International Journal of Research in Engineering, Science and Management
Volume-1, Issue-8, August 2018
www.ijresm.com
ISSN (Online): 2581-5782

Design and Implementation of Different Multipliers Using VHDL

Chandramohan Kumrawat[1], Deepak Sharma[2]

[1]*Student, Department of Electronics and Communications, LKCT, Indore, India*

[2]*Assistant Professor, Department of Electronics and Communications, LKCT, Indore, India*

***Abstract*: High speed multiplier Low power consumption and smaller area are some of the most important criteria for the fabrication of DSP systems and high performance systems. Project presents an efficient implementation of high speed multiplier using the shift and add method, Radix_2, Radix_4 modified Booth multiplier algorithm. To design these types of multiplier, different types of adders like sixteen bit full adder can be used. After that we will design a 4 tap delay FIR filter and in place of the multiplication and addition we would use that multipliers and adders which we have implemented. Then we will compare the working of different multipliers by comparing the power consumption by each of them. The project helps us to choose a better option between serial and parallel multiplier in fabricating different systems.**

Keywords: Multipliers, VHDL.

I. Introduction

Multipliers are key components of many high performance systems. However, Area and speed are usually conflicting constraints so that improving speed results mostly in larger areas. As a result, a whole spectrum of multipliers with different area-speed Constraints has been designed with fully parallel. These multipliers have moderate performance in both speed and area. The pipelining done at the digit level brings the benefit of constant operation speed irrespective of the size of' the multiplier.

The VHDL language supports these modelling needs at the algorithm or behavioral level, and at the implementation or structural level. It provides a versatile set of description facilities to model DSP circuits from the system level to the gate level. Recently, we have also noticed efforts to include circuit-level modelling in VHDL. At the system level we can build behavioral models to describe algorithms and architectures. We would use concurrent processes with constructs common to many high-level languages, such as if, case, loop, wait, and assert statements. VHDL also includes user-defined types, functions, procedures, and packages." In many respects VHDL is a very powerful, high-level, concurrent programming language. At the implementation level we can build structural models using component instantiation statements that connect and invoke subcomponents. The VHDL generate statement provides ease of block replication and control. A dataflow level of description offers a combination of the behavioral and structural levels of description. VHDL lets us use all three levels to describe a single component. Most importantly, the standardization of VHDL has spurred the development of model libraries and design and development tools at every level of abstraction. VHDL, as a consensus description language and design environment, offers design tool portability, easy technical exchange, and technology insertion.

II. Purpose of Implementation

Power consumption in VLSI DSPs has gained special attention due to the proliferation of high-performance portable battery-powered electronic devices such as cellular phones, laptop computers, etc. DSP applications require high computational speed and, at the same time, suffer from stringent power dissipation constraints. Multiplier modules are common to many DSP applications. The fastest types of multipliers are parallel multipliers. Among these, the Wallace multiplier is among the fastest. However, they suffer from a bad regularity. Hence, when regularity, high performance and low power are primary concerns, Booth multipliers tend to be the primary choice. Booth multipliers allow the operation on signed operands in 2's complement. They derive from array multipliers where, for each bit in a partial product line, an encoding scheme is used to determine if this bit is positive, negative or zero. The Modified Booth algorithm achieves a major performance improvement through radix-4 encoding. In this algorithm each partial product line operates on 2 bits at a time, thereby reducing the total number of the partial products. This is particularly true for operands using 16 bits or more.

III. System Design

A. Binary Multiplier

A Binary multiplier is an electronic hardware device used in digital electronics or a computer or other electronic device to perform rapid multiplication of two numbers in binary representation. It is built using binary adders. The rules for binary multiplication can be stated as follows [2],

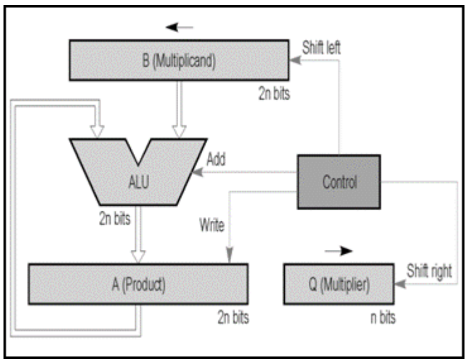

Fig. 1. Multiplication Hardware Implementation

1) If the multiplier digit is a 1, the multiplicand is simply copied down and represents the product.
2) If the multiplier digit is a 0 the product is also 0. For designing a multiplier circuit we should have circuitry to provide or do the following three things:
   1. It should be capable identifying whether a bit 0 or 1 is.
   2. It should be capable of shifting left partial products.
   3. It should be able to add all the partial products to give the products as sum of partial products. [2].

B. Booth Multiplier

The decision to use a Radix-4 modified Booth algorithm rather than Radix-2 Booth algorithm is that in Radix-4, the number of partial products is reduced to n/2. Though Wallace Tree structure multipliers could be used but in this format, the multiplier array becomes very large and requires large numbers of logic gates and interconnecting wires which makes the chip design large and slows down the operating speed.

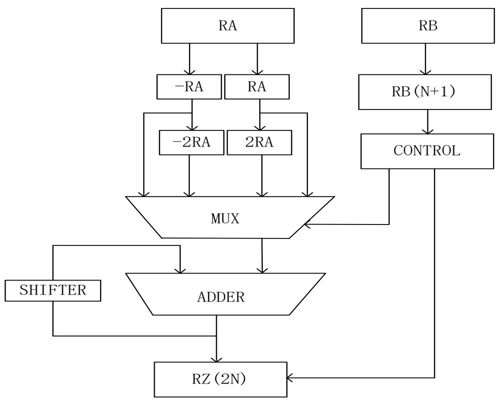

Fig. 2. Block diagram of Booth multiplier algorithm

1. Booth Multiplication Algorithm for radix-2

Booth algorithm gives a procedure for multiplying binary integers in signed –2's complement representation. I will illustrate the booth algorithm with the following Example, 2 ten x (- 4) ten 0010 two* 1100 two

Step 1: Making the Booth table

From the two numbers, pick the number with the smallest difference between a series of consecutive numbers, and make it a multiplier.
i.e., 0010 -- From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change, and so there are two changes on this one
1100 -- From 1 to 1 no change, 1 to 0 one change, 0 to 0 no change, so there is only one change on this one.

Therefore, multiplication of 2 x (– 4), where 2 ten (0010 two) is the multiplicand and (– 4) ten (1100 two) is the multiplier.

II. Let X = 1100 (multiplier)
Let Y = 0010 (multiplicand)
Take the 2's complement of Y and call it –Y –Y = 1110
III. Load the X value in the table.
IV. Load 0 for X-1 value it should be the previous first least significant bit of X
V. Load 0 in U and V rows which will have the product of X and Y at the end of operation.
VI. Make four rows for each cycle; this is because we are multiplying four bits numbers.

U	**V**	**x**	**x -1**	
0000	**0000**	**1100**	**0**	**Load Value**
				1stCycle
				2nd Cycle
				3rd Cycle
				4thCycle

Step 2: Booth Algorithm

Booth algorithm requires examination of the multiplier bits, and shifting of the partial product. Prior to the shifting, the multiplicand may be added to partial product, subtracted from the partial product, or left unchanged according to the following rules:

Look at the first least significant bits of the multiplier "X", and the previous least significant bits of the multiplier "X - 1".

I. 0 0 Shift only
   1 1 Shift only.
   0 1 Add Y to U, and shift
   1 0 Subtract Y from U, and shift or add (-Y) to U & shift
II. Take U & V together and shift arithmetic right shift which preserves the sign bit of 2's complement number. Thus a positive number remains positive, and a negative number remains negative.
III. Shift X circular right shifts because this will prevent us from using two registers for the X value shift only

U	**V**	**x**	**x-1**
0 0 0 0	0 0 0 0	1 1 0 0	0
0 0 0 0	0 0 0 0	0 1 1 0	0
0 0 0 0	0 0 0 0	0 0 1 1	0
1 1 1 0	0 0 0 0	0 0 1 1	0
1 1 1 1	0 0 0 0	1 0 0 1 ←	1
1 1 1 1	**1 0 0 0**	**1 1 0 0**	**1** ←

2. Booth Multiplication Algorithm for radix-

One of the solutions of realizing high speed multipliers is to enhance parallelism which helps to decrease the number of subsequent calculation stages. The original version of the Booth algorithm (Radix-2) had two drawbacks. They are:

1. The number of add subtract operations and the number of shift an operation becomes variable and

becomes inconvenient in designing parallel multipliers.

2. The algorithm becomes inefficient when there are isolated 1's. These problems are overcome by using modified Radix4

Booth algorithm which scan strings of three bits with the algorithm given below:

1) Extend the sign bit 1 position if necessary to ensure that n is even.
2) Append a 0 to the right of the LSB of the multiplier.
3) According to the value of each vector, each Partial Product will he 0, +y , -y, +2y or -2y. The negative values of y are made by taking the 2's complement and in this paper Carry-look-ahead (CLA) fast adders are used. The multiplication of y is done by shifting y by one bit to the left. Thus, in any case, in designing a n-bit parallel multipliers, only n/2 partial products are generated. Table I Radix4 Modified Booth algorithm scheme for odd values of i.

X (i)	X(i-1)	X(i-2)	y
0	0	0	+0
0	0	1	+y
0	1	0	+y
0	1	1	+2y
1	0	0	-2y
1	0	1	-y
1	1	0	-y
1	1	1	+0

IV. Expected Result

TABLE I
Array Multiplier

Number of Slices	229
Number of 4 input LUT's	302
Number of Bonded Input	16
Number of Bonded output	16
CLB Logic Power	104 mW

TABLE II
Radix 2 Booth Multiplier

Number of Slices	130
Number of 4 input LUT's	249
Number of Bonded Input	16
Number of Bonded output	17
CLB Logic Power	79mW

TABLE III
Radix 2 Booth Multiplier

Number of Slices	229
Number of 4 input LUT's	302
Number of Bonded Input	16
Number of Bonded output	16
CLB Logic Power	47 mW

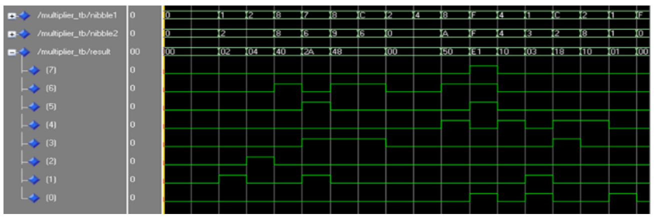

Fig. 3. Multiplier output

V. Conclusion

While comparing the radix 2 and the radix 4 booth multipliers we found that radix 4 consumes lesser power than that of radix 2. This is because it uses almost half number of iteration and adders when compared to radix 2. When all the three multipliers were compared we found that array multipliers are most power consuming and have the maximum area. This is because it uses a large number of adders. As a result it slows down the system because now the system has to do a lot of calculation. Multipliers are one the most important component of many systems. So we always need to find a better solution in case of multipliers. Our multipliers should always consume less power and cover less power. So through our project we try to determine which of the three algorithms works the best. In the end we determine that radix 4 modified booth algorithm works the best.

VI. Future Work

In our project we will try to determine the best solution to this problem by comparing a few multipliers. This project presents an efficient implementation of high speed multiplier using the shift and add method, Radix_2, Radix_4 modified Booth multiplier algorithm. In this project we compare the working of the multiplier by implementing each of them separately in FIR filter. For this purpose we will first design three different type of multipliers using shift method, radix 2 and radix 4 modified booth multiplier algorithm. To design these types of multiplier, different types of adders like sixteen bit full adder can be used. After that we will design a 4 tap delay FIR filter and in place of the multiplication and addition we would use that multipliers and adders which we have implemented. Then we will compare the working of different multipliers by comparing the power consumption by each of them. The project helps us to choose a better option between serial and parallel multiplier in fabricating different systems.

References

[1] Circuit Design using VHDL, by Pedroni, page number 285-293.
[2] VHDL by Sjoholm Stefan, VHDL by B. Bhaskar
[3] VHDL by B Bhaskar
[4] Parhami, Behrooz. Computer Arithmetic: Algorithms and Hardware Designs. New York: Oxford, 2005.
[5] Tam, Nguyen, Long Pham, Jon Benson. BOOTH'S Multiplier & 32 Bit ALU for ARM7 microprocessor ECE 345, Initial project proposal Date: FEB /8/ 2000
[6] Digital Signal Processing by Johnny R Johnson, PHI publications.
[7] Digital Signal Processing by Vallavraj & Salivhanan, TMH publications.

[8] Patterson, David and Hennessy, John. Computer Organization and Design - The Hardware/Software Interface, San Francisco: Morgan Kaufmann Publishers, 1998.
[9] Martinez – Peiro and Lars Wanhammar. Department of Electronic Eng. University of Valencia. High Speed, Low Complexity FIR Filter Using Multiplier Block Reduction and Polyphase Decomposition.
[10] Ruchi Sharma Analysis of Different Multiplier with Digital Filters Using VHDL Language. International Journal of Engineering and Advanced Technology, vol. 2, no. 1, October 2012.
[11] S. Chouhan, Y. Kumar, “Low Power Designing of FIR Filters,” in *International Journal of Advanced Technology & Engineering Research*, vol. 2, no. 2, pp. 59-67, 2012.
[12] Shambhavi S, K B ShivaKumar, M Z Kurian, H S Jayaramu. Multiplier Based On Add And Shift Method By Passing Zero. Department of EC, Sri Siddhartha Institute of Technology, Tumkur, Karnataka, India.
[13] Fábio Fabian Daitx Wagner S. Rosa Eduardo Costa Paulo Flores Sergio VHDL Generation of Optimized FIR Filters, 2008 International Conference on Signals, Circuits and Systems, Brazil.