High Performance FPGA-Based Decimal to Binary Conversion Schemes for Decimal Arithmetic

Sagar Kuthe
Student, Electronics and Telecommunication Engineering
Smt. Indira Gandhi College of Engineering, Navi Mumbai, Maharashtra, India
kuthesagar1299@gmai.com

P. Fizza Rao
Student, Electronics and Telecommunication Engineering
Smt. Indira Gandhi College of Engineering, Navi Mumbai, Maharashtra, India
fizzp2002rao@gmail.com

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

Preeti Sahu
sahupreeti20324@gmai.com
Student, Electronics and Telecommunication Engineering
Smt. Indira Gandhi College of Engineering, Navi Mumbai, Maharashtra, India

****

ABSTRACT:

While computers are best suited for binary arithmetic, humans are better suited to decimal arithmetic. Conversion methods fill this need. In order to facilitate decimal arithmetic, this study introduces high-performance circuits that convert decimals to binary. The BCD input bits are arranged into many sets. All of the groups' contributions to the final binary outcome are calculated independently. The final binary result is formed by adding these contributions. In comparison to other BCD-to-binary conversion circuits, the suggested ones work much better. Some bit-groupings may work better than others, as shown by the comparison. The Xilinx FPGA platform is used to create this BCD to Binary conversion circuit. For BCD inputs, the area and delay increase of this circuit is examined.

Keywords – Arithmetic, Binary Number, FPGA, Multiplication, Digital VLSI Design, SoC

I. INTRODUCTION

Financial, web-based, and scientific applications are only a few examples of modern applications that deal with decimal operands. When working with decimal operands, two main methods may be used for calculation. By working directly with the decimal operands, they may lessen the likelihood of rounding mistakes. Taking the decimal operands and converting them to binary, then doing the necessary math in binary, and then converting it back to decimal. The second approach has the benefit of making use of the widely used binary arithmetic hardware. In addition, the use of the already optimised binary hardware necessitates solutions such as the second approach in certain applications. To back up decimal arithmetic, we provide many high-performance designs for converting decimals to binary in this work. To get to the end result, we divide the BCD input into many groups of bits and then calculate their contributions. Consider the following examples: (346)10 or (0011 01000110) BCD, the BCD operand that has to be translated to binary. The binary output with a group size of 4 bits (one BCD digit) includes the digit 6 (011102). For example, (101000) is the result of adding 40 and 10.2. The value of (300)10 or (100101100) is affected by the digit 3.2. We get the final result by adding the contributions of each digit in the BCD operand. Thus, in binary, (101011010)2= (34610)10 is the consequence of adding (01102)2, (101000)2, and (100101100)2. This method outperforms previous methods for translating between binary and BCD on FPGAs, despite its seeming simplicity.

This exceptional performance is due to a number of factors. To begin, the FPGA computes all the contributions simultaneously, and then it adds them using binary addition, creating a fast circuit. Secondly, there are a certain number of variables that are directly proportional to the group size that determine the outputs of the contribution generating circuit for a particular group. The results of the group contribution generation, for instance, depend on the four bits that make up a 4-bit grouping. Each function in the FPGA only needs one look-up table (LUT), leading to a compact overall design, when the group size is chosen to match the look-up table size on an FPGA family.The fundamental idea behind this approach is to divide the input BCD number into sets of successive bits, starting from the least significant position and working their way up to the most significant. For instance, in a set of M groups, G0 would represent the least significant group and GM-1 would represent the most significant group. Each group's binary contribution changes according on its index or position. Digital circuitry that takes the group's bits as inputs and the binary contribution's bits as outputs creates it. In order to get the binary representation of the input BCD number, a binary addition step is used to combine the binary contributions from each group. When going from binary to binary, there are three distinct bit-grouping algorithms to choose from. There are three different types of bit grouping: 4-bit, 6-bit, and 8-bit. To code these methods, one uses Verilog dataflow modelling. We begin by evaluating the current state of the art in 4-bit grouping (one BCD digit grouping) techniques. Next, we use the FPGA partan 3-XC3S200-2 FT256 device on the Xilinx platform to assess the performance of several grouping schemes. These schemes include 4-bit, 6-bit, and 8-bit versions, each having 1BCD digit, 1.5 BCD digit, and 2 BCD digits, respectively.

II. LITERATURE SURVEY

The two main ways to do decimal arithmetic are(1) using binary hardware for BCD arithmetic and(2) directly manipulating BCD integers.

The people behind Decimal multiplication utilising compact BCD multiplier and Binary coded decimal digit multipliers both worked on architectures for BCD digit by BCD digit multiplication. The authors of two papers published in 2009—"A high performance unified BCD and binary adder/subtractor" and "High Speed multi-operand decimal adders"—proposed methods for working with several operands simultaneously.There are a number of binary-based multipliers, including those for iterative decimal multiplication and FPGA-based decimal multipliers that use integrated binary multipliers. These designs illustrate BCD arithmetic using binary hardware; to do this, we transform the BCD number to its binary counterpart, then execute the arithmetic in binary, and then convert the output back to BCD. Among the methods they emphasised were the tried-and-true method of serial division by 2. This method involves testing each BCD digit in the shifted number after shifting the BCD input one bit to the right. If the bit-coded decimal value is 8 or higher, either deduct 3 from the digit [8, 11] or clear the most significant bit and add 5 to the digit. Once all bits have been created, the operation is repeated. One further way to get the binary representation of a BCD number is to use direct calculation using the following binary formula:

With respect to Dn-1, the value of D is equal to D.half of n-1Add D0100 to D3103+D2102+D1101+10(n-1). One drawback of using this technique to convert BCD numbers is that it needs powers of 10 to be multiplied. The size of the multipliers required increases rapidly with the number of digits.

This is how the authors rephrase the above equation according to Horner's rule: The sum of all functions D is 10 plus D0 plus ((Dn-110+Dn2)10+.......). To further improve computing parallelism, the authors of [8] provide the following alternative formulations of this formula: If you multiply (Dn-110+Dn-2100+......) by 100, then add D110+D0 to get D. D is equal to the sum of (Dn-1100, Dn-210, and Dn-3).103+(D2102+D110+D0) adds up to 103+(D0+D110+D202).

D is equal to the sum of all possible values from 1 to 104 plus (D3103+D2102+D110+D0)).

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.

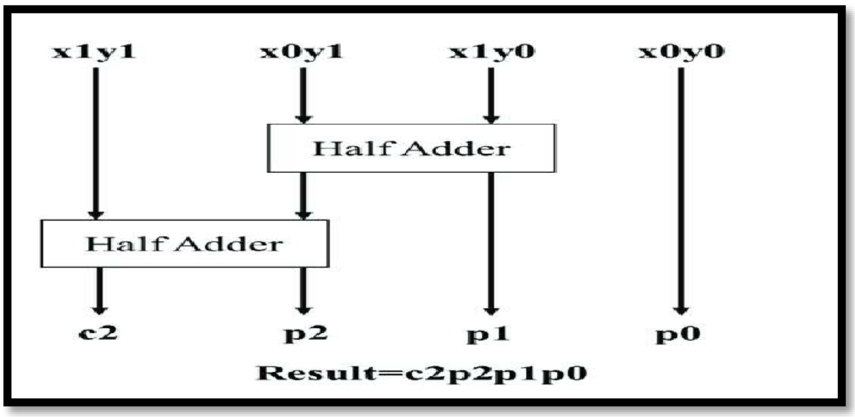


Figure 3.1. Performing operation on 4-bit data

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:

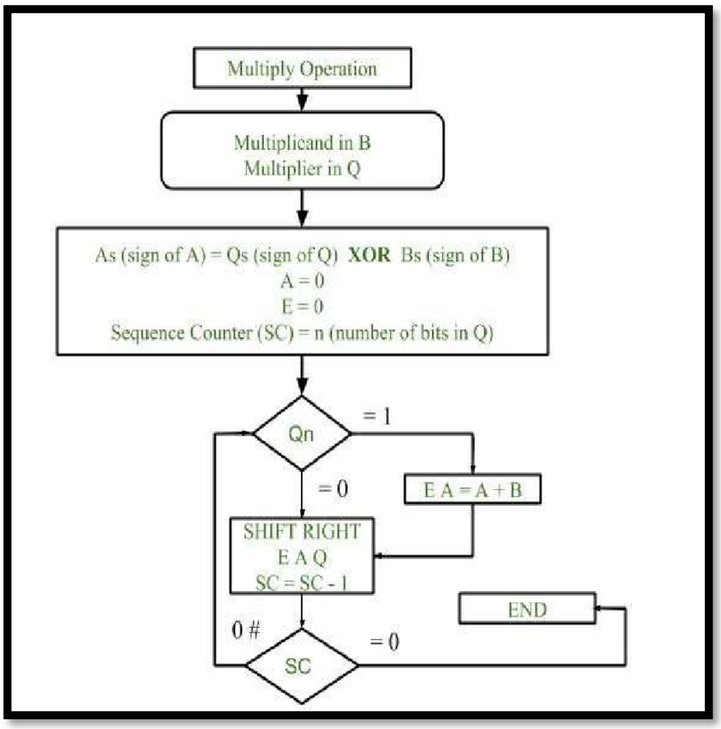


Figure 3.2. Flow chart

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.

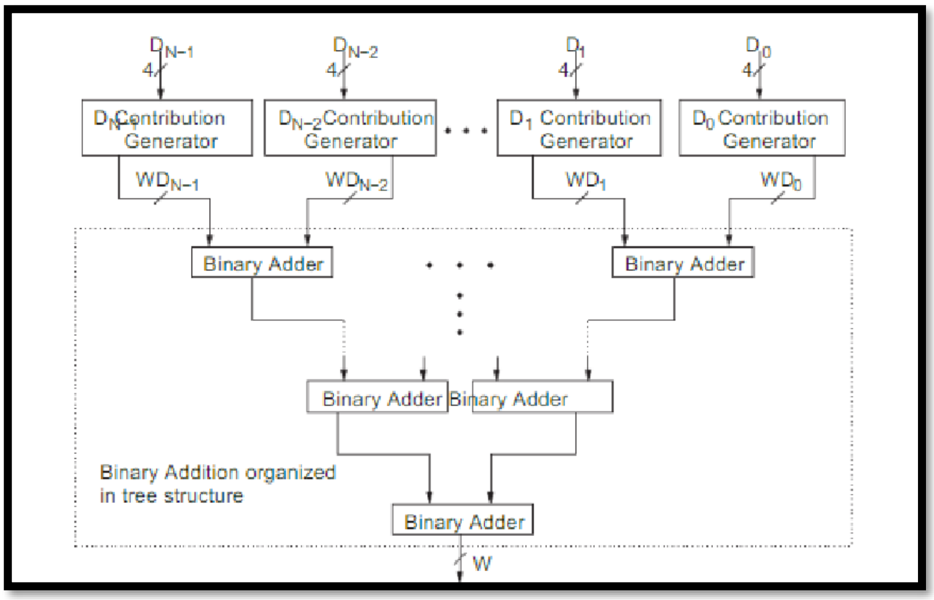

D_{N-1}
4
D_{N-2}
4
D_1
4
D_0
4
D_{N-1} Contribution Generator
D_{N-2} Contribution Generator
D_1 Contribution Generator
D_0 Contribution Generator
WD_{N-1}
WD_{N-2}
WD_1
WD_0
Binary Adder
Binary Adder
Binary Adder
Binary Adder
Binary Adder
Binary Addition organized in tree structure
W

Figure 4.1. Architecture fourth grouping scheme

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.

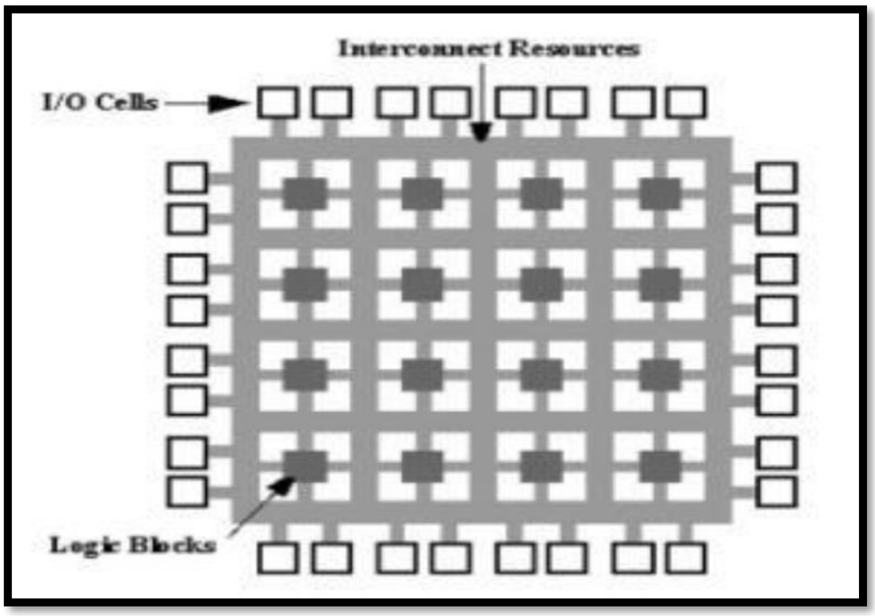


Figure 4.2. FPGA Architectural diagram presentation

Each group in this method is 6 bits (or 1.5 BCD digits) in size. There are N BCD digits in the input, labelled as DN-1, DN-2,... D1 D0. G0, G1, G2..., Gm are the group designations, with m=[2N/3-1]. For any positive integer i, the group Gi's least significant bits are the BCD digit D3i/2. Contrarily, the two least important bits of the BCD digit D3i/2+1 are the most significant bits of group Gi. The most important four bits of group Gi are the BCD digit D 3i-1/2+1, whereas the least significant two bits of group Gi are made of the most significant two bits of the BCD digit D 3i-1/2 for an odd integer i.

Consequently, in binary, they can only be 00, 01, 10, 11, or in decimal, 0, 1, 2, 3, but in BCD, the least important four bits may be any number from zero to nine.

When i is an odd integer, however, the maximum possible decimal value of the set is 98.103i + 12. This occurs because the two bits that are least important in the group are derived from the two bits that are most important in a BCD digit. So, in binary they may be 00, 01, or 10, in decimal they can be 0, 4, or 8, and in BCD they can be any number from 0 to 9.

Note that depending on the input BCD digits, the most significant group size in this approach may be 6 bits, 4 bits, or 2 bits.

This method uses a group size of 8 bits, which is equivalent to 2 BCD digits. Here, [N/2] is the number of groups. Each member of Group Gi is made up of two binary code digits: the right-hand member, D2i, and the left-hand member, D2i+1. Gi may be expressed as 99x102i, the biggest integer representation of it. Given that WGi is the size of the binary equivalent of group Gi, we may get WGi as [log2(99x 102i)] + 1, where 2i is the number of zeroes in the binary version of group Gi. This scheme's most important group may be either 8 bits or 4 bits in size, depending on the amount of BCD digits in the input.

V. 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**.**

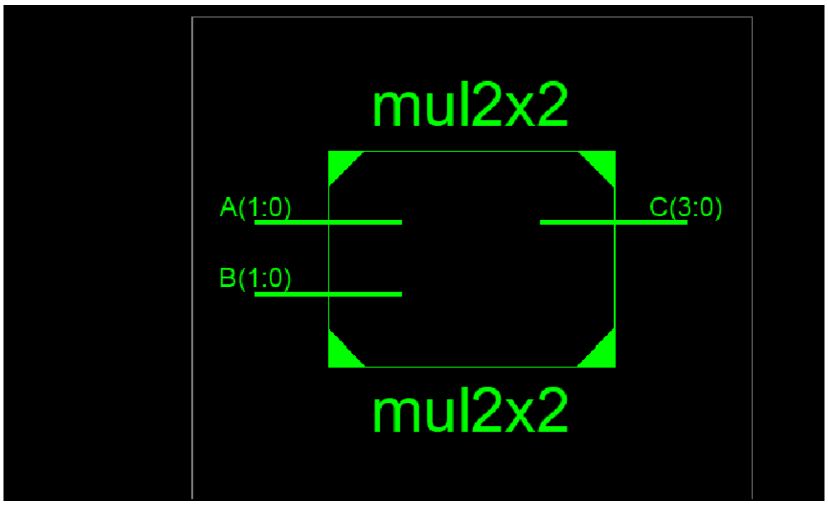
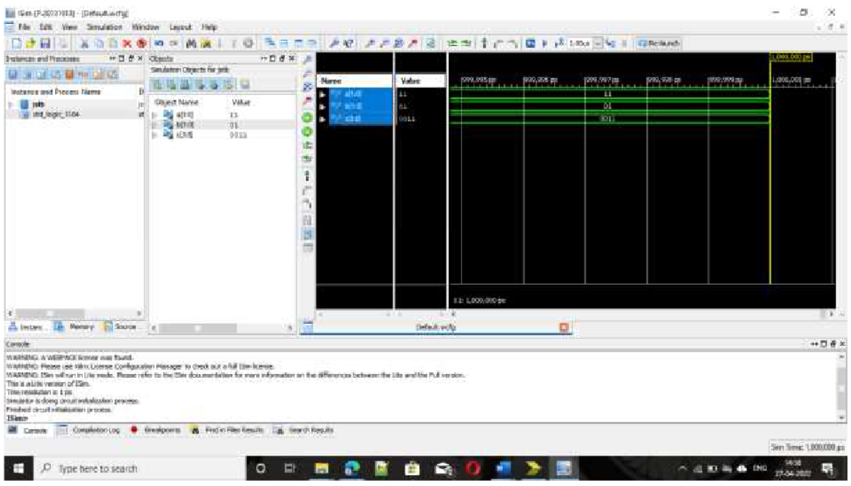


Figure 5.1. Half adder

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.



```
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

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.

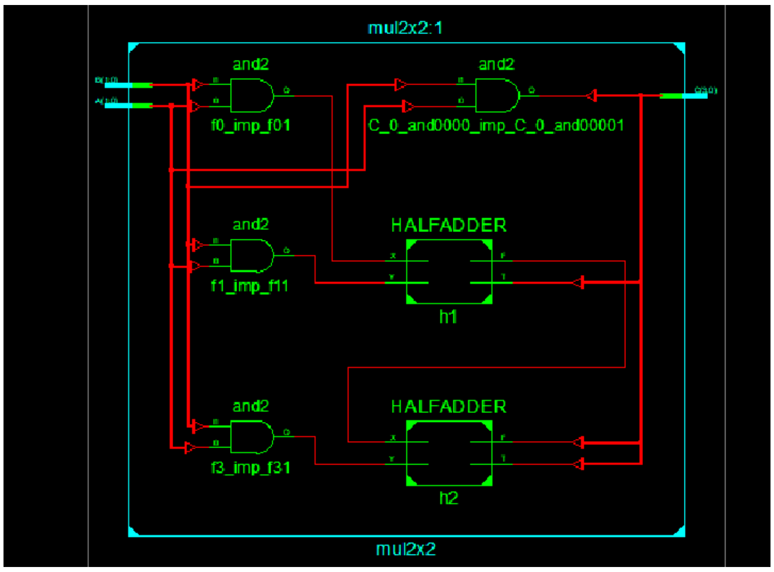


Figure 5.3. Test-bench RTL Schematic Diagram

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.

Figure 5.4. Actual project diagram

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 summarized. Three metrics are used for comparison: speed, area, and power usage. The delay parameter is used to measure the speed performance.

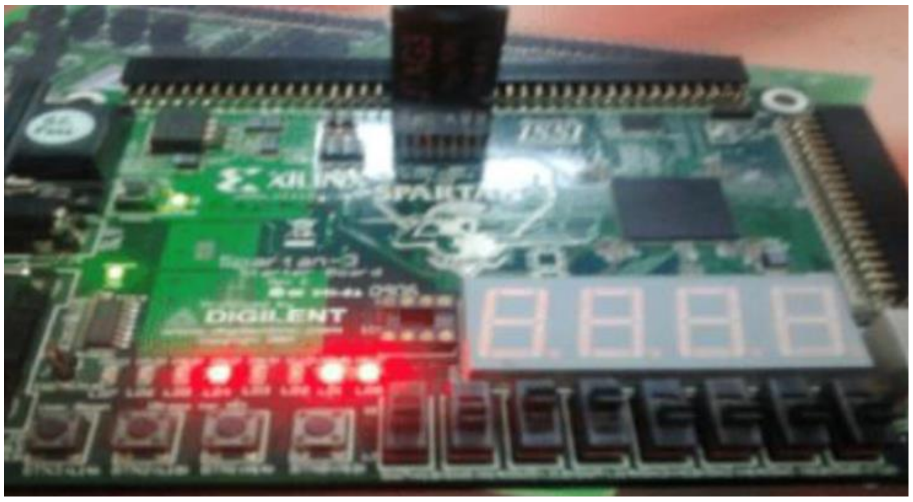


Figure 5.5. Implementation representation

The suggested 4-bit routing (one BCD digit grouping) system is first compared to various current schemes in terms of performance in this section. We next compare the efficiency of the various grouping methods on various FPGA devices equipped with 4-input and 6-input LUTs. The schemes include 4-bit, 6-bit, and 8-bit, each representing a different number of bits.Since the majority of current schemes are built on 4-input LUT FPGAs, we assume that 4-bit grouping will provide the best results in this scenario, thus we choose it to compare with other schemes. Along with six other preexisting techniques for BCD-to-binary conversion, we have also incorporated the 4-bit grouping scheme.

VI. CONCLUSSION

Our proposed method for supporting binary-based BCD arithmetic is a high-performance FPGA-based decimal-to-binary conversion technique. Using shifters instead of multipliers helps minimize latency and the number of LUTs used by the design compared to others. We provide a variety of efficient decimal-to-binary conversion algorithms that may be used to enable binary-based BCD arithmetic. In order to get the final binary output, we first divide the BCD input into many groups of bits and then calculate the binary contribution of each group. Then, FPGAs use binary addition to combine the contributions. This decision allows for a small design since every function that calculates a group's contribution from the circuit's outputs fits precisely into one look-up table.

Additionally, when the input BCD digit count increases, we have addressed in this study the area and delay growth of the suggested techniques on different FPGA generations. We may say in broad strokes that area rises exponentially whereas delay grows logarithmically.

References:

[1]. A. Vazquez, E. Antelo, P. Moutushi, A new family of high-performance parallel decimal multipliers, in: 18th IEEE Symposium on Computer Arithmetic, 2007, ARITH '07, pp. 195–204

[2]. Sukdev Singh, Puneeth Jain, Pankaj Sharma, Ramandeep Chalal, "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.

[3]. 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.

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

[5]. 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.

[6]. Mohmmad Javeed, Gella Ravikanth, "Design and Implementation of 64-BIT Multiplier by using carry save Adder", Proc. 10th IRF International Conference, pp. 45-47, Oct.2014.

[7]. 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.

[8]. K.S. Ganesh Kumar, J. Deva Prasannam & 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.

[9]. 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

[10]. R. Jaikumar, P. Poongodi and R. Lavanya," Implementation of high speed arithmetic logic using vedic mathematics techniques" ictact journal on microelectronics, february 2015

[11]. M. Ramalatha, K. Deena, Dayalan ,Dharani "High Speed Energy Efficient ALU Design usingVedic Multiplication echniques" ACTEA IEEE 2009.

[12]. N. Petra, D. D. Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo, "Design of fixed width multipliers with linear compensation function", IEEETrans. Circuits Syst., vol. 58, no. 5, pp. 947960, May 2011.

[13]. Jiun-Ping Wang, Shiann-RongKuang, and Shish-Chang Liang, "High-Accuracy Fixed-Width Modied Booth Multipliers for Lossy Applications", IEEE Trans. Circuits Syst., vol. 19, no. 1, pp. 52-60, January 2011.

[14]. Yuan-Ho Chen, T.-Y. Chang, and C.-Y. Li, "Area-Effective and Power-Efficient Fixed-Width Booth Multipliers Using Generalized Probabilistic Estimation Bias", IEEE Trans. Circuits Syst., vol. 1, no. 3, pp. 277-287, September 2011.

[15]. Yuan-Ho Chen and T.-Y. Chang, "A High-Accuracy Adaptive Conditional-Probability Estimator for Fixed-Width Booth Multipliers", IEEE Trans. Circuits Syst., vol. 59, no. 3, pp. 594-603, March 2012.

[16]. Shin-Kai Chen, Chih-Wei Liu, "Design and Implementation of High-Speed and Energy-Efcient Variable-Latency Speculating Booth Multiplier (VLSBM)", IEEE Trans. Circuits Syst. I, vol. 60, no. 10, pp. 26312643, October 2013.

[17]. Shen-Fu Hsiao, Jun-Hong Zhang Jian, and Ming-Chih Chen, "Low-Cost FIR Filter Designs Based on Faithfully Rounded Truncated Multiple Constant Multiplication/Accumulation", IEEE Trans. Circuits Syst. II, Express Briefs, 2013.

[18]. O. L. MacSorley, "High speed arithmetic in binary computers", Proc.IRE, vol.49,pp. 67-91, 1961.

[19]. Parhami, Behrooz, "Computer Arithmetic: Algorithms and Hardware Designs", Oxford University Press 2000.

[20]. David H. K. Hoe, Chris Martinez and Sri JyothsnaVundavalli, "Design and Characterization of Parallel Prefix Adders using FPGAs", Proc. IEEE, pp. 168172, 2011

[21]. Srinivasasamanoj R. ,M. Sri Hari and B. RatnaRaju, "High speed VLSI implementation of 256-bit Parallel Prefix Adders", International Journal of Wireless Communications and Networking Technologies, vol. 1, no. 1, 2012.