RENESAS

HW-RTOS

Real Time OS in Hardware

割込み応答性能をさらに高める HW-RTOS の機能

本ホワイトペーパでは、HW-RTOS において、さらに高い割込み性能と、ソフトウェアフレンドリな環境の双方を満足させる二つの機能を紹介する。一つは ISR のハードウェア化で、これにより割込みからタスク起動までの応答性能が向上し、さらにハンドラをソフトウェアで開発する必要がなくなる。もう一つは、OS 管理外割込みハンドラをマルチタスク環境で利用できる機能である。これらのきのうにより Non-RTOS 環境からマルチタスク環境への移行が容易になる。

R70WP0003JJ0100　2018年9月

1. 概要

本ホワイトペーパーでは割込み性能を向上させる HW-RTOS 特有の二つの機能を紹介する。

一つめは HW ISR 機能である。HW ISR は ISR (Interrupt Service Rotutine) 機能を定型化し、ハードウェア化した機能である。この機能により、割込み信号に対応したタスクを直接かつリアルタイムに起動することを可能にした。また ISR がハードウェア化されるため、アプリケーションソフトウェア技術者はハンドラを開発する手間から解放される。

二つめはダイレクト・インタラプト・サービスである。この機能は、OS 管理外割込みハンドラからシステムコールを発行する機能と同等の機能を提供する。この機能により、従来のソフトウェアでは不可能であった、OS 管理外割込みハンドラとタスクが同期・通信可能となる。すなわち、マルチタスク環境で OS 管理外割込みハンドラを利用できるようになる。これを利用することにより、高精度サーボモータのような極めて小さい割込みジッタを要求するアプリケーションであっても、マルチタスク環境でソフトウェア開発を実現することができる。すなわち従来 Non-RTOS であったシステムをマルチタスク環境に移行し、高いソフトウェア開発効率およびシステム信頼性を実現できる。

2. 割込みに関する一般的な知識

まず共通認識を得るため、割込みと RTOS に関する一般的な事項について述べる。

2.1. 割込み禁止期間の短縮

図 1 の上の図は割り込みが発生したときの動作を示している。割り込みが発生すると割込みに対応した ISR が起動される。この例では割り込み 1 が発生すると ISR1 が起動され、割り込み 2 が発生すると ISR2 が起動されている。問題は一般的に ISR は割込み禁止状態で動作することである。したがって、図 1 の下の図のように、ISR 1 の処理が長引くと、ISR 2 の処理の開始が遅延してしまう。これは、リアルタイム系において好ましいことではない。

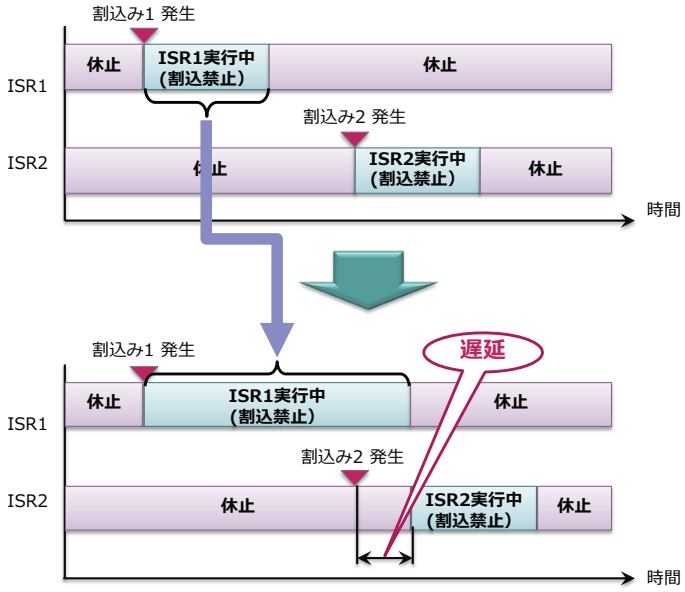

図1 ISRの起動

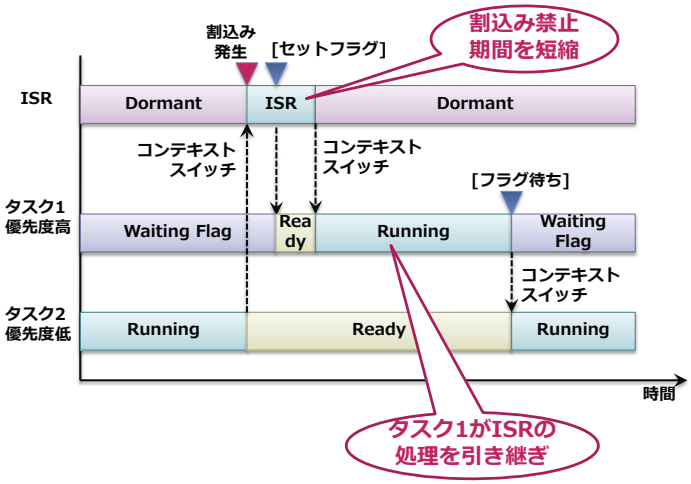

図2 割込み処理のタスクへの引き継ぎ

こうした問題を解決するために、ISR の処理をタスクに引き継ぐ方法がとられる。すなわち、ISR では極めてリアルタイム性の高い処理のみを行い、他の処理は ISR に対応したタスクを起動し、そのタスクに処理を継承するという方法である。図 2 はこれを示している。まずタスク 2 が実行状態にあるとする。ここで割込みが発生すると、ISR が起動される。ISR は、フラグ待ち状態にあるタスク 1 を起動する。このため、セットフラグのシステムコールを発行する。タスク 1 は待ち条件が解除され、Ready 状態になる。ISR はこれ以上仕事をせず、終了する。このとき、今まで実行状

図3　OS管理割込みとOS管理外割込み

態にあったタスク 2 より、タスク 1 のほうが優先度が高いため、タスク 1 が実行される。タスク 1 は、「タスク」であるため、割り込み許可状態で動作する。このため他の割り込みを遅延させることがない。このように ISR の処理をタスクで引き継ぐようにすると、ISR はシステムコールを発行するだけでよく、あとの処理はタスクに引く次ぐことができる。

このような目的で使用されるシステムコールは、セットフラグの他に「セマフォ解放」や「待ち状態の解除」などのシステムコールがよく利用される。

ここでは ISR でシステムコールは発行するだけとしたが、もちろんリアルタイム性の高い処理を ISR の中で実行しても良い。どうしてもすぐに処理をしなければならない処理のみを ISR の中で実行し、その他の処理はタスク 1 で実行することが望ましい。

以上のように割り込み処理において RTOS をうまく利用することにより、優先度に応じた処理をタイムリーに実行することができ、また割込み禁止期間をできるだけ短くすることができる。

2.2.　OS 管理割込みと OS 管理外割込み

割込みには OS 管理割込みと OS 管理外割込みが存在する。2.1 で示した、割込みによって起動されるハンドラは、RTOS が管理している割込みである。この割込みで起動されるハンドラを OS 管理割込みハンドラと言う（この文献では OS 管理割込みハンドラを ISR : Interrupt Service Rotutine と呼ぶことにする）。RTOS と ISR、タスクの優先関係を図 3 に示す。

この図からわかるように、タスクの優先度が一番低く、その次に ISR、そしてその次に RTOS の優先度が高い。RTOS が管理していると言うことは、OS が ISR を起動すると言うことである。すなわち割り込みが発生すると、その割り込みに対応した ISR を RTOS が起動する。図のように RTOS の方が ISR より優先度が高いため、もし割り込みが発生したとき、RTOS が処理中であれば ISR の起動を待たせることになる。したがって、割り込み発生から ISR 起動までには RTOS によるオーバヘッドが存在する。基本的に RTOS の処理の多くはクリティカルな処理であるため割込み禁止状態である。以上のように OS 管理割込みによる ISR の実行は遅延を伴うということを認識しておかなければならない。

もう一つ、OS 管理外割込みが存在する。OS 管理外割込みにより起動されるハンドラを、OS 管理外割込みハンドラという。図 3 で示すように、OS 管理外割込みハンドラは RTOS よりも優先度が高い。したがって、RTOS が処理中であっても、RTOS 処理を即座に中断しハンドラが起動される。この起動には全くソフトウェアが関与しないため、ハンドラの起動時間は CPU で定義されるインタラプト・レイテンシに一致する。高速のサーボ・モータ制御など、RTOS によるオーバヘッドが許容できないようなアプリケーションでは、この OS 管理外割込みハンドラを使用することがある。

ISR と OS 管理外割込みハンドラのもう一つの違いは、ハンドラ中からシステムコールを発行できるか否かである。ISR は RTOS の管理下で動作しているため、システムコールを発行できる。しかし OS 管理外割込みハンドラは、OS がクリティカル処理を実行中であっても起動されるわけであるから、当然 RTOS の機能を利用することはできない。したがって、図 2 で示したように、システムコールを使ってハンドラ処理の一部をタスクに引き継ぐということができない。割込みハンドラ中からシステムコールを発行できないと言うことは大変大きな問題である。一般的に全てのソフトウェア処理は、元をたどれば何らかの割込みに行き

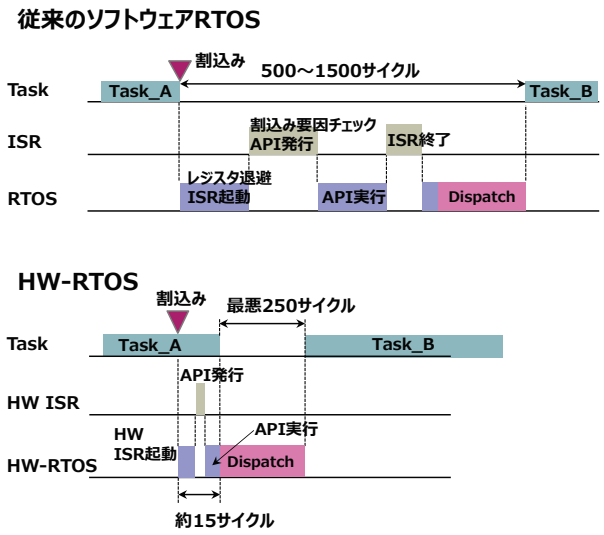

図4　割込み処理（ディスパッチなし）

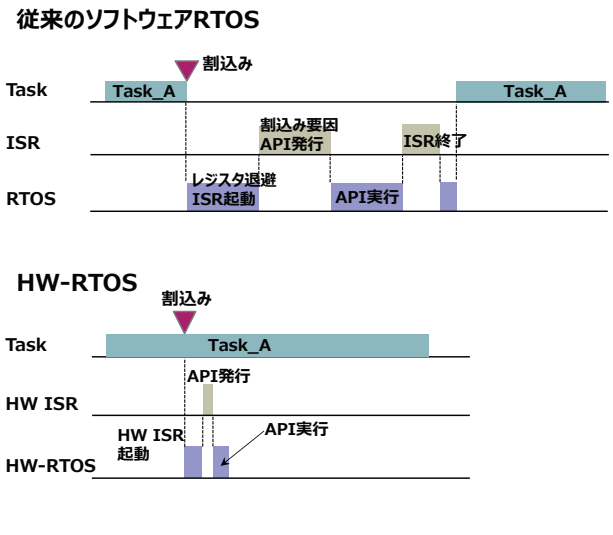

図5　割込み処理（ディスパッチあり）

着く。逆に言うと、一つの割り込みが発生すると、ある処理が起動され、その処理がまた他の処理を起動するという連鎖でシステムが成り立っている。したがって、ハンドラからシステムコール、特に同期・通信関連のシステムコールが発行できないというのは、ある意味致命的である。

以上、OS 管理割込みと OS 管理外割込みについて説明したが、整理すると以下の通りである。OS 管理割込みは、ISR 起動までのオーバヘッドが存在するが、ハンドラ中からシステムコールが発行可能である。OS 管理外割込みでは、ハンドラ起動時間は CPU のハードウェア性能と同等で最高速であるが、ハンドラからシステムコールが発行できない。

3.　HW ISR

3.1.　HW ISR の機能と動作

HW ISR とは、ISR 処理をハードウェア化した機能である。具体的に言うと、発生した割り込み信号に対応したシステムコールを、HW-RTOS 内部で発行させる機能である。図 2 で説明したように、ISR において発行したセットフラグシステムコールのような、他のタスクを起動するためのシステムコールをハードウェアが発行する。こうすることにより、図 2 の ISR 処理が全てハードウェアで実現できる。

図 4 を用いてわかりやすく説明する。図 4 の上の図「従来のソフトウェア RTOS」は、従来のソフトウェア RTOS における割り込み時の処理のタイミングチャートである。割り込みが発生すると、今まで実行状態であった Task A は中断され、処理は RTOS に移る。RTOS では Task A で使用していたコンテキストを退避し、ISR を起動する。ISR では割込みをチェックし、割込み信号に対応するシステムコールを発行する。図 2 の例ではセットフラグである。システムコールが発行されると、処理は RTOS に再び移動し、RTOS ではその発行されたシステムコールを実行する。システムコールが終了すると戻り値を ISR に返し、ISR は終了する。このシステムコール実行の結果、実行中のタスクより優先度の高い Task B が Ready 状態になったとすると、ここでタスク切換が必要である。RTOS は Dispatch 処理を行い、Task B が起動される。

以上が、従来のソフトウェアにおいて割り込みが発生したときの処理である。割込みが一度発生するとこのようにいろいろな処理が行われ、図のように 500 サイクルから 1,500 サイクルの CPU 時間を消費する。

図 4 の下の図は HW-RTOS における HW ISR 機能を用いた例である。Task A 実行中に割り込みが発生すると、HW-RTOS が起動される。さらに HW-RTOS 内の HW ISR 機能が呼び出され、HW ISR では割り込み

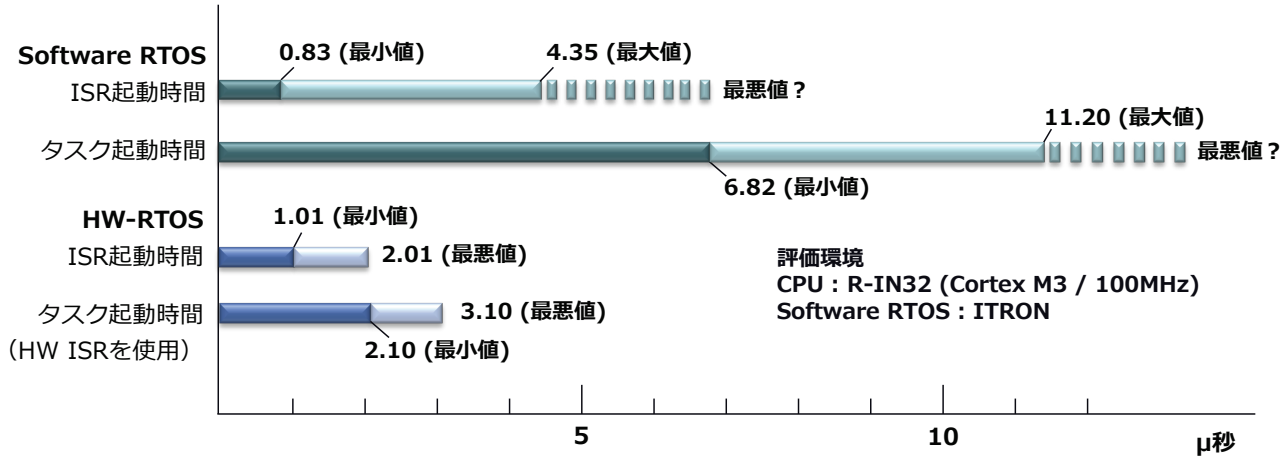

図6　割込み応答性能

信号に対応するシステムコールを発行する。この発行を受け、HW-RTOS はこのシステムコールを実行する。HW-RTOS はこのシステムコールの結果、実行中のタスクより優先度の高い Task B が Ready 状態になったことを検知すると、ライブラリソフトウェアに対しタスクスイッチを通知し、ライブラリソフトはこの指示に従いディスパッチ処理を行い、Task B が起動される。

以上が HW ISR を利用した HW-RTOS の処理である。図のように HW-RTOS が処理を行っている時間は 15 サイクル程度であり、他のほとんどの期間はライブラリソフトウェアが実行するディスパッチに消費される。もう一つ注目すべきポイントは、割り込み発生からディスパッチ開始までの期間、CPU は Task A の処理を続けていると言う点である。これは HW-RTOS と CPU が別のハードウェアであるため、並行動作ができるためである。

図 5 の上図は、図 4 の上図と同様、従来のソフトウェア RTOS における割込み発生時のタイミングチャートであるが、こちらは、ISR がシステムコールを発行し、システムコールが実行された結果、タスク切換が必要ない場合の事例である。ISR 処理終了後、処理がタスク A に戻っている。

図 5 の下図は、同様にタスク切換が必要ない場合の HW-RTOS のタイミングチャートである。図を見るとわかるように、CPU は Task A の処理をし続けている。これは HW-RTOS と CPU が並行動作可能であるため、このような処理が可能となっている。割り込みが発生しているにもかかわらず、CPU の処理を中断させることがなく、また全く CPU にオーバーヘッドがかからないと言うのは驚くべきことである。

3.2.　割り込み応答性能

図 6 に割り込み応答性能の測定結果を示す。 CPU は Cortex M3（100MHz）を使用し、ソフトウェア RTOS は ITRON を使用した。また、HW-RTOS も 100MHz のクロックで動作させた。

ISR の起動までの時間はソフトウェア RTOS では 0.83～4.35 マイクロ秒、HW-RTOS は 1.01～2.01 マイクロ秒であった。また、ISR でシステムコールを発行し、次のタスクが起動される迄の時間は 6.82～11.2 マイクロ秒、HW-RTOS は 2.10～3.10 マイクロ秒であった。タスク起動迄の時間測定では HW-RTOS は HW ISR を使用している。図で示したように、ソフトウェア RTOS の最大値は測定環境における最大値であり、最悪値は不明である。一方 HW-RTOS は最大値が理論上の最悪値であり、これ以上の値はとることがないと言うことも大きな特徴である。

以上のように、HW-RTOS および HW ISR を使用することにより大幅な割り込み応答性能の改善を得るこ

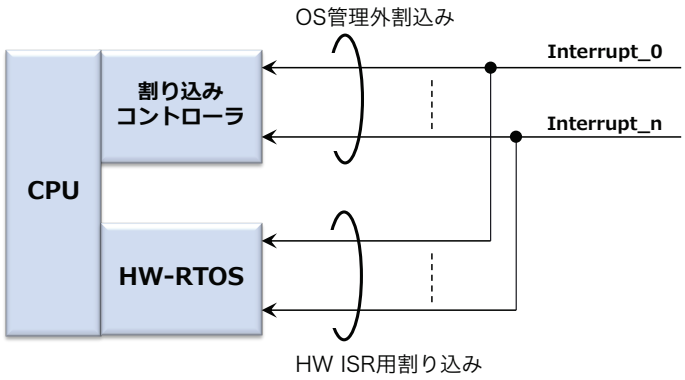

図7　割込みの構成

とができる。またこの測定では現れない、図 5 で示したようなオーバヘッドも大幅に削減されている。

3.3.　HW ISR のアドバンテージ

HW ISR において、発行できるシステムコールは、セットフラグ、セマフォ解放、起床、待ち強制解除の 4 種類であり、各割込み信号線毎にプログラマブルに設定可能である。

HW-RTOS では、HW ISR とソフトウェアによる ISR を併用することも可能である。HW ISR は HW-RTOS 内で実行されるため、ソフトウェア処理を HW ISR に入れ込むことはできない。したがってどうしても ISR でソフトウェア処理を行いたい場合はソフトウェア ISR を起動すれば良い。ただ、多くの HW ISR を使用している設計者はソフトウェア ISR を使用していない。ほとんど多くのアプリケーションにおいて、HW ISR によるタスク起動により十分なリアルタイム性能を得られるためである。もう一つ HW ISR を利用するメリットとして、ハンドラを設計しないで良いというメリットがある。全ての処理をタスクで作成できると言うことはソフトウェア技術者にとって大きな負担軽減になる。

4.　ダイレクト・インタラプト・サービス

4.1.　ダイレクト・インタラプト・サービスの概要

2.2 において、OS 管理割り込みはハンドラ内からシステムコールが発行でき、処理をタスクに引き継いでいけるが、割り込み応答速度が遅い、また一方 OS 管理外割込みは、割り込み応答速度が高速である一方タスクに処理を継承することができず、ソフトウェアシステムとして大きな課題が残ることを説明した。一方で、HW-RTOS の場合は割り込み応答速度が高速であり、HW ISR を利用して割り込みからタスクをダイレクトに起動しても十分なリアルタイム性が得られる。しかし、それ以上のリアルタイム性が必要な場合も考えられる。例えば、割り込みジッタが数十～数百ナノ秒を実現するためには HW ISR の性能では満足できない。

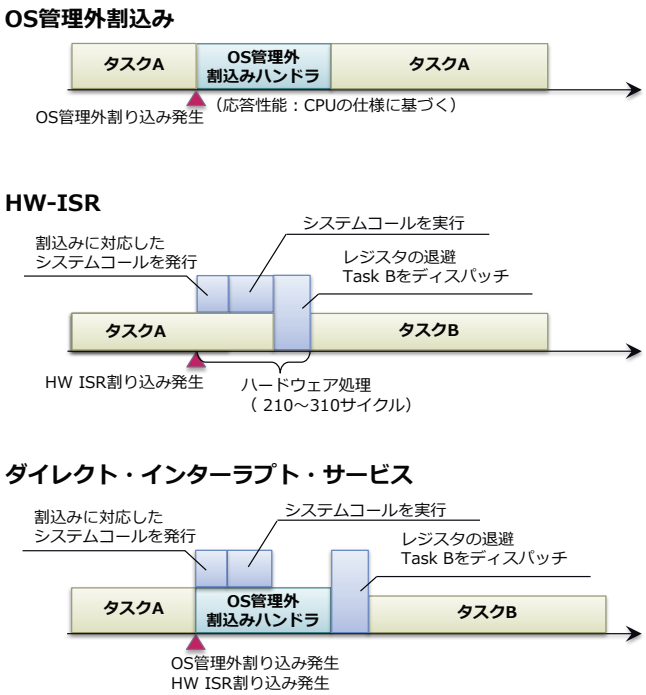

図8　ダイレクト・インタラプト・サービス

HW RTOS では OS 管理外割込みと同等の性能でかつ、タスクに処理を継承できる機能を提供している。これを「ダイレクト・インタラプトサービス」と呼ぶ。

図 7 は、HW-RTOS における割り込み関連の模式図である。インタラプト信号は、割り込みコントローラと HW-RTOS 双方に入力されている。割り込みコントローラに入力された信号は「OS 管理外割込み」として使用する。また HW-RTOS に入力されている割り込み信号は、HW ISR を起動するために使用する。それぞれの入力部にマスクレジスタがあり、入力信号を選択できるようになっている。一方をスルーに設定した場合、他方はマスクに設定する、と設定するのが普通の

利用方法である。しかし、双方をスルーに設定することにより、ダイレクト・インタラプト・サービスが実現できる。図 8 でこれを説明する。

図 8 の上図のタイミングチャートは。「割り込みコントローラ」のみ割込み信号線をスルー状態にした場合である。このとき割り込みが発生すると、OS 管理外割り込みハンドラが起動される。図 8 の真ん中の図のタイミングチャートは「HW-RTOS」のみ割込み信号線をスルーにした場合である。このとき、HW ISR が起動される。

割込み信号線の設定を、「割り込みコントローラ」、「HW-RTOS」共にスルーに設定すると、図 8 の下図のように動作する。すなわち、割込み信号発生と共に OS 管理外割込みハンドラが起動し、同時に HW-RTOS は内部的にその割込みに対応したシステムコールを発行する。このシステムコールは OS 管理外割込みハンドラと並行して実行され、この例ではタスク B が WAIT 状態から解除される。OS 管理外割込みハンドラ終了と同時にタスク B が起動される。

4.2. ダイレクト・インタラプト・ハンドラのアドバンテージ

このように、ダイレクト・インタラプト・ハンドラを利用することにより、OS 管理外割込みと割込みに対応したタスクの双方を起動することができる。これは OS 管理外割込みハンドラからシステムコールを発行して他のタスクと同期・通信を実行することと等価である。ようするに、OS 管理外割込みと同等の割り込み応答性能と、ハンドラ中からのシステムコール発行の 2 つを同時に実現することができる。

従来、高精度サーボ・モータ制御など割り込み応答ジッタを数十マイクロ秒以下にしたい場合、マルチタスク環境は諦めざるを得なかった。ところがダイレクト・インタラプト・サービスを利用することによりこれが可能になる。高精度サーボ・モータ制御をしつつ、ネットワーク制御などのマルチタスク処理を一つの CPU で処理することが可能になる。これにより、CPU の使用効率を高めるだけでなく、ソフトウェア開発環境の向上も望める。なぜなら、従来のサーボ・モータ制御では、RTOS が利用できないことから、複数の機能を実装するために、ソフトウエア技術者が職人技を使ってシステムを構築していた。これは、ともすると開発生産性を低下させるだけでなくシステムの信頼性の低下も招いていた。ダイレクト・インタラプト・サービスを利用することにより、こうした問題点を全て解決することができる。

5. 結論

割込みハンドラは一般的に割込み禁止状態で動作する。この割込み禁止状態をできるだけ短くするため、タスクに処理を継承することが重要である。このため、割込みハンドラから「セットフラグ」や「セマフォ解放」などのシステムコールを発行し、待ち状態のタスクを起動することにより、タスクに処理を継承する。また割込み処理の継承は、「一般に全ての処理は起源をたどれば割込みに行き着く」という原理から大変重要である。HW-RTOS では、HW ISR 機能により、上記割り込み発生から待ち状態タスクの待ち解除までの処理をハードウェアで実現した。このため、割込み信号に対応したタスクを、極めて高速に起動することが可能である。

また、割込み応答ジッタをさらに少なくしたいアプリケーションに対し、HW-RTOS ではダイレクト・インタラプト・サービスを提供する。これにより、OS 管理外割込みハンドラと、その割込みに対応したタスクの双方が起動される。このため従来大きな問題であった OS 管理外割込みハンドラとタスクの同期・通信が可能になり、また OS 管理外割込みハンドラからタスクに処理を継承できないという課題を完全に解消することができた。

このように HW-RTOS は単純にシステムコール処理をハードウェア化しただけでなく、割り込み応答性能の最少化とソフトウェアシステムとしての利用しやすさの双方を満足させることに成功した。

6. 参考文献

[1] N. Maruyama, T. Ishihara, H. Yasuura,“ An RTOS in Hardware for Energy Efficient Software-based TCP/IP

Processing ”in Proc. of IEEE Symposium on Application Specific Processors (SASP), 2010, pp. 13-18.

[2] N. Maruyama, T. Ishikawa, S. Honda, H. Takada, K. Suzuki, ”ARM-based SoC with Loosely coupled type hardware RTOS for industrial network systems”, in Proc. of Operating Systems Platforms for Embedded Real-Time applications (OSPERT’14), 2014, pp. 9-16.

[3] 丸山修孝，石原亨，安浦寛人 :” RTOS のハードウェア化によるソフトウェアベー ス TCP/IP 処理の高速化と低消費電力化”，電子情報通信学会論文誌 A Vol.J94-A No.9 pp.692-701 (2011).

[4] 丸山修孝,一場利幸，本田晋也，高田広章 :”マルチコア対応 RTOS のハードウェア 化による性能向上”，電子情報通信学会論文誌 D Vol. J96-D No.10 pp.2150-2162 (2013)

[5] 丸山修孝，石川拓也，本田晋也，高田広章，鈴木克信 :”疎結合ハードウェア RTOS 搭載産業ネットワーク用 SoC”，電子情報通信学会論文誌 D, Vol.J98-D No.4 pp.661-673 (2015).