
Faculty of Engineering
Cairo University

Cairo University

# Design A Binary Multiplier / Divider Array

*Submitted by: Mohamed Sayed*

*Under Supervision of: Prof.Dr/ Serag Habib*

# TABLE OF CONTENTS



# LIST OF FIGURES

# LIST OF TABLES

# ABSTRACT

Arithmetic Logic Operations (Process) is Very Important Functions in Digital Domain Processing and Any Processor requires Arithmetic Block to be able to do different functions and Boolean Expressions, so one of the most important Operations that are wide used in Digital Domain is Division and Multiplication Blocks. We will be able to design Multiplication and Division Block by Using a certain Sequence of Adder and Subtraction Cells. Using Gaijeski Y-chart, starting from the Algorithm on the architecture level and to verify the functionality we used the RTL language (Verilog code) and see the results on XILINX ISE simulator tool, then we moved to logic level when we used the standard cells from CUSCLIB library cells. Finally we go the layout on the circuit level.

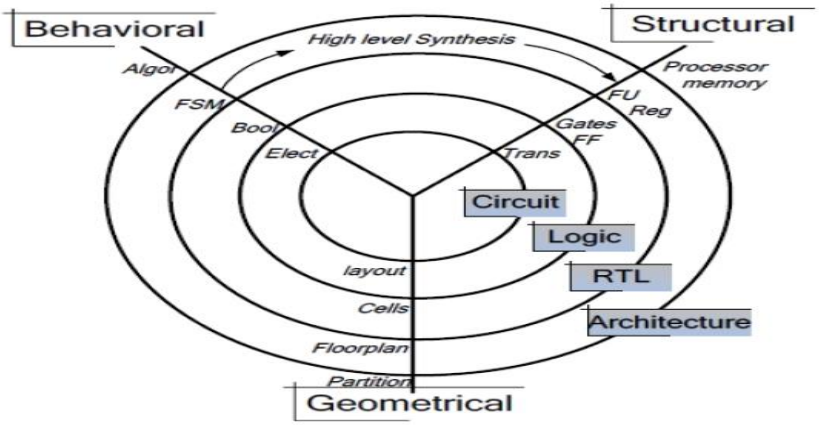


Figure 1: Gaijeski Y-chart

# Project Statement

It is required to design a binary multiplier / divider array using the SCN3M process.

- For division : dividend A 8-bit unsigned integer, the divisor B 4-bit unsigned integer, quotient Q 8-bit unsigned integer and remainder R be 4-bit unsigned integer.
- For multiplication: Both the two inputs are 4-bits unsigned integers and the output is 8-bits unsigned integers.

## Architecture level- algorithm and Schematic Procedure

Using UAC cells and control cells from REF2 shown in figures 2,3 as building blocks to design the binary multiplier / divider array , The technique used in the IAD is best understood to implement the non-restoring division algorithm for division and the repeated addition with save carry for multiplication.

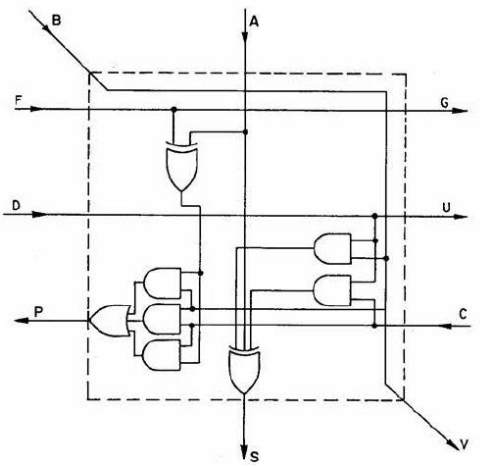


Figure 3: UAC Cell

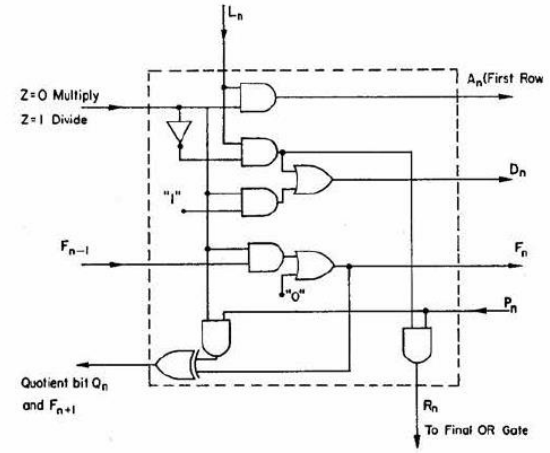


Figure 2 : Control Cell

We made some modifications to keep the functionality unviolated and to decrease the area:

- We removed Rn signal from control cell as we won't use it in our structure as it's used to get the MSB of 12-bit output and our design is 8 bit output only.
- We added row of 4 error detection cells (UAC) for remainder, to get correct results as show in the schematic in figure 4.

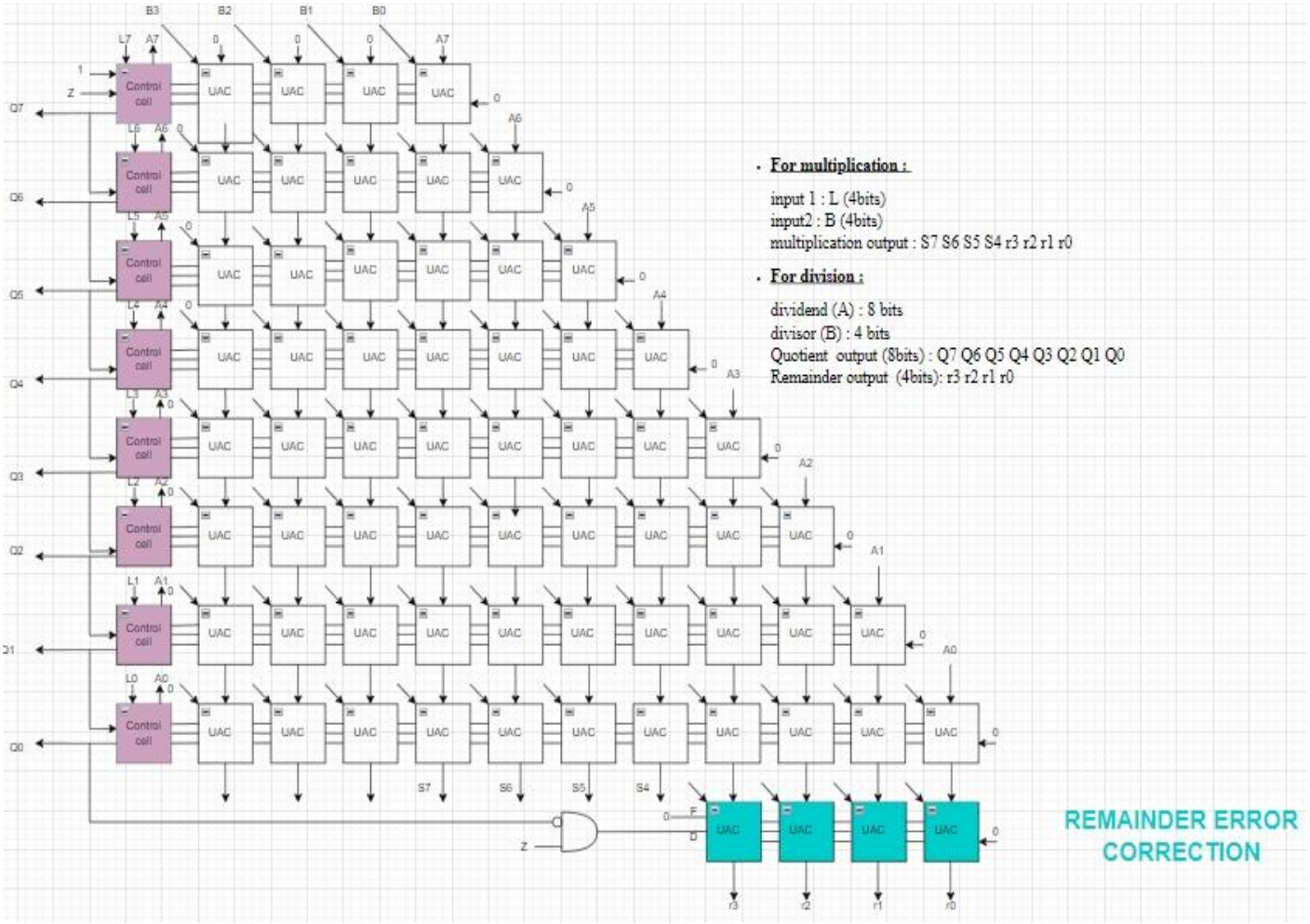


**Figure 4: Design schematic**

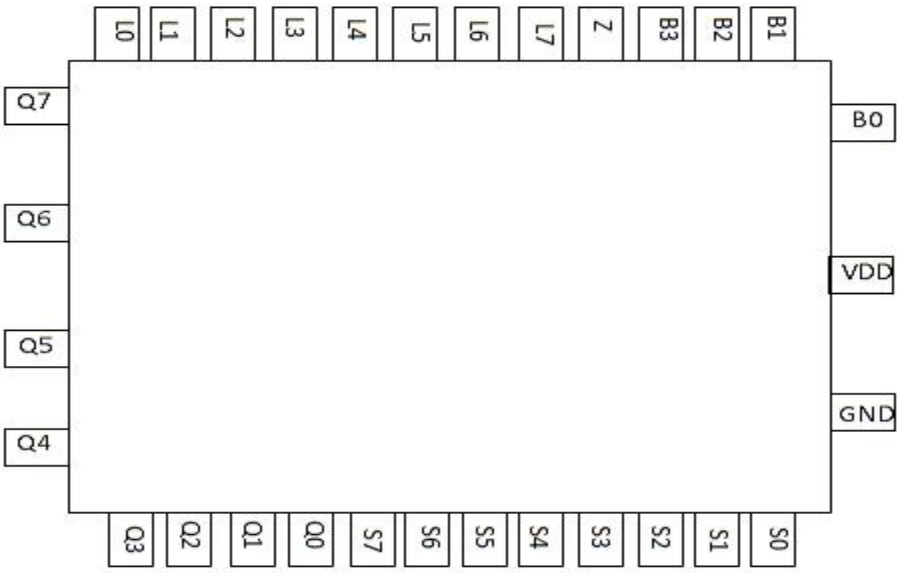


**Figure 5: Design black box using PADS**

# Logic Verification

Using XILINX ISE Simulator and Verilog RTL language to get accurate results.

## Corner cases and some random cases for division:

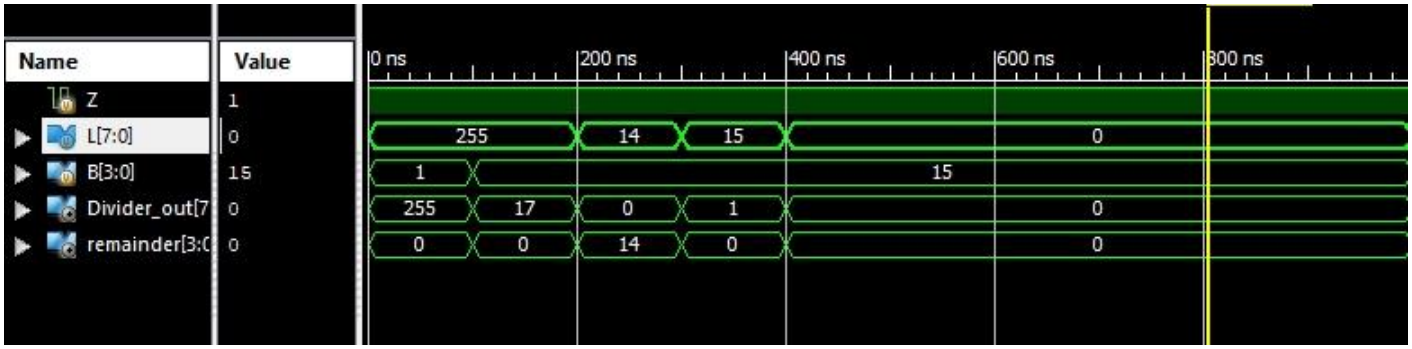


Figure 6: Logic verification Division results

## Corner cases and some random cases for multiplication:


Figure 7: Logic verification Multiplication results

# Design Layout

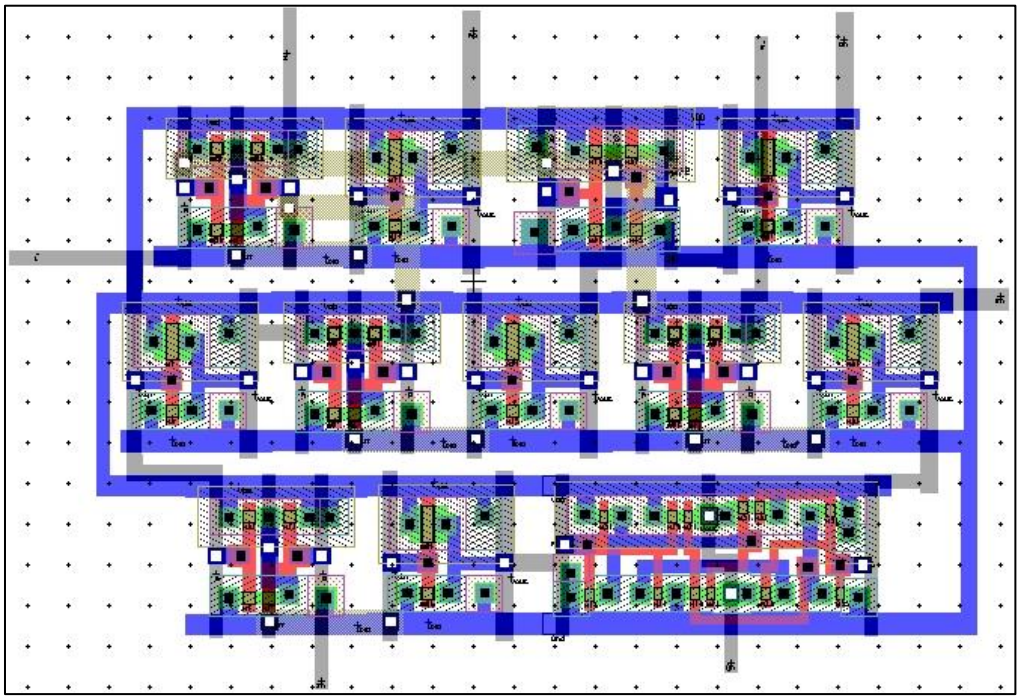

Figure 8: Control Cell layout

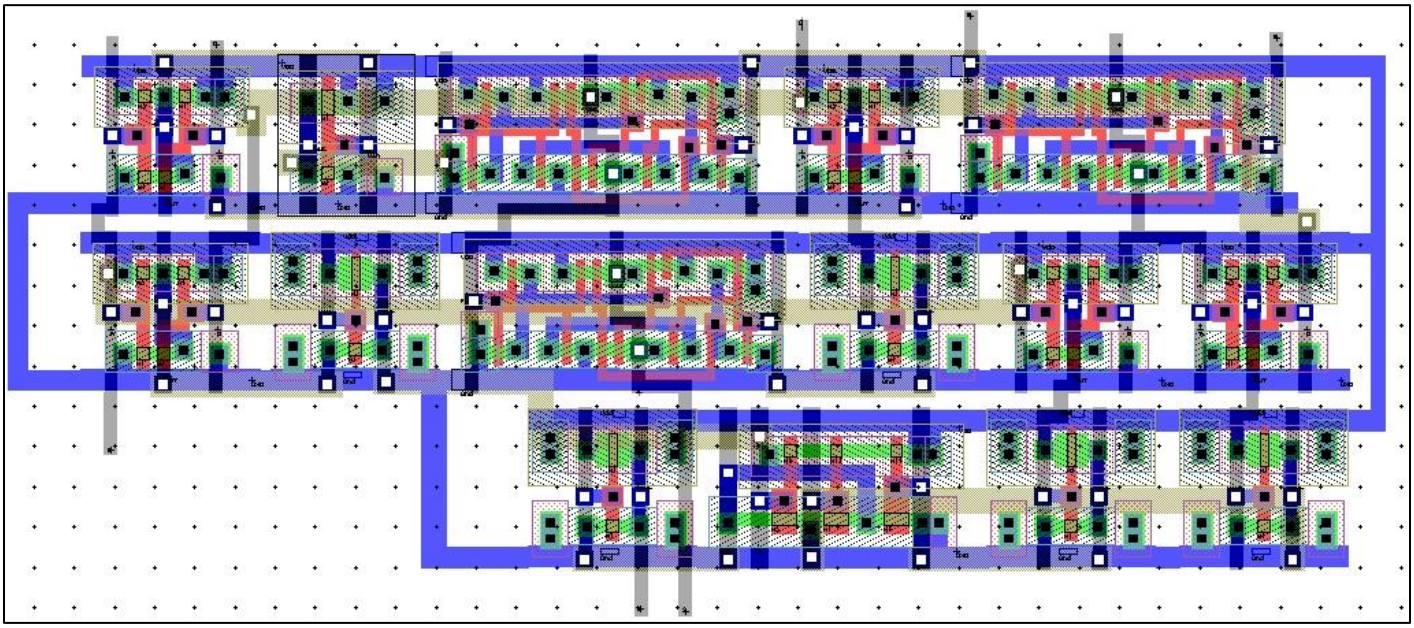

Figure 9: UAC Cell Layout

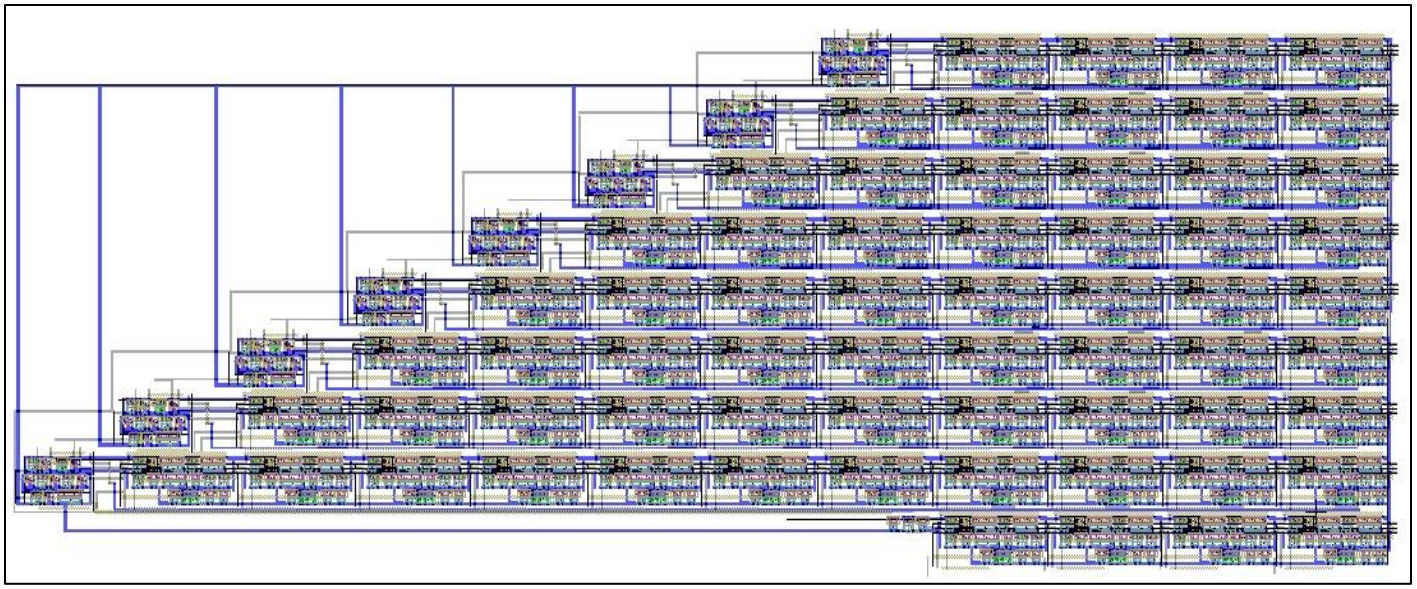

**Figure 10: Full design layout without PADs**

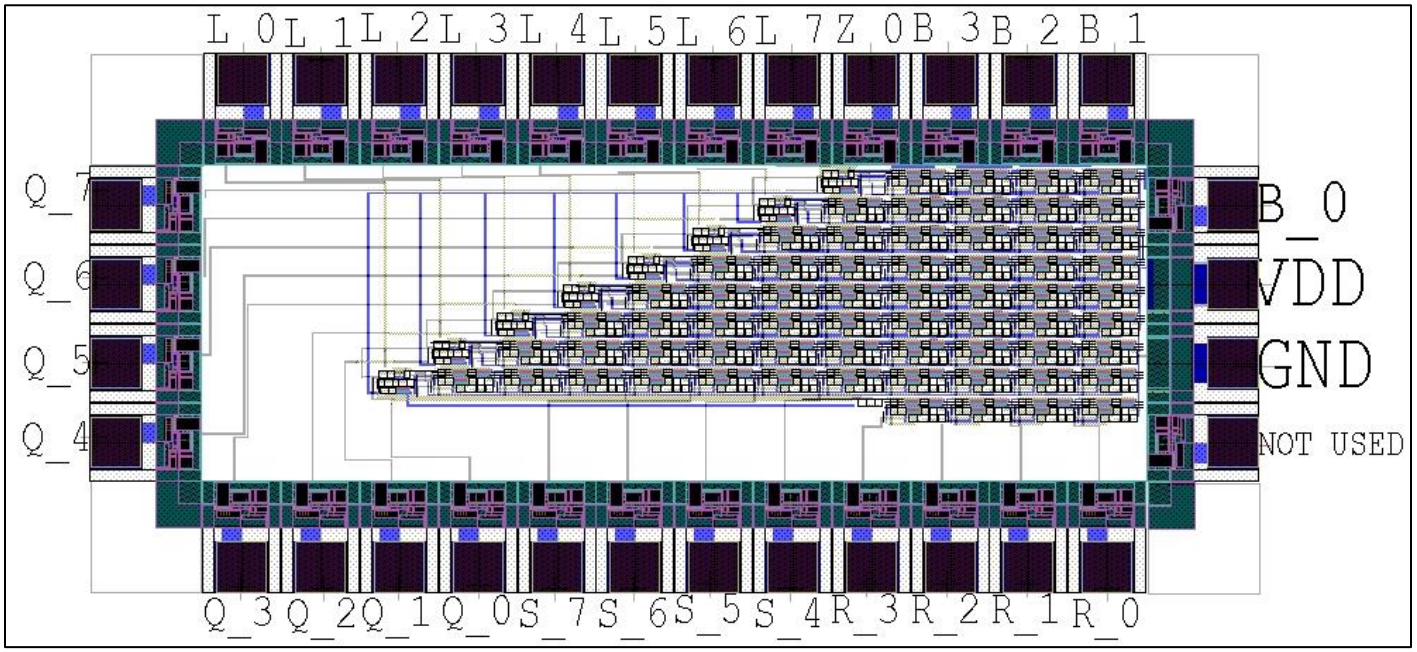


**Figure 11: Full design layout with PADs**

# Layout Simulation Results (Using PSPICE circuit simulator)

```
* INPUTS
VVDD 88 0 0v
VGND 104 0 0v
VLf7 183 0 0v
VLf6 357 0 0v
VLf5 664 0 0v
VLf4 973 0 0v
VLf3 1269 0 0v
VLf2 1484 0 0v
VLf1 2177 0 0v
VLf0 2540 0 0v
VZ 491 0 0v
VBf3 167 0 0v
VBf2 145 0 0v
VBf1 123 0 0v
VBf0 103 0 0v
```

Figure 12 : inputs nodes names

| Node's Name | Node's Number |
|---|---|
| VDD | 88 |
| Gnd | 104 |
| Lf7 | 183 |
| Lf6 | 357 |
| Lf5 | 664 |
| Lf4 | 973 |
| Lf3 | 1269 |
| Lf2 | 1484 |
| Lf1 | 2177 |
| Lf0 | 2540 |
| Z | 491 |
| Bf3 | 167 |
| Bf2 | 145 |
| Bf1 | 123 |
| Bf0 | 103 |

Table 1: inputs node names

```
* OUTPUTS
*Q0 2545
*Q1 2180
*Q2 2017
*Q3 1481
*Q4 976
*Q5 819
*Q6 482
*Q7 186
*r0 2699
*r1 2722
*r2 2737
*r3 2762
*Sf4 2335
*Sf5 2368
*Sf6 2403
*Sf7 2428
```

Figure 13: output nodes names

| Node's Name | Node's Number |
|---|---|
| Q0 | 2545 |
| Q1 | 2180 |
| Q2 | 2017 |
| Q3 | 1481 |
| Q4 | 976 |
| Q5 | 819 |
| Q6 | 482 |
| Q7 | 186 |
| R0 | 2699 |
| R1 | 2722 |
| R2 | 2737 |
| R3 | 2762 |
| Sf4 | 2335 |
| Sf5 | 2368 |
| Sf6 | 2403 |
| Sf7 | 2428 |

Table 2: output node names

Corner cases and some random cases for division:

**(Remainder in the first 2 plots and Quotient in the last 4 plots)**

- L=255 , B=1:

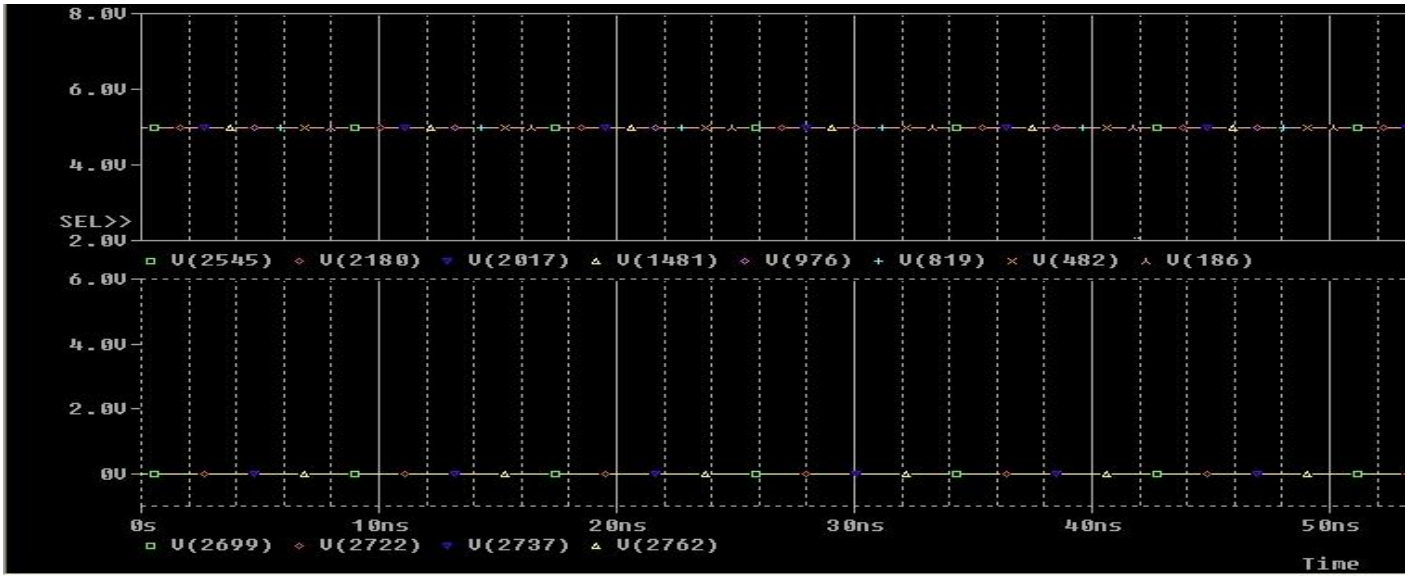


Figure 14: division of 255 by 1

Remainder (R3 R2 R1 R0) all = logic '0'

Quotient (Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0) all = logic '1' =255

- L=255 , B=15:

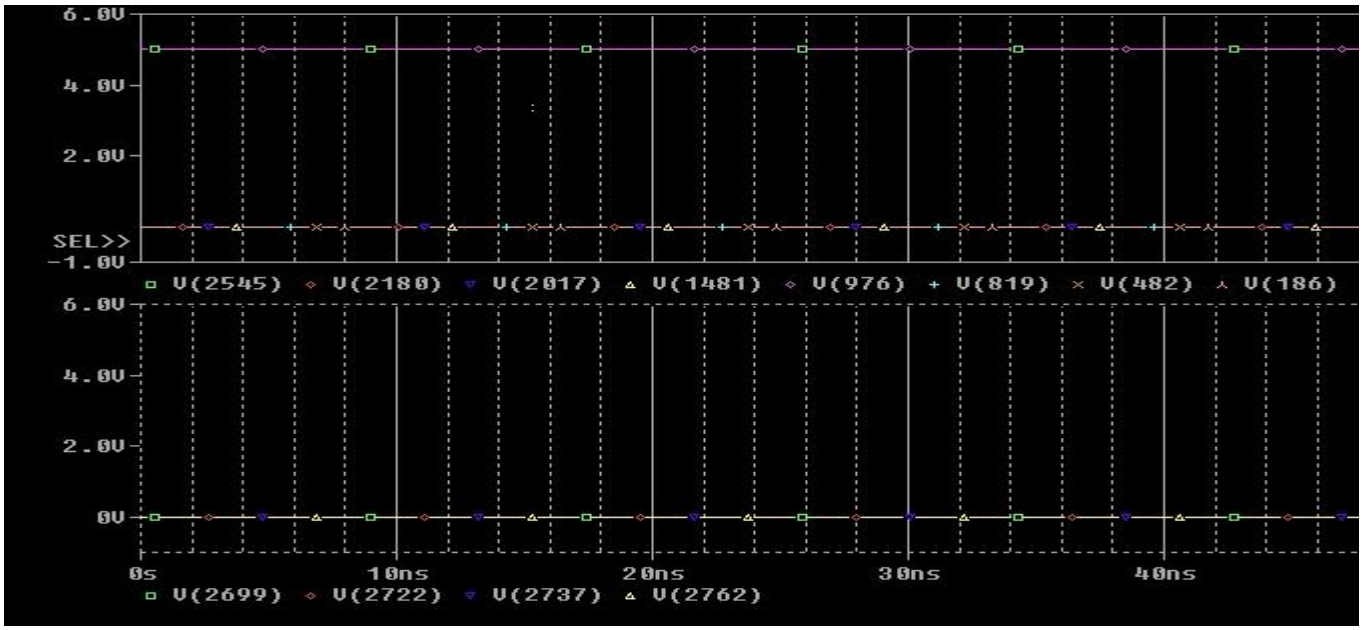


Figure 15: division of 255 by 15

Remainder (R3 R2 R1 R0) all = logic '0'

Quotient (Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0) = (00010001)=17

- L=14 , B=15:

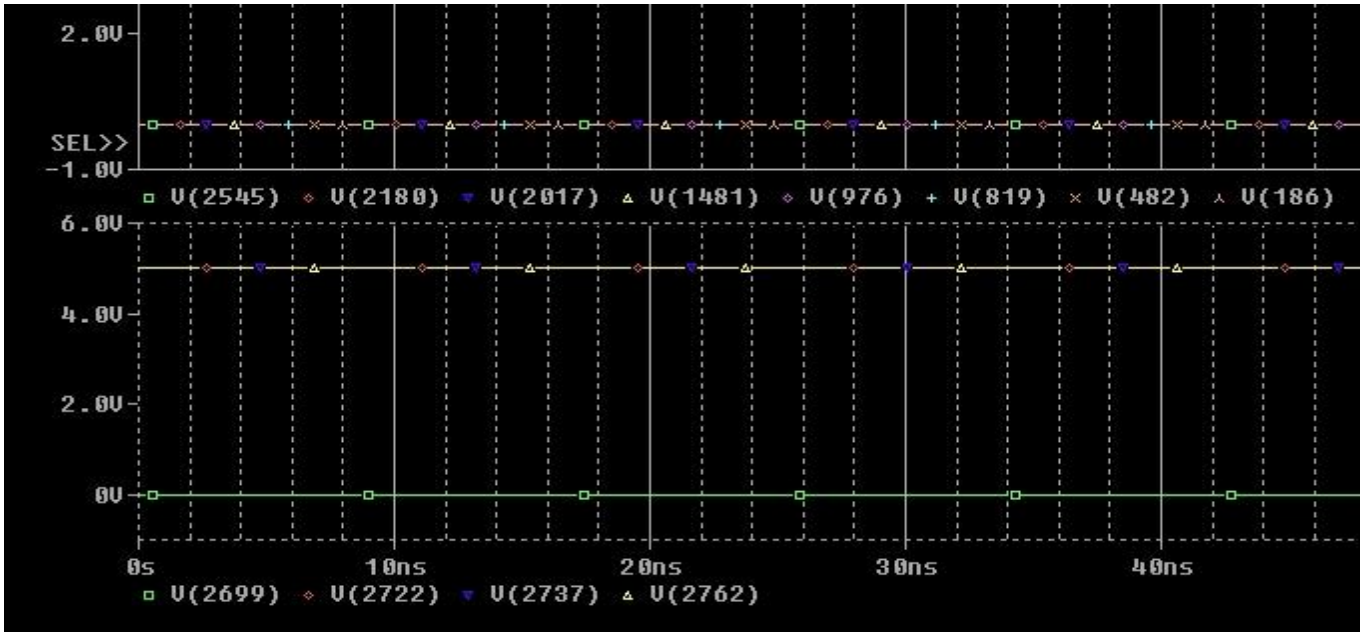


Figure 16: division of 14 by 15

Remainder (R3 R2 R1 R0) = (1 1 1 0)

Quotient (Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0) all = logic '0'

- L=15 , B=15:

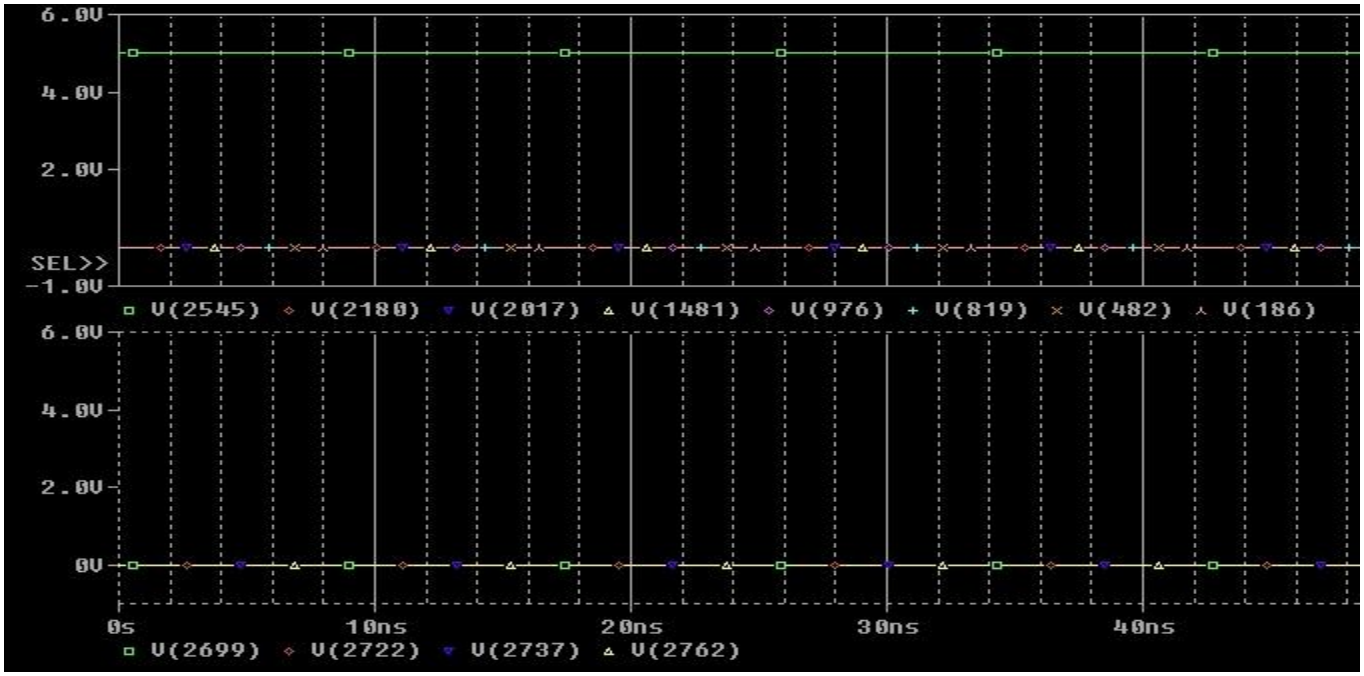


Figure 17: division of 15 by 15

Remainder (R3 R2 R1 R0) all= logic '0'

Quotient (Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0) = (0 0 0 0 0 0 0 1)

- L=0 , B=15:

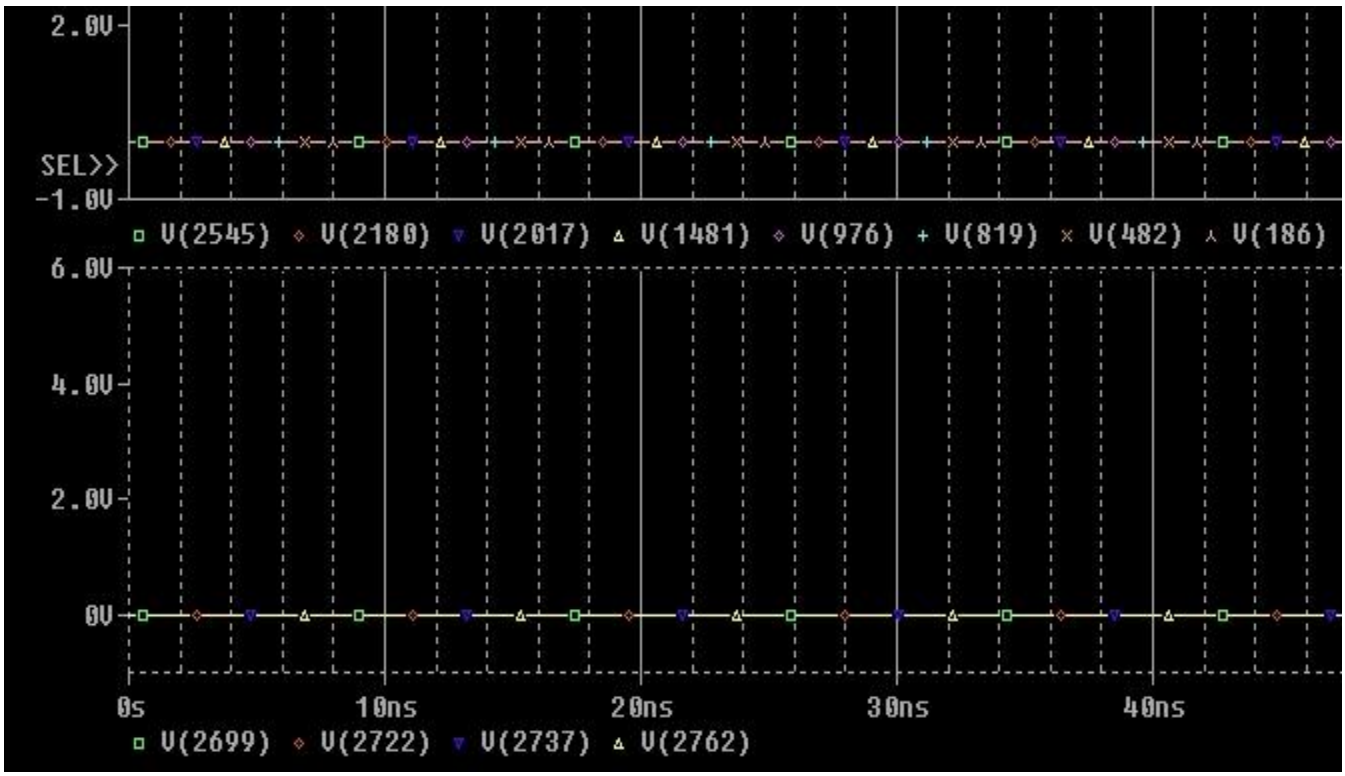


Figure 18: division of 0 by 15

All Remainder and Quotient = logic '0'

## Corner cases and some random cases for Multiplication:

- L=0,B=15:

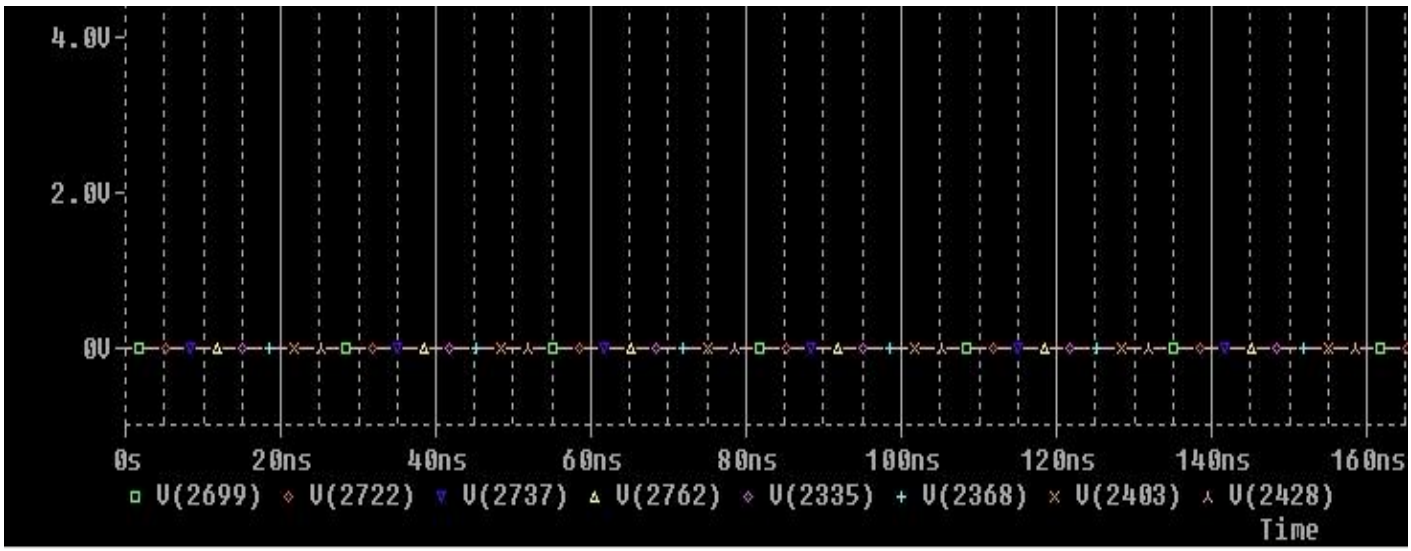


Figure 19: Multiplication of 0 by 15

Multiplication output (S7 S6 S5 S4 r3 r2 r1 r0) all = logic '0'

- L=15, B=15

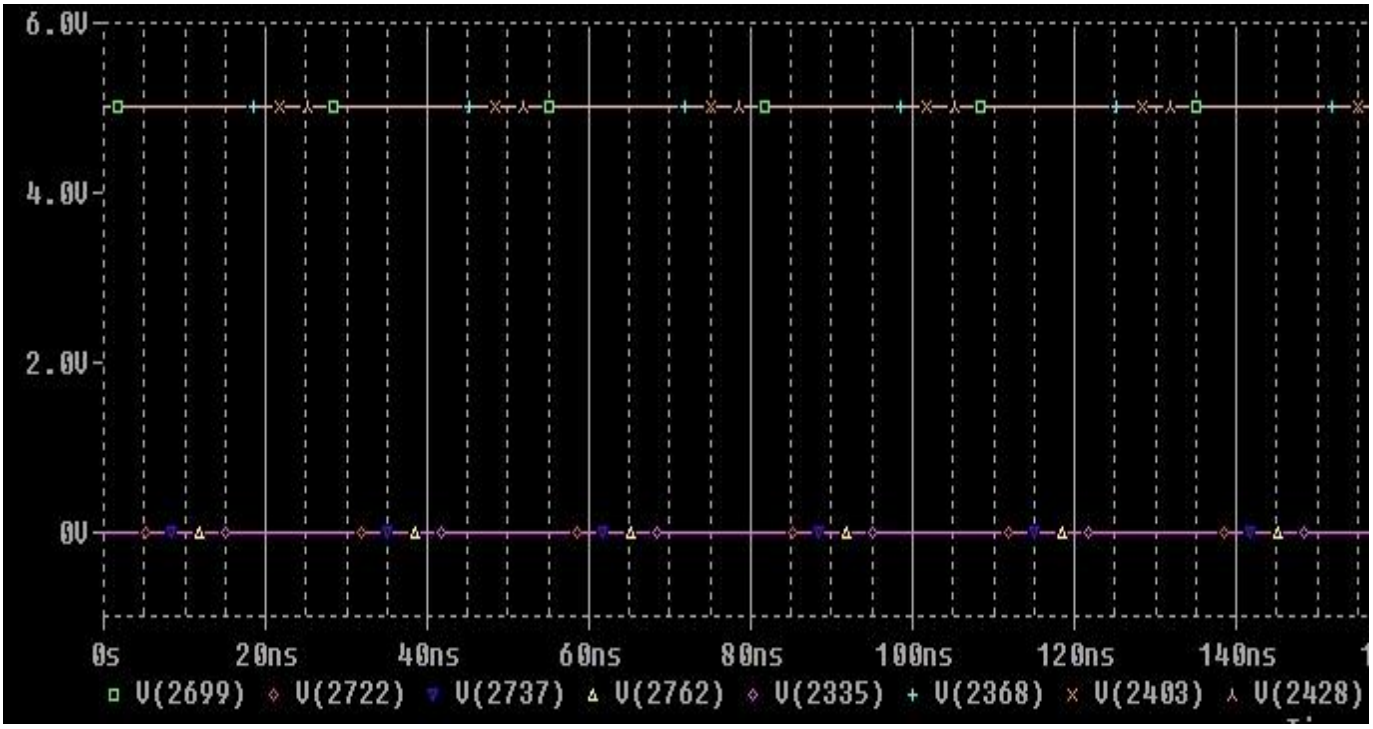


Figure 20: Multiplication of 15 by 15

Multiplication output (S7 S6 S5 S4 r3 r2 r1 r0) = (1 1 1 0 0 0 0 1) =225

- **L=0,B=0:**

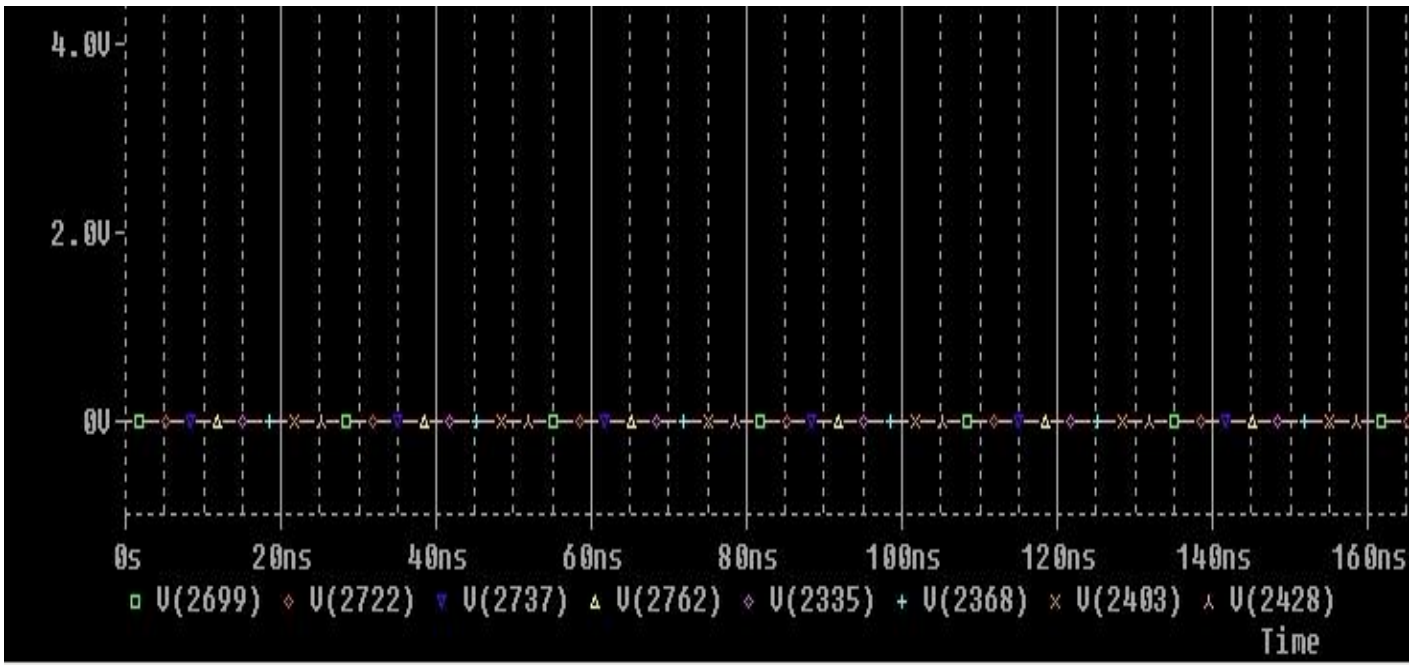


Figure 21: Multiplication of 0 by 0

Multiplication output (S7 S6 S5 S4 r3 r2 r1 r0) all = logic '0'

- L=1,B=15:

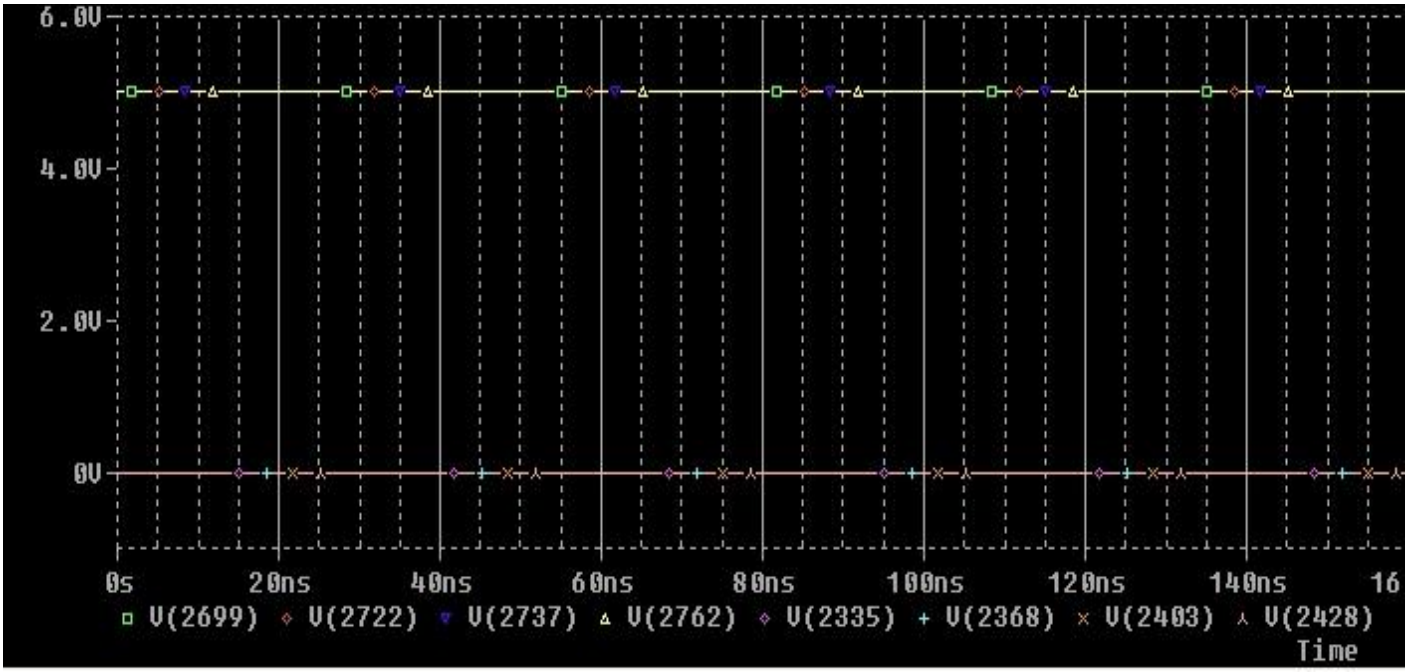


Figure 22: Multiplication of 1 by 15

Multiplication output (S7 S6 S5 S4 r3 r2 r1 r0) = (0 0 0 0 1 1 1 1) =15

- L=14,B=15:

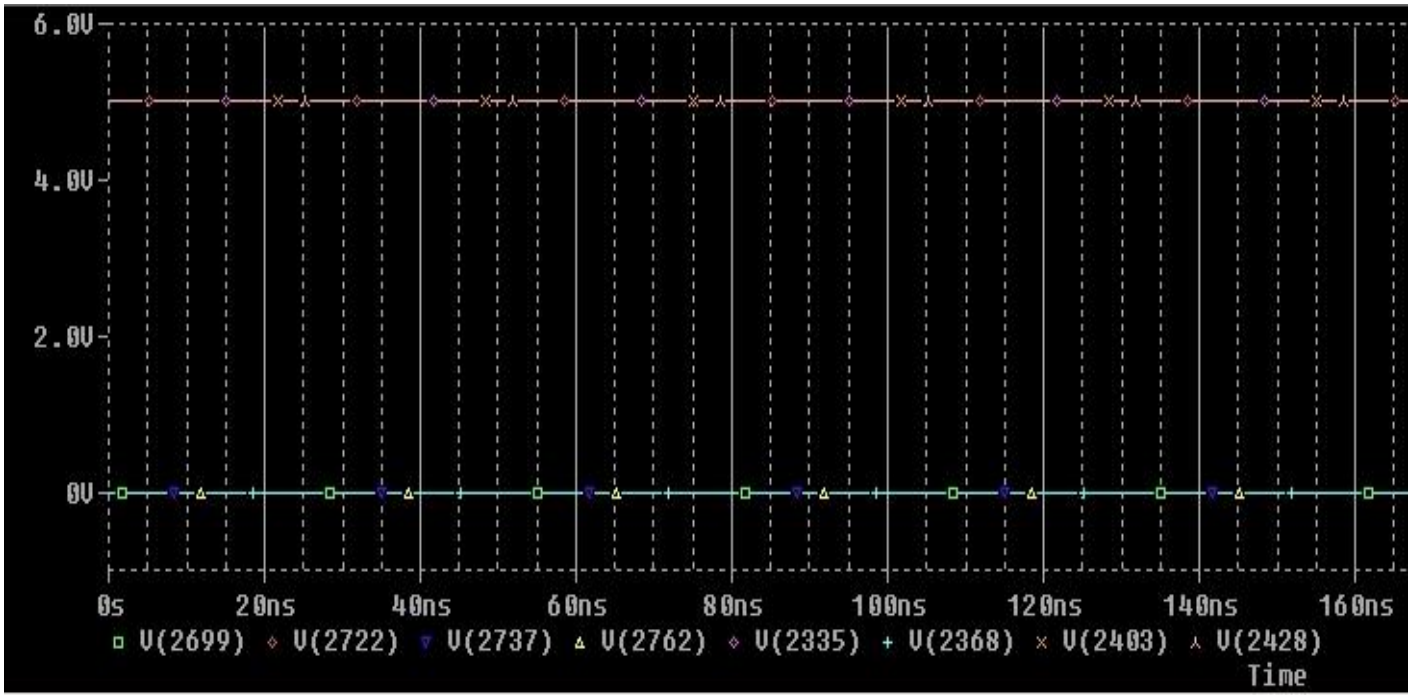


Figure 23: Multiplication of 15 by 14

Multiplication output (S7 S6 S5 S4 r3 r2 r1 r0) = (1 1 0 1 0 0 1 0)

## Critical Path determination

First we determined the critical path as it's the longest path in the array between an input and an output, and it has the worst case delay.

Our longest path is between the input (A7) and the last output (Q0) as shown in figure 24:


Figure 24: Critical Path determination

## Critical Path delay:

There are different paths from inputs to outputs in a digital circuit. Thus, naturally the output will include glitches or dynamic hazards. We have to read output at right time after it settles. This happens in all digital circuits, processors, etc.

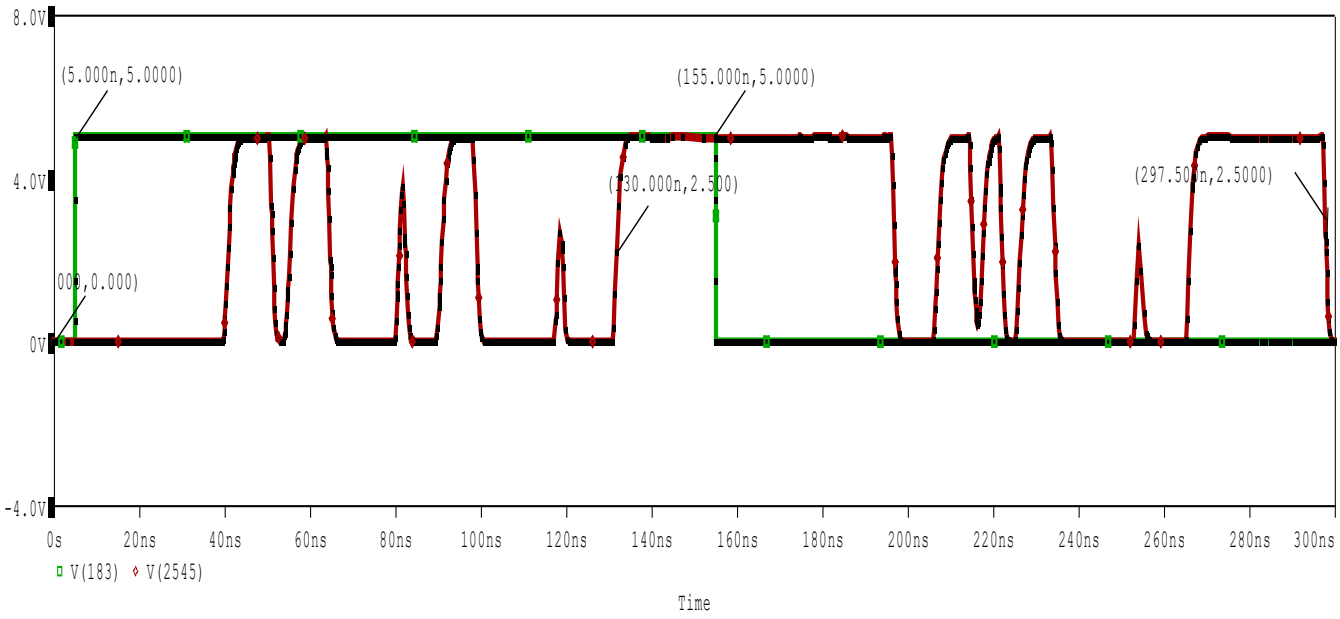


**Figure 25: Critical Path delay calculation**

As shown in figure 25:

Time propagation from high to low is the time from 50% input to 50% output when it settles to logic '1':

TPLH=130-5=**125nsec.**

Time propagation from low to high is the time from 50% input to 50% output when it settles to logic '0':

TPHL=297-155=**142nsec.**

*So the worst case delay is 142nsec.*

# Performance Calculation

## Power dissipation:

```
VOLTAGE SOURCE CURRENTS
NAME         CURRENT

VVDD        -9.695E-09
VGND         9.695E-09
VLf0         0.000E+00
VLf1         0.000E+00
VLf2         0.000E+00
VLf3         0.000E+00
VLf4         0.000E+00
VLf5         0.000E+00
VLf6         0.000E+00
VLf7         0.000E+00
VBf0         0.000E+00
VBf1         0.000E+00
VBf2         0.000E+00
VBf3         0.000E+00
VZ           0.000E+00

TOTAL POWER DISSIPATION   4.85E-08  WATTS
```

$P_{dissipation} = V_{DD} * I = 5 * 9.695 * 10^{-9} =$ **48.5 nWATTS**

## Design area without PADS:

Area= 4362.5 λ*1425λ=**$6216562.5\lambda^2$**

As we work with 0.5 µm N well CMOS process: λ=0.25 µm

∴ Total area=$6216562.5 * (0.25 * 10^{-6})^2 =$ **$0.3885\ \mu m^2$**

# CONCOLUSION

We designed a circuit of IAD benchmarked from ref1 and ref2 given with some modifications to get accurate results and reduce the area, we achieved area of 0.3885 $\mu m^2$ using the standard cells of CUSCLIB standard library. The design is verified using RTL Verilog language and Testbenches and simulated with XILINIX ISE, and it works, for the layout we used PSPICE circuit simulator and it works well.

| | |
|---|---|
| **Number of used cells** | 72 cells(60 UAC ,8 Control ,4 Error detection) |
| **Control cell area** | $28101.5\lambda^2=1.7563nm^2$ |
| **UAC area** | $41148\lambda^2=2.6nm^2$ |
| **Design area without PADS** | $6216562.5\lambda^{2=}0.3885\ \mu m^2$ |
| **Total power dissipation** | 48.5 nWATTS |
| **worst case delay** | 142nsec |

**Table 3: Final Design results**