ルネサス統合開発環境上で TOPPERSカーネルを使ってみよう！

（1） TOPPERS/ASPカーネルの High-performance Embedded Workshop 対応について

TOPPERSプロジェクト
宮城県産業技術総合センター
今井和彦

はじめに

- TOPPERS/ASPカーネル
  - 最新バージョンはGNUツールチェーンを用いて開発
- 本発表：他の処理系に移植する際の勘所をご紹介
  - 移植元：GNUツールチェーン
  - 移植先：ルネサス High-performance Embedded Workshop
    （以下、HEWと表記）
- 処理系がネックになって、TOPPERSカーネルの採用を見送っていた方は、これを機会にぜひご検討下さい。

ASPカーネルでサポートしている GNU以外の処理系

- SH2A
  - HEW

←今回、ご紹介する例

- M32C
  - HEW
- M16C
  - HEW
  - TM

処理系の内訳

	GNU開発環境	ルネサス開発環境
Cコンパイラ	gcc	shc
アセンブラ	gas	asmsh
リンカ	GNU ld	optlnk
全体のビルド管理	GNU make	HEW

移植の際のチェックポイント

1.「処理系の拡張機能（or 方言）」に起因する事項

(a) Cコンパイラ

- intサイズ
- コーリング・コンベンション
- インライン・アセンブラ
- 最適化問題

(b) アセンブラ

- プリプロセスの表記方法

(c) リンカ

2.「ビルド全体の監理方法」に起因する事項

– システム・コンフィギュレーション

- プロジェクトの分割
- 外部ツールとの連携

作業内容

時間の都合上、今回は
(※) の項目のみ解説します。

- intサイズに関連する定義
- コントロール・レジスタへのアクセス
- システム・コンフィギュレーション (※)
  - プロジェクト分割
  - 外部ツールとの連携
  - 割込みの出入口処理
- オフセットファイルの生成
  - アセンブラ・ルーチンからC言語構造体にアクセス
- クリティカル・セクション (※)
  - 最適化問題
- 非依存部のインライン関数
- アセンブラ用のインクルードファイル (※)

システム・コンフィギュレーションとは？

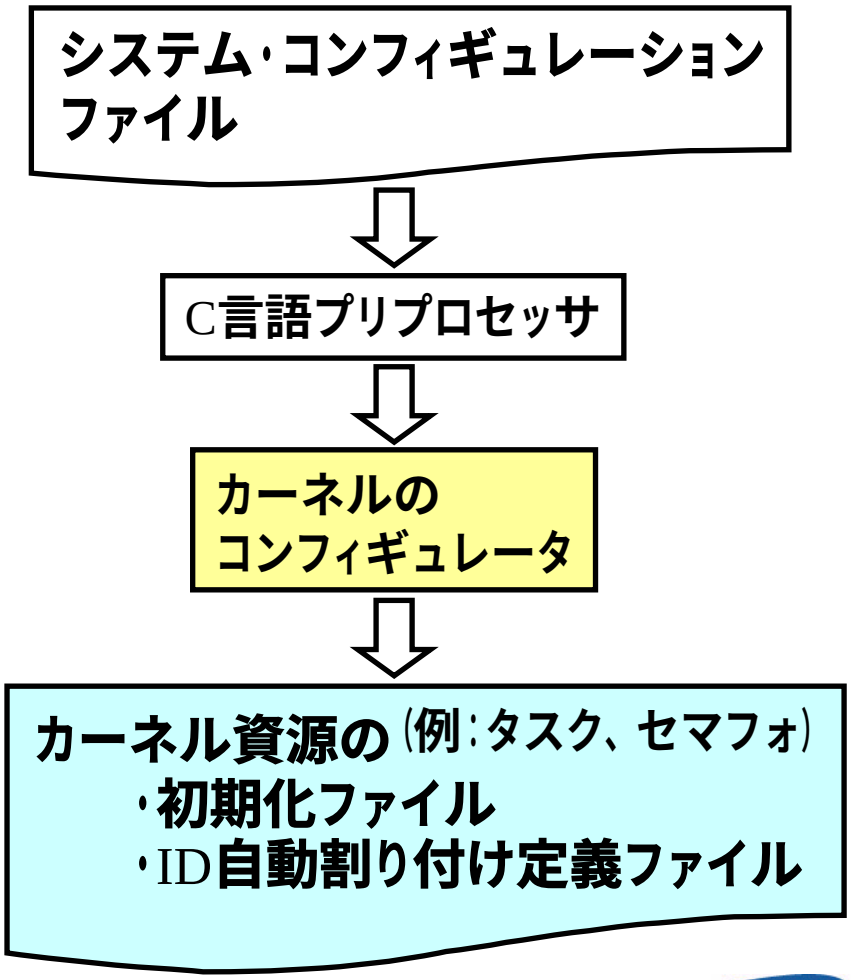
（カーネルやシステムサービスが管理する）

オブジェクト（例：タスク）の
- 生成情報
- 初期状態

から

カーネル（or システムサービス）の
- 構成情報
- 初期化情報

を含むファイルを生成する作業



ASPカーネルでは、
μITRON4.0仕様からコンフィギュレータの処理モデルを**大幅に変更**

機種依存で生成すべき内容

アプリケーション依存の情報はカーネル内で決め打ちにできない。

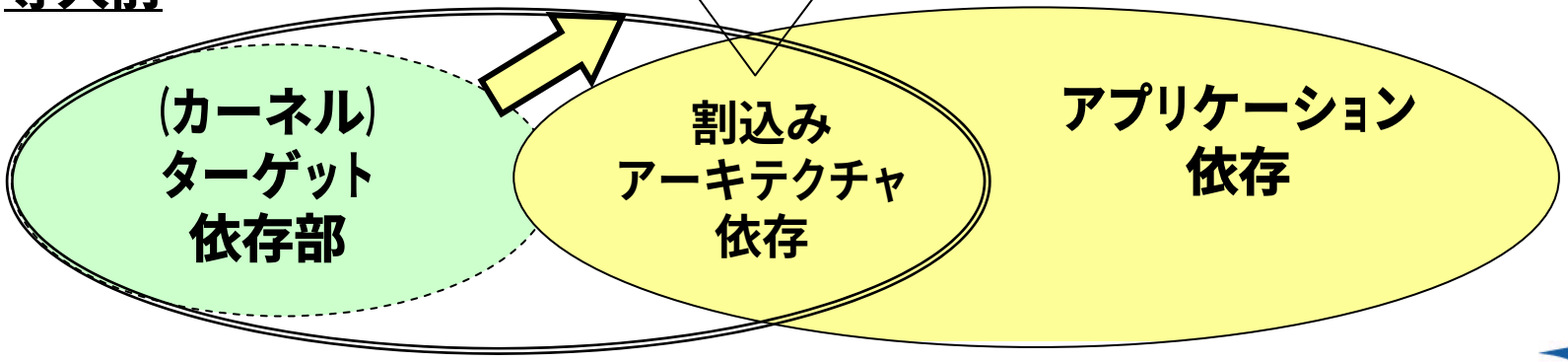
- 使用する割込み番号
- 割込みハンドラ／割込みサービスルーチン(の先頭番地)
- 割込みライン属性
  - 割込み優先度
  - トリガ方式：エッジトリガ／レベルトリガ

TOPPERS標準
割込み処理モデル

導入前

導入後

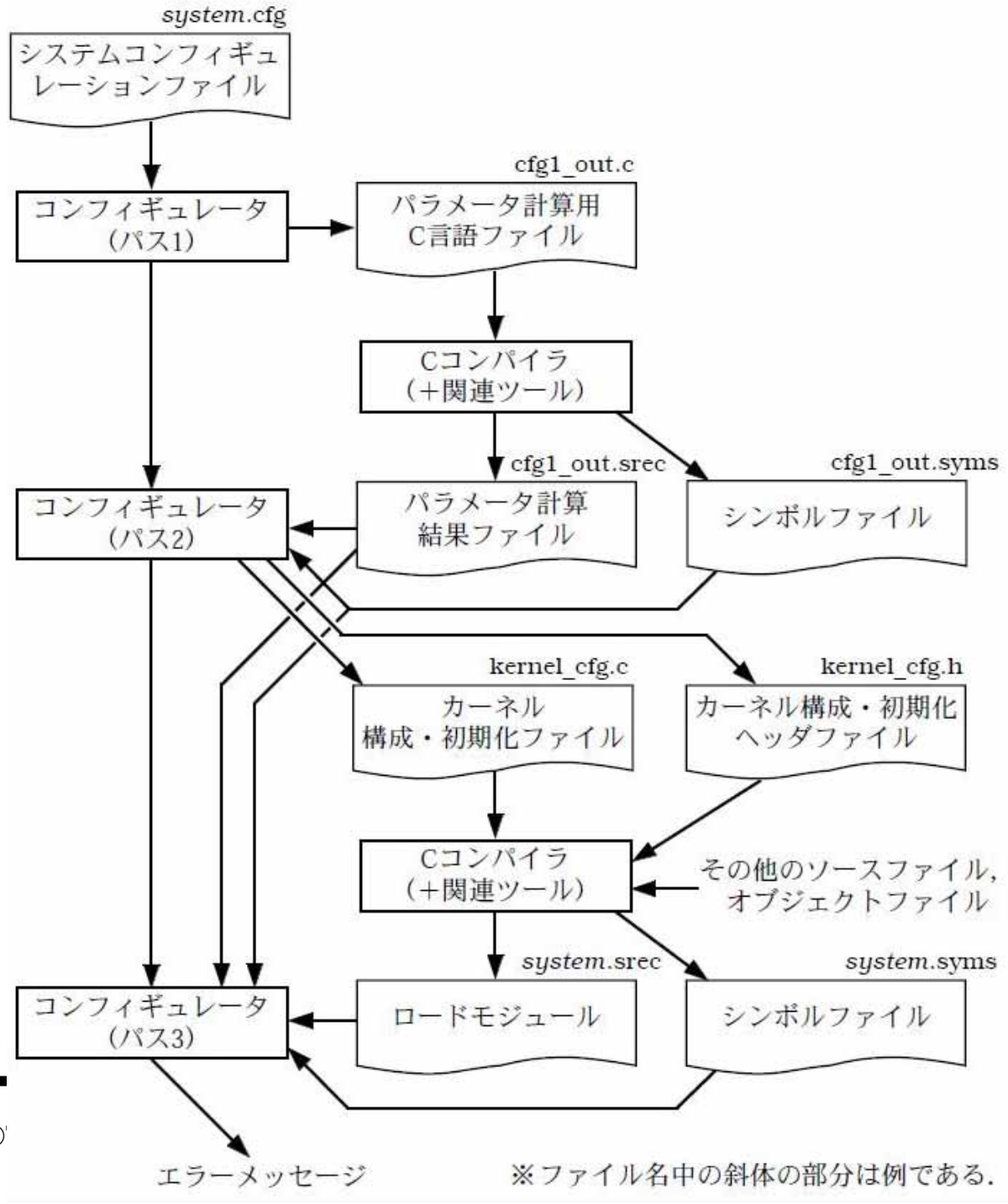
→コンフィギュレーション時にパラメータの値を評価する必要がある。



システム・コンフィギュレーションの手順

(静的APIの) パラメータの評価結果は処理系に依存 (コンパイラ、リンカ)

→途中で処理系を呼び出す必要があり、若干、複雑



システム・コンフィギュレーション(2)

- パス1
  - Cコンパイラを用いて、静的APIの整数定数式パラメータの値を決定する。
    - オブジェクト番号／属性／サイズ／数
    - 機能コード
    - 優先度
  - コンパイル後でないと決定できない値もある(sizeof演算子の評価結果など)
  - アドレス配置には依存しない。
- パス2

  上記の結果を用いて、カーネルの構成・初期化ファイルkernel_cfg.c, kernel_cfg.hを生成
  →この段階でアプリケーションを含むシステム全体のビルドに必要なソースファイルがそろう。
- パス3
  - 最終的な実行ファイルに対して、静的APIの一般定数式パラメータの妥当性をチェックする。

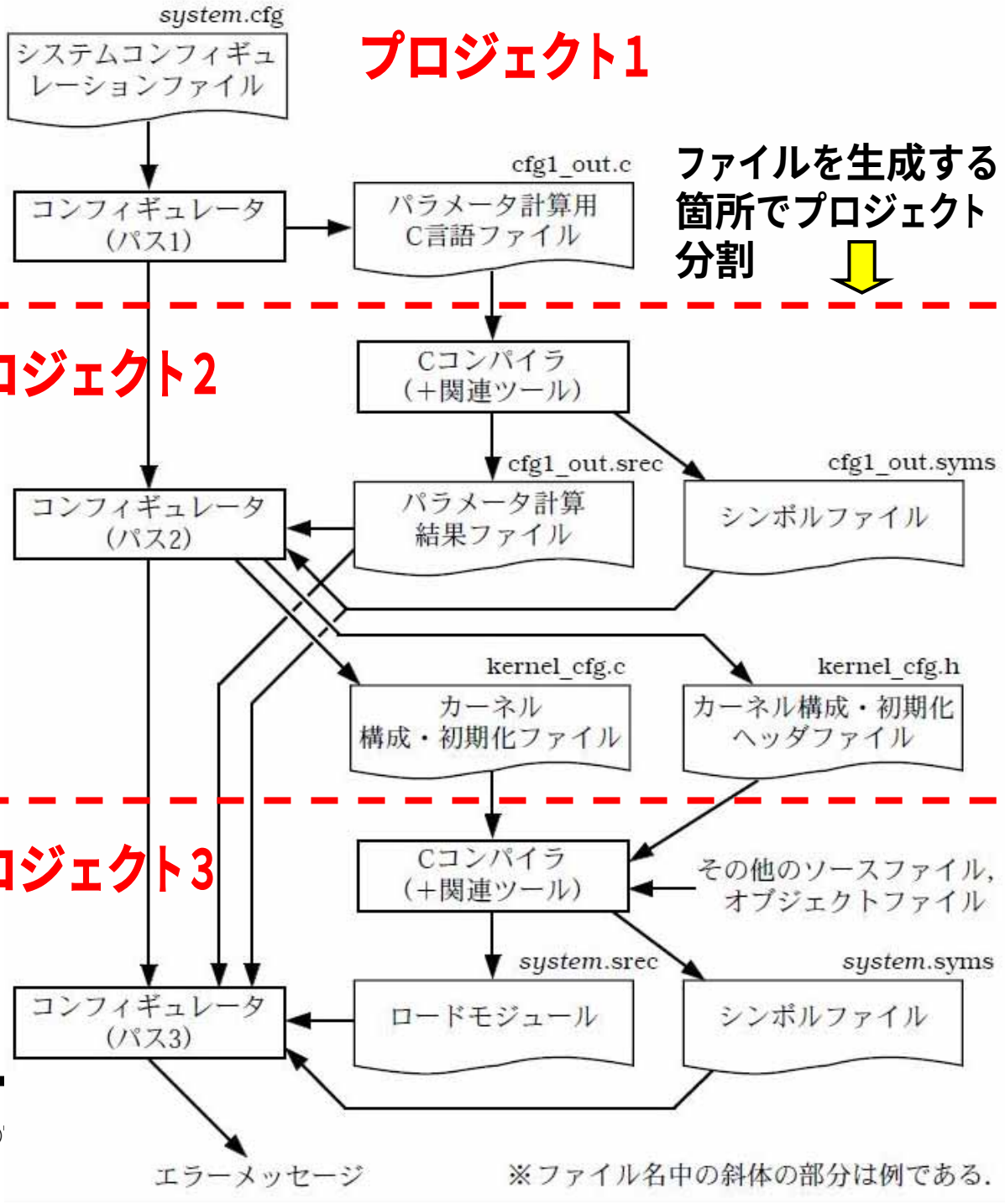
  ・アドレスを指定する可能性のあるパラメータ
  →リンク後でないと値が決まらない。
    - 処理単位のエントリ番地
    - メモリ領域の先頭番地
    - 拡張情報など

システム・コンフィギュレーションに関連して必要な修正

(1) プロジェクト分割
  - カスタム・フェーズ
(2) 外部ツールとの連携
(3) 割込みの出入口処理
  - インライン・アセンブラとシンボル参照

システム・コンフィギュレーションに関連して必要な修正 (1)

- プロジェクト分割
  - HEWでは、プロジェクトのビルドに先立ち、(インクルードファイルを含む)ソースファイルがすべてそろっているかチェックする。(ソースファイルがそろっていなければ、即ビルドエラー)
  - HEWでは、カスタムフェーズでソースファイルを自動生成する処理は、別プロジェクトとして分割する必要がある。



システム・コンフィギュレーションに関連して必要な修正（2）－1

・外部ツールとの連携
  ・コンフィギュレータ本体
  ・シンボルテーブルの変換shc_map_nm.exe
    パス2でシンボルテーブルをGNU nmコマンド互換の形式に変換する。

・相対パス問題
  ワークスペース・ディレクトリ以下にない外部ツールを相対パスで指定できない。
    →絶対パスでは、他のディレクトリ／パソコンに移すとビルドが通らない。

解決策：
  ワークスペース配下にバッチファイル(xxx.bat) を用意し、このバッチファイルから外部ツールを呼び出す。

システム・コンフィギュレーションに関連して必要な修正（2）－2

エラー検出問題

外部ツール（コンフィギュレータ）でエラーが発生しても、HEWでそれを認識できない場合がある。（素通りして、後のビルドフェーズでエラーが顕在化）

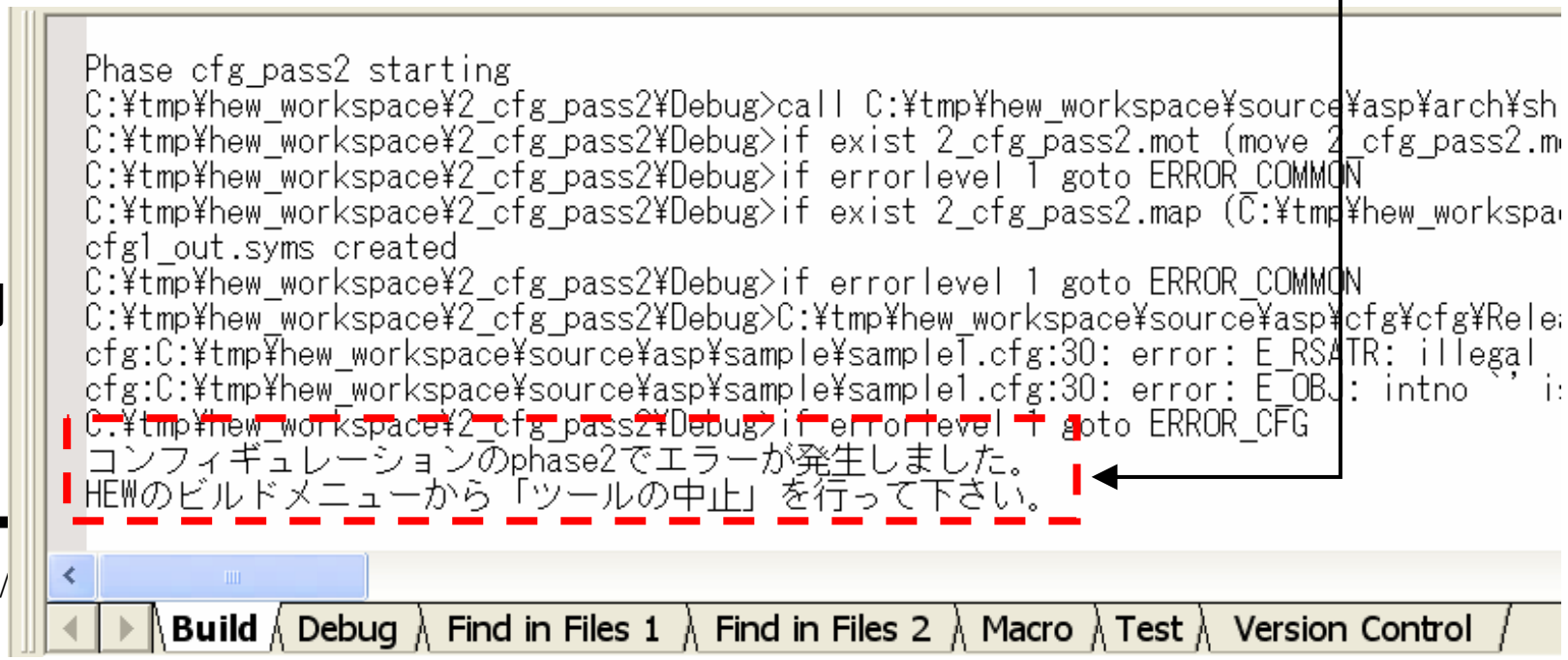
解決策：

外部ツールでエラーが発生したら、前述のバッチファイルにより

1．ユーザーにビルドを手動で停止させるようメッセージを表示

2．ビルドが先に進まないように、バッチファイル内で無限ループ

HEWのアウトプットウィンドウでの表示例



システム・コンフィギュレーションに関連して必要な修正（3）－1

カーネルの初期化・構成ファイルkernel_cfg.c**のターゲット依存部**

1．割込みの出入口処理

グローバルシンボルの参照が必要

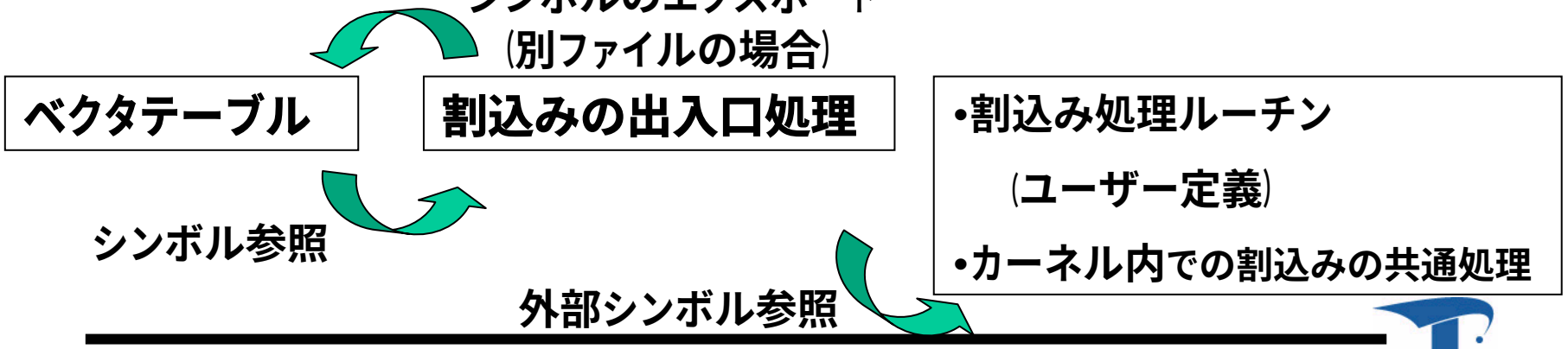
・ユーザー定義の割込み処理ルーチン

・カーネル内での割込みの共通処理への分岐

割込み要因に拠らない部分

2．ベクタテーブル

出入口処理をラベルとして参照しているので、同様



システム・コンフィギュレーションに関連して必要な修正（3）－2 （タイマ割込みの例）

GNU開発環境版 (kernel_cfg.c)

```
asm(".text                                  ¥n"
"               .section .vector_entry,¥"xa¥"       ¥n"
"    .global __kernel_target_timer_handler_140¥n"
"    .align 2                                       ¥n"
"__kernel_target_timer_handler_140:         ¥n"

(中略)
"               .align 2                            ¥n"
                /* 割込み禁止用マスク */
"_mask_ipm_target_timer_handler_140:        ¥n"
                /* ipm以外のビットはゼロで良い */
"               .long   __kernel_asm_mask_ipm       ¥n"
"_c_routine_target_timer_handler_140:       ¥n"
                /* C言語ルーチンの先頭アドレス */
"               .long   _target_timer_handler       ¥n"
                /* cpu_support.S内の分岐先アドレス */
"_common_routine_target_timer_handler_140: n"
"               .long   __kernel_int_entry ¥n"
                /* 割込み番号                  */
"_inhno_target_timer_handler_140: ¥n"
"               .long   140 ¥n"
```

シンボルのエクスポート

ルネサス開発環境版 (kernel_cfg_asm.src)

```
                .SECTION vector_entry, CODE, ALIGN=4
                .IMPORT _target_timer_handler
                .GLOBAL __kernel_target_timer_handler_140
__kernel_target_timer_handler_140:

(中略)
                ; C言語ルーチンの先頭アドレス
_c_routine_target_timer_handler_140:
                .DATA.L  _target_timer_handler
                ; prc_support.src内の分岐先アドレス
_common_routine_target_timer_handler_140:
                .DATA.L  __kernel_int_entry
_inhno_target_timer_handler_140:                ; 割込み番号
                .DATA.L  140
```

外部シンボルの参照

システム・コンフィギュレーションに関連して必要な修正（3）－3

・割込みの出入口処理

割込みの出入口処理では、外部とのシンボルのインポート／エクスポートが必要になる。

・ユーザー記述部分　と

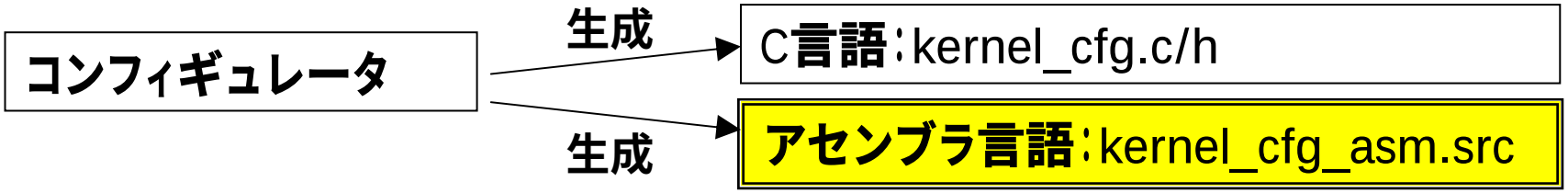
・コンフィギュレーション時の自動生成ファイル

の結合

GNU版では、インライン・アセンブラを用いて記述しているが、shcではインライン・アセンブラ内で外部シンボルを用いることができない。

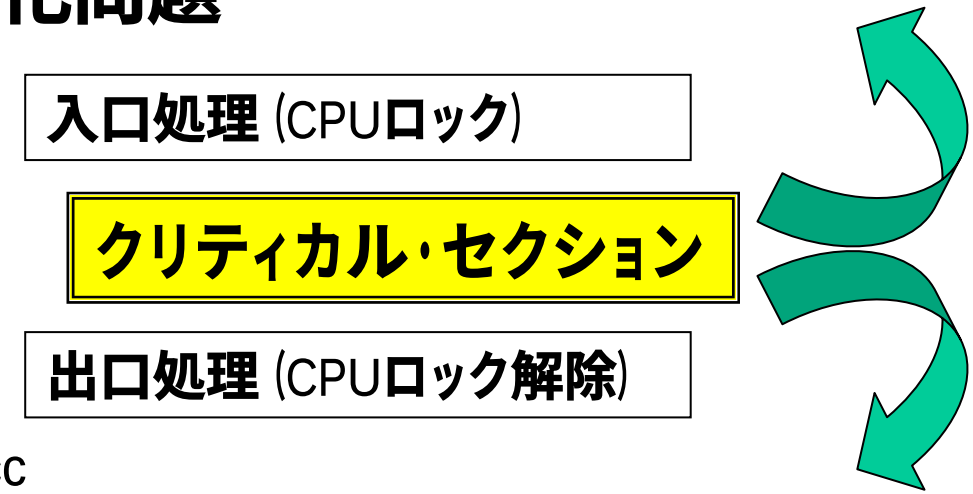
→割込みの出入口処理を、アセンブラ・ソースファイルとして分離

テンプレートファイルにて、コンフィギュレータの出力先をCソースファイル kernel_cfg.cから**アセンブラ・ファイルkernel_cfg_asm.srcに変更**



クリティカル・セクション

- 最適化問題



コンパイラの最適化により、クリティカル・セクションの出入口処理をまたいで、処理の順番が交換される場合がある！

- gcc

  gccでは、Asm(“”:::“memory”)）;を挿入するなどの処置が必要

- shc

  shcでは、組み込み関数（具体的にはset_sr())をまたいで、処理の順番を入れ替えるような最適化は行わないため、このような心配はない。

※詳細は、ポーティングガイドasp/doc/porting.txt
「1.6 クリティカルセクションの出入処理の実現に関する制約」を参照

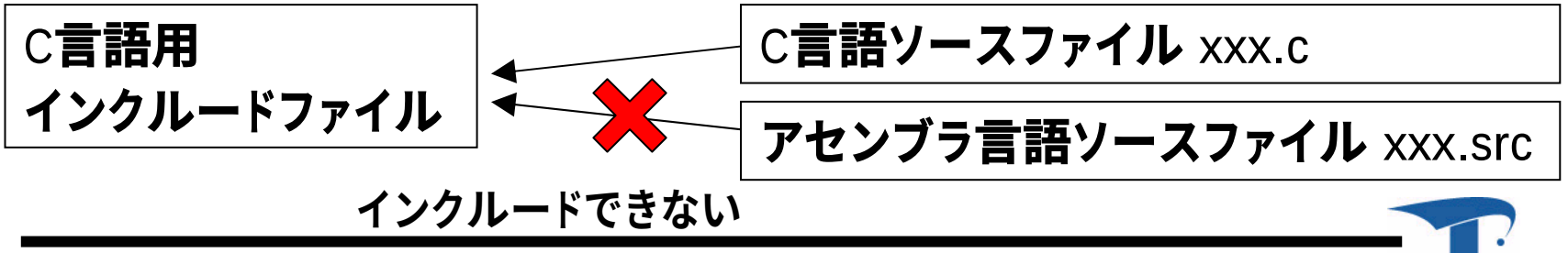
アセンブラ用のインクルードファイル

GNU開発環境では、Cソースファイルとアセンブラ・ソースファイルでプリプロセッサのディレクティブが共通なので、インクルードファイルを共有できる。
ルネサス開発環境では、インクルードファイルを共有できないので、アセンブラ用のインクルードファイルを別途、用意する。

- ・レジスタ定義
- ・クロック周波数の定義
- ・シンボル・リネーム xxxx_rename.h

例：kernel_rename.hの代替

```
_dispatch:        .DEFINE     "__kernel_dispatch"
```



その他

- アセンブラ・ソースコード
  - 外部シンボルのインポートを明示的に記述する。
  - 例：asp/arch/sh12a_shc/prc_support.src

    .IMPORT _p_runtsk
- 文字コード／改行コード
  - GNU開発環境：EUC＋LF
  - ルネサス開発環境：SJIS＋CR/LF

    処理系自体はEUCコードにも対応しているが、ユーザーの利便性を考慮して、SJISコードを使用
- ディレクトリ構成
  - ソースツリー全体がワークスペース配下にある方が扱いやすい。
  - 配布パッケージ（簡易パッケージ）は、そのようにディレクトリを構成

付録

Cコンパイラの比較（1）

	GNU gcc	ルネサス shc	ASPカーネルとの関係
intサイズ	両者とも同じ（32ビット）		
コーリング・コンベンション	両者とも同じ –r0〜r7を呼び出し側で保存 –引数：r4〜r7 –戻り値：r0		ディスパッチャの構造はそのままでOK

インライン関数	__inline__ void func(void)
	{
	.....
	}
		#pragma inline func	C99のinlineキーワード相当を想定
			inline void func(void) {.....}

Cコンパイラの比較 (2)

	GNU gcc	ルネサス shc	ASPカーネルとの関係
インライン・アセンブラ	__asm__ (“stc sr,%0” : "=r"(sr)); ・行単位で記述可能 ・C言語の変数とレジスタの対応を指定できる。 ・外部シンボルの参照も可	#pragma inline_asm func ・関数単位で記述 ・インライン・アセンブラ関数内部からは外部シンボルを参照不可	GNU版では、割込みの出入口処理をインライン・アセンブラで記述 →外部シンボルを参照している
呼出し元へリターンしない関数属性	__attribute__((__noreturn__))	なし	NoReturnマクロとして、ext_ker()でから呼び出されるtarget_exit()で使用

アセンブラの比較(1)

	GNU gas	ルネサス asmsh	ASPカーネル との関係
プリプロセッサ ディレクティブ	C言語のプリプロセッサ ディレクティブと共通		
–コメント	/* コメント */	; コメント	
–マクロ定義	#define N 10	N: .DEFINE "10"	
–条件付取り 込み	#ifdef DEBUG #else #endif	.AIFDEF DEBUG .AELSE .AENDI	
16進表記	0xff (C言語と同様)	H'FF	

アセンブラの比較(2)

	GNU gas	ルネサス asmsh	ASPカーネルとの関係
データ定義	.long 0x10	.DATA.L H'10	
グローバル・シンボル (外部への公開)	.global _dispatch	.GLOBAL _dispatch	
外部シンボルのインポート	省略可	.IMPORT _p_runtsk	
セクション名	.text (.data, .bss)	.SECTION P, CODE, ALIGN=4 (P, C, D, R, B)	

リンカの比較

	GNU ld	ルネサス optlink	ASPカーネルとの関係
未定義のシンボルの扱い (weak definition)	PROVIDE(_foo = 0); シンボル_fooが未定義であれば、値として、0を設定する。	-change=w=2310オプションで、左記のPROVIDE機能相当を実現	下記参照

ASPカーネルでは、下記の3つのシンボルで使用

- _hardware_init_hook：ターゲットシステム依存の初期化
- _software_init_hook：開発環境（特にライブラリ）に依存して必要な初期化処理
- _software_term_hook：開発環境（特にライブラリ）に依存して必要な終了処理

intサイズに関連する定義

C99互換の整数型を定義する。

- gccもshcも基本データ型(char/short/int/long)のビット幅が同じなので、gcc版の定義をそのまま流用できる。

asp/arch/shc/tool_stddef.h ←処理系依存(SH1〜SH4まで共通に使える)

```
typedef signed char         int8_t;     /* 符号付き8ビット整数 */
typedef unsigned char       uint8_t;    /* 符号無し8ビット整数 */

(中略)

typedef long                intptr_t;   /* ポインタを格納できる符号付き整数 */
typedef unsigned long       uintptr_t;  /* ポインタを格納できる符号無し整数 */
```

コントロール・レジスタへのアクセス

コントロール・レジスタ：プロセッサ内部のSR, VBR**など**
処理系が提供する組み込み関数（set_sr()**など）で対応できる。**

オフセットファイルの生成

アセンブラ・ルーチンからTCB構造体のメンバにアクセスするためのオフセット定義

現状、offset.inc内で決め打ちにしている。

```
TCB_p_tinib:            .DEFINE "(8)"
TCB_texptn:             .DEFINE "(H'14)"
TCB_sp:                 .DEFINE "(H'1c)"
TCB_pc:                 .DEFINE "(H'20)"
TINIB_exinf:            .DEFINE "(4)"
TINIB_task:             .DEFINE "(8)"

; バイトアクセス
TCB_enatex:             .DEFINE "(H'11)"
TCB_enatex_bit:         .DEFINE "(5)"
TCB_enatex_mask:        .DEFINE "(1 << TCB_enatex_bit)"
```


TOPPERS

非依存部のインライン関数

•gcc

```
__inline__ void func(void){}
```

•shc

```
#pragma inline func
```

ASPカーネルでは、C99のinlineキーワード相当の機能を想定している。
例えば、非依存部のt_syslog.hで定義されているインライン関数に対して、ターゲット依存部のみの修正で#pragma inlineを挿入する術がない。

例：asp/include/t_syslog.h 142行目

```
Inline void   ←#pragma inlineを挿入できない。
_syslog_0(uint_t prio, uint_t type)
{
        SYSLOG syslog;

        syslog.logtype = type;
        (void) syslog_wri_log(prio, &syslog);
}
```

要望

プロジェクトファイルをプロセッサ依存部とボード依存部に分離できると便利 (それぞれを順不同に編集したい)

- ビルド・オプション
- インクルード・ファイルへのサーチパス

GNU makeの場合

ボード依存のMakefileからプロセッサ依存Makefileをインクルード