Vol.2010-ARC-187 No.16
Vol.2010-EMB-15 No.16
2010/1/29

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

戸田 勇希[†1] 石浦 菜岐佐[†1]
神原 弘之[†2] 冨山 宏之[†3]

本稿では, CPU と密に結合するコプロセッサの合成に基づくハードウェア/ソフトウェア協調設計手法を提案する. 本手法は, バイナリコードの任意の区間を高位合成によってコプロセッサ化する. CPU とコプロセッサ間の制御の受け渡しは, コプロセッサによるプログラムカウンタの監視/更新により実現する. CPU とコプロセッサ間のデータ授受は, コプロセッサがレジスタファイルおよび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行う. コプロセッサの起動/終了やソフトウェア/ハードウェア間のデータの受け渡しのオーバヘッドが非常に小さい上, バイナリコードの書き換えは一切不要である. この手法に基づいて, AES 暗号化処理の一部を MIPS (R3000) 互換プロセッサに結合可能なコプロセッサとして FPGA 上に合成した結果, CPU 単体と比較して, LUT 数 6.8% の増加で実行サイクル数を 37% 短縮できた.

Hardware/Software Co-Design Based on Coprocessor Tightly Coupled with CPU

Yuki Toda,[†1] Nagisa Ishiura,[†1] Hiroyuki Kanbara[†2] and Hiroyuki Tomiyama[†3]

This article presents a novel hardware/software codesign method based on synthesis of coprocessors tightly coupled with CPUs. Given a software binary, its arbitorary part is synthesized into a coporcessor. The coprocessor watches and writes into the program counter of the CPU to achieve fast control transfer between the CPU and the coprocessor. The coprocessor directly accesses the register file and the forwarding unit of the CPU to exchange data with minimium delay. In a preliminary experiment, a part of the AES encription program is synthesized into a coprocessor, running with MIPS R3000 compatible CPU on FPGA. The coprocessor reduced the total cycles for computation by 37% with 6.8% increase in the LUT count.

1. はじめに

近年, 組み込みシステムの設計期間短縮に対する要求は益々高まっており, ハードウェア/ソフトウェアから成るシステムをいかに効率良く設計するかが重要な課題となっている. 高位合成[1] は, プログラミング言語等による動作記述によってハードウェアを効率的に設計するための技術として研究が行われてきたが, 最近ではハードウェア/ソフトウェアから成るシステムの設計の効率化という文脈での研究が盛んに行われるようになっている.

システム設計の観点では, 合成したハードウェアとソフトウェアのインターフェースが重要な課題となる. すなわち, 制御やデータをソフトウェアとハードウェアの間でいかに受け渡すか, またそれを仕様となる動作記述の中でいかに記述するかが重要になる. これまでに種々の手法が提案されているが[1)–8)] ハードウェアの起動/完了やデータ授受の際にオーバーヘッドが生じる, あるいは, ハードウェア呼出しのために特別な記述や通信設計, またそれに伴うソフトウェアの書き換えが必要となる等の問題がある.

これに対し本稿では, CPU と密に結合するコプロセッサの合成に基づく協調設計手法を提案する. 本手法は, バイナリコードの任意の区間を高位合成によってコプロセッサ化する. CPU とコプロセッサ間の制御の受け渡しは, コプロセッサが 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. また, CPU とコプロセッサ間のデータ授受は, コプロセッサがレジスタファイルおよび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行う. これにより, 従来手法と比べてコプロセッサの起動/終了やソフトウェア/ハードウェア間のデータの受け渡しのオーバヘッドを小さくできる. また, ソフトウェア (バイナリコード) の書き換えを一切行わずにハードウェアを呼び出すことができる.

この手法に基づいて, 我々の研究室で開発した ACAP (Assembly Compatible Architecture Prototyper)[5] をバックエンドとして用い, AES 暗号化処理の一部を MIPS (R3000) 互換プロセッサに結合可能なコプロセッサに合成した. FPGA Xilinx Spartan 3E をターゲットに論理合成を行った結果, CPU 単体で動作させる場合と比較して, 使用 LUT 数 6.8%

†1 関西学院大学
Kwansei Gakuin University
†2 京都高度技術研究所
ASTEM RI/Kyoto
†3 名古屋大学
Nagoya University

Vol.2010-ARC-187 No.16
Vol.2010-EMB-15 No.16
2010/1/29

の増加で, プログラム全体の実行にかかるサイクル数を 37% 短縮することができた.

2. 関 連 研 究

ソフトウェアの専用ハードウェア化による高速化に関しては, これまでに様々な手法が提案されている.

Mittal ら[2] は, DSP のアセンブリからレジスタ転送レベル回路の自動合成を行っている. DSP 上のソフトウェアを FPGA 実装に置き換えることを主な目的にしているが, ハードウェア単体の合成を想定しているため, ソフトウェアとの協調動作は考慮されていない.

System Builder[3] は, ソフトウェア中の指定した機能を高位合成によりハードウェア化するとともに, 必要な通信プリミティブやデバイスドライバの自動合成を行う. ハードウェアはソフトウェアからプロセスとして起動されるため, 比較的大きな機能のハードウェア化に適している.

CCAP[4], および ACAP[5] は, C やアセンブリで記述されたプログラム中の指定された関数をハードウェア化する. ハードウェア化された関数はソフトウェアや他のハードウェア関数から呼び出せるが, 制御の受け渡しにはポーリングを用いており, またメインメモリを通じて関数間のデータ授受を行うため, これらのためのオーバーヘッドが大きい. Stitt[6] らの “Binary Synthesis” は, バイナリコード中から関数より小さな範囲を対象としてハードウェア化を行うが, 制御やデータの受け渡しは CCAP 等と同様の方法を用いているためオーバヘッドが大きく, またハードウェアの呼び出しのためにバイナリコードの更新が必要になる.

Shee ら[7] 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. コプロセッサの開始/終了は特殊命令や特殊レジスタを用いて行う. 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, 特殊命令の追加やコードの書き換えが必要であり, また 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ことがある.

瀬戸ら[8] は, 高位合成を利用した命令セット拡張によるソフトウェアの高速化を提案している. 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, ハードウェアとソフトウェアの制御/データ授受のオーバヘッドは非常に小さい. しかし, ハードウェア化できる範囲は基本ブロックに限られており, また対象領域の大きさは追加可能な専用命令の最大数により限定される.

このように, 高位合成を利用したハードウェア/ソフトウェア協調設計手法においては,
(1) ハードウェアの起動やデータ授受のオーバヘッド,
(2) ハードウェア化できる範囲の制約,
(3) ハードウェア呼び出しに必要となるソフトウェア書き換えや命令追加の複雑さ,
等をいかに低減するかが課題となる.

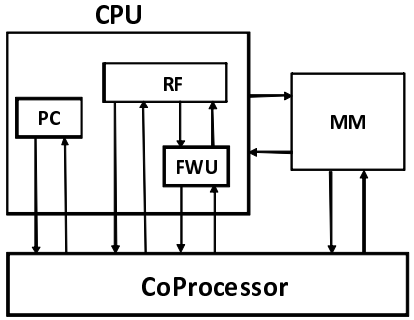

図 1 コプロセッサと CPU の接続

3. プログラムカウンタに基づくコプロセッサ呼出し

3.1 概　　要

本稿では, CPU と密に結合したコプロセッサに基づくハードウェア/ソフトウェア協調設計手法を提案する. 本手法は,
(1) バイナリファイルの任意区間をハードウェア化する,
(2) CPU のプログラムカウンタの監視/書き込みによりハードウェアの起動/終了を制御する,
(3) レジスタファイル, メインメモリ及びフォワーディングユニットへのアクセスにより CPU/コプロセッサ間のデータ授受を行う,
というものである. これにより, コプロセッサ起動/終了のために特別な命令や特殊レジスタの追加が不要であり, CPU とコプロセッサ間のデータ授受を非常に高速に行える. また, ソフトウェア (バイナリコード) の書き換えが全く不要という利点もある.

本手法における コプロセッサと CPU の接続法を図 1 に示す. コプロセッサは, CPU が実行するバイナリ/アセンブリコードのうち, ユーザが指定した任意の区間を ACAP[5] や

Vol.2010-ARC-187 No.16
Vol.2010-EMB-15 No.16
2010/1/29

Binary Synthesis[6] と同様の合成技術でハードウェア化したものである．コプロセッサは CPU のプログラムカウンタ (PC) の値を監視し，プログラムカウンタがハードウェア化した命令列の開始アドレスに到達すると処理を開始する．コプロセッサ動作中は CPU は停止する (NOP 動作を行う)．コプロセッサは処理を完了すると CPU のプログラムカウンタにソフトウェアの再開アドレスを書き込み，CPU が処理を再開する．

CPU/コプロセッサ間のデータ授受は，コプロセッサが CPU のレジスタファイル (RF) 及びメインメモリ (MM) に直接アクセスすることにより行う．また，コプロセッサから CPU のフォワーディングユニット (FWU) にもアクセスすることにより，さらにアクセスに要する時間を短縮する．

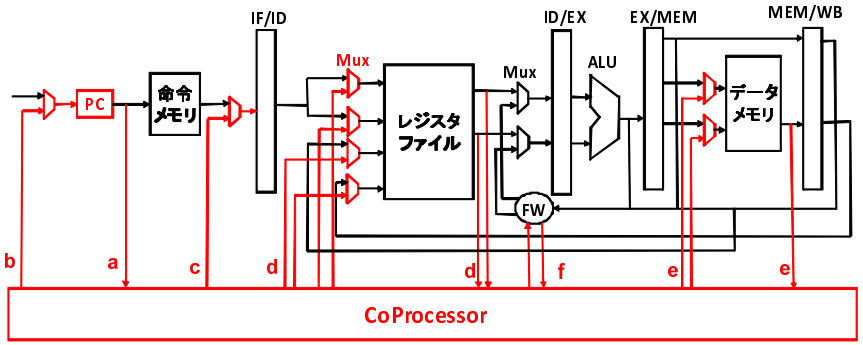

図 2 コプロセッサと MIPS の接続例

3.2 CPU とコプロセッサの接続と動作タイミング

以下では，MIPS (R3000) の単相 5 段パイプライン実装を例に，CPU とコプロセッサの接続と動作タイミングについて述べる．

3.2.1 CPU とコプロセッサの接続

図 2 はコプロセッサと MIPS の接続例である．図中の a によりコプロセッサが CPU のプログラムカウンタを監視する．コプロセッサ起動時は b からコプロセッサを開始した時点のアドレスを書き込み続け，c より CPU に NOP 命令を供給し続ける．処理が終了すれば b よりソフトウェアの再開番地をプログラムカウンタに書き込む．

コプロセッサは d, e により CPU のレジスタファイル，及びメインメモリにそれぞれにアクセスする．本手法では CPU とコプロセッサが同時にレジスタファイル/メインメモリにアクセスすることはないため，アクセスの制御は単純なセレクタにより行える．また，コプロセッサは f により CPU のフォワーディングユニットにもアクセスを行う．これにより，CPU とコプロセッサ間のデータハザードを回避するための待ちを最小化できる．

```
1: addu a1,a1,t2    ;f1
2: addu a3,t1,a3    ;f2
-------------------------
3: addu t1,a3,t1    ;f3
4: subu v1,a1,t1    ;f4
5: addu a2,v1,0x3   ;f5
6: addi a0,$0,0x1   ;f6
7: addu a0,a0,a2    ;f7
-------------------------
8: addu v0,a1,a0    ;f8
9: addu t2,v0,0x5   ;f9
```

} ハードウェア化

図 3 アセンブリプログラムの例

3.2.2 動作タイミング

図 3 に示す MIPS アセンブリの命令列に対し，3 〜 7 行目をコプロセッサとしてハードウェア化する場合を考える．

(1) 従来手法 (メモリマップト I/O を用いた起動/終了)

図 4 は，メモリ空間にマップされたハードウェアのレジスタへのアクセスによってコプロセッサの起動/終了の制御とデータ通信を行う手法の処理の流れを示したものである．

CPU は 2 行目の命令 f_2 の実行後，パラメタとして，a_3, t_1, a_1 の値をコプロセッサに渡す必要があるが，これを f_{a3}, f_{t1}, f_{a1} の 3 命令で行っている．次に命令 f_R の実行によりプロセッサを起動する．f_R の M ステージでコプロセッサの起動レジスタに値が書き込まれてからコプロセッサが実行を開始するため，ソフトウェアの最終命令 (f_2) の実行完了からハードウェアの実行開始までに 3 サイクルの待ちが生じる．

コプロセッサの実行の終了は特殊レジスタへの書き込みにより CPU に通知する (図では省略している) が，CPU はその後にロード命令 f'_{a0} によってコプロセッサの計算結果を読み出す．CPU の実行再開のタイミングは実装により異なるが，f'_{a0} の実行は最も早くても

Vol.2010-ARC-187 No.16
Vol.2010-EMB-15 No.16
2010/1/29

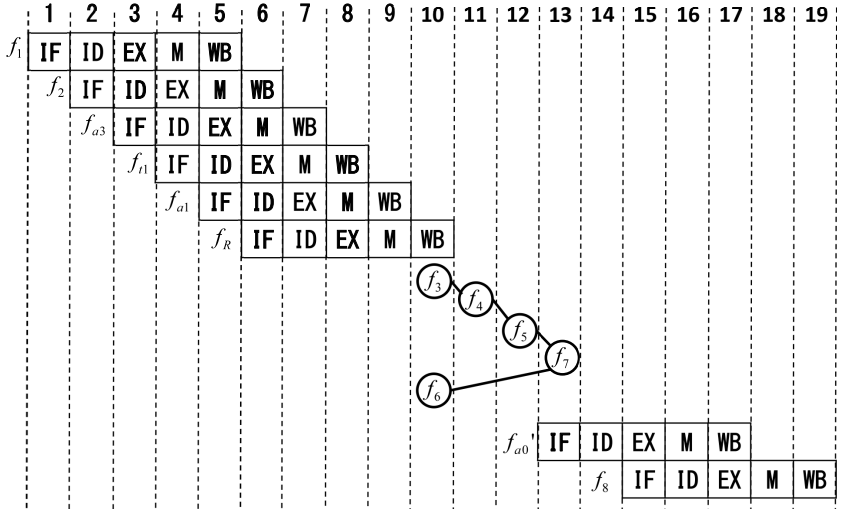

図 4 コプロセッサ起動/終了の流れ: 従来手法

13 サイクル目からになり, f_8 の実行はロード遅延のため 15 サイクル目からとなる. 従って, このプログラムの実行には全体で 19 サイクルが必要になる.

(2) 提案手法 (フォワーディングユニット非接続)

本稿で提案する手法によるコプロセッサの起動/終了のタイミングを図 5 に示す. 本手法ではコプロセッサの起動や CPU からパラメタを渡すための命令は不要である. コプロセッサは最初の命令 (f_3) の IF ステージの次のサイクルから動作を開始することができる. パラメタ a_3, t_1, a_1 の値は, コプロセッサがレジスタファイルに直接アクセスして取得する. ただし, これらのレジスタの読み出しはレジスタ書き込みの可能性がある f_2 の WB ステージ後 (7 サイクル目) まで実行できない.

コプロセッサが実行を完了し, PC に f_8 の番地を書き込むと, CPU はその次のサイクルから f_8 のフェッチを開始する. PC の更新は, コプロセッサと CPU のデータ依存関係が維持されれば, コプロセッサの全ての演算の完了を待たずに行うことができる. CPU がレジスタファイルを最も早く読むのは再開後の ID ステージなので, コプロセッサの最後のレジスタファイル書き込みの 1 サイクル前に PC を更新すればよい. 以上よりアセンブリ全体の実行に必要なサイクル数は 14 サイクルとなる.

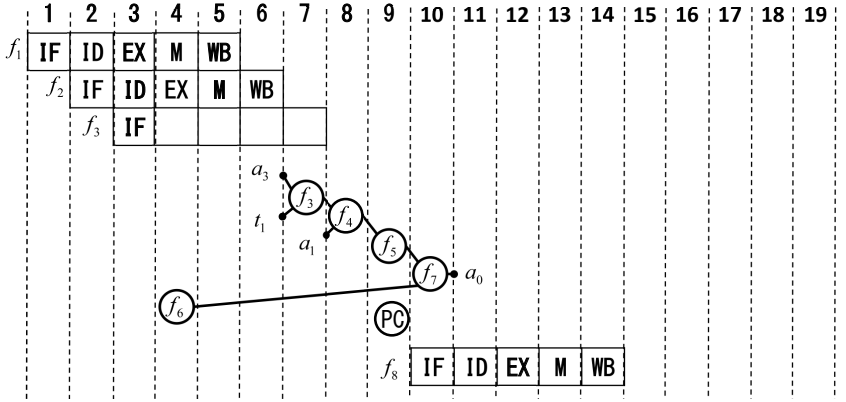

図 5 コプロセッサ起動/終了の流れ: 提案手法 (フォワーディングなし)

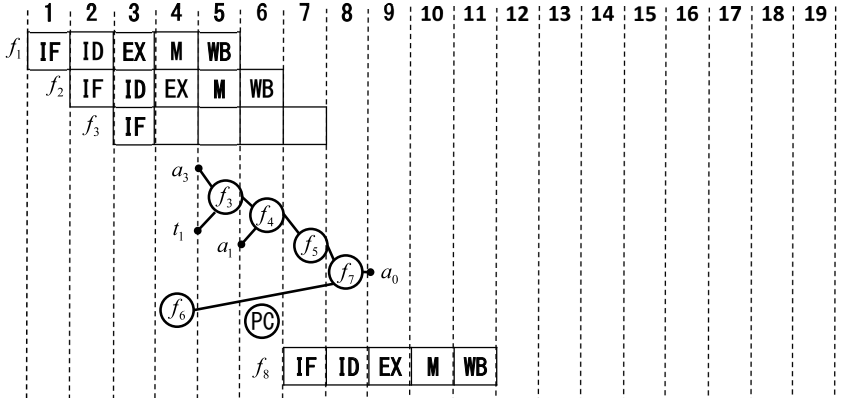

図 6 コプロセッサ起動/終了の流れ: 提案手法 (フォワーディングあり)

(3) 提案手法 (フォワーディングユニット接続)

さらにフォワーディングユニットへの接続を許した場合の処理の流れを図 6 に示す. コプロセッサの f_3 の実行は f_2 の WB ステージの完了を待つ必要がなく (f_2 がロード命令か否かは合成の条件として与える), また CPU も f_8 の EX ステージをコプロセッサの a_0 への書き込み直後に実行できる. そのため, アセンブリ全体の実行が 11 サイクルで行え, 特殊

Vol.2010-ARC-187 No.16
Vol.2010-EMB-15 No.16
2010/1/29

レジスタによる方式と比較して, 全体で 8 サイクルの削減が可能となる.

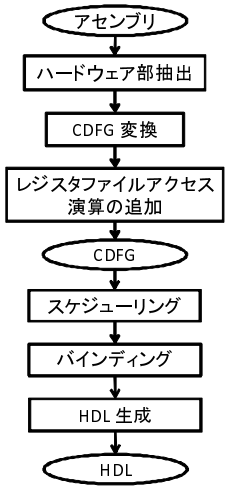

図 7 コプロセッサの合成の流れ

3.3 コプロセッサの合成

図 7 にコプロセッサの合成の流れを示す. 始めに, 対象となるソフトウェアのバイナリコードを逆アセンブルし, 得られたアセンブリコードからハードウェア化したい任意の区間を切り出す. これをコントロールデータフローグラフ (CDFG) に変換し, この CDFG に対しスケジューリング, バインディングを行ってハードウェア記述言語に変換する. 元のプログラムにおける汎用レジスタへのアクセスは, 最初のデータ取得と最終的な値の書き込みを除いて, コプロセッサのローカルレジスタへのアクセスに変換する. 汎用レジスタの読み書きは一種の演算として扱い, 瀬戸ら[8] の手法と同様に仮想的な「レジスタファイルアクセスユニット」を資源として, ポート数と等しい資源制約の下でスケジューリング/バインディングを行う. また, CPU とハードウェアのレジスタファイルへの読み込み/書き込み処理の衝突をさけるための制約を課す. 例えば図 3 の MIPS の例では, ハードウェア開始から 3 ステップ目までは CPU がレジスタファイルに書き込みを行う可能性があるため, コプロセッサからの書き込みを禁止する. 同様に, コプロセッサの動作が終了する前の 2 ステップの間は, CPU が再開後にレジスタファイルを読み出す可能性があるため, コプロセッサからの読み出しを禁止する.

4. 実　　験

本手法に基づき MIPS R3000 互換プロセッサ[9] と接続可能なコプロセッサを合成した. AES 暗号化の処理の一部を MIPS 用 gcc 3.4.6 でコンパイルして得られたバイナリを逆アセンブルし, その一部 (鍵のブロックデータ並び替え部; ループを含む 79 命令) をコプロセッサに合成した. 合成には ACAP[5] を用いたが, レジスタファイルアクセス演算の挿入等一部の処理は手動で行った. CPU 及びコプロセッサは 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 4 に示す. 「CPU 単体」は全ての計算を MIPS だけで行った場合を, 「CPU+HW」はコプロセッサを併用した場合を表す. 遅延はほとんど増えていない. LUT 数 6.8% の増加でサイクル数を 37% 削減することができた.

表 1 実験結果

	LUT 数	遅延 (ns)	サイクル数
CPU 単体	8287 (100.0%)	25.770 (100.0%)	1571 (100.0%)
CPU+HW	8853 (106.8%)	25.841 (100.3%)	982 (62.5%)

5. む　す　び

本稿では CPU と密に結合したコプロセッサに基づくハードウェア/ソフトウェア協調設計手法を提案した. コプロセッサからプログラムカウンタに対する監視/書き込みによる起動/終了の制御, 及びコプロセッサからのレジスタファイル, フォワーディングユニットへのアクセスより, CPU/コプロセッサ間の制御やデータの授受のオーバヘッドを最小化した. 本手法により合成したコプロセッサを MIPS R3000 互換プロセッサと協調動作させた結果, 小規模なハードウェア量の増加で計算時間を短縮する効果があることが確認できた.

今後の課題としてはアセンブリからの合成の完全自動化による設計の効率化や, ループパイプライニングなどによるハードウェアの更なる高速化が挙げられる.

謝　　辞

本研究を進めるにあたり御助言・御討論を頂きました元立命館大学の中谷嵩之氏に感謝します. また, ACAP を開発した元関西学院大学の池上達也氏, 関西学院大学の入谷賢孝氏を

Vol.2010-ARC-187 No.16
Vol.2010-EMB-15 No.16
2010/1/29

はじめ, 石浦研究室の諸氏に感謝します.

参 考 文 献

1) Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin: *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers (1992).
2) Gaurav Mittal, David C. Zaretsky, Xiaoyong Tang, and P. Banerjee: "Automatic translation of software binaries onto FPGAs," in Proc.41st DAC, pp.389–394 (June 2004).
3) 本田晋也, 冨山宏之, 高田広章, "システムレベル設計環境：SystemBuilder," 信学論 D-I, vol.J88-D-I, no.2, pp.163–174 (Feb.2005).
4) M.Nishimura, K.Nishiguchi, N.Ishiura, H.Kanbara, H.Tomiyama, Y.Takatsukasa, and M.Kotani: "High-level synthesis of variable accesses and function calls in software compatible hardware synthesizer CCAP," in Proc.the 13th SASIMI, pp.29–34 (Apr.2006).
5) 池上達也: "MIPS アセンブリを中間表現とする高位合成," 情処 関西支部大会 2008, pp.11–12 (Dec.2008).
6) Greg Stitt and Frank Vahid "Binary Synthesis," ACM Trans.on Design Automation of Electronic Systems, vol.12, no.3, article 34 (Aug.2007).
7) Seng Lin Shee, Sri Parameswaran, and Newton Cheung: "Novel architecture for loop acceleration: a case study," in Proc.CODES+ISSS '05, pp.297–302 (Sept.2005).
8) 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," 情処 DA シンポジウム 2006, pp.49–54 (July 2002).
9) 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," 情処 関西支部大会 2009, pp.A-09 (Sept.2009).