Vol.2014-SLDM-165 No.38
Vol.2014-EMB-32 No.38
2014/3/16

リアルタイムOSに対するMC/DC計測手法

廉 正烈[†1]　鴫原 一人[†1]　海上 智昭[†1]　本田 晋也[†1]　高田 広章[†1]

概要：
近年ソフトウェアに対する安全基準対応への要求が高まっている．ソフトウェアテストのメトリクスとして MC/DC があるが，リアルタイム OS に対する MC/DC 計測実施事例は公開されていない．また，我々が開発した AUTOSAR OS 仕様ベースのリアルタイム OS である TOPPERS/ATK2 では，メモリ保護機能やマルチコアに対応しているため，実行環境やカバレッジ取得方法に対する制約が強く，MC/DC 計測は困難であると推測される．そこで我々は，TOPPERS/ATK2 の MC/DC 計測に対する要件をまとめ，市販の MC/DC 計測ツールを使用した各要件の実現方法を考案し，TOPPERS/ATK2 に対する MC/DC を計測する手法を確立した．本手法により，TOPPERS/ATK2 を使用して開発した車載システム向けのソフトウェアに対しても MC/DC 計測が可能となる．本論文では，TOPPERS/ATK2 を用いたリアルタイム OS に対する MC/DC 計測手法について述べる．

キーワード：MC/DC，テスト，リアルタイム OS，AUTOSAR，組込みシステム，ツール

MC/DC measurement method for Real-time OS

Abstract:
The present paper reports a new MC/DC measurement method for Real-time OS. Recently, adherence to software safety requirements is increasingly becoming more important. Although MC/DC metrics are widely used in software testing, no record of MC/DC measurement on Real-time OS was found. Furthermore, as for TOPPERS/ATK2, a RTOS developed by the authors based on AUTOSAR specifications, measuring MC/DC was assumed to be hard due to the strict restrictions on coverage retrieval method and execution environment, since it supports memory protection function and multi-core. Therefore, the authors first summarized the requirements for measuring MC/DC of TOPPERS/ATK2, then considered several methodologies to fulfill the requirements by using commercial MC/DC measurement tools, and established a new MC/DC measurement method for TOPPERS/ATK2. The new method enables MC/DC measurement of TOPPERS/ATK2 based software for automotive systems.

Keywords: MC/DC, Test, Real-time OS, AUTOSAR, Embedded system, tools

1. はじめに

リアルタイム OS(以後，RTOS) は，ソフトウェアの中核を成すため，高い品質が求められる．ソフトウェアの品質を向上させるためには，ソフトウェアテストが不可欠であり，ソフトウェアテストの品質を評価する有効なメトリクスの 1 つとしてコードカバレッジがある．現在，自動車産業で適用が推められている自動車用機能安全規格 ISO26262 においては，コードカバレッジの基準として MC/ DC(Modified Condition/Decision Coverage)[1] 計測の要求事項が規定されている．

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 (NCES)[*1]は，車載システム向けプラットフォームの AUTOSAR OS 仕様[*2]に準拠した RTOS である TOPPERS/ATK2(AuTomotive Kernel version 2)[*3](以後，ATK2) を開発している．また，ATK2 を検証するのためのテストスイートである AKTSP(Automotive Kernel Test Suit Package) も開発している．AKTSP によるテス

[†1] 現在，名古屋大学
Presently with Nagoya University

[*1] http://www.nces.is.nagoya-u.ac.jp/
[*2] http://www.autosar.org/
[*3] http://www.toppers.jp/atk2.html

ト品質のメトリクスとしても，MC/DC を計測するべきであるが，RTOS は，ターゲットシステム上でしか実行できない場合があることや，テストプログラムが使用できるメモリサイズの制約といった，他のソフトウェアと比べて MC/DC 計測にいくつかの問題が存在する．ATK2 は，メモリ保護機能やマルチコアに対応しているが，これらの機能を持つ RTOS に対する MC/DC 計測は，さらに困難であり，実施事例も公開されていない．

そこで我々は，市販の MC/DC 計測ツールを調査し，メモリ保護機能とマルチコアに対応した RTOS に対する MC/DC を取得する手法を確立した．ターゲットシステム上で実行する RTOS から，ホスト PC 側でカバレッジ情報を取得する方法や，複数のテストプログラムによって取得したカバレッジ情報のマージ，テスト対象の処理を実行している間のみカバレッジを取得する方法を考案し，AKTSP を用いたテストによって，MC/DC 計測が可能であることを確認した．

本論文では，ATK2 を題材にして確立した，RTOS に対する MC/DC 計測手法について述べる．

2. RTOS に対する MC/DC 計測の要件

我々は，RTOS に対して MC/DC 計測を行う上での要件を検討した結果，以下の 6 要件をまとめた．本章では，各要件について述べる．

(1) カバレッジ情報の取得方式
(2) カバレッジ情報の保存方法
(3) カバレッジ情報のマージ
(4) カバレッジ取得状態の切り替え
(5) コマンドラインからの実行
(6) メモリ保護機能への対応

2.1 要件 (1) カバレッジ情報の取得方式

一般に，カバレッジ計測を行うには，以下の 2 つの方法がある．MC/DC 計測が可能な市販ツールにおいても，この 2 つの方式に分類される．

(i) プログラムをシミュレータ上で実行し，アセンブラコードレベルで通ったパスや判定結果を取得する
(ii) 対象とするソースコードに，カバレッジ取得用のコードを埋め込んで，カバレッジ情報を取得する

(i) の方式は，テスト対象のソースコードを変更しないというメリットがあるが，テスト対象のプログラムを実行可能なシミュレータが必要になる．我々が対象とする RTOS である ATK2 は，メモリ保護機能とマルチコアに対応しているが，これらの機能を搭載したシミュレータは我々が知る限り存在しない．したがって，ATK2 のカバレッジを取得するためには，(ii) の方法を取らざるを得ない．

2.2 要件 (2) カバレッジ情報の保存方法

PC 上で実行するソフトウェアであれば，カバレッジ情報の取得方法には関係なく，一般的に PC 上にカバレッジ情報を保存することが可能である．しかし，シミュレータが存在しない場合，RTOS はターゲットシステム上で実行せざるを得ない．メモリ制約の強い組込みシステムでは，ターゲットシステムでカバレッジ情報を保存できるケースは稀なため，ホスト PC へカバレッジ情報を転送して保存するといった手段が必要となる．

2.3 要件 (3) カバレッジ情報のマージ

RTOS のテストは，OS の状態や各オブジェクトの状態 (以後，システム状態) の組み合わせなどにより，膨大なテストケースを実施する必要がある．しかし，RTOS を実行するターゲットシステムでは，ROM/RAM といったメモリのサイズに制約があり，大きなテストプログラムを一度に実行することはできない．

したがって，複数のテストプログラムに分割して，テストを実施する必要があり，この場合，カバレッジ情報も複数に分割された状態で取得することになる．すべてのテストに対するカバレッジを確認するには，分割して取得したカバレッジ情報のマージが必要となる．

2.4 要件 (4) カバレッジ取得状態の切り替え

RTOS のソースコードは，システムサービスを実装した部分が大半であるので，カバレッジを確認するテストは，システムサービスの振る舞いを確認するテスト (以後，システムサービステスト) によって行うべきである．システムサービステストは，あるシステム状態において，テスト対象のシステムサービスを呼び出すことによるシステム状態の変化が，RTOS の外部仕様書 (以後，仕様書) に規定された通りであるかを確認する．例えば，低優先度のタスク (TASK1) から，タスクを起動するシステムサービスである ActivateTask を，高優先度のタスク (TASK2) に対して発行した場合，TASK1 はプリエンプトされ実行可能状態となり，TASK2 が実行状態へと遷移する，といった変化を確認する．

システムサービステストでは，テスト対象のシステムサービス呼出しの前に，意図したシステム状態を実現するためにも，様々なシステムサービスの呼出しが必要になるが，カバレッジ計測においては，テスト対象のシステムサービス呼出しに対するカバレッジのみを取得するべきである．これは，仕様書で規定された振る舞いを元に作成したテストケースによってパスしたカバレッジのみを取得することで，偶然的にパスしてしまうカバレッジを除外するためである．

また，カバレッジ取得用コードを埋め込んだプログラムは，一般に実行速度が著しく低下するため，テストプログ

ラム実行時の，すべてのソースコードに対するカバレッジを取得してしまうと，テスト実行時間の増長を招く．

以上の理由から，RTOS 起動時は，カバレッジ取得状態を無効としておき，テスト対象のシステムサービス呼出し直前でカバレッジ取得状態を有効として，システムサービスの処理がすべて完了した時点で，カバレッジ取得状態を再度無効とするような仕組みが必要となる．

2.5 要件 (5) コマンドラインからの実行

AKTSP では，テスト実施を自動化するための仕組みとして，ビルドや実行モジュールのロード，実行を行うためのシェルスクリプトも同梱しており，コマンドラインによるテストの自動実行が可能である．したがって，GUI ツールによるカバレッジ取得用コードの埋め込みが，複数のテストプログラム毎に必要となると，実行効率の著しい低下を招く．これに対する要件として，事前に RTOS のソースコードに対して，カバレッジ取得用コードを埋め込んでおき，テストプログラムのビルドはコマンドラインから実行可能であることが求められる．

また，AKTSP でのテストケース数は，20 万件以上存在するため，分割して実行する場合でも，相当数のカバレッジ情報を取得することになる．要件 (3) のカバレッジ情報をマージする点においても，GUI ツール上で 1 ファイルずつ追加していくような方法は，非現実的であるため，コマンドラインからマージ可能なカバレッジ情報であることが望ましい．

2.6 要件 (6) メモリ保護機能への対応

ATK2 におけるメモリ保護機能では，効率のよいアクセス制御のためのメモリ配置を実現するために，リンカスクリプトの作成も RTOS の機能が実現する．また，メモリ領域の初期化処理も RTOS 内で行うため，これらの処理が，カバレッジ取得用コードや，カバレッジ取得時に使用する標準ライブラリと干渉しない必要がある．

3. MC/DC 計測の実現

本章では，2 で述べた要件に合致するツールの選定，および確立した MC/DC 計測手法について述べる．

3.1 MC/DC 計測ツールの選定

我々が調査した限りで，MC/DC 計測が可能なフリーウェアは存在せず，市販ツールでは，以下の 3 つがあった．

(a) C++test [*4]
(b) カバレッジマスター winAMS [*5]
(c) LDRA Testbed [*6]

[*4] http://www.parasoft.com/cpptest
[*5] http://www.gaio.co.jp/product/dev_tools/pdt07_winams.html
[*6] http://www.ldra.com/index.php/ja/products-a-services/ldra-tool-suite/ldra-testbed

```
                    :
CLOG: 20 30 1C 00 00 00 00 30 50 30 20 00 00 00 00 30
CLOG: 50 30 20 00 00 00 00 30 50 30 20 00 00 00 00 30
CLOG: 20 30 1C 00 00 00 00 30 50 30 20 00 00 00 00 30
CLOG: 20 30 1C 00 00 00 00 30 50 30 20 00 00 00 00 30
CLOG: 50 30 20 00 00 00 00 30 50 30 20 00 00 00 00 30
                    :
```

図 1 カバレッジ情報の例
Fig. 1 Sample of coverage infomation

(b) は，シミュレータでの実行によりカバレッジ情報を取得するツールで，(a)，(c) は，カバレッジ取得用コードの埋め込みによってカバレッジ情報を取得するツールである．したがって，2.1 で述べた要件を満たすかを確認するために，我々は (a) と (c) を比較した．

どちらも基本的な機能は同じであったが，(c) は，パスの実行回数を 0 回か 1 回しか記録できない，RTOS 特有の複雑なソースコードに対して，カバレッジ取得用コードの埋め込みができないなど，そもそも我々のニーズに合致しなかった．逆に (a) は，我々が求める要件を満たしていたため，(a) を採用した．

3.2 要件の実現

本節では，2 で述べた要件の実現方法を説明する．

3.2.1 要件 (1) カバレッジ情報の取得方式

3.1 で述べた通り，採用した C++test は，カバレッジ取得用コードを埋め込んで，カバレッジ情報を取得するツールであるので，C++test を採用することにより，要件 (1) を満たした．

3.2.2 要件 (2) カバレッジ情報の保存方法

ATK2 では，実行するターゲットシステムへのポーティング作業の一部として，システムログ機能を提供する．システムログ機能は，UART などを使用して，OS 処理のトレースログや，システムサービスの異常事象などを，主にホスト PC へ出力する．システムログ機能により，どのようなターゲットシステムであっても，ASCII コードで表現できる文字列であれば，ホスト PC に送信し，テキストファイルに保存することができる．

C++test におけるカバレッジ情報は本来バイナリデータであったが，C++test が使用するカバレッジ取得用ライブラリを修正することで，テキストデータへ変換し，システムログ機能を使ってホスト PC へ送信することが可能となったため，要件 (2) を満たすことができた．テキストデータに変換したカバレッジ情報の例を図 1 に示す．

また，取得したテキストデータのカバレッジ情報を，元のバイナリデータに変換することで，C++test が解釈可能なカバレッジ情報を得ることができる．

なお，カバレッジ取得用ライブラリの修正，およびテキストデータをバイナリデータに変換するツール (以後，バ

イナリ変換ツール) の開発は，C++test に関する専門的な知識が必要であるので，C++test の販売代理店であり，技術サポートを行っているテクマトリックス株式会社[*7]に依頼した．

3.2.3 要件 (3) カバレッジ情報のマージ

3.2.2 で述べたように，カバレッジ情報はテキストデータとしてホスト PC で保存することができる．テキストデータの各行が，1 カバレッジ情報 (コード上のどの行，どの分岐をパスしたか) を意味するので，分割されたカバレッジ情報を取得した場合，単純にテキストデータを結合することで，カバレッジ情報をマージすることができる．よって，要件 (3) を満たした．

また，マルチコア対応のターゲットシステムにおいては，コア毎にカバレッジ情報が別々になったとしても，コア毎にカバレッジ情報を取得して結合することで，1 つのカバレッジ情報とすることができる．

3.2.4 要件 (4) カバレッジ取得状態の切り替え

C++test のカバレッジ取得用ライブラリは C 言語プログラムで提供されているため，カバレッジ取得関数の先頭で，カバレッジ取得状態の有効/無効を意味する変数を用意しておき，この変数を切り替える関数を，テスト対象のシステムサービス呼出しの前後で実行することで，意図した部分のみのカバレッジ情報を取得することが可能であった．よって，要件 (4) を満たした．

3.2.5 要件 (5) コマンドラインからの実行

C++test は，GUI ツールによって，カバレッジ取得用コードの埋め込みからビルドまで実行する仕組みになっている．しかし，複数のテストプログラムに対して，毎回 GUI ツールを実行してビルドするのは現実的ではない．また，複数のテストプログラムによってテストを実施する場合，ATK2 のソースコードには変更がないため，ATK2 のソースコードのみをライブラリ化した上で，各テストプログラムのみをコンパイルしてリンクすることで，全体のビルド時間を短縮することができる．

以上から，C++test では，カバレッジ取得用コードを埋め込んだ ATK2 のソースコードのライブラリ化を行うのみとして，テストプログラムのビルドは行わないことにした．これにより，カバレッジ取得用コードを埋め込んだ ATK2 のライブラリ (以後，カバレッジ対応 ATK2 ライブラリ) を使って，各テストプログラムをコンパイル，リンクすることで，カバレッジを取得するテストプログラムの実行がコマンドラインから可能になる．よって，要件 (5) を満たした．

また，カバレッジ情報はテキストファイルであるので，容易にコマンドラインからマージ可能であり，バイナリ変換ツールもコマンドラインで実行できるプログラムとして開発したため，要件 (5) を満たしている．

3.2.6 要件 (6) メモリ保護機能への対応

RTOS では，スタートアップモジュールなどもターゲットシステムに応じて実装するので，コンパイラが用意する標準ライブラリも使用しないことが一般的である．また，メモリ保護機能を使用する場合，標準のリンカスクリプトも使用しない．

カバレッジ取得用ライブラリでは，標準ライブラリに含まれる機能をいくつか使用しているが，すべての標準ライブラリをリンクしてしまうと，スタートアップモジュールを置き換えてしまうなどの副作用が発生してしまう恐れがある．そこで，我々はまず，必要な標準ライブラリをリストアップし，最小限の標準ライブラリだけをリンクすることにした．最小限の標準ライブラリだけでリンクを行ったところ，標準ライブラリが配置されるセクションが不足していたため，ATK2 が生成するリンカスクリプトに対して，必要なセクションが生成されるように修正を行った．

以上の対応を行い，カバレッジ取得用コードを埋め込んだテストプログラムを実行したところ，初期化処理において CPU 例外が発生する現象が発生した．ATK2 では，メモリ保護機能を使用する場合，BSS セクションおよび DATA セクションの初期化も OS 内の初期化処理で行うが，このメモリ初期化処理に対しても，カバレッジ取得用コードが埋め込まれてしまっていたことが原因であった．カバレッジ情報も一時的にメモリ上に展開するため，そのメモリ初期化処理と同時にカバレッジが取得され，CPU 例外が発生していたのである．そこで，カバレッジ取得を行う場合は，メモリ初期化処理に対してのみ，カバレッジ取得用コードを埋め込まないように対応することで問題を回避し，メモリ保護機能を使用した場合でも，MC/DC が取得できることを確認した．よって，要件 (6) を満たした．なお，メモリ初期化処理自体の MC/DC に関しては，別途レビューにて実施することにした．

4. 確立した MC/DC 計測手法

本章では，3 で実現した方法を用いて確立した，MC/DC 計測を行う手法について述べる．確立した MC/DC 計測手法による，作業フローを図 2 に示す．

4.1 カバレッジ対応 ATK2 ライブラリの作成

本作業は，GUI ツールである C++test によって行うが，最初に 1 回だけ実施すればよい．

まず，C++test にカバレッジ取得対象である，ATK2 のソースコードを登録する．さらに，ATK2 のビルドのために，ダミーのテストプログラムも登録する．コンパイルに関する設定などを行い，C++test により，カバレッジ取得用コードの埋め込み，およびカバレッジ対応 ATK2 ライブラリの生成を行う．この際，3.2.6 で述べたメモリ初期化処

[*7] https://www.techmatrix.co.jp/

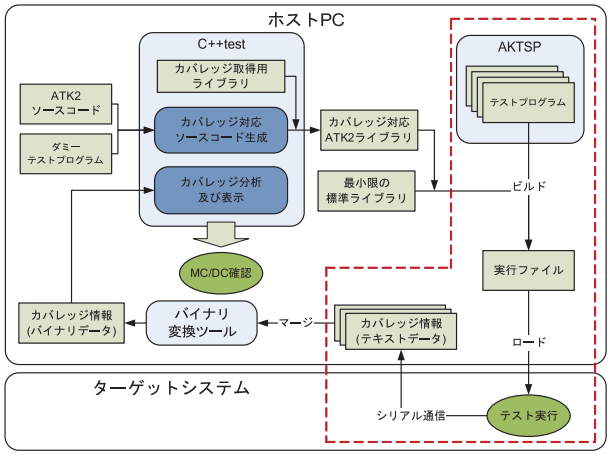

図 2 MC/DC 計測の流れ
Fig. 2 Flow of measuring MC/DC

理には，コンパイルオプションでカバレッジ取得用コードが埋め込まれないように設定する．

なお，カバレッジ取得用ライブラリは最初から登録されており，カバレッジ対応 ATK2 ライブラリに含まれる．

4.2 テストプログラムのビルドおよび実行

2.5 で述べたように，AKTSP ではテストプログラムのビルドや実行をコマンドラインからシェルスクリプトによって実行可能である．ここで，テストプログラムとリンクする ATK2 ソースコードを，カバレッジ対応 ATK2 ライブラリへ置き換えてビルドを行うようにシェルスクリプトを修正することにより，コマンドラインのままカバレッジを取得するテストプログラムを実行する．この際，3.2.6 で述べた，最小限の標準ライブラリもリンクする．

テストプログラム実行時，3.2.2 で述べたシステムログ機能によって，ホスト PC 側でテキストデータのカバレッジ情報を受信するので，これをテキストファイルに保存する．

このテストプログラムのビルド，ターゲットシステムへのロード，実行，カバレッジ情報の取得，という一連のサイクル (図 2 の破線で囲んだ部分) をテストプログラムの数だけ繰り返す．

4.3 カバレッジ情報のマージおよび変換

すべてのテストプログラムを実行した後，取得したすべてのカバレッジ情報を cat コマンドでマージする．マージしたカバレッジ情報を，バイナリ変換ツールで C++test が読込み可能なバイナリデータへ変換する．このマージ，および変換処理もコマンドラインで実行可能であるので，AKTSP の機能として追加した．

4.4 MC/DC の確認

本作業は，GUI ツールである C++test によって行うが，最後に MC/DC を確認する際にのみ実施するものである．

取得したバイナリデータのカバレッジ情報を，C++test に入力し，MC/DC を表示する．

5. MC/DC 計測の実施結果

本章では，4 の手法を用いて，実際に MC/DC 計測を実施した結果について述べる．

5.1 実施環境

実施に使用した環境は以下の通りである．なお，ATK2-SC3 は，メモリ保護機能をサポートした ATK2 の機能セットの名称である．

- RTOS：ATK2-SC3 Release 1.2.1 [*8]
- ターゲットシステム：
  Altera DE2-115 Development and Education Board
- コンパイラ：gcc version 4.1.2
- ホスト PC：
  - OS：Windows7 Professional 64bit
  - CPU：Intel Core i7-3770 3.40GHz
  - RAM：8GB

5.2 MC/DC の確認

ATK2-SC3 に対するシステムサービステストのテストケース数は，40,609 件である．対象としたターゲットシステムの場合，メモリ制約上，このテストケースを 40 個のテストプログラムに分割して実行する必要がある．AKTSP では，テストプログラム生成ツール [1] を使用することで，指定した分割数に応じたテストケースのみを実行することができる．

全システムサービステストを実施した結果，MC/DC は 60%であった．C++test で MC/DC を表示した画面を図 3 に示す．MC/DC 計測の対象とした OS のソースコードは，kernel フォルダ内のファイルのみである．

MC/DC が 100%ではなかったので，カバーしていないコードを確認したところ，システムサービステストのテスト設計上カバーしないコードと，テスト対象のシステムサービス実行中のカバレッジのみ取得することによってカバーしないコードのみであることを確認した．したがって，MC/DC を正しく取得できていると考えることができる．

なお，システムサービステストでカバーできないコードは，主に以下のようなものである．

- OS オブジェクトの初期化，終了処理
- ハードウェアタイマのカウントタイミングに依存する分岐
- OS 起動前など OS 外からのシステムサービス呼出しに対する処理
- システムサービスからは実行されないコード

MC/DC がカバーされていないコードを，C++test で表示した例を図 4 に示す．if 文に，2 つの条件式が AND 条件で指定されており，[true & true = true] の組み合わせ

[*8] http://www.toppers.jp/atk2-download.html

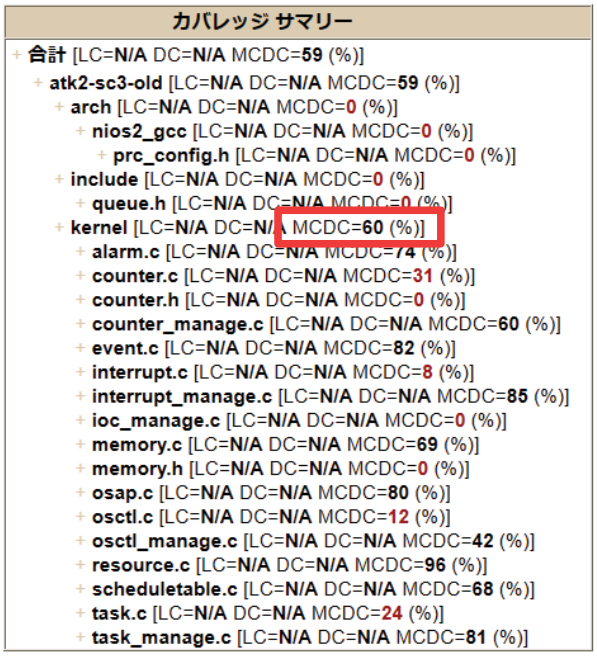

図 3 MC/DC の表示画面
Fig. 3 Screen image of MC/DC

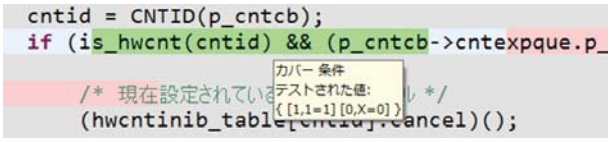

図 4 MC/DC がカバーされていないコードの例
Fig. 4 Sample of MC/DC uncovered code

と，[false & X = false] の組み合わせしか評価されていないことを意味する．後者の組み合わせは，左側の条件式が false の場合，AND 条件であるので，右側の条件式は評価されないことを示す．MC/DC は，右側の条件式の判定結果が独立して if 文全体の判定結果となる組み合わせも実施する必要があるので，[true & false = false] とするためのテストケースの追加が必要ということである．

5.3 手法以外の工夫

5.3.1 処理速度の改善

当初，本手法を用いて，ATK2-SC3 に対してシステムサービステストを実行したところ，実行時間は約 42 時間であった．カバレッジ取得用コードを埋め込んでいない元の状態であれば，約 2 時間であったので，約 21 倍になってしまっている．これは，カバレッジ情報を 1 行出力する度に，システムログ機能によってホスト PC に送信していたことによる，毎回の通信開始/終了処理のオーバーヘッドが原因であった．

そこで，我々はカバレッジ情報が出力される都度の送信ではなく，複数回分のカバレッジ情報をまとめて送信するように，C++test のカバレッジ取得用ライブラリを修正した．ただし，過度に多くのカバレッジ情報をまとめて送信しようとすると，今度は送信処理時間自体が長くなり，こちらがボトルネックになってしまう．実際にいくつかの回数で処理時間を測定し，1,600 回に 1 回程度の頻度でまとめて送信することで，最も処理時間が早くなることを確認した．

結果として，ATK2-SC3 に対するシステムサービステストテストの実行時間は約 8 時間となり，修正前のカバレッジ取得時間から約 1/5 程度に短縮することができた．

5.3.2 差分カバレッジ情報の取得

MC/DC が 100%ではなかったため，我々は MC/DC を 100%とするためのテストケースを別途追加し，最終的に ATK2-SC3 の MC/DC を 100%とすることができた．この際，システムサービステストに対するカバレッジ情報を，テキストデータで保存しておくことで，追加したテストケースに対するカバレッジ情報をマージすることができ，再度，システムサービステストを実施する必要をなくすことができた．これは，カバレッジ情報を容易にマージ可能としたことによるメリットである．

6. まとめ

本論文では，ATK2 を題材にして，RTOS に対する MC/DC を計測する手法について述べた．本手法では，カバレッジ取得用コードをテスト対象のソースコードに埋め込む方式の MC/DC 計測ツールである，C++test を用いて，MC/DC の計測を行った．ターゲットシステム上での実行であってもカバレッジ情報を取得する方法や，カバレッジ情報のマージ，指定した箇所のみのカバレッジ取得などの要件に対しては，カバレッジ取得用ライブラリのカスタマイズで対応した．また，AKTSP によるテスト実施においても，テスト実施の自動化を損なうことなく，MC/DC 計測が可能であることを確認した．

本手法を用いれば，ATK2 を使用して開発した車載システム向けのソフトウェアに対しても MC/DC 計測が可能となり，機能安全規格で規定された基準を満たすことを確認することができる．また，RTOS に限らず，ターゲットシステム上でしか実行できない組込みソフトウェアに対しても，システムログ機能と同等の機能を有すれば，MC/DC 計測が可能である．

謝辞

C++test の使用方法調査，ライブラリ開発に協力して下さった，テクマトリックス株式会社の皆様に謹んで感謝の意を表する．

参考文献

[1] 松本充広, “MC/DC による現実的な網羅のススメ”, キャッツ組込みソフトウェア研究所, 2009.
[2] 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.