FPGA 用 ARM プロセッサコアの設計

システム情報工学研究科　コンピュータサイエンス専攻　1年

201220718　金　紅坤

指導教員　児玉祐悦　　山口佳樹

2012 年 12 月 20 日

1　はじめに

1970 年代、マイクロプロセッサが登場した。当時、高価であったため、交通システムといった機械制御に限られていた。1980 年代、 マイクロプロセッサの低コスト化と性能向上により、家電製品などの民生機器分野まで、マイクロプロセッサを用いた制御方式が使われ始めた。1970 年代と比べ、大きな変化は専用バスの代わりに汎用バスが導入されたという点である。1990 年代、半導体技術の進歩により、従来集積できなかった複数の回路を1チップに実現することが可能になった。これはいわゆる SoC(System on Chip)設計である。システムを1チップに集積化することで、マイクロプロセッサシステムの小型化、高速化、低消費電力、コスト削減などが加速した。インターネットなどの情報インフラの普及に加え、機械や機器のネットワーク化・デジタル化が急速に進展した。近年、半導体の微細化が更に進み、FPGA といった再構成可能なチップ上にシステムを集積することも可能になっている。FPGA を用いたシステム設計は、開発期間短縮、仕様変更と設計失敗などに伴う設計リスクが小さい、アプリケーションに応じた柔軟な構造、などのメリットを持っており急速に普及してきた。

ARM プロセッサは携帯情報端末、ゲーム機、音楽プレーヤーなどの分野で、「ARM プロセッサは事実上標準」だと言われるほど 広く使われている。ARM は RISC(Reduced Instruction Set Computing)プロセッサで、ロード・ストアアーキテクチャでありながらもコード密度を重視した CISC 寄りの設計となっている。他のプロセッサと比べ、コアの面積は驚くべき小さい。回路面積が小さいという事は、チップ上のトランジスタの数が少なく、消費電力が抑えられることとコストが低いことを意味している。

本研究では、ARM アーキテクチャを基本とした自由度の高い、マルチコア時にプロセッサ間のデータ共有し易いソフトプロセッサコアの設計を目的とする。この提案の利点はまず、アプリケーションに応じて、構成の変更が簡単だということになる。また、ARM アーキテクチャを拡張することにより、マルチコア時に、コア数に対してスケーラブルな性能向上を保証しやすい。

本稿では、まず第2章で、FPGA プロセッサコアについて説明し、その特徴と本研究で提案する方向性について示す。第 3 章では ARM アーキテクチャについて、第4章では提案プロセッサのアーキテクチャをそれぞれ説明する。第5章で、現時点での実装結果を示す。最後に第6章で、現時点でのまとめと今後の課題について述べる。

2　FPGA 用プロセッサコア

2.1　プロセッサコアとは

半導体集積技術の向上により、FPGA の論理ゲート数は指数関数的に増加している [Paul2010]。これは、FPGA 上での大規模回路設計を可能にし組み込みシステムへの応用を広げたが、一方でスクラッチ開発を非常に困難なものにしている。そこで、SystemC や OpenCL などの高位記述言語を利用するのと併行し、IP コアと呼ばれるモジュールを基本とした設計手法が導入されている。近年、プロセッサそのものが IP コアとして提供されており、それを利用した SOPC (System On a Programmable Chip) 設計も広くみられるようになっている [James2006]。本章では、IP コアとして利用されているプロセッサを分類とその問題点について議論する。

2.2　ハードウェアプロセッサコア

FPGA は、回路情報用のメモリとそれに接続されたスイッチ群により、電気的にハードウェア構成を変更することができる。この柔軟性はアプリケーションに最適化したハードウェア構成を提案できるという点で非常に高いポテンシャルを持っている。一方、本来不要であるメモリやスイッチ群を備えていることから、乗算器のような良く使用されかつ最適な回路構造が決まっているハードウェアを改めて実装するには適していない。そこで、ハードウェア可変構造を有さないひと塊の電子回路(モジュール)として FPGA 内部に埋め込み、その利用をスイッチで切りかえるタイプの構造が提案されている。このタイプで実現されているプロセッサを一般にハードウェアプロセッサコアと呼ぶ。ハードウェアプロセッサコアは、表1に示すように数多くのものが提案されている。

表 1:FPGA のハードウェアプロセッサコア

FPGA シリーズ	プロセッサアーキテクチャ
Virtex 2 Pro [XILINX]	PowerPC 405 (Single or Dual)
Virtex 4 [XILINX]	PowerPC 405 (Single or Dual)
Virtex 5 [XILINX]	PowerPC 440 (Single or Dual)
Zynq [XILINX]	ARM Cortex-A9 (Dual)
Arria II GX [ALTERA]	Intel Atom E6x5C (Single)(*1)
Arria V [ALTERA]	ARM Cortex-A9 (Dual)
Cyclone V [ALTERA]	ARM Cortex-A9 (Single or Dual)
Quick MIPS [QuickLogic]	MIPS 4Kc

(*1) FPGA 部分が Intel Atom Processor に組み込まれている構成のため、他とは若干異なる。

ハードプロセッサコアを利用したシステムは数多く提案されており [Ahmed2010, Christopher2005]、複数のハードウェアプロセッサコアを協調動作させたマルチコア型の組み込み FPGA システムも存在する。

2.3　ソフトウェアプロセッサコア

ハードウェアプロセッサコアに対し、ハードウェア記述言語(HDL)や論理回路またはネットリストという形で提供され、ユーザが自分の回路設計に自由に追加・削減・変更できるタイプのプロセッサをソフトウェアプロセッサコアと呼ぶ。その数は 160 種類を超え、命令セットアーキテクチャ、浮動小数点演算の有無、キャッシュコントローラの有無、またライセンス形態(営利/非営利)など様々である。一般にソフトウェアプロセッサコアはフリーの IP コアとして利用できるものが多く [Opencores]、研究や製品開発だけでなく教育用途などにも幅広く利用されている。

2.4　プロセッサコアの現状と本研究による提案

2.2 節および 2.3 節で示したように、多くのプロセッサコアが提案されている。これらを機能面でまとめてみると表 3 のように示すことができる。

表 2:プロセッサコアの特徴

	ハードウェアプロセッサコア	ソフトウェアプロセッサコア
汎用性	高(開発環境が充実)	低～中(独自仕様もあり様々)
可変性	低(仕様の変更不可)	中～高(仕様の変更が容易)
拡張性	低(個数の変更不可)	低～中(マルチコア化などは容易)

今後、SoPC 設計はさらに加速すると考えられる。このとき、プロセッサコア単体の性能向上に加え、汎用性や開発環境の充実、またコア数に応じたスケーラビリティを保証するようなプロセッサコアが重要になることが容易に推測できる。

そこで本研究では、組み込みシステムで広く利用されている ARM アーキテクチャをターゲットとし、マルチコアおよびメニーコアシステムを構築する際に性能低下を極力避ける構造(隣接プロセッサ間のレジスタアクセスやキャッシュ制御など)を内包する新しいプロセッサコアについて提案を行う。

FPGA 向けの ARM ソフトウェアプロセッサコアとしては既に、Actel 社提供の ARM Cortex M1 [ACTEL]、オープンソース化された AMBER [Conor2011]、STORM [Stephan2012] などがある。しかし、いずれもマルチコア化やメニーコア化を視野に入れた効率的な回路構造になっているとは言い難い。また、設計思想がそもそも異なるため、これらをベースにした再開発も難しい。そこで本論文では、スクラッチ設計から始め、よりマルチコアおよびメニーコアに適したプロセッサ構造について議論する。

3 ARM アーキテクチャ

3.1 ARM の概要

ARM は、低消費電力型プロセッサとして組み込み機器を対象に提案され、32bit の RISC プロセッサ市場では全世界で最も利用されているアーキテクチャといえる。ARM ファミリーとアーキテクチャのバージョンの対応を表 3 に示す。

表 3: ARM プロセッサの概要

Architecture	Family	主な特徴
ARMv1	ARM1	ARM の登場
ARMv2	ARM2, ARM3	乗算命令、メモリコントローラ(MMU)、キャッシュの追加など
ARMv3	ARM6, ARM7DI, ARM7M	26→32bit のアドレス空間拡張、MMU の強化、3 ステージパイプラインなど
ARMv4	ARM7TDMI, ARM9TDMI	MMU 拡張、Thumb 追加、など
ARMv5	ARM9E, ARM10E	静的分岐予測、スーパスカラ、Jazelle DBX、など
ARMv6	ARM11, ARM11E, ARM Cortex-M	マルチコア、SIMD、Thumb2、動的電圧周波数制御(DVFS)など。
ARMv7	ARM Cortex-M3, ARM Cortex-R4, ARM Cortex-A8 ARM Cortex-A9	DSP、可変キャッシュ、L2 キャッシュ、投機実行、アウトオブオーダ実行、MPU など。
ARMv8	ARM Cortex-A50 ARM Cortex-A53	64bit 命令、暗号化命令など

本論文では、スクラッチ設計なので全機能の実装は困難であり、また Thumb モードと JAVA の Bytecode 実行を現時点で対象としないことから、ARMv3(ARM7DI)の命令セットアーキテクチャを選択した。ただし、本研究の目的はマルチコア化・メニーコア化にあることから、必要とされる機能および回路については ARM アーキテクチャによらず導入し、ARM アーキテクチャに導入されていない機能も積極的に採用する。

3.2 ARM のアーキテクチャの特徴

本節では、ARM アーキテクチャで特徴的なものを紹介する。

3.2.1 命令の条件付き実行

別の命令によってセットされたステータスフラグに応じて、命令の実行と非実行と決定できる。こうすることで、小さいな if 文の対応する機械コード生成時に分岐命令が避けられる。例えば、変数 i と j があり、i ＞ j の場合、c = a - b という演算を、i ≤ j の場合、c = b - a という演算をすると仮定する。下の機械コードで実現できる。

```
CMP     Ri, Rj          ; set condition
SUBGT Rc, Ra, Rb  ; if “GT” (greater   than),
                          ; c = a – b
SUBLE Rc, Rb, Ra   ; if “LE” (less than or ; equal),
                            c = b - a
```

3.2.2 演算と算術/論理シフトの1命令同時実行

ほとんどの演算命令で基本演算と当時に第2のオペランドに対する算術または論理シフト演算を実行できる。例えば、レジスタ R3 を左に3ビットをシフトして、R2 と加算して、R0 に代入するという動作を、

ADD R0, R2, R3, LSL #3

と表記して1命令で行うことができる。

3.2.3 フラグ・レジスタの条件付き設定

演算命令において、演算結果をフラグに反映するかどうかを選択できる。例えば、減算命令である SUB 命令で、

SUB R3, R2, R4

とした時、フラグ・レジスタは影響されないが、フラグ・セットを意味する記述子“S”を SUB 命令につけて、

SUBS R3, R2, R4

とした場合、フラグ・レジスタが更新される。これを上述した条件付き実行と組み合わせれば、有効に活用できる。

3.2.4 ブロックデータ転送命令

複数のレジスタを一括してロード/ストアする命令であり、R0～R15 を任意に組み合わせて指定できる。主に、スタック操作、コンテキストのストア/ロードに使われる。

3.2.5 thumb モード

ARMv4 以降では、命令長 16 ビットの thumb モードが導入された。32 ビットの命令と比べて、平均 65%の命令で同じ機能のプログラムを記述できる。現在、本研究では ARMv3 を対象として設計を進めているが、この thumb モードは追加予定の機能である。

4 ARMv3 の FPGA 実装

4.1 アーキテクチャ概要

ARMv3 をベースとした本論文で提案するプロセッサコアの概要を図1に示す。

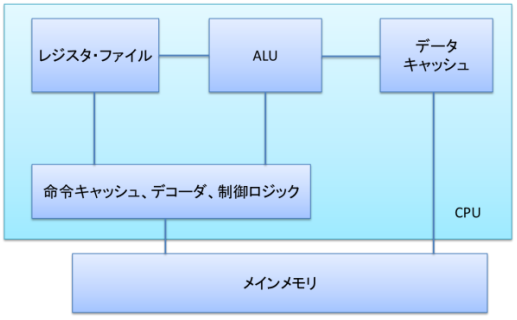

図 1：ARMv3 のブロック図

図 1 に示すように、本論文のプロセッサコアは 4 つのモジュールから構成されている。各モジュールは ARMv3 と互換性を持つが、以下の機能が追加されている。

- 命令ウィンドウの導入
- レジスタ・リネーミングの導入
- レジスタ・ファイルの増強(2 ポート→16 ポート)
- 隣接コアからのアクセスを想定した回路構造
- キャッシュコントローラの強化

以下、4.2 章で提案するプロセッサのアーキテクチャについて説明を行う。

4.2 提案プロセッサのアーキテクチャ

図 1 に示した 4 つのモジュールについて詳しく説明する。

4.2.1 パイプライン処理について

ARMv3 では 3 ステージパイプラインが採用されているが、本研究では拡張を考え命令ウィンドウやレジスタ・リネーミングなどの拡張を図っている。そこで、図 2 に示すように、命令フェッチ、デコーダ、リネーミング、命令ウィンドウ、レジスタ・ファイル、シフタ・乗算器、ALU、メモリアクセス、ライトバックの 9 ステージパイプラインとして設計することとした。

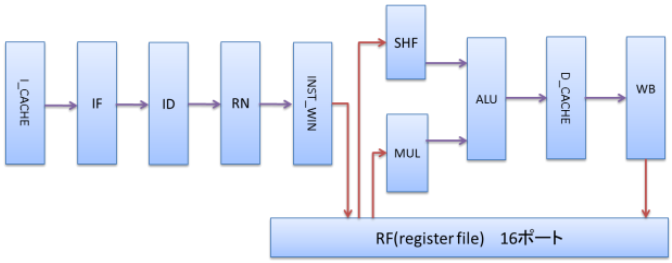

図 2 本研究におけるパイプライン構造(9 ステージパイプライン)

4.2.2 キャッシュコントローラについて

メインメモリアクセスはキャッシュメモリアクセスと比較すると 10 倍以上遅く、キャッシュミスが頻発するとボトルネックになりうる。しかし、キャッシュは有限であるため、キャッシュミスは必ず存在する。そこで、キャッシュヒット率の向上と効率的なアクセス手法が求められる。

本研究では、データ格納手法として 2-way のセットアソシアティブ方式、データ更新手法としてはライトバック方式を採用している。また、マルチコア化なども考慮し dirty ビットを導入している。これもキャッシュラインごとではなくキャッシュラインの各エントリに対して valid ビットと dirty ビットを設定することでメインメモリとの通信量の削減を図った。ライトバッファの回路構成は図 3 に示す。

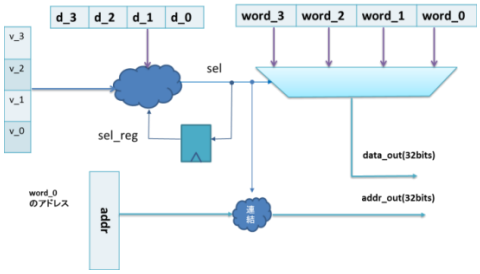

図 3：キャッシュ（ライトバッファ）の構成

4.2.3 レジスタ・ファイルについて

ARMv3 では、6 つのプロセッサモードがあり、合わせて 37 本のレジスタを持っている。図 4 にその概要を示す。

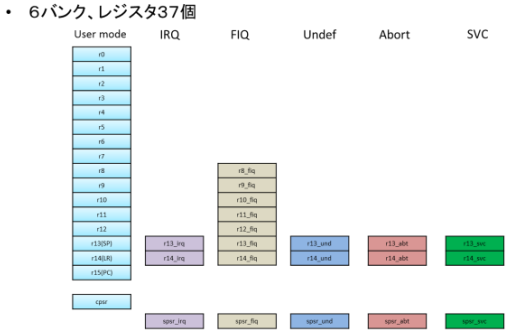

図 4:ARMv3 のレジスタ・ファイル

本研究のレジスタ・ファイルは、ブロックデータ転送命令(LDM, STM)などを考慮し、16 ポートを持っている。これにより、1 クロックでの転送が可能となる。また、マルチコア化を前提としているため、隣接するプロセッサコアのレジスタ・ファイルを直接アクセスでき、コア数に対しスケーラブルな性能向上を保証しやすい設計を導入している。このレジスタ・ファイルは FPGA の LE(logic element)を使って構成すれば、回路規模が膨大になってしまい、1 クロックでアクセス出来ない。そのため、高集積度のブロックラム(BRAM)を使用することにした。BRAM は 2 ポートしかなく、16 ポートを確保するために、レジスタ・ファイルを分割して BRAM に保存する。違うバンクのレジスタを同時にアクセスしないので、違うバンクのレジスタを同じ BRAM に保存する。レジスタ・ファイルを Verilog で記述し、Altera Cyclone IV E で実装した。BRAM を利用した構成と LE を利用した構成のリソースの使用量の比較は表 5 に示す。

4.2.4 レジスタ・リネーミングについて

アウトオブオーダ実行は、パイプラインに入ってきた命令のオペランドの依存関係を調べ、その関係性に基づき順序を動的に入れ換え実行する仕組みである。しかし、プロセッサのレジスタ数は限られているため、機械コード生成時にロケーションを再利用する必要がある。これにより、いわゆる「偽の依存関係」(Write After Read (WAR) と Write After Write (WAW)) が存在する。このデータハザードはレジスタ・リネーミングにより解消可能である。本研究では、一つの論理レジスタに対して、図 5 に示すように、8つの物理レジスタを使用することにした。物理レジスタはコミット(commit)ビットとバリッド(valid)ビットを持ち、順番に割り当てられる。

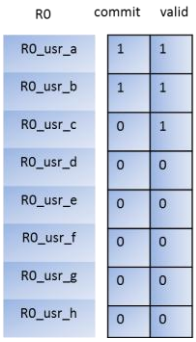

図 5：レジスタ・リネーミング

図 5 を例に説明すると、現在、R0_usr_d が使われているとする。次にレジスタ割り当てされるのは R0_usr_e である。分岐予測失敗または割り込み発生時に、現在使われているレジスタの直前のコミットされたレジスタに戻れば、プロセッサインオーダ状態へのロールバックができる。

5　実装結果

5.1　概要

本研究では、全ての回路は Verilog HDL によって記述され、各ブロックはモジュール化を前提に設計されている。ターゲットデバイスは、ALTERA Cyclone IV E (Terasic 社製 DE2) としている。現在、実装が完了している部分について報告を行う。

5.2　データキャッシュ

データキャッシュの実装は表 4 に示すことができる。データを一時的に記憶する領域として、FPGA 内部に実装されている記憶領域（BRAM: Block RAM）を用いて実装を行っている。このため、プロセッサの構造的から考えると、データキャッシュの部分が一番 BRAM を使用することになる。現在の実装では 16KB のサイズで全体の約 6%使用率となり、マルチコアで使用する上では妥当な範囲と考えている。また、データキャッシュのサイズは変更可能である。

表 4：データキャッシュのリソース使用量

	データ部	タグ部	コントローラ	アービタ	ライトバッファ	合計	FPGA全体
ロジックエレメント	272	359	181	80	789	1728 (1.5%)	114480
レジスタ	2	12	119	3	665	801 (0.7%)	114480
ブロックラム	16	8	1	0	0	25 (5.8%)	432

5.3　レジスタ・ファイル

レジスタ・ファイルには、Logic Element (LE) を利用する実装方法とキャッシュと同様に BRAM を利用する実装方法の 2 つが考えられる。この 2 つの実装結果について表 5 に示す。

表 5：レジスタ・ファイルの回路規模

	LE (logic element)	register	BRAM (block ram)
BRAMを利用した構成	963(0.8%)	0	16(3.7%)
LEを利用した構成	22095(19%)	1024(0.9%)	0
FPGA全体	114480	114480	432

レジスタ・ファイルで扱うデータ量は少ないが、表 5 より、LE を利用した構成では FPGA リソースの約 2 割を消費することがわかる。一方、BRAM 実装の問題点は、同時に 2 つのデータ（Dual-Port アクセス）しかアクセスできないという制約をもっている。そこで本研究では、BRAM を冗長して使用することで、レジスタ・ファイルとして要求される十分なポート数を確保している。このため無駄は多くなるが、全体の 4%以下で実装できることがわかっている。以上から、本研究ではレジスタ・ファイルも BRAM を利用して実装することにした。

5.4　デコーダ

デコーダでは、命令を認識し、命令の情報を取り出し、後段に渡す。主に3つのフィールドがある：制御フィールド、レジスタ・リスト、即値フィールド。リソースの使用量は表 6 に示す。

表 6：デコーダの回路規模

LE(logic element)	Register	Block ram
577	60	0

6　まとめと今後の課題

現在のリソースの全使用量を表 7 に示す。ALU 部では基本的に BRAM は使用しないため、LE 数の削減が今後の課題となる。現在の見積もりでは、5000 LEs 以下で実装できると考えており、マルチコア化を考える場合、BRAM の利用数を削減することが課題となることがわかっている。現在では、BRAM 数の制約から 1FPGA あたり 10 個のプロセッサコア実装が最大である。

表 7：現在のリソース使用率

LE(logic element)	Register	Block ram
3268	861	41
2.9%	0.75%	9.5%

完成していないモジュールを完成し全体を結合するのが目下の課題だが、表 7 に示されるように、LE 数については比較的余裕がある。そこで、Thumb モードについては今後追加し、ARMv4 に準拠することを視野にいれている。

また、マルチコア化を図6に示すが、現在進めている ARM コア間のデータ共有だけではなく、L1 キャッシュと主記憶の間の共有キャッシュ（L2 キャッシュ）を置くことを考えている。こちらの構成（バスやデータ転送方式）などについても検討していく予定である。

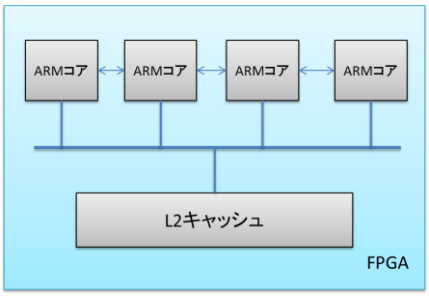

図 6：マルチコア化

7　参考文献

[ACTEL] http://www.actel.com

[Ahmed2010] Ahmed Karim Ben Salem, Slim Ben Othman and Slim Ben Saoud, *Field Programmable Gate Array-Based System-on-Chip for Real-Time Power Process Control*, American Journal of Applied Sciences Vol.7, No.1, pp.127-139, 2010.

[ALTERA] http://www.altera.com

[ARM7a] Advanced RISC Machines Ltd, *ARM7DI Data Sheet*, 1994.

[ARM7b] Advanced RISC Machines Ltd, *ARM7TDMI Data Sheet*, 1995.

[Christopher2005] Christopher R. Clark, Ripal Nathuji, and Hsien-Hsin S. Lee, *Using an FPGA as a Prototyping Platform for Multi-core Processor Applications*, In Workshop on Architectural Research using FPGA Platforms in conjunction with International Symposium on High-Performance Computer Architecture, Feb. 2005.

[Conor2011] Conor Santifort, *Amber2 Core Specification*, Amber Open Source Project, May 2011.

[James2006] James O. Hamblen and Tyson S. Hall, *Using System-on-a-Programmable Chip technology to Design Embedded Systems*, International Journal of Computers and Their Applications, Vol.13, No.3, 142-152, Sep. 2006.

[Mayan1993] Mayan Moudgill,Keshav Pingali, Stamatis Vassiliadis: Register Renaming and Dynamic Speculation: an Alternative Approach. September 15, 1993.

[Opencores] http://opencores.org

[Paul2010] Paul Dillien, *An overview of FPGA Market Dynamics*, SOCcentral, Feb., 2010.

[QuickLogic] http://www.quicklogic.com

[Stephan2012] Stephan Nolting, STORM SoC - System on Chip, Datasheet and Implementation Guide, May, 2012.

[XILINX] http://www.xilinx.com