An Asynchronous Multiplier

Brenda Luderman and Alexander Albicki

Department of Electrical Engineering
University of Rochester, Rochester, N.Y. 14627

Abstract

This paper discusses the design and performance of an 8-bit asynchronous multiplier. Self-timed adders with carry completion signals were implemented to create the asynchronous function of the multiplier. The technique of using bi-directional adders is incorporated in the design. One objective of this design was to estimate the longest multiplication time for a 2μ CMOS implementation of this 8-bit multiplier.

I. Introduction

Increasing the packing density of synchronous VLSI systems creates problems with clock distribution and clock skew. Utilizing self-timed operands in VLSI design can reduce these problems. However, it is necessary to determine whether or not the additional logic used in asynchronous operands significantly increases the system computational time. In an asynchronous multiplier, the only concern is the longest multiplication time. If the longest multiplication time is comparable to the computational time of a synchronous multiplier, then the average multiplication time for an asynchronous multiplier will be less than that of a synchronous multiplier.

Bi-directional adders provide a computational advantage to an asynchronous multiplier as they are capable of performing two separate multiplications during one multiplication time. An analysis of the signal propagation through a uni-directional multiplier is described first to illustrate the usefulness of bi-directional adders [1].

Consider the array of full adders for the 4-bit uni-directional multiplier shown in Figure 1. Each adder has three inputs, a, b, and Cin and two outputs, S and Cout. The b inputs to the first row of adders from right to left are the bit products m0q0, m1q0, m2q0, and m3q0 while the a inputs are all zero. The a inputs to the second row of adders are the sums generated from the adder stage immediately above it. The b inputs to the second row are the bit products, m0q1, m1q1, m2q1, and m3q1. The b and a inputs to the following rows are shown in Figure 1.

The multiplication time for this uni-directional multiplier can be divided into two time intervals, T1 and T2. During the interval T1, the right half of the multiplier computes the products P0-P3 while the shaded left half remains unused. During the interval T2, the carries generated from the right half are passed to the left. The left half of the multiplier computes the products P4-P7, while the right half of the multiplier is dormant.

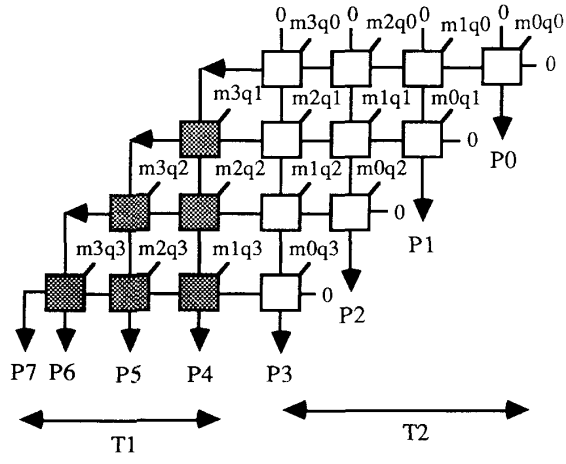

Figure 1. A 4-bit uni-directional multiplier.

To improve the efficiency of the multiplier, the idle portions can be used to initiate another multiplication sequence with the aid of bi-directional switches [1]. The symbol for a bi-directional full adder is shown in Figure 2a. When the adder operates in the forward direction the symbol in Figure 2b is used while the reverse direction is denoted by the symbol shown in Figure 2c.

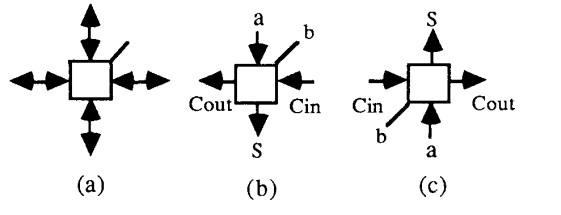

Figure 2. Symbols for a bi-directional full adder.

0-8186-2610-0/92 $3.00 © 1992 IEEE

When an additional column of adders is inserted in the left half of the array, a functionally symmetric multiplier is formed. This structure is shown in Figure 3 where the shaded adders represent the once idle portion of a uni-directional multiplier during T1. Using bi-directional switches, both sums P0-P3 and P0'-P3' are computed during the time interval T1. The carries generated from the last columns computing P3 and P3' are retained in latches when the multiplier is cleared.

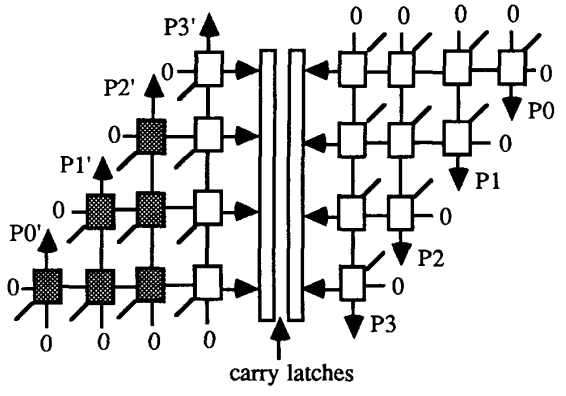

Figure 3. Symmetric multiplier during T1.

During the interval T2, the latched carries from the right, are passed to the left and the latched carries from the left, are passed to the right as shown in Figure 4. At the completion of T2, the sums P4-P7 and P4'-P7' are available.

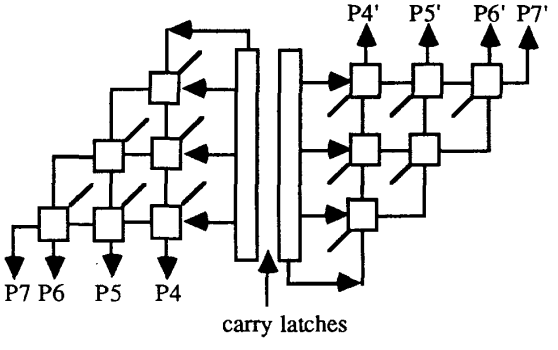

Figure 4. Symmetric multiplier during T2.

When the same multiplication is performed on both halve's of the multiplier, concurrent error checking is possible [1]. When different multiplications are performed in each half, two multiplications occur during one multiplication time.

The remaining portions of this paper discuss the design of the asynchronous multiplier, the multiplication control sequence, and an estimate of the maximum multiplication time.

II. Multiplier Design: Control Signals

The self-timed multiplier designed has the two principle blocks shown in Figure 5, a control block and an array block. The control block has two control signals to communicate the status of the multiplier externally, an input **REQ** requests or initiates the multiplication sequence, and an output **ACK** acknowledges the completion of the multiplication sequence. The remaining control signals, **R** and **F**, are for internal use and are dependent on the intermediate status of the multiplication sequence. The first half of the multiplication process is completed when the latch completion signal, **LC1**, is generated from the last sum latched. The asynchronous adders are then cleared using the reset control signal, **R**, and the bi-directional switches are changed using the forward control signal, **F**.

The multiplier array contains an 8 x 9 array of bi-directional asynchronous adders, two 8 bit intermediate carry latches, and four 8 bit product latches. The inputs to the adders are the qm products. Each adder has bi-directional logic dependent on the forward control signal, **F**, and reset enable/disable logic dependent on the reset control siganl, **R**. At the completion of the interval T1, a carry completion signal, **CC1R**, is generated from last adder operating in the reverse direction. This signal is used to latch the intermediate carries and the sums. At the completion of T2, the carry completion signal, **CC2R**, is used to latch the remaining sums. After the last sum is stored, the latch completion signal, **LC2**, is generated.

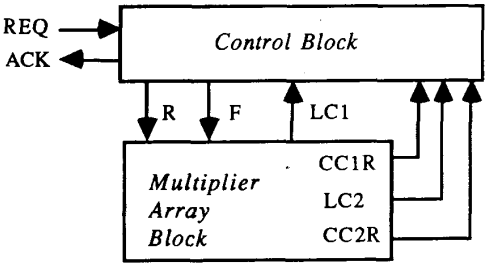

Figure 5. Block diagram for asynchronous multiplier.

III. Adder Design

The design of an asynchronous adder requires two carry-in and two carry-out signals [2]. Using two carry-out signals allows the completion of addition to be detected by ORing these two signals. The asynchronous adder can receive and generate either a 1-carry, a 0-carry, or neither but not both. The 0-carry-in is denoted as C0 and the 1-carry-in as C1. Similarly, the 0-carry-out is C0' and the 1-carry-out is C1'. Table I shows a truth table for an asynchronous adder where A and B are the normal inputs for a 1-bit full adder.

Table I

Truth table for an asynchronous adder.

A	B	C1	C0	S	C1'	C0'
0	0	0	1	0	0	1
0	0	1	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	0	1	0
1	0	0	1	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	1	1	0

A boolean equation describing the sum in terms of A, B, C1, and C0 can be given as the sum of products,

$$S = C1\overline{C0}(\overline{A}\,\overline{B} + AB) + \overline{C1}C0(\overline{A}B + A\overline{B}). \quad (1.a)$$

Following the derivation given in [2] and [3], an expression for C1' is determined in terms of A, B, and C1 only.

$$C1' = C1(\overline{A}B + A\overline{B}) + AB\,. \quad (1.b)$$

Similarly, an expression for C0' is determined in terms of A, B, and C0 only.

$$C0' = C0(\overline{A}B + A\overline{B}) + \overline{A}\,\overline{B}\,. \quad (1.c)$$

These three equations describe the function of an asynchronous adder.

There exists a discrepancy between the functional representation of (1.b) and (1.c), and the actual definition of the asynchronous adder. That is, the adder should be able to receive or generate a 1-carry, a 0-carry, or neither, but specifically, the neither carry is not specified by these equations. This problem is due to a unique initialization condition and is explained in the following example.

Consider the 4-bit asynchronous adder shown in Figure 6, the overall carry completion signal, CC, is the product of the carry completion signals from each stage. The individual carry completion signal from each stage is the sum of that stage's 0-carry-out and 1-carry-out. It is assumed that the addition completion of the fourth stage is generated after the third stage and the addition completion of the third stage is generated after the second and so forth. Ideally, the asynchronous 4-bit adder will report the addition completion, CC, after the addition completion of the fourth stage is generated.

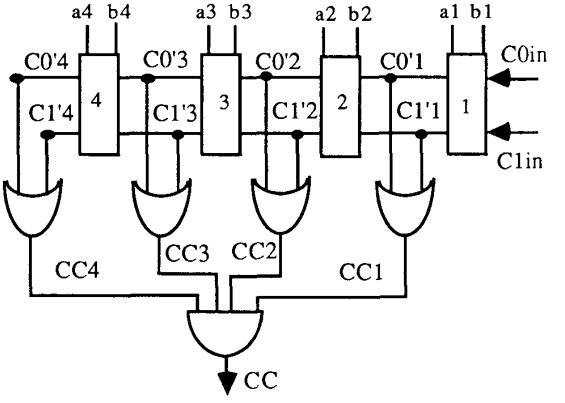

Figure 6. A 4-bit asynchronous adder.

However, it is noted that there exists a specific carry-input state in equations (1.b) and (1.c), where an incorrect carry completion signal, CC, will occur. Assume that all four adders are cleared such that a logic zero is present at all inputs and outputs. From equations (1.b) and (1.c), the carries and individual carry completion signals are determined when all the a and b inputs simultaneously are a logic 1, the 0-carry-input is 1 at the first stage and the 1-carry-input is 0. Now, let Δ represent the delay through the adder and the OR gate. From (1.d), after a time Δ, all the 1-carry-outputs are generated simultaneously. Also, the individual carry completion siganls are generated simultaneously after 2Δ. Since each stage's carry completion signal is generated, the overall completion signal, CC, is also generated. However, the effect of the carry-outputs from the first stage have not yet propagated through the second stage and subsequent stages. Since the carry-outputs have not propagated thoroughly, a false CC signal can be generated. To avoid this false state, a cleared state must be included to the asynchronous adder. Initializing the adder is accomplished by redefining the functional dependencies of C0' and C1'.

Instead of expressing C1' in terms of A, B and C1, C1' is determined in terms of A, B, C1, and C0.

$$C1' = C1\overline{C0}(\overline{A}B + A\overline{B}) + AB(\overline{C1}C0 + C1\overline{C0}). \quad (2.a)$$

Equation (2.a) can be simplified. Since condition C1 = 1 and C0 = 1 is prohibited and is considered a don't care (d) state, that is

$$[\,\overline{C1}\;C0\,]_{\text{given } C1C0 = d} = C0$$

and

$$[\,C1\;\overline{C0}\,]_{\text{given } C1C0 = d} = C1,$$

substituting, (2a) becomes

$$C1' = C1(\overline{A}B + A\overline{B}) + AB(C0 + C1). \qquad (2.b)$$

Similarly, an expression for C0' can be determined and simplified:

$$C0' = C0(\overline{A}B + A\overline{B}) + \overline{A}\,\overline{B}(C0 + C1). \qquad (2.c)$$

One more simplification can be made to equation (1.a), namely :

$$S = C1(\overline{A}\,\overline{B} + AB) + C0(\overline{A}B + A\overline{B}). \qquad (2.d)$$

Now, given equations (2.b) - (2.d), the carry completion signal, CC, is generated when the addition from the n-th stage is completed and the CC signal is the ORed sum of C0' and C1' from the n-th stage.

IV. Multiplier Array

In an n-bit multiplier, the adders are arranged in an n x n array. In contrast, Figure 7 shows half of a 4-bit multiplier. Each row after the first has an a input dependent on the sum generated from the row above. Since the first adder in each row is defined to have a 0-carry-in (C0 = 1), the adders in the rows below the first are operational when the mq products are present. This is undesirable in the design of an asynchronous multiplier since an incorrect sum will be generated from the last adder product, P3. Detecting when the correct sum is generated in the last adder is accomplished by using carry completion controls along the longest propagation path. This longest propagation path is found to be the diagonal path following the first adder in each row. Thus, carry completion control logic is included before the first adder in each row following the first. Since the 1-carry-input is 0 for the first adders, the carry completion logic controls when the 0-carry-input is generated. Using this control strategy, the carry completion signal, **CC**, is generated correctly and is used to latch the intermediate carries as well as the sums P0-P3. Sum P3 is correctly latched when the latch completion signal, **LC**, is generated.

For the second half of the multiplier array, the carry completion control is also used along the longest propagation path. However, the carry completion logic controls both the 0- and 1- carry inputs.

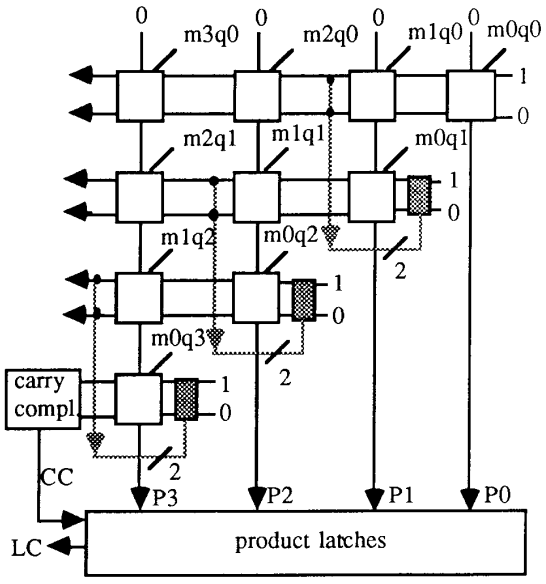

Figure 7. Half of an asynchronous multiplier with carry control logic.

V. Asynchronous Multiplication Control Sequence

Figure 8 shows a timing diagram for the control signals used in this design. An external control signal, **REQ**, initiates the multiplication sequence by disabling the reset, **R**, and the acknowledge, **ACK**, control signals. The carry completion signal, **CC1R**, is generated when the carries have propagated to the last adder found in time inverval T1. **CC1R** latches the intermediate carries and sums. Once the last sum is stored, the latch completion signal, **LC1**, is generated. **LC1** activates the reset control. The multiplier is cleared when **CC1R** makes the transition from 1 to 0. This transition activates the bi-directional control, **F**, and the second half of the multiplication sequence. The carry completion signal, **CC2R**, is generated when the carries have propagated to the last adder found in time interval T2. **CC2R** latches the remaining sums. After the last sum is stored, **LC2** is generated. **LC2** activates **ACK**, **F**, and **R** again.

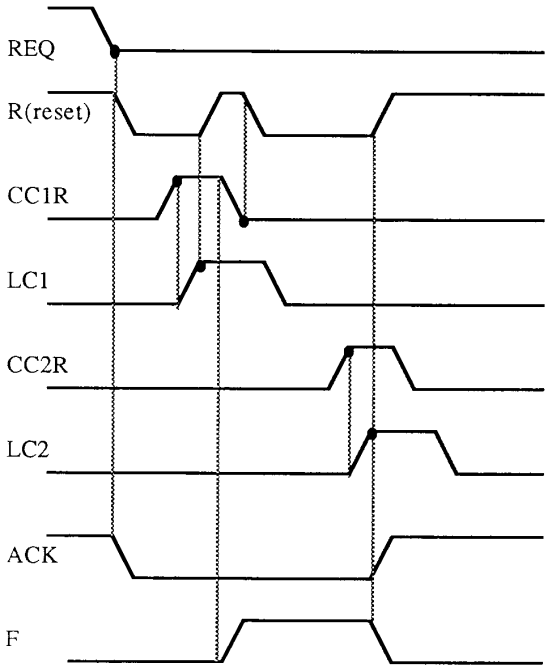

Figure 8. Timing diagram.

VI. Multiplication Time

An estimate of the multiplication time associated with the time interval T1 is made using a modified version of SPICE2G.6. The number of transistors defined in the SPICE input deck was approximately 3000. An internal memory management error occurred when this circuit was first simulated. Since SPICE2 is written to dynamically vary the number of words allocated to the job by the operating system, [4], it is believed that this 'dynamic' memory manager had problems with this circuit. This problem was eliminated when the array VALUE was dimensioned in each subroutine. Using this modified version, a simulation of half the multiplier was accomplished. The minimum size channel length used assumed to be 2 microns. From this simulation, the 0-carry output associated with the adder generating product P7 was delivered in 56ns. An estimate of the maximum multiplication time was computed using the simulation results associated with the individual control blocks. It takes 6ns for the first 36 miqi products to be present at the outputs of the B-control logic. The carry completion signal, **CC1R**, is generated in 3ns and the latch completion signal, **LC1**, is generated in 9ns. The reset control signal is generated in 15ns and the reset time for the asynchronous adder is 15ns. The time it takes the carry-outputs associated with the adder generating P15 is approximated with 56ns. The carry completion, **CC2R**, is generated in 3ns and the latch completion, **LC2**, is generated in 9ns. Finally, the acknowledge control, **ACK**, is generated in 9ns. Using all these simulated values, the estimated maximum multiplication time is 181ns. It should be mentioned that the carries generated from the last adder cell are 4ns faster than the generation of the product, P7.

VII. Conclusions

There are two ways to increase the multiplication time for this asynchronous 8-bit multiplier. First of all, decreasing the channel length will decrease the multiplication time. Although the digital logic used to design the control and array blocks has not been discussed, optimizing these blocks for maximum speed sill also increase the multiplication time.

If this multiplier is used to perform two multiplications during one time interval, then the multiplication time for one sequence would be half of 181ns or 91ns. This time is for a process in which the minimum channel length is 2 μm. This time, 91ns, can be compared with the TRW MPY-8 8 bit two's complement asynchronous multiplier of 130ns [5]. Although the multiplier designed can be used for unsigned inputs, the multiplication time will be less than the TRW MPY-8 when the sign bit is processed.

Acknowledgement

The authors of this paper would like to acknowledge Martin Slawson for his patience and effort in designing the layout for this multiplier.

References

[1] Chen, L. G. and Chen, T. H. , " Computation with Simultaneously Concurrent Error Detection Using Bi-Directional Operands", *Proc., Int'l. Conf. on Computer Design,* pp. 128-131, Boston, MA, 1989..

[2] Flores, Ivan. *The Logic of Computer Arithmetic.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1963, Chapter 4.

[3] Gschwind, Hans W. and McCluskey, Edward J., Design of Digital Computers, Springer-Verlag New York Inc., 1975, Chapter 8.

[4] Cohen, Ellis " Program Reference for SPICE2," ERL Memo No. ERL-M592, Electronics Research Laboratory, University of California, Berkeley, June, 1976.

[5] Short, Kenneth L. , *Microprocessors and Programmed Logic*, Englewood Cliffs, N.J., Prentice-Hall, Inc., 1987, Chapter 7.