Vol.2014-SLDM-168 No.5
2014/11/26

割込みハンドラのハードウェア化を実現するシステムレベル設計手法

安藤 友樹[1,a)]　本田 晋也[1]　高田 広章[1]　枝廣 正人[1]

概要：本論文は，割込みで駆動する専用 HW を設計可能な，制御システム向けのシステムレベル設計ツールについて述べる．制御システムは複雑化，処理の高度化が進み，それに伴い割込み処理の頻度が増加したことで，プロセッサ負荷の増加，消費電力の増加，割込み処理レイテンシの悪化といった問題が生じている．これらの問題を解決するため，割込みにより処理を開始し，処理中はセンサや入出力ハードウェアといったデバイスへ直接アクセスする専用ハードウェアが求められている．提案手法は，処理とデバイス間の通信，割込み，割込み処理を抽象化した制御システムモデルから，割込みで駆動するハードウェアを含む制御システムを設計可能である．モータ制御システムを対象とした評価実験により，提案手法を用いることで，プロセッサ負荷の削減，消費電力の削減，レイテンシの改善が可能なことを示す．

1. はじめに

近年，System-on-a-Chip の小型化，高性能化に加え，アクチュエータや入出力ハードウェア (HW) といったデバイスの小型化，高性能化により，高度な制御システムが実現可能となった．制御システムでは，制御処理とデバイスとの通信が不可欠である．例えば，典型的なセンサでは，アナログデジタル変換（ADC) 回路により，センサのアナログデータはデジタルデータへ変換される．その後，プロセッサで実行される制御処理がデジタルデータを読込む．また，このような処理とデバイス間の通信だけでなく，デバイスから非同期で通知される割込みも考慮する必要がある．一般に，制御システムは複数のデバイスで構成されており，プロセッサがデバイスから頻繁に通知される割込みを処理する．これら複数の割込みは互いに影響しあい，割込み処理のレイテンシが長くなる原因となっている．さらに，割込み処理によりプロセッサ負荷が増加し，プロセッサの消費電力増加の原因ともなっている．

レイテンシの改善とプロセッサ負荷の増加を解決する方法として，割込み駆動 HW がある．割込み駆動 HW とは，割込み処理に相当する処理を実行する専用 HW の一種である．割込み駆動 HW は割込みにより処理を開始し，処理中はデバイスへアクセスすることが多い．例えば，Atmel の SleepWalking 技術は割込み駆動 HW である [1]．この技術では，まず，割込みにより割込み駆動 HW が処理を開始し，プロセッサの起動が必要かを判断する．プロセッサの起動が必要な場合は，割込み駆動 HW がプロセッサへ起動通知を送信する．一方，プロセッサの起動が不要な場合は，割込み駆動 HW がデバイス操作を含む割込み処理を実行する．プロセッサは起動しないため，プロセッサ負荷が削減される．さらに，プロセッサは低電力モードで待機可能なため，消費電力の削減が可能となる．また，割込み駆動 HW が複数の割込み処理を並列に実行するため，割込み処理のレイテンシが改善される．このように，割込み駆動 HW により以下 3 つの利点が得られる．

- プロセッサ負荷の削減
- 低消費電力化
- 割込み処理のレイテンシの改善

そのため，高性能かつ低消費電力の制御システムを実現する上で，割込み駆動 HW は重要である．

プロセッサと専用 HW で構成されるシステムの設計手法として，システムレベル設計手法がある．システムレベル設計では，まず，ソフトウェア (SW) と HW を意識せず，システム全体を抽象度高く設計する．その後，処理の実装先を示す SW/HW 分割を決定する．SW/HW 分割の変更と評価を繰り返し，システムの詳細な仕様を決定する．このように，高い抽象度にてシステムの性能を評価し，設計後半での手戻りを防ぐことで，システムの設計を効率化する．

システムレベル設計を実現する環境として，多くのシステムレベル設計ツールが提案，開発されてきた．Metropolis[2] は，システムレベルのモデリングとシミュレーション環境を提供し，プロトタイプ実装の合成機能を含まないシステムレベル設計ツールである．また，PeaCE[3]，ARTEMIS[4]，System-On-Chip Environment[5]，System-CoDesigner[6] は，設計するシステムを独自のモデル記述に従って抽象度高く設計し，ターゲットアーキテクチャ向けのプロトタイプ実装を自動合成する機能を含むシステムレベル設計ツールである．しかし，これら既存のシステムレベル設計ツールは，専用 HW の一種である割込み駆動

[1] 名古屋大学，千種区, 名古屋市，464–8601, 日本
[a)] y_ando@ertl.jp

HW の設計に対応していない．

本論文は，制御システムを抽象度高く記述し，割込み駆動 HW を含む実装を自動合成する設計手法を提案する．提案手法は，我々が開発してきたシステムレベル設計ツールである SystemBuilder[7] を制御システム向けへ拡張し，割込み駆動 HW の自動実装を実現した．また，制御システム向けのシステムレベル設計を実現するために，制御システムモデルを提案した．制御システムモデルは，処理とデバイス間の通信，割込みと割込み処理を抽象度高く記述できる．制御システムモデルで記述された割込み処理は，プロセッサ上では割込みハンドラもしくはタスクとして実装され，専用 HW では割込み駆動 HW として実装される．そのため，制御システムで重要な指標である，様々なリアルタイム要件を満たす実装が実現可能である．さらに，設計ツールにより制御システムモデルから実装が自動合成されるため，効率的に制御システムの設計が可能となる．以上より，提案手法の貢献は以下 3 つである．

- 割込み駆動 HW を設計可能なツールの実現
- 制御システムのモデル記述を提案
- 制御システムの効率的な設計（実装の自動合成）

本論文では，まず，2 章で提案手法のコンセプトを述べる．次に，3 章で制御システムモデルの詳細を説明する．4 章で設計ツールによる自動実装を述べ，5 章で設計事例により提案手法の有効性を示す．最後に，6 章でまとめる．

2. 制御システム向け設計ツール

本章では，我々が開発してきたシステムレベル設計ツールである SystemBuilder[7] を制御システム向けに拡張した SystemBuilderCtr について述べる．また，制御システム向けのシステムレベル設計手法を実現するための要求を示す．

2.1 対象とする制御システム

対象とする制御システムの例（モータ制御システム）を**図 1** に示す．モータ制御システムは，目標回転速度でモータを回転させる制御を行う．制御処理は ADC 変換完了割込みで処理を開始し，デバイスから取得したセンサ値を元に，目標回転速度で回転するようにモータの制御値を計算する．その後，デバイスへ制御値を書き込むことでモータ制御を行う．なお，制御処理の一部は専用 HW でも実行可能である．メイン処理は，回転中のモータの監視と，デバイス，三角関数表と目標回転速度の初期化を行う．目標回転速度は変更可能であり，外部から送られたコマンドに従い，コマンド処理が設定する．

提案手法の設計対象は，点線で囲まれた範囲（処理，処理間の通信，処理とデバイス間の通信）であり，プロセッサ，バス，デバイスは含まれない．制御システムを構成する処理は，プロセッサと専用 HW で実行される．デバイスは AD 変換など，HDL で設計した方が効率が良い HW であり，既存のものを利用するため，設計対象に含まない．デバイスはデバイスレジスタを持ち，プロセッサや専用 HW と同一のメモリ空間にマッピングされる．そのため，処理とデバイス間で通信可能である．また，デバイスから非同期の通知 (e.g. 割込み) が処理に対して発生する．本論文では，この非同期の通知をイベントと呼び，イベントの発生により行う処理をイベント処理と呼ぶ．例えばセンサの場合，イベントに相当するのは，ADC 回路がセンサ値のアナログデータをデジタルデータへ変換した際に発生する ADC 変換完了割込みである．また，イベント処理に相当するのは，ADC 変換完了割込みにより処理を開始する制御処理 (割込みハンドラとして実行) となる．

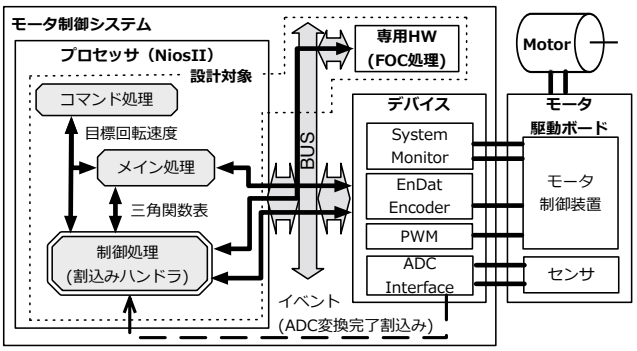

図 1 対象とする制御システムの例：モータ制御システム

2.2 提案手法のコンセプト

SystemBuilderCtr を用いた制御システムの設計フローを**図 2** に示す．SystemBuilderCtr は以下 3 つの入力を取る．

- 制御システムモデル
- マッピング
- アーキテクチャテンプレート

制御システムモデルは，図 1 に示した対象とする制御システムの処理と通信を抽象度高く表現したモデルである．これは，Kahn Process Network[8] を拡張したモデルであり，処理をプロセス，プロセス間およびプロセスとデバイス間の通信をチャネルとして表現する．全てのプロセスは C 言語とチャネルアクセス用 API（C 言語のマクロ）で記述される．制御システムモデルの詳細は，3 章で述べる．アーキテクチャテンプレートには，プロセッサの種類と数，メモリの種類と数，バスの構成，デバイスの情報など，制御システムの実装先となるアーキテクチャの情報が記述される．マッピングは，プロセスの実装先を示す．プロセス毎に SW もしくは HW として実装するか指定する．例えば，図 2 の MODEL-SW は全てのプロセスを SW として実装するマッピングである．一方，MODEL-HW は，Control プロセスを HW として，その他のプロセスを SW として実装するマッピングである．

SystemBuilderCtr は上記 3 つの入力から，マッピングに従い実装を自動合成する．その際，プロセスはプロセッサ

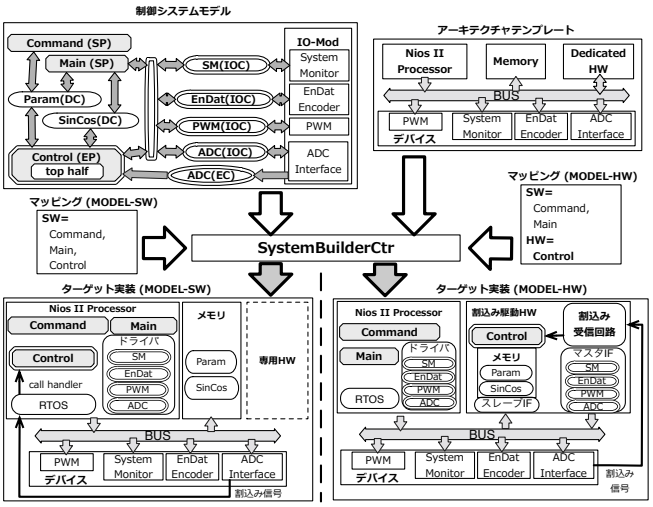

図 2 ツールフローとモータ制御システムの自動実装例

と専用 HW へ，チャネルはメモリへ配置される．チャネルのアクセス API はドライバ，マスタインタフェース (IF)，スレーブ IF として実装される．このように，マッピングに従った実装が自動合成されるため，SystemBuilderCtr を用いることで，マッピング記述を変更するだけで，図 2 に示したように様々な実装を評価，比較することが可能となる．

2.3 制御システムモデルとツール化への要求

前節で述べたコンセプトを実現するためには，図 1 に示す対象とする制御システムを，制御システムモデルとして記述できる必要がある．さらに，そのモデルから実装を自動合成できる設計ツールを実現する必要がある．そのための 5 つの要求を以下に示す．

- 要求 1: イベントとイベント処理を記述可能なモデル
- 要求 2: 処理とデバイス間の通信を記述可能なモデル
- 要求 3: 様々なリアルタイム要件に応じた実装が可能なモデル
- 要求 4: 割込み駆動 HW の実現 (イベント処理の HW 化)
- 要求 5: システム性能の変化が小さなモデル化

要求 1 はイベントとイベント処理をモデルで記述可能にする要求である．また，要求 2 は処理とデバイス間の通信をモデルで記述可能にする要求である．イベント，イベント処理，処理とデバイス間の通信は，図 1 に示す対象とする制御システムに含まれている．これらは制御システムの実現に必須であるため，モデルで記述できなければならない．要求 3 は制御システムで重要な様々なリアルタイム要件への要求である．イベント処理を SW として実装する場合，一般的に割込みハンドラとタスクの組み合わせで実現する．例えば，データの受信処理など，割込み発生直後に処理しないとデータが上書きされる可能性がある場合は，割込みハンドラとして実装する必要がある．一方，データ取得後の計算処理は，プロセッサの余裕があるときに実行すれば良いため，タスクとして実装可能である．このように，制御システム毎に求められるリアルタイム性の要件は異なる．そのため，それぞれの要件に応じた実装を実現可能とする必要がある．要求 4 は割込み駆動 HW を実現可能とする要求である．割込み駆動 HW に対応する処理は，イベント処理である．つまり，要求 3 を考慮すると，イベント処理は，SW では割込みハンドラとタスクの組み合わせで実装可能であり，さらに，HW では割込み駆動 HW として実装できなければならない．要求 5 はモデル化により，システム性能が大きく変化することを防ぐ要求である．なぜなら，制御システムは実行性能が変化することで，対象を制御できなくなる可能性があるためである．

3. 制御システムモデル

本章は 2.3 節で示した 5 つの要求を満たす制御システムモデルについて述べる．まず，制御システムモデルの概要を述べ，その後，詳細を説明する．

3.1 制御システムモデルの概要

制御システムモデルの例を**図 3** に示す．この図は，図 1 に示すモータ制御システムを，制御システムモデルで記述したものである．制御システムモデルは，デバイスを表現した IO-Mod，システムの処理を示すプロセス，プロセス間およびプロセスと IO-Mod 間の通信を示すチャネルで構成される．

プロセスは，標準プロセス，イベントプロセスの 2 種類がある．標準プロセスはイベント処理以外の処理を実現する．イベントプロセスはイベント処理を実現し，トップハーフとボトムハーフで構成される．**表 1** にプロセスの性質を示す．全てのプロセスは SW と HW へ実装可能である．標準プロセスは SW へ実装する場合，タスクとして実装される．一方，イベントプロセスは SW へ実装する場合，トップハーフは割込みハンドラとして，ボトムハーフはタスクとして実装されるため，制御システムモデルは要求 3 を満たす．また，イベントプロセスの HW 化により，割込み駆動 HW を実現できるため，制御システムモデルは要求 4 を満たす．

チャネルは，データチャネル (DC)，イベントチャネル (EC)，IO-Mod チャネル (IOC) の 3 種類がある．DC はプロセス間のデータの受け渡しを実現する．EC はイベントプロセスに対し IO-Mod で発生するイベントの受け渡しを実現する．IOC は IO-Mod とプロセス間で入出力されるデータの受け渡しを実現する．全てのプロセスは基本的に全てのチャネルを操作できる．ただし，イベントプロセスのトップハーフは DC の一部機能 (処理を止める通信機能) が禁止される．また，チャネルはプロセスの起床通知に利用される．DC は標準プロセスの起床通知に，EC はイベントプロセスの起床通知にそれぞれ利用される．次節以降，プロセスとチャネルの詳細を述べる．

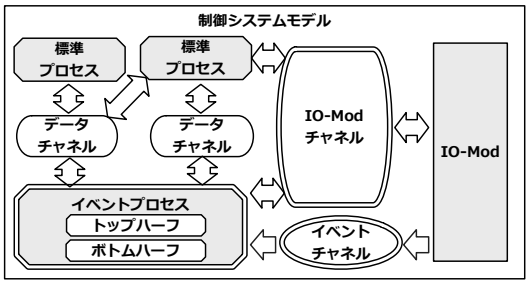

図 3 制御システムモデルの例

表 1 プロセスの性質の比較

プロセス	標準	イベントプロセス	
	プロセス	トップハーフ	ボトムハーフ
可能なマッピング	SW/HW	SW/HW	
SW での実装方法	タスク	割込ハンドラ	タスク
DC の操作	許可	一部禁止	許可
EC/IOC の操作	可能	可能	可能
起床通知チャネル	DC	EC	

3.2 標準プロセス (SP) とデータチャネル (DC)

標準プロセスには，イベント処理以外の一般的な処理を記述する．例えば，モータ制御システムの例では，Command プロセスと Main プロセスが標準プロセスである．標準プロセスは後述のブロッキング通信により起床するため，イベントにより起床するイベント処理は記述できない．

データチャネルはプロセス間でのデータの受け渡しを実現するためのチャネルであり，以下の 4 種類がある．

- ブロッキング通信
- ノンブロッキング通信
- メモリ通信
- 排他制御オブジェクト

ブロッキング通信は First-in-first-out のバッファを持つ同期通信である．プロセスの起床通知に利用可能であり，主にプロセス間の同期を実現するときに用いる．ノンブロッキング通信とメモリ通信は非同期通信であり，それぞれ単一データと複数データの通信に用いる．排他制御オブジェクトは接続されたプロセス間の排他処理に用いる．

3.3 イベントプロセス (EP) とイベントチャネル (EC)

イベントプロセスはイベント処理を実現する．イベントプロセスはトップハーフとボトムハーフで構成される．これらの主な違いは SW での実装方法である．トップハーフは割込みハンドラとして，一方，ボトムハーフはタスクとして実装される．イベントプロセスが HW へ実装される場合は，トップハーフとボトムハーフはまとめて一つの HW モジュールとして実装される．イベントプロセスは以下 3 通りの構成が可能である．

- トップハーフのみ
- ボトムハーフのみ
- トップハーフとボトムハーフの両方

イベントプロセスがトップハーフとボトムハーフの両方で構成される場合，イベント発生によりトップハーフが実行され，その後，ボトムハーフが実行される．この際，トップハーフの戻り値によりボトムハーフの起床の有無を制御できる．トップハーフにてボトムハーフ起床の必要性を判断することで，処理の増加を防ぐことができる．

イベントチャネルは，IO-Mod で発生するイベントの受け渡しを実現するチャネルである．例えば，ADC の変換完了割込みはイベントチャネルとして記述する．1 つのイベントチャネルにより，1 つのイベントと 1 つのイベントプロセスが接続される．イベントチャネルは，接続された IO-Mod で対応するイベントが発生すると，接続されたイベントプロセスへ起床通知を送信する．

このように，イベントとイベント処理をそれぞれチャネルとプロセスとして記述できるため，制御システムモデルは要求 1 を満たす．さらに，イベント処理をトップハーフとボトムハーフに分割することで，様々なリアルタイム要件へ対応できることから，制御システムモデルは要求 3 を満たす．また，イベントプロセスを HW へ実装した場合，割込み駆動 HW として実装されるため，制御システムモデルは要求 4 を満たす．

3.4 IO-Mod と IO-Mod チャネル (IOC)

IO-Mod はデバイスを表現したものである．例えば，IO-Mod の 1 つである ADC Interface は，センサからのアナログデータをデジタルデータへ変換し，デバイスレジスタへ格納する．プロセッサや専用 HW は，バス経由でこのデバイスレジスタへアクセス可能である．

IO-Mod チャネルは，IO-Mod とプロセス間でデータの受け渡しを実現するチャネルである．例えば，ADC Interface からデジタルデータを読み出す場合，プロセスから IO-Mod チャネルが提供する読み出し API を呼び出す．この API は，IO-Mod 内にあるデバイスレジスタからデータを読み出すように実装される．なお，1 つの IO-Mod に対し，1 つの IO-Mod チャネルが必要である．また，1 つの IO-Mod チャネルは複数のプロセスと接続可能である．IO-Mod チャネルにより，プロセスとデバイス間の通信をモデルで記述できるため，制御システムモデルは要求 2 を満たす．

4. 制御システムモデルからの自動実装

図 2 に制御システムモデルからの自動実装例を示す．まず，プロセスの実装方法について述べる．標準プロセスが SW へ実装される場合，リアルタイム OS(RTOS) のタスクとして実装される．標準プロセスが HW へ実装される場合，動作合成により HDL へ変換され，専用 HW へ実装される（図中に対応するものはない）．イベントプロセスが SW へ実装される場合，トップハーフは RTOS の割込みハンドラとして実装され，RTOS が割込みを受け取ると呼び出される．一方，ボトムハーフは RTOS のタスクとして実

装される（図中に対応するものはない）．イベントプロセスが HW へ実装される場合，トップハーフとボトムハーフは 1 つにまとめられ動作合成により HDL へ変換された後，専用 HW へ実装される．この際，割込み受信回路が追加されることで，割込み通知で起動する専用 HW である，割込み駆動 HW として実装される．

次にチャネルの実装方法について述べる．データチャネルはメモリへ配置される．データチャネルに接続されたプロセスが HW 実装される場合はスレーブ IF が生成される．イベントチャネルは，接続されたプロセスが SW として実装される場合は RTOS に割込みハンドラの起動要件として登録され，HW 実装される場合は割込み受信回路に変換される．IO-Mod チャネルは，接続されたプロセスが SW として実装される場合はドライバとして実装され，HW として実装される場合はマスタ IF へ変換される．

5. 評価実験

本章では，図 1 に示すモータ制御システムを対象とした評価実験について述べる．モータ制御システムは，Altera 社の DE2 FPGA ボードへ実装する [9]．FPGA ボード上は NiosII プロセッサが 1 つあり，Avalon バス経由でデバイスと接続されている．ADC Interface からは，16KHz 周期で ADC 完了割込みが生じ，制御処理が実行される．制御処理にて計算された制御値はデバイスに書き込まれることで，モータ駆動ボードに伝えられる．モータ駆動ボードは，設定された制御値に対応してモータを回転させる．

図 2 に，モータ制御システムの制御システムモデルと，ツールにより自動合成される 2 パターンの実装を示す．モータ制御システムの制御システムモデルは 3 つのプロセスで構成される．

- Command（標準プロセス）：コマンド処理に対応
- Main（標準プロセス）：メイン処理に対応
- Control（イベントプロセス）：制御処理に対応

すべてのプロセスは，制御パラメータを保持するデータチャネル Param と接続される．Control プロセスと Main プロセスは，三角関数表であるデータチャネル SinCos と，4 つの IO-Mod チャネル (SM, EnDat, PWM, ADC) に接続される．また，Control プロセスはイベントチャネル ADC に接続される．このモデル記述を入力とし，SystemBuilderCtr により，2 パターンの実装を自動生成した．

- MODEL-SW：全てのプロセスを SW 実装
- MODEL-HW：Control プロセスのみ HW 実装

比較対象として，モータ制御システムのサンプルとして提供された実装を用いる [10]．サンプルでは，制御アルゴリズムの一部である Field Oriented Control(FOC) 処理が専用ハードウェアへ実装可能である．そのため，提供されたサンプルは以下 2 つの実装がある．

- FOC-SW：すべての処理を SW 実装

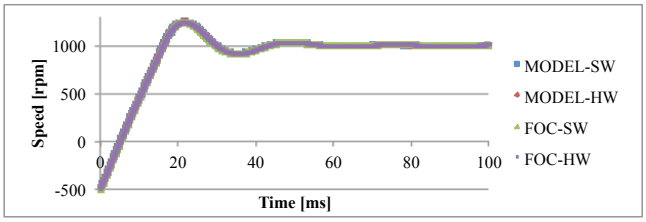

図 4 速度制御の品質比較

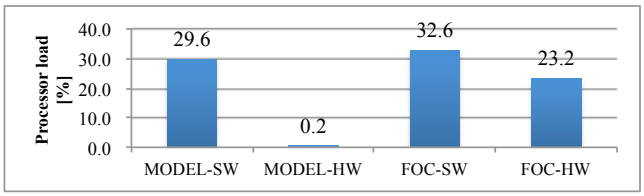

図 5 CPU 利用率の比較

- FOC-HW：FOC は専用 HW 実装，残りの処理は SW 実装

5.1 制御品質の評価

図 4 に目標回転速度を-500 回転/秒から+1000 回転/秒へ変更した際の，モータの回転速度の変化を示す．図より，全ての実装が，目標とする制御速度である+1000rpm へ収束することが確認できる．2 つのサンプル実装の平均値との誤差の二乗平均平方根は，MODEL-SW と MODEL-HW ともに 2.4%であった．よって，モデル化により制御品質が大きく損なわれないことが確認できた．

5.2 割込み駆動 HW によるプロセッサ負荷の削減

プロセッサ負荷を測定するため，Control プロセスのプロセッサ利用率を測定した．なお，Main プロセスと Command プロセスの動作は停止させた．**図 5** に CPU 利用率の比較を示す．Control プロセスをプロセッサで実行している MODEL-SW，FOC-SW，FOC-HW のプロセッサ利用率はそれぞれ 29.6%，32.6%，23.2%であり，MODEL-HW のそれは 0.2%であった．MODEL-HW の Control プロセスは割込み駆動 HW で実装されているため，プロセッサで割込み処理は実行されず，プロセッサ負荷が削減された．その分，プロセッサが低電力モードで待機する時間が長くなり，システムの低消費電力化が実現される．

5.3 実装の効率化

SystemBuilderCtr により自動合成されたコード行数の一覧を**表 2** に示す．表は MODEL-SW と MODEL-HW のそれぞれで自動合成された，設定ファイル，チャネルアクセス API の C 言語実装 (APIs)，C 言語ラッパーファイル (ラッパー)，HDL 記述の行数を示す．MODEL-SW と MODEL-HW は共に，合計で 600 行を越えるコードが自動生成されている．これらは異なる種類のファイルであり，人手による設計ではそれぞれの記述作法を習得する必要が

表 2 SystemBuilderCtr で自動合成されたコード行数

項目	設定ファイル	APIs	ラッパー	HDL
MODEL-SW	31	497	82	—
MODEL-HW	45	713	351	2,011

ある．SystemBuilderCtr を利用する場合，これらのファイルは自動生成されるため，制御システムモデルの記述作法のみを習得すれば良い．また，HDL 記述中の約 100 行はイベント処理の HW 化を実現する記述である．この記述は割込み信号毎に生成されるため，扱う割込みの数が増加することで記述量も増加し，設計者の負担が増加する．一方，SystemBuilderCtr による自動合成では，例え割込みの数が増加しようとも，割込み受信回路は自動合成されるため設計者の負担は増加しない．

次に，割込み処理の HW 化工程を，設計者が人手で行う場合と，SystemBuilderCtr を利用した場合の比較を示す．設計者が行う場合，システム設計，割込み受信回路の設計，マスタ IF の設計，SW-HW 間通信の設計，そして，FPGA への実装まで，全てを設計者が行う必要がある．一方，SystemBuilderCtr を利用する場合，まず，設計者はモデル記述を作成する．その後，SystemBuilderCtr により FPGA 上の実装は自動合成される．SystemBuilderCtr により，設計者はモデル記述のみ作成すれば良く，工程が大幅に削減される．以上のように，SystemBuilderCtr による設計効率化効果は非常に大きい．さらに，HDL 記述，ドライバの設計など，様々な知識を必要とする工程を削除できるため，熟練の設計者だけでなく，多くの設計者が割込み処理の HW 化を実現できる．

5.4 リアルタイム要件の確認

FPGA に ADC 回路の割込み信号と Control プロセスの起動/終了を取得する機構を追加し，実行プロファイルを取得した．Control プロセスの実行時間は，MODEL-SW では 15.5us であった．MODEL-HW のそれは 4us であり，HW 化により 3.9 倍速くなった．

また，割込み発生から Control プロセスの実行開始までのレイテンシを比較した．レイテンシは，MODEL-SW では 1,790ns，MODEL-HW では 30ns となった．クロック周波数は 100MHz のため，30ns は 3 クロックである．このように，イベントプロセスである Control を HW 化することで，レイテンシを僅か 3 クロックに短縮した．SW 実装の割込みハンドラのレイテンシと比較すると，約 1/60 である．本事例のモータ制御など，リアルタイム性が重要な制御処理では，割込み処理開始までのレイテンシは短い程良いとされる．割込み処理の HW 化により，

- 割込み処理開始までのレイテンシ短縮
- 割込み処理全体の高速化

が可能となり，リアルタイム性能の向上が可能である．

6. おわりに

我々は，割込みにより駆動し，デバイスにアクセスしつつ割込み処理を実行する割込み駆動 HW を設計可能な，制御システム向けシステムレベル設計ツールを実現した．また，割込み処理，処理とデバイス間の通信がある制御システムを抽象度高く記述可能な制御システムモデルを提案した．提案手法では，設計ツールにより，制御システムモデルから割込み駆動 HW の実装が自動合成可能である．モータ制御システムを対象とした評価実験により，制御システムモデルから実装した場合，人手による実装と比較し制御品質は悪化することなく，効率的に制御システムを設計可能なことを示した．さらに，割込み駆動 HW により，割込み処理のレイテンシ改善，プロセッサ負荷を削減できることを示した．

謝辞 本研究の一部は，（株）半導体理工学研究センターとの共同研究による．

参考文献

[1] Atmel Corporation, 入手先 ⟨http://www.atmel.com⟩ (2014.09.05)
[2] Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C. and SangiovanniVincentelli, A.: *Metoropolis: An Integrated Electronic System Design Environment*, Computer, 2003.
[3] Ha, S., Kim, S., Lee, C., Yi, Y., Kwon, S. and Joo, Y.P.: *PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems*, ACM Trans. Design Automation of Electronic Systems, 2007.
[4] Pimentel, A.D.: *The Artemis workbench for system-level performance evaluation of embedded systems*, International Journal of Embedded Systems, 2008.
[5] Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Abdi, S. and Gajski, D.D.: *System-on-Chip Environment: A SpecC-Based Framework for Heterogeneous MPSoC Design*, EURASIP Journal on Embedded Systems, 2008.
[6] Keinert, J., Streubühr, M., Schlichter, T., Falk, J., Gladigau, J., Haubelt, C., Teich, J. and Meredith, M.: *SystemCoDesigner - An Automatic ESL Synthesis Approach by Design Space Exploration and Behavioral Synthesis for Streaming Applications*, ACM Trans. Design Automation of Electronic Systems, vol. 14, no. 1, pp. 1–23, 2009.
[7] Shibata, S., Honda, S., Tomiyama, H. and Takada, H.: *Advanced SystemBuilder: A Tool Set for Multiprocessor Design Space Exploration*, In Proc. International SoC Design Conference (ISOCC), pp.79–82, 2010.
[8] Kahn, G.: *The semantics of a simple language for parallel programming*, Proceedings of IFIP Congress 74, pp.471–475, 1974.
[9] ALTERA, 入手先 ⟨http://www.altera.com/⟩ (2014.09.05)
[10] Multiaxis Motor Control Development Board, 入手先 ⟨http://www.altera.com/products/devkits/altera/kit-multi-axis-motor-control.html⟩ (2014.09.05)