システムＬＳＩ設計技術　103－20
（2001. 11. 30）

非同期式プロセッサにおける演算器構成の検討

柴田裕一郎† 小栗 清†

† 長崎大学工学部情報システム工学科
〒852–8521 長崎市文教町 1-14

E-mail: †{shibata,oguri}@cis.nagasaki-u.ac.jp

あらまし 効果的な非同期プロセッサの設計には，遅延仮定の粒度の決定が重要な鍵となる．細粒度の遅延仮定のもとでは，入力されるデータに応じてレイテンシの変化する演算器を構成することができる．逆に，同期回路のようにすべてのオペランドに対して最大遅延を仮定することにより，完了信号生成などのオーバヘッドを最小化することもできる．そこで本稿では，このような設計のトレードオフの判断に必要な根拠を示すために，さまざまな構成の演算器の遅延分布をベンチマークアプリケーションのトレースデータを用いたシミュレーションにより評価する．評価の結果，高い性能の演算器においては，演算器のレイテンシを可変にすることによるオーバヘッドは20%を超えてはならないことが明らかになった．

キーワード 非同期回路，演算器，加算器，非同期式プロセッサ

On the Structure of an Arithmetic Unit for Asynchronous Processors

Yuichiro SHIBATA† and Kiyoshi OGURI†

† Department of Computer and Information Sciences,
Faculty of Engineering, Nagasaki University
1-14 Bunkyo-machi, Nagasaki 852–8521 Japan

E-mail: †{shibata,oguri}@cis.nagasaki-u.ac.jp

Abstract One of the most important keys to efficient design of asynchronous processors is granularity of a delay assumption. With a fine grain delay assumption, an arithmetic unit whose latency can be varied depending on individual input data is possible. On the contrary, assuming the latency of the critical path for all operands like in synchronous circuits, overhead such as completion signal generation can be minimized. Aiming to give the basis for these design choices, this paper evaluates a latency distribution of various types of arithmetic units by simulating the circuits using trace data of typical benchmark applications. The results show that time overhead of a variable latency property should not exceed 20% for an efficient arithmetic unit.

Key words asynchronous circuit, arithmetic unit, adder, asynchronous processor

1. はじめに

半導体プロセス技術の目覚ましい発展によって，大規模なチップ上にますます微細化されたゲートを集積することが可能となっている．しかし，このような状況のもとではゲート遅延に比較して配線遅延が支配的となるため，大域的にクロック信号を分配する必要のある同期回路システムでは，このクロックラインが性能向上を制限していくと考えられる．

このように限界が明らかになりつつある同期回路に代って，共通クロック信号を使わずに，局所的な信号遷移を用いて分散的にレジスタ間の同期制御を行う非同期回路システムが注目を集めている [1] [2]．既にマンチェスタ大学の AMULET3 [3]，東京大学および東京工業大学の TITAC-2 [4]，カリフォルニア工科大学の miniMIPS [5] など，非同期式回路に基づくマイクロプロセッサも実装されており，その実現可能性と効果が示されている．

同期式回路の動作速度はレジスタ間の最大遅延（クリティカルパス遅延）によって決まる．このため，クリティカルパスを通らないデータが処理される場合には組合わせ回路部の計算が早く終了するが，この結果がレジスタに書き込まれるのは最大遅延時間後であり性能は処理するデータに依存しない．これに対して非同期式の回路では，各レジスタ間でデータ転送のタイミング制御を行うことによって，組合わせ回路部の計算が終了した時点で結果をレジスタに書き込むようにすることができる．したがって，動作性能がレジスタ間の平均の遅延によって決まるようになり，同期回路よりも高速な動作が期待される．

しかし，このようなタイミングの制御を行なうためには，組合せ回路部の処理完了を通知する信号が必要となる．たとえば，データに応じて柔軟に処理時間を制御するためには，2線2相式のシグナリングプロトコル [6] の採用などが必要になりそのオーバヘッドも無視できない．逆に同期回路と同様に，すべてのデータに対して組合せ回路の最大遅延を仮定すれば，単純な遅延線のみで完了信号を生成することができる．非同期回路では遅延仮定の粒度によって他にもさまざまな設計が可能であり，レイテンシをデータによって変化させることによる性能向上と，完了信号生成のオーバヘッドによる性能低下との間のトレードオフを考慮して決定する必要がある．

そこで本稿では，非同期回路の特性を生かしたプロセッサアーキテクチャ設計の第一歩として，まず演算命令の大部分を占める整数加減算命令に着目し，

表1　評価対象を行った加算器

記号	構成
RC	リプル・キャリー
PORC	部分最適化リプル・キャリー
PCL	疑似キャリー・ルックアヘッド
PCL2	16 ビット疑似キャリー・ルックアヘッド × 2
PCL4	8 ビット疑似キャリー・ルックアヘッド × 4
PCL8	4 ビット疑似キャリー・ルックアヘッド × 8
CLT	キャリー・ルックアヘッド・ツリー
Skip	キャリー・スキップ
Select	キャリー・セレクト

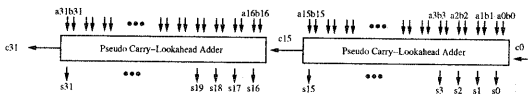

図1　PCL のカスケード接続型加算器

アプリケーション走行時のオペランドデータが加算器通過に要する遅延時間の分布をシミュレーションにより明らかにする．また，これらの遅延分布が加算器の構成によってどのように変化するかを検討する．

2. 評価対象の演算器

今回評価の対象とした加算器は表1に示す9種である．それぞれ2の補数表現された32ビットデータを扱うものとし，減算は減数のビット反転と最下位ビットへのキャリー入力を用いて行うこととした．以下に各加算器の構成について述べる．

RC（Ripple-Carry）：32個の全加算器を単純にカスケード接続した一般的なリプル・キャリー型の加算器．

PORC（Partially Optimized Ripple-Carry）：RC 型の加算器に論理合成系を用いて遅延最適化処理を施したもので，構成の一部がキャリー先見論理に置換えられている．

PCL（Pseudo Carry-Lookahead）：加算器全体を組合わせ回路として，論理合成および遅延時間の最適化を行った疑似キャリー・ルックアヘッド型加算器．得られる回路はキャリー・ルックアヘッド型に近づくことが知られているが，最適化にはヒューリスティックアルゴリズムが用いられるため，不規則性の高い構成となる．

PCL2（Pseudo Carry-Lookahead × 2）：図1に示すように，16 ビットの PCL 型加算器を2個カスケード接続したもの．

PCL4（Pseudo Carry-Lookahead × 4）：8ビットの PCL 型加算器を4個カスケード接続したもの．

PCL8（Pseudo Carry-Lookahead × 8）：4ビットの PCL 型加算器を8個カスケード接続したもの．

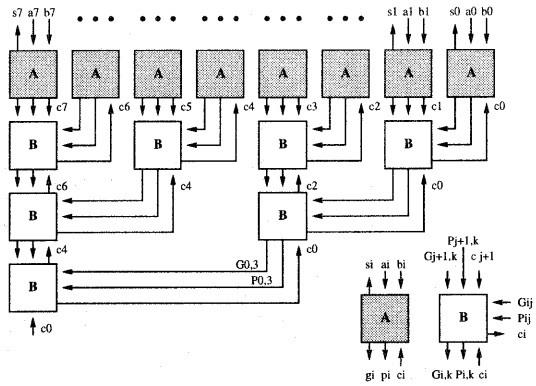

図 2　8 ビット CLT 型加算器

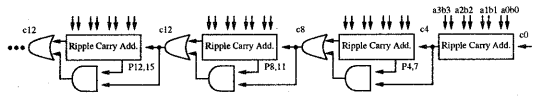

図 3　キャリー・スキップ型加算器

CLT（Carry-Lookahead Tree）：2 進木状に規則的に構成されたブロックを用いて約 $\log_2 n$ 段（n は加算のビット幅）の論理で段階的にキャリー先見の計算をするものであり，たとえば $n = 8$ のときには図 2 に示す構成を持つ [7].

図 2 において，上部へ入力される値が下向きに流れながら，第 i ビットから第 k ビットまでの区間でキャリーが生成されることを表す信号（G_{ik}）と，同区間に入力されるキャリーが上位に伝搬することを表す信号（P_{ik}）の計算が行われる．その後，図の下部から入力される最下位ビットへのキャリー入力（$c0$）と合流し，上向きに遡りながらキャリーが求められる．

Skip（Carry-Skip）：図 3 に示すように，複数の RC 型加算器をカスケード接続した構成を持つキャリー・スキップ型加算器である [7]．各 RC 型加算器ではその区間のキャリー伝搬信号（P_{ik}）を計算する論理も持つ．

初期状態では，これらの加算器へのキャリー入力はクリアされているものとする．計算の開始と同時に各加算器内でキャリーの伝搬が開始される．また，同時に P_{ik} 信号の計算も行われ，もしこの信号が 1 になった加算器へ下位からのキャリー入力があると，併設されたバイパス回路によって上位へキャリーのスキップが行われるので単純な RC 型よりも高速である．ただし，プリチャージなどによって各キャリー入力を毎回クリアするための動作が必要になる．また，何ビットごとにひとつの加算器にブロック化するかによっても性能が左右される．今回の評価には，

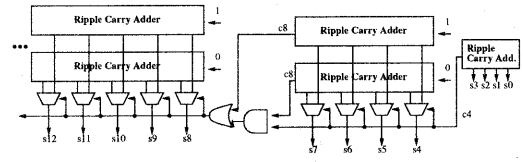

図 4　キャリー・セレクト型加算器

表 2　評価に用いたベンチマーク

プログラム	入力パラメータ
gcc	cccp.i
compress	100 q 2131
go	40 19
ijpeg	specmun.ppm
li	test.lsp

下位側から 2，8，7，6，5，4 ビットずつブロック化した構成のものを用いた．

Select（Carry-Select）：基本的には複数の RC ブロックをカスケード接続した構成を持つが，図 4 に示すように，上位にはそれぞれキャリー入力が 0 の場合と 1 の場合の 2 組ずつを用意しておき，下位側のキャリー出力が決定された時点でそれらの選択を行う仕組みを持つ [7].

この加算器の場合も，ブロック化のビット数が性能に影響を与える．今回の評価では下位側から 4，4，5，6，7，6 ビットずつにブロック化を行った．この構成は多くの場合に良好な性能を示すことが知られている [8].

3.　評価の手順

まず，並列計算機シミュレータ用ライブラリ ISIS [9] で構成された MIPS R3000 [10] のクロックレベルシミュレータ上でベンチマークを実行し，整数加算命令（add，addu，addiu）および整数減算命令（sub，subu）のオペランドのトレースを生成させた．用いたベンチマークは SPEC CINT95 から表 2 に示した 5 種である．ベンチマークのコンパイルには，GNU の MIPS 用コンパイラ（gcc-2.95.3）とアセンブラ（binutils-2.9.1）を用いた．なお，各ベンチマークのトレース対象命令のうち，始めの 100 万命令をそれぞれ評価の対象とした．

次に，前節で述べた各加算器を論理合成して verilog のネットリストを生成し，オペランドトレースを入力としたゲートレベルシミュレーションを行い遅延分布を評価した．評価用の半導体プロセスには VDEC（東京大学大規模集積システム設計教育センター）によって提供された 0.35μm の CMOS スタンダードセルを用い，セルライブラリには EXD 社のセ

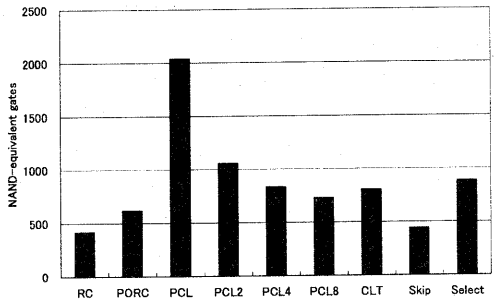

図 5　各演算器のゲート数

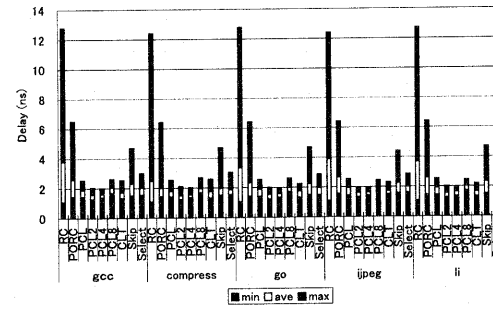

図 6　各演算器の遅延時間

ルジェネレータで生成されたものを使用した．ゲート遅延時間の評価にはセルライブラリの遅延モデルを用い，配線遅延については考慮しなかった．論理合成にはSynopsys社のDesign Compiler（2000.11）を用いたが，所望の構成の演算器を得るために適宜手作業によるネットリストの作成も行った．

4. 評価結果

4.1 ハードウェア量

図5に3節で述べた各演算器の合成結果を示す．図の縦軸は2入力NANDで換算したゲート数である．もっとも小規模のRCが約400ゲートで構成されたのに対し，もっとも大規模になったPCLではほぼ5倍の約2,000ゲートを要した．一方，同じキャリー・ルックアヘッド系のCLTは，その単純で規則的な構成のためRCの約2倍と比較的小規模に構成された．Selectはキャリー入力に応じた2組の加算器を持ち，さらにそれらの結果を選択するマルチプレクサ群も必要なためにRCの2倍を上回る大きさとなった．

論理合成系によって組合わせ回路としての遅延最適化を行ったPCL，PCL2，PCL4，PCL8を比較すると，PCLに比べPCL2が0.52倍，PCL4が0.41倍，PCL8が0.36倍となり，分割数が増えるにしたがって面積は小さくなった．また，PORCは合成系による最適化処理の結果，処理前のRCに比べて面積が約48%増大した．

4.2 最大遅延時間

図6に各演算器の遅延シミュレーションの結果をまとめる．縦軸には，各演算器のアプリケーションごとの最小遅延を「min」，平均遅延を「ave」，最大遅延を「max」として示している．まず最大遅延に着目すると，すべてのアプリケーションにおいてRCの遅延がもっとも大きく，ついでPORC，Skipとリプル・キャリーをベースとした構成のものがそれに続いた．反対にもっとも最大遅延が小さかったものはPCL4で，これもすべてのアプリケーションについて同じ結果となった．アプリケーション全体で見ると，RCの最大遅延は12.8 ns，PCL4では2.0 nsとなり両者の差は6.4倍と大きく開いた．

RCに論理最適化処理を加えたPORCの最大遅延は約6.5 nsとなり，RCに比べて約2倍の改善が示された．図5より，RCに対するPORCの面積増加が48%であることを考慮すると，この最適化処理の効果は高いといえるが，PCL4に比較するとなお3.3倍の値を示している．一方，PCL，PLC2の最大遅延はPCL4に比べてそれぞれ約26%と4.5%大きくなった．図5によればPCL，PCL2ともにPCL4に比べて面積も大きい．このことは，PCL4に使用されている8ビットの加算器が合成系による遅延最適化処理にもっとも適した粒度であり，それ以上のビット数の加算器を組合わせ回路として最適化すると品質が劣化することを示している．また，4ビットのPCLを8段カスケード接続したPCL8は，PCL4に比べて最大遅延が33%増大しており，分割によ最適化効果向上のメリットがカスケードによるクリティカルパス増大のデメリットに打ち消されたことが分かる．

CLTは面積ではPCL4に比べてわずかに（3.2%）小さかったが，最大遅延はPCL4よりも29%増大した．Skipは面積ではPCL4よりも47%小さかったのに対し，最大遅延は2.3倍に延びた．なお，今回の評価ではSkipに必要なキャリー線の初期化に必要な時間を考慮していないので，実際の遅延はさらに大きくなると考えられる．SelectはPCL4に対して面積が6.5%大きかったうえ最大遅延も50%長くなり，同期システムとして考えた場合のSelectの効果は，今回の評価条件では認められなかった．

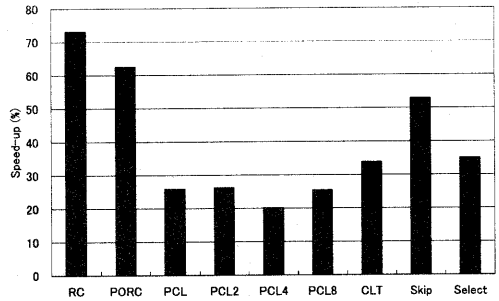

図7　最大遅延に対する平均遅延の速度向上率

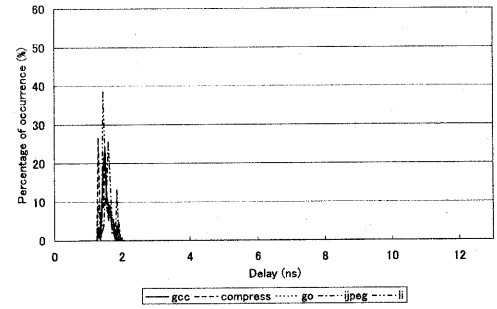

図8　遅延分布（PCL2）

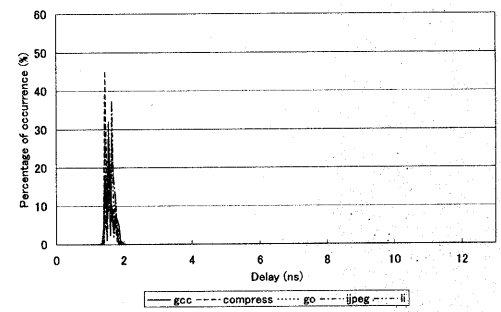

図9　遅延分布（PCL4）

4.3 平均遅延

次に図6に示した平均遅延（ave）に着目する．全体的な傾向としては，特にリプル・キャリー系の演算器においてアプリケーションによる影響が大きいこと，最大遅延に比べて演算器間の遅延差が縮まったことが分かる．また興味深いことに，アプリケーション全体で見た最大遅延ではPCL4がもっとも良好な結果を示したのに対し，平均遅延ではPCL2がPCL4を約3.1%上回り，今回の演算器の中でもっとも高い性能を示した．この様な最大遅延との性能順位の逆転は，PCLとCLT，PCL8とCLTの間でも見られたが，RCがもっとも大きな遅延を持ち，PORCがそれに続くという傾向は最大遅延と変らなかった．

図7は最大遅延に比べて平均遅延が何%向上したかを演算器ごとに示したものである．この値は非同期制御を導入し演算器の遅延を可変にすることによって達成可能な速度向上率の理論的上限と考えることができる．図7からも明らかなように，RC，PORC，Skipといったリプル・キャリー系の演算器では50%から70%の高い速度向上率を示している．一方で，もともと最大遅延が比較的小さいPCL系の演算器は20%台の向上率にとどまっている．このことは，基本性能の高い演算器を使用する場合には，完了信号の生成の時間的オーバヘッドを20%以内におさめないと，少なくとも処理時間の観点からはレイテンシを可変にする効果が得られないことを示している．

一方，速度向上の他にも非同期回路の導入により演算器をレイテンシ可変にするメリットを考えることができる．たとえば，2種類の演算器AとBがあり，Aに比べてBのハードウェア量が十分に小さく，Aの最大遅延よりもBの平均遅延がわずかでも小さければ，演算器Bを採用することにより，同期回路システムに比べて性能を維持しながらハードウェア量を削減できる可能性が生じる．そこで，再び図6に着目し，異なる演算器間での最大遅延と平均遅延を比較してみる．

最大遅延が2.0 nsともっとも短かかったPCL4よりも短い平均遅延を持つPCL4以外の演算器は，PCL（1.9 ns），PCL2（1.5 ns），PCL8（2.0 ns），CLT（1.7 ns），Select（2.0 ns）の5種である．図5によれば，このうちPCL4よりもハードウェア量が小さいのはPCL8，CLTの2種である．まず，PCL8について考えると，面積でのPCL4への優位性は約13%あるのに対し，平均遅延時間はPCL4の最大遅延とほとんど等しい．したがって，同期型のPCL4に対して，非同期型のPCL8のメリットを出すためには，完了信号生成のための時間的なオーバヘッドはほとんど許されないうえ，面積の増加も厳しく制限される．また，一方のCLTでは面積の優位性はPCL4に比べて約3%程度しかない．これらのことから，今回の条件では演算器のレイテンシを可変にすることによる面積削減の効果は望めないことが分かる．

4.4 遅延時間の分布

最後に，主な演算器（PCL2，PCL4，RC，Skip）の遅延分布を図8から図11に示す．これらの図は，横軸に遅延時間，縦軸に各アプリケーションごとでの頻度の割合を示したものである．RCでは遅延分

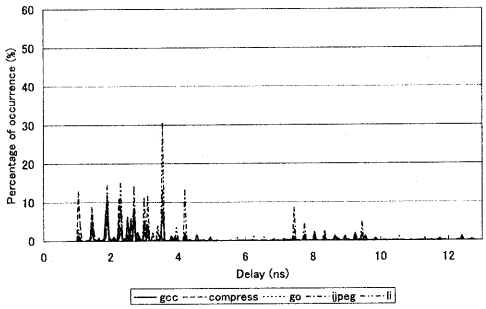

図 10　遅延分布（RC）

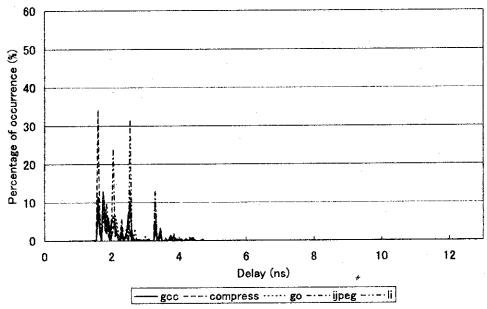

図 11　遅延分布（Skip）

布にかなり広がりが認められるのに対し，PCL2 や PCL4 では遅延の分布が密であることが分かる．また，PCL2 は PCL4 と最大遅延はほぼ同じであるが，PCL4 よりも分布がゆるやかに低遅延側に広がっている．これは，PCL4 ではひとつの演算器ブロック内での遅延と，カスケード接続による遅延がうまくバランスしていたのに対して，PCL2 では下位ブロックの桁上げの有無が遅延に与える影響の大きい構造であったことを示している．この構造の性質がうまく平均遅延の短縮に寄与したと考えられる．

RC は遅延分布そのものは広いが，多くの分布は最大遅延の約 30%より短遅延側に存在していることが分かる．これは，最大遅延に対する平均遅延の短縮率が 73%という図 7 の結果を裏付けるが，PCL2 や PCL4 が約 2 ns 以内の分布であるのに対し，RC では最大の分布点が 3.5 ns 付近にあり性能の差は依然大きい．Skip も RC に比べると効果的に遅延の分布を短遅延側に移動させることに成功しているが，構成要素がリプル・キャリー型であるため，PCL2 や PCL4 に比べると遅延分布は広範囲なままである．また，分布が大きく 2 つの範囲に分割されており，キャリーのスキップの効果がオペランドに対してやや限定的に働いていたことを示唆している．

また，各演算器とも遅延分布の中にいくつかの急峻な山を持っているが，この位置は同じ演算器でもアプリケーションによって異なることが分かる．このことは，遅延分布の検討に複数のアプリケーションを用いることの重要性を示している．

5. おわりに

本稿では非同期回路の特性を生かしたプロセッサアーキテクチャ設計の第一歩として，整数加減算ユニットの遅延分布を実アプリケーションのトレースデータに基づいて評価し，その結果を定量的に議論した．今後はレイアウトを考慮した配線遅延の影響や消費電力とのトレードオフも考慮しながら，より多角的に非同期データパスの構成を検討していくとともに，パイプラインや制御機構の構成が実行性能に与える影響についても評価していく予定である．

謝辞　本研究の遂行にあたって多くのコメントをいただいたソニー（株）S&S アーキテクチャセンター若林正樹氏に感謝します．

文　献

[1] 南谷崇: 非同期式マイクロプロセッサの動向, 情報処理, Vol. 39, No. 3, pp. 181–186 (1998).
[2] 小栗 清: 非同期回路設計, 情処研告, Vol. 2001, No. 42, pp. 51–58 (2001).
[3] Furber, S. B., Edwards, D. and Garside, J. D.: AMULET3: a 100 MIPS Asynchronous Embedded Processor, *Proc. International Conference on Computer Design*, pp. 329–334 (2000).
[4] Nanya, T., Takamura, A., Kuwako, M., Imai, M., Fujii, T., Ozawa, M., Fukasaku, I., Ueno, Y., Okamoto, F., Fujimoto, H., Fujita, O., Yamashina, M. and Fukuma, M.: TITAC-2: A 32-bit Scalable-Delay-Insensitive Microprocessor, *Proc. HOT CHIPS IX*, pp. 19–32 (1997).
[5] Martin, A., Lines, A., Manohar, R., M. Nyström, Penzes, P., Southworth, R., Cummings, U. and Lee, T.: The Design of an Asynchronous MIPS R3000 Microprocessor, *Proc. Conference on Advanced Research in VLSI*, pp. 164–181 (1997).
[6] Armstrong, D. B., Friedman, A. D. and Menon, P. R.: Design of Asynchronous Circuits Assuming Unbounded Gate Delays, *IEEE Trans. Computers*, Vol. C-18, No. 12, pp. 1110–1120 (1969).
[7] Goldberg, D.: Computer Arithmetic, *Computer Architecture: A Quantitative Approach* (Hennessy, J. and Patterson, D.), Morgan Kaufmann, 2nd edition, chapter A (1996).
[8] Weste, N. H. E. and Eshraghian, K.: *Principles of CMOS VLSI Design: A System Perspective*, Addison-Wesley (1985).
[9] 若林正樹, 天野英晴: 並列計算機シミュレータの構築支援環境, 信学会論文誌, Vol. J84-D-I, No. 3, pp. 247–256 (2001).
[10] Kane, G. and Heinrich, J.: *MIPS RISC Architecture*, Prentice Hall, 2nd edition (1992).