Vol.2014-ARC-208 No.8
2014/1/23

メモリ抽象化フレームワークPyCoRAMを用いたソフトプロセッサ混載FPGAアクセラレータの開発

高前田（山崎） 伸也[1,2,a)]　吉瀬 謙二[1,b)]

概要：本稿では，計算機アーキテクチャ研究会第 1 回プロセッサ設計コンテストに提出したデザインについて報告する．我々は，FPGA システムにおけるメモリ抽象化フレームワーク PyCoRAM を用いて，MIPS ソフトコアプロセッサ混載の FPGA アクセラレータを開発した．PyCoRAM は，計算カーネルのパイプライン構成などが定義されるユーザロジックの RTL デザインと，アプリケーションのメモリアクセスパターンを定義したコントロールスレッドと呼ばれるソフトウェアのモデルにより，FPGA 上で動作するアプリケーションを設計するフレームワークである．競技に用いられる 4 種のアプリケーションのうち，アクセラレータによる高速化が容易な行列積（320mm）の計算カーネル部を専用ハードウェアにより実現した．結果の出力や制御部，および他の 3 種のアプリケーション全体については，6 段パイプラインの MIPS プロセッサにより処理を行う．プロセッサには 2-way セットアソシアティブの L1 データキャッシュを追加し，メモリアクセスレイテンシの削減を図った．

1. はじめに

性能と電力効率の向上を目的に，汎用 CPU に加えて GPU や FPGA などを用いたアクセラレータ機構を持つ計算機システムが普及しつつある．特に，FPGA を用いたアクセラレータは，アプリケーションに特化したパイプラインとデータ供給機構をハードウェアとして FPGA 上に実現することにより，CPU や GPU と比較して，高い性能や電力効率を達成できることが報告されている．

本稿では，計算機アーキテクチャ研究会第 1 回プロセッサ設計コンテストに提出した，FPGA 上に実現されるプロセッシングシステムの設計について述べる．我々は，FPGA が持つオンチップおよびオフチップのメモリシステムの抽象化により，ポータブルなアクセラレータの開発を可能にするフレームワークの PyCoRAM[1] を用いて，ソフトコアプロセッサとアクセラレータロジックを混載する FPGA アクセラレータを開発した．

2. PyCoRAM

PyCoRAM は，CoRAM メモリアーキテクチャ [2] に基づいた，FPGA を用いたコンピューティングのための IP コア合成フレームワークである．CoRAM は，FPGA が持つメモリシステムを抽象化することにより，ポータブルなアプリケーションの実装を可能にするモデリング方式である．CoRAM のプログラミングモデルを**図 1** に示す．CoRAM では，FPGA オンチップとオフチップのデータ転送を，コントロールスレッドと呼ばれるソフトウェアによるモデルで記述する．CoRAM においては，アプリケーションの実装は，計算カーネルとコントロールスレッドの 2 つの部分に分離される．計算カーネルは，CoRAM の抽象的な機能ユニットに密に結合されたハードウェアとして，RTL によりモデリングされる．CoRAM のメモリユニットはオンチップネットワークを介して，オフチップメモリに結合される．コントロールスレッドは，オフチップ

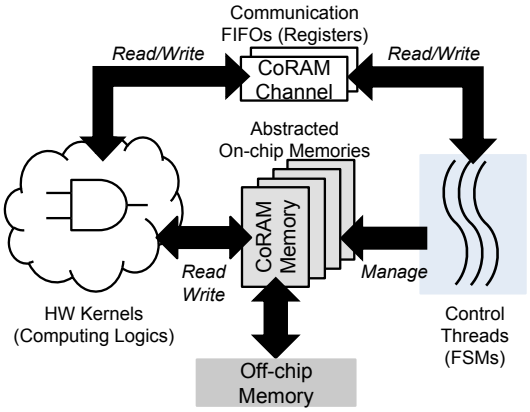

図 1　CoRAM プログラミングモデル

1　東京工業大学 大学院情報理工学研究科
2　日本学術振興会 特別研究員
a)　takamaeda@arch.cs.titech.ac.jp
b)　kise@cs.titech.ac.jp

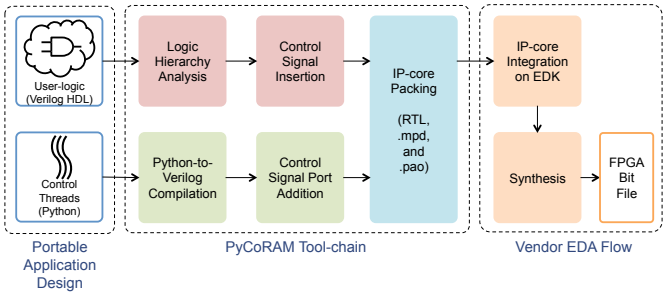

図 2 PyCoRAM におけるシステム開発フロー

メモリと CoRAM のメモリユニットとの間のデータ転送を管理する．コントロールスレッドと計算カーネルのロジックは CoRAM チャネルを介して，互いに通信が可能である．用途としては，コントロールスレッドがオンチップの CoRAM メモリに対してデータ転送が完了した後に，計算カーネル側にデータ転送完了を通知するといった使い方が挙げられる．

PyCoRAM は CoRAM のメモリシステムの抽象化を IP コアベースの FPGA システムの開発環境に統合したフレームワークである．PyCoRAM は，計算カーネルのロジックの RTL 記述とアプリケーションのメモリアクセスパターンを記述した Python のソースコードから，オフチップメモリとのデータ転送を行う DMA コントローラなどを含む IP コアを生成する．**図 2** に PyCoRAM を用いた FPGA アクセラレータの開発フローを示す．PyCoRAM は，(1)CoRAM の抽象的なメモリブロックを用いた計算カーネルの RTL デザインの変換ツールと，(2) メモリアクセスパターンを記述した Python のソースコードから FPGA 上でデータ転送を管理するハードウェア構成を出力する高位合成コンパイラ，の 2 つから構成される．入力された RTL デザインは，構文解析とデータフロー解析を経て，コントロールスレッドによるデータ転送の制御が可能な形に変換される．コントロールスレッドは Python から Verilog HDL のデザインを出力する高位合成コンパイラにより，状態遷移機械 (FSM) の形で RTL デザインに変換される．PyCoRAM は，IP コア合成の最終段では，出力される RTL デザインを EDA ツールにて IP コアとしてのいくつかの設定ファイルとトップレベルのデザインを出力する．PyCoRAM にて出力された IP コアを，EDA ツールに取り込み，他の IP コアなどと一緒にインターコネクトに接続し，通常の合成フローを辿て FPGA の回路イメージ（ビットファイル）を作成する．

図 3 に，PyCoRAM により合成された IP コアを含むシステム全体の構成例を示す．PyCoRAM が合成する IP コアは標準的なオンチップインターコネクトの AMBA AXI4 マスターインターフェスを持つ．オフチップメモリのインターフェース，および IP コア間を接続するインターコネクトは，ベンダーが提供する EDK (Embedded Development Kit) により自動的に合成される．そのため，システム開発者は IP コアの設計時にはオンチップインターコネクトやメモリインターフェースについては抽象化された機能ユニットとして扱えばよく，設計の切り分けが容易になる．

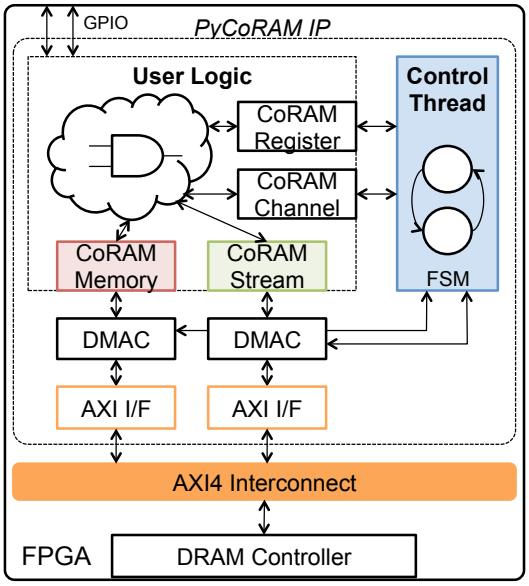

図 3 PyCoRAM を用いたシステム

図 4 にコントロールスレッドの記述例を示す．PyCoRAM では，コントロールスレッドはスクリプト言語の Python でモデリングされる．通常のプログラミングと同様のシンタックス（if 文，for 文，while 文）を用いてメモリアクセスパターンを記述できるため，設計の効率が高い．またアプリケーションの性能を大きく左右する計算カーネルの構成については，HDL やパイプライン指向の高位合成系により行うことで，性能とモデリング容易性の両立を可能にする．例では，配列の合計値を求める計算カーネルに対して，オフチップからオンチップの CoRAM メモリオブジェクトにデータ転送を行う．例の 1 行目および 2 行目では，制御対象の CoRAM オブジェクトが生成されている．これらの CoRAM オブジェクトに対するデータ転送は，コントロールスレッド記述中の対象のオブジェクトが持つ read メソッドおよび write メソッドを呼び出すことにより表現される．CoRAM メモリに対する read および write の操作は，オンチップの CoRAM メモリとオフチップの DRAM との間の DMA 転送として実現される．CoRAM チャネルに対する read および write の操作は，計算カーネル側とコントロールスレッドとの間に結ばれる FIFO に対する読み書きとして実現される．CoRAM メモリに対する read および write の操作は，完了を待ち合わせるブロッキング転送と，完了を待ち合わせないノンブロッキング転送のどちらも利用可能である．そのため，ダブルバッファリングなどの演算とデータ転送のオーバーラップさせるような高度な処理も記述可能である．

```
0  # PyCoRAM memory object and channel object
1  ram = CoramMemory(idx=0, datawidth=32, size=1024)
2  channel = CoramChannel(idx=0, datawidth=32)
3
4  ram_size = 128
5
6  # function definition
7  def array_sum(repeat_time):
8      addr = 0
9      sum = 0
10     for i in range(repeat_time):
11         # transfer from DRAM to CoRAM memory
12         ram.write(0, addr, ram_size)
13         # write a value to the channel
14         channel.write(addr)
15         # wait for a token from the user-logic
16         sum = channel.read()
17         addr += ram_size * 4
18     # display statement for Verilog simulation
19     print('sum=', sum)
20
21 # function call
22 array_sum(8)
```

図 4 コントロールスレッド（Array-sum）

```
CoramMemory1P
#(
 .CORAM_THREAD_NAME("thread_name"),
 .CORAM_ID(0),
 .CORAM_ADDR_LEN(ADDR_LEN),
 .CORAM_DATA_WIDTH(DATA_WIDTH)
 )
inst_memory
(.CLK(CLK),
 .ADDR(mem_addr),
 .D(mem_d),
 .WE(mem_we),
 .Q(mem_q)
 );
```

(a) CoRAM Memory

```
CoramChannel
#(
 .CORAM_THREAD_NAME("thread_name"),
 .CORAM_ID(0),
 .CORAM_ADDR_LEN(CHANNEL_ADDR_LEN),
 .CORAM_DATA_WIDTH(CHANNEL_DATA_WIDTH)
 )
inst_channel
(.CLK(CLK),
 .RST(RST),
 .D(comm_d),
 .ENQ(comm_enq),
 .FULL(comm_full),
 .Q(comm_q),
 .DEQ(comm_deq),
 .EMPTY(comm_empty)
 );
```

(b) CoRAM Channel

図 5 ユーザロジックにおける PyCoRAM オブジェクト

図 5 に計算カーネルの RTL 中の CoRAM オブジェクトの定義の例を示す．他のモジュールと同様に，計算カーネル中で CoRAM のオブジェクトのインスタンスを作成することで，抽象的なメモリシステムが利用可能である．CoRAM メモリインターフェースはブロック RAM に対するものと同じである．CoRAM チャネルのインターフェースは一般的な FIFO のものと同じである．そのため，CoRAM の方式を導入することにより計算カーネルの設計は複雑化しない．

3. プロセッサ設計コンテスト向け FPGA アクセラレータ

図 6 に我々がプロセッサ設計コンテスト向けに開発した FPGA アクセラレータの構成を，**表 1** にそのシステムの各パラメータをそれぞれ示す．アクセラレータ IP コアの合成には PyCoRAM 0.7 を用いた．AXI4 バスを含むシステム全体の合成には Xilinx Platform Studio 14.6 を用いた．コンテストで用いられる 4 種類のアプリケーション（ソート (310sort), 行列積 (320mm), 9 点ステンシル計算 (330stencil), 最短経路問題 (340spath)）のうち，ハードウェアによる実現が容易でかつ性能向上が容易な，行列積のための専用回路を実装した．表示や制御，およびその他 3 種のアプリケーション全体の実行のためには，6 段パイプラインの MIPS アーキテクチャのプロセッサコアを実装した．また，ソフトウェア部の性能向上のために，2 ウェイセットアソシアティブの L1 データキャッシュメモリを実装した．

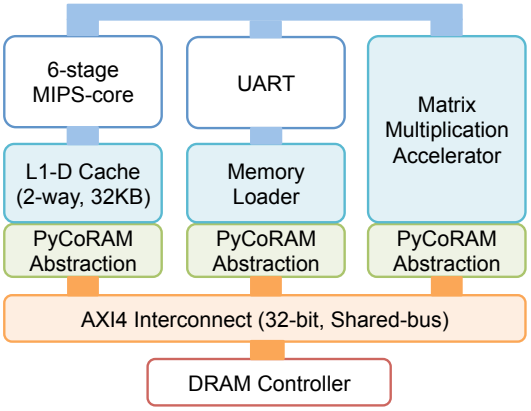

図 6 アクセラレータ構成

キャッシュメモリ，UART プログラムローダー，行列積アクセラレータの 3 つの AXI4 インターフェースが AXI4 インターコネクトを介してオフチップの DRAM に接続される．MIPS プロセッサコアはキャッシュメモリに対して，一般的なリードライトリクエストのインタフェースを介して接続される．プロセッサコアは UART 経由でプログラムがロードされるのを待機する．また，アクセラレータの制御ために，MIPS プロセッサコアとアクセラレータは接続される．

MIPS ソフトコアおよびアクセラレータ部はすべて 100MHz で動作する．メインメモリのインタフェースは最大動作周波数 400MHz の 75%の 300MHz で動作し，メモリバンド幅は 1.2GB/s である．MIPS ソフトコア・アクセラレータ部とメインメモリを接続するインターコネクトには AXI4 を用いた．動作周波数は 100MHz とした．ハードウェア使用量を考慮して，AXI4 インターコネクトのデータ幅を 32 ビット，接続方式を共有バス（エリア優先）とした．そのため，メモリバンド幅に対する AXI4 インターコネクトのバンド幅は 3 分の 1 である．

キャッシュメモリは 32KB，2 ウェイセットアソシアティブ，ラインサイズ 32B の構成である．データメモリに CoRAM メモリを用いて，オフチップメモリとのデータの入れ替えをコントロールスレッドとして実装した．行列積アクセラレータの計算カーネル部は，3 つの CoRAM メモリとそれに密に結合された乗算パイプラインで構成される．乗算パイプラインは 3 つの CoRAM メモリに対して通常のブロック RAM と同様にインターフェス・プロトコルでアクセスする．コントロールスレッドとして，オフチップからオンチップの CoRAM メモリへのデータ転送制御を実装した．行列積の計算ソースとなるデータの転送は，ノンブロッキング転送を合わせて用いてダブルバッファリング化を施し，データ転送待ち時間の削減を図った．

表 1　アクセラレータ構成

FPGA	Xilinx Spartan-6 LX45, 261 KB BRAM
DRAM	DDR2-800, 1.2 GB/s (300 MHz op.)
Interconnection	AMBA AXI4, shared bus (area-optimized), 32-bit width, 100 MHz op.
Soft CPU	6-stage pipeline, single-issue, MIPS-ISA, no-branch-prediction, 100 MHz op.
Cache	32KB, 2-way, 32-byte per line, 1-outstanding miss, no-prefetching
Matrix Multiplier Acc.	1-mult per cycle, 6-stage pipeline, double-buffered source memory, 100 MHz op.

4. まとめ

本稿では，第 1 回プロセッサ設計コンテストに提出したソフトプロセッサ混載 FPGA アクセラレータの構成と，開発に用いた FPGA システムのメモリ抽象化フレームワークの PyCoRAM の概要について述べた．これより，決勝に向けてソフトプロセッサおよびソフトウェアの最適化を行う予定である．また，ハードウェア使用量に応じて，行列積と同様にハードウェア化が容易であり，性能向上の見込が高いステンシル計算のハードウェアアクセラレータを追加を検討する．

参考文献

[1] Takamaeda-Yamazaki, S., Kise, K. and Hoe, J. C.: PyCoRAM: Yet Another Implementation of CoRAM Memory Architecture for Modern FPGA-based Computing, *Intersections of Computer Architecture and Reconfigurable Logic (CARL 2013)* (2013).

[2] Chung, E. S., Hoe, J. C. and Mai, K.: CoRAM: an in-fabric memory architecture for FPGA-based computing, *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, New York, NY, USA, ACM, pp. 97–106 (online), DOI: 10.1145/1950413.1950435 (2011).