



第1部

Shinobi2 ライブラリ編

この章では、セガライブラリの中心的なシステムの全体に関することと、他のライブラリとの関係を含んでいる機能などを、全般に解説します。

目次

1	システムライブラリについて	7
1.1	セガライブラリの特徴	7
1.1.1	静的ライブラリとしての提供	7
1.1.2	ライブラリの実行タイミング	7
1.1.3	コールバック関数	7
1.1.4	再入不可	8
1.2	メモリマップのマッピング	8
1.2.1	メモリマップ	9
1.3	CPU モード	11
1.4	アプリケーションが実行されるまで	13
1.4.1	概要	13
1.5	セクション及びセクション名について	15
1.5.1	セクションとは	15
1.5.2	セクション名について	15
1.6	ヒープ領域の変更について	18
1.7	スタックの変更について	20
1.8	ソフトウェアリセットの注意事項	21
2	システムライブラリ (shinobi2.lib) の構成	22
2.1	システムライブラリについて	22
2.2	各ライブラリの概要	23
3	CPU (SH4) 基本サービス関連のライブラリ	26
3.1	割り込みライブラリ (syInt、syChain) について	26
3.1.1	SH4 の割り込み処理	26
3.1.2	モジュール構成	26
3.1.3	CodeScape デバッガとの連携	27
3.2	キャッシュライブラリ (syCache) について	28
3.2.1	キャッシュとは	28
3.2.2	キャッシュの仕組み	28
3.2.3	ライトバック	29
3.2.4	キャッシュのコヒーレンシ	29
3.2.5	キャッシュのインバリデート	29
3.2.6	キャッシュのバージ	30
3.2.7	キャッシュのプリフェッチ	30
3.2.8	キャッシュエントリの確保	30
3.2.9	キャッシュの RAM モード	31
3.2.10	キャッシュ操作が必要な場面	31
3.2.11	キャッシュの操作で注意すべき点	31
3.2.12	関数一覧	32
3.3	ストアキューライブラリ (sySq) について	33
3.3.1	ストアキューとは	33
3.3.2	Shinobi2 への変更点	33
3.3.3	関数一覧	33
3.4	メモリマネージメントライブラリ (syMalloc) について	34
3.4.1	syMalloc のアルゴリズム	34
3.4.2	syMallocChangeIF 関数による malloc 関数の入れ替え	35
3.4.3	関数一覧	36
3.5	コールバックライブラリ (syCallback) について	37
3.5.1	コールバック関数の初期化	37
3.5.2	コールバック関数の登録	37
3.5.3	コールバック関数の終了	37
3.5.4	登録したコールバック関数の削除	37
3.5.5	関数一覧	37
3.6	タイマライブラリ (syTmr) について	38

3.6.1 SH4 のタイマユニットとの関連	38
3.6.2 タイマライブラリの関数の分類	38
3.6.3 タイマライブラリの初期化	40
3.6.4 関数一覧	40
3.7 システムマネージャライブラリ (syMng)	41
3.7.1 初期化	41
3.7.2 関数一覧	42
3.8 ファイバライブラリ (syFbr) について	43
3.8.1 機能	43
3.8.2 関数一覧	45
4 BootROM 関連のライブラリ	46
4.1 BootROM サービスライブラリ (syBt) について	46
4.1.1 関数一覧	46
4.2 BootROM フォントライブラリ (syBtfnt) について	46
4.2.1 アクセスチェック	46
4.2.2 BootROM に含まれる文字データ	46
4.2.3 BootROM のフォントデータの格納例	47
4.2.4 関数一覧	48
5 GD-ROM 関連のライブラリ	49
5.1 GD ファイルシステムライブラリ (gdFs) について	49
5.1.1 GD-ROM の基本事項	49
5.1.2 ファイル構造について	49
5.1.3 ファイルの配置	50
5.1.4 ディスクドアオープンの検出	51
5.1.5 ディスクの交換の検出時について	53
5.1.6 エラー発生時のリトライについて	54
5.1.7 GDFS の内部動作について	57
5.1.8 その他 & 注意点	58
5.1.9 関数一覧	59
6 ペリフェラル関連のライブラリ	61
6.1 ペリフェラルの概要	61
6.1.1 ペリフェラルの分類について	61
6.1.2 ドリームキャスト・コントロールポートの特徴	64
6.2 ペリフェラルデータライブラリ (pd) について	65
6.2.1 ペリフェラルポートについて	65
6.2.2 ペリフェラルの判定方法の関連事項	65
6.2.3 コントロールポートの初期化	66
6.2.4 ポートに接続されているペリフェラルの種別を取得	69
6.2.5 関数一覧	71
6.3 コントロールデバイスライブラリについて	72
6.3.1 コントローラのボタンの状態を取得	72
6.3.2 コントローラのサポートしているボタンを取得	73
6.3.3 デジタルボタンの状態	74
6.3.4 アナログボタンの状態	75
6.3.5 データ取得時のエラー	75
6.3.6 ペリフェラルデータの低レイテンシ取得	76
6.3.7 関数一覧	77
6.4 振動デバイスライブラリ (pdVib) について	78
6.4.1 振動の構造について	78
6.4.2 振動の処理を実行するタイミング	78
6.4.3 振動デバイスの接続方向	79
6.4.4 振動ユニットのパラメータ	79
6.4.5 振動ユニット数と同時設定可能なユニット数	80
6.4.6 振動ユニット情報の取得	80
6.4.7 振動させる	82
6.4.8 振動に必要なパラメータと設定例	83
6.4.9 振動を止める	84

6.4.10 自動停止時間の設定.....	84
6.4.11 「ぶるぶるばっく」専用関数.....	85
6.4.12 関数一覧.....	86
6.5 LCD デバイスライブラリ (pdLcd) について.....	87
6.5.1 ビジュアルメモリの LCD デバイスのスペック.....	87
6.5.2 ピクセルデータ形式について.....	87
6.5.3 ピクセルデータフォーマット.....	87
6.5.4 液晶の向き.....	88
6.5.5 液晶使用時の注意事項.....	88
6.5.6 関数一覧.....	88
6.6 タイマーデバイス (pdTmr) ライブラリについて.....	89
6.6.1 定義.....	89
6.6.2 関数一覧.....	89
6.7 ガンデバイスライブラリ (pdGun) について.....	90
6.7.1 ガンモードの設定.....	90
6.7.2 拡張ペリフェラルに関する制限事項.....	90
6.7.3 画面フラッシュ.....	91
6.7.4 ガン座標取得の位置の制限.....	91
6.7.5 ガンデバイスの優先順位.....	91
6.7.6 関数一覧.....	92
6.8 キーボードデバイスライブラリ (pdKbd) について.....	93
6.8.1 ハードウェアの取得方式について.....	93
6.8.2 構造体仕様.....	93
6.8.3 コントロールキー.....	93
6.8.4 LED 点灯状態.....	94
6.8.5 キーコード.....	94
6.8.6 キーボードのハードウェア情報.....	103
6.8.7 サンプルプログラム.....	104
6.8.8 関数一覧.....	104
6.9 マウスライブラリ (pdMouse) について.....	105
6.9.1 定数.....	105
6.9.2 構造体仕様.....	106
6.9.3 サンプルプログラム.....	107
6.9.4 関数一覧.....	108
6.10 記録デバイスライブラリ (bu) について.....	109
6.10.1 セーブ中のコントローラ及びメモリーカードの抜き差し処理.....	109
6.10.2 セーブ中のディスクドアのオープン、及びソフトリセット時の処理.....	109
6.10.3 ペリフェラルの同時使用制限.....	110
6.10.4 ペリフェラルデータの取得タイミング.....	112
6.10.5 コントロールポートについて.....	113
6.10.6 メモリーカードの容量.....	113
6.10.7 関数種別.....	114
6.10.8 一般アプリケーションへの組み込み.....	115
6.10.9 一般アプリケーションでのファイルアクセス.....	117
6.10.10 特殊アプリケーションでのファイルアクセス.....	124
6.10.11 実行ファイル.....	127
6.10.12 バックアップファイルフォーマット.....	131
6.10.13 動的ワーク確保.....	141
6.10.14 メモリーカードの切断時.....	142
6.10.15 動的ワーク確保に関する関数.....	143
6.10.16 関数一覧.....	143
6.11 音声入力デバイスライブラリ (ws) について.....	144
6.11.1 音声入力について.....	144
6.11.2 モジュール構成.....	144
6.11.3 音声入力ライブラリの特徴.....	145
6.11.4 音声データの取得方法.....	145
6.11.5 関数一覧.....	147

6.12	A/Vケーブルペリフェラルライブラリ (syCbl) について	148
6.12.1	Dev.Box の DIP SW. とケーブル種別	148
6.12.2	Dev.Box の ROTARY SW. と放送形式の種別	148
6.12.3	ケーブルの種類を判別	149
6.12.4	関数一覧	149
7	フラッシュメモリ関連のライブラリ	150
7.1	コンフィグレーションライブラリ (syCfg) について	150
7.1.1	初期化・終了	150
7.1.2	フラッシュメモリへの書き込み	150
7.1.3	関数一覧	150
7.2	プレイ履歴ライブラリ (uf) について	151
7.2.1	プレイ履歴ライブラリとは	151
7.2.2	動作概要	151
7.2.3	ライブラリ部	153
7.2.4	リフレッシュについて	154
7.2.5	関数一覧	156
8	その他のライブラリ	157
8.1	メモリコピーライブラリ (syMemCopy) 及びメモリセットライブラリ (syMemSet) について	157
8.1.1	メモリコピーライブラリ (syMemCopy) について	157
8.1.2	メモリセットライブラリ (SyMemSet) について	158
8.2	リアルタイムクロックライブラリ (syRtc) について	159
8.2.1	リアルタイムライブラリの初期化について	159
8.2.2	リアルタイムライブラリの終了	159
8.2.3	フラッシュメモリへの書き込み	159
8.2.4	関数一覧	159
8.3	ビデオ設定ライブラリ (syVideo) について	160
8.3.1	ビデオ設定ライブラリの概要	160
8.3.2	Dreamcast のケーブル	160
8.3.3	放送形式	161
8.3.4	その他	161
8.3.5	ケーブル判別・画面出力モード一覧表	161
8.3.6	Shinobi2 でのビデオ設定ライブラリへの変更経緯	161
8.3.7	関数一覧	162
8.4	シリアルライブラリ (syCom) について	163
8.4.1	シリアルライブラリの仕様について	163
8.4.2	通信の初期化と終了	163
8.4.3	データの送受信について	164
8.4.4	関数一覧	165
8.5	拡張機器判別ライブラリ (syExtChk) について	166
8.5.1	拡張機器判別ライブラリの概要	166
8.5.2	拡張機器判別ライブラリの初期化	166
8.5.3	拡張機器判別ライブラリの終了	166
8.5.4	情報取得関数使用時の注意事項	166
8.5.5	関数一覧	167
9	言語使用時の注意事項	168
9.1	アセンブラ使用時の注意	168
9.1.1	C 言語とのインターフェイス	168
9.1.2	ライブラリのレジスタ使用	168
9.1.3	その他注意事項	168
9.2	C++ の使用時の注意	169
9.2.1	グローバルオブジェクトのコンストラクタの初期化	169
10	オーバーレイ	172
10.1	オーバーレイの手法	172
10.2	オーバーレイ全般に対する注意	172
10.2.2	メニュー&ローダ	173
10.2.3	FSY ファイルを用いたオーバーレイ	174

10.2.4	関数テーブルを用いたオーバーレイ	175
10.2.5	CodeWarrior によるオーバーレイ	176
11	Shinobi 1 から 2 への主な変更点	177
11.1	システム全体に関する変更点	177
11.2	追加及び新規公開ライブラリ	178
11.3	削除されたライブラリ	181
11.4	既存ライブラリの変更点	182
12	補足資料	184
12.1	各ペリフェラル毎の固有情報	184
12.1.1	ドリームキャスト・コントローラ (HKT-7700)	184
12.1.2	レーシングコントローラ (HKT-7400)	185
12.1.3	アーケードスティック (HKT-7300)	186
12.1.4	ドリームキャスト・キーボード (HKT-7600)	187
12.1.5	キティバージョンのドリームキャスト付属キーボード (HKT-7601)	188
12.1.6	ドリームキャスト・ガン (HKT-7801)	189
12.1.7	つりコントローラ (HKT-8700)	190
12.1.8	ビジュアルメモリ (HKT-7000)	192
12.1.9	ぷるぷるぱっく (HKT-8600)	193
12.1.10	マイクデバイス ~音声認識・入力用ユニット~ (HKT-7200)	194
12.1.11	ツインスティック (HKT-7500)	195
12.1.12	マウス (HKT-9900)	196
12.1.13	マラカスコントローラ (HKT-9700)	197
12.1.14	ドリームアイ (HKT-9400)	199
12.1.15	ドリームアイ用マイクデバイス (HKT-9800)	200
12.2	ビデオケーブル判別/画面出力モード一覧表	201
	用語集	203

1 システムライブラリについて

ここでは、セガライブラリの中心的なシステムの全体に関連する部分や、他のライブラリとの関係を含んでいる機能などを全般に説明します。

1.1 セガライブラリの特徴

ここではセガライブラリの特徴について説明します。

ちなみに以下の特徴については、基本的に Shinobi2.lib 以外の全てのセガライブラリについても同様なものです。

1.1.1 静的ライブラリとしての提供

ライブラリについては、すべて静的なライブラリとして提供されます。アプリケーションプログラムについては、リンク時にアドレスが解決されます。

注 意 実行時に動的にアドレスの解決を行う機構は、サポートされていません。

1.1.2 ライブラリの実行タイミング

ライブラリを実行するタイミングは、以下の2種類に分類されます。

(1) 完了復帰型

ライブラリのコードが実行されるタイミングは、アプリケーションから呼び出されてから、その処理が終了するまでの間を言います。

(2) 即時復帰型

ファイルの読み込みなどの時間がかかる処理に関しては、処理要求の受付のみを行い、直ちに呼び出して元に戻るタイプの関数があります。この場合、要求に対する実際の処理は、各ライブラリのサーバ関数で行います。

サーバ関数は、一般に VBlank 割り込みに登録されており、毎 V に実行されます。

ノート 即時復帰型の関数には、完了を通知するためのコールバック関数もしくは進行状況を取得するためのステータス取得関数が準備されています。これにより処理の完了を知ることができます。

VBlank 割り込みのほかにも、デバイス毎の割り込みに対する処理については各ライブラリ（または、ドライバ）が行っています。一部の割り込みについては、ユーザーが割り込み発生時に処理を行えるようにコールバック関数の登録がサポートされています。

1.1.3 コールバック関数

コールバック関数は、ライブラリがユーザーの登録した関数を呼び出すための機構です。

コールバック関数によって、エラーや処理の完了などの特定のイベントが発生した際に、ユーザーのプログラムを実行することができます。

ただし、コールバック関数は、割り込みコンテキスト（割り込みが発生し、割り込み処理が終了するまでの処理の間）から呼ばれたりするため、ライブラリ関数の呼び出しなどに制限が発生します。コールバック関数では、フラグの設定や、変数の値の更新等に使用すべきでしょう。

1.1.4 再入不可

ライブラリは、リエントラント（再入可能）に実装されていません。従って、（アプリケーション上でマルチスレッドを実現したとしても）複数のスレッドからライブラリを呼び出すことについての動作保証はされません。

1.2 メモリマップのマッピング

Dreamcast のメインメモリは、標準で 16M バイト（64M ビット SD-RAM × 2）を装備し、0x0C000000 番地から 0x0CFFFFFF 番地の間にマッピングされています。

Dreamcast の CPU である SH4 のメモリアドレスは、上位 3 ビットに特別な意味をもたせており、残りの 29 ビットで外部メモリ空間にマッピングされています。（例えば、0x0C000000 番地・0x8C000000 番地と 0xAC000000 番地の外部メモリは、同一の物理メモリを指しています）

セガライブラリのアプリケーションは、0x8C000000 ~ 0x8CFFFFFF の領域内（上位 3 ビットが 100 の領域で、この領域を P1 領域と呼びます）で動作します。

ノート P1 領域は、キャッシュを用いたアクセスが可能です。MMU が有効になった場合でも、MMU によるアドレスの変換が行われない空間です。

1.2.1 メモリマップ

- メインメモリ (0x8C000000 ~ 0x8CFFFFFFF 番地) のメモリマップ

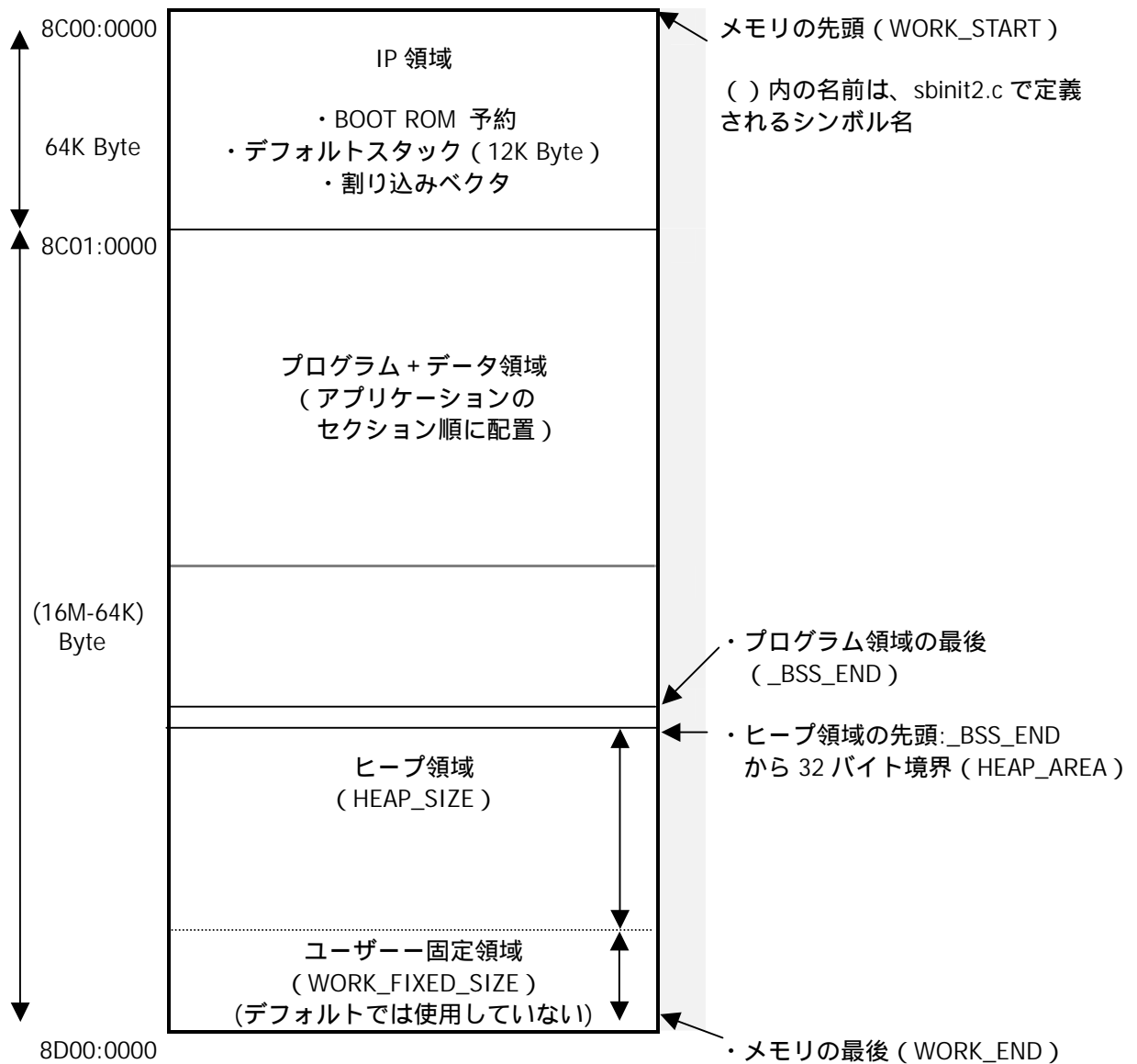


図 1 - 1 メモリマップ

通常はプログラム + データ領域と示した部分に、アプリケーションが配置されます。

注意 0x8C000000 ~ 0x8C00FFFFF までの 64K バイトの領域を IP 領域といい、アプリケーションはデフォルトスタック以外では、使用してはいけません。

● IP 領域 (0x8C00000 ~ 0x8C00FFFF 番地) のメモリマップ

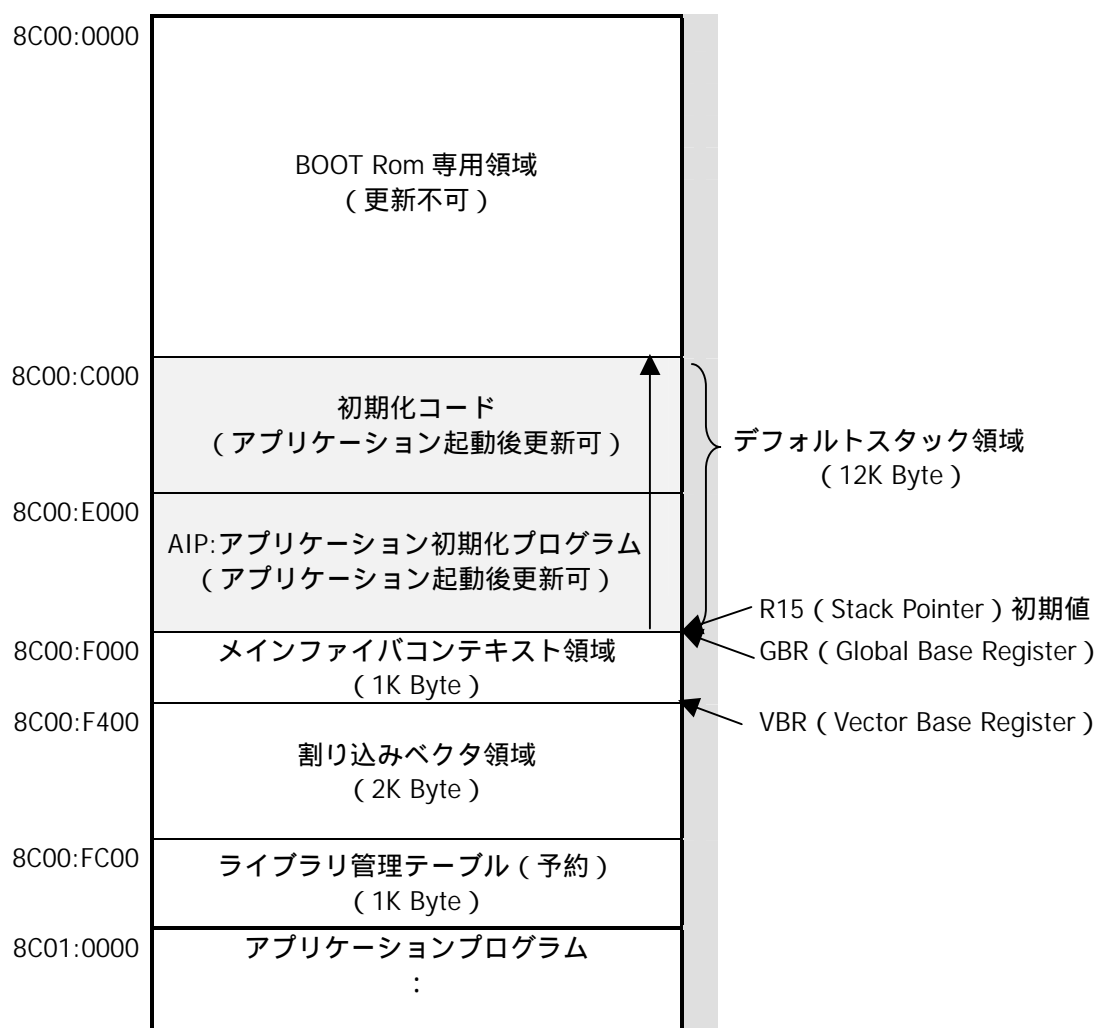


図 1 - 2 IP 領域

IP 領域には起動時に IP.BIN が置かれますが、このうちの一部は初期化後に不要になるため、アプリケーションのデフォルトのスタックとして使用しています。

デフォルトのスタックは、0x8C00C000 ~ 0x8C00EFFF の 12K バイトです。

これ以上のスタックが必要なアプリケーションの場合には、「1.7 スタックの変更について」で示す方法でスタックを変更する必要があります。

1.3 CPU モード

セガライブラリを使用する、アプリケーションの CPU モードについて説明します。

ノート ここで示すモードは、アプリケーションが実行される前に設定されます。また、基本的にアプリケーションからこのモードの設定を、変更してはいけません。

(1) CPU の特権モード (特権モード : SH4 レジスタ設定 : SR.MD = 1)

セガライブラリを使用するアプリケーションは、CPU の特権モードで動作します。これは特権モードがハードウェアのリソースを最大限に利用することが可能なためです。

逆に、このモードでは、不正アドレスのアクセスなどを検出しにくくなるというデメリットもあります。

ノート Shinobi2 では MMU を利用することで、不正アドレスのアクセスを Shinobi1 より見つけやすくなりました。

(2) SH4 の特権モード (レジスタバンク : SH4 レジスタ設定 : SR.RB = 1)

SH4 の特権モードでは、24 本の 32 ビット汎用レジスタにアクセスできます。

このうち 16 本のレジスタ (R0~R15) は通常アクセスが可能ですが、残りの 8 本はバンクレジスタ (R0_BANK~R7_BANK) としてアクセスします (詳しくは、「図 1-3 SH4 汎用レジスタ及びライブラリ使用範囲」参照して下さい)。

セガライブラリでは、通常時に R0_BANK1~R7_BANK1 が R0~R7 になるように設定しているため、通常時や割り込み/例外発生時のいずれの場合も、R0_BANK0~R7_BANK0 を使用することになります。

SR.MD=1, SR.RB=0

R0	R0_BANK0
R1	R1_BANK0
R2	R2_BANK0
R3	R3_BANK0
R4	R4_BANK0
R5	R5_BANK0
R6	R6_BANK0
R7	R7_BANK0

SR.MD=1, SR.RB=1 (アプリケーション実行時の設定)
通常時/割り込み・例外発生時とも

R0_BANK0
R1_BANK0
R2_BANK0
R3_BANK0
R4_BANK0
R5_BANK0
R6_BANK0
R7_BANK0

ライブラリ未使用

R0_BANK1	R0_BANK1
R1_BANK1	R1_BANK1
R2_BANK1	R2_BANK1
R3_BANK1	R3_BANK1
R4_BANK1	R4_BANK1
R5_BANK1	R5_BANK1
R6_BANK1	R6_BANK1
R7_BANK1	R7_BANK1

R0
R1
R2
R3
R4
R5
R6
R7

ライブラリ使用

R8	R8
R9	R9
R10	R10
R11	R11
R12	R12
R13	R13
R14	R14
R15	R15

図 1-3 SH4 汎用レジスタ及びライブラリ使用範囲

(3) CPU の丸め発生モード (丸めモード : SH4 レジスタ設定 : FPSCR.RM = 0)

浮動小数点命令の実行時に丸めが発生する場合の CPU の動作は、「近傍への丸め」に設定します。

ノ ー ト 「丸め」とは、演算結果の有効数字を少なくする場合の処理の意味です。
「近傍への丸め」は、丸めを行う場合に、表現可能な最も近い値に設定を行うという意味です。

注 意 Dreamcast SDK (以下 SDK に省略) Ver1.4J シリーズまでは、0 方向への丸め (丸めビット以下はすべて 0 にする) で動作しています。
現状のライブラリでの浮動小数点の丸めモードは、タイトル起動する前の SEGA ロゴ表示時に 1 度だけ設定をします。従って、SDK1.4J 以前の SDK で作成したアプリケーションと、SDK1.5J 以後のアプリケーションを交互に呼び出したりする場合には、FPSCR.RM を使用した SDK にあわせて再設定する必要があります。

1.4 アプリケーションが実行されるまで

ここでは、GD-ROM から最初の実行ファイル（通常 1st_read.bin）がメモリ上に読み込まれてから、ユーザーのアプリケーションプログラム main 関数が呼ばれるまでのプロセスを説明します。

1.4.1 概要

(1) アプリケーションの実行プロセス

アプリケーションが実行されるプロセスは、大まかに以下の順で処理されます。

- a. ハードウェアの初期化、SEGA ロゴの表示
- b. アプリケーション実行のための、共通の初期化（AIP=Application Initial Program）
- c. スタートアップモジュールの実行（strt1、strt2）
- d. ユーザーメインプログラム main（）関数の呼び出し

a.b.には IP（IP.BIN）に実行プログラムが含まれており、c.d.はアプリケーションにリンクされます。ELF ファイルを用いてデバッグしている場合は、a～d の全てがアプリケーションにリンクされます。

(2) アプリケーションの実行に必要な共通の初期化（AIP）の内容

- a. 割り込みを禁止にします。（SR.IMASK = 15）
- b. キャッシュコントロールレジスタ（CCR）をクリアします。
- c. プレイ履歴の記録を行うアプリケーションの場合は、起動の情報を本体のフラッシュメモリに書き込みます。
- d. SH4 のレジスタ初期値を設定します。

スタックポインタ	SP（R15）	0x8C00F400
ベクタベースレジスタ	VBR	0x8C00F400
ステータスレジスタ	SR	0x700000F0
浮動小数点ステータス / コントロールレジスタ	FPSRC	0x00040000

- e. アプリケーションの開始アドレスの P2 領域（0xAC010000）へ、ジャンプします。

(3) スタートアップモジュール（strt1.obj、strt2.obj）

- a. インストラクションキャッシュ（IC）の無効化をします。
- b. P1 領域に制御を移します。
- c. スタックポインタは（SP、R15）を 0x8C00F000 に設定し直します。0x8C00F000 から 0x8C00F3FF までの 1K byte はセガライブラリが使用します。
- d. 未初期化変数領域（BSS）を 0 クリアします。

オーバーレイを行う場合の BSS のクリアについては、実行するタイミングを考慮する必要があります。BSS のセクションの先頭アドレスがシンボル_BSG_BGN に、終了の次のアドレスがシンボル_BSG_END に設定されるため、以下のように設定をすれば BSS の 0 クリアができます。

リスト 1-1 未初期化変数領域のクリア

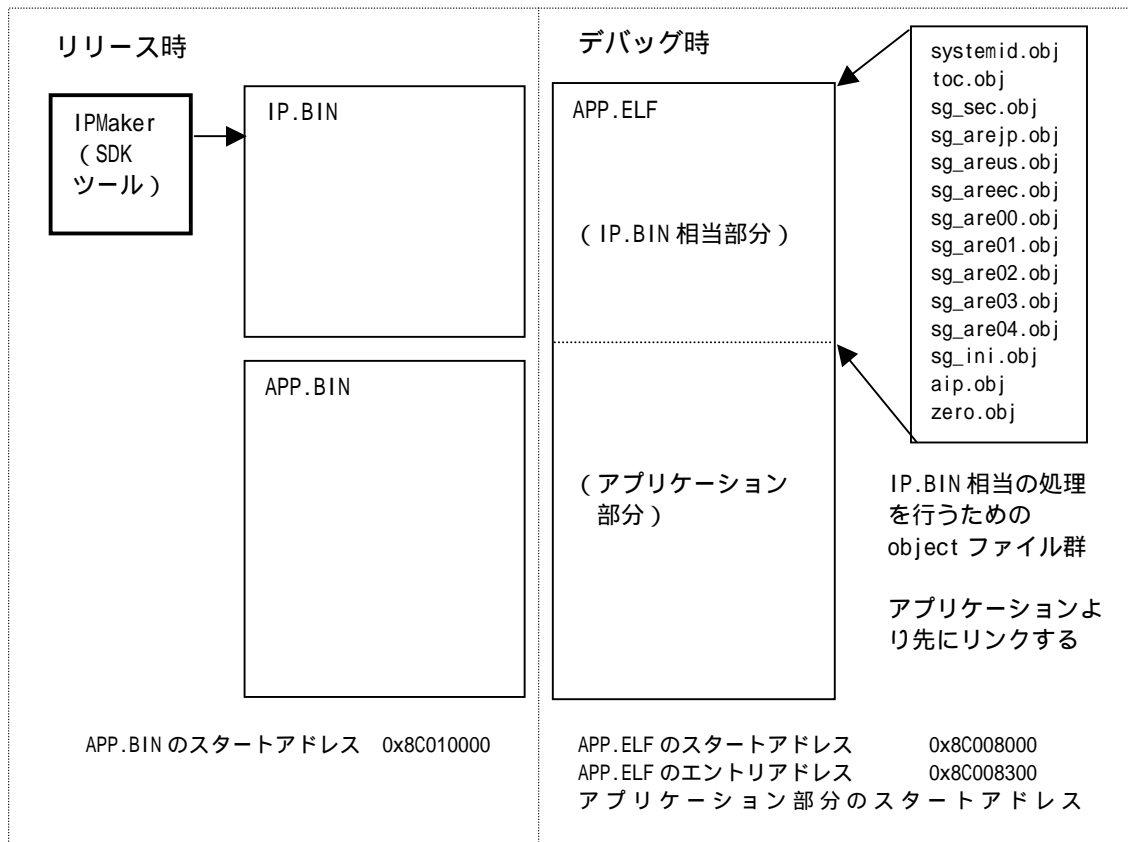
```
void ClearBSS(void)
{
    extern Uint8 *_B_BGN, *_B_END;
    extern Uint8 *_BSG_BGN, *_BSG_END;
    Uint8 *p;
    for (p = _B_BGN; p < _B_END; p++)
        *p = 0;
    for (p = _BSG_BGN; p < _BSG_END; p++)
        *p = 0;
}
```

- e. スタック領域 (0x8C00C000 ~ 0x8C00F000) を、"SEGASEGA..." という文字列でフィルします。
この領域をデバッガで表示すると、スタックが最大でどの程度利用されたかを知ることができます。
- f. main () 関数を呼び出します。

注 意 スタートアップ時に、C++言語のグローバルオブジェクトのコンストラクタの呼び出しを行う場合があります。しかしセガライブラリでは、アプリケーションが明示的に行う必要があります。詳細については、「9.2.1 グローバルオブジェクトのコンストラクタの初期化」を参照して下さい。

(4) デバッグ時の ELF ファイルについて

デバッグ時は IP を別にロードする手間を省くために、SDK のサンプルでの IP の部分のコードも ELF ファイルにリンクしています。



1.5 セクション及びセクション名について

ここでは、ユーザープログラムのセクションやセクションの名称について説明します。

1.5.1 セクションとは

セクションとは、プログラムをコード/データのように属性毎に分けたものです。アプリケーションをリンクする際には、このセクションの順にリンクされています。

1.5.2 セクション名について

セクションの名前は特に指定していない限りは、コンパイラがデフォルトの名称を出力します。デフォルトのセクションの名称は、SHC と CodeWarrior とで異なりますが、基本的に配置を同じに設定されるため、実行の際には影響がありません。

注 意 セガライブラリのセクションは、ユーザープログラムのセクションと異なるセクションを使用していますので、ご注意ください。

セクション名は大きく分けると、SHC コンパイラと CodeWarrior コンパイラの2種類に分かれます。

(1) SHC コンパイラ

ユーザープログラムは、P (コード)、C (定数データ)、D (初期化済みデータ)、B (未初期化データ)、セクションを使用します。

SHC コンパイラのリンカを使用した場合、セクションの配置はリンク時に、先に検出されたセクションの順に配置されます。

セガライブラリでは、この配置が一定に決まるようにスタートアップの中で全ての配置を「表 1-1 セクション一覧」のように定義しています。

アプリケーションでセクションを新たに定義すると、BSS (未初期化変数) セクションの次に配置されてしまいます。表 1 - 1 の No.20「特定ライブラリセクション」の中で、ユーザー用にいくつかセクションが定義されていますので、必要な場合はこちらから使用して下さい。

表 1 - 1 セクション一覧

#	セクション名	内 容	メモリ境界 (アライメントサイズ)	備 考
1	IP	IP プログラム用セクション (elf ファイルによるデバッグ時のみリンク)	アドレス固定	
2	DSGLH	アプリケーションの先頭ヘジャンプ	4	
3	DSGLE	予約	4	
4	PSG	ライブラリプログラム領域	4	
5	PSGSFD00 ~ 15	CRI Sofdec 専用プログラム領域	4	
6	PSGREVO ~ 3	ライブラリ用 プログラム領域 (予備)	4	
7	PUSER0 ~ 3	ユーザー用 プログラム領域 (予備)	4	
9	P	アプリケーションプログラム領域	4	
10	CSG	ライブラリ定数領域	4	
11	CSG32		32	
12	CSG_\$VTBL	ライブラリ仮想テーブル用セクション	4	
14	C	アプリケーション定数領域	4	
15	C32		32	
16	C_\$VTBL	ユーザー仮想テーブル用セクション	4	SHC のみ
17	.exception	例外ハンドラ用セクション	4	CW のみ
18	.exceptlist		4	CW のみ
19	.vtables	ユーザー仮想テーブル用セクション	4	CW のみ
20	DSG	ライブラリ初期化済みデータ領域	4	
21	DSG32		32	
22	DSGSFD00 ~ 15	CRI Sofdec 専用データ領域	4	
23	DSG_INIT_	ライブラリグローバルコンストラクタ領域	4	
24	DSG_END_	ライブラリグローバルコンストラクタ領域	4	
25	DSGRSV0 ~ 3	ライブラリ用 データ領域 (予備)	4	
26	DUSER0 ~ 3	ユーザー用 データ領域 (予備)	4	
28	D	アプリケーション初期化済みデータ領域	4	
29	D32		32	
30	D_INIT_	ユーザーグローバルコンストラクタ領域	4	SHC のみ
31	D_END_	ユーザーグローバルコンストラクタ領域	4	SHC のみ
32	RSG	(未使用)	4	
33	RSG32		32	
34	R	(未使用)	4	
35	R32		32	
36	BSG	ライブラリ未初期化データ領域	4	
37	BSG32		32	
39	B	ユーザー未初期化変数領域	4	
40	B32		32	
41	BGAP	アライメント調整用セクション	32	

網掛け部がライブラリの使用部分です。

セクション名でライブラリが使用するものは、P (コード)、C (定数)、D (データ)、B (BSS) のあとに、SG を付加します。更にアライメントが 32 Byte のものは 32 を付与します。

表 1 - 2 ユーザー用セクション一覧 (配置順)

No	セクション名	説 明	アライメントサイズ (境界)	備 考
1	PUSER0 ~ PUSER3	ユーザー用 プログラム領域 (予備)	4	CODE 属性 4 セクション
2	DUSER0 ~ DUSER3	ユーザー用 データ領域 (予備)	4	DATA 属性 4 セクション

(2) CodeWarrior コンパイラ

ユーザープログラムは、.text (コード)、.rodata (定数データ)、.data (初期化済みデータ)、.bss (未初期化データ)、セクションを使用します。

基本的にライブラリの使用するセクションは、(1) SHC コンパイラと同じです。配置も基本的には、SHC コンパイラと同じで、P .text、C .rodata、D .data、B .bss とそれぞれ読み替えて下さい。

CodeWarrior のリンク順は、スタートアップモジュールの定義ではなく、リンク時に参照されるテキストファイル (segalib2.lcf) で定義されます。

1.6 ヒープ領域の変更について

ヒープ領域は、`malloc()` などの関数がメモリを割り当ててる領域です。通常の設定 (`sbInit2.c`) では、プログラムの終わりからメモリの最後までを、ヒープ領域として割り当てます。

`sbInitSystem()` 関数の中で呼び出している `sy` 関数のパラメータを変更することで、ヒープの領域を変えることができます。

オーバーレイを実現するためにヒープ領域を変更して、空いた領域にオーバーレイの非常駐部をロードするようにします。

通常は、以下の設定になっています。

リスト 1-2 ヒープ領域の通常設定

```
...
extern UInt8* _BSS_END; /* BSG/BSG32 セクションの終了アドレス */
/* End address of BSG/BSG32 section */

#define WORK_START (0x8C000000) /* ワーク RAM の先頭アドレス */
#define WORK_END (0x8D000000) /* ワーク RAM の終了アドレス */

/* ワークとして領域を固定確保するサイズ */
/* _BSS_END から WORK_END-WORK_FIXED_SIZE までがヒープ領域になります */
/* WORK_END-WORK_FIXED_SIZE から WORK_END までが固定領域になります */
#define WORK_FIXED_SIZE (0x00000000)

/* syMalloc() の管理下に置くヒープの先頭アドレスとヒープのサイズ */
/* The top address of the heap area and heap size */
#define HEAP_AREA (((UInt32)_BSS_END | 0x80000000) + 0x1F) & 0xFFFFF0
#define HEAP_SIZE (WORK_END - WORK_FIXED_SIZE - HEAP_AREA)

...
/* メモリ管理の初期化 */
/* Initialize memory management. */
syMallocInit( (void*)HEAP_AREA, HEAP_SIZE );
...
```

`_BSS_END` には、プログラムの最後のアドレスがリンク時に設定されます。

`WORK_END` には、メモリの最後のアドレス `0x8d000000` が設定され、`HEAP_AREA` には、`_BSS_END` のアドレスを 32 バイトのアライメントに調整したアドレスが設定されます。

`HEAP_SIZE` には、使用可能なヒープサイズが設定され、これらの値で `syMalloc` ライブラリを初期化しています。

`WORK_FIXED_SIZE` に値を設定すると、メモリの最後からこのサイズ分だけ、ヒープから外されます。ヒープから外された領域は、ユーザーが自由に使用することが可能です。

- 例 メモリの最後 1M バイトをヒープ領域から外す

```
...  
#define WORK_FIXED_SIZE (0x00100000) /* ここのみ変更 */  
...  
syMallocInit (HEAP_AREA,HEAP_SIZE);  
...
```

注意 ヒープエリアの先頭を示すアドレスは、(1) 32 バイトアラインメント (2) P1 空間のアドレスを指定しなければならないことに注意して下さい。
また、HEAP_AREA のアドレスを変更する場合は、プログラムと衝突しないことをユーザー側で検証して下さい。

1.7 スタックの変更について

スタックのサイズが 12K バイトを超える場合などは、スタックの領域を変更する必要があります。スタックの変更は、(1) スタック領域の確保 (2) R15 レジスタの変更で行います。

(1) スタック領域の確保

ヒープ領域を変更して、その空きエリアなどを指定します。

(2) R15 レジスタの変更

C/C++からは直接できませんので、アセンブラを使用することになります。

また、R15 を変更するとその関数の内部で定義したローカル変数の参照や、親の関数に戻ることができなくなります。従って、main 関数を使ったスタックの変更後に、本来の main の関数を呼び出すなどの工夫が必要です。

● 例 メモリの最後 32K バイトをスタックとする

```
[main.c]
void main2(void) ;
void main(void)
{
    set_sp(0x8d000000) ;
    main2()
}
void main2(void)
{
    /* 本来の main() 関数 */
}

[sbinit.c]
#define WORK_FIXED_SIZE (32*1024)
sbInitSystem(...)
{
    ...
    syMalloc(HEAP_AERA, HEAP_SIZE) ;
    ...
}

[set_sp.src]
.export _set_sp
.section P, code, align = 4
_set_sp:
    rts
    movr4, r15
.end
```

1.8 ソフトウェアリセットの注意事項

ソフトウェアリセットを実現する際に問題となるのが、ライブラリの終了処理を行っていないために、再初期化することができなくなる場合があるということです。

ライブラリには、初期化関数を連続して2度コールした場合でも正常に終了するものがありますが、再初期化が必要な場合は原則として、初期化の順序と逆順に終了関数を呼び出した後で、再初期化を実行して下さい。

ノート ソフトウェアリセットに関しては、設計の最初に考慮しておくことで、後々のトラブルを避けることが可能です。

なお、Shinobi2 (SEGA Library Ver 2.0) から新たに追加された、サンプル及びフレームワークには、ソフトウェアリセットの処理も含まれています。

2 システムライブラリ（shinobi2.lib）の構成

ここでは、システムライブラリのライブラリの詳細について説明します。

2.1 システムライブラリについて

システム系ライブラリの shinobi2.lib は、割り込み / キャッシュ / メモリ / タイマの管理を行う低レベルの関数を含んでおり、他のライブラリに対して基本となるサービスを提供しています。

また、GD ファイルシステムやコントローラ及びバックアップメモリなどの各種ペリフェラルを制御するペリフェラルライブラリ等も含んでいます。

表 2 - 1 システムライブラリの構成

No.	対象カテゴリ	ライブラリ	ライブラリコード	備考
1	CPU (SH4) 基本サービス	割り込みライブラリ	syInt、syChain	
2		キャッシュライブラリ	syCache	
3		ストアキューライブラリ	sySq	
4		メモリマネージメントライブラリ	syMalloc	
5		コールバックライブラリ	syCallback	
6		タイマライブラリ	syTmr	
7		システムマネージャライブラリ	syMng	
8		ファイバライブラリ	syFbr	
9	BootROM	BootROM サービスライブラリ	syBt	
10		BootROM フォントライブラリ	syBtFnt	
11	GD-ROM	GD ファイルシステムライブラリ	gdFs	
12	ペリフェラル	ペリフェラルデータライブラリ	pd	
13		振動デバイスライブラリ	pdVib	
14		LCD デバイスライブラリ	pdLcd	
15		ガンデバイスライブラリ	pdGun	
16		マウスライブラリ	pdMouse	
17		記録デバイスライブラリ	bu	
18		音声入力デバイスライブラリ	ws	
19	フラッシュメモリ	コンフィグレーションライブラリ	syCfg	
20		プレイ履歴ライブラリ	uf	
21	その他	メモリコピーライブラリ	syMemCopy	
22		メモリセットライブラリ	syMemSet	
23		リアルタイムクロックライブラリ	syRtc	
24		ビデオ設定ライブラリ	syVideo	
25		シリアルライブラリ	syCom	
26		拡張機能判別ライブラリ	syExtChk	

2.2 各ライブラリの概要

(1) CPU (SH4) 基本サービス

- 割り込みライブラリ (syInt、syChain)
ライブラリ向けに、割り込みハンドラの登録・削除の機能を提供します。また割り込み発生時に、割り込みハンドラの呼び出しの制御を行います。
このライブラリは、ライブラリ向けであり、アプリケーション開発者向けには、別途コールバックライブラリが提供されています。
- キャッシュライブラリ (syCache)
SH4 のキャッシュメモリの操作及びキャッシュメモリのモードの取得・変更を行います。
- ストアキューライブラリ (sySq)
ストアキューを使用するために、必要なアドレス変換の機能を提供します。
- メモリマネージメントライブラリ (syMalloc)
ヒープメモリの管理を行うライブラリです。C 標準関数の malloc() 関数及び free() 関数等に相当する機能を提供します。
- コールバックライブラリ (syCallback)
アプリケーション開発者向けに、特定のハードウェアへの割り込みに対して、コールバック関数の登録が可能な機能を提供します。
- タイマライブラリ (syTmr)
タイマライブラリは、常に動き続けるタイマ (フリーランタイマ) の値を取得する機能と、一定時間の経過後に割り込みを発生させるタイマ割り込みの機能などを提供します。
- システムマネージャライブラリ (syMng)
システムマネージャライブラリは、次項目のファイバライブラリを使用して、5 つのファイバを生成します。それぞれのファイバに関数を登録すると、登録された関数の一定のルールでスケジューリングします。
- ファイバライブラリ (syFbr)
ファイバライブラリは、非常に単純なスレッド (=ファイバ) のライブラリです。
基本的に、ファイバを切り替えるためには、ユーザーがそのための関数を呼び出し実行する必要があります。

(2) Boot ROM

- Boot ROM サービスライブラリ (syBt)
このライブラリでは、BootROM メニューに戻る機能、及びディスク交換時に利用するディスクの種別を判定する機能、ディスクの ID を取得する機能を提供します。
- BootROM フォントライブラリ (syBtFnt)
BootROM フォントライブラリは、ブートロムにビットイメージで格納されているフォントデータを取得するためのライブラリです。

(3) GD-ROM

- GD ファイルシステムライブラリ (gdFs)
GD-ROM のファイルシステムを提供するためのライブラリです。また、GDDA に関する機能も提供します。

(4) ペリフェラル

- ペリフェラルデータライブラリ (pd)
コントロールポートに接続されたペリフェラルからの入力データ取得、及びペリフェラル自身の情報の取得を行うライブラリです。
- 振動デバイスライブラリ (pdVib)
コントローラ等に接続された、「ぶるぶるぱっく」をはじめとする振動デバイスを制御するライブラリです。
- LCD デバイスライブラリ (pdLcd)
コントローラ等に接続されたメモリカード (ビジュアルメモリ) のモノクロ液晶をコントロールする機能を提供するライブラリです。
- ガンデバイスライブラリ (pdGun)
ドリームキャスト・ガンを取り扱うための機能を提供するライブラリです。
- キーボードデバイスライブラリ (pdKbd)
「ドリームキャスト・キーボード」を利用するためのライブラリです。
- マウスライブラリ (pdMouse)
コントロールポートに接続されたマウスを制御するためのライブラリです。
- 記録デバイスライブラリ (bu)
メモリカード (ビジュアルメモリ) へのファイルの保存・読み込みを行います。
- 音声入力デバイスライブラリ (ws)
音声入力デバイスから入力されるデータを取得するためのライブラリです。
- AV ケーブルペリフェラルライブラリ (syCbl)
AV 端子に接続されている AV ケーブルペリフェラルの種類を判別するライブラリです。

(5) フラッシュメモリ

- コンフィグレーションライブラリ (syCfg)
ブートロム画面の設定メニューにある「言語」「音声」の内容を取得するライブラリです。
- プレイ履歴ライブラリ (uf)
Dreamcast 本体のフラッシュメモリに記録された、アプリケーション毎のプレイ履歴 (タイトルの起動回数など) の参照及び更新等を行うライブラリです。

(6) その他

- メモリコピーライブラリ (syMemCopy)
- メモリセットライブラリ (syMemSet)
メモリ配置により最適なメモリコピー機能を提供するライブラリです。
従来、Ninja ライブラリに含まれていたメモリコピーライブラリ (njMemCopy) を、システムライブラリである shinobi2.lib に移動し、プレフィックスを変更しました。
また、メモリセットライブラリを新規に追加しました。
- リアルタイムクロックライブラリ (syRtc)
Dreamcast のカレンダー (日付・時間) の取得及び更新を行う機能を提供するライブラリです。

- ビデオ設定ライブラリ (syVideo)
ビデオ関連の設定や状態 (VGA / NTSC / PAL 等) の取得を行うライブラリです。
従来のケーブルライブラリ (syCbl) を廃止し、代わりにこのライブラリを使用します。
- シリアルライブラリ (syCom)
SH4 内蔵の SCIF (FIFO 内蔵シリアルコミュニケーションインタフェース) を利用したシリアル通信を、可能にするライブラリです。
- 拡張機能判別ライブラリ (syExtChk)
Dreamcast の EXTENSION 端子に、周辺機器 (モデム / LAN アダプタ / その他) が接続されているかどうかを判別するライブラリです。

3 CPU（SH4）基本サービス関連のライブラリ

ここでは、Dreamcast 本体のシステム関係に関わる基本的な設定に関連したライブラリを説明します。

3.1 割り込みライブラリ（syInt、syChain）について

ライブラリ向けに、割り込みハンドラの登録・削除の機能を提供します。また割り込み発生時に、割り込みハンドラの呼び出しの制御を行います。

このライブラリは、ライブラリ向けであり、アプリケーション開発者向けには、別途コールバックライブラリが提供されています。

3.1.1 SH4 の割り込み処理

SH4 では例外や割り込みの発生時に、PC と SR の値を SPC と SSR にそれぞれコピーし、SR.BL=1、SR.MD=1、SR.RB=1 とした後、ベクタ領域にジャンプします。

割り込み / 例外の区別	ジャンプ先
例外	VBR+0x100
MMU 例外	VBR+0x400
割り込み	VBR+0x600

VBR（ベクタベースレジスタ）には、アプリケーション起動時に 0x8C00F400 を設定し、ベクタ領域は 0x8C00F400 – 0x8C00FBFF になります。

例外要因による飛び先の変更は、一般例外、MMU 例外、割り込みの区別のみで、詳細な例外の要因は、例外事象レジスタ、割り込み事象レジスタに保存されます。

外部割り込みを受け付けた場合には、割り込み発生元の割り込み要因レジスタをチェックして、更に割り込み要因の特定を行う必要があります。

3.1.2 モジュール構成

(1) syInt

初期化時に、ベクタ領域へ SH4 の割り込み要因を特定する割り込み処理ルーチンを登録し、基本的な割り込みの受付とレジスタの待避を行います。

このとき同時に SR.BL=0、SR.IMASK=15 として、上位ライブラリが登録したハンドラ内部で、多重割り込みを受け付ける準備を行います。その後で、上位ライブラリから登録されたハンドラを呼び出します。

上位ライブラリのハンドラから処理が戻ってきたら、待避していたレジスタを復帰し割り込み処理（RTE 命令）から抜けます。

syInt は、割り込み処理時に使用するスタックを切り替えることができます。具体的には、初期化時に割り込み処理で使用するスタックを指定しておくこと、割り込み処理受付時にスタックを切り替えて処理を行い、割り込み処理終了時には、元のスタックに復帰します。

ファイバライブラリ（後述）を使用する場合には、割り込み中は専用のスタックを使用するように設定する必要があります。

(2) syChain

syChain は syInt の上位ライブラリで、上位ライブラリに割り込みプライオリティー付きチェイニングを提供します。

ノート プライオリティー付きチェイニングとは、一つの割り込み要因に複数のハンドラを実行順序の指定付きで登録できる機能です。

各ライブラリがチェイニング機能を使用することで、新しいライブラリを追加した際にユーザーが VSYNC 割り込み関数をそのたびに変更する必要がなくなりました。

また、割り込み発生時の実行順序が指定できるため、各ライブラリのサーバ関数の実行順序を初期化関数の呼び出し順序に関わらず、最適の状態に保つことが可能になっています。

SH4 内部では、外部デバイスからの割り込みが割り込みレベル毎で1つしか特定できないため、syChain が外部にある割り込みコントローラの要因レジスタをチェックし、それぞれの割り込みを独立した要因としてハンドラ登録を可能にしています。

Shinobi2 ライブラリでは、Chain 関数の一部を公開し、ユーザーが作成した複数のハンドラを実行順序付きで、割り込みに登録することが可能になりました。

3.1.3 CodeScape デバッガとの連携

(1) ハンドラが登録されていない割り込みの扱い

割り込みハンドラが登録されていない割り込みが発生した場合に syInt は、CodeScape にそれを通知する機能を持っています。これにより CodeScape は、ハンドラが存在しない例外が発生したときに、ユーザーに対し通知を行います。

また、Dev.Box 以外の環境で動作している場合には、CodeScape に通知をせずに、例外要因が無くなるまで無限ループに陥ってしまいます。

(2) プロファイラ使用時の処理

CodeScape でプロファイルを行う際には、実行速度が低下するために通常よりも割り込み処理が多く発生し、プロファイリング結果に割り込み処理が過大に評価されてしまいます。

syInt には、プロファイリングを行う場合に対して、割り込みの入口でプロファイリングを一時停止し、割り込みの出口でプロファイリングを再開するためのコードが含まれています。通常はこのコードが実行されることはありませんが、CodeScape 上でプロファイリングを行うと、CodeScape によりこのコードが実行されるようになり、割り込み処理中の実行時間がプロファイリングの結果に影響を与えないようにします。

3.2 キャッシュライブラリ (syCache) について

SH4 のキャッシュメモリの操作及びキャッシュメモリのモードの取得・変更を行います。

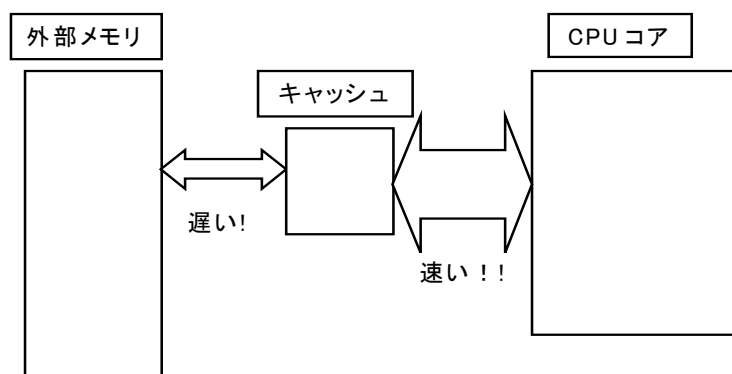
3.2.1 キャッシュとは

コンピュータ上は、通常 CPU は非常に高速な動作 (Dreamcast の CPU-SH7091 の場合 200MHz) をしていますが、それに比べメモリへのアクセスは非常に低速なため、CPU 本体が高速で動作していてもメモリアクセスが遅いため、高速な処理の足を引っ張ってしまう形になります。

キャッシュとは、遅いメインメモリと速い CPU との橋渡しをして、CPU の性能を落とさないためのモジュールのことで、非常に高速にアクセスが可能な RAM で構成されています。

もしキャッシュの中にメインメモリのコピーが保存されていれば、わざわざ遅いメインメモリを介さずに、高速にメモリの情報を引き出すことが可能になるのです。

ただし、こういった高速なアクセスが可能な RAM はコストがかさんでしまうため、メインメモリとしては使用できませんが、キャッシュなどのわずかな RAM に使用するのであれば、CPU のパフォーマンスを向上させることが可能です。



キャッシュは、外部メモリの内容をアドレス (場所) の情報と一緒に保存しています。

最初の読み込みは、時間がかかる外部メモリにアクセスしても、次に同じところの情報を引き出すときには、キャッシュに溜まっている内容を読みに行くので簡略化され速くなります。

3.2.2 キャッシュの仕組み

一般にキャッシュとして積まれているメモリは、メインメモリから比べると非常にわずかなものです。SH7091 の場合は、命令キャッシュ 8KB、データキャッシュ 16KB です。メインメモリ上のデータが、すべてキャッシュの中に収まるわけではありませんので、少ないキャッシュの RAM 上を効率良く使用する必要があります。

(1) キャッシュの構成

キャッシュは、構成上以下の 2 種類に分けられます。

- **アドレスアレイ**
メインメモリ上のアドレスを示します。
- **データアレイ**
メインメモリ上のアドレスの中身のデータを示します。

ノート 一つのアドレスアレイに対して、データアレイは 32 バイトあります。
この組をエントリと呼び、命令キャッシュで 256 エントリ、データキャッシュで 512 エントリあります。

(2) CPU のメモリアクセス

CPU がメモリアクセスする場合、まずキャッシュに該当するメモリの情報の有無を調べます。

この時、キャッシュ上に求めるアドレスのデータがあった場合（キャッシュヒット）は、そこをデータを拾ってきます。

キャッシュ上にデータが存在しない場合（キャッシュミスヒット）、メインメモリ上からデータを取得（キャッシュミスヒット時のペナルティ）します。

キャッシュミスヒット時には、一つのエントリをすべて埋めるために目的のアドレスの前後 32 バイト分をまとめて取得して、キャッシュエントリを埋めます。ミスヒット時には、ヒット時に比べて多くの時間が掛かってしまいます。これは、リード時であってもライト時であっても同様に、キャッシュにミスヒットがあった場合に必ずそのアドレス付近の 32 バイトを取得し、キャッシュエントリを埋めます。

ただし、キャッシュの資源に限りがありますので、いつでも利用が可能なキャッシュエントリが見つかるというわけではありません。

その場合にキャッシュコントローラは、一番後に使われたキャッシュエントリを選択し、そのエントリの不整合を修正した後、別のエントリとしてメモリ上から 32 バイトのデータを取り込みます。

この、キャッシュラインを選択するアルゴリズムを LRU といいいます。

参 照 詳細については、一般的な CPU（Pentium、PowerPC 等）の解説書等をご参照下さい。

3.2.3 ライトバック

CPU がメモリに対してアクセスを行う場合、キャッシュにエントリがあれば、そこから読み込むだけでなく書き込むことも行うことができます。

書き込んだ場合には、そのデータアレイにはダーティビットというビットが立てられ、メインメモリとキャッシュの間で不整合が発生していることが示されます。

当然キャッシュに書き込むだけでは、メインメモリへのデータの更新は行われませんが、メインメモリに対して読み書きするのに比べて、はるかに高速にアクセスすることができます。

極力、メインメモリへのアクセスを減らしキャッシュの中だけで完結させることで処理の高速化が図れます。

しかし、この状態ではキャッシュの内容と、メインメモリの実体の間ではデータに不整合が起こります。

この状態を回避するために、あるタイミングでキャッシュの内容をメインメモリに書き戻し、キャッシュとメインメモリの間の不整合を解消させる必要があります。

これをライトバックといいいます。

3.2.4 キャッシュのコヒーレンシ

CPU がメモリにアクセスする場合には、必ずキャッシュを通して行われます。

ところが、CPU モジュールの中には、DMA などのキャッシュを介さないで直接メモリにアクセスできるような場合も存在します。

DMA がキャッシュのエントリに含まれるアドレスに対して、データライト等の処理を行った場合には、キャッシュで管理しているデータと、実際のメモリの間には不整合が発生してしまいます。

このような、キャッシュとメインメモリの関係をコヒーレンシといい、上記のような場合には、コヒーレンシがあていない状態となります。

3.2.5 キャッシュのインバリデート

DMA などのアクセスによって、メインメモリとキャッシュの間に不整合が生じ、それがライトバックなどの手段でも、整合性を保つことができなくなった場合、そのエントリのキャッシュを無効化して、再度キャッシュラインにデータを取り込む必要が出てきます。

この操作をキャッシュのインバリデート（無効化）といい、エントリのインバリデートと全キャッシュのインバリデートに別れます。

この操作を行った場合、キャッシュのデータが破棄されインバリデートされたキャッシュエントリはアドレスアレイの有効ビットがおろされ、次のキャッシュ登録の候補となります。

セガライブラリにおいて DMA が発生する典型的なものは GD へのファイルアクセスですが、GDFS ライブラリにおいては、データ転送前にデータキャッシュのインバリデートを行います。

そのため通常、アプリケーションにおいてはインバリデートを意識する必要はありませんが、オーバーレイなどの目的でプログラムを読み込んだ場合には、意識する必要があります。GDFS ライブラリは命令キャッシュをインバリデートすることではなく、そのままオーバーレイされたプログラムを実行した場合、プログラムエリアの中にキャッシュにエントリがあるアドレスが含まれた場合、誤動作を起こします。

GD よりプログラムモジュールを読み込んだ場合は、読み込んだエリアに対して必ず、命令キャッシュのインバリデートを行う必要があります。

3.2.6 キャッシュのページ

キャッシュをライトバックした場合、そのキャッシュエントリは有効で、またそのエントリで示すアドレスに対して CPU がアクセスしようとした場合は、そのキャッシュエントリがそのまま利用されます。

しかしながら、もしライトバックしたエントリのアドレスにはアクセスする必要が無い場合、いつまでも必要の無いキャッシュエントリ持っていることは、キャッシュの効率上、あまり好ましいものではありません。

この場合、そのキャッシュエントリを書き戻した上、無効化してしまうほうが都合がよくなります。そういった操作はページと呼ばれ、ページしたキャッシュエントリは有効ビットがおろされ、次のキャッシュエントリの登録候補となります。

3.2.7 キャッシュのプリフェッチ

これまでに説明してきましたように、キャッシュに対してアクセスする場合、キャッシュヒットするか、キャッシュミスヒットするのかわかればパフォーマンスは大きく変わってきます。

キャッシュミスヒット時のペナルティはできるだけ避けるためには、できる限り効率のよいキャッシュの使い方をすると、前もって、キャッシュミスヒットすることが分かっている場合、キャッシュエントリを予め埋めておくという方法があります。

この後者の方法を実現するのがキャッシュのプリフェッチとなります。

プリフェッチは、データキャッシュに対してのみ行うことができます。

プリフェッチを有効に使用するためには、でき上がったアセンブラコードを追いかけて、何処でキャッシュミスヒットが発生するのかわかるとか、パイプラインシミュレータでデータのロード・ストアのパイプライン遅延が発生しているところを調べて挿入するなどの高度なテクニックを必要とします。

3.2.8 キャッシュエントリの確保

メモリアクセス時に、そのアドレスの中身に何が書かれていても、内容は更新される場合、例えば、バッファとして確保している領域で、その領域には一方的に計算結果が格納されるような場合、そこに何が書かれていようが、新しいデータで更新されることが分かっている場合に、普通にそのアドレスをアクセスし、そのアドレスがキャッシュエントリに含まれていない場合、通常のキャッシュ動作では、まず、そのアドレス付近 32 バイトでキャッシュエントリを埋め、その後該当するアドレスを示すキャッシュのデータアレイに対して書き込みます。

しかしこの操作は、一旦 32 バイトのキャッシュエントリを埋めるためにメインメモリから 32 バイトのデータを取得するという作業が発生するため、効率がよくありません。

このような、特別な場合にメモリからデータを取得することなくキャッシュエントリを確保するアセンブラ命令が用意されています。

MOVCA.L R0 @Rn

セガライブラリには、関数としてこの命令を実行するものが備わっています。

3.2.9 キャッシュの RAM モード

SH7091 には、データキャッシュの半分（8KB）を RAM として利用するモードが備わっています。

キャッシュの仕組み上、キャッシュ RAM を普通の RAM として用いた場合に、非常に高速なアクセスとなる事が分かっています。

しかしながらセガライブラリでは、アプリケーションがキャッシュを RAM モードにすることは禁止します。アプリケーションが RAM モードで正常動作する事を保証できないからです。

3.2.10 キャッシュ操作が必要な場面

セガライブラリ環境ではアプリケーションが独自に DMA を行う場面は存在しないためライブラリが提供するメモリ転送系関数の前や後に使用することになります。

(1) メインメモリ上にあるテクスチャを書き換えた直後

Ninja ライブラリの以下の関数では、データの転送に DMA を使用するため、メモリ上のデータを更新して関数コールを行う場合は、該当エリアのオペランドキャッシュのライトバック（メモリへの書き戻し）が必要です。

GD-ROM から読み出す場合は GDFS が内部でキャッシュライブラリを使用して必要なキャッシュ操作を行っているため、ユーザーによる操作は必要ありません。

表 3 - 1 キャッシュの操作が必要になる場合がある関数のリスト

njLoadTexture	njLoadTextureNum
njReLoadTextureNum	njReLoadTextureNumG
njLoadTexturePvmMemory	njReLoadVQCodebookNum
njReLoadVQCodebookNumG	njReLoadRectangleTexturePartNum
njReLoadRectangleTexturePartNumG	njReLoadTexturePartNum
njReLoadTexturePartNumG	njLoadTextureReq
njLoadTextureYUV420Num	njLoadTextureYUV420NumG
njLoadTexturePartLow	

(2) プログラムのロード後

GD から読み込んだプログラムを実行する前には、インストラクションキャッシュ（IC）の無効化が必要です。

GDFS は、ファイル読み込み後、オペランドキャッシュ（OC）のコヒーレンシは保証していますが、読み込むデータに対してプログラムかデータかの区別はできません。IC を常に無効化するのは不都合なので、ユーザー側で対処することとしています。

(3) ストアキュー使用時

その他、ストアキューを使用した場合にキャッシュとは全く別システムのメモリアクセスになるため、CPU からストアキューで書出したデータを読み出すためには、インバリデートが必要になります。

3.2.11 キャッシュの操作で注意すべき点

syCache ライブラリに指定できるキャッシュ操作の単位は、最低でもキャッシュライン単位（32 バイト）なので、実際にキャッシュ操作したい領域の先頭が、32 バイト境界に揃っていなかったり、後端が 32 バイトに揃っていなかったりすると、関係のないデータが壊される恐れがあります。

キャッシュ操作のミスによる動作不良は、メモリアクセスウォッチなどのブレークにはヒットしませんので、不良の特定が非常に困難で、またタイミングによっても起きたり起きなかったりします。実際にキャッシュ操作したアドレスが、たまたまコヒーレンシが保たれた状態であったり、そ

うでなかったりと動作状態によって変わるため、キャッシュの操作は細心の注意を払って行って下さい。

3.2.12 関数一覧

キャッシュライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syCacheICI	インストラクションキャッシュのすべてのエントリをインバリデート
syCacheInit	キャッシュ関数の初期化
syCacheMOVCAL	全キャッシュブロックに対する MOVCAL の実行
syCacheOCBI	全キャッシュブロックのインバリデート
syCacheOCBP	全キャッシュブロックのパージ
syCacheOCI	オペランドキャッシュのすべてのエントリをインバリデート
syCacheOCWB	全キャッシュブロックのライトバック
syCachePREF	指定された領域のプリフェッチ

3.3 ストアキューライブラリ (sySq) について

ストアキューを使用するために、必要なアドレス変換の機能を提供します。

3.3.1 ストアキューとは

ストアキューとは、SH4 の持つ外部メモリに対し高速転送を行うための書き込み専用のバッファのことです。

バッファの容量は (32 Byte × 2) のダブルバッファ構成となっており、片方のストアキューがメインメモリに対し転送を行っている間に、もう一方のストアキューに対して CPU からの書き込みを行えます。

3.3.2 Shinobi2 への変更点

Shinobi2 ライブラリでは、MMU がイネーブル設定になっているため、ストアキュー転送を行うためには、MMU のエントリにストアキュー転送用の仮想アドレスを割り当てる必要があります。この関数では、その割り当てを行います。

3.3.3 関数一覧

ストアキューライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
sySqInit	ストアキューライブラリの初期化
sySqFinish	ストアキューライブラリの終了処理
sySqStart	ストアキュー用の仮想アドレスの取得
sySqEnd	ストアキュー用の物理アドレスを取得

3.4 メモリ管理ライブラリ (syMalloc) について

ヒープメモリの管理を行うライブラリです。C 標準関数の malloc()関数及び free()関数等に相当する機能を提供します。

ノート セガライブラリにおいては、SHC++の new や、SHC ライブラリ関数の malloc、free など、syMalloc ライブラリ内の関数を使用するようになっています。

sbInitSystem 関数の中で定義されているように、通常はアプリケーションプログラムの後ろ端からメインメモリの後ろ端までをヒープ領域として管理するようになっていますが、syMallocInit の引数でどの領域を syMalloc に与えるヒープとするかを設定できるようになっています。

3.4.1 syMalloc のアルゴリズム

(1) FirstFit

syMalloc は、フリーメモリのリストをたどり、要求サイズを超えるフリーブロックを発見すると、即座にそのフリーブロックの後ろ端にブロックを生成し、ユーザーに与えます。このため、(特に工夫していない) BestFit 型アルゴリズムよりは、syMalloc の検索時間は短くなることが期待できます。

また、アルゴリズムが単純なために、動作が予測し易く、ユーザーが適切にアプリケーションのメモリ確保をコントロールすれば、効率の良いメモリの確保が可能です。

しかし、malloc の動作を予測すること無しに、アプリケーションが特別な確保と開放の順序の制御を行わない場合には、BestFit アルゴリズムの方が統計的にメモリ効率が良くなることも知られています。

(2) ヒープの使用方法

syMalloc は、ヒープは高位のアドレスから使用していきます。

これは、アプリケーションの比較的先頭で確保されることが多い Ninja ライブラリの頂点バッファを、できるだけ SDRAM の後ろ側に配置して高速なメモリアクセスを実現するためです。詳しくは、第 3 部 Ninja2 ライブラリ編 「11 高速化&テクニック」を参照して下さい。

従って、メモリを開放せずに確保を繰り返すと syMalloc の返すアドレスが、徐々にアドレスの低位に移動していきます。

(3) syMalloc での効率的なメモリ利用

メモリの分断化が起きていない状態を保ちつづけるためには、寿命の長いメモリをはじめに取得し、メモリ確保と開放のタイミングを入れ子構造になるように(後で確保したブロックを先に開放するように)することで分断を避けることができます。

syFree を行った後で syMalloc を行う場合には、その時点で開放可能な領域をすべて開放してから次の syMalloc を行うようにして、できる限り syFree と syMalloc が交互に呼ばれないようにするのが効率的なメモリ利用の方針の 1 つとなります。

リスト 3 - 1 分断化の例

```
Heap_1 = syMalloc (SIZE_1);
heap_2 = syMalloc (SIZE_2);
heap_3 = syMalloc (SIZE_3);
heap_4 = syMalloc (SIZE_4);
syFree (heap_4);
syMalloc (SIZE_4); /* ここで heap_4 の領域が使われてしまうため */
/*分断化が発生する */

syFree (heap_2);
syFree (heap_3);
```

リスト 3-2 分断化しない例

```

heap_1 = syMalloc (SIZE_1);
heap_2 = syMalloc (SIZE_2);
heap_3 = syMalloc (SIZE_3);
heap_4 = syMalloc (SIZE_4);
syFree (heap_2); /* heap_2, heap_3, heap_4 の開放順は分断とは関係ない */
syFree (heap_3);
syFree (heap_4);
syMalloc (SIZE_4); /* ここでは、heap_2, heap_3, heap_4 で開放された領域を含んだヒープから
一番高位のアドレスが使用されるため、分断化は起きない */

```

3.4.2 syMallocChangeIF 関数による malloc 関数の入れ替え

(1) syMallocChangeIF とは

分断化を避ける際に、上記の方法を取ればある程度のメモリの分断化は防げるはずなのですが、実際にはアプリケーションの構造上、そうならない場合も多くなります。

また、それぞれのアプリケーションの開発者が、それぞれの開発に必要なデバッグ機能やメモリ管理の手法を含んだ malloc ライブラリを利用している場合もあります。

このような場合のために、ユーザーの作成した malloc ライブラリを標準の malloc と置き換える機能が提供されています。それが syMallocChangeIF です。

この関数により、syMalloc (malloc)、syFree (free) などの関数の実体をアプリケーションの開発者が作成した malloc ライブラリを用いて置き換えることができます。

セガライブラリの関数の中で、syMalloc を使用している関数からの呼び出しも置き換えられますので、すべての syMalloc、malloc (C++ の new) のときに使用される malloc を置き換えることができます。

(2) 使用方法

新しく作成した malloc ライブラリが次のような関数群であるとする、syMallocChangeIF は次のようにして使用することができます。

リスト 3-3 syMallocChangeIF の使用方法

```

新しい malloc :
void *myMalloc ( Uint32 size );
void myFree ( void *ap );
void *myCalloc ( Uint32 nobj, Uint32 size );
void *myRealloc ( void *ap, Uint32 nbytes );

使用例 :
SYS_MALLOC_IF myMallocIF; /* これから変更する malloc ライブラリ */
SYS_MALLOC_IF *oldMallocIF; /* syMallocChangeIF をする前の malloc ライブラリ */
MyMallocIF.malloc = myMalloc;
MyMallocIF.free = myFree;
MyMallocIF.realloc = myRealloc;
MyMallocIF.calloc = myCalloc;
OldMallocIF = syMallocChangeIF ( &myMallocIF );

```

(3) 制限事項

syMallocChangeIF 関数を使用する際は、以下の点に注意して下さい。

a. アライメント

ただし、syMalloc の返すアドレスは 32 バイト境界にアライメントされていることが保証されている必要があるため、syMallocChangeIF で置き換える malloc ライブラリも、その条件を満たしている必要があります。

ただし、セガライブラリ関数が実行されるときにだけ、再度 syMallocChangeIF を呼び出した上で syMalloc を使用して元に戻す場合には、その必要はありません。

b. ヘッド領域

日立製の SHC++を使用するときには、SHC++がメモリブロックヘッダ内に予約領域を必要とするため、次のような制限があります。

具体的には、malloc が返すアドレスを Uint32 *ptr とした場合に、ptr-1 と ptr-2 の領域を SHC++が予約しているため、ユーザーの malloc ライブラリはこの領域を空けておく必要があります。

つまり、確保した前方 8 バイトのメモリ領域を空けておく必要があるということです。

c. malloc と free の関係

現在の syMallocChangeIF で free を行うときには、malloc が行われたときと同じ malloc ライブラリが設定されている必要があります。

3.4.3 関数一覧

メモリマネージメントライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syCalloc	メモリの確保
syFree	メモリ領域の開放
syMalloc	メモリの確保
syMallocChangeIF	malloc 系関数の入れ替え
syMallocFinish	Malloc システムの終了
syMallocInit	Malloc システムの初期化
syMallocStat	メモリ使用状況の取得
syRealloc	確保したメモリのサイズ変更

3.5 コールバックライブラリ (syCallback) について

アプリケーション開発者向けに、特定のハードウェアへの割り込みに対して、コールバック関数の登録が可能な機能を提供します。

3.5.1 コールバック関数の初期化

syCallbackInit を使いコールバック関数を初期化します。

3.5.2 コールバック関数の登録

syCallbackAddHandler を使いコールバック関数を登録します。

戻り値となるコールバックハンドルは、登録したコールバック関数を削除するときに用います。コールバック関数の登録を削除するには、syCallbackDeleteHandler()関数を用います。

ノート コールバック関数登録の関数 syCallbackAddHandler() で同じイベント番号を指定しても上書きはされません。

注意 登録したコールバック関数は、割り込み状態で実行されるため、処理内で割り込みを必要とする処理を行わないで下さい。

3.5.3 コールバック関数の終了

syCallbackFinish を使いコールバック関数を終了します。

3.5.4 登録したコールバック関数の削除

syCallbackAddHandler を使いコールバック関数を登録した関数を、syCallbackDeleteHandler を使って削除します。

3.5.5 関数一覧

コールバックライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syCallbackAddHandler	コールバック関数の登録
syCallbackDeleteHandler	登録したコールバック関数の削除
syCallbackFinish	コールバックライブラリの終了
syCallbackGetNestCount	割り込みのネスト回数の取得
syCallbackInit	コールバックライブラリの初期化

3.6 タイマライブラリ (syTmr) について

タイマライブラリは、常に動き続けるタイマ (フリーランタイム) の値を取得する機能と、一定時間の経過後に割り込みを発生させるタイマ割り込みの機能などを提供します。

3.6.1 SH4 のタイマユニットとの関連

SH4 には 3 つの 32 ビット (ダウンカウンタ) タイマを内蔵しています。これらは、TMU0、TMU1、TMU2 と区別されています。タイマライブラリは、これら 3 つのタイマをフリーランタイム、汎用タイマ、ライブラリ予約タイマというように 1 対 1 に割り当てることで機能を実現しています。

3.6.2 タイマライブラリの関数の分類

タイマライブラリは、大きく以下の 2 つのグループ関数群に分類されます。

- (1) フリーランタイム関数群
- (2) 汎用タイマ関数群

(1) フリーランタイム関数群

TMU0 はフリーランタイムとして使用します。このタイマ値を取得することで、どれだけ時間が経過したかなどの情報を知ることができます。

フリーランタイムが初期設定時の内容で常に動作し続けていることを前提の上で、他のライブラリ (タイムアウトチェック等) も作られています。

注意 ハードウェアを直接操作するなど、フリーランタイムを停止させたり解像度の変更をしてはいけません。

a. フリーランタイムの値の参照

フリーランタイムの値は syTmrGetCount で参照します。初期化されてタイマがスタートすると 1.28 μ 秒毎に 1 カウントづつカウントダウンします。

syTmrGetCount はこの TMU0 の値を、0xFFFF:FFFF から減じた値を返しています。このため見かけ上では 0 からカウントアップするタイマになります。

また TMU0 の解像度は 1.28 μ 秒ですから、 $1.28 \mu \text{秒} \times 2^{32} = \text{約 } 1.5 \text{ 時間}$ で一周します。

フリーランタイム関連の関数として、syTmrDiffCount、syTmrCountToMicro、syTmrCountToMicro があります。

b. 経過時間を取得する (syTmrDiffCount)

syTmrDiffCount は、経過時間を取得するために使用します。処理内容は 2 つの与えられたフリーランタイムの値を単純に第 2 引数の値から第 1 引数の値を引いた結果を返しているだけです。

c. 引数のカウンタ値を変換する (syTmrCountToMicro)

syTmrCountToMicro は引数のカウンタ値を μ 秒に変換します。フリーランタイムの値から、実際の経過時間を取得する際に使用します。

与えられたカウンタ値を 1.28 倍するものですが、カウンタ値を 100 で割った商 (a) と余り (b) に分けて $a \times 128 + b \times 100 / 128$ と算出しています。

d. 指定した秒数が何カウントにあたるかを計算する

syTmrMicroToCount は、逆に指定した秒数 (μ 秒) が何カウントにあたるかを計算します。引数を 1.28 で割り算をします。syTmrCountToMicro と同様に秒数を 128 で割った商 (c) と余り (d) に分けて、 $c \times 100 + d \times 100 / 128$ と算出しています。

(2) 汎用タイマ関数群

汎用タイマは、SH4 の TMU2 を使用しています。この関数群は、TMU2 の資源をユーザー自身が、自由に使用できるような関数がサポートされています。汎用タイマ関連の関数のプレフィックスは syTmrGen です (syTmrGen の Gen は General の意)。

syTmrGenSetClock は汎用タイマの解像度を設定します。

参 照 設定できる解像度については、リファレンスマニュアルを参照して下さい。

この関数を呼び出す前は、SYD_TMR_CLOCK_P4 が設定されています。

syTmrGenSetCount は現在のタイマのカウント値を設定します。

syTmrGenStart は TMU2 をスタートさせます。この関数を呼び出す前は、TMU2 は停止しています。すでにスタートしている場合は何もしません。

syTmrGenStop は TMU2 を停止します。すでに停止している場合は何もしません。停止中はタイマのカウントの値は更新されません。

a. フリーランタイムとして使用する場合

フリーランタイムとして使用する場合は、syTmrGenSetClock、syTmrGenSetCount でタイマの解像度と初期値を設定した後、必要な際には syTmrGenGetCount で現在のタイマを参照するということになります。

時間への変換には、syTmrGenCountToMicro 関数などが利用できます。

すでに記述したように SH4 のタイマはダウンカウンタであるため、syTmrSetCount で指定した値から値が減じられます。汎用タイマは、このことを意識しない通常の値が増えていくタイマとして扱える仕様となっています。

このため、タイマの初期値に (ユーザー指定初期値) × (-1) を指定します。ここから値がマイナスされるため、n カウント経過後にはタイマの値は (ユーザー指定初期値 + n) × (-1) となります。

これをユーザーが値を取得する際に、符号を反転して返すことでユーザー指定初期値 + n が、あたかもアップカウンタであるような動作をさせます。

syTmrGenGetCount は汎用タイマのカウントを取得するものです。

syTmrGenCountToMicro 及び syTmrGenMicroToCount は、それぞれカウント数を μ 秒に変換したり、またその逆の変換をするものです。syTmrCountToMicro や syTmrMicroToCount との違いは、タイマの解像度を設定できる点のみです。

b. タイマ割り込み用に使用する場合

タイマ割り込みとして使用する場合は、syTmrGenSetClock、syTmrGenSetCount でタイマの解像度と初期値を設定し、syTmrGenSetInt で割り込みハンドラを設定します。syTmrGenStart でタイマをスタートさせて、指定カウントの経過した後に割り込みが発生し、ハンドラが実行されます。

SyTmrGenSetInt は、割り込みハンドラを登録する関数です。割り込みが発生する条件は、設定したタイマがオーバフローする場合であるということに注意して下さい。

「フリーランタイムとして使用する場合」のところでも記述したように、汎用タイマはアップカウンタタイマとして見せています。

タイマカウンタが 0xFFFF:FFF0 と指定した場合、16 カウント後に割り込みが発生します。すなわち 16 カウント後に割り込みを発生させたい場合は、0xFFFF:FFF0 (= -16) をカウンタに設定する必要があります。

タイマが稼動中でもエラーリターンすることなく、割り込みハンドラの登録が可能です。すでに別の割り込みハンドラが登録されていた場合は、元のハンドラは無効になります。

割り込みハンドラに NULL を指定した場合、syTmrGenSetInt の中で指定時間だけウェイトします。汎用タイマの場合はタイマのスタート・ストップが完全にユーザーに開放されていますが、syTmrGenSetInt で割り込みハンドラに NULL を指定した場合のみ、ライブラリ内部でタイマをスタートさせています。

3.6.3 タイマライブラリの初期化

タイマライブラリの初期化には、syTmrInit 関数を実行します。これは、syHwInit 関数のなかで呼ばれているため、標準的な sbinit.c を使用して初期化していればユーザーが明示的に行う必要はありません（してはいけません）。

初期化関数では、フリーランタイマの初期化を行います。具体的には、TMU0 の解像度を P/64（P は周辺クロックモジュール=50MHz、（50/64）MHz）、すなわち 1 カウント 1.28 μ 秒 に設定します。また TMU0 のタイマカウンタ及びこの再設定値を 0xFFFFFFFF に設定し、タイマをスタートします。

3.6.4 関数一覧

コールバックライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syTmrCountToMicro	フリーランニングタイマーのカウンタ値を μ 秒に変換
syTmrDiffCount	フリーランニングタイマーのカウンタ値の差の取得
syTmrGenCancelInt	汎用タイマーによる割り込みの禁止
syTmrGenCountToMicro	汎用カウンタ値を μ 秒値に変換
syTmrGenDiffCount	汎用割り込みカウンタの差の取得
syTmrGenGetCount	汎用タイマー割り込みカウンタを取得
syTmrGenMicroToCount	μ 秒値を汎用カウンタ値に変換
syTmrGenSetClock	汎用タイマーの基準クロックを設定
syTmrGenSetCount	汎用タイマーの割り込みカウンタの設定
syTmrGenSetInt	汎用タイマーの割り込み関数の登録
syTmrGenStart	汎用タイマーの起動
syTmrGenStop	汎用タイマーの停止
syTmrGetCount	フリーランニングタイマーの値の取得
syTmrMicroToCount	μ 秒値をフリーランニングタイマーのカウンタ値に変換

3.7 システムマネージャライブラリ (syMng)

システムマネージャライブラリは、次項目のファイバライブラリを使用して、5つのファイバを生成します。それぞれのファイバに関数を登録すると、登録された関数の一定のルールでスケジューリングします。

3.7.1 初期化

システムマネージャを初期化します。

システムマネージャは、以下の5つのファイバを生成します。メインファイバは、通常のプログラムが動作するファイバです。その他のファイバをサブファイバと呼びます。ミドルウェアファイバはミドルウェアライブラリが使用しますので、通常ではユーザーが使用することはありません。

- | | |
|------------------------|------------|
| (1) SYD_MNG_FLVL_TMR | タイマファイバ |
| (2) SYD_MNG_FLVL_UHIGH | 高優先度ファイバ |
| (3) SYD_MNG_FLVL_MAIN | メインファイバ |
| (4) SYD_MNG_FLVL_MW | ミドルウェアファイバ |
| (5) SYD_MNG_FLVL_UIDL | アイドルファイバ |

どのレベルのファイバが実行されていても、V-Sync 割り込みが発生すると強制的に状態を保持したまま停止し、トップレベルファイバに制御を移します。その後、上記のファイバが順番に実行されます。

各サブファイバは、登録されたサブファイバ関数を順次実行する永久ループです。すべての関数が実行されると、もう一度最初から実行されます。

メインを除く各レベルのファイバ毎にサブファイバ関数を8個まで登録できます。次のレベルに制御を移すタイミングは以下の通りです。

- a. syMngSleep 関数が実行された時
- b. すべてのサブファイバ関数が FALSE を返した時
- c. タイマファイバ実行中に最大実行時間を過ぎた時

注 意 サブファイバ関数のどれか1個でも TRUE を返した場合は、次のレベルには制御を移しませんので注意が必要です。

デフォルトでは、V-Sync 割り込みが発生するとメインファイバに実行を移します。タイマファイバと高優先度ファイバは実行されません。

これらのファイバを実行したい場合は、syMngSetSfbrTopLvl 関数によって、トップレベルファイバを変更する必要があります。

システムマネージャは、初期化時に kmSetWaitVsyncCallback を使用しています。Vsync 待ちコールバック関数を登録したい場合は、Kamui や Ninja の関数を使用せずに syMngEntryWaitVsyncCallback 関数を使用して下さい。

また、サブファイバ関数内から C 言語の標準関数を含むすべてのライブラリ関数を、使用することはできません。

3.7.2 関数一覧

システムマネージャライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syMngInit	システムマネージャの初期化
syMngFinish	システムマネージャの終了
syMngEntrySfbrFunc	サブファイバ関数の登録
syMngDeleteSfbrFunc	サブファイバ関数の削除
syMngSwitch	サブファイバへのスイッチ
syMngDelayedSwitch	割り込み処理内からのサブファイバへのスイッチ
syMngSleep	次のレベルのサブファイバへのスイッチ
syMngCleanUpSfbrFunc	サブファイバ関数のクリーンアップ
syMngEntryWaitVsyncCallback	待ちコールバック関数の登録
syMngGetCurFlvl	カレントファイバレベルの取得
syMngGetFbr	ファイバハンドルの取得
syMngSetSfbrTopLvl	トップレベルファイバの設定
syMngSetTmrDur	タイマファイバの実行時間の設定

3.8 ファイバライブラリ (syFbr) について

ファイバライブラリは、非常に単純なスレッド (=ファイバ) のライブラリです。

基本的に、ファイバを切り替えるためには、ユーザーがそのための関数を呼び出し実行する必要があります。

ノート 通常は、ファイバライブラリを直接使用せず、前の項目で説明するシステムマネージャライブラリを使用して下さい。

3.8.1 機能

ファイバの生成や、ファイバの切り替え及びファイバの削除等が、サポートされています。

(1) ファイバの生成

ファイバを生成する際のファイバ数の上限は特に定めてなく、初期化時にユーザーが指定します。

(2) ファイバの切り替え

切り替え先のファイバを指定して、syFbrSwitchToFiber () 関数を呼び出します。これにより、指定のファイバに切り替えが発生します。その場合の切り替え先は、そのファイバが以前に CPU を放棄したところ (syFbrSwitchFiber()関数を呼び出した次か、もしくは次に述べる syFbrDelayedSwitchToFiber()関数によってファイバが切りかえられた直後) になります。

割り込みによってファイバを切り替えたい場合は、割り込みハンドラ (通常は割り込みに対するコールバック関数) で、切り替えの syFbrDelayedSwitchToFiber () 関数を呼び出します。syFbrDelayedSwitchToFiber では、すべての割り込み (多重割り込みの場合は最初に発生した割り込み) ハンドラが終了し、割り込まれた箇所に戻る直前に指定のファイバに切り替わります。

これは、割り込みスタックの内容をファイバライブラリが直接操作 (割り込まれたところのプログラムカウンタを、ファイバライブラリ内部の関数に置きかえる) して実現しています。

(3) ファイバ切り替え時のコールバック

ファイバの切り替え時にユーザーが登録した関数が呼び出せるコールバック関数に、登録機能を追加しました。コールバック関数の呼び出されるきっかけは、現在のファイバのコンテキストを保存後、次のファイバのコンテキストの復帰前です。コールバック関数を使用するスタックは、現在のファイバのスタックです。コールバック関数内での処理は極めて制限されます。フラグの更新を行うなどの処理にとどめて下さい。

(4) コンテキスト

ファイバライブラリが保存・復帰するコンテキスト (レジスタ) は、以下のとおりです。

- 全ての整数レジスタ (バンクレジスタ含む)
- 全ての浮動小数点レジスタ (バンクレジスタ含む)
- GBR、MACH、MACL、PR、FPSCR、FPUR、SR、PC
- ストアキュー (SQ) の内容

ノート コンテキストは、ファイバコンテキスト領域に保存されます。

ファイバのコンテキストは、以下の領域にとられます。

a. メインファイバ

メインファイバは、syFbrConvertThreadToFiber によって変換したファイバです。

このコンテキストは、以下のアドレス (0x8c00f000 ~ 0x8c00f3ff (1K バイト)) に固定的に配置されます (「図 1-2 IP 領域」を参照して下さい)。

b. その他の (syFbrCreateFiber によって生成した) ファイバ

その他のファイバのコンテキストは、各ファイバスタップの底 1K Byte の領域 (32 バイトに境界を調整後) に置かれます。

ファイバ作成時のパラメータと、メモリの関係は次のようになります。

- SyFbrCreateFiber (lpStack、dwStackSize、lpFunc、lpParam);

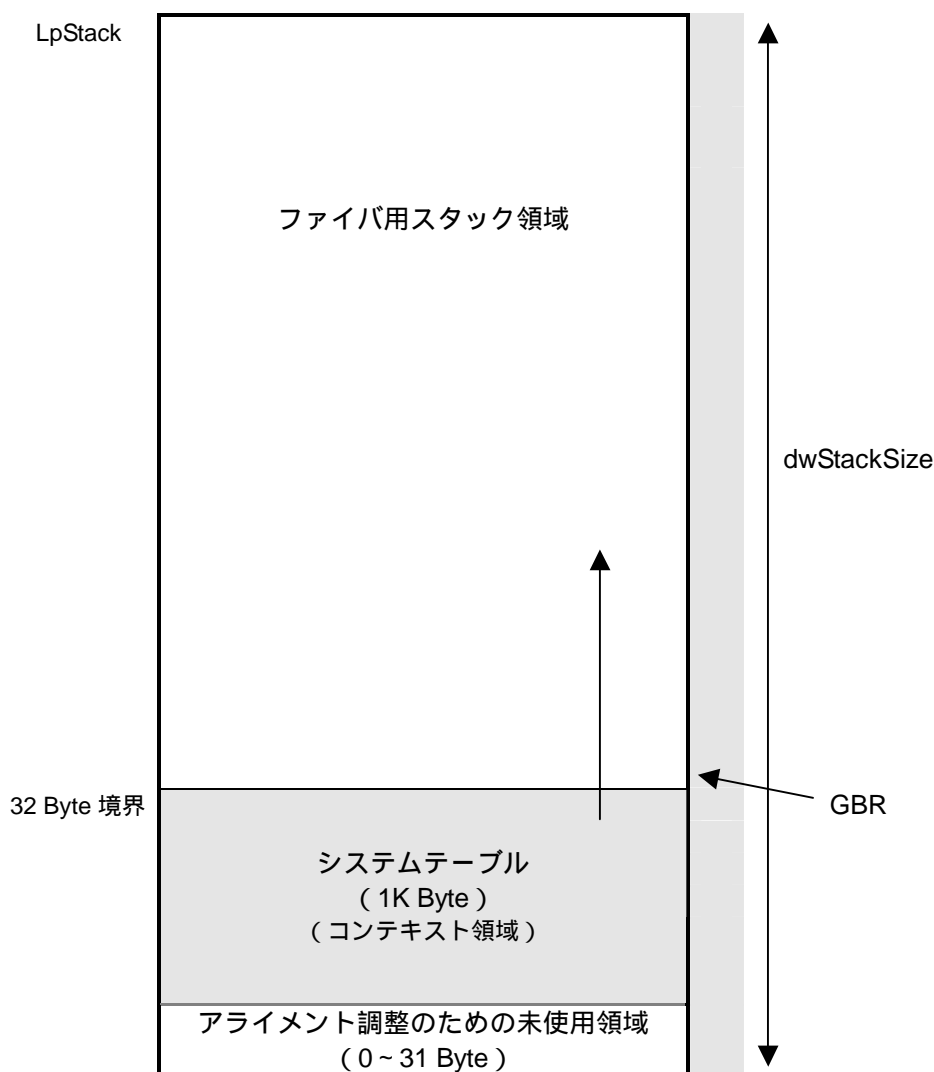


図 3 - 1 ファイバのコンテキストマップ

(5) GBR レジスタについて

Shinobi2 から、SH4 の GBR レジスタをライブラリで予約しました。GBR レジスタは、常にカレントファイバのコンテキストエリアをポイントするように設定されます。GBR レジスタは、以下の関数で更新されます。

- **syFbrInit**、**syFbrInitEx**

GBR を初期設定 (0x8c00f000) します。

- **syFbrSwitchToFiber**

切り替え先のファイバのシステムテーブルのアドレスに設定します。

- **syFbrDelayedSwitchToFiber**

切り替え先のファイバシステムテーブルのアドレスに設定します。ただし関数コール直後は変更されず、ファイバの切り替え処理後に変更します。

3.8.2 関数一覧

ファイバライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syFbrInit	ファイバライブラリの初期化
syFbrInitEx	ファイバライブラリの拡張初期化
syFbrExit	ファイバライブラリの終了
syFbrCreateFiber	ファイバの生成
syFbrDeleteFiber	ファイバの削除
syFbrSwitchToFiber	ファイバの切り替え
syFbrDelayedSwitchToFiber	割り込みを契機としたファイバの切り替え
syFbrConvertThreadToFiber	現在実行中の関数をファイバ (メインファイバ) に変換
syFbrGetCurrentFiber	実行中のファイバハンドルを取得
syFbrGetFiberData	実行中のファイバのファイバデータを取得
syFbrGetMainFiber	メインファイバのハンドルを取得
syFbrSetCallback	ファイバ切り替え時に呼び出されるコールバック関数を登録

4 BootROM 関連のライブラリ

ここでは、Dreamcast 専用のアプリケーションディスクを判別するライブラリ等について説明します。

4.1 BootROM サービスライブラリ (syBt) について

このライブラリでは、BootROM メニューに戻る機能、及びディスク交換時に利用するディスクの種別を判定する機能、ディスクの ID を取得する機能を提供します。

4.1.1 関数一覧

BootROM サービスライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syBtGetCurrentSystemID	ディスクのシステム ID 情報の取得
syBtGetBootSystemID	起動ディスクのシステム ID 情報の取得
syBtCheckDisc	ディスクの種別を判定
syBtExit	BootROM のメインメニューを表示

4.2 BootROM フォントライブラリ (syBtfont) について

BootROM フォントライブラリは、ブートルームにビットイメージで格納されているフォントデータを取得するためのライブラリです。

4.2.1 アクセスチェック

BootROM へのアクセスはバスの仕様上、GD-ROM のアクセスと重なると正常な値を取得できません。この競合を避けるため、BootROM へのアクセスのセマフォを確かめる必要があります (syBtFontChkSmph、syBtFontClrSmph)。

4.2.2 BootROM に含まれる文字データ

BootROM に含まれている文字のデータを下表に示します。

表 4- 1 BootROM に含まれる文字データ

分 類	コード	サイズ (横/縦)	1文字の バイト数	オフセット	サイズ
ASCII (*)	0x20 - 0x7e	12/24	36	0x00000000	0x00000d80
ISO 8859-1	0xa0 - 0xff	12/24	36	0x00000d80	0x00000d80
JIS X 0201	0xa0 - 0xff	12/24	36	0x00001b00	0x00000d80
Shift JIS part1	0x8140-0x8396	24/24	72	0x00002880	0x0000b910
Shift JIS part2	0x889f-0x9872	24/24	72	0x0000e190	0x00034E00
Shift JIS part3	0x989f-0xeaa4	24/24	72	0x00042f90	0x0003B970
VMS icons	No.0 - 128	32/32	128	0x00042f90	0x0007EF30

0x20 には、JIS X での 0x7e であるオーバーライン、0x7f には JIS X での 0x5c である円マークのフォントデータが入っています。0x20 はスペース、0x7f は非表示文字の文字コードです。

4.2.3 BootROM のフォントデータの格納例

フォントはビットイメージで BootROM 内に格納されています。

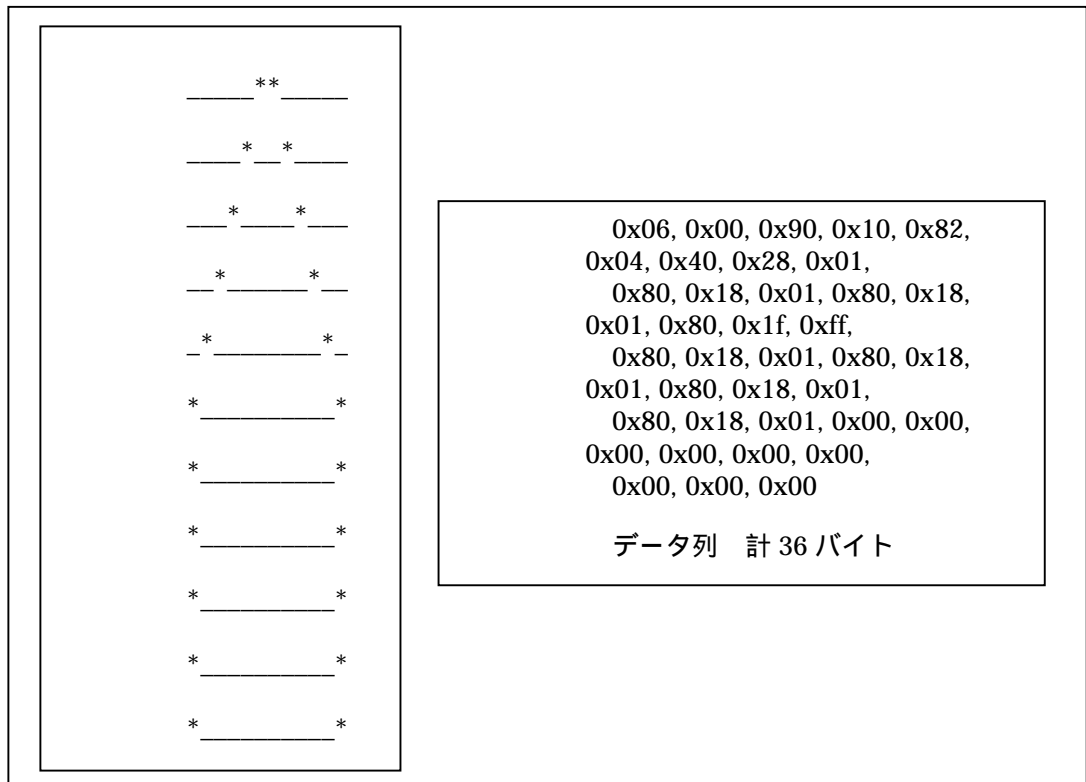


図 4 - 1 フォントデータ格納イメージ

(1) サンプルプログラム

次のリストは、BootROM から 1 文字ワークエリアにフォントイメージをコピーするサンプルです。

リスト 4 - 1 BootROM フォントサンプルプログラム

```
#define SMD_WRK_ADDR (0x8c020000)

sample ( )
{
    Uint32 smFntAddr, smTheFntAddr;
    Void *smWrkAddr;
    Sint16 smTheCode;

    SmFntAddr = (Uint32) syBtFntGet ( ) ;
    SmWrkAddr = (void *) SMD_WRK_ADDR;
    SmTheCode = 0x88a4; /* 愛 */

    /* evaluate the font address in Boot ROM */
    smTheFntAddr = smFntAddr;
    smTheFntAddr += syBtFntGetAddr ( SYE_BT_FNT_FONTSET_JP_LVL2_24, smTheCode ) ;

    /* font semaphore check */
    if ( syBtFntChkSmph ( ) == SYD_BT_FNT_SMPH_SUCCESS ) {
        /* copy 1 letter to work RAM from Boot ROM */
        memcpy ( (void *) smTheFntAddr, smWrkAddr, SYD_BT_FNT_24_SIZE_ZEN ) ;
        /* font semaphore clear */
        syBtFntClrSmph ( ) ;
    }
}
```

4.2.4 関数一覧

BootROM フォントライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syBtFntChkSmph	フォントセマフォのチェック
syBtFntClrSmph	フォントセマフォのクリア
syBtFntGet	フォントアドレスの取得
syBtFntGetAddr	フォント格納位置の取得
syBtFntGetInfo	フォントサイズの取得
syBtFntSjis2Jis	ShiftJIS コードから JIS コードへ変換

5 GD-ROM 関連のライブラリ

ここでは、Dreamcast 独自のディスク規格・GD-ROM に関連したライブラリの説明についてや GD-ROM の設定などに関して、知っておくと良い点や質問の多い点について説明します。

5.1 GD ファイルシステムライブラリ (gdFs) について

GD-ROM のファイルシステムを提供するためのライブラリです。また、GDDA に関する機能も提供します。

5.1.1 GD-ROM の基本事項

(1) GD-ROM フォーマットについて

Dreamcast の CD-ROM は GD-ROM (Giga byte Disk ROM) と呼ばれ、独自のフォーマットシステムを採用しています。

GD-ROM フォーマットのディスク構造は、高密と単密のエリアと IP 領域のエリアの3つのブロックに分かれています。

(2) GD-ROM フォーマットの特徴 (CD-ROM との比較)

回転速度が一定の機構で、同じに読み取る時間でも、データの読み取り部分 (レコードの針に相当する部分) が外周にあるほど、大量のデータを読み取ることができます。(通常の CD-ROM ドライブは線速度一定という形式を使用しているため、読み取り部分がどこにあっても同じ時間内に読みとれるデータの量は常に一定です。その代わりに、読み取り部分の位置によって回転の速度が変化します。実際に、ディスクドライブに相当する部分に耳を当てれば、ディスクの回転音が違うのが分かるでしょう)。つまり、一度に大量のデータを読み取る必要があるときは、そのデータを GD-ROM の外側に置くほうが高速に読み取れます。

上記のような理由で、Dreamcast で採用している CD-ROM は 4 倍 ~ 12 倍速と速度が変化します。最近の CD-ROM ドライブでは、“12 ~ 24 倍速” といった表記のものがありますが、これらはすべて同じ方式を採用した CD-ROM ドライブです。

5.1.2 ファイル構造について

従来の CD-ROM ドライブでもアクセスが可能な単密エリアが内周に置かれ、専用ドライブでのみアクセスが可能な高密エリアが、外周に配置されています。

上記のように GD-ROM のディスクは、構造上2つのブロックに分かれています。

(1) 単密エリア

一番内側のエリアを単密エリアと呼び、ここには一般的な CD-ROM の規格に従った形式でデータや CDDA などが収められます。単密エリアの内容については、CD プレイヤーやコンピュータでこのエリアを参照することが可能です。

ただし、このエリアは4分間程度しかないので、例えば「このディスクはセガ Dreamcast 用のゲームディスクです。ゲームのデータが入っていますので再生しないで下さい。」などといった、表示用のメッセージを入れる場合などが想定されています。

また、Dreamcast 用のソフトからこの領域にアクセスすることはできません。

(2) 高密エリア

外側のエリアは高密エリアで、ここにゲームのデータやプログラム (ゲームで使用する) CD-DA のデータなどを収納することができます。

5.1.3 ファイルの配置

ファイルが、ディスク上にどのように配置されるかを知っておくことで、ディスクアクセスを最適化し処理を高速化することができます。

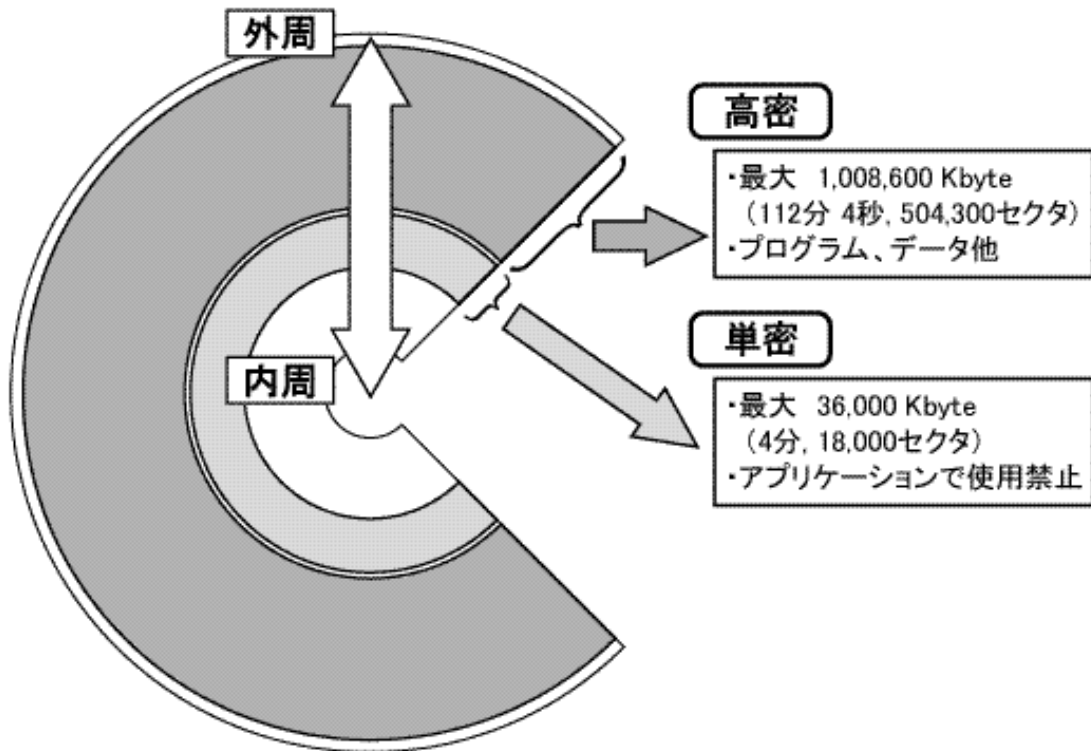


図 5 - 1 GD-ROM のディスク構造

a. GD-ROM フォーマットのデータの配置

アプリケーションのプログラム及びデータは高密エリアに配置され、単密エリアのデータはアプリケーション側から読むことができません。

標準の設定で GD-ROM を作成した場合は、ディレクトリ情報は内周にデータなどのファイルは外周に配置されます。

プログラム及びデータを読み込んだ後でディレクトリを読み込む (gdFsLoadDir、gdFsChangeDir など) と、GD のヘッドが外周から内周・内周から外周へと移動するため、約 1 秒程度の時間が掛かります (このうちディレクトリアクセスの時間は微々たるもので、時間のほとんどはシークの時間です)。

これを回避するためには、あらかじめアクセスするディレクトリを gdFsLoadDir 関数で先読みしておくといでしょう。

注意 頻繁にデータ (及びプログラム) とディレクトリを交互に読み込むと、シーク時間が増大し遅くなるだけでなく、ドライブの寿命を短くすることになりますので注意して下さい。

また、同一シーンで必要となるファイルを同じディレクトリにまとめておけば、頻繁にデータ領域とディレクトリ情報の領域をシークすることはなくなります。

ファイルの配置は、CRI CD-Craft のスクリプトファイルで記述された順に並びます。連続して読むファイルが、シーケンシャルに並んでいるほうが当然アクセスは、高速になります。

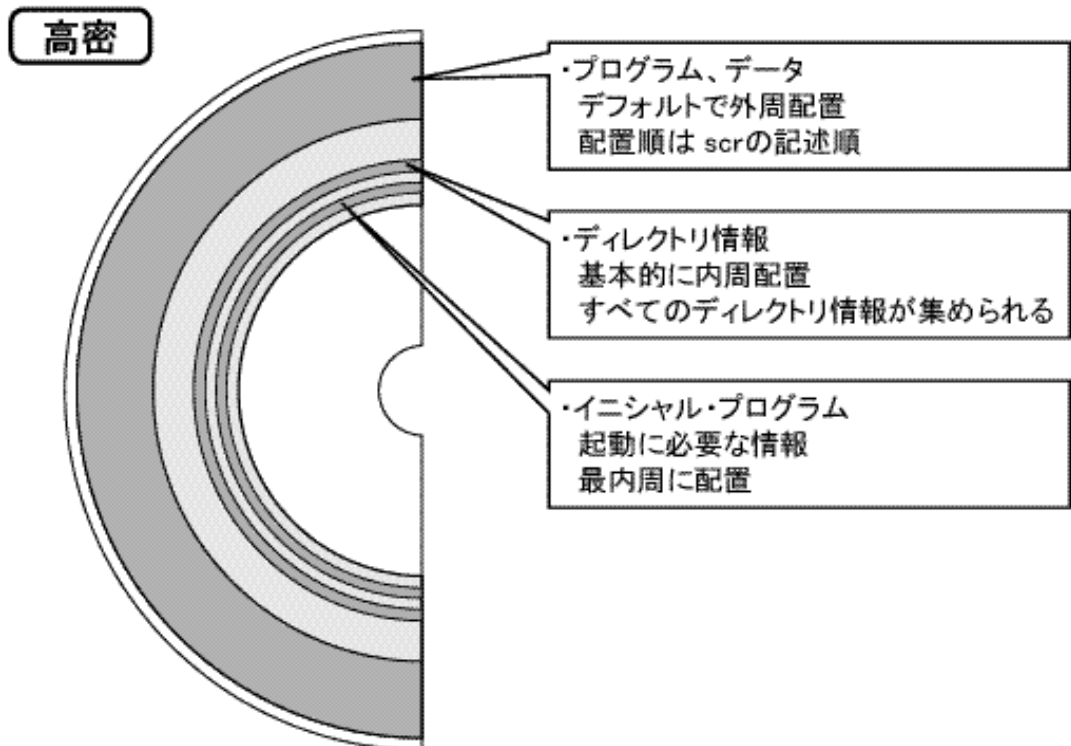


図5-2 高密エリアのデータ配置

5.1.4 ディスクドアオープンの検出

特に既定する例外を除いて、『本体の「オープンボタン」が押されディスクドアが開かれた場合、本体 BootROM のメニューへ進む。』規定になっています。ここでは、この実装例を示します。

(1) 実装の方針

ディスクドアのオープンは、ドライブのステータスの取得結果か、GD ドライブのエラーとしてエラーコールバックに通知されます。エラーコールバック中では、ライブラリの呼び出しに制限があります。このため、ドアオープンの検出のみコールバックで行い、ドアオープンに対する処理はメイン側で行います。

(2) 実装例

上記の方針に従った実装例を示します。

a. 初期化時

- ドアオープンのフラグをクリアします。
- GDFS のエラーコールバック関数を登録します。

b. エラーコールバックでの処理

- エラーの種別がディスクドアのオープン (GDD_ERR_TRAYOPEN) かディスクの交換を検出した (GDD_ERR_UNITATTENT) というものであれば、ドアオープンと判定し、フラグを設定します。

c. 毎フレームの処理

- ドライブのステータスを取得します。その結果がオープン（GDD_DRVSTAT_OPEN）の場合、ドアオープンと判定しフラグセットします。
ビジー（GDD_DRVSTAT_BUSY）の場合、これはドライブのステータス遷移中であるため、ドライブにダミーコマンドを送信してステータスを更新させます。
- ドアオープンのフラグをチェックし、セットされている場合は、使用中のライブラリの終了関数を呼び出し、BootROM にジャンプ（syBtExit（）関数）します。ただし、バックアップメモリの保存中などの場合は、必要な処理の完了を待ってから BootROM のメニューに戻ります。

リスト 5 - 1 ディスクドアオープン実装例（部分）

```
BEGIN ----- Part.1
Static Sint32 trayopen_flag; /* ドアオープンフラグ 0:close 1:Open */

Void errfunc(void *obj, Sint32 err)
{
    /* if the tray was opened ? */
    if (err == GDD_ERR_TRAYOPEND || err == GDD_ERR_UNITATTENT) {
        /* set the trayopen flag */
        trayopen_flag = 1;
    }
}
END ----- Part.1

[スタティック変数の初期化、コールバック関数の登録]

BEGIN ----- Part.2
    /* clear the abort flag */
    Abort = 0;

    /* set an error callback */
    gdFsEntryErrFuncAll(errfunc, (void *) 0);
END ----- Part.2

[メインループ処理]

BEGIN ----- Part.3
    /* check the drive status */
    dstat = gdFsGetDrvStat();
    if (dstat == GDD_DRVSTAT_OPEN) {
        trayopen_flag = 1;
    } else if (dstat == GDD_DRVSTAT_BUSY) {
        gdFsReqDrvStat();
    }

    /* if the trayopen flag is set */
    if (trayopen_flag) {
        /* trayopen 時の処理 */
    }
}
END ----- Part.3
```

(3) GD エミュレータ (GD-Workshop) でのデバッグ時の注意

GD エミュレータと実ドライブでは、以下のような仕様の違いがあります。

表 5 - 1 ドライブとエミュレータの仕様の違い

Dreamcast 本体	ドライブの状態は常に更新される。
Dev.Box (実ドライブ)	gdFsGetDrvStat は現在のドライブ状態を返す。
Dev.Box (エミュレータ)	ドライブの状態はドライブに処理を要求し、それが完了したときのみ更新される。 gdFsGetDrvStat は、最新のコマンド発行時の状態を返す。 GD エミュレータのドライブの状態を更新させるためには、ステータス更新用のコマンドを発行しなくてはならない。

エミュレータ使用時にドアオープンを行っても、先の例の gdFsGetDrvStat では、現在の状態を取得することはできません。

簡単な対処法として、定期的にコマンドを発行することでこの問題を回避できます。しかし、実ドライブでは意味がなくなりパフォーマンスの低下となりますので、開発のテスト目的での使用にのみ留めておくべきでしょう。

リスト 5 - 2 エミュレータ使用時の差分リスト

```
(実ドライブでは、この処理は不要です)
BEGIN ----- Patch (メインループに追加)
{
    static Sint32 LoopCount = 0;

    if ( ( (LoopCount++) % 60 ) == 0 ) {
        gdFsReqDrvStat();
    }
}
END ----- Patch
```

5.1.5 ディスクの交換の検出時について

ドライブ側がメディアの交換を検出したときには、GD のアクセスに対して GDD_ERR_UNITATTENT のエラーが発生します。

ディスク交換に未対応の場面では、本エラーに対してディスクドアオープンと同様の対処を行って下さい。

(1) ディスクの交換処理

ディスク交換の場面では、下記の手順に従ってディスクの交換処理を行って下さい。

a. メディアの認識処理 (syBtCheckDisc)

GDD_ERR_UNITATTENT を検出した状態では、ドライブに対してのすべてのアクセスが無効となります。これを解除するには、メディアの認識処理が必要になります。

メディアの認識処理には、syBtCheckDisc 関数を使用します。syBtCheckDisc 関数の呼び出しは 1 V に 1 回以下にして下さい。syBtCheckDisc 関数はコマンド投入型の関数で、syBtCheckDisc 関数のみを実行しループさせても処理は進みません。

メディアの認識処理に失敗した場合は、再度ディスクの入れ替えを要求します。

- b. システム ID のチェック (syBtGetBootSystemID、syBtGetCurrentSystemID)

メディア認識の正常終了後、ディスクのシステム ID をチェックして下さい。システム ID の取得には、syBtGetBootSystemID、syBtGetCurrentSystemID 関数を使用します。システム ID として返される製品番号と現在のディスクが何枚目かをチェックすることにより、目的のディスクに交換がされたかを判定します。

注 意 ディスクの認識チェックがされていない場合に、syBtGetCurrentSystemID が返すシステム ID には、不定な値が入ることになりますのでご注意ください。

- c. GDFS の再初期化 (gdFsReinit、gdFsFinish + gdFsInit)

目的のディスクであることが確認した場合は、gdFsReinit 関数を実行して GD のファイルシステムを初期化して下さい。もしくは、gdFsFinish 関数を実行後に gdFsInit 関数を実行して、初期化して下さい。

GD ファイルシステムは再初期化をしないと、正しくディスク情報を読み取ることができません。

再初期化した後は、先読みされたディレクトリ情報やオープン中のファイルハンドルは、すべて無効となります。またカレントディレクトリは、ルートディレクトリに変更されます。

5.1.6 エラー発生時のリトライについて

GDFS が返すエラーには、致命的なものと回復が可能なものがあります。ドライブからのファイルを読み込む失敗には回復可能なものがあり、この場合は有限回数のリトライを推奨しています。

(1) エラー定数の詳細

エラー定数の詳細を示します。エラーの種別については、以下の通りです。

N：正常、R：復帰可能、F：致命的、C：エラーコールバック発生、D：ドライブから発行

基本的にエラーコールバックが発生するのは、ドライブのアクセスするコマンドを発行した場合のみです。

表 5 - 2 GDFS のエラー定数の詳細

エラー定数	種 別	詳 細
GDD_ERR_OK	N	正常終了
GDD_ERR_INIT	F	ライブラリ初期化失敗
GDD_ERR_RESET	F	ドライブ初期化失敗 (回復可能な場合あり)
GDD_ERR_LIBOV	F	オーバーレイ初期化失敗
GDD_ERR_MOUNT	F	PVD や Root Directory を読み込み失敗
GDD_ERR_DISC	F	利用可能なメディアではない (CD-ROM、音楽 CD、不正な GD が挿入された場合など) 場合
GDD_ERR_DIRREC	F	ディレクトリレコード不正 (現在発生しない)
GDD_ERR_CANTOPEN	F	ファイルオープン失敗 (現在発生しない)
GDD_ERR_NOTFOUND	F	ファイルが見つからない
GDD_ERR_NOHNDL	F	空いているハンドルがない (現在発生しない)
GDD_ERR_ILLHNDL	F	不正なハンドルを使用した
GDD_ERR_NOTDIR	F	ディレクトリでないものを指定した
GDD_ERR_DIROVER	F	ディレクトリエントリ最大数をオーバーした
GDD_ERR_BUSY	R	他のコマンドを実行中。ドライブのステータスの遷移中
GDD_ERR_32ALIGN	F	32 バイトアラインでないアドレスを指定した
GDD_ERR_SIZE	F	転送サイズ指定が正しくない
GDD_ERR_SEEK	F	シークモード指定が正しくない
GDD_ERR_OFS	F	位置指定が正しくない

エラー定数	種 別	詳 細
GDD_ERR_ILLTMODE	F	転送モード指定が正しくない
GDD_ERR_READ	R	読み込みエラー
GDD_ERR_NOTREAD	F	読み込み中ではない
GDD_ERR_TOUT	F	ドライブから応答がない(20 秒間応答が無い)
GDD_ERR_EOF	F	ファイル終端を検出した(現在発生しない)
GDD_ERR_TRAYOPEND	FC	ディスクドアがオープンされている
GDD_ERR_SIZEOVER	F	転送サイズ指定が不正
GDD_ERR_FATAL	F	致命的エラー(現在発生しない)
GDD_ERR_UNDEF	F	未定義エラー(現在発生しない)
GDD_ERR_NOERR	NCD	報告すべきエラーはない
GDD_ERR_RECOVER	NCD	エラーを回復したことを検知した (基本的に正常状態に回復できている)
GDD_ERR_NOTREADY	FRCD	メディアの認識中(R)や、メディアが未装着(F)の ときに発生
GDD_ERR_MEDIA	FCD (R)	リトライできなかつたり、修正不能なエラーが発生。 主にメディアの傷等が原因(回復可能な場合あり)
GDD_ERR_HWARE	FCD (R)	ハードウェアの劣化等のため正常に動作できなかった場 合(回復可能な場合あり)
GDD_ERR_ILLREQ	FCD (R)	不正なコマンドを発行したり、ドライブが予期していな い状況を検知した(回復可能な場合あり)
GDD_ERR_UNITATTENT	FCD	メディアの交換を検出した
GDD_ERR_PROTECT	FCD (R)	汚れ等のために動作できない状態にあることを検知した
GDD_ERR_ABORT	FCD	中断を検知した
GDD_ERR_NOREADABLE	FC	読み込みできないメディア(GD-ROM 以外、正常なメ ディアでない等)である
GDD_ERR_CHECKBUSY	RC	メディア認識中のため BUSY である(基本的にはこ のエラーの後には GDD_ERR_UNITATTENT が発生 する)

(2) エラーのリトライ

GD からディスクをディスクからのファイルロードに失敗した場合は、有限回数リトライ処理（再びファイルを読み込みに行く）を行うことを推奨しています。

リード要求後のエラーステータスを取得し、読み込みが失敗した場合にはリトライを行います。具体的には、読み込みリクエスト後に GD のハンドルのステータスがエラーの場合、そのエラーの種別によりリトライを行うかどうかを決定します。

リスト 5 - 3 GDFS リトライの例（部分）

```
Sint32 nsct = 32;
GDFS gf;
Uint32 buf[0x4000];
Sint32 stat;
Sint32 errstat;
Sint32 retrycount;
Sint32 retrymax = 8;

gf = gdFsOpen("TEST.BIN", NULL);
if (gf == NULL) goto fatal;

retrycount = 0;

retry:
    gdFsReqRd32(gf, nsct, buf);

wait_status:
    stat = gdFsGetStat(gf);

    /* GDD_STAT_READ の間、時間を計測し、 */
    /* タイムアウト処理をいれることを推奨 */

    switch (stat) {
    case GDD_STAT_READ:
        goto wait_status;

    case GDD_STAT_ERR:
        errstat = gdFsGetErrStat(gf);
        switch (errstat) {
        case GDD_ERR_NOERR:
        case GDD_ERR_RECOVER:
            /* 通常これらは発生しない */
            /* break; */

        case GDD_ERR_NOTREADY:
        case GDD_ERR_MEDIA:
        case GDD_ERR_HWARE:
        case GDD_ERR_ILLREQ:
        case GDD_ERR_PROTECT:
        case GDD_ERR_ABORT:
        case GDD_ERR_CHECKBUSY:
            retrycount++;
            if (retrycount > retrymax) goto fatal;

            goto retry;

        case GDD_ERR_UNITATTENT:
        case GDD_ERR_TRAYOPEND:
            goto trayopen;

        case GDD_ERR_NOREADABLE:
        default:
            goto fatal;
        }
        break;
    case GDD_STAT_COMPLETE:
```



```

        break;

    case GDD_STAT_IDLE:
    default:
        goto fatal;
    }
    gdFsClose(gf);
    /* 読み込み完了, 次の処理へ */
    goto next_read;
fatal: /* 致命的エラー処理 */
trayopen: /* ディスクドアオープン処理 */

```

5.1.7 GDFS の内部動作について

GD-ROM ドライブや GDFS ライブラリ内部の動作について紹介します。

(1) GD ドライブ内のキャッシュ構造

GD-ROM ドライブは、内部に 128KB のキャッシュ用のメモリを持っています。

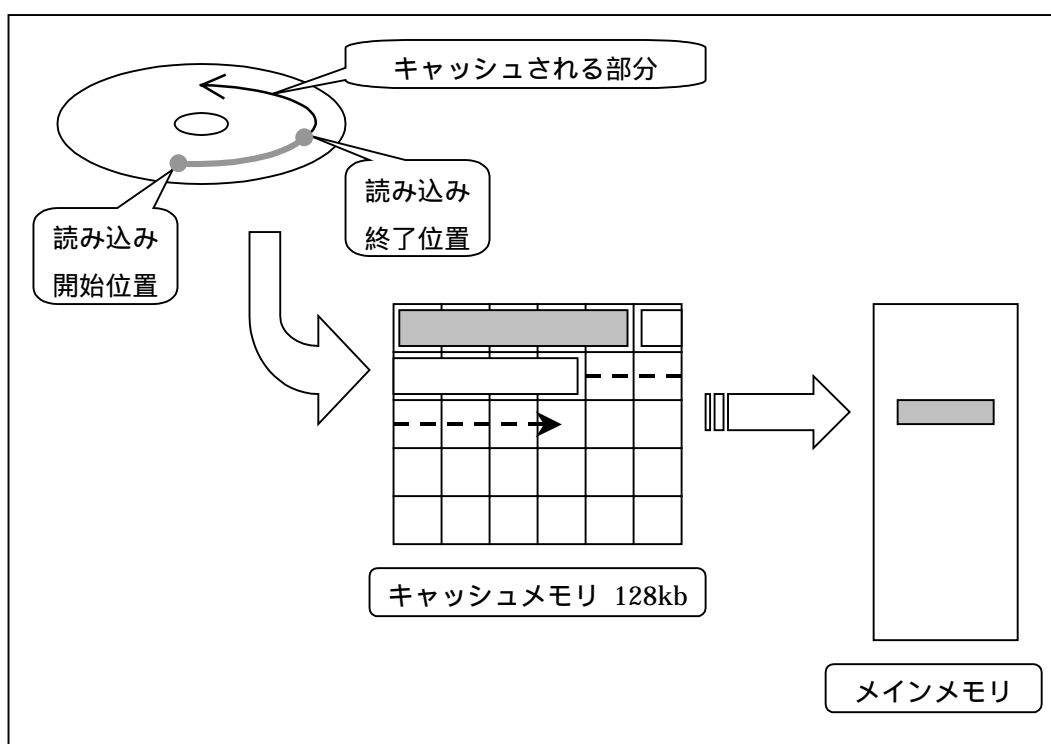


図 5 - 3 GD-ROM ドライブ内のキャッシング

指定のファイルの読み込み後も GD ドライブのヘッドは、続いているセクタを読み続けてキャッシュメモリに保存します。キャッシュメモリに空きがなくなったとき読み込みを終了し、ポーズ状態となります。

続くファイルを読み込むときに、キャッシュメモリ上にデータが存在していれば、GD-ROM にアクセスすることなく、キャッシュメモリから高速にファイルを転送することができます。

キャッシュされているセクタよりも先にシークした場合は、それ以前のキャッシュされたデータは捨てられます（ヘッド位置よりも前のセクタのキャッシュは破棄されるということ）。

ノート このキャッシュアルゴリズムは、連続でファイルを少量ずつ読み込む場合などに有利に作用します。ただし、このキャッシュアルゴリズムは、将来ハードの改良により変更される可能性がありますので、参考程度にとどめておいて下さい。

(2) GDFS のコールバックの発生タイミング

a. 読み込み終了コールバック (gdFsReqRd32)

即時復帰関数 (gdFsReqRd32 等) の実処理は、VBlank 割り込み内で行われています。読み込み終了コールバック、エラーコールバックはその VBlank 割り込み内で呼び出されます。これらのコールバック処理中は、VBlank 割り込みは禁止状態です。このためこのコールバック内では、実行できないライブラリ関数があります。

b. 転送終了コールバック

転送終了コールバックは、転送 DMA が終了した時点で呼び出されます。このコールバックでは VBlank 割り込みの禁止はされませんが、GD のアクセスにロックがかかった状態のため、いくつかの GDFS 関数 (ドライブにアクセスする関数) の呼び出しが制限されます。原則としてドライブアクセス関数は、gdFsTrans32 関数及び gdFsStopRd 関数のみ使用可能です。

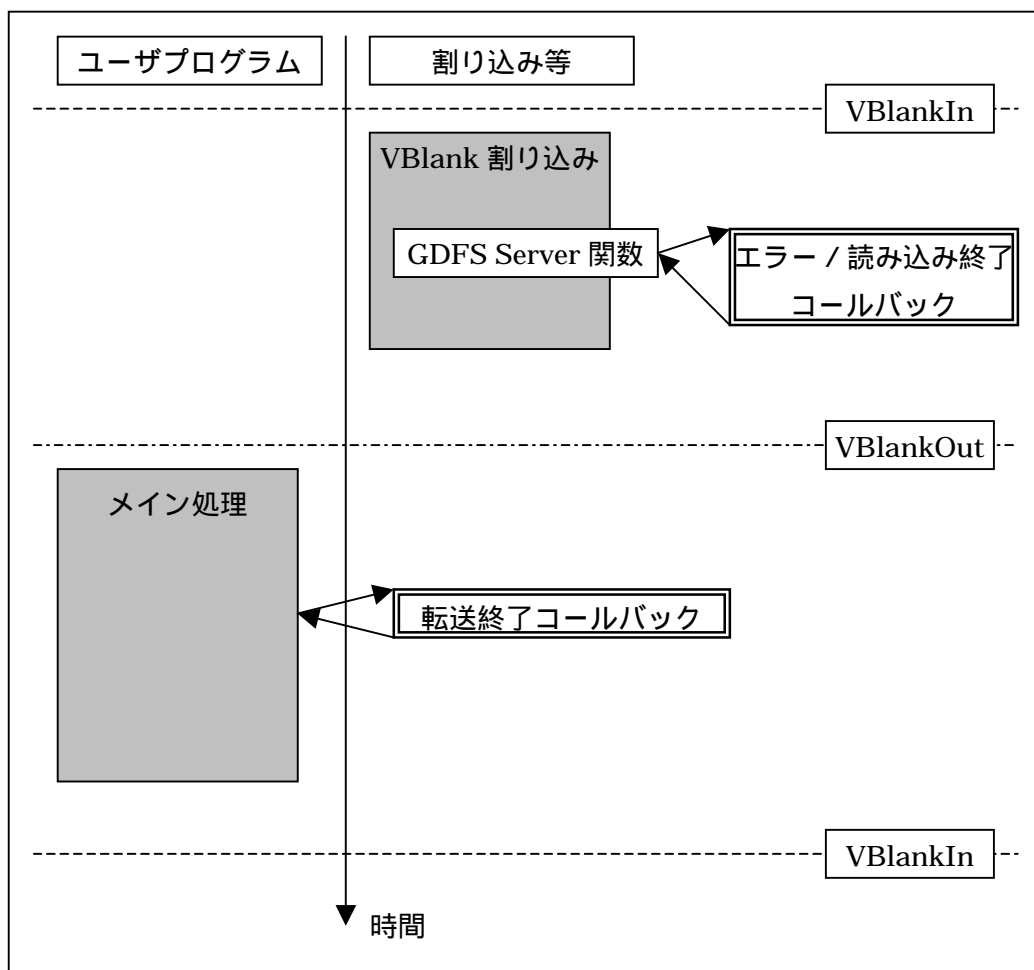


図 5 - 4 GDFS ライブラリのコールバックの発生タイミング

5.1.8 その他 & 注意点

● GD アクセス部分の実装

GDFS 関連のエラー処理はトラブルになることが多いので、(Ninja2 ライブラリの njLoadTexture 系のファイルをアクセスする関数も含めて) できる限り 1 箇所にまとめて置いて下さい。ライブラリの関数を直接呼ぶ代わりに、ユーザー側の関数でラップした関数を使用しておくと、後でエラー処理の追加やリトライ処理を追加することが容易になります。

5.1.9 関数一覧

GD ファイルシステムライブラリでは、以下の関数がサポートされています。
 詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
gdFsInit2	GD ファイルシステムの初期化
gdFsFinish	GD ファイルシステムの終了
gdFsCalcSctSize	バイト数からセクタ数を算出
gdFsChangeDir	カレントディレクトリの変更
gdFsCheckEof	ファイルの終端のチェック
gdFsClose	ファイルのクローズ
gdFsCreateDirhnh	ディレクトリハンドルの生成
gdFsDaGetInfo	DA の再生情報の取得
gdFsDaPause	DA 再生の一時停止
gdFsDaPlay	DA 再生の開始（トラック指定）
gdFsDaPlaySct	DA 再生の開始（セクタ指定）
gdFsDaRelease	DA 再生の一時停止の解除
gdFsDaStop	DA 再生の停止
gdFsEntryErrFunc	エラー発生時のコールバック関数の登録
gdFsEntryErrFuncAll	全ハンドルでのエラー発生時のコールバック関数の登録
gdFsEntryRdEndFunc	読み込み完了時のコールバック関数の登録
gdFsEntryTrEndFunc	転送完了時のコールバック関数の登録
gdFsExecServer	GD ファイルシステムのサーバ関数
gdFsGetDirInfo	ファイル情報の取得
gdFsGetDirrecSize	ディレクトリバッファのバイト数の取得
gdFsGetDrvStat	ドライブ状態の取得
gdFsGetErrStat	ハンドルのエラー状態の取得
gdFsGetFileFad	ファイルの FAD の取得
gdFsGetFileSctSize	ファイルのセクタサイズの取得
gdFsGetFileSize	ファイルサイズの取得
gdFsGetNumRd	読み込みバイト数の取得
gdFsGetStat	ハンドルの状態の取得
gdFsGetSysHn	システムハンドルの取得
gdFsGetToc	TOC の取得
gdFsGetTransStat	転送状態の取得
gdFsGetWorkHn	動作中のハンドルの取得
gdFsGetWorkSize	ライブラリのワークエリアバイト数の取得
gdFsIsTrayOpen	ディスクドアの開閉状態を取得
gdFsLoadDir	ディレクトリレコードの読み込み
gdFsMount	GD ルートディレクトリのマウント
gdFsMovePickup	ピックアップを次の位置に移動
gdFsOpen	ファイルのオープン
gdFsOpenRange	セクタ指定によるファイルオープン
gdFsRead	ファイルの読み込み
gdFsReMount	GD ファイルシステムの再初期化
gdFsReqDrvStat	ドライブ状態の更新
gdFsReqGdRd	GD バッファへの読み込みリクエスト
gdFsReqRd32	ファイルの読み込みリクエスト
gdFsSeek	ファイルのシーク

関数名	機 能
gdFsSetDir	カレントディレクトリの設定
gdFsStopRd	リクエストの中断
gdFsTell	ファイルの読み込みポインタの位置の取得
gdFsTrans32	GD バッファからの転送指定

6

ペリフェラル関連のライブラリ

ここでは、Dreamcast の一連の周辺機器についての基本的な説明や情報を説明します。

参照 各ペリフェラルの固有情報については、章末の補足資料「12.1 各ペリフェラル毎の固有情報」を、参照して下さい。

6.1 ペリフェラルの概要

ペリフェラルとは、Dreamcast の一連の周辺機器のことを総称して言います。

ここでは、ペリフェラルに関して知っておくと良い点や基本項目について、簡単に情報をまとめて説明します。

6.1.1 ペリフェラルの分類について

ペリフェラルの分類については、以下の5種類に大別されます。

(1) コントローラペリフェラル

Dreamcast 本体のコントロールポート (A~D) に、挿し込むタイプのペリフェラル群を指します。

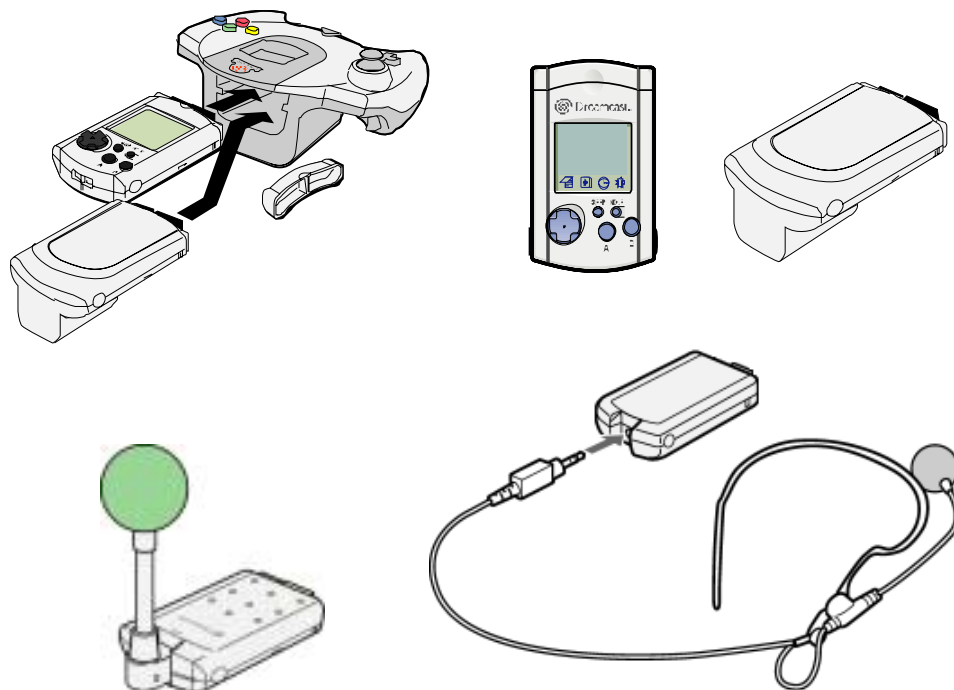
実際の製品としては、「ドリームキャスト・コントローラ」や「ドリームキャスト・キーボード」「アーケードスティック」「レーシングコントローラ」「マウス」などが、これに相当します。



(2) 拡張ソケットペリフェラル

「ドリームキャスト・コントローラ」のような拡張ソケットを備えたペリフェラルに、挿入するタイプのペリフェラルを指します。

「ビジュアルメモリ」「ぶるぶるばっく」「マイクデバイス」がこれに相当します。

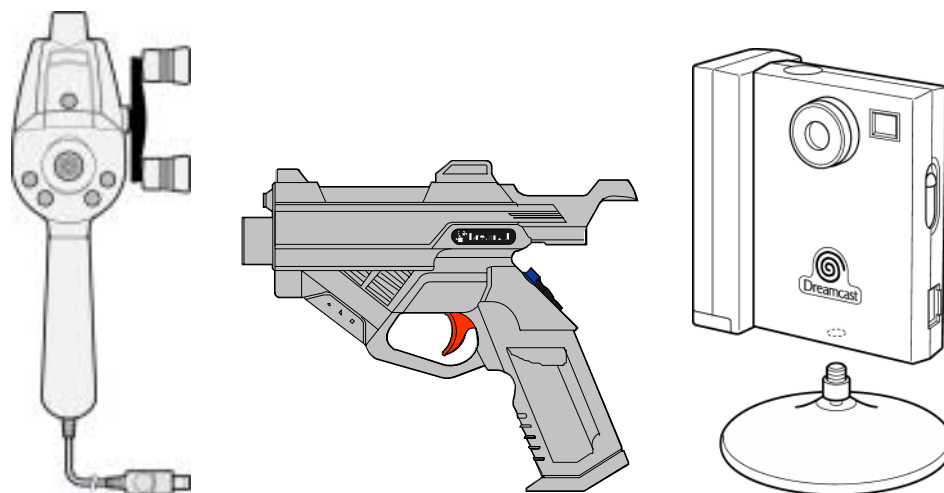


(3) 複合ペリフェラル

一つのペリフェラルに対し、複数のペリフェラルが統合されている方式のペリフェラルを指します。

これは同一のポートに複数のタイプのペリフェラルが接続されているのではなく、本来拡張ソケットに挿入されるようなペリフェラル（拡張ペリフェラル）を、あらかじめ内部に持ったペリフェラルのことを指します。

実際の製品としては、「つりコントローラ」「ドリームキャスト・ガン」「ドリームアイ」もこの系統に属します。



(4) ケーブルペリフェラル

Dreamcast 本体の後部に接続するビデオ出力端子に、接続されるペリフェラルのことを指します。

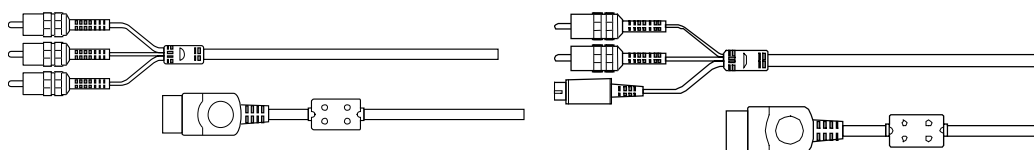
Dreamcast では、アプリケーション上で接続されているケーブルの種類を自動的に判別することが可能です。

注意 ただし、接続されているケーブルが判別された結果に対し、そのアプリケーションの画面モードに切り替えて表示する必要がありますので注意して下さい。

本体付属の「ステレオ AV ケーブル」や市販の製品では、「S 端子ケーブル」や「VGA ボックス」がこれに相当します。

● ステレオ AV ケーブル、S 端子ケーブル

テレビモニターに対し接続するためのケーブルです。それぞれの専用の端子（ビデオ入力端子や S 端子）に接続します。



S 端子は AV ケーブル端子でのビデオ出力に比べ、より鮮明な画像を得ることができます。

注意 アプリケーションプログラム中では、通常のビデオ出力か S ビデオ出力かについて、特に意識する必要はありません。

● VGA ボックス

パソコンなどで用いられる VGA 方式のモニターに出力する際に使用するケーブルです。

VGA 方式のモニターでは、テレビモニターよりも更に鮮明な画像を得られます。

ただし、アプリケーションプログラムについては、VGA 端子出力であることを意識したプログラムを、設定しなければなりません。

VGA ボックスには、画面モードの切り替えスイッチが付いています。これにより通常のビデオ出力と VGA 出力の切り替えができます。



a. EXTENSION (拡張) バスペリフェラル

Dreamcast 付属のモデムを外した際に現れる EXTENSION バス（G2 バス）に、接続するペリフェラルを指します。

ノート Dev.Box では、「モデム」を取り外すことはできませんが、別形状の EXTENSION バス端子が背面にあります。

現在のところは、「モデム」「ブロードバンドアダプタ」がこれに相当します。

6.1.2 ドリームキャスト・コントロールポートの特徴

- ドリームキャストのコントロールポートは標準で4基装備
コントロールポートには、最大4つまでのコントローラペリフェラルが接続できます。

ノート 現在のところ、サターン用「マルチターミナル6」のようにポートを分岐させるペリフェラルは、計画されていません。

- それぞれのポートに、（論理的には）最大5つの拡張ペリフェラルが接続可能
コントロールポートに接続したコントローラペリフェラルが拡張ソケットを持っている場合は、それぞれの拡張ソケットに対して拡張ペリフェラルを接続できます。その際の拡張ペリフェラルの仕様では、最大5つまでが接続できます。
一般的にコントローラペリフェラルは、1つまたは2つ程度の拡張ソケットを装備しています。また、外部の拡張ソケットを持たずにペリフェラルの内部に拡張ペリフェラルを内蔵しているタイプのペリフェラルもあります。
- ペリフェラルの仕様については、ペリフェラル自身に対して聞くことができます
アプリケーションは、接続されているペリフェラルに問い合わせることで「接続されているペリフェラルが何であるか?」「そのペリフェラルはどんな仕様なのか?」の情報を得ることができます。その場合、得た情報をベースに対応するペリフェラルにあわせた各種の設定を、行う必要があります。
- 各ペリフェラルには、最大電流と待機電流の値が記録されています
ペリフェラルの中には、振動ペリフェラルなどの多くの電力の消費を必要とするものがあります。コントロールポートが供給できる電力には上限があり、各ペリフェラルはこの値を超えることはできません。
- 特殊なペリフェラルに対しても、専用のライブラリが用意されています
以下の特殊なタイプの複合ペリフェラルに対しても、それぞれライブラリが用意されています。

つりコントローラ	振動デバイスが2つ搭載されています。振動用の関数で2つの振動デバイスを効果的に使用することにより、表現豊かな振動を実現できます。
ガン	専用の初期化関数があります。 ただし、ガンを有効にしている間は、他のデバイスが使えなくなる場合があるので注意して下さい。
キーボード	現在のところ、キーが押されているかどうかの判断しかできません。オートリピートやFEPなどについては、各アプリケーションからのサポートが必要になります。
マイクデバイス	音声の入力のみサポートしています。現在のところ、音声認識などの設定はアプリケーション側で行う必要があります。
マウス	接続の確認や押されているボタンの情報を取得したりマウスを動かした際の移動量を取得します。
ぶるぶるばっく	専用の振動関数群があります。

6.2 ペリフェラルデータライブラリ (pd) について

コントロールポートに接続されたペリフェラルからの入力データ取得、及びペリフェラル自身の情報の取得を行うライブラリです。

Dreamcast には標準で 4 つのペリフェラルの差込口があり、それぞれの差込口にはアナログコントローラやガンデバイスなどの入力装置を接続できる以外に、Dreamcast 専用コントローラには、液晶の小さな画面を備え外部バックアップメモリとしての機能を持つ、ビジュアルメモリという装置を装着することもできます。

6.2.1 ペリフェラルポートについて

ペリフェラルポートは、通称 Maple (メイプル) と呼ばれるセガオリジナルのシリアルプロトコルで接続され、最大約 2 Mbps の転送能力を持っています。

6.2.2 ペリフェラルの判定方法の関連事項

- ID 管理廃止の背景

Dreamcast では、コントロールポートに接続するペリフェラルを ID で管理していません。どのデバイスを有しているかという情報と共に、そのペリフェラルがコントローラデバイスを有している場合は、どのキーを有しているかの情報が返されるようになっています。

ID で管理を行わずにキーで管理することで、将来的に別種のペリフェラルが発売した際にも、既存のアプリケーションの動作に必要なボタンが実装されていれば、動作可能にできるような柔軟な対応が可能になります。

- id メンバ (PDS_PERIPHERAL 構造体) について

古い SDK での、pdGetPeripheral 関数が返す PDS_PERIPHERAL 構造体の id メンバは、コントローラの ID を示す値が設定されていました。実際には、その当時にサポートしていたペリフェラルのボタン (サポート情報 +) と、完全にペリフェラルどうしを区別することはできません。

注 意 現在では、このメンバには support メンバと同じ内容が設定されていますので、参照しないように注意して下さい。

- info/type メンバ

ペリフェラルの判定には、PDS_PERIPHERAL 構造体中のペリフェラル固有情報を示す info メンバ (PDS_PERIPHERALINFO) が指す type メンバから行います。

type メンバには 0 (未接続) か、コントローラデバイス、記録デバイス、LCD デバイス、タイマーデバイス、音声入力デバイス、キーボードデバイス、ガンデバイス、振動デバイスもしくは、これらの論理和が設定されています。

例えば、接続されているのがキーボードデバイスかを知りたい場合は、type のみで判定できます。

注 意 コントローラデバイスと判定されても、それがドリームキャストコントローラかレーシングコントローラかの区別はできませんので、注意して下さい。

- コントローラペリフェラルの判別

現時点で発売がされているペリフェラルに関しては、ペリフェラルがサポートしているボタンサポートの情報によりペリフェラルの種別を区別することができます。

注 意 ただし、現在のレーシングコントローラと同じボタンサポートの新しいペリフェラルが出た場合に、そのペリフェラルは必ずレーシングコントローラになるという保証はありませんので注意して下さい。

- 特定のペリフェラルを判断したい場合

厳密に、特定の種類のペリフェラルかどうかを判定したい場合（例えばボタンのキーアサイン設定画面で、特定のペリフェラルの場合は表示する画面を変更するような場合）は、ペリフェラル固有情報の製品名（product_name）から判定することができます。

この製品名は、同一のペリフェラルであれば、同一の製品名が設定されることが保証されています。

注 意 製品名による判別は、特殊アプリケーション専用のコントローラを用いる場合などにものみ使用し、一般的なコントローラやジョイスティック等の判別にはなるべく使用しないで下さい。

6.2.3 コントロールポートの初期化

コントロールポートを利用する前に行う初期化には、以下の2種類の初期化関数が用意されています。

- pdInitPeripheral()関数
- pdInitPeripheralEx()関数

注 意 すべてのアプリケーションの場合については、起動時の際に初期化関数を実行しなければなりません。

(1) pdInitPeripheral()関数による初期化

アプリケーションが利用するペリフェラルをコントロールポート A～D 及び、それぞれに2つの拡張ポートを想定した場合の初期化関数です。

この関数は、コントロールポート A～D のすべてに対応していますが、その分に必要なワークエリアも約 96KB と大きいサイズの容量になってしまいます。

a. pdInitPeripheral()関数の引数

plogic	論理モード
recvbuf	コントロールポート受信バッファ
sendbuf	コントロールポート送信バッファ

b. 引数（plogic）

PDS_PERIPHERAL 構造体のデジタルボタン情報ビットの正論理、負論理を選択するものです。

定 義	意 味
PDD_PLOGIC_ACTIVE	正論理
PDD_PLOGIC_NEGATIVE	負論理

c. 引数 (recvbuf、sendbuf)

送信バッファと受信バッファのポインタを指定するものです。

送信バッファは、ハードウェアに送るコマンドを作成するためのバッファです。

受信バッファは、ペリフェラルからのデータを格納するためのバッファです。

また、バッファアドレスが 32 バイト境界でない場合は、自動的にアサインするため、32 バイト分余分に確保するのがよいでしょう。

pdInitPeripheral()関数が必要とする送信バッファと受信バッファは、以下のように固定の値を指定します。

```
/* グローバルワークの宣言 */
Uint8 RecvBuf[1024 * 24 * 2 + 32];
Uint8 SendBuf[1024 * 24 * 2 + 32];

/* コントローラライブラリの初期化 */
pdInitPeripheral( PDD_PLOGIC_ACTIVE, RecvBuf, SendBuf );
```

(2) pdInitPeripheralEx()関数による拡張初期化

アプリケーションが利用するコントロールポートを、あらかじめ指定して行うように拡張された初期化関数です。

使用するコントロールポートを限定することにより、ワークメモリの大幅な削減とライブラリの内部処理の高速化が可能になります。

a. pdInitPeripheralEx()関数の引数

plogic	論理モード
pertbl	使用ペリフェラルテーブル
recvbuf	コントロールポート受信バッファ
sendbuf	コントロールポート送信バッファ
num	ペリフェラル

b. 引数 (plogic)

pdInitPeripheralEx()関数と同様に PDS_PERIPHERAL 構造体の、デジタルボタン情報ビットの正論理、負論理を選択するものです。

定 義	意 味
PDD_PLOGIC_ACTIVE	正論理
PDD_PLOGIC_NEGATIVE	負論理

c. 引数 (pertbl)

使用ペリフェラルテーブルを指定し、どのペリフェラルを使用するかを決めるものです。ペリフェラルポート番号の配列で指定するものです。

例えば、以下のような配列となります。

```
const Sint32 Devs[] = {
    PDD_PORT_A0, PDD_PORT_A1, PDD_PORT_A2, /* ポート A0/A1/A2 */
    PDD_PORT_B0, PDD_PORT_B1, PDD_PORT_B2, /* ポート B0/B1/B2 */
    PDD_PORT_C0, PDD_PORT_C1, PDD_PORT_C2, /* ポート C0/C1/C2 */
    PDD_PORT_D0, PDD_PORT_D1, PDD_PORT_D2, /* ポート D0/D1/D2 */
};
```

d. 引数 (recvbuf、sendbuf)

送信バッファと受信バッファのポインタを指定しますが、これらは固定ではなく利用するコントロールポート数から、あらかじめ計算された値を指定するものです。

PDM_WORK_SIZE マクロを使用して、必要分確保して下さい。

送信バッファは、ペリフェラル1つにつき 2KB 必要です。

PDM_WORK_SIZE マクロは、引数×ポート数を確保するマクロです。

以下の例では、ポート数 12 個分のワークを確保しています。

```
UInt8 SendBuf[PDM_WORK_SIZE(12) + 32];
```

受信バッファについても、ペリフェラル1つにつき 2KB 必要です。

以下の例では、ペリフェラル 12 個分のワークを確保しています。

これらのバッファは、連続している必要はありません。

```
UInt8RecvBuf[12][PDM_WORK_SIZE(1) + 32];
```

e. 引数 (num)

受信バッファアドレステーブルです。

これは、どのペリフェラルがどのワークを使用するかを指定するテーブルです。

注 意 使用ペリフェラルテーブルで指定したペリフェラルには、必ずバッファを割り当てて下さい。割り当てない場合は、ライブラリが正常に動作しません。

```
const void* RecvBufTbl[] = {
    RecvBuf[0],          /* A0 のワーク 2KB */
    RecvBuf[1],          /* A1 のワーク 2KB */
    RecvBuf[2],          /* A2 のワーク 2KB */
    RecvBuf[3],          /* B0 のワーク 2KB */
    RecvBuf[4],          /* B1 のワーク 2KB */
    RecvBuf[5],          /* B2 のワーク 2KB */
    RecvBuf[6],          /* C0 のワーク 2KB */
    RecvBuf[7],          /* C1 のワーク 2KB */
    RecvBuf[8],          /* C2 のワーク 2KB */
    RecvBuf[9],          /* D0 のワーク 2KB */
    RecvBuf[10],         /* D1 のワーク 2KB */
    RecvBuf[11],         /* D2 のワーク 2KB */
};
```

以上のパラメータを、pdInitPeripheralEx()関数に渡します。

```
pdInitPeripheralEx(PDD_PLOGIC_ACTIVE, Devs, RecvBufTbl, SendBuf,
    sizeof(Devs) / sizeof(Devs[0]));
```

初期化が終われば、ペリフェラルに対しアクセスができます。その結果、振動ペリフェラルや LCD、メモリーカード（バックアップ）ライブラリなどが使用できるようになります。

注 意 ただし、ここで使用しない設定にしたペリフェラルについては、認識しません。未接続として判断されますので、注意して下さい。

それぞれのポートのルートペリフェラル（A0、B0、C0、D0）を使用しない設定にした場合は、そのポートの拡張ポート（A1、A2 等）も使用できませんので注意して下さい。

● 関数一覧

関 数	機 能
pdInitPeripheral	ペリフェラルライブラリの初期化
pdInitPeripheralEx	ペリフェラルライブラリの拡張初期化
PDM_WORK_SIZE	ワークサイズ計算マクロ

6.2.4 ポートに接続されているペリフェラルの種別を取得

コントロールポートに接続されているペリフェラルの種別を取得するには、pdGetPeripheralInfo()関数を用います。この関数は取得したいポートを引数に与えて実行することで、PDS_PERIPHERALINFO 構造体のポインタを返します。引数のポートは、各コントロールポート A～D に加え、拡張ソケットに接続されているペリフェラルの指定をすることもできます。

引数 port に指定するポート番号は、以下のとおりです。

定 義	意 味
PDD_PORT_A0	コントロールポート A ペリフェラル本体
PDD_PORT_A1	コントロールポート A 拡張ソケット 1
PDD_PORT_A2	コントロールポート A 拡張ソケット 2
PDD_PORT_A3	コントロールポート A 拡張ソケット 3
PDD_PORT_A4	コントロールポート A 拡張ソケット 4
PDD_PORT_A5	コントロールポート A 拡張ソケット 5
PDD_PORT_B0	コントロールポート B ペリフェラル本体
PDD_PORT_B1	コントロールポート B 拡張ソケット 1
PDD_PORT_B2	コントロールポート B 拡張ソケット 2
PDD_PORT_B3	コントロールポート B 拡張ソケット 3
PDD_PORT_B4	コントロールポート B 拡張ソケット 4
PDD_PORT_B5	コントロールポート B 拡張ソケット 5
PDD_PORT_C0	コントロールポート C ペリフェラル本体
PDD_PORT_C1	コントロールポート C 拡張ソケット 1
PDD_PORT_C2	コントロールポート C 拡張ソケット 2
PDD_PORT_C3	コントロールポート C 拡張ソケット 3
PDD_PORT_C4	コントロールポート C 拡張ソケット 4
PDD_PORT_C5	コントロールポート C 拡張ソケット 5
PDD_PORT_D0	コントロールポート D ペリフェラル本体
PDD_PORT_D1	コントロールポート D 拡張ソケット 1
PDD_PORT_D2	コントロールポート D 拡張ソケット 2
PDD_PORT_D3	コントロールポート D 拡張ソケット 3
PDD_PORT_D4	コントロールポート D 拡張ソケット 4
PDD_PORT_D5	コントロールポート D 拡張ソケット 5

以下に、PDS_PERIPHERALINFO 構造体を示します。

```
typedef struct {
    Uint32 type;                /* ペリフェラルのタイプ */
    Uint32 reserved[3];        /* 予約 */
    Uint8 is_root;              /* ルートデバイスフラグ */
    Uint8 area_code;            /* エリアコード */
    Uint8 connector_dir[2];     /* 拡張ソケットの向き */
    Sint8 product_name[32];     /* 製品名 */
    Sint8 license[64];          /* ライセンス文字列 */
    Uint16 stdby_pow;           /* スタンバイ消費電流 */
    Uint16 max_pow;             /* 最大消費電流 */
} PDS_PERIPHERALINFO;
```

ペリフェラルのタイプを返すメンバ type には、次の値が定義されています。

定 義	意 味
PDD_DEVTYPE_CONTROLLER	コントローラデバイス
PDD_DEVTYPE_STORAGE	記録デバイス (メモリーカード)
PDD_DEVTYPE_LCD	LCD デバイス
PDD_DEVTYPE_TIMER	タイマーデバイス
PDD_DEVTYPE_SOUNDINPUT	音声入力デバイス
PDD_DEVTYPE_KEYBOARD	キーボードデバイス
PDD_DEVTYPE_LIGHTGUN	ガンデバイス
PDD_DEVTYPE_VIBRATION	振動デバイス
PDD_DEVTYPE_POINTING	ポインティングデバイス

それぞれはビットで表されています。一つのペリフェラルで、複数のタイプを所有する場合があります。例えば、「ビジュアルメモリ」は、「メモリーカード」「LCD」「タイマー」の3つタイプを返します。

ペリフェラルのエリアコードを示すメンバ area_code には、次の値が定義されています。

定 義	意 味
PDD_DEVAREA_USA	北アメリカ地区
PDD_DEVAREA_JAPAN	日本地区
PDD_DEVAREA_ASIA	アジア地区
PDD_DEVAREA_EUROPE	ヨーロッパ地区

それぞれはビットで表され、複数のエリアに対応したペリフェラルもあります。

拡張ソケットの向きは次の値が定義されています。

ペリフェラルの拡張ソケットの向きを示すメンバ connector_dir [2] には、次の値が定義されています。

定 義	意 味
PDD_CONDIR_TOPSIDE	上
PDD_CONDIR_BOTTOMSIDE	下
PDD_CONDIR_LEFTSIDE	左
PDD_CONDIR_RIGHTSIDE	右

これらの値は、ペリフェラルがルートデバイス (ポートに直接接続されるペリフェラル) である場合、拡張ソケットが付いている方向を表します。

ペリフェラルの製品名を示すメンバ product_name [32] には、ペリフェラルによってそれぞれ登録されている名称の文字列が返ります。

ライセンス文字列を示すメンバ license [64] には、製造社の名称が返ります。

スタンバイ電流を示すメンバ stdby_pow、及び最大電流を示すメンバ max_pow の単位は 0.1mA (ミリアンペア) です。

6.2.5 関数一覧

ペリフェラルデータライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
pdMouseInit	キーボードライブラリの初期化
pdKbdExit	キーボードライブラリの終了
pdInitPeripheral	コントロールポートの初期化
pdInitPeripheralEx	コントロールポートの拡張初期化
pdExitPeripheral	コントロールポートの終了処理
pdExecPeripheralServer	そのフレームのペリフェラルデータの作成
pdGetPeripheral	コントローラのボタン状態の取得
pdGetPeripheralDirect	ペリフェラルデータの取得（低レイテンシ用）
pdGetPeripheralInfo	ペリフェラルの固有情報の取得
pdIsPadReset	ソフトリセットコマンドの入力チェック
pdKbdGetData	キーボードデータの取得
pdKbdGetInfo	キーボードのハードウェア情報の取得
pdSetIntFunction	コントロールポート割り込み関数の登録
pdTmrAlarm	ペリフェラルのアラームを鳴らす
pdTmrGetTime	タイマーデバイスの時刻取得
pdTmrIsReady	タイマーデバイスの接続チェック
pdTmrSetTime	タイマーデバイスの時刻設定

6.3 コントロールデバイスライブラリについて

6.3.1 コントローラのボタンの状態を取得

ボタン情報を取得するには、pdGetPeripheral()関数を使用します。取得した PDS_PERIPHERAL 構造体のメンバを参照するだけで、容易にコントローラの入力を得ることができます。

- PDS_PERIPHERAL 構造体

```
typedef struct {
    Uint32 id;
    Uint32 support;
    Uint32 on;
    Uint32 off;
    Uint32 press;
    Uint32 release;
    Uint16 r;
    Uint16 l;
    Sint16 x1;
    Sint16 y1;
    Sint16 x2;
    Sint16 y2;
    Sint8* name;
    void* extend;
    Uint32 old;
    PDS_PERIPHERALINFO* info;
} PDS_PERIPHERAL;
```

メンバ	意 味
id	デバイス ID
support	ボタン・レバーのサポート状態
on	デジタルボタン状態
off	デジタルボタン状態（反転）
press	デジタルボタンダウンエッジ状態
release	デジタルボタンアップエッジ状態
r	アナログ軸 R の値（0～255）
l	アナログ軸 L の値（0～255）
x1	アナログ軸 X1 の値（-128～127）
y1	アナログ軸 Y1 の値（-128～127）
x2	アナログ軸 X2 の値（-128～127）
y2	アナログ軸 Y2 の値（-128～127）
name	デバイス名称
extend	拡張データアドレス（未使用）
old	予約
info	ペリフェラル情報構造体へのポインタ

6.3.2 コントローラのサポートしているボタンを取得

コントローラデバイスには、さまざまなボタンやレバー類が存在します。方向ボタンを2つ持つもの、アナログが2チャンネルのものなど、デバイス毎に有しているボタンやレバーが異なります。それを調べるには、PDS_PERIPHERAL のメンバ support を参照します。

以下の表に示されるようにビットアサインが決まっており、ボタンやレバーを持つ場合は1、持たない場合は0が格納されています。

コントローラデバイスは、方向ボタン A、スタートボタン、A ボタン、B ボタンを持つことが必須となっているため、これらのボタンのみを使用してアプリケーションを組むことにより、互換性を維持することができます。

ボタン・レバービット位置	ビット位置指定用定数
方向ボタン A 上	PDD_DEV_SUPPORT_KU
方向ボタン A 下	PDD_DEV_SUPPORT_KD
方向ボタン A 左	PDD_DEV_SUPPORT_KL
方向ボタン A 右	PDD_DEV_SUPPORT_KR
方向ボタン B 上	PDD_DEV_SUPPORT_KUB
方向ボタン B 下	PDD_DEV_SUPPORT_KDB
方向ボタン B 左	PDD_DEV_SUPPORT_KLB
方向ボタン B 右	PDD_DEV_SUPPORT_KRB
スタートボタン	PDD_DEV_SUPPORT_ST
A ボタン	PDD_DEV_SUPPORT_TA
B ボタン	PDD_DEV_SUPPORT_TB
C ボタン	PDD_DEV_SUPPORT_TC
X ボタン	PDD_DEV_SUPPORT_TX
Y ボタン	PDD_DEV_SUPPORT_TY
Z ボタン	PDD_DEV_SUPPORT_TZ
D ボタン	PDD_DEV_SUPPORT_TD
アナログ軸 R	PDD_DEV_SUPPORT_AR
アナログ軸 L	PDD_DEV_SUPPORT_AL
アナログ軸 X1	PDD_DEV_SUPPORT_AX1
アナログ軸 Y1	PDD_DEV_SUPPORT_AY1
アナログ軸 X2	PDD_DEV_SUPPORT_AX2
アナログ軸 Y2	PDD_DEV_SUPPORT_AY2

アプリケーションは、このボタンやレバーサポート情報を参照し、適切にデータを取り扱って下さい。

6.3.3 デジタルボタンの状態

デジタルボタン情報は、on、off、press、release の4つのメンバに格納されており、用途によって使い分けます。各メンバのビット毎にボタンが割り振られており、ボタンが押されていれば1、押されていない場合は0となります（正論理）。また、論理状態は初期化関数 `pdInitPeripheral()` 呼び出し時に負論理に設定することもできます。

デジタルボタン	ビット位置指定用定数
方向ボタン A 上	PDD_DGT_KU
方向ボタン A 下	PDD_DGT_KD
方向ボタン A 左	PDD_DGT_KL
方向ボタン A 右	PDD_DGT_KR
A ボタン	PDD_DGT_TA
B ボタン	PDD_DGT_TB
C ボタン	PDD_DGT_TC
D ボタン	PDD_DGT_TD
X ボタン	PDD_DGT_TX
Y ボタン	PDD_DGT_TY
Z ボタン	PDD_DGT_TZ
L ボタン	PDD_DGT_TL
R ボタン	PDD_DGT_TR
スタートボタン	PDD_DGT_ST
方向ボタン B 上	PDD_DGT_KUB
方向ボタン B 下	PDD_DGT_KDB
方向ボタン B 左	PDD_DGT_KLB
方向ボタン B 右	PDD_DGT_KRB
方向ボタン B 右	PDD_DGT_KRB

メンバ	意 味
Uint32 on	ボタンが押されているとき（ボタンダウン）、対応するビットが1になります。押されていないボタンのビットは0になります。
Uint32 off	On メンバをビット反転したものです。すなわち、ボタンが押されていないとき（ボタンアップ）、対応するビットが1になります。
Uint32 press	ボタンが押されていない状態から、押された状態に変化したとき（ボタンダウンエッジ）、対応するビットが1になります。そうでないビットは0になります。
Uint32 release	ボタンが押されている状態から、押されていない状態に変化したとき（ボタンアップエッジ）、対応するビットが1になります。そうでないビットは0になります。

負論理に設定した場合、button、on、off、press、release のすべてのビットは反転して格納されます。

なお、デジタル LR ボタン情報は、L/R トリガー情報からソフトウェアで擬似的に生成しています。

注 意 L/R トリガーを持たないデバイスの場合、ビットは変化しませんので注意して下さい（デジタル LR ボタンを物理的に持つデバイスはありません）。

6.3.4 アナログボタンの状態

アナログ軸情報は、メンバ `r`、`l`、`x1`、`y1`、`x2`、`y2` の6つのメンバに格納されています。未接続及びそのボタン、レバーを持たないデバイスの場合は、センター位置と同じデータが格納されます。

メンバ	符 号	レンジ	センター位置	レバー / ボタン
Uint16 <code>r</code>	なし	0 ~ 255	0	Rトリガー
Uint16 <code>l</code>	なし	0 ~ 255	0	Lトリガー
Sint16 <code>x1</code>	あり	-128 ~ 0 ~ 127	0	アナログ方向キーX1
Sint16 <code>y1</code>	あり	-128 ~ 0 ~ 127	0	アナログ方向キーY1
Sint16 <code>x2</code>	あり	-128 ~ 0 ~ 127	0	アナログ方向キーX2
Sint16 <code>y2</code>	あり	-128 ~ 0 ~ 127	0	アナログ方向キーY2

6.3.5 データ取得時のエラー

デバイスとの通信時に、何らかのエラーが発生する場合があります。その場合、ボタン等の正常なデータが得られないため、構造体は下の表のような値に設定されます。

メンバ	設定値
<code>id</code> , <code>name</code> , <code>extend</code>	前回の値
デジタルボタン	すべてボタンが押されていない状態
アナログ軸	センター位置

エラーが発生したかどうかは、関数 `pdGetPeripheralError()` で調べることができます。

エラーコード	内 容
<code>PDD_ERR_OK</code>	エラーなし
<code>PDD_ERR_RETRY</code>	ボタンデータが正常に取得できない
<code>PDD_ERR_GENERIC</code>	未定義エラー

エラーが発生した場合は、ライブラリでは回復手段が存在しません。アプリケーションで適切な処理を行って下さい。

6.3.6 ペリフェラルデータの低レイテンシ取得

従来の pdGetPeripheal()関数では、実際にコントローラからのデータが得られてから、アプリケーションに反映されるまでに 1 V ~ 2 V のレイテンシ (遅延) が発生します。

この遅延は、通常のアプリケーションでは全く意識する必要はありませんが、通信対戦などをおこなうアプリケーションでは、できる限り早くコントローラのデータを取得し送信処理を行う必要があるため無視できません。

ここで解説する関数を使用すれば、ハードウェアからの割り込み (Maple 割り込み) と同時に、最小限の遅延でペリフェラルデータを取得できます。

(1) 使用方法

割り込み登録用関数 pdSetIntFunction()を使用して、ユーザーの割り込み関数を登録します。

```
void UserFunc(void)
{
    :
}

pdSetIntFunction(UserFunc);
```

これにより、コントロールポートからの割り込みが発生すると同時に UserFunc が呼び出されます。この関数中で、pdGetPeripheralDirect()を使用することにより、コントローラのデータを取得することができます。

```
PDS_PERIPHERAL UserBuf;

void UserFunc(void)
{
    pdGetPeripheralDirect(PDD_PORT_A0, &UserBuf, NULL, NULL);
}
```

なお、従来の pdGetPeripheral()とは全く独立して動作しますので、pdGetPeripheral()関数の動作には影響を与えません。また、pdGetPeripheralDirect()関数は、特にこの割り込み関数中だけでなく、任意のタイミングで使用可能です。

(2) 注意事項

- 上記ユーザー割り込み関数中であっても、従来の pdGetPeripheral()を使用して取得したデータは、前フレームのデータであり、最小レイテンシのデータとはなりません。
pdGetPeripheralDirect()を使用して下さい。
- pdGetPeripheralDirect()に渡すバッファは、同じコントローラに対しては同じバッファを常に指定するようにして下さい。この関数は、PDS_PERIPHERAL 構造体の press、release メンバを、同構造体の old メンバを参照して生成します。つまり前回の値に依存します。異なるバッファを渡してしまうと、press、release メンバが正しく取得できません。
- VM へのファイルアクセス、液晶表示、ぶるぶるぱっくの振動等を行うと、ハードウェアの処理時間が長くなり、レイテンシ減少の効果を最大限に得ることができません。

6.3.7 関数一覧

コントロールデバイスライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関 数	機 能
pdGetPeripheral	コントローラのボタン状態を取得
pdGetPeripheralError	コントローラに関するエラーを取得
pdSetIntFunction	コントロールポートの割り込み関数を登録
pdGetPeripheralDirect	コントローラのボタン状態を取得（低レイテンシ用）

6.4 振動デバイスライブラリ (pdVib) について

コントローラ等に接続された、「ぶるぶるばっく」をはじめとする振動デバイスを制御するライブラリです。

振動デバイスライブラリには、一般振動デバイスの制御を行うための汎用関数と『ぶるぶるばっく』のみに対応した簡易型の専用関数があります。

6.4.1 振動の構造について

振動は、振動デバイス内のモーターが回転することにより発生します。このため、ペリフェラルの形状や重さ、内部構造の違いにより、同じ振動のパラメータを与えても振動の感じ方はそれぞれ異なります。

例えば、「ぶるぶるばっく」と「つりコントローラ」では、振動の構造は異なります。

6.4.2 振動の処理を実行するタイミング

振動デバイス进行操作するための関数は、基本的にコマンドを登録してすぐに復帰する即時復帰タイプの関数です。

登録したコマンドがデバイスに送られ、実際にデバイスが振動等の処理を実行するのは、次のV割り込みのタイミングとなります。また、登録したコマンドが実行されていない場合は、次の要求を発行してもBUSYを返します。

(1) 振動デバイスに対して処理を行う場合

1つのVに対して、1関数のみとなります。

例えば、

```
pdVibMxSetStopTime ( );  
pdVibMxStart ( );
```

と連続してコールしても前のコマンドが処理されていないので、pdVibMxStart () はBUSYの状態となり処理されません。

(2) トレイオープンなどの発生によって振動デバイスを停止する場合

SEGA Library Version 2.0 からは、ペリフェラルライブラリの終了関数 pdExitPeripheral() を呼び出すと自動的に振動が停止します。従って、特に振動停止処理を行う必要はありません。

6.4.3 振動デバイスの接続方向

コントローラの拡張コネクタの向きによって、ユーザーから見て振動の方向が異なる場合があります。たとえば、標準コントローラに振動パックを接続すると、振動の方向は反転してしまいます。振動デバイスの接続方向を確認するには、pdVibGetDirection()関数を使用します。

```
Sint32 dir;
dir = pdVibGetDirection(PDD_PORT_A2)
switch (dir) {
    case PDD_VIB_DIRECTION_NORMAL: /* 通常 */
        break;
    case PDD_VIB_DIRECTION_FLIP: /* 逆さま */
        break;
    case PDD_VIB_DIRECTION_LEFT: /* 左向き */
        break;
    case PDD_VIB_DIRECTION_RIGHT: /* 右向き */
        break;
}
```

6.4.4 振動ユニットのパラメータ

振動デバイス内部に存在し振動源となるハードウェアを「振動ユニット」と呼びます。

振動ユニットは、1 デバイス内に複数存在でき、1 ~ 15 個のユニットがサポートされています。ユニットには固有のパラメータがあり、次のようなものがあります。

パラメータ	内 容
振動ポジション	ユニットが、デバイスのどの位置に設置されているかを示します。前後左右 4 種類のポジションがあります。
振動軸	ユニットは、振動する軸（向き）を持ちます。軸なし、X 軸、Y 軸、Z 軸の 4 種類があります。
振動強度	ユニットには、振動の強度が 8 段階可変のものと、固定のものがあります。
連続振動	ユニットには、次に振動の設定をするまで連続して振動していただける連続振動をサポートしているものがあります。
振動方向	ユニットには、プラスとマイナスの振動方向を指定できるものがあります。
任意振動波形	ユニットには、任意の波形を作成しそのとおりに振動させることができるものがあります。
振動数	振動周波数のことです。

6.4.5 振動ユニット数と同時設定可能なユニット数

pdVibGetInfo()関数で振動デバイスが持つユニットの数や、振動の同時設定が可能なユニット数を、取得します。

```
PDS_VIBINFO info;
if (pdVibGetInfo(PDD_PORT_A2, &info) == PDD_VIBERR_OK) {
    /* 情報取得に成功 */
    printf("振動ユニットを%d 個内蔵しています。¥n", info.units);
    printf("同時に振動設定が可能なユニットは%d 個です。¥n", info.se_units);
} else {
    /* 振動デバイスは接続されていない */
}
```

同時設定可能なユニット数とは、1 回のコマンドの発行（1 回の pdVibStart()関数呼び出し）で振動が可能なユニットの数のことを指します。

6.4.6 振動ユニット情報の取得

pdVibEnumerateUnit()関数で、デバイスの持つすべての振動ユニットの情報を取得します。このユニット情報を取得するにはデバイスとの通信が必要なため、1 ユニットあたり 1INT の時間が必要です。

振動ユニット情報が格納される、PDS_VIBUNITINFO 構造体を次に示します。

```
typedef struct {
    Uint8 position; /* 振動ユニット位置 */
    Uint8 axis; /* 振動軸 */
    Uint8 pow_enable; /* 振動強度可変フラグ */
    Uint8 cont_enable; /* 連続振動フラグ */
    Uint8 dir_enable; /* 方向指定フラグ */
    Uint8 wave_enable; /* 任意振動波形フラグ */
    Uint8 min_freq; /* 最小振動周波数 */
    Uint8 max_freq; /* 最大振動周波数 */
} PDS_VIBUNITINFO;
```

振動ユニットの位置を示すメンバ position は、次の値を返します。

値	意 味
PDD_VIB_POS_FRONT	デバイスの前部に位置
PDD_VIB_POS_BACK	デバイスの後部に位置
PDD_VIB_POS_LEFT	デバイスの左側に位置
PDD_VIB_POS_RIGHT	デバイスの右側に位置

振動軸を示すメンバ axis は、次の値を返します。

値	意 味
PDD_VIB_AXIS_NONE	軸方向なし（回転運動等）
PDD_VIB_AXIS_X	X 軸方向（プレイヤーに対して左右方向）
PDD_VIB_AXIS_Y	Y 軸方向（プレイヤーに対して上下方向）
PDD_VIB_AXIS_Z	Z 軸方向（プレイヤーに対して前後方向）

振動強度可変フラグを示すメンバ pow_enable は、0 で固定、1 で 8 段階可変を示します。

連続振動フラグを示すメンバ count_enable は、0 で連続振動不可、1 で可能であることを示します。

方向指定フラグを示すメンバ `dir_enable` は、0 で振動方向固定、1 で振動方向指定可能（+、- で正方向、逆方向指定可能）であることを示します。

任意振動波形フラグを示すメンバ `wave_enable` は、0 で任意の振動波形の指定は不可、1 で可能であることを示します。

最小振動周波数を示すメンバ `min_freq` は、正の数でその周波数を、0 で周波数指定不可を示します。

最大振動周波数を示すメンバ `max_freq` は、正の数でその周波数を、0 で周波数指定不可を示します。

● 例

```

Uint32 flag;
Sint32 user_enum_unit_func(Uint32 unit, Sint32 stat,
const PDS_VIBUNITINFO* info, Uint32 param)
{
    switch (stat) {
        case PDD_VIBERR_NO_VIBRATOR:
            printf("振動デバイスはあります。¥n");
            flag = TRUE;          /* コールバックはこれ以降発生しない */
            break;
        case PDD_VIBERR_OK:
            printf("ユニット %d の情報¥n", unit);
            printf("位置 : %d¥n", info->position);
            :
            :
            if (unit == param) flag = TRUE; /* 全ユニットのコールバック終了 */
            break;
    }
    return PDD_VIBRET_OK;
}

Sint32 ret;
flag = FALSE;
if (pdVibGetInfo(PDD_PORT_A2, &info) != PDD_VIBERR_OK) return NG;
if (pdVibEnumerateUnit(PDD_PORT_A2, user_enum_unit_func, info.units) !=
    PDD_VIBERR_OK) return NG;
while (!flag);          /* 全ユニットのコールバックを待つ */

```

`user_enum_unit_func()`関数は、ユニットの個数の回数分呼び出され、次々とユニットの情報を取得します。

注意 `pdVibEnumerateUnit()`関数自体は即時復帰であり、コールバックは1 INT に1回ずつ発生するという点に注意が必要です。

6.4.7 振動させる

振動ユニットを振動させるには、pdVibStart()関数を使用します。この時、振動させるためのパラメータを構造体にセットする必要があります。pdVibEnumerateUnit()関数で取得したユニット情報を参照して、適切にパラメータを指定して下さい。

pdVibStart()関数の第3パラメータは、振動パラメータの個数、すなわち同時に振動設定をしたいユニットの個数のことです。これは振動デバイス情報にある「同時に振動設定可能なユニット数」を超えることはできません。

```
Sint32 ret;
PDS_VIBPARAM param;

param.unit = 1;          /* ユニット番号1(固定) */
param.flag = PDD_VIB_FLAG_CONTINUOUS; /* 連続振動あり */
param.power = 7;         /* 正方向に7の強さ(最大)で */
param.freq = 20;         /* 周波数10.5Hzで */
param.inc = 0;           /* 収束、発散なし(0) */

ret = pdVibStart(PDD_PORT_A2, &param, 1); /* 振動させる */

if (ret != PDD_VIBERR_OK) {
    /* 振動させることができなかった */
}
```

上の例では、ユニット1のみを振動させています。複数のユニットを振動させるには次のように、パラメータを構造体の配列として宣言し、使用します。

```
Sint32 ret;
PDS_VIBPARAM param[2];

param[0].unit = 1;          /* ユニット番号1 */
param[0].flag = PDD_VIB_FLAG_CONTINUOUS; /* 連続振動あり */
param[0].power = 7;         /* 正方向に7の強さ(最大)で */
param[0].freq = 20;         /* 10.5Hzで */
param[0].inc = 0;           /* 収束、発散なし(0) */

param[1].unit = 2;          /* ユニット番号2 */
param[1].flag = PDD_VIB_FLAG_EXHALATION; /* 単発発散振動 */
param[1].power = -7;        /* 負方向に7の強さ(最大)で */
param[1].freq = 15;         /* 8Hzで */
param[1].inc = 1;           /* 発散周期1 */

ret = pdVibStart(PDD_PORT_A2, param, 2); /* 振動させる */

if (ret != PDD_VIBERR_OK) {
    /* 振動させることができなかった */
}
```

pdVibStart()関数の第3パラメータは振動パラメータの個数、すなわち同時に振動設定をしたいユニットの個数です。これは振動デバイス情報にある「同時に振動設定可能なユニット数」を超えることはできません。

6.4.8 振動に必要なパラメータと設定例

pdVibStart()関数に渡す PDS_VIBPARAM 構造体の各メンバの意味と、設定例を解説します。

各振動ユニットへのパラメータを設定する際には、pdVibEnumerateUnit()関数で取得したユニット情報を参照し、適切な値を設定して下さい。パラメータが不正の場合、そのユニットあるいはデバイスの全ユニットが振動しない場合があります。

また、ハードウェアによっては予期せぬ動作を起こす場合がありますが、ライブラリではパラメータのチェックは行なっていません。

```
typedef struct {
    Uint8 unit;          /* ユニット番号 */
    Uint8 flag;          /* 振動フラグ */
    Sint8 power;         /* 強さ */
    Uint8 freq;          /* 振動周波数 */
    Uint8 inc;           /* 振動勾配周期 */
    Uint8 reserved[3]; /* 予約 */
} PDS_VIBPARAM;
```

(1) メンバ unit

ユニット番号を示すメンバ unit には、振動パラメータを送る振動ユニットを指定します。複数の振動ユニットのうちの、何番のユニットが対象になるかを指定します。

(2) メンバ flag

振動フラグを示すメンバ flag には次の値を指定します。

定 義	意 味
PDD_VIB_FLAG_CONTINUOUS	連続振動を指定します。振動自動停止時間が来るまで振動を続けます。指定しないと単発振動（ワンショット）となります。PDS_VIBUNITINFO 構造体で count_enable が 0 の場合、この指定はできません。
PDD_VIB_FLAG_EXHALATION	振動がだんだん強くなるよう指定します（発散振動）。連続振動時には、最大強度に達すると再び power で指定された強度から発散振動を開始します。PDS_VIBUNITINFO 構造体で pow_enable が 0 の場合、この指定はできません。
PDD_VIB_FLAG_CONVERGENCE	振動がだんだん弱くなるよう指定します（収束振動）。連続振動時には、最小強度に達すると再び power で指定された強度から収束振動を開始します。PDS_VIBUNITINFO 構造体で pow_enable が 0 の場合、この指定はできません。

PDD_VIB_FLAG_CONVERGENCE と PDD_VIB_FLAG_EXHALATION は同時に指定できません。

(3) メンバ power

強さを示すメンバ power には、-7 ~ 0 ~ +7 の値を指定します。正で正方向、負で逆方向に振動します（0 より -7 の値の方が数値は小さいが振動は大きくなります）。PDS_VIBUNITINFO 構造体で pow_enable が 0 の場合、この指定は無視されます。また、dir_enable が 0 の場合、この指定の符号は無視されます。

(4) メンバ freq

振動周波数を示すメンバ freq には、PDS_VIBUNITINFO 構造体で取得した最大振動周波数と最小振動周波数の範囲で指定します。指定する数値に対し実際の周波数は、ここで指定した値に 1 を足して 2 で割った値となります。「ぶるぶるぱっく」の場合、指定するのは、15 ~ 59 ですが、実際の周波数は 8Hz ~ 30Hz となります。

(5) メンバ inc

振動勾配周期を指定するメンバ inc には、収束や発散させる場合の周期を指定します。
指定できる範囲は、1 ~ 255 です。収束を発散させない場合は、無視されます。

6.4.9 振動を止める

振動を止める場合についても、VibStart()関数を使用します。
PDS_VIBPARAM.power の値を 0 にして、pdVibStart()を呼び出して下さい。

```
Sint32 ret;
PDS_VIBPARAM param;

param.unit = 2;
param.flag = 0;
param.power = 0;
param.freq = 0;
param.inc = 0;

ret = pdVibStart(PDD_PORT_A2, &param, 3);      /* 振動を止める */

if (ret != PDD_VIBERR_OK) {
    /* 止められなかった */
}
```

上の例では、ユニット 2 のみを停止させています。複数のユニットの振動を停止させるには、振動させる場合と同様に、パラメータを構造体の配列として宣言し、振動強度を 0 に設定して下さい。

6.4.10 自動停止時間の設定

自動停止時間を設定するには、pdVibSetStopTime()関数を使用します。

```
Sint32 ret;
Uint8 units[] = {1, 3, 5}; /* 設定したいユニット番号を格納した配列 */
Uint8 times[] = {10, 20, 30}; /* ユニット毎の設定値を格納した配列 */

ret = pdVibSetStopTime(PDD_PORT_A2, units, times, 3); /* 設定 */
if (ret != PDD_VIBERR_OK) {
    /* 設定できなかった */
}
```

この例では、ユニット 1、3、5 の自動停止時間をそれぞれ 10 (約 2.5 秒)、20 (約 5 秒)、30 (約 7.5 秒) に設定しています。

6.4.11 「ぶるぶるぱっく」専用関数

ユーザーが容易に振動デバイスを扱えるように、「ぶるぶるぱっく」専用の関数を用意しています。

基本的には汎用の振動関数群と使い方は同じですが、ハードウェアの仕様についてあらかじめ分かっているため、振動ユニットの情報を取得する必要がありません。また、振動ユニットが一つしか搭載されていないので、その分引数も単純化されています。

ノート 汎用の振動関数を使用して「ぶるぶるぱっく」を振動させることも可能です。
pdVibMx ~ 関数は、「ぶるぶるぱっく」専用の関数です。

● 「ぶるぶるぱっく」の仕様

項 目	内 容
振動ユニットの個数	1
振動ユニットの位置	前
振動ユニットの振動軸	なし
連続振動	可
振動方向設定（正方向、逆方向）	可
設定可能振動強度	-7 ~ 0 ~ +7
任意振動波形設定	不可
設定可能振動周波数	0FH(15) ~ 3BH(59) (8Hz ~ 30Hz)

(1) ぶるぶるぱっくを振動させる (pdVibMxStart())

「ぶるぶるぱっく」を振動させるには、pdVibMxStart()関数を実行します。このときに構造体 PDS_VIBPARAM へ指定する値は、pdVibStart()関数に指定するものと同様です。

```
Sint32 ret;
PDS_VIBPARAM param;

param.unit = 1;          /* ユニット番号1(固定) */
param.flag = PDD_VIB_FLAG_CONTINUOUS; /* 連続振動あり */
param.power = 7;         /* 正方向に7の強さ(最大)で */
param.freq = 20;         /* 周波数10.5Hzで */
param.inc = 0;           /* 収束、発散なし(0) */

ret = pdVibStart(PDD_PORT_A2, &param, 1); /* 振動させる */

if (ret != PDD_VIBERR_OK) {
    /* 振動させることができなかった */
}
```

(2) ぶるぶるぱっくを停止させる (pdVibMxStop())

「ぶるぶるぱっく」を停止させるには、pdVibMxStop()関数を実行します。

```
Sint32 ret;

ret = pdVibMxStop(PDD_PORT_A2);
if (ret != PDD_VIBERR_OK) {
    /* 停止させることができなかった */
}
```

- (3) ぶるぶるぱっくの自動停止時間を設定する (pdVibMxSetStopTime())
 「ぶるぶるぱっく」を停止させるには、pdVibMxSetStopTime()関数を実行します。

```
Sint32 ret;
Uint32 time;

Time = 10;      /* 自動停止時間を 10(約 2.5 秒) にする */
ret = pdVibMxSetStopTime (PDD_PORT_A2, time); /* 設定する */

if (ret != PDD_VIBERR_OK) {
    /* 設定できなかった */
}
```

6.4.12 関数一覧

振動デバイスドライブライブラリでは、以下の関数がサポートされています。
 詳しい内容については、関数リファレンスをご参照下さい。

関 数	機 能
pdVibIsReady	振動ペリフェラルの接続状況の取得
pdVibGetInfo	振動ペリフェラルの情報の取得
pdVibEnumerateUnit	振動ペリフェラルのユニット情報の取得
pdVibGetDirection	振動ペリフェラルの接続方向の取得
pdVibStart	振動ペリフェラルの振動開始
pdVibSetStopTime	振動ペリフェラルの振動時間の設定

- ぶるぶるぱっく専用関数一覧

関 数	機 能
pdVibMxIsReady	ぶるぶるぱっくの接続を調べる
pdVibMxStart	振動を開始
pdVibMxStop	振動を停止
pdVibMxSetStopTime	振動自動停止時間を設定

6.5 LCD デバイスライブラリ (pdLcd) について

コントローラ等に接続されたメモリカード（ビジュアルメモリ）のモノクロ液晶をコントロールする機能を提供するライブラリです。

ノート 現在、ビジュアルメモリ以外に LCD デバイスを持つペリフェラルは存在しません。

6.5.1 ビジュアルメモリの LCD デバイスのスペック

項 目	内 容
解像度	横 48×縦 32 ピクセル
階調数	2 調（モノクロ）
液晶のベース色	白
コントラスト調整	なし

6.5.2 ピクセルデータ形式について

ピクセルデータはソフトウェアで取り扱いやすいように、1 ピクセルを 8 ビット（1 バイト）で表現します。下位 4 ビットで階調を表現しますが、このライブラリ上ではビット 3 のみを参照します。

白（液晶のベース色）にするピクセルに対しては、ビット 3 を 0 にしたデータ（0x00、0x07 等）

黒にするピクセルに対しては、ビット 3 を 1 にしたデータ（0x0f、0x08 等）を格納して下さい。

bit							
7	6	5	4	3	2	1	0
アプリケーション依存				階調データ (0~15)			

- グラフィックの表示

低レベル関数を使用して、LCD デバイスに直接グラフィックを表示します。この関数は 1 ビット 1 ピクセルのデータを、そのまま LCD デバイスに送信し表示します。

6.5.3 ピクセルデータフォーマット

LCD デバイスの液晶画面全体では、(48×32=) 1536 バイトのピクセルデータになります。ピクセルデータは左から右、上から下に連続して格納します。

- ピクセルデータのメモリエイメージ

0x8C100000	(0, 0)	(1, 0)	(2, 0)	...	(47, 0)
0x8C100030	(0, 1)	(1, 1)	(2, 1)	...	(47, 1)
:					:
0x8C1005D0	(0, 31)	(1, 31)	(2, 31)	...	(47, 31)

ピクセルデータのメモリエイメージ（アドレスは例）ソフトウェア上の記述には、以下の様な方法を用いることができます。

```
const Uint8 cgdata[48 * 32] = {
    0x00, 0x00, 0x0f, 0x0f, ...
    :
};
```

6.5.4 液晶の向き

コントローラの拡張ソケットの向きによって、ユーザーから見ると液晶画面が逆さまになったり、横を向いていたりする場合があります。

例えば、標準コントローラにビジュアルメモリを接続した場合は、液晶画面は逆向きになってしまいます。

方 向	定 義	
正方向	PDD_LCD_DIRECTION_NORMAL	
逆向き	PDD_LCD_DIRECTION_FLIP	
左向き	PDD_LCD_DIRECTION_LEFT	
右向き	PDD_LCD_DIRECTION_RIGHT	

6.5.5 液晶使用時の注意事項

液晶デバイスとライブラリとの併用時には、毎フレーム（毎 INT）液晶の表示を行うことは避け、最低でも 1 フレーム以上の間隔を空けて下さい。そうでない場合、記録デバイスライブラリのビジュアルメモリへのアクセスが行えず、マウント処理やロード・セーブ等のファイル処理に時間が掛かったり、エラーとなる場合がありますので、ご注意下さい。

6.5.6 関数一覧

LCD デバイスデバイスライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関 数	機 能
pdVmsLcdWrite	LCD デバイスの表示
pdVmsLcdWrite1	LCD デバイスの表示（ 1 ピクセル 1 ビット形式）
pdLcdGetDirection	LCD デバイスの向きの検出
pdVmsLcdIsReady	LCD デバイスの接続状況の取得

6.6 タイマーデバイス (pdTmr) ライブラリについて

コントローラに接続されたタイマーデバイスを取り扱う機能を提供するライブラリです。
現在のところ、ビジュアルメモリ以外にタイマーデバイスを持つペリフェラルは、存在しません。

以下は、タイマーデバイスの設定時刻を格納する構造体です。

6.6.1 定義

- PDS_TIME 構造体

```
typedef struct {
    Uint16 year;
    Uint8 month;
    Uint8 day;
    Uint8 hour;
    Uint8 minute;
    Uint8 second;
    Uint8 dayofweek;
} PDS_TIME;
```

メンバ	意 味
year	年
month	月
day	日
hour	時
minute	分
second	秒
dayofweek	曜日 (日=0、月=1、...、土=6)

dayofweek メンバは時刻設定時には、必ず有効な値を設定し、時刻取得時には、参照せず年月日から計算するようにすることで、将来のタイマーデバイスに対応することができます。

6.6.2 関数一覧

タイマーデバイスデバイスライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関 数	機 能
pdTmrSetTime	時刻を設定する
pdTmrGetTime	時刻を取得する
pdTmrAlarm	アラームを直接コントロールする
pdTmrIsReady	タイマーデバイスがあるかどうか調べる

6.7 ガンデバイスライブラリ (pdGun) について

ドリームキャスト・ガンを取り扱うための機能を提供するライブラリです。

6.7.1 ガンモードの設定

ドリームキャスト・ガンは、TV 画面の走査線を検出する従来の Saturn と同じ方式を採用しています。

ガンデバイスを使用するには、コントローラライブラリを「ガンモード」と呼ばれる特殊な状態に設定する必要があります。ガンモードに設定している間は、「6.9.3 ペリフェラルの同時使用制限」に記述してあるような制限が発生します。

これは、以下のような理由によるものです。

- a. ガンデバイスは通常のコントローラやキーボード、ビジュアルメモリといったペリフェラルと大幅に異なるアクセス方式でコントロールしているため
ガンデバイスをコントロールするには、画面をフラッシュさせて 1 V の間にコントロールポートを占有し、走査線を検出する必要があります。
- b. ペリフェラルとの通信上のタイムアウト時間を短く設定しているため
通信タイムアウトの設定が通常の設定のままだと、1 V の間がフルにコントロールポートを占有することができず画面上部の座標が検出できなくなります。
この場合は、タイムアウト設定が短いため、VM のアクセスは不可能です。
- c. LCD デバイスやマイクデバイスなどの、データ量の多いペリフェラルとの通信を避けなければいけないため。
データ量の多いペリフェラルへアクセスを行うと、b と同様に 1 V の間がフルにコントロールポートを占有することができず画面上部の座標が検出できなくなります。
この場合は、データ量が多いため LCD デバイスやマイクデバイスのアクセスは不可能です。

6.7.2 拡張ペリフェラルに関する制限事項

ガンモード中では、拡張ペリフェラルのアクセスに、大幅な制限が加わります。

以下のデバイスは、ガンモード時には使用できません。

- 記録デバイス (メモリーカード)
- LCD デバイス
- タイマーデバイス

特に、記録デバイスについては、メモリーカード (バックアップ) ライブラリの buExit() 関数を使用して、終了しておいて下さい。使用してもマウント処理に失敗し、メモリーカードを認識することができません。

ガンモードでは、記録デバイスが使用できないため、アプリケーションでは次のような方法でセーブを行います。

- 通常はガンモードで動作し、セーブ、ロード画面のみ通常モードにします。
- この場合、セーブ画面に入る段階などでバックアップライブラリの初期化を行い、セーブ画面を抜けるときに終了します。

6.7.3 画面フラッシュ

ガンモード中にガンのトリガーを引くと、ガン座標の検出のためにライブラリが内部で自動的に 1INT の間、画面フラッシュさせます。ガン座標とは、ガンの照準が画面に向かっていているときトリガーが引かれ、その時に検出した座標のことを指します。

画面フラッシュの色は、デフォルト状態では白 (0x00c0c0c0) です。画面フラッシュを行わないと、ガン座標を正しく検出できません。

6.7.4 ガン座標取得の位置の制限

ドリームキャスト・ガンでは、画面上の左右端または下端でのガン座標が取得できない場合があります。この現象は、左右端についてはオフセットによる左右の折り返し点が厳密に決定できない、もしくは下端についてはレンズの画角内で常に一番下の座標がデータとなるために起こります。

また TV モニターは、機種により上下左右が同一に表示されるとは限らないため、640×480 の時で少なくとも上下左右の端 16 ピクセル、320×240 の時で 8 ピクセルを、ガン座標取得の対象としないようにしなければなりません。

6.7.5 ガンデバイスの優先順位

コントロールポートを占有できるのは、1V に 1 つのガンのみです。従って、1V で複数のガンの座標検出を行うことはできません。

ライブラリでは、ポート A から優先的にデータを取得しています。従って複数のガンのトリガーが同時に引かれた場合、最も若い番号のポートのトリガーが優先的に判断され、座標の検出を行います。

(1) キャリブレーション調整

使用する TV モニタや画面のモード等により、ハードウェアから得られる座標は大幅に異なる場合があります。

必ずキャリブレーションを調整する画面を用意し、ユーザーの環境に合わせてガンの照準調整を行う必要があります。

キャリブレーションとは、ハードウェアから取得できる座標とスクリーン座標を対応させる作業で、SDK 付属サンプルでは、以下のような手順でキャリブレーションを行っています。

- a. スクリーン左上座標 (32、40) に表示したターゲットをユーザーに撃たせることで、座標 (32、40) に対応したガン座標を取得する。
- b. スクリーン右下座標 (640 - 32、480 - 40) に表示したターゲットをユーザーに撃たせることで、座標 (640 - 32、480 - 40) に対応したガン座標を取得する。
- c. スクリーン中心座標 (320、240) に表示したターゲットをユーザーに撃たせることで、座標 (320、240) に対応したガン座標を取得する。

6.7.6 関数一覧

ガンデバイスデバイスデバイスライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関 数	機 能
pdGunEnter	ポートをガンモードに設定
pdGunGetLatchedPort	ガン座標を検出したポートの取得
pdGunGetPosition	ガン座標の取得
pdGunLeave	ポートを非ガンモードに設定
pdGunSetCallback	トリガー設定用コールバック関数の登録
pdGunSetFlashColor	画面フラッシュの色指定
pdGunSetTrigger	疑似トリガーの生成

6.8 キーボードデバイスライブラリ (pdKbd) について

「ドリームキャスト・キーボード」を利用するためのライブラリです。

6.8.1 ハードウェアの取得方式について

Dreamcast で使用するキーボードのハードウェアは、従来の MAKE、BREAK コードを返す方式とは異なり、その時点で押されているキーコードを複数取得し返す構造になっています。

アプリケーションからキー入力を行う場合には、1 フレーム (1 INT) 毎に毎回データを取得し、前回との差分によってキーアップ、キーダウン、キーリピートの処理を行う必要があります。

6.8.2 構造体仕様

- PDS_KEYBOARD

```
typedef struct {
    Uint8 ctrl;
    Uint8 led;
    Uint8 key[6];
    PDS_KEYBOARDINFO *info;
} PDS_KEYBOARD;
```

メンバ	意 味
ctrl	特殊キーの状態
led	LED の点灯状態
key	キーデータ
info	キーボードのハードウェア情報

6.8.3 コントロールキー

特殊キーの状態を示すメンバ ctrl には、現在どのコントロールキーが押されているかの情報が格納されます。

キーが押されている場合 対応するビットが 1

キーが押されていない場合 対応するビットが 0 になります。

- 対応するビットの定義

ビット位置指定定数	押されているキー
PDD_KEY_CTRL_RGUI	S2 キー
PDD_KEY_CTRL_RALT	右 ALT
PDD_KEY_CTRL_RSHIFT	右シフト
PDD_KEY_CTRL_RCTRL	右コントロール
PDD_KEY_CTRL_LGUI	S1 キー
PDD_KEY_CTRL_LALT	左 ALT
PDD_KEY_CTRL_LSHIFT	左シフト
PDD_KEY_CTRL_LCTRL	左コントロール

6.8.4 LED 点灯状態

LED の点灯状態を示すメンバ `led` には、現在どの LED が点灯しているかの情報が格納されます。
 LED が点灯している場合 対応するビットが 1
 LED が点灯していない場合 対応するビットが 0 になります。

- 対応するビットの定義

ビット位置指定定数	LED
PDD_LED_SHIFT	シフト
PDD_LED_POWER	電源
PDD_LED_KANA	カナ
PDD_LED_SCRLOCK	Scroll Lock
PDD_LED_CAPLOCK	Caps Lock
PDD_LED_NUMLOCK	Num Lock

注 意 LED のないキーボードには、このメンバは意味がありません。

6.8.5 キーコード

キーデータを示すメンバ `key` には、押されているキーコードが格納されています。キーコードは、最初に押されたキーから 6 番目に押されたキーまでのキーコードが `key[0]` から `key[5]` に順に格納されます。

- (1) 押されているキーが 6 個に満たない場合
残りのメンバには 0x00 が格納されます。
- (2) 押されているキーが 7 個以上を同時に押している場合
検出することはできません。
- (3) 7 個以上のキーが押された場合
`key[0]` に 0x01 が格納されます。

ハードウェアの構成上、押すキーの組み合わせによっては 3 つ以上の同時押しでも同様にキーコードの検出は不可能になり、`key[0]` に 0x01 が格納されます。

メンバ `key` には、押されているキーコードが格納されています。

キーコードは、6 番目に押されたキーまでのキーコードが `key[0]` から `key[5]` に順に格納されます。押されているキーが 6 個に満たない場合、残りのメンバには 0x00 が格納されます。7 個以上のキーの同時押しを検出することはできません。7 個以上のキーが押された場合、`key[0]` に 0x01 が格納されます。また、ハードウェアの構成上、押すキーの組み合わせによっては 6 個以下の同時押しでも同様にキーコード検出不可能になり、`key[0]` に 0x01 が格納されます。

- キーボードの各国キーコード表 (国・英国・日本)

Code	Code (hexadecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK)	Japanese 92 Keyboard
0	00h	No operation	No operation	No operation
1	01h	Roll-over error	Roll-over error	Roll-over error
2	02h	POST Fail	POST Fail	POST Fail
3	03h	Undefined error	Undefined error	Undefined error

Code	Code (hexadecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK)	Japanese 92 Keyboard
4	04h	"a", "A"	"a", "A"	"a", "A"
5	05h	"b", "B"	"b", "B"	"b", "B"
6	06h	"c", "C"	"c", "C"	"c", "C"
7	07h	"d", "D"	"d", "D"	"d", "D"
8	08h	"e", "E"	"e", "E"	"e", "E"
9	09h	"f", "F"	"f", "F"	"f", "F"
10	0Ah	"g", "G"	"g", "G"	"g", "G"
11	0Bh	"h", "H"	"h", "H"	"h", "H"
12	0Ch	"i", "I"	"i", "I"	"i", "I"
13	0Dh	"j", "J"	"j", "J"	"j", "J"
14	0Eh	"k", "K"	"k", "K"	"k", "K"
15	0Fh	"l", "L"	"l", "L"	"l", "L"
16	10h	"m", "M"	"m", "M"	"m", "M"
17	11h	"n", "N"	"n", "N"	"n", "N"
18	12h	"o", "O"	"o", "O"	"o", "O"
19	13h	"p", "P"	"p", "P"	"p", "P"
20	14h	"q", "Q"	"q", "Q"	"q", "Q"
21	15h	"r", "R"	"r", "R"	"r", "R"
22	16h	"s", "S"	"s", "S"	"s", "S"
23	17h	"t", "T"	"t", "T"	"t", "T"
24	18h	"u", "U"	"u", "U"	"u", "U"
25	19h	"v", "V"	"v", "V"	"v", "V"
26	1Ah	"w", "W"	"w", "W"	"w", "W"
27	1Bh	"x", "X"	"x", "X"	"x", "X"
28	1Ch	"y", "Y"	"y", "Y"	"y", "Y"
29	1Dh	"z", "Z"	"z", "Z"	"z", "Z"
30	1Eh	"1", "!"	"1", "!"	"1", "!"
32	20h	"3", "#"	"3", "£"	"3", "#"
33	21h	"4", "\$"	"4", "\$"	"4", "\$"
34	22h	"5", "%"	"5", "%"	"5", "%"
35	23h	"6", "^"	"6", "^"	"6", "&"

Code	Code (hexadecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK)	Japanese 92 Keyboard
36	24h	"7", "&"	"7", "&"	"7", ""
37	25h	"8", "*"	"8", "*"	"8", "("
38	26h	"9", "("	"9", "("	"9", ")"
39	27h	"0", ")"	"0", ")"	"0", "~"
40	28h	"Enter"	"↵"	"Enter"
41	29h	"Esc"	"Esc"	"Esc"
42	2Ah	" (Backspace)	" (Backspace)	"Back Space"
43	2Bh	"Tab"	"Tab"	"Tab"
44	2Ch	Spacebar	Spacebar	Spacebar
45	2Dh	"-", "_"	"-", "_"	" - ", " = "
46	2Eh	"=", "+"	"=", "+"	"^", " "
47	2Fh	"[", "{"	"[", "{"	"@", " ^ "
48	30h	"]", "}"	"]", "}"	"[", "{"
49	31h	"¥", " "	Not Used	Not Used
50	32h	Not Used	"#", "~"	"]", "}"
51	33h	".", "." ," , "	".", "." ," , "	"," , "+"
52	34h	"" , "" ," , "	"" , "@"	"," , "*"
53	35h	"`", " _ "	"`", "~"	"半角/全角"
54	36h	"," , "<"	"," , "<"	"," , "<"
55	37h	"," , ">"	"," , ">"	"," , ">"
56	38h	"/", "?"	"/", "?"	"/", "?"
57	39h	"Caps Lock"	"Caps Lock"	"Caps Lock"
58	3Ah	"F1"	"F1"	"F1"
59	3Bh	"F2"	"F2"	"F2"
60	3Ch	"F3"	"F3"	"F3"
61	3Dh	"F4"	"F4"	"F4"
62	3Eh	"F5"	"F5"	"F5"
63	3Fh	"F6"	"F6"	"F6"
64	40h	"F7"	"F7"	"F7"
65	41h	"F8"	"F8"	"F8"
66	42h	"F9"	"F9"	"F9"

Code	Code (hexadecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK)	Japanese 92 Keyboard
67	43h	"F10"	"F10"	"F10"
68	44h	"F11"	"F11"	"F11"
69	45h	"F12"	"F12"	"F12"
71	47h	"Scroll Lock"	"Scroll Lock"	"Scroll Lock"
72	48h	"Pause"	"Pause"	"Pause"
73	49h	"Insert"	"Insert"	"Insert"
74	4Ah	"Home"	"Home"	"Home"
75	4Bh	"Page Up"	"Page Up"	"Page Up"
76	4Ch	"Delete"	"Delete"	"Delete"
77	4Dh	"End"	"End"	"End"
78	4Eh	"Page Down"	"Page Down"	"Page Down"
79	4Fh	" "	" "	" "
80	50h	" "	" "	" "
81	51h	" "	" "	" "
82	52h	" "	" "	" "
83	53h	["Num Lock"]	["Num Lock"]	Not Used
84	54h	["/"]	["/"]	Not Used
85	55h	["*"]	["*"]	Not Used
86	56h	["-"]	["-"]	Not Used
87	57h	["+"]	["+"]	Not Used
88	58h	["Enter"]	["Enter"]	Not Used
89	59h	["1", "End"]	["1", "End"]	Not Used
90	5Ah	["2", " "]	["2", " "]	Not Used
91	5Bh	["3", "Pg Dn"]	["3", "Pg Dn"]	Not Used
92	5Ch	["4", " "]	["4", " "]	Not Used
93	5Dh	["5"]	["5"]	Not Used
94	5Eh	["6", " "]	["6", " "]	Not Used
95	5Fh	["7", "Home"]	["7", "Home"]	Not Used
96	60h	["8", " "]	["8", " "]	Not Used
97	61h	["9", "Pg Up"]	["9", "Pg Up"]	Not Used
98	62h	["0", "Ins"]	["0", "Ins"]	Not Used

Code	Code (hexadecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK)	Japanese 92 Keyboard
99	63h	[".", "Del"]	[".", "Del"]	Not Used
100	64h	Not Used	"¥", " "	Not Used
101	65h	"S3"	"S3"	"S3"
135	87h	Not Used	Not Used	"¥", "_"
136	88h	Not Used	Not Used	"カタカナ", "ひらがな"
137	89h	Not Used	Not Used	"¥", " "
138	8Ah	Not Used	Not Used	"変換"
139	8Bh	Not Used	Not Used	"無変換"

● 欧州のキーボードのキーコード表（英国以外）

Code	Code (hexadecimal)	France 105 Keyboard	Germany 105 Keyboard	Spain 105 Keyboard	Italy 105 Keyboard
0	00h	No operation	No operation	No operation	No operation
1	01h	Roll-over error	Roll-over error	Roll-over error	Roll-over error
2	02h	POST Fail	POST Fail	POST Fail	POST Fail
3	03h	Undefined error	Undefined error	Undefined error	Undefined error
4	04h	"q", "Q"	"a", "A"	"a", "A"	"a", "A"
5	05h	"b", "B"	"b", "B"	"b", "B"	"b", "B"
6	06h	"c", "C"	"c", "C"	"c", "C"	"c", "C"
7	07h	"d", "D"	"d", "D"	"d", "D"	"d", "D"
8	08h	"e", "E", "€"	"e", "E", "€"	"e", "E", "€"	"e", "E", "€"
9	09h	"f", "F"	"f", "F"	"f", "F"	"f", "F"
10	0Ah	"g", "G"	"g", "G"	"g", "G"	"g", "G"
11	0Bh	"h", "H"	"h", "H"	"h", "H"	"h", "H"
12	0Ch	"i", "I"	"i", "I"	"i", "I"	"i", "I"
13	0Dh	"j", "J"	"j", "J"	"j", "J"	"j", "J"
14	0Eh	"k", "K"	"k", "K"	"k", "K"	"k", "K"
15	0Fh	"l", "L"	"l", "L"	"l", "L"	"l", "L"
16	10h	",", "?"	"m", "M", "μ"	"m", "M"	"m", "M"
17	11h	"n", "N"	"n", "N"	"n", "N"	"n", "N"
18	12h	"o", "O"	"o", "O"	"o", "O"	"o", "O"
19	13h	"p", "P"	"p", "P"	"p", "P"	"p", "P"
20	14h	"a", "A"	"q", "Q", "@"	"q", "Q"	"q", "Q"
21	15h	"r", "R"	"r", "R"	"r", "R"	"r", "R"
22	16h	"s", "S"	"s", "S"	"s", "S"	"s", "S"
23	17h	"t", "T"	"t", "T"	"t", "T"	"t", "T"
24	18h	"u", "U"	"u", "U"	"u", "U"	"u", "U"
25	19h	"v", "V"	"v", "V"	"v", "V"	"v", "V"
26	1Ah	"z", "Z"	"w", "W"	"w", "W"	"w", "W"
27	1Bh	"x", "X"	"x", "X"	"x", "X"	"x", "X"
28	1Ch	"y", "Y"	"z", "Z"	"y", "Y"	"y", "Y"
29	1Dh	"w", "W"	"y", "Y"	"z", "Z"	"z", "Z"
30	1Eh	"&", "1"	"1", "!"	"1", "!", " "	"1", "!"

Code	Code (hexadecimal)	France 105 Keyboard	Germany 105 Keyboard	Spain 105 Keyboard	Italy 105 Keyboard
31	1Fh	"é", "2", "~"	"2", " ", "2"	"2", " ", "@"	"2", " "
32	20h	"", "3", "#"	"3", " § ", "3"	"3", ". ", "#"	"3", "£"
33	21h	"", "4", "{"	"4", "\$"	"4", "\$"	"4", "\$"
34	22h	"(", "5", ""	"5", "%"	"5", "%"	"5", "%"
35	23h	"-", "6", " "	"6", "&"	"6", "&", ""	"6", "&"
36	24h	"è", "7", "`"	"7", "/", "{"	"7", "/"	"7", "/"
37	25h	"_", "8", "¥"	"8", "(", ""	"8", "("	"8", "("
38	26h	"ç", "9", "^"	"9", ")", ""	"9", ")"	"9", ")"
39	27h	"à", "0", "@"	"0", "=", "}"	"0", "="	"0", "="
41	29h	"Echap"	"ESC"	"ESC"	"ESC"
42	2Ah	"Delete"(BS)	"Delete"(BS)	"Delete"(BS)	"Delete"(BS)
43	2Bh	"Tab"	"Tab"	"Tab"	"Tab"
44	2Ch	"Spacebar"	"Spacebar"	"Spacebar"	"Spacebar"
45	2Dh	")", " ° ", ""	"ß", "?", "¥"	"", "?"	"", "?"
46	2Eh	"=", "+", "}"	"´", "¨", "`"	"ı", "ç"	"ı", "^"
47	2Fh	"^", " ¨ "	"ü", "Ü"	"¨", "^", ""	"è", "é", ""
48	30h	"\$ ", "£", "¤"	"+", "*", "~"	"+", "*", ""	"+", "*", ""
49	31h	Not Use	Not Use	Not Use	Not Use
50	32h	"*", "µ"	"#", ""	"ç", "Ç", "}"	"ù", " § "
51	33h	"m", "M"	"ö", "Ö"	"n", "N"	"ò", "ç", "@"
52	34h	"ù", "%"	"ä", "Ä"	"´", "¨", " ", "{"	"à", " ° ", "#"
53	35h	"2"	"^", " ° "	"o", "a", "¥"	"¥", " "
54	36h	"., " "	" " ", " "	" " ", " "	" " ", " "
55	37h	".", "/"	" ", ". "	".", ". "	".", ". "
56	38h	"!", " § "	"-", " _ "	"-", " _ "	"-", " _ "
57	39h	"Caps Lock"	"Caps Lock"	"Blocq Mayus"	"Caps Lock"
58	3Ah	"F1"	"F1"	"F1"	"F1"
59	3Bh	"F2"	"F2"	"F2"	"F2"
60	3Ch	"F3"	"F3"	"F3"	"F3"
61	3Dh	"F4"	"F4"	"F4"	"F4"
62	3Eh	"F5"	"F5"	"F5"	"F5"

Code	Code (hexadecimal)	France 105 Keyboard	Germany 105 Keyboard	Spain 105 Keyboard	Italy 105 Keyboard
63	3Fh	"F6"	"F6"	"F6"	"F6"
64	40h	"F7"	"F7"	"F7"	"F7"
65	41h	"F8"	"F8"	"F8"	"F8"
66	42h	"F9"	"F9"	"F9"	"F9"
67	43h	"F10"	"F10"	"F10"	"F10"
68	44h	"F11"	"F11"	"F11"	"F11"
69	45h	"F12"	"F12"	"F12"	"F12"
70	46h	"impr ecran"	"Druck"	"Impr Pant"	"Stamp"
71	47h	"Arret defil"	"Rollen"	"Bloq Despl"	"Bloc Scorr"
72	48h	"Pause"	"Pause"	"Pausa"	"Pausa"
73	49h	"Inser"	"Einfg"	"Insert"	"Ins"
74	4Ah	"HOME"	"Pos1"	"Inicio"	"?"
75	4Bh	"Page Up"	"Bild "	"Re Pag"	"Pag "
76	4Ch	"Suppr"	"Entf"	"Supr"	"Canc"
77	4Dh	"Fin"	"Ende"	"Fin"	"Fine"
78	4Eh	"Page Down"	"Bild "	"Av Pag"	"Pag "
79	4Fh	" "	" "	" "	" "
81	51h	" "	" "	" "	" "
82	52h	" "	" "	" "	" "
83	53h	["Verr Num"]	["Num Lock"]	["Bloq Num"]	["Broc Num"]
84	54h	["/"]	["/"]	["/"]	["/"]
85	55h	["*"]	["*"]	["*"]	["*"]
86	56h	["-"]	["-"]	["-"]	["-"]
87	57h	["+"]	["+"]	["+"]	["+"]
88	58h	["Entr"]	["Enter"]	["Intro"]	["Invio"]
89	59h	["1", "Fin"]	["1", "Ende"]	["1", "Fin"]	["1", "Fine"]
90	5Ah	["2", " "]	["2", " "]	["2", " "]	["2", " "]
91	5Bh	["3", "Page Down"]	["3", "Bild "]	["3", "AvPag"]	["3", "Pag "]
92	5Ch	["4", " "]	["4", " "]	["4", " "]	["4", " "]
93	5Dh	["5"]	["5"]	["5"]	["5"]

Code	Code (hexadecimal)	France 105 Keyboard	Germany 105 Keyboard	Spain 105 Keyboard	Italy 105 Keyboard
94	5Eh	["6", " "]	["6", " "]	["6", " "]	["6", " "]
95	5Fh	["7", "?"]	["7", "Pos1"]	["7", "Inicio"]	["7", "?"]
96	60h	["8", " "]	["8", " "]	["8", " "]	["8", " "]
97	61h	["9", "Page Up"]	["9", "Bild "]	["9", "RePag"]	["9", "Pag "]
98	62h	["0", "Inser"]	["0", "Einfg"]	["0", "Ins"]	["0", "Ins"]
99	63h	[".", "Suppr"]	["", "Entf"]	[".", "Supr"]	[".", "Canc"]
100	64h	"<", ">"	"<", ">", " "	"<", ">"	"<", ">"
101	65h	"S3"	"S3"	"S3"	"S3"
135	87h	Not Used	Not Used	Not Used	Not Used
136	88h	Not Used	Not Used	Not Used	Not Used
137	89h	Not Used	Not Used	Not Used	Not Used
138	8Ah	Not Used	Not Used	Not Used	Not Used
139	8Bh	Not Used	Not Used	Not Used	Not Used

LED を持つキーボードは、現在のところ予定されておりません。

6.8.6 キーボードのハードウェア情報

● PDS_KEYBOARDINFO

```
typedef struct {
    Uint8 lang;
    Uint8 type;
    Uint8 led;
    Uint8 led_ctrl;
} PDS_KEYBOARDINFO;
```

メンバ lang には、キーボード言語種別が格納されます。次のように定義されています。

定 義	意 味
PDD_KBDLANG_JP	日本
PDD_KBDLANG_US	米国
PDD_KBDLANG_UK	英国
PDD_KBDLANG_GERMANY	ドイツ
PDD_KBDLANG_FRANCE	フランス
PDD_KBDLANG_ITALY	イタリア
PDD_KBDLANG_SPAIN	スペイン
PDD_KBDLANG_SWEDEN	スウェーデン
PDD_KBDLANG_SWITZER	スイス
PDD_KBDLANG_NETHER	オランダ
PDD_KBDLANG_PORTUGAL	ポルトガル
PDD_KBDLANG_LATIN	ラテンアメリカ
PDD_KBDLANG_CANFRENCH	カナディアンフレンチ
PDD_KBDLANG_RUSSIA	ロシア
PDD_KBDLANG_CHINA	中国
PDD_KBDLANG_KOREA	韓国

メンバ type には、キーボードタイプが格納されています。次のように定義されています。

定 義	意 味
PDD_KBDTYPE_89	89 キー
PDD_KBDTYPE_92	92 キー
PDD_KBDTYPE_101	101 キー
PDD_KBDTYPE_102	102 キー
PDD_KBDTYPE_104	104 キー
PDD_KBDTYPE_105	105 キー
PDD_KBDTYPE_106	106 キー
PDD_KBDTYPE_109	109 キー
PDD_KBDTYPE_87	87 キー
PDD_KBDTYPE_88	88 キー

メンバ led には、そのキーボードが LED を持っているかどうかビット毎に格納されています。1 のビットは LED あり、0 のビットは LED なしをあらわしています。

次のように定義されています。

定 義	意 味
PDD_LED_SHIFT	シフト
PDD_LED_POWER	電源
PDD_LED_KANA	カナ
PDD_LED_SCRLOCK	Scroll Lock
PDD_LED_CAPLOCK	Caps Lock

PDD_LED_NUMLOCK	Num Lock
-----------------	----------

メンバ `led_ctrl` には、そのキーボードが LED を持っている場合は、LED の点灯や消灯の制御をホスト (Dreamcast) が行うか、キーボード自身が行うかの情報が格納されています。ただし、現状のライブラリでは LED の制御には対応していません。

制御方式は、次のように定義されています。

定 義	意 味
PDD_KBDCTRL_HOST	ホスト (Dreamcast) で制御
PDD_KBDCTRL_KEYBOARD	キーボード自体が制御

注 意 ヨーロッパ系言語キーボードに対応するアプリケーションについては、特に注意が必要となります。

6.8.7 サンプルプログラム

```
Sint32 get_key(void)
{
    PDS_KEYBOARD* kbd;
    Sint32 i;
    /* ポート A のキーボードデータを取得 */
    kbd = pdKbdGetData(PDD_PORT_A0);

    /* キーボードが接続されていたら */
    /* 押されてるキーが表示されている */
    if (kbd != NULL) {
        for (i = 0; i < 6; i++) {
            njPrintH(NJM_LOCATION(i * 4, 10), kbd->key[i]);
        }
    }

    return NJD_USER_CONTINUE;
}
```

6.8.8 関数一覧

キーボードデバイスライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
<code>pdKbdGetData</code>	キーボードデータを取得
<code>pdKbdInit</code>	キーボードライブラリの初期化
<code>pdKbdExit</code>	キーボードライブラリの終了
<code>pdKbdGetInfo</code>	キーボードのハードウェア情報の取得

6.9 マウスライブラリ (pdMouse) について

コントロールポートに接続されたマウスを制御するためのライブラリです。

ノート マウスは、ポインティングデバイス (PDD_DEVTYPE_POINTING) です。

マウスライブラリを使用すると、

- マウスの接続確認をする。
- マウスの移動量を取得する。
- マウスのボタン情報を取得する。

といった処理が実行可能です。

6.9.1 定数

- マウスの持つボタンのビットアサイン

定数名	説 明
PDD_MOUSE_BUTTON_M	中央ボタン
PDD_MOUSE_BUTTON_R	右ボタン
PDD_MOUSE_BUTTON_L	左ボタン
PDD_MOUSE_BUTTON_S	サブボタン (海外向けのマウス用)
PDD_MOUSE_BUTTON_KU	十字キー上
PDD_MOUSE_BUTTON_KD	十字キー下
PDD_MOUSE_BUTTON_KL	十字キー左
PDD_MOUSE_BUTTON_KR	十字キー右
PDD_MOUSE_BUTTON_WU	ホイール上回転
PDD_MOUSE_BUTTON_WD	ホイール下回転

ノート ホイールが回転した情報については、ボタン情報のビット 16 及びビット 17 に情報を返します。

- 座標軸のビットアサイン

定数名	説 明
PDD_MOUSE_AXIS_X	X 軸
PDD_MOUSE_AXIS_Y	Y 軸
PDD_MOUSE_AXIS_Z	Z 軸 (ホイール)
PDD_MOUSE_AXIS_3	第 3 軸
PDD_MOUSE_AXIS_4	第 4 軸
PDD_MOUSE_AXIS_5	第 5 軸
PDD_MOUSE_AXIS_6	第 6 軸
PDD_MOUSE_AXIS_7	第 7 軸

6.9.2 構造体仕様

- PDS_MOUSE_BUTTON 構造体

マウスのボタン情報と、接続 / 未接続状態を格納する構造体です。

```
typedef STRUCT {
    Uint32      on ;
    Uint32      off ;
    Uint32      press ;
    Uint32      release ;
    Uint32      old ;
    Uint8       connect ;
    Uint8       opt ;
    Uint8       reserved[ 2 ] ;
} PDS_MOUSE_BUTTON
```

メンバ	意 味
on	マウスのボタン状態
off	マウスのボタン状態
press	マウスのボタン状態
release	マウスのボタン状態
old	予約
connect	マウスの接続状態 1 で接続、0 で未接続
opt	オプションパラメータ（詳細未定）
reserved[2]	予約

ノート on、off、press、release の各メンバは、マウスのボタンの状態を示すもので、押されているボタンに対応するビットが1になります。

- PDS_MOUSE_AXIS 構造体

マウスの軸情報（移動量）を格納する構造体です。

```
typedef STRUCT {
    Uint8       over ;
    Uint8       reserved ;
    Sint16      x ;
    Sint16      y ;
    Sint16      z ;
    Sint16      axis[ 5 ] ;
} PDS_MOUSE_AXIS
```

メンバ	意 味
over	オーバーフローフラグ
reserved	予約
x	X 方向の移動量（-512～511）
y	Y 方向の移動量（-512～511）
z	ホイールの移動量（-512～511）
axis[5]	その他の軸の移動量（-512～511）

ノート オーバーフローフラグは、オーバーフローが発生した軸に対応するビットが1になります。1 の場合には、その軸のデータは無効です。

- PDS_MOUSE_INFO 構造体
マウスの情報を格納する構造体です。

```
typedef STRUCT {
    Uint8    button ;
    Uint8    axis ;
} PDS_MOUSE_INFO
```

メンバ	意 味
button	ボタンの有無
axis	軸の有無

ノート ボタンの有無は、対応するビットが1の場合に、マウスがそのボタンを備えていることを示します。
軸の有無は、対応するビットが1の場合に、マウスがその軸を備えていることを示します。

6.9.3 サンプルプログラム

以下に、サンプルプログラムを示します。

```
#define MOUSE_PORT    PDD_PORT_D0

Sint32 gMouseX;
Sint32 gMouseY;

/* ユーザーV 割り込み関数(あらかじめVに登録しておく) */
void user_vsync(void)
{
    PDS_MOUSE_AXIS* pa;

    pa = pdMouseGetAxis(MOUSE_PORT);
    if (!(pa->over & PDD_MOUSE_AXIS_X)) gMouseX += pa->x;
    if (!(pa->over & PDD_MOUSE_AXIS_Y)) gMouseY += pa->y;
    if (gMouseX < 0)    gMouseX = 0;
    if (gMouseY < 0)    gMouseY = 0;
    if (gMouseX > 639) gMouseX = 639;
    if (gMouseY > 479) gMouseY = 479;
}

Sint32 Usermain(void)
{
    PDS_MOUSE_BUTTON* pb;

    if (pdMouseInitIsReady(MOUSE_PORT) {
        pd = pdMouseGetButton(MOUSE_PORT);
        if (pd > press & PDD_Mouse_Button_L) {
            /* 左ボタンがクリックされた */
        }
    }
}
```

6.9.4 関数一覧

マウスライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
pdMouseInit	ライブラリの初期化
pdMouseExit	ライブラリの終了処理
pdMouseGetButton	ボタン情報の取得
pdMouseGetAxis	座標軸情報の取得
pdMouseGetInfo	マウス情報の取得
pdMouseIsReady	マウスの接続確認
pdMouseExecServer	サーバ関数

6.10 記録デバイスライブラリ (bu) について

メモ리카ード (ビジュアルメモリ) へのファイルの保存・読み込みを行います。

6.10.1 セーブ中のコントローラ及びメモリーカードの抜き差し処理

セーブ中のメモ리카ードの抜き差しに関する作成基準は、以下のとおりです。

作成基準「15.4 セーブ」

必須：セーブ中は、オートセーブの場合をのぞき、画面上にセーブ中であることを示す表記を行うこと。

注 意 セーブ中にメモリーカードを抜き差しして生じるトラブルはユーザー責任であり、注意を促す以外に回避策はありません。

(1) メモリーカードの抜き差しによるファイル破損について

セーブ中のメモリーカードの抜き差しによってファイルが破損した場合、不完全なファイルがメモリーカードに残ってしまいます (空きエリアは減少する)。

このファイルを本体が読み出した場合には、ロード関数 buLoadFile/Ex がエラーを検出しアプリケーションに通知するため、ファイルが破損していることをチェックすることが可能です。

またロード関数がエラーを検出できない場合においても、セーブ時に buMakeBackupFileImage 関数を用いて CRC コードを付加している場合には解析関数 buAnalyzeBackupFileImage によってエラーを検出可能です。

また、メモリーカードに十分な空きエリアが存在する場合は、次に不完全なファイルと同名のファイルでも正しくファイルを保存することができます。

6.10.2 セーブ中のディスクアのオープン、及びソフトリセット時の処理

ファイルのセーブ中は、データ保護のため、通常の実成基準の例外が適用されます。

作成基準「4.2.3 セーブ実行中にディスクアが開かれた場合の例外事項」

必須：セーブ実行中にディスクアが開かれた場合は、セーブが終了するまで待ち、終了を確認したらすぐに「ドリームキャスト起動直後のメインメニュー」へ制御を移すこと。

GDFS のエラーコールバック関数内で、ディスクアが開かれたかどうかを調べ、フラグとしてグローバル変数等に保存しておきます。そして、セーブ終了後にこのフラグを調べ、ディスクアが開かれていることを検出していれば、本体 BOOT ROM のメイン画面に進んで下さい。

GDFS のエラーコールバック関数は V 割り込み内で発生します。この関数中ではペリフェラル割り込みが取得できず、バックアップライブラリの処理が続行できません。従って GDFS のエラーコールバック関数内に、セーブ終了を待ったり、バックアップライブラリの終了関数が成功するまで繰り返すようなコードを書いた場合、無限ループとなる場合がありますので、こういったコーディングは避けるようにして下さい。

作成基準「15.4 セーブ」

必須：セーブ中は、「ソフトウェアリセット」の実行を無効にすること。

同様にセーブ中はソフトリセットコマンドの入力を検出しても、その時点ではソフトリセットの処理を行わず、フラグを立てるのみとし、セーブが終了した時点で改めて処理を行って下さい。

表 6 - 1 主なペリフェラルのペリフェラル情報

ペリフェラル名	type	support	備 考
未接続	0	-	
ドリームキャスト・コントローラ	CONTROLLER	000F06FE	本体付属コントローラ
レーシングコントローラ	CONTROLLER	000700FE	
アーケードスティック	CONTROLLER	000007FF	
ドリームキャスト・キーボード	KEYBOARD	00000000	
ドリームキャスト・ガン	CONTROLLER LIGHT_GUN	000000FE	コントローラデバイスとガンデバイスの2つのタイプを併せ持つ
つりコントローラ	CONTROLLER	003F06FE	拡張ソケット 5 に振動デバイス内蔵
Ascii Mission Stick	CONTROLLER	000F06FE	ドリームキャスト・コントローラと完全互換
ビジュアルメモリ	STORAGE LCD TIMER	-	
ぶるぶるぱっく	VIBRATION	-	
マイクデバイス	SOUNDINPUT	-	
ツインスティック	CONTROLLER	0000FEFE	
マウス	POINTING	00000000	
マラカスコントローラ	CONTROLLER	003C090F	
ドリームアイ	CONTROLLER	00000000	
ドリームアイ用マイクデバイス	SOUNDINPUT	00000000	ドリームアイ付属のマイクデバイス

- ペリフェラルの判別の目的

ペリフェラルの判別は、例えば接続されているペリフェラルによってキーアサイン画面のグラフィックを変更したり、使いやすいキーアサインを自動選択するなどの積極的な目的にのみ用いて下さい。特定コントローラ以外は、使用できないようにするなどの消極的な目的に対しては用いないで下さい。

6.10.3 ペリフェラルの同時使用制限

Dreamcast では1つのポートに、拡張ソケットやペリフェラルを含めて複数のペリフェラルが接続できます。

ただし、接続するペリフェラルの組み合わせによっては、同時に使用することが制限される場合があります。

以下に具体的な組み合わせを挙げて説明します。

- ガンデバイスとビジュアルメモリ（記録、LCD、タイマーデバイス）

ガンデバイスをガンモードに設定している場合は、ビジュアルメモリに代表される記録デバイス、LCD デバイス、タイマーデバイスにはアクセス及び実行ができません。

注 意 特に記録デバイスの扱いについては、メモリーカードライブラリの終了を行ってからガンモードに移行する必要があります。

- ガンデバイスとマイクデバイス

ガンデバイスをガンモードにしている場合は、マイクデバイスの使用はできません。

- ガンデバイスと振動デバイス

ガンデバイスをガンモードにしている場合でも、振動デバイスを使用することは可能です。

- 振動デバイスとマイクデバイス

振動デバイスが振動中でも、マイクデバイスの使用は可能です。

注 意 振動デバイスの振動音を、マイクが拾う可能性がありますので注意して下さい。

- 振動デバイスとビジュアルメモリ

振動デバイスの振動中に、ビジュアルメモリへのデータの保存を含むすべてのデバイスへのアクセスは可能です。

- マイクデバイスとビジュアルメモリ

マイクデバイスとビジュアルメモリを含めた、すべてのデバイスとの同時利用は問題ありません。

- 振動デバイスの電流チェック

作成基準「18.1.2 振動ペリフェラルの電流チェック」の必須事項「振動デバイスへの対応数量は以下の規定に従うこと」については、以下の制限を設けることにより対応して下さい。

- 振動デバイスの同時動作は最大4台まで。
- 「つりコントローラ」のように拡張ソケット#5の振動デバイスが存在する場合は、他の同時動作は1台まで。

- 例

つりコントローラ 2 台	O K
つりコントローラ 1 台 + ぷるぷるパック 1 台	O K
つりコントローラ 1 台 + ぷるぷるパック 2 台	N G
つりコントローラ 2 台 + ぷるぷるパック 1 台	N G

6.10.4 ペリフェラルデータの取得タイミング

pdGetPeripheal () 関数では、実際にコントローラデバイスからのデータが得られてからアプリケーションに反映されるまでに、およそ 1 V のレイテンシ (遅延) が発生します。

このレイテンシは、通常のアプリケーションでは全く意識する必要はありませんが、通信対戦などを行うアプリケーションの場合に、できる限り早くコントローラのデータを取得し、送信処理を行う必要があります。

このため、低レイテンシペリフェラルデータ取得関数 pdGetPeripheralDirect が追加されています。ここでは、通常の pdGetPeripheal とのデータを取得するタイミングの比較を示します。

- 通常 : pdGetPeripheral



図 6 - 1 ペリフェラルデータの取得タイミング (通常)

- 毎 V (syChain に登録) で Maple DMA を開始
 - DMA 終了割り込み (通常同じ V で終了) でデータの取得
 - 次に呼ばれる njWaitVSync のペリフェラルサーバでデータの加工
 - pdGetPeripheral でデータの取得
- 低レイテンシ : pdSetIntFunction/pdGetPeripheralDirect

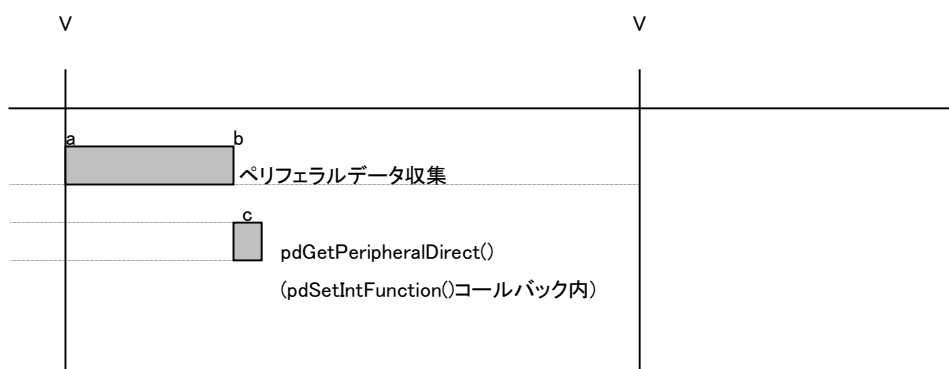


図 6 - 2 ペリフェラルデータの取得タイミング (低レイテンシ)

- 毎 V (syChain に登録) で Maple DMA を開始
- DMA 終了割り込み (通常同じ V で終了) でデータの取得
- DMA 終了割り込みのコールバック関数内 (pdSetIntFunction で登録した関数内) で、pdGetPeripheralDirect を呼び出すことによりデータの取得

従って pdGetPeripheralDirect では、同一の V で入力されたデータを同じ V ですることが可能になります。

アプリケーションは2Vで動作し、コントローラ入力を1Vで取得したいような場合にも利用できます。しかしその場合には、ボタンアップ・ボタンダウンのエッジ（押された瞬間や離された瞬間）を取りこぼさないような工夫が必要です。

6.10.5 コントロールポートについて

Dreamcast ではコントロールポート A～D の拡張ソケットを持つペリフェラルに対し、それぞれ最大2個までのメモリーカードが接続可能です。物理的には、合計で最大8個までのメモリーカードが接続できます。

初期化時に指定した任意の記録デバイスにアクセス可能です。

ノート 以後、各記録デバイスを便宜上、"ドライブ"と呼んで説明する場合があります。

ドライブ番号	定数	接続場所
0	BUD_DRIVE_A1	コントロールポート1の拡張ソケット1に接続された記録デバイス
1	BUD_DRIVE_A2	コントロールポート1の拡張ソケット2に接続された記録デバイス
2	BUD_DRIVE_B1	コントロールポート2の拡張ソケット1に接続された記録デバイス
3	BUD_DRIVE_B2	コントロールポート2の拡張ソケット2に接続された記録デバイス
4	BUD_DRIVE_C1	コントロールポート3の拡張ソケット1に接続された記録デバイス
5	BUD_DRIVE_C2	コントロールポート3の拡張ソケット2に接続された記録デバイス
6	BUD_DRIVE_D1	コントロールポート4の拡張ソケット1に接続された記録デバイス
7	BUD_DRIVE_D2	コントロールポート4の拡張ソケット2に接続された記録デバイス

6.10.6 メモリーカードの容量

このライブラリでは、初期化時に指定した最大容量までのメモリーカードに対応します。記録デバイスのアクセスはすべてブロック単位で行われ、1ブロックは512バイトです。現在、次の記録デバイスを持つペリフェラルが発売されています。

容量	名称	最大ファイル数（最大ブロック数）
128KB	VM（ビジュアルメモリ）	200

ノート 現在発売されているメモリーカードは、ビジュアルメモリのみです。

6.10.7 関数種別

記録デバイスを取り扱う関数には、TYPE A と TYPE B の 2 種類の関数があり、以下のような特徴があります。

タイプ	説 明
TYPE A	<ul style="list-style-type: none">・ 即時完了復帰するタイプです。buStat()によってステータスを取得する必要はなく、コールバックありません。・ このタイプの関数の終了直後には、他のファイルシステム関数を呼び出すことができます。・ このタイプの関数は主に、記録デバイス接続時にキャッシュされる情報を参照するだけのものが多いため、即時完了復帰が可能です。
TYPE B	<ul style="list-style-type: none">・ 即時復帰しますが、その時点で処理は終了しておらず、buStat()によってステータスを毎フレーム取得して終了を待つ必要があります。・ ステータスが終了を示さない限り、同じドライブに対してファイルシステム関数を呼び出すことはできません。呼び出した場合はエラーを返します。・ 処理終了時に指定の完了コールバック関数をコールバックします。・ 処理中に経過コールバック関数をコールバックします。・ このタイプの関数の返値は、BUD_ERR_OK または BUD_ERR_BUSY です。・ 引数で示されるアドレスにデータを取得する関数（buLoadFile()など）の場合、ステータスが完了を示すまでは有効データが格納されないので注意して下さい。・ 引数で示されるアドレスのデータを記録デバイスに書き込む関数（buSaveFile()等）の場合、ステータスが完了を示すまでデータの内容を保証して下さい。・ buGetLastError()により、最後に発生したエラーを知ることができます。

6.10.8 一般アプリケーションへの組み込み

このライブラリには多くの関数が用意されていますが、それらのほとんどは BOOT ROM や一部のファイルユーティリティ的な機能を含むアプリケーションのみが使用することを想定されています。一般的なゲームアプリケーションの場合であれば、データのロード・セーブの状態取得などの一部の関数を使用するだけで、記録デバイスへのファイルアクセスが容易に行えます。

(1) バックアップライブラリの組み込み

a. バックアップライブラリの初期化

バックアップライブラリの初期化には、ライブラリのワークエリアの先頭アドレス、対応するドライブ数、最大容量、そして初期化完了コールバック関数が必要になります。

```
#include <shinobi.h>

#define MAX_DRIVES 8
Uint32 bupWork[BUM_WORK_SIZE(BUD_CAPACITY_128KB, MAX_DRIVES) / sizeof(Uint32)];
static void init_callback(void)
{
}

void BackupInit(void)
{
    buInit(BUD_CAPACITY_128KB, BUD_USE_DRIVE_ALL, bupWork, init_callback);
}

void BackupExit(void)
{
    buExit();
}
```

この例では、8 個すべてのドライブに対応し、それぞれ 128 KB までの記録デバイスに対応するよう初期化しています。コントローラ A に接続された記録デバイス（2 個）のみに対応する場合は、以下のようにします。

```
#define MAX_DRIVES 2
void BackupInit(void)
{
    buInit(BUD_CAPACITY_128KB, BUD_USE_DRIVE_A1 | BUD_USE_DRIVE_A2,
    bupWork, init_callback);
}
```

初期化時に指定されなかったドライブには、記録デバイスが接続されても認識しません。

注 意 ユーザーフレンドリーという観点から、特に理由のない限りすべてのドライブに対応するようにして下さい。

b. コールバック関数の登録

必要であれば、ファイルアクセス等の処理完了時、経過時のコールバック関数を登録することが可能です。それには以下のようにします。

```
Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 stat, Uint32 param)
{
    return BUD_CBRET_OK;
}

Sint32 progress_callback(Sint32 drive, Sint32 op, Sint32 count, Sint32 max)
{
    return BUD_CBRET_OK;
}

static void init_callback(void)
{
    buSetCompleteCallback(complete_callback);
    buSetProgressCallback(progress_callback);
}
```

例のように、処理完了時に呼び出したい関数を buSetCompleteCallback()関数で、一定処理経過時に呼び出したい関数を buSetProgressCallback()関数で、それぞれ登録します。登録は初期化コールバック関数で行って下さい。

上記の例では、コールバック関数では何もせずにただ BUD_CBRET_OK を返しています。

コールバック関数を利用することにより、ロード、セーブの終了や記録デバイスの接続、取り外しを容易に検出することができます。また、長時間かかる処理の進行具合も知ることができます。

コールバック関数の詳細については後述します。

c. 組み込みに関する関数

関 数	機 能	タイプ
buInit	ライブラリを初期化	A
buExit	ライブラリを終了	A
buSetCompleteCallback	処理完了時のコールバック関数を登録	A
buSetProgressCallback	一定処理経過時のコールバック関数を登録	A
BU_INITCALLBACK	初期化終了時のコールバック関数型	-
BU_COMPLETECALLBACK	処理完了時のコールバック関数型	-
BU_PROGRESSCALLBACK	一定処理経過時のコールバック関数型	-

6.10.9 一般アプリケーションでのファイルアクセス

このライブラリには多くの関数が用意されていますが、それらのほとんどは BOOT ROM や一部のファイルユーティリティ的な機能を含むアプリケーションのみが、使用することを想定されています。

一般的なゲームのアプリケーションの場合であれば、データのロード、セーブ、状態取得など一部の関数を使用するだけで、記録デバイスへのファイルアクセスが容易に行えます。

(1) 記録デバイスの接続確認

記録デバイスが接続されているかどうかを調べるには、buIsReady()関数を使用します。

```
if (buIsReady(BUD_DRIVE_A1)) {
    /* 記録デバイスが接続されている */
}
```

(2) 記録デバイスの情報取得

a. フォーマット状態 (buIsFormatDisk())

記録デバイスがフォーマットされているかどうかを調べるには、buIsFormatDisk()関数を使用します。

```
switch (buIsFormatDisk(BUD_DRIVE_A1)) {
    case BUD_ERR_OK:
        /* フォーマットされている */
        break;
    case BUD_ERR_NO_DISK:
        /* 記録デバイスが接続されていない */
        break;
    case BUD_ERR_BUSY:
        /* 記録デバイスは BUSY 状態のため調べられない */
        break;
}
```

注 意 フォーマットされていない場合、フォーマット以外の処理をすることはできません。

b. 空き容量 (buGetDiskFree())

バックアップの空き容量を調べるには、buGetDiskFree()関数を使用します。

```
Sint32 free;
free = buGetDiskFree(BUD_DRIVE_A1, BUD_FILETYPE_NORMAL);
if (free < 0) return NG;
```

空き容量の取得に成功すれば、free には 0 以上の値が返されます。これはその記録デバイスの空きブロック数 (1 ブロックは 512 バイト) となります。

エラーが発生した場合には、負の値が返されます。

c. 最大容量その他の情報 (buGetDiskInfo())

バックアップの情報は、buGetDiskInfo()関数を使用しても調べることができます。

BUS_DISKINFO 構造体の詳細は構造体リファレンスを参照して下さい。

```
BUS_DISKINFO info;
Sint32 ret;
ret = buGetDiskInfo(BUD_DRIVE_A1, &info);
if (ret < 0) return NG;
```

d. ファイルの存在確認

ファイルアクセスの前に、指定のファイルが存在するかどうかを調べることができます。

```
Sint32 ret;
ret = buIsExistFile(BUD_DRIVE_A1, "SAVEDATA_001");
switch (ret) {
    case BUD_ERR_OK:
        /* ファイルがある */
        break;
    case BUD_ERR_FILE_NOT_FOUND:
        /* ファイルなし */
        break;
    default:
        /* その他のエラー */
        break;
}
```

e. 記録デバイスのアクセス状態

記録デバイスにアクセス中（何らかのファイルアクセスやフォーマット等の処理中）かどうかを調べるには、次のように設定します。

```
if (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
    /* 処理中 */
}
```

処理中である場合には、そのドライブに対して他の処理をすることはできません。他の処理を行う関数を呼び出した場合はエラーが返ります。

(3) ファイルのセーブ

a. セーブのリクエスト

ファイルをセーブするには、buSaveFile()関数を使用します。

```
Sint32 ret;
Sint32 blocks, flag;
BUS_TIME time;
extern UInt8 SaveData[];          /* セーブするデータ(512*10 バイト) */

blocks = 10;                      /* ファイルサイズは10ブロック */
flag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0x00);
/* ベリファイを行う | コピーフラグを 00H に設定する */
/* タイムスタンプの設定 */
/* 1998/6/25(木) 23:59:59 */
time.year = 1998;
time.month = 6;
time.day = 25;
time.hour = 23;
time.minute = 59;
time.second = 59;
time.dayofweek = 4;

ret = buSaveFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, blocks, &time, flag);

if (ret == BUD_ERR_OK) {
    /* セーブリクエストに成功した */
} else {
    /* セーブリクエストに失敗した(BUSY) */
}
```

b. セーブの終了判定とエラーチェック

ファイルセーブが完了したかどうかは、buStat()関数を使用して調べます。

```
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
    /* 処理中 */
}
```

セーブ時にエラーが発生したかどうかは、関数 buGetLastError() を使用して調べます。

```
if (buGetLastError(BUD_DRIVE_A1) == BUD_ERR_OK) {
    /* セーブ成功 */
} else {
    /* エラー発生 */
}
```

c. セーブに要する時間

ファイルセーブに要する時間は、ビジュアルメモリの場合ほぼ以下の式で計算できます。

$$\text{time(フレーム数)} = (\text{ブロック数} + 2) * 5$$

上記の例では、10 ブロックセーブしていますので、約 60 フレーム (1 秒) が掛かることになります。

ただし、この時間はあくまでも予想時間であり、実際には他のドライブのアクセスの有無やエラーの有無により変化することがあります。また、ベリファイを行う場合はデータの読み出し処理を行うため、セーブ時間は多少長くなります。

(4) ファイルのロード

a. ロードのリクエスト (buLoadFile())

ファイルをロードするには、buLoadFile()関数を使用します。

```
Sint32 ret;
Sint32 blocks;
extern UInt8 SaveData[]; /* ロードバッファ */

blocks = buGetFileSize(BUD_DRIVE_A1, "SAVEDATA"); /* ファイルサイズを取得 */

if (blocks <= 0) return NG;
ret = buLoadFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, blocks);

if (ret == BUD_ERR_OK) {
    /* ロードリクエストに成功した */
} else {
    /* ロードリクエストに失敗した(BUSY) */
}
```

b. ロードの終了判定とエラーチェック (buStat(), buGetLastError())

ファイルのロードが完了したかどうかのチェックは、buStat()関数を使用して調べます。

```
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
    /* 処理中 */
}
```

ロード時にエラーが発生したかどうかについては、buGetLastError()関数を使用して調べます。

```
if (buGetLastError(BUD_DRIVE_A1) == BUD_ERR_OK) {  
    /* ロード成功 */  
} else {  
    /* エラー発生 */  
}
```

c. ロードに要する時間

ファイルロードに要する時間の取得は、ビジュアルメモリの場合についてはほぼ以下の式で計算できます。

time(フレーム数) = ブロック数

上記の例では、10 ブロックロードしていますので、約 10 フレーム (167 ミリ秒) 掛かることになります。

ただし、この時間はあくまでも予想時間であり、実際には他のドライブのアクセスの有無や、エラーの有無により変化することがあります。

(5) ファイルの削除

a. 削除のリクエスト (buDeleteFile())

ファイルを削除するには、buDeleteFile()関数を使用します。

```
Sint32 ret;  
  
ret = buDeleteFile(BUD_DRIVE_A1, "SAVEDATA");  
  
if (ret == BUD_ERR_OK) {  
    /* 削除リクエストに成功した */  
} else {  
    /* 削除リクエストに失敗した (BUSY) */  
}
```

b. 削除の終了判定 (buStat())

ファイル削除が完了したかどうかは、buStat()関数を使用して調べます。

```
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {  
    /* 処理中 */  
}
```

削除時にエラーが発生したかどうかは、関数 buGetLastError()を使用して調べます。

```
if (buGetLastError(BUD_DRIVE_A1) == BUD_ERR_OK) {  
    /* 削除成功 */  
} else {  
    /* エラー発生 */  
}
```

c. (ビジュアルメモリでの) 削除に要する時間

ファイル削除に要する時間は、ビジュアルメモリの場合はほぼ5フレームになります。

ただし、この時間はあくまでも予想時間であり、実際には他のドライブのアクセスの有無、エラーの有無により変化することがあります。

(6) コールバック関数

ロードやセーブをはじめとして、時間を要する処理のバックグラウンド処理を容易にするために、コールバックをサポートしています。

a. 完了コールバック

完了コールバックはロードやセーブなどの処理が完了した時点で、ユーザーがあらかじめ登録しておいた関数が呼び出されるものです。

完了コールバック関数は、例えば以下のように記述します。

```
Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 stat, Uint32 param)
{
    switch (op) {
        case BUD_OP_MOUNT:
            printf("記録デバイス%d が接続されました。¥n", drive);
            break;
        case BUD_OP_UNMOUNT:
            printf("記録デバイス%d が取り外されました。¥n", drive);
            break;
        case BUD_OP_LOADFILE:
            printf("ロードが終了しました。¥n");
            break;
        case BUD_OP_SAVEFILE:
            printf("セーブが終了しました。¥n");
            break;
        case BUD_OP_DELETEFILE:
            printf("削除が終了しました。¥n");
            break;
    }

    return BUD_CBRET_OK;
}
```

パラメータ drive にはドライブ番号が、op には下表のオペレーションコードが渡されます。オペレーションコードにはこれ以外にも、各関数に対応したものがあります。

オペレーションコード	処 理	対応関数
BUD_OP_MOUNT	記録デバイスの接続	記録デバイスの接続、buRemount()
BUD_OP_UNMOUNT	記録デバイスの切断	記録デバイスの取り外し
BUD_OP_DELETEFILE	ファイルの削除	buDeleteFile()
BUD_OP_LOADFILE	ファイルのロード	buLoadFile()
BUD_OP_SAVEFILE	ファイルのセーブ	buSaveFile()

また、その処理が成功したかどうか、何らかのエラーが発生したかどうかなどの完了ステータスが stat に渡されます。どの処理がどういう完了ステータスをとるのかなどの詳細については関数リファレンスを参照して下さい。

param にはユーザーが指定したパラメータが渡されます。

コールバック関数中で、ファイルシステム関数を呼び出すことができます。これにより、例えば記録デバイスの接続コールバック関数中で、buGetDiskInfo()等の関数を使用して情報を取得したり、ファイルの存在を確認したり、buLoadFile()を使用して自動的にファイルを読み込む等の処理が可能です。

b. 記録デバイスの接続と切断

完了コールバックを利用して、記録デバイスの接続と取り外しを容易に検出することができます。

```
Uint32 gBackupRAMReady[8];

Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 stat, Uint32 param)
{
    switch (op) {
        case BUD_OP_MOUNT:
            if (buIsFormatDisk(drive) == BUD_ERR_OK) {
                gBackupRAMReady[drive] = TRUE;
            }
            break;
        case BUD_OP_UNMOUNT:
            gBackupRAMReady[drive] = FALSE;
            break;
    }

    return BUD_CBRET_OK;
}
```

c. 経過コールバック

経過コールバックを利用すると、どの程度処理が経過したかを容易に知ることができます。

```
Uint32 gProgress[8];

Sint32 progress_callback(Sint32 drive, Sint32 op, Sint32 count, Sint32 max)
{
    gProgress[drive] = count * 100 / max;
    printf("%d %% completed.\n", gProgress[drive]);

    return BUD_CBRET_OK;
}
```

d. コールバックに関する注意事項

- コールバック関数は、Maple 割り込みをトリガとして（割り込みハンドラとして）呼び出されます。複数のプロセス（スレッド）からの呼び出しに対応していないような、他のライブラリ関数やアプリケーション関数を呼び出さないようにして下さい。
- ファイルアクセス中に記録デバイスが取り外された場合、まず現在処理中の処理に対して、完了コールバックが発生します。その際の完了ステータスは BUD_ERR_NO_DISK となります。引き続き同一割り込み中に、バックアップ RAM 切断コールバック（BUD_OP_UNMOUNT）が発生します。
- TYPE B 関数の戻り値が BUD_ERR_BUSY であった場合、コールバックは発生しません。TYPE B 関数の戻り値をチェックせずに、無条件にコールバックを待つようなコーディングは避けて下さい。

e. エラー処理

ファイルアクセスには、予期せぬエラーが発生することがあります。今回記録デバイスがリムーバブルメディアであるため、ロードやセーブ中に記録デバイスが抜かれたりすることが予想されます。

f. 情報取得に関する関数

関 数	機 能	タイプ
buIsReady	記録デバイスが接続されているかどうかを調べる	A
buIsFormat	記録デバイスがフォーマット済みかどうかを調べる	A
buGetDiskFree	空き容量を調べる	A
buGetDiskInfo	記録デバイスの各種情報を取得する	A
buIsExistFile	指定ファイルが存在するかどうか調べる	A
buGetFileSize	指定ファイルのサイズを調べる	A
buGetFileInfo	指定ファイルの各種情報を取得する	A
buStat	処理が完了しているかどうか調べる	A
buGetLastError	最後に発生したエラーを取得する	A

構造体	機 能
BUS_DISKINFO	記録デバイスの情報を格納
BUS_FILEINFO	ファイルの情報を格納
BUS_TIME	タイムスタンプを格納

6.10.10 特殊アプリケーションでのファイルアクセス

ファイル管理ユーティリティのようなアプリケーションの場合、記録デバイスにあるすべてのファイルのファイル名を取得したり、ファイルのコピー・情報の表示等を行う必要があります。ここでは、そのようなアプリケーションのために用意されている関数を解説します。

(1) ディレクトリ情報の取得 (buFindFirstFile()、buFindNextFile())

記録デバイスにあるファイルやファイル名をすべて取得するには、buFindFirstFile()とbuFindNextFile()を使用します。

```
Sint32 ret, files, blocks, total;
char fname[16];
BUS_FILEINFO info;

files = blocks = total = 0;

/* 第1ファイルのファイル名取得 */
ret = buFindFirstFile(drive, fname);
if (ret < 0) {
    if (ret == BUD_ERR_FILE_NOT_FOUND) goto end;
    else goto err;
}
if (buGetFileInfo(drive, fname, &info) < 0) goto err;
blocks = info.blocks;
total += blocks;
files++;

do {
    printf("%12s %10d bytes(%3d blocks)¥n", files, blocks * 512, blocks);
    /* 第2ファイル以降のファイル名取得 */
    ret = buFindNextFile(drive, fname);
    if (ret < 0) {
        if (ret == BUD_ERR_FILE_NOT_FOUND) goto end;
        else goto err;
    }
    if (buGetFileInfo(drive, fname, &info) < 0) goto err;
    blocks = info.blocks;
    total += blocks;
    files++;
} while (files < FILE_MAX);

end:
return OK;
err:
return NG;
```

(2) ファイルのコピー (buLoadFile()、buSaveFile())

基本的には、ファイルをコピーする関数を用意されていません。コピーをする場合は、buLoadFile()と buSaveFile()を使用してコピーする必要があります。

```
Sint32 ret;
Sint32 blocks;
Sint32 saveflag;
extern UInt8 CopyBuffer[]; /* コピーバッファ */
BUS_FILEINFO info;

/* ファイル情報の取得 */
ret = buGetFileInfo(BUD_DRIVE_A1, "SAVEDATA_001", &info);
if (ret < 0) return NG;
blocks = info.blocks;

/* ファイルロード */
ret = buLoadFile(BUD_DRIVE_A1, "SAVEDATA_001", CopyBuffer, blocks);
if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

/* セーブフラグの設定：ベリファイ ON、コピーフラグを 00H とする */
saveflag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0x00);

/* ファイルのセーブ */
ret = buSaveFile(BUD_DRIVE_A2, "SAVEDATA_001", CopyBuffer, blocks,
                 &info.time, saveflag);

if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;
```

上記の例では、ドライブ A1 から B1 にファイルをコピーしています。

実際のアプリケーションでは、以下のような点に注意が必要です。

- a. コピー先の記録デバイスの接続、フォーマットチェック
- b. コピー先の空き容量チェック
- c. コピー先の同名ファイルの存在チェック

また、BOOT ROM 及びビジュアルメモリ単体の、ファイル管理画面でのファイルコピーとの互換性を保つために以下の点に注意して下さい。

- 元のファイルの日付を、コピー先のファイルにも反映させて下さい。
- コピー元のファイルのコピーフラグは、コピー先のファイルには反映されません。(コピー先ファイルのコピーフラグは 00H として下さい)
- コピーフラグが FFH のファイル(コピー不可ファイル)を、コピーができてはなりません。

(3) ファイルの一部読み込み (buLoadFileEx())

全ファイルの特定ブロックだけを読み込み、ヘッダを解析する等の処理が必要になる場合があります。その場合には、ファイルの一部読み込みが可能な buLoadFileEx()関数を使用します。

```
Sint32 ret;
Sint32 blocks;
Sint32 start_block;
extern UInt8 LoadBuffer[]; /* ロードバッファ */

/* ファイル一部ロード */

start_block = 0;
blocks = 1;
ret = buLoadFileEx(BUD_DRIVE_A1, "SAVEDATA_001", LoadBuffer,
start_block, blocks);
if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;
```

この例ではファイルの先頭ブロックを、1ブロックだけ読み込んでいます。

(4) フォーマット (buFormatDisk())

記録デバイスは出荷時にフォーマットされていますが、何らかの場合でフォーマットをする必要が発生した場合には、buFormatDisk()関数を使用してフォーマットできます。

```
Sint32 ret;
BUS_TIME time;
UInt8 volume[32];
Sint32 icon_no;

time.year = 1998; /* タイムスタンプの設定 */
time.month = 6; /* 1998/6/25(木) 23:59:59 */
time.day = 25;
time.hour = 23;
time.minute = 59;
time.second = 59;
time.dayofweek = 4;

icon_no = 0; /* アイコン番号の設定 */

/* ボディカラー青を設定する */
memset(volume, 0, sizeof(volume));
volume[0] = 0x01; /* ボディカラー情報あり */
volume[1] = 0xbf; /* B */
volume[2] = 0x00; /* G */
volume[3] = 0x00; /* R */
volume[4] = 0xff; /* A */

ret = buFormatDisk(BD_DRIVE_A1, volume, icon_no, &time, TRUE);
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;
```

a. アイコン番号

BOOT ROM のファイル管理画面で表示される「デフォルトアイコン」の番号を指定します。

BOOT ROM は、124 種類のデフォルトアイコンを持っていますので、0～123 の値を指定して下さい。また、デフォルトアイコンではなくアプリケーション側で任意のアイコンを保存し、BOOT ROM のファイル管理の画面で表示させることができます。

b. ボディカラー情報

BOOT ROM のファイル管理画面で表示される記録デバイスのボディカラー情報を持たせることができます。これはポリウムデータに以下の形式で指定して下さい。

● ボディカラー情報を持たせる場合

+0	+1	+2	+3	+4	+5	...	+31
01H	BLUE	GREEN	RED	ALPHA	00H	...	00H

ポリウムデータの先頭に 01H を格納して下さい。それ以外は禁止です。

BLUE、GREEN、RED にそれぞれ青、緑、赤の色成分を 0～255 で設定します。

ALPHA（透明度）の値は 128～255 の間で設定して下さい。127 以下の値は BOOT ROM での表示が見づらくなるため指定禁止です。

● ボディカラー情報を持たせない場合

+0	+1	+2	+3	+4	+5	...	+31
00H	00H	00H	00H	00H	00H	...	00H

すべて 00H として下さい。ボディカラー情報を持たせない場合、BOOT ROM のファイル管理画面ではデフォルトカラー（白）が使用されます。

(5) ファイルアクセスに関する関数

関 数	機 能	タイプ
buFindFirstFile	第 1 ファイルのファイル名を取得する	A
buFindNextFile	次のファイルのファイル名を取得する	A
buLoadFileEx	読み込み開始ブロックとブロック数を指定してファイルをロードする	B
buFormatDisk	記録デバイスをフォーマットする	B

6.10.11 実行ファイル

記録デバイスの中でも、ビジュアルメモリのような携帯ゲーム機を兼ねた記録デバイスには、実行ファイルを保存することができます。実行ファイルが保存されたビジュアルメモリは、それ単体で携帯ゲーム機として機能します。

(1) 実行ファイルの特徴及び制限

実行ファイルには、以下の特徴があります。

- 記録デバイスの先頭ブロックから順に、常に連続して格納されます。
- 最大サイズは 128 ブロック（64KB）です。
- 1 つの記録デバイスに対して、1 つしか存在することができません。

(2) 実行ファイルの存在確認

記録デバイスに実行ファイルがあるかどうかは、buFindExecFile()関数を使用して調べます。

```
Sint32 ret;
char fname[16];

/* 実行ファイルのファイル名取得 */
ret = buFindExecFile(BUD_DRIVE_A1, fname);

switch (ret) {
    case BUD_ERR_OK:
        /* 実行ファイルあり。 fname にはファイル名が格納されている。 */
        printf("実行ファイル:%s¥n", fname);
        break;
    case BUD_ERR_FILE_NOT_FOUND:
        printf("実行ファイルはありません。¥n");
        break;
    default:
        printf("エラー¥n");
        break;
}
```

(3) 実行ファイル用空き容量の取得

記録デバイスに実行ファイルを書き込める空き容量があるかどうかは、buGetDiskFree()関数を使用して調べます。

```
Sint32 free;

free = buGetDiskFree(BUD_DRIVE_A1, BUD_FILETYPE_EXECUTABLE);

if (free < 0) return NG;
```

空き容量の取得に成功すれば、free には0以上の値が返されます。これはその記録デバイスの実行ファイルが書き込み可能な空きブロック数となります。

エラーが発生した場合には、負の値が返されます。

(4) 実行ファイル用空き容量の確保

実行ファイルを書き込むためには、先頭ブロックから連続した空きエリアが必要です。

先ほどの buGetDiskFree(drive, BUD_FILETYPE_EXECUTABLE)では、この先頭ブロックからの連続空きブロック数を調べています。

連続空きブロック数が書き込みたい実行ファイルのサイズより少ない場合でも、記録デバイス全体の空き容量が足りていれば、デフラグ処理を行うことによって連続したブロックを確保できる場合があります。デフラグ処理は以下の手順で行って下さい。

- a. buGetDiskFree(drive, BUD_FILETYPE_EXECUTABLE)を使用して、実行ファイルが書き込み可能な連続ブロック数を取得します。
- b. (1)で取得したブロック数が十分であれば、実行ファイルをそのままセーブできます。デフラグ処理を行う必要はありません。
- c. 連続ブロックが足りない場合、buGetDiskFree(drive, BUD_FILETYPE_NORMAL)を使用して、全体の空きブロック数を取得します。
- d. 全体の空きブロック数が足りなければ、デフラグ処理をしても無意味ですので、実行ファイルの保存はできません。
- e. 全体の空きブロック数が足りていれば、デフラグ処理によって連続ブロックを確保することができます。buDefragDisk()を使用してデフラグ処理を行います。


```

Sint32 ret;
Uint32 DefragWork[512 / sizeof(Uint32)];

ret = buDefragDisk(BUD_DRIVE_A1, DefragWork);
if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;

```

注 意 デフラグ処理は記録デバイス全体を書き換える処理です。これにはそれなりの処理時間がかかります。仮にデフラグ中に記録デバイスが抜かれたりすれば、その記録デバイスの内容はほとんどの場合、全て破壊されます。デフラグは、実行ファイル保存時に、上記の手順に従って必要な場合のみ行うようにして下さい。

(5) 実行ファイルのセーブ

実行ファイルをセーブするには、buSaveExecFile()関数を使用します。

```

Sint32 ret;
Sint32 blocks, flag;
BUS_TIME time;
extern Uint8 SaveData[];          /* セーブするデータ(512*10 バイト) */

blocks = 10;                      /* ファイルサイズは10ブロック */
flag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0xff);
/* ベリファイを行う | コピーフラグをFFHに設定する */

time.year = 1998;                 /* タイムスタンプの設定 */
time.month = 6;                   /* 1998/6/25(木) 23:59:59 */
time.day = 25;
time.hour = 23;
time.minute = 59;
time.second = 59;
time.dayofweek = 4;

ret = buSaveExecFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, blocks, &time, flag);

if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;

```

(6) 実行ファイルの部分更新

実行ファイルには、ビジュアルメモリ等の携帯ゲーム機で動作し、データ保存のために自己一部書き換えをすることが考えられます。Dreamcast 本体アプリとの連動のために、本体からも実行ファイルの一部を書き換える必要がある場合があります。

実行ファイルの場合は通常のデータファイルよりも大きいファイルである場合が多く、通常の buSaveExecFile()関数でファイル全体を書き換えるには時間が掛かります。

またセーブ中に取り外された場合は実行ファイルが実行不可能になる場合がほとんどです。

こういった用途のために、実行ファイルに限って一部書き換え可能な関数が用意されています。

```
Sint32 ret;  
ret = buRewriteExecFile(BUD_DRIVE_A1, "GAMEFILE_VM", buf, 15, 2);
```

この例ではファイルの 15 ブロック目から連続した 2 ブロックを、直接 (ファイルの先頭ブロックを 0 ブロックとして) 書き換えています。

(7) 実行ファイルに関する関数

関 数	機 能	タイプ
buFindExecFile	実行ファイルのファイル名を取得する	A
buSaveExecFile	実行ファイルをセーブする	B
buDefragDisk	連続空きブロックを確保する	B
buRewriteExecFile	実行ファイルの一部を直接書き換える	B

6.10.12 バックアップファイルフォーマット

記録デバイスに保存するファイルは、どのようなものでも良いわけではありません。ドリームキャスト・ソフトウェア作成基準に従った、正しい形式のファイルを保存しなくてはなりません。ここでは、バックアップファイルの外部インタフェースと内部フォーマットについて記述します。

参照 ドリームキャスト・ソフトウェア作成基準も合わせて参照して下さい。

(1) ファイルの外部インタフェース

a. ファイル名

ファイル名は必ず 12 文字指定します。ファイル名には半角英大文字、一部記号等、下図の網掛け部のコードが使用可能です。これ以外の文字は指定禁止となります。ライブラリ関数に指定する場合は、NULL 文字を含めて 13 バイトとなります。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	@	P	.	p			-	タ	ミ			
1			!	1	A	Q	a	q			.	ア	チ	ム		
2			"	2	B	R	b	r			「	イ	ツ	メ		
3			#	3	C	S	c	s			」	ウ	テ	モ		
4			\$	4	D	T	d	t			、	エ	ト	ヤ		
5			%	5	E	U	e	u			、	オ	ナ	ユ		
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7			'	7	G	W	g	w			ア	キ	ヌ	ラ		
8			(8	H	X	h	x			イ	ク	ネ	リ		
9)	9	I	Y	i	y			ウ	ケ	ノ	ル		
A			*	:	J	Z	j	z			エ	コ	ハ	レ		
B			+	;	K	[k	{			オ	サ	ヒ	ロ		
C			,	<	L	\	l				ト	シ	フ	ワ		
D			-	=	M]	m	}			ユ	ス	ヘ	ン		
E			.	>	N	^	n	~			ヨ	セ	ホ	"		
F			/	?	O	-	o				ッ	ソ	マ	*		

1つのアプリケーションで複数のファイルを作成する場合は、必ず頭の9文字を統一し、後ろの3文字を変更して下さい。BOOT ROMのファイル管理画面には、同一ゲームのファイルをまとめてコピーする等の機能があります。同一ゲームであるかどうかの判断は、ファイル名の先頭9文字で行っているため、ファイル名が適切でないとこれらの機能が正常に動作しません。

• ファイル名の例

「SAVEDATA_SYS」「SAVEDATA_001」「SAVEDATA_002」等

b. ファイル属性

ファイル属性には、以下のようなものがあります。

• コピーフラグ

コピーフラグとは、00H～FFHまでの値を取るフラグです。ファイル保存時に任意に設定できます。

この値がFFHであるファイルは、BOOT ROMのファイル管理画面及びビジュアルメモリ単体のファイル管理画面でコピーできません。コピー禁止フラグとして使用できます。

コピーフラグ00H～FEHのファイルは、BOOT ROMのファイル管理画面及びビジュアルメモリ単体のファイル管理画面でコピーすると、コピー先ファイルのコピーフラグは00Hになります。

• 実行可能

ビジュアルメモリ等の携帯ゲーム機を兼ねた記録デバイスの場合、実行可能ファイルを1つだけ保存することができます。実行ファイルのサイズは最大128ブロックとなっています。

c. ファイルの内部フォーマット

バックアップファイルには、BOOT ROM のファイル管理画面やビジュアルメモリ単体のファイル管理画面で表示するためのコメント、アイコン等のデータを必ず格納しなくてはなりません。

バックアップファイルは、下のようなフォーマットになっています。なお、ワードデータ、ロングワードデータはリトルエンディアン形式とします。

ファイルフォーマットは通常ファイルと実行可能ファイルでヘッダの位置が異なっているため注意が必要です。

+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F
+0000	VM コメント(16 バイト)														
+0010	BOOT ROM コメント (32 バイト)														
+0030	ゲーム名(ソートアイテム)(16 バイト)														
+0040	アイコン 枚数	アイコン スピード	ビジュア ルタイプ	CRC			セーブデータサイズ			(予約)					
+0050	(予約)														
+0060	アイコンパレットデータ (32 バイト)														
+0080	アイコンピクセルデータ1 (512 バイト)														
+0280	アイコンピクセルデータ2 (512 バイト)														
+0480	アイコンピクセルデータ3 (512 バイト)														
+nnnn	ビジュアルデータ(パレット／ピクセル) (2048/4544/8064 バイト)														
+nnnn	アプリケーションセーブデータ														

固定長領域

可変長領域

ユーザー領域

固定長ヘッダ領域

可変長ヘッダ領域

ユーザー領域

+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F	
+0000	実行ファイル用リセットベクタ、プログラムコード等 (512 バイト)															ユーザー領域 1
+0200	VM コメント(16 バイト)															
+0210	BOOT ROM コメント (32 バイト)															
+0230	ゲーム名(ソートアイテム)(16 バイト)															
+0240	アイコン 枚数	アイコン スピード	ビジュア ルタイプ	CRC	ファイルサイズ	(予約)										固定長ヘッダ領域
+0250	(予約)															
+0260	アイコンパレットデータ (32 バイト)															
+0280	アイコンピクセルデータ1 (512 バイト)															
+0480	アイコンピクセルデータ2 (512 バイト)															可変長ヘッダ領域
+0680	アイコンピクセルデータ3 (512 バイト)															
+nnnn	ビジュアルデータ(パレット/ピクセル) (2048/4544/8064 バイト)															
+nnnn	プログラムコード、データ															ユーザー領域 2

- 固定長ヘッダ（網掛け部）は必須データです。必ず全ての項目を正しく指定する必要があります（CRC、セーブデータサイズを除く）。
- （予約）となっている領域は全て 00H を格納して下さい。
- 固定長ヘッダのうち、ビジュアルメモリ単体のファイル管理画面で参照するのは VM コメントのみとなっています。
- 可変長ヘッダを格納しない場合、アプリケーションセーブデータは+0280H から格納することができます（通常ファイルの場合）。
- ファイルのアクセスは常にブロック単位で行われます。ファイル全体のサイズが 512 バイトの倍数になるようにアプリケーションセーブデータを調整し、ファイルの最終ブロックに不定データがなるべく格納されることのないようにして下さい。
- 固定長ヘッダのサイズが 280H バイトであるため、バックアップファイルは最低 2 ブロックを消費することになります。

d. VM コメント

BOOT ROM とビジュアルメモリ単体のファイル管理画面の両方で表示されるコメントデータです。16 バイト以内で指定して下さい。下図の網掛け部の文字が使用できます。これ以外の文字は指定禁止となります。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	@	P	·	p				-	タ	ミ		
1			!	1	A	Q	a	q			.	ア	チ	ム		
2			"	2	B	R	b	r			「	イ	ツ	メ		
3			#	3	C	S	c	s			」	ウ	テ	モ		
4			\$	4	D	T	d	t			、	エ	ト	ヤ		
5			%	5	E	U	_e	u			・	オ	ナ	ユ		
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7			'	7	G	W	g	w			ア	キ	ヌ	ラ		
8			(8	H	X	h	x			イ	ク	ネ	リ		
9)	9	I	Y	i	y			リ	ケ	ノ	ル		
A			*	:	J	Z	j	z			エ	コ	ハ	レ		
B			+	;	K	[k	{			オ	サ	ヒ	ロ		
C			,	<	L	\	l	!			ト	シ	フ	ワ		
D			-	=	M]	m	}			エ	ス	ヘ	ン		
E			.	>	N	^	n	~			ヨ	セ	ホ	"		
F			/	?	O	-	o				ッ	ソ	マ	*		

e. BOOT ROM コメント

BOOT ROM のファイル管理画面で表示されるコメントデータです。全角文字（2 バイト）、半角文字（1 バイト）混在で、32 バイトまで指定できます。全角文字の場合は英数・かな・カナ・第1、第2水準までの SHIFT-JIS コードで指定します。半角文字の場合は上図の文字が指定できます。

f. ゲーム名（ソートアイテム）

BOOT ROM のファイル管理画面でファイルを表示する際、ソートするためのキーとして使用されます。以下の表を参照して、16 バイトで格納して下さい。ゲーム名が 16 バイトに満たない場合、残りの領域には 00H を格納して下さい。

10 進	16 進	Font
0	0	Space
1	1	ア
2	2	ァ
3	3	イ
4	4	ィ
5	5	ウ
6	6	ヴ
7	7	ゥ
8	8	エ
9	9	ェ
10	0A	オ
11	0B	ォ
12	0C	カ
13	0D	ガ
14	0E	キ
15	0F	ギ
16	10	ク
17	11	グ
18	12	ケ
19	13	ゲ
20	14	コ
21	15	ゴ
22	16	サ
23	17	ザ
24	18	シ
25	19	ジ
26	1A	ス
27	1B	ズ
28	1C	セ
29	1D	ゼ
30	1E	ソ
31	1F	ゾ

10 進	16 進	Font
32	20	タ
33	21	ダ
34	22	チ
35	23	ヂ
36	24	ツ
37	25	ヅ
38	26	ッ
39	27	テ
40	28	デ
41	29	ト
42	2A	ド
43	2B	ナ
44	2C	ニ
45	2D	ヌ
46	2E	ネ
47	2F	ノ
48	30	ハ
49	31	バ
50	32	パ
51	33	ヒ
52	34	ビ
53	35	ピ
54	36	フ
55	37	ブ
56	38	プ
57	39	ヘ
58	3A	ベ
59	3B	ペ
60	3C	ホ
61	3D	ボ
62	3E	ポ
63	3F	マ

10 進	16 進	Font
64	40	ミ
65	41	ム
66	42	メ
67	43	モ
68	44	ヤ
69	45	ャ
70	46	ユ
71	47	ュ
72	48	ヨ
73	49	ョ
74	4A	ラ
75	4B	リ
76	4C	ル
77	4D	レ
78	4E	ロ
79	4F	ワ
80	50	ヰ
81	51	ヱ
82	52	ヲ
83	53	ン
84	54	0
85	55	1
86	56	2
87	57	3
88	58	4
89	59	5
90	5A	6
91	5B	7
92	5C	8
93	5D	9

g. アイコン枚数

格納するアイコンの枚数を 1 ~ 3 で指定します。最低 1 枚のアイコンを必ず格納しなくてはなりません。

h. アイコンアニメーションスピード

2 枚、3 枚のアイコンを格納する場合のパターン切り替えのスピードをフレーム数（1/30 秒単位）で指定します。

i. ビジュアルタイプ

格納するビジュアルコメントのタイプを指定します。

ビジュアルタイプ	格納するデータ
ビジュアルデータなし	0000H
TYPE A	0001H
TYPE B	0002H
TYPE C	0003H

j. CRC コード

ファイル先頭から終わりまでをすべて加味した CRC コードです。ただし、CRC の利用はアプリケーションまかせとなっています。BOOT ROM のファイル管理画面では参照しません。

これは、実行ファイルの場合はビジュアルメモリ単体で動作し、データ保存のため自己書き換えが発生する場合があるためです。

CRC を利用しない場合は 0000H を格納しておきます。後述のユーティリティ関数を利用すると、CRC の生成、チェックが自動で行えます。

k. セーブデータサイズ/ファイルサイズ

アプリケーションのセーブデータ部分のサイズをバイト数で指定します。実行可能ファイルの場合はファイル全体のバイト数を指定して下さい。BOOT ROM のファイル管理画面では参照しません。

l. アイコンデータ (パレットデータ/ピクセルデータ 1、2、3)

BOOT ROM のファイル管理画面で表示されるアイコンです。セーブデータには最低 1 枚のアイコンを必ず格納しなくてはなりません。以下のフォーマットでデータを作成して下さい。

項 目	内 容
サイズ	横 32×縦 32 ピクセル
カラー形式	4 ビット (16 色パレット)
パレット形式	ARGB4444 形式×16 色 2 バイト (16 ビット) で 1 色を表現 リトルエンディアン形式
枚数	最低 1 枚必須、3 枚まで格納可能 パレットは共通
格納方式	パレットデータを 16 色分格納する。続いてピクセルデータを、左上から右下への情報を連続したデータとして格納する。 1 バイトで 2 ピクセルを表す。上位 4 ビットが左のドット、下位 4 ビットが右のドットとなる。
サイズ	1 枚の場合 544 バイト 2 枚の場合 1056 バイト 3 枚の場合 1568 バイト

2 枚、3 枚のアイコンを格納した場合、そのアイコンを指定したフレーム数 (1/30 秒単位) で切り替えて、アニメーション表示されます。

m. ビジュアルコメント

BOOT ROM のファイル管理画面で表示されるビジュアルコメントです。ファイル中にこのビジュアルコメントを格納するかどうかは任意となります。格納する場合は、以下のフォーマットでデータを作成して下さい。

ビジュアルコメントにはピクセルデータ形式によって TYPE A、B、C の 3 種類があり、以下のようなフォーマットになっています。

● ビジュアルコメント TYPE A

項 目	内 容
サイズ	横 72×縦 56 ピクセル
カラー形式	16 ビットダイレクトカラー (ARGB4444)
パレット形式	なし
枚数	1 枚または無し
格納方式	左上から右下へのピクセル情報を連続したデータとして格納する。2 バイト (16 ビット) で 1 ピクセルを表す。リトルエンディアン形式。
サイズ	8064 バイト (16 ブロック)

● ビジュアルコメント TYPE B

項 目	内 容
サイズ	横 72×縦 56 ピクセル
カラー形式	8 ビット (256 色パレット)
パレット形式	ARGB4444 形式×16 色 / (16 ビット) で 1 色を表現 / リトルエンディアン形式
枚数	1 枚または無し
格納方式	パレットデータを 256 色分格納する。続いてピクセルデータを、左上から右下への情報を連続したデータとして格納する。1 ピクセルを 1 バイトで表す。
サイズ	4544 バイト (9 ブロック)

● ビジュアルコメント TYPE C

項 目	内 容
サイズ	横 72×縦 56 ピクセル
カラー形式	4 ビット (16 色パレット)
パレット形式	ARGB4444 形式×16 色 / (16 ビット) で 1 色を表現 / リトルエンディアン形式
枚数	1 枚または無し
格納方式	パレットデータを 16 色分格納する。続いてピクセルデータを、左上から右下への情報を連続したデータとして格納する。 1 バイトで 2 ピクセルを表す。上位 4 ビットが左のドット、下位 4 ビットが右のドットとなる。
サイズ	2048 バイト (4 ブロック)

注 意 ビジュアルコメント TYPE A は使用ブロック数が多く、ビジュアルタイプ B や C に比べて特に高いクオリティを持つわけでもないため、なるべく使用しないで下さい。

n. アプリケーションセーブデータ

アプリケーションのデータを格納します。ファイルのアクセスは常にブロック単位で行われます。ファイル全体のサイズが 512 バイトの倍数になるようにアプリケーションセーブデータを調整し、ファイルの最終ブロックに不定データがなるべく格納されることのないようにして下さい。

(2) バックアップファイルユーティリティ関数

バックアップファイルのイメージをメモリ上に作成したり、メモリ上に読み込んだファイルを解析するユーティリティ関数が用意されています。

注意 これらの関数は通常のファイルにのみ使用することができ、実行可能ファイルに使用することはできませんので注意して下さい。

a. バックアップファイルイメージの作成

以下のような BUS_BACKUPFILEHEADER 構造体を宣言し、パラメータを設定します。

```
extern UInt8 game_name[]; /* ゲーム名(ソートアイテム) */
extern UInt16 icon_palette[]; /* アイコンパレットデータ */
extern UInt8 icon_data[]; /* アイコンピクセルデータ */
extern UInt8 visual_data[]; /* ビジュアルデータ */
extern UInt8 save_data[]; /* アプリケーションセーブデータ */

BUS_BACKUPFILEHEADER hdr; /* バックアップファイルヘッダ */

memset(&hdr, 0, sizeof(hdr));
strcpy(hdr.vms_comment, "VMS_COMMENT"); /* VM コメントの設定 */
strcpy(hdr.btr_comment, "BOOT ROM 全角コメント"); /* BOOT ROM コメントの設定 */
memcpy(hdr.game_name, game_name, 16); /* ゲーム名(ソートアイテム)の設定 */
hdr.icon_palette = icon_palette; /* アイコンパレットデータの設定 */
hdr.icon_data = icon_data; /* アイコンピクセルデータの設定 */
hdr.icon_num = 1; /* アイコン枚数の設定 */
hdr.icon_speed = 1; /* アイコンスピードの設定 */
hdr.visual_data = visual_data; /* ビジュアルコメントの設定 */
hdr.visual_type = BUD_VISUALTYPE_C; /* ビジュアルタイプの設定 */
hdr.save_data = save_data; /* アプリケーションセーブデータ */
hdr.save_size = 0x400; /* アプリケーションセーブデータサイズ */
```

この例では、アイコン 1 枚に TYPE C のビジュアルデータ、400H のアプリケーションデータを格納すると仮定して、構造体を設定しています。

構造体を設定した後で buCalcBackupFileSize()関数を使用して、作成したいバックアップファイルが何ブロックを消費するかの計算をします。この関数では、ファイル全体のバイト数が 512 バイトの倍数でない場合、次の 512 バイト境界までがファイル全体であると仮定してブロック数を計算します。

```
Sint32 nblock;
nblock = buCalcBackupFileSize(hdr.icon_num, hdr.visual_type, hdr.save_size);
```

注意 ファイルイメージの作成には、作成先のワークメモリが必要となります。

ここでは、syMalloc()関数を利用してメモリを確保する例を示します。

```
void* buf;

buf = syMalloc(nblock * 512); /* 使用ブロック数×512 バイトを確保する */
if (buf == NULL) {
    /* メモリが確保できない */
    return NG;
}
```

buMakeBackupFileImage()関数を使用して、メモリ上にファイルイメージを作成します。この関数では、ファイル全体のバイト数が512バイトの倍数でない場合、次の512バイト境界までの残りの領域は00Hで埋められます。ただし、この領域はCRC計算には関係ありません。

```

nblock = buMakeBackupFileImage(buf, &hdr);
if (nblock < 0) {
    /* 構造体に不正なパラメータがある */
    return NG;
}

```

最後に、buSaveFile()関数を使用して、メモリの内容を実際に記録デバイスにセーブします。

注 意 必ずセーブを終了させてから、確保したメモリを開放して下さい。

(3) バックアップファイルイメージの取得

メモリ上に読み込まれたファイルを解析し、BUS_BACKUPFILEHEADER 構造体を作成する関数が用意されています。

```

extern Uint8 save_data[];          /* アプリケーションセーブデータ */
Sint32 ret;
BUS_BACKUPFILEHEADER hdr;         /* バックアップファイルヘッダ */

ret = buAnalyzeBackupFileHeader(&hdr, save_data);
switch (ret) {
    case BUD_ERR_OK:
        /* 正しく解析できた */
        break;
    case BUD_ERR_BUPFILE_CRC:
        /* 正しく解析できたが、CRC が異なっている */
        break;
    case BUD_ERR_BUPFILE_ILLEGAL:
        /* 正しい形式のファイルではない */
        break;
}

```

戻り値がBUD_ERR_OKであれば、構造体の各メンバが正しく設定されています。

戻り値がBUD_ERR_BUPFILE_ILLEGALの場合、正しい形式のバックアップファイルが読み込まれていない可能性があります。

戻り値がBUD_ERR_BUPFILE_CRCの場合、固定長ファイルヘッダのCRCに書かれている値と、実際に計算したCRCの値が異なっています。この場合ファイルの一部が正常に読み込めていない可能性があります。

(4) バックアップファイルユーティリティ関数

関 数	機 能	タイプ
buMakeBackupFileImage	バックアップファイル形式メモリイメージを作成する	A
buCalcBackupFileSize	バックアップファイルイメージのブロックサイズを計算する	A
buAnalyzeBackupFileImage	メモリ上のファイルイメージを解析する	A

(5) ボリュームアイコン

これまで解説したバックアップファイルとは全く別に、「ボリュームアイコン」と呼ばれるファイルを記録デバイスに保存することができます。

以下の形式に従ったファイルを、「ICONDATA_VMS」というファイル名で保存すると、BOOT ROM のファイル管理画面で、デフォルトアイコンの代わりに表示されます。これにより、アプリケーション独自のアイコンを自由に付けることができます。

ボリュームアイコンは、ビジュアルメモリ等の液晶を持つメモリーカードの液晶画面に表示するためのモノクロアイコンと、TV 画面に表示するためのカラーアイコンをそれぞれ 1 枚ずつ持つことができます。

ボリュームアイコンのファイルフォーマットは下図のようになっています。なお、図中のワード、ロングワードデータはリトルエンディアンとします。

注 意 ボリュームアイコンのファイルは、BOOT ROM のファイル管理画面では他のファイルと異なり、ファイル一覧に表示されないため、コピーや削除等ができませんので注意して下さい。

	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F
+0000	VM コメント(16 バイト)															
+0010	モノクロアイコン オフセット				カラーアイコン オフセット				(予約 00H)							
+0020	モノクロアイコンデータ															
+00A0	カラーアイコンパレットデータ															
+00C0	カラーアイコンピクセルデータ															

- a. VM コメント
バックアップファイルの VM コメントに準じます。
- b. モノクロアイコンオフセット
00000020H を必ず格納して下さい。
- c. カラーアイコンオフセット
000000A0H を必ず格納して下さい。
- d. モノクロアイコンデータ
BOOT ROM のファイル管理画面で、ビジュアルメモリの液晶に表示されるモノクロのアイコンです。

項 目	内 容
サイズ	横 32 × 縦 32 ピクセル
格納方式	ピクセルデータを左上から右下への情報を連続したデータとして格納する。 1 バイトで 8 ピクセルを表す。LSB 側が右、MSB 側が左となる。 1 が黒、0 が下地の色になる。
サイズ	128 バイト

e. カラーアイコン（パレットデータ/ピクセルデータ）

BOOT ROM のファイル管理画面で、TV 画面に表示されるカラーアイコンです。

項 目	内 容
サイズ	横 32 × 縦 32 ピクセル
カラー形式	4 ビット（16 色パレット）
パレット形式	ARGB4444 形式 × 16 色 2 バイト（16 ビット）で 1 色を表現 リトルエンディアン形式
格納方式	パレットデータを 16 色分格納する。続いてピクセルデータを、左上から右下への情報を連続したデータとして格納する。 1 バイトで 2 ピクセルを表す。上位 4 ビットが左のドット、下位 4 ビットが右のドットとなる。
サイズ	544 バイト

6.10.13 動的ワーク確保

これまで解説した通常の方法では、バックアップライブラリの動作にはかなりの容量のワークバッファが必要となります。特に 1M バイトの大容量記録デバイス 8 個に対応するとすると、400K バイト以上のワークを、固定的に確保する必要があります。

この章で解説する新機能「動的ワーク確保」の機能を使用することにより、必要な記録デバイスのみを必要なときだけマウントすることができるようになり、ワーク容量の大幅な削減が可能です。

(1) 動的ワーク確保の使用方法

a. ライブラリの初期化（buInit()）

初期化関数 buInit() に指定するワークアドレスに NULL を指定することにより、動的ワークの確保モードとなり新機能が利用可能になります。

その場合は以下のように初期化して下さい。なお、第 2 パラメータの対応する最大容量は無視されます。

```
void BackupInit(void)
{
    buInit(BUD_CAPACITY_128KB, USE_DRIVE_ALL, NULL, init_callback);
}
```

b. 記録デバイスの接続

記録デバイスが接続されると、完了コールバック関数が呼び出されます。

その際の引数は、以下のようになります。

引 数	意 味
drive	ドライブ番号（0～7）
op	BUD_OP_CONNECT(新コールバックオペレーションコード)
stat	接続された記録デバイスの容量フラグ BUD_CAPACITY_128KB：容量 128KB BUD_CAPACITY_256KB：容量 256KB BUD_CAPACITY_512KB：容量 512KB BUD_CAPACITY_1MB：容量 1MB
param	ユーザーパラメータ

この BUD_OP_CONNECT コールバックは、動的ワーク確保モードの場合にのみ発生するコールバックです。

(2) ワークの確保とドライブのマウント

```
/* stat で示される容量のワーク1ドライブ分を確保する */  
work = syMalloc(BUM_WORK_SIZE(stat, 1));  
if (work == NULL) return NG;
```

このワークにはアプリケーション責任において、必ず確実に利用可能なメモリ領域を必要分確保し指定して下さい。ドライブへのアクセス中にワークを開放したりワークの内容を破壊した場合の動作は保証されません。

ワークの確保に成功したら、ドライブをマウントします。

```
Sint32 err;  
  
err = buMountDisk(drive, work, BUM_WORK_SIZE(stat, 1));  
if (err != BUD_ERR_OK) return NG;
```

これにより、従来メモリーカードの接続時に自動的に行われていたマウントの処理が行われ、マウントが完了したら BUD_OP_MOUNT のコールバックが発生します。コールバック発生後は、通常どおりのファイルアクセスが可能です。

(3) ドライブのアンマウントとワークの開放

必要なファイルアクセスが終了し、そのドライブにアクセスする必要がなくなったら、アンマウントすることができます。アンマウント後には確保したワークを開放することができ、メモリを効率よく利用することができます。

```
Sint32 err;  
  
err = buUnmount();  
if (err != BUD_ERR_OK) return NG;
```

buUnmount()関数は即時完了復帰関数のため、関数呼び出しから戻ればアンマウントは完了しています。よって、buUnmount()関数呼び出し直後でワークを開放することができます。

この例では先ほど syMalloc()関数でワークを確保したので、syFree()関数でワークを開放します。ただし、この関数が BUSY を返した場合は、そのドライブに対して何らかのファイルアクセスが行われているため、アンマウントできなかったことを示します。この場合はワークを開放してはなりません。

```
syFree(work);
```

なお、どのドライブに記録デバイスがいつ接続され、取り外されるかは当然のことながらまったく予測が付きません。syMalloc()、syFree()あるいは malloc()、free()を利用してワークの確保、開放をする場合は、それらメモリマネージメント関数のアルゴリズムを十分理解した上、使い方によってはメモリのフラグメンテーションが発生する可能性があることを念頭において使用して下さい。

6.10.14 メモリーカードの切断時

すでにマウントしている記録デバイスが取り外された場合は、従来どおり BUD_OP_UNMOUNT コールバックが発生します。この場合は、自動的にアンマウントの処理が行われているため、コールバック発生時点でワークを開放しても問題ありません。

```
switch (op) {  
    case BUD_OP_UNMOUNT:  
        if (work) syFree(work);  
        break;  
    :  
}
```

6.10.15 動的ワーク確保に関する関数

関 数	機 能	タイプ
buMountDisk	記録デバイスをマウントする	B
buUnmount	記録デバイスをアンマウントする	A

6.10.16 関数一覧

記録デバイスライブラリでは、以下の関数がサポートされています。
 詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
buInit	メモリーカードファイルシステムの初期化
buExit	メモリーカードファイルシステムの終了
buAnalyzeBackupFileImage	ファイルヘッダイメージの作成
buCalcBackupFileSize	イメージのファイルサイズの計算
buDefragDisk	メモリーカードの最適化
buDeleteFile	ファイルの削除
buFindExecFile	実行ファイルの名前の取得
buFindFirstFile	第1ファイルの名前の取得
buFindNextFile	次のファイルの名前の取得
buFormatDisk	メモリーカードのフォーマット
buGetDiskFree	メモリーカードの空き容量の取得
buGetDiskInfo	メモリーカード情報の取得
buGetFileInfo	ファイルの情報の取得
buGetFileSize	ファイルのブロック数を取得
buGetLastError	最後に発生したエラーを戻す
buIsExistFile	ファイルの有無を取得
buIsFormat	メモリーカードがフォーマット済みかどうかのチェック
buIsReady	メモリーカードの接続状況を戻す
buLoadFile	ファイルのロード
buLoadFileEx	ファイルのロード（開始ブロック指定）
buMakeBackupFileImage	ファイルイメージの取得
buMountDisk	メモリーカードのマウント
buSaveExecFile	実行ファイルのセーブ
buSaveFile	ファイルのセーブ
buSetCompleteCallback	完了コールバック関数の登録
buSetFileAttr	ファイルの属性変更
buSetProgressCallback	経過コールバック関数の登録
buStat	処理の終了の検知
buUnmount	メモリーカードのアンマウント

6.11 音声入力デバイスライブラリ (ws) について

音声入力デバイスから入力されるデータを取得するためのライブラリです。

6.11.1 音声入力について

Dreamcast 本体で使用する音声入力デバイスを通して入力されたデータは、専用の音声入力ライブラリによって処理されます。

このライブラリでは、入力されるデータのサンプリングを行うだけで実際のアプリケーションでは、音声認識などの処理をする必要があります。

6.11.2 モジュール構成

音声入力ライブラリは、次の 2 つのモジュールから構成されています。

- (1) 取り込みモジュール (WSB : Wave Sampling Buffer Module)
音声入力デバイスからリングバッファに音声データを取り込むモジュールです。
- (2) 取り出しモジュール (WSS : Wave Sampling Stream Module)
リングバッファから音声データを取り出すモジュールです。

ノート この解説では、コントロールポートのドライバのワーク領域から音声入力ライブラリのリングバッファへのデータ転送を「取り込み」、リングバッファから音声認識エンジンなどのワーク領域への転送を「取り出し」と定義して説明します。また、リングバッファへ書き込むことを「書き込み」、リングバッファから読み出すことを「読み出し」とします。

6.11.3 音声入力ライブラリの特徴

このライブラリでは、内部にリングバッファを保持しています。このリングバッファは、以下の2つの特徴を持っています。

- 音声入力デバイスからの音声データは絶え間なくリングバッファに取り込まれます。そのため、リングバッファからデータが取り出されたかどうかに関わらず、データは上書きされます。
- リングバッファ内のデータを効率的かつ多目的に使用できるように、非同期に複数位置からデータを取り出すことを可能としています。

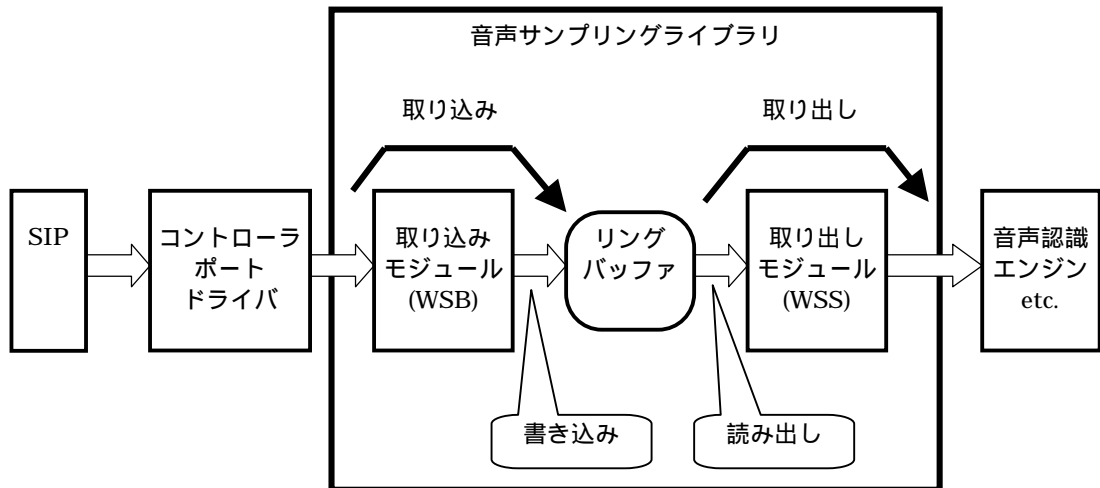


図 6 - 4 音声データの取得モデル

6.11.4 音声データの取得方法

リングバッファへの取り込みは、サーバ関数 `wsBufExecServer` を V-Sync 割り込み毎に呼び出すことで、自動的に行われます。`wsBufStart` 関数と `wsBufStop` 関数によって、バッファの取り込みの開始と停止を制御することができます。

リングバッファからのデータの取り出しは、データが必要なタイミングに `wsStmCopyPcm` 関数を呼び出すことで、取り出しを行うことができます。

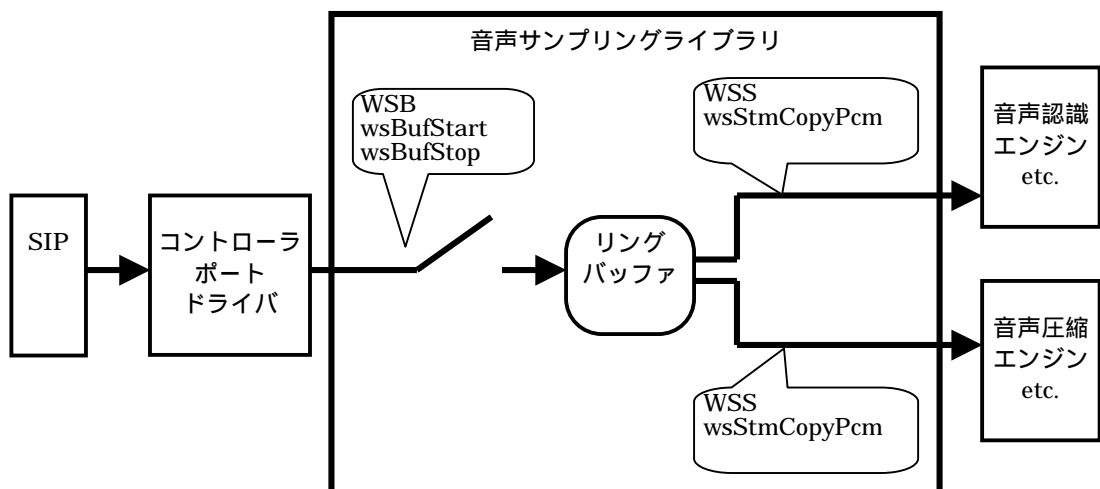


図 6 - 5 音声データの取得手順

● 例

```
/* application main function */
void main(void)
{
    WSBUF      wsbuf; /* a handle of WSB control */
    WSSSTM      wsstm; /* a handle of WSS control */
    WSS_BUF_PRM prm; /* parameters for WSB handle */
    Sint32      pno = 1; /* a number of a device */
    Sint16      data[DATA_SIZE]; /* a buffer for getting data */
    Sint32      nsmpl, cpsmpl;

    /* setting parameters for WSB */
    prm.sfreq = WSD_BUF_SFREQ_11KHZ; /* sampling frequency.F11.025kHz */
    prm.bps = WSD_BUF_SFREQ_16BIT; /* bits per sample F16bit */
    prm.gain = (WSD_BUF_PRM_AMPMAX-WSD_BUF_PRM_AMPPIX)>>2;
/* medium AMP GAIN */
    prm.blksize = 11025*5; /* 5 seconds buffer size */
    prm.nblk = 1; /* one block */
    prm.bufptr = (Sint16 *)0xC100000; /* address of a ring buffer */
    wsbuf = wsBufCreate(pno, &prm); /* creation of a WSB handle */
    wsstm = wsStmCreate(wsbuf); /* creation of a WSS handle */
    wsBufStart(wsbuf); /* start sampling */
    while(1) {
        nsmpl = wsStmGetNumSmpl(wsstm);
        if (nsmpl > 0) {
            cpsmpl = wsStmCopyPcm(wsstm, &data, 256); /* get 256 samples */
            wsStmAddRdPos(wsstm, 128); /* advance read pointer 128
samples */
        }

        /* process the data for a user*/
    }
    wsBufStop(wsbuf); /* stop sampling */
    wsStmDestroy(wsstm); /* destroy a WSS handle */
    wsBufDestroy(wsbuf); /* destroy a WSB handle */
}
```

6.11.5 関数一覧

音声入力デバイスライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
wsInit	音声サンプリングライブラリの初期化
wsFinish	音声サンプリングライブラリの終了
wsBufCreate	取り込み制御ハンドルの生成
wsBufDestroy	取り込み制御ハンドルの削除
wsBufStart	音声データの取り込みの開始
wsBufStop	音声データの取り込みの停止
wsBufExecServer	サーバ関数
wsBufGetStat	WSB モジュールのステータスの取得
wsBufGetErr	WSB モジュールのエラー取得
wsBufGetNumSmpl	リングバッファ中のサンプル数の取得
wsBufGetWrPos	リングバッファ中の書き込み位置の取得
wsBufGetSfreq	サンプリング周波数情報の取得
wsBufGetBitPerSmpl	量子化ビット数情報の取得
wsBufGetAmpGain	AMP ゲインの取得
wsBufSetAmpGain	音声入力デバイスの AMP ゲインの設定
wsBufGetSBFOV	音声入力デバイスバッファのオーバーフロービットの取得
wsStmCreate	取り出し制御ハンドルの生成
wsStmDestroy	取り出し制御ハンドルの消去
wsStmGetWsbufl	取り出し制御ハンドルから取り込み制御ハンドルの取得
wsStmGetNumSmpl	リングバッファ中のサンプル数の取得
wsStmSeekRdPos	読み出し位置のシーク
wsStmGetRdPos	読み出し位置の取得
wsStmAddRdPos	読み出し位置の更新
wsStmIsOverflow	オーバーフロー情報の取得
wsStmCopyPcm	音声データの取り出し
wsStmCopySample	音声データの取り出し
WSS_POS	書き込み位置
WSS_BUF_PRM	割り込み制御ハンドルの設定パラメータ

6.12 AVケーブルペリフェラルライブラリ (syCbl) について

AV 端子に接続されている AV ケーブルペリフェラルの種類を判別するライブラリです。

Dreamcast は、AV 端子に接続されている AV ケーブルペリフェラルの種類を判別することができます。アプリケーションはこの情報を調べることで、可能な画面モードを設定する必要があります。

ここでは、AV ケーブルペリフェラルの判別方法ならびに直接関係の深い画面モードについても説明します。

6.12.1 Dev.Box の DIP SW.とケーブル種別

開発用器材「Dev.Box」では、ドリームキャストと実機と同様の AV 端子はありません。そのため、前面パネルの DIP SW (ディップスイッチ) でケーブル種別をエミュレートします。

Switch1	Switch2	ケーブル種別
OFF	OFF	コンポジットケーブル
OFF	ON	RESERVED
ON	OFF	R G B ケーブル
ON	ON	V G A ケーブル

6.12.2 Dev.Box の ROTARY SW.と放送形式の種別

放送形式 (NTSC/PAL) は、あらかじめドリームキャスト本体内部情報に設定されています。ドリームキャスト実機では本体メモリに放送形式が登録されています。開発用器材「Dev.Box」では前面パネルの ROTARY SW. (ロータリースイッチ) を操作する事により、放送形式の種別をエミュレートします。

Number	意 味
0	日本 : NTSC
4	北米 : NTSC
6	ブラジル : PAL-M
7	アルゼンチン : PAL-N
9	欧州 : PAL

6.12.3 ケーブルの種類を判別

現在接続されているケーブルの種類を知るには、syCblCheck()関数を用います。

この関数では njInitSystem()関数のために NTSC、PAL、VGA の3種類の放送形式を返します。PAL-Mの場合はNTSCを、PAL-Nの場合はPALを返します。

定 義	意 味
SYE_CBL_NTSC	NTSC
SYE_CBL_VGA	VGA
SYE_CBL_PAL	PAL

次の例は、syCblCheck()関数を使った例です。

● 例

```
void Init_Cable(void)
{
    switch( syCblCheck() ) {
        case SYE_CBL_VGA:
            /* VGA 用の設定 */
            njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1);
            break;
        case SYE_CBL_NTSC:
            /* NTSC 用の設定 */
            njInitSystem(NJD_RESOLUTION_320x240_NTSCNI,
NJD_FRAMEBUFFER_MODE_RGB565, 1);
            break;
        case SYE_CBL_PAL:
            /* PAL 用の設定 */
            njInitSystem(NJD_RESOLUTION_320x240_PALNI,
NJD_FRAMEBUFFER_MODE_RGB565, 1);
            break;
    }
}
```

6.12.4 関数一覧

関 数	機 能
syCblCheck	簡易モニタ選択用種別の取得

7 フラッシュメモリ関連のライブラリ

ここでは、本体内部のフラッシュメモリやプレイ履歴の機能等に関連するライブラリについて、それぞれ説明します。

7.1 コンフィグレーションライブラリ (syCfg) について

ブートロム画面の設定メニューにある「言語」「音声」の内容を取得するライブラリです。

7.1.1 初期化・終了

フラッシュメモリの性質上、アクセス時にバッファ (16K Byte) が必要となり、このバッファのアドレスの設定をコンフィグレーションライブラリの初期化時に行っています。初期化時はバッファのアドレスを内部的に保存するのみで、フラッシュメモリに直接アクセスするのは、それぞれの参照・更新の関数で行います。

(1) 初期化の失敗

バッファに NULL を渡した場合は、初期化が失敗します。現在のところ初期化が失敗するのは、この場合のみです。

7.1.2 フラッシュメモリへの書き込み

コンフィグレーションライブラリのサウンドのステレオ / モノラルの設定は、本体フラッシュメモリに保存します。

(1) フラッシュメモリの注意事項

フラッシュメモリには書き込みの上限値が存在するため、コンフィグレーションライブラリの呼び出しを最小限にして下さい。

オプション画面で毎 V コンフィグレーションライブラリを呼び出すような方法を行わずに、書き込みの内容が確定した時点でライブラリを呼び出すという方法を取って下さい。

7.1.3 関数一覧

コンフィグレーションライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syCfgInit	コンフィグレーション関数群の初期化
syCfgExit	コンフィグレーション関数群の終了処理
syCfgGetIndividualID	本体 ID の取得
syCfgGetLanguage	言語設定の取得
syCfgGetSoundMode	ステレオ / モノラルの設定情報の取得
syCfgSetSoundMode	ステレオ / モノラルの設定情報の変更

7.2 プレイ履歴ライブラリ (uf) について

Dreamcast 本体のフラッシュメモリに記録された、アプリケーション毎のプレイ履歴（タイトルの起動回数など）の参照及び更新等を行うライブラリです。

7.2.1 プレイ履歴ライブラリとは

プレイ履歴ライブラリは、タイトル毎のデータを保存し読み出すためのものです。ufInitSystem で初期化を行い、以降は ufPlayTime を呼び出す毎に、内部のデータを更新していきます。最終的に ufSynch を呼び出した時点でデータを書き込みます。

(1) プレイ履歴とは

各アプリケーションタイトルについて、それを起動した Dreamcast の本体に、（タイトル名、起動回数、前回起動日時、プレイ時間、攻略率、その他の情報）などを、プレイ履歴として保存することができます。

アプリケーションはこれらの値を取得することによって、過去の自社アプリケーションのプレイ実績等や、その Dreamcast 本体のユーザー固有のプレイ情報を得ることができます。

このプレイ履歴によってアプリケーションの進行や情報などを、変化させることが可能になります。

7.2.2 動作概要

(1) 動作説明

ufSynch が自動的に呼び出されるしくみとして ufAutoSynch が用意されています。引数に ufPlayTime を何回か呼び出したときに ufSynch を呼び出すかを指定します。この値を 1 にすると、ufPlayTime を呼び出すたびにデータの書き込みをし、0 にすればタイトル製作者側で ufSynch を呼び出さない限りは更新されません。

プレイ履歴は、Dreamcast のアプリケーションの起動時及び実行中に、Dreamcast 本体内蔵のフラッシュメモリ（本体メモリ）に、情報を蓄積することによって実現しています。

(2) プレイ履歴の構成

プレイ履歴は、アプリケーション起動時に処理が行われる AIP 部と、アプリケーションがライブラリを使用してアクセスするライブラリ部より成ります。

(3) AIP 部分

アプリケーション起動時のプレイ履歴の保存処理は、IP 内の AIP と呼ばれる部分で行われます。実行されるのは、アプリケーション起動時の「ライセンス SEGA ロゴ」が表示されているタイミングです。

a. IP について

アプリケーションの初回起動時に、IP に記述された商品番号をタグとして、そのアプリケーション用のフィールドが確保され、以下の情報が本体に記録されます。

表 7 - 1 AIP で記録されるプレイ履歴情報（初回起動時）

商品番号	IP で指定した商品番号
商品名（英文）	IP で指定した商品名（英文）
商品名（和文）	IP で指定した商品名（和文）
ジャンル	IP で指定したジャンル
初回起動日時	最初に起動された時の本体時計の時刻を保存する

プレイ履歴のフィールドは最大 100 タイトル分格納できます。100 タイトルを超えてプレイ履歴を使用しているアプリケーションを起動した場合は、前回起動日時が最も古いタイトルに上書きされます。

すでにそのタイトルのプレイ履歴の保存されているアプリケーションの起動時には、以下の情報が更新されて記録されます。

表 7-2 AIP で記録されるプレイ履歴情報（2 回目以降）

前回起動日時	起動された時の本体時計の時刻
起動回数	起動された回数
起動時間帯	1 日を 24 の時間帯に振り分けた、起動回数

(4) Ipmaker について

AIP 部でのプレイ履歴を記録するためには、Dreamcast SDK の IPmaker で IP を作成する時に、プレイ履歴を使用するにチェックを入れて必要事項を入力して下さい。

ELF ファイルを作成してデバッグを行う場合は、システム ID 領域（systemid.obj、zero.obj）に正しい製品番号を入れたものを使用して下さい。これは SDK のプレイ履歴のサンプルにある、systemid.src、zero.src をアセンブルして作成します。また、AIP 部分（aip.obj）を、プレイ履歴を保存する（aipf.obj）に差し替えて実行して下さい。

プレイ履歴は、アプリケーション本体で一切考慮しなくても、この AIP 部の処理のみで起動した回数など最低限の情報が自動的に保存されるようになります。

続編等で参照して利用することを考慮し、一応対応しておくといったことが可能になります。

ノート 国内のセガタイトルは全て、AIP 部のプレイ履歴の保存に対応しています。

(5) 注意事項

ufInitSystem 関数を用いてプレイ履歴ライブラリの初期化を行う際には、セガ・ライブラリの初期化過程のなかで、割り込みのマスクを解除したあと、pdExecPeripheralServer 関数が 2 回以上呼ばれている必要があります。

プレイ履歴ライブラリはイニシャライズ時に、Dreamcast に接続されているペリフェラル情報を取得していますが、pdExecPeripheralServer 関数が 2 回呼ばれるまで、正しいペリフェラル情報が取得できない場合があります。

注意 本体メモリへの書き込み頻度は、1 時間あたり 12 回程度（平均 5 分に 1 回以下の頻度）とし、それより高い頻度で書き込みを行わないように注意して下さい。
また、プレイ履歴ライブラリで書き込みを行うと、フラッシュメモリのガーベージコレクションが発生する場合があります。この発生頻度を低減させ、フラッシュメモリの耐久性を確保するため、及びゲーム中でのリフレッシュの発生頻度を低減させる理由によるためです。

(6) プレイ履歴ライブラリの仕様

ufPlayTime は自動保存の ON/OFF 関係なくデータを更新し ufSynch は自動保存 ON/OFF フラグを参照して、更新するかしないかを判断します。これは、可能な限りデータがあるかどうかをチェックする事で処理を軽くするためです。それ以外の関数は、すべて自動保存フラグの値を参照して更新する/しないを判断します。

(7) ワークエリアについて

プレイ履歴ライブラリではフラッシュメモリに書き込むという仕組み上、ある程度のワークエリアを必要とします。最低 64KB+60byte が必要で、かつこれらは連続している必要があります。

(8) リフレッシュ動作について

フラッシュメモリは追記方式を使っているため、追記可能領域がなくなった場合、フラッシュメモリの内容の更新を行う必要があります。これをリフレッシュと呼びますが、この動作を行うときに更に 64KB+32KB が必要になります。

そのため、プレイ履歴ライブラリでは、ufPlayTime を呼び出したときにどの程度追記が可能かという値を返します。この値をみて、ufSynch を呼び出すかどうかを決めて下さい。

ノート 現在このリフレッシュが起きると、最高で 4 秒程度プログラムが停止することが分かっています。

(9) プログラムの場所について

プレイ履歴ライブラリは内部に若干の静的なワークを使っています。この値を保持するために、常駐部にリンクして下さい。

7.2.3 ライブラリ部

保存されたプレイ履歴をアプリケーションで参照する場合で、プレイ履歴の AIP 部でサポートしていない部分を更新するときなどは、プレイ履歴ライブラリを使用してフラッシュメモリにアクセスします。

(1) ライブラリ使用上の注意点

プレイ履歴ライブラリを使用する上で、以下の点に特に注意して下さい。

a. プレイ履歴の参照について

プレイ履歴ライブラリは、アプリケーションを起動したタイトルのプレイ履歴のみ更新が可能です。プレイ履歴の参照については、タイトルの商品コードが分かればそのタイトルのプレイ履歴を参照することは可能です。

また、本体にすでに記録されているプレイ履歴の全てのタイトルの商品コードを取得することができますので、本体に記録されている全てのタイトルのプレイ履歴を参照することが可能です。

b. プレイ履歴の領域へのアクセスについて

プレイ履歴ライブラリを使用しないで、フラッシュメモリのプレイ履歴の領域にアクセスを行わないようにして下さい。

c. AIP 部でプレイ履歴の保存を行わないアプリケーションの場合について

AIP 部でプレイ履歴の保存を行わないアプリケーションの場合は、プレイ履歴ライブラリを使用することはできません。その場合はすべてエラーになります。

d. プレイ履歴ライブラリを使用してアプリケーションを作成する場合

別紙の「ドリームキャスト・ソフトウェア作成基準」に、十分に気をつけて下さい。プレイ履歴情報の取り扱いについては十分な注意が必要になります。

e. プレイ履歴でのリフレッシュ処理について

プレイ履歴はフラッシュメモリに追記方式で格納しているため、一定量の書き込みを行うたびに、リフレッシュというガベージコレクション処理が行われます。この処理中は約 1 ~ 4 秒の間他の処理を行うことができなくなることがあります。また、リフレッシュは書き込みの量に比例して起こりますので、通常の場合のプレイ履歴の更新は、平均 5 分に 1 回以下の頻度にして下さい。

f. 履歴フィールドについて

ufPlayTime () 関数によって保存され、ufReadAllHead () 関数によって取得することのできるプレイ時間の履歴フィールドは、そのタイトルをプレイした 1 日を 24 分割した時間帯毎のプレイ時間の合計を表しています。

このフィールドについて AIP 部では、アプリケーションを起動した時間帯に対して 1 分を加算しています。

これにより、ライブラリ部での対応を行っているアプリケーションの場合は起動時にかかった時間を 1 分と概算するとともに、ライブラリ部での対応を行っていないアプリケーションについては時間帯毎の起動回数を示すフィールドと解釈することが可能です。AIP 部のみの場合は、ufPlayTime () 関数による保存が行われないためです。

(2) 初期化 (ufInitSystem ())

プレイ履歴ライブラリは ufInitSystem () 関数を使用して、初期化を行います。

この関数によって初期化を行う前に、sbInitSystem () による初期化を完了し、更に pdExecPeripheral 関数を 2 回以上呼びだしておく必要があります。

ufInitSystem () 関数はプレイ履歴ライブラリを初期化するときに、本体に接続されているペリフェラルの情報を取得していますが、これは、ペリフェラルの情報を取得する pdExecPeripheralServer () 関数が 2 回以上呼ばれないと正しいペリフェラルの値が取得できない場合があるためです。

(3) ワークメモリ (ufChangeWorkBuffer ())

プレイ履歴ライブラリは、64KB (work1) + 60 Byte (work2) + 64KB (work3) + 32KB (work4) のワークメモリを必要とします。

プレイ履歴はフラッシュメモリに保存しており、読み書きにバッファを使用するため一定量のワークメモリを必要とします。

このうち、work3 及び work4 はリフレッシュを行う時のみに使用しますので、リフレッシュを行わないときは NULL を指定しておいて、リフレッシュを行う時のみに ufChangeWorkBuffer () 関数を使用してバッファを与えるということが可能です。

この場合、自動的にリフレッシュが起これないように ufAutoSynch () 関数を使用して、データが自動的に更新されないように設定する必要があります。

リフレッシュを行う必要があるかどうかは、ufPlayTime () 関数の戻り値の残りの追記可能パケット数が 2 以下になったかどうかで判定します。

7.2.4 リフレッシュについて

(1) フラッシュメモリ管理ライブラリ

プレイ履歴ライブラリはフラッシュメモリ管理ライブラリを使用して、本体フラッシュメモリへのアクセスを行っています。

本体フラッシュメモリへの書き込みは、バイト単位で行うことができますが、消去の際はブロック単位でしか行うことができません。

プレイ履歴ライブラリは 32KB のブロック及びリフレッシュ時のバックアップ用に、64KB のブロックを使用しています。

このフラッシュメモリは、一度書き込みしたバイトへは、消去してからでないと再び書き込むことはできません。従って、仮に 1 バイトを書き換えたい場合でも、ブロック全体を保存して消去し変更したい個所を書き換えてから、それを書き戻さなくてはなりません。これを解消するために、仮想フラッシュメモリの空間を用意し、実際には追記書き込みを行って、書き換えの制限を緩和し行うものがフラッシュメモリ管理ライブラリです。

(2) 追加書き込みについて

フラッシュメモリ管理ライブラリでは

オフセット 2 Byte

データ本体 60 Byte

チェック符号 (CRC) 2 Byte

以上の合計 64 Byte 単位で追加書き込みをしていきます。この単位をパケットと呼びます。

ブロックがパケットでいっぱいになって追加書き込みをできる領域が無くなった時に、リフレッシュが起こります。

リフレッシュとは、追記パケットを整理し重複を無くす作業のことです。ufPlayTime () 関数が返す追記可能パケット数はこの値のことです。

(3) リフレッシュ処理

フラッシュメモリのリフレッシュを行う時に、ブロックの消去を行う必要がありますが、この消去処理は他の処理と並行して行うことができません。

更に、消去処理は、ブロックのサイズやデバイスの劣化度にもよりますが、数百ミリ秒程度かかります。よって、リフレッシュが行われる可能性がある時の ufPlayTime () 関数や ufSync () 関数は画面フェイドアウト後や Now Loading 画面など、ゲーム画面が停止しても構わないタイミングで行う必要があります。(ハード仕様上、実際のディスクアクセス中にはフラッシュメモリにアクセスを行うことはできません。)

そこで、通常は ufPlayTime () 関数で残りパケット数を確認しつつ書き込みを行い、残りパケット数が少なくなった場合にリフレッシュを行うという方法が考えられます。登録されているプレイ履歴の数や内容にもよりますが、1度リフレッシュを行った後は100パケット以上は確保されます。

プレイ履歴の書き込み時にリフレッシュが発生した場合にブロック消去及び再書き込みを行いますが、この処理を行っている間に本体電源が落とされるとブロック全体の情報が消滅するおそれがあります。

このことを回避するために、AIP 部及びプレイ履歴ライブラリではリフレッシュを行う場合に、別のブロックにデータを退避してバックアップを作成します。

次回起動時に AIP 部でプレイ履歴のブロックが不正な状態であると判断した場合は

バックアップを復帰する処理を行います。

これを実現するために、リフレッシュ処理を行うときには最大2ブロック分の消去処理が入り、更に、ブロック全体の書き込みにも数百 ms の時間を必要としますので、最大で数秒間停止する可能性があります。

7.2.5 関数一覧

プレイ履歴ライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
ufAutoSynch	プレイ履歴更新間隔の指定
ufChangeAutoSave	自動保存フラグの設定
ufChangeWorkBuffer	プレイ履歴ワークエリアの変更
ufConnect	ネットワーク接続時刻の記録
ufDisconnect	ネットワーク接続時間の記録
ufExitSystem	プレイ履歴機能の終了処理
ufGetControllerList	コントローラリストの取得
ufGetExtControllerList	拡張コントローラリストの取得
ufInitSystem	プレイ履歴機能の初期化
ufPlayTime	プレイ時間の保存
ufReadAllHead	指定したタイトルの全項目のデータの読出し
ufReadAllProduct	プレイ履歴からの商品番号リストの取得
ufSafeRefresh	プレイ履歴メモリのリフレッシュ
ufSynch	プレイ履歴の書き込み
ufTitleDelete	タイトルの履歴の消去

8 その他のライブラリ

ここでは、特殊なライブラリについて、それぞれ説明します。

8.1 メモリコピーライブラリ (syMemCopy) 及びメモリセットライブラリ (syMemSet) について

メモリ配置により最適なメモリコピー機能を提供するライブラリです。

従来、Ninja ライブラリに含まれていたメモリコピーライブラリ (njMemCopy) を、システムライブラリである shinobi2.lib に移動し、プレフィックスを変更しました。

また、メモリセットライブラリを新規に追加しました。

8.1.1 メモリコピーライブラリ (syMemCopy) について

(1) メモリ内容をコピーする

本体のメインメモリ内でメモリ内容のコピーを行います。

a. バイト単位でコピーする。

SyMemCopy 関数を使用し、src から dst へ num バイト単位分コピーします。

• 8 バイト単位でコピーする

syMemCopy8 関数を使用し、src から dst に (num × 8) バイト分コピーします。

• 16 バイト単位でコピーする

syMemCopy16 関数を使用し、src から dst に (num × 16) バイト分コピーします。

• 32 バイト単位でコピーする

syMemCopy32 関数を使用し、src から dst に (num × 32) バイト分コピーします。

b. ワード単位でコピーする。

SyMemCopy2 関数を使用し、src から dst に num ワードコピーします。

c. ロング単位でコピーする

syMemCopy4 関数を使用し、src から dst に num ロングコピーします。

d. ストアキューを使用してコピーする

syMemCopySQ 関数を使用して、src から dst に (num × 32) バイトストアキューを使ってコピーします。

その場合に、コピー先を指定する引数*dst とコピー元を指定する引数*src は、32 バイトアライメントである必要があります。

ノート 転送量が 64K バイト以下の場合、syMemCopy32 関数を使った方が高速です。

(2) 関数一覧

メモリコピーライブラリでは、以下の関数がサポートされています。

詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syMemSet	メモリセット (バイト単位)
syMemSet2	メモリセット (ワード単位)
syMemSet4	メモリセット (ロング単位)
syMemSet8	メモリセット (8 バイト単位)

関数名	機 能
syMemSet16	メモリセット (16 バイト単位)
syMemSet32	メモリセット (32 バイト単位)
syMemSetSQ	メモリセット (ストアキューの使用)

8.1.2 メモリセットライブラリ (SyMemSet) について

(1) メモリをセットする

本体のメモリに対して様々な形式でセットします。

a. バイト単位でセットする

syMemSet 関数を使用し、dst から num バイトを dat で埋めます。

• 8 バイト単位でセットする

syMemSet8 関数を使用し、dst から (num × 8) バイトをロングワードサイズのデータ dat で埋めます。

• 16 バイト単位でセットする

syMemSet16 関数を使用し、dst から (num × 16) バイトをロングワードサイズのデータ dat で埋めます。

• 32 バイト単位でセットする

syMemSet32 関数を使用し、dst から (num × 32) バイトをロングワードサイズのデータ dat で埋めます。

b. ワード単位でセットする。

SyMemSet2 関数を使用し、dst から (num × 2) バイトをワードサイズのデータ dat で埋めます。

c. ロング単位でセットする

SyMemSet4 関数を使用し、dst から (num × 4) バイトをロングワードサイズのデータ dat で埋めます。

d. ストアキューを使用してセットする

syMemSetSQ 関数を使用し、dst から (num × 32) バイトをロングワードサイズのデータ dat で埋める際に、ストアキューを使用します。

その場合に、コピー先を指定する引数*dst は、32 バイトアライメントである必要があります。

ノート メモリセットする領域が 64K バイト以下の場合、syMemSet32 関数を使った方が高速です。

(2) 関数一覧

メモリセットライブラリでは、以下の関数がサポートされています。

詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syMemSet	メモリセット (バイト単位)
syMemSet2	メモリセット (ワード単位)
syMemSet4	メモリセット (ロング単位)
syMemSet8	メモリセット (8 バイト単位)
syMemSet16	メモリセット (16 バイト単位)
syMemSet32	メモリセット (32 バイト単位)
syMemSetSQ	メモリセット (ストアキューの使用)

8.2 リアルタイムクロックライブラリ (syRtc) について

Dreamcast のカレンダー（日付・時間）の取得及び更新を行う機能を提供するライブラリです。

8.2.1 リアルタイムライブラリの初期化について

syRtcInit 関数を使用して、リアルタイムクロック関数群の初期化、及びワークエリアの確保を行います。

リアルタイムクロック関数群は、G2 バス上にある AICA のリアルタイムクロックにアクセスし、日付・時刻に関する情報を取得および設定します。

AICA のリアルタイムクロックは 32 ビットのカウンタで、1 秒で 1 カウントしていきます。

8.2.2 リアルタイムライブラリの終了

syRtcFinish 関数を使用して、リアルタイムクロック関数の終了およびワークエリアの解放を行います。

8.2.3 フラッシュメモリへの書き込み

コンフィグレーションライブラリのサウンドのステレオ/モノラルの設定は、本体フラッシュメモリに保存します。

- フラッシュメモリの注意事項

フラッシュメモリには書き込みの上限値が存在するため、コンフィグレーションライブラリの呼び出しを最小限にしてください。

オプション画面で毎 V コンフィグレーションライブラリを呼び出すような方法を行わずに、書き込みの内容が確定した時点でライブラリを呼び出すという方法を取ってください。

8.2.4 関数一覧

音声入力デバイスライブラリでは、以下の関数がサポートされています。

詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syRtcCompareDate	日付時刻の比較
syRtcCountToDate	カウント数から日付時刻への変換
syRtcDateToCount	日付時刻からカウント数への変換
syRtcExecServer	日付時刻に関する内部情報の更新（サーバ関数）
syRtcFinish	リアルタイムクロック関数群の終了
syRtcGetDate	日付時刻の取得
syRtcGetStat	サーバ関数の状態の取得
syRtcInit	リアルタイムクロック関数群の初期化
syRtcSetDate	日付時刻の設定
syRtcSetServerMode	サーバ関数の動作の設定

8.3 ビデオ設定ライブラリ (syVideo) について

ビデオ関連の設定や状態 (VGA / NTSC / PAL 等) の取得をうライブラリです。

従来のケーブルライブラリ (syCbl) を廃止し、代わりにこのライブラリを使用します。

8.3.1 ビデオ設定ライブラリの概要

Ninja2 ライブラリを使用せずに行う、ビデオ関連の設定や状態の取得を行うライブラリです。次の機能があります。

- ビデオモード (VGA、NTSC、PAL) の取得
- 放送形式の取得
- 接続ケーブルの取得
- ボーダーカラーの設定

通常は、syVideoGetVideoMode 関数を用いてこの情報を取得し、グラフィックライブラリの画面モードの設定を行います。

syVideoGetVideoMode 関数は、内部で以下に説明する syVideoGetCableType 関数と syVideoGetBroadcastType 関数を使用しています。

8.3.2 Dreamcast のケーブル

Dreamcast の AV 端子に接続する画像 / 音声出力用ケーブルには、ステレオ AV / S 端子ケーブル、VGA ボックス、RGB ケーブルの 3 種類が存在します。

ケーブルライブラリを使用すると、どのケーブルが接続されているかを判別することができます。

アプリケーションは、接続されたケーブルを考慮して、sbInitSystem () 関数の初期化パラメータを変更し、VGA モニタやテレビに画像が正しく出力されるようにする必要があります。

IP 作成用ツール「Ipmaker」に VGA 対応アプリケーションかどうかを指定する項目がありますが、この項目は、非 VGA 対応アプリケーションの起動時に VGA ボックスが接続されている場合、メッセージを表示して起動しないようにするために用意されているものですので、VGA 対応アプリケーションはアプリケーション内で接続ケーブルを判別して、正しくグラフィックライブラリの初期化を行って下さい。

また、ケーブルの種類による処理の変更 (出力画面のイメージを変更等) にも使用することができます。

syVideoGetCableType 関数は、関数を呼び出した時点での接続しているケーブルの種類を返します。この関数は、主に RGB ケーブルとビデオケーブルの接続を判定し、処理を変更したい場合に使用します。

通常の場合は、簡易版の syVideoGetVideoMode 関数を使用して、NTSC/PAL と VGA を判別すれば十分です。

8.3.3 放送形式

Dreamcast は仕向地に応じて NTSC (日本 / 米国)、PAL (欧州)、PAL-M (ブラジル)、PAL-N (アルゼンチン) のいずれかの放送形式が本体に記録されています。ステレオ AV / S 端子ケーブルまたは RGB ケーブルに画像を出力する場合、放送形式に応じたグラフィックライブラリの画面モードの初期化を行う必要があります。VGA ボックスに画像を出力する場合は放送形式による違いはありません。

syVideoGetBroadcastType 関数を使用すると、Dreamcast 本体の放送形式を判別することができます。この関数は、主に PAL と PAL-N または NTSC と PAL-M の判別を行い、処理を変更したい場合に使用します。

通常の場合は、簡易版の syVideoGetVideoMode 関数を使用して、NTSC と PAL を判別すれば十分です。

8.3.4 その他

アプリケーションを作成する場合、グラフィックライブラリの画面モードの初期化の他に、処理を変更する必要がある点が多数あります。

8.3.5 ケーブル判別・画面出力モード一覧表

章末に「ビデオケーブル判別 / 画面出力モード一覧表」を補足しましたので、そちらを参照して下さい。

8.3.6 Shinobi2 でのビデオ設定ライブラリへの変更経緯

Shinobi2 環境では、アプリケーションプログラムは画面モードを設定する際に、Dreamcast に接続してあるケーブルを直接参照せず、BootROM が接続ケーブルによって設定した画面モードを参照するように、変更しました。

ユーザーがアプリケーション起動後に接続ケーブルを変更した場合に、従来のケーブルライブラリではアプリケーションが変更後のケーブルに対し画面設定を行ってしまい、本体起動時とアプリケーションでの画面設定が異なってしまいます。

上記に対応するため、ビデオ設定ライブラリを Shinobi2 で新規に追加しました。

ビデオ設定ライブラリを使用してアプリケーション起動時の Dreamcast の設定画面モードを取得することができます。

このビデオ設定ライブラリを使用して、Ninja2 に自動的に適切な画面モード (VGA, NTSC, PAL など) を設定するモードが追加されました。

また、放送形式や接続ケーブルの種類によって動作を変更したいアプリケーションのために、放送形式及び本体起動時の接続ケーブルの種類を取得する関数も用意しました。

ビデオ設定ライブラリを使用することによって、従来のケーブルライブラリと同様の処理を、起動時の接続ケーブルに従って行うことができます。

これにともない、従来のケーブルライブラリは削除されました。

ただし、RGB ケーブル接続時は RGB ケーブルが接続されていることを確認するため、ビデオ設定ライブラリは接続ケーブルを判別しており、RGB ケーブルであった場合は、画面モードを強制的に RGB に設定します。

また、Ninja2 を使用しないビデオ設定関連の関数も作成しました。

Ninja2 の初期化を行う前に、Ninja2 を使用せず、ボーダーカラーの設定を行うことができます。

8.3.7 関数一覧

ビデオ設定ライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syVideoInit	ビデオ設定ライブラリの初期化
syVideoExit	ビデオ設定ライブラリの終了
syVideoGetBroadcastType	放送形式の取得
syVideoGetCableType	接続ケーブル種別の取得
syVideoGetVideoMode	ビデオモードの取得
syVideoSetBorderColor	ボーダーカラーの設定
syVideoWaitVBlankSetBorderColor	ボーダーカラーの設定

8.4 シリアルライブラリ (syCom) について

SH4 内蔵の SCIF (FIFO 内蔵シリアルコミュニケーションインタフェース) を利用したシリアル通信を、可能にするライブラリです。

8.4.1 シリアルライブラリの仕様について

本ライブラリの仕様は以下のとおりです。また、この仕様は将来変更の可能性があります。

項 目	内 容
対応ハードウェア	SH4 内蔵 SCIF
通信転送速度	9600、19200、38400、57600、115200BPS、230400BPS、260KBPS、312KBPS、390KBPS、520KBPS、1.5MBPS
データビット	7、8
ストップビット	1、2
パリティ	なし、偶数、奇数
フロー制御	なし、ハードウェア

8.4.2 通信の初期化と終了

(1) ライブラリの初期化 (syComInit())

ライブラリを初期化するには、syComInit()関数を使用します。

```
Uint8 recv_buf[0x1000];

syComInit(recv_buf, 0x1000);
```

パラメータは、データの受信に使用するリングバッファのアドレスと、そのサイズです。バッファサイズは必ず(1 < n)となるようなサイズ(0x1000、0x400 等)を指定して下さい。それ以外のサイズでは正しく受信が行えなくなります。

(2) 通信経路のオープン (syComOpen())

通信を開始する前に、通信経路をオープンする必要があります。それには syComOpen()関数を使用します。

```
syComOpen(SYD_COM_SPEED_38400);
```

例のように通信速度を引数として syComOpen()を呼び出します。なお、この関数で通信経路をオープンした場合の通信パラメータは以下の通りです。

パラメータ	内 容
データビット	8 ビット
ストップビット	1 ビット
パリティ	なし
フロー制御	なし

(3) 通信経路のクローズ (syComClose())

通信を終了し、通信経路をクローズするには、syComClose()関数を使用します。

```
syComClose();
```

(4) ライブラリの終了 (syComExit())

ライブラリを終了するには、syComExit() 関数を使用します。

```
syComExit();
```

(5) パラメータを指定して通信経路をオープンする (syComOpenEx())

syComOpenEx()関数を利用すると、通信に関する詳細なパラメータを指定して通信経路をオープンすることができます。

```
SYS_COM_PARAM param;

param.speed = SYD_COM_SPEED_38400;
param.databit = SYD_COM_DATA_8BIT;
param.stopbit = SYD_COM_STOP_1BIT;
param.parity = SYD_COM_PARITY_OFF;
param.flow = SYD_COM_FLOW_HARD;

syComOpenEx(&param);
```

● オープンパラメータの詳細

メンバ	意味	指定する値	説明
speed	通信速度	SYD_COM_SPEED_9600 ~ SYD_COM_SPEED_1M	通信速度を指定します。
databit	データビット	SYD_COM_DATA_8BIT	データビット8ビットを指定します。
		SYD_COM_DATA_7BIT	データビット7ビットを指定します。
stopbit	ストップビット	SYD_COM_STOP_1BIT	ストップビット1ビットを指定します。
		SYD_COM_STOP_2BIT	ストップビット2ビットを指定します。
parity	パリティ	SYD_COM_PARITY_OFF	パリティなしを指定します。
		SYD_COM_PARITY_EVEN	偶数パリティを指定します。
		SYD_COM_PARITY_ODD	奇数パリティを指定します。
Flow	フロー制御	SYD_COM_FLOW_OFF	フロー制御なしを指定します。
		SYD_COM_FLOW_HARD	ハードウェアフローを指定します。

8.4.3 データの送受信について

(1) データの送信 (syComPutWait())

データを送信するには、1 バイトデータ送信関数 syComPutWait()を使用します。

```
syComPutWait('A')
syComPutWait('B')
syComPutWait('C')
```

この例では、「A」「B」「C」という文字を1バイトずつ送信しています。

この関数を連続して使用した場合は、前回のデータが送信中であった場合、送信完了を待ってからデータを送信します。

通常のデータ送信であれば、上記の関数で十分です。しかし、この関数では送信可能になるまで待機して送信を行います。

この待機時間を無くしたい場合や、送信可能かどうかを調べたい場合は、関数 syComIsPut()と syComPutq()を組み合わせて使用して下さい。

- 例 送信可能であれば、「A」という文字を送信する。

```
if (syComIsPut()) {
    syComPutq('A')
}
```

(2) データの受信 (syComIsGet(), syComGet())

データの受信は、syComIsGet()とsyComGet()関数を使用していきます。

```
char data;

if (syComIsGet()) {
    data = syComGet();
}
```

上記の例のようにsyComIsGet()関数を使用し、受信バッファにデータがあるかどうかを確認した後、データがあれば、syComGet()関数を使用してバッファからデータを取り出します。

8.4.4 関数一覧

シリアルライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syComInit	低レベル通信ライブラリの初期化
syComExit	低レベル通信ライブラリの終了
syComClose	通信経路のクローズ
syComGet	受信バッファから1バイトのデータ取得
syComIsGet	受信バッファにデータがあるかどうかを取得
syComIsPut	データ送信可能かどうかの取得
syComOpen	通信速度を指定した通信経路のオープン
syComOpenEx	オープンパラメータを指定した通信経路のオープン
syComPutq	1バイトのデータ送信(ウェイトなし)
syComPutWait	1バイトのデータ送信

8.5 拡張機器判別ライブラリ (syExtChk) について

Dreamcast の EXTENSION 端子に、周辺機器 (モデム / LAN アダプタ / その他) が接続されているかどうかを判別するライブラリです。

EXTENSION 端子 (G2 BUS EXTENSION) に接続される予定の周辺機器の種類が増加し、また周辺機器が接続されていないと正常に動作しないアプリケーションを作成するために、必要な周辺機器が接続されているかをアプリケーションが判別する方法を提供する必要が出てきました。

また、同期 / 非同期デバイスによって判別方法が違ふことと、接続されていないデバイスへのアクセス (存在確認のみでも) を行なった場合に発生する、G2 CPU Access Timeout 割込みをクリアすることに対応した、接続及び判別を一元的に管理するライブラリとして提供する必要があります。

8.5.1 拡張機器判別ライブラリの概要

ライブラリシステムの初期化の際に接続されている周辺機器の判別を行い、そのライブラリを使用して接続状態を取得することが可能です。周辺機器の抜き差しを、電源を入れたままの状態で行なうことは禁止されているため、ライブラリについても活線挿抜には対応していません。

8.5.2 拡張機器判別ライブラリの初期化

Sint32 syExtChkInit を使って初期化します。

注意 この関数は、syHwInit()関数内で呼び出されるため、ユーザーは通常呼び出す必要はありません。
この関数を呼び出すには、G2 ライブラリが初期化されている必要があります (G2 ライブラリは syHwInit()関数内で初期化されます)。
従って、syHwInit()関数より前に使用することはできませんので注意して下さい。

8.5.3 拡張機器判別ライブラリの終了

int32 syExtChkExit を使って終了します。

この関数を呼び出した後は、再び syExtChkInit()関数を使用して初期化することが可能になります。

注意 通常は、syHwFinish ()関数内で呼び出されるため、ユーザーは通常呼び出す必要はありません。

8.5.4 情報取得関数使用時の注意事項

EXTENSION 端子に接続される拡張機器の、情報を取得する関数 (syExtChkConnect、syExtChkModemCountry、syExtChkModemSpeed) を使用する前に、sbInitSystem()関数内の syHwInit()関数から拡張機器判別ライブラリの初期化を行っておく必要があります。

初期化をせずに使用した場合は、返値は常にエラーとなります。

8.5.5 関数一覧

拡張機器判別ライブラリでは、以下の関数がサポートされています。
詳しい内容については、関数リファレンスをご参照下さい。

関数名	機 能
syExtChkConnect	引数に指定した機器が EXTENTION 端子に接続されているかどうかを返す
syExtChkExit	拡張機器判別ライブラリの終了
syExtChkInit	拡張機器判別ライブラリの初期化
syExtChkModemCountry	EXTENTION 端子に接続されているモデムのカントリーコードを返す
syExtChkModemSpeed	EXTENTION 端子に接続されているモデムのスピードを返す

9 言語使用時の注意事項

ここでは、各プログラミング言語を使用する際の注意事項などを説明します。

9.1 アセンブラ使用時の注意

変換プログラムのアセンブラを使用する際に、注意する点について説明します。

9.1.1 C 言語とのインターフェイス

SHC コンパイラ使用時は、C / C++コンパイラユーザーズマニュアル 2.2.4「アセンブラプログラムとの結合」を参照して下さい。

CodeWarrior 使用時も、引数・戻り値などのレジスタの引渡し規則及び戻り値の規則は、ビットフィールドを除いて日立 SHC / C++と同じです。

9.1.2 ライブラリのレジスタ使用

(1) 初期設定について

アプリケーションの起動時には以下のように設定をします。

- R0 ~ R7 が R0_BANK1 ~ R7_BANK1 に対応します。バンクレジスタは、R0_BANK0 ~ R7_BANK0 に対応します。
- FPSRC= 0x00040000 (FPSRC.FR = 0, FPSRC.SZ=0) で初期化されます。
- FR0 ~ FR15 は FPR0_BANK0 ~ FPR15_BANK0 に対応します。

(2) ライブラリのバンクレジスタ使用

- ライブラリは汎用バンクレジスタ (R0_BANK0 ~ R7_BANK0) を使用していません。アプリケーション開発者に開放されています。
- バンク浮動小数点レジスタ (FPR0_BANK1 ~ FPR15_BANK1) は、ライブラリが使用しています。ユーザーは、破壊してはいけません。FPRn_BANK1 をユーザーが使用する場合、使用前に保存し、ライブラリ呼び出し前に復帰して下さい。
- ライブラリを呼び出す場合は、FPSRC.SZ=0 FPSRC.FR=0 でなければなりません。

(3) 割り込み時に保存されるレジスタ

セガライブラリでは割り込み発生時に、以下のレジスタが割り込みの前後で保存・復帰がされます。

SSR、SPC、PR、GBR、VBR、MACH、MACL、R0 ~ R15、SR、FPUL、FPSRC、FR0 ~ FR15、XFR0 ~ XFR15

9.1.3 その他注意事項

Dreamcast で使用されている CPU (SH4) のハードウェアマニュアルは公開されていますが、アプリケーションが独自に (セガライブラリを経由することなく) CPU のハードウェアリソースを使用した場合の動作は保証されません。

必ずライブラリで提供されている機能を使用するのみとし、独自に CPU のハードウェアリソースを操作することは絶対に行わないで下さい。

9.2 C++の使用時の注意

C++言語を使用する際に、注意する点などについて説明します。

9.2.1 グローバルオブジェクトのコンストラクタの初期化

(1) 関数呼び出しによるグローバルオブジェクトのコンストラクタの初期化

グローバルオブジェクトとは、関数外に定義されたクラスのオブジェクトです。一般に（PC上の環境では）これらのオブジェクトのコンストラクタは、main（）関数が呼ばれる前に（例えばスタートアップで）実行されます。

セガライブラリ環境下では、グローバルのコンストラクタは明示的に関数を呼び出すことにより行います。これは、セガライブラリがヒープ領域の設定をユーザー側で行えるようになっており、この設定後でなければコンストラクタの実行が正しく動作しないことが発生するためです。

(2) ユーザーのグローバルオブジェクト及びライブラリのグローバルオブジェクト

グローバルオブジェクトには、ユーザーソースの中で記述する（ユーザーグローバルオブジェクト）とセガライブラリが使用している（ライブラリグローバルオブジェクト）があります。

ユーザーグローバルオブジェクトのコンストラクタの呼び出しは、ユーザーがC++言語を使用し、かつグローバルのオブジェクトを作成した場合に必要です。各コンパイラは、これを行うための関数（SHC Ver5.1はセガライブラリとして提供）を準備しています。

ライブラリグローバルオブジェクトは、ユーザーがC++言語を使用するか否かに関係なく呼び出す必要があります。

また、セガライブラリ自体は、SHCを使用して記述されていますので、CodeWarrior,GNUのコンパイラを使用している場合は、関数呼び出しのほか、リンカコマンドファイルにライブラリのコンストラクタが存在するセクションの先頭及び終了アドレスを取得する必要があります。

これは、ライブラリのグローバルオブジェクトのコンストラクタを呼び出すのを、DSG_INIT_セクションに存在するアドレスを関数ポインタとして呼び出す事により、実現するからです。

オブジェクトの種類 コンパイラ	ユーザーオブジェクト	ライブラリオブジェクト
SHC Ver5.1	syStartGlobalConstructor	関数で、両方のオブジェクトを初期化
SHC Ver6.0AD	_CALL_INIT	syStartGlobalConstructorSegaLib
CodeWarrior	__call_static_initializers	リンカコマンドファイル及びユーザー
GNU	__do_global_ctors	

これらをまとめたものを sbinit2.c で、syStartGlobalConstructorAlt()関数で、定義してます。
この関数を呼び出す事で、ユーザーオブジェクト・ライブラリオブジェクトいずれも初期化することができます。

```
/* Global constructor/destructor (C++ using) setting */
/* Please define your compile system */
/*   Nothing   : Hitachi SHC V5.x */
/*   __HITACHI__ : Hitachi SHC V6.x~ */
/*   __MWERKS__ : Metrowerks CodeWarrior */
/*   __GNUC__   : GNU */

/*-----
C++ global constructor support for All environment
-----*/
void syStartGlobalConstructorAlt ( void )
{
#ifdef __HITACHI__
    syStartGlobalConstructorSegaLib();
    _call_init();
#else
    #if defined(__MWERKS__) || defined(__GNUC__)
        void (*pf)();
        Uint32 *adr;

        for( adr = (Uint32 *) (&_START_DSG_INIT_);
            adr < (Uint32 *) (&_END_DSG_INIT_);
            adr++ ){
            if( !( ( *adr == 0x00000000) || ( *adr == 0xFFFFFFFF) ) ){
                pf = (void(*)())( *adr );
                (*pf)();
            }
        }
    #endif
    #ifdef __MWERKS__
        __call_static_initializers(); /* For CodeWarrior */
    #endif /* __MWERKS__ */
    #ifdef __GNUC__
        __do_global_ctors(); /* For GCC */
    #endif /* __GNUC__ */
    #else
        syStartGlobalConstructor(); /* For SHC V5.1 (Default) */
    #endif /* __MWERKS__ or __GNUC__ */
#endif /* __HITACHI__ */
}
```

- CodeWarrior (R3) での例
デフォルトのリンカコマンドファイルは、既に対応済みです。

a. KEEP_SECTION 内の編集

KEEP_SECTION に DSG_INIT_,DSG_END_ セクションを追加し、

```
KEEP_SECTION{
    IP,
    DSGLH,
    DSGLE,
    PSGSFD01,
    exception,
    BGAP,
    DSG_INIT_, # この2つのセクション名が含まれている事を確認する。
    DSG_END_   #
}
```

b. セグメント・シンボルの追加

DSG_INIT_セクションと、DSG_END_セクションの開始・終了アドレスをプログラムから参照できるようにする。

```
... (前略)...
..data_sg_init :
{
    ALIGNALL(0x04);
    __START_DSG_INIT_ = . ;
    GROUP(ROOT)(DSG_INIT_)
    __END_DSG_INIT_   = . ;

    ALIGNALL(0x04);
    __START_DSG_END_ = . ;
    GROUP(ROOT)(DSG_END_)
    __END_DSG_END_   = . ;

    . = ALIGN(0x04);
} > .data_sg_init
```

10 オーバーレイ

ここでは、オーバーレイについての説明や手法について説明します。

10.1 オーバーレイの手法

オーバーレイは、メインメモリに入りきれないような大きなプログラムを実行するときに、これをいくつかのブロックに必要なに応じて分け、ディスクから呼び出して実行する場合の手法のことをいいます。

アプリケーションプログラムの一部分を GD-ROM に格納しておき、必要なに応じてそれをロードして使用することによって、全てのプログラムを常時メモリに置いておく必要がなくなります。

ここでは、常にメモリに存在するプログラムを常駐部、GD-ROM からロードされるプログラムを非常駐部と呼び、常駐部が非常駐部を必要に応じてロードする実行の手法について説明します。

オーバーレイされる非常駐部のプログラムは固定アドレスでリンクされているものとし、DLL (ダイナミックリンクライブラリ) のように動的なアドレスの解決は行いません。

10.2 オーバーレイ全般に対する注意

(1) メモリ管理

オーバーレイを行う際はメモリマップを考慮し、必要な領域が破壊されないように気をつけて下さい。

通常のセガライブラリ環境では、`sbInitSystem ()` 関数でアプリケーションプログラムの末尾 (`_BSG_END : B` セクションの末尾を定義したシンボル) からメモリの最後 (`0x8CFFFFFF` 番地) までは、`syMalloc ()` 関数などが使用するヒープ領域として確保するようになっています。`sbInitSystem ()` を変更し、オーバーレイによるプログラムのロードされる領域を確保して下さい。

(2) キャッシュの整合性

非常駐部のバイナリプログラムを GD-ROM から読み込む場合、DMA 転送を使用してメモリにロードされているため、読み込んだプログラムコードと CPU のインストラクションキャッシュメモリのコヒーレンシが保証されません。

このためアプリケーションではバイナリプログラムを GD-ROM から読み込んだ後、インストラクションキャッシュ (IC) を無効化する必要があります。具体的には、非常駐部を読み込み後、`syCacheICI ()` 関数をコールします。

(3) 非常駐部の初期化

非常駐部をロードした時に、非常駐部のプログラムの呼び出しを行う前に、非常駐部側の初期化を行う必要があります。

非常駐部の初期化として、BSS セクション (= 未初期化変数の領域 = B、BSG、BSG32 セクション) のクリアを行う必要があります。

通常のセガライブラリ環境では、BSS セクションの初期化はスタートアップモジュール (`strt1.obj`、`strt2.obj`) が行っていますが、非常駐部の BSS セクションは非常駐部をロードするたびにアプリケーションで行う必要があります。

また、非常駐部を作成するときには非常駐部のセクションの配置や、リンクアドレスを標準環境から変更して下さい。

参 照 詳しくは、「1.5 セクション及びセクション名について」をご参照下さい。

また、ロードするたびに非常駐部の静的変数やグローバル変数は初期化されてしまいます。

その場合はユーザープログラムだけでなく、初期化・終了を含めたセガライブラリの動作の保証ができませんので、ライブラリは常に常駐部にリンクされるようにして下さい (数学関数などの処

理がその場で完了した状態を保持しないライブラリ関数は、非常駐部側にリンクをしても問題がない場合があります）。

10.2.2 メニュー＆ローダ

メニューやローダについて厳密な意味でオーバーレイとはいえませんが、最初に簡単な例を示します。

(1) メニュー＆ローダの概要

a. 常駐部

常駐部は、メニューと非常駐部のローダ機能（+ ）を持つプログラムとして、非常駐部側を単独で動作できる（例えば各シーンの）プログラムにします。

入力に従って必要な非常駐部のプログラムを GD-ROM から読み込み、常駐部の終了処理を終えたあとに非常駐部側の先頭にジャンプします。

b. 非常駐部

非常駐部は、独自でライブラリを初期化から開始して常駐部側に戻る場合は、ライブラリ側を終了し常駐部の特定のアドレスにジャンプします。

非常駐部のみで閉じた1つのプログラムになっているのが、この方式の特徴です。

非常駐部側から常駐部へ戻る場合には、常駐部の先頭アドレス（0xAC010000、「1.4 アプリケーションが実行されるまで」を参照下さい）にジャンプすることによって、常駐部の先頭（スタートアッププログラム）から実行します。

(2) メリットとデメリット

この方式は、実現方法やデバッグが容易にできます。しかし、ライブラリや各非常駐部で共通に必要な処理についても、各プログラムがそれぞれにリンクされるため、各プログラムのサイズは大きくなります。

また、この方法だけでは常駐部～非常駐部間での互いの情報のやり取りをすることができませんので、必要な場合、特定の領域をその情報領域にするなどの工夫が必要になります。

(3) コンパイル・リンク方法

共通情報を持たせない場合は、非常駐部のリンク時に開始アドレスを、常駐部に重ならない領域に変更します。リンク時のスタートアドレスを非常駐部の置かれるアドレスに修正して、.BIN ファイルを作成します。エントリアドレスは、特に指定する必要はありません。

(4) オーバーレイの読み込み

オーバーレイでの非常駐部の読み込みに関しては、他の方式と同様にキャッシュの整合性をとる必要があります。

(5) その他

スタートアップ時にはスタックの設定を行っていません。そのため、非常駐部から常駐部へ戻る場合に、必ずスタックポインタを初期状態に戻す必要があります。

10.2.3 FSY ファイルを用いたオーバーレイ

FSY ファイルを使用したオーバーレイの例を示します。

これは、日立 SHC コンパイラのリンカを使用した場合のみ適用できます。CodeWarrior のリンカには、-fsymbol オプションに相当する機能がないからです。

(1) コンパイル方法

オーバーレイを行う時に、非常駐部と常駐部が互いに関数を呼び出したりグローバル変数にアクセスする必要が生じます。

常駐部及び非常駐部をコンパイルする時に、互いに関数やグローバル変数のアドレスの解決を、リンカの作成する FSY ファイルを用いて行うことができます。

FSY ファイルは、リンカに-fsymbol オプションを付けた場合に出力されるファイルで、指定したセクションのシンボル名とアドレスを全てアセンブラの.EQU 擬似命令で結び付けた、アセンブルソースファイル（シンボル情報）になっています。

そこで、以下の手順で常駐部及び非常駐部のコンパイルを行って、お互いに関数やグローバル変数にアクセスできるようにします。

- a. 以下のものを常駐部に存在するように関数テーブルなどで宣言する。
 - (常駐部では使用しない) 非常駐部間で共用する関数
 - (常駐部では使用しない) グローバル変数や静的変数を持つ関数
 - ライブラリ関数
- b. 常駐部で使用する「非常駐部に存在する関数やグローバル変数」のアドレスを、ダミーで解決するシンボル情報（非常駐部のシンボルを.EQU 命令で 0 に定義する）を用いて、常駐部をコンパイルしリンクする。

常駐部をリンクする際に常駐部のシンボル情報を-fsymbol オプションで出力しておく。
- c. 非常駐部を、で出力した常駐部のシンボル情報と一緒にコンパイル、リンクする。この時に、常駐部で使用する非常駐部のシンボル情報を-fsymbol オプションで出力しておく。
- d. で出力した非常駐部のシンボル情報を使用して、再び常駐部をコンパイルする。

(2) オーバーレイモジュールの読み込み

非常駐部を呼び出す前に、通常のファイルと同様に GD ファイルシステムの機能を使用して、メモリ上に読み込みます。読み込んだ後に、キャッシュの整合性をとるために、syCacheICI () 関数を使用してインストラクションキャッシュのインバリデーションを行います。

非常駐部の BSS セクションのクリアを行ってから非常駐部の関数をコールします。

(3) デバッグ方法

常駐部のソースコードデバッグは、常駐部の elf ファイルを使用して通常通り行えます。

ロードした非常駐部のソースコードデバッグを行うには、非常駐部もデバッグ情報付きでコンパイルしておき、非常駐部のロード後に

Codescape の File->Load Program File->Load Options->Load Symbols/Debug only を用いて、非常駐部のデバッグ情報のみをロードすることによって行えます。

注 意 現在、常駐部と非常駐部の両方のデバッグ情報を同時に読み込むことはできません。

(4) その他

オーバーレイのプログラムで常駐部を変更した場合、常駐部のシンボル情報が変更されますので、非常駐部を、新しい常駐部のシンボル情報で再ビルドする必要があります。

また、非常駐部を変更した場合も、その非常駐部を使用している常駐部は新しい非常駐部のシンボル情報で再ビルドする必要があります。

10.2.4 関数テーブルを用いたオーバーレイ

ここでは、関数テーブルを使用したオーバーレイの例を示します。

(1) 関数テーブル

非常駐部と常駐部が互いの関数の呼び出しを実現するために、非常駐部及び常駐部がそれぞれ相手にも利用できるような関数のアドレスを、メンバとするテーブルに作成します。

そして、オーバーレイを行って非常駐部をロードした時に、非常駐部と常駐部でテーブルの先頭アドレスを互いに取得します。

非常駐部と常駐部の間にまたがる関数呼び出しを行う場合は、全てこのテーブルを使用した間接参照で行います。

これにより、互いに関数の実体のアドレスを取得せずにそれぞれをリンクすることが可能になります。

● 常駐部

```
typedef struct {
    void (*Mlib_func1) ( void );
    void (*Mlib_func2) ( void );
} OVERLAY_MAIN_TABLE;
OVERLAY_MAIN_TABLE mTbl = {
    lib_func1,
    lib_func2
};
LoadOverlay ( &mTbl ); /* 非常駐部をロード後、mTbl を引数として非常駐部に渡す */
...
```

● 非常駐部

```
sub_module_main ( OVERLAY_MAIN_TABLE *MT ) {
    (MT->Mlib_func1) (); /* lib_func1 の呼び出し */
    (MT->Mlib_func2) (); /* lib_func2 の呼び出し */
    ...
}
```

(2) FSY を使った方法に対するメリットデメリット

a. メリット

- 非常駐部のリンク時に常駐部の関数のアドレスが分からなくてもよい場合、常駐部に変更があった場合でも非常駐部のリコンパイルの必要がありません。また、逆に非常駐部を変更した場合も同様です。
- 日立 SHC に特化した方法でないため、CodeWarrior 環境でも使用できます。

b. デメリット

- 非常駐部と常駐部の間の関数呼び出しを全てテーブルを用いて行うため、関数呼び出しのたびにオーバーヘッドが発生します。関数呼び出しの部分のコードが複雑になります。
- 関数テーブルを作成する時に、使用する関数すべてを一括して宣言しなければなりません。関数テーブルの分のメモリを消費します。

(3) その他

オーバーレイモジュールの読み込み方法やデバッグ方法は FSY を使った方法と同様に行うことができます。常駐部や非常駐部の関数テーブルに変更があった場合は新しい関数テーブルを使用しての全体のリコンパイルが必要になります。

10.2.5 CodeWarrior によるオーバーレイ

CodeWarrior for Dreamcast の Release2 以降では、オーバーレイのサポート機能が標準で組み込まれます (Release1 には組み込まれていません)。

オーバーレイのプロジェクトでは、通常行う常駐部のソースファイルの設定の他に、次の設定を行います。

- 非常駐部の開始アドレスを指定します。
- 非常駐部にロードされる、各プログラムの、ソースファイルを指定します。

常駐部 非常駐部間の関数呼び出しは、通常の間数呼び出しとして記述するだけでよく、特別な制約はありません。

これにより、常駐部の ELF ファイルと非常駐部の各バイナリプログラムファイル (64 バイトのヘッダ付き) が生成されます。

CodeWarrior (CodeScape も同様) デバッガは、非常駐部側に置かれたブレークポイントも記憶してるため、効率的なデバッグが可能になります。

参 照 詳細は、CodeWarrior に付属するドキュメント「CodeWarrior Targeting Dreamcast」を参照して下さい。

11 Shinobi 1 から 2 への主な変更点

ここでは Shinobi1 から Shinobi2 に変更した内容を解説します。

11.1 システム全体に関する変更点

(1) MMU モード

MMU モードを MMU OFF から MMU ON に変更しました。これはメモリの不正アクセスを検出を行いやすくするためです。なお、MMU ON とすることによる性能の劣化はありません。

a. 従来アプリケーションからの変更

MMU の TLB は、ストアキューアクセス用に設定されます。

セガライブラリ環境下で動作するアプリケーションは、CPU (SH4) の P1 領域で動作します。この領域は、MMU モードが ON/OFF に関わらず MMU のアドレス変換は行いません。

参 照 「SH7091 プログラミングマニュアル 3.3 メモリ空間」を参照して下さい。

従って、通常の動作では、MMU のモードの変更を意識することはありません。

b. 不正なアクセスの検出

例えば、0 番地アクセス (P0 領域: 0x00000000) をした場合、CPU はアドレス変換をしようと TLB を参照しますが、TLB には、この番地に対応するアドレスはありません。そこで、CPU は例外を発生し TLB の入れ替えを要求しますが、セガライブラリではこの例外を処理しません。本来このようなアクセスは起きないからです。

結果として、未定義例外となり、CodeScape などのデバッガが、このアクセスの発生したところでプログラムをストップさせます。これにより不正アクセスの箇所がわかりデバッグの効率の向上が期待できます。

なお、P0、P3 領域は上記のように不正アクセスの検出ができますが、その他の領域例えば P1 領域の 0x80000000 にアクセスしても上記の検出はできません。

(2) GBR レジスタ

Shinobi2 から CPU の GBR (グローバルベースレジスタ) をライブラリで使用します。

Shinobi2 からは、システムマネージャライブラリ (syMng) の提供により、複数のファイバ (非常に原始的なスレッドと考えて下さい) を使用できるようになりました。GBR レジスタは各ファイバの常に、現在のファイバのコンテキストエリアの先頭をポイントするために使用します。

なお、GBR レジスタは C の組込み関数 `set_gbr` 等で更新したり、アセンブラプログラムで GBR レジスタを書きかえれば変更されますが、これら以外で通常 C/C++ でプログラミングしている限りでは、更新されることはありません。

(3) メモリマップ

メモリマップが若干変更になりました。スタック領域が 1Kbyte 少なくなり 12Kbyte になります (0x8c00:c000 ~ 0x8c00:f000)。

(4) セクション定義

プログラムの属性 (コード、データ等の) 配置を決定するセクションの順も Shinobi1 から変更しました。なお、この変更はアプリケーションプログラムには、ほとんど影響がありません。

11.2 追加及び新規公開ライブラリ

(1) コールバックライブラリ

コールバックライブラリは、各種ハードウェア割り込みに、ライブラリユーザーがコールバック関数を登録できるライブラリです。

1つの割り込みに対して、複数のコールバック関数を登録することができ、その実行順序はハンドラ登録時に優先度を設定することにより制御できます。

コールバックライブラリから呼び出される関数は、割り込み中で動作することになりますので、注意して使用して下さい。

- 登録できる割り込みの種類

VBALNK-IN、HBLANK、End of Render、GD の読み込み終了等です。

- 主な関数

関数名	機 能
syCallbackInit	ライブラリの初期化
syCallbackFinish	ライブラリの終了
syCallbackAddHandler	コールバック関数の登録
syCallbackDeleteHandler	コールバック関数の削除

(2) マウスライブラリ

マウスライブラリは、コントロールポートに接続されるマウスを制御するためのライブラリです。

- 主な関数

関数名	機 能
pdMouseInit	ライブラリの初期化
pdMouseExit	ライブラリの終了処理
pdMouseGetButton	ボタン情報の取得
pdMouseGetAxis	座標軸情報の取得
pdMouseGetInfo	マウス情報の取得
pdMouseIsReady	マウスの接続確認
pdMouseExecServer	サーバ関数

(3) メモリコピー / メモリセットライブラリ

従来、Ninja ライブラリに含まれていたメモリコピーライブラリ (njMemCopy??) を、システムライブラリである shinobi2.lib に移動しプレフィックスを変更しました。

また、メモリセットライブラリを新規に追加しました。

- 主な関数 (メモリコピーライブラリ)

関数名	機 能
syMemCopy	メモリコピー (バイト単位)
syMemCopy2	メモリコピー (ワード単位)
syMemCopy4	メモリコピー (ロング単位)
syMemCopy8	メモリコピー (8 バイト単位)
syMemCopy16	メモリコピー (16 バイト単位)
syMemCopy32	メモリコピー (32 バイト単位)
syMemCopySQ	メモリコピー (ストアキューを使用)

● 主な関数（メモリセットライブラリ）

関数名	機 能
syMemSet	メモリセット（バイト単位）
syMemSet2	メモリセット（ワード単位）
syMemSet4	メモリセット（ロング単位）
syMemSet8	メモリセット（8 バイト単位）
syMemSet16	メモリセット（16 バイト単位）
syMemSet32	メモリセット（32 バイト単位）
syMemSetSQ	メモリセット（ストアキューを使用）

(4) ビデオケーブルライブラリ（syVideo）

ビデオ関連の設定や状態の取得を行なうライブラリです。従来のケーブルライブラリを廃止しその代わりに使用するライブラリです。

- ビデオモード（VGA, NTSC, PAL）の取得
- 放送形式/接続ケーブルの取得
- ボーダーカラーの設定

このライブラリで取得する画面モードは、起動時ブートルームが設定した内容*であり、ユーザーがアプリケーション起動後に接続ケーブルを変更した場合でも、アプリケーションでの画面設定は、本体起動時と同じになります。

● 主な関数

関数名	機 能
syVideoInit	ビデオモード関数及び放送形式関数の初期化
SyVideoExit	ビデオモード関数、放送形式関数の終了.
SyVideoGetBroadcastType	Dreamcast 本体の放送形式を返します。
SyVideoGetCableType	Dreamcast 起動時に接続されていたケーブルの種類を返します。
SyVideoGetVideoMode	現在設定されているビデオモードを返します。
SyVideoSetBorderColor	ボーダーカラーの設定
SyVideoWaitVBlankSetBorderColor	ボーダーカラーの設定

(5) ファイバライブラリ（syFbr）

従来、ライブラリのためのライブラリ（内部ライブラリ）であったものを公開しました。

ファイバライブラリは、非常に原始的なスレッド（ファイバ）のライブラリで、ほとんど全てのレジスタとSQを待避復帰することにより、マルチスレッドのサポートを行います。

他のライブラリ（MPEG Sofdec 等）が使用することが前提になっており、制限事項が多くあります。

- 1つの割り込みで切りかえられる先は1つ。
- 多重割り込みが発生したときは最初に発生した割り込みで切り替わる。

注 意 また、セガライブラリはリエントラントに設計されていません。従って本ライブラリの使用は十分注意して利用して下さい。

通常は、次のシステムマネジメントライブラリを使用することをお勧めします。

● 主な関数

関数名	機 能
syFbrConvertThreadToFiber	現在実行中の関数をファイバ（メインファイバ）に変換
syFbrCreateFiber	ファイバの生成
syFbrDelayedSwitchToFiber	割り込みを契機としたファイバの切り替え
syFbrDeleteFiber	ファイバの削除
syFbrGetCurrentFiber	実行中のファイバハンドルを取得
syFbrGetFiberData	実行中のファイバのファイバデータを取得
syFbrGetMainFiber	メインファイバのハンドルを取得
syFbrInitEx	ファイブライブラリの（拡張）初期化
syFbrSwitchToFiber	ファイバの切り替え

(6) システムマネージャライブラリ（syMng）

システムマネージャライブラリは、ファイバを生成・切替を管理します。

ユーザーはシステムマネージャライブラリを使用して、ファイバに関数を登録できます。

システムマネージャライブラリによる、ファイバの切り替えの仕様は、syMngInit()関数の関数リファレンスを参照して下さい。

● 主な関数

関数名	機 能
syMngInit	システムマネージャの初期化
syMngFinish	システムマネージャの終了
syMngEntrySfbrFunc	サブファイバ関数の登録
syMngDeleteSfbrFunc	サブファイバ関数の削除
syMngEntryWaitVsyncCallback	Vsync 待ちコールバック関数の登録
syMngGetCurFlvl	カレントファイバレベルの取得
syMngGetFbr	ファイバハンドルの取得
syMngSetSfbrTopLvl	トップレベルファイバの設定
syMngSetTmrDur	タイマファイバの実行時間の設定
syMngSleep	次のレベルのサブファイバへのスイッチ
syMngSwitch	サブファイバへのスイッチ
syMngDelayedSwitch	割り込み処理内からのサブファイバへのスイッチ

(7) ストアキューライブラリ（sySq）

Shinobi2 より MMU が ON になります。通常 SHC の P1 領域で動作するアプリケーションはこれを意識する必要はありませんが、ストアキュー（SQ）へのアクセスする場合は仮想アドレスで参照する必要があります。ストアキューライブラリでは、このアドレス変換を行います。

なお、本ライブラリの初期化・終了処理は syHwInit の内部で行われますので、ユーザーがコールする必要はありません。

なお、ストアキューへのアクセスはライブラリ等の整合性をとるために慎重に行う必要があります。

参 照 MMU のマッピング仕様は、shinobi2 資料編を参照して下さい。

● 主な関数

関数名	機 能
sySqStart	物理メモリに割当てられた SQ 用の仮想アドレスを返す
sySqEnd	SQ 用の仮想アドレスに割当てられた物理メモリのアドレスを返す

(8) 低レベルシリアルライブラリ (syCom)

SH4 内蔵 SCIF (FIFO 内蔵シリアルコミュニケーションインタフェース) を利用したシリアル通信を可能にするライブラリです。

● 主な関数

関数名	機 能
syComInit	ライブラリの初期化
syComExit	ライブラリの終了
syComOpen	通信経路のオープン
syComClose	通信経路のクローズ
syComPutWait	データを 1 バイト送信
syComIsPut	データが送信可能かどうか調べる
syComPutq	データを 1 バイト送信 (ウェイトなし)
syComIsGet	受信バッファにデータがあるかどうか確認
syComGet	受信バッファから 1 バイトデータを取り出す

11.3 削除されたライブラリ

(1) ビデオケーブルライブラリ (syCbl)

ビデオケーブル種別の参照の方法を変更したため、ケーブルライブラリは削除しました (既存のサンプルプログラムが、他のライブラリの未初期化の状態でケーブルライブラリの関数を使用していたため、あえて削除しました)。

● 削除関数と代替関数

削除関数	代替関数
syCblCheck	syVideoGetVideoMode (VA/NTSC/PAL/RESERVED)
syCblCheckBroadcast	syVideGetBroadcastType (NTSC/PAL/PAL-M/PAL-N)
syCblCheckCable	syVideoGetCableType (VGA/VBS,S,RF/RGB)

11.4 既存ライブラリの変更点

(1) キャッシュライブラリ (syCache)

CPU (SH4) のキャッシュモードの取得・変更が可能な関数 (マクロ) を追加しました。

- 追加関数

関数名	機 能
syCacheGetForm	キャッシュモードの取得
syCacheSetForm	キャッシュモードの更新

一般にキャッシュライブラリをユーザーが、操作する必要はありません。

(2) タイマライブラリ (syTmr)

割り込み発生までのカウント指定関数 (syTmrGenSetIntCount) の追加をしました。

syTmrGenSetIntCount(n)は、syTmrGenSetCount(-n)と等価です。

syTmrSetCount, syTmrSetIntCount で設定した場合のカウントの変化を示します (内部カウンタは、CPU の内部のカウンタです)。

a. 割り込み発生までのカウント指定 (新規)

- 例:

```
syTmrGenSetIntCount(5)
内部カウンタ      5, 4, 3, 2, 1, 0 (割り込み発生), -1, -2, -3, ...
syTmrGenGetCount  -5, -4, -3, -2, -1, 0, 1, 2, 3, ...
```

b. 割り込み発生までのカウント指定

- 例:

```
syTmrGenSetCount(-5)
内部カウンタ      5, 4, 3, 2, 1, 0 (割り込み発生), -1, -2, -3, ...
syTmrGenGetCount  -5, -4, -3, -2, -1, 0, 1, 2, 3, ...
```

c. フリーランタイマのカウント指定

- 例:

```
syTmrGenSetCount(5)
内部カウンタ      -5, -6, -7, -8, -9, -10, -11, -12, ...
syTmrGenGetCount   5, 6, 7, 8, 9, 10, 11, 12, ...
```

- 追加関数

関数名	機 能
syTmrGenSetIntCount	割り込み発生までの時間を設定
syTmrFinish	タイマライブラリを終了

(3) GDFS ライブラリ (gdFs)

ライブラリの初期化処理を、ライブラリ内のワークエリアの初期化処理 (gdFsInit2) と、GD-ROM のルートディレクトリのマウント処理 (gdFsMount) に分離しました。

これに伴い従来の初期化関数 (gdFsInit) を廃止しました。

これは、初期化処理の中で GD-ROM にアクセスする関数を分離し、処理時間を要する部分がわかるようにしました (従来 sbInitSystem の中で gdFsInit を呼び出していたため、これをそのまま利用したプログラムでは、GD-ROM の初期化中、真っ暗な画面を表示し続けてしまうことがありました)。

前記の対応として、再初期化処理 (gdFsReInit) を再マウント処理 (gdFsReMount) に変更しました。

● 削除関数と代替関数

削除関数	代替関数
gdFsInit	gdFsInit2, gdFsMount
gdFsReInit	gdFsReMount

(4) ペリフェラルデータ取得ライブラリ (pdPad)

● njPrintPeripheralInfo 関数の廃止しました。

ライブラリ間の独立性を高めるため、Ninja ライブラリを前提とする関数を廃止しました。

● pdGetPeripheralError 関数の廃止しました。

過去の開発機 (DevBox 以前) では、この関数の意味がありましたが、現在の DevBox ではこの関数がエラーを返すことはありません。

このため廃止しました。

● 削除関数と代替関数

削除関数	代替関数
njPrintPeripheralInfo	sample¥peri¥print¥main.c のに実装例があります。 (PrintPeripheralInfo())
pdGetPeripheralError	廃止

(5) システムライブラリ (syHw)

ハードウェア初期化関数は syHwInit2 関数は削除され、syHwInit のみにになりました。

syHwInit 関数は、引数をとるように変更しました。この引数には、割り込み専用スタック (割り込みコンテキストで動作するプログラムが使用するスタック) の「底のアドレス」を指定します。

通常は、sbinit2.c の設定をそのままご使用下さい。

● 例

```

Uint32 gIntStack[20*1024/sizeof(Uint32)];
...;
syHwInit(&gIntStack[20*1024/sizeof(Uint32)]);

```

shHwInit の処理では、各種低レベルライブラリ (割り込み、MMU, G2 バス、シリアル等) の初期化とともに、FPSCR レジスタの初期設定 (H'00040000) を追加しました。

(6) スタートアップ

BGAP セクション (32 バイト) を、'SEGA' 文字列でフィルします。通常 BGAP セクションは、プログラム領域とヒープ領域の境を示します。

ノート Nindows2 ライブラリでは、この文字列をチェックし、ヒープ領域があふれた場合など、この文字列が破壊されると、その旨のメッセージを「コンソールウィンドウ」に表示します。

12 補足資料

12.1 各ペリフェラル毎の固有情報

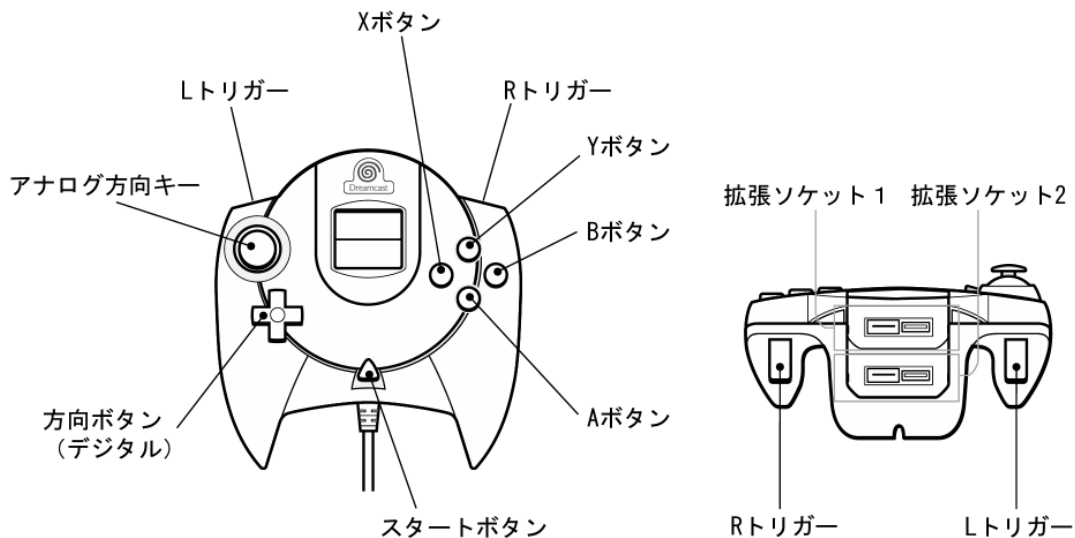
現在までに、発売されているもしくは発売予定のペリフェラルについて、各固有の仕様をそれぞれ説明します。

12.1.1 ドリームキャスト・コントローラ（HKT-7700）

ドリームキャスト商品付属のコントローラです。拡張ソケットを2つ持っています。

- ペリフェラル情報

メンバ	情 報
type	CONTROLLER
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	2
product_name	"Dreamcast Controller"
license	"Produced By or Under License From SEGA ENTERPRISES, LTD."
stdby_pow	43.0mA
max_pow	50.0mA



- 各ボタンの定義

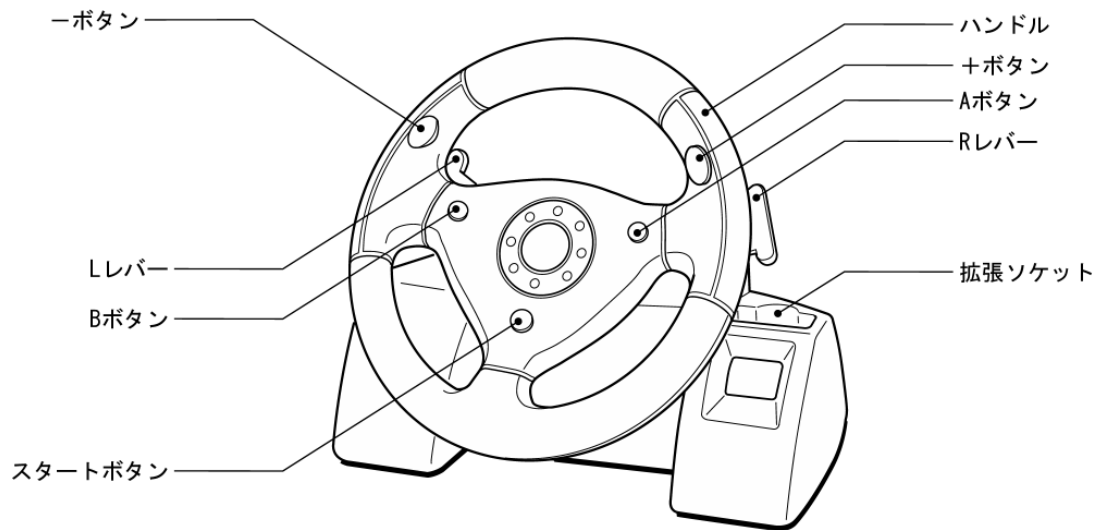
ボタンまたはトリガー	定義または値の場所
A ボタン	PDD_DGT_TA
B ボタン	PDD_DGT_TB
X ボタン	PDD_DGT_TX
Y ボタン	PDD_DGT_TY
方向ボタン	上: PDD_DGT_KU 下: PDD_DGT_KD 右: PDD_DGT_KR 左: PDD_DGT_KL
アナログ方向キー	X 方向: PDS_PERIPHERAL 構造体の x1 メンバ Y 方向: PDS_PERIPHERAL 構造体の y1 メンバ
R トリガー	PDS_PERIPHERAL 構造体の r メンバ / PDD_DGT_TR
L トリガー	PDS_PERIPHERAL 構造体の l メンバ / PDD_DGT_TL
スタートボタン	PDD_DGT_ST

12.1.2 レーシングコントローラ (HKT-7400)

車のハンドル状のデバイスを持ったコントローラです。主に、レースゲームなどに利用されます。拡張ソケットを1つ持っています。

● ペリフェラル情報

メンバ	情 報
type	CONTROLLER
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	2
product_name	"Racing Controller"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	44.0mA
max_pow	55.0mA



● 各ボタンの定義

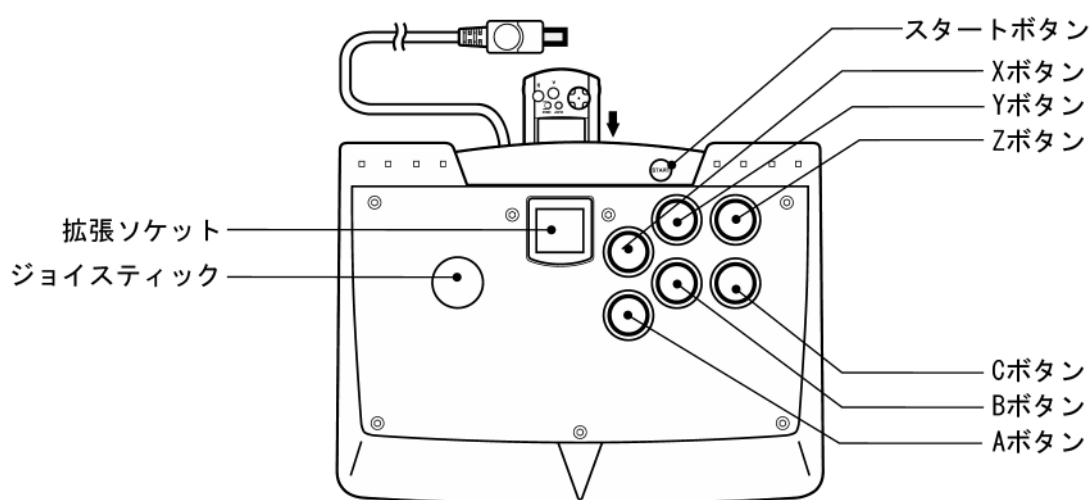
ボタンまたはトリガー	定 義
A ボタン	PDD_DGT_TA
B ボタン	PDD_DGT_TB
+ ボタン	PDD_DGT_KU
- ボタン	PDD_DGT_KD
R ボタン	PDS_PERIPHERAL 構造体の r メンバ / PDD_DGT_TR
L ボタン	PDS_PERIPHERAL 構造体の l メンバ / PDD_DGT_TL
ハンドル	右回転 : PDS_PERIPHERAL 構造体の x1 メンバ 0 ~ 127 / PDD_DGT_KR 左回転 : PDS_PERIPHERAL 構造体の x1 メンバ -128 ~ 0 / PDD_DGT_KL
スタートボタン	PDD_DGT_ST

12.1.3 アーケードスティック (HKT-7300)

アーケードゲームなどに使われるスティックを持ったコントローラーです。プレイヤーの激しい操作にも絶えるよう頑丈に作られています。拡張ソケットを1つ持っています。

- パリフェラル情報

メンバ	情 報
type	CONTROLLER
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	2
product_name	"Arcade Stick"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	17.0mA
max_pow	30.0mA



- 各ボタンの定義

ボタンまたはトリガー	定 義
A ボタン	PDD_DGT_TA
B ボタン	PDD_DGT_TB
C ボタン	PDD_DGT_TC
X ボタン	PDD_DGT_TX
Y ボタン	PDD_DGT_TY
Z ボタン	PDD_DGT_TZ
ジョイスティック	上: PDD_DGT_KU 下:PDD_DGT_KD 右: PDD_DGT_KR 左:PDD_DGT_KL
スタートボタン	PDD_DGT_ST

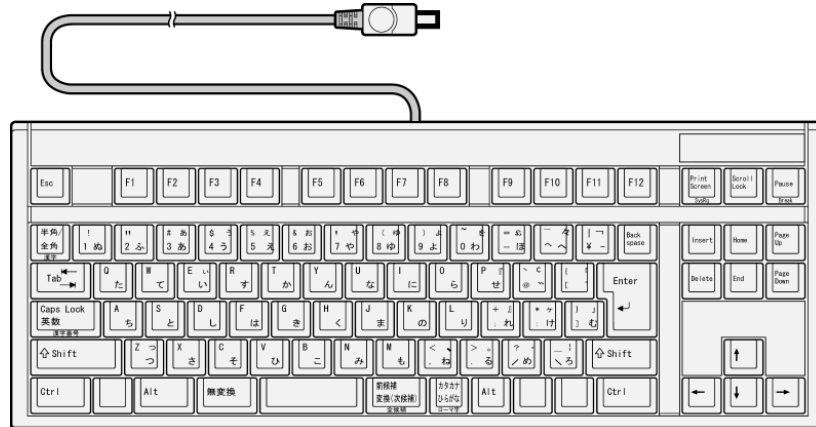
12.1.4 ドリームキャスト・キーボード (HKT-7600)

パソコン用のキーボードとほぼ同じ形状のコントローラです。コンパクトかつ安価に押さえるために、数字キー（Num キー）はありません。また拡張ソケット也没有ありません。

- ペリフェラル情報

メンバ	情 報
Type	KEYBOARD
is_root	1
area_code	PDD_DEVAREA_JAPAN
connector_dir	0
product_name	"Keyboard"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	35.0mA
max_pow	40.0mA

- ドリームキャスト・キーボード



日本用

セガライブラリ環境では、このペリフェラルを使うために専用の関数が用意されています。ただし、この関数では他のペリフェラルと同様に、キーが押されているかどうかの判定しかできません。従って、タイプしたキーの一時的な記憶や漢字変換（FEP）などは、他に用意する必要があります。

関数・定義	機 能
pdKbdGetData	キーボード押下情報取得関数
PDS_KEYBOARD	キーボードデータ構造体

1 米国及び欧州で発売されるキーボードでは、area_code にそれぞれ PDD_DEVAREA_USA、PDD_DEVAREA_EUROPE がセットされています。形状はそれぞれ違っています。

12.1.5 キティバージョンのドリームキャスト付属キーボード（HKT-7601）

キティバージョン仕様のドリームキャストに付属するキーボードです。ドリームキャスト・キーボードに比べて、キーボードの横幅が小さくなっています。

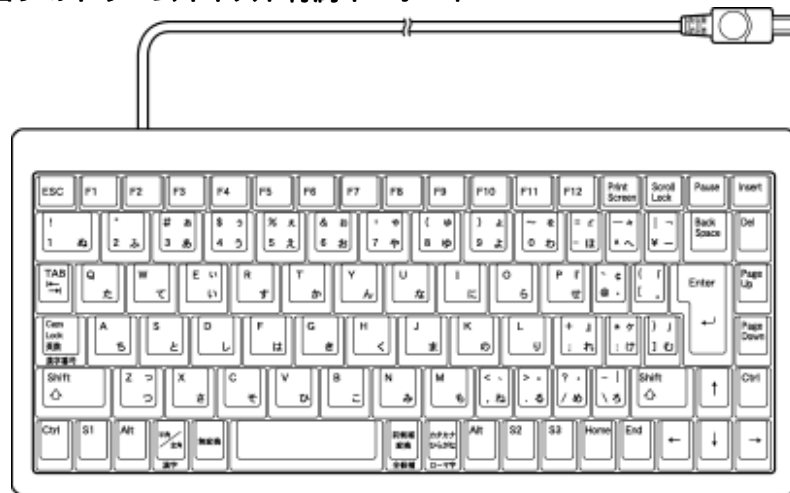
コンパクトかつ安価に押さえるために、数字キー（Num キー）はありません。また拡張ソケット也没有ありません。

● ペリフェラル情報

メンバ	情 報
Type	KEYBOARD
is_root	1
area_code	PDD_DEVAREA_JAPAN
connector_dir	0
product_name	"Keyboard"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	35.0mA
max_pow	40.0mA

注 意 キティバージョンのキーボードはペリフェラル情報を取得した際に、「ドリームキャスト・キーボード」と同様、キーの数が 92 Key という情報が返ってきます。しかし、キティバージョンのキーボードは「ドリームキャスト・キーボード」に比べて、実際にはキーが 3 つ（「Print Screen」、「右 Alt」、「右 Ctrl」）少なくなっています。また、個々のキー配置も異なりますので注意が必要です。

● キティバージョンのドリームキャスト付属キーボード



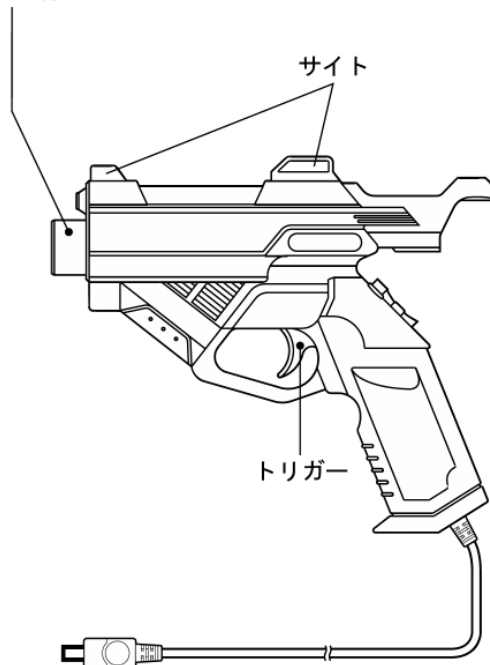
12.1.6 ドリームキャスト・ガン (HKT-7801)

主に、ガンシューティングゲーム用の入力デバイスです。拡張ソケットを1つ持っています。セガライブラリ環境では、このペリフェラルを使用するための専用の関数が用意されています。

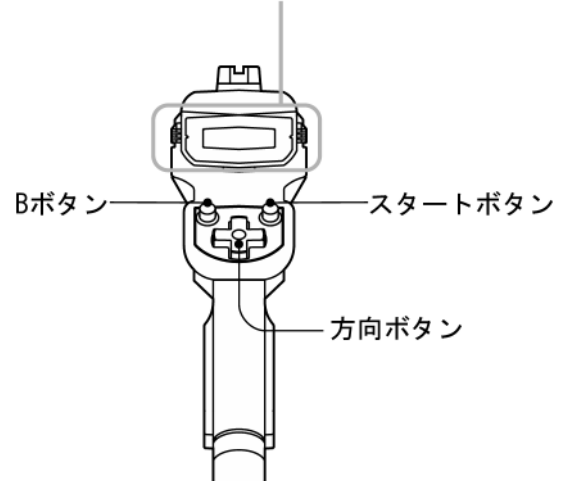
- ペリフェラル情報

メンバ	情 報
type	CONTROLLER / LIGHT-GUN
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	1
product_name	"Dreamcast Gun"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	22.0mA
max_pow	30.0mA

レンズ（銃口）



拡張ソケット



ドリームキャストのガンは、TV 画面の走査線を検出する従来の Saturn と同じ方式を採用しています。

- 各ボタンの定義

ボタンまたはトリガー	定 義
トリガー（A ボタン）	PD_DGT_TA
B ボタン	PD_DGT_TB
方向ボタン	上: PDD_DGT_KU 下:PDD_DGT_KD 右: PDD_DGT_KR 左:PDD_DGT_KL
スタートボタン	PD_DGT_ST

12.1.7 つりコントローラ (HKT-8700)

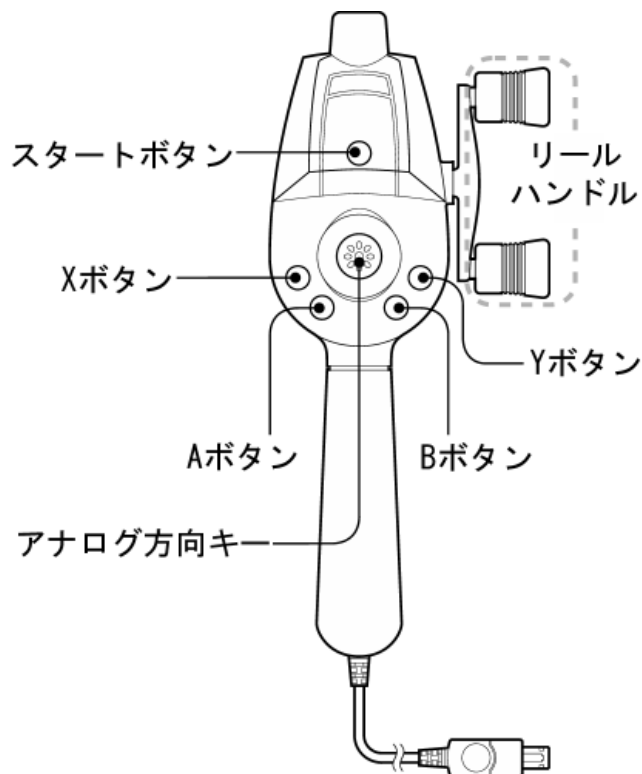
主に、つりゲームに使えるようリールを持ったコントローラです。内部に「ぶるぶるぱっく」と同じような振動デバイスを内蔵しています。振動デバイスは拡張ポート 5 番にあります。拡張ソケットはありません。

● CONTROLLER デバイス (ポート 0)

メンバ	情 報
type	CONTROLLER
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	
product_name	"Dreamcast Fishing Controller"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	60mA
max_pow	240mA

● VIBRATION デバイス (ポート 5)

メンバ	情 報
type	VIBRATION
is_root	0
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	
product_name	"Dreamcast Fishing Controller"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	60mA
max_pow	240mA



● 各ボタンの定義

ボタンまたはトリガー	定 義
A ボタン	PD_DGT_TA
B ボタン	PD_DGT_TB
X ボタン	PDD_DGT_TX
Y ボタン	PDD_DGT_TY
リールハンドル	回転速度:PDS_PERIPHERAL 構造体の r メンバ / PDD_DGT_TR
アナログ方向キー	X 方向:PDS_PERIPHERAL 構造体の x1 メンバ Y 方向:PDS_PERIPHERAL 構造体の y1 メンバ 上: PDD_DGT_KU 下:PDD_DGT_KD 右: PDD_DGT_KR 左:PDD_DGT_KL
X 移動加速	PDS_PERIPHERAL 構造体の x2 メンバ
Y 移動加速	PDS_PERIPHERAL 構造体の y2 メンバ
Z 移動加速	PDS_PERIPHERAL 構造体の l メンバ / PDD_DGT_TL
スタートボタン	PD_DGT_ST

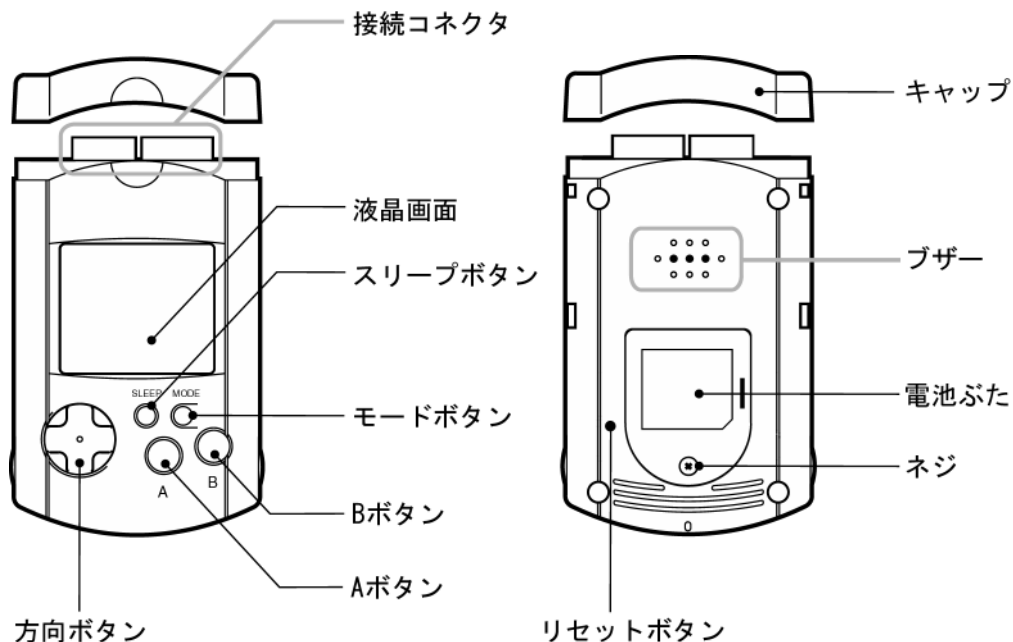
12.1.8 ビジュアルメモリ (HKT-7000)

アプリケーションのデータを保存し管理するためのメモリーカードとして使用できるペリフェラルです。

コントローラに接続しない状態では、メモリー内のデータファイルの表示や消去ができ、ビジュアルメモリどうしを2台接続することで、ファイルのコピー操作等の操作を行うことも可能です。

● ペリフェラル情報

メンバ	情 報
type	STORAGE / LCD / TIMER
is_root	0
area_code	PDD_DEVAREA_JAPAN
connector_dir	
product_name	"Visual Memory"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	12.4mA
max_pow	13.4mA



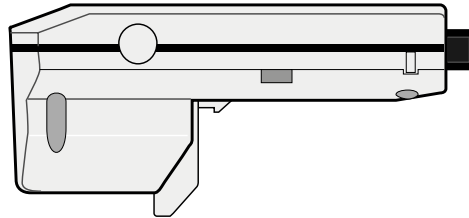
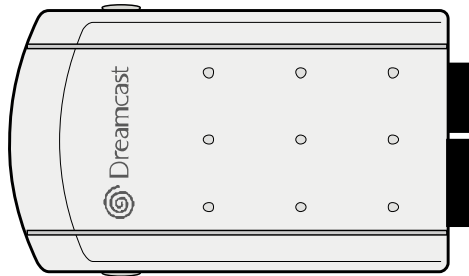
米国向け及び欧州向けのビジュアルメモリでは、area_code に PDD_DEVAREA_USA / PDD_DEVAREA_JAPAN / PDD_DEVAREA_ASIA / PDD_DEVAREA_EUROPE が設定されています。

12.1.9 ぶるぶるぱっく (HKT-8600)

振動ペリフェラルです。セガライブラリ環境では、振動ペリフェラル用の関数のほかに、このペリフェラルを使用するための専用の関数が用意されています。

- ペリフェラル情報

メンバ	情 報
type	VIBRATION
is_root	0
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	
product_name	"Puru Puru Pack"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	20.0mA
max_pow	160.0mA

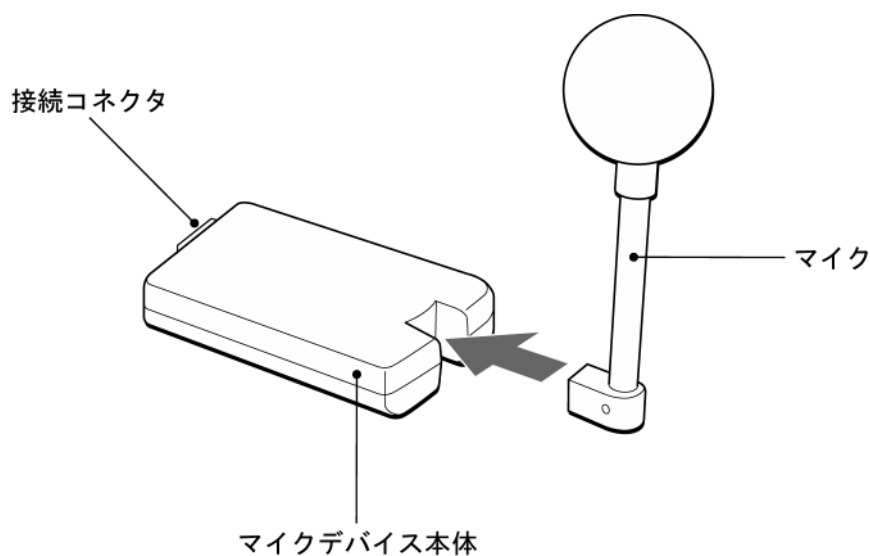


12.1.10 マイクデバイス ~ 音声認識・入力用ユニット~ (HKT-7200)

音声サンプリングペリフェラルです。先端にマイクを取りつけて使います。セガライブラリ環境では、このペリフェラルを使うための専用の関数を用意しています。

- ペリフェラル情報

メンバ	情 報
type	SOUNDINPUT
is_root	0
area_code	PDD_DEVAREA_USA / -_JAPAN
connector_dir	
product_name	"SoundInputPeripheral (S.I.P.)"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	30.0mA
max_pow	30.0mA



12.1.11 ツインスティック（HKT-7500）

弊社タイトル「電腦戦機バーチャロン」専用に作成されたペリフェラルです。

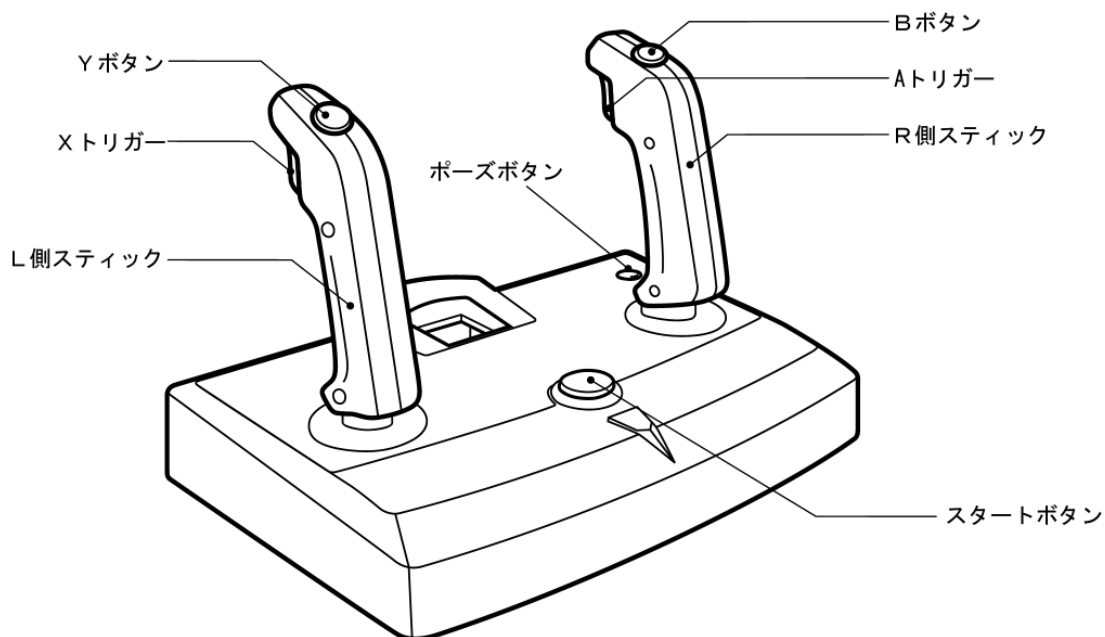
● ペリフェラル情報

メンバ	情 報
Type	CONTROLLER
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	2
product_name	"Twin Stick"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	22.0mA
max_pow	30.0mA

● 各ボタンの定義

ボタンまたはトリガー	相当するボタン（ ）	定 義
左スティック	方向ボタン 1	PDD_DGT_KU
左スティック	方向ボタン 1	PDD_DGT_KD
左スティック	方向ボタン 1	PDD_DGT_KL
左スティック	方向ボタン 1	PDD_DGT_KR
左スティックトリガー	X ボタン	PDD_DGT_TY
左スティックボタン	Y ボタン	PDD_DGT_KDB
右スティック	方向ボタン 2	PDD_DGT_KUB
右スティック	方向ボタン 2	PDD_DGT_KDB
右スティック	方向ボタン 2	PDD_DGT_KLB
左スティック	方向ボタン 2	PDD_DGT_KRB
右スティックトリガー	A ボタン	PDD_DGT_TA
右スティックボタン	B ボタン	PDD_DGT_TB
START	START ボタン	PDD_DGT_ST
PAUSE	D ボタン	PDD_DGT_TD

他のコントローラデバイスでの標準的な呼び名です。



12.1.12 マウス (HKT-9900)

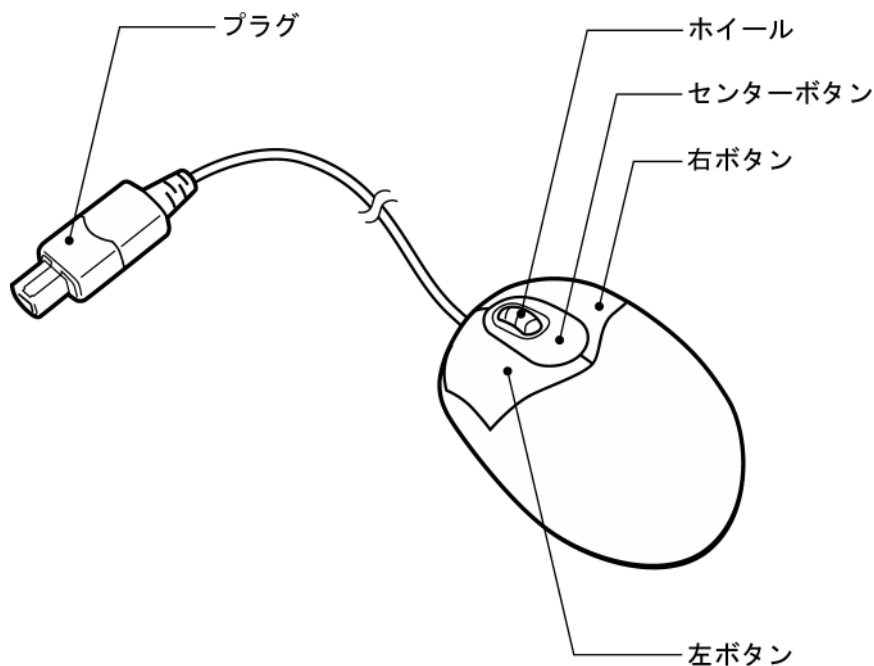
マウス対応のアプリケーション向けに作成されたペリフェラルです。

● ペリフェラル情報

メンバ	情 報
type	POINTING
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	0
product_name	"Dreamcast Mouse"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	40.0mA
max_pow	50.0mA

● 各ボタンの定義

ボタンまたはトリガー	定 義
センターボタン	PDD_MOUSE_BUTTON_M
右ボタン	PDD_MOUSE_BUTTON_R
左ボタン	PDD_MOUSE_BUTTON_L
ホイール上回転	PDD_MOUSE_BUTTON_WU
ホイール下回転	PDD_MOUSE_BUTTON_WD



12.1.13 マラカスコントローラ（HKT-9700）

弊社タイトル「サンバDEアミーゴ」専用、作成されたペリフェラルです。

アナログ XY は左手用が 1、右手用が 2 になります。

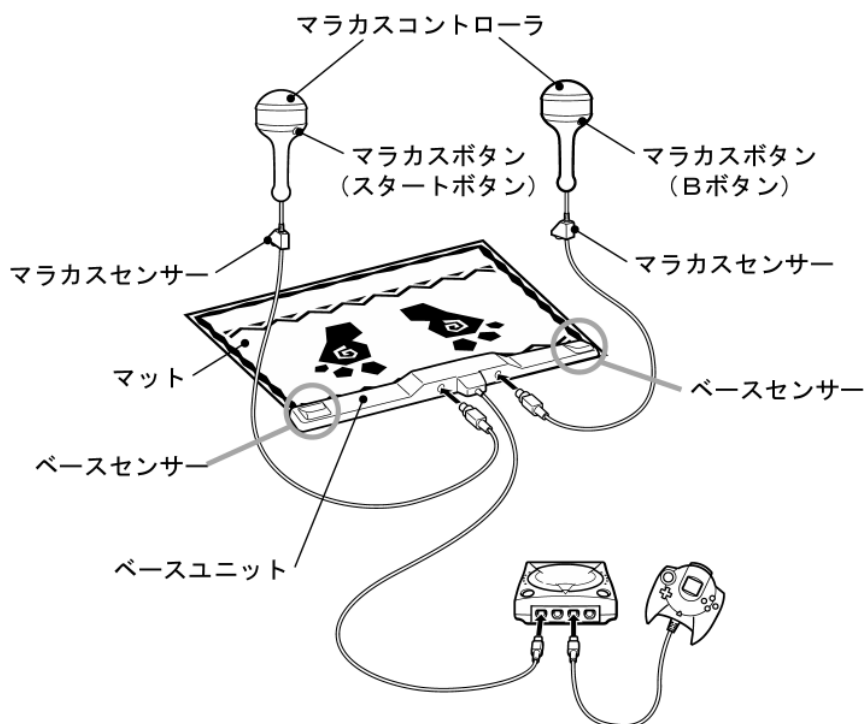
マラカスのアナログ座標の範囲は、標準コントローラとは異なります。標準コントローラの円形よりは広いですが、真四角ではなく若干横長な四角になります。

● ペリフェラル情報

メンバ	情 報
type	CONTROLLER
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	
product_name	"Maracas Controller"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	110mA
max_pow	135mA

● 各ボタンの定義

ボタンまたはトリガー	定 義
右マラカス内 (マラカスを振ると 0n)	PDD_DGT_TA
左マラカス内 (マラカスを振ると 0n)	PDD_DGT_TB
右マラカスの柄 (ボタンを押すと 0n)	PDD_DGT_ST
左マラカスの柄 (ボタンを押すと 0n)	PDD_DGT_TC
右マラカスの認識範囲 Flag (範囲外で 0n)	PDD_DGT_TD
左マラカスの認識範囲 Flag (範囲外で 0n)	PDD_DGT_TZ
左マラカスの X 座標	PDD_DEV_SUPPORT_AX1
左マラカスの Y 座標	PDD_DEV_SUPPORT_AY1
右マラカスの X 座標	PDD_DEV_SUPPORT_AX2
右マラカスの Y 座標	PDD_DEV_SUPPORT_AY2



- (2 P 同時使用時の) マラカス使用についての注意事項
1 P 専用ソフトの場合には、通常のペリフェラル用関数で処理可能ですが、2 P 同時にプレイする場合にのみ、音波の相互干渉を避けるために、専用のライブラリを使用する必要があります。

12.1.14 ドリームアイ (HKT-9400)

「ドリームアイ」は、Dreamcast の 1 ペリフェラルとして位置付けられ、映像を取り込むためのデバイスです。

Dreamcast 本体と接続することで、動画カメラとして機能します。また、単体でデジタルカメラとしての撮影機能があります。

- ドリームアイ本体

「ドリームアイ」は、ホスト上から 1 つのデバイスと 5 つの拡張デバイスとして同時に認識されます。これにより、画像データを 1int 内に最大 5 回までホストに送信することが可能です。

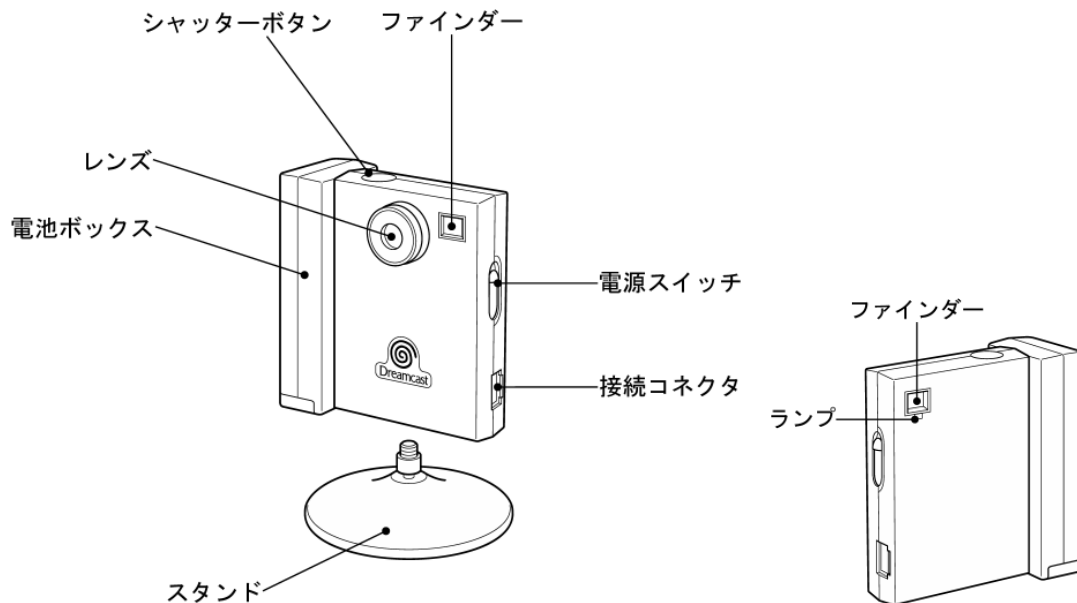
拡張デバイスの使用数は、アプリケーションに依存します。

- CONTROLLER デバイス (ポート 0)

メンバ	情 報
type	CONTROLLER
is_root	1
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	
product_name	"Dreamcast Camera Flash Device"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	200mA
max_pow	350mA

- 仮想デバイス (ポート 1 ~ 5)

メンバ	情 報
type	
is_root	0
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	
product_name	"Dreamcast Camera Flash LDevice"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	0mA
max_pow	0mA

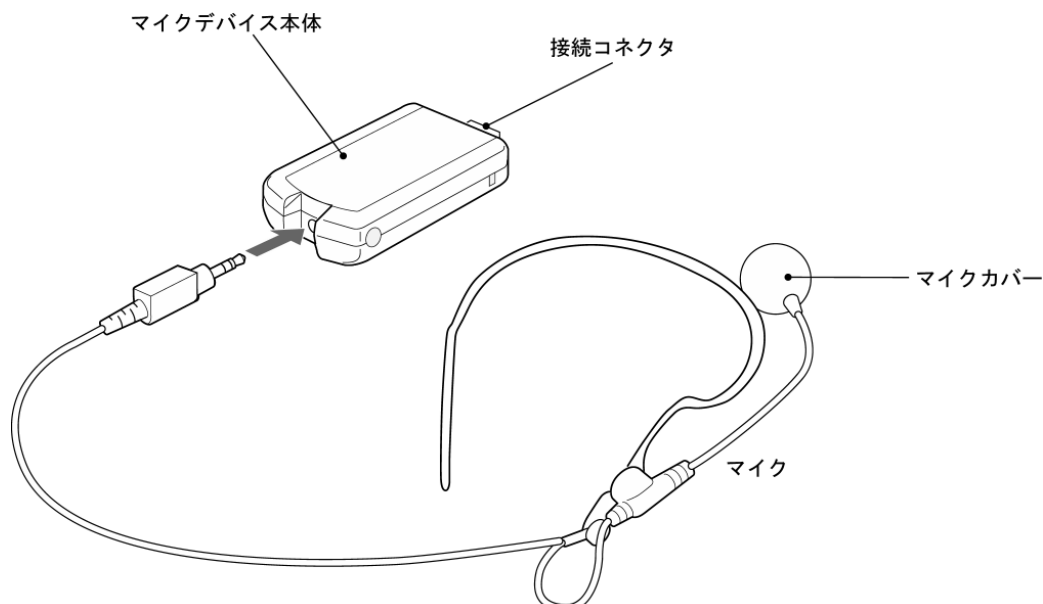


12.1.15 ドリームアイ用マイクデバイス (HKT-9800)

「ドリームアイ」に付属する、ヘッドセット型の音声サンプリングペリフェラルです。マイクデバイス本体に、ヘッドセット型のマイクデバイスを取り付けて使用します。セガライブラリ環境では、このペリフェラルを使うための専用の関数を用意しています。

- ペリフェラル情報

メンバ	情 報
type	SOUNDINPUT
is_root	0
area_code	PDD_DEVAREA_USA / -_JAPAN / -_ASIA / -_EUROPE
connector_dir	
product_name	"MicDevice for Dreameye"
license	"Produced By or Under License From SEGA ENTERPRISES,LTD."
stdby_pow	30.0mA
max_pow	30.0mA



12.2 ビデオケーブル判別 / 画面出力モード一覧表

本体仕向地	放送形式	接続ケーブル	syVideoGetVideoMode関数	SyCbICheckCable関数	syCbICheckBroadcast関数	フレームレート	備考
日本 (アジア)	NTSC	ステレオ AV/S端子	SYD_VIDEO_MODE_NTSC	SYD_VIDEO_CABLE_TYPE_VBS	SYD_VIDEO_BROAD_TYPE_NTSC	60	
		VGA	SYD_VIDEO_MODE_PAL	SYD_VIDEO_CABLE_TYPE_VGA		60	
		(RGB)	SYD_VIDEO_MODE_NTSC	SYD_VIDEO_CABLE_TYPE_RGB		60	
アメリカ (北米)	NTSC	ステレオ AV/S端子	SYD_VIDEO_MODE_NTSC	SYD_VIDEO_CABLE_TYPE_VBS	SYD_VIDEO_BROAD_TYPE_NTSC	60	
		VGA	SYD_VIDEO_MODE_VGA	SYD_VIDEO_CABLE_TYPE_VGA		60	
		(RGB)	SYD_VIDEO_MODE_NTSC	SYD_VIDEO_CABLE_TYPE_RGB		60	
ヨーロッパ	PAL	ステレオ AV/S端子	SYD_VIDEO_MODE_PAL	SYD_VIDEO_CABLE_TYPE_VBS	SYD_VIDEO_BROAD_TYPE_PAL	50	
						60	(*1)
		VGA	SYD_VIDEO_MODE_VGA	SYD_VIDEO_CABLE_TYPE_VGA		60	
		RGB	SYD_VIDEO_MODE_PAL	SYD_VIDEO_CABLE_TYPE_RGB		50	
(ブラジル)	PAL-M	ステレオ AV/S端子	SYD_VIDEO_MODE_NTSC	SYD_VIDEO_CABLE_TYPE_VBS	SYD_VIDEO_BROAD_TYPE_PALM	60	(*2)
		VGA	SYD_VIDEO_MODE_VGA	SYD_VIDEO_CABLE_TYPE_VGA		60	
		RGB	SYD_VIDEO_MODE_NTSC	SYD_VIDEO_CABLE_TYPE_RGB		60	
(アルゼンチン)	PAL-N	ステレオ AV/S端子	SYD_VIDEO_MODE_PAL	SYD_VIDEO_CABLE_TYPE_VBS	SYD_VIDEO_BROAD_TYPE_PALN	50	
		VGA	SYD_VIDEO_MODE_VGA	SYD_VIDEO_CABLE_TYPE_VGA		60	
		RGB	SYD_VIDEO_MODE_PAL	SYD_VIDEO_CABLE_TYPE_RGB		50	

網掛け部分は、00/08/01 現在未発売のハードウェアです。

*1 : PAL60 モード (一部の TV は対応しない)

*2 : PAL-M のフレームレートは 60

注意 Ninja2 ライブラリの初期化関数 njInitSystem()のパラメータに、NJD_RESOLUTION_****_AUTO_****が追加されています。
PAL60 モードを除き、この AUTO を含むパラメータを指定することで、本体仕向地、放送形式及び接続ケーブルにあった設定を行えます。

Shinobi2 ライブラリ編

日本語	英 語	内 容
AIP		Application Initial Program の略。イニシャルプログラム(IP)に含まれるプログラムで、アプリケーションプログラムが起動する直前に必要な共通処理を行う。具体的には CPU のレジスタの初期設定やプレイ履歴を記録（指定した場合のみ）する
DMA		Direct Memory Access の略。CPU を介さずに周辺ハードウェア・メインメモリ間でデータの転送を行うこと。shinobi2.lib 関連では、GD-ROM やペリフェラル関連のライブラリで使用している。CPU との同期を取るために、通常、DMA 終了の割り込みもしくはコールバック関数が提供される
INT		Vsync 割り込みから次の Vsync 割り込みまでの期間。1INT は 1/60 秒（PAL の場合は 1/50 秒）
IP	IP	Initial Program の略。アプリケーション実行前に必要な処理を行うプログラム。BOOT ROM がアプリケーション実行前に GD から読みこみ実行する。通常 IPmaker というツールで作成する
IP 領域	IP area	メインメモリのうちの H'8c000000 から H'8c00ffff までの 64K バイトのこと。この領域は IP がロードされる他、システム予約領域のため原則としてアプリケーションは使用してはならない
Vsync 割り込み	Vsync interrupt	垂直同期割り込み。Vblank-In 割り込みと同意。1/60 秒毎に発生（PAL は 1/50 秒）。アプリケーションまたはライブラリとも本割り込みと同期して処理を行うものが多い
アライメント	alignment	メモリ配置・境界を表す。例えば、32 バイトアライメントされた領域といった場合、その領域は 32 の倍数のアドレス（16 進表記で下 2 桁が 00,20,40,60,80,a0,c0,e0）で始まる。
グローバルオブジェクト	global object	C++言語で、関数外で定義された変数（記憶領域）。特にクラス（構造体）の型の変数を呼ぶ
グローバルコンストラクタ	global constructor	グローバルオブジェクトのクラスのコンストラクタをこのように呼ぶ。セガライブラリでは、グローバルコンストラクタはライブラリ関数を使用して明示的に実行する必要がある
コールバック(関数)	callback (function)	ライブラリ等がユーザの登録した関数を呼び出す機構。これにより処理の完了やエラーの発生等、イベントが発生した際にユーザの処理が可能となる
サーバ関数	server function	ライブラリ等で、定期的に一定のサービスを行う関数。Vsync 割り込み発生時に実行されるなど、ライブラリ関数呼出と同期せずに動作する
システム ID	system ID	IP に含まれる各アプリケーションまたは GD-ROM 毎の情報。アプリケーション名、その GD-ROM の総枚数、そのうちの何枚目か等の情報を持つ。ディスク交換の必要なゲームでは、システム ID の内容を比較する事で正しく交換されたかを確認できる
スタートアップモジュール	start-up module	ユーザプログラムと一緒にリンクされるモジュール（オブジェクト）で、main()関数が実行される前の初期化を行う。未初期化変数エリアの 0 クリアは、スタートアップモジュールで行う
セクション	section	プログラムをコード、データ、定数等の属性毎にわけたもの。ライブラリプログラムとユーザプログラムは、異なるセクションになる。プログラムをリンクすると同じセクションはまとめられる
ヒープ領域	heap area	malloc()関数等で、プログラムから動的にメモリを割り付けるため

日本語	英 語	内 容
		のメモリ領域。セガライブラリでは、プログラムの最後（BGAP セクションの次）からメインメモリの最後までが、ヒープ領域として割り当てられる。アプリケーション側でヒープ領域の変更は可能
ハンドル	handle	一般には、オブジェクトを識別するための ID（識別子）のこと。例えば、gdfs ライブラリが使用しているファイルハンドルは、オープンしているファイルと 1 対 1 に対応する識別子になる
レイテンシ	latency	遅延。特定の操作を行ってからそれが反映されるまでの時間。パッド操作を行った結果が画面に反映されるまでの“レイテンシ”。ネットワークのレイテンシといった場合、送信側がデータを送信してから受信側が受取るまでに掛かる時間
ディスクドアオープン	disk door open	アプリケーション実行中に、ディスクのドアを開けること。アプリケーションの作成基準として、一部の例外を除いてディスクドアオープン時には、ブートルームのメニューに戻らなければならない
ソフトウェアリセット	software reset	アプリケーションプログラムでサポートするリセット機能。通常、コントローラで一定の操作をすると（1）タイトルループからは、本体ブートルームのメニューへ（2）アプリケーションがスタートした後は、タイトル画面へ移る
ELF ファイル	ELF file	ELF は Executable and Linkable Format で、オブジェクトファイルや実行可能ファイル標準的形式。ELF ファイルといった場合、オブジェクトファイルをリンクした後の実行可能ファイル（通常拡張子を.elf）
サブコマンドファイル	sub command file	コンパイラ、リンカ等のコマンドラインツールを実行するときに指定するオプションを記述したファイル。コマンド実行時、多数のオプションを指定する代わりに、サブコマンドファイル名を指定するだけで同等の効果となる
セキュリティ	security	正式な Dreamcast 用 GD-ROM 以外では、一切のアプリケーションが起動しないために保護すること（もしくはそのための機能）
ファーストリードファイル	1st Read file	アプリケーション起動時に、最初に（厳密には ip.bin の次に）ブートルームに読み込まれ実行されるファイル