＝ TOPPERS Automotive Kernel ユーザズマニュアル＝
（SH-2E プロセッサ ターゲット依存部）
（Release 1.0 対応）

1. SH7058F ターゲット依存部の概要

1.1 ターゲットプロセッサとシステム

ターゲットプロセッサ（ＣＰＵ）として、SH-2E コアを内蔵した SH7058F をサポートした。
内部リソースは、システムタイマとして、CMTチャネル０ を使用した。

他に、サンプルプログラムの動作用として、以下を割当てた。
- ・コマンド送受信用
  SCI1
- ・サンプルカウンタ用
  CMT(コンペアマッチタイマ)チャネル１
- ・ハードウェアタイマ ISR1 用
  ATU-II(アドバンストタイマユニット) ダウンカウンタ８Ａ
- ・ハードウェアタイマ ISR2 用
  ATU-II ダウンカウンタ８Ｅ

ターゲットシステム（ＳＹＳ）は，株式会社コンピューテックスの SH7058Fを搭載した評価用ボード 「SH7058F EVA BOD」をサポートした。

1.2 開発環境と実行環境

開発環境には，(株)ルネサステクノロジの開発環境(HEW ver.4)を用いる.
実行時は、マイコン内蔵のフラッシュメモリに書き込んで実行する。
High-performance Embedded Workshop Version 4.04.01.001
SuperH RISC engine Standard Toolchain (V.9.0.3.0)

2. SH-2E ターゲット依存部の機能

2.1 リソース
SH-2E ターゲット依存部では，次のプロセッサの内部リソースを使用する。
・CMT(コンペアマッチタイマ) チャネル0, チャネル1
・SCI1 (57600bps,8bit,NonParity,1stop)
・ATU-II(アドバンストタイマユニット) チャネル８ カウンタＡ，Ｅ

ターゲットシステムには、7segLED が取り付けられているが、使用しない。

2.2 割込み
2.2.1 割込み優先度
TOPPERS Automotive Kernelでは、プロセッサの割り込み制御として、割込み全禁止・許可機能と割込みマスクレベルでの禁止・許可機能の両方を持つことを想定している。しかし、SH-2E では、割込みマスクレベル機能しか持たない。そのため、割込み全禁止・許可に対しては、割込みマスクレベルを最大に上げることで、実現する。

2.2.2 割込み関連のカーネル内部関数
割込み全禁止は、カーネル内部の lock_cpu() が使用している。lock_cpu() はカーネルリソースの更新時に使用するため、この内部で、割込みマスクレベルを変更(set_ipl())することがある。
よって、lock_cpu() 中に割込みマスクレベル変更要求が来た場合には、変数に値を保持し、unlock_cpu() 時に割込みマスクレベルとして、設定する。

2.3 基本タスクでのスタック共通化
2.3.1 スタック共通化の必要性
本サンプルでは、基本タスクでのスタックを共通化する処理を組み込んでいる。
（詳細は、別紙「基本タスクスタック共通化検討」を参照）
TOPPERS Automotive Kernelでは、割込みハンドラの入口処理にて、タスクスタックから割込みスタックへの切り替えを行ため、最小限のレジスタを退避する。
SH2 系プロセッサでは、割込み処理では、より上位の割り込みを許可する仕様であるため、レジスタの退避中に上位の割込みが発生した場合、多重にスタックへレジスタを退避することとなる。
つまり、各タスクで割込みレベル数分のRAMを確保する必要がある。
これを回避するため、WAITING 状態が無い基本タスクで共通スタックを使用する。共通スタックは割込みスタックと兼用とする。

効果としては、同時に実行することはない同一優先度のタスクにおいても、スタックを個別に確保する必要がないため、さらにメモリ占有量を削減できる。

2.3.2 スタック共通化での注意点
OILの設定で、タスク毎にスタックサイズを指定するが、基本タスクか拡張タスクかの識別には、イベントの使用有無で行い、基本タスクには、４を指定する。
しかし、誤って、拡張タスクに４を指定した場合には、他のタスクのスタック破壊に到るため、注意が必要となる。

3. サンプルプロジェクトのビルド方法

サンプルプロジェクトは、３つのプロジェクトで構成してある。
・libkernel
  カーネル共通部分をビルドして、ライブラリ化するプロジェクト。
・sample
  サンプルプログラムと EVA7058用システムライブラリをビルドし、
  実行形式にするためのプロジェクト。
・simDebug
  サンプルプログラムと HEW のシミュレーションデバッガ用システム
  ライブラリをビルドし、デバッガ環境で実行するためのプロジェクト。

libkernel プロジェクトでビルドすると debug フォルダ下に
libkernel.rel が作成される。

sample プロジェクトでビルドすると、SG フォルダ下の sample1.oil を
参照し、SG を起動する。
その後、コンパイルし、libkernel.rel と結合する。

simDebug プロジェクトは sample と同等であるが、シミュレーション
デバッガ用の I/Oシミュレーションを使用するための処理を組み込んでいる。

4. シミュレーションデバッガの使用法

simDebug プロジェクトでビルドした場合、以下で操作する。
起動すると、I/Oシミュレーションの画面にメッセージが出力される。
実行したいコマンドをメモ帳などからコピーし、I/O画面でペーストする。
その後、RxD ボタンをクリックし、受信割り込みを発生させる。
HwInt1, HwInt2, SampleCnt は、適宜、割り込みを発生させる際に、
クリックする。

なお、simDebug プロジェクトでは、シミュレータの実行速度の都合上、
メインタスク起動用タイマの間隔を100ms から 10ms へ変更している。

5. 独自プロジェクトのビルド方法

サンプルプロジェクトでは、オプション類を HEW の標準から変更している。
変更箇所は、以下のとおり。
(1) インライン関数展開をコンパイラの最適化オプションで指定する。
(2) インラインアセンブラを有効にするため、コンパイラの出力ファイルを
    *.obj から *.src へ変更する。
また、多数のインクルードディレクトリを指定しているため、プロジェクトを
新規に構築するよりも、sample の環境を引き継いだ別のプロジェクトを作成
することが効率的である。
それには、HEW の機能である「プロジェクトテンプレートに変換」してから、
新規プロジェクトを作成するとよい。
詳細は、Help Library のクイックスタートガイドの下記の記述を参照のこと。
    TN 0014: 独自のプロジェクトジェネレータを作成するには?