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

2004－ＳＬＤＭ－115　（３）
2004／5／27

冗長２進数の絶対値計算を用いた整数除算回路

門脇　俊介†　　高木　直史†　　高木　一義†

† 名古屋大学大学院 情報科学研究科 情報システム学専攻
〒 464–8603 名古屋市千種区不老町
E-mail: †{kadowaki,ntakagi,ktakagi}@takagi.nuie.nagoya-u.ac.jp

あらまし　本稿では、非回復型の除算アルゴリズムに基づいた新しい整数除算回路を提案する。各部分剰余を冗長２進数で表し、その絶対値を計算するとともに、その部分剰余の符号も求めていくものである。その符号を用いて商の各桁を最上位桁から順に決定していく。n ビットの被除数、除数から n ビットの商と剰余を $O(n)$ で計算できる。32 ビットの被除数、除数から 32 ビットの商と剰余を出力する整数除算器として、提案するアルゴリズムに基づく回路を組合せ回路として実現した。順次桁上げ加算器を用いて実現した非回復型整数除算器と比較すると遅延時間は約 65% の減少であった。また、桁上げ先見加算器を用いて実現した非回復型整数除算器と比較すると遅延時間は約 25% の減少であった。

キーワード　整数除算、冗長２進数、絶対値計算

Integer Divider Using Absolute Value Computation of Redundant Binary Numbers

Shunsuke KADOWAKI†, Naofumi TAKAGI†, and Kazuyoshi TAKAGI†

† Department of Information Engineering, Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464–8603 Japan
E-mail: †{kadowaki,ntakagi,ktakagi}@takagi.nuie.nagoya-u.ac.jp

Abstract　A new integer divider based on non-restoring division algorithm is proposed. Each partial remainder is expressed with the redundant binary number. While calculating the absolute value, the sign of the partial remainder is also calculated. Each digit of the quotient is determined sequentially from the most significant digit using the sign. It only takes $O(n)$ to calculate n-bit quotient and n-bit remainder from n-bit dividend and n-bit divisor. The combinational circuit based on the proposed algorithm is realized as integer divider which outputs 32-bit quotient and 32-bit remainder from 32-bit dividend and 32-bit divisor. When it is compared with combinational circuit using ripple carry adders, delay decreased about 65%. When combinational circuit based on the proposed algorithm is compared with non-restoring integer divider using carry lookahead adders, delay decreased about 25%.

Key words　Integer Division, Redundant Binary Numbers, Absolute Value Computation

1. まえがき

加減乗除等の算術演算は、計算機などのディジタルシステムにおいて、基本演算として広く用いられており、従来からその高速化が望まれている。算術演算については、既にさまざまな研究がなされており、いろいろなアルゴリズムが提案され、実現されてきた。しかし、整数の被除数と除数から整数の商と剰余を求める整数除算では商の１桁を求めるのにワードの大きさに応じた加算もしくは減算を行う必要があり、演算に時間がかかる。また、整数除算をソフトウェアだけで実行した場合、整数除算器を持っていない CPU ではソフトウェアの繰り返し処理によって行われる。ソフトウェアで行っている処理をハードウェア化することで処理を高速化することができる。

除算法は、減算シフト型除算法と乗算型 (二次収束型) 除算法に大別される。減算シフト型除算法は、減算とシフトの繰り返しにより商を上位桁から順に求めていくもので、回復型除算法の他、非回復型除算法や SRT 除算法などが知られている。減算シフト型除算法に基づく除算器は、減算器とシフタからなる順序回路として、比較的容易に実現できる。また、組合せ回路として実現すると、規則正しいセル配列構造になる。しか

し、通常の2進表現を用いて演算を行う場合、各減算における桁借りの伝搬が起こってしまうため、計算に時間を要する。一方、乗算型除算法は、乗算の繰り返しにより除算を行うもので、Newton 法に基づく除算法、Goldschmidt の除算法、Chen の除算法等がある。乗算型除算法では除数の逆数の近似値を用いて計算を高速化するが、除数が正規化されていることが前提であるから、整数除算では適用が難しい。また、組合せ回路として実現すると膨大なハードウェア量が必要となってしまう。

本報告では、非回復型のアルゴリズムに基づいた新しい整数除算法を提案する。各部分剰余を冗長2進数で表し、その絶対値を計算するとともに、その部分剰余の符号も求めていくものである。その符号を用いて商の各桁を上位桁から順に決定していく。部分剰余を上位桁から求めていく計算をパイプライン処理することで演算の高速化を実現している。

本報告の整数除算法に基づき、n ビット整数除算器を組合せ回路として実現すると、回路の段数 (計算時間) は n に比例し、回路の素子数は n^2 に比例する。この整数除算器を 8 ビット、16 ビット、32 ビット、64 ビットで設計し、それぞれ遅延時間、面積について評価を行った。32 ビットでは、遅延時間は順次桁上げ加算器を用いた非回復型整数除算器の約半分になった。面積は約 2 倍であった。

2 章では、準備として非回復型整数除算法と冗長2進数について述べ、3 章で冗長2進数の絶対値計算を用いた新しい整数除算法について述べる。4 章で提案した回路の構成を示す。5 章で回路の評価とそれに基づく考察を述べ、6 章で結論を述べる。

2. 準　　備

2.1 非回復型整数除算法

本節では減算シフト型整数除算法として知られる非回復型整数除算法について述べる。本報告で提案する整数除算法は非回復型整数除算法に基づいている。

本報告では n ビット2進整数の除算について考える。n ビット2進整数 $[x_1x_2\ldots x_n]_2(x_i \in 0,1)$ は $\sum_{i=2}^{n} 2^{n-i}x_i - 2^{n-1}x_1$ という値を表わす。

非回復型除算法は減算シフト型除算法の一種である。減算シフト型整数除算法は次の漸化式で表わされる [1]。r は基数、D は除数、q_j は商の最上位桁からの j 桁目、R_j は q_j を決定した後の部分剰余である。R_0 を被除数 N とする。

$$R_j = R_{j-1} - q_j \times r^{n-j} \times D \quad (1 \leq j \leq n) \qquad (1)$$

本報告では基数 r を 2 とする。(1) の漸化式で、$q_j \in \{-1, 1\}$ とし、R_{j-1} が非負なら q_j を 1、負なら -1 とするのが非回復型整数除算法である。常に $-2^{n-j}D \leq R_j < 2^{n-j}D$ が成り立つ。$R_{j-1} - q_j 2^{n-j}D$ が負になっても、これをそのまま R_j とすることから、引き放し法、または非回復型除算法と呼ばれる。商を通常の2進表現に変換する必要があるが、2進表現の商 Z の j 桁目 z_j は q_{j+1} が 1 なら 1、-1 なら 0 となり、変換には特に計算の必要はない。剰余 R を $R \geq 0$ とするには、$R_n \geq 0$ なら $z_n = 1, R = R_n$ とし、$R_n < 0$ なら $z_n = 0, R = R_n + D$ とする。途中で $R_j = 0$ になれば、計算を終了するようにしてもよい。

商の変換を漸化式に組み込むことも可能である。$R_1 = N - 2^{n-1}D$ とし、$R_1 \geq 0$ ならば $q_1 = 1$、$R_1 < 0$ ならば $q_1 = 0$ とする。$j \geq 2$ について、$R_{j-1} \geq 0$ ならば $R_j = R_{j-1} - 2^{n-j}D$、$R_{j-1} < 0$ ならば $R_j = R_{j-1} + 2^{n-j}D$ とし、$R_j < 0$ であれば $q_j = 0$、$R_j \geq 0$ であれば $q_j = 1$ とする。以下の漸化式で部分剰余の計算を行い、その符号に基づき商を上位から決定していく。

$$R_j = \begin{cases} R_{j-1} + 2^{n-j}D & \text{if } R_{j-1} < 0 \\ R_{j-1} - 2^{n-j}D & \text{if } R_{j-1} \geq 0 \end{cases}$$

$$q_j = \begin{cases} 0 & \text{if } R_j < 0 \\ 1 & \text{if } R_j \geq 0 \end{cases}$$

2.2 冗長2進数

本報告で利用する冗長2進表現は、Avizienis が提案した SD(Signed Digit) 表現 [2] の一種で、基数 2 の拡張 SD 表現である。$X = [x_1x_2\ldots x_n]_{SD2}$ は、$\sum_{i=1}^{n} 2^{n-i}x_i$ という値を表わし、各桁 x_i は $\{-1, 0, 1\}$ の要素である。以後、各桁の -1 を $\bar{1}$ と表わす。1 つの冗長2進表現は 1 つの数を表わすが、1 つの数をいくつかの冗長2進表現で表わすことができる。例えば、'5' は $[0101]_2$ の他 $[011\bar{1}]_{SD2}$、$[1\bar{1}01]_{SD2}$ などと表わすことができる。このように、冗長2進表現は冗長性をもつ。ただし、'0' の表現は一意である。

2 進表現から冗長2進表現への変換には特に計算の必要はない。2 進整数が正の場合、$[x_1x_2\ldots x_n]_2$ と冗長2進数 $[x_1x_2\ldots x_n]_{SD2}(x_i \in 0, 1)$ は、ともに、$\sum_{i=1}^{n} 2^{n-i}x_i$ という値を表わす。2 進整数が負の場合、つまり、最上位ビットが 1 の場合は最上位ビットを $\bar{1}$ に変換するだけで冗長2進表現に変換することができる。

冗長2進表現から2進表現への変換は、2 つの 2 進数の通常の減算によって行える。冗長2進数 $[x_1x_2\ldots x_n]_{SD2}(x_i \in \bar{1}, 0, 1)$ は $\sum_{i=1}^{n} 2^{n-i}x_i = \sum_{x_i=1} 2^{n-i} - \sum_{x_i=\bar{1}} 2^{n-i}$ という値を表わす。従って、冗長2進表現で 1 になっている桁だけを 1 とする符号なし 2 進数 X^+ と $\bar{1}$ になっている桁だけを 1 とする符号なし 2 進数 X^- を考え、通常の減算 $X^+ - X^-$ により、同じ値を表わす 2 進数に変換できる。例えば、$[1\bar{1}01]_{SD2} = [1001]_2 - [0100]_2 = [0101]_2$ というように変換される。

2 進数の加減算においては、桁上げもしくは桁借りの伝搬のため、組合せ回路を用いても少なくとも演算数の桁数の対数に比例する計算時間が必要である。これに対し、冗長2進数体系では、その冗長性を利用し、加減算において桁上げもしくは桁借りが高々1 桁しか伝搬しないようにすることができ、組合せ回路による並列加減算が演算数の桁数に関係なく一定時間で行える。本報告では特に、減数が各桁非負の冗長2進表現である場合について考える。この場合、演算数が双方とも各桁非負とは限らない冗長2進表現である場合よりも容易になる。

冗長2進数と各桁非負の冗長2進数の減算について考える。減算は 2 ステップで行われる。ステップ 1 では、各桁におい

て、被減数と減数から中間桁借りと中間差を求める。ステップ1での計算規則を表1に示す。冗長2進表現では'−1'を2桁で$[\bar{1}1]_{SD2}$と表わすことができるので、被減数が-1で減数が0の場合、もしくは被減数が0で減数が1の場合、中間桁借りを1、中間差を1とする。ステップ2では、各桁において、ステップ1で求めた中間差から1つ下位の桁の中間桁借りを減算する。表1からわかるように、中間差は0か1であり、中間桁借りも0か1であるから、ステップ2では桁借りは生じない。このように、冗長2進数と各桁が非負の冗長2進数の減算においては、桁借りが高々1桁しか伝搬しないようにすることができ、従って、組合せ回路による並列減算が演算数の桁数に関係なく一定時間で行える。

表1　冗長2進数と各桁が非負の冗長2進数の減算のステップ1の計算規則

減数＼被減数	-	$\bar{1}$	0	1
0	1,1	0,0	0,1	
1	1,0	1,1	0,0	

中間桁借り、中間差

3. 提案アルゴリズム

本章では、冗長2進数の絶対値計算を用いた整数除算アルゴリズムを提案する。3.1で、提案するアルゴリズムに用いる冗長2進数の絶対値計算法について述べ、3.2で冗長2進数の絶対値計算を用いた整数除算アルゴリズムを提案する。

3.1 冗長2進数の絶対値計算法

本稿で提案する整数除算アルゴリズムでは、商1桁を求める各反復での部分剰余の計算に絶対値計算を用いる。そこで、本節では冗長2進数の絶対値を計算するアルゴリズムについて述べる。[3]で述べられている冗長2進数の絶対値の計算法では、入力の冗長2進数の絶対値を計算するとともに、符号も出力する。ある冗長2進数が正であるか負であるかは最上位の0でない桁の符号によって判定できる。

まず、入力数の最上位桁から順に調べ、0でない桁を見つけると符号が決まる。最初に見つけた0でない桁が1であれば、符号を正とし、入力数をそのまま出力する。最初に見つけた0でない桁が$\bar{1}$であれば、符号を負とし、入力数のそれぞれの桁を正負反転して出力する。図1にアルゴリズムを示す。Fは符号を判定するフラグで、Xが正ならばpositive、Xが負ならばnegative、その段階でXの符号が決定していなければnodec(no decision)とする。

入力：冗長2進数 X ($X=[x_1,\ldots,x_n]_{SD2}$)
出力：正の冗長2進数 $Y=|X|$ ($Y=[y_1,\ldots,y_n]_{SD2}$)、
　　X の符号出力 S

ステップ1

$F_0 = \text{nodec}$

ステップ2

for $j=1$ to n do

begin

$$y_j = \begin{cases} x_j & \text{if } F_{j-1}=\text{positive} \\ -x_j & \text{if } F_{j-1}=\text{negative} \\ x_j & \text{if } F_{j-1}=\text{nodec and } x_j \geqq 0 \\ -x_j & \text{if } F_{j-1}=\text{nodec and } x_j=-1 \end{cases}$$

$$F_j = \begin{cases} F_{j-1} & \text{if } F_{j-1}=\text{positive or negative} \\ \text{positive} & \text{if } F_{j-1}=\text{nodec and } x_j=+1 \\ \text{negative} & \text{if } F_{j-1}=\text{nodec and } x_j=-1 \\ \text{nodec} & \text{if } F_{j-1}=\text{nodec and } x_j=0 \end{cases}$$

end

ステップ3

$$S = \begin{cases} \text{positive} & \text{if } F_n=\text{positive or nodec} \\ \text{negative} & \text{if } F_n=\text{negative} \end{cases}$$

図1　冗長2進数の絶対値計算アルゴリズム

3.2 冗長2進数の絶対値計算を用いた整数除算アルゴリズム

本節では冗長2進数の絶対値計算を用いた整数除算アルゴリズムを提案する。2.1の非回復型整数除算アルゴリズムでは、各反復において部分剰余を計算し、部分剰余が正ならば商を1として次の反復では減算、部分剰余が負ならば商を0として次の反復では加算ということを繰り返していた。このアルゴリズムでは、nビットの整数除算の場合、各反復にnビットの加減算を行う必要があり、その加減算が終了しなければ正負判定が行えない。加減算に順次桁上げ加算器を用いると$O(n)$の計算時間がかかるため、整数除算全体として$O(n^2)$の計算時間がかかる。加減算に桁上げ先見加算器を用いても$O(\log n)$の計算時間がかかり、整数除算全体として$O(n\log n)$の計算時間がかかる。本報告では、各反復で加減算を行う代わりに前の反復での部分剰余の絶対値をとり、それから除数を減ずることにより部分剰余を求める。各反復で部分剰余の絶対値をとるため正負判定を行う必要がなく、常に減算を行えばよい。減算は冗長2進数体系で上位桁から順に行っていく。商決定のために部分剰余の最初に出てきた0でない桁($\bar{1}$か1)の符号を求める。3.1の絶対値計算アルゴリズムにより、部分剰余の絶対値計算と符号計算は同時に行える。

nビットの被除数と除数に対し、nビットの商と剰余を求める。R_jをj回目の繰り返しでの部分剰余、q_jを商の最上位桁からj桁目とする。R_0は被除数を正の冗長2進数に変換したもの、Dは除数を正の数に変換したものとする。正の数への変換は被除数、除数ともに定数時間で行える。S_jは各部分剰余R_jの符号を表し、正ならば0、負ならば1とする。次の漸化式

で計算を行う。

$$R_j = |R_{j-1}| - 2^{n-j}D \quad (1 \leqq j \leqq n)$$

$$q_j = \begin{cases} \overline{S_j} & \text{if } j = 1 \\ q_{j-1} \oplus S_j & \text{if } j \geqq 2 \end{cases}$$

商 Q は上位桁から順に2進表現で求められていく。被除数と除数がともに正もしくはともに負ならば $Q = [q_1 \ldots q_n]$ を商とし、被除数と除数の符号が異なっていれば Q の補数をとったものを商とする。剰余は冗長2進数で表されているので、2進表現に変換する。被除数が正ならば剰余を正、被除数が負ならば剰余を負とする。図2にアルゴリズムを示す。

入力:被除数 X、除数 Y(いずれも n ビット2進整数)
出力:商 Z、剰余 R(いずれも n ビット2進整数)
S_j は各部分剰余 R_j の符号を表し、正のとき0、負のとき1

ステップ1

$R_0 \leftarrow X$ (正の冗長2進数に変換)

$$D = \begin{cases} Y & \text{if } Y \geqq 0 \\ -Y & \text{if } Y < 0 \text{ (補数をとる)} \end{cases}$$

ステップ2

$R_1 = R_0 - 2^{n-1}D$ (冗長2進で計算)

$$S_1 = \begin{cases} 0 & \text{if } R_1 \leqq 0 \\ 1 & \text{if } R_1 < 0 \end{cases}$$

$q_1 = \overline{S_1}$

ステップ3

for $j = 2$ to n do

begin

$R_j = |R_{j-1}| - 2^{n-j}D$ (冗長2進で計算)

$$S_j = \begin{cases} 0 & \text{if } R_j \leqq 0 \\ 1 & \text{if } R_j < 0 \end{cases}$$

$q_j = q_{j-1} \oplus S_j$

end

ステップ4

$Q = [q_1 \ldots q_n]$

$$Z = \begin{cases} Q & \text{if } Q \geqq 0 \\ -Q & \text{if } Q < 0 \text{ (補数をとる)} \end{cases}$$

$$R' = \begin{cases} ||R_n| - D| & \text{if } q_n = 0 \\ |R_n| & \text{if } q_n = 1 \end{cases}$$

$$R = \begin{cases} R' & \text{if } X \geqq 0 \\ -R' & \text{if } X < 0 \text{(補数をとる)} \end{cases}$$

図2 冗長2進数の絶対値計算を用いた整数除算アルゴリズム

図3に提案アルゴリズムを用いた整数除算の計算例を示す。5ビットの被除数 $X = [01110]_2$ を5ビットの除数 $Y = [00011]_2$ で除し、5ビットの商 Z と剰余 R を出力する。$R_0 = X$ とし、冗長2進数体系での減算を繰り返す。

図3の計算例では、最後の部分剰余の絶対値 $|R_5|$ が $[0000000001]_{SD2}$ となっているが、$q_5 = 0$ なので、$R = |[00001]_{SD2} - [00011]_2| = [00010]_{SD2}$ が剰余となる。商は $Z = [00100]_2$ である。

4. 提案するアルゴリズムに基づく回路

4.1 組合せ回路実現

前章で提案した整数除算アルゴリズムに基づく除算器を組合せ回路として構成する。回路は減算用セル (sub)、絶対値計算用セル (abs)、商決定 (被除数変換) 用セル (conv)、擬似オーバーフロー補正用セル (over) からなる。商決定 (被除数変換) 用セル (conv) は部分剰余の符号により商を決定するとともに、次の反復での入力となる被除数桁の符号を変換して出力する。

入力は n ビットの正の被除数と除数とする。出力は n ビットの商と剰余である。冗長2進数の1桁を2ビットで表す。組合せ回路として実現したものを図4に示す。減算用のセル n 個と絶対値計算用のセル n 個が横一列に並んで、前章に示した整数除算アルゴリズムの反復1回分の計算を行い、最下位の商決定セルで商の1桁を決定する。各段の最上位には擬似オーバーフロー補正用のセルを配置する。n 段並べることで n 回の繰返し分の計算を行うことができる。被除数、除数が負の数の場合も

$X = [01110]_2, Y = [00011]_2 \quad X \div Y$

	R_0	0	0	0	0	0	0	1	1	1	0
−)	Y			0	0	0	1	1			
	R_1	0	0	0	$\bar{1}$	0	1	1	$\bar{1}$	1	0
							$S_1 = 1$ (negative)				$q_1 = 0$
	$\|R_1\|$	0	0	0	1	0	$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	0
−)	Y			0	0	0	1	1			
	R_2	0	0	0	1	$\bar{1}$	$\bar{1}$	$\bar{1}$	0	1	0
							$S_2 = 0$ (negative)				$q_2 = 0$
	$\|R_2\|$	0	0	0	1	$\bar{1}$	$\bar{1}$	$\bar{1}$	0	1	0
−)	Y				0	0	0	1	1		
	R_3	0	0	0	0	0	0	$\bar{1}$	1	1	0
							$S_3 = 1$ (negative)				$q_3 = 1$
	$\|R_3\|$	0	0	0	0	0	0	1	$\bar{1}$	$\bar{1}$	0
−)	Y					0	0	0	1	1	
	R_4	0	0	0	0	0	0	0	$\bar{1}$	0	0
							$S_4 = 1$ (negative)				$q_4 = 0$
	$\|R_4\|$	0	0	0	0	0	0	0	1	0	0
−)	Y						0	0	0	1	1
	R_5	0	0	0	0	0	0	0	0	0	1
							$S_5 = 0$ (positive)				$q_5 = 0$
	$\|R_5\|$	0	0	0	0	0	0	0	0	0	1

図3 提案アルゴリズムを用いた整数除算の計算例

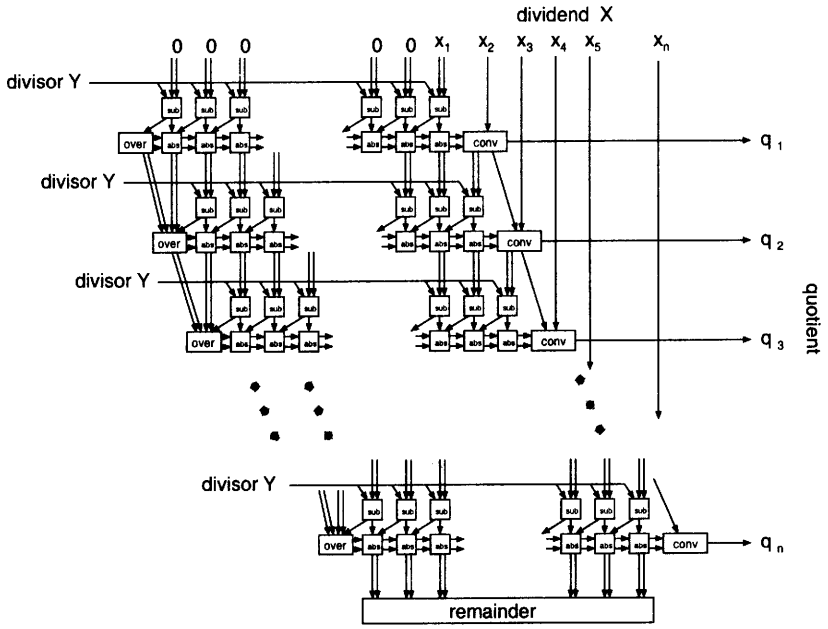

図 4　冗長 2 進数の絶対値計算を用いた整数除算回路 (組合せ回路)

扱える。被除数、除数の正数への変換はともに定数時間で行える。被除数については 2.2 で述べた 2 の補数表現された負数から冗長 2 進数への変換法を用い、さらに各桁の符号を反転すれば正数となる。除数は各桁の 0 と 1 を反転し、最下位の桁借りを 1 とすればよい。

4.2　順序回路実現

組合せ回路では反復 1 回分の計算を行う回路を n 段並べることで n 回の繰返し分の計算を行う。図 4 の回路では同じ計算を繰り返し行っている。そこで、繰り返しを行っている部分を取り出して、それを組合せ回路として実現し、その出力をレジスタに保持していくことで順序回路として実現できる。各反復での部分剰余をレジスタに保持していっても順序回路として実現できるが、1 クロックあたりの時間（サイクルタイム）が大きくなり、全体の計算時間が大きくなってしまう。そこで図 5 のような構成を考える。レジスタでは桁借りや符号などの中間的な値を保持していく。サイクルタイムは減算セル 2 個分と絶対値計算セル 2 個分の遅延時間をあわせた時間であり、n に依らず、定数時間である。最初の商の桁（最上位ビット）を出力するまでに $\frac{n}{4}+1$ クロックかかる。ある商の桁を出力してから次の商の桁を出力するまでは 1 クロックである。よって、$\frac{n}{4}+1+(n-1)=\frac{5}{4}n$ クロックで商の最下位桁が出力される。

5.　回路の評価と考察

5.1　組合せ回路

本章では、本報告で提案した整数除算法と従来の非回復型整数除算法を、組合せ回路で実現したものの比較を示す。それぞれ、8 ビット、16 ビット、32 ビット、64 ビットの場合について比較した。Verilog-HDL にて各整数除算器を記述し、セルライブラリにローム社 0.35μm プロセス用東京大学大規模集積システム設計教育センター (VDEC) 版 EXD 社ライブラリを用いて論理合成を行った。表 4、表 5 に順次桁上げ加算器、桁上げ先見加算器を用いた非回復型の整数除算回路を、表 6 に本報告で提案した回路の遅延時間、面積、ゲート数について示す。

表 2　順次桁上げ加算器を用いた非回復型整数除算回路

	遅延時間 [ns]	面積 [mm^2]	ゲート数 [NAND2 換算]
8 ビット	19.03	0.068255	919
16 ビット	62.83	0.270054	3637
32 ビット	222.45	1.091174	14696
64 ビット	829.97	4.322598	58217

表 3　桁上げ先見加算器を用いた非回復型整数除算回路

	遅延時間 [ns]	面積 [mm^2]	ゲート数 [NAND2 換算]
8 ビット	17.56	0.126428	1703
16 ビット	41.96	0.515055	6937
32 ビット	101.39	1.961451	26417
64 ビット	234.43	7.453733	100387

表 4　冗長 2 進数の絶対値計算を用いた整数除算回路

	遅延時間 [ns]	面積 [mm^2]	ゲート数 [NAND2 換算]
8 ビット	18.98	0.172852	2328
16 ビット	36.31	0.657122	8850
32 ビット	77.94	2.348634	31631
64 ビット	160.37	8.878933	119582

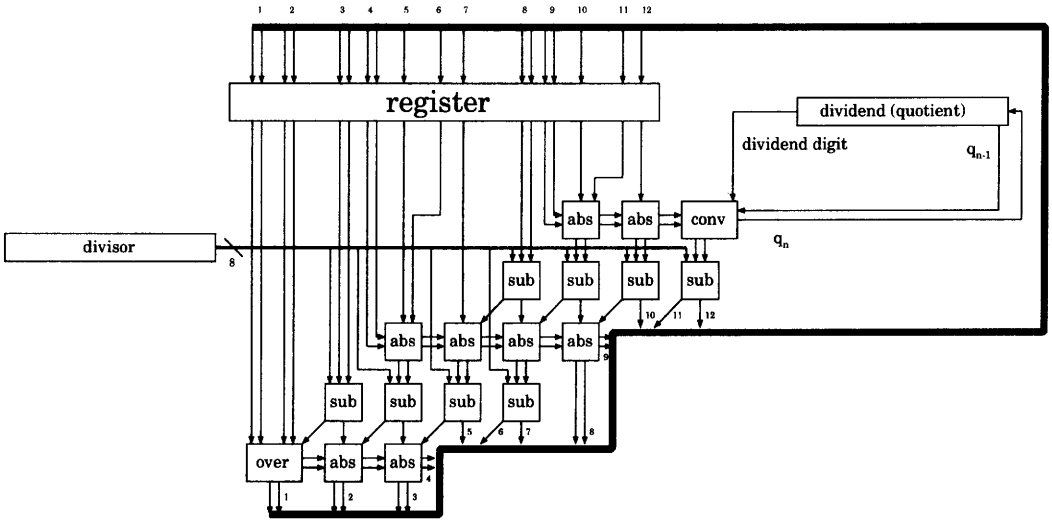

図 5　冗長 2 進数の絶対値計算を用いた整数除算回路 (8 ビットの順序回路)

順次桁上げ加算器を用いた従来の非回復型整数除算回路では、ビット数が 2 倍、4 倍、8 倍となるにつれ、遅延時間、面積は 4 倍、16 倍、64 倍になっている。つまり、遅延時間も面積もビット数の 2 乗に比例している。本報告で提案した回路ではビット数が 2 倍、4 倍、8 倍となるにつれ、面積は 4 倍、16 倍、64 倍となっているが、遅延時間は 2 倍、4 倍、8 倍としかならない。つまり、面積はビット数の 2 乗に比例しているが、遅延時間はビット数に比例している。8 ビットの整数除算回路では、桁上げ先見加算器を用いた非回復型整数除算回路のほうが提案する回路よりも面積、遅延時間ともに小さい。しかし、32 ビットの整数除算回路では桁上げ先見加算器を用いた非回復型整数除算回路に比べ、遅延時間は約 25% 減少している。面積は約 20% の増加である。本報告で提案した回路は、順次桁上げ加算器を用いた従来の非回復型整数除算回路に比べて面積は約 2 倍となっている。減算用セルと絶対値計算用セルをあわせた面積は FA 約 2 個分に相当するためである。

5.2 順序回路

順序回路では 1 クロックで減算セル 2 個と絶対値計算セル 2 個分の計算を行う。1 クロックあたりの遅延時間は FA 4 個分に相当する。被除数、除数が n ビットの場合、商 n ビットを出力するために必要なクロック数は $\frac{4}{5}n$ である。全体の計算時間 (商の最下位桁を出力するまでの時間) は 8 ビットでは 10 クロック、16 ビットでは 20 クロック、32 ビットでは 40 クロック、64 ビットでは 80 クロックとなる。面積は組合せ回路での 1 反復分にレジスタを合わせたものとなる。減算用セル n 個と絶対値計算用セル n 個、商決定用セル 1 個、擬似オーバーフロー補正用セル 1 個分にレジスタの面積を合わせたものが、n ビットの整数除算を行う順序回路の面積となる。

6. ま と め

本報告では非回復型の整数除算法に基づき、冗長 2 進数の絶対値計算を用いた整数除算回路を提案した。n ビットの整数除算回路を組合せ回路として実現した場合、順次桁上げ加算器を用いた非回復型の整数除算回路では遅延時間が $O(n^2)$ となり、素子数も $O(n^2)$ となる。また、桁上げ先見加算器を用いた非回復型の整数除算回路では遅延時間は $O(n \log n)$ となり、素子数は $O(n^2)$ となる。本報告の整数除算回路では、n ビットの整数除算での遅延時間は $O(n)$ である。素子数は $O(n^2)$ である。

従来の非回復型整数除算法では、商を求めるための各反復で加減算を行わなければならないが、本報告で提案する整数除算法では部分剰余を冗長 2 進数で表すことで桁借りもしくは桁上げが伝播しないようにし、また、部分剰余の絶対値を求めていくことで常に減算を行うようにしている。よって部分剰余の符号がわかっていなくても、次の計算を行うことができる。部分剰余を上位桁から求めていく計算をパイプライン処理することで、演算の高速化を実現した。

文　献

[1] 高木直史, "除算回路のアルゴリズム-減算シフト型か乗算型か-," 情報処理 Vol.37, No.3,1996 年 3 月.

[2] Avizienis,A. "Signed-Digit Number Representations for Fast Parallel Arithmetic," IRE Transactions on Electronic Computers, EC-10, pp.389-400, September 1961.

[3] Herbert Dawid and Heinrich Meyr, "The Differential CORDIC Algorithm: Constant Scale Factor Redundant Implementation without Correcting Iterations," IEEE Transactions on Computers, Vol.45, No.3, March 1996.

[4] 高木直史, 安浦寛人, 矢島脩三, "冗長 2 進表現を利用した VLSI 向き高速除算器," 電気通信学会論文誌,vol.J67-D,No.4,1984 年 4 月.

[5] David W.Matula, Alex Fit-Florea, "Prescaled Integer Division," 16th IEEE SYMPOSIUM on Computer Arithmetic, pp.63-68,2003.

[6] Patterson, Hennessy, 成田光彰訳, "コンピュータの構成と設計-ハードウエアとソフトウエアのインターフェース-," 日経 BP 社.