

Smalight® OS V3
リファレンスマニュアル
RL78版

マクセルシステムテック株式会社

ご注意

安全設計に関するお願い

- 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

- 本資料は、お客様が用途に応じた適切な弊社製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について弊社が所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、弊社は責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、弊社は、予告なしに、本資料に記載した製品または仕様を変更することがあります。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、弊社はその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。弊社は、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、弊社へご照会ください。
- 本資料の転載、複製については、文書による弊社の事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら弊社までご照会ください。

Smalight、および、Smalight のロゴは、マクセルシステムテック株式会社の登録商標です。

μTRON は、Micro Industrial TRON の略称です。

TRON は、The Realtime Operating system Nucleus の略称です。

その他、本書で登場するシステム名、製品名は各社の登録商標または商標です。

はじめに

このマニュアルは、従来、OS(Operating System)の導入が困難とされていた ROM・RAM 容量が少ないシングルチップ向けに開発されたリアルタイム・マルチタスク OS: Smalight OS (スマライトオーエス、SMArt & LIGHT Operating System) の使用方法について説明します。

Smalight OS をご使用になる前に本マニュアルをよく読んで理解してください。また下記関連マニュアルもお読みの上、理解してください。

本製品は、ルネサス エレクトロニクス(株)製 RL78 コアのデバイスに対応するものです。

【関連マニュアル】

- ・ 使用するデバイスのハードウェアマニュアル
- ・ 使用するコンパイラのマニュアル

目次

1. リアルタイムOS概説	1
1.1 リアルタイムOS	1
1.2 タスク	1
1.3 マルチタスク	1
1.4 コンテキスト	1
1.5 リアルタイムOSのスタック	2
2. Smalight OS概説	3
2.1 概要	3
2.2 特徴	3
2.3 サービスコール	4
2.4 前提条件	6
2.4.1 開発環境	6
2.4.2 構築環境	6
3. 機能	7
3.1 システム状態	7
3.2 タスクの状態	8
3.3 タスクのスケジューリング	9
3.3.1 タスク優先度の設定	10
3.3.2 タスクスケジューリングの例	11
3.4 タスク間同期・通信と排他制御	13
3.4.1 イベントフラグ	13
3.4.2 セマフォ	15
3.4.3 データキュー	16
3.5 時間管理	18
3.5.1 周期タイマハンドラ	18
3.5.2 システム時刻	18
3.5.3 周期ハンドラ	19
3.5.4 時間管理に関する注意事項	20
3.6 割込みハンドラ	21
3.6.1 <i>disp</i> 無割込みハンドラ	21
3.6.2 <i>disp</i> 有割込みハンドラ	22
3.6.3 多重割込み	23
3.7 トレース	25
3.7.1 トレースの取得	25
3.7.2 トレースの参照	25
3.7.3 トレースに関する注意事項	25
4. サービスコール	26
4.1 システム状態とサービスコール	26
4.2 各サービスコールの動作	28
4.2.1 サービスコール動作別の分類	28
4.2.2 タスクスイッチ型サービスコールの基本動作	28
4.2.3 <i>i</i> 付きサービスコールの基本動作	29
4.2.4 関数型サービスコールの基本動作	30
4.3 サービスコールの説明形式	31
4.4 タスク管理	32

4.4.1	<i>slp_tsk</i> タスクの起床待ち	32
4.4.2	<i>tslp_tsk</i> タスクの起床待ち(タイムアウト付き)	33
4.4.3	<i>wup_tsk, iwup_tsk</i> タスクの起床	34
4.4.4	<i>can_wup</i> タスク起床要求のキャンセル	35
4.4.5	<i>rot_rdq, irot_rdq</i> タスクのローテーション	36
4.4.6	<i>sus_tsk, isus_tsk</i> 他タスクのサスPEND	37
4.4.7	<i>rsm_tsk, irsm_tsk</i> サスPENDの解除	38
4.5	イベントフラグ	39
4.5.1	<i>wai_flg</i> イベントフラグ待ち	39
4.5.2	<i>twai_flg</i> イベントフラグ待ち(タイムアウト付き)	40
4.5.3	<i>set_flg, iset_flg</i> イベントフラグの設定	41
4.5.4	<i>clr_flg</i> イベントフラグのクリア	42
4.5.5	<i>evflg_init</i> イベントフラグの初期化	43
4.5.6	<i>EVTFLG_ATTR</i> イベントフラグ属性の設定(マクロ)	44
4.6	セマフォ	45
4.6.1	<i>wai_sem</i> セマフォの獲得	45
4.6.2	<i>twai_sem</i> セマフォの獲得(タイムアウト付き)	46
4.6.3	<i>sig_sem, isig_sem</i> セマフォの返却	47
4.6.4	<i>sem_init</i> セマフォの初期化	48
4.6.5	<i>SEM_ATTR</i> セマフォ属性の設定(マクロ)	49
4.7	データキュー	50
4.7.1	<i>recv_dtq</i> データキューからの受信	50
4.7.2	<i>trcv_dtq</i> データキューからの受信(タイムアウト付き)	51
4.7.3	<i>snd_dtq, isnd_dtq</i> データキューへの送信	52
4.7.4	<i>tsnd_dtq</i> データキューへの送信(タイムアウト付き)	53
4.7.5	<i>fsnd_dtq, ifsnd_dtq</i> データキューへの強制送信	54
4.7.6	<i>dtq_init</i> データキューの初期化	55
4.7.7	<i>DTQ_ATTR</i> データキューの属性設定	56
4.8	時間管理	57
4.8.1	<i>set_tim</i> システム時刻の設定	57
4.8.2	<i>get_tim</i> システム時刻の取得	58
4.8.3	<i>systim_init</i> 時間管理の初期化	59
4.8.4	<i>slos_cyclic_timer</i> 周期タイマ処理	60
4.9	周期ハンドラ	61
4.9.1	<i>sta_cyc</i> 周期ハンドラの開始	61
4.9.2	<i>stp_cyc</i> 周期ハンドラの停止	62
4.9.3	<i>cyc_init</i> 周期ハンドラの初期化	63
4.9.4	<i>CYC_ATTR</i> 周期ハンドラの属性設定	64
4.9.5	<i>CYC_CHG</i> 周期ハンドラの起動周期変更	65
4.9.6	<i>callback_cychdr</i> 周期ハンドラ本体(コールバック)	66
4.10	割込み処理支援	67
4.10.1	<i>callback_int</i> disp有割込みハンドラ本体	67
4.11	その他の初期化	68
4.11.1	<i>slos_init</i> OSの起動	68
4.11.2	<i>kinit</i> OS初期化処理(コールバック)	69
4.11.3	<i>uinit</i> ユーザ初期化処理(コールバック)	70
4.11.4	<i>stack_init</i> タスクstackの初期化(コールバック)	71
4.12	その他	72
4.12.1	<i>idle</i> アイドル処理(コールバック)	72
4.12.2	<i>systemdown</i> システムダウン処理(コールバック)	73
4.13	トレース	74
4.13.1	<i>set_trc</i> ユーザトレースの取得	74
4.13.2	<i>get_trc</i> トレースの参照	75
4.14	スタック操作	76
4.14.1	<i>GET_REG</i> ユーザタスクのstackからのレジスタ参照	76
4.14.2	<i>SET_REG</i> ユーザタスクのstackへのレジスタ設定	77

5. アプリケーションプログラムの作成	78
5.1 作成の手順	78
5.2 システムの起動処理	79
5.2.1 システム起動時の処理フロー	79
5.2.2 <i>Smalight OS</i> 提供のCS+プロジェクト	80
5.2.3 CS+プロジェクトの作成	80
5.2.4 コード生成(設計ツール)	82
5.2.5 CC-RL(ビルド・ツール)	84
5.2.6 初期化処理	84
5.3 タスク	87
5.4 割込みハンドラ	88
5.4.1 <i>disp</i> 無割込みハンドラ	88
5.4.2 <i>disp</i> 有割込みハンドラ	90
5.4.3 ベクターテーブル	92
5.4.4 割込みスタックについて	92
5.5 時間管理	93
5.5.1 時間管理の対象サービスコール	93
5.5.2 時間管理の初期化	93
5.6 イベントフラグ	94
5.6.1 イベントフラグの対象サービスコール	94
5.6.2 イベントフラグの初期化	94
5.7 セマフォ	95
5.7.1 セマフォの対象サービスコール	95
5.7.2 セマフォの初期化	95
5.8 データキュー	96
5.8.1 データキューの対象サービスコール	96
5.8.2 データキューの初期化	96
5.9 周期ハンドラ	97
5.9.1 周期ハンドラの対象サービスコール	97
5.9.2 周期ハンドラの作成	97
5.9.3 周期ハンドラの初期化	98
5.10 周期タイマハンドラの初期化	99
5.11 周期タイマハンドラの作成	99
6. 構築	100
6.1 ファイル・ディレクトリ構成と操作	100
6.2 構築手順	101
6.3 OS定義ファイル	103
6.4 コンフィギュレーションファイル	104
6.4.1 カーネルマスクレベルの設定	104
6.4.2 タスクの設定	105
6.4.3 周期タイマハンドラ周期時間の設定	107
6.4.4 イベントフラグ数の設定	107
6.4.5 セマフォ数の設定	108
6.4.6 周期ハンドラ数の設定	108
6.4.7 データキュー数の設定	109
6.5 CS+	110
6.5.1 共通オプション	110
6.5.2 コンパイル・オプション	110
6.5.3 アセンブル・オプション	110
6.5.4 リンク・オプション	111
6.5.5 リンク・ディレクトリの設定	111
6.5.6 ソースファイルの追加	111
6.5.7 ロードモジュールの生成	111
6.6 スタック使用量の算出	112
6.6.1 タスクスタック	112

6.6.2	割込みスタック	113
6.6.3	OSスタック	114
6.7	メモリ容量の最適化	115
6.7.1	OSライブラリ	115
6.7.2	OSライブラリのリビルド	116
6.8	トレースを有効にする	118
6.8.1	OSライブラリの再構築	118
6.8.2	アプリケーションプロジェクトの設定	118
7.	サンプルプログラム	119
8.	コンフィギュレータ	120
付録A	変更履歴	121
付録B	制限事項	122
付録C	RL78に関する注意事項	123
C.1	PSWレジスタについて	123
C.2	RAMパリティ・エラー検出機能について	124
C.3	SmalightOS ROMセクション配置について	124
付録D	スタック仕様	125
付録E	データ型、リターンコード	126
付録F	トラブルシューティング	127
F.1	リンクエラーが発生する	127
付録G	応用例	128
G.1	GET_REG/SET_REGの活用	128
G.2	擬似的なter_tsk, sta_tsk	129
G.3	stack_initを利用したパラメータ引渡し	130
付録H	トレース情報	131
H.1	テーブル仕様	131
H.2	トレース種別	132

図・表・リスト目次

図3-1 システム状態	7
図3-2 タスク状態遷移図	8
図3-3 タスクのスケジューリング	9
図3-4 タスクスケジュールの例	11
図3-5 割込み禁止状態でのタスク切替え	12
図3-6 イベントフラグの使用例	14
図3-7 イベントフラグを利用したデータ送信	14
図3-8 セマフォの使用例	15
図3-9 データキューの概念	16
図3-10 データキューの使用例	17
図3-11 時間管理の注意事項	18
図3-12 周期ハンドラの起動タイミング	19
図3-13 周期ハンドラの呼び出し	19
図3-14 disp無割込みハンドラの基本動作	21
図3-15 disp有割込みハンドラの基本動作	22
図3-16 多重割込み動作例	23
図3-17 多重割込み実装上の注意事項①	24
図3-18 多重割込み実装上の注意事項②	24
図4-1 タスクスイッチ型サービスコールの基本動作	28
図4-2 付きサービスコールの基本動作	29
図4-3 データキューへの強制送信時、空きがない場合の動作(データ数=3)	54
図5-1 アプリケーション作成フロー	78
図5-2 システム起動時のフロー	79
図5-3 CS+プロジェクト作成後のディレクトリ構成	81
図5-4 Smalight OSのコピー	81
図5-5 disp無割込みハンドラのフロー	88
図5-6 disp有割込みハンドラのフロー	90
図6-1 構築手順①	101
図6-2 構築手順②	102
図6-3 関数のスタック計算方法	112
図6-4 タスクのスタック計算方法	112
図6-5 disp有割込みハンドラのスタック切り替えタイミング	113
図8-1 Smalight Configurator	120
図8-2 Smalight Configuratorの入出力	120
図D-1 RL78のスタック仕様	125
図H-1 トレースのテーブル仕様	131
 表2-1 サービスコール一覧①	4
表2-2 サービスコール一覧②	5
表2-3 開発環境	6
表2-4 CS+プロジェクト、ビルド・モード一覧	6
表3-1 タスク優先度の設定例①	10
表3-2 タスク優先度の設定例②	10
表3-3 タスク優先度の設定例③	10
表3-4 割込みハンドラの定義	21
表4-1 システム状態とサービスコール①	26
表4-2 システム状態とサービスコール②	27
表4-3 タスクスイッチ型サービスコール	28

表4-4 i付きサービスコール.....	29
表4-5 関数型サービスコール	30
表5-1 CS+プロジェクト初期設定	80
表5-2 CS+プロジェクト作成ダイアログ設定	80
表5-3 Smalight OSの割込みレベル.....	84
表5-4 時間管理関連サービスコール一覧.....	93
表5-5 イベントフラグ関連サービスコール一覧	94
表5-6 セマフォ関連サービスコール一覧	95
表5-7 データキュー関連サービスコール一覧	96
表5-8 周期ハンドラ関連サービスコール一覧	97
表6-1 ファイル・ディレクトリ構成	100
表6-2 カーネルマスクレベル設定①.....	104
表6-3 共通オプション ビルド・モード設定	110
表6-4 コンパイル・オプション 追加のインクルード・パス設定	110
表6-5 コンパイル・オプション 定義マクロ設定.....	110
表6-6 アセンブル・オプション追加のインクルード・パス設定	110
表6-7 リンク・オプション ライブラリ設定	111
表6-8 SmalightOSに関するセクション名	111
表6-9 デフォルトで使用可能なサービスコール	115
表7-1 サンプルプログラム一覧	119
表A-1 マニュアル訂正(第1版⇒第2版)	121
表A-2 マニュアル訂正(第2版⇒第3版)	121
表A-3 マニュアル訂正(第3版⇒第4版)	121
表E-1 データ型一覧	126
表E-2 リターンコード一覧	126
表H-1 サービスコールトレースの格納情報	132
表H-2 サービスコールIDとトレース情報①.....	132
表H-3 サービスコールIDとトレース情報②.....	133
表H-4 ディスパッチトレースの格納情報	134
表H-5 アイドルトレースの格納情報	134
表H-6 ユーザトレースの格納情報	134
 リスト5-1 hdwinit関数(sample¥R5F100LE¥main¥r_systeminit.c)	82
リスト5-2 main関数(sample¥R5F100LE¥main¥r_main.c).....	83
リスト5-3 kinitコールバック(kinit.c).....	85
リスト5-4 uinitコールバック(user.c)	86
リスト5-5 タスク記述方法	87
リスト5-6 disp無割込みハンドラ(sample¥R5F100LE¥non-disp¥r_cg_timer_user.c)	89
リスト5-7 disp有割込みハンドラ(sample¥R5F100LE¥disp¥r_cg_timer_user.c)	91
リスト5-8 disp有割込みスタック(stack.asm)	92
リスト5-9 時間管理の有効設定 (slos.def)	93
リスト5-10 時間管理の初期化(kinit.c)	93
リスト5-11 イベントフラグの有効設定 (slos.def).....	94
リスト5-12 イベントフラグの初期化(kinit.c)	94
リスト5-13 セマフォの有効設定 (slos.def).....	95
リスト5-14 セマフォの初期化(kinit.c).....	95
リスト5-15 データキューの有効設定 (slos.def).....	96
リスト5-16 データキューの初期化(kinit.c).....	96
リスト5-17 周期ハンドラ記述方法.....	97
リスト5-18 周期ハンドラの有効設定 (slos.def).....	98
リスト5-19 周期ハンドラの初期化(kinit.c).....	98
リスト5-20 周期タイマハンドラの例.....	99
リスト6-1 OS定義ファイルの設定(slos.def).....	103
リスト6-2 カーネルマスクレベルの設定(config.c)	104
リスト6-3 タスクの設定①(config.c).....	105
リスト6-4 タスクの設定②(config.c).....	106

リスト6-5 周期タイマハンドラ周期時間の設定例(config.c)	107
リスト6-6 イベントフラグ数の設定(config.c)	107
リスト6-7 セマフォ数の設定(config.c)	108
リスト6-8 周期ハンドラ数の設定(config.c)	108
リスト6-9 データキュー数の設定(config.c)	109
リスト6-10 構築情報(slos.h)	116
リスト6-11 構築情報の変更(slos.h)	117
リスト6-12 トレース領域のサイズ設定(knl_trca.h)	118
リストG-1 GET_REG/SET_REGの活用例	128
リストG-2 擬似的なter_tsk, sta_tskの実装例	129
リストG-3 stack_initを利用したパラメータ引渡し	130

1. リアルタイム OS 概説

1.1 リアルタイム OS

OS の中で特に時間の制約が厳しい、リアルタイム性が求められるといった要求に特化した OS をリアルタイム OS といいます。

例えば、車に搭載されるカーナビゲーションシステムでは、目的地を通り過ぎてから目的地のガイドされたり、現在の位置を示すマップ表示が遅かったりしたのでは、カーナビゲーションシステムとしての機能が満たされません。

この様にリアルタイム性が求められるシステムで使用される OS がリアルタイム OS です。主に組み込みシステムでリアルタイム OS が採用されています。リアルタイム OS の代表格には、μTRON 仕様準拠 OS があります。

1.2 タスク

OS から見た処理の単位をタスクといいます。逆にユーザから見た処理の単位はジョブといいます。OS 上でジョブを実現するための手段として、タスクという処理単位で実装され、1つのジョブはひとつ、または複数のタスクからなりたっています。

1.3 マルチタスク

OS が複数のタスクを切り替えながら実行することをマルチタスクと言います（逆に同時に1つのタスクしか実行しない方式をシングルタスクと言います）。

タスク切り替えのことをディスパッチといいます。また、タスクスケジュールを行う OS の機能をディスパッチャといいます。ディスパッチャで実行タスクを決定する処理をタスクスケジューリングといいます。タスクを切り替えることにより、入出力待ちなどでタスクが待ち状態となっても、他のタスクが動作できるため、システム全体のスループットが向上します。

短い時間でタスクの切り替えを行うことで、複数のアプリケーション（タスク）が平行して動作している様に見えます。

1.4 コンテキスト

コンテキストとは CPU の状態（プログラムカウンタ、スタックポインタ、レジスタ など）を指します。

各々のタスクは個別にコンテキストをもっており、実行中の状態から、他のタスクへ切り替わるタイミングで、コンテキストを保存します。再度、そのタスクが実行されるとき、保存されたコンテキストから、中断前の状態に戻すことで矛盾なく、実行を再開することができます。

コンテキストの保存・復帰が行われるタイミングとしては、OS のサービスコール発行時、割込み発生時などがあります。

一般的にコンテキストはスタック領域、OS の管理領域に保存されます。

1.5 リアルタイム OS のスタック

リアルタイム OS のスタックは、タスクスタック、割込みスタック、OSスタックがあります。

(1)タスクスタック

タスクスタックは、タスクを実行するための必要な情報で、関数内のローカル変数、コンテキストなどが格納されます。タスクの切り替えが発生しても継続して実行するためには、タスク毎にスタックを準備する必要があります。

(2)割込みスタック

割込みスタックは、割込み関数内のローカル変数、割込み発生元への戻り情報などが格納されます。割込み発生元のスタックをそのまま使用する実装方法もありますが、多重割込みを考慮した場合、割込み発生元のスタック使用量の見積が困難となる、また、スタックサイズも大きくなることから、リアルタイム OS では割込みレベル毎に割込みスタックを準備し、スタックを切り替えて使用します。

(3)OS スタック

OS スタックは OS 実行中に使用されるスタックです。

リアルタイム OS が主に採用される組み込みシステムの特性上、使用できるメモリ容量に制限があります。限られるメモリを有効に使用するため、スタック使用量を計算して設定する必要があります。

スタックサイズを必要以上に指定した場合、使われない無駄なメモリが発生します。また、スタックサイズが少ない場合、指定されたスタックを超えてメモリの内容が不正に書き換えられ動作結果が不定となりますので、適切なスタックサイズを指定することが重要となります。

2. Smalight OS 概説

2.1 概要

近年、リアルタイム OS の必要性が高くなっています。しかし、ROM/RAM 容量が小さいシングルチップ・マイコンを使った組込みシステムの場合、従来のリアルタイム OS では ROM/RAM 容量が大きく、導入を断念するケースが多いようです。本リアルタイム OS は小容量コンパクトなリアルタイム・マルチタスク OS です。

2.2 特徴

(1) ITRON 仕様ライクな API

ITRON 仕様に完全に準拠したものではありませんが、ITRON 仕様ライクな API 提供でサービスコールのイメージがつかみやすく、また、将来 ITRON 仕様準拠 OS への置き換え時、移植が容易です。

(2) ビルディングブロック方式による ROM/RAM 最適化

必要最低限の機能だけに限定したシンプルな OS で、ROM/RAM 容量の小さいシングルチップに最適です。アプリケーションで必要とする機能だけを選択して構築できます。リソースの限られたシングルチップマイコンでも ROM/RAM サイズの最適化が図れます。

最新のサポート MCU と ROM/RAM 容量などについては、弊社ホームページを参照ください。

(<http://www.systemtech.maxell.co.jp/solution/smalight-index/>)。

(3) タスクスケジュール方式: 優先度に基づいたスケジューリング

優先度ベースのスケジューリングに加えて、最下位優先度タスクについては FCFS(First Come First Service)によるスケジューリングをサポートしています。

(4) 各種オブジェクトの最大数(タスク, イベントフラグ, セマフォ, データキュー, 周期ハンドラ): 127 個

(5) サンプルのベクターテーブル、ユーザタスク、割込みハンドラを提供しています。

(6) コンフィギュレーションファイルと GUI ツールによる簡単な構築ができます。

2.3 サービスコール

以下にサービスコール一覧を示します。

表2-1 サービスコール一覧①

区分	サービスコール名称	説明
タスク管理関連	slp_tsk	タスクの起床待ち
	tslp_tsk	タスクの起床待ち(タイムアウト付き)
	wup_tsk, iwup_tsk	タスクの起床
	can_wup	タスク起床要求のキャンセル
	rot_rdq, irot_rdq	タスクのローテーション
	sus_tsk, isus_tsk	他タスクのサスPEND
	rsm_tsk, irsm_tsk	サスPENDの解除
イベントフラグ	wai_flg	イベントフラグ待ち
	twai_flg	イベントフラグ待ち(タイムアウト付き)
	set_flg, iset_flg	イベントフラグの設定
	clr_flg	イベントフラグのクリア
	evtflg_init	イベントフラグの初期化
	EVTFLG_ATTR ^{(*)1}	イベントフラグの属性設定
セマフォ	wai_sem	セマフォの獲得
	twai_sem	セマフォの獲得(タイムアウト付き)
	sig_sem, isig_sem	セマフォの返却
	sem_init	セマフォの初期化
	SEM_ATTR ^{(*)2}	セマフォの属性設定
データキュー	rcv_dtq	データキューからの受信
	trcv_dtq	データキューからの受信(タイムアウト付き)
	snd_dtq, isnd_dtq	データキューへの送信
	tsnd_dtq	データキューへの送信(タイムアウト付き)
	fsnd_dtq, ifsnd_dtq	データキューへの強制送信
	dtq_init	データキューの初期化
	DTQ_ATTR ^{(*)1}	データキューの属性設定
時間管理	set_tim	システム時刻の設定
	get_tim	システム時刻の取得
	systim_init	時間管理の初期化
	slos_cyclic_timer	周期タイマ処理
周期ハンドラ	sta_cyc	周期ハンドラの開始
	stp_cyc	周期ハンドラの停止
	cyc_init	周期ハンドラの初期化
	CYC_ATTR ^{(*)1}	周期ハンドラの属性設定
	CYC_CHG ^{(*)1}	周期ハンドラの起動周期変更
	callback_cychdr ^{(*)2}	周期ハンドラ本体

^{(*)1} マクロです。

^{(*)2} コールバックルーチンです。

^{(*)3} 本機能を使用するためにはOSを再構築する必要があります。

表2-2 サービスコール一覧②

区分	サービスコール名称	説明
割込み関連	callback_int ^(*2)	disp有割込みハンドラ本体
その他の初期化	slos_init	OSの起動
	kinit ^(*2)	OS初期化処理
	uinits ^(*2)	ユーザ初期化処理
	stack_init ^(*2)	タスクス택の初期化
その他	idle ^(*2)	アイドル処理
	systemdown ^(*2)	システムダウン処理
トレース	set_trc ^(*3)	ユーザトレースの取得
	get_trc ^(*3)	トレースの参照
スタック操作	GET_REG ^(*1)	ユーザタスクのスタックからのレジスタ参照
	SET_REG ^(*1)	ユーザタスクのスタックのレジスタ設定

^(*1) マクロです。

^(*2) コールレバッカルーチンです。

^(*3) 本機能を使用するためにはOSを再構築する必要があります。

2.4 前提条件

2.4.1 開発環境

本製品の開発環境を以下に示します。他の開発環境への移行は、ユーザの責任において行ってください。

表2-3 開発環境

ツール名
ルネサス エレクトロニクス(株) 製 CS+ for CC V6.00.00
(ビルド・ツール CC-RL V1.05.00) ^(*)1)

^(*)1) CS+、及び、ビルド・ツールは上位互換のみ保証されています

2.4.2 構築環境

本製品では、以下のルネサス エレクトロニクス(株)製 CS+プロジェクトを提供しています。

表2-4 CS+プロジェクト、ビルド・モード一覧

ディレクトリ名	プロジェクト		ビルド・モード	説明
	ファイル	名称		
RL78_v310r	u-ap.mtpj	u-ap	obj_RL78m	RL78用(ミディアム・モデル)
			obj_RL78s	RL78用(スマート・モデル)

3. 機能

3.1 システム状態

システム状態は、システム初期化部、OS 部、タスク部、割込み部で管理します。

システム初期化部はシステムの初期化を行う処理で、kinit、uinit、stack_init コールバックルーチンがシステム初期化部に属します。OS 部は OS 実行中の状態です。タスク部はタスク実行中の状態です。割込み部は割込み実行中の状態です(周期ハンドラの実行中は割込み部です)。

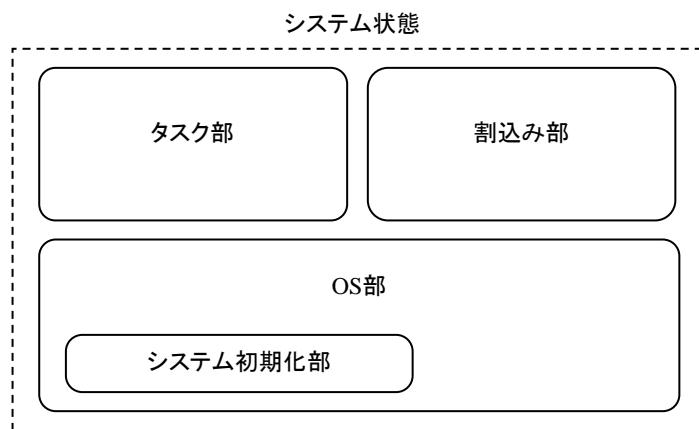


図3-1 システム状態

3.2 タスクの状態

タスク状態は Running, Ready, Waiting, Suspended の 4 種類で管理します。以下に状態遷移図を示します。

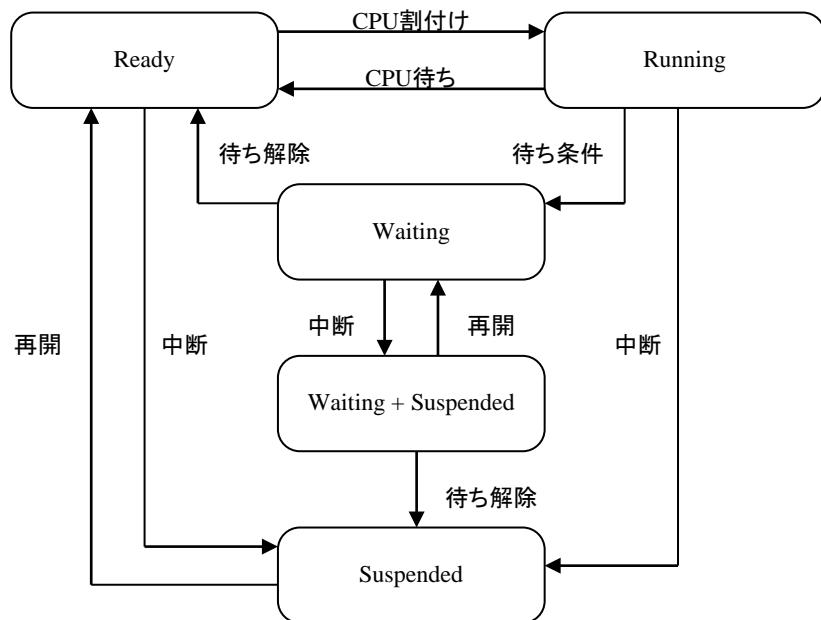


図3-2 タスク状態遷移図

3.3 タスクのスケジューリング

本 OS は優先度ベースのスケジューリングと FCFS(First Come First Service)によるスケジューリングの2つの方式をサポートしています。この2つの方式は同時に使用することができます。

優先度ベースのスケジューリングの対象となるタスクをプライオリティタスク(Priority Task)といいます。FCFS(First Come First Service)によるスケジューリングに対象となるタスクをローテーションタスク(Rotation Task)といいます。ローテーションタスクは、最低優先度のタスクとして管理されます。

プライオリティタスクの優先度は、タスク ID 番号と同じ値で、値が小さい程、優先度が高くなります(つまり、プライオリティタスクは同一優先度レベルのタスクが存在しません)。プライオリティタスクが Ready 状態になるとタスク優先度に従って Running 状態になります。一番優先度の高いプライオリティタスクが一度 Running 状態になると Waiting 状態、もしくは Suspended 状態になるまで、タスクの切替えは発生しません。

ローテーションタスクは、優先度が一番低いタスクで、同一優先度に複数のタスクを定義することができます。実行順序は FCFS (First Come First Service)にて管理されます。

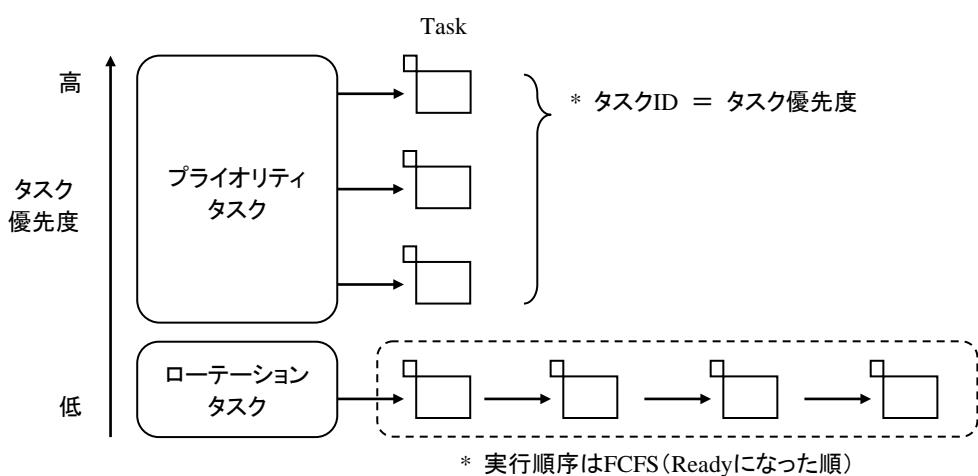


図3-3 タスクのスケジューリング

3.3.1 タスク優先度の設定

プライオリティタスク、ローテーションタスクは構築により静的に決定します。構築により、どの様にタスク優先度が設定されるかを具体的に説明します。構築情報では、タスク数とプライオリティタスク数を指定します。これにより、各タスクの優先度が決定します。

(1) タスク数=4, プライオリティタスク数=4

表3-1 タスク優先度の設定例①

タスクID	種類	優先度
1	プライオリティタスク	1番優先度が高い
2	プライオリティタスク	2番目に優先度が高い
3	プライオリティタスク	3番目に優先度が高い
4	プライオリティタスク	4番目に優先度が高い

(2) タスク数=4, プライオリティタスク数=2

表3-2 タスク優先度の設定例②

タスクID	種類	優先度
1	プライオリティタスク	1番優先度が高い
2	プライオリティタスク	2番目に優先度が高い
3	ローテーションタスク	3番目に優先度が高い
4	ローテーションタスク	3番目に優先度が高い

(3) タスク数=4, プライオリティタスク数=0

表3-3 タスク優先度の設定例③

タスクID	種類	優先度
1	ローテーションタスク	1番優先度が高い
2	ローテーションタスク	1番優先度が高い
3	ローテーションタスク	1番優先度が高い
4	ローテーションタスク	1番優先度が高い

構築方法の詳細は「6.4.2 タスクの設定」を参照ください。

3.3.2 タスクスケジューリングの例

以下にサービスコールによるタスク制御の例を示します。例では、3 個のタスク(プライオリティタスク=1 個、ローテーションタスク=2 個)で各種サービスコールを発行した場合の状態遷移を示しています。

【解説】

- ① タスク 1 は、プライオリティタスクです。プライオリティタスクは優先度が高いタスクで、待ち要因が解除されるとすぐ Running 状態となります。
- ② タスク 2,3 は、ローテーションタスクです。お互いが実行可能な状態(Running, Ready 状態)のとき、rot_rdq サービスコールを発行することでタスクの実行権が変更されます。
- ③ Waiting 状態のタスクに sus_tsk サービスコールを発行すると Waiting + Suspended 状態となります。両方の待ち要因が解除されることで Ready 状態となります。
- ④ 実行可能な状態(Running, Ready 状態)のタスクがない場合は、アイドル状態となります。

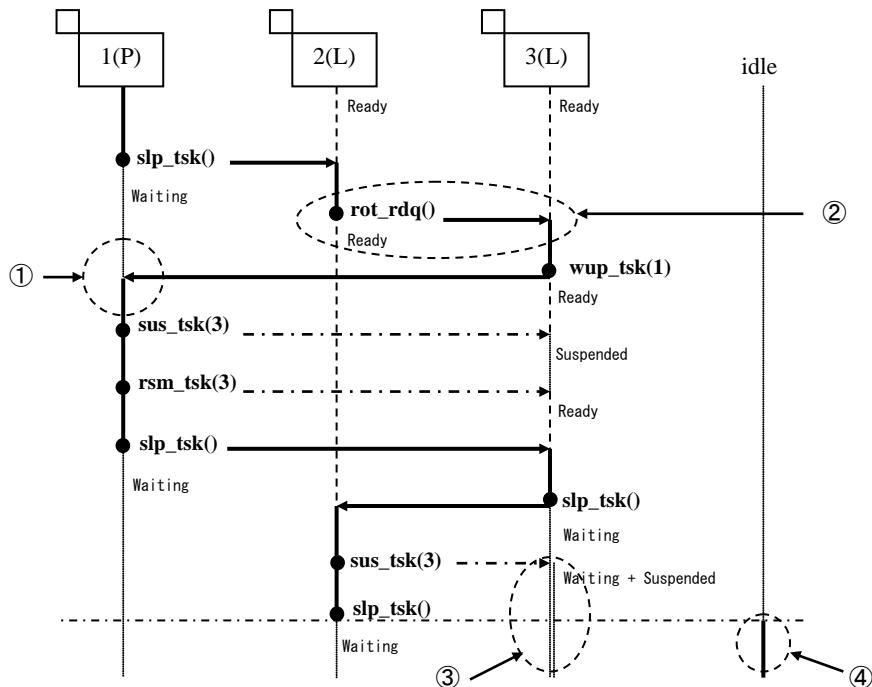


図3-4 タスクスケジュールの例

各サービスコールは基本的にエラーチェックを一切行いません。割込み禁止状態のタスクがタスクスイッチ型サービスコール(P28参照)を発行することでタスクの切替えが発生します。割込みを抑止(保留)したい場合にはタスクスイッチ型サービスコールを発行しないでください。

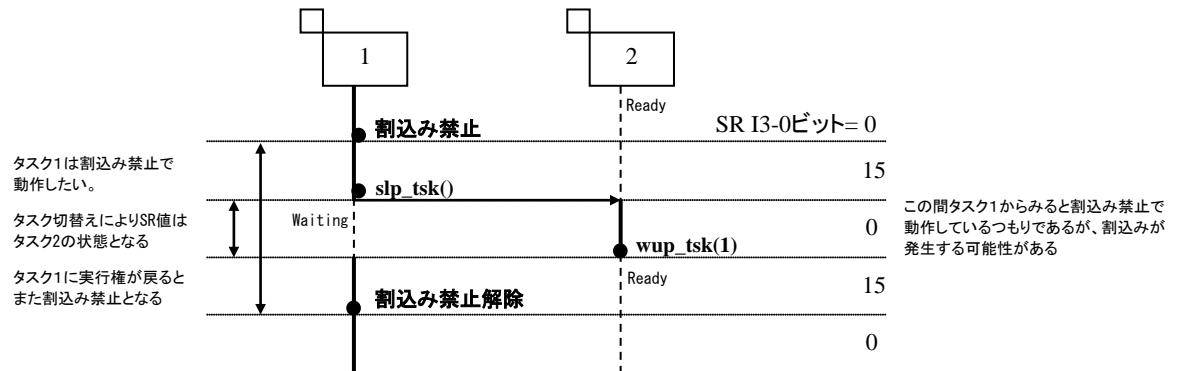


図3-5 割込み禁止状態でのタスク切替え

3.4 タスク間同期・通信と排他制御

3.4.1 イベントフラグ

イベントフラグは、タスク間同期制御のひとつで少量のデータ通信機能としても利用できます。

イベントフラグを使用することにより、複数の事象発生の組み合せによるタスク間同期処理を行うことが出来ます。イベントフラグは、事象に対応したビットの集合体で、1つの事象発生を1ビットで表し、フラグのオン(1)/オフ(0)で行います。Smalight OS のイベントフラグは 16 ビットです。

(1) 基本動作

事象(イベント)発生を待つタスクは事象に対応したビットを指定してイベントフラグで待ちます。事象発生を通知するタスク(または割込みハンドラ)は事象に対応したビットをイベントフラグに設定することで、イベント待ちのタスクの待ちを解除します。

(2) イベントフラグの待ち条件

イベントフラグの待ち条件として、指定したビットのいずれかが発生するまで待つ EVFLG_TA_OR 属性(OR 条件)、指定したビットの全てが発生するまで待つ EVFLG_TA_AND 属性(AND 条件)があります。

イベントフラグ待ちが解除されたときは、wai_flg のリターンパラメータで解除時のビットパターンが戻ります。OR 条件で解除された場合、待ち解除された要因(ビット)で確認できます。

(3) イベントフラグ待ちの待ち行列(待ちキュー)設定

イベントフラグで複数のタスクが同時に Waiting 状態(イベント待ち)となります。そのとき、タスクはイベントフラグの待ち行列に入れます。

待ち行列に入るとき、EVFLG_TA_TFIFO(FIFO 順)属性、または、EVFLG_TA_TPRI(優先度順)属性に従ってキューイングされます。この属性で待ち解除されるタスクの順番が決定されます。

(4) 待ち解除時のイベントフラグクリア属性

EVFLG_TA_CLR 属性(クリア属性)を設定したとき、待ち解除のタイミングで該当するイベントフラグの全てのビットを 0 クリアします。

クリア属性によりイベントフラグを使用した動作に違いがあります。同じイベントフラグで複数のタスクが待っているとき、クリア属性が設定されていなければ、待ち解除の条件を満たす全てのタスクが待ち解除されます。

クリア属性が設定されていれば、待ち行列をサーチして待ち解除の条件を満たすタスクを見つけた時点で、イベントフラグがクリアされます。他に条件が満たすタスクがあっても待ち解除されません。

以下にイベントフラグの使用例を示します。例は、クリア属性なし、タスク優先度はタスク 1 が高い場合を想定しています。

【解説】

- ① タスク 1 は `clr_flg` サービスコールを発行し、該当するフラグをクリア(0)します。
- ② タスク 1 は `wai_flg`(待ちパターン=1) サービスコールを発行し、指定したフラグがセットされてないので、フラグがセットされるのを待ちます。タスク 1 は `Waiting` 状態になります。
- ③ タスク 2 で `set_flg` サービスコールを発行し、フラグを設定します。これにより待ち条件を満たしたタスク 1 は `Ready` 状態となり、`Waiting` 状態から解除されます。クリア属性なしなので、イベントフラグは 1 のままでなります。

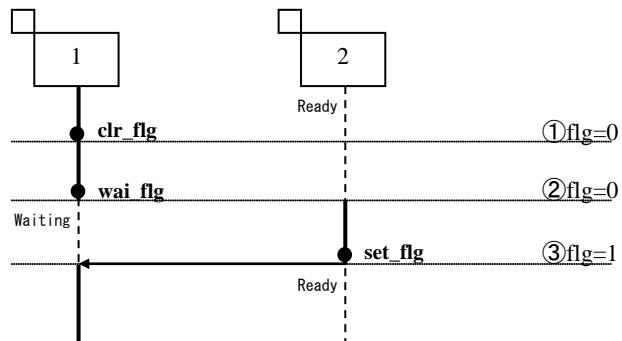


図3-6 イベントフラグの使用例

イベントフラグを利用して、16 ビットのデータ送信が可能です。以下に使用例を示します。例は、クリア属性あり、イベントフラグのチェック条件を OR 条件とします。タスク 1 は、待ちパターン 0xFFFF でイベントフラグ待ちに入ります。タスク 2 で、送信したいデータ(0x1234)をイベントフラグ設定します。

イベントフラグのチェック条件が OR 条件なので、1 ビットでも ON になれば、待ち解除されることを利用したデータ送信となります。

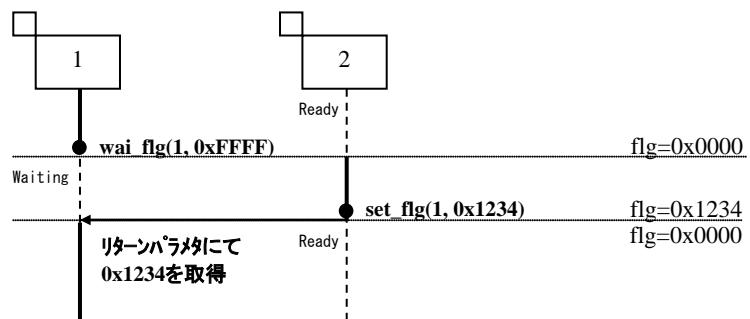


図3-7 イベントフラグを利用したデータ送信

3.4.2 セマフォ

セマフォは、資源(各種 I/O, 共有メモリ、非リエントラント関数など)の排他制御に使用します。

複数のタスクから同時に使用することができない資源を複数のタスクで共有する場合に使用します。使用方法は、資源に対応するセマフォを用意し、資源を使用する前にセマフォを獲得し、資源の使用が終わったらセマフォを返却します。セマフォが獲得できない場合、他タスクで使用中と判断し、セマフォが返却されるのを待ちます。

(1) 基本動作

セマフォを獲得するとセマフォカウント(セマフォ資源数: 資源の並列獲得可能な数)をデクリメント(-1)します。セマフォの獲得ではセマフォカウントが 1 以上のとき、Waiting 状態(セマフォ待ち)にならずにセマフォを獲得できます。セマフォカウントが 0 のとき、Waiting 状態(セマフォ待ち)に入ります。セマフォを返却すると、セマフォカウントをインクリメント(+1)します。

(2) セマフォ資源数の初期値

セマフォ資源数とは、セマフォで排他制御する資源の並行して獲得可能な数です。セマフォ資源数が 2 であれば、2 つのタスクが同時に資源を使用することができます。

(3) セマフォの待ち行列(待ちキュー)設定

セマフォの獲得で複数のタスクが同時に Waiting 状態(セマフォ待ち)となることがあります。そのとき、タスクはセマフォ待ちの待ち行列に入ります。

待ち行列に入るとき、SEM_TA_TFIFO(FIFO 順)属性、または、SEM_TA_TPRI(優先度順)属性に従ってキューイングされます。この属性でセマフォを獲得するタスクの順番が決定されます。

以下にセマフォの使用例を示します。例は、セマフォ資源数の初期値は 1、タスク優先度はタスク 1 が高い場合を想定しています。

【解説】

- ① タスク 2 は wai_sem サービスコールを発行しセマフォを獲得します。資源数(semcnt)は 0 となります。
- ② タスク 1 は wai_sem サービスコールを発行しますが、資源数(semcnt)は 0 のため Waiting 状態になり、セマフォが返却されるのを待ちます。
- ③ タスク 2 で sig_sem サービスコールを発行しセマフォを返却します。これによりタスク 1 はセマフォを獲得し Ready 状態となります。タスク優先度によりタスク 1 が実行(Running 状態)されます。
- ④ タスク 1 は sig_sem サービスコールを発行しセマフォを返却します。資源数(semcnt)は 1 となります。

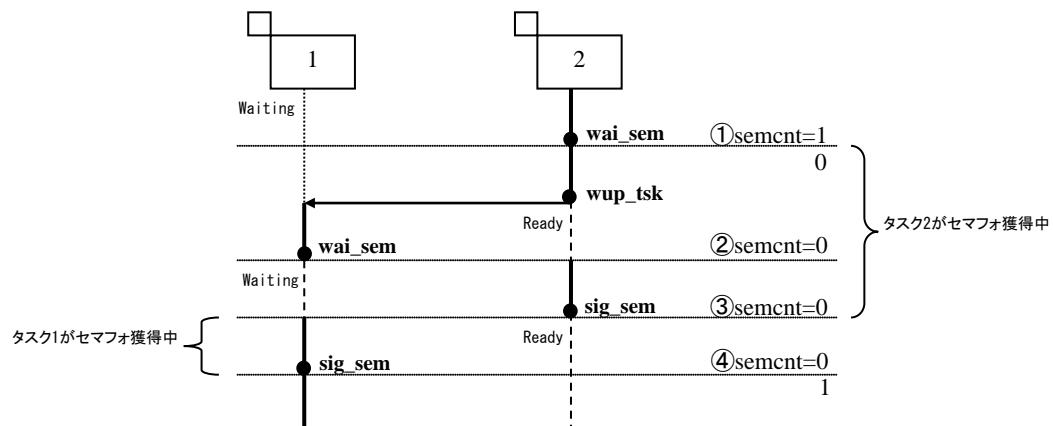


図3-8 セマフォの使用例

3.4.3 データキュー

データキューはタスク間データ通信に使用します。

データキューは1ワードのデータ転送を行います。データ送信するとデータキューにデータが送られます。データ受信はデータキューからデータを受信します。データキューはリングバッファで FIFO(First In First Out)で管理されます。Smalight OS のデータキューは 32 ビット / ワードで実装されます。

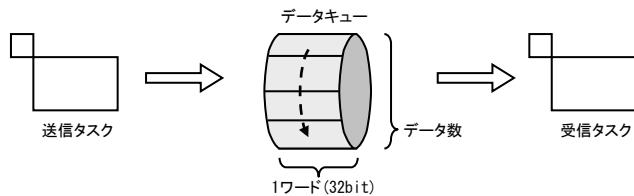


図3-9 データキューの概念

(1) 基本動作

データ送信時、データキューの空きがあればデータキューにデータを設定します。データキューの空きがない場合、タスクはデータキューに空きができるまで Waiting 状態(データキュー待ち)となります。

データ受信時、データキューにデータが存在すればデータキューからデータを取り出します。受信すべきデータがデータキューに存在していない場合は Waiting 状態(データキュー待ち)となります。

(2) データキュー送信待ちの待ち行列(待ちキュー)設定

データキューへの送信で複数のタスクが同時に Waiting 状態(データキュー送信待ち)となることがあります。そのとき、タスクはデータキュー送信待ちの待ち行列(待ちキュー)に入れます。

待ち行列に入るとき、DTQ_TA_TFIFO(FIFO 順)属性、または、DTQ_TA_TPRI(優先度順)属性に従ってキューイングされます。この属性で待ち解除されるタスクの順番が決定されます。

※ 尚、データキュー受信待ちの待ちキューは、常に FIFO 順でキューイングします。

(3) データ数

データキューを構成するリングバッファは、データ数 0~127 個の値を任意に設定できます。データ数が 0 個の場合、送受信すると送信側と受信側、双方が揃うまでタスクは Waiting 状態になります。データ数を多くすることで、データ送信時、Waiting 状態に入る可能性が低くなると言えます。

以下にデータキューの使用例を示します。例は、データ数は 1、タスク優先度は高い順にタスク 1、タスク 2、タスク 3 となります。

【解説】

- ① タスク 1 は `snd_dtq` サービスコールを発行しデータ H'11111111 を送信します。データキューの空きが 1 のため、データ送信が完了します。データキューの空きは 0 となります。
- ② タスク 1 は `slp_tsk` サービスコールを発行し Waiting 状態となります。Ready 状態のタスク 2 が実行(Running 状態)されます。
- ③ タスク 2 は `snd_dtq` サービスコールを発行しデータ H'22222222 を送信します。データキューの空きが 0 のため、データ送信されず Waiting 状態となり、データキューへの送信完了を待ちます。Ready 状態のタスク 3 が実行(Running 状態)されます。
- ④ タスク 3 は `recv_dtq` サービスコールを発行し、データキューからデータ H'11111111 を受信します。データキューに空きが発生したので、データキューへの送信待ちで Waiting 状態のタスク 2 が送信完了し、実行(Running 状態)されます。タスク 3 は Ready 状態です。
- ⑤ タスク 2 は `slp_tsk` サービスコールを発行し Waiting 状態となります。Ready 状態のタスク 3 が実行(Running 状態)されます。
- ⑥ タスク 3 は `recv_dtq` サービスコールを発行し、データキューからデータ H'22222222 を受信します。データキューの空きは 0 となります。

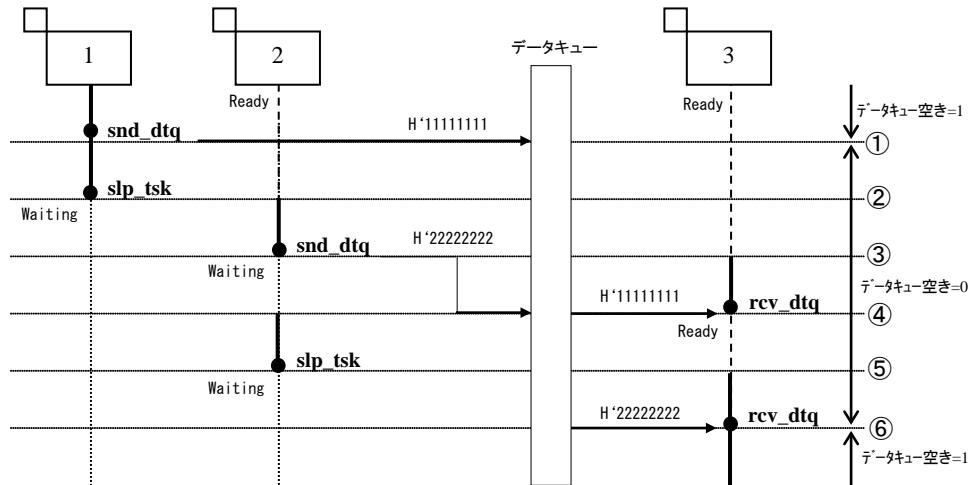


図3-10 データキューの使用例

3.5 時間管理

時間管理として、以下の機能を提供します。

- ・タイムアウト付きサービスコール(tslp_tsk, twai_flg, twai_sem, trcv_dtq, tsnd_dtq)
- ・システム時刻の設定・参照(set_tim, get_tim)
- ・周期ハンドラ (cyc_init, sta_cyc, stp_cyc)

3.5.1 周期タイマハンドラ

時間管理の3つの機能を実現するために、周期的なタイマ割込みハンドラを用意する必要があります。本割込みハンドラを周期タイマハンドラといいます。周期タイマハンドラから、slos_cyclic_timer サービスコールを発行することで時間管理を実現します。

3.5.2 システム時刻

システム時刻とは、符号無し 48bit のカウンタで、周期タイマハンドラによって更新されます。

サービスコール(tslp_tsk 等)のタイムアウト時間パラメータや周期ハンドラの周期時間の単位は 1msec ですが、周期タイマハンドラの周期時間は任意に 1msec 以上の値を設定できます。詳細は「6.4.3 周期タイマハンドラ周期時間の設定」を参照ください。

図 3-11に周期時間を 250msec に設定した場合の tslp_tsk(1) と tslp_tsk(500) で、タイムアウトにより Waiting 状態から Ready 状態に遷移する様子を示します。周期タイマハンドラの周期時間は小さい方が誤差は小さくなりますが、システムトータルで考えると周期タイマハンドラの動作回数が増えることでシステム全体のパフォーマンスが劣化する場合がありますので注意が必要です。

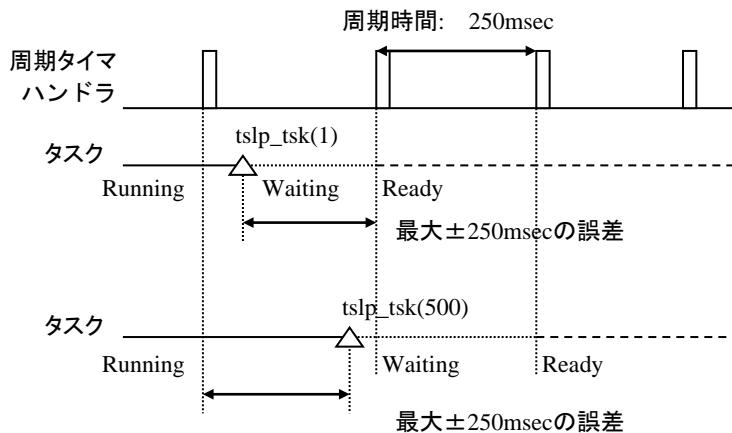


図3-11 時間管理の注意事項

3.5.3 周期ハンドラ

周期的に実行される処理を呼び出す機能を周期ハンドラといいます。周期ハンドラの開始から停止までの間、指定した起動周期で、周期ハンドラが呼び出されます。

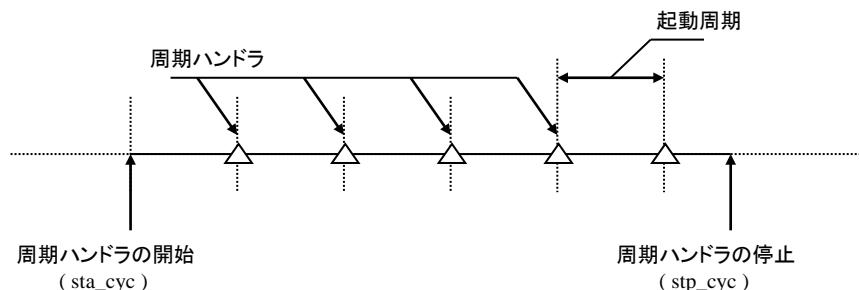


図3-12 周期ハンドラの起動タイミング

周期ハンドラの初期化、および、属性の設定(起動周期、周期ハンドラ関数の指定)は、システム初期化処理で行います。属性の設定で CYC_TA_STA 属性(初期状態で周期ハンドラ動作が開始)を指定すると、周期ハンドラの動作が開始した状態となります。

周期ハンドラは、周期タイマハンドラ(ユーザ任意の周期割込)で呼び出す slos_cyclic_timer サービスコールの延長で呼び出されます。それにより割込みマスクレベルは、カーネルマスクレベルで動作します。周期ハンドラを呼び出す時には、周期タイマハンドラの割込みレベルに設定してコールされます(本文中に記載されるカーネルマスクレベルの詳細は「6.4.1 カーネルマスクレベルの設定」を参照ください)。

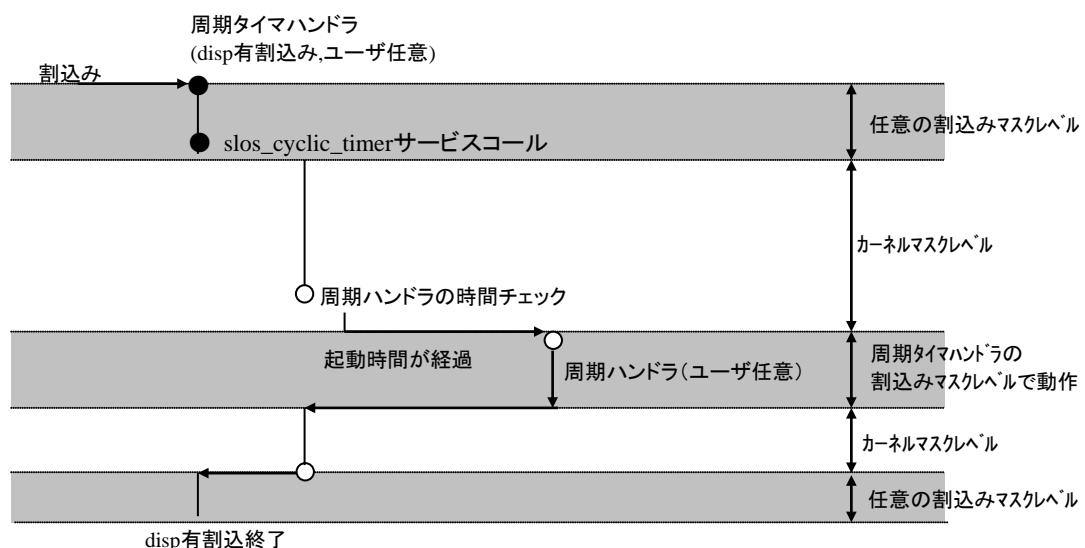


図3-13 周期ハンドラの呼び出し

3.5.4 時間管理に関する注意事項

(1) 時間管理機能のオーバーヘッドについて

周期タイマハンドラ(ユーザ任意)にてコールする `slos_cyclic_timer` サービスコールにて、以下の処理が実行されます。

- ・ システム時刻の更新
- ・ 登録された周期ハンドラの管理、及び、起動周期を経過した全ての周期ハンドラの起動と実行
- ・ t 付きサービスコールにて待ち状態タスクのタイムアウト処理

※ 構築や、使用するサービスコールによって、上記処理が発生しない場合もあります

これらの処理はカーネルマスクレベル、または、使用する周期タイマハンドラの割込みレベルにて実行されるため、`slos_cyclic_timer` サービスコールの実行時間が長くなるとシステム全体のパフォーマンスが劣化、システム時刻の遅延等の問題が起こる可能性があります。例えば、周期タイマハンドラ(ユーザ任意)の周期時間が 1msec とき、周期起動が 1msec の周期ハンドラが起動され、ハンドラ処理時間が 1msec 以上の実行時間を要した場合、永久にハンドラ処理が繰り返し実行され、他のタスク処理が実行されない事態となる可能性もあります。

これらの問題を回避するため、以下の点について注意が必要です。

- ・ 周期ハンドラは可能な限り短く記述してください。
- ・ 登録する周期ハンドラを極力少なくしてください。
- ・ 周期ハンドラの周期時間、 t 付きサービスコールにて指定するタイムアウト時間はなるべく大きな値を指定してください。
- ・ 周期タイマハンドラ(ユーザ任意)の周期は、なるべく大きな値としてください。

3.6 割込みハンドラ

本OSでは割込みハンドラを、以下の2種類に分類しています。この分類により作成方法が異なります。表中に記載されるカーネルマスクレベルの詳細は「6.4.1 カーネルマスクレベルの設定」を参照ください。

表3-4 割込みハンドラの定義

割込み区分	説明
disp無割込みハンドラ (#pragma interrupt)	1. OSサービスコールを発行しない、割込み発生元へ戻る割込み。 2. NMI 3. カーネルマスクレベルを超えるレベルの割込み
disp有割込みハンドラ (#pragma rtos_interrupt)	OSサービスコールを発行してタスク制御やディスパッチを行うための割込み。カーネルマスクレベル以下のレベルの割込み。 割込みの最後でdispサービスコールを発行します。dispサービスコールを発行することにより割込み発生元には戻らず、ディスパッチャへ制御を遷します。

3.6.1 disp無割込みハンドラ

disp無割込みハンドラでは、OSサービスコールを発行してはいけません。disp無割込みハンドラは割込み発生元に戻るため、必要最小限のレジスタ退避のみで処理することができます。以下にdisp無割込みハンドラの基本動作を示します。disp無割込みハンドラでサービスコールを発行した場合の動作は保証されません。

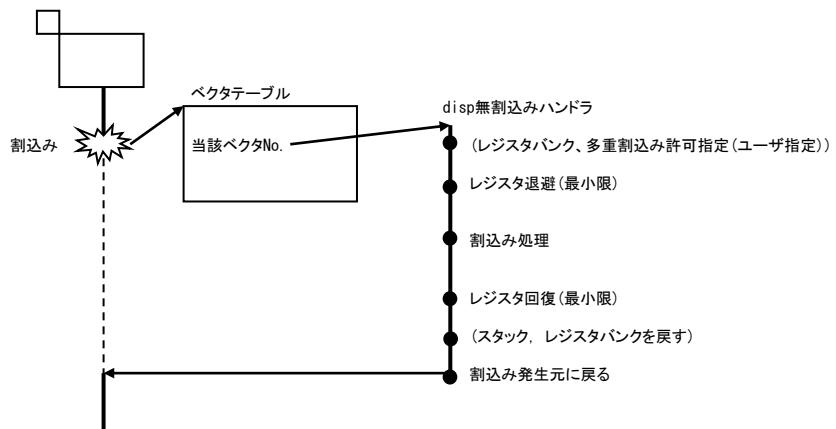


図3-14 disp無割込みハンドラの基本動作

3.6.2 disp 有割込みハンドラ

disp 有割込みハンドラは割込み発生元に戻らず、ディスパッチへ制御を遷すことができます。以下に disp 有割込みハンドラの基本動作を示します。

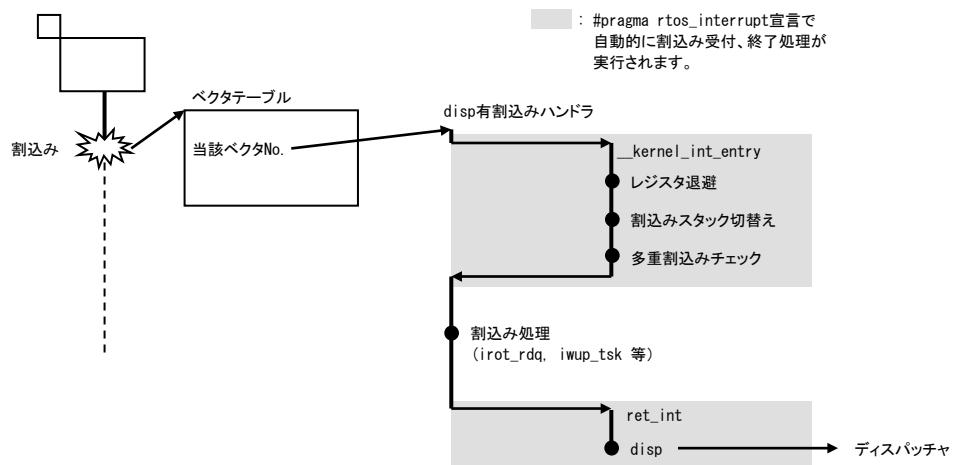


図3-15 disp有割込みハンドラの基本動作

3.6.3 多重割込み

多重割込みの動作例を以下に示します。

【解説】

- ① タスクは基本的にマスクレベルに3で動作させてください。3以外で実行した場合、disp 有割込みが発生しても、ディスパッチされず割込み発行元に戻ります。タスク1,2はローテーションタスク(同一のタスク優先度)です。
 - ② 割込みコントロールレベル2のdisp 有割込み実行中に、割込みコントロールレベル1のdisp 有割込みが発生したケースです。割込みコントロールレベル1の割込みが終了しても多重割込み中は、割込み発行元に戻ります。
 - ③ NMIはノンマスカブル割込みです。ノンマスカブル割込みでOSサービスコールは発行してはいけません。
 - ④ 割込みコントロールレベル2のdisp 有割込みです。割込みが終了すると割込み発行元には戻らず、ディスパッチャへ制御を遷します。
 - ⑤ 割込みコントロールレベル1のdisp 有割込みです。多重割込み中でないので、割込み発行元には戻らず、ディスパッチャへ制御を遷します。

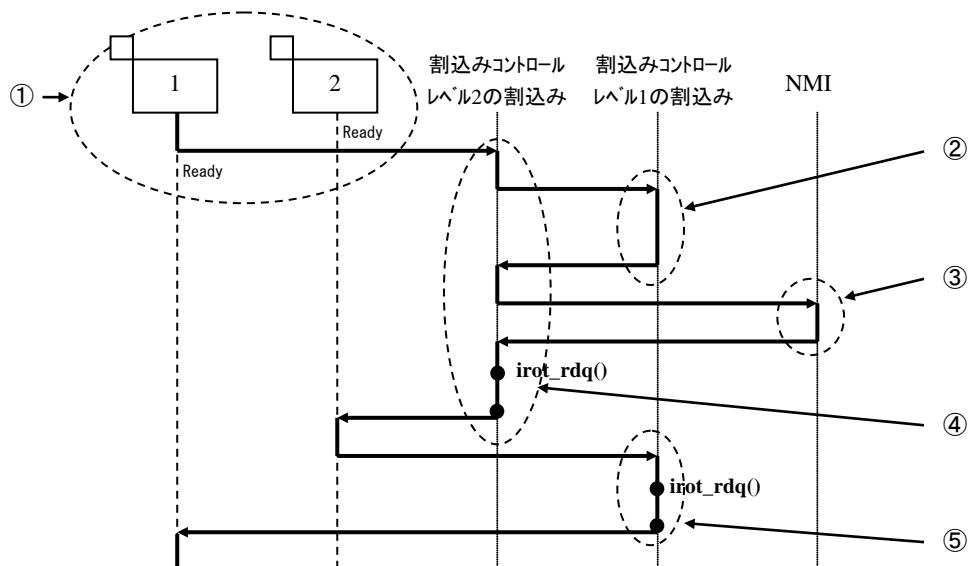


図3-16 多重割込み動作例

図3-17に示す様に多重割込みで下位レベルの disp 無割込みの場合、上位レベル割込みにて発行したサービスコールによる状態変更が反映されず、OS 管理情報が不正となる可能性が生じます。この場合、以後の動作は保証されません。

多重割込みを前提に割込みからサービスコールを発行する場合、カーネルマスクレベル以下の割込みは disp 有割込みとして実装してください(図3-18)。

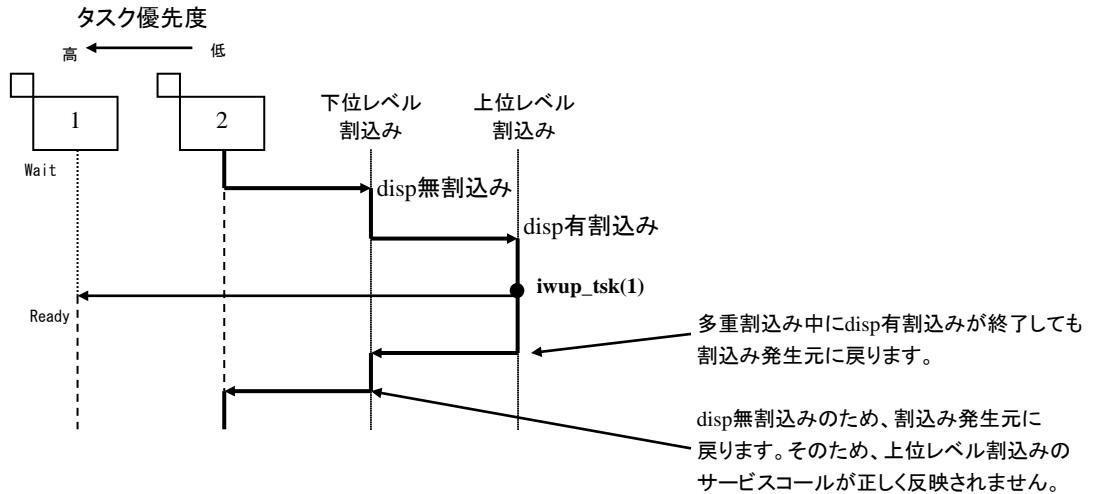


図3-17 多重割込み実装上の注意事項①

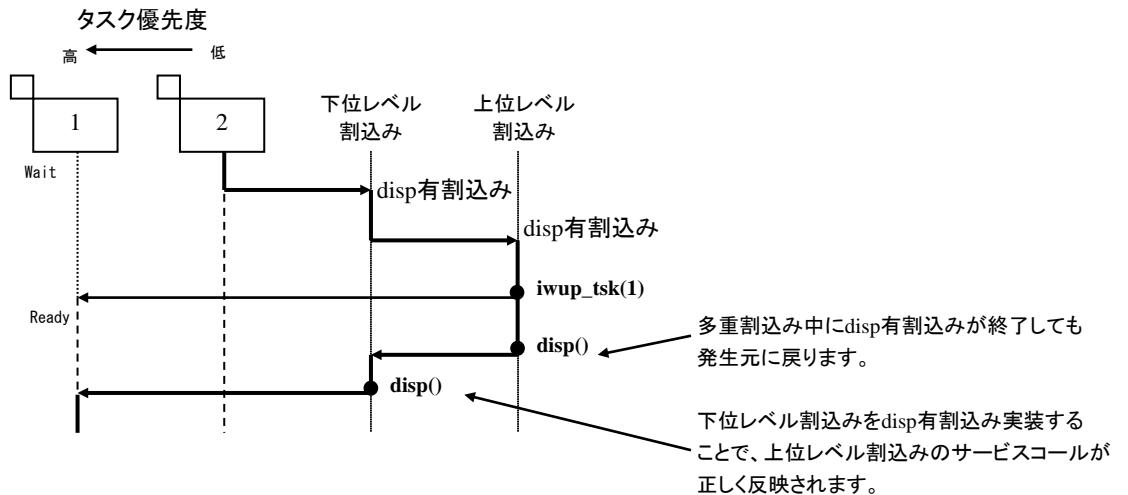


図3-18 多重割込み実装上の注意事項②

3.7 トレース

トレースはサービスコールやタスクディスパッチなどの履歴を保存します。取得したトレース情報からプログラムの実行状況を解析できます。トレース領域はリングバッファとなっており、古いトレース情報から上書きされます。

3.7.1 トレースの取得

トレース機能はデフォルトでは使用できません。トレース機能はOSライブラリの再構築することで自動的に取得できます。トレース機能を有効にする方法は「6.8 トレースを有効にする」を参照ください。

トレース機能で自動的に取得される情報、及び、トレースフォーマットの詳細は「付録 H トレース情報」を参照ください。

トレース機能では割込みの発生など個別に履歴を残したいタイミングで、set_trc サービスコールを発行することで、ユーザ独自のトレース情報を取得することが可能です。

3.7.2 トレースの参照

トレース情報の参照方法には以下の方法があります。

- (1) 直接メモリを参照する。エミュレータ等のデバッガや、デバッグモニタ機能(ユーザ実装)のメモリダンプ機能などでメモリを参照。
- (2) トレース参照機能(get_trc サービスコール)を利用したトレース情報の取得。
- (3) デバッガ提供メーカーにてサポートされる Smalight OS トレース参照機能の利用。利用可能なデバッガ製品情報の詳細は、弊社ホームページ(<http://www.systemtech.maxell.co.jp/solution/smalight-index/>)を参照ください。

3.7.3 トレースに関する注意事項

- (1) トレース機能を有効になると動作性能が低下します。
- (2) トレース機能を無効(デフォルト)にした時と、トレース機能を有効にした時では、動作タイミングが異なります。変更する際は、充分なシステム評価を行ってください。
- (3) メモリ資源が少ない環境では、充分なトレース領域が確保できない場合がございます。
- (4) 取得されないサービスコールがございます。詳細は「H.2 トレース種別」を参照ください。

4. サービスコール

4.1 システム状態とサービスコール

各サービスコールと発行可能なシステム状態の一覧を示します。

発行可能なシステム状態以外でサービスコールを発行した場合の動作は、保証されません。

表4-1 システム状態とサービスコール①

T:タスク部 I:割込み部 O:システム初期化部

区分	サービス コール	説明	発行可能なシステム状態		
			T	I	O
タスク管理	slp_tsk	タスクの起床待ち	○	×	×
	tslp_tsk	タスクの起床待ち(タイムアウト付き)	○	×	×
	wup_tsk	タスクの起床	○	×	×
	iwup_tsk	タスクの起床	×	○	×
	can_wup	タスク起床要求のキャンセル	○	○	×
	rot_rdq	タスクのローテーション	○	×	×
	irot_rdq	タスクのローテーション	×	○	×
	sus_tsk	他タスクのサスPEND	○	×	×
	isus_tsk	他タスクのサスPEND	×	○	×
	rsm_tsk	サスPENDの解除	○	×	×
イベントフラグ	wai_flg	イベントフラグ待ち	○	×	×
	twai_flg	イベントフラグ待ち(タイムアウト付)	○	×	×
	set_flg	イベントフラグの設定	○	×	×
	iset_flg	イベントフラグの設定	×	○	×
	clr_flg	イベントフラグのクリア	○	○	×
	evtflg_init	イベントフラグの初期化	×	×	○
	EVTFLG_ATTR ^(*)1)	イベントフラグの属性設定	×	×	○
セマフォ	wai_sem	セマフォの獲得	○	×	×
	twai_sem	セマフォの獲得(タイムアウト付)	○	×	×
	sig_sem	セマフォの返却	○	×	×
	isig_sem	セマフォの返却	×	○	×
	sem_init	セマフォの初期化	×	×	○
	SEM_ATTR ^(*)1)	セマフォの属性設定	×	×	○
データキュー	rcv_dtq	データキューからの受信	○	×	×
	trcv_dtq	データキューからの受信(タイムアウト付)	○	×	×
	snd_dtq	データキューへの送信	○	×	×
	isnd_dtq	データキューへの送信	×	○	×
	tsnd_dtq	データキューへの送信(タイムアウト付)	○	×	×
	fsnd_dtq	データキューへの強制送信	○	×	×
	ifsnd_dtq	データキューへの強制送信	×	○	×
	dtq_init	データキューの初期化	×	×	○
	DTQ_ATTR ^(*)1)	データキューの属性設定	×	×	○

※ 小文字のサービスコールはサブルーチンコールです。

(*)1) マクロです。

表4-2 システム状態とサービスコール②

区分	サービス コール	説明	発行可能なシステム状態		
			T	I	O
時間管理	set_tim	システム時刻の設定	○	○	○
	get_tim	システム時刻の取得	○	○	○
	systim_init	時間管理の初期化	×	×	○
	slos_cyclic_timer	周期タイマ処理	×	○	×
周期ハンドラ	sta_cyc	周期ハンドラの開始	○	○	×
	stp_cyc	周期ハンドラの停止	○	○	×
	cyc_init	周期ハンドラの初期化	×	×	○
	CYC_ATTR ^(*)1)	周期ハンドラの属性設定	×	×	○
	CYC_CHG ^(*)1)	周期ハンドラの起動周期変更	×	○	×
	callback_cychdr ^(*)1)	周期ハンドラ本体	—	—	—
割込み関連	callback_int	disp有割込みハンドラ本体	—	—	—
その他の初期化	slos_init	OSの起動	×	×	○
	kinit	OS初期化処理	—	—	—
	uinit	ユーザ初期化処理	—	—	—
	stack_init	タスクスタックの初期化	—	—	—
その他	idle	アイドル処理	—	—	—
	systemdown	システムダウン処理	—	—	—
トレース	set_trc	ユーザトレースの取得	○	○	×
	get_trc	トレースの参照	○	○	×
スタック操作	GET_REG ^(*)1)	タスクスタックからのレジスタ参照	○	○	×
	SET_REG ^(*)1)	タスクスタックのレジスタ設定	○	○	×

※ 小文字のサービスコールはサブルーチンコールです。

(*)1) マクロです。

4.2 各サービスコールの動作

4.2.1 サービスコール動作別の分類

各サービスコールを動作別に分類すると以下の様になります。以下、本分類に沿って各サービスコールの基本動作について説明します。

- ・タスクスイッチ型サービスコール
- ・i付きサービスコール
- ・関数型サービスコール

4.2.2 タスクスイッチ型サービスコールの基本動作

タスクスイッチ型サービスコールは、タスク部から発行するとタスクスイッチする可能性があるサービスコールです。タスクスイッチ型サービスコールは以下の通りです。

表4-3 タスクスイッチ型サービスコール

タスクスイッチ型 サービスコール	slp_tsk, tslp_tsk, wup_tsk, rot_rdq, sus_tsk, rsm_tsk, wai_flg, twai_flg, set_flg, wai_sem, twai_sem, sig_sem, rcv_dtq, trcv_dtq, snd_dtq, tsnd_dtq, fsnd_dtq
---------------------	---

タスクスイッチ型サービスコールの基本動作を説明します。

- ① タスクスタックにレジスタを退避します。
- ② カーネルマスクレベルに変更します。カーネルマスクレベルはコンフィギュレーションファイルの設定にて決定します。
- ③ スタックをOSスタックに切替えます。
- ④ 各サービスコールの処理を実行します。
- ⑤ 一時的にマスクレベルを解放します。これにより一時的に全ての保留中の割込みが実行されます。これは割込み禁止時間短縮のため、行われます。
- ⑥ ディスパッチャを実行します。ディスパッチャとは優先度やタスク状態に従い、実行するタスクを選択してタスクへ処理を遷移します。実行できるタスクがない場合は、アイドル状態となります。

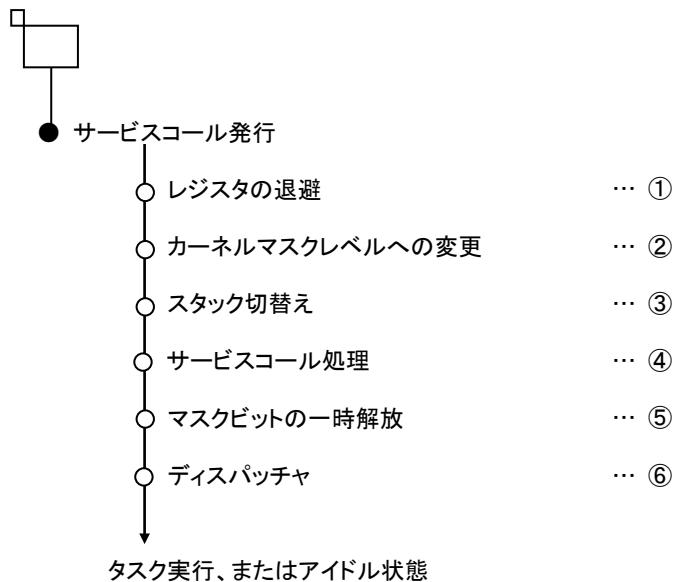


図4-1 タスクスイッチ型サービスコールの基本動作

4.2.3 i付きサービスコールの基本動作

i付きサービスコールは、割込み部から発行するサービスコールです。i付きサービスコールは以下の通りです。

表4-4 i付きサービスコール

i付きサービスコール	iwup_tsk, irot_rdq, isus_tsk, irsm_tsk, iset_flg, isig_sem, isnd_dtq, ifsnd_dtq
------------	--

i付きサービスコールの基本動作を説明します。

- ① カーネルマスクレベルに変更します。カーネルマスクレベルはコンフィギュレーションファイルの設定にて決定します。
- ② スタックを OS スタックに切替えます。
- ③ 各サービスコールの処理を実行します。
- ④ ②で変更したスタックを元に戻します。
- ⑤ ①で変更したマスクレベルを元に戻します。
- ⑥ 発行元へ戻ります。
- ⑦ i付きサービスコールではディスパッチャが実行されずに発行元へ戻ります。disp 有割込みハンドラの最後で必ず disp サービスコールを発行してください。

i付きサービスコールはタスクスイッチ型サービスコールと比較すると、ディスパッチャが実行されない等の違いがあります。

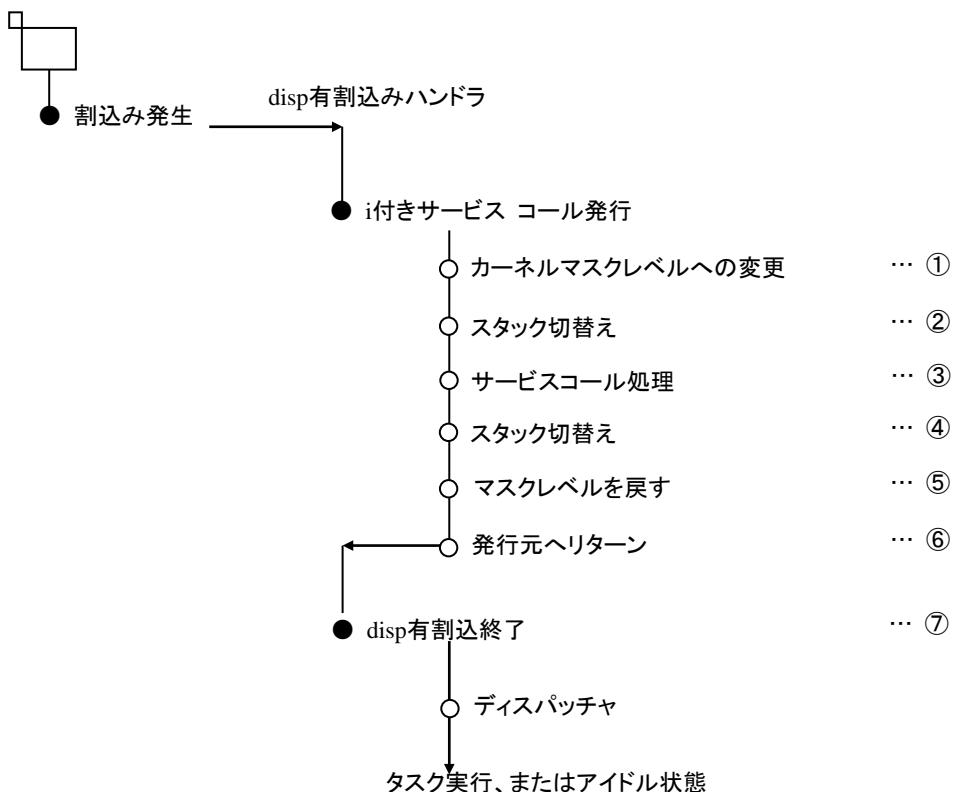


図4-2 i付きサービスコールの基本動作

4.2.4 関数型サービスコールの基本動作

タスクスイッチ型サービスコール、i付きサービスコールに分類されない関数型サービスコールは、通常の関数と同様の動作をします。

表4-5 関数型サービスコール

関数型 サービスコール	can_wup, clr_flg, set_tim, get_tim, sta_cyc, stp_cyc, set_trc, get_trc その他、初期化数など
----------------	--

4.3 サービスコールの説明形式

本節では、サービスコール仕様についての詳細な説明を以下の形式で行っています。

No. サービスコール名	機能	【発行可能なシステム状態】
C言語インターフェース		
マネージャコール呼出し形式		
パラメータ		
型	パラメータ	パラメータの意味
.	.	.
.	.	.
.	.	.
リターンパラメータ		
型	パラメータ	パラメータの意味
.	.	.
.	.	.
リターン値／エラーコード		
リターン値またはニモニック	リターン値またはエラーコード	リターン値またはエラーコードの意味
.	.	.
.	.	.
.	.	.
解説		
.....		

4.4 タスク管理

4.4.1 slp_tsk

タスクの起床待ち

【タスク部】

C言語インターフェース

```
W ercd = slp_tsk ( void );
```

パラメータ

無し

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd [① | ② | ③ | ④] (4 バイト)

①③④ 未使用(0)

② リターンコード

E_OK(H'00)

正常終了

解説

自タスクをWaiting状態(起床待ち)に移行します。このタスクの起床待ちは、wup_tskサービスコールにより解除されます。wup_tskサービスコールによるタスク起床要求は255回まで記憶されます(起床要求回数という)。起床要求回数が1以上のとき 本サービスコールを発行した場合は、起床要求回数がデクリメント(-1)され、Waiting状態(起床待ち)には移行せず、そのまま実行を続けます。

リターンパラメータercdには必ずE_OK(0)が返ります(※)。

本サービスコールはタスク部のみ発行できます。

(※) スタック操作支援サービスコールを利用することにより、リターンコードの書き換えが可能です。その場合、ユーザにて任意の意味付けをして使用してください。詳細は「G.1 GET_REG/SET_REGの活用」を参照ください。

4.4.2 tslp_tsk

タスクの起床待ち(タイムアウト付き)

【タスク部】

C言語インターフェース

```
W ercd = tslp_tsk (W tim);
```

パラメータ

W	tim	タイムアウト時間(msec)
---	-----	----------------

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①③④	未使用(0)				
②	リターンコード				
	E_OK(H'00)		正常終了		
	E_TMOOUT(H'CE)		ポーリング失敗、または、タイムアウト		

解説

自タスクをタイムアウト有りで Waiting 状態(起床待ち)に移行します。このタスクの起床待ちはタイムアウト、または、wup_tsk サービスコールにより解除されます。wup_tsk サービスコールによるタスク起床要求は 255 回まで記憶されます(起床要求回数という)。起床要求回数が 1 以上のとき 本サービスコールを発行した場合は、起床要求回数がデクリメント(-1)され、Waiting 状態(起床待ち)には移行せず、そのまま実行を続けます。そのときのリターンパラメータ ercd は E_OK(0)が返ります。

パラメータ tim には、タイムアウト時間(msec)を設定します。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、Waiting 状態(起床待ち)には移行せず、リターンパラメータ ercd は E_OK(0)が返ります。tim に TMO_FEVR(-1)を設定した場合は slp_tsk と同じ動作となります。

本サービスコールはタスク部のみ発行できます。

本サービスコールは Smalight のプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.4.3 wup_tsk, iwup_tsk

タスクの起床

【タスク部】 wup_tsk

【割込み部】 iwup_tsk

C言語インターフェース

```
void wup_tsk ( UB tid );
void iwup_tsk ( UB tid );
```

パラメータ

UB	tid	タスクID
----	-----	-------

リターンパラメータ

無し

解説

slp_tskサービスコール、または、tslp_tskサービスコールによりWaiting状態(起床待ち)になっているタスクの起床待ちを解除します。起床待ちでないタスクを指定した場合、タスク起床要求が記憶されます。タスク起床要求は255回まで記憶されます(起床要求回数という)。

パラメータtidには起床させたいタスクのタスクIDを指定してください。tidに0(アイドル)や、存在しないタスクのタスクIDを指定しないでください。

wup_tskサービスコールはタスク部から発行します。

iwup_tskサービスコールは割込み部から発行します。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.4.4 can_wup

タスク起床要求のキャンセル

【タスク部】【割込み部】

C言語インターフェース

```
W ercd = can_wup ( UB tid );
```

パラメータ

UB	tid	タスクID
----	-----	-------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①③	未使用(0)				
②	リターンコード				
		E_OK(H'00)			正常終了
④					キャンセルした起床要求回数(8bit)

解説

記憶されたタスクの起床要求をキャンセルし、キャンセルした起床要求回数を返します。起床要求は255回まで記憶されます(起床要求回数という)。

パラメータtidには起床要求をキャンセルしたいタスクのタスクIDを指定してください。tidに0(アイドル)をした場合、自タスク対象タスクとします。tidには存在しないタスクのタスクIDを指定しないでください。割込み部からcan_wupサービスコールを発行する場合、パラメータtidに0(アイドル)を指定しないでください。

リターンパラメータ ercd にはリターンコード、及び、キャンセルした起床要求の回数が返ります。リターンコード E_OK(0)のときのみ、キャンセルした起床要求の回数が返ります。起床要求が記憶されていない場合は0となります。

本サービスコールは関数型サービスコールでタスク部、割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.4.5 rot_rdq, irot_rdq

タスクのローテーション

【タスク部】 rot_rdq
【割込み部】 irot_rdq

C言語インターフェース

```
void rot_rdq( void );
void irot_rdq( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

ローテーションタスクのレディキューはFCFS(First Come First Service)で管理しています。本サービスコール発行によりローテーションタスクのレディキュー先頭タスクをレディキュー最後尾につなぎかえます。

ローテーションタスクのレディキューは、実行中(Running状態)のローテーションタスクがWaiting状態になるか、本サービスコールを発行しない限り、他のローテーションタスクが実行されません。本サービスコールを用いて、意図的にローテーションタスクのレディキューを操作する必要があります。

本サービスコールをプライオリティタスクが実行中(Running状態)に発行した場合でも、ローテーションタスクのレディキュー操作を行います。そのときローテーションタスクのレディキュー先頭にいたタスクは無条件にレディーキュー最後尾に移動されますので、実行されないまま先頭から最後尾に移動される可能性があります。

rot_rdqサービスコールはタスク部から発行します。

irot_rdqサービスコールは割込み部から発行します。

4.4.6 sus_tsk, isus_tsk

他タスクのサスPEND

【タスク部】sus_tsk

【割込み部】isus_tsk

C言語インターフェース

```
void sus_tsk ( UB tid );
void isus_tsk ( UB tid );
```

パラメータ

UB	tid	タスクID
----	-----	-------

リターンパラメータ

無し

解説

tidで指定したタスクをSuspended状態に移行させます。Suspended状態は、rsm_tskサービスコールにより解除されれます。

サスPEND要求は他の要因でWaiting状態のタスクに対しても有効です。他の要因でWaiting状態のタスクへサスPEND要求を行うと、重複した待ち状態(Waiting + Suspended状態)となります。その場合、両方の要因が解除されないとReady状態になりません。Suspended状態のタスクに対してサスPEND要求しても、重複したSuspended状態にならないので、一回のrsm_tskサービスコールでSuspended状態が解除されます。

パラメータtidにはサスPENDさせたいタスクのタスクIDを指定してください。tidに0(アイドル)や、存在しないタスクのタスクIDを指定しないでください。

sus_tskサービスコールはタスク部から発行します

isus_tskサービスコールは割込み部から発行します。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.4.7 rsm_tsk, irsm_tsk

サスPENDの解除

【タスク部】 rsm_tsk

【割込み部】 irsm_tsk

C言語インターフェース

```
void rsm_tsk ( UB tid );
void irsm_tsk ( UB tid );
```

パラメータ

UB	tid	タスクID
----	-----	-------

リターンパラメータ

無し

解説

sus_tskサービスコールによりSuspended状態になっているタスクのサスPENDを解除します。Suspended状態でないタスクを指定した場合、その要求は無視され、一切記憶されません。

パラメータtidにはサスPENDを解除させたいタスクのタスクIDを指定してください。tidに0(アイドル)や、存在しないタスクのタスクIDを指定しないでください。また、Running状態のタスクが自タスクへのrsm_tskサービスコールを発行してはいけません。

rsm_tskサービスコールはタスク部から発行します。

irsm_tskサービスコールは割込み部から発行します。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.5 イベントフラグ

4.5.1 wai_flg

イベントフラグ待ち

【タスク部】

C言語インターフェース

```
W ercd = wai_flg (UB fid, UH ptn);
```

パラメータ

UB	fid	イベントフラグID
UH	ptn	待ちビットパターン(16bit)

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①	未使用(0)				
②	リターンコード				
	E_OK(H'00)				正常終了
③④	待ち解除時のビットパターン(16bit)				

解説

fid で指定されるイベントフラグが、ptn で指定される待ちビットパターンの待ち解除条件を満たすのを待ちます。既に待ち解除要因を満たす場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ ptn には、16 ビットの待ちビットパターンを設定します。

本サービスコールはタスク部のみ発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.5.2 twai_flg

イベントフラグ待ち(タイムアウト付き)

【タスク部】

C言語インターフェース

```
W ercd = twai_flg(UB fid, UH ptn, W tim);
```

パラメータ

UB	fid	イベントフラグID
UH	ptn	待ちビットパターン(16bit)
W	tim	タイムアウト時間(msec)

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①	未使用(0)				
②	リターンコード				
	E_OK(H'00)	正常終了			
	E_TMOOUT(H'CE)	ポーリング失敗、または、タイムアウト			
③④	待ち解除時のビットパターン(16bit)				

解説

fid で指定されるイベントフラグが、ptn で指定される待ちビットパターンの待ち解除条件を満たすか、タイムアウトするのを待ちます。既に待ち解除要因を満たす場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ ptn には、16 ビットの待ちビットパターンを設定します。

パラメータ tim には、タイムアウト時間(msec)を設定します。待ちビットパターンの待ち解除条件を満たさない場合、tim で指定したタイムアウト時間が経過した時点で待ちが解除されます。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、イベントフラグをポーリングして戻ります。tim に TMO_FEVR(-1)を設定した場合は wai_flg と同じ動作となります。

本サービスコールはタスク部のみ発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.5.3 set_flg, iset_flg

イベントフラグの設定

【タスク部】 set_flg
【割込み部】 iset_flg

C言語インターフェース

```
void set_flg (UB fid, UH ptn);  
void iset_flg (UB fid, UH ptn);
```

パラメータ

UB	fid	イベントフラグID
UH	ptn	セットするビットパターン(16bit)

リターンパラメータ

無し

解説

fid で指定されるイベントフラグを、ptn で指定された値との論理和(OR)で更新します。更新の結果、待ちタスクの解除条件を満たせば、待ちタスクを Waiting 状態から Ready 状態に移行します。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ ptn には、16 ビットのセットするビットパターンを設定します。

set_flg はタスク部から発行できます。

iset_flg は割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.5.4 clr_flg

イベントフラグのクリア

【タスク部】【割込み部】

C言語インターフェース

```
void clr_flg (UB fid, UH ptn);
```

パラメータ

UB	fid	イベントフラグID
UH	ptn	クリアするビットパターン(16bit)

リターンパラメータ

無し

解説

fid で指定されるイベントフラグを、ptn で示された値との論理積(AND)に更新します。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ ptn には、16 ビットのクリアするビットパターンを設定します。

本サービスコールは関数型サービスコールでタスク部、割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.5.5 evtflg_init

イベントフラグの初期化

【システム初期化部】

C言語インターフェース

```
void evtflg_init ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

イベントフラグに関する待ちキューなどの初期化を行います。イベントフラグを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

4.5.6 EVTFLG_ATTR

イベントフラグ属性の設定(マクロ)

【システム初期化部】

C言語インターフェース

```
void EVTFLG_ATTR (UB fid, UB attr);
```

パラメータ

UB	fid	イベントフラグID
UB	attr	イベントフラグ属性

リターンパラメータ

無し

解説

fid で指定されるイベントフラグの属性を設定します。イベントフラグ ID 毎に属性設定が可能です。

パラメータ fid にはイベントフラグ ID を指定します。fid には 0 や、定義したイベントフラグ数を超える値を設定しないでください。

パラメータ attr でイベントフラグ属性を設定します。設定できる属性は以下の通りです。

- ・ イベントフラグ待ちの待ちキュー設定
EVFLG_TA_TFIFO(FIFO 順、H'00)、または、EVFLG_TA_TPRI(優先度順、H'01)
- ・ イベントフラグの待ち条件
EVFLG_TA_AND(全てのビットが揃うまで、H'00)、
または、
EVFLG_TA_OR(いずれかのビットが揃うまで、H'04)
- ・ 待ち解除時のイベントフラグクリア属性
EVFLG_TA_CLR(H'02)

イベントフラグ初期化したとき、イベントフラグ属性の初期値は(EVFLG_TA_TFIFO | EVFLG_TA_AND)です。本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックにて evtflg_init サービスコール後に発行してください。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.6 セマフォ

4.6.1 wai_sem

セマフォの獲得

【タスク部】

C言語インターフェース

```
W ercd = wai_sem (UB sid);
```

パラメータ

UB	sid	セマフォID
----	-----	--------

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd [① | ② | ③ | ④] (4 バイト)

①③④ 未使用(0)

② リターンコード

E_OK(H'00)

正常終了

解説

sid で指定されるセマフォを獲得します。セマフォの獲得ができない場合 Waiting 状態となり、セマフォが返却されるのを待ちます。セマフォを獲得できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ sid にはセマフォ ID を指定します。sid には 0 や、定義したセマフォ数を超える値を設定しないでください。

本サービスコールはタスク部のみ発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.6.2 twai_sem

セマフォの獲得(タイムアウト付き)

【タスク部】

C言語インターフェース

```
W ercd = twai_sem (UB sid, W tim);
```

パラメータ

UB	sid	セマフォID
W	tim	タイムアウト時間(msec)

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①③④	未使用(0)				
②	リターンコード				
	E_OK(H'00)		正常終了		
	E_TMOOUT(H'CE)		ポーリング失敗、または、タイムアウト		

解説

sid で指定されるセマフォを獲得します。セマフォの獲得ができない場合 Waiting 状態となり、セマフォが返却されるか、タイムアウトするのを待ちます。セマフォを獲得できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ sid にはセマフォ ID を指定します。sid には 0 や、定義したセマフォ数を超える値を設定しないでください。

パラメータ tim には、タイムアウト時間(msec)を設定します。セマフォの獲得ができない場合、tim で指定したタイムアウト時間が経過した時点で待ちが解除されます。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、セマフォ獲得をポーリングして戻ります。tim に TMO_FEVR(-1)を設定した場合は wai_sem と同じ動作となります。

本サービスコールはタスク部のみ発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.6.3 sig_sem, isig_sem

セマフォの返却

【タスク部】 sig_sem

【割込み部】 isig_sem

C言語インターフェース

```
void sig_sem (UB sid);  
void isig_sem (UB sid);
```

パラメータ

UB	sid	セマフォID
----	-----	--------

リターンパラメータ

無し

解説

sid で指定されるセマフォを返却します。セマフォ待ちのタスクがあれば、Waiting 状態から Ready 状態に移行します(セマフォ待ちタスクのセマフォ獲得順序は、セマフォ属性により決定します)。獲得していないセマフォを返却してはいけません。

パラメータ sid にはセマフォ ID を指定します。sid には 0 や、定義したセマフォ数を超える値を設定しないでください。

sig_sem はタスク部から発行できます。

isig_sem は割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.6.4 sem_init

セマフォの初期化

【システム初期化部】

C言語インターフェース

```
void sem_init(void);
```

パラメータ

無し

リターンパラメータ

無し

解説

セマフォに関する待ちキューなどの初期化を行います。セマフォを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

4.6.5 SEM_ATTR

セマフォ属性の設定(マクロ)

【システム初期化部】

C言語インターフェース

```
void SEM_ATTR (UB sid, UB semcnt, UB attr);
```

パラメータ

UB	sid	セマフォID
UB	semcnt	セマフォ資源数の初期値
UB	attr	セマフォ属性

リターンパラメータ

無し

解説

sid で指定されるセマフォの属性を設定します。セマフォ ID 毎に属性設定が可能です。

パラメータ sid にはセマフォ ID を指定します。sid には 0 や、定義したセマフォ数を超える値を設定しないでください。

パラメータ semcnt でセマフォ資源数の初期値を設定します。0~127 の値を設定してください。

パラメータ attr でセマフォ属性を設定します。設定できる属性は以下の通りです。

- セマフォ待ちの待ちキュー設定
SEM_TA_TFIFO(FIFO 順、H'00)、または、SEM_TA_TPRI(優先度順、H'01)

セマフォ初期化したとき、セマフォ属性の初期値は、セマフォ資源数が 1、セマフォ待ちの待ちキュー設定が SEM_TA_TFIFO(FIFO 順)となります。本サービスコールはシステム初期化部からのみ発行できます。unit コールバックにて sem_init サービスコール後に発行してください。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.7 データキュー

4.7.1 rcv_dtq

データキューからの受信

【タスク部】

C言語インターフェース

```
W ercd = rcv_dtq(UB qid, W *data);
```

パラメータ

UB	qid	データキューID
W	*data	受信データを格納する領域のポインタ

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①③④	未使用(0)				
②	リターンコード				
	E_OK(H'00)				正常終了

解説

qid で指定されるデータキューからデータを受信し、data に返します。データキューからのデータ受信した結果、送信待ちタスクがいれば、データキューへの送信を行い Waiting 状態から Ready 状態に移行します。

データキューにデータがない場合 Waiting 状態となり、データキューにデータが送信されるのを待ちます。データキューからデータを受信できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ *data には受信データを格納する領域のポインタを指定します。受信したデータは *data で指定した領域に格納されます。

本サービスコールはタスク部のみ発行できます。

本サービスコールは Smalight のプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.7.2 trcv_dtq

データキューからの受信(タイムアウト付き)

【タスク部】

C言語インターフェース

W ercd = trcv_dtq (UB qid, W *data, W tim);

パラメータ

UB	qid	データキューID
W	*data	受信データを格納する領域のポインタ
W	tim	タイムアウト時間(msec)

リターンパラメータ

W ercd リターンコード

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①③④	未使用(0)				
②	リターンコード				
	E_OK(H'00)	正常終了			
	E_TMOOUT(H'CE)	ポーリング失敗、または、タイムアウト			

解説

qid で指定されるデータキューからデータを受信し、data に返します。データキューからのデータ受信した結果、送信待ちタスクがいれば、データキューへの送信を行い Waiting 状態から Ready 状態に移行します。

データキューにデータがない場合 Waiting 状態となり、データキューにデータが送信されるのを待ちます。データキューからデータを受信できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ *data には受信データを格納する領域のポインタを指定します。受信したデータは *data で指定した領域に格納されます。

パラメータ tim には、タイムアウト時間(msec)を設定します。データキューからの受信ができない場合、tim で指定したタイムアウト時間が経過した時点で待ちが解除されます。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、データキュー受信をポーリングして戻ります。tim に TMO_FEVR(-1)を設定した場合は rcv_dtq と同じ動作となります。

本サービスコールはタスク部のみ発行できます。

本サービスコールは Smalight のプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.7.3 snd_dtq, isnd_dtq

データキューへの送信

【タスク部】 snd_dtq

【割込み部】 isnd_dtq

C言語インターフェース

```
W ercd = snd_dtq (UB qid, W data);  
W ercd = isnd_dtq (UB qid, W data);
```

パラメータ

UB	qid	データキューID
W	data	送信データ

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①③④	未使用(0)				
②	リターンコード				
	E_OK(H'00)		正常終了		
	E_TMOUT(H'CE)		ポーリング失敗、または、タイムアウト		

解説

qid で指定されるデータキューに、data で指定されるデータを送信します。データキューへデータ送信した結果、受信待ちタスクがいれば、受信待ちタスクの先頭のタスクにデータを渡して、Waiting 状態から Ready 状態に移行します。

データキューに空きがない場合、Waiting 状態となり、データキューに空きができるまで待ちます。データキューへの送信できた場合は、Waiting 状態にならずそのまま実行を続けます。

(但し、isnd_dtq でデータキューに空きがない場合、待ちには入らずリターンパラメタ ercd に E_TMOUT(H'CE) が返ります)

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ data には送信データを指定します。

snd_dtq はタスク部から発行できます。

isnd_dtq は割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.7.4 tsnd_dtq

データキューへの送信(タイムアウト付き)

【タスク部】

C言語インターフェース

```
W ercd = tsnd_dtq (UB qid, W data, W tim);
```

パラメータ

UB	qid	データキューID
W	data	受信データ
W	tim	タイムアウト時間(msec)

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①③④	未使用(0)				
②	リターンコード				
	E_OK(H'00)	正常終了			
	E_TMOOUT(H'CE)	ポーリング失敗、または、タイムアウト			

解説

qid で指定されるデータキューに、data で指定されるデータを送信します。データキューへデータ送信した結果、受信待ちタスクがいれば、受信待ちタスクの先頭のタスクにデータを渡して、Waiting 状態から Ready 状態に移行します。

データキューに空きがない場合、Waiting 状態となり、データキューに空きができるまで待ちます。データキューへの送信できた場合は、Waiting 状態にならずそのまま実行を続けます。

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ data には送信データを指定します。

パラメータ tim には、タイムアウト時間(msec)を設定します。データの送信ができない場合、tim で指定したタイムアウト時間が経過した時点で待ちが解除されます。設定できる時間の最大値は H'7fffffff です。tim に TMO_POL(0)を設定した場合、データキュー受信をポーリングして戻ります。tim に TMO_FEVR(-1)を設定した場合は snd_dtq と同じ動作となります。

本サービスコールはタスク部のみ発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.7.5 fsnd_dtq, ifsnd_dtq

データキューへの強制送信

【タスク部】 fsnd_dtq
【割込み部】 ifsnd_dtq

C言語インターフェース

```
W ercd = fsnd_dtq (UB qid, W data);
W ercd = ifsnd_dtq (UB qid, W data);
```

パラメータ

UB	qid	データキューID
W	data	送信データ

リターンパラメータ

W	ercd	リターンコード
---	------	---------

リターン値／エラーコード

ercd	①	②	③	④	(4 バイト)
①③④	未使用(0)				
②	リターンコード				
	E_OK(H'00)				正常終了
	E_ILUSE(H'E4)				データ数が 0 のデータキューを指定した

解説

qid で指定されるデータキューに、data で指定されるデータを強制送信します。データキューへデータ送信した結果、受信待ちタスクがいれば、受信待ちタスクの先頭のタスクにデータを渡して、Waiting 状態から Ready 状態に移行します。

データキューに空きがない場合には、データキューに格納される一番古いデータを抹消し空き領域を確保してから、データキューへの送信を行います。指定したデータキューのデータ数が 0 の場合、リターンパラメータ ercd に E_ILUSE(H'E4)を返します。

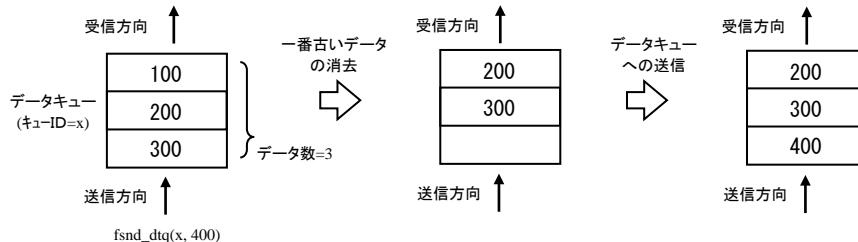


図4-3 データキューへの強制送信時、空きがない場合の動作(データ数=3)

パラメータ qid には受信対象のデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ data には送信データを指定します。

fsnd_dtq はタスク部から発行できます。

ifsnd_dtq は割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.7.6 dtq_init

データキューの初期化

【システム初期化部】

C言語インターフェース

```
void dtq_init(void);
```

パラメータ

無し

リターンパラメータ

無し

解説

データキューに関する待ちキューなどの初期化を行います。データキューを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

4.7.7 DTQ_ATTR

データキューの属性設定

【システム初期化部】

C言語インターフェース

```
void DTQ_ATTR (UB qid, UB attr, B cnt, W *dtq);
```

パラメータ

UB	qid	データキューID
UB	attr	データキュー属性
B	cnt	データ数
W	dtq	データキュー領域の先頭番地

リターンパラメータ

無し

解説

qid で指定されるデータキューの属性を設定します。データキューID 毎に属性設定が可能です。

パラメータ qid にはデータキューID を指定します。qid には 0 や、定義したデータキュー数を超える値を設定しないでください。

パラメータ attr でデータキュー属性を設定します。設定できる属性は以下の通りです。

- データキュー待ち(データ送信時)の待ちキュー設定

DTQ_TA_TFIFO(FIFO 順、H'00)、または、DTQ_TA_TPRI(優先度順、H'01)

※ データ受信時の待ちキューは、常に FIFO 順でキューイングされます。

パラメータ cnt でデータの数を指定します。0~127 の値を設定してください。

パラメータ dtq でデータキューの先頭番地を指定します。データ数分の 32bit 配列を用意し、本パラメータに指定してください。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行つております。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.8 時間管理

4.8.1 set_tim

システム時刻の設定

【タスク部】【割込み部】

C言語インターフェース

```
void set_tim (SYSTIM *tim);
```

パラメータ

SYSTIM	*tim	システム時刻(msc)を格納した領域のポインタ
--------	------	-------------------------

```
typedef struct _systim {  
    UH htim;      /* システム時刻 上位16ビット */  
    UW ltim;      /* システム時刻 下位32ビット */  
} SYSTIM;
```

リターンパラメータ

無し

解説

システム時刻を設定します。単位は msec です。

システム時刻は 48bit(符号無し)で管理しています。

本サービスコールは関数型サービスコールでタスク部、割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.8.2 get_tim

システム時刻の取得

【タスク部】【割込み部】

C言語インターフェース

```
void get_tim (SYSTIM *tim);
```

パラメータ

SYSTIM	*tim	システム時刻(msc)を格納する領域のポインタ
--------	------	-------------------------

```
typedef struct _systim {  
    UH htim;      /* システム時刻 上位16ビット */  
    UW ltim;      /* システム時刻 下位32ビット */  
} SYSTIM;
```

リターンパラメータ

無し

解説

システム時刻を取得します。単位は msec です。

システム時刻は 48bit(符号無し)で管理しています。

本サービスコールは関数型サービスコールでタスク部、割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.8.3 systim_init

時間管理の初期化

【システム初期化部】

C言語インターフェース

```
void systim_init ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

時間管理に関する待ちキューなどの初期化を行います。時間管理、周期ハンドラを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinitコールバックから発行してください。

4.8.4 slos_cyclic_timer

周期タイマ処理

【割込み部】

C言語インターフェース

```
void slos_cyclic_timer ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

周期タイマハンドラ(割込み部)から本サービスコールを発行します。時間管理の周期タイマ処理を実行します。

4.9 周期ハンドラ

4.9.1 sta_cyc

周期ハンドラの開始

【タスク部】【割込み部】

C言語インターフェース

```
void sta_cyc (UB cid);
```

パラメータ

UB	cid	周期ハンドラID
----	-----	----------

リターンパラメータ

無し

解説

cid で指定される周期ハンドラの動作を開始します。動作開始により、周期時間のカウントが開始され、周期時間が経過したタイミングで周期ハンドラ関数が実行されます。既に周期ハンドラが開始されている場合は何も行いません。

パラメータ cid には周期ハンドラ ID を指定します。cid には 0 や、定義した周期ハンドラ数を超える値を設定しないでください。

本サービスコールは関数型サービスコールでタスク部、割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.9.2 stp_cyc

周期ハンドラの停止

【タスク部】【割込み部】

C言語インターフェース

```
void stp_cyc (UB cid);
```

パラメータ

UB	cid	周期ハンドラID
----	-----	----------

リターンパラメータ

無し

解説

cid で指定される周期ハンドラの動作を停止します。周期ハンドラ停止中の場合は何も行いません。

パラメータ cid には周期ハンドラ ID を指定します。cid には 0 や、定義した周期ハンドラ数を超える値を設定しないでください。

本サービスコールは関数型サービスコールでタスク部、割込み部から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.9.3 cyc_init

周期ハンドラの初期化

【システム初期化部】

C言語インターフェース

```
void cyc_init ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

周期ハンドラに関する管理領域などの初期化を行います。周期ハンドラを使用する場合、必ず、初期化が必要となります。

本サービスコールはシステム初期化部からのみ発行できます。kinitコールバックから発行してください。

4.9.4 CYC_ATTR

周期ハンドラの属性設定

【システム初期化部】

C言語インターフェース

```
void CYC_ATTR (UB cid, UB attr, void (*hdr)(UB), W cyc);
```

パラメータ

UB	cid	周期ハンドラID
UB	attr	周期ハンドラ属性
void (*) (UB)	hdr	周期ハンドラ関数
W	cyc	周期時間(msec)

リターンパラメータ

無し

解説

cid で指定される周期ハンドラの属性を設定します。周期ハンドラ ID 毎に属性設定が可能です。

パラメータ cid には周期ハンドラ ID を指定します。cid には 0 や、定義した周期ハンドラ数を超える値を設定しないでください。

パラメータ attr で周期ハンドラ属性を設定します。設定できる属性は以下の通りです。

- 周期ハンドラの初期動作設定

CYC_TA_NON (初期状態で周期ハンドラ動作が停止、H'00)

または

CYC_TA_STA (初期状態で周期ハンドラ動作が開始、H'01)

パラメータ hdr で周期ハンドラ関数を指定します。

パラメータ cyc で周期時間(msec)を指定します。設定できる時間の最大値は H'7fffffff です。周期時間が 0 に設定されている場合は起動しても、開始されません。

本サービスコールはシステム初期化部からのみ発行できます。kinit コールバックから発行してください。

本サービスコールは Smalight のプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.9.5 CYC_CHG

周期ハンドラの起動周期変更

【タスク部】【割込み部】【システム初期化部】

C言語インターフェース

```
void CYC_CHG (UB cid, W cyc);
```

パラメータ

UB	cid	周期ハンドラID
W	cyc	周期時間(msec)

リターンパラメータ

無し

解説

cid で指定される周期ハンドラの起動周期を変更します。μTRON4.0 仕様「周期ハンドラの起動位相」の実装や、起動周期の動的変更が可能です。

パラメータ cid には周期ハンドラ ID を指定します。cid には 0 や、定義した周期ハンドラ数を超える値を設定しないでください。

パラメータ cyc で周期時間(msec)を指定します。設定できる時間の最大値は H'7fffffff です。周期時間が 0 に設定されている場合は起動しても、開始されません。

本サービスコールは割込み部(周期ハンドラ実行中)から発行できます。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行つております。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.9.6 callback_cychdr

周期ハンドラ本体(コールバック)

【-】

C言語インターフェース

```
void callback_cychdr ( UB cid );
```

パラメータ

UB	cid	周期ハンドラID
----	-----	----------

リターンパラメータ

無し

解説

周期ハンドラ本体(コールバック)です。関数名は任意の名称です。

周期ハンドラの動作を開始し、指定した周期時間が経過した時点で、本コールバックが呼び出されます。任意の周期ハンドラ処理を記載してください。本コールバックが実行されるとき、システム状態は割込み部です。

パラメータ cid には周期ハンドラ ID が設定されます。

使用方法の詳細は「3.5.3 周期ハンドラ」を参照ください。

4.10 割込み処理支援

4.10.1 callback_int disp 有割込みハンドラ本体

【-】

C言語インターフェース

```
void callback_int ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

disp割込みハンドラ本体です。関数名は任意の名称です。

割込み要因のクリアなど必要な処理を行ってください。本コールバック関数はdisp有割込みハンドラに使用します。実行中のシステム状態は割込み部です。

使用方法の詳細は「5.4 割込みハンドラ」を参照ください。

4.11 その他の初期化

4.11.1 slos_init OS の起動

【システム初期化部】

C言語インターフェース

```
void slos_init ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

Smalight OSを起動します。main関数から、割込み禁止状態で、slos_init サービスコールを発行してください。呼び出し元へは戻りません。詳細は「5.2.4 コード生成(設計ツール)」(2)を参照ください。

4.11.2 kinit

OS 初期化処理(コールバック)

【-】

C言語インターフェース

```
void kinit ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

slos_init(OS初期化処理)から呼び出されるコールバックです。イベントフラグやセマフォの初期化・属性設定などのOS初期化を行ってください。本コールバックが実行されるとき、システム状態はシステム初期化部です。

本コールバックルーチンの詳細は「5.2.6 初期化処理」を参照ください。呼び出されたとき、スタックはOSスタックを使用しています。

4.11.3 uinit

ユーザ初期化処理(コールバック)

【-】

C言語インターフェース

```
void uinit ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

slos_init(OS初期化処理)から呼び出されるコールバックです。ユーザシステムに応じた初期化を行ってください。本コールバックが実行されるとき、システム状態はシステム初期化部です。

呼び出されたとき、スタックはOSスタックを使用しています。必要に応じてOSスタックエリアサイズを大きくしてください。設定方法は「5.2.6 初期化処理」を参照ください。

4.11.4 stack_init

タスクスタックの初期化(コールバック)

【-】

C言語インターフェース

```
void stack_init ( UB tid, TSTACK *sp );
```

パラメータ

UB	tid	タスクID
TSTACK	*sp	スタックポインタ

リターンパラメータ

無し

解説

OS初期化処理で呼び出されるコールバックです。各タスクの開始アドレス等の情報を各タスクのスタックに格納する処理を行います。登録したタスク数分だけ発行されます。本コールバックが実行されるとき、システム状態はシステム初期化部です。

4.12 その他

4.12.1 idle

アイドル処理(コールバック)

【-】

C言語インターフェース

```
void idle ( void );
```

パラメータ

無し

リターンパラメータ

無し

解説

アイドル状態(実行可能なタスクが存在しない)のとき、実行されるコールバックルーチンです。割込みマスクを空けて、ループ処理には入ります。

省電力モードへの対応など、必要に応じて本コールバックルーチンを変更してください。

呼び出されたとき、スタックはOSスタックを使用しています。必要に応じてOSスタックエリアサイズを大きくしてください。

4.12.2 systemdown

システムダウン処理(コールバック)

【-】

C言語インターフェース

```
void systemdown ( UH code, UH detail );
```

パラメータ

UH	code	システムダウンコード
UH	detail	詳細コード

リターンパラメータ

無し

解説

システムダウン時に実行されるコールバックルーチンです。割込み禁止状態で呼出されます。ユーザ側でコールすることも可能ですが、その場合、割込み禁止状態で発行してください。

以下に予め登録されるシステムダウンの要因を示します。

code	detail	要因
0xFFFFFu	0x0000u	不正な多重有割込み受付(255回のdisp有割込みの多重受付)

4.13 トレース

4.13.1 set_trc ユーザトレースの取得

【タスク部】【割込み部】

C言語インターフェース

```
void set_trc (UB dat1, UB dat2, UW dat3);
```

パラメータ

UB	dat1	ユーザトレース情報1
UB	dat2	ユーザトレース情報2
UW	dat3	ユーザトレース情報3

リターンパラメータ

無し

解説

ユーザ独自のトレースを取得します。set_trc サービスコールで取得したトレースはトレースフォーマットの種別(Type)に'U'(H'55)が設定されます。トレースフォーマットの詳細は「付録 H トレース情報」を参照ください。

パラメータ dat1, dat2, dat3 は取得される情報を識別するためにユーザが自由に使用できます。例えば、

dat1=ユーザトレース種別(1:割込み)

dat2=ベクタ番号

dat3=割込み毎の詳細情報

などの情報識別をすることで、システム動作履歴として独自のトレース情報を取得できます。

本サービスコールは関数型サービスコールでタスク部、割込み部から発行できます。

本サービスコールを使用する場合、OS再構築(リビルド)が必要です。詳細は「6.8 トレースを有効にする」を参照ください。

4.13.2 get_trc

ト雷斯の参照

【タスク部】【割込み部】

C言語インターフェース

```
void get_trc (W no, KNL_TRCTBL **trc);
```

パラメータ

W	no	トレス番号
KNL_TRCTBL	**trc	トレス情報アドレスを格納する領域のポインタ

【時間管理を使用していない場合】

```
typedef struct _knl_trctbl {  
    UB      type;      /* トレス種別 */  
    UB      tid;       /* タスクID */  
    UB      dat1;      /* トレス情報1 */  
    UB      dat2;      /* トレス情報2 */  
    UW      dat3;      /* トレス情報3 */  
} KNL_TRCTBL;
```

【時間管理を使用している場合】

```
typedef struct _knl_trctbl {  
    UB      type;      /* トレス種別 */  
    UB      tid;       /* タスクID */  
    UB      dat1;      /* トレス情報1 */  
    UB      dat2;      /* トレス情報2 */  
    UW      dat3;      /* トレス情報3 */  
    UH      rsv;       /* 未使用 */  
    UH      htim;      /* システム時刻 上位16ビット */  
    UW      ltim;      /* システム時刻 下位32ビット */  
} KNL_TRCTBL;
```

リターンパラメータ

無し

解説

パラメータ no で指定したトレス情報を取得します。

パラメータ no にはトレス番号を指定します。トレス番号は 0 から N-1^{(*)1}まで指定できます。トレス番号が 0 のとき一番古いトレス情報となり、N-1^{(*)1}のとき一番新しいトレス情報となります。N は GET_TRCNUM マクロで取得できます。

(*)1 N=トレス領域に格納できるエントリ数

パラメータ trc にはトレス情報アドレスを格納する領域のポインタを指定します。サービスコール発行後、トレス情報アドレスが格納されます。

取得したトレス情報の内容が、オール 1 のとき、トレス領域に格納できるエントリ数分のトレス情報が取得されていない状態を指します。

本サービスコールは Smalight のプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

本サービスコールを使用する場合、OS再構築(リビルド)が必要です。詳細は「6.8 トレスを有効にする」を参照ください。

4.14 スタック操作

4.14.1 GET_REG

ユーザタスクのスタックからのレジスタ参照

【タスク部】【割込み部】

C言語インターフェース

```
H val = GET_REG ( UB tid, W offset );
```

パラメータ

UB	tid	タスクID
W	offset	スタックオフセット

リターンパラメータ

H	val	レジスタの値
---	-----	--------

解説

Ready, Waiting状態タスクのスタックに格納されているレジスタの値を参照します。Running状態のタスクに対しては使用できません。本サービスコールはマクロです。

パラメータ tid で参照したいスタックのタスクIDを指定します。tidに0(アイドル)や、存在しないタスクのタスクIDを指定しないでください。

パラメータ offset で参照したいレジスタのオフセットを指定します。オフセットは2の倍数で指定してください。オフセットの詳細は「付録D. スタック仕様」を参照ください。

なお、PC、DE、BC以外のレジスタの操作は推奨しません。使用例については「付録G.応用例」を参照ください。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

4.14.2 SET_REG

ユーザタスクのスタックへのレジスタ設定

【タスク部】【割込み部】

C言語インターフェース

```
void SET_REG ( UB tid, W offset, H value);
```

パラメータ

UB	tid	タスクID
W	offset	スタックオフセット
H	value	設定値

リターンパラメータ

無し

解説

Ready, Waiting状態タスクのスタックに格納されているレジスタの値を変更します。Running状態のタスクに対しては使用できません。本サービスコールはマクロです。

パラメータ tid で参照したいスタックのタスクIDを指定します。tidに0(アイドル)や、存在しないタスクのタスクIDを指定しないでください。

パラメータ offset で設定したいレジスタのオフセットを指定します。オフセットは2の倍数で指定してください。オフセットの詳細は「付録D. スタック仕様」を参照ください。

パラメータ value にて設定値を指定します。

なお、PC、DE、BC以外のレジスタの操作は推奨しません。使用例については「付録G.応用例」を参照ください。

本サービスコールはSmalightのプログラムサイズを抑える工夫のひとつとしてパラメータのエラーチェックを行っておりません。不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。パラメータには使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

5. アプリケーションプログラムの作成

5.1 作成の手順

以下にアプリケーション作成フローを示します。

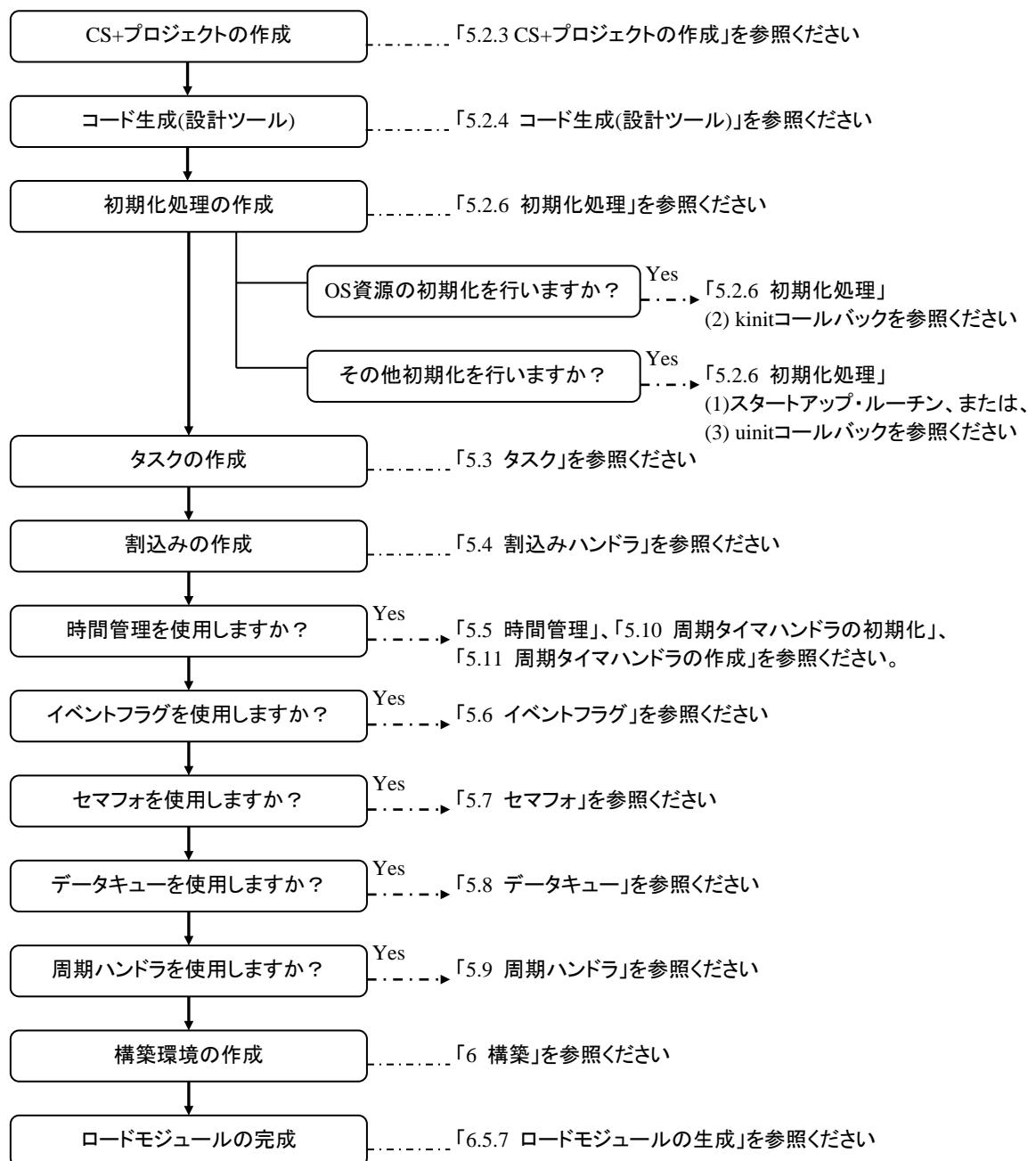


図5-1 アプリケーション作成フロー

5.2 システムの起動処理

5.2.1 システム起動時の処理フロー

システム起動時のフローを以下に示します。

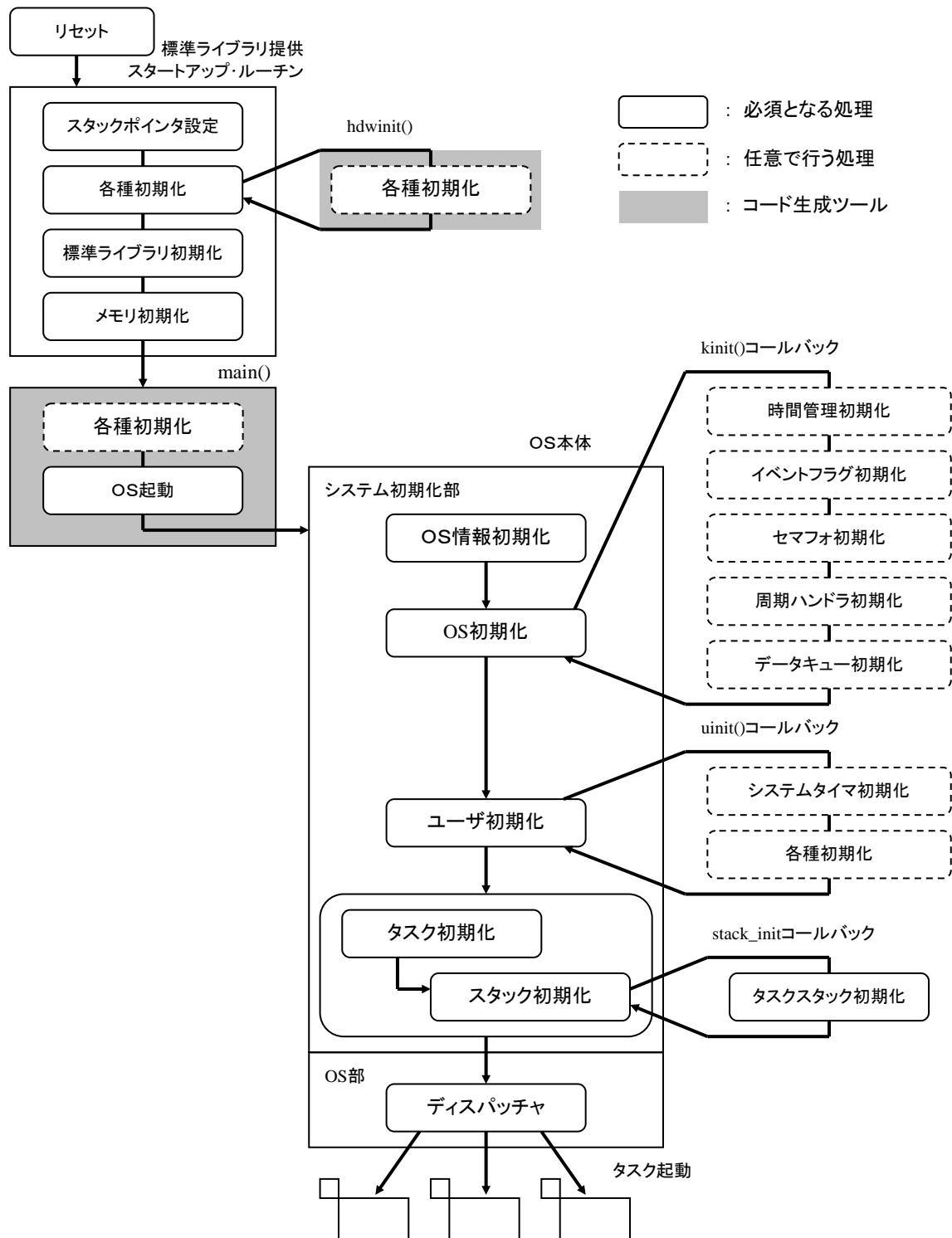


図5-2 システム起動時のフロー

5.2.2 Smalight OS 提供の CS+プロジェクト

Smalight OS インストールすると CS+プロジェクトがファイルコピーされます。ファイル・ディレクトリ構成の詳細は「6.1 ファイル・ディレクトリ構成と操作」を参照ください。

CS+プロジェクトの詳細を以下に示します。本 CS+プロジェクトを流用し、新しい CS+プロジェクトを作成してください。詳細は次章以降を参照ください。

表5-1 CS+プロジェクト初期設定

マイクロコントローラ	RL78/G13 (R5F100LE) ROM サイズ： 64K バイト RAM サイズ： 4096 バイト
コード生成(設計ツール)	クロック発生回路：端子割り当て設定は確定済み ^{(*)1} その他設定はデフォルト値

^{(*)1} 確定済み設定の詳細は、CS+プロジェクトを参照ください。また、ビルド前にコード生成の実行が必要となります。

5.2.3 CS+プロジェクトの作成

CS+プロジェクトの作成手順を以下に示します。

- ① CS+を起動します。
- ② [ファイル]-[新規作成]-[新しいプロジェクトを作成]メニューを選択すると[プロジェクト作成]ダイアログが表示されます。
- ③ [プロジェクト作成]ダイアログにて、以下の設定を行います。

表5-2 CS+プロジェクト作成ダイアログ設定

使用するマイクロコントローラ	任意(ツリービューより選択)
プロジェクトの種類	アプリケーション(CC-RL)
プロジェクト名	任意
作成場所	任意
プロジェクト名のフォルダを作成する	任意
既存プロジェクトのファイル構成を流用する。	チェックする
流用元のプロジェクト	Smalight OS 提供プロジェクト(u-ap.mtpj)を指定
プロジェクト・フォルダ以下の構成ファイルを コピーして流用する。	チェックする

④ 新規作成した CS+プロジェクトのディレクトリ構成は以下の様になります。

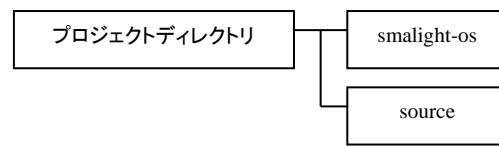


図5-3 CS+プロジェクト作成後のディレクトリ構成

⑤ 新規作成した CS+プロジェクトのディレクトリに、Smalight OS をコピーします。

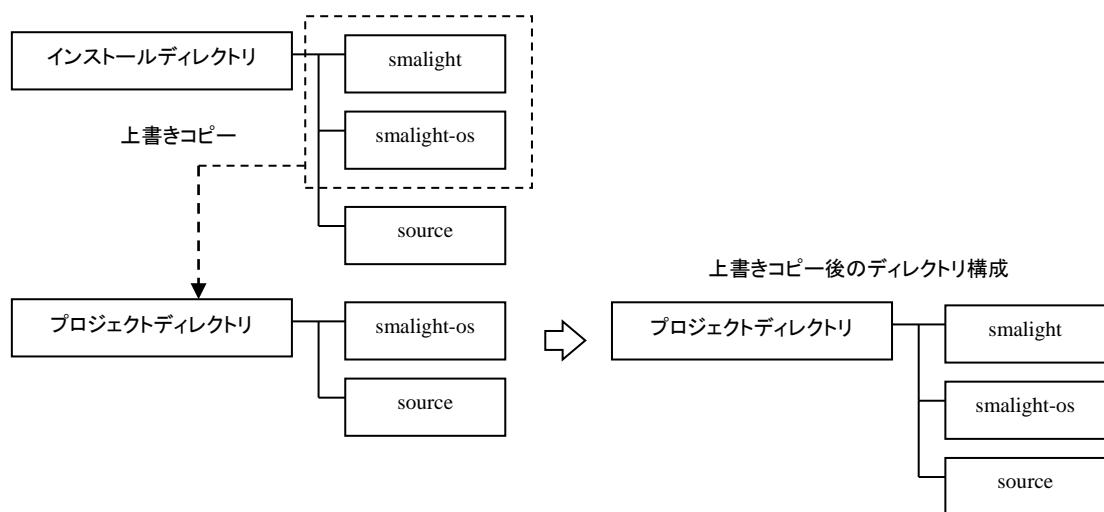


図5-4 Smalight OSのコピー

5.2.4 コード生成(設計ツール)

コード生成(設計ツール)とは、CS+が提供するGUIベースで各種情報を設定することでCPUが提供している周辺機能(クロック発生回路の機能、ポートの機能など)を制御するうえで必要なソース・コードを出力する機能です。

Smalight OSを利用したシステム開発においても、CPUの周辺機能に関するソース・コードはコード生成(設計ツール)の利用を推奨します。詳細は使用するコード生成(設計ツール)のマニュアルを参照ください。

但し、コード生成されたコードがRTOS用に組込むのに適していない場合がございます。必要に応じて、修正してご利用ください。

コード生成(設計ツール)により、main()関数等を含むソースコードを、ビルド前に必ず出力する必要があります。出力される主要ソースコードについて説明します。

(1) hwinit()関数

hwinit()関数は標準ライブラリが提供するスタートアップ・ルーチンより呼び出されます。

【解説】

- ① hwinit()関数はコード生成(設計ツール)で自動生成されます。
- ② 割込み禁止状態でSmalight OSを起動する必要があります。割込み許可するEI()関数をコメントアウトしてください。

リスト5-1 hwinit関数(sample\YR5F100LE\main\Yr_systeminit.c)

```
/*
**-----
** Abstract:
**     This function initializes hardware setting.
**     :
**-----
*/
void hwinit(void)                                ... ①
{
    DI();
    R_Systeminit();      EI()関数をコメントアウト
    //EI();               ... ②
}

/* Start user code for adding. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
```

(2) main()関数

main()関数は標準ライブラリが提供するスタートアップ・ルーチン実行後、呼び出されます。

【解説】

- ① コード生成(設計ツール)で生成されたインクルードファイルが、自動的にインクルード宣言されます。
- ② main()関数で Smalight OS を起動する場合は必ず slos.h をインクルードしてください。
- ③ main()関数はコード生成(設計ツール)で自動生成されます。
- ④ 割込み禁止状態で Smalight OS を起動する必要があります。必要に応じて、割込み禁止する DI()関数を発行してください。
- ⑤ 各種初期化を行います。リストは、タイマ TMA0 の開始例です(サンプルには記述されません)。
- ⑥ OS を起動します。

リスト5-2 main関数(sample\RF5F100LE\main\rf_main.c)

```
*****
** Include files
*****
*/
#include "xxxxxxxxxx.h"                                ... ①
/* Start user code for include. Do not edit comment generated here */
#include "slos.h"    追加                                ... ②
/* End user code. Do not edit comment generated here */

/*
-----
*/
** Abstract:
**      This function implements main function.
      :
**
-----
*/
void main(void)                                         ... ③
{
    /* Start user code. Do not edit comment generated here */
    DI();          割込み禁止状態で OS 起動      ... ④
    R_TAU0_Channel0_Start();    各種初期化(タイマ TAU0 の開始例)  ... ⑤

    slos_init();    OS 起動      ... ⑥
    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}

/* Start user code for adding. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
```

(3) 割込み

コード生成(設計ツール)の設定に応じて、割込みハンドラが自動生成されます。自動生成されたコードを元に Smalight OS 用の割込みを作成します。割込み実装の詳細は「5.4 割込みハンドラ」を参照ください。

① 割込み優先順位について

コード生成(設計ツール)にて割込みを設定する場合、優先順位を選択します。優先順位と Smalight OS の割込みレベルは以下の対応となります。

表5-3 Smalight OSの割込みレベル

コード生成(設計ツール) 割込みの優先順位	割込み実行中の ISP[1:0]	Smalight OS 管理 マスクレベル	備考
高	ISP[1:0]=- -	NMI ^(*)1)	
レベル1	ISP[1:0]=00	0	
レベル2	ISP[1:0]=01	1	
低	ISP[1:0]=10	2	

(*)1) PSW の ISP[1:0]ビットでは割込みマスクできない割込み優先度となります。PSW の IE ビットでのみ割込みマスクできます。

カーネルマスクレベルより高いマスクレベルの割込みは disp 無割込み、カーネルマスクレベル以下のマスクレベルの割込みは disp 有割込みで実装してください。カーネルマスクレベルの詳細は「6.4.1 カーネルマスクレベルの設定」を参照ください。

5.2.5 CC-RL(ビルド・ツール)

CC-RL(ビルド・ツール)のプロパティから、各種コンパイルオプションを設定します。設定の詳細は「6.5 CS+」を参照ください。

5.2.6 初期化処理

(1) スタートアップ・ルーチン

スタートアップ・ルーチンはコンパイラ提供の標準ライブラリを利用します。詳細は使用するコンパイラのマニュアルを参照ください。

(2) kinit コールバック

kinit コールバック(OS 初期化処理)では以下の処理を行ってください。

【解説】

- ① kinit コールバックを記述する場合は必ず slos.h をインクルードしてください。
- ② 時間管理の初期化
 - 時間管理を使用する場合には初期化を行ってください。
 - 時間管理の詳細は「5.5 時間管理」を参照ください。
- ③ イベントフラグの初期化
 - イベントフラグを使用する場合には初期化を行ってください。
 - イベントフラグの詳細は「5.6 イベントフラグ」を参照ください。
- ④ セマフォの初期化
 - セマフォを使用する場合には初期化を行ってください。
 - セマフォの詳細は「5.7 セマフォ」を参照ください。
- ⑤ 周期ハンドラの初期化
 - 周期ハンドラを使用する場合には初期化を行ってください。
 - 周期ハンドラの初期化の詳細は「5.9 周期ハンドラ」を参照ください。
- ⑥ データキューの初期化
 - 時間管理を使用する場合には初期化を行ってください。
 - データキューの初期化の詳細は「5.8 データキュー」を参照ください。

リスト5-3 kinitコールバック(kinit.c)

```
#include "slos.h" ... ①

:

void kinit(void)
{
    /* initialize */

    /* systime initialize */ ... ②
    /* systim_init(); */

    /* event flag initialize */ ... ③
    /* evtflg_init(); */
    /* EVTFLG_ATTR((UB)1u, (EVFLG_TA_AND | EVFLG_TA_CLR | EVFLG_TA_TPRI)); */

    /* semaphore initialize */ ... ④
    /* sem_init(); */
    /* SEM_ATTR((UB)1u, 1, SEM_TA_TPRI); */

    /* cyclic handler initialize */ ... ⑤
    /* cyc_init(); */
    /* CYC_ATTR((UB)1u, (CYC_TA_STA), cychdr1, 1000L); */
    /* CYC_ATTR((UB)2u, (CYC_TA_NON), cychdr2, 5000L); */

    /* data-que initialize */ ... ⑥
    /* dtq_init(); */
    /* DTQ_ATTR((UB)1u, (UB)DTQ_TA_TFIFO, (B)DTQ1_SIZE, knl_dtq1); */
    /* DTQ_ATTR((UB)2u, (UB)DTQ_TA_TPRI, (B)DTQ2_SIZE, knl_dtq2); */
}
```

(3) uinit コールバック

uinit コールバック(ユーザ初期化処理)では以下の処理を行ってください。

【解説】

① uinit コールバックを記述する場合は必ず slsos.h をインクルードしてください。

② その他初期化。

周期タイマハンドラの初期化、ポート、システム資源などのその他初期化を行ってください
(ユーザ任意)。 main()関数で初期化処理を記載済みの場合、修正不要です。

リスト5-4 uinitコールバック(user.c)

```
#include "slsos.h"                                ... ①

:

void uinit(void)
{
    /* initialize */                                ... ②
}
```

(4) stack_init コールバック

OS 初期化処理で呼び出されるコールバックです。stack_init コールバック(ユーザ初期化処理)では各タスクの開始アドレス等の情報を各タスクのスタックに格納する処理を行います。登録したタスク数分だけ発行されます。

5.3 タスク

タスクは C 言語で記述します。タスクの記述方法を以下に示します。

※ タスク関数は原則、永久ループで作成し、タスク関数を終了してはいけません。

【解説】

- ① タスクから OS サービスコールを発行する際は、必ず slos.h をインクルードしてください。
- ② 通常の関数同様、タスクの処理を記述します。
- ③ タスク関数は原則、永久ループで作成してください。
- ④ タスク関数は関数を抜けることができません。タスク関数の終了時には、slp_tsk サービスコールを発行して Waiting 状態にします。タスク関数の終了時の Waiting 状態を解除しないでください。タスク関数を抜けた場合の動作は保証されません。

リスト5-5 タスク記述方法

```
#include "slos.h"                                ... ①

:
void tsk01(void)                                ... ②
{
    /* タスクの処理 */
    while(1) {                                     ... ③
        ...
    }
    slp_tsk();                                     ... ④
}
```

5.4 割込みハンドラ

5.4.1 disp 無割込みハンドラ

disp 無割込みハンドラのフローを以下に示します。

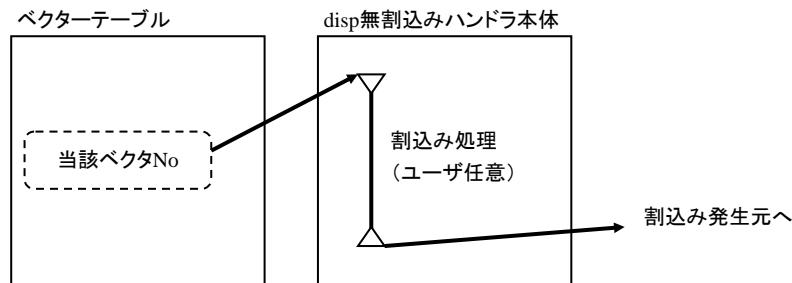


図5-5 disp 無割込みハンドラのフロー

(1) disp 無割込みハンドラ

disp 無割込みハンドラは C 言語で記述します。割込み処理はコード生成(設計ツール)により自動生成してください。詳細は使用するコード生成(設計ツール)のマニュアルを参照ください。

【解説】

- ① コード生成(設計ツール)で生成されたインクルードファイルが、自動的にインクルード宣言されます。必要に応じて、インクルードファイルを追加してください。
- ② 割込み関数が `#pragma interrupt` で宣言されます。必要に応じてレジスタバンク指定などの割込み仕様の設定を追加してください。
- ③ 割込みハンドラの処理を記述します。disp 無割込みハンドラでは、OS サービスコールを発行してはいけません。なお、disp 無割込みでは発生元のスタックを使用します。

リスト5-6 disp無割込みハンドラ(sample¥R5F100LE¥non-disp¥r_cg_timer_user.c)

```
*****  
Includes  
*****  
#include "r_cg_middleware.h" ... ①  
#include "r_cg_timer.h"  
/* Start user code for include. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */  
#include "r_cg_userdefine.h"  
*****  
Pragma directive  
*****  
#pragma interrupt r_tau0_channel0_interrupt(vect=INTTMO0) ... ②  
/* Start user code for pragma. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */  
*****  
Global variables and functions  
*****  
/* Start user code for global. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */  
*****  
* Function Name: r_tau0_channel0_interrupt  
* Description : This function is INTTMO0 interrupt service routine.  
* Arguments   : None  
* Return Value : None  
*****  
static void __near r_tau0_channel0_interrupt(void)  
{  
    /* Start user code. Do not edit comment generated here */ ... ③  
    /* End user code. Do not edit comment generated here */  
}  
/* Start user code for adding. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */
```

5.4.2 disp 有割込みハンドラ

disp 有割込みハンドラのフローを以下に示します。図中の点線(-----)処理は、多重割込みの時、または、割込み発生元タスクの割込みマスクレベルが 3 以外(PSW の ISP[1:0]ビット 3 以外)の時に実行されます。その場合、disp サービスコール発行によりディスパッチャへ制御が遷移せずに、割込み発生元に戻ります。

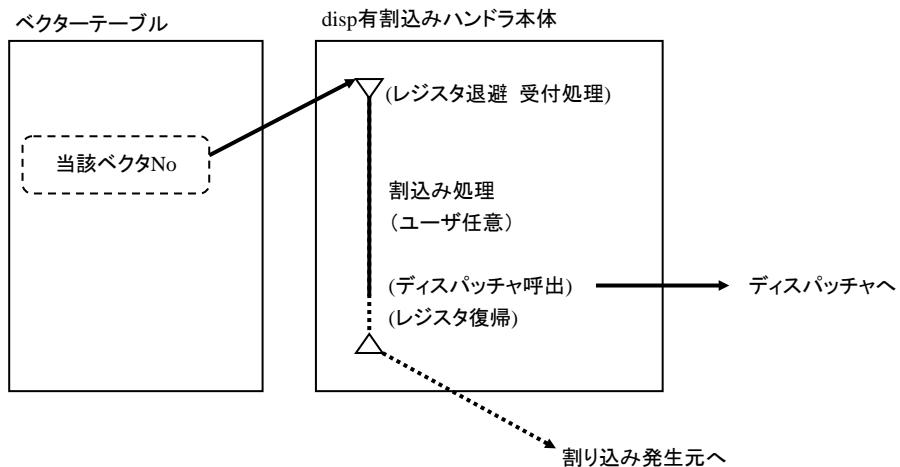


図5-6 disp有割込みハンドラのフロー

(1) disp 有割込みハンドラ

disp 有割込みハンドラは C 言語で記述します。割込み処理はコード生成(設計ツール)により自動生成してください。詳細は使用するコード生成(設計ツール)のマニュアルを参照ください。

コード生成(設計ツール)で生成した割込み関数を disp 有割込みハンドラに変更します。

【解説】

- ① disp 有割込みハンドラを実装する際は、必ず slos.h をインクルードしてください。
- ② #pragma interrupt で宣言される割込み関数を、#pragma rtos_interrupt 宣言に変更します。必要に応じてレジスタバンク指定などの割込み仕様の設定を追加してください。
- ③ 割込みハンドラの処理を記述します。

リスト5-7 disp有割込みハンドラ(sample\RS5F100LE\disp\r_cg_timer_user.c)

```
*****  
Includes  
*****  
#include "r_cg_macrodriver.h"  
#include "r_cg_timer.h"  
/* Start user code for include. Do not edit comment generated here */  
#include "slos.h" 追加 ... ①  
/* End user code. Do not edit comment generated here */  
#include "r_cg_userdefine.h"  
*****  
Pragma directive  
*****  
変更  
#pragma rtos_interrupt r_tau0_channel0_interrupt(vect=INTTM00) ... ②  
/* Start user code for pragma. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */  
*****  
Global variables and functions  
*****  
/* Start user code for global. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */  
*****  
* Function Name: r_tau0_channel0_interrupt  
* Description : This function is INTTM00 interrupt service routine.  
* Arguments   : None  
* Return Value : None  
*****  
static void __near r_tau0_channel0_interrupt(void)  
{  
    /* Start user code. Do not edit comment generated here */ ... ③  
    /* End user code. Do not edit comment generated here */  
}  
/* Start user code for adding. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */
```

5.4.3 ベクターテーブル

ベクターテーブルは、#pragma interrupt 宣言、または、#pragma rtos_interrupt 宣言することで、自動的に生成されます(ソースコード生成はされません)。

5.4.4 割込みスタックについて

(1) disp 無割込みの割込みstack

disp 無割込みの割込みスタックは割込み発生元のスタックが使用されます。割込み発生元となるタスクスタック、割込み優先度が低い割込みのスタックには、上位レベルにあたる disp 無割込みで使用するスタックを考慮したスタックサイズを設定してください。割込みスタックの計算方法については「6.6.2 割込みスタック」を参照ください。

(2) disp 有割込みの割込みスタック

同一レベルの割込みは共通のスタックが使用されます。同一レベルの割込みの中で、一番スタックを使用する割込みスタックサイズを割込みスタックとして確保してください。

割込みスタックの計算方法については「6.6.2 割込みスタック」を参照ください。

【解説】

- ① disp 有割込みスタックを実装する際は、必ず slos.inc をインクルードしてください。
 - ② disp 有割込みスタックのシンボルを外部定義宣言します。実装されない割込みレベルについてもシンボルは削除しないでください。
 - ③ 割込みレベル毎にスタックサイズを設定してください。実装されない割込みレベルについてもシンボルは削除しないでください。

リスト5-8 disp有割込みスタック(stack.asm)

```
$ INCLUDE ( slos. inc ) ... ①

; EXTERN
; -----
; PUBLIC
; -----
; . PUBLIC _intstackLV0 ... ②
; . PUBLIC _intstackLV1
; . PUBLIC _intstackLV2

; :
; -----
; Interrupt stack
; -----
intB_n .DSEG UNIT ... ③
    .DS    80H          ; for LV0 interrupt
_intstackLV0:
    .DS    80H          ; for LV1 interrupt
_intstackLV1:
    .DS    80H          ; for LV2 interrupt
_intstackLV2:
```

5.5 時間管理

5.5.1 時間管理の対象サービスコール

対象サービスコールは以下の通りです。

表5-4 時間管理関連サービスコール一覧

区分	サービスコール名称	説明
タスク管理関連	tslp_tsk	タスクの起床待ち(タイムアウト付き)
イベントフラグ	twai_flg ^(*)1)	イベントフラグ待ち(タイムアウト付き)
セマフォ	twai_sem ^(*)2)	セマフォの獲得(タイムアウト付き)
周期ハンドラ	sta_cyc ^(*)3)	周期ハンドラの開始
	stp_cyc ^(*)3)	周期ハンドラの停止
データキュー	trcv_dtq ^(*)4)	データキューからの受信(タイムアウト付き)
	tsnd_dtq ^(*)4)	データキューへの送信(タイムアウト付き)
時間管理	set_tim	システム時刻の設定
	get_tim	システム時刻の取得
	systim_init	時間管理の初期化
	slos_cyclic_timer	周期タイマ処理

(*)1 イベントフラグの設定が必要となります。

(*)2 セマフォの設定が必要となります。

(*)3 周期ハンドラの設定が必要となります。

(*)4 データキューの設定が必要となります。

5.5.2 時間管理の初期化

- (1) OS 定義ファイルの "#define SYSTIME" 行を有効にします(無効にする場合は、コメントアウトしてください)。

リスト5-9 時間管理の有効設定 (slos.def)

```
/*----- configuration define -----*/
:
#define SYSTIME
:
```

- (2) kinit コールバックに初期化関数 systim_init サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されています)。

リスト5-10 時間管理の初期化(kinit.c)

```
void kinit(void)
{
    /* initialize */

    /* systime initialize */
    systim_init();
    :
}
```

- (3) ユーザ任意の周期タイマハンドラにて、slos_cyclic_timer サービスコールを追加します。詳細は「5.10 周期タイマハンドラの初期化」を参照ください。

- (4) ユーザプログラムにて"slos.h"をインクルードすることで、サービスコールの発行が可能です。

5.6 イベントフラグ

5.6.1 イベントフラグの対象サービスコール

対象サービスコールは以下の通りです。

表5-5 イベントフラグ関連サービスコール一覧

区分	サービスコール名称	説明
イベントフラグ	wai_flg	イベントフラグ待ち
	twai_flg ^(*)1)	イベントフラグ待ち(タイムアウト付き)
	set_flg, iset_flg	イベントフラグの設定
	clr_flg	イベントフラグのクリア
	evtflg_init	イベントフラグの初期化
	EVTFLG_ATTR ^(*)2)	イベントフラグの属性設定

(^{*)1}) 時間管理機能の設定が必要となります。

(^{*)2}) マクロです。

5.6.2 イベントフラグの初期化

(1) OS 定義ファイルの ”#define EVENTFLG” 行を有効にします(無効にする場合は、コメントアウトしてください)。

リスト5-11 イベントフラグの有効設定 (slos.def)

```
/*----- configuration define -----*/
:
#define EVENTFLG
:
```

(2) kinit コールバックに初期化関数 evtflg_init サービスコール、EVTFLG_ATTR サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されています)。

リスト5-12 イベントフラグの初期化(kinit.c)

```
void kinit(void)
{
:
/* event flag initialize */
evtflg_init();
EVTFLG_ATTR((UB)1u, (EVFLG_TA_AND | EVFLG_TA_CLR | EVFLG_TA_TPRI));
:
}
```

(3) ユーザプログラムにて"slos.h"をインクルードすることによる、サービスコールの発行が可能です。

5.7 セマフォ

5.7.1 セマフォの対象サービスコール

対象サービスコールは以下の通りです。

表5-6 セマフォ関連サービスコール一覧

区分	サービスコール名称	説明
セマフォ	wai_sem	セマフォの獲得
	twai_sem ^(*)1)	セマフォの獲得(タイムアウト付き)
	sig_sem, isig_sem	セマフォの返却
	sem_init	セマフォの初期化
	SEM_ATTR ^(*)2)	セマフォの属性設定

(*)1 時間管理機能の設定が必要となります。

(*)2 マクロです。

5.7.2 セマフォの初期化

- (1) OS 定義ファイルの ”#define SEMAPHORE” 行を有効にします(無効にする場合は、コメントアウトしてください)。

リスト5-13 セマフォの有効設定 (slos.def)

```
/*----- configuration define -----*/
:
#define SEMAPHORE
:
```

- (2) kinit コールバックに初期化関数 sem_init サービスコール、SEM_ATTR サービスコールを追加します
(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されています)。

リスト5-14 セマフォの初期化(kinit.c)

```
void kinit(void)
{
:
/*
 * semaphore initialize */
sem_init();
SEM_ATTR((UB)1u, 1, SEM_TA_TPRI);
:
}
```

- (3) ユーザプログラムにて"slos.h"をインクルードすることによる、サービスコールの発行が可能です。

5.8 データキュー

5.8.1 データキューの対象サービスコール

対象サービスコールは以下の通りです。

表5-7 データキュー関連サービスコール一覧

区分	サービスコール名称	説明
データキュー	rcv_dtq	データキューからの受信
	trcv_dtq ^(*)1)	データキューからの受信(タイムアウト付き)
	snd_dtq, isnd_dtq	データキューへの送信
	tsnd_dtq ^(*)1)	データキューへの送信(タイムアウト付き)
	fsnd_dtq, ifsnd_dtq	データキューへの強制送信
	dtq_init	データキューの初期化
	DTQ_ATTR ^(*)2)	データキューの属性設定

^(*)1) 時間管理機能の設定が必要となります。

^(*)2) マクロです。

5.8.2 データキューの初期化

- (1) OS 定義ファイルの”#define DATAQUE”行を有効にします(無効にする場合は、コメントアウトしてください)。

リスト5-15 データキューの有効設定 (slos.def)

```
/*----- configuration define -----*/
:
#define DATAQUE
:
```

- (2) kinit コールバックに初期化関数 dtq_init サービスコール、DTQ_ATTR サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されています)。

リスト5-16 データキューの初期化(kinit.c)

```
/* ----- data que ----- */
#define DTQ1_SIZE 1
#define DTQ2_SIZE 2
W knl_dtq1[DTQ1_SIZE];
W knl_dtq2[DTQ2_SIZE];

void kinit(void)
{
:
/*
 * data-que initialize */
dtq_init();
DTQ_ATTR((UB)1u, (UB)DTQ_TA_TFIFO, (B)DTQ1_SIZE, knl_dtq1);
DTQ_ATTR((UB)2u, (UB)DTQ_TA_TPRI, (B)DTQ2_SIZE, knl_dtq2);
:
}
```

- (3)ユーザプログラムにて"slos.h"をインクルードすることによる、サービスコールの発行が可能です。

5.9 周期ハンドラ

5.9.1 周期ハンドラの対象サービスコール

対象サービスコールは以下の通りです。

表5-8 周期ハンドラ関連サービスコール一覧

区分	サービスコール名称	説明
周期ハンドラ	sta_cyc ^(*)1)	周期ハンドラの開始
	stp_cyc ^(*)1)	周期ハンドラの停止
	cyc_init ^(*)1)	周期ハンドラの初期化
	CYC_ATTR ^{(*)1)(*)2)}	周期ハンドラの属性設定
	CYC_CHG ^{(*)1)(*)2)}	周期ハンドラの起動周期変更
	callback_cychdr ^{(*)1)(*)3)}	周期ハンドラ本体

(*)1 時間管理機能の設定が必要となります。

(*)2 マクロです。

(*)3 コールバックルーチンです。

5.9.2 周期ハンドラの作成

周期ハンドラの記述方法を以下に示します。

【解説】

- ① 必ず slos.h をインクルードしてください。
- ② 周期ハンドラ本体の関数(cychdr1)は callback_cychdr コールバックです。関数名称はユーザ任意です。引数 cid は周期ハンドラ ID です。どの周期ハンドラ ID に割り当てられた周期ハンドラ関数かを判断することができます。
- ③ 周期ハンドラ処理を記述します。周期ハンドラ内では割込み部から発行可能なサービスコールを使用できます。

リスト5-17 周期ハンドラ記述方法

```
#include "slos.h"                                ... ①
:
void cychdr1(UB cid)                            ... ②
{
    /* 周期ハンドラの処理 */                      ... ③
}
```

5.9.3 周期ハンドラの初期化

- (1) OS 定義ファイルの”#define CYC_HDR”行を有効にします(無効にする場合は、コメントアウトしてください)。周期ハンドラを使用する場合、時間管理の初期化が必須となります(初期化の詳細は「5.5.2 時間管理の初期化」を参照ください)。

リスト5-18 周期ハンドラの有効設定 (slos.def)

```
/*----- configuration define -----*/
:
#define CYC_HDR
:
```

- (2) kinit コールバックに初期化関数 cyc_init サービスコール、CYC_ATTR サービスコールを追加します(kinit のサンプルにはコメントアウトした状態で本初期化処理が記載されております)。周期ハンドラを使用する場合、時間管理の初期化が必須となります(初期化の詳細は「5.5.2 時間管理の初期化」を参照ください)。

リスト5-19 周期ハンドラの初期化(kinit.c)

```
/* ----- cyclic handler ----- */
extern void cychdr1(UB cid);

void kinit(void)
{
    :
    /* cyclic handler initialize */
    cyc_init();
    CYC_ATTR((UB)1u, (CYC_TA_NON), cychdr1, 1000L);
    :
}
```

- (3)ユーザプログラムにて"slos.h"をインクルードすることによる、サービスコールの発行が可能です。

5.10 周期タイマハンドラの初期化

main()関数、または、uinit コールバックに周期タイマハンドラを利用するタイマモジュールを初期化します。使用するタイマモジュールはユーザ任意です。初期化にて一定周期のタイマ割込みを発生させてください。

作成した周期タイマハンドラの周期時間をコンフィギュレーションファイルにて設定する必要があります。詳細は「6.4.3 周期タイマハンドラ周期時間の設定」を参照ください。

5.11 周期タイマハンドラの作成

「5.10 周期タイマハンドラの初期化」にて初期化したタイマモジュールの周期割込みを、disp 有割込みハンドラとして登録します。disp 有割込みハンドラの詳細は「5.4.2 disp 有割込みハンドラ」を参照ください。作成した disp 有割込みハンドラにて、slos_cyclic_timer サービスコールを発行します。

リスト5-20 周期タイマハンドラの例

```
*****  
Includes  
*****  
#include "r_cg_macrodriver.h"  
#include "r_cg_timer.h"  
/* Start user code for include. Do not edit comment generated here */  
#include "slos.h"    追加  
/* End user code. Do not edit comment generated here */  
#include "r_cg_userdefine.h"  
*****  
Pragma directive  
*****  
变更  
#pragma rtos_interrupt r_tau0_channel0_interrupt(vect=INTTM00)  
/* Start user code for pragma. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */  
*****  
Global variables and functions  
*****  
/* Start user code for global. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */  
*****  
* Function Name: r_tau0_channel0_interrupt  
* Description : This function is INTTM00 interrupt service routine.  
* Arguments   : None  
* Return Value : None  
*****  
static void __near r_tau0_channel0_interrupt(void)  
{  
    /* Start user code. Do not edit comment generated here */  
    slos_cyclic_timer();    追加  
    /* End user code. Do not edit comment generated here */  
}  
/* Start user code for adding. Do not edit comment generated here */  
/* End user code. Do not edit comment generated here */
```

※ 詳細はサンプルプログラムを参照ください。

6. 構築

6.1 ファイル・ディレクトリ構成と操作

以下に Smalight OS インストール後の主要なファイル構成を示します。インストールパスはデフォルト("⟨システムドライブ⟩:\smalight")を想定しています。

表6-1 ファイル・ディレクトリ構成

ディレクトリ／ファイル	説明
⟨Smalight⟩	—
└ <os>	OS 格納
└ <RL78_vvvv⟩	RL78 用
└ <sample>	サンプルプログラム格納
└ <CC-RL_V105>	ビルド・ツール CC-RL V1.05 用
└ u-ap.mtpj	ユーザプログラム CS+プロジェクトファイル
└ <smalight>	プロジェクト情報格納
└ u-ap.mtpj	Smalight OS 構築用 CS+プロジェクトファイル
└ <DefaultBuild>	未使用
└ <lib_slos_xxxxx>	Smalight OS ライブライ生成・格納 (*.lib)
└ <obj_xxxxx>	ロードモジュール生成 (*.lmp, *.hex, *.map 等)
└ <smalight-os>	Smalight OS 格納
└ config.c	コンフィギュレーションファイル
└ idle.c	アイドル処理
└ kinit.c	OS 初期化処理
└ slos.def	OS 定義ファイル
└ slos.h	OS ヘッダ
└ slos.inc	OS ヘッダ<ASM>
└ stackini.c	タスクスタック初期化処理
└ sysdown.c	システムダウンルーチン
└ <sys>	Smalight OS 本体ソース
└ <source>	ユーザアプリケーション格納
└ stack.asm	スタック定義ファイル<ASM>
└ user.c	ユーザプログラム
└ <doc>	マニュアル格納

vvv : バージョン／リビジョン

xxxxx : RL78m=RL78(ミディアム・モデル)

RL78s=RL78(スマート・モデル)

6.2 構築手順

構築手順を以下に示します。

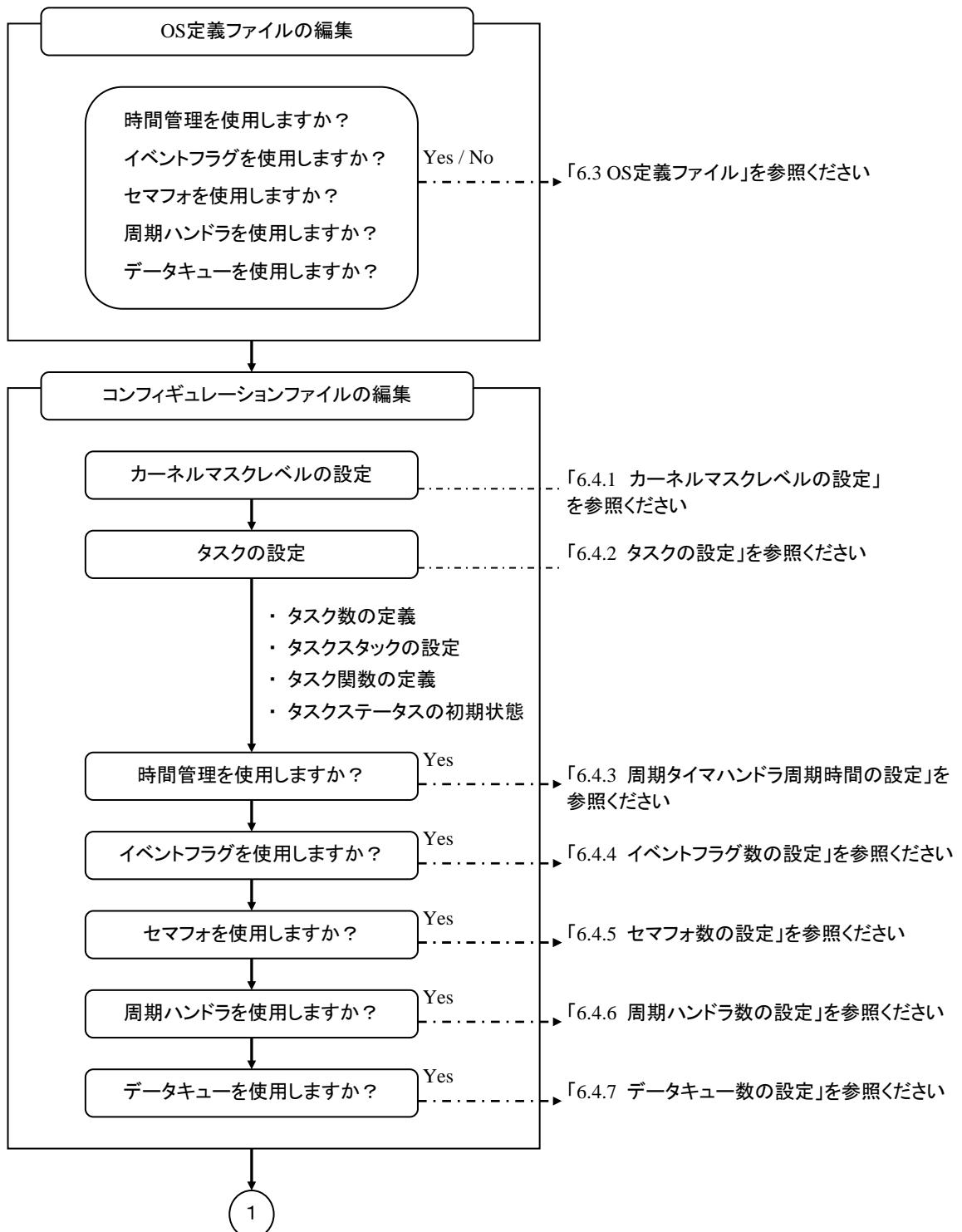


図6-1 構築手順①

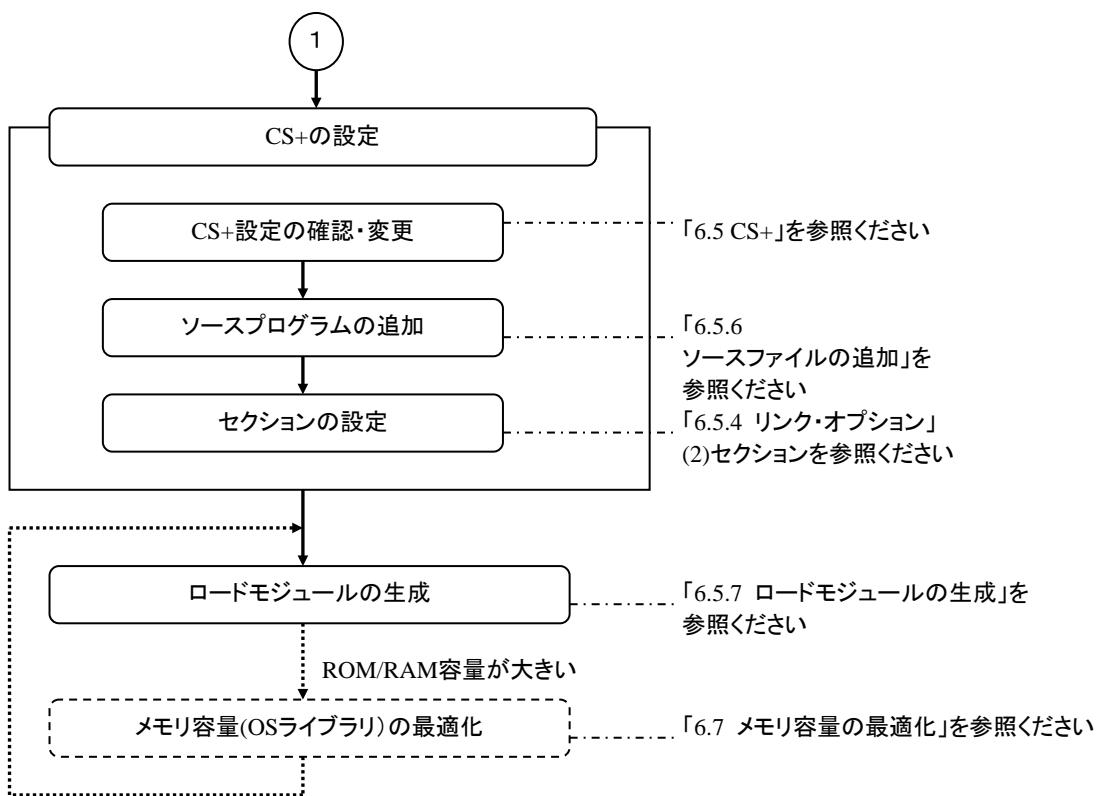


図6-2 構築手順②

6.3 OS 定義ファイル

OS 定義ファイルでは、各機能の有効、無効設定を行います。設定可能な機能は、イベントフラグ、セマフォ、時間管理、周期ハンドラ、データキューです。無効に設定した場合、その機能は一切使用できなくなります。

無効にした場合、コンフィギュレーションファイル(config.c)の設定内容は無効となります。また、OS 初期化処理(kinit.c)に無効にした機能の初期化処理を記載してはいけません。その場合、ビルドでリンクエラーが発生します。

【解説】

- ① #define EVENTFLG
有効にするとイベントフラグが有効となります。使用しない場合、コメントアウトしてください。
- ② #define SEMAPHORE
有効にするとセマフォが有効となります。使用しない場合、コメントアウトしてください。
- ③ #define SYSTIME
有効にすると時間管理が有効となります。使用しない場合、コメントアウトしてください。
- ④ #define CYC_HDR
有効にすると周期ハンドラが有効となります。使用しない場合、コメントアウトしてください。
- ⑤ #define DATAQUE
有効にするとデータキューが有効となります。使用しない場合、コメントアウトしてください。

リスト6-1 OS定義ファイルの設定(slos.def)

```
/*----- configuration define -----*/
//#define EVENTFLG ... ①
//#define SEMAPHORE ... ②
//#define SYSTIME ... ③
//#define CYC_HDR ... ④
//#define DATAQUE ... ⑤
```

6.4 コンフィギュレーションファイル

6.4.1 カーネルマスクレベルの設定

コンフィギュレーションファイルでカーネルマスクレベルを設定してください。

【解説】

① カーネルマスクレベル: KNL_MSK_LEVEL

カーネルマスクレベルを設定してください。設定したカーネルマスクレベルにより OS 動作中の割込みマスクが決定します。

リスト6-2 カーネルマスクレベルの設定(config.c)

```
/*----- Kernel Mask -----*/
#define KNL_MSK_LEVEL      0u      /* kernel mask level (2-0) */      ... ①
```

表6-2 カーネルマスクレベル設定①

カーネルマスク レベル	設定内容
0	OS動作中の割込みマスクを0に設定します(PSWレジスタのISP[1-0]ビットを00に設定)
1	OS動作中の割込みマスクを1に設定します(PSWレジスタのISP[1-0]ビットを01に設定)
2	OS動作中の割込みマスクを2に設定します(PSWレジスタのISP[1-0]ビットを10に設定)

6.4.2 タスクの設定

コンフィギュレーションファイルでタスクの設定をしてください。設定項目を以下に示します。

- ・タスク数の定義
- ・タスクのスタックサイズの定義
- ・タスクスタックの実体の定義
- ・タスクスタックの終端アドレスの定義
- ・タスク関数の定義
- ・タスクステータスの初期状態

【解説】

- ② タスク数:KNL_TCB_NUM
タスク数を設定してください。設定値は1～127まで指定できます。
- ③ プライオリティタスクの数:KNL_TCB_PRI_NUM
プライオリティタスクの数を設定してください。設定値は0～KNL_TCB_NUMです。
ローテーションタスクの数は、KNL_TCB_NUM - KNL_TCB_PRI_NUMで決定します。
- ④ タスク毎のスタックサイズ:TCBx_SIZE
タスク毎のスタックサイズを設定してください。タスク数用意してください。タスクのスタックサイズ算出方法は「6.6.1 タスクスタック」を参照ください。
- ⑤ タスク毎のスタック領域:tcb_stackx
タスク毎のスタック領域の実態が用意されます。タスク数用意してください。
- ⑥ タスク毎のスタックポインタ初期値:TCBspInit
タスク毎のスタックポインタ初期値を設定します。タスク数用意してください。

リスト6-3 タスクの設定①(config.c)

```
/*----- TCB Number -----*/
#define KNL_TCB_NUM      3                                ... ②
#ifndef KNL_BB_PRIORITY
#define KNL_TCB_PRI_NUM  1                                ... ③
#endif
/*----- TCB Stack Size -----*/
#define TCB1_SIZE        0x120u                            ... ④
#define TCB2_SIZE        0x120u
#define TCB3_SIZE        0x120u

W tcb_stack1[(TCB1_SIZE/(UH)sizeof(W))];                ... ⑤
W tcb_stack2[(TCB2_SIZE/(UH)sizeof(W))];
W tcb_stack3[(TCB3_SIZE/(UH)sizeof(W))];

/*----- TCB Stack addr -----*/
const TCBSP TCBspInit[KNL_TCB_NUM] = {
{ &tcb_stack1[TCB1_SIZE/(UH)sizeof(W)],    /* TCB1 */
{ &tcb_stack2[TCB2_SIZE/(UH)sizeof(W)],    /* TCB2 */
{ &tcb_stack3[TCB3_SIZE/(UH)sizeof(W)] }     /* TCB3 */
};
```

- ⑦ タスク毎の開始アドレス:TCBaddrInit
タスク毎の開始アドレスを設定します。関数の場合、外部参照の定義が必要です。タスク数用意してください。
- ⑧ タスク毎の初期タスク状態:TCBstInit
タスク毎の初期タスク状態を設定します。設定できる初期状態は、Ready 状態(CTCB_ST_RDY)、及び、Waiting 状態(起床待ち:CTCB_ST_SLP)のみです。タスク数用意してください。Waiting 状態で初期登録したタスクは wup_tsk、または iwup_tsk サービスコールにより起床させます。

リスト6-4 タスクの設定②(config.c)

```

/*----- TCB Start addr Init -----*/
/** Refer to exterior ***/
    extern void tsk01(void);           /* TCB1 */
    extern void tsk02(void);           /* TCB2 */
    extern void tsk03(void);           /* TCB3 */
/** Table for start addr init **/
```

```

const TCBADDR TCBaddrInit[KNL_TCB_NUM] = {
    { tsk01 },                      ... ⑦
    { tsk02 },
    { tsk03 }
};

/*----- TCB Status Init -----*/
/*
 * Task status Init : CTCB_ST_RDY | CTCB_ST_SLP | ...
 *                   ("slos.h" Refer to for details.)
 */
const UB TCBstInit[KNL_TCB_NUM] = {
    CTCB_ST_RDY,                   /* TCB1 */
    CTCB_ST_RDY,                   /* TCB2 */
    CTCB_ST_RDY                   /* TCB3 */
};

```

6.4.3 周期タイマハンドラ周期時間の設定

コンフィギュレーションファイルで周期タイマハンドラ周期時間を設定してください。

【解説】

- ⑨ 周期タイマハンドラ周期時間:SYSTIM_CYCLIC_TIM
単位は msec です。周期タイマハンドラはユーザ任意作成する必要があります。詳細は「5.10 周期タイマハンドラの初期化」「5.11 周期タイマハンドラの作成」を参照ください。

リスト6-5 周期タイマハンドラ周期時間の設定例(config.c)

```
/*----- SYSTEM TIME -----*/
#ifndef SYSTIME
#define SYSTIM_CYCLIC_TIM  250uL      /* systim time(msec) */      ... ⑨
#endif
```

6.4.4 イベントフラグ数の設定

コンフィギュレーションファイルでイベントフラグ数を設定してください。

【解説】

- ⑩ イベントフラグの数:EVFLG_NUM
イベントフラグ数を設定ください。設定値は 1~127 まで指定できます。イベントフラグ ID は 1,2,...n までとなります。

リスト6-6 イベントフラグ数の設定(config.c)

```
/*----- EVENTFLG -----*/
#ifndef EVENTFLG
#define EVFLG_NUM    1      /* FLG Number */      ... ⑩
#endif
```

6.4.5 セマフォ数の設定

コンフィギュレーションファイルでセマフォ数を設定してください。

【解説】

⑪ セマフォの数:SEM_NUM

セマフォ数を設定ください。設定値は1～127まで指定できます。セマフォIDは1,2,...nまでとなります。

リスト6-7 セマフォ数の設定(config.c)

```
/*----- SEMAPHORE -----*/
#ifndef SEMAPHORE
#define SEM_NUM      1           /* Semaphore Number */
#endif
```

6.4.6 周期ハンドラ数の設定

コンフィギュレーションファイルで周期ハンドラ数を設定してください。

【解説】

⑫ 周期ハンドラの数:CYC_NUM

周期ハンドラ数を設定ください。設定値は1～127まで指定できます。周期ハンドラIDは1,2,...nまでとなります。

リスト6-8 周期ハンドラ数の設定(config.c)

```
/*----- CYCLIC HANDLER -----*/
#ifndef CYC_HDR
#define CYC_NUM      1           /* Cyclic Handler */
#endif
```

6.4.7 データキュー数の設定

コンフィギュレーションファイルでデータキュー数を設定してください。

【解説】

- ⑬ データキューの数:DTQ_NUM
データキュー数を設定ください。設定値は1～127まで指定できます。データキューIDは1,2,...nまでとなります。

リスト6-9 データキュー数の設定(config.c)

```
/*----- DATAQUE -----*/
#ifndef DATAQUE
#define DTQ_NUM      1           /* DTQ Number */      ...
#endif
```

6.5 CS+

Smalight OS は CS+ プロジェクトで提供されます。プロジェクトの詳細は「表 2-4 CS+ プロジェクト、ビルド・モード一覧」を参照ください。

以下、プロジェクトに含まれる "u-ap" プロジェクトの CC-RL(ビルド・ツール)のプロパティについて説明します。また、説明されない詳細な設定については、付属プロジェクトの設定をご確認ください。操作方法の詳細は使用する CS+ のマニュアルを参照ください。

6.5.1 共通オプション

以下に示すビルド・モードが準備されます。使用する CPU に応じて選択してください。操作方法の詳細は使用する CS+ のマニュアルを参照ください。

表6-3 共通オプション ビルド・モード設定

ビルド・モード	説明
obj_RL78m	ミディアム・モデル用ビルド・モード
obj_RL78s	スマート・モデル用ビルド・モード

6.5.2 コンパイル・オプション

以下に示す追加のインクルード・パス設定が必要です(デフォルトで設定済みです)。操作方法の詳細は使用する CS+ のマニュアルを参照ください。

表6-4 コンパイル・オプション 追加のインクルード・パス設定

追加のインクルード・パス	説明
smalight-os	Smalight OS のヘッダファイル格納

以下に示す定義マクロがデフォルトで設定済みです。

表6-5 コンパイル・オプション 定義マクロ設定

定義マクロ	Value	説明
SLOS_VERSION	vvrr	Smalight OS のバージョン・リビジョンを定義します。

6.5.3 アセンブル・オプション

以下に示す追加のインクルード・パス設定が必要です(デフォルトで設定済みです)。操作方法の詳細は使用する CS+ のマニュアルを参照ください。

表6-6 アセンブル・オプション追加のインクルード・パス設定

追加のインクルード・パス	説明
smalight-os	Smalight OS のヘッダファイル格納

6.5.4 リンク・オプション

以下に示す追加の入力ライブラリの設定が必要です(デフォルトで設定済みです)。操作方法の詳細は使用するCS+のマニュアルを参照ください。

表6-7 リンク・オプション ライブラリ設定

追加のライブラリ・パス	使用するライブラリ・ファイル	説明
<u>smalight-os</u> ¥lib_slos_xxxx	smalight-os.lib	Smalight OSのライブラリ

xxxxx : RL78m=RL78用(ミディアム・モデル)

RL78s=RL78用(スマート・モデル)

6.5.5 リンク・ディレクティブの設定

リンク・ディレクティブ・ファイルにより、セクションの配置を指定することができます。リンク・ディレクティブ・ファイルはコード生成(設計ツール)でサンプルコードを出力されます。必要に応じてセクションの配置を変更ください。詳細は使用するCS+のマニュアルを参照ください。

以下に、Smalight OSに関するセクション名一覧を示します。

表6-8 SmalightOSに関するセクション名

区分	セクション名	説明
ROM	smaP_n	コード部(near領域配置)
	smaP_f	コード部(far領域配置) *1
	smaC_n	ROMデータ(near領域配置)
	smaC_f	ROMデータ(far領域配置)
	smaR_n	near初期化データ(初期値あり)
	smaR_f	far初期化データ(初期値あり)
RAM	smaB_n	データ領域(初期値なし、near領域配置)
	smaB_f	データ領域(初期値なし、far領域配置) *2
	smaD_n	データ領域(初期値あり、near領域配置)
	smaD_f	データ領域(初期値あり、far領域配置) *2
	ossB_n	OSスタック領域(near領域配置)
	intsB_n	割込みスタック領域(near領域配置)

【注】記載のセクション名が存在しない場合がございます。

*1:ミディアムモードではsmaP_fセクションとなります。SmalightOSのROMセクションはアドレス0000H-FFFFH(64Kバイト空間)に配置する必要があるため、far領域には配置できません。

*2:内部RAMのみを使用する場合、通常はfar領域のセクションは使用しません。

6.5.6 ソースファイルの追加

作成したアプリケーションのソースファイル(Cソース、ASMソース)を、CS+に登録します。操作方法の詳細は使用するCS+のマニュアルを参照ください。

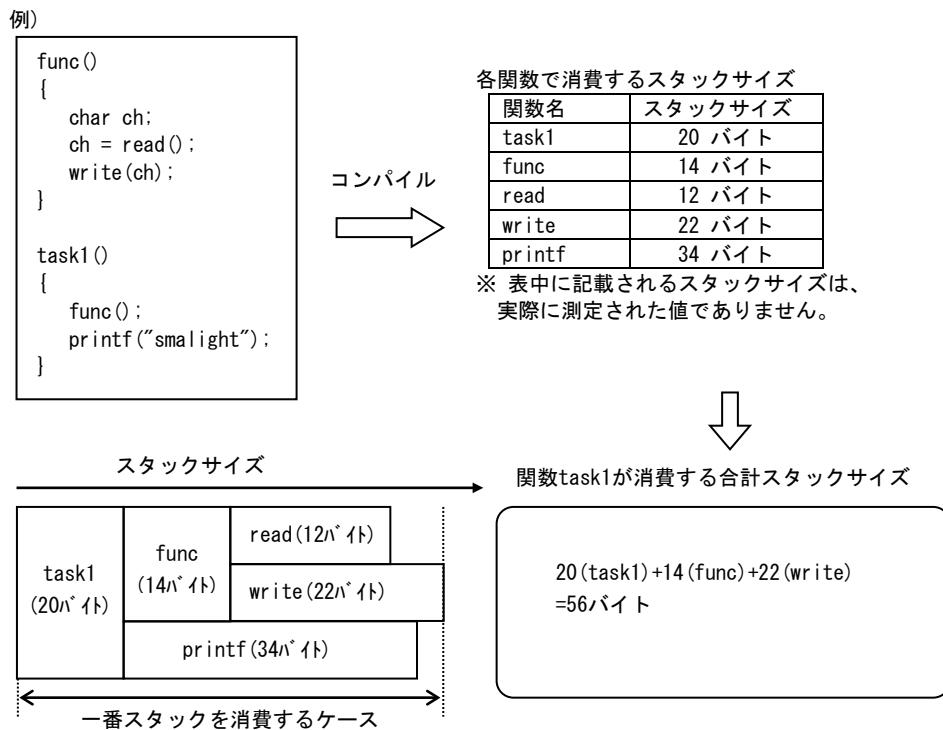
6.5.7 ロードモジュールの生成

CS+にてビルドしてください。操作方法の詳細は使用するCS+のマニュアルを参照ください。

6.6 スタック使用量の算出

6.6.1 タスクスタック

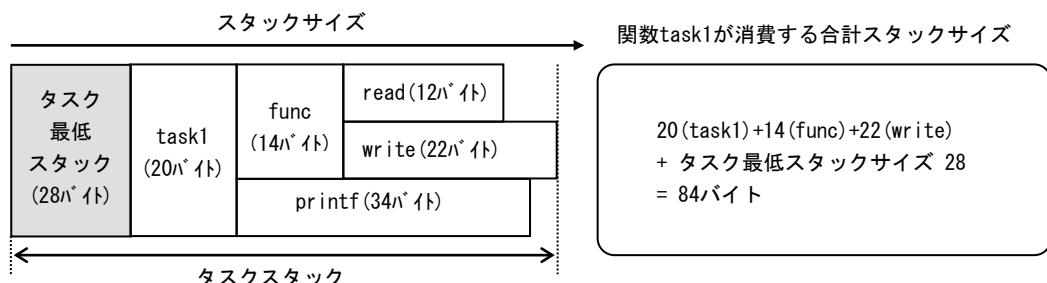
タスクのスタックサイズの計算方法について説明します。スタックサイズは各関数のスタックサイズと関数のネストが関係します。下記に関数のスタック計算方法を示します。



そこにOS使用時のタスク最低スタックサイズを加算したものが、タスクスタックとして必要なサイズとなります。タスク最低スタックサイズとは、タスク実行中の割込み受け付けに必要なスタックサイズ、及び、サービスコール発行に必要なスタックサイズを考慮したものです。

各関数で消費するスタックサイズは、コンパイラ付属のツール Stack Usage Tracer にて確認できます。

タスク最低スタックサイズは 28 バイト(サービスコールによるレジスタ退避分 14 バイト+disp 有割込みハンドラによるスタック使用 14 バイト)です。各関数で消費するスタックサイズは、コンパイラ付属のツール Call Walker にて確認できます。



6.6.2 割込みスタック

(1) disp 無割込みハンドラ

disp 無割込みハンドラの実装は#pragma interrupt を使用するため、割込み関数のスタックサイズを求めてください。各関数で消費するスタックサイズは、コンパイラ付属のツール Call Walker にて確認できます。

(2) disp 有割込みハンドラ

disp 有割込みハンドラのスタックについて説明します。disp 有割込みハンドラは#pragma rtos_interrupt を使用して実装します。下記は、disp 有割込み発生からディスパッチャへ制御を移すまでのスタック切り替えタイミングです。

- ① disp 有割込みハンドラでは、まず割込み受付処理(__kernel_int_entry)が実行されます。割込み発生元のタスクスタック(アイドル状態の時は OS スタック)にレジスタ退避後、割込みスタックに切り替えます。割込み受付処理で、割込みスタックを 2 バイト使用します。
- ② disp 有割込みハンドラは、割込みスタックを使用して処理されます。
- ③ disp 有割込みハンドラの出口で、割込み終了処理(__ret_int)が実行されます。ディスパッチャが実行される場合、OS スタックに切り替えられます。多重割込み中は割込み発生元へ戻ります。

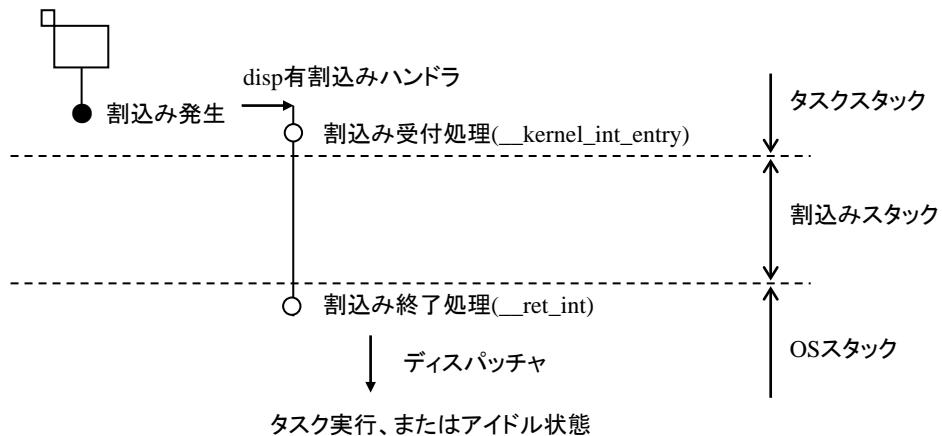


図6-5 disp有割込みハンドラのスタック切り替えタイミング

disp 有割込みスタックのスタックサイズは、disp 有割込みハンドラ(callback_int)で使用するスタックサイズ +2 バイトとなります。但し、disp 有割込みハンドラ処理中に上位レベルの割込みを受け付ける場合は、その分の割込みスタックを確保してください。

各関数で消費するスタックサイズは、コンパイラ付属のツール Stack Usage Tracer にて確認できます。

6.6.3 OS スタック

OS スタックは OS 動作中に使用されます。stack.asm にてデフォルトで設定される OS スタック初期値のままで問題ありません。

但し、kinit コールバックルーチン、uinit コールバックルーチン、stack_init コールバックルーチン、idle コールバックルーチンは、OS スタックで呼び出されます。その処理中に設定値以上のスタックを使用する場合は、コンフィギュレーションファイルの OS スタック設定値を変更する必要があります。

6.7 メモリ容量の最適化

6.7.1 OS ライブライ

デフォルトでは、以下のサービスコールが使用可能です。アプリケーションから呼び出されたサービスコールの機能のみが OS ライブライから選択されて、最適化したロードモジュールを生成します。アプリケーションで使用しない OS の機能は生成したロードモジュールには含まれないため、ROM/RAM 容量を節約することができます。

時間管理、イベントフラグ、セマフォ、周期ハンドラ、データキューに関するサービスコールは「5.5 時間管理」「5.6 イベントフラグ」「5.7 セマフォ」「5.9 周期ハンドラ」「5.8 データキュー」に従って設定することにより、各々の機能が使用できるようになります。

表6-9 デフォルトで使用可能なサービスコール

区分	サービスコール名称	説明
タスク管理関連	slp_tsk	タスクの起床待ち
	wup_tsk, iwup_tsk	タスクの起床
	rot_rdq, irot_rdq	タスクのローテーション
	sus_tsk, isus_tsk	他タスクのサスPEND
	rsm_tsk, irsm_tsk	サスPENDの解除
割込み関連	callback_int ^(*)2)	disp 有割込みハンドラ本体
その他の初期化	slos_init	OSの起動
	kinit ^(*)2)	OS初期化処理
	uinit ^(*)2)	ユーザ初期化処理
	stack_init ^(*)2)	タスクスタックの初期化
その他	idle ^(*)2)	アイドル処理
スタック操作	GET_REG ^(*)1)	ユーザタスクのスタックからのレジスタ参照
	SET_REG ^(*)1)	ユーザタスクのスタックのレジスタ設定

^(*)1) マクロです。

^(*)2) コールバックルーチンです。

6.7.2 OS ライブラリのリビルド

構築情報を変更してOSライブラリを再構築(リビルド)することにより、OSライブラリのROM/RAM容量を最適化することができます。構築情報は、slos.hに含まれます。不要な機能をコメントアウトして再構築(リビルド)してください。以下にslos.hに含まれる構築情報について説明します。

- ① プライオリティタスクの要否: KNL_BB_PRIORITY
プライオリティタスクが不要なシステムではコメントアウトして再構築してください。デフォルトではプライオリティタスクを使用します。
- ② ウェイクアップカウントの要否: KNL_BB_WUPCNT
ウェイクアップカウント(タスク起動要求の記憶機能)が不要なシステムではコメントアウトして再構築してください。デフォルトで当該機能はライブラリに含まれます。
ウェイクアップカウント(タスク起動要求の記憶機能)は、V3から追加された機能です。V2ベースのアプリケーションをV3からへ移植する場合、タスク管理 slp_tsk, wup_tsk の動作が変わる可能性があります。
本機能を無効にすることでV2ベースのタスク管理 slp_tsk, wup_tsk になります。
- ③ 時間管理機能の要否: KNL_BB_SYSTIME
時間管理機能が不要なシステムではコメントアウトして再構築してください。デフォルトで当該機能は、ライブラリに含まれます。本機能を外すことで周期ハンドラ機能も外されます。
- ④ 周期ハンドラの要否: KNL_BB_CYCHDR
周期ハンドラが不要なシステムではコメントアウトして再構築してください。デフォルトで当該機能はライブラリに含まれます。
- ⑤ イベントフラグの要否: KNL_BB_EVTFLG
イベントフラグが不要なシステムではコメントアウトして再構築してください。デフォルトで当該機能はライブラリに含まれます。
- ⑥ セマフォの要否: KNL_BB_SEMAPHORE
セマフォが不要なシステムではコメントアウトして再構築してください。デフォルトで当該機能はライブラリに含まれます。
- ⑦ データキューの要否: KNL_BB_DATAQUE
データキューが不要なシステムではコメントアウトして再構築してください。デフォルトで当該機能はライブラリに含まれます。

リスト6-10 構築情報(slos.h)

```
/*----- Building block-----*/
#define KNL_BB_PRIORITY ... ①
#define KNL_BB_WUPCNT ... ②
#define KNL_BB_SYSTIME ... ③
#ifndef KNL_BB_SYSTIME
#define KNL_BB_CYCHDR ... ④
#endif
#define KNL_BB_EVTFLG ... ⑤
#define KNL_BB_SEMAPHORE ... ⑥
#define KNL_BB_DATAQUE ... ⑦
```

以下はプライオリティタスクを使用しない場合の設定です。リストの様にコメントアウトしてから、OS ライブライアリを再構築(リビルド)してください。

リスト6-11 構築情報の変更(slos.h)

```
/*----- Building block-----*/
/* #define KNL_BB_PRIORITY */
#define KNL_BB_WUPCNT ... ②
#define KNL_BB_SYSTIME ... ③
#ifndef KNL_BB_SYSTIME
#define KNL_BB_CYCHDR ... ④
#endif
#define KNL_BB_EVTFLG ... ⑤
#define KNL_BB_SEMAPHORE ... ⑥
#define KNL_BB_DATAQUE ... ⑦
```

Smalight OS 構築用 CS+プロジェクト sols.mtpj をクリックして CS+を起動します。ビルド・モードから再構築する OS ライブライアリのメモリモデルを選択してください。リビルドして OS ライブライアリを再構築してください。

6.8 トレースを有効にする

トレース機能はデフォルトでは使用できません。トレース機能はOSライブラリの再構築することで自動的に取得できます。トレース機能を有効にする手順は以下の通りです。

6.8.1 OS ライブラリの再構築

- (1) 時間管理の有効・無効設定(slos.def の #define SYSTIM 定義)を決定し、設定します。
- (2) Smalight OS 構築用 CS+プロジェクト sols.mtpj をクリックして CS+を起動します。ビルド・モードから再構築する OS ライブラリのメモリモデルを選択してください。
- (3) C コンパイラオプションのマクロ定義で”KNL_TRACE”を設定します。
- (4) トレース定義ファイル(knl_trca.h)にてトレース領域のサイズを設定します。

リスト6-12 トレース領域のサイズ設定(knl_trca.h)

```
:
/*----- Trace Area Size -----*/
#define KNL_TRC_BSIZE      0x400L
:
```

トレース領域に確保できるエントリー数:Nの計算方法は以下の通りです。トレースフォーマットの詳細は「付録 H トレース情報」を参照ください。

$N = (KNL_TRC_BSIZE - 4) / \text{sizeof}(KNL_TRCTBL)$
※ Nが0となる設定はNGです。

注意事項 : OS ライブラリを再構築するとき、時間管理機能の有効・無効(slos.def の #define SYSTIM 定義)設定が影響します。時間管理機能の有効・無効設定を変更した場合、OS ライブラリのビルドを再度行う必要があります。

- (5) リビルドして OS ライブラリを再構築してください。

6.8.2 アプリケーションプロジェクトの設定

CS+プロジェクト ”u-ap.mtpj” をクリックして CS+を起動します。再構築した Smalight OS ライブラリで、再度ビルドしてください。

7. サンプルプログラム

各種機能を使用したシンプルなサンプルプログラムを用意しました。各種機能の動作確認や、サービスコールのパラメータ指定(コーディング方法)の確認などにご利用ください。

表7-1 サンプルプログラム一覧

No	ディレクトリ名	内容	備考
1	<smalight>¥os¥RL78_vvvv¥sample¥fig	イベントフラグの使用例	
2	<smalight>¥os¥RL78_vvvv¥sample¥sem	セマフォの使用例	
3	<smalight>¥os¥RL78_vvvv¥sample¥dtq	データキューの使用例	
4	<smalight>¥os¥RL78_vvvv¥sample¥cyc	周期ハンドラの使用例	(*)
5	<smalight>¥os¥RL78_vvvv¥sample¥R5F100LE¥	R5F100LE コードサンプル	
	interrupt¥non-disp	disp 無割込みの記述サンプル	
	interrupt¥disp	disp 有割込みの記述サンプル	
	main	hdwinits, main 関数サンプル	

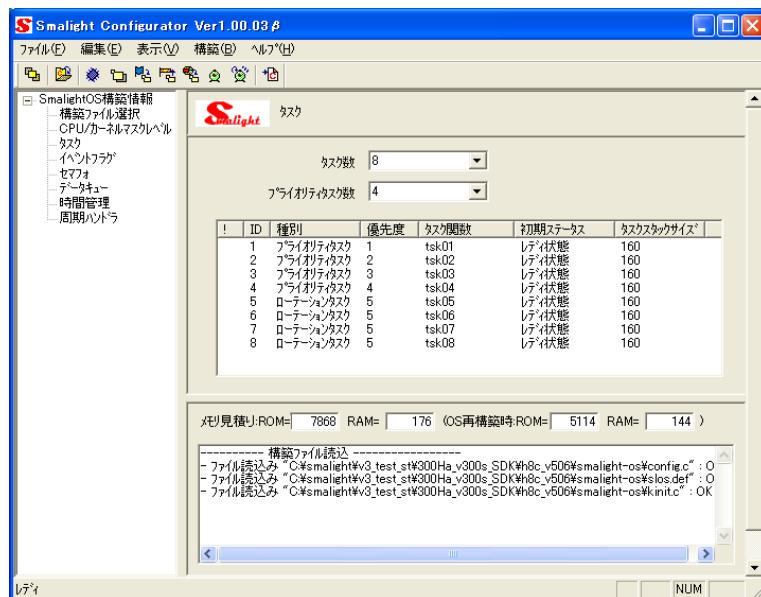
vvv : バージョン/リビジョン

(*) 周期タイマハンドラを用意する必要があります(ユーザ任意)。

8. コンフィギュレータ

Smalight OS V3 から、GUI ベースの構築支援ツール「Smalight Configurator」をサポートしました。本ツールを使用することで「OS 資源の初期化(kinit.c)」、「OS 定義ファイル(slos.def)」、「コンフィギュレーションファイルの編集(config.c)」を自動的に生成することができます。

「Smalight Configurator」は弊社ホームページ(<http://www.systemtech.maxell.co.jp/solution/smalight-index/>) からダウンロードしてご使用頂けます。詳細は「Smalight Configurator」のマニュアルをご参照ください。



※記載された仕様、デザイン等は予告なく変更する場合があります

図8-1 Smalight Configurator

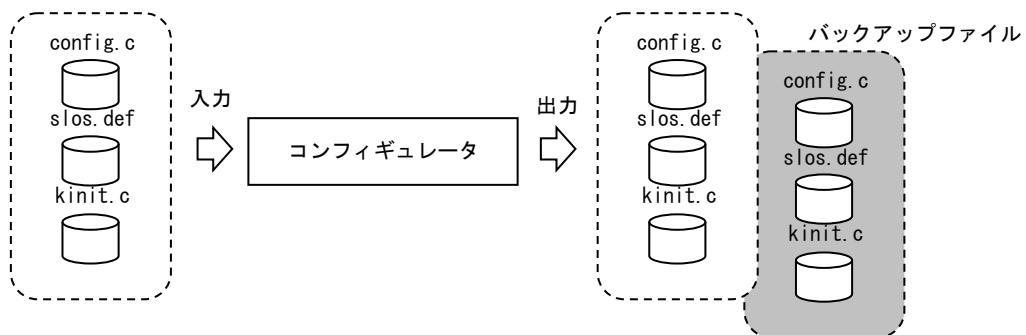


図8-2 Smalight Configuratorの入出力

付録A 変更履歴

【V3.10】

初版

【V3.10】

(1) マニュアルを訂正しました(第1版⇒第2版)

表A-1 マニュアル訂正(第1版⇒第2版)

項目	内容
リスト5-20 周期タイマハンドラの例	サンプルリストの誤記を修正しました。
リストG-1 GET_REG/SET_REGの活用例	サンプルリストの誤記を修正しました。

【V3.10】

(1) マニュアルを訂正しました(第2版⇒第3版)

表A-2 マニュアル訂正(第2版⇒第3版)

項目	内容
C.3	注意事項を追加しました。

【V3.10】

(1) マニュアルを訂正しました(第3版⇒第4版)

表A-3 マニュアル訂正(第3版⇒第4版)

項目	内容
CC-RLコンパイラ対応	CC-RLコンパイラ対応箇所を修正しました。
社名変更対応	社名に関わる記述箇所を修正しました。

付録B 制限事項

- (1) Smalight OS ではプログラムサイズを抑える工夫のひとつとして、サービスコールのパラメータに対するエラーチェックなど OS 内部のチェックを最小限に留めております。そのため、構築の設定ミスやサービスコールにて不正なパラメータを指定することで、メモリの内容が不正に書き換えられるなど、動作結果が不定となる場合があります。構築の設定やサービスコールのパラメータなど、使用範囲外の値や不正アドレスなどを指定しないようご注意ください。

付録C RL78 に関する注意事項

C.1 PSWレジスタについて

(1) システム起動中の PSW レジスタ

Smalight OS 起動からいずれかのタスクが起動(または、アイドル状態遷移)されるまで、PSW レジスタは初期化されません。必要に応じて、main()関数、または、uinit()コールバックにて初期化してください。但し、Smalight OS 起動時、IE ビット(割込み許可フラグ)は”0”(割込み禁止)のままとしてください。

- リセット直後の PSW レジスタ: 0x06(IE=0,RSB[1:0]=0,ISP[1:0]=3)

(2) タスク実行中の PSW レジスタ

タスク起動時の PSW レジスタの値はデフォルトで以下の様になっております。タスク実行中は基本的に RSB[1:0]は”0”、ISP[1:0]は”3”としてください。

- タスクの PSW レジスタ: 0x86(IE=1,RSB[1:0]=0,ISP[1:0]=3)

タスク実行中、一時的な ISP[1:0]ビット、IE ビットの操作(割込みのマスク)は自由に設定しても問題ありません。必要に応じて、変更してください。

(3) OS 実行中の PSW レジスタ

OS 実行中の PSW レジスタの値はデフォルトで以下の様になっております。

- PSW レジスタ初期値: 0x8x(IE=1,RSB[1:0]=0,ISP[1:0]=i)
 i = カーネルマスクレベル

(4) disp 有割込みの PSW レジスタ

disp 有割込みを受け付けると CPU により以下の様になります。

- PSW レジスタ初期値: 0x0x(IE=0,RSB[1:0]=0,ISP[1:0]=i)
 i = 受付けた割込みレベル

disp 有割込みハンドラでは以下の様に PSW を変更してから実行します。IE ビット(割込み許可ビット)は”1”(割込み許可)に変更するため、上位レベルの割込みは受付け可能となります(多重割込み)。

- PSW レジスタ初期値: 0x8x(IE=1,RSB[1:0]=0,ISP[1:0]=i)
 i = 受付けた割込みレベル

(6) disp 無割込みの PSW レジスタ

disp 無割込みを受け付けると CPU により以下の様になります。

- PSW レジスタ初期値: 0x0x(IE=0,RSB[1:0]=0,ISP[1:0]=i)
 i = 受付けた割込みレベル

割込み宣言(#pragma interrupt)により、レジスタバンク(RSB)指定、多重割込み許可指定が可能です。設定に従い PSW レジスタが変更されます。

C.2 RAMパリティ・エラー検出機能について

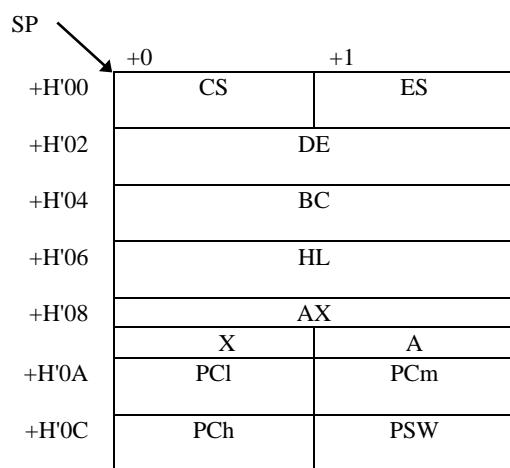
RL78 の内蔵機能に RAM パリティ・エラー検出機能がございます。これにより初期化されない内蔵 RAM をリードすると内部リセットが発生する恐れがございます。必要に応じて内蔵 RAM を初期化してから、Smalight OS を起動してください。RAM パリティ・エラー検出機能の詳細は使用する CPU のマニュアル ハードウェア編(安全機能)を参照ください。

C.3 SmalightOS ROMセクション配置について

SmalightOS の ROM セクションはアドレス 0000H-FFFFH(64K バイト空間)に配置する必要があります。

付録D スタック仕様

Ready, Waiting, Suspended, Waiting + Suspended 状態のタスク及び、disp 有割込み発生時のユーザスタックは以下に示すスタック仕様で格納されています。



図D-1 RL78のスタック仕様

付録E データ型、リターンコード

Smalight で規定するデータ型の一覧を以下に示します。

表E-1 データ型一覧

型	意味
B	符号付き8ビット整数
H	符号付き16ビット整数
W	符号付き32ビット整数
UB	符号無し8ビット整数
UH	符号無し16ビット整数
UW	符号無し32ビット整数
FP	void型関数へのポインタ
CFP	void型関数へのポインタ(ベクタ用)

サービスコールのリターンコード一覧を以下に示します。

表E-2 リターンコード一覧

シンボル	値	内容
E_OK	0 (H'00)	正常終了
E_ID	-18 (H'EE)	不正ID番号
E_ILUSE	-28 (H'E4)	サービスコール不正使用
E_TMOUT	-50 (H'CE)	ポーリング失敗またはタイムアウト

付録F トラブルシューティング

F.1 リンクエラーが発生する

Smalight OS の構築／設定に問題がある場合、リンクエラーが発生する恐れがあります。

- (1) イベントフラグ、セマフォ、データキュー、時間管理、周期ハンドラ、トレースなどの OS 機能を無効にしているにもかかわらず、関連するサービスコールを使用しているとき、以下のリンクエラーが発生します。

E0562310:Undefined external symbol "_knl_xxxxxx" referenced in "ファイル名"

xxxxxx : “evt”が含まれるとき、イベントフラグ無効でイベントフラグサービスコールが使用されている
“sem”が含まれるとき、セマフォ無効でセマフォサービスコールが使用されている
“dtq”が含まれるとき、データキュー無効でデータキューが使用されている
“tim”が含まれるとき、時間管理無効で時間管理サービスコールが使用されている
または、周期ハンドラ無効で周期ハンドラサービスコールが使用されている
“cyc”が含まれるとき、周期ハンドラ無効で周期ハンドラサービスコールが使用されている
“trc”が含まれるとき、トレース無効でトレースサービスコールが使用されている

【チェック項目】

- ① 無効設定のサービスコールが発行されていないか確認してください。
- ② サービスコールを使用している機能を有効にしてください。

- (2) 構築ファイルにて、定義したタスク関数の実体がないとき、以下のリンクエラーが発生します。

E0562310:Undefined external symbol "タスク関数" referenced in "ファイル名"

【チェック項目】

- ① config.c にて定義されるタスク関数名に誤りがないか確認ください。
- ② config.c にて定義されるタスク関数に相当する関数が存在しているか確認ください。

- (3) kinit.c ファイルにて、定義した周期ハンドラ関数の実体がないとき、以下のリンクエラーが発生します。

E0562310:Undefined external symbol "周期ハンドラ関数" referenced in "ファイル名"

【チェック項目】

- ① kinit.c にて定義される周期ハンドラ関数名に誤りがないか確認してください。
- ② kinit.c にて定義される周期ハンドラ関数に相当する関数が存在しているか確認してください。

付録G 応用例

G.1 GET_REG/SET_REGの活用

GET_REG/SET_REG の活用例を示します。例えば、slp_tsk からの起床要因が 2 種類(外部割込み、タイマ割込み)ある場合、割込みにて iwup_tsk と SET_REG を発行します。SET_REG で任意のデータをスタック上の DE, BC データに設定すること slp_tsk のリターンコードとして利用することができます。

リストG-1 GET_REG/SET_REGの活用例

```
void task1(void) /* タスク 1 */
{
    W ercd;
    ercd = slp_tsk(); /* 割込みを待つ */
    if(ercd == 0x12345678L) {
        /* 外部割込要因で起床した */
    } else if(ercd == 0x87654321L) {
        /* タイマ割込要因で起床した */
    }
    slp_tsk();
}
void intProc1(void) /* 外部割込ハンドラ */
{
    /* interrupt process */
    iwup_tsk(1u); /* タスク 1 を起床 */
    SET_REG(1u, 2, 0x1234); /* タスク 1 の DE, BC を操作 */
    SET_REG(1u, 4, 0x5678); /* (リターンコード設定と同意) */
}
void intProc2(void) /* タイマ割込ハンドラ */
{
    /* interrupt process */
    iwup_tsk(1u); /* タスク 1 を起床 */
    SET_REG(1u, 2, 0x8765); /* タスク 1 の DE, BC を操作 */
    SET_REG(1u, 4, 0x4321); /* (リターンコード設定と同意) */
}
```

※ 本例題の内容はイベントフラグにて容易に実現できます。

G.2 擬似的なter_tsk, sta_tsk

シングルチップ上でマルチタスクOSを使用した場合、小容量RAMのためタスクスタック確保が困難な場合があります。並行動作しないタスクであれば、1つのタスク定義で複数のタスクを動作させることができます(擬似的なter_tsk, sta_tsk)。

リストG-2 擬似的なter_tsk, sta_tskの実装例

```
void task1_proc1(void) /* タスク 1(処理 1) */
{
    /* 初期設定でタスク 1 はこの関数を定義している */
    :
    wup_tsk(2u); /* タスク操作をするタスク 2 を起床 */
    slp_tsk(); /* 本タスクの処理が終了したら slp_tsk でスリープ */
}
void task1_proc2(void) /* タスク 1(処理 2) */
{
    /* タスク 1 自体を本関数に変更します。*/
    :
    slp_tsk();
}

void task2_proc(void) /* タスク 2 */
{
    TSTACK *sp;
    UB i;

    slp_tsk();
    i = 1u; /* 指定タスク ID はタスク ID-1 */
    i--;

    sp = knl_tcbSpInit[i].sp;
    sp--;
    knl_tcbsp[i].sp = sp; /* スタックポインタの初期化 */

    memset((void *)sp, 0, (size_t)10);

    sp->cs = 0x00u; /* CS 書き換え */
    sp->es = 0x0Fu; /* ES 書き換え */
    sp->psw_pc = (UW)&task1_proc2; /* PC 書き換え */
    sp->psw_pc &= 0x00FFFFFFuL; /* PSW 書き換え */
    sp->psw_pc |= 0x86000000uL;

    wup_tsk(1u); /* タスク 1 を起床 */
    slp_tsk();
}
```

G.3 stack_initを利用したパラメータ引渡し

デフォルトの stack_init ではタスクスタックの CS, ES, PC, PSW だけを初期化しています。BC, AX を設定することでタスク関数の引数を設定できます。

リストG-3 stack_initを利用したパラメータ引渡し

```
void stack_init(UB tid, TSTACK *sp)
{
    extern const TCBADDR knl_tcbAddrInit[];

    if(tid == 1) {
        sp->bc = 0;           /* BC(引数1上位)=0 */
        sp->ax = 0;           /* AX(引数1下位)=0 */
    } else if(tid == 2) {
        sp->bc = 0;           /* BC(引数1上位)=0 */
        sp->ax = 1;           /* AX(引数1下位)=1 */
    }

    memset((void *)sp, 0, (size_t)10);

    sp->cs = 0x00u;
    sp->es = 0x0Fu;
    sp->psw_pc = (UW)knl_tcbAddrInit[tid-1u].tsk;
    sp->psw_pc &= 0x00FFFFFFuL;
    sp->psw_pc |= 0x86000000uL;
}

void task_proc(W pra) /* タスク1,2 兼用関数 */
{
    if(pra == 0) {
        /* タスク1の処理 */
    } else if(pra == 1) {
        /* タスク2の処理 */
    }
    slp_tsk();
}
```

付録H トレース情報

H.1 テーブル仕様

トレース格納時に以下のテーブル仕様にて格納されます。

	+0	+1	+2	+4
+H'00	トレース種別	タスク ID	トレース情報 1	トレース情報 2
+H'04			トレース情報 3	
+H'08	リザーブ		システム時間(上位 16 ピット)	
+H'0C			システム時間(下位 32 ピット)	

※ 網掛け部は時間管理を使用している場合のみ有効となります。

図H-1 トレースのテーブル仕様

【時間管理を使用していない場合】

```
typedef struct _knl_trctbl {
    UB    type;          /* トレース種別 */
    UB    tid;           /* タスクID */
    UB    dat1;          /* トレース情報1 */
    UB    dat2;          /* トレース情報2 */
    UW    dat3;          /* トレース情報3 */
} KNL_TRCTBL;
```

【時間管理を使用している場合】

```
typedef struct _knl_trctbl {
    UB    type;          /* トレース種別 */
    UB    tid;           /* タスクID */
    UB    dat1;          /* トレース情報1 */
    UB    dat2;          /* トレース情報2 */
    UW    dat3;          /* トレース情報3 */
    UH    rsv;           /* 未使用 */
    UH    htim;          /* システム時刻 上位16ビット */
    UW    ltim;          /* システム時刻 下位32ビット */
} KNL_TRCTBL;
```

H.2 ト雷斯種別

(1) サービスコールトレス

サービスコールが発行されたタイミングにてトレスが取得されます。取得されるトレス情報は以下の内容となります。

表H-1 サービスコールトレスの格納情報

項目	設定値	備考
トレス種別	‘S’(H’53)	
タスク ID	発行元タスク ID	(*)1)
トレス情報 1	サービスコール ID	(*)2)
トレス情報 2	個別情報	(*)3)
トレス情報 3	発行元アドレス	(*)4)
システム時間	システム時間	(*)5)

(*)1 i 付き、関数型サービスコールでの場合、発行元タスク ID ではなく Running 状態のタスク ID が格納されます。Running 状態のタスクがない場合、0(アンドル)が格納されます。

(*)2 サービスコール ID の詳細は「表 H-2 サービスコール ID とトレス情報①」を参照ください。

(*)3 サービスコール毎に個別情報が格納されます。詳細は「表 H-2 サービスコール ID とトレス情報①」を参照ください。

(*)4 i 付き、関数型サービスコールでの場合、発行元アドレスは取得できません。発行元アドレスとは発行されたサービスコールからの戻りアドレスを意味します。

(*)5 時間管理機能が無効のとき、システム時間は取得されません。

表H-2 サービスコールIDとトレス情報①

サービスコール ID	サービスコール名	トレス情報 2	トレス情報 3	備考
1	slp_tsk	未使用(0)	発行元アドレス	
2	tslp_tsk	未使用(0)	発行元アドレス	
3	wup_tsk	引数: タスク ID	発行元アドレス	
4	iwup_tsk	引数: タスク ID	未使用(0)	
5	can_wup	引数: タスク ID	未使用(0)	
6	rot_rdq	未使用(0)	発行元アドレス	
7	irot_rdq	未使用(0)	未使用(0)	
8	sus_tsk/isus_tsk	引数: タスク ID	未使用(0)	
9	-	-	-	
10	rsm_tsk/irsm_tsk	引数: タスク ID	未使用(0)	
11	-	-	-	
12	wai_flg	引数: イベントフラグ ID	発行元アドレス	
13	twai_flg	引数: イベントフラグ ID	発行元アドレス	
14	set_flg	引数: イベントフラグ ID	発行元アドレス	
15	iset_flg	引数: イベントフラグ ID	未使用(0)	
16	clr_flg	引数: イベントフラグ ID	未使用(0)	
17	EVT_INIT	-	-	
18	EVT_ATTR	-	-	
19	wai_sem	引数: セマフォ ID	発行元アドレス	
20	twai_sem	引数: セマフォ ID	発行元アドレス	
21	sig_sem	引数: セマフォ ID	発行元アドレス	
22	isig_sem	引数: セマフォ ID	未使用(0)	
23	SEM_INIT	-	-	
24	SEM_ATTR	-	-	

※ 網掛けされたサービスコールはトレス取得対象外です。

表H-3 サービスコールIDとトレース情報②

サービスコール ID	サービスコール名	トレース情報 2	トレース情報 3	備考
25	rcv_dtq	引数: データキュー ID	発行元アドレス	
26	trcv_dtq	引数: データキュー ID	発行元アドレス	
27	snd_dtq	引数: データキュー ID	発行元アドレス	
28	isnd_dtq	引数: データキュー ID	未使用(0)	
29	tsnd_dtq	引数: データキュー ID	発行元アドレス	
30	fsnd_dtq	引数: データキュー ID	発行元アドレス	
31	ifsnd_dtq	引数: データキュー ID	未使用(0)	
32	DTQ_INIT	-	-	
33	DTQ_ATTR	-	-	
34	set_tim	未使用(0)	未使用(0)	
35	get_tim	未使用(0)	未使用(0)	
36	slos_cyclic_timer	-	-	
37	SYSTIM_INIT	-	-	
38	sta_cyc	引数: 周期ハンドラ ID	未使用(0)	
39	stp_cyc	引数: 周期ハンドラ ID	未使用(0)	
40	CYC_INIT	-	-	
41	CYC_ATTR	-	-	
42	CYC_CHG	-	-	
43	cyc_hdr	-	-	
44	-	-	-	
45	-	-	-	
46	disp	未使用(0)	未使用(0)	
47	callback_int	-	-	
48	slos_init	-	-	
49	kinit	-	-	
50	uinit	-	-	
51	stack_init	-	-	
52	idle	-	-	
53	GET_REG	-	-	
54	SET_REG	-	-	

※ 網掛けされたサービスコールはトレース取得対象外です。

(2) ディスパッチトレース

OS 実行(ディスパッチャ)からタスク実行へ遷移するタイミングにてトレースが取得されます。取得されるトレース情報は以下の内容となります。

表H-4 ディスパッチトレースの格納情報

項目	設定値	備考
トレース種別	'@'(H'40)	
タスク ID	ディスパッチされたタスク ID	
トレース情報 1	未使用(0)	
トレース情報 2	未使用(0)	
トレース情報 3	未使用(0)	
システム時間	システム時間	

(3) アイドルトレース

OS 実行(ディスパッチャ)からアイドル処理へ遷移するタイミングにてトレースが取得されます。取得されるトレース情報は以下の内容となります。

表H-5 アイドルトレースの格納情報

項目	設定値	備考
トレース種別	'I'(H'49)	
タスク ID	未使用(0)	
トレース情報 1	未使用(0)	
トレース情報 2	未使用(0)	
トレース情報 3	未使用(0)	
システム時間	システム時間	

(4) ユーザトレース

set_trc サービスコールを発行したタイミングにてトレースが取得されます。取得されるトレース情報は以下の内容となります。

表H-6 ユーザトレースの格納情報

項目	設定値	備考
トレース種別	'U'(H'55)	
タスク ID	発行元タスク ID	(*1)
トレース情報 1	ユーザ任意の値	(*2)
トレース情報 2	ユーザ任意の値	(*2)
トレース情報 3	ユーザ任意の値	(*2)
システム時間	システム時間	

(*1) 発行元がタスク部でない場合、Running 状態のタスク ID が格納されます。Running 状態のタスクがない場合、0(アイドル)が格納されます。

(*2) set_trc サービスコールの引数で指定されるユーザトレース情報 1～3 が設定されます。

Smalight®

Smalight® OS V3 リファレンスマニュアル RL78版
SLRL78KNL03MJ-4

発行年月 2018年1月 第4版
発 行 マクセルシステムテック株式会社
編 集 マクセルシステムテック株式会社

©マクセルシステムテック株式会社 2018