TOPPERS 活用アイデア・アプリケーション開発コンテスト

部門　：活用アイデア部門　　アプリケーション開発部門
作品のタイトル　：　Ruby 版 TOPPERS コンフィギュレータ
作成者　：　富士ソフト株式会社（代表：鴫原一人）
対象者　：　TOPPERS ソフトウェア開発者(特にポーティングを行う方)
使用する開発成果物　：
・ASP カーネル R1.9.0 (Skyeye シミュレータ簡易パッケージ)
・TTSP Release 1.1.2
・TOPPERS 新世代カーネル用コンフィギュレータ(cfg.exe)
・TOPPERS 新世代カーネル用コンフィギュレータ仕様書
・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書

目的・狙い

現在TOPPERSカーネルで使用されているTOPPERS新世代カーネル用コンフィギュレータ(以後、現コンフィギュレータ)は、新規マイコンへのポーティングの際、独自言語であるマクロプロセッサ用言語(以後、tf)を使用する必要がある。また、現コンフィギュレータが使用する Boost ライブラリのバージョン等、環境に依存する問題も度々発生する。そこで、Ruby でコンフィギュレータを開発し、tf も Ruby で実装可能とすることで、これらの問題を解決できないかと考えた、Ruby 版 TOPPERS コンフィギュレータ(以後、新コンフィギュレータ)を試作した。

アイデア/アプリケーションの概要

- 現コンフィギュレータ仕様書をベースに、Ruby で新コンフィギュレータを開発する
- tf で実装していた処理を Ruby で書きなおす
  - 通常の Ruby ファイルと区別するため拡張子を trb とする
- ASP カーネル及び TTSP を使用して、現コンフィギュレータとの互換性確認を行う
- 新コンフィギュレータのメリット、デメリットを考察する
- 現コンフィギュレータの trac に発行されているチケットが､新コンフィギュレータで解決可能か検討する

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

1 現コンフィギュレータの問題点

1.1 独自言語

TOPPERS カーネル開発者(特に TOPPERS カーネルのポーティングを行う者)は、現コンフィギュレータが提供する tf を使用する必要がある。しかし、tf は TOPPERS の独自言語であり、初めて TOPPERS カーネルを開発、ポーティングする方は、まず tf を習得しなければならない。tf は、RTOS のコンフィギュレーションに特化した特殊な言語仕様であり、習得が容易とは言えず、デバッグ容易性も低い。また、独自言語がゆえ、テキストエディタによるソースコードの色分け等にもデフォルトで対応しておらず、可読性も低い。

1.2 開発言語

現コンフィギュレータは、C++で開発されており、Boost ライブラリを使用している。C++コンパイラや Boost ライブラリのバージョンは、常にバージョンが上がっていくので、現コンフィギュレータの修正時、ビルドが正常に行えない問題が度々発生する(本資料作成時点で、現コンフィギュレータの trac にビルドに関連する 3 件のチケットが発行されている)。また、実行モジュールも、Windows、MacOS、Linux など環境に合わせて用意する必要がある。

2 代替言語

前章で述べた現コンフィギュレータの問題点を解決するには、現コンフィギュレータ及び tf の構造を抜本的に解決する必要があると考えた。

まず、多くの開発者が使用する tf は、可読性、実装容易性、習得容易性を重視する必要がある。これを実現するには、独自言語をやめ、一般的に使用されている言語を採用するのが妥当と考え、簡潔に処理を記述でき、可読性が高いことで定評がある Ruby を選定した。Ruby は、日本発のプログラミング言語であり、世界的にも普及率が高いため、TOPPERS プロジェクトが採用する言語としても妥当である。なお、tf ファイル(tf で記述されたファイルを指す)相当のファイルを Ruby で実現したファイルの拡張子は、rb ではなく、trb とすることで明確に区別することにした。

Ruby は、実行中の Ruby ファイルから別の Ruby ファイルを呼び出して実行することができることから、コンフィギュレータ自体も Ruby で開発することにした。これにより、コンフィギュレータが解釈したコンフィギュレーション情報をそのままのデータで、trb ファイルに渡すことができる。また、Ruby は、実行時の引数オプション取得機能も豊富で、現コンフィギュレータの引数オプションを全くそのまま実現することができる。

Windows には Ruby の実行環境が標準で入っていないが、TOPPERS カーネルでは、make や perl などの GNU コマンドが揃っていることを前提とすることが多く、Ruby の実行環境を揃えることは困難ではない。インターネットから Ruby の実行環境を入手すること

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

も容易である。

3 Ruby でのコンフィギュレータ実装

3.1 文字列＋数値

tf では、ユーザがシステムコンフィギュレーションファイル(以後、cfg ファイル)に記述した文字列と、その文字列をコンパイラを用いて評価した数値の 2 つのデータを 1 つの変数として保持する。これにより、出力するファイルには、ユーザが書いたままの文字列を使用し、エラーチェック等の演算処理では数値を使用する、といった使い方ができる。これはコンフィギュレータの機能として有用であるので、新コンフィギュレータでも同等の機能を実装した。具体的には、Ruby で標準的に用意されている文字列クラス(String クラス)を継承した StrVal クラスを用意し、数値データも保持するようにした。tf 同様、”+” や” -“などの演算子と組み合わせて使用した場合は、数値側で評価するように、String クラスの演算子をオーバーライドした。なお、明示的に数値を取り出したい場合、tf では” +” などの演算子を変数名に付加していたが、trb では、より明示的になるよう変数名に” .val” を付加することにした。

3.2 ファイル出力

tf は、テンプレートファイルの名前の通り、基本的には tf ファイルに書いた内容を、ファイルに出力する前提となっており、出力内容を演算結果によって変更したい箇所にのみ$で囲んだマクロ命令を記述する形式となっている。しかし、現状の tf ファイルは、演算処理の実装の方が多くなり、テンプレートファイルという形式を取るメリットは小さくなっている。そこで、新コンフィギュレータでは、テンプレートファイル形式ではなく、ファイル出力クラスを用意し、出力する文字列をインスタンスに追加していき、最後にファイル出力を行うメソッドを呼び出す形式とした。

なお、Ruby はヒアドキュメントを使用できるので、まとまった固定的な文字列は tf 同様に、そのまま記述することができる。tf のように改行(NL)を明示的に記述する必要がないため、tf より可読性、実装容易性が高い。

3.3 syms ファイル、srec ファイル処理

Ruby には強力なテキスト処理クラス群が用意されているため、syms ファイルや srec ファイルに対する処理は簡潔に実装することができた。syms ファイルは、シンボルをキー、アドレスを値とするハッシュに保持する。srec は、srec データ内のデータ列を 1 つの文字列で丸ごと保持するクラスを用意し、指定したアドレスとサイズのデータを返すメソッドを用意した。また、エンディアンと符号の有無も加えることで、指定したデータの値を返すメソッドを用意した。これらの実装により、SYMBOL、BCOPY、PEEK といった関数

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

は数行で実現することができた。

3.4 値取得シンボルテーブル

値取得シンボルテーブルで指定された値は、trb 内でグローバル変数として扱えるようにした。Ruby のグローバル変数は先頭に”$”が付与されるので、tf で値取得シンボルテーブルの名称のまま使用していた処理に関しては、”$”を付与する必要がある。

3.5 trb に渡されるコンフィギュレーション情報

現コンフィギュレータでは、静的 API テーブルに記述された識別子をベースに、リスト形式でコンフィギュレーション情報が tf に渡される。新コンフィギュレータでは、静的 API テーブルに記述された識別子をベースに、ハッシュ形式で trb へコンフィギュレーション情報を渡すことにした。これにより、インデックス管理が不要となり、直感的にコンフィギュレーション情報を把握することが可能となった。

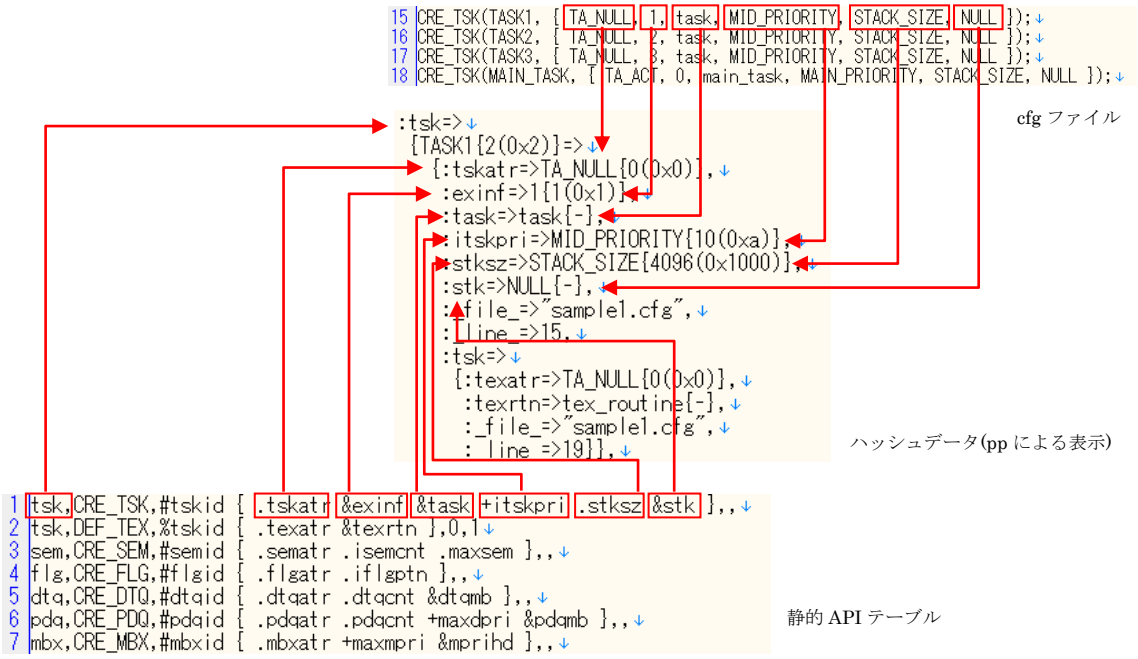

図 1 trb に渡されるハッシュデータの例(CRE_TSK)

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

4 TTSP による現コンフィギュレータとの互換性確認

TTSP Release 1.1.2 を用いて、API テスト、SIL テストを実施し、全件パスすることを確認した。特に静的 API テストでは、tf から trb への変更ミス等を的確に検出でき、TTSP の有用性も確認できた。

また、新コンフィギュレータ開発において、現コンフィギュレータおよび ASP の tf ファイルにおける問題点をいくつか検出し、現コンフィギュレータのチケットを 3 件(#150～#152)、ASP カーネルのチケットを 2 件(#348、#349)を発行した。新コンフィギュレータでは、これらの問題を解決済みである。

5 現コンフィギュレータとの比較

5.1 メリット

5.1.1 冗長コードの削減

全コンフィギュレーション情報が 1 つのハッシュで渡されるため、各オブジェクトに共通な処理をまとめることができる。

```
63 $FOREACH id TSK.ID_LIST$
64 ^    #define $id$^    $+id$$NL$
65 $END$
66 $FOREACH id SEM.ID_LIST$
67 ^    #define $id$^    $+id$$NL$
68 $END$
69 $FOREACH id FLG.ID_LIST$
70 ^    #define $id$^    $+id$$NL$
71 $END$
72 $FOREACH id DTQ.ID_LIST$
73 ^    #define $id$^    $+id$$NL$
74 $END$
75 $FOREACH id PDQ.ID_LIST$
76 ^    #define $id$^    $+id$$NL$
77 $END$
78 $FOREACH id MBX.ID_LIST$
79 ^    #define $id$^    $+id$$NL$
80 $END$
81 $FOREACH id MPF.ID_LIST$
82 ^    #define $id$^    $+id$$NL$
83 $END$
84 $FOREACH id CYC.ID_LIST$
85 ^    #define $id$^    $+id$$NL$
86 $END$
87 $FOREACH id ALM.ID_LIST$
88 ^    #define $id$^    $+id$$NL$
89 $END$
```

図 2 冗長コード削減の例(オブジェクト ID 定義)

5.1.2 break、continue(next)

tf では、ループ内で break や continue といった制御文が使用できず、別途変数を用意するなどして、同等の処理を実現しているが、Ruby ではこれらの制御文を使用できるので、

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

処理を簡潔に記述でき、可読性も高い。

```
75    $val = 0$
76    $FOREACH i RANGE(0,struct_size-1)$
77        $tmp_val = PEEK(top + i, 1)$
78        $IF val == 0 && tmp_val != 0$
79            $val = tmp_val$
80            $offset = i$
81        $END$
82    $END$
```

```
74  nVal = 0
75  nOffset = 0
76  (0..(nStructSize - 1)).each{|i|
77    nVal = PEEK(nTop + i, 1)
78    if (nVal != 0)
79      nOffset = i
80      break
81    end
82  }
```

図 3 break の使用例(DEFINE_BIT 関数)

5.1.3 複数行コメントアウト

tf では、C 言語の/**/のように行単位でのコメントアウトができず、まとまった処理を一括してコメントアウトできない。Ruby では、=begin/=end によって、行単位のコメントアウトが可能であるので、デバッグ時に便利である。

5.1.4 カバレッジ取得

Ruby R1.9 系であれば、simplecov というライブラリを用いて容易にカバレッジを取得することができる。TTSP 実行時に、新コンフィギュレータ(cfg.rb)と、各 trb ファイルのカバレッジを測定したところ、テスト時に実行されていないコードが明確になった。現コンフィギュレータの trac でも tf のカバレッジ取得に対する要望のチケット(#27)が発行されている。

All Files (85.53%)

All Files (85.53% covered at 47759.2 hits/line)

9 files in total. 1320 relevant lines. 1129 lines covered and 191 lines missed

Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
asp/kernel/genoffset.trb	73.47 %	136	49	36	13	81.6
asp/kernel/kernel_check.trb	81.54 %	158	65	53	12	577.0
cfg.rb	82.83 %	1506	728	603	125	86379.5
asp/kernel/kernel.trb	90.17 %	1000	417	376	41	240.9
asp/arch/arm_gcc/common/core.trb	100.0 %	43	12	12	0	133.2
asp/arch/arm_gcc/common/core_check.trb	100.0 %	17	1	1	0	39.0
asp/arch/arm_gcc/common/core_offset.trb	100.0 %	40	11	11	0	40.0
asp/target/at91skyeye_gcc/target.trb	100.0 %	98	36	36	0	383.3
asp/target/at91skyeye_gcc/target_check.trb	100.0 %	17	1	1	0	39.0

Showing 1 to 9 of 9 entries

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

```
260. #
261. #  イベントフラグ
262. #
263. cKerCfgC.comment_header("Eventflag Functions")                                   106
264.
265. # イベントフラグID番号の最大値
266. cKerCfgC.add2("const ID _kernel_tmax_flgid = (TMIN_FLGID + TNUM_FLGID - 1);")    106
267.
268. # イベントフラグ初期化ブロックの生成
269. if ($hCfgData[:flg].size() != 0)                                                 106
270.   cKerCfgC.add("const FLGINIB _kernel_flginib_table[TNUM_FLGID] = {")               5
271.   $hCfgData[:flg].each{|cID, hParam|                                                 5
272.     # flgatrが（[TA_TPRI] | [TA_WMUL] | [TA_CLR]）でない場合（E_RSATR）
273.     if ((hParam[:flgatr] & ~($TA_TPRI|$TA_WMUL|$TA_CLR)) != 0)                     151
274.       error_illegal4("E_RSATR", hParam, "flgatr", hParam[:flgatr], cID, "CRE_FLG")     1
275.     end
276.
277.     # iflgptnがFLGPTNに格納できない場合（E_PAR）
278.     if ((hParam[:iflgptn] & ~((1 << $TBIT_FLGPTN) - 1)) != 0)                      151
279.       error_is_xxx4("E_PAR", hParam, "iflgptn", hParam[:iflgptn], cID, "CRE_FLG", "too large")
280.     end
281.
282.     # イベントフラグ初期化ブロック
283.     cKerCfgC.add("#{TAB}{ (#{hParam[:flgatr]}), (#{hParam[:iflgptn]}) },")          151
284.   }
```

図 4 TTSP 実行時のカバレッジ取得結果

5.1.5 Ruby による実装

Ruby を用いて実装できることで、Ruby が提供する強力なテキスト処理を使用することができ、trb 側でクラスを定義することもできる。ASP カーネル用の各オブジェクト処理クラスを定義して、FMP カーネルや HRP2 カーネルでは、ASP カーネルのクラスを継承して実装する、といった開発も可能になる等、今後、より利便性を高めるための取り組みに対して、大きな可能性が提供される。また、Ruby に関する情報はインターネット上に豊富にあるため、trb ファイル開発工数の削減も期待できる。

なお、新コンフィギュレータの開発は約 5 人日、ASP カーネル内の tf ファイルを trb ファイルに書き換えるのは約 1 人日で完了したことからも、Ruby による開発効率の高さが確認できた。無論、現コンフィギュレータと Ruby に関して一定の知識があることが前提であるが、現コンフィギュレータの対象 C++ソースコードが約 15,000 行であるのに対し、新コンフィギュレータの Ruby ソースコードは、1,500 行程度であることからも、開発規模は劇的に小さくなっている。

5.2 デメリット

5.2.1 JOINEACH

現コンフィギュレータには、構造体定義等を行う際に最後の要素だけ”,”を付けない、といったループ処理を行う JOINEACH という制御文がある。Ruby にはこの制御文が存在しないため、一旦",”を付与して、最後に”,”を削除するメソッド(chop_comma)を用意することで対応した。tf では記述した文字列がそのままファイルに出力されるので、一旦記述したコ

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

ードを取り消すことはできないが、trb では、ファイル出力クラスで保持しているデータを修正可能であるので、本対応が可能となる。

```
$ 割込み要求ライン初期化テーブル
$IF LENGTH(INT.ORDER_LIST)$
^    const INTINIB _kernel_intinib_table[TNUM_INTNO] = {$NL$
^    $JOINEACH intno INT.ORDER_LIST ",¥n"$
^        $TAB$[ ($INT.INTNO[intno]$), ($INT.INTATR[intno]$), ($INT.INTPRI[intno]$) ]
^    $END$$NL$
^    };$NL$
$ELSE$
^    TOPPERS_EMPTY_LABEL(const INTINIB, _kernel_intinib_table);$NL$
$END$$NL$
$END$
```

```
# 割込み要求ライン初期化テーブル
if ($hCfgData[:int].size() != 0)
  cKerCfgC.add("const INTINIB _kernel_intinib_table[TNUM_INTNO] = {")
  $hCfgData[:int].each{|cID, hParam|
    cKerCfgC.add("#{TAB}[ (#{hParam[:intno]}), (#{hParam[:intatr]}), (#{hParam[:intpri]}) ],")
  }
  cKerCfgC.chop_comma()
  cKerCfgC.add2("};")
else
  cKerCfgC.add2("TOPPERS_EMPTY_LABEL(const INTINIB, _kernel_intinib_table);")
end
```

図 5 JOINEACH と chop_comma の例

5.2.2 変数を使った条件式判定

Ruby は真偽の判定に 1 と 0 を使用することはできない。つまり、1 か 0 が代入されている変数をそのまま if 文の条件式に使用することができないため、明示的に一致するべき値との等式を記述する必要がある(図 6 左)。一方で、Ruby では、未定義のグローバル変数を参照すると nil が返り、nil を数値と比較すると偽と判定されるため、ALT 関数による変数定義チェックはしなくてもよいことになる(図 6 右)。

```
$ 割込み要求ラインの初期化に必要な情報
$IF !OMIT_INITIALIZE_INTERRUPT || ALT(USE_INTINIB_TABLE,0)$
```

```
# 割込み要求ラインの初期化に必要な情報
if (($OMIT_INITIALIZE_INTERRUPT == 0) || ($USE_INTINIB_TABLE == 1))
```

図 6 変数による条件式の例

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

6 現コンフィギュレータ trac のチケット

本資料作成時点で、4 章で述べたチケットを除いて、16 件のチケットが発行されていた。これらのチケットを分類し、新コンフィギュレータでの対応について検討する。

6.1 C++に依存する問題

Boost ライブラリの問題をはじめ、C++による開発に依存したチケットが 5 件である。これらは新コンフィギュレータを使用することで、すべて解決される。
※対象チケット：#135、#141、#146、#148、#149

6.2 Ruby であれば容易に解決可能な問題

Ruby であれば対応が容易なチケットが 3 件ある。#27 のカバレッジに関しては前述の通り、取得できることを確認済みである。#78 の GNU NM 以外のシンボルテーブル対応に関しても、Ruby によるシンボルテーブルパーサ部分を抜き出せば、ユーザが修正することが容易である。#119 の浮動小数点も、Ruby は標準でサポートしているので問題ない。文字列＋数値という形式で浮動小数点もサポートする場合、srec ファイルから読み込む処理が必要になるが、これも大きな問題ではない。
※対象チケット：#27、#78、#119

6.3 その他の問題

現コンフィギュレータ、新コンフィギュレータに関係なく対応が必要なもの、あるいは対応が容易なもの、仕様レベルで検討が必要なものが 8 件あった。いずれも新コンフィギュレータで対応が困難であるものは存在しない。
※対象チケット：#4、#5、#9、#25、#117、#123、#128、#147

7 現コンフィギュレータとの共存

Ruby による新コンフィギュレータのメリットが大きいことは確認できたが、既存の tf ファイルを trb ファイルに書き換えなければならない問題がある。また、TOPPERS カーネル開発者が、tf と trb を選択できるようにすることで、現コンフィギュレータとの振る舞いの違い等を確認し、新コンフィギュレータの評価を行えることも重要である。そこで、現コンフィギュレータと新コンフィギュレータを共存する方法について検討した。

結果、以下に挙げる 3 項目の修正のみで、対応可能であり、TTSP も configure スクリプトの引数に-R を与えるだけで、新コンフィギュレータによるテストが可能となった。

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

7.1 configure スクリプト

configure スクリプトには、”-R”オプションを新設し、このオプションを付与すると、cfg.exe ではなく、cfg.rb が Makefile の$(CFG)に設定されるようにする。

```
62 #  -r                    トレースログ記録のサンプルコードを使用するかどうか
63 #                        の指定
64 #  -p <perl>             perlのパス名（明示的に指定する場合）
65 #  -g <cfg>              コンフィギュレータ（cfg）のパス名
66 #  -P <num>              プロセッサ数（マルチプロセッサ対応カーネルの場合）
67 #  -o <options>          共通コンパイルオプション（COPTSに追加）
68 #  -O <options>          共通シンボル定義オプション（CDEFSに追加）
69 #  -k <options>          共通リンカオプション（LDFLAGS等に追加）
70
71 #  使用例(1)
72 #
73 #  % ../configure -T dve68k_gcc -D GDB_STUB -A perf1 -a ../test -U histogram.o
74 #
75 #  使用例(2)
```

```
62 #  -r                    トレースログ記録のサンプルコードを使用するかどうか
63 #                        の指定
64 #  -p <perl>             perlのパス名（明示的に指定する場合）
65 #  -g <cfg>              コンフィギュレータ（cfg）のパス名
66 #  -P <num>              プロセッサ数（マルチプロセッサ対応カーネルの場合）
67 #  -o <options>          共通コンパイルオプション（COPTSに追加）
68 #  -O <options>          共通シンボル定義オプション（CDEFSに追加）
69 #  -k <options>          共通リンカオプション（LDFLAGS等に追加）
70 #  -R                    Ruby版コンフィギュレータを使用するかどうかの指定
71
72 #  使用例(1)
73 #
74 #  % ../configure -T dve68k_gcc -D GDB_STUB -A perf1 -a ../test -U histogram.o
75 #
76 #  使用例(2)
```

```
201 else {
202     $srcabsdir = $srcdir = $pwd;
203 }
204
205 $perl = $opt_p ? $opt_p : get_path("perl", ("/usr/local/bin", "/usr/bin"));
206 $cfg = $opt_g ? $opt_g : "¥$(SRCDIR)/cfg/cfg/cfg";
207 $cfgfile = $opt_g ? $opt_g : $srcdir."/cfg/cfg/cfg";
208 $templatedir = $opt_t ? $opt_t : $srcdir."/sample";
209
210 #
211 #  -Tオプションの確認
212 #
213 unless ($opt_T) {
214     print STDERR "configure: -T option is mandatory¥n";
```

行: 213 列: 18/18 文字: 18/18 コードページ: 51932(euc-jp) Unix

```
202 else {
203     $srcabsdir = $srcdir = $pwd;
204 }
205
206 $perl = $opt_p ? $opt_p : get_path("perl", ("/usr/local/bin", "/usr/bin"));
207 $cfg = $opt_g ? $opt_g : $opt_R ? "¥$(SRCDIR)/cfg/cfg/cfg.rb" : "¥$(SRCDIR)/cfg
208 $cfgfile = $opt_g ? $opt_g : $srcdir."/cfg/cfg/cfg";
209 $templatedir = $opt_t ? $opt_t : $srcdir."/sample";
210
211 #
212 #  -Tオプションの確認
213 #
214 unless ($opt_T) {
215     print STDERR "configure: -T option is mandatory¥n";
```

行: 207 列: 1/90 文字: 1/90 コードページ: 51932(euc-jp) Unix

```
206 $cfg = $opt_g ? $opt_g : "¥$(SRCDIR)/cfg/cfg/cfg";
207
208
```

```
207 $cfg = $opt_g ? $opt_g : $opt_R ? "¥$(SRCDIR)/cfg/cfg/cfg.rb" : "¥$(SRCDIR)/cfg/cfg/cfg";
208
209
```

図 7 configure スクリプトの修正箇所

7.2 共通 Makefile

ベースとする Makefile には、$(CFG)の拡張子が”.rb”かどうかに応じて、テンプレートファイルの拡張子を trb か tf かに切り替える。また、tf 固定としていたテンプレートファイルの拡張子を環境変数$(TF_EXT)に置き換える。

```
96
97 #
98 #  ユーティリティプログラムの名称
99 #
100 PERL = @(PERL)
101 CFG = @(CFG)
102
103 #
104 #  オブジェクトファイル名の定義
105 #
106 OBJNAME = asp
```

```
306     $(OBJCOPY) -O srec -S $(CFG1_OUT) cfg1_out.srec
307     $(CFG) --pass 2 --kernel asp $(INCLUDES) ¥
308             -T $(TARGETDIR)/target.tf $(CFG_TABS) $<
309     touch -r kernel_cfg.c kernel_cfg.timestamp
310
311 #
312 #  カーネルライブラリファイルの生成
```

```
330     $(NM) -n $(OBJFILE) > $(OBJNAME).syms
331     $(OBJCOPY) -O srec -S $(OBJFILE) $(OBJNAME).srec
332     $(CFG) --pass 3 --kernel asp $(INCLUDES) ¥
333             --rom-image $(OBJNAME).srec --symbol-table $(OBJNAME).syms ¥
334             -T $(TARGETDIR)/target_check.tf $(CFG_TABS) $<
335
336 #
337 #  バイナリファイルの生成
338 #
339 $(OBJNAME).bin: $(OBJFILE)
```

```
96
97 #
98 #  ユーティリティプログラムの名称
99 #
100 PERL = @(PERL)
101 CFG = @(CFG)
102
103 #
104 #  テンプレートファイルの拡張子選択
105 #
106 ifeq ($(suffix $(CFG)),.rb)
107     TF_EXT = trb
108 else
109     TF_EXT = tf
110 endif
111
112 #
113 #  オブジェクトファイル名の定義
114 #
115 OBJNAME = asp
```

```
315     $(OBJCOPY) -O srec -S $(CFG1_OUT) cfg1_out.srec
316     $(CFG) --pass 2 --kernel asp $(INCLUDES) ¥
317             -T $(TARGETDIR)/target.$(TF_EXT) $(CFG_TABS) $<
318     touch -r kernel_cfg.c kernel_cfg.timestamp
319
320 #
321 #  カーネルライブラリファイルの生成
```

```
339     $(NM) -n $(OBJFILE) > $(OBJNAME).syms
340     $(OBJCOPY) -O srec -S $(OBJFILE) $(OBJNAME).srec
341     $(CFG) --pass 3 --kernel asp $(INCLUDES) ¥
342             --rom-image $(OBJNAME).srec --symbol-table $(OBJNAME).syms ¥
343             -T $(TARGETDIR)/target_check.$(TF_EXT) $(CFG_TABS) $<
344
345 #
346 #  バイナリファイルの生成
347 #
348 $(OBJNAME).bin: $(OBJFILE)
```

図 8 共通 Makefile の修正箇所

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.

7.3 依存部 Makefile

アーキテクチャ依存部、ターゲット依存部の Makefile では、tf 固定としていたテンプレートファイルの拡張子を環境変数$(TF_EXT)に置き換える。

```
28 #
29 #  依存関係の定義
30 #
31 kernel_cfg.timestamp: $(COREDIR)/core.tf
32 $(OBJFILE): $(COREDIR)/core_check.tf
33
34 #
35 #  コンフィギュレータ関係の変数の定義
36 #
37 CFG_TABS := $(CFG_TABS) --cfg1-def-table $(COREDIR)/core_def.csv
38
39 #
40 #  オフセットファイル生成のための定義
41 #
42 OFFSET_TF = $(COREDIR)/core_offset.tf
43
```

```
28 #
29 #  依存関係の定義
30 #
31 kernel_cfg.timestamp: $(COREDIR)/core.$(TF_EXT)
32 $(OBJFILE): $(COREDIR)/core_check.$(TF_EXT)
33
34 #
35 #  コンフィギュレータ関係の変数の定義
36 #
37 CFG_TABS := $(CFG_TABS) --cfg1-def-table $(COREDIR)/core_def.csv
38
39 #
40 #  オフセットファイル生成のための定義
41 #
42 OFFSET_TF = $(COREDIR)/core_offset.$(TF_EXT)
43
```

```
79 #
80 #  依存関係の定義
81 #
82 cfg1_out.c: $(TARGETDIR)/target_def.csv
83 kernel_cfg.timestamp: $(TARGETDIR)/target.tf
84 $(OBJFILE): $(TARGETDIR)/target_check.tf
85
86 #
87 #  オフセットファイル生成のための定義
88 #
89 OFFSET_TF := $(TARGETDIR)/target_offset.tf
90
91 #
92 #  プロセッサ依存部のインクルード
93 #
94 include $(SRCDIR)/arch/$(PRC) $(TOOL)/common/Makefile.core
```

```
79 #
80 #  依存関係の定義
81 #
82 cfg1_out.c: $(TARGETDIR)/target_def.csv
83 kernel_cfg.timestamp: $(TARGETDIR)/target.$(TF_EXT)
84 $(OBJFILE): $(TARGETDIR)/target_check.$(TF_EXT)
85
86 #
87 #  オフセットファイル生成のための定義
88 #
89 OFFSET_TF := $(TARGETDIR)/target_offset.$(TF_EXT)
90
91 #
92 #  プロセッサ依存部のインクルード
93 #
94 include $(SRCDIR)/arch/$(PRC) $(TOOL)/common/Makefile.core
```

図 9 依存部 Makefile の修正箇所

8 今後の展望

新コンフィギュレータの開発により、tf の利便性を損なうこと無く、Ruby 版のコンフィギュレータへ移行できることを確認できた。また、現コンフィギュレータと共存可能であることも確認できた。TOPPERS ソフトウェア開発者の中で、新コンフィギュレータに対するニーズがあれば、TOPPERS 運営委員の活動の一環として、新コンフィギュレータの開発を行いたいと考えている。必要に応じて、tf を trb へ変換するスクリプトの開発等も検討したい。

第 3 世代カーネルである ASP3 カーネルから、現コンフィギュレータと新コンフィギュレータを共存し、開発者が tf と trb を選択できる運用として頂けると幸いである。今後、FMP3 カーネル、HRP3 カーネル開発の際にはそれぞれのカーネルにも対応していきたい。

以上

Copyright ©2015 FUJI SOFT INCORPORATED, All rights reserved.