International Journal of Multidisciplinary Research and Technological Advancement (IJMRTA)

Volume 1 Issue 1, Year Nov-Dec 2023 *Available at www.ijmrta.in*

****

Applying Field Programmable Gate Arrays to Implement Multiplication operation

SHASHIKUMAR RAM
Student, Electronics and Telecommunication Engineering
Smt. Indira Gandhi College of Engineering, Navi Mumbai, Maharashtra, India

SURAJ PATIL
Student, Electronics and Telecommunication Engineering
Smt. Indira Gandhi College of Engineering, Navi Mumbai, Maharashtra, India

ROHAN NIKAM
Student, Electronics and Telecommunication Engineering
Smt. Indira Gandhi College of Engineering, Navi Mumbai, Maharashtra, India

JAYESH WAGHMARE
Student, Electronics and Telecommunication Engineering
Smt. Indira Gandhi College of Engineering, Navi Mumbai, Maharashtra, India

UMAKANT GOHATRE
Assistant Professor, Electronics and Telecommunication Engineering
Smt. Indira Gandhi College of Engineering, Navi Mumbai, Maharashtra, India

****

ABSTRACT:

Multiplication is at the core of many real-time signal processing algorithms used today. Its primary use case was functional unit processing for signals and images. Here in this study, we model the efficient design of several multiplication algorithms. In addition, the article presents a novel method of multiplication based on the barrel shifter, which offers a variation on the algorithm for adding and shifting that was previously discussed. Studies focusing on four algorithms: the array multiplier, the shift-and-add multiplier, the Wallace tree multiplier, and the Vedic vertical crosswise multiplication technique. In further research, we will compare several multipliers according to criteria such as logical resource utilisation, latency, power consumption, and area. The experimental setup employs a sparten-3 XC3S400 FPGA for hardware description and implementation, as well as parametric analysis. Many built-in, compatible facilities for parameter analysis, such as XPE for power analysis, are included in the Xilinx ISE-simulation

tool. At long last this paper presents the results of simulations for the aforementioned multiplicands with 8, 16, and 32 bits.

Keywords – FPGA, Multiplication, Digital VLSI Design, SoC, ASIC

I. INTRODUCTION

These days, high speed integrated circuits are necessary for a lot of application-specific processor implementations including real-time signal processing, picture processing, encryption, and data manipulation. The math multiplier is a common tool in signal processing algorithms. On SOC, you may find and use a variety of multiplication algorithms. The most well-known methods for multiplying numbers are the Booth, Wallace tree, Karatsuba, and Braun algorithms. In our research, we are analysing these many characteristics and are going to present a modified multiplication algorithm based on a barrel shifter. However, many of these algorithms have their own restrictions in terms of speed, on-chip size, usage of logic gates, energy, and cost per cell. Within the span of a single clock cycle, a barrel shifter may move N positions to the left or right by an arbitrary quantity (according to need) [1]. If this practical behaviour of the barrel shifter can be studied, it might be utilised to create a multiplier that takes the multiplicand number and multiplies it by the multiplier number, which is a perfect square. An extensible multiplier block with configurable parameters (such as input parameter length, computing time, and occupied space), accessible via a standard interface, is the object of this design and implementation effort. Once used only for glue logic, field-programmable gate arrays (FPGAs) have now expanded into the creation of reconfigurable systems and architectures. The increased flexibility and strong computational architectural features offered by field-programmable gate arrays (FPGAs)—including embedded multipliers, programmable logic components, DSP blocks, and Block RAMs—have led to their widespread use as a platform for a wide range of applications. Matrix operations for floating point matrices are the subject of active research towards FPGA implementation.

II. LITERATURE SURVEY

Due to their widespread usage in scientific computing, the design and implementation of double precision floating-point matrix multipliers using field-programmable gate arrays (FPGAs) is a relatively new field of study. The literature is scattered with research on the realisation of matrix multipliers on FPGAs. This study makes an effort to compile and provide a literature review covering the last decade that discusses several facets of designing and realising matrix multipliers using field-programmable gate arrays (FPGAs). What follows is the outline for the rest of the article. Part II delves into the salient aspects of FPGA. Several methods and approaches for designing and implementing fixed- and floating-point matrix multipliers on FPGAs are covered in Section III. In section IV, we have a concise outline of the areas of application where the matrix multiplier is crucial. In Section V, we have summarised the results. 174 Presented at the

MTE. In contrast to other design methods, such as application-specific integrated circuits (ASICs) or entire bespoke ICs, field-programmable gate arrays (FPGAs) have a far shorter design turnaround time and may execute abstract logic operations. Because of their adaptability, FPGAs have been gaining market share in the integrated circuits industry. Although FPGAs have a shorter time to market and more flexibility than ASICs, these benefits come at the expense of slower processing speeds and higher power consumption. Logic blocks (LB) in a conventional field-programmable gate array (FPGA)—shown in Fig. 20.26—are able to execute any digital logic function with the help of certain sequential components and arithmetic units [760]. The connections between the LBs are provided by the switch boxes (SBs). The SBs are equipped with pass transistors that link the incoming and outgoing routing tracks. In order to tailor the LBs to a particular task, memory circuits manage the pass transistors. The power consumption of FPGAs may be rather high due to the SBs, which are the main components of the connection delay between the LBs.

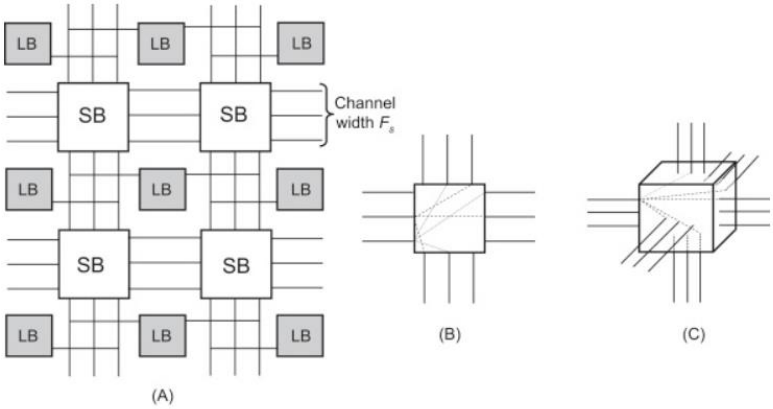

Figure 2.1. Typical FPGA architecture, (A) 2-D FPGA, (B) 2-D switch box, and (C) 3-D switch box. A routing track can connect three outgoing tracks in a 2-D SB

Extending FPGAs to the third dimension can improve performance while decreasing power consumption as compared to conventional planar FPGAs. A generalization of FPGAs to the third dimension would include multiple planar FPGAs, wafer or die bonded to form a 3-D system. The crucial difference between a 2-D and 3-D FPGA is that the SB provides communication to five LBs in a 3-D system rather than three neighbouring LBs as in a 2-D FPGA (see Figs. 20.26B and 20.26C). Consequently, each incoming interconnect segment connects to five outgoing segments rather than three outgoing segments. The situation is somewhat different for the bottom and topmost tier of a 3-D FPGA, but in the following discussion, for simplicity this difference is neglected. Since the connectivity of a 3-D SB is greater, additional pass transistors are required in each SB, increasing the power consumption, memory requirements to configure the SB, and, possibly, the interconnect delay. The decreased interconnect length and greater connectivity can compensate, however, for the added complexity and power of the 3-D SBs.

To estimate the size of the array beyond which the third dimension is beneficial, the shorter average interconnect length offered by the third dimension and the increased complexity of the SBs should be simultaneously considered [761]. Incorporating the hardware resources (e.g., the number of transistors) required for each SB and the average interconnect length for a 2-D and 3-D FPGA, the minimum number of LBs for a 3-D FPGA to outperform a 2-D FPGA is determined from the solution of the following equation, (20.27) Fs,2−D23N1/2=Fs,3−DN1/3,

where $F_{s,2\text{-}D}$ and $F_{s,3\text{-}D}$ are, respectively, the channel width of a 2-D and 3-D FPGA, respectively, and N is the number of LBs. Solving (20.27) yields N=244, a number that is well exceeded in modern FPGAs. Since the pass transistors, employed both in 2-D and 3-D SBs, contribute significantly to the interconnect delay, degrading the performance of an FPGA, those interconnects that span more than one LB can be utilized. These interconnect segments are named after the number of LBs that is traversed by these segments

III. MODEL PROJECT BLOCK DIAGRAM AND FLOWCHART

Many modern processors take their multiplication algorithms from the Vedic multiplier, which appears in the sixteen sutras of the Indian Vedas. Its column-wise addition and bit-wise multiplication make up the Vedic multiplier, which finds the product term simultaneously. As a benchmark for efficient multiplication algorithms, it ranks among the top. This paper explains the vertical crosswise multiplication technique and how to utilise its basic block effectively. Following the block layout below, this paper explains how to build a basic Vedic block that can process two-bit data. With four of these blocks, you can process four-bit data.

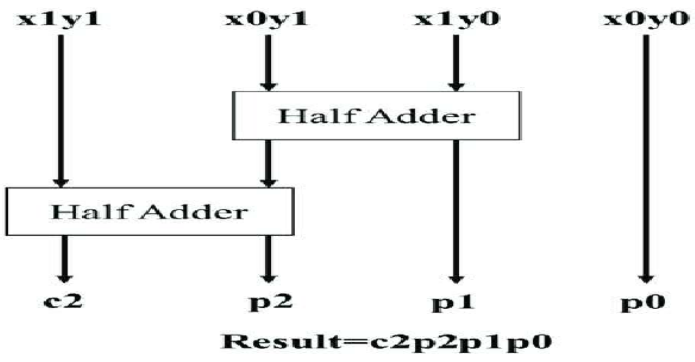

Figure 3.1. Performing operation on 4-bit data

Array multiplier is a method for doing bit-by-bit multiplication. After being created and saved in memory, partial products are then passed on to the array of summation.

Prior to its implementation, dedicated memory space is required since it operates on data in array form. The use of a carry propagation adder for array multiplication has been detailed by Broun. The pattern looks to be a large pour of HA and FA, and it is consistent across all bit combinations, as seen below:

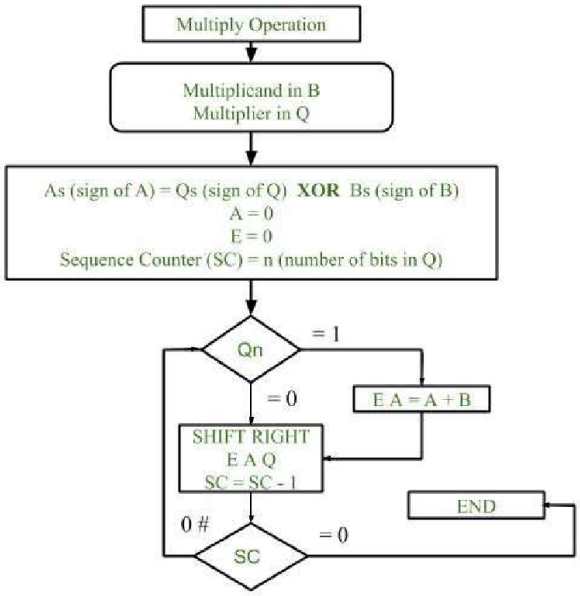

Figure 3.2. Flow chart

The first configuration involves storing the multiplicand in the B register and the multiplier in the Q register.

1. In the beginning, the multiplicand is kept in the B register and the multiplier in the Q register.

2. The XOR functionality is used to compare the signs of registers B (Bs) and Q (Qs). If the signs are same, the output of the XOR operation is 0, otherwise it is 1. The result is saved in As (the sign of the A register).

Please take note that the initial value for register A and the E flip-flop is 0. The number of bits in the multiplier is represented by the beginning value of the sequence counter, which is n.

Third, we examine the least significant bit of the multiplier. The result is allocated to the A register using the carry bit in flip-flop E if it is 1, and the contents of register B are added with the multiplicand. A one-position rightward shift is applied to the contents of E, A, and Q, meaning that the content of E is moved to the most significant bit (MSB) of A and the least significant bit of A is moved to the most significant bit of Q.

4. In the event that Qn equals zero, the shift right operation is applied only to the content of E A Q.

5. A value of 1 is subtracted from the Sequence counter.

6. Verify the value of the Sequence counter (SC). If it is 0, then proceed to complete the operation. If not, then repeat the procedure with the final result in registers A and Q.

IV. PROJECT METHODOLOGY AND WORKING EXPLANATION

Using Vivado Design Suite, the MPA multiplication VHDL code is built for the Artix-7 FPGA [15]. The designed multiplier does the multiplication operation on arrays of MPA numbers using many multiplying units that act in simultaneously. There is a single DSP48E1 module in this series of FPGAs that makes up each multiplication unit [16]. As the number of units that need to be multiplied increases, the created multiplier can easily handle it. This implies that the multiplier's number of multiplying units may be adjusted to meet different processing demands. Due to space constraints, we only provide a single multiplier that can multiple two arrays of integer MPA numbers in this contribution. By applying the same logic to floating-point multiplication, we may get the result in a mantissa array of fixed precision (i.e., fixed-length) by factoring in exponents and rounding.You may code it to compute various arithmetic functions using the DSP48E1 module that is included in Artix-7 FPGA.

Nevertheless, the A · B + C + CARRY IN operation is the only one used in this project.

Here is the most extensive list of operand widths: Type A: 25 bits, Type B: 18 bits, and Type C: 48 bits • Input - one bit. The symmetric operands are chosen because the asymmetric operand multiplier, which has 25 bits by 18 bits, is challenging to build and store data with (i.e., it stores partial results in an accumulator). Hence, 18 × 18 bits should be the maximum width of operands in the multiplying unit. One drawback of using a non-electronic number (one that isn't a power of 2) is that it reduces the utilisation of the FPGA chip's BRAM memory block.

As a result, the operand C has a size of 32 bits and its width is set to 16 x 16. Such a DSP module setup puts the delay at 4 clock cycles for excellent performance. A single extra clock cycle is needed to process the data coming into and going out of the DSP module.

In addition, a register is necessary for the data input multiplexer to keep operating at a high speed. at a result, the minimal delay is defined at 6 clock cycles, the figure that was ultimately settled upon throughout design. Thus, the latent.

The power of FPGA is derived from its adaptability. Although designing anything is possible, it might be a time-consuming process. My goal in this research is to demonstrate the versatility and scalability of field-programmable gate arrays (FPGAs). It takes time to multiply, and it may occasionally determine the system's overall performance. The power of digital signal processor (DSP) chips is compared with their multiplication and addition (MAC) numbers in a second because the impulse response or Fourier transform of the discrete signal is computed by multiplying and adding a large number of samples. Applications requiring massive MAC operations, such as adaptive noise cancellations, machine vision, HDTV, military radar, and many more, find FPGAs, which provide bespoke solutions, useful. On FPGA, there are a lot of methods to do the multiplication. When speed is paramount, it may be done in a parallel or pipelined fashion; otherwise, it can be designed with a minimal footprint to keep costs down.

I will use a 32*32 multiplier for the test since the DE0-Nano device has 149 programmable pins. More bits for multiplication are possible, but they cannot be assigned to the pins. Therefore, I just employ the multiplier design; this is a demonstration of many techniques.

V. PROJECT RESULT

Discuss the design and implementation of the suggested model of a 2*2 binary multiplier in this area. A Xilinx programming mimic that is driven by outline information. With version 14.7, Xilinx has released its programming system**.**

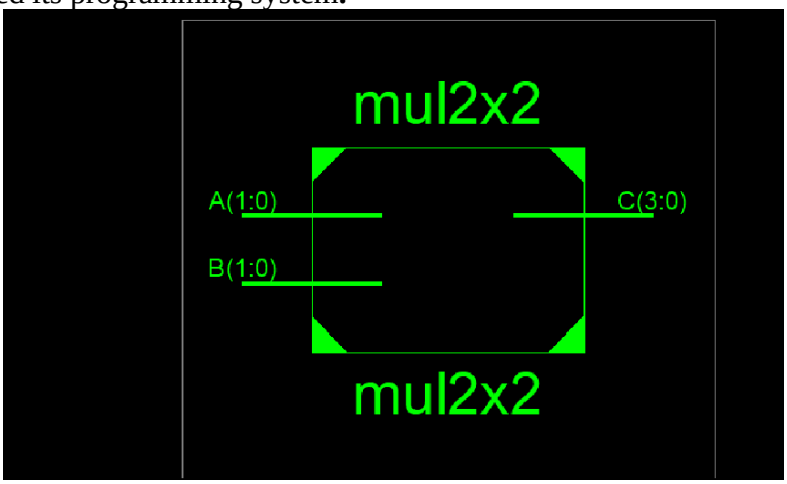

Figure 5.1. Half adder

RTL in the circuit design cycle. RTL is used in the logic design phase of the integrated circuit design cycle. An RTL description is usually converted to a gate-level description of the circuit by a logic synthesis tool. The synthesis results are then used by placement and routing tools to create a physical layout. The Fig 1,Fig 2 and Fig 3 represents the RTL schematic of 2*2 binary multiplier. In which it gives the information regarding logic of the design in the form of symbols like adders , multipliers.

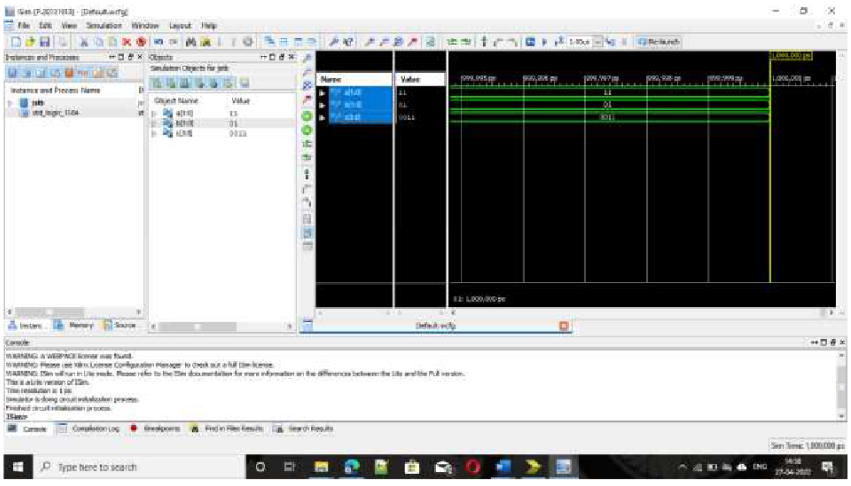

Figure 5.2. simulation timing diagram of 2*2 binary multiplier

When to Run the Simulation, You can find out how long the simulation took by looking at the diagram that shows the duration of the clock signal. The 2*2 multiplier's simulation outcome with respect to a and b is shown in figure 4.A and b are inputs, and c is the result. Then, after receiving the input, hold off for 100 nanoseconds.

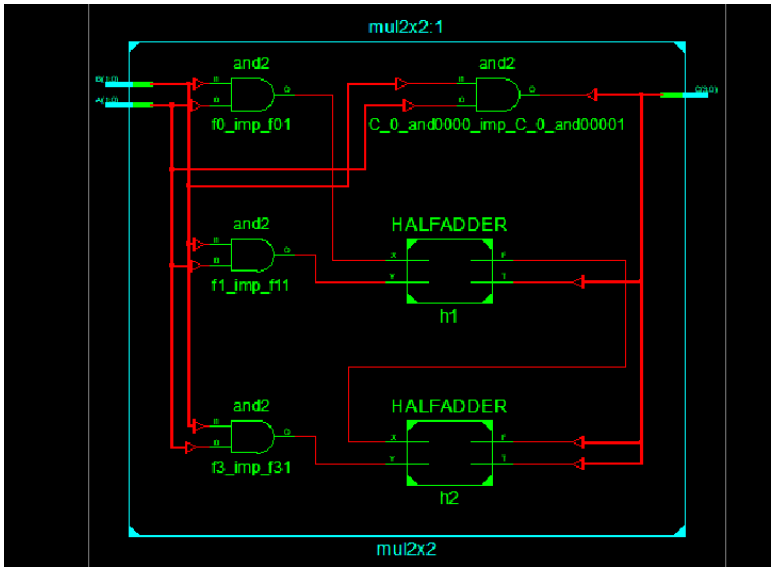

Figure 5.2. Test-bench RTL Schematic Diagram

This multiplier makes use of the shift-and-add approach, which is coded in VHDL using Xilinx 14.5 and then implemented on the XC3S500E device. The operands must be 8 bits wide for the multiplier to work. A 2-bit parallel prefix adder is used to perform the addition function. An assortment of parallel prefix adder variations, including BK, Skalansky, KS, HC, LF, Knowles models, and the suggested hybrid model, are used to evaluate the Multiplier block's performance. Based on several variations of the parallel prefix adder, the results of the Multiplier block are summarised. Three metrics are used for comparison: speed, area, and power usage. The delay parameter is used to measure the speed performance.

VI. CONCLUSSION

Ultimately, this research work has explored how to perform the multiplication operation using Field Programmable Gate Arrays (FPGAs). It becomes clear that FPGAs provide a potential platform for effectively performing multiplication operations in many computer applications after examining their designs, programming approaches, and optimisation strategies in depth. The research emphasises the benefits of using field-programmable gate arrays (FPGAs), including its capacity for parallel processing, ease of reconfiguration, and little power consumption. Experts in the field may use these features to create and execute efficient multiplication processes that use

little power. Flexible gate arrays (FPGAs) are a flexible solution for a wide variety of computing tasks, from signal processing to cryptographic algorithms, thanks to their versatility.

In addition, the study highlights the significance of optimisation techniques to boost the effectiveness of FPGA multiplication processes. Achieving ideal performance is greatly aided by techniques like pipelining, parallelization, and algorithmic optimisations. It is expected that future advancements in FPGA design and programming tools will lead to even more efficient and scalable multiplication implementations as technology progresses. Finally, digital design and computing professionals and academics have a promising new direction to pursue using FPGAs while performing multiplication operations.

References:

[1]. Z.Huang, "High-Level optimization Techniques for Low power Multiplier Design", PhD dissertation, University of California, LOS Angeles, June 2003

[2]. L.Raja, B.M.Prabhu, K.Thanushkodi, "Design of low power Dual Threshold Voltage Adder Module", Elsevier, International Conference on Communication Technology and System Design, 2011.

[3]. MohmmadJaveed, GellaRavikanth, "Design and Implementation of 64 BIT Multiplier by using carry save Adder", Proc. 10th IRF International Conference, pp. 45-47, Oct.2014.

[4]. Sukdev Singh, Puneeth Jain, Pankaj Sharma, RamandeepChalal, "Design and synthesis of various Multipliers using VHDL: Performance Analysis Approach", International Journal of Electronics and Computer Science Engineering, Vol.3, N0.3, pp.339-347.

[5]. Sarita Singh and Sachin Mittal, "VHDL design and Implementation for optimum Delay & Area for Multiplier and Accumulator unit by 32 bit- sequential Multiplier", International Journal of Engineering Trends and Technology(IJETT), Vol.3, No.5, pp.683-686, 2012.

[6]. Ruchi Sharma, "Analysis of Different multiplier with Digital Filters using VHDL Language", International Journal of Engineering and Advanced Technology(IJEAT),Vol.2, No.1, pp.45-48, Oct.2012.

[7]. K.S.Ganesh Kumar, J.DevaPrasannam&M.Anitha Christy, "Analysis of Low power Area and High Speed Multipliers for DSP applications", International Journal of Emerging Technology & Advanced Engineering (IJETAE), Vol.4, No.3, pp.278- 282, March 2014.

[8]. Giovanni D Aliesio, "8-by-8 Bit Shift/Add Multiplier", Digital Design and & synthesis COEN 6501, Department of Electrical & Computer Science Engineering, Concordia University, Dec 2003