Connor Prior
Nahom Solomon
Seth Iris Canonigo
Tedla Boke

MIPS Register File and ALU Report

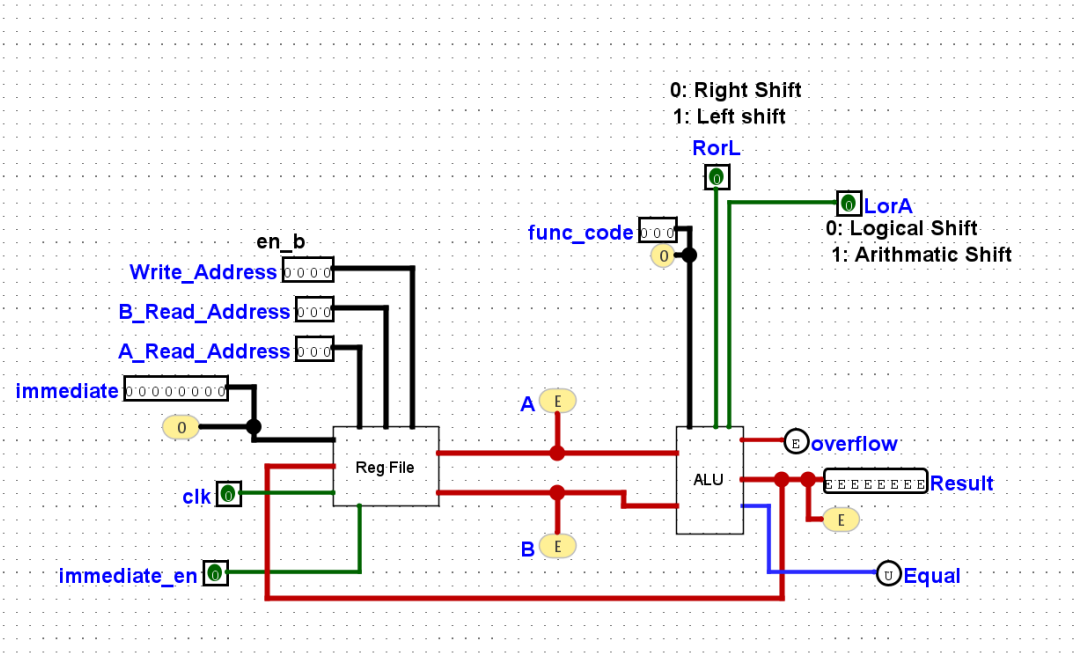
The purpose of this project is to create a calculator that stores 8-bit numbers into a register file, runs two numbers from that register file through an Arithmetic Logic Unit (or ALU), and outputs the result. The two primary blocks in this project are the Register File and the ALU. The register file will take an input and store the value into an 8-bit register, and will be able to hold values in 7 different registers.

The ALU will be able to perform the following operations:

- Add
- Subtract
- Bitwise AND/OR/NOR/XOR
- Shift Left/Right
- Comparison (greater than, less than, equal to)

The result from the ALU can also be stored back into the register file.

Top Level



The main inputs for our project are:

Immediate(8 bit): the value we want to store in the register file.

Write_Address(4 bit): the most significant bit (MSB) will determine whether or not we right to the register and the last 3 bits will store the address of the register we want to write to.

A_Read_Address/B_Read_Address(3 bit): the addresses of the registers we want to run through the ALU.

Immediate_en(1 bit): chooses to store Immediate OR Result at the write address

Clk(1 bit): enables us to write the immediate/result on rising edge

func_code(3 bit): The code for the operation we want to run using the two inputs to the ALU.

RorL(1 bit): when using shift, chooses whether to shift left or right.

LorA(1 bit): when using shift, chooses whether to shift logical or arithmetic.

The main outputs are:

Result(8 bit): the result of the operation given by func_code after running the values from the register file through the ALU.

Overflow(1 bit): signifies if an add operation produced overflow.

Equal(1 bit): signifies if the two inputs A and B are equal

Register File

The purpose of the register file in our project is to store 8 bit values for use in the ALU. On every rising edge of **clk**, depending on the MSB the value in **Write_Address**, the value from either **Immediate** or **Result** (depending on whether **immediate_en** is 1 or 0) will be stored in the register file designated by the last 3 bits of **Write_Address**. The register file does this by running the write address through a decoder to decide what register to store the value in. The value is then stored in one of our 8-bit registers which stores the value using 8 D flip flops triggered by **clk** to either hold the old value or write the new value. Our register file has a total of 8 registers, but only 7 of them(with addresses 0-7) can be written to, with the 8th register holding a constant value of 0.

The register file has two outputs: A and B. These outputs are the values stored at the register in the register file designated by **A_Read_Address** and **B_Read_Address**. These two 8 bit values will be used in the calculation done by the ALU.

Arithmetic Logic Unit(ALU)

The ALU will be used to calculate the 8 bit **Result** output based on the two 8 bit values from the registers we will call A and B. The type of operation performed by the ALU will be dependent on the value of the input **func_code**. Once the func code decides an operation to perform, the two input values will be run through the appropriate subcircuit to calculate the correct output.

We will use the following function codes and operations:

000: Addition (A+B)

001: Subtraction (A-B)
010: Bitwise AND (A and B)
011: Bitwise OR (A or B)
100: Bitwise XOR (A xor B)
101: Bitwise NOR (A nor B)
110: Shift
111: Comparison

The **Addition** operation will add the values of A and B together to produce an 8 bit result using 8 mirror adders, with an extra bit outputted separately to depict overflow. The **Subtraction** operation will be performed using the same adders in combination with XOR gates. The subtraction operation will not be able to handle negative results. The **AND/OR/XOR/NOR** operations will perform their respective combinational logic with each individual bit from inputs A and B.

The **Comparison** operation will use combinational logic to determine if A is greater than, less than, or equal to B. The output will depend on the result:

Result	Output
A > B	0000 0100
A = B	0000 0010
A < B	0000 0001

Finally, the **Shift** operation will shift the value in A a number of times based on the value in B (i.e. if the value in B is 5, A will be shifted 5 times). The direction of the shift will be determined by the input **RorL**. The type of shift (Arithmetic or Logical) will be determined by the input **LorA**. After determining the parameters for the shift operation, the value of A will be run through a barrel shifter and the shifted value will go to the output.

Also, an additional Equal block was added to check if A and B are equal.

Hierarchy:

- RegandALUv3
    - REGISTER_FILEv2
        - REG8v2
            - REGv2
                - REG8
                - DFF

                                - OAI21
                                - AOI21
                                - INV
                        - 8BitMUX2to1
                            - MUX2to1
                                - Transmission
                                - INV
                - BUFFER8v2
                    - BUFFER2
                        - INV
                    - INV
            - REG0v2
                - BUFFER8v2
                    - BUFFER2
                        - INV
                    - INV
            - 8BitMUX2to1
                - MUX2to1
                    - Transmission
                    - INV
            - Decoder_4x8
                - INV
                - NAND4
            - Decoder_INV_4x8
                - Decoder_4x8
                    - INV
                    - NAND4
                - INV
    - Equalv2
        - XOR
        - NOR
    - ALU
        - 8bitADDSUB
            - 1bitAddSub
                - XOR2
                    - INV
                - FullAddr
                - INV
            - BUFFER8
                - TRANSMISSION
                - INV
            - XOR2
            - INV
        - 8bitAND2
            - 8AND2
                - AND2
            - BUFFER8
                - TRANSMISSION
                - INV
        - 8bitOR2
            - OR2x8
                - OR2
            - BUFFER8
                - TRANSMISSION

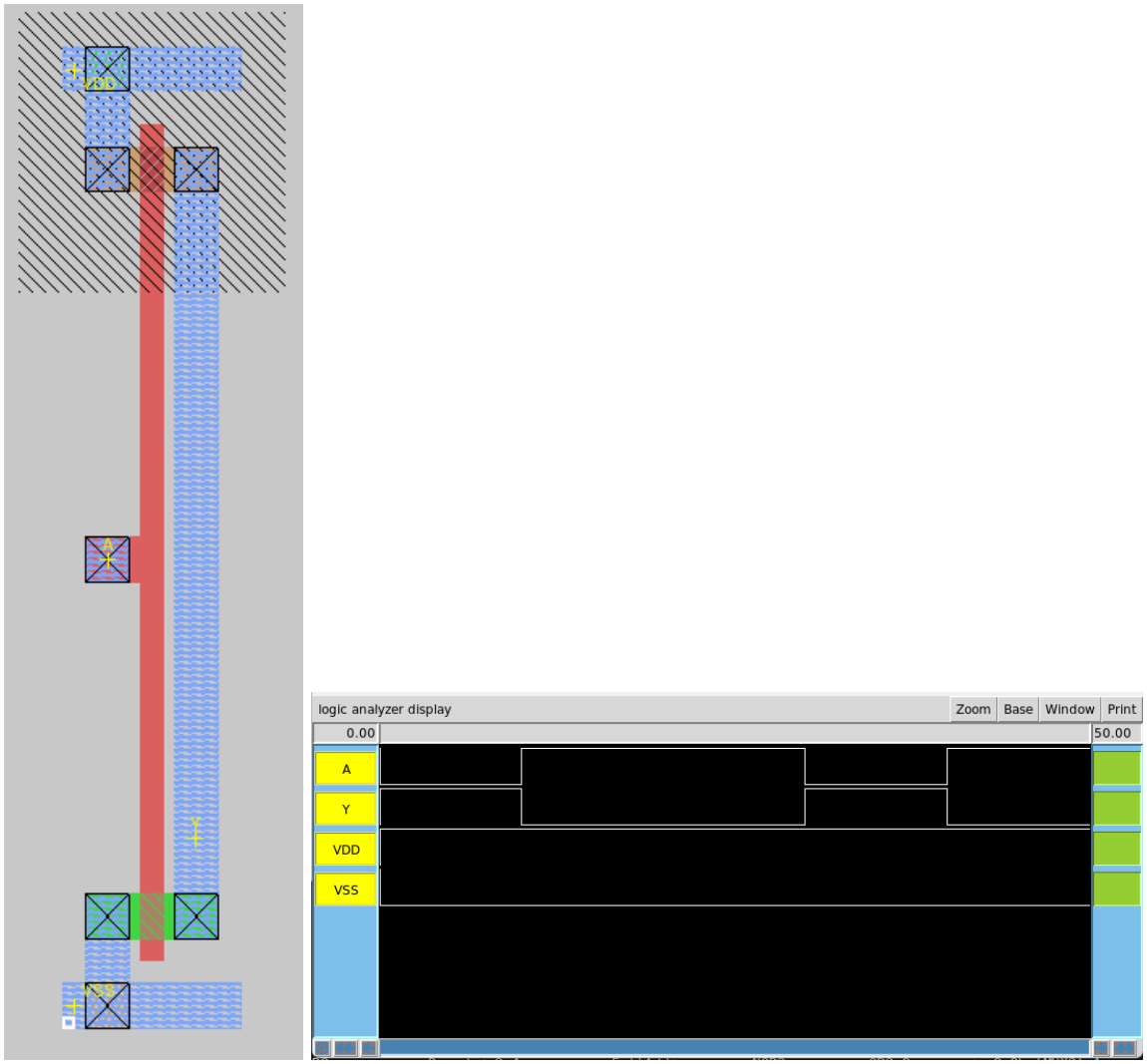
        - INV
    - 8XOR2
        - XOR2
        - BUFFER8
            - TRANSMISSION
            - INV
    - 8bitNOR2
        - NOR2x8
        - BUFFER8
            - TRANSMISSION
            - INV
    - BARREL_SHIFT
        - 8bitMUX3to1
            - MUX3to1
                - MUX2to1
                    - TRANSMISSION
                    - INV
        - AND2
        - BUFFER8
            - TRANSMISSION
            - INV
    - Comparitor8
        - Comparator4
            - Comparator1
                - AND2
                - NOR2
                - INV
            - AND2
            - NOR4
            - INV
        - BUFFER8
            - TRANSMISSION
            - INV
        - AOI21
        - AND2
    - Decoder_2x4
        - NAND3
        - INV
    - XNOR2
        - XOR2
        - INV
    - NAND3
    - INV

Total Parts List:

1. INV
2. BUFFER
3. BUFFER2
4. BUFFER8
5. BUFFER8v2
6. NOR2
7. NOR2x8
8. 8bitNOR2
9. NOR4
10. OR2
11. OR2x8

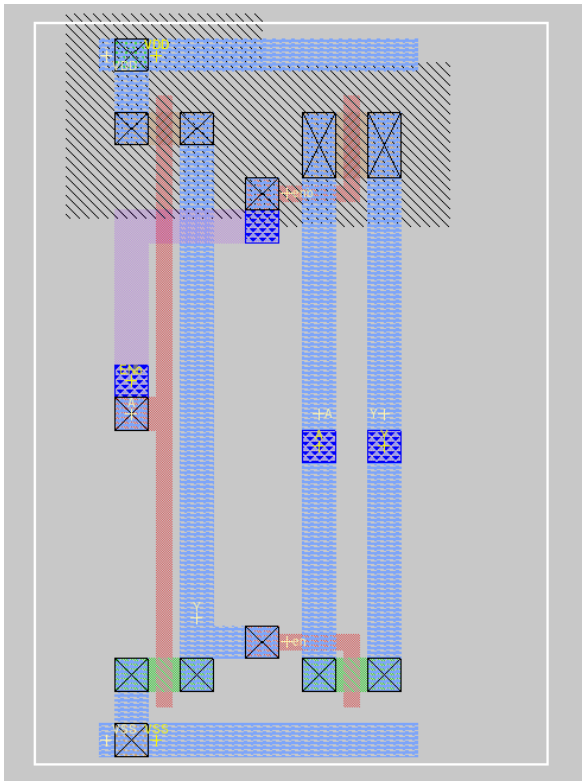
12. 8bitOR2
13. NAND3
14. NAND4
15. AND2
16. 8AND2
17. 8bitAND2
18. XOR2
19. XNOR2
20. 8XOR2
21. TRANSMISSION
22. MUX2x1
23. 8bitMUX2to1
24. MUX3to1
25. 8bitMUX3to1
26. OAI21
27. AOI21
28. DFF
29. FullAddr
30. Decoder_2x4
31. Decoder_4x8
32. Decoder_Inv_4x8
33. reg8
34. REGv2
35. REG8v2
36. REG0v2
37. 1bitAddSub
38. 8bitAddSub
39. Comparitor1
40. Comparitor4
41. Comparitor8
42. BARREL_SHIFT
43. Equalv2
44. REGISTER_FILEV2
45. ALU
46. REGandAL

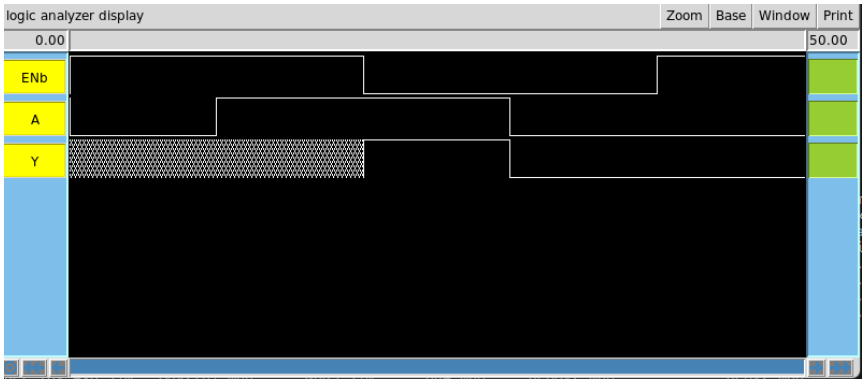
1. **INV**



Inverts the input and outputs the result

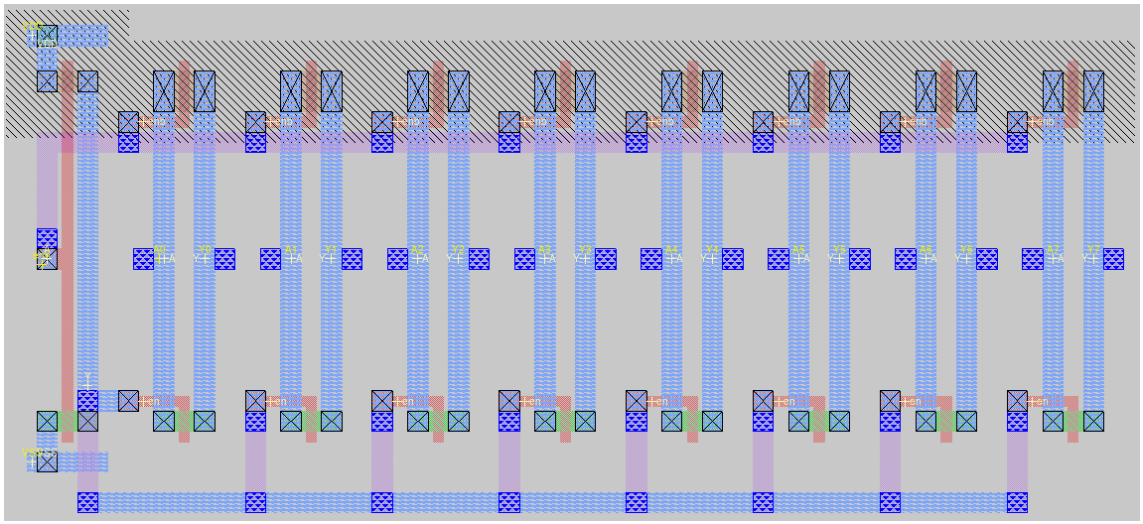
2. BUFFER:





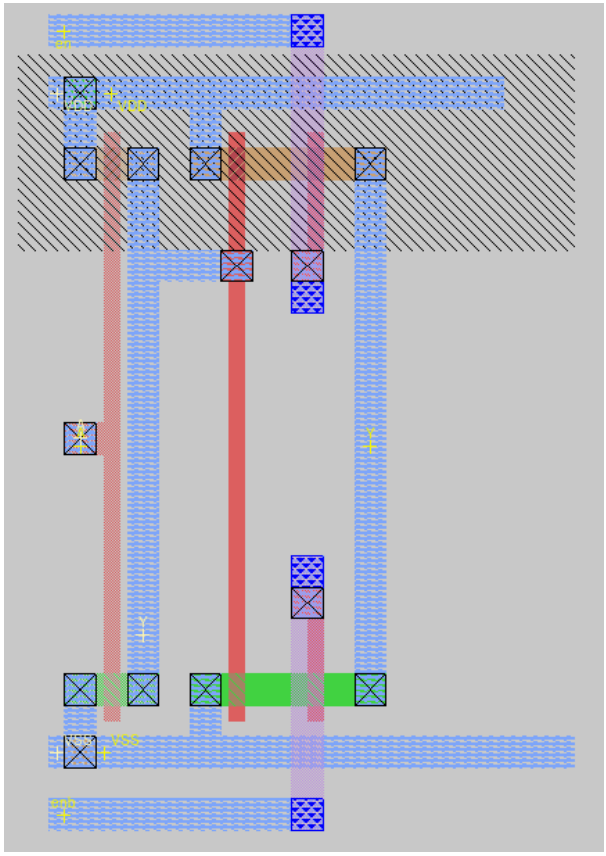
Transmission gate-based tristate buffer with an inverter that allows a signal from A to Y when ENb is off, and doesn't allow a new signal through if ENb is on.

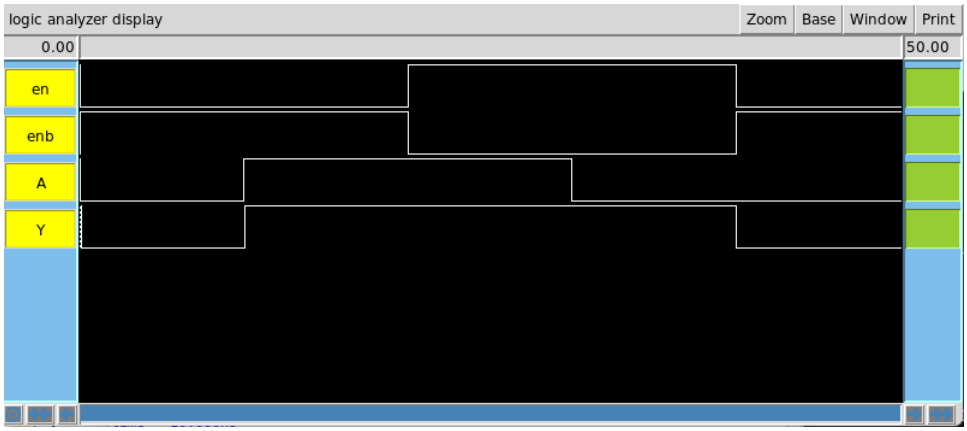
3. BUFFER8:



An array of eight 1 bit BUFFERs using transmission gates

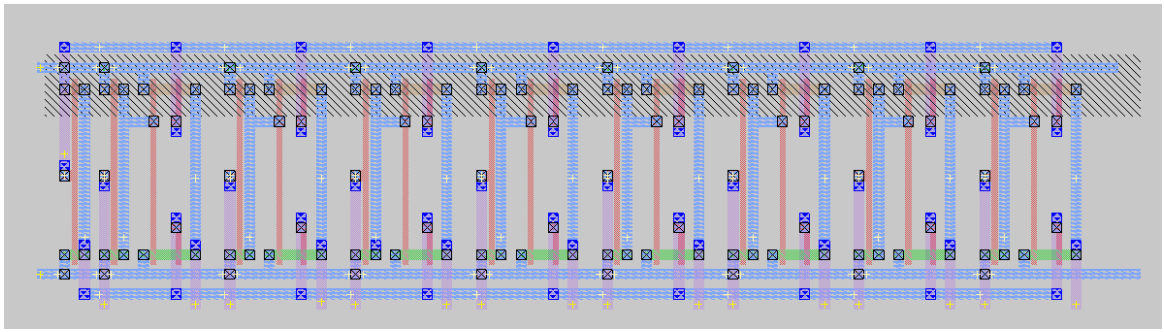
4. **BUFFER2:**

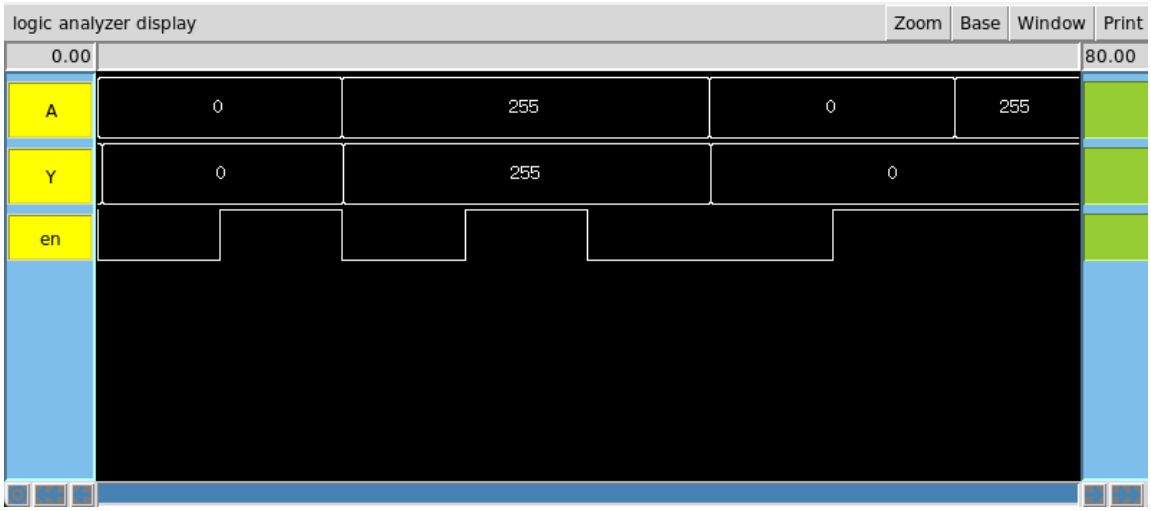




Tristate Buffer. Unlike the transmission gate buffers, it doesn't let signals connect back through from the output of the buffer when disabled.

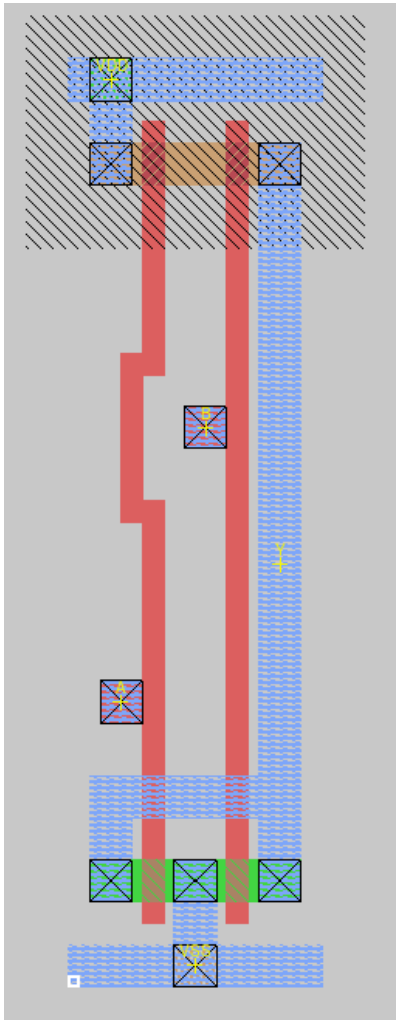
5. BUFFER8v2:

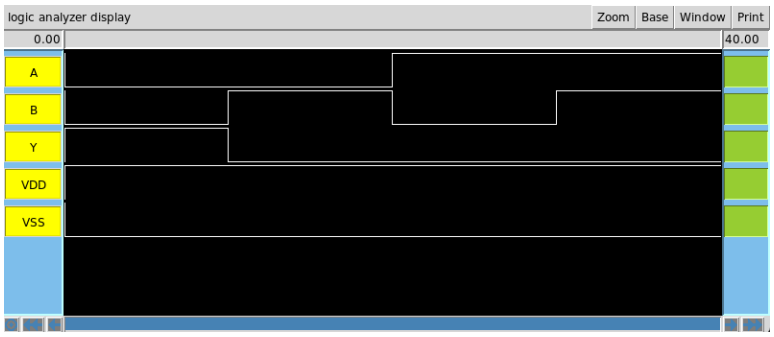




An array of eight BUFFER2s

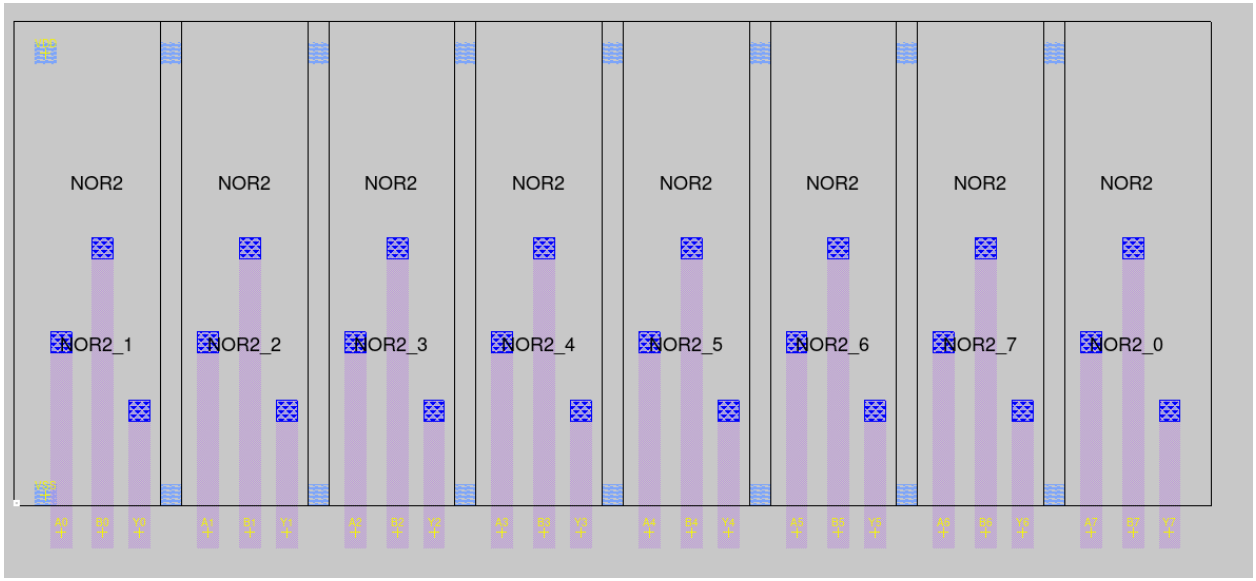
6. NOR2:

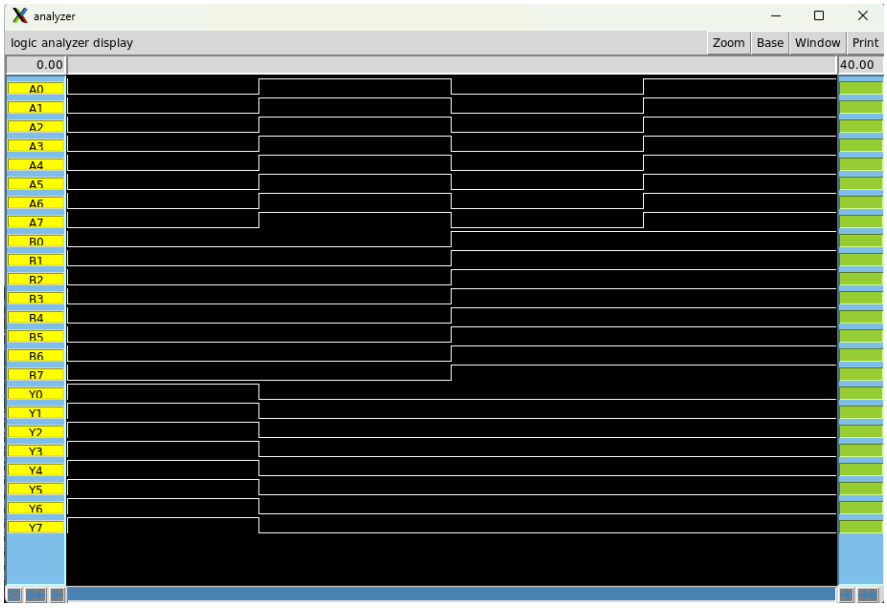




If either inputs A or B are 1, output 0. Otherwise, output 1.

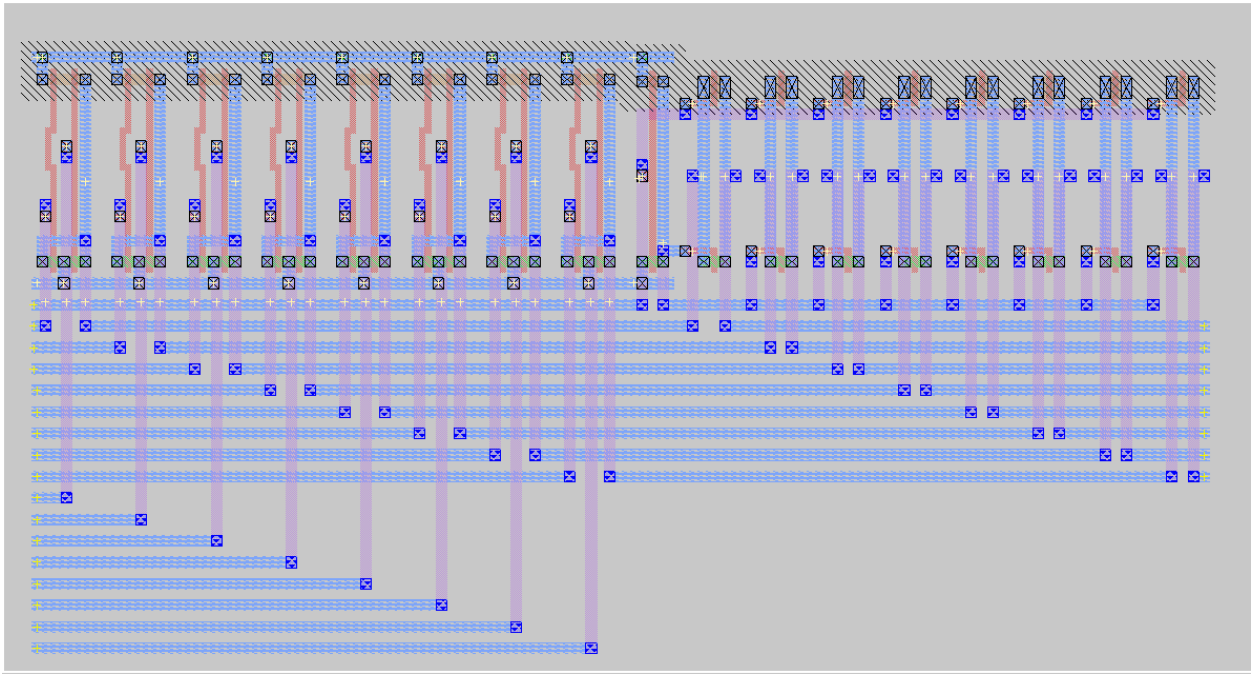
7. NOR2x8:

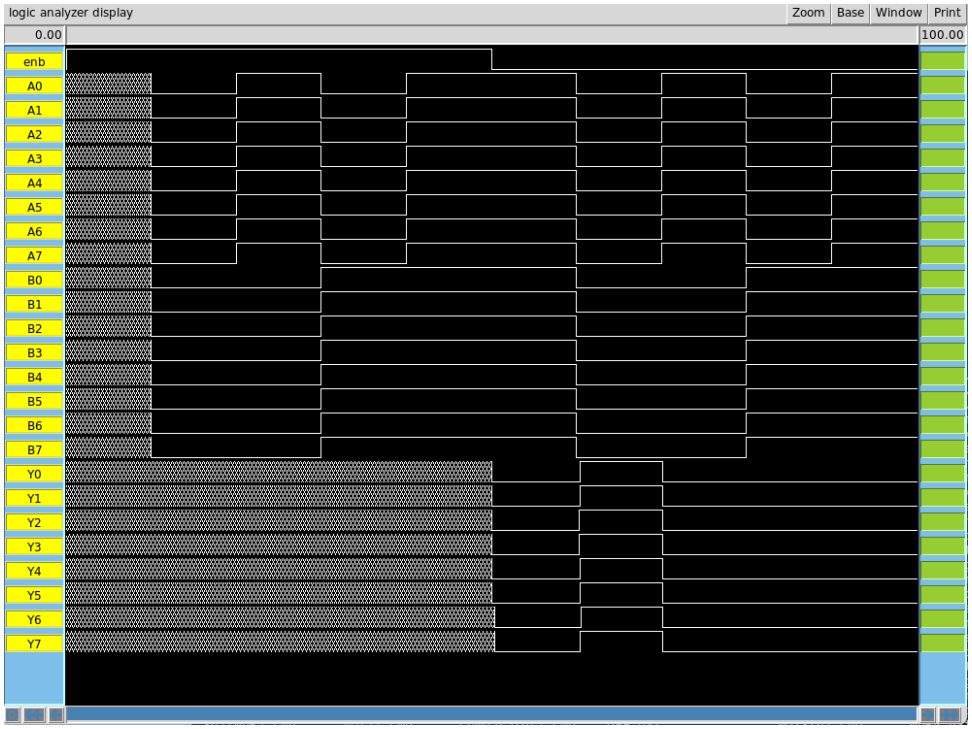




An array of 8 Nor2 Gates

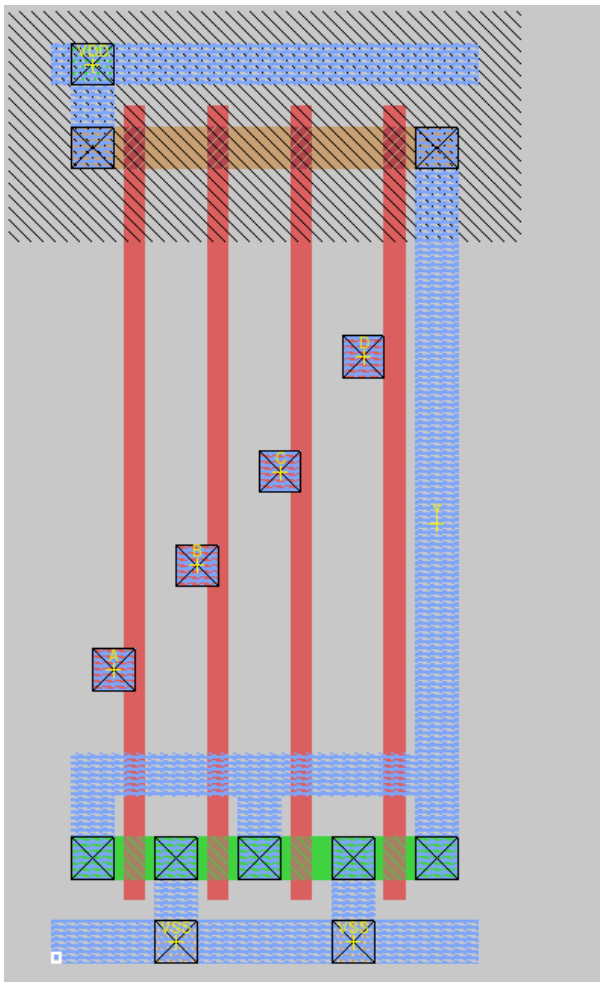
8. 8bitNOR2:

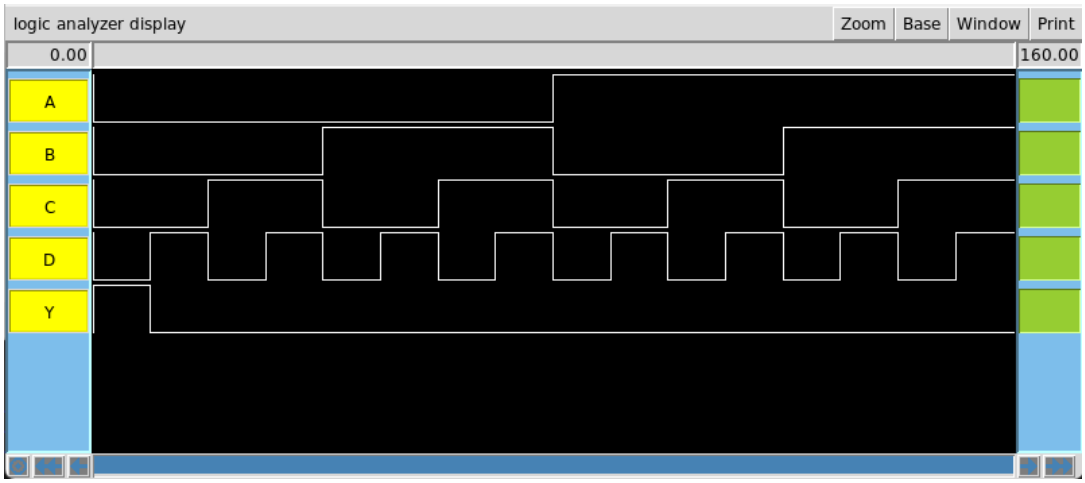




Our NOR2x8 with an 8 bit Buffer

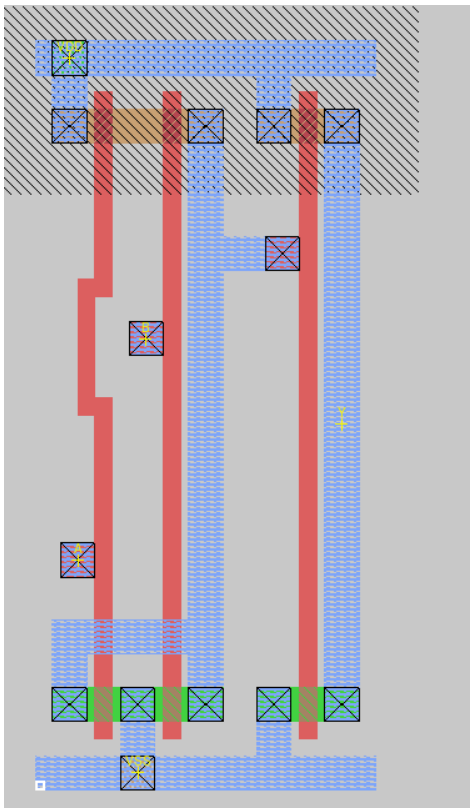
9. NOR4:





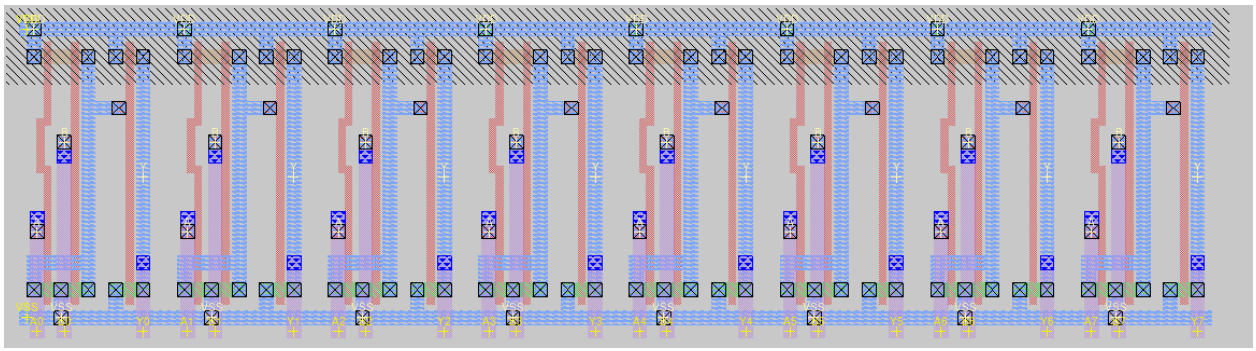
4 Input NOR Gate

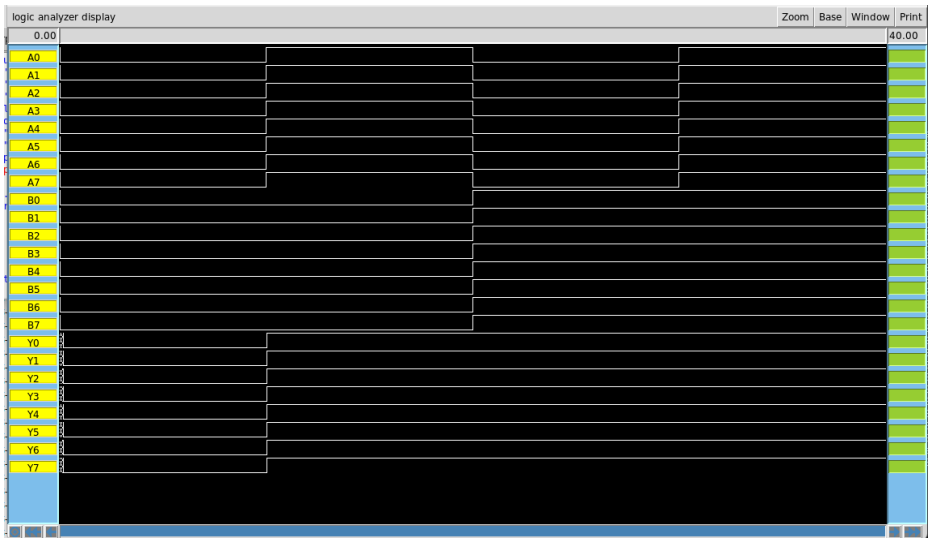
10. OR2:



2 input OR gate that inverts the output of a NOR gate. When either inputs A or B are equal to 1, output 1. Otherwise, output 0.

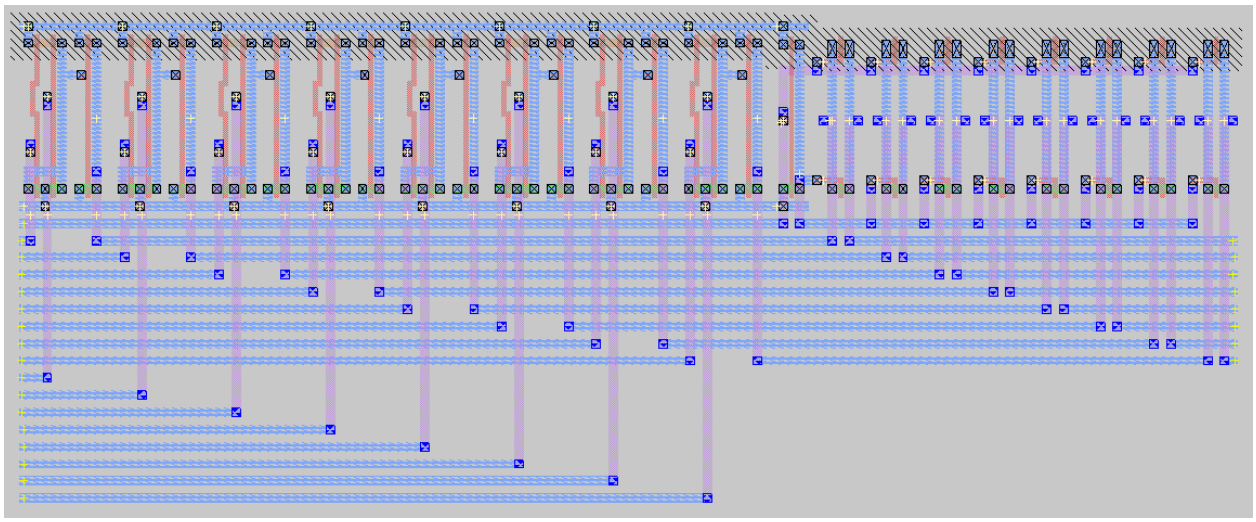
11. OR2x8:

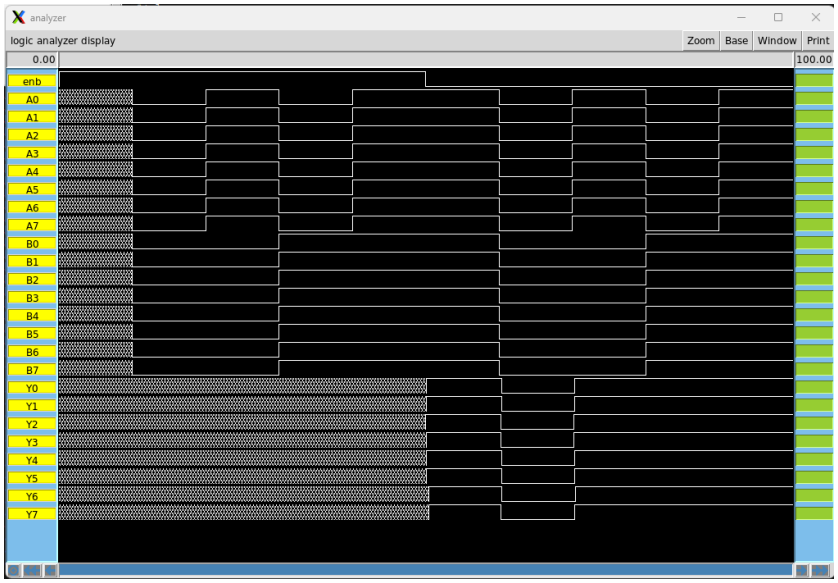




Array of 8 2 input OR Gates

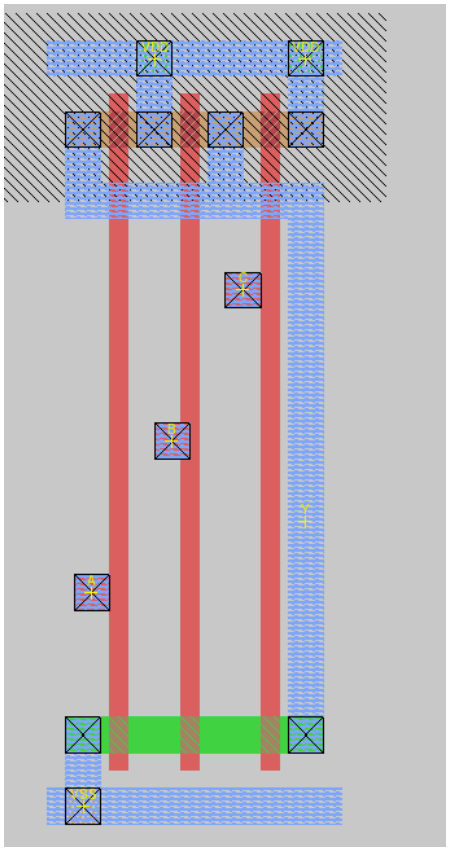
12. 8bitOR2:

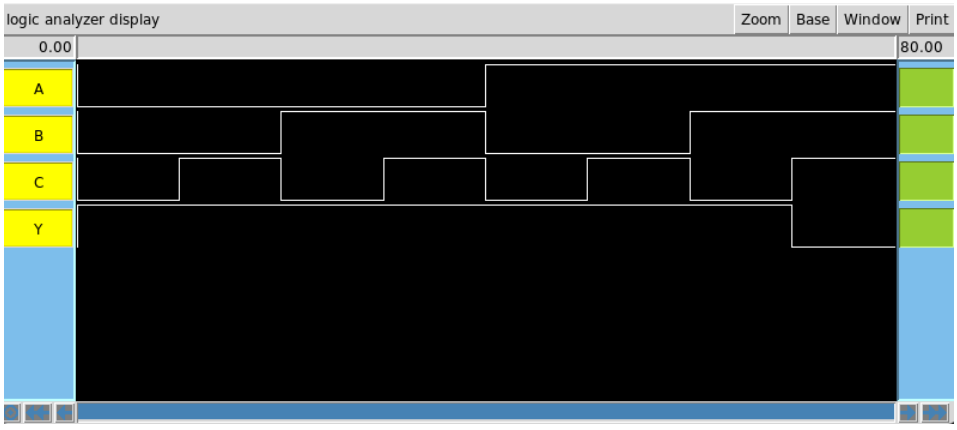




Or2x8 run through an 8 bit buffer using transmission gates

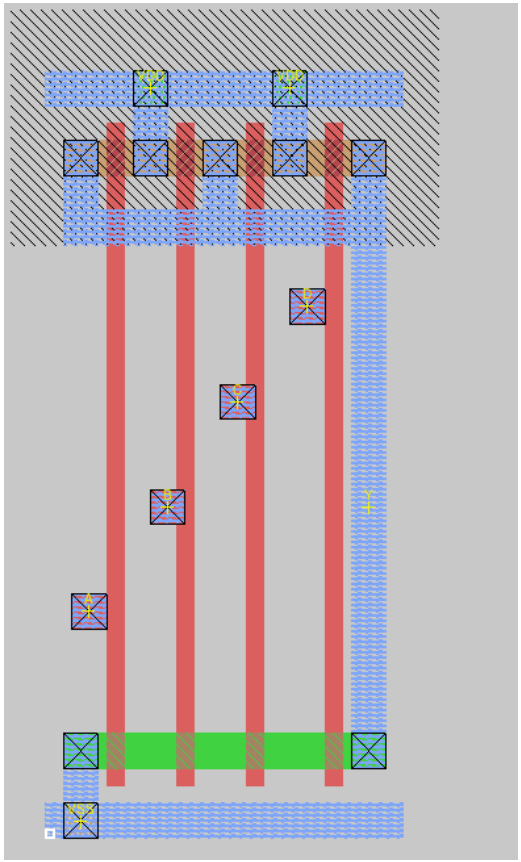
13. NAND3:

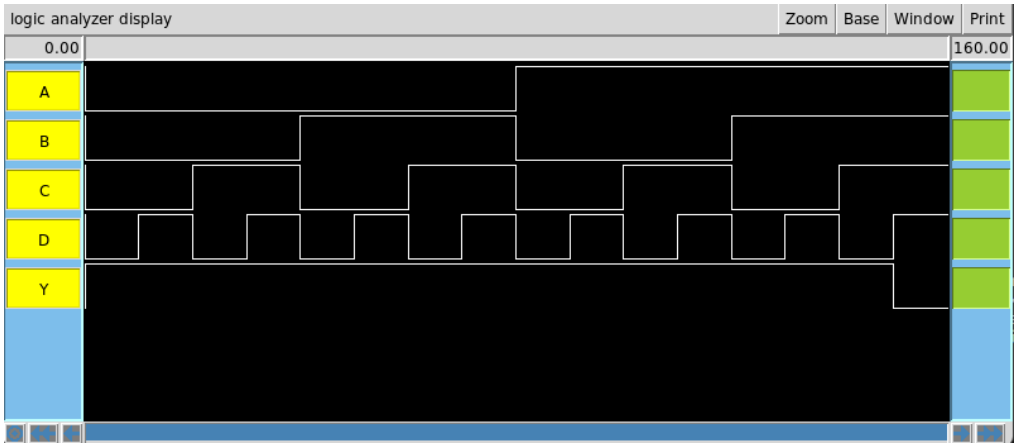




3 Input NAND Gate. If both A and B are 1, then output will be 0. Otherwise, output will be 1

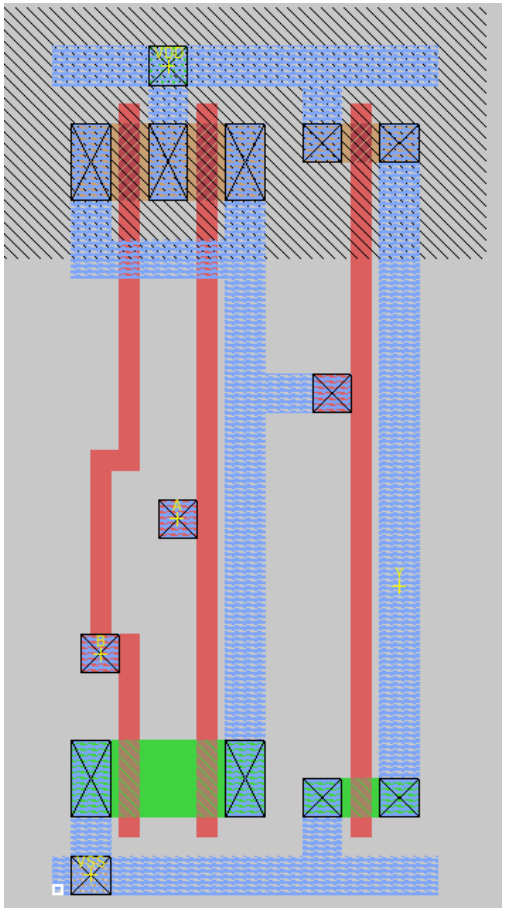
14. NAND4:

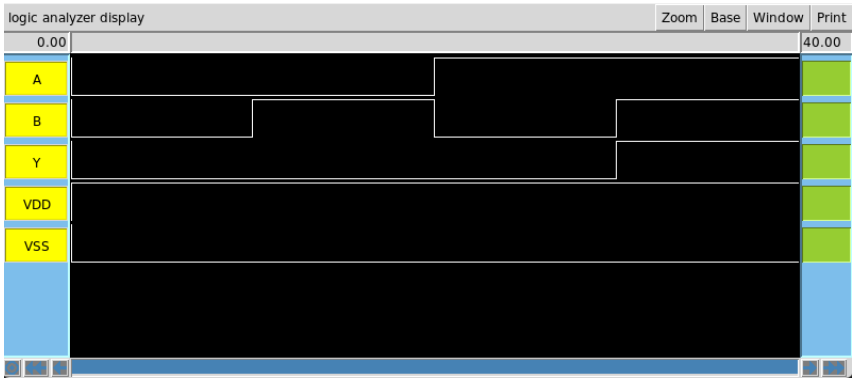




4 Input NAND Gate. If both A and B are 1, then output will be 0. Otherwise, output will be 1.

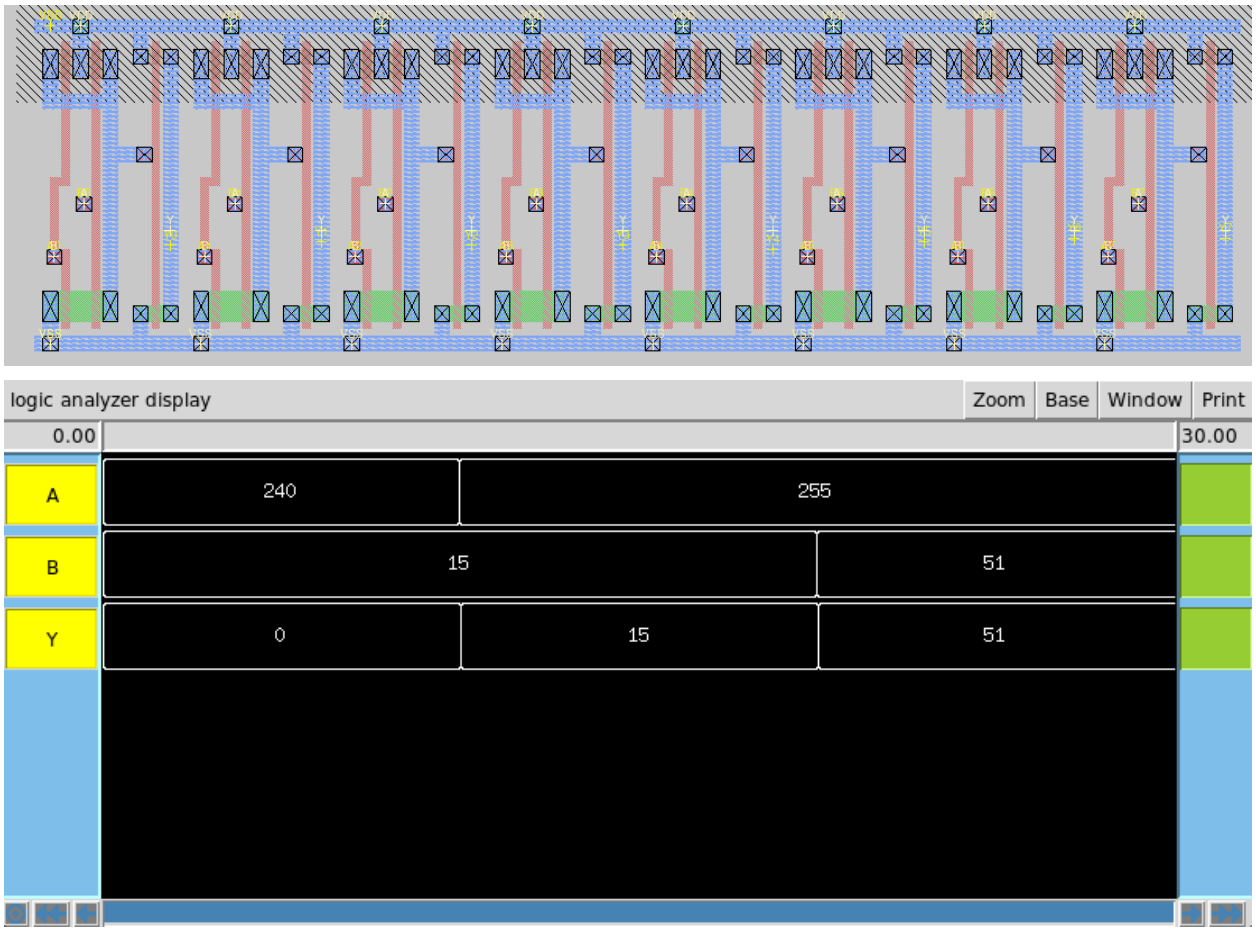
15. AND2:





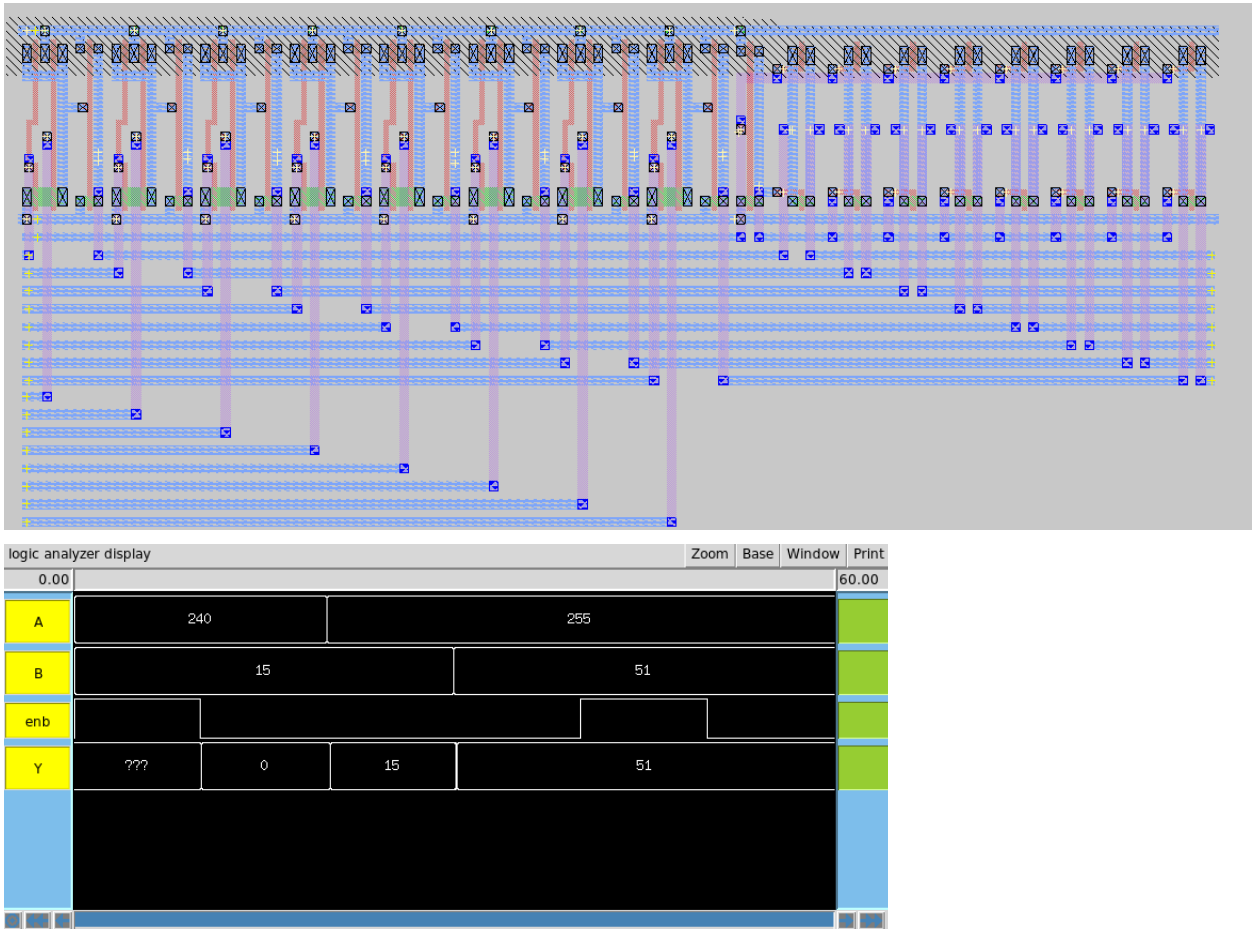
2 input AND gate made by inverting a NAND gate. If both A and B are 1, then output will be 1. Otherwise, output will be 0.

16. **8AND2:**



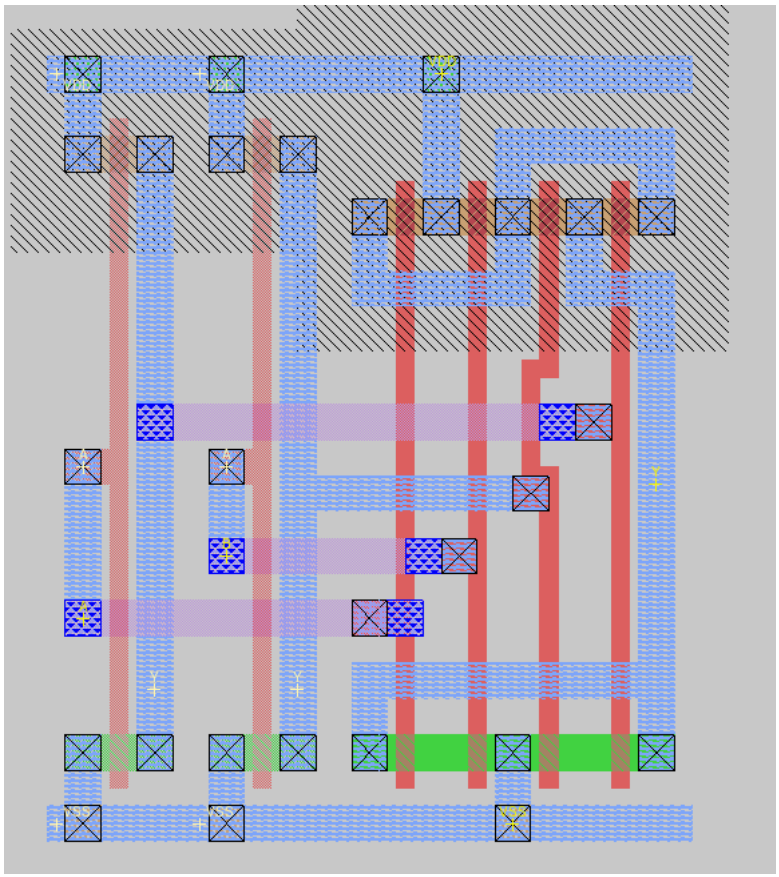
Array of 8 AND2 Gates

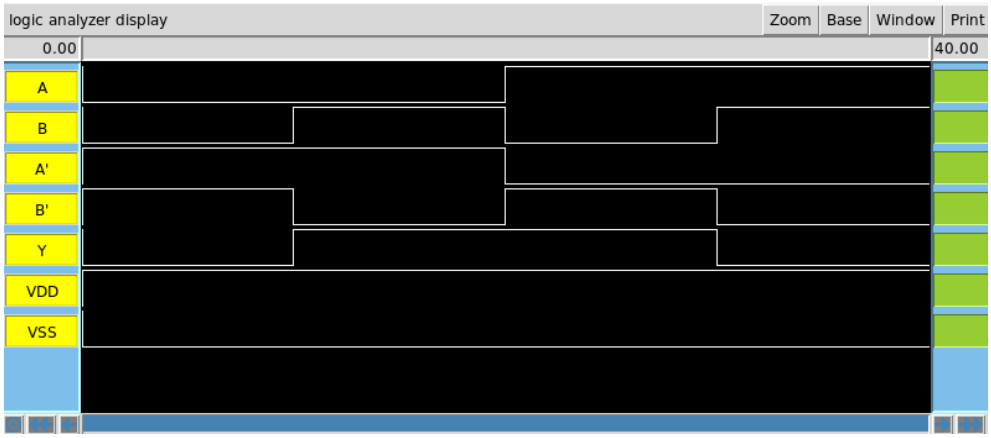
17. **8bitAND2:**



8AND2 with outputs that run through BUFFER8 with enable.

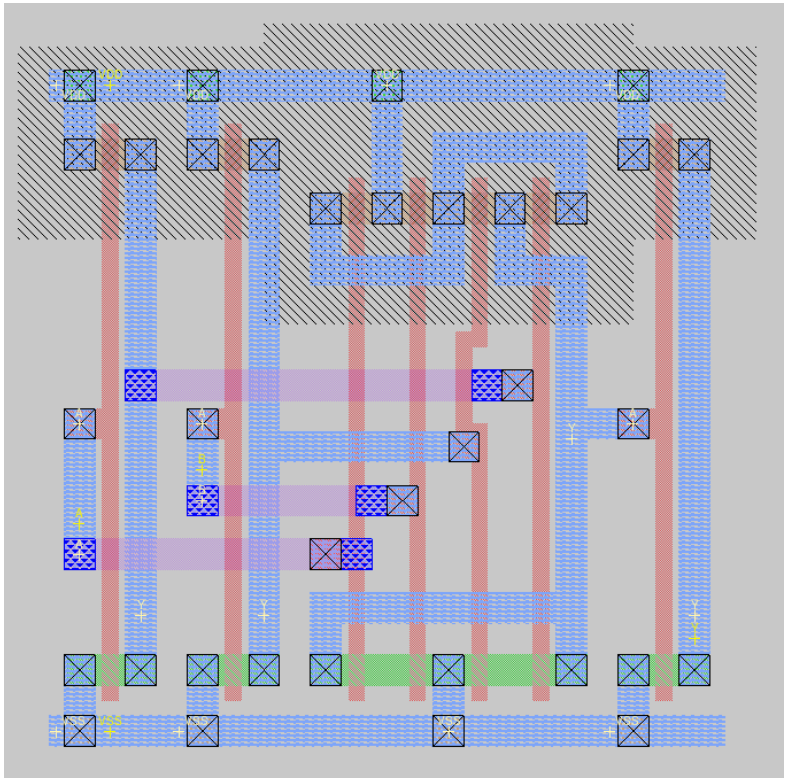
18. XOR2:





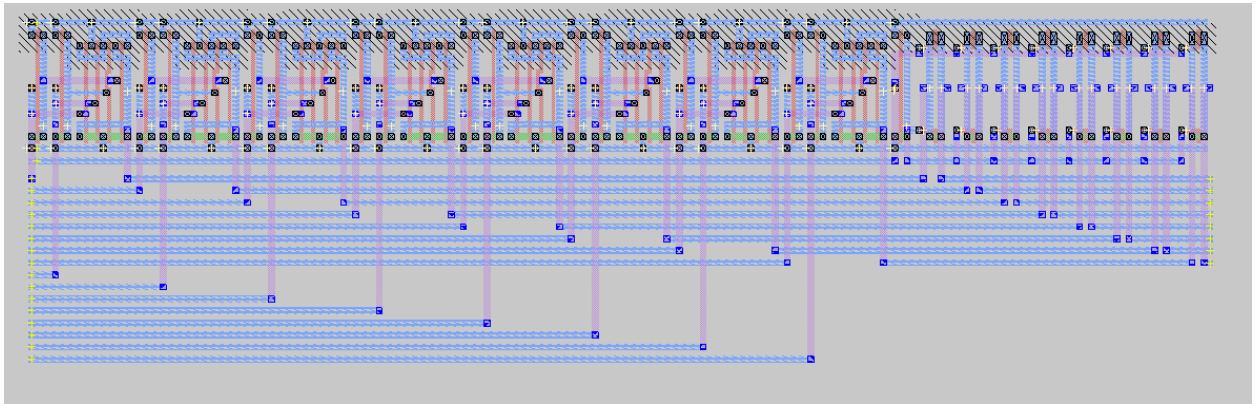
2 input XOR gate using 2 inverters to get A' and B'. Outputs 1 when A is 0 and B is 1 or when B is 0 and A is 1. Otherwise, outputs 0.

19. XNOR2:



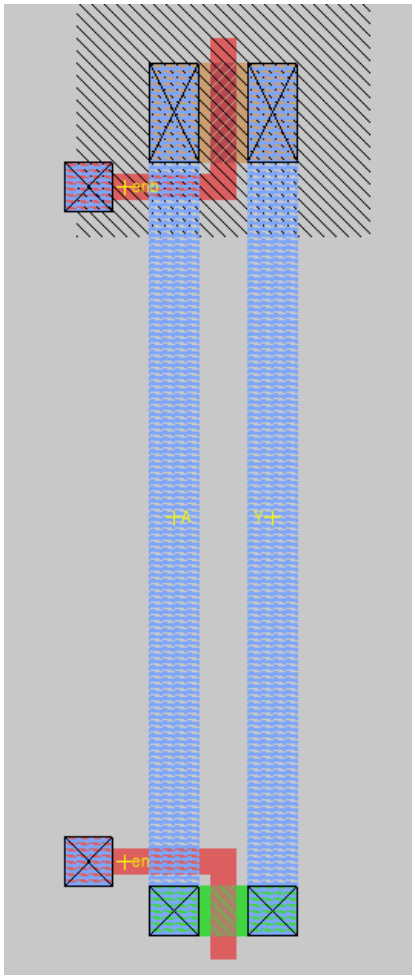
2 input XNOR gate made using an XOR2 gate and an inverter

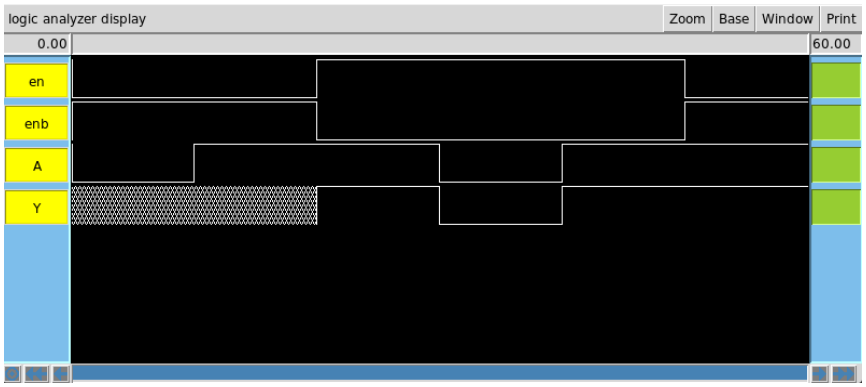
20. 8XOR2:



8 bit XOR2 using an array of 8 XOR2 Gates

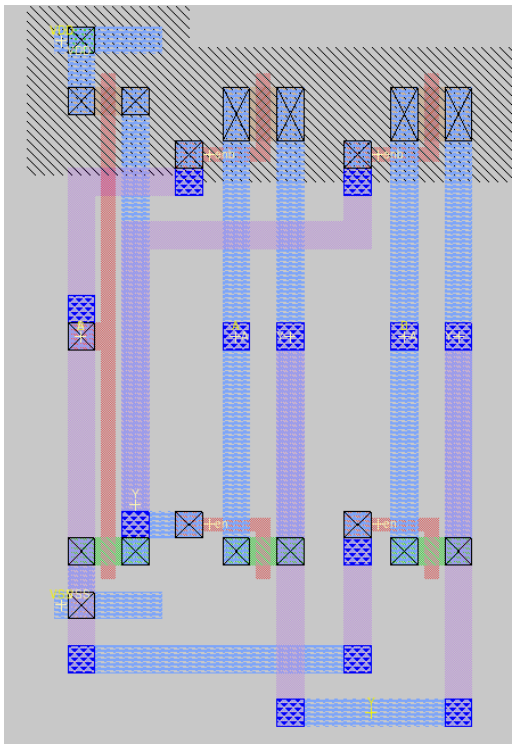
21. TRANSMISSION:

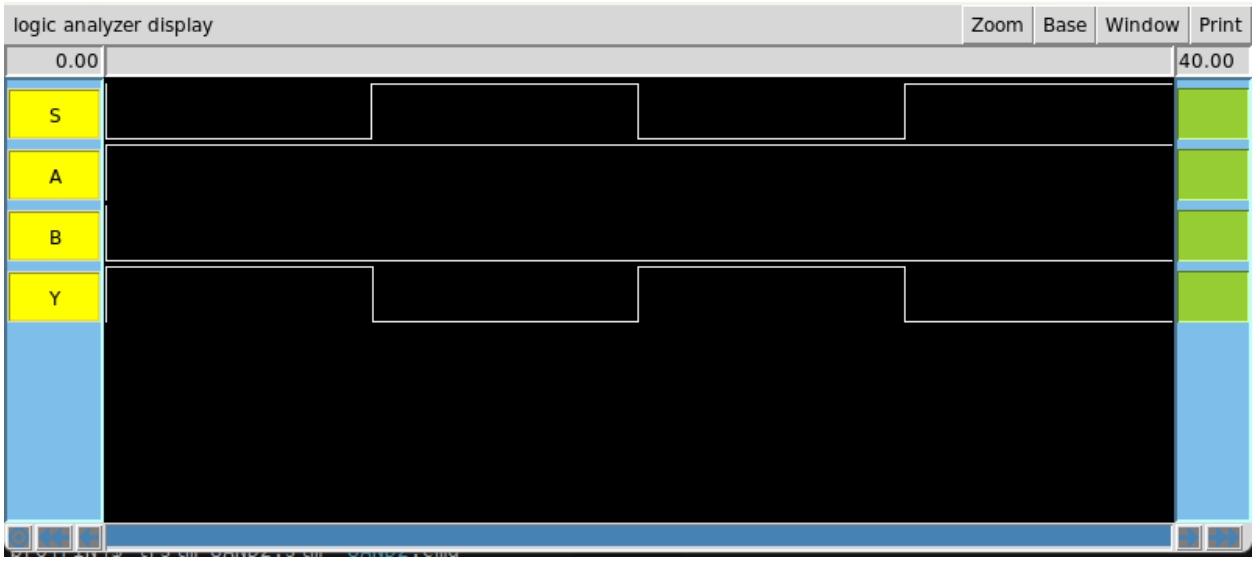




Tristate Buffer with enable made using transmission gates. Will let the value of input A through to the output when enable is off.

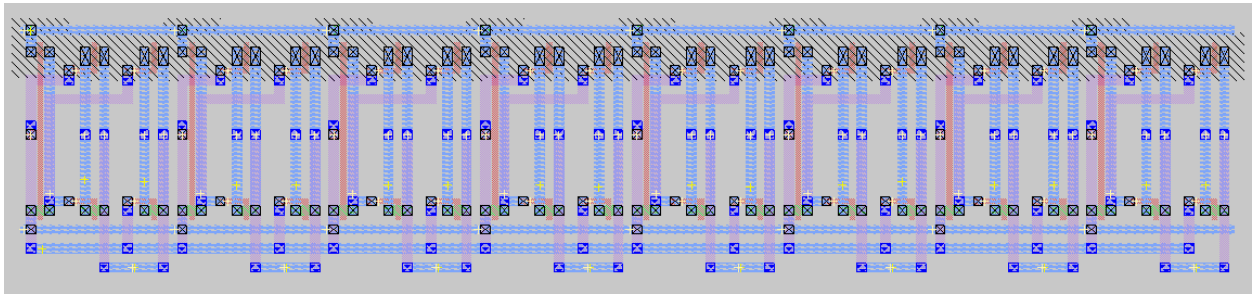
22. **MUX2to1:**

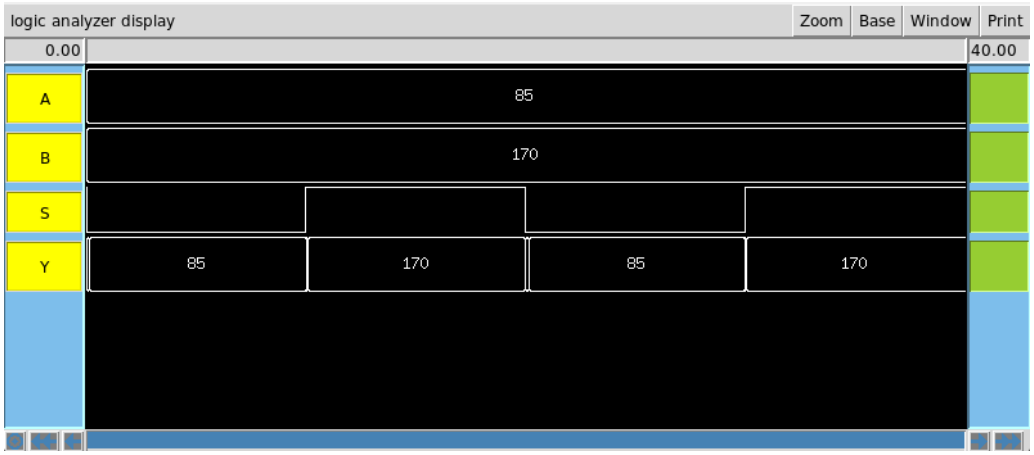




2 to 1 Multiplexer made using 2 transmission gates and an inverter. Will select one of A or B to output to Y based on whether the selector/enable bit (S) is on/off.

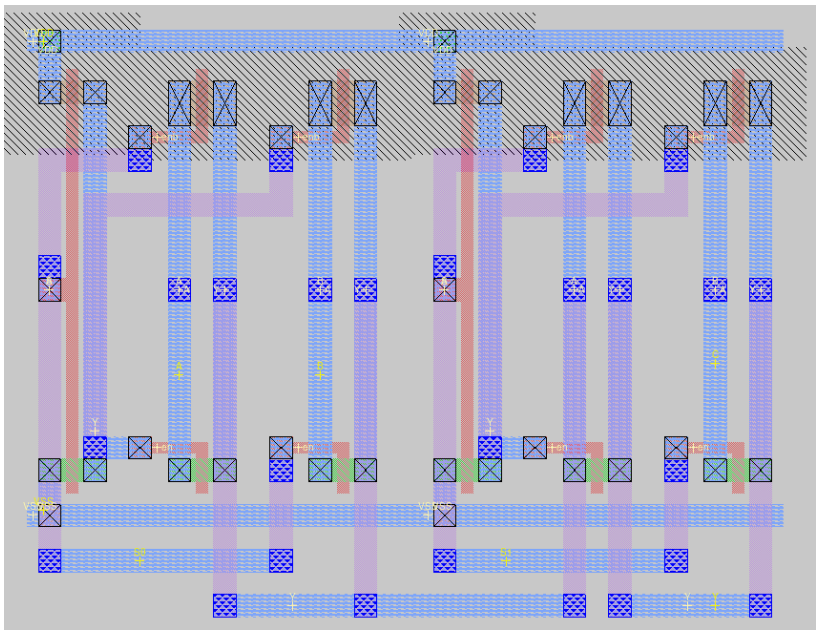
23. **8bitMux2to1:**

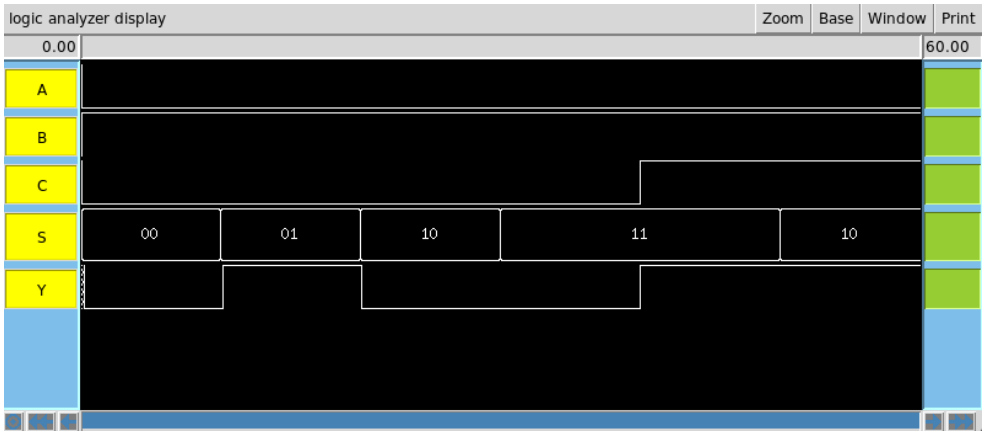




Array of 8 MUX2to1s with 16 (8x2) inputs, a selector input, and 8 outputs.

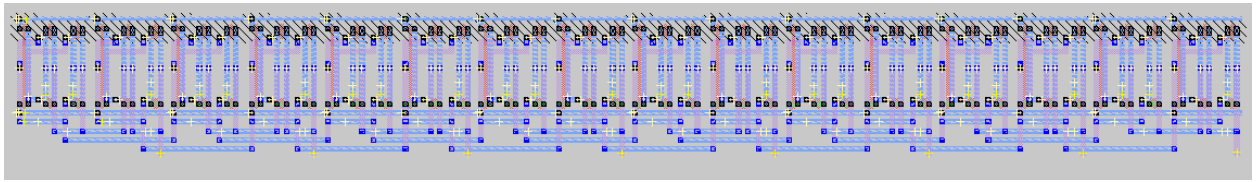
24. MUX3to1:

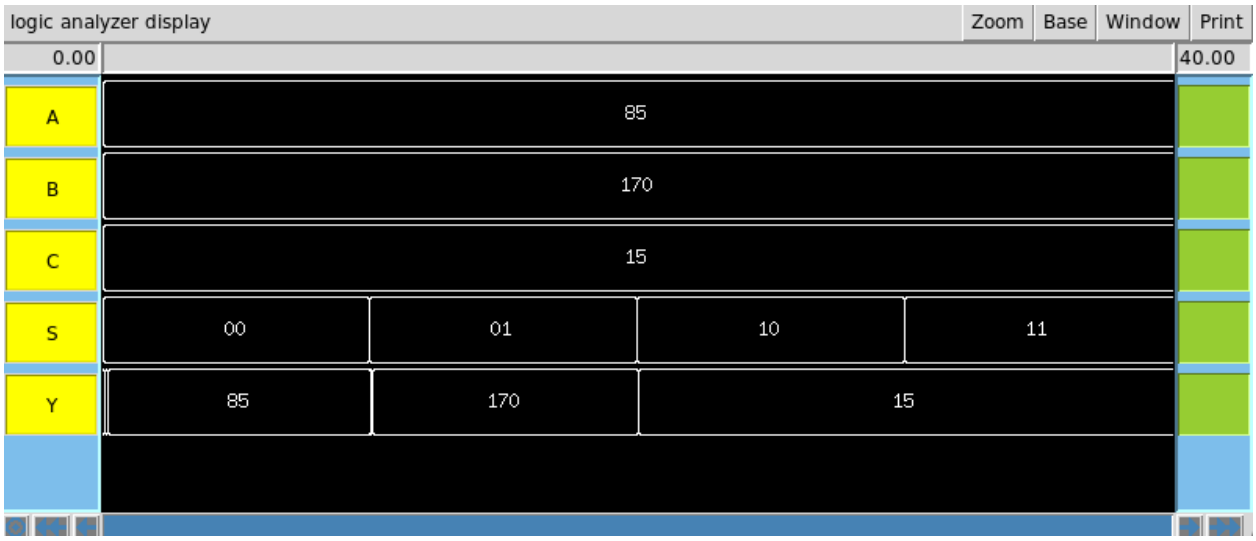




3 input multiplexer made using 2 MUX2to1s

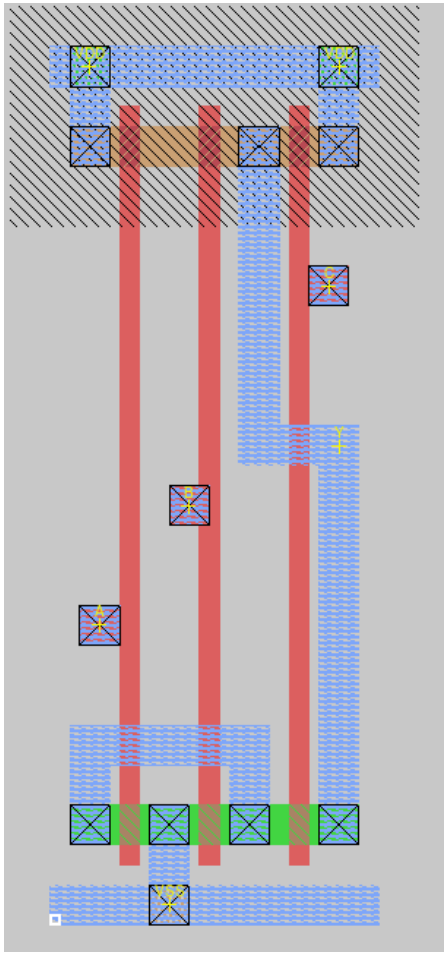
25. **8bitMUX3to1:**

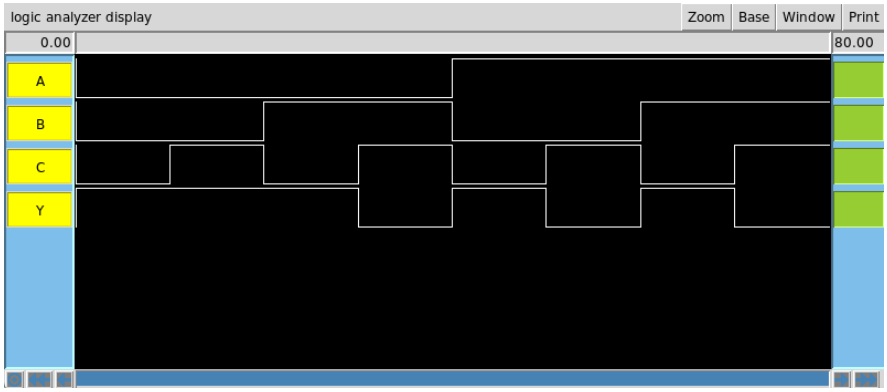




Array of 8 MUX3to1s

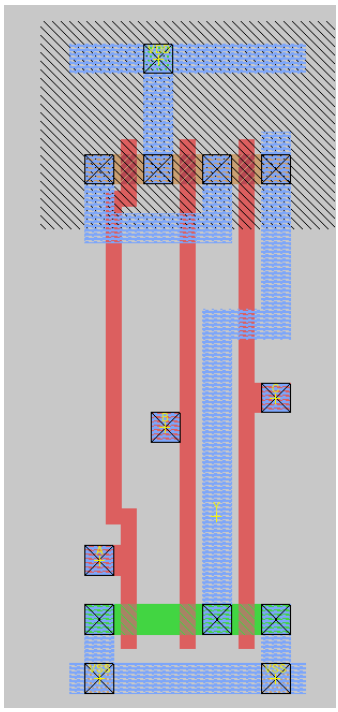
26. <u>OAI21:</u>

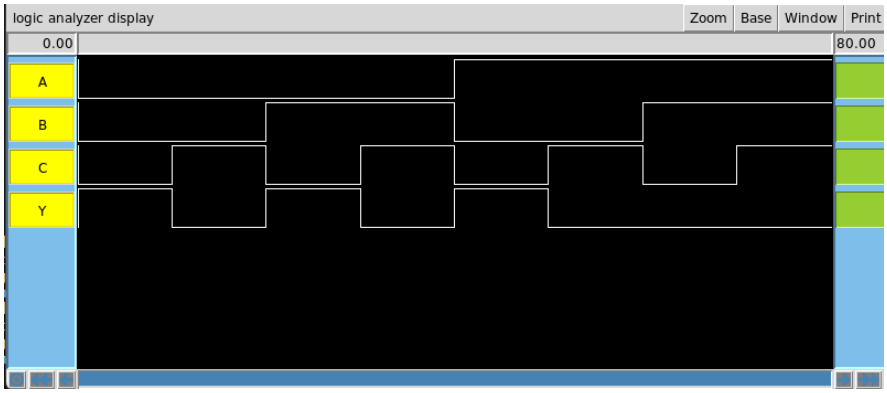




OR AND Inverter. Takes 3 inputs(A,B,C). ORs A and B together, ANDs C with the result, and outputs the inversion of that result.

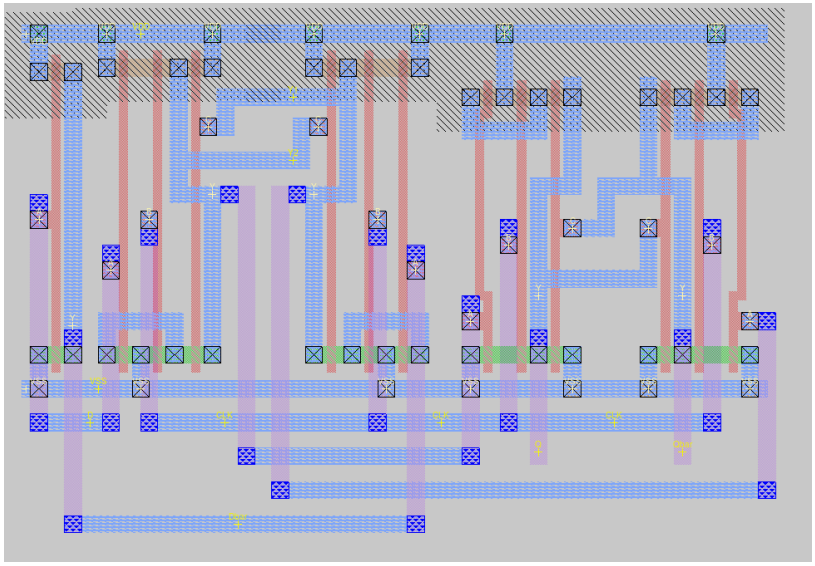
27. AOI21:

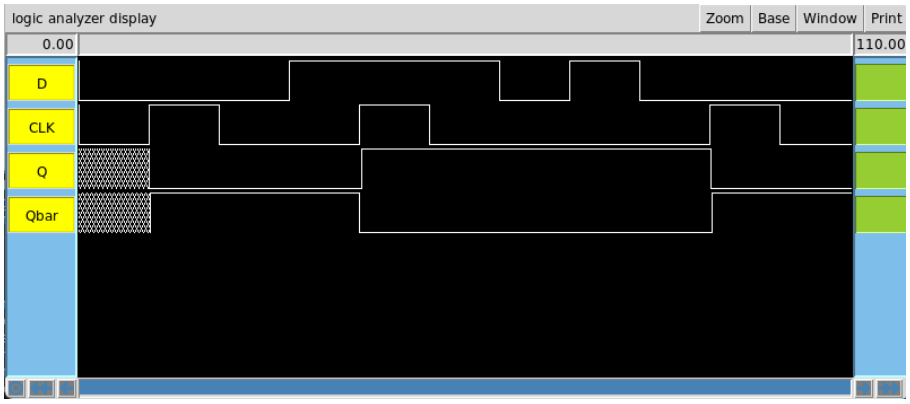




AND OR Inverter. Takes 3 inputs(A,B,C). ANDs A and B together, ORs C with the result, and outputs the inversion of that result.

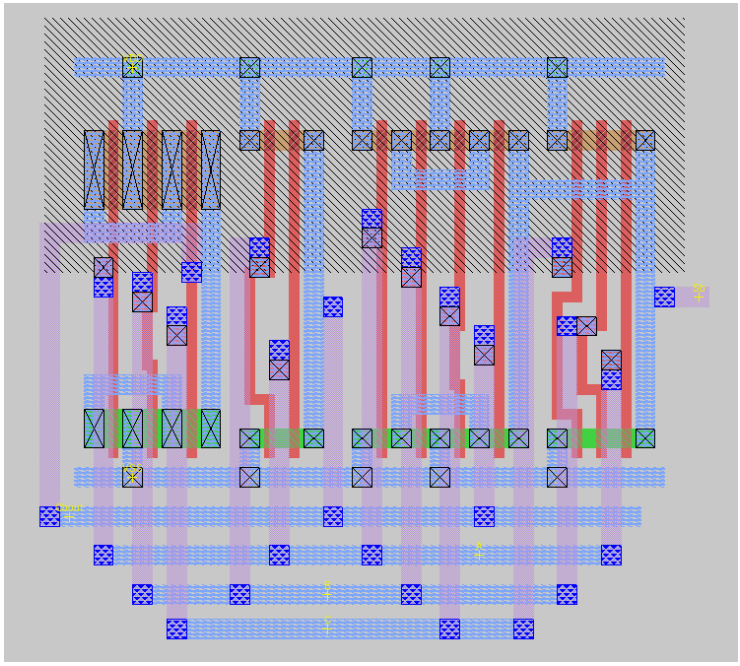
28. DFF:

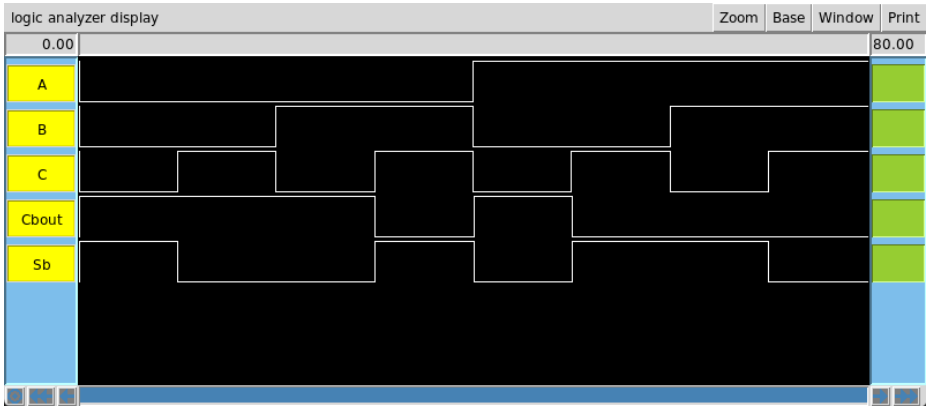




D Flip Flop. when clk rises, stores a new value from D when clk is triggered. Otherwise, hold the value and don't take a new value from D.

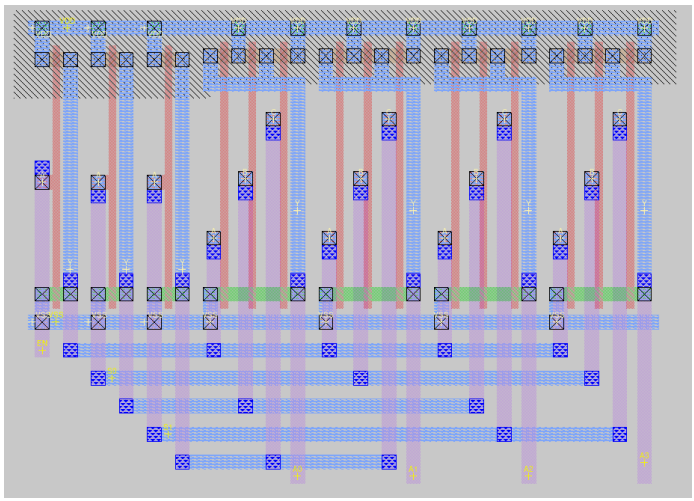
29. **FullAddr:**

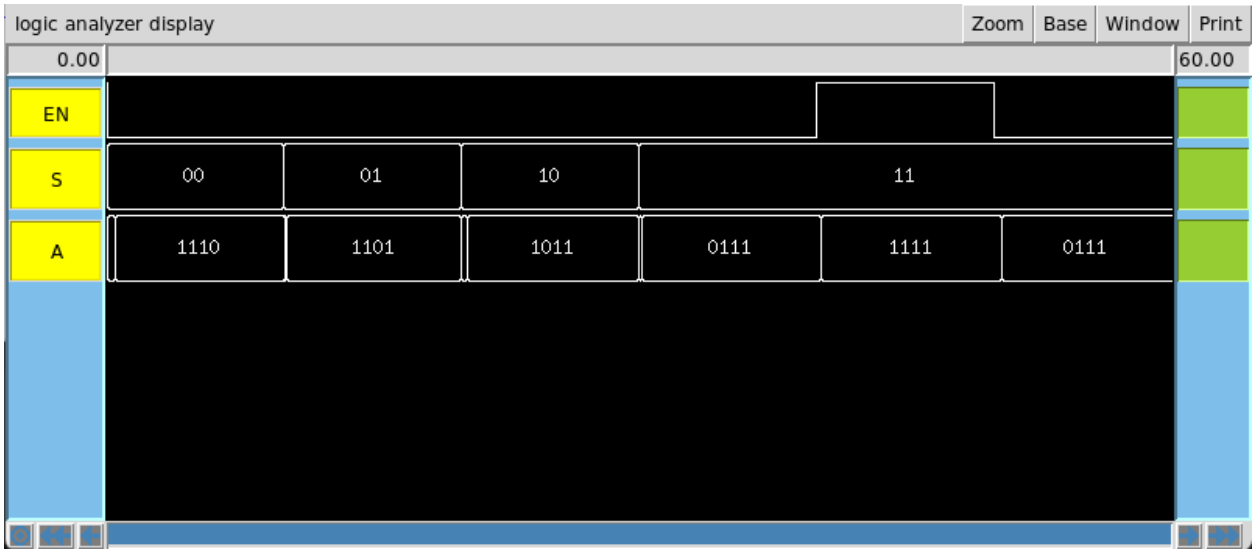




Full Adder. A and B are the main inputs, with C being a Carry In bit. Adds together these values to output the result of the addition in Sb and Cbout.

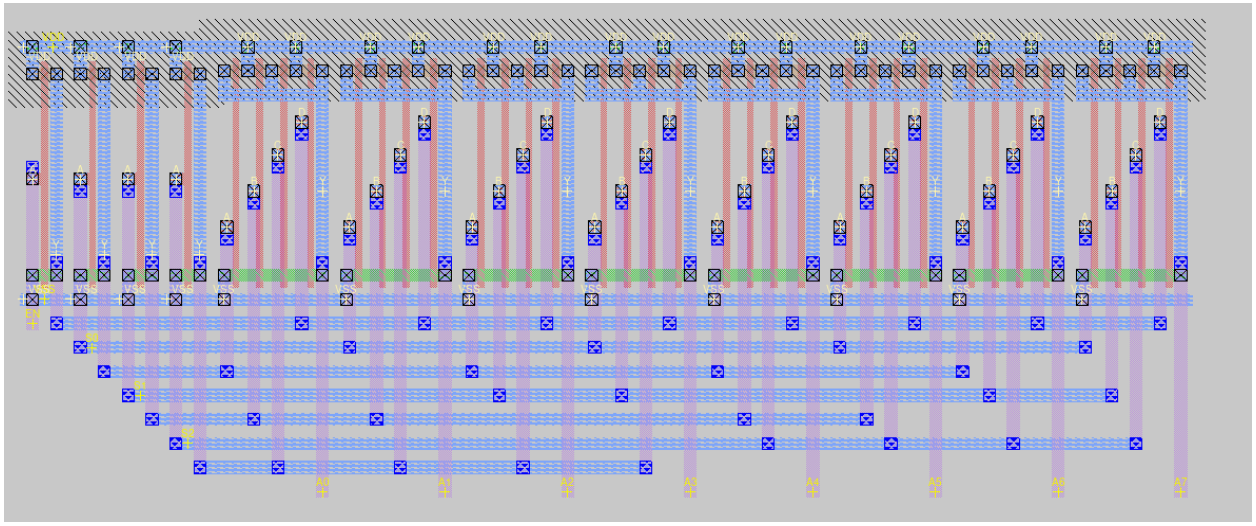
30. **Decoder_2x4**:





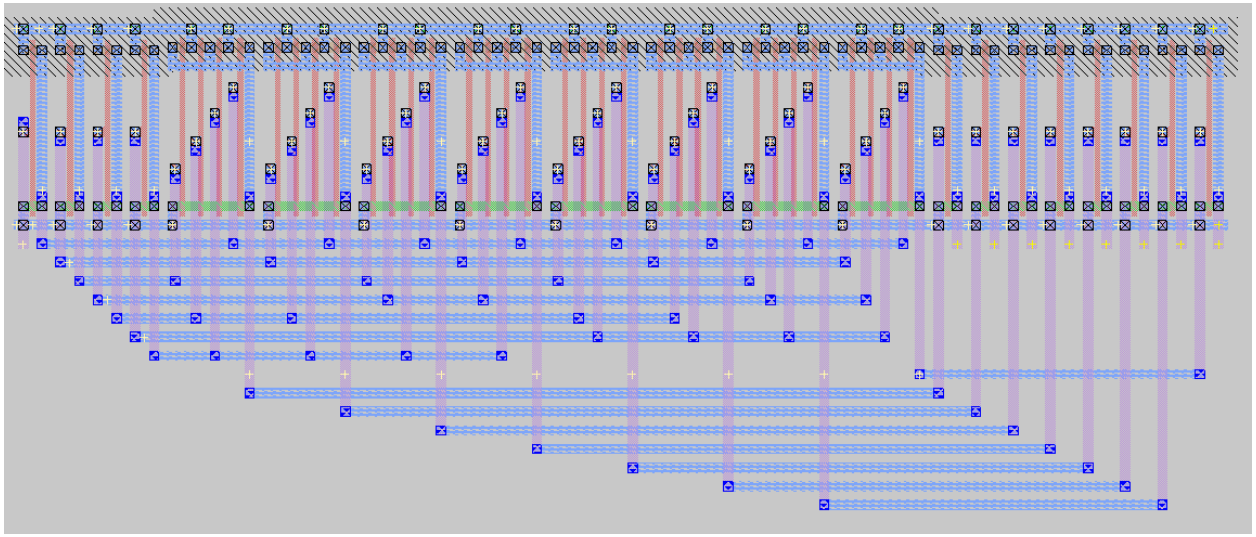
Activates/Deactivates 4 enable signals (A) based on the selection bits (S). EN will let a signal change when it is off, and will not let the signal change when it is on.

31. Decoder_4x8:



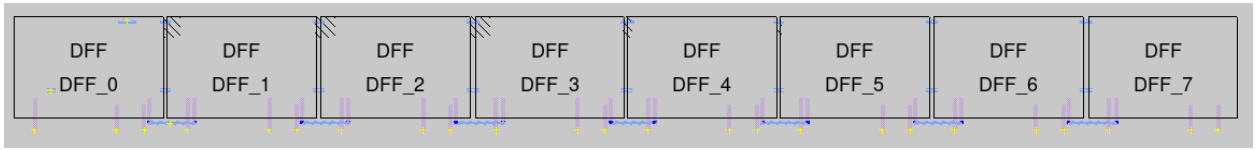
Activates/Deactivates 8 enable signals (A) based on the selection bits (S). EN will let a signal change when it is off, and will not let the signal change when it is on.

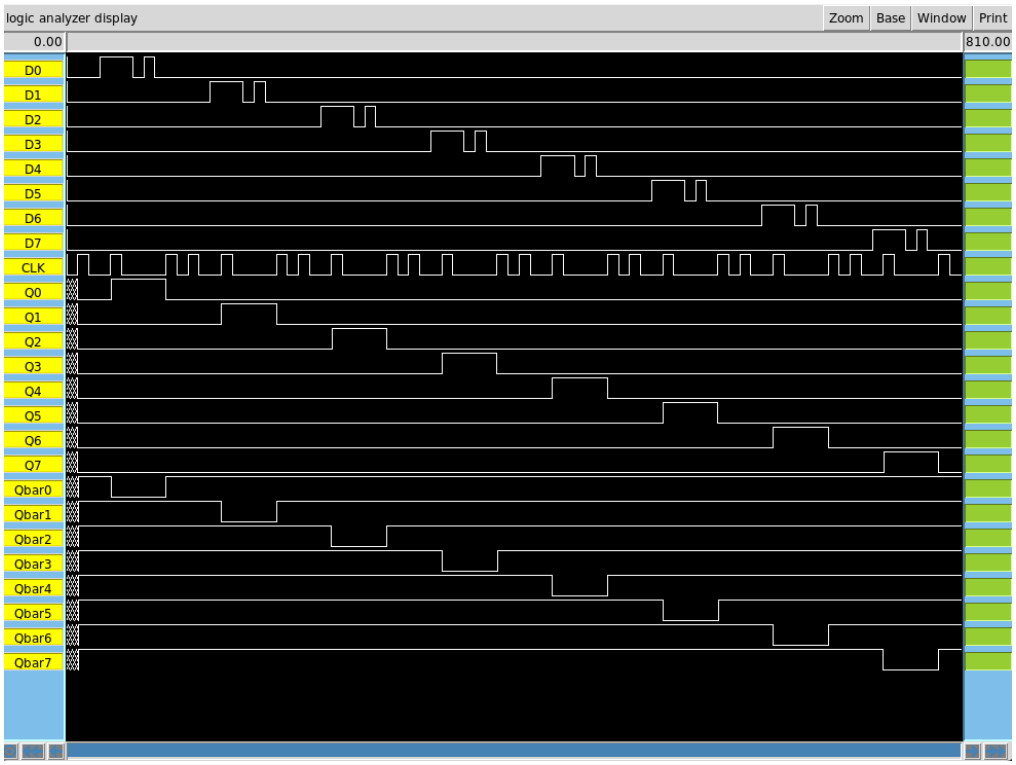
32. Decoder_Inv_4x8:



Decoder_4x8 except it inverts the enable outputs.

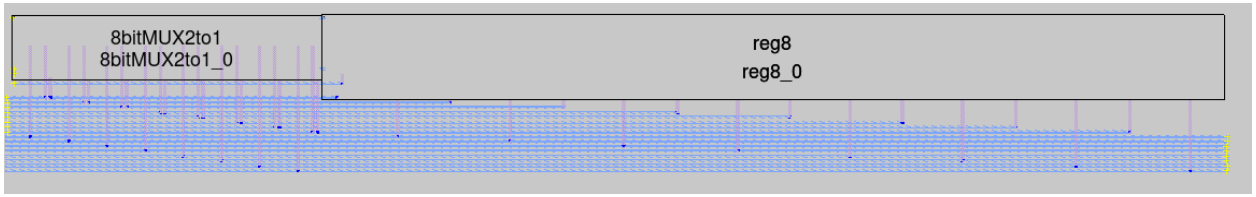
33. **reg8:**

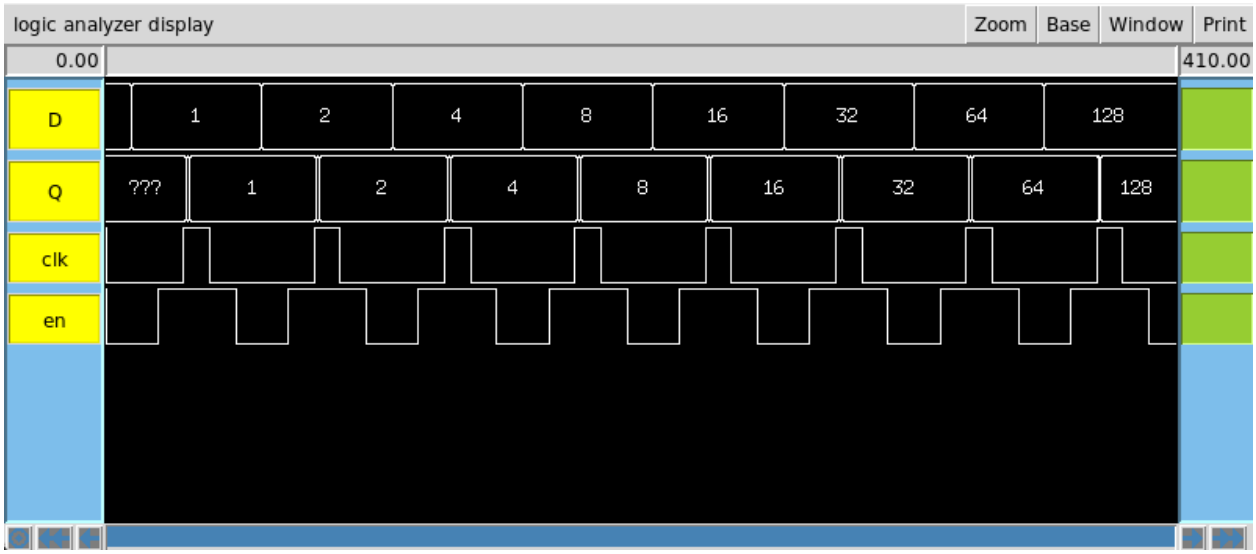




Array of 8 D Flip Flops using the same clk signals.

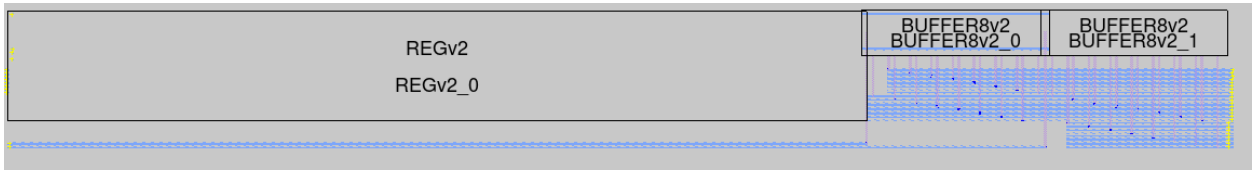
34. REGv2

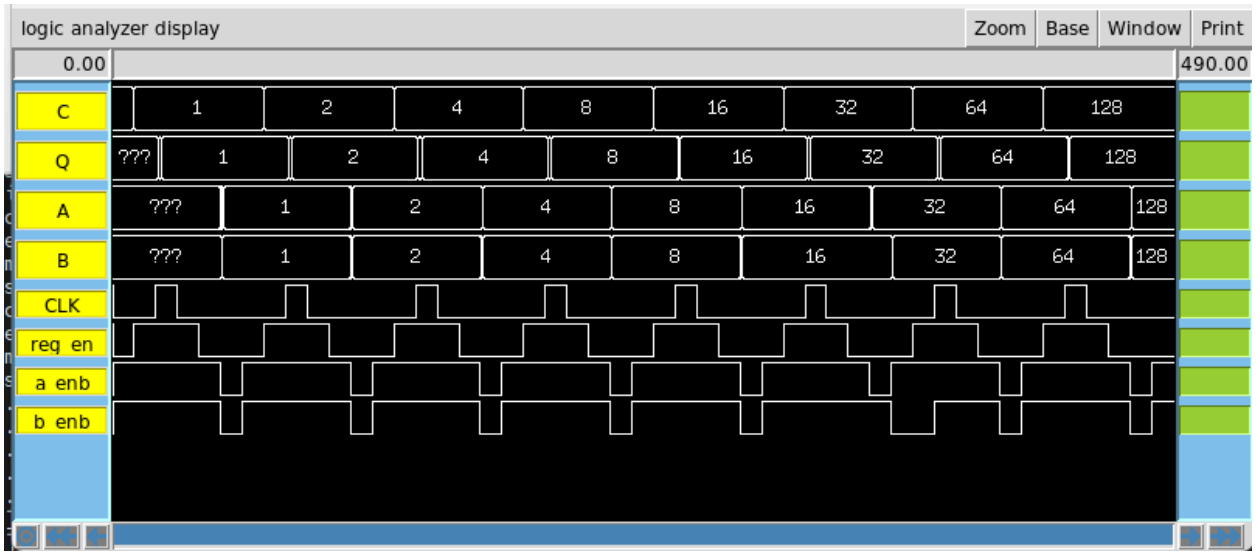




reg8 with multiplexers so the enable signal can control whether or not the DFFs will hold values or take in new values.

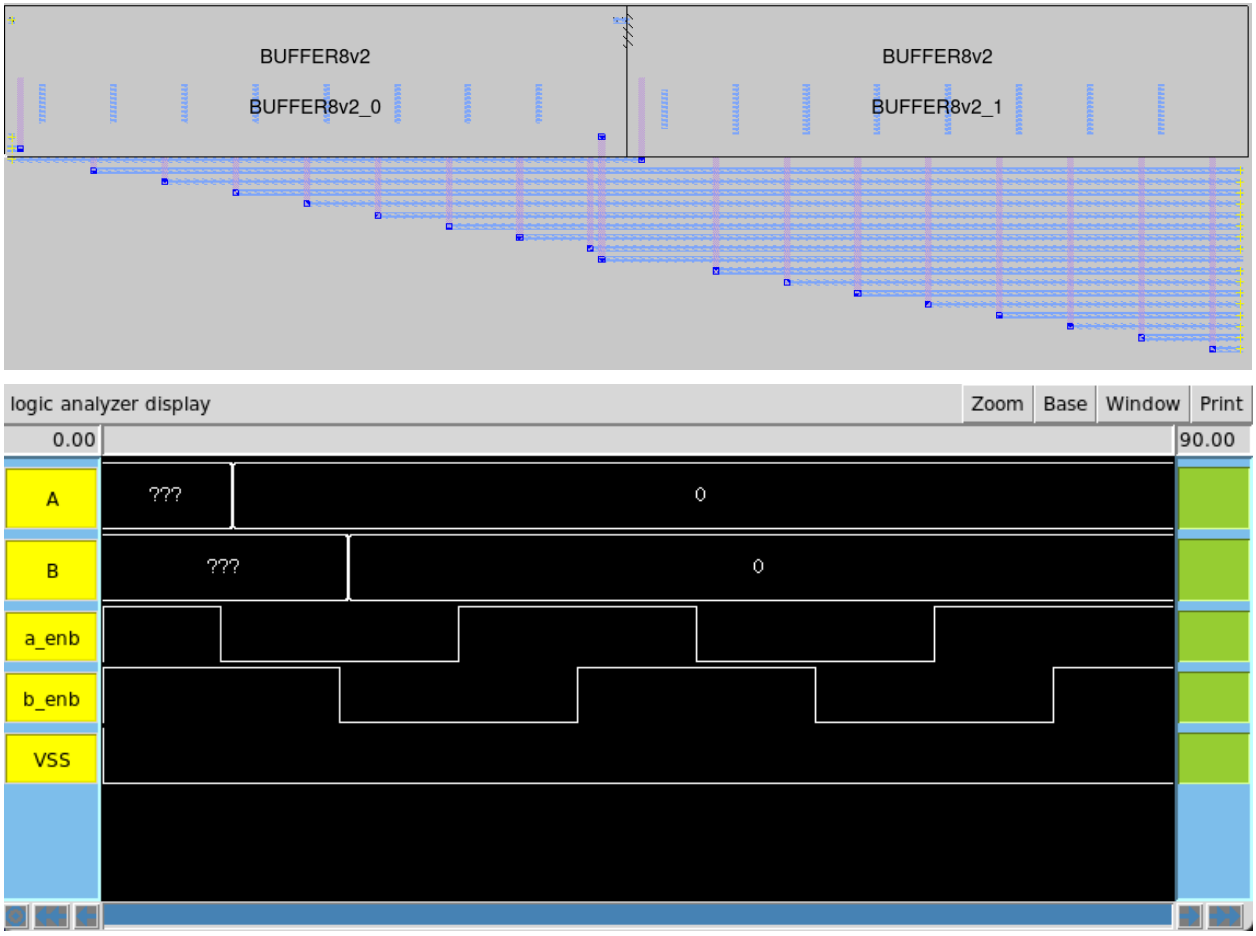
35. **REG8v2:**





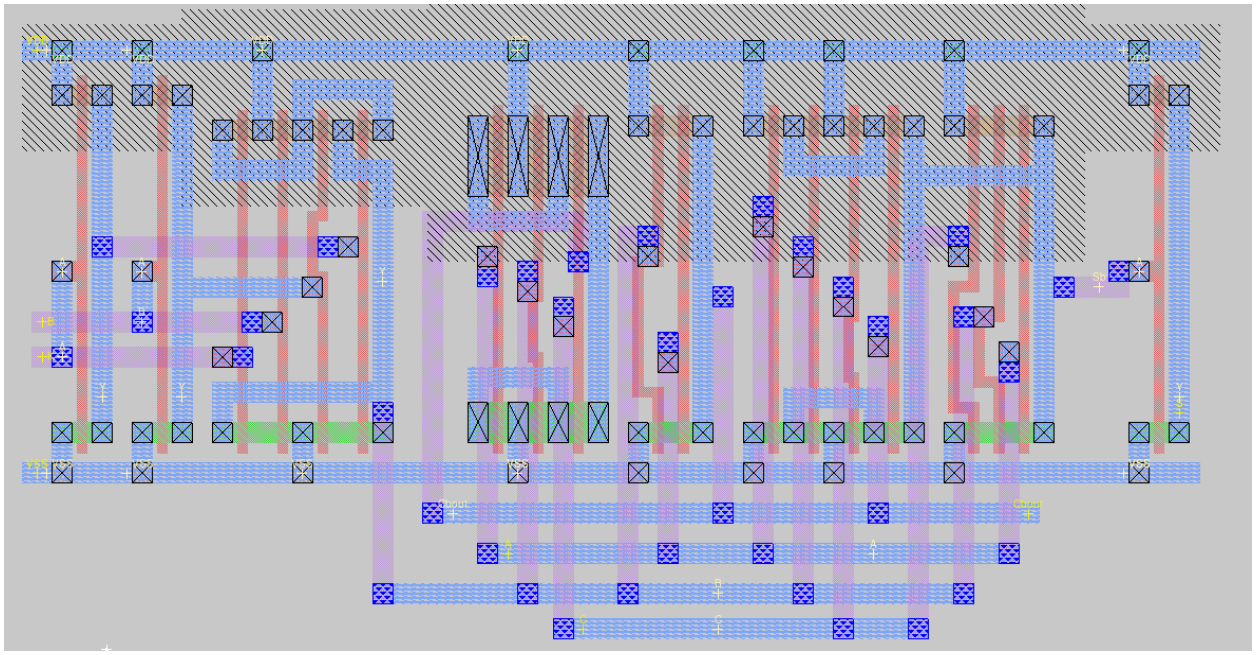
REGv2 with 2 8 bit buffers so we can control if the signal is read from the register to output A or B using a_enb and b_enb.

36. **REG0v2:**



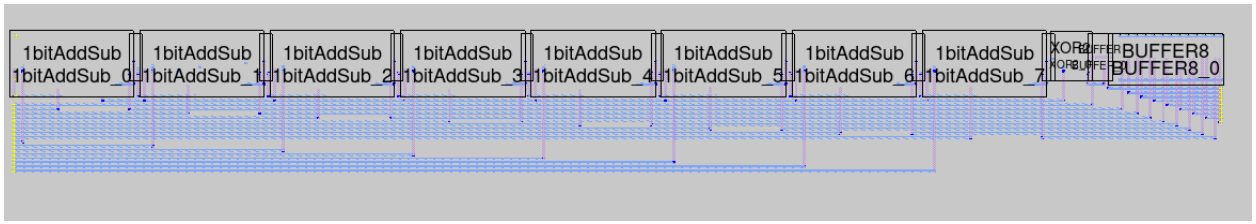
Since register 0 just needs to output the value 0 and takes no inputs, we can use 2 8 bit buffers with inputs shorted to VSS to decide whether to read the value 0 into all 8 bits of A or B using a_enb or b_enb.

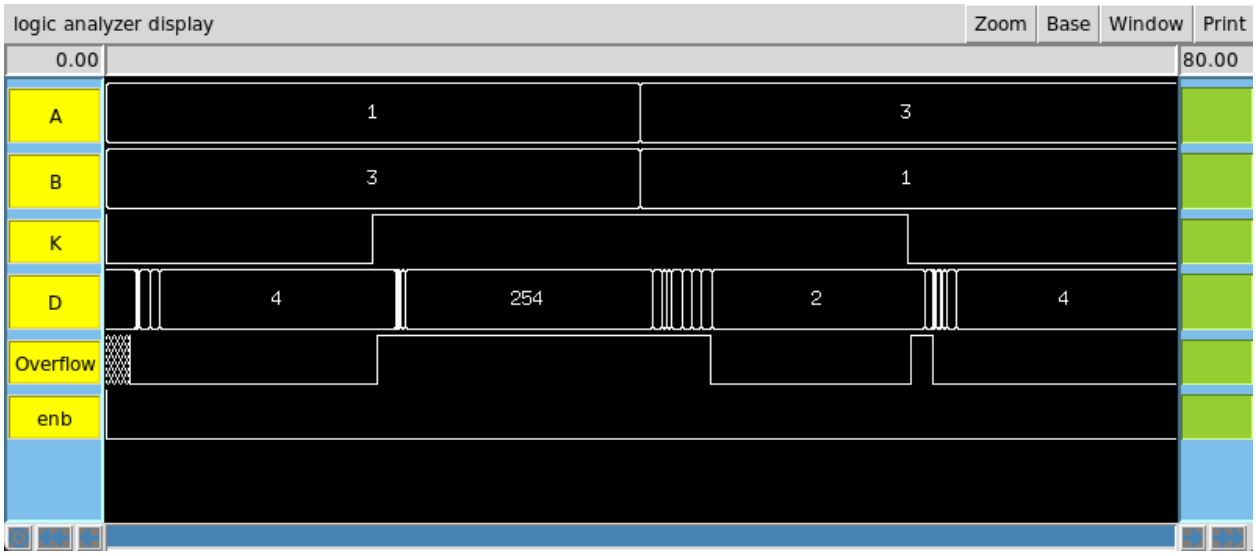
37. **1bitAddSub:**



Full Adder with a XOR2 gate to perform subtraction.

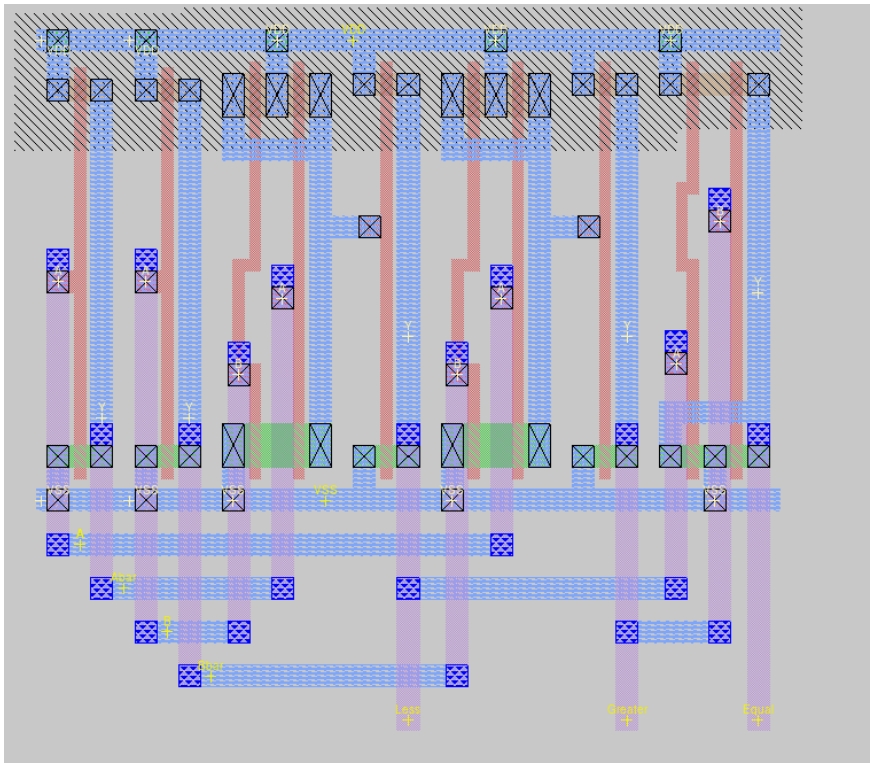
38. <u>8bitADDSUB:</u>

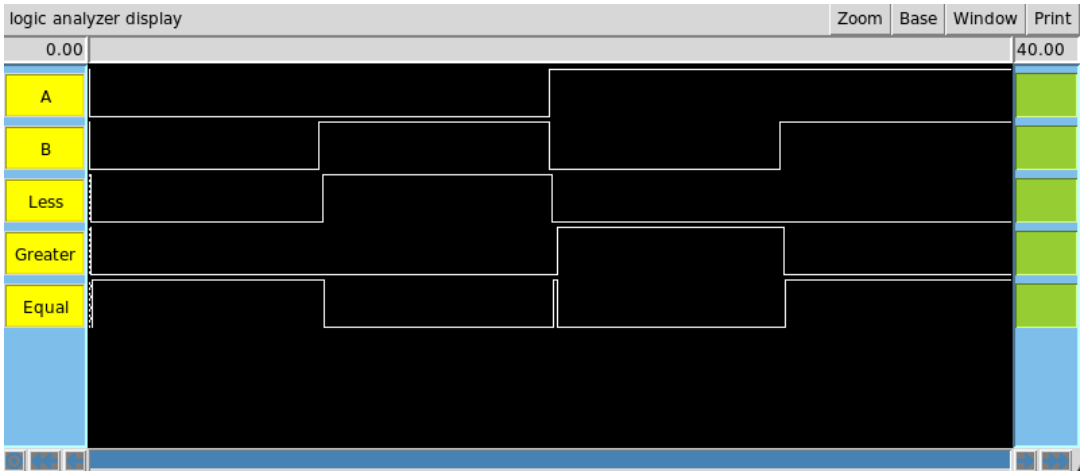




Array of 8 1 bit Adders with an 8 bit buffer that enables output when enb is 0. Overflow is detected using the output of the XOR2 gate.

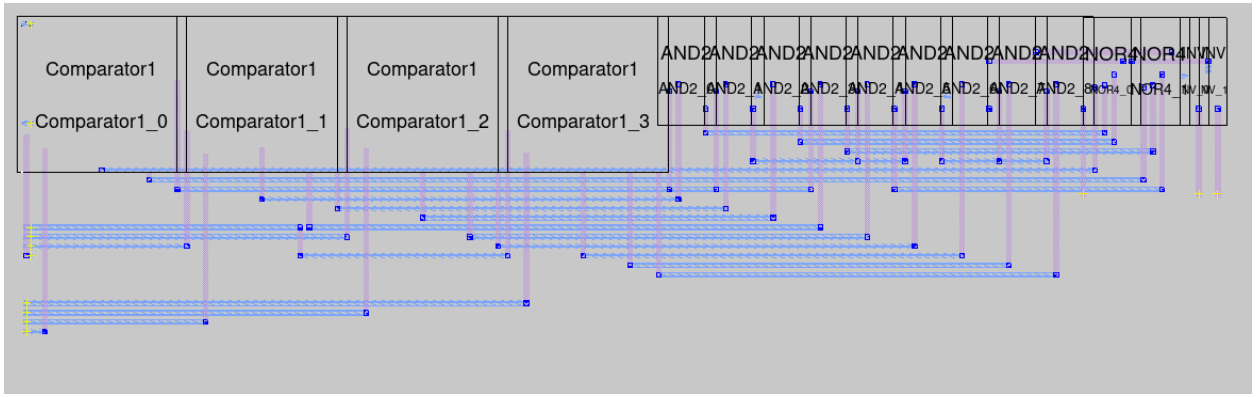
39. Comparator1:

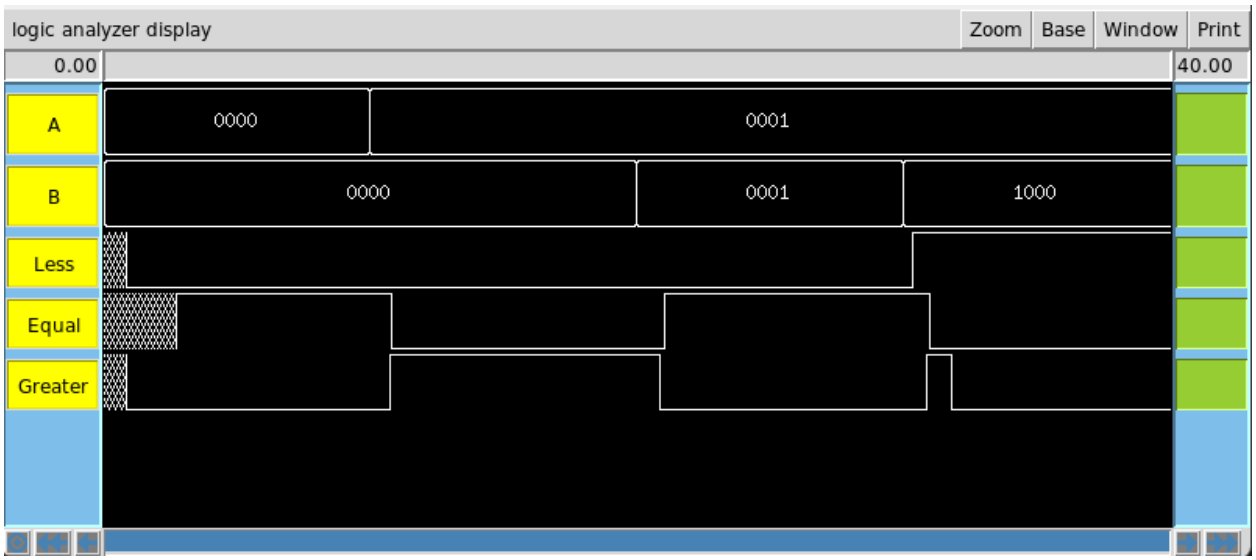




Compares the bits from inputs A and B, and outputs if the A is less than, greater than, or equal to B.

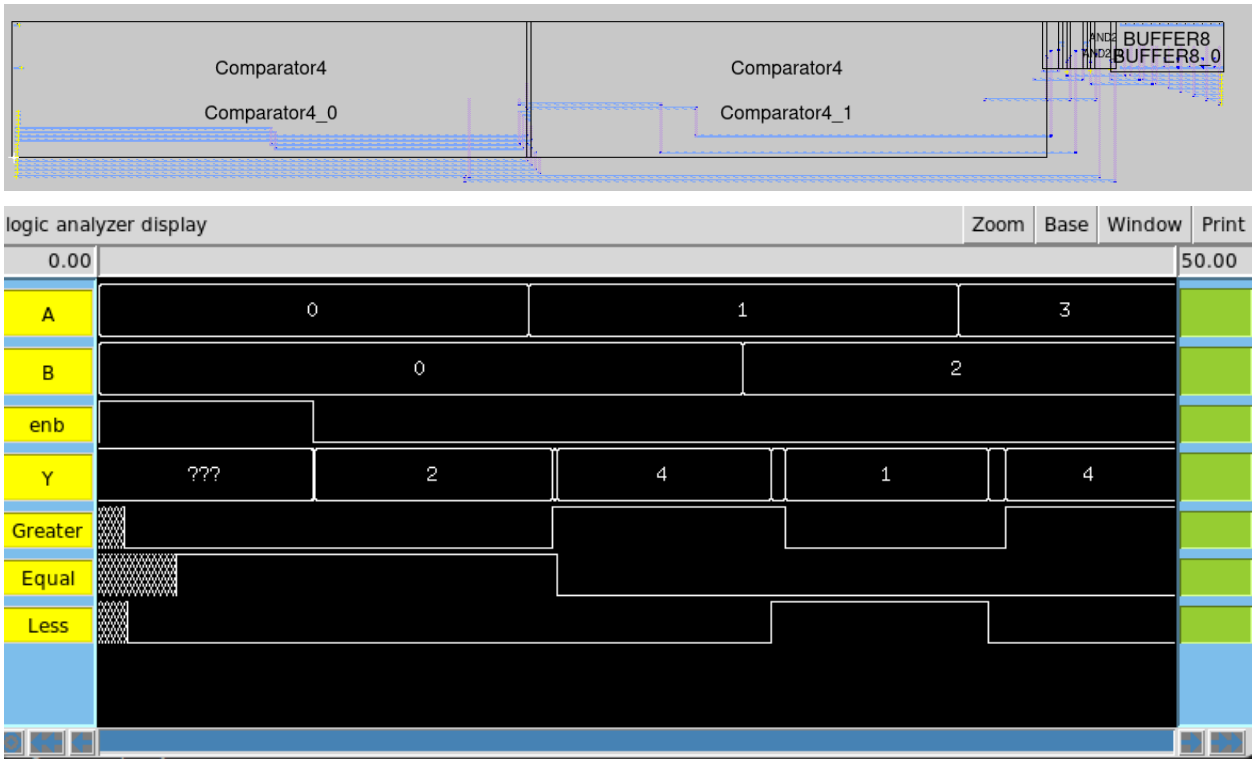
40. Comparator4:





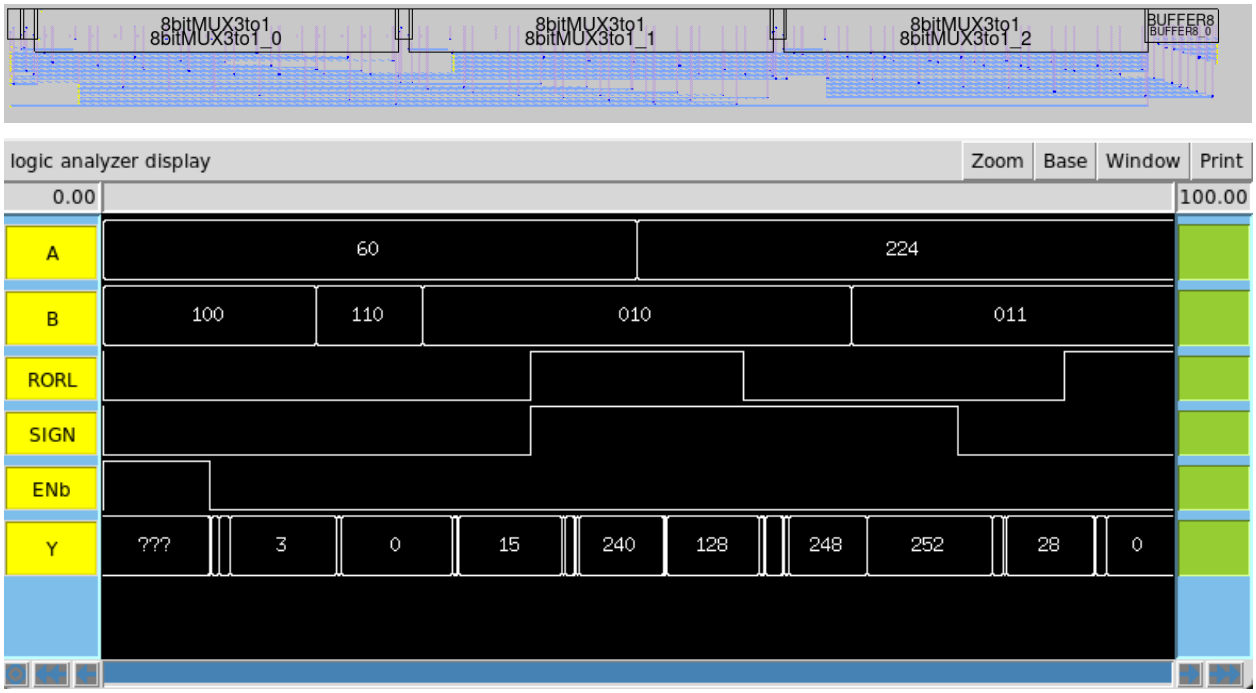
Uses an Array of Comparator1s, AND2 gates, and two NOR4 gates with inverters to compare the 4 bits of A to the 4 bits of B and check if A is less than, greater than, or equal to B.

41. Comparator8:



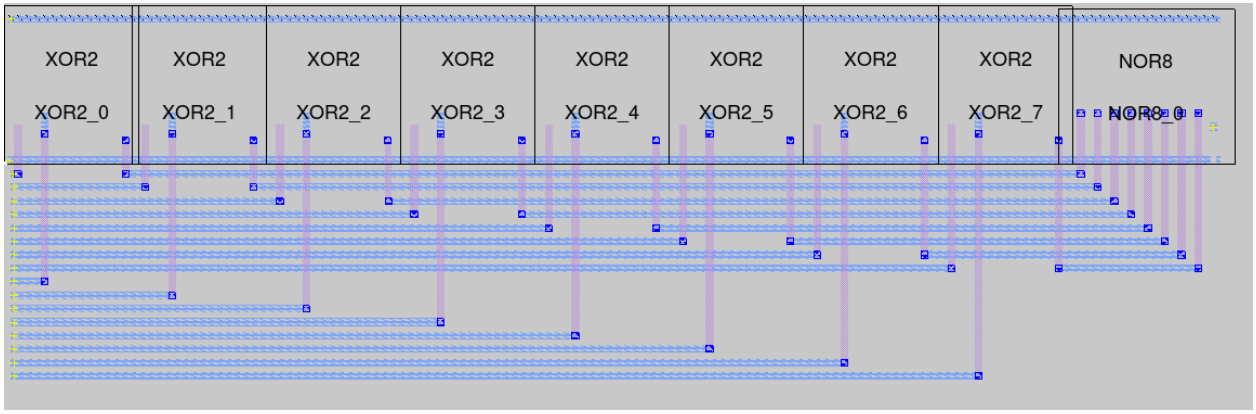
Compares the 8 bits of 8 to the 8 bits of B using two Comparitor4s and some combinational logic. Enabled when enb is 0.

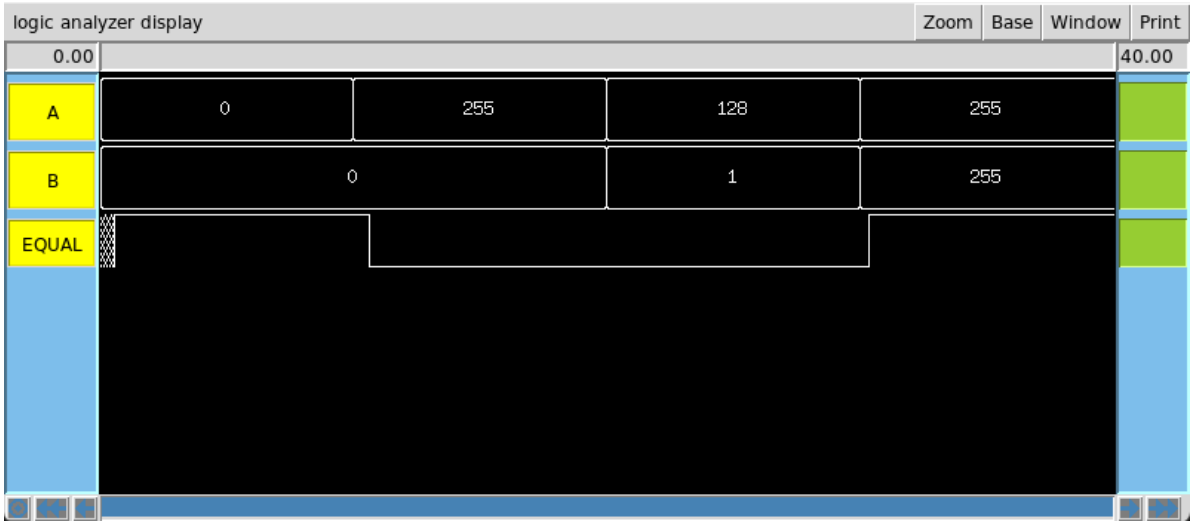
42. BARREL_SHIFT:



Bit Shifter made using Multiplexers. BUFFER8 used to add an enable signal.
Shifts the value of A a number of times equal to the value of B. RORL chooses whether to shift right or left. SIGN lets us know whether to sign extend or not. ENb will enable the barrel shifter when it is off.

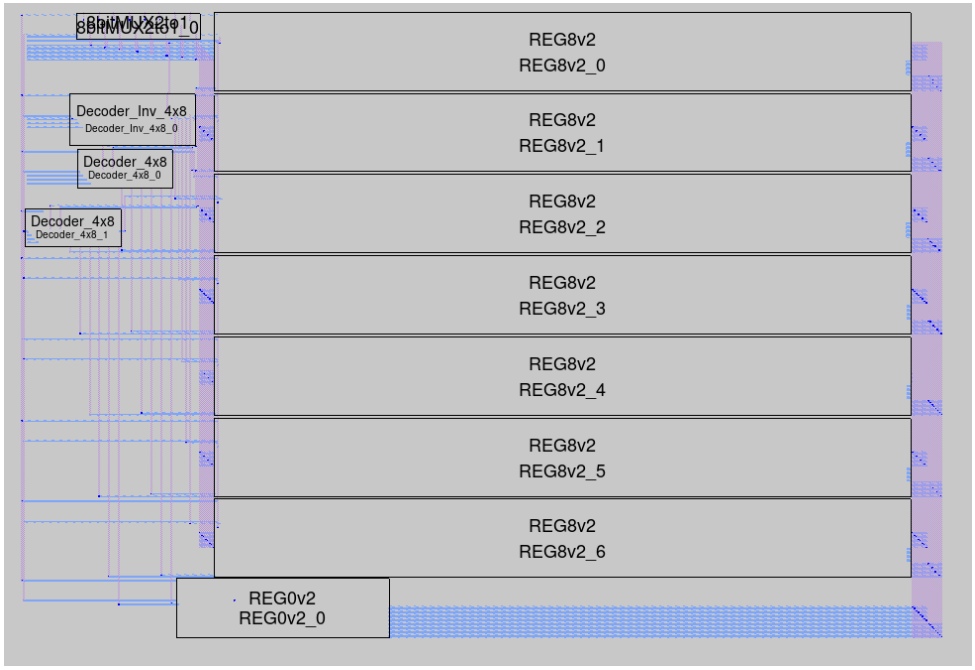
43. Equalv2:

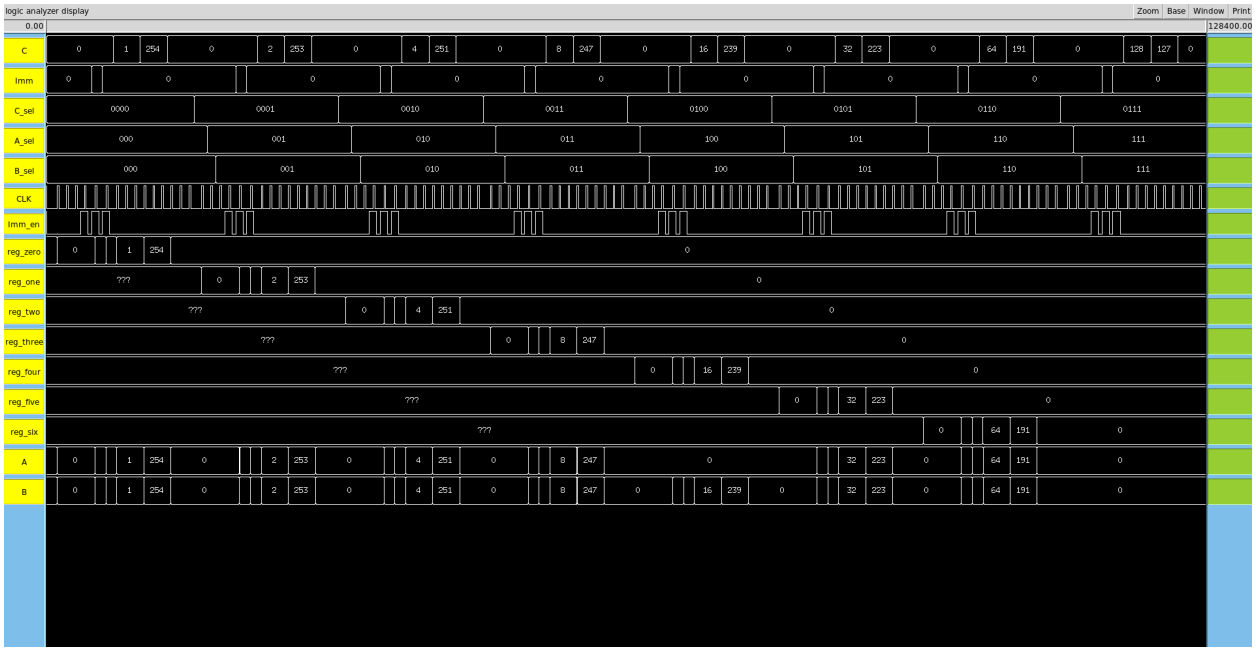




Checks if A and B are equal to each other using XOR gates.

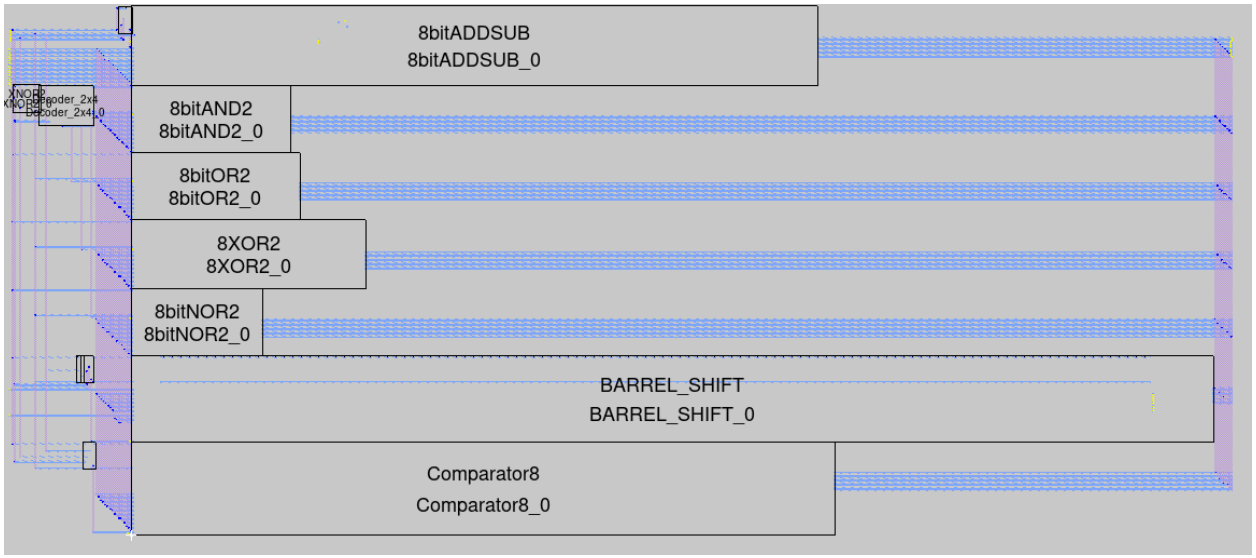
44. REGISTER_FILEv2:





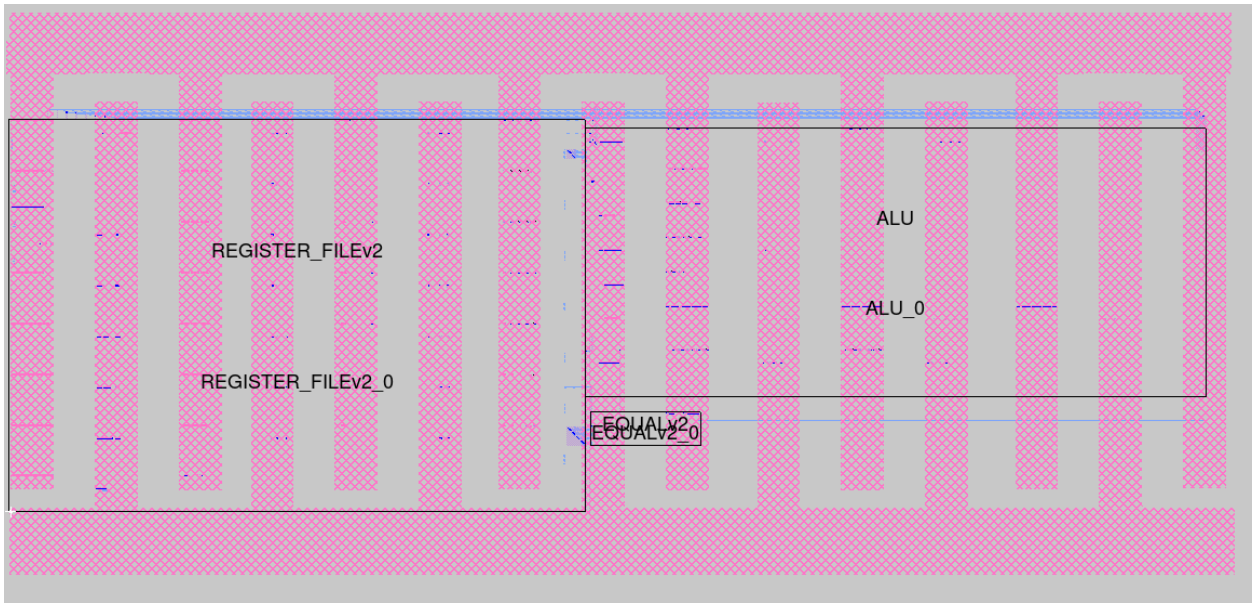
Stores the values of either C or Imm based on Imm_en in one of the registers based on Csel. Reads values from register into outputs A and B based on A_sel and B_sel. Writing happens at every rising edge of clk.

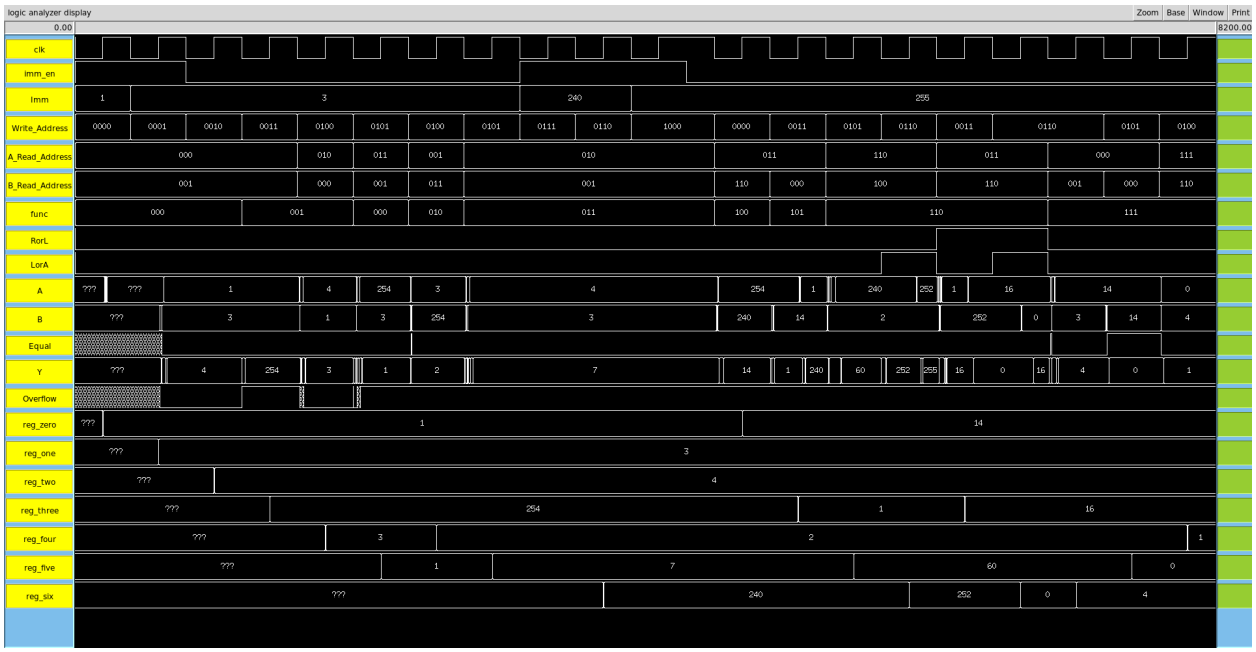
45. ALU



Outputs

46. REGandALUv3:





Full Description in Top Level Description
Test Cases for Simulation in Test cases excel sheet