Vol.2011-SLDM-149 No.13
Vol.2011-EMB-20 No.13
2011/3/18

マルチプロセッサ対応 RTOS に対する API テストの実施

金　榮柱[†1]　金　スンヨプ[†1]　金　ハンソル[†1]
金　賢敏[†1]　竹谷美里[†2]　木村貴寿[†3]
鴫原一人[†4]　森　孝夫[†4]　本田晋也[†4]
山本雅基[†4]　高田広章[†4]

我々は，マルチプロセッサ対応 RTOS が提供する API の信頼性を検証するためのテストを実施した．テスト対象とする RTOS は，TOPPERS/FMP カーネルとした．本論文では，API テストの設計思想，検出した不具合，およびテストの有用性について述べる．

The API Testing for Multiprocessor RTOS

Youngjoo KIM,[†1] Seungyup KIM,[†1] Hansol KIM,[†1]
Hyoungmin KIM,[†1] Misato TAKEYA,[†2]
Takatoshi KIMURA,[†3] Kazuto SHIGIHARA,[†4]
Takao MORI,[†4] Shinya HONDA,[†4]
Masaki YAMAMOTO[†4] and Hiroaki TAKADA[†4]

We carried out API testing for multiprocessor RTOS to verify the reliability of them. The object RTOS was TOPPERS/FMP kernel. This paper introduces design concept of the API testing and detected bugs.And then, we describe the consideration on the effectiveness of this test.

†1 株式会社デジタルクラフト
†2 富士ソフト株式会社
†3 日本電気通信システム株式会社
†4 名古屋大学

1. はじめに

名古屋大学大学院情報科学研究科附属組込みシステム研究センター (NCES)[1] は，組込みシステムの分野においてもマルチプロセッサの需要が高まっていることに対応し，マルチプロセッサ対応 RTOS の検証手法を研究するコンソーシアム型の研究組織を立ち上げた．本コンソーシアムでは，最初に API に着目したテスト (以下，API テスト) を実施した．対象としたマルチプロセッサ対応 RTOS は，TOPPERS/FMP カーネル (以下，FMP カーネル) である．

API テストでは，多様なマルチプロセッサシステムのハードウェアアーキテクチャに対応するため，テストプログラム生成ツールを開発し，様々なターゲットシステムに対応するテストを可能とした．API テストを，シミュレータと実機で実施し，FMP カーネルの不具合を 48 件検出した．我々は，このテストからマルチプロセッサ対応 RTOS に特有の不具合の傾向を掴み，それらを検出するための知見を獲得することができた．

本論文では，実施した FMP カーネルの API テストの結果について述べる．本論文の構成は，次の通りである．2 章で TOPPERS 新世代カーネルや FMP カーネルについて述べ，3 章で実施した API テストについて述べる．4 章では FMP カーネルに対するテスト実施結果について述べ，5 章で結果に対する評価と考察を述べる．

2. TOPPERS 新世代カーネル

TOPPERS 新世代カーネルとは，TOPPERS プロジェクト[2] において μITRON 仕様[3] をベースとして開発している一連の RTOS の総称である．TOPPERS 新世代カーネルは，TOPPERS/ASP カーネル (以下，ASP カーネル) をベースとして，メモリ保護，マルチプロセッサ，カーネルオブジェクトの動的生成，機能安全などに対応した一連のカーネルで構成される．仕様は，「TOPPERS 新世代カーネル統合仕様書」[4](以下，統合仕様書) としてまとめられている．

2.1 ASP カーネル

ASP カーネルは，TOPPERS 新世代カーネルの基盤となるシングルプロセッサ対応 RTOS である．ASP カーネルに実装された API は，カーネル動作中に呼び出し可能な API(以下，API) が 104 個，あらかじめシステムコンフィギュレーションファイルに記述してオブジェクト生成に用いる API(以下，静的 API) が 17 個の，計 121 個である．

Vol.2011-SLDM-149 No.13
Vol.2011-EMB-20 No.13
2011/3/18

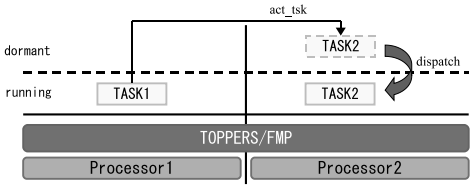

図 1 act_tsk の実行イメージ
Fig. 1 Run image of act_tsk

2.2 FMP カーネル

FMP カーネルは，ASP カーネルを拡張することにより開発された，マルチプロセッサ対応 RTOS である．FMP カーネルには，ASP カーネルに対して，API が 17 個，静的 API が 1 個追加され，計 139 個の API および静的 API が存在する．

API のテスト設計時に考慮すべき仕様について 3 点を述べる．

2.2.1 既存の API のマルチプロセッサ拡張

ASP カーネルの API の多くは，FMP カーネルでは異なるプロセッサに割り付けられたオブジェクトの操作が可能なように拡張されている．例えば，act_tsk は，マルチプロセッサ拡張により，発行元とは異なるプロセッサに割り付けられているタスクを起動可能とする．act_tsk の実行イメージを**図 1** に示す．

2.2.2 FMP カーネルにおいて追加された新たなタスクの状態

ASP/FMP カーネルでは，カーネルがディスパッチを行わないディスパッチ保留状態が存在する．ディスパッチ保留状態では，実行状態のタスクが強制待ち状態に遷移することはない．しかし FMP カーネル上で動作するタスクは，異なるプロセッサに割付けられているタスクに対して操作ができる．また，ディスパッチ保留状態はプロセッサ毎に独立して管理される．そのため，ディスパッチ保留状態のプロセッサにある実行状態のタスクに対して，異なるプロセッサから強制待ち状態へ遷移させる API を呼び出した場合，実行状態であったタスクのディスパッチも保留される．この間，タスクは実行され続けるが，カーネル管理上のタスクの状態は強制待ちとなっている．FMP カーネルでは，このような過渡的な状態が有り得るため，強制待ち状態［実行継続中］(以下，過渡状態) というタスクの状態を追加した．**図 2** に過渡状態を含めた状態遷移図を示す．

2.2.3 多様なバリエーションへの対応

マルチプロセッサシステムには，プロセッサの種類や数，メモリ，タイマ，プロセッサ間排他制御方法等に関して，様々なハードウェアアーキテクチャが存在し，RTOS ではそれらのアーキテクチャを出来る限りサポートすることが求められる．

FMP カーネルでは，このようなマルチプロセッサシステムのハードウェアアーキテクチャの違い (以下，バリエーション) に対し，様々なバリエーションに対応できるように実装されている．

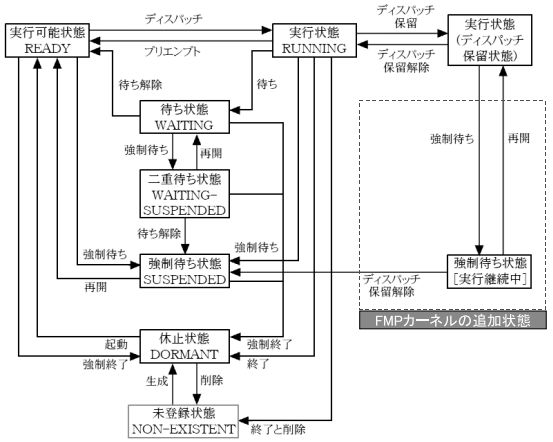

図 2 タスクの状態遷移図
Fig. 2 State Transition Diagram of Task

3. API テスト

3.1 API テストの概要

API テストとは，ASP カーネルおよび FMP カーネルが提供している 139 個の API が，統合仕様書に定められている通りに正しく振舞うことを確認するテストである．API 発行後の振る舞いは，その多くが，CPU ロック状態やディスパッチ禁止状態などのカーネルの状態や，各オブジェクトの状態に影響を受ける．また，API 発行後，それらの状態が変化する．そこで，API テストでは，カーネル状態およびオブジェクトの状態を特定の状態にして API を発行し，仕様通りの状態の変化が起きることを確認する．

API テストから着手した理由は，API が仕様通りに動作することを前提に，ユーザアプリケーションが開発されることや，API の実装が RTOS のソースコードの大部分を占めて

Vol.2011-SLDM-149 No.13
Vol.2011-EMB-20 No.13
2011/3/18

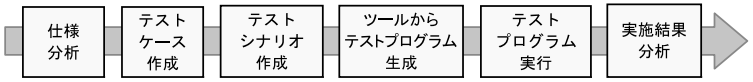

図 3 API テストの流れ図
Fig. 3 Flow of API Test

いることから，RTOS が保証するべき最も重要な要素だからである[5]．

3.2 API テストの流れ

API テストでは，最初に，統合仕様書に記述されている API の仕様を分析し，テストケースを抽出する．次に，抽出されたテストケースを時間経過に沿った振る舞いの形で具体化し，テストシナリオを作成する．テストシナリオを実行するテストプログラムは，テストプログラム生成ツール (TTG)[6] を用いて自動生成する．**図 3** に API テストの流れを示す．

3.3 テストケース

テストケースは，統合仕様書に記載されている全 API の仕様のカバレッジ (以下，仕様カバレッジ) を 100%とするように抽出する．統合仕様書は，抽象的な表現で記述されているため，テストケースの作成者によって，抽出するテストケースにばらつきが生じる可能性がある．そのため，テストの実施範囲や同値分割の粒度などのポリシーを定め，API 毎に隔たりなく，テストケースを抽出する．

さらに，抽出した全テストケースを実行することによって，RTOS のソースコードカバレッジが 100%となることを確認する．なお，不具合の検出率を向上させる目的で，命令網羅[⋆1]ではなく，条件網羅[⋆2]によるソースコードカバレッジを確認する．

テストケースの例を**図 4** に示す．この例は，休止状態のタスクを起動する API である act_tsk が，他プロセッサのタスクに対して正しく動作することを確認するためのテストケースの 1 つである．

3.4 テストシナリオ

テストシナリオは，仕様から抽出したテストケースを，具体化したものである．3.1 節で述べたように，API テストでは，API 発行によるシステム状態の変化を確認する．具体的には，テストプログラムにて，API 発行前のシステム状態 (前状態) を実現し，その状態でテスト対象となる API を発行 (処理) し，API 発行後のシステム状態 (後状態) を確認する．

⋆1 すべての命令を少なくとも 1 回は実行する
⋆2 すべての条件式の判定による真偽を少なくとも 1 回は実行する

他プロセッサに割り付けられている対象タスクの状態が休止状態である場合は，対象タスクに対してタスク起動時に行うべき初期化処理が行われ，対象タスクは実行できる状態になること．

図 4 テストケースの例
Fig. 4 Example of a Test Case

前状態
　プロセッサ 1 の優先度 (中) の TASK1 が実行状態
　プロセッサ 2 の優先度 (中) の TASK2 が実行状態
　プロセッサ 2 の優先度 (高) の TASK3 が休止状態

処理
　TASK1 が act_tsk(TASK3) を発行し，
　エラーコードとして E_OK が返る

後状態
　TASK2 が実行可能状態となる
　TASK3 が実行状態となる

図 5 テストシナリオの例
Fig. 5 Example of a Test Scenario

```
pre_condition:
 TASK1:
  type   : TASK
  tskstat: running
  prcid  : 1
 TASK2:
  type   : TASK
  tskpri : TSK_PRI_MID
  tskstat: running
  prcid  : 2
 TASK3:
  type   : TASK
  tskpri : TSK_PRI_HIGH
  tskstat: dormant
  prcid  : 2

do:
 id     : TASK1
 syscall: act_tsk(TASK3)
 ercd   : E_OK

post_condition:
 TASK2:
  tskstat: ready
 TASK3:
  tskstat: running
```

図 6 TESRY データの例
Fig. 6 Example of a TESRY Data

このようにテストプログラムの実行内容を「前状態」,「処理」,「後状態」の 3 つの項目に分けて表現したものを，API テストのテストシナリオとする．

図 4 のテストケースの例を，テストシナリオで表現したものを**図 5** に示す．プロセッサ 2 に優先度の低い TASK2 と高い TASK3 が存在して，前状態では，TASK2 が実行されており (実行状態)，TASK3 が休止状態である．処理では，プロセッサ 1 にある実行状態の TASK1 がプロセッサ 2 にある TASK2 に対して act_tsk を発行し，戻り値としてエラーコード E_OK を受け取る．後状態では，処理の結果として，TASK3 のほうが優先度が高いため，ディスパッチが発生し，TASK3 に処理が切り替わり (実行状態)，TASK2 は実行待ち (実行可能状態) となることを定義している．

Vol.2011-SLDM-149 No.13
Vol.2011-EMB-20 No.13
2011/3/18

3.5 テストプログラムの生成

テストプログラムは TTG を使用して生成する．TTG の入力には，テストシナリオを YAML 形式[7] で記述したものを使用する．YAML 形式によるテストシナリオの表現方法を TESRY(TEst Scenario for Rtos by Yaml) 記法と呼び，TESRY 記法でテストシナリオを記述したものを TESRY データと呼ぶ．

図 5 のテストシナリオを，TESRY 記法で記述したものを，**図 6** に示す．TESRY 記法では，システム状態を，前状態として “pre_condition” 内に，後状態として “post_condition” 内に記述し，発行する API とその引数と，期待する戻り値を “do” 内に記述する．

3.6 テストケースの取捨選択

2.2.3 節で述べたように，FMP カーネルのテストにおいては，ターゲットの仕様に合わせて，実施するテストケースを取捨選択する必要がある．TTG では TESRY データの内容から実行可能かを判断してテストケースを取捨選択する機能を設けて，ターゲットに区別されず，テストが可能になった．

例えば，タイマ方式がグローバルタイマ方式のみサポートされるターゲットをテストする場合は，システム設定ファイルにターゲットがグローバルタイマ方式をサポートすることを指定することで，テストプログラム生成時にローカルタイム方式でのみ実施可能な TESRY データは自動的に間引かれる．TTG による TESRY データの取捨選択のイメージを**図 7** に示す．

また，一部のテストケースでは，生成したテストプログラム内で特別なテストライブラリが必要となる．例えば，API 発行時に指定したシステム時刻に正しく処理が実行されることを確認するテストケースでは，指定したシステム時刻におけるシステム状態を確認する必要がある．このようなテストケースを実現するためには，システム時刻を制御する関数が必要である．また，テストケースによっては，意図的な割込みの発生や意図的な CPU 例外の発生などが必要となることがある．

TTG は，これらのテストライブラリが準備されている前提でテストプログラムを生成するが，システム時刻や割込み，CPU 例外の操作は，ターゲットシステムによって実装が異なるため，テスト対象とするターゲットシステム毎にテストライブラリを実装する必要がある．ただし TTG では，テストライブラリの有無をシステム設定ファイルに定義することで，テストライブラリがない場合でも実行できるテストケースだけを取捨選択することも可能である．

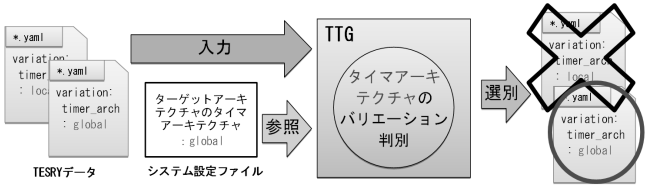

図 7 TTG による TESRY データの取捨選択
Fig. 7 Selection of TESRY Data by TTG

4. テスト結果

我々は，2009 年度の後半から 2010 年度にかけて，FMP カーネルの API テストを実施した．本章では，実施した API テストの結果を示す．

4.1 概　　要

テスト対象は，FMP カーネルのすべての API(静的 API を含む) である．FMP のテストケースは，ASP カーネルの API テストケースも流用[⋆1] する．作成したテストケース数は 4,240 件である．実施の際に使用したプラットフォームは，シミュレータの SkyEye[8]，実機の NaviEngine[9] である．なお，それぞれのプラットフォーム毎に，バリエーションの設定パターンをいくつか用意してテストを行った．

生成されたテストプログラムは 694,259 行であった．以下，使用したプラットフォーム毎にテスト結果を示す．

4.2 SkyEye でのテスト結果

本テストでは，TOPPERS カーネル向けに拡張した SkyEye[10] を使用した．SkyEye はシミュレータであるため，FMP カーネルでサポートする様々なバリエーションのテストが可能である．そのため，4,240 件のすべてのテストケースを実行することができた．SkyEye で実施したバリエーションの設定パターンを**表 1** に示す．

SkyEye によるテストで検出された不具合は 79 件であった．なお，以下の実行環境における，1 つのバリエーションに対するテストの実行時間は約 3 分であった．

- Intel Core2 Quad 2.66GHz

⋆1 シングルプロセッサで行われるテストケースをマルチプロセッサではプロセッサ 1 つのみ使用されるテストケースとして扱う

Vol.2011-SLDM-149 No.13
Vol.2011-EMB-20 No.13
2011/3/18

表 1 SkyEye におけるバリエーションの設定パターン
Table 1 Variation Pattern for SkyEye

タイマ方式	グローバル	ローカル	
ロック方式	-	プロセッサ/ジャイアント	
スピンロック方式	ネイティブ/エミュレート	ネイティブ/エミュレート	
マスタプロセッサ ID	1/2/3/4	1/2/3/4	
システム時刻管理プロセッサ ID	1/2/3/4	-	
バリエーション設定パターン数	32	16	= 合計：48

- RAM3.21GB
- WindowsXP SP3
- Cygwin(1.7.5)

4.3 NaviEngine でのテスト結果

実機ではプロセッサに ARM11 コアを 4 つ内蔵した NaviEngine を使用した．NaviEngine では，SkyEye でのテストでは検出されなかった 5 件の不具合が新たに検出された．

NaviEngine におけるバリエーションは以下である．

- プロセッサ数: 4
- タイマ方式: ローカル
- ロック方式: プロセッサ
- スピンロック方式: ネイティブ
- マスタプロセッサ ID: 1

上記バリエーションにおけるテストケース数は，4,223 件であった．テスト実行時間は約 7 分 45 秒であった．

5. 結果の分析と考察

5.1 FMP カーネルの不具合の分類

API テストで検出した不具合の件数と分類を**表 2** 示す．さらに FMP カーネルの不具合 48 件は，以下のように分類できる．

- ソースコメント誤記 ：1 件
- ターゲット非依存部の不具合 ：34 件
- ターゲット依存部の不具合 ：13 件

5.2 特徴的な不具合の考察

本節では，マルチプロセッサの特徴的な不具合と思われる 2 つの事例に対して考察する．

表 2 FMP カーネルの不具合検出件数
Table 2 Detection number of defect of FMP Kernel

	2009 年度	2010 年度	合計
統合仕様書	15	18	33
コンフィギュレータ	2	1	3
FMP カーネル	11	37	48
合計	28	55	84

5.2.1 不要なソースコードの混入

不要なソースコードの混入は，3 件検出された．マルチプロセッサ対応 RTOS では，レースコンディションによって様々な状況が発生する．例えば，異なるプロセッサ上のタスクの状態を操作する API を実行中に，別のプロセッサから対象のタスクを操作する API を発行した場合，処理の実行順序によって結果が異なる．しかし，RTOS はどのようなタイミングで複数の API が発行されたとしても，一貫性と生存性を保つ必要がある．このため，RTOS のソースコードは発生しうる状況に合わせて，多くの分岐処理を必要とする．しかし，フェールセーフを求める RTOS の設計思想においては，より安全な動作となるように条件分岐を設けるので，不要な条件分岐の混入が起きやすくなると考えられる．

不要なソースコードが混入する不具合は，全テストケースに対するソースコードカバレッジを分析することで検出した．まず，仕様カバレッジを 100%としたすべてのテストケースを実行した結果として得られるソースコードカバレッジを確認する．ソースコードカバレッジが 100%となっていない箇所が存在する場合，仕様に現れない実装依存のソースコードであるか，仕様の記述漏れである可能性がある．表 2 における統合仕様書の不具合が後者に当たる．前者に対しては，ホワイトボックステストとして新たにテストケースを追加した．ホワイトボックステストでも網羅できないと判定されたパスは，不要なソースコードと判断することができる．

5.2.2 実行タイミングに依存して発生するエラー

実行タイミング依存で発生する不具合は，4 件検出された．仕様ベースのテスト設計では期待結果が決定的動作となるもののうち，実際には期待結果が非決定的動作となるケースが存在した．ここでは，タスクを休止状態とする API である ext_tsk に検出した不具合の例で説明する．2.2.2 節で述べたように，タスクには，カーネル管理上は強制待ち状態であるが，処理は継続している過渡状態が存在する．過渡状態のタスクから ext_tsk を発行した場合でも，タスクは休止状態となるべきであるが，実行タイミングによって休止状態とな ら

ず，強制待ち状態となる不具合が検出された．

本不具合の発生原因は以下である．ext_tsk を始め，実行状態のタスクを実行状態以外の状態へ遷移させる API では，対象タスクが割り付けられているプロセッサの排他制御用ロックを取得してから処理を開始する．排他制御用ロックを他のプロセッサ上のタスクが取得していた場合は，リトライのためにループ処理を行う．ここで，FMP カーネルでは割込み応答性を上げるため，ループ処理内で微小時間であるが割込みを許可する期間を設けている．しかし，これが不具合が原因で，微小時間内にタイマ割込みなどの割込みを受け付けると，カーネルの処理においてディスパッチが発生し，過渡状態であったタスクは ext_tsk 実行中に強制待ちとなってしまう．

SkyEye においては，上記現象の再現率は約 25%であった．このような不具合は，決定的な結果を期待して設計されたテストケースでは，一度 ext_tsk のテストを実施しただけでは，検出することが困難である．これに対し，我々は全テストケースを連続試行的に実施することで検出した．

5.3 RTOS のテストスイートとしての有用性

RTOS の実装は，ターゲット依存部とターゲット非依存部に大別される．ターゲット非依存部は，RTOS を適用するターゲットシステムのハードウェア仕様に依存しない実装である．ターゲット依存部は，ターゲット非依存部をあらゆるターゲットシステムで共通的に使用可能とするためのターゲットシステムに依存した実装である．

5.2.1 節で述べたソースコードカバレッジ測定の対象は，ターゲット非依存部のみである．これは，ターゲット依存部はターゲットシステム毎に存在するため，カバレッジを測定することに限界があるためである．しかし，5.1 節で述べたように，API テストにおいてもターゲット依存部の不具合を 13 件検出した．これは，API の実装からターゲット依存部を呼び出しているため，API が仕様通りに振る舞うことをテストすることで，ターゲット依存部が正しく実装されていることも同時にテストできているためである．

つまり，新たなターゲットシステムに FMP カーネルをポーティングする際に，本研究の API テストは，ターゲット依存部を正しく実装できているかの検証にも有用である．また，RTOS ではコンパイラのバージョンアップ等による不具合の事例があるように，実行モジュールに何らかの変更があった場合も，API テストは活用できると考えられる．

6. ま と め

本研究では，2009 年度の後半から FMP カーネルに対する API テストを実施した．マルチプロセッサシステムでは，多様なバリエーションがあるため，様々なバリエーションが対応できるようにテストプログラム生成ツールを開発した．API テストは，シミュレータと実機で実施した．その結果，FMP カーネルの不具合を 48 件検出した．

我々は，このテストからマルチプロセッサ対応 RTOS に特有な不具合の傾向を掴み，それらを検出するための知見を得た．1 つは，マルチプロセッサ対応 RTOS には不要な条件分岐が入りやすく，それに対して仕様カバレッジとソースコードカバレッジを 100%とする手法が有効であるという示唆である．もう 1 つは，仕様では決定的動作をすると規定されているにも関わらず，実際にはレースコンディションによって非決定的動作をするという不具合が混入することがあり，それに対して連続試行テストが有効であるという示唆である．

今後の課題として，統合仕様書や FMP カーネルが変更された場合，API テストとのトレーサビリティを確保することが求められるが，現在は未対応である．統合仕様書や FMP カーネル，テストケース，テストプログラムはすべてテキストファイルで管理されているため，今後，このファイル間のトレーサビリティを保つためのツールについて研究する予定である．

参 考 文 献

1) 名古屋大学大学院情報科学研究科附属組込みシステム研究センター, http://www.nces.is.nagoya-u.ac.jp/
2) TOPPERS プロジェクト, http://www.toppers.jp/
3) 坂村健/監修, 高田広章/編, μITRON4.0 仕様 Ver.4.02.00, トロン協会 (2004)
4) TOPPERS 新世代カーネル統合仕様書, http://www.toppers.jp/docs/tech/ngki_spec-120.pdf
5) 鴫原一人, 松浦光洋, 金ハンソル, 金スンヨプ, 馬鋭, 廉正烈, 金榮柱, 木村貴寿, 眞弓友宏, 本田晋也, 山本雅基, 高田広章, “組込みリアルタイム OS の API テストの実施, ” ソフトウェアテストシンポジウム 2010, pp.1-8, 2010.
6) 鴫原一人, 眞弓友宏, 本田晋也, 高田広章, “マルチプロセッサ対応 RTOS を対象としたテストシナリオ記述法とテストプログラム生成ツール, ” 情報処理学会研究会 (Vol.2010-EMB-18 No.1) , pp.1-8, 2010.
7) YAML, http://yaml.org/
8) SkyEye, http://www.skyeye.org/
9) NaviEngine, http://www2.renesas.com/automotive/ja/assp/naviengine.html
10) 安積卓也, 古川貴士, 相庭裕史, 柴田誠也, 本田晋也, 冨山宏之, 高田広章, ” オープンソース組込みシステム向けシミュレータのマルチプロセッサ拡張”, コンピュータソフトウェア, Vol.27, No.4, pp.24-42, Nov. 2010.