社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－ＳＬＤＭ－120（３）
2005／5／19

加算器の消費電力/面積/速度の形式による比較

味元 伸太郎† 水口 貴之† 橘 昌良‡

†‡高知工科大学大学院工学研究科電子・光システム工学コース
〒782-8502 高知県香美郡土佐山田町宮の口 185

E-mail: †{095319w, 095318n}@gs.kochi-tech.ac.jp, ‡tachibana.masayoshi@kochi-tech.ac.jp

あらまし 本論文では、加算器の形式別の消費電力・面積・速度の比較結果について報告する。今回報告する加算器の形式は、Ripple carry adder, Carry look-ahead adder, Carry select adder, Carry skip adder, Carry save adder, Hybrid adder の 6 形式である。形式別に加算器を設計し、速度、面積、消費電力についてシミュレーションを行うことにより、システムを構築する際に最も適した演算形式を選択するための参考データを得る。

キーワード 逐次桁上げ、桁上げ先見、RCA、CLA、CSlA、CSkA、CSvA、HBA

Comparison of power consumption, area and speed by form of adders

Shintaro MIMOTO† Takayuki MINAKUCHI† Masayoshi TACHIBANA‡

†‡Electronic and Photonic Systems Engineering Cource, Kochi University of Technology
185 Miyanokuchi, Tosayamada-cho, Kochi 782-8502 Japan

Abstract This paper report comparison result of area, power comsumption and speed form of adders. We report six form of adders Ripple carry adder, Carry look-ahead adder, Carry select adder, Carry skip adder, Carry save adder and Hybrid adder in this paper. We design those six form of adders, then simulate area, power consumption and speed. Purpose this report to collect reference data for system level optimization of LSI.

Keyword ripple carry, carry look ahead, RCA, CLA, CSlA, CSkA, CSvA, HBA

1. はじめに

加算器と一口に言っても様々なアルゴリズムが考案されており、それぞれに長所と短所が存在する。その長所と短所の多くはトレードオフの関係となっている。

回路の性能を示す大きな指針として、面積制約、速度制約、消費電力制約の 3 点が挙げられるが、いずれの制約においても、それぞれの制約が他の制約に密接に関わっている。例えば、速度制約を重視して設計した回路は面積、消費電力ともに増加するという具合である。

全ての制約を高レベルでクリアすることは難しいため、構成するシステム別に求められる性能を把握し、重視する制約を決定する必要がある。つまり、システムの最適化が重要となってくる。

今回の研究では、加算器の基本的な演算形式を取り上げ、その性能を比較した。加算器については多くの参考書で触れられているが、その内容は多くのものが演算速度についてのものであり、面積、消費電力について記述されたものは非常に少ない。現在はそれぞれの演算形式の性能を比較したデータはほとんど存在していないため、一定の基準の下に加算器を設計することで、加算形式そのものの性能を比較したい。研究結果をまとめ、最適化の際の参考データを作成することが今回の研究の目的である。

2. 各加算形式の特徴

今回の研究では、

- Ripple carry adder (RCA)
- Carry look-ahead adder (CLA)
- Carry select adder (CSlA)
- Carry skip adder (CSkA)
- Carry save adder (CSvA)
- Hybrid adder (HBA)

の 6 種の加算器ついてとり上げる。RCA は最も回路構成が単純な、加算器の最も基本的な形である。CLA、CSlA、CSkA は、桁上げを先見する方法の中でも代表的なものである。CSvA はマルチオペランド演算の最も基本的な形式である。既存の加算形式を 2 つ以上組み合わせた加算器を HBA といい、その組み合わせは無数に存在する。今回は演算速度を重視した HBA を設計し、既存の演算形式のシミュレーション結果と比較する。

2.1 Ripple carry adder

Ripple carry adder(以下 RCA)は、Full adder(以下 FA)を演算したいオペランドのビット数だけ接続し、下位の桁上げ出力を上位の桁上げ入力とすることで構成する。図 2.1 に、n ビット RCA のブロック図を示す。

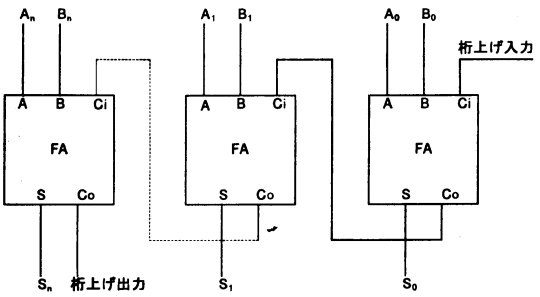

図 2.1 n ビット RCA ブロック図

回路構成が最もシンプルで回路素子数を抑えることができ、面積が小さい。しかし、加数，被加数の入力は全ビット同時に加算器に入力されるのに対し、桁上げ入力は下位からの桁上げ出力となるため、最終的な結果を得る為には桁上げが全ての FA を通過する必要がある。

2.2 Carry look-ahead adder

桁上げを高速化するために最も一般的に用いられる形式が、Carry look-ahead adder(以下 CLA)である。各ビットの和計算とは別に、桁上げを計算する専用の回路を設けることで桁上げの高速化を図る。桁上げ専用の回路を構成することにより面積は大きくなる。

和計算用の Half adder（以下 HA）と一つの桁上げ発生回路で構成しようとすると、ビット数が多くなればなるほどゲート数は膨大になり、ファンインが大きくなる。そこで速度の低下を抑えるために、グループ分割を行う。一般に 4 ビットグループに分割される事が多いので今回の回路構成は 4 ビットグループとし、ツリー構造としている。図 2.2 に、16 ビット CLA のブロック図を示す。

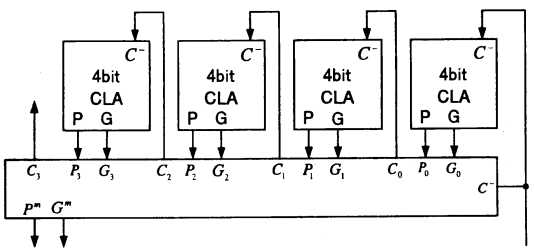

図 2.2 16 ビット CLA ブロック図

2.3 Carry select adder

Carry select adder(以下 CSlA)は、桁上げ入力が'0'と'1'の場合の 2 通りの和を演算する回路を持ち、桁上げ入力を得てどちらかを選択する。図 2.3 に CSlA のブロック図を示す。

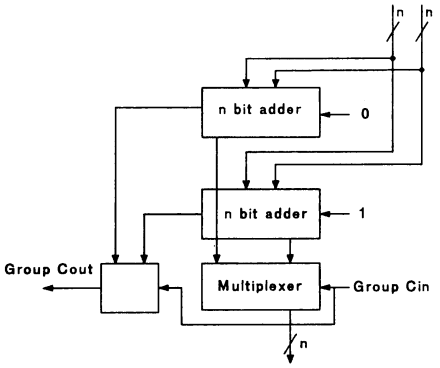

図 2.3 CSlA ブロック図

図 2.3 のように n ビットの入力は桁上げ'0'の n bit adder と桁上げ'1'の n bit adder に同時に入力され、Group Cin を受け取ってから Multiplexer で和を選択する。n bit adder からの桁上げも、Group Cin によって選択される。CSlA は高速化が期待できる半面、回路規模が大きくなるという欠点を持つ。

2.4 Carry skip adder

Carry skip adder(以下 CSkA)は、加算器をグループに分割し、桁上げが発生しないグループを検出することで演算を高速化する。図 2.4 に、12 ビット CSkA のブロック図を示す。

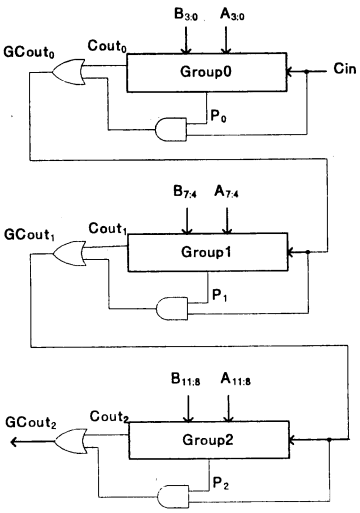

図 2.4 12 ビット CSkA ブロック図

回路構成は、RCA にグループ桁上げを検出する回路を付加したものとなる。

2.5 Carry save adder

マルチオペランド演算を行う際に一般的に用いられるのが Carry save adder(以下 CSvA)である。CSvA は、入力 A,B,C を受け取り S,Co を出力する 3-2 カウンタを基本要素とする。これはつまり、Full adder と同様の動作である。図 2.5.1 に 3 オペランド CSvA、図 2.5.2 に 4 オペランド CSvA のブロック図を示す。

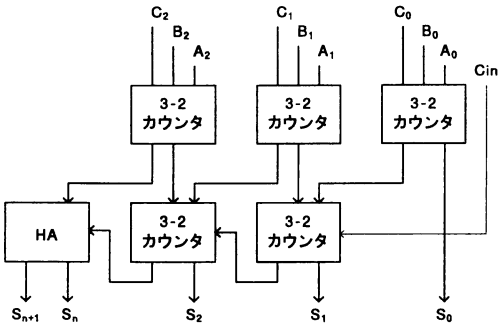

図 2.5.1 3 オペランド CSvA ブロック図

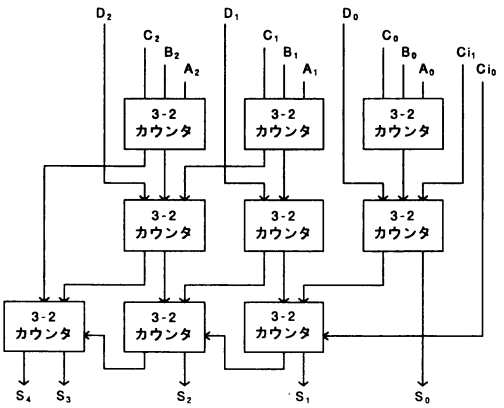

図 2.5.2 4 オペランド CSvA ブロック図

前に挙げた方式を用いてマルチオペランド演算を行う場合、2 オペランドの和の結果が算出されるのを待ち、さらに次のオペランドとの加算を行うというように非常に演算に時間が掛かる。そこで、中間和と桁上げという考え方を用いて、FA を並列に並べた構造が CSvA である。これにより、中間和に次のオペランドを加算することで、演算を最後まで待つ必要がなくなり、演算時間を短縮することが可能である。また、全てのオペランドが加算された後は RCA により桁上げを伝播させれば最終的な和が得られる。演算時間は RCA と 3-2 カウンタ数段分を合わせたものとなる。

2.6 Hybrid adder

Hybrid adder(以下 HBA)は、先に述べたような加算形式を組み合わせたものである。今回の研究では、和計算に CSlA の簡易版を、桁上げ演算にはマンチェスター加算器の簡易版を用いる。図 2.6 に、32 ビット HBA のブロック図を示す。

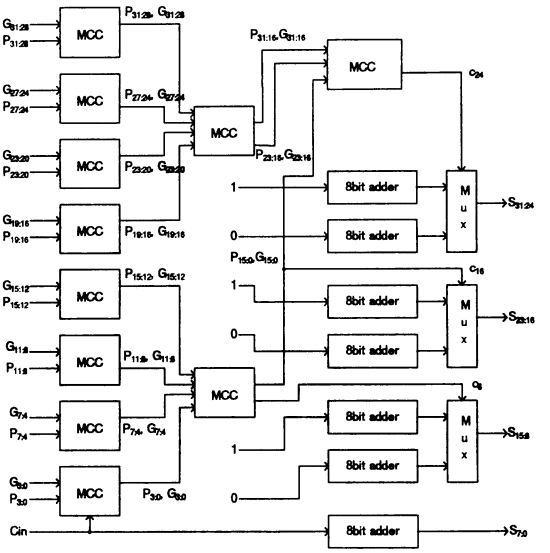

図 2.6 32 ビット HBA ブロック図

3. 演算器の論理合成、シミュレーション手法

2 章で挙げた 6 種の演算器について、VDEC(大規模集積回路システム設計教育研究センター)より提供されている CAD を用いて行う。

ハードウェア記述言語 VHDL を用い、それぞれの加算形式について、8 ビットを最小、64 ビットを最大として設計する。設計した回路の論理合成には Synopsys 社提供の Design Compiler を用いる。ライブラリは、京都大学版の Rohm0.35um ライブラリを使用する。

論理合成後、面積、演算速度の予測値を得る。論理合成の際には、トランジスタレベルでの最適化は行わず、ライブラリのパラメータをそのまま採用する。今回の研究目的は加算形式別の性能比較であるが、トランジスタレベルでの最適化まで行うと、測定結果が演算形式の特長によるものかトランジスタの最適化の結果によるものかの判別が難しくなるためである。なお、今回の研究では、回路の配線容量については考慮していない。

演算速度のシミュレーションについては、最も演算時間を要するパス、すなわちクリティカルパスを比較することとした。

消費電力のシミュレーションには、同じく Synopsys 社提供の Nanosim を用いる。ライブラリは論理合成時と同様に、京都大学版 Rohm0.35um ライブラリを用いる。先ほど合成した演算器に、10[ns]毎にランダムな入力パターンを与え、その演算に消費する電力を得る。入力パターンは、C 言語の疑似乱数発生関数を用いて作成する。

4. シミュレーション結果

4.1 論理合成後の面積

図 4.1 に、論理合成後の面積を示す。

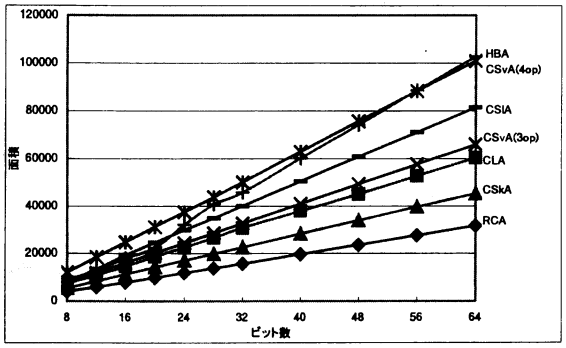

図 4.1　論理合成後の面積

回路構成の最も単純な RCA が最小、次に RCA に桁上げ発生検出回路を付加した CSkA が小さいという結果になった。桁上げ先見回路を持つ CLA は RCA の 2 倍に近い面積となった。桁上げが'1'、'0'の場合の両方を同時に計算する CSlA は、RCA の 2 倍以上の面積になった。CSvA については、3 オペランドのものは RCA のほぼ 2 倍、4 オペランドのものは RCA のほぼ 3 倍となっている。HBA は CSlA に加えて桁上げ演算専用の回路を設けているため、今回設計した 2 オペランド加算器の中では最も面積が大きくなっている。

4.2 消費電力のシミュレーション結果

図 4.2 に、消費電力のシミュレーション結果を示す。

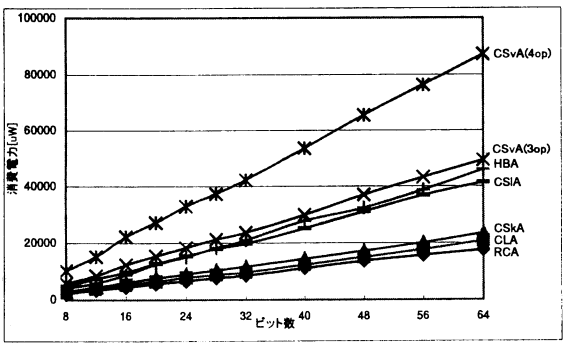

図 4.2　消費電力シミュレーション結果

回路面積では CLA は RCA の 2 倍近く、CSkA は RCA と CLA の中間に位置していたが、消費電力の値は均衡する結果となった。特に、CSkA と CLA では面積は CLA の方が大規模であるが、消費電力は CLA の方が少ないという結果が得られた。

CSlA と HBA は、面積においては CSvA(3op)より大きくなっているが、消費電力においては双方とも CSvA(3op)を下回っている。CSvA(4op)に至っては、面積は HBA とほぼ同等であるが、消費電力は他の加算器を引き離して高い値を示した。

4.3 演算速度のシミュレーション結果

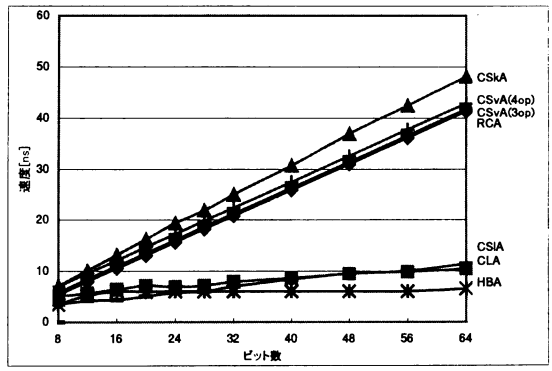

図 4.3　演算速度のシミュレーション結果

大きく分けて 2 つのグループに分かれた。高速な HBA、CLA、CSlA はいずれも桁上げを先見する形式である。高速化を主眼に置いて設計した HBA は、いずれの加算形式よりも高速に演算を行うという結果になった。

桁上げを先見する方式の中で、CSkA のシミュレーション結果は思わしくなく、全加算形式の中で最も低速という結果になった。

RCA、CSvA(3op)、CSvA(4op)の 3 種はいずれも演算時間に殆ど変化が無かった。

5. 考察

4.2 章の結果を見ると、逐次桁上げを用いる形式は消費電力特性が思わしくなく、桁上げを先見する形式は消費電力特性が良いという傾向が見て取れる。

逐次桁上げ方式は、入力を受け取ってから LSB から順に和が確定していく。桁上げが最上位まで伝播する間、上位ビットは本物ではない桁上げを随時受け取ることとなり、その結果、回路の遷移回数が増大する。これが逐次桁上げ方式の消費電力特性が悪化する原因である。

対して、桁上げ先見方式は高速に上位への桁上げを確定する。そのため、逐次桁上げ方式と比較して下位

ビットの遷移が上位ビットに影響を及ぼさず、演算を高速に収束することが可能であり、消費電力の低減に繋がっている。

一般に、RCA は小面積、低消費電力と言われているが、今回の研究結果から一概にそうとは言えないということが判明した。回路を構成する素子数を抑えるよりも、回路の遷移回数を抑えることが低消費電力化の鍵と言える。

演算速度については、高速なグループと低速なグループの 2 つに分かれたが、低速なグループの加算形式はいずれも基本構成に RCA を用いた回路である。

その中でも、CSkA の演算速度は桁上げ先見方式にも関わらず最も遅いという結果が得られた。今回の演算速度の結果は最悪の条件、つまり最も演算に時間の掛かる場合を想定している。CSkA の場合、2 章で述べたように RCA に桁上げ検出用の回路を付加したものであるが、最悪の場合は各グループ内の RCA の部分全てを通る事となり、その場合における演算時間は通常の RCA の演算時間に加え、各グループに存在する OR ゲートの演算時間が加算される。

3 オペランドや 4 オペランドを加算する CSvA の演算時間は、RCA とほぼ同じ値となっている。これは CSvA が最終段に RCA を持つためである。

既存の形式を組み合わせ、HBA を作成することは演算速度を高速化する上で非常に有用である。HBA の組み合わせは無数に存在するが、今回設計した加算機は和演算に RCA を用いている。この部分をより高速な加算形式に置き換えることで、更なる高速化が期待できる。演算時間の高速化には、桁上げ先見方式を組み合わせることが有効である。

6. 今後の展望

本研究では RCA, CLA, CSlA, CSkA, CSvA, HBA の 6 つについてシミュレーションを行ったが、他にも Ling adder や Conditional sum adder, Prefix adder といった方式も存在し、今後はこれらについての研究を進めていきたいと考えている。

また、HBA は組み合わせる演算方式によって、更に高速化，低消費電力化することができる可能性があるので、これらの点について研究・改良していきたいと考えている。

本研究は東京大学大規模集積システム設計教育研究センターを通し、シノプシス株式会社の協力で行われたものである。

文　献

[1] 長谷川裕恭,"VHDL によるハードウェア設計入門,"CQ 出版社, 東京, 1995.

[2] Israel Koren,"Computer Arithmetic Algorithms Second Edition,"A K Peters,Ltd, Canada, 2001.

[3] Behrooz Parhami, "Computer Arithmetic ALGORITHMS AND HARDWARE DESIGNS, "OXFORD UNIVERSITY PRESS, New York, 2000.