情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-EMB-18 No.1
2010/8/9

# 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

鴫原一人[†1]　眞弓友宏[†2]
本田晋也[†1]　高田広章[†1]

$\mu$ITRON 仕様をベースとしたマルチプロセッサ対応 RTOS のテストスイート開発を効率化するために，テストシナリオを形式化して記述する TESRY 記法を定め，TESRY 記法を用いて作成したデータからテストプログラムを生成するツールを開発した．開発したツールを用いて，API のテストを約 3,000 件開発し，開発の効率化を確認した．また，31 件の不具合を検出した．

# A Method to Describe Test Scenarios and a Test Program Generator for Multiprocessor RTOS

Kazuto SHIGIHARA,[†1] Tomohiro MAYUMI,[†2]
Shinya HONDA[†1] and Hiroaki TAKADA [†1]

In this paper, we present a method to describe test scenarios (the TESRY notation) aimed at efficiently developing test suites for multiprocessor RTOS that support the $\mu$ITRON specification. We developed a test program generator based on the TESRY notation, which was used to develop about 3,000 API test cases, and confirmed the efficiency of the proposed approach. We also detected 31 bugs in the RTOS.

†1 名古屋大学
Nagoya University
†2 名古屋大学 (現所属 (株) デンソークリエイト)
Nagoya University (currently: DENSO CREATE INC.)

## 1. はじめに

近年，組込みシステムにおいてマルチプロセッサの利用が増加している．その背景に，低消費電力で動作し性能を向上させる方法としてマルチプロセッサが有効だという点がある．TOPPERS プロジェクト[1] では，$\mu$ITRON[2] 仕様をベースとしたシングルプロセッサ対応の RTOS である TOPPERS/ASP カーネル (以下，ASP カーネル) を拡張して，マルチプロセッサに対応した TOPPERS/FMP カーネル (以下，FMP カーネル) を開発し，オープンソースとして公開している．これらの RTOS は TOPPERS 新世代カーネルと総称される．

ASP カーネルおよび FMP カーネルの開発は，非営利で行われていることや，製品への組込み時には利用者側での改変や拡張が行われることが一般的である．したがって，RTOS のテストの開発と実施は，RTOS 開発者側では行わずに，利用者側が行うことが通例である．しかしながら，RTOS に対するテストの工数は膨大であるので，利用者側でテストを開発するのは負荷が大きく，さらにマルチプロセッサに対応した RTOS に関しては，そもそもテスト経験が少ないために，テスト手法が確立されていない．

Linux などのオープンソースの OS に対しては，テストスイートを開発するプロジェクト[3)–7)] が存在するが，オープンソースの RTOS に対しては取り組みの事例がない．また，膨大となるテストスイート開発の工数を低減させる手法ついては，これらのプロジェクトでは検討されていない．

そこで，我々は，複数の企業と団体の参加を得て，RTOS のテストを行うコンソーシアム型の研究組織を立ち上げ，ASP カーネルと FMP カーネルに対するテストスイート開発と，テスト実施の工数を低減する手法を検討している．

コンソーシアム型研究では，まず RTOS に対して必要なテストの項目を洗い出し，テストスイート開発の第一段階として，API が仕様通りに正しく振る舞うことをテストする API テストの開発を行った．API は，ASP カーネル，FMP カーネル合わせて 139 個用意されており，1 つの API に対して，30 件程度のテストケースが必要となるため，テストケースは数千件と見積もられた．それぞれのテストケースに対するテストプログラムをハンドコーディングで開発すると，開発者により記述の差が発生し，保守性や可読性が低くなるため，品質を保つのが困難になる．そのため，数千件のテストケースを一定の品質で開発，管理するには膨大な工数が見込まれる．

この問題を解決するため，本研究では，テストシナリオを形式的に表現する記法を考案し，その記法を用いて記述したデータからテストプログラムを生成するツール TTG(Toppers


© 2010 Information Processing Society of Japan

Test Generator) を開発した．考案した記法は，TESRY(TEst Scenario for Rtos by Yaml) 記法と呼び，TESRY 記法によってテストシナリオを記述したデータファイルを，TESRY データと呼ぶことにした．テストプログラムを機械的に生成することにより，テストプログラムの品質を一定に保つことが可能になる．さらに，開発対象がテストシナリオになることにより，保守性，可読性を向上させることができる．

なお，TTG 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」[8),9)] の OJL(On the Job Learning) のテーマとして開発した．

## 2. TOPPERS 新世代カーネル

TOPPERS 新世代カーネルとは，TOPPERS プロジェクトにおいて $\mu$ITRON 仕様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，信頼性と安全性とソフトウェアポータビリティを向上させるために，$\mu$ITRON 仕様に拡張と改良が加えられており，その仕様は，「TOPPERS 新世代カーネル統合仕様書」[10)] (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 (出発点) となる RTOS であり，TOPPERS 新世代カーネル統合仕様に準拠している．

ASP カーネルに実装される API は，カーネル動作中に呼び出し可能なもの (以下，API) が 104 個，予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もの (以下，静的 API) が 17 個の，計 121 個である．

### 2.2 FMP カーネル

FMP カーネルは，ASP カーネルを，マルチプロセッサ向けに拡張したもので，機能分散型と対称型の両方のタイプのマルチプロセッサシステムに適用できる．

FMP カーネルの仕様の主な特徴として，タスクを設計時にプロセッサに割り付けること，カーネルはシステム稼働中にタスクを移動させないこと，システム稼働中にタスクを別のプロセッサに移動させる API を用意していること，などがあげられる．

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．

## 3. API テスト

我々は RTOS を開発，利用した経験から，重点的にテストするべき機能や，不具合が発生しやすい処理を整理し，RTOS に対して実施すべきテストの項目を選定した．その中から，まず API に着目したテストを実施した．これは API が，RTOS のソースコードの大部分を占めていることや，RTOS を利用するユーザへ提供されるインターフェイスとして，仕様通りに動作することが重要と考えたからである．本章では，実施した API テストの概要，および開発したツールの必要性について述べる．

### 3.1 API テストの概要

API に着目したテストとは，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ものである．我々はこれを API テストと呼ぶことにした[11)]．

API テストの開発の流れは次の通りである．統合仕様書からテスト対象である API の仕様と振舞いを理解し，テストケースを抽出する．テストケースの例を**図 1** 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，正しく動作するためのテストケースの 1 つである．テストケース毎にテストシナリオを作成し，テストシナリオを実現するためのテストプログラムを開発する．テストプログラム実行により，ソースコードカバレッジを取得して，事前に確認したソースコードカバレッジと照らし合わせ，テストプログラムが想定したパスを通過していることを確認する．

### 3.2 テストシナリオ

多くの API は，発行により，カーネルの状態や，タスクやセマフォといったカーネルオブジェクトの状態 (システム状態) が変化する．API テストでは，あるシステム状態のとき，API を発行することで，仕様通りのシステム状態の変化が起きることを検証する．具体的には，API 発行前のシステム状態 (前状態) を定義し，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) を確認するテストプログラムを実行する．このようにテスト内容を「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して取り扱う．

図 1 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 2** に示す．優先度の低い TASK1 と高い TASK2 が存在して，前状態は，TASK1 が実行されている状態 (実行状態)，TASK2 が休止状態である．処理は，TASK1 が 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 を受け取ることである．処理の結果，TASK2 のほうが優先度が高いため，ディスパッチが発生し，TASK2 に処理が切り替わり (実行状態)，TASK1 は実行待ち (実行可能状態) となることを，後状態で定義している．


© 2010 Information Processing Society of Japan

対象タスクの優先度が実行状態のタスクより高い場合，
かつ対象タスクが休止状態である場合は，
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 Example of a 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 Example of a 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

### 3.3 テストプログラム生成ツールの必要性

すべての API の仕様と振舞いをテストするために，テストケースを抽出したところ，全 API に対するテストケースは約 3,000 件となった．テストシナリオからテストプログラムを手作業で開発すると，以下のような問題のために，テストスイート開発とテスト実施の工数が膨大となることが予想される．

**(a)** テストプログラムの可読性，保守性の低下
**(b)** マルチプロセッサの同期処理実装の非効率性
**(c)** テストの開発工数と実施工数のトレードオフ
**(d)** 同一テストシナリオに対する RTOS 毎の重複開発
**(e)** テストシナリオの正当性確認の必要性

問題点 (a) は，テストケース毎に，複数の開発者がそれぞれテストプログラムを開発することから発生する．例えば，同じ前状態であっても，必要となるシステム状態の設定順序や確認方法には，何通りも実現方法があるため，統一することは困難である．開発者毎にテストプログラムにばらつきが生じると，可読性が低下し，レビューや修正作業が困難となり，保守性も低下する．

問題点 (b) は，マルチプロセッサに対応したテストプログラム開発の問題である．FMP カーネルに対するテストプログラムでは，テストシナリオに記述された内容を実現するために，プロセッサ間で適切な同期処理を行う必要がある．しかし，一般にマルチプロセッサに対する同期処理は複雑であるため，テストシナリオ毎に実装するのは非効率である．

問題点 (c) は，テスト実施の容易性に関する問題である．一般に組込みシステムソフトウェアにおいては，実行モジュールをターゲットシステムにロードして実行する．そのため，ターゲットシステムによっては，テストケース毎に実行モジュールが分かれていると，テストケースの数だけロードと実行を繰り返す必要があり，テスト実施に多大な工数を要する．このようなターゲットシステムにおいては，1 つの実行モジュールで全テストケースを実施できると，テスト実施の工数を大幅に削減することができる．しかし，数千件のテストケースを，1 つのテストプログラムで開発するのは，各テストケースのタスクや変数などが，互いに干渉しないように設計する必要があるため，現実的ではない．

問題点 (d) は，ASP カーネルと FMP カーネルのプログラムの記法の違いに起因する．FMP カーネルは ASP カーネルを拡張したものであるので，ASP カーネル用に作成したテストシナリオは，FMP カーネルに対しても 1 プロセッサ上で動作させることでテスト実施が可能である．しかし，ASP カーネルと FMP カーネルとでは，コンフィギュレーションファイルの記述や，テストライブラリに用意された関数が異なるなど，同じテストシナリオでも，テストプログラムの内容が異なる．つまり，同じ内容のテストシナリオにも関わらず，RTOS 毎に別々のテストプログラムを開発，保守する必要がある．

問題点 (e) は，作成したテストシナリオ自体の正当性に対する問題である．作成したテストシナリオが，RTOS の仕様として矛盾している場合など，テストシナリオに問題がある場合，テストプログラムをビルドもしくは実行するまで気付けない可能性がある．

これらの問題をすべて解決するため，テストシナリオからテストプログラムを生成するツールを開発することにした．

## 4. TTG の概要

本章では，TTG が生成するテストプログラムおよび TTG の要件について述べる．

なお，生成したテストプログラムの正当性については，テストプログラム実行時に取得す


© 2010 Information Processing Society of Japan

るソースコードカバレッジが，想定したパスを通っているか確認することで，検証する．

### 4.1 生成するテストプログラム

テストプログラムでは，テストシナリオで指定されたタスクとは別に，前状態の実現や後状態のチェック用のタスク (メインタスク) を用意する．

メインタスクは，テストに必要なシステム状態の設定を行い，前状態を実現する．実現したシステム状態が，前状態に記述された内容と一致していることの確認も行う．前状態を実現した後に，処理に指定された処理単位において，記述された処理内容をそのまま実行する．戻り値が指定されていれば，指定された戻り値と一致しているかの確認を行う．処理の後で，システム状態が後状態に記述された内容と一致しているかを確認する．これをテストシナリオの数だけ繰り返し，1 つのテストプログラムを生成する．

テストプログラムは，後状態がテストシナリオで指定されたシステム状態と異なる状態となった場合は，エラーを出力する．システム状態の確認方法として，RTOS に用意されている状態参照機能を持つ API(以下，状態参照 API) の結果を利用して判定を行う．そのため，状態参照 API 自身に対する正当性を確認するテストは別途必要となる．また，状態参照 API の結果以外にもプログラム中にチェックポイントをいれ，想定した順序で通過することを確認することでも判定を行う．

### 4.2 要　　件

3.3 節で述べた問題点を解決するため，TTG は以下の要件に基づいて開発を行った．

(1) 形式化したテストシナリオ記法からのテストプログラム生成
(2) マルチプロセッサへの対応
(3) 複数のテストシナリオ統合
(4) 選択した RTOS に対するテストプログラム生成
(5) テストシナリオの正当性チェック

要件 (1) として，TTG は，5 章で述べる TESRY 記法に準拠してテストシナリオを記述した TESRY データを，入力データとして読み込む．これにより，問題点 (a) を解決できる．

要件 (2) として，問題点 (b) を解決するため，マルチプロセッサに対するテストプログラムにおいては，TTG が適切な同期処理をテストプログラムに組み込むこととした．

要件 (3) として，問題点 (c) を解決するため，複数のテストシナリオを指定して，1 つのテストプログラムを生成することを可能とした．また，複数のテストシナリオを含むテストプログラムを生成した際，テストプログラム実行時のメモリ使用量を抑えるため，テストシナリオ間でタスクのスタック領域を共有させる．

要件 (4) として，問題点 (d) を解決するため，TTG 実行時に，指定した RTOS を対象とするテストプログラムを生成する機能を設けることとした．

要件 (5) として，問題点 (e) を解決するため，TESRY データ内に誤記や矛盾が存在すると，TTG 実行時にエラーを出力することとした．

なお，制約事項として，以下に示す内容のテストケースについては対象外とした．これは，TTG で対応する工数と，該当するテストケースが少数であることとの兼ね合いによって対応しないと判断したものである．

- 状態参照 API の正当性を確認するテスト
- 実行状態のタスクが存在しないテスト
- カーネル初期化前，カーネル終了処理開始後のテスト
- CPU 例外発生を必要とするテスト
- 割込み発生を必要とするテスト
- 静的 API のテスト

## 5. TESRY 記法

本章では，TESRY 記法について述べる．

### 5.1 構　　成

TESRY 記法は，テストシナリオを YAML 形式[12] で記述したものである．YAML を採用した理由は，テストシナリオをデータ構造で階層的に表現できること，改行とインデントによって構造を表すので可読性が高いこと，などである．

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 とその引数と戻り値を “do” 内に記述する．なお，前状態と後状態とで変化しないパラメータは省略可能とする．

図 2 のテストシナリオを TESRY 記法で記述したものを**図 3** に示す．

### 5.2 前状態，後状態の定義

pre_condition，post_condition に対して，テスト対象とする ASP カーネルおよび FMP カーネルに存在する全カーネルオブジェクト，および CPU ロック状態やディスパッチ禁止状態などのシステム状態を記述可能とする．例として，タスクの状態定義の仕様を**図 4** に示す．“TASK1” はタスクに付与する固有の ID である．

### 5.3 処理の定義

do として，テスト対象とする API を，発行する処理単位 (id)，引数を含む実行コード


© 2010 Information Processing Society of Japan

```
TASK1:
  type    : TASK /* オブジェクト種別 */
  tskstat : [ running | ready | waiting | /* 実行状態 | 実行可能状態 | 待ち状態 | */
              dormant | suspended |       /* 休止状態 | 強制待ち状態 | */
              waiting-suspended |         /* 二重待ち状態 | */
              running-suspended ]         /* 強制待ち状態 (実行継続中)(FMP カーネル) */
  wobjid  : [ 対象待ちオブジェクト ID |     /* 対象待ちオブジェクト ID | */
              SLEEP | DELAY ]             /* slp_tsk() による待ち | dly_tsk() による待ち */
  actcnt  : [ 0 | 1 ] /* 起動要求キューイング数 */
  wupcnt  : [ 0 | 1 ] /* 起床要求キューイング数 */
  itskpri : [ HIGH | MID | LOW | 整数 ] /* 起動時優先度 */
  tskpri  : [ HIGH | MID | LOW | 整数 ] /* 現在優先度 */
  porder  : [ 整数 ] /* 同一優先度タスク内での優先順位 */
  exinf   : [ 整数 ] /* 拡張情報 */
  prcid   : [ 整数 ] /* 割付けプロセッサ ID(FMP カーネル) */
  actprc  : [ 整数 ] /* 次回起動時割付けプロセッサ ID(FMP カーネル) */
  bootcnt : [ 整数 ] /* 起動回数 */
  lefttmo : [ 整数 ] /* タイムアウト待ち時間 */
```

**図 4** タスクの状態定義
Fig. 4 Definition of Task Condition

(syscall)，戻り値 (ercd) に分けて記述する．図 3 の例では，処理単位:TASK1 が，実行コード:act_tsk(TASK2) を発行して，戻り値:E_OK が返ることを示している．

なお，API を発行する処理単位が，後状態でオブジェクト待ちやタイムアウト待ちとなる場合，戻り値に何が返ってくるかは TTG が生成するテストプログラムに依存するため，省略することが可能である．この場合，テストプログラムでは戻り値のチェックは行わない．

### 5.4 経過時間指定

3.2 節で述べたように，RTOS の API は，発行直後にシステム状態が変化するものが多いが，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．例えば，dly_tsk は，API 発行直後に発行したタスクを待ち状態とし，引数で指定した時間経過後に，実行可能状態とする．このような API が正しく振舞うことをテストする場合，テストシナリオで時間経過を明示的に指定する必要がある．

TESRY 記法では，前状態を基準時刻として，後状態の中で指定した時刻でのシステム状態をチェックする記述を定めた．**図 5** に dly_tsk の TESRY データの例を示す．post_condition に経過時間 (ミリ秒) を指定している．“0”，“3”，“4” は経過した時間を示す．“0” は発行直後であり，API を発行したタスクが，待ち状態 (waiting) へと変化し，タイマーとなる値

```
pre_condition:
  TASK1:
    type   : TASK
    tskpri : MID
    tskstat: running
  TASK2:
    type   : TASK
    tskpri : MID
    tskstat: ready

do:
  - id     : TASK1
    syscall: dly_tsk(3)
    ercd   : E_OK

post_condition:
  0:
    TASK1:
      tskstat: waiting
      wobjid : DELAY
      lefttmo: 3
    TASK2:
      tskstat: running
  3:
    TASK1:
      lefttmo: 0
  4:
    TASK1:
      tskstat: ready
```

**図 5** 時間指定の例
Fig. 5 Example of Time Assign

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

do_0:
  - id     : TASK1
    syscall: slp_tsk()
    ercd   : E_RLWAI

post_condition_0:
  TASK1:
    tskstat: waiting
    wobjid : SLEEP
  TASK2:
    tskstat: running

do_1:
  - id     : TASK2
    syscall: rel_wai(TASK1)
    ercd   : E_OK

post_condition_1:
  TASK1:
    tskstat: running
  TASK2:
    tskstat: ready
```

**図 6** 複数の処理/後状態の例
Fig. 6 Example of Multiple ‘‘do’’ and ‘‘post_condition’’

(lefttmo) が設定される．“3” は 3 ミリ秒後の状態であり，lefttmo が時間とともに減少し，0 になったことを示す．“4” で 4 ミリ秒となり，引数で指定した時間が経過したため，タスクは実行可能状態 (ready) となる．

### 5.5 複数の処理/後状態

API テストでは，API の仕様を網羅するために，仕様に定義されたすべての戻り値をテ


© 2010 Information Processing Society of Japan

ストする必要がある．戻り値を確認するテストケースによっては，API を発行するだけでは，その API からリターンせず，その後，別のタスクに何らかの API を発行させることでリターンさせ，意図した戻り値を返す必要がある．

例えば，slp_tsk という API は，発行したタスクが起床待ち状態となるが，その後，別のタスクから rel_wai という待ち状態を強制解除する API を発行されると，戻り値として E_RLWAI が返るという仕様がある．この仕様をテストする場合，do の id を slp_tsk，ercd を E_RLWAI と記述しても post_condition で起床待ち状態となっているため，戻り値に E_RLWAI が返ったことを確認できない．

そこで，TESRY 記法では，複数の API を指定した順序で発行するように指定できる記述を定めた．**図 6** に例を示す．1 回目の API 発行として TASK1 が slp_tsk を発行し，起床待ちとなる．続いて，実行状態となった TASK2 が TASK1 に対して rel_wai を発行する．TASK1 のほうが優先度が高いため，再び TASK1 が実行状態となり，戻り値 E_RLWAI が返ったことを確かめる．

### 5.6 処理単位の複数回起動

タスクや周期ハンドラなどの処理単位は，テストシナリオの内容によっては，複数回起動する可能性がある．テストとして，想定した回数だけ正しく起動したことを確かめるためには，テストプログラムにおいて，起動回数に応じた処理を行う必要がある．

そこで，TESRY 記法では，処理単位の起動回数というパラメータを設け，複数回起動した場合に，何回目の起動かを判断できるようにする．タスクの場合，図 4 の bootcnt が該当するパラメータである．

### 5.7 マルチプロセッサ対応

図 6 の例において，TASK1 がプロセッサ 1，TASK2 がプロセッサ 2 に割り付けれていて，TASK2 がプロセッサ 2 で実行状態 (running) であった場合，テストプログラムでは，どのタスクがどのプロセッサへ割り付けられているかを設定する必要がある．

そこで，マルチプロセッサで必要となる prcid，actprc を定義した．TTG はこのパラメータを元に，片方のプロセッサだけ処理が進んでしまうといった問題が発生しないよう，適切な同期処理を実現する．

## 6. テストプログラム生成

本章では，TTG のテストプログラム生成処理について述べる．

### 6.1 プログラム生成処理概要

TESRY データに記述されたテストシナリオを実現するためには，対象とする RTOS の仕様を考慮した上で，実行可能なテストプログラムを生成する必要がある．TTG は，入力された TESRY データを解析し，テストシナリオで要求されるシステム状態やオブジェクトを用意し，適切な順序で処理が実行されるソースコードを生成する．

ここでは前状態に記述されたシステム状態を実現する方法について述べる．**図 7** に前状態を作成するフローを示す．

まずメインタスクを起動して，同期通信オブジェクトの設定や，オブジェクト待ちのタスクの作成を行う．実行状態のタスクが存在する場合は，タスクを起動し，以降の処理は，そのタスクで行う．ディスパッチ禁止状態，割込み優先度マスクの設定を行った後で，前状態が完成したことを状態参照 API を使用して確認する．これは，アラームハンドラ/周期ハンドラの起動中や，CPU ロック状態では，状態参照 API を発行できないという RTOS の仕様を考慮したためである．前状態を確認した後，テストシナリオに合わせて，アラームハンドラ/ 周期ハンドラを起動し，最後に CPU ロック状態の設定を行う．これは CPU ロック状態では，タイマー割込みが発生しないために，アラームハンドラ/ 周期ハンドラが起動できないためである．

実行状態のタスクが存在しない場合は，ディスパッチ禁止状態を設定し，前状態の確認を行ってからテストシナリオに合わせて，アラームハンドラ/ 周期ハンドラを起動する．これは，ディスパッチ禁止状態であっても，状態参照 API が発行でき，アラームハンドラ/周期ハンドラを起動することも可能だからである．

処理や後状態についても同様に RTOS の仕様に基づいて，ソースコードを生成する．

### 6.2 スタック共有

組込みシステムで用いられるハードウェアでは，使用できるメモリ容量に制限があることが多い．そのため，テストプログラムが使用するメモリサイズが大きくなると複数のテストプログラムに分割し，プログラムのロードから実行までを何度も繰り返す必要がある．

TTG が生成するテストプログラムでは，テストシナリオ毎に必要なタスクを用意するが，そのテストシナリオが終わるとそのタスクは以降使われることはない．そこで，タスクに必要なメモリ領域の中で多くを占めるスタック領域を，すべてのテストシナリオで共有し，再利用することで，実行時のメモリ使用量を抑えた．

### 6.3 プログラムフロー出力

テストがフェイルとなった場合，テストプログラムはソースファイルの行数，もしくは想


© 2010 Information Processing Society of Japan

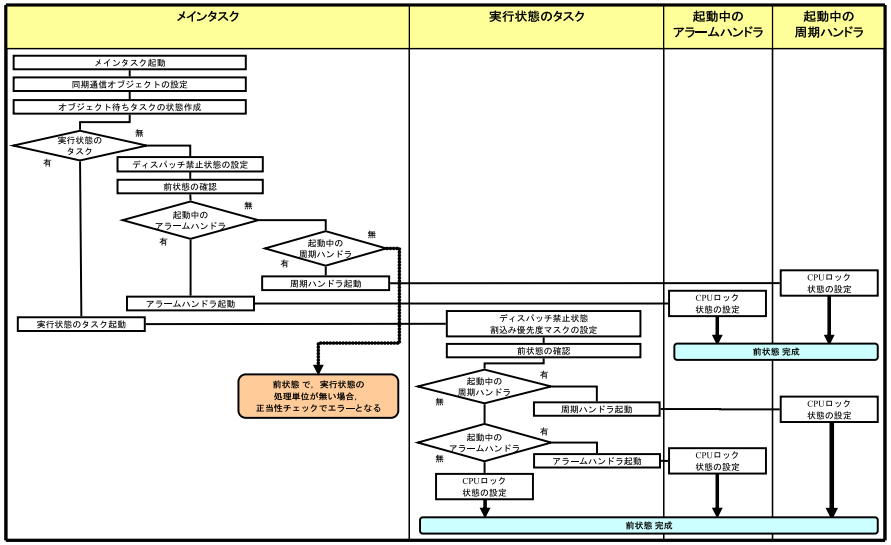


**図 7** 前状態作成フロー
Fig. 7 The ‘‘pre_condition’’ Creation Flow

定外の順に通過したチェックポイントの番号を出力する．TTG が生成するテストプログラムは，開発した全テストシナリオを入力とした場合，約 20 万行になり，そこからフェイルの原因を探るには，容易ではないことが予想される．そこで，TTG ではテストプログラム生成時，TTG が設定するチェックポイントを含む API 呼び出し順序を可視化したフロー図を，HTML ファイルとして出力可能とした．テストがフェイルとなったときにこのファイルと実際の結果を比較することで，原因箇所を容易に調査可能となる．

## 7. 評　　価

テストスイートは開発中であるが，これまでに開発した API テストは，静的 API を除いて，ASP カーネルが 1,597 件，FMP カーネルが 1,307 件である．本章では，TTG を用いた開発の評価について述べる．

なお，TTG は Ruby で開発し，総行数は 9,633 行であった．

### 7.1 サポート範囲の妥当性

ASP カーネルのテストケースにおいて，48 件が 4.2 節で述べた制約事項により，TTG で対応できないテストケースであった．つまり，約 97%のテストプログラムを TTG で機械的に生成することができた．残り 3%のテストケースを，ハンドコーディングで開発した結果，テストプログラムの総行数は 6,483 行であった．

制約事項となるテストケースの大部分を占める，状態参照 API の正当性を確認するテストを，TTG で対応可能とする場合，状態参照 API と同等の機能を有する関数を，別途開発する必要がある．この関数のプログラム，および関数を検証するためのプログラムを実装すると，10,000 行規模となることが見積もられていたため，制約事項とした判断は妥当であったと言える．

なお，FMP は開発途中であり，制約事項のテストケースを開発していないため，ここでの評価は行わない．

### 7.2 開発の効率化

ASP カーネルでは，TESRY データの総行数が 52,352 行であるのに対し，TTG が生成した全テストケースを実行するためのテストプログラムは，220,814 行となった．

FMP カーネルでは，TESRY データの総行数が 59,883 行であるのに対し，TTG が生成した全テストケースを実行するためのテストプログラムは，252,952 行となった．

単純に開発規模が 4 分の 1 程度まで低減できていることや，TESRY データの可読性の高さによる修正の容易さからも，テストスイート開発の工数を削減し，効率化できたと考えられる．また，ASP カーネル用の全テストケースは，FMP カーネルに対して流用できるため，さらに工数を削減することができる．

### 7.3 スタック共有の効果

開発した全 TESRY データを入力としたときのテストプログラムに対して，スタック共有を行った場合と行わなかった場合の，スタック領域のメモリ使用量について比較した．

ASP カーネル用テストプログラムのメモリ使用量は，スタック共有を行わなかった場合で約 3,112KB，スタック共有を行った場合で約 178KB となり，6%以下に削減できた．

FMP カーネル用テストプログラムのメモリ使用量は，スタック共有を行わなかった場合で約 4,267KB，スタック共有を行った場合で約 505KB となり，12%以下に削減できた．

以上の結果より，スタック共有の有効性が確認できた．

### 7.4 検出した不具合

本ツールの開発およびテストケースの抽出作業の中で，テスト対象とした ASP カーネル，FMP カーネル，統合仕様書に関して，31 件の不具合を検出した．検出した不具合は，以下のような内容である．


© 2010 Information Processing Society of Japan

- 仕様上の不具合の例
  - 曖昧な用語定義/エラー条件の整備
  - ソースコード上考慮されている振舞いが，仕様に記載されていない
- ソースコード上の不具合の例
  - 統合仕様書の記載と異なる実装が存在する
  - 不要な条件分岐が存在する
  - ターゲット依存部の実装不具合

API テストを開発したことで，特に検出することが多かったのは，ターゲット依存部の実装不具合である．これは様々なターゲットに対して TTG が生成したテストプログラムを実行することで，API の中で呼び出されているターゲット依存部が，正しく実装されていることをテストできるためである．

マルチプロセッサ対応 RTOS である FMP カーネルに関しては，ターゲット依存部の実装がより複雑であるため，今後さらにテストスイートとしての実用性が期待できる．

## 8. 今後の展望

### 8.1 テスト範囲の展開

今回開発した TERSY データには，データキューへ送信するデータ内容や，イベントフラグの待ちビットパターンなど，任意に指定できるパラメータは具体的な値で記述している．しかし，RTOS のテストスイートは使用する企業やその用途によって求めるテストの範囲が異なることから，テスト範囲を固定化したパッケージでは利便性に欠け，利用者が限られるので実用的ではない．さらに，テスト範囲を固定化して作業すると，TESRY データの開発や保守が困難になることも明確になった[13]．今後，テストケースを動的に展開する機能を TTG へ追加することを検討している．

### 8.2 ターゲットのバリエーション対応

FMP カーネルでは，以下のようなターゲットのバリエーションに依存して，実施不可となるテストケースが多数存在する．

- プロセッサ数
- ロック方式
- スピンロック方式
- タイマー方式
- ターゲット依存関数の有無

テストスイートとしては，すべてのターゲットを網羅するテストケースを開発するが，ユーザがターゲットに合わせてテストケースを手作業で取捨選択するのは利便性を大きく損なう．

そこで，ユーザが事前に定義したターゲットのバリエーションに合わせて，TTG が自動的にテストケースの取捨選択を行う機能を検討している．

## 9. 謝　　辞

TTG の開発にあたり，多くの仕様の提案，不具合の指摘を頂き，ツールとしての有用性や品質の向上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) 坂村健/監修，高田広章/編：$\mu$ITRON4.0 仕様 Ver.4.02.00，トロン協会 (2004).
3) Linux Test Project: “http://ltp.sourceforge.net/”.
4) Paul Larson, Nigel Hinds, Rajan Ravindran and Hubertus Franke: Improving the Linux Test Project with Kernel Code Coverage Analysis, *Linux Symposium* (2003).
5) Manoj Iyer: Analysis of Linux Test Project's Kernel Code Coverage, *LTP Whitepaper* (2002).
6) Subrata Modak and Balbir Singh: Building a Robust Linux kernel piggybacking The LinuxTest Project, *Linux Symposium* (2008).
7) Martin Bligh and Andy P.Whitcroft: Fully Automated Testing of the Linux Kernel, *Linux Symposium* (2006).
8) 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/”.
9) 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).
10) TOPPERS 新世代カーネル統合仕様書：“http://www.toppers.jp/docs/tech/ngki_spec-120.pdf”.
11) 鴫原一人，松浦光洋，眞弓友宏，本田晋也，山本雅基，高田広章ほか：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).
12) YAML: “http://yaml.org/”.
13) 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「組込みシステム (EMB)」 (2010).


© 2010 Information Processing Society of Japan