THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

Design and Implementation of Optimized Floating Point Matrix Multiplier Based on FPGA

Maruti L. Doddamani
IV Semester, M.Tech (Digital Electronics), Department of Electronics & Communication
S.D.M. College of Engineering and Technology, Dharwad, Karnataka, India
Mala L. M.
Assistant Professor, Department of Electronics & Communication
S.D.M. College of Engineering and Technology, Dharwad, Karnataka, India

Abstract:
In computing, floating point describes a method of representing an approximation in real numbers that can support a wide range of values. A binary multiplier is an integral part of the ALU subsystem found in many processors. Integer multiplication can be inefficient and costly, in time and hardware, depending upon the representation of signed numbers. Booth algorithms suggest technique for multiplying signed numbers that works well for both positive and negative multipliers. This work modifies Weng and Duh's design with modified Booth multiplier in multiplication to reduce the number of partial products. As a result, the improved floating point matrix multiplier having better performance with shorter delay has been designed compared to Weng and Duh's design.

Key words*: floating point multiplication, Kogge Stone adder, modified Booth multiplier*

1. Introduction

Although computer architecture is sometimes viewed as a specialized part of CPU design, still the discrete component designing is also a very important aspect. A tremendous variety of algorithms have been proposed for use in floating point systems. Many scientific computing applications demand high numerical stability and accuracy and hence are usually using floating point arithmetic [1]. Recently, there has been a research on efficiently improving the performance of floating point matrix multiplication, some of which are implemented on FPGA devices [1,2]. Our work of floating point will focus exclusively on the IEEE floating point standard 754 because of its rapidly increasing acceptance. Multiplying floating point numbers is a major requirement for DSP applications involving large dynamic range. This paper focuses on single precision normalized floating point matrix multiplication where delay is reduced and performance is improved. The multiplication can be partitioned into three sequential steps: generating partial products, summing up the partial products which were produced in former step until two vectors remain, and finally summing up the remaining two vectors by using a Kogge Stone adder (KSA).

Two methods are generally adopted to form the partial products in the first step. The first method uses 2-input AND gates to generate partial products and the second method applies radix-4 Booth encoding, which has been used by Yeh and Jen[3]. The former is quite simple and ordinary. The latter is more complex on implementation, but it is widely used to reduce the number of partial products. Based on the architecture of floating point matrix multiplier, this work modifies the final addition unit by using Kogge stone adder (KSA) to generate the product by summing up the remaining two vectors in final step. Modified Booth encoder is employed during generating partial products. The height of the partial product matrix can be reduced. The segments of sign extension bits of all partial products are merged into only one singular vector and a prominent bit.

Floating-point implementation on FPGAs has been the interest of many researchers. In [2], an IEEE 754 single precision pipelined floating point multiplier was implemented on multiple FPGAs (4 Actel A1280). In [3], a custom 16/18 bit three stage pipelined floating point multiplier that doesn't support rounding modes was implemented. In [4], a single precision floating point multiplier that doesn't support rounding modes was implemented using a digit-serial multiplier: using the Altera FLEX 8000 it achieved 2.3 MFlops. In [5], a parameterizable floating point multiplier was implemented using the software-like language Handel-C, using the Xilinx XCV1000 FPGA; a five stages pipelined multiplier achieved 28MFlops. In [6], a latency optimized floating point unit using the primitives of Xilinx Virtex II FPGA was implemented with a latency of 4 clock cycles. The multiplier reached a maximum clock frequency of 100 MHz.

As a result, the delay is reduced. Comparing with Weng and Duh's design the proposed multiplier unit and addition unit has 5.32% and 82.08% of improvement in delay, respectively.

2. Background

In 1985, IEEE published the standard 754[6]. It provides a standard for binary floating point number format. The standard included different length of format. This work designs a floating-point matrix multiplier for 32-bit single precision format. Nonetheless, the design can be transformed into other formats easily.

Fig1. Shows the IEEE 754 single precision binary representation; it consists of one sign bit (S), an 8-bit exponent (E), and 23-bit fraction (F) or mantissa. If an extra bit is added to a fraction then it is called significand. If the exponent is greater than 0 and smaller than 255, and there is 1 in the MSB of the significand then number is said to be normalized one. The sign bit represents the sign of the floating point number. Zero represents positive, otherwise negative. The exponent part is biased by 2t-1, where t is the number of bits in exponent part. Thus the bias in single precision is 127.

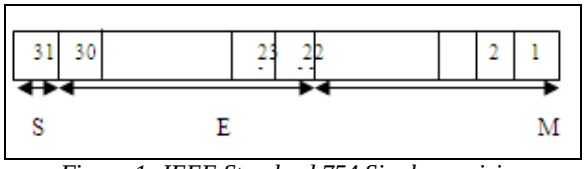

Figure 1: IEEE Standard 754 Single precision

Multiplying two numbers in floating point format is done by 1- multiplying the mantissa part of two numbers, 2- adding the exponent part of two numbers and obtained result is subtracted from bias, 3 – sign is calculated by XOR ing the sign part of two numbers. To represent the multiplication result as normalized number there should be 1 in the MSB of the result.

Nevertheless, the floating point multiplication and floating point adder are modular. With multipliers and adders, floating point matrix multiplier with better performance can be achieved.

3. Floating Point Multiplication Algorithm

As stated in the introduction, normalized floating point numbers have the form of Z= (-1S) * 2 (E - *Bias*) * (1.M). To multiply two floating point numbers the following is done:

- Multiplying the significand; i.e. (1.M1*1.M2).
- Placing the decimal point in the result
- Adding the exponents; i.e. (E1 + E2 – *Bias*)
- Obtaining the sign; i.e. s1 xor s2.
- Normalizing the result; i.e. obtaining 1 at the MSB of the results‟ significand
- Rounding the result to fit in the available bits

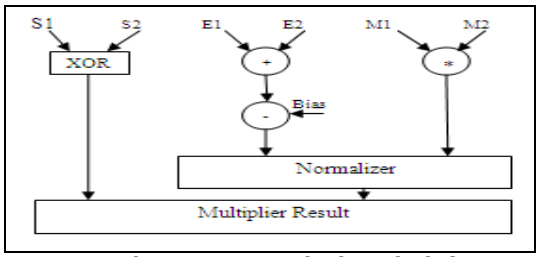

Figure 2: Floating Point Multiplier Block diagram

Figure 2 shows the block diagram of multiplication of two floating point multiplier (A & B). It consists of two precision formats namely single precision (32 bit) and double precision (64 bit). The single precision consists of sign (1 bit), exponent (8 bit) and significand (23 bit). To design the floating point multiplier, instead of using carry look ahead adder(CLA) kogge stone adder and instead of normal multiplier use booth multiplier. These are efficient in the present market and consume very less area and produce less delay. By this method an efficient floating point matrix multiplier is designed and implemented on FPGA.

4. Hardware Of Floating Point Multiplier

4.1. Sign Bit Calculation

Multiplying two numbers results in a negative sign number if and only if one of the multiplied numbers is of a negative value. By the aid of a truth table we find this can be obtained by XOR ing the sign of two inputs.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: XOR Truth Table

4.2. 8 Bit Kogge Stone Adder

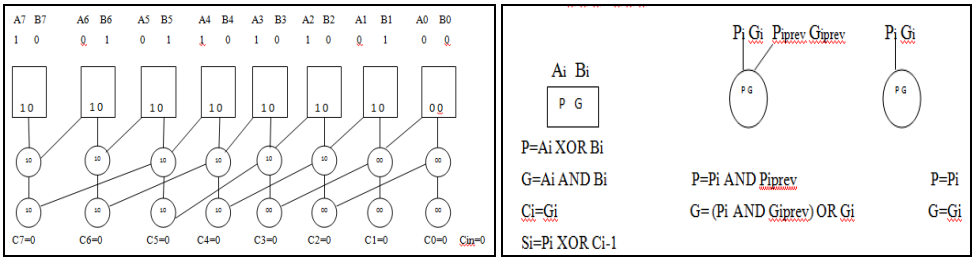

Figure 3: 8 bit kogge stone adder

This unsigned adder is responsible for adding the exponent of the first input to the exponent of the second input and subtracting the Bias (127) from the addition result (i.e. A_exponent + B_exponent - Bias). The result of this stage is called the intermediate exponent. The add operation is done on 8 bits, and there is no need for a quick result because most of the calculation time is spent in the significand multiplication process (multiplying 24 bits by 24 bits); thus we need a moderate exponent adder and a fast significand multiplier.

The Kogge–Stone adder concept was developed by 'Peter M. Kogge' and 'Harold S. Stone' in 1974.A 8-bit Kogge–Stone adder is shown to the right. Each vertical stage produces a "propagate" and a "generate" bit, as shown. The culminating generate bits (the carries) are produced in the last stage (vertically), and these bits are XOR'd with the initial propagate after the input (the red boxes) to produce the sum bits. The advantage of Kogge stone adder is widely considered the fastest adder design possible, common design for high-performance adders in industry and has a lower fan-out at each stage, which increases performance.

The table shows the comparison of various adders with respect to the delay i.e., Ripple carry adder has 20.394 ns, carry look ahead adder 18.644ns and Kogge stone Adder has 16.509ns.

ADDERS	**Results obtained Delay (ns)**
Ripple carry adder	20.394
Carry look ahead Adder	18.644
Kogge Stone Adder	16.509

Table 2: Comparison of various Adders with delay

4.3. Modified Booth Multiplier

It is possible to reduce the number of partial products by half, by using the technique called radix 4 Booth Encoding. The basic idea is that, instead of shifting and adding for every column of the multiplier term and multiplying by 1 or 0, we only take every second column, and multiply by ±1, ±2, or 0, to obtain the same results. Radix-4 booth encoder performs of encoding the multiplicand based on multiplier bits. It will compare three bits at a time with overlapping technique. Grouping starts from the LSB, and the first block only uses two bits of the multiplier and assumes a zero for the third bit as shown by figure 4.

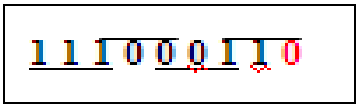

Figure 4: 3 bit pairing as per booth Encoding

The functional operation of Radix-4 booth encoder is shown in Table 3. It consists of eight different types of states and during these states we can obtain the outcomes, which are multiplication of multiplicand with 0,-1,-2 consecutively.

Block	Partial Product
000	0
001	1 * Multiplicand
010	1 * Multiplicand
011	2 * Multiplicand
100	-2 * Multiplicand
101	-1 * Multiplicand
110	-1 * Multiplicand
111	0

Table 3: Eight different possible states

5. Results

The Simulation result for sign bit calculation is shown in fig 5

.

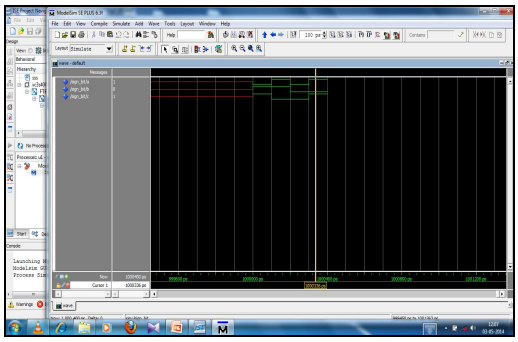

Figure 5: sign bit calculation

- The Simulation result for Kogge Stone Adder is shown in fig 6

.

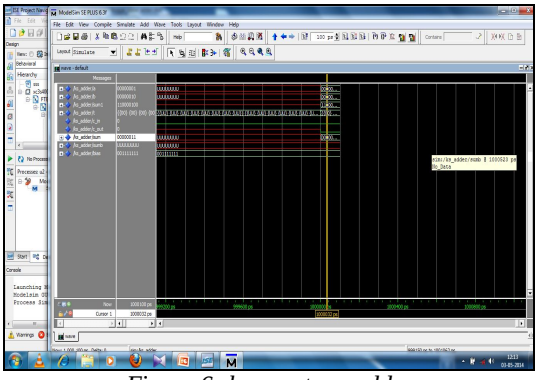

Figure 6: kogge stone adder

- The Simulation result for Modified Booth Multiplier is shown in fig 7

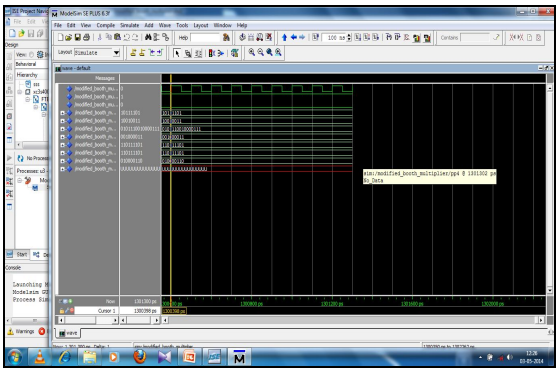

Figure 7: Modified Booth Multiplier

- The simulation result for two 4 X 4 matrix multiplication as shown in fig 8

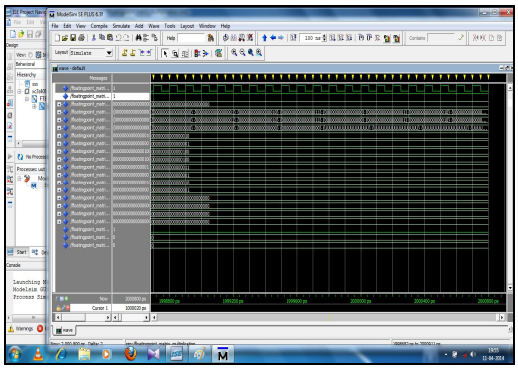

Figure 8: 4 x4 matrix multiplication

In this work, a common approach is used to estimate the improved floating point multiplication so that delay is reduces of addition unit and multiplication unit. The table 3 shows the comparison of delay of each component unit for our, Weng and Duh's works

	My Work (Spartan 3)	**Weng and Duh's**	**Yang and Duh's**
	Delay(ns)	Delay(ns)	Delay(ns)
Multiplication unit	22.282	37	39
Addition unit	25.360	33	33
KSA (final Addition)	16.509	96	96

Table 4: Comparison of each Component

In our work the delay of matrix multiplication is 16.146 compared to Yang and Duh's work. So, in this work we reduced delay to greater extent and efficiently improved the performance of the matrix multiplication.

6. Conclusion

The delay of multiplication unit and addition unit in our work is 22.282 and 16.509 respectively. By contrast, those in Weng and Duh's work are 37 and 96. Thus the delay in multiplication unit and addition unit improves by 5.32% and 82.08% respectively. In the simplest design of a 4*4 floating point multiplication the improvements compared with Weng and Duh's design is 10.27% in delay respectively.

7. References

1. IEEE 754-2008, IEEE Standard for Floating- Point Arithmetic, 2008.
2. F. Bensaali, A.Amira, and R. Sotudeh "floating – point matrix product on FPGA," in: Proc.ACS/IEEE International Conference on Computer Systems and Applications, pp. 466- 473, May.2007.
3. Wei- Ting Weng and Dyi-Rong Duh "Improved Floating point Matrix Multiplier".
4. W.C.Yeh and C.W.Jen, "High speed Booth encoded parallel multiplier design," IEEE Trans. Comput., vol.49, no.7 , pp. 692-701, Jul. 2000.
5. C.S. Wallace, " A suggestion for a fast multiplier," IEEE Trans. Comput., vol. 13, no.2, pp. 14-17, Feb, 1964.
6. L. Dadda, "Some schemes for parallel multipliers," Alta Frequenza, vol.34,pp. 349- 356, 1965.
7. IEEE Standard for Binary Floating Point Arithmetic, ANSI/IEEE Standard 754-`1985.
8. L.C.Yang and D.R.Duh, "Optimized design of a floating point matrix multiplier," in:Proc. National Computer Symp., pp. 300-308, Nov.20.
9. Mohamed Al-Ashrafy, Ashraf Salem, Wagdy Anis, An Efficient Implementationof Floating Point Multiplier, 978-1-4577-0069-9/11/2011 IEEE.
10. L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," Proceedings of 83 the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM‟96), pp. 107–116, 1996