社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

機械語の複数部分を高速化する CPU密結合型ハードウェアアクセラレータ

佐竹　俊亮† 　石浦菜岐佐† 　田村　真平† 　冨山　宏之†† 　神原　弘之†††

† 関西学院大学 理工学部 〒669–1337 兵庫県三田市学園 2-1
†† 立命館大学 理工学部 〒525–8577 滋賀県草津市野路東 1 丁目 1-1
††† 京都高度技術研究所 〒600–8813 京都市下京区中堂寺南町 134 番地

あらまし　本稿では, 機械語プログラムの指定区間を CPU と密結合するハードウェアアクセラレータに変換する手法において, 複数の区間を処理対象にするためのハードウェア構成法を提案する. それぞれの区間をハードウェアに変換して CPU に並列接続するのではなく, 一つのハードウェアモジュールが複数の区間の処理を実行できる構成をとる. これにより, 複数の処理の間でハードウェア資源や制御が共有できるため, ハードウェアの利用効率が良くなる. また, プログラムの複数区間が合成可能になれば, アクセラレータからのソフトウェアサブルーチンの呼び出し等, 複雑な制御構造を扱うことも可能になる.

キーワード　組み込みシステム, ハードウェア/ソフトウェア協調設計, ハードウェアアクセラレータ, 高位合成

Speeding up Multiple Sections of Binary Code by Hardware Accelerator Tightly Coupled with CPU

Satake SHUNSUKE†, Nagisa ISHIURA†, Tamura SHIMPEI†, Hiroyuki TOMIYAMA††, and Hiroyuki KANBARA†††

† Kwansei Gakuin University, 2-1 Gakuen, Sanda, Hyogo, 669–1337, Japan
†† Ritsumeikan University, 1-1-1 Noji-Higashi Kusatsu, Shiga, 525–8577, Japan
††† ASTEM RI/KYOTO, 134 Chudoji Minamimachi, Shimogyo-ku, Kyoto, 600–8813, Japan

Abstract　This article presents an improvement over the hardware accelerator tightly coupled with a CPU. While the previously proposed method assumes only a single fragment from a binary code to be synthesized into an accelerator, our method attempts to accelerate multiple fragments. Instead of connecting multiple acceleratos corresponding to the fragments in parallel, a single hardware module is synthesized which is capable of accelerating the multiple sections. This enables sharing of datapath resources as well as the control among multiple tasks, which makes the accelerator cost-efficient. Furthermore, the capability of handling multiple code fragments makes it possible to synthesize complex control structures, such as calling software subroutines from a hardware accelerator, into hardware.

Key words　embedded systems, hardware/software codesign, hardware accelerator, high-level synthesis

1. はじめに

近年, 組み込みシステムには厳しい電力制約下で益々多くの複雑な機能の実装が求められるようになっている. CPU とソフトウェアの組合せでこの要求が実現できない場合には, 処理の一部分をハードウェア化することによりその高速化/低消費電力化が図られるが, ソフトウェアとハードウェアから成るシステムをどのような構成で実現するか, また, 益々厳しくなる設計期間短縮化の要求下でこのようなシステムをいかに効率的に設計するかが課題となっている.

CPU で実行されるソフトウェアの一部をハードウェア化することにより高速化する手法は種々提案されている. Stitt [3] の提案する Binary Synthesis は, 機械語プログラムをハードウェアに合成するものであり, CPU による処理のボトルネックになる部分をこの手法で自動的にハードウェア化できる. しかし, 制御の受け渡しはポーリングによる方法しか提案されておらず,

制御のためのオーバヘッドが大きい．また，リロケーションまで含めた機械語プログラムの修正が必要になる．瀬戸 [2] はソフトウェアが行っている処理の一部を高位合成技術によってカスタム命令の系列に変換して高速化する方法を提案している．ソフトウェアとハードウェアの制御切換えのオーバヘッドはほとんどなく，CPU のパイプライン構造やフォワーディングユニットをそのまま利用できるため，データの受け渡しのオーバヘッドもほとんどない．しかし，ハードウェア化できる範囲は単一の基本ブロックに限られ，またその規模も追加可能な専用命令数により限定される．また，この方法も機械語の修正が必要になる．Shee [1] は C/C++ から高位合成を用いて CPU から起動可能なコプロセッサを合成する手法を提案している．コプロセッサは CPU の汎用レジスタにアクセスできるためデータの受け渡しが高速に行える．しかし，CPU とコプロセッサのデータハザードを回避するための待ち合わせが必要になる等，CPU とコプロセッサの制御の切換えにオーバーヘッドが生じる．

このように，ソフトウェアを部分的にハードウェア化する手法に関しては，(1) ハードウェアの起動やデータ授受のオーバーヘッド，(2) ハードウェア化できる範囲の制約，(3) ソフトウェアの修正の複雑さ，が解決しなければならない課題として挙げられる．

これらの課題に対し，戸田 [4] は，CPU と密に結合するハードウェアの合成手法を提案している．この手法は，Binary Synthesis と同様，機械語プログラムの指定部分 (区間) を高位合成技術によりハードウェア化するものである．しかし，CPU とハードウェア間の制御の受け渡しは，ハードウェアが CPU のプログラムカウンタを監視/更新することにより実現するため，オーバヘッドがほとんどない．また，ハードウェアはレジスタファイルだけでなく，フォワーディングユニット，メインメモリに直接アクセスするため，CPU とのデータ授受のオーバヘッドも非常に小さい．また，元の機械語プログラムやコンパイラは全く修正する必要はない．

しかし，[4] はプログラムの単一区間のハードウェア化しか想定しておらず，プログラムの複数部分をハードウェア化することはできない．また，このため，この手法を適用できる制御構造も限定されていた．

これに対し本稿では，[4] の手法を複数区間のハードウェア化が可能なものに拡張する．単純に複数のハードウェアを並列接続するとハードウェアの利用効率が悪くなるため，単一のハードウエアで複数区間の処理を実行する，すなわち，複数区間の処理の間でハードウェア資源を共有する方法を提案する．複数区間がハードウェア化できることにより，単純な基本ブロックやループだけでなく，ハードウェア化される区画への飛び込み，ハードウェア化される区間からのソフトウェアサブルーチンの呼び出し等，広範な制御構造を扱えるようになる．

以下，本稿では，2 章で [4] の手法について概観した後，3 章で複数区間のハードウェア化手法と新たにハードウェア化可能となる制御構造について述べる．4 章では CPU に MIPS を用いた場合の実験結果について述べ，5 章で本稿を総括する．

2. CPU と密結合するアクセラレータ [4]

戸田の手法 [4] は，図 1 に示すように，機械語プログラム中の指定された区間をハードウェア化するものである．ハードウェア化する区間は基本ブロックでもよいし，条件分岐やループを含んでもよい．区間の大きさに制限はないが，数命令〜数十命令程度を想定している．

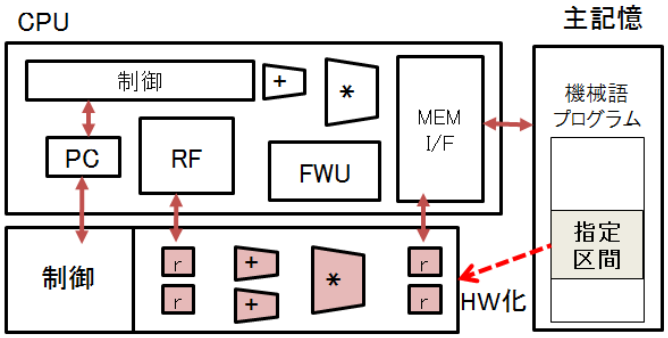


図 1　CPU と密結合するアクセラレータ

ハードウェア (以下本稿では「アクセラレータ」と呼ぶ) は CPU のプログラムカウンタ (PC) の値を監視し，その値が区間の先頭番地に達すると処理を開始する．アクセラレータは，起動されると PC の値を固定し，CPU に NOP を供給し続ける．処理が完了すると，実行終了番地の次の番地を PC に書き込み，CPU に制御を戻す．この方式により，CPU とアクセラレータの制御の受け渡しにおけるオーバーヘッドを極めて小さくなる．

アクセラレータは，CPU のレジスタファイルやフォワーディングユニットに直接アクセスすることによりデータを受け取り計算結果を受け渡す．またアクセラレータは CPU とメモリインタフェースを共有することにより CPU と同じメモリ空間にアクセスする．

しかし，文献 [4] では単一の区間を実行するアクセラレータしか想定しておらず，複数区間のハードウェア化には言及していない．

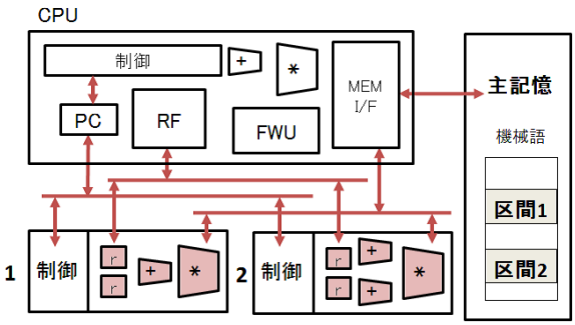
3. 機械語の複数区間のハードウェア化

3.1 複数のアクセラレータの統合

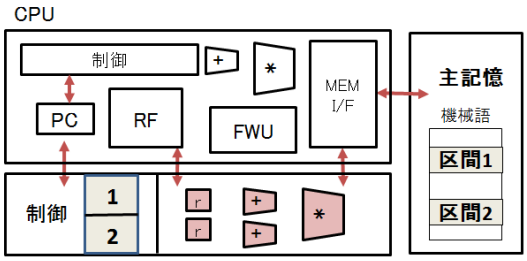
機械語の複数の区間それぞれハードウェア化し，図 2(a) のように単純に並列接続すれば，複数区間の実行を高速化できる．アクセラレータはそれぞれ PC を監視し，自分の実行番地に達すれば処理を開始する．　この際，起動したアクセラレータが CPU とハードウェア間のバスを占有する．

アクセラレータは同時に 1 つしか動作しないにもかかわらず，それぞれがデータパスを有する．また，全てのアクセラレータが PC を監視することになり，ハードウェアの利用効率が悪い．

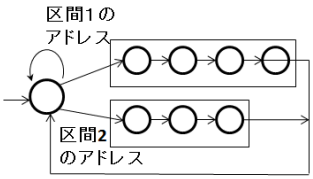
そこで本稿では，複数のハードウェアを 1 つに統合する構成を提案する．図 2(b) に提案するハードウェア構成を示す．制御部および演算器，レジスタ等のハードウェア資源は複数の区間で共有する．制御は図 2(c) のように，PC を監視し，その値がい



(a) 複数アクセラレータの並列接続



(b) 1 つのアクセラレータへの統合



(c) 統合されたアクセラレータの制御回路

図 2　機械語の複数区間のアクセラレータ化

ずれかの区間の先頭アドレスに一致した場合, その処理を実行する状態に遷移する. 処理終了後は, PC に次のアドレスを書き込んで CPU に制御を戻し, PC を監視する状態に戻る.

3.2　ハードウェア化可能な制御

機械語の複数区間をハードウェア化すると, 単にハードウェア化できる部分を増やせるだけでなく, ハードウェア化可能な制御構造の範囲も拡大できる. 本手法により新たにハードウェア化が可能になった制御構造を以下に示す.

(1) ハードウェア化区間への飛び込み

図 3 のようにソフトウェアからハードウェア化する区間に飛び込みがある場合, これまでの方法ではこの区間をハードウェアにすることができなかった. しかし, 当該区間を 2 つに分割して両区間の先頭アドレスを監視するようにすれば, このような制御構造のハードウェア化が可能になる.

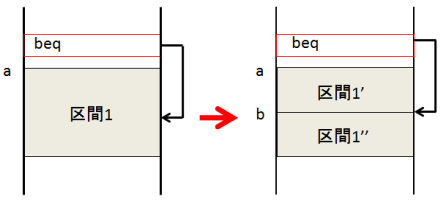


図 3　ハードウェア区間への飛び込み

(2) ソフトウェア関数の呼び出し

図 4 のように, 区間がソフトウェアのサブルーチン呼び出しを含む場合, 呼び出し (戻り番地の保存やレジスタの退避を行って, 区間から飛び出すこと) は従来でも可能であったが, サブルーチンからの戻りが不可能であった. しかし, 呼出し点の直後で区間を分割すれば, (1) と同様の方法で戻りも可能になる.

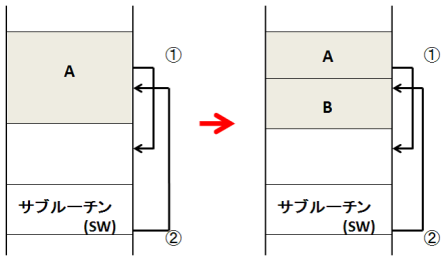
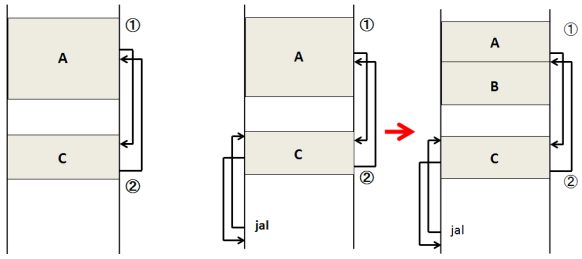


図 4　ソフトウェア関数の呼び出し

(3) ハードウェア関数の呼び出し

サブルーチンをハードウェア化してソフトウェアから呼び出すことは, 従来の手法でも可能であった. サブルーチンの起動は区間の実行開始と同じであり, 戻りは保存されたアドレスへの分岐として実現できる.

図 5(a) のようにサブルーチンを含むハードウェアがハードウェアからのみ呼び出される場合は, 呼び出しや戻りは PC への書き込みを介さず, ハードウェアの状態遷移で実現できる. これに対し図 5(b) のようにサブルーチンを含むハードウェアがソフトウェアからもハードウェアからも呼ばれる場合は, プログラムカウンタを介した制御の切り替えが必要になる. これは (2) と同様に呼び出し側のハードウェアを分割すれば実現可能である.



(a) ハードウェアのみからの呼び出し　　(b) ソフトウェアとハードウェア両方からの呼び出し

図 5　ハードウェア関数の呼び出し

4.　実装と実験

4.1　実　　装

MIPS R3000 互換プロセッサ [5] にアクセラレータを接続する設計を行った. MIPS とアクセラレータの接続関係の概要を図 6 に示す. アクセラレータは図中の (a) で PC を監視し, 区間の先頭番地に達すればその区間の処理を起動する. アクセラレータ起動時は, 同じ番地を (b) から PC に書き込み続け, CPU の処理を停止させるための NOP を (c) より供給し続ける. CPU 側に制御を渡す場合には, 実行終了番地の次の番地を

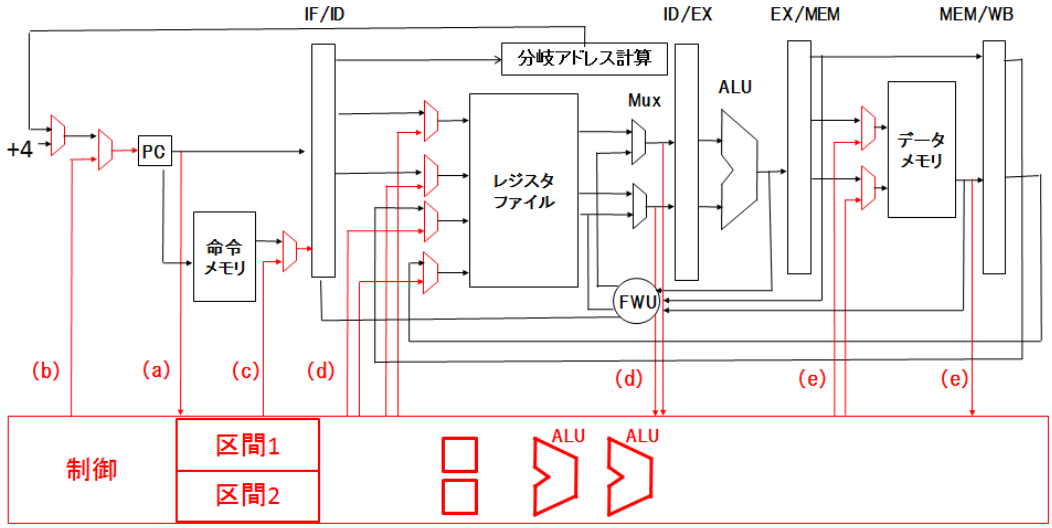


図 6　MIPS とアクセラレータの接続

(b) より PC に書き込む．データの授受には (d)(e) を用い，レジスタファイルやメインメモリにアクセスする．この構成ではフォワーディングユニットへのデータ供給は行っていない．

動作タイミングを図 7 に示す．図中の①で PC の値がアクセラレータの実行開始番地に達すると，次のサイクルでアクセラレータが処理を開始する．アクセラレータの起動前に CPU で実行されている命令とデータ依存がない演算は，②のサイクルから実行できる．依存関係があるものは，③のようにデータハザードが起きないサイクル以後に実行する．アクセラレータは④で PC に値を書き込むことにより，CPU に制御を戻す．CPU の実行再開は，データハザードが生じない範囲でアクセラレータの実行と重複していてもよい．

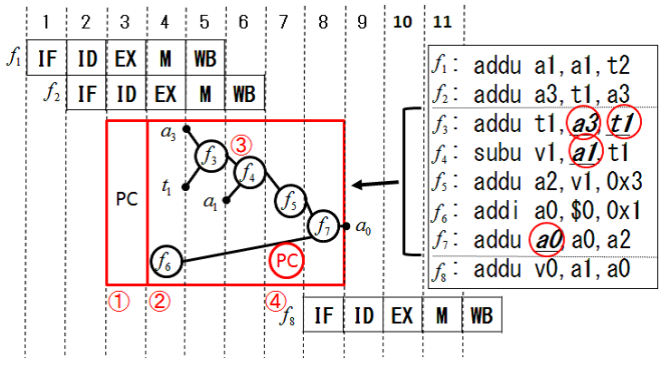


図 7　動作タイミング

4.2　動作確認

簡単なプログラムで MIPS と接続したアクセラレータの動作確認を行った．プログラムを図 7 に示す．動作確認には ModelSim Xilinx Edition-III 6.4b を用い，各テストを MIPS 単体で実行した結果と，指定区間をハードウェア化して実行した結果得られる 1 番レジスタの値を比較することにより行った．アクセラレータは手で設計した．

テスト 1 (図 8(1)) は，MIPS から 2 つの区間 HW1 と HW2 が起動でき，実行後 MIPS に制御が戻せることを確認するものである．

テスト 2 (図 8(2)) は，MIPS が起動したアクセラレータが途中で MIPS に制御を戻すことなく HW1 から HW2 に分岐することを確認するものである．

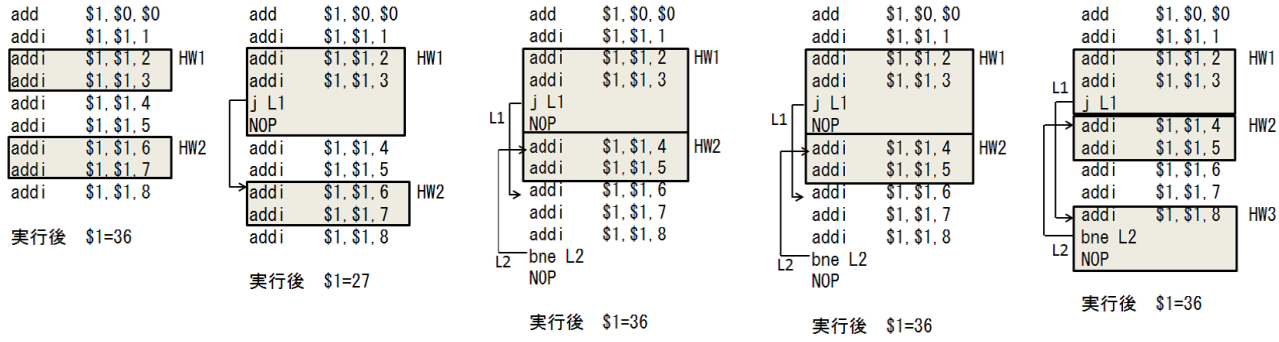
テスト 3 (図 8(3)) ，テスト 4 (図 8(4)) ，テスト 5 (図 8(5)) は 3.2 節で述べた制御であり，それぞれハードウェア化区間への飛び込み，ハードウェアからソフトウェアサブルーチンの呼び出し，ハードウェアからハードウェアサブルーチンの呼び出しの動作を確認をしている．

以上のテスト全てについて，アクセラレータを使う場合と使わない場合で同じ結果が得られることが確認できた．

4.3　サイクル数と回路規模の評価

AES 暗号化処理の一部をハードウェア化し，サイクル数と回路規模を評した．ハードウェア化した箇所はブロックデータの並び替えを行う 2 区間である．いずれの区間とも 20 命令から成り，2 重ループを含む．ハードウェア化は手動で行った．チェイニングを行っており，1 サイクルで 2 回の加算を行っているが，ループパイプライニングは施していない．CPU 及びアクセラレータは FPGA Xilinx Spartan 3E 上に Xilinx ISE 11.1 で論理合成した．また，動作のシミュレーションは ModelSim Xilinx Edition-III 6.1e により行った．

結果を表 1 に示す．「CPU 単体」は全ての計算を MIPS だけで行った場合を，「CPU+ハードウェア (1 区間)」はハードウェアを併用した場合を表す．「CPU+ハードウェア (2 区間)」はハードウェアを 2 つ併用した場合を表す．「サイクル数」は対象区間の実行のみに要したサイクル数である．当該処理に関しては，LUT (Look Up Table) 数 14.3% の増加でサイクル数を 49.1% 削減することができた．2 区間を独立にハードウェア化した場合の LUT 数はおよそ 22.2%増なので，これに比べると本稿の構成法では 2 区間をハードウェア化するための LUT 数増を約 64%に抑制できている．



(1) テスト 1　(2) テスト 2　(3) テスト 3　(4) テスト 4　(5) テスト 5

図 8　動作確認のテスト

表 1　ハードウェア量と実行時間の評価

	LUT 数	遅延 (ns)	サイクル数
CPU 単体	6967 (100.0%)	24.576 (100.0%)	414 (100.0%)
CPU+HW(1 区間)	7743 (111.1%)	31.328 (127.5%)	314 (75.8%)
CPU+HW(2 区間)	7963 (114.3%)	31.440 (127.9%)	211 (50.9%)

5. むすび

本稿では, CPU 密結合型ハードウェアアクセラレータにより機械語の複数部分を高速化する手法を提案した. 単一のハードウェアモジュールで複数区間の処理を実行することにより, ハードウェアの利用効率を向上させた. また, プログラムの単一区間だけをハードウェア化する場合に比べ, より複雑な制御構造をアクセラレータ化できるようになった.

今回, アクセラレータの設計は人手で行ったが, 現在機械語からアクセラレータを自動合成する処理系の開発を進めており, その完成と最適化の強化が今後の課題である.

謝　辞

本研究を行う上で多くのご助言, ご協力頂いた元立命館大学の中谷嵩之氏, 元京都大学の矢野正治氏に感謝します. また, 本研究に関してご協力頂いた織野真琴氏, 伊藤直也氏はじめ, ご討論頂いた関西学院大学石浦研究室の諸氏に感謝します.

文　献

[1] Seng Lin Shee, Sri Parameswaran, and Newton Cheung: "Novel architecture for loop acceleration: a case study," in Proc. Workshop on Hardware/Software Codesign+International Symposium on System Synthesis '05, pp. 297–302 (Sept. 2005).

[2] 瀬戸謙修, 藤田昌宏: "高位合成技術を利用したカスタム命令自動生成手法," 情報処理学会 DA シンポジウム 2006, pp. 49–54 (July 2002).

[3] Greg Stitt and Frank Vahid: "Binary Synthesis," ACM Trans. on Design Automation of Electronic Systems, vol. 12, no. 3, article 34 (Aug. 2007).

[4] 戸田 勇希, 石浦菜岐佐: "CPU と密に結合したコプロセッサによるハードウェア/ソフトェア協調設計," 情報処理学会研究報告, 2010-ARC-187-16/2010-EMB-15-16, (Jan. 2010).

[5] 神原弘之, 金城良太, 矢野正治, 戸田勇希, 小柳滋: "パイプラインプロセッサを理解するための教材: RUE-CHIP1 プロセッサ," 情報処理 関西支部大会 2009, pp. A-09 (Sept. 2009).