



# JMバッジボード上での TOPPERS/JSP カーネル

---

## JM バッジボード上での TOPPERS/JSP カーネル ユーザマニュアル

Rev. C

# 目次

1 概要.....	3
2 参考資料.....	3
3 ColdFire V1 用の TOPPERS/JSP カーネル.....	3
3.1 ハードウェア概要.....	4
3.2 TOPPERS/JSP カーネルのアーキテクチャ、ソースコード及び機能.....	4
3.2.1 TOPPERS のアーキテクチャ概要.....	4
3.2.2 ソースツリー.....	6
3.2.3 システム機能.....	7
3.3 MCF51JM 上の TOPPERS/JSP カーネル構成.....	15
3.3.1 MCF51JM128 上の構成.....	15
3.3.2 CPU 例外及び割り込みハンドラの設定.....	15
3.3.3 ブート TOPPERS.....	17
3.4 開発環境.....	17
4 プログラミング.....	17
4.1 アプリケーションの作成.....	17
4.2 アプリケーションのビルド.....	19
4.3 イメージをバッジへロードする方法.....	19
4.4 アプリケーションの実行.....	19
5 TOPPERS とデモアプリケーション.....	19

## 1 概要

---

本文書は iTRON の実装、TOPPERS/JSP カーネルリアルタイム・オペレーティングシステム ( OS ) とその上で動作するアプリケーションを、フリースケール JM バッジボード上の MCF51JM128 ColdFire® V1 マイクロコントローラ向けに開発する方法について記述するものです。

TOPPERS/JSP カーネルは、ロイヤルティフリーでオープンソースの組込み・リアルタイム OS であり、TOPPERS プロジェクトにより開発されました。JSP は「Just Standard Profile」の頭字語であり、 $\mu$ iTRON4.0 仕様のスタンダードプロファイル規定に従って実装されています。ソースコードは <http://sourceforge.jp/projects/toppers-cf/>にある [ColdFire V2](#) 用の TOPPERS/JSP カーネルに基づいています。ここで使用しているバージョンは 1.4.3-3 です。

バッジボードは、ColdFire® V1 マクロコントローラに基づいた MCF51JM128 と、 $\mu$ iTRON 4.0 仕様に準拠した TOPPERS/JSP カーネルを特長としています。

本ユーザマニュアルは、バッジボード上の TOPPERS/JSP カーネル用の iTRON アプリケーションを開発するプログラマ向けのものです。

## 2 参考資料

---

- JM バッジボードユーザマニュアル
- [TOPPERS/JSP for ColdFire V2](#) ( 本資料は TOPPERS パッケージに含まれています )
- Porting GNU C Programs to Freescale's CodeWarrior™ C Compiler
- [\$\mu\$ iTRON4.0 Specification](#)
- [\$\mu\$ iTRON4.0 Device Driver Guideline Specification](#)

## 3 ColdFire V1 用の TOPPERS/JSP カーネル

---

## 3.1 ハードウェア概要

フリースケール JM バッジボード上の MCF51JM128 ColdFire® V1 マイクロコントローラが開発のターゲットです。64 ピン LQFP パッケージの MCF51JM128 は、JM バッジボードのホストコントローラであり、MCF51JM デバイスは Flexis ファミリの中で最新の製品です。これらのデバイスは S08JM ファミリと、ピン、ペリフェラル、及びツールセットに互換性があります。デモアプリケーションで使用され、TOPPERS/JSP カーネルソースコードによって変更されるバッジボードの機能をまとめると以下のようになります。

- ColdFire V1 コア
- RTC
- USB コントローラ
- I<sup>2</sup>C バスインタフェース
- タイマモジュール ( TPM )
- 汎用 I/O
- ADC
- 加速度計
- 容量タッチセンサ
- LED マトリクス

## 3.2 TOPPERS/JSP カーネルのアーキテクチャ、ソースコード及び機能

### 3.2.1 TOPPERS のアーキテクチャ概要

TOPPERS/JSP カーネルのソースコードはハードウェアに依存しない ( カーネル独立 ) 部分とハードウェアに依存する部分の 2 つに主に分けられます。

独立部分は、ハードウェアと独立して実装されるカーネルと API 関数を含んでおり、µiTRON4.0 仕様に準拠しています。そのコードベースが GNU C で書かれています。カーネルモジュール及びシステム機能の特徴は 3.3.3 節に述べてあります。

依存部分は、プロセッサとターゲットボードシステムに従って実装されています。コードはターゲットシステムのハードウェアアーキテクチャに基づき、通常 GNU C、GNU のインライン ASM、及びアセンブリで書かれています。

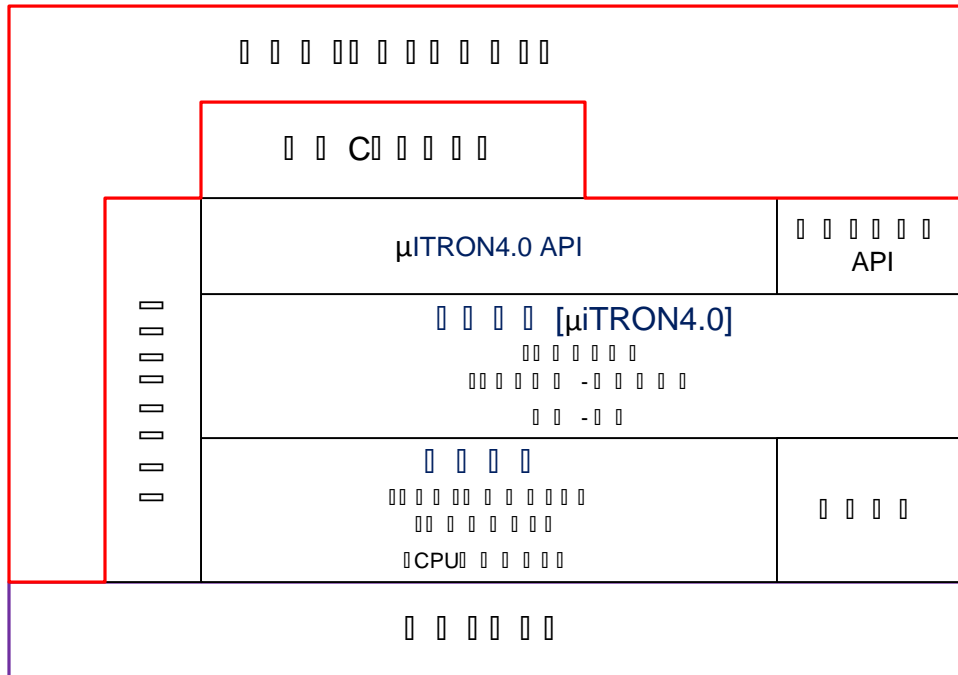


図 1 TOPPERS アーキテクチャ概要

TOPPERS/JSP カーネルにおいて、依存部分はさらに 2 つの部分に分けられています。

- プロセッサ依存部分
- システム依存部分

プロセッサ依存部分は、プロセッサアーキテクチャに依存するコードであり、プロセッサの初期化、TOPPERS/JSP カーネルにより管理されるベクタテーブルの導入及び割り込みハンドラと CPU 例外ハンドラの設定を含んでいます。さらに、割り込み優先度レベルを管理する特別な関数を実装することもできます。ColdFire アーキテクチャには、割り込み管理のために、割り込みマスクの変更と割り込みマスクの取得という 2 つのルーチンが必要です。

システム依存部分は、システムに依存する部分のために書かれたコードであり、JM128 バッジボードのデバイスの専用のものであります。それらのデバイスとは、メモリマップ、汎用 I/O、ADC、加速度計と I<sup>2</sup>C バスです。

Coldfire V1 用 TOPPERS/JSP カーネルの依存部分の構成は、3.3.4 節に記述してあります。

### 3.2.2 ソースツリー

MCF51JM128 用の TOPPERS/JSP カーネルの例を開発・実行・シミュレーションするには、マクロコントローラ IDE6.2 版用のフリースケール Code Warrior を使用します。図 2、図 3 にはプロジェクトソースツリー及びプロジェクトの例を示します。

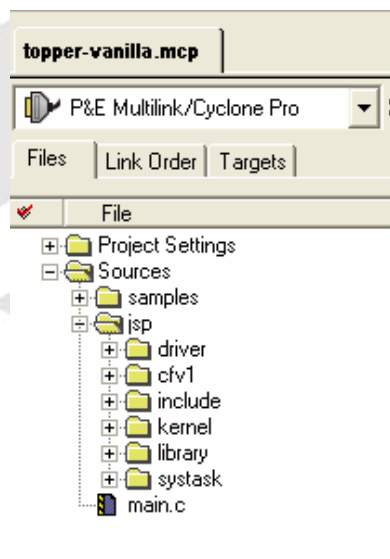


図 2 CodeWarrior での TOPPERS/JSP カーネルのソースツリー

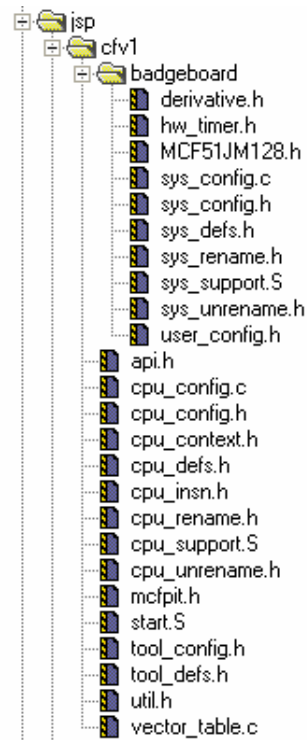


図 3 CodeWarrior での依存部分のソースツリー

### 3.2.3 システム機能

本節では、TOPPERS/JSP カーネルのシステム機能について説明します。

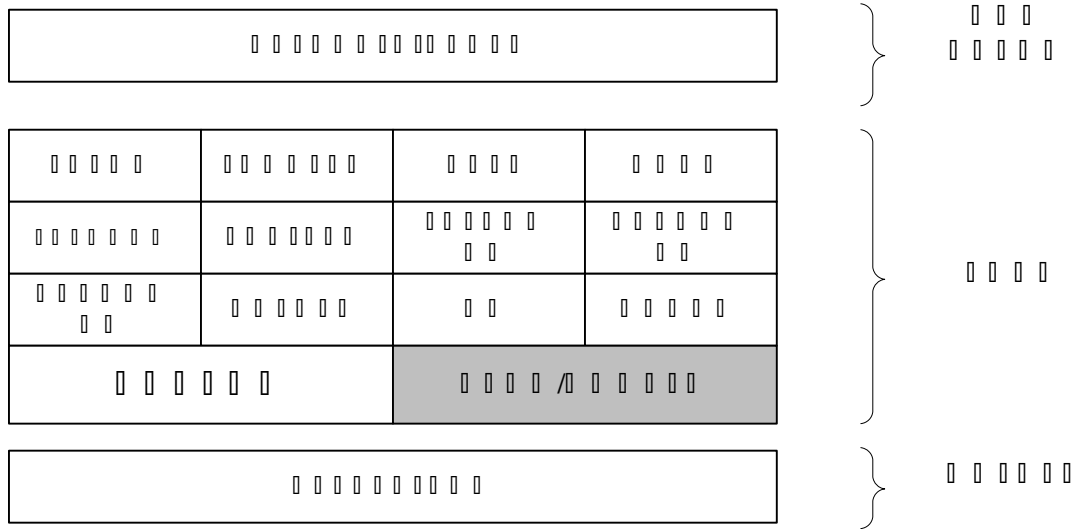


図 4 カーネルコンポーネントアーキテクチャ

**スケジューラ:** タスク優先度に基づいて処理キューを作成し、その実行を制御することで、より優先度の高いタスク (優先度の値が低いタスク) が先に処理されるようにします。

**タスク管理モジュール:** タスク状態の直接制御とタスク状態へのリファレンスを提供します。

**タスク同期モジュール:** タスク状態を直接制御してタスクの同期化します。

**割り込み管理モジュール:** 割り込みハンドラと外部割り込みサービスルーチンを管理します。

**時間管理モジュール:** 時間に依存する処理を提供します。( JSPカーネルはアラームハンドラを提供していません )

**システム状態管理モジュール:** JSPカーネルのシステム状態を取得します。

**システム構成管理モジュール:** JSPカーネルの版数等の情報を指定します。

**同期・通信モジュール**：タスクに依存しないオブジェクトを通して、タスク間の同期と通信を行います。そのオブジェクトとは、セマフォ、データキュー、イベントフラグやメールボックスです。

- **イベントフラグ**：1ビットが1イベントを表すビットパターンで表現される同期オブジェクトです。
- **セマフォ**：相互排除と同期用のオブジェクトです。1つのセマフォはリソースカウンタを使用して、未使用リソースの状況とその数を示します。
- **メールボックス**：共有メモリにあるメッセージを送受信する、同期と通信のためのオブジェクトです。
- **データキュー**：データエレメントと呼ばれる1語メッセージを送受信する、同期と通信のためのオブジェクトです。

**メモリプール管理モジュール**：ソフトウェアのダイナミックメモリ管理を行います。

JSPカーネルシステム機能を次の各テーブルに示します。

### 3.2.3.1 タスク管理機能

機能名	内容	API	実装状況
CRE_TSK	タスクの生成	静的	実装済
act_tsk	タスクの起動	C 言語	実装済
iact_tsk	タスクの起動	C 言語	実装済
can_act	タスク起動要求のキャンセル	C 言語	実装済
ext_tsk	自タスクの終了	C 言語	実装済
ter_tsk	タスクの強制終了	C 言語	実装済
chg_pri	タスク優先度の変更	C 言語	実装済
get_pri	タスク優先度の参照	C 言語	実装済

JSPカーネル：8機能が実装済み。

### 3.2.3.2 タスク依存同期機能

機能名	内容	API	実行状態
slp_tsk	タスクをスリープにさせる	C 言語	実装済
tslp_tsk	タスクをスリープにさせる ( タイムアウトあり )	C 言語	実装済
wup_tsk	タスクの起床要求	C 言語	実装済
iwup_tsk	タスクの起床	C 言語	実装済
can_wup	タスク起床要求のキャンセル	C 言語	実装済
rel_wai	待ち状態の強制解除	C 言語	実装済
irel_wai	タスク待ち状態の強制解除	C 言語	実装済
sus_tsk	タスク強制待ち状態への移行	C 言語	実装済
rsm_tsk	強制待ち状態タスクの再開	C 言語	実装済
frsm_tsk	強制待ち状態タスクの強制開始	C 言語	実装済
dly_tsk	タスクの遅延	C 言語	実装済

JSPカーネル : 11機能が実装済み。

### 3.2.3.3 タスク例外処理機能

機能名	内容	API	実行状態
DEF_TEX	タスク例外処理ルーチンの定義	静的	実装済
ras_tex	タスク例外処理の要求	C 言語	実装済
iras_tex	タスク例外処理の要求	C 言語	実装済
dis_tex	タスク例外処理の禁止	C 言語	実装済
ena_tex	タスク例外処理の許可	C 言語	実装済
sns_tex	タスク例外処理禁止状態の参照	C 言語	実装済

JSPカーネル : 6機能が実装済み。

### 3.2.3.4 同期・通信機能

#### セマフォ

機能名	内容	API	実行状態
CRE_SEM	セマフォの作成	静的	実装済
sig_sem	セマフォリソースの開放	C 言語	実装済
isig_sem	セマフォリソースの開放	C 言語	実装済
wai_sem	セマフォリソースの獲得	C 言語	実装済
pol_sem	セマフォリソースの獲得 ( ポーリング )	C 言語	実装済

twai_sem	セマフォリソースの獲得 ( タイムアウトあり )	C 言語	実装済
----------	--------------------------	------	-----

JSPカーネル : 6機能が実装済み。

#### イベントフラグ

機能名	内容	API	実行状態
CRE_FLG	イベントフラグの作成	静的	実装済
set_flg	イベントフラグの設定	C 言語	実装済
iset_flg	イベントフラグの設定	C 言語	実装済
clr_flg	イベントフラグのクリア	C 言語	実装済
wai_flg	イベントフラグ待ち	C 言語	実装済
pol_flg	イベントフラグ待ち ( ポーリング )	C 言語	実装済
twai_flg	イベントフラグ待ち ( タイムアウトあり )	C 言語	実装済

JSPカーネル : 7 機能が実装済み。

#### データキュー

機能名	内容	API	実行状態
CRE_DTQ	データキューの作成	静的	実装済
snd_dtq	データキューへの送信	C 言語	実装済
psnd_dtq	データキューへの送信 ( ポーリング )	C 言語	実装済
ipsnd_dtq	データキューへの送信 ( ポーリング )	C 言語	実装済
tsnd_dtq	データキューへの送信 ( タイムアウトあり )	C 言語	実装済
fsnd_dtq	データキューへの強制送信	C 言語	実装済
ifsnd_dtq	データキューへの強制送信	C 言語	実装済
rcv_dtq	データキューからの受信	C 言語	実装済
prcv_dtq	データキューからの受信 ( ポーリング )	C 言語	実装済
trcv_dtq	データキューからの受信 ( タイムアウトあり )	C 言語	実装済

JSPカーネル : 10機能が実装済み。

#### メールボックス

機能名	内容	API	実行状態
CRE_MBX	メールボックスの作成	静的	実装済
snd_mbx	メールボックスへの送信	C 言語	実装済
rcv_mbx	メールボックスからの受信	C 言語	実装済
prcv_mbx	メールボックスからの受信 ( ポーリング )	C 言語	実装済
trcv_mbx	メールボックスからの受信 ( タイムアウトあり )	C 言語	実装済

JSPカーネル : 5機能が実装済み。

### 3.2.3.5 拡張同期・通信機能

#### ミューテックス

機能名	内容	API	実行状態
-----	----	-----	------

JSPカーネル：実装済み機能なし。

#### メッセージバッファ

機能名	内容	API	実行状態
-----	----	-----	------

JSPカーネル：実装済み機能なし。

#### ランデブポート

機能名	内容	API	実行状態
-----	----	-----	------

JSPカーネル：実装済み機能なし。

### 3.2.3.6 メモリプール管理機能

#### 固定長メモリプール

機能名	内容	API	実行状態
CRE_MPF	固定長メモリプールの作成	静的	実装済
get_mpf	固定長メモリブロックの獲得	C 言語	実装済
pget_mpf	固定長メモリブロックの獲得 (ポーリング)	C 言語	実装済
tget_mpf	固定長メモリブロックの獲得 (タイムアウトあり)	C 言語	実装済
rel_mpf	固定長メモリブロックの開放	C 言語	実装済

JSPカーネル：5機能が実装済み。

#### 可変サイズメモリプール

機能名	内容	API	実行状態
-----	----	-----	------

JSPカーネル：実装済み機能なし。

### 3.2.3.7 時間管理機能

#### システム時刻管理

機能名	内容	API	実行状態
set_tim	システム時刻の設定	C 言語	実装済

get_tim	システム時刻の参照	C 言語	実装済
isig_tim	タイムティックの供給	C 言語	実装済

JSPカーネル：3機能が実装済み。

#### 周期ハンドラ

機能名	内容	API	実行状態
CRE_CYC	周期ハンドラの作成	静的	実装済
sta_cyc	周期ハンドラの動作開始	C 言語	実装済
stp_cyc	周期ハンドラの動作停止	C 言語	実装済

JSPカーネル：3機能が実装済み。

#### アラームハンドラ

機能名	内容	API	実行状態

JSPカーネル：実装済み機能なし。

#### オーバランハンドラ

機能名	内容	API	実行状態

JSPカーネル：実装済み機能なし。

### 3.2.3.8 システム状態管理機能

機能名	内容	API	実行状態
rot_rdq	タスクの優先順位の回転	C 言語	実装済
irotd_rdq	タスクの優先順位の回転	C 言語	実装済
get_tid	実行状態のタスクIDの参照	C 言語	実装済
iget_tid	実行状態のタスクIDの参照	C 言語	実装済
loc_cpu	CPUのロック	C 言語	実装済
iloc_cpu	CPUのロック	C 言語	実装済
unl_cpu	CPUロックの解除	C 言語	実装済
iunl_cpu	CPUロックの解除	C 言語	実装済
dis_dsp	ディスパッチのディセーブル	C 言語	実装済
ena_dsp	ディスパッチのイネーブル	C 言語	実装済
sns_ctx	コンテキストの参照	C 言語	実装済
sns_loc	CPUロック状態の参照	C 言語	実装済
sns_dsp	ディスパッチ状態の参照	C 言語	実装済
sns_dpn	ディスパッチ保留状態の参照	C 言語	実装済

JSPカーネル：14機能が実装済み。

### 3.2.3.9 割込み管理機能

機能名	内容	API	実行状態
DEF_INH	割込みハンドラの定義	静的	実装済
dis_int	割込みのディセーブル	C 言語	実装済
ena_int	割込みのイネーブル	C 言語	実装済
chg_ixx	割込みマスクの変更	C 言語	実装済
get_ixx	割込みマスクの参照	C 言語	実装済

JSPカーネル：5機能が実装済み。

### 3.2.3.10 サービスコール管理機能

機能名	内容	API	実行状態

JSPカーネル：実装済み機能なし。

### 3.2.3.11 システム構成管理機能

機能名	内容	API	実行状態
DEF_EXC	CPU例外ハンドラの定義	静的	実装済
ATT_INI	初期化ルーチンの追加	静的	実装済

JSPカーネル：2機能が実装済み。

JSPカーネルには、 $\mu$ iTRON 4.0のスタンダードプロファイルに従って実装された73機能と11静的機能があります。

## 3.3 MCF51JM 上の TOPPERS/JSP カーネル構成

### 3.3.1 MCF51JM128 上の構成

Coldfire V1 プロセッサの特徴及び MCF51JM128 のシステムアーキテクチャに従い、TOPPERS/JSP カーネルの依存部分での構成は以下のようになっています。

- メモリマップ設定：MCF51JM128 上の SRAM とフラッシュのメモリセクションは Project.lcf ファイルに設定されており、プログラマが自由に変更することができます。以下はリンクファイルの例です。

```
MEMORY {  
  bootcode      (RX) : ORIGIN = 0x00000410, LENGTH = 0x000033F0  
  code          (RX) : ORIGIN = 0x00003A00, LENGTH = 0x0001C600  
  vectorram    (RWX) : ORIGIN = 0x00800000, LENGTH = 0x00000200  
  userram      (RWX) : ORIGIN = 0x00800200, LENGTH = 0x00003E00  
}
```

- タイマレジスタ：リアルタイムカウンタ (RTC) モジュールは機能のスケジューリングに使われます。タイマハンドラは 8ms の間隔で実行するように設定されています。
- シリアルドライバ：MCF51JM128 に構成されていません。

### 3.3.2 CPU 例外及び割込みハンドラの設定

MCFJM128 には、ベクタテーブルで宣言されている関数ポインタが 111 個あります。MCF51JM128 プロセッサ用のベクタテーブル設定は以下の通りです。

#### 3.3.2.1 割込みハンドラ

TOPPERS のカーネルでは、タイマに 1 つの割込みが用いられます。

CPU ロック状態は、依存部分の `disint()` と `enaint()` ルーチンで変更出来ます。また、ユーザ管理の割込みに対する追加関数は、サービス呼び出しとして実装されています。

UH ipm : 割込み優先度マスクの設定値。全割込みを有効にするには、この値を「0」に、全割込みを無効にするには、この値を「1」に設定します。

get\_ipm : 現在の IPM 値を取得します。

chg\_ipm : IPM 値を変更します。

### 3.3.2.2 CPU 例外ハンドラ

JSP カーネルには 2 種類の例外ハンドラがあります。

- システム例外ハンドラ
- タスク例外ハンドラ

vector\_table.c における TOPPERS/JSP カーネル設定のソースコードは以下の通りです。

```
FP vector_table [111] = {
    (FP)INIT_STACK,      /* 0:スタックポインタの初期値 */
    RESET_VECTOR,       /* 1: パワーオンリセット */
    ACC_VECTOR,         /* 2: アクセスエラー */
    ADD_VECTOR,         /* 3: アドレスエラー */
    II_VECTOR,          /* 4: 一般的な無効説明 */
    RESERVED_VECTOR,    /* 5: ゼロに分ける */
    RESERVED_VECTOR,
    II_VECTOR,          /* 7: システム予約 */
    RESERVED_VECTOR,    /* 8: 特権違反 */
    RESERVED_VECTOR,    /* 9: トレース */
    RESERVED_VECTOR,    /* 10: a エミュレータを列に並べる */
    RESERVED_VECTOR,    /* 11: f エミュレータを列に並べる */
    ...
}
```

### 3.3.3 ブート TOPPERS

イメージにはカーネル、デバイスドライバやアプリケーションコードが含まれています。このイメージは CODE エリアの開始アドレスに置かれています。ブートローダが実行を完了してから、TOPPERS/JSP カーネルの動作が始まります。

## 3.4 開発環境

開発ホスト : Windows 用 Code Warrior 6.2

ターゲットシステム : JM バッジボード

ツール : TOPPERS/JSP カーネルコンフィギュレータ ( TOPPERS パッケージの /cfg/cfg.exe パスに含まれています。 )

## 4 プログラミング

---

本節では TOPPERS/JSP カーネルアプリケーションを作成する方法について説明します。

### 4.1 アプリケーションの作成

アプリケーションソースコードは以下の 2 つの部分から構成されます。

- 構成ファイル(\*.cfg)
- C ソースコード(\*.c)とヘッダファイル(\*.h)

C ファイル名は構成ファイル名と同じでなければなりません。構成ファイルの機能は、前節で述べたシステム機能テーブルにある静的 API のみでしか呼び出すことができません。構成情報は通常構成ファイル内に定義され、以下の種類があります。

- タスク情報
- タスク起動アドレス
- 例外ハンドラ
- 割り込みハンドラ

- 周期ハンドラ

C ファイルには、構成ファイルで定義したタスクのメインボディを書き込むことができます。以下の例は、プログラマがバッジボードで動作するアプリケーションを作成する方法を理解するための助けとするものです。

これは、スイッチングタスクと周期時間についての例です。

```
----- sample.cfg -----
#define _MACRO_ONLY
#include "sample1.h"

INCLUDE("\sample1.h\");
CRE_TSK(TASK1, { TA_HLNG, (VP_INT) 1, task, MID_PRIORITY, 256, NULL });
CRE_TSK(MAIN_TASK, { TA_HLNG|TA_ACT, 0, main_task, MAIN_PRIORITY,256, NULL });

DEF_EXC(CPUEXC1, { TA_HLNG, cpuexc_handler });

#include "timer.cfg"

----- sample.c -----
void task(VP_INT exinf)
{
    for (;;) {
        if (counter >= 625) {
            counter = 0;
            wup_tsk(MAIN_TASK);
        }
    }
}

void main_task(VP_INT exinf)
{
    int i = 0;

    act_tsk(TASK1);
    counter = 0;
    for (;;) {
        if (counter >= 625) {
            counter = 0;
            slp_tsk();
        }
    }
}
```

この例を TOPPERS/JSP カーネルと共にコンパイルする前に、jsp/cfg ディレクトリにある「cfg.exe」というオリジナル GNU ツールを使用してカーネルと例との構成を行い、カーネルとアプリケーションとのコンパイル用の kernel\_cfg.c と kernel\_id.h 2 つのファイルを生成することが必要となります。

生成コマンド：

```
cfg\cfg.exe -cfg sample.cfg -kernel_id kernel_id.h -kernel_cfg kernel_cfg.c
```

## 4.2 アプリケーションのビルド

生成したファイル kernel\_cfg.c と kernel\_id.h をプロジェクトディレクトリにコピーすると、プロジェクトをビルドできます。ビルドに成功した場合、結果はイメージファイル Project.abs.s19 となります。

## 4.3 イメージをバッジへロードする方法

バッジ用のブートローダを使用してイメージをフラッシュにダウンロードします。S19 イメージファイルをブートへロードする方法については、JM バッジボードユーザマニュアルを参照してください。

## 4.4 アプリケーションの実行

アプリケーションをフラッシュへロードした後、電源ボタンでバッジボードを再起動させ、フラッシュ RAM からデモアプリケーションを実行します。

# 5 TOPPERS とデモアプリケーション

---

本パッケージのソースコードには TOPPERS/JSP カーネルとデモアプリケーションが含まれています。本プロジェクトにはデモアプリケーションが 9 つあり、それらは、メッセー

ジの編集、スクロール速度、振動デモ、傾斜デモ、マジック 8、パドルボール、Pong、マウスデモと音楽デモです。これらのアプリケーションはそれぞれ、EditMessage タスク、ScrollSpeed タスク、ShakeDemo タスク、TiltDemo タスク、Magic8 タスク、PaddleBall タスク、Pong タスク、Mouse タスクと Music タスクに相当する別々のタスクとして実装されています。

デモアプリケーションのタスクの初期化は、以下のような kernel\_cfg.c に置かれています。

```
const TINIB_kernel_tinib_table[TNUM_TSKID] = {
    {0x00u | 0x02u, (VP_INT)(0)      , (FP)(main_task)      , INT_PRIORITY(5), __TROUND_STK_UNIT(256),
    __stack_MAIN_TASK, TA_NULL, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 1), (FP)(EditMessage_task) , INT_PRIORITY(10),
    __TROUND_STK_UNIT(128), __stack_EDITMESSAGE_TASK , 0, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 2), (FP)(ScrollSpeed_task) , INT_PRIORITY(10),
    __TROUND_STK_UNIT(128), __stack_SCROLLSPEED_TASK , 0, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 3), (FP)(ShakeDemo_task)   , INT_PRIORITY(10),
    __TROUND_STK_UNIT(128), __stack_SHAKEDEMO_TASK   , 0, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 4), (FP)(TiltDemo_task)    , INT_PRIORITY(10), __TROUND_STK_UNIT(128),
    __stack_TILTDemo_TASK , 0, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 5), (FP)(Magic8_task)      , INT_PRIORITY(10), __TROUND_STK_UNIT(128),
    __stack_MAGIC8_TASK   , 0, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 6), (FP)(PaddleBall_task) , INT_PRIORITY(10), __TROUND_STK_UNIT(128),
    __stack_PADDLEBALL_TASK , 0, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 7), (FP)(Pong_task)        , INT_PRIORITY(10), __TROUND_STK_UNIT(128),
    __stack_PONG_TASK      , 0, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 8), (FP)(Mouse_task)       , INT_PRIORITY(10), __TROUND_STK_UNIT(128),
    __stack_MOUSE_TASK     , 0, (FP)(NULL)},
    {0x00u      , (VP_INT)(( VP_INT ) 8), (FP)(Music_task)       , INT_PRIORITY(10), __TROUND_STK_UNIT(128),
    __stack_MUSIC_TASK     , 0, (FP)(NULL)}
};
```

また、main タスク ( VP\_INT exinf ) という名のメインタスクは、ユーザからのインプットキーによってタスク選択が行われる sample1.c に置かれています。

```
switch(key){
    /* Edit Message Demo */
    case 1:
        act_tsk(EDITMESSAGE_TASK);
        ext_tsk();
        break;
    /* Scroll Speed */
    case 2:
        act_tsk(SCROLLSPEED_TASK);
```

```
    ext_tsk();
    break;
/* Shake Demo */
case 3:
    act_tsk(SHAKEDEMO_TASK);
    ext_tsk();
    break;
/* Tilt Demo */
case 4:
    act_tsk(TILTDEMO_TASK);
    ext_tsk();
    break;
/* Magic 8 Demo */
case 5:
    act_tsk(MAGIC8_TASK);
    ext_tsk();
    break;
/* Paddle Ball Demo */
case 6:
    act_tsk(PADDLEBALL_TASK);
    ext_tsk();
    break;
/* Pong Demo */
case 7:
    act_tsk(PONG_TASK);
    ext_tsk();
    break;
case 8:
default:
    goto re_select;
    break;
}
```

本パッケージのソースコードは、MCF51JM128用に構成されており、次の手順でデモアプリケーションをビルド・動作させることができます。

- ステップ 1 : 「Make」 ボタンをクリックして.s19 ファイルをビルドします。
- ステップ 2 : バッジにデモアプリケーションをロードします。4.3 節を参照してください。



デモアプリケーションの使用の詳細については、JM バッジボードユーザマニュアルの第 2 節、参照資料をご覧ください。

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.  
© Freescale Semiconductor, Inc. 2008. All rights reserved.



Freescale およびFreescale のロゴマークは、フリースケール社の商標です。  
文中に記載されている他社の製品名、サービス名等は、それぞれ各社の商標  
です。

© Freescale Semiconductor, Inc. 2008. All rights reserved.