RESEARCH ARTICLE OPEN ACCESS

High Speed and Area Efficient Booth Multiplier Using SQRT CSLA with Zero Finding Logic

P.Pavani Sushma[1], J.Priyanka[2],R.Lalitha[3],K.Manoj[4],N.Divya[5],V.Suma[6]
Department Of Electronics And Communication Engineering,Liet,Jonnada,Jntuk,Vizianagaram

ABSTRACT
Addition is one of the common and widely used fundamental arithmetic operation in many VLSI systems. The critical elements in general purpose and digital-signal processing processors are High performance VLSI integer adders as they are employed in the design of Arithmetic-Logic Units, in floating-point arithmetic data paths and in address generation units. The performance parameters for any adder are area, speed and delay. By using Square Root Carry Select Adder (SQRT CSLA), speed can be achieved. In designing new architecture, the Tradeoff between those parameters plays the major role. We can reduce area by using Zero Finding Logic (ZFC) technique, from the structure of SQRT CSLA. By using the Modified architecture we can reduce area.
We can implement Booth multiplier by using the CSLA and SQRT CSLA with Zero finding logic. Implementation of Booth multiplier by using CSLA and SQRT CSLA with Zero finding logic is proposed for better speed applications and efficient area applications.

Keywords: And Or Inverter (AOI); Binary to Excess-one Converter (BEC); Carry Select Adder(CSLA);FPGA (FieldProgrammable Gate Array); Half Adder (HA); Look up Table (LUT); Ripple Carry Adder(RCA); Square Root Carry Select Adder(SQRT CSLA); Very Large Scale Integrated Circuits(VLSI); Zero Finding Logic (ZFC).

I. INTRODUCTION

In Advanced digital processors, Low Power, Area efficient and high performance VLSI designs plays an important role . In rapidly growing mobile industry, for design of digital circuits, not only faster units are concerned but also smaller area and less power become major concerns. Reducing area and power consumption are key factors in mobile electronics for increasing portability and battery life. power dissipation is an important design constraint even in servers and desktop computers. The heart of computer arithmetic is Addition and the work horse of a computational circuit is arithmetic unit. They are the necessary components of a data path, e.g. in microprocessors or a signal processor. An adder can be designed in many ways. The Ripple delay is linearly proportional to N, if there is N-bit RCA. Thus the RCA gives highest delay of all adders for large values of N. The Carry Look-Ahead Adder (CLA) gives fast results but it consumes large area. Speed of addition is limited by carry in digital adders that plays a major role in computations. Carry select adder (CSLA) is used in many computational systems as it is one of the fastest adders to improve the carry propagation delay by independently generating multiple carries and then it selects a carry to generate the sum. But the area of CSLA increases with the use of dual RCAs. A CSLA with Binary to Excess-1 Converter (BEC) is designed. It is used to reduce the area of the CSLA, but the delay overhead is increasing. An add one circuit can be used to design a CSLA to further decrease the area and to keep the delay constant or equal as basic CSLA.

It proposes the design of 8-bit, 16-bit, 32-bit, 64-bit and 128-bit square root CSLA (SQRT CSLA) using add one circuit with significant reduction in area. Using add one circuit, the performance in terms of area and delay are evaluated for SQRT CSLA and are compared with the existing SQRT CSLA and SQRT CSLA using Binary to Execess-1 Converter (BEC). Using Verilog, the proposed design is developed and by using ISIM simulator functional simulation is performed.

By using square root carry select adder using Zero Finding Logic, we can reduce area and delay and increase the speed.
Implementation of ALU by using modified Square Root Carry Select Adder(SQRTCSLA) is proposed for Low power and area-efficient applications and for better speed applications. The paper delivers the design and implementation of 8-Bit ALU by using modified SQRT CSLA and also compares it with the design of SQRT CSLA using Zero finding logic in terms of total number of basic gates. The design entry is done by using Verilog Hardware Description Language (HDL) and simulated by using ISIM Simulator. By using Xilinx ISE 12.1, it is synthesized and implemented.

II. CARRY SELECT ADDER:

The ripple carry adder consists of many single bit full-adders in cascaded form. The circuit is simple and area-efficient architecture. But, each full-adder can start operation only when the previous carry-out signal is ready . So, the computation speed is slow. N bits adder is divided into M parts in the carry select adder. Here, Each part of adder consists of two ripple carry adders with Cin=0 and Cin=1, respectively as shown in fig1. By using the multiplexer, the correct output result according to the logic state of carry-in signal can be selected In carry select adder, the current adder stage does not need to wait the previous stage's carry-out signal. So, the carry-select adder can compute faster. Before the arrival of carry-in signal, the summation result is ready. So, the correct computation result can be obtained only by waiting for one multiplexer delay in each single bit adder. The carry propagation delay can be reduced by M times in the carry select adder, when compared with the carry ripple adder. When compared with other adders, the carry select adder is faster and intermediate.

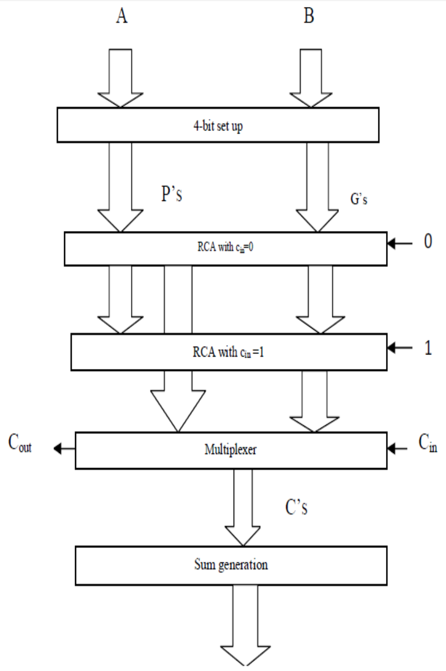


Fig.1 Four-bit carry-select module

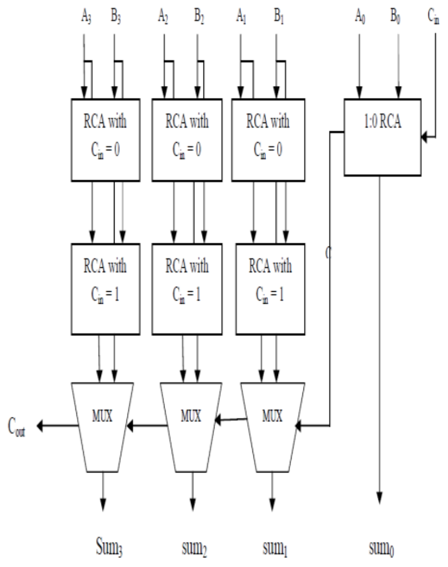


Fig 2. 4- Bit carry select adder

III. LINEAR CARRY SELECT ADDER:

By chaining a number of equal length adder stages, a linear carry select adder can be constructed as shown in fig 2. We can derive the propagation delay of the module in worst case.

The propagation delay of the adder is linearly proportional to N. The block select signal that selects between the 0 and 1solution still has to ripple through all stages in the worst case. This is the reason for the linear behavior.

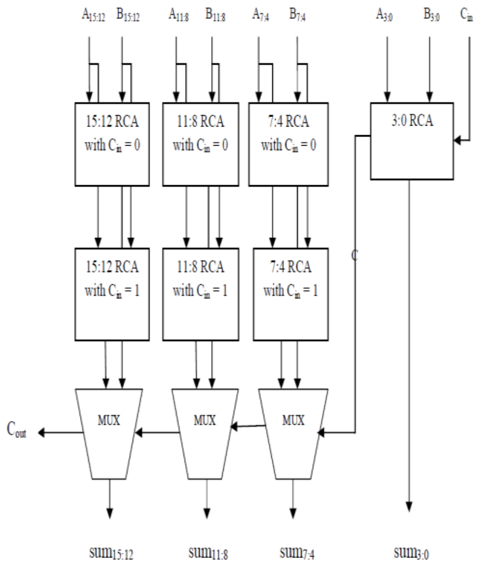
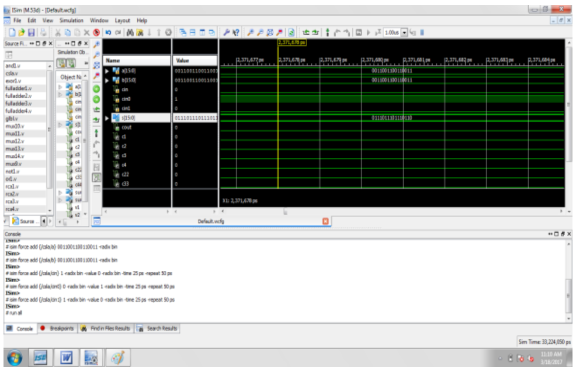
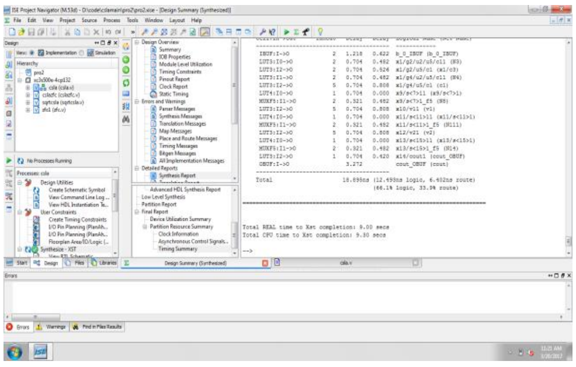


Fig 3. 16-Bit linear carry select adder



```
ISim>
# isim force add {/csla/a} 0011001100110011 -radix bin
ISim>
# isim force add {/csla/b} 0011001100110011 -radix bin
ISim>
# isim force add {/csla/cin} 1 -radix bin -value 0 -radix bin -time 25 ps -repeat 50 ps
ISim>
# isim force add {/csla/cin0} 0 -radix bin -value 1 -radix bin -time 25 ps -repeat 50 ps
ISim>
# isim force add {/csla/cin1} 1 -radix bin -value 0 -radix bin -time 25 ps -repeat 50 ps
ISim>
# run all
```



```
    IBUF:I->O              2   1.218   0.622  b_0_IBUF (b_0_IBUF)
    LUT3:I0->O             2   0.704   0.482  x1/g2/u2/u5/c11 (N3)
    LUT3:I2->O             2   0.704   0.526  x1/g2/u5/c1 (x1/c3)
    LUT3:I1->O             2   0.704   0.482  x1/g4/u2/u5/c11 (N4)
    LUT3:I2->O             5   0.704   0.808  x1/g4/u5/c1 (c1)
    LUT4:I0->O             1   0.704   0.000  x9/s<7>11 (x9/s<7>1)
    MUXF5:I1->O            2   0.321   0.482  x9/s<7>1_f5 (N8)
    LUT3:I2->O             5   0.704   0.808  x10/v11 (v1)
    LUT4:I0->O             1   0.704   0.000  x11/s<11>11 (x11/s<11>1)
    MUXF5:I1->O            2   0.321   0.482  x11/s<11>1_f5 (N111)
    LUT3:I2->O             5   0.704   0.808  x12/v21 (v2)
    LUT4:I0->O             1   0.704   0.000  x13/s<15>11 (x13/s<15>1)
    MUXF5:I1->O            2   0.321   0.482  x13/s<15>1_f5 (N14)
    LUT3:I2->O             1   0.704   0.420  x14/cout1 (cout_OBUF)
    OBUF:I->O                  3.272          cout_OBUF (cout)
   ----------------------------------------
   Total                      18.895ns (12.493ns logic, 6.402ns route)
                                       (66.1% logic, 33.9% route)

=========================================================================


Total REAL time to Xst completion: 9.00 secs
Total CPU time to Xst completion: 9.30 secs

-->
```

IV. SQRT CARRY SELECT ADDER:

The structure of the 16-bit regular SQRT CSLA is shown in Fig. 3. It has five groups which consists of different sizes of RCA. it is essential to locate the critical timing path first, in order to optimize a design. If we Consider the case of a 16-bit linear carry-select adder, assume that the full-adder and multiplexer cells have identical propagation delays equalto a normalized value of 1 in order to simplify the discussion the critical path of the adder ripples through the multiplexer networks of the subsequent stages can be determined by this analysis.

In the last adder stage, consider the multiplexer gate. The two carry chains of the block and the block-multiplexer signal from the previous stage are the inputs to this multiplexer. between the arrival times of the signals, a major mismatch can be observed. Long before the multiplexer signal arrives, the results of the carry chain are stable. The delay through both paths must be equilized.

By adding more bits to the subsequent stage in the adder, this can be achieved by progressively. It requires more time for the generation of the carry signals. For example ,the first stage can add 2 bits, the second has 3, the third contains 4, and so forth. This type of adder is called square root carry select adder.

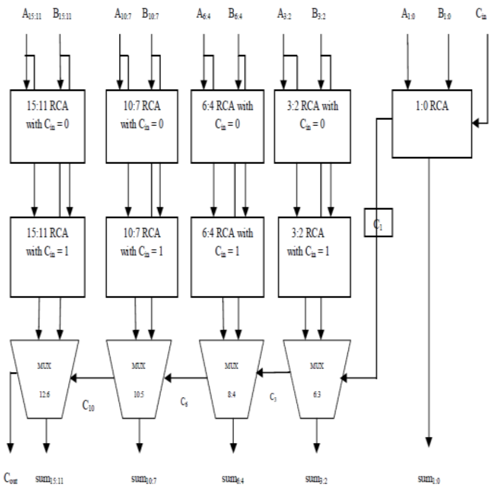
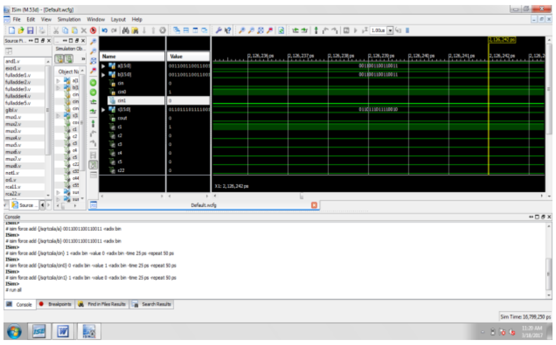
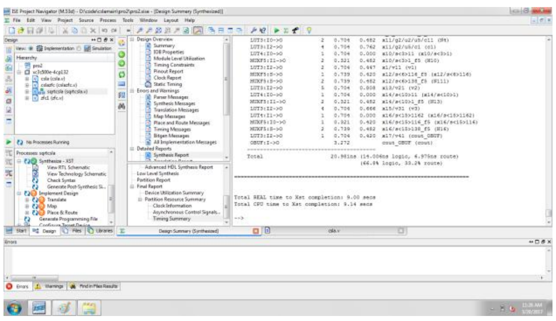


Fig 4. 16 - Bit square root carry select adder



```
ISim>
# isim force add {/sqrtcsla/a} 0011001100110011 -radix bin
ISim>
# isim force add {/sqrtcsla/b} 0011001100110011 -radix bin
ISim>
# isim force add {/sqrtcsla/cin} 1 -radix bin -value 0 -radix bin -time 25 ps -repeat 50 ps
ISim>
# isim force add {/sqrtcsla/cin0} 0 -radix bin -value 1 -radix bin -time 25 ps -repeat 50 ps
ISim>
# isim force add {/sqrtcsla/cin1} 1 -radix bin -value 0 -radix bin -time 25 ps -repeat 50 ps
ISim>
# run all
```



```
    LUT3:I0->O             2   0.704   0.482  x11/g2/u2/u5/c11 (N4)
    LUT3:I2->O             4   0.704   0.762  x11/g2/u5/c1 (c1)
    LUT4:I0->O             1   0.704   0.000  x10/s<3>11 (x10/s<3>1)
    MUXF5:I1->O            2   0.321   0.482  x10/s<3>1_f5 (N10)
    LUT3:I2->O             2   0.704   0.447  x1/v11 (v1)
    MUXF5:S->O             1   0.739   0.420  x12/s<6>116_f5 (x12/s<6>116)
    MUXF5:S->O             2   0.739   0.482  x12/s<6>138_f5 (N111)
    LUT3:I2->O             5   0.704   0.808  x13/v21 (v2)
    LUT4:I0->O             1   0.704   0.000  x14/s<10>11 (x14/s<10>1)
    MUXF5:I1->O            2   0.321   0.482  x14/s<10>1_f5 (N13)
    LUT3:I2->O             4   0.704   0.666  x15/v31 (v3)
    LUT4:I1->O             1   0.704   0.000  x16/s<15>1162 (x16/s<15>1162)
    MUXF5:I0->O            1   0.321   0.420  x16/s<15>116_f5 (x16/s<15>116)
    MUXF5:S->O             2   0.739   0.482  x16/s<15>138_f5 (N16)
    LUT3:I2->O             1   0.704   0.420  x17/v41 (cout_OBUF)
    OBUF:I->O                  3.272          cout_OBUF (cout)
   ----------------------------------------
   Total                      20.981ns (14.006ns logic, 6.976ns route)
                                       (66.8% logic, 33.2% route)

=========================================================================


Total REAL time to Xst completion: 9.00 secs
Total CPU time to Xst completion: 9.14 secs

-->
```

V. SQRT CSLA USING ZERO FINDING LOGIC

Instead of RCA with Cin=l, this adder uses add one circuit. if the results for Cin = 0 is known the result for Cin=l can be found by adding one to the result for Cin=0. This is the main principle used in this adder. Thus, the ripple carry adder for Cin=l in a block can be replaced by an Add one circuit. The area of SQRT CSLA can be further reduced with an efficient design of an add one circuit for designing add one circuit, Complement scheme is used. The Complement scheme states that by adding one is just inverting each So bit starting from the least significant bit until the first zero is found. using an add one circuit instead of a RCA with Cin=1, the 16-bit SQRT CSLA is designed as shown in fig. 4

The complex logics can be alternatively implemented usingCPL (complementary pass transistor logic). CPL is extremely fast and efficient. Some of the examples of CPL circuits are shown in figure 5.

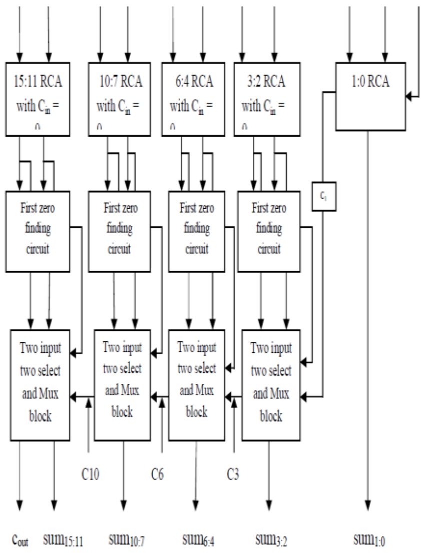
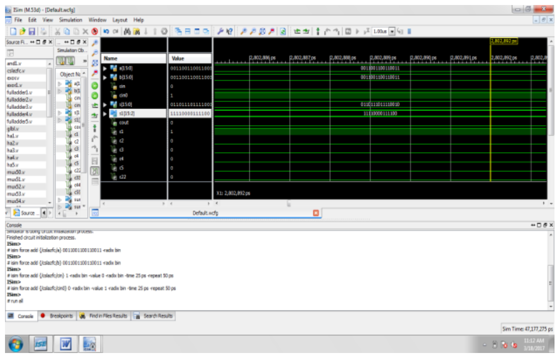
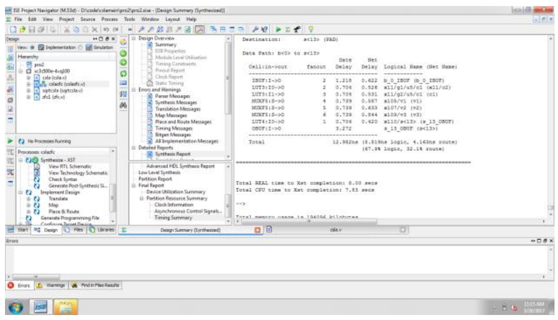

15:11 RCA with C_{in} = 0
10:7 RCA with C_{in} = 0
6:4 RCA with C_{in} = 0
3:2 RCA with C_{in} = 0
1:0 RCA
C_1
First zero finding circuit
First zero finding circuit
First zero finding circuit
First zero finding circuit
Two input two select and Mux block
Two input two select and Mux block
Two input two select and Mux block
Two input two select and Mux block
C10
C6
C3
Cout
sum15:11
sum10:7
sum6:4
sum3:2
sum1:0

Fig5.16-Bit SQRT CSLA with ZFC circuit.



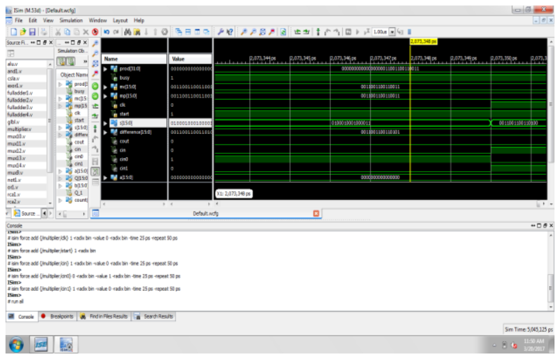


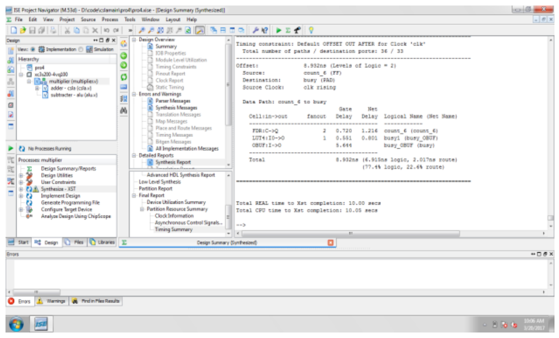
```
Destination:        s<13> (PAD)

Data Path: b<0> to s<13>
                                 Gate     Net
  Cell:in->out      fanout     Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O             2   1.218   0.622  b_0_IBUF (b_0_IBUF)
   LUT3:I0->O            2   0.704   0.526  x11/g1/u5/c1 (x11/c2)
   LUT3:I1->O            3   0.704   0.531  x11/g2/u5/c1 (c1)
   MUXF5:S->O            4   0.739   0.587  x105/v1 (v1)
   MUXF5:S->O            5   0.739   0.633  x107/v2 (v2)
   MUXF5:S->O            6   0.739   0.844  x109/v3 (v3)
   LUT4:I0->O            1   0.704   0.420  x110/s<13> (s_13_OBUF)
   OBUF:I->O                 3.272          s_13_OBUF (s<13>)
  ----------------------------------------
  Total                     12.982ns (8.819ns logic, 4.163ns route)
                                     (67.9% logic, 32.1% route)

=========================================================================


Total REAL time to Xst completion: 8.00 secs
Total CPU time to Xst completion: 7.83 secs

-->

Total memory usage is 194096 kilobytes
```

Here, in this architecture, it consists of five carry select stages (CSS). only adders are present in first CSS and the remaining stages consist of adders, add one circuit, first zero finding circuit and multiplexers. Mirror adders are the adders that are used in the construction of RCAs. Until it doesn't find zero in the input number, the first zero finding circuit generates 0. After finding the 0 in the input number it generates 1. Nmos and Pmos chain is the first zero finding circuit. A multiplexer based on add one circuit is proposed. In order to choose in between sum and complement of sum, a multiplexer is needed for each bit.

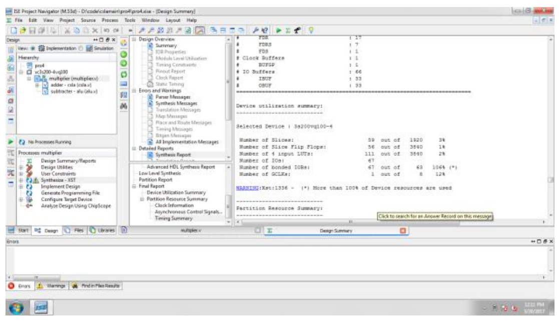
VI. BOOTH'S MULTIPLIER USING CSLA:

Booth's multiplier algorithm is impelemented by using carry select adder. Here , the area and delay can be reduced.





```
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 36 / 33
-------------------------------------------------------------------------
Offset:              8.932ns (Levels of Logic = 2)
  Source:            count_6 (FF)
  Destination:       busy (PAD)
  Source Clock:      clk rising

  Data Path: count_6 to busy
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDR:C->Q              2   0.720   1.216  count_6 (count_6)
     LUT4:I0->O            1   0.551   0.801  busy1 (busy_OBUF)
     OBUF:I->O                 5.644          busy_OBUF (busy)
    ----------------------------------------
    Total                      8.932ns (6.915ns logic, 2.017ns route)
                                       (77.4% logic, 22.6% route)

=========================================================================


Total REAL time to Xst completion: 10.00 secs
Total CPU time to Xst completion: 10.05 secs

-->
```



```
#      FDR                         : 17
#      FDRS                        : 7
#      FDS                         : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 66
#      IBUF                        : 33
#      OBUF                        : 33
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s200vq100-4

 Number of Slices:                       59  out of   1920     3%
 Number of Slice Flip Flops:             56  out of   3840     1%
 Number of 4 input LUTs:                111  out of   3840     2%
 Number of IOs:                          67
 Number of bonded IOBs:                  67  out of     63   106% (*)
 Number of GCLKs:                         1  out of      8    12%

WARNING:Xst:1336 -  (*) More than 100% of Device resources are used

---------------------------
Partition Resource Summary:
---------------------------
```

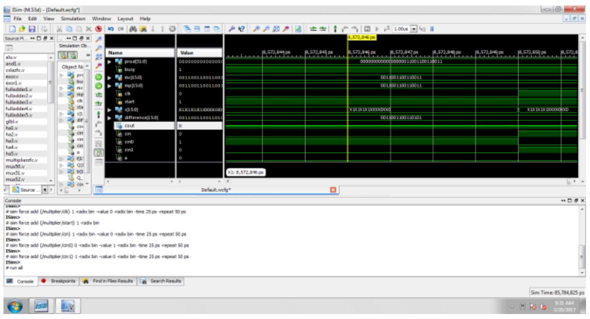
VII. BOOTH'S MULTIPLIER USING SQRT CSLA WITH ZERO FINDING LOGIC:

Booth's multiplication algorithm is multiplies two signed binary numbers in two's complement notation. It is a multiplication algorithm. by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values *A* and *S* to a product *P*, and then by performing a rightward arithmetic shift on *P*, Booth's algorithm can be implemented. Let us consider that **m** and **r** are the multiplicand and multiplier, respectively; and let the number of bits in **m** and **r** are represented by x and y respectively.

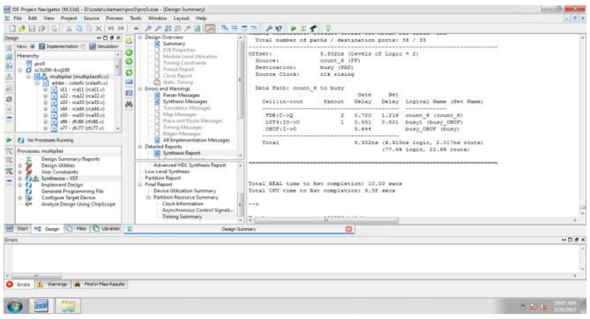
The rules to implement the Booth's multiplier are:

1. The initial value is represented by P. Determine the values of A and S. The length should be equal to $(x + y + 1)$ to all of these numbers.
   a. A: most significant (leftmost) bits must be filled with the value of **m**. remaining $(y + 1)$ bits must be filled with zeros.
   b. S: The most significant bits must be filled with the value of (−**m**) in the two's complement notation. The remaining $(y + 1)$ bits must be filled with zeros.
   c. P: The most significant bits of x must be filled with zeros. Append the value of **r** to the right of this value. The least significant (rightmost) bit must be filled with a zero.
2. Now, the two least significant (rightmost) bits of P must be determined.
   a. If the two least significant (rightmost) bits are 01, then we need to find the value of $P + A$. Any overflow can be Ignored.
   b. If the two least significant (rightmost) bits are 10, then we need to find the value of $P + S$. Any overflow can be Ignored.
   c. If the two least significant (rightmost) bits are 00, do not perform any addition or subtraction operation. We can use the value of P directly in the next step.
   d. If the two least significant (rightmost) bits are 11, do not perform any addition or subtraction operation. We can use the value of P directly in the next step.
3. Shift the value that was obtained in the 2nd step arithmetically to the right by a single place. Now, Let P be equal to this new value.
4. Repeat the steps of 2 and 3 until they have been done y times.
5. Now, the least significant (rightmost) bit from P must be Dropped. This is the product value of **m** and **r**.

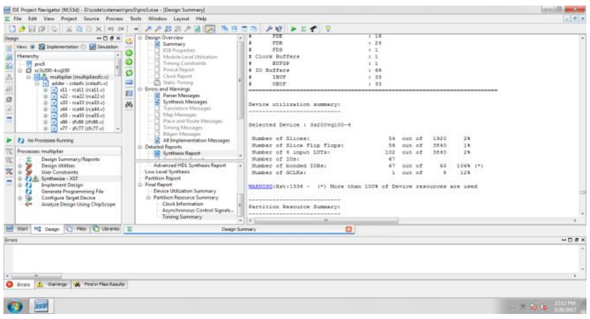
   Now, replace the Arithmetic logical unit adder with the SQRT CSLA Zero finding logic. By replacing the Arithmetic logical unit with SQRT CSLA Zero finding logic, we are going to reduce the number of gates used in the implementation of Booth's multiplier.



```
# isim force add {/multiplier/clk} 1 -radix bin -value 0 -radix bin -time 25 ps -repeat 50 ps
ISim>
# isim force add {/multiplier/start} 1 -radix bin
ISim>
# isim force add {/multiplier/cin} 1 -radix bin -value 0 -radix bin -time 25 ps -repeat 50 ps
ISim>
# isim force add {/multiplier/cin0} 0 -radix bin -value 1 -radix bin -time 25 ps -repeat 50 ps
ISim>
# isim force add {/multiplier/cin1} 1 -radix bin -value 0 -radix bin -time 25 ps -repeat 50 ps
ISim>
# run all
```



```
  Total number of paths / destination ports: 36 / 33
-------------------------------------------------------------------------
Offset:              8.932ns (Levels of Logic = 2)
  Source:            count_6 (FF)
  Destination:       busy (PAD)
  Source Clock:      clk rising

  Data Path: count_6 to busy
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDR:C->Q              2   0.720   1.216  count_6 (count_6)
     LUT4:I0->O            1   0.551   0.801  busy1 (busy_OBUF)
     OBUF:I->O                 5.644          busy_OBUF (busy)
    ----------------------------------------
    Total                      8.932ns (6.915ns logic, 2.017ns route)
                                       (77.4% logic, 22.6% route)

=========================================================================


Total REAL time to Xst completion: 10.00 secs
Total CPU time to Xst completion: 9.38 secs

-->
```



```
#      FDE                         : 16
#      FDR                         : 24
#      FDS                         : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 66
#      IBUF                        : 33
#      OBUF                        : 33
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s200vq100-4

 Number of Slices:                      54  out of   1920     2%
 Number of Slice Flip Flops:            56  out of   3840     1%
 Number of 4 input LUTs:               102  out of   3840     2%
 Number of IOs:                         67
 Number of bonded IOBs:                 67  out of     63   106% (*)
 Number of GCLKs:                        1  out of      8    12%

WARNING:Xst:1336 -  (*) More than 100% of Device resources are used

---------------------------
Partition Resource Summary:
---------------------------
```

VIII. CONCLUSION

By the implementation of Booth's Multiplier using Square root carry select adder with Zero Finding Logic, we can reduce the number of gates that are used in the design implementation of Booth's Multiplier. We can also reduce the area and delay as the number of gates decreases in Booth's Multiplier using Square root carry select adder with Zero Finding Logic when compared with Booth's Multiplier using carry select adder.

REFERENCES

[1]. *Chi-hau Chen (1992). Signal processing handbook. CRC Press. p. 234. ISBN 978-0-8247-7956-6.*

[2]. B.Ramkumar and Harish M Kittur, "Low-Power and Area-Efficient Carry Select Adder" IEEE transactions on very large scale integration (VLSI) systems, vol. 20, no.2, February 2012]

[3]. 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., 2005, vol.4, pp. 4082–4085.

[4]. 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.

[5]. 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.

[6]. O. J. Bedrij, "Carry-select adder," IRE Trans. Electron. Comput., pp.340–344, 1962.

[7]. 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).

[8]. U. Sajesh Kumar, K. Mohamed Salih and K. Sajith, Deisgn and Implementation of Carry Select Adder without using Mutilpexers,1*st International Conference on Emerging Technology in Electronics, Communication and Networking*, (2012).

[9]. K. Bala Sindhuri, K. Padma Vasavi, I. Santi Prabha and N. Udaya Kumar, VLSI Architecture for Linear Carry Select Adder with Zero Finding Logic, 6*th International Advanced Computing Conference IACC 2016*, pp. 31, February (2016).