き込みを行います。
図 6-7 Flash 書き込みコマンド実行結果例を参照ください。

(5)　戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.8　exec_cmd_sfr

(1)　概要

sfr コマンド処理

(2)　C 言語形式

static ER_RET exec_cmd_sfr(int32_t argc, uint8_t* argv[]);

(3)　パラメータ

I/O	パラメータ	説明
I	int32_t argc	引数の数
I	uint8_t* argv[]	引数文字列（コマンド名、アドレス、データ長）

(4)　機能

ターミナルソフトにて「sfr」と入力された際の処理です。
exec_flash_read()処理を呼び出し、フラッシュの読み出しをします。
図 6-8 Flash 読み込みコマンド実行結果例 を参照ください。

(5)　戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.9　exec_cmd_sfe

(1)　概要

sfe コマンド処理

(2)　C 言語形式

static ER_RET exec_cmd_sfe(int32_t argc, uint8_t* argv[]);

(3)　パラメータ

I/O	パラメータ	説明
I	int32_t argc	引数の数
I	uint8_t* argv[]	引数文字列（コマンド名、アドレス）

(4)　機能

ターミナルソフトにて「sfe」と入力された際の処理です。
exec_flash_erace()処理を呼び出し、フラッシュの指定のセクタ消去を行います。
図 6-9 Flash 消去コマンド実行結果例 を参照ください。

(5)　戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.10　exec_cmd_flash_info

(1)　概要

flash_info コマンド処理

(2)　C 言語形式

static ER_RET exec_cmd_flash_info(int32_t argc, uint8_t* argv[]);

(3)　パラメータ

I/O	パラメータ	説明
I	int32_t argc	引数の数
I	uint8_t* argv[]	引数文字列

(4)　機能

ターミナルソフトにて「flash_info」と入力された際の処理です。
デバイス情報を表示します。
図 6-10 Flash デバイス情報表示コマンド実行結果例　を参照ください。

(5)　戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.11 exec_flash_write

(1) 概要

フラッシュ書き込み処理

(2) C 言語形式

static ER_RET exec_flash_write(int32_t argc, uint8_t* argv[]);

(3) パラメータ

I/O	パラメータ	説明
I	int32_t argc	引数の数
I	uint8_t* argv[]	引数文字列（コマンド名、アドレス、データ）

(4) 機能

パラメータやセクタの判定を行い、フラッシュへの書き込みを行います。

(5) 戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.12　exec_flash_read

(1)　概要

フラッシュ読み込み処理

(2)　C 言語形式

static ER_RET exec_flash_read(int32_t argc, uint8_t* argv[]);

(3)　パラメータ

I/O	パラメータ	説明
I	int32_t argc	引数の数
I	uint8_t* argv[]	引数文字列（コマンド名、アドレス、データ長）

(4)　機能

パラメータの判定を行い、フラッシュより読み込みを行います。

(5)　戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.13　exec_flash_erase

(1)　概要

フラッシュ削除処理

(2)　C 言語形式

static ER_RET exec_flash_erase(int32_t argc, uint8_t* argv[]);

(3)　パラメータ

I/O	パラメータ	説明
I	int32_t argc	引数の数
I	uint8_t* argv[]	引数文字列（コマンド名、アドレス、データ長）

(4)　機能

パラメータ、セクタの判定を行い指定したフラッシュ領域の削除を行います。

(5)　戻り値

戻り値	意味
ER_OK	エラーなし
ER_NG	エラー

6.5.14　btld_flash_write

(1)　概要

フラッシュ書き込み処理

(2)　C 言語形式

static ER_RET btld_flash_write(uint16_t* buf, uint32_t addr, uint32_t size);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t* buf	バッファアドレス
I	uint32_t addr	プログラムベースアドレス
I	uint32_t size	プログラムサイズ

(4)　機能

引数によりシリアル・フラッシュ ROM の任意の位置へ 1-16Byte のデータをビッグエンディアンで書き込みます。

(5)　戻り値

戻り値	意味
ER_OK	成功
ER_NG	パラメータエラー

6.5.15　btld_flash_erase

(1)　概要

フラッシュ削除処理

(2)　C 言語形式

static ER_RET btld_flash_erase(uint32_t addr, uint32_t size);

(3)　パラメータ

I/O	パラメータ	説明
I	uint32_t addr	プログラムベースアドレス
I	uint32_t size	プログラムサイズ

(4)　機能

指定されたアドレスが存在するセクタを 1 セクタ消去します。

(5)　戻り値

戻り値	意味
ER_OK	成功
ER_NG	パラメータエラー

6.5.16　hex2dec

(1)　概要

10 進数⇔16 進数 変換処理

(2)　C 言語形式

static ER_RET hex2dec(uint8_t* str);

(3)　パラメータ

I/O	パラメータ	説明
I	uint8_t* str	16進文字列

(4)　機能

引数にて指定された 16 進文字列を 10 進数へと変換します。

(5)　戻り値

戻り値	意味
ER_OK	成功
ER_NG	パラメータエラー

6.5.17　dmac_memcpy

(1)　概要

dmac_memcpy 関数

(2)　C 言語形式

static void *dmac_memcpy(void *dst, const void *src, uint32_t n);

(3)　パラメータ

I/O	パラメータ	説明
I	void *dst	宛先アドレス
I	const void *src	コピー元アドレス
I	uint32_t n	コピーバイト数

(4)　機能

メモリに直接アクセスしメモリコピーを行います。

(5)　戻り値

戻り値	意味
ER_OK	成功
ER_NG	パラメータエラー

6.6　フローチャート

6.6.1　メイン処理

サンプルコードのメイン処理のフローチャートを示します。

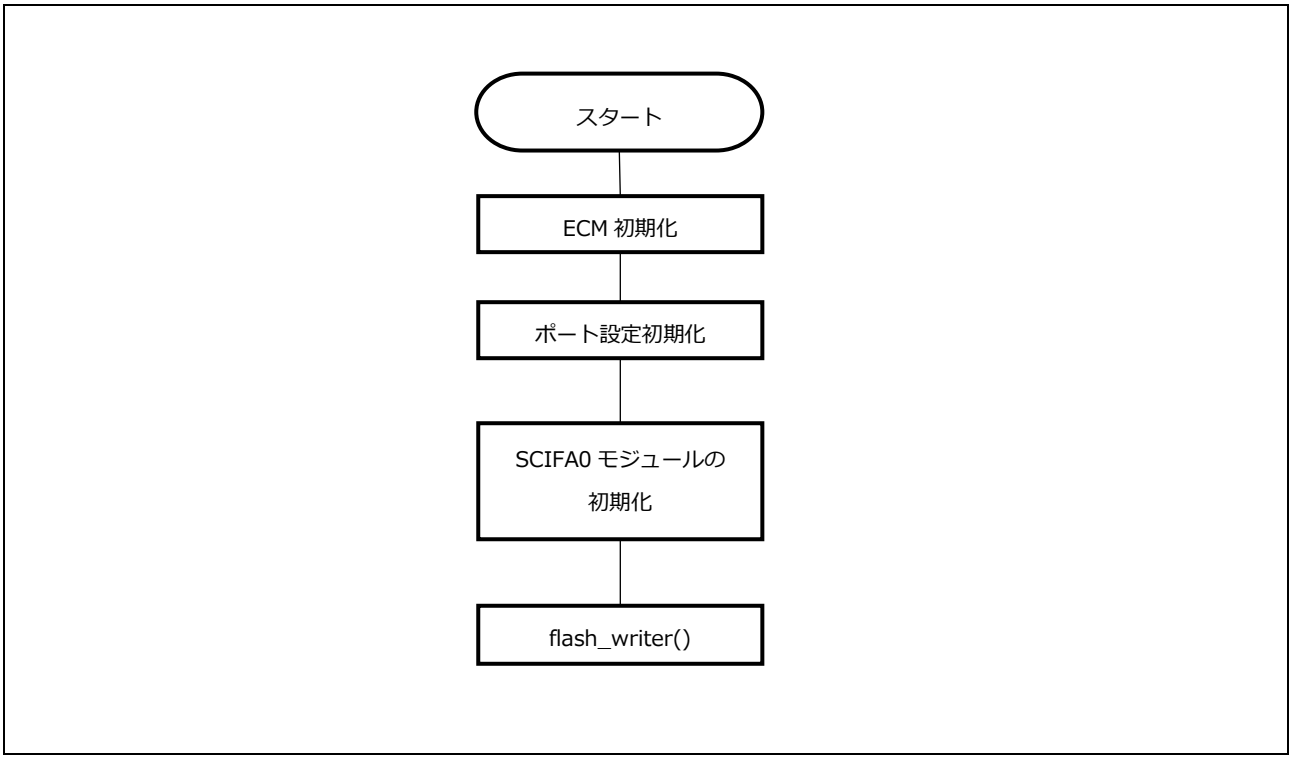


図 6-1 サンプルコードのメイン処理

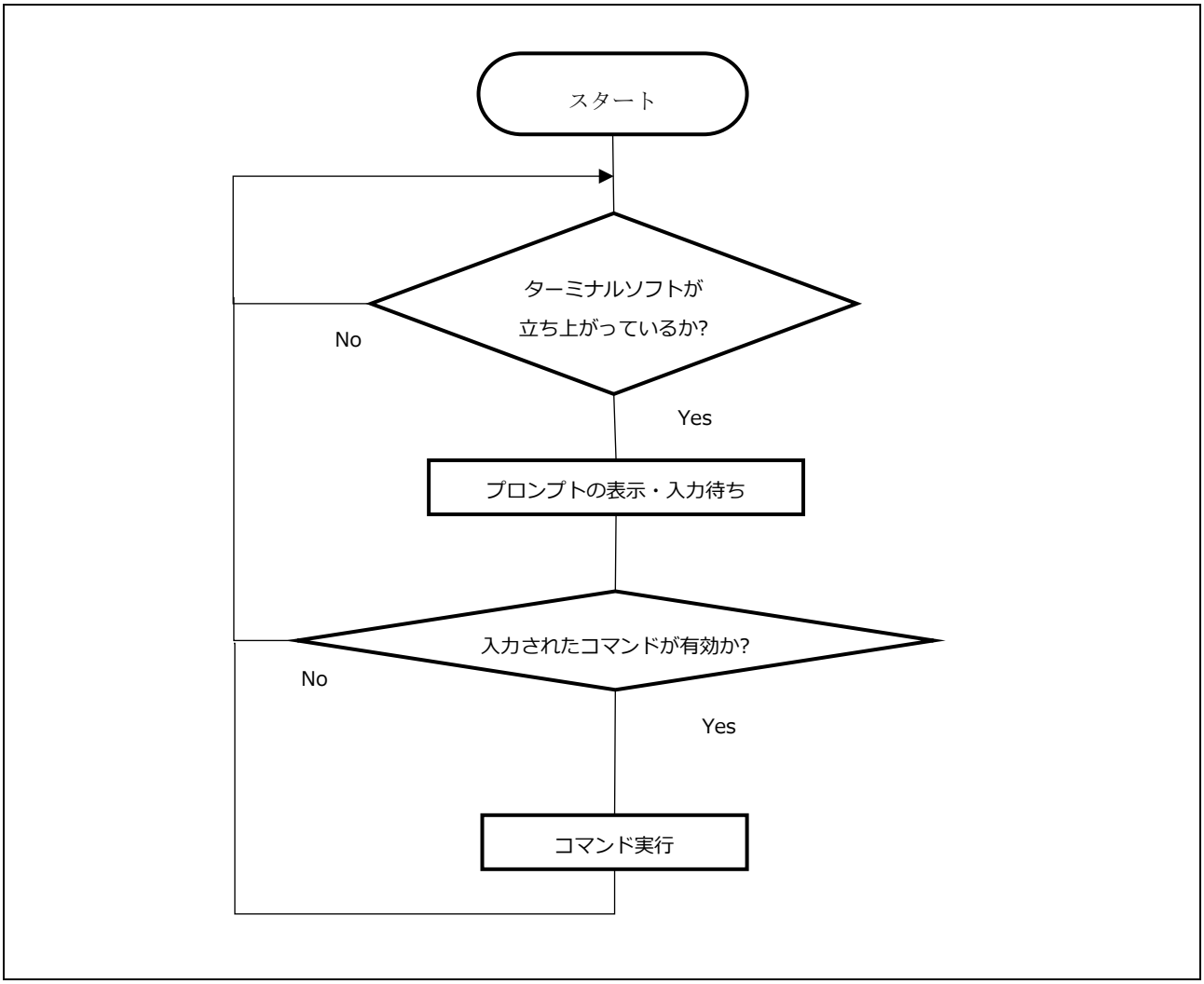


図 6-2 flash_writer()処理

6.7　チュートリアル

6.7.1　動作概要

flash_writer サンプルプログラムの機能概要を表 6-4 動作概要に示します。また、図 6-3 にシステムブロック図を示します。

表 6-4 動作概要

機能	概要
通信チャネル	チャネル0（SCIFA0）を使用
シリアル通信方式	調歩同期式
クロック	SERICLK = 150MHz
送信／受信	シリアルデータ送信／受信
転送速度	115200bps
キャラクタ長	8 ビット
ストップビット長	1 ビット
パリティ機能	なし
ハードウェアフロー制御	なし
動作概要	コマンドにて動作を選択
動作結果表示	コンソールに結果出力

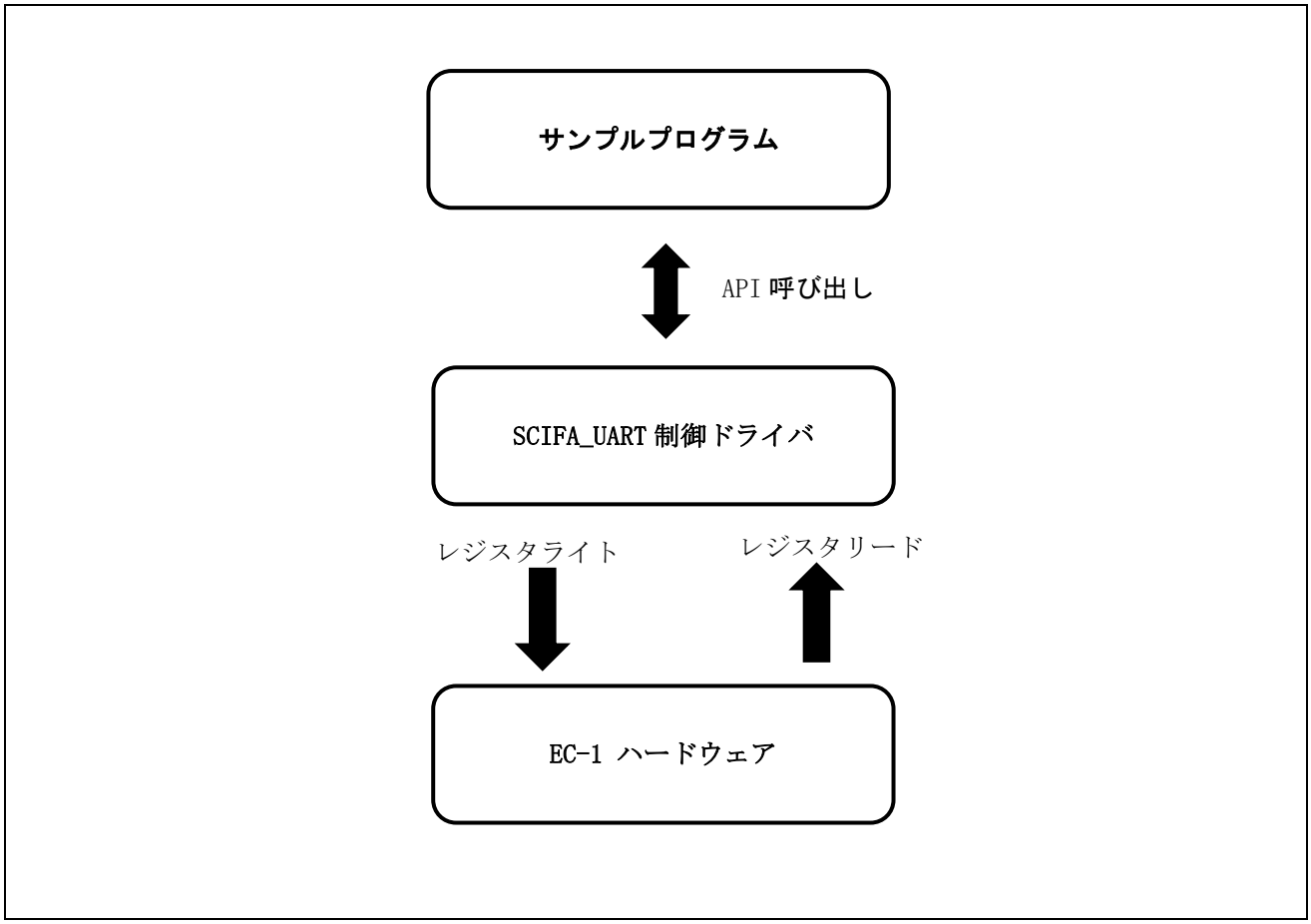


図 6-3 システムブロック図

6.7.2　使用準備

本サンプルプログラムは SCIFA の調歩同期式通信を用い、ホスト PC と RS-232C インタフェースの COM ポート通信を行います。通信設定については 5.7.3 ターミナルソフト（Tera Term）に記載しています。

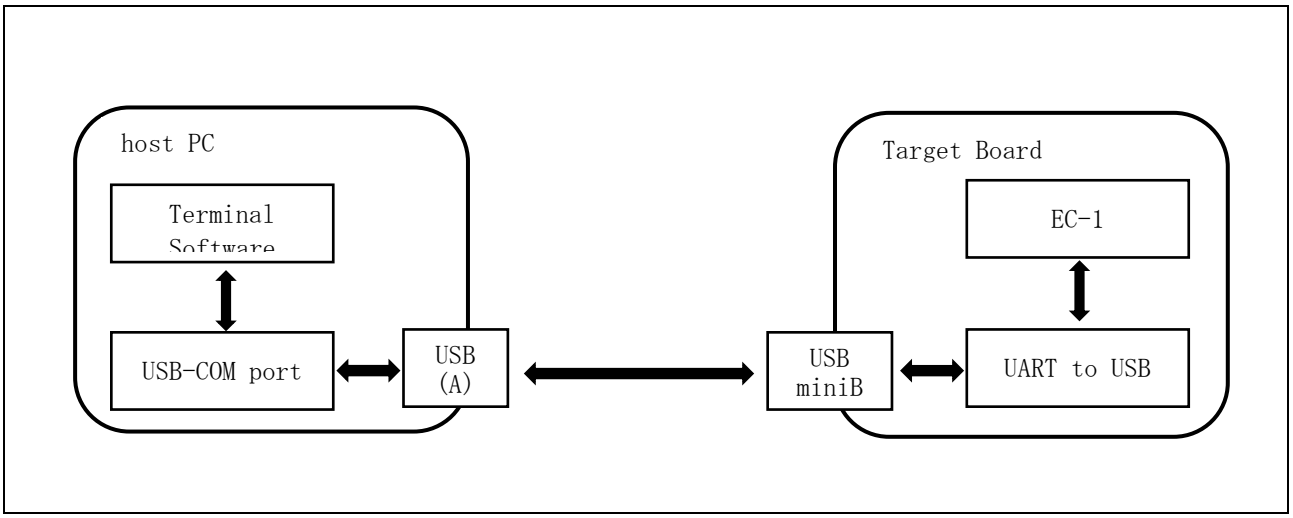


図 6-4 接続方法

6.7.3　ターミナルソフト（Tera Term）

ホスト PC のターミナルソフトの設定は下記のとおりです。

・転送速度：115200bps
・キャラクタ長：8 ビット
・ストップビット長：1 ビット
・パリティ機能：なし
・フロー制御：なし

6.7.4　サンプルプログラムの機能

ターミナルソフト（Tera Term）を起動しておき、本サンプルプログラムを起動します。
図 6-5 のようなコマンドプロンプト「EC-1>」が表示されている場合にホスト PC からのコマンドを受け付けます。

図 6-5 サンプルプログラム起動時画面表示

コマンドの内容を表 6-5 に示します。

表 6-5 コマンド一覧

書式	内容	制御内容
help	ヘルプ表示	制御コマンド一覧を表示
sfw [address][data]	SerialFlash 書き込み	SerialFlash へ[address]から [data]を NByte 書き込む ※1、※2
sfr [address][length]	SerialFlash 読み込み	SerialFlash の[address]から[length]Byte 読み込む ※1
sfe [address]	SerialFlash 消去	SerilaFlash の[address]から 1 セクタ分消去または全消去を行う ※1
flash_info	Flash デバイス情報表示	SerialFlash のデバイス情報を表示

※1　“0x”を付けずに 16 進数で指定する。
※2　[data]は 1 から 16byte の可変長で指定可能

6.7.5　サンプルプログラム実行例

・ヘルプコマンド（help）

制御コマンド一覧を表示します。

図 6-6 ヘルプコマンド実行結果例

・Flash 書き込みコマンド（sfw）

Flash の任意の位置へ 1-16Byte のデータをビッグエンディアンで書き込みます。

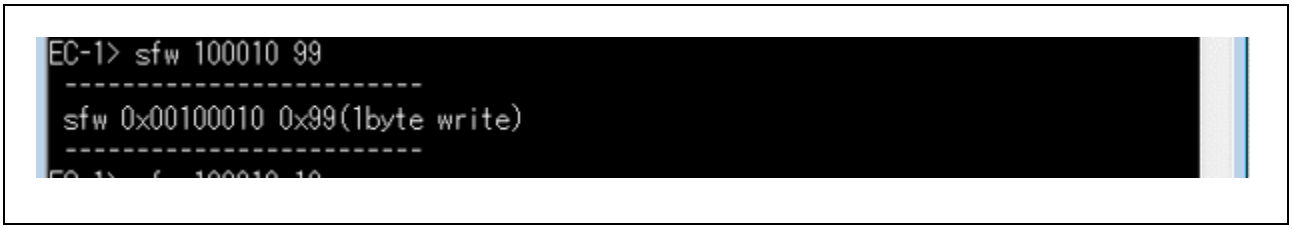


図 6-7 Flash書き込みコマンド実行結果例

・Flash 読み込みコマンド（sfr）

指定された位置から指定バイト数分 Flash の内容を表示します。

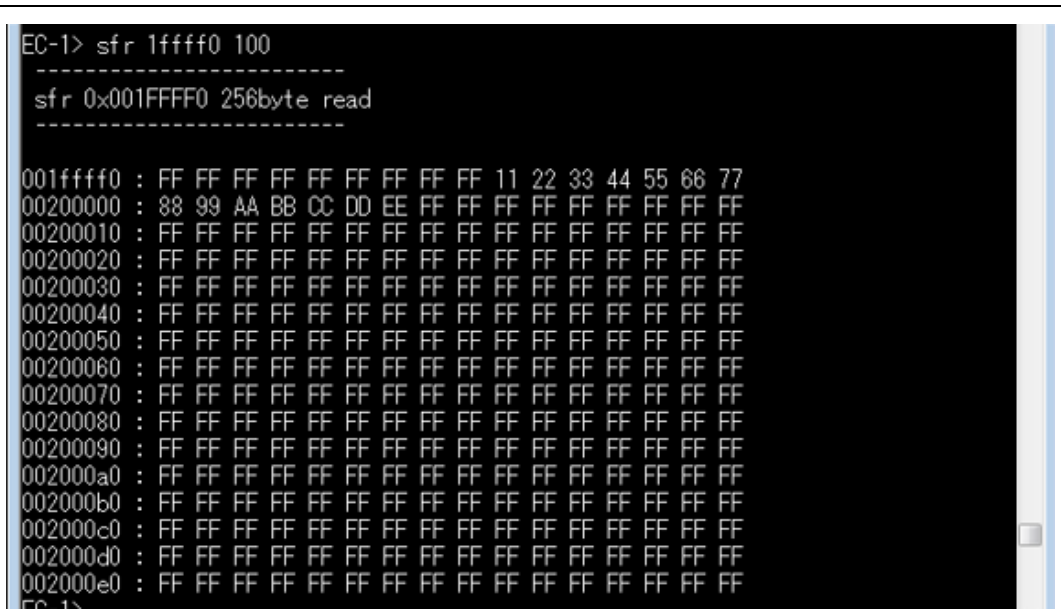


図 6-8 Flash読み込みコマンド実行結果例

・Flash 消去コマンド（sfe）

指定されたアドレスが存在するセクタを 1 セクタ消去します。

address に"all"を指定することで全消去します。

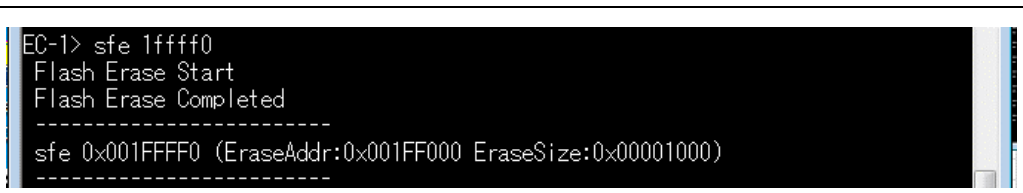


図 6-9 Flash消去コマンド実行結果例

・Flashデバイス情報表示（flash_info）

SerialFlash のデバイス情報を表示します。

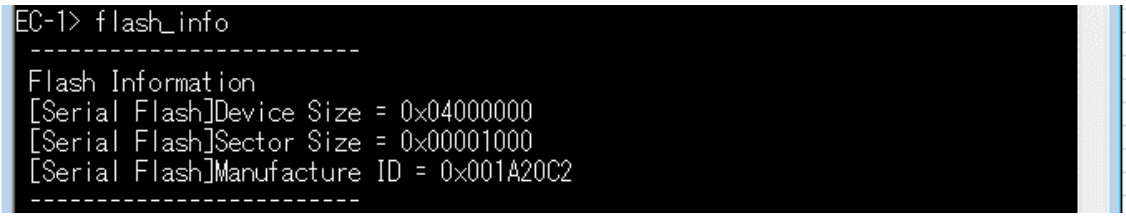


図 6-10 Flashデバイス情報表示コマンド実行結果例

7. USB ファンクションサンプルソフト

7.1 概要

本章では、評価ボード上に実装される USB ファンクションを制御する USB ファンクションドライバを使用したサンプルプログラムについて説明します。

USB ファンクションサンプルプログラムの特長を以下に示します。
USB ファンクションドライバは以下のモジュール(PCD,PCDC)の組み合わせにより動作します。

(1) ・Peripheral control driver (PCD)

PCD は、USB2.0HS ファンクションモジュールの H/W 制御を行います。PCD は Renesas が提供するサンプルデバイスクラスドライバ又はユーザが作成したペリフェラルデバイスクラスドライバ(PDCD)と組み合わせることで動作します。
以下に本モジュールがサポートしている機能を示します。
・ デバイスの接続／切断、サスペンド／レジューム、USB バスリセット処理
・ パイプ 0 でコントロール転送
・ パイプ 1～9 でバルク転送、インタラプト転送、アイソクロナス転送（CPU/DMA アクセスを選択可能）
・ USB1.1 / 2.0 / 3.0 ホストと Enumeration
制限事項
本モジュールには以下の制限事項があります。
・ パイプ情報設定関数でパイプ使用方法を制限しています。
→受信パイプは SHTNAK 機能でトランザクションカウンタを使用します。
・ 型の異なるメンバで構造体を構成しています。
（コンパイラによって構造体メンバのアドレスアライメントずれが発生することがあります）
・データ転送中のサスペンドはサポートしていません。データ転送が完了したことを確認の上、サスペンドを実行して下さい。

※詳細は「7.8 ペリフェラルコントロールドライバ（PCD）」を参照してください。
※詳細は「7.9 ペリフェラルデバイスクラスドライバ（PDCD）」を参照してください。

(2) ・USB Peripheral Communications Device Class Driver (PCDC)

PCDC は、USB コミュニケーションデバイスクラス仕様の Abstract Control Model として実装されていて PCD と組み合わせることでペリフェラルデバイスクラスドライバ（PDCD）として動作し、USB ホストとの通信を行うことができます。
以下に本モジュールがサポートしている機能を示します。
・ USB ホストとのデータ転送
・ CDC クラスリクエストに応答
・ コミュニケーションデバイスクラスノーティフィケーション送信サービスの提供

※詳細は「7.10 ペリフェラルコミュニケーションデバイスクラス（PCDC）」を参照してください。

用語一覧

APL : Application program
CDC : Communications Devices Class
H/W : Renesas USB device
PCD : Peripheral control driver
PDCD : Peripheral device class driver (device driver and USB class driver)
PCDC : Periperal Communications Devices Class
USB : Universal Serial Bus

対象デバイス

EC-1

本サンプルを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

7.2 定数一覧

表 7-1 にサンプルコードで使用する定数を示します。

表 7-1 サンプルコードで使用されている定数

定数名	設定値	内容
CDC_DATA_LEN	512	USB 転送データ長
EVENT_MAX	5	イベント最大数
TASK_LOOPS_BETWEEN_INSTRUCTIONS	0x500000	初期接続メッセージ格納アドレス
ALIGN_SIZE	0x20	アライメントサイズ

7.3　構造体/共用体/列挙型一覧

以下に、サンプルコードで使用する構造体/共用体/列挙型を示します。

表 7-2 DEV_INFO_t構造体

メンバ名	内容
uint16_t　state;	State of the application
uint16_t　event_cnt;	Event count
uint16_t　event[EVENT_MAX];	Event no.

表 7-3 STATE_t列挙型

メンバ名	内容
STATE_ATTACH;	アタッチ処理
STATE_DATA_TRANSFER	データ転送処理
STATE_DETACH	デタッチ処理

表 7-4 EVENT_t列挙型

メンバ名	内容
EVENT_NONE	イベントなし
EVENT_DETACH	USB デバイス切断
EVENT_CONFIGERD	USB デバイス接続完了
EVENT_USB_READ_START	USB データ受信要求
EVENT_USB_READ_COMPLETE	USB データ受信完了
EVENT_USB_WRITE_START	USB データ送信要求
EVENT_USB_WRITE_COMPLETE	USB データ送信完了
EVENT_COM_NOTIFY_START	Class Notification“SerialState”　送信要求
EVENT_COM_NOTIFY_COMPLETE	Class Notification“SerialState”　送信完了

表 7-5 USB_PCDC_APL_STATE列挙型

メンバ名	内容
APP_STATE_IDLE	アイドル状態
APP_STATE_ECHO_MODE	エコーモード

7.4　関数一覧

「表 7-6　関数一覧」を以下に示します。

表 7-6　関数一覧

関数名	概要	スコープ	定義ファイル
main	メイン処理	global	main.c
port_init	ポート設定初期化	global	main.c
icu_init	割り込み設定	global	main.c
usbf_main	USB ファンクションメイン処理	global	r_usb_pcdc_apl.c
cdc_connect_wait	USB 接続待機処理	global	r_usb_pcdc_apl.c
cdc_detach_device	デタッチ処理	global	r_usb_pcdc_apl.c
cdc_data_transfer	データ転送処理	global	r_usb_pcdc_apl.c
cdc_read_complete	USB 受信完了コールバック関数	global	r_usb_pcdc_apl.c
cdc_write_complete	USB 送信完了コールバック関数	global	r_usb_pcdc_apl.c
cdc_demo_complete	デモ手順送信完了コールバック関数	global	r_usb_pcdc_apl.c
cdc_configured	デバイスコンフィガード用コールバック関数	global	r_usb_pcdc_apl.c
cdc_detach	デタッチ処理コールバック関数	global	r_usb_pcdc_apl.c
cdc_default	デフォルト処理コールバック関数	global	r_usb_pcdc_apl.c
cdc_suspend	サスペンド処理コールバック関数	global	r_usb_pcdc_apl.c
cdc_resume	レジューム処理コールバック関数	global	r_usb_pcdc_apl.c
cdc_interface	インタフェース処理コールバック関数	global	r_usb_pcdc_apl.c
cdc_registration	デバイスドライバ登録	global	r_usb_pcdc_apl.c
apl_init	APL 初期化	global	r_usb_pcdc_apl.c
cdc_event_set	イベント発行	global	r_usb_pcdc_apl.c
cdc_event_get	イベント取得	global	r_usb_pcdc_apl.c

7.5　関数詳細

7.5.1　main

(1)　概要

サンプルプログラムメイン処理

(2)　C 言語形式

void main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。

- ECM 初期化
- 割り込み設定
- ポート設定初期化
- SCIFA0 設定
- usbf_main()プログラムの呼び出し

(5)　戻り値

なし

7.5.2 port_init

(1) 概要

ポート設定初期化

(2) C 言語形式

void port_init (void);

(3) パラメータ

なし

(4) 機能

ポート設定の初期化を行います。

(5) 戻り値

なし

7.5.3 icu_init

(1) 概要

割り込み設定

(2) C 言語形式

void icu_init (void);

(3) パラメータ

なし

(4) 機能

割り込み処理を有効にします。

(5) 戻り値

なし

7.5.4　usbf_main

(1)　概要

USB ファンクションメイン処理

(2)　C 言語形式

void usbf_main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。図 7-1 にフローチャートを記載しています。
ステートとそのステートに関連するイベントにより管理を行っています。
ステートに関連するイベントの確認を行い、そのイベントに応じた処理を行います。
そのイベント処理後、アプリケーション は、必要に応じてステートを変化させます。

- STATE_ATTACH　アタッチ処理
- STATE_DATA_TRANSFER　データ転送処理
- STATE_DETACH　デタッチ処理

(5)　戻り値

なし

7.5.5　cdc_connect_wait

(1)　概要

USB 接続待機処理

(2)　C 言語形式

uint16_t cdc_connect_wait(void);

(3)　パラメータ

なし

(4)　機能

ステートが STATE_ATTACH 状態且つ USB ホストに接続がない状態の際にコールされる関数となります。USB ホストとの Enumeration が完了するとコールバック関数 cdc_configured()がイベント EVENT_CONFIGURD を発行します。

EVENT_CONFIGURD では、ステートを STATE_DATA_TRANSFER に変更し、EVENT_USB_READ_START を発行します。

詳細は「7.6.2 ステートとイベントの管理」を参照してください。

(5)　戻り値

戻り値	意味
USB_TRUE	成功
USB_FALSE	失敗

7.5.6　cdc_detach_device

(1)　概要

デタッチ処理

(2)　C 言語形式

void cdc_detach_device(void);

(3)　パラメータ

なし

(4)　機能

ステートが STATE_DETACH 状態時にコールされる関数となります。
変数のクリア処理、ステートを STATE_ATTACH に変更します。
詳細は「7.6.2 ステートとイベントの管理」を参照してください。

(5)　戻り値

なし

7.5.7　cdc_deta_transfer

(1)　概要

データ転送処理

(2)　C 言語形式

uint16_t cdc_data_transfer(void);

(3)　パラメータ

なし

(4)　機能

ステートが STATE_DATA_TRANSFER 状態の際にコールされる関数となります。
USB ホストから最初のデータを受信するまでの間、定期間隔で USB ホストに初期接続
メッセージを送信します。最初のデータ受信後は、USB ホストからのデータを受信し、その受信したデータをそのまま USB ホストに送信するループバック処理を行います。
詳細は「7.6.2 ステートとイベントの管理」を参照してください。

(5)　戻り値

戻り値	意味
USB_TRUE	成功
USB_FALSE	失敗

7.5.8　cdc_read_complete

(1)　概要

USB 受信完了コールバック関数

(2)　C 言語形式

void cdc_read_complete(USB_UTR_t *mess);

(3)　パラメータ

I/O	パラメータ	説明
I	USB_UTR_t　*mess	メッセージ

(4)　機能

ステートが STATE_DATA_TRANSFER 時にデータ受信処理が完了すると本コールバック関数がコールされます。
EVENT_USB_READ_COMPLETE を発行します。

(5)　戻り値

なし

7.5.9　cdc_write_complete

(1)　概要

USB 送信完了コールバック関数

(2)　C 言語形式

void cdc_write_complete(USB_UTR_t *mess);

(3)　パラメータ

I/O	パラメータ	説明
I	USB_UTR_t　*mess	メッセージ

(4)　機能

ステートが STATE_DATA_TRANSFER 時にデータ送信処理が完了すると本コールバック関数がコールされます。
EVENT_USB_WRITE_COMPLETE を発行します。

(5)　戻り値

なし

7.5.10　cdc_demo_complete

(1)　概要

デモ手順送信完了コールバック関数

(2)　C 言語形式

void cdc_demo_complete(USB_UTR_t *mess);

(3)　パラメータ

I/O	パラメータ	説明
I	USB_UTR_t　*mess	メッセージ

(4)　機能

ステートが STATE_DATA_TRANSFER 時に USB ホストに定期間隔で初期接続メッセージを送信する度にコールされるコールバック関数となります。

(5)　戻り値

なし

7.5.11　cdc_configured

(1)　概要

デバイスコンフィガード用コールバック関数

(2)　C 言語形式

void cdc_configured(uint16_t data1);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　data1	※この引数は使用しません。

(4)　機能

USB ステートが遷移した時、PCD から呼び出されるコールバック関数になります。
ステートが configured 時の処理を行います。
ステートが STATE_ATTACH 時に USB ホストとの Enumeration が完了すると本コールバック関数がコールされます。
EVENT_CONFIGURD を発行します。

(5)　戻り値

なし

7.5.12　cdc_detach

(1)　概要

デタッチ処理コールバック関数

(2)　C 言語形式

void cdc_detach(uint16_t data1);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　data1	※この引数は使用しません。

(4)　機能

USB ステートが遷移した時、PCD から呼び出されるコールバック関数になります。
ステートが detach 時の処理を行います。
ステートを STATE_DETACH に変更します。

(5)　戻り値

なし

7.5.13　cdc_default

(1)　概要

デフォルト処理コールバック関数

(2)　C 言語形式

void cdc_default(uint16_t mode);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　mode	動作するデバイススピード USB_HSCONNECT：Hgih-Speed USB_FSCONNECT：Full-Speed USB_NOCONNECT：No connect

(4)　機能

USB ステートが遷移した時、PCD から呼び出されるコールバック関数になります。
ステートが default 時の処理を行います。
USB バスリセット処理時の設定処理を行います。

(5)　戻り値

なし

7.5.14　cdc_suspend

(1)　概要

サスペンド処理用コールバック関数

(2)　C 言語形式

void cdc_suspend(uint16_t data1);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　mode	※この引数は使用しません。

(4)　機能

USB ステートが遷移した時、PCD から呼び出されるコールバック関数になります。
ステートが suspend 時の処理を行います。

(5)　戻り値

なし

7.5.15　cdc_resume

(1)　概要

レジューム処理用コールバック関数

(2)　C 言語形式

void cdc_resume(uint16_t data1);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　data1	※この引数は使用しません。

(4)　機能

USB ステートが遷移した時、PCD から呼び出されるコールバック関数になります。
ステートが resume 時の処理を行います。

(5)　戻り値

なし

7.5.16　cdc_interface

(1)　概要

インタフェース処理用コールバック関数

(2)　C 言語形式

void cdc_interface(uint16_t data1);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　data1	※この引数は使用しません。

(4)　機能

USB ステートが遷移した時、PCD から呼び出されるコールバック関数になります。
ステートが interface 時の処理を行います。

(5)　戻り値

なし

7.5.17 cdc_registration

(1) 概要

デバイスドライバ登録

(2) C 言語形式

void cdc_registration(void);

(3) パラメータ

なし

(4) 機能

USB ドライバの初期設定を行います。

(5) 戻り値

なし

7.5.18 apl_init

(1) 概要

APL 初期化

(2) C 言語形式

void apl_init(void);

(3) パラメータ

なし

(4) 機能

サンプルアプリケーションの初期化処理を行います。

(5) 戻り値

なし

7.5.19　cdc_event_set

(1)　概要

イベント発行

(2)　C 言語形式

void cdc_event_set(uint16_t event);

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　event	イベント

(4)　機能

引数に指定したイベントの発行を行います。

(5)　戻り値

なし

7.5.20　cdc_event_get

(1)　概要

イベント取得

(2)　C 言語形式

uint16_t cdc_event_get(void);

(3)　パラメータ

なし

(4)　機能

イベントを取得して戻り値として返します。

(5)　戻り値

戻り値	意味
	イベント

7.6　フローチャート

7.6.1　アプリケーション（APL）

APL は以下の処理を行います。

① APL は、ステートとそのステートに関連するイベントにより管理を行っています。
APL では、まず、 ステートの確認を行います。なお、このステートは、APL が管理する構造体「表 7-2 DEV_INFO_t 構造体」のメンバに格納されています。

② APL はそのステートに関連するイベントの確認を行い、そのイベントに応じた処理を行います。
そのイベント処理後、APL は必要に応じてステートを変化させます。なお、このイベントは、APL が管理する構造体「表 7-2 DEV_INFO_t 構造体」のメンバに格納されています。

以下に、APL の処理概要を示します。

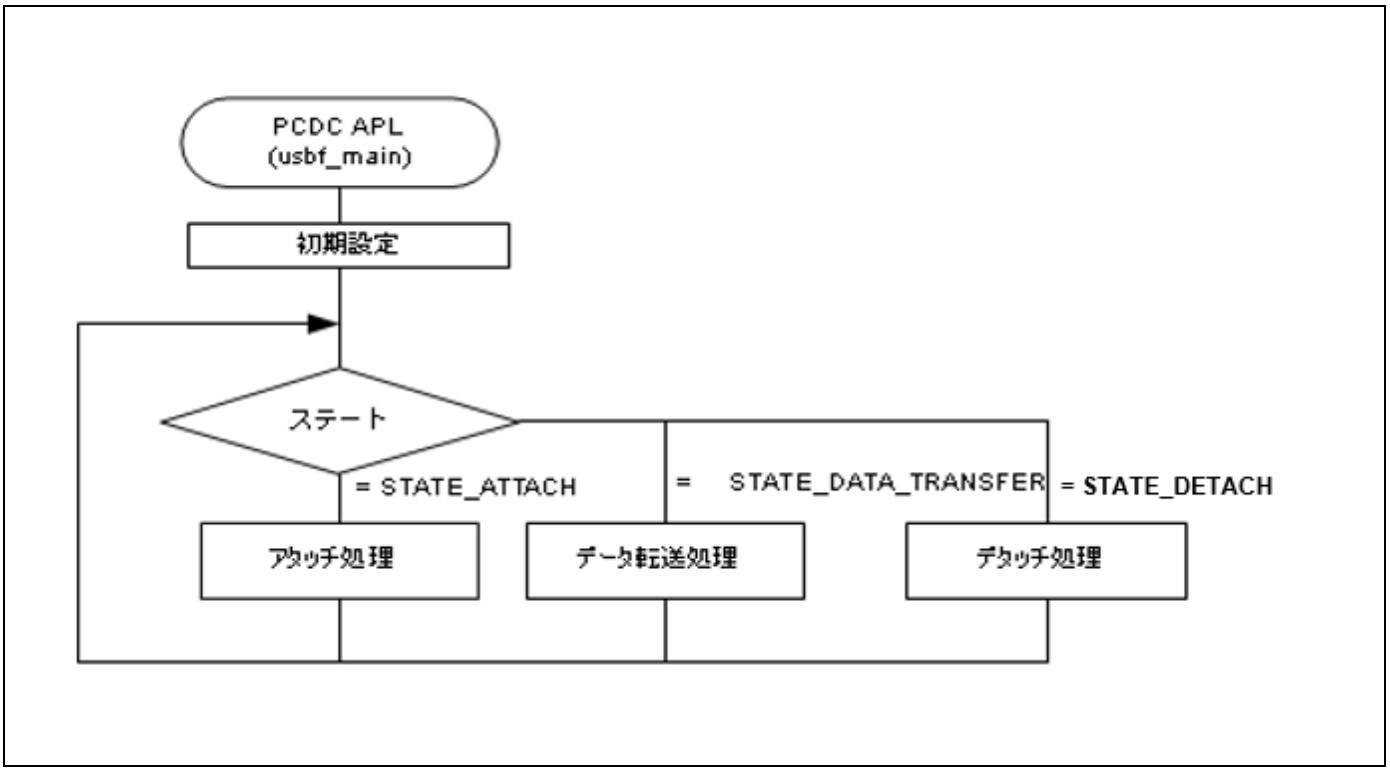


図 7-1 サンプルコードのメイン処理

表 7-7 ステート一覧

ステート	ステート処理概要	関連イベント
STATE_ATTACH	アタッチ処理	EVENT_CONFIGURD
STATE_DATA_TRANSFER	データ転送処理	EVENT_USB_READ_START
		EVENT_USB_READ_COMPLETE
		EVENT_USB_WRITE_START
		EVENT_USB_WRITE_COMPLETE
		EVENT_COM_NOTIFY_START
		EVENT_COM_NOTIFY_COMPLETE
STATE_DETACH	デタッチ処理	—

表 7-8 イベント一覧

イベント	概要
EVENT_CONFIGURD	USB デバイス接続完了
EVENT_USB_READ_START	USB データ受信要求
EVENT_USB_READ_COMPLETE	USB データ受信完了
EVENT_USB_WRITE_START	USB データ送信要求
EVENT_USB_WRITE_COMPLETE	USB データ送信完了
EVENT_COM_NOTIFY_START	Class Notification“SerialState” 送信要求
EVENT_COM_NOTIFY_COMPLETE	Class notification”SerialState” 送信完了
EVENT_NONE	イベントなし

7.6.2　ステートとイベントの管理

ステートとイベントは、以下の構造体によって管理されています。この構造体は、APL が用意している構造体です。

イベント取得処理で、取り出すイベントが存在しない場合、”EVENT_NONE”がセットされます。

```
typedef struct DEV_INFO                 /* Structure for CDC device control */
{
        uint16_t state;                 /* State for application */
        uint16_t event_cnt;             /* Event count */
        uint16_t event[EVENT_MAX];      /* Event. */
}
DEV_INFO_t;
```

次ページよりステートごとの処理概要を示します。

(1)　アタッチ処理（STATE_ATTACH）

このステートでは、USB ホストに接続され、Enumeration が完了したことを APL に通知する処理を行い、ステートを STATE_DATA_TRANSFER に変更します。

① APL では、はじめに初期化関数がステートを STATE_ATTACH にセットし、イベントを EVENT_NONE にセットします。
② USB ホストに接続するまで STATE_ATTACH 状態が継続され、cdc_connect_wait()がコールされます。USB ホストに接続した後、USB ホストとの Enumeration が完了するとコールバック関数 cdc_configured()が USB ドライバよりコールされ、このコールバック関数がイベント EVENT_CONFIGURD を発行します。なお、 このコールバック関数は、USB_PCDREG_t 構造体のメンバ devconfig に設定した関数です。
③ EVENT_CONFIGURD では、ステートを STATE_DATA_TRANSFER に変更し、イベント EVENT_USB_READ_START を発行します。

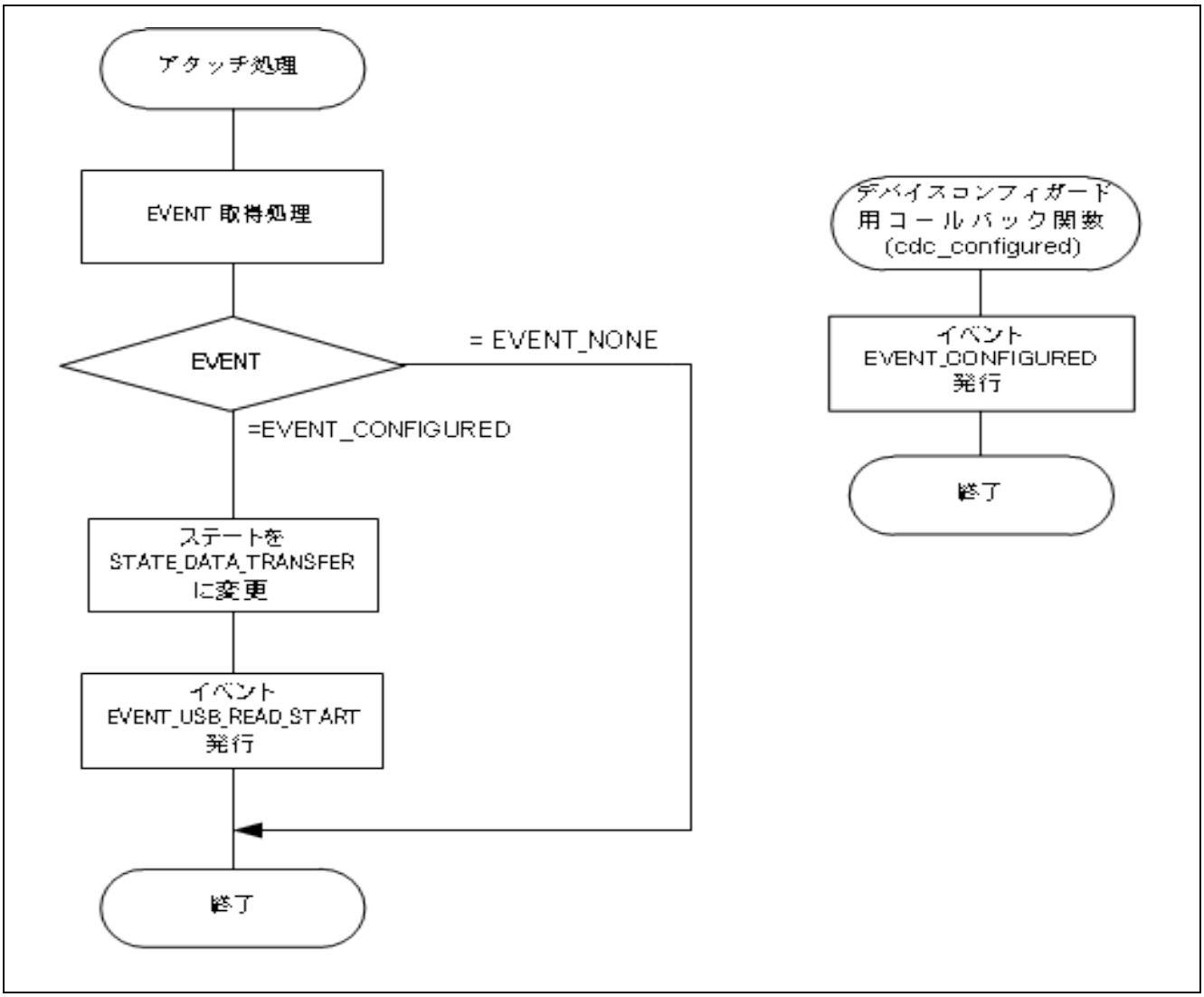


図 7-2 サンプルコードのアタッチ処理

(2)　データ転送処理（STATE_DATA_TRANSFER）

このステートでは、USB ホストから最初のデータを受信するまでの間、定期間隔で USB ホストに初期接続メッセージを送信します。最初のデータ受信後は、USB ホストからのデータを受信し、その受信したデータをそのまま USB ホストに送信するループバック処理を行います。

初期接続メッセージ

① USB ホストに接続すると USB ホストに対し以下のメッセージが一定周期で送信されます。
PCDC.Virtual serial COM port. Type characters into terminal.
The target will receive the characters over USB CDC, then copy them to a USB transmit buffer to be echoed back over USB."

② PC 上の Terminal ソフトからデータが転送されると”Echo Mode.”が Terminal ソフト上に表示されます。本表示以降は、ループバック処理が行われます。

データ受信処理

① EVENT_USB_READ_START では、USB ホストから送信されるデータを受信するため USB ドライバに対しデータ受信要求を行います。

② データ受信処理が完了するとコールバック関数 cdc_read_complete()がコールされます。このコールバック関数では、EVENT_USB_READ_COMPLETE を発行します。

③ EVENT_USB_READ_COMLETE では、イベント EVENT_USB_WRITE_START を発行します。

データ送信処理

① EVENT_USB_WRITE_START では、上記受信したデータを USB ホストに送信するため、USB ドライバに対しデータ送信要求を行います。

② データ送信転送処理が完了するとコールバック関数 cdc_write_complete()がコールされます。このコールバック関数では、EVENT_USB_WRITE_COMPLETE を発行します。

③ EVENT_USB_WRITE_COMLETE では、イベント EVENT_USB_READ_START を発行します。これにより、上記のデータ受信処理①が再度処理されます。

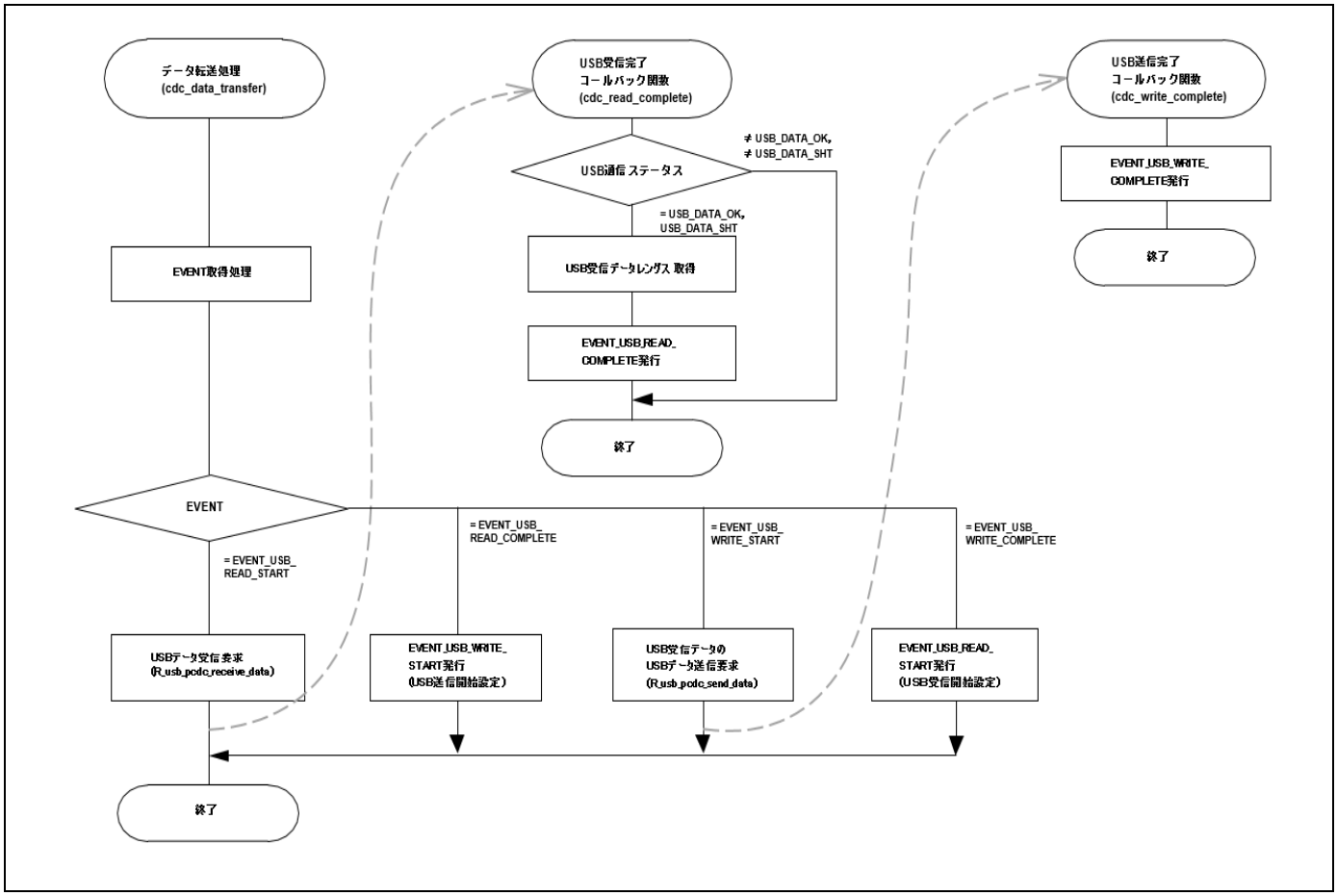


図 7-3 サンプルコードのデータ転送処理

(3)　デタッチ処理（STATE_DETACH）

接続された USB ホストから切断するとコールバック関数 cdc_detach()が USB ドライバよりコールされます。このコールバック関数では、ステートを STATE_DETACH に変更します。STATE_DETACH では、変数のクリア処理、ステートを STATE_ATTACH に変更するなどの処理を行います。

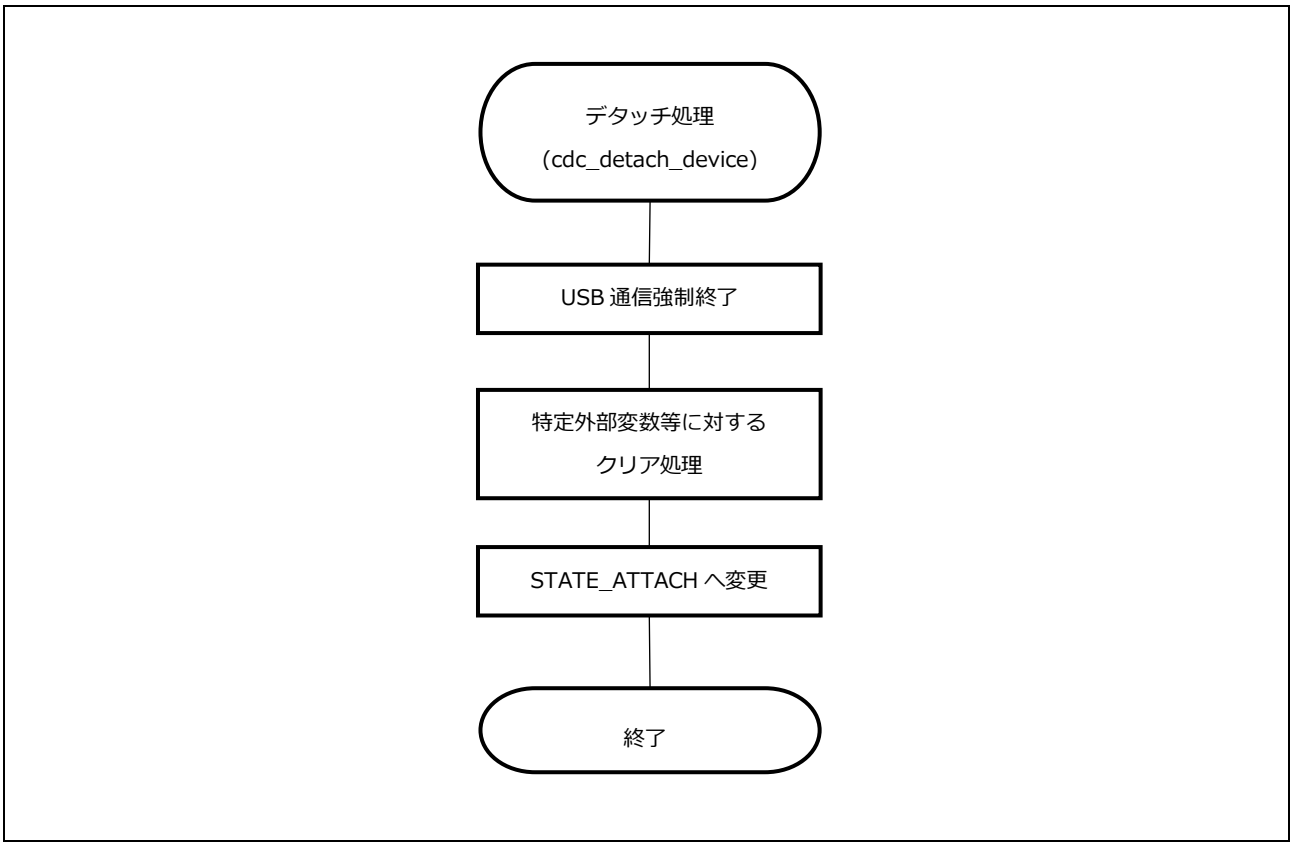


図 7-4 サンプルコードのデタッチ処理

7.7　チュートリアル

7.7.1　動作概要

サンプルアプリケーションの主な機能を以下に示します。

1. USB の初期設定をする。
2. CDC クラスリクエストに応答する。
3. USB ホストからデータを受信する。
4. その受信データを USB ホストへ送信する。

図 7-5 にシステムブロック図を示します。

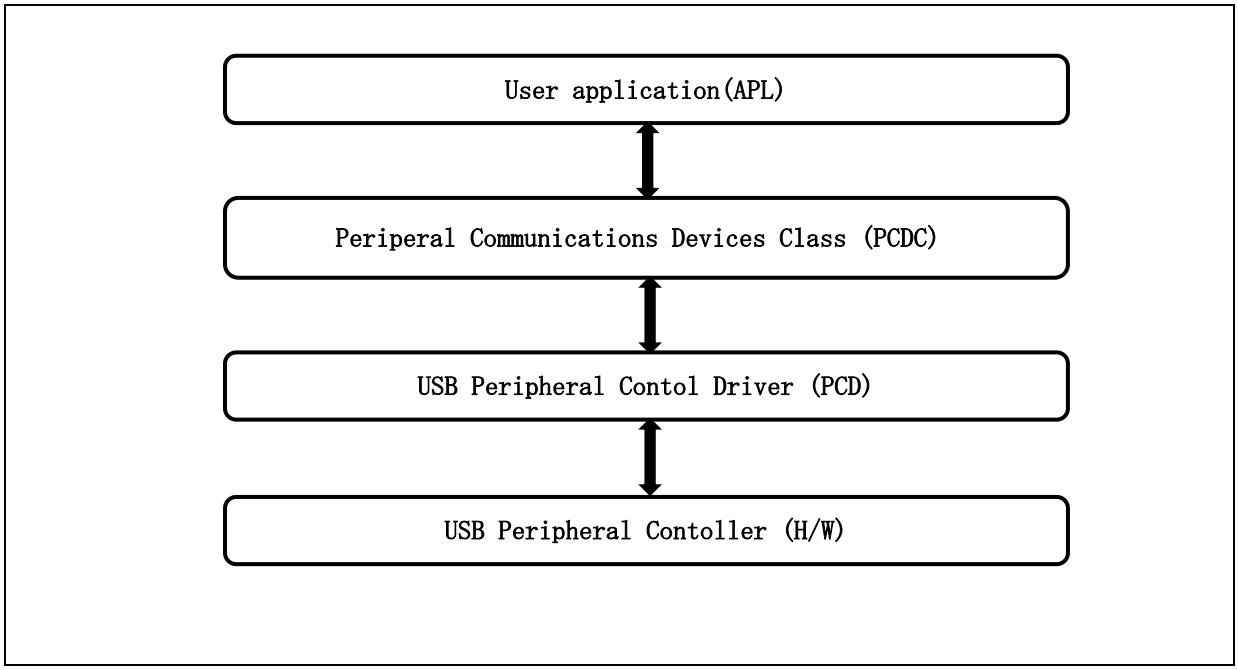


図 7-5 システムブロック図

7.7.2　使用準備

動作環境例を図 7-6 に示します。
※../Source/Driver/usbf/pcdc/ utilities 以下にあるデバイスドライバのファイルをインストールください。

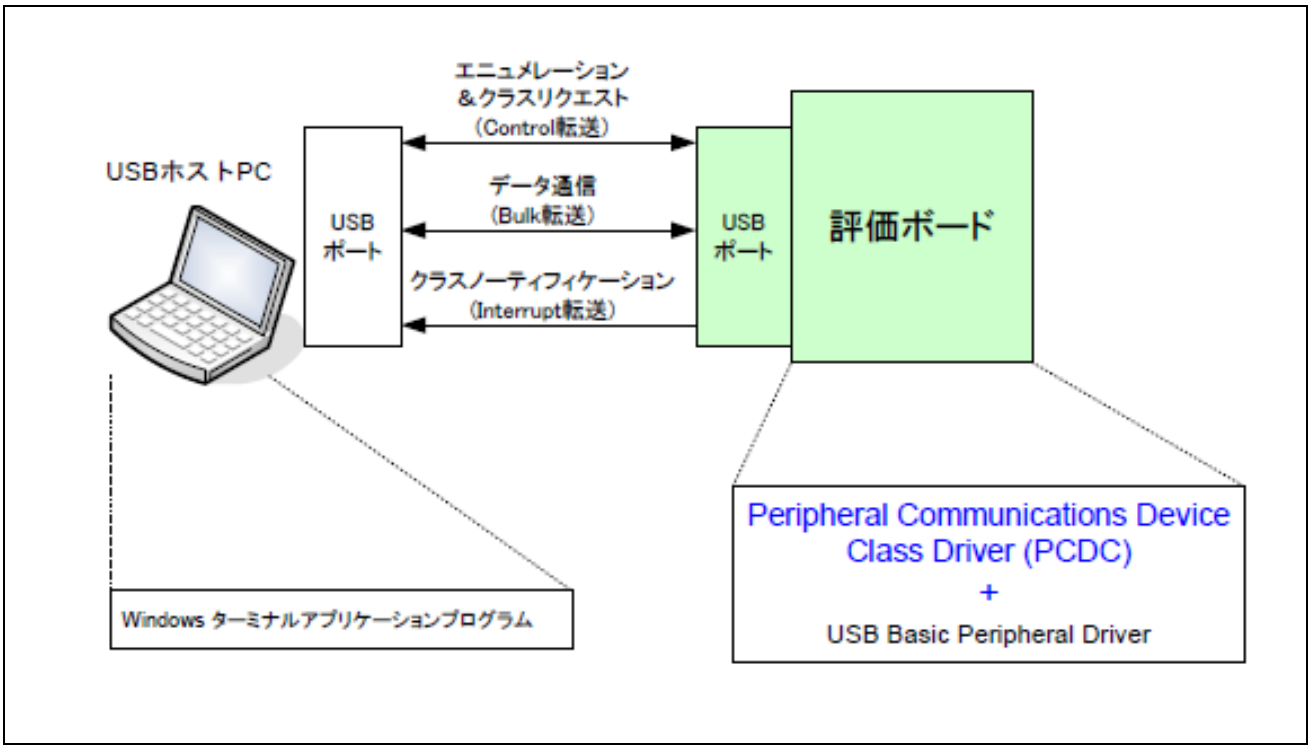


図 7-6 接続方法例

7.7.3　ターミナルソフト（Tera Term）

ホスト PC のターミナルソフトの設定は下記のとおりです。
・転送速度：115200bps
・キャラクタ長：8 ビット
・ストップビット長：1 ビット
・パリティ機能：なし
・フロー制御：ハードウェア制御

7.7.4　サンプルプログラム設定

初期設定例を以下に示します。

初期設定例

```
void usbf_main(void)
{
   /* USB ドライバの初期設定参照 */
   pcdc_registration();
   /* USB モジュールの起動参照 */
   R_USB_Open();

   apl_init();
   /* メインルーチン参照 */
   pcdc_main();
}
```

USB ドライバの初期設定

USB ドライバの設定では、PCD に対するクラスドライバの情報登録を行います。
ユーザシステムに合わせた PDCD を作成する必要があります。
作成した PDCD の各種情報を API 関数 R_usb_pstd_DriverRegistration ()を使用して USB_PCDREG_t 構造体に登録する必要があります。登録方法の詳細は、「3.10.5 ペリフェラルデバイスクラスドライバ(PDCD)登録」を参照してください。

USB モジュールの起動

R_USB_Open()を呼び出すことで、USB モジュールをハードウェアマニュアルの初期設定シーケンスに従って設定し、USB 割り込みハンドラの登録と USB 割り込み許可設定を行います。

メインルーチン

USB ドライバは初期設定後アプリケーションのメインルーチン内で割り込み処理関数（R_usb_pstd_poll）を呼びだすことで動作します。メインルーチン内で R_usb_pstd_poll()を呼び出すことで割り込みの有無を確認し、割り込みが発生していた場合割り込みに応じた処理を行います。

```
void pcdc_main(void)
{
    while(1)
    {
        R_usb_pstd_poll();

        switch( cdc_dev_info.state )
        {
            case STATE_DATA_TRANSFER:
                cdc_data_transfer();
                break;
            case STATE_ATTACH:
                cdc_connect_wait();
                break;
            case STATE_DETACH:
                cdc_detach_device();
                break;
            default:
                break;
        }
    }
}
```

7.7.5　サンプルプログラム実行例

(1)EC-1 の USB 仮想 COM ポートとホスト PC を接続し、ターミナルソフトを起動します。
(2)受信データ待ち状態となり一定周期で初期接続メッセージが送信されます。
(3) PC 上の Terminal ソフトからデータが転送されると”Echo Mode.”が Terminal ソフト上に表示されます。本表示以降は、ループバック処理が行われます。

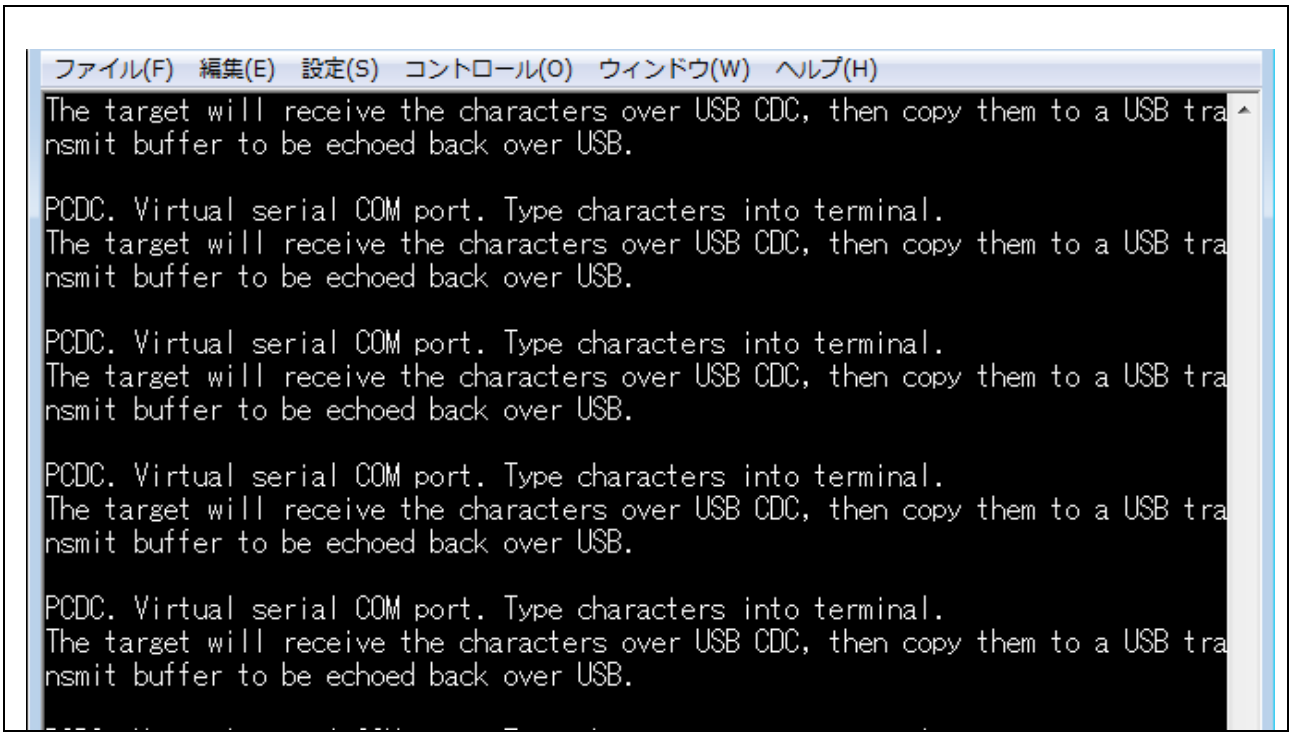


図 7-7 受信データ待ち状態

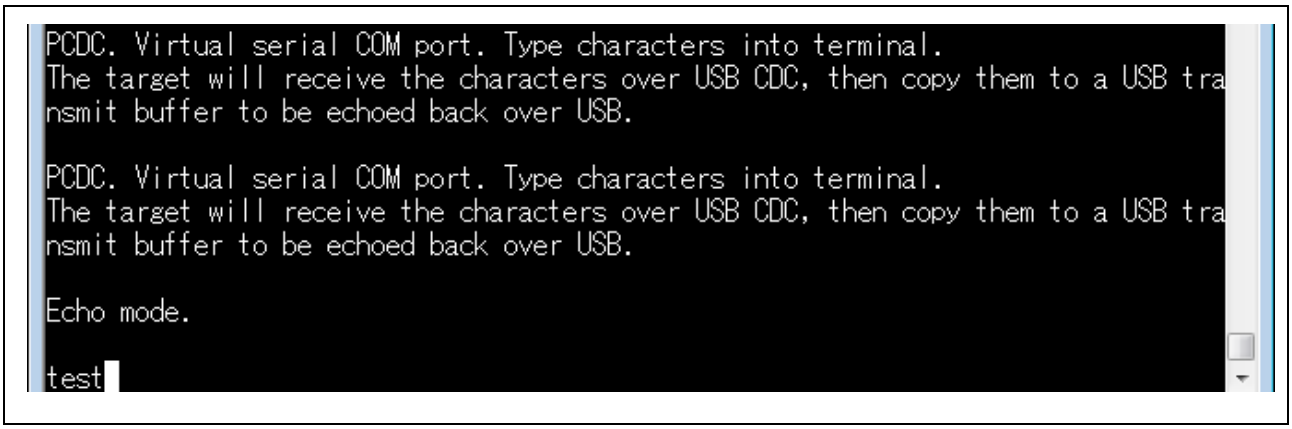


図 7-8 ループバック処理

7.8　ペリフェラルコントロールドライバ（PCD）

7.8.1　基本機能

PCD は、H/W 制御用のプログラム。PCD は USB の割り込み要因に応じた H/W 制御を行い、コールバック関数を通じて PDCD に結果を通知します。

また、PDCD から発行される要求を解析し、H/W の制御を行います。

PCD の機能を以下に示します。

1. Control 転送（ControlRead/ControlWrite/ No-data Control）
2. Data 転送（Bulk /Interrupt）および結果通知
3. データ転送の中断（全パイプ）
4. USB バスリセット信号検出およびリセットハンドシェイク結果通知
5. サスペンド/レジューム検出
6. VBUS 割り込みによるアタッチ/デタッチ検出
7. クロック停止（Low Power Mode）状態への H/W 制御および復帰

7.8.2　PCD に対する要求発行

PCD に対して H/W 制御要求を発行する場合およびデータ転送を行う場合は、API 関数を用います。PCD は上位レイヤーからの要求に対し、コールバック関数を用いて結果を通知します。

PCD はクラス/ベンダリクエストを処理する API を持ちません。

7.8.3　USB リクエスト

PCD が対応しているスタンダードリクエストを以下に示します。

GET_STATUS

GET_DESCRIPTOR

GET_CONFIGURATION

GET_INTERFACE

CLEAR_FEATURE

SET_FEATURE

SET_ADDRESS

SET_CONFIGURATION

SET_INTERFACE

PCD は上記以外のリクエストには STALL 応答します。

なお、受信した USB リクエストがデバイスクラスリクエスト又はベンダクラスリクエストの場合、それらのリクエストを判断する関数を作成し、ドライバに登録してください。

以下に、各スタンダードリクエスト受信時のコントロール転送ステージに対する処理を示します。

ControlRead : 正しいリクエストが受信できた場合、ホストに送信するデータを作成し、送信要求を行う。

ControlWrite : 正しいリクエストが受信できた場合、ホストからのデータ受信を要求する

NoDataControl : 正しいリクエストが受信できた場合、ステータス応答を行う

ControlRead-Status : ステータス応答を行う

ControlWrite-Status : ステータス応答を行う

Control 転送終了 : PDCD にホストからの要求発生を通知する

7.8.4　API 関数

PDCD は、PCD API 関数によって H/W の制御要求を行います。

一覧は「表 3-69 USB ファンクションドライバ関数一覧」を参照ください。

詳細は「3.10 USB ファンクション制御」を参照ください。

7.8.5　PCD コールバック関数

以下に PCD コールバック関数を示します。

表 7-9 R_usb_pstd_TransferStart のコールバック

呼出形式	(*USB_UTR_CB_t)(USB_UTR_t *);
引数	USB_UTR_t*　：　USB_UTR_t ポインタ
戻り値	—
説明	データ転送終了（アプリケーションから指定されたサイズのデータ転送終了およびショートパケット受信等によるトランスファー終了）で実行されます。 送受信の残りデータ長およびエラーカウントが更新されます。

表 7-10 コントロール転送のコールバック

呼出形式	(*USB_CB_TRN_t)(USB_REQUEST_t * ,uint16_t)
引数	USB_REQUEST_t *　：　リクエスト情報 uint16_t　：　ステージ情報
戻り値	—
説明	スタンダードリクエスト以外のコントロール転送受信時に実行されます。

表 7-11 USBステート変化時のコールバック

呼出形式	(*USB_CB_t)(uint16_t)
引数	uint16_t　：　※説明欄参照
戻り値	—
説明	USB ステートが遷移したとき、PCD から呼び出されます。 各ステートにてコールバック関数に設定される引数は以下になります。 ステート　引数概要 default　動作するデバイススピード USB_HSCONNECT：Hgih-Speed USB_FSCONNECT：Full-Speed USB_NOCONNECT：No connect configured　Configuration number detach　使用しません suspend　リモートウェイクアップイネーブルフラグ USB_TRUE　：許可 USB_FALSE　：禁止 resume　使用しません interface　Alternate number

7.8.6　割り込み処理

PCD は、USB 割り込みが発生すると、割り込みハンドラで要因の判定を行い、結果を USB_INT_t 構造体に格納します。PCD は、割り込みハンドラ内では H/W 制御を行いません。

H/W 制御は、ユーザがアプリケーション内で R_usb_pstd_poll()を呼び出すことで行います。

ご注意

USB_INT_t 構造体で宣言されたグローバル変数 usb_gpstd_UsbInt は、PCD が管理を行います。本変数の値をユーザが変更すると、PCD は正常に動作しなくなるため、変更しないでください。

7.9　ペリフェラルデバイスクラスドライバ（PDCD）

7.9.1　ペリフェラルデバイスクラスドライバの登録

ユーザシステムに合わせた PDCD を作成する必要があります。

作成した PDCD の各種情報を API 関数 R_usb_pstd_DriverRegistration ()を使用して USB_PCDREG_t 構造体に登録する必要があります。登録方法の詳細は、「3.10.5 ペリフェラルデバイスクラスドライバ(PDCD)登録」を参照してください。

7.9.2　ペリフェラルコントロール転送

PCD が提供する API 関数を使用したペリフェラルコントロール転送のプログラム例を示します。

例では、クラスリクエストを受信した場合のコントロール転送について示します。

コントロール転送処理に必要な関数を以下に示します。

- クラスリクエスト処理関数（ユーザのシステムに合わせて作成してください。この関数については、「7.9.3 クラスリクエスト処理関数」を参照してください。）
- データステージ用 API 関数（R_usb_pstd_ControlRead() / R_usb_pstd_ControlWrite()）
- ステータスステージ用 API 関数（R_usb_pstd_ControlEnd() / R_usb_pstd_ SetStall()）

7.9.3　クラスリクエスト処理関数

PCD はクラスリクエストを受信した時、USB_PCDREG_t 型で宣言されたグローバル変数

usb_gpstd_ReqReg にリクエストの内容を格納した後、USB_PCDREG_t 構造体のメンバ ctrltrans に登録されたクラスリクエスト処理関数を呼び出します。この関数の引数にはリクエスト情報およびコントロール転送のステージ情報が設定されています。なお、クラスリクエスト処理関数は、データステージおよびステータスステージで呼び出されます。

次ページに、クラスリクエスト処理関数例を示します。

＜クラスリクエスト処理関数例＞

```
void usb_pvendor_UsrCtrlTransFunction(USB_REQUEST_t *preq, uint16_t ctsq)
{
    if( preq->ReqTypeType == USB_CLASS )
    {
        switch( ctsq )
        {
        /* Idle or setup stage */
        case USB_CS_IDST: usb_ pvendor _ControlTrans0(preq); break;
        /* Control read data stage */
        case USB_CS_RDDS: usb_ pvendor _ControlTrans1(preq); break;
        /* Control write data stage */
        case USB_CS_WRDS: usb_ pvendor _ControlTrans2(preq); break;
        /* Control read nodata status stage */
        case USB_CS_WRND: usb_ pvendor _ControlTrans3(preq); break;
        /* Control read status stage */
        case USB_CS_RDSS: usb_ pvendor _ControlTrans4(preq); break;
        /* Control write status stage */
        case USB_CS_WRSS: usb_ pvendor _ControlTrans5(preq); break;

        /* Control sequence error */
        case USB_CS_SQER: R_usb_pstd_ControlEnd((uint16_t)USB_DATA_ERR); break;
        /* Illegal */
        default: R_usb_pstd_ControlEnd((uint16_t)USB_DATA_ERR); break;
        }
    }
    else
    {
        R_usb_pstd_SetPipeStall ((uint16_t)USB_PIPE0);
    }
}
```

1. データステージ処理

受信したリクエストをサポートしている場合は、R_usb_pstd_ControlRead()/R_usb_pstd_ControlWrite()の API を使用してホストに対し、データ転送を行なってください。

対応していないリクエストの場合は、R_usb_pstd_SetStall の API をコールし、ホストに対し STALL を返してください。

なお、上記のクラスリクエスト処理関数では、以下の関数が呼び出されます。

a). usb_ pvendor _ControlTrans1()

b). usb_ pvendor _ControlTrans2()

2. ステータスステージ処理
setup ステージまたはデータステージで問題がない場合は、引数に USB_DATA_END を指定して R_usb_pstd_ControlEnd()の API をコールしてください。
setup ステージまたはデータステージでサポートしていないパラメータ等を受信した場合は、R_usb_pstd_SetStall()の API を呼び出して、STALL を返してください。

なお、上記のクラスリクエスト処理関数では、以下の関数がコールされます。
a). usb_ pvendor _ControlTrans3()
b). usb_ pvendor _ControlTrans4()
c). usb_ pvendor _ControlTrans5()

7.10　ペリフェラルコミュニケーションデバイスクラス（PCDC）

7.10.1　基本機能

PCDC は、コミュニケーションデバイスクラス仕様 Abstract Control Model サブクラスに準拠しています。

PCDC の主な機能を以下に示します。

1. USB ホストからの機能照会に対する応答
2. USB ホストからのクラスリクエストに対する応答
3. USB ホストとのデータ通信
4. USB ホストへのシリアル通信エラー報告

7.10.2　Abstract Control Model 概要

Abstract Control Model サブクラスは、USB 機器と従来のモデム（RS-232C 接続）との間を埋める技術で、従来のモデムを使用するアプリケーションプログラムが使用可能です。

以下に PCDC がサポートするクラスリクエスト・クラスノーティフィケーションを記します。

7.10.3　クラスリクエスト（ホスト→デバイスへの通知）

PCDC がサポートするクラスリクエストを以下に示します。

表 7-12 CDCクラスリクエスト

リクエスト	コード	説明	対応
SendEncapsulatedCommand	0x00	プロトコルで定義された AT コマンド等を送信する。	×
GetEncapsulatedResponse	0x01	SendEncapsulatedCommand で送信したコマンドに対するレスポンスを要求する。	×
SetCommFeature	0x02	機器固有の 2 バイトコードや、カントリー設定の禁止／許可を設定する。	×
GetCommFeature	0x03	機器固有の 2 バイトコードや、カントリー設定の禁止／許可状態を取得する。	×
ClearCommFeature	0x04	機器固有の 2 バイトコードや、カントリー設定の禁止／許可設定をデフォルト状態に戻す。	×
SetLineCoding	0x20	通信回線設定を行う。（通信速度、データ長、パリティビット、ストップビット長）	○
GetLineCoding	0x21	通信回線設定状態を取得する。	○
SetControlLineState	0x22	通信回線制御信号 RTS、DTR の設定を行う。	○
SendBreak	0x23	ブレーク信号の送信を行う。	×

Abstract Control Model リクエストについては、USB Communications Class Subclass Specification for PSTNDevices Revision 1.2 の Table11：Requests-Abstract Control Model を参照して下さい。

7.10.4　クラスリクエストのデータフォーマット

PCDC がサポートするクラスリクエストのデータフォーマットを以下に記します。

(1). SetLineCoding

UART 回線設定を行う為にホストがデバイスに対して送信するクラスリクエストです。

SetLineCoding データフォーマットを以下に示します。

表 7-13 SetLineCodingフォーマット

bmRequestType_t	bReques	wValue	wIndex	wLength	Data
0x21	SET_LINE_CODING (0x20)	0x00	0x00	0x07	表 7-14 参照

表 7-14 Line Coding Structure フォーマット

Offset	Field	Size	Value	Description
0	DwDTERate	4	Number	データ端末の速度（bps）
4	BcharFormat	1	Number	ストップビット 0 - 1 Stop bit 1 - 1.5 Stop bit 2 - 2 Stop bit
5	BparityType	1	Number	パリティ 0 - None 1 - Odd 2 - Even
6	BdataBits	1	Number	データビット （5、6、7、8）

(2). GetLineCoding

UART 回線設定状態を要求する為にホストがデバイスに対して送信するクラスリクエストです。

GetLineCoding データフォーマットを以下に示します。

表 7-15 GetLineCoding フォーマット

bmRequestType_t	bReques	wValue	wIndex	wLength	Data
0xA1	GET_LINE_CODING (0x21)	0x00	0x00	0x07	表 7-14 参照

(3). SetControlLineState

UART のフロー制御用信号を設定する為にホストがデバイスに対して送信するクラスリクエストです。PCDC では RTS/DTR の制御をサポートしていません。

SET_CONTROL_LINE_STATE データフォーマットを以下に示します。

表 7-16 SET_CONTROL_LINE_STATE フォーマット

bmRequestType_t	bReques	wValue	wIndex	wLengt	Data
0x21	SET_CONTROL_LINE_STATE (0x22)	表 7-17 参照	0x00	0x00	None

表 7-17 Control Signal Bitmap フォーマット

Bit Position	Description
D15～D2	予約（0 にセット）
D1	DCE の送信機能を制御 0 - RTS OFF 1 - RTS ON
D0	DTE がレディ状態かの通知 0 - DTR not present 1 - DTR present

7.10.5　クラスノーティフィケーション（デバイス→ホストへの通知）

PCDC のクラスノーティフィケーション対応/非対応を以下に示します。

表 7-18 CDCクラスノーフィケーション

ノーフィケーション	コード	説明	対応
NETWORK_CONNECTION	0x00	ネットワーク接続状況を通知する	×
RESPONSE_AVAILABLE	0x01	GET_ENCAPSLATED_RESPONSE への応答	×
SERIAL_STATE	0x20	シリアル回線状態を通知する	○

(1). SerialState

UART の状態変化を検出した場合、ホストへ状態通知を行います。

PCDC ではオーバーランエラー、パリティエラー、フレーミングエラー検出をサポートしていま す。状態通知は正常状態からエラー検出した場合に行います。エラーを連続検出しても状態通知を連続送信しません。

SerialState データフォーマットを以下に示します。

表 7-19 SerialState フォーマット

bmRequestType_t	bReques	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE (0x20)	0x00	0x00	0x02	表 7-20 参照

表 7-20 UART State bitmap フォーマット

Bit Position	Field	Description
D15～D7		予約
D6	bOverRun	オーバーラン検出
D5	bParity	パリティエラー検出
D4	bFraming	フレーミングエラー検出
D3	bRingSignal	着信（Ring signal）を感知した
D2	bBreak	ブレーク信号検出
D1	bTxCarrier	Data Set Ready：回線が接続されて通信可能
D0	bRxCarrier	Data Carrier Detect：回線にキャリア検出

7.10.6　PC の仮想 COM ポートについて

Windows OS 搭載 PC は CDC デバイスを仮想 COM ポートとして利用することが可能です。

Windows OS 搭載 PC に PCDC を実装した評価ボードを接続すると、Enumeration 設定に続き、CDC クラスリクエストの GetLineCoding 及び SetControlLineState を行った後、仮想 COM デバイスとしてデバイスマネージャに登録されます。Windows デバイスマネージャに仮想 COM ポートとして登録された後は、Windows OS 標準搭載のハイパーターミナル等のターミナルアプリで評価ボードとデータ通信が可能です。

ターミナルアプリのシリアルポート設定を行うことで、クラスリクエスト SetLineCoding による通信回線設定が可能です。ターミナルアプリのウインドウから入力したデータ（又はファイル送信）は評価ボードへ転送され、評価ボードから転送したデータはターミナルアプリのウインドウに出力されます。

ターミナルアプリによっては最後に受信したデータがマックスパケットサイズの場合、継続するデータがあると判断して受信データをターミナルに表示しないことがあります。この場合、マックスパケットサイズ未満のデータを受信することで、それまでに受信したデータがターミナルに表示されます。

7.10.7　API

すべての API 呼び出しとそれをサポートするインタフェース定義は r_usb_pcdc_if.h に記載しています。

本モジュールのコンフィグレーションオプションの設定は、r_usb_pcdc_config.h で行います。

表 7-21 にオプション名および設定値に関する説明を示します。

表 7-21 PCDC コンフィグレーションオプション

定義名	デフォルト値	説明
USB_PCDC_USE_PIPE_IN	USB_PIPE1	Bulk IN 転送パイプ番号
USB_PCDC_USE_PIPE_OUT	USB_PIPE2	Bulk OUT 転送パイプ番号
USB_PCDC_USE_PIPE_STATUS	USB_PIPE6	Interrupt IN 転送パイプ番号

表 7-22 に PCDC API 一覧を示します。

表 7-22 PCDC API 一覧

関数名	機能概要
R_usb_pcdc_SendData	USB 送信処理
R_usb_pcdc_ReceiveData	USB 受信処理
R_usb_pcdc_SerialStateNotification	クラスノーティフィケーション”SerialState”
R_usb_pcdc_ctrltrans	CDC 用コントロール転送処理

7.11　パイプ情報テーブル

PDCD は、パイプ 1～9 を使用してデータ通信を行う場合、パイプ設定を定義するための情報テーブルを保持する必要があります。パイプ情報テーブルは、データ転送を行うために必要なレジスタに設定する値と FIFO ポートの転送方法（CPU 転送または DMA 転送）の設定値で構成されます。

以下に、パイプ情報テーブルの構成例を示します。

```
uint16_t usb_gvendor_smpl_eptbl[] =
{
USB_PIPE1,                                          ←パイプ定義項目 1
USB_BULK | USB_BFREOFF | USB_DBLBON |
USB_CNTMDON | USB_SHTNAKON | USB_EP1 |
USB_DIR_P_IN,                                       ←パイプ定義項目 2
USB_BUF_SIZE(1024) | USB_BUF_NUMB(8),               ←パイプ定義項目 3
512,                                                ←パイプ定義項目 4
USB_IFISOFF | USB_IITV_TIME(0u),                    ←パイプ定義項目 5
USB_CUSE,                                           ←パイプ定義項目 6
 :
 :
USB_PDTBEND,
}
```

パイプ情報テーブルは、以下 6 項目（uint16_t×6）で構成されます。

1.パイプウインドウ選択レジスタ（0x64 番地）

2.パイプコンフィグレーションレジスタ（0x68 番地）

3.パイプバッファ指定レジスタ（0x6A 番地）

4.パイプマックスパケットサイズレジスタ（0x6C 番地）

5.パイプ周期制御レジスタ（0x6E 番地）

6.FIFO ポート使用方法

(1).　パイプ定義項目 1

パイプウインドウ選択レジスタに設定する値を指定します。

パイプ選択：使用するパイプ（USB_PIPE1～USB_PIPE9）を指定してください

制限事項

―

(2). パイプ定義項目 2

パイプコンフィグレーションレジスタに設定する値を指定します。

転送タイプ　：　USB_BULK/USB_INT/USB_ISO のいずれかを指定してください

BRDY 割り込み動作指定　：　USB_BFREOFF を指定してください

ダブルバッファモード　：　USB_DBLBON/USB_DBLBOFF のどちらかを指定してください

連続転送モード　：　USB_CNTMDON/USB_CNTMDOFF のどちらかを指定してください

SHTNAK 動作指定　：　USB_SHTNAKON/USB_SHTNAKOFF のどちらかを指定してください

転送方向　：　USB_DIR_P_OUT、USB_DIR_P_IN のどちらかを指定してください

エンドポイント番号　：　パイプに対するエンドポイント番号(EP1～EP15)を指定してください

制限事項

1.転送タイプは選択したパイプにより、設定可能な値が異なります。詳細は、ユーザーズマニュアルを参照してください。

2.転送方法を受信（USB_DIR_P_OUT）に設定した場合、SHTNAK 動作指定は USB_SHTNAKON を設定してください。

(3). パイプ定義項目 3

パイプバッファ指定レジスタに設定する値を指定します。

バッファサイズ　：　パイプのバッファサイズを 64 バイト単位で指定してください

バッファ番号　：　パイプのバッファの先頭ブロック番号を指定してください

制限事項

・ 使用するバッファ領域が重複しないように設定してください。

・ パイプ定義項目 2 で、ダブルバッファモードを USB_DBLBON に設定した場合、設定した 2 倍 のバッファ領域が必要になります。

(4). パイプ定義項目 4

パイプマックスパケットサイズレジスタに設定する値を指定します。

マックスパケットサイズ　：　パイプのマックスパケットサイズを指定してください

制限事項

－

(5). パイプ定義項目 5

パイプ周期制御レジスタに設定する値を指定します。

ISO IN バッファフラッシュ　：　USB_IFISOFF を指定してください

インターバル間隔　：　転送インターバルタイミングの値（0-7）を指定してください

制限事項

・ 転送タイプを USB_ISO 以外に設定した場合、ISO IN バッファフラッシュは USB_IFISOFF を 設定してください。

・ USB_PIPE3～USB_PIPE5 を選択した場合、インターバル間隔は値 0 を設定してください

(6). パイプ定義項目 6

パイプが使用する FIFO ポートとアクセス方法を指定します。設定可能な値を以下に示します。

USB_CUSE ： CFIFO に対し、CPU アクセスする

USB_D0DMA ： D0FIFO に対し、DMA アクセスする（サイクルスチルモード）

USB_D0DMA_C ： D0FIFOBn に対し、DMA アクセスする（32byte 連続アクセスモード）

USB_D1DMA_C ： D1FIFOBn に対し、DMA アクセスする（32byte 連続アクセスモード）

制限事項

・ 受信方向のパイプはトランザクションカウンタを使用します。

・ USB_D1DMA に対応したサンプル関数はありません。

・ DMA 転送を行う場合、異なるパイプに同じ FIFO ポートへのアクセス方法を設定することはできません。

（例）USB_PIPE1 に USB_D0DMA_C を設定している状態で USB_PIPE2 に USB_D0DMA_C 又は USB_D0DMA を設定することができません

(7). その他制限事項

・ デバイスクラスでトランスファー単位の通信同期を取ってください。

・ パイプ情報テーブルの最後に必ず USB_PDTBLEND を書いてください。

7.12　ユーザ定義情報ファイル

PCD はユーザ定義情報ファイル（r_usb_basic_config.h）を書き換えることにより、
PCD の機能設定を行います。システムに合わせて、下記項目を変更してください。

(1). DMA 転送設定
DMA 転送使用の有無を設定します。
#define USB_DMA_PP USB_DMA_USE_PP ：DMA 転送使用
#define USB_DMA_PP USB_DMA_NOT_USE_PP ：DMA 転送未使用

[初期設定]
#define USB_DMA_PP USB_DMA_USE_PP

(2). Low Power Mode 設定
Low Power Mode の使用、未使用を設定します。
#define USB_CPU_LPW_PP USB_LPWR_USE_PP ：Low Power Mode を使用する
#define USB_CPU_LPW_PP USB_LPWR_NOT_USE_PP ：Low Power Mode を使用しない

[初期設定]
#define USB_CPU_LPW_PP USB_LPWR_USE_PP

(3). デバック情報出力関数設定
デバック情報表示の有効/無効を設定します。
#define USB_DEBUG_OUTPUT_PP USB_DEBUG_ON_PP ：デバック情報表示有効
#define USB_DEBUG_OUTPUT_PP USB_DEBUG_OFF_PP ：デバック情報表示無効

[初期設定]
#define USB_DEBUG_OUTPUT_PP USB_DEBUG_OFF_PP

[注意]
UART および表示デバイスなどにデバッグ情報を出力するマクロです。シリアルドライバ、表示デバイス用ドライバが必要になります。

7.13　データ転送

7.13.1　データ転送要求

データ転送要求は、API 関数の R_usb_pstd_TransferStart()の引数に転送情報を設定した USB_UTR_t 構造体を渡すことで行うことができます。
USB_UTR_t 構造体の詳細は、「表 3-21 USB_UTR_t 構造体」を参照してください。

7.13.2　転送結果の通知

PCD は、データ転送が終了すると PDCD に対してデータ転送要求時に登録されたコールバック関数でデータ転送の終了とシーケンストグルビット情報を通知します。転送結果は、USB_UTR_t 構造体のメンバ（status）に格納されます。
以下に、通信結果を示します。
USB_DATA_NONE : データ送信が正常終了した場合
USB_DATA_OK : データ受信が正常終了した場合
USB_DATA_SHT : データ受信は正常終了したが指定されたデータ長未満で終了した場合
USB_DATA_OVR : 受信データがサイズオーバーした場合
USB_DATA_ERR : 無応答もしくはオーバ／アンダーランエラーを検出した場合
USB_DATA_STALL : STALL 応答もしくは MaxPacketSize エラーを検出した場合
USB_DATA_STOP : データ転送を強制終了した場合

7.13.3　データ受信時の注意事項

・ショートパケットを受信した場合、残り受信予定のデータ長を USB_UTR_t 構造体の tranlen に格納し、トランスファーを終了します。受信したデータがユーザバッファより大きい場合、ユーザバッファまでのデータを FIFO バッファから読み出し、トランスファーを終了します。ユーザバッファ領域がトランスファーサイズの容量分を確保できない場合は、他のデータ領域をクリアしてしまうことがあります。
・DMA 使用時は、受信したデータを格納するバッファは転送対象 PIPE の MAX パケットサイズの整数倍の領域が必要です。MAX パケットサイズが 64 バイトの場合、受信データサイズに対し必要なバッファサイズの例を以下に示します。

表 7-23 Buffer size example (max packet size is 64bytes)

Receive Data Size	Buffer Size [bytes]
64 バイト以下	64　(MAX パケットサイズ ×1)
65 バイト以上、128 バイト以下	128　(MAX パケットサイズ ×2)
・・・	・・・
449 バイト以上、512 バイト以下	512　(MAX パケットサイズ ×8)
・・・	・・・

7.13.4　データ転送例

以下に、データ転送例を示します。
① APL または PDCD で USB_UTR_t 構造体の以下のメンバに転送ステータスを設定する
　keyword　:　パイプ番号
　tranadr　:　データバッファ
　tranlen　:　転送サイズ
　complete　:　データ転送終了時に実行されるコールバック関数

② R_usb_pstd_TransferStart()を呼び出し、データ転送要求を行う。
③ データ転送終了後、①で complete に設定したコールバック関数が呼び出され、転送結果の通知が行われる。（「7.13.2 転送結果の通知」を参照してください）

7.14　DMA 転送

7.14.1　基本仕様

PCD は、DMA コントローラを使用することでユーザバッファと FIFO バッファ間のデータ転送を DMA 転送で行うことが可能です。
DMA 転送は 32 バイト連続アクセスモードとサイクルスチルモードのどちらかを選択することが可能です。
各アクセスモードの設定方法は、「7.11 パイプ情報テーブル」を参照してください。
表 7-24、表 7-25 に、各アクセスモードの概要を示します。

表 7-24 32バイト連続アクセスモード

USB モジュール設定		
使用FIFO ポート	D0FIFOBn ポート/D0FIFO ポート	D1FIFOBn ポート/D1FIFO ポート
アクセスビット幅	送信：32bit/8bit 受信：32bit	送信：32bit/8bit 受信：32bit
割込み	送信：D0FIFO 割り込み/BEMP 割り込み 受信：BRDY 割り込み	送信：D1FIFO 割り込み/BEMP 割込 受信：BRDY 割り込み
データ転送サイズ	送信：FIFO サイズ×N＋端数 受信：(FIFO サイズ/32）×32	送信：FIFO サイズ×N＋端数 受信：(FIFO サイズ/32）×32
DMA モジュール設定		
チャネル	DMAC0_0	DMAC0_1
転送データ単位	送信：256bit/8bit 受信：256bit	送信：256bit/8bit 受信：256bit
転送モード	シングル転送	シングル転送
DMA モード	レジスタモード	レジスタモード
インターバル機能	未使用	未使用
スキップ機能	未使用	未使用
サスペンド機能	未使用	未使用
バッファ読み出し機能	未使用	未使用
割込み	送信：転送完了割り込み 受信：ー	送信：転送完了割り込み 受信：ー

表 7-25 サイクルスチルモード

USB モジュール設定		
使用FIFO ポート	D0FIFO ポート	ー
アクセスビット幅	送信：32bit/8bit 受信：32bit	ー
割込み	送信：D0FIFO 割り込み/BEMP 割り込み 受信：BRDY 割り込み	ー
データ転送サイズ	送信：FIFO サイズ×N＋端数 受信：(FIFO サイズ/4）×4	ー
DMA モジュール設定		
チャネル	DMAC0_0	ー
転送データ単位	送信：32bit/8bit 受信：32bit	ー
転送モード	シングル転送	ー
DMA モード	レジスタモード	ー
インターバル機能	未使用	ー
スキップ機能	未使用	ー
サスペンド機能	未使用	ー
バッファ読み出し機	未使用	ー
割込み	送信：転送完了割り込み 受信：ー	ー

8. WDTA サンプルソフト

8.1 概要

本章では、評価ボード上に実装される WDT 0 チャネル(WDT0)を制御する WDTA ドライバを使用したサンプルプログラムについて説明します。

WDTA サンプルプログラムの特長を以下に示します。

・ウォッチドッグタイマ（WDTA）動作後、コンペアマッチタイマ（CMT）による定周期（223.7ms）でリフレッシュ動作させます。
・外部割り込み（IRQ2）によるソフトウェアウエイトの発生でリフレッシュ動作が停止し、リセットが発生します。リセット発生後に LED2 が点灯します。

対象デバイス

EC-1

本サンプルを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

8.2 定数一覧

表 8-1 にサンプルコードで使用する定数を示します。

表 8-1 サンプルコードで使用されている定数

定数名	設定値	内容
CMT0_CLOCK_PCLKD_512	3	Count clock = PCLKD/512
CMT0_CMI0_ENABLE	1	Enable CMI0 interrupt
CMT0_INTERVAL_TIME	0x7FFF	Set interval time to 223.7ms
CPG_CMT0_START	1	Start CMT0 count

8.3　関数一覧

「表 8-2　関数一覧」を以下に示します。

表 8-2　関数一覧

関数名	概要	スコープ	定義ファイル
main	メイン処理	local	main.c
port_init	ポート設定初期化	local	main.c
ecm_init	ECM 設定初期化	local	main.c
icu_init	割り込み設定	local	main.c
cmt_init	CMT モジュール初期化	local	main.c
wdt0_init	WDT0 初期化処理	local	main.c
soft_wait	ソフトウェアウェイト	local	main.c
R_IRQ2_isr	IRQ2 割り込み（IRQ 端子割り込み 2）処理	local	main.c
R_IRQ21_isr	IRQ21 割り込み（コンペアマッチタイマ（CMI0））処理	local	main.c

8.4　関数詳細

8.4.1　main

(1)　概要

メイン処理

(2)　C 言語形式

void main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。

- ポート設定初期化
- ECM 初期化
- CMT 初期化
- ICU 初期化
- WDT 初期化

上記の初期設定を行い、CMT0 の動作を開始させます。その後、メインループにて LED1 の点灯/消灯を繰り返します。

(5)　戻り値

なし

8.4.2 port_init

(1) 概要

ポート設定初期化

(2) C 言語形式

void port_init (void);

(3) パラメータ

なし

(4) 機能

ポート設定の初期化を行います。

(5) 戻り値

なし

8.4.3 ecm_init

(1) 概要

ECM 設定初期化

(2) C 言語形式

void ecm_init (void);

(3) パラメータ

なし

(4) 機能

ECM 設定の初期化を行います。

(5) 戻り値

なし

8.4.4 icu_init

(1) 概要

割り込み設定

(2) C 言語形式

void icu_init (void);

(3) パラメータ

なし

(4) 機能

割り込み処理を有効にします。

(5) 戻り値

なし

8.4.5 cmt_init

(1) 概要

CMT モジュール初期化

(2) C 言語形式

void cmt_init (void);

(3) パラメータ

なし

(4) 機能

CMT チャネル 0 を初期化します。

(5) 戻り値

なし

8.4.6 wdt0_init

(1) 概要

WDT0 初期化処理

(2) C 言語形式

void wdt0_init (void);

(3) パラメータ

なし

(4) 機能

WDT0 を初期化し、WDT0 のカウント動作を開始させます。

(5) 戻り値

なし

8.4.7 soft_wait

(1) 概要

ソフトウェアウェイト処理

(2) C 言語形式

void soft_wait(void);

(3) パラメータ

なし

(4) 機能

NOP を使用したソフトウェアウェイト処理を行います。

(5) 戻り値

なし

8.4.8　R_IRQ2_isr

(1)　概要

IRQ2 割り込み（IRQ 端子割り込み 2）処理

(2)　C 言語形式

void R_IRQ2_isr (void);

(3)　パラメータ

なし

(4)　機能

ソフトウェアウエイト処理（約 3 秒）を実行します。この間にウォッチドッグタイマをアンダーフローさせ、ECM リセットを発生させます。
リセット解除後のリセット判定にて、LED2 を点灯させます。

(5)　戻り値

なし

8.4.9 R_IRQ21_isr

(1) 概要

IRQ21 割り込み（コンペアマッチタイマ（CMI0））処理

(2) C 言語形式

void R_IRQ21_isr (void);

(3) パラメータ

なし

(4) 機能

ウォッチドッグタイマのリフレッシュ動作を行います。

(5) 戻り値

なし

8.5　フローチャート

8.5.1　メイン処理

サンプルコードのメイン処理のフローチャートを示します。

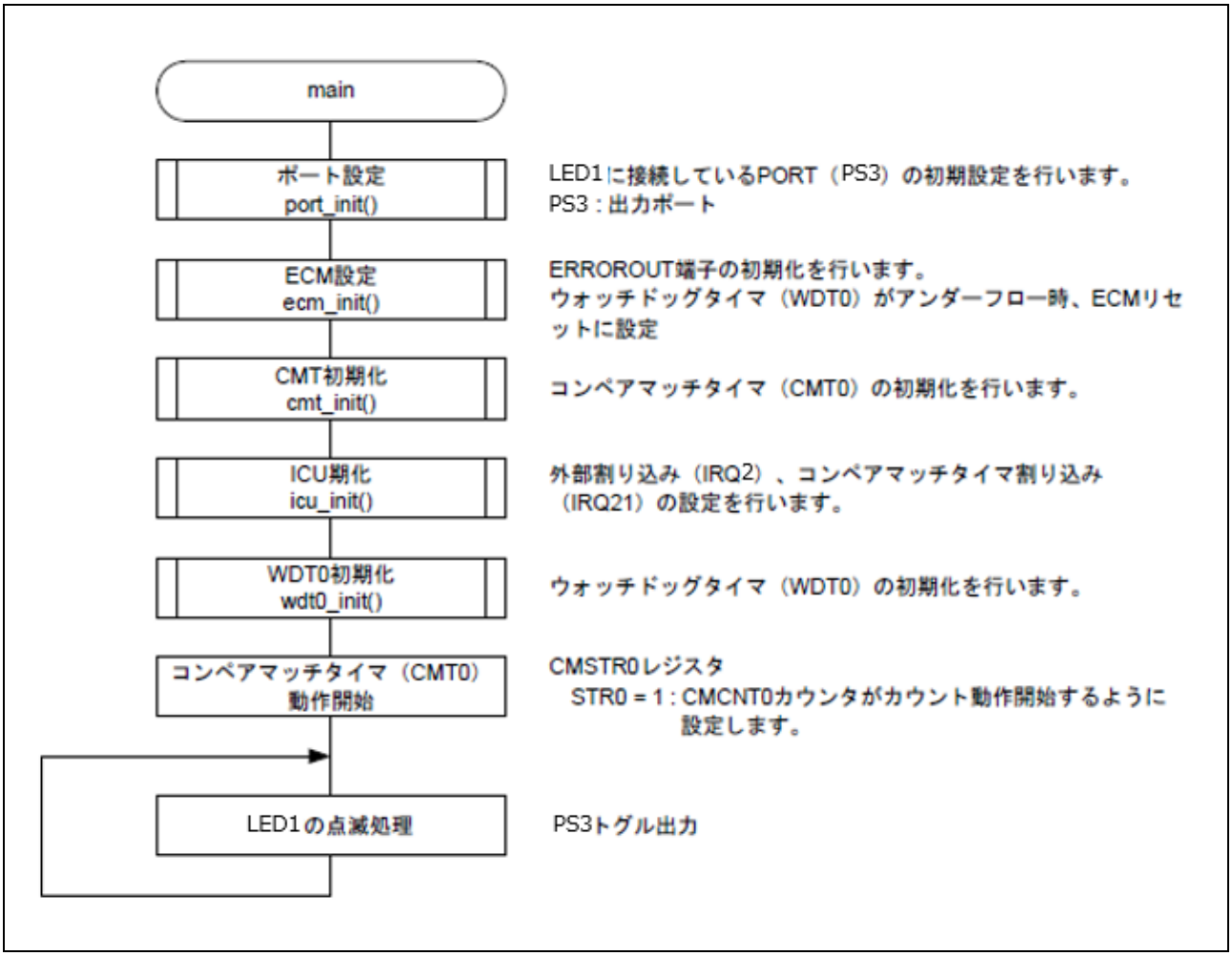


図 8-1 サンプルコードのメイン処理

8.5.2　WDT0 初期化処理

WDT0 初期化処理のフローチャートを示します。

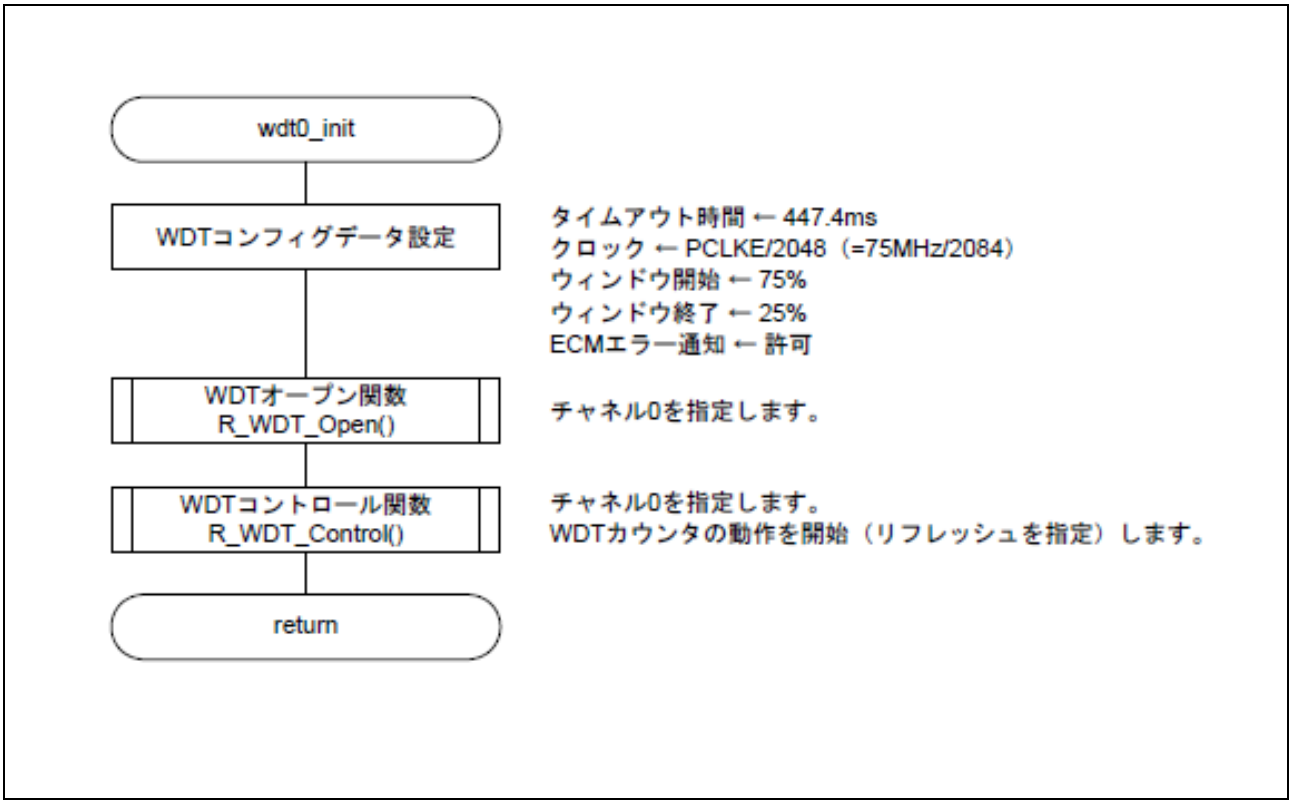


図 8-2 wdt0初期化処理

8.5.3　WDT オープン関数

WDT オープン関数のフローチャートを示します。

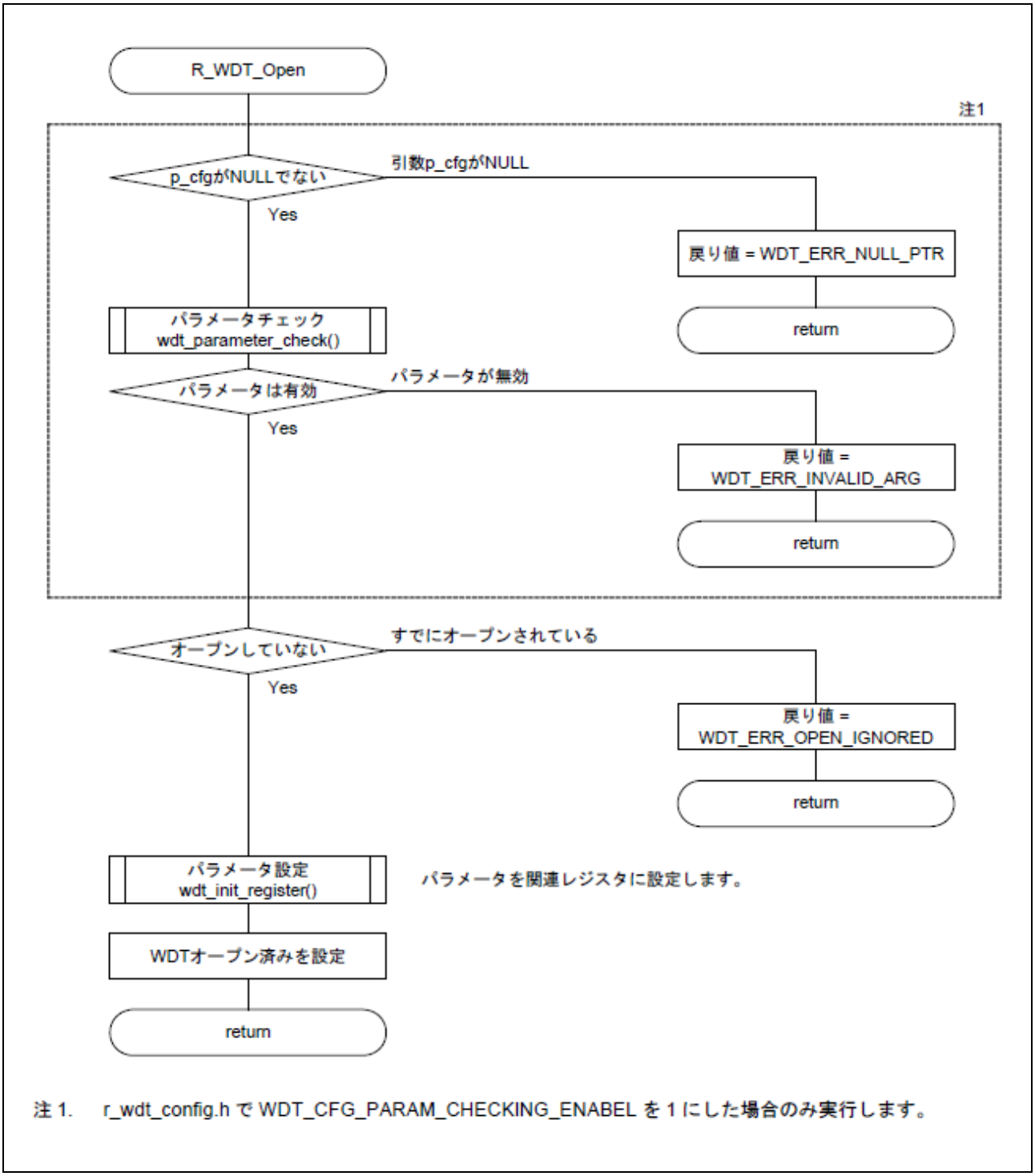
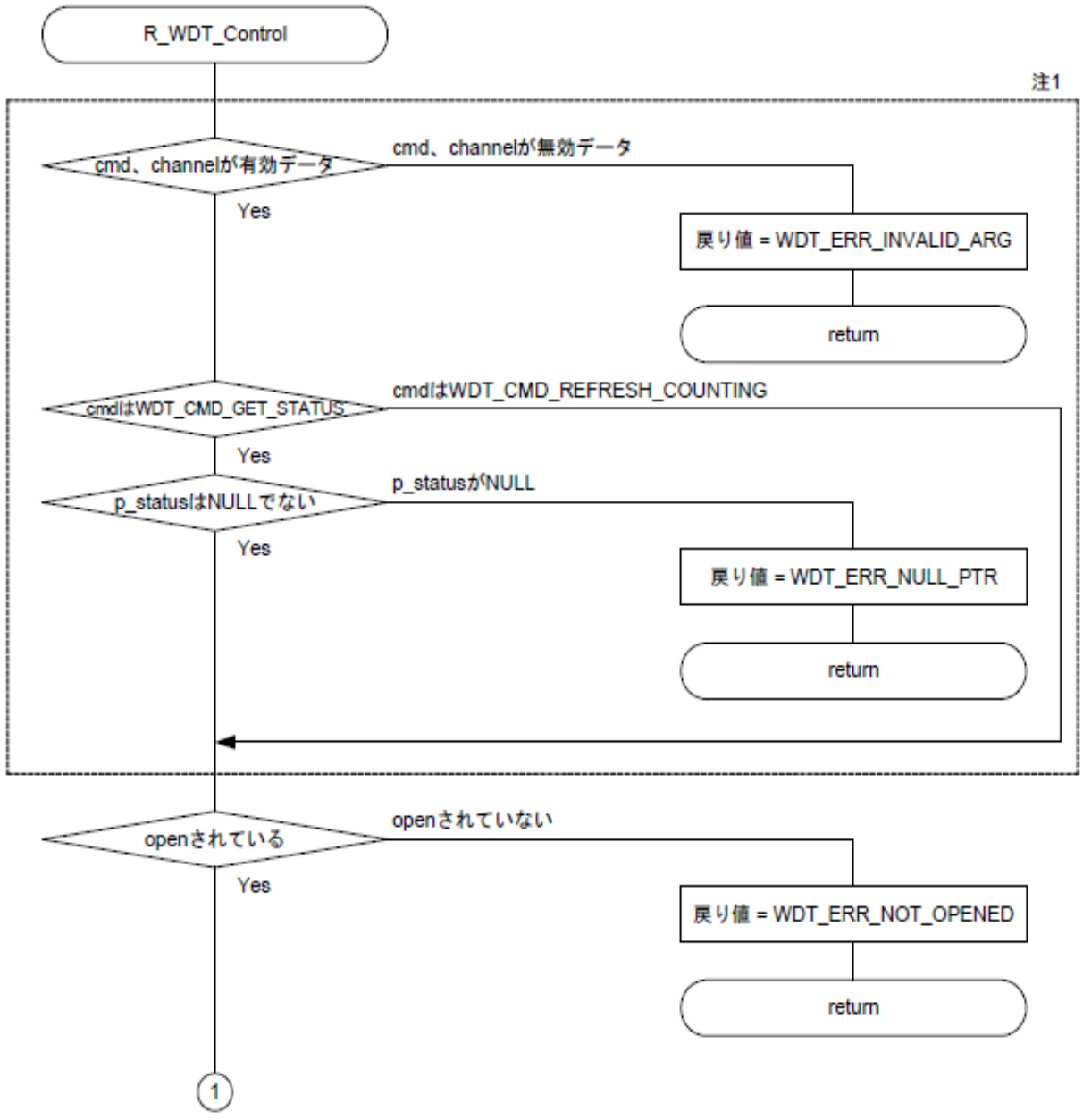


図 8-3 WDTオープン処理

8.5.4　WDT コントロール関数

WDT コントロール関数のフローチャートを示します。



注 1.　r_wdt_config.h で WDT_CFG_PARAM_CHECKING_ENABEL を 1 にした場合のみ実行します。

図 8-4 WDTコントロール処理

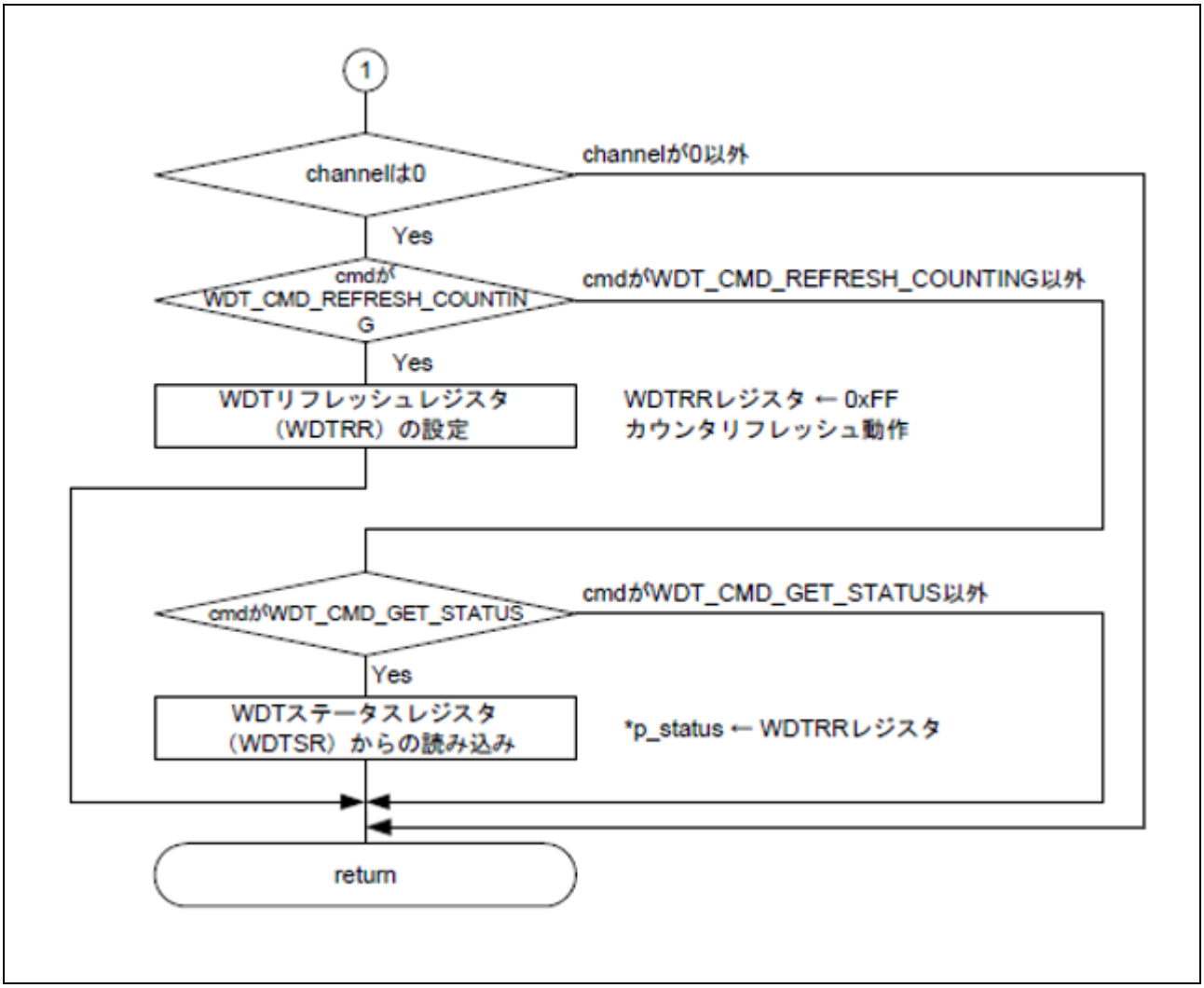


図 8-5 WDTコントロール処理

8.5.5　IRQ2 割り込み（IRQ 端子割り込み 2）処理

IRQ2 割り込み（IRQ 端子割り込み 2）処理のフローチャートを示します。

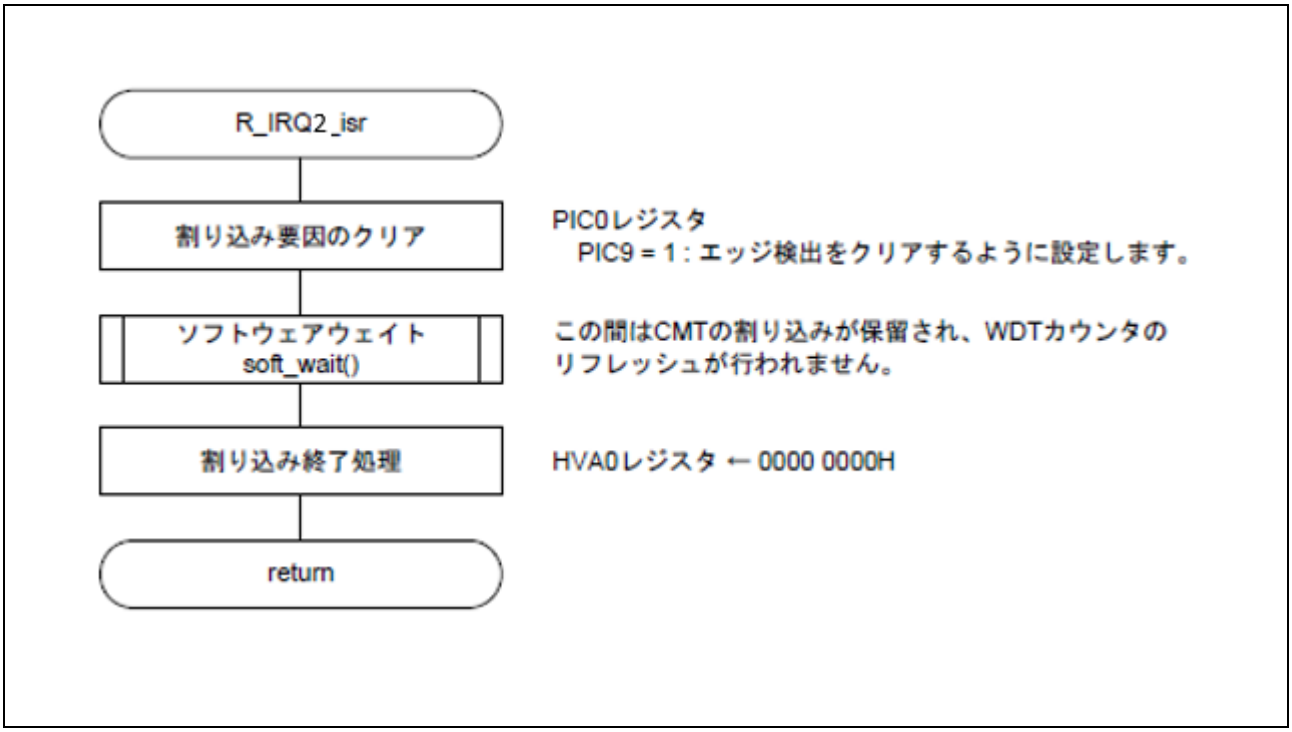


図 8-6 IRQ2 割り込み（IRQ 端子割り込み2）処理

8.5.6　IRQ21 割り込み（コンペアマッチタイマ 0 割り込み）処理

IRQ21 割り込み（コンペアマッチタイマ ch0 割り込み）処理のフローチャートを示します。

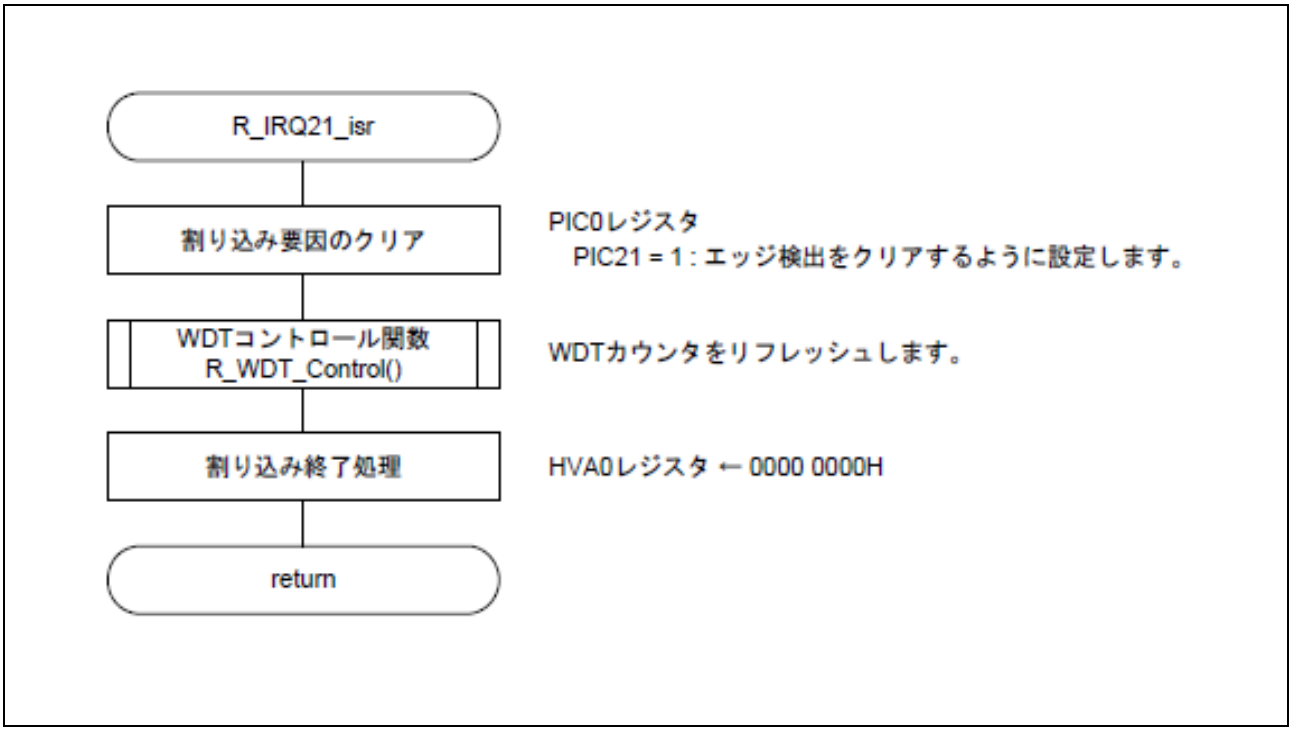


図 8-7 IRQ21 割り込み（コンペアマッチタイマch0 割り込み）処理

8.6　チュートリアル

8.6.1　動作概要

本サンプルプログラムでは、ウォッチドッグタイマ（WDTA）の初期設定を行い、以降、コンペアマッチタイマのインターバル割り込みで定期的（227.3ms）にリフレッシュ動作を行います。

Dip スイッチ(SW2.3) を切り替えると外部端子割り込み 2 が発生し、割り込み内でソフトウェアウエイト処理が実施されます。この間は CMT のインターバル割り込みが保留されるため、ウォッチドッグタイマ（WDTA）のリフレッシュ動作が停止します。そのため、ウォッチドッグタイマのカウンタ値がアンダーフローし、エラーコントロールモジュール（ECM）にエラー通知を行い、ECM リセットが発生します。そしてリセット解除後のリセット判定処理にて LED2 が点灯します。

本サンプルプログラムの機能概要を表 8-3 動作概要に示します。また、図 8-8 にタイミング図を示します。

表 8-3 動作概要

機能	概要
通信チャネル	チャネル0（WDT0）を使用
クロック	PCLKE/2048（=75MHz/2048）
タイムアウト	16384cycle（=447.4ms）
ウィンドウ	開始 : 75% 終了 : 25%
ECM エラー通知	許可

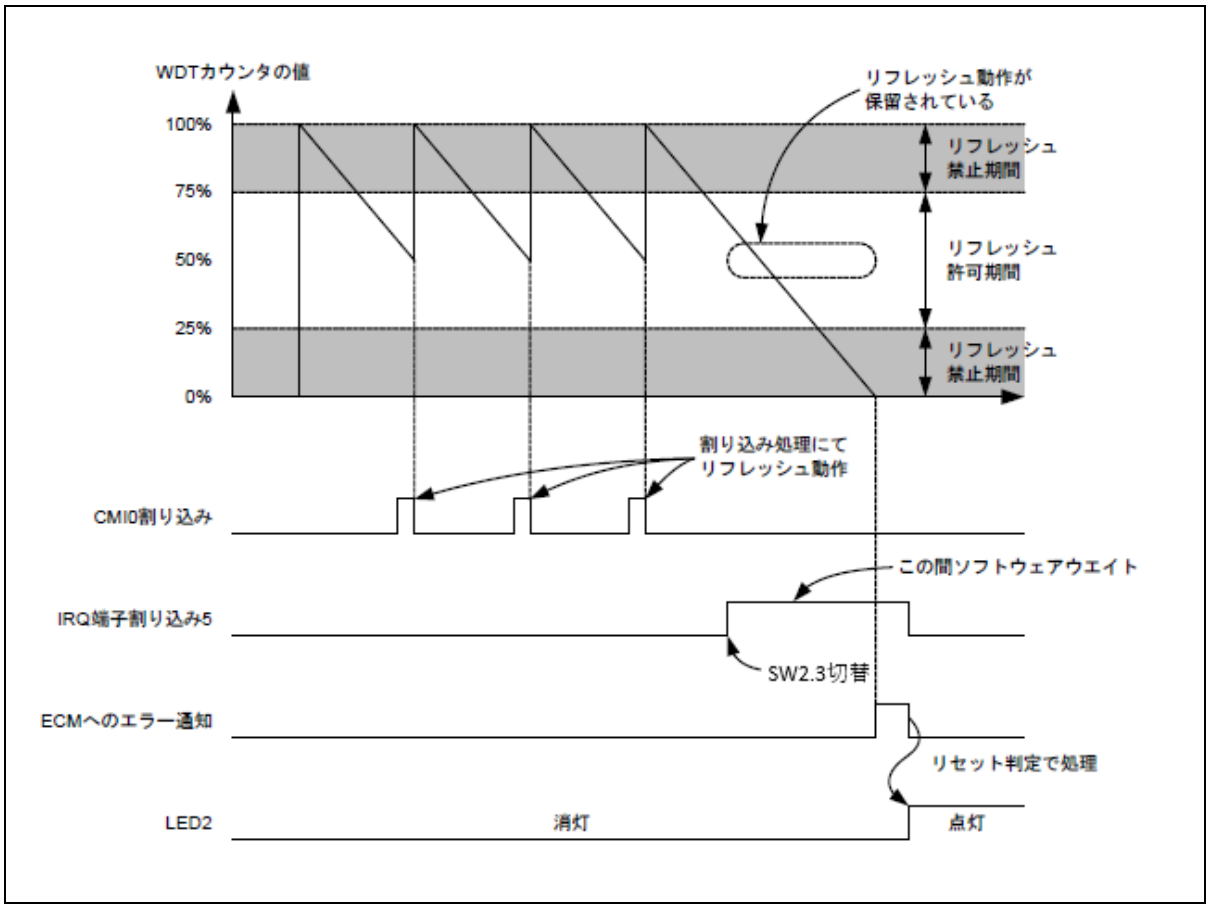


図 8-8 タイミング図

9. IWDTA サンプルソフト

9.1 概要

本章では、評価ボード上に実装される 独立ウォッチドッグタイマ(IWDT)のリセット制御を行うサンプルプログラムについて説明します。

IWDTA サンプルプログラムの特長を以下に示します。

・独立ウォッチドッグタイマ（IWDTA）動作後、コンペアマッチタイマ（CMT）による定周期（273ms）でリフレッシュ動作させます。

・外部割り込み（IRQ2）によるソフトウェアウエイトの発生でリフレッシュ動作が停止し、リセットが発生します。リセット発生後に LED2 が点灯します。

対象デバイス

EC-1

本サンプルを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

9.2　定数一覧

表 9-1 にサンプルコードで使用する定数を示します。

表 9-1 サンプルコードで使用されている定数

定数名	設定値	内容
CMT0_CLOCK_PCLKD_512	3	Count clock = PCLKD/512
CMT0_CMI0_ENABLE	1	Enable CMI0 interrupt
CMT0_INTERVAL_TIME	0x9C35	Set interval time to 273ms
CPG_CMT0_START	1	Start CMT0 count

9.3　関数一覧

表 9-2 にサンプルコードで使用する関数を示します。

表 9-2　関数一覧

関数名	概要	スコープ	定義ファイル
main	メイン処理	local	main.c
port_init	ポート設定初期化	local	main.c
ecm_init	ECM 設定初期化	local	main.c
icu_init	割り込み設定	local	main.c
cmt_init	CMT モジュール初期化	local	main.c
iwdt_init	IWDT 初期化処理	local	main.c
soft_wait	ソフトウェアウェイト	local	main.c
R_IRQ2_isr	IRQ2 割り込み（IRQ 端子割り込み 2）処理	local	main.c
R_IRQ21_isr	IRQ21 割り込み（コンペアマッチタイマ（CMI0））処理	local	main.c

9.4　関数詳細

9.4.1　main

(1)　概要

メイン処理

(2)　C 言語形式

void main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。

- ポート設定初期化
- ECM 初期化
- CMT 初期化
- ICU 初期化
- IWDT 初期化

上記の初期設定を行い、CMT0 の動作を開始させます。その後、メインループにて LED1 の点灯/消灯を繰り返します。

(5)　戻り値

なし

9.4.2 port_init

(1) 概要

ポート設定初期化

(2) C 言語形式

void port_init (void);

(3) パラメータ

なし

(4) 機能

ポート設定の初期化を行います。

(5) 戻り値

なし

9.4.3　ecm_init

(1)　概要

ECM 設定初期化

(2)　C 言語形式

void ecm_init (void);

(3)　パラメータ

なし

(4)　機能

ECM 設定の初期化を行います。
リセット処理を有効にします。

(5)　戻り値

なし

9.4.4 icu_init

(1) 概要

割り込み設定

(2) C 言語形式

void icu_init (void);

(3) パラメータ

なし

(4) 機能

割り込み処理を有効にします。

(5) 戻り値

なし

9.4.5 cmt_init

(1) 概要

CMT モジュール初期化

(2) C 言語形式

void cmt_init (void);

(3) パラメータ

なし

(4) 機能

CMT チャネル 0 を初期化します。

(5) 戻り値

なし

9.4.6 iwdt_init

(1) 概要

IWDT 初期化処理

(2) C 言語形式

void iwdt_init (void);

(3) パラメータ

なし

(4) 機能

IWDT を初期化し、IWDT のカウント動作を開始させます。

(5) 戻り値

なし

9.4.7 soft_wait

(1) 概要

ソフトウェアウェイト処理

(2) C 言語形式

void soft_wait(void);

(3) パラメータ

なし

(4) 機能

NOP を使用したソフトウェアウェイト処理を行います。

(5) 戻り値

なし

9.4.8　R_IRQ2_isr

(1)　概要

IRQ2 割り込み（IRQ 端子割り込み 2）処理

(2)　C 言語形式

void R_IRQ2_isr (void);

(3)　パラメータ

なし

(4)　機能

ソフトウェアウエイト処理（約 3 秒）を実行します。この間に独立ウォッチドッグタイマをアンダーフローさせ、ECM リセットを発生させます。
リセット解除後のリセット判定にて、LED2 を点灯させます。

(5)　戻り値

なし

9.4.9 R_IRQ21_isr

(1) 概要

IRQ21 割り込み（コンペアマッチタイマ（CMI0））処理

(2) C 言語形式

void R_IRQ21_isr (void);

(3) パラメータ

なし

(4) 機能

独立ウォッチドッグタイマのリフレッシュ動作を行います。

(5) 戻り値

なし

9.5　フローチャート

9.5.1　メイン処理

サンプルコードのメイン処理のフローチャートを示します。

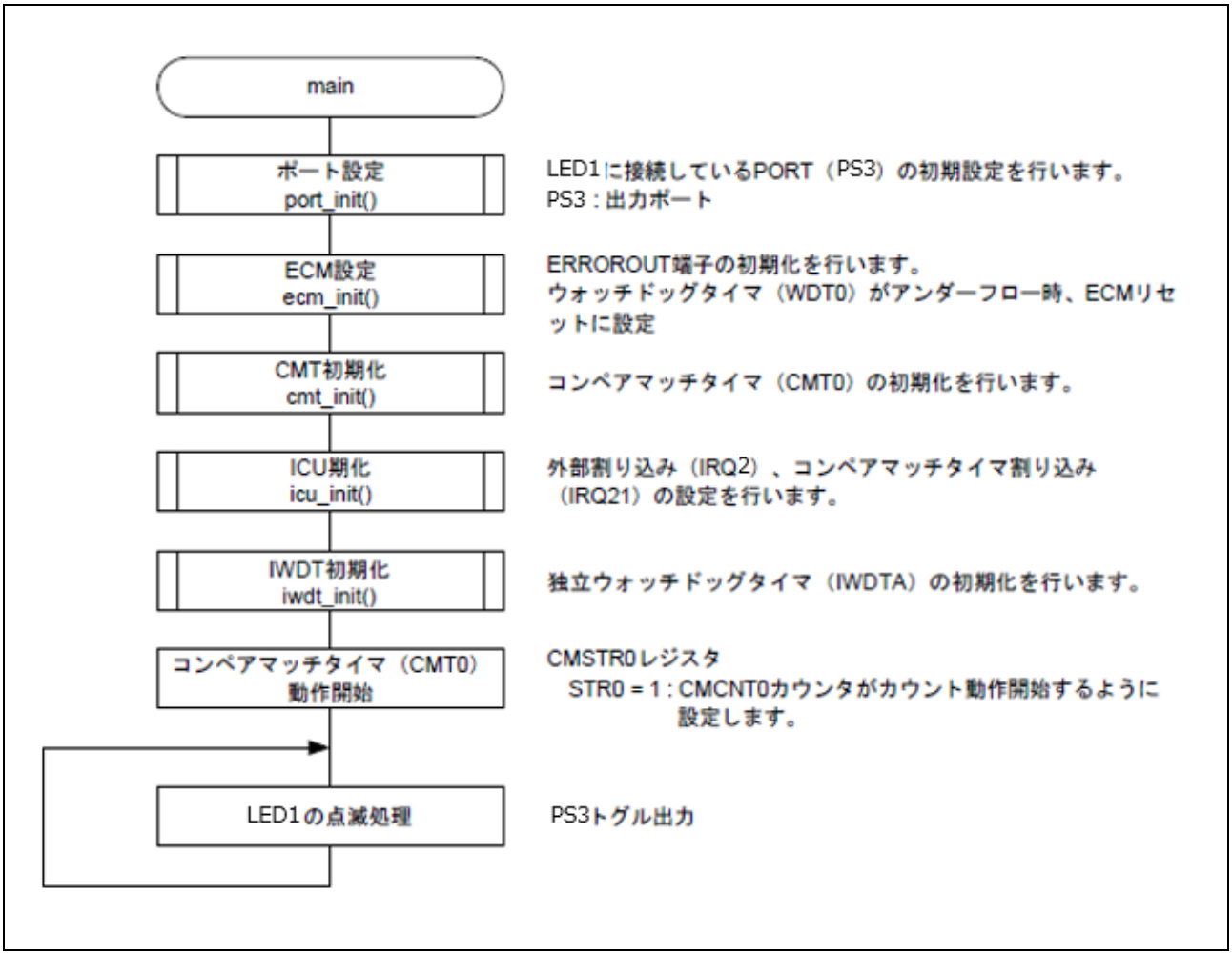


図 9-1 サンプルコードのメイン処理

9.5.2　IWDT 初期化処理

IWDT 初期化処理のフローチャートを示します。

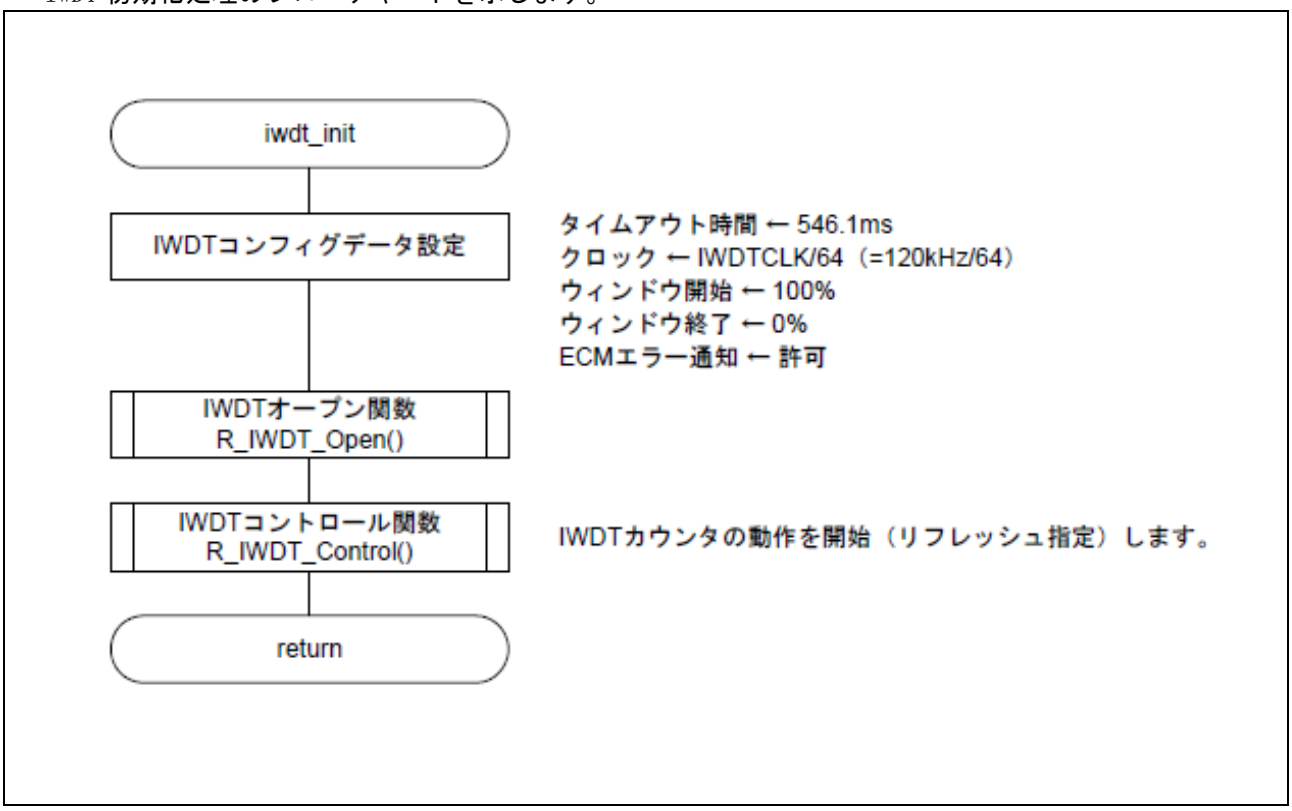
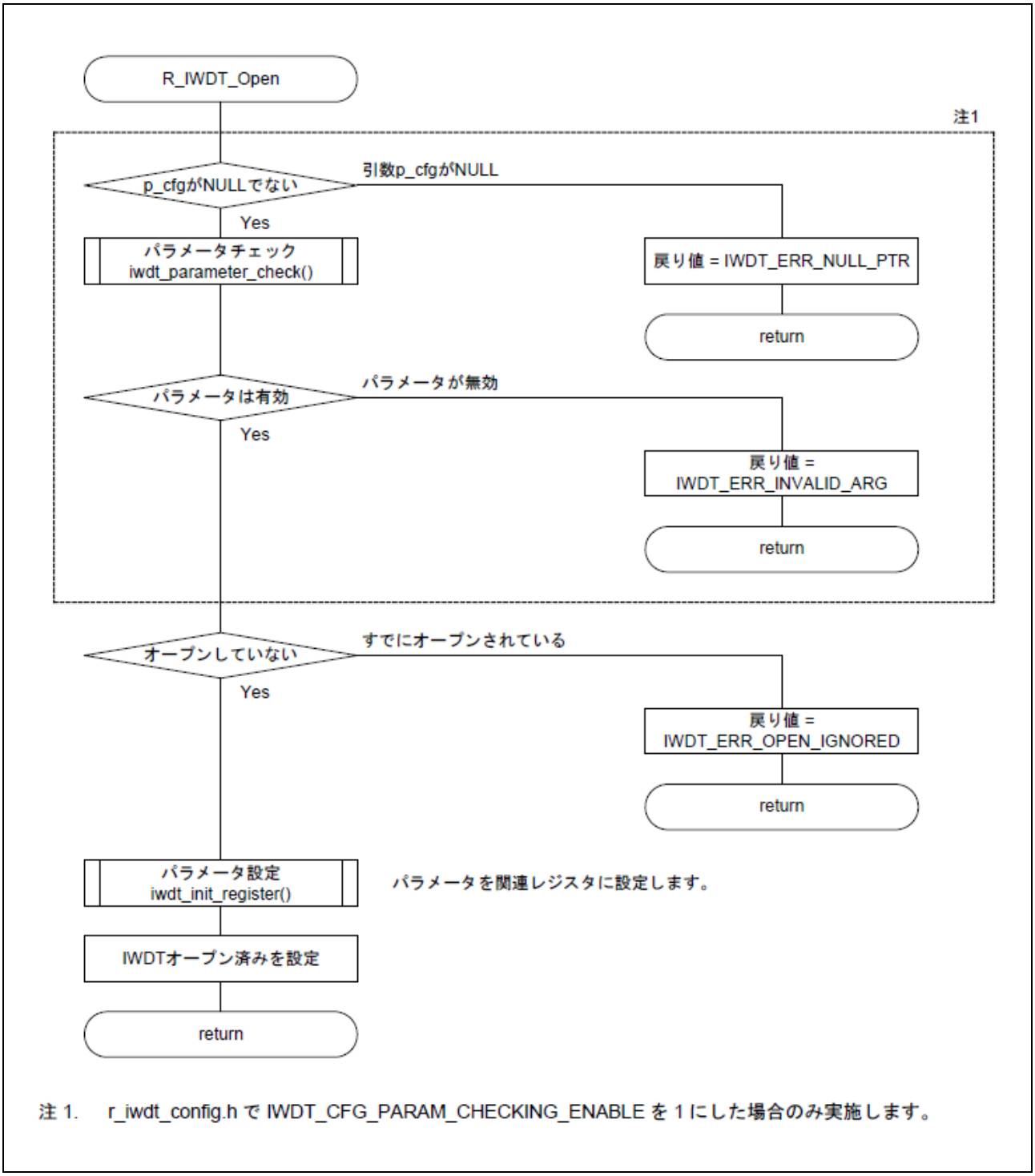


図 9-2 IWDT初期化処理

9.5.3　IWDT オープン関数

IWDT オープン関数のフローチャートを示します。



注 1.　r_iwdt_config.h で IWDT_CFG_PARAM_CHECKING_ENABLE を 1 にした場合のみ実施します。

図 9-3 IWDTオープン処理

9.5.4　IWDT コントロール関数

IWDT コントロール関数のフローチャートを示します。

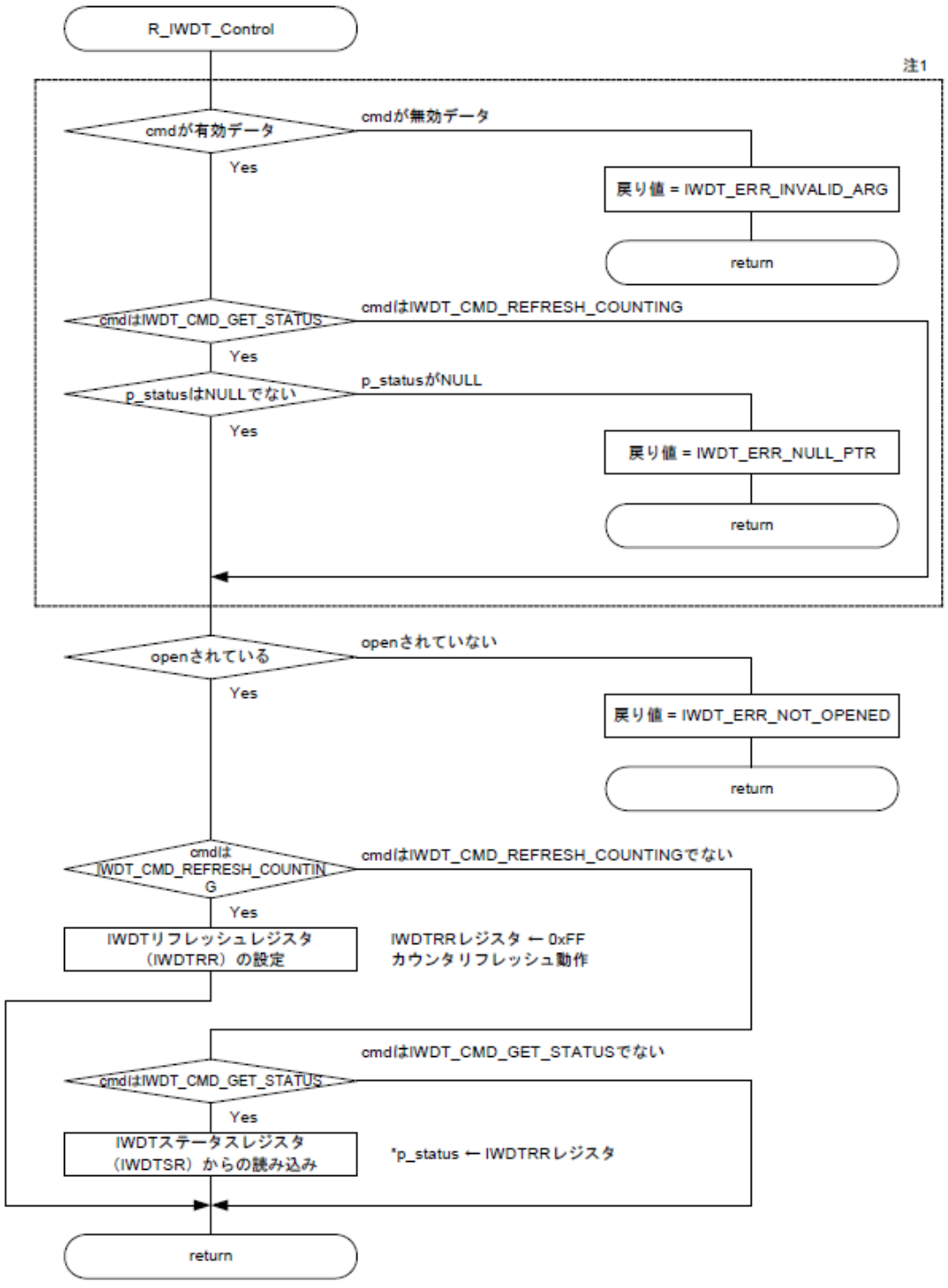


図 9-4 IWDTコントロール処理

9.5.5　IRQ2 割り込み（IRQ 端子割り込み 2）処理

IRQ2 割り込み（IRQ 端子割り込み 2）処理のフローチャートを示します。

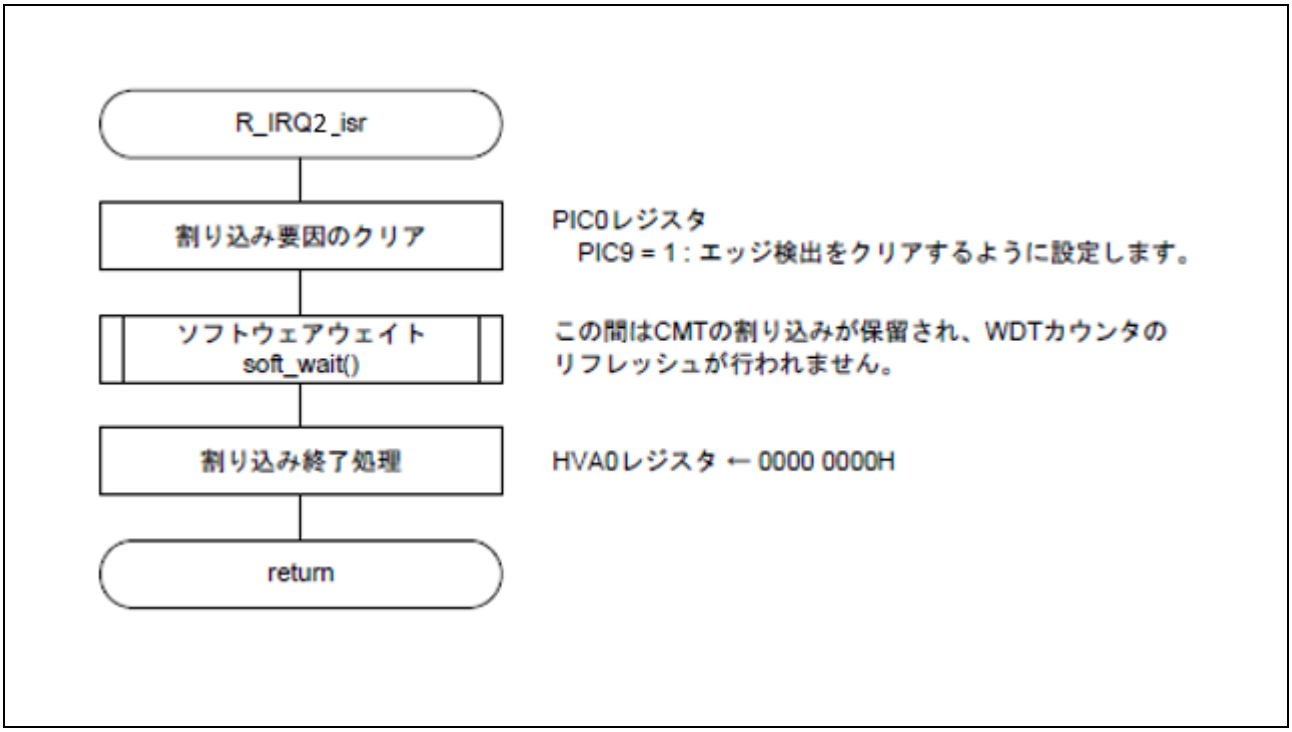


図 9-5 IRQ2 割り込み（IRQ 端子割り込み2）処理

9.5.6　IRQ21 割り込み（コンペアマッチタイマ 0 割り込み）処理

IRQ21 割り込み（コンペアマッチタイマ ch0 割り込み）処理のフローチャートを示します。

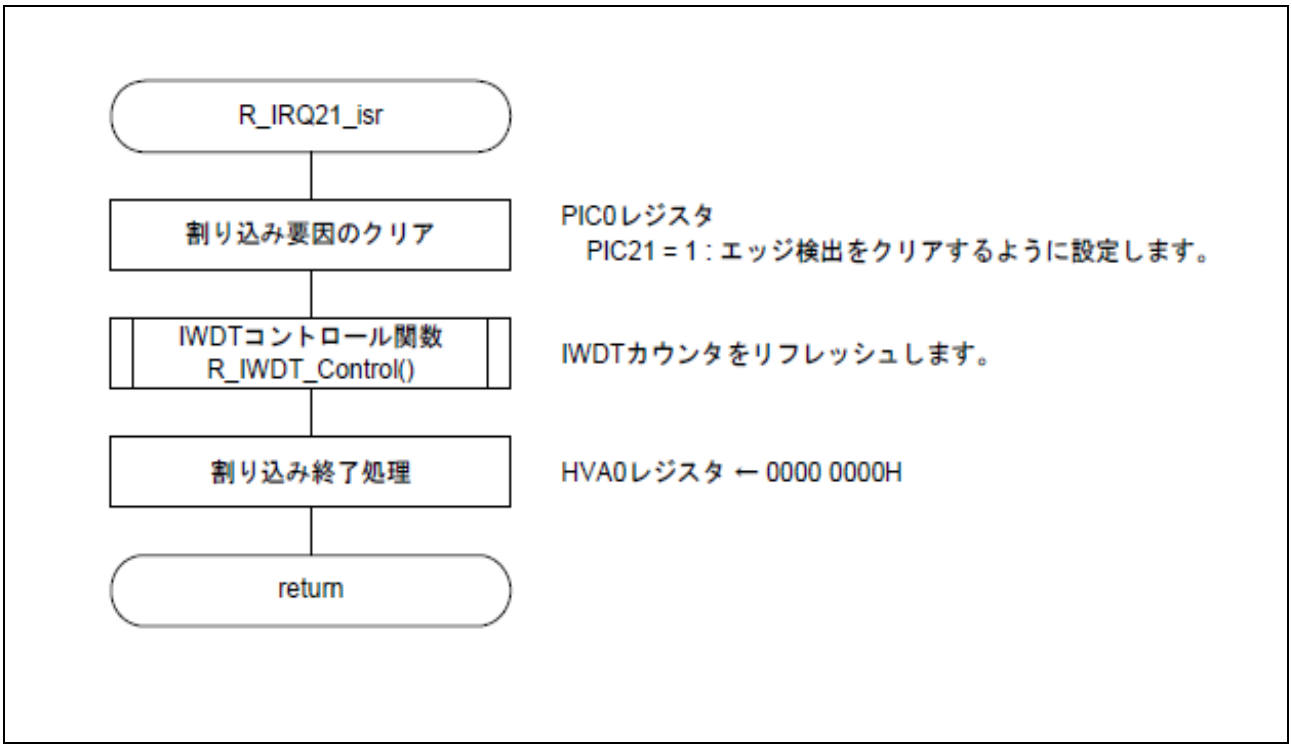


図 9-6 IRQ21 割り込み（コンペアマッチタイマch0 割り込み）処理

9.6　チュートリアル

9.6.1　動作概要

本サンプルプログラムでは、独立ウォッチドッグタイマ（IWDTA）の初期設定を行い、以降、コンペアマッチタイマのインターバル割り込みで定期的にリフレッシュ動作を行います。

Dip スイッチ(SW2.3) を切替すると外部端子割り込み 2 が発生し、割り込み内でソフトウェアウエイト処理が実施されます。この間は CMT のインターバル割り込みが保留されるため、独立ウォッチドッグタイマ（IWDTA）のリフレッシュ動作が停止します。そのため、独立ウォッチドッグタイマのカウンタ値がアンダーフローし、エラーコントロールモジュール（ECM）にエラー通知を行い、ECM リセットが発生します。そしてリセット解除後のリセット判定処理にて LED2 が点灯します。

本サンプルプログラムの機能概要を表 9-3 動作概要に示します。また、図 9-7 にタイミング図を示します。

表 9-3 動作概要

機能	概要
クロック	IWDTCLK/64（=120kHz/64）
タイムアウト	1024cycle（=546.1ms）
ウィンドウ	開始 ：100% 終了 ：0%
ECM エラー通知	許可

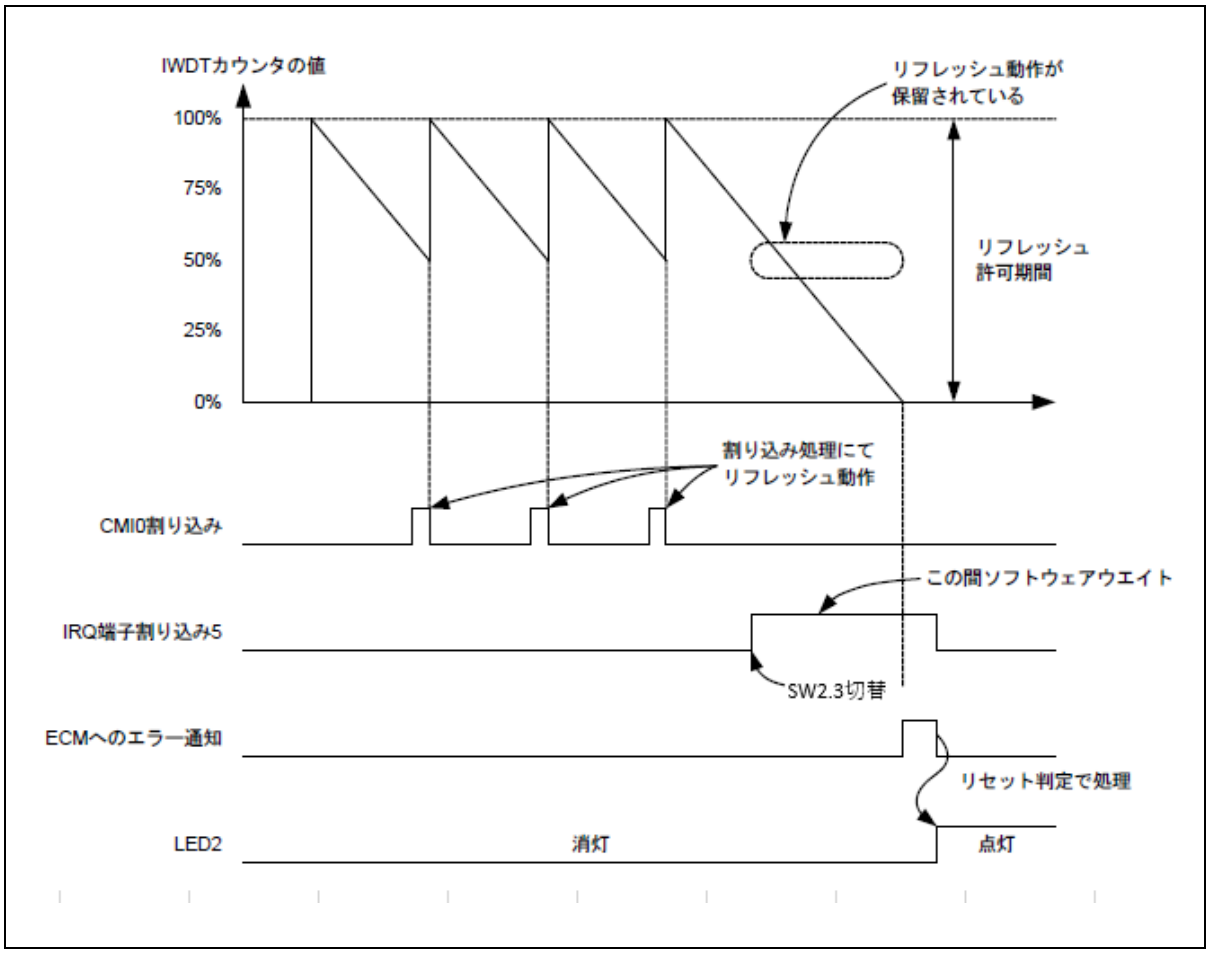


図 9-7 タイミング図

10. RIIC サンプルソフト

10.1　概要

本章では、評価ボード上に実装される I2C バスインタフェース機能（RIIC）を使用し評価ボード上に実装されている EEPROM（R1EX24016ASAS0G）の Read/Write を実行するサンプルプログラムについて説明します。

RIIC サンプルプログラムの特長を以下に示します。

・マスタ送信、マスタ受信対応

・通信モードはファストモードに対応（最大転送速度は 400kbps）

制限事項

本サンプルプログラムには以下の制限事項があります。

（1）DMA と組み合わせて使用することはできません。

（2）RIIC のタイムアウト機能には対応していません。

（3）RIIC の NACK アービトレーションロスト機能に対応していません。

（4）10 ビットアドレスの送信に対応していません。

（5）スレーブデバイス時、リスタートコンディションの受付に対応していません。
リスタートコンディション直後のアドレスで本モジュールのアドレスを指定しないでください。

対象デバイス

EC-1

本サンプルを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

10.2 定数一覧

表 10-1 にサンプルコードで使用する定数を示します。

表 10-1 サンプルコードで使用されている定数

定数名	設定値	内容
WRITE_VAL	0xA5	Data values to be written to the EEPROM.
PAGE_SIZE	0x10	The maximum number of bytes that can be written once.
PORT_OUTPUT	0x3U	Setting value to the PDR register.
PORT_PULL_UP_DOWN_NONE	0x0U	Setting value to the PCR register.
PORT_GPIO_MODE	0x0U	Setting value to the PMR register.
CMT_STOP	0x0U	Setting value to the CMSTR0 register. (timer stop)
CMT_START	0x1U	Setting value to the CMSTR0 register. (timer start)
CMT_INT_DISABLE	0x0U	Setting value to the CMCR0 register. (interrupt disable)
CMT_INT_ENABLE	0x1U	Setting value to the CMCR0 register. (interrupt enable)
CMT_CLOCK_SELECT	0x0U	Setting value to the CMCR0 register. (Division ratio of PCLKD)
CMT_COMPARE_VAL	9375U	Setting value to the CMCOR0 register. (compare match period)
PRCR_WRITE_DISABLE	0x0000A500UL	Setting value to the PRCR register. MSTPCRA write protected)
PRCR_WRITE_ENABLE	0x0000A502UL	Setting value to the PRCR register. MSTPCRA write enabled)
MSTPCRA_CMT0_STOP	0x1U	Setting value to MSTPCRA register. (CMT0 stop)
MSTPCRA_CMT0_START	0x0U	Setting value to MSTPCRA register. (CMT0 start)
EEPROM_SIZE	2048U	EEPROM size
RIIC_CB_SUCCESS	0L	Notification driver successful completion
RIIC_CB_WAIT	1L	Wait Driver Callback
RIIC_CB_NACK	2L	Notification driver NACK
RIIC_CB_FAIL	-1L	Notification driver fail termination
CMT_CB_SUCCESS	0L	Notification driver successful completion
CMT_CB_WAIT	1L	Wait Driver Callback
ON	1U	Setting value to LED ON
OFF	0U	Setting value to LED OFF
LED1(x)	PORTS.PODR.BIT.B3 = (x)	Control LED1
LED2(x)	PORTS.PODR.BIT.B2 = (x)	Control LED2
LED3(x)	PORTS.PODR.BIT.B1 = (x)	Control LED3
LED4(x)	PORTS.PODR.BIT.B0 = (x)	Control LED4
FAIL_SAFE_COUNT	0xFFFFFFFFU	fail safe counter
HVA_DUMMY_DATA	0xC0FFEEU	Dummy data values to be written to inform the end-of-interrupt.
RECEIVE_ACK	0L	-
RECEIVE_NACK	-1L	-

10.3　関数一覧

表 10-2 にサンプルコードで使用する関数を示します。

表 10-2　関数一覧

関数名	概要	スコープ	定義ファイル
main	メイン処理	local	main.c
ecm_init	ECM 設定初期化	local	main.c
icu_init	割り込み設定	local	main.c
isr_cmt	CMT モジュール初期化	local	main.c
cb_riic	RIIC コールバック関数	local	main.c
wait_riic_callback	コールバック呼び出し待機処理	local	main.c
wait_1ms	ウェイト処理	local	main.c
ee_read	EEPROM データ読み出し	local	main.c
ee_write	EEPROM データ書き込み	local	main.c
init_led	LED 設定初期化	local	main.c
wreite_read_check	EEPROM データチェック	local	main.c

10.4　関数詳細

10.4.1　main

(1)　概要

メイン処理

(2)　C 言語形式

void main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。
処理内容は 10.5.1 メイン処理　を参照してください。

(5)　戻り値

なし

10.4.2 ecm_init

(1) 概要

ECM 設定初期化

(2) C 言語形式

void ecm_init (void);

(3) パラメータ

なし

(4) 機能

ECM 設定の初期化を行います。

(5) 戻り値

なし

10.4.3 icu_init

(1) 概要

割り込み設定

(2) C 言語形式

void icu_init (void);

(3) パラメータ

なし

(4) 機能

割り込み処理を有効にします。

(5) 戻り値

なし

10.4.4 isr_cmt

(1) 概要

CMT 割り込みハンドラ設定

(2) C 言語形式

static void isr_cmt(void)

(3) パラメータ

なし

(4) 機能

CMT の割り込みハンドラを設定します。

(5) 戻り値

なし

10.4.5 cb_riic

(1) 概要

RIIC ドライバコールバック関数

(2) C 言語形式

static void cb_riic(void)

(3) パラメータ

なし

(4) 機能

RIIC ドライバから呼び出されるコールバック関数となります。

(5) 戻り値

なし

10.4.6 wait_riic_callback

(1) 概要

コールバック呼び出し待機処理

(2) C 言語形式

static int32_t wait_riic_callback(void)

(3) パラメータ

なし

(4) 機能

コールバック関数が呼び出されるまで待機します。

(5) 戻り値

なし

10.4.7 wait_1ms

(1) 概要

ウェイト処理

(2) C 言語形式

static void wait_1ms(void)

(3) パラメータ

なし

(4) 機能

コールバックが通知されるまで待機します。

(5) 戻り値

なし

10.4.8　ee_read

(1)　概要

EEPROM データ読み出し

(2)　C 言語形式

static riic_return_t ee_read(uint16_t address, uint32_t size, uint8_t *p_buf)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	address	EEPROM読み込み開始アドレス
I	uint32_t	size	読み込みサイズ(byte)
I	uint8_t	*p_buf	読み込んだデータを格納するバッファのアドレス。

(4)　機能

引数の値により EEPROM に書き込まれたデータを読み込みます。
読み込み中は LED4 を点灯させます。

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_LOCK_FUNC	他のタスクがAPI をロックしている場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合
RIIC_ERR_OTHER	現状態に該当しない不正なイベントが発生した場合

10.4.9　ee_write

(1)　概要

EEPROM データ書き込み

(2)　C 言語形式

static riic_return_t ee_write(uint16_t address, uint32_t size, uint8_t *p_buf)

(3)　パラメータ

I/O	パラメータ		説明
I	uint16_t	address	EEPROM書き込み開始アドレス
I	uint32_t	size	書き込みサイズ(byte)
I	uint8_t	*p_buf	データを書き込むバッファのアドレス。

(4)　機能

引数の値により EEPROM にデータを書き込みます。
書き込み中は LED3 を点灯させます。

(5)　戻り値

戻り値	意味
RIIC_SUCCESS	問題なく処理が完了した場合
RIIC_ERR_LOCK_FUNC	他のタスクがAPI をロックしている場合
RIIC_ERR_INVALID_CHAN	存在しないチャネルの場合
RIIC_ERR_INVALID_ARG	不正な引数の場合
RIIC_ERR_OTHER	現状態に該当しない不正なイベントが発生した場合

10.4.10 init_led

(1) 概要

LED 設定初期化

(2) C 言語形式

static void init_led(void)

(3) パラメータ

なし

(4) 機能

LED のポート設定を初期化します。

(5) 戻り値

なし

10.4.11　write_read_check

(1)　概要

EEPROM データチェック

(2)　C 言語形式

static int32_t write_read_check(void)

(3)　パラメータ

なし

(4)　機能

リード/ライトデータをチェックし、一致している場合、LED2 を点灯し続けます。
不一致の場合、消灯します。

(5)　戻り値

なし

10.5　フローチャート

10.5.1　メイン処理

サンプルコードのメイン処理のフローチャートを示します。

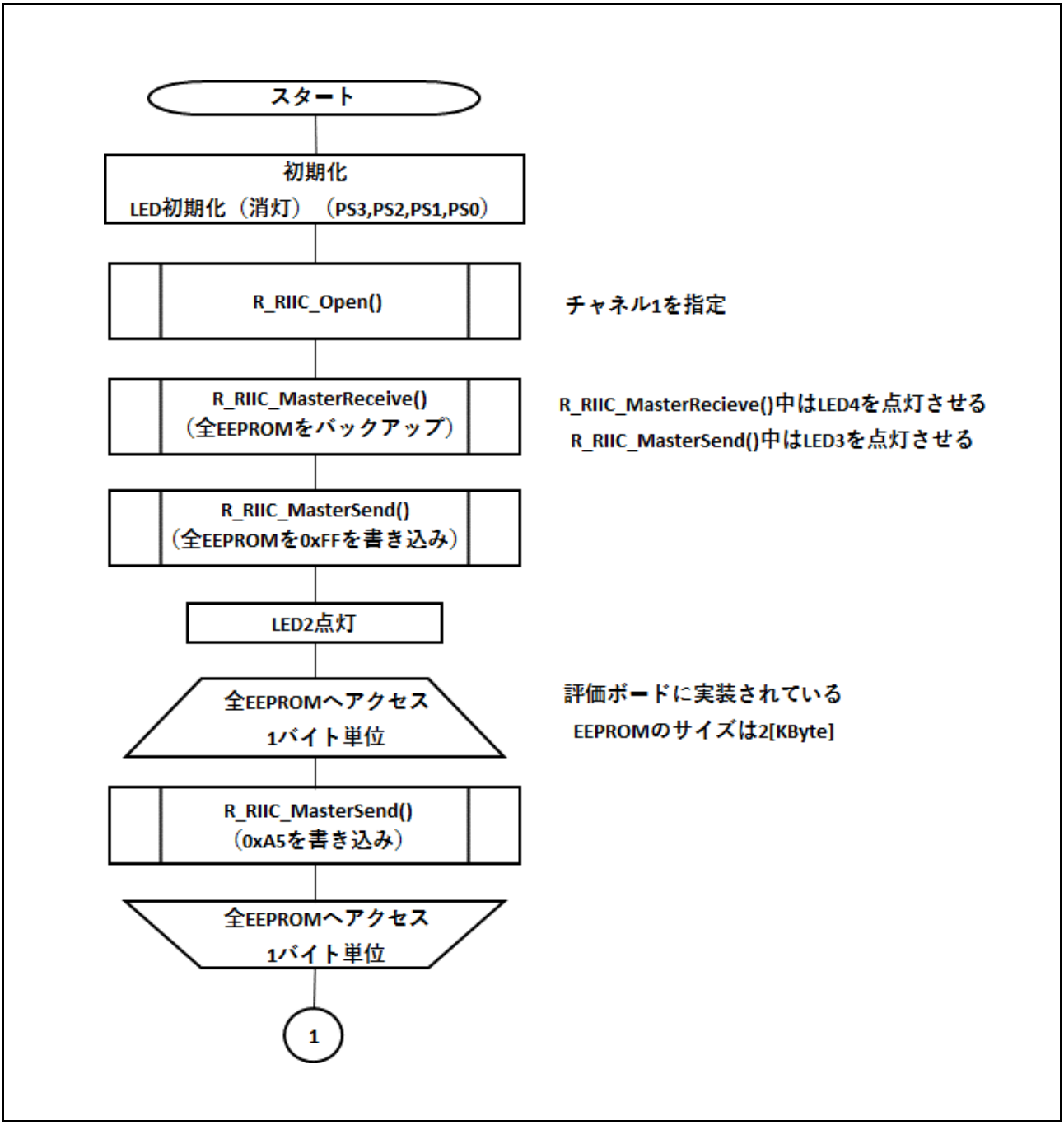


図 10-1 サンプルコードのメイン処理

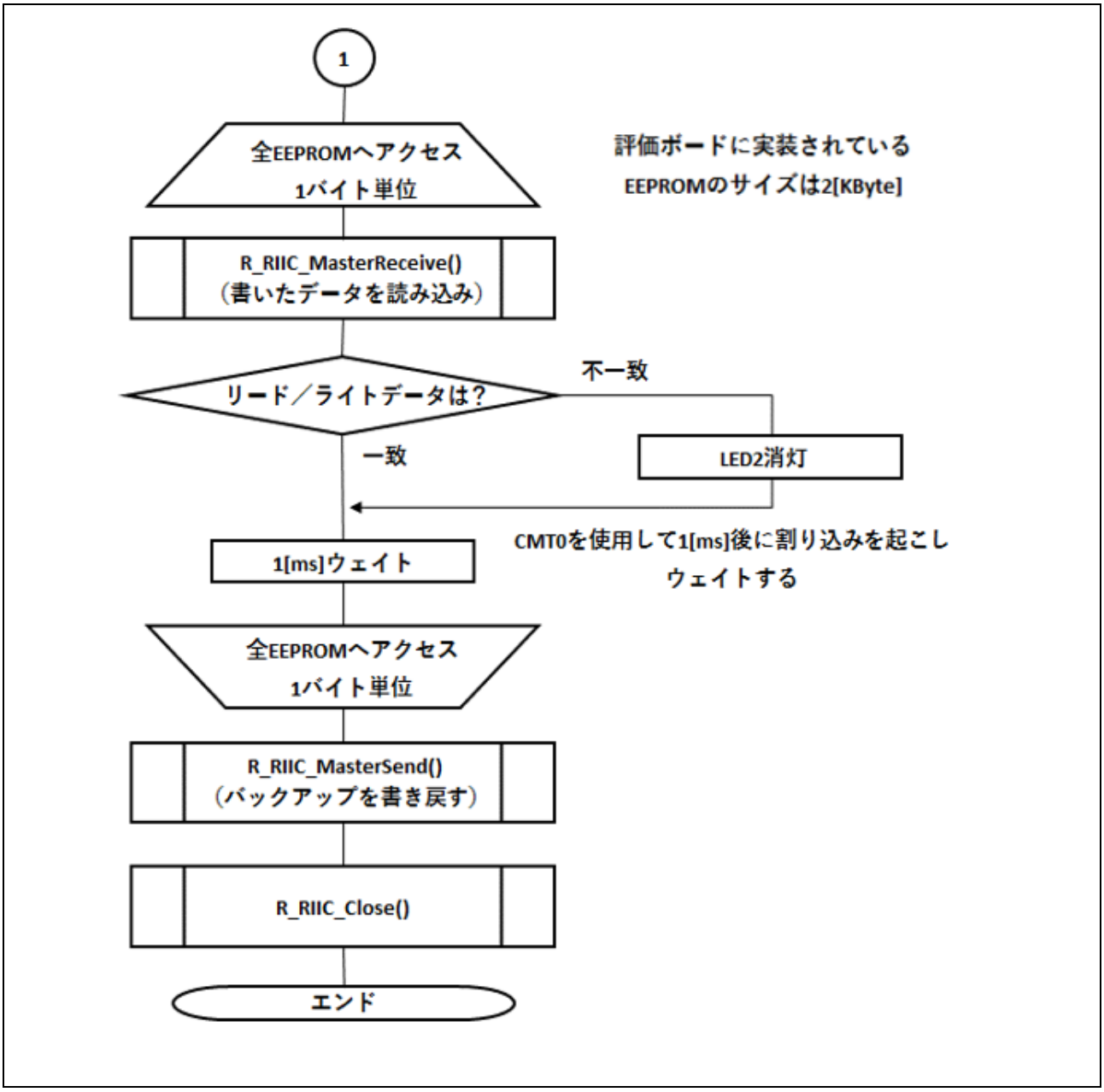


図 10-2 サンプルコードのメイン処理

10.5.2　コールバック処理

サンプルコードのコールバック処理のフローチャートを示します。

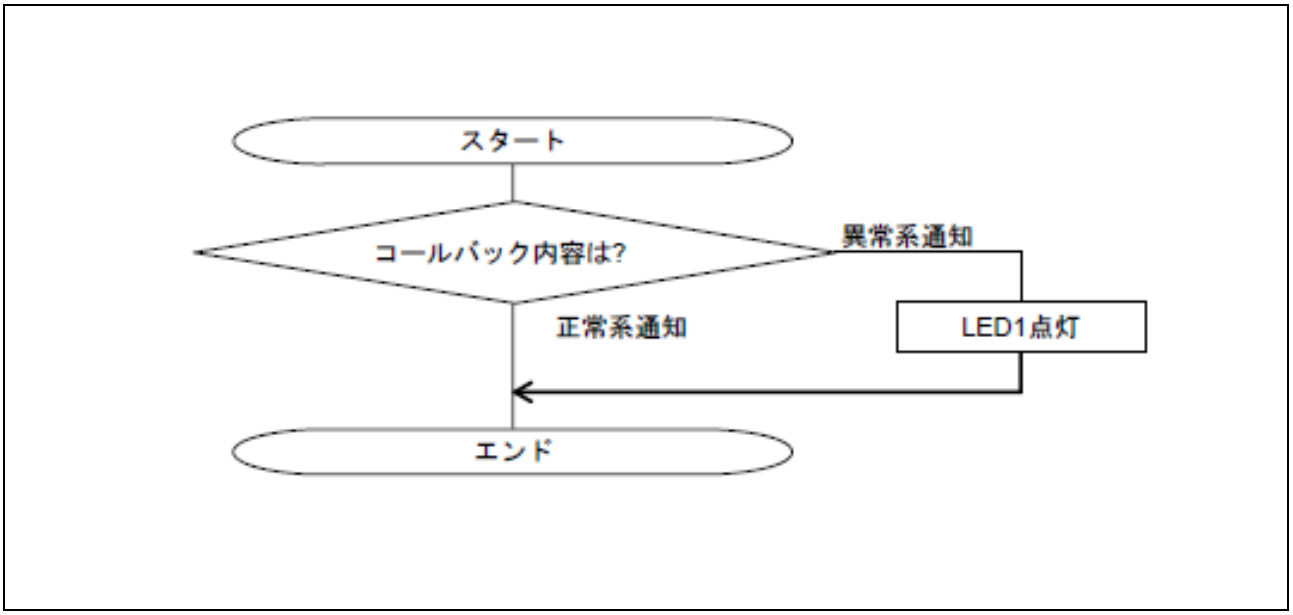


図 10-3 サンプルコードのコールバック処理

10.5.3　コンペアマッチタイマ割り込み処理

サンプルコードのコンペアマッチタイマ割り込み処理のフローチャートを示します。

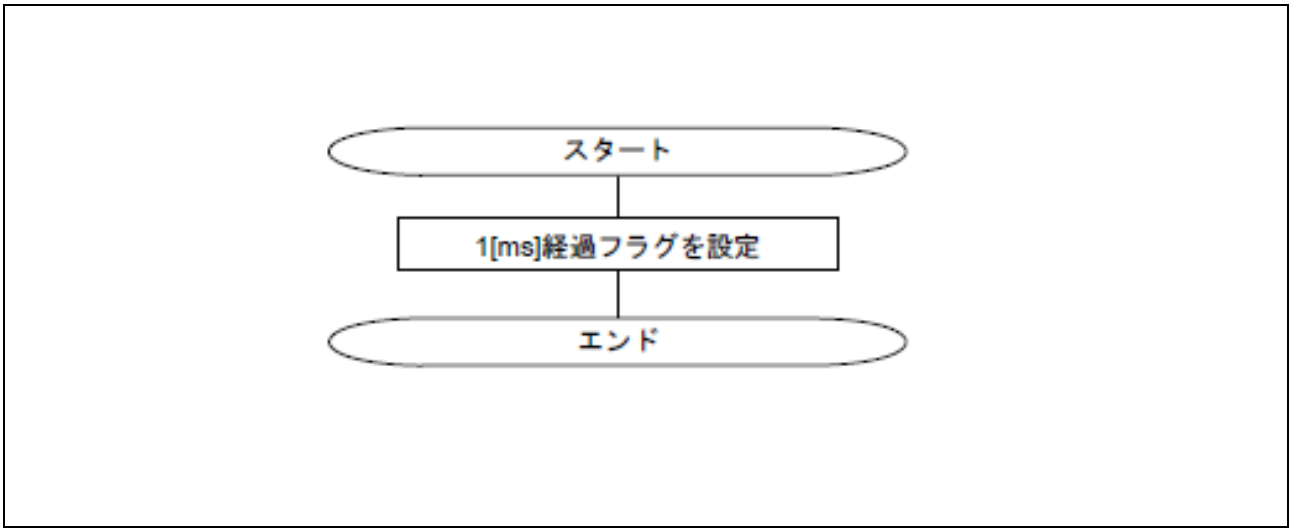


図 10-4 サンプルコードのコンペアマッチタイマ割り込み処理

10.6　チュートリアル

10.6.1　動作概要

RIIC サンプルプログラムの機能概要を表 6-4 動作概要に示します。また、図 6-3 にシステムブロック図を示します。

表 10-3 動作概要

機能	概要
兼用端子設定	• PC6、PC7 を SCL1、SDA1 に設定
RIIC の通信チャネル	• EEPROM が接続されているチャネル1 に設定
割り込み要因（割り込み優先度）	• RIIC モジュール 送信終了（1）／受信終了（1）／送信バッファエンプティ（1）／異常検出（1） • CMT モジュール（1[ms] 検出用） コンペアマッチ(15)
転送速度設定	• 400[kbps]
動作モード	• マスタ送信／マスタ受信
動作概要	1. EEPROM の全内容をバックアップ（RAM） 2. EEPROM の全体に0xFF を書き込む 3. EEPROM の全体に0xA5 を書き込む 4. 書き込み内容の確認 5. EEPROM を変更前の内容を書き戻す （EEPROM への Read/Write アクセスを行う毎に1[ms] のインタバールを入れる）
動作結果表示	• LED1 点灯　RIIC の通信異常を検出 • LED2 点灯 テストパターンが一致 • LED3 点灯 EEPROM へ write 中 • LED4 点灯 EEPROM のread 中

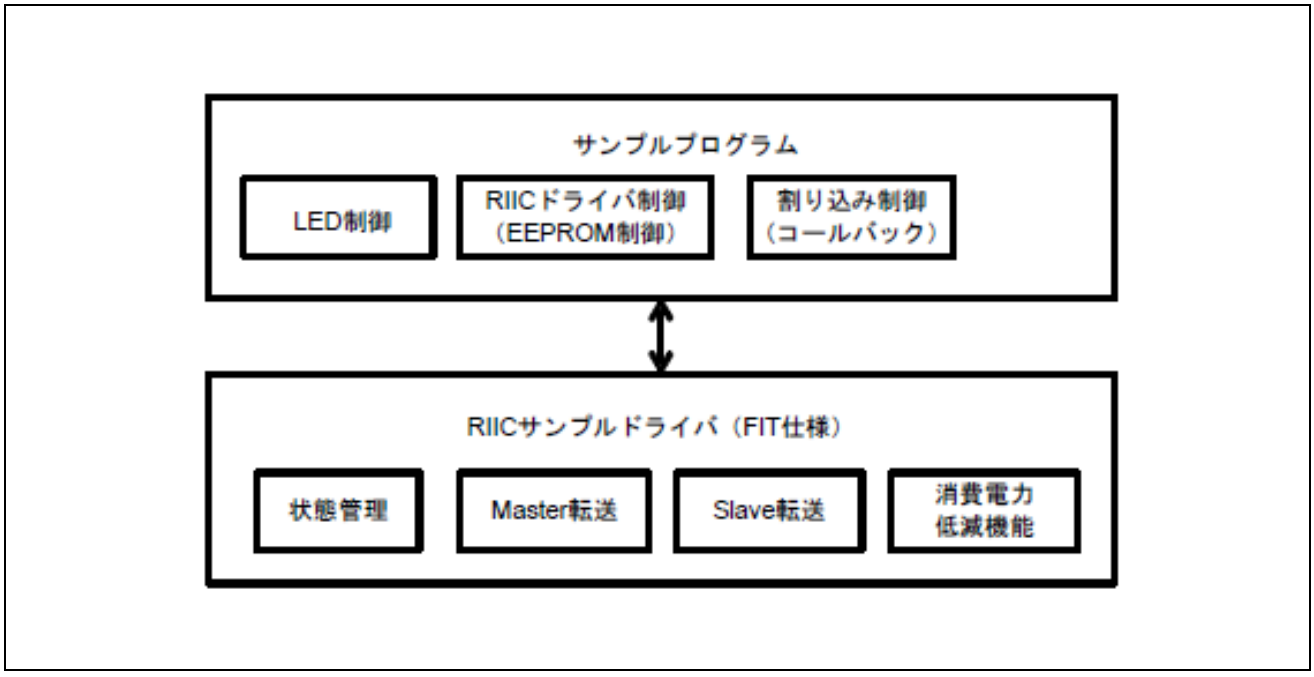


図 10-5 システムブロック図

(1)　RIIC サンプルドライバ状態遷移図

図 10-6 に RIIC サンプルドライバの状態遷移図を示します。

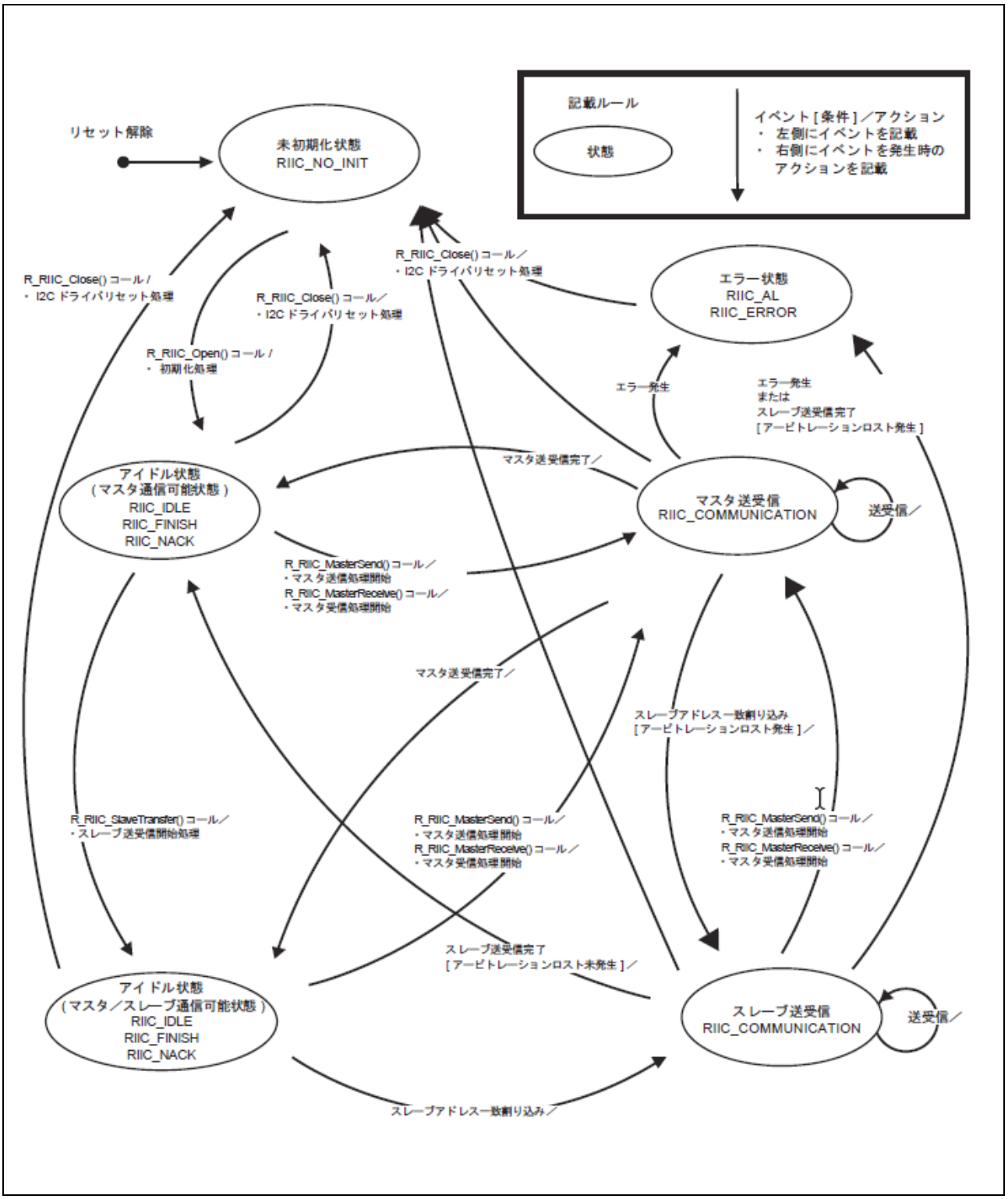


図 10-6 RIICサンプルドライバの状態遷移図

(2)　RIIC サンプルドライバ状態遷移時の各フラグ

I2C 通信情報構造体メンバには、デバイス状態フラグ（dev_sts）があります。デバイス状態フラグには、そのデバイスの通信状態が格納されます。また、このフラグにより、同一チャネル上の複数のスレーブデバイスの制御を行うことができます。表 10-4 に状態遷移時のデバイス状態フラグの一覧を示します。

表 10-4 状態遷移時のデバイス状態フラグの一覧

状態	デバイス状態フラグ（dev_sts）
未初期化状態	RIIC_NO_INIT
アイドル状態	RIIC_IDLE RIIC_FINISH RIIC_NACK
通信中 （マスタ送信、マスタ受信、 スレーブ送信、スレーブ受信）	RIIC_COMMUNICATION
アービトレーションロスト検出状態	RIIC_AL
エラー	RIIC_ERROR

(3)　RIIC サンプルドライバアービトレーションロスト検出機能

本モジュールは、以下に示すアービトレーションロストを検出することができます。なお、RIIC は、以下に加えて、スレーブ送信時におけるアービトレーションロストの検出にも対応していますが、本モジュールは対応していません。

(i) バスビジー状態で、スタートコンディションを発行したとき
既に他のマスタデバイスがスタートコンディションを発行して、バスを占有している状態（バスビジー状態）でスタートコンディションを発行すると、本モジュールはアービトレーションロストを検出します。

(ii) バスビジー状態ではないが、他のマスタより遅れてスタートコンディションを発行したとき
本モジュールは、スタートコンディションを発行するとき、SDA ラインを Low にしようとします。しかし、他のマスタデバイスがこれよりも早くスタートコンディションを発行した場合、SDA ライン上の信号レベルは、本モジュールが出力したレベルと一致しなくなります。このとき、本モジュールはアービトレーションロストを検出します。

(iii) スタートコンディションが同時に発行されたとき
複数のマスタデバイスが、同時にスタートコンディションを発行すると、それぞれのマスタデバイス上でスタートコンディションの発行が正常に終了したと判断されることがあります。その後、それぞれのマスタデバイスは通信を開始しますが、以下に示す条件が成立すると、本モジュールはアービトレーションロストを検出します。

(a) それぞれのマスタデバイスが送信するデータが異なる場合
データ通信中、本モジュールは SDA ライン上の信号レベルと、本モジュールが出力したレベルを比較しています。そのため、スレーブアドレス送信を含むデータ送信中に、SDA ライン上と本モジュールが出力したレベルが一致しなくなると、本モジュールはその時点でアービトレーションロストを検出します。

(b) それぞれのマスタデバイスが送信するデータは同じだが、データの送信回数が異なる場合
上記 a.に合致しない場合（スレーブアドレスおよび送信データが同じ）、本モジュールはアービトレーションを検出しませんが、それぞれのマスタデバイスがデータを送信する回数が異なる場合であれば、本モジュールはアービトレーションロストを検出します。

10.6.2　使用準備

RIIC サンプルプログラムを動作させるには基盤の改修が必要となります。

(1) 基盤上に実装されている EEPROM への Read/Write を行えるよう端子を接続。
・J9 コネクタ 2pin(SDA1 端子) – U8 コネクタ 5pin(SDA 端子)
・J9 コネクタ 3pin(SCL1 端子) – U8 コネクタ 6pin(SCL 端子)

(2)I2C を使用する為 J10 コネクタの 1pin と 2pin を接続。

図 10-7 に基盤改修内容を示します。

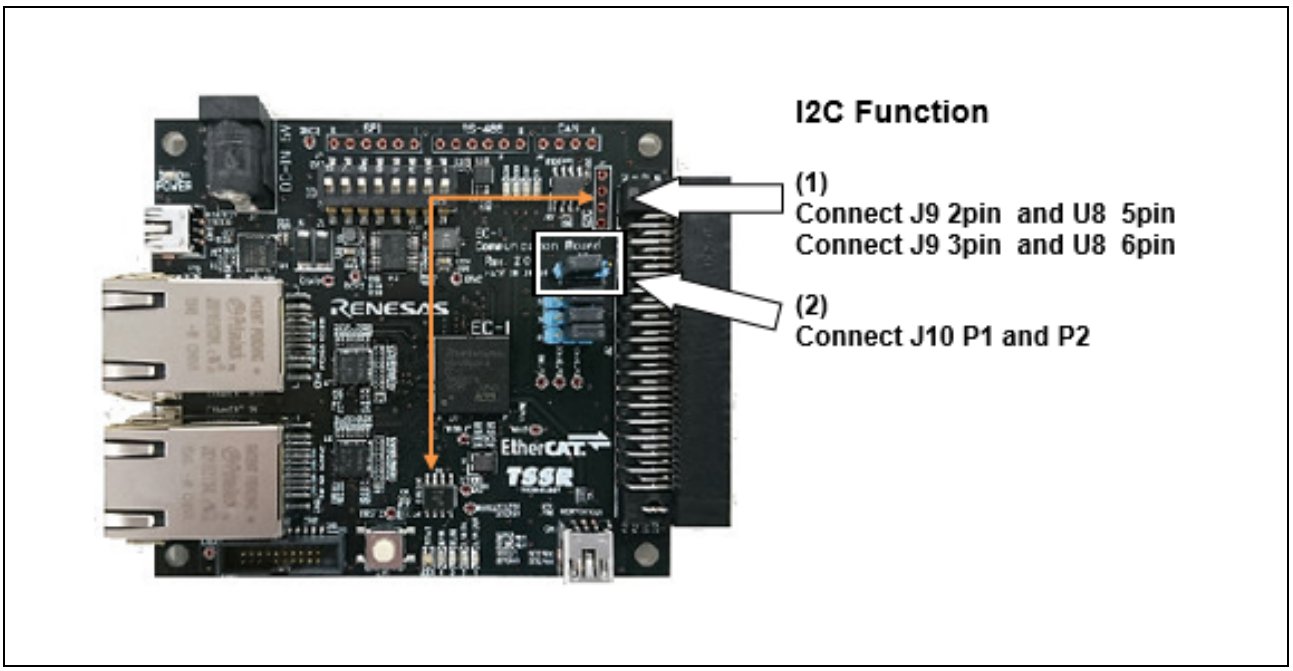


図 10-7 RIICサンプルプログラムを動作させるための基盤改修内容

11. USB ホストサンプルソフト

11.1　概要

本章では、評価ボード上に実装される USB ホストを制御する USB ホストドライバを使用したサンプルプログラムについて説明します。

USB ホストサンプルプログラムの特長を以下に示します。
USB ホストサンプルソフトは以下のモジュール(USB-BASIC-F/W, HMSC)の組み合わせにより動作します。

(1)　・USB-BASIC-F/W

USB-BASIC-F/W は、ホストドライバ（HCD）、ホストマネージャ（MGR）、ハブクラスドライバ（HUBCD）から構成されていて USB2.0HS ホストモジュールの H/W 制御を行います。
MGR は、接続されたデバイスの状態管理とエニュメレーションをします。また、アプリケーション（APL）がデバイス状態を変更する場合は、APL から該当の API 関数をコールすることにより MGR が HCD もしくは HUBCD に状態変更を要求します。
HUBCD は、USB ハブのダウンポートに接続されたデバイスのエニュメレーションと状態管理をするサンプルプログラムです。
USB-BASIC-F/W は Renesas が提供するサンプルデバイスクラスドライバ又はユーザが作成したホストデバイスクラスドライバと組み合わせることで動作します。
以下に本モジュールがサポートしている機能を示します。

- デバイスの接続／切断、サスペンド／レジューム、USB バスリセット処理
- コントロール転送、バルク転送、インターラプト転送、アイソクロナス転送
- Full-Speed / High-Speed ファンクションデバイスとエニュメレーション
  (動作スピードはデバイスにより異なります)
- 転送エラー判定および転送リトライ
- USB-BASIC-F/W をカスタマイズせずに、複数のデバイスクラスドライバを実装することが可能

制限事項
本モジュールには以下の制限事項があります。

- 型の異なるメンバで構造体を構成しているため、コンパイラによって構造体メンバのアドレスアライメントずれが発生することがあります。
- Host 動作時、接続した HUB および HUB ダウンポート接続デバイスのサスペンド/レジュームに対応しておりません。
- データ転送中のサスペンドはサポートしておりません。データ転送が完了したことを確認の上、サスペンドを実行して下さい。
- 過電流検出時にコールバック関数による通知はしますが、実処理はユーザシステムに合わせて作成してください。

※USB-BASIS-F/W の詳細は「11.8 USB-BASIC-F/W」を参照してください。

(2)　・Host Mass Storage Class driver (HMSC)

HMSC は、USB-BASIC-F/W と組み合わせることで、USB Host マスストレージクラスドライバとして動作します。

HMSC は、USB マスストレージクラスの BOT プロトコルで構築されています。ファイルシステムと組み合わせることで BOT 対応の USB ストレージ機器と通信を行うことが可能です。ファイルシステムは FatFs を使用することを前提に作成しています。

以下に本モジュールがサポートしている機能を示します。

- 接続された USB ストレージ機器のデバイス照合（動作可否判定）
- BOT プロトコルによるストレージコマンド通信
- USB マスストレージサブクラスの SFF-8070i(ATAPI)と SCSI に対応
- 複数の USB ストレージ機器を接続可能

制限事項

本モジュールには以下の制限事項があります。

- 型の異なるメンバで構造体を構成しています。

（コンパイラによっては構造体のメンバにアドレスアライメントずれを発生することがあります。）

- 論理ユニット番号 0 (LUN0) のみサポートします。
- セクタサイズが 512 バイトの USB ストレージ機器のみサポートします。
- READ_CAPACITY コマンドに応答しないデバイスはセクタサイズを 512 バイトとして処理します。

※HMSC の詳細は「11.9 ホストマスストレージクラスドライバ（HMSC）」を参照してください。

用語一覧

APL : Application program
ATAPI : AT Attachment Packet Interface
BOT : Mass storage class Bulk Only Transport
FSI : File System Interface
HCD : Host Control Driver of USB-BASIC-F/W
HMSC : Host Mass Storage Class driver
HMSCD : Host Mass Storage Class Driver unit
HMSDD : Host Mass Storage Device Driver
HUBCD : Hub Class Driver
LUN : Logical Unit Number
LBA : Logical Block Address
MGR : Peripheral device state manager of HCD
SCSI : Small Computer System Interface
Scheduler : タスク動作を簡易的にスケジューリングするモジュール
Task : 処理の単位
USB-BASIC-F/W : USB basic firmware for EC-1 グループ
USB : Universal Serial Bus

対象デバイス

EC-1

本サンプルを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

11.2　定数一覧

サンプルコードで使用する定数を示します。

表 11-1 サンプルコードで使用されている定数

定数名	設定値	内容
LED1	PORTS.PODR.BIT.B3	LED ポート設定
LED2	PORTS.PODR.BIT.B2	-
LED3	PORTS.PODR.BIT.B1	-
LED4	PORTS.PODR.BIT.B0	-
USER_DATA_SIZE	512	ユーザ定義データサイズ

11.3　構造体/共用体/列挙型一覧

サンプルコードで使用する構造体/共用体/列挙型を示します。

表 11-2 DEV_INFO_t構造体

メンバ名	内容
uint16_t　state;	State for application
uint16_t　devadr;	Device address

表 11-3 STATE_t列挙型

メンバ名	内容
STATE_WAIT	接続待ち
STATE_DRIVE	ドライブ認識中
STATE_READY	ドライブ接続中
STATE_WRITE	ファイルライト
STATE_READ	ファイルリード
STATE_COMPLETE	処理完了
STATE_ERROR	エラー発生

11.4　関数一覧

USB ホストドライバサンプルにて使用している関数一覧を以下に示します。

表 11-4　関数一覧

関数名	概要	スコープ	定義ファイル
main	メイン処理	local	main.c
port_init	ポート設定初期化	local	main.c
icu_init	割り込み設定	local	main.c
usbh_main	USB ホストメイン処理	local	r_usb_main.c
usb_cstd_PortInit	ポート設定初期化	local	r_usb_main.c
usb_cstd_IntInit	割り込み設定	local	r_usb_main.c
usb_cstd_IntEnable	割り込み処理有効	local	r_usb_main.c
usb_cstd_IntDisable	割り込み処理無効	local	r_usb_main.c
PowerOnUSBh	USB ホスト初期設定	local	r_usb_main.c
AhbPciBridgeInit	AHB-PCI Bridge PCI Communication Register 初期設定	local	r_usb_main.c
R_USBH_isr	USB ホスト割り込み処理	local	r_usb_main.c
msc_main	MSC メイン処理	global	r_usb_hmsc_apl.c
msc_drive	MSC ドライブ情報取得処理	global	r_usb_hmsc_apl.c
msc_data_ready	MSC ドライブ接続処理	global	r_usb_hmsc_apl.c
msc_data_write	MSC ドライブファイルライト処理	global	r_usb_hmsc_apl.c
msc_data_read	MSC ドライブファイルリード処理	global	r_usb_hmsc_apl.c
msc_configured	デバイスエニュメレーション完了時処理	global	r_usb_hmsc_apl.c
msc_detach	デタッチ処理	global	r_usb_hmsc_apl.c
msc_suspend	サスペンド処理	global	r_usb_hmsc_apl.c
msc_resume	レジューム処理	global	r_usb_hmsc_apl.c
msc_drive_complete	ドライブ情報取得完了処理	global	r_usb_hmsc_apl.c
msc_init	サンプル APL 初期化処理	global	r_usb_hmsc_apl.c
msc_registration	クラスドライバ登録処理	global	r_usb_hmsc_apl.c

11.5　関数詳細

11.5.1　main

(1)　概要

サンプルプログラムメイン処理

(2)　C 言語形式

void main (void);

(3)　パラメータ

なし

(4)　機能

サンプルプログラムのメイン処理です。
下記の処理を行います。

- ECM 初期化
- 割り込み設定
- ポート設定初期化
- usbh_main()プログラムの呼び出し

(5)　戻り値

なし

11.5.2 port_init

(1) 概要

ポート設定初期化

(2) C 言語形式

void port_init (void);

(3) パラメータ

なし

(4) 機能

ポート設定の初期化を行います。

(5) 戻り値

なし

11.5.3 icu_init

(1) 概要

割り込み設定

(2) C 言語形式

void icu_init (void);

(3) パラメータ

なし

(4) 機能

割り込み処理を有効にします。

(5) 戻り値

なし

11.5.4　usbh_main

(1)　概要

USB ホストメイン処理

(2)　C 言語形式

void usbh_main (void);

(3)　パラメータ

なし

(4)　機能

USB ホストドライバのメイン処理です。
USB ドライバの初期設定を行います。

- USB-BASIC-F/W の API 関数（R_usb_hstd_MgrOpen()）を呼び出し、HCD タスクと MGR タスクを登録する。
- HUB クラスドライバ API 関数（R_usb_hhub_Registration()）を呼び出し、HUB タスクを登録する。
- クラスドライバ登録用構造体（USB_HCDREG_t）の各メンバに情報を設定後、USB-BASIC-F/W の API 関数（R_usb_hstd_DriverRegistration()を呼び出し、クラスドライバを登録する。
- HMSC クラスドライバの API 関数（R_usb_hmsc_driver_start()）を呼び出し、HMSC タスクと HSTRG タスクを登録する。

初期設定後、メインルーチン内にて定期的にイベントの取得とそのイベントに対する処理を行います。

(5)　戻り値

なし

11.5.5 usb_cstd_PortInit

(1) 概要

ポート設定初期化

(2) C 言語形式

void usb_cstd_PortInit (void);

(3) パラメータ

なし

(4) 機能

ポート設定の初期化を行います。

(5) 戻り値

なし

11.5.6 usb_cstd_IntInit

(1) 概要

割り込み設定

(2) C 言語形式

void usb_cstd_IntInit (void);

(3) パラメータ

なし

(4) 機能

USB ホストの割り込み処理を設定します。

(5) 戻り値

なし

11.5.7　usb_cstd_IntEnable

(1)　概要

USB ホスト割り込み処理有効

(2)　C 言語形式

void usb_cstd_IntEnable (void);

(3)　パラメータ

なし

(4)　機能

USB ホストの割り込み処理を有効にします。

(5)　戻り値

なし

11.5.8 usb_cstd_IntDisable

(1) 概要

USB ホスト割り込み処理無効

(2) C 言語形式

void usb_cstd_IntDisable (void);

(3) パラメータ

なし

(4) 機能

USB ホストの割り込み処理を無効にします。

(5) 戻り値

なし

11.5.9　PowerOnUSBh

(1)　概要

USB ホスト初期設定

(2)　C 言語形式

void PowerOnUSBh (void);

(3)　パラメータ

なし

(4)　機能

下記の USB ホストの初期設定を行います。
・I / O 設定（P66 を VBUSEN、P77 を OVERCUR にそれぞれ設定）
・CPG 設定（USB クロックオン、USB リセット解除）
・AHB-PCI ブリッジ設定
・PHY 内蔵 PLL スタートアップ

(5)　戻り値

なし

11.5.10 AhbPciBridgeInit

(1) 概要

AHB-PCI Bridge PCI Communication Register 初期設定

(2) C 言語形式

void AhbPciBridgeInit (void);

(3) パラメータ

なし

(4) 機能

AHB-PCI ブリッジ設定を行います。

(5) 戻り値

なし

11.5.11 R_USBH_isr

(1) 概要

USB ホスト割り込み処理

(2) C 言語形式

void R_USBH_isr(void);

(3) パラメータ

なし

(4) 機能

USB ホストの割り込み処理が有効に設定されている場合に
EHCI と OHCI の割り込みハンドラを呼び出します。

(5) 戻り値

なし

11.5.12　msc_main

(1)　概要

MSC メイン処理

(2)　C 言語形式

void msc_main(void);

(3)　パラメータ

なし

(4)　機能

MSC のメイン処理を行います。
ステート遷移により管理を行いステートに応じた処理を行います。
「11.6.1 アプリケーション（APL）」にフローチャートを記載しています。

(5)　戻り値

なし

11.5.13　msc_drive

(1)　概要

MSC ドライブ情報取得処理

(2)　C 言語形式

uint16_t msc_drive(uint16_t drvno)

(3)　パラメータ

I/O	パラメータ		説明
I	int16_t	drvno	ドライブ番号

(4)　機能

ステートが STATE_DRIVE 時に呼び出される処理となります。
処理概要については「11.6.2 ステートの管理」を参照してください。

(5)　戻り値

戻り値	意味
USB_TRUE	成功
USB_FALSE	失敗

11.5.14　msc_data_ready

(1)　概要

MSC ドライブ接続処理

(2)　C 言語形式

uint16_t msc_data_ready(uint16_t drvno)

(3)　パラメータ

I/O	パラメータ		説明
I	int16_t	drvno	ドライブ番号

(4)　機能

ステートが STATE_READY 時に呼び出される処理となります。
処理概要については「11.6.2 ステートの管理」を参照してください。

(5)　戻り値

戻り値	意味
USB_TRUE	成功
USB_FALSE	失敗

11.5.15　msc_data_write

(1)　概要

MSC ドライブファイルライト処理

(2)　C 言語形式

uint16_t msc_data_write(uint16_t drvno)

(3)　パラメータ

I/O	パラメータ		説明
I	int16_t	drvno	ドライブ番号

(4)　機能

ステートが STATE_WRITE 時に呼び出される処理となります。
処理概要については「11.6.2 ステートの管理」を参照してください。

(5)　戻り値

戻り値	意味
USB_TRUE	成功
USB_FALSE	失敗

11.5.16　msc_data_read

(1)　概要

MSC ドライブファイルリード処理

(2)　C 言語形式

uint16_t msc_data_read(uint16_t drvno)

(3)　パラメータ

I/O	パラメータ		説明
I	int16_t	drvno	ドライブ番号

(4)　機能

ステートが STATE_READ 時に呼び出される処理となります。
処理概要については「11.6.2 ステートの管理」を参照してください。

(5)　戻り値

戻り値	意味
USB_TRUE	成功
USB_FALSE	失敗

11.5.17　msc_configured

(1)　概要

デバイスエニュメレーション完了時処理

(2)　C 言語形式

void msc_configured(uint16_t devadr)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　devadr	ドライバアドレス

(4)　機能

MSC デバイスが接続されてエニュメレーションが完了すると呼ばれるコールバック関数となります。
ドライブオープン関数 R_usb_hmsc_StrgDriveOpen()をコールし、ステートを STATE_DRIVE に変更します。

(5)　戻り値

なし

11.5.18　msc_detach

(1)　概要

デタッチ処理

(2)　C 言語形式

void msc_detach(uint16_t devadr)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　devadr	デバイスアドレス

(4)　機能

USB デバイス切断時に呼ばれるコールバック関数になります。
変数の初期化、ドライブ接続状態の解除およびステートを STATE_WAIT に変更する処理を行います。

(5)　戻り値

なし

11.5.19　msc_suspend

(1)　概要

サスペンド処理

(2)　C 言語形式

void msc_suspend(uint16_t devaddr)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　devaaddr	ドライブアドレス

(4)　機能

USB デバイスがサスペンド状態に移行時に呼ばれるコールバック関数になります。
サンプルでは特に処理はありません。

(5)　戻り値

なし

11.5.20　msc_resume

(1)　概要

レジューム処理

(2)　C 言語形式

void msc_resume(uint16_t devaddr)

(3)　パラメータ

I/O	パラメータ	説明
I	uint16_t　devaaddr	ドライブアドレス

(4)　機能

USB デバイスがサスペンド状態解除時に呼ばれるコールバック関数になります。
サンプルでは特に処理はありません。

(5)　戻り値

なし

11.5.21　msc_drive_complete

(1)　概要

ドライブ情報取得完了処理

(2)　C 言語形式

void msc_drive_complete(USB_UTR_t *utr)

(3)　パラメータ

I/O	パラメータ	説明
I	USB_UTR_t　*utr	構造体ポインタ

(4)　機能

ステートが STATE_DRIVE 時、ドライブ情報取得処理が完了すると呼ばれるコールバック関数となります。
ステートを STATE_READY に変更します。

(5)　戻り値

なし

11.5.22 msc_init

(1) 概要

サンプル APL 初期化処理

(2) C 言語形式

void msc_init(void)

(3) パラメータ

なし

(4) 機能

サンプルアプリケーションの初期化処理を行います。
ステートを STATE_WAIT に設定します。

(5) 戻り値

なし

11.5.23　msc_registration

(1)　概要

クラスドライバ登録処理

(2)　C 言語形式

void msc_registration(void)

(3)　パラメータ

なし

(4)　機能

クラスドライバ登録用構造体（USB_HCDREG_t）の各メンバに情報を設定後、USB-BASIC-F/W の API 関数（R_usb_hstd_DriverRegistration()を呼び出し、クラスドライバを登録する。

(5)　戻り値

なし

11.6　フローチャート

11.6.1　アプリケーション（APL）

APL は、ステート遷移により管理を行っています。
表 11-5 にステート一覧を示します。

表 11-5 ステート一覧

ステート	概要
STATE_WAIT	接続待ち
STATE_COMPLETE	デタッチ処理
STATE_ERROR	エラー発生
STATE_DRIVE	ドライブ認識
STATE_READY	ドライブ接続中
STATE_WRITE	ファイルライト
STATE_READ	ファイルリード

図 11-1 に APL の処理フローチャートを示します。

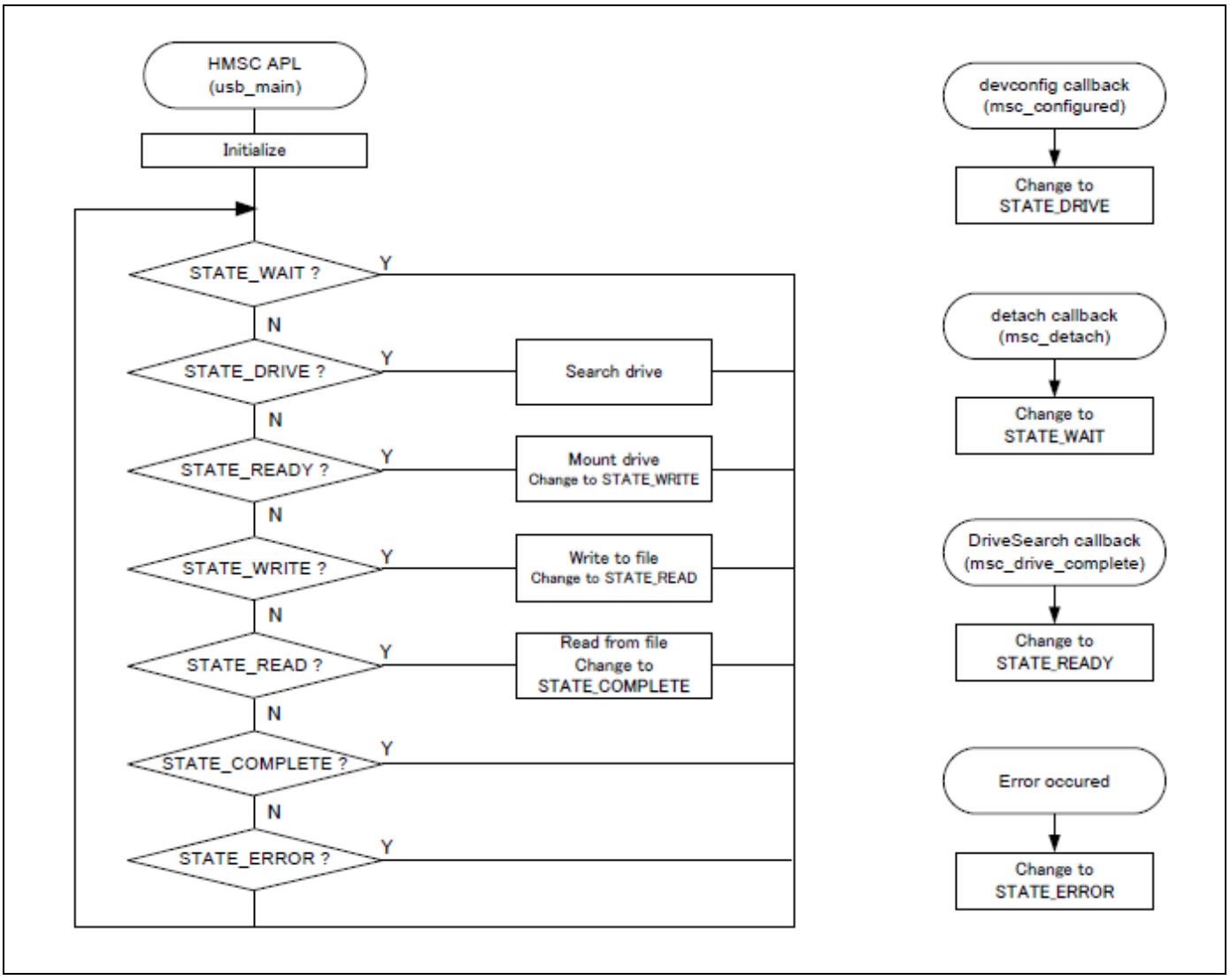


図 11-1 サンプルコードのメイン処理

11.6.2　ステートの管理

以下にステートごとの処理概要を示します。

(1)　接続待ち（STATE_WAIT）

このステートでは、MSC デバイスが接続されるのを待ちます。エニュメレーションが完了すると、ステートを STATE_DRIVE に変更します。

① 初期化関数がステートを STATE_WAIT にします。
② MSC デバイスが接続されるまで STATE_WAIT 状態を継続します。
③ MSC デバイスが接続されてエニュメレーションが完了すると、USB_HCDREG_t 構造体のメンバ devconfig に設定されたコールバック関数 msc_configured()が USB ドライバよりコールされます。
④ ステートを STATE_DRIVE に変更します。

(2)　ドライブ情報取得（STATE_DRIVE）

このステートでは、接続された MSC デバイスのドライブ情報取得処理をして、ステートを STATE_READY に変更します。

① ドライブ認識中フラグ変数（drive_search_lock）を確認して、オフであれば処理を開始します。
② drive_search_lock をオンにします。
③ R_usb_hmsc_StrgDriveSearch()をコールして、MSC デバイスにクラスリクエスト GetMaxLUN とストレージコマンドを送信し、ドライブ情報取得処理を行います。
④ ドライブ情報取得処理が完了すると、R_usb_hmsc_StrgDriveSearch()に登録したコールバック関数 msc_drive_complete()がコールされます。
⑤ ステートを STATE_READY に変更します。

(3)　ドライブ接続（STATE_READY）

このステートでは、認識したドライブのマウント処理をして、ステートを STATE_WRITE に変更します。

① 認識したドライブ番号から f_mount()をコールして接続します。
② ステートを STATE_WRITE に変更します。

(4)　ファイルライト（STATE_WRITE）

このステートでは、接続したドライブにファイルライト処理をして、ステートを STATE_READ に変更します。

① f_open()をコールして、ファイル作成＋書き込みモードでファイルオープンします。
② f_write()をコールして、全て’a’である 512 バイトのファイル hmscdemX.txt を作成します。ファイル名中の X はドライブ番号に対応します。例えば、ドライブ 1 の場合は hmscdem1.txt になります。
③ f_close()をコールして、ファイルクローズします。
④ ステートを STATE_READ に変更します。

(5)　ファイルリード（STATE_READ）

このステートでは、接続したドライブにファイルリード処理をして、ステートを STATE_COMPLETE に変更します。

① f_open()をコールして、ファイル作成＋読み出しモードでファイルオープンします。
② f_read()をコールして、ファイル hmscdemX.txt を読み出します。
③ 全て’a’の 512 バイトのデータか確認します。
④ f_close()をコールして、ファイルクローズします。
⑤ ドライブ番号に対応した LED を点灯します。
⑥ ステートを STATE_COMPLETE に変更します。

(6)　処理完了（STATE_COMPLETE）

サンプルアプリケーションの処理が正常終了したとき、このステートになります。

(7)　エラー終了（STATE_EERROR）

サンプルアプリケーションの処理が異常終了したとき、このステートになります。

(8)　デタッチ処理

接続された MSC デバイスが切断されると USB ドライバよりコールバック関数 msc_detach()がコールされます。このコールバック関数では、変数の初期化、ドライブ接続状態の解除およびステートを STATE_WAIT に変更する処理を行います。なお、コールバック関数 msc_detach()は、USB_HCDREG_t 構造体のメンバ devdetach に設定した関数です。

11.7　チュートリアル

11.7.1　動作概要

サンプルアプリケーションの主な機能を以下に示します。

1. MSC デバイスが接続されるとエニュメレーション処理をします。
2. MSC デバイスのエニュメレーションが完了するとドライブ情報取得処理をします。
3. MSC デバイスに 512 バイトのサイズのファイルを書き込みます。
4. MSC デバイスに書き込んだファイルの読み込みをします。
5. ファイル内容を比較して一致した場合に、ドライブ番号に対応した LED を点灯します。

図 11-2 にシステムブロック図を示します。

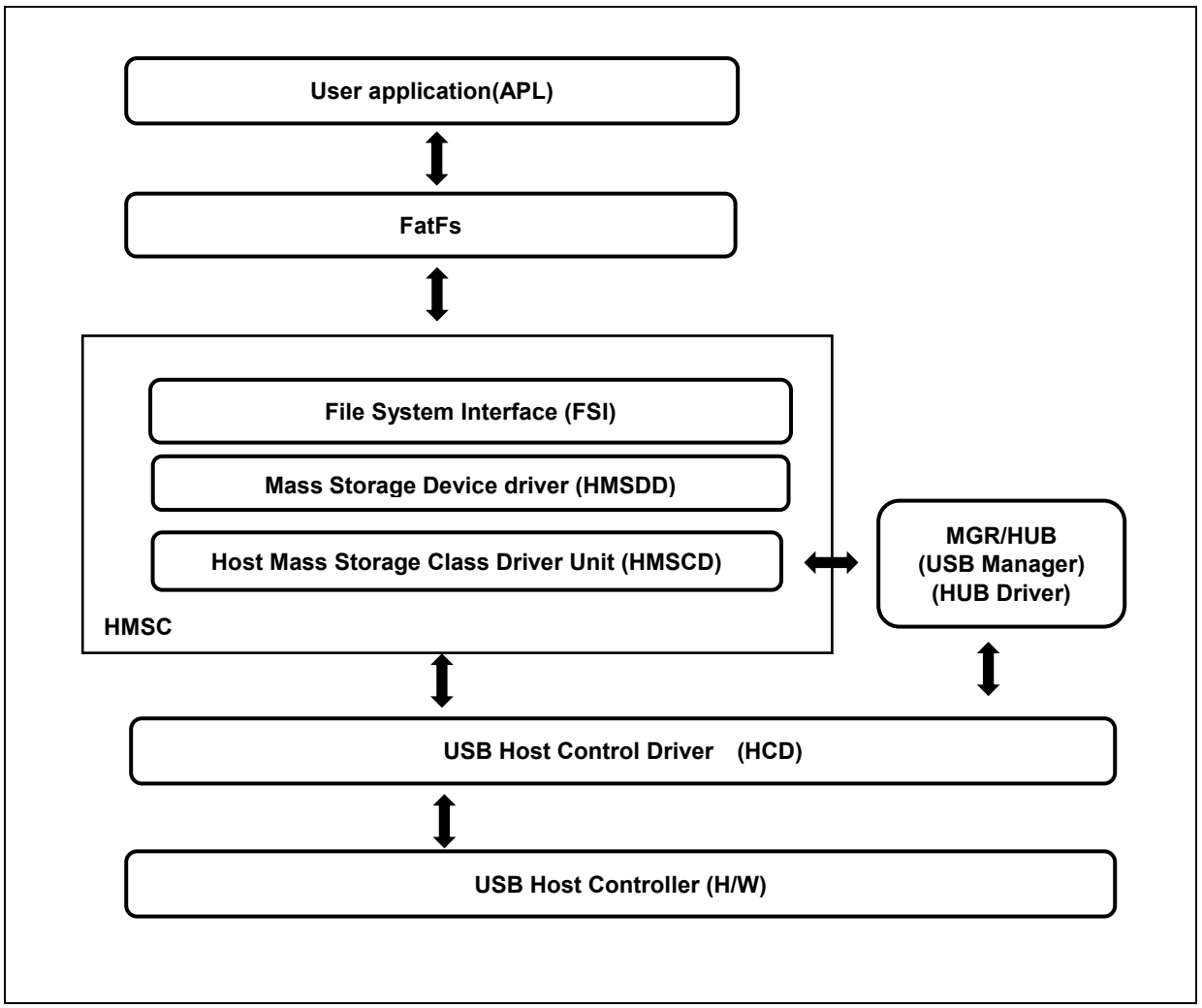


図 11-2 システムブロック図

11.7.2　使用準備

図 11-3 に HMSC の動作環境例を示します。

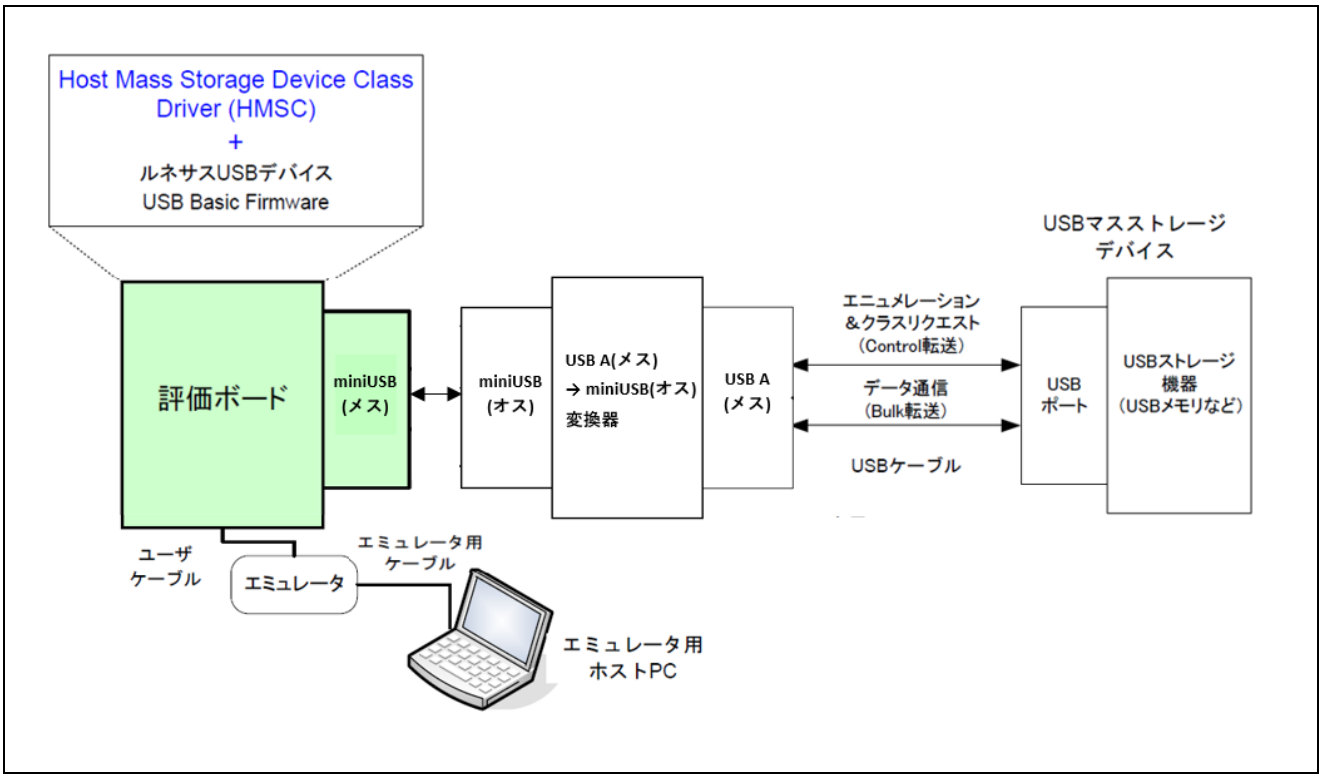


図 11-3 接続方法例

USB ホストサンプルプログラムを動作させるには基盤の改修が必要となります。

(1) USB ホストでは A16 端子を過電流検出端子として使用しますが、本サンプルプログラムでは過電流検出機能は動作させないため、3.3V と A16 端子を接続。

(2) USB ホストとしてデバイスへの給電を行うために 5.0V と VBUS 端子を接続。

図 11-4 に基盤改修内容を示します。

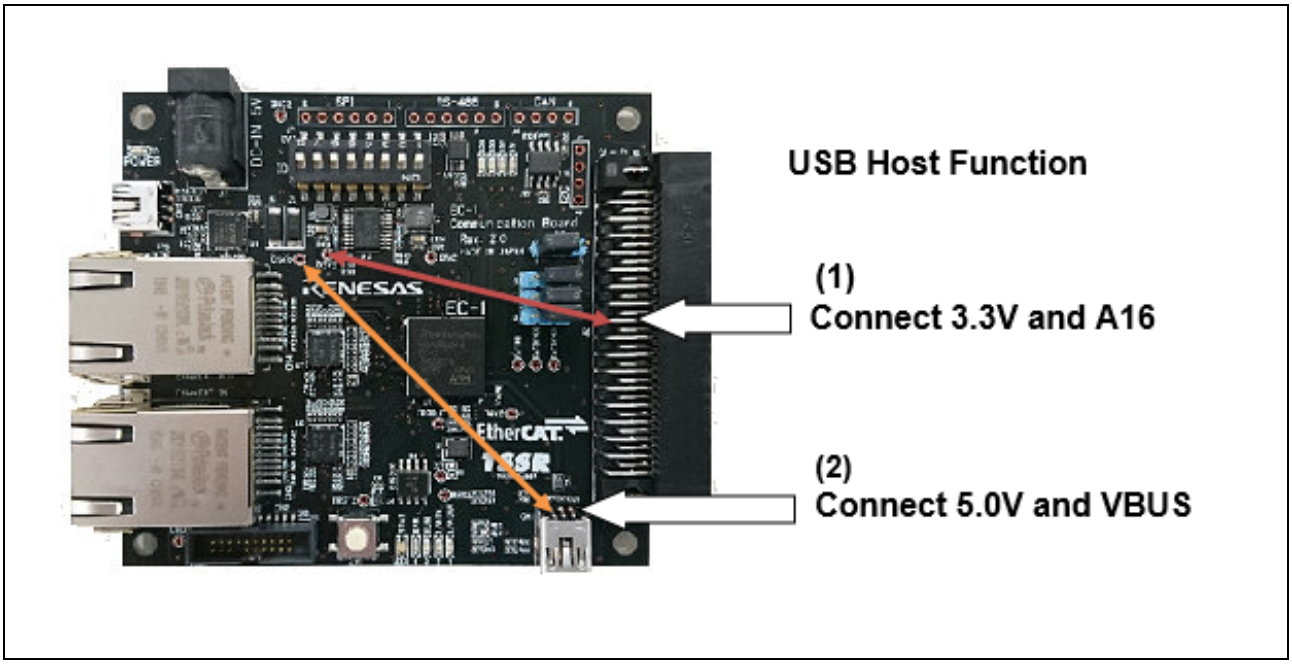


図 11-4 USBホストサンプルプログラムを動作させるための基盤改修内容

11.7.3　サンプルプログラム設定

初期設定例を以下に示します。

```
void usb_hmsc_main(void)
{
  /* MCU の設定参照 */
  usb_mcu_setting();

  /* USB ドライバの設定参照 */
  R_usb_hstd_MgrOpen();
  R_usb_cstd_SetTaskPri(USB_HUB_TSK, USB_PRI_3); // (注)
  R_usb_hhub_Registration(USB_NULL); // (注)
  msc_registration();
  R_usb_hmsc_driver_start();

  /* メインルーチン参照 */
  usb_hapl__mainloop();
}
```

(注) HUB を使用する場合のみ、本関数を呼び出す必要があります。

MCU 設定

USB モジュールをハードウェアマニュアルの初期設定シーケンスに従って設定し、USB 割り込みハンドラの登録と USB 割り込み許可設定をしています。

USB ドライバ設定

USB ドライバの設定では、スケジューラへのタスク登録及び USB-BASIC-F/W に対するクラスドライバの情報登録を行います。以下に、手順を示します。

1. USB-BASIC-F/W の API 関数（R_usb_hstd_MgrOpen()）を呼び出し、HCD タスクと MGR タスクを登録する。
2. HUB クラスドライバ API 関数（R_usb_hhub_Registration()）を呼び出し、HUB タスクを登録する。
3. クラスドライバ登録用構造体（USB_HCDREG_t）の各メンバに情報を設定後、USB-BASIC-F/W の API 関数（R_usb_hstd_DriverRegistration()を呼び出し、クラスドライバを登録する。
4. HMSC クラスドライバの API 関数（R_usb_hmsc_driver_start()）を呼び出し、HMSC タスクと HSTRG タスクを登録する。

USB_HCDREG_t で宣言された構造体に設定する情報例を以下に示します。

```
void msc_registration(void)
{
  /* クラスドライバ登録用構造体 */
  USB_HCDREG_t driver;
  /* USB の規格で定められたクラスコードを設定 */
  driver.ifclass = (uint16_t)USB_IFCLS_MSC;
  /* ターゲットペリフェラルリストを設定 */
  driver.tpl = (uint16_t*)&usb_gapl_devicetpl; (注 1)
  /* エニュメレーション中に行われるクラスチェック関数を設定 */
  driver.classcheck = &R_usb_hmsc_class_check;
  /* エニュメレーション完了時に呼び出される関数を設定 */
  driver.devconfig = &msc_configured;
  /* USB デバイス切断時に呼ばれる関数を設定 */
```

```
        driver.devdetach = &msc_detach;
        /* デバイスをサスペンド状態に移行時に呼ばれる関数を設定 */
        driver.devsuspend = &msc_suspend;
        /* デバイスのサスペンド状態解除時に呼ばれる関数を設定 */
        driver.devresume = &msc_resume;
        /* HCD へクラスドライバ情報を登録 */
        R_usb_hstd_DriverRegistration(&driver);
    }
```

（注 1） TPL(Target Peripheral List)はアプリケーション内で定義してください。
詳細は「11.8.6 ターゲットペリフェラルリスト（TPL）」を参照してください。

メインルーチン

USB ドライバは初期設定後アプリケーションのメインルーチン内でスケジューラ（R_usb_cstd_Scheduler()）を呼び出すことで動作します。

メインルーチン内で R_usb_cstd_Scheduler()を呼ぶことでイベントの有無を確認し、イベントがある場合、スケジューラにイベントが発生していることを通知するためのフラグをセットします。

R_usb_cstd_Scheduler()呼び出し後、R_usb_cstd_CheckSchedule()を呼び出しイベントの有無を確認してください。また、イベントの取得とそのイベントに対する処理は定期的に行う必要があります。（注 1）

```
void usb_hapl_mainloop(void)
{
    while(1) /* メインルーチン */
    {
        /* イベントの確認と取得、フラグセット (注 1) */
        R_usb_cstd_Scheduler();
        /*イベント有無の判定、フラグクリア */
        if(USB_FLGSET == R_usb_cstd_CheckSchedule())
        {
            R_usb_hstd_MgrTask(); /* MGR タスク */
            R_usb_hhub_Task(); /* HUB タスク (注 3) */
            R_usb_hmsc_task(); /* HMSC タスク */
            R_usb_StrgDriveTask(); /* HSTRG タスク */
        }
        msc_main(); /* APL */
    }
}
```

(注 2)

（注 1） R_usb_cstd_Scheduler()でイベントを取得後、処理を行う前に再度 R_usb_cstd_Scheduler()で他のイベントを取得すると、最初のイベントは破棄されます。イベント取得後は必ず各タスクを呼び出し、処理を行ってください。

（注 2） これらの処理は、アプリケーションプログラムのメインループ内に必ず記述してください。

（注 3） HUB を使用する場合のみ、本関数を呼び出す必要があります。

11.7.4　FatFs 組み込み

本サンプルプログラムをビルドするには、お客様にて FatFs を組み込む必要があります。
FatFs の組み込み手順を以下に示します。

FatFs は、下記の URL でオープンソースとして公開されています。
http://elm-chan.org/fsw/ff/00index_e.html

1. ページを下へスクロールすると、ダウンロード用のリンクが「Resources」セクション内に見つかります。
2. 「FatFs R0.13」をクリックして FatFs をダウンロードし、任意のフォルダへ保存して下さい。

本サンプルプログラムは、バージョン R0.13 用に作成されています。もし、FatFs が更新されている場合には、「Old release」リンク内からこのバージョンを探して下さい。

ダウンロードした FatFs の ZIP ファイル(ff13.zip)を展開し、図 11-5 に示す通りにサンプルプログラムのワークスペース内へ移動して下さい。
サンプルプログラムは、このフォルダ構成以外では動作しませんので、ご注意下さい。

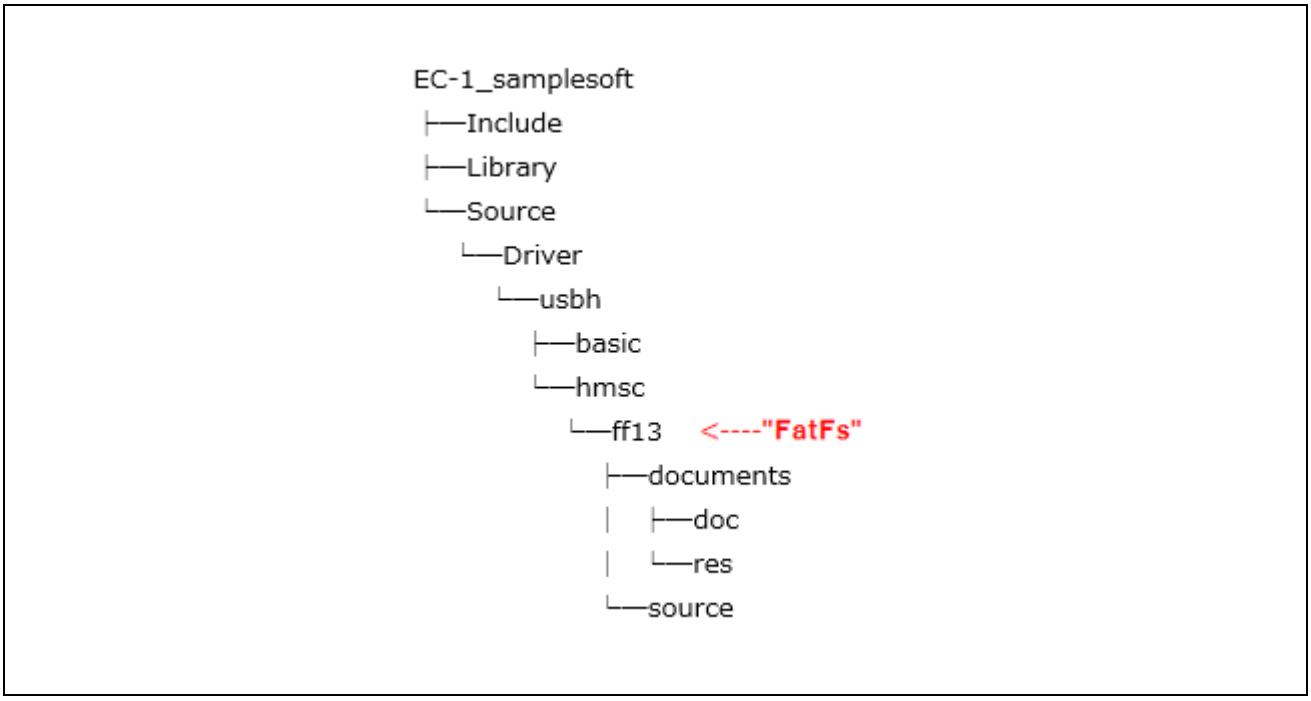


図 11-5 FatFsを展開するフォルダ

サンプルプログラムのワークスペースを開き、サンプルプロジェクトをビルドして下さい。
サンプルプログラムは、FatFs をリンクするように設定されているため、設定を変更する必要はありません。
お客様の製品へ FatFs を組み込む場合には、FatFs のライセンスをご確認いただき、お客様の責任にて実施して下さい。

11.8　USB-BASIC-F/W

11.8.1　スケジューラ機能

本モジュールは、スケジューラ機能を使用して各タスクや H/W の要求をタスクの優先順位にしたがって管理します。また優先順位が同じタスクに複数の要求が発生した場合は FIFO 構造で要求を実行します。
タスク間の要求はメッセージの送受信で実現しています。また要求の終了はコールバック関数によりタスクに応答されるためスケジューラ本体を改変することなくユーザシステムに適合したクラスドライバの実装が可能です。
スケジューラ機能の詳細は、「11.8.10　non-OS スケジューラ」を参照してください。

11.8.2　ホストコントロールドライバ（HCD）

・基本機能

HCD は、H/W 制御用のプログラムです。HCD の機能を以下に示します。

1. Control 転送（ControlRead/ControlWrite/No-dataControl）および結果通知
2. Data 転送（Bulk /Interrupt/Isochronus）および結果通知
3. データ転送の強制終了（全パイプ）
4. USB 通信エラー判定および転送リトライ
5. USB バスリセット信号送出およびリセットハンドシェイク結果通知
6. サスペンド信号/レジューム信号送出
7. 割り込みによるアタッチ/デタッチ検出

・HCD に対する要求発行

HCD に対して H/W 制御要求を発行する場合は、API 関数を用います。

なお、接続されているデバイスステートの更新要求は、HCD が直接制御する場合と USB ハブ経由で制御する場合があります。MGR がデバイスアドレスを判断し制御要求を HCD / HUBCD タスクに発行します。

HCD は上位タスクからの要求に対して、コールバック関数を用いて結果を通知します。

・複数のデバイスを接続する場合

接続された HUB のデバイスアドレスを”1”としてエニュメレーションします。また、HUBCD は、ダウンポートに接続されたデバイスに対して”2”以降のデバイスアドレスを自動的に割り振ります。

複数のデバイスとエニュメレーション及び通信が可能ですが、複数のデバイスに対し同時にエニュメレーションをしません。この場合、最初に接続されたデバイスのエニュメレーションが終了してから次に接続されたデバイスのエニュメレーションを開始します。

また、USB バス占有率の計算はしません。

11.8.3　ホストデバイスクラスコントロールドライバ（HDCD）

・登録

ユーザシステムに合わせた HDCD を作成する必要があります。

作成した HDCD の各種情報を API 関数 R_usb_hstd_DriverRegistration()を使って USB_HCDREG_t 構造体に登録してください。

登録方法の詳細は、3.14.3 章を参照してください。

・タスク設定

HDCD がスケジューラを使用する場合は、「11.8.10　non-OS スケジューラ」を参照してください。

r_usb_basic_config.h ファイルに、追加するタスク毎にタスク ID、メールボックス ID、メモリプール ID を追加してください。

タスク優先度設定は、API 関数 R_usb_cstd_SetTaskPri()を使って設定してください。

これらの項目設定時には、以下の点に注意してください。

・ タスク ID は続き番号を使用し、重複しないように設定してください。

・ メールボックス ID とメモリプール ID は、タスク ID と同じ値を設定してください。

・ 追加するタスクの優先度は、HCD(1), MGR(2), HUB(3)より低く(4～7)設定してください。優先度は 0 が最高で、7 が最低です。

以下に、r_usb_basic_config.h の設定追加例を示します。

#define USB_SMP_TSK USB_TID_6　：タスク ID

#define USB_SMP_MBX USB_SMP_TSK　：メールボックス ID

#define USB_SMP_MPL USB_SMP_TSK　：メモリプール ID

必要がある場合は、r_usb_basic_config.h ファイルで、以下の項目を設定してください。

詳細は「3.2.11　USB ホスト制御」を参照ください。

・タスク ID の最大値設定

・メモリプールの最大数設定

・メッセージプールの最大数設定

・クラスチェック処理

クラスチェックコールバック関数でエニュメレーション中にクラスチェック処理をする必要があります。

処理完了時に API 関数 R_usb_hstd_ReturnEnuMGR()を使って結果を通知してください。

11.8.4　ホストマネージャ（MGR）

・基本機能

MGR は、HCD と HDCD 間の機能を補完するタスクです。MGR の機能を以下に示します。

1. HDCD の登録
2. 接続されたデバイスの状態管理
3. 接続されたデバイスのエニュメレーション
4. ディスクリプタからエンドポイント情報の検索

・USB 標準リクエスト

MGR は、接続されたデバイスに対してエニュメレーションをします。

MGR が発行する USB 標準リクエストを以下に示します。また、デバイスから取得したディスクリプタ情報は内部保存され、API 関数にてパイプ情報として登録することができます。

GET_DESCRIPTOR（Device Descriptor）

SET_ADDRESS

GET_DESCRIPTOR（Configuration Descriptor）

SET_CONFIGURATION

・クラスチェック

MGR は、エニュメレーション時に GET_DESCRIPTOR リクエストで取得した情報を HDCD に通知し、接続されたデバイスが動作可能であるか確認します。

HDCD からの API 関数 R_usb_hstd_ReturnEnuMGR()での応答によりエニュメレーションを継続します。

11.8.5　HUB クラスドライバ（HUBCD）

・基本機能

HUBCD は、接続された USB ハブのダウンポートの状態を管理し、HCD と HDCD 間の機能を補完します。

HUBCD の機能を以下に示します。

1. USB ハブのエニュメレーション
2. USB ハブのダウンポートに接続されたデバイスの状態管理
3. USB ハブのダウンポートに接続されたデバイスのエニュメレーション

・ダウンポートの状態管理

HUBCD は USB ハブが接続されるとすべてのダウンポートに対して

1) ポートパワー許可（HubPortSetFeature：USB_HUB_PORT_POWER）
2) ポートの初期化（HubPortClrFeature：USB_HUB_C_PORT_CONNECTION）
3) ポートステータスの取得（HubPortStatus：USB_HUB_PORT_CONNECTION）

をし、ダウンポートのデバイス接続状況を確認します。

・ダウンポートへのデバイス接続

HUBCD は USB ハブからダウンポートのデバイスアタッチの通知を受けると、USB ハブに対して USB リセット信号要求を発行（HubPortSetFeature：USB_HUB_PORT_RESET）します。

その後、MGR タスクの資源を使用して、接続されたデバイスとエニュメレーションをします。

HUBCD はリセットハンドシェイクの結果を保存し、接続されたデバイスに対して使用していないデバイスアドレスを順次割り振ります。

・クラスリクエスト

HUBCD がサポートしているクラスリクエストを表 11-6 に示します。

表 11-6 USBハブクラスリクエスト

リクエスト	実装状況	関数名	機能
ClearHubFeature	×		
ClearPortFeature	○	usb_hhub_PortClrFeature	USB_HUB_C_PORT_CONNECTION USB_HUB_C_PORT_RESET
ClearTTBuffer	×		
GetHubDescriptor	○	usb_hhub_GetHubDescriptor	ハブディスクリプタ取得
GetHubStatus	×		
GetPortStatus	○	usb_hhub_GetStatus	ポートステータス取得
ResetTT	×		
SetHubDescriptor	×		
SetHubFeature	×		
SetPortFeature	○	usb_hhub_PortSetFeature	USB_HUB_PORT_POWER USB_HUB_PORT_RESET
GetTTState	×		
StopTT	×		

11.8.6　ターゲットペリフェラルリスト（TPL）

USB-BASIC-F/W は、サポートする USB デバイスを指定することができます。
サポートする USB デバイスの指定は、ターゲットペリフェラルリスト(TPL)に、その USB デバイスのプロダクト ID およびベンダ ID をセットで指定してください。
なお、サポートする USB デバイスを特定する必要が無い場合は、ベンダ ID に USB_NOVENDOR、プロダクト ID に USB_NOPRODUCT を設定してください。
以下に、TPL の作成例を示します。

```
const uint16_t usb_gapl_devicetpl[] =
{
    2, /* Number of list */
    0, /* Reserved */
    0xXXXX, /* Vendor ID */
    0xXXXX, /* Product ID */
    0xYYYY, /* Vendor ID */
    0xYYYY, /* Product ID */
};
```

[Note]

1. TPL は、アプリケーションプログラム用および Hub 用(r_usb_hhubsys.c)の 2 つが用意されています。コンプライアンステスト対応設定を選択している時は、この 2 つの TPL には、USB_NOVENDOR と USB_NOPRODUCT を設定しないようにしてください。(コンプライアンス対応設定については、「3.2.11　USB ホスト制御」を参照してください。)

2. コンプライアンステスト対応設定を選択している時、Hub 用 TPL には、コンプライアンステストで使用する Hub の VENDOR ID と PRODUCT ID を設定してください。

11.8.7 API 関数

MGR、HUBCD および HDCD は、API 関数で H/W の制御要求をします。
ANSI C99 を使用しています。これらの型は stdint.h で定義されています。
本ドライバの API はルネサスの API の命名基準に従っています。
すべての API 呼び出しとそれをサポートするインターフェース定義は r_usb_basic_if.h に記載しています。
別章「表 3-73」に API 関数一覧を示します。

11.8.8　コールバック関数

コールバック関数を使用することで、イベントをポーリングする必要がなくなります。
例えば、ユーザアプリケーションが HCD に対してデータ転送要求をする場合、コールバック関数を登録することで、デバイスとのデータ通信を行った結果を受け取ります。
以下に、コールバック関数を示します。

表 11-7 R_usb_hstd_TransferStartコールバック

名称	データ転送要求のコールバック関数		
呼出形式	void (*USB_UTR_CB_t)(USB_UTR_t*)		
引数	USB_UTR_t*		R_usb_hstd_TransferStart()関数の引数のUSB_UTR_t ポインタ
戻り値	ー	ー	ー
説明	データ転送終了（アプリケーションから指定されたサイズのデータ転送終了およびショートパケット受信等によるトランスファー終了）でコールされます。		
注意事項			

表 11-8 classcheckコールバック

名称	classcheck コールバック関数		
呼出形式	void (*USB_CB_CHECK_t)(uint16_t **)		
引数	uint16_t	**table	table[0]デバイスディスクリプタ先頭アドレス table[1]コンフィグレーションディスクリプタ先頭アドレス table[2]インタフェースディスクリプタ先頭アドレス table[3]ディスクリプタチェック結果 table[4]HUB スペック table[5]ポート番号(未使用) table[6]通信速度(未使用) table[7]デバイスアドレス
戻り値	ー	ー	ー
説明	レジストレーションで登録したクラスコード（USB_HCDREG_t 構造体のメンバ ifclass）と受信したディスクリプタのクラスコードが一致した場合に、USB_HCDREG_t 構造体のメンバ classcheck に登録した関数がコールされます。 table[3]：チェック結果 USB_OK　：HDCD 動作可能 USB_ERROR：HDCD 動作不能 のいずれかでHDCD が動作可能か否かを応答してください。 table[4]：HUB スペック(HUB ドライバ用) USB_FSHUB：Full-Speed HUB の場合 USB_HSHUBS：Hi-Speed HUB(Single)の場合 USB_HSHUBM：Hi-Speed HUB(Multi)の場合 table[7]：デバイスアドレス Host がSet Address で割り当てたデバイスアドレス		
注意事項			

表 11-9 State Changeコールバック

名称	StateChange コールバック関数		
呼出形式	void (*USB_CB_t)(uint16_t)		
引数	uint16_t	devaddr	接続されたデバイスのアドレス
戻り値	ー	ー	ー
説明	StateChange が発生した場合に、レジストレーションで USB_HCDREG_t 構造体のメンバに登録された関数がコールされます。 devconfig：Set_Configuration リクエスト発行時に実行されます。 devdetach：デタッチ検出時に実行されます。 devsuspend：サスペンド遷移時に実行されます。 devresume：レジューム遷移時に実行されます。		
注意事項			

11.8.9　USB 通信

(1)　パイプ

パイプは、USB デバイスのエンドポイントとの論理的な接続路です。
コントロール転送をするデフォルトパイプ（パイプ番号 0）は、USB-BASIC-F/W 内部で設定されます。
データ転送をするためには、クラスチェック処理時に API 関数の R_usb_hstd_SetPipe()をコールして USB デバイスのディスクリプタからパイプ情報を設定する必要があります。パイプ情報は、エンドポイントディスクリプタ毎にパイプ番号 1 から順に割り振られます。
パイプの最大数はコンフィグファイルにより変更できます。「3.2.11　USB ホスト制御」を参照してください。
パイプに設定する情報は、下記の通りです。
・ インターフェース番号
・ エンドポイント番号
・ エンドポイント転送タイプ
・ エンドポイント方向
・ マックスパケットサイズ
・ デバイスアドレス
データ転送要求には、パイプ番号を指定する必要があります。
API 関数の R_usb_hstd_GetPipeID()をコールすることでパイプ番号を取得できます。
デバイスが切断された場合には、パイプ情報領域を開放するためパイプ情報をクリアする必要があります。
API 関数の R_usb_hstd_ClearPipe()をコールすることでパイプ情報をクリアできます。

(2)　データ転送要求

API 関数の R_usb_hstd_TransferStart()の引数に転送情報を設定した USB_UTR_t 構造体を渡すことで、データ転送要求ができます。
USB_UTR_t 構造体の詳細は、「表 3-21 USB_UTR_t 構造体」を参照してください。

(3)　転送結果の通知

USB-BASIC-F/W は、データ転送が終了すると HDCD に対してデータ転送要求時に登録されたコールバック関数でデータ転送の終了を通知します。
転送結果は、USB_UTR_t 構造体のメンバ（result）に格納されます。
以下に、通信結果を示します。
USB_CTRL_END : Control 転送が正常に終了した場合
USB_DATA_NONE : データ送信が正常終了した場合
USB_DATA_OK : データ受信が正常終了した場合
USB_DATA_SHT : データ受信は正常終了したが指定されたデータ長未満で終了した場合
USB_DATA_OVR : 受信データがサイズオーバーした場合
USB_DATA_ERR : 無応答もしくはオーバ／アンダーランエラーを検出した場合
USB_DATA_STALL : STALL 応答もしくは MaxPacketSize エラーを検出した場合
USB_DATA_STOP : データ転送を強制終了した場合

(4)　転送のリトライ

USB-BASIC-F/W は、各パイプの無応答に対して通信再実行をします。無応答を検出した場合は HDCD に対して USB_DATA_ERR を応答します。

(5) 受信時の注意事項

ショートパケットを受信した場合は、残り受信予定のデータ長を USB_UTR_t 構造体の tranlen に格納して転送終了します。受信したデータがバッファサイズより大きい場合は、バッファサイズまでのデータを FIFO バッファから読み出して転送終了します。

(6) コントロール転送

USB-BASIC-F/W は「11.8.4 ホストマネージャ（MGR）」の USB 標準リクエストに記載されていないリクエスト（ベンダ、クラスリクエスト）をデバイスに送信する場合、USB_UTR_t 構造体のメンバに以下の設定をした後に、R_usb_hstd_TransferStart()を呼び出す必要があります。

- keyword ：パイプ番号（USB_PIPE0）
- tranadr ：データバッファ（データステージ）
- tranlen ：転送サイズ
- setup ：セットアップデータ
- complete ：コールバック関数

データ転送終了時に complete に設定したコールバック関数が呼び出され、転送結果が通知されます。
（「7.13.2 転送結果の通知」を参照してください）
以下に、ベンダリクエスト送信例を示します。

＜ベンダリクエスト送信例＞

```
uint8_t usb_gsmp_VendorRequestData[16];
USB_SETUP_t usb_gsmp_VendorRequest;
USB_UTR_t usb_gsmp_ControlUtr;
USB_UTR_CB_t usb_gsmp_VendorRequestCb;

USB_ER_t usb_hsmp_VendorRequestProcess(void)
{
    USBC_ER_t err;

    /* セットアップデータ設定 */
    usb_gsmp_VendorRequest.type = ((bRequest << 8) | bmRequestType);
    usb_gsmp_VendorRequest.value = wValue;
    usb_gsmp_VendorRequest.index = wIndex;
    usb_gsmp_VendorRequest.length = wLength;
    usb_gsmp_VendorRequest.devaddr = devaddr;

    /* 転送パイプ設定(デフォルトパイプ) */
    usb_gsmp_ControlUtr.keyword = USB_PIPE0;
    /* 送信データ設定*/
    usb_gsmp_ControlUtr.tranadr = usb_gsmp_VendorRequestData;
    /* 転送サイズ設定 */
    usb_gsmp_ControlUtr.tranlen = usb_gsmp_VendorRequest.length;
    /* Setup コマンド設定 */
    usb_gsmp_ControlUtr.setup = &usb_gsmp_VendorRequest;
    /* コールバック関数設定 */
    usb_gsmp_ControlUtr.complete = &usb_gsmp_VendorRequestCb;

    /* データ送信要求 */
    err = R_usb_hstd_TransferStart(&usb_gsmp_ControlUtr);

    return err;
}
```

(7)　データ転送

USB-BASIC-F/W は、USB_UTR_t 構造体のメンバに以下の設定をした後に、R_usb_hstd_TransferStart()を呼び出す必要があります。
・ keyword　：パイプ番号
・ tranadr　：データバッファ
・ tranlen　：転送サイズ
・ complete　：コールバック関数

データ転送終了時に complete に設定したコールバック関数が呼び出され、転送結果が通知されます。（「7.13.2 転送結果の通知」を参照してください）
以下に、BulkIn 転送例を示します。

＜BulkIn 転送例＞
```
uint8_t usb_gsmp_VendorBulkInData[512];
USB_UTR_t usb_gsmp_DataUtr;
USB_UTR_CB_t usb_gsmp_VendorBulkInCb;

USB_ER_t usb_hsmp_VendorBulkInProcess(uint16_t devaddr)
{
    USB_ER_t err;

    /* 転送パイプ設定 */
    usb_gsmp_DataUtr.keyword = R_usb_htsd_GetPipeID(devaddr, USB_EP_BULK, USB_EP_IN, 0);
    /* 送信データ設定*/
    usb_gsmp_DataUtr.tranadr = usb_gsmp_VendorBulkInData;
    /* 転送サイズ設定 */
    usb_gsmp_DataUtr.tranlen = 512;
    /* コールバック関数設定 */
    usb_gsmp_DataUtr.complete = &usb_gsmp_VendorBulkInCb;

    /* データ送信要求 */
    err = R_usb_hstd_TransferStart(&usb_gsmp_DataUtr);

    return err;
}
```

11.8.10　non-OS スケジューラ

(1)　概要

non-OS スケジューラは、タスク優先度に従いタスクスケジューリングをします。
non-OS スケジューラの特徴を以下に示します。
・ 各タスクや H/W の要求をタスクの優先順位に従って管理する。
・ 優先順位が同じタスクから複数の要求が発生した場合、FIFO 構造で要求を処理する。
・ 要求の終了はコールバック関数によりタスクに通知されるため、スケジューラを変更することなくユーザシステムに適合したクラスドライバの実装が可能。

(2)　non-OS スケジューラマクロ

ユーザが使用可能なスケジューラマクロを以下に示します。

表 11-10 スケジューラマクロ

スケジューラマクロ	登録関数	概要
R_USB_SND_MSG	R_usb_cstd_SndMsg	メッセージBOX を指定して、メッセージを送信します。
R_USB_WAI_MSG	R_usb_cstd_WaiMsg	スケジューラを指定回数実行後にUSB_SND_MSG を実行します。
R_USB_RCV_MSG	R_usb_cstd_RecMsg	指定したメールBOX にメッセージ受信があるか確認します。
R_USB_PGET_BLK	R_usb_cstd_PgetBlk	メッセージを格納する領域を確保します。
R_USB_REL_BLK	R_usb_cstd_RelBlk	メッセージを格納する領域を開放します。

(3)　non-OS スケジューラ API 関数

non-OS スケジューラで使用する API 関数一覧を別章「表 3-74」 に示します。

11.9　ホストマスストレージクラスドライバ（HMSC）

HMSC は、FSI と HMSDD と HMSCD で構成されており、USB-BASIC-F/W と結合しています。
本章では、HMSC の機能について示します。

11.9.1　クラスリクエスト

HMSC がサポートするクラスリクエストを示します。

表 11-11 クラスリクエスト

リクエスト	コード	説明	対応
Mass Storage Reset	0xFF	プロトコルエラーを解除	○
GetMaxLUN	0xFE	デバイスがサポートする最大ユニット数を取得	○

○：実装　×：未実装

11.9.2　ストレージコマンド

HMSC がサポートするストレージコマンドを示します。

表 11-12 ストレージコマンド

コマンド名	コード	説明	対応
TEST_UNIT_READY	0x00	ペリフェラル機器の状態確認	○
REQUEST_SENSE	0x03	ペリフェラル機器の状態取得	○
FORMAT_UNIT	0x04	論理ユニットのフォーマット	×
INQUIRY	0x12	論理ユニットのパラメータ情報取得	○
MODE_SELECT6	0x15	パラメータ指定	○
MODE_SENSE6	0x1A	論理ユニットのパラメータ取得	×
START_STOP_UNIT	0x1B	論理ユニットのアクセス許可／禁止	×
PREVENT_ALLOW	0x1E	メディアの取り出し許可／禁止	○
READ_FORMAT_CAPACITY	0x23	フォーマット可能な容量取得	○
READ_CAPACITY	0x25	論理ユニットの容量情報取得	○
READ10	0x28	データ読み出し	○
WRITE10	0x2A	データ書き込み	○
SEEK	0x2B	論理ブロックアドレスに移動	×
WRITE_AND_VERIFY	0x2E	確認付きデータ書き込み	×
VERIFY10	0x2F	データ確認	×
MODE_SELECT10	0x55	パラメータ指定	×
MODE_SENSE10	0x5A	論理ユニットのパラメータ取得	○

○：実装　×：未実装

11.9.3　USB ストレージ機器の照合

USB-BASIC-F/W は、エニュメレーション時に GET_DESCRIPTOR リクエストで取得した情報を HMSC にコールバック関数で通知します。HMSCD は、このコールバック関数として登録するための API 関数 R_usb_hmsc_ClassCheck()を用意しています。R_usb_hmsc_ClassCheck()は、ディスクリプタ情報を解析して、照合結果を USB-BASIC-F/W の API 関数 R_usb_hstd_ReturnEnuMGR()で通知します。結果に問題がなければ、USB-BASIC-F/W はエニュメレーションを完了します。

11.9.4　USB ストレージ機器の情報取得

エニュメレーション完了後は、HMSDD の API 関数 R_usb_hmsc_StrgDriveSearch()をコールすることで USB ストレージ機器の情報取得処理をすることができます。この処理の完了は、R_usb_hmsc_StrgDriveSearch()コール時に登録するコールバック関数で通知されます。
GetMaxLUN リクエストの応答結果に関わらず、ユニット数 0 として動作します。よって、LUN0 のみのサポートとなります。
以下に USB ストレージ機器の情報取得シーケンスを示します。

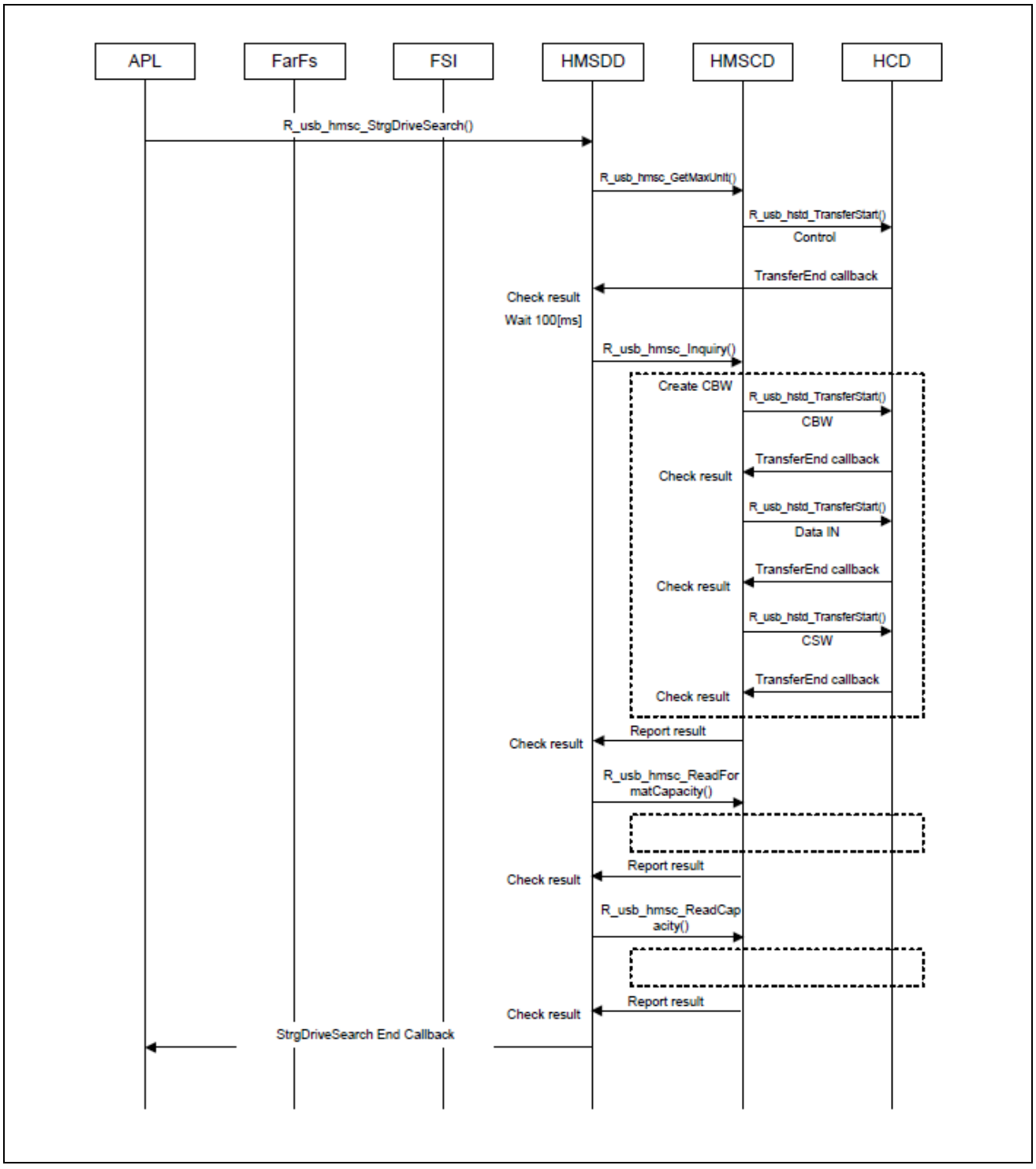


図 11-6 USBストレージ機器の情報取得シーケンス

11.9.5　USB ストレージ機器へのアクセス

情報取得完了後は、FatFs の API 関数で USB ストレージ機器にアクセスすることができます。
FatFs の API 関数をコールすると、ファイルシステム処理途中で FSI がコールされ HMSDD が動作します。
HMSDD は、処理に応じた HMSCD の API 関数をコールします。
それによって動作する HMSDC は、クラスリクエストの発行や BOT プロトコルに従った USB パケットの作成をします。
BOT プロトコルでは LBA に従ってデータ転送を行い、バイト数で転送サイズを指定します。
以下に USB ストレージ機器へのアクセスシーケンスを示します。

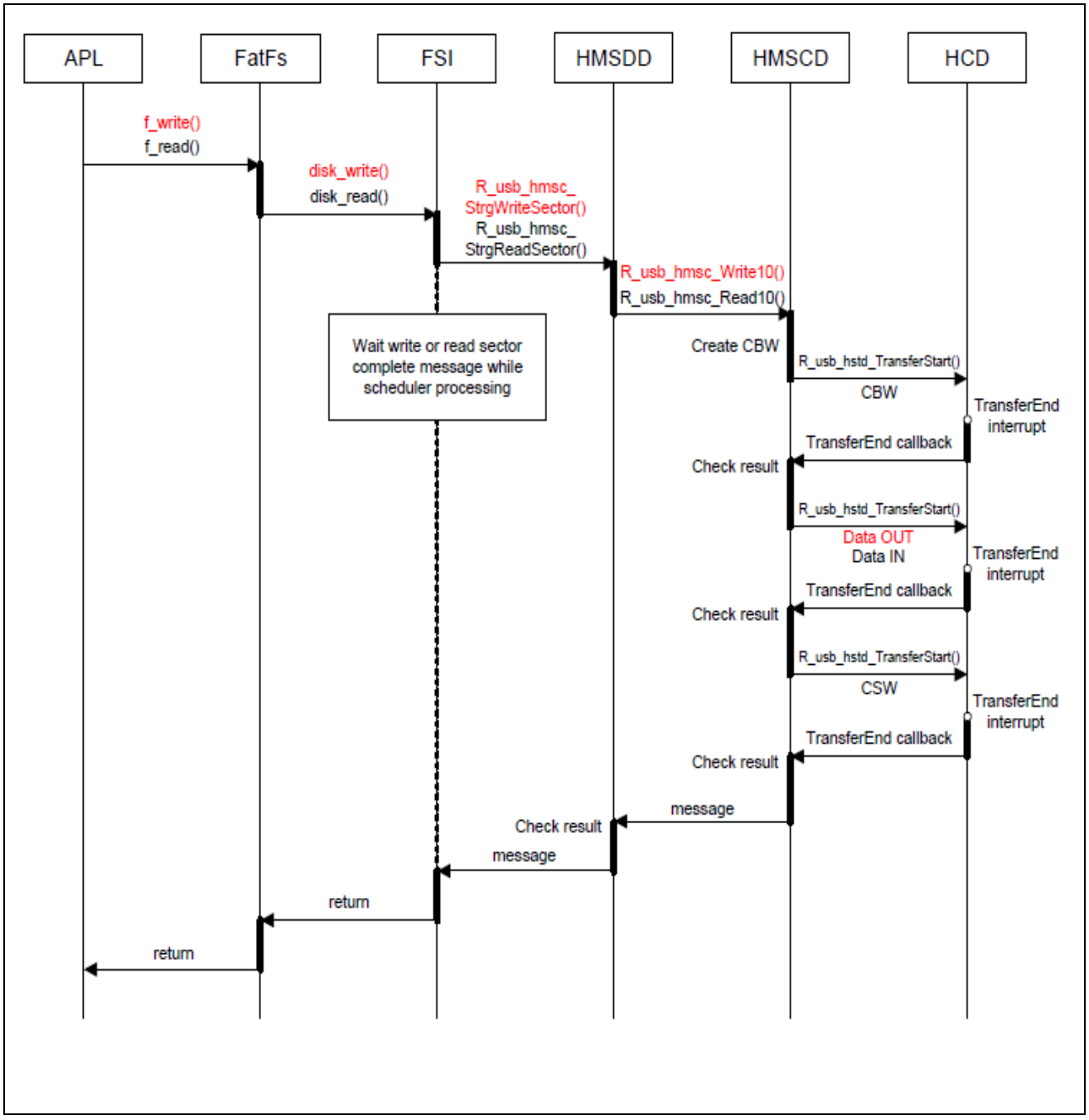


図 11-7 USBストレージ機器へのアクセスシーケンス

11.9.6　HMSCD API 関数

HMSCD の API 関数は主に HMSDD 向けに用意しています。
通常、アプリケーションで使用する関数は、R_usb_hmsc_Task()、R_usb_hmsc_driver_start()、R_usb_hmsc_Class_Check()の 3 つです。
別章「表 3-75」に HMSCD の API 関数を示します。

11.9.7　HMSDD API 関数

別章「表 3-76」に HMSDD API の関数を示します。

11.9.8　FSI 関数

FatFs は、単なるファイルシステムレイヤなので、ストレージデバイス制御レイヤは含まれません。
FatFs は、下位レイヤに対しインタフェースを要求しており、使用するプラットフォームやストレージデバイスに対応した制御関数を提供する必要があります。
HMSC は、この制御関数のサンプル（FSI 関数）を用意しています。FatFs の仕様を確認の上、必要があればシステムに合わせて変更してください。
別章「表 3-77」に FSI 関数一覧を示します。

11.9.9　スケジューラ設定

HMSC のスケジューラ設定を示します。

表 11-13 スケジューラ設定

関数名	タスク ID	優先度	メールボックス名	メモリプール名	概要
R_usb_hmsc_StrgDriveTask	USB_HSTRG_TSK	USB_PRI_3	USB_HSTRG_MBX	USB_HSTRG_MPL	HSTRG タスク
R_usb_hmsc_task	USB_HMSC_TSK	USB_PRI_3	USB_HMSC_MBX	USB_HMSC_MPL	HMSCD タスク
R_usb_hub_task	USB_HUB_TSK	USB_PRI_3	USB_HUB_MBX	USB_HUB_MPL	HUB タスク
R_usb_hstd_MgrTask	USB_MGR_TSK	USB_PRI_2	USB_MGR_MBX	USB_MGR_MPL	MGR タスク
r_usb_hstd_HciTask	USB_HCI_TSK	USB_PRI_1	USB_HCI_MBX	USB_HCI_MPL	HCD タスク

12. ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.12.01	-	初版発行
1.10	2017.05.10	46	3.2.8　WDTA 制御 追加
		46	3.2.9　IWDTA 制御 追加
		47	3.2.10　RIIC 制御
		50	3.2.11　USB ホスト制御
		432	8.　WDTA サンプルソフト
		451	9.　IWDTA サンプルソフト
		468	10.　RIIC サンプルソフト
		489	11.　USB ホストサンプルソフト
1.20	2018.09.17	2.ファイル構成	
		20	表 2-4 E2studio Ver6 対応のためファイル構成を修正 usbf/pcdc/ utilities/より CDC_Demo_Win7.inf を削除
		22	表 2-5 E2studio Ver6 対応のためファイル構成を修正 usbh_sample/GCC へ EC-1_init_serial_boot.gsi を追加
		24	表 2-6 E2studio Ver6 対応のためファイル構成を修正 Templates/GCC/serial_boot へ EC-1_init_serial_boot.gsi を追加 Templates/GCC へ usrheap.c を追加
		3.ドライバ	
		29	表 3-15 構造体メンバ追加
		39	表 3-47 送受信 FIFO バッファ段数 128 メッセージ設定を削除
		43	表 3-50 定数の記載漏れを修正
		57	表 3-66 関数名を修正。説明を追記。関数追加。
		110~123	3.7 RSPI 制御 関数名を修正
		111	3.7.2 R_RSPIm_Pin_Init 関数を追加。
		115	3.7.6 R_RSPIm_ClearState 関数を追加。
		4.CAN サンプルソフト	
		247	表 4-3 ローカル関数を追加
		249	4.4.1 main 戻り値を修正（void → int）
		250	4.4.2 can_main_init 関数を追加
		251	4.4.3 ecm_init 関数を追加
		252	4.4.4 icu_init 関数を追加
		300	図 4-2 図中の割り込み、コールバック関数の設定例を修正
		301	図 4-3 図中の割り込み、コールバック関数の設定例を修正
		305	図 4-6 図中の割り込み、コールバック関数の設定例を修正
		306	図 4-7 図中の割り込み、コールバック関数の設定例を修正
		307	図 4-8 図中の割り込み、コールバック関数の設定例を修正
		312	図 4-12 図中の割り込み、コールバック関数の設定例を修正 タイトルを変更
		313	図 4-13 サンプルコードのメッセージを受信しながらメッセージを送信するテスト（2/2）を追加
		316	図 4-16 図中の割り込み、コールバック関数の設定例を修正 タイトルを変更
		317	図 4-17 タイトルを変更
		318	図 4-18 図中の割り込み、コールバック関数の設定例を修正 タイトルを変更
		319	図 4-19 タイトルを変更

1.20	2018.09.17	320	図 4-20 図中の割り込み、コールバック関数の設定例を修正 タイトルを変更
		321	図 4-21 タイトルを変更
		322	図 4-22 図中の割り込み、コールバック関数の設定例を修正 タイトルを変更
		323	図 4-23 タイトルを変更
		324~331	4.5.6 コールバック処理 CAN0 に関する関する記述を削除、
		332	表 4-4 割り込み要因について CAN0 に関する記述を削除、割り込み優先度の誤りを修正
		333	4.6.3 使用準備（送信テスト・受信テスト）端子番号の誤りを修正
		5.RSPI サンプルソフト	
		343	5.1 概要 使用チャネルを 1 から 0 に変更
		354	図 5-1 タイトルを修正
		355	図 5-2 タイトルを修正
		356	表 5-2 使用チャネルを 1 から 0 に変更
		357	図 5-4 チャネル 0 の設定に修正
		358	5.5.5 サンプルプログラム実行例 説明を修正
		7.USB ファンクションサンプルソフト	
		390	表 7-4 EVENT_DETACH を表へ追加
		391	表 7-6 port_init のスコープを global へ修正
		393	7.5.2 static を削除
		11.USB ホストサンプルソフト	
		501	表 11-4 soft_wait 関数は不要のため削除
		505	soft_wait 関数削除
		533	Fat モジュールのバージョンを修正（R0.11 → R0.13）

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. 未使用端子の処理
   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】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. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】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。
5. 製品間の相違について
   【注意】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、内部ROM、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、ノイズ輻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。お客様の機器・システムの設計において、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には、お客様の責任において行ってください。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、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、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。
5. 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。
6. 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。
7. 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。
8. 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。
9. 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。
10. 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。
11. 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。
12. 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.4.0-1　2017.11)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口　http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/