RX ファミリ

MMC モード MMCIF ドライバ Firmware Integration Technology

要旨

本アプリケーションノートは、Firmware Integration Technology (FIT)を使用した MMC モード MMCIF ドライバについて説明します。本ドライバはルネサス エレクトロニクス製 RX ファミリ MCU 内蔵マルチメディアカードインタフェース（MMCIF）を使用して、マルチメディアカード（以下、MMC カードと略す）およびエンベディッド・マルチメディアカード（以下、eMMC と略す）を MMC モードで制御します。以降、本ドライバを MMCIF ドライバと称します。

動作確認デバイス

MMCIF 内蔵の RX ファミリ MCU

本アプリケーションノート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。

対象コンパイラ

・Renesas Electronics C/C++ Compiler Package for RX Family

・GCC for Renesas RX

・IAR C/C++ Compiler for Renesas RX

各コンパイラの動作確認内容については「6.1 動作確認環境」を参照してください。

関連ドキュメント

- ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
- RX ファミリ DMA コントローラ DMACA 制御モジュール Firmware Integration Technology (R01AN2063)
- RX Family DTC モジュール Firmware Integration Technology (R01AN1819)
- RX ファミリ コンペアマッチタイマ（CMT）モジュール Firmware Integration Technology (R01AN1856)
- RX ファミリ ロングワード型キューバッファ（LONGQ）モジュール Firmware Integration Technology (R01AN1889)

目次

1. 概要

1.1 MMCIF ドライバとは

本ドライバは別途提供している FAT ファイルシステムと組み合わせて使用することにより、MMC カードや eMMC のストレージデバイスに対してファイルアクセスが可能になります。

なお、MMC カードと eMMC の総称を MMC と略します。

本ドライバは API として、プロジェクトに組み込んで使用します。本ドライバの組み込み方については、「2.12 FIT モジュールの追加方法」を参照してください。

1.2 MMCIF ドライバの概要

表 1-1、表 1-2 に本ドライバの機能を示します。

表 1-1　MMCIF 機能一覧

項目	機能
準拠した規格	JEDEC STANDARD JESD84-A441 JEDEC STANDARD JESD84-B50
MMC 制御ドライバ	1 ブロック=512 バイトとするブロック型デバイスドライバ
MMC 動作電圧	2.7-3.6 V のみ、かつ 3.3V 信号レベルをサポート
MMC Bus インタフェース	MMC モード（1 ビット／4 ビット／8 ビット）をサポート
MMC 制御可能数	1 デバイス／チャネル
MMC Speed mode	Backward-compatible mode と High-speed mode をサポート MMCIF ドライバが Speed mode を判別し、マウント実行
MMC メモリ容量	byte access mode による 2G バイト以下メディアと sector access mode による大容量メディア（2G バイト超）をサポート
MMC メモリ制御対象	ユーザ領域のみをサポート ブート領域の制御はサポート対象外
MMC 検出機能	MMC カードの検出が可能
Boot Operation モード	サポート対象外
Background Operation	サポート対象外
High Priority Interrupt（HPI）	サポート対象外

表 1-2　MCU 機能一覧

項目	機能
対象 MCU	MMCIF 搭載の RX ファミリ MCU
MCU 内データ転送方式	Software 転送／DMAC 転送／DTC 転送の選択が可能 DMAC 転送／DTC 転送を行う場合、別途 DMAC 転送／DTC 転送のプログラムが必要
時間待ち処理	1ms カウンタ基準による待ちをサポート 別途ユーザ側にて 1ms 毎に、1ms 呼び出し用インターバルタイマカウント処理関数のコールが必要
OS 対応時の置換可能処理	時間待ち処理を OS の自タスク遅延処理に置き換えることが可能
エンディアン	ビッグエンディアン／リトルエンディアン対応
その他の機能	Firmware Integration Technology（FIT）に対応

1.2.1 アプリケーション構成図

MMCIF ドライバを使用して FAT ファイルシステムを構築する場合のアプリケーション構成図を図 1-1 に示します。

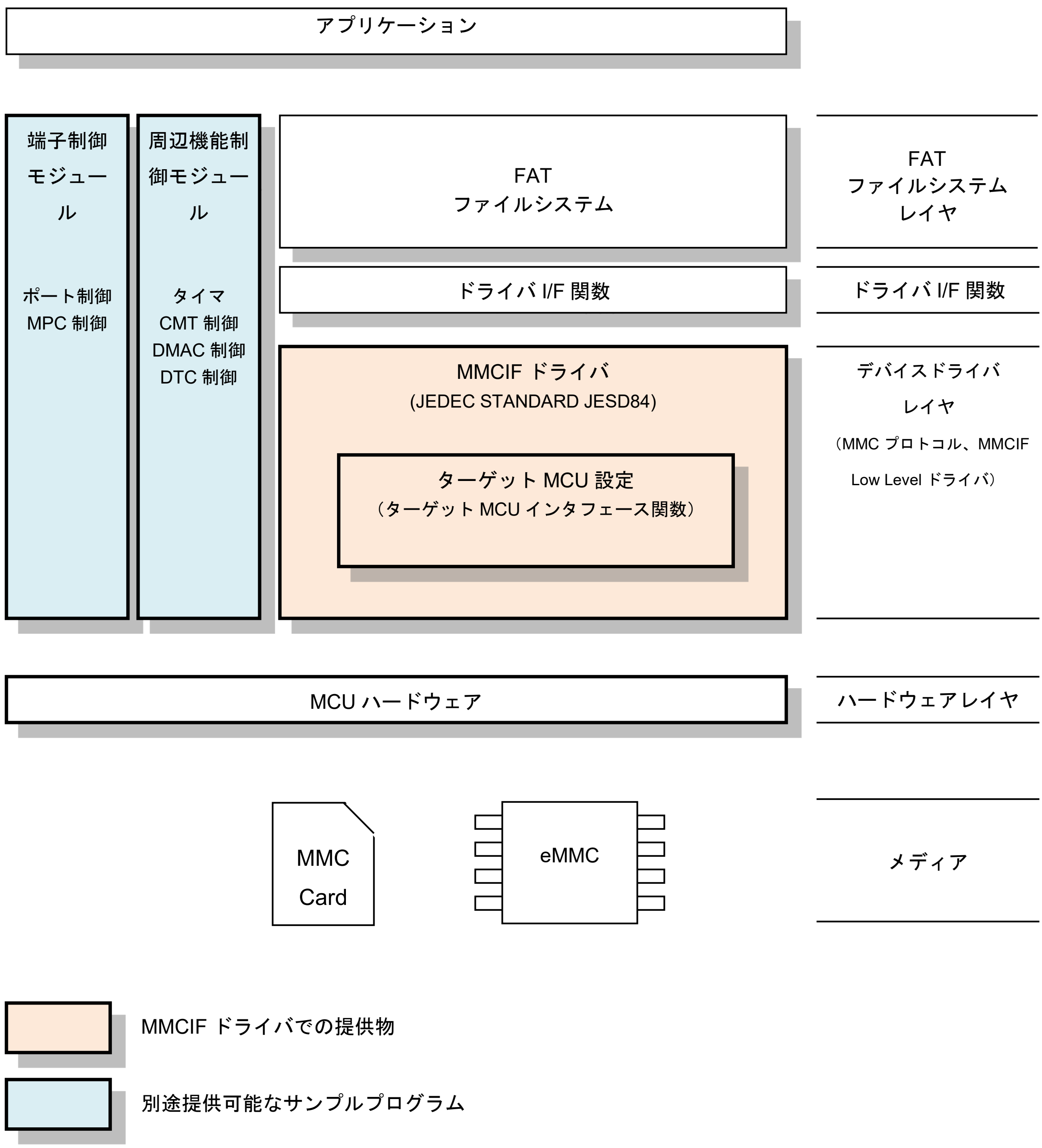


図 1-1　アプリケーション構成図

1.2.1.1　FAT ファイルシステム

MMC をファイル管理する場合に使用するソフトウェアです。別途 FAT ファイルシステムが必要です。必要に応じて以下から入手してください。

大容量メディア（2G バイト超）を使用する場合、MMCIF ドライバの上位層であるファイルシステムが FAT32 をサポートしている必要があります。

オープンソース FAT ファイルシステム M3S-TFAT-Tiny：https://www.renesas.com/mw/tfat

1.2.1.2　ドライバ I/F 関数

ルネサス エレクトロニクス製 FAT ファイルシステム API と MMCIF ドライバ API を接続するレイヤのソフトウェアです。必要に応じて、上記 M3S-TFAT-Tiny の Web ページから入手してください。

RX ファミリ M3S-TFAT-Tiny メモリドライバインタフェースモジュール Firmware Integration Technology

1.2.1.3　MMCIF ドライバ

JEDEC STANDARD JESD84 の MMC プロトコル制御と MMCIF Low Level アクセス制御を行うソフトウェアです。

1.2.1.4　周辺機能制御モジュール（サンプルプログラム）

タイマ制御、DMAC 制御、DTC 制御を行うソフトウェアです。サンプルプログラムが入手可能です。先頭ページの「関連ドキュメント」を参照し、入手してください。

1.2.1.5　端子制御モジュール（サンプルプログラム）

MMCIF 制御のための端子制御用ソフトウェアです。使用する MCU リソースは、ポート制御（MMCIF 機能制御と MMC カード電源用ポート制御）、MPC 制御（MMCIF 機能制御）です。

端子割り当てについては、使用端子が競合しないように、システムで一括して端子割り当てすることを推奨します。

なお、RX Family MCU RSK ボート用に合わせたサンプルプログラムを同梱済です。格納先は FITDemos です。参考にし、システムに合わせて組み込んでください。

1.3　API の概要

MMCIF ドライバは、JEDEC STANDARD JESD84 のプロトコルを使ったデバイスドライバです。

表 1-3 に本ドライバに含まれる API 関数を示します。

表 1-3　API 関数一覧

関数	関数説明
R_MMCIF_Open()	ドライバのオープン処理
R_MMCIF_Close()	ドライバのクローズ処理
R_MMCIF_Get_CardDetection()	挿入確認処理
R_MMCIF_Mount()	初期化処理
R_MMCIF_Unmount()	終了処理
R_MMCIF_Read_Memory()	リード処理　注 1
R_MMCIF_Read_Memory_Software_Trans()	リード処理（Software 転送）
R_MMCIF_Write_Memory()	ライト処理　注 1
R_MMCIF_Write_Memory_Software_Trans()	ライト処理（Software 転送）
R_MMCIF_Control()	ドライバのコントロール処理 　MMC_SET_STOP コマンド
R_MMCIF_Get_ModeStatus()	モードステータス情報取得処理
R_MMCIF_Get_CardStatus()	カードステータス情報取得処理
R_MMCIF_Get_CardInfo()	レジスタ情報取得処理
R_MMCIF_Int_Handler0()	割り込みハンドラ
R_MMCIF_Int_Handler1()	割り込みハンドラ
R_MMCIF_Cd_Int()	挿抜割り込み設定処理 （挿抜割り込みコールバック関数登録処理を含む）
R_MMCIF_IntCallback()	プロトコルステータス割り込みコールバック関数登録処理
R_MMCIF_Get_ErrCode()	ドライバのエラーコード取得処理
R_MMCIF_Get_BuffRegAddress()	データレジスタのアドレス取得処理
R_MMCIF_1ms_Interval()	インターバルタイマカウント処理
R_MMCIF_Set_DmacDtc_Trans_Flg()	DMAC/DTC 転送完了フラグセット処理
R_MMCIF_Set_LogHdlAddress()	LONGQ モジュールのハンドラアドレス設定処理　注 2
R_MMCIF_Log()	エラーログ取得処理　注 2
R_MMCIF_GetVersion()	ドライバのバージョン情報取得処理

注 1: 初期化処理時の動作モードのデータ転送設定として、DMAC 転送もしくは DTC 転送を設定する場合は、別途 DMAC 制御プログラムもしくは DTC 制御プログラムが必要です。設定方法は「1.4.4(2) DMAC／DTC の制御方法」を参照してください。

注 2 : 別途 LONGQ FIT モジュールが必要です。

1.4 処理例

1.4.1 クイックスタートガイド

Renesas Starter Kits（以降、RSK とする）を用いて、MMC カードへの読み出し／書き込み処理を行う手順を以下に示します。

1.4.1.1 ハードウェア設定

MMCIF 搭載マイコンの RSK は SD カードソケットを搭載済です。

マイコンの SDHI 制御端子と MMCIF 制御端子（4-bit／1-bit バス）は同一に割り当てられています[1]。そのため、RSK 上の SD カードソケットは、9 ピンの MMC カードソケットとして扱うことができます。

また、eMMC メーカがデバイス評価用途向けに eMMC を搭載した SD メモリカード形状互換基板を提供しています。

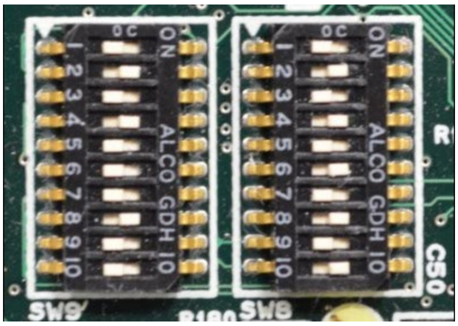
したがって、MMCIF 搭載マイコンの RSK と eMMC を搭載した SD メモリカード互換基板を使って、4-bit／1-bit バス eMMC を評価することができます。

ターゲット MCU の RSK 毎に次の設定を行ってください。

(1) RSK for RX64M／RX71M

SD カードソケットを有効にするため、以下のとおり設定してください。

SW9		SW8	
ピン番号	設定	ピン番号	設定
Pin 1	OFF	Pin 1	OFF
Pin 2	ON	Pin 2	ON
Pin 3	OFF	Pin 3	OFF
Pin 4	ON	Pin 4	ON
Pin 5	OFF	Pin 5	OFF
Pin 6	OFF	Pin 6	ON
Pin 7	OFF	Pin 7	OFF
Pin 8	ON	Pin 8	ON
Pin 9	OFF	Pin 9	OFF
Pin 10	OFF	Pin 10	ON



1 Renesas Starter Kit for RX65N-2MB (型名；RTK50565N2SxxxxxBE)を除く RSK が対象です。

(2) RSK for RX65N

SD カードソケットを有効にするため、以下のとおり設定してください。

SW7		SW8	
ピン番号	設定	ピン番号	設定
Pin 1	OFF	Pin 1	OFF
Pin 2	ON	Pin 2	ON
Pin 3	OFF	Pin 3	OFF
Pin 4	ON	Pin 4	ON
Pin 5	OFF	Pin 5	OFF
Pin 6	ON	Pin 6	ON
Pin 7	OFF	Pin 7	OFF
Pin 8	ON	Pin 8	ON
Pin 9	OFF	Pin 9	OFF
Pin 10	ON	Pin 10	OFF

(3) RSK for RX65N-2MB

設定不要です。

(4) RSK for RX72M

設定不要です。

(5) RSK for RX72N

設定不要です。

1.4.1.2　ソフトウェア設定

以下の手順に従い、プロジェクト環境にソフトウェアを組み込んでください。

1. e^2 studio上で新規プロジェクトを作成し、RX Driver Packageをダウンロードする。
2. e^2 studioのFITモジュールが格納されているフォルダ（通常はC:¥Renesas¥e2_studio¥FITModules）に r_mmcif_rx_vX.XX.zipとr_ mmcif_rx_vX.XX.xmlを格納する。
3. RXファミリe^2 studioに組み込む方法（R01AN1723）の「FITモジュールの組み込み方法」を参照し、r_bsp、r_mmcif_rx、r_cmt_rxをプロジェクトに組み込む。
4. サンプルプログラムr_mmcif_rx_demo_main（注１）をプロジェクトに格納する。

サンプルプログラムのコンフィギュレーションオプションの設定を行う。設定方法は「5.3 コンパイル時の設定」を参照。

5. r_mmcif_rx_config.hの#define MMC_CFG_DRIVER_MODEのメディア対象を“MMC_MODE_MMC（MMCカード）”にする。（注２）

注１：製品パッケージ内の FITDemos フォルダに同梱されています。

注２：r_mmcif_rx_pin.c に r_mmcif_demo_power_on()／r_mmcif_demo_power_off()関数があります。これらの関数は、MMC カード制御を想定した処理のため、MMC_CFG_DRIVER_MODE のメディア対象が“MMC_MODE_MMC”の場合に RSK 上の SD カードソケットに電源電圧を供給します。そのため、eMMC を搭載した SD メモリカード互換基板を MMC カードと認識させることで、eMMC に電源電圧を供給します。

1.4.2 基本制御

1.4.2.1 サポートコマンドについて

MMCIF ドライバは、以下のコマンドを使用します。

以下の表は、MMC コマンドと MMC 仕様書バージョンとユーザーズマニュアル ハードウェア編と本 MMCIF ドライバのサポート状況を示したものです。JEDEC STANDARD JESD84 欄の数値は、コマンドをサポートしたバージョン（但し、4.41 以降からのバージョン）を示します。

表 1-4　サポートコマンド一覧（－：記載なし、〇：サポート、×：未サポート）

コマンド	JEDEC STANDARD JESD84	マイコンのサポート範囲	本製品（MMCIF ドライバ）	備考
CMD0	A441	〇	〇	MMC 初期化で使用
CMD1	A441	〇	〇	MMC 初期化で使用
CMD2	A441	〇	〇	MMC 初期化で使用
CMD3	A441	〇	〇	MMC 初期化で使用
CMD4	A441	〇	〇	MMC 初期化で使用
CMD5	A441	〇	×	MMCIF ドライバでは未使用
CMD6	A441	〇	〇	MMC 初期化で使用
CMD7	A441	〇	〇	MMC 初期化で使用
CMD8	A441	〇	〇	MMC 初期化で使用
CMD9	A441	〇	〇	MMC 初期化で使用
CMD10	A441	〇	×	MMCIF ドライバでは未使用
CMD11	A441　注 1	－	×	MMCIF ドライバでは未使用
CMD12	A441	〇	〇	リード／ライト処理で使用
CMD13	A441	〇	〇	リード／ライト処理で使用
CMD14	A441	〇	〇	MMC 初期化で使用
CMD15	A441	〇	×	MMCIF ドライバでは未使用
CMD16	A441	〇	〇	MMC 初期化で使用
CMD17	A441	〇	〇	リード／ライト処理で使用
CMD18	A441	〇	〇	リード／ライト処理で使用
CMD19	A441	〇	〇	MMC 初期化で使用
CMD20	A441　注 1	－	×	MMCIF ドライバでは未使用
CMD21-22	Reserved	－	－	－
CMD23	A441	〇	〇	リード／ライト処理で使用
CMD24	A441	〇	〇	リード／ライト処理で使用
CMD25	A441	〇	〇	リード／ライト処理で使用
CMD26	A441	〇	×	MMCIF ドライバでは未使用
CMD27	A441	〇	×	MMCIF ドライバでは未使用
CMD28	A441	〇	×	MMCIF ドライバでは未使用
CMD29	A441	〇	×	MMCIF ドライバでは未使用
CMD30	A441	〇	×	MMCIF ドライバでは未使用
CMD31	A441	〇	×	MMCIF ドライバでは未使用
CMD32-34	Reserved	－	－	－
CMD35	A441	〇	×	MMCIF ドライバでは未使用
CMD36	A441	〇	×	MMCIF ドライバでは未使用
CMD37	Reserved	－	－	－
CMD38	A441	〇	×	MMCIF ドライバでは未使用

CMD39	A441	○	×	MMCIF ドライバでは未使用
CMD40	A441	○	×	MMCIF ドライバでは未使用
CMD41	Reserved	－	－	－
CMD42	A441	○	×	MMCIF ドライバでは未使用
CMD43-54	Reserved	－	－	－
CMD55	A441	○	×	MMCIF ドライバでは未使用
CMD56	A441	○	×	MMCIF ドライバでは未使用
CMD57-63	Reserved	－	－	－
Root Operation		○	×	MMCIF ドライバでは未使用

注１：JEDEC STANDARD JESD84-B45 にて削除されたコマンド

1.4.2.2 データバッファと MMC 上のデータとの関係

MMCIF ドライバは、送信／受信データポインタを引数として設定します。RAM 上のデータバッファのデータ並びと送信／受信順番の関係を図 1-2 に示すように、エンディアンに関係なく、送信データバッファの並びの順に送信し、また、受信の順に受信データバッファに書き込みます。

ホスト送信時

RAM 上の送信データバッファ（バイト表示）

0	1	・・・	508	509	510	511

データの送信順番

スレーブデバイスへの書き込み（バイト表示）

0	1	・・・	508	509	510	511

データの受信順

ホスト受信時

スレーブデバイスからの読み出し（バイト表示）

0	1	・・・	508	509	510	511

データの送信順

RAM 上の受信データバッファ（バイト表示）

0	1	・・・	508	509	510	511

受信データバッファへの書き込み

図 1-2　転送データの格納

1.4.2.3 マウント時の動作電圧設定について

R_MMCIF_Mount()関数の引数には動作電圧を設定する必要があります。MMC 初期化処理時に MMC が設定された動作電圧で動作できないと判断した場合、MMC は Inactive State に遷移します。

MMC カードの場合、R_MMCIF_Unmount()関数をコールし、アンマウント状態にした後、MMC カードを抜去してください。その後、再度挿入し、動作電圧を設定し直して、再度マウント処理を行ってください。

eMMC の場合、R_MMCIF_Unmount()関数をコールし、アンマウント状態にした後、eMMC への電源供給を停止してください。その後、eMMC への電源供給を再開し、動作電圧を設定し直して、再度マウント処理を行ってください。

1.4.2.4　MMC_CLK の停止

MMCIF ドライバは、消費電力を下げるために API 関数実行中のみ MMC_CLK を出力し、API 関数終了時に MMC_CLK 出力を停止します。

1.4.2.5　MMCIF ステータス確認

MMC カードの操作を行う上で、通信の終了検出等 MMCIF のステータス確認や MMC カードの挿抜検出を行う必要があります。ここでは、MMCIF ドライバの API 関数使用時のステータス確認方法について説明します。

(1) ステータス確認方法

MMCIF ドライバは、MMCIF のステータス確認方法として、MMCIF 割り込みとソフトウェアポーリングの 2 種類を選択できます。

確認するステータスとして、以下があります。

・MMC カード挿抜検出

・MMC プロトコル

表 1-5 に MMCIF ドライバの API 関数で確認するステータスを示します。

表 1-5　確認するステータス

分類	ステータス	備考
MMC カード挿抜（R_MMCIF_Cd_Int()関数で割り込み許可／禁止設定）	MMC カード挿入／抜去状態	R_MMCIF_Get_CardDetection()関数で検出可能
MMC プロトコル（R_MMCIF_Mount()関数で割り込み許可設定）	レスポンス受信完了	コマンド送信毎に発生
	データ転送要求	512 バイト転送毎に発生
	プロトコルエラー	CRC エラー等発生時
	タイムアウトエラー	レスポンス応答無し等発生時

(2) 設定方法

MMC カード挿入確認方法として割り込みを選択する場合は、R_MMCIF_Mount()関数にて MMC プロトコルのステータス確認も割り込み（MMC_MODE_HWINT 設定）を選択してください。

なお、MMCIF 割り込みに対応する割り込みハンドラとして、R_MMCIF_Int_HandlerX()関数（X は、チャネル番号）をシステムに登録済です。

(3) ソフトウェアポーリングと割り込みによる MMC カード挿抜確認

MMC カード挿抜割り込みの許可／禁止設定に関わらず、R_MMCIF_Get_CardDetection()関数を使って、MMC カードの挿入状態を確認できます。

R_MMCIF_Cd_Int()関数で割り込み許可（MMC_CD_INT_ENABLE）を設定した場合は、MMC カード挿抜割り込み発生時にコールバック関数を実行します。そのため、MMC カードの挿抜に対するリアルタイムの処理が可能です。MMC カード挿抜割り込みコールバック関数は、R_MMCIF_Cd_Int()関数で登録してください。

(4) ソフトウェアポーリングによる MMC プロトコルステータス確認

MMC プロトコルのステータス確認方法として、R_MMCIF_Mount()関数でポーリング（MMC_MODE_POLL）を設定した場合は、リード／ライト処理の中で、MMC カードとの通信時のレスポンス受信待ちやデータ転送完了待ちをソフトウェアポーリングで確認します。

ソフトウェアポーリング設定時は、ターゲット MCU インタフェース関数 r_mmcif_dev_int_wait()関数を使用し、この関数内で割り込みステータスフラグレジスタ取得処理（r_mmcif_get_intstatus()関数）をコールし、割り込みステータスフラグレジスタ（CEINT）を確認します。
図 1-3 にポーリングを使用した場合の MMC プロトコルステータス確認のフローチャート例を示します。

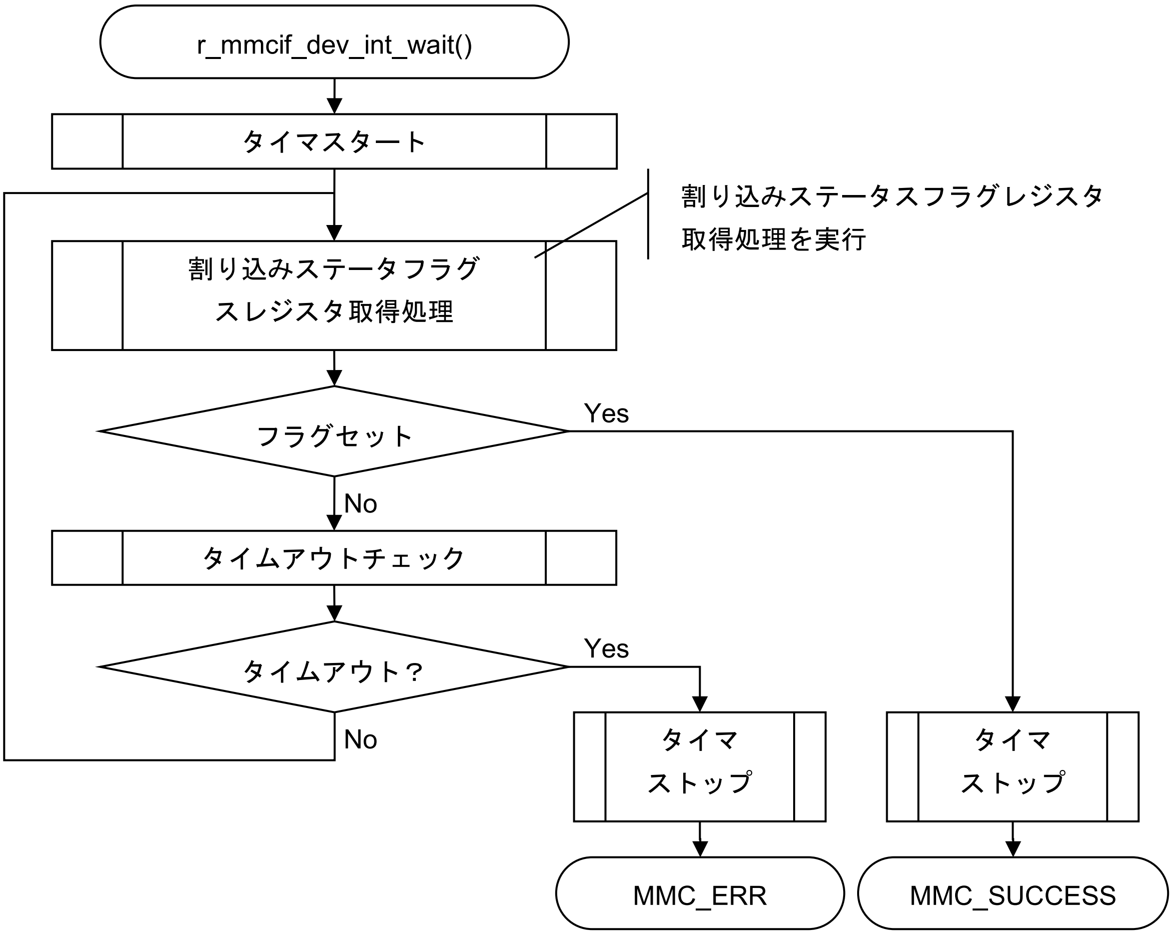


図 1-3　ソフトウェアポーリングによる MMC プロトコルステータス確認

(5) 割り込みによる MMC プロトコルステータス確認方法

MMC プロトコルのステータス確認方法として、R_MMCIF_Mount()関数で割り込み（MMC_MODE_HWINT）を設定した場合は、ステータス確認の割り込み発生でステータスを内部バッファにセットします。

ステータス確認の割り込み発生時、ユーザが登録済のコールバック関数をコールすることができます。MMC プロトコルステータス割り込みコールバック関数は R_MMCIF_IntCallback()関数で登録してください。

割り込み待ち設定時、ターゲット MCU インタフェース関数 r_mmcif_dev_int_wait()関数を使用し、この関数内で割り込みステータスフラグレジスタ取得処理（r_mmcif_get_intstatus()関数）を実行し、割り込み発生状態を確認します。
図 1-4 に割り込みを使用した場合の MMC プロトコルステータス確認のフローチャート例を示します。

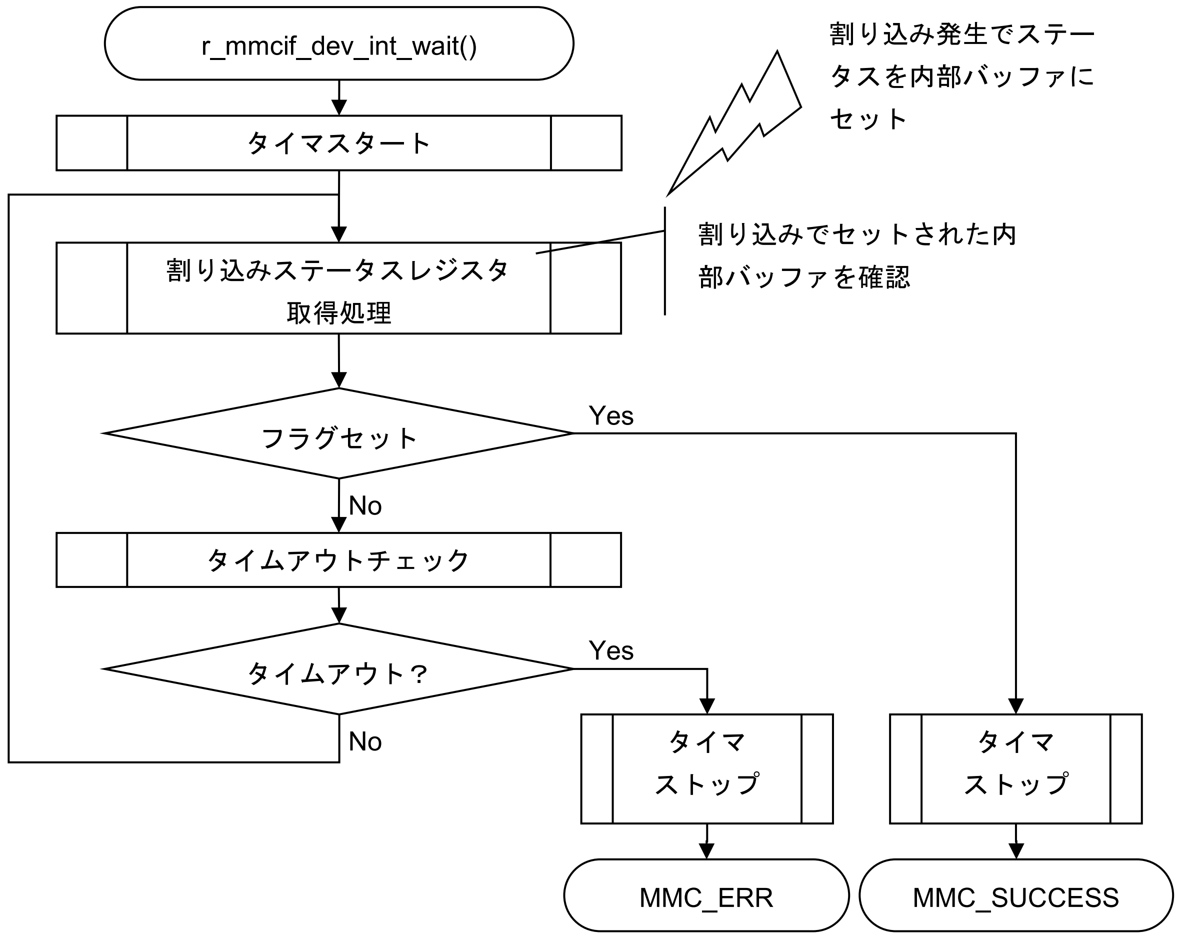


図 1-4　割り込みによる MMC プロトコルステータス確認例

1.4.3 エラー時の制御

1.4.3.1 エラー発生時の処理方法

リード処理／ライト処理等でエラーが発生した場合、処理のリトライを推奨します。

処理のリトライにも関わらずエラーが発生する場合、MMC カードの挿抜を実施し、MMC カードを再初期化してください。MMC カードの挿抜に関わる処理方法は、「4.4」「4.5」を参照してください。eMMC の場合、電源供給を一旦停止し、再供給後、初期化してください。

また、MMCIF ドライバの上位アプリケーションとしてファイルシステム等を使用する場合、MMC カードの挿抜処理の前に、事前に上位アプリケーションで必要な処理を実行してください。

1.4.3.2 Transfer State (tran)遷移後のエラー終了処理

Transfer State (tran)遷移後にエラーが発生した場合、データ転送の有無に関わらず、CMD12 を発行します。CMD12 の発行は、Transfer State (tran)状態に遷移させることを目的としています。但し、ライト処理中に CMD12 が発行された際、MMC カードがビジー状態に遷移する場合があります。そのため、次のリード／ライト関数コール時にエラーを返す場合があります。

1.4.3.3 エラーログ取得方法

MMCIF ドライバのソースコードをご使用ください。また、別途 LONGQ FIT モジュールを入手してください。

エラーログを取得するために、以下の設定を行ってください。

(1) R_LONGQ_Open()の設定

LONGQ FIT モジュールの R_LONGQ_Open()の第三引数 ignore_overflow は“1”を設定してください。これによりエラーログバッファは、リングバッファとして使用することが可能です。

(2) 制御手順

R_MMCIF_Open()をコールする前に、以下の関数を順番にコールしてください。設定方法例は「R_MMCIF_Set_LogHdlAddress()」を参照してください。

1. R_LONGQ_Open()
2. R_MMCIF_Set_LogHdlAddress()

(3) R_MMCIF_Log()の設定

エラー取得を終了する場合、コールしてください。設定方法例は「R_MMCIF_Log()」を参照してください。

1.4.4 他モジュールの制御

1.4.4.1 タイマ

タイムアウト検出目的で使用します。

1 ミリ秒毎に R_MMCIF_1ms_Interval()をコールしてください。但し、r_mmcif_dev.c の r_mmcif_dev_int_wait()および r_mmcif_dev_wait()を OS 処理に置き換える場合には不要です。

1.4.4.2 DMAC／DTC の制御方法

DMAC 転送もしくは DTC 転送を使用する場合の制御方法を説明します。

MMCIF ドライバでは、DMAC または DTC の転送起動、および転送完了待ちを行います。その他の DMAC レジスタもしくは DTC レジスタへの設定は DMAC FIT モジュールもしくは DTC FIT モジュールを使用するか、ユーザ独自で処理を作成してください。

なお、DMAC 転送設定の場合、DMAC 起動が完了した際の転送完了フラグのクリアはユーザが行う必要があります。

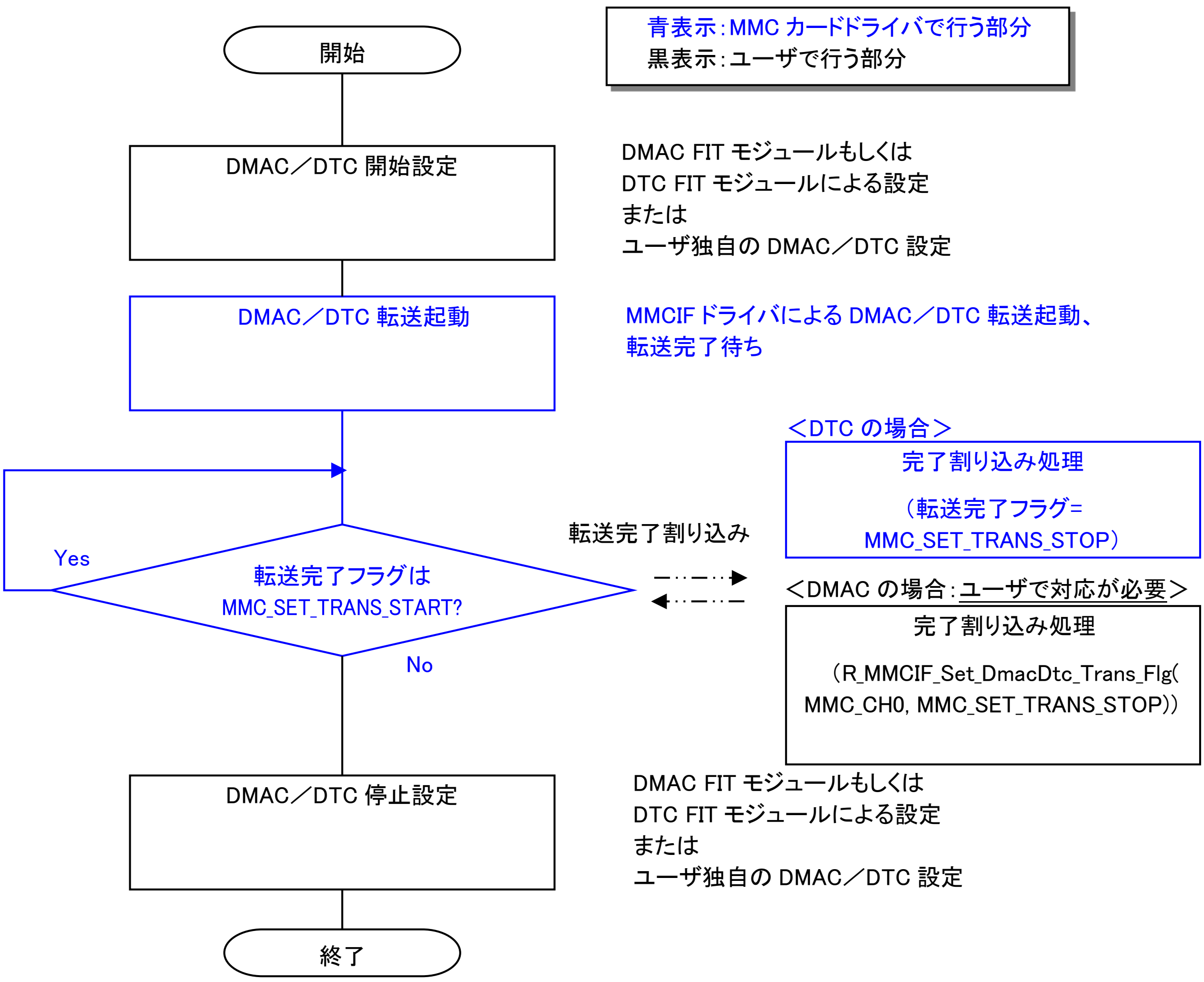
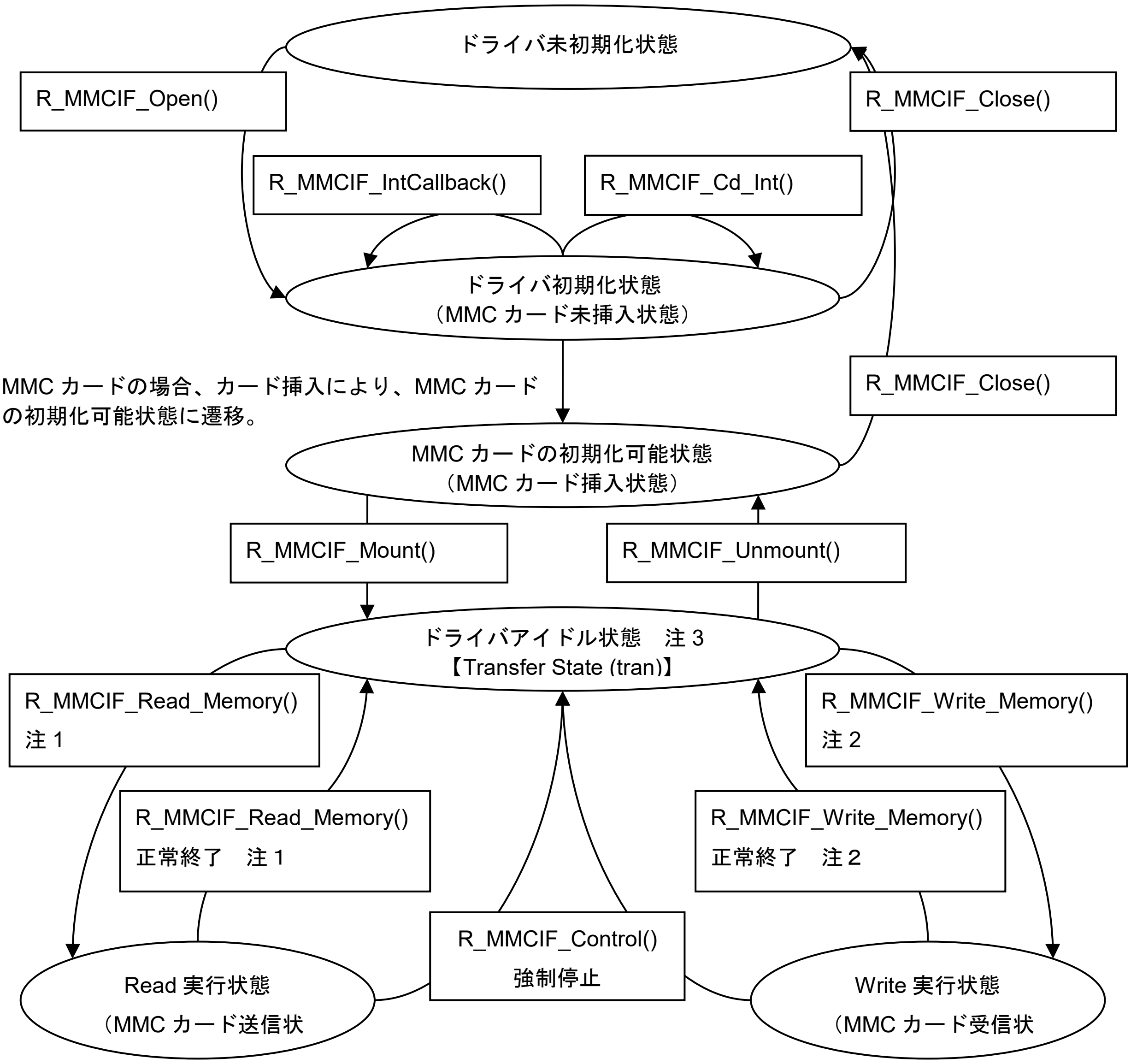


図 1-5　DMAC 転送および DTC 転送設定時の処理

1.5　状態遷移図

図 1-6 に本ドライバの状態遷移図を示します。



注 1：R_MMCIF_Read_Memory_Software_Trans()を含む、MMC 送信状態遷移コマンド処理も同様
注 2：R_MMCIF_Write_Memory_Software_Trans()を含む、MMC 受信状態遷移コマンド処理も同様
注 3：【】は、JEDEC STANDARD JESD84 記載の MMC の状態を示す。

図 1-6　MMCIF ドライバの状態遷移図

1.6 制限事項

1.6.1 ご使用上の注意事項

- 引数の設定規則、レジスタの保証規則
  本ライブラリで提供する関数は、C言語で記述したアプリケーションプログラムから呼び出されることを前提に作成されています。MMCIFドライバの引数の設定規則やレジスタの保証規則は、Cコンパイラの設定規則および保証規則に準じています。関連マニュアルをご参照ください。
- セクションについて
  初期値無し領域のセクションは、0に初期化してください。
- 割り込みコールバック関数使用時の注意事項
  割り込みコールバック関数は、割り込みハンドラのサブルーチンとして呼び出されます。
- 使用にあたっては、ハードウェアに合わせて、ソフトウェアを設定してください。

1.6.2 MMC の電源供給の注意事項

MMC の仕様に基づいて、電源電圧を供給する必要があります。JEDEC STANDARD JESD84 の Power-up の章を参照してください。

特に、MMC カードの抜去後の MMC カードの再挿入制御、もしくは MMC カードの電源の切断後の再投入制御を行う場合は、電圧値と電圧維持期間についての規定を参照し、システム側で回路や切断／再投入の制御タイミングを設けてください。

正しい電源投入切断処理が行われていない場合、MMC カードの挿抜により、電源等が不安定になり、MCU がリセット状態になる可能性があります。

動作電圧に達した後、マウント処理 R_MMCIF_Mount()関数を実行してください。電源電圧供給開始後、動作電圧に達するまでの時間が不足している場合、時間調整してください。

また、MMC 電源電圧供給停止後、MMC の抜去可能電圧に達するまでの時間待ち処理は、アプリケーションプログラムで対応する必要があります。

1.6.3 MMC カードの挿抜検出に関する注意事項

通信中において、MMC カードが抜かれた場合、コマンドに対するレスポンス異常となり、結果的に MMCIF ドライバがエラーを返します。また、エラーが返るまでの時間は、実行中の処理に依存します。

ただし、通信中に、一瞬挿抜が行われた場合、以下のように MMCIF ドライバがエラーを返さない可能性があります。

- 定周期でのポーリングによるカード検出を行う場合、1 周期内の抜挿は MMCIF ドライバでは検出できず、MMC からのレスポンス異常が無いと判断した場合は、処理を続けます。
- 書き込み中に一瞬抜挿すると、MMCIF ドライバが書き込み完了と認識してしまう可能性があります。これは、MMC が書込み Busy 信号完了を H 出力で通知するためです。（MMC_Dn（n は、0-7）端子は、外部抵抗でプルアップされています。）

ハードウェア割り込み制御や、ポーリング周期の見直し等で、MMC カードが抜かれた場合の検出方法をシステムに適した方法で対応してください。

1.6.4　ソフトウェア・ライトプロテクト対応について

MMCIF ドライバは、ソフトウェアよるプロテクト状態制御機能をサポートしていません。

他システムで設定したソフトウェア・ライトプロテクト領域に対して書き込むと、エラーを返します。

1.6.5　MMC カード挿抜時のチャタリング制御

MMCIF ドライバでは MMC を挿抜したときに発生するチャタリング除去制御を行いません。また、ハードウェアにも除去する機能はありません。MMC カードを制御する場合、「R_MMCIF_Get_CardDetection()」と「R_MMCIF_Cd_Int()」の Example、及び、サンプルプログラムを参考にチャタリングを考慮したカード検出処理を作成してください。

2. API 情報

本 FIT モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- MMCIF

2.2 ソフトウェアの要求

このドライバは以下の FIT モジュールに依存しています。

- r_bsp Rev.5.20 以上
- r_dmaca_rx（DMACA FIT モジュールを用いて、DMAC 転送を使用する場合のみ）
- r_dtc_rx（DTC FIT モジュールを用いて、DTC 転送を使用する場合のみ）
- r_cmt_rx（コンペアマッチタイマ CMT FIT モジュールを使用する場合のみ）

他タイマやソフトウェアタイマで代用できます。

2.3 サポートされているツールチェーン

本ドライバは「6.1 動作確認環境」に示すツールチェーンで動作確認を行っています。

2.4 使用する割り込みベクタ

MMCIF ドライバのマクロ定義 MMC_CFG_DRIVER_MODE が MMC_MODE_HWINT の時、MMCIF 割り込みが有効になります。MMCIF ドライバのオープン処理 R_MMCIF_Open()をコールするまでにシステムの割り込みを許可してください。

表 2-1 に MMCIF ドライバが使用する割り込みベクタを示します。

表 2-1　使用する割り込みベクタ一覧

デバイス	割り込みベクタ
RX64M RX65N RX66N RX71M RX72M RX72N	MMCIF バッファアクセス割り込み（MBFAI）（ベクタ番号：45） GROUPBL1 割り込み（ベクタ番号: 111） ● MMC 検出割り込み（CDETIO）（グループ割り込み要因番号：6） ● エラー／タイムアウト割り込み（ERRIO）（グループ割り込み要因番号：7） ● 通常動作割り込み（ACCIO）（グループ割り込み要因番号：8）

2.4.1 割り込み処理内からコール可能な API

表 2-2 に割り込み処理内からコール可能な API（推奨）を示します。

割り込みコールバック関数は、割り込みハンドラのサブルーチンとして呼び出されます。

表 2-2　割り込みハンドラ内からのコールを許可する MMCIF ドライバ・API 関数一覧

関数名	機能概要	備考
R_MMCIF_Control()	ドライバのコントロール処理	MMC_SET_STOP（強制停止要求コマンド）

2.5 ヘッダファイル

すべての API 呼び出しと使用されるインタフェース定義は r_mmcif_rx_if.h に記載しています。

ビルド毎の構成オプションは、r_mmcif_rx_config.h で選択します。

```
#include "r_mmcif_rx_if.h"
```

2.6 整数型

このドライバは 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。

2.7 コンパイル時の設定

本ドライバのコンフィギュレーションオプションの設定は、r_mmcif_rx_config.h で行います。

RX64M RSK を使用する場合の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。

Configuration options in *r_mmcif_rx_config.h*	
#define MMC_CFG_USE_FIT ※デフォルト値は“有効”	MMCIF ドライバの BSP 環境での使用有無を選択してください。 無効にした場合、r_bsp 等の FIT モジュール制御を無効にします。また、別途、処理を組み込む必要があります。 有効にした場合、r_bsp 等の FIT モジュール制御を有効にします。
#define MMC_CFG_CHx_INCLUDED ※チャネル 0 のデフォルト値は“有効” ※“x”はチャネル番号	該当チャネルを使用するかを選択してください。 無効にした場合、該当チャネルに関する処理をコードから省略します。 有効にした場合、該当チャネルに関する処理をコードに含めます。 複数チャネルをサポートする MCU の場合、多チャネルの定義を追加する必要があります。
#define MMC_CFG_DRIVER_MODE (MMC_MODE_HWINT | MMC_MODE_eMMC | MMC_MODE_1BIT) ※デフォルト値は“ステータス確認：ハードウェア割り込み、データ転送：Software 転送、メディア対応：eMMC、MMC バス幅：MMC モード 1 ビットバス”	この定義をマウント処理 R_MMCIF_Mount()関数の引数 p_mmc_Config ->mode として使用できます。 「3.4 R_MMCIF_Mount()」を参照し、p_mmc_Config ->mode を定義してください。
#define MMC_CFG_CHx_CD_ACTIVE (0) ※デフォルト値は“無効” ※“x”はチャネル番号	MMC_CD 端子の割り当てが不要な場合、個別に MMCIF ドライバの制御対象から外すことができます。 端子を MMCIF 機能に割り当てて制御する場合、(1)に設定してください。 制御対象から外す場合、(0)に設定してください。 制御対象から外した場合、その端子を他用途（汎用入出力ポート等）に利用できます。MMCIF ドライバは機能的に制御を外すものであり、使用端子設定機能を持っていません。そのため、他用途で利用する場合、別途、使用端子も併せて設定してください。使用チャネル毎に設定が必要です。
#define MMC_CFG_DIV_HIGH_SPEED MMC_DIV_2 /* 52MHz or less clock */ ※デフォルト値は“MMC_DIV_2”（2 分周） 注 1	High-speed mode のクロック周波数定義です。JEDEC STANDARD JESD84 の場合、最大クロックは 52MHz です。 MMCIF クロック周波数設定ビット（CLKDIV[3:0]）に PCLKB の分周比を設定してください。設定値は MMC_DIV_2～MMC_DIV_1024（2 分周～1024 分周）としてください。 例えば、PCLKB=60MHz、High-speed mode のクロック周波数=30MHz の場合、MMC_DIV_2（2 分周）を設定してください。
#define MMC_CFG_DIV_BACKWARD_COM_SPEED MMC_DIV_4 /* 26MHz or less clock */ ※デフォルト値は“MMC_DIV_4”（4 分周） 注 1	Backward-compatible mode のクロック周波数定義です。JEDEC STANDARD JESD84 の場合、最大クロックは 26MHz です。設定方法は、上記 High-speed mode と同様です。 例えば、PCLKB=60MHz、Backward-compatible mode のクロック周波数=15MHz の場合、MMC_DIV_4（4 分周）を設定してください。
#define MMC_CFG_DIV_INIT_SPEED MMC_DIV_1024 /* 400KHz or less clock */ ※デフォルト値は“MMC_DIV_1024”（1024 分周） 注 1 注 2	Card identification mode のクロック周波数定義です。JEDEC STANDARD JESD84 の場合、最大クロックは 400kHz です。設定方法は、上記 High-speed mode と同様です。 例えば、PCLKB=60MHz、Card identification mode のクロック周波数=58KHz の場合、MMC_DIV_1024（1024 分周）を設定してください。
#define MMC_CFG_TIMEOUT_TRANS	書き込みデータ／読み出しデータ転送時のタイムアウト設定です。

(0x000000a0ul)　/* CECLKCTR register : Write data/read data timeout */ ※デフォルト値は“0x000000a0ul”	クロックコントロールレジスタ（CECLKCTRL）の SRWDTO[3:0]のタイムアウト値を bit 7-4 に設定してください。
#define MMC_CFG_TIMEOUT_RESBUSY (0x00000f00ul)　/* CECLKCTR register : Response busy timeout */ ※デフォルト値は“0x00000f00ul”	レスポンスビジー時のタイムアウト設定です。 クロックコントロールレジスタ（CECLKCTRL）の SRBSYTO[3:0]のタイムアウト値を bit 11-8 に設定してください。
#define MMC_CFG_TIMEOUT_RES (0x00002000ul)　/* CECLKCTR register : Response timeout */ ※デフォルト値は“0x00002000ul”	レスポンス受信時のタイムアウト設定です。 クロックコントロールレジスタ（CECLKCTRL）の SRSPT[3:0]のタイムアウト値を bit 13-12 に設定してください。
#define MMC_CFG_CHx_INT_LEVEL　(10)　/* MMC channel x interrupt level */ ※デフォルト値は“(10)”	通常動作割り込み（ACCIO）、エラー／タイムアウト割り込み（ERRIO）、MMC 検出割り込み（CDETIO）レベルを設定してください。
#define MMC_CFG_CHx_INT_LEVEL_DMADTC (10)　/* MMC channel x DMA/DTC interrupt level */ ※デフォルト値は“(10)”	MMCIF バッファアクセス割り込み（MBFAI）レベルを設定してください。 DMAC／DTC を使用して、MMCIF バッファにデータを書き込む場合、または、MMCIF バッファからデータを読み出す場合の割り込みレベルです。
/* #define MMC_CFG_LONGQ_ENABLE */ ※デフォルト値は“無効”	LONGQ FIT モジュールを使ったエラーログ取得機能を利用する場合に定義してください。 この機能を利用する場合、デバッグ用モジュール（この定義を有効にした作成した専用モジュール）を使用し、かつ LONGQ FIT モジュールを組み込む必要があります。

注 1 : MMC_DIV_n（n は、整数で分周比を示す。）は、MMCIF の PCLK の分周比を示します。使用する MCU の電気的特性により、JEDEC STANDARD JESD84 の最大転送周波数を設定できない場合があります。設定可能な最大転送周波数は、使用する MCU のユーザーズマニュアルハードウェア編を参照してください。

注 2 : Card identification mode 時は、MMC の CMD ラインがオープンドレインになります。そのため、動作可能なクロック周波数は、MMC_CMD 端子のプルアップ抵抗値と端子の負荷容量に依存します。「4.6 ハードウェア設定」も参照してください。

2.8　コードサイズ

本ドライバのコードサイズを下表に示します。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.7 コンパイル時の設定」のコンフィギュレーションオプションによって決まります。

下表の値は下記条件で確認しています。

モジュールリビジョン: r_mmcif_rx rev1.07

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(統合開発環境のデフォルト設定に"-lang = c99"オプションを追加)

GCC for Renesas RX 4.08.04.201902

(統合開発環境のデフォルト設定に" -std=gnu99"オプションを追加)

IAR C/C++ Compiler for Renesas RX version 4.12.1

(統合開発環境のデフォルト設定)

コンフィグレーションオプション: デフォルト設定

表 2-3　コードサイズ

ROM、RAM およびスタックのコードサイズ（注１）				
デバイス	分類	使用メモリ		
		Renesas Compiler	GCC	IAR Compiler
RX65N	ROM 注２	9650 バイト	18,004 バイト	12,749 バイト
	RAM 注２、注３	276 バイト（ワークバッファ）注４	624 バイト	624 バイト
	最大使用ユーザスタック	320 バイト	-	288 バイト
	最大使用割り込みスタック	40 バイト	-	68 バイト

注１：リトルエンディアン時の値です。エンディアンにより、上記のメモリサイズは、異なります。
注２：サイズは、データ転送方式等の設定に依存します。
注３：読み出し／書き込みのためのデータバッファを含みません。
注４：ワークバッファの詳細はドライバのオープン処理関数を参照してください。

2.9　引数

API 関数の引数である構造体を示します。この構造体は、API 関数のプロトタイプ宣言とともに r_mmcif_rx_if.h に記載されています。

2.9.1　e_mmc_enum_cmd 構造体定義

```
typedef enum e_mmc_enum_cmd
{
    MMC_SET_STOP,
} mmc_enum_cmd_t;
```

2.9.2　e_mmc_enum_trans 構造体定義

```
typedef enum e_mmc_enum_trans
{
   MMC_SET_TRANS_STOP,
   MMC_SET_TRANS_START
} mmc_enum_trans_t;
```

2.9.3 mmc_cmd_t 構造体定義

```
typedef struct
{
   mmc_enum_cmd_t cmd;
   uint32_t       mode;          /* Lock/Unlock operation code */
   uint8_t        *p_buff;
   uint32_t       size;
} mmc_cmd_t;
```

2.9.4 mmc_cfg_t 構造体定義

```
typedef struct
{
   uint32_t       mode;          /* MMC Driver operation mode */
   uint32_t       voltage;       /* Operation voltage */
} mmc_cfg_t;
```

2.9.5 mmc_access_t 構造体定義

```
typedef struct
{
   uint8_t        *p_buff;
   uint32_t       lbn;
   int32_t        cnt;
   uint32_t       mode;
   uint32_t       rw_mode;
} mmc_access_t;
```

2.9.6 mmc_card_status_t 構造体定義

```
typedef struct
{
   uint32_t    card_sector_size; /* Sector size (user area) */
   uint8_t     csd_structure;    /* CSD structure
                                0 : CSD version No.1.0
                                1 : CSD version No.1.1
                                2 : CSD version No.1.2
                                3 : Version is coded in the CSD_STRUCTURE byte
                                         in the EXT_CSD register */
   uint8_t     speed_mode;       /* Card speed mode
                        Supported speed bit 5,4 : 0 0 Backward-compatible
                                                  0 1 High-speed 26MHz Max
                                                  1 1 High-speed 52MHz Max
                        Current speed   bit 1,0 : 0 0 Backward-compatible
                                                  0 1 High-speed 26MHz Max
                                                  1 1 High-speed 52MHz Max */
   uint8_t     csd_spec;         /* MMC spec version */
   uint8_t     if_mode;          /* Bus width */
   uint8_t     density_type;     /* Card density type */
   uint8_t     rsv[3];           /* Reserve */
} mmc_card_status_t;
```

2.9.7 mmc_card_reg_t 構造体定義

```
typedef struct
{
   uint32_t ocr[1];          /* OCR value */
   uint32_t cid[4];          /* CID value */
   uint32_t csd[4];          /* CSD value */
   uint32_t dsr[1];          /* DSR value */
   uint32_t rca[1];          /* RCA value */
} mmc_card_reg_t;
```

2.10 戻り値／エラーコード

API 関数の戻り値を示します。この列挙型は、API 関数のプロトタイプ宣言とともに r_mmcif_rx_if.h で記載されています。

MMCIF ドライバの API 関数は、その処理の途中でエラーが発生した場合、戻り値にエラーコードを返します。また、R_MMCIF_Mount()関数／R_MMCIF_Read_Memory()関数／R_MMCIF_Write_Memory()関数注 1 実行後に、R_MMCIF_Get_ErrCode()関数を使ってエラーコードを取得できます。

表 2-4 にエラーコードを示します。なお、表にない値は、将来のための予約です。

注 1：R_MMCIF_Read_Memory_Software_Trans()関数／R_MMCIF_Write_Memory_Software_Trans()関数も同様です。

表 2-4　エラーコード

マクロ定義	値	意味	
MMC_SUCCESS_LOCKED_CARD	1	正常終了（カードロック状態）	正常終了。但し、カードロック状態。
MMC_SUCCESS	0	正常終了	正常終了
MMC_ERR	-1	一般エラー	R_MMCIF_Open()関数が実行されていない、引数パラメータエラー等
MMC_ERR_WP	-2	ライトプロテクトエラー	ライトプロテクト状態の MMC への書き込み
MMC_ERR_HOST_TOE	-9	ホストタイムアウトエラー	r_mmcif_dev_int_wait()関数のエラー
MMC_ERR_CARD_LOCK	-11	カードロックエラー	R1 レスポンスのカードステータスエラー（CARD_IS_LOCKED）
MMC_ERR_CARD_UNLOCK	-12	カードアンロックエラー	R1 レスポンスのカードステータスエラー（LOCK_UNLOCK_FAILED）
MMC_ERR_CARD_CRC	-13	カード CRC エラー	R1 レスポンスのカードステータスエラー（COM_CRC_ERROR）
MMC_ERR_CARD_ECC	-14	カード ECC エラー	R1 レスポンスのカードステータスエラー（CARD_ECC_FAILED）
MMC_ERR_CARD_CC	-15	カード CC エラー	R1 レスポンスのカードステータスエラー（CC_ERROR）
MMC_ERR_CARD_ERROR	-16	カードエラー	R1 レスポンスのカードステータスエラー（ERROR）
MMC_ERR_NO_CARD	-18	カード未挿入エラー	カードが挿入されていない
MMC_ERR_CPU_IF	-30	ターゲット MCU インタフェース関数エラー	ターゲット MCU インタフェース関数エラー（r_mmcif_dev_int_wait()関数以外）
MMC_ERR_STOP	-31	強制停止エラー	R_MMCIF_Control()関数による強制停止状態
MMC_ERR_CMD	-32	コマンド発行エラー	MMCIF 内部エラー（コマンド発行）
MMC_ERR_BUFACC	-33	MMCIF バッファアクセスエラー	MMCIF 内部エラー（MMCIF バッファへの不正アクセス）

MMC_ERR_WRITE	-34	書き込みデータエラー	MMCIF 内部エラー （CRC status トークンのステータスまたはエンドビット）
MMC_ERR_READ	-35	読み出しデータエラー	MMCIF 内部エラー （読み出しデータの CRC16 またはエンドビット）
MMC_ERR_RESPIND	-36	レスポンスインデックスエラー	MMCIF 内部エラー （レスポンスの command index フィールド値または check bits フィールド値にエラー）
MMC_ERR_RESP	-37	レスポンスエラー	MMCIF 内部エラー （レスポンスまたは Boot Acknowledge にエラー）
MMC_ERR_CRC_TOE	-38	CRC status タイムアウトエラー	MMCIF 内部エラー （CRC status トークンが受信できない場合）
MMC_ERR_WRITE_TOE	-39	書き込みデータタイムアウトエラー	MMCIF 内部エラー （データ書き込み時にビジー状態が指定期間以上続いた場合）
MMC_ERR_READ_TOE	-40	読み出しデータタイムアウトエラー	MMCIF 内部エラー （指定期間内にデータを受信できない場合）
MMC_ERR_RESPB_TOE	-41	レスポンスビジータイムアウトエラー	MMCIF 内部エラー （レスポンスのビジー状態が指定期間以上続いた場合）
MMC_ERR_RESP_TOE	-42	レスポンスタイムアウトエラー	MMCIF 内部エラー （レスポンスまたは Boot Acknowledge が指定期間内に受信できない場合）
MMC_ERR_FAST_IO	-43	Fast I/O レスポンスエラー	Fast I/O データ異常
MMC_ERR_CHANGE_BUS	-44	バステストエラー	バステスト処理時のエラー
MMC_ERR_SWITCH	-47	SWITCH コマンドエラー	R1 レスポンスのカードステータスエラー（SWITCH_ERROR）
MMC_ERR_CSD_RLEN	-49	READ_BL_LEN エラー	CSD レジスタ[83:80]ビット 最大読み出しブロック長エラー
MMC_ERR_CSD_VER	-50	CSD Ver.エラー	CSD Ver.5.0 以降の Ver.エラー 但し、本エラー判定は行わない。
MMC_ERR_CSD_WLEN	-54	WRITE_BL_LEN エラー	CSD レジスタ[25:22]ビット 最大書き込みブロック長エラー

MMC_ERR_OUT_OF_RANGE	-80	引数範囲外エラー	R1 レスポンスのカードステータスエラー（OUT_OF_RANGE）
MMC_ERR_ADDRESS_ERROR	-81	アドレスエラー	R1 レスポンスのカードステータスエラー（ADDRESS_ERROR）
MMC_ERR_BLOCK_LEN_ERROR	-82	ブロック長エラー	R1 レスポンスのカードステータスエラー（BLOCK_LEN_ERROR）
MMC_ERR_ILLEGAL_COMMAND	-83	異常コマンドエラー	R1 レスポンスのカードステータスエラー（ILLEGAL_COMMAND）
MMC_ERR_CBSY_ERROR	-87	コマンドエラー	MMCIF 内部エラー （コマンドビジー）
MMC_ERR_NO_RESP_ERROR	-88	レスポンスなしエラー	MMCIF 内部エラー （レスポンスを受信できない）
MMC_ERR_ADDRESS_BOUNDARY	-89	バッファアドレスエラー	引数のバッファアドレスエラー アドレスが 4 バイト境界でない
MMC_ERR_INTERNAL	-99	内部エラー	ドライバ内部エラー

2.11 コールバック関数

本ドライバでは、MMC カードの挿抜割り込み、又は MMC プロトコルステータス割り込みが発生したタイミングで、ユーザが設定したコールバック関数を呼び出します。

コールバック関数の登録方法は「3.15 R_MMCIF_Cd_Int()」「3.16 R_MMCIF_IntCallback()」を参照してください。

コールバック関数の発生タイミングは「1.4.2(5) MMCIF ステータス確認」を参照してください。

2.12 FIT モジュールの追加方法

本ドライバは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(2)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(3)の方法を使用してください。

(1) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(3) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

2.13 for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

以下に記述例を示します。

while 文の例：

```
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}
```

for 文の例：

```
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

do while 文の例：

```
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /*
WAIT_LOOP */
```

3. API 関数

R_MMCIF_Open()

MMCIF ドライバの API を使用する際、最初に使用する関数です。

Format

mmc_status_t R_MMCIF_Open(
    uint32_t channel,
    void *p_mmc_WorkArea
)

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_mmc_WorkArea*

4 バイト境界のワーク領域のポインタ（領域サイズ 164 バイト）

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*
MMC_ERR_CPU_IF	*ターゲット MCU インタフェース関数エラー*
MMC_ERR_ADDRESS_BOUNDARY	*引数のバッファアドレスエラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

引数 channel で設定した MMCIF チャネルリソースを取得し、MMCIF ドライバと MMCIF チャネルを初期化します。また、その MMCIF チャネルリソースを占有します。

MMCIF ドライバのクローズ処理を終了させるまで、ワーク領域を保持し、その内容をアプリケーションプログラムで変更しないでください。

Example

```
uint32_t        g_mmcif_work[164/sizesof(uint32_t)];

/* ==== Please add the processing to set the pins. ==== */

if (R_MMCIF_Open(MMC_CH0, &g_mmcif_work) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

本関数実行前に、端子設定が必要です。「4.4 MMC カードの挿入と電源投入タイミング」を参照してください。

本関数が正常終了しない場合、R_MMCIF_GetVersion()関数、R_MMCIF_Log()関数、R_MMCIF_Set_LogHdlAddress()関数以外のライブラリ関数が使用できません。

本関数が正常終了した場合、挿抜割り込みを許可できます。MMC カード挿抜割り込みを使用する場合は、本関数実行後、R_MMCIF_Cd_Int()関数にて挿抜割り込みを許可にしてください。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

本関数実行前後で、端子の状態は変化しません。

R_MMCIF_Close()

使用中の MMCIF ドライバのリソースを開放する関数です。

Format

mmc_status_t R_MMCIF_Close(
　uint32_t channel
)

Parameters

channel
　チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMCIF ドライバの全ての処理を終了し、引数 channel で設定した MMCIF チャネルのリソースを解放します。

その MMCIF チャネルをモジュールストップ状態に設定します。

本関数実行後、挿抜割り込みは禁止状態になります。

R_MMCIF_Open()関数で設定したワーク領域は、本関数実行後は使用されません。他用途に使用できます。

Example

```
/* ==== Please add the processing to set the pins. ==== */

if (R_MMCIF_Close(MMC_CH0) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

本関数実行後に、端子設定が必要です。「4.5 MMC カードの抜去と電源停止タイミング」を参照してください。また、本関数実行前に R_MMCIF_Open()関数によるドライバのオープン処理が必要です。

本関数実行前後で、端子の状態は変化しません。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Get_CardDetection()

MMC カードの挿入状態を確認する関数です。

Format

```
mmc_status_t R_MMCIF_Get_CardDetection(
    uint32_t channel
)
```

Parameters

channel
　チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

Return Values

MMC_SUCCESS	*MMC_CD 端子レベルは Low、またはカード検出無効時*
MMC_ERR	*MMC_CD 端子レベルは High*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC カードの挿入状態を確認します。

＜MMC_CFG_CHx_CD_ACTIVE == 1（カード検出有効）の場合＞

MMC_CD 端子レベルが Low の場合、MMC_SUCCESS を返します。

MMC_CD 端子レベルが High の場合、MMC_ERR を返します。

＜MMC_CFG_CHx_CD_ACTIVE == 0（カード検出無効）の場合＞

常に MMC_SUCCESS を返します。

Example

```
mmc_status_t r_mmcif_pin_check_card_detection(uint32_t channel, uint8_t
derection)
{
#if (MMC_CFG_DRIVER_MODE & MMC_MODE_MMC)
   mmc_status_t       ret_old = MMC_ERR;
   mmc_status_t       ret_new = MMC_ERR;
   uint32_t           chat_cnt = MMC_CFG_CHAT_CNT;
   uint32_t           loop_cnt = MMC_CHAT_LOOP_CNT;

   /* ==== Check card insertion ==== */
   if (MMC_CD_REMOVE == derection)
   {
      ret_old = MMC_SUCCESS;
      ret_new = MMC_SUCCESS;
   }
   do
   {
      ret_new = R_MMCIF_Get_CardDetection(MMC_CH0);
      if (ret_new != ret_old) /* Status change */
      {
         if (((MMC_SUCCESS == ret_new) && (MMC_CD_INSERT == derection)) ||
             ((MMC_SUCCESS != ret_new) && (MMC_CD_REMOVE == derection)))
         {
            chat_cnt--;
            if (0 == chat_cnt)
            {
               return MMC_SUCCESS;
            }

            if (true != r_mmcif_pin_softwaredelay(1, MMC_DELAY_MILLISECS))
            {
               return MMC_ERR;
            }
         }
         else
         {
            chat_cnt = MMC_CFG_CHAT_CNT;
         }
      }
      else
      {
         chat_cnt = MMC_CFG_CHAT_CNT;
      }
      loop_cnt--;
   }
   while(0 != loop_cnt);
   return MMC_ERR;
#else
   return MMC_SUCCESS;
#endif /* (MMC_CFG_DRIVER_MODE & MMC_MODE_MMC) */
}
```

Special Notes

カード挿入検出で使用する場合、本関数実行後に、端子設定が必要です。「4.4 MMC カードの挿入と電源投入タイミング」を参照してください。また、本関数実行前に R_MMCIF_Open()関数によるドライバのオープン処理が必要です。

カード抜去検出で使用する場合、本関数実行前に、端子設定が必要です。「4.5 MMC カードの抜去と電源停止タイミング」を参照してください。

また、本関数実行前に R_MMCIF_Open()関数による初期化処理が必要です。

MMC カード挿抜検出端子として、MMC カードソケットの CD 端子に接続した MMC_CD 端子を使用します。

MMCIF には、MMC カードを挿抜したときに発生するチャタリングをハードウェアで除去する機能はありません。本 Example を参考にチャタリングを考慮したカード検出処理を作成してください。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

MMC_CD 端子の処理方法は、「4.6 ハードウェア設定」を参照してください。

MMC カード検出後、MMC カードへの電源電圧供給処理が必要です。

R_MMCIF_Mount()

MMC を初期化し、マウント可能状態からドライバアイドル状態にする関数です。

Format

```
mmc_status_t R_MMCIF_Mount(
    uint32_t channel,
    mmc_cfg_t *p_mmc_Config
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

** p_mmc_Config*

動作設定情報構造体

mode ：動作モード

「表 3-1　MMCIF ドライバ　動作モード mode」のマクロ定義に示す種別毎の論理和で動作モードを設定してください。

voltage ：電源電圧

MMC カードに供給する電源電圧（設定値は「表 3-2　電源電圧 voltage」のマクロ定義を参照）を設定してください。設定した電源電圧で動作できない MMC カードは初期化されません。「1.4.2(3) マウント時の動作電圧設定について」も参照してください。

表 3-1　MMCIF ドライバ　動作モード mode

種別	マクロ定義	値（ビット）	定義
ステータス確認方式	MMC_MODE_POLL	0x0000	Software polling
	MMC_MODE_HWINT	0x0001	Hardware interrupt
データ転送方式	MMC_MODE_SW	0x0000	Software 転送
	MMC_MODE_DMA（注 1、注 4）	0x0002	DMAC 転送（注 3）
	MMC_MODE_DTC（注 2、注 4）	0x0004	DTC 転送（注 3）
メディア対応方式	MMC_MODE_MMC	0x0000	MMC カード
	MMC_MODE_eMMC	0x0020	eMMC
MMC バス対応方式	MMC_MODE_1BIT（注 5）	0x0100	MMC モード 1 ビットバス
	MMC_MODE_4BIT（注 5）	0x0400	MMC モード 4 ビットバス
	MMC_MODE_8BIT（注 5）	0x0800	MMC モード 8 ビットバス

注 1：別途 DMAC 制御プログラムが必要です。

注 2：別途 DTC 制御プログラムが必要です。

注 3：使用するライブラリ関数によっては Software 転送を行います。

注 4：MMC_MODE_DMA と MMC_MODE_DTC を同時にセットしないでください。

注 5：MMC_MODE_1BIT、MMC_MODE_4BIT、MMC_MODE_8BIT のうち、１つをセットしてください。
MMC_MODE_8BIT の場合、バステストにより、8 ビット→4 ビット→1 ビットバスの順で接続を行います。
MMC_MODE_4BIT の場合、バステストにより、4 ビット→1 ビットバスの順で接続を行います。
MMC_MODE_1BIT の場合、バステストにより、1 ビットバスで接続を行います。

表 3-2　電源電圧 voltage

電源電圧[V]	マクロ定義	値（ビット）
2.7-2.8	MMC_VOLT_2_8	0x00008000
2.8-2.9	MMC_VOLT_2_9	0x00010000
2.9-3.0	MMC_VOLT_3_0	0x00020000
3.0-3.1	MMC_VOLT_3_1	0x00040000
3.1-3.2	MMC_VOLT_3_2	0x00080000
3.2-3.3	MMC_VOLT_3_3	0x00100000
3.3-3.4	MMC_VOLT_3_4	0x00200000
3.4-3.5	MMC_VOLT_3_5	0x00400000
3.5-3.6	MMC_VOLT_3_6	0x00800000

表 3-3　MMCIF 制御端子

端子名	eMMC 1 ビットモード	eMMC 4 ビットモード	eMMC 8 ビットモード	MMC カード 1 ビットモード	MMC カード 4 ビットモード	MMC カード 8 ビットモード
MMC_CLK	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御
MMC_CMD	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御
MMC_D0	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御
MMC_D1	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御
MMC_D2	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御
MMC_D3	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御	MMCIF 制御
MMC_D4	対象外	対象外	MMCIF 制御	対象外	対象外	MMCIF 制御
MMC_D5	対象外	対象外	MMCIF 制御	対象外	対象外	MMCIF 制御
MMC_D6	対象外	対象外	MMCIF 制御	対象外	対象外	MMCIF 制御
MMC_D7	対象外	対象外	MMCIF 制御	対象外	対象外	MMCIF 制御
MMC_CD	対象外	対象外	対象外	MMCIF 制御	MMCIF 制御	MMCIF 制御

Return Values

MMC_SUCCESS	*正常終了*
MMC_SUCCESS_LOCKED_CARD	*正常終了、かつ、MMC カードはロック状態*
上記以外	*エラー終了（詳細はエラーコードを参照ください）*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC のマウント処理を行います。MMC カードの検出後に、本関数を実行してください。

戻り値が、MMC_SUCCESS の場合、MMC は Transfer State (tran)へ遷移し、MMC のリード／ライトアクセスが可能になります。MMC_SUCCESS_LOCKED_CARD の場合、MMC は Transfer State (tran)へ遷移しますが、MMC リード／ライトアクセスはできません。別途、ロック状態を解除してください。

Example

```
mmc_cfg_t       mmc_Config;

/* ==== Please add the processing to set the pins. ==== */

mmc_Config.mode = MMC_CFG_DRIVER_MODE;
mmc_Config.voltage = MMC_VOLT_3_3;
if (R_MMCIF_Mount(MMC_CH0, &mmc_Config) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

MMCIF ドライバは、High-speed mode と Backward-compatible mode を判別し、マウントします。

本関数実行前に、端子設定が必要です。「4.4 MMC カードの挿入と電源投入タイミング」を参照してください。また、本関数実行前に R_MMCIF_Open()関数による初期化処理が必要です。

エラー終了の場合、R_MMCIF_Unmount()関数をコールし、アンマウント状態にした後、再度マウント処理を行ってください。

マウント正常終了後、再マウント処理を行う前に、アンマウント処理を行ってください。

p_mmc_Config の voltage に 2.7-3.6V の任意の値を設定した場合、動作電圧 2.7-3.6V として扱います。

R_MMCIF_Cd_Int()関数を使用する場合、p_mmc_Config の mode のステータス確認として MMC_MODE_HWINT を設定してください。

R_MMCIF_Unmount()

MMC のマウントを解除し、Transfer 状態からドライバアイドル可能状態にする関数です。

Format

```
mmc_status_t R_MMCIF_Unmount(
    uint32_t channel
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC のアンマウント処理を行います。また、Transfer 状態で本関数をコールした場合、MMC の extended CSD register を初期化します。

MMC カードの場合、MMC カードを取り外し可能な状態にします。また、本関数をコールし MMC カードをマウント解除状態にした場合であっても、MMC カードの挿抜割り込みおよび MMC カード挿抜確認用割り込みコールバック関数は有効です。

Example

```
if (R_MMCIF_Unmount(MMC_CH0) != MMC_SUCCESS)
{
    /* Error */
}

/* ==== Please add the processing to set the pins. ==== */
```

Special Notes

本関数実行後、MMC カードを抜去する場合、端子設定が必要です。「4.5 MMC カードの抜去と電源停止タイミング」を参照してください。また、本関数実行前に R_MMCIF_Open()関数による初期化処理を行ってください。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Read_Memory()

リード処理を実行する関数です。

Format

```
mmc_status_t R_MMCIF_Read_Memory(
    uint32_t channel,
    mmc_access_t *p_mmc_Access
)
```

Parameters

channel

　チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_mmc_Access*

　アクセス情報構造体

　**p_buff　：読み出しバッファポインタ*

　　4 バイト境界のアドレスを設定してください。

　lbn　：読み出し開始ブロック番号

　cnt　：ブロック数

　　設定できる最大値は、65,535 です。

　mode　：転送モード（設定不要：変更禁止）

　rw_mode　：読み出しモード

　　表 3-4 に設定値を示します。eMMC の場合、MMC_PRE_DEF を設定してください。MMC カードの場合は MMC_OPEN_END を設定してください。

表 3-4　MMCIF ドライバ　読み出しモード rw_mode

種別	マクロ定義	値（ビット）	対象 MMC
Open-ended	MMC_OPEN_END	0x00000000	MMC カード
Pre-defined	MMC_PRE_DEF	0x00000001	eMMC

Return Values

MMC_SUCCESS　　*正常終了*

上記以外　　*エラー終了（詳細はエラーコードを参照ください）*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

引数 p_mmc_Access の lbn で設定したブロックから引数 p_mmc_Access の cnt ブロック分のデータを読み出し、引数 p_mmc_Access の p_buff に格納します。

本関数開始時に、MMC カードの抜去を検出した場合、処理を中止しエラー終了を返します。

本関数開始時に、R_MMCIF_Control()の MMC_SET_STOP（強制停止要求）コマンドによる強制停止要求を検出した場合、強制停止要求をクリアし、処理を中止しエラー終了を返します。

ブロックデータの読み出しには、以下のコマンドを使用します。

　1 ブロック：READ_SINGLE_BLOCK コマンド（CMD17）

　2 ブロック以上：READ_MULTIPLE_BLOCK コマンド（CMD18）

Example

```
#define TEST_BLOCK_CNT   (4)
#define BLOCK_NUM        (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_clear(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_r_buff[0];
mmc_Access.lbn     = 0x10000000;
mmc_Access.cnt     = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Read_Memory(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

リードのエラー終了の場合、再度リード処理を行うことを推奨します。

転送ブロック数が 65,535 を超える場合は、分割して、コールしてください。FAT ファイルシステム等上位アプリケーションプログラムからコールする場合に注意してください。

なお、1 ブロックのサイズは、512 バイトです。

R_MMCIF_Read_Memory_Software_Trans()

リード処理（Software 転送）を実行する関数です。

Format

mmc_status_t R_MMCIF_Read_Memory_Software_Trans(
　uint32_t channel,
　mmc_access_t *p_mmc_Access
)

Parameters

channel
　チャネル番号 - 使用する MMCIF チャネル番号（0 起算）
**p_mmc_Access*
　アクセス情報構造体
　**p_buff　：読み出しバッファポインタ*
　　アドレス境界の制限はありません。処理の高速化のため、4 バイト境界のアドレス設定を推奨します。
　lbn　：読み出し開始ブロック番号
　cnt　：ブロック数
　　設定できる最大値は、65,535 です。
　mode　：転送モード（設定不要：変更禁止）
　rw_mode　：読み出しモード
　　表 3-5 に設定値を示します。eMMC の場合、MMC_PRE_DEF を設定してください。MMC カードの場合は MMC_OPEN_END を設定してください。

表 3-5　MMCIF ドライバ　読み出しモード rw_mode

種別	マクロ定義	値（ビット）	対象 MMC
Open-ended	MMC_OPEN_END	0x00000000	MMC カード
Pre-defined	MMC_PRE_DEF	0x00000001	eMMC

Return Values

MMC_SUCCESS	*正常終了*
上記以外	*エラー終了（詳細はエラーコードを参照ください）*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

引数 p_mmc_Access の lbn で設定したブロックから引数 p_mmc_Access の cnt ブロック分のデータを読み出し、引数 p_mmc_Access の p_buff に格納します。

マウント処理時の動作モードのデータ転送設定に関わらず、Softrware 転送を行います。

本関数開始時に、MMC カードの抜去を検出した場合、処理を中止しエラー終了を返します。

本関数開始時に、R_MMCIF_Control()の MMC_SET_STOP（強制停止要求）コマンドによる強制停止要求を検出した場合、強制停止要求をクリアし、処理を中止しエラー終了を返します。

ブロックデータの読み出しには、以下のコマンドを使用します。

　1 ブロック：READ_SINGLE_BLOCK コマンド（CMD17）

　2 ブロック以上：READ_MULTIPLE_BLOCK コマンド（CMD18）

Example

```
#define TEST_BLOCK_CNT   (4)
#define BLOCK_NUM        (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_clear(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_r_buff[0];
mmc_Access.lbn     = 0x10000000;
mmc_Access.cnt     = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Read_Memory_Software_Trans(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

リードのエラー終了の場合、再度リード処理を行うことを推奨します。

転送ブロック数が 65,535 を超える場合は、分割して、コールしてください。FAT ファイルシステム等上位アプリケーションプログラムからコールする場合に注意してください。

なお、1 ブロックのサイズは、512 バイトです。

R_MMCIF_Write_Memory()

ライト処理を実行する関数です。

Format

```
mmc_status_t R_MMCIF_Write_Memory(
    uint32_t channel,
    mmc_access_t *p_mmc_Access
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_mmc_Access*

アクセス情報構造体

**p_buff ：書き込みバッファポインタ*

4 バイト境界のアドレスを設定してください。

lbn ：書き込み開始ブロック番号

cnt ：ブロック数

設定できる最大値は、65,535 です。

mode ：転送モード（設定不要：変更禁止）

rw_mode ：書き込みモード

表 3-6 に設定値を示します。eMMC の場合、MMC_PRE_DEF を設定してください。MMC カードの場合は MMC_OPEN_END を設定してください。

表 3-6　MMCIF ドライバ　書き込みモード rw_mode

種別	マクロ定義	値（ビット）	対象 MMC
Open-ended	MMC_OPEN_END	0x00000000	MMC カード
Pre-defined	MMC_PRE_DEF	0x00000001	eMMC

Return Values

MMC_SUCCESS　　*正常終了*

上記以外　　*エラー終了（詳細はエラーコードを参照ください）*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

引数 p_mmc_Access の lbn で設定したブロックから引数 p_mmc_Access の cnt ブロック分の領域に引数 p_mmc_Access の p_buff のデータを書き込みます。

本関数開始時に、MMC カードの抜去を検出した場合、処理を中止しエラー終了を返します。

本関数開始時に、R_MMCIF_Control()の MMC_SET_STOP（強制停止要求）コマンドによる強制停止要求を検出した場合、強制停止要求をクリアし、処理を中止しエラー終了を返します。

ブロックデータの書き込みには、以下のコマンドを使用します。

1 ブロック：WRITE_SINGLE_BLOCK コマンド（CMD24）

2 ブロック以上：WRITE_MULTIPLE_BLOCK コマンド（CMD25）

Example

```
#define TEST_BLOCK_CNT   (4)
#define BLOCK_NUM        (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_set(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_w_buff[0];
mmc_Access.lbn     = 0x10000000;
mmc_Access.cnt     = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Write_Memory(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

ライトのエラー終了の場合、再度ライト処理を行うことを推奨します。

転送ブロック数が 65,535 を超える場合は、分割して、コールしてください。FAT ファイルシステム等上位アプリケーションプログラムからコールする場合に注意してください。

なお、1 ブロックのサイズは、512 バイトです。

R_MMCIF_Write_Memory_Software_Trans()

ライト処理（Software 転送）を実行する関数です。

Format

```
mmc_status_t R_MMCIF_Write_Memory_Software_Trans(
    uint32_t channel,
    mmc_access_t *p_mmc_Access
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_mmc_Access*

アクセス情報構造体

**p_buff ：書き込みバッファポインタ*

アドレス境界の制限はありません。処理の高速化のため、4 バイト境界のアドレス設定を推奨します。

lbn ：書き込み開始ブロック番号

cnt ：ブロック数

設定できる最大値は、65,535 です。

mode ：転送モード（設定不要：変更禁止）

write_mode ：書き込みモード

表 3-7 に設定値を示します。eMMC の場合、MMC_PRE_DEF を設定してください。MMC カードの場合は MMC_OPEN_END を設定してください。

表 3-7 MMCIF ドライバ 書き込みモード rw_mode

種別	マクロ定義	値（ビット）	対象 MMC
Open-ended	MMC_OPEN_END	0x00000000	MMC カード
Pre-defined	MMC_PRE_DEF	0x00000001	eMMC

Return Values

MMC_SUCCESS	*正常終了*
上記以外	*エラー終了（詳細はエラーコードを参照ください）*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

引数 p_mmc_Access の lbn で設定したブロックから引数 p_mmc_Access の cnt ブロック分の領域に引数 p_mmc_Access の p_buff のデータを書き込みます。

マウント処理時の動作モードのデータ転送設定に関わらず、Softrware 転送を行います。

本関数開始時に、MMC カードの抜去を検出した場合、処理を中止しエラー終了を返します。

本関数開始時に、R_MMCIF_Control()の MMC_SET_STOP（強制停止要求）コマンドによる強制停止要求を検出した場合、強制停止要求をクリアし、処理を中止しエラー終了を返します。

ブロックデータの書き込みには、以下のコマンドを使用します。

1 ブロック：WRITE_SINGLE_BLOCK コマンド（CMD24）

2 ブロック以上：WRITE_MULTIPLE_BLOCK コマンド（CMD25）

Example

```
#define TEST_BLOCK_CNT   (4)
#define BLOCK_NUM        (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_set(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_w_buff[0];
mmc_Access.lbn     = 0x10000000;
mmc_Access.cnt     = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Write_Memory_Software_Trans(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

ライトのエラー終了の場合、再度ライト処理を行うことを推奨します。

転送ブロック数が 65,535 を超える場合は、分割して、コールしてください。FAT ファイルシステム等上位アプリケーションプログラムからコールする場合に注意してください。

なお、1 ブロックのサイズは、512 バイトです。

R_MMCIF_Control()

ドライバのコントロール処理を実行する関数です。

Format

```
mmc_status_t R_MMCIF_Control(
    uint32_t channel,
    mmc_cmd_t *p_mmc_Cmd
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_mmc_Cmd*

コントロール情報構造体

cmd　：コマンドマクロ定義

mode　：モード

**p_buff　：送信バッファポインタ*

size　：送信サイズ

Return Values

MMC_SUCCESS　　*正常終了*

上記以外　　*エラー終了（詳細はエラーコードを参照ください）*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC カードの制御ユーティリティです。

制御可能なコマンドを「表 3-8　コマンド一覧」に示します。次ページ以降にコマンド毎に詳細を示します。

表 3-8　コマンド一覧

コマンドマクロ定義 cmd	モード mode	送信内容 *p_buff	送信サイズ size	制御内容
MMC_SET_STOP （強制停止要求コマンド）	設定無効	設定無効	設定無効	強制停止要求状態に遷移 リード／ライト処理実行中に本関数コールにより、強制停止要求コマンドを発行した場合、転送処理の強制停止を要求します。

Example

次ページ以降にコマンド毎に示します。

Special Notes

本関数実行前に R_MMCIF_Open()関数によるドライバのオープン処理と R_MMCIF_Mount()関数による初期化処理が必要です。R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

(a) MMC_SET_STOP

リード／ライト処理を強制終了します。

Return Values

MMC_SUCCESS　　*正常終了*

Description

強制停止を要求し、MMCIF ドライバを強制停止状態に遷移させます。

アプリケーションプログラムによる処理を中断したい場合に、割り込み処理内からコールすることができます。

MMC に対してデータ転送途中であった場合、Transfer State (tran)に状態遷移させる目的で、MMC に対して CMD12 を発行し、転送途中でリード／ライト処理を強制終了し、エラー終了を返します。

なお、ライト処理中に本関数を実行した場合、CMD12 が発行され、MMC がビジー状態に遷移する場合があります。そのため次のリード／ライト関数コール時にエラー終了を返す場合があります。その場合、再度リード／ライト処理を行うこと推奨します。ライト中であった場合、MMC が Ready 状態になるまで時間待ちが必要です。

また、転送完了後以降のタイミングで強制停止要求された場合、強制停止要求状態のままリターンします。

Example

```
mmc_cmd_t       mmc_Cmd;

mmc_Cmd.cmd = MMC_SET_STOP;
mmc_Cmd.mode = 0;
mmc_Cmd.p_buff = 0;
mmc_Cmd.size = 0;

if (R_MMCIF_Control(MMC_CH0, &mmc_Cmd) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

ライト処理中に強制停止させた場合、MMC のデータは保証されません。

ライブラリ関数での強制停止要求確認ポイントは、以下のとおりです。

（1）リード／ライト処理開始後で、MMC へのコマンド発行前

（2）Software 転送時、512 バイトブロック単位の転送完了後で、次ブロック転送の前

（3）DMAC 転送／DTC 転送時、常時受け付け可能

また、強制停止要求のクリアは、以下の場合に行われます。

（1）R_MMCIF_Read_Memory()関数／R_MMCIF_Write_Memory()関数[注1]実行中に強制終了処理が行われた場合。

（2）強制停止状態で R_MMCIF_Read_Memory()関数／R_MMCIF_Write_Memory()関数[注1]をコールした場合。この場合、処理開始時に強制停止要求を検出し、処理を中止し、エラー終了を返します。

注 1 : R_MMCIF_Read_Memory_Software_Trans()関数／R_MMCIF_Write_Memory_Software_Trans()関数も同様です。

R_MMCIF_Get_ModeStatus()

転送モードステータス情報を取得する関数です。

Format

mmc_status_t R_MMCIF_Get_ModeStatus(
　　uint32_t channel,
　　uint8_t *p_mode
)

Parameters

channel

　チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_mode*

　モードステータス情報格納ポインタ（1 バイト）。値は、「表 3-1　MMCIF ドライバ　動作モード mode」のデータ転送のマクロ定義を参照してください。

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

データ転送モードステータス情報を取得し、モードステータス情報格納ポインタに格納します。

Example

```
uint8_t * p_mode;

if(R_MMCIF_Get_ModeStatus(MMC_CH0, p_mode) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Get_CardStatus()

カードステータス情報を取得する関数です。

Format

```
mmc_status_t R_MMCIF_Get_CardStatus(
    uint32_t channel,
    mmc_card_status_t *p_mmc_CardStatus
)
```

Parameters

channel

　チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_mmc_CardStatus*

　カードステータス情報構造体ポインタ

　card_sector_size　: ユーザ領域ブロック数

　csd_structure　: CSD register の CSD_STRUCTURE [127:126]

　　0 : CSD version No. 1.0
　　1 : CSD version No. 1.1
　　2 : CSD version No. 1.2（Version 4.1-4.2-4.3）
　　3 : Version is coded in the CSD_STRUCTURE byte in the EXT_CSD register

　speed_mode　: Speed mode

＜対象 MMC サポート mode＞		＜現在の転送 mode＞	
bit5	bit4	bit1	bit0
0	0 : Backward-compatible mode	0	0 : Backward-compatible mode
0	1 : High-speed mode 26MHz（最大）	0	1 : High-speed mode 26MHz（最大）
1	1 : High-speed mode 52MHz（最大）	1	1 : High-speed mode 52MHz（最大）

　csd_spec　: CSD register の SPEC_VERS [125:122]

　　0 : MMC_SPEC_10　/* MMC system spec: 1.0-1.2　*/
　　1 : MMC_SPEC_14　/* MMC system spec: 1.4　*/
　　2 : MMC_SPEC_20　/* MMC system spec: 2.0-2.2　*/
　　3 : MMC_SPEC_30　/* MMC system spec: 3.1-3.2-3.31　*/
　　4 : MMC_SPEC_40　/* MMC system spec: 4.0-4.1-4.2-4.3-4.4-4.41　*/

　if_mode　: Data bus width mode

　　0 : 1-bit
　　1 : 4-bit
　　2 : 8-bit

　density_type　: Access mode (OCR bit[30:29])

　　0 : Byte mode
　　1 : Sector mode

Return Values

MMC_SUCCESS　　*正常終了*
MMC_ERR　　*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC カードのカードステータス情報を取得し、カードステータス情報構造体に格納します。

Example

```
mmc_card_status_t     mmc_CardStatus;

if (R_MMCIF_Get_CardStatus(MMC_CH0, &mmc_CardStatus) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Get_CardInfo()

MMC カードレジスタ情報を取得する関数です。

Format

```
mmc_status_t R_MMCIF_Get_CardInfo(
    uint32_t channel,
    mmc_card_reg_t *p_mmc_CardReg
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_mmc_CardReg*

MMC のレジスタ情報構造体ポインタ

ocr[1]

OCR 情報

cid[4]

CID 情報

csd[4]

CSD 情報

dsr[1]

DSR 情報

rca[1]

RCA 情報

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC カードレジスタ情報を取得し、MMC カードのレジスタ情報構造体に格納します。

Example

```
mmc_card_reg_t    mmc_CardReg;

if (R_MMCIF_Get_CardInfo(MMC_CH0, &mmc_CardReg) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Int_Handler0()

割り込みハンドラです。

Format

```
void R_MMCIF_Int_Handler0(
	void *vect
)
```

Parameters

**vect*

ベクタテーブル

Return Values

なし

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMCIF ドライバの割り込みハンドラです。

MMCIF に対応する割り込み要因の処理ルーチンとしてシステムに組み込み済です。

MMC カード挿抜割り込み設定コールバック関数、およびステータス確認割り込みコールバック関数を登録している場合は、本関数内からコールバック関数をコールします。

Example

システムに組み込み済であため、設定不要です。

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

他チャネル使用時は、同様にチャネル毎に割り込みハンドラを作成してください。（例：チャネル 1 用 R_MMCIF_Int_Handler1()）

R_MMCIF_Int_Handler1()

割り込みハンドラです。

Format

void R_MMCIF_Int_Handler1(
    void *vect
)

Parameters

**vect*
ベクタテーブル

Return Values

なし

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMCIF ドライバの割り込みハンドラです。

MMCIF に対応する割り込み要因の処理ルーチンとしてシステムに組み込み済です。

MMC カード挿抜割り込み設定コールバック関数、およびステータス確認割り込みコールバック関数を登録している場合は、本関数内からコールバック関数をコールします。

Example

システムに組み込み済であため、設定不要です。

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

他チャネル使用時は、同様にチャネル毎に割り込みハンドラを作成してください。（例：チャネル 1 用 R_MMCIF_Int_Handler0()）

R_MMCIF_Cd_Int()

MMC カード挿抜割り込み（挿抜割り込みコールバック関数登録処理を含む）を設定する関数です。

Format

```
mmc_status_t R_MMCIF_Cd_Int(
      uint32_t channel,
      int32_t enable,
      mmc_status_t (*callback)(int32_t)
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

enable

MMC カード挿抜割り込みの禁止／許可設定

MMC_CD_INT_ENABLE を設定した場合は、MMC カード挿抜割り込みを許可します。

MMC_CD_INT_DISABLE を設定した場合は、MMC カード挿抜割り込みを禁止します。

callback

登録するコールバック関数

ヌルポインタを設定した場合、コールバック関数は登録されません。コールバック関数を使用する場合は、MMC カードの挿入前に本関数を実行しコールバック関数を登録してください。

(int32_t)には MMC_CD 端子の検出状態が格納されます。

0 : MMC_CD_INSERT（MMC_CD 端子の立ち下りを検出）

1 : MMC_CD_REMOVE（MMC_CD 端子の立ち上がりを検出）

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC カード挿抜割り込みを設定し、コールバック関数を登録します。

本関数で登録したコールバック関数は、割り込みハンドラのサブルーチンとして、MMC カード挿抜割り込み発生時にコールされます。

なお、MMC カード挿抜割り込みの許可／禁止設定に関わらず、R_MMCIF_Get_CardDetection()関数で MMC カードの挿抜状態を確認できます。

Example

```
uint32_t g_cd_int;

/* Callback function */
mmc_status_t r_mmcif_cd_callback(int32_t cd)
{
   g_cd_int = 1;
   /* ==== Disable card detect interrupt ==== */
   if (R_MMCIF_Cd_Int(MMC_CH0, MMC_CD_INT_DISABLE, 0) != MMC_SUCCESS)
   {
      /* Error */
   }
   return MMC_SUCCESS;
}

/* main */
void main(void)
{
   if (R_MMCIF_Cd_Int(MMC_CH0, MMC_CD_INT_ENABLE, r_mmcif_cd_callback) !=
MMC_SUCCESS)
   {
      /* Error */
   }

   /* ==== Check card insertion ==== */
   g_cd_int = 0;
   while (1)
   {
      if (1 == g_cd_int)
      {
         g_cd_int = 0;
         if (r_mmcif_pin_check_card_detection(MMC_CH0, MMC_CD_INSERT) ==
MMC_SUCCESS)
         {
            /* ==== Enable card detect interrupt ==== */
            if (R_MMCIF_Cd_Int(MMC_CH0, MMC_CD_INT_ENABLE,
r_mmcif_cd_callback) != MMC_SUCCESS)
            {
               /* Error */
            }
            break;
         }
         else
         {
            /* ==== Enable card detect interrupt ==== */
            if (R_MMCIF_Cd_Int(MMC_CH0, MMC_CD_INT_ENABLE,
r_mmcif_cd_callback) != MMC_SUCCESS)
            {
               /* Error */
            }
         }
      }
      else
      {
         /* Do nothing. */
      }
  }
}
```

Special Notes

カード検出を有効にするためには#define MMC_CFG_CHx_CD_ACTIVE を”1”に設定してください。

本関数実行前に R_MMCIF_Open()関数による初期化処理が必要です。

MMC カード挿抜割り込みは、本関数実行後に MMC カードの挿抜により発生します。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_IntCallback()

MMC プロトコルステータス割り込みコールバック関数を登録する関数です。

Format

```
mmc_status_t R_MMCIF_IntCallback(
    uint32_t channel,
    mmc_status_t (*callback)(int32_t)
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

callback

登録するコールバック関数

ヌルポインタを設定した場合、コールバック関数は登録されません。コールバック関数を使用する場合は、R_MMCIF_Mount()関数実行前にコールバック関数を登録してください。

(int32_t)には常に 0 が格納されます。

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC プロトコルステータス割り込みコールバック関数を登録します。

本関数で登録したコールバック関数は、割り込みハンドラのサブルーチンとして、MMC のプロトコルステータス変化による割り込み発生（ACCIO または ERRIO）時にコールされます。

Example

```
/* Callback function */
mmc_status_t r_mmcif_callback(int32_t cd)
{
    /* ACCIO, ERRIO */
    return MMC_SUCCESS;
}

if (R_MMCIF_IntCallback(MMC_CH0, r_mmcif_callback) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理が必要です。

登録したコールバック関数内でタスクの待ち状態の解除等の処理を行います。

本関数で登録するコールバック関数は、MMC カード挿抜割り込みコールバック関数と異なります。

本関数で登録したコールバック関数は、MMC カード挿抜割り込み発生時にはコールされません。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Get_ErrCode()

ドライバのエラーコードを取得する関数です。

Format

mmc_status_t R_MMCIF_Get_ErrCode(
　　uint32_t channel
)

Parameters

channel
　チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

Return Values

エラーコード - エラーコードを参照

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

R_MMCIF_Mount()関数／R_MMCIF_Read_Memory()関数／R_MMCIF_Write_Memory()関数注 1 の実行時に発生したエラーのエラーコードを返します。再びライブラリ関数を実行することで、エラーコードはクリアされます。

注 1 : R_MMCIF_Read_Memory_Software_Trans()関数／R_MMCIF_Write_Memory_Software_Trans()関数も同様

Example

```
mmc_cfg_t       mmc_Config;
mmc_status_t    error_code = MMC_SUCCESS;

/* ==== Please add the processing to set the pins. ==== */

mmc_Config.mode = MMC_CFG_DRIVER_MODE;
mmc_Config.voltage = MMC_VOLT_3_3;
if (R_MMCIF_Mount(MMC_CH0, &mmc_Config) != MMC_SUCCESS)
{
   /* Error */
   error_code = R_MMCIF_Get_ErrCode(MMC_CH0);
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理が必要です。

アプリケーションプログラムで MMCIF ドライバのエラーコードを取得する場合に使用してください。

R_MMCIF_Get_BuffRegAddress()

MMCIF のデータレジスタ（CEDATA）のアドレスを取得する関数です。

Format

mmc_status_t R_MMCIF_Get_BuffRegAddress(
　　uint32_t channel,
　　uint32_t *p_reg_buff
)

Parameters

channel
　チャネル番号 － 使用する MMCIF チャネル番号（0 起算）

**p_reg_buff*
　データレジスタ（CEDATA）アドレスポインタ

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

データレジスタ（CEDATA）のアドレスを取得し、バッファに格納します。

DMAC 転送／DTC 転送使用時のデータレジスタアドレスを設定する場合等に使用します。

Example

```
uint32_t     reg_buff = 0;

if (R_MMCIF_Get_BuffRegAddress(MMC_CH0, &reg_buff) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理が必要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Get_ExtCsd()

MMC の Extended CSD 情報を取得する関数です。

Format

mmc_status_t R_MMCIF_Get_ExtCsd(
    uint32_t channel,
    uint32_t *p_ext_csd_buffer
)

Parameters

channel
チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

**p_ext_csd_buffer*
Extended CSD 受信バッファポインタ（512 バイト）

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

MMC の Extended CSD 情報を、引数の p_ext_csd_buffer に格納します。

Example

```
uint8_t  g_mmc_extcsd[512];

if (R_MMCIF_Get_ExtCsd(MMC_CH0, &g_mmc_extcsd[0]) != MMC_SUCCESS)
{
    /* Error */
}
```

Special Notes:

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_1ms_Interval()

MMCIF ドライバの内部タイマカウンタをインクリメントする関数です。

Format

void R_MMCIF_1ms_Interval(
    void
)

Parameters

なし

Return Values

なし

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

関数が呼ばれる毎に内部タイマカウンタをインクリメントします。

Example

```
uint32_t    g_cmt_channel;

void r_cmt_callback(void * pdata)
{
   uint32_t channel;

   channel = (uint32_t)pdata;
   if (channel == g_cmt_channel)
   {
      R_MMCIF_1ms_Interval();
   }
}

/* Create CMT timer */
R_CMT_CreatePeriodic(1000, &r_cmt_callback, &g_cmt_channel);    /* 1ms */
```

Special Notes

必ず 1 ミリ秒毎に呼び出してください。但し、r_mmcif_dev.c の r_mmcif_dev_int_wait()および r_mmcif_dev_wait()を OS 処理に置き換える場合には不要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Set_DmacDtc_Trans_Flg()

DMAC/DTC 転送完了フラグをセットする関数です。

Format

```
mmc_status_t R_MMCIF_Set_DmacDtc_Trans_Flg(
    uint32_t channel,
    uint32_t flg
)
```

Parameters

channel

チャネル番号 - 使用する MMCIF チャネル番号（0 起算）

flg

DMAC/DTC 転送完了フラグ - MMC_SET_TRANS_STOP

Return Values

MMC_SUCCESS　*正常終了*

MMC_ERR　*一般エラー（チャネル異常）*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

DMAC/DTC 転送完了フラグをセットします。

DMAC/DTC 転送完了フラグの処理方法を「表 3-9 DMAC 転送／DTC 転送時のフラグ処理方法」に示します。転送状態により、DMAC/DTC 転送完了フラグの処理方法が異なります。

DMAC の場合、DMAC の転送完了時に発生する割り込みハンドラ内で、MMC_SET_TRANS_STOP をセットし、本関数をコールしてください。

DTC の場合、MMCIF MBFAI 割り込みハンドラ内で MMC_SET_TRANS_STOP をセットするため、ユーザ処理は不要です。

転送中にエラーが発生した場合、DMAC および DTC に関わらず、ユーザ側で MMC_SET_TRANS_STOP をセットし、本関数をコールしてください。

表 3-9　DMAC 転送／DTC 転送時のフラグ処理方法

データ転送	正常終了時	エラー終了時
DMAC 転送	転送中 DMAC の転送完了時に発生する割り込みハンドラ内で、R_MMCIF_Set_DmacDtc_Trans_Flg()を実行し、転送完了状態に設定してください。	転送中 ユーザ側で R_MMCIF_Set_DmacDtc_Trans_Flg()を実行し、転送完了状態に設定してください。
DTC 転送	転送完了 （DTC ハンドラ内で転送完了処理を実行） ユーザ処理は不要です。	同上

Example

＜DMAC 転送 正常終了時＞

```
void r_dmaca_callback(void)
{
   R_MMCIF_Set_DmacDtc_Trans_Flg(MMC_CH0, MMC_SET_TRANS_STOP);
}
```

＜転送エラー終了時＞

```
#define TEST_BLOCK_CNT   (4)
#define BLOCK_NUM        (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_clear(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_r_buff[0];
mmc_Access.lbn     = 0x10000000;
mmc_Access.cnt     = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Read_Memory(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
   R_MMCIF_Set_DmacDtc_Trans_Flg(MMC_CH0, MMC_SET_TRANS_STOP);
}
```

Special Notes

本関数実行前に R_MMCIF_Open()関数による初期化処理と R_MMCIF_Mount()関数によるマウント処理が必要です。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

R_MMCIF_Set_LogHdlAddress()

LONGQ FIT モジュールのハンドラアドレスを設定する関数です。

Format

mmc_status_t R_MMCIF_Set_LogHdlAddress(
    uint32_t user_long_que
)

Parameters

user_long_que
LONGQ FIT モジュールのハンドラアドレス

Return Values

MMC_SUCCESS　　　　　　　　　　*正常終了*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

LONGQ FIT モジュールのハンドラアドレスを MMCIF ドライバに設定します。

Example

```
#define MMC_USER_LONGQ_MAX           (8)    /* Max error log count*/
#define MMC_USER_LONGQ_BUFSIZE       (MMC_USER_LONGQ_MAX * 4)
                                            /* Error log buffer size */
#define MMC_USER_LONGQ_IGN_OVERFLOW  (1)    /* Ignore overflow of error log
buffer.*/

uint32_t            g_mmc_user_longq_buf[MMC_USER_LONGQ_BUFSIZE];
                                            /* Error log buffer */
static longq_hdl_t  p_mmc_user_long_que; /* LongQ handler */
longq_err_t         err = LONGQ_SUCCESS;
uint32_t            user_long_que = 0;

err = R_LONGQ_Open(g_mmc_user_longq_buf,
                   MMC_USER_LONGQ_BUFSIZE,
                   MMC_USER_LONGQ_IGN_OVERFLOW,
                   &p_mmc_user_long_que);
if (LONGQ_SUCCESS != err)
{
   /* Error */
}
user_long_que = (uint32_t)p_mmc_user_long_que;
if (R_MMCIF_Set_LogHdlAddress(user_long_que) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

LONGQ FIT モジュールを使用し、エラーログを取得するための準備処理です。R_MMCIF_Open()をコールする前に処理を実行してください。

別途 LONGQ FIT モジュールを組み込んでください。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

MMC_CFG_LONGQ_ENABLE が無効のときにこの関数が呼び出された場合、この関数はなにもしません。

R_MMCIF_Log()

エラーログを取得する関数です。

Format

uint32_t R_MMCIF_Log(
　uint32_t flg,
　uint32_t fid,
　uint32_t line
)

Parameters

flg
　0x00000001（固定値）

fid
　0x0000003f（固定値）

line
　0x00001fff（固定値）

Return Values

0　　　　*正常終了*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

エラーログを取得します。

エラーログ取得を終了する場合、コールしてください。

Example

```
#define USER_DRIVER_ID      (1)
#define USER_LOG_MAX        (63)
#define USER_LOG_ADR_MAX    (0x00001fff)

mmc_cfg_t       mmc_Config;

/* ==== Please add the processing to set the pins. ==== */

mmc_Config.mode = MMC_CFG_DRIVER_MODE;
mmc_Config.voltage = MMC_VOLT_3_3;
if (R_MMCIF_Mount(MMC_CH0, &mmc_Config) != MMC_SUCCESS)
{
   /* Error */
   R_MMCIF_Log(USER_DRIVER_ID, USER_LOG_MAX, USER_LOG_ADR_MAX);
}
```

Special Notes

別途 LONGQ FIT モジュールを組み込んでください。

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

MMC_CFG_LONGQ_ENABLE が無効のときにこの関数が呼び出された場合、この関数はなにもしません。

R_MMCIF_GetVersion()

ドライバのバージョン情報を取得する関数です。

Format

uint32_t R_MMCIF_GetVersion(
    void
)

Parameters

なし

Return Values

上位 2 バイト	*メジャーバージョン（10 進表示）*
下位 2 バイト	*マイナーバージョン（10 進表示）*

Properties

r_mmcif_rx_if.h にプロトタイプ宣言されています。

Description

ドライバのバージョン情報を返します。

Example

```
uint32_t version;
version = R_MMCIF_GetVersion();
```

Special Notes

R_MMCIF_Get_ErrCode()関数によるエラーコード取得はできません。

4. 端子設定

MMC FIT モジュールを使用するためには、マルチファンクションピンコントローラ（MPC）で周辺機能の入出力信号を端子に割り付ける（以下、端子設定と称す）必要があります。

e^2 studio の場合はスマート・コンフィグレータの端子設定機能を使用することができます。スマート・コンフィグレータの端子設定機能を使用すると、端子設定画面で選択したオプションに応じて、ソースファイルが出力されます。そのソースファイルで定義された関数を呼び出すことにより端子を設定できます。詳細は表 4-1 を参照してください。

端子設定の制御手順は「4.4 MMC カードの挿入と電源投入タイミング」と「4.5 MMC カードの抜去と電源停止タイミング」を参照してください。

表 4-1　スマート・コンフィグレータが出力する関数一覧

出力される関数名	機能
R_MMCIF_PinSetInit()	MMCIF 端子の初期設定を行います。 実行後は MMC_CD 端子のみ有効です。
R_MMCIF_PinSetTransfer()	MMCIF 端子を MMC コマンド発行可能状態にします。 実行後は全 MMCIF 端子が有効です。
R_MMCIF_PinSetDetection()	MMCIF 端子を MMC コマンド発行禁止状態にします。 実行後は MMC_CD 端子のみ有効です。
R_MMCIF_PinSetEnd()	MMCIF 制御無効状態にします。 実行後は全 MMCIF 端子が無効です。

4.1 MMC バス 1 ビット通信の端子設定

通信に使用する MMC バスが 1 ビットであっても、MMC バスの端子を 4 ビットに設定してください。MMC_D0 端子以外の MMC バスを制御せずに MMC コマンドを発行すると、MMC が SPI モードになる可能性があります。

4.2 MMC カード電源制御端子設定

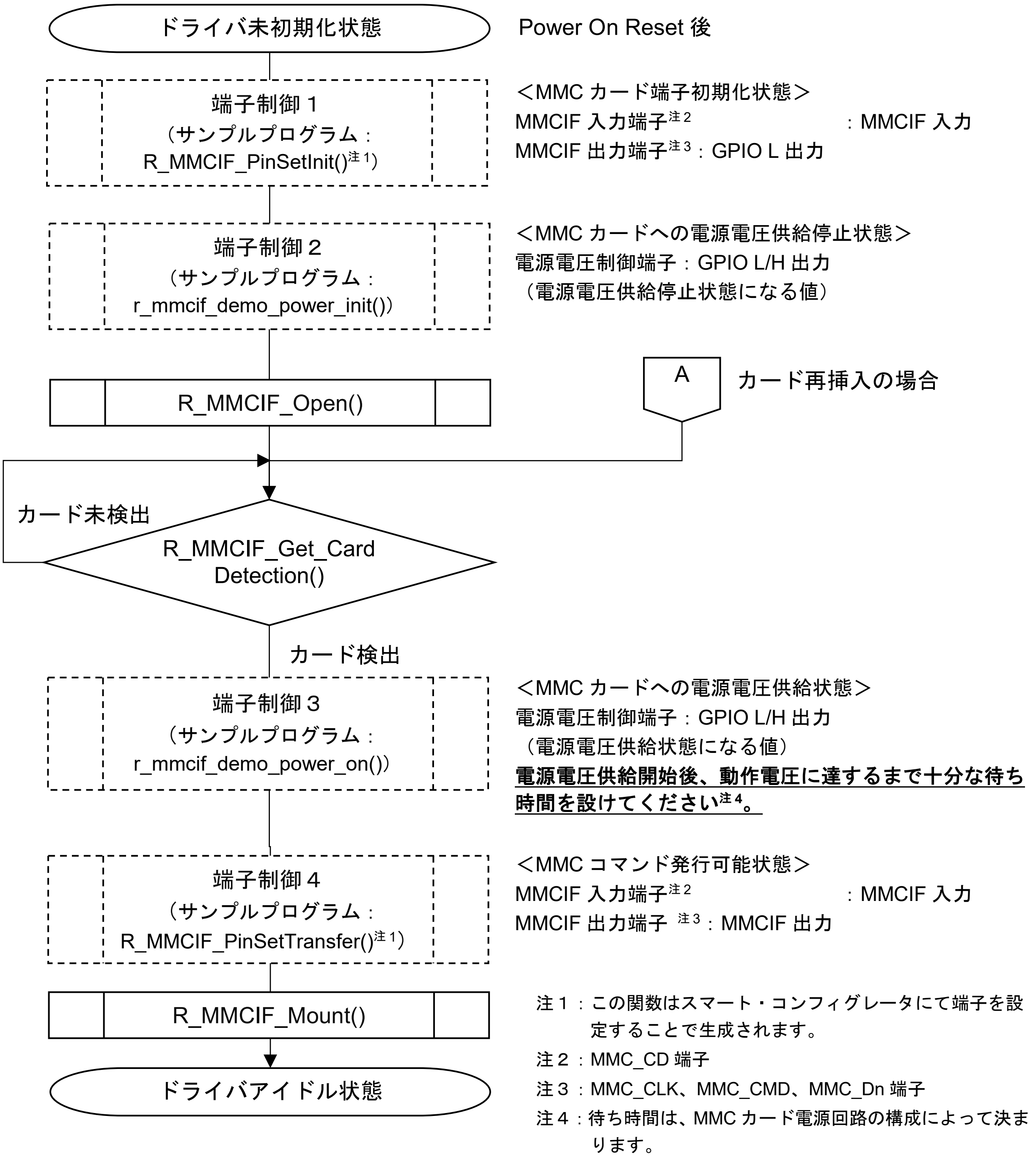
MMC カードの電源電圧制御端子はスマート・コンフィグレータの端子設定機能に含まれていないため、別途作成してください。

4.3 MMC リセット端子設定

MMC リセット端子（MMC_RES#）は MMCIF ドライバでは制御対象外のため、別途作成してください。

4.4 MMC カードの挿入と電源投入タイミング

制御手順を図 4-1、表 4-2 に示します。MMC カードの挿入は、R_MMCIF_Open()関数の正常終了後、MMC カードへの電源電圧供給停止状態、かつ MMCIF 出力端子を L 出力状態で行ってください。



注 1：この関数はスマート・コンフィグレータにて端子を設定することで生成されます。
注 2：MMC_CD 端子
注 3：MMC_CLK、MMC_CMD、MMC_Dn 端子
注 4：待ち時間は、MMC カード電源回路の構成によって決まります。

図 4-1 MMC カードの挿入と電源投入タイミング

表 4-2　MMC カード挿入時のユーザ設定方法

処理	対象端子	端子設定	実行後の端子状態
端子制御 1	MMCIF 入力端子 注 1	PMR 設定：汎用入出力ポート PCR 設定：入力プルアップ抵抗無効 注 3 PDR 設定：入力 MPC 設定：MMCIF PMR 設定：周辺モジュール	MMCIF 入力 （MMC カード検出可能状態）
	MMCIF 出力端子 注 2	PMR 設定：汎用入出力ポート DSCR 設定：高駆動出力 PCR 設定：入力プルアップ抵抗無効 注 3 PODR 設定：L 出力 PDR 設定：出力 MPC 設定：Hi-z	GPIO L 出力
端子制御 2	電源電圧制御端子	PMR 設定：汎用入出力 PCR 設定：入力プルアップ抵抗無効 注 4 PODR 設定：L 出力／H 出力（電源電圧供給停止状態になる値を出力） PDR 設定：出力	GPIO L/H 出力 （電源電圧供給停止状態）
端子制御 3	電源電圧制御端子	PODR 設定：L 出力／H 出力（電源電圧供給状態になる値を出力）	GPIO L/H 出力 （電源電圧供給状態）
端子制御 4	MMCIF 入力端子 注 1	MPC 設定：MMCIF PMR 設定：周辺モジュール	MMCIF 入力
	MMCIF 出力端子 注 2	MPC 設定：MMCIF PMR 設定：周辺モジュール	MMCIF 出力 （MMC コマンド発行可能状態）

注 1：MMC_CD 端子

注 2：MMC_CLK、MMC_CMD、MMC_Dn 端子

注 3：MCU 外部でプルアップされることを想定しているため、MCU 内蔵プルアップは無効にしてください。

注 4：システムに合わせて設定を見直してください。

4.5 MMC カードの抜去と電源停止タイミング

制御手順を図 4-2、表 4-3 に示します。MMC カードの抜去は、ドライバアイドル状態でのR_MMCIF_Unmount()関数の正常終了後、MMC カードへの電源電圧供給停止状態で行ってください。また、意図せず MMC カードが抜去された場合でも、同様の手順で電源電圧供給を停止してください。

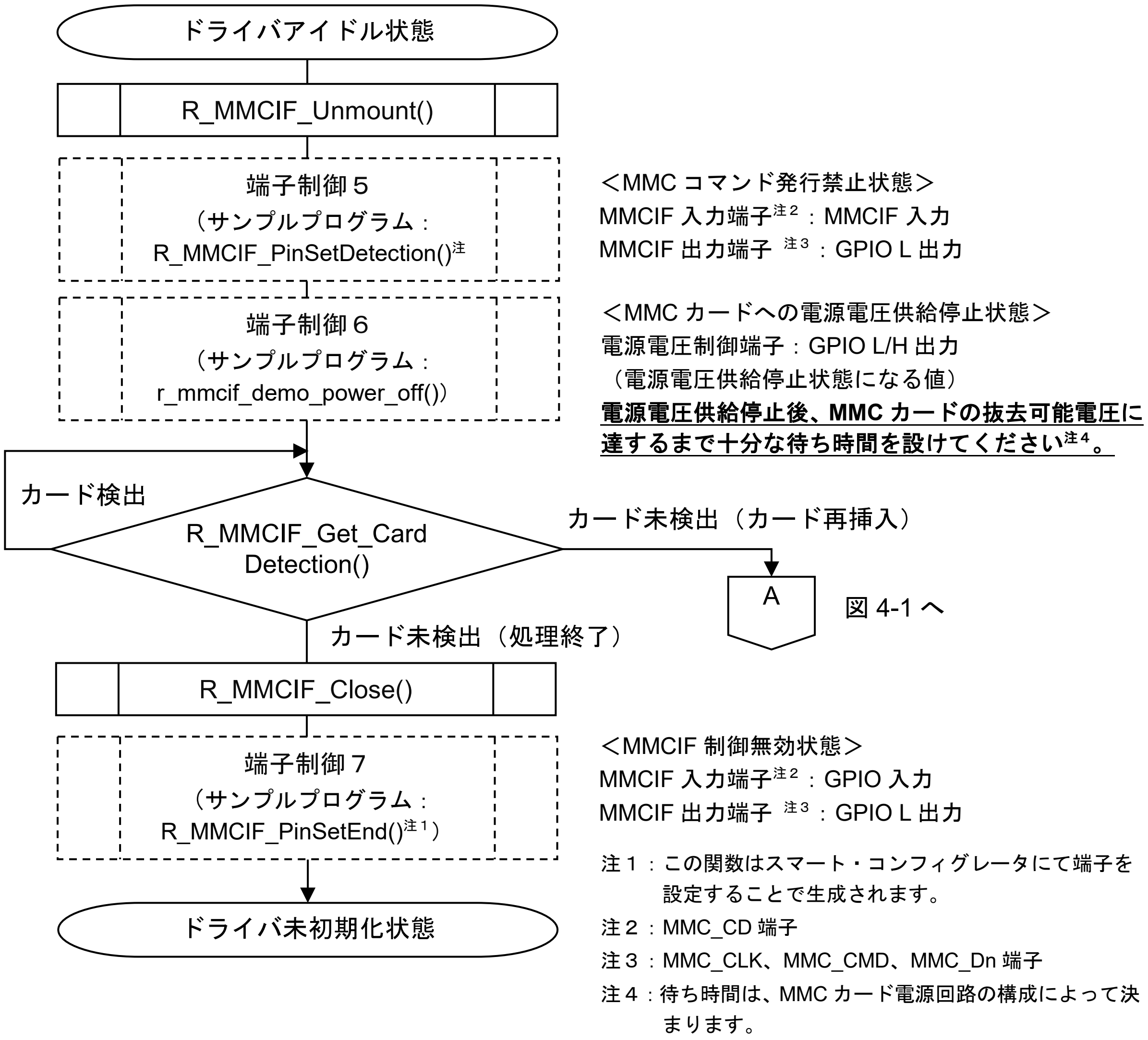


図 4-2　MMC カードの抜去と電源停止タイミング

表 4-3　MMC カード抜去時のユーザ設定方法

処理	対象端子	端子設定	実行後の端子状態
端子制御５	MMCIF 入力端子　注 1	MPC 設定：MMCIF PMR 設定：周辺モジュール	MMCIF 入力
	MMCIF 出力端子　注 2	PMR 設定：汎用入出力ポート MPC 設定：Hi-z	GPIO L 出力
端子制御６	電源電圧制御端子	PODR 設定：L 出力／H 出力（電源電圧供給停止状態になる値を出力）	GPIO L/H 出力 （電源電圧供給停止状態）
端子制御７	MMCIF 入力端子　注 1	PMR 設定：汎用入出力ポート MPC 設定：Hi-z	GPIO 入力
	MMCIF 出力端子　注 2	PMR 設定：汎用入出力ポート MPC 設定：Hi-z	GPIO L 出力

注 1：MMC_CD 端子

注 2：MMC_CLK、MMC_CMD、MMC_Dn 端子

4.6 ハードウェア設定

MCU 内蔵 MMCIF を使って、1 ビット／4 ビット／8 ビットバスの MMC モード制御を行います。

接続可能な MMC は、1 台／チャネルです。

4.6.1 ハードウェア構成例

「4.6.2」「4.6.3」に接続図を示します。

プルアップ抵抗値は、JEDEC STANDARD JESD84 を参照して決めてください。また、高速で動作させた場合を想定し、各信号ラインの回路的マッチングを取るためのダンピング抵抗やコンデンサの付加を検討してください。

4.6.1.1 端子説明

MMC_CLK 端子

プルアップ処理は、JEDEC STANDARD JESD84 に規定が無いため、記載していません。

MMC_CMD 端子

MMC は、Card identification mode 時、CMD ラインがオープンドレインになります。そのため、JEDEC STANDARD JESD84 では、CMD と DAT0～DTA7 のプルアップ抵抗規格を別々に規定しています。使用する環境に合わせて、プルアップ抵抗を決定してください。

MMC_CD 端子

MMC カードを使用する場合、MMC_CD 端子に MMC カード未挿入時に H が入力、MMC カード挿入時に L が入力されるように回路を構成してください。

MMC_CD 端子の制御は、r_mmcif_rx_config.h で設定してください。制御を無効にした場合、MMC_CD 端子は、MMC 以外の周辺機能として使用できます。

表 4-4　MMC_CD 端子の#define 定義設定 r_mmcif_rx_config.h

#define MMC_CFG_CHx_CD_ACTIVE	MMC_CD 端子	ターゲット MMC
(0)	制御しない	eMMC
(1)	制御する	MMC カード

MCCRES#端子

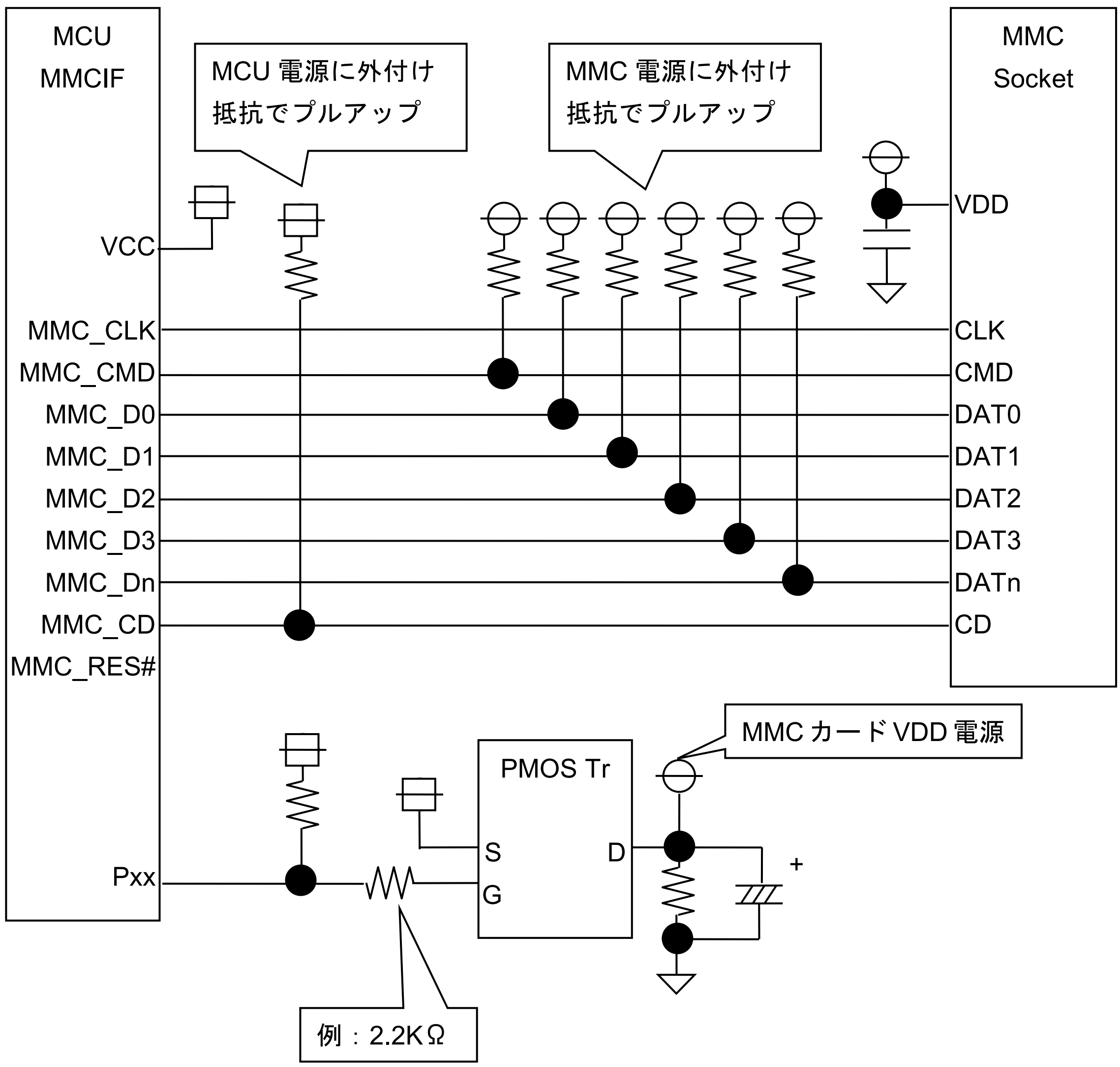
MMC_RES#端子は制御しません。MMC 以外の周辺機能として使用できます。但し、Extended CSD の RST_n_FUNCTION [162]の Bit[1:0]は 0x0（default）でご使用ください。

MMC 用電源電圧制御端子

MMC 用電源電圧制御が必要な場合、外付け PMOS Tr 等で回路を構成してください。電源－GND 間に十分大きな容量のコンデンサと電荷放電用抵抗を付けてください。MCU 端子にゲート容量の大きい PMOS Tr を直接制御する場合、MCU の電気的特性（出力 Low レベル許容電流／出力 High レベル許容電流）を参照し、MCU 端子－PMOS Tr ゲート間に電流制限抵抗を入れてください。

4.6.2 MMC（リムーバブルメディア：MMC カード）ソケットの場合

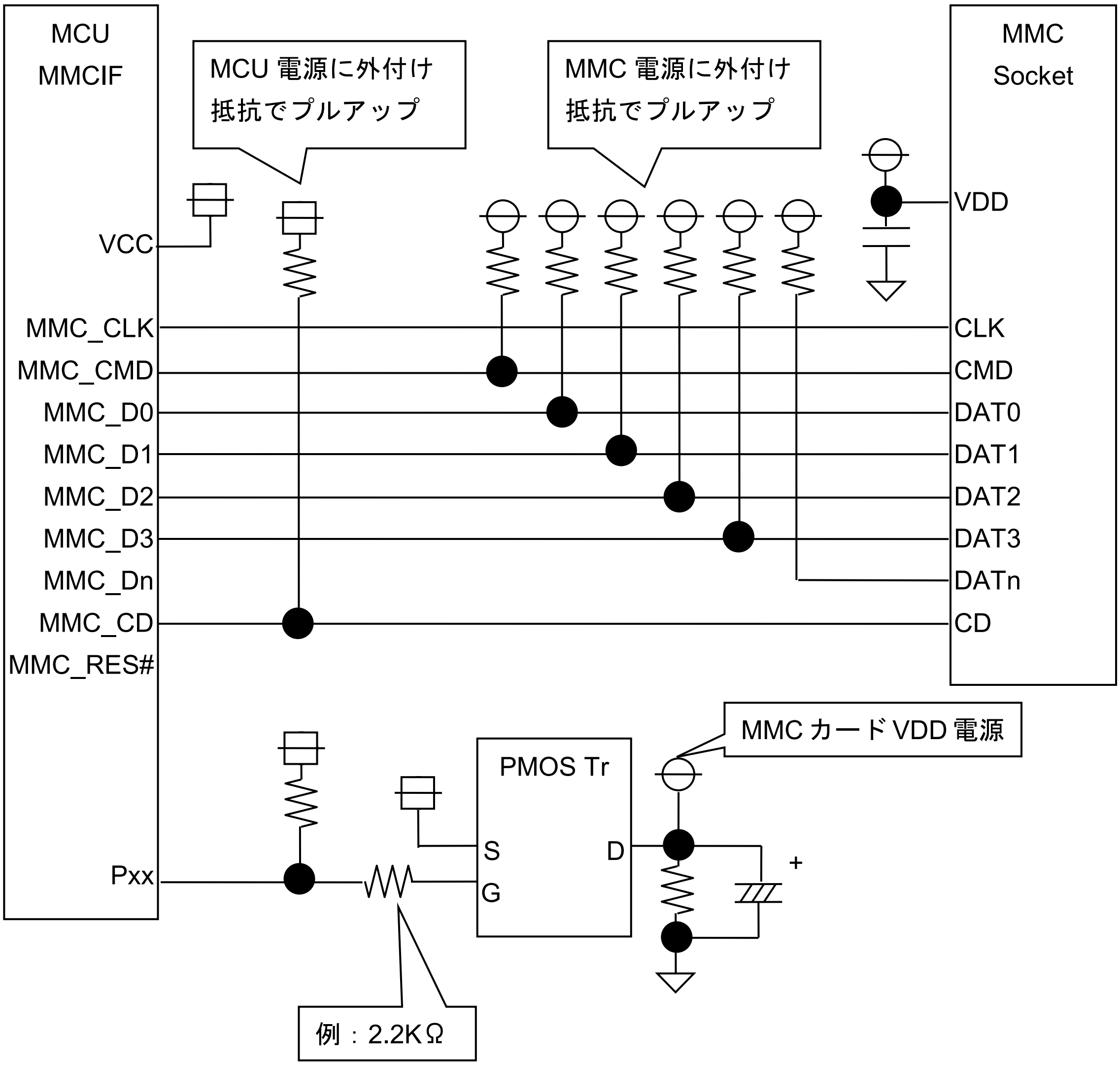
4.6.2.1 8bit バス接続例



DATn の n は 4-7 を示す。各 DAT4-7 にプルアップ抵抗を設けてください。

図 4-3　MCU と 8bit バス MMC（リムーバブルメディア：MMC カード）ソケットの接続例

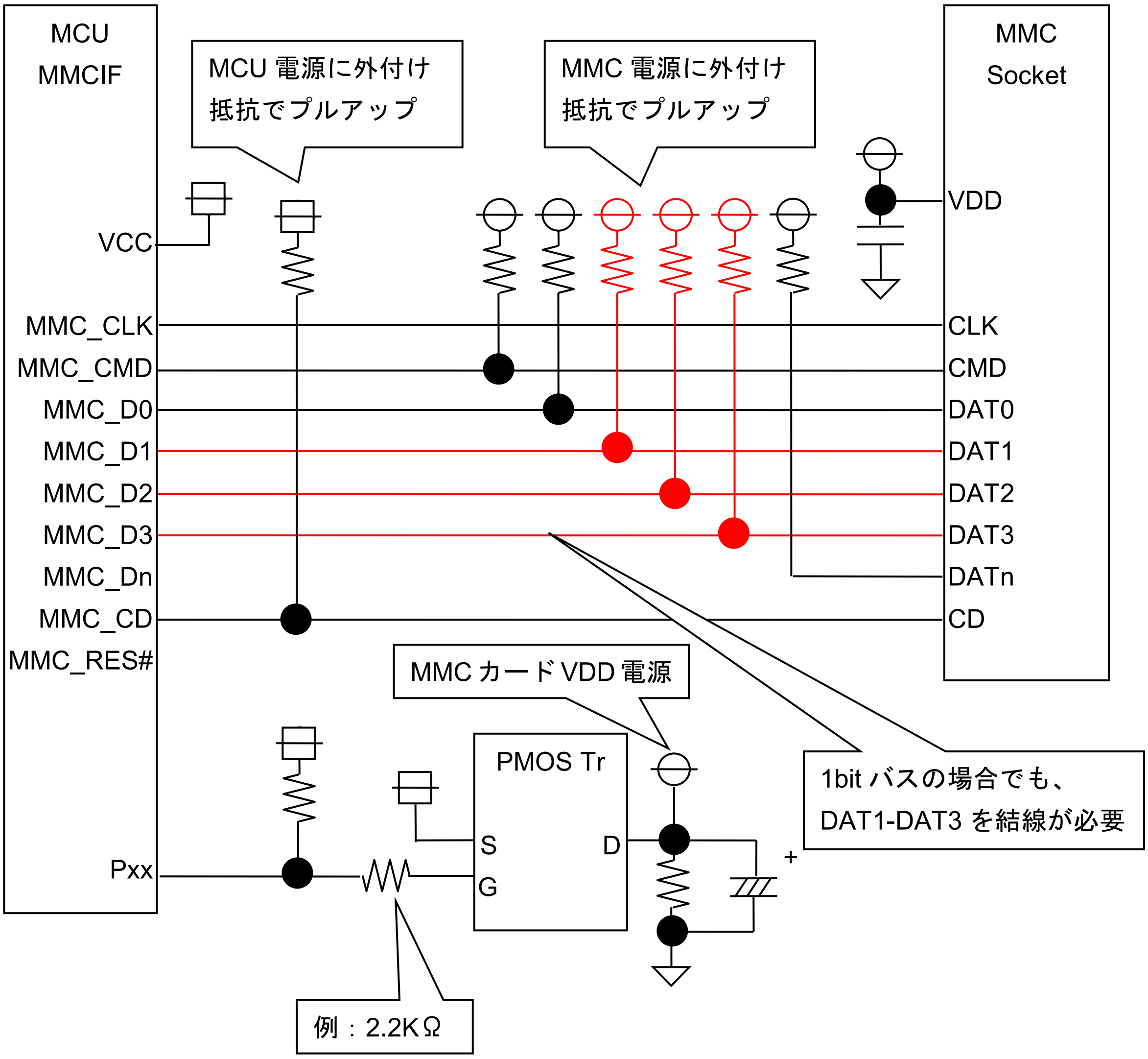
4.6.2.2　4bit バス接続例



DATn の n は 4-7 を示す。各 DAT4-7 にプルアップ抵抗を設けてください。

図 4-4　MCU と 4bit バス MMC（リムーバブルメディア：MMC カード）ソケットの接続例

4.6.2.3　1bit バス接続例



DATn の n は 4-7 を示す。各 DAT4-7 にプルアップ抵抗を設けてください。

図 4-5　MCU と 1bit バス MMC（リムーバブルメディア：MMC カード）ソケットの接続例

4.6.2.4 MCU リソース

MMCIF ドライバは、以下の MCU リソースを使用します。

表 4-5 使用端子と機能

使用するリソース	入出力	内容
MMC_CLK　注１	出力	MMC クロック出力（必須）
MMC_CMD　注１	入出力	MMC コマンド出力／レスポンス入力（必須）
MMC_D0　注１	入出力	MMC データ 0（必須）
MMC_D1　注１、注３	入出力	MMC データ 1（必須）
MMC_D2　注１、注３	入出力	MMC データ 2（必須）
MMC_D3　注１、注３	入出力	MMC データ 3（必須） バスサイズが 1 ビットの場合であっても、SPI モードへの遷移禁止制御目的のため、制御が必要です。
MMC_D4　注１、注３	入出力	MMC データ 4（オプション）
MMC_D5　注１、注３	入出力	MMC データ 5（オプション）
MMC_D6　注１、注３	入出力	MMC データ 6（オプション）
MMC_D7　注１、注３	入出力	MMC データ 7（オプション）
MMC_CD　注１、注４	入力	MMC カード検出入力（オプション）
MMC_RES#　注５	出力	MMC リセット（制御対象外）
MCU 電源	―	MCU の電源、MMC_CD 端子のプルアップ電源（必須）
MMC 電源	―	MMC の電源、MMC_CMD 端子／MMC_Dn 端子のプルアップ電源（必須）
Pxx（MMC 電源電圧制御ポート） （汎用入出力端子の割り当て） 注１、注２	出力	MMC 電源電圧制御出力（オプション） 電源制御に必要な数を割り当ててください。 回路構成により、H アクティブ／L アクティブ制御を行ってください。

注１：ユーザ側で端子を割り当ててください。

注２：ユーザ側で端子を割り当てて、端子制御してください。

注３：使用最大バスサイズにしたがって、以下の設定が可能です。

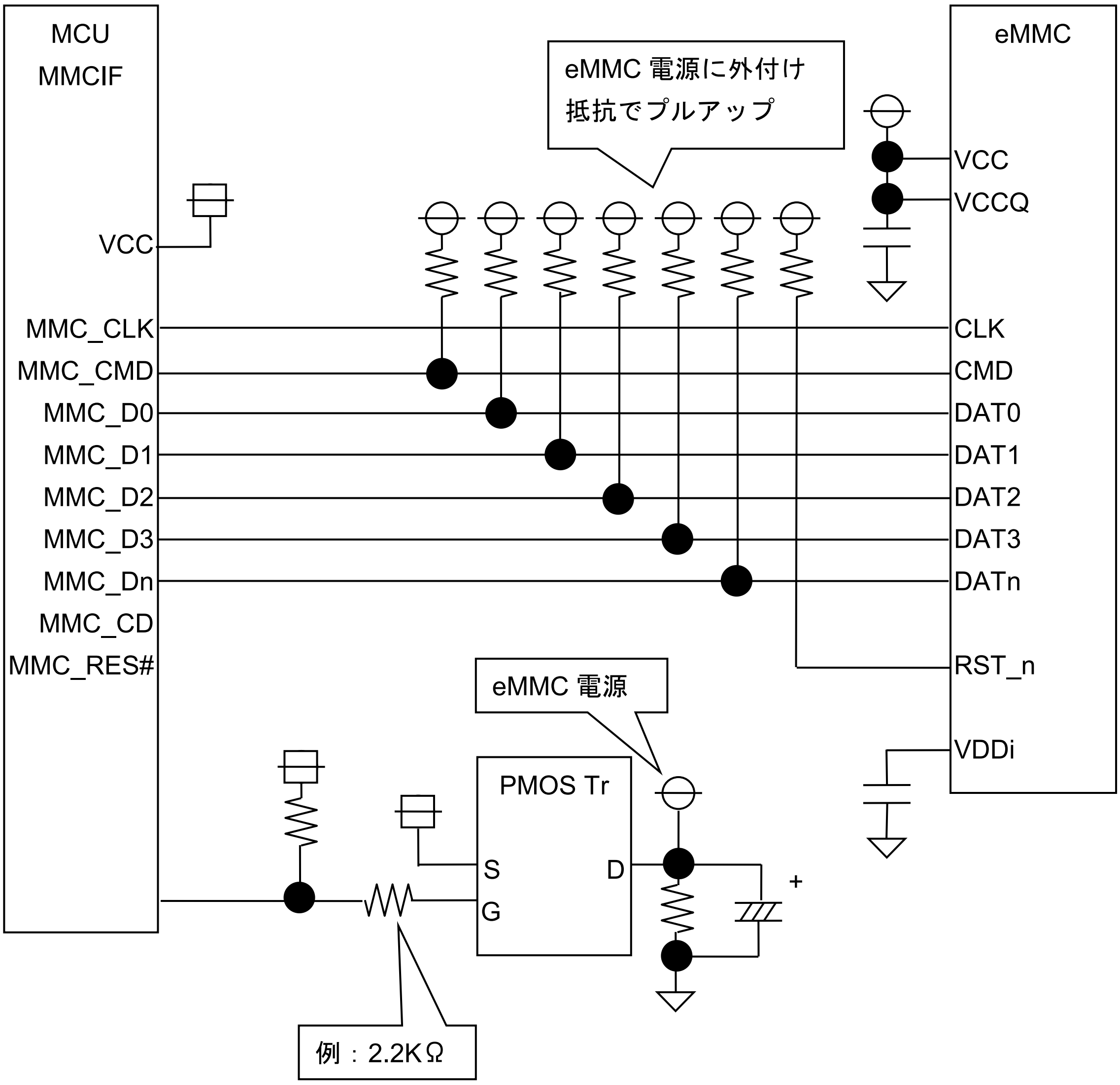
MMC バスで使用する最大バス幅	必須の MMC_Dn 端子割り当て
8	MMC_Dn（n は、0-7 を示す。）
4	MMC_Dn（n は、0-3 を示す。） MMC_D4-MMC_D7 は他用途に使用可。
1	MMC_Dn（n は、0-3 を示す。） MMC_D4-MMC_D7 は他用途に使用可。

注４：MMCIF ドライバ r_mmcif_rx_config.h で MMC カード検出機能 MMC_CFG_CHx_CD_ACTIVE を“1（有効）”にしてください。

注５：MMC カードにリセット端子はありません。MMC 以外の周辺機能として使用できます。

4.6.3　MMC（エンベディッド・マルチメディアカード：eMMC）の場合

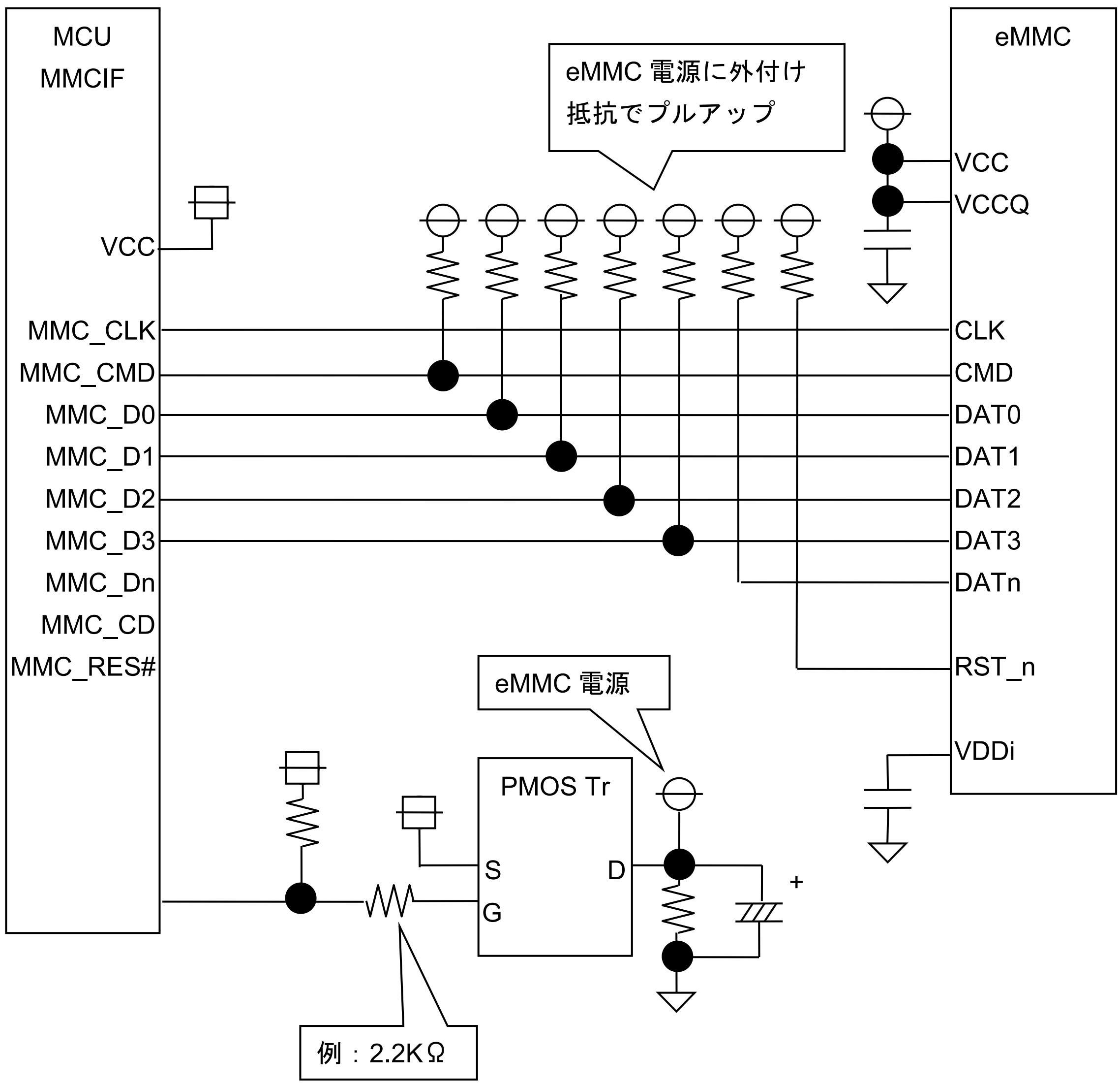
4.6.3.1　8bit バス接続例



DATn の n は 4-7 を示す。各 DAT4-7 にプルアップ抵抗を設けてください。
eMMC の VCC, VCCQ に共に 2.7-3.6V の同一電源を供給する場合を示す。

図 4-6　MCU と 8bit バス MMC（エンベディッド・マルチメディアカード：eMMC）の接続例

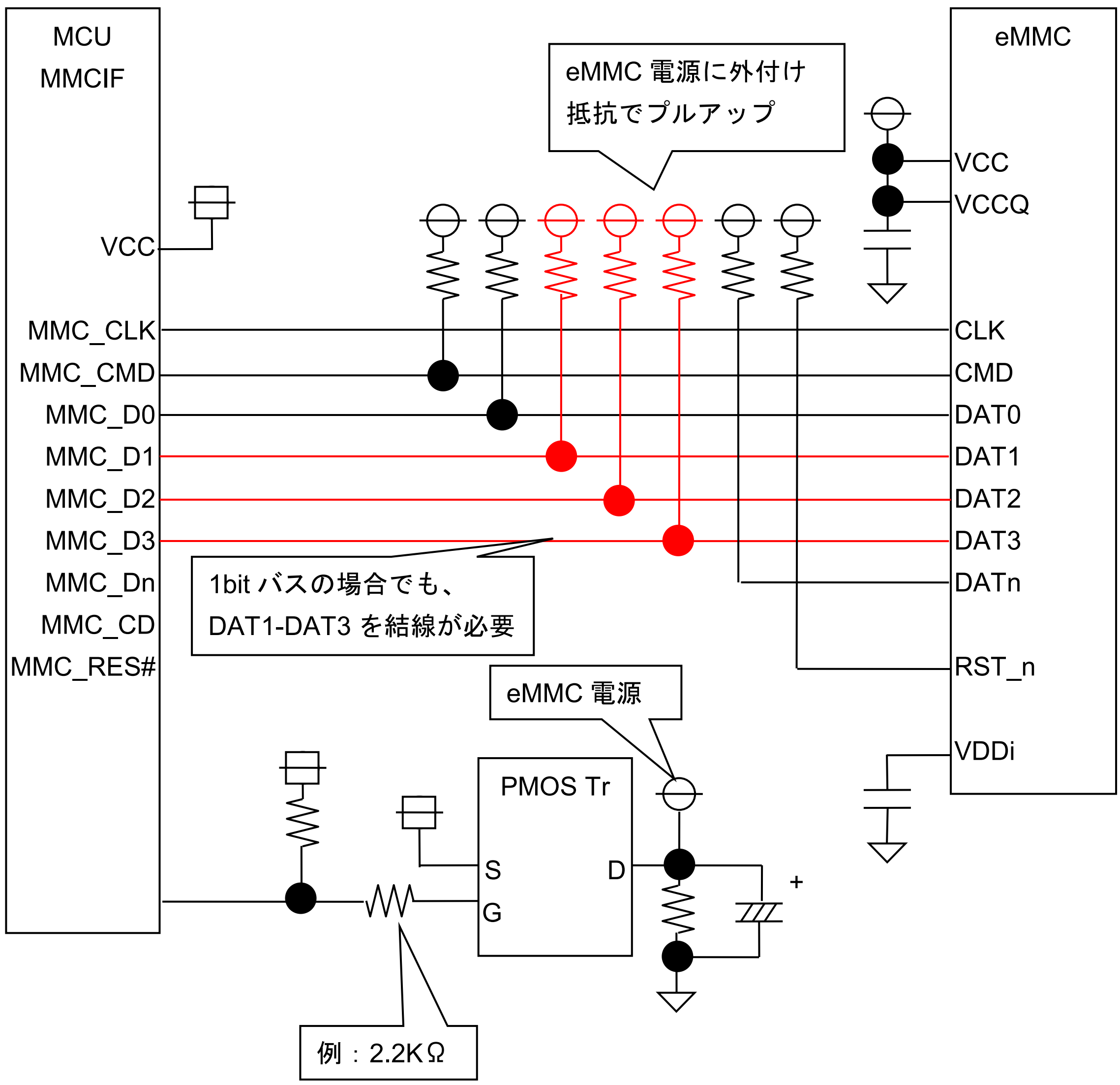
4.6.3.2　4bit バス接続例



DATn の n は 4-7 を示す。各 DAT4-7 にプルアップ抵抗を設けてください。
eMMC の VCC, VCCQ に共に 2.7-3.6V の同一電源を供給する場合を示す。

図 4-7　MCU と 4bit バス MMC（エンベディッド・マルチメディアカード：eMMC）の接続例

4.6.3.3　1bit バス接続例



DATn の n は 4-7 を示す。各 DAT4-7 にプルアップ抵抗を設けてください。
eMMC の VCC, VCCQ に共に 2.7-3.6V の同一電源を供給する場合を示す。

図 4-8　MCU と 1bit バス MMC（エンベディッド・マルチメディアカード：eMMC）の接続例

4.6.3.4 MCU リソース

MMCIF ドライバは、以下の MCU リソースを使用します。

表 4-6 使用端子と機能

使用するリソース	入出力	内容
MMC_CLK　注１	出力	MMC クロック出力（必須）
MMC_CMD　注１	入出力	MMC コマンド出力／レスポンス入力（必須）
MMC_D0　注１	入出力	MMC データ 0（必須）
MMC_D1　注１、注３	入出力	MMC データ 1（必須）
MMC_D2　注１、注３	入出力	MMC データ 2（必須）
MMC_D3　注１、注３	入出力	MMC データ 3（必須） バスサイズが 1 ビットの場合であっても、SPI モードへの遷移禁止制御目的のため、制御が必要です。
MMC_D4　注１、注３	入出力	MMC データ 4（オプション）
MMC_D5　注１、注３	入出力	MMC データ 5（オプション）
MMC_D6　注１、注３	入出力	MMC データ 6（オプション）
MMC_D7　注１、注３	入出力	MMC データ 7（オプション）
MMC_CD　注１、注４	入力	MMC カード検出入力（制御対象外）
MMC_RES#　注５	出力	MMC リセット（制御対象外）
MCU 電源	－	MCU の電源（必須）
eMMC 電源　注６	－	eMMC の電源、MMC_CMD 端子／MMC_Dn 端子のプルアップ電源（オプション）
Pxx（MMC 電源電圧制御ポート） （汎用入出力端子の割り当て） 注１、注２	出力	MMC 電源電圧制御出力（オプション） 電源制御に必要な数を割り当ててください。 回路構成により、H アクティブ／L アクティブ制御を行ってください。

注１：ユーザ側で端子を割り当ててください。

注２：ユーザ側で端子を割り当てて、端子制御してください。

注３：使用最大バスサイズにしたがって、以下の設定が可能です。

MMC バスで使用する最大バス幅	必須の MMC_Dn 端子割り当て
8	MMC_Dn（n は、0-7 を示す。）
4	MMC_Dn（n は、0-3 を示す。） MMC_D4-MMC_D7 は他用途に使用可。
1	MMC_Dn（n は、0-3 を示す。） MMC_D4-MMC_D7 は他用途に使用可。

注４：MMCIF ドライバ r_mmcif_rx_config.h で MMC カード検出機能 MMC_CFG_CHx_CD_ACTIVE を“0（無効）”にしてください。この設定により MMC カード検出端子の制御は無効になるため、MMC 以外の周辺機能として使用できます。

注５：MMCIF ドライバではリセット端子は制御しません。MMC 以外の周辺機能として使用できます。但し、Extended CSD の RST_n_FUNCTION [162]の Bit[1:0]は 0x0（default）でご使用ください。

注６：MCU 電源と eMMC 電源の制御電圧が異なる場合、eMMC 用の電源を設けてください。

5. サンプルプログラム

5.1 概要

FITDemos にサンプルプログラムを同梱しています。本サンプルプログラムでは、「4.4 MMC カードの挿入と電源投入タイミング」、「4.5 MMC カードの抜去と電源停止タイミング」、「1.6.5 MMC カード挿抜時のチャタリング制御」、MMC カードへの読み出し／書き込みの処理を行います。

なお、サンプルプログラムでは、読み出し／書き込みモードのデフォルト設定を Pre-defined（MMC_PRE_DEF）としています。Pre-defined の場合、MMC カードが動作しない可能性があります。MMC カードを使用する場合は Pre-defined から Open-ended（MMC_OPEN_END）に変更してください。変更箇所はサンプルプログラムの 115 行目から 118 行目です。

5.2 状態遷移図

図 5-1 に状態遷移図を示します。

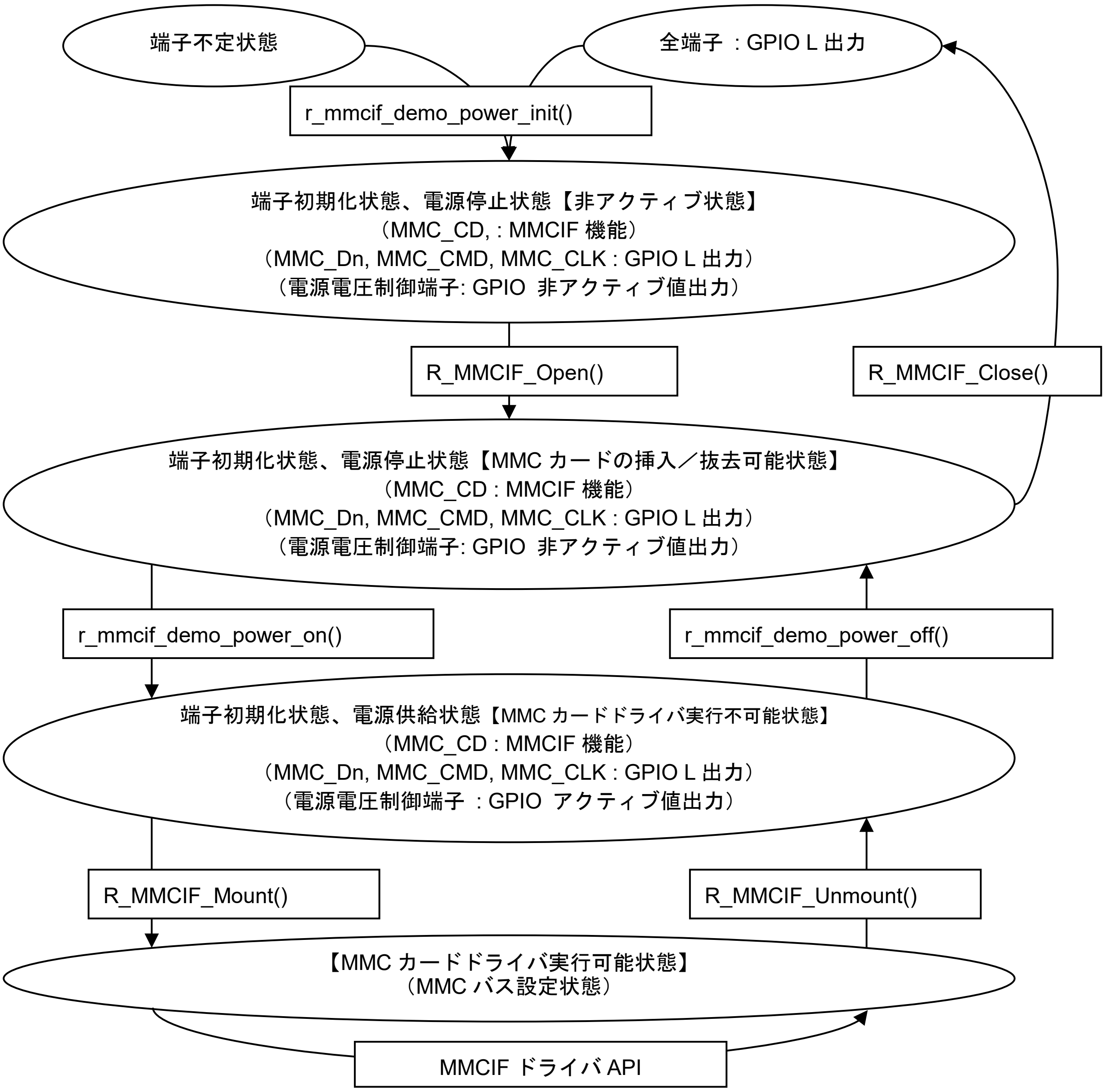


図 5-1 状態遷移図

5.3 コンパイル時の設定

サンプルプログラムのコンフィギュレーションオプションの設定は、r_mmcif_rx_demo_pin_config.h で行います。

RX64M RSK または RX65N RSK を使用する場合の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。

Configuration options in *r_mmcif_rx_demo_pin_config.h*	
#define MMC_CFG_MODE_SW (1) #define MMC_CFG_MODE_DMAC (0) #define MMC_CFG_MODE_DTC (0) ※デフォルト値は"Software 転送"を選択。	サンプルプログラムで使用する転送モードを設定します。 転送モードを 1 つ有効にしてください。（1：有効、0：無効） DMAC 転送または DTC 転送を行う場合、別途、DMAC FIT モジュールもしくは DTC FIT モジュールが必要です。
#define MMC_CFG_POWER_PORT_NONE ※デフォルト値は“無効”	MMC カードを使用する場合の定義です。 MMC カード電源制御が不要な場合、定義を有効にしてください。 MMC カード電源制御が必要な場合、定義を無効にしてください。
#define MMC_CFG_POWER_HIGH_ACTIVE (1) ※デフォルト値は“1（High を供給）”	MMC カードを使用し、かつ、カード電源制御が必要な場合に設定する定義です。 “1”の場合、カード電源回路を有効にするために、カード電源回路を制御しているポートに High を供給します。 “0”の場合、カード電源回路を有効にするために、カード電源回路を制御しているポートに Low を供給します。
#define MMC_CFG_CHAT_CNT (300) ※デフォルト値は“300（300ms ウェイト）”	MMC カードを使用する場合の定義です。 カード挿抜時のチャタリングカウンタです。1 カウントあたり、1ms のウェイトを行います。 システムに合わせて設定してください。
#define MMC_CFG_POWER_ON_WAIT (100) ※デフォルト値は“100（100ms ウェイト）”	MMC カードを使用する場合の定義です。 MMC カード用電源回路に電源供給開始後、動作電圧に達するまでのウェイト時間を設定してください。1 カウントあたり、1ms のウェイトを行います。 システムに合わせて設定してください。
#define MMC_CFG_POWER_OFF_WAIT (100) ※デフォルト値は“100（100ms ウェイト）”	MMC カードを使用する場合の定義です。 MMC カード用電源回路に電源供給停止後、MMC の抜去可能電圧に達するまでのウェイト時間を設定してください。1 カウントあたり、1ms のウェイトを行います。 システムに合わせて設定してください。
#define MMC_CFG_POWER_CHx_PORT ※CHx の“x”はチャネル番号（x=0 or 1）	チャネル x 用の電源電圧制御端子に割り付けるポート番号を設定してください。 設定値の前後にシングルコーテーション「‘」「’」をつけてください。
#define MMC_CFG_POWER_CHx_BIT ※CHx の“x”はチャネル番号（x=0 or 1）	チャネル x 用の電源電圧制御端子に割り付けるビット番号を設定してください。 設定値の前後にシングルコーテーション「‘」「’」をつけてください。

5.4 API 関数

サンプルプログラム内 API 関数を以下に示します。必要に応じて、関数の追加／修正してください。

表 5-1　API 関数一覧

関数名	機能概要
r_mmcif_demo_power_init()	電源電圧制御端子設定の初期化処理
r_mmcif_demo_power_on()	電源電圧の供給開始処理
r_mmcif_demo_power_off()	電源電圧の供給停止処理
r_mmcif_demo_softwaredelay()	時間待ち処理

5.4.1 r_mmcif_demo_power_init()

MMCIF ドライバで使用する MMCIF 端子設定を初期化する関数です。また、eMMC または MMC カードの電源電圧制御端子の設定を初期化する関数です。

Format

```
mmc_status_t r_mmcif_demo_power_init(
      uint32_t channel
)
```

Parameters

channel

チャネル番号　　使用する MMCIF チャネル番号（0 起算）

Return Values

MMC_SUCCESS　　*正常終了*

Description

r_mmcif_rx_demo_pin_config.h で指定した MMCIF ドライバが使用する MMC_CD, MMC_Dn, MMC_CMD, MMC_CLK 端子の設定を初期化します。また、eMMC または MMC カードの電源電圧制御端子の設定を初期化します。

Special Notes

電源電圧制御端子について、以下のとおり設定します。

・ポートモードレジスタ（PMR）を汎用入出力ポートに設定します。

・プルアップ制御レジスタ（PCR）を入力プルアップ抵抗無効に設定します。

・端子出力を非アクティブ状態に設定します。

5.4.2 r_mmcif_demo_power_on()

eMMC または MMC カードの電源電圧制御端子を制御し、電源供給を開始する関数です。

Format

```
mmc_status_t r_mmcif_demo_power_on(
    uint32_t channel
)
```

Parameters

channel

チャネル番号	使用する MMCIF チャネル番号（0 起算）

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Description

eMMC または MMC カードの電源電圧制御端子を制御し、電源供給を開始します。その後、r_mmcif_rx_demo_pin_config.h の MMC_CFG_POWER_ON_WAIT で設定された時間経過後に結果を返します。

Special Notes

必要に応じて修正してください。

電源電圧供給開始後、動作電圧に達するまでの時間待ちのため、r_mmcif_demo_softwaredelay()関数を実行します。待ち時間は「5.3 コンパイル時の設定」の「MMC_CFG_POWER_ON_WAIT」で設定してください。

本関数実行前に r_mmcif_demo_power_init()関数による初期化処理が必要です。

5.4.3 r_mmcif_demo_power_off()

eMMC または MMC カードの電源電圧制御端子を制御し、電源供給を停止する関数です。

Format

```
mmc_status_t r_mmcif_demo_power_off(
      uint32_t channel
)
```

Parameters

channel

チャネル番号	使用する MMCIF チャネル番号（0 起算）

Return Values

MMC_SUCCESS	*正常終了*
MMC_ERR	*一般エラー*

Description

eMMC または MMC カードの電源電圧制御端子を制御し、電源供給を停止します。その後、r_mmcif_rx_demo_pin_config.h の MMC_CFG_POWER_OFF_WAIT で設定された時間経過後に結果を返します。

Special Notes

電源電圧供給停止後、抜去可能電圧に達する動作電圧に達するまでの時間待ちのため、r_mmcif_demo_softwaredelay()関数を実行します。待ち時間は「5.3 コンパイル時の設定」の「MMC_CFG_POWER_OFF_WAIT」で設定してください。

本関数実行前に r_mmcif_demo_power_init()関数による初期化処理が必要です。

5.4.4　r_mmcif_demo_softwaredelay()

時間待ちを行う際に使用する関数です。

Format

```
bool r_mmcif_demo_softwaredelay(
    uint32_t delay,
    mmc_delay_units_t units
)
```

Parameters

delay

タイムアウト時間（単位：units で設定）

units

マイクロ秒：MMC_DELAY_MICROSECS

ミリ秒：MMC_DELAY_MILLISECS

秒：MMC_DELAY_SECS

Return Values

true	*正常終了*
false	*パラメータエラー*

Description

時間待ち処理を行います。

タイムアウト時間 delay になると、true を返します。

Special Notes

表 5-2 に時間待ち処理を示します。本関数は、設定時間を待つ機能のみのため、OS の自タスク遅延処理（例：μ自タスク遅の dly_tsk()）等に置き換えることが可能です。

表 5-2　時間待ち処理

分類	内容　＜＞中の値は提供時の設定値を示す
MMC カード電源 On 時の電圧安定待ち時間	MMC カード用電源回路に電源供給開始後、動作電圧に達するまでの待ち時間＜100ms＞ ※待ち時間は MMC_CFG_POWER_ON_WAIT で変更可能。
MMC カード電源 Off 時の電圧安定待ち時	MMC カード用電源回路に電源供給停止後、MMC カードの抜去可能電圧に達するまでの待ち時間＜100ms＞ ※待ち時間は MMC_CFG_POWER_OFF_WAIT で変更可能。

5.5　待ち処理の OS 処理への置き換え方法

サンプルプログラムで発生する時間待ち処理 r_mmcif_demo_softwaredelay()を OS の自タスク遅延処理（例：μITRON の dly_tsk()）に置き換えることができます。

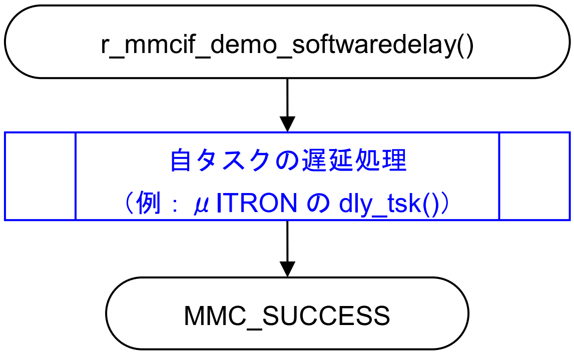


図 5-2　OS の自タスク遅延処理を使った時間待ち例

5.6 mmcif_demo_rskrx64m，mmcif_demo_rskrx65n，mmcif_demo_rskrx64m_gcc, mmcif_demo_rskrx65n_gcc

コードをコンパイルし、ターゲットボードにダウンロードし、実行すると、初期化後に LED0 が点灯します。MMCIF モジュールが正しくオープンされると、LED1 が点灯します。MMC にデータが正しく書き込まれると、LED2 が点灯します。MMC からデータが正しく読みだされると、LED3 が点灯します。MMCIF モジュールが正しくクローズされると、全ての LED が消灯します。

セットアップと実行

1. r_mmcif_rx_config.h でチャネル 0 のドライバサポートを有効にします。

```
#define MMC_CFG_CH0_INCLUDED
```

2. データ転送モジュールの選択

DMAC 転送モードを使用する場合、r_mmcif_rx_pin_config.h 内の MMC_CFG_MODE_DMAC を 1 に設定してください。

```
#define MMC_CFG_MODE_DMAC (1)
```

DTC 転送モードを使用する場合、r_mmcif_rx_pin_config.h 内の MMC_CFG_MODE_DTC を 1 に設定してください。

```
#define MMC_CFG_MODE_DTC (1)
```

デフォルトでは、転送モードはソフトウェア転送になっています。

3. RSK ボードを PC に接続します（Renesas E1 エミュレータを使用）。外部電源として DC 5V 3A の電源アダプタを RSK ボードの電源ジャック(PWR)に接続する必要があります。本サンプルアプリケーションをビルドし、ボードにダウンロードします。

4. ルネサス e2 studio IDE の Renesas Views tab をクリック -> デバッグの項目内にある Renesas Debug Virtual Console を選択してください。

5. ログと LED をチェックすることで、MMC への 3 セクタ(512 バイト/セクタ)の書き込みと読み出しを確認してください。

サポートされるボード

RSKRX64M, RSKRX65N

5.7 ワークスペースにデモを追加する

デモプロジェクトは、本アプリケーションノートで提供されるファイルの FITDemos サブディレクトリにあります。ワークスペースにデモプロジェクトを追加するには、「ファイル」 >> 「インポート」を選択し、「インポート」ダイアログから「一般」の「既存プロジェクトをワークスペースへ」を選択して「次へ」ボタンをクリックします。「インポート」ダイアログで「アーカイブ・ファイルの選択」ラジオボタンを選択し、「参照」ボタンをクリックして FITDemos サブディレクトリを開き、使用するデモの zip ファイルを選択して「終了」をクリックします。

5.8 デモのダウンロード方法

デモプロジェクトは、RX Driver Package には同梱されていません。デモプロジェクトを使用する場合は、個別に各 FIT モジュールをダウンロードする必要があります。「スマートブラウザ」の「アプリケーションノート」タブから、本アプリケーションノートを右クリックして「サンプル・コード（ダウンロード）」を選択することにより、ダウンロードできます。

6. 付録

6.1 動作確認環境

本ドライバの動作確認環境を以下に示します。

表 6-1　動作確認環境 (Ver.1.03)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio V6.2.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.08.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.03
使用ボード	Renesas Starter Kit for RX64M (型名：R0K50564MSxxxBE) Renesas Starter Kit for RX71M (型名：R0K50571MSxxxBE) Renesas Starter Kit for RX65N (型名：RTK500565NSxxxxxBE) Renesas Starter Kit for RX65N-2MB (型名：RTK50565N2SxxxxxBE)

表 6-2　動作確認環境 (Ver.1.04)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio V7.3.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.01.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.04

表 6-3　動作確認環境 (Ver.1.05)

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e^2 studio V7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C コンパイラ	ルネサス エレクトロニクス製 C/C++ compiler for RX family V.3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 4.08.04.201803 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.01 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.05
使用ボード	Renesas Starter Kit+ for RX64M (型名：R0K50564Mxxxxxx)

表 6-4　動作確認環境 (Ver.1.06)

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e2 studio V7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C コンパイラ	ルネサス エレクトロニクス製 C/C++ compiler for RX family V.3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 4.08.04.201902 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.06
使用ボード	Renesas Starter Kit+ for RX72M (型名：RTK5572Mxxxxxxxxxx)

表 6-5　動作確認環境 (Ver.1.07)

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e2 studio V7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C コンパイラ	ルネサス エレクトロニクス製 C/C++ compiler for RX family V.3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 4.08.04.201902 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.07
使用ボード	Renesas Starter Kit+ for RX72N (型名：RTK5572Nxxxxxxxxxx)

表 6-6　動作確認環境 (Ver.1.10)

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e2 studio 2022-10 IAR Embedded Workbench for Renesas RX 4.20.3
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.04.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.202202 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99 リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、統合開発環境のデフォルト設定に以下のオプションを追加 -WI,--no-gc-sections これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。
	IAR C/C++ Compiler for Renesas RX version 4.20.3 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.10
使用ボード	Renesas Starter Kit+ for RX64M (型名：R0K50564Mxxxxxx) Renesas Starter Kit+ for RX65N (型名：RTK5005651Cxxxxxxx)

表 6-7　動作確認環境 (Ver.1.20)

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e2 studio 2024-10 IAR Embedded Workbench for Renesas RX 5.10.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.06.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.202411 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99 リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、統合開発環境のデフォルト設定に以下のオプションを追加 -WI,--no-gc-sections これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。
	IAR C/C++ Compiler for Renesas RX version 5.10.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.20
使用ボード	Renesas Starter Kit for RX65N-2MB (型名：RTK50565N2SxxxxxBE)

表 6-8　動作確認環境 (Ver.1.21)

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e² studio 2025-10 IAR Embedded Workbench for Renesas RX 5.10.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.07.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.202411 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99 リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、統合開発環境のデフォルト設定に以下のオプションを追加 -Wl,--no-gc-sections これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。
	IAR C/C++ Compiler for Renesas RX version 5.10.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.21
使用ボード	-

6.2 トラブルシューティング

(1) Q : 本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A : FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e² studio を使用している場合
  アプリケーションノート RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q : 本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「This MCU is not supported by the current R_MMCIF_rx module.」エラーが発生します。

A : 追加した FIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。

6.3 OS 処理への置き換え方法

本ドライバで発生するステータス割り込み処理と時間待ち処理を OS 処理に置き換えることができます。以下に、関数一覧と詳細を示します。

表 6-9 ターゲット MCU インタフェース関数一覧

関数名	機能概要
r_mmcif_dev_int_wait()	ステータス割り込み待ち処理
r_mmcif_dev_wait()	時間待ち処理

(1) r_mmcif_dev_int_wait() 注 1

ステータス割り込みを待つ際に使用する関数です。

Format

```
mmc_status_t r_mmcif_dev_int_wait(
      uint32_t channel,
      int32_t time
)
```

Parameters

channel
　チャネル番号　　使用する MMCIF チャネル番号（0 起算）

time
　タイムアウト時間（単位：ミリ秒）

Return Values

MMC_SUCCESS　　*正常終了（割り込み要求発生）*
MMC_ERR　　*一般エラー*

Description

MMC とのプロトコル通信時の割り込み待ち処理を行います。

割り込み要求を確認できた場合は、MMC_SUCCESS を返します。

タイムアウト時間 time 時間内に割り込み要求を検出できなかった場合は、MMC_ERR を返します。

割り込み待ち処理は、割り込みを使用した処理を実装済です。

本関数内で割り込みステータスフラグレジスタ取得処理（r_mmcif_get_intstatus()関数）をコールして割り込み要求が発生しているかを確認します。

Special Notes:
MMC との通信時のレスポンス受信待ち時間やデータ転送完了待ち時間を他の処理に割り当てることができます。
以下の図 6-1 は、OS の自タスク遅延処理（例：μ自タスク遅の dly_tsk()）を使用した場合の使用例です。
但し、OS 処理は r_mmcif_dev_int_wait()関数にユーザ独自で組み込んでください。

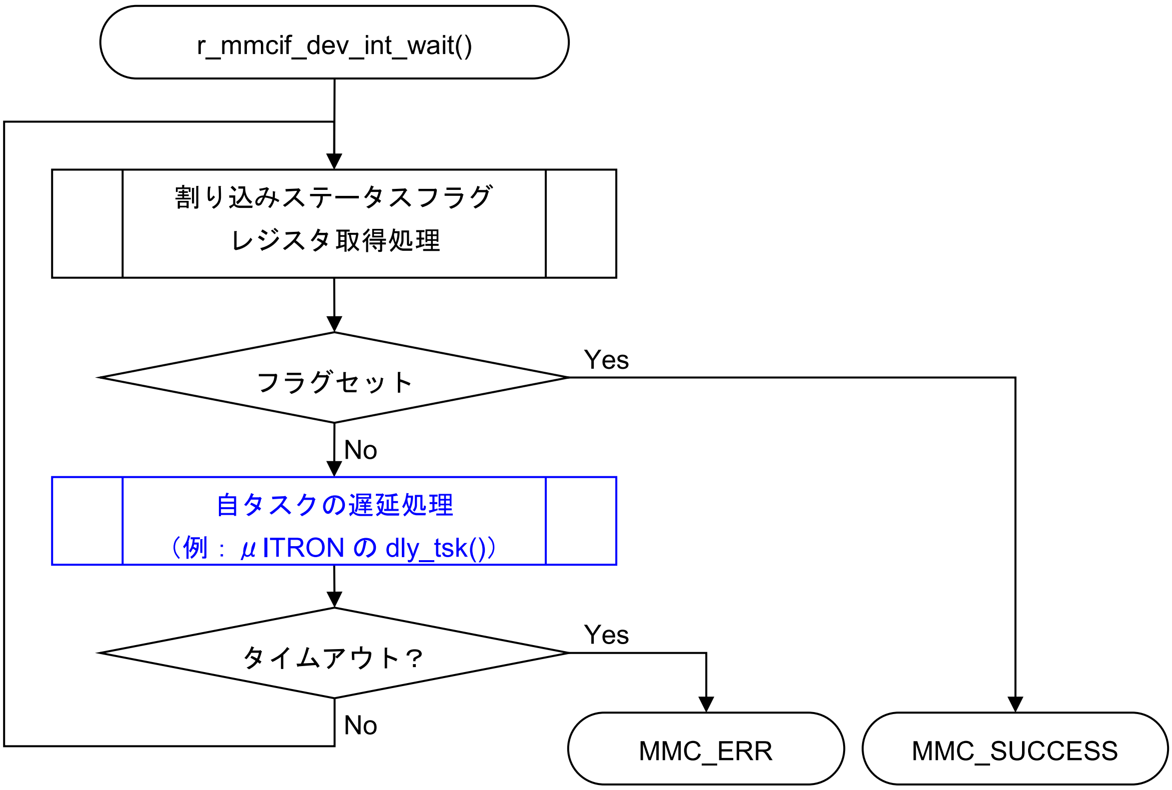


図 6-1　OS の自タスク遅延処理を使った MMC プロトコルステータス確認例

以下の図 6-2 は、OS のイベントフラグセット待ち処理を使用した場合の使用例です。使用する場合、r_mmcif_dev_int_wait()関数の割り込みステータスフラグレジスタ取得処理（r_mmcif_get_intstatus()関数）をイベントフラグセット待ち処理に置き換え、かつ、MMC プロトコルステータス割り込みコールバック関数に起床処理を追加してください。

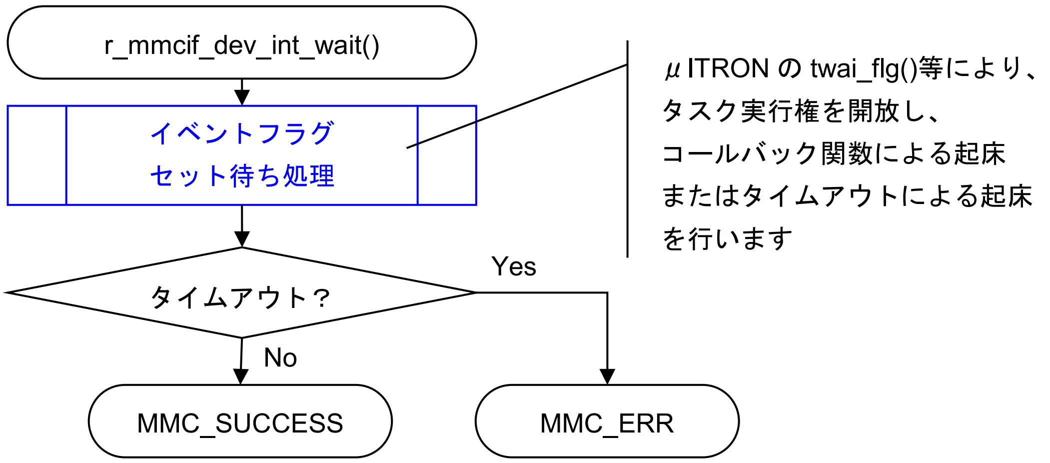


図 6-2　OS のウェイトタスク処理を使った MMC プロトコルステータス確認例

(2)　r_mmcif_dev_wait()

時間待ちを行う際に使用する関数です。

Format

mmc_status_t r_mmcif_dev_wait(
　　uint32_t channel,
　　int32_t time
)

Parameters

channel
　チャネル番号　　使用する MMCIF チャネル番号（0 起算）

time
　タイムアウト時間（単位：ミリ秒）

Return Values

MMC_SUCCESS　　*正常終了（割り込み要求発生）*
MMC_ERR　　*一般エラー*

Description

時間待ち処理を行います。
タイムアウト時間 time になると MMC_SUCCESS を返します。

Special Notes:

表 6-7 にステータス確認を伴わない時間待ち処理を示します。本関数は設定時間を待つ機能のみのため、OS の自タスク遅延処理（例：μ自タスク遅の dly_tsk()）等に置き換えることが可能です。

表 6-10　ステータス確認を伴わない時間待ち処理

分類	内容　< >中の値は提供時の設定値を示す
MMC の初期化処理時の 74 クロック発生	Card identification mode：MMC 初期化のための 74 クロック発生時間待ち＜3ms＞（最大 3ms。最小 2ms を確保）
MMC の初期化処理時の Ready 状態遷移検出	Card identification mode：MMC の Ready 状態への遷移待ち＜5ms＞（最大 1 秒） MMC の場合、5ms 間隔で CMD1 を発行し、最大 200 回繰り返す。
マウント／リード／ライト処理の異常時の Ready 状態遷移検出	Data transfer mode：マウント／リード／ライト処理の CMD13 発行後の MMC の Ready 状態への遷移待ち＜1ms＞（タイムアウトまで、1ms の時間待ちを複数回繰り返す）

< >中の値は提供時の設定値です。

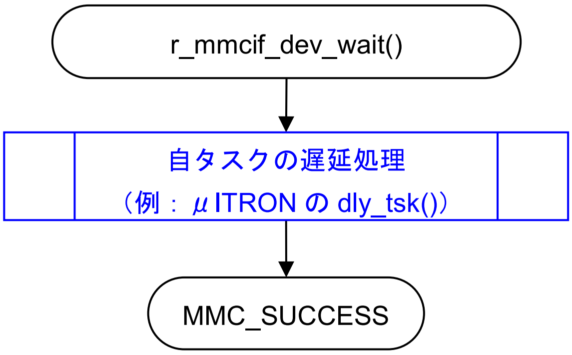


図 6-3　OS の自タスク遅延処理を使った時間待ち例

7. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境
RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデートの対応について

該当するテクニカルアップデートはありません。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.03	2018.03.31	-	初版発行 MMC モード MMCIF ドライバ・ソフトウェア RTM0RX0000DMMC0 Ver.1.02 ユーザーズマニュアル（R01UW0118）を本アプリケーションノートに変更した。
1.04	2019.02.01	6	有償 FAT に関する記述を削除
		87	表 6 2　動作確認環境 (Ver.1.04)を追加
		-	機能関連 Smart Configurator での GUI によるコンフィグオプション設定機能に対応 ■内容 GUI によるコンフィグオプション設定機能に対応するため、設定ファイルを追加。
1.05	2019.05.20	—	以下のコンパイラに対応 ・GCC for Renesas RX ・IAR C/C++ Compiler for Renesas RX
		—	表 1.3 を削除
		—	表 6.3 と表 6.4 を削除
		1	「対象コンパイラ」を追加
		1	「関連ドキュメント」　R01AN1723、R01AN1826、R20AN0451 を削除
		21	「2.2 ソフトウェアの要求」　依存する r_bsp モジュールのリビジョンを追加
		24	「2.8 コードサイズ」を更新
		89	表 6.3 動作確認環境 (Ver.1.05)を追加
1.06	2019.07.30	-	RX72M に関連した変更
		24	「2.8 コードサイズ」の内容を更新
		30	「2.13 for 文、while 文、do while 文について」を追加
		31-67	API 説明ページの「Reentrant」項目を削除
		90	表 6.4 動作確認環境 (Ver.1.06)を追加
1.07	2019.11.22	-	RX66N、RX72N に関連した変更
		9	「1.4.1 クイックスタートガイド」に、RX72N のハードウェア設定を追加
		21	「2.4 使用する割り込みベクタ」に、RX66N と RX72N デバイスを追加
		24	「2.8 コードサイズ」を更新
		65-66	「3.22 R_MMCIF_Set_LogHdlAddress()」と「3.23 R_MMCIF_Log()」の「Special Notes」を修正
		90	表 6.5 動作確認環境 (Ver.1.07)を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
1.10	2022.12.27	83	デモプロジェクトの更新と追加 「5. デモプロジェクト」に RSKRX65N を追加。
		89	「5.6. mmcif_demo_rskrx64m, mmcif_demo_rskrx65n, mmcif_demo_rskrx64m_gcc, mmcif_demo_rskrx65n_gcc」を追加
		89	「5.7. ワークスペースにデモを追加する」を追加。
		92	「6.1 動作確認環境」： Rev.1.10 に対応する表を追加。
		—	デモプロジェクトの更新と追加 Linux 対応のため、インクルードヘッダファイルパスの形式を更新しました。
1.20	2024.12.31	29	「2.12 FIT モジュールの追加方法」から FIT configurator の記述を削除。
		93	「6.1 動作確認環境」： Rev.1.20 に対応する表を追加。
		—	API 関数のコメントを Doxygen スタイルに変更 FIT Configurator の説明を削除しました。 ファイルの説明を更新。
1.21	2025.03.15	94	「6.1 動作確認環境」： Rev.1.21 に対応する表を追加。
		—	FIT モジュールの免責事項と著作権を更新。

製品ご使用上の注意事項

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

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

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

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

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

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

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

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

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

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

www.renesas.com/contact/

商標について

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