

配線量を低減する項積分法による演算回路の設計

6K-4

小林 芳直

日本アイ・ビー・エム株式会社東京基礎研究所

1 概略

ディープサブミクロンの VLSI では配線負荷が信号遅延の大きな要因となることが知られている。配線量を減らす事ができれば回路の動作速度を上げることができる。演算回路の中では符号拡張とよばれるビット操作があり、この操作は変数の符号ビットのレプリカを作るので配線負荷が大きくなる。ここで提案する項積分法は、この符号拡張を定数の加算という別の演算に置き換えることにより、配線負荷を軽減しようというものである。この項積分法には別の機能もある。それは演算の前後で変数の符号を自由に選べることである。この機能を使えば演算回路の前後にある極性を調整するための回路を簡略化できる。0.25 μ の ASIC において本方式を採用した結果、回路サイズの削減と高速化に効果があったので報告する。

2 定数加算による符号拡張

整数を表した 2 進数は、符号ビットが負の数で、符号ビット以外のビットを正の数として扱う事ができる。

変数のビット数を拡張する場合には符号拡張と呼ばれるビット操作が行われる。これは拡張するために必要な個数だけ、符号ビットのレプリカを上位に並べていくものである。例えば 8 ビットの数を 16 ビットに拡張するときには、符号ビット P を 8 個追加して、“PPPPPPPP-----”というように符号ビットを合計 9 個並べる。ビット拡張した後の変数については最上位のビットが新たな符号ビットになり、残りのビットはもとの符号ビットも含めて正の数として扱われる^[1]。

さてここで、もとの 8 ビットの整数 I に +128 の定数を加算した数を考えると、この加算結果 (I + 128) の変域は、0 ~ +255 になり常に正の数となる。この正の変数 (I + 128) は、もとの整数 I の MSB を反転することによっても得られる。ここでこの正の変数 (I + 128) に 16 ビットで (-128) を表す “1111111110000000” を加算すれば、加算結果は 16 ビットになり、その値はもとの 8 ビットの変数 I の値に等しい。これは符号拡張が定数の加算で置き換えられた事になる (図 1)。

両者を比較すると定数加算法では新たに加算器が必要になるが、見かけの符号ビットの負荷が減る。

$$\begin{array}{r}
 \overline{a_7} \ a_6 \ a_5 \ a_4 \ a_3 \ a_2 \ a_1 \ a_0 \\
 + \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\
 \hline
 (a_7 \ a_7 \ a_7 \ a_7 \ a_7 \ a_7 \ a_7 \ a_7 \ a_6 \ a_5 \ a_4 \ a_3 \ a_2 \ a_1 \ a_0) \\
 = A
 \end{array}$$

図 1 定数加算による符号拡張

3 項積分法

2 進数の整数は符号ビットの負のビットと、その他の正のビットで構成されている。ここで正のビットはそのビットが成立すると変数の値が大きくなるので正論理で表し、負ビットについては、そのビットが成立すると変数の値が小さくなるのでこれを負論理であらわすことにする。このように、もとの変数のビット毎の大小の傾向と、ビットが成立しているかどうかの論理積を計算し、すべて加算したものはもとの変数と一致する。これを積分形式で表すと

$$S = \int \text{微分係数} \times \text{項}$$

この式には積分定数が発生する。これは負ビットを負論理で表すときに発生する余分なオフセットであり、このオフセットをキャンセルするための定数を加算するともとの変数を再現する事が出来る。

4 加算

変数の加算は符号ビットを反転したものをそのまま加算する。8 ビットの整数 A, B の加算ならこのとき 256 (128 + 128) のオフセットが発生するので、これをキャンセルするために -256 (“100000000”) を定数として加算する。加算した結果は 9 ビットになるので加算する定数は 9 ビット必要である (図 2)。

$$\begin{array}{r}
 \overline{a_7} \ a_6 \ a_5 \ a_4 \ a_3 \ a_2 \ a_1 \ a_0 \\
 \overline{b_7} \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \ b_0 \\
 + \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\
 \hline
 A + B
 \end{array}$$

図 2 項積分法による加算

5 減算

引き算をするときには、減数についてビットの正負を入れ替えばよいので、符号ビットを正の数、その他のビットを負の数として扱えば項積分法による表現にすることができる。この減数を被減数に加算すれば引き算になる。

8ビットの整数A, Bについて, A-Bの引き算をする場合には, 正の数Aは符号ビットが負の数なのでここにオフセット(128)が発生し, 減数Bについては符号ビット以外が負の数になるのでここに(127=64+32+16+8+4+2+1)が発生する。これらを合計した255(=128+127)が8ビットの引き算で発生するオフセットである。このオフセットの2の補数の-255="100000001"なる定数を加算すると引き算の結果を得ることができる(図3)。

従来の方法では符号拡張をしてから引き算するので符号のレプリカの部分についても回路が必要だが, この項積分法を使えば, 拡張された部分については定数演算しかないので回路は簡単になる。

$$\begin{array}{r}
 \overline{a_7} \ a_6 \ \overline{a_5} \ \overline{a_4} \ \overline{a_3} \ \overline{a_2} \ \overline{a_1} \ \overline{a_0} \\
 b_7 \ \overline{b_6} \ \overline{b_5} \ \overline{b_4} \ \overline{b_3} \ \overline{b_2} \ \overline{b_1} \ \overline{b_0} \\
 + \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \\
 \hline
 A - B
 \end{array}$$

図3 項積分法による減算

6 積算

積算は符号ビットが‘1’のときに2の補数を加える方法が使われていた^{[2][3]}。

ここでは項積分法による積算を実行するためには, 積項の正負について次のように定義する。まず正のビットどうし, 負のビットどうしの積項は正の積項として正論理で表す。そして正のビットと負のビットの積項は負の積項として負論理で表す。負の積項があるところは定数が加算されているものとみなして, この定数をすべて加算し, 加算した結果の2の補数を積分定数として加算する。

4ビットと6ビットの整数を積算した場合には24個の積項のうち8個の積項が負論理になり, ここに472(=128+64+32+128+64+32+8)のオフセットが発生し, この2の補数である-472("1000101000")を定数として加算する必要がある。従来の方法と比べると変数が4個減り, 定数が2増えている。総合的に見ると, 乗数, 被乗数の符号ビットの負荷が少なくなっているだけでなく, 回路も削減されている事が分かる(図4)。

$$\begin{array}{r}
 \begin{array}{cccccc} & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\ \times & & & & b_3 & b_2 & b_1 & b_0 \end{array} \\
 \hline
 \overline{a_5 b_0} \ a_4 b_0 \ a_3 b_0 \ \overline{a_2 b_0} \ \overline{a_1 b_0} \ \overline{a_0 b_0} \\
 \overline{a_5 b_1} \ a_4 b_1 \ a_3 b_1 \ \overline{a_2 b_1} \ \overline{a_1 b_1} \ \overline{a_0 b_1} \\
 \overline{a_5 b_2} \ a_4 b_2 \ a_3 b_2 \ \overline{a_2 b_2} \ \overline{a_1 b_2} \ \overline{a_0 b_2} \\
 a_5 b_3 \ \overline{a_4 b_3} \ \overline{a_3 b_3} \ \overline{a_2 b_3} \ \overline{a_1 b_3} \ \overline{a_0 b_3} \\
 \hline
 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0
 \end{array}$$

A × B

図4 定数加算法による積算

7 負出力を得る積和算

項積分法はビットの正負と演算結果の大小関係に注目した計算法である。演算回路の入力に変数の2の補数があるは, その変数に関しては符号ビットを正ビット, 他のビットを負ビットとして扱うことにより, そのまま演算回路の入力とすることができる。また出力についても極性を自由に選ぶ事ができる。演算結果の2の補数が必要ときには, 演算回路の中で扱うすべての正のビットを負論理, 負のビットを正論理というように論理を反転して扱えば直接, 演算結果の2の補数を得ることができる(図5)。

$$\begin{array}{r}
 \begin{array}{cccc} a_3 & a_2 & a_1 & a_0 \\ \times & b_3 & b_2 & b_1 & b_0 \end{array} \\
 \hline
 \overline{a_3 b_0} \ a_2 b_0 \ a_1 b_0 \ \overline{a_0 b_0} \\
 \overline{a_3 b_1} \ a_2 b_1 \ a_1 b_1 \ \overline{a_0 b_1} \\
 \overline{a_3 b_2} \ a_2 b_2 \ a_1 b_2 \ \overline{a_0 b_2} \\
 a_3 b_3 \ \overline{a_2 b_3} \ \overline{a_1 b_3} \ \overline{a_0 b_3} \\
 \hline
 \begin{array}{cccc} c_3 & c_2 & c_1 & c_0 \\ \overline{d_3} & \overline{d_2} & \overline{d_1} & \overline{d_0} \end{array} \\
 \hline
 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \\
 \hline
 -(A \times (-B) + C - D)
 \end{array}$$

図5 正負混合の演算

8 まとめ

演算回路の新しい設計パラダイムとして項積分法を提案している。計算の理論からこのような構成が変数の数が最小である事がわかる^[4]。

0.25 μ の ASIC にこの方式を適用して演算回路を設計したところ, 24 ビット精度の積和算器において 7%程度の高速度を図る事ができた。また回路サイズも削減されている事が分かった。

文献

- [1] コンピュータ・アーキテクチャ, ヘネシー & パターソン, 日経BP.
- [2] Computer Architecture and Parallel Processing, Kai Hwang, Faye A. Briggs, McGraw-Hill.
- [3] コンピュータの高速演算方式, Hwang 近代科学社.
- [4] ファインマン計算機科学, A. ヘイ, R. アレン, 岩波書店処理.