National Journal of Advanced Research
Online ISSN: 2455-216X; Impact Factor: RJIF 3.42
www.allnationaljournal.com/njar
Volume 2; Issue 1; January 2016; Page No. 25-29

Reduced control lines in multiplexer performing the ALU operation using QLUT

[1] Karthika P, [2] Solomon Deva Doss S
[1] Student, Applied Electronics Department, Infant Jesus College of Engineering, Tamil Nadu, India
[2] Assistant Professor, ECE Department, Infant Jesus College of Engineering, Tamil Nadu, India

Abstract
Digital design is an astounding and very massive field. The applications of digital design are exist in our day to day life, containing calculators, computers, Mobiles, video cameras etc. The Aim of this paper is designing ALU and Reversible ALU. The ALU is mainly utilized in computer and is employed during most instruction executions. Hence the power consumption of the ALU is a major concern. Binary Digit Numbers are known to allow limited carry propagation with extra complex addition operation. Limitations of this binary system are computational speed which limits formation and propagation of carry specifically when the number of bits increases. Design of binary logic circuits is easily possible when the interconnection is less. As the immense number of interconnection requirement has become a major limitation to the designs using binary logic. To overcome this problem by using Multiple- Valued Logic (MVL). Quaternary logic proves to be advantageous as it reduces dynamic power consumption, increases computational ability, data density and requires less number of interconnects. So that the arithmetic unit design this gives better results than the binary circuits. It also produces higher information storage density, more exact arithmetic operations and less complexity. This ALU consists of ten operations, four arithmetic and six logical operations. The arithmetic operations include addition, subtraction, division and multiplication and the logical operations include NAND, AND, OR, NOT, shift and Rotation. Reversible ALU or Information-lossless circuits have applications in communication, computer graphics, digital signal processing and cryptography. Reversibility used to prevents loss of information and energy efficient computations are considered. It operates at 195MHz frequency at the same time as power consuming 52mW using Xilinx14.2 VHDL and the simulation results using ModelSim.

Keywords: Quaternary Logic, Reversible ALU, Digital Design, VHDL

1. Introduction
In past year, VLSI designer face the main challenge was to cut back chip area by using efficient optimization techniques [1]. Now, as most of commercial electronic products are portable like mobile, laptops etc. These require more battery backup. So, lot of research is going on to reduce power consumption. Then the next phase is to increase the speed of operation to achieve fast calculations as, in today's electronic equipment millions of instructions are executed per second. Speed of operation is one of the major constraints in designing DSP processors. Now, as most of commercial electronic products are portable like mobile, laptops etc. These require more battery backup. So, lot of research is going on to reduce power consumption. Thus, there are three performance parameters on which a VLSI designer has to optimize their design i.e. area, speed and power. It is very difficult to achieve all constraints for particular design, therefore depending on demand or application some compromise between constraints has to be made. The new approach is to solve the area problems using multiple-valued logic (MVL) inside the VLSI chip. Power dissipation has become the main area of concern in VLSI design. Reversible logic has its basics from thermodynamics of information processing. According to this, traditional irreversible circuits generate heat due to the loss of information during computation. In order to avoid this information loss the conventional circuits are modelled using reversible logic. Landauer [1961] showed that the circuits designed using irreversible elements dissipate heat due to the loss of information bits [2]. It is proved that the loss of one bit of information results in dissipation of KT*log2 joules of heat energy where K is the Boltzmann constant and T is the temperature at which the operation is performed. Benett [1973] showed that this heat dissipation due to information loss can be avoided if the circuit is designed using reversible logic gates [3]. A gate is considered to be reversible only if for each and every input there is a unique output assignment. Hence there is a one to one mapping between the input and output vectors. A reversible logic gate is an n-input, n-output device indicating that it has same number of inputs and outputs. A circuit that is built from reversible gates is known as reversible logic circuit. In this paper, we design a 16 bit reversible ALU that can perform ten operations, four arithmetic and six logical operations. The arithmetic operations include addition, subtraction, division and multiplication and the logical operations include NAND, AND, OR, NOT, shift and Rotation. All the modules are simulated in ModelSim SE 6.5 and synthesised using Xilinx ISE 14.2. This paper is organized as follows. In Section II, we review the basic concepts of ALU. Then, in Section III and a overview of binary and quaternary LUTs. In section IV, the quaternary LUT (QLUT) topologies with the circuits proposed are discussed. Section V presents the simulation results of the proposed design. Finally, Section VI summarizes and concludes this paper.

2. ALU Description
An arithmetic logic unit (ALU) is a digital electronic circuit that performs arithmetic and bitwise logical operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers.

An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

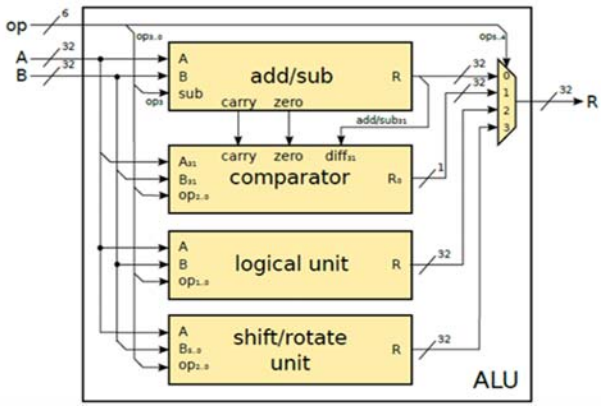


Fig 1: The internal architecture of the ALU.

The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also exchanges additional information with a status register, which relates to the result of the current or previous operations.

An ALU is a combinational logic circuit, meaning that its outputs will change asynchronously in response to input changes. In normal operation, stable signals are applied to all of the ALU inputs and, when enough time (known as the "propagation delay") has passed for the signals to propagate through the ALU circuitry, the result of the ALU operation appears at the ALU outputs. The external circuitry connected to the ALU is responsible for ensuring the stability of ALU input signals throughout the operation, and for allowing sufficient time for the signals to propagate through the ALU before sampling the ALU result.

In general, external circuitry controls an ALU by applying signals to its inputs. Typically, the external circuitry employs sequential logic to control the ALU operation, which is paced by a clock signal of a sufficiently low frequency to ensure enough time for the ALU outputs to settle under worst-case conditions.

For example, a CPU begins an ALU addition operation by routing operands from their sources (which are usually registers) to the ALU's operand inputs, while the control unit simultaneously applies a value to the ALU's opcode input, configuring it to perform addition. At the same time, the CPU also routes the ALU result output to a destination register that will receive the sum. The ALU's input signals, which are held stable until the next clock, are allowed to propagate through the ALU and to the destination register while the CPU waits for the next clock. When the next clock arrives, the destination register stores the ALU result and, since the ALU operation has completed, the ALU inputs may be set up for the next ALU operation.

Function

A number of basic arithmetic and bitwise logic functions are commonly supported by ALUs. Basic, general purpose ALUs typically include these operations in their repertoires:

Arithmetic operations

- Add: A and B are summed and the sum appears at Y and carry-out.
- Add with carry: A, B and carry-in are summed and the sum appears at Y and carry-out.
- Subtract: B is subtracted from A (or vice-versa) and the difference appears at Y and carry-out. For this function, carry-out is effectively a "borrow" indicator. This operation may also be used to compare the magnitudes of A and B; in such cases the Y output may be ignored by the processor, which is only interested in the status bits (particularly zero and negative) that result from the operation.
- Subtract with borrow: B is subtracted from A (or vice-versa) with borrow (carry-in) and the difference appears at Y and carry-out (borrow out).
- Two's complement (negate): A (or B) is subtracted from zero and the difference appears at Y.
- Increment: A (or B) is increased by one and the resulting value appears at Y.
- Decrement: A (or B) is decreased by one and the resulting value appears at Y.
- Pass through: all bits of A (or B) appear unmodified at Y. This operation is typically used to determine the parity of the operand or whether it is zero or negative.

Bitwise logical operations

- AND: the bitwise AND of A and B appears at Y.
- OR: the bitwise OR of A and B appears at Y.
- Exclusive-OR: the bitwise XOR of A and B appears at Y.
- One's complement: all bits of A (or B) are inverted and appear at Y.

Bit shift operations

ALU shift operations cause operand A (or B) to shift left or right (depending on the opcode) and the shifted operand appears at Y. Simple ALUs typically can shift the operand by only one bit position, whereas more complex ALUs employ barrel shifters that allow them to shift the operand by an arbitrary number of bits in one operation. In all single-bit shift operations, the bit shifted out of the operand appears on carry-out; the value of the bit shifted into the operand depends on the type of shift.

- Arithmetic shift: the operand is treated as a two's complement integer, meaning that the most significant bit is a "sign" bit and is preserved.
- Logical shift: a logic zero is shifted into the operand. This is used to shift unsigned integers.
- Rotate: the operand is treated as a circular buffer of bits so its least and most significant bits are effectively adjacent.
- Rotate through carry: the carry bit and operand are collectively treated as a circular buffer of bits.

Complex operation

Although an ALU can be designed to perform complex functions, the resulting higher circuit complexity, cost, power consumption and larger size makes this impractical in many cases. Consequently, ALUs are often limited to simple functions that can be executed at very high speeds (i.e., very short propagation delays), and the external processor circuitry is responsible for performing complex functions by orchestrating a sequence of simpler ALU operations.

For example, computing the square root of a number might be implemented in various ways, depending on ALU complexity:

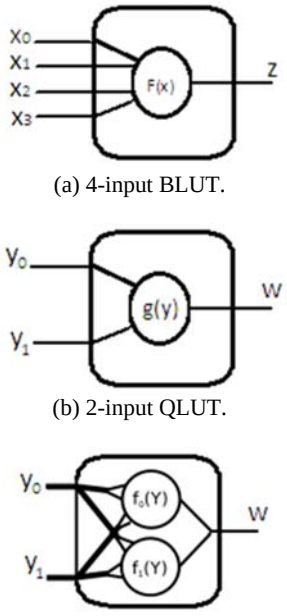
- Calculation in a single clock: a very complex ALU that calculates a square root in one operation.
- Calculation pipeline: a group of simple ALUs that calculates a square root in stages, with intermediate results passing through ALUs arranged like a factory production line. This circuit can accept new operands before finishing the previous ones and produces results as fast as the very complex ALU, though the results are delayed by the sum of the propagation delays of the ALU stages.
- Iterative calculation: a simple ALU that calculates the square root through several steps under the direction of a control unit.

The implementations above transition from fastest and most expensive to slowest and least costly. The square root is calculated in all cases, but processors with simple ALUs will take longer to perform the calculation because multiple ALU operations must be performed.

3. Overview of binary and Quaternary LUTs

Standard Lookup Tables are essentially reminiscences, which implement a given logic function. A LUT is an array indexing operator, in which the output is mapped by the input based on the configuration memory. The configuration values are initially stored in the LUT configuration memory, and according to the input, the logic value in the addressed position is assigned to the output. By means of properly programming the LUT configuration memory, the LUT can put into effect any logic function with the given quantity of inputs and outputs, making it very sensible to implement reconfigurable hardware, including FPGAs. Values are initially stored in the lookup table structure, and once inputs are applied, the logic value in the addressed position is assigned to the output. The capacity of a LUT $|c|$ is given by way of

$$|c| = nb^k \qquad (2)$$



(a) 4-input BLUT.

(b) 2-input QLUT.

(c) QLUT Function

Fig 2: Binary and quaternary lookup tables

Wherein *n* is the number of outputs, *k* is the number of inputs and *b* is the number of logic values. For example, a 4-input binary lookup table with one output is able to store $1\times 2^4 = 16$ Boolean values. For the cause of this work, only 1-output LUTs (n = 1) are mentioned on this paper.

A binary function implemented by a Binary Lookup Table (BLUT) is defined as f:$B^k \rightarrow B$. The total number of different functions $|F|$ that can be implemented in a BLUT with *k* input variables is given by

$$|F| = b^{|c|} \qquad (3)$$

Where

b = $|B|$ (i.e. b = 2 in the binary case).

Quaternary functions are basically generalizations of binary functions. A quaternary function implemented by a quaternary lookup table (QLUT) is defined as g: $Q^k \rightarrow Q$, over a set of quaternary variables Y = (Y_0; _ _ _; Y_i; _ _ _; Y_{k-1}), where the values of a variable Y_i, as the values of the function g(Y), can be in Q= {0, 1, 2 3}. As in the binary case, the number of possible function in QLUTs is given by (3), where b = 4. Figure 2b illustrates a 2-input quaternary function implemented in a QLUT. Note that the function g(Y) performs exactly the same function as the two binary BLUTs, f0(Y) and f1(Y), as depicted in Figure 2c, where f0 represents the least significant Boolean values and f1 represents the most significant ones. Since a quaternary variable y is capable of representing twice as much information as a binary variable x, we consider the cardinality of $|Q| = 2 \times |B|$ in our experiments. In other words, we count on that two binary variables may be grouped which will represent a quaternary variable. Such that technique pursuits at reducing both the whole number of connections and the number of gates.

4. Design of proposed QLUT

The proposed 2-input and 1-output QLUT is shown in Fig. 3. For this given QLUT complexity, 16 quaternary configuration inputs are necessary, one for each feasible aggregate of the two quaternary inputs. The configuration word defines the reconfigurable quaternary function. In practice, the input signals are used to select which one of the configuration inputs is attached to the output. The principle concept of the paper proposed is to reduce the interconnection present in the existing circuit. The combinatorial block logic capabilities will assist in understanding the function and switch operation inside the circuit proposed. The subsequent table.2 gives the better readability of the signal fetching and switches operations. The proposed QLUT consists of two main blocks: a 16-1 multiplexer using an array of switches, that establishes a low-resistance path between one configuration input and the output according to the input values; and a quaternary-to-binary decoder, including of a 2-bit analog-to-digital (ADC) frontend followed by combinational logic used to generate the control signals feeding the multiplexer. These blocks are described within the following sections.

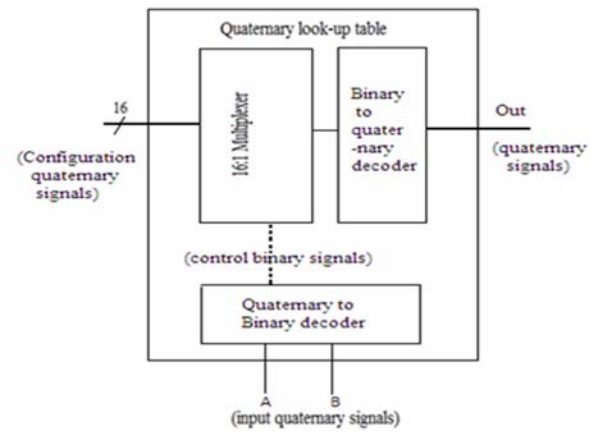


Fig 3: Proposed quaternary LUT overview

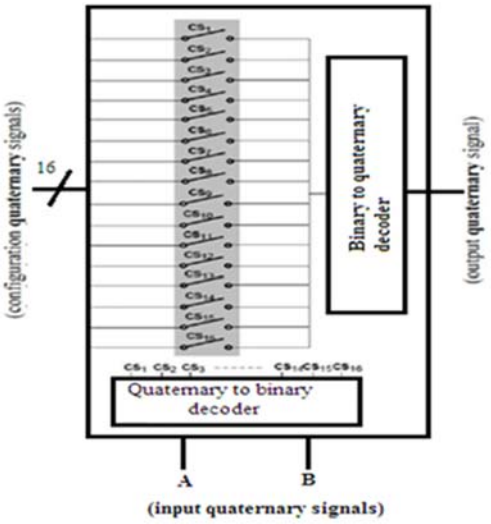


Fig 4: Proposed quaternary LUT Expanded

Table 2: combinatorial block logic functions

A	B	X_A	Y_A	X_B	Y_B	C_s (activebit)
0_4	0_4	0_2	0_2	0_2	0_2	Cs(1)
1_4	0_4	0_2	0_2	0_2	1_2	Cs(2)
2_4	0_4	0_2	0_2	1_2	0_2	Cs(3)
3_4	0_4	0_2	0_2	1_2	1_2	Cs(4)
0_4	1_4	0_2	1_2	0_2	0_2	Cs(5)
1_4	1_4	0_2	1_2	0_2	1_2	Cs(6)
2_4	1_4	0_2	1_2	1_2	0_2	Cs(7)
3_4	1_4	0_2	1_2	1_2	1_2	Cs(8)
0_4	2_4	1_2	0_2	0_2	0_2	Cs(9)
1_4	2_4	1_2	0_2	0_2	1_2	Cs(10)
2_4	2_4	1_2	0_2	1_2	0_2	Cs(11)
3_4	2_4	1_2	0_2	1_2	1_2	Cs(12)
0_4	3_4	1_2	1_2	0_2	0_2	Cs(13)
1_4	3_4	1_2	1_2	0_2	1_2	Cs(14)
2_4	3_4	1_2	1_2	1_2	0_2	Cs(15)
3_4	3_4	1_2	1_2	1_2	1_2	Cs(16)

A) 16x1 multiplexer

The mux used here's a 16x1, so it has 16 input pins and 1 output pin. This mux has usually four control lines in binary logic. Through the usage of the quaternary logic, the control lines are reduced to two. This is done with the brand new quaternary to binary decoder part introduced in the lookup table. The decoder reduces the fetch line of the mux from four to two. Why do we choose mux for this purpose is we can design a mux into an adder, subtracted, and divider or can use it for any other arithmetic or logical functions? The look up table will guide the mux to do various operations. The mux used has 16 pins, so we have switches that need to be operated very speedy to get the correct output. In the proposed method, here we use the direct technique to do the work in a single clock cycle. In the present technique four clock cycles are used.

B) Quaternary-To-Binary Decoder

The 2-bit quaternary-to-binary decoder allows the use of a single row of switches to drive the input configuration signals to the output of the QLUT. To do so, it is necessary to generate 16 control signals, to be applied in the clk inputs of each switch. These switches are employed to connect one quaternary configuration input to the output. To generate the required control signals, the quaternary variables are decoded into binary, allowing the use of binary logic gates. Thus, an ADC frontend is necessary, considering the analog nature of the quaternary signals. The decoder is the essential part to reduce the control pins from four to two. It will fetch the quaternary inputs and decode it into binary so as to perform the regular operation of the system designed. The designed system can be an adder or subtracted or divider etc. Here in the proposed design, an adder circuit is used to understand the quaternary logic and its other related benefits in the system.

C) Binary-To-Quaternary Decoder

The binary-to-quaternary decoder is used at the output side. The decoder is used because the internal operations are done in binary and the actual input/output are quaternary. So we need a convertor, that is, a decoder to convert the output generated from binary to quaternary. The internal functionality are similar and just the reverse of quaternary to binary decoder.

5. Simulation Results and Discussion

It is needed to test whether the design works to meet the given specification to ensure that designed entity is correct. This is verified by the process of simulation. The process of simulation uses a test bench to test the design whether it behaves correctly by stimulating it with artificial input and monitoring the output. The simulation is carried out by using the ModelSim 6.3a tool and having the test bench and the behavioral design code for 32-bit ALU in the same project folder. We observed from the simulation results that the 32-bit ALU implemented by the above described method and code, worked successfully for all the input combinations and the select codes according to the given specification in Table III.

The ModelSim simulation results are shown below

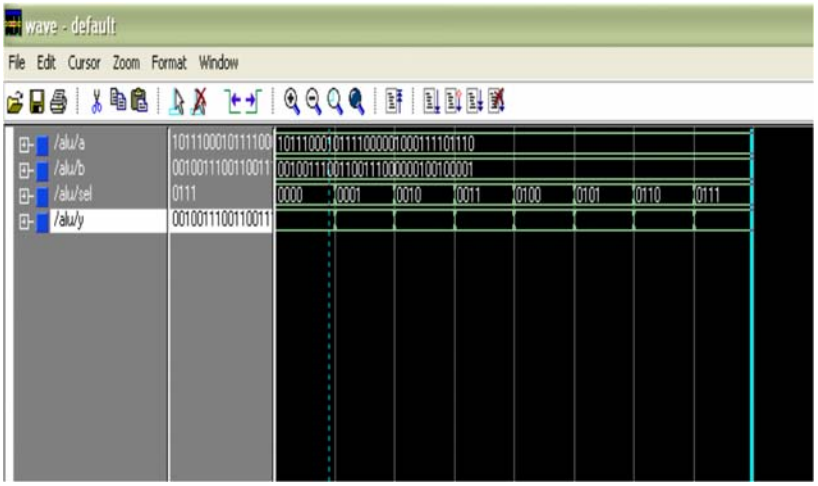


Fig 5.1: Simulation Result for Arithmetic Unit

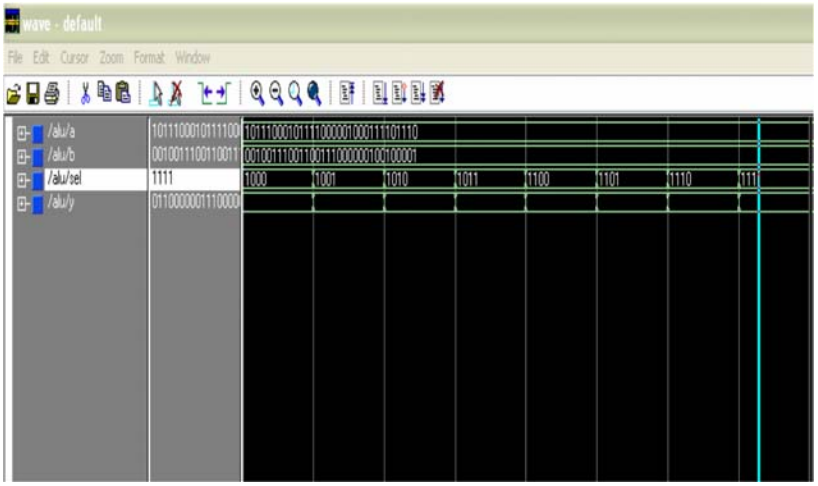


Fig 5.2: Simulation Result for Logic Unit

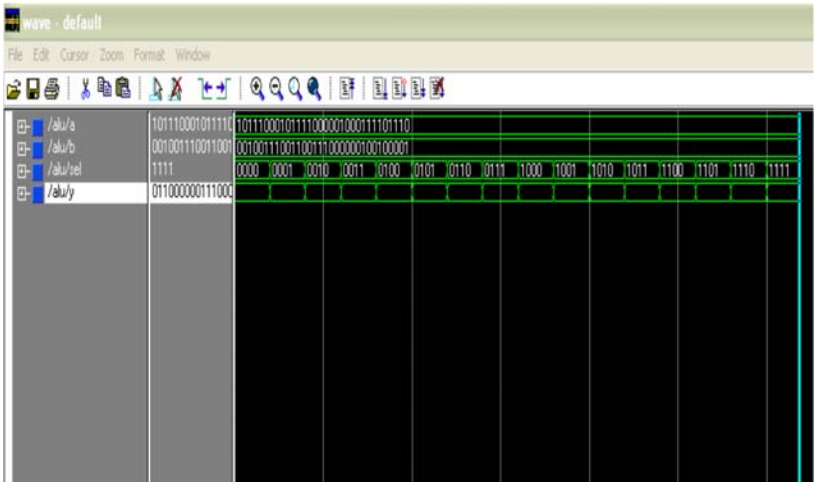


Fig 5.3: Simulation Result for the Whole Implemented 32- bit ALU

6. Conclusion

On this paper, we have reported an progressive QLUT layout that may be used for multiple valued logic. The proposed design is illustrated with the half and full adder design. The proposed design can be prolonged to many arithmetic and logical circuits. From the simulation results obtained, the presented design is a valid solution to reduce the interconnections impact, without increasing power consumption or dropping performance. In future this idea, which is beneficial to done in large scale, depending upon its effective implementation as it shows in the seeds to develop various real projects

7. References

1. Hasan Krad, AwsYousif Al-Taie. Performance Analysis of a 32-Bit Multiplier with a Carry-Look-Ahead Adder and a 32- bit Multiplier with a Ripple Adder using VHDL, Journal of Computer Science, Asadi, P. and K. Navi "A novel high-speed 54-54-bit multiplier. 2008; 4(4):305-308
2. Landauer R. Irreversibility and heat generation in the computing process, *IBM J. Research and Development.* 1961; 5(3):183-191.
3. Bennett CH. Logical reversibility of computation, *IBM J. Research and Development*, 1973; 17:525-532,
4. Diogo Brito, Jorge Fernandes, Paulo Flores, Jose Monteiro, Taimur Rabuske. Senior member IEEE Quaternary Logic Look up Table in standard CMOS IEEE transaction on very large scale integration system, 2015.
5. Nagamani AN, Nischai S. PES institute of Technology, Karnataka, Quaternary High Performance Arithmetic Logic Unit Design Euromica Conference, 2011.
6. Marcus Ritt, Carlos Arthur Lang Lisboa, Luigi Carro, Cristiano Lizzari. A cost effective Technique for mapping BLUTs to QLUTs in FPGAs IEEE conference, 2010.
7. Prashant Y, Shende Dr Kshirsagar RV. Quaternary Multiplier using VHDL international journal paper, 2013.
8. JeongBeom Kim, Dept. of Elec. Engg. Kangwon Natioinal university, Chunchon, south korea, Area Efficient multiplier using current mode Quaternary logic Technique, 10th IEEE conference, 2010.
9. Agarwal S, Pavankumar VK, Yokesh R. Energy-efficient high performance circuits for arithmetic units IEEE International Conference.
10. Tang AJJ, Reyes JA. Comparative analysis of low power multiplier architectures IEEE Symposium 2011.
11. Mr J, lanthendral, sathyavathi NS. A Novel Voltage-Mode LUT Using Clock Boosting Technique in Standard Cmos. 2014; 2(4):12-20.