Comparative Study of Multiplier On The Basis Of Power, Area and Speed

DEEPALI K.CHAVAN

International Institute of Information Technology Pune, Maharashtra
deepali.chavan84@gmail.com, deepalic_aug10@vlsi.isquareit.ac.in

Abstract: *Multiplications are expensive and slow operations. The performance of many computational problems often is dominated by the speed at which a multiplication operation can be executed. This observation has, for instance, prompted the integration of complete multiplication units in state of the art digital signal processors and microprocessors.*
For portable application where power consumption is most important parameter, one should reduce the power dissipations much as possible. The proposed algorithm for radix 2 multiplier is one the best algorithm for high speed and low power consumption multiplier.
Multipliers are, in effect, complex adder arrays. Therefore, the majority of the topics discussed in the preceding section are of value in this context as well. The analysis of the multiplier gives us some further insight in to how to optimise the performance (or the area)of complex circuit topologies. After a short discussion of the multiply operation, we discuss basic array multiplier. We also discuss different approaches to partial product generation, accumulation and their final summation.

Keywords : **low power ic design,**

I. INTRODUCTION

During the recent past, significant progress in the field of computational fluid dynamics (CFD) has also been made, and CFD is gradually becoming established as an efficient tool in vehicle design. It is recognized that complex flow fields are not easily represented in terms of a closed solution. CFD technology allows for the visualization of complex flow phenomena in a virtual environment that can significantly enhance the learning experience. It has the potential to explore cause-effect relationships through open-ended analyses, and extends analyses beyond what is possible using traditional experimentation, because the end user can easily visualize complex flow phenomena using color contour plots and velocity vector plots. Additional features available to compute many derived parameters along with user-friendly graphical operations allow highlighting the region of interest for detailed analyses.

II. THE MULTIPLIER

Multiplications are expensive and also operations. The performance of many computational problems often is dominated by the speed at which a multiplication operation can be executed. This observation has, for instance, prompted the integration of complete multiplication units in state of the art digital signal processor and microprocessors. Multipliers are, in effect, complex adder arrays.Therefor,the majority of the topics discussed in the preceding section are of value in this context as well. The analysis of the multiplier gives us some further insight in to how to optimise the performance (or the area) of complex circuit topologies. After a short discussion of the multiply operation, we discuss basic array multiplier. We also discuss different approaches to partial product generation, accumulation and their final summation.

THE MULTIPLIER: Definition

Consider two unsigned binary numbers X and Y that are M and N bits wide, respectively. To introduce the multiplication operation, it is useful to express X and y in the binary representation

$$X=\sum_{N}^{I=0} Yi2^i$$

$$Y=\sum_{N}^{I=0} Yi2^i$$

The simplest way to perform a multiplication is use a single two input adder. For inputs that are M and N bits wide, the multiplication takes M cycles , using an N bit adder. This shift and add algorithm for multiplication adds together M partial products. Each partial product is generated by multiplying the multiplicand with a bit of the multiplier-which, essentially, is an AND operation and by shifting the result on the basis of the multiplier bit's position. A faster way to implement multiplication is to resort to an approach similar to manually computing a multiplication. All the partial products are generated at the same time and organized in an array. A multioperand addition is applied to compute the final product.

III. BOOTH MULTIPLIER

Partial product result from logical AND of multiplicand X with a multiplier bit Yi, Each row in the partial-product array is either a copy of the multiplicand or a row of zeroes. Careful optimization of partial product generation can lead to some substantial delay and area reductions. Note that in most cases the partial product array has many zeros that have no impact on the result and thus represent a waste of effort when added .In the case of multiplier consisting of all ones ,all the partial product exit ,while in the case of all zeros ,there is none .This observation allows us to reduce the number of generated partial products by half.
Assume, for example, an eight-bit multiplier of form 0111110, which produce six nonzero partial-product rows. One can substantially reduce the number non zeros by

recording this number of in to different format. The reader can verify that the form 10000010.Using this format, we have add only two partial products, but the final adder has to able to perform subtraction as well. This type of information is called Booth recording, and it reduces partial product to , at most one half. It ensures that for every two consecutive bits, at most one bit will be 1 or -1.reducing the number of partial products is equivalent to reducing the number of additions, which leads to speed up as well as an area reduction. Formally, this transformation is equivalent to formatting the multiplier word in to base -4 schemes, instead of the usual the binary format.

Note that 10101----1 represents the worst case multiplier input because it generates the most partial products (one half).While the multiplication with {0,1} is equivalent to AND operation , multiplying with {-2,-1,0,1,2} requires a combination of inversion and shift logic. The encoding can be performed on the fly and requires some simple logic gates.

Having a variable size partial product array is not practical for multiplier design, and a modified booth's recording is the most often used instead. The multiplier is portioned into three-bit groups that overlap by one bit. Each group of three is recorded, as shown in table, and forms one partial product. The resulting number of partial produces equals half of the multiplier width. The input bits to the recording process are the two current bits, combined with the upper bit from the next group, moving from msb to lsb.
In simple terms, the modified booth's recording essentially examines the multiplier for string of ones from msb to lsb and replaces them with a leading 1,and a -1at the end of string. For example .011 is understood as the beginning of a string of ones and is therefore replaced by a leading 1 (or 100), while 110 is seen as the end of a string and is replaced by a -1 at the least significant position (or 0-10)

IV. PARTIAL PRODUCT ACCUMULATION

After the partial product is generated, they must be summed. This accumulation is essentially a multioperand addition. A straight forward way to accumulate partial products is by using a number of adders that will form an array – hence, the name, array multiplier. A more sophisticated procedure performs the addition in a tree format.
Formula for conversion

Y={-2*Bi-1}+Bi+Bi+1.

V. THE ARRAY MULTIPLIER

The composition of an array multiplier is shown in figure .There is one to in topological correspondence between this hardware structure and the manual multiplication shown figure. The generation of N partial product require N*M two bit AND gates(in the style of figure)most of the area of multiplier is devoted to the adding of the N partial products, which requires N-1 M bit adders. The shifting of the partial products for their proper alignment is performed by simple routing and does not require any logic. The overall structure can easily be compacted into a rectangle, resulting in very efficient layout. Due to the array organisation, determining the propagation delay of this circuit is not straightforward. Consider the implementation of figure .the partial sum address is implemented as ripple carry structures. Performance optimisation requires that the critical timing path be indentified first.

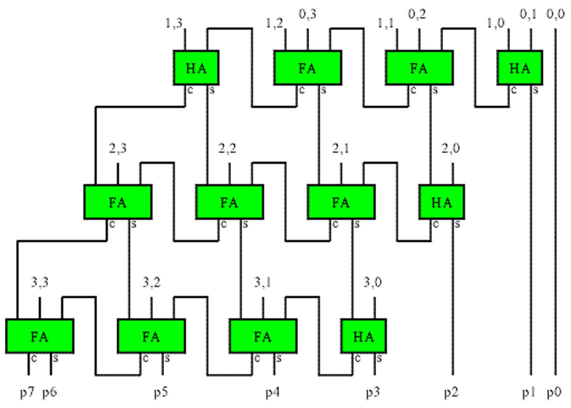


Fig1.Array Multiplier

VI. CARRY SAVE MULTIPLIER

Due the large number of almost identical critical path, increasing the performance of the structure of figure through transistor sizing yields marginal

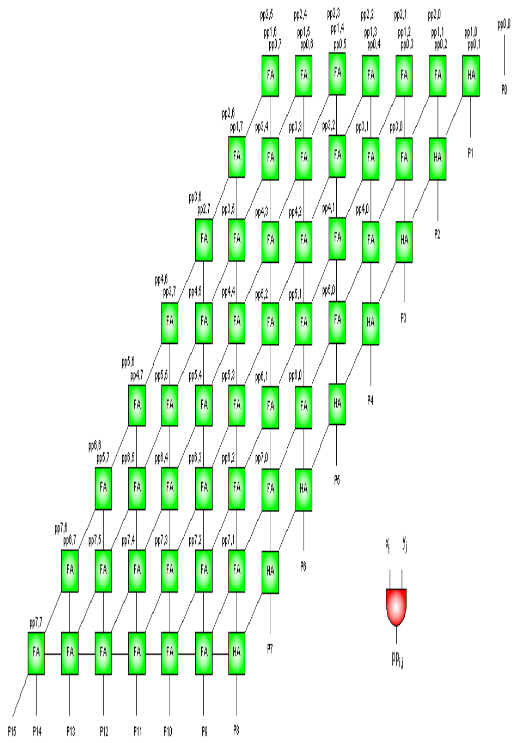
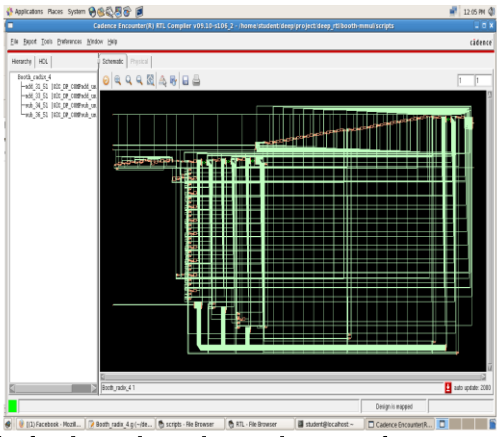


Fig2.Carry Save Multiplier

Benefits: A more efficient realization can be obtained by noticing that the multiplication result does not change when the output carry bits are passed diagonally downwards

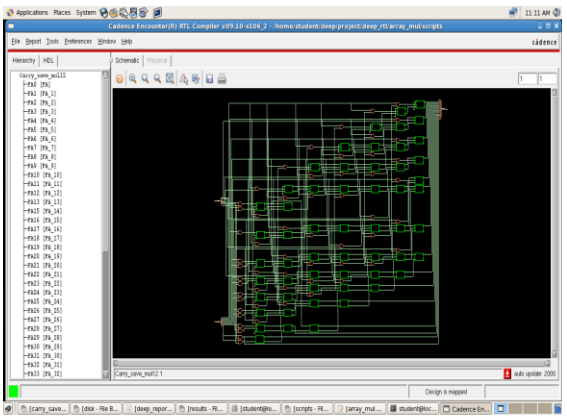


instead of only to the right, as shown in figure, we include an extra adder called vector merging adder to generate the final result. The resulting multiplier is called carry save multiplier, because the carry bits are not immediately added, but rather are saved for the next adder stage. In the final stage, while the structure has slightly increased area cost, it has the advantage it has the advantage that the worst case critical path is shorter and uniquely defined

VII. RESULT

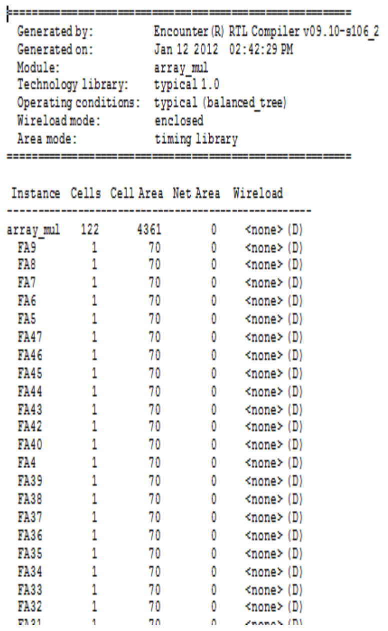
RT L FOR CARRY SAVE MULTIPLIER

RTl FOR BOOTH _RADIX4 MULTIPLIER



RTL FOR ARRAY MULTIPLIER

AREA REPORT FOR CARRY SAVE MULTIPLIER

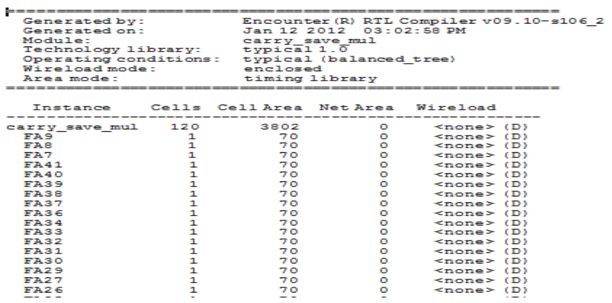


```
=================================================
  Generated by:          Encounter (R) RTL Compiler v09.10-s106_2
  Generated on:          Jan 12 2012   02:42:29 PM
  Module:                array_mul
  Technology library:    typical 1.0
  Operating conditions:  typical (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
=================================================

 Instance  Cells  Cell Area  Net Area   Wireload
------------------------------------------------
array_mul   122      4361         0     <none> (D)
  FA9         1        70         0     <none> (D)
  FA8         1        70         0     <none> (D)
  FA7         1        70         0     <none> (D)
  FA6         1        70         0     <none> (D)
  FA5         1        70         0     <none> (D)
  FA47        1        70         0     <none> (D)
  FA46        1        70         0     <none> (D)
  FA45        1        70         0     <none> (D)
  FA44        1        70         0     <none> (D)
  FA43        1        70         0     <none> (D)
  FA42        1        70         0     <none> (D)
  FA40        1        70         0     <none> (D)
  FA4         1        70         0     <none> (D)
  FA39        1        70         0     <none> (D)
  FA38        1        70         0     <none> (D)
  FA37        1        70         0     <none> (D)
  FA36        1        70         0     <none> (D)
  FA35        1        70         0     <none> (D)
  FA34        1        70         0     <none> (D)
  FA33        1        70         0     <none> (D)
  FA32        1        70         0     <none> (D)
  FA31        1        70         0     <none> (D)
```

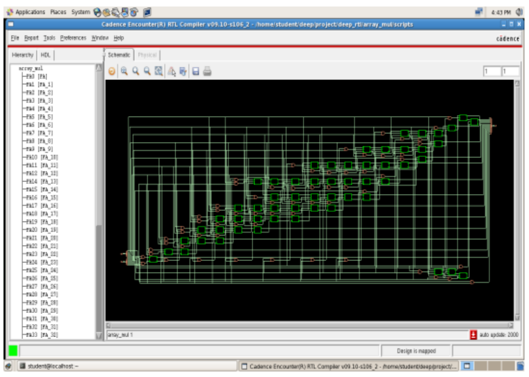
AREA REPORT FOR BOOTH RADIX_4 MULTIPLIER

```
=================================================
  Generated by:          Encounter (R) RTL Compiler v09.10-s106_2
  Generated on:          Jan 11 2012   12:06:03 PM
  Module:                Booth_radix_4
  Technology library:    typical 1.0
  Operating conditions:  typical (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
=================================================

   Instance    Cells  Cell Area  Net Area   Wireload
---------------------------------------------------
Booth_radix_4    216      3978         0     <none> (D)

 (D) = wireload is default in technology library
```

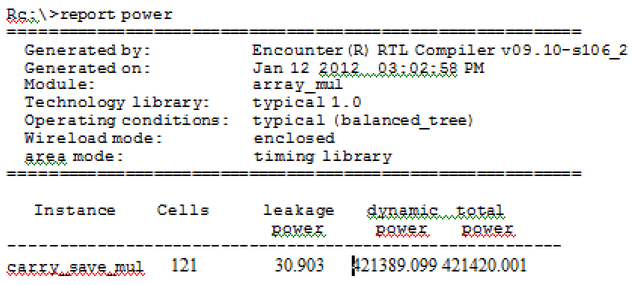
AREA REPORT FOR CARRY SAVE MULTIPLIER



```
============================================================
  Generated by:             Encounter (R) RTL Compiler v09.10-s106_2
  Generated on:             Jan 12 2012   03:02:58 PM
  Module:                   carry_save_mul
  Technology library:       typical 1.0
  Operating conditions:     typical (balanced_tree)
  Wireload mode:            enclosed
  Area mode:                timing library
============================================================

   Instance      Cells  Cell Area  Net Area   Wireload
-----------------------------------------------------------
carry_save_mul    120       3802         0     <none> (D)
  FA9               1         70         0     <none> (D)
  FA8               1         70         0     <none> (D)
  FA7               1         70         0     <none> (D)
  FA41              1         70         0     <none> (D)
  FA40              1         70         0     <none> (D)
  FA39              1         70         0     <none> (D)
  FA38              1         70         0     <none> (D)
  FA37              1         70         0     <none> (D)
  FA36              1         70         0     <none> (D)
  FA34              1         70         0     <none> (D)
  FA33              1         70         0     <none> (D)
  FA32              1         70         0     <none> (D)
  FA31              1         70         0     <none> (D)
  FA30              1         70         0     <none> (D)
  FA29              1         70         0     <none> (D)
  FA27              1         70         0     <none> (D)
  FA26              1         70         0     <none> (D)
```

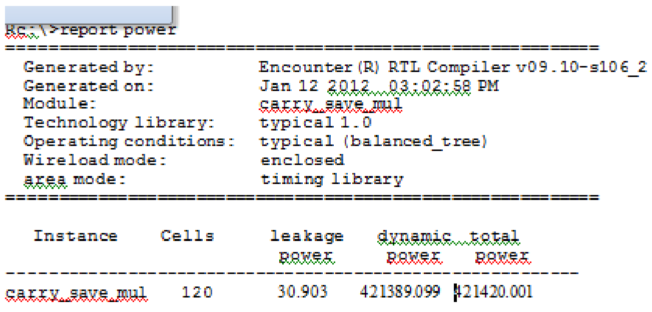

Cadence Encounter(R) RTL Compiler v09.10-s106_2 - /home/student/deep/project/deep_rtl/array_mul/scripts
File Report Tools Preferences Window Help

POWER REPORT FOR ARRAY MULTIPLIER



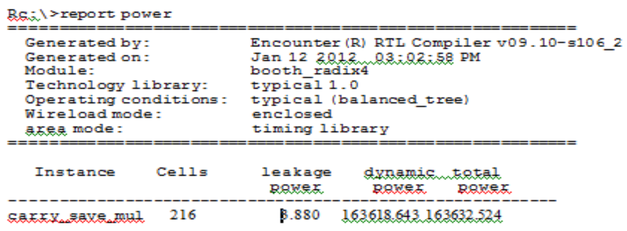
```
Rc:\>report power
============================================================
  Generated by:             Encounter (R) RTL Compiler v09.10-s106_2
  Generated on:             Jan 12 2012   03:02:58 PM
  Module:                   array_mul
  Technology library:       typical 1.0
  Operating conditions:     typical (balanced_tree)
  Wireload mode:            enclosed
  area mode:                timing library
============================================================

   Instance      Cells     leakage    dynamic    total
                            power      power     power
-----------------------------------------------------------
carry_save_mul    121      30.903    421389.099 421420.001
```

POWER REPORT FOR CARRY SAVE MULTIPLIER



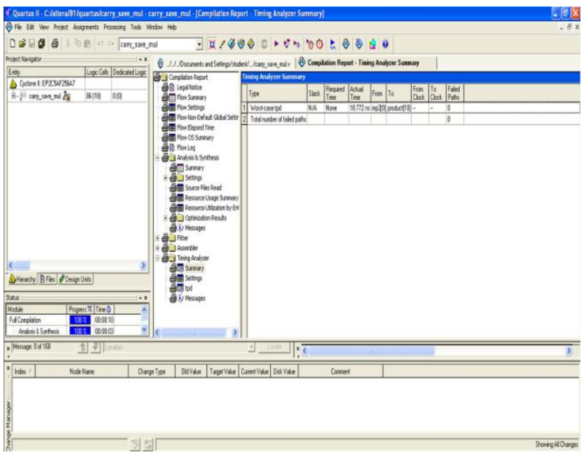
```
Rc:\>report power
============================================================
  Generated by:             Encounter (R) RTL Compiler v09.10-s106_2
  Generated on:             Jan 12 2012   03:02:58 PM
  Module:                   carry_save_mul
  Technology library:       typical 1.0
  Operating conditions:     typical (balanced_tree)
  Wireload mode:            enclosed
  area mode:                timing library
============================================================

   Instance      Cells     leakage    dynamic    total
                            power      power     power
-----------------------------------------------------------
carry_save_mul    120      30.903    421389.099 421420.001
```

THE POWER REPORT FOR BOOTH_RADIX4 MULTIPLIER



```
Rc:\>report power
============================================================
  Generated by:             Encounter (R) RTL Compiler v09.10-s106_2
  Generated on:             Jan 12 2012   03:02:58 PM
  Module:                   booth_radix4
  Technology library:       typical 1.0
  Operating conditions:     typical (balanced_tree)
  Wireload mode:            enclosed
  area mode:                timing library
============================================================

   Instance      Cells     leakage    dynamic    total
                            power      power     power
-----------------------------------------------------------
carry_save_mul    216      3.880     163618.643 163632.524
```

TIMING REPORT FOR CARRY SAVE MULTIPLIER

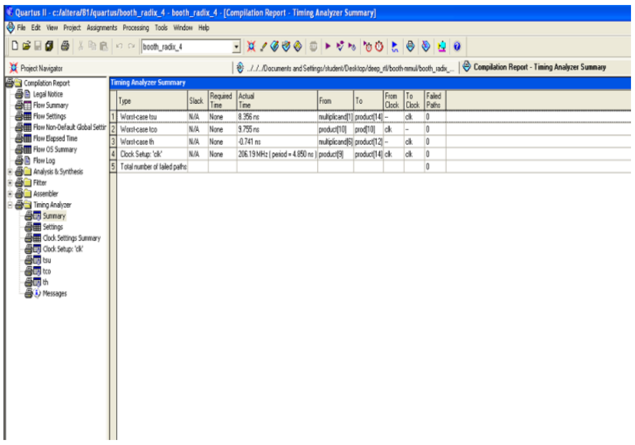


Compilation Report - Timing Analyzer Summary

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	18.772 ns	inp2[0]	product[10]	--	--	0
2	Total number of failed paths								0

TIMING REPORT FOR BOOTH_RADIX_4

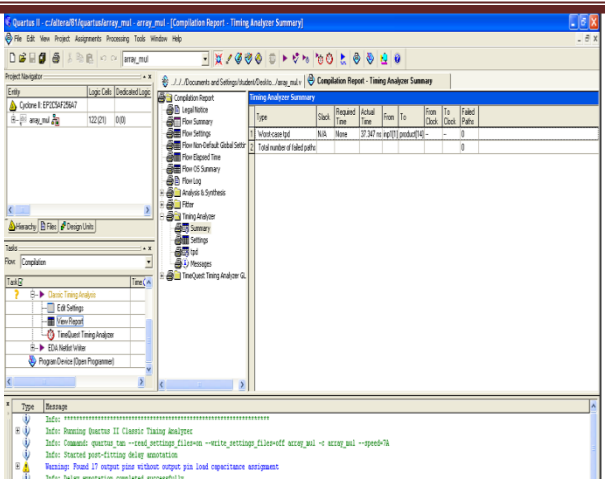


Quartus II - c:/altera/81/quartus/booth_radix_4 - booth_radix_4 - [Compilation Report - Timing Analyzer Summary]

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	8.356 ns	multiplicand[1]	product[14]	--	clk	0
2	Worst-case tco	N/A	None	9.755 ns	product[10]	prod[10]	clk	--	0
3	Worst-case th	N/A	None	-0.741 ns	multiplicand[6]	product[12]	--	clk	0
4	Clock Setup: 'clk'	N/A	None	206.19 MHz (period = 4.850 ns)	product[9]	product[14]	clk	clk	0
5	Total number of failed paths								0

THE TIMING REPORT FOR4 ARRAY_MULTIPLIER



Quartus II - c:/altera/81/quartus/array_mul - array_mul - [Compilation Report - Timing Analyzer Summary]

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	37.347 ns	inp1[1]	product[14]	--	--	0
2	Total number of failed paths								0

Entity	Logic Cells	Dedicated Logic
Cyclone II: EP2C5AF256A7		
array_mul	122 (21)	0 (0)

```
Info: ******************************************************************
Info: Running Quartus II Classic Timing Analyzer
Info: Command: quartus_tan --read_settings_files=on --write_settings_files=off array_mul -c array_mul --speed=7A
Info: Started post-fitting delay annotation
Warning: Found 17 output pins without output pin load capacitance assignment
Info: Delay annotation completed successfully
```

RESULT TABLE

TYPE (MULTIPLIER	AREA	SPEED	POWER
ARRAY MULTIPLIER	4361	37.342(ns)	421420.001(nW)
CARRY SAVE	3802.075	18.772(ns)	256195.667(nW)
BOOTH RADIX 4	3978.374	4.150(ns)	163632.542(nW)

CONCLUSION

When all three multipliers were compared we found that array multipliers are most power consuming multipliers and have maximum area. This is because it usage a large number of adders. As a result it slows down the system now the system has to do a lot of conclusion.

Multipliers are one of the most important component of many systems. So we always need to find a better solution in case of multipliers. Our multipliers should always consume less power and cover less power. So through our project we try to determine which of the three algorithm works the best. In the end we determine that radix 4 modified booth algorithm works the best.

3. REFRENCES

[1] Power area scalable pipelined area multiplier: Hanno lee
[2] Digital Integrated circuit Design: Rabaey
[3] Design And Implantation Of Different multipliers using VHDL: Mouita Ghosh
[4] Design Of High speed Modified Booth Multipliers Operating at GHz ranges: Soojin Kim and Kyeongsoon Cho
[5] Implantation of 4 bit array multiplier using Verilog HDL and it's testing on Spartan 2 FPGA: Leach Malvino and saha