**Abstract**

**Square Root Carry Select Adder using BEC for Area Efficient Processors**

by Sri Shanmukha Naraharisetti

Arithmetic functions are the basic operations in many data processors. In most of the digital adders the speed of addition is limited by time taken for the carry to propagate through the adder. Square Root Carry Select Adder (SQRT CSLA) is one of the adders which is used to perform the addition operations very fast [1], but due to the ripple carry adders (RCA) used in the SQRT CSLA module the number of AOI gates used here are more compared to the AOI gates used in the modified SQRT CSLA which leads them to occupy more area. Modified SQRT CSLA limits the number of gates (area) by using Binary to Excess 1 Converters (BEC) in place of RCA’s.

The area, power and delay parameters of the modified SQRT CSLA using BEC are determined from simulations with synopsys tools and compared with SQRT CSLA which uses RCA. Based on the comparison of results of the ordinary module and modified module, the better design of the two models is illustrated.

**Acknowledgements**

I wish to express appreciation to my Master’s thesis advisor, Professor David Rancour for guiding me throughout the completion of my thesis. Thanks to his enormous patience and knowledge in the area, I was able to complete successfully the requirements of my thesis. I feel really grateful for the opportunity to work on this thesis, in which I have given the best of myself.

I would also like to thank Mr. Elliot Mednick, my supervisor in Advanced Micro Devices (AMD), I worked as a Co-op Engineer there and I learned to use new tools and a new assembler that only AMD uses for its processors which are a part of the design of a processor. I also would like to thank the other members of my thesis committee, Professor Gaurav Khanna and Professor Lance Fiondella. I appreciate all that I have also learned from these professors, as they as well are a very valuable asset to the university.

**Table of Contents**

List of Figures…………………………………………………………………………..viii List of Tables……………………………………………………………………………...x

[1. Introduction 1](#_Toc436926834)

[1.1. Problem Statement 2](#_Toc436926835)

[1.2. Why only BEC 2](#_Toc436926836)

[1.3. Language and Tools used 3](#_Toc436926837)

[1.4. Advantages of Modified Model over Ordinary SQRT CSLA 3](#_Toc436926838)

[1.5. Applications 3](#_Toc436926839)

[2. Adders used in this Thesis 4](#_Toc436926840)

[2.1. Carry Select Adder 4](#_Toc436926841)

[2.2. Adders used in the Construction of the SQRT CSLA 6](#_Toc436926843)

[2.2.1. ripple carry adder. 6](#_Toc436926844)

[2.2.2. half adder. 7](#_Toc436926846)

[2.2.3. full adder. 8](#_Toc436926849)

[3. Multiplexer 11](#_Toc436926852)

[4. Design and Simulation of SQRT CSLA in VCS Simulator 13](#_Toc436926855)

[4.1. Implementation of SQRT CSLA in VCS Simulator 13](#_Toc436926856)

[4.1.1. design compiler. 14](#_Toc436926858)

[4.1.2. IC compiler. 15](#_Toc436926859)

[4.2. Waveforms Simulation 16](#_Toc436926860)

[4.3. Schematics 17](#_Toc436926862)

[4.4. Schematic View of 16 bit SQRT CSLA (Design Compiler) 18](#_Toc436926864)

[4.5. Functioning of the SQRT CSLA 18](#_Toc436926866)

[4.6. Inner View of 5 bit RCA of the SQRT CSLA 20](#_Toc436926867)

[4.7. Layout View of 16 bit SQRT CSLA (IC Compiler) 21](#_Toc436926869)

[4.8. Power, Area and Timing Reports of SQRT CSLA 22](#_Toc436926871)

[4.8.1. power consumed by the SQRT CSLA module (power analysis report). 22](#_Toc436926872)

[4.8.2. area evaluation of SQRT CSLA module. 24](#_Toc436926874)

[4.8.3. timing report of SQRT CSLA module. 25](#_Toc436926876)

[5. Modified SQRT CSLA using BEC](#_Toc436926878) **[Error! Bookmark not defined.](#_Toc436926878)**27

[5.1. Design and Simulation of Modified SQRT CSLA 29](#_Toc436926881)

[5.1.1. waveforms simulation of modified SQRT CSLA in VCS compiler. 29](#_Toc436926882)

[5.2. Schematics 30](#_Toc436926884)

[5.2.1. schematic view of 16 bit MSQRT CSLA (design compiler). 31](#_Toc436926886)

[5.2.2. gate level schematic of BEC. 33](#_Toc436926889)

[5.2.3. layout view of 16 bit MSQRT CSLA (IC compiler). 34](#_Toc436926891)

[5.3. Power, Area and Timing Reports of MSQRT CSLA 35](#_Toc436926893)

[5.3.1. power consumed by the MSQRT CSLA module (power analysis report). ..35](#_Toc436926894)

[5.3.2. area evaluation of MSQRT CSLA module. 36](#_Toc436926896)

[5.3.3. timing report of MSQRT CSLA module. 37](#_Toc436926898)

[5.3.4. table comparing my results with others results. 38](#_Toc436926900)

[5.4. Graph Comparing the Power, Area and Timing of both Models 40](#_Toc436926902)

[5.4.1. graph comparing the area of SQRT CSLA and MSQRT CSLA. 40](#_Toc436926903)

[5.4.2. graph comparing the power of SQRT CSLA and MSQRT CSLA. 41](#_Toc436926906)

[5.4.3. graph comparing the timing of SQRT CSLA and MSQRT CSLA. 42](#_Toc436926909)

[6. Conclusion and Future Work 44](#_Toc436926912)

[6.1. Conclusion 44](#_Toc436926913)

[6.2. Future Work 44](#_Toc436926914)

[References 46](#_Toc436926915)

Appendix A………………………………………………………………………………48

Appendix B………………………………………………………………………………54

List of Figures

[Figure 2.1: Regular 16 bit SQRT CSLA [6] 6](#_Toc436926842)

[Figure 2.2: Block diagram of RCA 7](#_Toc436926845)

[Figure 2.3: Half Adder 8](#_Toc436926847)

[Figure 2.4: Full Adder 9](#_Toc436926850)

[Figure 3.1: 2:1 mux 12](#_Toc436926853)

[Figure 3.2: 4:1 mux 12](#_Toc436926854)

[Figure 4.1: Implementation of SQRT CSLA in VCS Simulator 13](#_Toc436926857)

[Figure 4.2: SQRT CSLA simulations in VCS simulator 16](#_Toc436926861)

[Figure 4.3: Block diagram of 16 bit SQRT CSLA 17](#_Toc436926863)

[Figure 4.4: Schematic view of 16 bit SQRT CSLA 18](#_Toc436926865)

[Figure 4.5: Inner view of 5 bit RCA 20](#_Toc436926868)

[Figure 4.6: Layout diagram of SQRT CSLA 21](#_Toc436926870)

[Figure 4.7: Power analysis report of SQRT CSLA 22](#_Toc436926873)

[Figure 4.8: Area analysis report of SQRT CSLA 24](#_Toc436926875)

[Figure 4.9: Timing analysis report of SQRT CSLA 25](#_Toc436926877)

Figure 5.1: Modified SQRT CSLA [6] 27

[Figure 5.2: A 4 bit BEC [6]](#_Toc436926880) **[Error! Bookmark not defined.](#_Toc436926880)**27

[Figure 5.3: Simulation of modified SQRT CSLA in VCS simulator 29](#_Toc436926883)

[Figure 5.4: Block diagram of 16 bit SQRT MCSLA 30](#_Toc436926885)

[Figure 5.5: Schematic view of MSQRT CSLA 31](#_Toc436926887)

[Figure 5.6: Schematic of MSQRT CSLA with the 4 bit BEC zoomed 32](#_Toc436926888)

[Figure 5.7: logic diagram of a 4 bit BEC 33](#_Toc436926890)

[Figure 5.8: Layout diagram of MSQRT CSLA 34](#_Toc436926892)

[Figure 5.9: Power analysis report of MSQRT CSLA 35](#_Toc436926895)

[Figure 5.10: Area analysis report of MSQRT CSLA 36](#_Toc436926897)

[Figure 5.11: Timing analysis report of MSQRT CSLA 37](#_Toc436926899)

Figure 5.12: Graph comparing the area analysis 40

[Figure 5.13: Graph comparing the percentage difference in area 41](#_Toc436926905)

Figure 5.14: Graph comparing the power analysis 41

[Figure 5.15: Graph comparing the percentage difference of power 42](#_Toc436926908)

[Figure 5.16: Graph comparing the timing analysis 43](#_Toc436926910)

[Figure 5.17: Graph comparing the percentage difference of delay 43](#_Toc436926911)

List of Tables

[Table 2.1: Truth Table of Half Adder 8](#_Toc436926848)

[Table 2.2: Truth Table of Full Adder…………………………………………………….10](#_Toc436926851)

[Table 2.3: Comparing Area, Delay and Power Results from the Papers [6, 10] 38](#_Toc436926901)

1. **Introduction**

As the scale of integration increases, 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 power consumption remain to be the two major design goals, area has become a critical concern in today’s VLSI system design.

Addition usually widely impacts the overall performance of digital systems and a crucial arithmetic function. In electronic applications, adders are most widely used. Applications where these are used include multipliers, DSP to execute various algorithms like FFT, FIR, and IIR. Wherever concept of multiplication comes adders come in to the picture. As we know, millions of instructions per second are performed in microprocessors.

A VLSI designer must optimize the area along with timing and power in a design so that there is enough resource availability for future enhancement on the chip. These three constraints together are very difficult to satisfy, so depending on demand or application some compromise between constraints has to be made. Ripple carry adders exhibit the most compact design but the slowest speed. Whereas carry look ahead is the fastest one but consumes more area. The carry select adder acts as a compromise between the two adders.

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 SQRT CSLA is used in many computational systems to alleviate the problem of carry propagation delay by independently generating multiple carries and then selecting a carry to generate the sum.

**1.1. Problem Statement**

The regular SQRT CSLA architecture uses ripple carry adders which uses more gates there by utilizes more Area. The problem in ordinary SQRT CSLA design is the number of full adders are increased then the circuit complexity also increases. Since number of full adder cells are greater, this doubles the area of the design.

The structure of CSLA is such that there is further scope of reducing the area . Simple and efficient gate level modification is used in order to reduce the area of SQRT CSLA.

In this thesis, a modified 16 bit SQRT CSLA will be developed which uses n+1 bit, where n is the adder bit width replacing one of the n (n is the adder bit width) bit RCA with carry input bit 1 in each group from the second group to generate partial sum and carry, the results of modified SQRT CSLA will be compared with regular SQRT CSLA results. The analysis of the result show that the SQRT CSLA with BEC is better than the conventional SQRT CSLA in conserving area with considered amount of delay.

**1.2. Why only BEC**

In this thesis an n+1 BEC is used to replace a n bit RCA because the BEC uses less number of gates than that of the n bit full adder structure of the RCA which reduces the area. Since my objective in this thesis is to optimize the area used by the AOI gates a BEC would be a very good replacement than using any other alternative. There are some replacements which have a little power advantage like the carry skip Adder, Though it is fast it consumes most of the area , similarly a single RCA though it is compatible compared to some adders it is not fast enough .These kind of designs are used only in power efficient applications based on the requirement.The SQRT CSLA using BEC eliminates the delay and Excess area being occupied and acts as a compromise for these both adders.

**1.3. Language and Tools used**

* Verilog (HDL)
* Synopsys (VLSI software tool suite)

**1.4. Advantages of Modified Model over Ordinary SQRT CSLA**

* Less area (less complexity)

**1.5. Applications**

* Arithmetic logic units
* High Speed multiplications
* Advanced microprocessor design
* Digital signal processing

1. **Adders used in this Thesis**

In electronics, an adder or summer is a [digital circuit](http://en.wikipedia.org/wiki/Digital_circuit) that performs [addition](http://en.wikipedia.org/wiki/Addition) of numbers. In many [computers](http://en.wikipedia.org/wiki/Computer) and other kinds of processors, adders are used not only in the [arithmetic logic unit](http://en.wikipedia.org/wiki/Arithmetic_logic_unit)s, but also in other parts of the processor, where they are used to calculate addresses, table indices, etc.

Although adders can be constructed for many numerical representations, such as [binary-coded decimal](http://en.wikipedia.org/wiki/Binary-coded_decimal) or [excess-3](http://en.wikipedia.org/wiki/Excess-3), the most common adders operate on [binary](http://en.wikipedia.org/wiki/Binary_numeral_system) numbers. Other [signed number representations](http://en.wikipedia.org/wiki/Signed_number_representations) require a more complex adder.

**2.1. Carry Select Adder and SQRT CSLA**

A carry select adder is formed by chaining up equal number of adder stages, it comes in the category of conditional sum adder. The conditional sum adder works in some conditions. Sum and carry are calculated by assuming input carry as 1 and 0 prior the input carry comes. When actual carry input arrives, the actual calculated values of sum and carry are selected using a multiplexer. The conventional carry select adder consists of k/2 bit adder for the lower half of the bits i.e. least significant bits and for the upper half i.e. most significant bits (MSB’s) two k/2 bit adders. One adder assumes carry input as one for performing addition and another assumes carry input as zero. The carry out calculated from the last stage i.e. least significant bit stage is used to select the actual calculated values of output carry and sum. The selection is done by using a multiplexer. This technique of dividing adder into stages increases the area utilization but addition operation is faster.

The SQRT CSLA is constructed by equalizing the delay through two carry chains (inputs, carry output) and the block multiplexer signal from the previous stage, In a SQRT CSLA the ideal number of full adders per block is equal to the square of no of bits being added therefore it is called a SQRT CSLA. The SQRT CSLA generally is a combination of RCA’s and multiplexers, 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 SQRT 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. However ,the SQRT 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 (Input Carry 0)Cin=0 and (Input Carry 1)Cin=1, then the final sum and carry are selected by the multiplexers (mux) [6].The basic idea of this work is to use Binary to Excess-1 Converter(BEC) instead of RCA with Cin=1 in the regular SQRT CSLA to achieve lower area. The main advantage of this BEC logic comes from the lesser number of logic gates than the n-bit Full Adder (FA) structure.

The SQRT CSLA architecture in the figure below is a combination of Ripple carry adders and multiplexers. The RCA with Group 2 i.e. the 2bit RCA with Cin 0[3:2] and RCA with Cin 1[3:2] drives its output to two multiplexers, sum and carry are driven as outputs by the two RCA’s. Depending upon which input is to be selected, the select line of the mux selects two bits out of 4 Input bits and generates the sum bits. Similarly the other groups also function with RCA sending their outputs as inputs to the multiplexers.

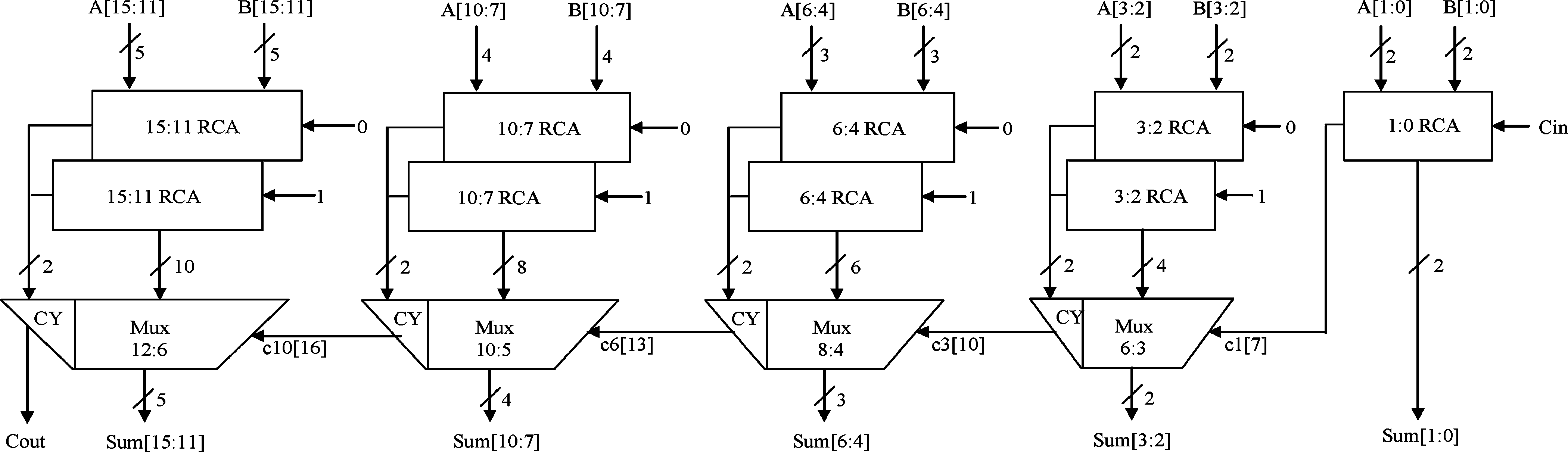


Figure 2.1: Regular 16 bit SQRT CSLA [6]

**2.2. Adders used in the Construction of the SQRT CSLA**

**2.2.1. ripple carry adder.**

Concatenating the N full adders forms N bit Ripple carry adder. In a RCA design the carry out of previous full adder becomes the input carry for the next full adder. As carry ripples from one full adder to the other, it traverses along the longest critical path and exhibits worst case delay. N is the size of the RCA or number of full adders in the design and based on the value of N the size of the RCA is determined.

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 is 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.

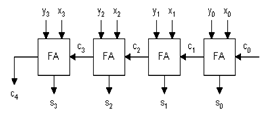


Figure 2.2: Block diagram of RCA

**2.2.2. half adder.**

The half adder adds two one-bit binary numbers. It has two Inputs A and B and two outputs sum(S) and carry(C).

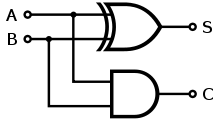
[](http://en.wikipedia.org/wiki/File:Half_Adder.svg)

Figure 2.3: Half Adder

The Boolean expression for half Adder is given as

S=A ⊕ B and C=A∙B

Table 2.1: Truth Table of Half Adder

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**2.2.3. full adder.**

A full adder adds binary numbers and accounts for values carried in as well as out. In the table 2.2 A & B are the operands and Cin is a bit carried in from the next less significant stage. The full-adder is usually a component in a cascade of adders, which adds 8, 16, 32, etc. binary numbers.

In the SQRT CSLA design full adders are used in the construction of ripple carry adders. A full adder in the schematic always comes into light when the carry in is set to one whenever the value of Cin is 0 the full adder gets converted to a half adder.

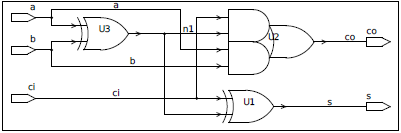


Figure 2.4: Full Adder

A full adder can be implemented in many different ways such as with a custom [transistor](http://en.wikipedia.org/wiki/Transistor)-level circuit or composed of other gates. One example implementation is with

S = A ⊕ B ⊕ Cin.

Cout = (A∙B) + (Cin . (A ⊕ B)).

In this implementation, the final [OR gate](http://en.wikipedia.org/wiki/OR_gate) before the carry out output may be replaced by an [XOR gate](http://en.wikipedia.org/wiki/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

Cout = (A∙B) ⊕ (Cin . (A ⊕ B)).

Table 2.2: Truth Table of Full Adder

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Cin | Cout | S |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

1. **Multiplexer**

A multiplexer is a device that selects one of several analog or digital input signals and forwards the selected input into a single line. A multiplexer of inputs has n select lines, which are used to select which input line to send to the output. Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth.

An electronic multiplexer can be considered as a multiple-input, single-output switch. The schematic symbol for a multiplexer is an isosceles trapezoid with the longer parallel side containing the input pins and the short parallel side containing the output pin.

In digital circuit design, the selector wires are of digital value. In the case of a 2-to-1 multiplexer, a logic value of 0 would connect A to the output while a logic value of 1 would connect B to the output which can be seen in the below figure. In larger multiplexers, the number of selector pins is equal to where **n** is the number of inputs.

The schematic in the figure below shows a 2:1 multiplexer where A and B are the two inputs, S0 is the selector input, and Z is the output. The select wire connects the desired input to the output. A straightforward realization of this 2:1 multiplexer would need 2 AND gates, an OR gate, and a NOT gate. The Boolean expression for a 2:1 mux is given as **(\S0∙A+ S0 ∙ B)**.

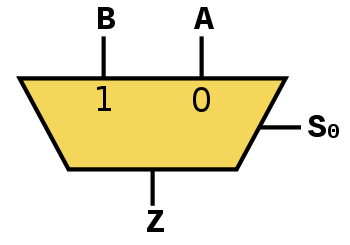
[](http://en.wikipedia.org/wiki/File:Multiplexer_2-to-1.svg)

Figure 3.1: 2:1 mux

Larger multiplexers are also common and, as stated above, require  selector pins for **n** inputs. Other common sizes are 4:1, 8:1, and 16:1. Since digital logic uses binary values, powers of 2 are used (4, 8, and 16) to control a number of inputs for the given number of selector inputs.

The schematic in the figure below shows a 4:1 multiplexer where A and B are the two inputs, S0 and S1 are the selector inputs, and Z is the output. The select wire connects the desired input to the output.

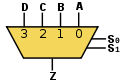
[](http://en.wikipedia.org/wiki/File:Multiplexer_4-to-1.svg)

Figure 3.2: 4:1 mux

**4. Design and Simulation of SQRT CSLA in VCS Simulator**

**4.1. Implementation of SQRT CSLA in VCS Simulator**

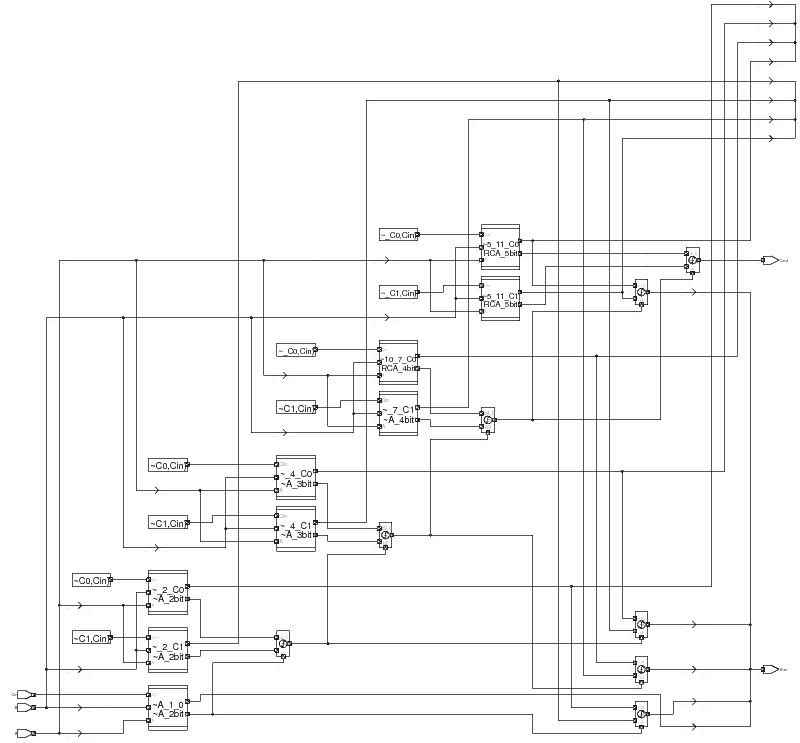


Figure 4.1: Implementation of SQRT CSLA in VCS Simulator

VCS is a high performance high capacity verilog simulator that is used for high level verification. It enables a user to compile and simulate verilog design. The design can be debugged and validated with the simulations. The simulated result can be viewed from the features provided by this tool. VCS supports all levels of design descriptions, VCS is integrated with many third party tools like the test bench tools.

VCS is used to compile and simulate the design. It builds the schematic from the imported verilog code when the code is run. After the code is synthesized with no errors we can view the waveforms by selecting the option add to waves and new wave form view. Selecting the inputs and outputs and running the wave for certain amount of simulation time in seconds will display some waves. We can view the outputs of the waveforms by setting the Radix to ASCII, Binary, Octal, Decimal and Hexadecimal form. If we don’t have a test bench we can just use the Force option and set the input values and based upon the output values we can check the design functionality is exactly as expected.

**4.1.1. design compiler.**

Design compiler is a Synthesis tool which takes an HDL file and a standard cell library as Input and produces a gate level netlist as output. It is a process of transforming RTL model into gate level netlist. It is used to synthesize the design, synthesis is the stage in the design flow where the code gets translated into a gate level schematic diagram.

The synopsys 32/28nm and 90nm Generic Libraries were designed specifically for teaching IC design with no IP restriction. Their targeted use is in the university classroom as well as Synopsys' own Customer Training Department for use with commercial customers. The standard cell library used for the synthesis here is 32/28nm. Design compiler performs many steps which includes RTL optimizations, RTL to unoptimized boolean logic, technology independent optimizations, and technology mapping to the available standard cells. The area, power and delay reports for both the models are generated by the design compiler and they can be seen in the report further.

**4.1.2. IC compiler.**

IC compiler performs place and route. This tool takes a synthesized gate-level netlist and a standard cell library as input and produces a layout as an output. There are some steps that the design has to go through for the fabrication of the chip in the IC compiler and they are as follows

* Design rule checks (DRC)
* Layout vs Schematic (LVS) check
* Parasitic Extraction
* Resimulation with Parasitics
* *Design rule checks –* Design rule checks series of parameters provided by the semiconductor manufacturers that enables the designer to verify the correctness of the mask set. The basic DRC checks are the spacing, width and enclose. The DRC check option is available from the verification menu.
* *Layout vs Schematic (LVS) check* – Layout vs Schematic is a test for all the connections if they are connected properly. This option compares the layout of a design with the original schematic and checks for the connection errors.

Since I have planned not to fabricate my chip I have worked on the software part in the Design compiler, performed the DRC and LVS checks in the IC compiler and removed the errors from the layout.

**4.2. Waveforms Simulation**

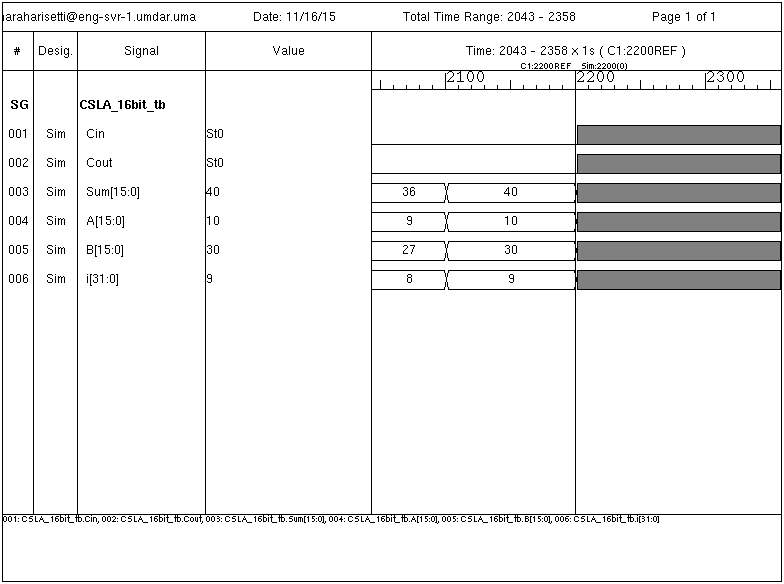


Figure 4.2: SQRT CSLA simulations in VCS simulator

Figure above illustrates the simulations results of 16 bit SQRT CSLA with the inputs as A, B and Cin (Carry select input). S is the sum of A, B and Cin andthe Cout is the Carry out (MSB). The simulation here is run for 2200 nanoseconds and the function of MSQRT CSLA remains the same i.e. addition like in the previous case but we are replacing the RCA with Cin 0 with a BEC. Since A, B are 16 bit decimal numbers and can be converted to binary their sum would be a 17 bit if there is a carry out and if there is no carry out the sum would be a 16 bit. There is no carry out here since small decimal values are used here but for five digit decimal number there will be a carry out.

In the above diagram during the simulation the radix is set to decimal and the sum is given as

S= A + B + Cin (0)

A=10 and B=30

S= 40

Since the SQRT CSLA is an adder it adds the inputs A, B and Cin and the output would be the sum of three of them with a carry out.

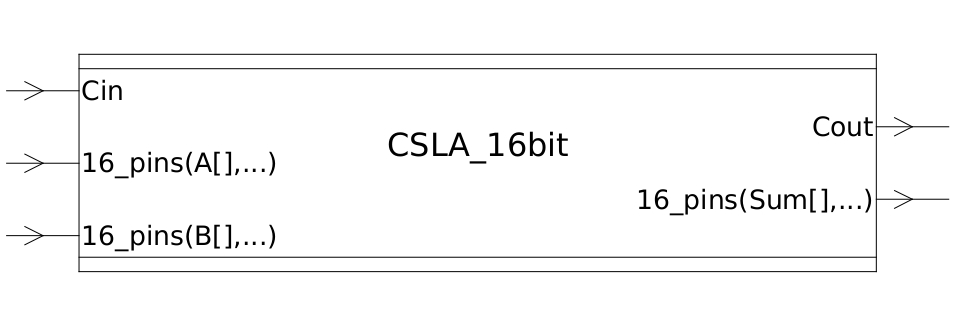
**4.3. Schematics**

Figure 4.3: Block diagram of 16 bit SQRT CSLA

The figure above is the block diagram of the ordinary SQRT CSLA module and the design has two 16 bit input bits are sum is a 16 it, there will be a 1 bit for carry input and the 1 bit carry output depends on the sum. If we go into the Block design we can see the schematic with different components that make up the entire module. Double clicking on the block design opens up the schematic of the module’s synthesized logic in the design compiler.

**4.4. Schematic View of 16 bit SQRT CSLA (Design Compiler)**

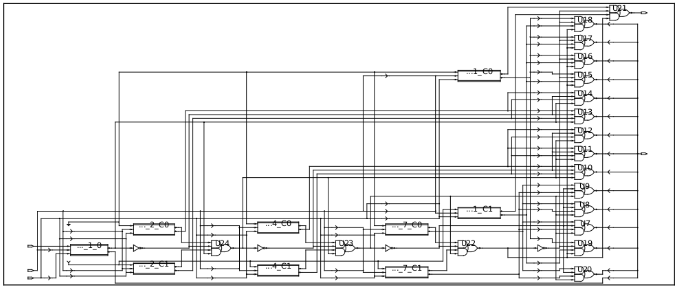


Figure 4.4: Schematic view of 16 bit SQRT CSLA

Figure above is the schematic of ordinary SQRT CSLA with 4 groups in which each group consists of 2 RCA and 1 multiplexer, each RCA is a combination of full adders, In total there are 9 RCA’s which are combination of 30 full adders.

**4.5. Functioning of the SQRT CSLA**

The First RCA in the first group with 2 bit inputs is a 2 bit RCA which drives the sum as one of the outputs and Cout from it to the multiplexer of next stage as a select line or control line.

The second group of the SQRT CSLA is also a 2 bit RCA with inputs A and B, it has two RCA’s one with Cin bit 0 and other with Cin bit 1. The two RCA’s together perform the sum operation in parallel and the correct sum and carry is selected with the help of 2:1 mux. The Cout of the previous stage is used as select line or control signal for the mux in this group and selects the correct sum from the RCA and if the value of select line is a binary 0 then it selects sum from RCA with Cin=0 and if the value of select line is a binary 1 then it selects sum from RCA with Cin 1. The carry out of the sum from both the RCA’s is also selected in the same way by the carry multiplexer depending on the value of the control line and the Carry of the sum.

Similarly the third group has a 3 bit inputs A and B and it also has two RCA’s with Cin as bit 0 and bit 1. These two RCA’s drive 3 bit sum and a carry out to the sum mux and the carry mux and based upon the carry out of the previous stage the correct sum and carry is selected either from RCA with Cin=0 or RCA with Cin=1 by the multiplexer.

Similarly the fourth group has two 4 bit inputs and it also has two RCA’s with bit 0 and bit 1. These two RCA’s drive 5 bits in parallel from two RCA to the mux and depending upon the carry out of the previous stage the delay and area for this group is also calculated.

The fifth group has two 5 bit inputs with two pairs of RCA’s with bit 0 and bit 1 as Cin the 5 bit sum and a carry bit if there is a carry present are driven as output to the 2:1 multiplexers as input. The mux here performs the same function as that of the previous stages. The delay and area are calculated for all groups together internally and total delay and area can are generated from the timing report and area report. In total 16 bit Sum from all groups together is generated as output, if the Cout is 1 then a 17 bit is generated, this can be seen for sum of large numbers.

**4.6. Inner View of 5 bit RCA of the SQRT CSLA**

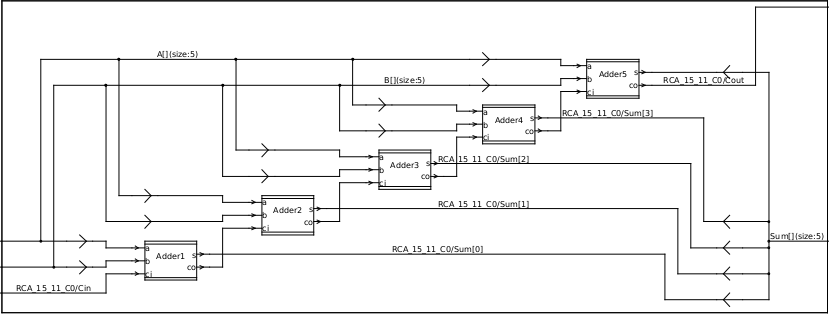


Figure 4.5: Inner view of 5 bit RCA

Figure above is the inner view of ripple carry adder which is a combination of 5 full adders. Here there is some propagation delay created before the Cout is valid and some delay also occurs when the sum is valid. Each adder in succession takes more propagation delay than the previous one and thus causes the overall delay. As carry ripples from one full adder to the other, it traverses along the longest critical path and exhibits worst case delay.

**4.7. Layout View of 16 bit SQRT CSLA (IC Compiler)**

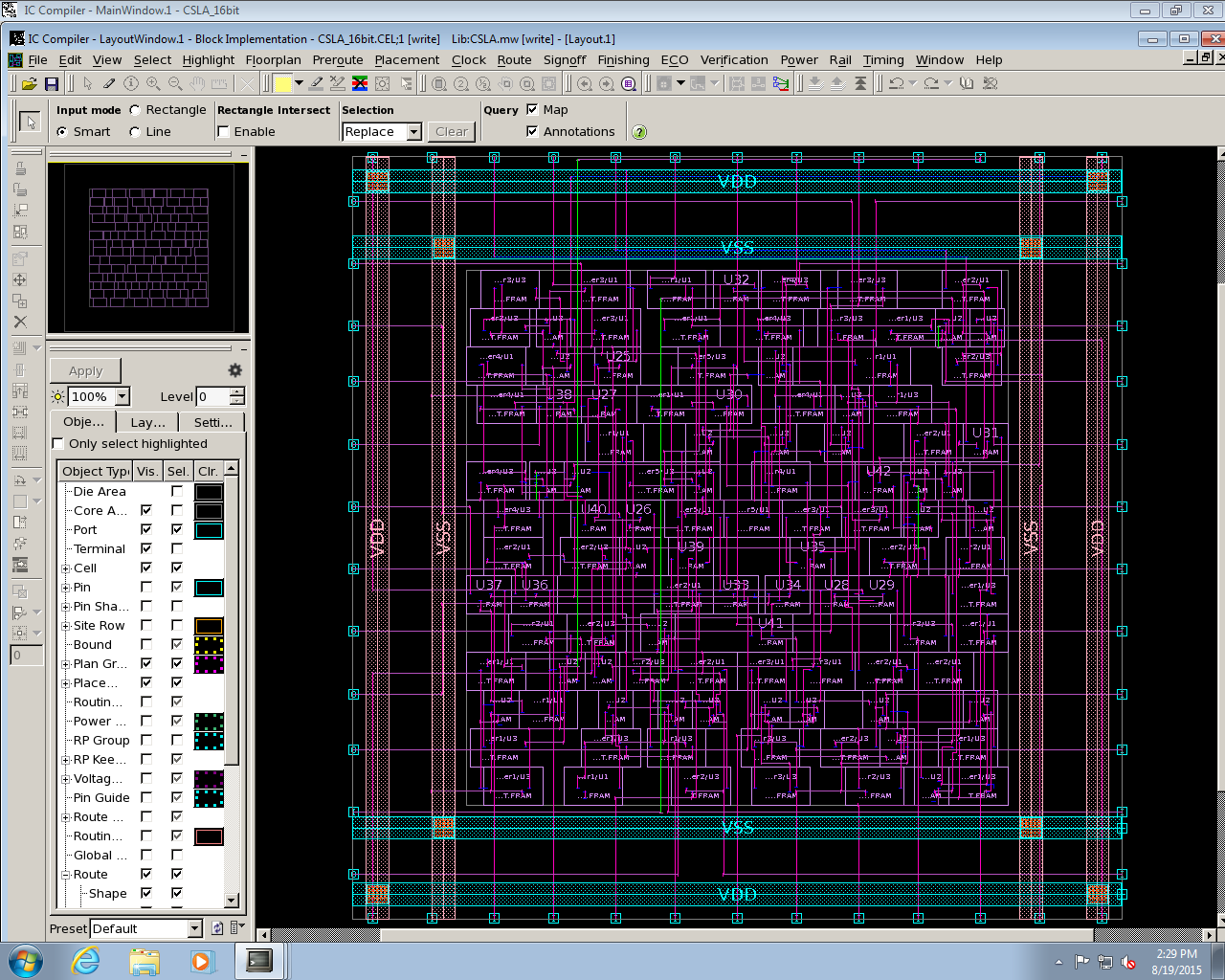


Figure 4.6: Layout diagram of SQRT CSLA

The above figure is the layout diagram of ordinary SQRT CSLA model and there are many cells which can be viewed as a separate cells if we go into the cell view. The layout of a module is used to perform the timing analysis and the area of individual cells also can be viewed in a layout view and these are already brought out in the reports.

The DRC and LVS checks on this design are performed and there are no errors in the DRC checks. In the LVS check there are few floating ports and since the floating port errors are to be treated as a false alarm we can negotiate them. After running the LVS check by unchecking the floating port errors and they are ignored in the next run.

**4.8. Power, Area and Timing Reports of SQRT CSLA**

**4.8.1. power consumed by the SQRT CSLA module (power analysis report).**

uw - Micro watts.

pw - Pico watts.

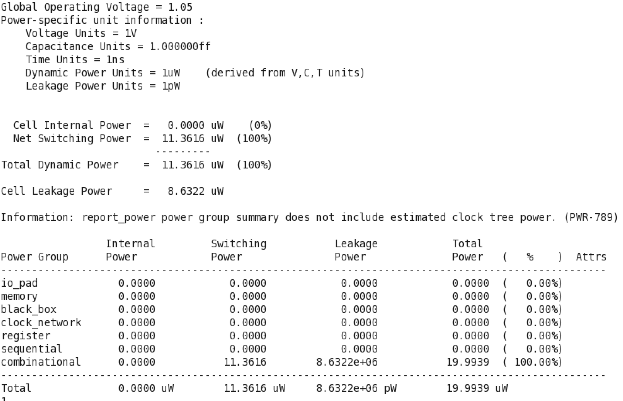


Figure 4.7: Power analysis report of SQRT CSLA

The above figure depicts the total power consumed by the ordinary SQRT CSLA module. The total power is the sum of internal power, Switching power and Leakage power. The following equation describes the four components of power. The circuit in the above figure is simulated using global operating voltage 1.05V at room temperature.

Total power = Internal power + Switching power + Leakage power.

The total power consumed by the ordinary SQRT CSLA is **19.99uw**. The power varies with the frequency and voltage. The relation between power and frequency, power and voltage is given as follows

**P= C.V2.f**

Where, capacitance (C) =1f

Voltage (V) =1.05V

Frequency (f) = 200MHz

As the frequency or voltage is increased the power also increases since the power is directly proportional to the frequency and voltage square.

**4.8.2. area evaluation of SQRT CSLA module.**

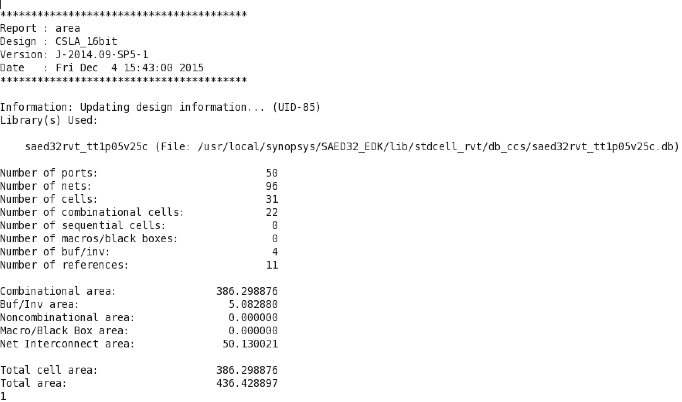


Figure 4.8: Area analysis report of SQRT CSLA

The Area consumed by the ports, nets, cells and other components used in the construction of SQRT CSLA can be viewed from the above table, the total area of the cell is the sum of all the ports, nets, cells, combinational cells, buffers and other interconnections. Since the RCA has more number of full adders and each full adder is a combination of AND, OR and XOR logic gates it occupies more area than that of a modified SQRT CSLA. The total area utilized by the SQRT CSLA is **436.42** square microns (um2).

**4.8.3. timing report of SQRT CSLA module.**

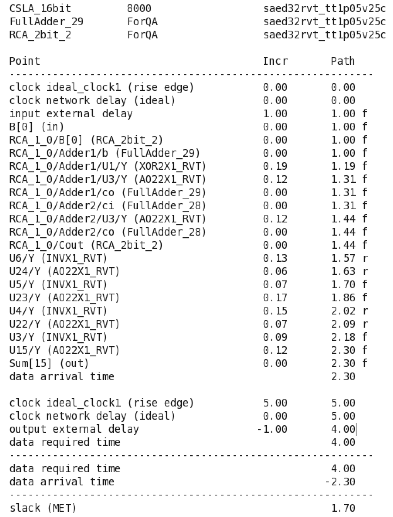


Figure 4.9: Timing analysis report of SQRT CSLA

The top 10 critical paths of the circuit are given from the timing report and it lets us know the worst path delay. The units in the timing report are given in Nano seconds, here **r** stands for rising edge and **f** stands for falling edge. The critical path is the path that causes the longest propagation delay between the input and output path. Here the critical path starts at Input port B[0] and ends at output port Sum[15].

If the circuit has calculated the worst path delay between two points with a very less positive slack then it is an indication of a good circuit which has utilized all of the timing for its computation. If the slack is negative then the design is not fast enough for the given clock period. The ideal clock period set in my simulation is 5ns (200MHz) with an input external delay of 1ns, the report gives the data arrival time (delay) and the slack. The slack in this case is 1.70 it is an indication of good circuit.

The delay for the ordinary SQRT CSLA is calculated from the critical path of the circuit and is **1.30n**s in this case. The delay or the critical path

The delay and slack are calculated from the timing report as follows

5-1-1=3ns and 3-1.30=1.70ns (slack)

Critical path =Data arrival time- input external delay=2.30-1=1.30ns.

**5. Modified SQRT CSLA using BEC**

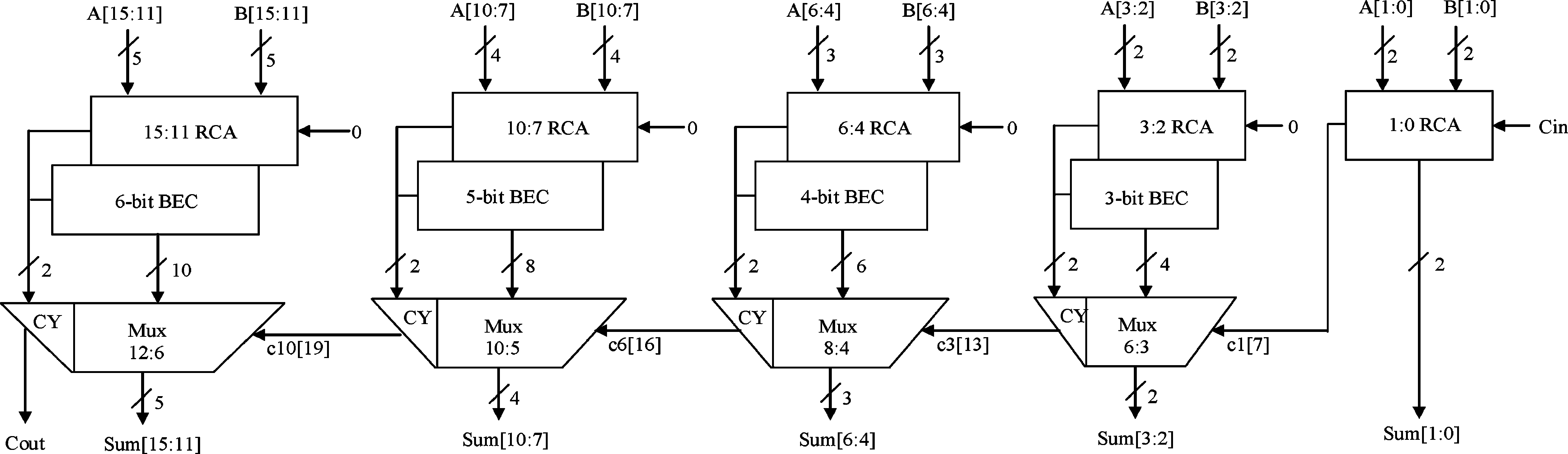


Figure 5.1: Modified SQRT CSLA [6]

The main idea of this work is to use BEC instead of the RCA with Cin=1 in order to reduce the area and power consumption of the regular SQRT CSLA. To replace the n bit RCA, an n+1 bit BEC is required.

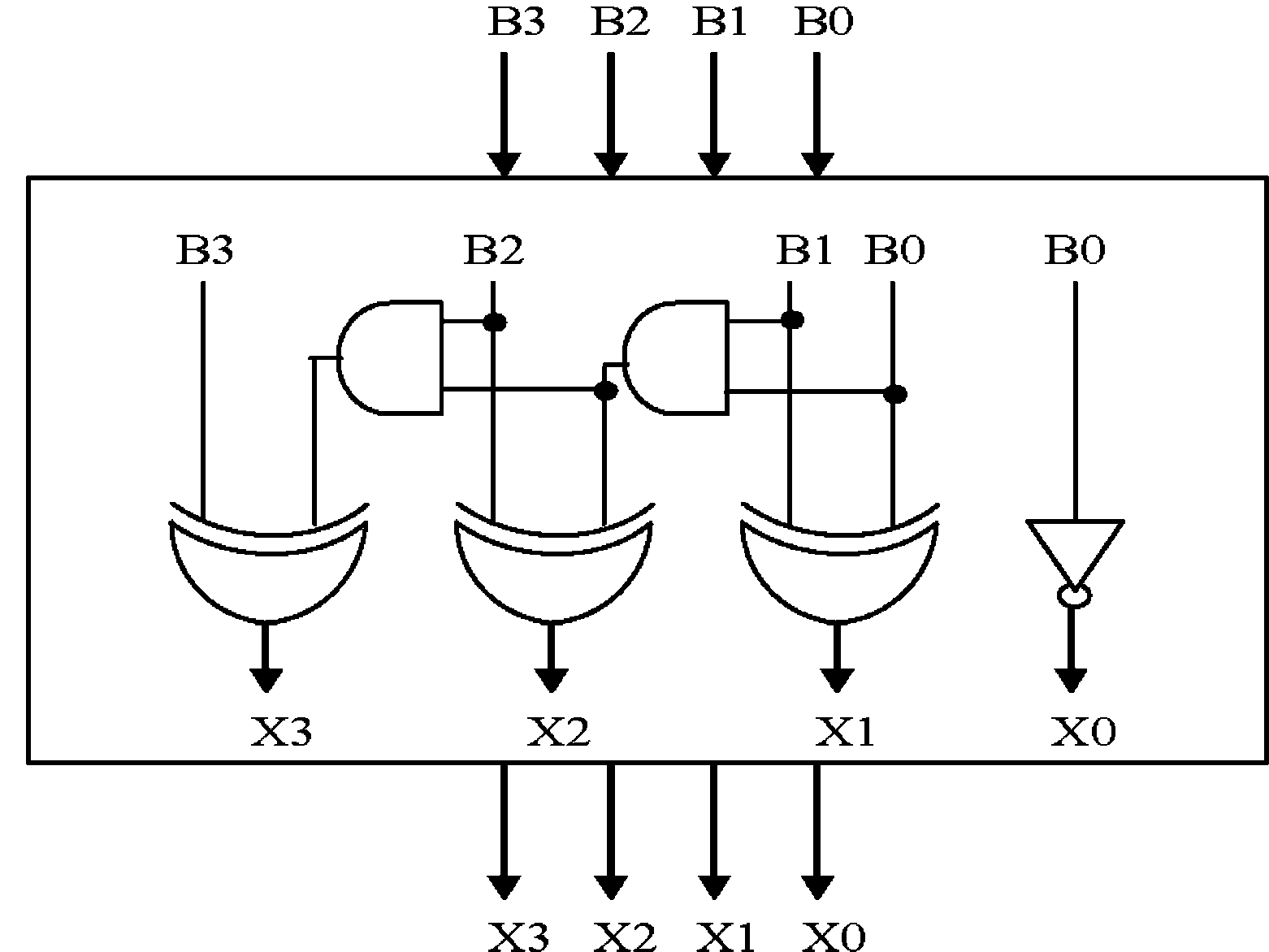


Figure 5.2: A 4 bit BEC [6]

Figure above illustrates how the basic function of the modified SQRT CSLA is obtained by using 4-bit BEC together with the mux. One input of the 8:4 mux gets inputs (B3, B2, B1, and B0) and another input of the mux is the BEC output. This produces the two possible partial results in parallel and the mux is used to select either the BEC output or the direct inputs according to the control signal Cin. The importance of the BEC logic

stems from the large silicon area reduction when the SQRT CSLA with large number of bits are designed.

If we consider a 3-bit BEC the Boolean expressions is listed as

X[0]=~B[0]

X[1]=B[0]^B[1]

Cout = B[2]^(B[0]&B[1])

The above logic is for a 3 bit BEC, Here ~B[0] is the invert of the first input bit, B[0]^B[1] are the OR of the two last input bits, B[2]^(B[0]&B[1]) is the AND of the 2 LSB ORED with the MSB. Similarly the logic for 4 bit, 5 bit and 6 bit BEC are also written and the remaining BEC’s can be seen in the schematic further.

The MSQRT CSLA also functions similar to the SQRT CSLA but in this model the sum bits of RCA with Cin as 0 are given as inputs to BEC instead of calculating sum in parallel like in the model using dual RCA’s. The BEC then considers the sum from the RCA as its input and then starts its sum operation and generates the output sum.

Depending upon the carry out of the previous stage the multiplexer selects the correct sum and carry either from the RCA or BEC. The carry out of the previous stage which is a binary 0 or binary1 acts as control signal for the multiplexer, if the control bit is 0 the sum is selected from the RCA and if the control bit is 1 the sum is selected from the BEC and this sum is used as output by the mux. The similar technique is used by the other groups also.

**5.1. Design and Simulation of Modified SQRT CSLA**

**5.1.1. waveforms simulation of modified SQRT CSLA in VCS compiler.**

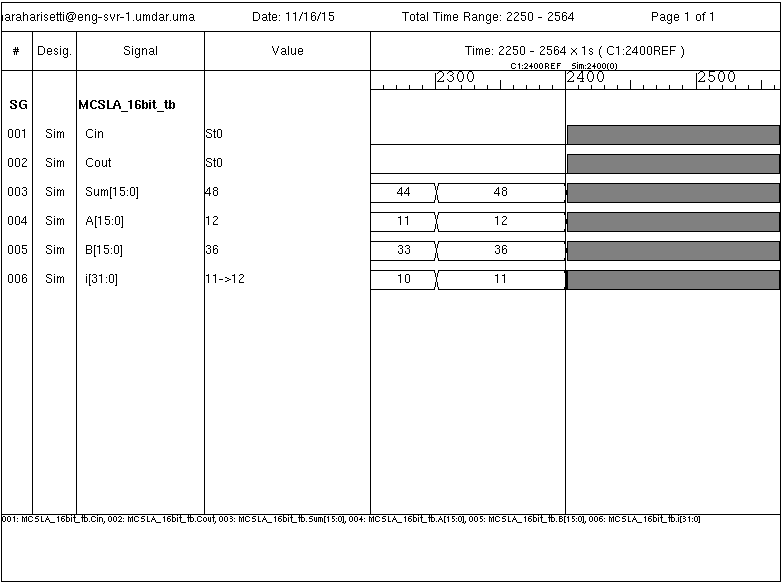


Figure 5.3: Simulation of modified SQRT CSLA in VCS simulator

Figure above illustrates the simulations results of 16 bit MSQRT CSLA with the inputs as A, B and Cin (Carry select input). S is the sum of A, B and Cin andthe Cout is the Carry out (MSB). The simulation here is run for 2400 nanoseconds, The Cin in the above waveform is 0. The function of MSQRT CSLA is just adding two numbers the result of A, B and Cin can be seen in the sum. Since A, B are 16 bit their sum also would a 17 bit if there is a carry out and if there is no carry out the sum would be a 16 bit.

In the above diagram for the simulation the radix is set to decimal and the sum is given as

S= A + B + Cin (0)

A=12, B=36

S= 48

Since the modified model is also an Adder and we are replacing one RCA with BEC the sum will not change i.e. addition of the inputs A, B and Cin and the result is same as that for the SQRT CSLA.

**5.2. Schematics**

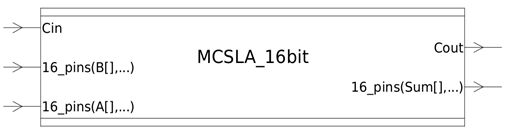


Figure 5.4: Block diagram of 16 bit SQRT MCSLA

The Block diagram of MSQRT CSLA in above figure is similar to that of the ordinary SQRT CSLA since it also uses 16 bits, but the difference is the modified model uses an n+1 BEC instead of an n bit RCA.

After browsing the design with the hierarchical view, then right click on the block design and select the schematic view option to display the schematic of the module’s synthesized logic.

**5.2.1. schematic view of 16 bit MSQRT CSLA (design compiler).**

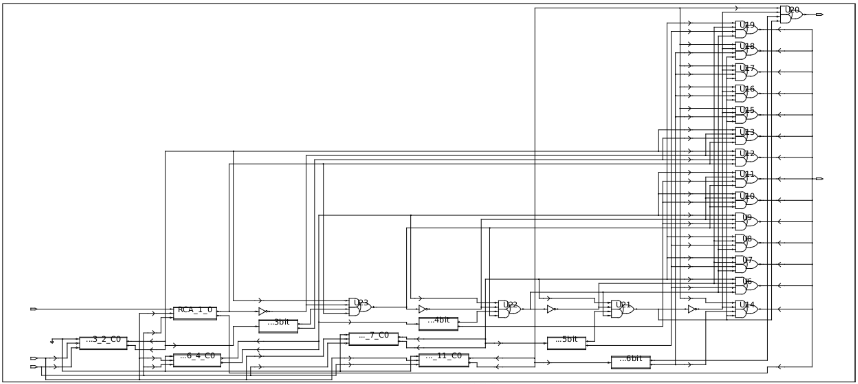


Figure 5.5: Schematic view of MSQRT CSLA

The above figure is the schematic of Modified SQRT CSLA and in this figure for every n bit RCA there is an n+1 bit BEC connected to it. The BEC can be seen in the figure below. There are 5 groups in this module and for each group delay and area are calculated internally by the compiler and then the total area and delay of the modified module is brought out.

The group 2 here has a 2 bit RCA and a 3 bit BEC is used instead of another RCA with a carry input 1, the sum bits of the RCA with carry input 0 are given as inputs to the BEC instead of calculating sum with Cin bit 1 as in the previous model. Depending on the carry out of the previous stage the sum is selected either from RCA or BEC with the help of 2:1 mux’s at each stage. The carry out from the sum is also selected by a multiplexer and it is sent as the control signal for next stage multiplexer. Area and delay for this group is calculated by the compiler in the back end.

Similarly for group 3, 4 and 5 also area and delay are calculated and the total area and delay for the entire module are brought up in the reports.

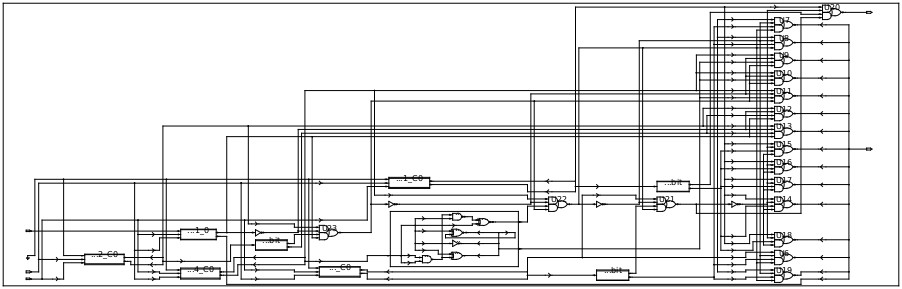


Figure 5.6: Schematic of MSQRT CSLA with the 4 bit BEC zoomed

In above figure we can see the 4 bit BEC is zoomed inside the schematic itself and it has its inputs from the 3 bit RCA. From the RCA .The 3 bit sum and 1 bit carry combine to form 4 inputs for the 4bit RCA and then based upon the carry out of previous stage the RCA and BEC are driven to the 2:1 multiplexers as inputs.

**5.2.2. gate level schematic of BEC.**

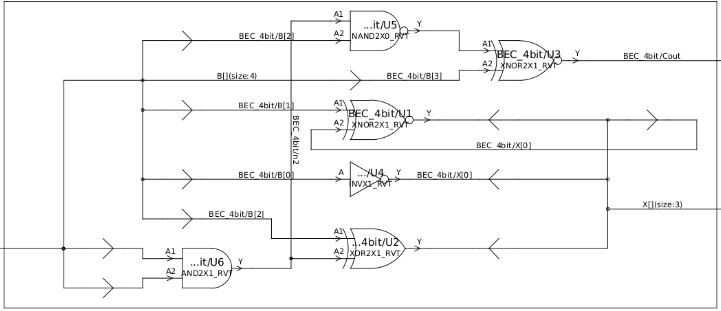


Figure 5.7: logic diagram of a 4 bit BEC

The above figure is a logic Diagram of the 4 bit BEC and it is a part of the schematic of MSQRT CSLA (fig 5.6) .The 4 bit BEC is zoomed to view gate level diagram and it works on the logic given below

X[0]=~B[0]

X[1]=B[0]^B[1]

X[2]=B[2]^(B[0]&B[1])

Cout=B[3]^(B[0]&B[1]&B[2])

The bits B[0] to B[3] are the sum bits of the RCA and the bits X[1] to Cout are the outputs of the BEC i.e. the sum bits of BEC.

**5.2.3. layout view of 16 bit MSQRT CSLA (IC compiler).**

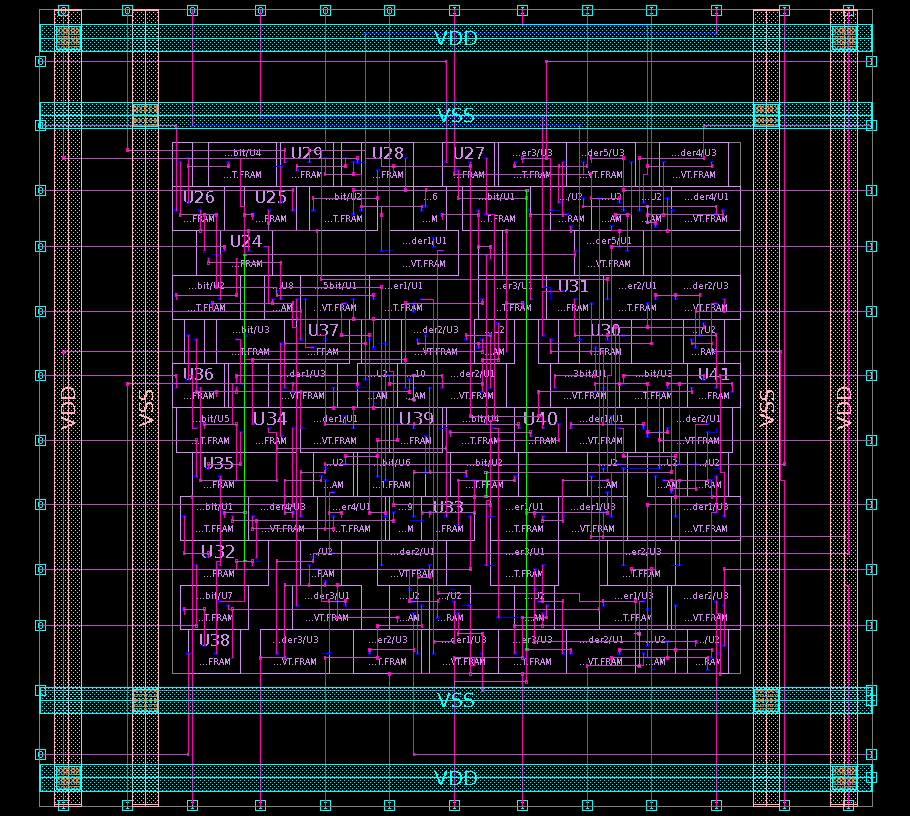


Figure 5.8: Layout diagram of MSQRT CSLA

The above figure is the layout diagram of ordinary modified SQRT CSLA model and there are many cells which can be viewed as a separate cells if we go into the cell view, each cell open ups into a FRAM view. The DRC and LVS checks on this design are performed and there are no errors in the DRC checks. In the LVS check there are few floating ports and since the pins in the floating port errors are an indication of false alarm when we rerun the LVS check by unchecking the floating port errors they are ignored.

**5.3. Power, Area and Timing Reports of MSQRT CSLA**

**5.3.1. power consumed by the MSQRT CSLA module (power analysis report).**

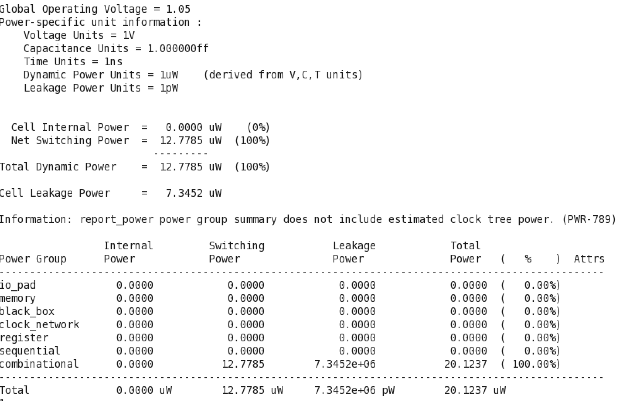


Figure 5.9: Power analysis report of MSQRT CSLA

The above figure depicts the total power consumed by the modified SQRT CSLA module and it is **20.12uw**. There is an increase of **0.6%** more power in this case. The leakage power is less in this case due to the less number of logic gates used here compared to the old model.

**5.3.2. area evaluation of MSQRT CSLA module.**

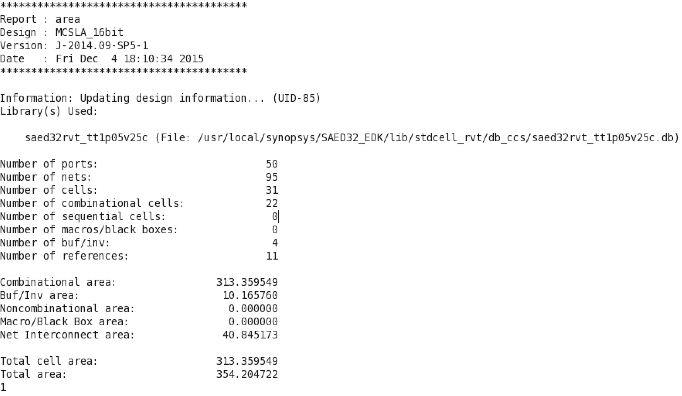


Figure 5.10: Area analysis report of MSQRT CSLA

The area consumed by the ports, nets, cells and other components used in the construction of SQRT CSLA can be viewed from the above table, the Total area is **354.20** Square microns. Since the BEC has less number of logic gates it occupies less area than that of a RCA in the SQRT CSLA. The MSQRT CSLA thus has an area advantage of **18.8%**.

**5.3.3. timing report of MSQRT CSLA module.**

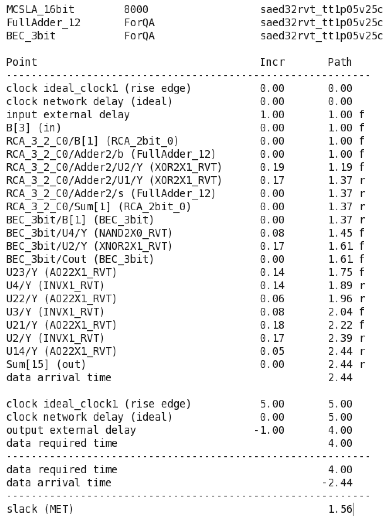


Figure 5.11: Timing analysis report of MSQRT CSLA

Figure above is the Timing report of the modified SQRT CSLA module and similar to the ordinary model this report also generates timing report for the top 10 critical paths of the circuit. The critical path starts at Input port bit B[3] and ends at output port Sum[15] and it is **1.44ns**.

**5.3.4. table comparing my results with results with reference papers.**

Table 2.3: Comparing Area, Delay and Power Results from the Papers [6, 10]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Adder | My results | | | | Results from paper [6] | | | Results from paper [10] |
| Parameter | Area (um2) | Combinational area (um2) | Delay (ns) | Power(uw) | Area (um2) | Delay (ns) | Power (uw) | Area (um2)  paper |
| 16bit SQRT CSLA | 436.42 | 386.29 | 1.30 | 19.99 | 2272 | 2.77 | 527.5 | 434 |
| 16bit MSQRT CSLA | 354.20 | 313.35 | 1.44 | 20.12 | 1929 | 3.04 | 471.8 | 337 |
| % increase or decrease | 18.8% | 18.8% | -10% | -0.6% | 15% | -9.8% | 10% | 22.5% |

Table above table shows my simulation results which are synthesized in synopsys software tool using SAED 32/28nm technology using a voltage of 1.05V at room temperature.

The area, delay and power of the SQRT CSLA and MSQRT CSLA which are generated from the reports are used in the table to find out the percentage increase or decrease. The positive numbers in the percentage column shows that there is a percentage decrease and the negative numbers are an indication of percentage increase.

My results show that the model using BEC has a decrease of 18.8% in area, the area without the buffers i.e. the combinational area also has been considered in the table. Eliminating the buffer/Inverter area and net interconnection area gives the combinational area which is the original area of the cell and the percentage decrease is same.

The delay comparison shows us a 10% increase in the delay for the new model when compared with the ordinary model, from the timing results we can see that the MSQRT CSLA takes only 0.14ns more time for its computation than the SQRT CSLA. The power consumed by the new model using BEC is increased by 0.6% compared to the ordinary SQRT CSLA.

The table 2.3 also shows the simulation results from paper [6] which are synthesized in cadence RTL compiler using TSMC 0.18um technology using a voltage of 1.8V and also shows area analysis of 16 bit model using BEC from paper [10] where they have used Modelsim for simulation and the design is synthesized using Xilinx 13.4, then the implementation is done in virtex5 FPGA kit.

From their results the ordinary model has an area of 434um2 and got reduced to 337um2 for the modified model. The FPGA numeric values differ when compared to ASIC design values so they have a 22.5% decrease of area for the new model.

From the results of paper [6] the new model has an area advantage of 16% which is 18.8% in my case. The delay for their new model has increased by 9.8% which is 10% in my case. Since I have a delay of only 10%, and the percentage delay can be changed by varying the delay in the clock.

The power consumed by the new model developed in paper [6] from their results has 10% decrease, the new model using BEC have an advantage of power compared to model using RCA. In my case the power for the new model has increased by 0.6% and this is due to the used in my design.

**5.4. Graph Comparing the Power, Area and Timing of both Models**

**5.4.1. graph comparing the area of SQRT CSLA and MSQRT CSLA.**

Figure 5.12: Graph comparing the area analysis

The above graph is the area analysis of SQRT CSLA and MSQRT CSLA, it compares my results with that of the results from the papers [6, 10]. The model using dual RCA is described by the blue color bar graph and the model using BEC is described by the orange color bar graph. In my thesis the area utilized by the SQRT CSLA is 436.42um2 and by the MSQRT CSLA is 354.20um2.

Figure 5.13: Graph comparing the percentage difference in area

The new model in my design has a percentage decrease of **18.8** compared to the old model. From paper [6] the area of MSQRT CSLA also has a percentage decrease of 15% (area got decreased from 2272um2 to 1929um2).The results from paper [10] show that they have a percentage decrease of 22.5% in their new model (area got decreased from 434um2 to 337um2).

**5.4.2. graph comparing the power of SQRT CSLA and MSQRT CSLA.**

Figure 5.14: Graph comparing the power analysis

Figure 5.15: Graph comparing the percentage difference of power

In the above graphs from my results the power consumed by SQRT CSLA is 19.99uw and MSQRT CSLA is 20.12uw, which leaves the new model with a 0.6% increase in its power. Comparing the results with the results of paper 6, they have a percentage decrease of 10% for the new model. The circuit which is designed in the paper [6] uses a 1.8V and uses 0.18um TSMC process and is simulated in a different tool using the netlist file so they have variations in power consumed by their circuit.

**5.4.3. graph comparing the timing of SQRT CSLA and MSQRT CSLA.**

The delay parameter for the MSQRT CSLA is calculated from the design compiler and is little bit high than that of the SQRT CSLA, this is because for the ordinary SQRT CSLA the delay is calculated only for the RCA’s but for the modified model it depends on both the RCA with Cin bit 0 and the BEC the BEC here has to wait for the sum to be calculated by the RCA. The sum from the RCA is used as input by the BEC to calculate the new sum so they doesn’t operate in parallel like for the SQRT CSLA case so this makes the data arrival time for the new model to increase.

Figure 5.16: Graph comparing the timing analysis

Figure 5.17: Graph comparing the percentage difference of delay

The MSQRT CSLA has a percentage increase of **10**.The timing results from paper [6] also can be clearly seen in the bar graph and they have a percentage decrease of 9.8 which clearly depicts that their model using BEC also takes more time than their model using RCA.

**6. Conclusion and Future Work**

**6.1. Conclusion**

In this thesis I have used a SQRT CSLA in which one model uses Dual RCA’s and other model uses a BEC replacing one of the RCA’s. The schematic designs for these models are developed using verilog code and synposys tool, this is a software implementation. The new model reduces the number of logic gates used in its design and thus this lets the overall design to consume less area compared to the original model. My aim in this thesis is to optimize area parameter but comparison between area, power and delay has been done for both the models to check the behavior of the models and the Modified model has a percentage decrease (18.8%) in area and there is a percentage increase (0.6%) in power which is little more compared to old model. There is a little bit more delay (10%) for the critical path in the modified model compared to the ordinary model. As the word size (number of bits) increases we can see more variation in the three parameters i.e. area, power and delay. Since the BEC is a good alternative which can be used in a SQRT CSLA and effectively utilizes area efficiently with considered amount of delay it can be used in area efficient applications. The modified SQRT CSLA is therefore a better model for VLSI hardware implementations in terms of area.

**6.2. Future Work**

The work done in this thesis has been designed for 16 bit SQRT CSLA and can be further extended for higher number of bits by cascading the 16 bit structure. New architectures can be designed in order to optimize all the parameters i.e. Power, delay and timing. Any design that can optimize all the three parameters is considered as a very good hardware design but based upon the demand of application some compromise has to be made between the parameters. A hardware design also requires high speed and low power in the functioning to sustain long enough in the market it will be a very good model if the three requirements together are met.

**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] Y. He, C. H. Chang, and J. Gu, “An area efficient 64-bit square root carry-select adder for low power applications,” in *Proc. IEEE Int. Symp.* *Circuits Syst.*, vol. 4, pp. 4082 –4085, 2005.

[6] B. Ramkumar and H. M. Kittur, “Low-power and area-efficient carry select adder” [Very Large Scale Integration (VLSI) Systems, *IEEE Transactions* on](http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=92) (Volume: 20, [Issue: 2](http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=6129939)), Feb. 2012.

[7] Y. Kim and L. S. Kim, “A low power carry select adder with reduced area” *Circuits and Systems*, Volume: 4, 6-9 May 2001.

[8] L. Mugilvannan and S. Ramasamy, “Lowpower and area efficient carry select adder using modified BEC-1converter”, *Communications and Networking Technologies (ICCCNT),* Tiruchengode, India,2013.

[9] K. Rawat, T. Darwish and M. Bayoumi, “A low power and reduced area Carry Select Adder”, *45th Midwest Symposium on Circuits and* *Systems*, vol.1, pp. 467- 470, March 2002.

[10] R. Priya and J.S. Kumar “Implementation and comparison of effective area efficient architectures for CSLA” Tirunelveli, India, 25-26 March 2013.

**Appendix A**

Verilog Code for 16 bit SQRT CSLA module

module CSLA\_16bit(

input [15:0] A,

input [15:0] B,

input Cin,

output [15:0] Sum,

output Cout

);

wire c1;

wire c2\_0,c2\_1;

wire c2;

wire c3\_0,c3\_1;

wire c3;

wire c4\_0,c4\_1;

wire c4;

wire c5\_0,c5\_1;

wire [15:2]Sum0;

wire [15:2]Sum1;

// 2 bit ripple carry adder

RCA\_2bit RCA\_1\_0 (

.Cin(Cin),

.A(A[1:0]),

.B(B[1:0]),

.Sum(Sum[1:0]),

.Cout(c1)

);

RCA\_2bit RCA\_3\_2\_C0 (

.Cin(1'b0),

.A(A[3:2]),

.B(B[3:2]),

.Sum(Sum0[3:2]),

.Cout(c2\_0)

);

RCA\_2bit RCA\_3\_2\_C1 (

.Cin(1'b1),

.A(A[3:2]),

.B(B[3:2]),

.Sum(Sum1[3:2]),

.Cout(c2\_1)

);

// carry multiplexer

// c2 is equal to c2\_0 when c1=0 else it is equal to c2\_1

assign c2=(c1==1'b0)?c2\_0:c2\_1;

// output multiplexer

/\* Sum[3:2]=sum0[3:2] when c1=0 else it is equal to Sum1[3:2] \*/

assign Sum[3:2]=(c1==1'b0)?Sum0[3:2]:Sum1[3:2];

RCA\_3bit RCA\_6\_4\_C0 (

.Cin(1'b0),

.A(A[6:4]),

.B(B[6:4]),

.Sum(Sum0[6:4]),

.Cout(c3\_0)

);

RCA\_3bit RCA\_6\_4\_C1 (

.Cin(1'b1),

.A(A[6:4]),

.B(B[6:4]),

.Sum(Sum1[6:4]),

.Cout(c3\_1)

);

assign c3=(c2==1'b0)?c3\_0:c3\_1;

assign Sum[6:4]=(c2==1'b0)?Sum0[6:4]:Sum1[6:4];

RCA\_4bit RCA\_10\_7\_C0 (

.Cin(1'b0),

.A(A[10:7]),

.B(B[10:7]),

.Sum(Sum0[10:7]),

.Cout(c4\_0)

);

RCA\_4bit RCA\_10\_7\_C1 (

.Cin(1'b1),

.A(A[10:7]),

.B(B[10:7]),

.Sum(Sum1[10:7]),

.Cout(c4\_1)

);

assign c4=(c3==1'b0)?c4\_0:c4\_1;

assign Sum[10:7]=(c3==1'b0)?Sum0[10:7]:Sum1[10:7];

RCA\_5bit RCA\_15\_11\_C0 (

.Cin(1'b0),

.A(A[15:11]),

.B(B[15:11]),

.Sum(Sum0[15:11]),

.Cout(c5\_0)

);

RCA\_5bit RCA\_15\_11\_C1 (

.Cin(1'b1),

.A(A[15:11]),

.B(B[15:11]),

.Sum(Sum1[15:11]),

.Cout(c5\_1)

);

assign Cout=(c4==1'b0)?c5\_0:c5\_1;

assign Sum[15:11]=(c4==1'b0)?Sum0[15:11]:Sum1[15:11];

endmodule

module RCA\_2bit(

input Cin,

input [1:0] A,

input [1:0] B,

output [1:0] Sum,

output Cout

);

wire Carry;

FullAdder Adder1 (

.s(Sum[0]),

.co(Carry),

.a(A[0]),

.b(B[0]),

.ci(Cin)

);

FullAdder Adder2 (

.s(Sum[1]),

.co(Cout),

.a(A[1]),

.b(B[1]),

.ci(Carry)

);

endmodule

module RCA\_3bit(

input Cin,

input [2:0] A,

input [2:0] B,

output [2:0] Sum,

output Cout

);

wire[1:0]Carry;

FullAdder Adder1 (

.s(Sum[0]),

.co(Carry[0]),

.a(A[0]),

.b(B[0]),

.ci(Cin)

);

FullAdder Adder2 (

.s(Sum[1]),

.co(Carry[1]),

.a(A[1]),

.b(B[1]),

.ci(Carry[0])

);

FullAdder Adder3 (

.s(Sum[2]),

.co(Cout),

.a(A[2]),

.b(B[2]),

.ci(Carry[1])

);

endmodule

module RCA\_4bit(

input Cin,

input [3:0] A,

input [3:0] B,

output [3:0] Sum,

output Cout

);

wire[2:0]Carry;

FullAdder Adder1 (

.s(Sum[0]),

.co(Carry[0]),

.a(A[0]),

.b(B[0]),

.ci(Cin)

);

FullAdder Adder2 (

.s(Sum[1]),

.co(Carry[1]),

.a(A[1]),

.b(B[1]),

.ci(Carry[0])

);

FullAdder Adder3 (

.s(Sum[2]),

.co(Carry[2]),

.a(A[2]),

.b(B[2]),

.ci(Carry[1])

);

FullAdder Adder4 (

.s(Sum[3]),

.co(Cout),

.a(A[3]),

.b(B[3]),

.ci(Carry[2])

);

endmodule

module RCA\_5bit(

input Cin,

input [4:0] A,

input [4:0] B,

output [4:0] Sum,

output Cout

);

wire[3:0]Carry;

FullAdder Adder1 (

.s(Sum[0]),

.co(Carry[0]),

.a(A[0]),

.b(B[0]),

.ci(Cin)

);

FullAdder Adder2 (

.s(Sum[1]),

.co(Carry[1]),

.a(A[1]),

.b(B[1]),

.ci(Carry[0])

);

FullAdder Adder3 (

.s(Sum[2]),

.co(Carry[2]),

.a(A[2]),

.b(B[2]),

.ci(Carry[1])

);

FullAdder Adder4 (

.s(Sum[3]),

.co(Carry[3]),

.a(A[3]),

.b(B[3]),

.ci(Carry[2])

);

FullAdder Adder5 (

.s(Sum[4]),

.co(Cout),

.a(A[4]),

.b(B[4]),

.ci(Carry[3])

);

endmodule

module FullAdder(s,co,a,b,ci);

output s,co;

input a,b,ci;

xor u1(s,a,b,ci);

and u2(n1,a,b);

and u3(n2,b,ci);

and u4(n3,a,ci);

or u5(co,n1,n2,n3);

endmodule

**Appendix B**

Verilog Code for 16 bit MSQRT CSLA module

module MCSLA\_16bit(

input [15:0] A,

input [15:0] B,

input Cin,

output [15:0] Sum,

output Cout

);

wire c1;

wire c2\_0,c2\_1;

wire c2;

wire c3\_0,c3\_1;

wire c3;

wire c4\_0,c4\_1;

wire c4;

wire c5\_0,c5\_1;

wire [15:2]Sum0;

wire [15:2]Sum1;

RCA\_2bit RCA\_1\_0 (

.Cin(Cin),

.A(A[1:0]),

.B(B[1:0]),

.Sum(Sum[1:0]),

.Cout(c1)

);

RCA\_2bit RCA\_3\_2\_C0 (

.Cin(1'b0),

.A(A[3:2]),

.B(B[3:2]),

.Sum(Sum0[3:2]),

.Cout(c2\_0)

);

/\*B is given as input for the BEC and c2\_0 and sum0 are assigned to it\*/

/\*X is the sum form BEC and Cout is carry of the BEC\*/

BEC\_3bit BEC\_3bit (

.B({c2\_0,Sum0[3:2]}),

.X(Sum1[3:2]),

.Cout(c2\_1)

);

/\* c2 is equal to c2\_0 when c1=0 else it is equal to c2\_1 \*/

assign c2=(c1==1'b0)?c2\_0:c2\_1;

/\* Sum[3:2]=sum0[3:2] when c1=0 else it is equal to Sum1[3:2] \*/

assign Sum[3:2]=(c1==1'b0)?Sum0[3:2]:Sum1[3:2];

RCA\_3bit RCA\_6\_4\_C0 (

.Cin(1'b0),

.A(A[6:4]),

.B(B[6:4]),

.Sum(Sum0[6:4]),

.Cout(c3\_0)

);

BEC\_4bit BEC\_4bit (

.B({c3\_0,Sum0[6:4]}),

.X(Sum1[6:4]),

.Cout(c3\_1)

);

assign c3=(c2==1'b0)?c3\_0:c3\_1;

assign Sum[6:4]=(c2==1'b0)?Sum0[6:4]:Sum1[6:4];

RCA\_4bit RCA\_10\_7\_C0 (

.Cin(1'b0),

.A(A[10:7]),

.B(B[10:7]),

.Sum(Sum0[10:7]),

.Cout(c4\_0)

);

BEC\_5bit BEC\_5bit (

.B({c4\_0,Sum0[10:7]}),

.X(Sum1[10:7]),

.Cout(c4\_1)

);

assign c4=(c3==1'b0)?c4\_0:c4\_1;

assign Sum[10:7]=(c3==1'b0)?Sum0[10:7]:Sum1[10:7];

RCA\_5bit RCA\_15\_11\_C0 (

.Cin(1'b0),

.A(A[15:11]),

.B(B[15:11]),

.Sum(Sum0[15:11]),

.Cout(c5\_0)

);

BEC\_6bit BEC\_6bit (

.B({c5\_0,Sum0[15:11]}),

.X(Sum1[15:11]),

.Cout(c5\_1)

);

assign Cout=(c4==1'b0)?c5\_0:c5\_1;

assign Sum[15:11]=(c4==1'b0)?Sum0[15:11]:Sum1[15:11];

endmodule

module RCA\_2bit(

input Cin,

input [1:0] A,

input [1:0] B,

output [1:0] Sum,

output Cout

);

wire Carry;

FullAdder Adder1 (

.s(Sum[0]),

.co(Carry),

.a(A[0]),

.b(B[0]),

.ci(Cin)

);

FullAdder Adder2 (

.s(Sum[1]),

.co(Cout),

.a(A[1]),

.b(B[1]),

.ci(Carry)

);

endmodule

module BEC\_3bit(

input [2:0] B,

output [1:0] X,

output Cout

);

assign X[0]=~B[0]; // invert of the first input bit

assign X[1]=B[0]^B[1];// the OR of the 2 last input bits

assign Cout=B[2]^(B[0]&B[1]);// the AND of the 2 LSB ORED with the MSB //

endmodule

module RCA\_3bit(

input Cin,

input [2:0] A,

input [2:0] B,

output [2:0] Sum,

output Cout

);

wire[1:0]Carry;

FullAdder Adder1 (

.s(Sum[0]),

.co(Carry[0]),

.a(A[0]),

.b(B[0]),

.ci(Cin)

);

FullAdder Adder2 (

.s(Sum[1]),

.co(Carry[1]),

.a(A[1]),

.b(B[1]),

.ci(Carry[0])

);

FullAdder Adder3 (

.s(Sum[2]),

.co(Cout),

.a(A[2]),

.b(B[2]),

.ci(Carry[1])

);

endmodule

module BEC\_4bit(

input [3:0] B,

output [2:0] X,

output Cout

);

assign X[0]=~B[0];

assign X[1]=B[0]^B[1];

assign X[2]=B[2]^(B[0]&B[1]);

assign Cout=B[3]^(B[0]&B[1]&B[2]);

endmodule

module RCA\_4bit(

input Cin,

input [3:0] A,

input [3:0] B,

output [3:0] Sum,

output Cout

);

wire[2:0]Carry;

FullAdder Adder1 (

.s(Sum[0]),

.co(Carry[0]),

.a(A[0]),

.b(B[0]),

.ci(Cin)

);

FullAdder Adder2 (

.s(Sum[1]),

.co(Carry[1]),

.a(A[1]),

.b(B[1]),

.ci(Carry[0])

);

FullAdder Adder3 (

.s(Sum[2]),

.co(Carry[2]),

.a(A[2]),

.b(B[2]),

.ci(Carry[1])

);

FullAdder Adder4 (

.s(Sum[3]),

.co(Cout),

.a(A[3]),

.b(B[3]),

.ci(Carry[2])

);

endmodule

module BEC\_5bit(

input [4:0] B,

output [3:0] X,

output Cout

);

assign X[0]=~B[0];

assign X[1]=B[0]^B[1];

assign X[2]=B[2]^(B[0]&B[1]);

assign X[3]=B[3]^(B[0]&B[1]&B[2]);

assign Cout=B[4]^(B[0]&B[1]&B[2]&B[3]);

endmodule

module RCA\_5bit(

input Cin,

input [4:0] A,

input [4:0] B,

output [4:0] Sum,

output Cout

);

wire[3:0]Carry;

FullAdder Adder1 (

.s(Sum[0]),

.co(Carry[0]),

.a(A[0]),

.b(B[0]),

.ci(Cin)

);

FullAdder Adder2 (

.s(Sum[1]),

.co(Carry[1]),

.a(A[1]),

.b(B[1]),

.ci(Carry[0])

);

FullAdder Adder3 (

.s(Sum[2]),

.co(Carry[2]),

.a(A[2]),

.b(B[2]),

.ci(Carry[1])

);

FullAdder Adder4 (

.s(Sum[3]),

.co(Carry[3]),

.a(A[3]),

.b(B[3]),

.ci(Carry[2])

);

FullAdder Adder5 (

.s(Sum[4]),

.co(Cout),

.a(A[4]),

.b(B[4]),

.ci(Carry[3])

);

endmodule

module BEC\_6bit(

input [5:0] B,

output [4:0] X,

output Cout

);

assign X[0]=~B[0];

assign X[1]=B[0]^B[1];

assign X[2]=B[2]^(B[0]&B[1]);

assign X[3]=B[3]^(B[0]&B[1]&B[2]);

assign X[4]=B[4]^(B[0]&B[1]&B[2]&B[3]);

assign Cout=B[5]^(B[0]&B[1]&B[2]&B[3]&B[4]);

endmodule

module FullAdder(s,co,a,b,ci);

output s,co;

input a,b,ci;

xor u1(s,a,b,ci);

and u2(n1,a,b);

and u3(n2,b,ci);

and u4(n3,a,ci);

or u5(co,n1,n2,n3);

endmodule