C_001

プログラマブルなロードストアユニットと演算部が協調する再構成可能プロセッサアーキテクチャ

A Reconfigurable Processor Architecture with Programmable Load/Store Unit Cooperating Execution Part

下尾 浩正 †　山脇 彰 ‡　岩根 雅彦 ‡
Kosei Shimoo　Akira Yamawaki　Masahiko Iwane

1 はじめに

再構成可能プロセッサ (RP:Reconfigurable Processor) は，アプリケーションごとに適した内部構成を取ることにより高性能かつ低消費電力な処理の実現を図る．近年，C プログラムからハードウェアを生成する高位合成技術 [1,2] が発達し，開発期間の削減に対する要求が厳しい組込み用途において RP の利用が進んでいる [3–5].

RP はプログラム中の並列性を資源が許す限り最大限抽出して処理の高速化を図るので，基本的に命令を逐次に実行する汎用プロセッサよりも，メモリアクセスの影響が性能向上に対して大きい．

DRP [3] や DAPDNA [4] はチップ上に 1～2 クロックでアクセスできる高速な組込みメモリを搭載している．EXPRESS-1 [5] は，市販 FPGA がベースであり，その組込みメモリに 1 クロックでアクセスできる．ただし，マルチメディア処理などは大量のデータを扱うため，全データを組込みメモリに予め置いておくことは容量的に困難である．さらに，扱うデータは動的に変化するので，そのつど，外部からデータを取得する必要がある．

メモリアクセスに対して汎用プロセッサと同様にキャッシュを持つ RP もある [6,7]．ただし，キャッシュはメモリ参照の局所性がないと効果がなく，ライン単位で暗黙的に主メモリとの置換えが発生することから消費電力も大きい [8]．それゆえ，組込み用途では，ユーザが明示的に配置を制御できる組込みメモリが使用される．高位合成技術におけるメモリアクセスへの取り組みは，ハードウェアに依存しない透過な関数レベルのインタフェースの提供が主であり，主メモリアクセスの高性能化は扱われていない [1,2]．当然，アプリケーションに適した専用メモリとそのインタフェースを用意することも可能だが [9,10]，それ自体の複雑さや再利用性の低さによって開発コストの増大につながり得る．

つまり，系統的で，かつ，メモリアクセスレイテンシの隠蔽が可能なインタフェースを持ち，ユーザが明示的にアロケーションを管理できるメモリアクセス機構は RP にとって有益と考える．本論文では，そのような観点から，プログラム中のロード・ストア命令列を実行する専用プロセッサ (LSU：Load/Store Unit) と演算命令列を実行する演算処理部 (EXU：EXecution Unit) が大容量レジスタを介して協調する RP アーキテクチャを提案する．

以降で，提案 RP アーキテクチャの概要と，メモリへのアクセス・レイテンシの隠蔽手法について述べる．そして，その特徴を評価するための基礎実験を行い，提案アーキテクチャの有効性を示す．

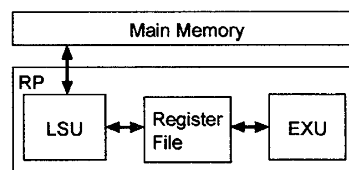

図 1　提案アーキテクチャの概念

2 提案 RP アーキテクチャ

2.1 概念

提案アーキテクチャは，図 1 のように簡潔な概念であり，プログラム中のロード・ストア命令列を実行する専用プロセッサ LSU と，演算命令列を実行する演算処理部 EXU，および，EXU に対して主メモリへの透過的なインタフェースを提供する大容量レジスタからなる．

LSU は，EXU とは並列に，メモリと大容量レジスタ間でデータを転送し，EXU は，メモリアクセスと並列に，レジスタに対してデータを読み書きする．LSU と EXU はデータの正しさを保障するために同期をとる．

LSU はロード・ストア列を実行する小規模な専用プロセッサであり，ストリームアクセスだけではなく，任意のランダムアクセスにも柔軟に対応する．

LSU と大容量レジスタは典型的な構成だが，EXU はアプリケーションごとに適したハードウェアが実装される．そのモジュールの開発コストは大きいと考えられ，再利用性や移植性が重要となる．提案アーキテクチャでは，主メモリや IO などの物理構成が大容量レジスタによって隠蔽される．すなわち，EXU に実装されるモジュールは，大容量レジスタに対する単純なインタフェースをサポートすればよく，デバイスが変わっても HDL プログラムを再合成するだけで移植できる．

2.2 プログラムの実行方式

プログラムの実行方式の概要を図 2 に示す．点線矢印がプログラムの変換過程を示す．

入力の C プログラム (図中左上) は，プログラム中のロード・ストア列と，それ以外の演算列とに変換される．2 つのコード列が LSU と EXU 上で並列に実行されるため，変換時にはデータの依存解析 [11] をもとに同期指示子も挿入される．変換されたフォーマットを専用言語として直接ユーザが入力することも可能である．

そして，ロード・ストア列は LSU のマシン語列に，演算列は EXU 上に実装される FSM(Finite State Machine) とデータパスからなるハードウェアスレッド (HWTD) へと変換される．LSU のマシン語列は，RP 上のメモ

† 佐世保工業高等専門学校, Sasebo National Collage of Technology
‡ 九州工業大学, Kyushu Institute of Technology

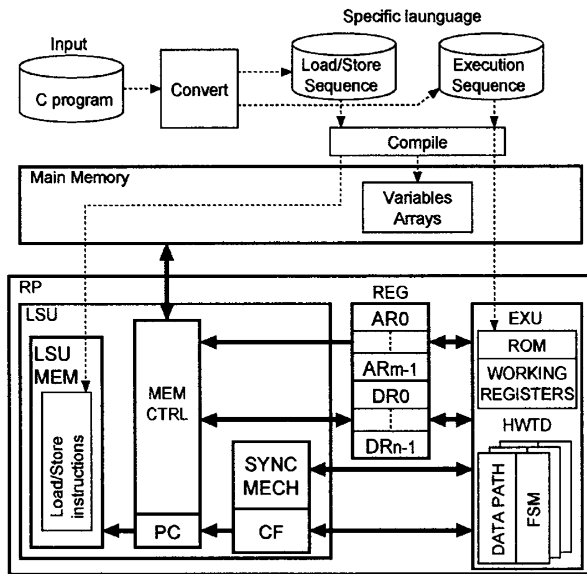

図2　プログラムの実行方式

リ (LSUMEM) に ROM として組み込まれるか，主メモリから必要に応じて読み出される．演算列に関しては，データ依存を満たす範囲で RP の資源が許す限り，複数のハードウェアスレッドに並列化される [1,2].

LSU と EXU, 及び, EXU 内のハードウェアスレッドは，同期をとりながら協調動作する．同期指示子に従って，LSU のマシン語では同期コードが各命令中のフィールドに，EXU では同期を実現するための状態が各ハードウェアスレッドの FSM に挿入される．同期は LSU 内の同期機構 (SYNCMECH) を介して実施され，サポートされる同期は生産者-消費者間同期とバリア同期である．

主メモリと EXU, 及び, EXU 内のハードウェアスレッド群は，大容量レジスタを介してデータを通信する．大容量レジスタはアドレス用 $(AR_0 \cdots AR_{m-1})$ とデータ用 $(DR_0 \cdots DR_{n-1})$ に分かれる．アドレス計算は EXU によって実施され，その結果が AR に格納される．LSU は，当該 AR 内のアドレスを用いて主メモリにアクセスし，DR と主メモリ間でデータを転送する．

EXU がアドレス計算をする際には，変数，及び，配列のベースアドレスとオフセットを格納した ROM を用いる．さらに，この ROM は変数の初期値も保持する．また，ワーキングレジスタは，DR へのアクセスを削減するために RP の資源が許す限り用意され，各ハードウェアスレッドが一時データの保存に使用する．

分岐条件の評価は EXU が実施し，その結果を CF(Condition Flag) に反映する．CF は任意長のレジスタであり，当該ビットをセットすることによって分岐条件が EXU から LSU に通知される．LSU は CF の内容を見て，PC(Program Counter) の内容を変更し分岐する．

2.3　LSU 命令

LSU 命令のフォーマットは図 3 に示すように，オペコード部 (OP)，同期コード部 (SYNC)，対象レジスタ部 (DR)，およびアドレスレジスタ部 (AR) から成る．また，条件分岐では，対象レジスタ部とアドレスレジスタ部が，条件コード部 (CC) と分岐アドレス部 (LADR) に読み替えられる．LSU 命令の特徴としては，同期の指定を命令に埋め込んでおり，命令の実行と同時に同期を実現できる点が挙げられる．

OP	SYNC		DR	AR	CNT	STR
	SOP	STRG	CC		LADR	

(a) Instruction format

```
Non operation      : NOP   SYNC
Load stream        : LDS   DRn, ARn,  CNT, STR, SYNC
Line load stream   : LLDS  DRn, ARn,  CNT, STR, SYNC
Store stream       : STRS  DRn, ARn,  CNT, STR, SYNC
Line Store stream  : LSTRS DRn, ARn,  CNT, STR, SYNC
Jump               : JMP   CC,  LADR, SYNC
```

(b) Mnemonic

図3　LSU 命令

まず，オペコードによって指定される命令を説明する．ロードストリーム命令 (LDS) は，メインメモリから大容量レジスタに変数を 1 ワード分読み出し，ラインロードストリーム命令 (LLDS) は変数を 1 ライン分読み出す．1 ラインとは複数ワードからなるブロックを意味し，そのワード数は実装依存である．ストアストリーム命令 (STRS) は，大容量レジスタからメインメモリに変数を 1 ワード分書き込み，ラインストアストリーム命令 (LSTRS) は 1 ライン分書き込む．以上のロード・ストア命令では，メモリとデータ転送するレジスタが対象レジスタ部で指定され，転送先アドレスを格納したレジスタがアドレスレジスタ部で指定される．また，転送回数 (CNT) と転送間のアドレス・ストライド (STR) も指定できる．なお，複数の転送を実施する際に同期が指定されていたら，各転送で同期が取られる．ジャンプ命令 (JMP) は，条件コード部で指定したビットが 1 ならば，LADR へジャンプする．無動作命令 (NOP) は指定された同期のみを実行する．

同期コード部は，無同期，リリース同期 (RLS)，ウェイト同期 (WAIT)，及び，バリア同期 (BAR) を指定する．同期コードの上位 2 ビット (SOP) を用いて 4 つの同期が指定され，残りのビット (STRG) で同期を取る LSU とハードウェアスレッドが指定される．リリース同期では，命令の実行時に STRG の当該ビットが 1 ならば，その命令は待たされ，0 ならば，命令の実行後に STRG の当該ビットを 1 にセットし，次命令に実行が移る．ウェイト同期は，命令の実行時に STRG の当該ビットが 0 ならば，その命令は待たされ，1 ならば STRG の当該ビットを 0 にリセットしてその命令を実行する．つまり，消費側が WAIT 付きの命令を実行して生産側が RLS するまで待つような生産者-消費者間の同期に使用される．バリア同期は，命令の実行時に STRG の当該ビットで指定された LSU やハードウェアスレッドがバリア同期付きの命令に到達するまで待つ．指定された LSU やハードウェアスレッドがすべてバリア同期付きの命令に到達したら，ブロックされていた命令を実行する．

2.4　同期機構

リリース同期とウェイト同期は N×N 個のフリップフロップ，バリア同期は N ビットのレジスタにより実現される (図 4)．N は LSU とハードウェアスレッドの総数であり，規則的で簡素な構成である．

リリース同期とウェイト同期では，行方向のウェイトレジスタ $(WAITR_0 \cdots WAITR_{N-1})$ と，列方向のリリースレジスタ $(RLSR_0 \cdots RLSR_{N-1})$ が LSU とハードウェアスレッドによって使用される．リリース同期を実行する際は，命令の実行後に自身の RLSR のリリース先に対応するビットを 1 にセットする．ウェイト同期を実行する際は，命令の実行前に自身の WAITR のリリース元に

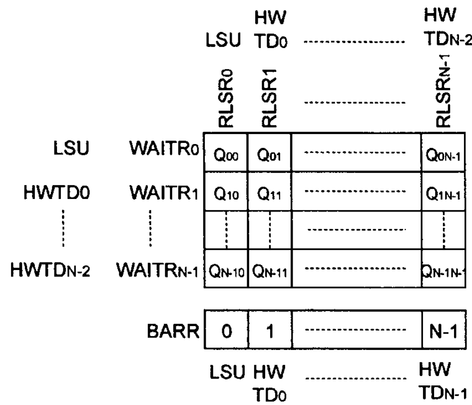

図4　同期機構

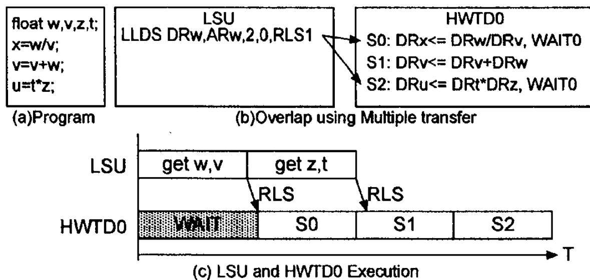

図5　メモリアクセスと演算実行のオーバラップ例

対応するビットが0ならば，そのビットが1になるまで待つ．ビットは図3(a)のSTRGによって指定される．

例えば，LSUがハードウェアスレッド0(HWTD0)に対してリリース同期し，HWTD0がLSUからのリリースをウェイト同期で待つ場合を考える．HWTD0は$WAITR_1$を読み出し，LSUに対応するビット0をチェックする．0ならばHWTD0は当該ビットが1になるまで待つ．LSUはリリース同期の際に$RLSR_0$を読み出し，HWTD0に対応するビット1をチェックする．0ならば，LSUは命令を実行し，当該ビットを1にセットして次の命令に移る．$WAITR_1$のビット0が1になったので，HWTD0はブロックされていた命令を実行する．

バリア同期では，同期を取るLSUやハードウェアスレッドの組をバリアグループとよぶ．バリアグループは，1つのバリアマスタとバリアスレーブからなる．バリアマスタはバリアグループをSTRGで指定し，バリアスレーブはSTRGがオール0である．バリアスレーブはバリアレジスタ(BARR)の自身に対応するビットをチェックし，0ならば1にセットして再び0になるまで待つ．バリアマスタはSTRGで指定されたビット群がすべて1になるまで待ち，すべて1になったらそれらのビットをリセットする．

3　メモリアクセスレイテンシの隠蔽

LSUは，EXUと独立にメモリと大容量レジスタ間でデータを転送し，EXUの演算実行とメモリアクセスをオーバラップさせる．図5にその例を示す．図5(a)の各変数は宣言通りの順番でメモリに配置されているとし，図5(b)のLLDS命令は1回で2ワードを読み出すとする．図5(b)のLLDSは2回のラインを読み出すが，1回目のライン転送でw，vがDRwとDRvへ，2回目でt，zがDRt，DRzへ読み出される．従って，生産者側のLSUはLLDSでRLS同期を，消費側のHWTD0ではS0とS2でWAIT同期を取る．

図5(c)は，LSUとHWTD0の動作の様子である．LSUの1回目の転送ではメモリアクセスレイテンシのためにS0の実行前にHWTD0で待ちが発生している．しかしながら，LSUの2回目の転送は，HWTD0のS0，S1の実行とオーバラップでき，HWTD0がS2を実行する際はWAIT同期で待ちが発生していない．

4　実験および考察

4.1　実験環境

実験では，図2と等価なモデルをVHDLによって設計し，RTLシミュレータによってクロック数を測定した．設計したLSUは3段パイプライン構成であり，ハードウェアスレッドを最大4個までサポートする．大容量レジスタは512個の32ビットレジスタを持つ(128個の4バンク構成)．各レジスタへの読み書き，及び，フラグのセット・リセットは1クロックで完了する．EXUでは単精度の浮動小数点演算器(加算，乗算，除算)を用いたが，それぞれ，12クロック，9クロック，28クロックのレイテンシである．メモリアクセスは，1ワードで2クロック，1ライン(4ワード)で7クロックかかる．

比較対象としてEmMIPS [12]にライトバックキャッシュとFPU(Floating Point Unit)を搭載したプロセッサコアを用いた．EmMIPSは5段パイプラインの32ビットスカラプロセッサであり，MIPSII命令のサブセット互換である(1つの遅延スロットを持つ)．FPUのレイテンシはEXUと同じである．キャッシュは16KBの容量で1ラインが4ワードからなる2ウェイセットアソシアティブ構成である．キャッシュアクセスはヒット時に1クロック，ミス時にライン転送で7クロックかかる(実験では初期参照ミスのみ発生)．

4.2　シミュレーション結果

評価にはFLOPSベンチマークの一部と，256個の配列要素の和(SUM)を用いた．前者は，図5と同様にオーバラップできる部分があった．後者に関しては，演算とメモリアクセスがオーバラップできるように，4回ループ展開した．つまり，現繰り返しで，次繰り返しで使う4ワードをLLDS命令で読み出すように再構成した．

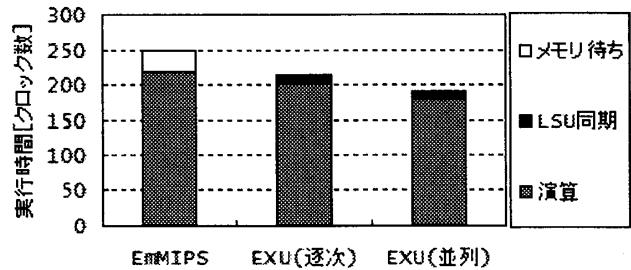

図6　FLOPSの結果

図6，図7に結果を示す．縦軸は実行時間(クロック数)である．内訳の演算はEmMIPSとEXUで演算にかかった時間を表し，メモリ待ちはEmMIPSがメモリアクセスでストールした時間である．LSU同期はLSUとEXU間で同期にかかった時間(同期待ちを含む)であり，EXUがメモリアクセスで待った時間と等価である．

EXU(逐次)は，演算列をプログラムの順番通り逐次に

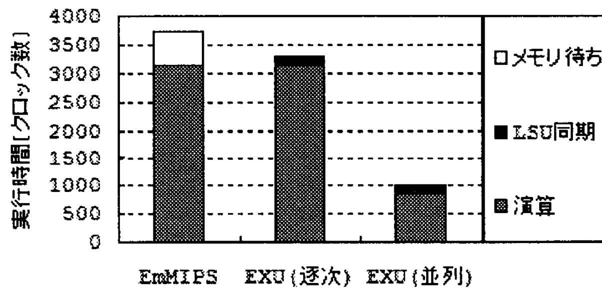

図7 SUMの結果

表1 FPGAへの実装結果

	LUT	FF	BRAM	SPEED
EmMISP	8104(37%)	2721(12%)	14(19%)	15.3ns
LSU+REG	993(4%)	179(1%)	5(6%)	11.2ns
FLOPS(S)	2754(12%)	1262(5%)	5(6%)	14.9ns
FLOPS(P)	3641(16%)	1980(9%)	5(6%)	14.8ns
SUM(S)	1563(7%)	714(3%)	5(6%)	17.2ns
SUM(p)	3707(17%)	2367(11%)	5(6%)	18.1ns

実行させた場合の結果である．一方，EXU(並列)は並列化できる部分を複数のハードウェアスレッドとして並列化した場合の結果であり，FLOPSでは並列度2，SUMでは並列度4である．

結果から，逐次版でもメモリアクセスを隠蔽できたことからEmMIPSに対してEXUは性能向上を達成できている．さらに並列化することで，EmMIPSに対して，FLOPSで1.30倍，SUMでの3.71倍の性能向上を得た．SUMでは演算時間の大幅な短縮にも関わらず，メモリアクセスがほぼ隠蔽できている．

4.3 FPGAへの実装

EmMIPS，LSUと大容量レジスタ，及び，各評価プログラムをFPGAへ実装した(表1)．FPGAはXilinx社のVirtex4 XC4VLX25-10FF668で，論理合成にはXilinx社のXSTを使用した(オプションはすべてデフォルト値)．括弧内のSは逐次版をPは並列版を意味する．

表1から，LSUと大容量レジスタのみの場合，ハードウェアの使用量はわずか4%であった．また，動作速度に関してもクリティカルパスはEXUにあり，LSUは動作速度を低下させていない．

EmMIPSに対してEXUはすべての場合でハードウェア規模が半分以下であり，汎用プロセッサに対してアプリケーションに特化できるRPの効果も確認できた．SUMでは動作速度がEmMIPSよりも若干遅くなっているが，クロック数の差から十分な性能を達成できている．

また，SUMの並列化版は，逐次版に対してハードウェア量を2倍以上に増加させたが，FLOPSでは1.3倍程度しか増加していない．これは，FLOPSではデータの真依存が多く，1つの式しか並列化できなかったことによる．それゆえ，図6のように性能差が逐次版と並列版でSUMよりも小さかった．

5 おわりに

RPはプログラムに適応したハードウェアに自身を再構成することによって処理を高速化する．そのため，基本的に命令を逐次に実行する汎用プロセッサよりも，メモリアクセスの高性能化はより重要な課題である．本論文では，プログラム中のロード・ストア命令列を実行するLSUと，演算命令列を実行するEXUが大容量レジスタを介して協調するRPアーキテクチャを提案した．

これにより，LSUが主メモリアクセスを行い，その間にEXUは演算処理に専念できる．さらに，EXUは簡単なレジスタアクセスを介して透過的に主メモリへアクセスでき，提案アーキテクチャは再利用性や移植性にも富むと考えられる．

実験を通して，メモリアクセスと演算処理のオーバラップにより，良好な性能が得られることが確認された．FPGAへ実装した結果，単純なロード・ストア専用プロセッサであるLSUと単なるレジスタからなる提案アーキテクチャはハードウェア規模も小さく，動作速度にも悪影響を与えなかった．

今後は，より本格的なアプリケーションを用いて提案アーキテクチャを評価する予定である．また，LSU命令列と演算命令列の効率的な分割手法やハードウェア化手法の検討，及び，実システムでの評価も課題である．

参考文献

[1] D. Pellerin, et al. *Practical FPGA Programming in C*. Prentice Hall, 2005.

[2] S. Gupta, et al. Hardware and Interface Synthesis of FPGA Blocks using Parallelizing Code Transformations. In *Proc.on Parallel and Distributed Computing and Systems*, pp. 392–397, 2003.

[3] 長谷川ほか．動的再構成可能プロセッサを用いたIPsec向け暗号処理アクセラレータの設計と実装．信学論, Vol. J89-D, No. 4, pp. 743–754, 2006.

[4] T. Sugawara, et al. Dynamically Reconfigurable Processor Implemented with IPFlex's DAPDNA Technology. *IEICE TRANS.INF.&SYST.*, Vol. E87-D, No. 8, pp. 1997–2003, 2004.

[5] 柴村ほか．EXPRESS-1:プロセッサ混載FPGAを用いた動的セルフリコンフィギャラブルシステム．信学論, Vol. J89-D, No. 6, pp. 1120–1129, 2006.

[6] J.R. Hauser, et al. Garp:A MIPS Processor with a Reconfigurable Coprocessor. In *Proc.of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 24–33, 1997.

[7] T.J. Todman, et al. Reconfigurable computing: architectures and design methods. *IEE Proc.-Comput.Digit.Tech*, Vol. 152, No. 2, pp. 193–207, 2005.

[8] 中村ほか．SCIMAにおける性能最適化手法の検討．情処学論, Vol. SIG5, No. 41, pp. 15–27, 2000.

[9] G. Kuzmanov, et al. Multimedia Rectangularly Addressable Memory. *IEEE Trans.on.Multimedia*, Vol. 8, No. 2, pp. 315–322, 2006.

[10] S. Martinez, et al. An Image Comparison Circuit Design. In *Int'l Proc.on Reconfigurable Computing and FPGA*, pp. 2–2, 2005.

[11] M.J. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley, 1995.

[12] 九州工業大学 岩根研究室．Embedded MIPS コア (EmMIPS). `http://www.ds.ecs.kyutech.ac.jp/`.