情報処理学会論文誌 Vol.57 No.8 1737–1751 (Aug. 2016)

メモリ保護機能を持つAUTOSAR仕様ベースのリアルタイムOSに対するテスト

鴫原 一人[1,a)] 石川 拓也[1] 本田 晋也[1] 高田 広章[1]

受付日 2015年11月17日, 採録日 2016年5月17日

概要：車載ソフトウェアでは，安全性確保のために，メモリ保護機能が重要となっている．車載分野向けのソフトウェアプラットフォーム仕様であるAUTOSARのリアルタイムOSには，メモリ保護機能が規定されており，我々はAUTOSAR仕様をベースとしたTOPPERS/ATK2を開発，公開している．しかし，メモリ保護機能を有するリアルタイムOSに対するテストの事例はなく，メモリ保護機能に対するテスト手法やテストの規模は明らかになっていない．本研究では，我々が過去に実施したμITRONベースのリアルタイムOSに対するテストの研究成果を活用し，TOPPERS/ATK2を対象として実施した，メモリ保護機能を有するリアルタイムOSに対するテストについて述べる．我々は，メモリ保護機能に対するテスト手法を確立し，組合せツールやテストプログラム生成ツールを活用することで，テスト実施効率を向上させた．結果，TOPPERS/ATK2の不具合を19件検出し，メモリ保護機能を有するリアルタイムOSに対するテストの有用性を確認した．

キーワード：組込みシステム，リアルタイムOS，AUTOSAR，メモリ保護，MPU，テスト，TOPPERS

The Tests for Real-time OS Based on AUTOSAR Specification with Memory Protection Function

Kazuto Shigihara[1,a)] Takuya Ishikawa[1] Shinya Honda[1] Hiroaki Takada[1]

Received: November 17, 2015, Accepted: May 17, 2016

Abstract: Memory protection function plays a crucial role in guaranteeing automotive software safety. AUTOSAR specification prescribes memory protection function for automotive Real-Time OS. We have developed and published a Real-Time OS "TOPPERS/ATK2" based on AUTOSAR specification in our earlier research. However, past research has not clarified test methods and scales for Real-Time OS with memory protection function. The present study aimed to utilize our previous findings on test methods for μIRON based Real-Time OS, and to describe test methods for TOPPERS/ATK2, an AUTOSAR based Real-Time OS with memory protection function. The authors established new test methods for memory protection function by utilizing a combination tool as well as a test program generator to streamline the test execution. The present test methods detected 19 defects in TOPPERS/ATK2, and the usability of our test methods for Real-Time OS with memory protection function was confirmed.

Keywords: Embedded Systems, Real-Time OS, AUTOSAR, Memory Protection, MPU, Test, TOPPERS

1. はじめに

近年，自動車に搭載される電子制御ユニット（以下，ECU）の高度化・複雑化が進み，車載ソフトウェアの開発コストが急増している．また，1つのECUで実現する機能の数も増えており，複数のアプリケーションを1つのプロセッサ上に混在させたいニーズが増え，アプリケーション間での故障や不具合の伝搬を防止するために，メモリ保護機能の必要性も高まっている．ECUのような組込みシステムでは，汎用OSとは異なり，リアルタイム性の観点から，メモリマネジメントユニット（MMU）ではなく，メモリプ

[1] 名古屋大学
Nagoya University, Nagoya, Aichi 464–8601, Japan
[a)] shigihara.kazuto@h.mbox.nagoya-u.ac.jp

ロテクションユニット（以下，MPU）を用いて保護を実現することが一般的である．MPU は，MMU とは異なりアドレス変換を行わないため，TLB（Translation Lookaside Buffer）を必要とせず，リアルタイム性の保証がしやすい．MPU は，アドレス空間をいくつかの領域に分け，その領域ごとにアクセス権を設定可能である．

欧州では，AUTOSAR [1] という ECU 向けのソフトウェアプラットフォーム仕様が策定・公開されている．ソフトウェアプラットフォームの標準化により，車載ソフトウェア開発コストの削減を目指しており，国内でも AUTOSAR が普及しつつある．AUTOSAR では，車載ソフトウェアで共通的に使用されるリアルタイム OS（以下，RTOS）や，通信スタック，メモリスタックなどのソフトウェア仕様が規定されている．AUTOSAR 仕様の RTOS（以下，A-OS）[2] には，メモリ保護機能が規定されており，この機能により，アプリケーション間での故障や不具合の伝搬防止を実現する．

我々は，A-OS 仕様をベースとした RTOS である TOPPERS/ATK2（以下，ATK2）[3] を開発，公開している．A-OS 仕様では，MPU を用いてメモリ保護機能を実現するように規定されているが，MPU を使った具体的なコンフィギュレーションの方法は規定されていない．我々は，MPU を用いたメモリ保護機能を有する μITRON 仕様 [4] ベースの RTOS である，TOPPERS/HRP2 カーネル（以下，HRP2）[5] の設計 [6] を参考に，未規定の仕様を定め，ATK2 のメモリ保護機能を実装した．

車載ソフトウェアは，機能安全規格（ISO26262）で規定されているように，高い安全性と品質保証が求められる．ATK2 は，車載ソフトウェアの中核をなし，メモリ保護機能自体が正しく動作しないと，システム全体の安全性が保証できなくなるので，ATK2 のメモリ保護機能自体のテストは重要である．また，MPU は，プロセッサによって設定可能な属性や領域数，アライメント制約が異なるため，メモリ保護機能に対応した RTOS のポーティングは複雑である．したがって，ポーティング後に，メモリ保護機能が正しく振る舞うことを網羅的かつ効率的にテストできることが望ましい．

RTOS のテスト手法に関する研究として，並列性のあるタスクに関する仕様に対して，モデル検査により網羅的に検証する手法が提案されている [7], [8], [9], [10], [11]．一方，メモリ保護に関する仕様は並列性を考慮する必要はなく，モデル検査を使用しなくても，テストケースの組合せ網羅は可能であると考えられる．メモリ保護機能に対するテストにおける課題は，アクセスするメモリ領域やアクセス時の条件の膨大な組合せに対応したテストを実施する方法である．過去の研究では，タスクの振舞いに特化していることや，テストケースの生成のみでテストプログラムの生成が行えないことから，本研究には適用できない．

そこで我々は，過去に μITRON ベースの RTOS に対して研究開発したテストスイートである TTSP [12] を活用し，ATK2 に対するテスト手法を確立し，実施した．本論文の貢献は，MPU を用いたメモリ保護機能を有する RTOS をテストするためのテスト手法を確立し，テストの効率性と有用性を示すことである．また，A-OS 仕様の RTOS およびメモリ保護機能を有する RTOS に対するテストの規模を明らかにすることである．そして，本研究の成果を用いて，メモリ保護機能を持つ RTOS の品質を向上させることも，目的の 1 つである．

本論文の構成は次のとおりである．2 章で，テスト対象とした ATK2 について述べ，3 章で TTSP について述べる．4 章で確立したメモリ保護機能に対するテスト手法について述べ，5 章でテスト実施結果について評価を行う．6 章で関連研究について述べ，7 章でまとめる．

2. A-OS 仕様と ATK2 のメモリ保護機能

本章では，A-OS 仕様と，ATK2 のメモリ保護機能について説明する．なお，A-OS 仕様には，4 つのスケーラビリティクラス（以下，SC）が定義されているが，本論文では，メモリ保護機能に対応したスケーラビリティクラス 3（以下，SC3）のみを対象とする．

2.1 A-OS 仕様

A-OS 仕様におけるメモリ保護機能では，タスクやリソースといったオブジェクトを，OS アプリケーション（以下，OSAP）というグループに所属させ，OSAP ごとにアクセスが許可されたオブジェクトやメモリ領域にのみアクセスを可能とすることで，保護を実現する．OSAP には，信頼 OSAP と非信頼 OSAP があり，信頼 OSAP に所属した処理単位はいっさいのアクセス制限を受けず，すべてのオブジェクトやメモリ領域にアクセスでき，非信頼 OSAP に所属した処理単位は，所属した OSAP に許可されたオブジェクトやメモリ領域にのみアクセスできる．なお，RTOS 上でプログラムを実行するタスクや割込みサービスルーチン（以下，ISR）といったオブジェクトを，本論文では処理単位と呼ぶ．

A-OS 仕様におけるメモリ保護機能は，以下に説明する API 保護とアクセス保護の 2 種類に分類される．

2.1.1 API 保護

API 保護は，OS から提供される API を非信頼 OSAP の処理単位から発行した際に実行されるメモリ保護機能である．非信頼 OSAP の処理単位から，あるオブジェクトに対して API を発行すると，OS がそのオブジェクトに対してアクセス権があるかをチェックし，アクセス権がない場合，API の処理を実行しない．また，引数にポインタを渡してデータを受け取る API の場合，API 呼び出し元の処理単位がポインタの指し示すアドレスへ，データを書き込

むアクセス権があるかを OS がチェックし，アクセス権がない場合，API の処理を実行せず，不正なデータの書き込みを防止する．

同一 OSAP に所属するオブジェクトどうしはアクセス権が与えられるが，これとは別に，オブジェクトごとにアクセス可能な OSAP を，コンフィギュレーション時に指定することで，他の OSAP のオブジェクトに対してもアクセス権を与えることができる．

アクセスが許可されていないオブジェクトやアドレスに対して API を発行した場合，違反を検出した結果として，OS がエラーフックという処理単位を実行する．なお，それぞれの OSAP に対して，OSAP 固有のエラーフック，OSAP 固有のスタートアップフック（OS 起動時に呼び出される），OSAP 固有のシャットダウンフック（OS 終了時に呼び出される）を定義することができる．

2.1.2 アクセス保護

アクセス保護は，API を使用することなく，非信頼 OSAP から変数や関数などのメモリ領域へ直接アクセスした際に実行されるメモリ保護機能である．非信頼 OSAP の処理単位から，あるメモリ領域に対してアクセスを行うと，MPU によってアクセス権があるかがチェックされ，アクセス権がない場合，アクセスが拒否され，CPU 例外が発生する．アクセス権のないメモリ領域へアクセスを行い，MPU による CPU 例外が発生することを，アクセス保護違反と呼ぶ．

A-OS 仕様には，MPU を用いて非信頼 OSAP の処理単位からの不正なアクセスを保護することは，要求仕様として規定されているが，具体的に MPU をどのように使用するかや，コンフィギュレーション方法などはいっさい規定されていない．これは，プロセッサによって MPU で設定できるメモリ領域数や設定方法が様々であり，標準化が困難であるためと考えられる．

アクセス保護違反が発生すると，違反を検出した結果として，OS がプロテクションフックという処理単位を実行する．OS のユーザは，プロテクションフックの中で，状況に応じた復帰処理やシャットダウンを行うことができる．プロテクションフックでは，プロテクションフックの戻り値によって，OS の振舞いを指定することが可能である．また，プロテクションフックを呼び出さないように設定することも可能であり，この場合，アクセス保護違反が発生すると，即 OS をシャットダウンする．これらのアクセス保護違反が発生した際に行う処理のことを，保護違反時処理と呼ぶ．

2.2 ATK2 のメモリ保護機能

本節では，HRP2 を参考に実装した ATK2 のメモリ保護機能について説明する．ATK2 によるメモリ保護設定の例を**図 1** に示す．

図 1 の例では，1 つの信頼 OSAP と 2 つの非信頼 OSAP

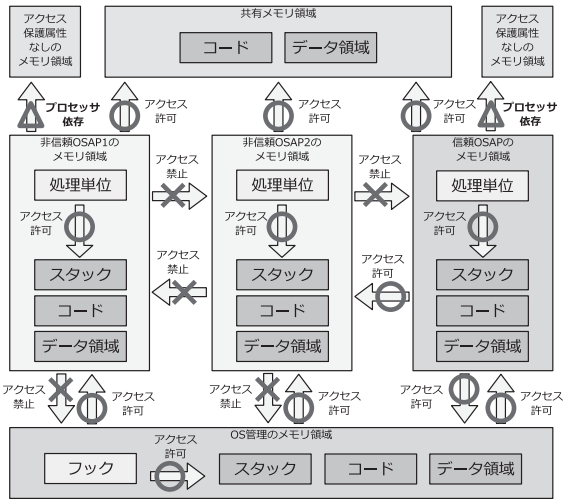


図 1 ATK2 によるメモリ保護設定の例
Fig. 1 An example of memory protection setting by ATK2.

（OSAP1 と OSAP2）を設定している．ほかに OS が使用する領域として，各 OSAP に含まれる OS オブジェクトやメモリ領域以外にも，OS 管理のメモリ領域，共有メモリ領域，アクセス保護属性なしのメモリ領域も存在する．非信頼 OSAP は，自身に所属する OS オブジェクトやメモリ領域はアクセスできるが，それ以外に対してはアクセスが禁止される．信頼 OSAP は，無制限にいずれのアクセスも許可される．なお，アクセス保護属性を設定していない，あるいはプロセッサの仕様上，アクセス保護属性を設定できないメモリ領域に対してアクセスを行った場合に，プロセッサがどのように振る舞うかは，プロセッサによって異なる．

2.2.1 A-OS 仕様と ATK2 実装の差異

A-OS 仕様で規定されているメモリ保護機能仕様には，実装するうえでの課題があり，ATK2 では実装していない仕様がある．具体的には，非信頼 OSAP の ISR 所属制限と，プロテクションフックの戻り値制限の 2 点がある．

- 非信頼 OSAP の ISR 所属制限
  A-OS 仕様では，非信頼 OSAP に ISR が所属することができる．ISR が非信頼 OSAP に所属するということは，ISR がメモリ保護違反を引き起こした場合に，その ISR を強制終了する可能性があるということである．一般に，ISR は早い応答性が求められる処理を実行する処理単位であるので，レジスタの退避，復帰やスタック切替えは最小限の処理にとどめるべきである．また，割込み優先度に応じて，複数の ISR が多重に発生して起動することがあるので，すべての ISR 用にスタックを 1 つだけ用意して，スタックを切り替えずに，すべての ISR を実行したほうが効率的である．しかし，非信頼 OSAP に所属する ISR は，メモリ保護違反などによって強制終了する可能性があるので，前述の処理の効率化を適用できず，割込み発生時

のオーバヘッドが大きくなってしまう．以上の理由から，ATK2では，ISRは信頼OSAPにのみ所属する実装とした．

- プロテクションフックの戻り値制限
  A-OS仕様において，プロテクションフックの戻り値として指定できるのは以下の5つである．
  - PRO_SHUTDOWN
    OSをシャットダウンする
  - PRO_TERMINATETASKISR
    アクセス保護違反を引き起こしたタスク/ISRを強制終了する
  - PRO_TERMINATEAPPL
    アクセス保護違反を引き起こしたタスク/ISRが所属するOSAPを強制終了する
  - PRO_TERMINATEAPPL_RESTART
    アクセス保護違反を引き起こしたタスク/ISRが所属するOSAPを再起動する
  - PRO_IGNORE
    無視する（アクセス保護違反は無視できないためシャットダウンされる）

  上記の中で，PRO_TERMINATEAPPLとPRO_TERMINATEAPPL_RESTARTは，動作中のOSAPを強制的に停止させるものである．OSAPには，リソースやカウンタといったOSオブジェクトも所属しているが，OSAPが停止した場合に，これらのOSオブジェクトに関連する他のOSオブジェクトがどのように振る舞うかが，A-OS仕様には規定されていない．たとえば，非信頼OSAPに所属しているリソース[*1]を，他のOSAPに所属しているタスクが取得している状態で，非信頼OSAPが強制終了された場合，取得中だったリソースがどう処理されるかが規定されていない．排他制御に使用中のリソースが，意図せず開放されてしまうのは問題であるが，開放しないままでいると，タスクが開放しようとした際に，リソースの所属OSAPが停止状態のためエラーとなってしまう．このように，A-OS仕様では，OSAPが停止する場合の仕様が不明確であるので，ATK2では，PRO_TERMINATEAPPLとPRO_TERMINATEAPPL_RESTARTに対する処理を未実装とした．また，ISRが非信頼OSAPに所属できないことから，PRO_TERMINATETASKISRで強制終了する対象は，タスクのみとなる．

2.2.2 API保護の実現方法

ATK2では，各オブジェクトが所属するOSAPをOSが管理し，API発行時に，呼び出し元の処理単位が所属するOSAPを確認し，APIの引数に与えれたオブジェクトやアドレスのチェックを行う．

あるタスクから別のタスクの状態を取得するAPIであるGetTaskStateを用いて，API保護について説明する．GetTaskStateの引数は，状態を取得したいタスクID（TaskID）と，取得した結果を格納する変数へのポインタ（State）の2つである．

例として，非信頼OSAP1に所属するTASK1が，異なるOSAP（非信頼OSAP2）に所属するTASK2に対して，GetTaskStateを呼び出すこととする．Stateには，通常呼び出し元のTASK1が指定した変数へのポインタが与えられる．Stateに不正なアドレスを指すポインタが指定される可能性があるので，TASK1からGetTaskStateが呼び出されると，まずStateが指し示すアドレスへ，TASK1が書き込み可能かを，APIの中でOSがチェックする．

ATK2では，各OSAPがアクセス可能なメモリ領域の情報を，コンフィギュレーションに応じて生成するメモリ保護情報ファイルに定義された内容から参照することできる．メモリ保護情報ファイルについては，2.2.4項で説明する．タスクの状態を確認する変数のサイズは1 byteであり，Stateが指し示すアドレスから1 byte分のメモリ領域に対して，TASK1が書き込み可能かをOSがチェックし，書き込み不可であれば，エラーとして後続の処理を実行しない．

TASK1からStateが指し示すアドレスへアクセス可能である場合，OSは次に，非信頼OSAP1がTASK2に対してアクセス可能かをチェックする．OSはまず，呼び出し元のTASK1が所属するOSAP（非信頼OSAP1）を取得し，TASK2にアクセス可能なOSAPのリストに，非信頼OSAP1が含まれているかをチェックする．リストに含まれていなければ，エラーとして後続の処理を実行しない．

2.2.3 アクセス保護の仕様設計

本項では，A-OS仕様で未規定であるアクセス保護に対して，我々が規定したATK2の仕様について説明する．

MPUは，アドレス空間をいくつかの領域に分け，その領域ごとにアクセス権を設定可能であるが，MPUで扱うことのできる領域の数は有限であり，多くの場合，8領域程度である[6]．そのため，同じアクセス権を設定する必要のあるコードやデータを，連続した番地に配置するように，静的にメモリ配置を行い，MPUに設定するメモリ領域の数を，MPUが扱うことのできる領域数に収まるようにする必要がある．これをふまえ，ATK2では，HRP2の設計を参考に，以下の設計とした．

- MPUに関する操作（初期化や設定の書き換え，有効化・無効化）は，OSの内部処理で行う．
- 同じアクセス権を設定する必要のあるコードやデータを，連続した番地に配置するように，OSが静的にメモリ配置を行う．このとき，MPUに設定するメモリ領域を，MPUの制限に合わせて適切にアライメント

[*1] タスク/ISR間の排他制御に用いるOSオブジェクト．

する．

- OS のユーザは，MPU に設定する具体的な番地やサイズを知ることなく，セクションやオブジェクトファイル単位でアクセス保護属性の設定ができる．ここで，セクションとは，リンカスクリプトが扱う単位であり，同じメモリ配置の属性を持ったコードやデータのまとまりである．
- 信頼 OSAP は，プロセッサの CPU コアを特権モード（アクセス保護が無効な状態）で実行し，非信頼 OSAP は，ユーザモード（アクセス保護が有効な状態）で実行する．
- 非信頼 OSAP から，OS が提供する API を呼び出した場合，ユーザモードから特権モードへ切り替えて処理を行い，API からリターンする際にユーザモードへ切り替える．

また，ATK2 が管理対象とするメモリ領域は，以下の 4 つの領域に分類した．

- 非信頼 OSAP のメモリ領域
- 共有メモリ領域
- OS 管理のメモリ領域
- アクセス保護属性が設定されていないメモリ領域

非信頼 OSAP のメモリ領域，共有メモリ領域，OS 管理のメモリ領域は別々のセクションに分かれ，各セクションにアクセス保護属性が設定される．信頼 OSAP のメモリ領域は，OS 管理のメモリ領域と区別する必要がないため，実質的に OS 管理のメモリ領域に包含される．アクセス保護属性が設定されていないメモリ領域は，非信頼 OSAP からの読み出し，書き込み，実行アクセスが禁止される．各メモリ領域は**表 1** に示したアクセス保護属性を設定することができる．

2.2.4 アクセス保護の実現方法

本項では，ATK2 のアクセス保護の実現方法について説明する．実現方法の概要図を**図 2** に示す．

まず，ユーザは実現したいオブジェクトや OSAP の設定を含めたメモリ保護に関する設定を記述したコンフィギュレーション情報ファイルを作成する（図 2 [1]）．AUTOSAR では，各モジュールのコンフィギュレーション情報を XML で記述する仕様となっており，通常は GUI ツールを用いてコンフィギュレーションを行い，XML を出力する．2.1.2 項で述べたとおり，アクセス保護設定に関するコンフィギュレーションパラメータに関しては，A-OS 仕様では定義されていないため，ATK2 では HRP2 の静的 API によるコンフィギュレーションをベースに，独自に拡張を行った．

XML で記述したコンフィギュレーション情報ファイルを入力として，ジェネレータと呼ぶツールにより，C 言語ファイル，リンカスクリプトを生成する（図 2 [2]）．C 言語ファイルには，OSAP を管理，制御するうえで必要な情報や，アクセス権の情報が，ATK2 ソースコードから使用できる形式で出力される．リンカスクリプトには，2.2.3 項で述べたとおり，有限である MPU の領域数を節約するために，最適なメモリ配置を実現するためのコードが出力される．これらの生成したファイルをメモリ保護情報ファイルと呼ぶ．

メモリ保護情報ファイルと，ATK2 ソースコード，各アプリケーションを実装したファイルを，コンパイラによってビルドして実行モジュールを作成する（図 2 [3]）．

実行モジュールがプロセッサにロードされ，非信頼 OSAP の処理単位が実行されると，MPU を使用したアクセス保護が機能する（図 2 [4]）．図 2 では，非信頼 OSAP1 の処理

表 1 メモリ領域に対するアクセス保護属性
Table 1 The protection setting for memory area.

非信頼 OSAP のメモリ領域	
	スタック （自処理単位からのみ読み出し，書き込み可能）
	コード （自 OSAP からのみ実行可能）
	専有リードオンリーデータ （自 OSAP からのみ読み出し可能）
	共有リード専有ライトデータ （自 OSAP からのみ書込可能，他 OSAP からは読み出しのみ可能）
	専有リードライトデータ （自 OSAP からのみ読み出し，書き込み可能）
共有メモリ領域	
	コード （全 OSAP から実行可能）
	共有リードオンリーデータ （全 OSAP から読み出しのみ可能，書き込み不可能）
	共有リードライトデータ （全 OSAP から読み出し，書き込み可能）
OS 管理のメモリ領域	
	スタック （OS，信頼 OSAP からのみ読み出し，書き込み可能）
	コード （OS，信頼 OSAP からのみ実行可能）
	専有リードオンリーデータ （自 OSAP からのみ読み出し可能）
	専有リードライトデータ （自 OSAP からのみ読み出し，書き込み可能）

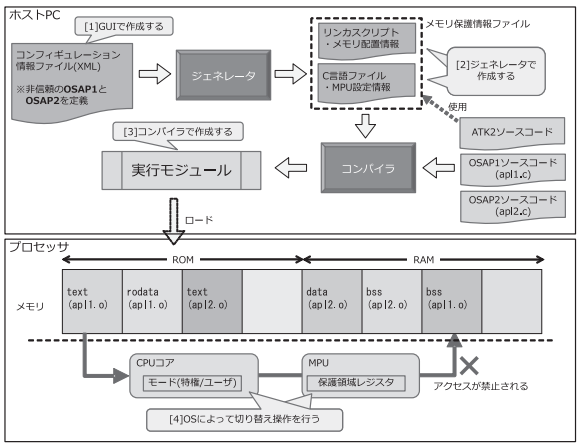


図 2 アクセス保護の実現方法
Fig. 2 The realization method of access protection.

単位から非信頼 OSAP2 のデータ領域へはアクセスができないコンフィギュレーションとなっている．したがって，OSAP1 の処理単位において，OSAP2 の bss 領域へのアクセスを CPU コア上で実行しようとしても，MPU がアクセス違反を検知し，CPU 例外を発生させることで，OSAP2 の bss 領域が保護される．

CPU コアが特権モードとユーザモードのどちらで動作するか，および MPU に設定する保護領域レジスタの情報を，OS が各 OSAP の実行状況に応じて，適切に切り替えることによって，システム全体としてのアクセス保護を実現する．

3. TTSP の概要

我々は，これまでに μITRON ベースの RTOS である，TOPPERS/ASP カーネル（以下，ASP）[13] と TOPPERS/FMP カーネル [14] に対応したテストスイートを開発する過程で，RTOS に対するテストプロセスやテスト手法に関して研究を行った [15]．開発したテストスイートは TTSP（TOPPERS Test Suite Package）と命名した．この研究成果をふまえ，ATK2 に対するテストは，TTSP を A-OS 仕様に適合し，さらにメモリ保護機能に対するテストを追加して実施することにした．本章では，TTSP の概要について説明する．

3.1 API テスト

TTSP が対象とするテストは，主に RTOS がアプリケーションに提供する API に対するテストである．これは，RTOS の API に不具合や仕様に反する処理が存在すると，RTOS 上で動作するアプリケーションが誤作動する原因となりうることや，API の実装が RTOS のソースコードの大部分を占めていることから，RTOS のテストにおいて，最も重要であると考えたからである．

多くの API は，発行により，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）を変化させる．そこで，API テストでは，あるシステム状態で API を発行し，仕様どおりのシステム状態の変化が起きることを確認する．具体的には，テストプログラムで，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．この一連のテストの流れをテストシナリオと呼ぶ．

RTOS の仕様書から，テストで確認する API 発行に対する RTOS の振舞いをテストケースとして抽出し，テストケースごとにテストシナリオを作成する．テストケースの例を**図 3** に示し，図 3 のテストケースに対応するテストシナリオの例を**図 4** に示す．

```
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 3 テストケースの例
Fig. 3 An example of the test case.

```
前状態
  低優先度の TASK1 が実行状態
  高優先度の 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 4 テストシナリオの例
Fig. 4 An example of the test scenario.

```
pre_condition:
  TASK1:
    type   : TASK
    tskpri : LOW
    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 5 TESRY 記法の例
Fig. 5 Example of the TESRY notation.

3.2 TESRY 記法

次節で説明するテストプログラム生成ツールのために，テストシナリオの記述方法として，TESRY（TEst Scenario for Rtos by Yaml）記法を定めた．TESRY 記法で記述したデータファイルを TESRY データと呼ぶ．図 4 のテストシナリオを TESRY 記法で記述した例を**図 5** に示す．

3.3 TTG

TTG（Toppers Test Generator）は，TESRY データ（テストシナリオ）を入力として，テストプログラムを生成するツールである．以下に，TTG によるテストプログラム生成の概要について説明する．

TTG は，まず TESRY データに定義された前状態を実現するために，RTOS の仕様に基づいて，各オブジェクトを対象とする状態へ遷移させるソースコードを生成する．前状態を実現した後で，システム状態を確認するための API（以下，システム状態確認 API）を用いてすべてのオブジェクトが，TESRY データに定義された前状態と一致してい

ることを確認するソースコードを生成する．次に，TESRY データに定義された処理を，そのまま実行するソースコードを生成する．API の戻り値が指定されている場合，戻り値のチェックを行うソースコードも生成する．最後に，システム状態確認 API を用いて，すべてのオブジェクトが，後状態で定義された状態となっていることを確認するソースコードを生成する．

また，TTG は，複数の TESRY データを入力することで，それらをまとめたテストプログラムを生成する．TTG へ入力する TESRY データの数を調整することで，ターゲットシステムが搭載するメモリサイズに収まるテストプログラムを生成することが可能である．

TTG，および TESRY 記法の詳細については文献 [16] を参照のこと．

4. API テストの A-OS 仕様への適合とアクセス保護テスト

3.1 節で述べたとおり，ATK2 においても，API テストが重要であると考え，TTSP を A-OS 仕様に適合し，API テストを実施することにした．しかし，TTSP が対象とする RTOS は，メモリ保護機能に対応していないため，TTG は非信頼 OSAP に所属する処理単位が受ける制約を考慮していない．たとえば，メモリ保護機能がない RTOS のテストプログラムでは，システム状態確認 API や前状態を実現するための API は，どの処理単位からでも発行が可能である．しかし，メモリ保護機能がある場合，非信頼 OSAP に所属する処理単位は，アクセスできる OS オブジェクトやメモリ領域に制限があるので，TTG のテストプログラム生成処理に対応が必要である．

また，テストプログラムを RTOS 仕様の違いに対応するだけでなく，API 保護に対するテストケースを，API テストに追加する必要がある．たとえば，アクセス権のない OS オブジェクトを引数に与えてエラーとなることを確認するテストケースや，アクセス権のないメモリ領域へのポインタを引数に与えてエラーとなることを確認するテストケースなどの追加が必要となる．さらに，アクセス保護は，許可されていないアクセスを監視，検出する機能であるので，API テストだけを実施しても，アクセス保護に対するテストはできない．

そこで，API テストでは，API 保護に対するテストケースを追加し，API テストとは別に，アクセス保護が正しく動作することを確認するテスト（以下，アクセス保護テスト）を新たに実施した．本章では API テストの A-OS 仕様への適合と，新規に実施したアクセス保護テストについて説明する．なお，ATK2 向けのテストスイートは，AKTSP（Automotive Kernel Test Suite Package）と命名した．

```
pre_condition:
  TASK1:
    type   : TASK
    curpri : LOW
    tstat  : RUNNING
    ttype  : AK_BASIC
    spolicy: AK_SCHFULL
    osap   : NON_TRUSTED_OSAP1
  TASK2:
    type   : TASK
    inipri : HIGH
    tstat  : SUSPENDED
    ttype  : AK_BASIC
    spolicy: AK_SCHFULL
    osap   : NON_TRUSTED_OSAP2
do:
  id     : TASK1
  syssrv : ActivateTask(TASK2)
  rettype: StatusType
  retval : E_OK
post_condition:
  TASK1:
    tstat: READY
  TASK2:
    tstat  : RUNNING
    curpri : HIGH
```

図 6 A-OS 仕様に対応した TESRY 記法の例
Fig. 6 Example of the TESRY notation for A-OS.

4.1 API テスト

API テスト[*2]では，TESRY 記法，TTG に対して A-OS 仕様への適合を行った．

4.1.1 TESRY 記法の適合

TESRY 記法は，μITRON 仕様に対するテストシナリオの記述方法と同様に，A-OS 仕様で定められたオブジェクトごとに必要なパラメータを規定した．図 5 と同等のテストシナリオを，A-OS 仕様の TESRY 記法で記述した例を**図 6** に示す．

A-OS 仕様では，タスク種別 [ttype] として基本タスクと拡張タスクが指定できる．また，タスクごとにスケジューリングポリシ [spolicy] として，フルプリエンプティブかノンプリエンプティブかを指定できる．また，排他制御の仕組みにより，タスク起動時に優先度が変化することがあるので，現在優先度 [curpri] と初期優先度 [inipri] を区別する必要がある．さらに，タスクが所属する OSAP [osap] を指定する必要がある．

μITRON 仕様で，タスクを起動する API として発行していた act_tsk に相当する A-OS 仕様のタスクを起動する API は ActivateTask である．μITRON 仕様では，API の戻り値のデータ型は 1 種類であるが，A-OS 仕様では複数のデータ型が戻り値になるので，データ型 [rettype] も指定する必要がある．

4.1.2 TTG の適合

TTG を A-OS 仕様に適合させ，AKTG（Automotive Kernel Test Generator）と命名した．AKTG は，構成や

[*2] AKTSP では，仕様書の表記に合わせて API テストをシステムサービステストと呼称する．

生成テストプログラムの構造は TTG を踏襲し，前項で述べた A-OS 仕様に対応した TESRY データを入力として動作する．A-OS 仕様は，μITRON 仕様と同じく優先度ベースのフルプリエンプティブスケジューリングを行う RTOS であるので，TTG が生成するテストプログラムの基本構成を，そのまま使用可能であった．

メモリ保護機能のテスト実現のために，AKTG では主に以下の設計を行うことが必要であった．

- 非信頼 OSAP からのシステム状態確認 API 発行
  システム状態確認 API は，OS 管理のメモリ領域に配置されるデータを参照する必要があるため，システム状態確認 API を発行する処理単位が非信頼 OSAP に所属していた場合，アクセスができない．したがって，OS が提供する API と同様に，非信頼 OSAP からシステム状態確認 API を呼び出した場合，特権モードへ切り替えて実行することが必要である．
- 非信頼 OSAP からの前状態の実現
  前状態の実現のために，テストシナリオに登場する処理単位から，いくつかの API を発行する必要があるが，呼び出し元の処理単位が非信頼 OSAP の場合，呼び出し先の OS オブジェクトによっては，API 保護によりアクセス権エラーとなる可能性がある．そこで，前状態実現の処理に限り，信頼 OSAP が提供する信頼関数という機能を利用し，非信頼 OSAP からでも任意の OS オブジェクトに対して API を発行できるようにする必要がある．2.2.2 項で述べたとおり，OS が提供する API は，呼び出し元の所属 OSAP からアクセス可否を判定するので，システム状態確認 API とは異なり，特権モードで動作させても，アクセス権がなければエラーとなってしまう．したがって，前状態を実現するための API では，呼び出し元の所属 OSAP を信頼 OSAP に切り替える信頼関数を使用する必要がある．
- スタック領域の分離
  メモリ保護機能がない RTOS の場合，タスクが使用するスタック領域に区別がないので，テストシナリオに登場するタスクの数だけスタック領域を用意すればよい．信頼 OSAP のタスクは，タスクの数だけスタック領域を用意すればよいが，非信頼 OSAP のタスクは，各非信頼 OSAP からアクセス可能なメモリ領域に，それぞれの非信頼 OSAP に所属するタスクの数だけ，スタック領域を確保する必要がある．
- 非信頼 OSAP からの OS シャットダウン
  テストプログラム実行中に何らかのエラーが発生した場合，不正な処理が続行しないように，OS シャットダウンによりテストプログラムを終了させる必要がある．しかし，A-OS 仕様では，非信頼 OSAP の処理単位から OS シャットダウンを実行できない．そこで，前状態を実現するための API と同様に，OS シャットダウンを行う信頼関数を用意し，エラー発生時に限り，非信頼 OSAP からも OS シャットダウンを実行可能とする必要がある．

なお，TTG では，TESRY データに記載された 1 パラメータを，テストプログラムでも 1 行ずつ記述してチェック処理を実行していたのに対し，AKTG のテストプログラムでは，全パラメータチェックする関数呼び出し 1 行でチェックするように変更した．前項で述べたタスクのように，A-OS 仕様は μITRON 仕様と比べて，設定できるパラメータが多いので，1 パラメータを 1 行ずつ記述してチェックすると，1 つの関数のコードサイズが大きくなってしまう．コンパイラによっては，1 つの関数のコードサイズが大きいとコンパイルできなくなることがあるので，これを避けるため，全パラメータをチェックする関数呼び出し 1 行で実行するように変更して，コードサイズを削減した．

4.1.3 テストケース数増加への対応

A-OS 仕様でテスト対象とした API の数は 57 個であり，μITRON 仕様の 103 個と比べて半分程度しかなく，API テストのテストケース数は，TTSP よりも少なくなるものと思われた．しかし，4.1.1 項で述べたタスクのように，A-OS 仕様は μITRON 仕様と比べて，設定できるパラメータが多いことに加えて，信頼 OSAP と非信頼 OSAP の組合せや，API 保護に対するテストケースの追加が必要である．これらの理由により，TTSP で 1,629 件であったテストケース数が，AKTSP では 40,000 件以上になることが判明した．各パラメータの設定値が変わることで，API の振舞いも変わることから，パラメータの組合せを減らすことはできず，全組合せをテストする必要がある．しかし，40,000 件という膨大な TESRY データを手作業で作成するのは，保守性の面でも困難であるので，何らかの組合せツールを用いて，TESRY データ作成の効率化を行うべきと考えた．

テストケースの組合せ生成ツールはいくつか存在するが，テストケースの組合せだけ生成できても，膨大なテストケースに対して 1 つ 1 つ TESRY データを作成するのは現実的ではない．そこで，テストケースの組合せだけでなく，生成したテストケースごとの期待値（テスト結果）も論理的に生成可能である PictMaster [17] を選択した [18]．PictMaster は，Excel ベースで使用できるツール（VBA で実装）であり，VBA のコードがオープンソースで自由に拡張できることも，PictMaster を選択した理由である．

PictMaster は，組み合わせるパラメータとそれぞれのパラメータがとりうる値，各パラメータどうしの関係により発生する制約，組合せごとに得られる結果という 3 つのフィールドに分けてデータを入力し，入力内容に応じたすべてのパラメータの組合せと結果を出力する．PictMaster の標準の出力形式が Excel ファイル形式であったため，PictMaster の出力を TESRY データに変換しやすいように，csv ファ

パラメータ	組み合わせ対象値
call_origin	TASK, C2ISR
org_curpri	TSK_PRI_MID
org_ttype	AK_BASIC, AK_EXTENDED
org_spolicy	AK_SCHFULL, AK_SCHNON
other_tstat	SUSPENDED, READY
other_curpri	TSK_PRI_LOW, TSK_PRI_MID, TSK_PRI_HIGH

制約表

パラメータ	仕様制約1
call_origin	
other_tstat	READY
other_curpri	#TSK_PRI_HIGH
other_ttype	
other_evt_flg	
org_spolicy	AK_SCHFULL
other_actcnt	

TESRY生成表

TESRY_param	call_origin	org_ttype	org_spolicy	other_tstat
&pre_condition,	TASK, C2ISR			
&TASK1, type, TASK, tstat, RUNNING, inipri, TSK_PRI_MID,	TASK, C2ISR			
&ttype, AK_BASIC,	TASK, C2ISR	AK_BASIC		
&ttype, AK_EXTENDED,	TASK, C2ISR	AK_EXTENDED		
&spolicy, AK_SCHFULL,	TASK, C2ISR		AK_SCHFULL	
&spolicy, AK_SCHNON,	TASK, C2ISR		AK_SCHNON	

図 7 PictMaster ファイルの例
Fig. 7 An example of The PictMaster file.

イル形式で出力するように，PictMaster の VBA の修正を行った．そのほかにも，利便性向上のために VBA を修正したが，全体の修正量としては，実プログラム行数で 50 行程度であった．

ActivateTask の正常系テストケースに対して作成した PictMaster ファイルの例（抜粋）を**図 7** に示す．

まず，ActivateTask のテストケースにおいて考慮すべきパラメータと，その組合せ対象値を定義する．たとえば，ActivateTask を発行する処理単位（call_origin）や，ActivateTask を発行される側のタスクの状態（other_tstat）などである．

続いて，各パラメータを組み合わせるうえでの制約表を定義する．網掛け（色付け）したセルが制約条件となり，無地のセルが制約を受ける対象となる．仕様制約 1 では，ActivateTask を発行される側のタスクの状態（other_tstat）が実行可能状態（READY）の場合，かつ ActivateTask を発行するタスクがフルプリエンプティブ（AK_SCHFULL）の場合，ActivateTask を発行される側のタスクの現在優先度（other_curpri）は高（TSK_PRI_HIGH）にはならない（“#”は否定の意）という制約を設定している．この例では，呼び出し元タスクの優先度（org_curpri）を中（TSK_PRI_MID）で固定している．ここで，org_curpri が TSK_PRI_MID であるのに対し，other_curpri が TSK_PRI_HIGH であり，かつ呼び出し先のタスクの状態（other_tstat）が READY であると，優先度の高いタスクが実行可能状態であるにもかかわらず，優先度が低いタスクが動いている状態になってしまう．さらに，A-OS 仕様では，スケジューリング方式をタスクごとに決めることが可能で，呼び出し元タスクのスケジューリング方式（org_spolicy）がフルプリエンプティブ（AK_SCHFULL）の場合にのみ，この問題が起きるので，この制約も必要となる．このように，A-OS 仕様上，発生しえない組合せを，制約表により排除している．

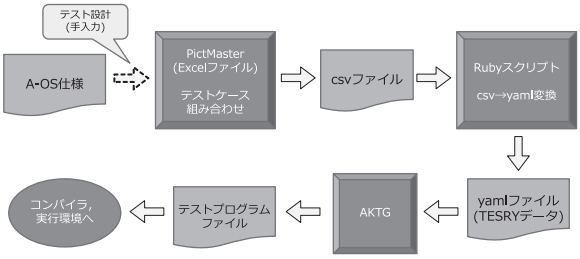


図 8 PictMaster を用いた処理フロー
Fig. 8 The process flow using the PictMaster.

そして，TESRY 生成表で，TESRY データの元データとなる csv ファイルに出力するデータを定義する．1，2 行目は，pre_condition と，TASK1 の状態（tstat）が実行状態（RUNNING）で，初期優先度が中（TSK_PRI_MID）という定義を，無条件ですべてのテストケースで生成している．3，4 行目では，TASK1 のタスク種別（ttype）が，基本タスク（AK_BASIC）か拡張タスク（AK_EXTENDED）のどちらかが定義される．同様に 5，6 行目では，TASK1 のタスクのスケジューリングポリシ（spolicy）が，フルプリエンプティブ（AK_SCHFULL）かノンプリエンプティブ（AK_SCHNON）のどちらかが定義される．これらの定義を行い，生成を行うと，制約条件に合致した，すべての組合せの TESRY データ用 csv ファイルが生成される．

この csv ファイルを，Ruby による簡単なスクリプトを用いて TESRY データへ変換し，AKTG への入力データとした．PictMaster を用いた処理フローを**図 8** に示す．

4.2 アクセス保護テスト

アクセス保護テストでは，API テストの仕組みを応用し，網羅的に，かつ効率的にアクセス保護テストを実施する手法を考案した．また，様々なプロセッサに対してアクセス保護テストを実施できるよう，プロセッサに依存した部分を明確に分離し，ポータビリティの向上を図った．本節では，アクセス保護に対するテストについて，まずテスト対象の処理とテストシナリオの例について述べ，実現方法について説明する．

4.2.1 テスト対象の処理

我々は，アクセス保護テストの対象とする処理を以下の 2 つと考えた．

(i) ユーザが設定したコンフィギュレーション情報から正しいメモリ保護情報ファイルが生成されるか（図 2 [2]）．
(ii) 生成された情報を用いて，OS による意図したアクセス保護が行われるか（図 2 [4]）．

(i) は，OS に付属するジェネレータに対するテストであ

る．ジェネレータのような入力パラメータの組合せが無数に存在し，包括的な検証が困難なツールに対する検証は，出力結果のレビューやバリデーションツールによって正当性，妥当性を担保することが現実的な解決手法といえる．そこで，(i) に対するテストは，レビューで行うものとし，表 1 に示したすべてのメモリ領域がジェネレータの生成情報に含まれるコンフィギュレーション情報ファイルを使用して，生成されたメモリ保護情報ファイルが正しく設定されることをレビューした．また，API テストでは，テストケースごとのコンフィギュレーションに応じたファイルが，ジェネレータによって生成され，ビルド・実行されるため，間接的にジェネレータのテストは実施されていると考えられる．なお，ジェネレータが生成したコードに対する検証手法や十分性に関しては，本研究の対象とはせず，今後の研究対象とする．

(ii) は，2.1.2 項で述べたアクセス保護に対するテストであり，我々はこれをアクセス保護テストの対象とした．

4.2.2 テストシナリオの例

前項での述べたアクセス保護テストのテストシナリオの一例を**図 9** (a) に示す．

まず，前状態から，後状態 1 において，信頼 OSAP1 から別の非信頼 OSAP2 へ処理単位を切り替える．これにより，CPU コアは特権モードからユーザモードに切り替わり，MPU は非信頼 OSAP2 のアクセス可能なメモリ領域が設定される．つまり，後状態 1 で，OS によって非信頼 OSAP2 に与えられた正しいアクセス権となるように，CPU コアと MPU が設定されているはずである．

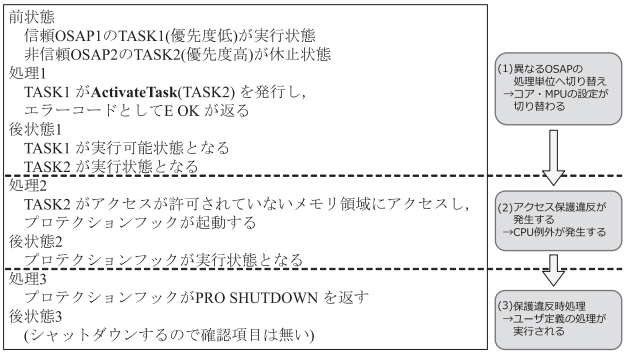
続いて，MPU の設定が正しいことをテストするために，処理 2 で，実行中の非信頼 OSAP2 のタスクから，非信頼 OSAP2 がアクセスを許可されていないメモリ領域へのアクセスを行う．MPU によりアクセス違反が検知され，CPU 例外が発生し，OS がプロテクションフックを起動する．

続いて，プロテクションフックから，OS をシャットダウンする指示である PRO_SHUTDOWN を返すことにより，OS はシャットダウンする．

このように，アクセス保護テストは，(1) 処理単位の切り替わり，(2) メモリアクセス，(3) 保護違反時処理/正常処理という流れで行うことにした．(1) は，処理単位の切替えパターンとして，タスクディスパッチや割込み発生時など CPU コアと MPU の設定変更が発生するすべての処理単位が切り替わる組合せを確認する．(2) は，メモリアクセスパターンとして，アクセスが禁止されたメモリ領域へアクセスした場合に，アクセス保護違反が検知されることに加え，アクセスが許可されたメモリ領域へアクセスした場合に正しくアクセスできることも確認する．これを，表 1 に示したすべてのメモリ領域と，アクセス保護属性なしのメモリ領域[*3]に対して確認する．(3) は，保護違反時処理パターンとして，アクセス保護違反が発生した場合に，ユーザが設定したとおりの保護違反時処理が実行されることを確認する．アクセス保護違反が発生しないテストケースの場合，メモリアクセスを行った処理単位の実行が継続することを確認する．

[*3] プロセッサによってはアクセス時の振舞いが不定となるので，テストできない可能性がある．

(a)



(b)

```
pre_condition:
  TASK1:
    type    : TASK
    tstat   : RUNNING
    curpri  : TSK_PRI_LOW
    osap    : TRUSTED_OSAP1
  TASK2:
    type    : TASK
    tstat   : SUSPENDED
    curpri  : TSK_PRI_HIGH
    osap    : NON_TRUSTED_OSAP2
  PROTECTION_HOOK1:
    type    : PROTECTION_HOOK
    hookstat: AK_STOP
do_1:
  id      : TASK1
  syssrv : ActivateTask(TASK2)
  rettype: StatusType
  retval : E_OK
post_condition_1:
  TASK1:
    tstat: READY
  TASK2:
    tstat: RUNNING
do_2:
  id    : TASK2
  syssrv: <アクセス保護違反を発生させる関数>
post_condition_2:
  PROTECTION_HOOK1:
    hookstat: AK_ACTIVATE
do_3:
  id    : PROTECTION_HOOK1
  syssrv: return(PRO_SHUTDOWN)
post_condition_3:
```

図 9 アクセス保護テストのテストシナリオと TESRY 記法の例
Fig. 9 An example of the test scenario and the TESRY notation for access protection test.

4.2.3 各パターンの組合せ

処理単位の切替えパターンを，**図 10** に示す．

パターン 1，パターン 6 は，それぞれ信頼 OSAP，OS 管理の処理単位から非信頼 OSAP1 の処理単位に切り替わるパターンである．これは，CPU コアが特権モードで動作している状態から，ユーザモードへ切り替わるのに加え，MPU が非信頼 OSAP1 のアクセス権の状態へ切り替わる．

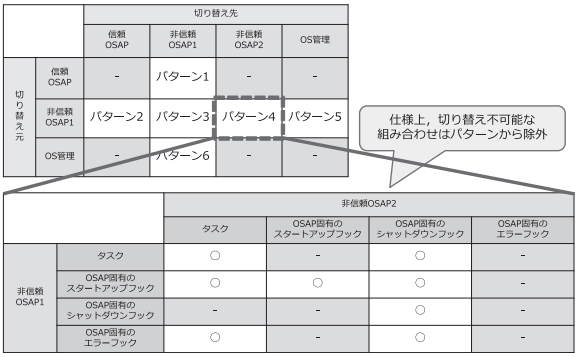


図 10 処理単位の切替えパターン
Fig. 10 The pattern of process unit switch.

パターン 2，パターン 5 は，それぞれ非信頼 OSAP1 の処理単位から，信頼 OSAP，OS 管理の処理単位に切り替わるパターンである．これは，CPU コアがユーザモードで動作している状態から，特権モードへ切り替わる．パターン 3 は，同じ非信頼 OSAP1 に所属する処理単位でも，異なるタスクの場合に，保護するスタック領域が切り替わるパターンである．これは，MPU のスタック領域に関する設定だけ切り替わる．パターン 4 は，非信頼 OSAP1 の処理単位から異なる非信頼 OSAP2 の処理単位に切り替わるパターンである．これは，MPU の非信頼 OSAP ごとの設定が切り替わる．

さらに，各パターンにおいて，切り替わる処理単位の組合せを考慮する．図 10 で拡大したパターン 4 を例に説明する．非信頼 OSAP には，タスク，OSAP 固有のスタートアップフック，シャットダウンフック，エラーフックが所属できる．ただし，各処理単位が動作している状態で，異なる非信頼 OSAP の処理単位へ遷移する可能性があるかどうかは，A-OS 仕様から判断できるため，仕様上，切替え不可能な組合せはパターンから除外することができる．パターン 4 では，図 10 に示した 8 つの処理単位の切替えがテスト対象となる．

テストに使用したプロセッサにおけるメモリアクセスパターンを，**図 11** に示す．ROM 領域は 8 種類，RAM 領域は 19 種類のメモリ領域がある．RAM 領域には，スモールデータや書き込み可否，スタックなどの種別が増えるため，ROM 領域より多くの領域が存在する．ほかに，メモリ保護の設定がされていない領域があるが，テストに使用したプロセッサでは，メモリ保護の設定ができない領域であり，アクセス時の振舞いが不定であるので，テスト対象外とした．

信頼 OSAP と OS 管理の処理単位は，特権モードで動作するため，アクセスできないメモリ領域は存在しない．また，図 10 で想定した処理単位から区別されるメモリ領域は以下の 6 つである．

(A) すべての OSAP，処理単位からアクセスできるメモリ領域

メモリ領域の種類		アクション番号				
		読出/先頭	読出/終端	書込/先頭	書込/終端	実行
ROM	OSの専用コード領域	0	1	2	3	4
	OSの専有リードオンリーデータ領域	5	3	7	8	9
	非信頼OSAP1の専有コード領域	0	1	10	11	2
	非信頼OSAP1の専有リードオンリー領域	3	4	12	13	14
	非信頼OSAP2の専有コード領域	0	1	15	16	2
	非信頼OSAP2の専有リードオンリー領域	3	4	17	18	19
	共有コード領域	0	1	20	21	2
	共有リードオンリーデータ領域	3	4	22	23	24
RAM	OSの専有リードライト領域（スモールデータ）	25	26	27	28	29
	非信頼OSAP1の専有リードライトデータ領域(スモールデータ)	5	6	7	8	30
	非信頼OSAP2の専有リードライトデータ領域(スモールデータ)	5	6	7	8	31
	共有リード専有ライトデータ(スモールデータ)(共有リード領域)	5	6	32	33	34
	共有リード専有ライトデータ(スモールデータ)(非OSAP1の専有ライト)	7	8	9	10	35
	共有リード専有ライトデータ(スモールデータ)(非OSAP2の専有ライト)	9	10	9	10	36
	共有リードライトデータ(スモールデータ)	11	12	13	14	37
	OSの専有リードライト領域	38	39	40	41	42
	OSの専有リードライト領域(スタック)	43	44	45	46	47
	非信頼OSAP1の専有リードライトデータ領域	11	12	13	14	48
	非信頼OSAP2の専有リードライトデータ領域	11	12	13	14	49
	共有リード専有ライトデータ領域(共有リード領域)	15	16	50	51	52
	共有リード専有ライトデータ領域(非OSAP1の専有ライト)	17	18	15	16	53
	共有リード専有ライトデータ領域(非OSAP2の専有ライト)	19	20	15	16	54
	共有リードライトデータ領域	21	22	23	24	55
	非信頼OSAP1のタスク1スタック	0 0	1 1	2 2	3 3	4
	非信頼OSAP1のタスク2スタック	0 0	1 1	2 2	3 3	4
	非信頼OSAP2のタスク1スタック	0 0	1 1	2 2	3 3	4
	非信頼OSAPのフックのスタック	0 0	1 1	2 2	3 3	4
その他	メモリ保護の設定がされていない領域	×	×	×	×	×

すべてのOSAP，処理単位からアクセスできるメモリ領域へのアクセス
非信頼OSAP1からのみアクセスできないメモリ領域へのアクセス
非信頼OSAP2からのみアクセスできないメモリ領域へのアクセス
全ての非信頼OSAPからアクセスできないメモリ領域へのアクセス
対象スタックを使用する処理単位からアクセスできるスタック領域へのアクセス
対象スタックを使用する処理単位からアクセスできないスタック領域へのアクセス
アクセス保護属性なしのメモリ領域

図 11 メモリアクセスパターン（抜粋）
Fig. 11 The pattern of memory access.

(B) 非信頼 OSAP1 からのみアクセスできないメモリ領域
(C) 非信頼 OSAP2 からのみアクセスできないメモリ領域
(D) すべての非信頼 OSAP からアクセスできないメモリ領域
(E) 対象スタックを使用する処理単位からアクセスできるスタック領域へのアクセス
(F) 対象スタックを使用する処理単位からアクセスできないスタック領域へのアクセス

つまり，これら 6 つのメモリ領域に分割して，各処理単位からアクセスすることで，すべての組合せを網羅することができる．テストに使用したプロセッサでは，読み出し・書き込み・実行のアクセス保護属性を設定可能であったので，各メモリ領域に対しては，読み出しの先頭アドレス，終端アドレス，書き込みの先頭アドレス，終端アドレス，先頭アドレスの実行の 5 パターンでアクセスを行うことにした．

図 11 に示したとおり，各メモリ領域に定義可能なメモリ領域の種類が，先にあげた 6 つのメモリ領域のどれに該当するかを色分けし，メモリ領域ごとに 0 から連番で番号を付与する．この番号をアクション番号と呼ぶ．テストプログラムにおけるメモリ領域へのアクセスは，6 つのメモリ領域ごとに，アクセス用の関数（以下，メモリアクセス関数）を用意しておき，メモリアクセス関数の引数にアクション番号を与える実装とする．ただし，(E)，(F) のスタック領域に関しては，アクセス対象のスタックごとに，アクセスするメモリ領域を切り替えるため，アクション番号はそれぞれの領域に対して 0 から連番となる．

ATK2 では，保護違反時処理パターンは，以下の 5 つと

なる．

(I) プロテクションフックの戻り値がPRO_SHUTDOWN（OSをシャットダウンする）

(II) プロテクションフックの戻り値がPRO_TERMINATETASKISR（アクセス保護違反を引き起こしたタスクを強制終了する）

(III) プロテクションフックの戻り値がPRO_IGNORE（アクセス保護違反は無視できないため，シャットダウンされる）

(IV) プロテクションフックの戻り値が規定外（規定外の戻り値の場合，シャットダウンされる）

(V) プロテクションフックなし（アクセス保護違反発生時点で，OSをシャットダウンする）

アクセス保護違反が発生しないテストケースでは(V)のみとする．

4.2.4 AKTG，PictMasterの使用

アクセス保護テストは，各パターンの組合せにより，テストケース数が膨大となるので，APIテスト同様，AKTGによるテストプログラムの生成と，PictMasterによるテストケースの生成を用いた効率化が求められる．

TESRY記法は，前状態，処理，後状態の組合せだけでなく，後続する処理，後状態を続けて記述することができる[16]．したがって，図9に示したようなアクセス保護テストのテストシナリオを，TESRY記法で記述すれば，AKTGによってテストプログラムを生成することが可能である．図9(a)のテストシナリオを，TESRY記法で記述した例を図9(b)に示す．

前項で述べた，処理単位の切替えパターンと保護違反時処理パターンについては，PictMasterにより，容易に組み合わせが可能であり，メモリアクセスパターンを除いたパターンを組み合わせたTESRYデータは，APIテストと同様の手法で生成することができた．このTESRYデータは，プロセッサに依存しない．

4.2.5 プロセッサに依存する部分の明確化

メモリアクセスパターンは，プロセッサやMPUによって定義できるメモリ領域の種類が異なり，読み出し・書き込み・実行の区別がつかない可能性もある．つまり，アクション番号の上限値がプロセッサごとに異なる．したがって，前項で述べたプロセッサに依存しないTESRYデータに対して，何らかの方法で，プロセッサに依存するメモリアクセスパターンを組み合わせる必要がある．

そこで，まず前項で述べたPictMasterによる組み合わせの段階では，プロセッサに依存しないTESRYデータの基本パターンのみ生成しておく．具体的には，各メモリアクセス関数の引数を「xxx」といった識別子にしておき，アクション番号による組合せは行わない．テスト対象のプロセッサが確定し，メモリアクセスパターンが確定した段階で，メモリアクセス関数ごとの上限値を設定したシェルスクリプトによって，基本パターンのTESRYデータを，必要な数分のTESRYデータへ複製し，メモリアクセス関数の引数を数値に変換することにした．

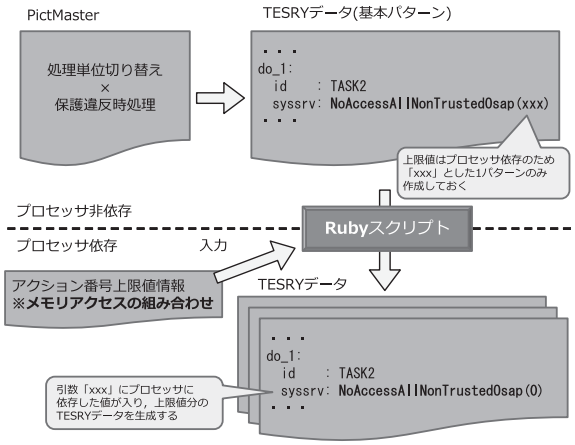


図 12 アクセス保護テスト設計のデータフロー

Fig. 12 The data flow of design for access protection test.

アクセス保護テスト設計のデータフローを**図 12** に示す．この設計により，新たなプロセッサに対してアクセス保護テストを実施する際は，メモリアクセスパターンを，プロセッサに応じて作成し，シェルスクリプトにアクション番号上限値情報を入力するのみで対応できる．なお，メモリアクセス関数も，引数に応じたアクセス処理を行う必要があるため，プロセッサごとに実装が必要となる．

5. 評価

我々は，4章で述べた手法を用いて，ATK2に対してテストを実施した．ATK2の開発，テストには，Terasic社のAltera DE2-115 Development and Education Board（以下，評価ボード）を使用した．評価ボードには，Altera社のFPGA向けのNios2プロセッサが搭載されている．Nios2のMPUでは，コード領域用の領域を32個，データ領域用の領域を32個と使用でき，読み出し，書き込み，実行のすべてを区別してアクセス制御できる．

本章では，AKTSPを用いて，ATK2をテストした結果について評価する．**表 2** に，テスト実施結果の一覧を示す．TTGでは，TESRYデータの行数に対するテストプログラムの行数が約3.7倍であったのに対し，AKTGでは約2倍となっているのは，4.1.2項で述べたコードサイズ削減の影響である．なお，TTG，AKTGともにRubyで開発し，TTGの総行数は19,328行であったのに対し，AKTGは39,182行であった．コード増加の理由は，メモリ保護機能への対応である．

5.1 APIテスト

5.1.1 テストケース数

まず，A-OS仕様のSCの中で，メモリ保護機能に対応していないスケーラビリティクラス1（SC1）に対するAPI

表 2 テスト実施結果一覧
Table 2 Results of tests.

		ASP	SC1	SC3
OS 仕様				
	API 数	103	35	57
	メモリ保護機能	なし	なし	あり
API テスト				
	PictMaster シート数	-	112	223
	テストケース数	1,675*	7,334#	40,633#
	TESRY データ行数	60,648*	431,834#	2,837,186#
	テストプログラム行数	224,719#	949,757#	5,884,643#
アクセス保護テスト				
	PictMaster シート数	-	-	185
	テストケース数	-	-	43,244#
	TESRY データ行数	-	-	5,715,216#
	テストプログラム行数	-	-	12,731,538#

*:手動で作成，#:ツールによる生成

テストと，ASP に対する API テストについて比較を行う．API の数は，ASP に比べて SC1 には 3 分の 1 程度しか存在しないが，テストケース数は約 4.4 倍となっている．4.1.3 項で述べたとおり，A-OS 仕様は組み合わせるパラメータが多いことで，メモリ保護機能がない場合でも，μITRON 仕様に比べてテストケース数が大幅に増加することが確認できた．

さらに，SC1 と SC3 では，API の数は約 1.6 倍であるが，テストケース数は約 5.5 倍になっている．これは，メモリ保護機能があることにより，タスクなどのオブジェクトに信頼，非信頼の組合せが増えることが原因である．メモリ保護機能への対応により，追加が必要となるテストの規模を明らかにすることができた．

5.1.2 実施工数

PictMaster の Execl ファイルは，API ごとに作成したが，各ファイルには制約表や TESRY 生成表の可読性を向上させるために，正常系，異常系などのカテゴリごとに，別シートで作成した．1 つのシート作成は，レビューを含めても 1 人日程度であるので，SC1 全体で約 5.6 人月，SC3 全体で約 11 人月規模の作業である．一方，SC1 と SC3 で合計約 327 万行規模の TESRY データを人手で開発，保守することは現実的でない．1 人日で 50 件の TESRY データを作成できたとしても，SC1 と SC3 のすべての TESRY データを作成するには，約 48 人月要するため，API テストの実施工数を大幅に削減できたといえる．

5.1.3 組合せ結果の正当性

PictMaster を用いて，意図した組合せが生成されていることは，レビューで確認した．csv ファイルは，Excel で開くことにより，行のフィルタ機能や並べ替えが使用可能となるので，組合せが網羅されているか，容易に確認することができる．また，AKTG によって生成したテストプログラムを実行した結果として，各 API の実行回数とソースコードカバレッジが想定どおりの結果となることを確認した．

5.1.4 品質の向上

本研究の目的の 1 つは，メモリ保護機能を持つ RTOS の品質を向上させることである．本手法を用いることにより，特定の処理単位から呼び出した場合にのみ発生する不具合を検出できた．これにより，ATK2 の品質向上に貢献し，この目的を達成したと考えられる．人手で実施した場合，テストケース削減のために実施されない可能性もあるため，PictMaster により，すべての処理単位から呼び出すように組み合わせた効果があったといえる．なお，API テストにより，検出した ATK2 の API に関する不具合は 7 件であった．

5.2 アクセス保護テスト

5.2.1 実施工数

PictMaster の Execl ファイルは，図 10 に示した 6 つのパターンごとに作成したが，プロテクションフックの有無などによって 1 つのテストプログラムとして実行できないテストケースの組合せを分け，別シートで作成した．API テストと異なり，アクセス保護テストの PictMaster ファイルは，基本構成が同じであるので，1 つのシート作成は，レビューを含めても 0.3 人日程度であり，全体で約 2.8 人月規模の作業である．API テスト同様，アクセス保護テストも実施工数を大幅に削減できたといえる．

5.2.2 品質の向上

アクセス保護テストにおいても，特定の処理単位切替えの組合せのみで発生する不具合を検出できたことから，メモリ保護機能を持つ RTOS の品質向上を達成したと考えられる．不具合の中には，特定の処理単位切替えの組合せのみ検出されるものも存在したため，API テスト同様，網羅的な組合せの効果があったといえる．なお，アクセス保護テストにより，検出した ATK2 のアクセス保護に関する不具合は 5 件であった．

また，この 5 件とは別に，ルネサスエレクトロニクス社の RH850/F1L など Nios2 以外のプロセッサへ ATK2 をポーティングした際に，ATKSP のアクセス保護テストにより不具合を検出した事例が 7 件報告されている．主に以下の 3 つのカテゴリに分類される不具合である．

- 処理単位切替え時のスタックポインタ設定ミス
- MPU に設定する情報のミス
- プロセッサ，コンパイラが持つ制約の誤解

いずれも，プロセッサ，コンパイラに依存した不具合である．アクセス保護のポーティングは，プロセッサやコンパイラの仕様に依存する要素が多く，難易度も高いため，不具合を生みやすい．ポーティング後に，アクセス保護に対して網羅的にテストを実施することに有用性があるといえる．

5.3 A-OS 仕様への対応

2.2.1 項で述べたとおり，ATK2 には，実装していない A-OS 仕様が存在する．これらの仕様を実装した場合に，提案したテスト手法を適用する方法について考察する．

非信頼 OSAP に ISR が所属する場合，処理単位の切替えパターンに，非信頼 OSAP に所属する ISR を追加するだけよい．OSAP の強制終了，再起動に対応する場合，保護違反時処理パターンに，PRO_TERMINATEAPPL と PRO_TERMINATEAPPL_RESTART を戻り値とする組合せを追加するだけでよい．

以上から，A-OS 仕様への対応は，各パターンの組合せを追加するのみでよく，容易に実施可能であると考えられる．加えて，AKTG に，非信頼 OSAP に所属する ISR に対するプログラム生成処理を追加する必要があるが，技術的な課題はない．

6. 関連研究

RTOS のテスト手法に関する研究はいくつか存在する [7], [8], [9], [10], [11], [19]．文献 [19] では，RTOS に適用可能なソフトウェアテスト手法について述べているが，具体的なテストプログラムの実装には言及していない．文献 [7], [8], [9], [10], [11] は，モデル検査によりテストケースの組合せ生成を行っているが，本研究が対象とするメモリ保護機能に対するテストケースは，組み合わせツールの PictMaster で生成可能であり，モデル検査の必要性は低いと考えている．また，一般的なソフトウェアに対する研究として，テストケースの組合せ手法や，テストプログラムを生成するツールに関するいくつかの研究が存在する [20], [21], [22], [23], [24]．いずれの研究も，特定の機能に特化しているものや，メモリ保護機能を持つ RTOS に対するテストプログラムの生成には対応していないものであり，本研究には適用できない．

メモリ保護機能に対応した RTOS のテストに関しては，μITRON 仕様のメモリ保護機能拡張（PX 仕様）に対応した TOPPERS/HRP を対象とした，高信頼化技術のハンドブックが提案されている [25]．しかし，メモリ保護機能に対して，具体的にどのような手法でテストを実施するかまでは言及されていない．

メモリ保護機能に対するテストとしては，Linux カーネルの W⊕X プロテクション機能に対してモデル検査を行う研究がある [26]．しかし，モデル検査により，W⊕X プロテクション機能の解析，問題抽出を行っているのみで，テストケースやテストプログラムの生成には言及していない．

7. まとめ

本研究では，メモリ保護機能に対応した RTOS である ATK2 に対するテスト手法を確立し，API テスト，およびアクセス保護テストを実施した．これにより，ATK2 の不具合を 19 件検出し，ATK2 の品質向上に貢献した．

国内の AUTOSAR のニーズの高まりを受け，ATK2 が利用されるケースも増加している．我々は，本テスト手法の確立以降，様々なプロセッサに ATK2 をポーティングし，AKTSP を使用しており，アクセス保護テストの有用性を実感している．

今後の取り組みとしては，ジェネレータが生成したコードに対する検証手法に関する研究を予定している．また，メモリ保護機能に対する要求仕様が漏れなくテストされているかを確認するために，要求仕様とテストケースのトレーサビリティの確認も検討している．

謝辞 本研究の推進に協力してくださった，ATK2 コンソーシアムの皆様に謹んで感謝の意を表する．

参考文献

[1] AUTOSAR (online), available from ⟨http://www.autosar.org/⟩ (accessed 2015-10-24).
[2] Specification of Operating System (online), available from ⟨http://www.autosar.org/fileadmin/files/releases/4-0/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf⟩ (accessed 2015-10-24).
[3] TOPPERS/ATK2 (online), available from ⟨http://www.toppers.jp/atk2.html⟩ (accessed 2015-10-24).
[4] 坂村 健（監修），高田広章（編）：μITRON4.0 仕様 Ver.4.02.00，トロン協会 (2004).
[5] TOPPERS/HRP2 カーネル（オンライン），入手先 ⟨http://www.toppers.jp/hrp2-kernel.html⟩（参照 2015-10-24）.
[6] 石川拓也，本田晋也，高田広章：静的なメモリ配置を行うメモリ保護機能を持ったリアルタイム OS，コンピュータ ソフトウェア，Vol.29, No.4, pp.161–181 (2012).
[7] Chen, J. and Aoki, T.: Conformance Testing for OSEK/VDX Operating System Using Model Checking, *Software Engineering Conference* (*APSEC*), pp.274–281 (2011).
[8] Choi, Y. and Byun, T.: Constraint-based test generation for automotive operating systems, *Software & Systems Modeling*, pp.1–18 (2015).
[9] Fang, L., Kitamura, T., Ngoc Do, T.B. and Ohsaki, H.: Formal Model-Based Test for AUTOSAR Multi-core RTOS, Software Testing, *Verification and Validation* (*ICST*), pp.251–259 (2012).
[10] 青木利晃，山崎真吾：モデル検査によるリアルタイムオペレーティングシステムの設計検証，組込みシステムシンポジウム 2008 論文集，pp.159–166 (2008).
[11] 加藤 淳，神武直彦，春山真一郎，狼 嘉彰：モデル検査を用いて組込みシステムにおけるソフトウェアとハードウェアの協調動作に関する要求仕様の不整合を検出する手法，組込みシステムシンポジウム 2009 論文集，pp.65–70 (2009).
[12] TOPPERS/TTSP (online), available from ⟨http://www.toppers.jp/ttsp.html⟩ (accessed 2015-10-24).
[13] TOPPERS/ASP カーネル（オンライン），入手先 ⟨http://www.toppers.jp/asp-kernel.html⟩（参照 2015-10-24）.
[14] TOPPERS/FMP カーネル（オンライン），入手先 ⟨http://www.toppers.jp/fmp-kernel.html⟩（参照 2015-10-24）.
[15] 鴫原一人，一場利幸，本田晋也，高田広章：μITRON ベースのマルチプロセッサ向け RTOS のテスト，情報処理学

会論文誌，Vol.53, No.12, pp.2682–2701 (2012).
[16] 鴫原一人，眞弓友宏，森　孝夫，本田晋也，高田広章：μITRON ベースの RTOS 向けテストプログラム生成ツール，電子情報通信学会論文誌 D，Vol.J95-D, No.4, pp.870–884 (2012).
[17] PictMaster (online), available from ⟨https://osdn.jp/projects/pictmaster/⟩ (accessed 2015-10-24).
[18] 風間佳之，平橋　航，鴫原一人，海上智昭，本田晋也，高田広章：AUTOSAR OS に対するテストケースおよびテストプログラムの自動生成，ソフトウェアテストシンポジウム 2012 予稿集，pp.17–24 (2012).
[19] Tsoukarellas, M.A., Gerogiannis, V.C. and Economides, K.D.: Systematically testing a real-time operating system, *Micro*, pp.50–60, IEEE (1995).
[20] 丹野治門，張　暁晶，星野　隆：結合テストにおけるテスト項目自動生成手法の提案と評価，電子情報通信学会技術研究報告，Vol.110, No.227, pp.37–42 (2010).
[21] 土屋達弘，菊野　亨：ペアワイズテスト：ソフトウェアテストの効率化を求めて，電子情報通信学会論文誌 D，Vol.90, No.10, pp.2663–2674 (2007).
[22] 秋山浩一：ソフトウェアテストの最新動向：3. 組合せテストの設計，情報処理学会誌，Vol.49, No.2, pp.140–146 (2008).
[23] Belinfante, A., Feenstra, J., de Vries, R., Tretmans, J., Goga, N., Feijs, L. and Mauw, S.: Formal test automation: a simple experiment, *International Workshop on the Testing of Communication Systems* (*IWTCS'99*), pp.179–196 (1999).
[24] Jeron, T. and Morel, P.: Test generation derived from model-checking, *Computer Aided Verification* (*CAV'99*), Vol.1633 of LNCS, pp.108–122 (1999).
[25] 佐藤伸子，石濱直樹，川崎朋実，片平真史：宇宙機搭載用リアルタイム OS に適用した高信頼化技術のハンドブック化，組込みシステムシンポジウム 2011 論文集，pp.25-1–25-7 (2011).
[26] Liakh, S., Grace, M. and Jiang, X.: Analyzing and improving Linux kernel memory protection: A model checking approach, *ACSAC '10, Proc. 26th Annual Computer Security Applications Conference*, pp.271–280 (2010).

鴫原 一人

2003 年武蔵工業大学（現，東京都市大学）工学部電子通信工学科卒業．同年首都圏松下テクニカルサービス（現，パナソニック コンシューマーマーケティング）(株) 入社．2007 年富士ソフト（株）入社．2010 年から 2014 年ま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に出向し，マルチプロセッサ向けリアルタイム OS，AUTOSAR の研究に従事．2015 年から APTJ（株）に出向．名古屋大学大学院情報科学研究科博士後期課程に在学．

石川 拓也

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特任助教．2011 年名古屋大学大学院情報科学研究科博士課程前期課程修了．2013 年同博士課程後期課程修了．同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研究員を経て，2015 年より現職．リアルタイムスケジューリング理論，リアルタイム OS，組込みシステム向けコンポーネント技術の研究に従事．博士（情報科学）．

本田 晋也　（正会員）

2002 年豊橋技術科学大学大学院情報工学専攻修士課程修了．2005 年同大学院電子・情報工学専攻博士課程修了．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助教等を経て，2014 年より名古屋大学大学院情報科学研究科情報システム学専攻准教授，リアルタイム OS，ソフトウェア・ハードウェアコデザインの研究に従事．博士（工学）．2002 年度情報処理学会論文賞受賞．ACM，IEEE，電子情報通信学会，日本ソフトウェア科学会各会員．

高田 広章　（正会員）

名古屋大学未来社会創造機構教授．同大学大学院情報科学研究科教授・附属組込みシステム研究センター長を兼務．1988 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同専攻助手，豊橋技術科学大学情報工学系助教授等を経て，2003 年より名古屋大学大学院情報科学研究科情報システム学専攻教授．2014 年より現職．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術等の研究に従事．オープンソースのリアルタイム OS 等を開発する TOPPERS プロジェクトを主宰．博士（理学）．IEEE，ACM，電子情報通信学会，日本ソフトウェア科学会，自動車技術会各会員．