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

車載システム向けのOS管理外割込みパーティショニング機構

本田 晋也[1]　岡部 亮[2]　攝津 敦[2]

概要：近年，車載システムの複雑化・高性能化が進んでいる．複雑化に対応するため，AUTOSAR プラットフォームの使用が一般化しており，更に機能安全への対応のため，RTOS の保護機構によるパーティショニングを行うことも多い．一方，高性能化として高速な割込み応答が求められているが，RTOS が提供する通常の割込み機構では要件を満たさないという問題がある．そこで，RTOS 実行中も割込みを受付可能な，OS 管理外の割込みと呼ばれる機構により，実現する手法が用いられている．しかしながら，OS 管理外の割込みは，メモリ保護や時間保護が有効でないという問題がある．本研究では，RTOS と独立した OS 管理外割込みの保護機構を提案し，提案機構を実装して，実行オーバヘッドや RTOS の変更量等を評価した．

A RTOS Unmanaged Interrupt Partitioning mechanism for Automotive System

Shinya Honda[1]　Ryo Okabe[2]　Atsushi Settsu[2]

Abstract: In recent years, automotive systems have become more complicated and high performance. Using AUTOSAR platform is becoming common in order to manage the complexity. In addition, the space and time partitioning of a RTOS(RTOS partitioning) is used to confirm to the function safety standards. Although fast interrupt response time is required, the normal interrupt handler provided by RTOS can't satisfy the requirement. It can be realized by a mechanism called an unmanaged interrupt called ISR1 that is received even during RTOS execution. However, the RTOS partitioning is not effective for ISR1. In this paper, we propose an RTOS independent interrupt handler partitioning framework that acts as ISR1. We implemented the proposed mechanism and evaluated the execution overhead and the change amount.

1. はじめに

近年，車載システムの高機能化に伴い，ECU 上で実行される車載アプリケーションの複雑化・大規模化が進んでいる．車載アプリケーションは複数の機能により構成されており，それぞれが異なる安全度水準（ASIL）を持つ．そのため，現状の車載アプリケーションは複数の安全度水準の機能が混在するミックスドクリティカルシステムとなっている．ミックスドクリティカルシステムをメモリ保護等や時間保護といった保護機構を使用せずに実現すると，全機能のうち最も高い安全度水準に合わせて全ての機能を開発する必要があり，開発工数が大きくなるという問題がある．

複雑化・大規模化する車載アプリケーション開発を効率化するため，ソフトウェアプラットホーム仕様である AUTOSAR[1] が提案され，広く用いられている．AUTOSAR 仕様のソフトウェアプラットホームは，それ自身が大規模なソフトウェアであるため，一般的には外部ベンダから購入して使用する．そのため，機能安全の観点からユーザ側でのソースコードの変更は許容されない．

前述の開発工数の増大を解決する方法として，AUTOSAR では，OS 仕様として，時間保護仕様やメモリ保護仕様が策定され公開されている [2]．メモリ保護仕様では，タスクや割込みハンドラをメモリ保護を有効にして実行することができ，許可されないメモリへのアクセスを検出することが可能である．しかしながら，メモリ保護を用いるとタスクや割込みハンドラの起動オーバヘッドや応答時間が悪化することが分かっている [3]．

車載アプリケーションによっては，モータ制御など数十マイクロ秒単位の周期と高速な割込み応答が求められる機能（短周期処理）が存在する．現状のシステムでは，短周期処理を OS の管理外割込み（ISR1）として実現している．ISR1 は，AUTOSAR-OS 仕様で規定されている割込みであり，OS 実行中であっても割込みを受け付ける機構であり，高速な割込み応答を実現することが可能である．一方，ISR1 からは OS の API は発行できず，各種保護も有効とならない．そのため，低い安全度水準が割り当てられた短

[1] 名古屋大学 大学院情報学研究科
[2] 三菱電機株式会社 情報技術総合研究所

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

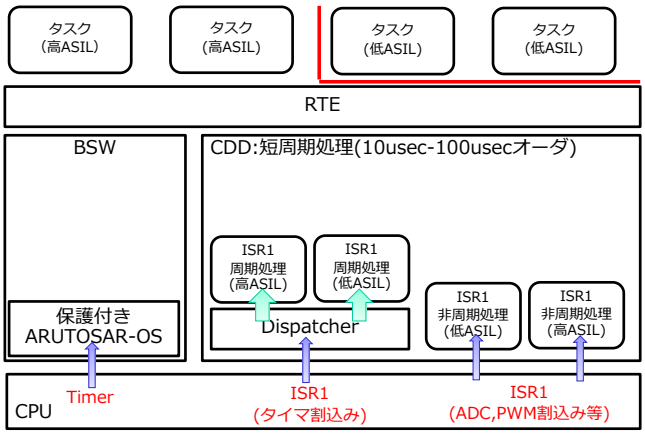

図 1　対象とする車載アプリケーションの構成
Fig. 1　Target Automotive System

周期処理であっても，保護が有効でないため，高い安全度水準で開発する必要があり，開発工数が増大するという問題がある．

この問題を解決する方法としては，低い安全度水準が割り当てられた短周期処理を保護が有効な割込みハンドラとして実行する方法があるが，この方法では前述のように保護のオーバヘッドにより，高速な割込み応答時間を実現できないという問題がある．別の方法として，車載システム向けの仮想化機構を用いる手法 [4], [5], [6], [7] もあるが，既存のシステムから大きく構成が変わってしまうため，導入が困難であるという問題がある．

本研究では，車載システムにおいて，高速な割込み応答時間と保護が必要な処理を実現するための保護フレームワークを提案する．提案機構は RTOS とは独立しており，ISR1 として実行され，各種保護機構を有効にして対象の処理を呼び出す．提案手法の機能の設計を行い，車載システム向けのプロセッサを対象に実装する．そして，AUTOSAR-OS と組み合わせることにより，提案手法を評価する．

本研究のコントリビューションは次の通りである．

- 車載システム向けの RTOS と独立に動作する，高速割込み応答と保護を両立する保護フレームワークの提案．
- 保護フレームワークにより実現可能な割込み応答時間や実行オーバヘッドを提示．

2. 対象とする車載アプリケーション

本研究の対象とする車載アプリケーションの構成を図 1 に示す．ソフトウェアプラットフォームとしては，AUTOSAR を用いている．システムは，タスクと割込みハンドラで構成されており，短周期処理は，後述する割り込みハンドラの一種である ISR1 として実行される．各処理は求められる安全度水準が異なり，タスクの場合は，後述する保護機構によりパーティショニングされている．

2.1 AUTOSAR

AUTOSAR は，車載ソフトウェアのポータビリティを向上させる目的で定められた業界標準仕様である．RTOS を中心とした BSW と呼ばれる複数のソフトウェアモジュールが定義されている．AUTOSAR で定められている RTOS を AUTOSAR-OS と呼ぶ．AUTOSAR-OS には複数のバリエーションがあり，保護機構を持つバリエーションを保護付き AUTOSAR-OS と呼ぶ．

表 1　AUTOSAR OS の割込みハンドラの種類
Table 1　ISR Types on AUTOSAR OS

名称	OSAPI 呼び出し	OS 実行中の割込み	保護	起動オーバヘッド
ISR1	不可能	可能	なし	小
ISR2	可能	不可能	なし	中
NT-ISR2	可能	不可能	あり	大

高い信頼性が要求される車載システムの開発においては，機能安全プロセスに準拠した開発が必要となる．この際，OS としては，保護付き AUTOSAR-OS を用いる必要がある．更に，保護付き AUTOSAR-OS もシステムを構成するソフトウェアの一部であるため，機能安全プロセス従った各種の設計ドキュメントが必要となる．保護付き AUTOSAR-OS は一般に OS ベンダから購入して使用するため，設計ドキュメントも同時に購入する．OS のソースコードを変更すると，安全性を担保するため，再テストや設計ドキュメントの作り直しが必要となる．そのため，OS のソースコードは，OS ベンダから購入した状態で変更を行わず用いることが一般的である．

2.1.1 保護機構

保護付き AUTOSAR-OS では，プログラムは， OS アプリケーション（OSAP）と呼ばれるグループに分けられる． OSAP には保護が有効な非信頼 OSAP と，無効な信頼 OSAP がある．非信頼 OSAP は，メモリ保護としてアクセス可能なメモリや I/O を制限可能である．そのため，非信頼 OSAP はユーザモードで動作させ，信頼 OSAP と OS を含む BSW は特権モードで動作させる．非信頼 OSAP から BSW の機能（例えば OS の API 呼び出し）を使用するたには，ソフトウェア割込みによる呼び出しが必要となる．時間保護に関しては，タスクや割込みハンドラ（ISR）毎に実行時間や起動間隔を監視する機構を提供する．

2.1.2 割込み機構

保護付き AUTOSAR-OS は，**表 1** に示す 3 種類の割込み機構を提供する．ISR1 は OS の API（OSAPI）を呼び出すことが出来ないが，OS 実行中に割り込むことが可能である．ISR2 は通常の ISR，NT-ISR2 は保護付き AUTOSAR-OS において，保護が有効となった状態で実行される ISR である．そのため，他の ISR と比較して起動オーバヘッド（プロセッサが割込みを受け付けた後，ユーザ記述のハンドラを呼び出すまでの時間）が大きい．

2.2 短周期処理とその実現方法

短周期処理は複数存在し，周期処理と非周期処理に分類できる．どちらの処理も数十マイクロ秒オーダ間隔で処理を行う必要があり，割込みハンドラとして実現されている．周期処理は，1 つのタイマの割込みハンドラ（図中の Dispatcher）から呼び出される．

短周期処理は処理毎に求められる安全度水準が異なる．そのため，本来なら低い安全度水準の処理は NT-ISR2 と

するべきであるが，前述の通り，NT-ISR2 は起動オーバヘッド大きく，OS 実行中は禁止されるため割込み応答時間が長いという問題がある．そのため現状では，全ての短周期処理を ISR1 として実現している．そのため，現状の開発では，次の問題が発生している．

- （課題 1） 低い安全度水準の処理の開発工数が大きい
  低い安全度水準の処理を，保護機構の対象外である ISR1 として実装している．これにより，低い安全度水準の処理を高い安全度水準で開発する必要があり，本来の安全度水準で開発する場合と比較して，開発工数が大きくなるという問題がある．
- （課題 2） 起動抜けや起動遅れを検出できない
  低い安全度水準のタスクによる割込み禁止や，デバイスの不具合により，周期処理に起動抜けや起動遅れが発生することがある．短周期処理は，時間保護の対象外である ISR1 として実装しているため，これらの問題が発生した際の問題の切り分けや不具合解析が困難である．非周期割込みに関しては，周期処理でないためジッタが存在しないため本来はジッタ監視は必要ないが，PWM キャリア割込みのようにタイマ割込みではないが実質的な周期割込み処理に関しては，ジッタ監視を行いたいという要望がある．

3. 機構検討

前述の問題を解決するためには，監視・保護機構を有効にして，短周期処理を実行する必要がある．まず監視・保護機構の検討の前に対象システムの非機能要件について延べ，次に必要な監視・保護機構について説明する．

3.1 非機能要件

機構の検討に関して，必要な非機能要件を挙げる．

（非機能要件 1） OS を変更しない，もしくは変更箇所を局所化・極小化すること．

（非機能要件 2） 実行オーバヘッドを極力小さくすること．具体的には，後述する短周期処理のユースケースにおいて 10%以下の実行オーバヘッドであること．

（非機能要件 3） 短周期処理の割込み応答時間を車載マイコンで 5μs 程度で実現できること

(非機能要件 1) は，2.1 節で説明したように，機能安全の観点から，OS を変更することは困難であるため定めた．ただし OS の核となる箇所以外の変更は許容する．(非機能要件 2) は，実行オーバヘッドが大きいと，使用するプロセッサを性能が高い物に変更する必要があり，コスト増となるため定めた．(非機能要件 3) は，短周期処理は可能な限り高速な割込み応答時間が要求されるため，通常の ISR1 とできるだけ同等の割り込み応答時間、かつ、システム要件で許容される値として 5μs の目標値を定めた

3.2 監視・保護機能

前述の課題を解決するためには，短周期処理の実行に関して，次に挙げる監視・保護機能が必要と考えられる．

[実行時間監視] 短周期処理の実行時間に上限を設けて監視する．上限時間を超えた場合は，割込みを発生させて実行を中断する．

[割込み発生回数監視] 非周期処理の契機となる割込みの一定時間内での発生回数に上限を設ける．上限を超えた場合は，エラー処理を行う．

[ジッタ監視] 周期割り込みが発生してから周期処理が起動されるまでの時間 (ジッタ) が設定値以上になっていないかチェックする．

[メモリ保護] 短周期処理がアクセス可能なメモリや実行可能な命令を制限する．

これらの監視・保護機能を用いることにより，前述の課題は次のように解決可能である．

（課題 1）の解決方法

低い安全度水準の短周期処理に対して，[実行時間監視][割込み発生回数監視] [メモリ保護] を適用することにより，高い安全度水準の短周期処理からパーティショニングすることができ，本来の安全度水準で開発が可能となる．

（課題 2）の解決方法

短周期処理や，PWM キャリア割り込みなどの実質的な周期割り込み処理に対し，[ジッタ監視] を行うことで起動抜けや起動遅れを検出可能となり，不具合の切り分けが容易になる．

4. ISR1 保護フレームワーク

前述の監視・保護機能を短周期処理に適応する機構を提案する．提案機構の概要を図 2 に示す．以降，提案機構を ISR1 保護フレームワークと呼ぶ

ISR1 保護フレームワークは各種監視・保護機能を持ち，各短周期処理の代わりに ISR1 として OS に登録する．ISR1 の契機となる割込みが発生すると，ISR1 保護フレームワークが呼び出される．ISR1 保護フレームワークはまず，[実行回数監視] や [ジッタ監視] を実施した後，対応する短周期処理を呼び出す．その際，対象の短周期処理の安全度水準が低い場合は，[実行時間保護][メモリ保護] を有効にして呼び出す．ここで，安全度水準が高い短期周期処理を実行する保護を有効としない呼び出しを Trusted ISR1（T-ISR1），安全度水準が低い短期周期処理を実行する保護を有効とする呼び出しを Non Trusted ISR1（NT-ISR1）と呼ぶ．なお，タイマ割込みでは 1 つの割込みから複数の短期周期処理を呼び出す必要があるため，どの短周期処理をどの順序で呼び出すかは，ユーザ記述の関数（図中の Dispatcher）で指定する．

4.1 監視・保護機能の実現方法

3.2 節で述べた監視・保護機能の実現方法について述べる．

[実行時間監視] の実現

専用のタイマ（実行時間監視タイマ）用いて NT-ISR1 の実行時間を監視する．実行時間監視タイマの割込み処理は最高優先度の ISR1 として OS に登録して ISR1 保護フレームワークを呼び出す．

[割込み発生回数監視] の実現

実現方法としては，短周期処理の ISR1 から呼び出される ISR1 保護フレームワーク毎にカウンタを用意して，ISR1

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

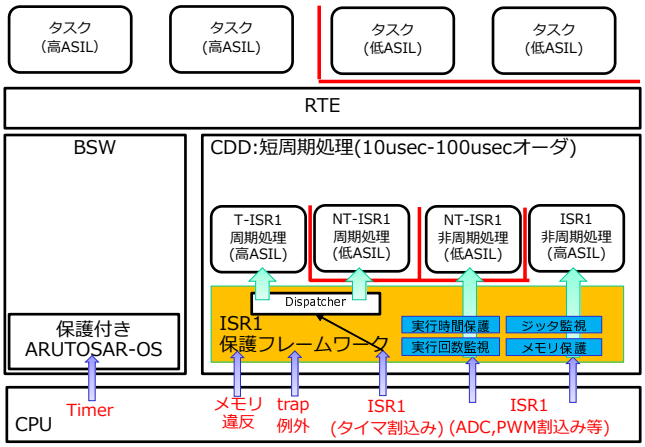

図 2 提案機構
Fig. 2 Proposed Mechanism

保護フレームワークが呼び出されるとこのカウンタをインクリメントした後，発生回数の上限を超えていないか確認する．カウンタのクリア方法は 2 種類あり，独立した周期タイマを用意して，その割り込みハンドラを ISR1 として登録して，呼び出されるとクリアする方法と，短周期処理の周期割込みでクリアする方法がある．

[ジッタ監視] の実現

タイマにより発生する割込みにより呼び出される周期処理の場合は，ISR1 保護フレームワーク実行時にタイマカウントを読むことでジッタが分かる．この際，検出精度は使用するタイマの精度に依存する．実質的な周期割込みに関しては，ISR1 保護フレームワークでは特に保護機構を用意せず，ユーザー定義のジッタ監視処理を呼び出す仕組みを用意する．

[メモリ保護] の実現

MPU とプロセッサの非特権モードを用いて実現する．具体的には，NT-ISR1 実行時に MPU による保護を有効として，非特権モードで NT-ISR1 を実行する．なお，保護付き AUTOSAR-OS においても MPU は使用するため，ISR1 保護フレームワークにおいて MPU を使用する際には，保護付き AUTOSAR-OS が MPU を使用している可能性があるので，その時点の MPU の設定を全て保存して，ISR1 保護フレームワーク終了時に設定を戻す必要がある．メモリ保護違反時には，プロセッサに登録したメモリ保護違反時処理が呼び出される．この処理については，保護付き AUTOSAR-OS と共有する必要がある．

4.2 実装

前節までで述べた機構を，車載システム用のプロセッサである，ルネサスエレクトロニクス社の RH850 を対象に実装した．実装の詳細について，短周期処理の割込み発生から NT-ISR1 が呼び出されるまでと，NT-ISR1 の処理が終了して，割込みから戻るまでの処理に分けて説明する．

4.2.1 NT-ISR1 呼び出しフロー

短周期処理の割込み発生から NT-ISR1 が呼び出されるまでの処理の流れを図 3 に示す．

短周期処理の実行の契機となる割込みが発生すると，ISR1 として（1）ISR1 エントリ (アセンブラ) が実行される．ここでは，割込み前のレジスタやスタック等のコンテキストを保存し，ISR1 保護フレームワーク用のスタックに切り替える．また，ISR1 保護フレームワーク自身の多重割込みを可能とするための処理を行う．次に（2）ISR1 エントリ (C 言語) が呼び出される．ISR1 保護フレームワークが多重に呼びだされた場合は，実行時間監視タイマの停止，[ジッタ監視]，割込みを許可，[割込み発生回数監視] の順で実施する．また，割り込まれた処理が MPU を使用している可能性があるため，MPU の設定を保存する．続く（3）Dispatcher では，短周期処理を安全度水準に応じて，T-ISR1 か NT-ISR1 として呼び出す．Dispatcher は，2.2 節で述べたように，ユーザが記述する関数である．T-ISR1 は単に関数呼び出しとして呼び出されるため，これ以降の処理は必要ない．

NT-ISR1 の場合は，（4）NT-ISR1 呼び出し (C 言語) により，MPU を設定して [メモリ保護] を有効にした後，実行時間監視タイマをスタートさせて [実行時間監視] を有効とする．そして，（5）NT-ISR1 呼び出し (アセンブラ) により，ここまでのコンテキストを保存して，プロセッサの実行モードをユーザーモードに変更して，（6）NT-ISR1 呼び出し 2 (アセンブラ) を呼び出す．最後にユーザーが記述した（7）NT-ISR1 本体を呼び出す．

4.2.2 NT-ISR1 リターンフロー

NT-ISR1 から割込み前の処理までに戻る流れを図 4 に示す．

（8）NT-ISR1 本体からリターンすると，（9）NT-ISR1 リターン (アセンブラ) が呼び出されて，例外呼び出し命令である trap を実行する．これにより（10）trap ハンドラが特権モードで呼び出され，レジスタの復帰とスタックの入れ替えがなされた後，（11）NT-ISR1 リターン 2 を実行し，MPU と実行時間監視タイマを停止して [メモリ保護] と [実行時間監視] を無効とする．その後，（12）Dispatcher にリターンし，他に実行していない短周期処理があれば，（3）に戻って同様に呼び出す．全ての短周期処理を呼び出した後，（13）ISR1 リターン (C 言語) に戻って，MPU を復帰，割込みの禁止の後，多重割込みの際は実行監視タイマを復帰して最後に（14）ISR1 リターン (アセンブラ) に戻る．ISR1 リターンでは，コンテキスト及びスタックの復帰の後，元の処理にリターンする．

4.2.3 NT-IPSR1 中断時の処理

NT-IPSR1 は，保護を有効にして実行するため，次の要因で処理を中断する必要がある．

- 実行時間超過
- メモリ違反

中断後の振る舞いとしては，NT-ISR1 呼び出し処理にリターンしてエラーを返す必要がある．この振る舞いを実現するため，それぞれの要因毎に以下のように実現する．

実行時間超過時は，実行時間監視用のタイマの割込みハンドラ (ISR1) が実行される．このハンドラは，ISR1 中で最高優先度で登録しておく．ハンドラでは，タイマ割込みをクリアして，trap ハンドラ（図 4（10））を関数呼び出しで呼び出す．その際，実行時間が超過したことを知らせるために，trap ハンドラの引数にエラーコードを渡す．

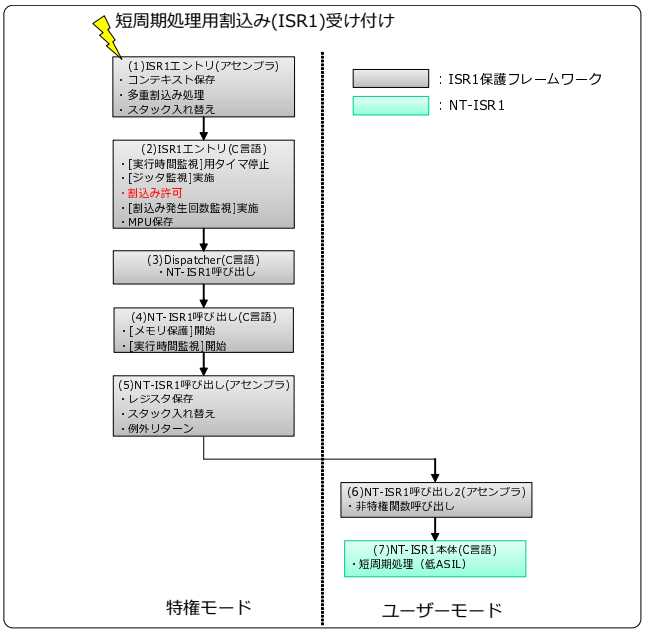

図 3 NT-ISR1 呼び出しフロー
Fig. 3 NT-ISR1 Call Flow

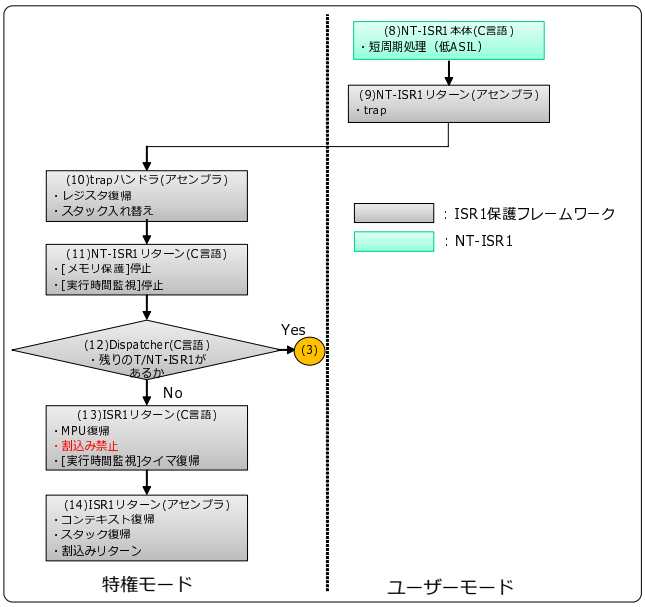

図 4 NT-ISR1 リターンフロー
Fig. 4 NT-ISR1 Return Flow

メモリ違反時は，メモリ保護違反時処理が呼び出される．この処理は保護付き AUTOSAR-OS と共有している．そのため，ISR1 保護フレームワークが動作していることを示すフラグを用意して，ISR1 保護フレームワークが動作していれば，実行時間超過時と同様に trap ハンドラを関数呼び出しで呼び出す．

5. 評価

前節で述べた提案機構の実装が 3.1 節で述べた非機能要件を満たすか評価する．用いた評価環境は次の通りである．

- マイコン：RH850F1H 120MHz
- コンパイラ：GHS コンパイラ
- RTOS：TOPPERS/ATK2[8]

5.1 OS 変更量評価

（非機能要件 1）の充足を確認するため，OS の変更箇所を評価する．OS の変更箇所としては，次の 2 点が挙げられる．他は，ISR1 として実現するため，OS のソースコードを変更する必要はない．

- メモリ保護違反時処理
- ソフトウェア例外（trap）発生時処理

メモリ保護違反時処理は，メモリ違反が発生した際に呼び出される処理である．NT-ISR1 実行時は，メモリ保護を有効とするため，メモリ保護違反が発生すると呼び出される．メモリ保護違反時処理は，OS がメモリ保護違反時の処理を置いている．そのため，OS のメモリ保護違反時処理を変更して，NT-ISR1 実行時にメモリ保護違反が発生した場合は，保護フレームワークを呼び出すように変更する．

ソフトウェア例外発生時処理は，図 4 の（9）NT-ISR1 本体からのリターンに必要である．メモリ保護違反時処理と同様に OS も API 呼び出し等で使用するため，呼び出し部分を変更して，NT-ISR1 実行時に呼び出された場合は，ISR1 保護フレームワークの trap ハンドラ（図 4（10））を呼び出すように変更する必要がある．なお，RH850 では trap ベクタを 2 個持ち，ベクタ番号で呼び出す trap ハンドラを変更可能であるため， OS と使用する trap ベクタを分けるという方法もある．

上記のような変更が必要となるが，どちらの箇所も例外のエントリ処理であり，OS の中心となる部分の変更ではないため，機能安全上，大きな問題にはならないと考え，（非機能要件 1）を満たしていると言える．ISR1 保護フレームワーク動作時は例外ベクタを変更するという方法もあるが，影響が大きいため，個別のエントリ処理を変更する方がよいと言える．

5.2 割込み応答時間評価

提案機構が（非機能要件 3）を充足しているか評価するため，提案機構と既存機構の最悪割込み応答時間を評価した．評価対象は次の通りである．

- 1. ISR1(SC1)：メモリ保護なし OS の ISR1 の割込み応答時間
- 2. ISR2(SC1)：メモリ保護なし OS の ISR2 の割込み応答時間
- 3. ISR2(SC3)：メモリ保護あり OS の ISR2 の割込み応答時間
- 4. NT-ISR2(SC3)：メモリ保護あり OS の非特権 ISR2 の割込み応答時間
- 5. ISR1：提案機構の ISR1 の割込み応答時間
- 6. NTISR1：提案機構の NT-ISR1 の割込み応答時間

割込み応答時間としては，割込みが発生してから割込みハンドラが実行されるまでの実行時間に，OS やフレームワーク自身が各割込みを禁止する割込み禁止区間の最長時間を加算したものを割込み応答時間とした．

測定結果を図 5 に示す．提案機構は各種保護の有効化や MPU の保存復帰が必要なため，実行時間自体はメモリ保護 OS 上の ISR2 や NT-ISR2 より大きいが，これらは，メ

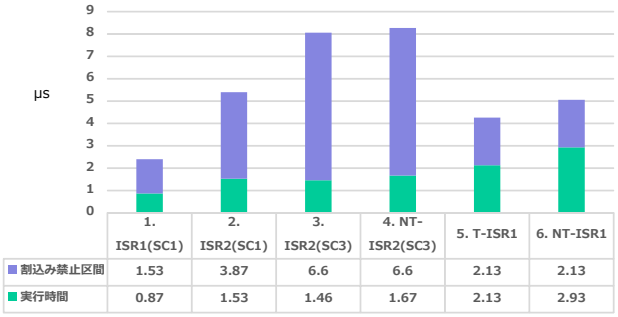

図 5 割込み応答時間評価
Fig. 5 Evaluation of Interrupt Response Time

表 2 短周期処理の構成
Table 2 Configuration of the Short Cyclic Process

名称	ASIL	周期（us）	実行時間（us）
PWM 処理	高	100	5
ADC 処理	高	100	20
周期処理 1	高	500	50
周期処理 2	低	500	25
周期処理 3	低	500	25

モリ保護 OS 自体の割込み禁止区間が長いため，割込み応答時間は，提案機構の方が短くなっている．（非機能要件 3）で要求される 5μs に対して，メモリ保護 OS 上の ISR2 や NT-ISR2 は満たしていないのに対して，提案手法は 5.06μs とほぼ要求を満たしていると言える．

提案機構の割込み応答時間を短縮する方法として，OS と使用する MPU の領域を分割する方法が挙げられる．MPU は複数の領域で構成されており，OS によっては全ての領域が必要でない場合がある．そこで，提案機構で使用していない領域を使用し，NT-ISR1 実行時に有効とする領域を切り替えることにより，MPU の切り替えのオーバヘッドを低減することが可能である．ただし，この手法を OS 側が許容するかは OS 毎に確認する必要がある．

5.3 ユースケースを用いた実行オーバヘッド評価

提案機構が（非機能要件 2）を充足しているか評価するため，適用システムのユースケースを用いて実行オーバヘッドを評価した．

ユースケースの短周期処理の構成を表 2 に示す．PWM 処理や ADC 処理は本来は周期処理ではないが，評価のために最短周期で実行される周期処理とした．

表 2 の短周期処理を ISR1 保護フレームワーク上で動作させた場合の ISR1 保護フレームワークの実行オーバヘッドを計測した．測定方法は次の通りである．まず，最低優先度で動作するアイドル処理を用意し，短周期処理を実行しない場合の短周期処理の周期に対して十分に長い時間内でアイドル処理を実行した時間を計測する．次に，短周期処理を実行し，同じ時間内でアイドル処理を実行した時間を計測する．これらの結果から，アイドル処理の実行時間の減少割合を求め，さらに短周期処理の周期の最小公倍数の周期中の短周期処理の実行時間の割合を引くことで，ISR1 保護フレームワークの実行オーバヘッドを求める．

評価の結果，アイドル処理の実行時間は 52.2%減少した．短周期処理の周期の最小公倍数の周期中の短周期処理の実行時間の割合は，45%（500ms 中に 225ms）であるため，ISR1 保護フレームワークの実行オーバヘッドは，7.2%（52.2% - 45%）と 10%以下であり，（非機能要件 2）を満たしていると言える．

6. おわりに

本論文では，性能要件から OS 管理外の割込みとして実行する必要がある低い安全度水準の短周期処理に保護を適用する ISR1 保護フレームワークを提案した．提案機構は，OS 管理外の割込みである ISR1 として動作し，各種監視・保護機能を有効にして低い安全度水準の短周期処理を実行することにより，高速な割込み応答時間と保護を両立する．提案機構を実装し評価した結果，非機能要件も満たすことを示した．

今後の課題としては，OS が使用する MPU の領域と別の領域を用いることによる実行オーバーヘッドの低減や，実システムへの適用等が挙げられる．

謝辞 本研究の一部は JSPS 科研費 JP26330062 の助成を受けたものです．

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] 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, Nov 2012.
[4] D. Reinhardt and G. Morgan, ”An embedded hypervisor for safety-relevant automotive E/E-systems,” Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014), Pisa, pp. 189-198 2014.
[5] 本田晋也, 鈴木均, 樋口正雄, 福井昭也, ”車載システム向けハードウェア仮想化支援機能による RTOS 一体型仮想マシンモニタ,” 情報処理学会 OS 研究会，Mar. 2017.
[6] 本田晋也，岡部亮，攝津敦， 車載システム向けの仮想マシンの割込み応答性向上手法, 情報処理学会研究報告, Vol.2017-SLDM-179,No.14, pp. 1-6, 沖縄, Mar 2017.
[7] リアルタイム制御システムに適した V850 CPU 向け仮想化技術 (online), available from ⟨https://www.renesas.com/ja-jp/about/press-center/news/2010/news20100929.html⟩ (accessed 2017-01-19).
[8] TOPPERS/ATK2 (online),available from ⟨http://www.toppers.jp/atk2.html⟩ (accessed 2015-10-24).