Vol.2011-SLDM-149 No.43
Vol.2011-EMB-20 No.43
2011/3/19

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

一場利幸[†1] 高瀬英希[†1] 鴫原一人[†1]
本田晋也[†1] 高田広章[†1]

マルチプロセッサ環境においては，複数プロセッサの実行タイミングに依存して実行するシナリオが異なることがある．ソースコードの全シナリオをテストするためには，そのようなタイミング依存シナリオを実行する手法が必要である．この手法として，テストプログラムを繰り返し実行する手法が考えられるが，タイミング依存シナリオを実行するまで長時間かかる可能性がある．本論文では，プロセッサの実行タイミングを制御可能な機構を提案し，タイミング依存シナリオを実行可能であることを示す．この機構を実装したシミュレータを利用し，マルチプロセッサ向けリアルタイム OS のタイミング依存シナリオをすべて実行可能であることを確認した．

The Simulation Mechanism Executing Timing-dependent Scenarios in a Multiprocessor Environment

Toshiyuki Ichiba,[†1] Hideki Takase,[†1]
Kazuto Shigihara,[†1] Shinya Honda[†1]
and Hiroaki Takada [†1]

In a multiprocessor environment, a scenario depends on an executing timing of processors. A methodology of executing it is necessary to test all scenarios of source code. One of the methodologies is an executing test program repeatedly. However, an executing time is maybe long. This paper proposes a mechanism of executing timing-dependent scenarios. We execute all timing-dependent scenarios of a real-time OS for multiprocessor using simulator with the mechanism.

1. はじめに

近年，大規模化・複雑化が進む組込みシステムにおいても，マルチプロセッサの重要性が増している．その背景には，消費電力の増大を抑えつつ処理性能の向上を図るためには，クロック周波数を上げるよりも，コア数を増やしたほうが有利であるという状況がある．

組込みシステムでは，ソフトウェアの再利用性やリアルタイム性の確保といった理由からリアルタイム OS（以下，RTOS）を用いるのが一般的となっている．我々は，ITRON 仕様をベースとしたシングルプロセッサ用の RTOS である TOPPERS/ASP カーネル（以下，ASP カーネル）と，マルチプロセッサに拡張した TOPPERS/FMP カーネル（以下，FMP カーネル）を開発してオープンソースとして公開している[1)]．

組込みシステムのソフトウェアを原因とする不具合が問題視されている中で，RTOS の品質の重要性が高まっている．我々は，RTOS の品質を高める取組みとして，ASP カーネル及び FMP カーネルに対するテストスイートの開発を実施している[2)]．テストスイートの目標として，ソースコードの条件網羅[⋆1]によるソースコードカバレッジ（以下，カバレッジ）を 100%とすることを目標としている．ASP カーネルに対しては，文献 2) において，API 呼び出しに着目したテスト（テストプログラム）によって，100%カバレッジを達成することができた．

一方，FMP カーネルにおいては，ASP カーネルと同様にテストプログラムを開発し実施したが，カバレッジは 100%とならなかった．これは，マルチプロセッサ環境においては，タイミング依存実行パス（シナリオ）が存在するためである．タイミング依存実行パスとは，プロセッサ間の実行タイミングが特定のパターンのときに実行されるパスである．

タイミング依存実行パスを実行する手法としては，プログラムを繰り返し実行するテスト（以下，連続試行テスト）が考えられる．しかし，連続試行テストでは，タイミング依存実行パスを実行する確率が低い場合，テスト対象のパスを実行するのに長い時間を要する．また，テストによっては，特定のパスを確実に実行したいという要求がある．

本論文では，マルチプロセッサ環境におけるタイミング依存実行パスを決定的に実行するためのプロセッサの実行タイミング制御機構を提案する．本手法の特徴はソフトウェアで実現可能な実行タイミング制御はソフトウェアで実現し，ソフトウェアで実現が困難な実行タ

†1 名古屋大学 大学院情報科学研究科
Graduate School of Information Science, Nagoya University
⋆1 条件式の判定による真偽を少なくとも 1 回は実行する

Vol.2011-SLDM-149 No.43
Vol.2011-EMB-20 No.43
2011/3/19

イミング制御のみをハードウェア（プロセッサのデバッグ機構やシミュレータ）で実現することにより，比較的単純なハードウェア機構で実現できることである．提案機構は，特定のプロセッサがプログラム中の指定した箇所を実行した場合（フック）に実行する処理（アクション）をあらかじめハードウェア機構に指定する．指定可能なアクションは任意のプロセッサの実行の停止や再開及び割込みの発生である．この機構とテストプログラムを組み合わせることで，タイミング依存実行パスを決定的に実行する．提案機構を実装したシミュレータを利用することで，マルチプロセッサ対応 RTOS のタイミング依存実行パスをすべて実行可能であることを示す．

本論文の構成は，次の通りである．2 章でタイミング依存実行パスについて述べ，3 章で実行タイミングを制御することでタイミング依存シナリオを実行可能であることを述べる．4 章では，実行タイミングを制御する機構を実装したシミュレータの評価を行う．5 章で関連研究について述べ，6 章でまとめる．

2. タイミング依存の実行パス

本章では，まず，シングルプロセッサ向け RTOS ではタイミング依存実行パスが存在しないことについて述べ，次にマルチプロセッサ向け RTOS でタイミング依存実行パスが存在する理由について述べる．また，タイミング依存実行パスを実行する手法である連続試行テストとその問題点について述べる．

2.1 シングルプロセッサ向け RTOS の実行パス

シングルプロセッサ向け RTOS として，ASP カーネルのテストを例に，シングルプロセッサ向け RTOS にタイミング依存実行パスが存在しないことを説明する．

RTOS のソースコードの多くは API に関連しているため，API に対する網羅的なテストを開発することで，カバレッジ 100%を達成することができると考えた．API 処理内のある実行パスを実行するかどうかは，API 呼び出し前の状態によって定まるため，API のテストプログラムは，実行したいパスを実行するための前状態を実現した後に API 処理を行い，その結果，期待した後状態となることを確認する．

API のテストプログラムの例として，セマフォ資源を獲得する `wai_sem` のテストについて述べる．まず，`wai_sem` の仕様では，セマフォ資源数が 1 以上あれば資源を獲得できるが，セマフォ資源数が 0 の場合は `wai_sem` を呼び出したタスクが待ち状態となる．前者のパスを実行するテストは，セマフォを 1 つ用意し，その資源数を 1 とした上で，そのセマフォに対して `wai_sem` を発行し，その結果，資源を取得することを確認すればよい．同様に，後者のパスを実行するテストもセマフォを 1 つ用意し，その資源数を 0 とした上で，そのセマフォに対して `wai_sem` を発行し，その結果が待ち状態となることを確認すればよい．他の API についても，同様にテストシナリオを作成することができる．

しかし，API 処理中に割込みが発生し，割込み処理を実行すると，パスが変化する可能性がある．`wai_sem` の例では，前状態でセマフォ資源数が 0 であっても，割込み処理で `isig_sem` を実行するとセマフォ資源数が 1 となり，パスが変化する．そのため，ASP カーネル（および FMP カーネル）では，API 実行時に割込みを禁止することで，API 処理中に実行パスが変わることがなく，ある前状態で API を呼び出すと，後状態が一意に定まる．

ASP カーネルでは，以上のような API テストを 1,668 件開発・実施することで，ターゲットシステム毎に異なる実装部分を除いて，カバレッジ 100%を達成することができた．

2.2 マルチプロセッサ向け RTOS の実行パス

FMP カーネルを例にマルチプロセッサ向け RTOS には，タイミング依存実行パスが存在することを示す．

FMP カーネルは ASP カーネルをベースにしているため，仕様とソースコードに共通点が多い．そこで，ASP カーネルの API テストと同様の考え方で FMP カーネルの API テストを 2,586 件開発した．このテストを実施した結果，カバレッジは 93.0%となり，多くのパスは ASP カーネルと同様に実行タイミングに依存しなかった．一方，実行出来なかった 7.0%のコードはタイミング依存実行パスである．

マルチプロセッサ環境では，並列して動作するプログラム（プロセッサ）の実行順序（タイミング）によって，プログラム振る舞いすなわち実行パスが異なることがある．具体的には，あるプログラムの分岐が，他の並列動作するプログラムとの間の共有資源の状態に依存しており，かつ分岐に影響を与えるタイミングで共有資源が変更される可能性がある場合，どの分岐が実行されるかは，並列して動作するプログラムの実行タイミングに依存する．

FMP カーネルにおける共有資源としては，タスク等のカーネルオブジェクトがある．API はこれらのカーネルオブジェクトを操作するため，API の先頭でロックによるプロセッサ間の排他制御を行っている．そのため，API 内部では基本的に実行中にカーネルオブジェクトすなわち共有資源の他のプロセッサからの変更は発生せずタイミング依存実行パスとはならない．

しかしながら，ロックを取得するコードはロックを他のプロセッサが取得している場合としていない場合の 2 種類の分岐を持ち，どちらを通るかは他のプロセッサとの実行タイミングに依存するタイミング依存実行パスとなっている．FMP カーネルの API の概要を**図 1**

Vol.2011-SLDM-149 No.43
Vol.2011-EMB-20 No.43
2011/3/19

```
 1: void api_example()
 2: {
 3:     // preprocess
 4:   retry:
 5:     if( !acquire_lock() ){
 6:        // ロック取得失敗時
 7:        enable_int();
 8:        disable_int();
 9:        goto retry;
10:     }
11:     // main process
12:     release_lock();
13:     // postprocess
14: }
```

図 1 API 例
Fig. 1 an example of API

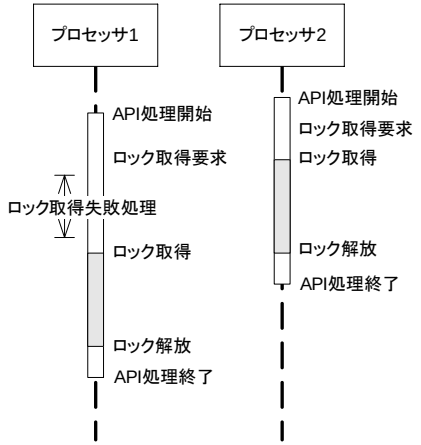

図 2 ロック取得失敗
Fig. 2 Lock acquisition failure

に示す．acquire_lock 関数は，ロック取得を 1 回試みて成功した場合に true を失敗した場合に false をリターンする．7-9 行目は，ロック取得に失敗したときに実行されるタイミング依存実行パスである．通常のスピンロックでは，ロック取得に失敗してもロック取得の試行を繰り返すのみだが，FMP カーネルでは，割込み応答性を確保するため，スピンロックの試行に失敗する毎に割込みを許可する．7-9 行目の処理が実行される実行タイミングを**図 2** に示す．図 2 では，プロセッサ 1 がロック取得を試みるがプロセッサ 2 が既にロックを取得しているため，ロック取得に失敗し，タイミング依存実行パスを実行する．

FMP カーネルでは，他にもロックの取得に関連するタイミング依存実行パスが複数存在する．図 1 の例では実行パスが変化しても最終的にはロックを取得して API を実行する．どちらの実行パスを実行したかは，実行結果に影響を与えないため，実行結果は一意である．しかしながら，実行パスによっては，API の実行結果が変化するものも存在する．

2.3 タイミング依存実行パスの実行の困難さ

タイミング依存実行パスの中には，分岐のいずれかが，並列動作するプログラムによる共有資源の書き換えでしか発生せず，かつ書き換えが分岐に影響を与えるタイミングが微小な期間にしか存在しない場合，その分岐を意図的に発生させることが困難になる．

FMP カーネルはそのような分岐がいくつか存在する．なぜならば，FMP カーネルはリアルタイム性確保のため，カーネルオブジェクトを操作する処理の中で，一部の割込みを許可しているからである．そのため，FMP カーネルのテストにおいてそのような分岐を網羅することが困難になっている．

2.4 連続試行テスト

タイミング依存実行パスを実行する方法として，タイミング依存実行パスを実行する可能性のあるテストプログラムを繰り返し実行する連続試行テストが考えられる．連続試行テストは，連続試行することにより，プロセッサ間の実行タイミングが “いつか” タイミング依存実行パスを実行するタイミングとなることを期待した手法である．そのため，実行タイミングが微小な期間にしか存在しない場合，連続試行テストではタイミング依存実行パスを実行するまでに長い時間を要する可能性がある．

また，通常パスとは API 実行結果が異なるタイミング依存実行パスに関しては，テストが実施できないという問題がある．API テストでは，API 発行後に期待した後状態となることを確認する．通常パスとタイミング依存実行パスで実行結果が異なる場合は，それぞれのパスで異なる後状態の確認が必要となる．しかしながら，テストプログラムからは，OS 内部のコードがどちらの実行パスを通ったか判断できないため，正しい後状態の確認ができない．

3. プロセッサ実行タイミング制御

前章で述べた問題を解決するためには，プロセッサの実行タイミングを指定して決定的にタイミング依存実行パスを実行する必要がある．ソフトウェアでもバリア同期等により，ある程度の同期は可能であるが，タイミング依存実行パスを実行するには不十分である．そこで，本研究では，タイミング依存実行パスを実行ために必要となるプロセッサの実行タイミング制御機能について述べる．

3.1 プロセッサ実行タイミングを指定したテスト

複数プロセッサの実行タイミングを指定し，そのタイミングに従って実行を進めると，タイミング依存実行パスを決定的に実行することが可能である．図 1 のタイミング依存実行パス（7-9 行目）を実行するように実行タイミングを指定したテストを実施することを考える．まずプロセッサ 1 を停止させた状態で，プロセッサ 2 から API を発行し API の先頭でロックを取得させた後に API の内部で実行を停止する．次に，プロセッサ 1 に API を発行させ，API の先頭でロックの取得を試みる（acquire_lock 関数を呼び出す）．このようなタイミングで実行することにより，プロセッサ 1 は必ずロック取得に失敗するため，タイ

Vol.2011-SLDM-149 No.43
Vol.2011-EMB-20 No.43
2011/3/19

ミング依存実行パスを決定的に実行できる．

3.2 プロセッサ実行タイミング制御の要件

FMP カーネルに存在するタイミング依存実行パスに対してそれらを実行するため必要な機能を検討し，プロセッサ実行タイミング制御機構に必要な要件（機能）を以下のように定めた．

（1）任意のアドレス実行時に，各プロセッサの実行と停止を制御することができる
（2）任意のアドレス実行時に，任意のプロセッサに割込みを発生させることができる
（3）上記制御の有効と無効を切り替えることができる

要件の策定にあたっては，ハードウェアやプロセッサシミュレータに対する要求を可能な限り低くする方針とした．そのため，ソフトウェアで実現可能な同期はソフトウェアで実現することとし，粒度の細かい（例えばクロック単位の）プロセッサ間の実行同期は必要としない．

要件（1）は，プロセッサ間の実行順序を制御するために必要となる．バリア同期やチェックポイントを用いればソフトウェアでも同様の制御は可能であるが，テストにおいては API 内部は変更することはできないため，用いることはできない．クロック単位で実行順序を指定する方法もあるが，タイミング依存実行パスを実行するという観点からは，アドレス単位（命令単位）で実行順序を制御できれば十分である．

要件（2）は，割込みハンドラからで呼び出す API についてタイミング依存実行パスのテストを実施するために必要である．

要件（3）については，タイミング依存実行パスのテストは，要件（1）と（2）の制御をあるアドレスに対して実施するが，タイミング依存実行パスを実行した後，要件（1）と（2）の制御を登録したアドレスを再度実行する可能性がある．タイミング依存実行パスを実行した後，再度登録アドレスを実行した際は要件（1）と（2）の制御が不要である．そのため，これらの制御について有効と無効を切り替えられることが望ましい．

3 つの要件を満たすことでタイミング依存実行パスを実行できるのは，FMP カーネルのように，別プロセッサに属するオブジェクトに対する操作を直接行う RTOS である．一方，別プロセッサに属するオブジェクトに対する操作をそのプロセッサに依頼する RTOS では，プロセッサの動作を停止してしまうと，停止したプロセッサに属するオブジェクトへの操作を行うことができない．

3.3 実行タイミング制御機構

前章で述べたプロセッサ実行タイミング制御の要件を実現するための具体的な機能について述べる．提案機構は，アドレスを指定するフックと実行するアクションを登録することで，実行タイミングの指定を行う．指定されたアドレスを実行した際に，指定されたアクションが実行される．これらの指定は，テストプログラム中で行う．

フックには，PC フック，カレントフックの二種類が存在する．PC フックは，登録したアドレスを実行した際にアクションを実行する．関数のアドレスを指定すると，関数を呼び出した時点でアクションを実行する．カレントフックは，プログラム実行中で登録されたらすぐにアクションを実行する．PC フックにより，任意のアドレスを指定することができるが，カレントフックを利用することにより，テストプログラムの作成が容易になる．

アクションには，割込み制御アクション，プロセッサ動作制御アクションの二種類が存在する．割込み制御アクションは，指定したコアに対して割込みを発生させることができる．プロセッサ動作制御アクションは，プロセッサの実行や停止を制御することができる．

フックとアクションを登録した後，有効と無効を設定できる．アクションを実行した後にそのまま有効とする，もしくは無効とするという設定が可能である．

フックとアクション，さらにこれらの有効と無効の設定により，前述の要件を全て満たすことができる．

提案機構において，実行タイミングを指定するために用いる関数を以下に示す．プロセッサ動作制御アクションの指定は，1 ビットが 1 プロセッサに対応しており，ビット値が 1 ならば実行，0 なら停止することを表す．

（1）`test_add_pchook_active(uint_t *pc, uint_t active_prc, bool_t cont)`
PC フックでプロセッサ動作制御アクションを登録する
（2）`test_add_pchook_raise(uint_t *pc, uint_t int_prc, bool_t cont)`
PC フックで割込み制御アクションを登録する
（3）`test_add_currenthook_active(uint_t active_prc)`
カレントフックでプロセッサ動作制御アクションを登録する
（4）`test_add_currenthook_raise(uint_t int_prc)`
カレントフックで割込み制御アクションを登録する

提案機構を用いて図 1 のタイミング依存実行パスのテストを行うプログラムを**図 3** に示す．`task11` と `task12` はプロセッサ 1 で，`task21` はプロセッサ 2 で動作するタスクである．`task11` は `task12` より優先度が低い．`task11` では，`acquire_lock` 関数呼び出し時に，プロセッサ 1 を止めプロセッサ 2 を実行することを登録し，一度アクションを実行した後，登録した制御は無効となる．プロセッサ 1 が実行，プロセッサ 2 が停止の状態で，`task11`

Vol.2011-SLDM-149 No.43
Vol.2011-EMB-20 No.43
2011/3/19

```
 1: void task11()
 2: {
 3:    test_add_pchook_active(acquire_lock, 0x2, ONE_SHOT); //t1
 4:    api_example();
 5: }
 6:
 7: void task12()
 8: {
 9:    test_add_currenthook_active(0x3); //t3
10: }
11:
12: void task21()
13: {
14:    test_add_pchook_active(release_lock, 0x1, ONE_SHOT); //t2
15:    act_tsk(TASK12);
16: }
```

図 3 実行タイミングを指定したテストプログラム
Fig. 3 Test program specifying execution timing

と task21 が実行可能状態，task12 が休止状態のとき，図 3 を実行すると，**図 4** のような順序で実行される．t1, t2, t3 でプロセッサ動作制御アクションが実行される．act_tsk は api_example と同じロックをとり，タスクを起動する API である．task11 はロック取得に失敗すると，割込み処理を受付け，より優先度の高い task12 にディスパッチする．

4. 評　　価

文献 3) のシミュレータに対して提案手法を適用した（TimingSim）．提案手法の有用性の評価のため，マルチプロセッサ対応 RTOS のタイミング依存の実行パスを連続試行テストと TimingSim により実行を試みる．対象としたマルチプロセッサ対応 RTOS は，FMP カーネル 1.2.0 であり，タイミング依存実行パスが 87 件存在する．

まず，連続試行テストによりタイミング依存実行パスの実行を試みた．実行には，FPGA に NiosII プロセッサを 2 個搭載したシステムを用いた．プロセッサの動作周波数は 50MHz である．連続試行テストは各タイミング依存実行パス毎に，そのパスを通る可能性のある前状態で API の呼び出しを繰り返し行う．5 つの関数に存在するタイミング依存実行パスを実行するまでに要した時間を**表 1** に示す．時間中の “未実行” は 2 時間以上試行を繰り返してもタイミング依存実行パスを通らなかったことを意味している．

wai_sem では図 1 のようにロック取得に失敗したときのパスを実行するのに要する時間を計測しており，短時間で実行できた．また，t_acquire_tsk_lock_self と ter_tsk についても短時間で実行できた．一方，acquire_nested_tsk_lock_without_preemption と wait_tmout は，2 時間以上実行してもタイミング依存実行パスを実行しなかった．

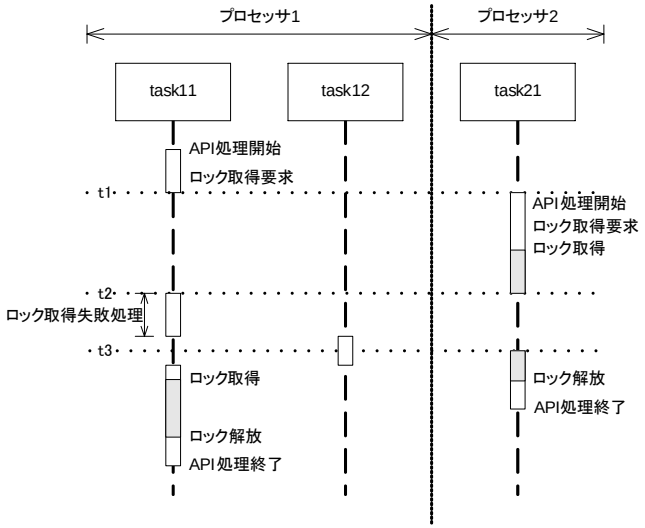

図 4 実行タイミングを指定したテストの実行シーケンス
Fig. 4 Sequence of test specifying execution timing

acquire_nested_tsk_lock_without_preemption については，ロック取得失敗時に割込み処理を受け付けない点が他のタイミング依存実行パスと異なる．wait_tmout 関数は FMP カーネルのシステムティックのためのタイマ割込みから呼び出される関数である．タイマ割込みは 1m 秒毎に発生するため，実行される頻度が他の連続試行テストと比較して低いため，実行できなかったと考えられる．

次に，TimingSim を用いて，FMP カーネルのタイミング依存実行パスの実行を試みた．連続試行テストと同様に，各タイミング依存実行パス毎に，そのパスを通る可能性のある前状態で API 発行するプログラムを用意した．さらに，プログラム毎にタイミング依存の実行パスを実行するための各プロセッサ実行タイミングを設計し，そのタイミングを 3.3 節で示した実行タイミング制御機構を用いて TimingSim へ指示するようにコードを追加した．

Vol.2011-SLDM-149 No.43
Vol.2011-EMB-20 No.43
2011/3/19

表 1 タイミング依存の実行パスの実行に要した時間
Table 1 The execution time of timing-dependent scenarios

関数名	時間
wai_sem	0.1 秒
t_acquire_tsk_lock_self	0.4 秒
ter_tsk	2.2 秒
acquire_nested_tsk_lock_without_preemption	未実行
wait_tmout	未実行

このプログラムと TimingSim を用いることにより，FMP カーネルに存在する全てのタイミング依存実行パスを実行可能であることを確認した．連続試行テストで実行が不可能であった 2 種類のパスに関しても実行することができた．それぞれのテストで行った実行タイミング制御は，acquire_nested_tsk_lock_without_preemption で 4 回，wait_tmout で 3 回である．

5. 関 連 研 究

タイミング依存の実行パスを実行する方法として，ソフトウェアとハードウェアのそれぞれのアプローチが存在する．

ソフトウェアによる方法として，マルチスレッドアプリケーションの単体テスト用ツールである ConTest[4] が挙げられる．ConTest は，テストプログラムの実行タイミングをずらして繰り返し実行することで，タイミング依存実行パスの実行を試み，バグの検出をサポートする．ConTest は連続試行テストを効率化するものであり，ConTest を利用しない場合と比べてテストの実施時間を削減することが可能であるが，実行確率が低いパスを実行するには時間を要すると考えられる．また，タイミング依存実行パスを決定的に実行することができない．

マルチプロセッサ上で動作するソフトウェアのデバッグをサポートするデバッグハードウェアが提案されている[5]．このデバッグハードウェアを利用することで，プロセッサの実行タイミングを指定することができ，提案手法とほぼ同様のことができる．しかし，デバッガでは割込みを考慮しておらず，割込み処理が関わるタイミング依存の実行パスを実行することができない．

また，タイミング依存の実行パスのないプログラミングモデルを提供する OS が提案されている[6]．これらの手法の対象はアプリケーションであるため，OS のタイミング依存の実行パスを無くすことはできない．同様の考え方で OS のタイミング依存の実行パスを無くすためには，ハードウェアにより OS を決定的に動作させる必要があるが，そのようなハードウェアは我々の知る限り存在しない．存在したとしても，性能が大きく限定されると予想される．

6. おわりに

マルチプロセッサ環境では，実行タイミングに依存してプログラムの実行シナリオが変化する．そのようなタイミング依存実行パスを実行するには，プロセッサの実行状態を指定したテストが有効である．本論文では，プロセッサの実行タイミングを制御可能な機構を提案し，タイミング依存シナリオを実行可能であることを示した．この機構を実装したシミュレータを利用し，マルチプロセッサ向けリアルタイム OS のタイミング依存シナリオをすべて実行可能であることを確認した．

今後の課題として，本論文で提案した機構をハードウェアにより実現することが挙げられる．

参 考 文 献

1) TOPPERS プロジェクト：http://www.toppers.jp/.
2) 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 東京 (2010).
3) 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 (2010).
4) Edelstein, O., Farchi, E., Nir, Y., Ratsaby, G. and Ur, S.: Multithreaded Java program test generation, *IBM Systems Journal*, Vol.41, No.1, pp.111 –125 (2002).
5) Mayer, A., Siebert, H. and McDonald-Maier, K.: Debug support, calibration and emulation for multiple processor and powertrain control SoCs, *Design, Automation and Test in Europe, 2005. Proceedings*, pp.148 – 152 Vol. 3 (2005).
6) Aviram, A., Weng, S.-C., Hu, S. and Ford, B.: Efficient System-Enforced Deterministic Parallelism, Technical report, 9th OSDI (2010). Comments: 14 pages, 12 figures, 3 tables.