Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

組込み向け高信頼デュアルOSモニタのマルチコアアーキテクチャへの適用

太田貴也[†1] Daniel Sangorrin[†1] 一場利幸[†1]
本田晋也[†1] 高田広章[†1]

本研究では組込み向け高信頼デュアル OS モニタ SafeG のマルチコアプロセッサ対応を行った．SafeG は組込み向けシングルコアプロセッサ上で，リアルタイム OS(RTOS) と汎用 OS を同時実行するために提案・実装された小規模なソフトウェアモジュールである．SafeG をマルチコアに対応させるために，シングルコアプロセッサ向け SafeG の起動処理，割込みハンドリング，OS 切り替え処理を変更した．評価の結果，実行オーバヘッドは約 0.17μs の増加，プログラムサイズは約 1.3KB の増加で，SafeG をマルチコアに対応できることを確認した．

Application of a high-reliability embedded dual-OS monitor to a muliticore architecture

TAKAYA OHTA,[†1] DANIEL SANGORRIN,[†1]
TOSHIYUKI ICHIBA,[†1] SHINYA HONDA[†1]
and HIROAKI TAKADA [†1]

In this research we extended SafeG, a high-reliability dual-OS monitor, with support for multi-core processors. SafeG is a small software module designed to concurrently execute a real-time OS (RTOS) and a general-purpose OS (GPOS) on top of the same embedded single-core processor. To extend SafeG to support for multi-core prosessors, we changed SafeG code such as startup, interrupt handling, and OS switching. As the result of evaluation, the execution time and the program size of extended SafeG increase only 0.17 μs and 1.3 KB, respectively.

1. はじめに

組込みシステムにはリアルタイム性や信頼性が求められることが多い．近年，組込みシステムの利用形態は多様化，複雑化しており，リアルタイム性を持ちつつ，汎用 OS を利用したいという要求がある．一方，組込みシステムにおいてもマルチコアシステム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．

リアルタイム性を持ちつつ，汎用 OS を利用したいという要求に答えるために，リアルタイム OS(RTOS) と汎用 OS を同時に実行し，両者の特徴を併せ持ったシステム構築を行う方法が考えられる．複数の OS を同時に実行する技術として，ハイブリッド OS 方式や仮想マシン方式がある．

ハイブリッド OS 方式として Linux on ITRON[1] があげられる．この方式では，汎用 OS と RTOS が共にシステム上のすべてのリソースにアクセス可能な権限で実行されるので，実行性能を高めやすい．しかし，汎用 OS にバグやセキュリティホールなどがあった場合に，汎用 OS から RTOS リソースへアクセスがなされる可能性がある．この場合，RTOS を汎用 OS から保護できないといった問題がある．

仮想マシン方式として，Xen[2] や L4[3] などの方式があげられる．仮想マシンを提供するシステムソフトウェアは，ハイパーバイザ (仮想マシンモニタ) と呼ばれている．仮想マシン方式には完全仮想化や準仮想化といった実現方式がある．完全仮想化では，ゲスト OS に変更を加えることなく，複数のゲスト OS を同時に実行できる．準仮想化では，ゲスト OS の改変が必要であるが，完全仮想化よりも仮想マシンのオーバヘッドを抑えることができる．これらの方式は，OS 間保護が可能であり，仮想マシンの構成を自由に変更できるため，柔軟性の高いシステム構築ができるという特徴を持っている．しかし，完全仮想化で提供される仮想マシンはオーバヘッドが大きく，リアルタイム性は考慮されていない．準仮想化では仮想マシンオーバヘッドを少なくすることができるものの，ゲスト OS への変更量が多くなることが問題である．また，これらの方式では，ハイパーバイザの規模が大きいために，コード検証が困難になる．

以上のように既存の方式の多くは，組込みシステムへの適用時に考慮すべき多くの問題を

†1 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University

Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

抱えている．そこで，組込みシステム向けのデュアル OS 環境として，ハイブリッド OS 方式を高信頼化した方式である，高信頼デュアル OS モニタ方式[4),5)] が提案・実装された．この方式では，セキュリティ支援ハードウェア ARM TrustZone[6),7)] を利用することで，RTOS と汎用 OS を同時実行し，RTOS を保護しつつ，システム全体のリアルタイム性を保証することが可能となっている．この方式では，SafeG と呼ばれる小規模なソフトウェアモジュールによって，OS 切り替えが行われる．しかし，SafeG はシングルコアプロセッサ上でしか動作せず，マルチコアプロセッサ上では動作しない．そのため，前述のようなマルチコアプロセッサの恩恵を受けることができない．

そこで本研究では，マルチコアシステム上で SafeG を利用するために，マルチコア対応 SafeG の提案・実装を行った．SafeG をマルチコアに対応させるために，シングルコアプロセッサ向け SafeG の起動処理，割込みハンドリング，OS 切り替え処理を変更した．評価実験として，SafeG のプログラムサイズおよび実行時間オーバヘッドを計測した．この結果，SafeG をマルチコアアーキテクチャに適用することで生じる，実行時間オーバヘッドの増加は約 0.17μs に，プログラムサイズの増加は約 1.3KB に抑えられたことを確認できた．

以降この論文では，シングルコアプロセッサ向けの SafeG を単に SafeG と呼び，マルチコアプロセッサ向けに拡張を施した SafeG を SafeG-MP と呼ぶことにする．

2. シングルコアプロセッサ向けデュアル OS モニタ (SafeG)

SafeG は ARM プロセッサのセキュリティ支援技術 TrustZone を用いることで，シングルコアプロセッサ上で，高信頼なデュアル OS 環境構築を可能とするソフトウェアモジュールである．ここでは，ARM TrustZone と SafeG のアーキテクチャについて述べる．

2.1 TrustZone

TrustZone は ARM プロセッサのセキュリティ支援技術であり，ARMv6 アーキテクチャの拡張技術として最初に実装された．TrustZone はプロセッサにトラスト状態，ノントラスト状態という 2 つの状態の概念をもつ．それぞれの状態のプロセッサには特権モード，非特権モードが存在する．また多くのレジスタはトラスト状態，ノントラスト状態で個別に用意される．トラスト状態のプロセッサの振る舞いは従来のプロセッサと同一であるが，ノントラスト状態のプロセッサからは，特権モードであっても，トラスト状態のプロセッサが使用すると設定されたデバイス，メモリ領域へのアクセスが禁止されるといった制限が儲けられる．2 つの状態のレジスタのうち，MMU 制御用などの一部のレジスタはバンクレジスタとなっている．これらバンクレジスタは，プロセッサの状態に合わせて自動的に切り替えられる．しかし，汎用レジスタはバンクレジスタになっておらず，ソフトウェア側で復帰・保存処理を行う必要がある．

2 つのプロセッサ状態の制御を行うために，TrustZone の実装されたプロセッサにはセキュアモニタモードと呼ばれるモードが追加された．トラスト状態とノントラスト状態の切り替え処理は，このセキュアモニタモードを経由して行う．

ARM プロセッサには高速割込み (FIQ) と通常割込み (IRQ) という，2 種類の割込みが存在する．2 つの状態のプロセッサにはそれぞれ個別の割込みベクタテーブルが存在する．プロセッサがトラスト状態のときに発生した割込みはトラスト状態の割込みベクタ (トラストベクタ) で，ノントラスト状態で発生した割込みはノントラスト状態の割込みベクタ (ノントラストベクタ) でハンドリングする．また，セキュアモニタモードにも割込みベクタテーブル (モニタベクタ) が存在し，FIQ，IRQ が発生した場合，セキュアモニタモードのベクタを実行するか，現在の状態のベクタを実行するか指定できる．

ARM プロセッサでは，割込みを禁止する際，IRQ と FIQ を個別に禁止することが可能である．TrustZone においては，ノントラスト状態で FIQ を禁止できないように制限することが可能である[⋆1]．トラスト状態にはこの制限はない．つまり，任意に FIQ と IRQ を禁止可能である．

2.2 SafeG

SafeG は組込み向けのシングルコアプロセッサ上で，RTOS と汎用 OS を同時に実行するために研究，開発された非常に小規模なソフトウェアモジュールである．この方式は従来のハイブリッド OS 方式を，セキュリティ支援ハードウェアにより高信頼化したものである．具体的には，RTOS の処理を優先することでリアルタイム性の保証ができ，RTOS が使うリソースを汎用 OS からハードウェア的に保護することができる．**図 1** は，SafeG による方式を示した図である．

SafeG はセキュアモニタモードで実行される．また，RTOS はトラスト状態のプロセッサで，汎用 OS はノントラスト状態のプロセッサで実行される．RTOS が使うリソースをトラスト状態のプロセッサが利用する領域に設定する．これにより，汎用 OS から RTOS のハードウェアリソースを保護することが可能である．

SafeG による方式では，次のように RTOS 側処理を優先して，リアルタイム性の保証を行う．割込みについては，まず FIQ と IRQ を一旦 SafeG でハンドリングするために，各

⋆1 ノントラスト状態で IRQ は常に禁止可能．

Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

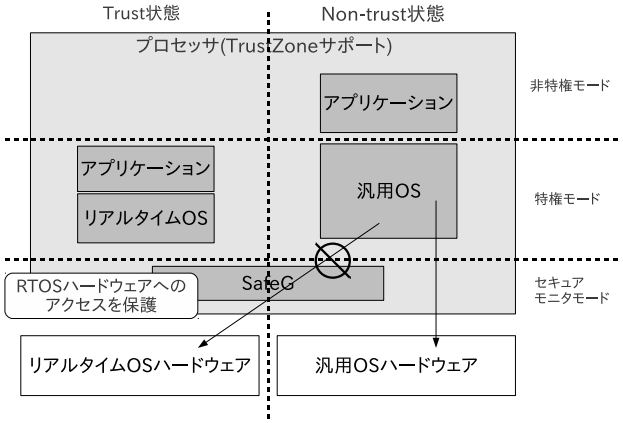

図 1 SafeG による方式
Fig. 1 Concept of SafeG

プロセッサ状態で割込み発生時にモニタベクタを実行するように設定する．そして，トラスト状態のプロセッサで FIQ，ノントラスト状態のプロセッサでは IRQ を使うように設定する．さらにノントラスト状態のプロセッサで，FIQ を禁止できないようにする．トラスト状態のプロセッサでは IRQ を常に禁止する．以上の設定により，RTOS の割込みは FIQ で入力される．また，汎用 OS の割込みは IRQ で入力される．RTOS 実行中は，IRQ は受け付けない．したがって，RTOS 実行中に汎用 OS の割込みが受け付けられ，RTOS 処理が中断されることはない．また，汎用 OS 実行中に FIQ を禁止できないので，汎用 OS から RTOS 側割込みを禁止することはできない．これによって，RTOS の処理を優先することが可能である．

システム稼働時に SafeG が提供する機能は OS 切り替えと，割込みの分配処理に限定される．これは SafeG が，検証容易性とオーバヘッドを最小限に抑えるため，最小限の機能に限定するという方針に基づいて作られているためである．

SafeG には次の 5 種類の実行パスが存在する．

(1) RTOS 実行中に FIQ 入力
(2) 汎用 OS 実行中に FIQ 入力

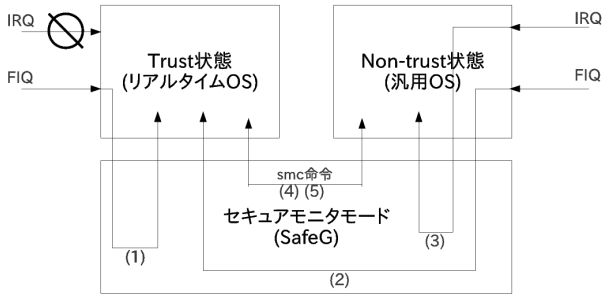

図 2 SafeG の実行パス
Fig. 2 Execution path of SafeG

(3) 汎用 OS 実行中に IRQ 入力
(4) RTOS 実行中に OS 切り替えを依頼
(5) 汎用 OS 実行中に OS 切り替えを依頼

実行パスをまとめると**図 2** のようになる． すべての割込みは一旦 SafeG でハンドリングされ，その後適切な OS でハンドリングする．

RTOS 実行時には IRQ を常に禁止するように設定するため，汎用 OS 側の割込み処理の影響を受けない．また，RTOS の使用するデバイスはすべて FIQ として入力されるので，自身の使用するデバイスからの割込みは受け付けられる [実行パス (1)]．

汎用 OS 実行時には FIQ も IRQ も受け付ける．FIQ が入力された場合，RTOS 側の割込み処理が入ったことになる．この場合はコアのトラスト/ノントラスト状態が切り替えられ，RTOS 側でハンドリングされる [実行パス (2)]．もし IRQ が入力された場合には汎用 OS の使用するデバイスからの割込みを受け付けたことになるので，汎用 OS 自身がハンドリングする [実行パス (3)]．

各 OS は任意の時点で SafeG を呼び出し，SafeG に OS 切り替えを依頼することができる [実行パス (4),(5)]．SafeG の呼び出しはトラップ命令の一種である，SMC(Secure Monitor Call) 命令により実現される．SMC 命令を実行すると SMC 例外が起こり，SafeG の SMC ハンドラにジャンプする．このハンドラで OS 切り替え処理を行い，切り替え対象 OS の前回切り替えを行った際の状態を復帰する．OS 切り替え時，SafeG はプロセッサ状態が切り替わる際にバンクされていないレジスタの復帰・保存処理を行う必要がある．レジスタの復帰・保存処理は，OS の切り替わる全てのパス (2),(4),(5) で行われる．

Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

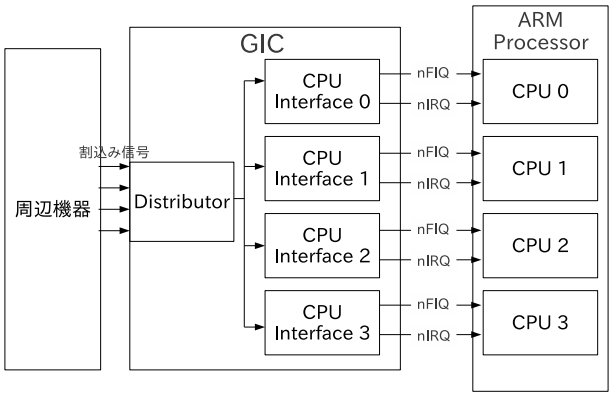

図 3 TrustZone 対応マルチコアプロセッサにおける割込みアーキテクチャ
Fig. 3 Interrupt Architecture for Multicore Processor with TrustZone

3. マルチコアプロセッサ向けデュアル OS モニタ (SafeG-MP)

ここでは，マルチコアプロセッサにおける TrustZone の仕組みと，SafeG-MP の設計および実装について説明する．

3.1 TrustZone 搭載マルチコアプロセッサ

TrustZone をサポートしたマルチコアアーキテクチャは，ARMv7-A アーキテクチャより採用されており，その実装として，Cortex-A9 MPCore[8] があげられる．このプロセッサは対称型シングルチップマルチコアプロセッサである．すなわち，ひとつのチップ上に同じプロセッサコアが複数実装されている．したがって，各プロセッサコアは同一のアドレス空間を持っている．TrustZone の実装されているマルチコアプロセッサでは，コアそれぞれでトラスト/ノントラスト状態を独立に指定できる．

割込みは**図 3** に示すように，一旦割込みコントローラ (GIC[9]) で受け付けられる．割込みをどのコアに入力するかどうかは，GIC で設定することができる．図 3 に示す Distributor によって，設定されたコアへ周辺機器の割込みを分配する． 割込みが発生した場合，割込みを受け付けたコアでは割込みベクタテーブルにジャンプし，適切な割込みハンドラを実行する．シングルコアの場合と同様に，割込みベクタテーブルには，トラストベクタ，ノントラストベクタ，モニタベクタの 3 つのベクタテーブルが存在する．また，各コアで FIQ と IRQ 発生時に，モニタベクタを実行するか，現状態のベクタを実行するかもシングルコア同様に設定可能である．割込みが発生した場合に実行するベクタテーブルのアドレスは各コアごとに個別に指定可能である．したがって割込み発生時，メモリ上の共通のベクタテーブルに対して各コアからアクセスを行うか，コアごとに異なるベクタテーブルを用意するかが選択できる．

3.2 SafeG-MP の設計方針

SafeG-MP では SafeG の設計方針をベースとして，マルチコアプロセッサ拡張を行う．SafeG-MP の設計方針として，以下を設定した．

(a) 2 つのマルチコアプロセッサ向け OS の同時実行

SafeG-MP では，対称型マルチコアアーキテクチャでの動作を対象とする OS を，2 つ同時実行することとする．実際のシステムにおいては，OS の切り替えが不要なコアが存在する可能性があるが，SafeG-MP の実装では全てのコアで OS 切り替えが可能であることが望ましい．SafeG 同様にトラスト状態で RTOS を，ノントラスト状態で汎用 OS を実行することを想定する．

(b) 小規模

SafeG-MP はシステム信頼性の要となるソフトウェアであるため，小規模で検証が容易なシステムであることが望ましい．そこで，マルチコアプロセッサ対応による，プログラムサイズ増加を最小限にとどめる必要がある．そのために，SafeG-MP は割込みの配分と OS 切り替えのみを行い，OS スケジューリングなどは直接行わないこととする．

(c) 低実行時間オーバヘッド

SafeG-MP の実行により，RTOS のリアルタイム性に影響が生じないようにする必要がある．そのために，SafeG-MP 自身の実行時間オーバヘッドはできる限り小さくする．

(d) プロセッサ数に対するスケーラビリティ

プロセッサ数の変更に対して，容易に対応可能とするように SafeG-MP を実装する．プログラムサイズと実行時間はコア数が増えても変化が生じないようにする．ただし，コア数ごとの退避領域の増加は避けられないので，データ領域の増加は許容する．

(e) リアルタイム性の確保

Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

リアルタイム処理は RTOS が担当する．リアルタイム性を確保するために，RTOS が汎用 OS より優先して実行されるようにする．

(f) RTOS の保護

RTOS は機器制御に用いられることがある．そのため，汎用 OS からの影響で RTOS のデータが破壊されることがあってはならない．したがって，RTOS は汎用 OS から保護される必要がある．

(g) 最小の OS 変更点

OS への変更点はできる限り少ないことが望ましい．とくに OS のコア部分 (割込みエントリ，ディスパッチャ内部のコードなど) を変更しない方針で設計・実装を行う．

3.3 SafeG-MP の実装

3.3.1 複数コアからの SafeG-MP バイナリ共有

SafeG は OS 切り替え時に，プロセッサのレジスタ情報をメモリ上に退避する．マルチコアプロセッサにおいては各コアごとにレジスタが存在するので，シングルコアプロセッサを対象とした従来の SafeG をそのまま使用すると，別のコアの退避領域が上書きされる可能性がある．このため，SafeG をマルチコアプロセッサ向けに拡張するにあたり，退避領域の上書きに対する対処を行う必要がある．

SafeG-MP の実現方法として，コアごとに SafeG-MP のバイナリを用意する方法と，コアごとに用意すべき領域のみ個別に用意して，SafeG-MP は単一のバイナリとする方法が考えられる．コアごとに SafeG-MP のバイナリを用意する方法では，コードがコア数に比例した分だけ必要になる．また，各コアのモニタベクタテーブルのアドレスが変わるので，それぞれのコアごとに TrustZone の初期化方法を変える必要がある．SafeG-MP を単一のバイナリとする方法では，SafeG-MP が退避領域からロードする際，SafeG-MP を実行しているコアと一致する領域からロードを行うように SafeG のコードを変更する必要がある．

2 つの方法を比較すると，後者の方法では退避領域のメモリ領域増加のみでマルチコアに対応できる．そのため，前者よりコア数増加に対するスケーラビリティが高い．また，プログラムサイズの増加も最小限に抑えることが期待できる．したがって，後者の方が設計方針 (b) および (d) を満たすのに適した方法である．以上の理由より，SafeG-MP は単一のバイナリとすることとした．

なお，この SafeG-MP の実現方式では，各コアで SafeG-MP が実行される際，それぞれの SafeG-MP は各コアごとに用意した領域にしかアクセスしない．したがって，SafeG-MP 内部で排他制御を行う必要はない．

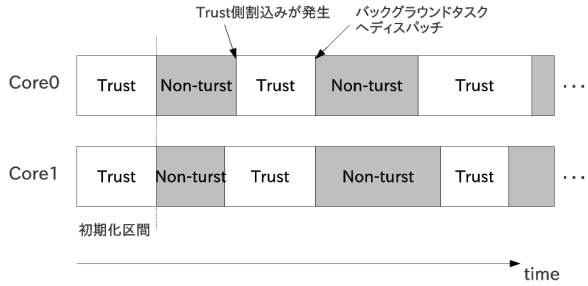

図 4 アイドルスケジューリング
Fig. 4 Idle scheduling

3.3.2 割込みハンドリング

SafeG 同様に SafeG-MP でも，FIQ，IRQ を一旦 SafeG-MP でハンドリングした後，割込みソースに対応する OS のハンドラを実行する．したがって，FIQ，IRQ が発生した場合，トラスト/ノントラスト状態に対応する通常のベクタではなく，モニタベクタを実行するようにすべてのコアを設定する．

マルチコアアーキテクチャにおいては 2.2 節で説明した実行パスをすべてのコアでもつことになる．つまり SafeG-MP は割込みが発生したコアで実行され，そのコアにおいてのみ割込みソースに応じて適切な OS のハンドラへジャンプするだけである．これは SafeG-MP 自身が割込みの各コアへの分配処理を行わないことを意味する．どの割込みソースをどのコアに割り振るかは，OS による割込みコントローラの設定による．SMC 例外発生時，または，汎用 OS 実行時に FIQ が発生した場合には OS 切り替え処理が行われる．この際，現在実行されているコアに関する退避領域に対して，レジスタの復帰と保存がなされる．

3.3.3 OS スケジューリング

設計方針 (e) を満たすために，OS スケジューリングとして，各コアで RTOS を優先的にスケジューリングする方式を採用した．この方式では，汎用 OS は RTOS の処理がないときにのみ実行され，汎用 OS 実行中に RTOS 側の割込みが入力された場合，即座に RTOS を実行する．これは汎用 OS が RTOS の最低優先度タスクとしてスケジューリングされることを意味する．このスケジューリングをアイドルスケジューリングと呼ぶ．アイドルスケジューリングにおいて，OS 切り替えの契機となるのは，RTOS の割込みと次に示すバックグラウンドタスクによる SMC 例外である．**図 4** は，アイドルスケジューリング方式の様子を示した図である．バックグラウンドタスク (BTASK) は各コアで RTOS がアイドルと

Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

なった場合に汎用 OS に実行 OS を切り替えるタスクである．このタスクは RTOS 上の全てのコアに配置する．処理内容は SMC 命令を発行し，OS の切り替えを SafeG-MP に依頼することである．BTASK の優先度は他のどのタスクよりも低く設定されている．結果として RTOS の処理が無くなった時点で BTASK が実行され，汎用 OS へ OS 切り替えが行われる．

4. 評　　価

評価環境として，TrustZone をサポートしたマルチコアプロセッサ Cortex-A9 MPCore を搭載した評価ボード KZM-CA9-01 を使用した．このボードの Cortex-A9 MPCore は 4 コア構成，コアクロック 400MHz，L1 キャッシュは命令・データ共に 32KB である．主記憶として使用できるメモリは 256MB の DDR2 SDRAM である．実装と評価環境簡略化のため，同時に動作させる 2 つの OS はどちらもマルチコアアーキテクチャ向けの RTOS として，TOPPERS/FMP カーネル[10](以下，FMP カーネルという) を使用した．これら 2 つの FMP カーネルのうち，一方をトラスト状態で，もう一方をノントラスト状態で実行した．以降，それぞれの FMP カーネルをトラスト側 FMP，ノントラスト側 FMP と呼ぶ．

評価項目として，設計方針 (b)(c)(d) について評価するために，SafeG-MP のプログラムサイズおよび，実行時間オーバヘッドを評価した．また，設計方針 (e) について評価するために，FMP カーネルの API 実行時間を SafeG-MP を使用した場合と，FMP カーネル単体で実行した場合で計測した．なお，計測対象とする API はマルチコア対応 OS 特有のものとして，プロセッサ間割込みを使用した API(act_tsk) の実行時間について計測した．設計方針 (g) について評価するために FMP カーネルに加えた変更点について評価した．最後に，実装した SafeG-MP が設計方針を満たすものであったかどうかを考察した．

4.1 SafeG-MP のプログラムサイズ

コアごとに退避領域を追加したことでプログラムサイズが増加する．プログラムサイズの増加量を知るために，SafeG と SafeG-MP のバイナリファイル (ELF 形式) のコードデータサイズ比較を行った．**表 1** が比較結果である．

SafeG-MP の方がデータサイズが増加しているのが確認できる．テキスト領域と BSS 領域の両方が増加している．テキスト領域に関しては，初期化時と通常の OS 切り替えや，割込みハンドリング時のコア ID の比較を行うコードの追加によるものと考えられる．BSS 領域に関しては，退避領域をコアごとに追加したことによる増加分であると考えられる．コアごとに必要な退避領域のサイズは 304byte である．BSS 領域の増加分はおおよそ 3 つのコアの退避領域と一致する．増加分の合計は 1344byte であり，SafeG-MP は SafeG のおよそ 1.6 倍のサイズである．評価環境として使用したボードの主記憶は 256MB である．汎用 OS の実行を想定しているため，使用できるメモリサイズは大きい．また，汎用 OS 自身が使用すると考えられるメモリ領域は，SafeG-MP のサイズに比べ非常に大きいと考えられる．したがって，1344byte のサイズ増加は許容できるものと考えられ，SafeG-MP は小規模なサイズで実装できたといえる．

表 1　SafeG と SafeG-MP のプログラムサイズ比較 (バイト)
Table 1　Comparision of program size between SafeG and SafeG-MP

	text	data	bss	合計
SafeG	1936	0	448	2384
SafeG-MP	2320	0	1408	3728

表 2　SafeG と SafeG-MP の実行時間オーバヘッド比較
Table 2　Comparision of execution overhead between SafeG and SafeG-MP

実行パス	実行時間 (マルチコア未対応)	実行時間 (マルチコア対応)
(1) RTOS 実行中に FIQ 入力	0.25 μs	0.25 μs
(2) 汎用 OS 実行中に FIQ 入力	0.68 μs	0.88 μs
(3) 汎用 OS 実行中に IRQ 入力	0.29 μs	0.29 μs
(4) RTOS 実行中に SMC 命令発行	0.63 μs	0.82 μs
(5) 汎用 OS 実行中に SMC 命令発行	0.68 μs	1.15 μs

4.2 SafeG-MP の実行時間オーバヘッド

SafeG-MP では，SafeG に対して異なる領域へレジスタ情報を退避するように変更を加えた．このため OS 切り替え時，各 OS のコンテキストを復帰するために，実行コアの ID を取得する処理が必要である．この処理により，SafeG より実行時間オーバヘッドが増加している．

このオーバヘッドの増加量を評価するために，図 2 で示した，各実行パスの SafeG と SafeG-MP それぞれの実行時間を計測した．計測はモニタベクタの各割込みハンドラの入り口から出口までの経路の実行時間を Cortex-A9 コア搭載のパフォーマンスカウンタを使用して行った．**表 2** は SafeG と SafeG-MP の実行時間の比較を行った結果である．

表 2 から各パスにおけるオーバヘッド増加は平均 0.172μs に抑えられていることが確認できる．実行パス (1) と (3) では，SafeG と SafeG-MP で実行時間の変化が見られない．この実行パスでは OS 切り替えが起こらない．したがって，プロセッサ状態は変化しないのでレジスタの退避と復帰の必要はない．そのため，SafeG と SafeG-MP でこの実行パス

Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

に変更を加える必要がなかった．こうした理由により，実行パス (1) と (3) では SafeG と SafeG-MP では実行時間の変化がみられなかったものと考えられる．実行パス (5) では，SafeG に対して SafeG-MP の実行時間オーバヘッド増加が大きい．ベースとした SafeG には，Integrated scheduling[11] と呼ばれる仕組みが実装されている．これにより実行パス (5) には，同じ SMC 命令を扱う実行パス (4) より，レジスタの復帰と保存処理が 1 回分多く存在する．これが実行時間オーバヘッドを大きくしている原因であると考えられる．

実行コアの ID を調べるために，コア ID の取得を行うコプロセッサへのアクセス命令の追加が必要であった．レジスタ情報の復帰および退避を行う際，取得した ID をインデックスとして，あらかじめ作成した退避領域のベースアドレスを保存したテーブルを引くようにした．これにより，コア数が増加した場合でも，SafeG-MP への変更点としては，退避領域とそのベースアドレスを保存したテーブルのメモリ領域増加のみで対応可能である．

4.3 各 OS の実行時間オーバヘッド

FMP カーネルでは，プロセッサをまたぐ API はプロセッサ間割込みを使用する．プロセッサ間割込みを使用した API の一例として，act_tsk がある．act_tsk は対象のタスクに対して起動要求を行う API である．対象タスクが休止状態である場合には，対象タスクに対してタスク起動時に行うべき初期化処理が行われ，対象タスクは実行できる状態になる．act_tsk を発行するタスクの割り付けられたコアと別のコアに割り当てられたタスクに対して act_tsk を発行する場合，プロセッサ間割込みが利用される． 評価実験では 2 コア構成とし，**図 5** のようにコア 0 に割り付けられたタスクから，コア 1 に割り付けられた休止状態のタスクを起動し，対象タスクが起動するまでの時間を計測した．

SafeG-MP による FMP カーネルの API 実行時間の変化を計測するために，計測は SafeG-MP を使用した場合と，SafeG-MP を使用せず FMP カーネル単体で実行した場合について計測した．SafeG-MP を用いた場合，2 つの FMP カーネルが実行されるので，計測はトラスト側 FMP，もしくはノントラスト側 FMP で行うパターンが考えられる．また，それぞれの場合に想定される状況として，コア 1 が計測を行う側のコアの状態と異なる場合が考えられる．したがって，評価実験では次の項目について評価を行った．(1) から (3) がトラスト側 FMP での計測項目であり，(4) と (5) がノントラスト側 FMP の計測項目である．

(1) SafeG-MP を用いず，直接トラスト側 FMP を実行した場合のプロセッサ間 act_tsk 実行時間
(2) SafeG-MP を用いて，コア 1 がトラスト状態のときに act_tsk を発行した場合のトラスト側 FMP のプロセッサ間 act_tsk 実行時間

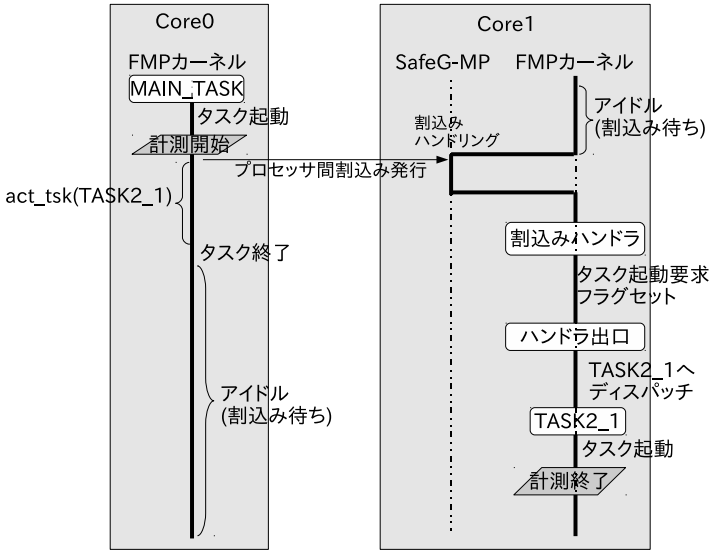

図 5 act_tsk 評価実験の内容
Fig. 5 Experiment of API(act_tsk) evaluation

(3) SafeG-MP を用いて，コア 1 がノントラスト状態のときに act_tsk を発行した場合のトラスト側 FMP のプロセッサ間 act_tsk 実行時間
(4) SafeG-MP を用いず，直接ノントラスト側 FMP を実行した場合のプロセッサ間 act_tsk 実行時間
(5) SafeG-MP を用いて，コア 1 がノントラスト状態のときに act_tsk を発行した場合のノントラスト側 FMP のプロセッサ間 act_tsk 実行時間

ノントラスト側 FMP ではコア 1 がトラスト状態のとき，コア 1 に対して IRQ(ノントラスト側 FMP のプロセッサ間割込み) を発生することができない．そのため，コア 1 がトラスト状態の場合の計測は行えない．

以降，計測結果と考察について述べる．まずトラスト側 FMP での計測結果を**図 6** に示す．この計測結果は，act_tsk 実行時間分布を示したヒストグラムである．計測項目 (1) の場合，すなわちトラスト側 FMP 単体で実行した場合には，1.65 μs 付近に最頻値が見られる．SafeG-MP を用いた場合は計測項目 (2) と計測項目 (3) である．どちらも計測項目 (1)

Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

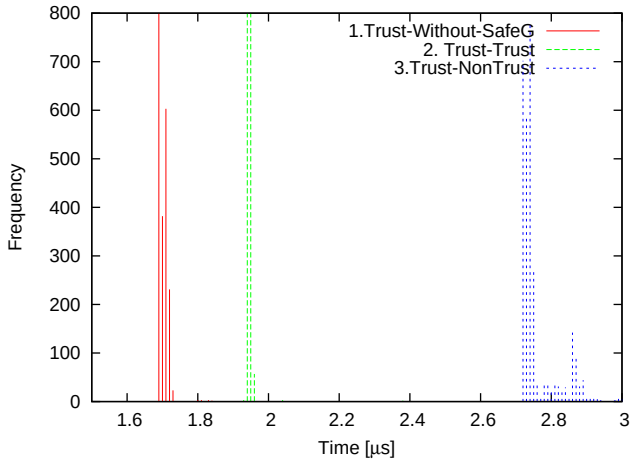

図 6 トラスト側 FMP の act_tsk 計測結果
Fig. 6 Evaluation results of API(act_tsk) on Trust side FMP

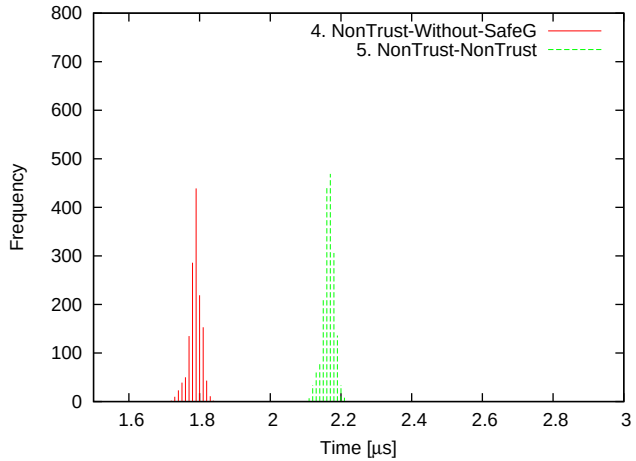

図 7 ノントラスト側 FMP の act_tsk 計測結果
Fig. 7 Evaluation results of API(act_tsk) on Non-trust side FMP

の場合に比べて act_tsk によって対象タスクが起動するまでに長い時間がかかっている．これは SafeG-MP が一旦プロセッサ間割込みをハンドリングすることで生じるオーバヘッド増加分であると考えられる．

計測項目 (2) の場合は 1.95μs 付近に分布が見られ，計測項目 (3) の場合は 2.7 μs 付近に最頻値が見られる．計測項目 (1) の場合の実行時間分布と比べると，計測項目 (2) に関しては約 0.3μs の増加，計測項目 (3) に関しては，約 1μs の増加である．計測項目 (2) は RTOS 実行中に FIQ 入力があった場合 [実行パス (1)]，計測項目 (3) は汎用 OS 実行中に FIQ 入力があった場合 [実行パス (2)] であると考えられる．計測項目 (2) と (3) の増加分は SafeG-MP の FIQ ハンドリングのオーバヘッドと比較するとほぼその値が一致する．

次に，**図 7** にノントラスト側 FMP での計測結果を示す．計測項目 (4) が SafeG-MP を用いず，ノントラスト側 FMP を直接実行した場合の結果である．計測項目 (5) が SafeG-MP を用いた場合の計測結果である．この結果でも，SafeG-MP を使用した場合の方が対象タスクが起床するまでの時間が長くなっており，SafeG-MP の実行時間オーバヘッドによるものと考えられる．計測項目 (4) に関しては 1.8μs 付近に最頻値が見られ，計測項目 (5) に関しては，2.15μs 付近に最頻値が見られる．これらの最頻値を比較すると，SafeG-MP を用いた場合 (計測項目 (5) の場合)，約 0.35μs 実行時間が増加しているが，これは SafeG-MP の汎用 OS 実行中に IRQ 入力があった場合 [実行パス (3)] による増加分にほぼ等しい．

以上の結果より，FMP カーネルの act_tsk 実行時間は SafeG-MP を用いることで増加しているものの，その増加分は SafeG-MP のオーバヘッド分に抑えられていることが確認できる．

4.4 FMP カーネルの変更量

実装を行う上で FMP カーネル使用リソースの変更および TrustZone に関する設定が必要であった．FMP カーネル使用リソースとして，メモリ領域の変更および，デバイスの変更を行った．本研究で行った FMP カーネルへの変更点は，ターゲット依存部分への変更に限られており，コア部分 (割込みエントリ，ディスパッチャ内部のコードなど) への変更はなかった．今回の実装では，FMP カーネルを 2 つ同時に実行したが，ノントラスト状態で実行する OS としてより複雑な汎用 OS を実行する場合も使用デバイスの変更で対応が可能であると考えられる．

4.5 考　　察

以上の評価結果と 3.2 節で述べた設計方針を比較し，実装した SafeG-MP が設計方針を満たしているかを考察する．まず，2 つのマルチコアプロセッサ向け OS を同時実行しているため，設計方針 (a) を満たしている．今回は，ノントラスト状態で FMP カーネルを実行

Vol.2011-ARC-195 No.26
Vol.2011-OS-117 No.26
2011/4/14

した．Linux などの，より複雑な汎用 OS の実行は今後の課題とする．設計方針 (b) については，プログラムサイズ評価の部分で述べたように，SafeG-MP は想定している環境に対して小規模なサイズで実装できたといえる．設計方針 (c) を評価するために，SafeG-MP の実行時間オーバヘッドを計測した．SafeG-MP の実行時間オーバヘッドは，SafeG に比べて平均約 0.17μs 上昇している結果となった．実行時間オーバヘッドを増加させる原因となるのは，コア ID 比較を行うためのコードのみである．コア ID を調べる処理は，SafeG をマルチコア対応させるにあたって避けられない処理であり，必要最小限の実行時間オーバヘッド増加に抑えられたといえる．設計方針 (d) に関しては，3.3.1 節で述べたように，コア数を変えることで SafeG-MP で変化が生じるのは，退避領域のメモリ領域 (データ領域) 増加分のみである．したがって，設計方針 (d) を満たしているといえる．設計方針 (e) については，各コアで SafeG 同様の実行パスを持っており，RTOS 処理を優先できるので，満たせているといえる．設計方針 (f) は RTOS(トラスト側 FMP) の保護に関する方針であった．各 OS が異なるプロセッサの状態で実行されるので，トラスト側 FMP の使用するリソースへノントラスト側 FMP カーネルからアクセスすることができないように設定できる．設計方針 (g) に関しては，FMP カーネルへの変更点はターゲット依存部分への変更に限られており，コア部分 (割込みエントリ，ディスパッチャ内部のコードなど) への変更はなかったので満たせている．以上より，実装した SafeG-MP は設計方針をすべて満たしている．

5. おわりに

高信頼なデュアル OS 環境を構築する，SafeG による方式をマルチコアアーキテクチャに適用するために，マルチコア対応 SafeG(SafeG-MP) を実装した．この SafeG-MP を実機に搭載し，2 つのマルチコアプロセッサ向け OS を同時実行することができた．

現在の SafeG-MP による方式では，OS スケジューリングに関して OS のコア間同期処理を考慮していない．とくにスピンロック取得状態のコアで OS 切り替えが起きた場合，別のコアに配置されたプロセスのスピンロック取得時間が長くなる可能性がある．このスピンロック取得時間増加の問題はロックホルダープリエンプションと呼ばれており，仮想マシン方式においても，こうした問題が生じることが知られている[12]．今後はより複雑な汎用 OS への対応と，OS のコア間同期処理を考慮した OS スケジューリング手法の提案を行っていく予定である．

参 考 文 献

1) 保田信長，飯山真一，冨山宏之，高田広章，中島　浩：Linux と ITRON によるハイブリッド OS の設計と実装 (実時間処理, 組込システム及び一般)，情報処理学会研究報告. SLDM, [システム LSI 設計技術]， Vol.2004, No.33, pp.45–50 (2004-03-18).
2) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *SIGOPS Oper. Syst. Rev.*, Vol.37, pp.164–177 (2003).
3) The L4 microkernel family: “http://os.inf.tu-dresden.de/L4/”.
4) 中嶋健一郎，本田晋也，手嶋茂晴，高田広章：セキュリティ支援ハードウェアによるハイブリッド OS システムの高信頼化 (リアルタイムシステム)，情報処理学会研究報告. EMB, 組込みシステム， Vol.2008, No.116, pp.1–7 (2008-11-20).
5) Sangorrin, D., Honda, S. and Takada, H.: Dual Operating System Architecture for Real-Time Embedded Systems, *Proceedings of the 6th International Workshop on Operating Systems and Platforms for Embedded Real-Time Applications (OSPERT)*, Brussels, Belgium, pp.6–15 (2010).
6) ARM Ltd: ARM Security Technology. Building a Secure System using TrustZone Technology, PRD29-GENC-009492C.
7) ARM Ltd: ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Errata markup, DDI0406B_errata_2009_Q3.
8) ARM Ltd: Cortex-A9 Technical Reference Manual Revision: r2p2, DDI0388F.
9) ARM Ltd: PrimeCell Generic Interrupt Controller(PL390) Technical Reference Manual Revision: r0p0, DDI0416B.
10) TOPPERS 新世代カーネル統合仕様書Release1.2.0： “http://www.toppers.jp/docs/tech/ngki_spec-120.pdf”.
11) Sangorrin, D., Honda, S. and Takada, H.: Integrated Scheduling in a Real-Time Embedded Hypervisor, 情報処理学会 組込みシステム研究会第 18 回研究発表会，公立はこだて未来大学 (2010).
12) Uhlig, V., LeVasseur, J., Skoglund, E. and Dannowski, U.: Towards scalable multiprocessor virtual machines, *Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium - Volume 3*, Berkeley, CA, USA, USENIX Association, pp.4–4 (2004).