Vol.2014-SLDM-168 No.6
2014/11/26

システムレベル設計における制御システム向けプロファイル機構

繆 同徳[1,a)]　安藤 友樹[1,b)]　本田 晋也[1,c)]　高田 広章[1,d)]　枝廣 正人[1,e)]

概要：本論文ではシステムレベル設計における制御システム向けのプロファイル機構を述べる．制御システムを設計する際，センサーやアクチュエータから非同期で通知される割込みと，割込みにより優先的に処理を開始する割込み処理を考慮する必要がある．但し，既存のプロファイル機構は割込みの関連情報を取得できない．本論文では，制御システムの開発を支援するために，割込みの関連情報を取得可能なプロファイル手法を提案する．提案手法を設計事例に適用し，効果を評価した．

キーワード：プロファイラ，制御システム，システムレベル設計，FPGA

1. はじめに

近年制御システムに利用される SoC が大規模，複雑化している．例えば，専用ハードウェア（HW）により処理を高速化するロボット制御用の SoC 等がある．このような複雑な SoC を用いたシステムの設計期間が長くなっており，効率的な開発を実現するために，システムレベル設計ツール SystemBuilder[1] が開発されてきた．SystemBuilder を用いてアーキテクチャ探索を進めて行く際には，性能評価を行う．システムが制約を満たさない場合，処理のボトルネックを解析することにより，ボトルネックとなる処理（プロセス）が見つかり，その性能向上を行う．そのため，アーキテクチャ探索では，ボトルネックの解析手法が重要になる．

ボトルネック解析には HW ミュレーションや実行履歴取得ツールを使い，処理時間の長いプロセスを見つける手法がよく使われる．HW シミュレーションは ModelSim[2] 等の HW シミュレータを使い，PC 上で HW の振る舞いをシミュレーションする手法である．HW シミュレータでは，正確さと実行速度のトレードオフ関係がある．そのため，シミュレーションの正確さを求める場合，シミュレータの実行速度が遅くなる．一方，gprof[3] 等のソフトウェア（SW）での実行履歴取得ツールでは，コンパイルとリンク時にオプションを付けることより，プログラムに実行履歴取得機構を組み込む．設計者は，実行履歴取得機構が組み込まれたプログラムを実機上で実行し，関数の処理頻度や処理時間などプログラムの振る舞いを測定する．実行履歴取得ツールは実機上でプログラムの情報を取得するため，正しいプログラムの実行情報が取得可能である．しかしながら，gprof 等一般的な実行履歴取得ツールは SW しか対応しないため，HW の実行情報を取得できない．SoC の設計は HW と SW の協調設計であるため，ボトルネック解析を行うためには，SW と HW の両方の実行情報を同期して取得する必要がある．

SystemBuilder は，HW へ実装したプロセス（HW プロセス）と SW へ実装したプロセス（SW プロセス）の実行履歴を取得できる，プロファイラを提供している [1]．しかし，従来のプロファイラは制御システムで重要な，非同期のイベント（割込み）と，イベントにより実行される処理であるイベントプロセス（割込みハンドラ等）に対応していない [6]．制御システムのボトルネック解析を支援するため，イベントとイベントプロセスのプロファイルを取得できる制御システム向けのプロファイラを実現する必要がある．本研究は，制御システム関連の情報を抽出することで，システムの性能向上を支援するプロファイラの開発を目的とする．

2. 制御システム向けのプロファイラ

制御システムを設計する際，イベントとイベントプロセスを考慮する必要がある．イベントの発生タイミング，イ

[1] 名古屋大学情報科学研究科
Graduate School of Information Science Nagoya University, Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan
[a)] miaw@ertl.jp
[b)] y_ando@ertl.jp
[c)] honda@ertl.jp
[d)] hiro@ertl.jp
[e)] eda@ertl.jp

ベントプロセスの実行状況を取得，可視化することで，割込みの発生割合や，割込みレイテンシ等の情報を抽出可能である．しかし，SystemByuilder の従来のプロファイラは SW プロセスと HW プロセスの実行情報しか取得できない．

SystemBuilder の制御システム対応に合わせて，プロファイラの制御システム対応が必要になった．そこで，制御システムの特徴を抽出し，プロファイラに必要な機能を定めた．まず，可視化情報としてイベントやイベントプロセス等の情報に対応した．次に，割込み起動回数や割込みレイテンシの最大値等波形表示だけでは分析し辛い情報に対して，統計情報を追加した．そして，実行情報を柔軟に取得するために，プロファイラの停止方法を 4 種類に拡張した．

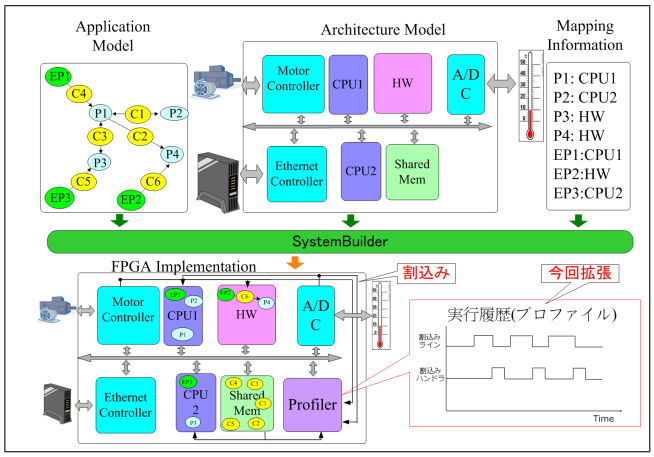

図 1　制御システムの設計と実装の例

2.1　関連研究

プロファイラは SW のみ，HW のみ，SW/HW 混在のアプローチがある．gprof は SW プロファイラの例としてあげられる．gprof はコンパイルとリンク時にオプションを付けることより，プログラムに実行履歴取得用のコードを関数の前後に挿入する．挿入したコードは割込みを生成し，プログラムカウンタ（PC）をサンプリングする．PC の値により，関数の実行時間を計測する．gprof は実行オーバヘッドが大きいため，時間制限が厳しい組込みシステムに向いていない．

SnoopP[4] と LEAP[5] は HW プロファイラの例としてあげられる．SnoopP は FPGA の特徴を利用し，各関数の実行時間を計測する．具体的には，SnoopP はプロファイリングしたい関数と同じ数のセグメントを準備している．セグメントは 2 つのコンパレータと 1 つのカウンタより構成される．コンパレータは PC の値と特定ののアドレス範囲を比較する．PC の値が特定アドレス範囲に入っている時，カウンタはカウントを開始する．LEAP は SnoopP を改良したものである．アドレスハッシュとカウンタ削減等の手法により消費電力削減と，消費電力プロファイルへの対応が実現された．SnoopP と LEAP の特徴は HW からプロファイル情報を取得するため，実行オーバヘッドが生じず，サイクル精度のプロファイルを提供することである．但し，SnoopP と LEAP は SW プロセスのプロファイルしか対応していないため，HW プロセスのプロファイルは取得できない．

2.2　システムレベル設計ツール SystemBuilder

我々の研究室は大規模，複雑なシステムの開発を支援するために，システムレベル設計ツール SystemBuilder を開発した．SystemBuilder を用いた設計フローの例を図 1 に示す．SystemBuilder はアプリケーションモデル（C 言語記述），マッピング情報とアーキテクチャモデルを入力すると，SW 記述，HW 記述，通信チャネル，デバイスドライバ，割込みハンドラ，タスク，IP コア，バスインターフェース回路を自動生成する．生成された SW 記述はコンパイラにより実行可能な SW バイナリファイルに変換される．また，生成された HW 記述は合成ツールにより，HW イメージファイルに変換される．これらのファイルを FPGA ボードにダウンロードすると，システムは実行される．

SystemBuilder は設計効率を向上するために，プロファイル機能を提供する．SystemBuilder が提供しているプロファイラは，SW/HW 混在構造により SW プロセスと HW プロセスのプロファイルを取得している．gprof，SnoopP や LEAP と異なり，SystemBuilder は PC からプログラムの実行プロファイルを取得するのではなく，プロセス間の通信チャネルにより各プロセスの実行プロファイルをサイクル精度で取得できる．プロセスの起動と終了時のチャネル通信にて，データを他のプロセスに渡すため，通信チャネルからプロファイル取得できる．この手法は実行オーバヘッドと面積オーバヘッドがあるが，されらは許容範囲内である [1]．しかしながら，制御システムで重要なイベントとイベントプロセスはチャネルにより実行されため，既存のプロファイラでは，これらのプロファイルを取得できない．

2.3　制御システムの特徴

制御システムは，制御処理を実行する SoC と，センサ，アクチュエータ，ネットワークといったシステム外部との入出力装置で構成される．典型的な制御システムは、まずセンサなど入力装置から情報を取得する．この際，センサからいつ得られるか分からない入力を待ち続けるのは効率的ではないため、センサから非同期で送られる通知（割込み）により、情報を取得し処理を開始する。この割込みにより開始する処理は割込み処理と呼ばれる。次に、取得した情報を元に制御処理を実行し，出力値を求める．その後，求めた出力値をアクチュエータなど出力装置へ出力する．このように、制御システムは入力，制御処理，出力により

フィードバックする機能を提供する。これら一連の流れを繰り返すことで、様々な制御を実現している。

制御処理の一例として、自動車の衝突被害軽減ブレーキがある。このシステムでは，超音波センサなどが入力装置であり，ブレーキが出力装置となる。また、メインプロセッサは衝突被害軽減制御だけでなく、様々な処理を実行することを想定する。超音波センサは自動車とその周囲の障害物の距離を測定する。自動車と障害物の距離がある一定値より短くなった場合，超音波センサはメインプロセッサに対し割込みを発生させる。通常時のメインプロセッサは様々な処理を実行している。しかしながら、超音波センサから割込みが入力されると、割込み処理である衝突被害軽減の制御を優先的に実行する。なぜなら、決められた時間内にこの処理を終えない場合、障害物と衝突するためである。制御処理では、ブレーキを効かせるか効かせないかを判断し、その結果を出力装置であるブレーキに伝える。このように、制御システムにおいては、割込みと割込み処理が重要な役割を担っている。

2.4 制御システム対応への課題

SystemBuilder のプロファイラを制御システムへ対応させるための課題は 2 つある．1 つ目は，イベントやイベントプロセス等のプロファイルへの対応である．2.3 節で述べたようにしたようにイベントやイベントプロセスは制御システムに重要な役割を担っている．そのため，制御システムを設計する時，割込みレイテンシの最大値や割込み発生回数等割込み関連の情報は必要とされる．しかし，現状の SystemBuilder のプロファイラはイベントプロセスとイベントの情報は取得できないため，イベント関連の情報を取得できるようにすることが必要である．

2 つ目は，現状のプロファイラのトレースパターンは固定であり，柔軟性にかけていることである．従来のプロファイラは SW で指定したタイミングでトレースを開始し，実行履歴をメモリに保存する．この手法には，2 つの問題点がある．1 つ目は，FPGA ボードのオンチップメモリは容量の制限があり，全ての情報をメモリに記録することはできないである．2 つ目は，実行履歴の波形表示のみでは，分析が困難なことである．制御システムを設計する時は割込みの発生頻度や割込みレイテンシの最大値等の情報が必要とされる場合がある．しかし，波形表示のみの実行情報では，割込みレイテンシの最大値や起動回数等情報はすぐ分からない．そのため，これらの統計情報を取得できる必要がある．

3. プロファイラの設計と実装

2.4 節に述べた課題に対し，我々は割込み関連情報の取得に着目した．割込み関連情報を取得でき，取得するトレースを柔軟に変更できるプロファイル機構を設計する．

3.1 設計

制御システム向けのプロファイル機構では，以下の機能を実現する．

- 可視化情報の拡張
  - SW プロセスと HW プロセス
  - SW イベントプロセス，HW イベントプロセスと
  - イベント
- 統計情報の拡張
  - 割込み/プロセス発生回数
  - 割込みレイテンシの最大値
- プロファイラ停止方式の拡張
  - SW 停止：従来（図 2(a)）
  - SW 停止：リングバッファ（図 2(b)）
  - トリガー式（図 2(c)）
  - SnapShot 式（図 2(d)）

可視化情報について述べる．制御システムに対応するために，可視化情報を拡張した．SW プロセスと HW プロセスの実行情報は SystemBuilder 従来のプロファイラが対応している．SW イベントプロセス，HW イベントプロセスとイベントは 2.3 節に述べた制御システムの特徴である．制御システムを解析するために，SW/HW イベントプロセスとイベント実行情報を取得する．これらの情報を所得，可視化することで，イベントの発生，イベントプロセスの実行時間と割込みレイテンシ等システムの実行状況を解析できる．また，取得した情報を用いて制御システムのボトルネック解析や他の拡張機能と併せて，制約を満たさない場所を特定できる．

統計情報について述べる．波形表示だけでは分析し辛いために，2 つの統計情報機能を拡張した．1 つ目は割込み/プロセス発生回数統計機能である．FPGA ボードのオンチップメモリでは，全ての実行履歴を保存するメモリが足りない．そのため，割込みの発生回数の統計機能を追加した．利点としては割込みの発生回数しか記録しないために，長時間記録できる．2 つ目は割込みレイテンシの最大値を記録する機能である．この機能では，SW イベントプロセスと HW イベントプロセス両方の割込みレイテンシの情報が取得できる．また，どの程度レイテンシが長いか解析できる．さらに，取得した割込みレイテンシの最大値をトリガー条件（後述）にすれば，ボトルネックを特定できる．

プロファイラ停止方法について述べる．履歴保存用メモリのオーバフロー問題を回避するために，4 つのプロファイラ停止方法を実現する．1 つ目は，従来の SW 停止方式（図 2(a)）である．これは SW が指定したタイミングで記録を開始，停止する方式である．2 つ目は，従来の履歴保存用メモリをリングバッファ化する方式（図 2(b)）である．この方式で，メモリオーバフロー問題を回避し，停止タイミング前の各プロセスの実行状況を取得できる．3 つ目は，トリガー式（図 2(c)）である．トリガー式は特定パ

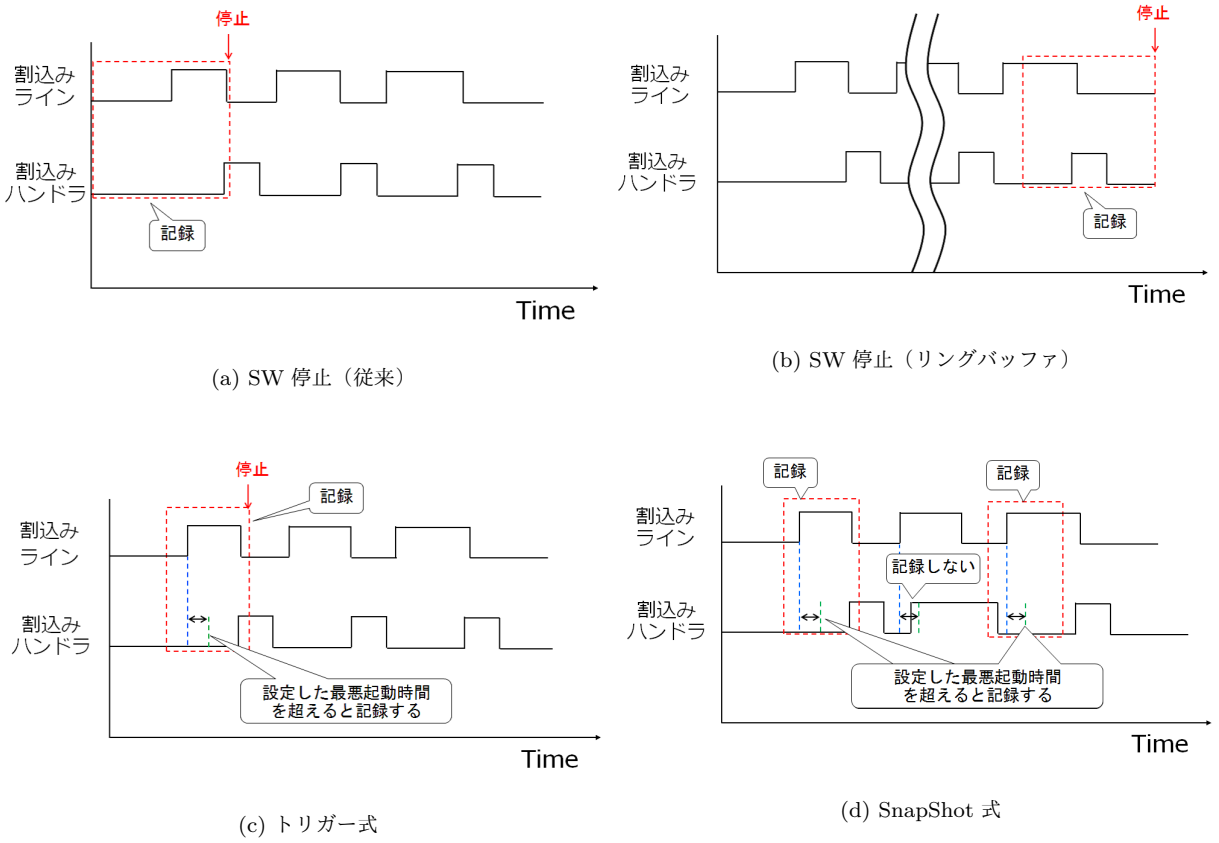

図 2　プロファイラの停止方法

ターンを 1 回のみ記録する方式である．トリガーの起動条件は使用者が設定する．プロファイラは起動後，実行履歴をリングバッファに書き込む続ける．そして，トリガーが起動すると，プロファイラは停止する．トリガー式では特定のパターン前後の実行履歴を確認することができる．そのため，割込みがどのプロセスに影響を与えているかを特定できる．4 つ目は，SnapShot 式（図 2(d)）である．SnapShot 式は特定のパターンを複数回記録する方式である．SnapShot 式を用いて，特定パターンの発生が後に影響するかが分かる．

3.2　可視化情報の実装と取得方法

この節では，制御システム向けのプロファイラ機能のうち，可視化情報の実装と取得方法について述べる．

3.2.1　実装

図 3 は制御システム向けのプロファイラ構成を示す．HW に実装される要素を以下に示す．

- Timer：システムのサイクル数を計測する．
- Tracer：各プロセスのプロファイル情報を取得し，FIFO を通して，Writer に履歴データを渡す．SW プロセス，SW イベントプロセスとの間はバスで接続されており，SW プロセスの実行状態が変化する際，SW プロセスはバス経由で対応するトレースレジスタの状態を変更する．HW プロセス，HW イベントプロセスとイベントは 1 ビットのラインで接続されている．SW プロセスと同様に，実行状態が変わる際，対応するトレースレジスタの状態を変更する．履歴データは，トレースレジスタから取得した各プロセスの実行状態と，Timer から取得した相対時間で構成される．上位 16 ビットが相対実行時間を示し，下位 16 ビットはビット毎に対応するプロセスの状態を示す．
- Writer：Writer は Tracer から貰った履歴データを履歴保存用メモリに書き込む．また，Writer と Tracer を分離することにより，FIFO がフルにならない限り，Tracer は履歴データを作成することが可能である．
- **履歴保存用メモリ**：システムの実行情報を保存する．システム実行終了後，SW が履歴保存用メモリから実行履歴を読み出す．

RTOS[7][8] が割込み要求を受け付けると，まず，現在実行しているタスクと低優先度割込みハンドラを退避させ，高優先度割込みハンドラを実行する．高優先度の割込みハンドラの実行が終了すると，RTOS は退避しているタスクより優先度の高いタスクがあるかを判定する．退避しているタスクより優先度の高いタスクがある場合，ASP はディスパッチャに入り，タスクを切り替える．一方，退避しているタスクより優先度の高いタスクがない場合，退避して

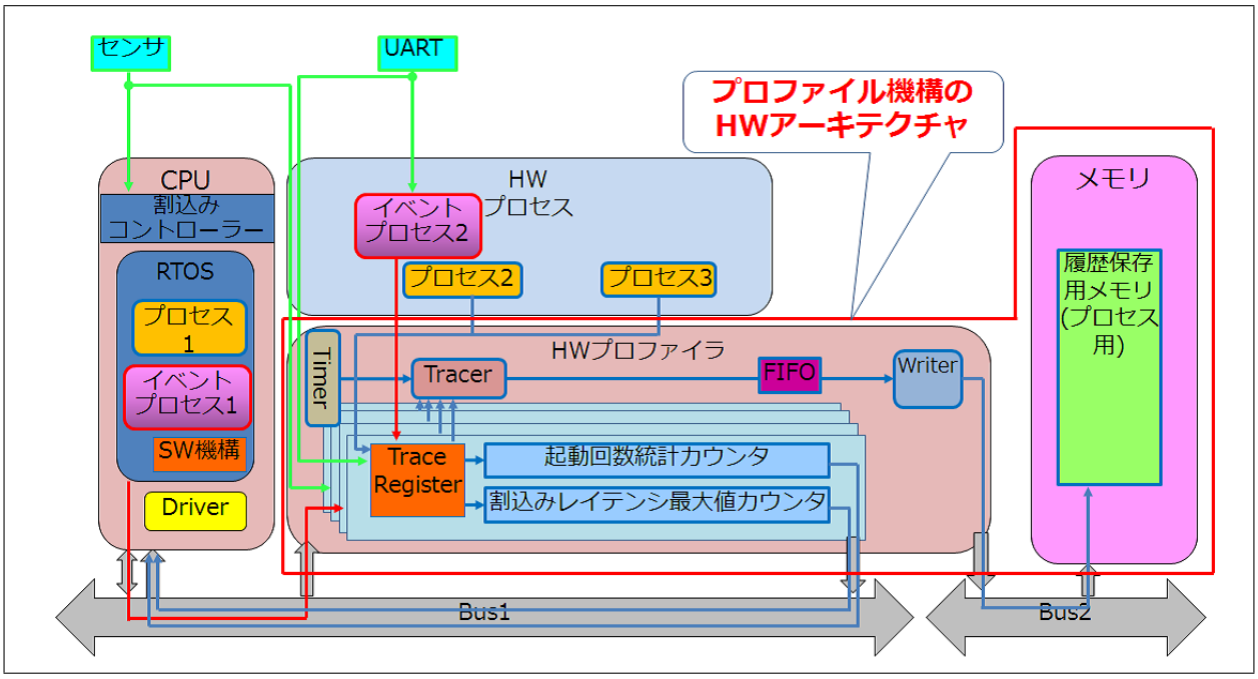

図 3　HW プロファイル機構

いるタスクを復帰し，実行を再開する．従って，SW 実装されたプロセスの実行情報を取得するためには，割込みハンドラの実行前後，ディスパッチャ前後，タスク復帰後にプロセスの実行状態を取得すれば良い．そこで，以下に示す 4 つの関数を追加し，SW プロセスの情報を取得する．

- Enter_Interrupt()：割込み処理開始時に呼び出される．まず実行しているタスク ID・割込みハンドラ ID を取得し，停止するタスク ID・割込みハンドラ ID に対するトレースレジスタを 0（休止状態にする）．次に，実行する割込みハンドラ ID を取得し，割込みハンドラ ID に対応するトレースレジスタを 1（実行状態）にする．
- Leave_Interrupt()：割込み終了時に呼び出される．実行している割込みハンドラ ID を取得し，対応するトレースレジスタを 0（休止状態）にする．
- Enter_DSP()：ディスパッチャ処理開始時に呼び出される．実行しているタスク ID を読み，対応するトレースレジスタを 0（休止状態）にする．タスクを切り替える必要がある場合，ディスパッチャに入り，実行タスクを切り替える．タスクを切り替える必要がない場合，RTOS はタスクを復帰する．
- Leave_DSP()：ディスパッチャ処理終了時に呼び出される．ディスパッチャ後とタスク復帰後は同じタスクを実行するため，実行するタスク ID を読み，対応するトレースレジスタを 1（実行状態）にする．

3.2.2　プロファイル取得の流れ

プロファイル取得の流れを以下に示す．

(1) **システム記述変更**：プロファイルの取得を開始/停止したい箇所で開始関数/停止関数を追加する．

(2) **SystemBuilder による合成**：SystemBuilder で合成する時，プロファイラ有効化オプションを付加する．SystemBuilder はプロファイル取得関数と HW プロファイル機構を合成する．同時に，各プロセス，イベントとイベントプロセスに割り当てられた ID を記録した ID ファイルを生成する．

(3) **バイナリ/イメージファイル生成**：外部ツールとコンパイラを用いて，合成された記述から SW バイナリと FPGA 用の HW イメージに生成する．

(4) **履歴の取得と解析**：生成したファイルを FPGA 上で実行することで，履歴データを取得する．取得した履歴データと ID ファイルを入力として，解析ツールを実行すると波形表示の実行履歴が生成される．設計者は実行履歴等の情報を用いて，ボトルネック解析を行う．

3.3　統計情報の実装

2.4 節に述べた波形表示のみの実行情報では，割込みレイテンシの最大値や起動回数等の統計情報の解析は困難である．本節では，この課題を解決するプロセス／割込み起動回数計測機能と割込みレイテンシ計測機能について述べる．

3.3.1　プロセス/割込み起動回数

プロセス/割込み起動回数カウンタは HW レベルで起動回数を計測したいプロセス/割込みの実行状態と 1 ビットのラインで接続される．カウンタは指定された範囲内かつ実行状態に変化する度に 1 を足すことで起動回数を計測する．計測終了後，SW でカウンタの値を取得し，表示する．

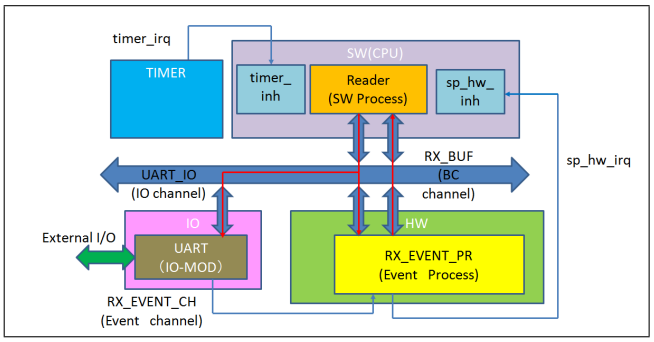

図 4　評価対象システム

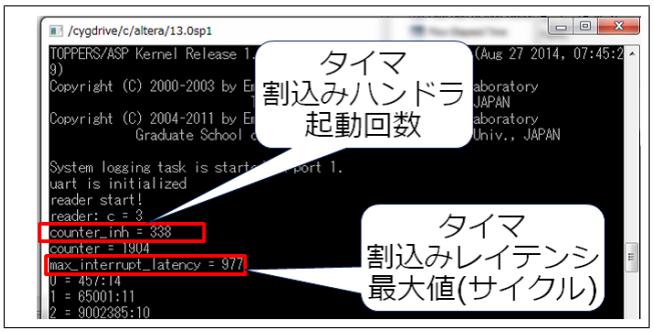

図 5　取得した実行履歴

3.3.2　割込みレイテンシの最大値

割込みレイテンシの最大値計測カウンタは計測したいイベントと HW イベントプロセスを 1 ビットのラインで接続する．SW イベントプロセスはトレースレジスタで接続する．カウンタはイベント発生からイベントプロセスが起動するまでのサイクル数を計測する．計測した割込みレイテンシが前の割込みレイテンシより大きい場合，その値を出力レジスタに上書きする．計測終了後，SW から出力レジスタの値を取得し，表示する．

4.　評価

本章は，3 章で設計，実装したプロファイラの評価について述べる．環境は以下の通りである．

- FPGA：Altera DE-4, StratixIV
- プロセッサ：NiosII（50Mhz）

図 4 は実験システムを示した．このシステムはイベント，SW イベントプロセスと HW イベントプロセスを含む．データが UART へ入力されると，UART は割込み信号を HW イベントプロセスに送信する．次に，HW イベントプロセス（ex_event）が UART のデータレジスタからデータを読み出し，1 を加算し，SW プロセス（reader）に渡す．SW プロセスは受け取ったデータを出力する．また，このシステムは定期的に起動するタイマ割込み（timer_irq），とそのイベントプロセス（timer_inh）,SW-HW 間通信行うための割込み（sp_hw_irq）とそのイベントプロセス（sp_hw_inh）を含む．以上の割込み，イベントプロセスの実行情報を取得することで，機能を検証する．

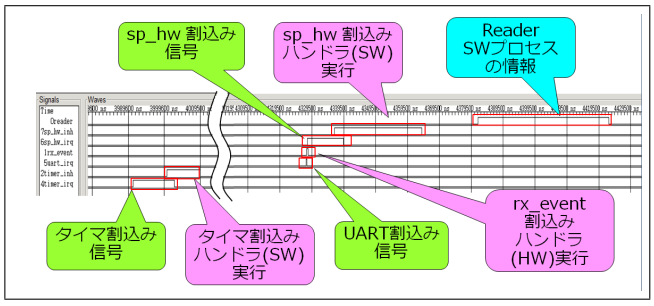

図 6　取得した実行履歴を可視化した結果

図 5 は取得した実行履歴を示す．赤枠はタイマ割込みとタイマ割込みレイテンシの最大値を示す．図 6 は図 5 に取得した実行履歴データを可視化ツールで波形表示した一部を示す．図 6 から割込み，SW イベントプロセスと HW イベントプロセスの実行情報を取得できることを確認した．

5.　まとめと今後の課題

制御システム向けのプロファイラを設計し，SW 機構と HW 機構を実装した．制御システム向けのプロファイラを用いることでイベントやイベントプロセス等，制御システムを開発する時に必要な情報が取得できる．さらに，情報取得の柔軟性と分析効率を向上するために，統計情報機構の設計，実装と，プロファイラの停止方法の設計を行った．今後の課題はプロファイラの停止方法の実装と，自動合成ツールの作成である．

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] S. Shibata, S.Honda, H.Tomiyama, and H. Takada, Advanced SystemBuilder: A Tool Set for Multiprocessor Design Space Exploration, In Proc. International SoC Design Conference (ISOCC), pp.79-82, 2010
[2] Mentor Graphics Modelsim
http://www.mentor.com/products/fv/modelsim/
[3] Gnu gprof.
https://sourceware.org/binutils/docs-2.21/gprof/
[4] L. Shannon and P. Chow, Using reconfigurability to achieve real-time profiling for hardware/software codesign. FPGA'04, pp.22-24, 2004
[5] L. Shannon and P. Chow, Low-Cost Hardware Profiling of Run-Time and Energy in FPGA Embedded Processors, Application-Specific Systems, Architectures and Processors (ASAP), pp.61-68, 2011
[6] Y. ANDO, Y. ISHIDA, S. HONDA, H. TAKADA, M. EDAHIRO, System-level design method considering the interrupt processing, 信学技報, vol. 113, no. 320, pp.119-124, 2013
[7] Toppers ASP kernel
https://www.toppers.jp/asp-kernel.html
[8] Toppers FMP kernel
https://www.toppers.jp/fmp-kernel.html