Vol.2018-ARC-230 No.6
Vol.2018-SLDM-183 No.6
Vol.2018-EMB-47 No.6
2018/3/7

メモリ保護対応RTOSにおける非信頼関数の実現手法

鴫原 一人[1,a)]　本田 晋也[1]　高田 広章[1]

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，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．これまで，プラットフォーム側のソフトウェアは，特権モードで動作させる前提となっていたが，機能安全に対応する場合，特権モードで動作させるソフトウェアの開発コストは高く，膨大な AUTOSAR プラットフォームのソフトウェアの開発コストが肥大化する問題がある．そこで，AUTOSAR の Release 4.2 では，特権モードで動作するアプリケーションから，ユーザモードへ切り替えてプラットフォームのサービスを呼び出すための非信頼関数が追加された．しかし，非信頼関数は，メモリ保護違反等が発生すると処理の途中であっても強制終了されることがある上，非信頼関数をネストして呼び出す場合，どのように使用するスタックを管理し，呼出し元のコンテキストへ復帰するかといった実装上の課題が多く，実現手法の例は報告されていない．本研究では，AUTOSAR における非信頼関数のユースケースを分析し，非信頼関数を実現する手法を提案する．

An Implementation Method of Non-Trusted Function on RTOS with Memory Protection

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

Abstract:
Memory protection function plays a crucial role in guaranteeing automotive software safety. AUTOSAR is a standardized software platform for automotive software. AUTOSAR specification prescribes memory protection function for automotive Real-Time OS, and this function prevents to propagate failures between applications. Although it have assumed that software of platform is executed in the supervisor mode, the development cost of software executed in the supervisor mode is high for functional safety, and the development of AUTOSAR software platform needs huge cost. In AUTOSAR Release 4.2, the non-trusted function to call services of platform in the user mode from applications in the supervisor mode is specified. However the non-trusted function have problems for implementation such as forced termination in executing and usage of stacks for non-trusted function in nested call. In this paper, we analyze use cases of non-trusted function in AUTOSAR, and propose an implementation method of non-trusted function.

1. はじめに

近年，自動車に搭載される電子制御ユニットの高度化・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．車載ソフトウェアの開発コスト削減のため，AUTOSAR という ECU 向けのソフトウェアプラットフォーム (以下，SPF) 仕様が策定・公開され，各国の自動車業界において標準となりつつある．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 SPF に含まれる各モジュールは BSW(Basic SoftWare) と呼ばれる．AUTOSAR 仕様の RTOS(以下，A-OS)[1] 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．

[1] 名古屋大学，APTJ 株式会社
Nagoya Uniersity and APTJ Co., Ltd.
[a)] shigihara@ertl.jp

AUTOSAR では，仕様のみ公開しており，設計や実装は，AUTOSAR SPF を開発，販売するソフトウェアベンダに委ねられているが，BSW は約 100 個存在するため，BSW の開発には膨大なコストを要する．さらに，機能安全に対する場合，アプリケーションを高い安全レベルで開発すると，BSW も同じ安全レベルで開発する必要があり，開発コストはさらに肥大化する．そこで，AUTOSAR の Release 4.2 では，BSW を低い安全レベルで開発しても，そのまま使用可能とするためのコンセプトが導入された [2]．具体的には，特権モードで動作するアプリケーションから，メモリアクセスを制限できるユーザモードへ切り替えて BSW のサービスを呼び出すための非信頼関数が，A-OS 仕様に追加された．非信頼関数は，呼び出した関数の処理の途中でメモリ保護違反等が発生すると強制終了されることがある上，非信頼関数をネストして呼び出す場合，どのように使用するスタックを管理するかといった実装上の課題も多く，実現手法の例は報告されていない．

本論文では，非信頼関数の 5 つのユースケースを特定

Vol.2018-ARC-230 No.6
Vol.2018-SLDM-183 No.6
Vol.2018-EMB-47 No.6
2018/3/7

し，それぞれのユースケースを実現する上での 2 つの課題を明確にした．また，明確にした課題を解決した実現手法を考案し，実際に非信頼関数のユースケースに対応可能であることを確認した．本論文のコントリビューションは，AUTOSAR における非信頼関数のユースケースを明確化し，非信頼関数実行に使用するスタックの管理方法，強制終了時の復帰方法に対する問題を解決した非信頼関数実現手法を提案することである．

2. A-OS 仕様

本節では，A-OS 仕様において，メモリ保護を実現するための機能である OS アプリケーションおよび非信頼関数について説明する．

2.1 OS アプリケーション

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 毎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，特権モードで動作し，一切のアクセス制限を受けず，すべてのメモリ領域にアクセスできる．信頼 OSAP は，非信頼 OSAP に対して，1 つ以上の信頼関数を提供することができる．信頼関数とは，ユーザ定義の処理を，非信頼 OSAP に所属する処理単位から特権モードで実行するための機能であり，CallTrustedFunction というシステムサービスを呼び出すことにより実行する．本論文では，システムサービスを含め，ユーザモードから特権モードへ切り替えて呼び出す関数を総称して信頼関数と呼ぶ．

非信頼 OSAP に所属した処理単位は，ユーザモードで動作し，所属した OSAP に許可されたメモリ領域にのみアクセスできる．非信頼 OSAP に所属した処理単位が許可されていないメモリ領域にアクセスした場合は，メモリ保護違反として，対象の非信頼 OSAP の強制終了や再起動，もしくは OS シャットダウンを行うことが可能である．また，TerminateApplication というシステムサービスにより，任意のタイミングで非信頼 OSAP を強制終了することも可能である．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．

機能安全に対応する場合，信頼 OSAP に所属させるソフトウェアは，同じシステム上で最も高い安全レベルで開発する必要があり，一般的に信頼 OSAP 向けソフトウェアの開発コストは高い．また，Release 4.2 では，BSW が約 100 個存在するため，すべての BSW を信頼 OSAP に所属させるには，BSW の開発コストが肥大化する問題がある．これに対し，BSW の開発コストを抑えるため，Release 4.2 では，BSW を非信頼 OSAP に所属させるコンセプトが導入された．これに伴い，アプリケーションから非信頼 OSAP に所属する BSW が提供するサービスを呼び出す必要があるので，A-OS 仕様に非信頼関数が導入された．具体的には，OSAP のコンフィギュレーションパラメータに OsTrustedApplicationWithProtection が新設された．

OsTrustedApplicationWithProtection は，boolean 型で設定するパラメータであり，信頼 OSAP に対してのみ true を設定できる．OsTrustedApplicationWithProtection に true を設定された信頼 OSAP(以下，保護付き信頼 OSAP) は，非信頼 OSAP と同様にメモリ領域に対するアクセス制限を受けることになり，振舞いとしては非信頼 OSAP と同等となる．つまり，保護付き信頼 OSAP と非信頼 OSAP との違いは，信頼関数の提供可否のみであるので，本論文では，保護付き信頼 OSAP が提供する信頼関数を非信頼関数と呼び，保護付き信頼 OSAP と非信頼 OSAP を区別せずに非信頼 OSAP と呼ぶ．アプリケーションから使用する BSW を非信頼 OSAP に所属させる場合，BSW を非信頼 OSAP に所属させ，BSW が提供するサービスを非信頼関数を用いて呼び出すことで，信頼 OSAP に所属するアプリケーションを保護する．

本論文では，信頼 OSAP に所属するタスクを信頼タスク，非信頼 OSAP に所属するタスクを非信頼タスクと呼ぶ．

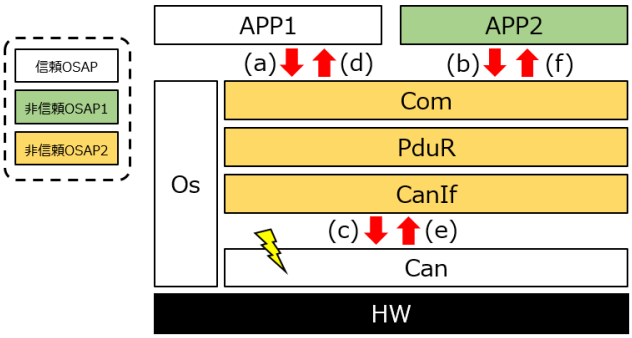

図 1 CAN 通信スタックにおけるユースケースの例

2.2 非信頼関数

非信頼関数は，信頼関数とは逆に，ユーザ定義の処理をユーザモードで実行するための機能であり，非信頼 OSAP が他の OSAP に対して提供する．非信頼関数も，信頼関数と同様に，CallTrustedFunction によって呼び出す．以下に，AUTOSAR において BSW を非信頼 OSAP に所属させた場合の非信頼関数のユースケースについて説明する．

図 1 に，CAN 通信スタックの例を示す [*1]．アプリケーションから送信されたデータは，Com，PduR，CanIf という BSW を通じて，Can という CAN 通信用のドライバモジュールによって，ハードウェア (以下，HW) へ CAN メッセージが送信される．また，HW で受信した CAN メッセージは，逆の順序で，アプリケーションへ渡される．APP1，および Can は信頼 OSAP に，APP2 は非信頼 OSAP1 に，Com，PduR，CanIf は非信頼 OSAP2 に所属するものとする．Can は割込みによる受信などを行うことから ISR による実装が必要になるので，信頼 OSAP としている [*2]．Com，PduR，CanIf は，同一非信頼 OSAP に所属するため，これらのモジュール間のサービス呼出しは，直接関数呼出しするものとする．

図 1 において，非信頼関数の実装上，考慮するべきユースケースは以下の 5 つ存在する．

(1) 信頼 OSAP からの非信頼関数呼出し (a)
APP1 から Com が提供するサービスを呼び出すため，信頼タスクから非信頼関数を呼び出せる必要がある．

(2) 他の非信頼 OSAP からの非信頼関数呼出し (b)

[*1] 実際にはアプリケーションと BSW の間に Rte というモジュールが入るが，簡単のため省略している
[*2] 本論文では，ISR は信頼 OSAP にのみ所属する前提としている

APP2 から Com が提供するサービスを呼び出すため，非信頼タスクから異なる非信頼 OSAP が提供する非信頼関数を呼び出せる必要がある．

(3) 非信頼関数からの信頼関数呼出し (c)(d)
CanIf から Can が提供するサービスを呼び出すため，また，Com が APP1 のコールバックを呼び出すため，非信頼関数実行中に，他の信頼 OSAP が提供する信頼関数を呼び出せる必要がある．つまり，非信頼関数から，信頼関数を呼び出せる必要がある．

(4) 非信頼関数のリエントラント呼出し (a)(b)
APP1 と APP2 の優先度や実行周期が異なる場合，例えば，APP2 で Com のサービスを実行中に，APP1 が割込み，APP1 からも Com のサービスを呼び出すことがある．したがって，非信頼関数はリエントラントに呼出し可能である必要がある．

(5) 非信頼関数のネスト呼出し (e)(f)
Can の受信割込みで起動した ISR から CanIf が提供するサービスを呼び出し，さらに Com が APP2 のコールバックを呼び出すため，異なる非信頼 OSAP に所属する非信頼関数はネストして呼び出せる必要がある．

なお，非信頼関数実行中に，非信頼関数が所属する非信頼 OSAP が強制終了された場合，実行中の非信頼関数も中断され，呼出し元の処理単位へリターンする．

3. 非信頼関数の実現

A-OS 仕様では，MPU(Memory Protection Unit) 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 を使った具体的なメモリ保護の実現方法や，コンフィギュレーションの方法は規定されていない．そこで，本研究では，A-OS 仕様をベースとし，メモリ保護機能に対応したオープンソースの RTOS である TOPPERS/ATK2(以下，ATK2)[3] の設計を元に，非信頼関数の検討を行った．ATK2 は，Release 4.0 に対応しており，非信頼関数は実装されていない．

非信頼関数を実現する手法として，UNIX などの POSIX 準拠の OS で提供されるシグナルハンドラのように，OS がコンテキストの切替えを行い，ユーザモードでユーザ定義の関数を呼び出す方法が考えられる．しかし，RTOS においては，コンテキストの切替えはタスクディスパッチ同様，オーバーヘッドが大きいため，非信頼関数は，同期的な関数呼出しとする必要がある．

本章では，ATK2 のスタック設計について説明し，非信頼関数を実装する上での以下の 2 つの課題について述べる．

- 非信頼関数呼出しに使用するスタックをどのように確保するか
- 非信頼関数実行中に強制終了された場合，どのように呼出し元のコンテキストへ復帰するか

3.1 ATK2 のスタック設計

信頼タスクは，常に特権モードで動作するため，タスク動作中に使用する唯一のスタック (以下，信頼タスク用スタック) を用意する．信頼タスクから信頼関数を呼び出す場合も，信頼タスク用スタックの続きを使用する．非信頼タスクは，ユーザモードでタスクの処理を実行する際に使用するスタック (以下，ユーザスタック) と，信頼関数など特権モードで動作する際に使用するスタック (以下，システムスタック) を用意する．非信頼タスク実行中に，ユーザモードから特権モードへ切り替える際，OS 内部処理で，trap 命令等で特権モードへ切り替えた上で，スタックポインタをシステムスタックへ切り替える．

非信頼タスクでもスタック領域を 1 つだけ用意する方法も考えられるが，特権モードでユーザスタックの続きを使用すると，ユーザモードで動作中にスタックポインタが不正な位置 (スタック範囲外等) となった状態で特権モードに切り替えた場合，不正なメモリ領域を破壊してしまう恐れがある．特権モードへの切替え時，スタックポインタの位置をチェックするより，使用するスタックを切り替えたほうが，実装を簡潔にできるため，非信頼タスクには，システムスタックも用意している．各スタックのサイズは，コンフィギュレーション時に，タスク毎に指定することでユーザが静的に決定する．

なお，ATK2 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 カーネルの設計をベースとしているため，詳細な設計については，文献 [5][6] を参照のこと．

3.2 実装上の課題

3.2.1 非信頼関数呼出しに使用するスタック

2.2 節で述べた各ユースケースにおいて，非信頼関数呼出しに使用するスタック (以下，非信頼関数実行用スタック) について検討する．

ユースケース (1) に対応するには，信頼タスクにも，非信頼関数実行用スタックとして，システムスタックとは別にユーザスタックを用意すればよい．しかし，ユースケース (2) では，非信頼タスクに対し，タスク自身を実行するためのユーザスタックに加え，別の非信頼 OSAP が提供する非信頼関数を実行するための非信頼関数実行用スタックも必要となる

ユースケース (3) のように，非信頼関数から信頼関数を呼び出す場合，3.1 節で述べたように，非信頼関数実行用スタックとは異なるスタックへ切り替える必要がある．また，ユースケース (4) のように，非信頼関数がリエントラントに呼び出される場合，呼出し元のタスク毎に非信頼関数実行用スタックを用意する必要がある．さらに，ユースケース (5) のように，非信頼関数をネストして呼び出す場合，呼び出される非信頼関数毎にスタックを用意し，スタックを切り替えながら呼び出す必要がある．

これらのスタックを，非信頼関数を呼び出すタスク側にすべて用意しておくと，多くのスタック領域が必要となり，RAM を大量に消費してしまうことが懸念される．また，タスクディスパッチ時に，どのスタックをどの位置まで使用していたかを保持する方法が複雑になることも懸念される．

3.2.2 非信頼関数実行中の強制終了処理

本節では，各ユースケースにおいて，呼出し中の非信頼関数が所属する非信頼 OSAP が強制終了 (再起動の意味も含む) された場合の処理について検討する．

ユースケース (1) と (2) では，非信頼関数呼出し中に強制終了された場合に備え，非信頼関数呼出し時のコンテキスト情報 (プログラムカウンタやスタックポインタ) を，呼び出したタスク側に保持しておき，強制終了時に復帰すればよい．ユースケース (3) は，信頼関数呼出しで使用するスタックも破棄して，(1)(2) と同様に，非信頼関数呼出し時のコンテキストへ復帰すればよい．ユースケース (4) では，強制終了された非信頼 OSAP に所属する非信頼関数

Vol.2018-ARC-230 No.6
Vol.2018-SLDM-183 No.6
Vol.2018-EMB-47 No.6
2018/3/7

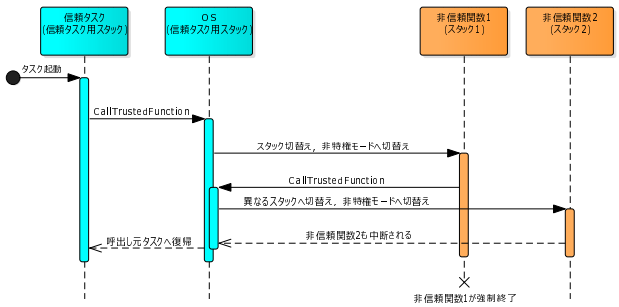

図 2　非信頼関数ネスト呼出し中の強制終了シーケンスの例

を呼出し中のすべてのタスクを，非信頼関数呼出し時のコンテキストへ復帰すればよい．このように，ユースケース (1)〜(4) までは，非信頼関数呼出し時に，非信頼関数呼出し時のコンテキスト情報を，非信頼関数を呼び出すタスク側に保持させることで，実現は可能であると考えられる．

しかし，ユースケース (5) の場合，非信頼関数をネストして呼び出している間に，途中で呼び出している非信頼関数が，ISR から TerminateApplication により強制終了される可能性があるため，(1)〜(4) と同様には実現できない．非信頼関数ネスト呼出し中に強制終了されるシーケンスの例を図 2 に示す．図 2 では，まず信頼タスクから非信頼関数 1 を呼び出し，非信頼関数 1 から非信頼関数 2 を呼び出す．そして，非信頼関数 2 実行中に割込みによって起動した ISR が TerminateApplication を呼び出し，非信頼関数 1 が所属する OSAP を強制終了することにより，非信頼関数 1 が強制終了される [*3]．信頼タスクは，呼出し中の非信頼関数 1 が強制終了されたため，非信頼関数 2 を含めた非信頼関数の呼出しを中断し，強制終了された非信頼関数 1 を呼び出した時点まで復帰する必要がある．

つまり，非信頼関数をネストして呼び出す場合，呼出し中のどの非信頼関数が強制終了されたとしても，その非信頼関数を呼び出した時点まで，呼出し元タスクのコンテキストを復帰できる必要があり，ネストした数だけ復帰用のコンテキスト情報を保持する必要がある．

4. 提案手法

3 章で述べた通り，非信頼関数の実装には 2 つの大きな課題がある．本章では，これら 2 つの問題を解決し，非信頼関数を実現する手法を提案する．

4.1 非信頼関数実行用スタックの確保

タスクが使用するスタックサイズは，タスク毎に異なるため，一般的に RTOS では，タスクに対してスタックサイズを指定し，タスク毎にスタック領域を確保する．メモリ保護対応 RTOS において，非信頼タスクが信頼関数呼出しで，特権モードに切り替わった際に使用するシステムスタックについても，同様にタスク毎に保持すればよい．非信頼関数を呼び出すタスクに対しても，タスク毎に非信頼関数実行用スタックを用意する手法が考えられるが，以下のデメリットがある．

例えば，異なる非信頼 OSAP に所属する 2 つの非信頼関数 A，B があり，非信頼関数 A から B をネストして呼び出すとする．また，非信頼関数 A を呼び出すタスクが 3 つあるとし，非信頼関数 A をリエントラントには呼び出さないものとする．タスク側に非信頼関数実行用スタックを持たせる場合，各タスクに非信頼関数 A と B を呼び出すためのスタックを持たせる必要があるため，計 6 つの非信頼関数実行用スタックが必要となる．

そこで，非信頼関数実行用スタックは，タスクではなく，非信頼関数側に用意する手法を検討する．ATK2 では，信頼関数が使用するスタックサイズを指定するパラメータ OsTrustedFunctionStackSize があるので，これを非信頼関数が使用するスタックサイズとしても使用する．本パラメータに加え，非信頼関数では，リエントラントに呼び出す可能性のあるタスク数を指定するパラメータ OsReentrantNum を新設する．非信頼関数が定義された場合は，OsTrustedFunctionStackSize で指定されたサイズのスタック領域を OsReentrantNum の数だけ確保するわけである．前述の例では，リエントラントに呼び出すことはないため，非信頼関数 A と B を呼び出すためのスタックは 1 つずつ計 2 つのみ用意すればよく，スタック領域を節約することができる．

非信頼関数をリエントラントに呼び出す場合は，どちらの手法でも用意するスタック領域の数は変わらないが，非信頼関数がリカーシブル呼出しを行う実装では，以下の違いがある．タスク側にスタック領域を用意する場合，リカーシブル呼出しで使用するスタック領域も用意する必要があり，非信頼関数呼出し時，リカーシブル呼出しかどうかを判定して，スタック領域を切り替える処理が必要になってしまう．一方，非信頼関数側にスタック領域を用意する場合は，リカーシブルに呼び出される回数も含めて，OsReentrantNum を設定すればよく，リカーシブルに非信頼関数を呼び出す場合の処理を区別する必要はない．

非信頼関数呼出し時，OsReentrantNum の数だけ確保されたスタック領域のうち，どのスタック領域を使用するかは，以下のように決定する．1 つの非信頼関数に対し，使用中のスタックを管理する変数を用意する．OS 起動時に，すべてのタスクを未使用状態として初期化しておき，非信頼関数呼出し時に空いているスタック領域を選択し，変数を使用中へ更新する．空いているスタック領域が存在しなければ，CallTrustedFunction はエラーとしてリターンする．非信頼関数を呼び出すタスク側には，どの非信頼関数を呼び出しているかと，どのスタック領域を使用しているかの情報を保持しておくことで，タスクディスパッチ時に使用するスタックを特定する．

なお，実装において OsReentrantNum を 32 以下に制限することで，使用中スタックを管理する変数は，32 ビットの 1 変数とすることができ，空いているスタックを調べるには，ビットサーチ命令 [*4] を使用することで高速に処理することが可能である．

4.2 非信頼関数の実行

3.2.2 節で述べた課題を解決するため，以下を検討する必要がある．

(1) 非信頼関数を呼び出しているタスクをどのように管理するか
(2) 非信頼関数からの信頼関数呼出し時，どのスタックを使用するか
(3) 非信頼関数強制終了時，どのように呼出し元タスクへ

[*3] 図 2 では ISR による強制終了の部分は省略している

[*4] ターゲットとするマイコンがサポートしている必要がある

Vol.2018-ARC-230 No.6
Vol.2018-SLDM-183 No.6
Vol.2018-EMB-47 No.6
2018/3/7

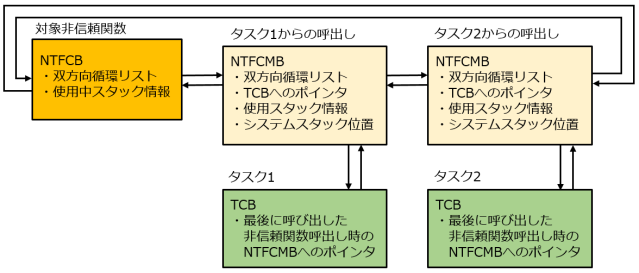

図 3 NTFCMB による実装イメージ

復帰するか

(1) は，OsReentrantNum によりリエントラントに呼び出される数が分かっているため，呼出し中タスクの情報を格納する配列データを，OsReentrantNum のサイズで用意する方法が考えられる．しかし，タスクが非信頼関数実行中に待ち状態に遷移する可能性があるため，リエントラントに非信頼関数を呼び出した際，LIFO の順で非信頼関数の呼出しが終了するとは限らない．したがって，配列データから呼出しを終了したタスクの情報を抜き出す処理の実行時間が一定にならず，状況によっては CallTrustedFunction の応答時間が長くなってしまう．

(2) は，呼出し元タスクが，非信頼関数を呼び出した際に使用していたシステムスタックの続きを使用すればよいが，非信頼関数実行中に，使用可能なシステムスタックの位置を参照できる必要がある．また，非信頼関数からのリターン時，強制終了時に，非信頼関数呼出し時のシステムスタックの位置に復帰する必要がある．

(3) を実現するためには，非信頼関数呼出し時のコンテキスト情報を，非信頼関数を呼び出す度に保持していく必要がある．非信頼関数をネストして呼び出した場合，非信頼関数からのリターン時，直前に呼び出した非信頼関数の所属する OSAP のメモリ保護の状態へ戻す必要があるため，どの非信頼関数を呼び出していたかも，保持しておく必要がある．なお，非信頼関数は強制終了される可能性があるので，呼出し先退避レジスタ [*5] が破壊されたまま呼出し元に返る可能性がある．したがって，呼出し先退避レジスタもコンテキスト情報として保持する必要がある．

これらの要件を満たす実装として，呼出し中のタスクを，以下の情報を持つ構造体の双方向循環リストで管理する手法を検討する．本構造体を非信頼関数呼出し管理ブロック (NTFCMB) と呼ぶ．

- 双方向循環リスト構造体
- 非信頼関数呼出し元タスクを示す情報
- 非信頼関数呼出し時に使用したスタックを示す情報
- 非信頼関数呼出し時のシステムスタックの位置

非信頼関数毎に非信頼関数の情報を管理するデータブロック (NTFCB) においても，NTFCMB と接続するための双方向循環リスト構造体を持ち，非信頼関数が呼び出される度に，リストへ NTFCMB を繋いでいく．タスク毎にタスクの情報を管理するデータブロック (TCB) には，最後に呼び出した非信頼関数の NTFCMB へのポインタを持つ．NTFCB と TCB は，非信頼関数とタスクの数に応じて，RAM 上に静的に用意する．NTFCMB を用いた実装イメージを図 3 に示す．

4.2.1 非信頼関数の呼出し手順

タスクから非信頼関数を呼び出す手順は以下となる．

CallTrustedFunction 呼出し時，呼出し元タスクのシステムスタック上に NTFCMB を確保し，対象非信頼関数の NTFCB の双方向循環リストに繋ぐ．非信頼関数の強制終了時に，NTFCMB から呼出し元タスクを識別するために，呼出し元タスクの TCB へのポインタを NTFCMB へ設定する．使用する非信頼関数実行用スタックを決定し，NTFCMB へ使用するスタック情報を設定する．NTFCB の使用中スタック情報に含まれる対象スタックの状態を，使用中へ更新する．非信頼関数呼出し時のタスクの戻り番地を，非信頼関数から復帰するための関数へ設定する．以下の情報 (以下，NTF コンテキスト情報) を呼出し元タスクのシステムスタックへ退避し，退避後のシステムスタックの位置を，NTFCMB へ設定する．

- 呼出し先退避レジスタ一式
- 非信頼関数呼出し時点で TCB に設定されていた NTFCMB へのポインタ
- 非信頼関数呼出し時点のシステムスタックの位置

4.2.2 非信頼関数からのリターン手順

非信頼関数呼出しから，タスクへリターンする場合，非信頼関数から復帰するための関数により以下の順に実行する．

NTFCMB に退避していたシステムスタックの位置をスタックポインタに設定し，NTF コンテキスト情報を復帰する．別の非信頼関数を呼出し中であった場合は，復帰した元の NTFCMB へのポインタから，使用するスタック情報を取り出し，スタックポインタに設定する．NTFCB の使用中スタック情報に含まれる対象スタックの状態を，未使用へ更新する．呼出し元タスクのシステムスタック上に確保していた NTFCMB を，対象 NTFCB の双方向循環リストから削除する．双方向循環リストのため，最後に呼び出したタスクでなくても，容易に削除が可能である．

4.2.3 非信頼関数の非信頼関数強制終了手順

非信頼関数強制終了時の手順は以下となる．

強制終了された非信頼関数の NTFCB の双方向循環リストに繋がれた NTFCMB を辿り，各タスクを順に処理していく．強制終了された OSAP に複数の非信頼関数が所属する場合は，すべての非信頼関数に対して同じ処理を行う．

まず，TCB で管理している NTFCMB へのポインタを，最初に呼び出された非信頼関数へ更新する．具体的には，タスク毎に，NTFCMB に退避したシステムスタック位置が最もスタックの初期位置に近い非信頼関数の NTFCMB へのポインタを設定する．これは，あるタスクが，同一非信頼 OSAP に所属する 2 つの非信頼関数をネスト呼出ししていた場合に，どちらを先に呼び出したかを判定するための対処である．非信頼関数をネストして呼び出す毎に，NTFCMB に退避するシステムスタック位置は，スタックを使用する方向に値が変化していくため，最もスタックの初期位置に近い非信頼関数の呼出しを復帰場所として採用すればよい．非信頼関数をリカーシブルに呼び出している場合でも，本処理で対応可能である．

復帰する非信頼関数の呼出しを特定した後は，正常に非信頼関数からリターンする場合同様，システムスタックに退避していたコンテキスト情報を復帰することで，最初に非信頼関数を呼び出した時点へ復帰することができる．呼び出した非信頼関数が強制終了された場合に，CallTrustedFunction で特別なエラーを返すようにする場合は，強制終了処理にて，呼出し中タスクの戻り番地を変更し，正常処理に，返り値を変更する処理を追加すればよい．

[*5] callee saved register の意

表 1　非信頼関数対応前後の ROM/RAM 使用量 (byte) の変化

対象データ	対応前	対応後
タスク毎の RAM データ (TCB)	45	49
OSAP 毎の ROM データ	21	27
非信頼関数毎の ROM データ	0	20 + (8 * n)
非信頼関数毎の RAM データ (NTFCB)	0	12

n:OsReentrantNum

つまり，呼出し元タスクのシステムスタック上に，非信頼関数の呼出し情報を順に上乗せしていき，リターン時および強制終了時は，対象非信頼関数の NTFCMB に退避したシステムスタック位置から，NTF コンテキスト情報を取り出すことで，どの非信頼関数呼出しの状態へも復帰を可能とするわけである．なお，本手法を用いる場合，システムスタックは多くの情報を退避したり，非信頼関数からの信頼関数呼出しでも使用するため，これらの使用量を見積もった上でスタックサイズを設定することに注意が必要である．

5. 評価

我々は，4 章で述べた手法を，ATK2 と同等の設計を持つ商用 AUTOSAR OS に対して実装を行い，2.2 節で述べたユースケースに対応できることを確認した．対象としたマイコンは，ビットサーチ命令をサポートしていたため，4.1 節で述べたように，OsReentrantNum を 32 以下に制限した．本章では，非信頼関数に対応する前後での，ROM/RAM 使用量および，各処理時間の変化について評価する．

表 1 に，各データの ROM/RAM 使用量の変化を示す．タスク毎に必要となる RAM データの増加は，図 3 に示した最後に呼び出した非信頼関数呼出し時の NTFCMB へのポインタの 4byte のみであった．非信頼関数を追加する毎に RAM データが 12byte 増加するが，これは NTFCB の双方向循環リスト 8byte と，使用中スタック情報 4byte であり，必要最低限の RAM データである．ROM データは，非信頼関数の数および OsReentrantNum に応じて大きくなっていくが，近年のマイコンの ROM サイズは大きいため，問題にならないと考えられる．なお，表 1 の値は，構造体のパディングを考慮せずメンバ変数のサイズ合計値を記載しているため，実際はパディングに収まり，増加量が少なくなるものもあった．

既存の処理に対して，非信頼関数に対応することにより処理時間に変化が発生したのは，以下の 4 つの処理である．

- CallTrustedFunction
- タスクディスパッチ処理
- 非信頼 OSAP からの信頼関数呼出し処理
- 非信頼 OSAP 強制終了処理

(1) は，CallTrustedFunction 内で，呼出し先が，信頼関数か非信頼関数かを判別する処理を 2 箇所追加するのみであったため，既存の信頼関数呼出し時間への影響は少なかった．(2) は，タスクディスパッチ時に，切替え先のタスクが非信頼関数実行中かどうかを判定する分岐が増えるのみであり，非信頼関数呼出しを行わない場合の処理に対しては，3 命令増加するのみであった．(3) は，信頼関数を呼び出したタスクが，非信頼関数実行中かどうかを判定する分岐が増えるのみであり，既存処理に対しては，やはり 3 命令増加するのみであった．(4) は，強制終了された非信頼 OSAP に，実行中の非信頼関数がいくつ存在するかによって，処理時間が増加していくため，増加時間は一定とはならない．しかし，現実的には，非信頼 OSAP 強制終了時に，多くの非信頼関数がリエントラントに実行状態になっている状況は考えにくいため，(4) の増加が問題になることは考えにくい．

以上から，正常処理である (1)〜(3) への影響は最小限のオーバーヘッド増加に留めることができていると言える．また，(4) に対しては，双方向循環リストを使用しているため，最悪応答時間を見積もることは容易である．

6. 関連研究

A-OS 仕様に対し，独自に非信頼関数を拡張した事例は存在するが，実装言語は AspectC++であり，非信頼関数で使用するスタックの確保方法や非信頼関数が所属する OSAP が強制終了された際の実装については言及されていない [7]．

A-OS 仕様に準拠した商用 RTOS は複数存在するが，非信頼関数の実装やそれに関連する文献が公表されている事例は発見できなかった．ユーザーズマニュアルが公開されているルネサスエレクトロニクス社製の RV850 では，OsTrustedApplicationWithProtection は非サポートとされている [8]．

以上のように，非信頼関数の具体的な実装に関する事例は発見できなかった．

7. まとめ

本論文では，A-OS 仕様で規定された非信頼関数の実現手法を提案し，現実的なオーバーヘッド増加で，実現可能であることを確認した．今後の研究として，非信頼関数を使用した複雑なシナリオを含む網羅的なテストの実施や，現状サポート対象としていない，ISR からの非信頼関数呼出しについても検討している．

参考文献

[1] Specification of Operating System (AUTOSAR Release 4.2.2): available from ⟨http://www.autosar.org/⟩
[2] Guide to BSW Distribution (AUTOSAR Release 4.2.2): available from ⟨http://www.autosar.org/⟩
[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] 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).
[6] 中嶋 健一郎, 山田 真大, 長尾 卓哉, 山崎 二三雄, 武井 千春, 本田 晋也, 高田 広章：ARMv6 アーキテクチャを用いたメモリ保護 RTOS のユーザスタック保護の設計と評価, 情報処理学会研究報告, Vol.2009-EMB-014, No.8, (2009).
[7] Stilkerich, Michael and Lohmann, Daniel and Schröder-Preikschat, Wolfgang: *Memory Protection at Option*, Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety, (2010)
[8] RV850 リアルタイム・オペレーティング・システム ユーザーズマニュアル 機能編 Rev.1.05：入手先 ⟨https://www.renesas.com/ja-jp/doc/products/tool/doc/007/r20ut2768jj0105-rv850.pdf⟩