Optimal Designs for Multipliers and Multiply-Accumulators

Paul F. Stelling[1]
Dept. of Computer Science
University of California at Davis
Davis, CA 95616
Email: stelling@cs.ucdavis.edu

Vojin G. Oklobdzija
Dept. of Electrical and Computer Engineering
University of California at Davis
Davis, CA 95616
Email: vojin@ece.ucdavis.edu

Keywords: Parallel Multiplier, Multiply-accumulate, Final Adder, Algorithms, VLSI circuits.

ABSTRACT

Multiply and Multiply-Accumulate (MAC) are important and expensive operations. They are frequently used in general computing and are especially critical to the performance of Digital Signal Processing and video/graphics applications. As a result, any improvement in the delay for performing these operations could have a positive impact on clock speed, instruction time, and processor performance. We show how the performance of the parallel multiplier can be improved by customizing the final adder to the input delays of the signals from the Partial Product Reduction Tree (PPRT), as opposed to using a more traditional design optimized for inputs of uniform delay.

We present strategies for designing optimal Hybrid Adders for the specified profiles that include Conditional Sum blocks. We use these strategies to develop a Hybrid Adder for a 32-bit parallel multiplier whose final adder input delays correspond to the output delays of an optimal TDM PPRT. The resulting adder improves by over 20% the incremental delay of a standard Conditional Sum adder.

Finally, we use an additional innovation and apply our results to multiply-accumulators, giving a multiply-accumulator design that is (for most sizes) as fast as multipliers of the same size. Thus a single (optimal multiply-accumulate) circuit can be used for both operations without delay penalty, allowing MAC to be efficiently and effectively implemented as an instruction in RISC CPUs.

1. Introduction.

The problem of designing improved multipliers has gained a lot of attention recently. Here we examine the problem of designing optimal final adders for parallel multipliers and show how to apply parallel multiplier design approaches to derive optimal multiply-accumulator (MAC) designs.

Parallel multipliers work in a manner similar to that used in long multiplication. First the partial products of the factors are generated, then they are combined using a Partial Product Reduction Tree (PPRT) to give two bits per column (i.e., two (2n-1) bit numbers), which are added in a Final Adder (FA) (see Figure 1).

The problem of constructing fast and efficient PPRTs has been previously addressed [1,2,6,8,13]. In this paper we address the design of Final Adders using optimal Hybrid Adders that incorporate conditional sum blocks. Finally, we show how an additional innovation can be used to incorporate our multiplier results into the design of MACs, yielding circuits that are (for most sizes) as fast as multiplier circuits.

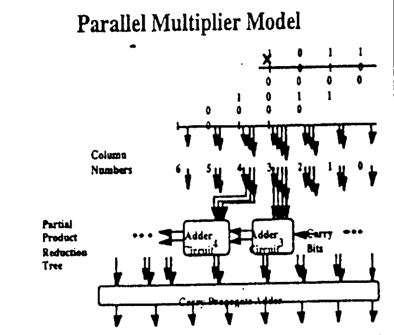

Figure 1: *Logical representation of a parallel multiplier.*

The problem of designing optimal adders when all input bits arrive at the same time is one that has been well studied [12]. A related problem that has recently gained attention [5,9,10] is the design and implementation of efficient adder circuits when the bit arrival times are arbitrary (but known in advance), as in the case of the FA for a parallel multiplier. The input profile that we use here is the output profile of the optimal TDM PPRT for a 32-bit parallel as described by Martel, et al [2,3], and shown in Figure 2. (Our approach works for more general profiles, but we limit ourselves here due to space considerations.)

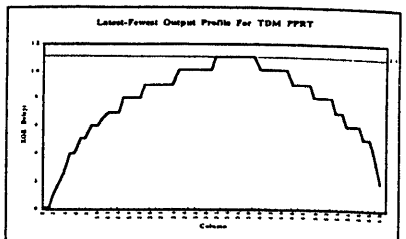
Figure 2: *The output profile of th optimal TDM PPRT for a 32-bit multiplier. There is only one outpu bit for position 0, and two output bits for positions 1 through 62.*

The usual practice has been to add the last two rows of partial products using as the FA one of the fast schemes such as Carry Lookahead (CLA) or Conditional Sum (CSA). This concept was first challenged by Oklobdzija in [5], where it was shown that under non-equal signal arrival profiles some of the commonly known fast schemes for addition do not perform well. In fact, the optimal FA is really a "hybrid" consisting of a number of blocks that can encompass several different types of adders. Such adders were introduced and analyzed by Stelling and Oklobdzija in [9,10], where they developed optimal adders using Ripple-Carry, Carry-Skip, and Carry-Select blocks under a simple timing model. Here we extend that model to include the use of Conditional Sum blocks, achieving over 20% improvement in the incremental delay of the Symmetric Conditional Sum adder.

2. Gate Delays and Conditional Sum Blocks.

As in [2,3,7], we calculate delays by normalizing relative to the time delay of an XOR gate, based on the delays in 1-micron array-based logic [14]. These delays vary with the fan-out of the gate, and are shown in Table 1. (Note that we use this convention for convenience, and our results do not rely on it. Also, the delays in [14] are provided only for output line counts that are powers of 2, the numbers in parentheses represent the numbers of output lines for which we used the given delay values.)

All n-bit adder schemes can be characterized as being made up of blocks that each take as input a carry-in bit (except possibly the first block) and some input bits corresponding to the k columns in the block, and generate sum bits for the block and a carry-out value that is propagated in some fashion to appropriate blocks with more significant bits. For such designs as Ripple-Carry, Carry-Skip, and Carry-Lookahead the column input bits are the initial inputs and/or signals generated from them (e.g., carry generate and carry propagate signals).

Carry-Select blocks take as input a carry-in bit, alternative sum bits, and alternative carry-out bits. (The alternative bits have the appropriate values assuming that the carry-in bit has value 0 or value 1.) The Carry-Select block then uses the carry-in signal to select for output the appropriate sum bit and a carry-out bit signals. (Note that this selection can be done using either standard non-inverting multiplexors or inverting multiplexors. We use multiplexor to refer to both types.)

Conditional Sum blocks are essentially nested Carry-Select blocks where the smallest blocks consist of gate combinations that generate the sum and carry-out signals for the column inputs given each of the possible carry-in values (0 and 1). Given two input values A and B, the needed outputs can be generated as follows (note that each signal is needed only in one of complemented/uncomplemented form):

- **Carry-in = 0:** Sum: A XOR B in 1 XOR delay;

Device	Input	Number of Output Lines					
	Line(s)	1	2	4	8 (6-10)	16 (11-18)	32 (19+)
Inv Mux	S	0.5	0.5	0.5	0.75	1	1.25
	A,B	0.875	0.875	1	1	1	1.125
Mux	S	1	1	1	1	1.125	1.5
	A,B	0.75	0.875	0.875	1	1	1.125
NAND		0.5	0.5	0.5	0.625	0.875	1.125
NOR		0.5	0.5	0.625	0.75	1	1.75
NOT		0.25					

Table 1: *Input to output (equivalentXOR) delays for devices by number of output lines.*

Complement of sum: A NXOR B in 1.25 XOR delays; Carry-out: A AND B in 0.75 XOR delays; and Complement of carry-out: A NAND B in 0.5 XOR delays.

- **Carry-in = 1:** Sum: A NXOR B in 1.25 XOR delays; Complement of sum: A XOR B in 1 XOR delay; Carry-out: A OR B in 0.75 XOR delays; Complement of carry-out: A NOR B in 0.5 XOR delays.

Thus it takes 1.25 XOR delays to generate either the sum or the complemented sum values for both carry-in values. Since the only sum signals whose polarity (complemented or non-complemented) matters are the final output sum values, we design our Conditional Sum blocks using the faster of standard multiplexors and inverting multiplexors. If the number of inverting multiplexors applied to the sum signal is even (odd), then we generate and use the un-complemented (complemented) sum signal to get the correct ouput. Note that gates that generate (or select) sum signals (or complemented sum signals) always have a single output line (to an enclosing block or sub-block, or to the final output).

Carry-Select sub-blocks are of two types: those whose carry-out signal is *propagated* to an enclosing Carry-Select block, and whose carry-out signals are *cascaded* to a subsequent block. For cascaded carry-outs, the complemented signals can be used in place of non-complemented signals by merely switching the A and B inputs to the multiplexor, so that the complement of the carry-out can always be generated and used, regardless of the number of times it will be propagated.

3. Adder Delay Analysis.

We now address the use of Conditional Sum blocks in the FA. We generalize the notion of a Conditional Sum block to that of a Carry-Select block that encloses (contains) a number of Carry-Select sub-blocks (the lowest-level block need not be for a single bit position as in the traditional Conditional Sum adder). We denote by $B_{i:j}$ the j^{th} sub-block enclosed by B_i, where the nesting can be done to arbitrary depth.

The optimal Conditional Sum adder for the uniform input profile is one which cascades (log n(carries, with the k^{th} carry selecting on 2^k positions, as in the case where the n bits to be added are subdivided into groups of $2^{\log(n-1)}$ and n-$2^{\log(n-1)}$. Slightly abusing definitions, we call this the *Symmetric Conditional Sum Adder*.

Based on the above delays, we modelled the delays of a Symmetric Conditional Sum Adder for adding two 62-bit numbers (the size of the inputs to the final adder of a 32-bit multiplier). Figure 3 shows the delays of that adder when used to add two numbers whose signals are all available at time 0 in total time equivalent to 6.375 XOR delays. Figure 4 shows the output delays of the same adder when applied to the profile of , giving a maximum output delay of 17.375 XOR delays. Thus when applied to the profile the latest output is the same as if all of the input signals arrived at the same time as the latest signal, even though many of the signals arrive significantly earlier.

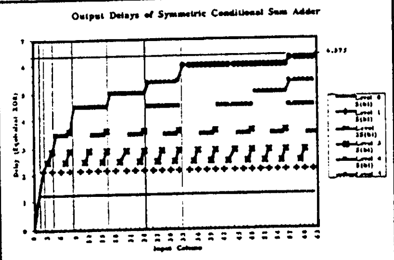

Figure 3: *The delays of the Symmetric Conditional Sum adder when applied to uniform delay inputs.*

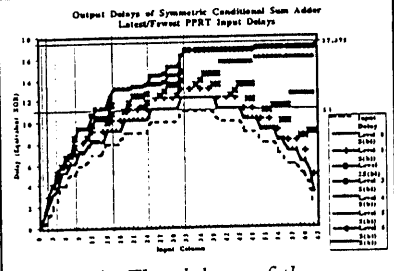

Figure 4: *The delays of the Symmetric Conditional Sum adder when applied to the 32-bit optimal TDM PPRT output profile.*

4. Optimal Hybrid Adder Delays.

We now examine the problem of designing an optimal Hybrid Adder for the profile of using Ripple-Carry, Carry-Skip, and arbitrarily nested Carry-Select and Conditional Sum blocks. We first derive a lower bound on the delay of such an adder, and then show that the bound can be achieved.

4.1. A lower bound.

We use the optimallity of symmetric Conditional Sum adders for uniform input arrival profiles to derive a lower bound on the adder delay. First we observe that there are eight columns (numbers 32 through 39) with the maximum delay of 11 XORs. Given that Conditional Sum adders are faster than Ripple-Carry, Carry-Skip, and Carry-Lookahead adders given a uniform input profile, we use a Symmetric Conditional Sum block on those positions. It will remain to combine that block with adder blocks for the remaining columns. There must be at least two such blocks, which we will call B_0 and B_2, over columns 1-31 and columns 40-62, respectively (we call the block over columns 32-39 B_1). The fastest way these could be combined is by Carry Selection; either by using cascading carry selection from B_0 to B_1 to B_2; or by cascading carry selection from B_0 to a larger block consisting of B_1 and B_2, with those two combined by cascading carry select. Assuming fan-out 1, the (alternative) sum and carry outputs for B_1 will be generated at times 14.625 and 14.375, respectively. Suppose that the carry-out signal from B_0 can be generated arbitrarily early (at time t_0), as can the alternative sum and carry-out output signals for B_2 (at time t_2). Then under the first approach the final output sum bits for columns 32 through 39 will be available at time $\max(t_0+1.0,\ 14.625+0.75) = 15.375$, cascading carry-out bit from B_1 would be available at time $\max(t_0+1.25,\ 14.375+1.125) = 15.5$, and the sum bits and carry-out bit from the columns in B_2 would be available at time $\max(15.5+0.5,\ t_2+0.875) = 16.0$. (This bound is achieved so long as t_0 (14.25 and t_2 (15.125.) Under the second approach the cascading carry-out bits from B_1 would arrive at time 14.75 (due to fan-out of 24) and finish selecting the alternative B_2 sum and carry bits at time $\max(14.75+0.5,\ t_2+.875)=15.25$, so that the final sum bits for B_1 would be available at time $\max(t_0+1.0,\ 14.625+.75)$ (16.0, and the final output sum bits for B_2 would be available at time $\max(t_0+1.0,\ 15.25+0.75) = 16.0$. (This bound is achieved so long as t_0 (15.0 and t_2 (14.375.) Thus we have a lower bound on the total delay of 16.0, achievable by both approaches, but with differing bounds on the delays of the output signals from B_0 and B_2.

4.2. Achieving the lower bound.

To achieve the lower bound established above we must meet the t_0 and t_2 bounds above. We now describe how we achieve that goal.

First we note that a Ripple-Carry Block will suffice for the first 5 positions, since the delay of the carry-out from that block will not impact the overall adder delay [9,10]. Next, we note that for any two bit positions i and $i+1$ where the input delays are j and $j+1$, we can use a (simpler) Ripple-Carry sub-block and achieve the same delays as with a Symmetric Conditional Sum block. Finally, we note that in general it is advantageous with nested Carry-Skip blocks to cascade the carry-out signals whenever the multiplexor output delays resulting from the (cascaded) select line would be (the delays from the (sum and carry-out) data lines. This will result in both faster and simpler (fewer devices) adder designs (due to shallower nesting than would be needed with propagation).

We use these principles in our design, breaking columns 1 through 30 into 10 blocks, over bit positions 1-5, 6-7, 8-9, 10-11, 12-13, 14-15, 16-19, 20-23, 24-26, and 27-30. The first of these blocks is a Ripple-Carry block, and the remainder are combination Ripple-Carry and Conditional Sum blocks, with the carry-outs cascaded from each block to the next. Using this approach the last carry-out from B_0 is generated at time 13.875 (<14.25) if the first approach is used, and at time 14.125 (< 15.0) if the second approach is used (due to greater fan-out).

Next we consider columns 40 through 62. We achieve the t_2 bounds by using multi-level nested Carry-Skip sub-blocks, with the first level division being ((40-43), (44-62)), the next being ((44-50), (51-62)), etc. When it is possible to do so without making the delays too large, the underlying blocks are simplified, for example, a Ripple-Carry design is used as the base adder for the sub-block over columns 58-62. In this manner we generate the needed signals by time 13.75 (<14.375<15.125).

The latest signal from the adder is thus generated at time 16.0 by both approaches, giving over 20% improvement in the incremental delay over the traditional Symmetric Conditional Sum adder (from 6.375 xors to 5.0 xors). The delays of the adder using the first approach is shown in Figure 5. Figure 6 compares the output delay profiles of the Symetric Conditional Sum adder and the Hybrid Adder using the first approach. (A block diagram of the resulting Final Adder is omitted due to space limitations.)

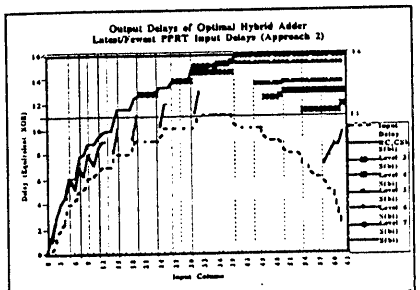

Figure 5: *The delays of the optimal Hybrid Adder for the optimal 32-bit TDM PPRT output profile using Approach 2. The solid vertical lines denote the block divisions, and the dashed vertical lines denote the divisions of major sub-blocks.*

5. Multiply-Accumulator Design

We can apply the design approach described above to MACs (which take as input two n-bit factors and a $2n$-bit addend), with the addition of one further inovation [11]. We recognize that we can include the addend in with the partial products as an input to the Partial Product Reduction Tree (PPRT) as shown in Figure 7. The number of inputs to each column then coincides with the number of inputs to the next larger column in the PPRT for an (n+1)-bit multiplier. Thus the optimal PPRT and Final Adder circuits for an (n+1)-bit parallel multiplier can be used for an n-bit MAC. But for most sizes of multipliers n and (n+1) the total delay is the same. Our research shows that this result can be achieved for most sizes of MACs, including the common case of 16-bit factors and a 32-bit addend (also 32-bit factors and 64-bit addend). Thus MAC can be implemented in a circuit with the same delay as a multiplier. (The delay graphs and circuit diagrams for those cases are omitted due to space limitations).

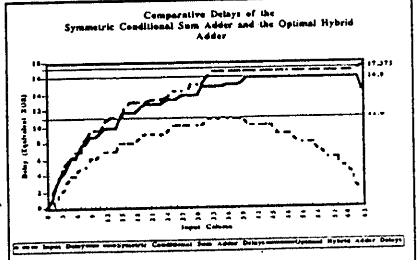

Figure 6: *The comparitive delays of the Symmetric Conditional Sum adder and the optimal Hybrid Adder (Approach 1) when applied to the optimal 32-bit TDM PPRT output profile.*

Thus, multiply-accumulate circuitry can be used in most cases for both operations with no delay penalty, both reducing the complexity of the ALU and allowing the MAC instruction to be implemented without penalty in any ALU that includes multiplication.

6. Conclusions.

We have shown how to design an Hybrid Adder made up of blocks of Ripple-Carry and Conditional Sum blocks. Our method has produced a structure which we showed to be optimal for the case of a 32x32-bit multiplier. The improvement in incremental delay is estimated to be >20% over commonly used symetric Conditional Sum schemes. This improvement translates to >7.9% reduction in total multiplication time. We have also shown how to combine these results with the innovation of including the MAC addend with the partial products as an input to the PPRT to create optimal multiply-accumulators that are as fast as multipliers for

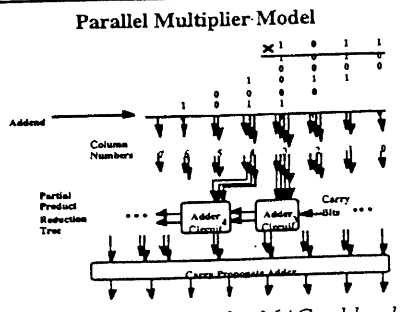

Figure 7: *Including the MAC addend with the partial products in a parallel multiplier.*

sizes of interest.

We are presently working on extending our results to find the simplest (fewest gates) adders with the optimal delay. We are also working on applying our results to other applications resulting in non-uniform input delays. With our approach we have determined that we can build faster adders using simpler circuitry, but we have not yet completed optimal designs.

References

[1] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design*, John Wiley and Sons, 1979.

[2] Charles Martel, Vojin Oklobdzija, R. Ravi, and Paul F. Stelling, "Design Strategies for Optimal Multiplier Circuits", *Proceedings of the 12th Symposium on Computer Arithmetic* , pp. 42-49 (1995).

[3] Charles Martel, Paul F. Stelling, Vojin Oklobdzija, and R. Ravi, "Optimal Circuits for Parallel Multipliers", in press, *IEEE Transacton on Computers*, 1996.

[4] Vojin G. Oklobdzija and Earl R. Barnes, "Some Optimal Schemes for ALU Implementation in VLSI Technology", *Proceedings of the 7th Symposium on Computer Arithmetic*, 1985.

[5] Vojin G. Oklobdzija, "Design and Analysis of Fast Carry-Propagate Adder Under Non-Equal Input Signal Arrival Profile", *Proceedings of the 28th Asilomar Conference on Signals, Systems, and Computers*, 1994.

[6] Vojin G. Oklobdzija and David Villeger, "Improving Multiplier Design by Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology", in press, *IEEE Transactions on VLSI*, 1995.

[7] Vojin G. Oklobdzija, David Villeger, Simon S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Transaction on Computers*, March 1996.

[8] P. Song and G. De Michelli, ``Circuit and architecture trade-offs for high speed multiplication," *IEEE Journal of Solid State Circuits*, 26, 1991.

[9] Paul F. Stelling and Vojin G. Oklobdzija, "Design Strategies for the Final Adder in a Parallel Multiplier", *Conference Record of the 29th Asilomar Conference on Signals, Systems, and Computers*, pp.591-595, 1995.

[10] Paul F. Stelling and Vojin G. Oklobdzija, "Design Strategies for Optimal Hybrid Final Adders in a Parallel Multiplier", *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, December 1996.

[11] Paul F. Stelling and Vojin G. Oklobdzija, "Implementing Multiply-Accumulate Operation in Multiplication Time", *in press, Proceedings of the 13th Symposium on Computer Arithmetic.*

[12] Earl E. Swartzlander, ed., *Computer Arithmetic Vol. 1 and 2*, IEEE Computer Society Press, (1990).

[13] C. S. Wallace, ``A Suggestion for a Fast Multiplier," *IEEE Transaction on Computers*, EC 13, pp.14-17, 1964.

[14] *1.0-Micron Array-Based Products Databook*, LSI Logic Corporation, September 1991.

1(Research supported by NSF grants CCR-94-03651 and CCR-91-03937.