**DESIGNING OF A LESS DELAY AND AREA EFFICIENT 64 BIT BINARY ADDER**

*A Project Report*

*Submitted in partial fulfillment of the requirement for the award of the degree of*

**Bachelor of Technology**

*in*

**Electronics and Communication Engineering**

*by*

**Shashivardhan Reddy Malagari**

**14BEC0126**

*Under the guidance of*

**Prof. RAGUNATH G**

**School of Electronics Engineering**

**Vellore Institute of Technology, Vellore-632014**



April 2018

**DECLARATION**

I hereby declare that the project work entitled “**DESIGNING OF A LESS DELAY AND AREA EFFICIENT 64 BIT BINARY ADDER**” submitted by **Shashivardhan Reddy Malagari**, for the award of the degree of *Bachelor of Technology in Electronics and Communication Engineering* to Vellore Institute of Technology is a record of bonafide work carried out by me under the supervision of **Prof RAGUNATH G**.

I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place :Vellore **Signature of the Candidate**

Date : April, 2018

**CERTIFICATE**

This is to certify that the project work entitled **“DESIGNING OF A LESS DELAY AND AREA EFFICIENT 64 BIT BINARY ADDER”** submitted by **Shashivardhan Reddy Malagari**, School of Electronics Engineering, Vellore Institute of Technology, for the award of the degree of *Bachelor of Technology in Electronics and Communication Engineering*, is a record of bonafide work carried out by him/her under my supervision, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The report fulfills the requirements and regulations of the institute and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : **Signature of the Guide**

**The project work is satisfactory / unsatisfactory**

**Internal Examiner External Examiner**

Approved by

**Head of the Department**

Department of Communication Engineering

School of Electronics Engineering

# ACKNOWLEDGEMENTS

The completion of the project is a result of comprehensive efforts from many people. First of all, I would want to thank my guide, Prof. **RAGUNATH G**, for the constant support he has been providing me with throughout the project. The research team of the Department of Micro and Nano-electronics has constantly guided me and helped me in tackling various problems I faced during the course of theproject.

My sincere thanks to SENSE school for giving me an opportunity to work towards my interest and providing me enough resources to complete the project. I would also want to thank the HOD of Department of Communication for guiding me on the various aspects throughout the project.

My friends and family have always been supportive and without their constant support and motivation I could not have pulled through and completed the project.

At last, I would want to thank VIT UNIVERSITY for providing the students such an ambient atmosphere and advanced infrastructure which helps us and constantly motivates us to come up with innovativeideas.

14BEC0126 Shashivardhan Reddy Malagari

**Executive summary**

There is a significant demand for Binary adders, especially in commercial and financial applications. With the advancement in the technology there is tremendous increase in demand for electronic devices with low power consumption and area efficient. So I have come up with new concept and proposed a new circuit for the binary adder.The existing circuit introduces more delay because of large number of gates required at each stage. However in our proposed circuit we have eliminated few gates which in turn reduces delay of the adder, power required and area of the circuit, hence reduces the complexity. The results are synthesized in Cadence Verilog design environment which has shown comparison between Area, Power and timing of two circuits. Carry dependent and independent circuit can be one of the fastest adders to perform arithmetic operations. From the structure of Carry dependent and independent circuitit is clear that there is scope for reducing the area and power consumption.This work uses a simple and efficient gate level modification to significantly reduce the area and power of Carry dependent and independent circuit. Based on this modification 64 -bit adder with Carry dependent and independent circuit is compared with the existing ripple carry adderand CSLA. The proposed design has reduced area and power as compared with CSLA.This work evaluates the performance of the proposed designs in terms of delay area power .

|  |  |  |
| --- | --- | --- |
| **CONTENTS** | | **Page No.** |
|  | **Acknowledgement** | 4 |
|  | **Executive Summary** | 5 |
|  | **Table of Contents** | 6i |
|  | **List of Figures** | 8 |
|  | **List of Tables** | 10 |
|  | **List of Terms and Abbreviations** | 11 |
|  | **List of Symbols and Notations** | 12 |
| 1 | **INTRODUCTION** | 13 |
|  | 1.1 Motivation | 17 |
| 1.2 Background | 18 |
| 1.3 Objective  1.4 Literature review | 21  21 |
| 1.5 Organization of the report | 25 |
| 2 | **PROJECT DESCRIPTION AND GOALS** | 26 |
| 3 | **TECHNICAL SPECIFICATION** | 27 |
| 4 | **DESIGN APPROACH AND DETAILS** | 33 |
|  | 4.1 Design Approach | 33 |
| 4.2 Codes and Standards | 34 |
|  |  |
| 5 | **SCHEDULE, TASKS AND MILESTONES** | 35 |
| 6 | **PROJECT DEMONSTRATION** | 37 |
| 7 | **CONCLUSION** | 67 |
| 8 | **REFERENCES** | 68 |
|  |  |  |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Table no** | **Title** | **Page no** |
| 1 | 64 bit ripple carry adder | 42 |
| 2 | 2 bit carry circuit | 43 |
| 3 | 2 bit adder circuit | 44 |
| 4 | 3 bit carry circuit | 46 |
| 5 | 3 bit adder circuit | 47 |
| 6 | 4 bit carry circuit | 50 |
| 7 | 4 bit adder circuit | 51 |
| 8 | 5 bit carry circuit | 56 |
| 9 | 5 bit adder circuit | 57 |
| 10 | Simulation parameters | 65 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| Figure  no | Title | Page no |
| 1 | Diagram of adder circuit | 19 |
| 2 | Two bit adder circuit | 20 |
| 3 | Basic simulation flow of model sim | 30 |
| 4 | Basic project flow of model sim | 30 |
| 5 | Multiple Library flow of Model sim | 31 |
| 6 | Carry Independent and Dependent circuit | 33 |
| 7 | Basic building block of CSLA | 37 |
| 8 | Uniform sized CSLA | 39 |
| 9 | Varied Sized CSLA | 40 |
| 10 | 64 Bit Ripple carry adder circuit | 42 |
| 11 | 64 Bit Ripple Carry adder Diagram | 42 |
| 12 | 2 Bit carry diagram | 43 |
| 13 | 2 Bit Carry output diagram | 44 |
| 14 | 2 Bit adder circuit | 44 |
| 15 | Output of 2 bit adder circuit | 45 |
| 16 | Three bit carry circuit | 46 |
| 17 | Output of three bit carry circuit | 47 |
| 18 | Three bit adder circuit | 47 |
| 19 | Output of three bit adder circuit | 48 |
| 20 | Four bit carry circuit | 50 |
| 21 | Output of four bit carry circuit | 50 |
| 22 | Four bit adder circuit | 51 |
| 23 | Output of four bit adder circuit | 51 |
| 24 | Five bit carry circuit | 56 |
| 25 | Output of five bit carry circuit | 56 |
| 26 | Five bit adder circuit | 57 |
| 27 | Output of five bit adder circuit | 57 |
| 28 | 64 bit adder ckt with(4\*16) | 61 |
| 29 | 64 bit adder ckt with (2\*1+3\*2+4\*1+5\*10) | 64 |
| 30 | 64 bit output diagram | 64 |
| 31 | Graphs(Delay) | 65 |
| 32 Graph(Area,Power) 66 | | |

**List of Terms and Abbreviation**

in Carry in

C out Carry out

S Sum

C Carry

CLA Carry Look Ahead Adder

Pi Carry Propagate

Gi Carry Generate

RLA Ripple Carry Adder

VLSI Very Large Scale integrated

CSLA Carry Select Adder

DPD Densely Packed Decimal

BCD Binary Coded Decimal

BID Binary Integer Decimal

SQRT CSLA Square Root Carry Select Adder

ASIC Application-specific integrated circuit

BEC Binary To Excess Convertor

FA Full Adder

CS Carry Select

CPD Carry Propagation Delay

MUX Multiplexer

OVI Open Verilog International

CAE Computer-aided engineering

PLIProgramming Language Interface

I/O Input Output Interface

**LIST OF SYMBOLS AND NOTATIONS**

**SYMBOL NOTATION**

**+** OR gate

**.** AND gate

**⊕** XOR gate

**1.INTRODUCTION**

In arithmetic circuits, adder is an inevitable circuit that performs addition of numbers. Besides Adders play major role in Multiplication. Several types of adders (i.e Ripple carry adder, Carry select adder, Carry skip adder, Carry look ahead adder and ssoon) are available. Among them Carry select adder and Carry look ahead adders are fastest adders but they take more power and area. In most of the applications fast adders / low power adders are required. So in this project I have designed a new adder which is faster than carry select adder with lesser area and low power.

**Half adder**

The half adder adds two one-bit binary numbers *A* and *B*. It has two outputs, *S* and *C* (the value theoretically carried on to the next addition); the final sum is 2C + S. The simplest half-adder design, pictured on the right, incorporates an XOR gate for *S* and an AND gate for *C*. With the addition of an OR gate to combine their carry outputs, two half adders can be combined to make a full adder

**Full addder**

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as *A*, *B*, and *C*in; *A* and*B* are the operands, and *C*in is a bit carried in from the next less significant stage. The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers.

A full adder can be implemented in many different ways such as with a custom transistor-level circuit or composed of other gates. One example implementation is with and C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)).

In this implementation, the final OR gatebefore the carry-out output may be replaced by an XOR gate without altering the resulting logic. Using only two types of gates is convenient if the circuit is being implemented using simple IC chips which contain only one gate type per chip. In this light, Cout can be implemented as C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)).

A full adder can be constructed from two half adders by connecting *A* and *B* to the input of one half adder, connecting the sum from that to an input to the second adder, connecting *Ci* to the other input and OR the two carry outputs. Equivalently, *S* could be made the three-bit XOR of *A*, *B*, and *Ci*, and *Cout* could be made the three-bit majority function of *A*, *B*, and *Ci*.

### RIPPLE CARRY ADDER

It is possible to create a logical circuit using multiple full adders to add *N*-bit numbers. Each full adder inputs a *Cin*, which is the *Cout* of the previous adder. This kind of adder is a ripple carry adder,since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder.

The layout of a ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The gate delay can easily be calculated by inspection of the full adder circuit. Each full adder requires three levels of logic. In a 64-bit [ripple carry] adder, there are 32 full adders, so the critical path (worst case) delay is 3 (from input to carry in first adder) + 63\* 2 (for carry propagation in later adders) = 129 gate delays. A design with alternating carry polarities and optimized AND-OR-Invert gates can be about twice as fast.

**CARRY LOOKAHEAD ADDER**

To reduce the computation time, engineers devised faster ways to add two binary numbers by using carry-lookahead adders. They work by creating two signals (*Pi* and *Gi*) for each bit position, based on if a carry is propagated through from a less significant bit position (at least one input is a '1'), a carry is generated in that bit position (both inputs are '1'), or if a carry is killed in that bit position (both inputs are '0'). In most cases, *Pi* is simply the sum output of a half-adder and *Gi* is the carry output of the same adder. After *Pi* and *Gi* are generated the carries for every bit position are created. Some advanced carry-lookahead architectures are the Manchester carry chain, Brent–Kung adder, and the Kogge–Stone adder.

Some other multi-bit adder architectures break the adder into blocks. It is possible to vary the length of these blocks based on the propagation delay of the circuits to optimize computation time. These block based adders include the carry bypass adder which will determine *P* and *G* values for each block rather than each bit, and the carry select adder which pre-generates sum and carry values for either possible carry input to the block.

A carry-lookahead adder (CLA) is a type of adder used in digital logic. A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits. It can be contrasted with the simpler, but usually slower, ripple carry adder for which the carry bit is calculated alongside the sum bit, and each bit must wait until the previous carry has been calculated to begin calculating its own result and carry bits (see adder for detail on ripple carry adders). The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits. The Kogge-Stone adder and Brent-Kung adder are examples of this type of adder.

Charles Babbage recognized the performance penalty imposed by ripple carry and developed mechanisms foranticipating carriage in his computing engines.[[1]](http://en.wikipedia.org/wiki/Carry-lookahead_adder#cite_note-0) Gerald Rosenberger of IBM filed for a patent on a modern binary carry-lookahead adder in 1957.[[2]](http://en.wikipedia.org/wiki/Carry-lookahead_adder#cite_note-1)

A ripple-carry adder works in the same way as pencil-and-paper methods of addition. Starting at the rightmost (least significant) digit position, the two corresponding digits are added and a result obtained. It is also possible that there may be a carry out of this digit position (for example, in pencil-and-paper methods, "9+5=4, carry 1"). Accordingly all digit positions other than the rightmost need to take into account the possibility of having to add an extra 1, from a carry that has come in from the next position to the right.

This means that no digit position can have an absolutely final value until it has been established whether or not a carry is coming in from the right. Moreover, if the sum without a carry is 9 (in pencil-and-paper methods) or 1 (in binary arithmetic), it is not even possible to tell whether or not a given digit position is going to pass on a carry to the position on its left. At worst, when a whole sequence of sums comes to ...99999999... (in decimal) or ...11111111... (in binary), nothing can be deduced at all until the value of the carry coming in from the right is known, and that carry is then propagated to the left, one step at a time, as each digit position evaluated "9+1=0, carry 1" or "1+1=0, carry 1". It is the "rippling" of the carry from right to left that gives a ripple-carry adder its name, and its slowness. When adding 32-bit integers, for instance, allowance has to be made for the possibility that a carry could have to ripple through every one of the 32 one-bit adders.

Carry lookahead depends on two things:

1. Calculating, for each digit position, whether that position is going to propagate a carry if one comes in from the right.
2. Combining these calculated values to be able to deduce quickly whether, for each group of digits, that group is going to propagate a carry that comes in from the right.

Supposing that groups of 4 digits are chosen. Then the sequence of events goes something like this:

1. All 1-bit adders calculate their results. Simultaneously, the lookahead units perform their calculations.
2. Suppose that a carry arises in a particular group. Within at most 3 gate delays, that carry will emerge at the left-hand end of the group and start propagating through the group to its left.
3. If that carry is going to propagate all the way through the next group, the lookahead unit will already have deduced this. Accordingly,before the carry emerges from the next group the lookahead unit is immediately (within 1 gate delay) able to tell the next group to the left that it is going to receive a carry - and, at the same time, to tell the next lookahead unit to the left that a carry is on its way.

The net effect is that the carries start by propagating slowly through each 4-bit group, just as in a ripple-carry system, but then move 4 times as fast, leaping from one lookahead carry unit to the next. Finally, within each group that receives a carry, the carry propagates slowly within the digits in that group.

The more bits in a group, the more complex the lookahead carry logic becomes, and the more time is spent on the "slow roads" in each group rather than on the "fast road" between the groups (provided by the lookahead carry logic). On the other hand, the fewer bits there are in a group, the more groups have to be traversed to get from one end of a number to the other, and the less acceleration is obtained as a result.

Deciding the group size to be governed by lookahead carry logic requires a detailed analysis of gate and propagation delays for the particular technology being used.

It is possible to have more than one level of lookahead carry logic, and this is in fact usually done. Each lookahead carry unit already produces a signal saying "if a carry comes in from the right, I will propagate it to the left", and those signals can be combined so that each group of (let us say) four lookahead carry units becomes part of a "supergroup" governing a total of 16 bits of the numbers being added. The "supergroup" lookahead carry logic will be able to say whether a carry entering the supergroup will be propagated all the way through it, and using this information, it is able to propagate carries from right to left 16 times as fast as a naive ripple carry. With this kind of two-level implementation, a carry may first propagate through the "slow road" of individual adders, then, on reaching the left-hand end of its group, propagate through the "fast road" of 4-bit lookahead carry logic, then, on reaching the left-hand end of its supergroup, propagate through the "superfast road" of 16-bit lookahead carry logic.

Again, the group sizes to be chosen depend on the exact details of how fast signals propagate within logic gates and from one logic gate to another.

For very large numbers (hundreds or even thousands of bits) look ahead carry logic does not become any more complex, because more layers of super groups and super super groups can be added as necessary. The increase in the number of gates is also moderate: if all the group sizes are 4, one would end up with one third as many lookahead carry units as there are adders. However, the "slow roads" on the way to the faster levels begin to impose a drag on the whole system (for instance, a 256-bit adder could have up to 24 gate delays in its carry processing), and the mere physical transmission of signals from one end of a long number to the other begins to be a problem. At these sizes carry-save adders are preferable, since they spend no time on carry propagation at all.

**1.1 MOTIVATION**

The challenge of the verifying a large design is growing exponentially. There is a need to define new methods that makes functional verification easy. Several strategies in the recent years have been proposed to achieve good functional verification with less effort. Recent advancement towards this goal is methodologies. The methodology defines a skeleton over which one can add flesh and skin to their requirements to achieve functional verification.

As the scale of integration keeps growing, more and more sophisticated signal processing systems are being implemented on a VLSI chip. These signal processing applications not only demand great computation capacity but also consume considerable amount of energy. While performance and Area remain to be the two major design tools, power consumption has become a critical concern in today’s VLSI system design. The need for low-power VLSI system arises from two main forces. First, with the steady growth of operating frequency and processing capacity per chip, large currents have to be delivered and the heat due to large power consumption must be removed by proper cooling techniques. Second, battery life in portable electronic devices is limited. Low power design directly leads to prolonged operation time in these portable devices.

Addition is the most common and often used arithmetic operation on micro processor ,digital signal processor, especially digital commuters also it serves as a building block for synthesis for all other arithmetic operations therefore regarding the efficient implementation of an arithmetic unit, the binary adder structure become a very critical hardware unique

In Digital adders, the speed of addition is limited by the time required to propagate a carry through the adder. The sum of each bit position is generated sequentially only after the previous bit position has been sum and a carry propagated into the next position.

The major speed limitation in any adder is in the production of carries and many authors have considered the additional problems.The carry dependent and independent circuit is used to moderate the problem of carry propogation delay by generating the sum through the carry generated circuit.

Carry Select Adder (CSLA) is one of the fastest adders used in many data-processing processors to perform fast arithmetic functions. From the structure of the CSLA, it is clear that there is scope for reducing the area and power consumption in the CSLA. This work uses a simple and efficient gate-level modification to significantly reduce the area and power of the CSLA. Based on this modification 64-Bit ADDER architecture have been developed and compared with the regular CSLA architecture. The proposed design has reduced area and power as compared with the regular CSLA

**1.2 BACKGROUND**

|  |  |  |  |
| --- | --- | --- | --- |
| BINARY ADDITION   |  |  |  | | --- | --- | --- | |  |  |  |   Now that we know binary numbers, we will learn how to add them. Binary addition is much like your normal everyday addition (decimal addition), except that it carries on a value of 2 instead of a value of 10.  For example: in decimal addition, if you add 8 + 2 you get ten, which you write as 10; in the sum this gives a digit 0 and a carry of 1. Something similar happens in binary addition when you add 1 and 1; the result is two (as always), but since two is written as 10 in binary, we get, after summing 1 + 1 in binary, a digit 0 and a carry of 1.  Therefore in binary: 0 + 0 = 0 0 + 1 = 1 1 + 0 = 1 1 + 1 = 10 (which is 0 carry 1)  **Example.** Suppose we would like to add two binary numbers 10 and 11. We start from the last digit. Adding 0 and 1, we get 1 (no carry). That means the last digit of the answer will be one. Then we move one digit to the left: adding 1 and 1 we get 10. Hence, the answer is 101. Note that binary 10 and 11 correspond to 2 and 3 respectively. And the binary sum 101 corresponds to decimal 5: is the binary addition corresponds to our regular addition.  **ADDERS**  An adder, not surprisingly, is a circuit whose output is the binary sum of its inputs. Since adders are needed to perform arithmetic, they are an essential part of any computer. In future labs, you will be using the adder that you design in this lab to perform arithmetic in a working microprocessor. A full adder has three inputs (A, B, Carry in) and two outputs (SUM, Carry out), as shown in Figure 1. Inputs A and B represent bits of the two binary numbers that are being added, and SUM represents a bit of the resulting sum.  A B  Adder Circuit  Cout Cin    Sum  F**ig .1** Diagram of adder circuit  The Carry in and Carry out signals are used when adding numbers that are more than one bit long. To understand how these signals are used, consider how you would add the binary numbers 01 and 01 by hand:  1  01  + 01  10  You first add the two least significant bits. Since 1+1=10 (in binary), you place a zero in the least significant bit of the sum and carry the 1. Then you add the next two bits with the carry, and place a 1 in the most significant bit of the sum.  When a sum is performed using full adders, each adder handles a single column of the sum. Figure 2 shows how to build a circuit that adds two two-digit binary numbers using two full adders. The Carry out for each bit is connected to the carry in of the next most significant bit. Each bit of the three-bit numbers being added is connected to the appropriate adder’s inputs and the three SUM outputs make up to full 3-bit sum result.  Note that the leftmost Carry in input is unnecessary, since there can never be a carry into the first column of the sum. This allows us to use a half adder for the first bit of the sum. A half adder is similar to a full adder, except that it lacks a Carry in and is thus simpler to implement. Two bit adder is shown in Figure 2.  A[1] B[1] A[0] B[0]    Adder circuit  Adder circuit  Cout C 1Cin  S[1] S[2]  **Fig .2**Two bit adder circuit  **1.3 OBJECTIVE** |

A adder has to be designed with high speed, low power and lesser area. Because most of the fast adders require more area. One of the way to achieve the fast adder is by reducing the delay of the carry chain. In this project, delay of the carry chain is reduced without increasing area much. Further the carry is determined by carry dependent circuit and carry independent circuit. The delay of the particular block is 2 gates delay after carry arrives. So it is faster than carry select adder. More over area is also less compare to carry select adder because only one adder is used in each sub-block/adder.

**1.4 LITERATURE REVIEW**

Low area/power decimal addition with carry-select correction and carry-select sum-digits MortezaDorrigiv a, GhassemJaberipura,b,n Realization of decimal arithmetic operations, via hardware circuits and/or software packages that operate on binary coded decimal (BCD) data, has been an attractive research subject for decades [1–10]. This is in contrast to converting decimal numbers to their binary equivalent (not always accurate in case of decimal fractions [11]), performing binary arithmetic, and converting the results back to decimal. However, several application domains such as most monetary and business computations (e.g., banking and electronic commerce) require decimal arithmetic operators to directly process decimal data for accurate computations. There-fore, to respond to these consumer needs, the computer industry has reacted in two ways. Some manufacturers have preferred to support their binary processors with libraries of optimized soft-ware programs that simulate decimal arithmetic operations with no conversion of decimal fractions to binary. For example, this trend has influenced the IEEE 754–2008 standard for decimal floating-point arithmetic [12] to include Binary Integer Decimal (BID) encoding for significands of decimal floating-point data. On the other hand, there are commercialized binary digital processors that host radix-10 arithmetic via hardware units that also directly operate on binary coded decimal data [13–15]. This is to support the requirements of high performance and low power decimal applications [16]. To reduce storage cost of decimal data, IEEE 754–2008 Densely Packed Decimal (DPD) encoding is used at the cost of conversion to binary coded decimal (BCD) encoding, and the reverse, before and after each arithmetic operation, respectively [12]. Such decimal arithmetic hardware realizations often share circuits with binary arithmetic units for cost reduction [15]. For example, unified add/subtract units and fused binary/decimal architectures, that save some silicon area, have been presented in several research articles (e.g., [17–22]) and patents (e.g., [23–30]). In speculative radix-10 addition, sum of equally weighted oper-and's digits is added by 6 [31]. However, some recent works (e.g., [3]) add 6 to all digits of one operand, which allows for further binary addition of the augmented operand and the intact one. So, faster decimal addition can be realized with an embedded word-wide binary adder, leading to considerable hardware sharing with binary addition unit [30]

Low-Power and Area-Efficient Carry Select Adder B. Ram kumar and Harish M KitturAbstract:Carry Select Adder (CSLA) is one of the fastest adders used in many data- processing processors to perform fast arithmetic functions. From the structure of the CSLA, it is clear that there is scope for reducing the area and power consumption in the CSLA. This work uses a simple and efficient gate-level modification to significantly reduce the area and power of the CSLA. Based on this modification 8-, 16-, 32-, and 64-b square-root CSLA (SQRT CSLA) architecture have been developed and compared with the regular SQRT CSLA architecture. The proposed design has reduced area and power as compared with the regular SQRT CSLA with only a slight increase in the delay. This work evaluates the performance of the proposed designs in terms of delay, area, power, and their products by

hand with logical effort and through custom design and layout in 0.18- m CMOS process technology. The results analysis shows that the proposed CSLA structure is better than the regular SQRT CSLA.

*Index Terms—*Application-specific integrated circuit (ASIC), area-efficient, CSLA, low power.

Design of area- and power-efficient high-speed data path logic systems are one of the most substantial areas of research in VLSI system design. In digital adders, the speed of addition is limited by the time required to propagate a carry through the adder. The sum for each bit

position in an elementary adder is generated sequentially only after the previous bit position has been summed and a carry propagated into the next position. The CSLA is used in many computational systems to alleviate the problem of carry propagation delay by independently generating multiple carries and then select a carry to generate the sum [1]. However, the CSLA is not area efficient because it uses multiple pairs of Ripple Carry Adders (RCA) to generate partial sum and carry by considering carry input 􀀀\_\_ \_ \_ and 􀀀\_\_ \_ \_, then the final sum and carry are selected by the multiplexers (mux). The basic idea of this work is to use Binary to Excess-1 Converter (BEC) instead of RCA with 􀀀\_\_ \_ \_ in the regular CSLA to achieve lower area and power consumption [2]–[4]. The main advantage of this BEC logic comes from the lesser number of logic gates than the 􀀀-bit Full Adder (FA) structure. The details of the BEC logic are discussed in Section III.

This brief is structured as follows. Section II deals with the delay and area evaluation methodology of the basic adder blocks. Section III presents the detailed structure and the function of the BEC logic. The SQRT CSLA has been chosen for comparison with the proposed design as it has a more balanced delay, and requires lower power and area [5], [6]. The delay and area evaluation methodology of the regular and modified SQRT CSLA are presented in Sections IV and V, respectively. The ASIC implementation details and results are analyzed in Section VI. Finally, the work is concluded in Section VII.

* Area–Delay–Power Efficient Carry-Select Adder Basant Kumar Mohanty, *Senior Member, IEEE*, and Sujit Kumar Patel

*Abstract*—In this brief, the logic operations involved in conventional carry select adder (CSLA) and binary to excess-1 converter (BEC)-based CSLA are analyzed to study the data dependence and to identify redundant logic operations. We have eliminated all the redundant logic operations present in the conventional CSLA and proposed a new logic formulation for CSLA. In the proposed scheme, the carry select (CS) operation is scheduled before the calculation of *final-sum*, which is different from the conventional approach. Bit patterns of two anticipating carry words (corresponding to *c*in = 0 and 1) and fixed *c*in bits are used for

logic optimization of CS and generation units. An efficient CSLA design is obtained using optimized logic units. The proposed CSLA design involves significantly less area and delay than the recently proposed BEC-based CSLA. Due to the small carry-output delay, the proposed CSLA design is a good candidate for square-root (SQRT) CSLA. A theoretical estimate shows that the proposed SQRT-CSLA involves nearly 35% less area–delay–product (ADP) than the BEC-based SQRT-CSLA, which is best among the existing SQRT-CSLA designs, on average, for different bit-widths. The application-specified integrated circuit (ASIC) synthesis result shows that the BEC-based SQRT-CSLA design involves 48%

more ADP and consumes 50% more energy than the proposed SQRT-CSLA, on average, for different bit-widths. *Index Terms*—Adder, arithmetic unit, low-power design.

**L**OW-POWER, area-efficient, and high-performance VLSI systems are increasingly used in portable and mobile devices, multistandard wireless receivers, and biomedical instrumentation [1], [2]. An adder is the main component of an arithmetic unit. A complex digital signal processing (DSP) system involves several adders. An efficient adder design essentially improves the performance of a complex DSP system. A ripple carry adder (RCA) uses a simple design, but carry propagation delay (CPD) is the main concern in this adder.

Carry look-ahead and carry select (CS) methods have been suggested to reduce the CPD of adders. A conventional carry select adder (CSLA) is an RCA–RCA configuration that generates a pair of *sum* words and *output carry* bits corresponding the anticipated input-carry (*c*in = 0 and 1) and selects one out of each pair for *final-sum* and *final-output-carry* [3]. A conventional CSLA has less CPD than an RCA, but the design is not attractive since it uses a

dual RCA. Few attempts have been made to avoid dual use of RCA in CSLA design. Kim and Kim [4] used one RCA and one add-one circuit instead of two RCAs, where the add-one

circuit is implemented using a multiplexer (MUX). He *et al.* [5] proposed a square-root (SQRT)-CSLA to implement largebit-width adders with less delay. In a SQRT CSLA, CSLAswith increasing size are connected in a cascading structure. Themain objective of SQRT-CSLA design is to provide a parallelpath for carry propagation that helps to reduce the overalladder delay. Ramkumar and Kittur [6] suggested a binary toBEC-based CSLA. The BEC-based CSLA involves less logicresources than the conventional CSLA, but it has marginallyhigher delay. A CSLA based on common Boolean logic (CBL)is also proposed in [7] and [8]. The CBL-based CSLA of [7]involves significantly less logic resource than the conventionalCSLA but it has longer CPD, which is almost equal to thatof the RCA. To overcome this problem, a SQRT-CSLA basedon CBL was proposed in [8]. However, the CBL-based SQRTCSLAdesign of [8] requires more logic resource and delaythan the BEC-based SQRT-CSLA of [6]. We observe that logicoptimization largely depends on availability of redundant operationsin the formulation, whereas adder delay mainly dependson data dependence. In the existing designs, logic is optimizedwithout giving any consideration to the data dependence. Inthis brief, we made an analysis on logic operations involved in

conventional and BEC-based CSLAs to study the data dependence and to identify redundant logic operations. Based on this analysis, we have proposed a logic formulation for the CSLA.

The main contribution in this brief are logic formulation based on data dependence and optimized carry generator (CG) and CS design.

Based on the proposed logic formulation, we have derived an efficient logic design for CSLA. Due to optimized logic units, the proposed CSLA involves significantly less ADP than the existing CSLAs. We have shown that the SQRT-CSLA using the proposed CSLA design involves nearly 32% less ADP and consumes 33% less energy than that of the corresponding SQRT-CSLA. The rest of this brief is organized as follows. Logic formulation of CSLA is presented in Section II. The proposed CSLA is presented in Section III and the performance comparison is presented in Section IV. The conclusion is given

in Section V.

**1.5 ORGANIZATION OF REPORT**

**Chapter 2**:This Chapter gives an idea about what are the combination we are going to take for the 64 bit binary adder

**Chapter 3**:his chapter give an idea about the use of the Model Sim and the procedure of its simulation

**Chapte 4**:This chapter gives an idea about how we are going to follow the carry independent and dependent circuit according to the requirement of the project and the IEEE codes and standards

**Chapter 5:**It gives the scheduling timeand tasks

**Chapter 6:**It gives the idea about the existing circuits of carry look ahead adder and Ripple Carry Adder

**Chapter 7**:It gives the idea of our designing of the project

**2. PROJECT DESCRIPTION AND GOALS**

The main goal of our project is to reduce the area and also the delay of the circuit. I developed a 64 bit adder with different combinations of carry dependent and in-dependent circuit.For 64 bit adder two different combinations were designed such as 4\*16 and 2\*1+3\*2+4\*1+5\*10. In every combination carry dependent and in-dependent cicuit is placed to produce the carry next to the different or same adder in the 64 bit adder itself.itself.In 2\*1+5\*6,after two bit addition,the carry from the carry dependentand in-dependent circuit is passed as Cin to the next 5 bit addition.In such a way the proposed 64 bit adder is designed. Therefore increasing its performance but also increasing the area.

We also focus on reducing the number of logic gates and hence reducing the power consumption. So, several combinations were designed to reduce the power,area and also delay.

So in real time if the number of logic gates are reduced so the cross sectional area of the chip decrease.

**3.Technical Specifications**

VERILOG HDL

Verilog-HDL was released in 1983 by Gateway Design System Corporation, together with a Verilog-HDL simulator. In 1985, the language and simulator were enhanced with the introduction of the Verilog-XL simulator. In 1989, Cadence Design Systems, Inc. bought the Gateway Design System Corporation, and in early 1990, Verilog- HDL and Verilog-XL were separated into two products. Verilog-HDL, until then a proprietary language, was released into the public domain to facilitate the dissemination of knowledge relating to Verilog-HDL and to allow Verilog-HDL to compete with VHDL, which already existed as a nonproprietary language. In 1990, Open Verilog International (OVI) was formed as an industry consortium consisting of computer-aided engineering (CAE) vendors and Verilog-HDL users to control the language specification. In 1995, Verilog-HDL was reviewed and adopted as IEEE Standard 1364 (becoming IEEE Std 1364-1995). In 2001, the standard was reviewed, the latest version of the standard now being IEEE Std 1364-2001.

* Verilog HDL has evolved as a standard hardware description language. Verilog HDL offers many useful features for hardware design.
* Verilog HDL is a general-purpose hardware description language that is easy to learn and easy to use. It is similar in syntax to the C programming language. Designers with C programming experience will find it easy to learn Verilog HDL.
* Verilog HDL allows different levels of abstraction to be mixed in the same model. Thus, a designer can define a hardware model in terms of switches, gates, RTL, or behavioural code. Also, a designer needs to learn only one language for stimulus and hierarchical design.
* Most popular logic synthesis tools support Verilog HDL. This makes it the language of choice for designers. All fabrication vendors provide Verilog HDL libraries for post logic synthesis simulation. Thus, designing a chip in Verilog HDL allows the widest choice of vendors.
* The Programming Language Interface (PLI) is a powerful feature that allows the user to write custom C code to interact with the internal data structures of Verilog. Designers can customize a Verilog HDL simulator to their needs with the PLI.

The speed and complexity of digital circuits has increased rapidly. Designers have responded by designing at higher levels of abstraction. Designers have to think only in terms of functionality. CAD tools take care of the implementation details. With designer assistance, CAD tools have become sophisticated enough to do a close-to-optimum implementation.

The most popular trend currently is to design in HDL at an RTL level, because logic synthesis tools can create gate-level net lists from RTL level design. Behavioral synthesis has recently emerged. As these tools improve, designers will be able to design directly in terms of algorithms and the behavior of the circuit, and then use CAD tools to do the translation and optimization in each phase of the design. Behavioralmodeling will be used more and more as behavioral synthesis matures. Until then, RTL design will remain very popular.

Formal verification techniques are also appearing on the horizon. Formal verification applies formal mathematical techniques to verify the correctness of Verilog HDL descriptions and to establish equivalency between RTL and gate level net lists. However, the need to describe a design in Verilog HDL will not go away.

For very high speed and timing-critical circuits like microprocessors, the gate level netlist provided by logic synthesis tools is not optimal. In such cases, designers often mix gate-level description directly into the RTL description to achieve optimum results. This practice is opposite to the high-level design paradigm, yet it is frequently used for high-speed designs because designers need to squeeze the last bit of timing out of circuits and CAD tools sometimes prove to be insufficient to achieve the desired results.

A trend that is emerging for system-level design is a mixed bottom-up methodology where the designers use existing Verilog HDL modules, basic building blocks, or vendor-supplied core blocks to quickly bring up their system simulation. This is done to reduce development costs and compress design schedules. For example, consider a system that has a CPU, graphics chip, I/O chip, and a system bus. The CPU designers would build the next-generation CPU themselves at an RTL level, but they would use behavioral models for the graphics chip and the 1/0 chip and would buy a vendor-supplied model for the system bus. Thus, the system-level simulation for the CPU could be up and running very quickly and long before the RTL descriptions for the graphics chip and the 1/0 chip are completed.

**MODELSIM – ALTRA**

**Assumptions**

I assume that you are familiar with the use of your operating system. You should also be familiar with the window management functions of your graphic interface: Open Windows, OSF/Motif, CDE, KDE, GNOME, or Microsoft Windows 2000/XP. We also assume that you have a working knowledgeof the language in which your design and/or test bench is written (i.e., VHDL, Verilog, etc.). Although ModelSim™ is an excellent tool to use while learning HDL concepts and practices, this document is not written to support that goal.

**Modelsim introduction**

ModelSim is a verification and simulation tool for VHDL, Verilog, System Verilog, and mixed language designs. This lesson provides a brief conceptual overview of the ModelSim simulation environment. It is divided into four topics, which you will learn more about in subsequent lessons

Basic Simulation Flow

The following diagram shows the basic steps for simulating a design in ModelSim.



Fig.3Basic Simulation Flow of Model sim

Basic Simulation Flow - Overview Lab [Creating the Working Library]:

In ModelSim, all designs are compiled into a library. You typically start a new simulation in ModelSim by creating a working library called "work," which is the default library name used by the compiler as the default destination for compiled design units.

* **Compiling Your Design**

After creating the working library, and compile your design units into it. The ModelSim library format is compatible across all supported platforms. Its can simulate your design on any platform without having to recompile your design. Loading the Simulator with Your Design and Running the Simulation with the design compiled, load the simulator with your design by invoking the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL).Assuming the design loads successfully, the simulation time is set to zero, and you enter a run command to begin simulation.

* **Debugging Your Results**

If you don’t get the results you expect, you can use ModelSim’s robust debugging environment to track down the cause of the problem.

Project Flow

A project is a collection mechanism for an HDL design under specification or test. Even though you don’t have to use projects in ModelSim, they may ease interaction with the tool and are useful for organizing files and specifying simulation settings. The following diagramshows the basic steps for simulating a design within a ModelSim project.

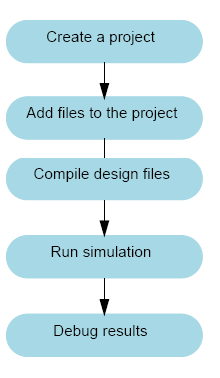


Fig. 4 Project flow of Model Sim

As you can see, the flow is similar to the basic simulation flow. However, there are two important differences:

* Do not have to create a working library in the project flow; it is done for you automatically.
* Projects are persistent. In other words, they will open every time you invoke ModelSim unless you specifically close them.

Multiple Library Flow

ModelSim uses libraries in two ways: 1) as a local working library that contains the compiled version of your design; 2) as a resource library. The contents of your working library will change as you update your design and recompile. A resource library is typically static and serves as a parts source for your design. It can create your own resource libraries, or they may be supplied by another design team or a third party (e.g., a silicon vendor). It specifies which resource libraries will be used when the design is compiled, and there are rules to specify in which order they are searched. A common example of using both a working library and a resource library is one where your gate-level design and test bench are compiled into the working library, and the design references gate-level models in a separate resource library. The diagram below shows the basic steps for simulating with multiple libraries.

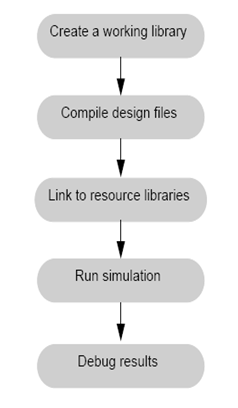


Fig.5 Multiple Library flow of Model sim

Debugging Tools

ModelSim offers numerous tools for debugging and analyzing your design. Several of these tools are covered in subsequent lessons, including:

* Using projects
* Working with multiple libraries
* Setting breakpoints and stepping through the source code
* Viewing waveforms and measuring time
* Viewing and initializing memories
* Creating stimulus with the Waveform Editor
* Automating simulation
  + 1. Basic Simulation

In this lesson you will go step-by-step through the basic simulation flow:

1. Create the Working Design Library

2. Compile the Design Units

3. Load the Design

4. Run the Simulation

**4.DESIGN AND APPROACH DETAILS**

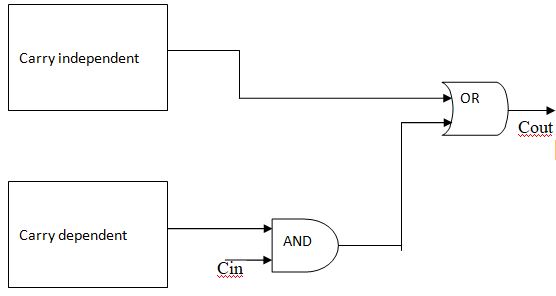
**4.1DESIGN APPROACH**

We developed a 64 bit adder with different combinations of carry dependent and in-dependent circuit.For 64 bit adder three different combinations were designed with 4\*16 and 2\*2+3\*2+4\*1+5\*10 adders. In every combination carry dependent and in-dependent cicuit of the particular bit is placed to produce the carrynext to the different or same bit adder in the 64 bit adder itself.In 2\*2+3\*2+4\*1+5\*10,after two bit addition,the carry from the carry dependent and in-dependent circuit is passed as Cin to the next two bit addition.In such a way the proposed 64 bit adder is designed. Therefore increasing its performance but also increasing the area.

Carry in-dependent means the Cout is independent on the previous carry. In this Cout should be in-dependent of Cin. So this makes the Cout do not wait for the value from AND gate of Cin and makes it Carry in-dependent.

Carry dependent means the Cout is dependent on previous carry. In this Cout waits for the value of Cin.

So the switching capacity of the circuit is very fast and hence the carry propagation delay decreases. The carry dependent and in-dependent circuit is shown in Figure 6.



**Fig.6** Carry Independent and Dependent circuit

In this we are building carry circuit in order to make the addition process fast. For example take two bit addition of 33,21. Addition of units digit 3,1 takes place in binary numbers 11,01.So after the addition the carry is produced. This carry is added to the next sum digits 3,2. The addition of 3,2 takes place only when carry is available from 3,1 addition, which consumes a lot of time. Therefore, a carry circuit is placed to reduce the delay in addition.

**4**.1 **Codes and Standards**

We have used Verilog to simulate and verify the working of various gates and the circuits proposed by us during the project.

Verilog has been standardized as IEEE 1364. It is a hardware description language. It is basically used to design and verify the result of circuits at register transfer level. The Verilog Hardware Description Language (Verilog HDL) became an IEEE standard in 1995 as IEEE Std 1364-1995. It was designed to be simple, intuitive, and effective at multiple levels of abstraction in a standard textual format for a variety of design tools, including verification simulation, timing analysis, test analysis, and synthesis. It is because of these rich features that Verilog has been accepted to be the language of choice by an overwhelming number of IC designers.

Verilog contains built-in primitives, logic gates, user-definable primitives, switches, and wired logic. It also has device pin-to-pin delays and timing checks. The mixing of abstract levels is provided by the data types: nets and variables. Continuous assignments, in which expressions of both variables and nets can continuously drive values onto nets, provide the basic structural construct. Procedural assignments, in which the results of calculations involving variable and net values can be stored into variables, provide the basic behaviouralconstruct.

A design consists of a set of modules, each of which has an I/O interface, and a description of its function, which can be structural, behavioural, or a mix. These modules are formed into a hierarchy and are interconnected with nets. The Verilog language is extensible via the Programming Language Interface (PLI) and the Verilog Procedural Interface (VPI) routines. The PLI/VPI is a collection of routines that allows foreign functions to access information contained in a Verilog HDL description of the design and facilitates dynamic interaction with simulation. Applications of PLI/VPI include connecting to a Verilog HDL simulator with other simulation and CAD systems, customized debugging tasks, delay calculators, andannotators.

**5. SCHEDULE,TASKS AND MILESTONES**

Table 5.1: Timeline

|  |  |  |
| --- | --- | --- |
| **Sl. #** | **Month** | **Work done** |
| 1 | December,2017 | Literature Review |
| 2 | January, 2018 | Focussed on carry circuit. |
| 3 | February, 2018 | Implementation of proposed circuit |
| 4 | March, 2018 | Preparing Poster and thesis |
| 5 | April, 2018 | Final Viva-Voce |

**DECEMBER:**

The most of the month went for literature review. We went through many papers and realised the design of all adders. We mostly studied different kinds of circuits and designed and applied different concepts. We understood the basics of our project and went through various previously published research papers to understand how to come up with the new efficient design that would reduce power consumption. This was necessary as we could find out the area, power and delay of the design. Also the designs of the previous works were implemented in Verilog and the results were verified.

**January:**

After understanding the various parameters of the circuit, we started coming up with ideas of reducing area, power keeping the delay as equal as possible. So inorder to reduce the delay we focused on carry circuit. We learned that faster the carry is generated , the faster is addition. We built the circuits for 4-bit,3-bit and 2-bit.

**February:**

So after building the carry circuits we learned that area and power consumption can be reduced by minimizing the logic gates. So this must not compromise our expected result. The result was that we were able to put forward designs which were much more efficient in terms of Area, Power Consumption, Delay and Total Logical Complexity as compared to the previous works.Verilogcodings were written for 4-bit,3-bit and 2-bit and verified.

**March:**

After verifying the results in verilog for both circuits,we extended our work into 16-bit,32- bit,64-bit.We moved on to synthesis part which was implemented in Cadence system design environment. We compared the efficiency with the existing adder circuits based on three parameters i.e, Area, Power consumption and Delay. As expected, we got the desired results. Preparation of the thesis and poster was done.

**APRIL:**

Poster presentation, thesis submission and final viva-voce.

6. PROJECTDEMONSTRATION

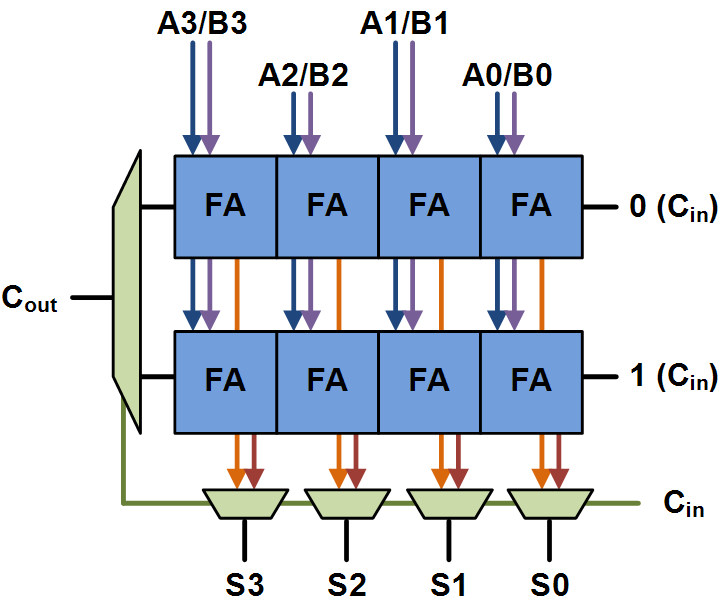
6.1 EXISTING DESIGN

CARRY SELECT ADDER

In electronics, a carry select adder is a particular way to implement an adder, which is a logic element that computes the (n+1) bit sum of two n-bit numbers. The carry-select adder is simple but rather fast, having a gate level depth of O(\sqrt n).The carry-select adder generally consists of two ripple carry adders and a multiplexer. Adding two n-bit numbers with a carry-select adder is done with two adders (therefore two ripple carry adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known.

The number of bits in each carry select block can be uniform, or variable. In the uniform case, the optimal delay occurs for a block size of \lfloor \sqrt n \rfloor. When variable, the block size should have a delay, from addition inputs A and B to the carry out, equal to that of the multiplexer chain leading into it, so that the carry out is calculated just in time. The O(\sqrt n)delay is derived from uniform sizing, where the ideal number of full-adder elements per block is equal to the square root of the number of bits being added, since that will yield an equal number of MUX delays.

**Basic building block**



**Fig.7** Basic building block of CSLA

Above is the basic building block of a carry-select adder, where the block size is 4. Two 4-bit ripple carry adders are multiplexed together, where the resulting carry and sum bits are selected by the carry-in. Since one ripple carry adder assumes a carry-in of 0, and the other assumes a carry-in of 1, selecting which adder had the correct assumption via the actual carry-in yields the desired result.

## Uniform-sized adder

A 64-bit carry-select adder with a uniform block size of 4 can be created with fifteen of these blocks and a 4-bit ripple carry adder. Since carry-in is known at the beginning of computation, a carry select block is not needed for the first four bits. The delay of this adder will be 4 full adder delays, plus three MUX delays.

**64-Bit CSLA(4\*16)**

63:60 RCA

11:8 RCA

7:4 RCA

3:0 RCA

A[63:60] B[63:60] A[11:8] B[11:8] A[7:4] B[7:4] A[3:0] B[3:0]

0 ..... 0 0

63:60 RCA

11:8 RCA

7:4 RCA

1 ... . 1 1

4

cy 63:60 mux ..... cy 11:8 mux C8 cy 7:4 mux C4

S[3:0]

4 4 4

Cout Sum[63:60] Sum[11:8] Sum[7:4]

**Fig 8** 64 uniform size.

**Variable-sized adder**

A 64-bit carry-select adder with variable size can be similarly created. Here we show an adder with block sizes of 2\*1+3\*2+4\*1+5\*10. This break-up is ideal when the full-adder delay is equal to the MUX delay, which is unlikely. The total delay is two full adder delays, and thirteen mux delays.

**64 bit CSLA (2\*1+3\*2+4\*1+5\*10)**

63:59 RCA

7:5 RCA

4:2 RCA

1:0 RCA

A[11:8] B[11:8] A[7:5] B[7:5] A[4:2] B[4:2] A[1:0] B[1:0]

0 0 0

63:59 RCA

7:5 RCA

4:2 RCA

1 1 1

2

cy 11:8 mux cy 7:5 mux C5 cy 4:2 mux C2

S[1:0]

4 3 3

Sum[11:8] Sum[7:5] Sum[4:2]

16:12 RCA

16:12 RCA

A[16:12] B[16:12] A[63:59] B[63:69]

63:59 RCa

0 .......... 0

63:69 RCA

1 ............... 1

16:12 mux cy ..................... 63:59 mux cy Cout

Sum[16:12] Sum[63.59]

**Fig.9** Non uniform Size

**RIPPLE CARRY ADDER**

Multiple full adder circuits can be cascaded in parallel to add an N-bit number. For an N- bit parallel adder, there must be N number of full adder circuits. A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage. In a ripple carry adder the sum and carry out bits of any half adder stage is not valid until the carry in of that stage occurs.A propagation delay inside the logic circuitry is the reason behind this. Propagation delay is time elapsed between the application of an input and occurrence of the corresponding output. Consider a NOT gate, When the input is “0” the output will be “1” and vice versa. The time taken for the NOT gate’s output to become “0” after the application of logic “1” to the NOT gate’s input is the propagation delay here. Similarly the carry propagation delay is the time elapsed between the application of the carry in signal and the occurrence of the carry out (Cout) signal. Circuit diagram of a 16-bit ripple carry adder is shown below.The block diagram of the 64 bit ripple carry adder is shown in figure 7

**64 Bit Ripple Carry adder**

A 64 bit operand B 64 bit operand

1bit full adder

1bit full adder

1bit full adder

1bit full adder

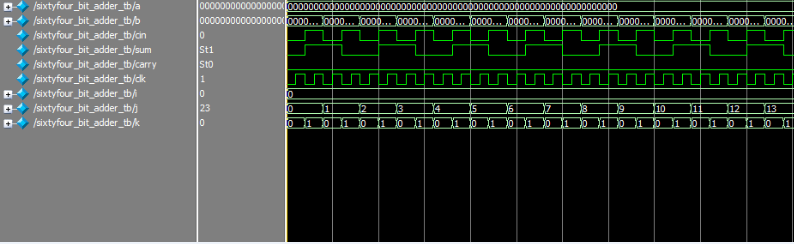
...............

Sum 64 bit operand

Cout 1 bit operand

**Fig10.**64 bit ripple carry adder

The output of the 64 bit ripple carry adder is shown in figure 11.

****

**Fig 11.**Output of the 64 bit ripple carry adder

**SIMULATION RESULT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 10179 | 2574 | 181.88 | 1309998.986 | 131180.866 |

6.2 PROPOSED DESIGN

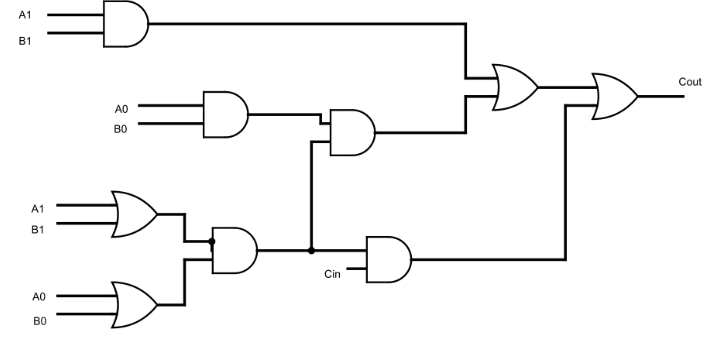
**2 BIT CARRY CIRCUIT**

The carry dependent and in-dependent circuit works as follows:

The carry in-dependent produces Carryout as 1 when the sum is greater than or equal to 4.This does not dependent on previous carry and produces result fast. The combinations for which Carryout do not depend on the previous carry are 1,3; 2,2; 2,3; 3,1; 3,2; 3,3.

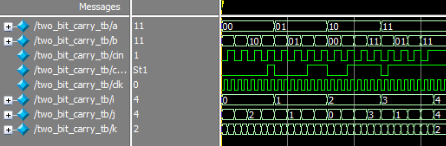
The carry dependent produces Carryout when the previous carry is 1 and the sum is equal to 3.It dependents on the previous carry and produces result when the previous carry is known. The combinations for which Carryout depends on the previous carry are 0,3; 1,2; 2,1. Therefore for remaining all combinations Carryout should be zero.

Two bit Carry dependent and in-dependent circuit is shown in Figure 12.



**Fig.12** 2 Bit Carry Diagram

The output of two bit carry is shown in Figure 13

****

**Fig.13** 2 Bit Carry output diagram

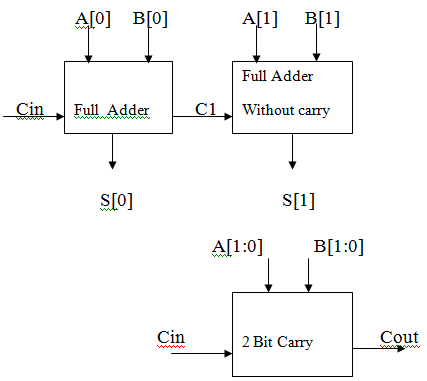
**SIMULATION RESULT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 349 | 32 | 2.317 | 1277.109 | 1279.426 |

**2-BIT ADDER CIRCUIT**

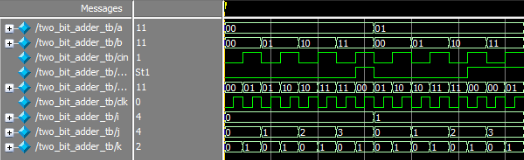
2 bit adder is used to find the addition of two 2-bit numbers

The two bit adder circuit is shown figure 14. The output of two bit adder circuit is shown in figure 15.two bit adder is used to find addition of two two bit numbers.



**Fig.14** Two bit adder circuit.

The output of two bit adder circuit is shown in Figure 15.



**Fig.15** Output of two bit adder circuit

**SIMULATION RESULT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 453 | 102 | 7.178 | 4270.766 | 4277.943 |

**3- BIT CARRY CIRCUIT**

The carry dependent and in-dependent circuit works as follows:

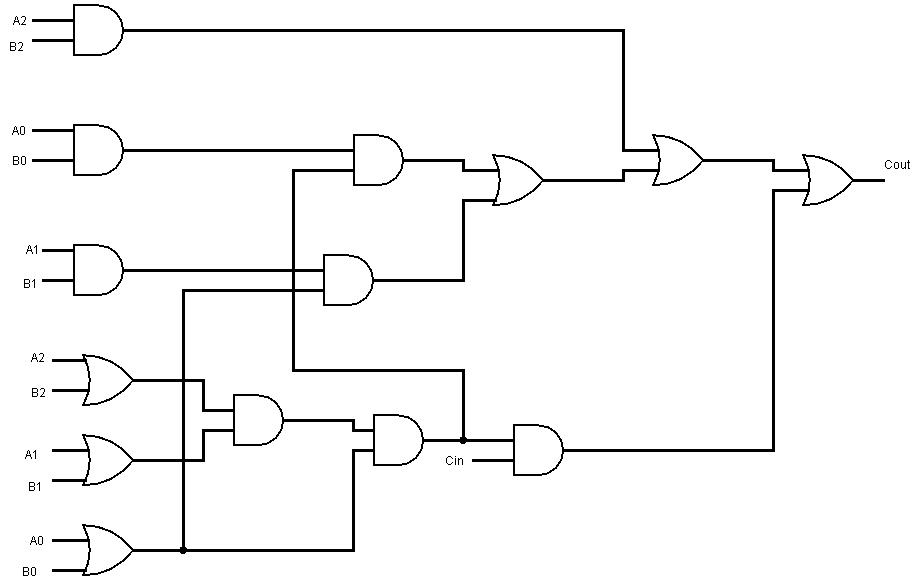
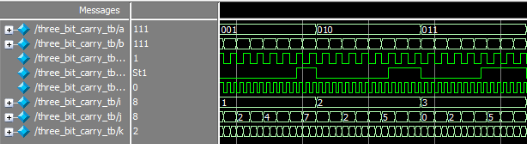
The carry in-dependent produces Carryout as 1 when the sum is greater than or equal to 8.This does not dependent on previous carry and produces result fast. The combination of inputs for which Carryout do not depend on the previous carry are.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Combi-nations | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  | X |
| 2 |  |  |  |  |  |  | X | X |
| 3 |  |  |  |  |  | X | X | X |
| 4 |  |  |  |  | X | X | X | X |
| 5 |  |  |  | X | X | X | X | X |
| 6 |  |  | X | X | X | X | X | X |
| 7 |  | X | X | X | X | X | X | X |

The carry dependent produces Carryout when the previous carry is 1 and the sum is equal to 7.It dependents on the previous carry and produces result when the previous carry is known. The combinations for which Carryout depends on the previous carry are

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Combi-nations | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  | X |
| 1 |  |  |  |  |  |  | X |  |
| 2 |  |  |  |  |  | X |  |  |
| 3 |  |  |  |  | X |  |  |  |
| 4 |  |  |  | X |  |  |  |  |
| 5 |  |  | X |  |  |  |  |  |
| 6 |  | X |  |  |  |  |  |  |
| 7 | X |  |  |  |  |  |  |  |

Therefore for remaining all combinations Carryout should be zero.

The Three bit carry dependent and in-dependent circuit and output of the circuit is shown in Figure 16 and Figure 17 respectively. **Fig16**Three bit carry ciruit 

**Fig.17** Output of three bit carry circuit

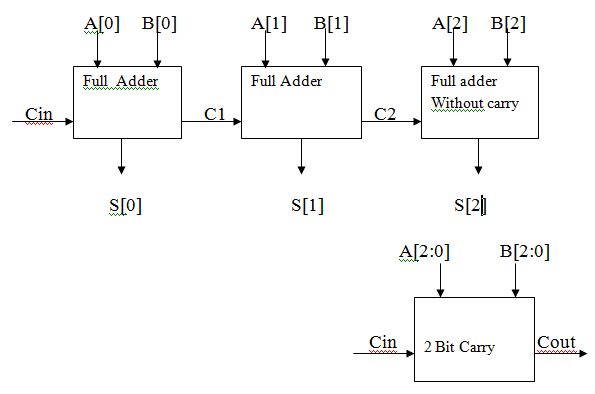
**SIMULATION RESULT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 486 | 49 | 3.655 | 2330.754 | 2334.409 |

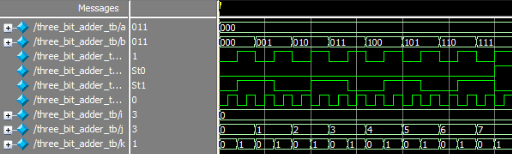
**3 BIT ADDER CIRCUIT**

3 bit adder is used to find the addition of two 3-bit numbers.

The three bit adder circuit is shown figure 18. The output of three bit adder circuit is shown in figure 19.Three bit adder is used to find addition of two three bit numbers.



**Fig18** Three bit adder circuit

****

**Fig 19** Output ofthree bit adder circuit

**SIMULATION RESULT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 610 | 159 | 11.347 | 7508.142 | 7619.489 |

**4-BIT CARRY CIRCUIT**

The carry dependent and in-dependent circuit works as follows:

The carry in-dependent produces Carryout as 1 when the sum is greater than or equal to 16.This does not dependent on previous carry and produces result fast. The combination of inputs for which Carryout do not depend on the previous carry are.

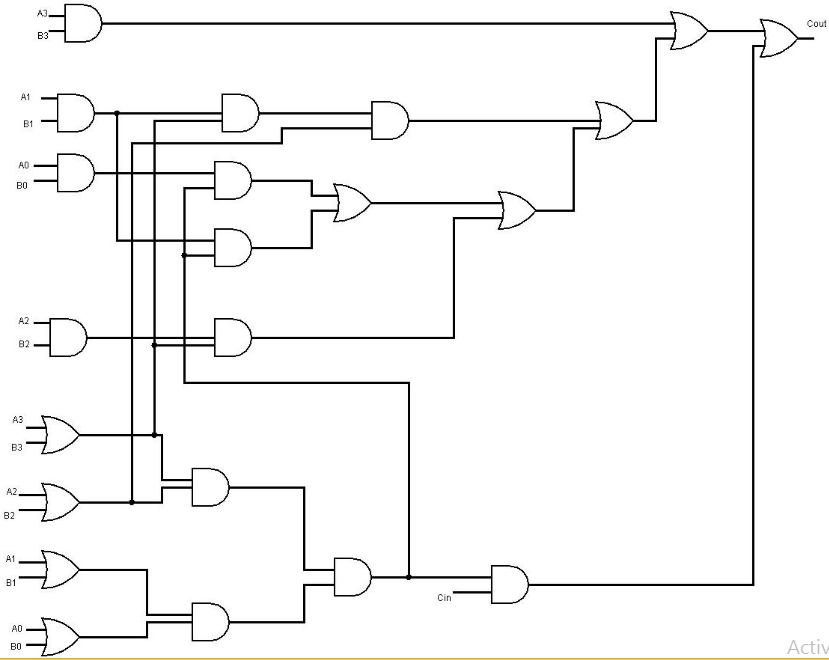
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Combinations** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **0** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **1** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **X** |
| **2** |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **X** | **X** |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  |  | **X** | **X** | **X** |
| **4** |  |  |  |  |  |  |  |  |  |  |  |  | **X** | **X** | **X** | **X** |
| **5** |  |  |  |  |  |  |  |  |  |  |  | **X** | **X** | **X** | **X** | **X** |
| **6** |  |  |  |  |  |  |  |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** |
| **7** |  |  |  |  |  |  |  |  |  | **X** | **X** | **X** | **X** | **x** | **X** | **X** |
| **8** |  |  |  |  |  |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **9** |  |  |  |  |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **10** |  |  |  |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **11** |  |  |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **12** |  |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **13** |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **14** |  |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **15** |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |

The carry dependent produces Carryout when the previous carry is 1 and the sum is equal to 15.It dependents on the previous carry and produces result when the previous carry is known. The combinations for which Carryout depends on the previous carry are

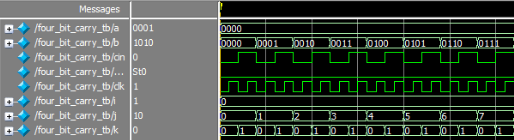
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **combinations** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **0** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **X** |
| **1** |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **X** |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |  |  | **X** |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  | **X** |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  | **X** |  |  |  |  |
| **5** |  |  |  |  |  |  |  |  |  |  | **X** |  |  |  |  |  |
| **6** |  |  |  |  |  |  |  |  |  | **X** |  |  |  |  |  |  |
| **7** |  |  |  |  |  |  |  |  | **X** |  |  |  |  |  |  |  |
| **8** |  |  |  |  |  |  |  | **X** |  |  |  |  |  |  |  |  |
| **9** |  |  |  |  |  |  | **X** |  |  |  |  |  |  |  |  |  |
| **10** |  |  |  |  |  | **X** |  |  |  |  |  |  |  |  |  |  |
| **11** |  |  |  |  | **X** |  |  |  |  |  |  |  |  |  |  |  |
| **12** |  |  |  | **X** |  |  |  |  |  |  |  |  |  |  |  |  |
| **13** |  |  | **X** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **14** |  | **X** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **15** | **X** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Therefore for remaining all combinations Carryout should be zero.

The Three bit carry dependent and in-dependent circuit and output of the circuit is shown in Figure 20 and Figure 21 respectively.

****

**Fig.20** Four bit carry circuit



**Fig 21** Output of four bit carry circuit

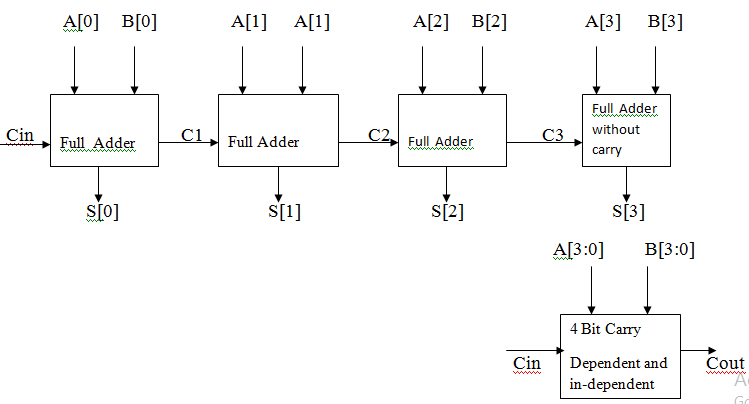
**SIMULATION RESULT**

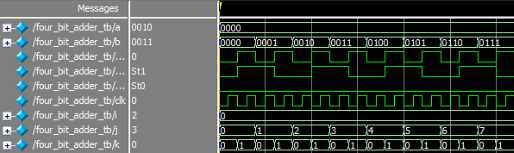
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 666 | 81 | 5.999 | 3541.539 | 3547.538 |

**4-BIT ADDER CIRCUIT**

The addition two 4-bit numbers are done by 4-bit adder.

The four bit adder circuit is shown figure 22. The output of four bit adder circuit is shown in figure 23.Four bit adder is used to find addition of two four bit numbers.

 **Fig.22** Four bit adder circuit



**Fig .23** Output of four bit adder circuit

**SIMULATION RESULT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 767 | 231 | 16.545 | 10410.746 | 10427.291 |

**5 -BIT CARRY**

The carry dependent and in-dependent circuit works as follows:

The carry dependent produces Carryout when the previous carry is 1 and the sum is equal to 31.It dependents on the previous carry and produces result when the previous carry is known. The combinations for which Carryout depends on the previous carry are

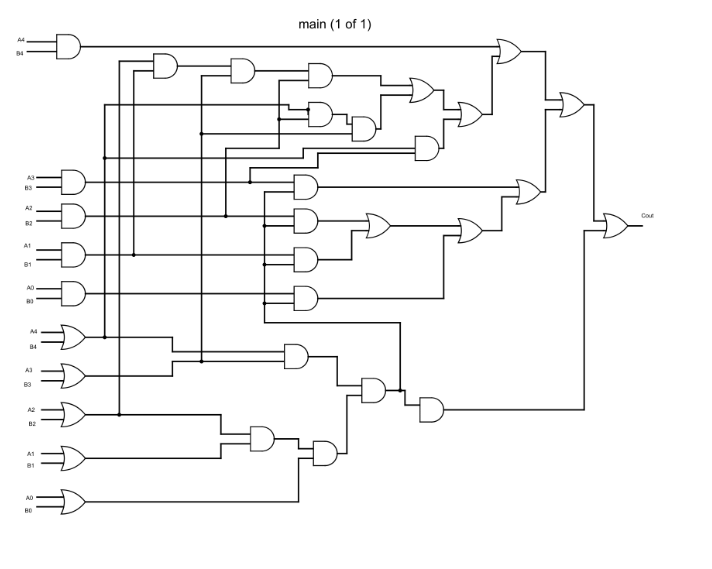
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inp  uts | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 19 |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 21 |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 22 |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 23 |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 24 |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 25 |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 26 |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 27 |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 28 |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 29 |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 31 | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

The carry in-dependent produces Carryout as 1 when the sum is greater than or equal to 32.This does not dependent on previous carry and produces result fast. The combination of inputs for which Carryout do not depend on the previous carry are.

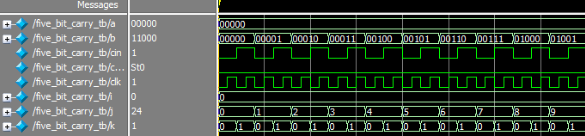
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inp  uts | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X |
| 11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X |
| 12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X |
| 13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 14 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 17 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 18 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 19 |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 21 |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 22 |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 23 |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 24 |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 25 |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 26 |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 27 |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 28 |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 29 |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 30 |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 31 |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

Therefore for remaining all combinations Carryout should be zero.

The Three bit carry dependent and in-dependent circuit and output of the circuit is shown in Figure 24 and Figure 25 respectively.

****

**Fig.24** Five bit carry circuit

****

**Fig.25** Outputof five bit carry circuit

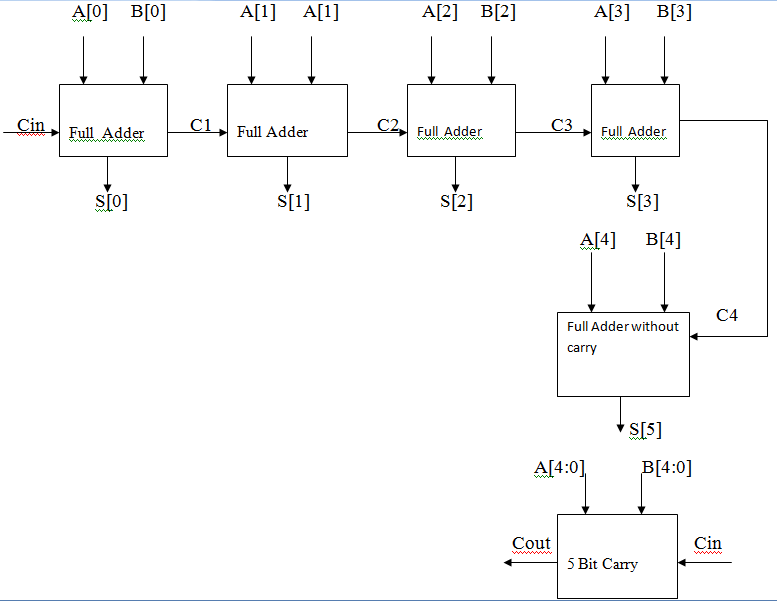
**SIMULATION RESULT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 721 | 116 | 8.690 | 5251.050 | 5259.740 |

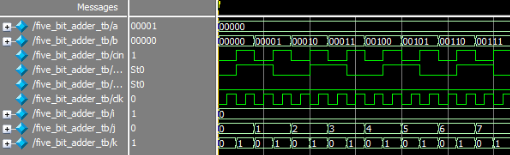
**5- BIT ADDER CIRCUIT**

The addition two 5-bit numbers is done by 5-bit adder.

The two bit adder circuit is shown figure 26. The output of two bit adder circuit is shown in figure 27

****

**Fig 26**. Five bit adder circuit

****

**Fig.27** Outputof five bit adder circuit

**SIMULATION RESULT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Timing (ps) | Report Area (µm2) | Report Power (nw) | | |
| Leakage | Dynamic | Total |
| 923 | 307 | 22.043 | 13596.114 | 13618.157 |

**64 Bit adder (4\*16)**

4 bit sum

4 bit sum

4 bit sum

**PROPOSED 1**

A[3:0] B[3:0] A[7:4] B[7:4] A[11:8] B[11:8]

4 bit carry

4 bit carry

Cin C4 C8

S[3:0] S[7:4] S[11:8]

A[3:0] B[3:0] A[7:4] B[7:4] A[11:8] B[11:8]

4 bit sum

4 bit carry

Cin C4 C8

A[23:20] B[23:20] A[19:16] B[19:16] A[15:12] B[15:12]

4 bit sum

4 bit sum

C20 C16 C12

S[23:20] S[19:16] S[15:12]

A[23:20] B[23:20] A[19:16] B[19:16] A[15:12] B[15:12]

4 bit carry

4 bit carry

4 bit carry

C24 C16 C12 C20 C16 C12

4 bit sum

4 bit sum

4 bit sum

A[27:24] B[27:24] A[31:28] B[31:28] A[35:32] B[35:32]

4 bit carry

4 bit carry

C24 C28 C32

S[27:24] S[31:28] S[35:32]

A[27:24] B[27:24] A[31:28] B[31:28] A[35:32] B[35:32]

4 bit sum

4 bit carry

C24 C28 C32

A[47:44] B[47:44] A[43:40] B[43:40] A[39:36] B[39:36]

4 bit sum

4 bit sum

C44 C40 C36

S[47:44] S[43:40] S[39:36]

A[47:44] B[47:44] A[43:40] B[43:40] A[39:36] B[39:36]

4 bit carry

4 bit carry

4 bit carry

C48 C16 C12 C44 C40 C36

4 bit sum

4 bit sum

4 bit sum

A[51:48] B[51:48] A[55:52] B[55:52] A[59:56] B[59:56]

4 bit carry

4 bit carry

C48 C52 C56

S[51:48] S[55:52] S[59:56]

A[51:48] B[51:48] A[55:52] B[55:52] A[59:56] B[59:56]

4 bit sum

4 bit carry

C48 C52 C56

A[63:60] B[63:60]

C60

S[63:60]

A[39:36] B[39:36]

4 bit carry

**Fig 28** 64 Bit Adder Circuit(4\*16) C64 C60

**64 Bit adder (2\*2+3\*2+4\*1+5\*10)**

2 bit sum

3 bit sum

2 bit sum

**PROPOSED 2**

A[1:0] B[1:0] A[3:2] B[3:2] A[6:4] B[6:4]

3 bit carry

2 bit carry

Cin C2 C4

S[1:0] S[3:2] S[6:4]

A[1:0] B[1:0] A[3:2] B[4:2] A[6:4] B[6:4]

3 bit sum

2 bit carry

Cin C2 C4

A[18:14] B[18:14] A[13:10] B[13:10] A[9:7] B[9:7]

4 bit sum

5 bit sum

C14 C10 C7

S[18:14] S[13:10] S[9:7]

A[18:14] B[18:14] A[13:10] B[13:10] A[9:7] B[9:7]

5 bit carry

4 bit carry

3 bit carry

C19 C16 C12 C14 C10 C7

5 bit sum

5 bit sum

5 bit sum

A[23:19] B[23:19] A[28:24] B[28:24] A[33:29] B[33:29]

5 bit carry

5 bit carry

C19 C24 C29

S[23:19] S[28:24] S[33:29]

A[23:19] B[23:19] A[28:24] B[28:24] A[33:29] B[33:29]

5 bit sum

5 bit carry

C19 C24 C29

A[48:44] B[48:44] A[43:39] B[43:39] A[38:34] B[38:34]

5 bit sum

5 bit sum

C44 C39 C34

S[48:44] S[43:39] S[38:34]

A[48:44] B[48:44] A[43:39] B[43:39] A[38:34] B[38:34]

5 bit carry

5 bit carry

5 bit carry

C49 C16 C12 C44 C39 C34

5 bit sum

5 bit sum

5 bit sum

A[53:49] B[53:49] A[58:44] B[58:54] A[63:59] B[63:59]

5 bit carry

5 bit carry

C49 C54 C59

S[53:49] S[58:54] S[63:59]

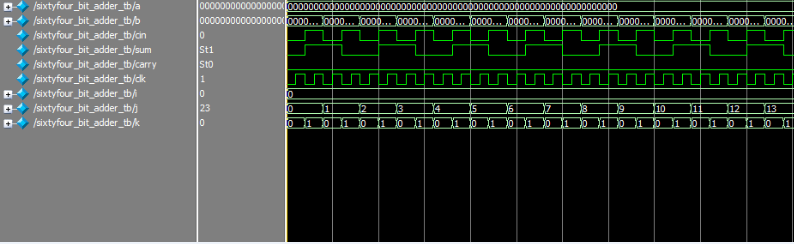
A[53:49] B[53:49] A[58:44] B[58:54] A[63:59] B[63:59]

5 bit carry

C49 C54 C59 C64

**Fig 29** 64 Bit adder circuit(2\*1+3\*2+4\*1+5\*10)

**Output Waveforms**

****

**Fig 30** 64 Bit adder output

**SIMULATION RESULTS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Size | Adders | Delay(ps) | Area(µm2) | Power(nw) | | |
|  |  |  |  | Leakage | Switching | Total |
| 64 | 4\*16  CSLA  Proposed | 4398  2296 | 5207  1851 | 368.123  132.482 | 264767.662  88065.675 | 266165.662  88198.15 |
| 2\*2+3\*2+4\*1+5\*10  CSLA  Proposed | 4029  3296 | 5271  3823 | 374.281  274.351 | 258319.732  177981.658 | 258694.013  178256.009 |

**GRAPHS**

**Fig.31** Delay graph

**Fig.32** Area graph

**Fig.33** Total power graph

**7.CONCLUSION**

I have analysed the logic operations involved in the conventional Ripple carry adder, CSLA and proposed circuit.I have eliminated all the redundant logic operations of the conventional Ripple carry adder and CSLA and proposed a new 64 bit adder circuit.In the proposed scheme carry is determined by carry dependent circuit and carry independent circuit. The delay of the particular block is 2 gates delay after carry arrives.The prosposed circuit design involves significantly less area, power and delay for 64 bit adder(4\*16 ,2\*2+3\*2+4\*1+5\*10) when compared to CSLA. The area of the proposed architecture in(4\*16) is shown nearly 64% reduction and in(2\*2+3\*2+4\*1+5\*10) by 27.47% using the optimizations used by the Cadence tool software. The area of the proposed architecture in(4\*16) is shown nearly 64% reduction and in(2\*2+3\*2+4\*1+5\*10) by 27.47% using the optimizations used by the Cadence tool software. The Delay in proposed adder(4\*16) has decreased by nearly 47.7% and for(2\*2+3\*2+4\*1+5\*10) 18.19% than the existing CSLA. The power dissipated by the existing model is compared with the proposed CSLA and it is noted that it produce a deduction of power in(4\*16) by 66.863% and in(2\*2+3\*2+4\*1+5\*10) by 31.1%.

**8. REFERENCES**

[1] O. J. Bedrij, “Carry-select adder,” *IRE Trans. Electron. Comput.*, pp.

340–344, 1962.

[2] B. Ramkumar, H.M. Kittur, and P. M. Kannan, “ASIC implementation

of modified faster carry save adder,” *Eur. J. Sci. Res.*, vol. 42, no. 1, pp.

53–58, 2010.

[3] T. Y. Ceiang and M. J. Hsiao, “Carry-select adder using single ripple

carry adder,” *Electron. Lett.*, vol. 34, no. 22, pp. 2101–2103, Oct. 1998.

[4] Y. Kim and L.-S. Kim, “64-bit carry-select adder with reduced area,”

*Electron. Lett.*, vol. 37, no. 10, pp. 614–615, May 2001.

[5] J. M. Rabaey*, Digtal Integrated Circuits—A Design Perspective*.

Upper Saddle River, NJ: Prentice-Hall, 2001.

[6] Y. He, C. H. Chang, and J. Gu, “An area efficient 64-bit square root

carry-select adder for lowpower applications,” in *Proc. IEEE Int. Symp.*

*Circuits Syst.*, 2005, vol. 4, pp. 4082–4085.

[7] Cadence, “Encounter user guide,” Version 6.2.4, March 2008

[8] K. K. Parhi, *VLSI Digital Signal Processing*. New York, NY, USA:Wiley,

1998.

[9] A. P. Chandrakasan, N. Verma, and D. C. Daly, “Ultralow-power electronics

for biomedical applications,” *Annu. Rev. Biomed. Eng.*, vol. 10, pp. 247–

274, Aug. 2008.

[10] O. J. Bedrij, “Carry-select adder,” *IRE Trans. Electron. Comput.*,

vol. EC-11, no. 3, pp. 340–344, Jun. 1962.

[11] Y. Kim and L.-S. Kim, “64-bit carry-select adder with reduced area,”

*Electron. Lett.*, vol. 37, no. 10, pp. 614–615, May 2001.

[12] Y. He, C. H. Chang, and J. Gu, “An area-efficient 64-bit square root carryselect

adder for low power application,” in *Proc. IEEE Int. Symp. Circuits*

*Syst.*, 2005, vol. 4, pp. 4082–4085.

[13] B. Ramkumar and H.M. Kittur, “Low-power and area-efficient carry-select

adder,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 2,

pp. 371–375, Feb. 2012.

[14] I.-C. Wey, C.-C. Ho, Y.-S. Lin, and C. C. Peng, “An area-efficient carry

select adder design by sharing the common Boolean logic term,” in *Proc.*

*IMECS*, 2012, pp. 1–4.

[15] S.Manju and V. Sornagopal, “An efficient SQRT architecture of carry select

adder design by common Boolean logic,” in *Proc. VLSI ICEVENT*, 2013,

pp. 1–5.

[16] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*,

2nd ed. New York, NY, USA: Oxford Univ. Press, 2010.

**CURRICULLAM VITAE**

**Name**  :Shashivardhan Malagari

**Father’s Name** :Mohan Reddy Malagari

**Date of birth** :04/05/1996

**Nationality**  :Indian

**Sex**  :Male

**Company Placed** :Not yet placed

**Permanant Address** : Kandawada village , chevalla, Hyderabad

**Mobile Number** :7871496156

**Email-Id** :shashivardhan.2014@vit.ac.in

**CGPA**  :7.91