Vol.2014-SLDM-165 No.16
Vol.2014-EMB-32 No.16
2014/3/15

プログラマブルSoCのためのシステム設計環境における フロントエンドの実装と事例評価

東 遼平[1,a)]　高瀬 英希[1]　高木 一義[1]　高木 直史[1]

概要：我々は，プログラマブル SoC 上のシステムをソフトウェア向けの高級言語のみで設計できる環境を検討してきた．本研究では，このシステム設計環境におけるフロントエンドを実装する．フロントエンドでは，設計者が構成情報で指定したソフトウェアとハードウェア間のインタフェースが自動生成される．インタフェースはデータを直接送受信するものを用意し，メモリにデータを格納してそのアドレスを送受信するものを検討する．イベントフラグのビット表現とインタフェースの処理シーケンスを規定し，複数のハードウェアが並列に実行できるようにする．さらに，適用事例を示すことで本環境の有用性を評価する．

Implementation of a Front-End and Case Study of the System Design Environment for Programmable SoC

Ryohei Azuma[1,a)]　Hideki Takase[1]　Kazuyoshi Takagi[1]　Naofumi Takagi[1]

Abstract: We have studied an environment which make it possible for designers to design systems based on programmable SoCs with only programming language for software. In this paper, we implement a front-end of our system design environment. In the front-end, interfaces between software and hardware can be automatically generated according to the given configuration information. One type of the interface is transfer data directly between software and hardware. We also study the interface which transfer the memory address of the data location. By regulating the sequence of the interface and the bit representation of the event flag, multiple hardware modules can operate in parallel. Furthermore, we show the case study of proposed system design environment to assess the usefulness.

1. はじめに

近年，プロセッサと FPGA を 1 チップに集積したプログラマブル SoC が登場し，注目を集めている．本 SoC が従来の SoC とは異なる点として，プロセッサと FPGA の通信が高速であること，および，搭載されるハードマクロのプロセッサが高性能であることが挙げられる．プログラマブル SoC では，プロセッサと FPGA は専用のオンチップバスで接続される．このため，プロセッサと FPGA が別チップとして接続された従来のヘテロジニアスシステムとは異なり，両者の高速な通信が可能である．また，Xilinx 社 Zynq-7000 All Programmable SoC[*1] [1] や Altera 社 SoC FPGA [2] のようなプログラマブル SoC では，高速なクロック周波数で動作できる組込みシステム向けプロセッサが搭載されている．プログラマブル SoC のプロセッサは高性能であるため，異なる負荷と柔軟性が求められるタスク[*2]を混在させたシステムの実現が期待される．このようなシステムの例として，カーナビゲーションシステムが挙げられる．GPS や経路探索のような高速かつ柔軟な処理が求められるタスクはプロセッサ上で，マップ表示のように処理速度が求められるタスクはハードウェアに合成して処理することが全体として効率がよいと考えられる．

ソフトウェアとハードウェアの協調設計において設計者は，システムの設計にあたって双方の深い知識が求められる．プログラマブル SoC はプロセッサと FPGA の接続が固有であり，そのインタフェース設計に関する知識も必要

[1] 京都大学 大学院情報学研究科
Graduate School of Informatics, Kyoto University
[a)] azuma@lab3.kuis.kyoto-u.ac.jp

[*1] 以降，単に Zynq-7000 と記す．

[*2] 本稿では，システム内の実行単位をタスクと定義する．

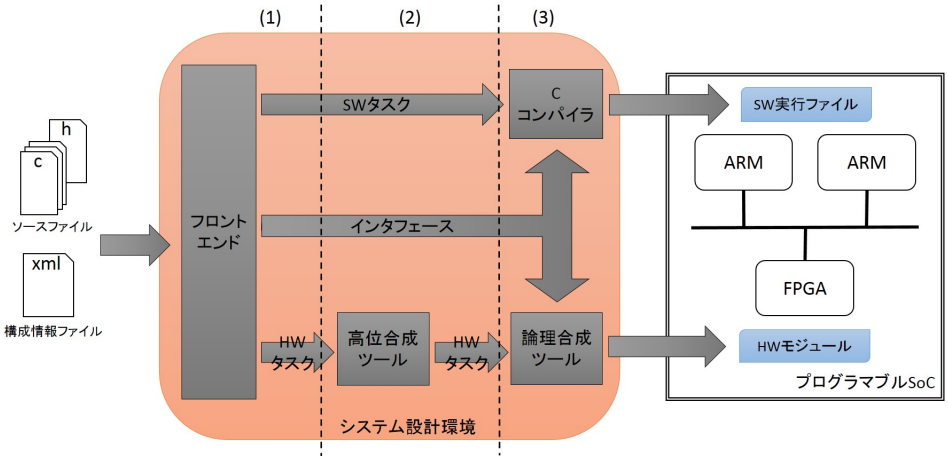

図 1 プログラマブル SoC のためのシステム設計環境の全体像

とされる．既存の協調設計環境として [3], [4] があるが，これらはソフトウェアを FPGA 上のソフトコアで実行することを前提としており，プログラマブル SoC を採用したシステム設計に適用できない．また，配列などで大規模なデータを転送する必要がある場合には，インタフェースにおける通信時間がシステム性能のボトルネックになりうる．

そこで，我々は，プログラマブル SoC のためのシステム設計環境を検討してきた [5]．このシステム設計環境を用いることにより，設計者はソフトウェア向けの高級言語の知識のみでシステム全体の設計が可能となる．システム設計におけるインタフェースの設計では，構成情報においてインタフェースを指定するのみで，システム設計環境がプログラマブル SoC に固有の通信インタフェースを自動的に生成する．ただし，この設計環境を用いて生成可能なインタフェースは，データを直接通信する種類のみで，ハードウェアとする関数は 1 つしか指定できないという制約があった．

本稿では，プログラマブル SoC のためのシステム設計環境内におけるフロントエンドのインタフェースの生成部分を拡張する．複数のハードウェアが存在するシステムに対応するため，イベントフラグのビット表現およびインタフェースの処理シーケンスを規定する．イベントフラグとは，インタフェースで通信される制御コマンドを識別するためのデータである．制御コマンドにより，ソフトウェアやハードウェアの動作を制御する．生成されるインタフェースとしては，[5] で提案した種類に加えて，DDR メモリあるいはプロセッサのキャッシュにデータを格納してそのアドレスを送受信するものを検討する．さらに本稿では，実装した本環境の適用事例を示し，その有用性を評価する．

本稿の構成は以下の通りである．2 章では，プログラマブル SoC のためのシステム設計環境を紹介する．3 章では，これまで検討してきたプログラマブル SoC のためのシステム設計環境のフロントエンドの実装について説明する．4 章では，適用事例を示して，その結果を考察する．5 章では，本稿のまとめと今後の方針を述べる．

```
<conf>
    <device>Zynq</device>
    <sw_task>
        <os>baremetal</os>
    </sw_task>
    <hw_task>
        <name>hoge</name>
        <interface>gp</interface>
    </hw_task>
    <obj_func>speed</obj_func>
</conf>
```

図 2 構成情報の記述例

2. システム設計環境

2.1 全体像

文献 [5] で提案してきたプログラマブル SoC のためのシステム設計環境の全体像を，**図 1** に示す．このシステム設計環境は，タスク解析，タスク・インタフェース合成，および，実行可能モジュール生成からなる．この環境の入力は，ソフトウェア向け高級言語のソースファイル，および，構成情報ファイルである．システム設計環境では，ソースファイルはネイティブな C 言語で記述される．構成情報ファイルとは，設計したシステム内の各関数の処理方法やインタフェースが記述されたテキストデータのことである．構成情報には，ターゲットとなるプログラマブル SoC や OS の有無といったシステム情報，ハードウェアで処理する関数とそのインタフェースの指定，目的関数などの情報を記述する．システム設計環境において構成情報は **図 2** のように XML 形式で記述されたテキストデータとして与えられる．システム環境の出力は，ソフトウェア実行ファイルおよびハードウェアモジュールである．

システム設計環境における処理単位をタスクと呼び，タスクには，SW タスクおよび HW タスクに分類する．前者

は，プロセッサで実行されるプログラムのことである．後者は，FPGA 上に合成されて実行される専用回路のことである．どちらのタスクも，コンパイル時点まではデバイスに依存しない記述である．

2.2 ワークフロー

プログラマブル SoC のためのシステム設計環境のワークフローは，次の 3 つのフェーズから構成される．

（1）タスク解析

構成情報に基づいて入力の C ソースファイルを解析し，SW タスクと HW タスクに分割する．さらに，ソフトウェアとハードウェア間のインタフェースのための情報を生成する．これらの処理は 図 1 のフロントエンド内で行われる．

（2）タスク・インタフェース合成

高位合成を適用して HW タスクを合成する．高位合成とは，ソフトウェア向け高級言語の記述を RTL 記述に合成する技術のことである．高位合成ツールには，C 言語から Verilog HDL を合成する LegUp [6] を採用する．さらに，前フェーズで生成したインタフェースの構成情報を含むテキストデータからインタフェースモジュールを合成する．

（3）実行可能モジュール生成

ターゲットのプログラマブル SoC 上で実行可能な形式のモジュールを生成する．SW タスクはデバイスに対応した C コンパイラにより，プロセッサで実行可能なバイナリにコンパイルされる．一方，HW タスクはブロック RAM，および，インタフェース情報を含むモジュールと統合され，論理合成ツールにより FPGA 上に構成可能な実行形式であるビットストリームが生成される．ここで，C コンパイラおよび論理合成ツールは構成情報内のデバイスの情報により選択される．

2.3 生成されるインタフェース

プログラマブル SoC のためのシステム設計環境では，1 つのシステムに対して制御用途およびデータ用途の 2 種類のインタフェースを生成する．前者は，制御のためのイベントフラグを送受信するためのインタフェースである．後者は，データの受け渡しのためのインタフェースである．

3. システム設計環境のフロントエンドの実装

プログラマブル SoC のためのシステム設計環境のフロントエンドは，タスクの切り分けおよびインタフェース生成の機能を有する．本稿では，複数 HW タスクに対応したインタフェースを生成する手法を提案し，その実装について述べる．

3.1 方針

まず，データを直接通信するインタフェースにおける通信方式を値渡しによるデータ通信と呼び，複数個の HW タスクが存在するシステムの設計にも対応できるようにする．これは，イベントフラグのビット表現およびインタフェースの処理シーケンスを規定することで実現する．次に，DDR メモリあるいはキャッシュにデータを格納してそのアドレスを送受信するインタフェースを検討する．これは，Zynq-7000 の AXI-HP および AXI-ACP ポートを利用して実現する．この通信方式を参照渡しによるデータ通信と呼ぶ．

3.2 複数の HW タスクへの対応

制御用インタフェースにおいて，一度に送受信可能なデータのビット幅は 32 ビットである．制御用インタフェースで送受信されるイベントフラグは，HW タスク開始制御コマンド `ctrl_start`，HW タスク終了制御コマンド `ctrl_fin`，返り値要求制御コマンド `ctrl_req`，および，要求完了制御コマンド `ctrl_ack` の 4 つの制御コマンドの識別に利用する．`ctrl_start` は，HW タスクを起動するための SW タスクから HW タスクへ転送される制御コマンドである．値が 1 となったとき，HW タスクが実行を開始する．`ctrl_fin` は，HW タスクの実行が終了したときに 1 となる制御コマンドである．`ctrl_req` は，HW タスクの実行終了後，HW タスクの返り値を要求するための制御コマンドである．値が 1 となったとき，返り値の通信を開始する．HW タスクの返り値を選択するセレクタの入力に接続される．`ctrl_ack` は，HW タスクの返り値の送信完了したときに 1 となる制御コマンドである．`ctrl_start` および `ctrl_req` は，SW タスクから HW タスクへ送信される制御コマンドである．一方，`ctrl_fin` および `ctrl_ack` は，SW タスクが HW タスクから受信する制御コマンドである．これらの 2 組の同方向の制御コマンドは，それぞれ見かけ上は同じデータであるため，**表 1** のように，イベントフラグの最上位ビットを同方向の制御コマンドの識別のために利用している．

HW タスクが複数個存在するシステムを想定して，HW タスクにはタスクの切り分け時に ID を割り当てる．ここで，ID が n である HW タスクを `HWn` と呼ぶこととする．`HWn` の制御コマンドは，対応する方向のイベントフラグの n ビット目の桁に割り当てられる．例えば，`HW0`，`HW8`，

表 1 イベントフラグによる制御コマンドの割り当て

送信元	送信先	最上位ビット	制御コマンド
SW	HW	0	`ctrl_start`
SW	HW	1	`ctrl_req`
HW	SW	0	`ctrl_fin`
HW	SW	1	`ctrl_ack`

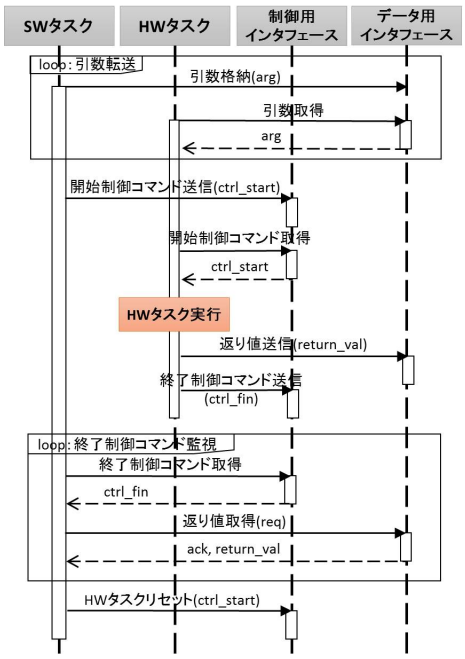

図 3 値渡しによるデータ通信の処理シーケンス

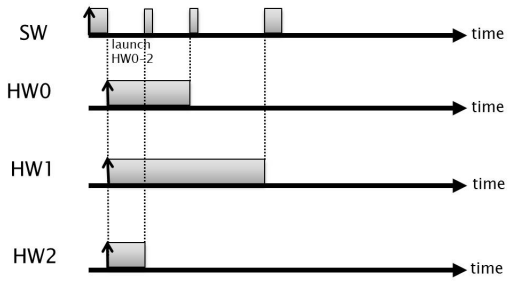

図 4 複数の HW タスクの並列実行

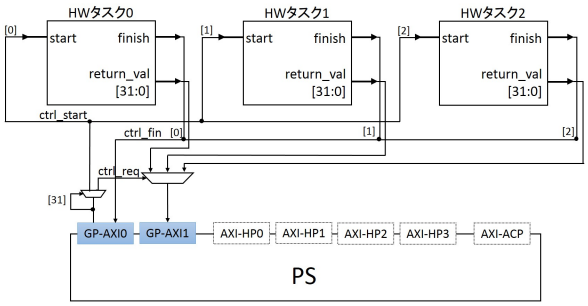

図 5 複数の HW タスクに対応したインタフェースの回路図

HW9，HW10，HW11，および，HW26 を同時に起動する場合は，HW タスクへ送信するイベントフラグの値を 0x04000f01 に設定すればよい．このようなイベントフラグのビット表現により，最大 31 個の HW タスクを同時に制御することができる．HW タスク終了制御コマンド，返り値要求制御コマンド，および，要求完了制御コマンドについても同様のビット表現によって制御が行われる．

例として，2 つの HW タスクが，GP-AXI ポートを使用し，値渡しによるデータ通信を行うように指定したシステムを考える．このシステムのソフトウェアとハードウェア間の処理シーケンスは，**図 3** のようになる．このシーケンス図のように，HW タスクの実行完了後，SW タスクは HW タスクの返り値を要求する．このとき，どの HW タスクの返り値を取得するか選択しなければならない．そこで，ctrl_req に取得すべき HW タスクの ID に対応した桁に 1 を設定し，制御用インタフェースに送信する．このイベントフラグにより，適切な HW タスクの返り値が選択され，データ用インタフェースを介して返り値を取得することができる．

以上で解説したように，イベントフラグのビット表現によって，複数の HW タスクが存在するシステムに対応できるようになる．さらに，**図 4** のように HW タスクを並列に実行することが可能となる．この図では，HW タスクへ送信するイベントフラグを 0x00000007 と設定することで 3 つの HW タスクを同時に起動することができる．イベントフラグのビット表現に対応したインタフェースの回路図を **図 5** に示す．

3.3 参照渡しによるデータ通信

プログラマブル SoC のためのシステム設計環境でサポートしている通信方式は，値渡しによるデータ通信である．しかし，HW タスクが複数個ある場合には，イベントフラグの識別といった制御のために 32 ビットのビット幅を最大限に利用できず，int 型のように，ネイティブなデータ型を完全にサポートできない．さらに，HW タスクは，単純なスカラー型のデータだけでなく，配列のような大規模かつメモリ上で連続するデータを引数とする可能性がある．しかし，値渡しによるデータ通信では，配列に対しても，逐次的にデータをハードウェアへ送信することとなり，システム全体の性能の低下を招くことが予想される．そこで，チップ上に搭載されている DDR メモリやキャッシュを格納場所として，このアドレスを通信する参照渡しによるデータ通信を検討する．

参照渡しによるデータ通信では，HW タスクの引数および返り値は DDR メモリ，PS のオンチップメモリ，あるいは，キャッシュにデータを格納する．参照渡しによるデータ通信は，検討の段階であり，まだサポートできていないが，現時点での構想を解説する．参照渡しによるデータ通信は，**図 6** のシーケンス図のように，データ用インタフェースを介して引数および返り値を格納しているメモリのアドレスを送受信することで実現される．図中のメモリを DDR メモリ，または，オンチップメモリとした場合は，データ用インタフェースのための通信ポートに AXI-HP を利用する．一方，メモリをキャッシュとした場合は，データ用インタフェースのための通信ポートに AXI-ACP を利用する．参照渡しによるデータ通信の処理シーケンスを説

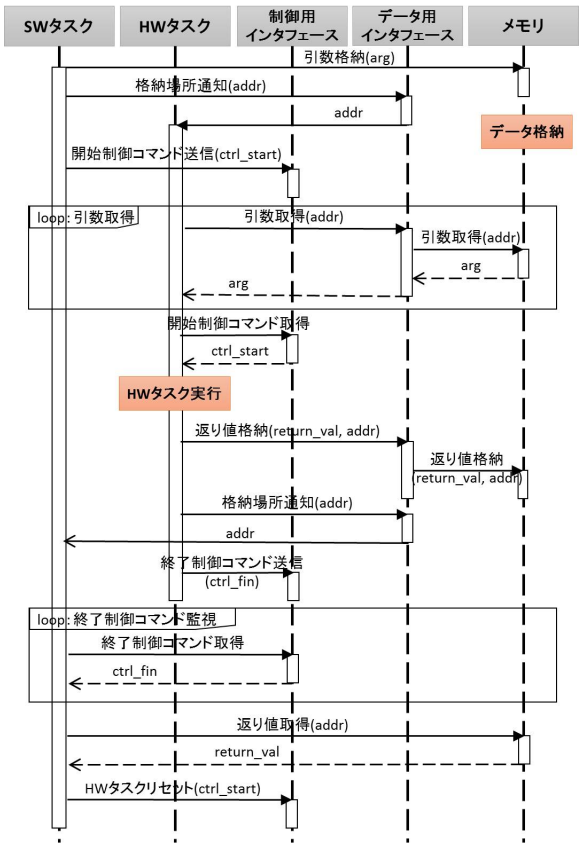

図 6 参照渡しによるデータ通信の処理シーケンス

明する．まず，HW タスクの引数および返り値のデータの個数や大きさから，格納アドレスを静的に決定する．この処理は，フロントエンドにて行われる．SW タスクは，HW タスクの引数の値をメモリに格納する．次に，格納アドレスを HW タスクへ通知する．通知の完了後，SW タスクは HW タスクに開始制御コマンドを送信する．開始制御コマンドの送信完了後，メモリに格納されたデータは，HW タスクからの要求によりデータ用インタフェースを介して PL へ送信される．HW タスクの実行が終了すると，HW タスクの返り値が，HW タスクからメモリへ送信される．その後，SW タスクは終了制御コマンドを監視し，HW タスクの終了を検知すると，SW タスクはメモリから返り値を取得する．最後に，SW タスクが開始制御コマンドをリセットすることで通信が終了する．

参照渡しによるデータ通信を指定する場合には，構成情報内<interface>にて，‘hp’，または，‘acp’ を指定する．HW0，および，HW1 の通信ポートに AXI-HP，HW2 の通信ポートに AXI-ACP を利用する場合のシステムにおけるインタフェースの回路図は **図 7** のようになる．

‘gp’ を指定した場合の値渡しによるデータ通信の回路から，HW タスクの出力に ‘addr’ が追加される．‘addr’ はデータを格納するメモリのアドレスである．引数および返り値を格納するアドレスは異なるため，メモリからデータを読み込む時の ‘addr’ の値は，引数を格納するためのアドレスとなる．一方，書き込み時は，返り値を格納するためのアドレスとなる．ここで，返り値の書き込み状態になるのは，‘finish’ が 1 のとき，すなわち，HW タスクの実行が終了した時のみである．それ以外のときは，引数の読み込み状態になるように出力を制御する．このように，‘addr’ における出力値の切り替えは終了信号の値によって制御することが可能である．また，返り値のアドレスは，静的に決定しているため，SW タスクはアドレスの指定のみで返り値を取得できる．従って，ctrl_req は不要になる．

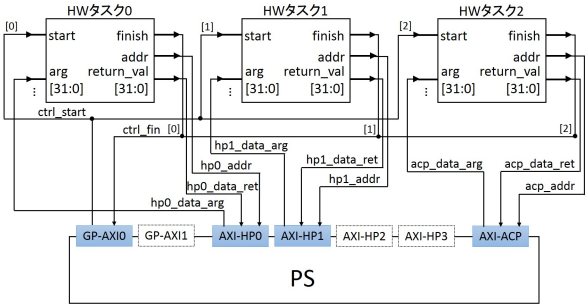

図 7 参照渡しによるデータ通信のインタフェースの回路図の例

4. 事例評価

4.1 評価環境と適用事例

実装したプログラマブル SoC のためのシステム設計環境によって生成されたシステムの性能評価を行う．評価には，ZedBoard [7] を使用する．ZedBoard は，型番が XC7Z020-CLG484-1 である Zynq-7000 を搭載している．ARM Cortex-A9 の最大クロック周波数は，667MHz である．

評価基準はシステム全体の実行時間とする．実行時間は，ARM Cortex-A9 のウォッチドッグタイマによって計測する．SW タスクが開始した時点のタイマ値と SW タスクが終了した時点のタイマ値の差分をとることで，システム全体の実行時間を計測する．

評価事例を解説する．評価には次の 3 つのタスクに含む C プログラムを使用する．

- 配列
  3 次元配列の全要素の和を計算する．
- 反復
  1 ずつ増加する値を加算し，これを繰り返す．
- 分岐
  if，else if，else，および，switch といった分岐構造を含む．

なお，以上の 3 つのタスクを HW タスクとして合成した回路のサイクル数は，それぞれ 1883，2573，および，35 と見積もられている．これらのサイクル数は，動作レベルのシミュレーションにより算出される．

3 つのタスクをそれぞれ SW タスク，あるいは，HW タスクとする全 8 種類のシステムの測定結果を比較する．イン

タフェースは，値渡しによるデータ通信を使用する．HW タスクを含む場合には，逐次実行および並列実行の 2 種類を行う．ここで，並列実行とは，HW タスクが 2 つ以上存在する場合には， 3.2 節で説明した HW タスク間の並列実行のことである．また，HW タスクの返り値の要求の順序をサイクル数が少ない順に入れ替えている．さらに，HW タスクが 1 つの場合には，HW タスクの実行中に SW タスクを実行するデバイス間の並列実行を行っている．

4.2 測定結果と考察

適用事例の全 8 種類のシステムの測定結果を **表 2** に示す．HW タスクの周波数は，80MHz の周波数を与えて実行している．実行時間の単位はすべて μs である．

まず，構成情報による HW タスクの指定について考察する．本稿の事例評価では，同じ入力の設計データに対して全 8 種類のタスクの指定を行った．従来のプログラマブル SoC における設計であれば，タスクの実行方法の選択，ハードウェア設計，ソフトウェア設計，および，インタフェース設計の 4 工程が必要であった．しかし，システム設計環境を適用した設計では，設計者が行う設計はタスクの実行方法の選択，および，ソフトウェア設計の 2 工程のみである．さらに，従来のインタフェース設計は，プログラマブル SoC 固有のオンチップバスの設定をしなければならないが，構成情報に関数名，および，そのインタフェースに使用する通信ポートの指定のみによるインタフェース設計を実現した．これによって，提案するプログラマブル SoC のためのシステム設計環境の有用性が示せた．

次に，測定結果について，SW タスクと HW タスクの比較，逐次実行と並列実行の比較の観点から考察する． 表 2 のように，すべてのタスクを SW タスクとして実行した場合の実行時間が最小となった．これは，HW タスクが動作可能な最大周波数のおよそ 8 倍程度の周波数でプロセッサが動作しているからだと考えられる．逐次実行と並列実行を比較する． 表 2 が示すように，HW タスクを 1 つとした場合の SW タスクと HW タスクの並列実行は，SW タスクの実行時間が HW タスクの実行時間と比べて極めて小さい．また，SW タスクの実行時間は，ソフトウェアとハードウェア間の通信時間のばらつきよりも小さい値になった．従って，デバイス間の並列実行による有用性が今回の事例では示すことができなかった．一方，HW タスク間の並列実行については，実行時間が最も大きい HW タスクのみが HW タスクとして実行するシステムの実行時間とほぼ同じ値となった．例えば，配列および繰り返しのタスクを HW タスクとしたときの実行時間と，繰り返しのタスクのみを HW タスクとしたときの実行時間はほぼ同じであることがわかる．

表 2 測定結果

タスク			実行時間 [μs]	
配列	反復	分岐	逐次	並列
SW	SW	SW	1.25	-
SW	SW	HW	1.94	2.35
SW	HW	SW	54.2	55.5
HW	SW	SW	34.0	34.3
SW	HW	HW	57.9	38.3
HW	SW	HW	44.4	38.5
HW	HW	SW	119	60.6
HW	HW	HW	126	69.8

5. おわりに

本稿では，我々が提案してきたプログラマブル SoC のためのシステム設計環境におけるフロントエンドの実装を拡張した．インタフェース生成では，イベントフラグのビット表現を規定し，設計されるシステムに複数のハードウェアがある場合にも対応可能とした．また本研究では，タスクが複数存在する事例に本設計環境を適用し，評価した．本設計環境を用いることにより，構成情報の書き換えのみで，複数種類のシステムを生成することができた．

今後の課題としては，インタフェースやターゲットデバイスなどの構成情報における選択肢を拡充することが挙げられる．さらに，大規模な事例に対して本システム設計環境を適用することも課題として挙げられる．

謝辞 本研究の一部は JSPS 科研費 10171422 の助成による．

参考文献

[1] Xilinx Inc.: Zynq-7000 All Programmable SoC, available from ⟨http://japan.xilinx.com/content/xilinx/ja/products/silicon-devices/soc/zynq-7000.html⟩(2014.02.13).
[2] Altera 社: アルテラ SoC の概要，入手先 ⟨http://www.altera.co.jp/devices/processor/soc-fpga/overview/proc-soc-fpga.html⟩(2014.02.13).
[3] Ha, S., et al.,: PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems, *ACM Trans. of TODAES*, Vol. 12, No. 24, pp. 1–25 (2007).
[4] Honda, S., et al.,: RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip, *Proc. of ASP-DAC '07*, pp. 336–341 (2007).
[5] 東遼平，他：プログラマブル SoC のためのシステム設計環境の検討と SW-HW インタフェース生成手法の実装，SLDM164，pp. 1–6 (2014).
[6] Canis, A., et al.,: LegUp: High-level synthesis for FPGA-based processor/accelerator systems, *Proc. of FPGA 2011*, pp. 33–36 (2011).
[7] Avnet Internix Inc.: Zedboard, available from ⟨http://www.zedboard.org/⟩(2014.02.13).