Vol.2016-ARC-219 No.24
Vol.2016-SLDM-175 No.24
Vol.2016-EMB-40 No.24
2016/3/25

プログラマブルSoCのためのシステム設計環境における SW/HW間通信方式の比較評価

谷　祐輔[†]　高瀬　英希[†]　大川　猛[††]　高木　一義[†]　高木　直史[†]

† 京都大学 大学院情報学研究科, 京都府京都市左京区吉田本町
†† 宇都宮大学 大学院工学研究科, 栃木県宇都宮市陽東 7-1-2

あらまし　近年の組込みシステムでは，プロセッサと FPGA を同一チップに集積したデバイスであるプログラマブル SoC が注目されている．本研究では，プログラマブル SoC のためのシステム設計環境について，SW/HW 間通信に着目した比較評価を示す．評価対象は，著者らが開発中の SWORDS フレームワーク，Xilinx 社の SDSoC，および，Xillybus 社の Xillybus である．それぞれの環境について，設計フローおよび設計可能なシステム構成を比較し，特徴や設計手法としての有用性を議論する．さらに，行列積演算を例題として，各環境が提供する SW/HW 間通信方式を用いて設計したシステムを評価し，通信性能のデータサイズとの相関性を比較する．

キーワード　プログラマブル SoC，FPGA，システム設計環境，バス通信

A Comparative Evaluation of SW/HW Communication Methods on System Design Environments for Programmable SoCs

Yusuke TANI[†], Hideki TAKASE[†], Takeshi OHKAWA[††], Kazuyoshi TAKAGI[†], and Naofumi TAKAGI[†]

† Graduate School of Informatics, Kyoto University, Yoshida-Honmachi, Sakyo-ku, Kyoto
†† Graduate School of Engeneering, Utsunomiya University, Yoto 7-1-2, Utsunomiya, Tochigi

Abstract　A programmable SoC, which integrates processors and FPGA on the same chip, has become attracted attention in embedded systems. This paper shows comparative evaluation of SW/HW communication methods on system design environments for Programmable SoCs. We evaluate SWORDS framework that we are developing, SDSoC of Xilinx, Inc. and Xillybus of Xillybus, Inc.. For each of the environments, we discuss the feature and usefulness as a design methodology by comparing the design flow and system configurations that can be designed. Furthermore, we quantitatively evaluate the system designed with SW/HW communication methods provided by each environment by matrix product calculation. Additionally, we compare the correlation between communication performance and data size.

Key words　Programmable SoCs，FPGA，System Design Environment，Bus Communication

1. はじめに

近年の組込みシステムでは，プロセッサと FPGA を組み合わせてヘテロジニアス構成を取るデバイスであるプログラマブル SoC が注目されている．負荷が大きい処理は FPGA 上のハードウェア（HW）で，柔軟性の求められる処理はプロセッサ上のソフトウェア（SW）で実行させることで，システムの高性能化と低消費電力化が実現できる．これによって，近年の組込みシステムにおいてますます高まっている高性能化や複雑化の要求に応えることが期待できる．

従来のデバイスに比べたプログラマブル SoC の優位点としては，搭載されるプロセッサが Cortex-A9 といった比較的高性能なものであること，さらに，プロセッサと FPGA が同一のチップに集積され高速なオンチップバスで接続されていることが挙げられる．プログラマブル SoC の例として，Xilinx 社の Zynq-7000 All Programmable SoC（Zynq）[1]，および Altera 社の SoC FPGA [2] が挙げられる．

近年，Zynq のためのソフトウェア志向のシステム設計環境として，我々が開発中の SoftWare-Oriented Design and Synthesis（SWORDS）フレームワーク [3] [4] [5]，Xilinx 社が提供し

Vol.2016-ARC-219 No.24
Vol.2016-SLDM-175 No.24
Vol.2016-EMB-40 No.24
2016/3/25

ている SDSoC [6]，および，Xillybus 社が提供する Xillybus [7] などが提案されている．これらのシステム設計環境を活用することで，FPGA 上への HW 設計および SW/HW 間通信に関する深い開発知識を有していないシステム設計者でも，プログラマブル SoC 上に高品質なシステムの設計が可能となる．また，それぞれの環境では，実行時における SW/HW 間通信やこれをサポートするオペレーティングシステム（OS）について複数の方式が提供されている．

そこで本研究では，SWORDS，SDSoC，および Xillybus の SW/HW 間通信に着目した比較評価を示す．まず，それぞれの設計環境における設計フローと設計指針，設計可能なシステム構成，および，選択可能な OS・SW/HW 間通信方式について示す．さらに，行列積演算を例題として，各環境が提供する SW/HW 間通信方式を用いて設計したシステムを比較評価する．また，各通信方式における通信性能のデータサイズとの相関性を明らかにする．以上により，各環境の長所および短所を定性的かつ定量的に議論する．

2. Zynq-7000 All Programmable SoC

Zynq はプロセッシングシステム（PS） とプログラマブルロジック（PL）で構成されている．PS はデュアルコアの Cortex-A9，キャッシュ，オンチップメモリ，外部メモリインターフェース，DMA コントローラおよび入出力ペリフェラルで構成されている．PL は FPGA を中心として，RAM，DSP，プログラマブル入出力ブロック，シリアルトランシーバおよび A-D コンバータで構成されている．

PS-PL 間の通信バスには，AMBA AXI4 が採用されている．2010 年には最新の AMBA 4.0 が公開され，AXI4 [8] が策定された．Zynq には，表 1 に示す 3 種類の通信ポートが備わっている．

PS-PL 間における通信では，一方がマスタとなりもう一方がスレーブになることが要求される．マスタとはデータ通信を開始するイニシエータとなるモジュールであり，スレーブとは通信を開始したマスタからデータを受け取って，応答するモジュールである．GP ポートは PS がマスタとなるものが 2 個，PL がマスタとなるものが 2 個用意されている．GP-AXI（GP）は汎用ポートであり，PS と PL いずれもマスタとなれる．AXI-HP（HP）は，オンチップメモリまたは DDR メモリへのアクセスが可能である．AXI-ACP（ACP）は，L2 キャッシュを介したアクセスが可能である．マスタとスレーブを接続するため，Xilinx 社が提供する IP コアである AXI Interconnect [11] を使用してポートと AXI モジュールを接続する．Zynq のポートと通信量の相関性については，文献 [9] [10] において評価が行われている．ただし，既存研究では，システム設計環境まで考慮した議論には至っていない．

Zynq における AXI4 のプロトコルの実装は，表 2 に示す 3 種類が用意されている．メモリマップ方式で最大 256 ワードバースト転送可能なプロトコルである AXI4-Master，回路規模が小さい AXI4-Lite，アドレスフェーズを持たずストリーミングに特化したプロトコルである AXI4-Stream がある．AXI4-Lite はバースト転送は不可能だが，一部の信号を省き AXI4-Master と比べて実装が簡単化されている．AXI4-Stream にはアドレスチャネルが存在せず，ストリーミングに特化したプロトコルである．

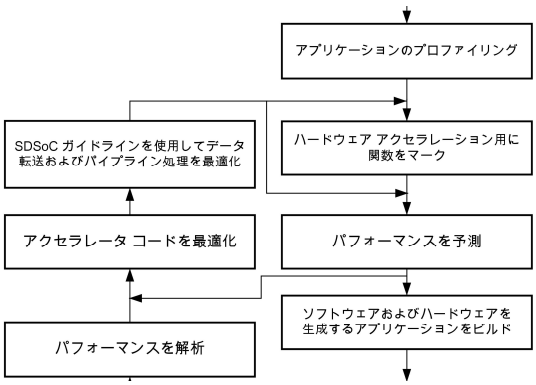

図 1　SDSoC 環境の開発フロー（ [12] より引用）

3. システム設計環境および通信方式

本章では，SDSoC，SWORDS および Xillybus における設計フローおよび設計可能なシステム構成と通信方式について説明し，それらの特徴を示す．

3.1 SDSoC 環境

SDSoC 環境とは，Xilinx 社が提供する協調設計システムを設計するための開発環境である [12]．Zynq 上の協調システムを SW 向け高級言語である C/C++のみで設計することを可能とする．本研究では，バージョン 2015.4 を評価対象とする．

3.1.1 設計フロー

SDSoC によるシステム開発のフローを図 1 に示す [12]．まず，設計者は，C/C++で記述されたソースコードのプロファイリングを行い，SW 実行時のソースコード内のそれぞれの関数の実行時間を計測する．次に，プロファイリング情報を元に，HW 化する関数を指定する．このとき，HW 化される関数の FPGA リソース量およびクロックサイクル数の予測値を得ることができる．その後，ビルドを実行し，SD カードに書き込み可能なイメージファイルが生成される．プラグマを追加することによって PS-PL 間の通信方式を明示的に指定することも可能である．ボード上での実行結果を基に，HW 化する関数の再検討やプラグマの追加による HW のパイプライン化の指定，通

表 1　Zynq の通信ポート

	ビット幅	ポート数	マスタ
GP-AXI	32	4	PS/PL
HP	32/64	4	PL
ACP	64	1	PL

表 2　Zynq の AXI プロトコル

	チャネル	バースト転送
AXI4-Lite	アドレス/データ	不可
AXI4-Master	アドレス/データ	可（256 回まで）
AXI4-Stream	データのみ	可（回数無制限

Vol.2016-ARC-219 No.24
Vol.2016-SLDM-175 No.24
Vol.2016-EMB-40 No.24
2016/3/25

信方式の変更などによってシステム全体の性能をより改善する．

SDSoC での合成フローについて説明する．SDSoC コンパイラはソースコードとプラグマを解析し，SW，コネクティビティ，HW に分割する．SW と HW の通信を担うコネクティビティは，指定されていない場合は 3.1.2 項に述べる通信方式のうち 1 つが自動で選択される．HW は，Vivado HLS によってアクセラレータ IP として合成される．最後に，Vivado によってコネクティビティおよび HW からビットストリームが論理合成される．これと同時に，コネクティビティの選択から得られる情報からドライバが生成され，SW とドライバを統合して SW の実行可能ファイルがコンパイルされる．

3.1.2 設計可能なシステム構成

SDSoC では，HW 化される関数の引数に対応して，表 3 に示す 7 種類の SW/HW 間通信方式が提供されている．スカラーの引数は，常に AXI-LITE として合成される．300B 以下の配列は，AXI-FIFO として合成できる．AXI-FIFO は通信速度は低速であるが，使用リソースを抑えられる．

AXI4-Stream を使用する通信方式の構造を図 2 に示す．AXI4-Stream では，DMA を介した SW/HW 間通信が行われる．AXI4-Stream を使用した配列引数の通信には，AXI_DMA_SG，AXI_DMA_SIMPLE および AXI_DMA_2D の 3 種類の DMA 転送を選択できる．ただし，AXI_DMA_SIMPLE および AXI_DMA_2D を使用する場合は，SDSoC のライブラリである sds_alloc() によってメモリ領域を確保し，物理的に連続するアドレスに対して通信する配列を配置する必要がある．さらに，AXI_DMA_2D を使用する場合，各次元のサイズを指定するためのプラグマ SDS_data_dim および転送される配列の領域を指定するプラグマである SDS_data_copy が必要となる．

AXI4-Master を使用する通信方式の構造を図 3 に示す．AXI4-Master を使用する場合，sds_alloc() によってメモリ領域を確保する必要がある．AXI4-Master を使用した配列引数の通信には，AXI-MASTER および ZERO_COPY の 2 種類を選択できる．ZERO_COPY を使用する場合，引数にアクセスするときは毎回 DDR メモリにアクセスされる．AXI-Master を使用する場合，memcpy() を使用して引数を内部変数にコピーする．AXI4-Stream または AXI4-Master を使用する配列引数は，キャッシュコヒーレンシを保つように指定すれば ACP ポート，キャッシュ不可能と指定すれば HP ポートに接続される．

表 3 SDSoC の関数引数の通信方式

通信方式	プロトコル	ポート	引数の型
AXI-LITE	Lite	GP	スカラー
AXI_FIFO	Lite	GP	配列（300B 以下）
AXI_DMA_SG	Stream	HP/ACP	配列
AXI_DMA_SIMPLE	Stream	HP/ACP	配列（8MB 以下）
AXI_DMA_2D	Stream	HP/ACP	配列
AXI-MASTER	Master	HP/ACP	配列
ZERO_COPY	Master	HP/ACP	配列

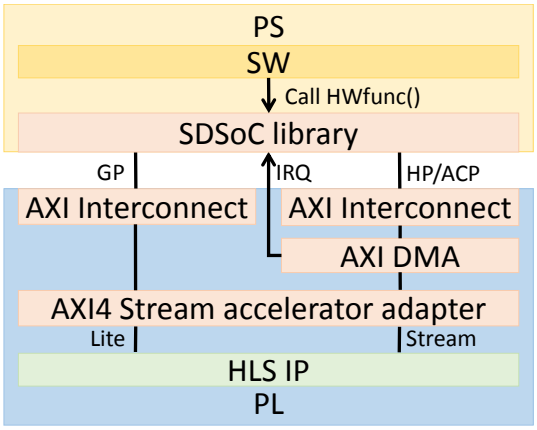

図 2 SDSoC における Stream 通信方式の構造

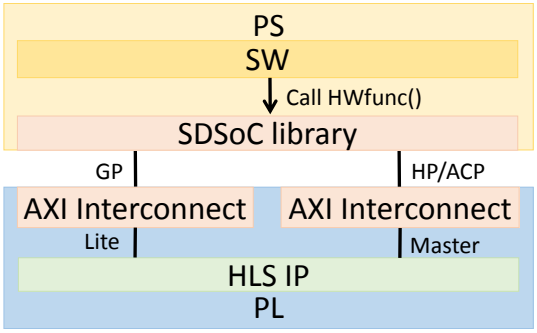

図 3 SDSoC における Master 通信方式の構造

SDSoC におけるソフトウェアは，ベアメタル構成，OS として FreeRTOS または Linux を用いる構成の実行方式が提供される．FreeRTOS に関しては，本研究の評価で用いる ZedBoard [13] などサポートされていないものがある．Linux のディストリビューションは，Xilinx 社が提供する Petalinux である．

3.1.3 特　　徴

SDSoC の利点として，まず，関数間の並列性の抽出に対応していることが挙げられる．HW が複数合成されたシステムでは，プラグマの追加によってシステムの高性能化が実現できる．次に，HW を含むシステムの性能を短時間でプロファイリングができる点が挙げられる．ただし，パフォーマンスの予測値は悲観的であり，その精度は高くない．

SDSoC の欠点としては，まず，AXI4-Stream を使用する場合の問題点が挙げられる．配列が引数に指定された場合，プラグマを指定しないときは AXI4-Stream で通信を行うように設定され，入力または出力どちらかに限定する必要がある．次に，AXI4-Master を使用する場合の問題点が挙げられる．AXI4-Master を使用する場合，対象となる引数は HW 化する関数内で memcpy 関数を明示的に記述し，HW 内のメモリにコピーするようにソースコードを書き換える必要がある．

3.2 SWORDS

SWORDS フレームワークは，著者らが開発中のプログラマブル SoC のためのシステム設計環境である [3]．ソフトウェア志向の設計環境であり，システム設計者に HW および通信バスの深い開発知識を不要とする．SW/HW 間通信には，HP また

Vol.2016-ARC-219 No.24
Vol.2016-SLDM-175 No.24
Vol.2016-EMB-40 No.24
2016/3/25

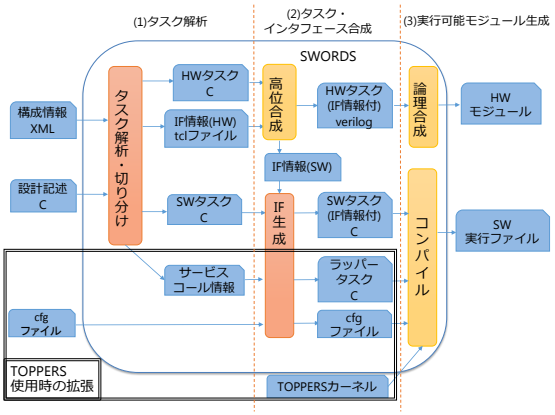

図 4　SWORDS の合成フロー

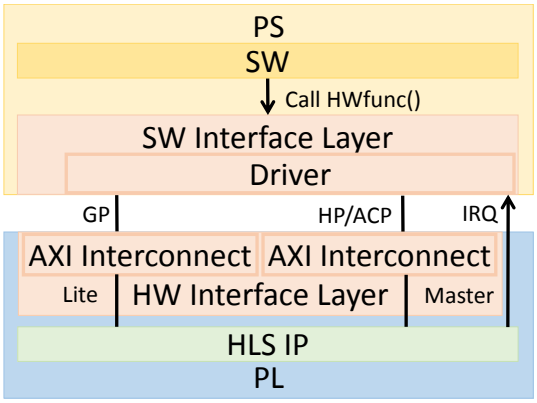

図 5　SWORDS におけるインタフェースの構造

は ACP ポートを使用したインタフェースが自動生成される [4].

3.2.1　設計フロー

SWORDS では，設計者は C ソースコードに加えて構成情報ファイルを入力として記述する．構成情報は，XML 形式で定義されているテキストデータとして記述される [4]．構成情報ファイルでは，対象のプログラマブル SoC や HW タスク化する関数および通信インタフェースなどを指定する．構成情報により，設計者は 3.2.2 項に示す HW タスクおよび HW タスクの引数の通信方式を選択できる．ここで，タスクとはシステムの処理単位のことを指し，タスクは SW 向け高級言語における関数単位と定義している．

SWORDS の合成フローを図 4 に示す．合成フローは次の 3 つのフェーズで構成されている．

- タスク解析

構成情報に基づき入力の C ソースファイルを解析し，アプリケーションを SW タスクと HW タスクに分割する．さらに，SW と HW 間のインタフェース生成のための情報を生成する．

- タスク・インタフェース合成

HW タスクとインタフェースの合成を行う．HW タスクの高位合成には，Vivado HLS を使用する．タスク解析時に生成された SW タスクには，ドライバを含むインタフェースに関する情報が追加される．HW に関するインタフェース情報は，HDL 記述に含まれる．

- 実行可能モジュール生成

プロセッサのクロスコンパイラおよび FPGA の論理合成ツールによって，ターゲット上で実行可能なモジュールを生成する．

3.2.2　設計可能なシステム構成

SWORDS によって設計された SW/HW 間通信は，図 5 に示すインタフェース上で実現される．インタフェースは SW インタフェースレイヤ，ポートおよび HW インタフェースレイヤから構成される．SW インタフェースレイヤは，ポートを介して HW タスクの実行とインタフェース上の通信を制御する機能を提供するドライバを含んでいる．HW インタフェースレイヤは，表 2 に示すプロトコルを実現する AXI モジュール，およびインターコネクトから構成されている．インターコネクトは，ポートと AXI モジュールを接続する AXI Interconnect で構成される．

HW タスクのスカラーの引数の通信のためのインタフェースは，AXI4-Lite モジュールとして合成される．配列の引数については，構成情報の指定に従い AXI4-Lite モジュールまたは AXI4-Master モジュールとして合成される．AXI4-Lite モジュールとして合成された引数は，GP ポートに接続される．AXI4-Master モジュールとして合成された引数は，構成情報の指定に従い HP または ACP ポートに接続される．

SWORDS では，インタフェース生成時に HW タスクの関数の C 言語記述を，構成情報の指定に応じて自動的に修正する．具体的には，AXI4-Master モジュールにおいてバースト転送を行う場合に，AXI4-Master モジュールを使用する引数を名前替えし，入出力のタイミングに応じた memcpy 関数を挿入して SW と HW 内のメモリの通信を実現する．また，HP ポートに接続される Master モジュールが存在する場合，データキャッシュの適切な操作のための記述を追加する．

SWORDS は，OS としてベアメタルおよび TOPPERS/FMP を使用するシステムをサポートしている [5]．TOPPERS/FMP は，ITRON4.0 仕様に準拠したオープンソースのリアルタイム OS である．TOPPERS/FMP 上のシステムを実現する場合，図 4 に示す合成フローの拡張が必要となる．コンフィギュレーション（cfg）ファイルが SWORDS の入力として扱われ，タスク解析では，サービスコール情報が抽出されてインタフェースの生成に使用される．インタフェース生成では，HW タスクを制御するラッパータスクが生成されて cfg ファイルにその情報が追加される．実行可能モジュール生成時には，TOPPERS/FMP のソースコードとリンクされる．

3.2.3　特　　徴

SWORDS の利点としては，まず，設計となる C ソースコードを変更することなく様々なシステム構成の選択が容易に実現可能であることが挙げられる．XML 形式の構成情報ファイルの記述変更のみで，システム構成の変更が可能である．さらに，インタフェース生成に関するソースコードの記述変更も自動的に行われるため，設計生産性の向上が期待される．次に，リアルタイム OS である TOPEERS/FMP を活用したリアルタイムシステムを構築できることが挙げられる．HW タスクはラッパータスクを介して操作するため，カーネルから SW タスクと

Vol.2016-ARC-219 No.24
Vol.2016-SLDM-175 No.24
Vol.2016-EMB-40 No.24
2016/3/25

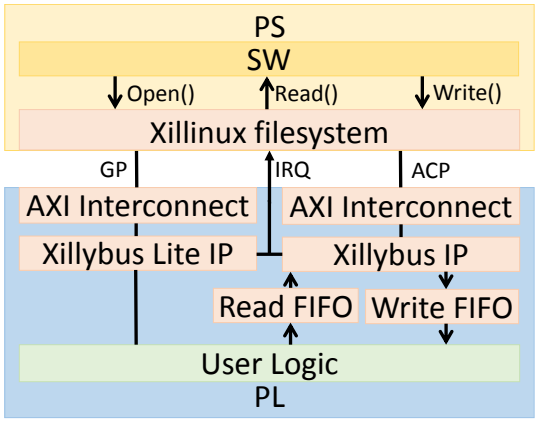

図 6　Xillybus における通信方式の構造

透過的に管理することができる．また，HW タスクから SW タスクに対してサービスコールを実行することもできる．

SWORDS の欠点としては，AXI4-Stream を用いた通信に対応していないことが挙げられる．SDSoC と比較して，通信方式の選択肢が少なくなっている．

3.3　Xillybus

Xillybus とは，Xillybus 社が提供する Zynq 上でユーザ・ロジックを容易に実現するためのフレームワークである [7]. Xillybus では，Linux から HW にアクセスできるようなインタフェースおよびドライバの雛形を設計者に提供する．

3.3.1　設計フロー

設計者は PS の仕様定義およびユーザ・ロジックを記述する．ユーザ・ロジックは HDL によって独自に設計したもの，既存の IP ブロック，および，Vivado HLS を用いて C 言語記述から合成した IP のいずれも使用できる．

Xillybus での合成フローを述べる．まず，PS 仕様定義，Xillybus IP および Xilinx 社の提供する IP からネットリストを合成する．ネットリストとユーザ・ロジックから Vivado によって論理合成・配置配線を行い，ビットストリームを合成する．最後に，ビットストリームおよび Xillinux カーネルイメージからブートイメージを合成する．Xillinux とは，Xillybus 社が提供する Ubuntu LTS 12.04 を基とする Linux ディストリビューションである．

3.3.2　設計可能なシステム構成

Xillybus では，OS として Xillinux を使用するシステムを生成する．Xillinux とユーザ・ロジックの通信は，図 6 に示す読出／書込 FIFO チャネルを介して行われる．FIFO チャネルは仮想ファイルシステム上のファイルとして扱われ，SW は低水準ファイル入出力関数である open()，read()，および write() を使用してアクセスできる．GP ポートと ACP ポートは AXI Interconnect を通じて Xillybus Lite IP と Xillybus IP にそれぞれ接続されている．Xillybus Lite IP とは，AXI Interconnect とユーザ・ロジックの制御入出力を接続する IP であり，Xillybus IP とは，AXI Interconnect と FIFO を接続する IP である．

```
for (i = 0; i < MATRIX_SIZE; i++){
    for (j = 0; j < MATRIX_SIZE; j++){
    #pragma HLS PIPELINE II=1
        c[i][j] = 0;
        for (k = 0; k < MATRIX_SIZE; k++){
            c[i][j] += a[i][k] * b[k][j];
        }
    }
}
```

図 7　行列積プログラム

3.3.3　特　　徴

Xillybus の利点として，HW へのアクセスを抽象化されたファイルへのアクセスとして実装していることが挙げられる．このため，データの読み出しと書き込みのスレッドを分けた場合でも，アクセス制御を低レベルで行う必要が無い．

Xillybus の欠点としては，Xillybus IP およびデバイスドライバがブラックボックスで提供されるため，用途に応じた性能チューニングが難しいことが挙げられる．

4.　定量的評価

SDSoC，SWORDS および Xillybus によって生成されるシステムを比較するため，Zynq XC7Z020-1CLG484 を搭載した評価ボード ZedBoard [13] を用いて，SDSoC，SWORDS および Xillybus によって設計されたシステムを比較評価した．設計対象は，図 7 に示す行列積演算の C プログラムを用いた．入力は unsigned long long 型の一辺 4，8，16 または 32 の 4 種類の二次元配列の組とした．高位合成において，三重ループのうち中間のループにパイプライン化のプラグマを指定した．

SDSoC における評価対象の設計は，Linux 上の設計とする．SW で実行するもの，行列積の関数を HW 化して通信方式に Stream（AXI_DMA_SIMPLE）を指定したもの，および，Master（AXI-Master）を指定したものの 3 種類について計測した．SWORDS における評価対象の設計は，ベアメタル上の設計とする．SW のみで実行するもの，および，行列積の関数を HW 化して通信方式に HP ポートと接続する AXI4-Master を使用するもの，ACP ポートと接続する AXI4-Master を使用するものの 3 種類について時間を計測した．HP ポートと接続する場合，キャッシュを無効化する．HW タスクの完了通知の取得は，ポーリングおよび割込みによるものの両方の時間を計測した．AXI4-Master モジュールを使用する設計は，ビット幅を 64bit としている．Xillybus における評価対象の設計は，SW で実行するものおよび HW で実行するものとする．

全ての環境において Vivado 2015.4 を使用する．HW 合成時の目標周波数は 100MHz とした．計測時間を表 4 に示す．計測時間の単位は us である．HW の使用リソースを表 5 に示す．

まず，各環境が提供する SW/HW 間通信方式を比較する．データサイズが小さい場合，SWORDS による設計が最も性能が高くなったことが分かる．サイズが大きい場合，SDSoC の Master が有効となった．Xillybus では HW 化の効果が見られず，実行時間も非常に大きかった．

Vol.2016-ARC-219 No.24
Vol.2016-SLDM-175 No.24
Vol.2016-EMB-40 No.24
2016/3/25

次に，SDSoC および SWORDS でのデータサイズとの相関性を述べる．HW 実行では，サイズが大きいほど実行時間が増加するが，SW 実行時と比較して増加割合が少ないことが分かる．また，使用リソース量はサイズが大きいほど増加するが，SDSoC で Stream を使用する場合は Master と比較して少ないことが分かる．Stream を使用する場合は通信と計算の処理が並列に行われるため，計算処理がデータ通信を待つ時間が長くなる．Master 通信では，必要なデータの通信を一括に処理してから，計算処理が行われる．このため，Stream を使用する場合は Master を使用する場合と比較して並列に実行する計算量が少なくなり，使用リソース量が減少し実行時間の増加割合が大きくなる．

SWORDS では，ポーリングでタスク終了検知を行う場合，HP と ACP ではどちらも同程度の実行時間となった．ただし，HP による通信はキャッシュに格納されないため，SW タスクが HW タスクの実行結果を使用する場合はシステム全体の性能の劣化が考えられる．本実験では，SWORDS で生成される設計では他のタスクを実行していないため，ポーリングが高速となっている．割込みを使用する場合は，ACP を使用するものが高速となる．これは，キャッシュを無効化しているため，割込みハンドラの処理に必要な時間が増加するからである．

紙面の都合上結果の掲載は割愛するが，SDSoC の AXI4-Stream における 3 種類の DMA 転送方式も比較した．その結果，AXI_DMA_SIMPLE が 3 種類のうち最も高速となった．これは，今回使用したプログラムのデータは物理的に連続するアドレス空間に配置されており規則正しくアクセスされるため，AXI_DMA_SG および AXI_DMA_2D におけるデータ収集のオーバヘッドが余分に掛かったためである．

表 4　行列積演算の実行時間 (us)

環境	通信方式	サイズ			
		4	8	16	32
SDSoC	Stream	28	30	74	392
	Master	21	23	40	157
	Soft	8	49	382	2972
SWORDS	HP-pol	4	8	31	201
	HP-intr	46	50	70	237
	ACP-pol	4	8	32	198
	ACP-intr	11	14	38	205
	Soft	8	50	379	2953
Xillybus	Hard	478	607	2580	8570
	Soft	17	36	369	2967

表 5　行列積演算の使用リソース

	環境	通信方式	FF	LUT	LUT RAM	BRAM	DSP
4	SDSoC	Stream	10729	6442	640	14.5	12
		Master	5624	2998	1110	7.5	40
	SWORDS	HP	4139	2509	1029	5	40
		ACP	4139	2509	1029	5	40
	Xillybus	FIFO	6903	5853	1146	10	40
8	SDSoC	Stream	11555	6668	689	15	24
		Master	7670	4069	1910	7.5	80
	SWORDS	HP	6185	3568	1829	5	80
		ACP	6185	3567	1829	5	80
	Xillybus	FIFO	8954	6898	1946	10	80
16	SDSoC	Stream	13045	7139	851	15	48
		Master	11746	6003	3509	7.5	160
	SWORDS	HP	10204	5544	3424	5	160
		ACP	10204	5543	3424	5	160
	Xillybus	FIFO	12979	8860	3541	10	160
32	SDSoC	Stream	16842	8608	1302	15	96
		Master	25122	35919	6725	8.5	220
	SWORDS	HP	23656	33823	6630	6	220
		ACP	23656	33834	6630	6	220
	Xillybus	FIFO	26455	37139	6745	11	220

5. おわりに

本稿では，SWORDS フレームワーク，SDSoC，および Xillybus の SW/HW 間通信に着目し，システム設計環境としての定性的かつ定量的な比較評価を行った．今後の方針としては，通信時間と計算時間を切り分けた詳細な解析や，他のアプリケーションでの評価が挙げられる．また，SWORDS フレームワークを改良し，AXI4-Stream を使用するインタフェースに対応することも検討している．

謝辞　本研究の一部は，JSPS 科研費 26870303 および総務省 SCOPE(No.0159-0112) の助成による．

文　献

[1] Xilinx 社，“Zynq-7000 All Programmable SoC”. http://japan.xilinx.com/content/xilinx/ja/products/silicon-devices/soc/zynq-7000.html
[2] Altera 社，“アルテラ SoC FPGA の概要”．http://www.altera.co.jp/devices/processor/soc-fpga/overview/proc-soc-fpga.html
[3] Hideki Takase, et al., “A Study of a Software-Centric System Design Environment for Programmable SoCs,” In Proc. of ITC-CSCC 2014, pp.737–740, 2014.
[4] 谷祐輔，他，“プログラマブル SoC のためのシステム設計環境における SW/HW インタフェース生成手法,” 電子情報通信学会技術研究報告, Vol. 115, No. 109，pp.73–78，2015.
[5] 畑山拓也，他，“プログラマブル SoC におけるリアルタイムシステム構築のためのソフトウェア指向の協調設計環境,” 電子情報通信学会技術研究報告, Vol. 115, No. 343，pp.27–32，2015.
[6] Xilinx 社，“SDSoC 開発環境”．http://japan.xilinx.com/products/design-tools/software-zone/sdsoc.html
[7] Xillybus 社，“Xillybus”. http://xillybus.com/
[8] Shaila S Math, et al., “Data transactions on system-on-chip bus using AXI4 protocol,” Recent Advancements in Electrical, Electronics and Control Engineering, pp.423–427, 2011.
[9] Silva, João, et al., “Comparison of On-chip Communications in Zynq-7000 All Programmable Systems-on-Chip,” Embedded Systems Letters, IEEE 7.1, pp.31–34, 2015.
[10] Sadri, M, et al., “Energy and Performance Exploration of Accelerator Coherency Port Using Xilinx ZYNQ,” Proceedings of the 10th FPGAworld Conference, p.5, 2013.
[11] Xilinx 社，“AXI Interconnect”. http://japan.xilinx.com/products/intellectual-property/axi_interconnect.html
[12] Xilinx 社，“SDSoC 環境ユーザーガイド”. http://japan.xilinx.com/support/documentation/sw_manuals_j/xilinx2015_2/ug1027-intro-to-sdsoc.pdf
[13] Avnet Internix Inc, “Zedboard”. http://www.zedboard.org/