International Journal of
Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 4 | Pages 1099-1106 | e-ISSN: 2347-470X

Optimization of Power and Area Using VLSI Implementation of MAC Unit Based on Additive Multiply Module

M Nagabushanam[1], Skandan Srikanth[2], Rushita Mupalla[3], Sushmitha S Kumar[4] and Swathi K[5]

[1]*Assistant Professor, Department of Electronics and Communication Engineering, M.S Ramaiah Institute of Technology, Bangalore, Karnataka 560054, India; nagabhushanam1971@msrit.edu*
[2]*Student, Department of Electronics and Communication Engineering, M.S Ramaiah Institute of Technology, Bangalore, Karnataka, India; skandans4@gmail.com*
[3]*Student, Department of Electronics and Communication Engineering, M.S Ramaiah Institute of Technology, Bangalore, Karnataka, India; rushita194@gmail.com*
[4]*Student, Department of Electronics and Communication Engineering, M.S Ramaiah Institute of Technology, Bangalore, Karnataka, India; sushmikumar8@gmail.com*
[5]*Student, Department of Electronics and Communication Engineering, M.S Ramaiah Institute of Technology, Bangalore, Karnataka, India; swathikantharajskg@gmail.com*

*__Correspondence:__ M. Nagabushanam; nagabhushanam1971@msrit.edu

ABSTRACT- The development of Digital Signal Processors (DSPs), graphical systems, Field Programmable Gate Arrays (FPGAs)/ Application-Specific Integrated Circuits (ASICs), and multimedia systems all rely heavily on digital circuits. The need for high-precision fixed-point or floating-point multipliers suitable for Very Large-Scale Integration (VLSI) implementation in high-speed DSP applications is developing rapidly. An integral part of any digital system is the multiplier. In digital systems as well as signal processing, the adder and multiplier seem to be the fundamental arithmetic units. Problems arise when using a multiplier in the realms of area, power, complexity, and speed. This paper details a more efficient MAC (Multiply- Accumulate) multiplier that has been tuned for space usage. The proposed design is more efficient, takes up less room, and has lower latency than conventional designs. The performance of the Additive Multiply Module (AMM) multiplier is measured against that of existing multipliers, where it serves as a module in the MAC reducing area and delay.

Keywords: AMM, Booth Multiplier, Dadda Multiplier, Cadence, MAC.

ARTICLE INFORMATION

Author(s): M Nagabushanam, Skandan S, Rushita M, Sushmitha S Kumar and Swathi K;

Received: 14/07/2022; **Accepted**: 19/09/2022; **Published**: 15/12/2022;
e-ISSN: 2347-470X;
Paper Id: IJEER 1407-08;
Citation: 10.37391/IJEER.100455
Webpage-link:
www.ijeer.forexjournal.co.in/archive/volume-10/ijeer-100455.html

crossref member
CROSSREF.ORG
THE CITATION LINKING BACKBONE

Publisher's Note: FOREX Publication stays neutral with regard to Jurisdictional claims in Published maps and institutional affiliations.

1. INTRODUCTION

The most crucial component of DSP is the MAC unit. The system's overall speed and efficiency are determined by the MAC unit. The MAC unit functions like a co-processor for such primary CPU to lighten the load. Therefore, creating a supercharged MAC unit is vital for microprocessors and DSP operations. Additionally, there is a growth in demand for electrical devices with high efficiencies, such as computers, smartphones, microcontrollers, and microprocessors. The most significant component of a MAC unit is the multiplier as shown in *figure 1*, which determines its efficiency. The multiplier consumes a significant extent of time delay, space, and power.

The most crucial component in planning is multipliers. The Multiplier and Accumulation units that make up a MAC unit and is responsible for math operations and its working is as shown in *figure 2*. There are numerous multiplication algorithms, and you can select one based on how well it performs. Modified Booth multiplier, Booth multiplier, Dadda tree multiplier, Array multiplier, AMM multiplier, and Wallace tree multiplier are some examples of different multiplication algorithms. The AMM, Booth, and Dadda multipliers are used in place of the multiplier block in this MAC unit. The basic full adder is what the adder block consists of. To compare the three parameters, namely power, area, and delay, the synthesis findings are carried out for both 8-bit and 16-bit MAC units.

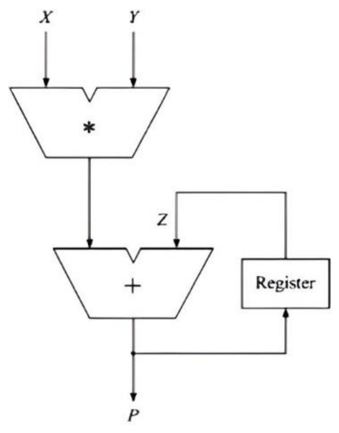


Figure 1: MAC unit basic block diagram

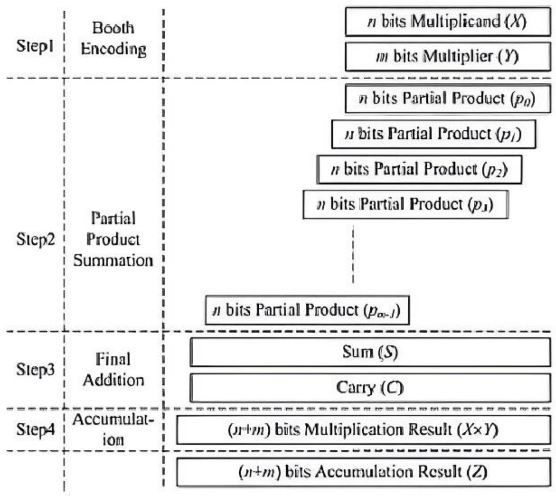

Step1
Booth Encoding
n bits Multiplicand (X)
m bits Multiplier (Y)
n bits Partial Product (p_0)
n bits Partial Product (p_1)
n bits Partial Product (p_2)
n bits Partial Product (p_3)
Step2
Partial Product Summation
n bits Partial Product (p_{m-1})
Step3
Final Addition
Sum (S)
Carry (C)
Step4
Accumulat-ion
(n+m) bits Multiplication Result (X×Y)
(n+m) bits Accumulation Result (Z)

Figure 2: Working of MAC unit

2. LITERATURE SURVEY

This study offers a thorough and unbiased analysis of the MAC Unit implementation utilizing various multipliers. Array multiplier and Booth multiplier are some of the several multipliers that have been studied in *Table [1]*, and the data flow diagram of Booth multiplier is as shown in *figure 3*.

Table 1: Parameter comparison table between array multipliers and booth multipliers

Parameters	Array Multiplier	Booth Multiplier
Power Utilization	High	Medium
Area	Highest	Least
Complexity	Simple	Complex
Delay	Highest	Least
Implement	Easy	Difficult

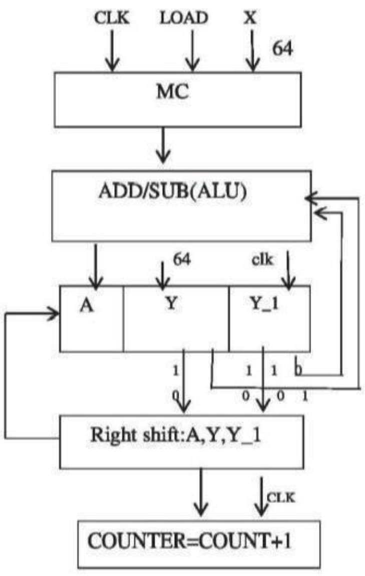


Figure 3: The data flow of a 64-bit Booth multiplier

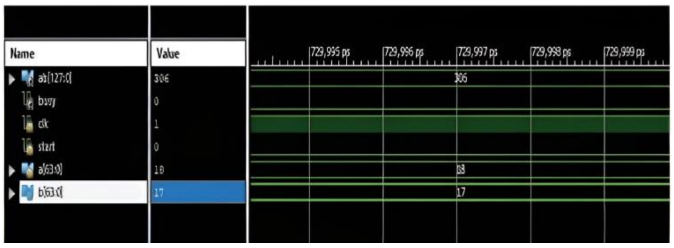


Figure 4: 64-bit Booth multiplier simulation results

The MAC unit's simulation result in Xilinx with EDA play area system is shown in *figure 4*.

Table 2: Device Utilization

Slice Logic	Used	Available	Utilization
Number of bonded IOBS	259	300	86%
Number of occupied Slices	125	10250	1%
Number of Slice Registers	200	82000	1%
Number of Slice LUT	425	41000	1%

Table 2 shows the device utilization of MAC Unit design using various multipliers. The number of used occupied slices is the least, the number of used slices LUT is the highest, and the number of bonded IOBS shown as 86%.

Table 3: Delay and Power Results

Size	32-bit	64-bit
Power (mW)	201.2	203.2
Delay (ns)	2.78	332

The analysis of the delay report for the MAC Unit design utilizing various multipliers is as shown in *table 3*. The 64-Bit MAC Unit uses hardly any energy so it may be used to develop less power-consuming devices most effectively. It uses 203.47 mW of electricity. The analysis of power in VLSI design is critical. The paper [2] displays the total power dissipation and cell area of various MAC units. When compared to other MAC units that are either created using the modified Booth or Dadda multipliers, the Wallace tree multiplier-based MAC expends less space, dissipates less power, and also causes less delay. As a result, a 64-bit MAC unit was created that makes use of the Wallace multiplier and carry-save adder. The arrangement is created utilizing Verilog-HDL and RTL compiler constructed with normal 180 nm TSMC technology libraries. Similar to earlier work, an 8-bit MAC is created using several adders and a multiplier.

The size and delay values of an 8-bit MAC unit that has a variety of adders and multipliers implemented are shown in Table IV. The Dadda Multiplier, Wallace Multiplier, and Modified Booth Algorithm are the multipliers that were used in the comparative analysis. The study employed three different adders: (i) Carry Look Ahead adder (ii) Carry Select Adder (CSeA) and (iii) Carry Save Adder (CSaA). The graphs are aligned with various 8-bit MAC unit types.

Table 4: Area and Delay Report for Different 8-Bit MAC

Sl. No		MAC Name	Area Report		Delay (ps)
			Cells	Cell Area (μm^2)	
1	Multiplier Adder	Modified Booth Algorithm Carry Look Ahead (MCLMAC)	221	7677	4995
2	Multiplier Adder	Dadda Multiplier Carry Look Ahead (DCLMA)	202	7904	4213
3	Multiplier Adder	Wallace Multiplier Carry Look Ahead (WCLMAC)	90	4398	3064
4	Multiplier Adder	Modified Booth Algorithm Carry Select (MCEMAC)	202	7418	4556
5	Multiplier Adder	Dadda Multiplier Carry Select (DCEMAC)	183	7637	4125
6	Multiplier Adder	Wallace Multiplier Carry Select (WCEMAC)	108	5013	1890
7	Multiplier Adder	Modified Booth Algorithm Carry Save (MCSMAC)	185	6819	4545
8	Multiplier Adder	Dadda Multiplier Carry Save (DCSMAC)	166	7099	3890
9	Multiplier Adder	Wallace Multiplier Carry Save (WCSMAC)	61	2947	1026

When two 128-bit numbers, x, and y, are assumed, the result generates partial products and carries C and S, respectively. Two ripples carry adders are utilized when adding two numbers with a half adder. The ripple carry adder (RCA) will take as much time as n full adders since it has to wait for the sum bit before it can generate the previous carry bit. Nonetheless, the CSaA generates all of the output values simultaneously, using less time overall than ripple-carry adders to complete the computation. As a result, the last stage uses Parallel-In Parallel-Out (PIPO) as an accumulator.

Table 5: Power Report for Various 8-bit MAC Unit

Sl. No	MAC Name	Power Report		
		Total Power (nW)	Leakage Power (nW)	Dynamic Power (nW)
1	MCLMAC	1813161	252	1812909
2	DCLMAC	1849307	270	1849037
3	WCLMAC	903852	132	903720
4	MCEMAC	1713749	242	1713506
5	DCEMAC	1721371	260	1721111
6	WCEMAC	657797	156	657640
7	MCSMAC	1518013	218	1517795
8	DCSMAC	1508825	237	1508588
9	WCSMAC	435478	79	435399

3. METHODOLOGY

3.1 AMM Multiplier

In this proposed paper, the implementation of an 8-bit with a 16-bit of MAC unit with various multipliers such as AMM, Booth Multiplier, and Dadda Multiplier was done. The Additive Multiple Modules get additional inputs by append them for the input operands product. A 4x2 AMM performs the arithmetic operation which is p = ax+y+z. In this case, (a) and (b) each represent the 4-bit multiplicand as well as a 2-bit multiplier. As 4+2=6, the product p consists of 6 bits. Four bits make up z.

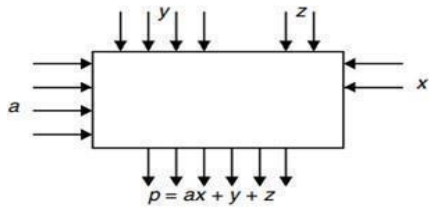


Figure 5: Block diagram of 8-bit AMM Multiplier

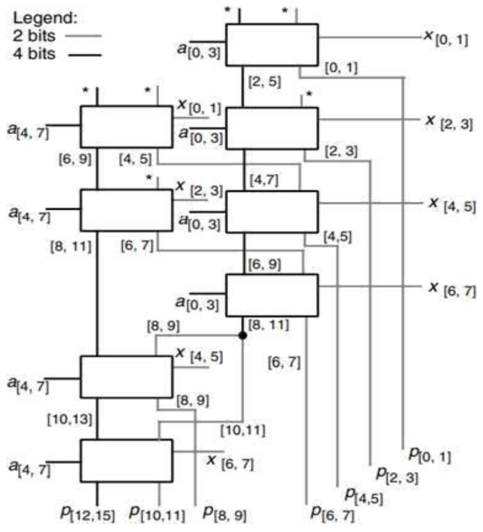


Figure 6: 8x8 AMM Multiplier built using 4x2 AMM Multiplier

Figure 5 shows a basic block diagram of a 4x2 AMM multiplier and *figure 6*. Shows an 8-bit AMM multiplier built using two 4x2 AMM multipliers. *Figure 7* indicates the product generation for given inputs using the dot method used in an AMM multiplier.

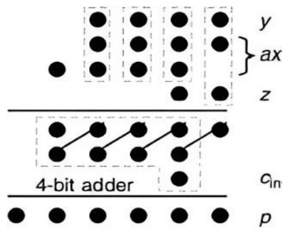


Figure 7: Dot Notations of AMM Multiplication

3.2 DADDA Multiplier
It builds up the total of partial products using various full and half adders. While conceptually identical to a Wallace tree multiplier, this version benefits from fewer gates and marginally improved performance due to a revised reduction tree. *Figure 8* shows the various stages that go on in a Dadda multiplier to get the desired output.

Dadda Multiplier uses the following steps to generate output:

- Multiply every bit of a_1, by every single bit of a_2, resulting in Voltage1 (V_1) results.
- When using full and half adders, reduce the amount of the partial products as in every stage until there are nearly no bits remaining for each weight.
- The addition of the final result is done using a traditional adder.

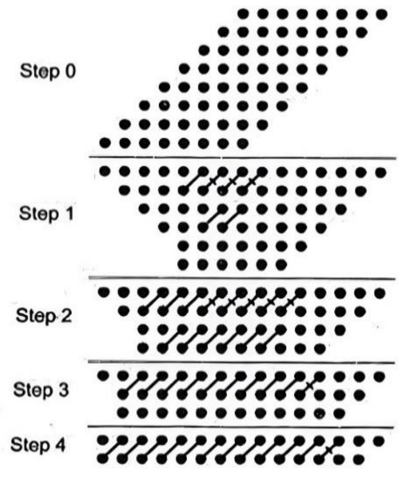


Figure 8: Dot notation of 8x8 Dadda Multiplier

3.3 Booth Multiplier
The general block diagram of Booth multiplier is as shown in *Figure 9*. Two signed binary numbers are multiplied using the two's complement notation via Booth's multiplication algorithm as shown in *Figure 10*. Although the multiplicand and product representations are often both in two's complement representation, any integer that supports subtraction and addition will work as well, it is not necessary to choose one. There are numerous modifications and improvements on certain specifics. The functionality is frequently explained as transforming strings of 1's into high-order +1's and low-order 1's at the endpoints of the string in the multiplier. The net impact is regarded as a negative of the proper value when a string passes through the MSB because there is no high-order+1.

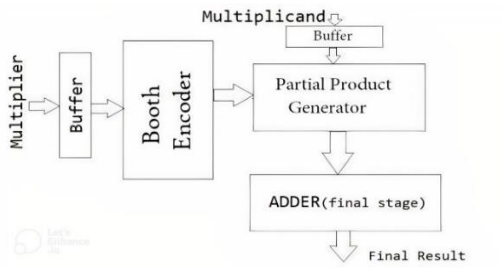
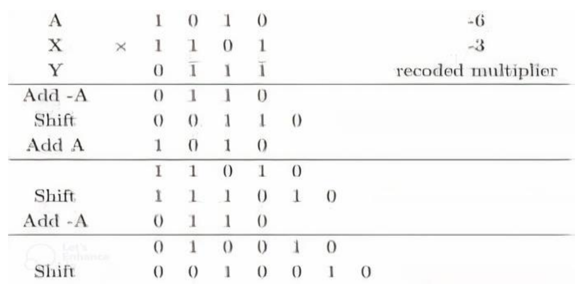


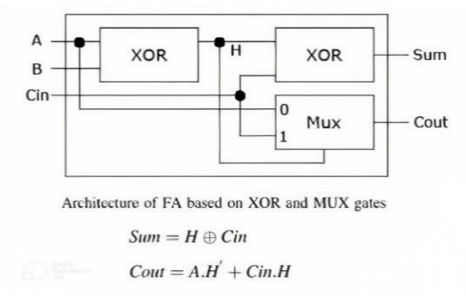
Figure 9: Block diagram of Booth Multiplier



A		1	0	1	0			-6
X	×	1	1	0	1			-3
Y		0	$\bar{1}$	1	$\bar{1}$			recoded multiplier
Add -A		0	1	1	0			
Shift		0	0	1	1	0		
Add A		1	0	1	0			
		1	1	0	1	0		
Shift		1	1	1	0	1	0	
Add -A		0	1	1	0			
		0	1	0	0	1	0	
Shift		0	0	1	0	0	1	0

Figure 10: Booth Multiplication Algorithm

3.4 Full Adder with XOR
The adder known as a "full adder" adds three inputs and generates two outputs. *A* and *B* make up the first two inputs and C_{in} is the third input. The normal output is denoted as *S*, which is the sum, while the output carry is designated as C_{out} as shown in *figure 11*. Eight bits can be used to form a single-byte adder using full adder logic, and the carry bit can be cascaded from one adder to the next. A full adder is employed since a 1-bit half-adder cannot use a carry-in bit when one is available, therefore another 1-bit adder must be used. Three operands are added via a 1-bit complete adder, which produces 2-bit results.



$$Sum = H \oplus Cin$$

$$Cout = A.H' + Cin.H$$

Figure 11: Full Adder with XOR

3.5 Accumulator Unit
The accumulator, which is a register, is where the products' total is kept. It is commonly employed in MAC units and ALUs (Arithmetic Logic Units). It may be unnecessary to perform additional summing operations by saving values in the accumulator. The delay time of the accumulator should be swift enough to keep up with fast adders. An accumulator is often used as a register to hold interim logical or numerical data in multistep calculations. The block diagram of a register unit that serves as an accumulator is shown in *Figure 12*. It acts as a temporary repository for these calculations.

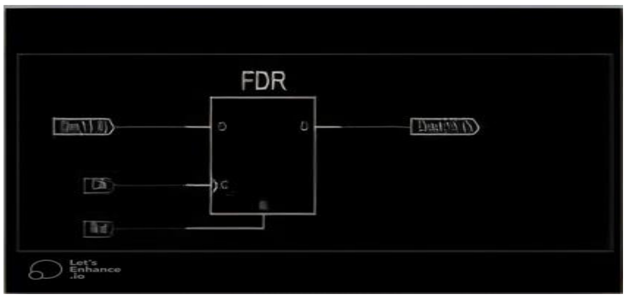


Figure 12: Accumulator Unit

It acts as a short-term repository for these calculations. The value is gradually overwritten to hold the interim results each time one of these actions is carried out. For example, in an operation that requires adding many numbers, the accumulator would initially store the result of adding the first two integers. After the subsequent number is added, the net result then replaces the previous result in the accumulator. This procedure is repeated until the total amount is available and all the numbers have been added. This total is calculated and written to the main memory or another register afterward.

4. RESULTS AND DISCUSSION

4.1 MAC Unit with AMM Amplifier

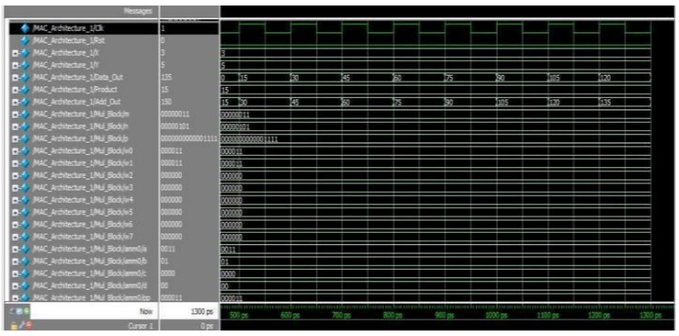
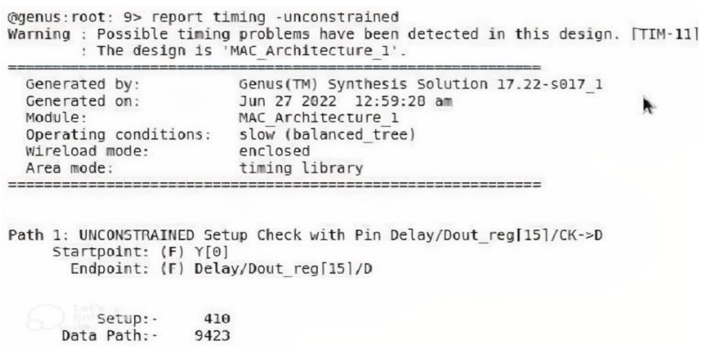


Figure 13: Simulation results of 8-bit MAC unit using AMM Multiplier

The following are the synthesis reports generated for an 8-bit MAC unit that has an AMM Multiplier implemented in it. The results of a simulation of 8-bit MAC unit with an AMM multiplier are shown in *figure 13*. The MAC unit's power, delay, and area results are depicted in *figures 14, 15, and 16.*

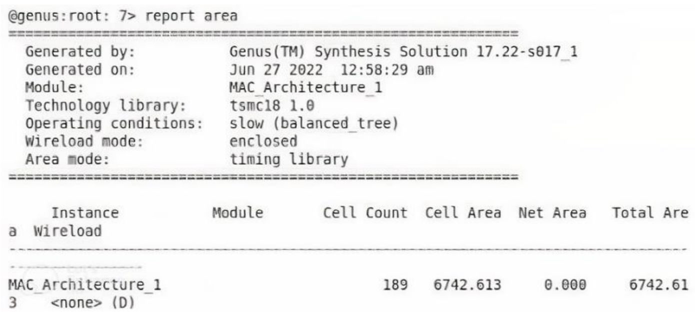


```
@genus:root: 9> report timing -unconstrained
Warning : Possible timing problems have been detected in this design. [TIM-11]
        : The design is 'MAC_Architecture_1'.
============================================================
  Generated by:            Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:            Jun 27 2022  12:59:20 am
  Module:                  MAC_Architecture_1
  Operating conditions:    slow (balanced_tree)
  Wireload mode:           enclosed
  Area mode:               timing library
============================================================

Path 1: UNCONSTRAINED Setup Check with Pin Delay/Dout_reg[15]/CK->D
     Startpoint: (F) Y[0]
       Endpoint: (F) Delay/Dout_reg[15]/D


         Setup:-      410
     Data Path:-     9423
```

Figure 14: Delay synthesis for 8-bit MAC unit with AMM

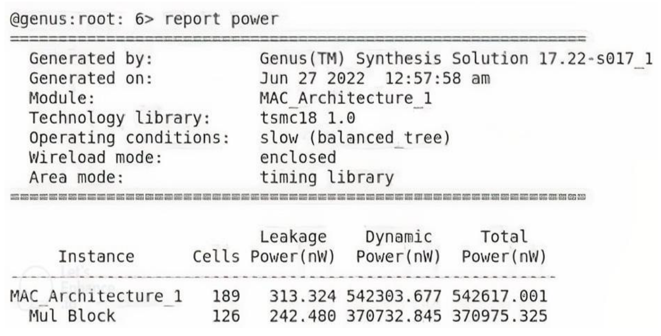


```
@genus:root: 7> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:           Jun 27 2022  12:58:29 am
  Module:                 MAC_Architecture_1
  Technology library:     tsmc18 1.0
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

     Instance          Module        Cell Count  Cell Area  Net Area   Total Are
a  Wireload
------------------------------------------------------------------------------

MAC_Architecture_1                          189   6742.613     0.000     6742.61
3    <none> (D)
```

Figure 15: Area synthesis for 8-bit MAC unit with AMM

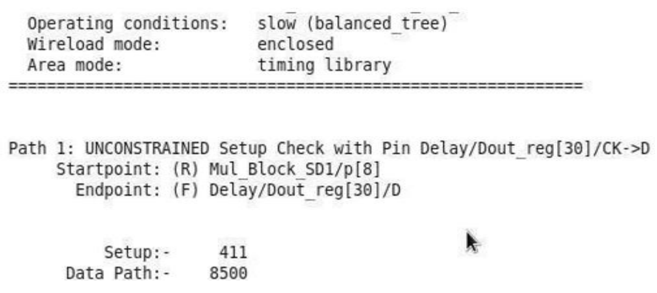


```
@genus:root: 6> report power
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:           Jun 27 2022  12:57:58 am
  Module:                 MAC_Architecture_1
  Technology library:     tsmc18 1.0
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                                Leakage   Dynamic     Total
     Instance          Cells  Power(nW)  Power(nW)  Power(nW)
--------------------------------------------------------------
MAC_Architecture_1      189    313.324 542303.677 542617.001
  Mul_Block             126    242.480 370732.845 370975.325
```

Figure 16: Power synthesis for 8-bit MAC unit with AMM

In order to make a thorough comparison, the AMM synthesis results are provided for the 16-bit MAC unit. The delay, power, and area report for the 16-bit MAC unit of AMM multiplier is shown in *figures 17, 18, and 19*.

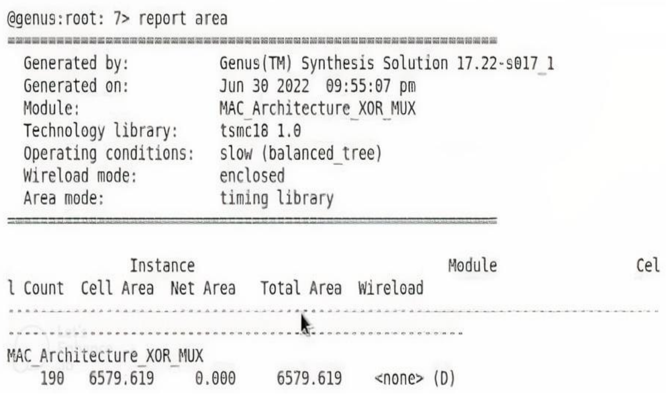


```
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

Path 1: UNCONSTRAINED Setup Check with Pin Delay/Dout_reg[30]/CK->D
     Startpoint: (R) Mul_Block_SD1/p[8]
       Endpoint: (F) Delay/Dout_reg[30]/D


         Setup:-      411
     Data Path:-     8500
```

Figure 17: Delay synthesis for 16-bit MAC unit with AMM

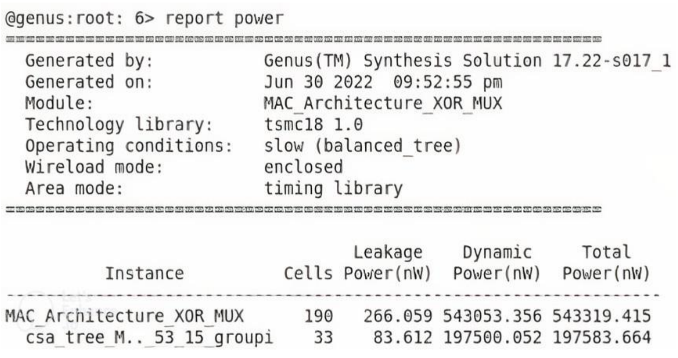


```
@genus:root: 7> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:           Jun 30 2022  09:55:07 pm
  Module:                 MAC_Architecture_XOR_MUX
  Technology library:     tsmc18 1.0
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

              Instance                     Module              Cel
l Count  Cell Area  Net Area   Total Area  Wireload
------------------------------------------------------------------
--------------------------------------------------

MAC_Architecture_XOR_MUX
   190   6579.619     0.000     6579.619    <none> (D)
```

Figure 18: Area synthesis for 16-bit MAC unit with AMM



```
@genus:root: 6> report power
============================================================
  Generated by:            Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:            Jun 30 2022  09:52:55 pm
  Module:                  MAC_Architecture_XOR_MUX
  Technology library:      tsmc18 1.0
  Operating conditions:    slow (balanced_tree)
  Wireload mode:           enclosed
  Area mode:               timing library
============================================================

                                    Leakage   Dynamic     Total
         Instance           Cells  Power(nW)  Power(nW)  Power(nW)
--------------------------------------------------------------------
MAC_Architecture_XOR_MUX      190    266.059 543053.356 543319.415
  csa_tree_M.._53_15_groupi    33     83.612 197500.052 197583.664
```

Figure 19: Power synthesis for 16-bit MAC unit with AMM

4.2 MAC Unit with Booth Amplifier

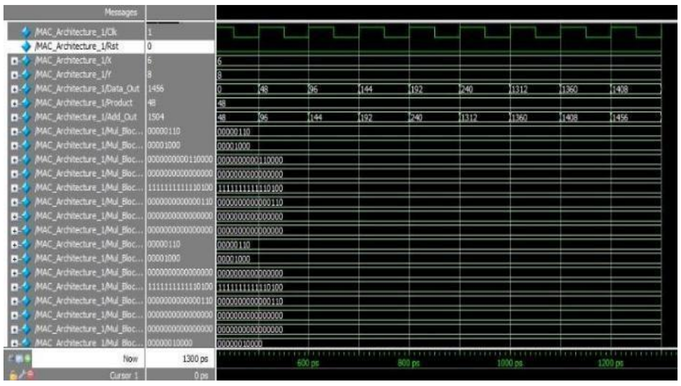
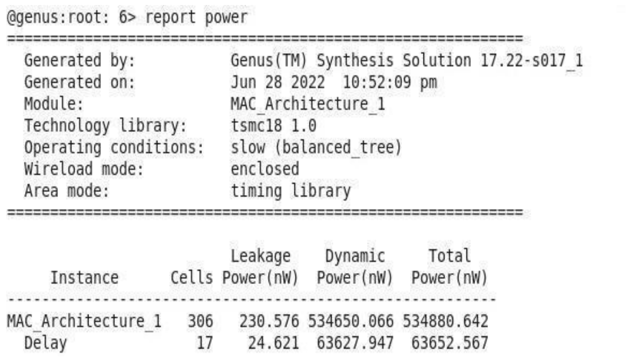


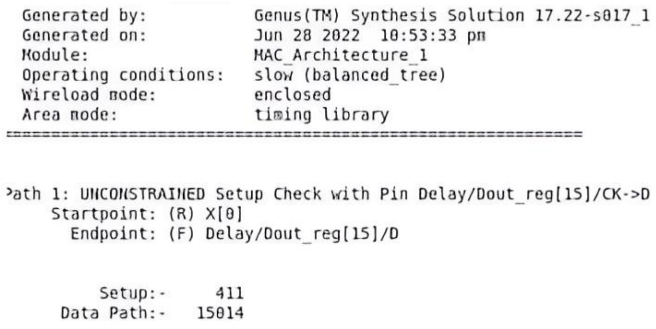
Figure 20: Simulation results of MAC unit using Booth Multiplier



```
@genus:root: 6> report power
============================================================
  Generated by:            Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:            Jun 28 2022  10:52:09 pm
  Module:                  MAC_Architecture_1
  Technology library:      tsmc18 1.0
  Operating conditions:    slow (balanced_tree)
  Wireload mode:           enclosed
  Area mode:               timing library
============================================================

                        Leakage    Dynamic     Total
     Instance      Cells Power(nW)  Power(nW)  Power(nW)
-------------------------------------------------------
MAC_Architecture_1   306   230.576 534650.066 534880.642
  Delay               17    24.621  63627.947  63652.567
```

Figure 21: 8-Bit MAC Unit Power Synthesis using a Booth Multiplier

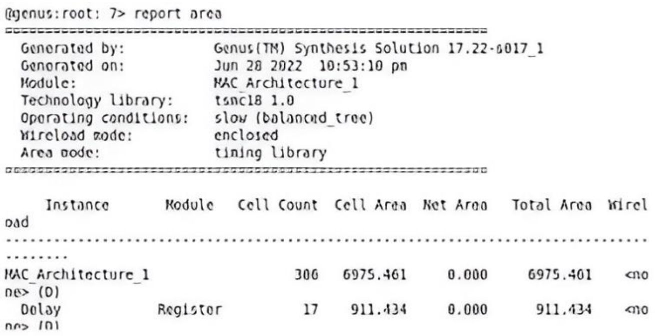


```
  Generated by:            Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:            Jun 28 2022  10:53:33 pm
  Module:                  MAC_Architecture_1
  Operating conditions:    slow (balanced_tree)
  Wireload mode:           enclosed
  Area mode:               timing library
============================================================

Path 1: UNCONSTRAINED Setup Check with Pin Delay/Dout_reg[15]/CK->D
     Startpoint: (R) X[0]
       Endpoint: (F) Delay/Dout_reg[15]/D


         Setup:-      411
     Data Path:-    15014
```

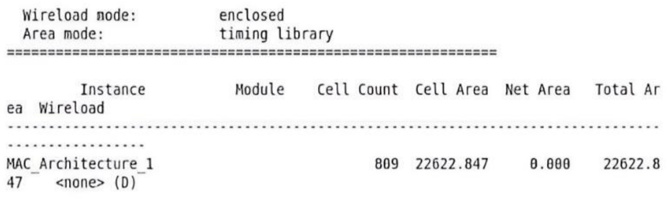
Figure 22: 8-Bit MAC Unit Delay Synthesis using a Booth Multiplier



```
@genus:root: 7> report area
============================================================
  Generated by:            Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:            Jun 28 2022  10:53:10 pm
  Module:                  MAC_Architecture_1
  Technology library:      tsmc18 1.0
  Operating conditions:    slow (balanced_tree)
  Wireload mode:           enclosed
  Area mode:               timing library
============================================================

    Instance       Module    Cell Count  Cell Area  Net Area   Total Area  Wirel
oad
-------------------------------------------------------------------------------
--------
MAC_Architecture_1                306    6975.461     0.000     6975.401    <no
ne> (D)
  Delay            Register        17     911.434     0.000      911.434    <no
ne> (D)
```

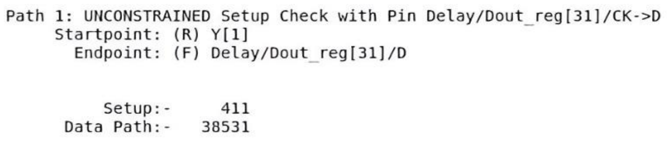
Figure 23: Area synthesis for 8-Bit MAC unit with Booth Multiplier



```
  Wireload mode:            enclosed
  Area mode:                timing library
============================================================

        Instance          Module    Cell Count  Cell Area  Net Area   Total Ar
ea  Wireload
----------------------------------------------------------------------------
------------------
MAC_Architecture_1                        809  22622.847     0.000    22622.8
47    <none> (D)
```

Figure 24: Area synthesis for 16-Bit MAC unit with Booth Multiplier

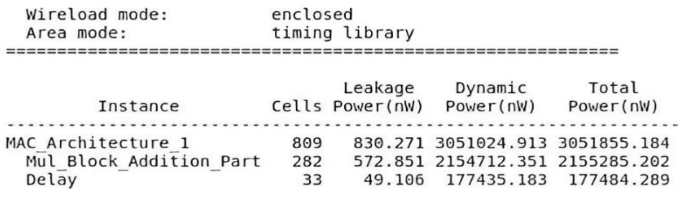


```
Path 1: UNCONSTRAINED Setup Check with Pin Delay/Dout_reg[31]/CK->D
     Startpoint: (R) Y[1]
       Endpoint: (F) Delay/Dout_reg[31]/D


         Setup:-      411
     Data Path:-    38531
```

Figure 25: 16-Bit MAC Unit Delay Synthesis using a Booth Multiplier

The results for MAC unit using Booth multiplier are shown above. The simulation findings of MAC unit with the Booth multiplier are displayed in *Figure 20. Figures 21, 22, and 23* depict the results of the synthesis of an 8-bit MAC unit using the Booth multiplier, while *Figures 24, 25, and 26* depict the outcomes of the synthesis of the 16-bit MAC unit with the Booth multiplier.



```
  Wireload mode:            enclosed
  Area mode:                timing library
============================================================

                                    Leakage    Dynamic      Total
        Instance            Cells  Power(nW)   Power(nW)    Power(nW)
-----------------------------------------------------------------------
MAC_Architecture_1            809    830.271 3051024.913 3051855.184
  Mul_Block_Addition_Part     282    572.851 2154712.351 2155285.202
  Delay                        33     49.106  177435.183  177484.289
```

Figure 26: Area synthesis for 8-Bit MAC unit with Booth Multiplier

4.3 MAC Unit with Dadda Amplifier

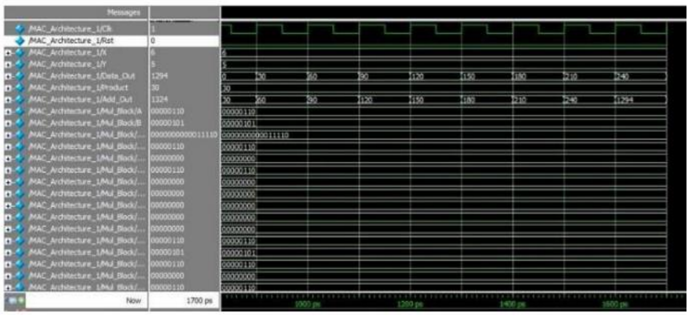
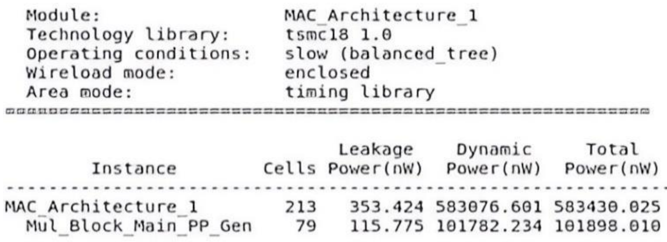


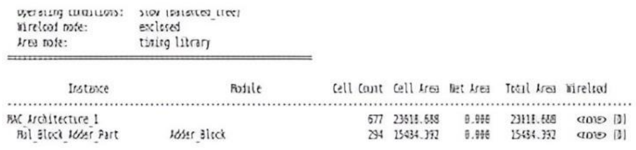
Figure 27: Simulation results for MAC unit using Dadda Multiplier



```
  Module:                   MAC_Architecture_1
  Technology library:       tsmc18 1.0
  Operating conditions:     slow (balanced_tree)
  Wireload mode:            enclosed
  Area mode:                timing library
============================================================

                                  Leakage    Dynamic     Total
        Instance          Cells  Power(nW)  Power(nW)   Power(nW)
-----------------------------------------------------------------
MAC_Architecture_1          213   353.424 583076.601 583430.025
  Mul_Block_Main_PP_Gen      79   115.775 101782.234 101898.010
```

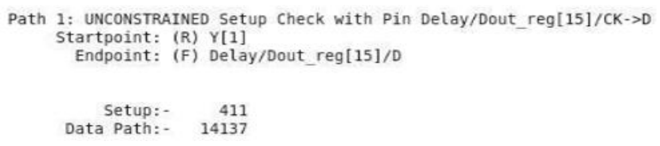
Figure 28: Power synthesis for 8-bit MAC unit with Dadda Multiplier



```
Operating conditions:   slow (balanced_tree)
Wireload mode:          enclosed
Area mode:              timing library
==================================================

        Instance              Module         Cell Count  Cell Area  Net Area  Total Area  Wireload
----------------------------------------------------------------------------------------------------
MAC_Architecture_1                                  677  23018.688     0.000   23018.688  <none> (D)
  Mul_Block_Adder_Part     Adder_Block              294  15484.392     0.000   15484.392  <none> (D)
```

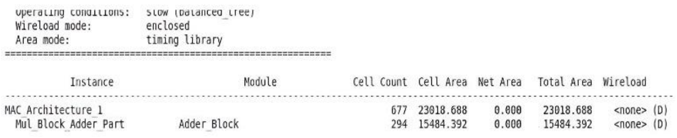
Figure 29: Area synthesis for 8-bit MAC unit with Dadda multiplier



```
Path 1: UNCONSTRAINED Setup Check with Pin Delay/Dout_reg[15]/CK->D
      Startpoint: (R) Y[1]
        Endpoint: (F) Delay/Dout_reg[15]/D


           Setup:-      411
       Data Path:-    14137
```

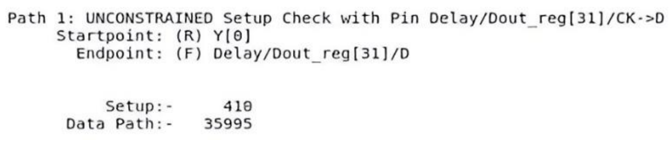
Figure 30: Delay synthesis for 8-bit MAC unit with Dadda multiplier



```
Operating conditions:   slow (balanced_tree)
Wireload mode:          enclosed
Area mode:              timing library
==================================================

        Instance              Module         Cell Count  Cell Area  Net Area  Total Area  Wireload
----------------------------------------------------------------------------------------------------
MAC_Architecture_1                                  677  23018.688     0.000   23018.688  <none> (D)
  Mul_Block_Adder_Part     Adder_Block              294  15484.392     0.000   15484.392  <none> (D)
```

Figure 31: Area synthesis for 16-bit MAC unit with Dadda multiplier



```
Path 1: UNCONSTRAINED Setup Check with Pin Delay/Dout_reg[31]/CK->D
      Startpoint: (R) Y[0]
        Endpoint: (F) Delay/Dout_reg[31]/D


           Setup:-      410
       Data Path:-    35995
```

Figure 32: Delay synthesis for 16-bit MAC unit with Dadda multiplier

The above shown are the results for the MAC unit using the Dadda multiplier. *Figure 27* represents the simulation results of the MAC unit using a booth multiplier. *Figure 28, 29 and 30* show the synthesis report of an 8-bit MAC unit with a Dadda multiplier and *figure 31,32 and 33 show* the synthesis results of the16-bit MAC unit with the Dadda multiplier.

Table 6: Comparison table for 8-bit MAC Unit

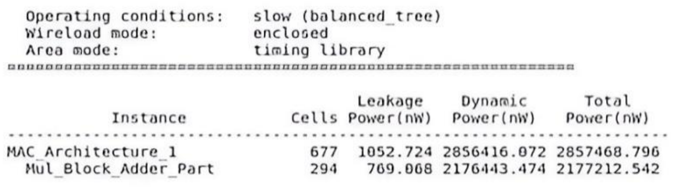
MAC Unit	Power (nW)	Area (µm^2)	Delay (nS)
MAC unit with AMM	542617	6742.6	9423
MAC unit with Booth Multiplier	534880	6975.46	15014
MAC unit with Dadda Multiplier	583430.3	7098.5	14137

Table 6 shows the synthesis results for the MAC Unit with 8-Bit has been summarized and it is seen that the MAC unit with 8-bit AMM Multiplier has a better design based on Area and Time Delay.

Table 7: Comparison table for 16-bit MAC Unit

MAC Unit	Power(nW)	Area (µm^2)	Delay(nS)
MAC unit with AMM	543319	6579	8500
MAC unit with Booth Multiplier	3051855.184	22622.84	38531
MAC unit with Dadda Multiplier	2857468.8	23018.688	35995

The 16-Bit MAC Unit synthesis findings are reported in *Table 7*, and it is clear that the 16-Bit MAC unit of AMM Multiplier provides a better design in terms of time delay and area.



```
Operating conditions:   slow (balanced_tree)
Wireload mode:          enclosed
Area mode:              timing library
==============================================================

                                     Leakage     Dynamic      Total
        Instance           Cells  Power(nW)   Power(nW)   Power(nW)
---------------------------------------------------------------------
MAC_Architecture_1           677   1052.724 2856416.072 2857468.796
  Mul_Block_Adder_Part       294    769.068 2176443.474 2177212.542
```

Figure 33: Power synthesis for 16-bit MAC unit with Dadda multiplier

From the results obtained, the conclusions are made that a MAC unit with an AMM multiplier gives a faster execution speed it can also be noticed that with an increase in the number of bits the area and delay significantly reduce that giving us a high-speed MAC unit.

5. CONCLUSION

The proposed work designs a modified MAC unit using the AMM multiplier and compares it with the Dadda and Booth multiplier for 8-bit and 16-bit and found that the MAC with AMM multiplier is better in terms of area, for 8-bit and is reduced by 3.4% compared to the Booth multiplier, and 5.2% with respect to Dadda multiplier. The delay is reduced by 37.2% in Booth multiplier and 33.3% in Dadda multiplier respectively. Similarly, for a 16-bit MAC multiplier, the area is reduced by 71% in comparison to the Booth multiplier and 71% for the Dadda multiplier. The delay is reduced by 77.9% and 76.3% respectively. These findings demonstrate that when bit count increases, delay and area decrease with AMM compared to conventional multipliers, making bit count ideal for high-speed DSP applications.

REFERENCES

[1] Akash Shaw, Vikas Gupta,2020, "Novel Design of High Speed 64-bit Optimized MAC Unit", IEEE International Conference for Innovation in Technology(INOCON)Bengaluru, India. Nov 6-8, 2020.

[2] Chilakala Jayanth Reddy, A. Prasad "High Performance and Implementation of 64-Bit MAC Units and Their Delay Comparison for Binary Multipliers" International Journal of VLSI System Design and Communication Systems Volume.04, IssueNo.09, September-2016.

[3] L. Jenila and R. Aroul Canessane (2022), Cross Layer Based Dynamic Traffic Scheduling Algorithm for Wireless Multimedia Sensor Network. IJEER 10(2), 399-404. DOI: 10.37391/IJEER.100256.

[4] Kadi Jaya Ramesh, Chepuri Pravalika, Dr.Rajkumar Sarma, "Multiply and Accumulate Architectures for Digital signal Processing and digital image Processing", Turkish Journal of Computer and Mathematics Education Vol.12 No. 11 (2021).

[5] Sufia Banu and Shweta Gupta (2022), Design and Leakage Power Optimization of 6T Static Random Access Memory Cell Using Cadence Virtuoso. IJEER 10(2), 341-346. DOI: 10.37391/IJEER.100246.

[6] Zhen gu, Shuguo Li, "Optimized Interpolation of Four-Term Karatsuba Multiplication and a Method of Avoiding Negative Multiplicands", IEEE Transactions On Circuits And Systems—I: Regular Papers, 2021.

[7] Moslem Heidarpur, Mitra Mirhassani, "An Efficient and High-Speed Overlap-Free Karatsuba-Based Finite-Field Multiplier for FGPA Implementation", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 29, No. 4, 2021.

[8] B.Hemalatha, Dr.Hari Shankar Srivastava, V. Vinay Kumar, “Design of MAC Unit For DSP Applications using Verilog HDL”, International Journal of Research in Advent Technology, Vol.7, No. 4S, April 2019.

[9] Priyanka Nain, Dr. G. S.Virdi, “Multiplier-Accumulator (MAC) Unit”, International Journal of Digital Application & Contemporary Research, Volume 5, Issue 3, October 2016.

[10] Pratibhadevi Tapashetti, Dr. Rajkumar, B. Kulkarni and Dr. S. S. Patil, “MAC Architectures Based on Modified Booth Algorithm”, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 5, Issue 12, December 2016.

[11] S. Aruna, S. Venkatesh and K. Srinivasa Naik, “A Low Power and High-Speed Array Multiplier Using On-The-Fly Conversion", International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-7, Issue-5S4, February 2019.

[12] K. Praveen Reddy and S. Aruna Mastani, "Implementation Of High-Performance 64-Bit Mac Unit for Dsp Processor", International journal of Current Engineering and Scientific Research, Vol.2, Issue-11, 2015.

[13] K. Rajesh, G. Umamaheswara Reddy, “FPGA Implementation of Multiplier-Accumulator Unit using Vedic multiplier and Reversible gates”. International Conference on Inventive Systems and Control (ICISC 2019).

[14] Duncan J.M Moss, David Boland, Philip H.W Leong, “A Two speed, Radix-4 Serial parallel Multiplier”, IEEE Transaction on Very Large Scale Integration System, Vol. 27, No. 4, pp: 769-777, 2019.

[15] Roshani Pawar, Dr. S. S. Shriramwar, "Design & Implementation of Area Efficient Low Power High-Speed MAC Unit using FPGA", IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI-2017).

[16] Suganthi Venkatachalam, Elizabeth Adams, Hyuk Jae Lee, Seok Bum Ko, "Design and Analysis of Area and Power Efficient Approximate Booth Multipliers". IEEE Transactions on Computers Vol. 68, Issue: 11, Nov. 1 2019.

[17] Chandrashekara M N, Rohit S, “Design of 8 Bit Vedic Multiplier Using Urdhya Triyagbhyam Sutra with Modified Carry Save Adder”, 2019 4th International Conference on Recent Trends on Electronics, Information, Communication& Technology (RTEICT 2019), MAY 17th & 18th 2019.

[18] R. Anitha, Sarat Kumar Sahoo,”A 32 Bit MAC Unit Design using Vedic Multiplier and Reversible Logic Gate” International Conference on Circuit, Power and Computing Technologies (ICCPCT)2015.

[19] Kavindra Dwivedi, R.K Sharma, "Hybrid Multiplier-based Optimized MAC Unit". 9th ICCCNT 2018 July 10-12, 2018, IISC, Bengaluru, India.

[20] M. Masadeh, O. Hasan, and S. Tahar, "Comparative study of approximate multipliers," in Proc. ACM Great Lakes Symp. VLSI, 2018, pp. 415–418.