Analog and Digital VLSI Design

8-bit Odd Parity Detector.

Digital Assignment

A PROJECT REPORT BY:

NAME	ID
Akash Pratap Singh	2018A8PS0462P
Abhinandan Sharma	2018A3PS0095P
Surbhi Goel	2018A8PS0424P
Akshay Mittal	2018A8PS0772P

EEE F313/ INSTRF313 – Analog & Digital VLSI Design

Under the Guidance of

Prof. Anu Gupta

(Instructor In-charge *EEEF313/ INSTR F313)*

EEE Department,

BITS Pilani, Pilani Campus

ACKNOWLEDGEMENTS

We would like to extend our gratitude to Dr Anu Gupta for giving us the opportunity to have hands on experience of layout design. Our theoretical concepts were put to practical use which was in itself an extremely satisfying job.

We would like to thank the team of teaching assistants for providing us with the appropriate open source softwares and demo lectures on how to use them.

At last, we thank our parents and friends for moral support and encouragement.

PROBLEM 80

(a) Design an 8-Bit odd Parity Generator/ Detector at 1GHz with load capacitance of 1pF.

Design & Explanation of the Circuit:

Odd Parity Checker

A bit stream is given as a serial input to the circuit. The circuit considers 8 bits at a time and gives the output as 1 if **odd** number of ones were present. Then, after **8 clock pulses,** the output is given as 1 if **odd** number of ones were present in those first **16** bits. In this way, Parity of the bit stream is detected at the output after every **8 clock pulses.** Hence, the output after 8*n cycles is one if there were odd number of ones present in the first 8*n bits of the bit stream.

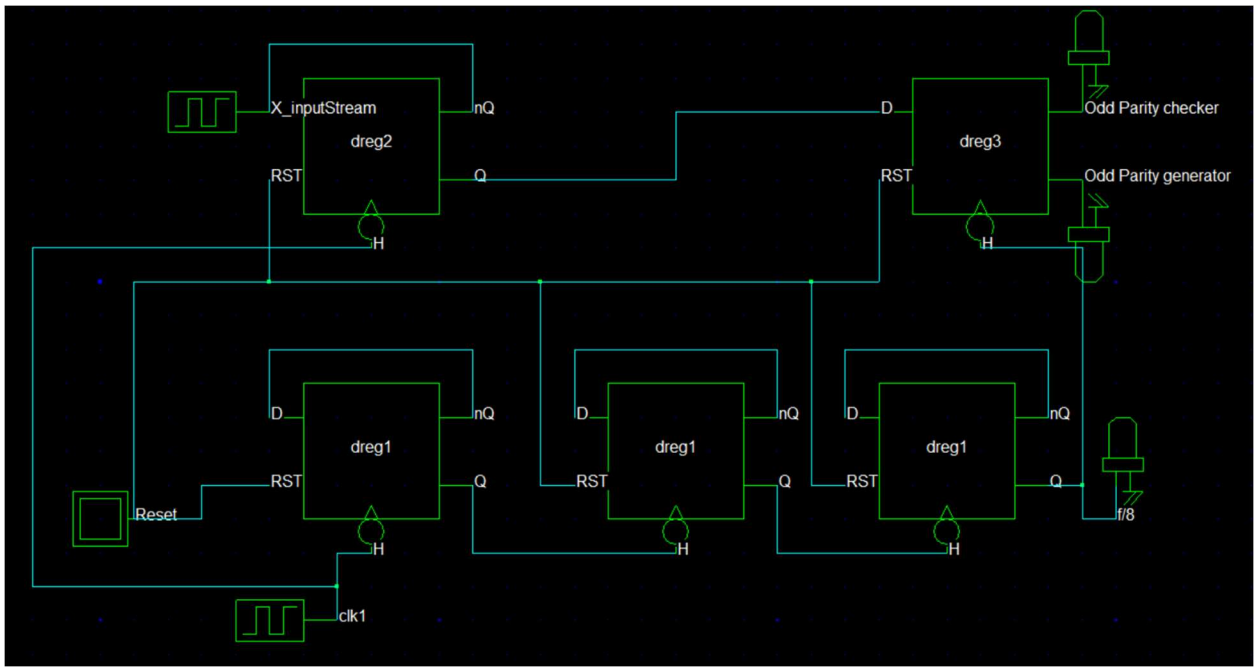
The final output (q_b) is the odd parity detector (logic 1 if odd number of ones). The output (~q_b) is the odd parity generator (logic 1 if even number of ones).

For example,

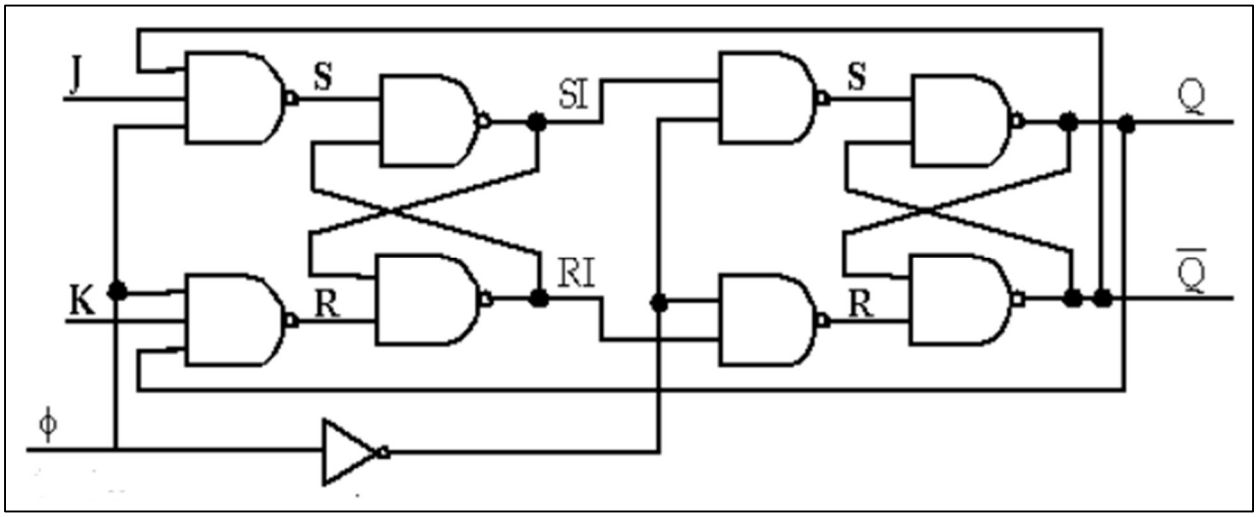
Input	:	10100010 01001010 11010100 010010101 01110110
q_a	:	11000011 10001100 10011000 011100110 01011011
q_b	:	1 0 0 0 1
~q_b	:	0 1 1 1 0

This technique of cumulative parity detection helps us find the overall parity after every 8 cycles and is a more **convenient way of making the overall parity odd.**

Flip Flop Level Schematic –



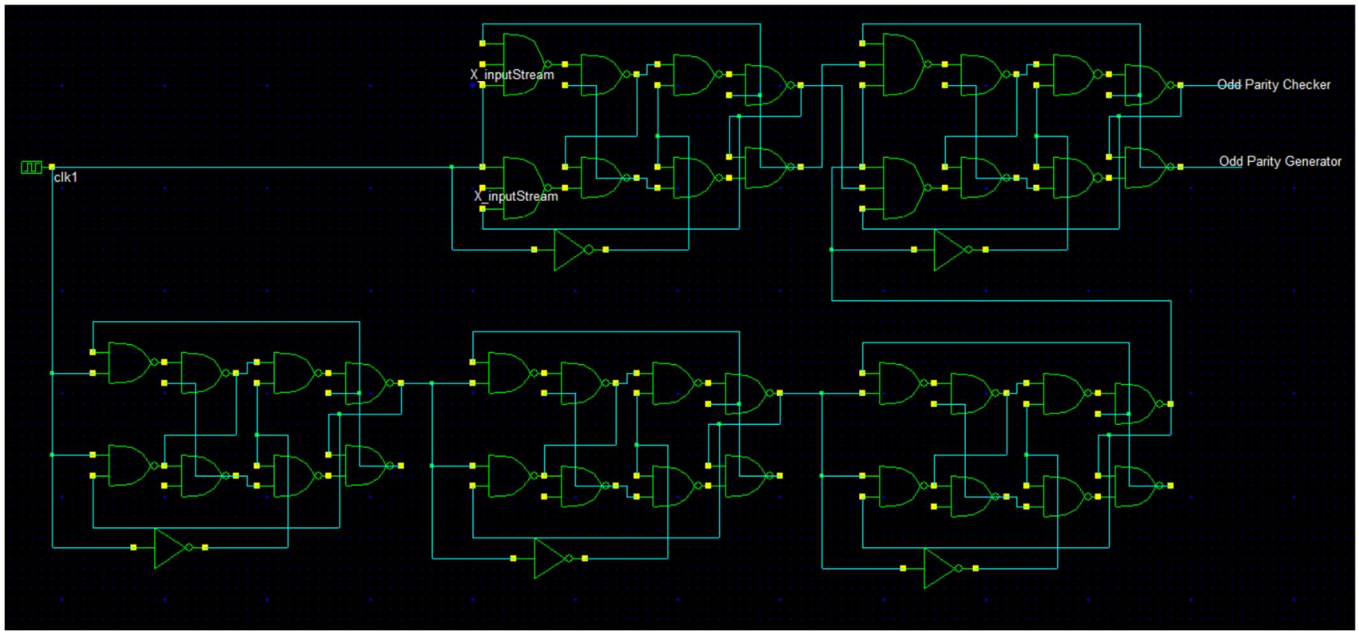
Gate level schematic of J,K flip flop –



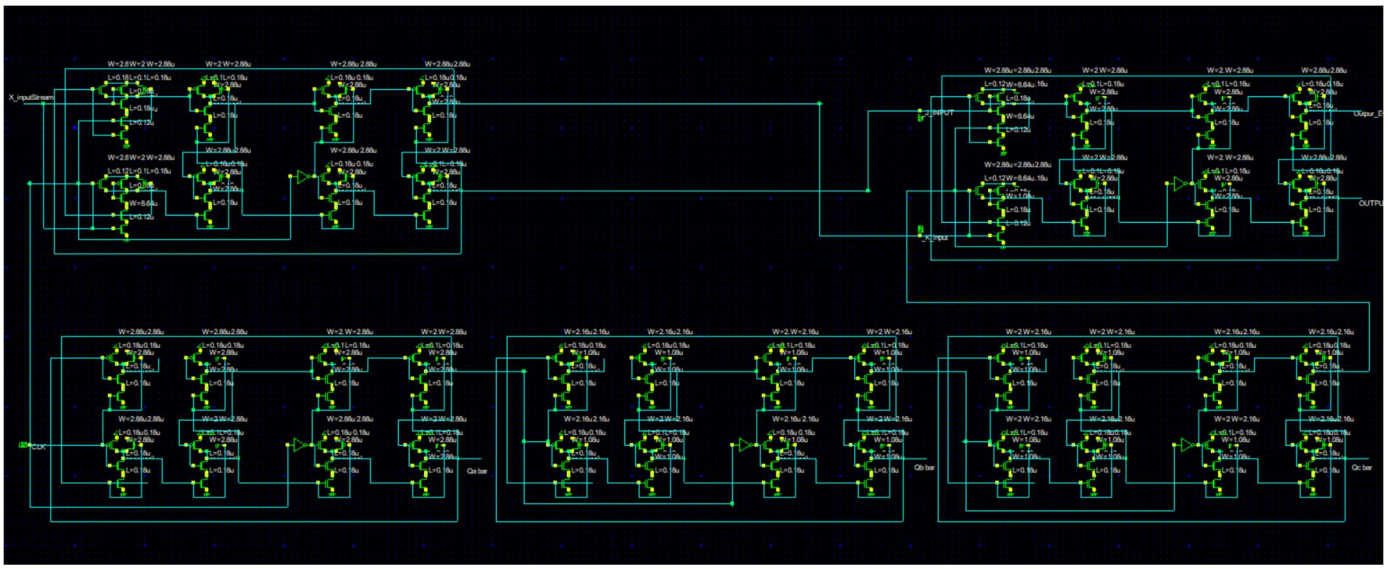
When Using it in the form of frequency divider (T-flip flop), Input '**J**' & '**K**' are supposed to be set to '1' so that it can work in toggle configuration.

Hence, in the T Flip Flops of clock divider circuit, we will use 2 input NAND gate instead of 3 input as if third input is set to 1, it will not affect the output.

<u>Gate level design of Odd Parity checker/generator</u>



Transistor Level Designs:-

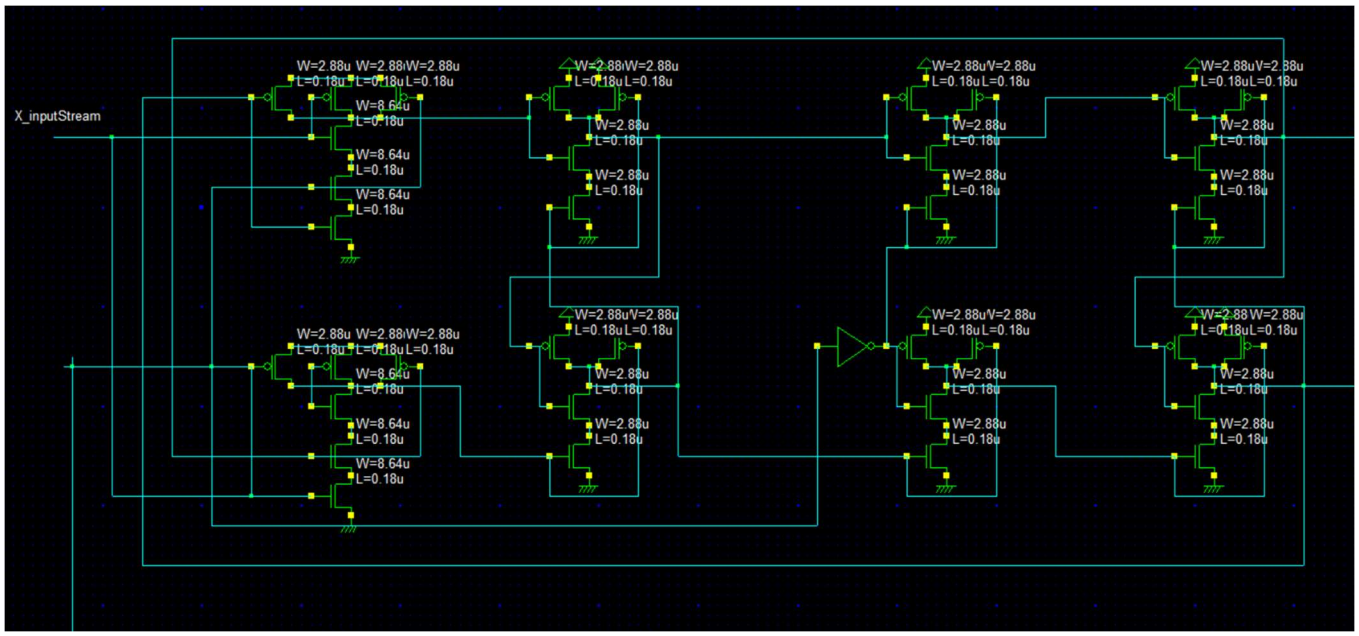


Complete Schematic

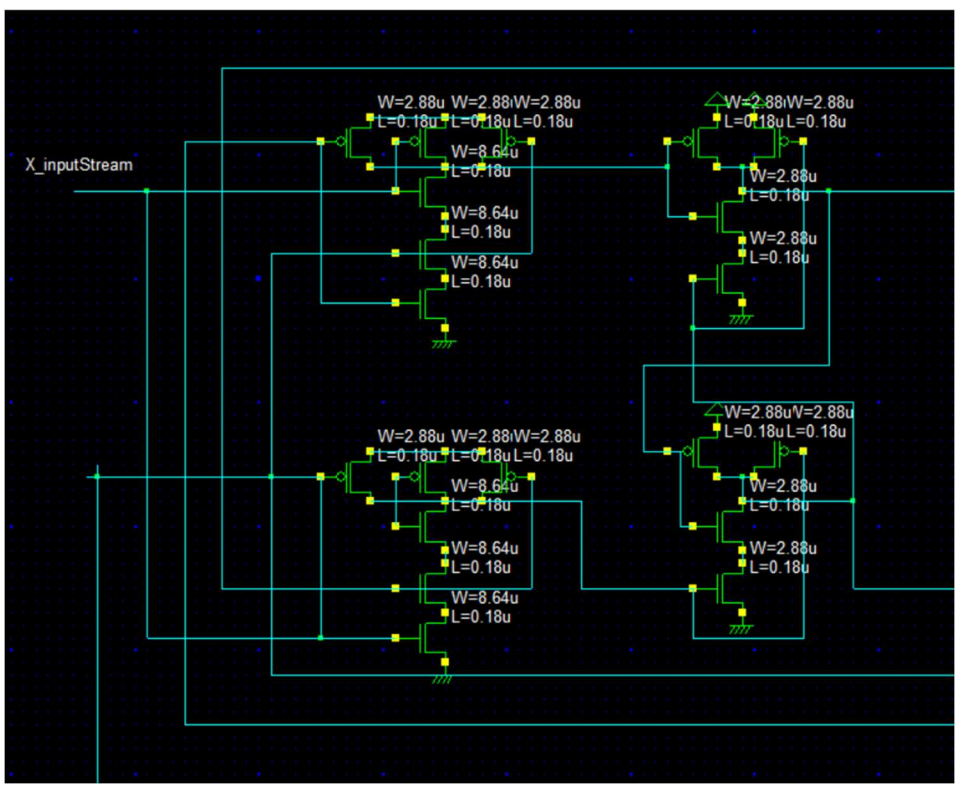
All these views are zoomed in upcoming pages.

- Top left view – T flip flop which receives input and ensures that the value in the line connected to its output is set to ‘1’ if odd number of 1’s received in input.
- Top Right view – D flip flop which passes the output after 8 bits have been received.
- Bottom T flip flops – These are being used for dividing frequency. One T Flip Flop halves the frequency, so three T flip flops reduces it by 8 times. Hence, the positive edge arrives the D Flip Flop only after 8 clock pulse. Hence, cumulative Parity is checked 8 bits at a time.

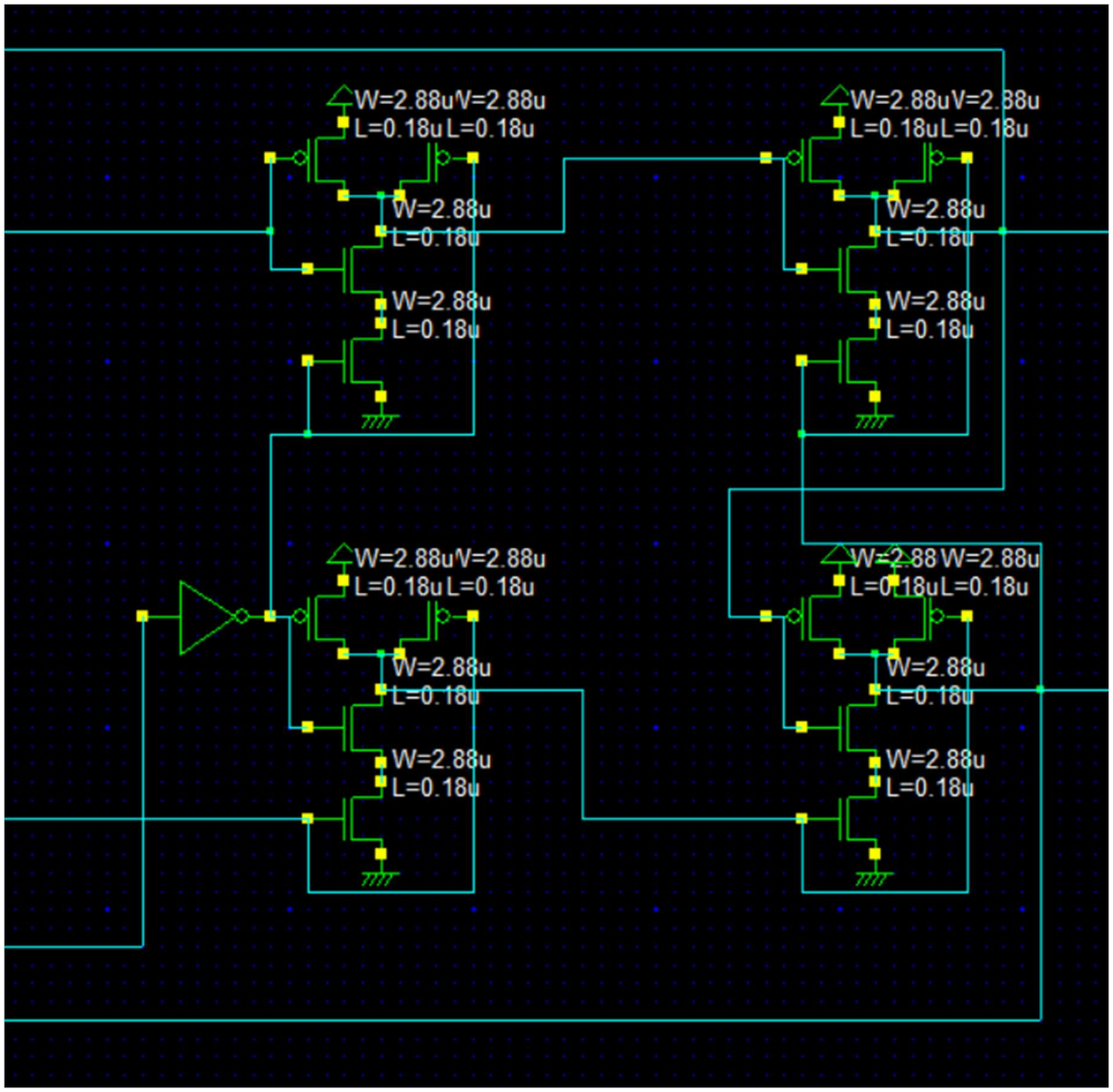
1) Top left view (T flip-flop with X_inputStream)-



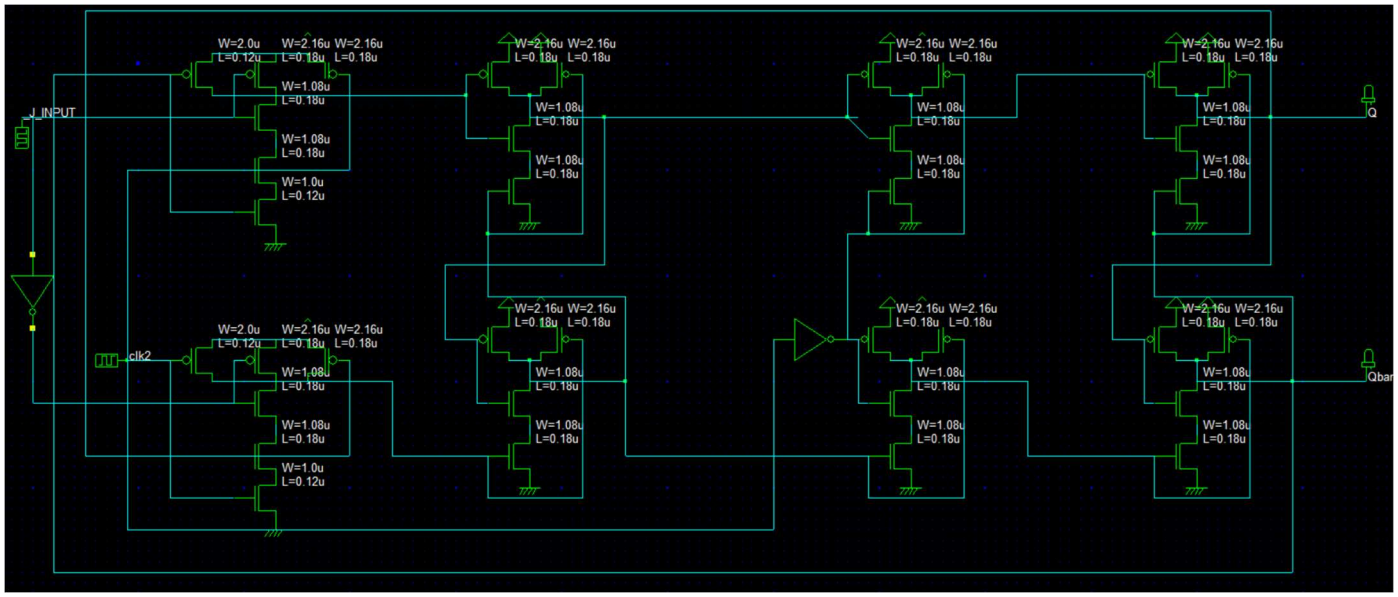
First and Second stage (Clock same as the input clock into the bottom row flip flops)-



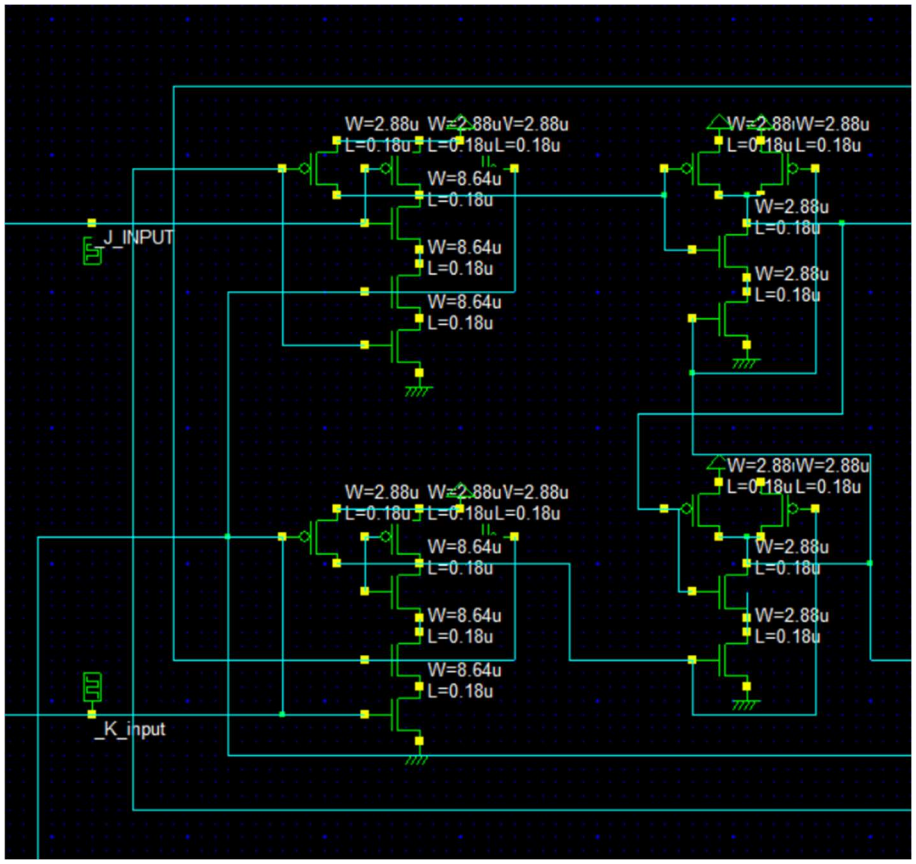
Third & Fourth Stages (Clock inverted of the previous stages)-



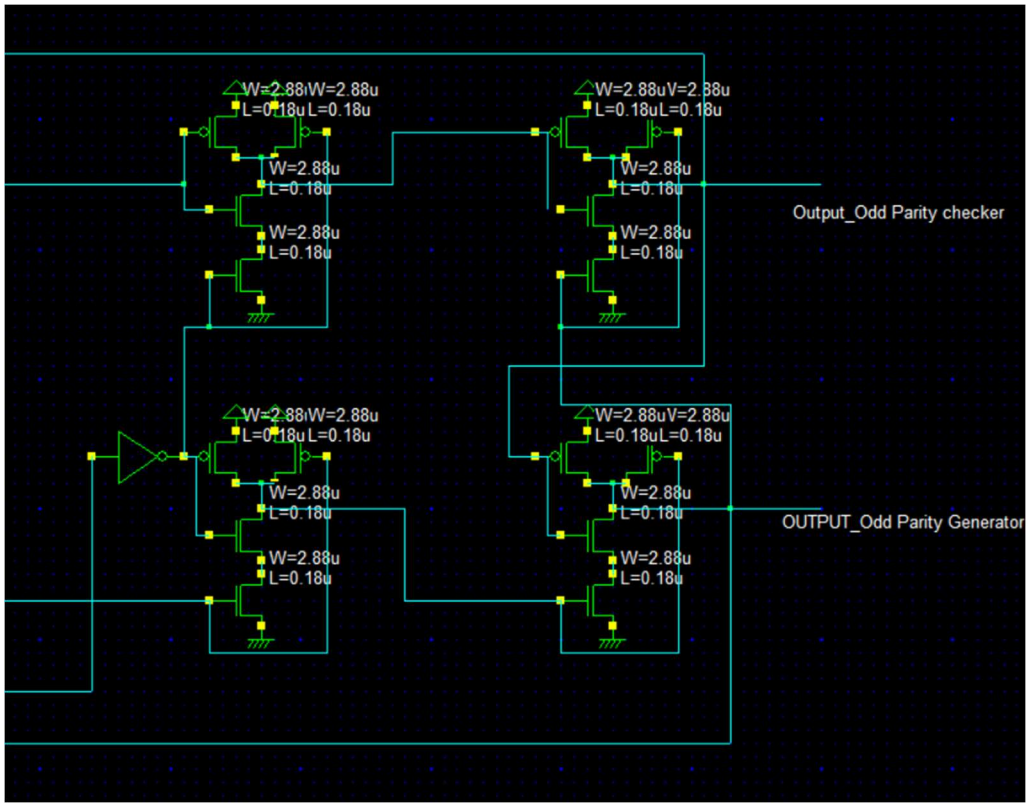
2) Top Right view (D flip flop MOS implementation)–



First(3 input NANDs) & Second Stage(2 i/p NANDs)-

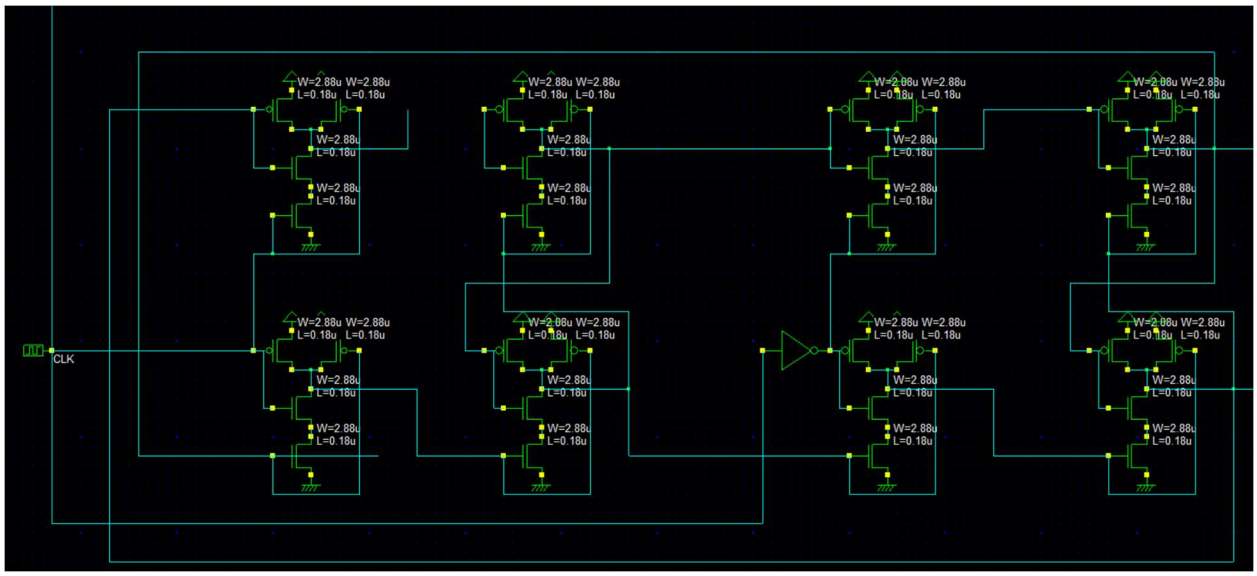


Third and Fourth Stage(2 i/p NANDs)–



c) Bottom View(T flip-flops used for dividing frequency, three of these used in bottom row of full schematic)–

T flip flop Transistor level design –



It is implemented using only 2 input NAND gates. Since third input set to ‘1’ for toggle configuration in J-K flip flop design. If input is ‘1’ then output of a NAND gate is not affected. Hence removal of MOS for third input will not affect the results.

W and L Values of the MOSFETs

The TSMC 180nm technology is used and hence, all the lengths (L values) are taken to be 0.18 um. The Logical Effort model is used to calculate the widths (W values).

From the cmos018.rul file provided to us, we found the values of mobilities of nMOS and pMOS.

u_n / u_p = 0.038 / 0.02 = 1.9

Hence, now we know that $(W/L)_{peq} = 1.9 * (W / L)_{neq}$
In Microwind, the widths must be an integer, therefore we approximate this result to:

$W_{peq} = 2 * W_{neq}$

We treat an inverter as the reference and find all values of W with reference to the W value of the inverter.

The minimum W permissible (due to the constraints on the contact – metal spacing) in Microwind is 6*lambda (W_{min} = 0.54unm). So, we varied the value of W to find the total propagation delay of the inverter. Here, there was a tradeoff between Area, speed and parasitics. So, we chose W to be 16*lambda (1.44um). Reason for this value is that

the delay stopped to decrease drastically after 14*lambda. Also, by using this value, we could easily accommodate 2 contacts in the diffusion region itself (otherwise, multiple metal wires would have to be drawn outside the diffusion region leading to much larger area consumption as distance between two metal wires must be atleast 4*lambda). Hence,

W_{min} = 1.44um

Three types of gates are constructed using the MOSFETs.

a) **2-input NAND Gate**

	W (um)	**L (um)**	**W/L**
nMOS	2*W_{min} = 2.88 (32*lambda)	0.18	16
pMOS	2*W_{min} = 2.88 (32*lambda)	0.18	16

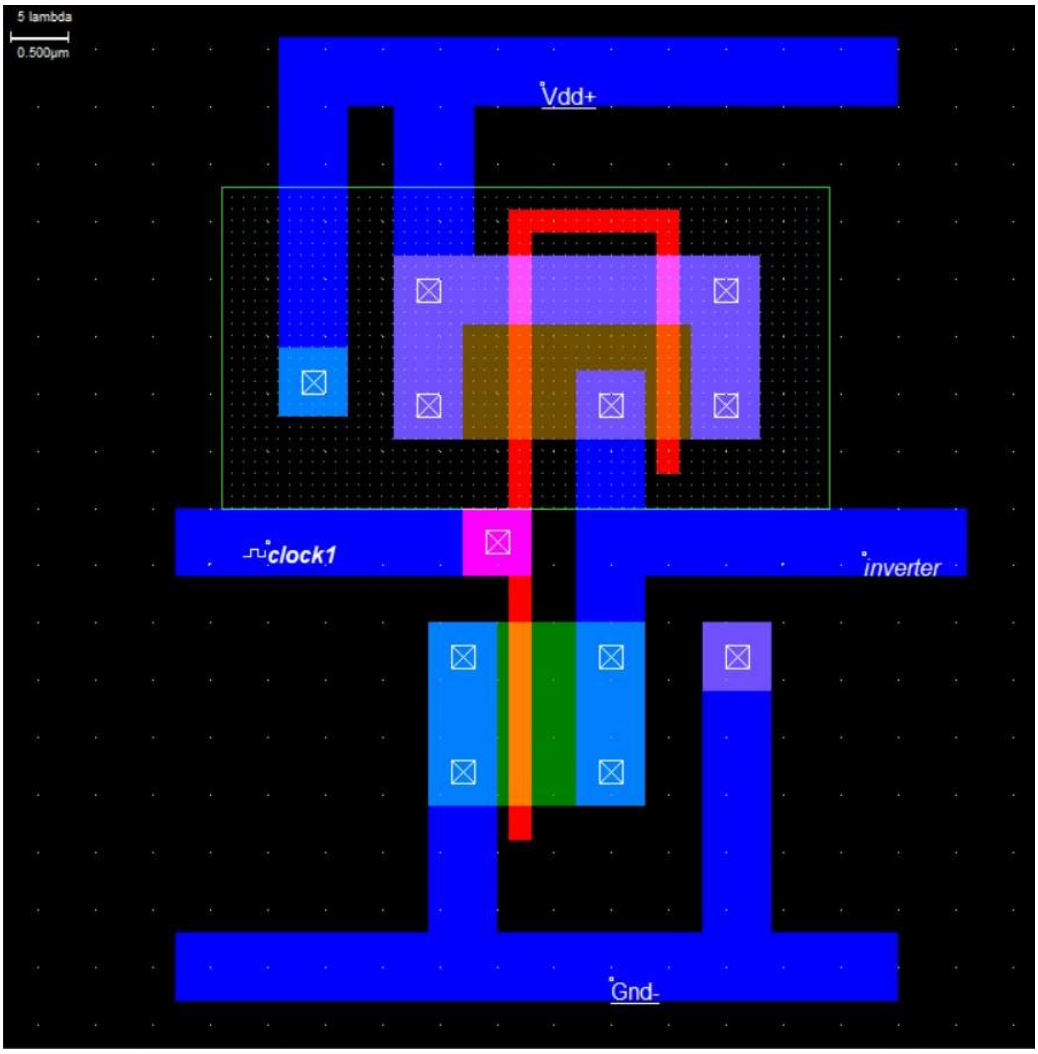
b) **3-input NAND Gate**

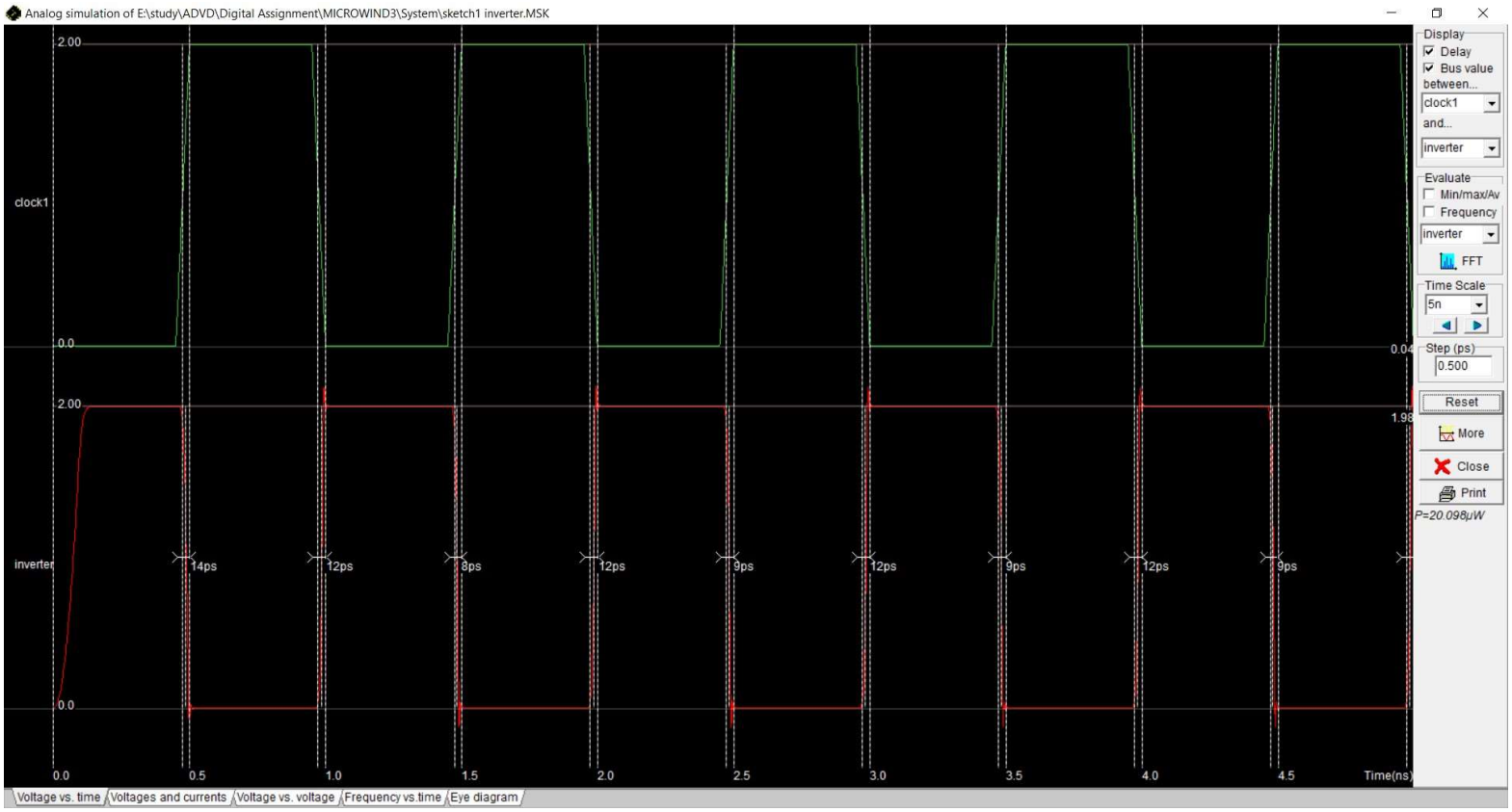
	W (um)	**L (um)**	**W/L**
nMOS	6*W_{min} = 8.64 (96*lambda)	0.18	48
pMOS	2*W_{min} = 2.88 (32*lambda)	0.18	16

c) Reference Inverter

	W (um)	L (um)	W/L
nMOS	W_{min} = 1.44 (16*lambda)	0.18	8
pMOS	2*W_{min} = 2.88 (32*lambda)	0.18	16

<u>Calculating Dealy of Reference Inverter</u>





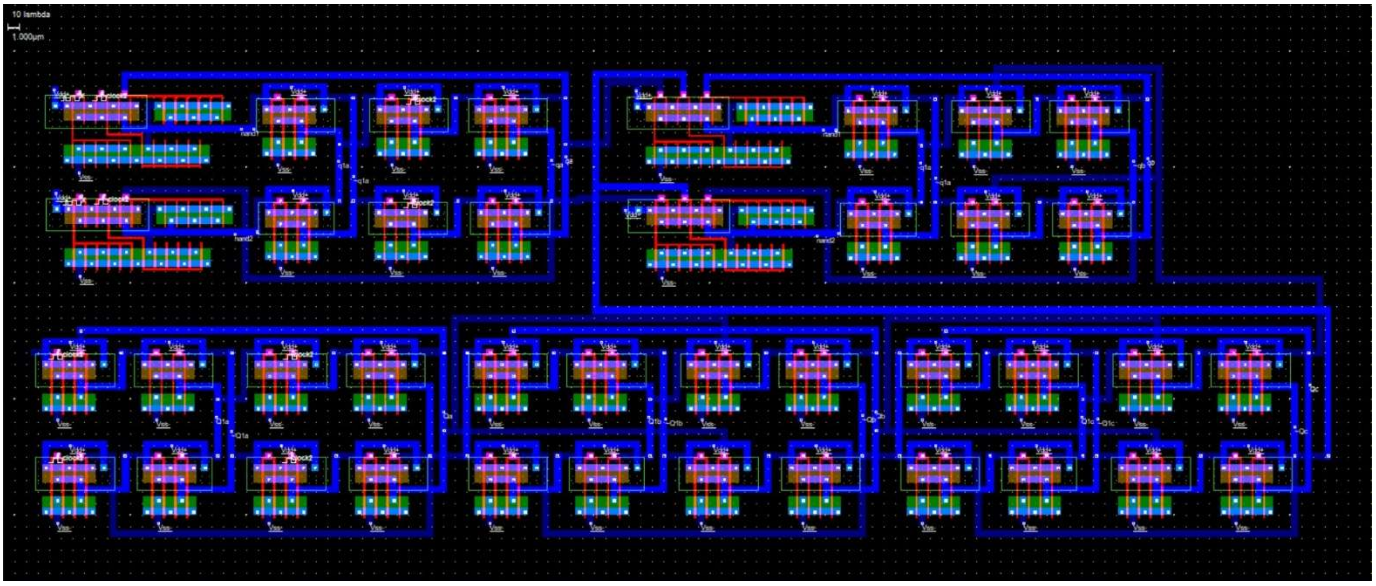
T_{pHL} = 9ps

T_{pLH} = 12ps

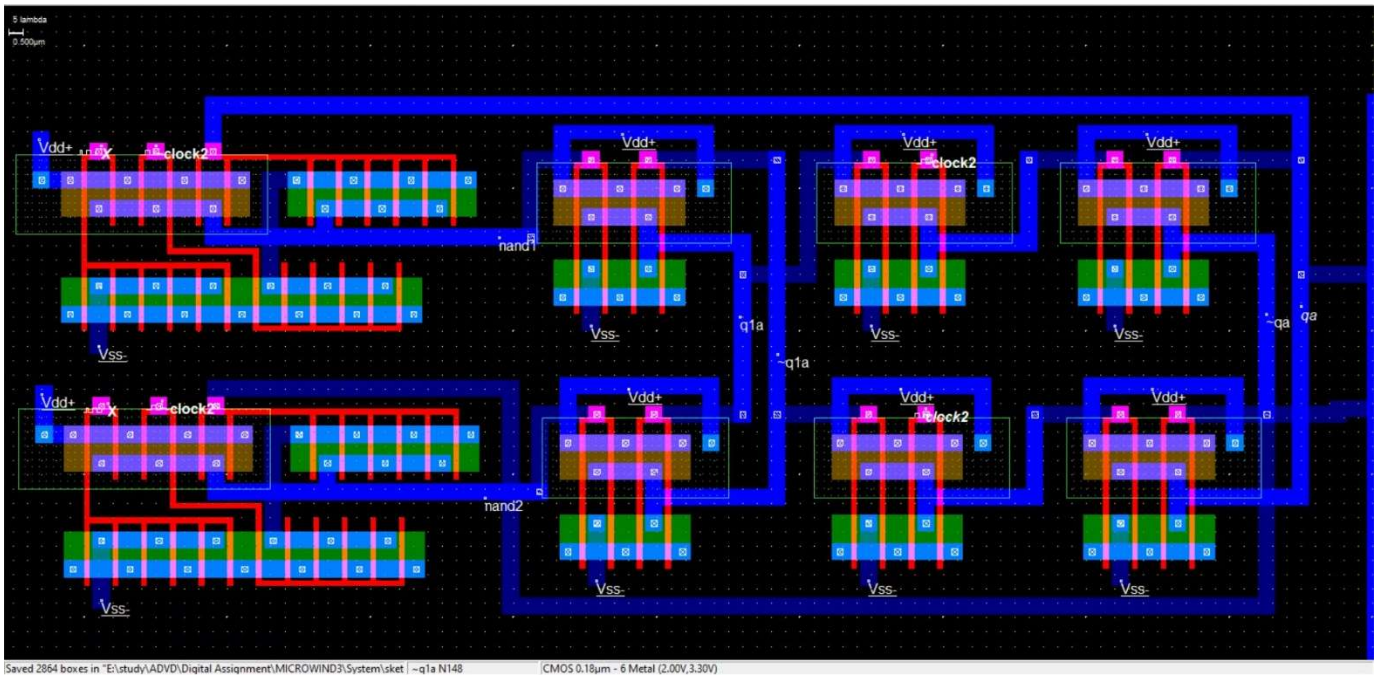
Hence, $T_p = (T_{pHL} + T_{pLH})/1$

$= 10.5$ ps

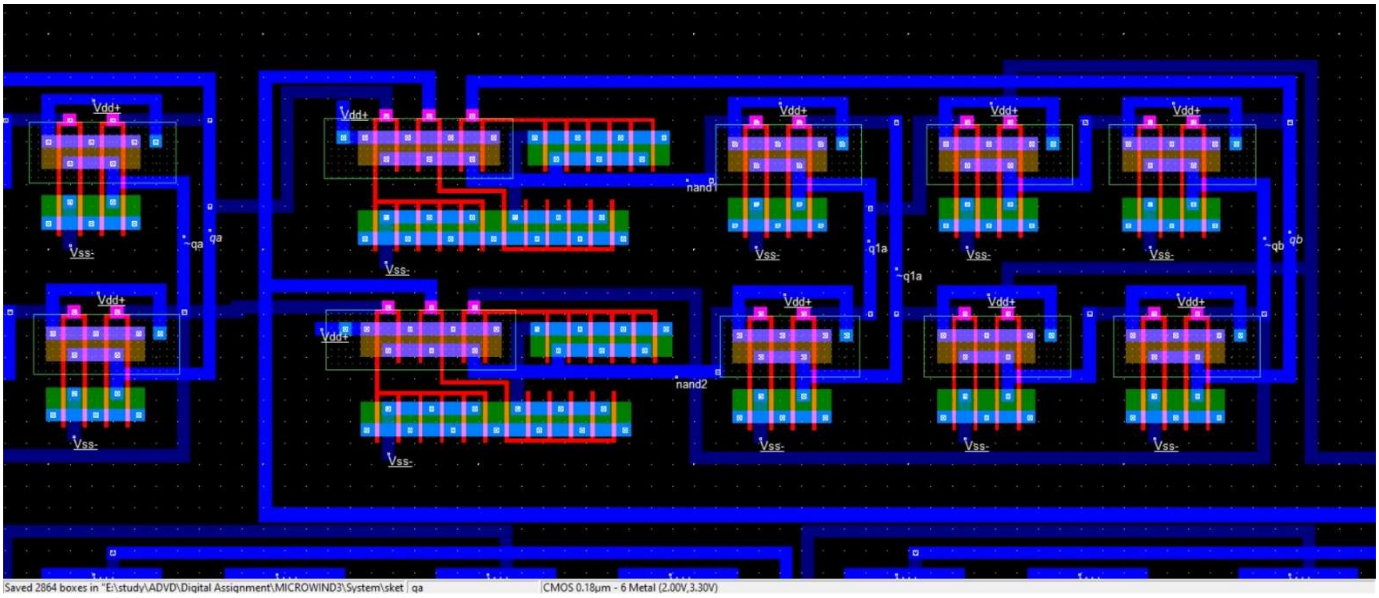
Layout of the Schematic



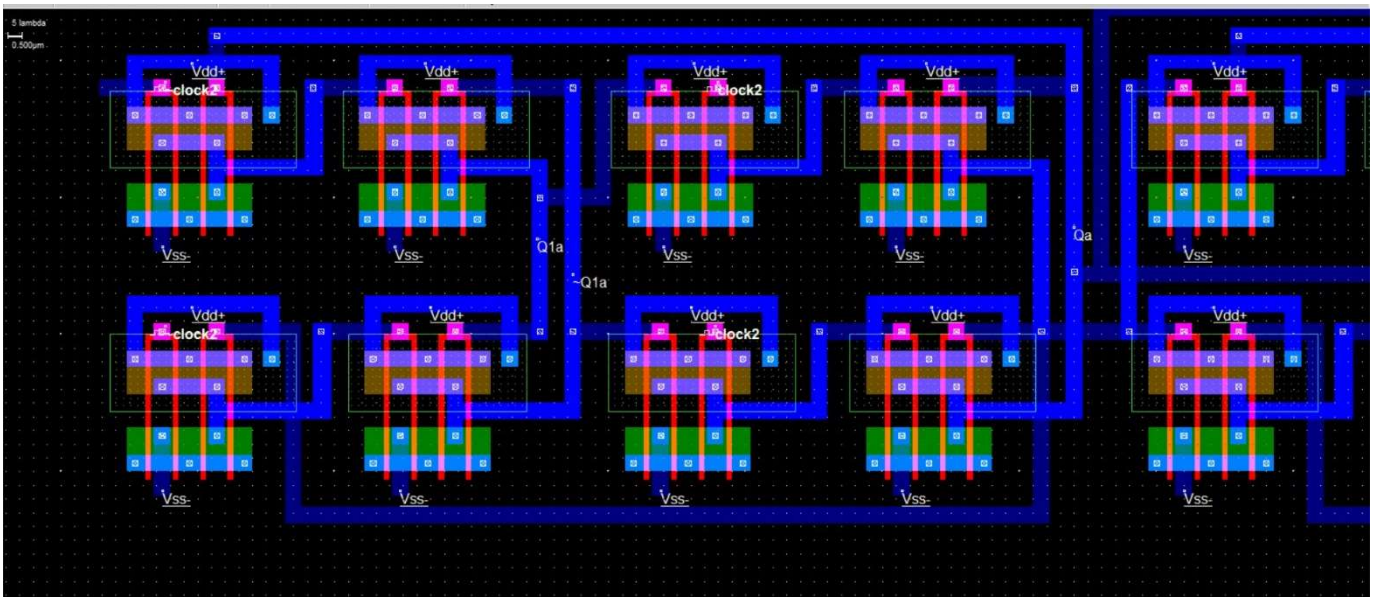
a) Top Left View



b) Top Right View

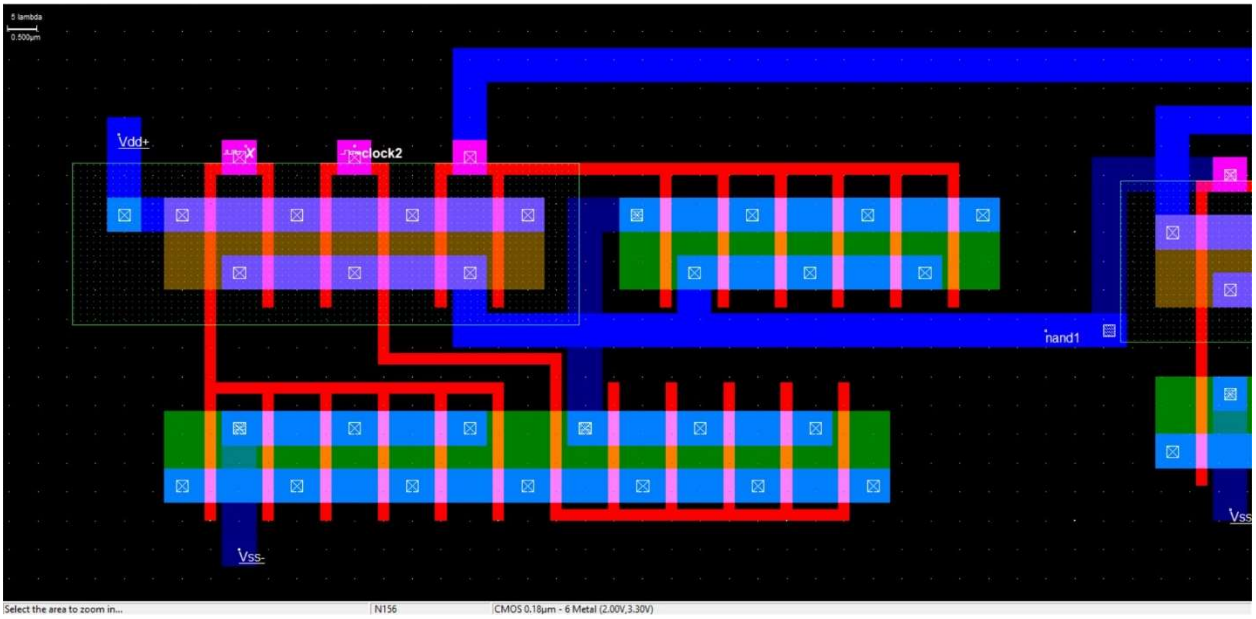


c) Bottom Left View

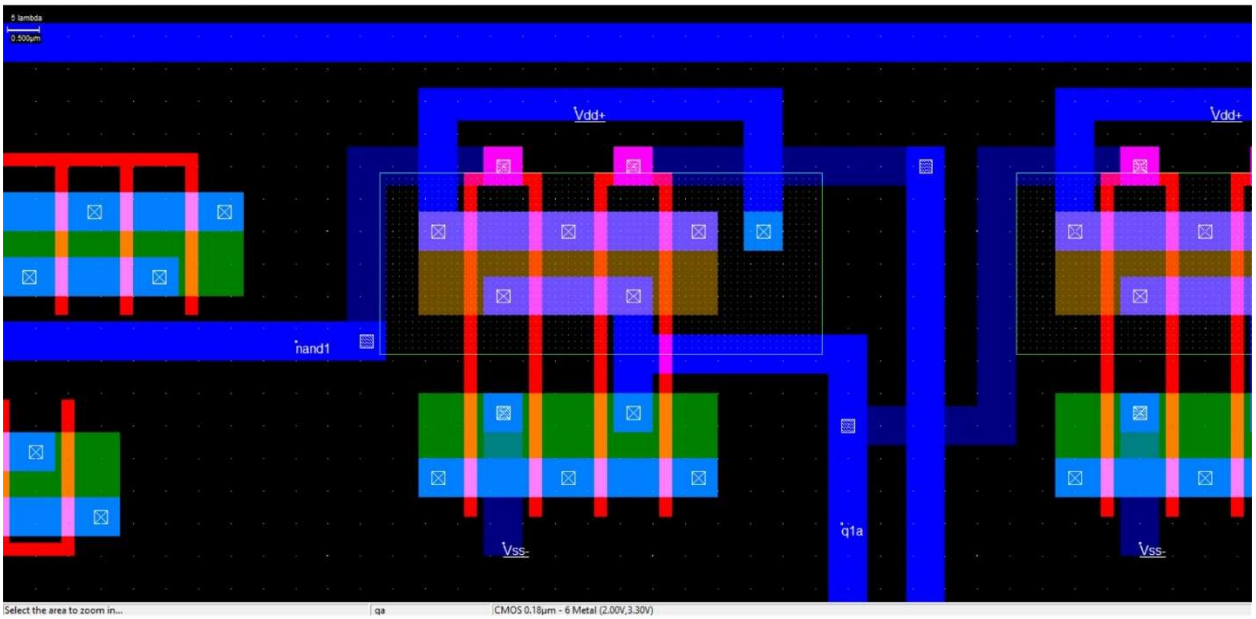


Gates Used

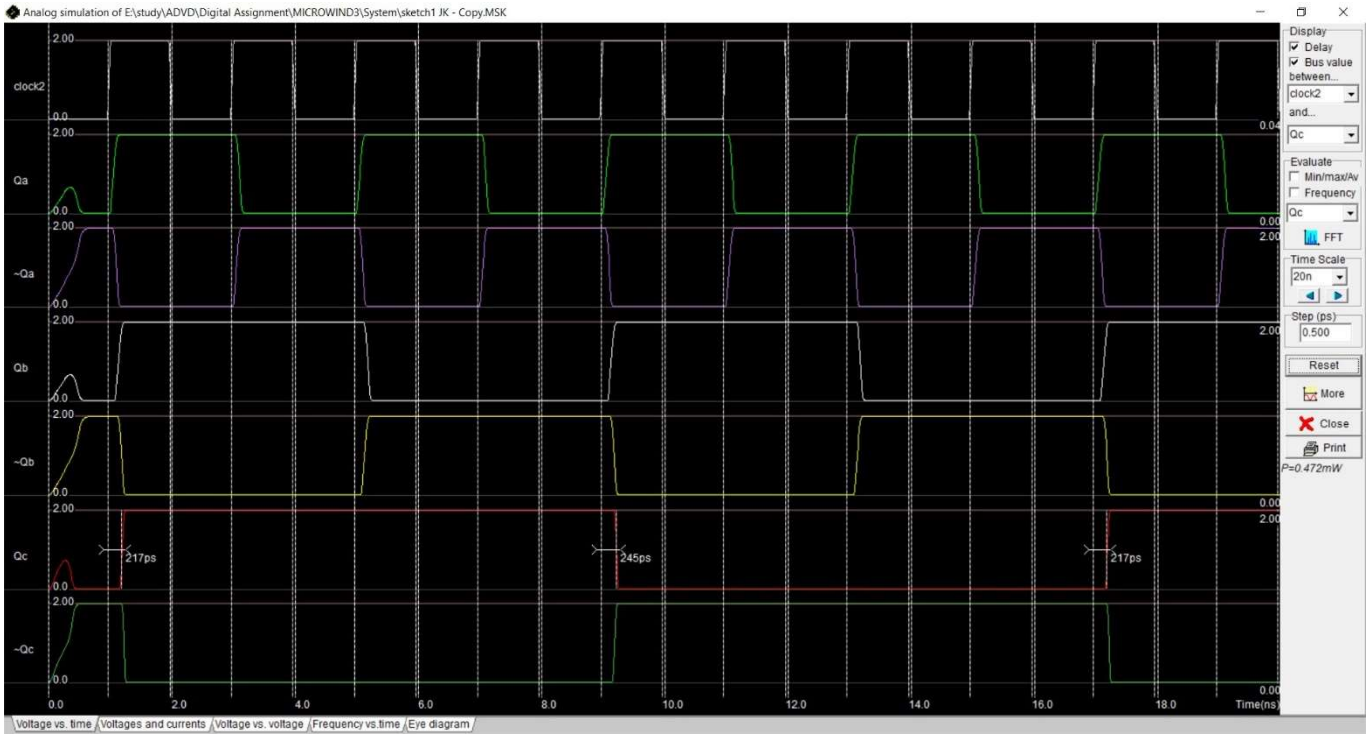
3-input NAND Gate



2-input NAND Gate



Waveform of Clock Divider Circuit

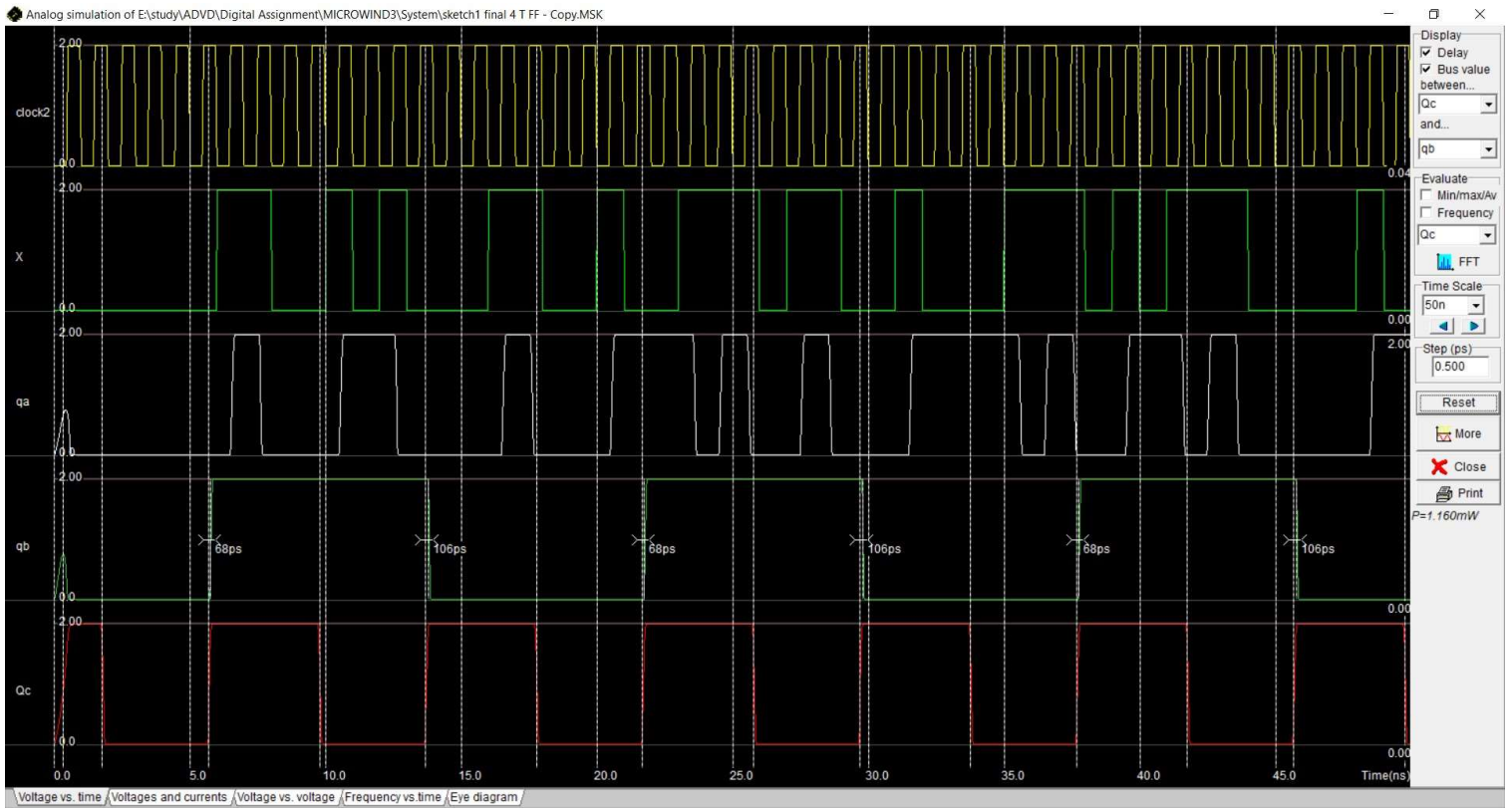


Q_c is fed to the D flip flop in the top right view, so that it captures the parity after every 8 bits.

Three T flops made entirely using 2-input NAND gates are used to divide the given clock frequency (1 GHz) by 8.

Clock2 is the given clock.
Q_c is fed to the D flip flop.

Final Timing Diagram



Clock2: given clock (1 GHz)

X : input stream

q_a : Output of top right T flip flop (Single odd parity detector)

q_b : Final output from D Flip Flop

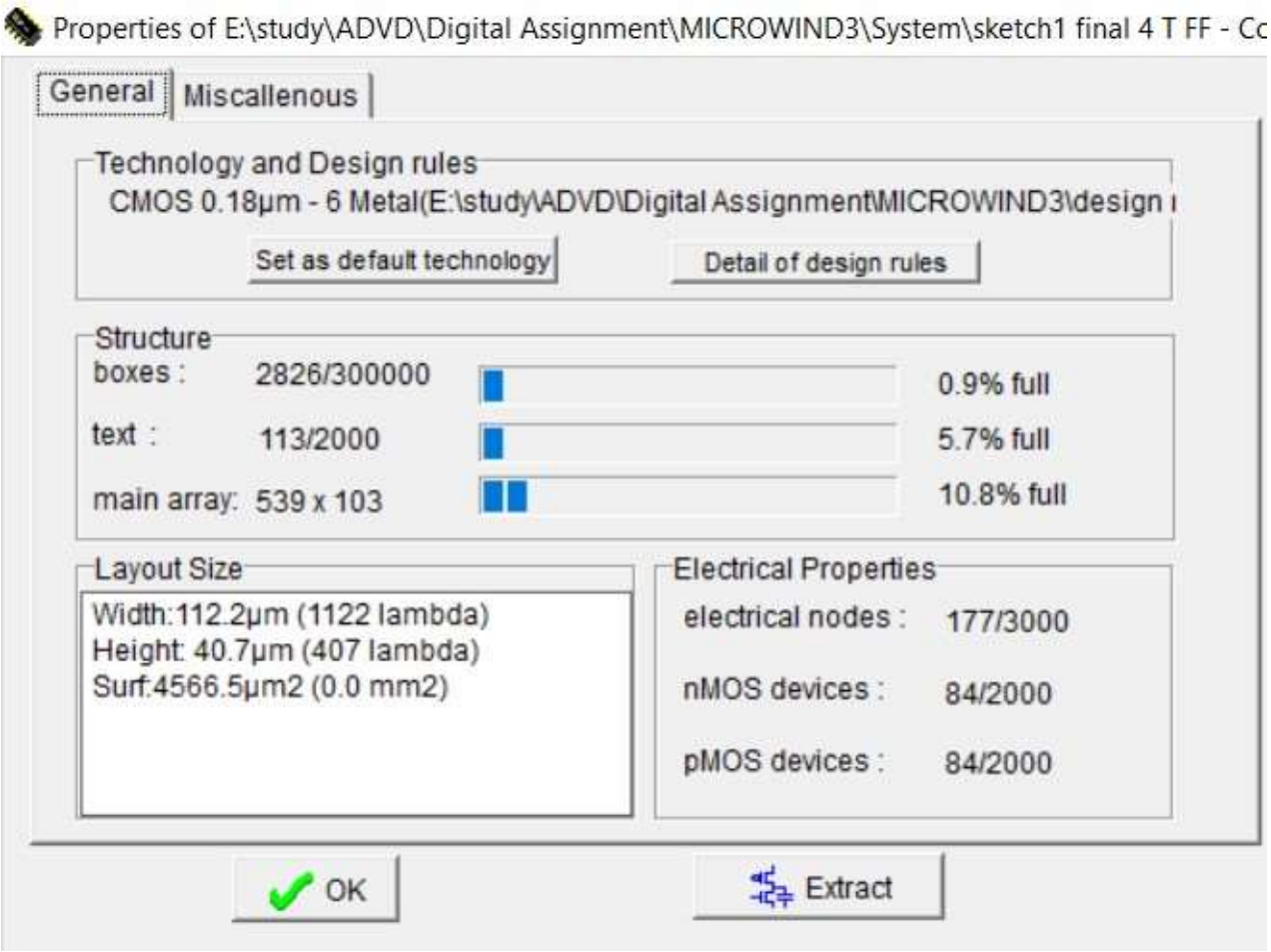
Q_c : Output from clock divider circuit (input to D Flip Flop)

Due to some internal delays and matching purposes, the checking starts from t=6ns. Subsequently, we get positive edges of Q_c at t = 14ns, 22ns, 30ns, 38ns, 46ns.

It can be seen that q_a is single parity detector and its value is '1' number of ones in X up to that point is odd.

It is also clear from here that whenever Q_c gets a positive edge, q_b takes the value of q_a, and this is how we get the parity value after every 8 clock cycles of clock2.

CONCLUSION



Problems Faced

The biggest problem faced is that the gate count is huge. We explored a wide range of architectures to implement the T flip flop in lesser number of gates, but we were unable to verify its functionality in DSCH. Hence, we had to go for the NAND based architecture.

Due to large size, Microwind got extremely slow and started to hang occasionally. So we tried and made the design in parts and imported them to keem the size of working file small

We discovered that the same circuit giving correct waveforms in Microwind, gave wrong results in DSCH simulation. So, relied on our theoretical concepts and went on to build the layout which fortunately ended up giving the desired result.

In NAND based architecture also, because of large W values in nMOS of 3-input NAND gate, we had to design a multi finger layout but keep the total area used minimum, so we adjusted one nMOS region in line with the pMOS region making a very efficient design.

INNOVATION/ OPTIMIZATION

A general approach for an 8 bit parity checker would be to XOR all the bits, and the output is '1' if input has odd number of bits. XORing can be linear or tree structure. Although gate count is same but delay is reduced in tree structure. For this combinational circuit, we would have to change all the 8 inputs manually and then check the output.
But we wanted a sequential design where the input keeps coming and we continuously get the output without doing anything.
So, we decided to pass the input through a flip flop and hence, at every clock pulse, we would get new input which would go in the combinational XOR circuit and give us the output before the next clock pulse.

Now there were two option – parallel input or serial input.
For parallel input, we would have required 8 Flip Flops and 7 XOR gates. But, in our design where we used serial input bit stream, only 5 Flip Flops are required. Hence, we saved a lot of hardware by going for serial input approach.

But using serial input means that we have to use a single bit parity checker (T Flip Flop), and we wanted an 8-bit parity checker! So, we decided to capture the value of this T Flip Flop using a D flip flop after every 8 clock pulses. In this way, the

final output from the D flip flop is nothing but 8-bit cumulative parity detector.

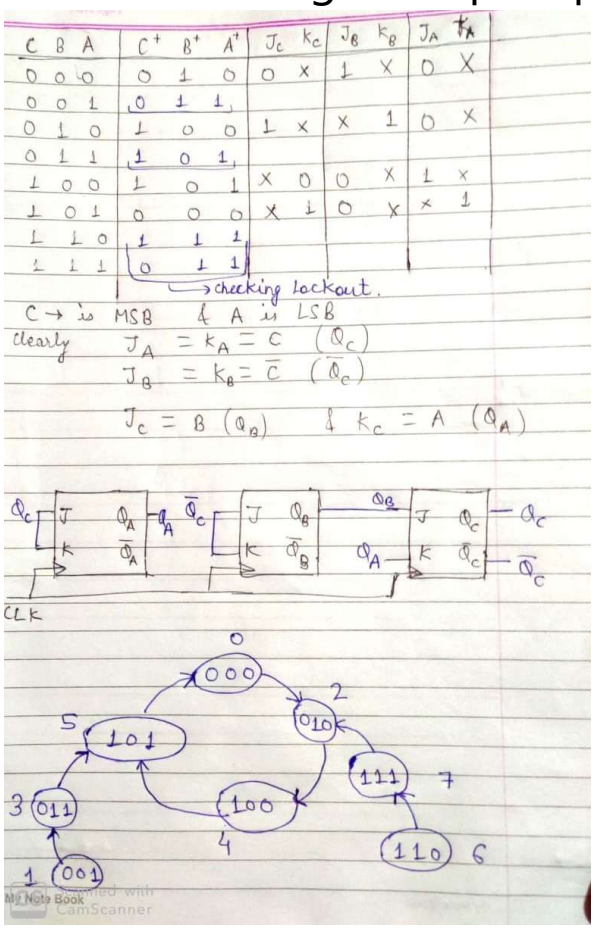
For this to happen, the positive edge at the D flip Flop must arrive after 8 clock pulses and hence we divided the original clock frequency by 8 using 3 T flip flops in series.

Optimizations in Layout

1. All the distances between metal wires were kept minimum (satisfying design rules)
2. The multi finger arrangement in 3-input NAND gate is made such that minimum area is used. We used euler's path and then further tried to do maximum sharing of source and drain regions and came up with an area efficient design.

PART-B

Design a synchronous counter which counts the sequence 0-2-4-5-0 using J K Flip Flop. Avoid Lock out.



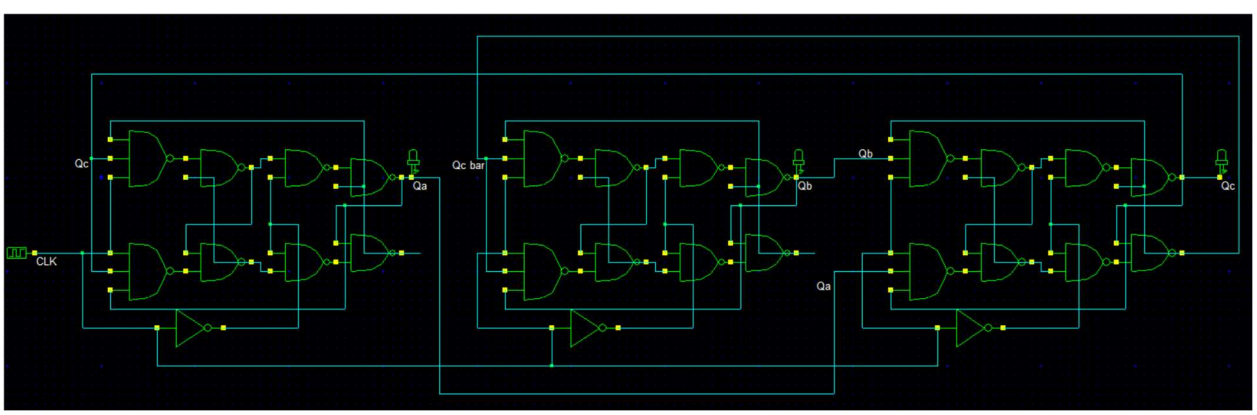
C B A	C^+ B^+ A^+	J_C K_C	J_B K_B	J_A K_A
0 0 0	0 1 0	0 x	1 x	0 x
0 0 1	0 1 1			
0 1 0	1 0 0	1 x	x 1	0 x
0 1 1	1 0 1			
1 0 0	1 0 1	x 0	0 x	1 x
1 0 1	0 0 0	x 1	0 x	x 1
1 1 0	1 1 1			
1 1 1	0 1 1			

checking lockout.

C → is MSB & A is LSB

clearly $J_A = K_A = C \ (Q_C)$

$J_B = K_B = \bar{C} \ (\bar{Q}_C)$

$J_C = B \ (Q_B)$ & $K_C = A \ (Q_A)$

Gate level Schematic :