TOPPERSカンファレンス2013
チュートリアルセッション

修正版

TOPPERSコンフィギュレータの進んだ使い方

2013年6月21日

高田 広章

NPO法人 TOPPERSプロジェクト 会長

名古屋大学 大学院情報科学研究科 教授
附属組込みシステム研究センター長

Email: hiro@ertl.jp　URL: http://www.ertl.jp/~hiro/

概要と目次

概要

- TOPPERS新世代カーネル向けのコンフィギュレータは，新たな静的APIを容易に作れる仕組みになっている
- ASPカーネルに対して新たな静的APIを追加する方法を中心に，コンフィギュレータの進んだ使い方を解説する

目次

- TOPPERS新世代カーネル向けコンフィギュレータとは？
- 静的APIを追加してみよう
- コンフィギュレータの仕組み
- 実行時間分布集計モジュールに対する静的APIを作る
- コンフィギュレータの歴史と今後
- より詳しく知るために－まとめに代えて

TOPPERS新世代カーネル向け コンフィギュレータとは？

静的OS (Static Operating System)

静的OSとは？

- 使用するOS資源（タスク，セマフォなど）を静的に（設計時に）定義するOS

  例）多くのμITRON仕様OS，TOPPERS OSのほとんど

  OSEK/VDX仕様OS，AUTOSAR OS仕様（OS資源を動的に生成するAPIが仕様に規定されていない）

- ! 専用システムであるという組込みシステムの特性を活かしたOS技術

静的OSの利点

- メモリ容量（特にRAM容量）を小さくできる
- システム起動時間を短縮できる
- システムの動作中にメモリ不足になることがない
- 静的な情報を使った最適化が可能

使用するOS資源の設定方法（コンフィギュレーション記述）

（1）コンフィギュレーション記述の言語を定め，そこからツールにより構成・初期化ファイルを生成する方法

- μITRON仕様／TOPPERS新世代カーネルの静的API
- OSEK/VDX仕様のOIL（OSEK Implementation Language）
- AUTOSAR仕様ではXMLのフォーマットを規定

→この生成ツールを「コンフィギュレータ」と呼ぶ（OSEK/VDX，AUTOSARでは「ジェネレータ」と呼ばれている）

（2）GUIベースのコンフィギュレーションツールを用意する方法

（3）構成・初期化ファイルを直接記述する方法（C言語の初期化文の形で記述するのが一般的）

- 静的な情報を使った最適化は難しい

静的API

- オブジェクトの生成情報や初期状態などを定義するために，システムコンフィギュレーションファイル中に記述するためのインタフェース

静的APIの文法

- C言語の関数呼出しと同様の文法で記述．ただし構造体（へのポインタ）は，各フィールドの値を { } で囲んで列挙
- サービスコールと同等の機能を持つ静的APIは，サービスコール名を大文字にした名称とし，パラメータも同一とする

静的APIの利点

- サービスコールと静的APIを個別に覚える必要がない
- 静的APIのパラメータに式を記述できる（制限あり）
- システムコンフィギュレーションファイル中に，条件ディレクティブ（#ifdef など）を記述できる

静的APIの例

```
CRE_TSK( ID tskid, { ATR tskatr, intptr_t exinf,
        TASK task, PRI itskpri, SIZE stksz, STK_T stk } );
```

対応するサービスコール

```
ER ercd = cre_tsk ( ID tskid, T_CTSK *pk_ctsk ) ;
```

静的APIの記述例

```
CRE_TSK( TASK1, { TA_ACT, 0,
        task_main, 10, STACK_SIZE, NULL } );
```

別のヘッダファイルで
値を定義しておく

コンフィギュレータが
スタック領域を割り付ける

コンフィギュレータがタスクIDを割り付ける

静的APIのパラメータ

- 静的APIのパラメータは，次の4種類に分類される

(a) オブジェクト識別名

- オブジェクトのID番号を指定
- オブジェクトの名称を表す単一の識別名のみを記述

(b) 整数定数式パラメータ

- オブジェクト属性や優先度など，整数値を指定
- プログラムが配置される番地に依存せずに値の決まる整数定数式を記述できる

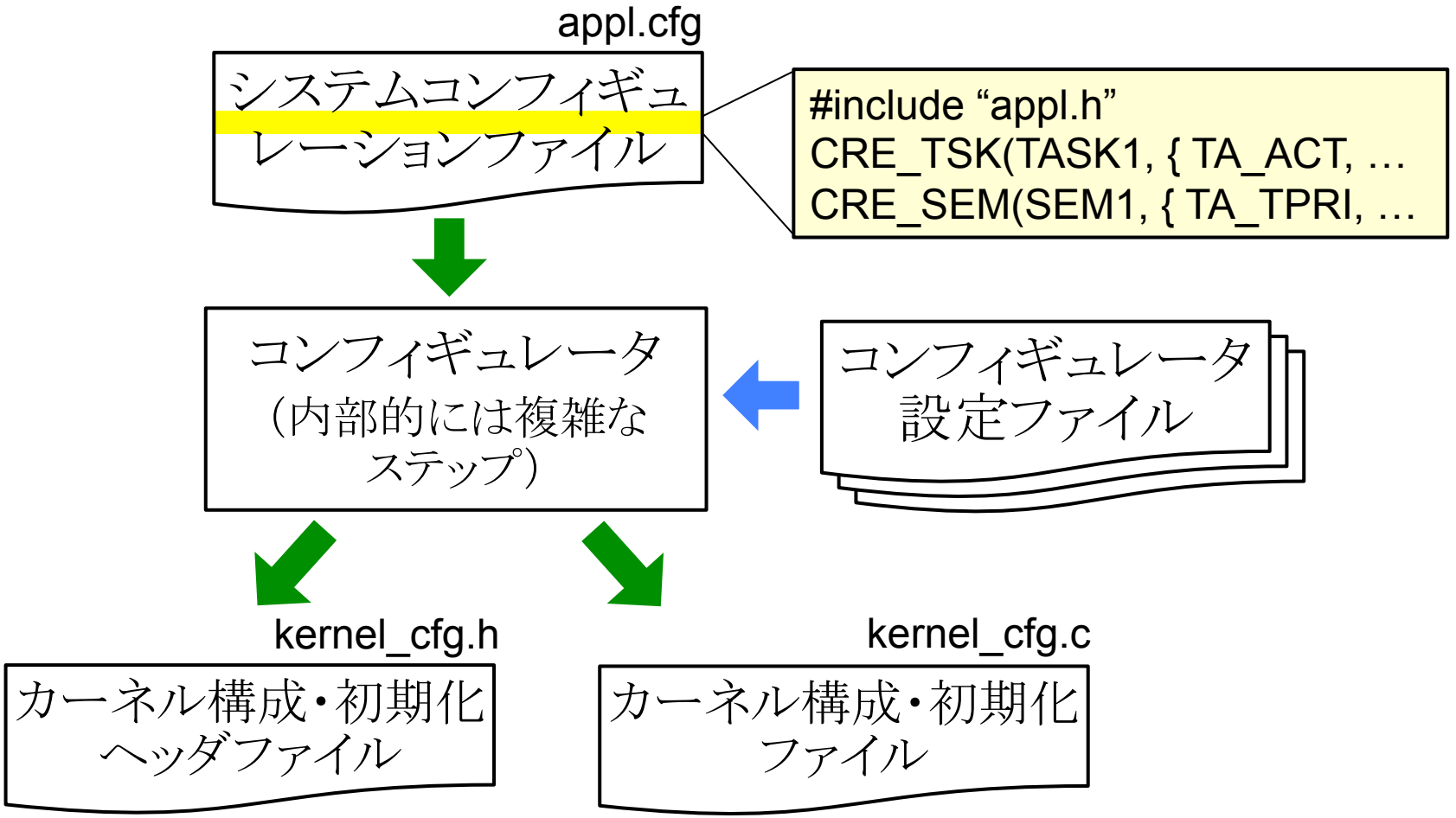
(c) 一般定数式パラメータ

- 処理単位のエントリ番地や拡張情報など，番地を指定する（可能性のある）パラメータ
- 任意の定数式を記述できる

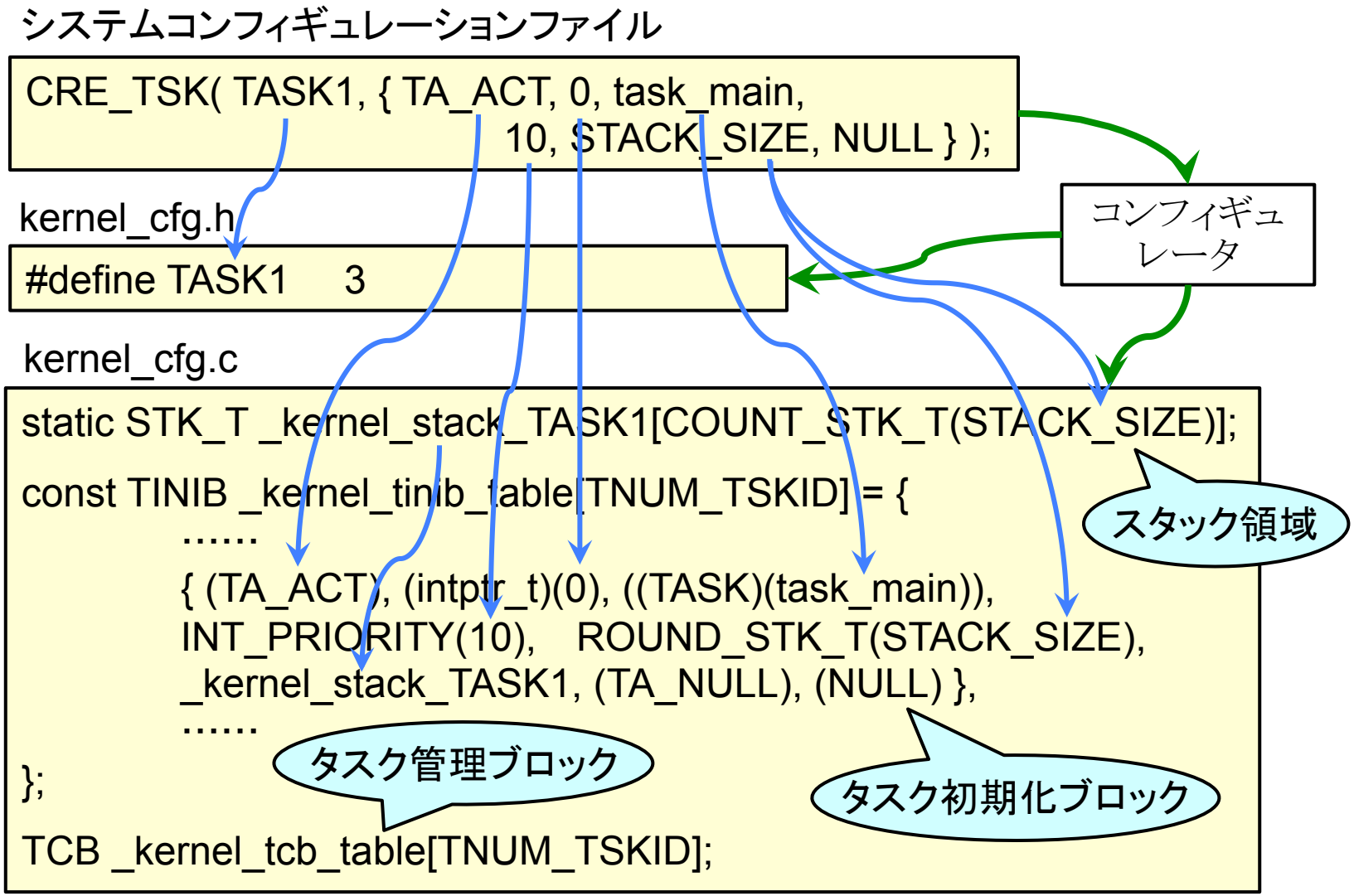
(d) 文字列パラメータ

コンフィギュレータの入出力

コンフィギュレータの入出力ファイル（ASPカーネルの場合）



コンフィギュレータの入出力の例（ASPカーネルの場合）



静的APIを追加してみよう

追加する静的API

- ASPカーネルに対して，プロセッサ依存の設定を行うための静的APIを追加してみる

追加する静的APIの仕様

- DEF_CCM(uint32_t cmode)
  - システムの初期化処理において，プロセッサのキャッシュモードを，cmodeに与えた値に設定する
  - cmodeに与える値は，以下のいずれかとする

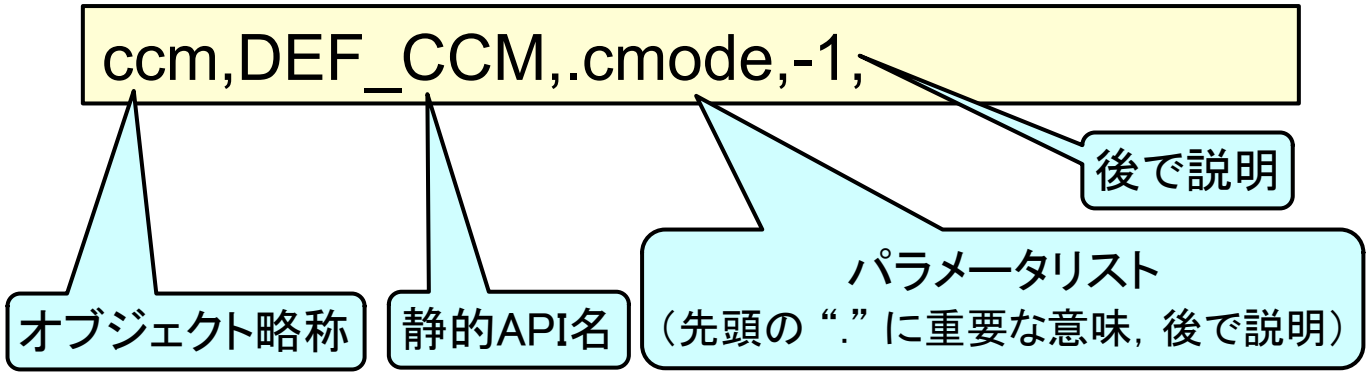
CACHE_OFF	0x0000U
CACHE_WTHROUGH	0x0001U
CACHE_WBACK	0x0002U

準備：定数のマクロ定義

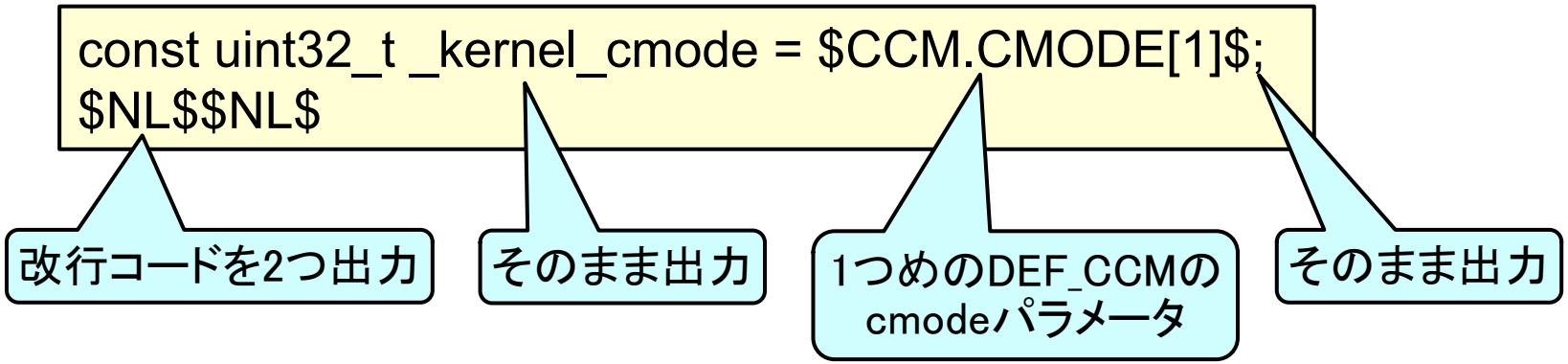
- 上記の定数のマクロ定義（#define）を，ヘッダファイル（例えば，target_kernel.h）に追加する

まずは最低限の修正で

静定APIテーブル（kernel/kernel_api.csvの最後）への追加



テンプレートファイル（kernel/kernel.tfの最後）への追加



！ 以上で修正完了

使ってみる(1)

システムコンフィギュレーションファイル

```
DEF_CCM(0x0001);
```

kernel_cfg.c

```
const uint32_t _kernel_cmode = 0x0001;
```

使ってみる(2)

システムコンフィギュレーションファイル

```
DEF_CCM(CACHE_WBACK);
```

kernel_cfg.c

```
const uint32_t _kernel_cmode = CACHE_WBACK;
```

- CACHE_WBACKのマクロ定義(#define)の追加を忘れると, コンフィギュレータでエラーになるので注意

テンプレートファイルの記述方法

- 基本的には，テンプレートファイルに記述された文字列がそのまま出力される
  - 前のページの「const uint32_t _kernel_cmode = 」「;」
- 「$」で囲まれた文字列は，マクロプロセッサによって評価され，その結果が出力される
  - 「$CCM.CMODE[1]$」は，配列「CCM.CMODE」の1番目の要素の意味で，1番目のDEF_CCMのcmodeパラメータの値が入る
- テンプレートファイルを読みやすくするために，改行と文頭のホワイトスペースは出力されない
  - 出力したい場合には，「NL」「SPC」「TAB」と記述する
- 行頭に「$」＋ホワイトスペースがある行は無視する(コメント)

とは言え，さすがにこれでは不十分(1)

システムコンフィギュレーションファイル

```
/* DEF_CCMがない */
```

kernel_cfg.c

```
const uint32_t _kernel_cmode = ;
```

- kernel_cfg.cがコンパイルエラーに…

とは言え，さすがにこれでは不十分(2)

システムコンフィギュレーションファイル

```
DEF_CCM(0x0003);
```

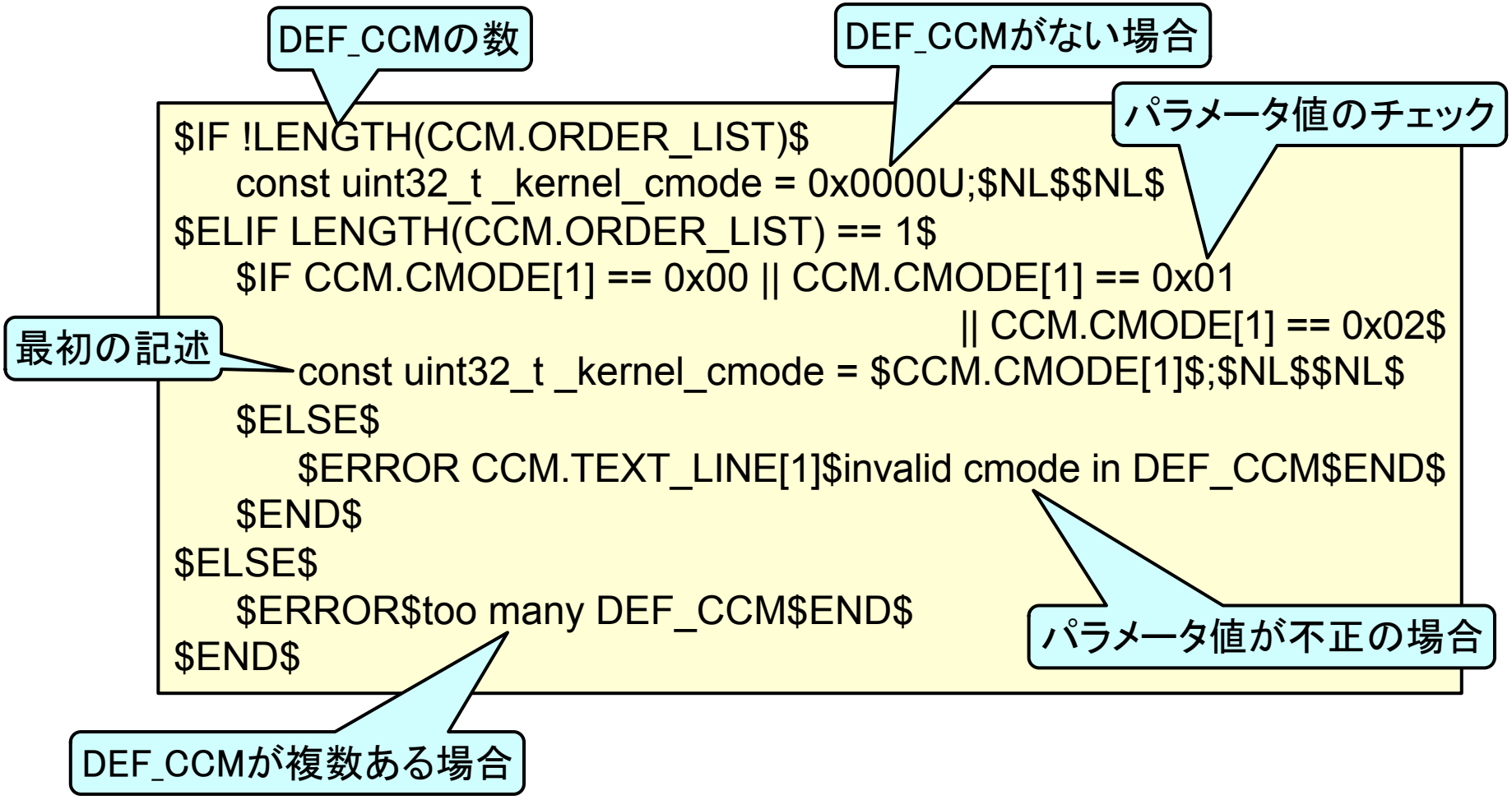
kernel_cfg.c

```
const uint32_t _kernel_cmode = 0x0003;
```

- 0x0003は指定できない値なのに…

エラーチェックを入れてみる

テンプレートファイル（kernel/kernel.tfの最後）の修正



使ってみる(1)

システムコンフィギュレーションファイル

```
/* DEF_CCMがない */
```

kernel_cfg.c

```
const uint32_t _kernel_cmode = 0x0000U;
```

使ってみる(2)

システムコンフィギュレーションファイル

```
DEF_CCM(0x0001);
DEF_CCM(0x0002);
```

```
cfg: error: too many DEF_CCM
```

使ってみる(3)

システムコンフィギュレーションファイル

```
DEF_CCM(0x0003);
```

```
cfg:sample1.cfg:27: error: invalid cmode in DEF_CCM
```

テンプレートファイルの記述方法

- 「$IF ＜式＞$」「$ELIF ＜式＞$」「$ELSE$」「END」で条件文が記述できる
  - 式の文法はC言語と類似
- 「CCM.ORDER_LIST」には，DEF_CCMの通し番号のリストが入る
  - 「LENGTH(CCM.ORDER_LIST)」は，システムコンフィギュレーションファイル中のDEF_CCMの数になる
- 「$ERROR ＜エラー行＞$」「END」で，囲まれた文字列がエラーメッセージとして出力される
  - エラーが1つでもあると，コンフィギュレーションに失敗する（テンプレートファイルは最後まで処理される）
- 「CCM.TEXT_LINE[*i*]」には，*i*番目のDEF_CCMの記述されているファイル名と行番号が入る

式による記述と数値の関係

パラメータに式を書いてみる

「+ 0」には意味はありません

システムコンフィギュレーションファイル

```
DEF_CCM(CACHE_WBACK + 0);
```

kernel_cfg.c

```
const uint32_t _kernel_cmode = CACHE_WBACK + 0;
```

この振舞いは不思議ではないですか？

- 「CCM.CMODE[1]」には，何が入っているのでしょうか？
- 「… || CCM.CMODE[1] == 0x02」というエラーチェックは通過しているので，2という数値が入っているはず
- 上の kernel_cfg.c には，式で出力されている

！ kernel_cfg.c を読みやすくするために，式にしている

テンプレートファイル変数に入る値

- テンプレートファイル変数には，文字列と数値の両方を同時に入れることができる（片方しか入っていない場合も）
  - 「CCM.CMODE[1]」には，“CACHE_WBACK + 0”という文字列と，2という数値の両方が入っている
  - 変数を出力する場合には，文字列が出力される
  - 変数に対して数値演算を行う場合には，数値が使われる
- 数値を出力したい場合には，数値演算すればよい

テンプレートファイル

```
const uint32_t _kernel_cmode = $+CCM.CMODE[1]$;
$NL$$NL$
```

この「+」により，数値の形で出力される

テンプレートファイル中での定数名

さきほどのテンプレートファイルの記述

```
……
    $IF CCM.CMODE[1] == 0x00 || CCM.CMODE[1] == 0x01
                                    || CCM.CMODE[1] == 0x02$
……
```

ここを下のように定数名で記述できないのか？

望ましいテンプレートファイルの記述

```
……
    $IF CCM.CMODE[1] == CACHE_OFF
                        || CCM.CMODE[1] == CACHE_WTHROUGH
                        || CCM.CMODE[1] == CACHE_WBACK$
……
```

- 単に上のように修正すると，エラーになる
  - マクロプロセッサが，定数名の数値を知らないため

値取得シンボルテーブル（kernel/kernel_def.csv）への追加

```
CACHE_NONE,CACHE_NONE
CACHE_WTHROUGH,CACHE_WTHROUGH
CACHE_WBACK,CACHE_WBACK
```

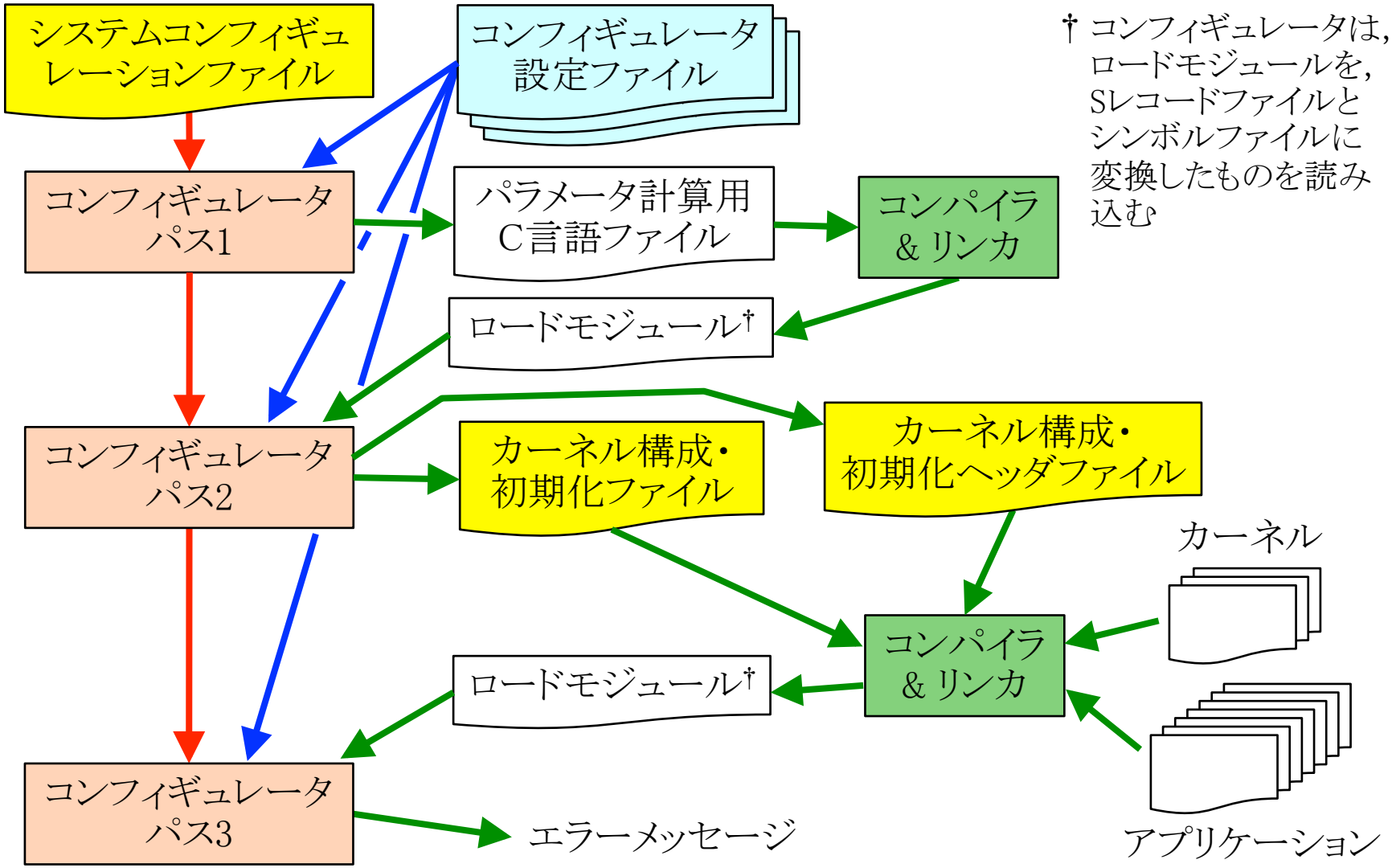
値を取得したい定数式

取得した値を格納する
テンプレートファイル変数名

- これにより，コンフィギュレータが3つの定数名の数値を取り出し，テンプレートファイル内で用いることができる

コンフィギュレータの仕組み

コンフィギュレータの内部の処理（ASPカーネルの場合）



パス1の役割

- 静的APIの整数定数式パラメータの値を取得するために，パラメータ計算用C言語ファイル（cfg1_out.c）を生成
- 同時に，値取得シンボルテーブルに記述したC言語定数式の値も取り出す（詳しくは後述）

パラメータ計算用C言語ファイルの内容の例

システムコンフィギュレーションファイル

```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task,
                     MAIN_PRIORITY, STACK_SIZE, NULL });
```

静的APIが有効か（条件ディレクティブで消えていないか）を判別するために使用

パラメータ計算用C言語ファイル

```
const unsigned_t TOPPERS_cfg_static_api_26 = 26;
const unsigned_t TOPPERS_cfg_valueof_tskatr_26 = ( unsigned_t )( TA_ACT );
const signed_t TOPPERS_cfg_valueof_itskpri_26 = ( signed_t )( MAIN_PRIORITY );
const unsigned_t TOPPERS_cfg_valueof_stksz_26 = ( unsigned_t )( STACK_SIZE );
```

パス2の役割

- カーネル構成・初期化ファイル（kernel_cfg.c）とカーネル構成・初期化ヘッダファイル（kernel_cfg.h）を生成

パス3の役割

- 一般定数式パラメータの値をチェックし，不正な場合にはエラーを報告
  - 一般定数式パラメータの値は，ロードモジュールを作るまで決まらない
- チェックするエラーの例
  - 処理単位の先頭番地が正しくアラインしているか
  - スタック領域の先頭番地が正しくアラインしているか

注意

- 各パスの役割は，ASPカーネルの場合である．HRP2カーネルでは最大パス4までであり，各パスの役割も異なる

コンフィギュレータの設定ファイル

- コンフィギュレータの動作は，以下の設定ファイルにより決定される

静定APIテーブル

- 静的APIとそのパラメータの一覧表
- すべてのパスで参照する

値取得シンボルテーブル

- コンフィギュレータが読み込む定数値の一覧表
- パス1で主に用いるが，すべてのパスで参照する

テンプレートファイル

- コンフィギュレータによるファイルの生成規則（より一般には，コンフィギュレータの処理手順）の記述
- パス2以降のパスで，パス毎に異なるテンプレートファイルを用いる

静定APIテーブル

静的APIテーブルとは？

- コンフィギュレータが解釈する静的APIの一覧と，そのパラメータを記述するためのファイル
- ターゲット非依存部は，kernel/kernel_api.csv

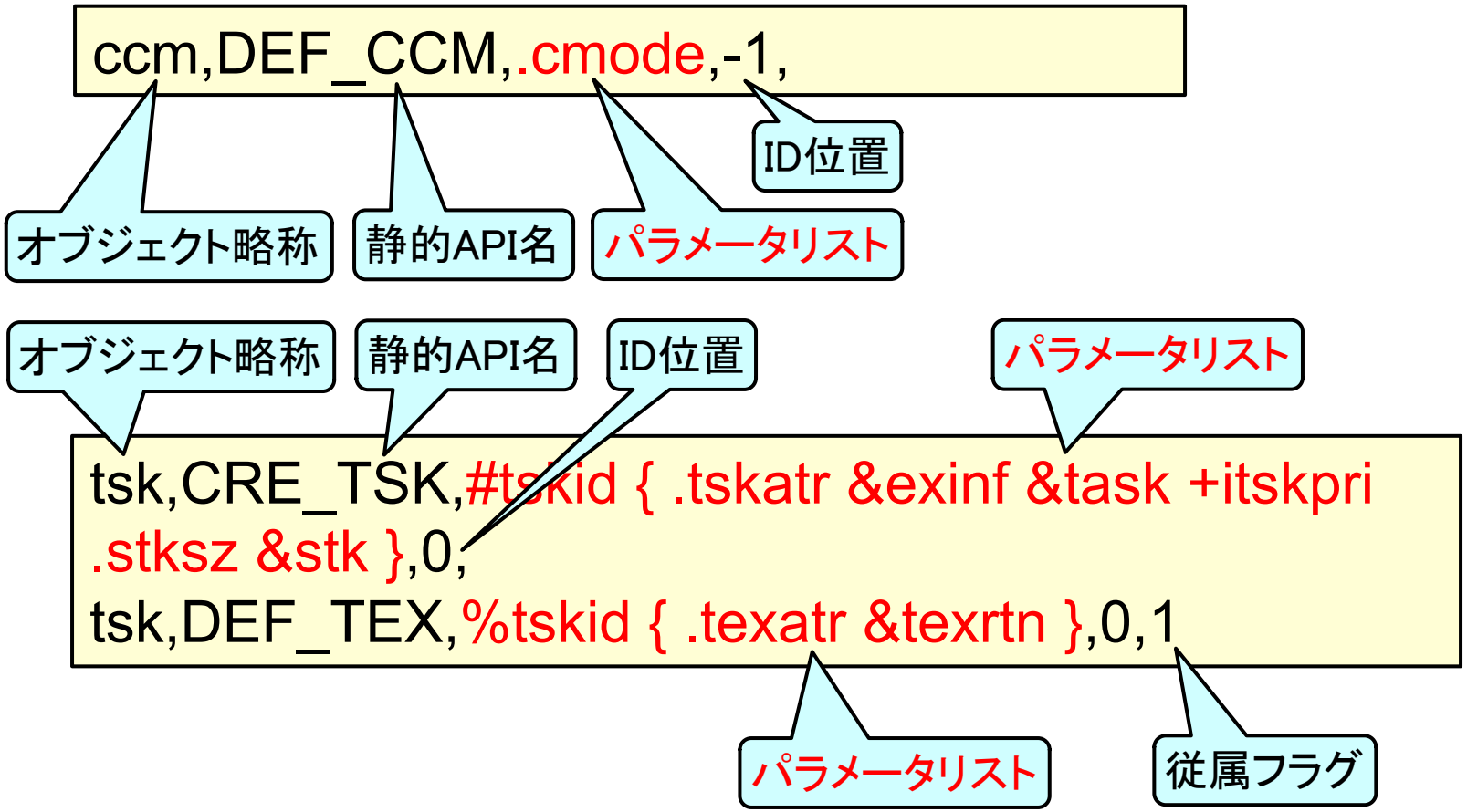
静的APIテーブルの指定方法

- cfgに対して，--api-tableオプションによって指定する
- 複数の静的APIテーブルを指定することができる

静的APIテーブルの書式

- CSV（コンマ区切り）形式で，各レコード（行）は次の形式
- 〔オブジェクト略称〕,〔静的API名〕,〔パラメータリスト〕,〔ID位置〕,〔従属フラグ〕

静的APIテーブルの記述例



オブジェクト略称

- オブジェクトの種類を表す文字列（通常は，サービスコール中でオブジェクトを表す3文字の略称）
- テンプレートファイル中では，大文字に変換され，パラメータ等を格納する変数名に使用される

ID位置

- オブジェクトIDが，パラメータリスト中の何番目にあるか（最初のパラメータであれば0，省略したら0とみなす）
- オブジェクトIDがない場合には，–1を指定する

従属フラグ

- 他の静的APIで登録されたオブジェクトに情報を追加する静的APIの場合に，1を指定する（省略したら0とみなす）

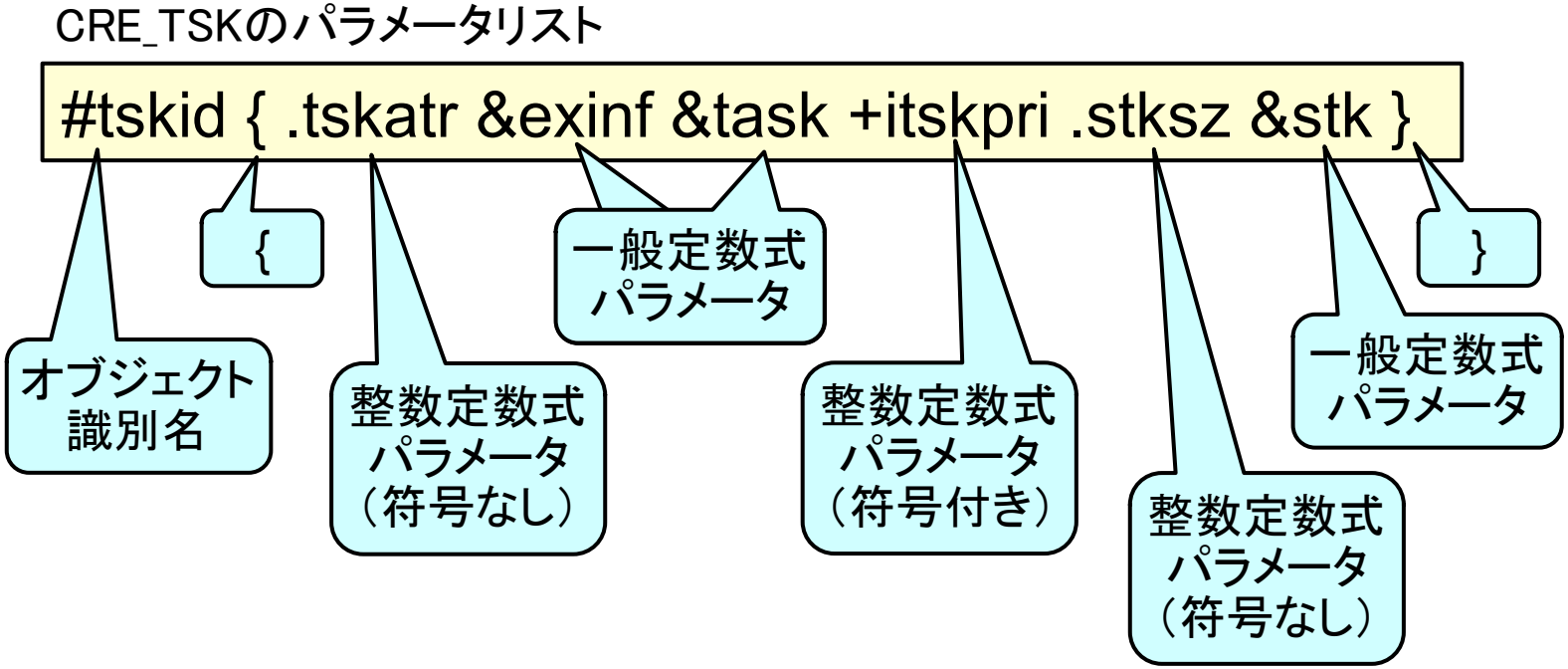
パラメータリスト

- 静的APIのパラメータ名を，空白文字で区切ってリストアップする
  - 中括弧（「{」「}」）もパラメータのように記述する
- パラメータの種別に応じて，各パラメータ名の前に次のいずれかの記号を付ける

#	オブジェクト識別名 … コンフィギュレータがID番号を割り付ける
%	オブジェクト識別名（定義済み）
.	整数定数式パラメータ（符号なし）
+	整数定数式パラメータ（符号付き）
&	一般定数式パラメータ
$	文字列パラメータ

- その他に，パラメータを省略できることや，同種のパラメータを複数記述できることを示す記法がある

パラメータリストの記述例



- 整数定数式パラメータは，テンプレートファイル内で，数値を取り出すことができる
- 一般定数式パラメータは，数値を取り出せない（ポインタの可能性があり，リンクしてみるまで数値が決まらない）

値取得シンボルテーブル

値取得シンボルテーブルとは？

- テンプレートファイル内で数値を取り出したいC言語の定数式の一覧を記述するためのファイル
- ターゲット非依存部は，kernel/kernel_def.csv

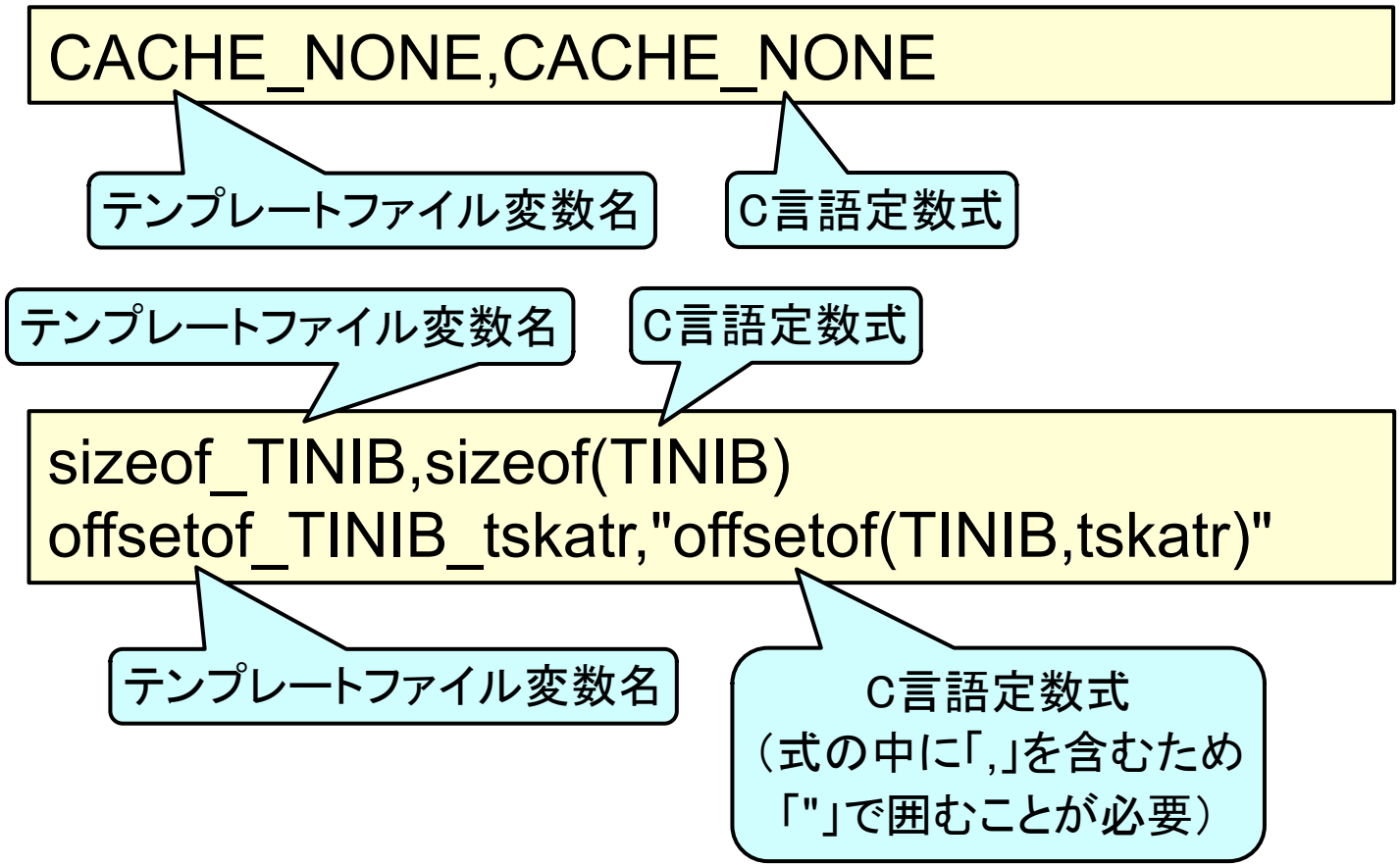
値取得シンボルテーブルの指定方法

- cfgに対して，--cfg1-def-tableオプションによって指定する
- 複数の値取得シンボルテーブルを指定することができる

値取得シンボルテーブルの書式

- CSV（コンマ区切り）形式で，各レコード（行）は次の形式
- 〔テンプレートファイル変数名〕,〔C言語定数式〕,〔符号付きフラグ〕,〔式が真の場合の定数式〕,〔式が偽の場合の定数式〕

値取得シンボルテーブルの記述例（1）



符号付きフラグ

- 式の評価結果が符号付きの整数型の場合には，「s」または「signed」を指定する
- 式の評価結果が符号なしの整数型の場合には，「u」を指定するか，指定を省略する

C言語定数式の拡張記法

- C言語定数式の先頭に「#」を付加した場合，〔C言語定数式〕から先頭の「#」を取ったものが，プリプロセッサディレクティブ#ifの条件と扱われる

式が真/偽の場合の定数値

- これを指定した場合，〔C言語定数式〕から先頭の「#」を取ったものが#ifの条件と扱われ，その値の真/偽に応じて，ここに指定した定数式の値がテンプレートファイル変数に設定される
- 省略した場合は，それぞれ1および0となる

値取得シンボルテーブルの記述例（2）

```
SIL_ENDIAN_BIG,#defined(SIL_ENDIAN_BIG)
```

SIL_ENDIAN_BIGが
マクロ定義されていれば1に，
そうでなければ0に設定される

TARGET_MIN_STKSZが
マクロ定義されていれば
TARGET_MIN_STKSZに，
そうでなければ0に設定される

```
TARGET_MIN_STKSZ,#defined(TARGET_MIN_STKSZ),,
TARGET_MIN_STKSZ,0
```

パラメータ計算用C言語ファイル

```
const unsigned_t TOPPERS_cfg_TARGET_MIN_STKSZ =
#if defined(TARGET_MIN_STKSZ)
(TARGET_MIN_STKSZ);
#else
(0);
#endif
```

テンプレートファイル

テンプレートファイルとは？

- コンフィギュレータのパス2以降において，ファイルを生成するためのルールを記述するためのファイル

テンプレートファイルの指定方法

- cfgに対して，--T（または，--template-file）オプションによって指定する
- テンプレートファイルは，パス毎に1つのみ指定できる
- パス2とパス3では，異なるテンプレートファイルを用いる

テンプレートファイルの文法

- 基本的には，テンプレートファイルの内容を，生成するファイルにそのまま出力する
- 「$」で囲まれた文字列は，マクロプロセッサで処理する

マクロプロセッサの概要

マクロプロセッサの役割

- テンプレートファイル中の「$」で囲まれた文字列を処理する

基本的なマクロ命令

- $FILE "<ファイル名>"$
  - 生成するファイルの名称を指定する

  例）$FILE "kernel_cfg.c”$
- $INCLUDE "<ファイル名>"$
  - 他のテンプレートファイルをインクルードする

  例）$INCLUDE "kernel/kernel.tf”$
- $ERROR ＜エラー発生箇所＞$＜エラーメッセージ＞END
  - エラーメッセージを出力する

マクロプロセッサが扱う値

- マクロプロセッサが扱う値は，文字列属性（システムコンフィギュレーションファイル中の字面）と数値属性（その値）を持つ．片方の属性しか設定されていない場合もある
  - 値を出力する場合には，文字列属性が出力される
  - 値に対して数値演算を行う場合には，数値属性が使われる．数値属性が設定されていない場合には，エラーとなる（“non-value is referred”）
- マクロプロセッサは，値の順序付きリストを扱うことができる
  - 単純な値は，長さが1の順序付きリストと等価である
  - 順序付きリストの定数は，等差数列でも記述できる

例）$INHNO_DEFINH_VALID = { 0x10,0x11,...,0x1f;
0x40,0x41,...,0xff }$

  - リストの個々の要素が，文字列属性と数値属性を持つことができる

テンプレートファイル変数

- 2種類のテンプレートファイル変数がある
- 単純変数
  - 単一の識別子（「.」を含むことができる）で表す

  例）CCM.ORDER_LIST
- 連想配列
  - 識別子の後に「[」「]」で囲まれた添え字を付けて表す

  例）CCM.CMODE[i]
  - 添え字には，数値のみが使える（厳密に言うと，添え字の値の数値属性が使われる）
  - 添え字が連続している必要はない（なので，連想配列）
- テンプレートファイル変数には，値（単純な値，順序付きリスト）を1つ格納できる
- スコープの概念はない（すべてグローバル変数）

静的APIのパラメータの読込み

- 静的APIのパラメータは，テンプレートファイル変数に読み込まれる
- 静的APIのパラメータの読込み例

静的APIテーブル

```
tsk,CRE_TSK,#tskid { .tskatr &exinf &task +itskpri
.stksz &stk },0,
```

TSK.ID_LIST … タスクのID番号のリスト（昇順）

TSK.ORDER_LIST … 同上（出現順）

TSK.TSKATR[tskid] … tskatrパラメータの値（整数定数式パラメータなので，文字列属性，数値属性とも設定）

TSK.EXINF[tskid] … exinfパラメータの値（一般定数式パラメータなので，文字列属性のみ設定）

この他にもいくつかのテンプレートファイル変数が設定

制御構文

- マクロプロセッサは，以下の制御構文をサポートしている
- 条件文
  - IF, $ELIF$, $ELSE$, END
- 繰り返し文
  - $FOREACH$, END
  - $JOINEACH$, END
  - $WHILE$, END
  - $JOINWHILE$, END
- JOINEACHの使用例

区切りに使う文字列

```
$JOINEACH i { 1,3,1,0 } ”, ”$(base + $i$)$END$
```

↓

```
(base + 1), (base + 3), (base + 1), (base + 0)
```

組込み関数

- マクロプロセッサは，数多くの組込み関数をサポートしている
- 値の操作
  - $VALUE$ … 文字列と数値からの値の生成
  - ALT … 値が未定義の場合の代替値
- 順序付きリストの操作
  - $LENGTH$ … 順序付きリストの長さの取得
  - AT … 順序付きリストの要素の取得
  - $APPEND$ … 順序付きリストの連結
  - $FIND$ … 順序付きリスト内の探索
  - $RANGE$ … 数値のリストの生成
  - $SORT$ … 順序付きリストのソート
  - $LSORT$ … 順序付きリストのユーザ定義ソート

組込み関数 – 続き

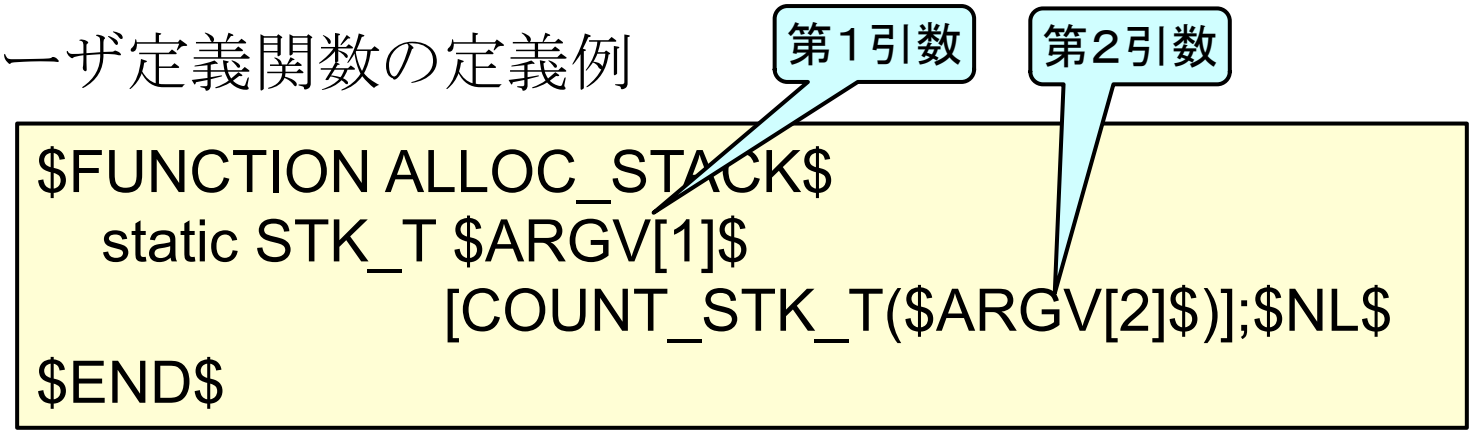
- 文字列の操作
  - EQ … 文字列の比較
  - $CONCAT$ … 文字列の連結
  - $FORMAT$ … フォーマット文字列の生成
  - $REGEX_REPLACE$ … 正規表現による置換

  などなど
- オブジェクトファイル操作
  - $SYMBOL$ … シンボルに対するアドレスの取得
  - $PEEK$ … アドレスの内容の取得
- その他
  - $ISFUNCTION$ … 関数定義があるかの判定
  - DIE … マクロプロセッサの強制終了

  などなど

ユーザ定義関数

- マクロプロセッサは，ユーザ定義関数を定義し，呼び出す機能を持つ
- ユーザ定義関数の定義例



- ユーザ定義関数の呼出し例

```
$IF ISFUNCTION("ALLOC_SSTACK")$
    $ALLOC_STACK(CONCAT("_kernel_stack_", tskid),
                                stksz)$
$END$
```

実行時間分布集計モジュールに対する静的APIを作る

実施する開発内容と方針

実行時間分布集計モジュールとは？

- ASPカーネルに付属のサポートライブラリの1つで，システムのリアルタイム性能を評価するために，プログラム区間の実行時間を計測し，その分布を集計・表示するためのライブラリ関数群

静的APIを作る動機

- 実行時間分布を格納するデータ構造（複数設けることができ，ID番号で識別する）は，init_hist関数で動的に初期化
- 実行時間分布を格納するメモリ領域を正しいサイズで確保するのは，ユーザの責任になっている
- データ構造の最大数は，ライブラリのコンパイル時に決定する必要がある

→ 静的APIを設けることにより，エレガントに解決可能

init_histの仕様

- void init_hist(ID histid, uint_t maxval, uint_t histarea[])
  - histid： 実行時間分布を格納するデータ構造のID番号
  - maxval： 記録する最大時間
  - histarea ：実行時間分布を格納するメモリ領域（maxvalに指定した値+1の要素数のuint_t型の配列として，ユーザ側で確保する）の先頭番地

実装する静的API

- CRE_HIST(ID histid, { uint_t maxval })
- データ構造を置くメモリ領域は，コンフィギュレータが確保

実装の方針

- ターゲット依存部のみの修正で実装する（ターゲット非依存部のファイルを修正しない）

生成したいファイル

- 次のような静的APIを記述した場合を例に考える

```
CRE_HIST(HIST_A, { 1000 });
CRE_HIST(HIST_B, { HIST2_MAXVAL });
```

- HIST2_MAXVALの定義は，システムコンフィギュレーションファイルからINCLUDEされているヘッダファイルのいずれかに含まれているものとする

カーネル構成・初期化ヘッダファイル（kernel_cfg.h）

- カーネル構成・初期化ヘッダファイル（kernel_cfg.h）には，以下の（ような）記述を生成したい

```
#define TNUM_HISTID     2
#define HIST_A   1
#define HIST_B   2
```

カーネル構成・初期化ファイル（kernel_cfg.c）

- カーネル構成・初期化ファイル（kernel_cfg.c）には，以下の（ような）記述を生成したい

データ構造を置くメモリ領域の定義

```
uint_t histarea_HIST_A[(1000) + 1];
uint_t histarea_HIST_B[(HIST2_MAXVAL) + 1];

const HISTINIB histinib_table[TNUM_HISTID] = {
    { 1000, histarea_HIST_A },
    { HIST2_MAXVAL, histarea_HIST_B }
};

HISTCB histcb_table[TNUM_HISTID];
```

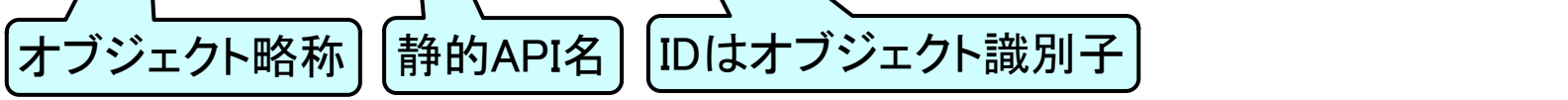
- HISTINIB構造体とHISTCB構造体の定義は，システムコンフィギュレーションファイルからINCLUDEされているヘッダファイルのいずれかに含まれているものとする

静的APIテーブルの作成

ターゲット依存の静的APIテーブルの準備

- ターゲット依存部のディレクトリに，以下の内容のtarget_api.csvを設ける

```
hist,CRE_HIST,#histid { .maxval },,
```



ターゲット依存の静的APIテーブルが使われるようにする

- ターゲット依存部のMakefile（Makefile.target）に以下の記述を追加

```
CFG_TABS := $(CFG_TABS) \
            --api-table $(TARGETDIR)/target_api.csv
```

テンプレートファイルの修正

- 以下の記述は，パス2のテンプレートファイルのターゲット依存部（target.tf）に対して追加する

カーネル構成・初期化ヘッダファイル（kernel_cfg.h）の生成

出力先のファイルの切換え

```
$FILE "kernel_cfg.h"$

#define TNUM_HISTID  $LENGTH(HIST.ID_LIST)$$NL$

$FOREACH histid HIST.ID_LIST$
    #define $histid$  $+histid$$NL$
$END$$NL$
```

histidの値の文字列属性

histidの値の数値属性

カーネル構成・初期化ファイル（kernel_cfg.c）の生成（不完全）

```
$FILE "kernel_cfg.c"$

$FOREACH histid HIST.ID_LIST$
    uint_t histarea_$histid$[($HIST.MAXVAL[histid]$) + 1];$NL$
$END$$NL$

const HISTINIB histinib_table[TNUM_HISTID] = {$NL$
$JOINEACH histid HIST.ID_LIST ",\n"$
    $TAB${ $HIST.MAXVAL[histid]$, histarea_$histid$ }
$END$$NL$
};$NL$
$NL$

HISTCB histcb_table[TNUM_HISTID];$NL$
$NL$
```

残る実装作業

実行時間分布集計モジュールの修正

- histogram.hとhistogram.cを，生成したコードを使うように修正する

テンプレートファイルを完全にする

- 前述のコードには，以下の改善が必要
- CRE_HISTがない場合への対応
  - 現状のコードでは，サイズが0の配列（標準のC言語では非対応）が生成される
  - 配列を生成せず，（リンクエラーを防ぐために）ラベルのみダミーで生成する方法を推奨
- エラーチェックの追加
  - maxvalパラメータが負の値の場合 → この状況を検出してエラーとする

コンフィギュレータの歴史と今後

TOPPERSコンフィギュレータの歴史

JSPカーネルのコンフィギュレータ

- コンフィギュレータ全体が，C++で作り込まれていた
- 生成するコードを変更するには，コンフィギュレータのソースコードの修正＆リコンパイルが必要

→ ターゲット毎に異なるコードを生成するのは面倒（実質的には不可能）

テンプレートファイルの導入検討

- ASPカーネルにおいて，割込みサービスルーチン（ISR）を効率的に実現するために，ターゲットプロセッサの割込みアーキテクチャに応じて，異なるコードを生成することが必須に
- 割込みに関連する記述を，テンプレートファイルに従って生成するためのコンフィギュレータの仕様を検討．実装することに

テンプレートファイルの全面的な導入

- 割込みに関連する記述をテンプレートファイルに従って生成できる機能があるなら，すべての記述をテンプレートファイルに従って実現することも可能に
- コンフィギュレータ本体から，カーネルに依存する部分を減らせるというメリット

→ TOPPERS新世代カーネルの共通コンフィギュレータに

徐々に機能拡張

- 各種の機能（特に，HRP2カーネルにおけるメモリ保護情報の生成）を実現するために，コンフィギュレータの機能を徐々に拡張．例えば，
  - 文字列操作機能の強化
  - ユーザ定義関数の導入
  - 省略可能なパラメータのサポート

他の入力形式への対応

- ATK1（OSEK/VDX OS仕様準拠）のジェネレータも共通化できるように，OIL（OSEK Implementation Language）記述に対応
  - OIL記述の内容をテンプレートファイル変数に読み込めば，その後の処理（マクロプロセッサ）は共通
  - 最初は，開発した会員により，“Contributed Software”のページにアップされた．現在は公式リリースに統合
- ATK2（AUTOSAR OS仕様準拠）にも用いるために，AUTOSAR XML記述にも対応
  - 現時点では，すべてのAUTOSAR XML記述に対応できていないため，さらなる拡張を計画

最新版

- Release 1.9.1（2013年4月5日付けでリリース）

TOPPERSコンフィギュレータの今後

今後の拡張の方向性

- カーネルに依存した機能のさらなる分離
- エラーメッセージの改善
- ミドルウェア（ソフトウェア部品）への対応
  - ミドルウェアのオブジェクトに応じて，必要なカーネルオブジェクトが変わる場合（例えば，シリアルポート1つ毎に，セマフォが2つ必要など）にどうするか？
  - そのための機能はすでに入っているが，それで十分かが未検証

大きい課題

？TECSとの関連は？

より詳しく知るために
～まとめに代えて～

より詳しく知るために

- TOPPERSコンフィギュレータには，以上で説明した以外にも，多くの機能がある

参考文献

- TOPPERS新世代カーネル用コンフィギュレータ仕様（最終更新：2012年12月19日）
- TOPPERS新世代カーネル用コンフィギュレータ内蔵 マクロプロセッサ仕様（最終更新：2012年11月6日）

参考コード

- TOPPRES/HRP2カーネルのテンプレートファイル（ターゲット非依存部，ターゲット依存部）
- TOPPERS/ATK2のテンプレートファイル