RX ファミリ

SD モード SD メモリカードドライバ 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 SD モード SD メモリカード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 内臓 SD ホストインタフェース（SDHI）を使用して、SD メモリカードを SD モードで制御します。以降、本ドライバを SD メモリカードドライバと称します。

SD 規格に対応したホスト機器を開発するには、SD Host/Ancillary Product License Agreement(SD HALA)の締結が必要です。

詳細は SD Association のサイトをご確認ください。
https://www.sdcard.org/

対象デバイス

・RX ファミリ

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)
- RX ファミリ SDHI モジュール Firmware Integration Technology (R01AN3852)

目次

1. 概要

1.1 SD メモリカードドライバとは

本ドライバは別途無償提供している下位層の SDHI FIT モジュールと組み合わせて使用することにより、SD メモリカードの制御が可能になります。また、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、SD メモリカードに対してファイルアクセスが可能になります。

なお、SD メモリと SDIO の共通する SDHI 制御ソフトウェアの場合、SD カードドライバと略します。

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 SD メモリカードドライバの概要

表 1-1、表 1-2 に本ドライバの機能を示します。

表 1-1　SDHI 機能一覧

項目	機能
準拠した規格	SD Specifications Part 1 Physical Layer Simplified Specification Ver.6.00 準拠
SDHI 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
動作対象 SD カード	SD メモリカード
SD カード動作電圧	2.7-3.6 V（High Voltage）のみ、かつ 3.3V 信号レベルをサポート
SD カード Bus インタフェース	SD モード（4 ビット）をサポート
SD カード制御可能数	1 デバイス／チャネル
SD カード Speed mode	Default Speed mode をサポート
SD カードメモリ容量	標準容量 SD メモリカード（SDSC）と大容量 SD メモリカード（SDHC、SDXC）をサポート
SD カードメモリ制御対象	ユーザ領域のみをサポート プロテクト領域の制御はサポート対象外
SD カード検出機能	CD 端子による検出のみをサポート

表 1-2　MCU 機能一覧

項目	機能
対象 MCU	SDHI 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

SD メモリカード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。

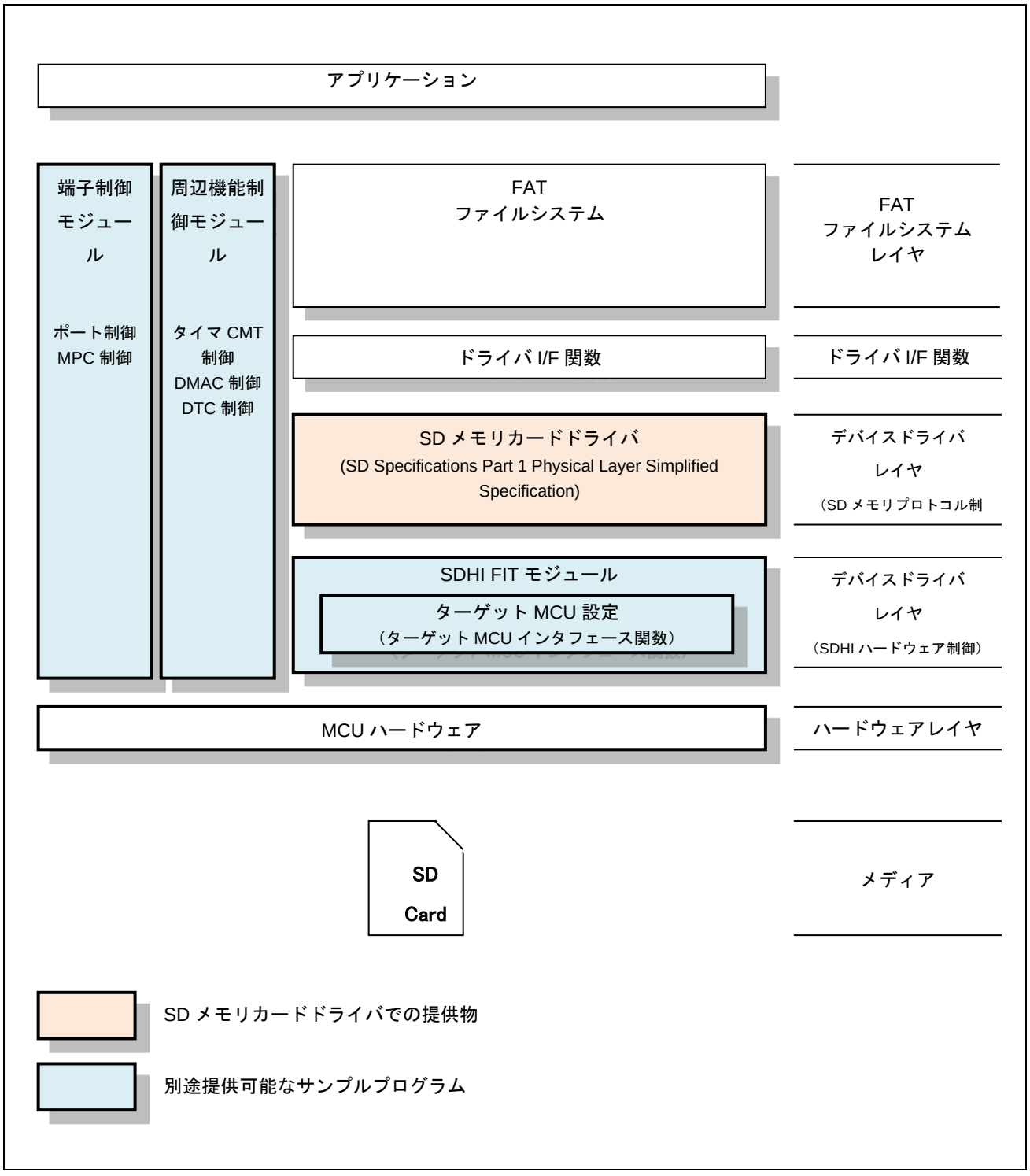


図 1-1　アプリケーション構成図

(1)　FAT ファイルシステム

SD メモリ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。

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-rx

(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 SD メモリカードドライバ 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

(3)　SD メモリカードドライバ

SD Specifications Part 1 Physical Layer Simplified Specification の SD メモリプロトコル制御を行うソフトウェアです。

(4)　SDHI FIT モジュール

SDHI ハードウェア制御を行うソフトウェアです。また、MCU に依存するターゲット MCU インタフェース関数および割り込み設定ファイルが含まれます。

(5)　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。

(6)　端子制御モジュール（サンプルプログラム）

SDHI 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（SDHI 機能制御と SD カード電源用ポート制御）、MPC 制御（SDHI 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

SD メモリカードドライバは、SD Specifications Part 1 Physical Layer Simplified Specification のプロトコルを使ったデバイスドライバです。

表 1-4 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_SDC_SD_Open()	ドライバのオープン処理
R_SDC_SD_Close()	ドライバのクローズ処理
R_SDC_SD_GetCardDetection()	挿入確認処理
R_SDC_SD_Initialize()	初期化処理
R_SDC_SD_End()	終了処理
R_SDC_SDMEM_Read()	リード処理　注 1
R_SDC_SDMEM_ReadSoftwareTrans()	リード処理（Software 転送）
R_SDC_SDMEM_ReadSoftwareTransSingleCmd()	リード処理（CMD17 シングル Software 転送）
R_SDC_SDMEM_Write()	ライト処理　注 1
R_SDC_SDMEM_WriteSoftwareTrans()	ライト処理（Software 転送）
R_SDC_SD_Control()	ドライバのコントロール処理 SDC_SD_SET_STOP コマンド
R_SDC_SD_GetModeStatus()	モードステータス情報取得処理
R_SDC_SD_GetCardStatus()	カードステータス情報取得処理
R_SDC_SD_GetCardInfo()	レジスタ情報取得処理
R_SDC_SDMEM_GetSpeed()	スピードクラス情報取得処理
R_SDC_SD_CdInt()	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_SDC_SD_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_SDC_SD_GetErrCode()	ドライバのエラーコード取得処理
R_SDC_SD_GetBuffRegAddress()	SD バッファレジスタのアドレス取得処理
R_SDC_SD_1msInterval()	インターバルタイマカウント処理
R_SDC_SD_SetDmacDtcTransFlg()	DMAC/DTC 転送完了フラグセット処理
R_SDC_SD_SetLogHdlAddress()	LONGQ モジュールのハンドラアドレス設定処理　注 2
R_SDC_SD_Log()	エラーログ取得処理　注 2
R_SDC_SD_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、SD 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)　ハードウェア設定

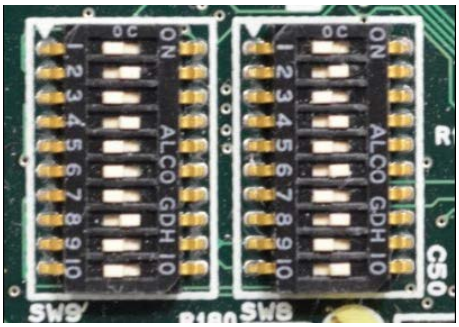
ターゲット MCU の RSK 毎に設定が必要です。

なお、RSK for RX231 の場合、PMOD SD Card 変換基板を別途入手してください。

(a)　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



(b)　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

(c)　RSK for RX65N-2MB

設定不要です。

(d)　RSK for RX231

PMOD SD Card 変換基板を RSK for RX231 上の PMOD2 に装着してください。

(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. e² 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. e² 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_sdc_sdmem_rx_vX.XX.zipとr_sdc_sdmem_rx_vX.XX.xml、r_sdc_sdmem_rx_vX.XX_extend.mdfを格納する。
3. Renesas e² studio スマート・コンフィグレータ ユーザーガイド（R01AN0451）を参照し、r_bsp、r_sdc_sdmem_rx、r_sdhi_rx、r_cmt_rx、r_dmaca_rx、r_dtc_rx、r_longqをプロジェクトに組み込む。
4. サンプルプログラムr_sdc_sdmem_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グ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。

注１：製品パッケージ内の FITDemos¥rx65n_1mb_rsk.zip¥src フォルダに同梱されています。

1.4.2　基本制御

(1)　サポートコマンドについて

SD メモリカードドライバは、以下のコマンドを使用します。

以下の表は、SD カードコマンドとユーザーズマニュアル ハードウェア編と本 SD メモリカードドライバのサポート状況を示したものです。

表 1-4　サポートコマンド一覧（○：サポート、×：未サポート）

コマンド	マイコンのサポート範囲	本ドライバ	備考
CMD0	○	○	SD メモリ初期化で使用
CMD2	○	○	SD メモリ初期化で使用
CMD3	○	○	SD メモリ初期化で使用
CMD4	○	○	SD メモリ初期化で使用
CMD5	○	×	未使用
CMD6	○	×	未使用
CMD7	○	○	SD メモリ初期化で使用
CMD8	○	○	SD メモリ初期化で使用
CMD9	○	○	SD メモリ初期化で使用
CMD10	○	×	未使用
CMD11	○	×	未使用
CMD12	○	○	SD メモリのリード／ライト処理で使用
CMD13	○	○	SD メモリのリード／ライト処理で使用
CMD15	○	×	未使用
CMD16	○	○	SD メモリ初期化で使用
CMD17	○	○	SD メモリのリード／ライト処理で使用
CMD18	○	○	SD メモリのリード／ライト処理で使用
CMD20	○	×	未使用
CMD24	○	○	SD メモリのリード／ライト処理で使用
CMD25	○	○	SD メモリのリード／ライト処理で使用
CMD27	○	×	未使用
CMD28	○	×	未使用
CMD29	○	×	未使用

CMD30	○	×	未使用
CMD32	○	×	未使用
CMD33	○	×	未使用
CMD38	○	×	未使用
CMD42	○	×	未使用
CMD52	○	×	未使用
CMD53	○	×	未使用
CMD55	○	○	SD メモリ初期化で使用
CMD56	○	×	未使用
ACMD6	○	○	SD メモリ初期化で使用
ACMD13	○	○	SD メモリ初期化で使用
ACMD22	○	○	SD メモリのライト処理で使用
ACMD23	○	○	SD メモリのライト処理で使用
ACMD41	○	○	SD メモリ初期化で使用
ACMD42	○	○	SD メモリ初期化で使用
ACMD51	○	○	SD メモリ初期化で使用

(2)　データバッファと SD カード上のデータとの関係

SD カードドライバは、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。

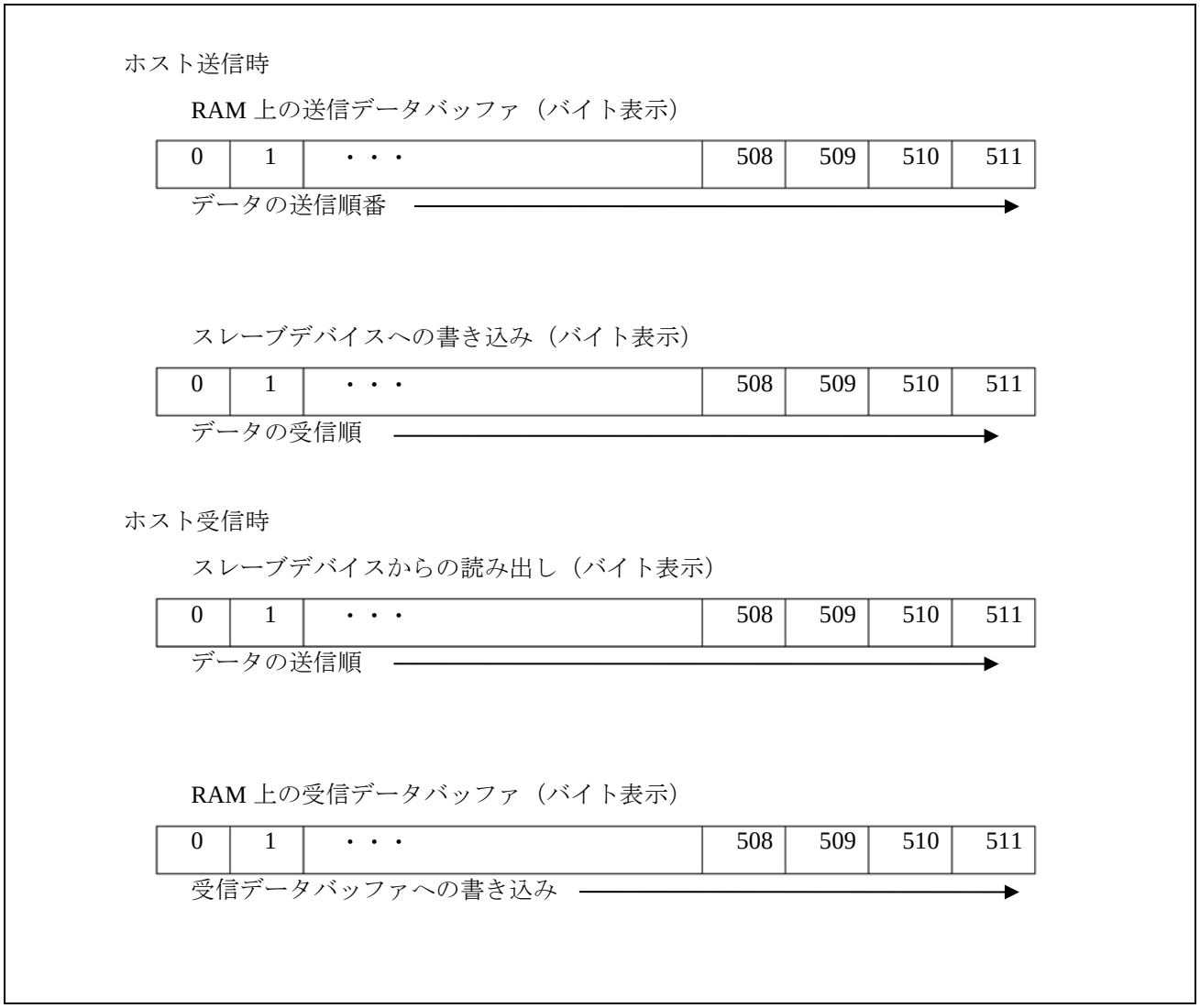


図 1-2　転送データの格納

(3)　初期化処理時の動作電圧設定について

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

SD カードの場合、R_SDC_SD_End()関数をコールし、SD カードの初期化可能状態にした後、SD 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、再度初期化処理を行ってください。

SD モジュールの場合、R_SDC_SD_End()関数をコールし、SD カードの初期化可能状態にした後、SD モジュールへの電源供給を停止してください。その後、SD モジュールへの電源供給を再開し、動作電圧を設定し直して、再度初期化処理を行ってください。

(4)　SDHI_CLK の停止

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

(5)　SDHI ステータス確認

SD カードの操作を行う上で、通信の終了検出等 SDHI のステータス確認や SD カードの挿抜検出を行う必要があります。ここでは、SD カードドライバの 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。

ステータス確認方法

SD カードドライバは、SDHI のステータス確認方法として、SDHI 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。

・SD カード挿抜検出

・SD プロトコル

・SDHI 割り込み

表 1-6 に SD カードドライバの 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
SD カード挿抜 （R_SDC_SD_CdInt()関数で割り込み許可／禁止設定）	SD カード 挿入／抜去状態	R_SDC_SD_GetCardDetection()関数で検出可能
SD プロトコル （R_SDC_SD_Initialize()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

設定方法

SD 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_SDC_SD_Initialize()関数にて SD プロトコルステータス確認方法に割り込み（SDC_SD_MODE_HWINT 設定）を選択してください。

なお、SD カードドライバは SDHI 割り込みに対応する割り込みハンドラとして、SDHI FIT モジュールの R_SDHI_IntHandlerX()関数（X は、チャネル番号）をシステムに登録済です。

ソフトウェアポーリングと割り込みによる SD カード挿抜確認

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

R_SDC_SD_CdInt()関数で割り込み許可（SDC_SD_CD_INT_ENABLE）を設定した場合は、SD 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、SD カードの挿抜に対するリアルタイムの処理が可能です。SD カード挿抜割り込みコールバック関数は、R_SDC_SD_CdInt()関数で登録してください。

ソフトウェアポーリングによる SD プロトコルステータス確認

SD プロトコルのステータス確認方法として、R_SDC_SD_Initialize()関数でポーリング（SDC_SD_MODE_POLL）を設定した場合は、リード／ライト処理の中で、SD 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、r_sdc_sd_int_wait()関数を使用し、この関数内で SD ステータスレジスタ 1,2 取得処理（r_sdc_sd_get_intstatus()関数）をコールし、SD ステータスレジスタ 1,2（SDSTS1, SDSTS2）を確認します。

図 1-3 にポーリングを使用した場合の SD 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。

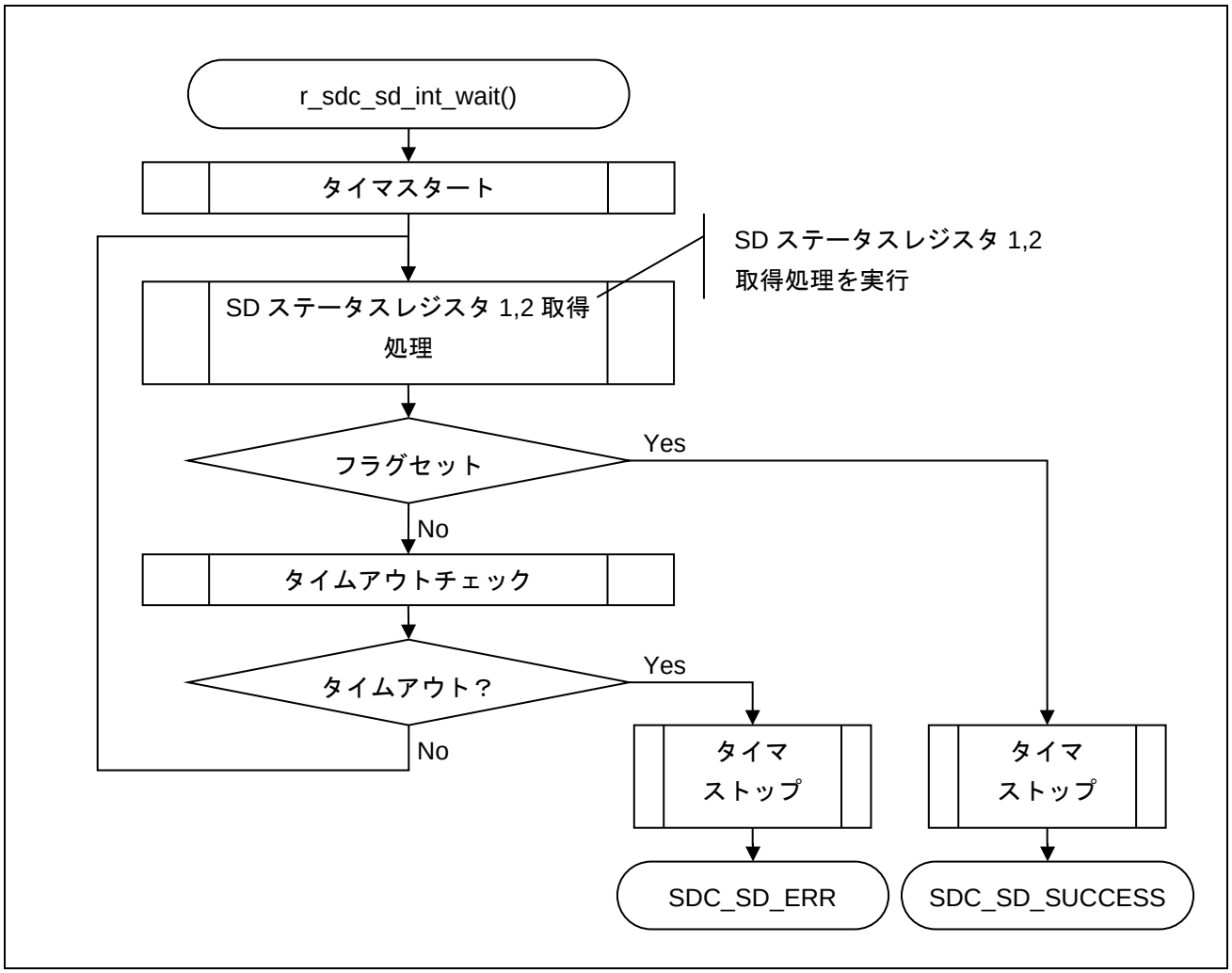


図 1-3　ソフトウェアポーリングによる SD プロトコルステータス確認

割り込みによる SD プロトコルステータス確認方法

SD プロトコルのステータス確認方法として、R_SDC_SD_Initialize()関数で割り込み（SDC_SD_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。SD プロトコルステータス割り込みコールバック関数は R_SDC_SD_IntCallback()関数で登録してください。

割り込み待ち設定時、r_sdc_sd_int_wait()関数を使用し、この関数内で SD ステータスレジスタ 1,2 取得処理（r_sdc_sd_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 SD 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。

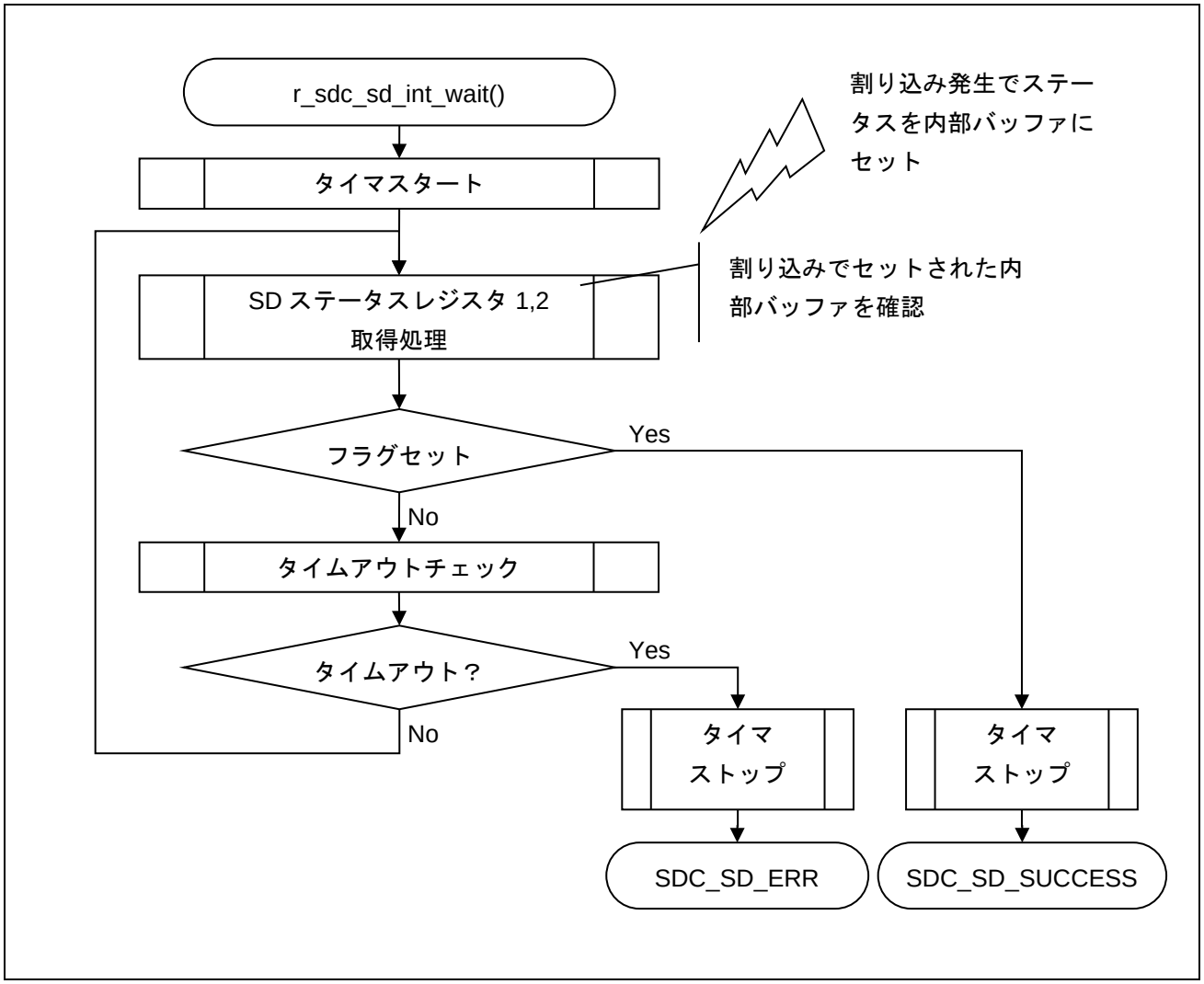


図 1-4　割り込みによる SD プロトコルステータス確認例

1.4.3　エラー時の制御

(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、SD カードの挿抜を実施し、SD カードを再初期化してください。SD カードの挿抜に関わる処理方法は、「4.1」「4.2」を参照してください。SD モジュールの場合、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、SD カード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、SD 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。

(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、SD 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。

(3)　エラーログ取得方法

SD メモリカード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。

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。

制御手順

R_SDC_SD_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「3.22 R_SDC_SD_SetLogHdlAddress()」を参照してください。

1. R_LONGQ_Open()
2. R_SDC_SD_SetLogHdlAddress()

R_SDC_SD_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「3.23 R_SDC_SD_Log()」を参照してください。

1.4.4　　他モジュールの制御

(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_SDC_SD_1msInterval()をコールしてください。但し、r_sdc_sd_config.c の r_sdc_sd_int_wait()および r_sdc_sd_wait()を OS 処理に置き換える場合には不要です。

(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。

SD カードドライバでは、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。

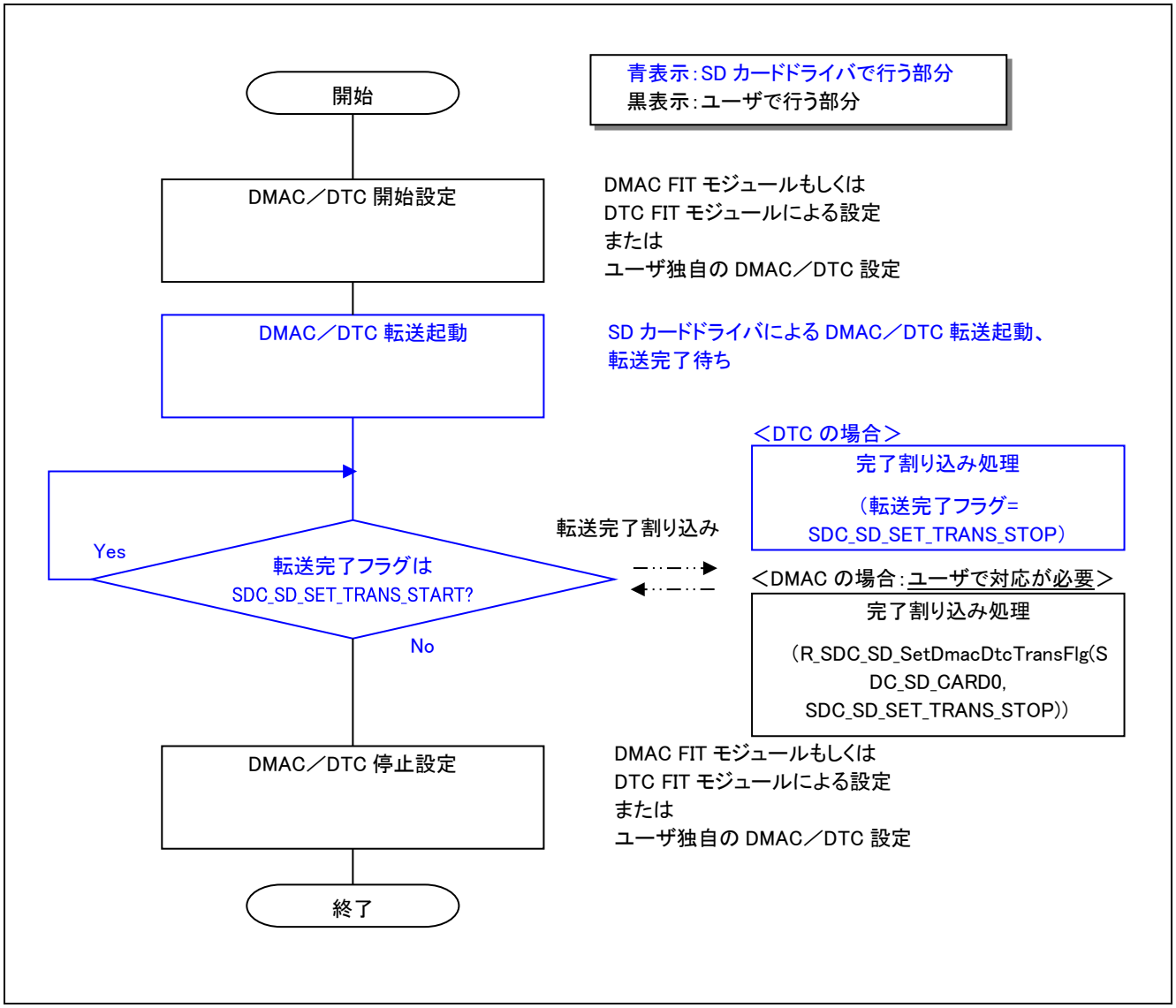
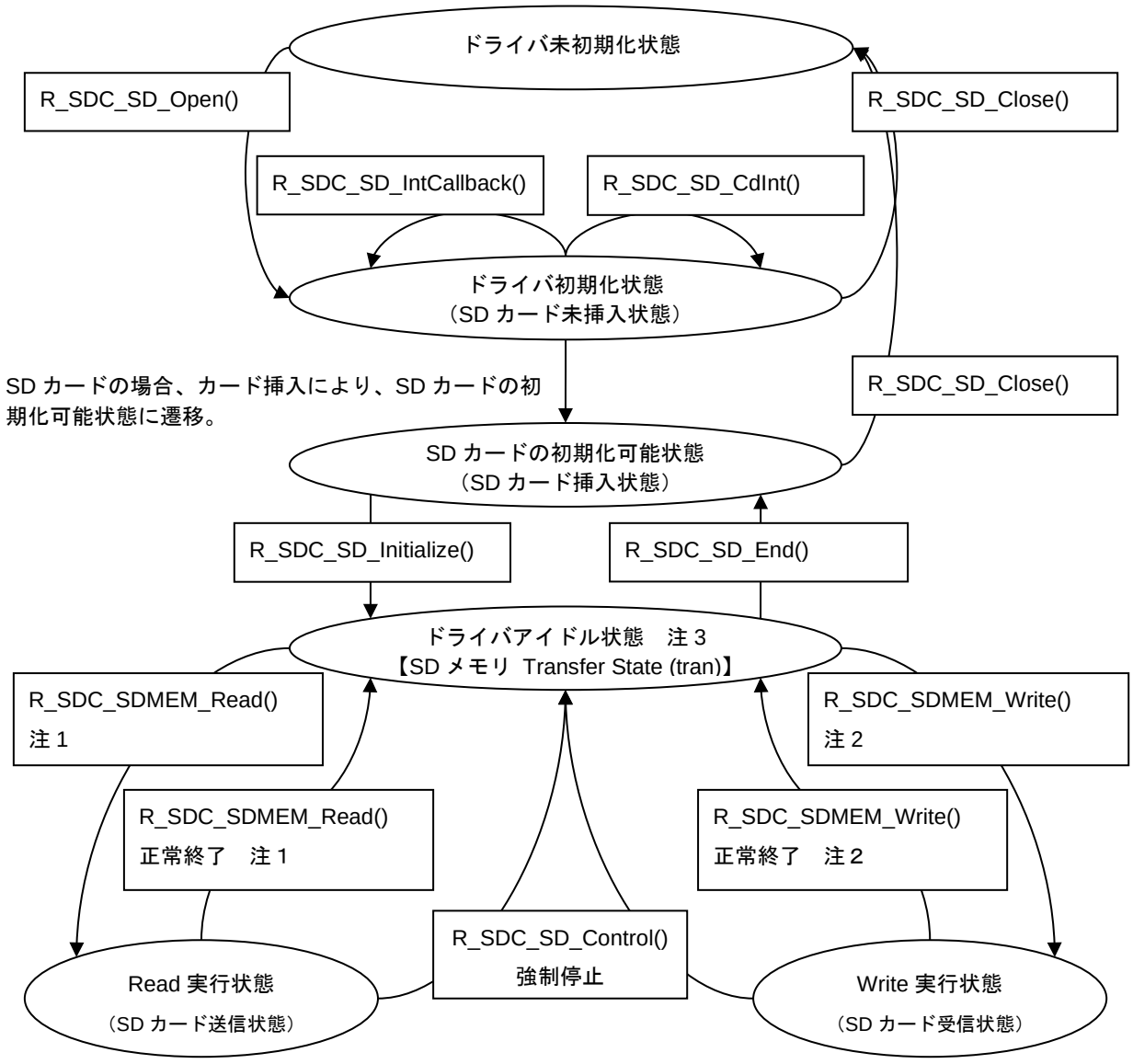


図 1-5　DMAC 転送および DTC 転送設定時の処理

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_SDC_SDMEM_ReadSoftwareTrans()を含む、SD カード送信状態遷移コマンド処理も同様
注 2：R_SDC_SDMEM_WriteSoftwareTrans()を含む、SD カード受信状態遷移コマンド処理も同様
注 3：【】は、SD Specifications Part 1 Physical Layer Simplified Specification 記載の SD メモリカードの状態を示す。

図 1-6　SD メモリカード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。SD カード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　SD カードの電源供給の注意事項

SD カード挿入後、SD カードの仕様に基づいて、SD カード電源電圧を供給する必要があります。SD Specifications Part 1 Physical Layer Specification の Power Scheme の章を参照してください。

特に、SD カードの抜去後の SD カードの再挿入制御、もしくは SD 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。

正しい時間調整が必要です。

また、電源電圧供給停止後、SD カード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　ソフトウェア・ライトプロテクト対応について

SD カードドライバは、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。

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。

- SDHI

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_sdhi_rx
- 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）
- r_longq_rx (エラーログ取得機能を利用する場合のみ)

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

SDHI FIT モジュールのマクロ定義 SDHI_CFG_MODE_INT が SDHI_MODE_HWINT の時、SDHI 割り込みが有効になります。SD メモリカードドライバのオープン処理 R_SDC_SD_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。SDHI 割り込みの詳細は SDHI FIT モジュールのアプリケーションノートを参照してください。

(1) 割り込み処理内からコール可能な API

表 2-1 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 2-1 割り込みハンドラ内からのコールを許可する SD カードドライバ・API 関数一覧

関数名	機能概要	備考
R_SDC_SD_Control()	ドライバのコントロール処理	SDC_SD_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_sdc_sd_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_sdc_sd_rx_config.h で選択します。

```
#include "r_sdc_sd_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グレーションオプションの設定は、r_sdc_sd_rx_config.h で行います。

スマート・コンフィグレータ を使用する場合は、ソフトウェアコンポーネント設定画面でコンフィグレーションオプションを設定できます。設定値はモジュールを追加する際に、自動的に r_sdc_sd_rx_config.h に反映されます。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。

Configuration options in r_sdc_sd_rx_config.h	
#define SDC_SD_CFG_STATUS_CHECK_MODE ※デフォルト値は(SDC_SD_MODE_HWINT) “ステータス確認：ハードウェア割り込み、	この定義を初期化処理 R_SDC_SD_Initialize()関数の引数 p_sdc_sd_config->mode として使用できます。 「初期化処理 R_SDC_SD_Initialize()」を参照し、p_sdc_sd_config->mode を定義してください。
#define SDC_SD_CFG_TRANSFER ※デフォルト値は SDC_SD_MODE_SW “データ転送方式：ソフトウェア転送“	この定義を初期化処理 R_SDC_SD_Initialize()関数の引数 p_sdc_sd_config->mode として使用できます。 「初期化処理 R_SDC_SD_Initialize()」を参照し、p_sdc_sd_config->mode を定義してください。
#define SDC_SD_CFG_ERROR_LOG_ACQUISITION ※デフォルト値は"0"	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 1 を定義してください。 この機能を利用する場合、LONGQ FIT モジュールを組み込む必要があります。

2.8　コードサイズ

本モジュールの ROM サイズ、RAM サイズ、最大使用スタックサイズを下表に示します。RX200 シリーズ、RX600 シリーズから代表して 1 デバイスずつ掲載しています。

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_sdc_sdmem rev.3.00

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 4.8.4.201803
(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)
IAR C/C++ Compiler for Renesas RX version 4.11.1
(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

ROM、RAM およびスタックのコードサイズ

デバイス	分類	使用メモリ					
		Renesas Compiler		GCC		IAR Compiler	
		パラメータチェックあり	パラメータチェックなし	パラメータチェックあり	パラメータチェックなし	パラメータチェックあり	パラメータチェックなし
RX231	ROM	2,636 バイト	2,079 バイト	4,460 バイト	3,396 バイト	3,760 バイト	2,932 バイト
	RAM	12 バイト		12 バイト		8 バイト	
	スタック (注 1)	164 バイト		-		124 バイト	
RX65N	ROM	5,595 バイト	4,424 バイト	9,588 バイト	7,324 バイト	7,777 バイト	6,147 バイト
	RAM	40 バイト		40 バイト		32 バイト	
	スタック (注 1)	188 バイト		-		148 バイト	

注1.　割り込み関数の最大使用スタックサイズを含みます。

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_sdc_sd_rx_if.h に記載されています。

(1)　e_enum_sdc_sd_cmd 構造体定義

```
enum e_enum_sdc_sd_cmd
{
    SDC_SD_SET_STOP,
    SDC_SD_SET_BUFFER
} enum_sdc_sd_cmd_t;
```

(2)　e_enum_sdc_sd_trans 構造体定義

```
enum e_enum_sdc_sd_trans
{
    SDC_SD_SET_TRANS_STOP,
    SDC_SD_SET_TRANS_START
} enum_sdc_sd_trans_t;
```

(3)　sdc_sd_cmd_t 構造体定義

```
typedef struct
{
    uint32_t    cmd;
    uint32_t    mode;
    uint8_t     *p_buff;
    uint32_t    size;
}sdc_sd_cmd_t;
```

(4)　sdc_sd_cfg_t 構造体定義

```
typedef struct
{
    uint32_t    mode;
    uint32_t    voltage;
}sdc_sd_cfg_t;
```

(5)　sdc_sd_access_t 構造体定義

```
typedef struct
{
    uint8_t     *p_buff;
    uint32_t    lbn;
    int32_t     cnt;
    uint32_t    mode;
    uint32_t    write_mode;
}sdc_sd_access_t;
```

(6)　sdc_sd_card_status_t 構造体定義

```
typedef struct
{
    uint32_t card_sector_size;
    uint32_t prot_sector_size;
    uint8_t  write_protect;
    uint8_t  media_type;
    uint8_t  csd_structure;
    uint8_t  speed_mode;
    uint8_t  io_speed_mode;
    uint8_t  rsv[3];
}sdc_sd_card_status_t;
```

(7)　sdc_sd_card_reg_t 構造体定義

```
typedef struct
{
    uint32_t    sdio_ocr[1];
    uint32_t    ocr[1];
    uint32_t    cid[4];
    uint32_t    csd[4];
    uint32_t    dsr[1];
    uint32_t    rca[2];
    uint32_t    scr[2];
    uint32_t    sdstatus[4];
    uint32_t    swtich_func_status[5];
  }sdc_sd_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_sdc_sd_rx_if.h で記載されています。

SD カードドライバの 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_SDC_SD_Initialize()関数／R_SDC_SDMEM_Read()関数／R_SDC_SDMEM_Write()関数

注 1 実行後に、R_SDC_SD_GetErrCode()関数を使ってエラーコードを取得できます。

表 2-3 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_SDC_SDMEM_ReadSoftwareTrans()関数／R_SDC_SDMEM_ReadSoftwareTransSingleCmd()関数／R_SDC_SDMEM_WriteSoftwareTrans()関数も同様です。

表 2-2　エラーコード

マクロ定義	値	意味	
SDC_SD_SUCCESS_LOCKED_CARD	1	正常終了	正常終了であるが、SD カードがロック中
SDC_SD_SUCCESS	0	正常終了	正常終了
SDC_SD_ERR	-1	一般エラー	R_SDC_SD_Open()関数が実行されていない、引数パラメータエラー等
SDC_SD_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 SD カードへの書き込み
SDC_SD_ERR_RO	-3	リードオンリーエラー	リードオンリーエラー

SDC_SD_ERR_RES_TOE	-4	レスポンスタイムアウトエラー	コマンドに対するレスポンスが 640 クロック（SDHI クロック）以内に受信できなかった
SDC_SD_ERR_CARD_TOE	-5	カードタイムアウトエラー	カードビジー状態のタイムアウト、リードコマンド後のデータ受信タイムアウト、ライトコマンド後の CRC ステータス受信タイムアウト（※） ※カードアクセスオプションレジスタ（SDOPT）の b7-b4（TOP）の設定に依存
SDC_SD_ERR_END_BIT	-6	エンドビットエラー	エンドビットを検出できなかった
SDC_SD_ERR_CRC	-7	CRC エラー	ホストが CRC エラーを検出した
SDC_SD_ERR_CARD_RES	-8	カードレスポンスエラー	
SDC_SD_ERR_HOST_TOE	-9	ホストタイムアウトエラー	r_sdc_sd_int_wait()関数のエラー
SDC_SD_ERR_CARD_ERASE	-10	カードイレースエラー	R1 レスポンスのカードステータスエラー（ERASE_SEQ_ERROR または ERASE_PARAM）
SDC_SD_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）
SDC_SD_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）
SDC_SD_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）
SDC_SD_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）
SDC_SD_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）
SDC_SD_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）
SDC_SD_ERR_CARD_TYPE	-17	未対応カード	未対応のカードと認識した
SDC_SD_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
SDC_SD_ERR_ILL_READ	-19	不正読み出しエラー	SD バッファレジスタリード方法が不正
SDC_SD_ERR_ILL_WRITE	-20	不正書き込みエラー	SD バッファレジスタライト方法が不正
SDC_SD_ERR_AKE_SEQ	-21	カード AKE エラー	R1 レスポンスのカードステータスエラー (AKE_SEQ_ERROR)
SDC_SD_ERR_OVERWRITE	-22	カード OVERWRITE エラー	R1 レスポンスのカードステータスエラー (CSD_OVERWRITE)
SDC_SD_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_sdc_sd_int_wait()関数以外）
SDC_SD_ERR_STOP	-31	強制停止エラー	R_SDC_SD_Control()関数による強制停止状態
SDC_SD_ERR_CSD_VER	-50	バージョンエラー	CSD レジスタバージョンエラー

SDC_SD_ERR_FILE_FORMAT	-52	ファイルフォーマットエラー	CSD レジスタファイルフォーマットエラー
SDC_SD_ERR_ILL_FUNC	-60	ファンクション No エラー	無効なファンクション No 要求エラー
SDC_SD_ERR_IFCOND_VOLT	-71	インタフェースコンディション・ボルテージエラー	インタフェースコンディションの電圧値が不正
SDC_SD_ERR_IFCOND_ECHO	-72	インタフェースコンディション・エコーバックエラー	インタフェースコンディションのエコーバックパターンが不正
SDC_SD_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）
SDC_SD_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）
SDC_SD_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）
SDC_SD_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）
SDC_SD_ERR_CMD_ERROR	-86	コマンドインデックスエラー	内部エラー （送信コマンドインデックスと受信コマンドインデックスが異なる）
SDC_SD_ERR_CBSY_ERROR	-87	コマンドエラー	SDHI 内部エラー　（コマンドビジー）
SDC_SD_ERR_NO_RESP_ERROR	-88	レスポンスなしエラー	SDHI 内部エラー （レスポンスを受信できない）
SDC_SD_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
SDC_SD_ERR_UNSUPPORTED_TYPE	-97	未対応 SDIO アクセスエラー	未対応 SDIO アクセスエラー
SDC_SD_ERR_API_LOCK	-98	API ロックエラー	API コール中に API をコールした場合のエラー
SDC_SD_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、SD カードの挿抜割り込み、又は SD 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「3.16 R_SDC_SD_CdInt()」「3.17 R_SDC_SD_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) SDHI 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、スマート・コンフィグレータを使用した(1)、(3)の追加方法を推奨しています。ただし、スマート・コンフィグレータは、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(2)、(4)の方法を使用してください。

(1)　e² studio 上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
e² studio のスマート・コンフィグレータを使用して、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)　e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、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)」を参照してください。

(3)　CS+上でスマート・コンフィグレータ を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版スマート・コンフィグレータを使用して、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。

(4)　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などで 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、「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

3.1 R_SDC_SD_Open()

SD カードドライバの API を使用する際、最初に使用する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_Open(
uint32_t    card_no,
uint32_t    channel,
void        *p_sdc_sd_workarea
)
```

Parameters

card_no

SD カード番号　使用する SD カード番号（0 起算）

channel

チャネル番号　使用する SDHI チャネル番号（0 起算）

*p_sdc_sd_workarea

4 バイト境界のワーク領域のポインタ（領域サイズ 200 バイト）

Return Values

SDC_SD_SUCCESS	正常終了
SDC_SD_ERR	一般エラー
SDC_SD_ERR_CPU_IF	ターゲット MCU インタフェース関数エラー
SDC_SD_ERR_ADDRESS_BOUNDARY	引数のバッファアドレスエラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

引数 card_no で制御する SDHI チャネルリソースを取得し、引数 channel で指定した SD カードドライバと SDHI FIT モジュールを初期化します。また、その SDHI チャネルリソースを占有します。

SD カード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_sdc_sd_work[200/sizesof(uint32_t)];

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

if (R_SDC_SD_Open(SDC_SD_CARD0, SDHI_CH0, &g_sdc_sd_work) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に、端子設定が必要です。「4.1SD 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_SDC_SD_GetVersion()関数、R_SDC_SD_Log()関数、R_SDC_SD_SetLogHdlAddress()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。SD カード挿抜割り込みを使用する場合は、本関数実行後、R_SDC_SD_CdInt()関数にて挿抜割り込みを許可にしてください。

R_SDC_SD_GetErrCode()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。

3.2　R_SDC_SD_Close()

使用中の SD カードドライバのリソースを開放する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_Close(
uint32_t    card_no
)
```

Parameters

card_no

SD カード番号	使用する SD カード番号（0 起算）

Return Values

SDC_SD_SUCCESS	正常終了
SDC_SD_ERR	一般エラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

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

その SDHI 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_SDC_SD_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_SDC_SD_Close(SDC_SD_CARD0) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行後に、端子設定が必要です。「4.2 SD 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_SDC_SD_Open()関数によるドライバのオープン処理が必要です。

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

3.3　R_SDC_SD_GetCardDetection()

SD カードの挿入状態を確認する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_GetCardDetection(
uint32_t    card_no
)
```

Parameters

card_no

SD カード番号	使用する SD カード番号（0 起算）

Return Values

SDC_SD_SUCCESS	SDHI_CD 端子レベルは Low、またはカード検出無効時
SDC_SD_ERR	SDHI_CD 端子レベルは High

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD カードの挿入状態を確認します。

＜カード検出有効の場合＞

SDHI_CD 端子レベルが Low の場合、SDC_SD_SUCCESS を返します。

SDHI_CD 端子レベルが High の場合、SDC_SD_ERR を返します。

＜カード検出無効の場合＞

常に SDC_SD_SUCCESS を返します。

Example

```
if (R_SDC_SD_GetCardDetection(SDC_SD_CARD0) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

カード検出を有効にするためには SDHI FIT モジュールの#define SDHI_CFG_CHx_CD_ACTIVE を”1”に設定してください。

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。「4.1 SD 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_SDC_SD_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。「4.2 SD 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。

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

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

SD カード検出後、SD カードへの電源電圧供給処理が必要です。

3.4　R_SDC_SD_Initialize()

SD カードを初期化し、SD カードの初期化可能状態からドライバアイドル状態にする関数です。

Format

```
sdc_sd_status_t R_SDC_SD_Initialize(
uint32_t        card_no,
sdc_sd_cfg_t    *p_sdc_sd_config,
uint32_t        init_type
)
```

Parameters

card_no

　SD カード番号　　使用する SD カード番号（0 起算）

*p_sdc_sd_config

　動作設定情報構造体

mode　：動作モード

　「表 3-1　SD カードドライバ　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

　SD 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 SD カードは初期化されません。「1.4.2(3) 初期化処理時の動作電圧設定について」も参照してください。

表 3-1　SD カードドライバ　動作モード mode

種別	マクロ定義	値（ビット）	定義
プロトコルステータス確認方法	SDC_SD_MODE_POLL	0x0000	ソフトウェアポーリング
	SDC_SD_MODE_HWINT	0x0001	割り込み
データ転送方式	SDC_SD_MODE_SW	0x0000	Software 転送
	SDC_SD_MODE_DMA（注 1、注 4）	0x0002	DMAC 転送（注 3）
	SDC_SD_MODE_DTC（注 2、注 4）	0x0004	DTC 転送（注 3）
メディア対応方式	SDC_SD_MODE_MEM	0x0000	SD メモリカード／SD メモリ
SD バス対応方式	SDC_SD_MODE_4BIT	0x0200	SD モード 4 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：使用する関数によっては Software 転送を行います。
注 4：SDC_SD_MODE_DMA と SDC_SD_MODE_DTC を同時にセットしないでください。

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	SDC_SD_VOLT_2_8	0x00008000
2.8-2.9	SDC_SD_VOLT_2_9	0x00010000
2.9-3.0	SDC_SD_VOLT_3_0	0x00020000
3.0-3.1	SDC_SD_VOLT_3_1	0x00040000
3.1-3.2	SDC_SD_VOLT_3_2	0x00080000
3.2-3.3	SDC_SD_VOLT_3_3	0x00100000
3.3-3.4	SDC_SD_VOLT_3_4	0x00200000
3.4-3.5	SDC_SD_VOLT_3_5	0x00400000
3.5-3.6	SDC_SD_VOLT_3_6	0x00800000

init_type　：初期化タイプ

　初期化対象を指定してください。値は「表 3-1　SD カードドライバ　動作モード mode」のメディア対応方式のマクロ定義を使用してください。

Return Values

SDC_SD_SUCCESS	正常終了
SDC_SD_SUCCESS_LOCKED_CARD	正常終了、かつ、SD 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_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD カードの初期化処理を行います。SD カードの検出後に、本関数を実行してください。

SD カードと認識した場合は、SD カード内の CD/DAT3 端子のプルアップを無効にします。

戻り値が、SDC_SD_SUCCESS の場合、SD カードは Transfer State (tran)へ遷移し、SD カードのリード／ライトアクセスが可能になります。SDC_SD_SUCCESS_LOCKED_CARD の場合、SD カードは Transfer State (tran)へ遷移しますが、SD カードリード／ライトアクセスはできません。この場合、エラーコードとして SDC_SD_ERR_CARD_LOCK が登録されます。ロックされた SD カードは特定のコマンドしか受け付けません。ロック状態の SD カードにリード／ライトアクセスするためには、SD カードをアンロック状態にした後に、再度 SD カードの初期化処理を行ってください。

Example

```
sdc_sd_cfg_t                    sdc_sd_config;

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

sdc_sd_config.mode = SDC_SD_CFG_DRIVER_MODE;
sdc_sd_config.voltage = SDC_SD_VOLT_3_3;
if (R_SDC_SD_Initialize(SDC_SD_CARD0, &sdc_sd_config, SDC_SD_MODE_MEM) !=
SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

SD カードドライバは、初期化時に Default Speed を設定します。

本関数実行前に、端子設定が必要です。「4.1 SD 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_SDC_SD_Open()関数によるドライバのオープン処理が必要です。

エラー終了の場合、R_SDC_SD_End()関数をコールし、SD カードの初期化可能状態にした後、再度初期化処理を行ってください。

初期化正常終了後、再初期化処理を行う前に、終了処理を行ってください。

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

SDHI は、3.3V 信号レベルのみをサポートします。そのため、SD カードドライバは、SD カード初期化時にコマンドの引数 S18R bit=0 を設定し ACMD41 を発行します。

3.5　R_SDC_SD_End()

ワーク領域の値をクリアし、ドライバアイドル状態から SD カードの初期化可能状態にする関数です。本関数を実行した場合でも SD カードの状態は変化しません。

Format

```
sdc_sd_status_t R_SDC_SD_End(
uint32_t    card_no,
uint32_t    end_type
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

end_type

終了タイプ

終了対象を指定してください。値は、R_SDC_SD_Initialize()関数の「表 3-1　SD カードドライバ　動作モード mode」のメディア対応方式のマクロ定義を使用してください。

Return Values

SDC_SD_SUCCESS　　正常終了

SDC_SD_ERR　　一般エラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD カードの終了処理を行います。

SD カードの場合、SD 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 SD カードの初期化可能状態にした場合であっても、SD カードの挿抜割り込みおよび SD 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_SDC_SD_End(SDC_SD_CARD0, SDC_SD_MODE_MEM) != SDC_SD_SUCCESS)
{
            /* Error */
}

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

Special Notes

本関数実行後、SD カードを抜去する場合、端子設定が必要です。「4.4 SD 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_SDC_SD_Open()関数によるドライバのオープン処理を行ってください。

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

3.6　R_SDC_SDMEM_Read()

リード処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDMEM_Read(
uint32_t           card_no,
sdc_sd_access_t    *p_sdc_sd_access
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

*p_sdc_sd_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）

write_mode ：書き込みモード（設定不要）

Return Values

SDC_SD_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_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

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

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

本関数開始時に、R_SDC_SD_Control()の SDC_SD_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（SDC_SD_ERR_STOP）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。

2 ブロック以下：READ_SINGLE_BLOCK コマンド（CMD17）

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

Example

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

sdc_sd_access_t         sdc_sd_access;
uint32_t
      g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff    = (uint8_t *)&g_test_r_buff[0];
sdc_sd_access.lbn       = 0x10000000;
sdc_sd_access.cnt       = TEST_BLOCK_CNT;

if(R_SDC_SDMEM_Read(SDC_SD_CARD0, &sdc_sd_access) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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。

DMAC 転送／DTC 転送設定時、データ転送期間中は SD カード挿抜割り込み禁止状態に設定します。

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。

3.7　R_SDC_SDMEM_ReadSoftwareTrans()

リード処理（Software 転送）を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDMEM_ReadSoftwareTrans(
uint32_t           card_no,
sdc_sd_access_t    *p_sdc_sd_access
)
```

Parameters

card_no

　SD カード番号　　使用する SD カード番号（0 起算）

*p_sdc_sd_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）

　write_mode　：書き込みモード（設定不要）

Return Values

SDC_SD_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_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

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

初期化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、SD カードの抜去を検出した場合、処理を中止しエラー終了（SDC_SD_ERR_STOP）を返します。

本関数開始時に、R_SDC_SD_Control()の SDC_SD_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（SDC_SD_ERR_STOP）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。

　2 ブロック以下：READ_SINGLE_BLOCK コマンド（CMD17）

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

Example

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

sdc_sd_access_t         sdc_sd_access;
uint32_t
      g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff    = (uint8_t *)&g_test_r_buff[0];
sdc_sd_access.lbn       = 0x10000000;
sdc_sd_access.cnt       = TEST_BLOCK_CNT;

if(R_SDC_SDMEM_ReadSoftwareTrans(SDC_SD_CARD0, &sdc_sd_access) !=
SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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。

3.8　R_SDC_SDMEM_ReadSoftwareTransSingleCmd()

リード処理（CMD17 シングル Software 転送）を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDMEM_ReadSoftwareTransSingleCmd(
uint32_t           card_no,
sdc_sd_access_t    *p_sdc_sd_access
)
```

Parameters

card_no

　SD カード番号　　使用する SD カード番号（0 起算）

*p_sdc_sd_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）

　write_mode　：書き込みモード（設定不要）

Return Values

SDC_SD_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_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

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

初期化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、SD カードの抜去を検出した場合、処理を中止しエラー終了（SDC_SD_ERR_STOP）を返します。

本関数開始時に、R_SDC_SD_Control()の SDC_SD_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（SDC_SD_ERR_STOP）を返します。

ブロックデータの読み出しには、CMD17 コマンドのみ使用します。

Example

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

sdc_sd_access_t         sdc_sd_access;
uint32_t
      g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff    = (uint8_t *)&g_test_r_buff[0];
sdc_sd_access.lbn       = 0x10000000;
sdc_sd_access.cnt       = TEST_BLOCK_CNT;

if(R_SDC_SDMEM_ReadSoftwareTransSingleCmd(SDC_SD_CARD0, &sdc_sd_access) !=
SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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。

3.9　R_SDC_SDMEM_Write()

ライト処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDMEM_Write(
uint32_t           card_no,
sdc_sd_access_t    *p_sdc_sd_access
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

*p_sdc_sd_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）

write_mode ：書き込みモード

設定値は「表 3-3　SD カードドライバ　書き込みモード write_mode」のマクロ定義に示す種別から１つ設定してください。

Return Values

SDC_SD_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_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

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

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

本関数開始時に、R_SDC_SD_Control()の SDC_SD_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（SDC_SD_ERR_STOP）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。

2 ブロック以下：WRITE_SINGLE_BLOCK コマンド（CMD24）
3 ブロック以上：WRITE_MULTIPLE_BLOCK コマンド（CMD25）

表 3-3　SD カードドライバ　書き込みモード write_mode

種別	マクロ定義	値（ビット）
プレイレーズを伴う書き込み	SDC_SD_WRITE_WITH_PREERASE	0x00000000
通常の書き込み	SDC_SD_WRITE_OVERWRITE	0x00000001

Example

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

sdc_sd_access_t         sdc_sd_access;
uint32_t
      g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff    = (uint8_t *)&g_test_w_buff[0];
sdc_sd_access.lbn       = 0x10000000;
sdc_sd_access.cnt       = TEST_BLOCK_CNT;
sdc_sd_access.write_mode      = SDC_SD_WRITE_OVERWRITE;

if(R_SDC_SDMEM_Write(SDC_SD_CARD0, &sdc_sd_access) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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。

SDC_SD_WRITE_WITH_PREERASE 設定時のライト処理中に SD カードの抜去が発生した場合、SDC_SD_WRITE_OVERWRITE 設定時と比べ、SD メモリのデータが失われる可能性が高くなります。

DMAC 転送／DTC 転送設定時、データ転送期間中は SD カード挿抜割り込み禁止状態に設定します。

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。

3.10　R_SDC_SDMEM_WriteSoftwareTrans()

ライト処理（Software 転送）を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDMEM_WriteSoftwareTrans(
uint32_t           card_no,
sdc_sd_access_t    *p_sdc_sd_access
)
```

Parameters

card_no

　SD カード番号　　使用する SD カード番号（0 起算）

*p_sdc_sd_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

　設定値は「表 3-4　SD カードドライバ　書き込みモード write_mode」のマクロ定義に示す種別から 1 つ設定してください。

Return Values

SDC_SD_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_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

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

初期化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、SD カードの抜去を検出した場合、処理を中止しエラー終了（SDC_SD_ERR_STOP）を返します。

本関数開始時に、R_SDC_SD_Control()の SDC_SD_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（SDC_SD_ERR_STOP）を返します。

2 ブロック以下：WRITE_SINGLE_BLOCK コマンド（CMD24）

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

表 3-4　SD カードドライバ　書き込みモード write_mode

種別	マクロ定義	値（ビット）
プレイレーズを伴う書き込み	SDC_SD_WRITE_WITH_PREERASE	0x00000000
通常の書き込み	SDC_SD_WRITE_OVERWRITE	0x00000001

Example

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

sdc_sd_access_t         sdc_sd_access;
uint32_t
      g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff    = (uint8_t *)&g_test_w_buff[0];
sdc_sd_access.lbn       = 0x10000000;
sdc_sd_access.cnt       = TEST_BLOCK_CNT;
sdc_sd_access.write_mode      = SDC_SD_WRITE_OVERWRITE;

if(R_SDC_SDMEM_WriteSoftwareTrans(SDC_SD_CARD0, &sdc_sd_access) !=
SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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。

SDC_SD_WRITE_WITH_PREERASE 設定時のライト処理中に SD カードの抜去が発生した場合、SDC_SD_WRITE_OVERWRITE 設定時と比べ、SD メモリのデータが失われる可能性が高くなります。

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。

3.11　R_SDC_SD_Control()

ドライバのコントロール処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_Control(
uint32_t        card_no,
sdc_sd_cmd_t    *p_sdc_sd_cmd
)
```

Parameters

card_no

　SD カード番号　　使用する SD カード番号（0 起算）

p_sdc_sd_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

SDC_SD_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_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD 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 3-5　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 3-5　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
SDC_SD_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_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。R_SDC_SD_GetErrCode()関数によるエラーコード取得はできません。

(a) SDC_SD_SET_STOP

リード／ライト処理を強制終了します。

Return Values

SDC_SD_SUCCESS	正常終了

Description

強制停止を要求し、SD カード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。

＜SD メモリの場合＞

SD カード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、SD カードに対して 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、SD 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、SD カードが 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。そのため、R_SDC_SDMEM_Read()関数／R_SDC_SDMEM_Write()関数注 1 をコールした場合、エラー終了（SDC_SD_ERR_STOP）を返します。

Example

```
sdc_sd_cmd_t                    sdc_sd_cmd;

sdc_sd_cmd.cmd = SDC_SD_SET_STOP;
sdc_sd_cmd.mode = 0;
sdc_sd_cmd.p_buff = 0;
sdc_sd_cmd.size = 0;

if (R_SDC_SD_Control(SDC_SD_CARD0, &sdc_sd_cmd) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

ライト処理中に強制停止させた場合、SD カード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、SD カード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_SDC_SDMEM_Read()関数／R_SDC_SDMEM_Write()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_SDC_SDMEM_Read()関数／R_SDC_SDMEM_Write()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_SDC_SDMEM_ReadSoftwareTrans()関数／R_SDC_SDMEM_ReadSoftwareTransSingleCmd()関数／R_SDC_SDMEM_WriteSoftwareTrans()関数も同様です。

3.12　R_SDC_SD_GetModeStatus()

転送モードステータス情報を取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_GetModeStatus(
uint32_t    card_no,
uint8_t     *p_mode
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（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　SD カードドライバ　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

SDC_SD_SUCCESS	正常終了
SDC_SD_ERR	一般エラー

Properties

r_sdc_sd_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_SDC_SD_GetModeStatus(SDC_SD_CARD0, p_mode) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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

3.13　R_SDC_SD_GetCardStatus()

カードステータス情報を取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_GetCardStatus(
uint32_t                card_no,
sdc_sd_card_status_t    *p_sdc_sd_card_status
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

*p_sdc_sd_card_status

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

prot_sector_size　：プロテクト領域ブロック数

write_protect　：ライトプロテクト情報（「表 3-6　ライトプロテクト情報 write_protect を参照）

media_type　：メディアタイプ（「表 3-7　メディアタイプ media_type」を参照）

csd_structure　：CSD 情報

0 : Standard Capacity カード（SDSC）

1 : High Capacity カード（SDHC, SDXC）

speed_mode　：予約

io_speed_mode　：予約

Return Values

SDC_SD_SUCCESS　　正常終了

SDC_SD_ERR　　一般エラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD 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。

表 3-6　ライトプロテクト情報 write_protect

マクロ定義	値（ビット）	定義
SDC_SD_WP_OFF	0x00	ライトプロテクト解除状態
SDC_SD_WP_HW	0x01	ハードウェア・ライトプロテクト状態
SDC_SD_WP_TEMP	0x02	CSD レジスタ TEMP_WRITE_PROTECT ビット ON
SDC_SD_WP_PERM	0x04	CSD レジスタ PERM_WRITE_PROTECT ビット ON
SDC_SD_WP_ROM	0x10	SD ROM

表 3-7　メディアタイプ media_type

マクロ定義	値（ビット）	定義
SDC_SD_MEDIA_UNKNOWN	0x00	不明
SDC_SD_MEDIA_SDMEM	0x20	SD メモリカード／SD メモリ
SDC_SD_MEDIA_SDIO	0x01	SDIO カード／SDIO
SDC_SD_MEDIA_COMBO	0x21	SD Combo カード（SD メモリと SDIO の論理和）

Example

```
sdc_sd_card_status_t          sdc_sd_card_status;

if (R_SDC_SD_GetCardStatus(SDC_SD_CARD0, &sdc_sd_card_status) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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

3.14　R_SDC_SD_GetCardInfo()

SD カードレジスタ情報を取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_GetCardInfo(
uint32_t             card_no,
sdc_sd_card_reg_t    *p_sdc_sd_card_reg
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

*p_sdc_sd_card_reg

SD カードのレジスタ情報構造体ポインタ

sdio_ocr[1] ：SDIO OCR 情報

ocr[1] ：SD メモリ OCR 情報

cid[4] ：SD メモリ CID 情報

csd[4] ：SD メモリ CSD 情報

dsr[1] ：SD メモリ DSR 情報

rca[2] ：SDIO、SD メモリ RCA 情報

scr[2] ：SD メモリ SCR 情報

sdstatus[4] ：SD メモリ SD Status 情報

switch_func_status[5] ：予約

Return Values

SDC_SD_SUCCESS　　正常終了

SDC_SD_ERR　　一般エラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

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

Example

```
sdc_sd_card_reg_t       sdc_sd_card_reg;

if (R_SDC_SD_GetCardInfo(SDC_SD_CARD0, &sdc_sd_card_reg) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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

3.15　R_SDC_SDMEM_GetSpeed()

SD カードの Speed Class 情報と Performance Move 情報を取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SDMEM_GetSpeed(
uint32_t    card_no,
uint8_t     *p_clss,
uint8_t     *p_move
)
```

Parameters

card_no

SD カード番号　使用する SD カード番号（0 起算）

*p_clss

Speed Class 情報格納ポインタ（1 バイト）（「表 3-8　Speed Class 情報」を参照）

*p_move

Performance Move 情報格納ポインタ（1 バイト）（「表 3-9　Performance Move 情報」を参照）

Return Values

SDC_SD_SUCCESS　正常終了

SDC_SD_ERR　一般エラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD カードの Speed Class 情報と Performance Move 情報を取得します。

表 3-8　Speed Class 情報

マクロ定義	値	定義
SDC_SD_SPEED_CLASS_0	0x00	Speed Class 0
SDC_SD_SPEED_CLASS_2	0x01	Speed Class 2
SDC_SD_SPEED_CLASS_4	0x02	Speed Class 4
SDC_SD_SPEED_CLASS_6	0x03	Speed Class 6
SDC_SD_SPEED_CLASS_10	0x04	Speed Class 10
（定義なし）	0x05 – 0xFF	Reserved

値と定義は SD Specifications Part 1 Physical Layer Simplified Specification と同じです。

表 3-9　Performance Move 情報

値	定義
0x00	Sequential Write
0x01	1MB/sec
0x02	2MB/sec
・・・	・・・
0xFE	254MB/sec
0xFF	Infinity

値と定義は SD Specifications Part 1 Physical Layer Simplified Specification と同じです。

Example

```
uint8_t           clss;
uint8_t           move;

if (R_SDC_SDMEM_GetSpeed(SDC_SD_CARD0, &clss, &move) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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

3.16　R_SDC_SD_CdInt()

SD 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

```
sdc_sd_status_t R_SDC_SD_CdInt(
uint32_t           card_no,
int32_t            enable,
sdc_sd_status_t    (*callback)(int32_t)
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

enable　：SD カード挿抜割り込みの禁止／許可設定

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

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

(*callback)(int32_t)　：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、SD 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 SDHI_CD 端子の検出状態が格納されます。

0 : SDC_SD_CD_INSERT（SDHI_CD 端子の立ち下りを検出）

1 : SDC_SD_CD_REMOVE（SDHI_CD 端子の立ち上がりを検出）

Return Values

SDC_SD_SUCCESS　　正常終了

SDC_SD_ERR　　一般エラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD 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、SD カード挿抜割り込み発生時にコールされます。

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

Example

```
/* Callback function */
sdc_sd_status_t r_sdc_sd_cd_callback(int32_t cd)
{
   if(cd & SDC_SD_CD_INSERT)
   {
      /* sdcard in */
   }
   else
   {
      /* sdcard out */
   }
   return SDC_SD_SUCCESS;
}

/* main */
void main(void)
{
            if (R_SDC_SD_CdInt(SDC_SD_CARD0, SDC_SD_CD_INT_ENABLE,
r_sdc_sd_cd_callback) != SDC_SD_SUCCESS)
            {
                  /* Error */
            }
```

Special Notes

カード検出を有効にするためには SDHI FIT モジュールの#define SDHI_CFG_CHx_CD_ACTIVE を”1”に設定してください。

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理が必要です。

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

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

3.17　R_SDC_SD_IntCallback()

SD 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

```
sdc_sd_status_t R_SDC_SD_IntCallback(
uint32_t           card_no,
sdc_sd_status_t    (*callback)(int32_t)
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

(*callback)(int32_t) ：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_SDC_SD_Initialize()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

SDC_SD_SUCCESS	正常終了
SDC_SD_ERR	一般エラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD 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、SD のプロトコルステータス変化による割り込み発生時にコールされます。

Example

```
/* Callback function */
sdc_sd_status_t r_sdc_sd_callback(int32_t channel)
{
            /* User program */

            return SDC_SD_SUCCESS;
}

if (R_SDC_SD_IntCallback(SDC_SD_CARD0, r_sdc_sd_callback) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理が必要です。

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、SD 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、SD カード挿抜割り込み発生時にはコールされません。

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

3.18　R_SDC_SD_GetErrCode()

ドライバのエラーコードを取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_GetErrCode(
uint32_t    card_no
)
```

Parameters

card_no

SD カード番号	使用する SD カード番号（0 起算）

Return Values

エラーコード	エラーコードを参照

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

R_SDC_SD_Initialize()関数／R_SDC_SDMEM_Read()関数／R_SDC_SDMEM_Write()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_SDC_SDMEM_ReadSoftwareTrans()関数／R_SDC_SDMEM_ReadSoftwareTransSingleCmd()関数／R_SDC_SDMEM_WriteSoftwareTrans()関数も同様

Example

```
sdc_sd_cfg_t                    sdc_sd_config;
sdc_sd_status_t         error_code = SDC_SD_SUCCESS;

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

sdc_sd_config.mode = SDC_SD_CFG_DRIVER_MODE;
sdc_sd_config.voltage = SDC_SD_VOLT_3_3;
if (R_SDC_SD_Initialize(SDC_SD_CARD0, &sdc_sd_config, SDC_SD_MODE_MEM) !=
SDC_SD_SUCCESS)
{
            /* Error */
            error_code = R_SDC_SD_GetErrCode(SDC_SD_CARD0);
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理が必要です。

アプリケーションプログラムで SD カード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。

3.19　R_SDC_SD_GetBuffRegAddress()

SD バッファレジスタのアドレスを取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SD_GetBuffRegAddress(
uint32_t    card_no,
uint32_t    *p_reg_buff
)
```

Parameters

channel

　SD カード番号　　使用する SD カード番号（0 起算）

*p_reg_buff

　SD バッファレジスタアドレスポインタ

Return Values

SDC_SD_SUCCESS	正常終了
SDC_SD_ERR	一般エラー

Properties

r_sdc_sd_rx_if.h にプロトタイプ宣言されています。

Description

SD バッファレジスタ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_SDC_SD_GetBuffRegAddress(SDC_SD_CARD0, &reg_buff) != SDC_SD_SUCCESS)
{
            /* Error */
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理が必要です。

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

3.20　R_SDC_SD_1msInterval()

SD カード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_SDC_SD_1msInterval(
void
)
```

Parameters

なし

Return Values

なし

Properties

r_sdc_sd_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_SDC_SD_1msInterval();
            }
}

main()
{
   /* 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_sdc_sd_config.c の r_sdc_sd_int_wait()および r_sdc_sd_wait()を OS 処理に置き換える場合には不要です。

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

3.21　R_SDC_SD_SetDmacDtcTransFlg()

DMAC/DTC 転送完了フラグをセットする関数です。

Format

```
sdc_sd_status_t R_SDC_SD_SetDmacDtcTransFlg(
uint32_t    card_no,
uint32_t    flg
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

flg

DMAC/DTC 転送完了フラグ　　SDC_SD_SET_TRANS_STOP

Return Values

SDC_SD_SUCCESS	正常終了
SDC_SD_ERR	一般エラー（チャネル異常）

Properties

r_sdc_sd_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 3-10 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、SDC_SD_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、SDHI SBFAI 割り込みハンドラ内で SDC_SD_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 SDC_SD_SET_TRANS_STOP をセットし、本関数をコールしてください。

表 3-10　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_SDC_SD_SetDmacDtcTransFlg()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_SDC_SD_SetDmacDtcTransFlg()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)
{
   volatile dmaca_return_t   ret_dmaca;
   dmaca_stat_t            p_stat_dmaca;

   /* check DMA end */
   /*** DMACA transfer end check ***/
   ret_dmaca = R_DMACA_Control(DMACA_CH0, DMACA_CMD_STATUS_GET,
(dmaca_stat_t*)&p_stat_dmaca);
   if (DMACA_SUCCESS != ret_dmaca)
   {
      return;
   }

   if (true == (p_stat_dmaca.dtif_stat))
   {
      ret_dmaca = R_DMACA_Control(DMACA_CH0, DMACA_CMD_DTIF_STATUS_CLR,
(dmaca_stat_t*)&p_stat_dmaca);
      R_SDC_SD_SetDmacDtcTransFlg(SDC_SD_CARD0, SDC_SD_SET_TRANS_STOP);
   }

   if (true == (p_stat_dmaca.esif_stat))
   {
      ret_dmaca = R_DMACA_Control(DMACA_CH0, DMACA_CMD_ESIF_STATUS_CLR,
(dmaca_stat_t*)&p_stat_dmaca);
   }

   return;
}

＜転送エラー終了時＞
#define TEST_BLOCK_CNT  (4)
#define BLOCK_NUM             (512)

sdc_sd_access_t         sdc_sd_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);
sdc_sd_access.p_buff    = (uint8_t *)&g_test_r_buff[0];
sdc_sd_access.lbn       = 0x10000000;
sdc_sd_access.cnt       = TEST_BLOCK_CNT;
sdc_sd_access.rw_mode   = SDC_SD_PRE_DEF;

if(R_SDC_SDMEM_Read(SDC_SD_CARD0, &sdc_sd_access) != SDC_SD_SUCCESS)
{
           /* Error */
           R_SDC_SD_SetDmacDtcTransFlg(SDC_SD_CARD0, SDC_SD_SET_TRANS_STOP);
}
```

Special Notes

本関数実行前に R_SDC_SD_Open()関数によるドライバのオープン処理と R_SDC_SD_Initialize()関数による初期化処理が必要です。

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

3.22　R_SDC_SD_SetLogHdlAddress()

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

```
sdc_sd_status_t R_SDC_SD_SetLogHdlAddress(
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

SDC_SD_SUCCESS	正常終了

Properties

r_sdc_sd_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 SD カードドライバに設定します。

Example

```
#define SDC_SD_USER_LONGQ_MAX                   (8)
#define SDC_SD_USER_LONGQ_BUFSIZE               (SDC_SD_USER_LONGQ_MAX * 4)
#define SDC_SD_USER_LONGQ_IGN_OVERFLOW    (1)

uint32_t
      g_sdc_sd_user_longq_buf[SDC_SD_USER_LONGQ_BUFSIZE];
static longq_hdl_t            p_sdc_sd_user_long_que;
longq_err_t                   err = LONGQ_SUCCESS;
uint32_t                            user_long_que = 0;

err = R_LONGQ_Open(g_sdc_sd_user_longq_buf,
                                        SDC_SD_USER_LONGQ_BUFSIZE,
                                        SDC_SD_USER_LONGQ_IGN_OVERFLOW,
                                        &p_sdc_sd_user_long_que);
if (LONGQ_SUCCESS != err)
{
            /* Error */
}
user_long_que = (uint32_t)p_sdc_sd_user_long_que;
if (R_SDC_SD_SetLogHdlAddress(user_long_que) != SDC_SD_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_SDC_SD_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_SDC_SD_GetErrCode()関数によるエラーコード取得はできません。

3.23　R_SDC_SD_Log()

エラーログを取得する関数です。

Format

```
uint32_t R_SDC_SD_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_sdc_sd_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)

sdc_sd_cfg_t                  sdc_sd_config;

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

sdc_sd_config.mode = SDC_SD_CFG_DRIVER_MODE;
sdc_sd_config.voltage = SDC_SD_VOLT_3_3;
if (R_SDC_SD_Initialize(SDC_SD_CARD0, &sdc_sd_config, SDC_SD_MODE_MEM) !=
SDC_SD_SUCCESS)
{
            /* Error */
            R_SDC_SD_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_SDC_SD_GetErrCode()関数によるエラーコード取得はできません。

3.24　R_SDC_SD_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_SDC_SD_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_sdc_sd_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_SDC_SD_GetVersion();
```

Special Notes

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

4.　端子設定

下位層の SDHI 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。

e2 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 1。

端子設定の制御手順は「4.1 SD カードの挿入と電源投入タイミング」と「4.2 SD 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。

[1] 端子設定機能の詳細は、RX ファミリ SDHI モジュール Firmware Integration Technology (R01AN3852JJ) を参照してください。

4.1　SD 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 4-1 に示します。SD カードの挿入は、R_SDC_SD_Open()関数の正常終了後、SD カードへの電源電圧供給停止状態、かつ SDHI 出力端子を L 出力状態で行ってください。

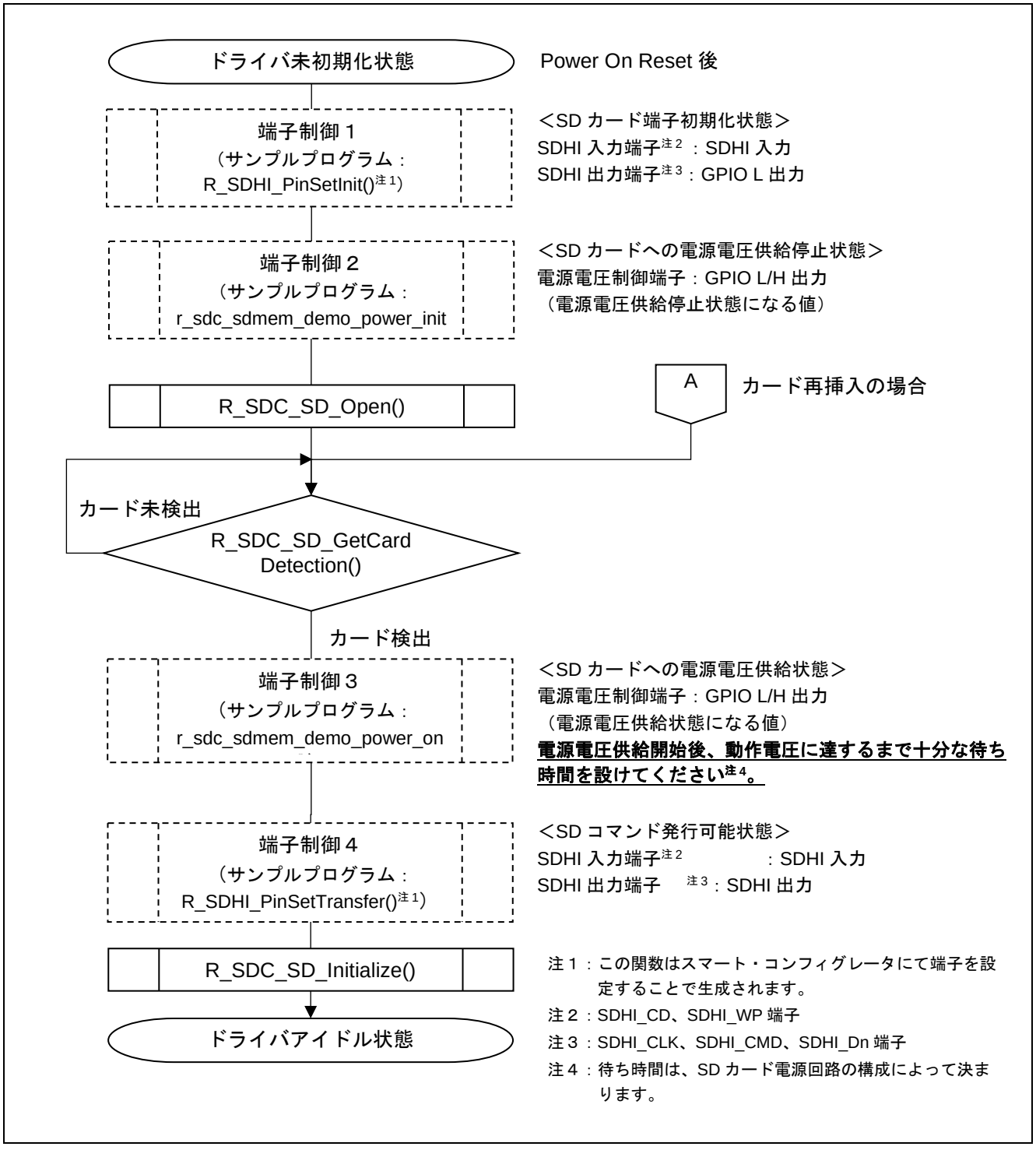


図 4-1　SD カードの挿入と電源投入タイミング

表 4-1　SD 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	SDHI 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：SDHI PMR 設定：周辺モジュール	SDHI 入力 （SD カード検出可能状態）
	SDHI 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	SDHI 入力端子 注 1	MPC 設定：SDHI PMR 設定：周辺モジュール	SDHI 入力
	SDHI 出力端子 注 2	MPC 設定：SDHI PMR 設定：周辺モジュール	SDHI 出力 （SD コマンド発行可能状態）

注 1：SDHI_CD、SDHI_WP 端子
注 2：SDHI_CLK、SDHI_CMD、SDHI_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。

4.2　SD 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 4-2 に示します。SD カードの抜去は、ドライバアイドル状態での R_SDC_SD_End()関数の正常終了後、SD 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 SD 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。

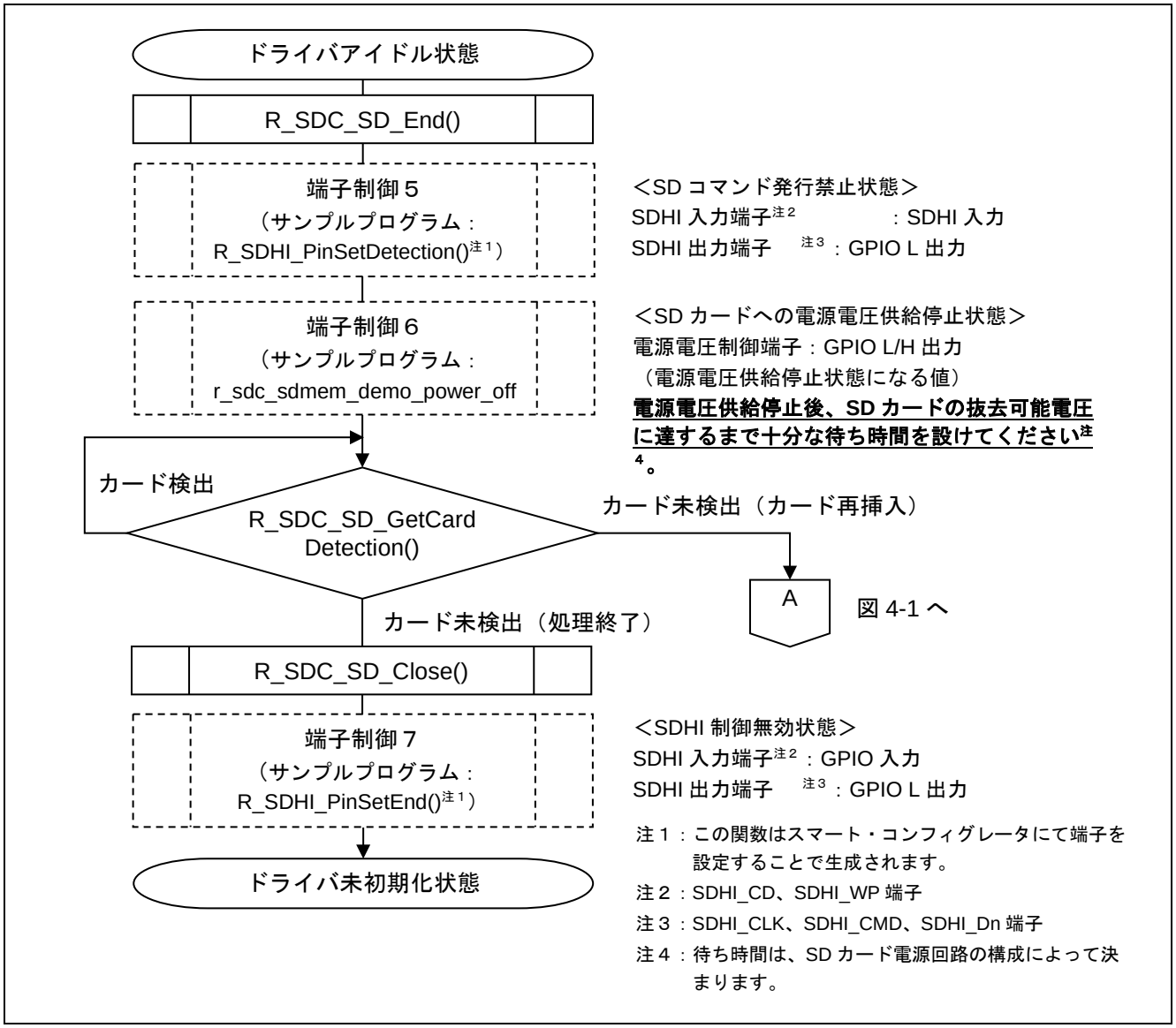


図 4-2　SD カードの抜去と電源停止タイミング

表 4-2　SD 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	SDHI 入力端子　注 1	MPC 設定：SDHI PMR 設定：周辺モジュール	SDHI 入力
	SDHI 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	SDHI 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
	SDHI 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：SDHI_CD、SDHI_WP 端子
注 2：SDHI_CLK、SDHI_CMD、SDHI_Dn 端子

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、「4.1 SD カードの挿入と電源投入タイミング」、「4.2 SD カードの抜去と電源停止タイミング」、SD カードへの読み出し／書き込みの処理を行います。

5.2　状態遷移図

図 5-1 に状態遷移図を示します。

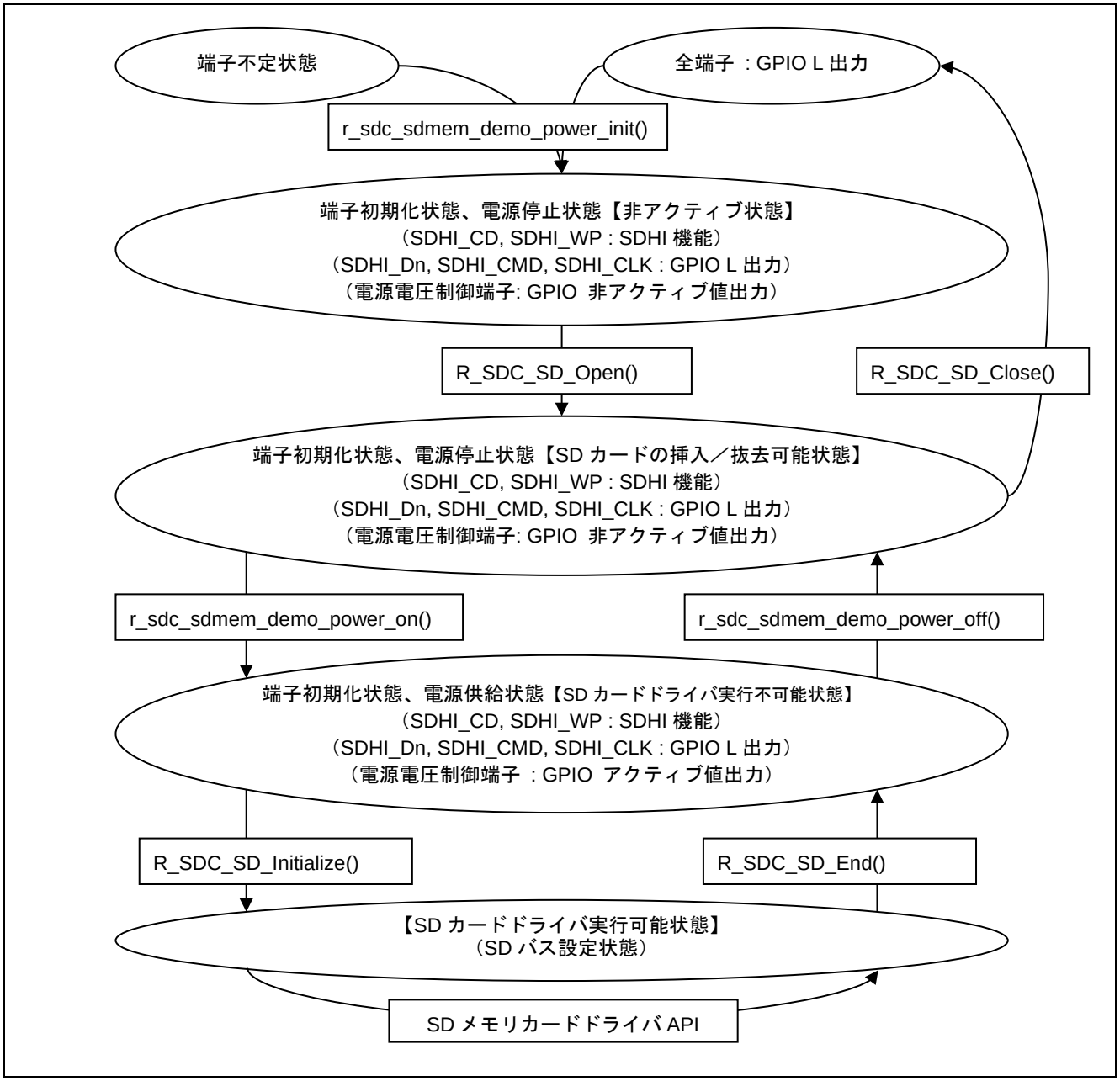


図 5-1　状態遷移図

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グレーションオプションの設定は、r_sdc_sdmem_rx_demo_pin_config.h で行います。

Configuration options in *r_sdc_sdmem_rx_demo_pin_config.h*	
#define SDC_SD_CFG_POWER_PORT_NONE ※デフォルト値は“無効”	SD カードを使用する場合の定義です。 SD 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。 SD 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 SDC_SD_CFG_POWER_HIGH_ACTIVE (1) ※デフォルト値は“1（High を供給）”	SD カードを使用し、かつ、SD 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、SD カード電源回路を有効にするために、SD 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、SD カード電源回路を有効にするために、SD 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 SDC_SD_CFG_POWER_ON_WAIT (100) ※デフォルト値は“100（100ms ウェイト）”	SD カードを使用する場合の定義です。 SD 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 SDC_SD_CFG_POWER_OFF_WAIT (100) ※デフォルト値は“100（100ms ウェイト）”	SD カードを使用する場合の定義です。 SD カード用電源回路に電源供給停止後、SD カード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 R_SDC_SD_CFG_POWER_CARDx_PORT ※CARDx の“x”は SD カード番号（x=0）	SD カード番号 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 R_SDC_SD_CFG_POWER_CARDx_BIT ※CARDx の“x”は SD カード番号（x=0）	SD カード番号 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_sdc_sdmem_demo_power_init()	電源電圧制御端子設定の初期化処理
r_sdc_sdmem_demo_power_on()	電源電圧の供給開始処理
r_sdc_sdmem_demo_power_off()	電源電圧の供給停止処理
r_sdc_sdmem_demo_softwaredelay()	時間待ち処理

(1)　r_sdc_sdmem_demo_power_init()

SD メモリカードドライバで使用する SD 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

```
sdc_sd_status_t r_sdc_sdmem_demo_power_init(
uint32_t card_no
)
```

Parameters

card_no

SD カード番号	使用する SD カード番号（0 起算）

Return Values

SDC_SD_SUCCESS	正常終了

Description

SD 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。

(2)　r_sdc_sdmem_demo_power_on()

SD 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

```
sdc_sd_status_t r_sdc_sdmem_demo_power_on(
uint32_t card_no
)
```

Parameters

card_no

SD カード番号	使用する SD カード番号（0 起算）

Return Values

SDC_SD_SUCCESS	正常終了
SDC_SD_ERR	一般エラー

Description

SD 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_sdc_sdmem_rx_demo_pin_config.h の SDC_SD_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_sdc_sdmem_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「SDC_SD_CFG_POWER_ON_WAIT」で設定してください。

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

(3)　r_sdc_sdmem_demo_power_off()

SD 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

```
sdc_sd_status_t r_sdc_sdmem_demo_power_off(
uint32_t card_no
)
```

Parameters

card_no

SD カード番号	使用する SD カード番号（0 起算）

Return Values

SDC_SD_SUCCESS	正常終了
SDC_SD_ERR	一般エラー

Description

SD 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_sdc_sdmem_rx_demo_pin_config.h の SDC_SD_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_sdc_sdmem_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「SDC_SD_CFG_POWER_OFF_WAIT」で設定してください。

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

(4)　r_sdc_sdmem_demo_softwaredelay()

時間待ちを行う際に使用する関数です。

Format

```
bool r_sdc_sdmem_demo_softwaredelay(
uint32_t delay,
sdc_sd_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：SDC_SD_DELAY_MICROSECS

ミリ秒：SDC_SD_DELAY_MILLISECS

秒：SDC_SD_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：μITRON 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
SD カード電源 On 時の電圧安定待ち時間	SD 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 SDC_SD_CFG_POWER_ON_WAIT で変更可能。
SD カード電源 Off 時の電圧安定待ち時	SD カード用電源回路に電源供給停止後、SD 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 SDC_SD_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_sdc_sdmem_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。

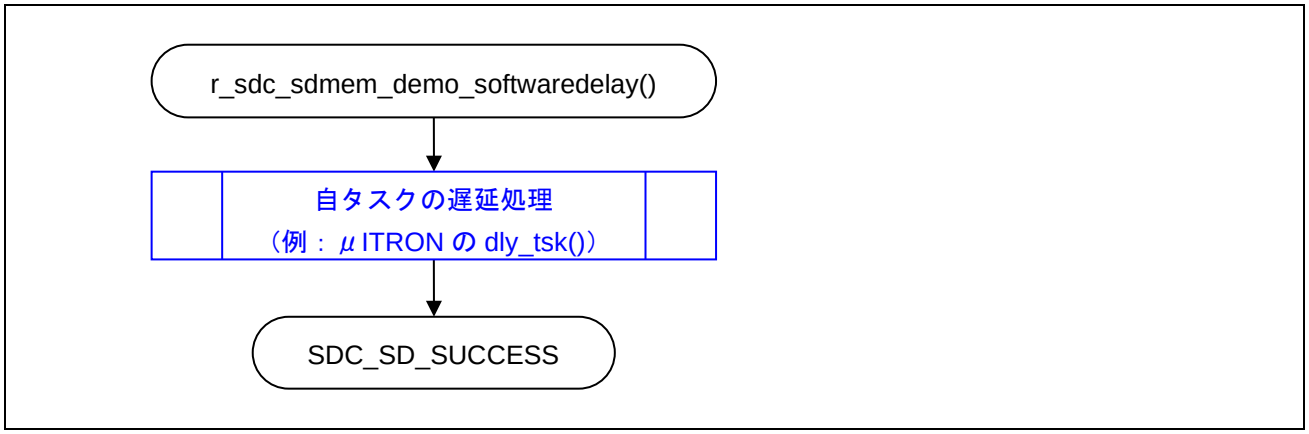


図 5-2　OS の自タスク遅延処理を使った時間待ち例

5.6　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

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 V6.3.0
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 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.2.02
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 RX231 (型名：R0K505231SxxxBE) 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 (Rev. 3.00)

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 V7.7.0 IAR Embedded Workbench for Renesas RX 4.13.1
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 V3.02.00 コンパイルオプション：統合開発環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 8.3.0.201904 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 4.13.1 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.3.00
使用ボード	Renesas Starter Kit for RX72M (型名：RTK5572Mxxxxxxxxxx)

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_sdc_sd_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　SD メモリ：SDXC カードの Default-Speed モード時の消費電力設定（ACMD41 発行時の XPS 設定）について

SD メモリカードドライバは、SDXC カードの初期化処理時にコマンドの引数 XPC=0 を設定し ACMD41 発行します。これにより、外部電源回路の VDD　電源供給能力に関わらず、SDSC カード、SDHC カードと同様に SDXC カードも Default-Speed モードで最大消費電力 0.36W（最大消費電流 100mA、3.6V）の動作となります。

6.4　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 6-3　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_sdc_sd_int_wait()	ステータス割り込み待ち処理
r_sdc_sd_int_mem_wait()	ステータス割り込み待ち処理（SD メモリ制御）
r_sdc_sd_int_err_mem_wait()	ステータス割り込み待ち処理（SD メモリエラー制御）
r_sdc_sd_wait()	時間待ち処理

(1)　r_sdc_sd_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

```
sdc_sd_status_t r_sdc_sd_int_wait(
uint32_t card_no,
int32_t time
)
```

Parameters

card_no

　SD カード番号　　使用する SD カード番号（0 起算）

time

　タイムアウト時間（単位：ミリ秒）

Return Values

SDC_SD_SUCCESS	正常終了（割り込み要求発生）
SDC_SD_ERR	一般エラー

Description

SD カード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、SDC_SD_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、SDC_SD_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 SD ステータスレジスタ 1,2 取得処理（r_sdc_sd_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。

注 1：　r_sdc_sd_int_mem_wait()関数／r_sdc_sd_int_err_mem_wait()関数も同様です。

Special Notes:

SD カード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：μITRON の dly_tsk()）を使用した場合の使用例です。但し、OS 処理は r_sdc_sd_int_wait()関数にユーザ独自で組み込んでください。

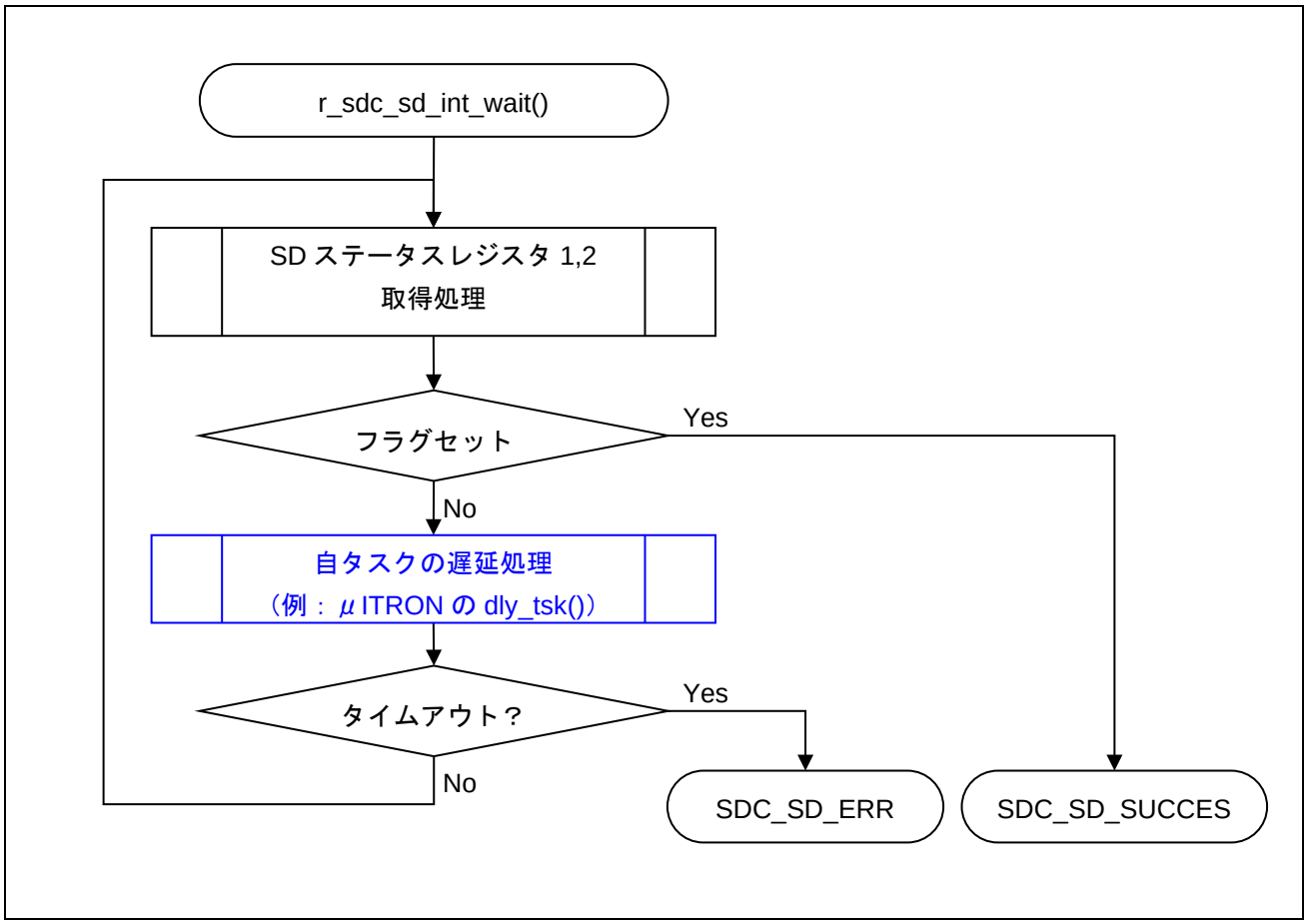


図 6-1　OS の自タスク遅延処理を使った SD プロトコルステータス確認例

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_sdc_sd_int_wait()関数の SD ステータスレジスタ 1,2 取得処理（r_sdc_sd_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、SD 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。

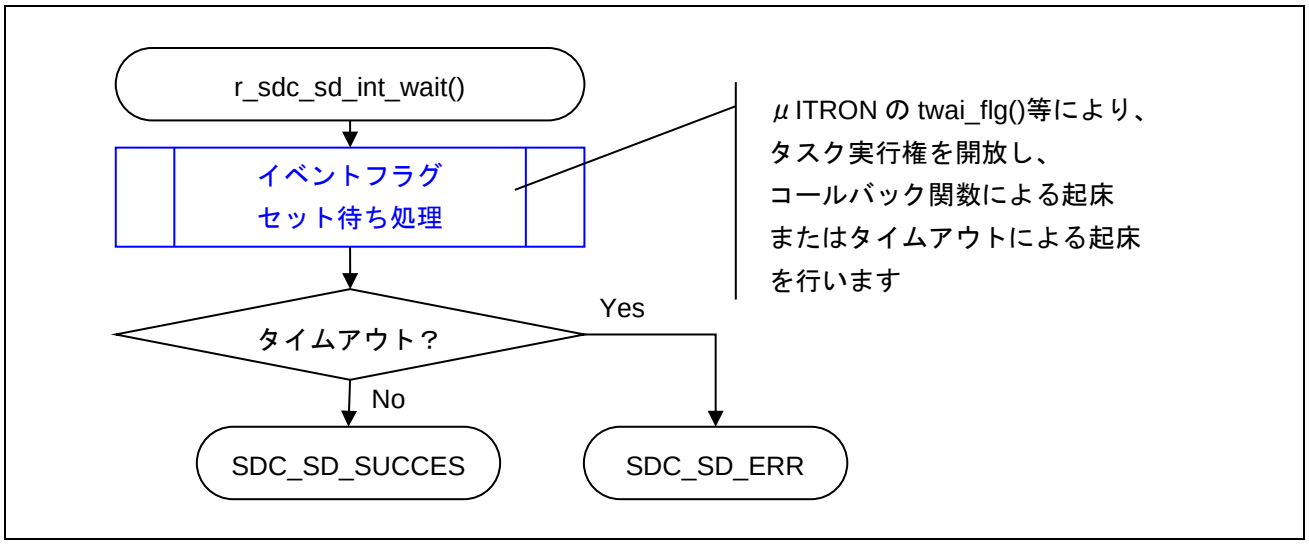


図 6-2　OS のウェイトタスク処理を使った SD プロトコルステータス確認例

(2)　r_sdc_sd_wait()

時間待ちを行う際に使用する関数です。

Format

```
sdc_sd_status_t r_sdc_sd_wait(
uint32_t card_no,
int32_t time
)
```

Parameters

card_no

　SD カード番号　　使用する SD カード番号（0 起算）

time

　タイムアウト時間（単位：ミリ秒）

Return Values

SDC_SD_SUCCESS	正常終了（割り込み要求発生）
SDC_SD_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 SDC_SD_SUCCESS を返します。

Special Notes:

表 6-5 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：μITRON の dly_tsk()）等に置き換えることが可能です。

表 6-4　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
SD カードの初期化処理時の 74 クロック発生	Card identification mode：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）
SD カードの初期化処理時の Ready 状態遷移検出	Card identification mode：Ready 状態への遷移待ち＜5ms＞（最大 1 秒） SD メモリの場合、5ms 間隔で ACMD41 を発行し、最大 200 回繰り返す。
SD メモリへの CMD9 発行処理後の時間待ち	Data transfer mode：SD メモリへの CMD9 発行処理後の応答時間待ち<3ms> SD メモリに対して CMD9 発行後の時間待ち （最大 3ms、最小 2ms を確保）

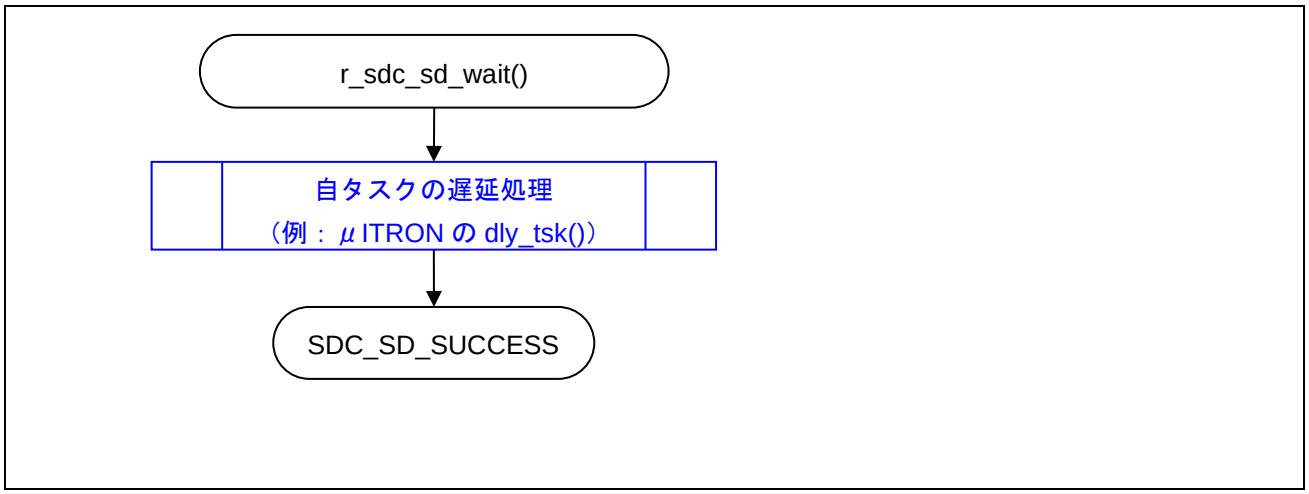


図 6-3　OS の自タスク遅延処理を使った時間待ち例

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

本モジュールは以下のテクニカルアップデートの内容を反映しています。

- TN-RX*-A195A/J
- TN-RX*-A196A/J
- TN-RX*-A197A/J

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
2.01	2018.02.28	-	初版発行 SD モード SD メモリカードドライバ・ソフトウェア RTM0RX0000DSDD0 Ver.2.00 ユーザーズマニュアル （R01UW0135）を本アプリケーションノートに変更した。
2.02	2018.06.29	-	SD Specifications Part 1 Physical Layer Simplified Specification に対応した。
3.00	2020.02.10	-	以下のコンパイラに対応 ・GCC for Renesas RX ・IAR C/C++ Compiler for Renesas RX API 関数の説明から「Reentrant」を削除 「r_sdhi_rx」との型の不整合を修正 「WAIT_LOOP」をコードに追加

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。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)

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。