Hardware Overhead Reduction of a QDI Booth Multiplier

B. Akhbari, V. Fatemi, H. Pedram, M. Naderi
Computer Engineering Department,
Amirkabir University of Technology, Tehran, Iran
bakhbari@itrc.ac.ir, {fatemi,pedram,naderi}@ce.aut.ac.ir

Abstract

This paper presents a new method to implement a multiplier using the Quasi Delay Insensitive (QDI) approach. QDI circuits allow unbounded delays on wires and gates, and require the difference among the delays in forks to be less than the delays of their terminating gates. To implement the Booth multiplier following the QDI approach, we considered Martin's method. In this method, an asynchronous circuit is considered as a set of cells that communicate through a handshaking protocol, and is synthesized from a high level definition through different levels of translation. The main problem related to the resulting circuits their considerable overhead due to the implementation of handshaking protocols. In our proposed method, the overhead is reduced 50% by separating the control and data path units. This solution increases the forks, and causes complexity in physical implementation. By applying some of the rules derived from Martin's method, the forks became locally limited to ease up the physical implementation.

1. Introduction

The two fundamental assumptions in logical circuits design are having discrete time and binary signals. In synchronous design, the property of having discrete time and the existence of a global clock pulse simplifies the design and solves the problems related to hazards, race, and instability. For asynchronous designs, the existence of binary signals is retained, but the discrete time property is weakened or totally removed. Canceling the global clock pulse makes the design more complex, but produces advantages such as avoiding the clock skew problem, reduction of consumed power, achieving average delay rather than the worst case delay, automatic adaptation to the physical conditions, and optimization of the heavily used sections [1].

According to their delay model, asynchronous design methods are grouped as: 1)Bounded delay model, 2)Micropipeline model, 3)Delay insensitive model (DI), 4)Quasi delay insensitive model (QDI), and 5)Speed independent model (SI).

The delay model points to the delay of circuit elements. In the bounded delay model, the gate delays and wire delays are considered to be limited within specified bounds. This assumption simplifies the design phase and complicates the implementation. For the SI model, wire delays are considered negligible with respect to gate delays, an unrealistic assumption for current technology. In the micropipeline model, the data path is designed following the bounded delay assumptions, while the controller design uses the DI model. Regarding the DI model, gate delays and wire delays are considered unbounded, an assumption that complicates the design phase, but simplifies the implementation. The QDI model follows the DI model with the exception that it requires the delay difference in concurrent forks to be less than the delays of their terminating gates. Martin presents a method to synthesis QDI circuits. This method starts from the high level description of the behavior, and reaches the transistor level design at the other end [2]. At the highest level, the asynchronous circuit is described in CSP (Communicating Sequential Processes) language as a set of processes that communicate through communication channels using a four phase handshaking algorithm. To transmit data asynchronously, each bit is encoded in dual rail and sent on a pair of wires. The dual rail code has three logical values: zero, one, and neutral. By applying a set of transformations, following a set of rules, and level to level translations, the high level description transforms to SLP and Production Rule (PR) forms. The PR form could be mapped directly to CMOS circuits.

One advantage of Martin's method is that it presents a well-defined method to extract data path and control circuits from the high level description, while other methods concentrate on control circuits only. Also it is possible to optimize the circuit during different levels of synthesis.

In this paper, an asynchronous QDI multiplier is designed and implemented following Martin's method. The modified Booth algorithm was used to produce partial multiply results, and the Wallace tree was the algorithm for adding up the partial results. Description at transistor level was derived using the CSP description and applying level to level translations. Utilizing the rules of this synthesis, transistor count is reduced as much as possible, and the concurrent forks are localized. Also, by taking advantage of a special structure that separates the control and data path units the handshaking overhead reduces in comparison to that of the method which uses standard handshaking cells.

Section one reviews Booth algorithm and the reduction tree structure. In section two asynchronous implementations are described. Section three is devoted to

optimizing the designed blocks. Simulation results are presented in section four, and conclusion follows.

2. Multiplication Algorithm

Multiplication methods are divided into serial and parallel categories. Serial multipliers consume small amount of chip area, and due to usage of repetitive methods they are slow. On the contrary, parallel multipliers are faster but require more chip area. Two types of parallel multipliers are tree multipliers and array multipliers. Array multipliers use ripple carry adders to sum up the partial products, thus increasing the input bits leads to slower operation. Two techniques are available to overcome this deficiency; applying Booth algorithm to reduce the number of partial products, and using Wallace reduction tree that is constructed from carry save adders. Ripple carry adder is only used to compute the final result in the last row. To speed up this adder, the interleaved cell of full adders are designed in such a way that the inverters in the carry chain (the critical path) are cancelled.

3. Implementing the asynchronous method

Two methods are available to implement the function $Y=f(X)$, where X and Y are n-bit words. In the first method, the function is designed as a set of cells that operate on a limited number of input bits, and produce limited number of output bits. The cells communicate through a 4-phase handshaking protocol. If the computation section of each cell is omitted, it transforms to a simple buffer that communicates with its adjacent cells by receiving data from left channel and writing to the right channel. (Figure 1).

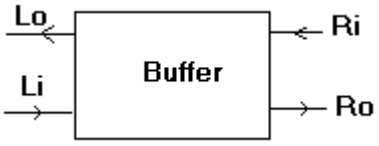

Figure 1. Asynchronous buffer.

The description of this simple buffer at SLP level follows:

$$Buffer = *[[Li]; Lo\uparrow; [\neg Li]; Lo\downarrow; [Ri]; Ro\uparrow; [\neg Ri]; Ro\downarrow]$$

Interchanging the "Return signal to zero" sections without violating the handshaking sequence will create several forms that are different from each other considering number of transitions, delay, and transistor count [3]. These widely used handshaking forms are shown below:

$$PCFB \equiv *[[\neg Ra \wedge L]; R\uparrow; La\uparrow; ([Ra]; R\downarrow), ([\neg L]; La\downarrow)]$$

$$PCHB \equiv *[[\neg Ra \wedge L]; R\uparrow; La\uparrow; [Ra]; R\downarrow; [\neg L]; La\downarrow]$$

$$WCHB \equiv *[[\neg Ra \wedge L]; R\uparrow; La\uparrow; [Ra \wedge \neg L]; R\downarrow; La\downarrow]$$

The computation section replaces signal R. Transition R↑validates output according to the defined functions, and R↓ invalidates signals following the invalidation of the inputs. The problem with this implementation is the increase in handshaking hardware in each cell. Its advantage is the possibility of pipelining the cells.

In the second method, we separate the signals related to the handshaking protocol from the computation part. $Y=f(X)$ is implemented as a collection of cells described at SLP level as $f=*\{[X]; Y\uparrow; [\neg X]; Y\downarrow\}$. That means the output of each cell goes valid as the inputs get valid, and it turns neutral when the inputs go neutral.

In order to transmit data asynchronously, the computation part is placed in a structure such as Figure 2 [4]. This structure consists of a completion detection tree (a binary tree made of c-elements) [5], input and output registers, and controller. Consequently, handshaking is done only in input and output circuits, and no exchange of control signals between internal cells is necessary. The advantage of this method is handshaking hardware reduction and forward delay improvement.

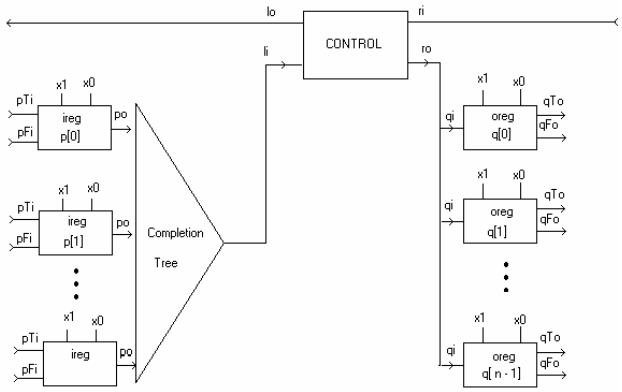

Figure 2. A structure to transfer data asynchronously.

4. Management and optimization of concurrent branches

The basic cells in this implementation are Booth encoder, specialized Booth encoder, partial product producer, full adder and half adder. Figure 3 illustrates the four basic blocks. The Booth encoder receives the sequential bits from the multiplier and specifies the operation on the multiplicand. The partial products producer multiplies the multiplicand by 1, -1, 2, -2, or 0, according to $X0$, $X1$ and s values of the multiplier. $X0$, $X1$, and S are computed as:

$X1 = (Y_{n+1}$ XOR $Y_n)$ AND $(Y_n$ XNOR $Y_{n-1})$

$X0 = Y_n$ XOR Y_{n-1}
S = NOT (Y_{n+1})

The partial product producer receives the output of the Booth encoder along with two sequential bits from the multiplier, and produces one bit of the partial product accordingly. To map the relationship f=*{[X]; Y↑;[⌐X];Y↓} to a CMOS equivalent circuit, the [X];Y↑ part is implemented in pull-down, and the [⌐X];Y↓ part is implemented in pull-up. The output signal is inversely produced, and an inverter is needed for each cell output.

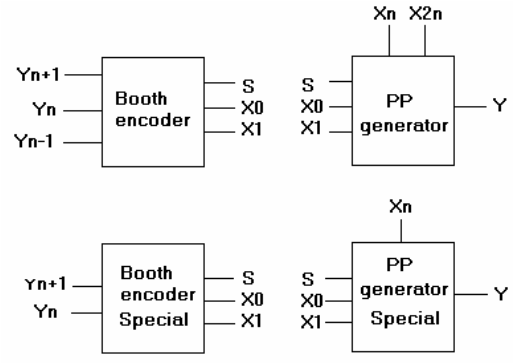

Figure 3. Multiplier basic blocks.

The circuit which is directly extracted from [X];Y↑ and [⌐X];Y↓ is not optimized, because all the inputs must be tested for valid/neutral in order to validate/neutralize the output. This leads to increased number of serial transistors in pull-down and pull-up networks.

It is noticeable that one can reduce the output validation/neutralization dependence on inputs by generating concurrent branches, hence, speed up the circuit. This technique may be used to improve speed and reduce transistor counts in basic cells [6].

As an example, the partial product producer cell has one output and five inputs in dual rail logic. Implementing [⌐X];Y↓ in pull-up requires 10 PMOS transistors in series which increases the delay for the neutralization phase. To solve this problem, testing inputs for neutralization is distributed among partial product producer cells as follows: Due to overlapping of bits in the multiplier, X_{2n} input in block n and X_n input in block n+1 are identical. Thus the neutralization test of input X_{2n} is left to the next block and non-local concurrent branches are generated between the two adjacent blocks. Also, the neutralization tests for three input bits of $X0$, $X1$, and S is distributed among three adjacent blocks that produce partial products and, non-local concurrent branches are generated for the three adjacent sequential blocks. Consequently, the numbers of pull up transistors are reduced from 10 to 4. Figure 4 displays the concurrent branches that are generated in a 4 bit multiplier,

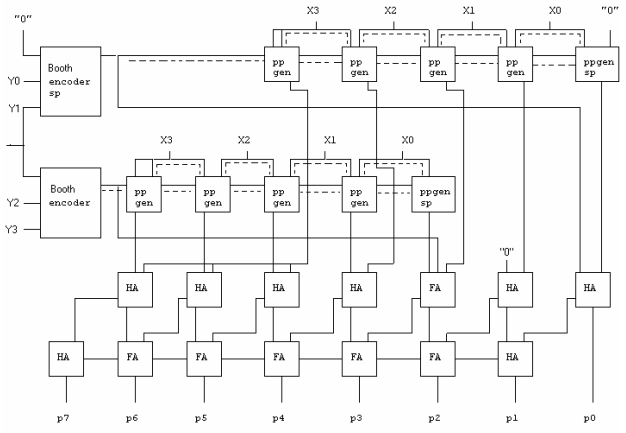

Figure 4. Concurrent branches in 4 bit multiplier.

To implement [X];Y↑ , if all the input bits contribute to evaluation of the output bit, validation of the output value indicates the validation of the inputs. Thus, the validation test of the inputs is cancelled, and only the f function remains in the pull down network. This leads to reduction of the number of serial transistors in the pull down. Due to the limited number of concurrent branches in a region, establishing the concurrency condition during physical implementation is possible with modern technology and it is not a difficult task.

5. Test conditions and simulation results

The 4, 8, 16, and 32 bit multipliers were implemented using the .8μm CMOS technology. The circuits were simulated in PSPICE at sixth level. In order to compute the average delay, a set of random vectors corresponding to the number of input bits were applied as inputs and the delays were averaged. The forward delay and the number of transistors in the computational section are summarized in table1.

Multiplier	Avg Forward Latency (ns)	Transistor Count
4 Bit	6.6	932
8 Bit	9	3198
16 Bit	11	13668
32 Bit	13.5	50806

Table 1. Forward delays and transistor counts in the computational unit.

Cycle time the sum of forward delay, neutralization phase delay, and handshaking delay. The following shows

the computation. The results and the transistor counts are summarized in Table 2.

$$CycleTime = Tv_{Functional} + Trise_{ireg} + Tv_{tree} + Trise_{lo} + Tn_{Functional} + Tfall_{ireg} + Tn_{tree} + Tfall_{lo}$$

Multiplier	Avg Cycle Time (ns)	Total Transistor Count
4 Bit	17.3	1224
8 Bit	21.6	3778
16 Bit	27.4	14840
32 Bit	31.9	53146

Table 2. Cycle time and total transistor count.

The results were compared with the simulation results obtained from an 8 bit adder designed with standard PCFB cells. The cycle time of the multiplier and its transistor count are reported to be 13.1 ns and 7983 respectively [7]. According to table 2 the transistor count in our design is halved. This improvement is due to the reduction of handshaking hardware. The improvement is achieved at the cost of increased cycle time.

6. Conclusion

This paper presented a design for a QDI asynchronous multiplier. In order to reduce the hardware overhead related to the implementation of the handshaking protocol in QDI circuits, data path and control sections were designed separately. The design of the data path unit utilized Booth algorithm which reduces the number of partial products, and used the reduction Wallace tree to speed up partial additions. Due to the cancellation of handshaking circuits from the cells, and because of separate implementation of the control unit, the hardware overhead of handshaking that is the main weakness of QDI circuits is halved in comparison to the conventional method of using standard cells. Accordingly, the cycle time is increased by 1.5. In the conventional method, all the outputs are neutralized simultaneously by a control signal, while in our proposed method, input neutralization is propagated through different layers before getting to the outputs. The forward delay in our method is less than that of the standard cell method. The reason is that the control signals are omitted from the cells, and the serial transistors in the pull down section in the data path are reduced.

The proposed method increases the concurrent branches, but these branches are limited locally, so the concurrency condition is easily implemented.

References

[1] Scott Hauck, "Asynchronous design methodologies An overview", IEEE Proc. ,Vol. 83 No.1 ,1995

[2] Alain J.Martin, "Synthesis of asynchronous VLSI circuits", California Institute of Technology, 1991

[3] Andrew Mattew Lines, "Pipelined asynchronous circuits", PhD thesis, California Institute of Technology,1995

[4 Tak Kwan Lee, "A general approach to performance analysis and optimization of asynchronous circuits", PhD thesis, California Institute of Technology, 1995

[5] Maitham Shams, Jo C.Ebergen, "A comparison of CMOS Implementation of an asynchronous circuit primitive the c-element", ISLPED, 1996.

[6] Alain J.Martin, "Asynchronous data paths and design of asynchronous adder", California Institute of Technology, 1992.

[7] Bahman Javadi, Mohsen Naderi, "Design and Synthesis of an asynchronous Booth multiplier by Martin method", Second Iranian Students Conference on Electrical Engineering ,1999