CMPS 585

VLSI DESIGN

PROJECT REPORT

LOW POWER - HIGH PERFORMANCE MAC UNIT

Completed by

AMIT VYAS AND AMBRISH VARMA

asv 8379 and akv7806@cacs.usl.edu

Lecturer:

DR. M. A. BAYOUMI

TABLE OF CONTENTS

TABLE OF CONTENTS 2

TABLE OF FIGURES 3

ABSTRACT 4

PROJECT OBJECTIVE 5

INTRODUCTION 5

- **Choice of Architecture 6**

IMPLEMENTING THE MAC UNIT 7

- **Multiplier 7**

  Booth Encoder 8

  Multiplier (Adder) Array 10

  - PP-MUX 10

  PP- FA and PP-HA 11

  - ADD Cell 12

  Final Stage Adder 12

  - Ripple Carry Adder 12
  - Conditional Sum 13

- **Accumulator 14**

PERFORMANCE OF THE MAC UNIT 15

- **Performance of the Multiplier 15**

  Booth Encoder 15

  - Generation of booth encoded digits: 15
  - (b)Generation of sign extension bits; 15
  - (c)Transistor sizing: 16

  Adder Array 17

  - (a)Designing the MUX unit; 17
  - (b)Designing the 1 BIT ADDER cell: 18
  - (c)Designing the half adder cell; 18
  - (d)Designing the ADD cell: 18
  - (e)Performance of MULT2 and MULT6: 19

  Ripple Carry Adder 19

  Conditional Sum Adder 20

- **Performance of the Accumulator 21**

RESULTS OBTAINED 22

FUTURE WORK 26

DISCUSSION AND CONCLUSION 26

LIST OF REFERENCES AND BIBLIOGRAPHY 27

APPENDIX 1-SPICE INPUTS FOR MAC.MAG 28

APPENDIX 2 - MAC.MT0 FILE 29

TABLE OF FIGURES

Figure 1: MAC Block Diagram 5
Figure 2 block diagram of the 8 X 8 multiplier using modified Booth algorithm.....7
Figure 3 Schematic of a 8 bit Booth Multiplier 8
Figure 4 : Table for Partial Product generation process.....8
Figure 5: Circuit to generate -1X,0X, and +1X 9
Figure 6: Circuit to implement +2X and -2X.....9
Figure 7: Block diagram of an Adder Array 10
Figure 8 Transistor level diagram of PP-MUX.....11
Figure 9: Transistor Level Diagram for PP FA.....11
Figure 10: Transistor level diagram and Layout of ADD cell 12
Figure 11: Ripple Carry Adder - Layout and block diagram 13
Figure 12Conditional Sum Adder block diagram 13
Figure 13 Accumulator Unit Layout.....14
Figure 14 BE -MUX Layout(left) and Mult2 layout (right) 16
Figure 15:1 Bit Adder with and without inverters 17
Figure 16: Transistor Diagram for 1 bit adder 18
Figure 17: Conditional Sum Layout.....20
Figure 18: MAC unit layout.....24
Figure 19: Final MAC unit Outputs.....25

ABSTRACT

This report describes the successful implementation of an 8 x 8 bit Multiplier and a 20 bit Accumulator (MAC) unit. The multiplier multiplies two 8 bit numbers – generating a 16 bit output. This output is added to a 20 bit number generating the final output.

The algorithm used to design the multiplier was Modified Booth Algorithm. 0.2 μm CMOS process parameters were used to perform SPICE simulations

The main objective of the project was to get a low power, high performance MAC unit that was modular and re-configurable.

PROJECT OBJECTIVE

The main objective of this project was to successfully implement the Multiplier and Accumulator unit keeping the constraints of low power operation and high performance.
To achieve the above mentioned goals, we had to take into considerations the parameters at different levels of designing, namely
1. Algorithmic level
2. Architectural level
3. Logic level
4. Circuit level

INTRODUCTION

High speed and low power MAC units are required for fast calculations in today's computing world. A fast and low power MAC unit is the requirement of various fields of science for example DSP applications.
In this project, we have designed and implemented an 8 X 8 bit multiplier unit and a 20 bit accumulator unit. The user enters three numbers X(8 bits), Y(8 bits) and A (20 bits) and a 20 bit output is obtained.
The block diagram of the design is shown next.

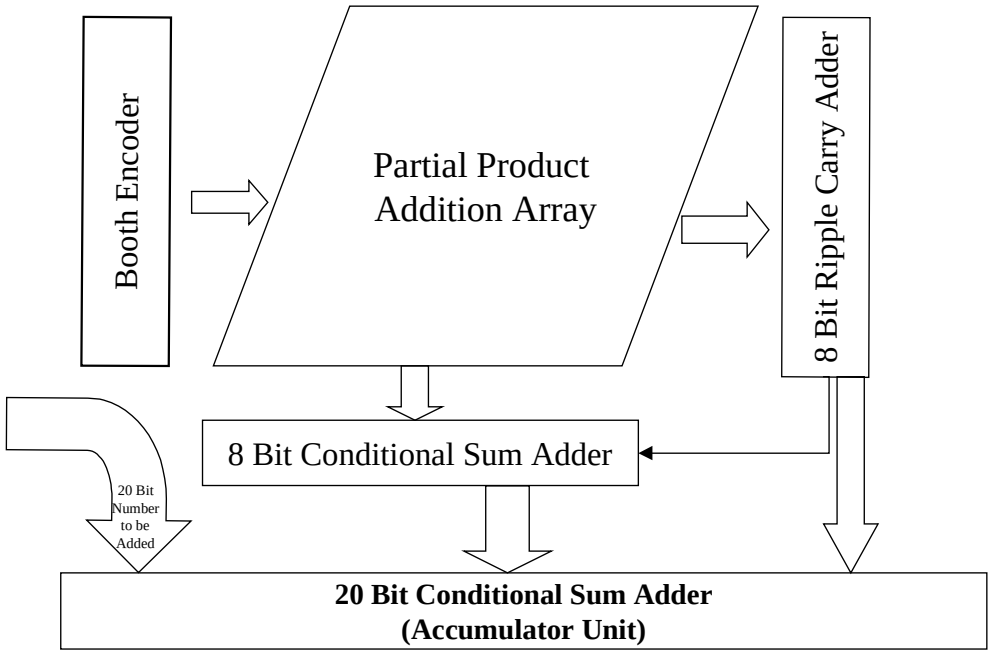


Figure 1: MAC Block Diagram

- **CHOICE OF ARCHITECTURE**

The choice of the multiplier algorithm to be used in this project was a crucial factor.
Few algorithms like Braun Multiplier, Baugh-Wooley Multiplier and Booth Multiplier were studied to decide on the multiplier to be used. Braun Multiplier does the multiplication the traditional way i.e generating the n partial products (for nXn multiplication) and then adding them to gain the product. The drawback was that it was primarily meant for unsigned number additions.
Baugh-Wooley multiplier represents the multiplier and multiplicand by two's complement form and does the multiplication. In this algorithm we can obtain the product for two signed numbers also.
The main disadvantage of the two algorithms that have been mentioned above is that the hardware required in both the algorithms was more when compare with the Booth Multiplier algorithm.
In Booth Multiplier, the multiplier bits are recoded to get only n/2 partial products from which we can get the final product.
We also had the option of choosing a tree based algorithm such as Wallace tree. Wallace tree is good for implementation when we have 16 bits or more numbers to be multiplied. As we were going for an 8 x 8 multiplier, Booth's algorithm was chosen as it was very modular and had less number of interconnects.
We implemented the multiplier using the modified Booth algorithm as our main objective was to have a fast and a modular multiplier. The booth algorithm is based on recoding the two's complement operand (i.e. the multiplier) in order to reduce the number of partial products to be added. This reduces the area as well as increases the speed of performance.

IMPLEMENTING THE MAC UNIT

The MAC unit consists of the multiplier and an accumulator unit. We first started implementing the multiplier, as it was the major portion of what we were planning to achieve.

- **MULTIPLIER**

The multiplier consisted of the
1) Booth encoder and Sign bits extension
2) Partial Product generator and Adder Array
3) One 2n bit adder or two n bit adders.

The block diagram of the 8 X 8 multiplier using modified Booth Algorithm is shown in figure

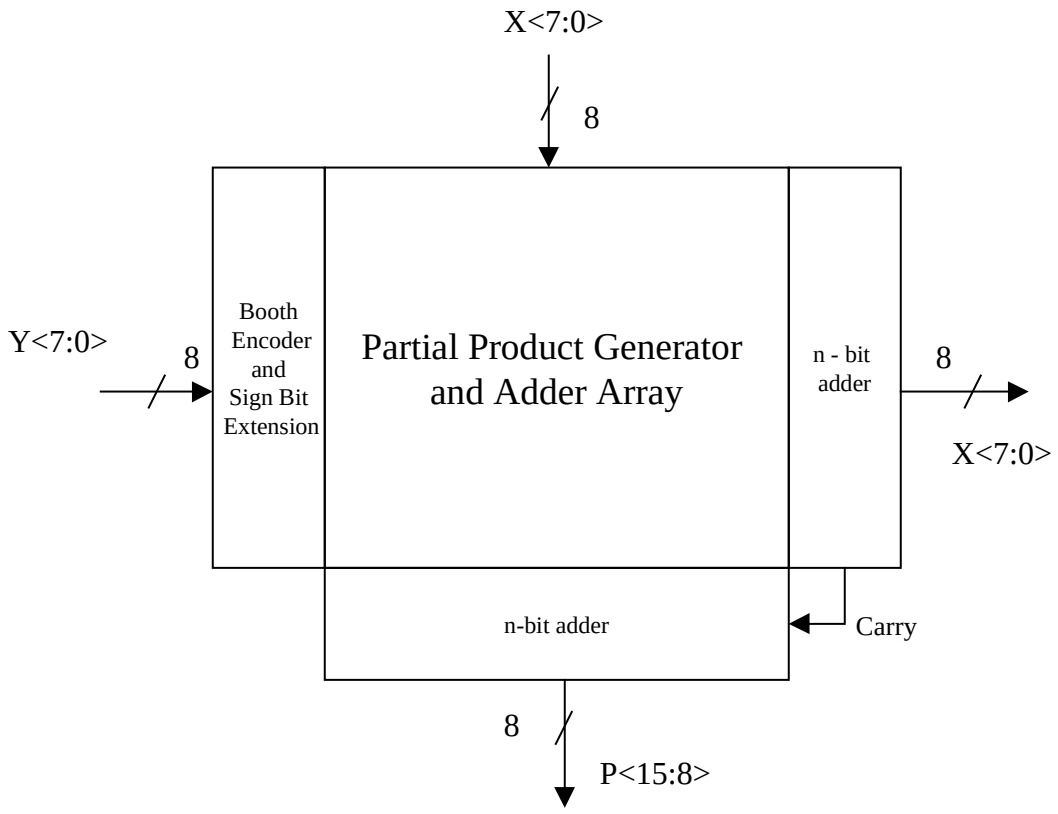


Figure 2 block diagram of the 8 X 8 multiplier using modified Booth algorithm

The schematic of the 8 bit booth multiplier is presented in the figure below

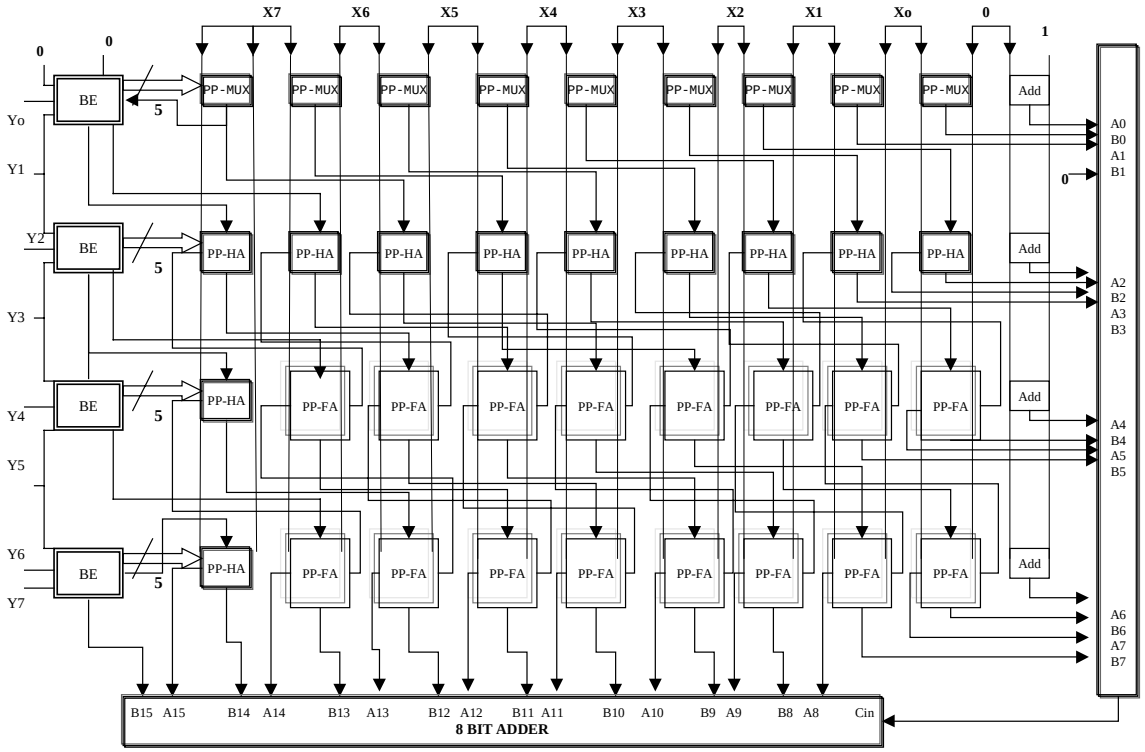


Figure 3 Schematic of a 8 bit Booth Multiplier

The basic blocks of the Multiplier are listed below.

Booth Encoder

The Booth encoder generates the five control signal for the entire multiplier unit from a group of three bits of the multiplier Y.
The algorithm recodes the multiplier (Y) in order to reduce the number of partial products to be added. The recoding of Y generates another number with the following

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Recoded digit	Operation on X	Explanation
0	0	0	0	0 x ***X***	Add 0 to PP
0	0	1	+1	+1 x ***X***	Add ***X*** to PP
0	1	0	+1	+1 x ***X***	Add ***X*** to PP
0	1	1	+2	+2 x ***X***	Shift left ***X*** one digit and add to PP
1	0	0	-2	-2 x ***X***	Add 2's Complement of ***X*** to PP and shift left one digit
1	0	1	-1	-1 x ***X***	Add 2's Complement of ***X*** to PP
1	1	0	-1	-1 x ***X***	Add 2's Complement of ***X*** to PP
1	1	1	0	0 x ***X***	Add 0 to PP

Figure 4 : Table for Partial Product generation process

five signed digits, -2, -1, 0 +1, +2. Each recoded digit in the multiplier performs a certain operation on the multiplicand (X).

The booth encoder was implemented using the following circuits

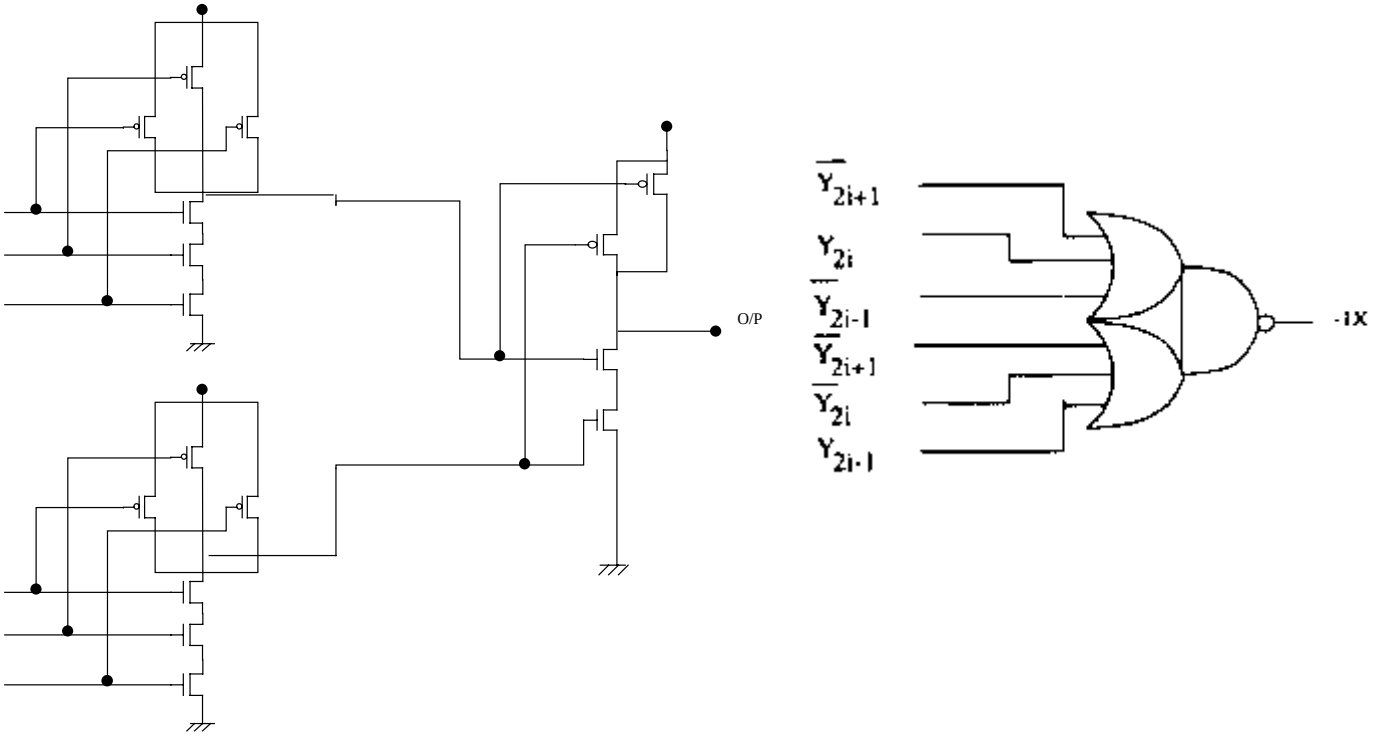
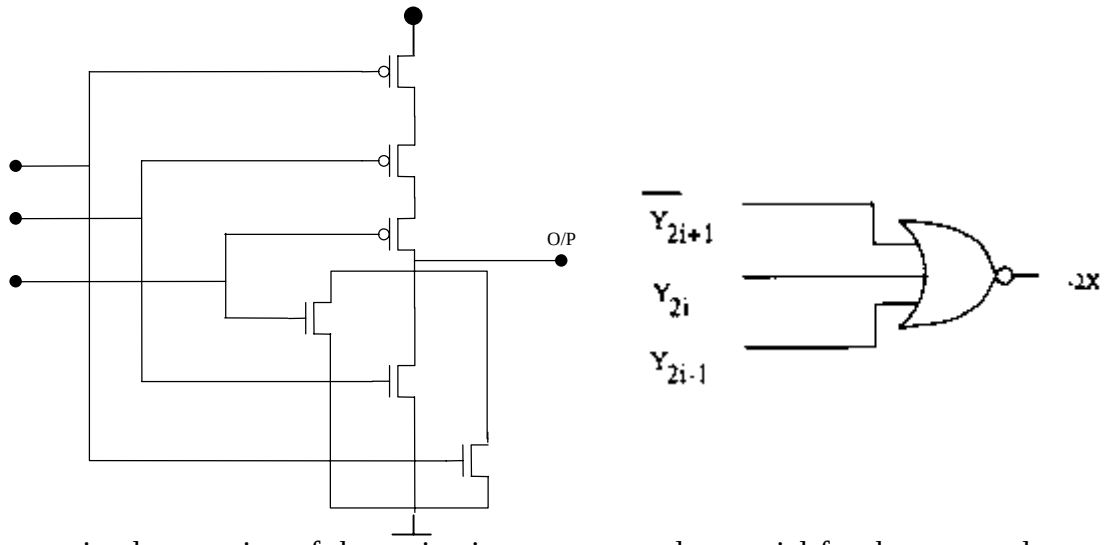


Figure 5: Circuit to generate -1X,0X, and +1X

Figure 6: Circuit to implement +2X and -2X



The correct implementation of these circuits was extremely cru cial for the expected operation of the entire multiplier as these circuits were instrumental in obtaining the partial products. The 5 control lines run through the 4 stages of our multiplier (one to generate each partial product)

Multiplier (Adder) Array

The multiplier array, that consist of multiplexers, half adders, full adders and the add cell (to add 0 or 1 to the LSB of the partial products)[1]. The carry save adder in the adder array is used here as it enables a very fast operation of the adding operation . Instead of rippling the carry from one end to the other in a partial product, it passes the carry diagonally to the block below the block that generated the carry. This ensures a comparatively very fast operation. The figure below illustrates the working of the carry save adder.

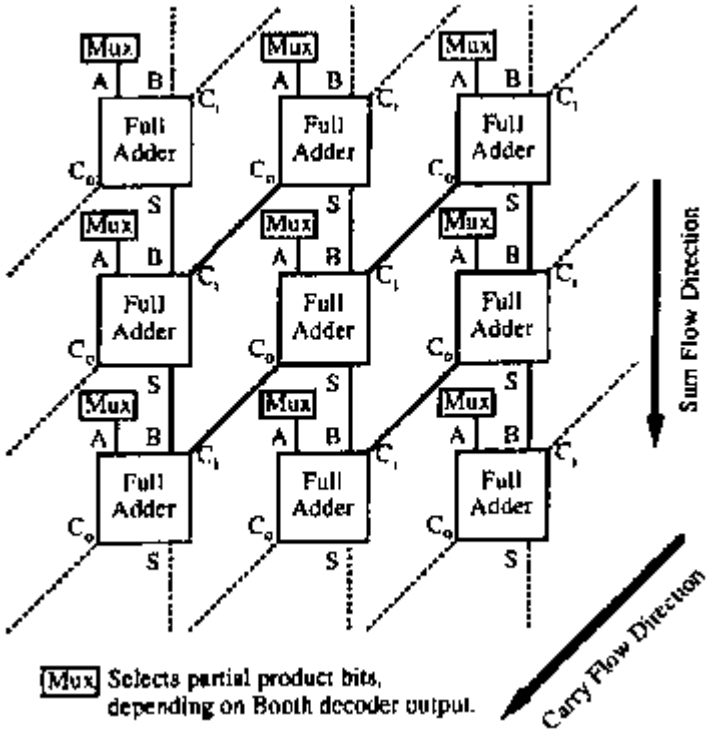


Figure 7: Block diagram of an Adder Array

The Multiplexer block, is discussed next

- *PP-MUX*

The first partial product is obtained with a row of multiplexers only as they do not have any other partial products to add to. The multiplexer choose between the i^{th} and the $i\text{-}1^{th}$ term of the multiplicand. This operation is to implement the 'left shift' that has to be performed when the control signals are +2 or –2.

[1] This is to implement the 2's complement. Whenever there is a need of implementing a 2's complement, the ADD cell adds a 1 to the LSB of the partial product else adds a 0.

The transistor level implementation of the PP-MUX is shown in the figure below.

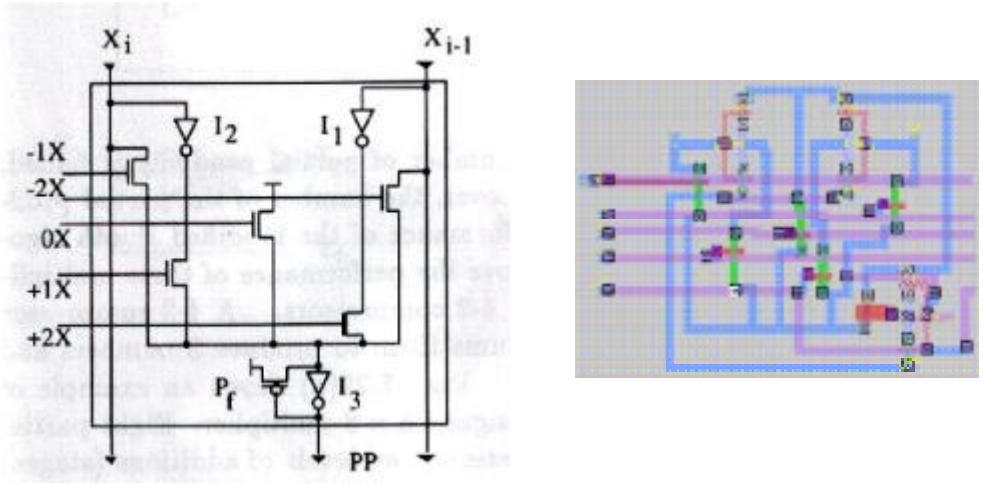


Figure 8 Transistor level diagram of PP-MUX

- *PP- FA and PP-HA*

The next level in the multiplier array is the 2nd partial product obtained as outputs from the half adder array. We use a half adder array as there are no carry ins from the previous array to be passed to the adders (as carry travel diagonally downwards.
After the half adder array, there are 2 levels of full adders to form the last 2 levels of the multiplier array. The transistor level diagrams of the Full Adders (and the Half Adders) are presented.

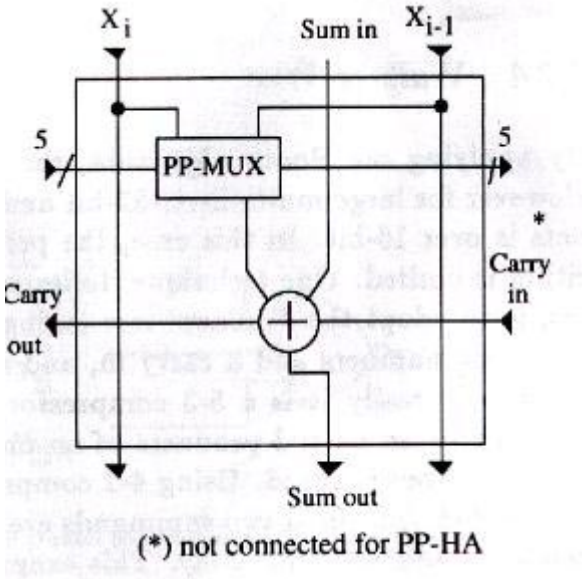


Figure 9: Transistor Level Diagram for PP FA

- *ADD Cell*

The Add cell in the multiplier array is the first column of cells (refer figure 3) these cells are needed to add a 0 or a 1 to the first bit of the partial product. This is required to implement the 2's complement to the partial product when the control signal is –1 or –2.

The transistor level diagram of the Add cell is shown along with the actual implementation of the ADD cell in magic.

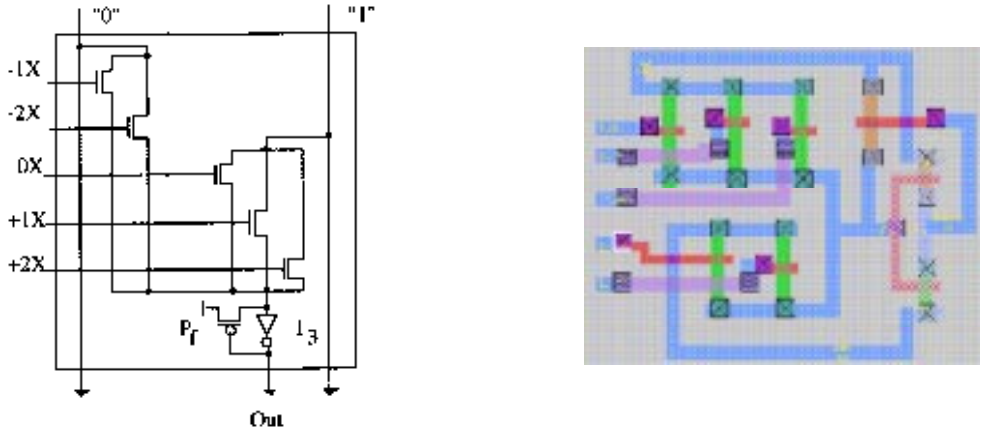


Figure 10: Transistor level diagram and Layout of ADD cell

Final Stage Adder

The final stage adder is to get the final output out of the multiplier unit by adding the partial products We have used 2 8 bit adders in the multiplier circuit. One is an 8 bit ripple carry adder while the other is an 8 bit conditional sum adder as discussed below

- *Ripple Carry Adder*

A ripple carry adder is used to sum the first 8 bits of the final output (which is 8x8=16 bits). We have used a ripple carry for this purpose as the 8 set of outputs that are going to the adder do not arrive at the same time (because they arrive from different levels of the partial product-refer figure 3). As such the use of a faster adder (for example conditional sum) would have been of no great advantage. The basic building block of the ripple carry adder

The block diagram of the ripple carry adder is shown in the figure below

The ripple carry adder is slow in speed but small in area. As we cannot help with speed with the first 8 outputs, we save on area in our design.

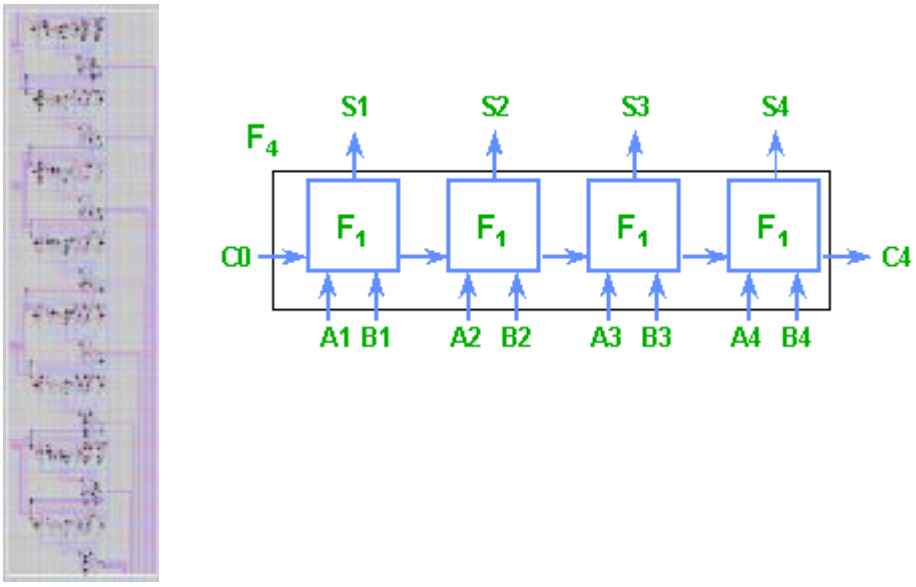


Figure 11: Ripple Carry Adder - Layout and block diagram

- *Conditional Sum*

The last 8 bits of the final output is obtained from a conditional sum adder. We use a conditional sum as all the 8 set of adder inputs arrive at the same time. This is because the 8 sets of inputs to the adder are coming from the last row of full adders – refer figure 3. We could have used a carry look-ahead adder for this part of our implementation, but the fact that the area of that adder is very large, we refrain from using it.

Our basic conditional block is a 4 bit adder as shown in figure 12. We use this 4 bit adder twice to implement an 8 bit adder.

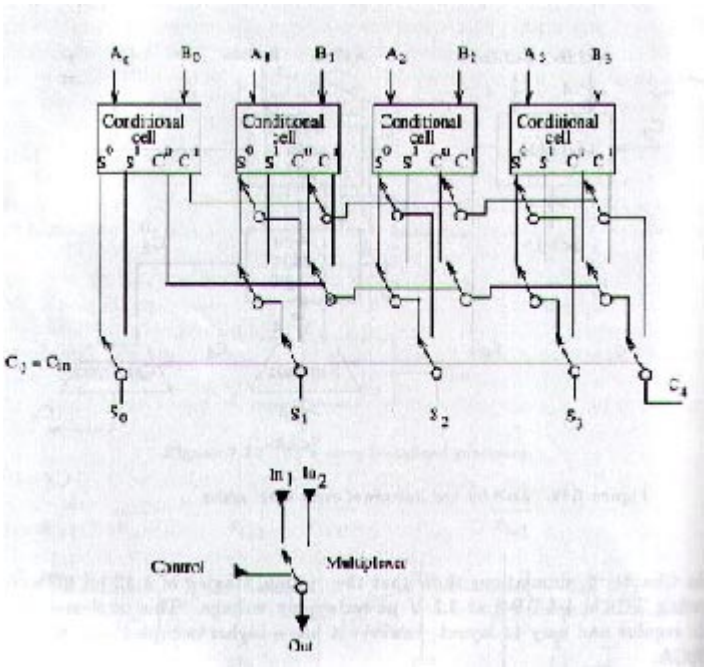


Figure 12Conditional Sum Adder block diagram

- **ACCUMULATOR**

Till now we had been discussing about the multiplier unit of the MAC unit. the accumulator unit of the MAC unit is simply a 20 bit adder that adds a user supplied 20 bit number to the multiplied 16 bit number, thus giving a 20 bit output.
The accumulator was implemented by cascading 5 of the 4 bit conditional sum adders.

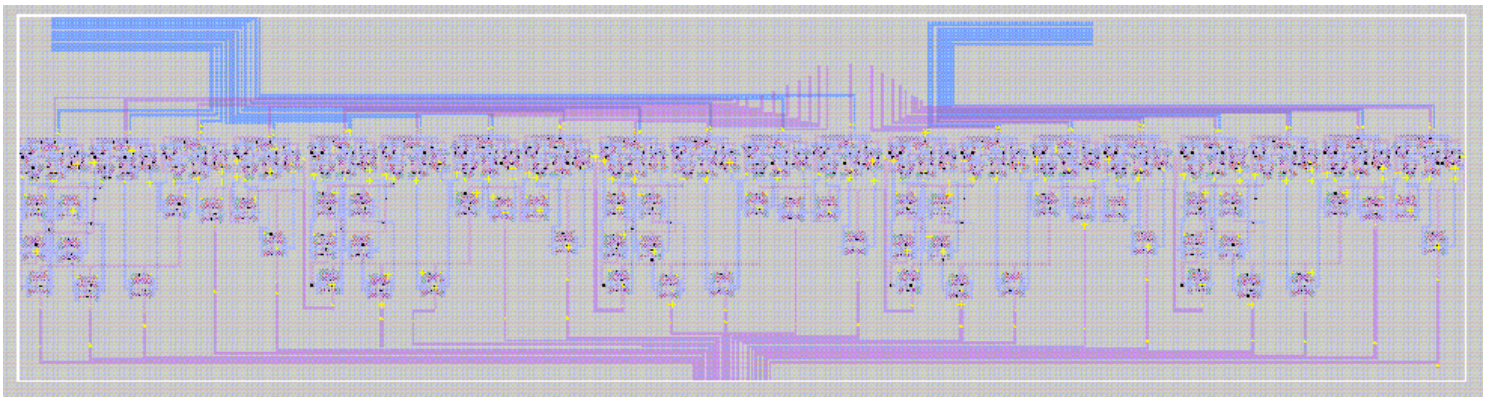


Figure 13 Accumulator Unit Layout

PERFORMANCE OF THE MAC UNIT.

The performance of the MAC unit is measured in terms of power, area and time delay. At each stage of design, the corresponding delay and power consumption was measured

- **PERFORMANCE OF THE MULTIPLIER**

The performance of the multiplier can be assessed by measuring the power and delay of the different components of the Multiplier. We started with the booth encoder block.

Booth Encoder

- *Generation of booth encoded digits:*

The initial cut down in area and gain in speed for our booth encoder was done at the logical level. The circuit initially thought was the logical diagram in figure5. But in CMOS the speed of a NAND gate is better than an OR gate and also the area is less. Hence instead of logically getting the output as shown in the fig5 as above, the transistor circuit of fig 5 was employed by inverting the inputs that are used for logical diagram.
Example: For getting the output of "-1X" the logic used was

~{ ((~Y2i+1)+(~Y2i)+(Y2i-1)) *((~Y2i+1)+(Y2i)+(~Y2i-1)) }

The realization of the above function requires two OR and one NAND gate.
But the same value of "-1X" can be achieved by using the logic function:

~{ ((Y2i+1)*~(Y2i)*(y2i-1))*((Y2i+1)*(Y2i)*~(Y2i-1)) }

This realization uses three NAND gates.
The complete booth decoder is as shown in fig5 and fig6.

Specifications: Delay: 0.26ns Power: 0.38 milliwatts

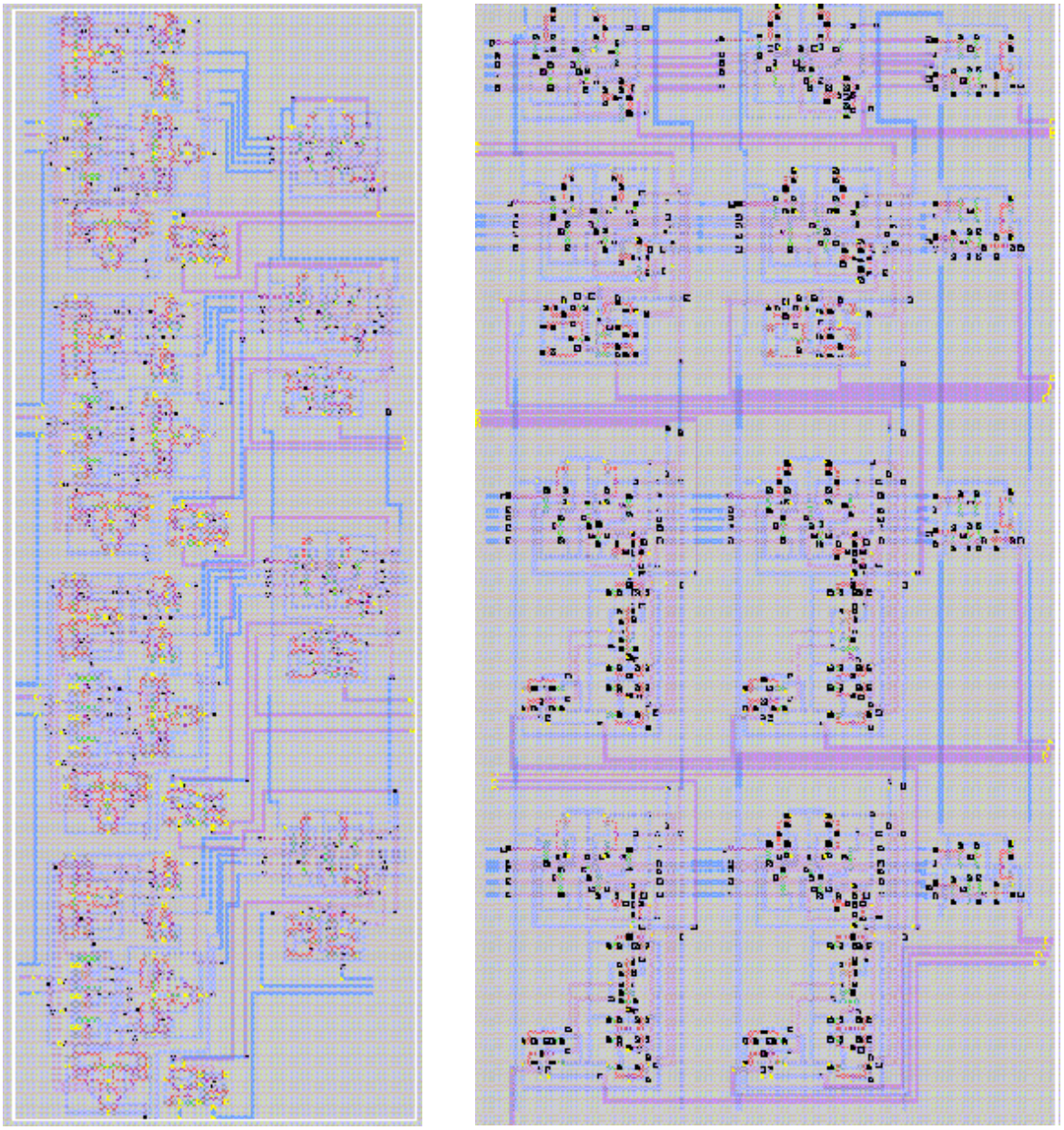
- *(b)Generation of sign extension bits;*

The sign extension bits were included in the booth encoder block for maintaining modularity.

Specifications: Time delay:0.06ns Power: 0.17 milliwatts

This block receives its input from the MUX block in each partial product generation stage (refer the block diagram). The output of this block then feeds the next partial product generation stage for the calculation of the sign bits.

Figure 14 BE -MUX Layout(left) and Mult2 layout (right)



In the actual layout, we connected an other stage of MUX and HA cells to the booth encoder and named it BE-MUX cell. The power and time delay of this combined cell was measured.

Specifications: Time delay: 1.998ns Power: 1.335 milliwatts

Thus the delay for this block was. 1.998ns. It is after this delay in time that the booth encoded digits along with the sign extension bits reached the adder array

- *(c)Transistor sizing:*

Since the booth encoded digits are routed throughout the units, high driving capability was required or the booth encoded lines. We hence, kept the sizes of the final transistor driving the lines having Width:Length ratio as 4:2. to ensure good quality of the signal.

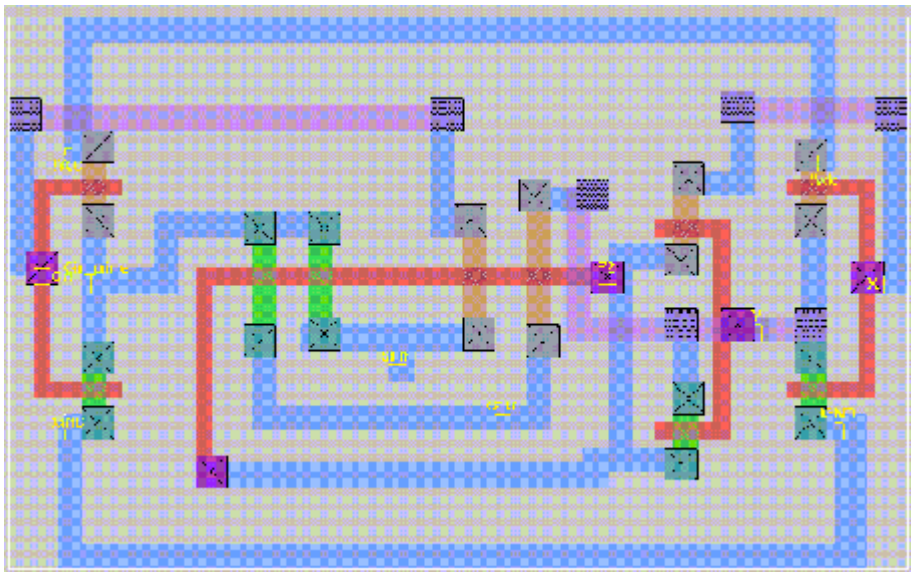
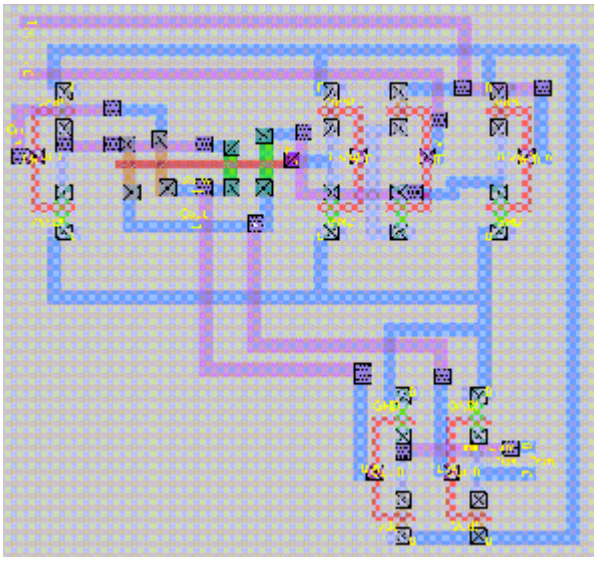


Figure 15:1 Bit Adder with and without inverters



Adder Array

The heart of our adder array is the 1 bit adder and the MUX unit.

- *(a)Designing the MUX unit;*

The basic circuit diagram for this block is as in fig.8. Since we use pass transistor logic for these MUX cells, we get some degradation in the strength of the signal. But since the output is always taken through an inverter, along with a weak feedback PMOS transistor, proper quality of the signal is ensured. Also, along with the elimination of any DC current, we get fast operation and low power consumption.

Specification: Time delay:0.32ns Power:9.7 microwatts

- *(b)Designing the 1 BIT ADDER cell:*

The main block that decides the speed of this array is the adder cell. A good choice of an adder is the one that uses less transistors and hence increases speed and reduces area.

The initial choice was the 10-transistor adder cell (from now on referred as 10Tcell for brevity). But implementing the adder and checking for its output showed the main problem. The output was degraded and was practically not usable. Hence to get a stable and perfect adder output, three inverters were added to buffer the output so that it could be routed safely throughout the adder array.Figure 15 shows the layout for the

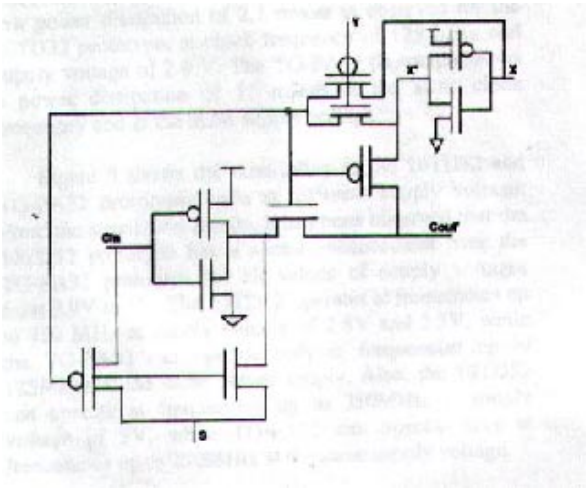


Figure 16: Transistor Diagram for 1 bit adder

1 bit adder cell with and without the inverter buffers

Specifications: Time delay:0.03ns Power: 84 microwatts

By increasing the number of transistors in the adder cell, the main objective of using the 10Tcell was lost, even then we decided to use this adder in our multiplier unit to investigate its working.

- *(c)Designing the half adder cell;*

It was felt that a better design for our half adder, like used in 10Tcell, which had lesser power and area with high speed, would add greatly to the performance of the adder array.

Specifications: Time delay:0.16ns Power:20.56 microwatts

- *(d)Designing the ADD cell:*

The ADD cell is implemented for generating the 0 or 1. This cell is implemented as shown in fig9. Again, we require a weak feedback transistor across the inverter for fast and low power operation.

Specifications:
Time delay:0.288ns
Power: 7.751 microwatts.

- *(e)Performance of MULT2 and MULT6:*

The basic blocks are connected using the carry save addition as explained above. To get a clear picture of the delay and the power consumption in the adder array and to make our design modular and reconfigurable, we splitted the adder array in two blocks namely MULT2 and MULT6.The MULT6 block and MULT2 blocks were tested individually. Lsyout of Mult2 is shown in Figure 14
The specifications as well as the output waveforms were noted.

MULT6
Specifications;
Time delay:2.91ns
Power:1.58 milliwatts

MULT2
Specifications;
Time delay;1.31ns
Power:0.62 milliwatts

Ripple Carry Adder

The choice of adder at this stage was to be decided based on our initial objective of low power and high performance.
In terms of high speed, the conditional adder was the best choice and carry look-ahead adder could also be used. For low power too, conditional sum could be better but as discussed before, at this stage all the inputs are not available at the same time. Thus the basic advantage of using conditional adder at this stage would be lost. Also there is certain delay between the arrival of inputs to the adder, hence thought of using a ripple carry adder so that by the time the second stage inputs are available, the rippled carry is available to the adder . the layout and block diagram of the ripple carry adder is shown in figure 10.
Since we were uncertain about the power consumption by an 8bit conditional sum and an 8bit ripple carry adder implemented using 10T cell, we calculated the power for each adders and compared.
We observed that the power for 4 bit conditional sum was 0.30 milliwatts. Hence, we could assume that the power for an 8bit conditional sum would be in the range of 0.60 milliwatts. The ripple adder had a power consumption of 0.998 milliwatts.
We could have saved almost 0.40milliwatts power for our design but the area would have increased. As the use of conditional sum is giving us the benefic of only low power but no gain in speed, we traded off 0.40 milliwatts power for an efficient area design. The use of low power, high-speed 10T-adder cell for implementing the ripple carry also supplemented our choice

Specifications:
Time delay:3.80ns
Power:0.9968 milliwatts

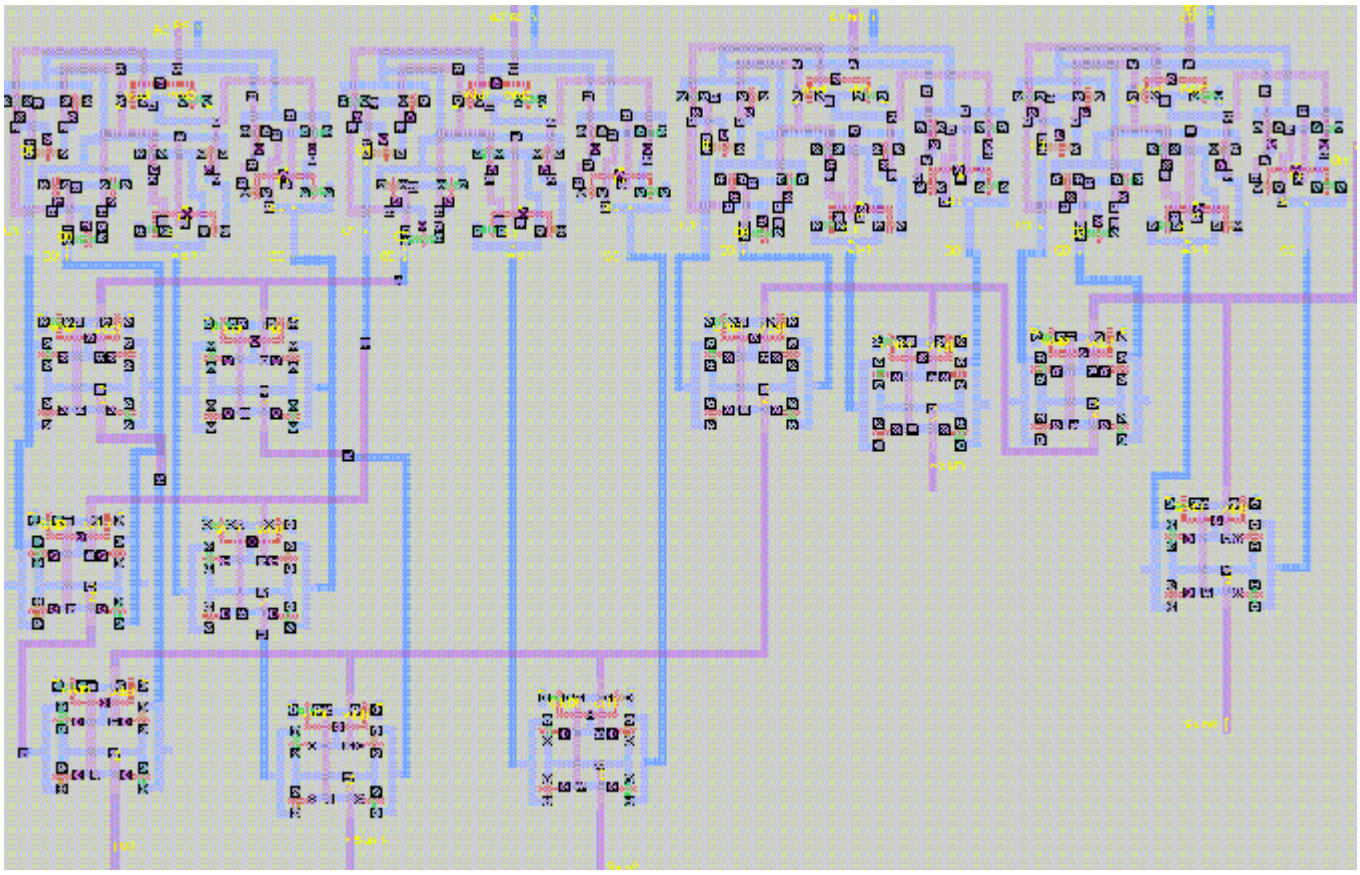


Figure 17: Conditional Sum Layout

Conditional Sum Adder

This adder is actually the bottleneck of the MAC unit. This is because unless the addition is done in this adder, the outputs can't proceed to the accumulator section for addition. Moreover this adder needs to be a ripple carry adder because the bits are to be added with the consideration of carry generated from the ripple carry adder. But after observing the working closely, we found that it is not needed that the entire adder be ripple carry adder.

At this stage all the inputs are available at once and it is reasonable to use a conditional sum adder here to gain speed. But the objective of high performance which comprises of speed and less area was to considered, we used 4bit conditional sum adder (figure 17) as the building block and then used ripple carry to get the 8 bit adder.

The power for this adder was 0.36 mlliwatts which is much lesser than an 8 bit ripple carry adder and the delay is 1.36ns which is again less compared to an 8 bit ripple carry adder. Although the delay is more than it would be in a complete 8 bit conditional adder, but here we do trade off between a complete ripple adder and a complete conditional adder to gain descent speed, low power and less area for the MAC unit.

Specifications: Time delay: 1.36ns Power:0.36 milliwatts

- **PERFORMANCE OF THE ACCUMULATOR**

The accumulator performance can greatly increase the performance of the MAC unit. But the bottleneck for the accumulator is the multiplier unit. Th accumulator must wait till it has correct logic values at its input for accumulation.
The accumulator unit must comply with our initial objective of low power and high performance. For these reasons ,we used again our 4 bit conditional cell as the building block and connected five of these blocks as a 20 bit ripple carry adder.By doing this we combine the less area advantage of the ripple carry adder and gain low power and speed through the conditional sum block. Though sub optimum

Specifications: Time delay:4ns Power:6.325 milliwatts

The accumulator is shown in figure 13

RESULTS OBTAINED

This table below summarizes the performance of the MAC unit. It provides the values of time delay and power consumption for the basic blocks of the circuit. Finally, the value for the complete MAC UNIT is also provided.
Few inferences about the basic blocks could be made after calculating the delay and power values.
1.The 1bit adder constructed using the 10T-cell improved the speed of the adder , even bettering the speed of an half adder.
2.The MUX cell is used extensively in the multiplier unit i.e 36 times. Hence if a better and faster MUX can be constructed, significant rise I speed can be achieved.
3. The half adder unit is used 11 times in the Multiplier unit and similarly the adder unit is employed 24 times. Hence, speed and power of the multiplier are directly affected due to the selection of half and full adders
4. The complete delay for the MAC unit is 8.69ns. Hence the correct values are available only after this time delay. This implies that the input signals to the MAC must change slow enough that before the inputs change again, the outputs have reached a correct stable logic state.
Thus the maximum frequency of the inputs is 25MHZ if the correct output values are required for .a period of 11ns. We can increase the frequency till 33MHZ, but then the correct values shall be available only for 7ns.

BLOCK	TIME DELAY(ns)	POWER CONSUMPTION
1-BIT ADDER	0.03ns	84 microwatts
HALF ADDER	0.16ns	20.56 microwatts
PP-MUX	0.32ns	9.7 microwatts
PP-HA	0.99ns	0.15 milliwatts
PP-FA	0.74ns	0.1629 milliwatts
ADD	0.288ns	7.751 microwatts
BOOTH ENCODER	0.26ns	0.3829 milliwatts
SIGN BIT EXTENSION	0.06ns	0.173 milliwatts
BE-MUX	1.998ns	1.335 milliwatts
MULT6	2.91ns	1.58 milliwatts
MULT2	1.31ns	0.62 milliwatts
RIPPLE	3.80ns	0.9968 milliwatts
4 BIT COND.SUM ADDER	0.64ns	0.30 milliwatts
8 BIT COND.SUM ADDER	1.36ns	0.36 milliwatts
20 BIT COND.SUM ADDER	4ns	1.638 milliwatts
MAC UNIT	**8.69ns**	**6.325 milliwatts**

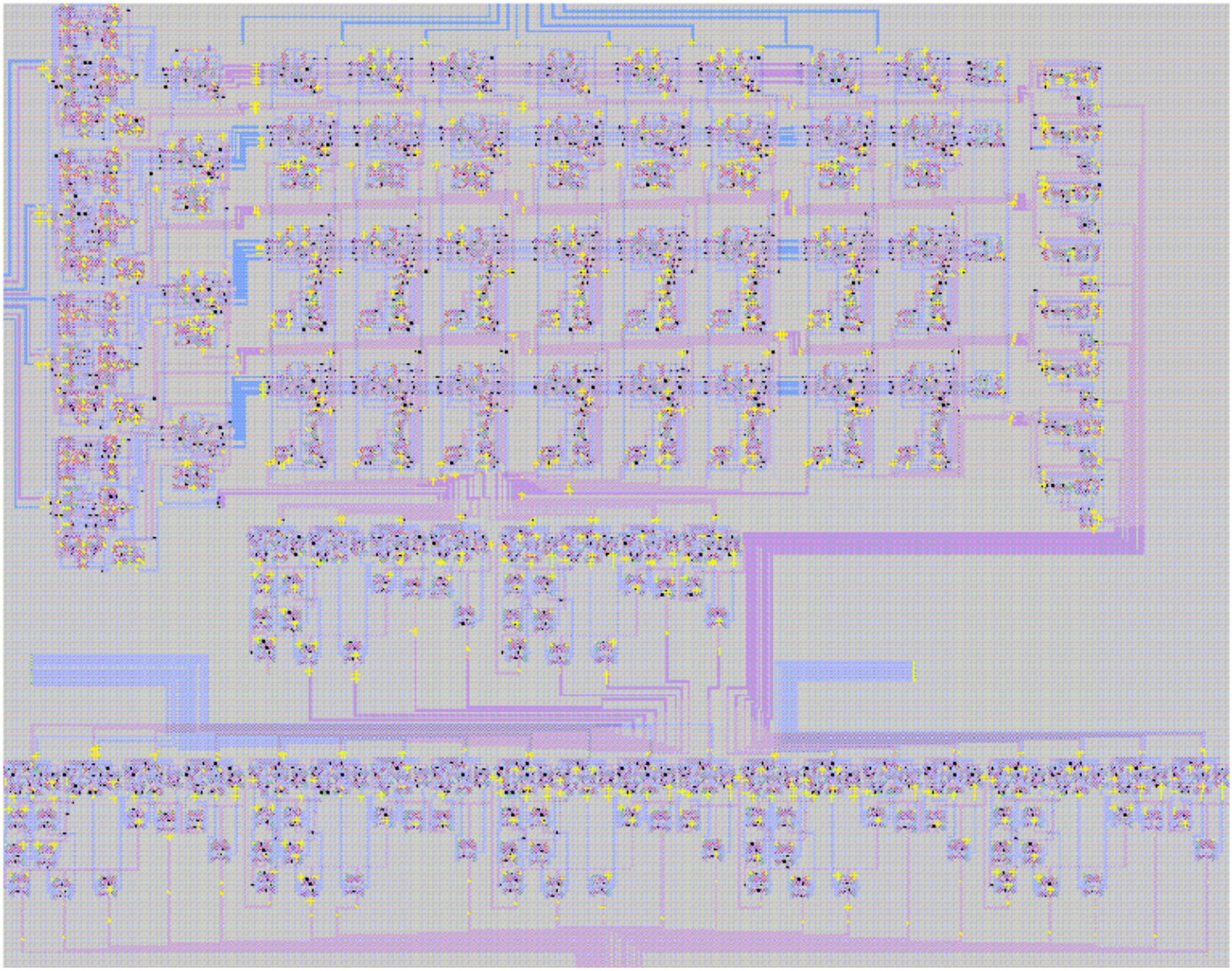


Figure 18: MAC unit layout

The above circuit is the complete MAC layout that was completed by our group in this project.

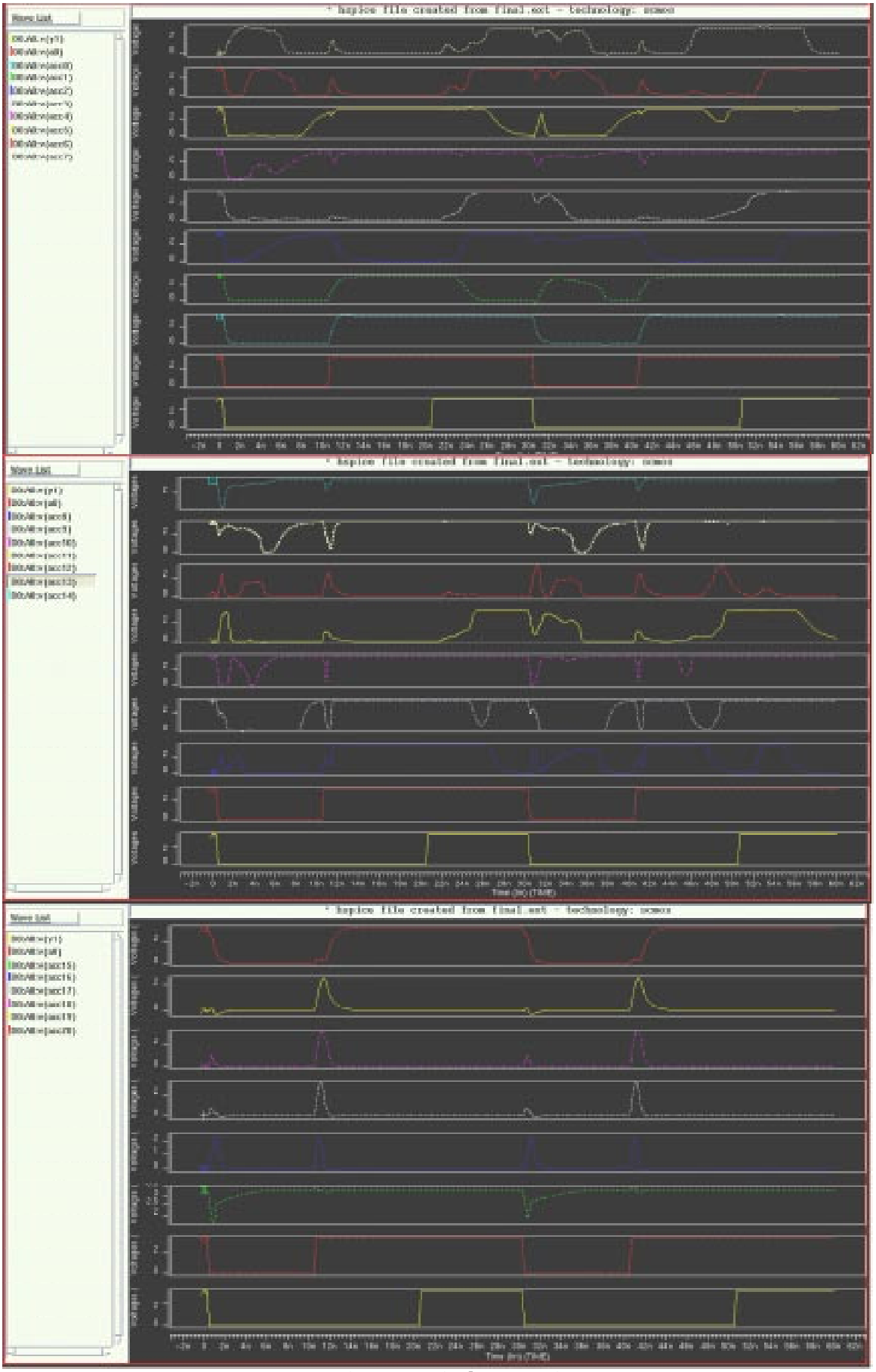


Figure 19: Final MAC unit Outputs

FUTURE WORK

Thus if the interconnects would have being lessened, we would have definitely reduced power consumption for our MAC unit.
It is still possible to achieve a much faster MAC unit if the objective of low power is kept as traded. Also, if the adder employed could be more efficient than the one used, better MAC unit could be designed.
There is plenty of scope of improvement in the accumulator unit. Our MAC unit requires a 20 inputs other than the multiplied result. For the Mac unit to be practically useful, a real accumulator that stores the result would really enhance the usability of the MAC.
Careful organization during the layout phase can decrease the power dissipation and increase speed. Also, a MAC unit for 16bit could be implemented as

DISCUSSION AND CONCLUSION

The entire MAC unit was completed and is working for any two numbers multiplication and the corresponding adding with an other 20bit number.
The main objective of getting a low power, high performance MAC unit was accomplished. **The power consumption of the MAC unit was found to be 6.325 milliwatts and the delay of the entire circuit was 8.69ns**.
The copy of the spice file is attached which was used for the circuit simulation.
Since it was necessary to determine the power consumption due to interconnections, after calculation power of the entire MAC unit, the spice file was edited. All the capacitances were removed and the circuit was simulated.
The power achieved was 2.099 milliwatts.
This power was subtracted from the total power and the power consumed due to interconnected was calculated to be as 6.32 milliwatts.
Observing the final waveforms it is found that the circuit has a lot of transients.
The reason for them is
1.the glitches introduced in the booth encoder
2.the delay that is present in the ripple carry adders
3.the delay that is present in the adder array.
It is possible to reduce these delays and there by reduce the complete delay time of the MAC unit if accurate layout and some synchronizing technique is used,
The summary of the MAC characteristics is

Technology	CMOS
Number of metal layers	2
Power supply voltage	3.3 volts
Operating frequency	25MHZ (can be exceeded till 33MHZ)
Power consumed at 25MHZ	6.325 milliwatts
Transistor count	2.2K
Time delay	8.69ns

LIST OF REFERENCES AND BIBLIOGRAPHY

1. Low-Power digital VLSI Design circuits and systems
   Kluwer Academic Publishers, June 1995
   Author:Elmasry, Mohamed I. /Bellaour, Abdellatif

2. Principles of CMOS VLSI Design
   A Systems Perspective
   Neil H. E Weste, Kamran Eshragian

3. . Multiplexer Based Array Multipliers
   Kiamal Z Pekmestzi
   IEEE Transactions on computers, Vol 48, No 1; January 1999

4. A 10-Transistor Low-Power High-Speed Full Adder Cell
   Hanan A. Mahmoud and Dr.Magdy Bayoumi
   University of louisiana at Lafayette
5. A swing Restored Pass-Transistor Logic-Based Multiply and Accumulate circuit for Multimedia Applications
   Akhilesh Parameswar, Hiroyuki Hara and Takayasu Sakurai
   IEEE Journal Of Solid State Circuits, Vol 31, No 6, June 1996
6. . General Data-Path Organization of a MAC unit for VLSI Implementation of DSP Processors
   Aamir A. Farooqui, Vojin G. Oklobdzija
   University of California, Davis, Integration Berkeley, California
7. Algorithmic Power Considerationd in Iterative Multipliers
   Ian O'Donnell, Dennis Yee
8. http://www.ecs.umass.edu/ece/koren/arith/simulator/Modbooth
   Modified Booth Simulator

APPENDIX 1-SPICE INPUTS FOR MAC.MAG

```
.model nfet NMOS level=13
.model pfet PMOS level=13
vdd Vdd 0 dc 3.3v
.meas tran sd2  trig at=4.1ns   targ v(Acc12) td=9.09ns  val='1.65' cross=1
.meas tran average-power avg power
vin1 Y_1 0 dc 0v
vin2 Y0 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin3 Y1 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin4 Y2 0 pulse(0v 3.3v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin5 Y3 0 pulse(0v 3.3v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin6 Y4 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin7 Y5 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin8 Y6 0 pulse(0v 3.3v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin9 Y7 0 pulse(0v 3.3v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin10 X7 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin11 X6 0 pulse(0v 3.3v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin12 X5 0 pulse(0v 3.3v0.3ns 0.2ns 0.2ns 20ns 30ns)
vin13 X4 0 pulse(0v 3.3v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin14 X3 0 pulse(0v 3.3v0.3ns 0.2ns 0.2ns 20ns 30ns)
vin15 X2 0 pulse(0v 3.3v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin16 X1 0 pulse(0v 3.3v0.3ns 0.2ns 0.2ns 20ns 30ns)
vin17 X0 0 pulse(0v 3.3v 0.3ns 0.2ns 0.2ns 20ns 30ns)
vin18 Fj 0 dc 0v
vin19 A0 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin20 A1 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin21 A2 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin22 A3 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin23 A4 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin24 A5 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin25 A6 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin26 A7 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin27 A8 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin28 A9 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin29 A10 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin30 A11 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin31 A12 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin32 A13 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin33 A14 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin34 A15 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin35 A16 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin36 A17 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin37 A18 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin38 A19 0 pulse(3.3v 0v 0.3ns 0.2ns 0.2ns 10ns 30ns)
vin39 x8/carry_in 0 dc 0v
.tran 3ns 60ns
.option post
.end
```

APPENDIX 2 - MAC.MT0 FILE

```
$DATA1 SOURCE='HSPICE' VERSION='98.2 '
.TITLE '* hspice file created from final.ext - technology: scmos'
 sd2 average-power temper alter#
  8.6901e-09  6.325e-03   25.0000     1.0000
```