Table of Contents

0. General Information

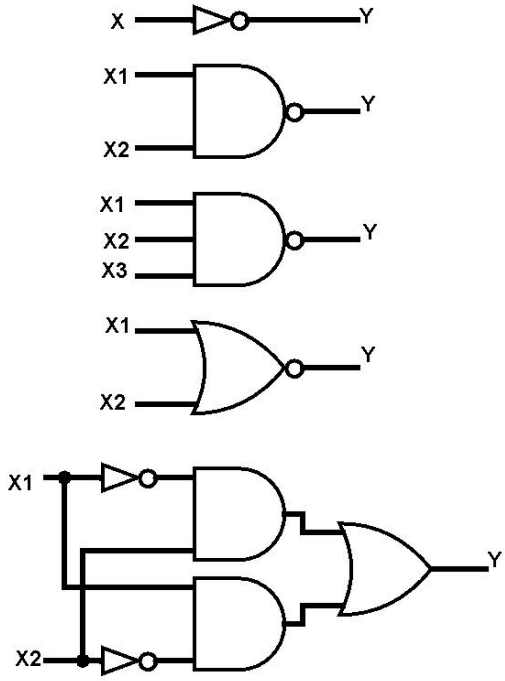
There is a folder for all the seven parts of the homework. The folders contain the Verilog modules and the test bench files. In addition to these I've also included the images of the manually drawn schematics (using Logisim) of the modules. The RTL and Technology schematics that are created after synthesis by Xilinx reside in the Schematics folder that exists in each of the seven folders except the last one since UDP files are not synthesizable as far as I know. Lastly "_test" suffix is added to the names of the test bench files. For example if the module is called "fulladder1bit.v" then the corresponding test bench file is called "fulladder1bit_test.v". In some parts I've used a single test bench for multiple modules. I've stated them explicitly in the comments section of the appropriate part. Synthesis summaries are in another folder called "Synthesis Summaries". I've also added simulation snapshot images in the folders where you can view (it is suggested) a larger version of them. The primitive gates and priority encoder are written behaviorally, all other things are written structurally. ISIM uses tcl (not do files as in ModelSIM) files, so all the batch simulation files are included in in the "tcl files" directory.

1. Primitive Gates

1.1. Simulation Snapshots

Y_not is the output of the inverter. Y_nand2 is the output of the nand2 gate and etc. We can see that at the start the values of the outputs of the gates are unknown. They become available only after the low to high delay for inverter(2ns), nand(2ns) and nor(3ns) and after the high to low delay for xor2(5 ns) gate. As you see the input for X1 and X2 covers all possibilities. I've changed their value at every 10ns.

1.2. Schematics

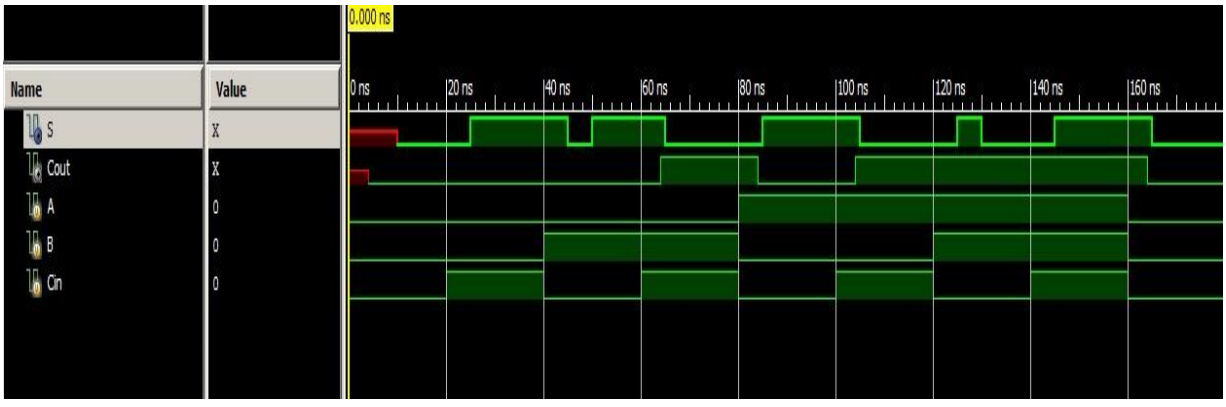


1.3. Comments

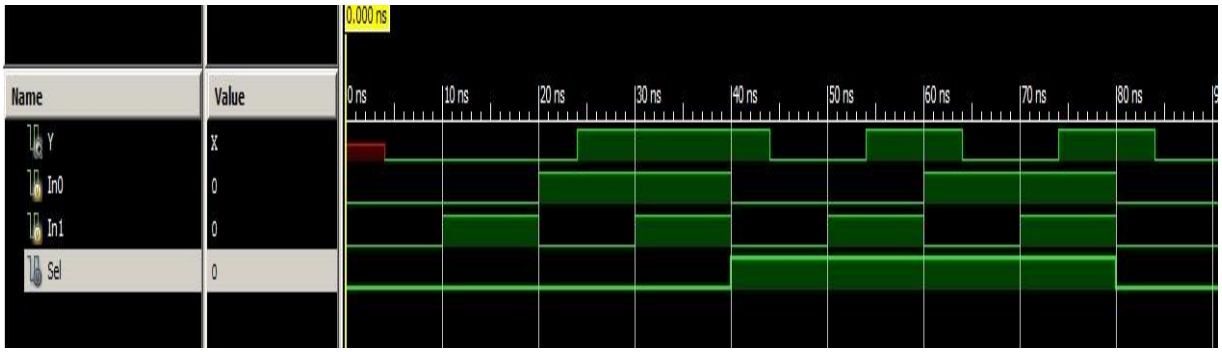
All gates are implemented as 2-input gates. Additional 3-input NAND is added which is used in the design of the D-Flip Flop. For the simulation of the gates I used a single test bench file. It instantiates all the 4 gates mentioned in the question. The 3-input NAND gate is not tested. But since I used it in the D-Flip Flop, I would have encountered problems in the simulation of the D-Flip Flop if the 3-input NAND gate had problems.

2. Adder and MUX

2.1. Simulation Snapshots

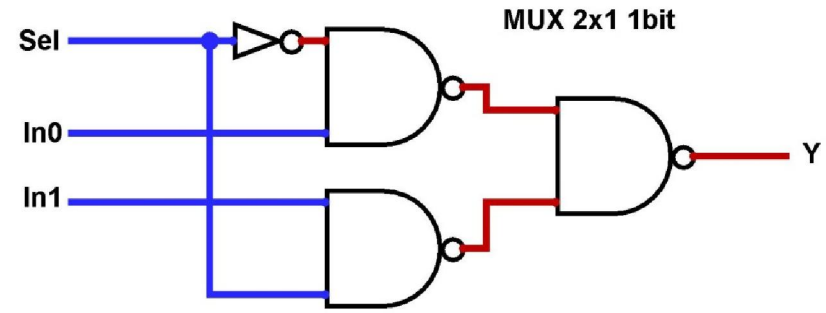


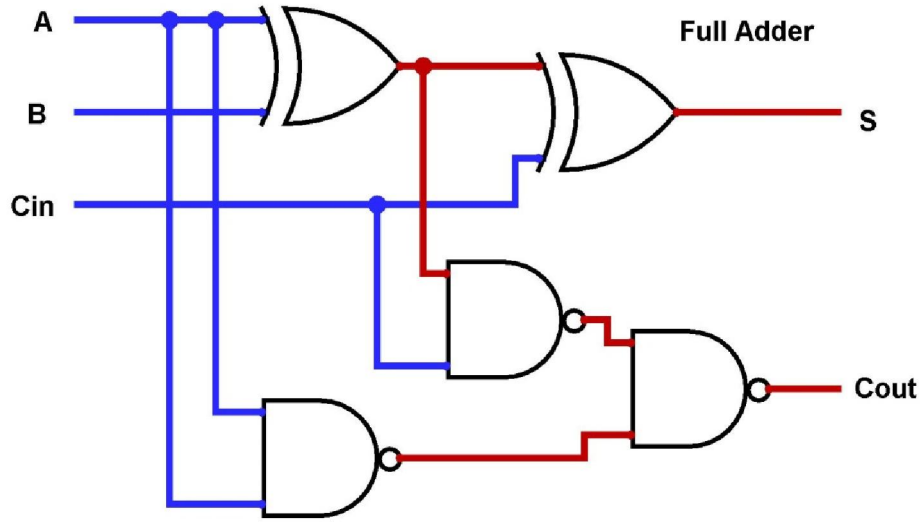
Since the sum result of the full adder has two xor2 gate delays which equals 10ns I've increased the time I vary my inputs from 10ns to 20ns for this simulation. As you can see the results at the end of the input cycles (20ns cycles) is correct. But we can see that there are some glitches at around 45ns and 125ns due to the different delays of the Sum and Carry Out paths.



This next snapshot is for the 2x1 mux. As you can see for all the different possibilities the result is correct. Since the largest possible delay is 2 nand gates and an inverter which is 6ns and I used 10ns cycles to vary the inputs, we have not encountered any problems in the results.

2.2. Schematics



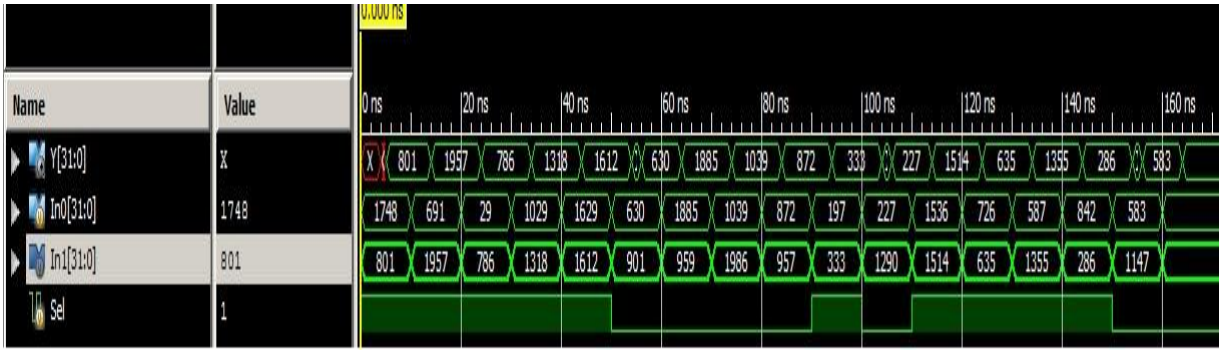


2.3. Comments

No additional comments.

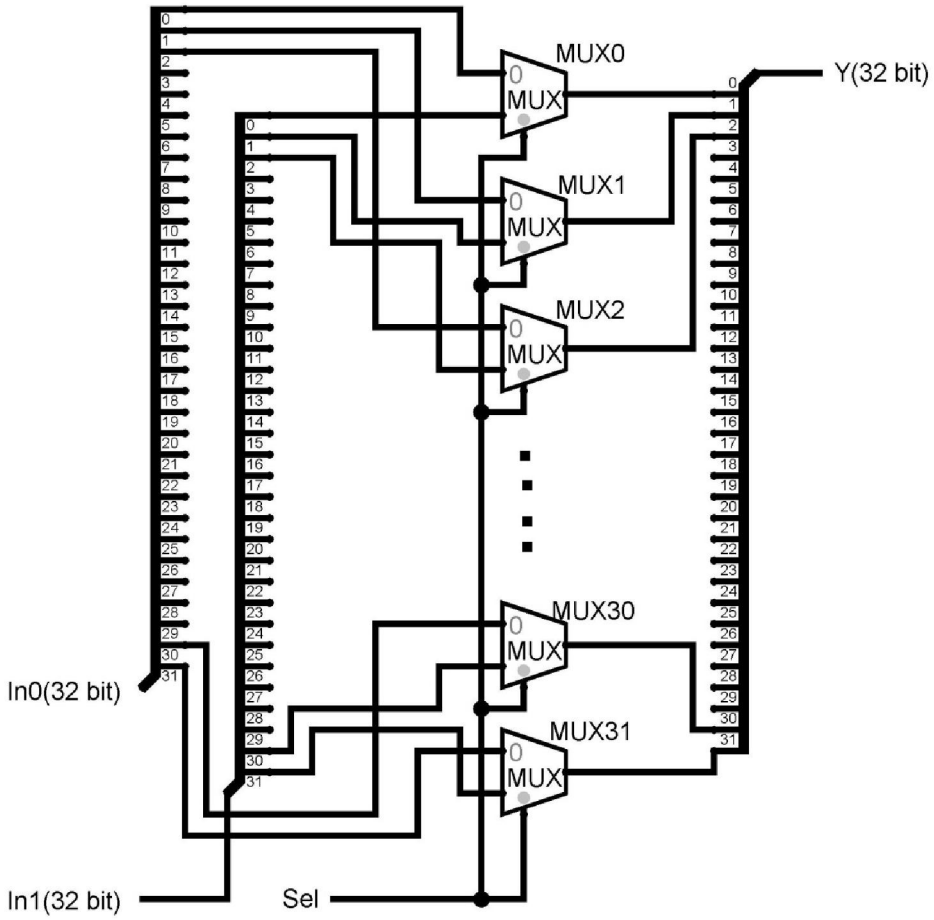
3. MUX 32-bit

3.1. Simulation Snapshots



Again here we see a correct result. The largest possible delay is 6ns as in the previous mux. Again some glitches can be seen at around 55ns, 105ns and 155ns. But they are resolved before the next cycle of data comes (Again I changed the data at regular 10ns intervals.).

3.2. Schematics

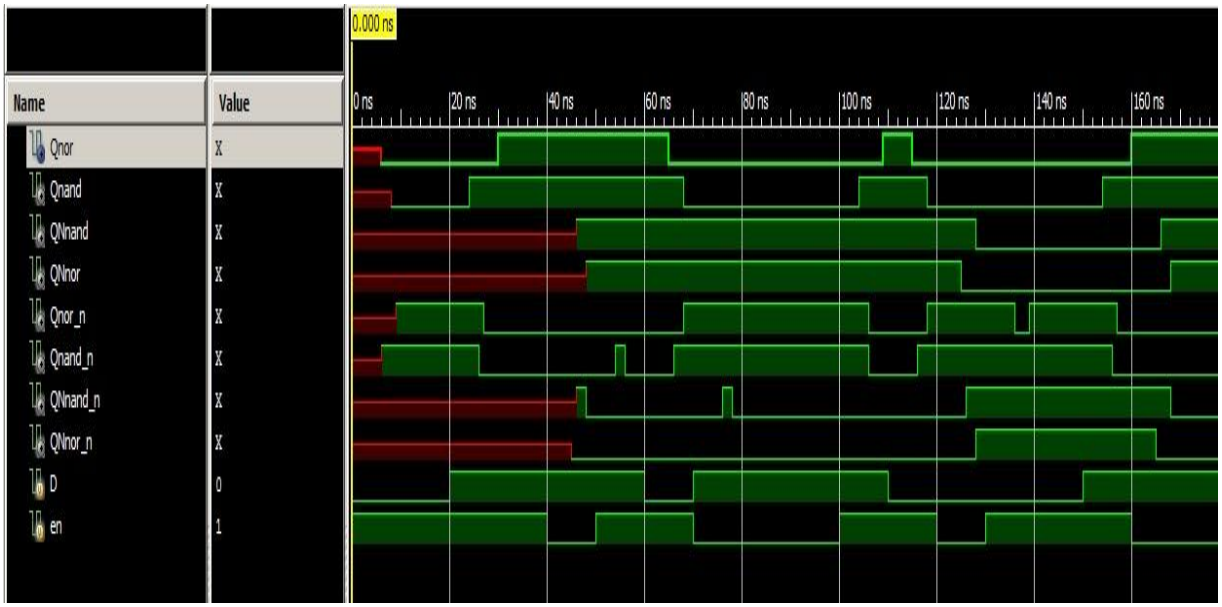


3.3. Comments

Testing all possible inputs here is not feasible. So I've used $random for generating 16 random test cases. You can change the initial seed for In0, In1 and Sel and then run the simulation multiple times to check for various other cases.

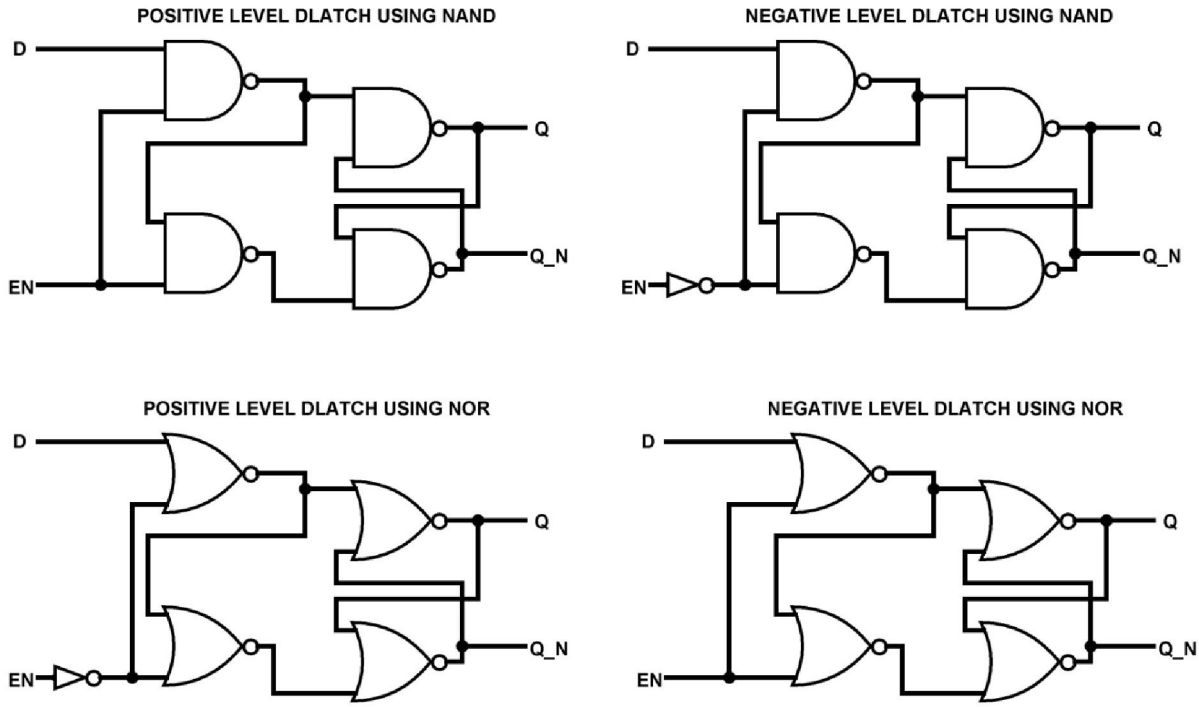
4. Gated Latches

4.1. Simulation Snapshots



I've used a single test for all the 4 latches. The Qnor, Qnand are positive level triggered latch outputs. QNnor and QNnand are negative level triggered latches. The "en" input is the trigger. _n is the suffix appended to the Q_n outputs of the latches. Here we can see the result is correct. Since the latches are level triggered and at the start "en" signal is 1 until around 40ns, we can see that the outputs of QNnor and QNnand are unknown for a long time. The outputs of the Qnor and Qnand latches are only unknown for a short duration until clock to q delay. When the "en" signal is low Qnand and Qnor retain their values, when "en" is high QNnand and QNnor retain their values as expected. But we can still see some unwanted glitchy behavior around for example 45ns, 75ns and 135ns. Edge Triggered D-Flip Flops will take care of this problem as you will see in the next section.

4.2. Schematics

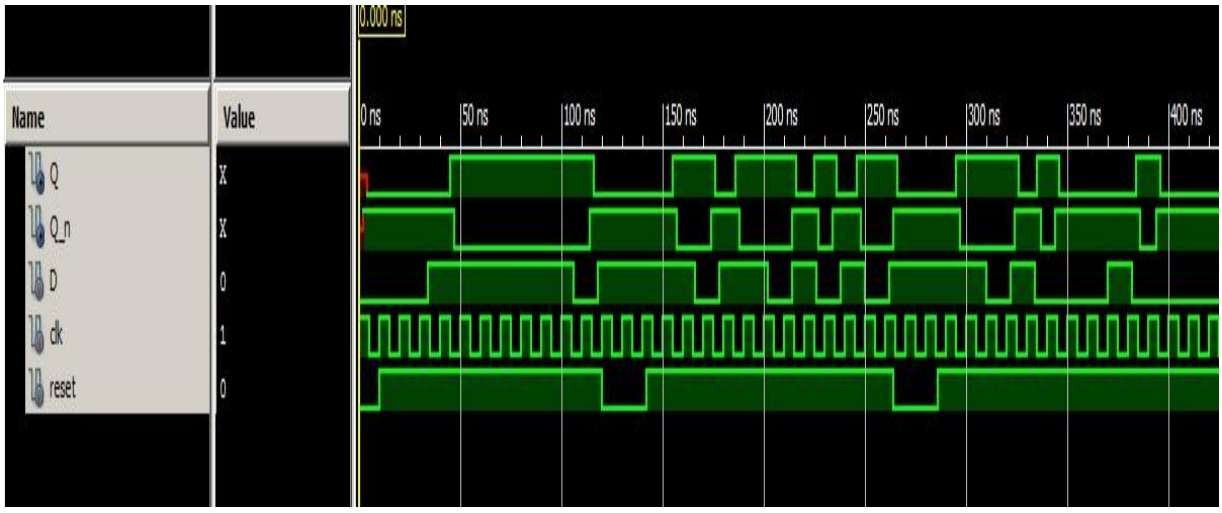


4.3. Comments

Note that the designs are for gated latches. Again here I've used a single test bench file for testing all the four different types of latches. Similar to the 32-bit mux I've used $random to generate some random cases to verify the gated latches.

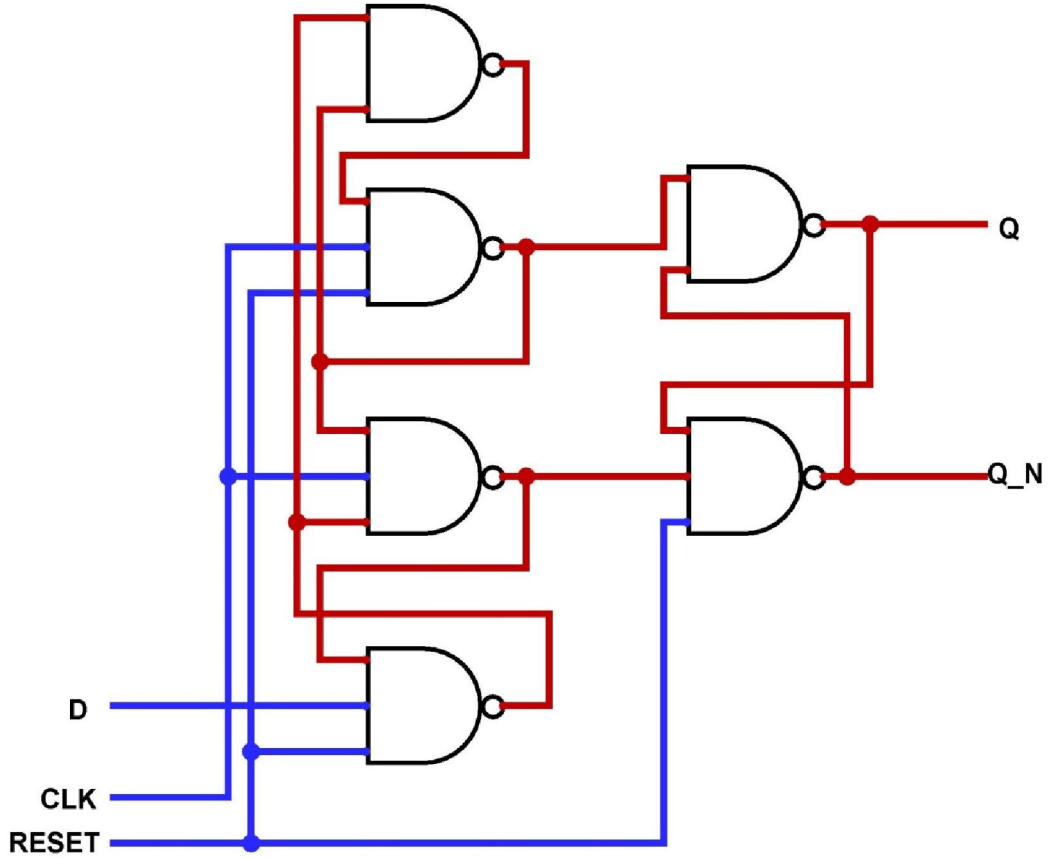
5. D-Flip Flop Positive Edge Triggered Asynchronous Reset

5.1. Simulation Snapshots



We can see that the flip flop works correctly and there are no glitches on the output. The inputs are only sampled around a small interval at the rising clock edge and the results are seen on the output of the next clock edge. Since I used 10ns full clock cycles and this is enough to cover the gate delays we don't encounter any problems with the output. We can also check that the asynchronous reset is working correctly by seeing that when reset is asserted the output changes directly to '0' without waiting for the next clock cycle and whatever the input data is. I've asserted the reset signal at two random points for this example. Rerunning the simulation will generate new random values for the D input and therefore other possible waveforms can be examined for any erroneous output.

5.2. Schematics

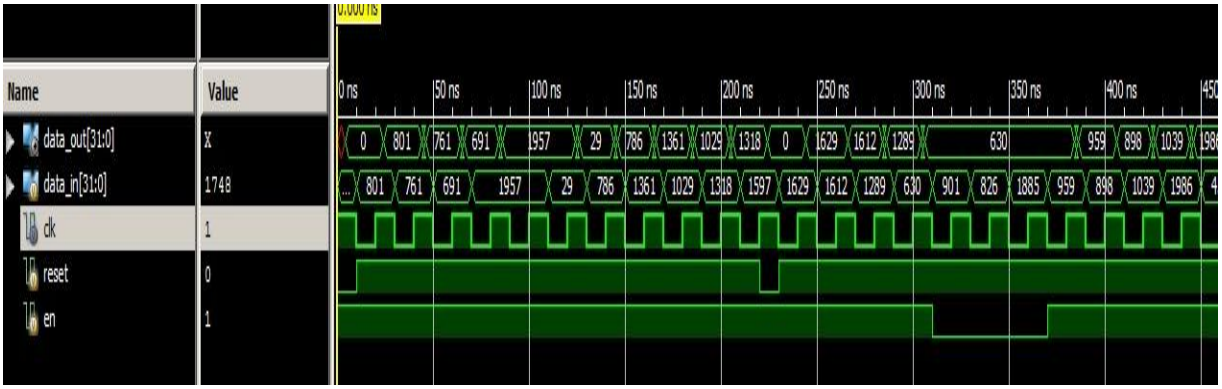


5.3. Comments

Again to simulate the D-Flip Flop I've used $random. But to verify asynchronous reset behavior I've included additional lines specifically for reset.

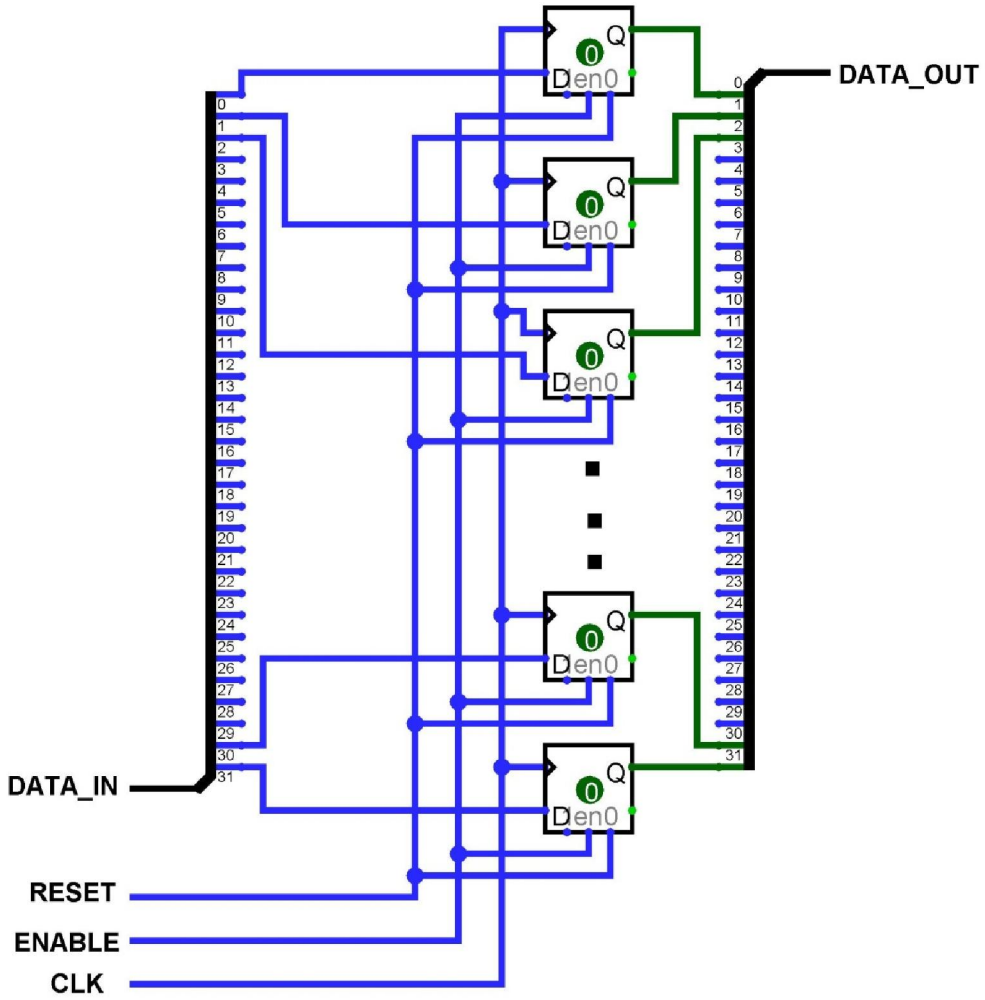
6. 32-bit Register

6.1. Simulation Snapshots



As you can see (you can see it better if you look at the jpeg file in the appropriate folder or run the simulation yourselves) the input is sampled only at the rising clock edge. Reset signal causes the output to be set to 0 irrespective of the input and when enable is low, the output stays the same, it does not get updated according to the input.

6.2. Schematics

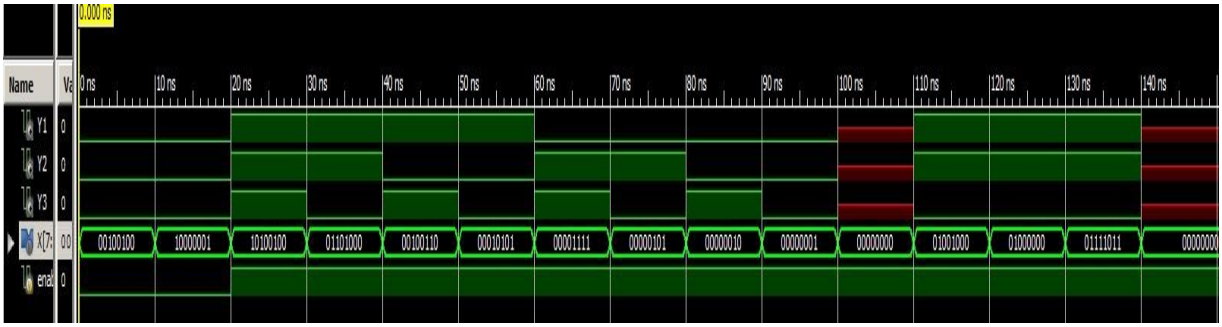


6.3. Comments

Again to simulate the 32-bit register I've used $random. Since a register also needs the enable signal I've included the design for a D-Flip Flop with enable in this folder. I used a mux with enable input as select input for the mux. If enable is '1' data_in is used as D signal for the flip flop, if enable is '0' the Q output of the flip flop is fed back to itself.

7. 8-input Priority Encoder

7.1. Simulation Snapshots



If we look at the output we can see that the results are correct. Again looking at the jpeg file in the folder would be a better idea. When enable signal is low, the output is 000. When the input is all 0's then the output is unknown as seen by the red parts in the snapshot.

7.2. Schematics

There is no schematic here since this is implemented as a UDP.

7.3. Comments

As far as I know UDP files can't have vector outputs or more than a single output. So I implemented three tables called prioencoder8x3bit1, prioencoder8x3bit2, prioencoder8x3bit3 for the first, second and third bits of the output of the priority encoder respectively. Including the enable bit for the encoder the number of all possible inputs becomes 2^9 = 512. Therefore I only used representative cases in the test bench. Output when enable is not set is set to '0'. Output when all bits are 0 is X (unknown). The remaining inputs work as expected from a priority encoder. The first index with bit set to '1' determines the output. The remaining lower indices are "don't cares".