User's Manual

32

SD モード SDIO ドライバ・ソフトウェア

RTM0RX0000DSDD3
Ver.2.00

ユーザーズマニュアル　ミドルウェア編

ルネサス 32 ビットマイクロコンピュータ
RX ファミリ

ルネサス エレクトロニクス
www.renesas.com

Rev.2.00　2017.07

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、その他の不適切に使用しないでください。かかる改造、改変、複製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、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、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できません。たとえ、意図し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。また、当社製品および技術を、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。

   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. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
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. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

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その総株主の議決権の過半数を直接または間接に保有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。

製品ご使用上の注意事項

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

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部 ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

このマニュアルの使い方

1.　目的と対象者

本マニュアルは、RXファミリ用SDモードSDIOドライバ・ソフトウェアに含まれるソフトウェアの構成、使用方法、組み込み手順について記述しています。

2.　数や記号の表記

本マニュアルは、特に説明のない限り、以下の表記規則に示す数字／信号の表記、および用語を使用して説明しています。

数字の表記：10進数はXXXX、16進数は0xXXもしくはXXXXh

目次

1.　概要

1.1　はじめに

本製品はルネサス エレクトロニクス製 RX ファミリ MCU 内蔵 SD ホストインタフェース（以下、SDHI と略す。）を使用した SDIO を SD モードで制御するデバイスドライバ（以下、SDIO ドライバもしくは SD カードドライバと略す。）です。本製品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、SDIO モジュールの制御が可能になります。

本マニュアルは、SDIO ドライバを使用して、アプリケーションを作成するための情報を提供することを目的としています。

なお、本製品は SDIO モジュールを対象としていますが、SD カードの仕様書に準拠したものであるため、本マニュアル上「SD カード」もしくは「SDIO カード」と記述しています。本マニュアル参照時には、「SD カード」もしくは「SDIO カード」を「SDIO モジュール」に置き換えてください。

1.2　機能概要

以下に機能を示します。

表 1-1　SDHI 機能一覧

項目	機能
準拠した規格	SD Specifications Part E1 SDIO Specification Ver. 3.00 準拠
SDHI 制御ドライバ	SDIO コマンド（CMD52, CMD53）のリード／ライト制御をサポート
動作対象 SD カード	株式会社村田製作所製 SDIO 無線 LAN モジュール（品名：LBWA17DZX6）
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（1 ビット／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 と High-Speed mode をサポート（注 1） SD カードドライバが Speed mode を判別し、初期化実行
SD カード検出機能	CD 端子による検出のみをサポート

注 1：使用するマイコンの仕様に依存します。

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	リトルエンディアン対応
その他の機能	Firmware Integration Technology（FIT）に対応

1.3　アプリケーション構成図

SDIO ドライバを使用して Wireless LAN 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。

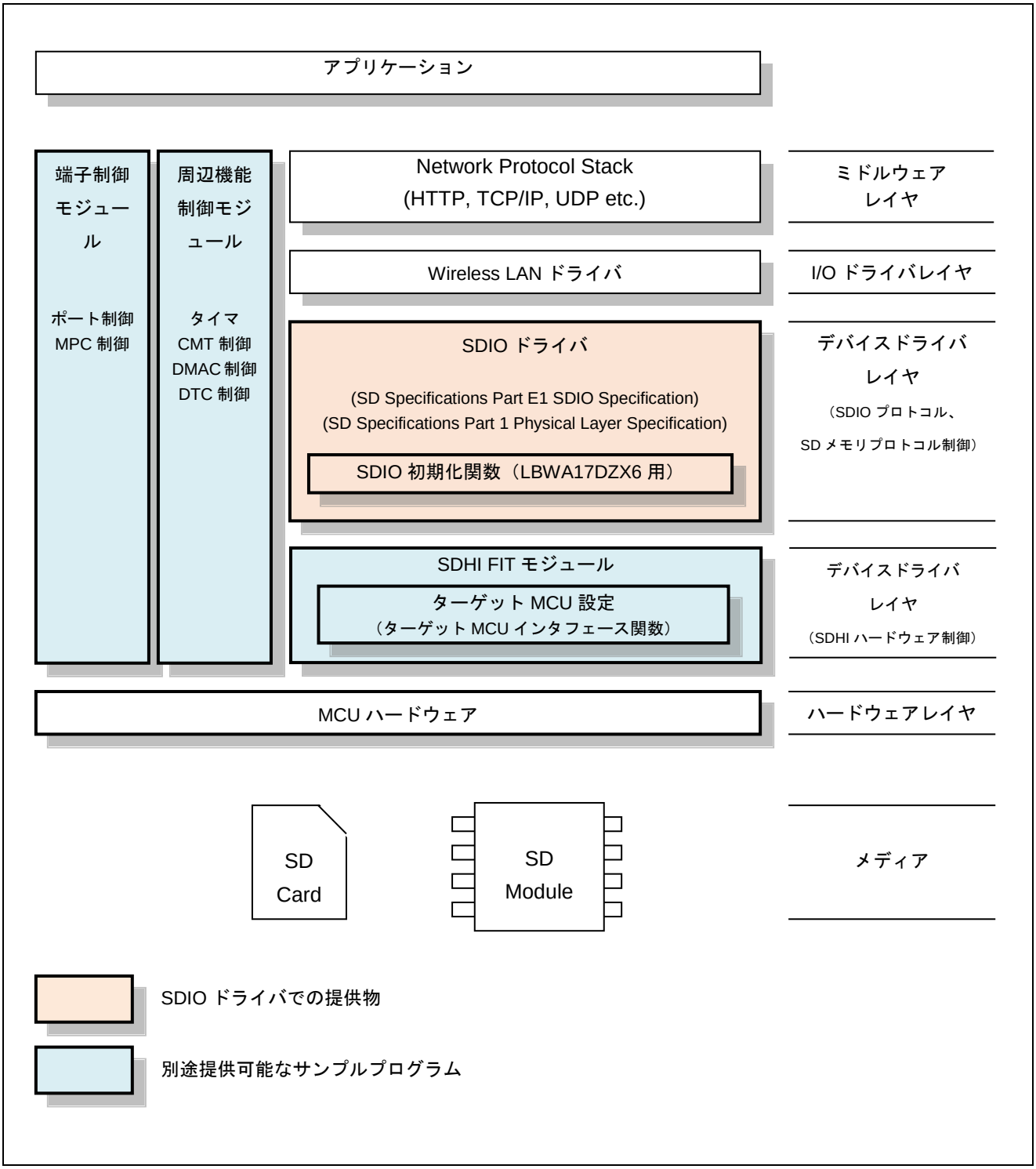


図 1-1　アプリケーション構成図（Wireless LAN の例）

(a) Network Protocol Stack

通信を実現するための通信プロトコル群のソフトウェアです。

(b) Wireless LAN ドライバ

Wireless LAN 制御用のネットワークドライバです。

(c) SDIO ドライバ

SD Specifications Part E1 SDIO Specification の SDIO プロトコル制御と SD Specifications Part 1 Physical Layer Specification の SD メモリプロトコル制御と SDHI Low Level アクセス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。
デバッグ用モジュールについては、「6.7 デバッグ用モジュールについて」を参照してください。
以下に、モジュールとライブラリファイル名を示します。

表 1-3 モジュールとライブラリファイル名

モジュール	ライブラリファイル名（Lib ファイル名）
製品組み込み用モジュール（リトルエンディアン）	r_sdc_sdio3_rx_little.lib
製品組み込み用モジュール（ビッグエンディアン）	サポート対象外
デバッグ用モジュール（リトルエンディアン）	r_sdc_sdio3_rx_little_eva.lib
デバッグ用モジュール（ビッグエンディアン）	サポート対象外

(d) 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。

(e) 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。「1.10 関連アプリケーションノート」を参照し、入手してください。

(f) 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 モジュール電源用ポート制御と SD モジュール専用ポート制御）、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 マイコンの 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。

1.4 動作環境

動作環境を以下に示します。

表 1-4 Ver.2.00 動作環境

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e^2 studio V6.0.0
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 V.2.07.00
	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
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.2.00
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)

1.5 Lib ファイル作成時の C コンパイラ

Lib ファイル作成時のコンパイラパッケージのバージョン、およびコンパイラオプションは、以下のとおりです。

C コンパイラのバージョンは「C/C++ compiler for RX family V.2.01.00」です。

統合開発環境 e^2 studio のバージョンは「V5.4.0.015」です。

(1) リトルエンディアン

表 1-5 コンパイル環境

項目	オプション
Compiler	-define=__RX –nomessage -nologo -isa=rxv2 -fpu -alias=noansi
Assembler	-nolistfile -nologo -isa=rxv2 -fpu
Standard Library	-lang=c99
Converter	–nomessage

(2) ビッグエンディアン

サポート対象外。

1.6　推奨 C コンパイラ、アセンブラ

「1.4 動作環境」で示すコンパイラパッケージのバージョン、もしくはそれ以降のものを使用ください。他の環境では、コンパイルエラーが発生する場合があります。

1.7　メモリサイズの参考値

メモリサイズの参考値を以下に示します。なお、コンパイル条件は、「1.4 動作環境」を参照してください。

表 1-6　メモリサイズ

使用メモリ	サイズ	備考
ROM　注１	ライブラリ＋ターゲット MCU： 17.1K　バイト	・対象 ¥r_sdc_sdio#_rx ディレクトリ配下のファイル （#は、英数字を示します。） ・Software 転送設定時
RAM　注１、注２	ライブラリ：128　バイト ワークバッファ（注３）： 288　バイト	
最大使用ユーザスタック	384　バイト	
最大使用割り込みスタック	56　バイト	

注１：サイズは、データ転送方式等の設定に依存します。

注２：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３：ドライバのオープン処理関数を参照してください。

1.8　サポートコマンドについて

SDIO 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 SD カードの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 SDIO ドライバのサポート状況を示したものです。SD カードの仕様書欄の数値は、コマンドをサポートしたバージョン（ただし、3.00 以降からのバージョン）を示します。

表 1-7　SDIO ドライバのサポートコマンド一覧（ー：記載なし、〇：サポート、×：未サポート）

コマンド	Part 1 Physical Layer	Part E1 SDIO	マイコンのサポート範囲 RX64M RX71M RX231	本製品（SDIO ドライバ）	備考
CMD0	3.00	3.00	〇	×	未使用
CMD1	Reserved	ー	ー	×	ー
CMD2	3.00	ー	〇	×	未使用
CMD3	3.00	ー	〇	〇	SDIO 初期化で使用
CMD4	3.00	ー	〇	×	未使用
CMD5	Reserved	3.00	〇	〇	SDIO 初期化で使用
CMD6	3.00	ー	〇	×	未使用
CMD7	3.00	ー	〇	〇	SDIO 初期化で使用
CMD8	3.00	ー	〇	×	未使用
CMD9	3.00	ー	〇	×	未使用
CMD10	3.00	ー	〇	×	未使用
CMD11	3.00	3.00	〇	×	未使用
CMD12	3.00	ー	〇	×	未使用
CMD13	3.00	ー	〇	×	未使用
CMD14	Reserved	ー	ー	×	ー
CMD15	3.00	ー	〇	×	未使用
CMD16	3.00	ー	〇	×	未使用
CMD17	3.00	ー	〇	×	未使用
CMD18	3.00	ー	〇	×	未使用
CMD19	3.00	3.00	ー	×	ー
CMD20	3.00	ー	〇	×	未使用
CMD21	Reserved	ー	ー	×	ー
CMD22	Reserved	ー	ー	×	ー
CMD23	3.00	ー	ー	×	ー
CMD24	3.00	ー	〇	×	未使用
CMD25	3.00	ー	〇	×	未使用
CMD26	Reserved	ー	ー	×	ー
CMD27	3.00	ー	〇	×	未使用
CMD28	3.00	ー	〇	×	未使用
CMD29	3.00	ー	〇	×	未使用
CMD30	3.00	ー	〇	×	未使用
CMD31	Reserved	ー	ー	ー	ー
CMD32	3.00	ー	〇	×	未使用
CMD33	3.00	ー	〇	×	未使用

CMD34-37	3.00	ー	ー	×	ー
CMD38	3.00	ー	○	×	未使用
CMD39	Reserved	ー	ー	ー	ー
CMD40	4.10	ー	ー	×	ー
CMD41	Reserved	ー	ー	ー	ー
CMD42	3.00	ー	○	×	未使用
CMD43-47	Reserved	ー	ー	ー	ー
CMD48	4.10	ー	ー	×	ー
CMD49	4.10	ー	ー	×	ー
CMD50	3.00	ー	ー	×	ー
CMD51	Reserved	ー	ー	ー	ー
CMD52	Reserved	3.00	○	○	SDIO レジスタへのリード／ライト処理で使用
CMD53	Reserved	3.00	○	○	
CMD54	Reserved	ー	ー	ー	ー
CMD55	3.00	ー	○	×	未使用
CMD56	3.00	ー	○	×	未使用
CMD57	3.00	ー	ー	×	ー
CMD58	4.10	ー	ー	×	ー
CMD59	4.10	ー	ー	×	ー
CMD60-63	Reserved	ー	ー	ー	ー
ACMD1-5	Reserved	ー	ー	ー	ー
ACMD6	3.00	ー	○	×	未使用
ACMD7-12	Reserved	ー	ー	ー	ー
ACMD13	3.00	ー	○	×	未使用
ACMD14-16	Reserved	ー	ー	ー	ー
ACMD17	Reserved	ー	ー	ー	ー
ACMD18	Reserved	ー	ー	ー	ー
ACMD19-21	Reserved	ー	ー	ー	ー
ACMD22	3.00	ー	○	×	未使用
ACMD23	3.00	ー	○	×	未使用
ACMD24	Reserved	ー	ー	ー	ー
ACMD25-26	Reserved	ー	ー	ー	ー
ACMD27-28	Reserved	ー	ー	ー	ー
ACMD29	Reserved	ー	ー	ー	ー
ACMD30-35	Reserved	ー	ー	ー	ー
ACMD36-37	Reserved	ー	ー	ー	ー
ACMD38	Reserved	ー	ー	ー	ー
ACMD39-40	Reserved	ー	ー	ー	ー
ACMD41	3.00	ー	○	×	未使用
ACMD42	3.00	ー	○	×	未使用
ACMD43-49	Reserved	ー	ー	ー	ー
ACMD50	ー	ー	ー	ー	ー
ACMD51	3.00	ー	○	×	未使用
ACMD52-54	Reserved	ー	ー	ー	ー
ACMD55	No exist	ー	ー	ー	ー
ACMD56-59	Reserved	ー	ー	ー	ー

注　：マイコンのサポート範囲欄が“ー”のコマンドに対しては、SD カードドライバが持つコマンド／レスポンスタイプを使用できる場合、サポートコマンドの拡張が可能です。

1.9 関連ソフトウェア

SDIO ドライバは、Firmware Integration Technology（以下、FIT と略す）を使った FIT モジュールです。以下、同様に FIT を使った他の機能制御モジュールを FIT モジュールもしくは“機能名”FIT モジュールと表します。SDIO ドライバは、他の FIT モジュールと組み合わせることにより、組み込みが容易になります。

SDIO ドライバは、以下の周辺機能を利用した FIT モジュールと組み合わせて制御可能です。

- DMA コントローラ（DMAC）
  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）
- データトランスファコントローラ（DTC）
  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）
- タイマ
  r_cmt_rx（コンペアマッチタイマ CMT FIT モジュール）：他タイマやソフトウェアタイマで代用可
  SD プロトコル・タイムアウト、DMAC 転送もしくは DTC 転送のタイアウト等、時間待ち処理に使用
- ロングキュー（LONGQ）－ソフトウェアモジュール
  エラーログ取得機能で使用する FIT モジュールです。

1.10 関連アプリケーションノート

本モジュールに関連するアプリケーションノートを以下に示します。合わせて参照してください。

- Firmware Integration Technology ユーザーズマニュアル (R01AN1833JU)
- ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685JJ)
- e² studio に組み込む方法 Firmware Integration Technology (R01AN1723JU)
- CS+に組み込む方法 Firmware Integration Technology (R01AN1826JJ)
- RX ファミリ DMA コントローラ DMACA 制御モジュール Firmware Integration Technology (R01AN2063JJ)
- RX Family DTC モジュール Firmware Integration Technology (R01AN1819JJ)
- RX ファミリ コンペアマッチタイマ（CMT）モジュール Firmware Integration Technology (R01AN1856JJ)
- RX ファミリ ロングワード型キューバッファ（LONQ）モジュール Firmware Integration Technology (R01AN1889JJ)
- RX ファミリ SDHI モジュール Firmware Integration Technology (R01AN3852JJ)

2. API 情報

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

SD カードドライバは以下のパッケージに依存しています。

- r_bsp
- r_sdhi_rx
- r_cgc_rx（クロック発生回路 CGC FIT モジュールが必要な RX ファミリ MCU を使用する場合のみ）
- 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イン

SD カードドライバは、「1.4 動作環境」に示すツールチェインで動作確認を行っています。

2.4 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"
```

2.5 整数型

SD カードドライバは 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。

2.6　コンパイル時の設定

SD カード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_sdc_sd_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_sdc_sd_rx_config.h*	
#define SDC_SD_CFG_CARD_NUM ※デフォルト値は“1”	制御する SD カード数を定義してください。
#define SDC_SD_CFG_DRIVER_MODE (SDC_SD_MODE_HWINT | SDC_SD_MODE_IO | SDC_SD_MODE_4BIT) ※デフォルト値は“ステータス確認 : ハードウェア割り込み、メディア対応 : SDIO、バス幅 : 4bit“	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_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。

2.7 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 で記載されています。

(1) sdc_sdio_daccess_t 構造体定義

```
typedef struct
{
    uint8_t     *p_buff;
    uint32_t    func;
    uint32_t    adr;
    uint32_t    raw_flag;
    uint32_t    rw_flag;
} sdc_sdio_daccess_t;
```

(2) sdc_sdio_access_t 構造体定義

```
typedef struct
{
    uint8_t     *p_buff;
    uint32_t    func;
    uint32_t    adr;
    int32_t     cnt;
    uint32_t    op_code;
    uint8_t     trans_mode;
    uint8_t     rsv[3];
} sdc_sdio_access_t;
```

(3) sdc_sdio_cis_t 構造体定義

```
typedef struct
{
    uint32_t    func;
    int32_t     cnt;
    uint32_t    *p_comm_cis_ptr;
    uint8_t     *p_cis_buff;
} sdc_sdio_cis_t;
```

2.8　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 カードドライバのライブラリ関数は、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_SDIO_Read()関数／R_SDC_SDIO_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 2-1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_SDIO_ReadDirect()関数／R_SDC_SDIO_ReadSoftwareTrans()関数／R_SDC_SDIO_WriteDirect()関数／R_SDC_SDIO_WriteSoftwareTrans()関数も同様です。

表 2-1　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

2.9 FIT モジュールの追加方法

本モジュールは、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)、(3)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(2)、(4)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) 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 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。

(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。

3.　API 関数

SDIO ドライバは、SD Specifications Part 1 Physical Layer Specification と SD Specifications Part E1 SDIO Specification のプロトコルを使ったデバイスドライバです。

表 3-1にライブラリ関数一覧を、表 3-2にSDIOレジスタ空間アクセス関数一覧を示します。

また、SDIO ドライバの状態遷移図を図 3-1に示します。

表 3-1　ライブラリ関数一覧

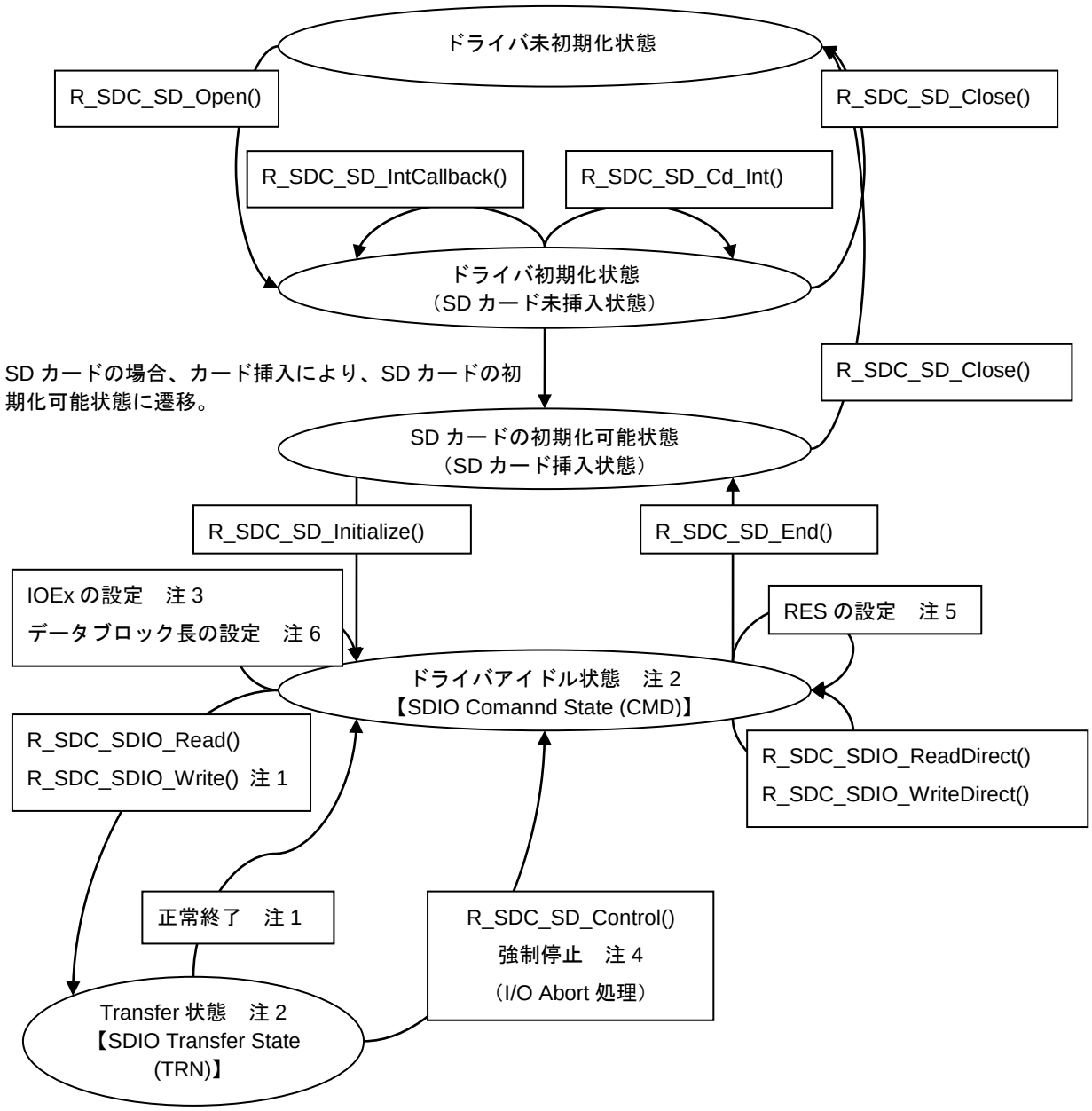
関数名	機能概要	備考
R_SDC_SD_Open()	ドライバのオープン処理	
R_SDC_SD_Close()	ドライバのクローズ処理	
R_SDC_SD_GetCardDetection()	挿入確認処理	
R_SDC_SD_Initialize()	初期化処理	
R_SDC_SD_End()	終了処理	
R_SDC_SDIO_ReadDirect()	SDIO のシングルレジスタリード処理	
R_SDC_SDIO_Read()	SDIO のレジスタリード処理	注 1
R_SDC_SDIO_ReadSoftwareTrans()	SDIO のレジスタリード処理（Software 転送）	
R_SDC_SDIO_WriteDirect()	SDIO のシングルレジスタライト処理	
R_SDC_SDIO_Write()	SDIO のレジスタライト処理	注 1
R_SDC_SDIO_WriteSoftwareTrans()	SDIO のレジスタライト処理（Software 転送）	
R_SDC_SDIO_GetBlocklen()	SDIO のデータブロック長情報取得処理	
R_SDC_SDIO_SetBlocklen()	SDIO のデータブロック長情報設定処理	
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_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_SDIO_IntCallback()	SDIO 割り込みコールバック関数登録処理	
R_SDC_SDIO_EnableInt()	SDIO 割り込み許可処理	
R_SDC_SDIO_DisableInt()	SDIO 割り込み禁止処理	
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「4.3.2 DMAC／DTC の制御方法」を参照してください。

注　2 : エラーログ機能は専用のライブラリモジュールで提供します。別途 LONGQ FIT モジュールが必要です。

表 3-2　SDIO レジスタ空間アクセス関数一覧

関数名	機能概要	備考
R_SDC_SDIO_IOAbort()	I/O Abort 処理	
R_SDC_SDIO_SetIOEnable()	IOEx（I/O Enable）設定処理	
R_SDC_SDIO_ClearIOEnable()	IOEx（I/O Enable）クリア処理	
R_SDC_SDIO_GetIOReady()	I/O Ready 取得処理	
R_SDC_SDIO_IOReset()	I/O Reset 処理	
R_SDC_SDIO_SetIntEnable()	IENM（Int Enable）設定処理	
R_SDC_SDIO_ClearIntEnable()	IENM（Int Enable）クリア処理	
R_SDC_SDIO_GetIntEnable()	Int Enable 取得処理	
R_SDC_SDIO_GetCIS()	CIS 取得処理	



注 1：R_SDC_SDIO_ReadSoftwareTrans()、R_SDC_SDIO_WriteSoftwareTrans()も同様。
注 2：【】は、SD Specifications Part E1 SDIO Specification 記載の SDIO バス状態を示す。
注 3：R_SDC_SDIO_WriteDirect()もしくは R_SDC_SDIO_SetIOEnable()を使用可能。
注 4：関数実行中は、SDHI の持つ SDIO abort ビットを制御し、I/O Abort 処理を実行。
注 5:R_SDC_SDIO_WriteDirect()もしくは R_SDC_SDIO_IOReset()を使用可能。引き続き、R_SDC_SD_End()を実行してください。
注 6：ブロックモードで転送する場合、事前に R_SDC_SDIO_SetBlocklen()を使用し設定してください。

図 3-1　状態遷移図

3.1 ライブラリ関数

SDIO への基本アクセス制御関数です。

(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 288 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。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
uint32_t       g_sdc_sd_work[288/sizeof(uint32_t)];

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

if (R_SDC_SD_Open(SDC_SD_CRAD0, 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。「4.1.1 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。
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。

(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。

Reentrant

異なるチャネルからリエントラントは可能です。

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。「4.1.2 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) 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。

Reentrant

異なるチャネルからリエントラントは可能です。

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。「4.1.1 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。「4.1.2 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。
SDHI_CD 端子の処理方法は、「5. ハードウェア設定」を参照してください。
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。

(4)　R_SDC_SD_Initialize()

＜SDIO の場合＞
SD カードを SDIO Initialization State (INI)から SDIO Command State (CMD)に遷移させる関数です。また、CMD52 発行により SD カード制御用内部情報を読み出し、ワーク領域に格納します。
本ドライバの状態は、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遷移します。

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 3-3　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 3-4　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。「4.1.4 初期化時の動作電圧設定について」も参照してください。

表 3-3　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
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	0x0020	SD メモリカード／SD メモリ
	SDC_SD_MODE_IO	0x0010	SDIO カード／SDIO
	SDC_SD_MODE_COMBO	0x0030	SD メモリと SDIO
SD バス対応方式	SDC_SD_MODE_1BIT	0x0100	SD モード 1 ビットバス
	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：使用するライブラリ関数によっては 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 3-4 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 3-3 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。

初期化完了後に、SD Combo の SD メモリもしくは SDIO のみを個別に再初期化したい場合、以下の手順で API 関数をコールしてください。

1. R_SDC_SDIO_IOReset()関数
2. R_SDC_SD_End()関数 注 1
3. R_SDC_SD_Initialize()関数 注 1

注 1 : 引数 end_type と init_type に再初期化したい対象である SD Combo の SD メモリもしくは SDIO を設定してください。

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。
＜SDIO の場合＞
戻り値が SDC_SD_SUCCESS の場合、SDIO は Command State (CMD)へ遷移し、SDIO のリード／ライトアクセスが可能になります。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
sdc_sd_cfg_t      sdc_sd_config;

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

sdc_sd_config.mode = SDCFG_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_IO) !=
SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

SD カードドライバは、High-Speed カードと Default Speed カードを判別し、High-Speed 設定を優先し、SD カードを初期化します。ただし、使用する MCU の SDHI が Default Speed のみをサポートする場合、Default Speed 設定で初期化します。なお、Low Speed カードの場合、Low Speed 設定で初期化します。
本関数実行前に、端子設定が必要です。「4.1.1 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 カードの抜去やハードウェア・リセットによる 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（SDメモリ時）もしくはCMD5（SDIO時）を発行します。

＜SDIO の場合＞
本関数の正常終了時に、SDHI の SDIO モードコントロールレジスタ（SDIOMD）の SDIO Interrupt 受け付け許可ビット（INTEN）が 1（SDIO Interrupt 受付許可）に設定され、SDIO ステータスレジスタ（SDIOSTS）の SDIO 割り込みフラグ（IOIRQ）をクリアし、SDIO 割り込みマスクレジスタ（SDIOIMSK）の IOIRQ 割り込みマスクビット（IOIRQ）を”0”（SDIO Interrupt の受け付け割り込み要求をマスクしない）に設定され、SDIO Interrupt 受付状態になります。
本関数の正常終了後に、SDIO のファンクション毎の有効化処理（IOEn=1 設定）を行ってください。なお、SDIO Interrupt 受付後は、SDIO Interrupt 受付禁止状態になります。再び SDIO Interrupt 受付許可状態にしたい場合は、R_SDC_SDIO_EnableInt()をコールしてください。
また、本関数で CD Disable=1 を設定済みのため、CMD53 発行を伴う SDIO へのアクセスが可能です。
本関数実行前に、必要に応じてハードウェア・リセット処理やI/O Reset処理を行ってください。

(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 3-3　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。

初期化完了後に、SD Combo の SD メモリもしくは SDIO のみを個別に再初期化したい場合、
以下の手順で API 関数をコールしてください。

1. R_SDC_SDIO_IOReset()関数
2. R_SDC_SD_End()関数　注 1
3. R_SDC_SD_Initialize()関数　注 1

注 1 : 引数 end_type と init_type に再初期化したい対象である SD Combo の SD メモリもしくは SDIO を設定してください。

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。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
if (R_SDC_SD_End(SDC_SD_CARD0, SDC_SD_MODE_IO) != 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。「4.1.2 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。

(6)　R_SDC_SDIO_ReadDirect()

CMD52 を使用し、レジスタのシングルリード処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_ReadDirect(
      uint32_t card_no,
      sdc_sdio_daccess_t * p_sdc_sdio_daccess
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

**p_sdc_sdio_daccess*

SDIO ダイレクト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。

func　：ファンクション番号（0 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

adr　：レジスタアドレス（0x00000 - 0x1FFFF）

raw_flag　：Read After Write フラグ（設定不要）

rw_flag　：リードライトフラグ（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_sdio_daccess の func で設定したファンクション番号の引数 p_sdc_sdio_daccess の adr で設定した I/O レジスタ空間から、CMD52 を使って 1 バイトデータを読み出し、引数 p_sdc_sdio_daccess 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。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
sdc_sdio_daccess_t sdc_sdio_daccess;
uint8_t io_buff[4] = {0, 0, 0, 0};

sdc_sdio_daccess.p_buff = &io_buff[0];
sdc_sdio_daccess.func = SDC_SDIO_FNO_0;
sdc_sdio_daccess.adr = SDC_SDIO_ADRS_IOENABLE;
if (R_SDC_SDIO_ReadDirect(SDC_SD_CARD0, &sdc_sdio_daccess) != 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。

(7)　R_SDC_SDIO_Read()

CMD53 を使用し、レジスタのリード処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_Read(
      uint32_t card_no,
      sdc_sdio_access_t * p_sdc_sdio_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_sdio_access*

SDIO アクセス情報構造体

p_buff　：読み出しバッファポインタ

DMAC 転送／DTC 転送の場合、4 バイト境界のアドレスを設定してください。

func　：ファンクション番号（0 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

adr　：レジスタアドレス（0x00000 - 0x1FFFF）

cnt　：バイト数

バイト数を設定してください。なお、DMAC 転送／DTC 転送設定時、設定できる値に制限があり、以下のとおり設定してください。

ブロック転送モード：データブロック長の整数倍（1-256）のバイト数

バイト転送モード：「表 3-5　DMAC 転送／DTC転送設定時の設定可能なバイト転送モード時のcnt値」で示す値を設定してください。

op_code　：Operation code

「表 3-6　SDカードドライバ　Operation code（op_code値）」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。ブロック転送モード設定時は、事前に R_SDC_SDIO_SetBlocklen()関数を使って、データブロック長を設定してください。

trans_mode　：転送モード（設定不要）

表 3-5　DMAC 転送／DTC 転送設定時の設定可能なバイト転送モード時の cnt 値

ファンクション番号	値
0	32
1	64
2	512

表 3-6　SD カードドライバ　Operation code（op_code 値）

種別	マクロ定義	値（ビット）	定義
アドレスモード	SDC_SDIO_FIXED_ADDR	0x00	固定アドレスモード
	SDC_SDIO_INCREMENT_ADDR	0x01	アドレスインクリメントモード
転送モード	SDC_SDIO_BYTE_MODE	0x00	バイト転送モード
	SDC_SDIO_BLOCK_MODE	0x10	ブロック転送モード

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_sdio_access の func で設定したファンクション番号の引数 p_sdc_sdio_access の adr で設定した I/O レジスタ空間から、CMD53 を使って引数 p_sdc_sdio_access の cnt バイト分のデータを読み出し、引数 p_sdc_sdio_access の p_buff に格納します。
ブロック転送開始時に 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。
本関数開始時に、ブロック転送モード設定時、データブロック長を確認します。データブロック長が未設定の場合、エラー終了（SDC_SD_ERR_BLOCK_LEN_ERROR）を返します。

Reentrant
異なるチャネルからリエントラントは可能です。

Example
＜ブロック転送モードの場合＞
```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;

/* ---- Set block length. ----*/
if (R_SDC_SDIO_SetBlocklen(SDC_SD_CARD0, 512, SDC_SDIO_FNO_1) != SDC_SD_SUCCESS)
{
   /* Error */
}

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)2048;
sdc_sdio_access.op_code = (SDC_SDIO_BLOCK_MODE | SDC_SDIO_INCREMENT_ADDR);
if ( R_SDC_SDIO_Read(SDC_SD_CARD0, &sdc_sdio_access) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

＜バイト転送モードの場合＞
```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)4;
sdc_sdio_access.op_code = (SDC_SDIO_BYTE_MODE | SDC_SDIO_INCREMENT_ADDR);
if ( R_SDC_SDIO_Read(SDC_SD_CARD0, &sdc_sdio_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。

(8)　R_SDC_SDIO_ReadSoftwareTrans()

CMD53 を使用し、レジスタのリード処理を実行する関数です。
R_SDC_SD_Initialize()関数により動作モードmodeをDMAC転送もしくはDTC転送に設定した場合でも、Software転送を実行します。

Format

```
sdc_sd_status_t R_SDC_SDIO_ReadSoftwareTrans(
    uint32_t card_no,
    sdc_sdio_access_t * p_sdc_sdio_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_sdio_access*
　SDIO 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。
　func　：ファンクション番号（0 - 最大 7）
　　SDIO がサポートするファンクション番号に依存します。
　adr　：レジスタアドレス（0x00000 - 0x1FFFF）
　cnt　：バイト数
　　バイト数を設定してください。
　op_code　：Operation code
　　「表 3-7　SDカードドライバ　Operation code（op_code値）」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。ブロック転送モード設定時は、事前に R_SDC_SDIO_SetBlocklen()関数を使って、データブロック長を設定してください。
　trans_mode　：転送モード（設定不要）

表 3-7　SD カードドライバ　Operation code（op_code 値）

種別	マクロ定義	値（ビット）	定義
アドレスモード	SDC_SDIO_FIXED_ADDR	0x00	固定アドレスモード
	SDC_SDIO_INCREMENT_ADDR	0x01	アドレスインクリメントモード
転送モード	SDC_SDIO_BYTE_MODE	0x00	バイト転送モード
	SDC_SDIO_BLOCK_MODE	0x10	ブロック転送モード

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_sdio_access の func で設定したファンクション番号の引数 p_sdc_sdio_access の adr で設定した I/O レジスタ空間から、CMD53 を使って引数 p_sdc_sdio_access の cnt バイト分のデータを読み出し、引数 p_sdc_sdio_access の p_buff に格納します。
ブロック転送開始時に 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。

Reentrant
異なるチャネルからリエントラントは可能です。

Example
＜ブロック転送モードの場合＞
```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;

/* ---- Set block length. ----*/
if (R_SDC_SDIO_SetBlocklen(SDC_SD_CARD0, 512, SDC_SDIO_FNO_1) != SDC_SD_SUCCESS)
{
   /* Error */
}

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)2048;
sdc_sdio_access.op_code = (SDC_SDIO_BLOCK_MODE | SDC_SDIO_INCREMENT_ADDR);
if ( R_SDC_SDIO_ReadSoftwareTrans(SDC_SD_CARD0, &sdc_sdio_access) !=
SDC_SD_SUCCESS)
{
   /* Error */
}
```

＜バイト転送モードの場合＞
```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)4;
sdc_sdio_access.op_code = (SDC_SDIO_BYTE_MODE | SDC_SDIO_INCREMENT_ADDR);
if ( R_SDC_SDIO_ReadSoftwareTrans(SDC_SD_CARD0, &sdc_sdio_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。

(9) R_SDC_SDIO_WriteDirect()

CMD52 を使用し、レジスタのシングルライト処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_WriteDirect(
    uint32_t card_no,
    sdc_sdio_daccess_t * p_sdc_sdio_daccess
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

**p_sdc_sdio_daccess*

SDIO ダイレクト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。

func ：ファンクション番号（0 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

adr ：レジスタアドレス（0x00000 - 0x1FFFF）

raw_flag ：Read After Write フラグ

SDC_SDIO_SIMPLE_WRITE：書き込みのみモード（*p_buff は書き込み時の値です。）

SDC_SDIO_READ_AFTER_WRITE：書き込み後のリード実行モード（*p_buff は書き込み後のレジスタからの読み出し値です。）

rw_flag ：リードライトフラグ（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_sdio_daccess の func で設定したファンクション番号の引数 p_sdc_sdio_daccess の adr で設定した I/O レジスタ空間に、CMD52 を使って p_sdc_sdio_daccess の p_buff の 1 バイト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。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
sdc_sdio_daccess_t sdc_sdio_daccess;
uint8_t io_buff[4] = {0, 0, 0, 0};

io_buff[0] = 0x01;
sdc_sdio_daccess.p_buff = &io_buff[0];
sdc_sdio_daccess.func = SDC_SDIO_FNO_0;
sdc_sdio_daccess.adr = SDC_SDIO_ADRS_IOENABLE;
sdc_sdio_daccess.raw_flag = SDC_SDIO_READ_AFTER_WRITE;
if (R_SDC_SDIO_WriteDirect(SDC_SD_CARD0, &sdc_sdio_daccess) != 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。

(10) R_SDC_SDIO_Write()

CMD53 を使用し、レジスタのライト処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_Write(
    uint32_t card_no,
    sdc_sdio_access_t * p_sdc_sdio_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_sdio_access*

SDIO アクセス情報構造体

**p_buff　：書き込みバッファポインタ*

DMAC 転送／DTC 転送の場合、4 バイト境界のアドレスを設定してください。

func　：ファンクション番号（0 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

adr　：レジスタアドレス（0x00000 - 0x1FFFF）

cnt　：バイト数

バイト数を設定してください。なお、DMAC 転送／DTC 転送設定時、設定できる値に制限があり、以下のとおり設定してください。

ブロック転送モード：データブロック長の整数倍（1-256）のバイト数

バイト転送モード：「表 3-8　DMAC 転送／DTC 転送設定時の設定可能なバイト転送モード時のcnt値」で示す値を設定してください。

op_code　：Operation code

「表 3-9　SDカードドライバ　Operation code（op_code値）」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。ブロック転送モード設定時は、事前に R_SDC_SDIO_SetBlocklen()関数を使って、データブロック長を設定してください。

trans_mode　：転送モード（設定不要）

表 3-8　DMAC 転送／DTC 転送設定時の設定可能なバイト転送モード時の cnt 値

ファンクション番号	値
0	32
1	64
2	512

表 3-9　SD カードドライバ　Operation code（op_code 値）

種別	マクロ定義	値（ビット）	定義
アドレスモード	SDC_SDIO_FIXED_ADDR	0x00	固定アドレスモード
	SDC_SDIO_INCREMENT_ADDR	0x01	アドレスインクリメントモード
転送モード	SDC_SDIO_BYTE_MODE	0x00	バイト転送モード
	SDC_SDIO_BLOCK_MODE	0x10	ブロック転送モード

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_sdio_access の func で設定したファンクション番号の引数 p_sdc_sdio_access の adr で設定した I/O レジスタ空間に、CMD53 を使って p_sdc_sdio_access の cnt バイト分のデータを書き込みます。
ブロック転送開始時に 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。
本関数開始時に、ブロック転送モード設定時、データブロック長を確認します。データブロック長が未設定の場合、エラー終了（SDC_SD_ERR_BLOCK_LEN_ERROR）を返します。

Reentrant
異なるチャネルからリエントラントは可能です。

Example
＜ブロック転送モードの場合＞
```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;

/* ---- Set block length. ----*/
if (R_SDC_SDIO_SetBlocklen(SDC_SD_CARD0, 512, SDC_SDIO_FNO_1) != SDC_SD_SUCCESS)
{
   /* Error */
}

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)2048;
sdc_sdio_access.op_code = (SDC_SDIO_BLOCK_MODE | SDC_SDIO_INCREMENT_ADDR);
if ( R_SDC_SDIO_Write(SDC_SD_CARD0, &sdc_sdio_access) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

＜バイト転送モードの場合＞
```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)4;
sdc_sdio_access.op_code = (SDC_SDIO_BYTE_MODE | SDC_SDIO_INCREMENT_ADDR);
if ( R_SDC_SDIO_Write(SDC_SD_CARD0, &sdc_sdio_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。
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。

(11) R_SDC_SDIO_WriteSoftwareTrans()

CMD53 を使用し、レジスタのライト処理を実行する関数です。
R_SDC_SD_Initialize()関数により動作モードmodeをDMAC転送もしくはDTC転送に設定した場合でも、Software転送を実行します。

Format

```
sdc_sd_status_t R_SDC_SDIO_WriteSoftwareTrans(
    uint32_t card_no,
    sdc_sdio_access_t * p_sdc_sdio_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_sdio_access*
　SDIO 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。
　func　：ファンクション番号（0 - 最大 7）
　　SDIO がサポートするファンクション番号に依存します。
　adr　：レジスタアドレス（0x00000 - 0x1FFFF）
　cnt　：バイト数
　　バイト数を設定してください。
　op_code　：Operation code
　　「表 3-10　SDカードドライバ　Operation code（op_code値）」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。ブロック転送モード設定時は、事前に R_SDC_SDIO_SetBlocklen()関数を使って、データブロック長を設定してください。
　trans_mode　：転送モード（設定不要）

表 3-10　SD カードドライバ　Operation code（op_code 値）

種別	マクロ定義	値（ビット）	定義
アドレスモード	SDC_SDIO_FIXED_ADDR	0x00	固定アドレスモード
	SDC_SDIO_INCREMENT_ADDR	0x01	アドレスインクリメントモード
転送モード	SDC_SDIO_BYTE_MODE	0x00	バイト転送モード
	SDC_SDIO_BLOCK_MODE	0x10	ブロック転送モード

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_sdio_access の func で設定したファンクション番号の引数 p_sdc_sdio_access の adr で設定した I/O レジスタ空間に、CMD53 を使って p_sdc_sdio_access の cnt バイト分のデータを書き込みます。
ブロック転送開始時に 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。

Reentrant
異なるチャネルからリエントラントは可能です。

Example
＜ブロック転送モードの場合＞
```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;

/* ---- Set block length. ----*/
if (R_SDC_SDIO_SetBlocklen(SDC_SD_CARD0, 512, SDC_SDIO_FNO_1) != SDC_SD_SUCCESS)
{
   /* Error */
}

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)2048;
sdc_sdio_access.op_code = (SDC_SDIO_BLOCK_MODE | SDC_SDIO_INCREMENT_ADDR);
if ( R_SDC_SDIO_WriteSoftwareTrans(SDC_SD_CARD0, &sdc_sdio_access) !=
SDC_SD_SUCCESS)
{
   /* Error */
}
```

＜バイト転送モードの場合＞
```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)4;
sdc_sdio_access.op_code = (SDC_SDIO_BYTE_MODE | SDC_SDIO_INCREMENT_ADDR);
if ( R_SDC_SDIO_WriteSoftwareTrans(SDC_SD_CARD0, &sdc_sdio_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。
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。

(12) R_SDC_SDIO_GetBlocklen()

CMD53 によるマルチブロック転送使用時のデータブロック長を取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_GetBlocklen(
    uint32_t card_no,
    int32_t * p_len,
    uint32_t func
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

**p_len*

データブロック長用バッファポインタ（4 バイト）

func

ファンクション番号（0 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

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 func で設定したファンクション番号の I/O デバイスのデータブロック長情報を引数 p_len に格納します。データブロック長の初期値は、0 が設定されています。マルチブロック転送実行前に、ファンクション毎にデータブロック長を設定してください。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
int32_t len = 0;

if (R_SDC_SDIO_GetBlocklen(SDC_SD_CARD0, &len, SDC_SDIO_FNO_1) != 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。

*p_lenが0xFFFFの場合、R_SDC_SDIO_SetBlocklen()関数コール時の設定されたデータブロック長が異常であったことを示します。再度R_SDC_SDIO_SetBlocklen()関数をコールし、データブロック長を再設定してください。

(13) R_SDC_SDIO_SetBlocklen()

CMD53 によるマルチブロック転送使用時のデータブロック長を設定する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_SetBlocklen(
    uint32_t card_no,
    int32_t len,
    uint32_t func
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

len

データブロック長（32, 64, 128, 256, 512）

func

ファンクション番号（0 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

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 func で設定したファンクション番号の Card Common Control Registers (CCCR)もしくは Function Basic Registers (FBR)の I/O block size にデータブロック長を設定します。
データブロック長の初期値は、0 が設定されています。マルチブロック転送実行前に、ファンクション毎にデータブロック長を設定してください。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
if (R_SDC_SDIO_SetBlocklen(SDC_SD_CARD0, 512, SDC_SDIO_FNO_1) != 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。

エラー終了の場合、引き続く R_SDC_SDIO_Read()関数、R_SDC_SDIO_ReadSoftwareTrans()関数、R_SDC_SDIO_Write()関数、R_SDC_SDIO_WriteSoftwareTrans()関数は正常に動作しません。

(14) 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,
    sdhi_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 3-11　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 3-11　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。

Reentrant

異なるチャネルからリエントラントは可能です。

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。
SDIOの場合,データ転送中のI/O Abort処理を実行します。

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。
＜SDIO の場合＞
SD カードに対してデータ転送途中であった場合、Command State (CMD)に状態遷移させる目的で、SD カードに対して CMD52 を使った I/O Abort 処理を実行し、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、CMD52 を使った I/O Abort 処理により、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_SDIO_Read()関数／R_SDC_SDIO_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

```
sdhi_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、設定されたデータブロック長分の転送完了後で、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_SDIO_Read()関数／R_SDC_SDIO_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_SDIO_Read()関数／R_SDC_SDIO_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（SDC_SD_ERR_STOP）を返します。

注 1：R_SDC_SDIO_ReadSoftwareTrans()関数／R_SDC_SDIO_WriteSoftwareTrans()関数／R_SDC_SDIO_ReadDirect()関数／R_SDC_SDIO_WriteDirect()関数も同様です。

(15) 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 3-3　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。

Reentrant

異なるチャネルからリエントラントは可能です。

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。

(16) 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 3-12　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 3-13　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　：Speed mode 情報

bit4	bit0
0 : Default Speed mode のみ対応カード	0 : Default Speed mode でアクセス
1 : High-speed mode 対応カード	1 : High-speed mode でアクセス

io_speed_mode　：SDIO Speed mode 情報

bit5-4	bit1-0
00 : Default Speed mode のみ対応カード	00 : Default Speed mode でアクセス
01 : High-speed mode 対応カード	01 : High-speed mode でアクセス
10 : Low Speed 対応カード（SDIO）	10 : Low Speed（Default Speed mode）でアクセス

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 3-12　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 3-13 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）

Reentrant
異なるチャネルからリエントラントは可能です。

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。

(17) 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（上位 16 ビット）

scr[2] ：SD メモリ SCR 情報

sdstatus[4] ：SD メモリ SD Status 情報

switch_func_status[5] ：SD メモリ Switch Function Status 情報

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。

Reentrant

異なるチャネルからリエントラントは可能です。

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。

(18) 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。

Reentrant

異なるチャネルからリエントラントは可能です。

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。

(19) 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。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
/* Callback function */
sdc_sd_status_t r_sdc_sd_callback(int32_t card_no)
{
    if(SDC_SD_CARD0 == card_no)
    {
        nop();/* int in */
    }
    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。

(20) R_SDC_SDIO_IntCallback()

SDIO 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_SDIO_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

SDIO 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、SDHI の SDIO 割り込み発生時にコールされます。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
/* Callback function */
sdc_sd_status_t r_sdc_sdio_callback(int32_t card_no)
{
    if(SDC_SD_CARD0 == card_no)
    {
        nop();/* int in */
    }
    return SDC_SD_SUCCESS;
}

if (R_SDC_SDIO_IntCallback(SDC_SD_CARD0, r_sdc_sdio_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および SD プロトコルステータス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および SD プロトコルステータス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。

(21) R_SDC_SDIO_EnableInt()

SDIO 割り込みを許可する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_EnableInt(
    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

SDIO ステータスレジスタ（SDIOSTS）の SDIO 割り込みフラグ（IOIRQ）をクリアし、SDIO 割り込みマスクレジスタ（SDIOIMSK）の IOIRQ 割り込みマスクビット（IOIRQ）を”0”（SDIO Interrupt の受け付け割り込み要求をマスクしない）に設定します。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

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

Special Notes

SD カードの SDIO 割り込み信号をネゲートした状態で本関数をコールしてください。アサートされた状態で本関数をコールした場合、再び SDIO 割り込みを検出する可能性があります。なお、SDIO Interrupt 受付後は、SDIO Interrupt 受付禁止状態になります。再び SDIO Interrupt 受付許可状態にしたい場合は、R_SDC_SDIO_EnableInt()をコールしてください。
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。
R_SDC_SD_GetErrCode()関数によるエラーコード取得はできません。

(22) R_SDC_SDIO_DisableInt()

SDIO 割り込みを禁止する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_DisableInt(
    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

SDIO 割り込みマスクレジスタ（SDIOIMSK）の IOIRQ 割り込みマスクビット（IOIRQ）を”1”（SDIO Interrupt の受け付け割り込み要求をマスクする）に設定します。

Reentrant

異なるチャネルからリエントラントは可能です。

Example

```
if (R_SDC_SDIO_DisableInt(SDC_SD_CARD0) != 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。

(23) 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_SDIO_Read()関数／R_SDC_SDIO_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_SDIO_ReadSoftwareTrans()関数／R_SDC_SDIO_WriteSoftwareTrans()関数／R_SDC_SDIO_WriteDirect()関数／R_SDC_SDIO_WriteSoftwareTrans()関数／R_SDC_SDIO_SetBlocklen()関数／R_SDC_SDIO_GetBlocklen()関数も同様

Reentrant

異なるチャネルからリエントラントは可能です。

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_IO) !=
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。

(24) 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

card_no

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。

Reentrant

異なるチャネルからリエントラントは可能です。

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。

(25) 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。

Reentrant
リエントラントは不可能です。

Example
```
uint32_t    g_cmt_card_no;

void r_cmt_callback(void * pdata)
{
   uint32_t card_no;

   card_no = *((uint32_t *)pdata);
   if (card_no == g_cmt_card_no)
   {
      R_SDC_SD_1msInterval();
   }
}

main()
{
    /* Create CMT timer */
    R_CMT_CreatePeriodic(1000, &r_cmt_callback, &g_cmt_card_no);    /* 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_user.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。

(26) 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 3-14 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 3-14 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

Reentrant

異なるチャネルからリエントラントは可能です。

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＞

```
uint32_t g_sdio_buff[2048/sizeof(uint32_t)];
sdc_sdio_access_t sdc_sdio_access;
uint32_t reg_buff = 0;

/* ---- Set block length. ----*/
if (R_SDC_SDIO_SetBlocklen(SDC_SD_CARD0, 512, SDC_SDIO_FNO_1) != SDC_SD_SUCCESS)
{
   /* Error */
}

if (R_SDC_SD_GetBuffRegAddress(SDC_SD_CARD0, &reg_buff) != SDC_SD_SUCCESS)
{
   /* Error */
}

sdc_sdio_access.p_buff = (uint8_t *)&g_sdio_buff[0];
sdc_sdio_access.func = SDC_SDIO_FNO_1;
sdc_sdio_access.adr = 0;
sdc_sdio_access.cnt = (int32_t)2048;
sdc_sdio_access.op_code = (SDC_SDIO_BLOCK_MODE | SDC_SDIO_INCREMENT_ADDR);

/* Enable DMACA FIT module. */
r_dmaca_enable(&sdc_sdio_access, SDIO_READ_OP, reg_buff, 512);

if ( R_SDC_SDIO_Read(SDC_SD_CARD0, &sdc_sdio_access) != SDC_SD_SUCCESS)
{
   /* Error */
   R_SDC_SD_SetDmacDtcTransFlg(SDC_SD_CARD0, SDC_SD_SET_TRANS_STOP);
}

/* Disable DMACA FIT module. */
r_dmaca_disable();
```

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。

(27) 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。

Reentrant
異なるチャネルからリエントラントは可能です。

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
デバッグ用モジュールを使用してください。
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。

(28) 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。

Reentrant

異なるチャネルからリエントラントは可能です。

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_IO) !=
SDC_SD_SUCCESS)
{
   /* Error */
   R_SDC_SD_Log(USER_DRIVER_ID, USER_LOG_MAX, USER_LOG_ADR_MAX);
}
```

Special Notes

デバッグ用モジュールを使用してください。
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。

(29) 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。

Reentrant
リエントラントは可能です。

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。

3.2 SDIO レジスタ空間アクセス関数

SDIO レジスタ空間の特定レジスタへのアクセス関数です。3.1 ライブラリ関数に示す関数を使った応用関数です。

(1) R_SDC_SDIO_IOAbort()

I/O Abort レジスタの ASx（x=0 - 2）を使った I/O Abort 処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_IOAbort(
    uint32_t card_no,
    uint32_t func
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

func

ファンクション番号（0 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

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

R_SDC_SDIO_WriteDirect()関数を使って、引数 func で設定したファンクション番号の AS[2:0]を設定します。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 、関数内でリトライし、処理を続けます。

本関数はドライバアイドル状態のみコール可能です。転送中にコールした場合エラーが返る可能性があります。

Reentrant

リエントラントは不可能です。

Example

```
if (R_SDC_SDIO_IOAbort(SDC_SD_CARD0, SDC_SDIO_FNO_1) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

他関数の実行中は、本関数を実行できません。他関数の実行中に I/O Abort 処理を実行した場合は、割り込み処理ルーチン内で R_SDC_SD_Control()関数（SDC_SD_SET_STOP コマンド）をコールしてください。

(2) R_SDC_SDIO_SetIOEnable()

I/O Enable レジスタの IOEx（x=1 - 最大 7）を 1 に設定する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_SetIOEnable(
    uint32_t card_no,
    uint32_t func
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

func

ファンクション番号（1 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

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

R_SDC_SDIO_ReadDirect()関数と R_SDC_SDIO_WriteDirect()関数を使って、引数 func で設定したファンクション番号の IOE のみを 1 に設定します。設定後、I/O Ready レジスタを読み出して、初期化完了を確認してください。

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 、関数内でリトライし、処理を続けます。

ファンクション番号に0を設定できません。

Reentrant

リエントラントは不可能です。

Example

```
if (R_SDC_SDIO_SetIOEnable(SDC_SD_CARD0, SDC_SDIO_FNO_1) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

なし

(3) R_SDC_SDIO_ClearIOEnable()

I/O Enable レジスタの IOEx（x=1 - 最大 7）を 0 にクリアする関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_ClearIOEnable(
    uint32_t card_no,
    uint32_t func
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

func

ファンクション番号（1 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

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

R_SDC_SDIO_ReadDirect()関数と R_SDC_SDIO_WriteDirect()関数を使って、引数 func で設定したファンクション番号の IOE のみを 0 にクリアします。

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 、関数内でリトライし、処理を続けます。

ファンクション番号に0を設定できません。

Reentrant

リエントラントは不可能です。

Example

```
if (R_SDC_SDIO_ClearIOEnable(SDC_SD_CARD0, SDC_SDIO_FNO_1) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

なし

(4) R_SDC_SDIO_GetIOReady()

I/O Ready レジスタの値を取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_GetIOReady(
    uint32_t card_no,
    uint8_t *p_data
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

**p_data*

I/O Ready 情報格納ポインタ（1 バイト）。

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

R_SDC_SDIO_ReadDirect()関数を使って、I/O Ready レジスタを読み出し、引数 p_data に格納します。
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 、関数内でリトライし、処理を続けます。

Reentrant

リエントラントは可能です。

Example

```
if (R_SDC_SDIO_GetIOReady(SDC_SD_CARD0, &io_buff[0]) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

なし

(5) R_SDC_SDIO_IOReset()

I/O Abort レジスタの RES を使った I/O Reset 処理を実行する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_IOReset(
    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*
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

R_SDC_SDIO_WriteDirect()関数を使って、I/O Abort レジスタの RES を 1 に設定します。
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 、関数内でリトライし、処理を続けます。
本関数はドライバアイドル状態のみコール可能です。転送中にコールした場合エラーが返る可能性があります。

Reentrant

リエントラントは可能です。

Example

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

Special Notes

他関数の実行中は、本関数を実行できません。他関数の実行中に I/O Abort 処理を実行した場合は、割り込み処理ルーチン内で R_SDC_SD_Control()関数（SDC_SD_SET_STOP コマンド）をコールしてください。

(6) R_SDC_SDIO_SetIntEnable()

Int Enable レジスタの IENM を使った割り込み許可処理を実行する関数です。

Format
sdc_sd_status_t R_SDC_SDIO_SetIntEnable(
    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*
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
R_SDC_SDIO_ReadDirect()関数と R_SDC_SDIO_WriteDirect()関数を使って、Int Enable レジスタの IENM のみを 1 に設定します。
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 、関数内でリトライし、処理を続けます。

Reentrant
リエントラントは可能です。

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

Special Notes
なし

(7) R_SDC_SDIO_ClearIntEnable()

Int Enable レジスタの IENM を使った割り込み禁止処理を実行する関数です。

Format
```
sdc_sd_status_t R_SDC_SDIO_ClearIntEnable(
    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*
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
R_SDC_SDIO_ReadDirect()関数と R_SDC_SDIO_WriteDirect()関数を使って、Int Enable レジスタの IENM のみを 0 にクリアします。
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 、関数内でリトライし、処理を続けます。

Reentrant
リエントラントは可能です。

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

Special Notes
なし

(8) R_SDC_SDIO_GetIntEnable()

Int Enable レジスタの値を取得する関数です。

Format
```
sdc_sd_status_t R_SDC_SDIO_GetIntEnable(
    uint32_t card_no,
    uint8_t *p_data
)
```

Parameters
card_no

SD カード番号　　使用する SD カード番号（0 起算）

**p_data*

Int Enable 情報格納ポインタ（1 バイト）。

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
R_SDC_SDIO_ReadDirect()関数を使って、Int Enable レジスタを読み出し、引数 p_data に格納します。
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 、関数内でリトライし、処理を続けます。

Reentrant
リエントラントは可能です。

Example
```
uint8_t io_buff[4] = {0, 0, 0, 0};

if (R_SDC_SDIO_GetIntEnable(SDC_SD_CARD0, &io_buff[0]) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes
なし

(9) R_SDC_SDIO_GetCIS()

CIS 情報を取得する関数です。

Format

```
sdc_sd_status_t R_SDC_SDIO_GetCIS(
    uint32_t card_no,
    sdc_sdio_cis_t * p_sdc_sdio_cis
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

** p_sdc_sdio_cis*

CIS 情報格納ポインタ

func ：ファンクション番号（0 - 最大 7）

SDIO がサポートするファンクション番号に依存します。

cnt ：バイト数（２のべき乗の値、かつ最大 512）

**p_comm_cis_ptr ：設定したファンクションの CIS ポインタ情報のバッファポインタ*

**p_cis_buf ：設定したファンクションの CIS の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。

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_sdio_cis の func で設定したファンクション番号の CIS 情報を引数 p_sdc_sdio_cis の cnt バイト分のデータを読み出し、引数 p_sdc_sdio_cis の p_cis_buff に格納します。
また、エラー終了の場合であっても、p_sdc_sdio_cis の p_comm_cis_ptr に CIS ポインタ情報（各ファンクションのアドレス 09h-0bh）が設定されます。
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 、関数内でリトライし、処理を続けます。

Reentrant

リエントラントは可能です。

Example

```
sdc_sdio_cis_t sdc_sdio_cis;
uint32_t cis_adr = 0;
uint32_t g_sdio_cis[3][512/sizeof(uint32_t)];

sdc_sdio_cis.func = SDC_SDIO_FNO_0;
sdc_sdio_cis.cnt  = 32;
sdc_sdio_cis.p_comm_cis_ptr = &cis_adr;
sdc_sdio_cis.p_cis_buff = (uint8_t *)&g_sdio_cis[SDC_SDIO_FNO_0][0];
if (R_SDC_SDIO_GetCIS(SDC_SD_CARD0, &sdc_sdio_cis) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

なし

4.　ソフトウェア詳細

4.1　基本制御

4.1.1　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。

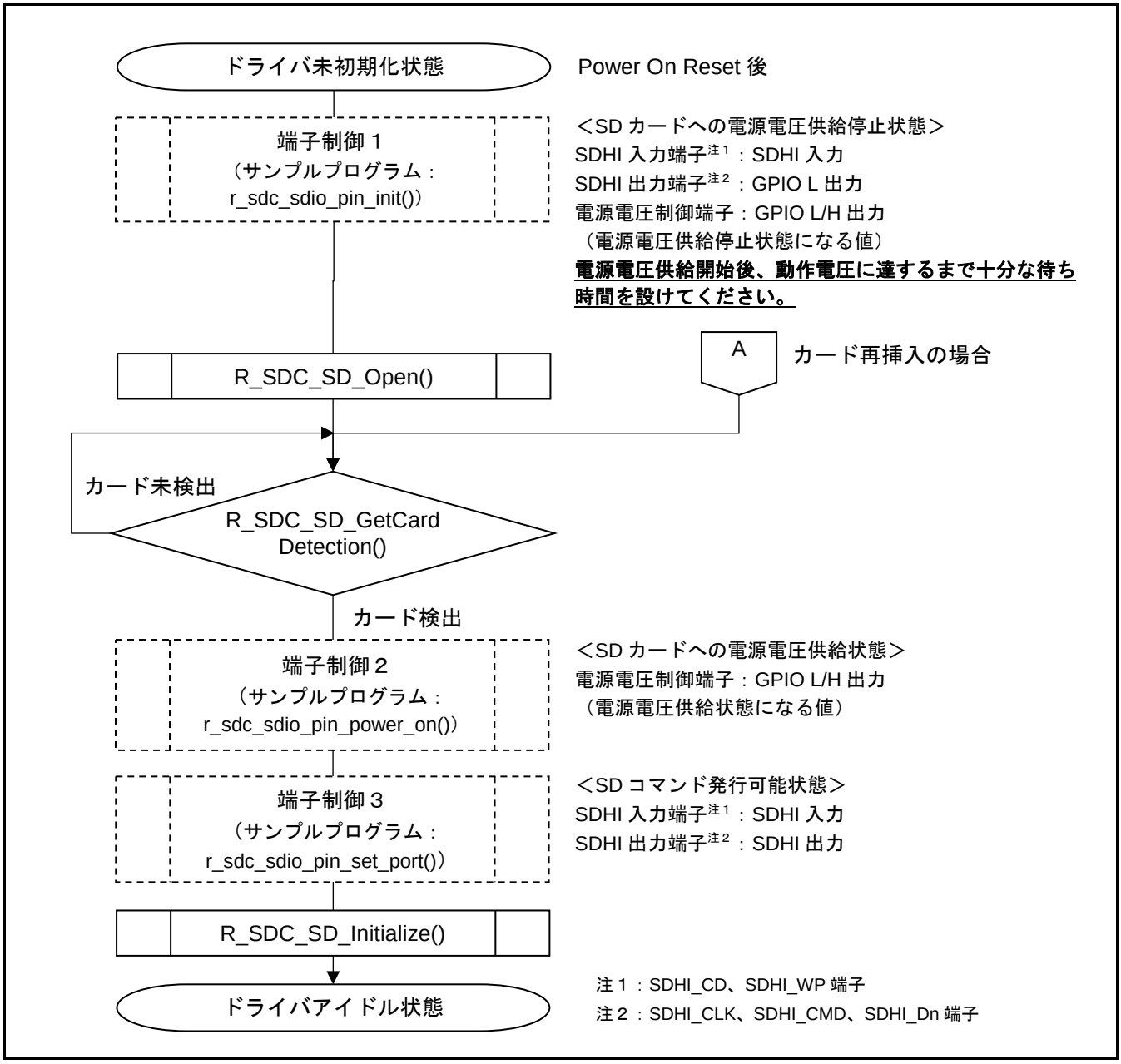


図 4-1　SD カードの挿入と電源投入タイミング

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：通常出力／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：SDHI PMR 設定：汎用入出力ポート	GPIO L 出力
	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 2	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 3	SDHI 入力端子 注 1	PMR 設定：周辺モジュール	SDHI 入力
	SDHI 出力端子 注 2	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。

4.1.2　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。

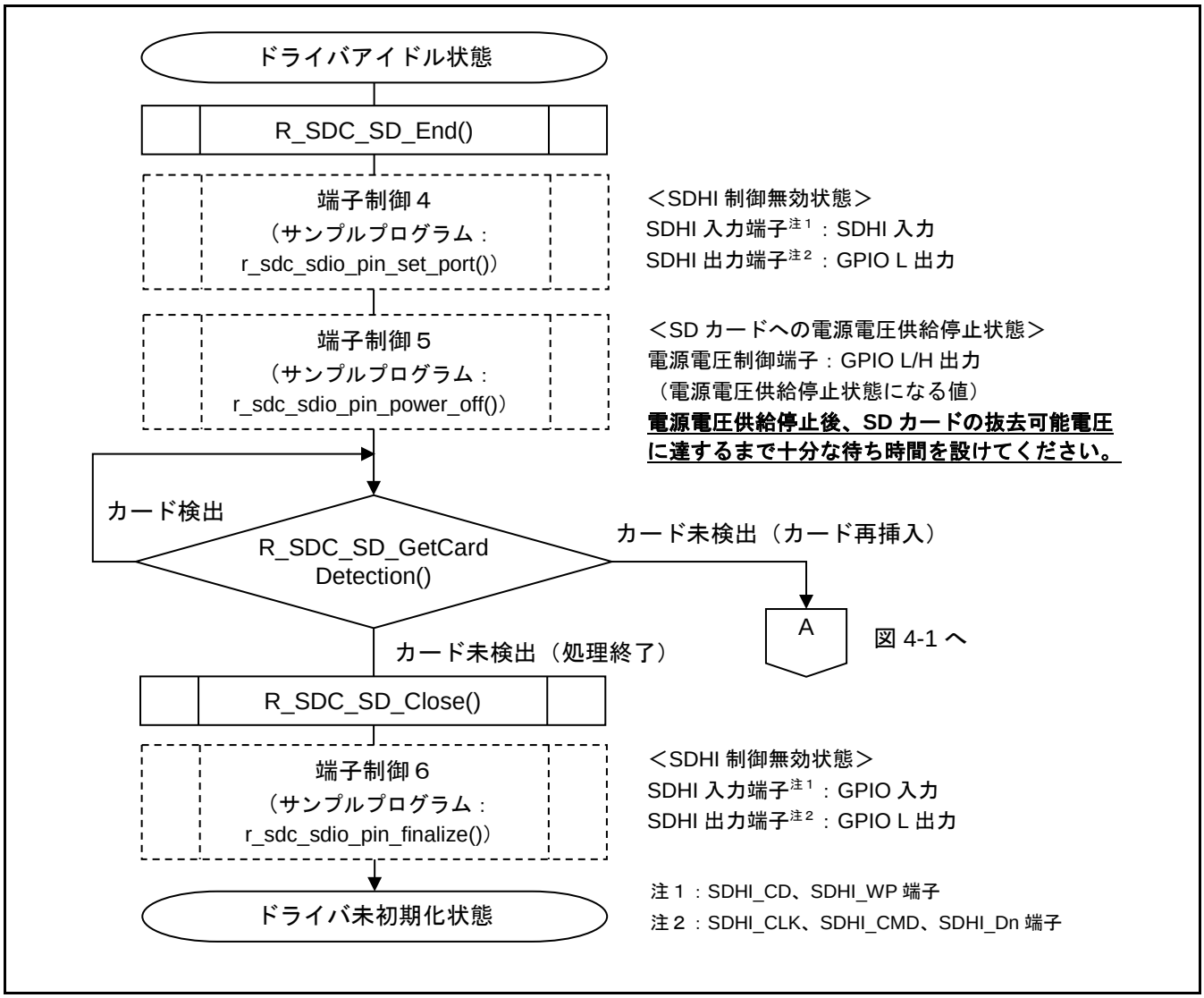


図 4-2　SD カードの抜去と電源停止タイミング

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 4	SDHI 入力端子　注 1	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	GPIO L 出力
端子制御 5	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 6	SDHI 入力端子　注 1	PMR 設定：汎用入出力ポート	GPIO 入力
	SDHI 出力端子　注 2	PMR 設定：汎用入出力ポート	GPIO L 出力

注 1：SDHI_CD、SDHI_WP 端子

注 2：SDHI_CLK、SDHI_CMD、SDHI_Dn 端子

4.1.3　データバッファと SD カード上のデータとの関係

SDHI 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 4-3 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 4-3　転送データの格納

4.1.4　初期化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。

4.1.5　割り込み処理内からコール可能な API

表 4-3 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 4-3　割り込みハンドラ内からのコールを許可する SDHI ドライバ・ライブラリ関数一覧

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）

4.1.6　SDHI_CLK の停止

SDHI ドライバは、消費電力を下げるためにライブラリ関数実行中のみ SDHI_CLK を出力し、ライブラリ関数終了時に SDHI_CLK 出力を停止します。
なお、SDHI_CLK 出力停止状態であっても、SDIO 割り込み要求を受け付け可能です。

4.1.7　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、SDHI ドライバのライブラリ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

SDHI 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 SDIO 割り込み

表 4-4 に SDHI ドライバのライブラリ関数で確認するステータスを示します。

表 4-4　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
SDHI SDIO 割り込み （R_SDC_SD_Initialize()関数で割り込み許可設定）	SDIO Interrupt 要求	SDIO Interrupt により発生
	EXPUB52 ステータス	割り込み要求受付設定（制御無しのため、発生要因無し）
	EXWT ステータス	割り込み要求受付設定（制御無しのため、発生要因無し）

(2)　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 プロトコルのステータス確認も割り込み（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。

(3)　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 カード挿抜時の割り込み発生時にコールバック関数の実行も可能です。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。

(4)　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 4-4 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。

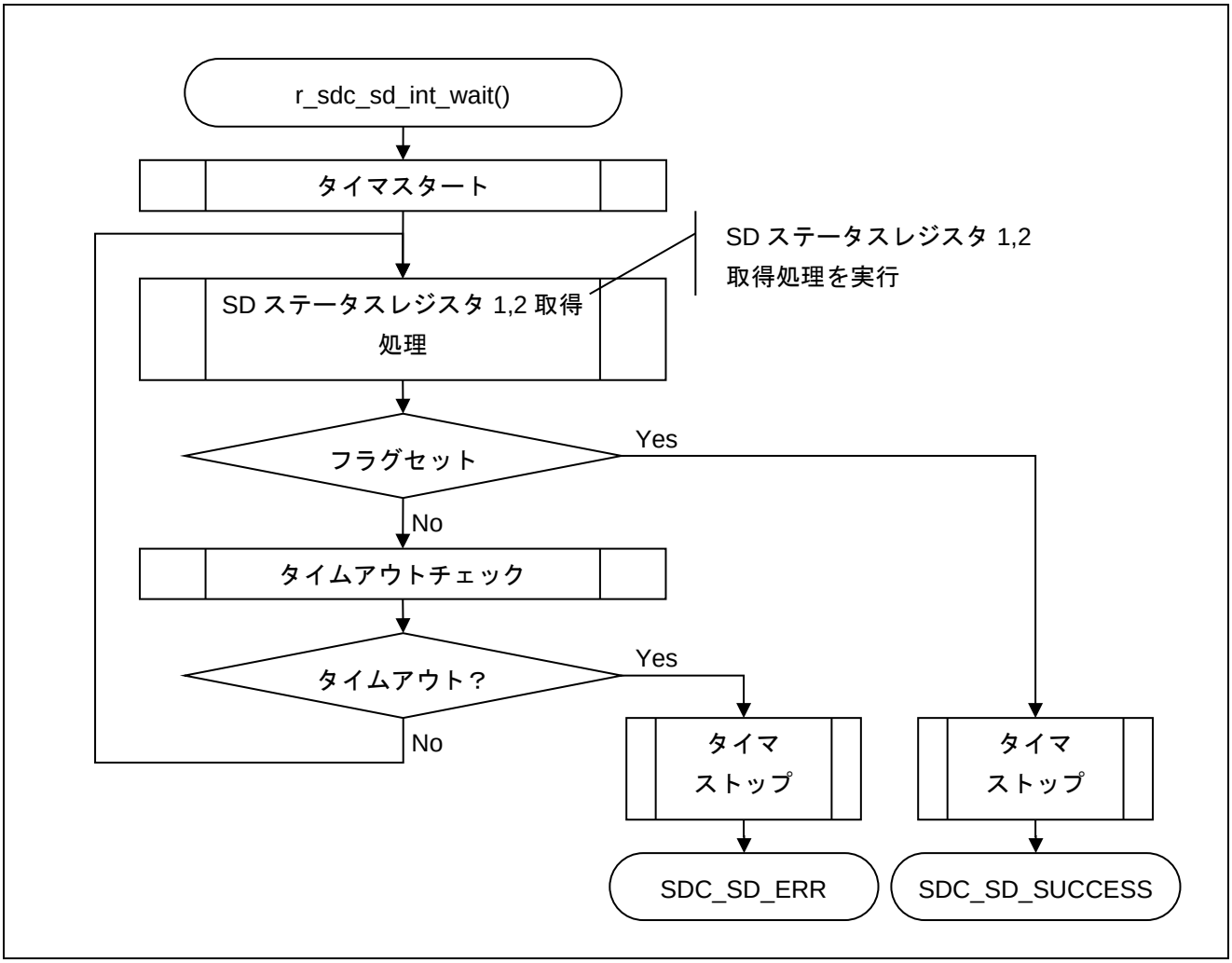


図 4-4　ソフトウェアポーリングによる SD プロトコルステータス確認

(5)　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 4-5 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。

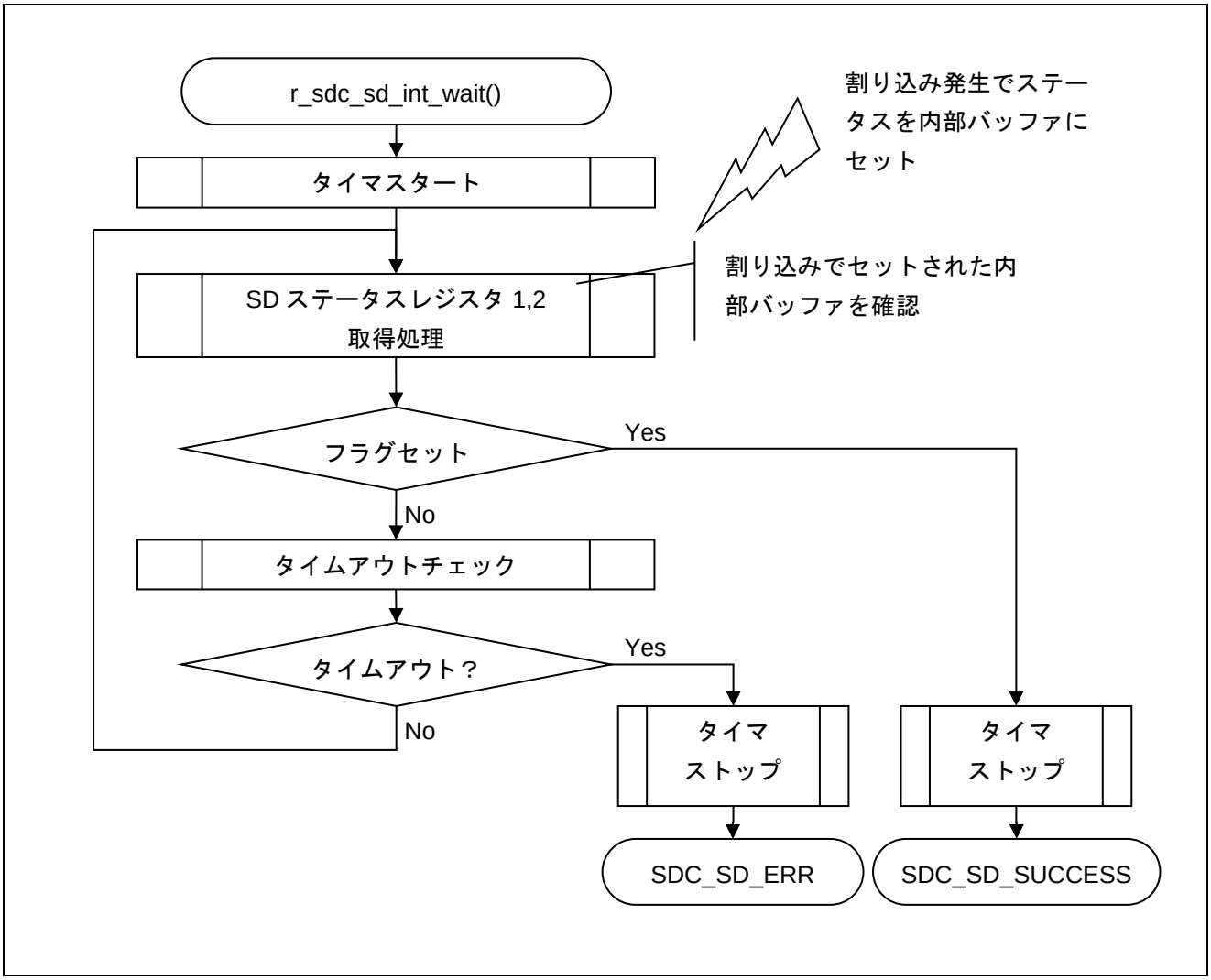


図 4-5　割り込みによる SD プロトコルステータス確認例

4.2　エラー時の制御

4.2.1　エラー発生時の処理方法

CMD53 を使ったリード処理／ライト処理等でエラーが発生した場合、SDIO ドライバは I/O Abort 処理を実行します。そのため、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 Specifications Part E1 SDIO Specification に基づき、上位のアプリケーションでの以下のエラーリカバリ処理を推奨します。

- 1：I/O Abort（エラー発生時、SDIO ドライバ内で I/O Abort 処理を実行）
- 2：I/O Enable のリセット
- 3：I/O Reset
- 4：電源供給停止と再供給

電源供給停止と再供給による再初期化を行う場合、「4.1.1」「4.1.2」を参照してください。

4.2.2　Transfer State (TRN)遷移後のエラー終了処理

Transfer State (TRN)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、I/O Abort 処理を実行します。I/O Abort 処理の実行は、Command State (CMD)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、I/O Abort 処理が実行された際、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。

4.2.3　エラーログ取得方法

前提として、パラメータチェック機能とエラーログ取得機能を共に有効にしたデバッグ用モジュールを、別途入手する必要があります。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_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「3.1 (27) R_SDC_SD_SetLogHdlAddress()」を参照してください。

1. R_LONGQ_Open()
2. R_SDC_SD_SetLogHdlAddress()

(3)　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「3.1 (28) R_SDC_SD_Log()」を参照してください。

4.3　他モジュールの制御

4.3.1　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。

4.3.2　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。

SDHI 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。

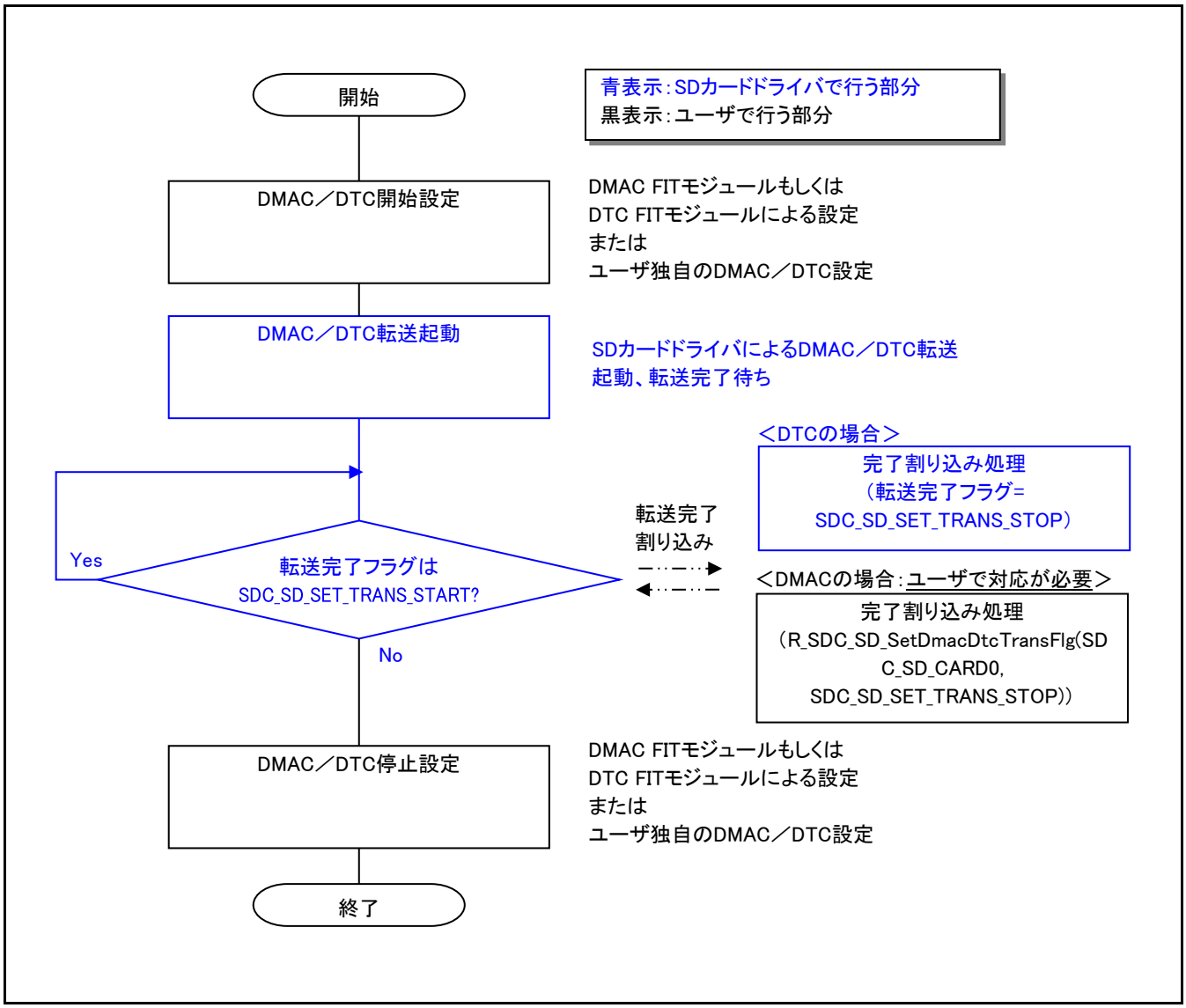


図 4-6　DMAC 転送および DTC 転送設定時の処理

4.4　待ち処理の 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 4-5　ターゲット MCU インタフェース関数一覧

関数名	機能概要	備考
r_sdc_sd_int_wait()	ステータス割り込み待ち処理（コマンド発行）	
r_sdc_sd_int_io_wait()	ステータス割り込み待ち処理（SDIO 制御）	
r_sdc_sd_int_err_io_wait()	ステータス割り込み待ち処理（SDIO 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ないます。
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_io_wait()関数／r_sdc_sd_int_err_io_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 4-7 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。

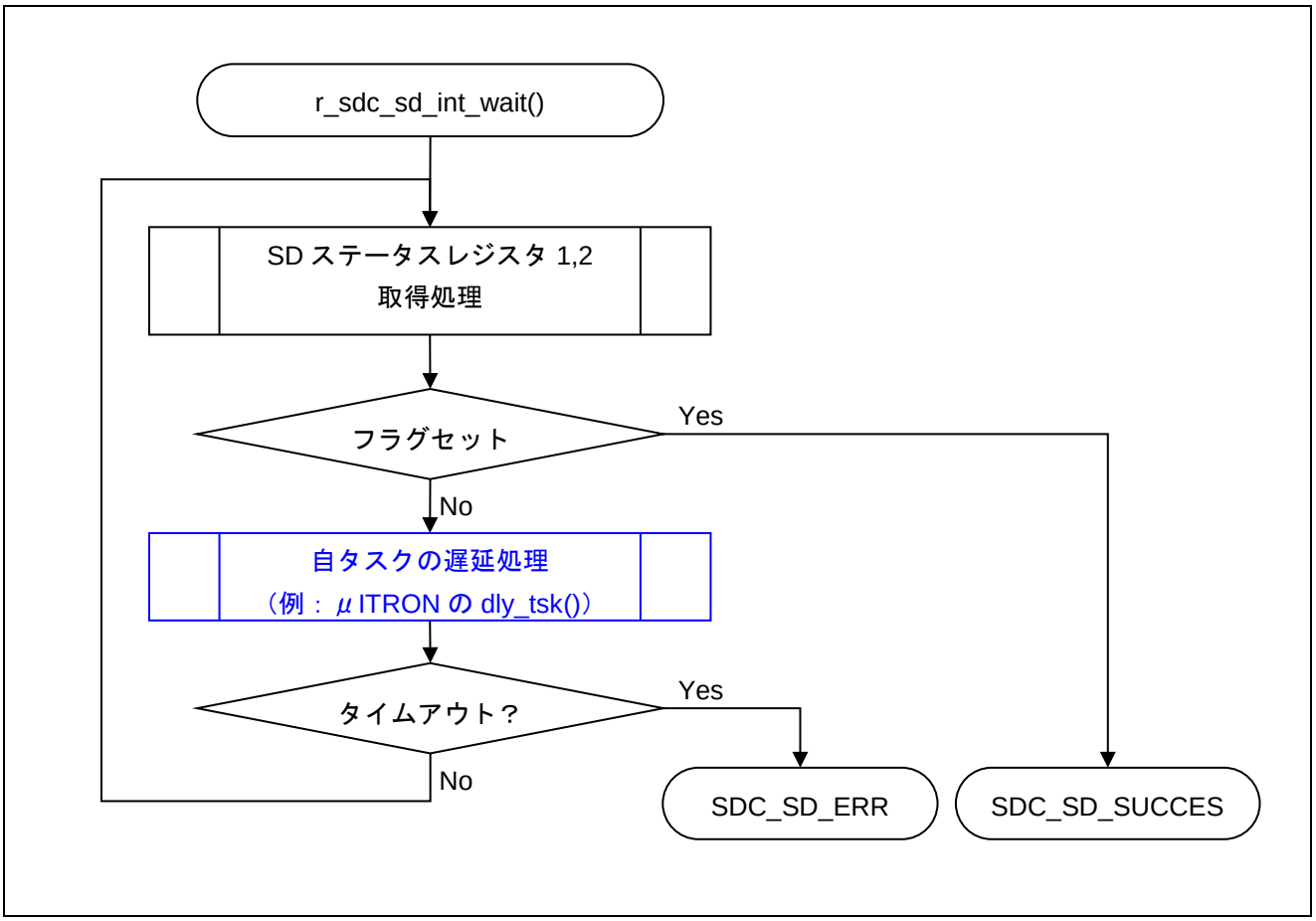


図 4-7　OS の自タスク遅延処理を使った SD プロトコルステータス確認例

以下の図 4-8 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。

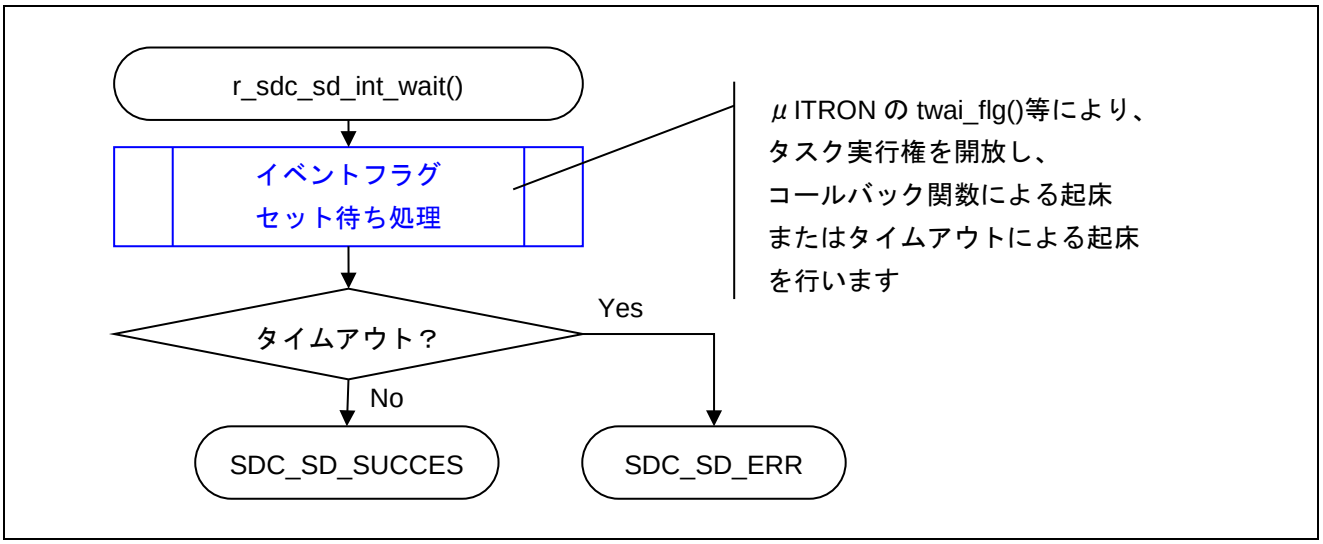


図 4-8　OS のウェイトタスク処理を使った SD プロトコルステータス確認例

(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

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

Special Notes:

表 4-6 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 4-6　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） SDIO の場合、5ms 間隔で CMD5 を発行し、最大 200 回繰り返す。

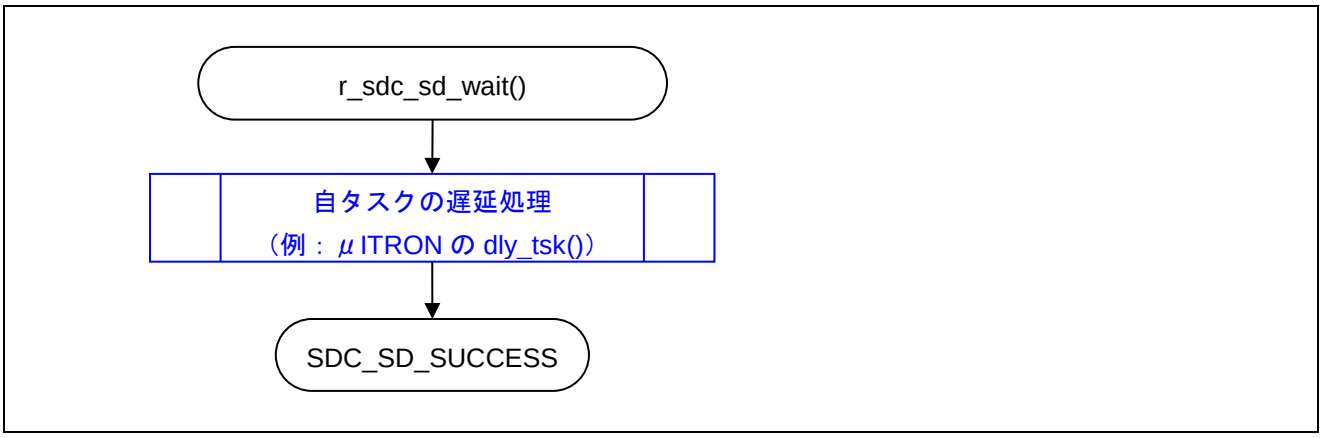


図 4-9　OS の自タスク遅延処理を使った時間待ち例

5.　ハードウェア設定

MCU 内蔵 SDHI を使って、1 ビット／4 ビットバスの SD モード制御を行います。
接続可能な SDIO モジュールは、1 台／チャネルです。

5.1　ハードウェア構成例

図 5-1 に接続図を示します。
SDIO モジュールの仕様書を参照し、システムに合った回路を検討してください。
プルアップ抵抗値は、SD Specifications Part 1 Physical Layer Specification 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。

5.1.1　端子説明

(a)　SDHI_CLK 端子

プルアップ処理は、SD Specifications Part 1 Physical Layer Specification に規定が無いため、記載していません。
SDHI_CLK－GND 間に、10-22pF 用のコンデンサパターンを設けることを推奨します。必要に応じて実装してください。

(b)　SDIO モジュール用電源電圧制御端子

SDIO モジュール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。

(c)　RESET 制御端子（SDIO モジュールの Reset 端子）

RESET（L アクティブ）を制御するために、RESET－GND 間に容量の大きいコンデンサを取り付ける場合、MCU の電気的特性（出力 Low レベル許容電流）を参照し、MCU 端子－コンデンサ間に電流制限抵抗を入れてください。RESET の入力特性（SDIO モジュール内にプルダウン抵抗を内蔵）も考慮し、抵抗値を検討してください。また、接続例では、電源供給停止時のコンデンサの電荷の高速放電目的のため、整流用ダイオードを設けています。

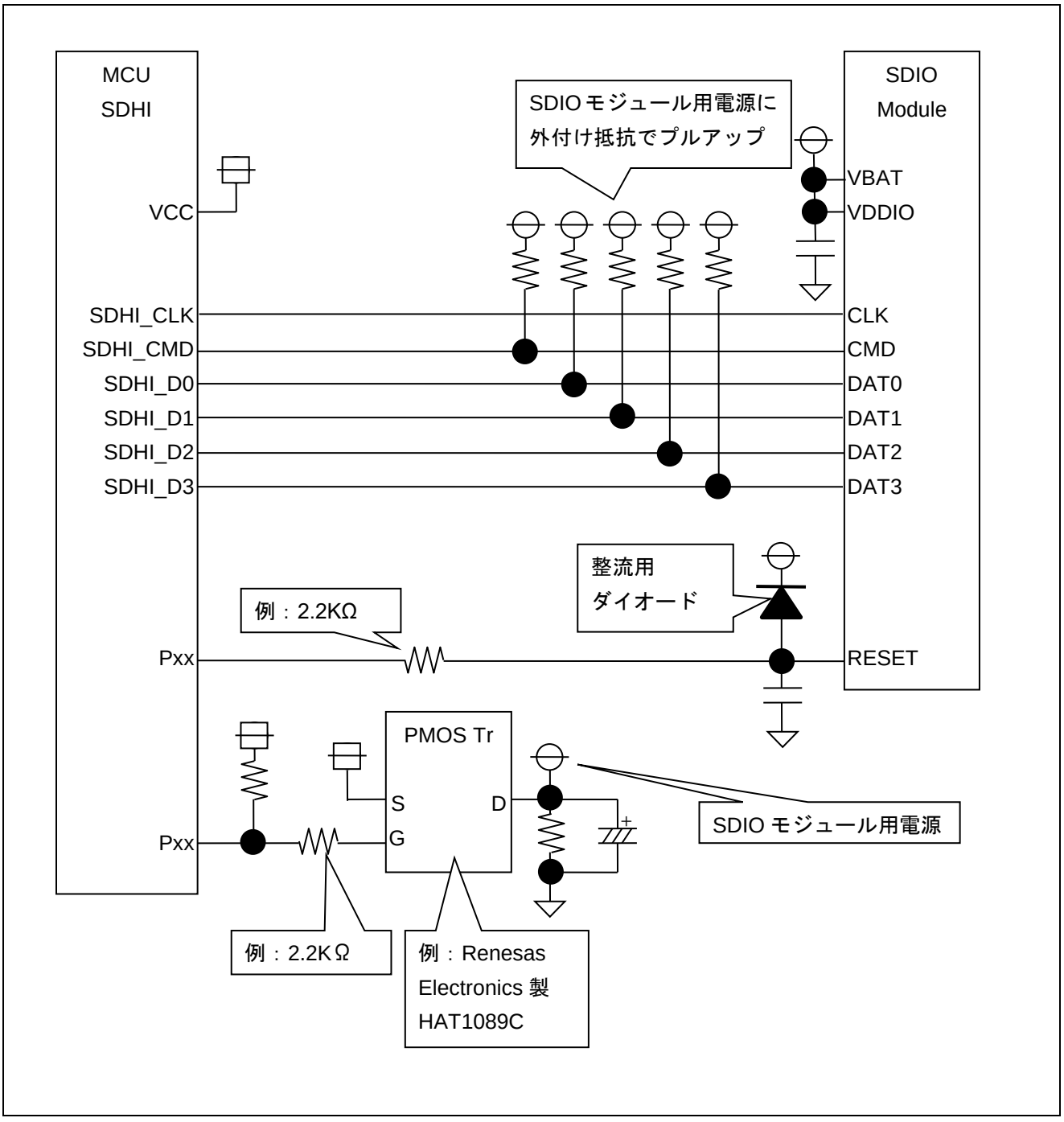


図 5-1　MCU と SDIO モジュールの接続例

5.2　MCU リソース

SD カードドライバは、以下の MCU リソースを使用します。

表 5-1　使用端子と機能

使用するリソース	入出力	内容
SDHI_CLK　注１	出力	SD クロック出力（必須）
SDHI_CMD　注１	入出力	SD コマンド出力／レスポンス入力（必須）
SDHI_D0　注１	入出力	SD データ 0（必須）
SDHI_D1　注１、注４	入出力	SD データ 1／SDIO アクセス割り込み（必須）
SDHI_D2　注１、注４	入出力	SD データ 2／リードウェイト（必須）
SDHI_D3　注１、注２、注４	入出力	SD データ 3（必須）
SDHI_CD　注５	入力	未使用
SDHI_WP　注６	入力	未使用
Pxx（電源電圧制御ポート） （汎用入出力端子の割り当て） 注１、注３	出力	電源電圧制御出力（オプション） 1 本／SDIO モジュール 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。
Pxx（RESET 制御ポート） （汎用入出力端子の割り当て） 注１、注３、注７	出力	RESET 制御（SDIO モジュールのリセット制御）（オプション） リセットする場合は 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：SDIO モジュールの CD/DAT3 端子を使った SD カード検出機能をサポートしていません。

注３：ユーザ側で端子割り当てください。SD カードドライバの制御対象外です。別途、制御ソフトウェアが必要です。

注４：SD バス 1 ビット制御の場合も必要です。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で制御します。

注５：SDHI FIT モジュールの r_sdhi_rx_config.h で SD カード検出機能 SDHI_CFG_CHx_CD_ACTIVE を“0（無効）”にしてください。この設定により SD カード検出端子の制御は無効になるため、SDHI 以外の周辺機能として使用できます。

注６：SDHI FIT モジュールの r_sdhi_rx_config.h でライトプロテクト検出機能 SDHI_CFG_CHx_WP_ACTIVE を“0（無効）”にしてください。この設定によりライトプロテクト検出端子の制御は無効になるため、SDHI 以外の周辺機能として使用できます。

注７：制御方法は、SDIO モジュールメーカに問い合わせてください。

6.　アプリケーション作成時の注意事項

6.1　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。
- API の引数として、ドライバのオープン処理 R_SDC_SD_Open()関数で設定した SD カード番号以外の SD カード番号を指定しないでください。
  製品組み込み用モジュールを使用し、各 API の引数として、未設定の SD カード番号を指定した場合、意図しないアドレスへのアクセスが発生する場合があります。
  デバッグ用モジュールでは、チャネルチェック機能も含むパラメータチェック機能があるため、初期の開発段階では、デバッグ用モジュールを使うことを推奨します。

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。

正しい電源投入切断処理が行われていない場合、SD 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、初期化処理 R_SDC_SD_Initialize()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。

6.3　miniSD と microSD を使用する場合の注意事項

miniSD のピン数は、11 ピンです。microSD のピン数は、8 ピンです。

従来の SD メモリカードのピン配置との比較を以下に示します。ピン番号を間違えないように設計してください。

ピン番号	スタンダード SD（SD モード）
#1	CD/DAT3
#2	CMD
#3	VSS
#4	VDD
#5	CLK
#6	VSS
#7	DAT0
#8	DAT1
#9	DAT2

ピン番号	miniSD（SD モード）
#1	CD/DAT3
#2	CMD
#3	VSS
#4	VDD
#5	CLK
#6	VSS
#7	DAT0
#8	DAT1
#9	DAT2
#10	-
#11	-

ピン番号	microSD（SD モード）
#1	DAT2
#2	CD/DAT3
#3	CMD
#4	VDD
#5	CLK
#6	VSS
#7	DAT0
#8	DAT1
#9	-
#10	-

6.4　コールバック関数登録処理の追加方法

SD カードドライバは、SDHI FIT モジュールのコールバック関数登録処理を用いて、コールバック関数の登録を行います。但し、登録処理を行うのは R_SDC_SD_Open()関数の引数 card_no（SD カード番号）に 0,1（SDC_SD_CARD_NO0, SDC_SD_CARD_NO1）を設定した場合のみです。複数の SD カードを制御する場合は、ユーザによりコールバック関数登録処理を追加してください。以下に、SD カード番号 2 用の処理を追加する例を示します。

(1)　コールバック関数をコピーし、リネームする。

r_sdc_sc_config.c にある r_sdc_sd_callback0()関数、r_sdc_sd_dtc_callback0()関数、r_sdc_sdio_callback0 関数をコピーしてください。そして、関数名の最後を”0”→”2”に変更してください。

(2)　SD カード番号の置換

r_sdc_sd_callback2()関数、r_sdc_sd_dtc_callback2()関数、r_sdc_sdio_callback2 関数にある SDC_SD_CARD_NO0 を SDC_SD_CARD_NO2 に置換してください。

(3)　コールバック関数登録処理の追加

r_sdc_sd_set_int_callback()の if (SDC_SD_CARD_NO0 == card_no)の if 分岐にリネームした関数を登録する処理を追加してください。

6.5　RSK の SD カードソケットを使った SD カード評価方法

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。

6.5.1　ハードウェア設定

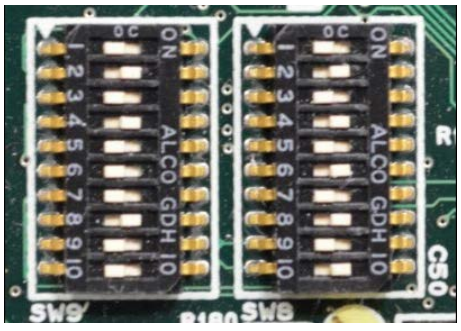
ターゲット MCU の RSK 毎に設定が必要です。

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。

(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



(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 RX231

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

6.5.2　　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を格納する。
3. RXファミリe² 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_sdc_sdio3_rx、r_sdhi_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_sdc_sdio_rx_demo_main（注１）をプロジェクトのsrcフォルダに格納する。
5. 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「6.6.1 コンパイル時の設定」を参照。

注１：製品パッケージ内の FITDemos フォルダに同梱しています。サンプルプログラムの詳細は「6.6 サンプルプログラムについて」を参照してください。

6.6　サンプルプログラムについて

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、「4.1.1 SD カードの挿入と電源投入タイミング」、「4.1.2 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。

6.6.1　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_sdc_sdio_rx_pin_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_sdc_sdio_rx_pin_config.h*	
#define SDC_SD_CFG_MODE_SW (1) #define SDC_SD_CFG_MODE_DMAC (0) #define SDC_SD_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 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 SDC_SD_CFG_RESET_PORT_NONE ※デフォルト値は“有効”	SD モジュールのリセット端子制御が不要な場合、定義を有効にしてください。 SD モジュールのリセット端子制御が必要な場合、定義を無効にしてください。
#define SDC_SD_CFG_RESET_CONTROL (2) ※デフォルト値は“2”	SD モジュールのリセット端子制御を有効に設定した場合に、設定する定義です。 定義する値により、以下に示すポート制御を行います。 ＜SD モジュールのリセット端子の状態＞

値	リセット有効	リセット無効
0	L 出力	H 出力
1	H 出力	L 出力
2	Hi-z	L 出力

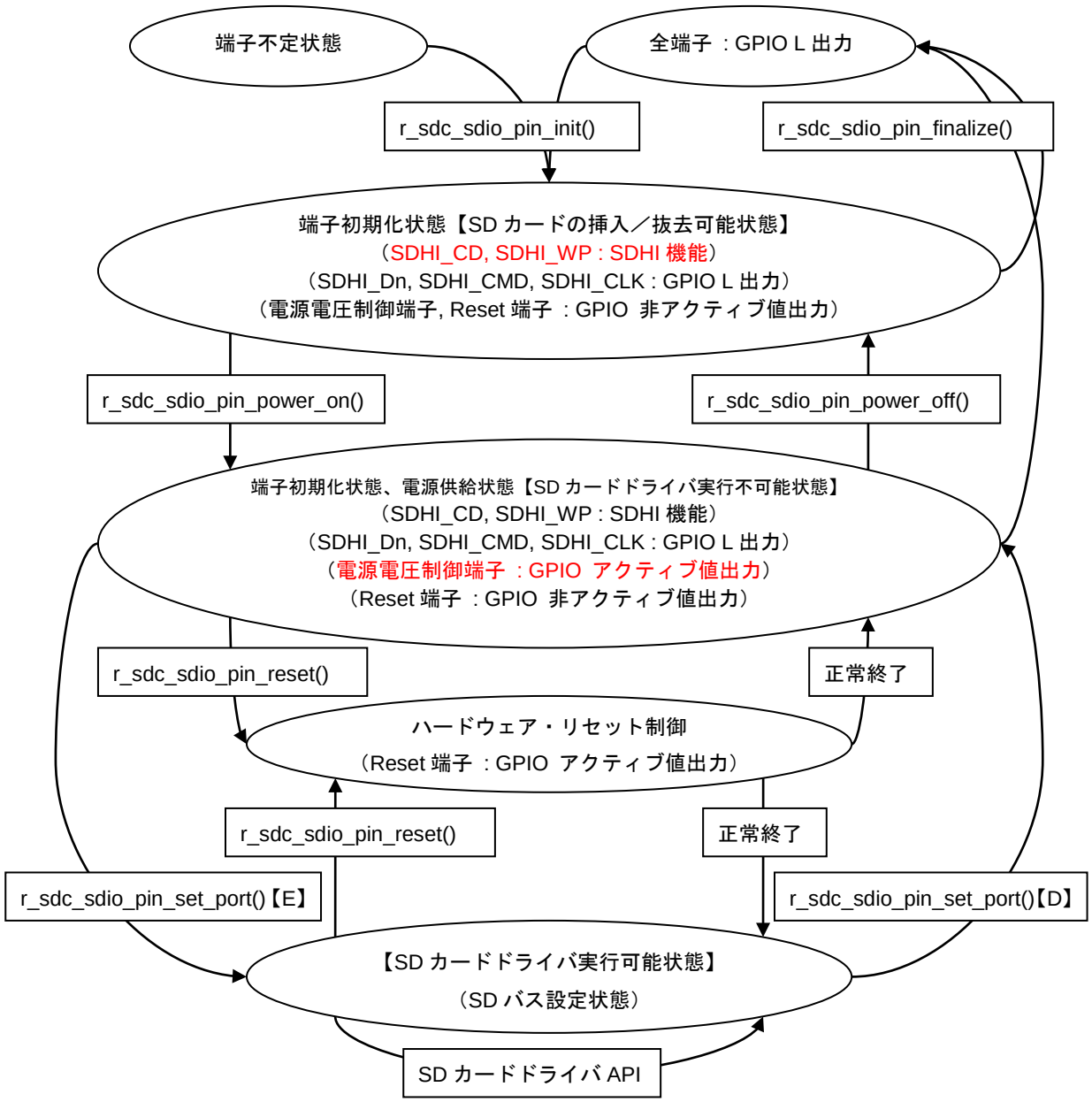
#define SDC_SD_CFG_RESET_ON_WAIT (100) ※デフォルト値は“100（100ms ウェイト）”	SD モジュールのリセット端子により、リセットした後、リセット状態を維持するウェイト時間を設定してください。1 カウントあたり、1ms のウェイトを行います。 システムに合わせて設定してください。
#define SDC_SD_CFG_RESET_OFF_WAIT (100) ※デフォルト値は“100（100ms ウェイト）”	SD モジュールのリセット端子により、リセット解除後のウェイト時間を設定してください。単純な時間待ちを行います。1 カウントあたり、1ms 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 R_SDC_SD_CFG_xxx_CARDx_PORT ※”xxx”は端子名（CD, WP, D0, D1, D2, D3, CMD, CLK, POWER, RESET） ※CARDx の“x”は SD カード番号（x>=0）	使用する MCU の各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_xxx_CARD0_BIT ※”xxx”は端子名（CD, WP, D0, D1, D2, D3, CMD, CLK, POWER, RESET） ※CARDx の“x”は SD カード番号（x>=0）	使用する MCU の各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。

6.6.2　端子制御用 API 関数

サンプルプログラム内の端子制御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 6-1　端子制御用 API 関数一覧

関数名	機能概要
r_sdc_sdio_pin_init()	端子設定の初期化処理
r_sdc_sdio_pin_power_on()	電源電圧の供給開始処理
r_sdc_sdio_pin_power_off()	電源電圧の供給停止処理
r_sdc_sdio_pin_reset()	ハードウェア・リセット制御処理
r_sdc_sdio_pin_set_port()	端子機能切り替え処理
r_sdc_sdio_pin_finalize()	端子設定の終了処理
r_sdc_sdio_pin_softwaredelay()	時間待ち処理



注 1 : 【E】は SDHI ドライバ実行可能状態への遷移用、【D】は SDHI ドライバ実行不可能状態への遷移用を示します。6.6.2(5) r_sdc_sdio_pin_set_port()を参照してください。

図 6-1　状態遷移図

(1)　r_sdc_sdio_pin_init()

SD カードドライバで使用する SDHI 端子設定を初期化する関数です。また、SDIO モジュールの電源電圧制御端子と Reset 端子の設定を初期化する関数です。

Format

```
sdc_sd_status_t r_sdc_sdio_pin_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

SDIO モジュールの電源電圧制御端子と Reset 端子の設定を初期化します。

Special Notes:

SD カードまたは SDIO モジュールの制御に必要な端子設定の初期化を目的とします。必要に応じて端子を追加／修正してください。

SDHI 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。
- 駆動能力制御レジスタ（DSCR）を”1”（高駆動出力）に設定します。ただし、本機能を未サポートの端子は設定しません。（注１）
- 駆動能力制御レジスタ２（DSCR2）を”1”（高速インタフェース用高駆動出力）に設定します。ただし、本機能を未サポートの端子は設定しません。（注１）
- 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。
- ポート出力データレジスタ（PODR）を Low 出力に設定します。
- ポート方向レジスタ（PDR）を設定します。
- マルチファンクションピンコントローラ（MPC）を SDHI 機能に設定します。
- SDHI_CD 端子および SDHI_WP 端子のポートモードレジスタ（PMR）を周辺モジュール機能に設定します。その結果、これらの端子は SDHI 機能状態になります。
- SDHI_Dn 端子、SDHI_CMD 端子および SDHI_CLK 端子のポートモードレジスタ（PMR）を汎用入出力（GPIO）機能に設定します。その結果、これらの端子は GPIO 機能状態になり、L を出力します。

電源電圧制御端子と Reset 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。

注１：下表にDSCRレジスタとDSCR2レジスタによる駆動能力設定を示します。高速なSDクロック周波数を前提としている為、サポートMCUのデフォルト設定は高駆動出力もしくは高速インタフェース用高駆動出力としています。過剰出力の場合は、DSCRとDSCR2を”0”に設定し、通常駆動出力で動作させてください。

DSCR2	DCSR	駆動能力	サポート MCU のデフォルト設定
0	0	通常駆動出力	-
1	0	高駆動出力	RX64M、RX71M、RX231、RX65N
1	Don’t care	高速インタフェース用高駆動出力	

(2)　r_sdc_sdio_pin_power_on()

SDIO モジュール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_sdio_pin_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

SDIO モジュール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_sdio_rx_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、sdc_sdio_pin_softwaredelay()関数を実行します。待ち時間は「6.6.1 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_sdio_pin_init()関数による初期化処理が必要です。

(3)　r_sdc_sdio_pin_power_off()

SDIO モジュール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_sdio_pin_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

SDIO モジュール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_sdio_rx_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、sdc_sdio_pin_softwaredelay()関数を実行します。待ち時間は「6.6.1 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_sdio_pin_init()関数による初期化処理が必要です。

(4)　r_sdc_sdio_pin_reset()

SDIO モジュールの Reset 端子を制御し、ハードウェア・リセットを実行する関数です。

Format

```
sdc_sd_status_t r_sdc_sdio_pin_reset(
    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

SDIO モジュールの Reset 端子を制御します。r_sdc_sdio_rx_pin_config.h の SDC_SD_CFG_RESET_ON_WAIT で設定された時間のハードウェア・リセット状態を維持します。その後、r_sdc_sdio_rx_pin_config.h の SDC_SD_CFG_RESET_OFF_WAIT で設定された時間のハードウェア・リセット解除状態を維持し、結果を返します。

ハードウェア・リセット状態の端子出力状態とハードウェア・リセット解除状態の端子出力状態の設定については、r_sdc_sdio_rx_pin_config.hのSDC_SD_CFG_RESET_CONTROLを使って設定可能です。

Special Notes:

ハードウェア・リセット状態の維持とハードウェア・リセット解除状態の維持のため、sdc_sd_pin_softwaredelay()関数を実行します。r_sdc_sdio_rx_pin_config.h の SDC_SD_CFG_RESET_ON_WAIT と SDC_SD_CFG_RESET_OFF_WAIT を設定してください。

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

(5)　r_sdc_sdio_pin_set_port()

SD カードドライバで使用する SDHI 端子を設定する関数です。

Format

```
sdc_sd_status_t r_sdc_sdio_pin_set_port(
    uint32_t card_no,
    int32_t port
)
```

Parameters

card_no

SD カード番号　　使用する SD カード番号（0 起算）

port

SD バス幅の設定

表 6-2　設定可能な port 値

マクロ定義	SDHI_CD	SDHI_WP	SDHI_Dn	SDHI_CMD	SDHI_CLK	備考
SDC_SD_PORT_1BIT	SDHI	SDHI	SDHI（注 1）	SDHI	SDHI	
SDC_SD_PORT_4BIT	SDHI	SDHI	SDHI	SDHI	SDHI	
SDC_SD_PORT_CD	SDHI	SDHI	GPIO	GPIO	GPIO	

注 1：SD バス 1 ビット制御の場合も必要です。SPI モードへの遷移禁止制御目的で制御します。

Return Values

SDC_SD_SUCCESS　　*正常終了*

Description

SD カードドライバが使用する SDHI_CD，SDHI_WP，SDHI_Dn，SDHI_CMD，SDHI_CLK 端子のポートモードレジスタ（PMR）を設定します。

Special Notes:

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

(6)　r_sdc_sdio_pin_finalize()

SD カードドライバで使用する SDHI 端子を初期値に設定する関数です。

Format

```
sdc_sd_status_t r_sdc_sdio_pin_finalize(
    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 カードドライバが使用する SDHI_CD，SDHI_WP，SDHI_Dn，SDHI_CMD，SDHI_CLK 端子を汎用入出力ポートに設定します。

Special Notes:

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。
マルチファンクションピンコントローラ（MPC）を初期値 Hi-Z に設定します。
本関数実行前に r_sdc_sdio_pin_init()関数による初期化処理が必要です。
マイコンのリセット後の状態と異なります。

(7)　r_sdc_sdio_pin_softwaredelay()

時間待ちを行う際に使用する関数です。

Format

```
bool r_sdc_sdio_pin_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 6-3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 6-3　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。
SD モジュールリセット有効時のリセット状態維持時間	SD モジュールのリセット端子により、リセットした後、リセット状態を維持するウェイト時間＜100ms＞ ※待ち時間は SDC_SD_CFG_RESET_ON_WAIT で変更可能。
SD カードリセット無効時の状態安定待ち時間	SD モジュールのリセット端子により、リセット解除後のウェイト時間＜100ms＞ ※待ち時間は SDC_SD_CFG_RESET_OFF_WAIT で変更可能。

6.6.3　　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_sdio_pin_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。

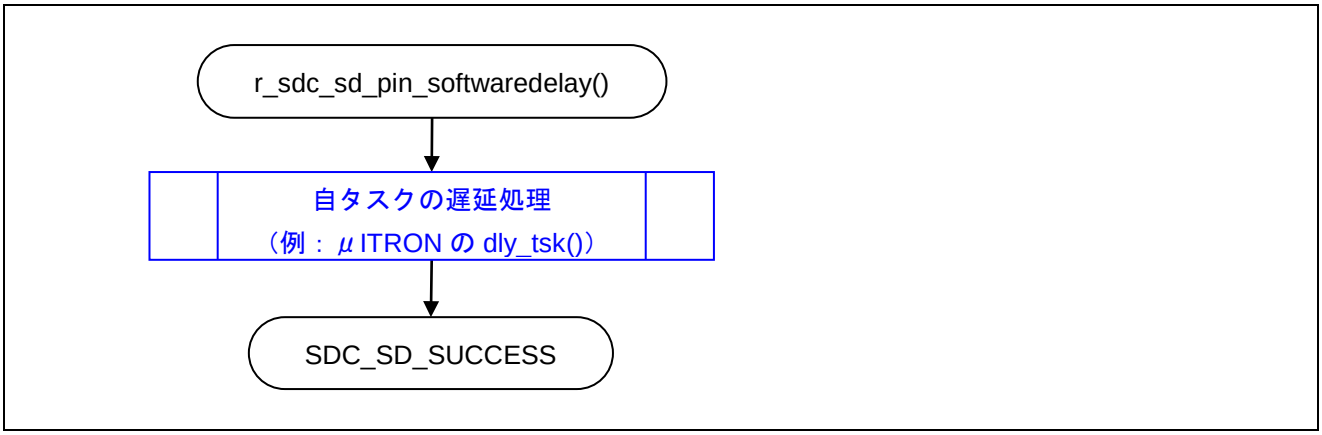


図 6-2　OS の自タスク遅延処理を使った時間待ち例

6.7　デバッグ用モジュールについて

デバッグ時にパラメータチェック機能とエラーログ取得機能を共に有効にしたデバッグ用モジュールの提供が可能です。

パラメータチェック機能を持つ関数を以下に示します。

初期評価では、デバッグ用モジュールをご利用願います。

表 6-4　ライブラリ関数一覧

関数名	機能概要	パラメータチェック有無
R_SDC_SD_Open()	ドライバのオープン処理	有り
R_SDC_SD_Close()	ドライバのクローズ処理	有り
R_SDC_SD_GetCardDetection()	挿入確認処理	有り
R_SDC_SD_Initialize()	初期化処理	有り（注１）
R_SDC_SD_End()	終了処理	有り
R_SDC_SD_SDIO_ReadDirect()	SDIO のシングルレジスタリード処理	有り（注１）
R_SDC_SD_SDIO_Read()	SDIO のレジスタリード処理	有り（注１）
R_SDC_SD_SDIO_ReadSoftwareTrans()	SDIO のレジスタリード処理（Software 転送）	有り（注１）
R_SDC_SD_SDIO_WriteDirect()	SDIO のシングルレジスタライト処理	有り（注１）
R_SDC_SD_SDIO_Write()	SDIO のレジスタライト処理	有り（注１）
R_SDC_SD_SDIO_WriteSoftwareTrans()	SDIO のレジスタライト処理（Software 転送）	有り（注１）
R_SDC_SD_SDIO_GetBlocklen()	SDIO のデータブロック長情報取得処理	有り（注１）
R_SDC_SD_SDIO_SetBlocklen()	SDIO のデータブロック長情報設定処理	有り（注１）
R_SDC_SD_Control()	ドライバのコントロール処理	有り
R_SDC_SD_GetModeStatus()	モードステータス情報取得処理	有り
R_SDC_SD_GetCardStatus()	カードステータス情報取得処理	有り
R_SDC_SD_GetCardInfo()	レジスタ情報取得処理	有り
R_SDC_SD_CdInt()	挿抜割り込み設定処理（SD カード挿抜割り込みコールバック関数登録処理を含む）	有り
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	有り
R_SDC_SDIO_IntCallback()	SDIO 割り込みコールバック関数登録処理	有り
R_SDC_SDIO_EnableInt()	SDIO 割り込み許可処理	有り
R_SDC_SDIO_DisableInt()	SDIO 割り込み禁止処理	有り
R_SDC_SDIO_GetErrCode()	ドライバのエラーコード取得処理	有り
R_SDC_SDIO_GetBuffRegAddress()	SD バッファレジスタのアドレス取得処理	有り
R_SDC_SD_1msInterval()	インターバルタイマカウント処理	無し
R_SDC_SD_SetDmacDtcTransFlg()	DMAC/DTC 転送完了フラグセット処理	有り
R_SDC_SD_SetLogHdlAddress()	LONGQ モジュールのハンドラアドレス設定処理	無し
R_SDC_SD_Log()	エラーログ取得処理	無し
R_SDC_SD_GetVersion()	ドライバのバージョン情報取得処理	無し

注１：パラメータエラー終了の場合、SDC_SD_ERR が返ります。この場合のエラーコードには前処理で発生したエラーコードが格納されています。前処理でエラーが発生していない場合、エラーコードには SDC_SD_ERR が格納されます。

表 6-5　SDIO レジスタ空間アクセス関数一覧

関数名	機能概要	パラメータチェック有無
R_SDC_SDIO_IOAbort()	I/O Abort 処理	有り
R_SDC_SDIO_SetIOEnable()	IOEx（I/O Enable）設定処理	有り
R_SDC_SDIO_ClearIOEnable ()	IOEx（I/O Enable）クリア処理	有り
R_SDC_SDIO_GetIOReady()	I/O Ready 取得処理	有り
R_SDC_SDIO_IOReset()	I/O Reset 処理	有り
R_SDC_SDIO_SetIntEnable()	IENM（Int Enable）設定処理	有り
R_SDC_SDIO_ClearIntEnable ()	IENM（Int Enable）クリア処理	有り
R_SDC_SDIO_GetIntEnable()	Int Enable 取得処理	有り
R_SDC_SDIO_GetCIS()	CIS 取得処理	有り

改訂記録	SD モード SDIO ドライバ・ソフトウェア RTM0RX0000DSDD3 ユーザーズマニュアル　ミドルウェア編

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.09.30	ー	初版発行
2.00	2017.07.31	-	ソフトウェアの階層構造を上位層と下位層に分離し、下位層のソフトウェア名称を SDHI FIT モジュールとした。
		-	ソフトウェアの名称を以下のとおりに変更した。 ・SDIO ドライバ : 元は SDHI IO ドライバであった。 ・SD カードドライバ : 元は SDHI ドライバであった。
		-	プリフィックスを「SDHI」から「SDC_SD」に変更した。
		-	6.3 FIT モジュール以外の環境で SDHI ドライバを使用する場合の組み込み方法を削除した。
		2	1.3 アプリケーション構成図　図 1-1　において、 デバイスドライバレイヤの階層を「SDIO ドライバ」と「SDHI FIT モジュール」に分割した。
		3	1.3 アプリケーション構成図 (c) SDIO ドライバ　表 1-3　において、 ライブラリファイル名は、元は「r_sdhi_sdio_rx_xxxx.lib」であった。
		4	1.4 動作環境　において、 以下のバージョンを更新した。 ・統合開発環境 ・C コンパイラ ・モジュールのバージョン ・使用ボード
		4	1.5 Lib ファイル作成時の C コンパイラ　において、 統合開発環境 e^2 studio のバージョンを更新した。
		5	1.7 メモリサイズの参考値　において、 メモリサイズを更新した。
		9	2.2 ソフトウェアの要求　において、 r_sdhi_rx を新規に追加した。
		10	2.6 コンパイル時の設定　において、 #define SDC_SD_CFG_CARD_NUM を新規に追加した。
		15	2.9 FIT モジュールの追加方法　において、 最新の内容に修正した。
		19 - 73	3.1 ライブラリ関数　において、 引数「card_no」は、元は「channel」であった。 また、「SD カード番号」は、元は「SDHI チャネル番号」であった。
		20	(1) R_SDC_SD_Open() Parameters　において、 引数「channel」を新規に追加した。 引数「*p_sdc_sd_workarea」の領域サイズは、元は「280 バイト」であった。
		20	(1) R_SDC_SD_Open() Description　において、 最新の内容に修正した。
		20	(1) R_SDC_SD_Open() Special Notes　において、 最新の内容に修正した。
		22	(3) R_SDC_SD_GetCardDetection() Special Notes　において、 最新の内容に修正した。

		25	(4) R_SDC_SD_Initialize() Special Notes　において、 最新の内容に修正した。
		26	(5) R_SDC_SD_End() Special Notes　において、 最新の内容に修正した。
		50	(18) R_SDC_SD_CdInt() Special Notes　において、 最新の内容に修正した。
		74	4.1.1 SD カードの挿入と電源投入タイミング　において、 元は「4.1.1 端子状態と SD カードの挿抜／電源投入タイミング」であった。
		76	4.1.2 SD カードの抜去と電源停止タイミング　において、 元は「4.1.1 端子状態と SD カードの挿抜／電源投入タイミング」であった。
		92 - 93	6.6 コールバック関数登録処理の追加方法 6.7 RSK の SD カードソケットを使った SD カード評価方法 新規に追加した。
		94 - 105	6.8 サンプルプログラムについて　において、 最新の内容に修正した。

SDモードSDIOドライバ・ソフトウェア
RTM0RX0000DSDD3 Ver.2.00　ユーザーズマニュアル
ミドルウェア編

発行年月日　2016年 9月30日　Rev.1.00
　　2017年 7月31日　Rev.2.00

発行　　ルネサス エレクトロニクス株式会社
　　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社 〒135-0061 東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/

RX ファミリ

ルネサス エレクトロニクス株式会社

R01UW0134JJ0200