Vol.2020-EMB-55 No.6
2020/12/11

時間パーティショニング機構を持つリアルタイムOSの性能評価手法

手塚　湧太郎[1]　本田　晋也[2,1]　大谷　寿賀子[1,3]　枝廣　正人[1]

概要：自動車や航空宇宙機などの分野の組込みシステムでは，一般的な組込みシステムに比べ高い信頼性が求められている．そこで，システムを複数のモジュールに分割して時間的にお互いの影響を無くす時間パーティショニング機構を持つリアルタイム OS(パーティショニング OS) が使用されつつある．本研究では，パーティショニング OS の実装を分析することで，時間パーティショニングに発生する時間誤差を明らかにし，その時間誤差を計測する手法を提案する．提案した計測手法をオープンソースのパーティショニング OS である TOPPERS/HRP3 カーネルに適応することにより，提案手法の有用性を示した．

1. はじめに

航空宇宙機や自動車といった高信頼性が求められる機器を制御する組込みシステムは，高い安全度水準が求められる．このようなシステムの機能 (ソフトウェア) をベアメタルや通常のリアルタイム OS を用いて実現すると，全てのソフトウェアをシステム中で最も高い安全度水準の機能が要求するプロセスで開発する必要があることや，ある機能を変更して時間的な振る舞いが変わると，全てのソフトウェアの再検証が必要となることなどから，開発工数やコストが大きくなるという問題がある．

この問題を解決する方法として，ソフトウェアを安全度水準毎にパーティションと呼ぶモジュールに分割し，お互いに影響しない Freedom From Interference(FFI) を満たしながら実行するパーティショニング機構を使用する方法がある．パーティショニングは，メモリパーティショニングと時間パーティショニングの 2 種類に分類できる．メモリパーティショニングは，あるパーティションが他のパーティションのメモリへアクセス出来ないように制限する機構である．時間パーティショニングは，あるパーティションの時間的な振る舞いが，他のパーティションに影響を及ぼさないようにする機構である．

メモリパーティショニングは対象のパーティションを実行する際に，MMU ないし MPU を設定して，ユーザモードで実行する．一方，時間パーティショニングは，TDMA スケジューリング [1] や階層型スケジューリングと呼ばれるスケジューリングにより，一定の時間幅（システム周期）を区切ったタイムウィンドウに特定のパーティションを割り付けて，タイムウィンドウ内はパーティション毎のスケジューラを実行する方法が一般的である．

パーティショニングの制御は，リアルタイム OS を使用する方法や，ハイパーバイザを使用する方法がある [2]．ハイパーバイザを使用する方法で実行を効率化するためには，ハードウェア仮想化支援機能が必要となるが，自動車制御システム向けや小規模組込みシステム向けのプロセッサでは，現状はこの機能を持たないものが主流であるため，リアルタイム OS で実現する方法が一般的である．パーティショニング機能を持つリアルタイム OS をパーティショニング OS と呼ぶ．

パーティショニング OS はユーザが要求するパーティショニングを高い精度で実現する必要がある．メモリパーティショニングは，空間的なパーティショニングであるため，パーティション実行前にパーティショニング OS が適切にハードウェアを設定すれば，要求されたパーティショニングを 100%の精度で実現することが出来る．一方，時間パーティショニングは，時間的な振る舞いのパーティショニングであるため，パーティショニングの実現自体に実行時間が必要であることや，割込みを用いて時間を区切ることによるジッタが発生する．そのため，ユ－ザーから要求（設定）されたパーティショニングに対して，ある程度の時間誤差が発生する可能性がある．例えば，ユーザがあるタイムウィンドウの実行時間を 500[μsec] と設定したとしても，実際には 498[μsec] しか実行されない可能性がある．また，パーティショニング OS が時間パーティショニングの実現のために処理を行う必要があり，ユーザが使

1　名古屋大学 情報学研究科
2　南山大学 理工学部
3　ルネサスエレクトロニクス株式会社

© 2020 Information Processing Society of Japan

用可能な時間が減る可能性がある．ユーザはこの時間を考慮してシステム周期やタイムウィンドウの設計を行う必要がある．すなわち，高信頼システムをパーティショニングOSを使用して実現する際には，どの様な要因でどの程度の時間誤差が発生するか，設計時に把握しておく必要がある．

パーティショニングOSが実現可能な時間パーティショニングの精度等を評価した研究は存在する[3]．しかしながら，TDMAスケジューリングを実現するパーティショニングOSの実装を分析して，どの様な時間誤差が発生する可能性があるか分析し，それらを計測する手法は確立していない．さらに，TDMAスケジューリングでは，パーティショニングはタイムウィンドウの時間が経過すると自動的に切り換えられるため，通常のプログラムの実行時間計測方法では，各種の時間誤差を計測するのは困難であるという問題もある．

そこで本研究では，オープンソースのパーティショニングOSであるTOPPERS/HRP3カーネル（以下HRP3カーネル）を題材として，時間パーティショニング機構の仕様と実装を分析し，どの様な時間誤差が発生する可能性があるかを明らかにする．そして，それらの時間誤差を計測するための手法について提案し，HRP3カーネルに適用してその正当性と有用性を評価する．

2. パーティショニング機構

本章では，パーティショニング機構の概要について説明したのち，本研究で評価対象とするパーティショニングOSの仕様と実装について解説する．

パーティショニング機構は，ソフトウェアをパーティションと呼ばれる空間的かつ時間的に独立したいくつかの保護領域に分割し，お互い影響を受けないように保護する．パーティショニング機構は，メモリパーティショニングと時間メモリパーティショニングの2種類に分類できる．

2.1 メモリパーティショニング

メモリパーティショニングは，あるパーティションが他パーティションのメモリへのアクセスを禁止することで，パーティションの誤動作の伝搬を防止する．メモリパーティショニングについては，我々はこれまで評価手法について研究を実施してきた[4]．

2.2 時間パーティショニング

時間パーティショニングは，あるパーティションの時間的な振る舞いの変化が，他のパーティションに影響を及ばさないようにする．我々が知る限りでは，**図1**に示すTDMAスケジューリングにより各パーティションを実行する方法が一般的である．TDMAスケジューリングは，時間枠にシステム周期を定める．そして，システム周期をいくつかのタイムウィンドウに分割する．タイムウィンドウには，いずれかのパーティションが割り当てられ実行される．パーティションに属するタスクは，割り当てられたタイムウィンドウ内でスケジューリングされ，CPU利用時間，実行順序，実行タイミングなどを保証(パーティションから見ると保護)する．CPU利用時間の保証とは，システム周期内でパーティションに割り付けたタイムウィンドウの時間（CPU時間）必ずパーティションが実行されることを保証することである．実行順序の保証とは，タイムウィンドウの割り付け順に必ずパーティションが実行されることを保証することである．実行タイミングの保証とは，タイムウィンドウを割り付けたシステム周期からのオフセットで必ずパーティションが実行されることを保証することである．言い換えると，タイムウィンドウの設定通りにパーティションが実行されることを示す．

表1 時間パーティショニングのレベル

	ユーザ割込み	CPU利用時間	実行順序	実行タイミング
レベル1	有効	保証	保証	
レベル2	無効	保証	保証	保証

どの項目を保証する必要があるかは，システム毎に異なるため，本論文では，**表1**に示す2段階のレベルを定義する．レベル1は，自動車制御向けのOSやハイパーバイザで採用されている方式である[1]．レベル2は，航空機向けのソフトウェアプラットホームの標準仕様であるARINC653[5], [6]で定義されている方法である．

2.3 TOPPERS/HRP3カーネル

本論文で分析の対象とするパーティショニングOSであるHRP3カーネルについてその仕様と実装について説明する．HRP3カーネルは，ITRON仕様をベースとして，メモリパーティショニングと時間パーティショニング機能を拡張したリアルタイムOSである．

2.3.1 メモリパーティショニング機能

HRP3カーネルでは，パーティションに相当するものとして保護ドメインと呼ぶカーネルオブジェクトのグループを低することが可能である．ある保護ドメインから他の保護ドメインのメモリにアクセスできないように設定できる．この機能は，MPUないしMMUによって実現される．

2.3.2 時間パーティショニング機能

HRP3カーネルでは，レベル1ないしレベル2の時間パーティショニングをサポートしている．システム周期やタイムウィンドウ切り換えタイミングは，それぞれ専用のタイマにより計時し，割込みを発生することで実現している．システム周期の長さや，その中でのタイムウィンドウの実行順序や実行時間はユーザが設計時に設定する．タイムウィンドウに保護ドメインが割り当てられ，所属するタスクはその時間内で優先度ベースのスケジューリングで実

© 2020 Information Processing Society of Japan

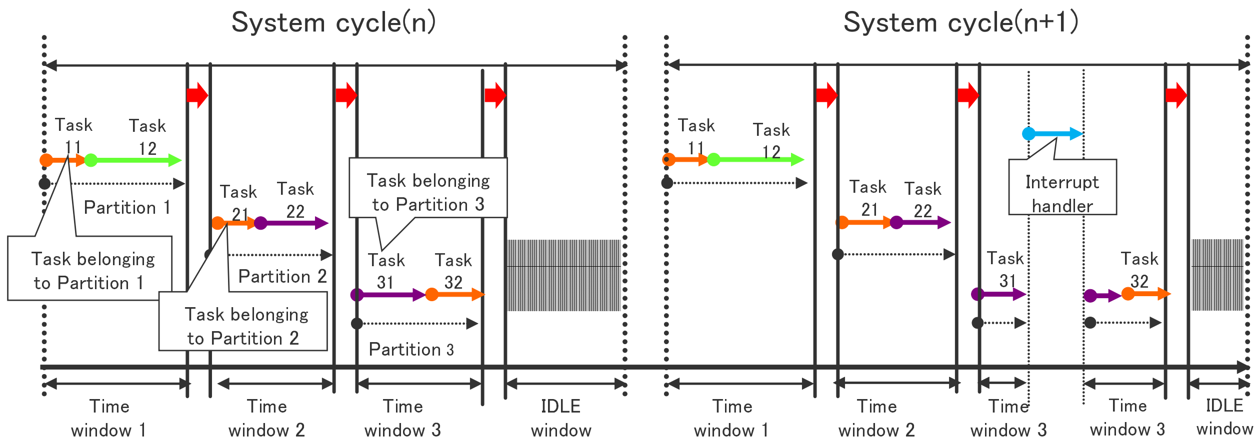

図 1　TDMA スケジューリング（n 周期はレベル 1 の例，n+1 周期はレベル 2 の例）

行される．

HRP3 カーネルでは，時間パーティショニングが保証するレベルをレベル 1 としてもレベル 2 としても使用可能である．

レベル 1 の場合，HRP3 カーネルでは TDMA スケジューリングを拡張して割込み（ユーザ割込み）を受け付け可能としている．この際，ユーザ割込みが受け付けられるとタイムウィンドウの計時用のタイマを一旦停止し，割込みの処理が終了すると，タイムウィンドウの計時用タイマを再開する．これにより，図 1 の n+1 周期に示す様に，割込みの処理の実行時間分，タイムウィンドウを後方にシフトすることになり，CPU 利用時間を保証する．タイムウィンドウがシステム周期を超えてシフトすると問題となるため，HRP3 カーネルでは，図 1 に示すように，アイドルウィンドウと呼ばれるどの保護ドメインも割り当てられていないタイムウィンドウをシステム周期の最後に設定する．そして，タイムウィンドウが後ろずれた時間分アイドルウィンドウが短くなることで，タイムウィンドウがシフトしたとしても，システム周期内に収まるようにする．あるシステム周期内で，アイドルウィンドウ以上の長さの割込み処理が発生しないようにするのは，ユーザ責任となる．

一方，割込みを一切有効にしなければ，レベル 2 の時間パーティショニングを実現することが出来る．この場合は，タイムウィンドウのシフトは発生しない．

また，時間パーティショニング機能は無効とすることが出来る．無効とした場合は，TDMA スケジューリングは行わず全ての保護ドメインに所属するタスクをまとめて優先度ベースでスケジューリングする．

3. パーティショニング OS の分析と評価項目の抽出

本章では，HRP3 カーネルを題材に，時間パーティショニングの実現方法を分析することで，発生しうる時間誤差について提示する．まず，発生する時間誤差は大きく 2 種類に分けることが可能である．本章では，それぞれについて，詳細な項目と発生する状況について説明する．

- 時間パーティショニングの実行オーバヘッド
  パーティショニング OS の実行により，ユーザが使用出来ない時間．
- 時間パーティショニングのジッタ
  パーティショニング OS の実行により，タイムウィンドウの CPU 利用時間や実行タイミングに発生するジッタ．

3.1　時間パーティショニングの実行オーバヘッド

発生する実行オーバヘッドとしては，パーティションの切り換え処理と割込み処理前後の実行オーバヘッドが挙げられる．

3.1.1　パーティションの切り換え処理オーバヘッド

パーティションの切り換え処理として発生する実行オーバヘッドであり，**図 2** の (a),(b),(c) の 3 種類に分類される．

- (a) システム周期切り換え (tm_{sys})
  システム周期を切り換える処理であり，アイドルウィンドウからシステム周期の先頭のタイムウィンドウに切り換える．システム周期につき 1 回発生する．
- (b) タイムウィンドウ切り換え (tm_{twd})
  あるタイムウィンドウから別タイムウィンドウに切り換える処理であり，システム周期につき，タイムウィンドウ数 −1 回発生する．
- (c) アイドルウィンドウ切り換え (tm_{idle})
  あるタイムウィンドウからアイドルウィンドウに切り換える処理であり，システム周期につき 1 回発生する．

システム周期中のタイムウインドウの数を num_{twd} とすると，ユーザは，システム周期内の以下の時間は使用することが出来ない．

$$tm_{sys} + tm_{twd} * (num_{twd} - 1) + tm_{idle} \quad (1)$$

© 2020 Information Processing Society of Japan

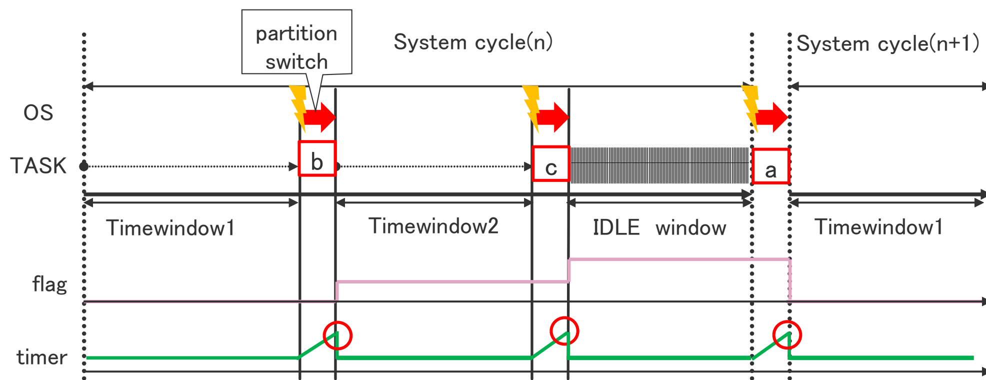

図 2 パーティションの切り換え処理オーバヘッド評価手法

3.1.2 割込み処理前後オーバヘッド

割込み処理の実行に伴う実行オーバヘッドであり，**図 3** における割込みハンドラの実行前後の OS が実行されている赤色区間である．HRP3 カーネルでは時間パーティショニングを使用する場合，割込みを受け付ける入口処理において，タイムウィンドウの停止の処理が使用しない場合から追加で実行されるため，割込みの応答時間が長くなると考えられる．また，割込みが終了する出口処理においても，停止したタイムウィンドウを再開する処理が追加が実行されるため，割込みからの復帰時間が長くなると考えれる．割込み処理の実行オーバヘッドを tm_{int} システム周期内の割込みの最大発生回数を $intcount_{max}$ とすると，ユーザは，システム周期内の以下の時間は使用することが出来ない．

$$tm_{int} * intcount_{max} \tag{2}$$

3.2 時間パーティショニングのジッタ

パーティショニング OS の実行により，タイムウィンドウの実行時間や実行タイミングにジッタが発生する．ユーザプログラムとは非同期に発生する割込み処理に伴う OS 処理（非同期 OS 処理）とユーザプログラムの実行を契機に実行される OS 処理（同期 OS 処理）により，タイムウィンドウにジッタが発生すると考えられる．

3.2.1 非同期 OS 処理によるジッタ

2.3.2 節で述べたように，タイムウィンドウ実行中に割込みが発生すると，割込みの入口処理でタイムウィンドウの計時用タイマを停止して，出口処理で再開する．実際には，入口処理では必要なレジスタを保存した後にタイマを停止し，出口処理ではタイマを再開した後にレジスタの復帰と例外処理からのリターン処理が実行される．入口処理でタイマを停止するまでの時間と出口処理でタイマを再開して元の処理に戻るまでの時間はタイムウィンドウの計時用タイマは計時を行うため，タイムウィンドウとして使用可能な実行時間すなわち CPU 利用時間は割込みが発生しなかった場合と比較して短くなる．そのため，時間パーティショニングのレベル 1 が保証できない．この時間を 0 にすることは本質的に不可能であるため，1 回の割込み当たりタイムウィンドウの CPU 利用時間がどの程度減るか評価し明らかにして，その結果を設計時に考慮して設計する必要がある．

3.2.2 同期 OS 処理によるジッタ

HRP3 カーネルでは，ユーザが API を呼び出して OS がその API を処理している間は全ての割込みを禁止している．また，ユーザが明示的に呼び出すのではなく，例外が発生した場合にも OS 処理が実行される．パーティションの切り換えの処理は割込みを契機に実行されるため，割込み発生時に同期 OS 処理の実行中であると，その処理が終了するまでパーティションの切り換えの処理が延期される．その結果，後続のタイムウィンドウの実行タイミングが変化するため，時間パーティショニングのレベル 2 では問題となる．例えば，**図 5** のように OS 実行を伴う処理が実行されているタイミング A で実行されるはずであったパーティションの切り換えが B へ延期され，C で切り換え処理が終了される．

4. 計測手法の設計

本章では，3 章で選定した評価項目を計測する手法について提案する．まず，計測手法の方針及び，通常のプログラムの実行時間とは異なり間接的な測定が必要であることについて述べたあと，各項目について具体的な計測方法について説明する．

4.1 計測手法の方針

実行時間を計測する方法は，ソフトウェアのみで計測する方法と外部機器を用いて計測する 2 種類の方法がある．外部機器を用いる方法としては，プログラムの計測したい区間を実行する前に GPIO を ON とし終了時に OFF とし，その信号の長さをロジックアナライザで計測する方法が挙げられる．外部機器を用いる方法は，外部機器を用意する必要がある点，計測項目によっては OS 内に手を入れる必要がある点，TDMA スケジューリングの性質上計測区間を正確に計測出来ないという点で問題がある．例えばタイ

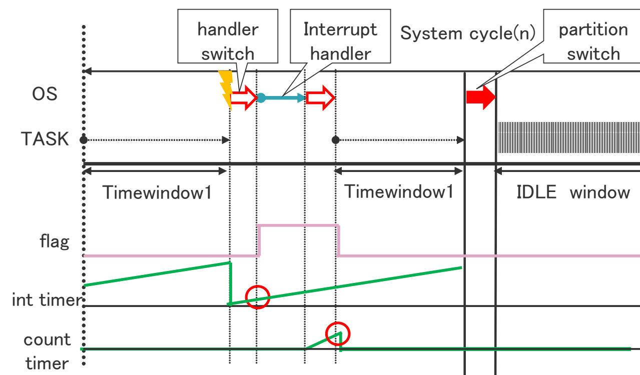

図 3 割込み処理前後オーバヘッド評価手法

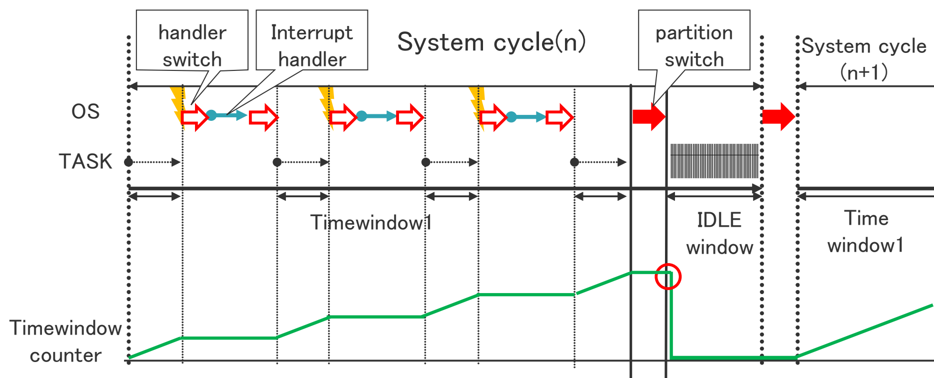

図 4 非同期 OS 処理によるジッタの評価手法

ムウィンドウの切り換え処理を計測する場合，OS のタイムウィンドウ切り換え処理の先頭で GPIO を ON にして，処理の最後に OFF にすることで計測する手法が考えられるが，タイムウィンドウ切り換え実行オーバヘッドは正確には，タイムウィンドウ用のタイマの割込みが入ってから，次のタイムウィンドウの処理が実行されるまでである．そのため，この評価方法では正確に測定対象区間を計測することはできない．

ソフトウェアのみで計測する方法は，主に OS に手を加えて計測する方法とアプリケーションのみで計測する方法がある．前述したように，OS に手を加えると評価結果に影響を与える可能性があるため，本研究では，ユーザアプリケーションとマイコン内蔵タイマのみを用いた計測手法を提案する．

4.2 間接的な測定の必要性

通常，性能評価では計測したい処理の前でタイマをスタートして，処理の終了後にタイマを停止して実行時間を計測する．例えば，タスク起動 API を測定したい場合は，タイマをフリーランニングモードで実行しておき，API を呼び出す直前と直後のタイマカウンタレジスタの値を読み込んでその差分を求めればよい．しかしながら，時間パーティショニングを評価する際にはこの手法は使用出来ない．例として，図 2 における (b) タイムウィンドウ切り換えのオーバヘッドを計測する方法を考える．この時，フリーランニングモードのタイマを Timewindow1 の末尾と Timewindow1 の先頭で読むことができれば計測することができる．しかしながら，Timewindow は前回のシステム周期 (System Cycle($n-1$)) で終了された箇所から処理を再開する．つまり，システム周期が切り換わると前回のタイムウィンドウの先頭や末尾であった処理が次のタイムウィンドウの先頭や末尾でなくなる．そのため，コード上でタイマ読むタイミングををタイムウィンドウの先頭や末尾として記すことができない．そこで，今回は評価項目ごとに計測手法を提案する．

4.3 時間パーティショニングによる実行オーバヘッド

4.3.1 パーティションの切り換え処理オーバヘッド

この項目では，共有変数によるフラグとタイマカウンタレジスタの操作を用いた測定手法を用いる．計測区間の前のタスクに計測開始時間記録タスクを，後ろのタスクに計測終了時間記録タスクを割り当てる．例えば，図 2 において (b) のタイムウィンドウ切り換えを測定する場合には，Timewindow1 に計測開始時間記録タスクを，Timewindow2 に計測終了時間記録タスクを割り当てる．それぞれのタスクは以下の処理を繰り返し実行する．

- 計測開始時間記録タスク
  (1) フラグを立てる．

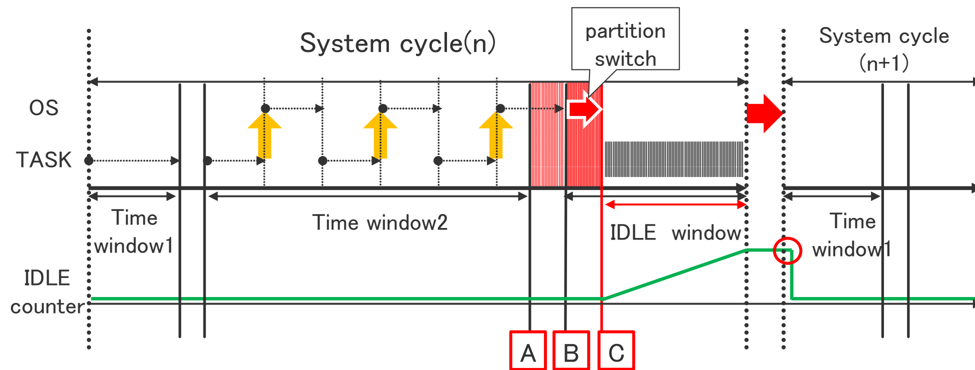

図 5 同期 OS 処理によるジッタの評価手法

(2) タイマカウンタを 0 にする．

- 計測終了時間記録タスク
  (1) フラグが立っている場合 (2) 以降を実行する．フラグ立っていない場合，評価を繰り返す．
  (2) タイマカウンタを読む．
  (3) フラグを降ろす．

この処理の組み合わせにより，タイムウィンドウ切り換え実行中はタイマカウンタがインクリメントされ，次のタイムウィンドウの先頭でタイマカウンタを読み込む事で，タイムウィンドウの切り換え処理の時間を計測することができる．

図 2 では，(a),(b),(c) すべての項目を計測するため，フラグの状態を 3 タイプで示したいるが，計測ではフラグの状態の中で必要な状態 2 つを使用する．

4.3.2 割込み処理前後オーバヘッド

割込みにはマイコン内蔵タイマを使用した周期割込みを用いる．周期割込みは，マイコン内蔵タイマを周期カウンタモードでスタートし，タイマカウンタレジスタが指定された値になると割込みを要求し，タイマカウンタレジスタがクリアされ，再びカウントアップを始める．そのため，周期割込みに使用しているタイマのタイマカウンタレジスタの値を割込みハンドラの先頭で読むことで応答時間を計測する．また，割込みからの出口処理時間は共有変数によるフラグとタイマカウンタレジスタの操作を用いて計測する．図 3 において，inttimer はカウントアップが設定した周期に達すると割込みを要求しカウント値のクリアを行う．その後，inttimer を割込みハンドラの先頭で読むことで，割込み応答処理でカウントされたオーバヘッドを得ることができるので応答時間を測定できる．また，出口処理時間は Interrupt handler に計測開始時間記録タスクの処理をループさせずに割り当て，Timewindow1 に計測終了時間記録タスクを割り当てることで測定できる．

4.4 時間パーティショニングのジッタ

4.4.1 非同期 OS 処理によるジッタ

この項目では，変数のカウントアップ処理による計測方法を用いる．変数のカウントアップ処理を行うタスクを実行し、割込みが発生する場合と発生しない場合について，変数の値を比較することで，割込み処理により減少したタイムウィンドウの CPU 時間を計測する．**図 4** においては，Timewindow1 に変数 Timewindow counter のカウントアップを行うタスクを実行することで割込みが発生した場合の変数 Timewindow counter の値を計測する．変数 Timewindow counter の値は，次のタイムウィンドウ (図 4 では IDLE window) で読み込み記録してカウント値を 0 に初期化する．この計測を割込みを発生させずに実行して，その差分をジッタとする．

4.4.2 同期 OS 処理によるジッタ

この項目では，同期 OS 処理によりタイムウィンドウのシフトが発生することよって，アイドルタイムウィンドウの CPU 利用時間が短くなることを利用して測定する手法を提案する．これは，システム周期のタイミングはタイムウィンドウとは異なるタイマで周期的に実現しているため，タイムウィンドウがシフトしたとしてもずれないためである．同期 OS 処理の呼び出しを繰り返すタスクをアイドルウィンドウの直前のタイムウィンドウで実行する．アイドルタイムウィンドウの CPU 利用時間は，割込みによるタイムウィンドウのジッタと同様に，カウントアップタスク行うタスク実行することで計測する．その上で，OS 処理を呼び出す場合とそうでない場合について，アイドルタイムウィンドウの CPU 利用時間を比較することで計測する．

5. 評価

本章では，4 章で提案した計測手法が実現可能であること，及び 3 章で述べたジッタが発生することを確認するため，HRP3 カーネル上で実装し評価した結果について説明する．**表 2** に評価環境について示す．小型の機器制御に持ちられるワンチップマイコンを使用した．

5.1 時間パーティショニングによる実行オーバヘッド

5.1.1 パーティションの切換え処理オーバヘッド

各項目について 1000 回の測定を行った．評価結果を**図 6**

© 2020 Information Processing Society of Japan

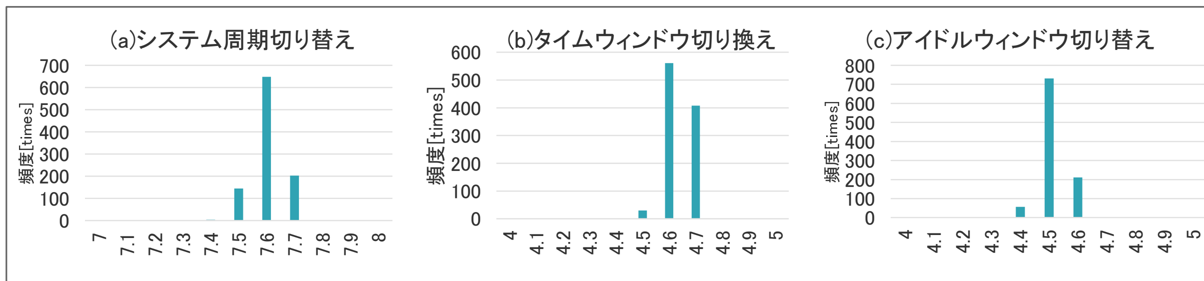

図 6　パーティションの切り換えオーバヘッドの計測結果 [μsec]

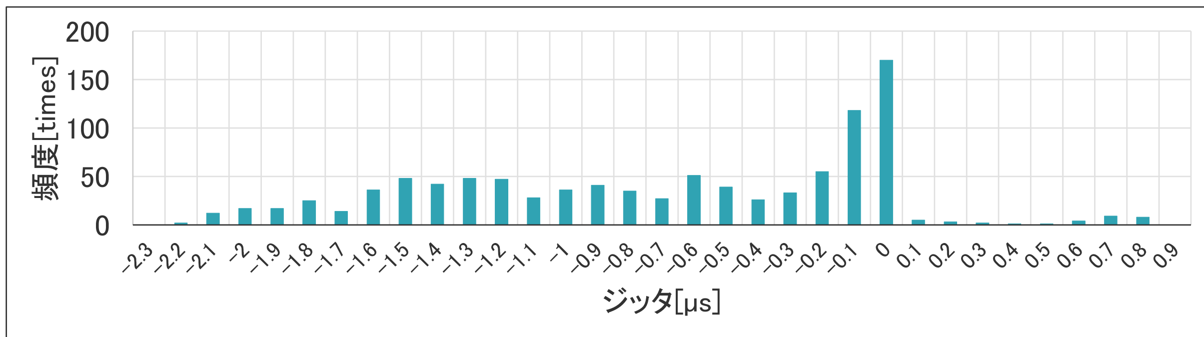

図 7　同期 OS によるジッタの計測結果

表 2　評価環境

ボード	HSBRX64M (北斗電子)
プロセッサ	RX64M (ルネサスエレクトロニクス)
キャッシュ	なし
ROM/RAM	内蔵フラッシュ・SRAM
コンパイラ	CC-RX (ルネサスエレクトロニクス)
コンパイルオプション	optimize = 2 (全体的に最適化)
動作周波数	120[MHz]
評価用タイマ	内蔵タイマ 60[MHz]

に示す．横軸は切り換え時間を示し，縦軸は計測回数を示すヒストグラムである．図 6(a) の結果から，7.5 から 8.0[μsec] 程度システム周期の切り換えのオーバヘッドが発生していることがわかる．そのため，例えば 200[μsec] のシステム周期で実行するシステムを構築するときには，タイムウィンドウとして使用可能な時間は 192[μsec] として設計する必要があると考えられる．また，図 6(b) からタイムウィンドウの切り換えにつき 4.5[μsec] 程度，図 6(c) からアイドルウィンドウの切り換えに 4.5[μsec] 程度の事項オーバヘッドが発生し，これらの時間はユーザが使用できない．タイムウィンドウの切り換えはユーザが設定するタイムウィンドウの数によって発生回数が決まる．そこで，システム周期には，発生するタイムウィンドウの回数とアイドルウィンドウの切り換えの実行オーバヘッドの合計値以上のアイドルウィンドウを確保しておく必要がある．

5.1.2　割込み処理前後オーバヘッド

計測条件としては，時間パーティショニングを使用する場合としない場合について 1000 回ずつ計測を行った．評価の結果，割込み応答時間は，時間パーティショニングを使用した場合，1000 回の試行のうち 1000 回すべてで 1.0[μsec] であった．時間パーティショニングを使用しない場合，1000 回の試行のうち 1000 回すべてで 0.5[μsec] であった．割込みの出口処理時間は，時間パーティショニングを使用した場合，1000 回の試行のうち 0.8[μsec] が 36 回，0.9[μsec] が 964 回，であった．時間パーティショニングを使用しない場合，1000 回の試行のうち 1000 回すべてで 0.5[μsec] であった．割込み応答時間は，時間パーティショニングを使用しない場合と使用した場合と比較して，0.5[μsec] 程度長くなることが分かった．割込みの出口時間は，時間パーティショニングを使用しない場合と使用した場合と比較して，0.4[μsec] 程度長くなることが分かった．この結果から，HRP3 カーネルでは，システム周期内時間は，1 システム周期内で想定する最悪割込み回数 $*1.9[\mu sec]$ 程度は，割込み処理前後オーバヘッドにより消費される可能性があることが分かった．さらに，各タイムウィンドウは，タイムウィンドウ内で受け付けた割込み処理の処理時間に加えて，受け付けた回数*1.9[μsec] 実行タイミングがシフトすることが分かった．

5.2　時間パーティショニングのジッタ

評価手法で用いる変数のカウントアップは以下のように設定した．1 カウントから時間へと変換する重みは 1 カウント当たり 0.1[μsec] とした．この重みは，別途カウントアップ処理をタイマで計測し，重みを算出した．計測結果のカウント値にそれをもとに重み付けすることにより，カウント値から時間への変換を行った．

© 2020 Information Processing Society of Japan

5.2.1 非同期 OS 処理によるジッタ

タイムウィンドウに発生させる非同期 OS 処理には，マイコン内蔵タイマを使用した周期割込みを指定した．割込みが発生する場合としない場合について計測を行った．タイムウィンドウで発生した割込み回数とループによるカウント値の差から一回当たりの割込みによるジッタを計測する．今回は，割込みが発生するタイムウィンドウの長さを 5000[μsec]，割込みの周期を 100[μsec] に設定してシステム周期を 10 回繰り返し計測を行った．

計測の結果，割込みが合計で 498 回タイムウィンドウ実行中に発生し，一回当たりの割込みによるタイムウィンドウのジッタは 1.0[μsec] であった．この結果から，HRP3 カーネルでは，各タイムウィンドウ時間は，1 システム周期内で想定する最悪割込み回数 $*1.0[\mu sec]$ 余分に設定する必要があることが分かった．

5.2.2 同期 OS 処理によるジッタ

同期 OS 処理として，タスク起動とタスク終了 API を用いた．また，周期的に API を呼び出してしまうと，OS 処理が行われていないタイミングに周期的に切り換え処理が発生しジッタを計測できない可能性があるため，不均一な周期で API を呼び出すように実装した．また，システム周期を 6000[μsec] に設定し，そのうちユーザドメインのタイムウィンドウに 4000[µsec] 割り当てた．つまり，計測するアイドルウィンドウ時間の理論値は 2000[μsec] 程度である．API の呼び出しがある場合とない場合についてそれぞれ 1000 回のタイムウィンドウ切り換えを実施した．

同期 OS 処理によるジッタの評価結果について，**図 7** に示す．API を呼び出した場合のアイドルウィンドウの時間から，API を呼び出さない場合のアイドルウィンドウ時間の最頻値を引いた値を，0.1[μsec] ごとの階級で示し，度数が計測回数を示すヒストグラムである．

この結果から，API 実行などの同期 OS 処理が発生するとアイドルウィンドウの時間が短くなる，つまりタイムウィンドウの切り換えタイミングが遅延することが確認できた．

6. おわりに

本研究では，パーティショニング OS の時間パーティショニング機構の仕様と実装を分析し，どの様な時間誤差が発生する可能性があるかを明らかにし，それらの時間誤差を計測するための手法について提案した．各評価項目について，提案した手法に基づいて，HRP3 カーネルで実装を行い，発生する時間誤差を示した．今後の課題としては，マルチコア対応のパーティショニング OS で評価手法の実装と計測を実施することである．

参考文献

[1] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein and M. Wolf, "Special Session: Future Automotive Systems Design: Research Challenges and Opportunities," 2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 1-7, 2008.
[2] A. Masrur, S. Dr ¨ ossler, T. Pfeuffer and S. Chakraborty,"VM-Based Real-Time Services for Automotive Control Applications",2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications,Macau SAR, China, 2010.
[3] U. Usug, Y. Yilmazer, A. Alptekin, H. Yilmaz,"An Approach for Verification of ARINC 653 Time Partitioning Concept",34th Digital Avionics Systems Conference, 2015.
[4] K. Shigihara, S. Honda, H. Takada,"Test Program Generator for AUTOSAR OS",13th European Dependable Computing Conference(EDCC 2017), pp.79-86, Geneva, Switzerland, 2017.
[5] P. J. Prisaznuk, "ARINC 653 role in Integrated Modular Avionics (IMA)," 2008 IEEE/AIAA 27th Digital Avionics Systems Conference, St. Paul, pp. 1.E.5-1-1.E.5-10, 2008.
[6] S. H. VanderLeest, "ARINC 653 hypervisor," 29th Digital Avionics Systems Conference, Salt Lake City, pp. 5.E.2-1-5.E.2-20, 2010.

© 2020 Information Processing Society of Japan