Design of Self-timed Asynchronous Booth's multiplier

Tin-Yau TANG, Chiu-Sing CHOY, Pui-Lam SIU, Cheong-Fat CHAN

Department of Electronic Engineering
The Chinese University of Hong Kong
Tel: (852) 2609 8272, Fax (852) 2603 5558

Abstract

This paper presents a design of multiplier for the multiplication of two 8-bit two-complement numbers. The multiplier applies the self-timed asynchronous methodology such that the multiplier can be assumed to operate on average case delay. And also, modified booth's algorithm [1] is used to reduce the number of partial product generated. As a result, the speed of the multiplier can be improved.

1. Introduction

Asynchronous design methodology has been studied actively in recent years. The main characteristic is using local distributed handshake signals instead of a global clock to control the operation of each function block in the system. Asynchronous methodology has certain advantages over its counterpart, the synchronous design [2]. It does not have the problem of clock skew since it does not have global clock. Other potential advantages include low power consumption and average-case delay performance.

In self-timed asynchronous circuits, each functional block is controlled by some handshake circuit such that each functional block is operated in correct order. Each functional block should also be able to acknowledge the completion of its operation to the handshake control circuit. The acknowledgement of the completion can be inherent in data signals or separated as a "complete" signal. In our multiplier design, SCCVSL (single-rail CMOS cascode voltage switch logic) [3] is used to generate the necessary "complete" signal.

The main advantage of using modified booth's algorithm [1] is that it generates fewer partial products. In ordinary multiplier, it is required to generate n partial products for n-bit multiplication. In our multiplier, only 4 partial products will be generated. Less partial product means less addition is needed to add up the partial products. As a result, it will have faster speed than the ordinary multiplier.

2. Implementation

2.1 Modified Booth's Algorithm [1]

In the modified booth's algorithm, an encoding technique is used to reduce the number of partial products. Different encoding techniques result different reductions of number of partial product. In our 8 × 8 multiplier, the multiplier (B) is divided into 4 substrings of 3 bits, with adjacent groups sharing a common bit. A '0' is also padded to the right of the multiplier B such that 4 complete substrings can be formed. Each substring is then decoded to give Y from the multiplicand (A) according the Table 1. With the above encoding scheme, 4 partial product will be generated for 8-bit × 8-bit signed multiplication. Each partial product is equal to Y multiplied by a scaling factor F. The scaling factor is equal to 1, 4, 16, 64 for the first, second, third and forth operation respectively. The final product is equal to the sum of the four partial products.

If $B=b_7b_6b_5b_4b_3b_2b_1b_0$, $B_1=b_1b_0 0$, $B_2=b_3b_2b_1$, $B_3=b_5b_4b_3$, $B_4=b_7b_6b_5$

$$P = A \times B = \sum_{n=1}^{4} F_n \times Y_n$$

where $F_1=1$, $F_2=4$, $F_3=16$, $F_4=64$

Bit pattern (B_n)	Operation (Y_n)
000	+0
001	+A
010	+A
011	+2A
100	-2A
101	-A
110	-A
111	-0

Table 1: Decoding table of 3-bits substring

2.2 hardware design

Fig. 1 shows the block diagram of the self-timed asynchronous booth's multiplier. The multiplier requires 4 cycles of operation to generate the final product. Each cycle generates one partial product and accumulates it with the partial product generated in pervious cycles. The multiplier is divided into two stages. The first stage mainly contains load/shift registers and booth's decoder. This stage is used to generate the partial product. The second stage mainly contains a 16-bit ripple-carry adder. This is used to add up all the partial products generated in each operation cycle. Besides the two stages, a central handshake controller is also included. It not only generate the internal request and reset signal for the two stages, but also handles the external request and complete signal.

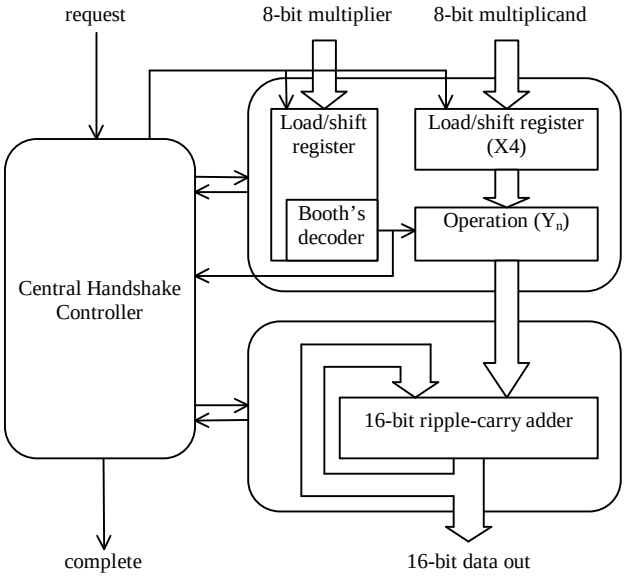

Fig. 1: The block diagram of the multiplier

Before the two 8-bit input data and the external request signal is received, all register and the adder will be cleared. The register for the multiplier is 9 bits long and the 8-bit multiplier will be loaded to the most significant 8 bits of the register. The register for the multiplicand is 16 bits long and the 8-bit multiplicand will be converted to 16-bit data by sign extent before loaded to the register. When the external request signal is received, the first operation cycle of the multiplier is started and the controller sends a signal to stage 1 such that the register in the stage 1 will load the two input data. As the same time, the booth's decoder decodes the least significant 3-bit of the register for multiplier. The decoder result is used to control the operation (double and/or negation) of the multiplicand to generate the correct partial product. The booth's decoder also indicates to the controller whether the partial product is zero or not.

After the stage 1 operation (generate partial product) has been finished, the stage 2 operation (addition) should be started. In the stage 2, an asynchronous version of ripple-carry adder is used. Although ripple-carry adder has large worst-case delay, its average case delay is approximately equal to or smaller than other adder such as carry look-ahead adder [1,4,5]. And also it requires less silicon area to implement. As we are using asynchronous methodology, average case delay can be assumed and so ripple-carry adder is used in our design.

According to our encoding method (Table 1), we can notice that there is about 25% of probability that the partial product is zero. No addition is needed for zero partial product. As a result, when the booth's decoder indicates to the controller that the partial product is zero, the operation of stage 2 will be aborted and directly enter into next operation cycle after the stage 1 operation. And so, the speed of the multiplier can be improved on average.

The second, third and forth operation cycle is similar to the first one. The content of the register for multiplier will shift right by 2 such that another 3-bit substring will be decoded. The content of the register for the multiplicand will also shift left by 2. This operation is equal to multiply 4 to the multiplicand and is necessary for the scaling factor of our modified booth's algorithm. The final product can be obtained in 4 operation cycles and the controller will send a complete signal when the final product comes out. The multiplier will reset to idle when the external request signal is inactive.

The multiplier is implemented by using AMS 0.6 μm technology. The chip size is about 2.2mm × 2.6mm. A number of full custom cells are first designed and whole layout is generated by automatic place and route. By simulation, the average evaluation time (request to complete) of the multiplier is about 90 ns in normal condition.

3. Conclusion

A self-timed asynchronous multiplier had been implemented. It performs average case delay. And also, by using modified booth's algorithm, the number of partial products is reduced. As a result, the speed of the multiplier is improved.

4. Reference

[1] Shlomo Waser and Michael J. Flynn, "Introduction to Arithmetic for Digital Systems Designers", CBS College Publishing.

[2] Scott Hauck, "Asynchronous Design Methodologies: An Overview", Proceedings of the IEEE, Vol. 83, No. 1, pp. 69-93, January 1995.

[3] Feng LI, Oliver Chiu Sing CHOY and Cheong Fat CHAN, "SCCVSL: A Single-Rail CMOS Cascode Voltage Switch Logic"

[4] Bruce Gilchrist, J.H. Pomerene and S.Y. Wong, "Fast Carry Logic for Digital Computers", IRE Transactions on Electronic Computers, Vol EC-4, pp 133-136, December 1955.

[5] George W. Reitwiesner, "The Determination of carry Propagation Length for Binary Addition", IRE Transactions on Electronic Computers, Vol EC-9, pp 35-38, 1960.