コンソーシアム型共同研究による マルチプロセッサ向けRTOSのテストスイート開発

森　孝夫[†]　鴫原一人[†]
本田晋也[†]　高田広章[†]

本論文では，我々が実施したマルチプロセッサ向けRTOSに関するコンソーシアム型共同研究の結果について，テストスイートの開発を中心に報告する．本研究は，マルチプロセッサ向けRTOSのテスト手法の確立，および組込みシステム用のマルチプロセッサ向けRTOSであるTOPPERS/FMPカーネルの実際のテスト実施を目的として，2009年度から2010年度までの2年間行われた．2年間の活動で，マルチプロセッサ向けRTOSに関するテストの全体像を設計し，RTOSのAPIに関するテストを完了するとともに，テストプログラム自動生成ツールやマルチプロセッサ向けRTOS環境で使用できるテストライブラリを開発した．また，本研究で開発したテストスイートは，ツールやライブラリとともに2011年度にオープンソースとして一般公開した．ここで蓄積された知見は，本研究に参加した企業にとって有用な知見となるとともに，名古屋大学が現在進めている次世代車載システム向けRTOSに関するコンソーシアム型共同研究にも活かされている．

Development of test suites for multiprocessor RTOS in a consortium collaborative research

Takao Mori,[†] Kazuto Shigihara,[†] Shinya Honda[†] and Hiroaki Takada[†]

The present paper reports the result of our collaborative research on RTOS for multiprocessor system, especially development of test suites. The goal of the research was the establishment and implication of a test methodology for TOPPERS/FMP kernel. During the research period (2009-2010), the overall testing strategy for multiprocessor RTOS was designed, and API testing for RTOS was conducted. A test program generator and test library for multiprocessor environments were also developed. The test suites developed based on the findings of the collaborative research have been publicized in the fiscal year of 2011 as an open source.

Findings of the present research have provided beneficial information to the member companies of the consortium collaborative research, and the findings have applied to another consortium collaborative research project on the next generation RTOS for automotive, which is currently undergoing at Nagoya University.

1. はじめに

近年，組込みシステムにおいてもマルチプロセッサが利用されるようになりつつある．その理由として，発熱を抑えつつ性能を向上させるためには，高いクロックでシングルプロセッサを用いるよりも，マルチプロセッサを用いた方が有利だということが挙げられる．マルチプロセッサの利用が広がるにつれて，マルチプロセッサ向けリアルタイムOS(RTOS)の必要性も高まっている．こうした必要性の高まりを受け，我々は，オープンソースのマルチプロセッサ向けRTOSである，TOPPERS/FMPカーネル(以下，FMPカーネル)の開発を進めている[1)]．FMPカーネルは，産業界で広く利用されているオープンソースのシングルプロセッサ向けRTOSであるTOPPERS/ASPカーネル(以下，ASPカーネル)の拡張である．

FMPカーネルは産業界での利用が想定されており，品質の確保は非常に重要である．そのため，利用前のテストは不可欠なアクティビティであると考えられる．しかし，これまでオープンソースのRTOSのテストは，多くの場合ユーザによって実施されており，オープンソースを用いることで工数を削減しようとする

† 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
Center for Embedded computing Systems, Graduate School of Information Science, Nagoya Univ

ユーザの狙いが達成されていない面があった．また，企業がテストを実施する場合，テスト設計の内容は当然クローズとなるので，マルチプロセッサ向け RTOS に関するテスト設計方法論，プロセスや知見はこれまであまりオープンにされてこなかった．

そこで，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 (以下，NCES) は，コンソーシアム型研究という新しい研究スキームを提案し，2009 年，複数の企業の協力を得て，マルチプロセッサ向け RTOS に関するコンソーシアムを立ち上げた．本研究の目的は，マルチプロセッサ向け RTOS に関するテスト手法を研究し，実際にマルチプロセッサ向け RTOS である FMP カーネルのテストを実施すると共に，テスト実施を通じて得られた知見，およびテスト実施に用いるテストスイートをオープンソースとして公開することである．

これらの目的を受けて，我々は ASP カーネルと FMP カーネルのテストを実施し，合計 98 件の不具合を発見すると共に，テスト実施に必要なテストケース記述記法，テストプログラム生成ツール，およびマルチプロセッサ環境のテストで必要となる同期手法を開発した．

本報告では，我々が 2 年間実施したマルチプロセッサ向け RTOS のテスト手法に関する研究，実施したテストの内容，および結果を報告し，それらを通じて得た知見を述べる．

本論文の構成は次の通りである．2 章では，まず本研究の概要について述べる．3 章では，ASP カーネルと FMP カーネルのテスト設計とテストプロセスを説明する．4 章では本研究で開発されたテスト手法について説明する．5 章では，テスト結果および検出された不具合の事例について述べる．6 章では，本研究の主要な成果物である TTSP について説明する．7 章では，全体をまとめた上で，現在進めている新しいコンソーシアム型共同研究を紹介する．

2. 研究の概要

2.1 コンソーシアム型共同研究のスキーム

コンソーシアム型共同研究では，NCES がいくつかの研究テーマを候補として提示し，共同研究に参加したい企業を複数募り，研究コンソーシアムを形成して研究・開発を行う．コンソーシアムは通常，年度毎に形成される．年度の初期には，候補となる研究テーマの中から具体的にどのテーマを実施するかが議論され，参加企業にとって興味のあるテーマが選択される．成果物の知財は，部分毎に担当した企業が保有し，他の参加企業はそれを自由に使用できるものとする．研究開発後は，参加企業の同意が得られる限り，一定期間後に成果を公開することを基本方針としている．成果を公開する時期は，通常，各年度の終了時より一年後である．

マルチプロセッサ向け RTOS に関するコンソーシアム型共同研究は，コンソーシアム型共同研究のスキームを適用した最初の研究プロジェクトであり，2009 年度から 2010 年度までの 2 年間実施された．2009 年度は，7 社および 1 公的機関からの参加があった．2010 年度は，5 社および 1 公的機関からの参加があった．

また，ツール開発の一部は，「OJL による最先端技術適応能力を持つ IT 人材育成拠点の形成」[5),11)] の OJL(On the Job Learning) のテーマとなり，学生の参加もあった．

2.2 本研究の内容

本研究の活動は，大きく 2 つに分けられる．1 つは，ASP カーネルと FMP カーネルのテスト実施である．テスト実施は，テスト設計，テストスイートの実装，テストの実行を含む．もう 1 つは，マルチプロセッサ向け RTOS のテスト実施に必要な技術や手法の研究開発である．

テスト実施では，最初に，ASP カーネルと FMP カーネルで実施すべきテストの全体像を設計した．その中で，API に関するテストを優先的に実施し，以後工数の範囲内で優先度をつけてテストを実施した．

また，テスト実施の過程で直面した問題を解決していくことで，マルチプロセッサ向け RTOS のテスト実施に必要な技術や手法を開発した．開発した技術や手法としては以下が挙げられる．

- API テストにおけるテストケース設計ポリシー[12),13),17)]
- テストシナリオを記述する TESRY 記法[15)]
- TESRY 記法で記述されたテストケースからテストプログラムを自動生成するツール TOPPERS Test Generator(TTG)[15)]
- テストライブラリ
- テストプログラムによるプロセッサ間同期の手法[18)]
- マルチプロセッサの実行タイミングを制御することのできる拡張 Instruction Set Simulator(ISS)[19)]

このように，本研究では，ASP カーネルと FMP カーネルのテスト実施，問題発見と解決のサイクルを繰り返すことで技術開発が進んでいった．

2.3 対象の RTOS

本研究の対象 RTOS は，ASP カーネルと FMP カー

ネルである．これらはいずれも，TOPPERS 新世代カーネルに位置づけられており，前者はシングルプロセッサ向けの RTOS，後者はそれを拡張したマルチプロセッサ向ッサ向けの RTOS である．それらの仕様は，TOPPERS 新世代カーネル統合仕様書[2](以下，統合仕様書) としてまとめられている．

ASP カーネルと FMP カーネルのソースコードはいずれも，ターゲット非依存部とターゲット依存部に分類される．ターゲット非依存部とは，カーネルのソースコードのうち，カーネルが動作するプロセッサ環境に依存しない箇所のことである．

次に，ASP カーネルと FMP カーネルの規模に関する情報を示す．ASP カーネルのターゲット非依存部のソースコードの行数は，ARM 用パッケージの場合，約 9,500 行である．また，ASP カーネルは，API として，カーネル動作中に呼び出し可能な API(以下，API) を 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) を 17 個，計 121 個を提供している．FMP カーネルのターゲット非依存部のソースコードの行数は，ARM 用パッケージパッケージの場合，約 19,000 行である．また，FMP カーネルは，ASP カーネルが提供する全ての API に加えて，カーネル動作時に呼び出し可能な API が 17 個，静的 API が 1 個追加されており，計 139 個の API および静的 API を提供している．

3. テスト設計とテストプロセス

本章では，ASP カーネルと FMP カーネルのソフトウェアテスト設計について概観する．

3.1 テストの目的設定とテスト設計の方針

我々は，本研究で実施するテストの目的として，実装済みの ASP カーネルおよび FMP カーネルの品質向上，および仕様と実装のトレーサビリティ確認を掲げた．前者はソフトウェアテスト本来の目的であり，このためには通常，品質評価に重きをおいたテストと，不具合を摘出することに重きをおいたテストの両方を行う[8)~10)]．他方，トレーサビリティ確認を掲げた理由は，RTOS はクリティカルなシステムに用いられることが多く，仕様として記述されていない動作をすることが好まれないという背景があるためである．これらの目的を鑑み，テスト設計を行うための以下の方針を作成した．

（1）外部仕様を可能な限り網羅する
（2）設計思想が正しく実装されていることを可能な限り確認する

表 1 テストカテゴリ一覧
Table 1 List of the test categories in the present study

1. 仕様ベースのテスト	
A	API テスト (ブラックボックステスト)
B	処理単位テスト
C	SIL テスト
D	クラス関連テスト
E	カーネル管理外割込みのテスト
F	カーネル起動/終了時の同期テスト
2. 設計・ソースコードベースのテスト	
A	API テスト (ホワイトボックステスト)
G	ターゲット依存テスト
H	共通部ロック関数テスト
I	スタートアップモジュールテスト
J	コンフィギュレータテスト
K	機能拡張・チューニングテスト
3. エラー推測テスト	
L	ロック区間テスト
M	割込み禁止区間テスト
N	タイマ割込みテスト
O	スピンロック中の割込みテスト
P	マイグレーションテスト
Q	割込み出入口処理テスト
R	CPU 例外出入口処理テスト
S	アイドル処理テスト
T	デッドロック回避テスト
U	バリア同期テスト

（3）一度もテストしていないコードをなくす
（4）不具合が発生しやすいと推測される箇所を重点的にテストする

3.2 テスト設計の全体像

前章の方針に沿ってテスト設計を行った結果，22 個のテストカテゴリを抽出し，それらを仕様ベース，設計・ソースコードベース，およびエラー推測の 3 つに分類した．結果を表 1 に示す．

3.3 仕様ベースのテストの設計

仕様ベースのテストでは，統合仕様書に記述されている外部仕様を網羅的にテストする．TOPPERS 新世代カーネルの仕様は，主に API，処理単位，SIL，カーネル管理外割込みに分類される．それぞれの分類ごとにテスト手法が異なるため，別のテストとして設計することにした．

API テストでは，統合仕様書に記載された全 API の振る舞い，具体的には，タスク状態やシステム状態によって変化する API の挙動を網羅的にテストする．

処理単位テストでは，カーネルが行う処理単位の優先順位の管理，および処理単位の起動時や終了時，および休止時の処理が正しいことを確認する．処理単位とは，タスクや割込みハンドラなど，一連の実行スレッドとなる単位のことである．具体的には，複数の処理単位を一定の状況に設定した上で，処理単位の起

動トリガとなり得るシステムの状態変化を起こし，その際に，最も優先される処理単位が実行され，付随する起動処理や終了処理などの振る舞いが統合仕様書に記載されている通りであることを確認する．

SIL テストでは，カーネルとは独立したシステムインタフェースレイヤ (SIL) モジュールが提供する API が，統合仕様書に記載された通りに振る舞うことを確認する．

カーネル管理外の割込みテストでは，カーネル管理外割込みを設定できることや，カーネル管理内割込みのうち最高優先度の割込みが動作中であっても，カーネル管理外割込み発生時には必ずカーネル管理外割込み処理が行われることなどをテストする．なお，カーネル管理外の割込みとは高い割込み応答性を求められるアプリケーションのために用意された，カーネル内でマスクしない割込みのことである．

なお，クラスに関する機能，およびカーネル起動/終了時の同期は，マルチプロセッサ向け RTOS で追加された機能であるので，特別にテスト対象とすることにした．

3.4 設計・ソースコードベースのテストの設計

設計ベースのテストでは，設計指針に沿ってカーネルが正しく実装されていることを確認する．テスト設計の基準としては，ASP カーネルや FMP カーネルのパッケージに添付される設計ドキュメントをベースとした．

設計ベースのテスト対象は，主にカーネル内部の各種データ構造やアルゴリズムである．これらは外部仕様を達成するための手段として重要な要素であるが，詳細は統合仕様書には記述されず，また仕様ベースのテストだけではテスト漏れを生じる可能性がある．これが，設計ベースのテストが別途必要と考えられる理由である．

例えば，FMP カーネル中の共通部ロック関数は，プロセッサ間排他制御を実現するためのデータ構造とアルゴリズムを用いて実装されている．しかし，その全ての振る舞いを，仕様ベースのテストで網羅できるとは限らない．また，詳細設計は統合仕様書のような外部仕様書には記述されない．このような機構をテストするためには，設計ドキュメントをベースとしたテスト設計が必要となる．

ソースコードベースのテストでは，範囲をターゲット非依存部のみとした．この中で，一度もテストしていないコードをなくすため，ソースコードのカバレッジに関しては，C1 カバレッジを 100%とするという目標を定めた．

表 2 テストプロセスのアクティビティ
Table 2 The activities of the testing process

名称	内容
テスト分析	テスト対象ソフトウェアのドキュメントおよびソースコードを分析し，テストポリシー (テスト条件とカバレッジ基準) を抽出するアクティビティ．
テストケース生成	テストポリシーに沿って，テストケースおよびテストシナリオを生成するアクティビティ．
テストプログラム生成	テストケースもしくはテストシナリオを元に，実行可能なテストプログラムを生成するアクティビティ．
テスト実行	テストプログラムを実機やシミュレータ上で実行し，テストを行うアクティビティ．

3.5 エラー推測テストの設計

RTOS をマルチプロセッサ向けに拡張したことによる追加要素や，仕様もしくは設計が複雑な箇所など，不具合が発生しやすいと推測される箇所に関して，エラー推測によるテストを追加した．

表 1 に挙げられているテストカテゴリのうち，スピンロック中の割込みテスト，マイグレーションテスト，デッドロック回避テスト，バリア同期テストは，RTOS をマルチプロセッサ向けに拡張したことによる追加要素である．その他は，設計が複雑な箇所である．

3.6 共通テストプロセス

我々が実施した各カテゴリのテストに共通となるテストプロセスに含まれるアクティビティを表 2 に示す．

これらのアクティビティは，幾つかのテストの実施経験から抽出された．しかし実際には，いずれのテストカテゴリを実施する場合にも行われる共通のアクティビティであると考えられる．

4. テスト手法

4.1 API テストにおけるテストケース設計ポリシー

ここでは，API テストにおけるテストケース設計ポリシーについて示す．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．ただし，統合仕様書は，抽象的な表現で記述されているため，同一の仕様記述から抽出されるテストケースが，開発者によってばらつく可能性がある．そこで，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 毎のばらつきを防ぐこととした．また，テストケースを，テストプログラムへ変換可能な形で具体化するテストシナリオのフォーマットを導入し，テス

```
低優先度の実行状態のタスクから，
高優先度の休止状態のタスク (対象タスク) に対して，
act_tsk を発行すると，対象タスクが実行状態になること．
```

図 1 テストケースの例
Fig. 1 An example of the test case

```
前状態
  優先度 (低) の TASK1 が実行状態
  優先度 (高) の TASK2 が休止状態

処理
  TASK1 が act_tsk(TASK2) を発行し，
  エラーコードとして E_OK が返る

後状態
  TASK1 が実行可能状態となる
  TASK2 が実行状態となる
```

図 2 テストシナリオの例
Fig. 2 An example of the test scenario

```
pre_condition:
  TASK1:
    type   : TASK
    tskpri : MID
    tskstat: running
  TASK2:
    type   : TASK
    tskpri : HIGH
    tskstat: dormant

do:
  id     : TASK1
  syscall: act_tsk(TASK2)
  ercd   : E_OK

post_condition:
  TASK1:
    tskstat: ready
  TASK2:
    tskstat: running
```

図 3 TESRY 記法の例
Fig. 3 Example of The TESRY Notation

ト設計アクティビティ中でテストシナリオを作成することとした．

以下に，本研究におけるテストケースとテストシナリオの定義を示す．

我々は，各種の標準[6)7)] を参考にして，テストケースを次のように定義している．テストケースとは，特定のプログラムパスの実行や，指定された要求に適合していることの検証など，特定の目的のために作成された，入力値，実行事前条件，期待結果の対である．API テストにおけるテストケースのサンプルを図 1 に示す．このテストケースは，テストされるべき状況を動作ルールの形で抽象的に示している．この例は，休止状態のタスクを起動する API である act_tsk が，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．

他方，テストシナリオは，テストケースで指定された状況を再現するための具体的なタスクや割込みの設定，および API 呼び出しの操作手順を示したものである．例を図 2 に示す．

4.2 TESRY 記法

TESRY 記法は，Test Scenario for Rtos by Yaml 記法の略称であり，次節で説明するツール TTG 開発のために考案した．TESRY 記法は，テストシナリオの記述に用いられる．前節のテストシナリオを TESRY 記法で記述した例を図 3 に示す．

TESRY 記法は YAML 形式[3)] に準拠しているので，計算機上のツールによって取り扱うことが容易である．このため，TESRY 記法の導入は，テストケースやテストシナリオの自動生成に関する研究[14)] の促進にも大きく貢献した．また，現在は API テストのテストシナリオ記述のみで使用されているが，事前条件および事後条件を RTOS のシステム状態で表わすことができ，かつテストのトリガが API 呼び出しとなるテストケースであれば問題なく記述可能である．

4.3 TTG

TTG は，TOPPERS Test Generator の略称である．TTG は，テストプログラム生成アクティビティで使用されるツールであり，入力された TESRY データで記述されたテストシナリオから，実行可能なテストプログラムを自動生成する機能を有している．

TTG を開発することにした動機は，テストケースに対応するテストプログラムを人手で開発すると，開発工数や保守性などに著しく問題が発生することである．ASP カーネルおよび FMP カーネルの全 API に対するテストケースは，それぞれ 1,785 件，4,378 件であり，これらに対応するテストプログラムを人手で実装する場合の開発工数は大きい．また，1 つのテストシナリオを実行することのできるテストプログラムは複数存在しうる．そのため，複数の開発者でテストプログラムを実装した場合，実装方法を統一することが困難となり，その意味でもレビューや修正作業の負担が大きくなることが予想された．

一方，テストケースからテストシナリオを作成する作業は，変換法則さえ決めてしまえば開発者間でそれほどばらつくことはなく，統一が容易である．そこで，人手で行う作業は TESRY 記法によるテストシナリオの作成までとし，そこから先は TTG を用いてテストプログラムを生成することにした．これにより，開発

工数の削減と保守性の向上を図ることができた[15),16)].

4.4 テストライブラリ

TTG で生成するテストプログラムでは，テストライブラリを使用する．テストライブラリとは，テストプログラムにおいて共通に用いられる処理を再利用可能な形でまとめたものである．したがって，テストライブラリはテストプログラムを自動生成する際の重要な要素である．以下に，代表的なライブラリを示す．

（1）状態参照関数
（2）タイムティック操作
（3）割込み・CPU 例外発生関数
（4）同期ライブラリ

(1) は，RTOS の 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 (以下，システム状態) を確認するライブラリである．API テストでは，API 発行前後のシステム状態を常に確認するため，すべてのオブジェクトに対する状態を確認するライブラリが必要となる．

(2) は，RTOS 内のシステム時刻を制御するライブラリである．API によっては，API 発行直後だけでなく指定した一定時間後にシステム状態を変化させる機能を持つものもある．これらの API をテストする際，テストプログラムによって意図的に，システム時刻を制御する必要があるので，システム時刻の停止，更新，再開を実行可能とするライブラリを用意した．

(3) は，割込みや CPU 例外を発生させるライブラリである．API には，割込みの禁止許可に関するものや，CPU 例外ハンドラからのみ発行可能なものが存在する．また，テストの内容によっては，割込みハンドラや CPU 例外ハンドラなどの，タスク以外の処理を使用する場合がある．これらのテストプログラムにおいては，意図したタイミングで割込みや CPU 例外を発生させる必要があるので，割込みや CPU 例外を発生させるライブラリを用意した．

(4) については，4.5.1 で説明する．

なお，テストライブラリはもともと TTG が生成するテストプログラムが使用することを想定して作られているが，テストライブラリを単独で利用することも可能である．テストプログラムをハンドコーディングする場合にも，このライブラリを利用することができる．

4.5 マルチプロセッサ環境における実行順序制御

マルチプロセッサ環境下でテストを実行すると，実行結果が並列動作するプログラムの実行順序に影響を受けることがある．したがって，API テストや処理単位テストのように，決定的動作を期待して設計されるテストには，テスト条件としてプログラムの実行順序を盛り込み，その順序通りに実行する必要がある．もし盛り込まれた実行順序の通りに実行できないと，結果の妥当性を確認することができない．

そこで我々は，プログラムの実行順序を制御し，テストプログラムおよびカーネルを決定的に動作させる手法を研究してきた．我々がこれまでに研究した代表的な手法は 2 つである．1 つは，TTG が生成するテストプログラムによってプロセッサ間同期を実現する手法である．もう 1 つは拡張 Instruction Set Simulator(拡張 ISS) を用いたタイミング制御実行手法である．以下，順に説明する．

4.5.1 テストプログラムによるプロセッサ間同期の手法

TTG が生成するテストプログラムによるプロセッサ間同期は，同期メカニズムと同期パターンの 2 つで実現される．同期メカニズムとは，同期ライブラリとして実装されている，プロセッサ間同期実現用の関数である．

他方，同期パターンとは，プロセッサ間同期が必要な状況・目的を，テストプログラムの立場で分類し，状況毎に同期メカニズムをどう使えばよいかをパターン化したものである．同期パターンは 10 種類開発されている．

なお，本同期設計は FMP カーネルの仕様に依存していないので，他のマルチプロセッサ対応 RTOS へも適用できると考えられる．

4.5.2 拡張 ISS を用いたタイミング制御実行手法

複数のプログラムが並列もしくは並行動作する環境では，プログラムの実行順序に依存して実行パスが定まる分岐が存在することがある．例えば，排他制御やデッドロック回避に関する処理が該当する．これらの分岐を網羅するために，プログラムを繰り返し実行する手法が考えられるが，特定のパスを決定的に実行することができない．また，プログラムの実際の実行順序を知ることが困難であるため，実行順序に依存する分岐に関するテストを効率的に行うことができない．

そこで，テストプログラムからプロセッサの実行を制御することで，プログラム中の特定のパスを決定的に実行する機構を ISS へ実装した．これを利用し，FMP カーネルの実行順序に依存したパスをすべて決定的に実行可能であることを確認した．

5. テスト結果

5.1 概　　要

テスト対象は，ASP カーネル Release1.4.0 と FMP

表 3 テスト環境一覧
Table 3 List of the test environments

RTOS	ボード	プロセッサ
ASP カーネル	SkyEye(シミュレータ)	AT91 システム
	AP-SH2A-0A	SH2A(SH7211)
	MS7727CP01	SH3(SH7727)
FMP カーネル	SkyEye(シミュレータ)	AT91 システム
	AP-SH2AD-0A	SH2A-DUAL(SH7205)
	NaviEngine	ARM11MPCore

表 4 テストによって発見された不具合
Table 4 Defects found by test

年度	2009	2010		合計
テストカテゴリ	API	API	SIL	
統合仕様書	15	18	1	34
コンフィギュレータ	2	1	0	3
ASP	12	1	0	13
FMP	11	37	0	48
合計	28	55	1	98

カーネル Release1.1.0 である．対応する統合仕様書のバージョンは 1.1.0 である．表 1 で示したテストの全体像の中で，我々は API テストと SIL テストの 2 つを完了した．

API テストのテストケースの件数は，ASP カーネル向けが 1,785 件，FMP カーネル向けが 4,378 件であった．TTG で生成したテストプログラムの行数は，ASP カーネル向けが約 226,000 行，FMP カーネル向けが約 704,000 行であった．

SIL テストのテストケースの件数は，ASP カーネル向けが 104 件，FMP カーネル向けが 287 件であった．テストプログラムのサイズは，ASP カーネル向けが 860 行，FMP カーネル向けが 1,357 行であった．なお，SIL テストは件数が少ないため，TTG を用いずテストプログラムをハンドコーディングした．

テストを実施した環境一覧を表 3 に示す．なお，マルチプロセッサ環境における SkyEye においては，シミュレータであるので，様々なハードウェア構成を模擬するバリエーション設定パターンを 48 個用意し，テストを実施した．

5.2 検出した不具合

表 4 に，API テスト，および SIL テストにて発見された不具合の数を示す．FMP カーネルの不具合 48 件のうち，実機を用いたテストによって検出されたターゲット依存部の不具合は 5 件であった．このことから，ターゲット依存部に関するソースコードベースのテストを実施しなかったにも関わらず，API テストによって，関連するターゲット依存部の不具合が検出できていることがわかる．

5.3 不具合の事例

マルチプロセッサ向け RTOS である FMP カーネルでは，シングルプロセッサでは発生しない不具合がいくつか検出された．ここでは，マルチプロセッサ特有の不具合として 3 つの事例を挙げる．

（1）ロック関数不備
（2）実行タイミングに依存する不具合
（3）新設されたタスクの状態に関する不具合

(1) は，排他制御やデッドロック回避のために用いる，ロック関数の不備が原因の不具合である．FMP カーネルでは，割込み応答性の確保のために，排他制御する範囲を段階的にする実装方法を採用している．このため，段階的に取得したロックは，取得したときとは逆の順序で解放する必要がある．API テストでは，これらのロック関数の解放順序が間違っているものや，適切なロック解放関数となっていない不具合が検出された．

(2) は，再現率が 100%とならない，実行タイミングに依存する不具合である．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想定であったが，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．例えば，割込み応答性の確保のために，一時的に割込みを許可する処理において，割込みが入った場合などである．このようなある一定の条件が揃った場合にのみ実行される処理に不具合があり，ヒートラン的にテストを繰り返し実施した際に，不具合が検出された．

(3) は，シングルプロセッサでは存在しなかったタスクの状態に関する不具合である．FMP カーネルでは，他のプロセッサのタスクに対しても API を発行可能であるため，シングルプロセッサにはない，新しいタスクの状態が存在する．この状態のタスクが API を発行する，もしくは API を発行される，という状況を，RTOS が考慮できていない処理が含まれており，不具合として検出された．

6. テストスイートの公開

本研究の主要な研究成果は，TOPPERS Test Suite Package(以下，TTSP) にまとめて，2011 年 5 月に公開した☆．

TTSP は，TOPPERS 新世代カーネルを対象とした，各種テストツール，テストプログラム，テストデータ，ドキュメントの統合体であり，本研究の最も主要な成果物である．このようにパッケージ化して提供す

☆ FMP カーネル向けは 2012 年 4 月公開予定

る目的は，本研究で開発したテストを，ユーザが対象とするターゲットシステムに適用する際の環境設定を局所化し，最小限の設定で実施可能とすることである．また，TTSP には，TTB(TOPPERS Test Builder) と呼ばれる，シェルスクリプトで作成された CUI のツールが含まれている．ユーザは，ターゲットシステム用に合わせて環境設定を行った後，TTB を実行し，実施したいテストを選択するだけで，実行モジュールを生成することが可能である．

7. ま と め

本論文では，NCES を中心としたコンソーシアムが 2009 年度から 2010 年度にかけて実施した，マルチプロセッサ向け RTOS のテストに関する研究について報告した．

本研究では，仕様ベース，設計・ソースコードベース，エラー推測の 3 つの観点によるテストの全体像を設計し，このうち仕様ベースの API テストと SIL テスト，および設計・ソースコードベースの API テスト (ホワイトボックステスト) を実施した．また，テスト手法として，テスト実施プロセスの各アクティビティを改善する過程で，テストシナリオを記述するための TESRY 記法，テストプログラム生成ツールの TTG，およびマルチプロセッサ向け RTOS のテストを決定論的に実行するためのプロセッサ間同期手法や拡張 ISS を開発した．最終的には，ASP カーネルと FMP カーネルに関する API テストを全て完了し，かつ対象としたソースコードカバレッジを 100%とし，合計 98 件の不具合を検出して RTOS の品質向上に貢献した．これらのテスト実施の記録は，今後両 RTOS の導入を検討している企業にとって有用なテストのエビデンスとなり，オープンソース RTOS 導入時のコストを削減する効果も生まれると考えられる．

本研究の後の取り組みとして，NCES では，2011 年度より，新たなコンソーシアム型共同研究である「次世代車載システム向け RTOS の仕様検討および開発に関するコンソーシアム型共同研究」[4] を実施中である．ここでは新たに AUTOSAR OS を対象としたテスト手法の研究が行われており，本共同研究の成果物と知見が活かされている．

謝辞 本研究の推進に協力して下さった，コンソーシアム型研究の皆様に謹んで感謝の意を表する．

参 考 文 献

1) TOPPERS プロジェクト, http://www.toppers.jp/
2) TOPPERS 新世代カーネル統合仕様書, http://www.toppers.jp/docs/tech/ngki_spec-130.pdf
3) YAML, http://yaml.org/
4) 「次世代車載システム向け RTOS の仕様検討および開発に関するコンソーシアム型共同研究を開始」プレス発表, http://www.nces.is.nagoya-u.ac.jp/press/nces-release-1105.pdf
5) OJL による最先端技術適応能力を持つ IT 人材育成拠点の形成, http://www.ocean.is.nagoya-u.ac.jp/
6) IEEE: IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990 (1990).
7) IEEE: IEEE Standard for Software Test Documentation, IEEE Std 829-1998 (1998).
8) Glenford J. Myers, Tom Badgett, Todd M. Thomas, Corey Sandler: The art of software testing, John Wiley & Sons. Inc, (2004).
9) 玉井哲雄: ソフトウェア工学の基礎, 岩波書店 (2004).
10) 保田勝通: ソフトウェア品質保証の考え方と実際—オープン化時代に向けての体系的アプローチ, 日科技連出版社 (1995).
11) 小林隆志, 沢田篤史, 山本晋一郎, 野呂昌満, 阿草清滋, On the Job Learning: 産学連携による新しいソフトウェア工学教育手法, 電子情報通信学会信学技報 SS2009-28, vol.109, no.170, pp.95–100 (2009).
12) 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 に対する API テストの実施, ソフトウェアテストシンポジウム 2010 予稿集, pp.46–53 (2010).
13) 松浦光洋, 金ハンソル, 眞弓友宏, 金スンヨプ, 廉正烈, 金榮柱, 木村貴寿, 鴫原一人, 馬鋭, 森孝夫, 本田晋也, 山本雅基, 高田広章: マルチプロセッサ対応 RTOS のテスト開発, 情報処理学会研究報告, 2010-EMB-16, No.10 (2010).
14) 鴫原一人, 森孝夫, 本田晋也, 山本雅基, 高田広章: RTOS のテスト自動生成システムに関する一考察, 情報処理学会研究報告, 2010-EMB-16, No.11 (2010).
15) 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, 情報処理学会研究報告, Vol.2010-EMB-18, No.1, pp1–8 (2010).
16) 金ハンソル, 浅見侑太, 阿部真也, 金スンヨプ, 金榮柱, 金賢敏, 竹谷美里, 木村貴寿, 鴫原一人, 森孝夫, 山本雅基, 本田晋也, 高田広章: 組込みリアルタイム OS 向けテストツールのマルチプロセッサ拡張, ソフトウェアテストシンポジウム 2011 予

稿集, pp.96–103 (2011).
17) 金榮柱, 金スンヨプ, 金ハンソル, 金賢敏, 竹谷美里, 木村貴寿, 鴫原一人, 森 孝夫, 本田晋也, 山本雅基, 高田広章: マルチプロセッサ対応 RTOS に対する API テストの実施, 情報処理学会研究報告, 2011-EMB-20, No.13 (2011).
18) 浅見侑太, 鴫原一人, 本田晋也, 山本晋一郎, 高田広章: マルチプロセッサ対応 RTOS 向けテストプログラム生成ツールにおけるプロセッサ間同期の実現, 情報処理学会研究報告, 2011-EMB-20, No.11 (2011).
19) 一場利幸, 高瀬英希, 鴫原一人, 本田晋也, 高田広章: マルチプロセッサ環境におけるタイミング依存のシナリオを実行可能なシミュレーション機構, 情報処理学会研究報告, 2011-EMB-20, No.43 (2011).