Inverse Polarity Techniques for High-Speed/Low-Power Multipliers

Pascal C.H. Meier
ECE Dept, Carnegie Mellon Univ.
5000 Forbes Ave.
Pittsburgh, PA 15217
(412)-268-6646
pascal@ece.cmu.edu

Rob A. Rutenbar
ECE Dept, Carnegie Mellon Univ.
5000 Forbes Ave.
Pittsburgh, PA 15217
(412)-268-3334
rutenbar@ece.cmu.edu

L. Richard Carley
ECE Dept, Carnegie Mellon Univ.
5000 Forbes Ave.
Pittsburgh, PA 15217
(412)-268-3597
lrc@ece.cmu.edu

1. ABSTRACT

Various high-speed techniques have been developed for multipliers, but with the increasing popularity of mobile computing, a recent goal has been to minimize power dissipation. A popular delay-reduction technique applied to adder circuits is polarity inversion of bits. As this optimization reduces transistor count, it also has the potential for lowering power dissipation, and can be effectively applied to Wallace tree partial product reduction stages. We illustrate how this technique reduces power, interconnect capacitance, and chip area. Power reduction of up to 25% is achieved.

1.1 Keywords

Multiplier, low power, inverse polarity.

2. Introduction

Recently emergent portable applications involving DSP tasks require high speed operation while minimizing energy so as to extend battery life. A major power-dissipating, high-delay block in such designs is the multiplier; several well-known techniques exist for multiplier delay reduction, and recently a broad number of power reduction methods have been proposed [1]. We examine the potential of an existing delay optimization method, which has a dual advantage in providing low power operation.

The technique which we term "inverse polarity" is applied to ripple adders in the following manner: two numbers, $A = [a_0\ a_1\ ...\ a_{n-1}]$ and $B = [b_0\ b_1\ ...\ b_{n-1}]$ are added to form $R = [r_0\ r_1\ ...\ r_{n-1}]$. The carry chain in each block of the ripple adder consists of two logic stages: the inverted-carry ($\overline{carry}$) stage and the inverter stage (Fig. 1a). In terms of the number of logic stages, the delay is *2n*. The inverse polarity technique attempts to remove the inverter stage by complementing the input bits at every other bit position. In this manner, an adder of the form in Fig. 1b results, and the delay in terms of logic stages is *n*. If these gates are upsized, the overall delay may be significantly reduced. On the other hand, if devices retain the same size, a power reduction may be obtained as the number of transistors implementing the adder is less. In this case, delay reduction will depend on the gate driving strength versus capacitive load.

In multipliers, the majority of CMOS gates are found in the partial product (PP) reduction stage. Closer analysis of the PP reduction network shows that the operation resembles a two-dimensional ripple addition, with *carries* propagating to higher order bit positions, and *sums* remaining at the same bit order (Fig. 1c) This insight leads to the idea of applying the inverse polarity technique to the PP reduction stage.

In the following sections, we will show how inverse polarity circuits can be used in Wallace tree multipliers [2] to lower power dissipation. We describe inverse polarity circuitry and develop a heuristic for implementing a Wallace tree PP reduction circuit using polarity inversion.

3. Circuit Design In Multipliers

Digital CMOS logic design consists of implementing a logic function while minimizing transistor count, subject to delay and power constraints. To this end, many CMOS logic gates are designed to drive large interconnect lines by using a two-stage structure: the first stage implements the logic, and the second stage consists of a buffer (*i.e.*, an inverter) to drive output capacitance. In this manner, input transistors may be small while the buffer can be made large, resulting in lower total transistor size.

While the logic stage/buffer structure provides strong drive for large loads, multiplier circuits have the interesting characteristic that large net capacitances are not commonly encountered. With a few exceptions, most connections are 2-point nets, and proper placement can ensure that connected components are located fairly close together, resulting in short wires. Therefore, other than the partial product

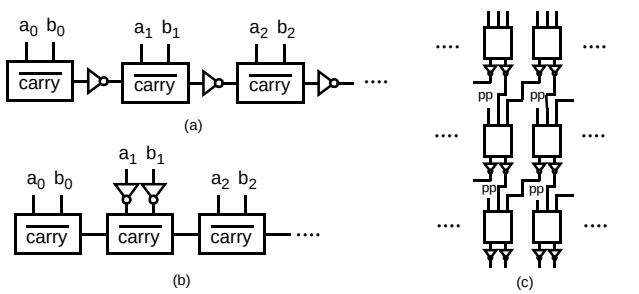

Figure 1. (a) Conventional ripple adder. (b) Inverted polarity optimization (c) Multiplier PPA structure (array).

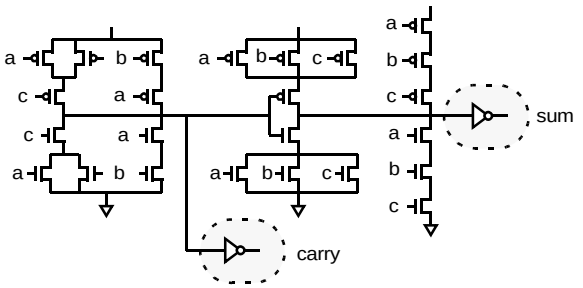

Figure 2. The T28 full adder (CSA) implementation.

generating circuitry and some of the final adder types, the advantage of a buffer structure is minimal. *This suggests that removal of output inverters may realize lower power dissipation with minor effects on delay.* If a logically equivalent implementation using fewer buffers can be assembled, the number of switching transistors and overall power will be reduced. We apply this technique to Wallace tree multipliers to determine resultant power savings.

3.1 Adder Designs

The fundamental building block in digital multipliers is the full adder, used in the partial product reduction phase to perform carry-save addition (therefore sometimes called a carry-save adder, or *CSA*). The CSA takes three inputs and calculates two outputs, *sum* and *carry*. The most commonly used implementation (modified from [3]) is shown in Fig. 2. The popularity of this particular design comes from the frugal use of transistors in implementing both the carry function and the exclusive-or (sum) function: 28 transistors are used, hence the designation "28T" cell. Note that this circuit incorporates the logic stage/buffer structure which is beneficial when driving large output capacitances.

The inverse polarity paradigm identifies bits as one of two polarities—that is, bits will represent the results of additions, *i.e.*, *sum* and *carry* (positive polarity—*POS*), or their complements $\overline{sum}$ and $\overline{carry}$ (negative polarity—*NEG*). Inverted polarity circuits require that an adder with POS inputs provide NEG outputs and vice-versa. Therefore:

$$CSA_{IP}(a,b,c) = (\overline{sum}, \overline{carry})$$

$$CSA_{IP}(\overline{a},\overline{b},\overline{c}) = (sum, carry).$$

Similarly for half adders:

$$HA_{IP}(a,b) = (\overline{sum}, \overline{carry})$$

$$HA_{IP}(\overline{a},\overline{b}) = (sum, carry).$$

The 28T implementation of the CSA can be transformed into a CSA_{IP} by simple removal of the inverters—this is because a complement of the circuit inputs to a CSA yields $\overline{sum}$ and $\overline{carry}$. The HA on the other hand cannot be so easily constructed. If the inputs to an HA are complemented, the results are not $\overline{sum}$ and $\overline{carry}$. Therefore, two versions of the HA_{IP} are required—one for POS inputs which yields $\overline{sum}$ and $\overline{carry}$ and another for NEG inputs that gives sum and carry (see Fig. 3).

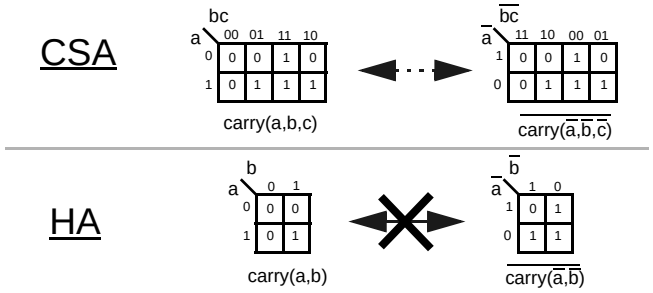

Figure 3. (a) For full adders $\overline{CSA}(\overline{a},\overline{b},\overline{c})$ = CSA(a,b,c), (b) but for half adders, $\overline{HA}(\overline{a},\overline{b})$!= HA(a,b).

3.2 Partial Product Reduction

Multiplication can be viewed as a series of shifts and adds of two numbers, the multiplicand and the multiplier. The multiplicand is shifted and added once for each non-zero bit of the multiplier. The resulting bits, called the partial product array (PPA), form a trapezoidal array where all bits in a column are added together using carry-save addition.

A greedy heuristic was proposed in [4] to construct a partial product reduction tree while minimizing logic depth. In this method, a priority queue stores the bits for each column, ordered by the largest static delay time of the bit. The algorithm proceeds on a column-by-column basis, starting at the lowest bit position, where the earliest arriving bits are added using a CSA or HA; the resulting sum bit goes into the priority queue of the current column, and the carry bit is placed in the queue of the next column.

To use inverted polarity elements, we require that all the inputs to a gate be of the same polarity, either POS or NEG. In array multipliers, this can be achieved fairly easily, since each logic level of adders can be of opposite polarities. In Wallace trees however, some adders' inputs come from signals of different logic levels (see Fig. 4). To create equal-polarity inputs, one must track bit polarity and in some cases, inverters must be provided to complement input bits.

3.3 Inverse Polarity Algorithm

To assemble the inverse polarity multiplier, we provide two priority queues for each column, one for POS and one for NEG bits. Bits of the same polarity are selected from the queue with the lowest delay bit. When nearly all the bits in a column are consumed, bits from both queues may be mixed, using inverters to normalize all bits to the same polarity. The basic algorithm appears in Fig. 5.

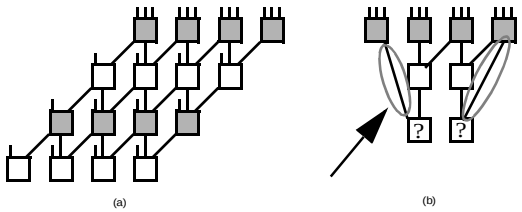

Figure 4. Inverse polarity - (a) arrays have regular alternating structure. (b) Wallace trees have connections which "skip" a level, causing polarity conflicts at subsequent adder inputs.

Since this greedy assembly algorithm uses the lowest delay bits each time it instantiates an adder, the procedure minimizes the growth of the maximum delay per column. The stopping condition is the only point where extra inverters are inserted, for the sole purpose of setting equal the input polarities of an adder. Once the partial product array has been reduced, two bits are present at each bit position. Note that the polarities of the bits may well be mixed, *i.e.*, at any given column, the final two bits may be both POS, both NEG, or one POS and one NEG. At this point, a final adder is created to generate the final result.

4. Results

A simple placement tool was written which creates a layout in two phases: 1) simulated annealing is used to group elements in the Wallace tree which have high connectivity, 2) final adder blocks are arranged using a procedural placement technique. Estimates of the footprint required for each circuit element were calculated for the MOSIS HP 0.5µm CMOS technology. Interconnect net length was calculated using a Steiner tree construction for multipoint nets with Manhattan distances for each segment (0.165fF/µm cap. to ground.) Delays are calculated using static timing analysis based on an HSPICE characterization of cells. Power is calculated using Star-sim from Avant! Corp., which allows fast power computation with high accuracy.

Tables 1 and 2 compare energy per operation between conventional vs. inverse polarity multipliers, with two different final adders. In all cases, minimum size devices were used to minimize power consumption. Results clearly indicate a power advantage for inverse polarity multipliers. A more pronounced advantage is seen in larger multipliers with carry-select adders; these have the greatest number of adder circuits, so reduced transistor count is most beneficial in these cases. A potential source of parasitic power dissipation in inverse polarity circuits is increased short circuit (totem pole) current due to more slowly falling/rising inputs, which result from the inverse polarity optimization. Detailed simulations found this effect to be insignificant.

```
Procedure TILE(column i) {
  /* put bits into queues POS_i or NEG_i; */
  while( (#bits{POS_i} + #bits{NEG_i}) > 4 ) {
     /* Pick queue to work on */
     if (earliest{POS_i} < earliest {NEG_i})
       choose_Q_i = POS_i;
     else choose_Q_i = NEG_i;

     Instantiate IP adder {choose_Q_i}
       add earliest bits of choose_Q_i;
       put output bits on ~choose_Q_i~ and ~choose_Q_i+1~;
  }
   STOP( );
}

Procedure STOP( ){
  switch (#bits{POS_i} + #bits{NEG_i}):
     case 3:
          if (#bits{POS} >= 2)
          -> use HA_IP on POS bits;
          if (#bits{NEG} >= 2)
          -> use HA_IP on NEG bits;
     case 4:
          if (#bits{POS} >= 3 )
          ->use CSA_IP on POS bits;
          if (#bits{NEG} >= 3 )
          ->use CSA_IP on NEG bits;
          if (#bits{POS} == 2 and #bits{NEG} == 2)
          -> find latest bit (lt_bit = POS/NEG)
          -> use CSA_IP on bits of type ~lt~_bit
              (use inverters to equalize inputs);
}
```

Figure 5. Basic inverted polarity CSA tiling algorithm.

Table 1: Energy / operation for 8 bit multipliers

	Conventional	Inverse Polarity	Power Reduction
Ripple	3.78e-11 J.	3.32e-11 J.	12.2%
Carry Select	4.93e-11 J.	4.63e-11 J.	6.1%

Table 2: Energy / operation for 16 bit multipliers

	Conventional	Inverse Polarity	Power Reduction
Ripple	4.27e-9 J.	3.49e-9 J.	18.3%
Carry Select	5.45e-9 J.	4.13e-9 J.	24.2%

Table 3: Delay, area, wire cap. for 16 bit multipliers

	WT-Ripple	IP WT -Ripple	WT -Carry Sel.	IP WT-Carry Sel.
Delay	38.9 ns	40.0 ns	35.0 ns	36.9 ns
Area	8586 μm^2	7368 μm^2	9804 μm^2	8576 μm^2
Wire Cap.	11,872 fF	11,234 fF	13,290 fF	12,059 fF

Delay, area and interconnect characteristics are shown in Table 3. Inverse polarity circuits are slightly slower than conventional circuits by 3% - 6%. Delay penalties can be eliminated by judicious use of carry-save adders: conventional adders for the critical path and inverse polarity adders for non-critical paths. The delay penalty is entirely due to interconnect capacitance of the Wallace tree; further refinement of the Wallace tree layout will also improve the performance of inverse polarity designs. Table 3 also shows area and interconnect capacitance numbers for 16-bit multipliers. As expected, area-of-implementation is smaller. Furthermore, total interconnect capacitance was less for inverse polarity designs, which is due to smaller area of implementation; closer components require less wiring.

5. Acknowledgements

This work was funded by DARPA under contract ADAAL 01-95-K-3527 and the NSF under contract MIP-9408457.

6. References

[1] B. Ackland, C.J. Nicol, “High Performance DSPs - What’s Hot and What’s Not?” ISLPED, 1998, pp. 1-6.

[2] C.S. Wallace, “Suggestions for a Fast Multiplier,” IEE Trans. Electronic Computers, EC-13, 1964, pp. 14-17.

[3] N. Weste and K.Eschraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1988.

[4] J. Fadavi-Ardekani, “M x N Booth Enc. Multiplier ...,” IEEE Tran. VLSI, Vol. 1, No. 2, June 1993, pp. 120-125.