

RZ/A2M グループ

簡易版 SD メモリカードドライバ：導入ガイド

要旨

本資料は、簡易版 SD メモリカードドライバを導入するための情報を提供します。

RZ/A2 用簡易版 SD メモリカードドライバ（以下 SD ドライバとします）は、ルネサス 32 ビット RISC マイクロコンピュータ RZ/A2M グループ用のソフトウェアライブラリです。SD ドライバは、FAT ファイルシステムと組み合わせることにより SD メモリカードおよび MMC カードに対するファイル操作が可能になります。

動作確認デバイス

RZ/A2M

本資料の表記規則

本資料は、特に説明のない限り以下の表記規則に示す用語を使用して説明しています。

表 1 記号

表記	意味
数字	マニュアル上に指定がない限り 10 進数を意味します。
0x	マニュアル上に指定がない限り 16 進数を意味します。
0b	マニュアル上に指定がない限り 2 進数を意味します。

表 2 用語

表記	意味
FAT	File Allocation Table を意味します。
exFAT	Extended FAT を意味します。
SD メモリカード	Secure Digital Memory Card を意味します。
MMC カード	Multi Media Card を意味します。
カード	SD メモリカードおよび MMC カードを意味します。
Default-Speed カード	PHYSICAL LAYER SPECIFICATION Ver1.10 で規定された SD クロック最大 25MHz に対応した SD メモリカードを意味します。
High-Capacity カード	PHYSICAL LAYER SPECIFICATION Ver2.00 で規定された 2GB（最大 32GB）を超えるメモリ容量を持つ SD メモリカードを意味します。
Standard-Capacity カード	2GB 以下のメモリ容量を持つ SD メモリカードを意味します。特に、PHYSICAL LAYER SPECIFICATION Ver1.01 と Ver1.10 に準拠する SD メモリカードは全て Standard-Capacity カードに属します。
eXtended-Capacity カード	PHYSICAL LAYER SPECIFICATION Ver3.00 で規定された 32GB(最大 2TB)を超えるメモリ容量を持つ SD メモリカードを意味します。

目次

1. SD ドライバ概要	6
1.1 機能概要	6
1.2 プログラム開発手順	7
2. ソフトウェア構成	8
3. アプリケーション開発手順	9
3.1 ライブラリ関数一覧	9
3.2 アプリケーション開発手順	10
3.2.1 概要	10
3.2.2 デバイスドライバ関数の作成	10
3.2.3 ターゲット CPU インタフェース関数の作成	11
3.3 ライブラリ関数の使用するメモリ	12
3.3.1 ライブラリ関数のワーク領域とバッファ領域	12
3.3.2 ライブラリ関数のワークおよびバッファ領域の指定	12
3.4 ステータス確認方式	13
3.4.1 ステータス確認方式	13
3.4.2 カード挿抜検出	13
3.4.3 割り込みによる SD プロトコル制御ステータス確認	14
3.4.4 ポーリングによる SD プロトコル制御ステータス確認	15
3.5 SDCLK 決定方法	16
3.5.1 クロック分周比	16
3.5.2 クロックの停止	16
3.6 セクタデータ転送方法	16
3.6.1 ソフトウェアによる転送方法	16
3.6.2 DMA による転送方法	16
3.7 High-Capacity カードおよび eXtended-Capacity カード対応	17
3.7.1 High-Capacity カードおよび eXtended-Capacity カード対応の選択	17
3.7.2 動作モードの取得	17
3.8 データ転送処理の高速化	18
3.8.1 カードのフォーマット形式	18
3.8.2 データ転送サイズ (SD メモリカード)	18
3.9 カードマウント	19
3.10 エラーコード	20
4. 関数リファレンス	22
4.1 関数リファレンスの読み方	22
4.2 ライブラリ関数	23
4.2.1 sd_init	24
4.2.2 sd_finalize	26
4.2.3 sd_set_buffer	27
4.2.4 sd_cd_int	28
4.2.5 sd_check_media	30

4.2.6	sd_set_seccnt.....	31
4.2.7	sd_get_seccnt	32
4.2.8	sd_mount.....	33
4.2.9	sd_unmount.....	36
4.2.10	sd_inactive.....	37
4.2.11	sd_read_sect.....	38
4.2.12	sd_write_sect.....	39
4.2.13	sd_get_type	41
4.2.14	sd_get_size	43
4.2.15	sd_iswp.....	44
4.2.16	sd_stop	45
4.2.17	sd_set_intcallback	46
4.2.18	sd_int_handler.....	47
4.2.19	sd_check_int.....	48
4.2.20	sd_get_reg.....	49
4.2.21	sd_get_rca.....	52
4.2.22	sd_get_sdstatus	53
4.2.23	sd_get_error	54
4.2.24	sd_set_cdtime	55
4.2.25	sd_set_responsetime	56
4.2.26	sd_get_ver.....	57
4.2.27	sd_lock_unlock	58
4.2.28	sd_get_speed.....	60
4.3	ターゲット CPU インタフェース関数.....	62
4.3.1	sddev_init	63
4.3.2	sddev_finalize.....	64
4.3.3	sddev_power_on	65
4.3.4	sddev_power_off	66
4.3.5	sddev_read_data.....	67
4.3.6	sddev_write_data	69
4.3.7	sddev_get_clockdiv	71
4.3.8	sddev_set_port.....	73
4.3.9	sddev_int_wait.....	74
4.3.10	sddev_loc_cpu.....	75
4.3.11	sddev_unl_cpu	76
4.3.12	sddev_init_dma	77
4.3.13	sddev_wait_dma_end	78
4.3.14	sddev_disable_dma	79
4.3.15	sddev_reset_dma.....	80
4.3.16	sddev_finalize_dma.....	81
4.4	デバイスドライバ関数.....	82
4.4.1	disk_status.....	83
4.4.2	disk_initialize	84
4.4.3	disk_read.....	87
4.4.4	disk_write.....	88
4.4.5	disk_ioctl.....	89
4.4.6	get_fattime.....	90

5. コンフィグオプション	91
6. アプリケーション作成時の制限事項	92
6.1 SD ドライバ使用時の注意事項	92
7. サンプルプログラム	93
7.1 機能概要	93
7.2 動作環境	94
7.3 動作確認条件	95
7.4 使用端子と機能	97
7.5 メモリサイズ	99
7.6 導入手順	100
7.7 注意事項	104
8. スマートコンフィグレータによるコンポーネント追加手順	105
8.1 コンポーネント追加	105
8.2 コンフィグ設定	107
8.2.1 SD カード検出オプション設定	107
8.2.2 ライトプロテクト信号検出オプション設定	108
8.2.3 SD カード検出コールバック関数設定	109
8.3 端子設定	110
8.3.1 CD 端子、WP 端子の設定	110
8.3.2 PD_1 端子 (SDVcc_SEL) の設定	111
8.3.3 PJ_1 端子 (SW3 キー入力) の設定	112
8.3.4 PC_1 端子 (LED1 (Yellowish-green)) の設定	113
8.4 コード生成	114
9. 参考ドキュメント	115
改訂記録	116

1. SD ドライバ概要

本章では、簡易版 SD ドライバソフトウェアライブラリの概要について説明します。

1.1 機能概要

SD ドライバは、SD メモリカードにアクセスするためのライブラリ関数群で構成されています。SD ドライバの特徴を次に示します。

■ SD ドライバの特徴

- ・ ファイルシステムと組み合わせることを前提としたコンパクトなソフトウェア構成
- ・ 4M バイト～2T バイトまでの SD メモリカードに対応
- ・ 4M バイト～2G バイトまでの MMC カードに対応
- ・ データ転送方法として、ソフトウェア転送または DMA 転送を選択可能
- ・ CPU 依存部分をターゲット CPU インタフェース関数として分離
- ・ High-Capacity カード対応
- ・ eXtended-Capacity カード対応
- ・ 省メモリ構成
- ・ バススピードモードは、default speed 固定

1.2 プログラム開発手順

SD ドライバを使用したアプリケーションプログラムの開発フローを図 1.1 に示します。SD ドライバを使用したアプリケーションを開発するためには、別途 FAT ファイルシステムが必要です。また、図中のデバイスドライバソースファイル、ターゲット CPU インタフェースソースファイルを作成する必要があります。デバイスドライバ関数およびターゲット CPU インタフェース関数の詳細については、第 3 章で説明しています。

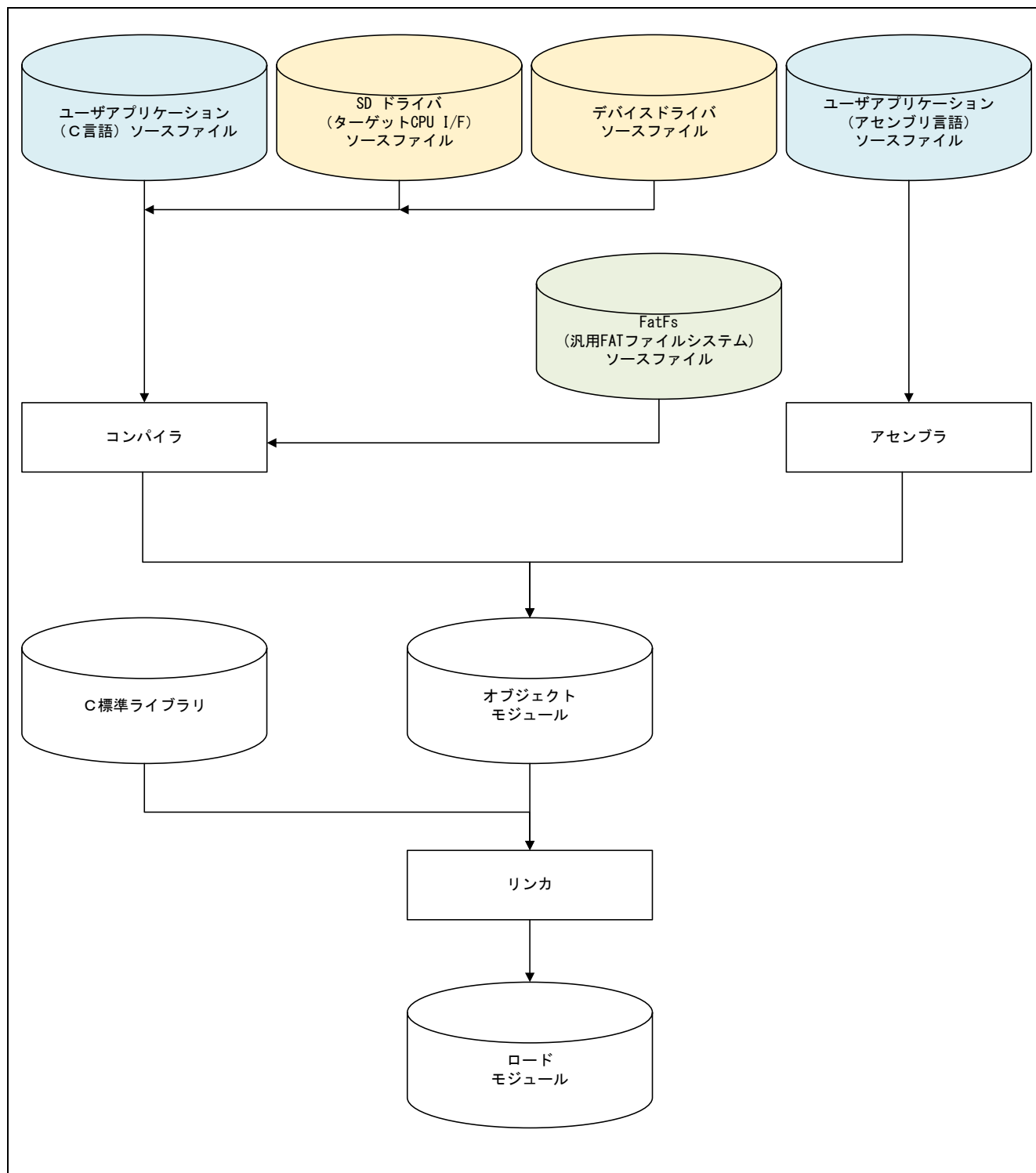


図 1.1 アプリケーションプログラム開発フロー

2. ソフトウェア構成

図 2.1 に、SD ドライバのソフトウェア構成を示します。

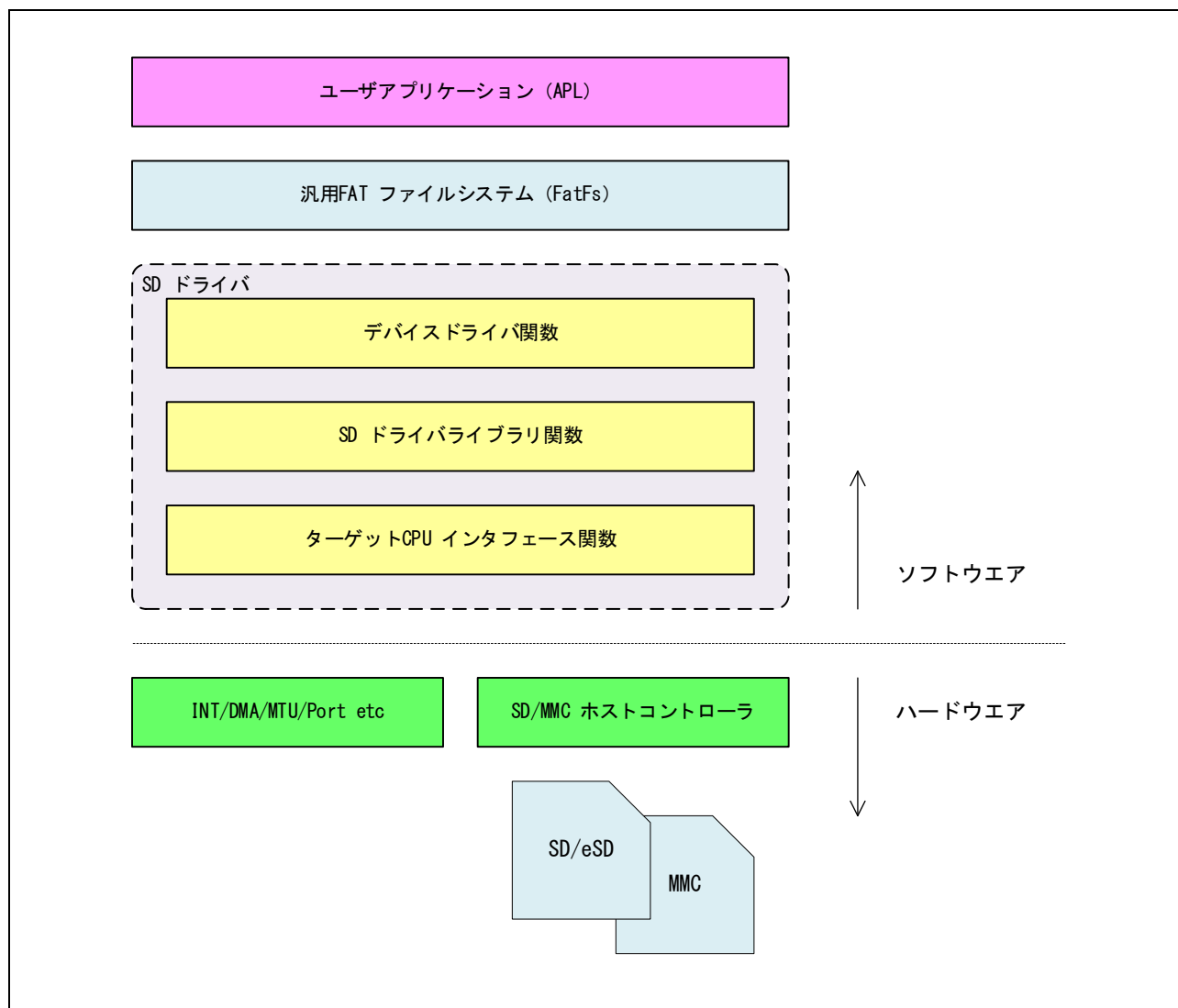


図 2.1 ソフトウェア構成図

3. アプリケーション開発手順

本章では、SD ドライバを使用したアプリケーションプログラムの開発手順やシステムへの組み込み方法について説明しています。

3.1 ライブラリ関数一覧

表 3.1 に SD ドライバのライブラリ関数を示します。

表 3.1 SD ドライバライブラリ関数一覧

関数名	機能概要
sd_init	ドライバの初期化
sd_finalize	ドライバの終了
sd_set_buffer	ライブラリ用バッファ領域の設定
sd_cd_int	カード挿抜割り込み設定
sd_check_media	カードの挿入確認
sd_set_seccnt	連続転送セクタ数の設定
sd_get_seccnt	連続転送セクタ数の取得
sd_mount	カードのマウント
sd_unmount	カードのマウントの解除
sd_inactive	カードの無効化
sd_read_sect	カードからのセクタリード
sd_write_sect	カードへのセクタライト
sd_get_type	カードタイプと動作モードの取得
sd_get_size	カードサイズの取得
sd_iswp	ライトプロテクト状態の取得
sd_stop	カード処理の強制停止
sd_set_intcallback	プロトコルステータス確認割り込みコールバック関数の登録
sd_int_handler	カード割り込みハンドラ
sd_check_int	カード割り込み要求確認
sd_get_reg	カードレジスタの取得
sd_get_rca	RCA レジスタの取得
sd_get_sdstatus	SD STATUS の取得
sd_get_error	ドライバエラーの取得
sd_set_cdtime	カード検出時間の設定
sd_set_responsetime	レスポンスタイムアウト時間の設定
sd_get_ver	ライブラリバージョンの取得
sd_lock_unlock	カードのロック・アンロック
sd_get_speed	カードスピードの取得

3.2 アプリケーション開発手順

本節ではターゲットシステムに SD ドライバを使用して SD メモリカード用 FAT ファイルシステムを構築する場合のアプリケーション開発手順について説明します。

3.2.1 概要

SD ドライバを使用して FAT ファイルシステムを構築するには、応用システムに依存する次のプログラムを作成する必要があります。

■ デバイスドライバ関数

使用する FAT ファイルシステムソフトウェアのデバイスドライバとして SD ドライバを組み込むための関数です。デバイスの初期化、リード、及びライト関数などをデバイスドライバ関数として作成します。デバイスドライバ関数の詳細に関しては「4.4 デバイスドライバ関数」をご参照ください。

■ ターゲット CPU インタフェース関数

ターゲットとなる CPU に SD ドライバを組み込むための関数です。割り込みコントローラの設定やデータ転送用の関数などを作成します。ターゲット CPU インタフェース関数の詳細に関しては「4.3 ターゲット CPU インタフェース関数」をご参照ください。

3.2.2 デバイスドライバ関数の作成

デバイスドライバ関数は、SD ドライバを FAT ファイルシステムに組み込むための関数です。デバイスドライバ関数内で、SD ドライバの動作モードの設定や SD ドライバのライブラリ関数が使用するワーク領域の確保を行います。

表 3.2 に FatFs（汎用 FAT ファイルシステム）（注）に SD ドライバを組み込む場合の SD ドライバのデバイスドライバ関数一覧を示します。他のファイルシステムに組み込む場合は、使用するファイルシステムの仕様に合わせてデバイスドライバ関数を作成してください。

デバイスドライバ関数の仕様の詳細は「4.4 デバイスドライバ関数」をご参照ください。

注：eXtended-Capacity カードを使用した FAT ファイルシステムを構築する場合は、exFAT に対応した FAT ファイルシステムが必要です。

表 3.2 デバイスドライバ関数一覧

分類	関数名	機能概要
デバイスドライバ関数	disk_status	デバイスの状態取得
	disk_initialize	デバイスの初期化
	disk_read	セクタデータの読み出し（論理セクタ単位）
	disk_write	セクタデータの書き込み（論理セクタ単位）
	disk_ioctl	その他のデバイス制御
	get_fattime	日付と時刻の取得

3.2.3 ターゲット CPU インタフェース関数の作成

ターゲット CPU インタフェース関数は、SD ドライバのライブラリ関数が使用する CPU の内蔵資源とのインタフェースを行う関数です。ポート制御や割り込みコントローラの設定、タイマの設定など行います。ターゲット CPU インタフェース関数は SD ドライバのライブラリ関数から呼び出されます。

表 3.3 に SD ドライバのターゲット CPU インタフェース関数一覧を示します。

ターゲット CPU インタフェース関数の仕様の詳細は「4.3 ターゲット CPU インタフェース関数」をご参照ください。

表 3.3 ターゲット CPU インタフェース関数一覧

分類	関数名	機能概要
ターゲット CPU インタフェース関数	sddev_init	H/W の初期化
	sddev_finalize	H/W の終了処理
	sddev_power_on	カードへの電源供給開始
	sddev_power_off	カードへの電源供給停止
	sddev_read_data	データリード処理
	sddev_write_data	データライト処理
	sddev_get_clockdiv	クロック分周比の取得
	sddev_set_port	カード用ポート設定
	sddev_int_wait	カード割り込み待ち
	sddev_loc_cpu	カード割り込みの禁止
	sddev_unl_cpu	カード割り込みの許可
	sddev_init_dma	データ転送用 DMA の初期化
	sddev_wait_dma_end	データ転送用 DMA 転送完了待ち
	sddev_disable_dma	データ転送用 DMA の禁止
	sddev_reset_dma	DMA のリセット
	sddev_finalize_dma	DMA の終了処理

3.3 ライブラリ関数の使用するメモリ

本節では、ライブラリ関数の使用するメモリ領域の定義方法、初期化の方法について説明します。

3.3.1 ライブラリ関数のワーク領域とバッファ領域

SD ドライバのライブラリ関数が使用するワーク領域は、アプリケーションにて確保する必要があります。アプリケーションにて確保したワーク領域をライブラリ関数の初期化関数に設定することにより SD ドライバライブラリ関数が動作します。

ワーク領域として表 3.4 に示すマクロ定義サイズ分の領域が必要です。ワーク領域は、8 バイト境界に配置してください。

ライブラリ関数のワーク領域の確保方法は、動的確保、静的確保のいずれでも構いませんが、SD ドライバのドライバ関数の初期化から終了までワーク領域を保持してください。また、ユーザプログラムから直接ライブラリ関数のワーク領域の内容を変更しないでください。ユーザプログラムによりワーク領域の内容を変更した場合の動作は保証しません。

図 3.1 にワーク領域定義例を示します。

また、ライブラリ関数内部で使用するカードとのデータ受け渡し用バッファ領域をアプリケーションにて確保する必要があります。バッファ領域は最小 512 バイトで、512 バイト単位でバッファ領域設定関数により設定します。バッファ領域は、マウント時のレジスタリードおよびフォーマット時の初期化データ領域として使用します

表 3.4 ライブラリ関数のワーク領域用マクロ定義

マクロ定義	機能概要
SD_SIZE_OF_INIT	ライブラリ関数のワーク領域サイズ (バイト)

```
/* SD ワーク領域定義 */
uint32_t sd_driver_work[SD_SIZE_OF_INIT/sizeof(uint32_t)];
```

図 3.1 ライブラリ関数のワーク領域定義例

3.3.2 ライブラリ関数のワークおよびバッファ領域の指定

ライブラリ関数のワーク領域指定は、初期化関数 sd_init 関数で行います。詳細は、sd_init 関数説明をご参照ください。また、ライブラリバッファ領域は sd_set_buffer 関数で行います。詳細は、sd_set_buffer 関数説明をご参照ください。

3.4 ステータス確認方式

カードの操作を行う上で、通信の終了検出など SD ホストコントローラのステータスの確認やカードの挿抜検出を行う必要があります。本節では、SD ドライバのライブラリ関数使用時のステータス確認方式について説明します。

3.4.1 ステータス確認方式

SD ドライバでは、SD ホストコントローラのステータス確認方式として、SD ホストコントローラ割り込み（以降、H/W 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。また、確認するステータスの種類としてカード挿抜検出と SD プロトコル制御があります。表 3.5 に SD ドライバのライブラリ関数にて確認するステータスを示します。

表 3.5 確認するステータス

分類	ステータス	備考
カード挿抜検出	カードの挿入	
	カードの抜取	
SD プロトコル制御	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 バイトまたはブロックサイズ転送毎に発生
	プロトコルエラー	CRC エラーなど発生時
	タイムアウトエラー	レスポンス応答なしなど発生時

sd_mount 関数により SD プロトコル制御のステータス確認方式の設定を行います。また、カード挿抜検出のステータス確認方式は、ライブラリ関数 sd_cd_int 関数にて設定を行います。設定方法の詳細については、各関数の関数説明をご参照ください。

ステータス確認方式に割り込みを選択した場合は、SD ホストコントローラ割り込みに対応する割り込みハンドラとして、ライブラリ関数 sd_int_handler 関数をシステムに登録する必要があります。また、SD ホストコントローラ内の割り込み設定はライブラリ関数にて行いますが、CPU の割り込みコントローラの設定など SD ホストコントローラ割り込みを有効にする処理は、ターゲット CPU インタフェース関数内で設定する必要があります。

注：カード挿抜検出のステータス確認方式に H/W 割り込みを選択した場合は、SD プロトコル制御のステータス確認方式も H/W 割り込みを選択してください。

3.4.2 カード挿抜検出

カード挿抜検出のステータス確認方式にソフトウェアポーリングを指定した場合は、ライブラリ関数 sd_check_media 関数にてカードの挿入確認が行えます。

カード挿抜検出のステータス確認方式に割り込みを設定した場合は、ライブラリ関数 sd_check_media 関数によるカードの挿入確認と、カード挿抜時の割り込み発生時にユーザ指定の関数をコールバック関数として呼び出すことが可能です。カードの挿抜割り込み発生時にユーザ指定の関数を呼び出しますので、カードの挿抜に対するリアルタイムの処理が可能です。カード挿抜発生時に呼び出すユーザ指定の関数は、ライブラリ関数 sd_cd_int 関数にて設定を行います。

3.4.3 割り込みによる SD プロトコル制御ステータス確認

SD プロトコル制御のステータス確認方式として H/W 割り込みを指定した場合は、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待ち時間を他の処理に割り当てることが可能です。SD プロトコル制御ステータス確認方式は、sd_mount 関数にて指定を行います。また、ステータス確認の割り込み発生時、ユーザ指定の関数をコールバック関数として呼び出すことができるので、割り込み発生待ち処理に対する柔軟な対応が可能です。割り込みコールバック関数は sd_set_intcallback 関数にて登録を行います。

割り込み待ち処理は、ターゲット CPU インタフェース関数 sddev_int_wait 関数として、ユーザが作成する必要があります。図 3.2 に割り込みを使用した場合の、SD プロトコル制御ステータス確認のフローチャート例を示します。

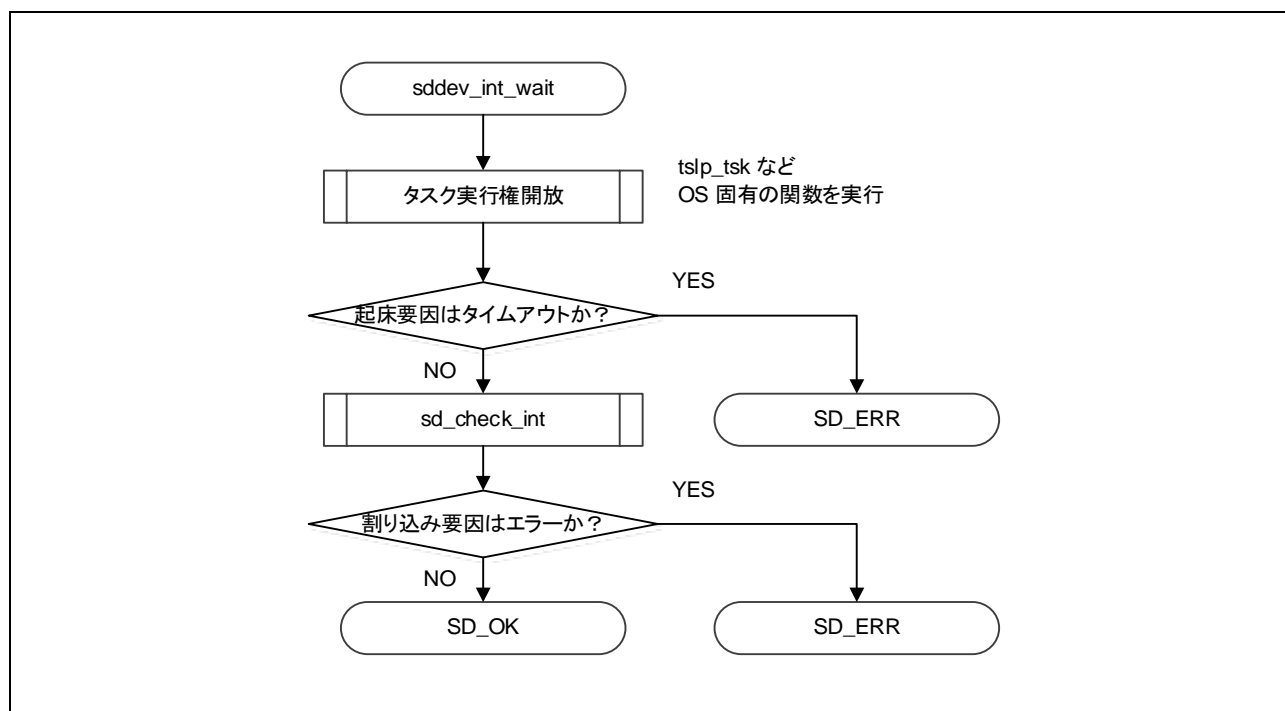


図 3.2 割り込みによるステータス確認例

3.4.4 ポーリングによる SD プロトコル制御ステータス確認

SD プロトコル制御のステータス確認方式としてソフトウェアポーリングを指定した場合は、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にて確認します。SD プロトコル制御ステータス確認方式は、sd_mount 関数にて指定を行います。

ソフトウェアポーリング時は、ライブラリ関数 sd_check_int 関数を用いてステータス変化の確認を行います。ソフトウェアポーリング処理はターゲット CPU インタフェース関数 sddev_int_wait 関数内で行います。図 3.3 にソフトウェアポーリングを使用した場合の、SD プロトコル制御ステータス確認のフローチャート例を示します。

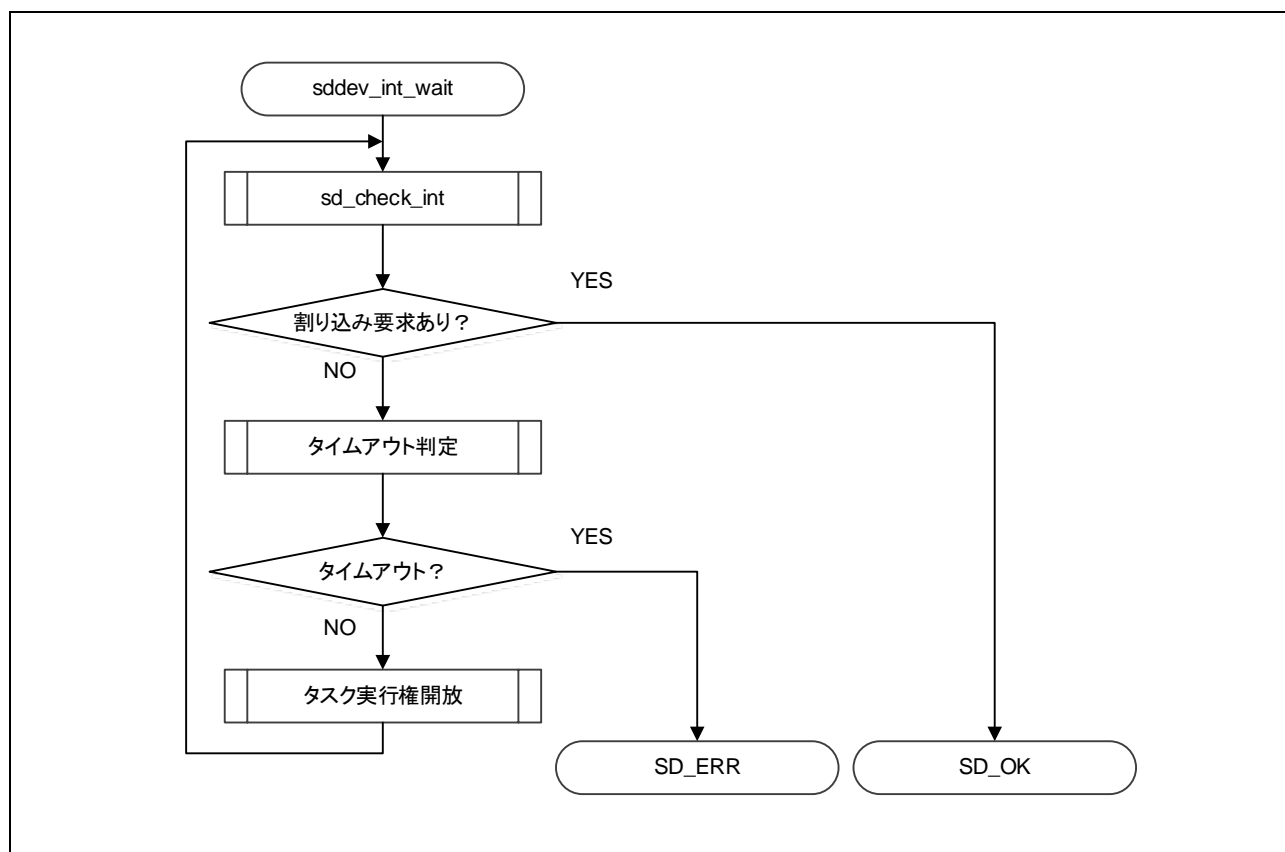


図 3.3 ソフトウェアポーリングによるステータス確認例

3.5 SDCLK 決定方法

カードに供給する SDCLK は、SD ホストコントローラ IP に供給するクロックを分周して出力します。ここでは、SD ホストコントローラから出力する SDCLK について説明します。

3.5.1 クロック分周比

SD ホストコントローラに供給するクロックはシステムにより異なるため、ターゲット CPU インタフェース関数 `sddev_get_clockdiv` 関数にて SDCLK として出力するクロック周波数の分周比をシステムに応じて決定する必要があります。詳細は `sddev_get_clockdiv` 関数説明をご参照ください。

SDCLK の周波数は、カード認識モードでは最大 400kHz、データ転送モードでは最大 25MHz になります。ただし、データ転送モードでの最大周波数は、ライブラリ関数内にてカードの CSD レジスタ内容から決定し、`sddev_get_clockdiv` 関数の引数として指定します。`sddev_get_clockdiv` 関数では、ライブラリ関数から指定された周波数を超えないように SDCLK の分周比を決定してください。

3.5.2 クロックの停止

SD ドライバでは、カードの消費電力を下げるためにライブラリ関数実行中のみ SDCLK を出力し、ライブラリ関数終了時 SDCLK 出力を停止します。

3.6 セクタデータ転送方法

SD メモリカードとのセクタデータ転送方法としてソフトウェアによる転送もしくは DMA による転送のいずれかを選択できます。本節では、SD ドライバライブラリ関数使用時のセクタデータ転送方法について説明します。セクタデータの転送は、ライブラリ関数 `sd_read_sect` 関数および `sd_write_sect` 関数使用時に行われます。

3.6.1 ソフトウェアによる転送方法

セクタデータの転送方法としてソフトウェア転送を選択した場合は、ターゲット CPU インタフェース関数 `sddev_read_data` 関数および `sddev_write_data` 関数にてセクタデータの転送を行います。転送方法の選択はライブラリ関数 `sd_mount` 関数にて指定します。

ソフトウェア転送方法の詳細は、`sddev_read_data` 関数および `sddev_write_data` 関数の関数説明をご参照ください。

3.6.2 DMA による転送方法

セクタデータの転送方法として DMA 転送を選択した場合は、CPU の DMA コントローラを使用してセクタデータの転送を行います。

DMA 転送を選択した場合は、ライブラリ関数 `sd_read_sect` 関数および `sd_write_sect` 関数に対し指定するバッファのアドレスに、8 バイト境界のアドレスを指定してください。

ライブラリ関数 `sd_read_sect` 関数および `sd_write_sect` 関数に対し指定したバッファのアドレスが 8 バイト境界でない場合、ライブラリ関数は DMA 処理を行わずソフトウェア転送処理を行います。

DMA コントローラの設定や終了確認等は、ターゲット CPU インタフェース関数 `sddev_init_dma` 関数、`sddev_wait_dma_end` 関数、`sddev_disable_dma` 関数にて行います。設定方法の詳細については、それぞれの関数説明をご参照ください。

3.7 High-Capacity カードおよび eXtended-Capacity カード対応

本ライブラリは、SD Memory Card Specifications Part1 Ver2.00 で規定された 2GB（最大 32GB）を超えるメモリ容量を持つ High-Capacity SD メモリカードおよび SD Memory Card Specifications Part1 Ver3.00 で規定された 32GB（最大 2TB）を超えるメモリ容量を持つ eXtended-Capacity SD メモリカードに対応しています。ただし、High-Capacity カードに対応するには、SD ドライバの上位層であるファイルシステムが FAT32 をサポートしている必要があります。また、eXtended-Capacity カードに対応するには、SD ドライバの上位層であるファイルシステムが exFAT をサポートしている必要があります。

3.7.1 High-Capacity カードおよび eXtended-Capacity カード対応の選択

本ライブラリは、eXtended-Capacity カードと High-Capacity カード、Standard-Capacity カードを自動判別してマウントすることができます。また、Standard-Capacity カードのみに対応することも可能です。判別方法は `sd_mount` 関数の引数で指定します。

表 3.6 にマウント時の指定と動作モードを示します。

表 3.6 挿入されているカードと動作モード(3)

マウント時の指定	挿入されているカード		
	eXtended-Capacity カード	High-Capacity カード	Standard-Capacity カード
SD_MODE_VER2X	eXtended-Capacity モード	High-Capacity モード	Standard-Capacity モード
SD_MODE_VER1X	エラー	エラー	Standard-Capacity モード

3.7.2 動作モードの取得

ライブラリの動作モードは、ライブラリ関数 `sd_get_type` 関数で取得できます。詳細は `sd_get_type` 関数の関数説明をご参照ください。

3.8 データ転送処理の高速化

ここでは、SD ドライバを使用したアプリケーションにて、データ転送を高速に行う方法について説明します。

3.8.1 カードのフォーマット形式

SD メモリカードは、推奨された FAT フォーマット形式の場合に高速なデータ転送処理が行える仕様になっています。そのため、PC などにおいて専用のユーティリティを使用せずに再フォーマットされた SD メモリカードは、最適な FAT フォーマット形式でない場合が多く、その結果、パフォーマンスが出ない場合があります。

3.8.2 データ転送サイズ (SD メモリカード)

SD ドライバでは、`sd_set_seccnt` 関数にて `sd_write_sect` 関数および `sd_read_sect` 関数での連続転送セクタ数の最大値を設定できます。データ転送速度は、この連続転送セクタ数と `sd_write_sect` 関数および `sd_read_sect` 関数に指定するデータ転送セクタ数に依存します。

`sd_set_seccnt` 関数に設定する連続転送セクタ数は、大きいほど転送処理の効率が良くなります。ただし、連続転送セクタ数を大きくした場合、データ転送の中止処理を必要とするアプリケーションでは、`sd_stop` 関数によるデータ転送の中止の応答速度が遅くなる場合があります。連続転送セクタ数は、256 以上 256 の倍数のセクタ数を推奨します。

`sd_write_sect` 関数および `sd_read_sect` 関数では、連続転送セクタ数を基準に転送を行います。`sd_write_sect` 関数および `sd_read_sect` 関数に指定するセクタ数が、連続転送セクタ数以下の場合は、指定されたセクタ数で転送を行います。`sd_write_sect` 関数または `sd_read_sect` 関数に指定するセクタ数が、連続転送セクタ数よりも大きい場合は連続転送セクタ数単位に分割して転送を行います。

セクタデータ書き込みの場合は、その書き込みサイズによりデータ転送速度が大きく異なります。

SD メモリカードの特性から 1 クラスターのセクタ数に満たないセクタ数の書き込みが最も効率が悪くなります。最適な書き込みサイズはアプリケーションにより異なりますが、`sd_write_sect` 関数に指定するセクタ数としてなるべく大きい値を指定した方が効率良く書き込みを行います。

セクタデータ読み出しの場合も同様に、`sd_set_seccnt` 関数で設定したセクタ単位で連続読み出し処理を行います。`sd_read_sect` 関数に指定するセクタ数として、なるべく大きい値を指定した場合に最も効率良く読み出しを行います。ただし、書き込みの場合に比べ大きな速度差はありません。

注：データ転送速度は、使用する SD メモリカードにより大きく異なります。

3.9 カードマウント

本ライブラリは、接続されたカード種別を自動判別してマウントすることができます。マウント処理は、sd_mount 関数より行われます。

図 3.4 に本ライブラリのカードマウント処理概要を示します。

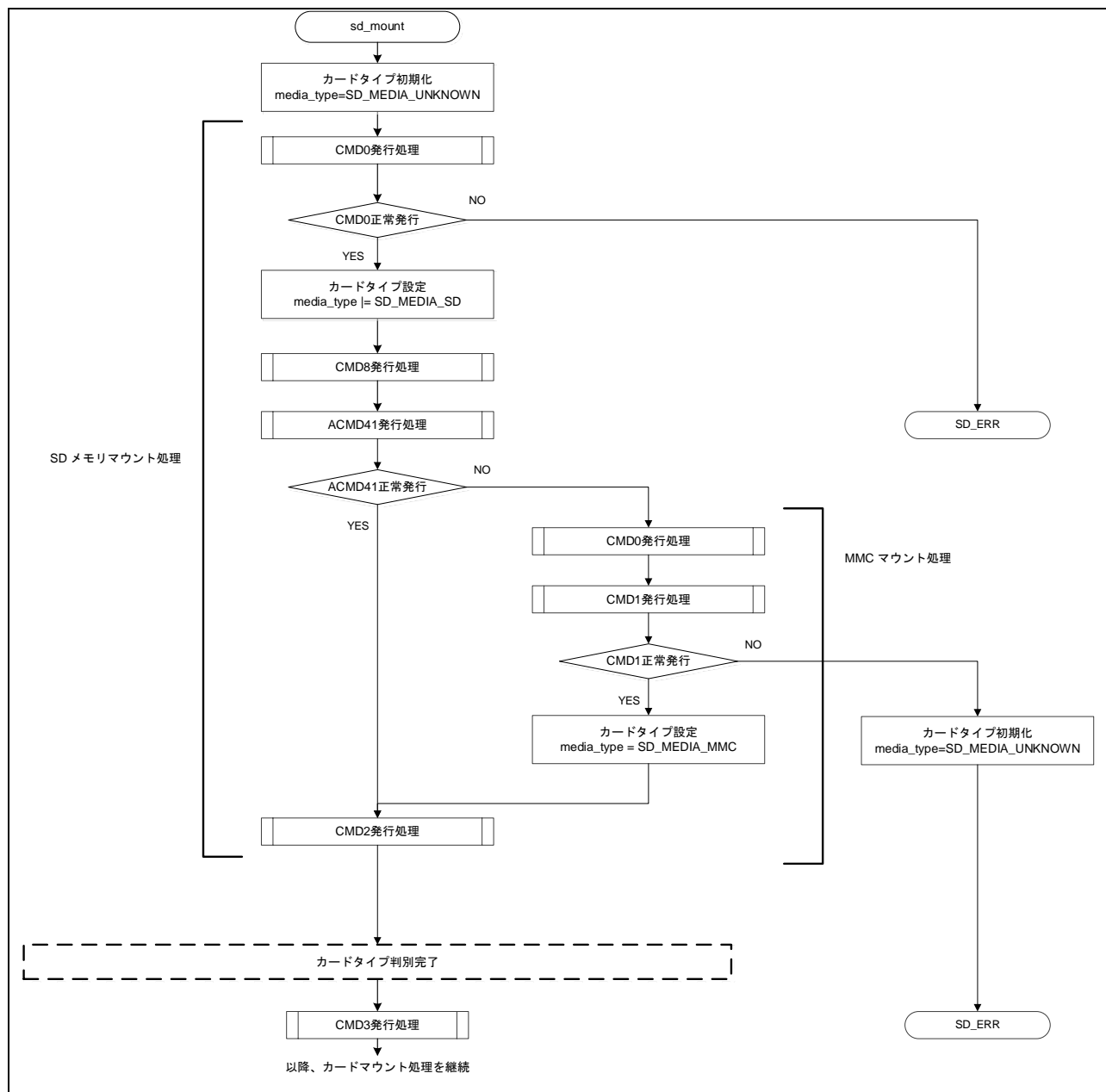


図 3.4 カードマウント処理概要

3.10 エラーコード

SD 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、戻り値にエラーを返します。表 3.7～表 3.8 に示すマクロ定義をライブラリ関数のエラーコードとして定義しています。なお、表にない値は、将来のための予約です。

ライブラリ関数 `sd_mount` 関数、`sd_read_sect` 関数および `sd_write_sect` 関数実行後は、ライブラリ関数 `sd_get_error` 関数でもエラー内容を取得することができます。他のライブラリ関数のエラー内容を `sd_get_error` 関数にて取得することはできません。

表 3.7 エラーコード(1/2)

マクロ定義	値	意味	エラー詳細
<code>SD_OK_LOCKED_CARD</code>	1	正常終了(カードロック状態)	ロック状態カードのマウントが正常終了
<code>SD_OK</code>	0	正常終了	正常終了
<code>SD_ERR</code>	-1	一般エラー	<code>sd_init</code> 関数が実行されていない、引数パラメータエラーなど
<code>SD_ERR_WP</code>	-2	ライトプロテクトエラー	ライトプロテクトさせているカードへの書き込み
<code>SD_ERR_RES_TOE</code>	-4	レスポンスタイムアウト	コマンドに対するレスポンスが 640 クロック (SDCLK) 以内に受信できなかった
<code>SD_ERR_CARD_TOE</code>	-5	カードタイムアウト	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
<code>SD_ERR_END_BIT</code>	-6	エンドビットエラー	エンドビットの検出ができなかった
<code>SD_ERR_CRC</code>	-7	CRC エラー	ホスト側での CRC エラー検出
<code>SD_ERR_HOST_TOE</code>	-9	ホストタイムアウトエラー	<code>sddev_int_wait</code> 関数のエラー
<code>SD_ERR_CARD_ERASE</code>	-10	カードイレースエラー	SD カードステータスエラー (ERASE_SEQ_ERROR or ERASE_PARAM) イレースシーケンスまたはイレースコマンドパラメータエラー
<code>SD_ERR_CARD_LOCK</code>	-11	カードロックエラー	SD カードステータスエラー (CARD_IS_LOCKED) ロックされているカードへの操作
<code>SD_ERR_CARD_UNLOCK</code>	-12	カードアンロックエラー	SD カードステータスエラー (LOCK_UNLOCK_FAILED) カードロック解除時のエラー
<code>SD_ERR_HOST_CRC</code>	-13	ホスト CRC エラー	SD カードステータスエラー (COM_CRC_ERROR) カード側での CRC エラー検出
<code>SD_ERR_CARD_ECC</code>	-14	カード ECC エラー	SD カードステータスエラー (CARD_ECC_FAILED) カード内部での ECC エラー発生
<code>SD_ERR_CARD_CC</code>	-15	カード CC エラー	SD カードステータスエラー (CC_ERROR) カード内部コントローラのエラー
<code>SD_ERR_CARD_ERROR</code>	-16	カードエラー	SD カードステータスエラー (ERROR) カード側のエラー

表 3.8 エラーコード(2/2)

マクロ定義	値	意味	エラー詳細
SD_ERR_CARD_TYPE	-17	未対応カード	未対応のカードと認識した
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
SD_ERR_ILL_READ	-19	不正読み出しエラー	sddev_read_data 関数または DMA 転送によるセクタデータ読み出し方法が不正
SD_ERR_ILL_WRITE	-20	不正書き込みエラー	sddev_write_data 関数または DMA 転送によるセクタデータ書き込み方法が不正
SD_ERR_AKE_SEQ	-21	認証プロセスシーケンスエラー	SD カードステータスエラー (AKE_SEQ_ERROR)
SD_ERR_OVERWRITE	-22	CSD オーバライトエラー	次のいずれかのエラー ・ CSD の読み取り専用セクションがカード内容と不一致 ・ copy または、permanent WP ビットを反転しようとした
SD_ERR_CPU_IF	-30	ターゲット CPU インタフェース関数エラー	ターゲット CPU インタフェース関数のエラー (sddev_int_wait 関数以外)
SD_ERR_STOP	-31	強制停止エラー	sd_stop 関数による強制停止
SD_ERR_CSD_VER	-50	CSD バージョンエラー	CSD structure version と SD Memory Card Physical Specification Version が不整合
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
SD_ERR_NOTSUP_CMD	-53	サポートされていないコマンドエラー	サポートされていないコマンドを指定した
SD_ERR_IFCOND_VER	-70	インタフェースコンディションバージョンエラー	インタフェースコンディションのバージョンが不正
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
SD_ERR_OUT_OF_RANGE	-80	引数範囲外エラー	SD カードステータスエラー (OUT_OF_RANGE)
SD_ERR_ADDRESS_ERROR	-81	アドレスエラー	SD カードステータスエラー (ADDRESS_ERROR)
SD_ERR_BLOCK_LEN_ERROR	-82	ブロック長エラー	SD カードステータスエラー (BLOCK_LEN_ERROR)
SD_ERR_ILLEGAL_COMMAND	-83	異常コマンドエラー	SD カードステータスエラー (ILLEGAL_COMMAND)
SD_ERR_RESERVED_ERROR18	-84	-	
SD_ERR_RESERVED_ERROR17	-85	-	
SD_ERR_CMD_ERROR	-86	コマンドインデックスエラー	SDIF 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
SD_ERR_CBSY_ERROR	-87	コマンドエラー	SDIF 内部エラー コマンドビジー
SD_ERR_NO_RESP_ERROR	-88	レスポンスなしエラー	SDIF 内部エラー レスポンスを受信できない
SD_ERR_INTERNAL	-99	内部エラー	SD ドライバ内部エラー

4. 関数リファレンス

4.1 関数リファレンスの読み方

本章では、SD ドライバライブラリ関数、ターゲット CPU インタフェース関数およびデバイスドライバ関数の詳細を示します。各関数詳細の読み方は以下のとおりです。

関数名		分類
機能概要		
書式	関数の呼び出し形式を示します。 <code>#include</code> “ヘッダファイル”で示すヘッダファイルは、この関数の実行に必要な標準ヘッダファイルです。必ずインクルードしてください。 I,O は、引数がそれぞれ入力データ、出力データであることを意味します。	
戻り値	関数の戻り値を示します。戻り値の後に「:」を付けて記載されているコメントは、その戻り値についての説明(リターン条件等)です。	
解説	関数の仕様について説明します。	
注意	注意事項があればここに示します。	
使用例	関数の使用例があればここに示します。	
作成例	関数の作成例があればここに示します。	

図 4.1 ライブラリ関数詳細の見方

4.2 ライブラリ関数

本節では、SD ドライバのライブラリ関数の詳細について説明します。表 4.1 にライブラリ関数の詳細を示します。

表 4.1 SD ドライバライブラリ関数一覧

関数名	機能概要
sd_init	ドライバの初期化
sd_finalize	ドライバの終了
sd_set_buffer	ライブラリ用バッファ領域の設定
sd_cd_int	カード挿抜割り込み設定
sd_check_media	カードの挿入確認
sd_set_seccnt	連続転送セクタ数の設定
sd_get_seccnt	連続転送セクタ数の取得
sd_mount	カードのマウント
sd_unmount	カードのマウントの解除
sd_inactive	カードの無効化
sd_read_sect	カードからのセクタリード
sd_write_sect	カードへのセクタライト
sd_get_type	カードタイプと動作モードの取得
sd_get_size	カードサイズの取得
sd_iswp	ライトプロテクト状態の取得
sd_stop	カード処理の強制停止
sd_set_intcallback	プロトコルステータス確認割り込みコールバック関数の登録
sd_int_handler	カード割り込みハンドラ
sd_check_int	カード割り込み要求確認
sd_get_reg	カードレジスタの取得
sd_get_rca	RCA レジスタの取得
sd_get_sdstatus	SD STATUS の取得
sd_get_error	ドライバエラーの取得
sd_set_cdtime	カード検出時間の設定
sd_set_responsetime	レスポンスタイムアウト時間の設定
sd_get_ver	ライブラリバージョンの取得
sd_lock_unlock	カードのロック・アンロック
sd_get_speed	カードスピードの取得

4.2.1 sd_init

sd_init

ライブラリ関数

ドライバの初期化

書式

```
#include "r_sdif.h"

int32_t sd_init(int32_t sd_port, uint32_t base, void *workarea, int32_t cd_port)
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    uint32_t base            | サンプリングクロックコントローラのベースアドレス
    void *workarea           | アクセスライブラリで使用するワーク領域
                             | (SD_SIZE_OF_INIT バイト分の領域が必要)
    int32_t cd_port          | 挿抜検出用ポートの設定
                             | SD_CD_SOCKET : 挿抜検出をカードソケット端子により行う
                             | SD_CD_DAT3  : 挿抜検出を CD/DAT3 端子により行う (予約)
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了
SD_ERR_CPU_IF : ターゲット CPU インタフェース関数エラー

解説

SD ドライバの初期化と SD ホストコントローラの初期化を行います。

引数 **base** により、サンプリングクロックコントローラのベースアドレスを指定します。ライブラリ関数では、ベースアドレスを元に SD ホストコントローラにアクセスします。サンプリングクロックコントローラのベースアドレスでない場合はエラー終了します。

引数 **workarea** で指定した **SD_SIZE_OF_INIT** (バイト) の領域を、ライブラリ関数のワーク領域として使用します。SD ドライバ処理を終了させるまで **workarea** で指定した領域を保持すると共にその内容をアプリケーションにて変更しないで下さい。また、**workarea** には 8 バイト境界のアドレスを指定してください。**workarea** が NULL または、8 バイト境界のアドレスでない場合はエラー終了します。

引数 **cd_port** により挿抜検出に使用するポート形式を指定します。

SD_CD_SOCKET を指定した場合は、SD カードソケットの CD 端子を挿抜検出に使用します。

SD_CD_DAT3 を指定した場合は、SD カードの CD/DAT3 端子を挿抜検出に使用します。

SD メモリカードおよび MMC カード共にサポートする場合は、ソケットの CD 端子を挿抜検出に使用してください。引数 **cd_port** に他の値を指定した場合はエラー終了します。

本関数実行後、挿抜検出割り込みは禁止です。挿抜割り込みを使用する場合は、**sd_cd_int** 関数にて挿抜割り込みを有効にしてください。

本関数よりターゲット CPU インタフェース関数 **sddev_init** 関数を呼び出します。**sddev_init** 関数内でカード操作に必要なポート設定や割り込み設定など SD ホストコントローラ以外の H/W の初期化を行ってください。

注意

本関数が正常終了しない場合、すべてのライブラリ関数が使用できません。

本関数では、カードに対する電源投入は行いません。

挿抜検出用ポートの設定はオプションです。SD カード検出オプション有効時のみ、設定値は有効です。詳細は、「5 コンフィグオプション」の SD カード検出オプションを参照してください。

挿抜検出用ポート **SD_CD_DAT3** 設定は、将来のための予約です。本バージョンにて **SD_CD_DAT3** を指定した場合、**SD_CD_SOCKET** と同じ動作を行います。

sd_get_error 関数によるエラー取得はできません。


```
使用例      /* ドライバの初期化からドライバの終了までの例 */
            #include "r_sdif.h"

            /* SD ドライバワーク領域定義 */
            uint32_t driver_work[SD_SIZE_OF_INIT/sizeof(uint32_t)];

            /* SD ドライババッファ領域定義 */
            uint32_t my_sd_buffer[512/sizeof(uint32_t)];

            /* サンプリングクロックコントローラのベースアドレスを設定します */
            #define HOST_IP_ADDR      (0xE8227000uL)

            void func(void)
            {
                /* ドライバの初期化 */
                if(sd_init(0, HOST_IP_ADDR, driver_work, SD_CD_SOCKET) != SD_OK){
                    /* 初期化失敗 */
                }

                /* SD ドライバ用バッファの設定 */
                sd_set_buffer(0, my_sd_buffer, sizeof(my_sd_buffer));

                /* ソフトウェアポーリング、DMA 転送、default-Speed カード対応、
                   High-Capacity カード対応、eXtended-Capacity カード対応でマウント */
                if(sd_mount(0
                           ,SD_MODE_POLL|SD_MODE_DMA|SD_MODE_DS|SD_MODE_VER2X
                           ,SD_VOLT_3_3) != SD_OK){
                    /* マウント失敗 */
                }

                /* カードへのアクセス処理 */

                /* マウントの解除 */
                sd_unmount(0);

                /* ドライバの終了処理 */
                sd_finalize(0);
            }
```

4.2.2 sd_finalize

sd_finalize

ライブラリ関数

ドライバの終了

書式

```
#include "r_sdif.h"
int32_t sd_finalize(int32_t sd_port)
    int32_t sd_port          |    SDHI チャンネル番号 (0 または 1)
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

SD ドライバのすべての処理を終了します。
本関数実行後は挿抜割り込みも禁止になります。
sd_init 関数に設定した SD ドライバのワーク領域は、本関数実行後開放してください。
本関数よりターゲット CPU インタフェース関数 sddev_finalize 関数を呼び出します。sddev_finalize 関数内にて SD メモリカード操作で使ったポート設定や割り込み設定など SD ホストコントローラ以外の H/W の終了処理を行ってください。

注意

sd_init 関数呼び出し前に本関数を実行した場合はエラー終了します。

使用例

sd_init 関数の使用例をご参照ください。

4.2.3 sd_set_buffer

sd_set_buffer

ライブラリ関数

ライブラリ用バッファ領域の設定

書式

```
#include "r_sdif.h"

int32_t sd_set_buffer(int32_t sd_port, void *buff, uint32_t size)
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    void *buff               | ライブラリバッファ領域へのポインタ
    uint32_t size            | バッファ領域のサイズ
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

ライブラリ関数で使用するバッファ領域を設定します。
引数 **buff** には、バッファ領域の先頭アドレスを、引数 **size** にはバッファ領域のサイズを指定します。
サイズに 0 を指定した場合は、エラー終了します。
バッファ領域 **buff** には 8 バイト境界に配置された 512 バイトの倍数の領域を指定してください。**buff** に 8 バイト境界でないポインタを指定した場合はエラー終了します。
バッファ領域には、少なくとも 512 バイトの領域が必要です。

注意

バッファ領域の設定は、**sd_mount** 関数実行前に行ってください。

使用例

sd_init 関数の使用例をご参照ください。

4.2.4 sd_cd_int

sd_cd_int

ライブラリ関数

カードの挿抜割り込みの設定

書式

```
#include "r_sdif.h"

int32_t sd_cd_int(int32_t sd_port, int32_t enable, int32_t (*callback)(int32_t, int32_t))
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    int32_t enable          | 挿抜割り込みの禁止許可設定
                             SD_CD_INT_ENABLE : 挿抜割り込み許可
                             SD_CD_INT_DISABLE : 挿抜割り込み禁止
    int32_t                 | 挿抜検出用コールバック関数指定
    (*callback)(int32_t, int32_t)
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

カード挿抜検出用割り込みの設定を行います。

挿抜割り込みの禁止許可設定 `enable` に `SD_CD_INT_ENABLE` を設定した場合は、挿抜割り込みを許可します。挿抜割り込みを許可した場合、ライブラリ関数 `sd_int_handler` 関数を SD ホストコントローラ割り込みの処理ルーチンとしてシステムに登録し、CPU の割り込みコントローラにより SD ホストコントローラ割り込みを有効にする必要があります。CPU の割り込みコントローラの設定は、`sddev_init` 関数内で行ってください。また挿抜割り込みを許可した場合、引数 `callback` に挿抜検出用コールバック関数を登録することにより、挿抜のイベントに対してユーザ処理を実行することが可能です。引数 `callback` にヌルポインタを指定した場合は、コールバック関数を登録しません。コールバック関数の詳細は、`sd_cd_callback` 関数説明をご参照ください。

挿抜割り込みの禁止許可設定 `enable` に `SD_CD_INT_DISABLE` を設定した場合、挿抜割り込みは禁止となります。挿抜割り込みを禁止にした場合、カードの挿抜確認は `sd_check_media` 関数で行ってください。

引数 `enable` に `SD_CD_INT_ENABLE` または `SD_CD_INT_DISABLE` 以外の値が設定された場合はエラーとなります。

注意

挿抜割り込みの禁止許可設定 `enable` に `SD_CD_INT_ENABLE` を設定した場合、`sd_mount` 関数で設定する動作モードのステータス確認方式には、必ず H/W 割り込み `SD_MODE_HWINT` を指定してください。

`sd_get_error` 関数によるエラー取得はできません。

挿抜割り込みは、本関数実行後にカードを挿抜することにより発生します。

本関数は、オプションです。SD カード検出オプション有効時のみ、使用可能です。

使用例

```
/* カードの挿抜割り込みの設定例 */  
#include "r_sdif.h"  
  
int32_t cd_callback(int32_t sd_port, int32_t in)  
{  
    /* 挿抜に対応した処理 */  
}  
  
void func(void)  
{  
    /* 挿抜割り込み許可、コールバック関数を登録 */  
    sd_cd_int(0, SD_CD_INT_ENABLE, cd_callback);  
}
```

4.2.5 sd_check_media

sd_check_media

ライブラリ関数

カードの挿入確認

書式	<pre>#include "r_sdif.h" int32_t sd_check_media(int32_t sd_port) int32_t sd_port I SDHI チャンネル番号 (0 または 1)</pre>
戻り値	<pre>SD_OK : カードが挿入されている SD_ERR : カードが挿入されていない</pre>
解説	<p>カードの挿入確認を行います。</p> <p>カードが挿入されている場合は、SD_OK を返します。</p> <p>カードが挿入されていない場合は、SD_ERR を返します。</p> <p>本関数は、挿抜検出として割り込みによる検出を選択した場合も使用できます。</p> <p>SD カード検出オプション無効時は、SD_OK（固定値）を返します。</p>
注意	<p>本関数実行前に、sd_init 関数による初期化が必要です。</p> <p>sd_get_error 関数によるエラー取得はできません。</p>
使用例	<pre>/* カードの挿入確認例 */ #include <stdio.h> #include "r_sdif.h" void func(void) { if(sd_check_media(0) == SD_OK){ printf("カードが挿入されています\n"); } else{ printf("カードが挿入されていません\n"); } }</pre>

4.2.6 sd_set_seccnt

sd_set_seccnt

ライブラリ関数

連続転送セクタ数の設定

書式

```
#include "r_sdif.h"

int32_t sd_set_seccnt(int32_t sd_port, int16_t sectors)
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    int16_t sectors          | 連続転送セクタ数 (3~0x7fff)
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

カードに対する連続転送セクタ数の最大値を設定します。sd_write_sect 関数または sd_read_sect 関数では、本関数で設定した連続転送セクタ数を基準に転送を行います。sd_write_sect 関数または sd_read_sect 関数に指定するセクタ数が、連続転送セクタ数以下の場合は、指定されたセクタ数で転送を行います。sd_write_sect 関数または sd_read_sect 関数に指定するセクタ数が、連続転送セクタ数よりも大きい場合は連続転送セクタ数単位に分割して転送を行います。

連続転送セクタ数の初期値は 256 セクタです。本関数による連続セクタ数の設定が行われなかった場合は、初期値を連続転送セクタ数とします。sd_init 関数実行時に初期値が設定されます。

引数 sectors に指定できる連続転送セクタ数は、3~0x7fff セクタです。この値以外を指定した場合は SD_ERR を返します。

注意

本関数実行前に、sd_init 関数による初期化が必要です。
sd_get_error 関数によるエラー取得はできません。

使用例

```
/* 連続転送セクタ数の設定例 */
#include <stdio.h>
#include "r_sdif.h"

void func(void)
{
    if(sd_set_seccnt(0, 1024) == SD_OK){
        printf("連続転送セクタ数を 1024 セクタに設定しました。¥n");
    }
}
```

4.2.7 sd_get_secCnt

sd_get_secCnt

ライブラリ関数

連続転送セクタ数の取得

書式

```
#include "r_sdif.h"
int32_t sd_get_secCnt(int32_t sd_port)
    int32_t sd_port          |    SDHI チャンネル番号 (0 または 1)
```

戻り値

≥1 : 連続転送セクタ数
SD_ERR : エラー終了

解説

カードに対する連続転送セクタ数を返します。
sd_init 関数による初期化前に本関数を実行した場合は、SD_ERR を返します。

注意

本関数実行前に、sd_init 関数による初期化が必要です。
sd_get_error 関数によるエラー取得はできません。

使用例

```
/* 連続転送セクタ数の取得例 */
#include <stdio.h>
#include "r_sdif.h"

void func(void)
{
    int32_t secCnt;

    secCnt = sd_get_secCnt(0);
    printf("連続転送セクタ数は%dセクタです。¥n", secCnt);
}
```


4.2.8 sd_mount

sd_mount

ライブラリ関数

カードのマウント

書式

```
#include "r_sdif.h"

int32_t sd_mount(int32_t sd_port, uint32_t mode, uint32_t voltage)
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    uint32_t mode            | 動作モード
    uint32_t voltage         | カード動作電圧
```

戻り値

SD_OK_LOCKED_CARD : 正常終了 (カードロック状態)
SD_OK : 正常終了
上記以外 : エラー終了 (詳細は「3.10 エラーコード」をご参照ください)

解説

カードのマウントを行ないます。

本関数が正常終了した場合、カードはトランスファステートへ遷移し、セクタのリード／ライトアクセスが可能になります。

ロック状態のカードのマウントが正常終了した場合、戻り値は **SD_OK_LOCKED_CARD** となります。ロックされたカードは特定のコマンドしか受け付けませんので、セクタのリード／ライトアクセスができません。ロック状態のカードにてセクタのリード／ライトアクセスするためには、カードのロック・アンロック関数よりカードをアンロックした後に、再度カードのマウントを行ってください。

なお、ロック状態のカードが受け付けるコマンドに関しては、SD PHYSICAL LAYER SPECIFICATION を参照ください。

引数 **mode** には、ライブラリ関数の動作モードを指定します。動作モード指定は表 4.2 のマクロ定義により行います。動作モードは表 4.2 に示す種別ごとに論理和で指定します。ステータス確認方式に **SD_MODE_HWINT** を設定した場合は、ライブラリ関数 **sd_int_handler** 関数を SD ホストコントローラ割り込みの処理ルーチンとしてシステムに登録し、CPU の割り込みコントローラにより SD ホストコントローラ割り込みを有効にする必要があります。CPU の割り込みコントローラの設定は、**sddev_init** 関数内で行ってください。

引数 **voltage** には、カードに供給する電圧範囲を指定します。電圧範囲指定は表 4.3 のマクロ定義により行ないます。指定した電圧で動作できないカードはマウントしません。

SD メモリカードと MMC カードの判別は本関数内で行います。SD メモリカードと認識した場合は、SD メモリカード内の CD/DAT3 端子のプルアップを無効にします。

注意

sd_cd_int 関数の挿抜割り込みの禁止許可設定 enable に SD_CD_INT_ENABLE を設定した場合、本関数で設定する動作モードのステータス確認方式には、必ず H/W 割り込み SD_MODE_HWINT を指定してください。

データアクセス方式に DMA 転送を使用する場合は、**sd_read_sect/sd_write_sect** に指定するバッファポインタが 8 バイト境界アドレスになるようにしてください。

sd_init 関数および **sd_set_buffer** 実行前に本関数を実行した場合はエラー終了します。

エラー終了の場合、本関数の呼び出し元でリトライ処理を行うことを推奨します。

使用例

```
/* カードのマウント例 */
#include "r_sdif.h"

void func(void)
{
    /* ソフトウェアポーリング、ソフトウェア転送、default-Speed カード対応、
       High-Capacity カード対応、eXtended-Capacity カード対応でカードのマウント */
    if(sd_mount(0
                ,SD_MODE_POLL|SD_MODE_SW|SD_MODE_DS|SD_MODE_VER2X
                ,SD_VOLT_3_3) != SD_OK){
        /* マウント失敗 */
    }
}

void func2(void)
{
    int32_t ret;
    char_t pwd[2];

    ret=sd_mount(0
                 ,SD_MODE_POLL|SD_MODE_SW|SD_MODE_DS|SD_MODE_VER2X
                 ,SD_VOLT_3_3);
    if(ret==SD_OK){
        /* マウント成功 */
    }
    else if(ret==SD_OK_LOCKED_CARD){
        /* マウント成功（カードロック状態）、パスワードを'12'でロック解除 */
        pwd[0] = 0x31;
        pwd[1] = 0x32;
        ret=sd_lock_unlock(0, 0x00, pwd, sizeof(pwd));
        if(ret==SD_OK){
            /* ロック解除成功、再マウント */
            ret=sd_mount(0
                        ,SD_MODE_POLL|SD_MODE_SW|SD_MODE_DS|SD_MODE_VER2X
                        ,SD_VOLT_3_3);
            if(ret==SD_OK){
                /* マウント成功 */
            }
            else{
                /* マウント失敗 */
            }
        }
        else{
            /* ロック解除失敗 */
        }
    }
    else{
        /* マウント失敗 */
    }
}
```

表 4.2 SD ドライバ動作モード

種別	マクロ定義	値	定義
ステータス確認方式	SD_MODE_POLL	0x00000000	ソフトウェアポーリング
	SD_MODE_HWINT	0x00000001	H/W 割り込み
データアクセス方式	SD_MODE_SW	0x00000000	ソフトウェア転送
	SD_MODE_DMA	0x00000002	DMA 転送
カード速度対応方式	SD_MODE_DS	0x00000000	Default-Speed モード対応
カード容量対応方式	SD_MODE_VER1X	0x00000000	Standard-Capacity カードのみ対応
	SD_MODE_VER2X	0x00000080	High-Capacity カード対応、eXtended-Capacity カード対応

表 4.3 動作電圧設定

動作電圧 (V)	マクロ定義	値
1.6-1.7	SD_VOLT_1_7	0x00000010
1.7-1.8	SD_VOLT_1_8	0x00000020
1.8-1.9	SD_VOLT_1_9	0x00000040
1.9-2.0	SD_VOLT_2_0	0x00000080
2.0-2.1	SD_VOLT_2_1	0x00000100
2.1-2.2	SD_VOLT_2_2	0x00000200
2.2-2.3	SD_VOLT_2_3	0x00000400
2.3-2.4	SD_VOLT_2_4	0x00000800
2.4-2.5	SD_VOLT_2_5	0x00001000
2.5-2.6	SD_VOLT_2_6	0x00002000
2.6-2.7	SD_VOLT_2_7	0x00004000
2.7-2.8	SD_VOLT_2_8	0x00008000
2.8-2.9	SD_VOLT_2_9	0x00010000
2.9-3.0	SD_VOLT_3_0	0x00020000
3.0-3.1	SD_VOLT_3_1	0x00040000
3.1-3.2	SD_VOLT_3_2	0x00080000
3.2-3.3	SD_VOLT_3_3	0x00100000
3.3-3.4	SD_VOLT_3_4	0x00200000
3.4-3.5	SD_VOLT_3_5	0x00400000
3.5-3.6	SD_VOLT_3_6	0x00800000

4.2.9 sd_unmount

sd_unmount

ライブラリ関数

カードのマウント解除

書式

```
#include "r_sdif.h"
int32_t sd_unmount(int32_t sd_port)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
```

戻り値

SD_OK	: 正常終了
SD_ERR	: エラー終了
SD_ERR_CPU_IF	: ターゲット CPU インタフェース関数エラー

解説

カードのマウントを解除して取り外し可能な状態にします。
本関数を実行しカードのマウントを解除した場合でも、カードの挿抜割り込みおよび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。

注意

使用例

```
/* カードのマウント解除例 */
#include "r_sdif.h"

void func(void)
{
    /* カードのマウントの解除 */
    sd_unmount(0);
}
```

4.2.10 sd_inactive

sd_inactive

ライブラリ関数

カードの無効化

書式	<pre>#include "r_sdif.h" int32_t sd_inactive(int32_t sd_port) int32_t sd_port SDHI チャンネル番号 (0 または 1)</pre>
戻り値	<pre>SD_OK : 正常終了 SD_OK 以外 : エラー終了（詳細は「3.10 エラーコード」をご参照ください）</pre>
解説	カードを任意のステートからインアクティブステートに遷移させ、無効にします。
注意	無効にしたカードを再度マウントするには、カードの挿抜が必要です。
使用例	<pre>/* カードの無効化例 */ #include <stdio.h> #include "r_sdif.h" void func(void) { /* カードを無効にする */ if(sd_inactive(0) != SD_OK){ printf("無効化に失敗しました\n"); } }</pre>

4.2.11 sd_read_sect

sd_read_sect

ライブラリ関数

カードからのセクタリード

書式

```
#include "r_sdif.h"

int32_t sd_read_sect(int32_t sd_port, uint8_t *buff, uint32_t psn, int32_t cnt)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
    uint8_t *buff            O    セクタデータの読み込み領域を示すポインタ
    uint32_t psn             I    読み出し開始物理セクタ番号
    int32_t cnt              I    読み出しセクタ数
```

戻り値

SD_OK : 正常終了
 SD_OK 以外 : エラー終了（詳細は「3.10 エラーコード」を参照ください）

解説

カードからセクタデータの読み出しを行います。
 物理セクタ番号 **psn** で指定したセクタから **cnt** セクタ分のデータを読み出し、引数 **buff** の示す領域に格納します。
 セクタデータの読み出しには、以下のコマンドを使用します。
 2 セクタ以下 : READ_SINGLE_BLOCK コマンド (CMD17)
 3 セクタ以上 : READ_MULTIPLE_BLOCK コマンド (CMD18)

引数 **buff** に NULL が指定された場合は、エラー終了します。
 また、本関数実行中に、カードが抜き取られた場合は処理を中止し、エラー終了します。
 (ただし、SD カード検出オプション有効時のみ)

注意

本関数には、物理セクタ番号を指定します。ファイルシステムから論理セクタ番号でセクタ番号が指定される場合は、論理セクタから物理セクタへの変換を行ってください。

エラー終了の場合、本関数の呼び出し元でリトライ処理を行うことを推奨します。

使用例

```
/* デバイスドライバ関数への実装例 */
#include "r_sdif.h"
int32_t offset ; /* 物理セクタへの変換用オフセット */

int32_t read_sector(int32_t side, uint8_t *buff, uint32_t secno, int32_t cnt)
{
    int32_t i;
    uint32_t psn;

    /* 物理セクタ番号への変換 */
    psn = secno + offset;

    /* 3 回のリトライ処理 */
    for(i=0; i < 3; i++){
        if(sd_read_sect(0, buff, psn, cnt) == SD_OK){
            return 0; /* 正常終了 */
        }
    }
    return -1; /* エラー終了 */
}
```

4.2.12 sd_write_sect

sd_write_sect

ライブラリ関数

カードへのセクタライト

書式

```
#include "r_sdif.h"

int32_t sd_write_sect(int32_t sd_port, uint8_t *buff, uint32_t psn, int32_t cnt, int32_t writemode)
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    uint8_t *buff            | 書き込みセクタデータ領域を示すポインタ
    uint32_t psn             | 書き込み開始物理セクタ番号
    int32_t cnt              | 書き込みセクタ数
    int32_t writemode        | 書き込みモード
                                SD_WRITE_WITH_PREERASE : プレイレースあり書き込み
                                SD_WRITE_OVERWRITE   : 上書きモード
```

戻り値

SD_OK : 正常終了
SD_OK 以外 : エラー終了（詳細は「3.10 エラーコード」を参照ください）

解説

カードへセクタデータの書き込みを行います。
引数 buff の示す領域のデータを、物理セクタ番号 psn で指定したセクタへ、cnt セクタ分書き込みます。

SD メモリカード時のみ書き込みモードに SD_WRITE_WITH_PREERASE を指定した場合は、書き込み対象セクタを消去してから書き込みます。MMC カードに対し SD_WRITE_WITH_PREERASE を指定しても、SD_WRITE_OVERWRITE と同じ動作を行います。

セクタデータの書き込みは、以下のコマンドを使用します。

2 セクタ以下 : WRITE_BLOCK コマンド (CMD24)

3 セクタ以上 : WRITE_MULTIPLE_BLOCK コマンド (CMD25)

引数 buff に NULL が指定された場合は、エラー終了します。

書き込み禁止状態のカードに対し本関数を実行した場合は、エラー終了します。

また、本関数実行中に、カードが抜き取られた場合は処理を中止します。

(ただし、SD カード検出オプション有効時のみ)

注意

本関数には、物理セクタ番号を指定します。ファイルシステムから論理セクタ番号でセクタ番号が指定される場合は、論理セクタから物理セクタへの変換を行ってください。

SD_WRITE_WITH_PREERASE 指定時の書き込み中にカードの抜取等が発生した場合、SD_WRITE_OVERWRITE 指定時と比べ、カードの内容が失われる可能性が高くなります。

エラー終了の場合、本関数の呼び出し元でリトライ処理を行うことを推奨します。

使用例

```
/* デバイスドライバ関数への実装例 */  
#include "r_sdif.h"  
int32_t write_sector(int32_t side, uint8_t *buff, uint32_t secno, int32_t cnt)  
{  
    int32_t i;  
    /* 3 回のリトライ */  
    for(i=0; i<3 ; i++){  
        /* 上書きモードで書き込み */  
        if(sd_write_sect(0, buff, secno, cnt, SD_WRITE_OVERWRITE)== SD_OK){  
            return 0;    /* 正常終了 */  
        }  
    }  
    return -1;    /* エラー */  
}
```


4.2.13 sd_get_type

sd_get_type

ライブラリ関数

カードタイプと動作モードの取得

書式

```
#include "r_sdif.h"

int32_t sd_get_type(int32_t sd_port, uint16_t *type, uint16_t *speed, uint8_t *capa)
    int32_t sd_port          I   SDHI チャンネル番号 (0 または 1)
    uint16_t *type           O   カードタイプ格納領域を示すポインタ
    uint16_t *speed         O   カード速度モード格納領域を示すポインタ
    uint8_t *capa           O   カード容量種別の格納領域を示すポインタ
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

マウントしているカードタイプ、カード速度モード、及びカード容量種別をそれぞれ、`type`、`speed` 及び `capa` の示す領域に格納します。

`type` には、マウントされているカードタイプとして以下の値を格納します。

SD_MEDIA_UNKNOWN

SD_MEDIA_MMC

SD_MEDIA_SD

`speed` には、マウントされているカード速度モードを以下のビットで格納します。0 固定です。

マウントされているカードが MMC カードの場合、0 が格納されます。

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
SPT							SPT
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CUR							CUR

ビット 8 (SPT ビット) は、マウントしているカードがサポートしている速度モードを表します。ビット 0 (CUR ビット) は、ライブラリがカードにアクセスする場合の速度モードを表します。

- ・ SPT ビットが 0b0 の場合、Default-Speed モードのカードをマウントしています。
- ・ CUR ビットが 0b0 の場合、ライブラリは Default-Speed モードでカードに対しアクセスを行います。ビット 1~7 およびビット 9~15 は予約ビットです。

`capa` には、マウントされているカードが High-Capacity カードまたは eXtended-Capacity カードであれば 1 が、Standard-Capacity カードであれば 0 が格納されます。

注意

引数がヌルポインタの場合、その情報は格納されません。

使用例

```
/* カードタイプと動作モードの取得例 */
#include <stdio.h>
#include "r_sdif.h"

void func(void)
{
    uint8_t  capa;

    /* カード容量種別の取得 */
    sd_get_type(0, NULL, NULL, &capa);

    if(capa == 1){
        printf("ライブラリは High-Capacity カードまたは eXtended-Capacity カードをマ
            ウントしています¥n");
    }
    else{
        printf("ライブラリは Standard-Capacity カードをマウントしています¥n");
    }
}
```

4.2.14 sd_get_size

sd_get_size

ライブラリ関数

カードサイズの取得

書式

```
#include "r_sdif.h"

int32_t sd_get_size(int32_t sd_port, uint32_t *user, uint32_t *protect)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
    uint32_t *user           O    ユーザ領域サイズ格納先を示すポインタ
    uint32_t *protect       O    プロテクト領域サイズ格納先を示すポインタ
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

カードのユーザ領域とプロテクト領域の物理的な容量をセクタ数で `user` および `protect` で示される領域に格納します。セクタ数×512 が領域のバイト数になります。

`user` または `protect` が NULL の場合、そのサイズは格納されません。
MMC カードの場合、プロテクト領域のサイズは常に 0 です。

注意

論理層のサイズは、ファイルシステム情報（マスタブートレコード、パーティションブートレコード）から取得する必要があります。

使用例

```
/* カードサイズの取得例 */
#include <stdio.h>
#include "r_sdif.h"

void func(void)
{
    uint32_t size, bytes;

    /* カードサイズの取得 */
    sd_get_size(0, &size, NULL);

    bytes = size * 512;
    printf("カードのサイズは、%d バイト、%d セクタです\n", bytes, size);
}
```

4.2.15 sd_iswp

sd_iswp

ライブラリ関数

ライトプロテクト状態の取得

書式

```
#include "r_sdif.h"
int32_t sd_iswp(int32_t sd_port)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
```

戻り値

SD_WP_OFF	: ライトプロテクトなし
SD_WP_HW	: H/W ライトプロテクト (オプション)
SD_WP_TEMP	: CSD レジスタ TMP_WRITE_PROTECT ビット ON
SD_WP_PERM	: CSD レジスタ PERM_WRITE_PROTECT ビット ON
SD_WP_ROM	: SD-ROM

解説

カードのライトプロテクト状態を返します。
カードがマウントされていない場合(sd_mount 関数を実行していない場合)の戻り値は不定です。

注意

H/W ライトプロテクトはオプションです。ライトプロテクト信号検出オプション有効時のみ、有効です。詳細は、「5 コンフィグオプション」の SD カード検出オプションを参照してください。

使用例

```
/* ライトプロテクト状態の取得例 */
#include <stdio.h>
#include "r_sdif.h"

void func(void)
{
    int32_t wp;

    /* 書き込み禁止状態の取得 */
    wp = sd_iswp(0);

    if(wp == SD_WP_OFF){
        printf("カードへの書き込みが可能です¥n");
    }
    else{
        printf("カードへの書き込みはできません¥n");
    }
}
```

4.2.16 sd_stop

sd_stop

ライブラリ関数

カード処理の強制停止

書式	<pre>#include "r_sdif.h" void sd_stop(int32_t sd_port) int32_t sd_port</pre> <p>I SDHI チャンネル番号 (0 または 1)</p>
戻り値	なし
解説	<p>カードに対するセクタのリード/ライト処理を強制終了します。</p> <p>アプリケーションによる処理の中断や割り込みなどによる抜き取り検出時に使用します。</p> <p>本関数の実行による強制停止時状態は、sd_read_sect 関数、sd_write_sect 関数、sd_mount 関数を実行するまで有効です。本関数の実行後、sd_read_sect 関数、sd_write_sect 関数を呼び出した場合、処理を行わずエラー終了します。</p> <p>sd_read_sect 関数、sd_write_sect 関数実行中に本関数を実行した場合、処理途中で転送処理を終了します。</p>
注意	書き込み処理中に、強制終了した場合、カードの内容は保証されません。
使用例	<pre>/* カード処理の強制停止例 */ #include "r_sdif.h" void func(void) { sd_stop(0); if(sd_read_sect(0, buffer, 0, 1) != SD_OK){ /* SD_ERR_STOP エラーになります */ } }</pre>

4.2.17 sd_set_intcallback

sd_set_intcallback

ライブラリ関数

プロトコルステータス確認割り込みコールバック関数の登録

書式

```
#include "r_sdif.h"

int32_t sd_set_intcallback(int32_t sd_port, int32_t (*callback)(int32_t, int32_t));
    int32_t sd_port          |   SDHI チャンネル番号 (0 または 1)
    int32_t                  |   登録するコールバック関数
    (*callback)(int32_t, int32_t)
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

SD メモリカードプロトコルステータス確認用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、SD メモリカードのプロトコルステータスが変化した場合の割り込み発生時呼び出されます。

登録したコールバック関数内で、割り込み発生待ちを行っているタスクの待ち状態を解除するなどの処理を行ないます。

コールバック関数を使用する場合は、sd_mount 関数実行前にコールバック関数を登録してください。コールバック関数を定義しない場合は、割り込み処理にてコールバック関数の呼び出しを行いません。またヌルポインタをセットした場合は、登録されているコールバック関数を削除します。

注意

本関数で登録するコールバック関数は、挿抜検出用コールバック関数とは別関数です。

本関数で登録したコールバック関数は、挿抜検出時には呼び出されません。

本関数のエラーは sd_get_error 関数では取得できません。

使用例

```
/* プロトコルステータス確認割り込みコールバック関数の登録例 */
#include "r_sdif.h"

int32_t my_sd_callback(int32_t sd_port, int32_t rsvd)
{
    /* SD ステータス確認待ちの起床処理など */
}

void func(void)
{
    /* ステータス確認用コールバック関数の登録 */
    sd_set_intcallback(0, my_sd_callback);

    sd_mount(0
        , SD_MODE_POLL | SD_MODE_DMA | SD_MODE_DS | SD_MODE_VER2X
        , SD_VOLT_3_3);
    /* 省略 */
    sd_unmount(0);
    /* ステータス確認用コールバック関数の削除 */
    sd_set_intcallback(0, NULL);
}
```

4.2.18 sd_int_handler

sd_int_handler

ライブラリ関数

カード割り込みハンドラ

書式

```
#include "r_sdif.h"
void sd_int_handler(int32_t sd_port)
    int32_t sd_port          |    SDHI チャンネル番号 (0 または 1)
```

戻り値 なし

解説 カードの割り込みハンドラです。

カードの挿抜検出割り込みおよびSD プロトコルステータス割り込みのどちらかを使用する場合は、SD ホストコントローラに対応する割り込み要因の処理ルーチンとしてシステムに組み込んでください。

挿抜割り込みコールバック関数およびステータス確認割り込みコールバック関数を登録している場合は、本関数内からコールバック関数を呼び出します。

4.2.19 sd_check_int

sd_check_int

ライブラリ関数

カード割り込み要求確認

書式	<pre>#include "r_sdif.h" int32_t sd_check_int(int32_t sd_port) int32_t sd_port SDHI チャンネル番号 (0 または 1)</pre>		
戻り値	SD_OK	:	割り込み要求あり
	SD_ERR	:	割り込み要求無し
解説	<p>カード 割り込み要求の発生状態を確認します。</p> <p>カードからの 割り込み要求が発生している場合、SD_OK を返します。</p> <p>カードからの 割り込み要求が発生していない場合、SD_ERR を返します。</p> <p>ターゲット CPU インタフェース関数 sddev_int_wait 関数内で SD プロトコルのステータス確認を行う場合に使用します。</p>		
注意	本関数は、sddev_int_wait 関数内のみで使用してください。		
使用例	<pre>/* カード割り込み要求確認例 */ #include "r_sdif.h" int32_t sddev_int_wait(int32_t sd_port, int32_t time) { /* 割り込み要求待ち */ while(1){ if(sd_check_int(sd_port) == SD_OK){ /* 割り込み要求あり */ break; } /* タイムアウト処理など */ } return SD_OK; }</pre>		

4.2.20 sd_get_reg

sd_get_reg

ライブラリ関数

カードレジスタの取得

書式

```
#include "r_sdif.h"

int32_t sd_get_reg(int32_t sd_port, uint8_t *ocr, uint8_t *cid, uint8_t *csd, uint8_t *dsr, uint8_t *scr)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
    uint8_t *ocr             O    OCR レジスタ内容格納先
    uint8_t *cid             O    CID レジスタ内容格納先
    uint8_t *csd             O    CSD レジスタ内容格納先
    uint8_t *dsr             O    DSR レジスタ内容格納先
    uint8_t *scr             O    SCR レジスタ内容格納先
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー

解説

カードの各レジスタの内容を引数 ocr、cid、csd、dsr、scr で示された領域に格納します。表 4.5～表 4.9 に各レジスタ値格納領域に格納されるレジスタのビット情報を示します。

各領域は表 4.4 に示すサイズの領域が必要です。本関数の呼び出し元で格納先の領域を確保してください。

ただし、引数がヌルポインタの場合は、値を格納しません。

各レジスタのデータは sd_mount 関数が正常終了した場合のみ有効です。sd_mount 関数を実行していない場合または正常終了していない場合の各レジスタの値は無効です。

DSR レジスタは、オプションレジスタのため、DSR レジスタが実装されていないカードの場合、すべて 0 が格納されます。

SCR レジスタは SD メモリカード専用です。MMC カードの場合はすべて 0 が格納されます。

注意

sd_get_error 関数によるエラー取得はできません。

使用例

```
/* OCR レジスタ、CID レジスタの取得例 */
#include "r_sdif.h"

void func(void)
{
    uint8_t ocr[4], cid[16];

    /* レジスタ情報取得 (CSD, DSR, SCR レジスタは取得しない) */
    if(sd_get_reg(0, ocr, cid, 0, 0, 0) != SD_OK){
        /* 情報取得失敗 */
    }
}
```

表 4.4 レジスタ値格納領域

引数名	対応するレジスタ	必要な領域サイズ
ocr	OCR レジスタ	4 バイト
cid	CID レジスタ	16 バイト
csd	CSD レジスタ	16 バイト
dsr	DSR レジスタ	2 バイト
scr	SCR レジスタ	8 バイト

表 4.5 OCR レジスタ値格納領域に格納される OCR レジスタのビット情報

OCR レジスタ値格納領域内 バイトオフセット	格納される OCR レジスタのビット情報
0	[31:24]
1	[23:16]
2	[15:8]
3	[7:0]

表 4.6 CID レジスタ値格納領域に格納される CID レジスタのビット情報

CID レジスタ値格納領域内 バイトオフセット	格納される CID レジスタのビット情報
0	ALL"0"
1	[127:120]
2	[119:112]
3	[111:104]
4	[103:96]
5	[95:88]
6	[87:80]
7	[79:72]
8	[71:64]
9	[63:56]
10	[55:48]
11	[47:40]
12	[39:32]
13	[31:24]
14	[23:16]
15	[15:8]

表 4.7 CSD レジスタ値格納領域に格納される CSD レジスタのビット情報

CSD レジスタ値格納領域内 バイトオフセット	格納される CSD レジスタのビット情報
0	ALL"0"
1	[127:120]
2	[119:112]
3	[111:104]
4	[103:96]
5	[95:88]
6	[87:80]
7	[79:72]
8	[71:64]
9	[63:56]
10	[55:48]
11	[47:40]
12	[39:32]
13	[31:24]
14	[23:16]
15	[15:8]

表 4.8 DSR レジスタ値格納領域に格納される DSR レジスタのビット情報

DSR レジスタ値格納領域内 バイトオフセット	格納される DSR レジスタのビット情報
0	[15:8]
1	[7:0]

表 4.9 SCR レジスタ値格納領域に格納される SCR レジスタのビット情報

SCR レジスタ値格納領域内 バイトオフセット	格納される SCR レジスタのビット情報
0	[63:56]
1	[55:48]
2	[47:40]
3	[39:32]
4	[31:24]
5	[23:16]
6	[15:8]
7	[7:0]

4.2.21 sd_get_rca

sd_get_rca

ライブラリ関数

RCA レジスタの取得

書式

```
#include "r_sdif.h"
int32_t sd_get_rca(int32_t sd_port, uint8_t *rca)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
    uint8_t *rca             O    RCA レジスタ内容格納先
```

戻り値

SD_OK : 正常終了

SD_ERR : エラー

解説

カードの RCA レジスタの内容を引数 rca で示された領域に格納します。表 4.10 に RCA レジスタ値格納領域に格納されるレジスタのビット情報を示します。

RCA 格納に 2 バイトの領域が必要です。本関数の呼び出し元で格納先の領域を確保してください。

ただし、引数がヌルポインタの場合は、値を格納しません。

RCA レジスタのデータは sd_mount 関数が正常終了した場合のみ有効です。sd_mount 関数を実行していない場合または正常終了していない場合の各レジスタの値は無効です。

注意

sd_get_error 関数によるエラー取得はできません。

使用例

```
/* RCA レジスタの取得例 */
#include "r_sdif.h"

void func(void)
{
    uint8_t rca[2];

    /* RCA レジスタ情報取得 */
    if(sd_get_rca(0, rca) != SD_OK){
        /* 情報取得失敗 */
    }
}
```

表 4.10 RCA レジスタ値格納領域に格納される RCA レジスタのビット情報

RCA レジスタ値格納領域内 バイトオフセット	格納される RCA レジスタのビット情報
0	[15:8]
1	[7:0]

4.2.22 sd_get_sdstatus

sd_get_sdstatus

ライブラリ関数

SD STATUS の取得

書式

```
#include "r_sdif.h"

int32_t sd_get_sdstatus(int32_t sd_port, uint8_t *sdstatus)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
    uint8_t *sdstatus        O    SD STATUS データ内容格納先
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー

解説

カードの SD STATUS 上位 16 バイトの内容を引数 `sdstatus` で示された領域に格納します。
データの格納に 16 バイトの領域が必要です。本関数の呼び出し元で格納先の領域を確保してください。
ただし、引数がヌルポインタの場合は、値を格納しません。

SD STATUS のデータは `sd_mount` 関数が正常終了した場合のみ有効です。`sd_mount` 関数を実行していない場合または正常終了していない場合の各レジスタの値は無効です。

SD STATUS は SD メモリカード専用です。MMC カードの場合はすべて 0 が格納されます。

注意

`sd_get_error` 関数によるエラー取得はできません。

使用例

```
/* SD STATUS の取得例 */
#include "r_sdif.h"

void func(void)
{
    uint8_t sdstatus[16];

    /* SD STATUS 情報取得 */
    if(sd_get_sdstatus(0, sdstatus) != SD_OK){
        /* 情報取得失敗 */
    }
}
```

4.2.23 sd_get_error

sd_get_error

ライブラリ関数

ドライバエラーの取得

書式

```
#include "r_sdif.h"
int32_t sd_get_error(int32_t sd_port)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
```

戻り値

エラーコード

解説

ライブラリ関数 `sd_mount` 関数、`sd_read_sect` 関数および `sd_write_sect` 関数の実行時、発生したエラーのエラーコードを返します。

エラーコードについては、「3.10 エラーコード」をご参照ください。

アプリケーションプログラムで SD ドライバのエラー詳細を取得する場合に使用します。

使用例

```
/* ドライバエラーの取得例 */
#include "r_sdif.h"

void func(void)
{
    int32_t err_code;

    if(sd_read_sect() < 0){
        /* SD ドライバのエラーを取得 */
        err_code = sd_get_error(0);
        if(err_code != SD_OK){
            /* SD ドライバのエラー発生 */
        }
    }
}
```

4.2.24 sd_set_cdtime

sd_set_cdtime

ライブラリ関数

カード検出時間の設定

書式

```
#include "r_sdif.h"

int32_t sd_set_cdtime(int32_t sd_port, uint16_t cdtime)
    int32_t sd_port          |    SDHI チャンネル番号 (0 または 1)
    uint16_t cdtime          |    カード検出時間設定値 (0x0000~0x000e)
```

戻り値

SD_OK	: 正常終了
SD_ERR	: エラー終了

解説 カード検出のためのカウント値を指定します。カード検出時間は、`cdtime` で指定したカウント値により次の式で決定されます。

$$\text{カード検出時間} = \text{CLK} \times 2^{10 + \text{cdtime}}$$

CLK : SD ホストコントローラ動作クロック (SD_CLK) サイクル時間

カウント値は、0x0000~0x000e の範囲で指定してください。

カウント値の初期値は 0x000e です。カウント値は `sd_init` 関数実行時に初期化されます。

使用例

```
/* カード検出時間の設定例 */
#include "r_sdif.h"

void func(void)
{
    if(sd_set_cdtime(0, 0x0a) == SD_OK){
        /* カード検出時間設定成功 */
    }
    else{
        /* カード検出時間設定失敗 */
    }
}
```

4.2.25 sd_set_responsetime

sd_set_responsetime

ライブラリ関数

レスポンスタイムアウト時間の設定

書式

```
#include "r_sdif.h"

int32_t sd_set_responsetime(int32_t sd_port, uint16_t responsetime)
    int32_t sd_port          |    SDHI チャンネル番号 (0 または 1)
    uint16_t responsetime    |    レスポンスタイムアウト設定値 (0x0000～0x000f)
```

戻り値

SD_OK	: 正常終了
SD_ERR	: エラー終了

解説

レスポンスタイムアウト時間を設定します。

レスポンスタイムアウト検出のためのカウント値を指定します。レスポンスタイムアウト検出時間は、responsetimeで指定したカウント値により次の式で決定されます。

レスポンスタイムアウト検出時間 = $SDCLK \times 2^{13+responsetime}$
SDCLK : SD クロック

カウント値は、0x0000～0x000f の範囲で指定してください。
カウント値の初期値は 0x000e です。カウント値は sd_init 関数実行時に初期化されます。

使用例

```
/* レスポンスタイムアウト時間の設定例 */
#include "r_sdif.h"

void func(void)
{
    if(sd_set_responsetime(0, 0x000a) == SD_OK){
        /* 設定成功 */
    }
    else{
        /* 設定失敗 */
    }
}
```


4.2.26 sd_get_ver

sd_get_ver

ライブラリ関数

ライブラリバージョンの取得

書式	<pre>#include "r_sdif.h" int32_t sd_get_ver(int32_t sd_port, uint16_t *sdhi_ver, char_t *sddrv_ver) int32_t sd_port I SDHI チャンネル番号 (0 または 1) uint16_t *sdhi_ver O SD ホストコントローラ IP バージョン格納先を示すポインタ char_t *sddrv_ver O SD ドライバライブラリバージョン格納先を示すポインタ</pre>
----	--

戻り値	SD_OK	: 正常終了
	SD_ERR	: エラー終了

解説 SD ホストコントローラIP のバージョンレジスタの内容をsdhi_ver で示される領域に格納します。また、本ライブラリのバージョンをASCII 文字列にてsddrv_ver で示される領域に格納します。SD ドライバライブラリバージョンの格納領域として32バイトの領域が必要です。

格納先を示すポインタが NULL の場合、その情報は格納されません。

使用例

```
/* ライブラリバージョンの取得例 */
#include "r_sdif.h"

void func(void)
{
    uint16_t ip_ver;
    char_t lib_ver[32];

    /* バージョンの取得 */
    sd_get_ver(0, &ip_ver, lib_ver);

    printf("IP のバージョンは 0x%04x です\n", ip_ver);
    printf("ライブラリのバージョンは%s です\n", lib_ver);
}
```

4.2.27 sd_lock_unlock

sd_lock_unlock

ライブラリ関数

カードのロック・アンロック

書式

```
#include "r_sdif.h"

int32_t sd_lock_unlock(int32_t sd_port, uint8_t code, uint8_t *pwd, uint8_t len)
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    uint8_t code             | オペレーションコード
    uint8_t *pwd             | パスワード
    uint8_t len              | パスワードの長さ (バイト) (1~16 バイト)
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

引数code にて指定したオペレーションコードにより、カードのロックまたはアンロックを行います。
表4.11～表4.12 にオペレーションコードの説明を示します。

注意

パスワード入力の際によりカードをアンロックできなくなることを防止するため、ロック状態のカードに対してパスワードの設定 (Bit0 : SET_PWD=1) をすることはできません。その場合、エラー終了します。パスワードの更新はカードをアンロックした後に行ってください。
パスワード変更時、旧パスワード長と新パスワード長の合計が16バイトを超える場合、パスワードを変更することができません。この場合、パスワードの解除(Bit1 : CLR_PWD=1) 後にパスワードの設定 (Bit0 : SET_PWD=1) を行なってください。

使用例

```
/* カードのロック・アンロック例 */
#include <stdio.h>
#include "r_sdif.h"

void func(void)
{
    uint8_t code;
    uint8_t pwd[2];

    /* パスワードを'12'に設定 */
    pwd[0] = 0x31;
    pwd[1] = 0x32;
    /* パスワード'12'でカードをロック */
    code = 0x05;
    if(sd_lock_unlock(0, code, pwd, sizeof(pwd)) != SD_OK){
        printf("ロック動作に失敗しました¥n");
        return;
    }
    printf("カードをロックしました¥n");
}
```

表 4.11 オペレーションコード説明(1)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0 に設定してください				ERASE	LOCK_ UNLOCK	CLR_PWD	SET_PWD

表 4.12 オペレーションコード説明(2)

項目	説明	備考
ERASE	1 : Forced Erase Operation	他のビット(Bit2-0)は無視されます。
LOCK_UNLOCK	1 : Lock Operation を実行 0 : Unlock Operation を実行	Set New password to PWD operation との同時実行が可能です。
CLR_PWD	1 : Clears PWD operation を実行	
SET_PWD	1 : Set New password to PWD operation を実行	

【注】 各オペレーションの詳細については SD PHYSICAL LAYER SPECIFICATION を参照ください。

4.2.28 sd_get_speed

sd_get_speed

ライブラリ関数

カードスピードの取得

書式

```
#include "r_sdif.h"

int32_t sd_get_speed(int32_t sd_port, uint8_t *clss, uint8_t *move)
    int32_t sd_port          I   SDHI チャンネル番号 (0 または 1)
    uint8_t *clss            O   スピードクラスの格納領域を示すポインタ
    uint8_t *move            O   転送速度の格納領域を示すポインタ
```

戻り値

SD_OK : 正常終了

SD_ERR : エラー終了

解説

マウントされているカードのスピードクラス、及び転送速度をそれぞれ、clss 及び move の示す領域に格納します。

clss には、マウントされているカードのスピードクラスの値を格納します。

表 4.13 スピードクラス

スピードクラス	値
Class 0 (C0)	0x00
Class 2 (C2)	0x01
Class 4 (C4)	0x02
Class 6 (C6)	0x03

move にはライブラリが Default-Speed モードでマウントされているカードの最低速度[MB/sec]を格納します。(speed class2～6 の時のみ値が格納されます。)

使用例

```
/* スピードクラスの取得例 */
#include <stdio.h>
#include "r_sdif.h"

void func(void)
{
    uint8_t  class;

    /* スピードクラスの取得 */
    sd_get_speed(0, &class, NULL);

    /* スピードクラス判別 */
    if(class == 0x01){
        printf("スピードクラスは 2 です¥n");
    }
    else if(class == 0x02){
        printf("スピードクラスは 4 です¥n");
    }
    else if(class == 0x03){
        printf("スピードクラスは 6 です¥n");
    }
    else {
        printf("スピードクラスは 0 です¥n");
    }
}
```

4.3 ターゲット CPU インタフェース関数

SD ドライバをシステムに組み込むためには、ターゲットシステムに応じたターゲット CPU インタフェース関数の作成が必要です。表 4.14 にターゲット CPU インタフェース関数一覧を示します。

表 4.14 ターゲット CPU インタフェース関数一覧

関数名	機能概要
sddev_init	H/W の初期化
sddev_finalize	H/W の終了処理
sddev_power_on	カードへの電源供給開始
sddev_power_off	カードへの電源供給停止
sddev_read_data	データリード処理
sddev_write_data	データライト処理
sddev_get_clockdiv	クロック分周比の取得
sddev_set_port	カード用ポート設定
sddev_int_wait	カード割り込み待ち
sddev_loc_cpu	カード割り込みの禁止
sddev_unl_cpu	カード割り込みの許可
sddev_init_dma	データ転送用 DMA の初期化
sddev_wait_dma_end	データ転送用 DMA 転送完了待ち
sddev_disable_dma	データ転送用 DMA の禁止
sddev_reset_dma	DMA のリセット
sddev_finalize_dma	DMA の終了処理

4.3.1 sddev_init

sddev_init

ターゲット CPU インタフェース関数

H/W の初期化

書式	<pre>#include "r_sdif.h" int32_t sddev_init(int32_t sd_port) int32_t sd_port SDHI チャンネル番号 (0 または 1)</pre>		
戻り値	SD_OK	:	正常終了
	SD_ERR	:	エラー終了
解説	<p>カードアクセスのために、SD ホストコントローラ以外に必要な CPU の H/W 資源の初期化を行います。本関数は、ライブラリ関数 <code>sd_init</code> 関数から呼び出されます。</p> <p>カード挿入検出用端子 (CD 端子) を有効にし、必要であれば SD ホストコントローラ割り込みの設定を行ってください。</p>		
注意	<p>カードの挿入検出開始からカードが挿入されていることを認識するまでの時間 (以降カード検出時間とします) として、<u>2²⁴ クロック (基準クロックは SD ホストコントローラへの供給クロック) が必要です</u>。挿入検出は SD ホストコントローラのカード挿入検出用端子を有効にしたときから開始しますので、<code>sd_init</code> 関数実行後からカード検出時間経過した後に、他のライブラリ関数を実行してください。カード検出時間経過前に他のライブラリ関数を実行した場合、カードの挿入/未挿入に関わらず、「カード未挿入エラー」になる場合があります。</p> <p>カード検出時間は、ライブラリ関数 <code>sd_set_cdtype</code> 関数で変更することができます。</p> <p><u>SD ホストコントローラ割り込みを有効にした場合は、<code>sd_mount</code> 関数で指定するステータス確認方式には <code>SD_MODE_HWINT</code> を指定してください。SD ホストコントローラ割り込みを有効にした場合、ソフトウェアポーリングは使用できません。</u></p>		

4.3.2 sddev_finalize

sddev_finalize

ターゲット CPU インタフェース関数

H/W の終了処理

書式	<pre>#include "r_sdif.h" int32_t sddev_finalize(int32_t sd_port); int32_t sd_port I SDHI チャンネル番号 (0 または 1)</pre>
戻り値	SD_OK : 正常終了
解説	<p>カード関連の H/W の終了処理を行いません。本関数は、ライブラリ関数 <code>sd_finalize</code> 関数から呼び出されます。</p> <p><code>sddev_init</code> 関数で設定した周辺 I/O で終了処理が必要なものがあれば、本関数内で処理を行いません。</p>

4.3.3 sddev_power_on

sddev_power_on

ターゲット CPU インタフェース関数

カードへの電源供給開始

書式

```
#include "r_sdif.h"
int32_t sddev_power_on(int32_t sd_port)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
```

戻り値

SD_OK	: 正常終了
SD_ERR	: エラー終了

解説

カードへの電源を供給します。本関数は、`sd_mount` 関数から呼び出されます。

カードの初期化時間確保のため、カードへの電源供給後、カードの動作電源電圧に達するまで、待ち時間を確保してください。パワーアップシーケンスの詳細は [SD Memory Card Specifications Part1 PHYSICAL LAYER SPECIFICATION](#) を参照してください。

4.3.4 sddev_power_off

sddev_power_off

ターゲット CPU インタフェース関数

カードへの電源供給停止

書式	#include "r_sdif.h"	
	int32_t sddev_power_off(int32_t sd_port)	
	int32_t sd_port	I SDHI チャンネル番号 (0 または 1)
戻り値	SD_OK	: 正常終了
	SD_ERR	: エラー終了
解説	カードへの電源供給を停止します。	
	パワーダウンシーケンスの詳細は SD Memory Card Specifications Part1 PHYSICAL LAYER SPECIFICATION を参照してください。	

4.3.5 sddev_read_data

sddev_read_data

ターゲット CPU インタフェース関数

データリード処理

書式

```
#include "r_sdif.h"

int32_t sddev_read_data(int32_t sd_port, uint8_t *buff, uint32_t reg_addr, int32_t num);
```

int32_t sd_port	I	SDHI チャンネル番号 (0 または 1)
uint8_t *buff	O	読み出したデータの格納先を示すポインタ
uint32_t reg_addr	I	ホストコントローラ I/P データレジスタアドレス
int32_t num	I	読み出しバイト数

戻り値

SD_OK	: 正常終了
SD_ERR	: エラー終了

解説

データレジスタ `reg_addr` からセクタデータを `num` バイト数分読み出し、`buff` で示される領域に格納します。

データレジスタアドレスは 8 バイトアライメントのアドレスが指定されます。データレジスタアドレスを `uint64_t` 型のポインタに変換してアクセスを行ってください。

読み出しバイト数の最大値は 512 バイトです。

注意

読み出しデータ格納先のバイトアライメントは、アプリケーションプログラムに依存します。
本関数は DMA 転送選択時も実装する必要があります。

作成例

```
/* データリード処理例 */
#include "r_sdif.h"

int32_t sddev_read_data(int32_t sd_port
                        ,uint8_t *buff
                        ,uint32_t reg_addr
                        ,int32_t num)
{
    int32_t i;
    int32_t cnt;
    uint64_t *reg;
    uint64_t *ptr_l;
    uint8_t *ptr_c;
    volatile uint64_t tmp;

    reg = (uint64_t *)(reg_addr);

    cnt = (num / 8);
    if(((uint32_t)buff & 0x7uL) != 0uL){
        ptr_c = (uint8_t *)buff;
        for(i = cnt; i > 0 ; i--){
            tmp = *reg;
            *ptr_c++ = (uint8_t)(tmp);
            *ptr_c++ = (uint8_t)(tmp >> 8);
            *ptr_c++ = (uint8_t)(tmp >> 16);
        }
    }
}
```

```
*ptr_c++ = (uint8_t)(tmp >> 24);
*ptr_c++ = (uint8_t)(tmp >> 32);
*ptr_c++ = (uint8_t)(tmp >> 40);
*ptr_c++ = (uint8_t)(tmp >> 48);
*ptr_c++ = (uint8_t)(tmp >> 56);
}

cnt = (num % 8);
if(cnt != 0){
    tmp = *reg;
    for(i = cnt; i > 0 ; i--){
        *ptr_c++ = (uint8_t)(tmp);
        tmp >>= 8;
    }
}
else{
    ptr_l = (uint64_t *)buff;
    for(i = cnt; i > 0 ; i--){
        *ptr_l++ = *reg;
    }

    cnt = (num % 8);
    if(cnt != 0){
        ptr_c = (uint8_t *)ptr_l;
        tmp = *reg;
        for(i = cnt; i > 0 ; i--){
            *ptr_c++ = (uint8_t)(tmp);
            tmp >>= 8;
        }
    }
}

return SD_OK;
}
```

4.3.6 sddev_write_data

sddev_write_data

ターゲット CPU インタフェース関数

データライト処理

書式

```
#include "r_sdif.h"

int32_t sddev_write_data(int32_t sd_port, uint8_t *buff, uint32_t reg_addr, int32_t num);

int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
uint8_t *buff            | 書き込みデータ格納先を示すポインタ
uint32_t reg_addr        | ホストコントローラ I/P データレジスタアドレス
int32_t num              | 書き込みバイト数
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

buff で示されるセクタデータを、num バイト分データレジスタ reg_addr へ書き込みます。
データレジスタアドレスは 8 バイトアライメントのアドレスが指定されます。データレジスタアドレスを uint64_t 型のポインタに変換してアクセスを行ってください。
書き込みバイト数の最大値は 512 バイトです。

注意

データレジスタは必ず 64 ビット長でアクセスを行ってください。
書き込みデータ格納先のバイトアライメントは、アプリケーションプログラムに依存します。
本関数は DMA 転送選択時も実装する必要があります。

作成例

```
/* データライト処理例 */
#include "r_sdif.h"

int32_t sddev_write_data(int32_t sd_port
                        ,uint8_t *buff
                        ,uint32_t reg_addr
                        ,int32_t num)
{
    int32_t i;
    uint64_t *reg = (uint64_t *) (reg_addr);
    uint64_t *ptr = (uint64_t *) buff;
    uint64_t tmp;

    num += 7;
    num /= 8;
    if(((uint32_t)buff & 0x7) != 0){
        for(i = num; i > 0 ; i--){
            tmp = *buff++ ;
            tmp |= *buff++ << 8;
            tmp |= *buff++ << 16;
            tmp |= *buff++ << 24;
            tmp |= *buff++ << 32;
            tmp |= *buff++ << 40;
            tmp |= *buff++ << 48;
            tmp |= *buff++ << 56;
            *reg = tmp;
        }
    }
    else{
        for(i = num; i > 0 ; i--){
            *reg = *ptr++;
        }
    }
    return SD_OK;
}
```

4.3.7 sddev_get_clockdiv

sddev_get_clockdiv

ターゲット CPU インタフェース関数

クロック分周比の取得

書式

```
#include "r_sdif.h"

uint32_t sddev_get_clockdiv(int32_t sd_port, int32_t clock);

int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
int32_t clock            | 設定するクロック周波数
                           SD_CLK_400KHZ
                           SD_CLK_1MHZ
                           SD_CLK_5MHZ
                           SD_CLK_10MHZ
                           SD_CLK_20MHZ
                           SD_CLK_25MHZ
                           SD_CLK_50MHZ
```

戻り値

クロック分周比

解説

カードへ供給するクロック（SDCLK）の分周比を決定し、その値を返します。
引数 clock には、設定すべきクロック周波数の上限が設定されます。SD ホストコントローラへの動作クロックから設定するクロック周波数 clock に近い値になるように分周比を決定します。

表 4.15 にクロック分周比と SDCLK を示します。
表 4.15 に示すクロック分周比マクロ定義を戻り値としてください。

表 4.15 クロック分周比

分周比	クロック分周比マクロ定義	動作クロック 132MHz 時の SDCLK
4	SD_DIV_4	33MHz
8	SD_DIV_8	16.5MHz
16	SD_DIV_16	8.3MHz
32	SD_DIV_32	4.1MHz
64	SD_DIV_64	2.1MHz
128	SD_DIV_128	1.0MHz
256	SD_DIV_256	515.6KHz
512	SD_DIV_512	257.8KHz

注意

引数 clock が SD_CLK_400KHZ の場合、SDCLK の下限は 100kHz です。
カードへ供給するクロックが、引数 clock のクロック周波数を超えるクロック分周比は戻り値とししないでください。

作成例

```
/* クロック分周比の取得例（動作クロック 132MHz 時） */
#include "r_sdif.h"

uint32_t sddev_get_clockdiv(int32_t sd_port, int32_t clock)
{
    uint32_t div;

    switch(clock){
    case SD_CLK_50MHZ:
        div = SD_DIV_4;          /* 132MHz/4 = 33MHz      */
        break;
    case SD_CLK_25MHZ:
        div = SD_DIV_8;          /* 132MHz/8 = 16.5MHz   */
        break;
    case SD_CLK_20MHZ:
        div = SD_DIV_8;          /* 132MHz/8 = 16.5MHz   */
        break;
    case SD_CLK_10MHZ:
        div = SD_DIV_16;         /* 132MHz/16 = 8.3MHz    */
        break;
    case SD_CLK_5MHZ:
        div = SD_DIV_32;         /* 132MHz/32 = 4.1MHz    */
        break;
    case SD_CLK_1MHZ:
        div = SD_DIV_256;        /* 132MHz/256 = 515.6kHz */
        break;
    case SD_CLK_400KHZ:
        div = SD_DIV_512;        /* 132MHz/512 = 257.8kHz */
        break;
    default:
        div = SD_DIV_512;        /* 132MHz/512 = 257.8kHz */
        break;
    }
    return div;
}
```


4.3.8 sddev_set_port

sddev_set_port

ターゲット CPU インタフェース関数

カード用ポート設定

書式

```
#include "r_sdif.h"

int32_t sddev_set_port(int32_t sd_port, int32_t mode);
```

int32_t sd_port	I	SDHI チャンネル番号 (0 または 1)
int32_t mode	I	設定するポートモード
		SD_PORT_SERIAL : シリアルポート設定
		SD_PORT_PARALLEL : パラレルポート設定

戻り値

SD_OK	: 正常終了
SD_ERR	: エラー終了

解説 実装不要です。常に戻り値として SD_OK を返してください。

4.3.9 sddev_int_wait

sddev_int_wait

ターゲット CPU インタフェース関数

カード割り込み待ち

書式

```
#include "r_sdif.h"

int32_t sddev_int_wait(int32_t sd_port, int32_t time);
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    int32_t time             | タイムアウト時間 (msec)
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

カードとのプロトコル通信時の割り込み待ち処理を行いません。
割り込み要求を確認できた場合は、SD_OK 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、SD_ERR を返します。

割り込み待ち処理は、ソフトウェアポーリングによる処理と、割り込みを使用した処理のどちらかで実装します。

割り込み要求は sd_check_int 関数により確認できます。本関数内で sd_check_int 関数を呼び出して割り込み要求が発生しているかを確認してください。

注意

本関数呼び出し前にプロトコル通信割り込みが発生している場合があります。特に H/W 割り込みを使用する場合は、本関数呼び出しに前に割り込み処理および割り込みコールバック関数が呼び出されている場合がありますので注意してください。

sd_mount 関数実行時、本関数を時間計測用処理に使用します。そのため、割り込みを確認できない場合は、必ずタイムアウト時間 **time** ミリ秒以上の時間経過後 SD_ERR を返すようにしてください。

作成例

```
/* カード割り込み待ち例 */
#include "r_sdif.h"

int32_t sddev_int_wait(int32_t sd_port, int32_t time)
{
    /* 割り込み要求待ち */
    while(1){
        if(sd_check_int(sd_port) == SD_OK){
            /* 割り込み要求あり */
            break;
        }
        /* タイムアウト処理など */
    }
    return SD_OK;
}
```

4.3.10 sddev_loc_cpu

sddev_loc_cpu

ターゲット CPU インタフェース関数

カード割り込み禁止

書式

```
#include "r_sdif.h"
int32_t sddev_loc_cpu(int32_t sd_port)
    int32_t sd_port          I    SDHI チャンネル番号 (0 または 1)
```

戻り値

SD_OK : 正常終了

解説

実装不要です。常に戻り値として SD_OK を返してください。

4.3.11 sddev_unl_cpu

sddev_unl_cpu

ターゲット CPU インタフェース関数

カード割り込み許可

書式 `#include "r_sdif.h"`
 `int32_t sddev_unl_cpu(int32_t sd_port)`
 `int32_t sd_port` **I** SDHI チャンネル番号 (0 または 1)

戻り値 `SD_OK` : 正常終了

解説 実装不要です。常に戻り値として `SD_OK` を返してください。

4.3.12 sddev_init_dma

sddev_init_dma

ターゲット CPU インタフェース関数

データ転送用 DMA の初期化

書式

```
#include "r_sdif.h"

int32_t sddev_init_dma(int32_t sd_port, uint32_t buff, int32_t dir);
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    uint32_t buff            | 転送するバッファの先頭アドレス
    int32_t dir              | 転送方向
                             | 0 : SD_BUF0 レジスタ → バッファ
                             | 1 : バッファ → SD_BUF0 レジスタ
```

戻り値

SD_OK : 正常終了

SD_ERR : エラー終了

解説

カードアクセスのための DMA 設定を行います。

表 4.16 に DMA コントローラ設定例を示します。

表 4.16 DMA コントローラ設定

DMA コントローラ	転送方向	
	dir=0	dir=1
DMAC チャンネル選択	SD アップストリーム	SD ダウンストリーム
DMAC バス幅選択	64 ビット固定	
ディスティネーションアドレス／ソースアドレス (8 バイト単位)	転送するバッファの先頭アドレス	

注意

sd_mount 関数の動作モードにてソフトウェア転送を選択した場合、本関数は使用しませんので空関数として実装してください。

4.3.13 sddev_wait_dma_end

sddev_wait_dma_end

ターゲット CPU インタフェース関数

データ転送用 DMA 転送完了待ち

書式

```
#include "r_sdif.h"

int32_t sddev_wait_dma_end(int32_t sd_port, int32_t cnt);
    int32_t sd_port          | SDHI チャンネル番号 (0 または 1)
    int32_t cnt              | DMA 転送バイト数
```

戻り値

SD_OK : 正常終了
SD_ERR : エラー終了

解説

sddev_init_dma 関数で設定した DMA 転送が完了するのを本関数内で待ちます。

DMA 転送完了待ちの方法はシステムに依存します。DMA 転送完了割り込みや、DMA コントローラのレジスタをポーリングするなどシステムに応じて実装してください。

DMA 転送が完了した場合に、SD_OK を返し、関数処理を終了してください。

タイムアウト時間は規定しませんが、DMA 転送バイト数 cnt を元にシステムに応じたタイムアウト時間を設定することを推奨します。タイムアウト時間内に全データの転送が完了しなかった場合や DMA 転送途中での処理中断など転送処理を完了できなかった場合は、SD_ERR を返してください。

注意

sd_mount 関数の動作モードにてソフトウェア転送を選択した場合、本関数は使用しませんので空関数として実装してください。

4.3.14 sddev_disable_dma

sddev_disable_dma

ターゲット CPU インタフェース関数

データ転送用 DMA の禁止

書式	<pre>#include "r_sdif.h" int32_t sddev_disable_dma(int32_t sd_port); int32_t sd_port I SDHI チャンネル番号 (0 または 1)</pre>		
戻り値	SD_OK	:	正常終了
	SD_ERR	:	エラー終了
解説	sddev_init_dma 関数で設定したカードアクセスのための DMA を禁止に設定します。 本関数は、sddev_wait_dma_end 関数による DMA 転送処理の終了確認後呼び出されます。		
注意	sd_mount 関数の動作モードにてソフトウェア転送を選択した場合、本関数は使用しませんので空関数として実装してください。		

4.3.15 sddev_reset_dma

sddev_reset_dma

ターゲット CPU インタフェース関数

DMA のリセット

書式

```
#include "r_sdif.h"
int32_t sddev_reset_dma(int32_t sd_port);
        int32_t sd_port          |    SDHI チャンネル番号 (0 または 1)
```

戻り値

SD_OK	: 正常終了
SD_ERR	: エラー終了

解説 DMA コントローラのリセットを行います。

注意

4.3.16 sddev_finalize_dma

sddev_finalize_dma

ターゲット CPU インタフェース関数

DMA の終了処理

書式

```
#include "r_sdif.h"
int32_t sddev_finalize_dma(int32_t sd_port);
    int32_t sd_port          |   SDHI チャンネル番号 (0 または 1)
```

戻り値

SD_OK	: 正常終了
SD_ERR	: エラー終了

解説 DMA コントローラの終了処理を行います。

注意

4.4 デバイスドライバ関数

本節では、FatFs（汎用 FAT ファイルシステム）から呼び出されるデバイスドライバ関数として SD ドライバを組み込む方法について説明します。FatFs の仕様を確認の上、必要があればシステムにあわせて変更してください。

他のファイルシステムに SD ドライバを組み込む場合は、使用するファイルシステムの仕様に合わせてデバイスドライバ関数を作成してください。

表 4.17 にデバイスドライバ関数一覧を示します。

表 4.17 デバイスドライバ関数一覧

デバイスドライバ関数名	機能概要
disk_status	デバイスの状態取得
disk_initialize	デバイスの初期化
disk_read	セクタデータの読み出し（論理セクタ単位）
disk_write	セクタデータの書き込み（論理セクタ単位）
disk_ioctl	その他のデバイス制御
get_fattime	日付と時刻の取得

4.4.1 disk_status

disk_statusデバイスドライバ関数

デバイスの状態取得

書式	<pre>#include "diskio.h" DSTATUS disk_status (BYTE pdrv) I 物理ドライブ番号(0～9)</pre>
戻り値	<pre>STA_NOINIT : デバイスが初期化されていないことを示すフラグ STA_NODISK : メディアが存在しないことを示すフラグ STA_PROTECT : メディアがライトプロテクトされていることを示すフラグ</pre>
解説	サンプルプログラムでは、デバイスの状態 (STA_NODISK、STA_NOINIT、STA_PROTECT) を返します。
作成例	<pre>pdrv に 2 以上が設定された場合は、STA_NODISK STA_NOINIT を返却します。 /* デバイスの状態取得例 */ #include "diskio.h" DSTATUS disk_status (BYTE pdrv) { DSTATUS ret; int32_t chk; ret = 0; /* 物理ドライブ番号が有効かを確認 */ if (pdrv > 1){ ret = (STA_NODISK STA_NOINIT); } else { /* カードが挿入されているかを確認 */ chk = sd_check_media((int32_t)pdrv); if (chk != SD_OK){ ret = (STA_NODISK STA_NOINIT); /* メディアが存在しない */ } /* カードタイプ情報がすでに取得されているかを確認 */ else if (sd_info[pdrv].type == SD_MEDIA_UNKNOWN){ ret = STA_NOINIT; /* デバイスが初期化されていない */ } /* カードがライトプロテクトかを確認 */ else if (sd_info[pdrv].iswp != SD_WP_OFF){ ret = STA_PROTECT; /* メディアがライトプロテクトされている */ } else { ret = 0; } } return ret; }</pre>

4.4.2 disk_initialize

disk_initialize

デバイスドライバ関数

デバイスの初期化

書式	<pre>#include "diskio.h" DSTATUS disk_initialize (BYTE pdrv) BYTE pdrv I 物理ドライブ番号(0~9)</pre>
戻り値	<p>STA_NOINIT : デバイスが初期化されていないことを示すフラグ</p> <p>STA_NODISK : メディアが存在しないことを示すフラグ</p> <p>STA_PROTECT : メディアがライトプロテクトされていることを示すフラグ</p>
解説	<p>サンプルプログラムでは、ドライバの初期化、カードのマウントを行います。</p> <p>本関数は FatFs の管理下であり、自動マウント動作により必要に応じて呼び出されます。アプリケーションからの呼び出しは禁止です。</p> <p>再初期化が必要なときは、FatFs の API 関数 (f_mount()) を使用してください。</p>
作成例	<pre>/* デバイスの初期化例 */ #include "diskio.h" /* サンプリングクロックコントローラ先のアドレスを設定 */ #define SDCFG_IP0_BASE (0xE8227000uL) /* チャンネル 0 */ #define SDCFG_IP1_BASE (0xE8229000uL) /* チャンネル 1 */ /* SD ドライババッファ領域サイズ */ #define SD_RW_BUFF_SIZE (1 * 1024) typedef struct { uint16_t type; int32_t iswp; } SD_INFO; /* SD ドライバワーク領域定義 */ uint32_t SDTestWork[2][SD_SIZE_OF_INIT/sizeof(uint32_t)]; /* SD ドライババッファ領域定義 */ uint32_t test_sd_rw_buff[2][SD_RW_BUFF_SIZE/sizeof(uint32_t)]; SD_INFO sd_info[2] = { { SD_MEDIA_UNKNOWN, SD_WP_OFF }, { SD_MEDIA_UNKNOWN, SD_WP_OFF } }; static const uint32_t sd_base_addr[2] = { SDCFG_IP0_BASE, SDCFG_IP1_BASE };</pre>

```
DSTATUS disk_initialize (BYTE pdrv)
{
    DSTATUS    ret;
    int32_t    chk;
    uint16_t    type;

    ret = (STA_NODISK | STA_NOINIT);

    /* 物理ドライブ番号が有効かを確認 */
    if (pdrv > 1){
        return ret;
    }

    /* カード情報を初期化 */
    sd_info[pdrv].type = SD_MEDIA_UNKNOWN;
    sd_info[pdrv].iswp = SD_WP_OFF;

    /* SD ドライバの初期化 */
    chk = sd_init((int32_t)pdrv
                  ,sd_base_addr[pdrv]
                  ,&SDTestWork[pdrv][0]
                  ,SD_CD_SOCKET);
    if (chk != SD_OK){
        return ret;
    }

    /* カードが挿入されているかを確認 */
    chk = sd_check_media((int32_t)pdrv);
    if (chk != SD_OK){
        return ret;
    }
    else{
        /* カードが挿入されている */
        ret &= ~STA_NODISK;

        /* カード挿抜割り込み設定 */
        chk = sd_cd_int((int32_t)pdrv, SD_CD_INT_ENABLE, NULL);
        if (chk != SD_OK){
            return ret;
        }

        /* SD ドライバ用バッファの設定 */
        chk = sd_set_buffer((int32_t)pdrv
                           ,&test_sd_rw_buff[pdrv][0]
                           ,SD_RW_BUFF_SIZE);
        if (chk != SD_OK){
            return ret;
        }
    }
}
```

```
/* H/W 割り込み、DMA 転送(64 ビット固定)、default-Speed カード対応、
   High-Capacity カード対応、eXtended-Capacity カード対応でカードをマウント */
chk = sd_mount((int32_t)pdrv
               ,SD_MODE_HWINT|SD_MODE_DMA|SD_MODE_DS|SD_MODE_VER2X
               ,SD_VOLT_3_3);
if (chk != SD_OK){
    return ret;
}

/* カードタイプと動作モードの取得 */
chk = sd_get_type((int32_t)pdrv, &type, NULL, NULL);
if (chk != SD_OK){
    return ret;
}
sd_info[pdrv].type = type;

/* ライトプロテクト情報の取得 */
chk = sd_iswp((int32_t)pdrv);
if (chk == SD_ERR){
    return ret;
}
sd_info[pdrv].iswp = chk;

/* 初期化完了 */
ret &= ~STA_NOINIT;

/* カードがライトプロテクトかを確認 */
if (sd_info[pdrv].iswp != SD_WP_OFF){
    ret |= STA_PROTECT;    /* メディアがライトプロテクトされている */
}
}
return ret;
}
```

4.4.3 disk_read

disk_read

デバイスドライバ関数

セクタデータの読み出し（セクタ単位）

書式

```
#include "diskio.h"

DRESULT disk_read (BYTE pdrv, BYTE* buff, DWORD sector, UINT count)
    BYTE pdrv          I   物理ドライブ番号(0~9)
    BYTE* buff          O   読み出しバッファへのポインタ
    DWORD sector        I   読み出し開始セクタ番号
    UINT count          I   読み出すセクタ数(1~128)
```

戻り値

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

解説

物理ドライブ番号 **pdrv** で指定したドライブからセクタデータの読み出しを行います。
開始セクタ番号 **sector** で指定したセクタから読み出すセクタ数 **count** 分のセクタデータを読み出し、ポインタ **buff** の示す領域に格納します。

作成例

```
/* セクタデータの読み出し例 */
#include "diskio.h"

DRESULT disk_read (BYTE pdrv, BYTE *buff, DWORD sector, UINT count)
{
    DRESULT ret;
    int32_t chk;

    ret = RES_PARERR;

    /* 物理ドライブ番号が有効かを確認 */
    if (pdrv > 1){
        return ret;
    }
    /* カードからセクタデータを読み出し */
    chk = sd_read_sect(pdrv, buff, sector, count);
    if (chk == SD_OK){
        ret = RES_OK;
    }
    else{
        ret = RES_ERROR;
    }
    return ret;
}
```

4.4.4 disk_write

disk_write

デバイスドライバ関数

セクタデータの書き込み（セクタ単位）

書式

```
#include "diskio.h"

DRESULT disk_write (BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)
    BYTE pdrv          | 物理ドライブ番号(0~9)
    const BYTE* buff    | 書き込むデータへのポインタ
    DWORD sector        | 書き込み開始セクタ番号
    UINT count          | 書き込むセクタ数(1~128)
```

戻り値

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

解説

物理ドライブ番号 **pdrv** で指定したドライブへセクタデータの書き込みを行います。
ポインタ **buff** で示されるデータを、開始セクタ番号 **sector** で指定したセクタへ書き込みセクタ数 **count** 分書き込みます。

作成例

```
/* セクタデータの書き込み例 */
#include "diskio.h"

DRESULT disk_write (BYTE pdrv, const BYTE *buff, DWORD sector, UINT count)
{
    DRESULT ret;
    int32_t chk;

    ret = RES_PARERR;
    /* 物理ドライブ番号が有効かを確認 */
    if (pdrv > 1){
        return ret;
    }
    /* カードへセクタデータを書き込み */
    chk = sd_write_sect(pdrv
                        ,(uint8_t *)buff
                        ,sector
                        ,count
                        ,SD_WRITE_OVERWRITE);

    if (chk == SD_OK){
        ret = RES_OK;
    }
    else{
        ret = RES_ERROR;
    }
    return ret;
}
```


4.4.5 disk_ioctl

disk_ioctl

デバイスドライバ関数

その他のデバイス制御

書式

```
#include "diskio.h"

DRESULT disk_ioctl (BYTE pdrv, BYTE cmd, void* buff)
    BYTE pdrv      |   物理ドライブ番号(0～9)
    BYTE cmd       |   制御コマンド
    void* buff     |   データ受け渡しバッファ
```

戻り値

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

解説

サンプルプログラムでは、全てのコマンドに対して処理をせず、RES_OK を返却します。

作成例

```
/* その他のデバイス制御例 */
#include "diskio.h"

DRESULT disk_ioctl (BYTE pdrv, BYTE cmd, void *buff)
{
    return RES_OK;
}
```

4.4.6 get_fattime

get_fattime

デバイスドライバ関数

日付と時刻の取得

書式 `#include "diskio.h"`
 `DWORD get_fattime (void)`

戻り値 日付と時刻

解説 現在のローカルタイムを **DWORD** 値にパックして返します。
 ビットフィールドは下記の通りです。
 bit 31:25 1980 年を起点とした年を 0~127 でセット
 bit 24:21 月を 1~12 の値でセット
 bit 20:16 日を 1~31 の値でセット
 bit 15:11 時を 0~23 の値でセット
 bit 10:5 分を 0~59 の値でセット
 bit 4:0 秒/2 を 0~29 の値でセット

サンプルプログラムでは、日付と時刻情報は設定せず、0x00000000 を返却します。

作成例 `/* 日付と時刻の取得例 */`
 `#include "diskio.h"`

 `DWORD get_fattime(void)`
 `{`
 `return 0x00000000;`
 `}`

5. コンフィグオプション

表 5.1 に SD ドライバのコンフィグオプションについて説明します。

表 5.1 コンフィグオプション

定義	内容
SD_CD_ENABLED	SD カード検出オプション。 SD カード検出を有効に設定します。
SD_CD_DISABLED	SD カード検出オプション。 SD カード検出を無効に設定します。SD カード検出無効時は、常時装填となります。
SD_WP_ENABLED	ライトプロテクト信号検出オプション。 ライトプロテクト信号検出を有効に設定します。
SD_WP_DISABLED	ライトプロテクト信号検出オプション。 ライトプロテクト信号検出を無効に設定します。ライトプロテクト信号無効時は、常時ライトプロテクト信号 OFF となります。
SD_CB_UNUSED	SD カード検出コールバック関数設定。 SD カード検出コールバック関数設定は使用されません。
SD_CB_USED	SD カード検出コールバック関数設定。 SD カード検出コールバック関数設定は使用されます。

6. アプリケーション作成時の制限事項

本章では、SD ドライバについての各種注意事項を説明します。

6.1 SD ドライバ使用時の注意事項

SD ドライバを使用してアプリケーションプログラムを作成する時の注意事項を次に示します。

■ 予約語

SD ドライバでは、次のキーワードが C 言語でアプリケーションを作成する場合の予約語となっています。

「sd_の文字列で始まるキーワード」

「_sd_の文字列で始まるキーワード」

アセンブリ言語でアプリケーションプログラムを作成するときは次のキーワードが予約語となります。

「_sd_の文字列で始まるキーワード」

「__sd_の文字列で始まるキーワード」

■ 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 ドライバの引数の設定規則やレジスタの保証規則は、クロスツールキット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。

■ OS 使用時の注意事項

SD ドライバはリエントラント性（同時に複数のプログラム（タスク）から利用できる機構）を保証していません。従って、リアルタイム OS を使用したアプリケーションでは、OS の機能を使用して排他制御を行う必要があります。

■ 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。割り込みコールバック関数内では割り込みハンドラ内で使用できる関数をご使用ください。なお、sd_int_handler 関数を除くすべての SD ドライバライブラリ関数は、割り込みハンドラ内では使用できません。

7. サンプルプログラム

本章では、サンプルプログラムを導入するための情報を提供します。

7.1 機能概要

本サンプルプログラムは、RZ/A2 SD ドライバのライブラリ関数を使用して、SD の基本的な制御を実現するために8種類のサンプル処理を準備しています。サンプル処理は、SW3の押下または、PC のターミナルからコマンドを入力して実行します。

また、SD カードとの接続の確立および切断は、SD カードスロットへSD カードを挿抜により、自動的に実行します。

表 7.1 にサンプル処理の一覧を示します。

表 7.1 サンプル処理の一覧

サンプル処理	サンプル処理の内容	コマンド
ヘルプ	使用可能なサンプルコマンドの一覧を表示。	HELP
一覧表示	指定されたディレクトリ配下のファイル、サブディレクトリの一覧を表示。	DIR (ディレクトリ名)
ファイル内容表示	指定されたファイルの内容を表示。 指定されたファイルが存在しない場合はエラーを表示。	TYPE (ファイル名)
ファイル書き込み	指定されたファイルに、文字列「Renesas FAT/exFAT sample.」を書き込み。 指定されたファイルが存在しない場合は新しいファイルを作成し、文字列を書き込み。	WRITE (ファイル名)
新規ファイル作成	指定されたファイルを新規で作成。既に存在する場合は、エラーを表示。	CREATE (ファイル名)
ファイル削除	指定されたファイルを削除。	DEL (ファイル名)
新規ディレクトリ作成	指定されたディレクトリを新規で作成。既に存在する場合は、エラーを表示。	MKDIR (ディレクトリ名)
ディレクトリ削除	指定されたディレクトリを削除。 ディレクトリ直下にファイル等が存在する場合は、エラーを表示。	RMDIR (ディレクトリ名)

7.2 動作環境

サンプルプログラムの動作環境を図 7.1 に示します。

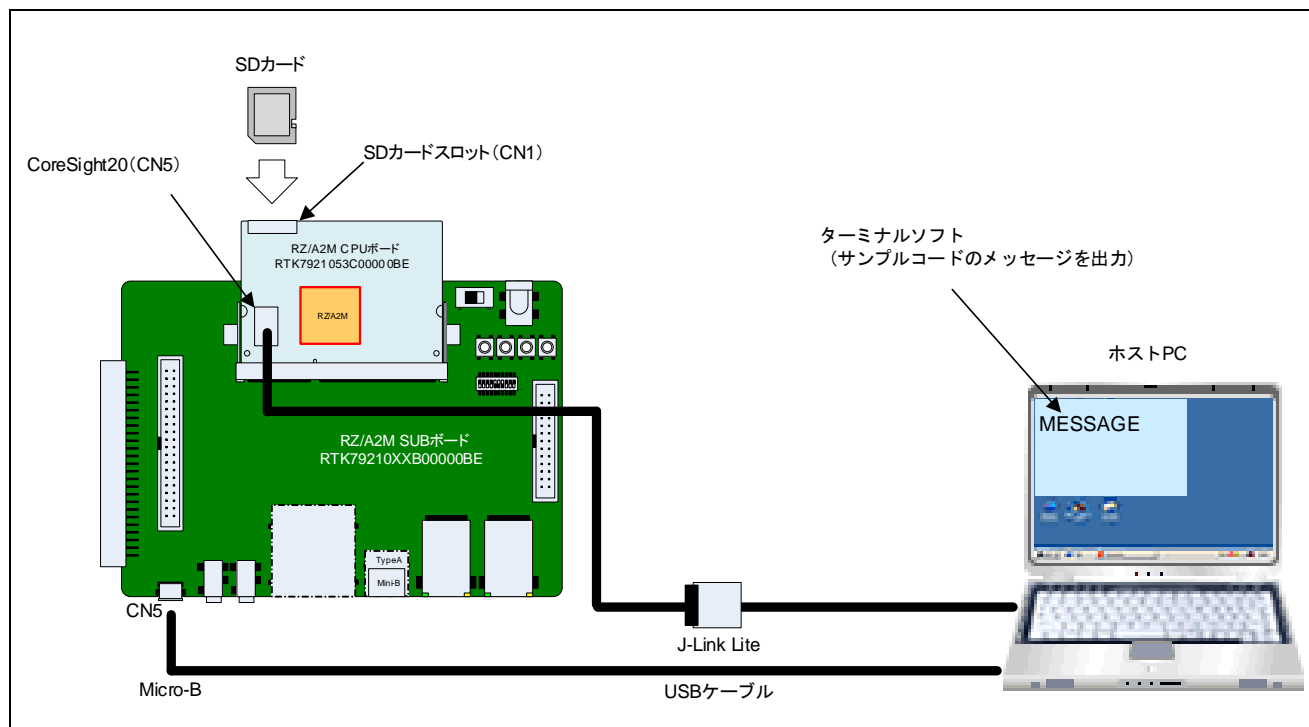


図 7.1 動作環境

7.3 動作確認条件

サンプルプログラムの動作確認条件を表 7.2 に示します。

表 7.2 動作確認条件

項目	内容
使用マイコン	RZ/A2M
動作周波数（注）	CPU クロック（I ϕ ）：528MHz 画像処理クロック（G ϕ ）：264MHz 内部バスクロック（B ϕ ）：132MHz 周辺クロック 1（P1 ϕ ）：66MHz 周辺クロック 0（P0 ϕ ）：33MHz QSPI0_SPCLK：66MHz CKIO：132MHz
動作電圧	電源電圧（I/O）：3.3V 電源電圧（1.8/3.3V 切替 I/O（PVcc_SPI））：3.3V 電源電圧（内部）：1.2V
統合開発環境	e2 studio V7.5.0
エミュレータ	J-Link Lite
C コンパイラ	GNU Arm Embedded Toolchain 6-2017-q2-update コンパイラオプション（ディレクトリパスの追加は除く） Release: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Os -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -Wstack-usage=100 -fabi-version=0 Hardware Debug: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -g3 -Wstack-usage=100 -fabi-version=0
動作モード	ブートモード 3 （シリアルフラッシュブート 3.3V 品）
ターミナルソフトの通信設定	<ul style="list-style-type: none"> 通信速度：115200bps データ長：8 ビット パリティ：なし ストップビット長：1 ビット フロー制御：なし
使用ボード	RZ/A2M CPU ボード RTK7921053C00000BE RZ/A2M SUB ボード RTK79210XXB00000BE

使用デバイス (ボード上で使用する機能)	<ul style="list-style-type: none">シリアルフラッシュメモリ (SPI マルチ I/O バス空間に接続) メーカー名 : Macronix 社、型名 : MX25L51245GXDRL78/G1C (USB 通信とシリアル通信を変換し、ホスト PC との通信に使用)LED1SW3SD カード Transcend 8GB (SDHC,class10) ファイルシステム : FAT32
-------------------------	---

【注】 クロックモード 1 (EXTAL 端子からの 24MHz のクロック入力) で使用時の動作周波数です。

7.4 使用端子と機能

サンプルプログラムの使用する端子と機能を、表 7.3～表 7.4 に示します。

表 7.3 使用端子と機能 1（チャンネル 0 の SD/MMC カードスロット）

端子名	入出力	機能	備考
SD0_CLK	出力	SD クロック SD クロック出力端子	3.3V 固定
SD0_CMD	入出力	SD コマンド SD コマンド出力/レスポンス入力信号	3.3V 固定
SD0_DAT0	入出力	SD データ 0 データ[Bit0]信号	3.3V 固定
SD0_DAT1	入出力	SD データ 1 データ[Bit1]/SDIO Interrupt 信号	3.3V 固定
SD0_DAT2	入出力	SD データ 2 データ[Bit2]/Read Wait 信号	3.3V 固定
SD0_DAT3	入出力	SD データ 3 データ[Bit3]/カード検出信号	3.3V 固定
SD0_CD	入力	SD カード検出 SD カード検出入力信号	オプション 3.3V 固定
SD0_WP	入力	SD ライトプロテクト SD ライトプロテクト入力信号	オプション 3.3V 固定
SD0_RST	出力	SD リセット SD リセット出力信号	3.3V 固定
PD_1	出力	SD コマンド/SD データ 0～3 の入出力電圧の切換	High = 3.3V 固定
PJ_1	入力	SW3 キー入力	
P6_0	出力	LED1(RED)	
PC_1	出力	LED1(Yellowish-green)	

表 7.4 使用端子と機能 2（チャンネル 1 の SD/MMC カードスロット）

端子名	入出力	機能	備考
SD1_CLK	出力	SD クロック SD クロック出力端子	3.3V 固定
SD1_CMD	入出力	SD コマンド SD コマンド出力/レスポンス入力信号	3.3V 固定
SD1_DAT0	入出力	SD データ 0 データ[Bit0]信号	3.3V 固定
SD1_DAT1	入出力	SD データ 1 データ[Bit1]/SDIO Interrupt 信号	3.3V 固定
SD1_DAT2	入出力	SD データ 2 データ[Bit2]/Read Wait 信号	3.3V 固定
SD1_DAT3	入出力	SD データ 3 データ[Bit3]/カード検出信号	3.3V 固定
SD1_CD	入力	SD カード検出 SD カード検出入力信号	オプション 3.3V 固定
SD1_WP	入力	SD ライトプロテクト SD ライトプロテクト入力信号	オプション 3.3V 固定

7.5 メモリサイズ

SD ドライバのメモリ使用量を、表 7.5 に示します。

表 7.5 SD ドライバのメモリ使用量

ROM (KB)	RAM (KB)	スタック (KB)
21.0	0.7	0.3

7.6 導入手順

サンプルプログラムの導入手順は以下のとおりです。

1. 動作環境準備

「7.2 動作環境」を参考に、サンプルプログラムを動作させる環境を準備します。

2. サンプルプログラムのビルド

ビルドを実行します。

3. デバッグ起動

デバッグを起動し、サンプルプログラムをロードします。

4. SD カードとの接続確立

サンプルプログラム実行することにより、自動的に SD カードのマウントを実行し、SD カードの接続を確立します。SD カードの接続の確立が失敗した場合は、LED1(RED)が点灯します。

SD カードのマウントは、電源 ON 時またはカード挿入時に自動的に実行します。

表 7.6 LED1 点灯パターン

状態	LED1(RED)	LED1(Yellowish-green)
正常時 ・ SD カードの接続の確立が正常 ・ SD カードの接続の切断が正常 など	消灯	消灯
エラー発生 ・ SD カードの接続の確立が失敗 ・ SD カードアクセス中にカードを抜き取る など	点灯	消灯
SD カードへのアクセス中	消灯	点滅

5. SD カードへのアクセス

SW3 を短押しすることにより、SD カードへの書き込みおよび読み出しを実行します。SD カードへのアクセス中は、LED1(Yellowish-green)が 10 回点滅します。SD カードへのアクセス中に、SD カードを抜き取った場合は、エラーが発生し、LED1(RED)が点灯します。

表 7.7 SW3 押下パターン

SW3 押下パターン	条件	サンプル処理
SW3 短押し	SW3 の押下 5 秒未満	SD カードへのアクセス
SW3 長押し	SW3 の押下 5 秒以上	[OS 版] SD カードへのアクセス
		[OS レス版] ターミナルコマンドが有効な状態へ移行。 以降、SW3 押下は無効。

6. SD カードとの接続切断

SD カードを抜き取るにより、自動的に SD カードのアンマウントを実行し、SD カードの接続を切断します。

7. SD カードとの再接続確立

SD カードを挿入することにより、自動的に SD カードの再マウントを実行し、SD カードの接続を確立します。

8. ターミナルコマンド

OS 版では、SD カードとの接続確立後、ウェルカムメッセージが表示され、ターミナルよりコマンドを使用することができます。使用可能なコマンドは、「表 7.1 サンプル処理の一覧」を参照。ただし、SW3 押下による SD カードへのアクセス中のコマンド発行は、無効です。

OS レス版では、SD カードとの接続確立後、SW3 を長押しすることにより、ウェルカムメッセージが表示され、ターミナルコマンドが受付可能になります。以降、SW3 押下は無効となります。再度 SW3 押下を有効にする場合は、SD カードを抜き取ってから再度挿入してください。

図は、exFAT 有効時の例です。以降同様。

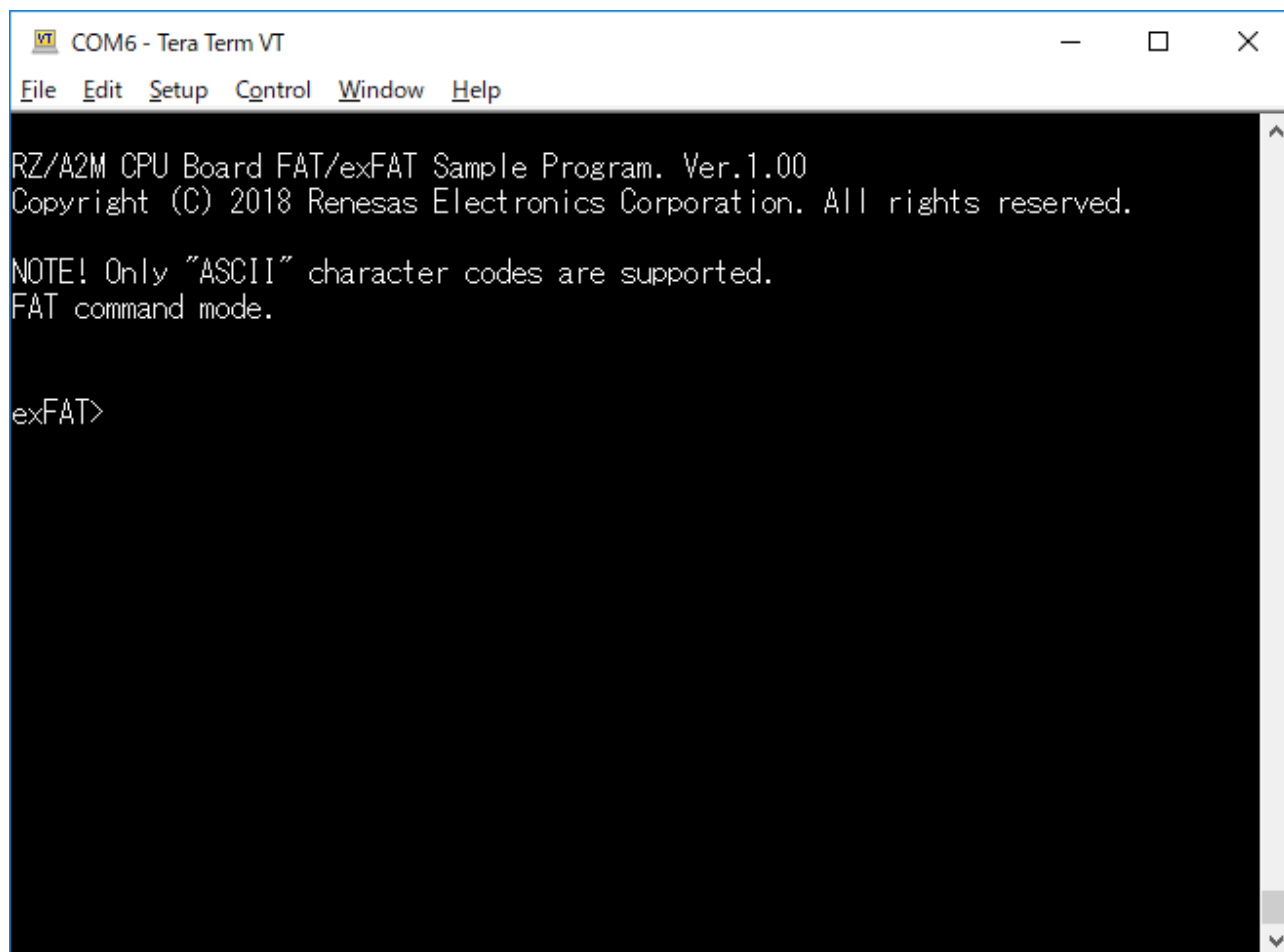
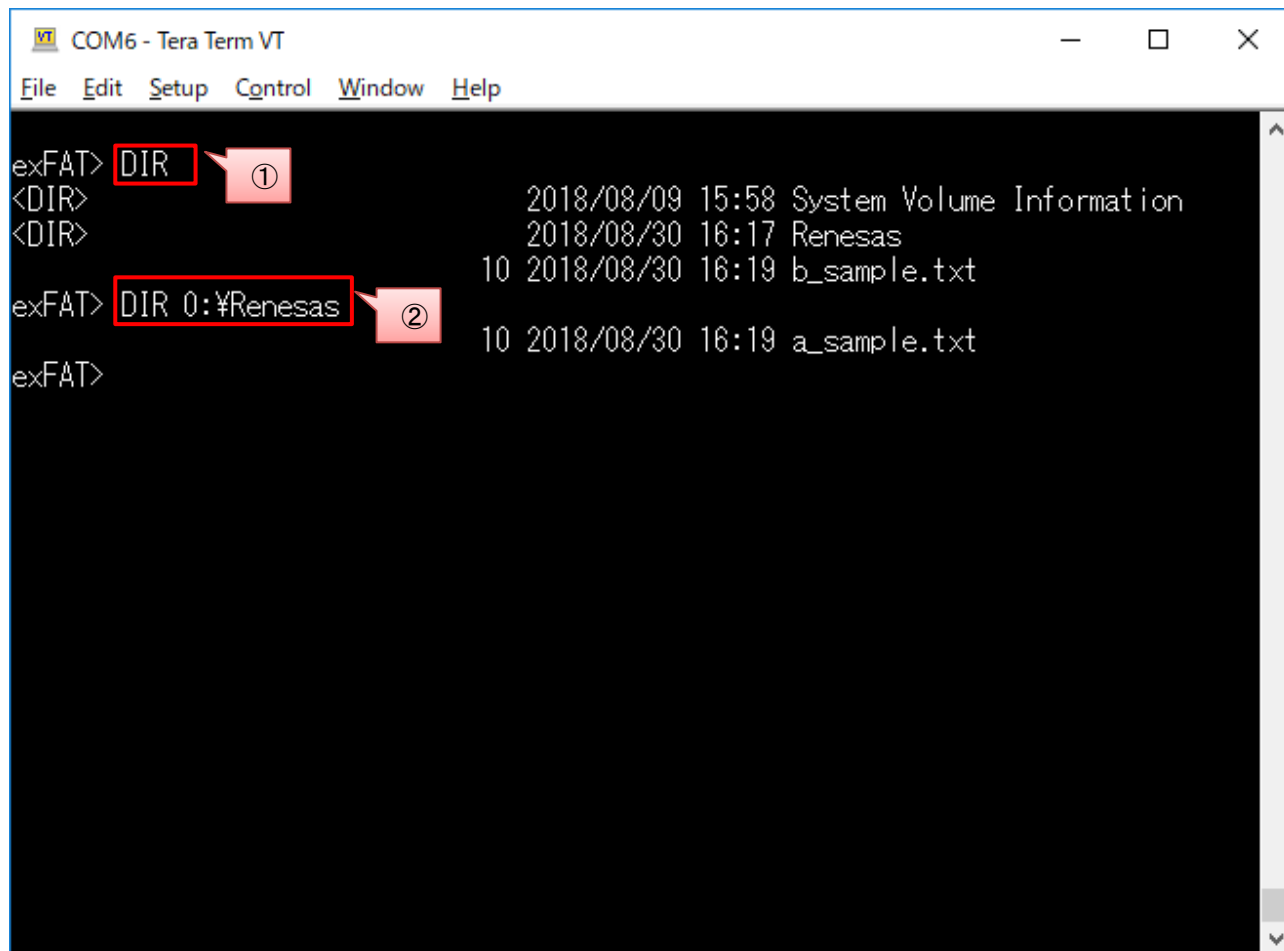


図 7.2 ウェルカムメッセージ表示例

9. 一覧表示

ターミナルより、コマンド「DIR」を発行（図中①（ディレクトリ「ルート」指定）、図中②（ディレクトリ「Renesas」指定）し、接続している SD カードの指定されたディレクトリ配下のファイル、サブディレクトリの一覧を表示します。



```
COM6 - Tera Term VT
File Edit Setup Control Window Help

exFAT> DIR
<DIR>
2018/08/09 15:58 System Volume Information
2018/08/30 16:17 Renesas
10 2018/08/30 16:19 b_sample.txt
exFAT> DIR 0:¥Renesas
10 2018/08/30 16:19 a_sample.txt
exFAT>
```

図 7.3 「DIR」 コマンド発行例

10. ファイル内容表示

ターミナルより、コマンド「TYPE」を発行（図中③）し、接続している SD カードの指定されたファイルの内容（図中④ HEX 表示）を表示します。

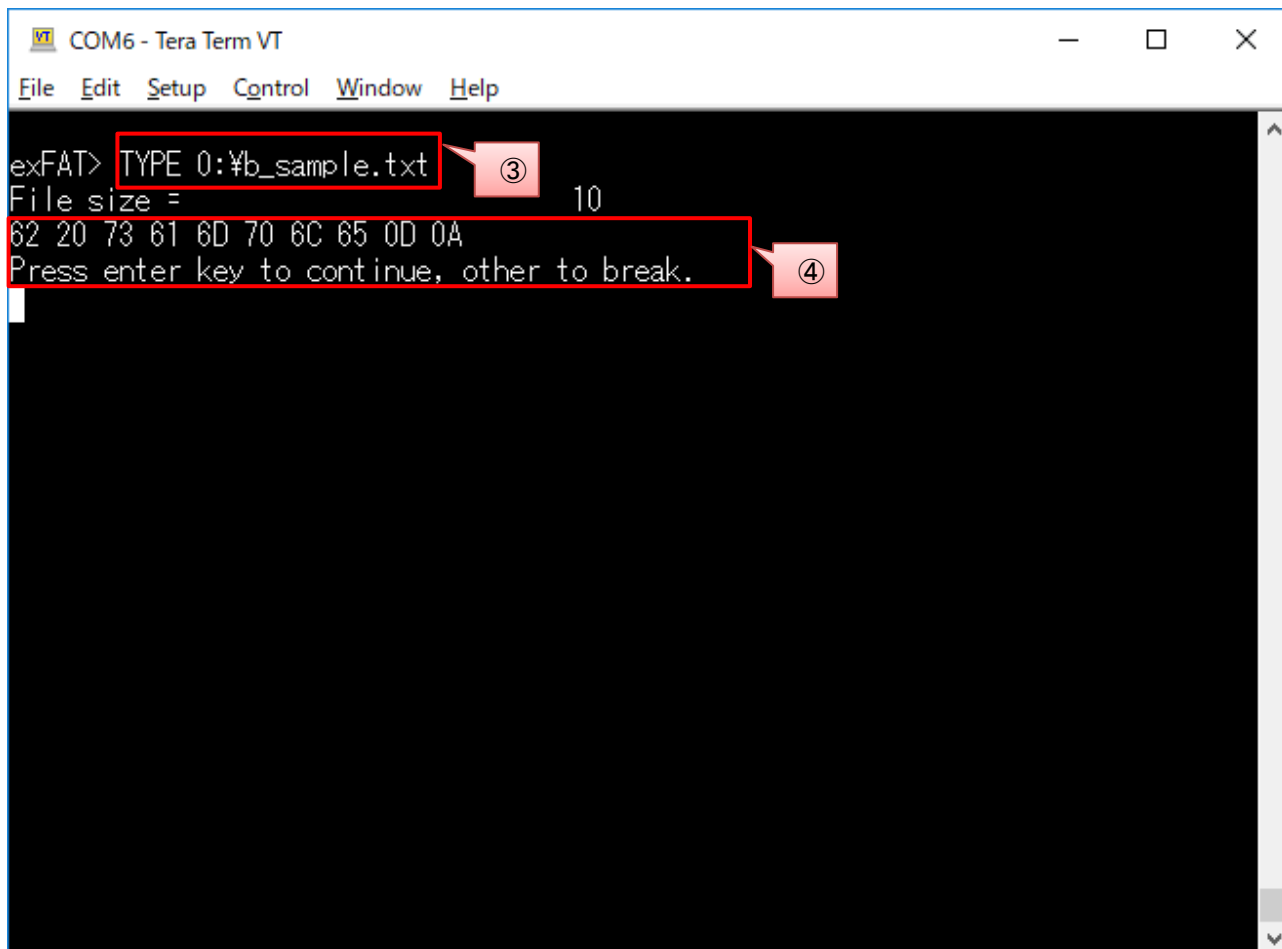
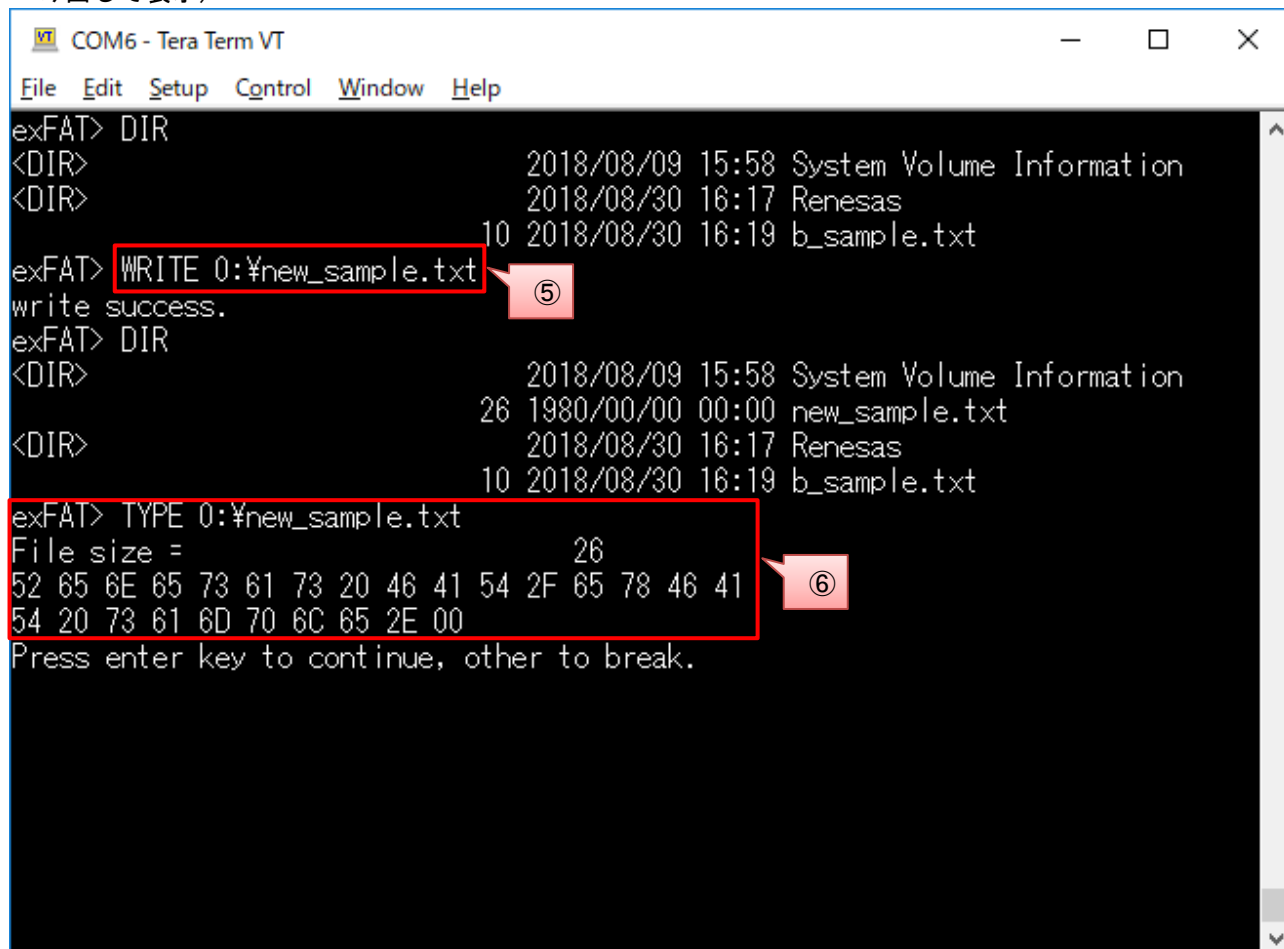


図 7.4 「TYPE」コマンド発行例

11. ファイル書き込み

ターミナルより、コマンド「WRITE」を発行（図中⑤）し、接続している SD カードの指定されたファイルに文字列「Renesas FAT/exFAT sample.」を書き込みます。（図中⑥ 書き込んだ文字列を読み出して表示）



The screenshot shows a Tera Term VT window titled "COM6 - Tera Term VT". The terminal displays the following sequence of commands and outputs:

```
exFAT> DIR
<DIR>                2018/08/09 15:58 System Volume Information
<DIR>                2018/08/30 16:17 Renesas
10 2018/08/30 16:19 b_sample.txt

exFAT> WRITE 0:¥new_sample.txt
write success.
exFAT> DIR
<DIR>                2018/08/09 15:58 System Volume Information
26 1980/00/00 00:00 new_sample.txt
<DIR>                2018/08/30 16:17 Renesas
10 2018/08/30 16:19 b_sample.txt

exFAT> TYPE 0:¥new_sample.txt
File size =          26
52 65 6E 65 73 61 73 20 46 41 54 2F 65 78 46 41
54 20 73 61 6D 70 6C 65 2E 00
Press enter key to continue, other to break.
```

Two red boxes with callout numbers are present: Box ⑤ highlights the command `WRITE 0:¥new_sample.txt`, and Box ⑥ highlights the hexadecimal data output of the `TYPE` command.

図 7.5 「WRITE」コマンド発行例

7.7 注意事項

お客様の製品へ FatFs を組み込む場合には、FatFs のライセンスをご確認いただき、お客様の責任にて実施して下さい。

8. スマートコンフィグレータによるコンポーネント追加手順

本章では、スマートコンフィグレータを使用して、SD ドライバコンポーネントを追加するための情報を提供します。

8.1 コンポーネント追加

コンポーネント追加の手順は以下のとおりです。

1. 「Components」タブを選択し、「Add component」鈕を押下します。

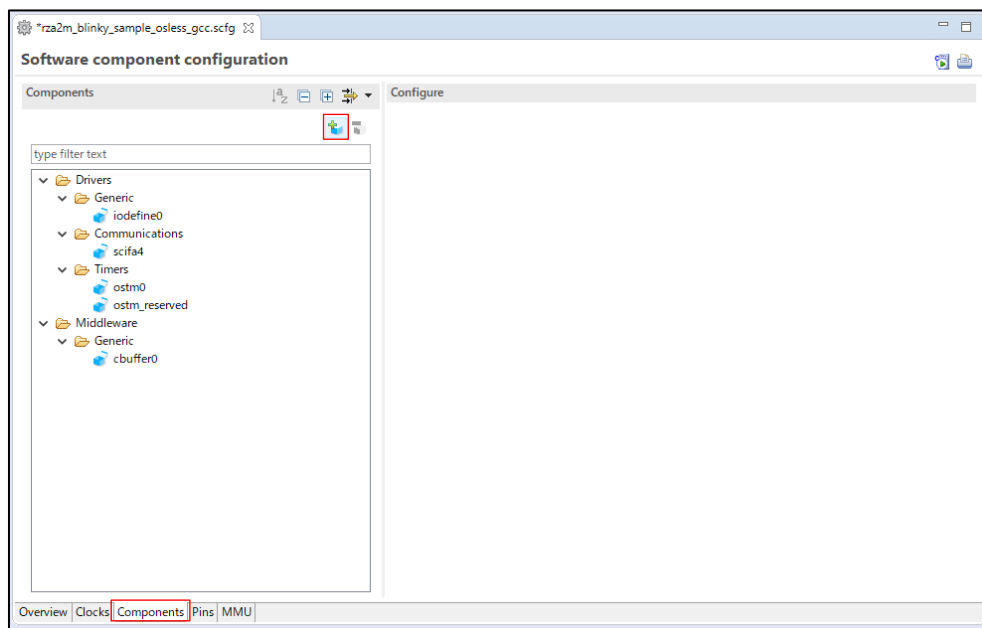


図 8.1 コンポーネント追加

2. SD ドライバコンポーネント「r_sdhi_simplified」を選択し、「Next >」鈕を押下します。

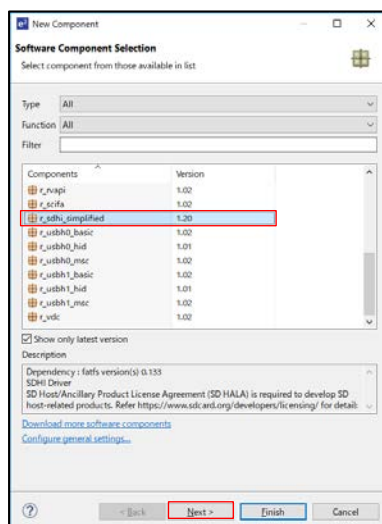


図 8.2 コンポーネント選択(1/2)

3. 「Finish」 鈕を押下します。（図はチャンネル 0 選択時です。）

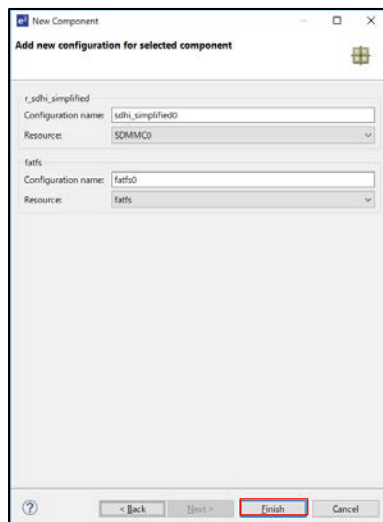


図 8.3 コンポーネント選択(2/2)

4. SD ドライバコンポーネント「sdhi_simplified0」、fatfs コンポーネント「fatfs0」が追加されます。

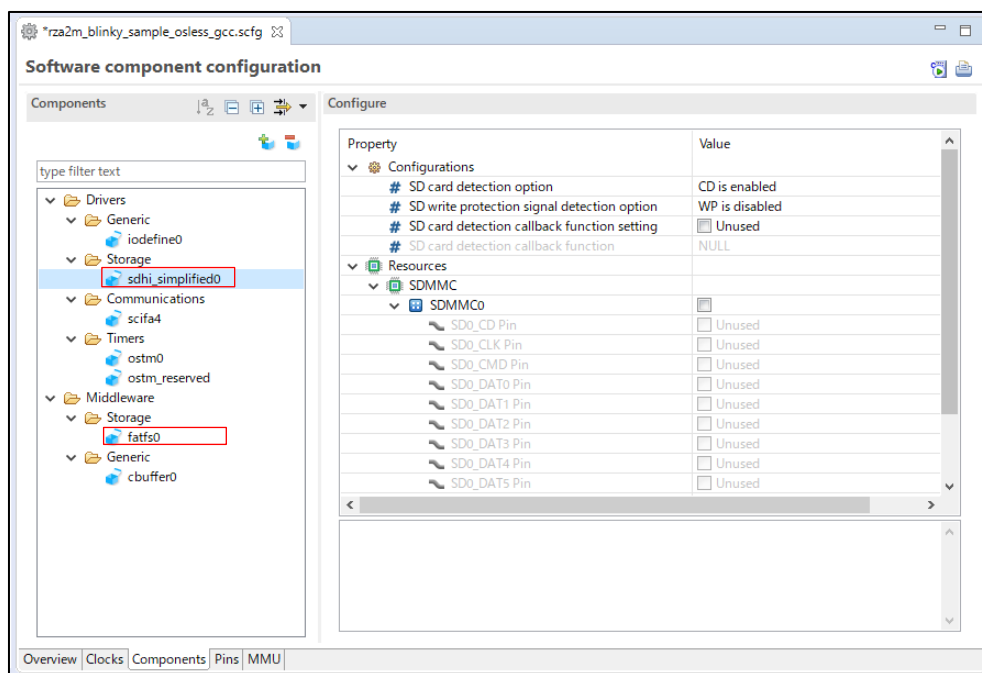


図 8.4 コンポーネント追加完了後の Components 画面

8.2 コンフィグ設定

コンフィグ設定の手順は以下のとおりです。「5 コンフィグオプション」を参考にしてください。
以降は、チャンネル0 の設定の例です。

8.2.1 SD カード検出オプション設定

1. 「Components」から「sdhi_simplified0」を選択、「Configure」-「Property」-「Configurations」-「SD card detection option」の「Value」を選択します。

デフォルト設定（SD カード検出有効）を使用します。

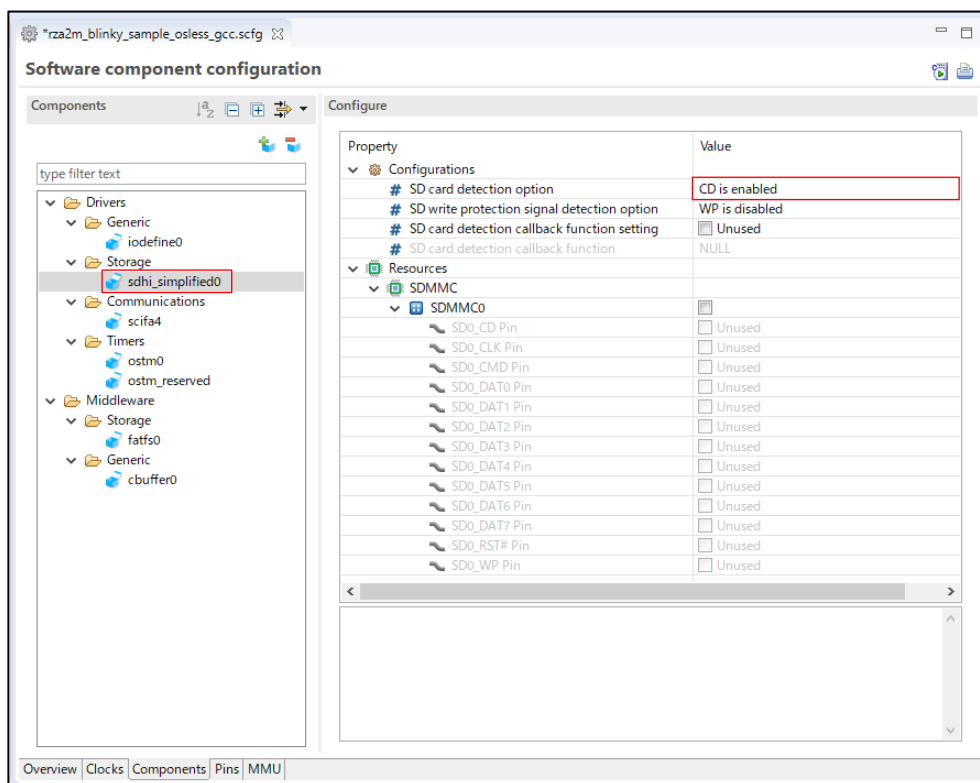


図 8.5 SD カード検出オプション設定

8.2.2 ライトプロテクト信号検出オプション設定

1. 「Components」から「sdhi_simplified0」を選択、「Configure」-「Property」-「Configurations」-「SD write protection signal detection option」の「Value」を選択します。

デフォルト設定（ライトプロテクト信号検出無効）を使用します。

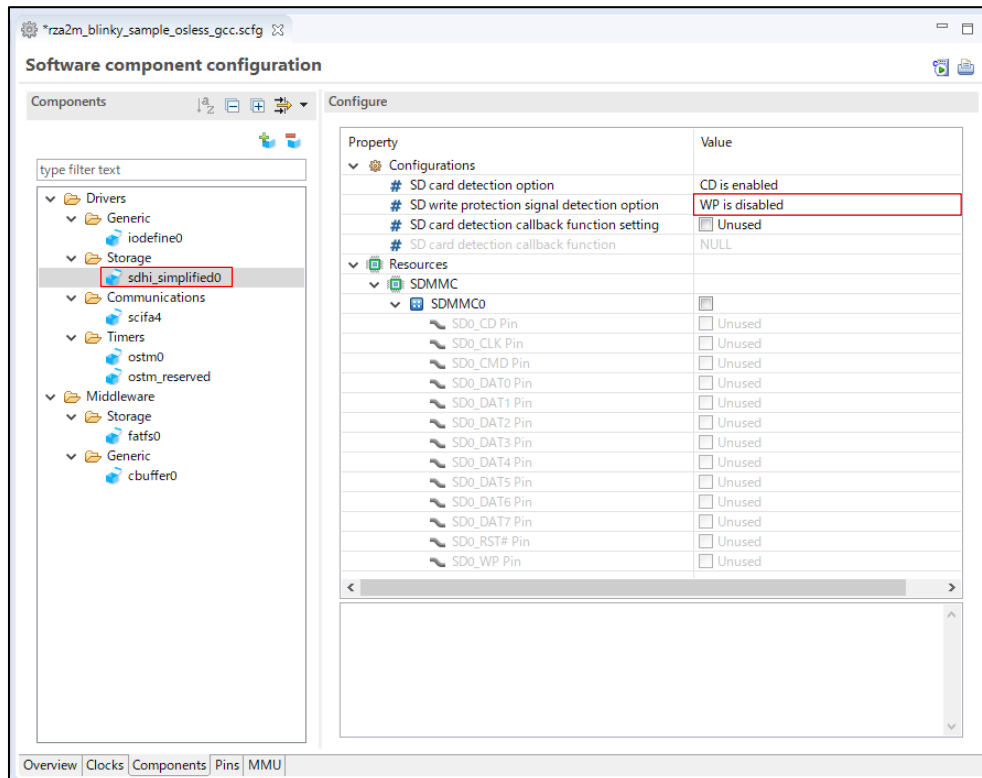


図 8.6 ライトプロテクト信号検出オプション設定

8.2.3 SD カード検出コールバック関数設定

1. 「Components」から「sdhi_simplified0」を選択、「Configure」-「Property」-「Configurations」-「SD card detection callback function setting」のチェックボックスを有効にします。

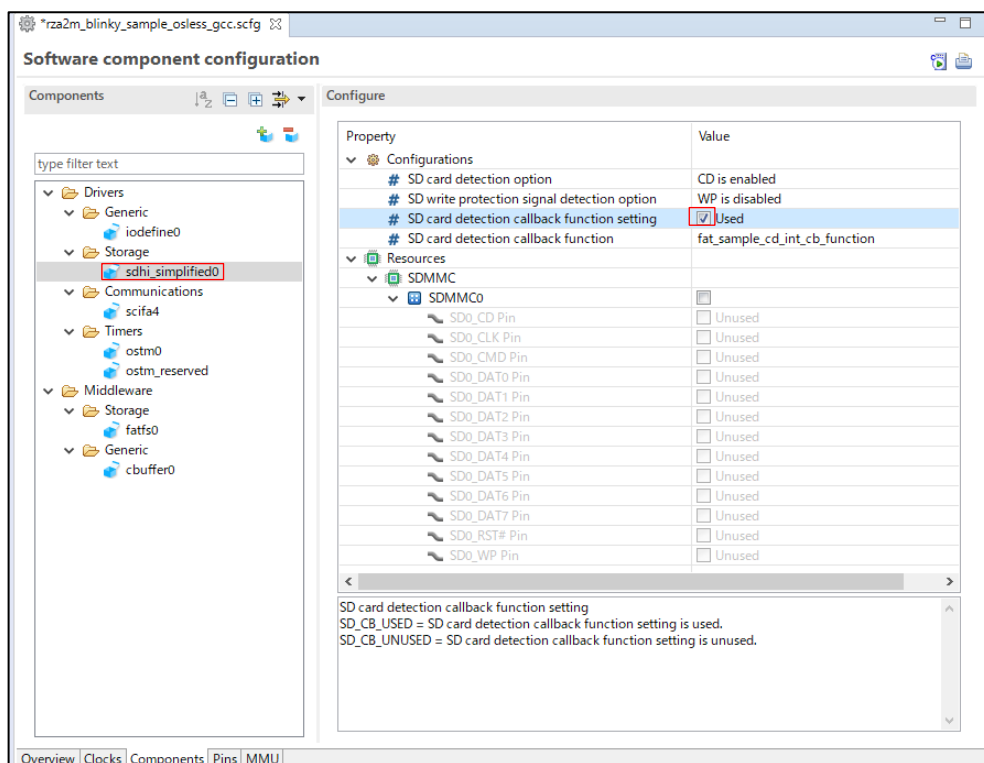


図 8.7 SD カード検出コールバック関数設定(1/2)

2. 「SD card detection callback function」の「Value」にコールバック関数名を入力します。

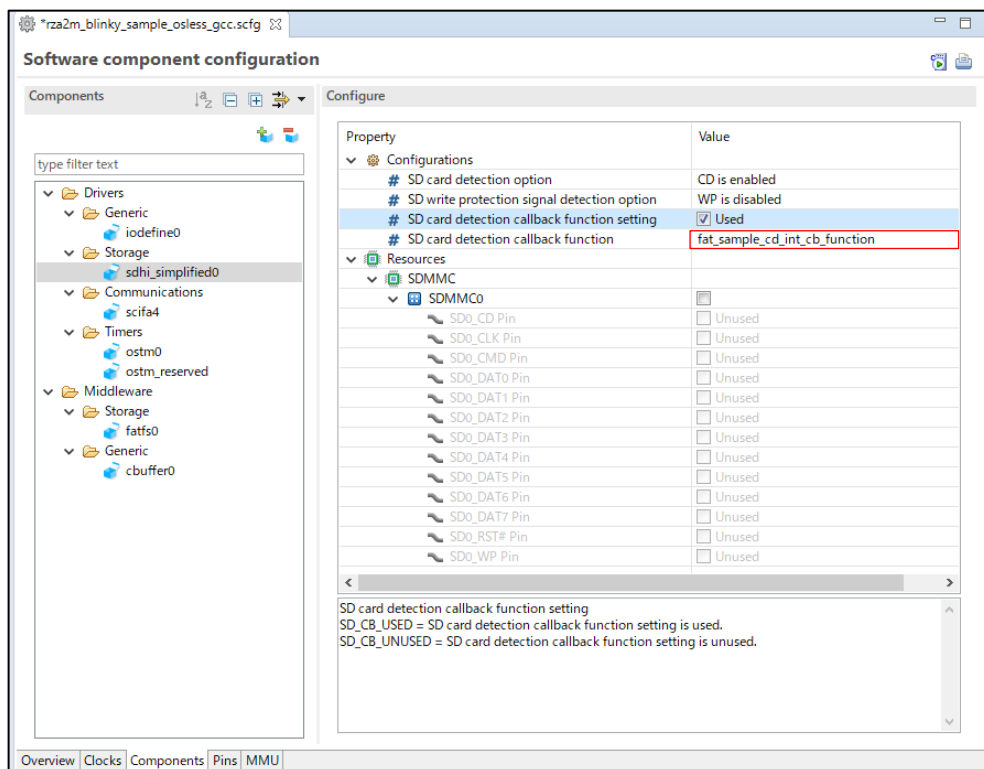


図 8.8 SD カード検出コールバック関数設定(2/2)

8.3 端子設定

端子設定の手順は以下のとおりです。使用する端子は「7.4 使用端子と機能」を参考にしてください。

以降は、チャンネル0 の設定の例です。

8.3.1 CD 端子、WP 端子の設定

1. 「Components」から「sdhi_simplified0」を選択、「Configure」-「Property」-「Resources」-「SDMMC」-「SDMMC0」のチェックボックスを全て有効にします。

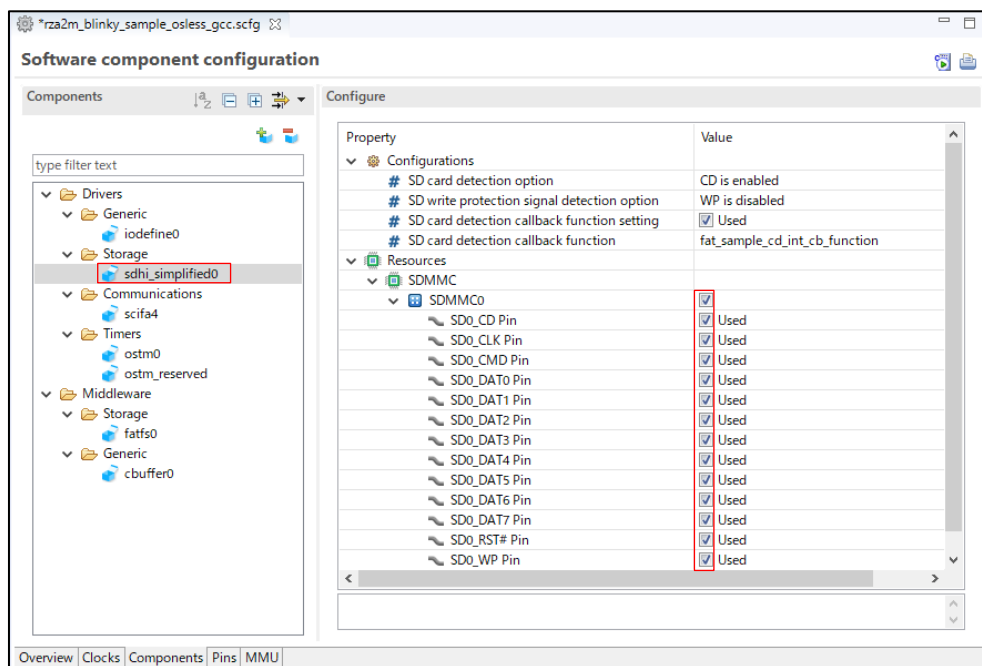


図 8.9 SDMMC0 端子の選択

2. 「Pins」-「Pin Function」タブを選択、「Hardware Resource」から「SD/MMC host interface」-「SDMMC0」を選択、「Pin Function」から SD0_CD と SD0_WP の割り当てる端子を選択します。

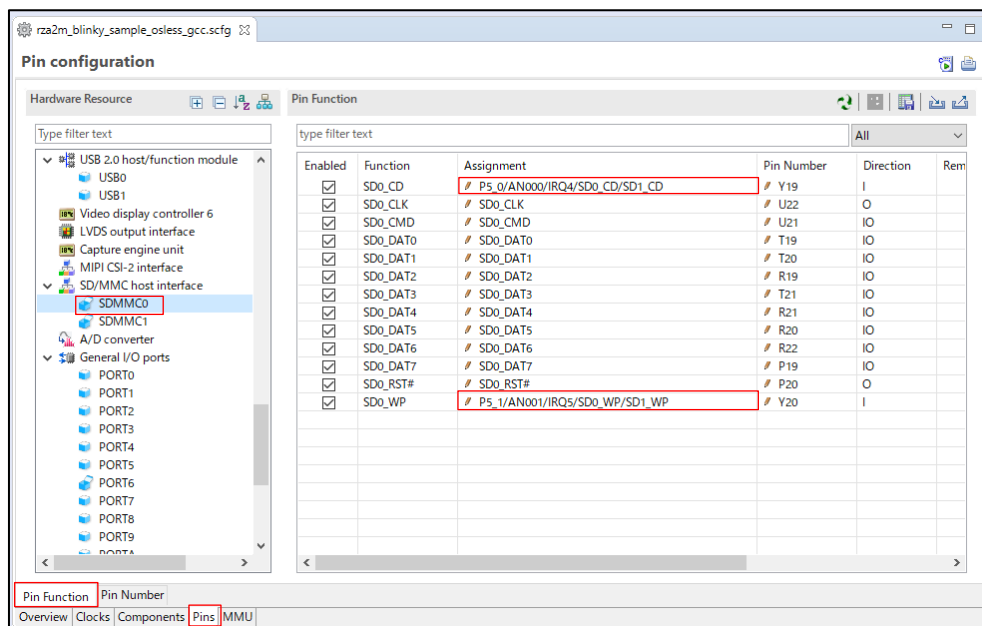


図 8.10 SD0_CD、SD0_WP 端子の割り当て

8.3.2 PD_1 端子（SDVcc_SEL）の設定

1. 「Hardware Resource」から「General I/O ports」-「PORTD」を選択し、PD_1 端子のチェックボックスを有効にします。

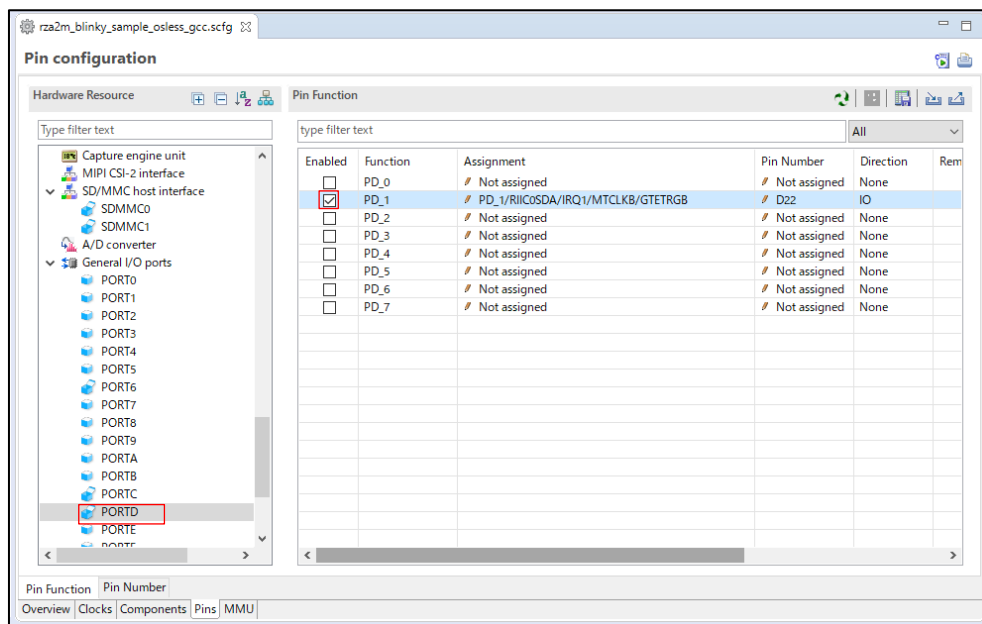


図 8.11 PD_1 端子の選択

2. 「Pin Number」タブを選択、「Pin configuration」から PD_1 端子の設定をおこないます。

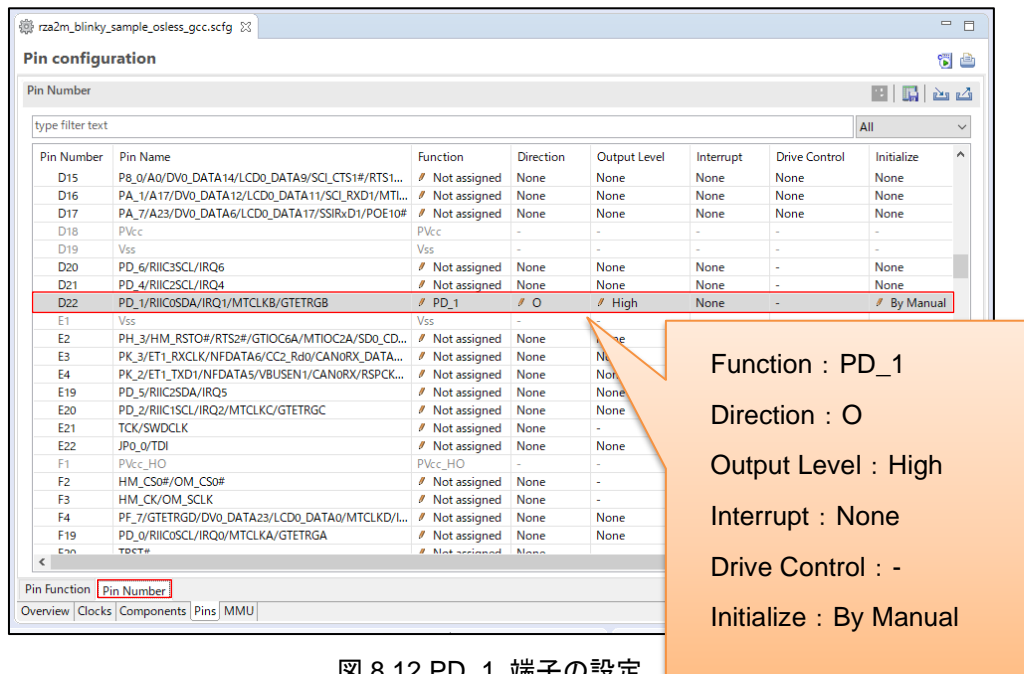


図 8.12 PD_1 端子の設定

8.3.3 PJ_1 端子（SW3 キー入力）の設定

1. 「Pin Function」タブを選択、PJ_1 端子のチェックボックスを有効にします。

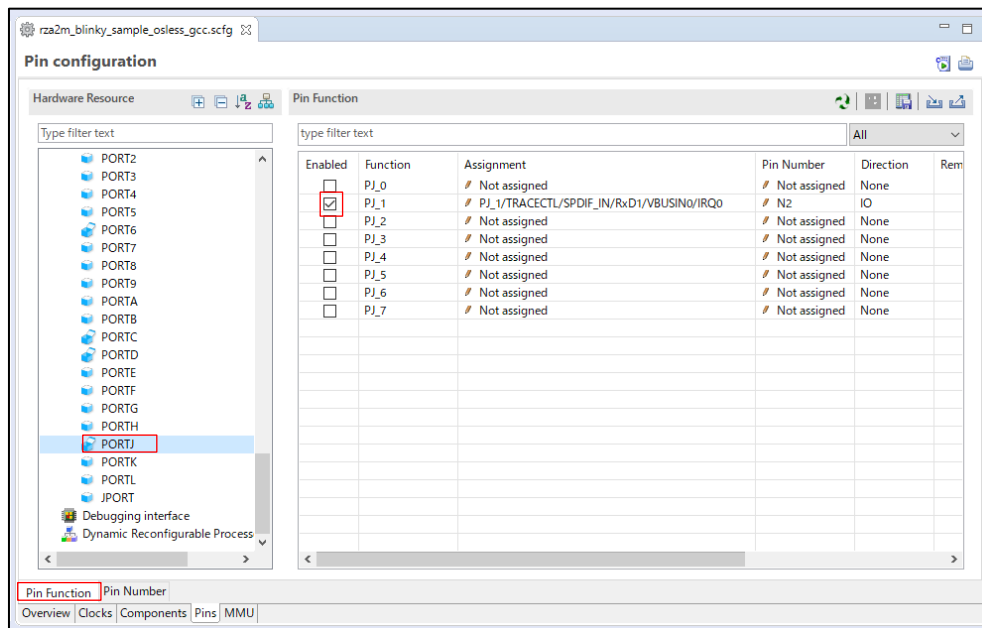


図 8.13 PJ_1 端子の選択

2. 「pin Number」タブを選択、PJ_1 端子の設定を行います。

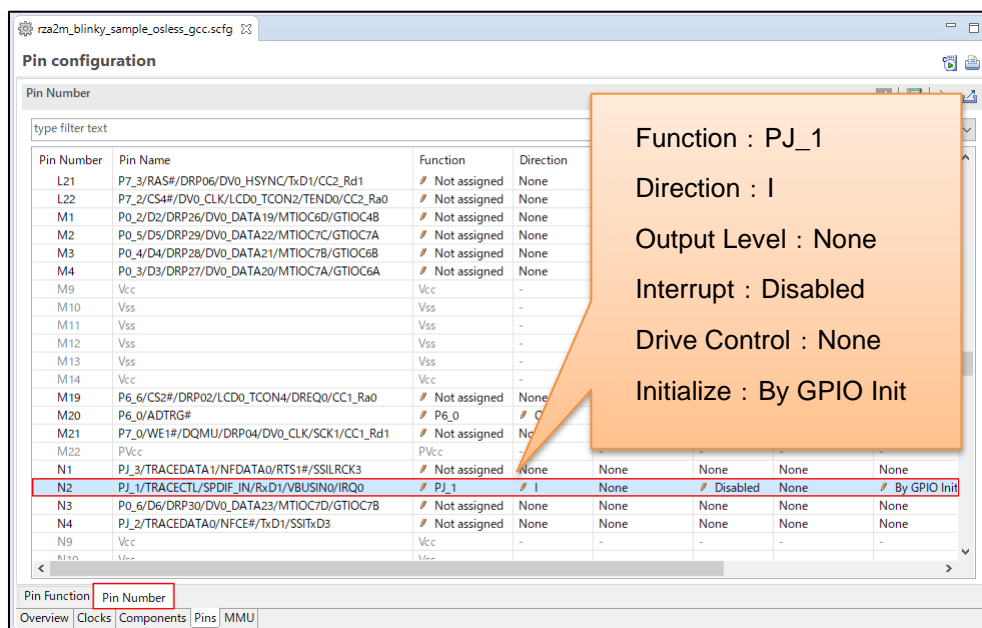


図 8.14 PJ_1 端子の設定

8.3.4 PC_1 端子（LED1（Yellowish-green））の設定

1. 「Pin Function」タブを選択、PC_1 端子のチェックボックスを有効にします。

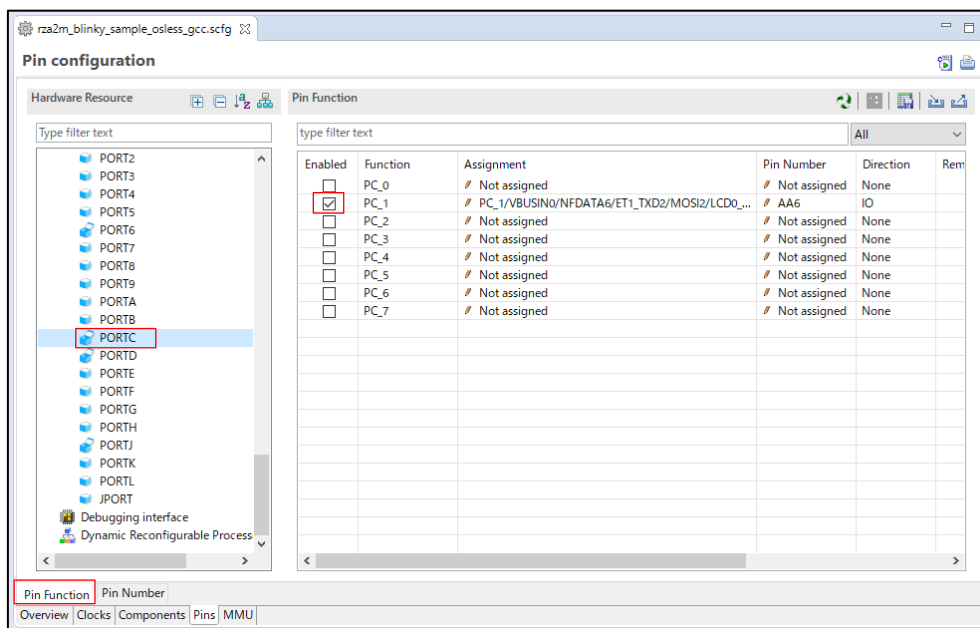


図 8.15 PC_1 端子の選択

2. 「Pin Number」タブを選択、PC_1 端子の設定を行います。

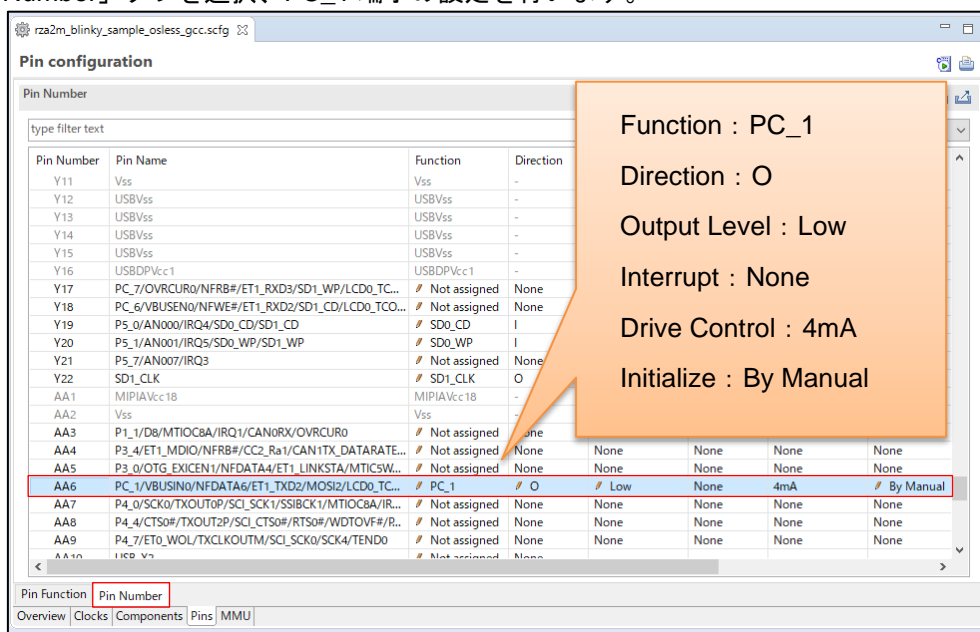


図 8.16 PC_1 端子の設定

8.4 コード生成

コード生成の手順は以下のとおりです。

1. 「Generate Code」 釦を押下します。

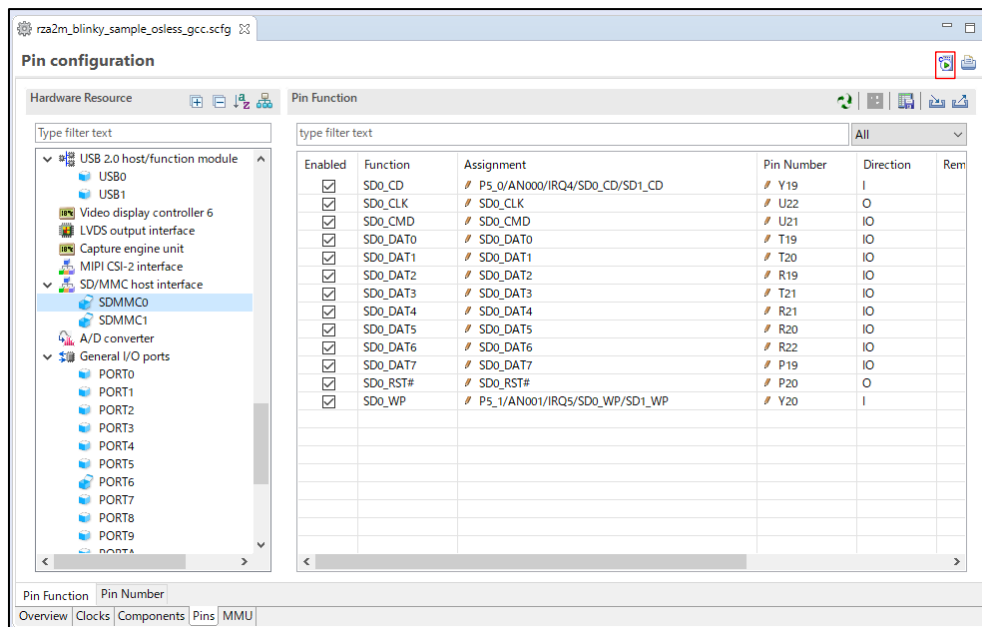


図 8.17 「Generate Code」 の選択

2. コードが生成されます。

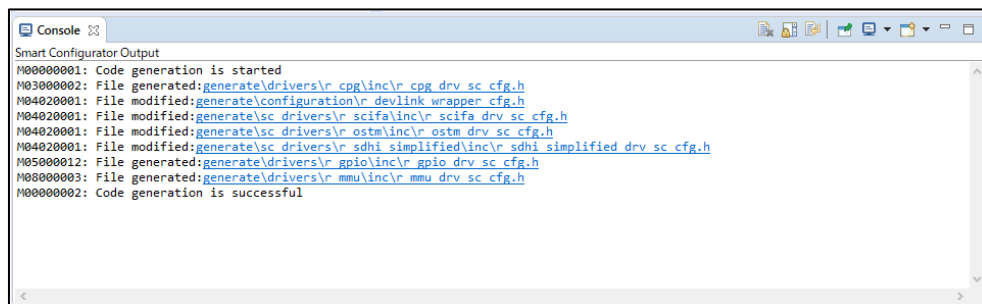


図 8.18 コード生成時のコンソール画面

9. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

RZ/A2M グループ ユーザーズマニュアル ハードウェア編

（最新版をルネサス エレクトロニクスホームページから入手してください。）

RTK7921053C00000BE（RZ/A2M CPU ボード）ユーザーズマニュアル

（最新版をルネサス エレクトロニクスホームページから入手してください。）

RTK79210XXB00000BE（RZ/A2M SUB ボード）ユーザーズマニュアル

（最新版をルネサス エレクトロニクスホームページから入手してください。）

Arm Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C

（最新版を Arm ホームページから入手してください。）

Arm Cortex™-A9 Technical Reference Manual Revision: r4p1

（最新版を Arm ホームページから入手してください。）

Arm Generic Interrupt Controller Architecture Specification - Architecture version2.0

（最新版を Arm ホームページから入手してください。）

Arm CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual Revision: r3p3

（最新版を Arm ホームページから入手してください。）

テクニカルアップデート／テクニカルニュース

（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：統合開発

統合開発環境 e2 studio のユーザーズマニュアルは、ルネサス エレクトロニクスホームページから入手してください。

（最新版をルネサス エレクトロニクスホームページから入手してください。）

規格書

SD Memory Card Specifications Part1 PHYSICAL LAYER Simplified SPECIFICATION, Ver6.00, August 29, 2018

Multi Media Card System Specifications, Ver4.1, Jan 2005

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.20	Jun.26.19	91	表 5.1 コンフィグオプション
		105-114	SD カード検出コールバック関数設定を追加 8 章「スマートコンフィグレータによるコンポーネント追加手順」を追加
		95	表 7.2 動作確認条件 項目"ターゲット"を削除
		99	表 7.5 SD ドライバのメモリ使用量 ROM サイズ更新
		—	誤記修正
1.10	May.17.19	95	表 7.2 動作確認条件 コンパイラオプション"-mthumb-interwork"を削除
1.00	Jan.01.19	—	初版発行

製品ご使用上の注意事項

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

1. 静電気対策

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

2. 電源投入時の処置

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

3. 電源オフ時における入力信号

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

4. 未使用端子の処理

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

5. クロックについて

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

6. 入力端子の印加波形

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

7. リザーブアドレス（予約領域）のアクセス禁止

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

8. 製品間の相違について

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

ご注意書き

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

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

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

当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。

6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関し、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものいたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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

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

(Rev.4.0-1 2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

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

www.renesas.com/contact/

商標について

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