Fast Multiplier Using Booth Encoding

Haocong Wang

mw814@scarletmail.rutgers.edu

January 3, 2021

Abstract

In this project, we design and implement a 32-bit fast multiplier using booth encoding. We use Verilog to finish the coding of the multiplier. All the codes are finished on EDAplayground. We summary and compare the difference between fast multiplier using booth encoding and other types of multipliers. We test the proposed multiplier with a testbench module. The result shows that our multiplier works fast and well.

I. Introduction

Binary multiplier is one of the basic electronic circuits in digital electronics, such as a computer, to multiply two binary numbers [1]. As a basic functional unit circuit, multiplier is widely used in various signal processing and conversion circuits. Multiplier can be implemented with a variety of computer arithmetic techniques, which involve computing a set of partial products, and then summing the partial products together.

A binary computer does exactly the same multiplication as decimal numbers do, but with binary numbers. Each long number is multiplied by one digit (either 0 or 1). Therefore, the multiplication of two binary numbers comes down to calculating partial products, shifting them left, and then adding them together.

The method, however, is slow since it involves many intermediate additions, which are time-consuming, so fast multipliers need to developed in order to be adaptive to the increasingly complex computing process. There are various techniques and algorithms to make multipliers faster, such as Wallace tree, BKM algorithm and Kochanski multiplication [1]. Booth encoding is one of the common methods to calculate multiplication fast and accurately. In this project, we choose booth encoding as our target to design and implement a simple 32-bit fast multiplier using Verilog and EDAplayground as the implementation tools.

II. Related Work

i. Multiplication

The method for multiplying numbers is based on calculating partial products, shifting them to the left and then adding them together. The most difficult part is to obtain the partial products, as that involves multiplying a long number by one digit. We use binary multiplication as an example to illustrate the process, as shown in figure 1.

ii. Booth Encoding

Booth encoding, or Booth's multiplication algorithm [2], is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation[1].

[1] Andrew Donald Booth invented this algorithm in 1950 at Birkbeck College in Bloomsbury, London.

```
    1011   (this is 11 in binary)
  x 1110   (this is 14 in binary)
  ======
    0000   (this is 1011 x 0)
   1011    (this is 1011 x 1, shifted one position to the left)
  1011     (this is 1011 x 1, shifted two positions to the left)
+ 1011     (this is 1011 x 1, shifted three positions to the left)
  ========
 10011010  (this is 154 in binary)
```

Figure 1: *Illustration of the binary multiplication process.*

Booth's algorithm examines adjacent pairs of bits of the 'N'-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, $y_{-1} = 0$. For each bit y_i, for i running from 0 to N 1, the bits y_i and y_{i-1} are considered. Where these two bits are equal, the product accumulator P is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times 2^i is added to P; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times 2^i is subtracted from P. The final value of P is the signed product. More details will be given in Section III.

III. Description of Data, Method and Model

In this section, we will discuss the details of the data, method and model used in our project.

i. Data

In our project, we choose 32-bit binary numbers as our input data and therefore, the output will be 64-bit binary numbers. We consider two scenarios, which are unsigned numbers and signed numbers [3]. We consider the boundary conditions and choose 3 groups of multiplicands and multipliers, as shown in table 1

ii. Method

As mentioned in Section II, we provide detailed introduction for the method used in our project, booth encoding, in this section.

Booth algorithm can be implemented by repeatedly adding one of two predetermined values A and S to a product P, then performing a rightward arithmetic shift on P [4]. Suppose m and r be the multiplicand and multiplier, respectively and x and y represent the number of bits in m and r.

First, we determine the values of A and S, and the initial value of P. For A, we fill the most significant bits with the value of m and the remaining y + 1 bits with zeros. For S, we fill the most significant bits with the value of m in two's complement notation and the remaining y + 1 bits with zeros. For P, we fill the most significant x bits with zeros and then append the value of r to the right of this. Then we fill the least significant bit with a zero.

Second, we determine the two least significant bits of P. If they are 01, we find the value of P + A. If they are 10, we find the value of P + S. If they are 00, we do nothing and use P directly in the next step. If they are 11, we do nothing and use P directly in the next step.

Third, we arithmetically shift the value obtained in the second step by a single place to the right. Let P now equal this new value. we then repeat steps 2 and 3 until they have been done y times and drop the least significant bit from P, which is now the product of m and r.

iii. Model

Here we provide some introduction to our designed model. First we allow users to input two 32-bit binary numbers, a and b, as the multiplier and multiplicand. We set abs_a and abs_b to store the absolute value of a and b, which will be used for signed and unsigned scenarios. Then we write Verilog code to perform booth encoding on these two binary numbers including shifting and adding [5]. The final results will be shown in EPWave.

IV. Experimental Procedure and Results

We use Verilog to write two modules. One is the designation of the multiplier and an-

Multiplicand	Multiplier
00000000000000000000000000001111	00000000000000000000000000000100
11111111111111111111111111111111	00000000000000000000000000000001
11111111111111111111111111111111	11111111111111111111111111111111

Table 1: *Input Data*

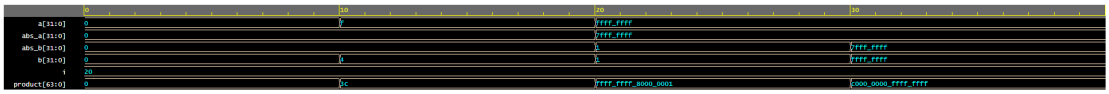

Figure 2: *Results of our experiments.*

Experiments	Results
Exp 1	3c
Exp 2	ffffffff80000001
Exp 3	c0000000ffffffff

Table 2: *Results*

other one is the test bench. All the codes are finished on EDAplayground. As mentioned in Section III, we use three groups of multiplicands and multipliers to test our designed multiplier, which are shown in table 1. We include the EPWave of the test bench in figure 2. The last line of EPWave shows the result of each group.

In order to make it clear to review, we also add a table to show the results, as shown in table 2. We use hexadecimal numbers to represent the products. The result of each multiplication is correct, which means our proposed multiplier is accurate and finish the calculation fast.

V. Conclusion

In this project, we implement a fast multiplier using booth encoding. We finish the designation with Verilog on EDAplayground. The test results show that our product can finish calculation fast and accurately.

References

[1] Binary multiplier, 2020. `https://en.wikipedia.org/wiki/Binary_multiplier`.

[2] Booth's multiplication algorithm, 2020. `https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm`.

[3] A. D. Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[4] D. Chandel, G. Kumawat, P. Lahoty, V. V. Chandrodaya, and S. Sharma. Booth multiplier: Ease of multiplication. *International Journal of Emerging Technology and Advanced Engineering*, 3(3):118–122, 2013.

[5] E. Deng, Y. Zhang, W. Kang, B. Dieny, J.-O. Klein, G. Prenat, and W. Zhao. Synchronous 8-bit non-volatile full-adder based on spin transfer torque magnetic tunnel junction. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(7):1757–1765, 2015.

[6] I. E. L. Manalo, E. R. Q. Montecillo, D. S. Colinares, J. D. L. Lingat, and R. B. Caldo. 12-bit verilog calculator with trigonometric functions.