社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－176 (3)
2008－EMB－ 7 (3)
2008／1／15

システムレベル設計向けプロファイラ

本田 晋也† 柴田 誠也† 冨山 宏之† 高田 広章†
†名古屋大学 大学院情報科学研究科 情報システム学専攻
Email: {honda,shibata,tomiyama,hiro}@ertl.jp

概要

我々は，組込みシステム開発向けに，ソフトウェアとハードウェアの区別ない並列プロセスで記述されたシステム記述から，ソフトウェアとハードウェアに分割された実装記述を自動生成するシステムレベル設計環境（SystemBuilder）を開発している．並列プロセスで構成されているシステム全体の最適化を行うには，システム全体の実行プロファイルを取得して解析する必要がある．加えて本システムでは，並列プロセスはソフトウェア・ハードウェアのどちらとしても実装される可能性があるため，実装先に依存しない実行プロファイルの取得方法が必要となる．そこで我々は，ソフトウェア・ハードウェア混在のシステムの実行プロファイルを低オーバヘッドで取得可能なプロファイラを作成した．本プロファイラにより，容易に分割後のシステムの実行プロファイルを取得することが可能となり，設計効率化が実現できる．開発したプロファイラを SystemBuilder に組み込み，JPEG デコーダの設計に適用し，その効果を評価した．

Profiler for System Level Design.

Shinya Honda† Shibata Seiya† Hiroyuki Tomiyama† Hiroaki Takada †
‡ Dept. of Information Engineering, Graduate School of Information Science, Naogya Univ.

Abstract

We developed system level design environment, named SystemBuilder. One of the most remarkable features in SystemBuilder is that it automatically generates software and hardware implementation from a system level specification with parallel process. In order to optimize the performance of a parallel process system, profile of the system such as execution time and waiting time is necessary. In this paper, we present a profiler for systemlevel design. This profiler provides a process implementation independent profiling with low execution overhead. We use results of the profiling to evaluate software hardware portioning. A JPEG decoder system has been designed in order to evaluate the effectiveness of the developed profiler.

1 はじめに

近年，組込みシステム開発においては，マイクロプロセッサの発達や，動作合成技術を用いた動作レベルのハードウェア設計の導入 [1, 2] により，ソフトウェアとハードウェアの流動化が進んでいる．この様な状況においては，設計制約を満たすよう設計対象の機能を適切にソフトウェアとハードウェアに分割することが重要となる．最適な分割方法の決定には，様々な分割での性能評価が必要である．実装前に評価する手法 [3] も手案されているが，正確な評価を行うためには分割方法に従い実際に実装する必要がある．

従来の設計手法では，設計の初期段階から設計対象をソフトウェア部分とハードウェア部分とに明確に区別し，それぞれを独立して設計する．そのため，分割方法の変更は設計上の大きな手戻りを意味し，多くの開発工数が必要となる．

そこで，我々はこれらの問題を解決するため，システム設計環境（SystemBuilder）[4][5] を開発した．SystemBuilder を用いた設計の設計フローを図 1 に示す．設計者は，設計対象の機能と通信を明確に区別して C 言語により記述する．この記述を入力とし，ソフトウェアとハードウェアへの分割方法を指定すると，指定された分割方法に従って，ソフトウェアとハードウェア及びその間インタフェースを自動合成する．次に FPGA 上で性能評価を行い，設計制約を満たしていない場合は，分割をやり直す．

SystmBuilder は，設計効率化を目的とした開発環境である．そのため，性能評価に関しても，効率のよい手法が求められる．ハードウェア化したプロセスは他のハードウェア化したプロセスやソフトウェア化したプロセスとは並列に動作するために，システム全体の最適化を行うには、個々のプロセスの実行時間を測定するだけでは難しく，他の並列プログ

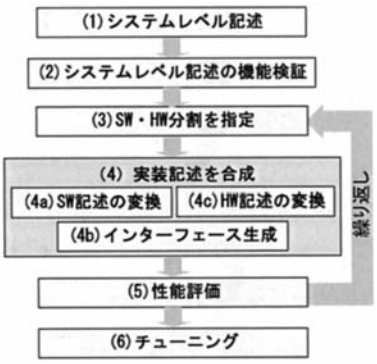

図 1: SystemBuilder を用いた設計フロー

ラミングモデルと同様に [6]，実行プロファイルを取得して解析する必要がある．さらに，本環境を用いた設計では，並列プロセスはソフトウェアとしてもハードウェアとしても実装されるため，実装先に依存しない実行プロファイルの取得方法が必要となる．

そこで我々は、ソフトウェア・ハードウェア混在のシステムのプロファイルを効率よく取得可能なプロファイラを開発した．本プロファイラにより，分割後のシステムの実行プロファイルを容易に取得することが可能となり，設計効率化が実現できる．

以下，本論文では，2 章で SystemBuilder の概要について説明し，次の 3 章でプロファイラの要件について，4 章でプロファイラの設計について述べる．5 章では，JPEG デコーダの設計への適用評価について述べる．

2 SystemBuilder

本章では，本研究で開発したプロファイラを組み込んだ SystemBuilder について述べる．

2.1 システムレベル記述

設計者は設計対象をソフトウェアとハードウェアの区別なく，プロセスとチャネルにより記述する（システムレベル記述）．システム記述の例を図 2 に示す．

プロセスは，プロセスと同じ名前の関数を本体としてこの関数から実行される．関数内部の記述に関してはソフトウェア化する場合には特に制約はない．ハードウェア化する場合の記述制約は，内部で呼び出す動作合成ツールの制約に準ずる．

プロセス間の通信はチャネルにより記述する．チャネルは，5 種類（ノンブロック，ブロック，メモリ，排他制御，リングバッファ）が用意されている．

それぞれのチャネルにはアクセス関数が用意されており，プロセスはこのアクセス関数を呼び出すことによりチャネルを介して通信を行う．

プロセスとチャネルの接続関係は，SDF ファイルと呼ばれる構成定義ファイルに記述する．SDF ファイルには，チャネルのバッファサイズや，バッファ個数といった属性も指定する．

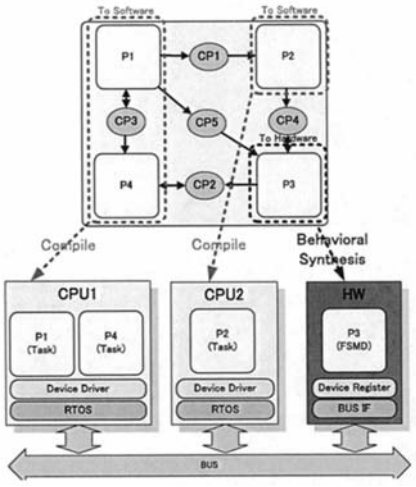

図 2: システム記述とマッピングの例

2.2 実装記述の合成

設計対象のソフトウェアとハードウェアへの分割指定はプロセス単位の粒度で指定可能である．分割指定は SDF に記述する．

2.2.1 ソフトウェアとプロセッサ間通信

ソフトウェア化するプロセスは，それぞれ ITRON 上のタスクとして実現する．プロセッサ内通信となるチャネルは，ITRON と同期・通信機能として実装する．プロセッサ間通信となるチャネルは，プロセッサ内通信と同じ実装方法により実装し，マルチプロセッサ対応 ITRON の機能により実現する．

2.2.2 ハードウェア

プロセス毎に動作合成ツールを呼出しハードウェアを合成する．動作合成ツールとしては，YXI 社の eXCite[7] を用いる．通信に関しては，チャネルとの接続情報から動作合成ツールのサポートする通信インタフェースを割り当てる．また，合成したプロセス間を接続する HDL ファイルを生成する．

2.2.3 インタフェース

ソフトウェアとハードウェア間のインタフェースとなるチャネルに対して（図 2 では CP2，CP4，CP5），デバイスドライバとバスインタフェースを生成する．

デバイスドライバには，アクセス関数，割込みハンドラ等が含まれる．手作業による設計では，設計者はハードウエア側のレジスタ構成，アドレス値，割込み番号等を考慮してデバイスドライバを設計する必要がある．一方，本システムではハードウェア構成と整合したデバイスドライバを自動合成するため，設計者はソフトウェア・ハードウェア間の通信の実現方法の詳細を考慮する必要がない．

バスインタフェースとして，バスに接続のためのアドレスデコーダや割込み管理回路と，各チャネルに応じた回路（バッファやメモリ）を生成する．

2.3 FPGA への実装

最終的には，ソフトウェアとハードウェア共に FPGA 上に実装される．具体的には，ソフトウェアはソフトコアのプロセッサ上で実行され，ハードウェアはそのペリフェラルとして実装される．

3 プロファイラの要件

システムレベル設計向けプロファイラの要件を，設計効率化を実現するよう，次のように定めた．

(1) プロセスの開始・停止タイミングの取得

(2) ソフトウェア・ハードウェアの混在したシステムのプロファイルの取得

(3) プロセスレベルのプロファイルの取得

(4) 低オーバヘッドでプロファイルを取得

プロセスは，チャネルを介してデータをやり取りして処理を進める．ある処理がボトルネックとなった場合，そのプロセスと通信する他のプロセスは待たされることになる．そのため，プロセス毎に (1) に示した開始・停止のタイミングを取得することにより，どのプロセスがボトルネックになっているかを特定することが可能となる．

プロセスは，ソフトウェアとしてもハードウェアとしても実装可能であるため，(2) が必要である．ソフトウェアのみのプロファイルを取得するならば，ソフトウェアプロファイラを，ハードウェアのプロファイルを取得するならば，RTL シミュレータの実行履歴を用いればよい．しかしながら，ソフトウェア，ハードウェアそれぞれのプロファイラを連動することは難しく，同じ時間軸のデータを取得することは困難である．そこで，新たにプロセスの実装先に依存しないプロファイルの取得方法が必要となる．

既存のソフトウェア・ハードウェア用のプロファイラは，関数レベルやモジュールレベルでプロファイル情報を取得する．これらの情報からプロセスの情報を得るのは，設計者にとっては困難であり，設計効率を悪化させてしまう．そのため，設計ツールによって，効率よくプロセスレベルのプロファイルを取得するしくみが必要となるため，(3) を挙げた．

(4) に関しては，プロファイルを取得することにより，プロファイル対象の実行時間が大きく変わってしまうと正確な評価ができないため，特に実行オーバーヘッドを低く抑えることが重要となる．

4 プロファイラの実現

本章では，前述の要件に従い開発したプロファイラの詳細について述べる．

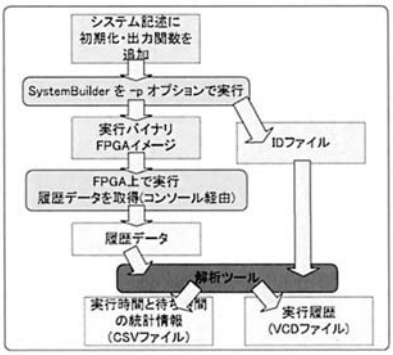

図 3: プロファイルの取得の流れ

4.1 プロファイルの取得の流れ

プロファイルの取得まで流れを図 3 に示す．まず設計者はシステム記述を変更する必要がある．具体的には，プロファイルの取得を開始したい箇所で初期化関数を，プロファイルの取得を停止し出力したい箇所で出力関数を呼び出す．これらの関数はソフトウェアとするプロセス内に記述しなければならない．

このシステム記述を入力として，プロファイラを有効化する”-p” オプションを付加して SystemBuilder を実行する．SystemBuilder は，プロファイル取得ルーチンと，後述するプロファイルハードウェアを含めた記述を合成する．同時に，各プロセスに割り当てられた ID を記録した ID ファイルも生成する．合成された記述は，外部の合成ツールやコンパイラを用いて最終的に，ソフトウェア側は実行バイナリ，ハードウェア側は FPGA イメージが生成される．

次に，生成された FPGA イメージで FPGA をコンフィギュレーションする．そして，その上で実行バイナリを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，CSV 形式の各プロセスの実行時間と待ち時間の統計情報と，VCD 形式の実行履歴が生成される．

設計者は，初期化と出力関数を記述するだけで，容易にプロセスレベルのプロファイルを取得できるため，要件 (3) を満たしている．

4.2 プロファイルの取得タイミング

要件 (1) から，プロファイラでは，各プロセスの実行開始タイミングと，待ち状態の開始タイミングの情報を取得する必要がある．

プロセスは，チャネルを呼び出すことにより待ち状態となり，待ち状態が解除されると実行が再開される．具体的には，ブロックする可能性があるチャネルを呼び出し，チャネル内のバッファがエンプティの場合に待ち状態となる．そして，他のプロセスによりチャネルにデータが書き込まれて待ち状態が解除されると実行を再開する．

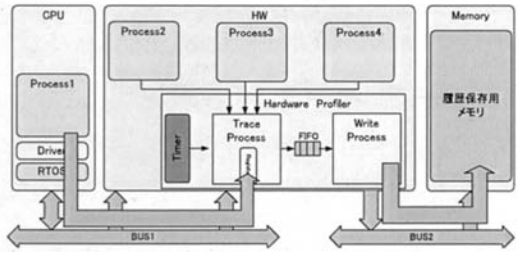

図 4: ハードウェアプロファイラ

そのため，チャネル内で状態が変化する場合に，変化する状態を記録すればよい．なお，待ち状態になる可能性があるチャネルは，ブロックチャネル，排他制御チャネル，リングバッファ制御チャネルの 3 種類のチャネルである．

4.3 ハードウェアプロファイラ

要件（2）より，取得するプロファイルは，プロセスの実装先がソフトウェアでもハードウェアでも同じ時間軸で取得する必要がある．また，要件（4）より，低実行オーバーヘッドで取得する必要がある．これらの要件を満たし，本システムでは最終的に FPGA 上にシステムを実装することを利用して，ハードウェアプロファイラを用いることとした．ハードウェアプロファイラを組み込んだシステムを図 4 に示す．

ハードウェアプロファイラは 2 個のプロセス（FSMD）によって構成されている．トレースプロセスは，各プロセスのプロファイル情報を取得する．ソフトウェア化されたプロセス（Process1）との間は，デバイスレジスタで接続されており，ソフトウェア化されたプロセスは，前述のチャネル内で待ち状態になると，このデバイスレジスタに'0' を書き込み，実行を再開すると，'1' を書き込む．ハードウェア化されたプロセス（Process2-4）とは，1bit のラインで接続されている．ソフトウェア化されたプロセスと同様に，チャネル内で待ち状態になるタイミングで'0' を書き込み，実行が再開されると'1' を書き込む．これらの書き込み処理は，SystemBuilder が，合成したチャネル内に自動的に挿入する．

トレースプロセスは，プロセスからの書き込みを常に監視して，書き込みがあった場合は，図 5 に示す履歴データを生成して FIFO に書き込む．履歴データは，上位 16bit に前回の履歴データ取得タイミングからの相対時間を記録する．下位 16bit は，ビット毎にプロセスの状態を対応させ，プロセスが実行状態なら'1'，待ち状態なら'0' とする．ビット毎にプロセスの状態を記録するため，同じタイミングでプロセスの状態が変化した場合は，両方の状態の変化を同じ履歴データ内に記憶可能であり，履歴データの効率化が可能である．

ライトプロセスは，FIFO から履歴データを読み込み，それを履歴保存用メモリに書き込む．メモリへの書き込みを，トレースプロセスと分離することにより，トレースプロセスは，FIFO がフルにならない限り，3 サイクルで，履歴データを作成することが可能である．ライトプロセスと履歴保存用メモリとは，専用のバスで接続されているため，書き込みによるシステムの実行時間への影響はない．

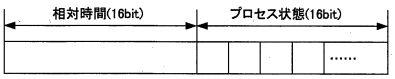

図 5: 履歴データのフォーマット

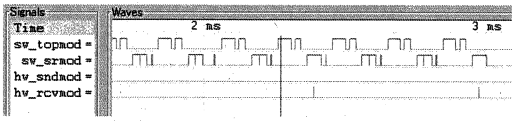

図 6: 実行履歴の例

ハードウェアプロファイラ自体も SystemBuilder で設計しているため，FIFO サイズやプロセスの内容は容易に変更することが可能である．

4.3.1 履歴データの処理

取得した履歴データは，そのままの形では設計者にとって利用が困難であるため，ツールにより変換する．変換するデータは 2 種類であり，1 つ目は，プロセス毎の実行時間と待ち時間の統計情報である．統計情報は，CSV フォーマットのファイルとなっている．2 つ目は，各プロセスの実行履歴を可視化するための Value Change Dump（VCD）ファイルである．VCD ファイルは各種ツールで表示することが可能である．図 6 に実行履歴の例を示す．左側に列挙されているのがプロセスであり，その左側は，プロセスの状態を表しており，実行状態を High で，待ち状態を Low で表現している．

5 JPEG デコーダによる評価

プロファイラの評価として，JPEG デコーダのデザインを用いて，実行オーバーヘッド及び，デザインの解析や最適化の効率化について評価した．

JPEG デコーダの概要図を図 7 に示す．JPEG デコーダは，8 個のプロセスにより構成されている．このうち，decoder, iquantize, idct, pshift, yuv2rgb の 5 個のプロセスは，ハードウェアとしても実装可能である．プロセス間を接続するチャネルはプロセスが並列動作した場合に，各プロセスがパイプライン動作可能なサイズのバッファを持つ．

評価環境には，FPGA に Altera 社の Stratix 1S40 を，プロセッサに Nios2 を用いた．動作周波数は 50MHz である．論理合成ツールには，QuartusII を用いた．実行時間の評価に用いた画像は 240x360 のカラー画像である．

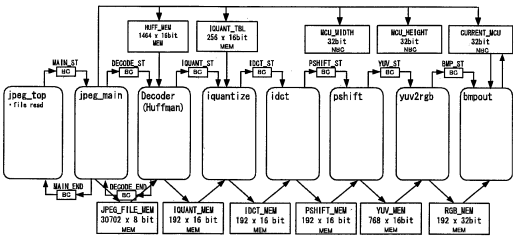

図 7: JPEG デコーダの記述

5.1 オーバヘッド

プロファイラを組込んだ場合の実行オーバーヘッドと面積オーバーヘッドを評価する.

5.1.1 実行オーバヘッド

JPEG デコーダにプロファイラを組み込んでいない場合の実行時間と組み込んだ場合の実行時間の比較を表 1 に示す. AllSw は, 全てのプロセスをソフトウェアとして合成した実装で, AllHw はハードウェア化可能な 5 個のプロセスを全てハードウェアとして合成した実装である.

ソフトウェア化したプロセスのプロファイルの取得には, 状態が変化する毎に, メモリ書き込みが必要になるために, ソフトウェア化したプロセスが多い AllSw では, 7%程度のオーバーヘッドが発生している. 一方, AllHw では, 主だったプロセスがハードウェアとなっているため, 実行オーバーヘッドは発生していない.

表 1: プロファイラの実行オーバヘッド

	無効	有効	増加率
AllSw	1597msec	1712msec	7.2%
AllHw	63msec	63msec	0%

5.1.2 面積オーバヘッド

JPEG デコーダにプロファイラを組み込んでいない場合と, 組み込んだ場合の面積の比較を表 2 に示す.

AllSw の方が増加分が多い. AllSw と AllHw では, 全く同等のプロファイルハードウェアを持つため, 論理合成ツールの最適化による変化だと考えられる.

表 2: プロファイラの面積オーバヘッド

	無効	有効	増加分
AllSw	4482LE	5418LE	936LE
AllHw	13645LE	14117LE	472LE

5.2 プロファイラの効果の評価

プロファイラの効果の評価として, JPEG デコーダに対して, 性能測定では発生している事象の分析が困難な状況をプロファイラを用いることで, 分析が可能となるか試みる.

JPEG デコーダを様々な分割方法で分割して SystemBuilder により実装し, その実行時間を計測した結果を表 3 に示す.

表 3: 分割方法による JPEG デコーダの実行時間

	AllSw	Mix1	Mix2	Mix3	AllHw
decoder	SW	SW	HW	SW	HW
iquantize	SW	SW	SW	HW	HW
idct	SW	HW	SW	HW	HW
pshift	SW	SW	SW	HW	HW
yuv2rgb	SW	SW	SW	HW	HW
実行時間*	1597	1155	1055	573	63
高速化率	0	1.38	1.51	2.79	25.4

*msec

処理が重いハフマン復号 (decoder) や, IDCT (idct) をハードウェア化することで, デコード時間が高速化している. ハードウェア化するプロセスが増加することでも高速化しており, 全てのプロセスをハードウェア化した場合が最も高速である. この事象自体は, ハードウェア化することでプロセス自体の実行時間が高速化するためであり, 通常の性能評価でも説明可能である.

しかしながら, 高速化率に着目すると, decoder 以外のプロセスをハードウェア化した場合 (Mix3) の高速化率は, 2.79 倍であるのに対して, 全てのプロセスをハードウェアした場合 (AllHw) は 25.4 倍と大きく高速化している. この事象は, 個別のプロセスの実行時間のみを考えると, decoder が処理が非常に重く, ハードウェア化による高速化が顕著に現れていると考えられる. しかしながら, decoder のみをハードウェア化した場合 (Mix2) では 1.51 倍しか高速化しておらず, 処理自体のハードウェア化による高速化は idct と大差ないことが分かる. 以上のことから, プロセス毎のハードウェア化による高速化だけでは, AllHw の高速化率は説明できない.

そこで, Mix3 と AllHw のプロファイルを取得して, その結果を比較した. Mix3 のプロファイル結果を図 8 に, AllHw のプロファイル結果を図 9 に示す.

Mix3 のプロファイル結果を見ると, idct 等のハードウェア化されているプロセスは高速に動作しているが, それらの動作間隔, すなわち待ち時間が長いことが分かる. これは, ソフトウェアである decoder がボトルネックとなり, 次に処理するデータが供給されないためである. そのため, ハードウェア化したプロセスが高速であっても, システム全体の高速化率は, 低速な decoder の影響のため低い.

一方, AllHw のプロファイル結果を見ると, decoder をハードウェア化して高速化したことにより, データの供給が高速になり, 他のプロセスの待ち時間が著しく短くなっている. すなわち, ハードウェア化したプロセスがスムーズなパイプライン処理を

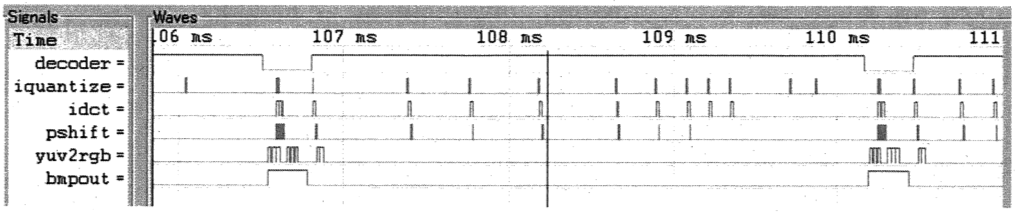

図 8: Mix3 のプロファイル結果

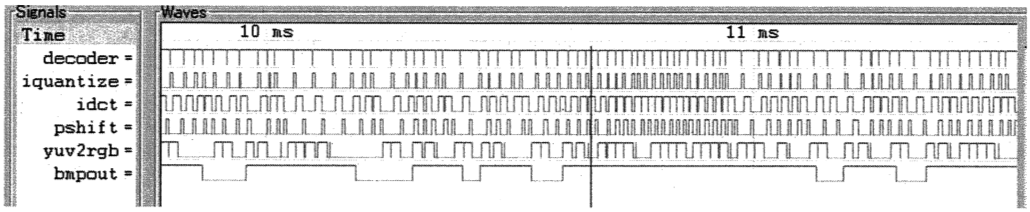

図 9: AllHw のプロファイル結果

実現している．そのため，Mix3 と比較して大幅に高速化している．

AllHw をチューニングして高速するための情報もプロファイル結果から得ることが可能である．

図 9 を見ると，decoder がほぼ待ち時間なしで動作している．それに対してソフトウェアである bmpout も含めた他のプロセスは待ち時間がある．これらから，decoder がボトルネックであり，decoder を高速化すればシステム全体の性能が向上することが分かる．一方，decoder 以外のプロセスを高速化しても，そのプロセスの待ち時間が増加するだけで，システム全体の高速化は実現できない．

以上により，プロファイラを用いることにより，単なる実行時間評価と比較して，発生している事象を分析できることを確認した．また，デザインのチューニングにも有用であることを示した．

6 まとめ

本論文では，組み込みシステム開発の効率化を目的とした SystemBuilder のためのプロファイラについて述べた．本プロファイラは，プロファイルハードウェアを用いることにより，ソフトウェアとハードウェアが混在したシステムのプロファイルを低オーバヘッドで取得することが可能である．JPEG デコーダを用いた評価により，システムのボトルネックを容易に分析できることを示した．

今後の課題としては，ソフトウェア化されたプロセスのプリエンプションのプロファイルへの反映や，プロセスの開始・停止のタイミング情報だけでなく，チャネルの呼び出しや通信データ等の情報が取得できれば最適化に有用な情報となるため，今後これらの情報も取得可能なよう拡張する予定である．

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] K. Wakabayashi, “C-based Behavioral Synthesis and Verification Analysis on Industrial Design Examples,” In Proc. of Asia and South Pacific Design Automation Conference, 2004.

[2] C. Sullivan, and A. Wilson, S. Chappell, “Using C Based Synthesis to Bridge the Productivity Gap,” In *Proc. of Asia and South Pacific Design Automation Conference*, pp. 349–354, 2004.

[3] K. Lahiri, A. Raghunathan, S. Dey, “System-Level Performance Analysis for Designing On-Chip Communication Architectures,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.20, pp.768-783, No.6, 2001

[4] 本田晋也, 冨山宏之, 高田広章, “システムレベル設計環境：SystemBuilder,” 電子情報通信学会論文誌, Vol. J88-D-I, No. 2, pp. 163-174, 2005.

[5] S. Honda, H. Tomiyama, H. Takada, “RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip,” 12th Asia and South Pacific Design Automation Conference(ASP-DAC), 2007.

[6] 上嶋明, 小畑 正貴, 金田悠紀夫, “Omni OpenMP コンパイラ用並列プログラム可視化ツール,” 情報処理学会論文誌, Vol.46, No.SIG_12(ACS_11), pp. 205-213, 2005

[7] http://www.yxi.com