DIRECT DIGITAL SYNTHESIZER

Design of ROM

Cyrus Miller

Department of Electrical and Computer Engineering

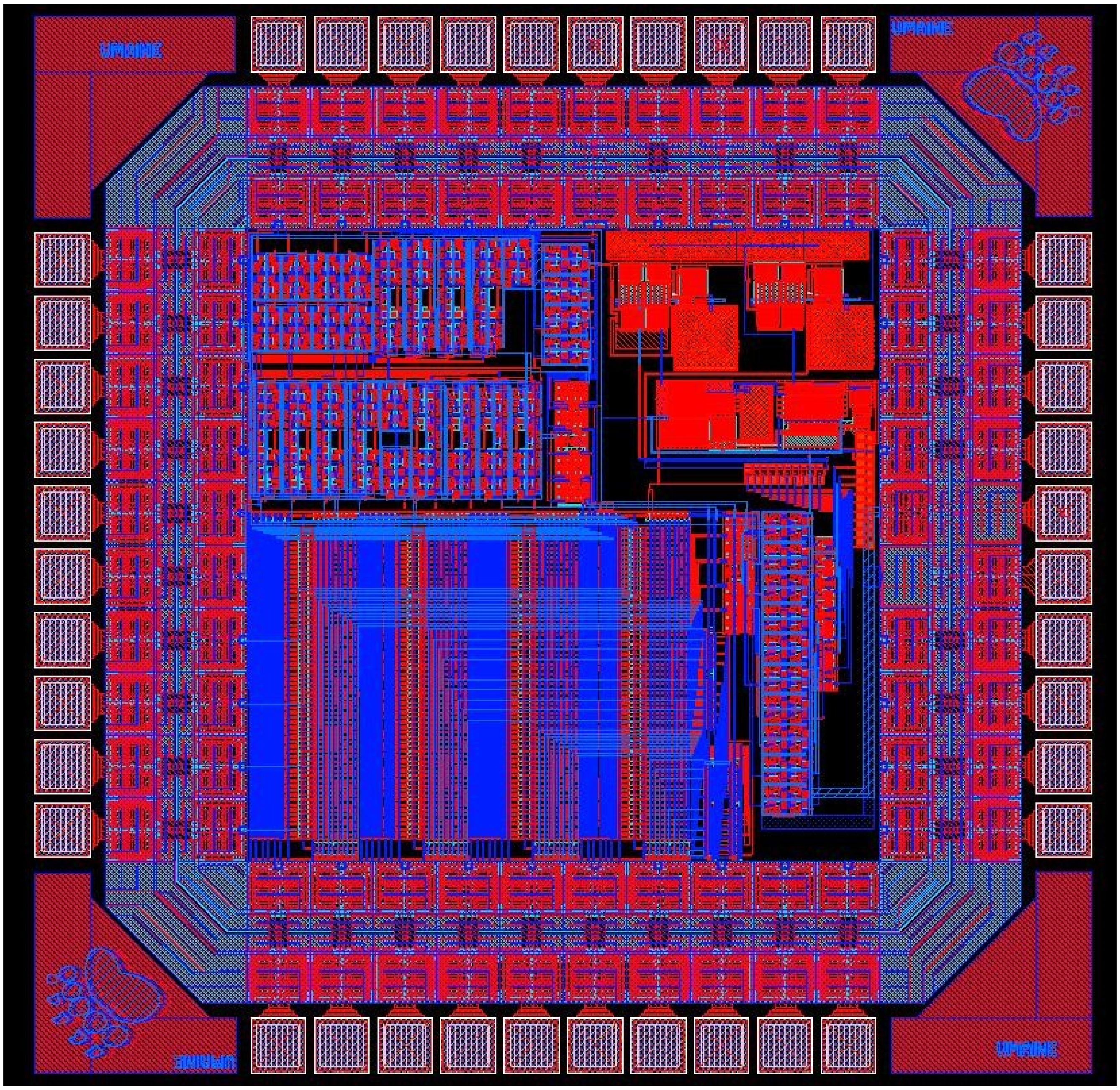
University of Maine

Orono, Maine

Table of Contents

Table of Figures

Table of Figures (continued)



http://www.eece.maine.edu/research/vlsi/2006/Miller/

1 Project Overview

We were tasked with building a Direct Digital Synthesizer (DDS), which play an important role in modern digital communications for sine wave generation. There were three different groups, each working on a separate building block in the development of this DDS. The three building blocks are the Accumulator, the ROM and the Digital to Analog converter. In this project, my group was to design a 256 bit ROM to store the sinusoidal values for DDS. We were to accept 8 inputs from the Accumulator and output 9 bits to the DAC to effectively model the output of a sine wave. The pieces involved were four 64 bit ROMs, two 2:1 multiplexers and a latch. The multiplexers are used to select the right ROM and their outputs are latched using D flip-flops. This ROM receives its 8 bit input from a 12 bit, 50 MHz accumulator designed by Aravind Reghu. The latched outputs are then fed to two 10 bit Digital to Analog converters; one designed by Steve Fortune and the other designed by Raghu Tumati.

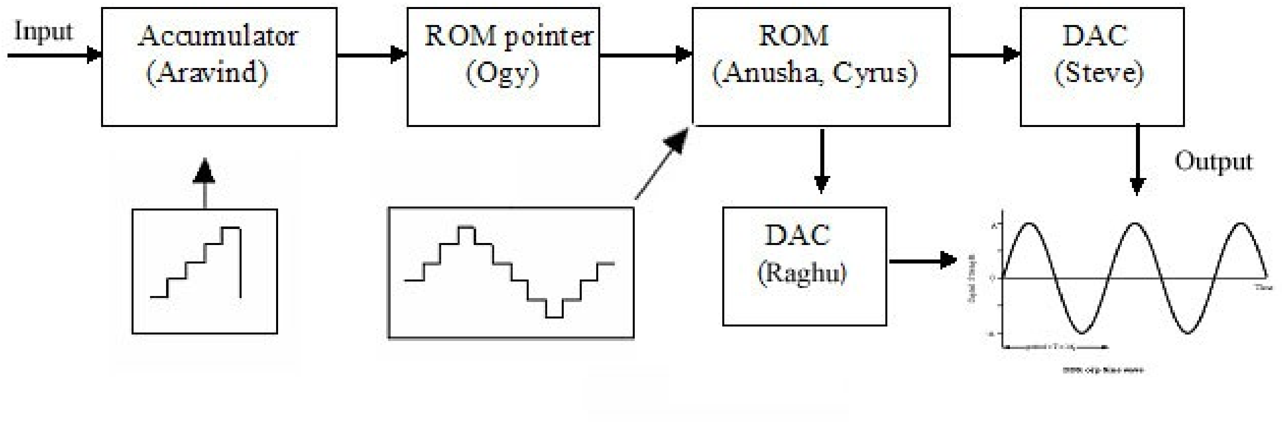


Fig. 1.1 DDS Block Diagram

As you can see in the diagram, the accumulator is just a repeating ramping input. It goes through all of the memory cells stored in the ROM and then repeats, not giving a sine wave at all. The sine wave comes from two things; the ROM itself and which quadrant the signal is in.

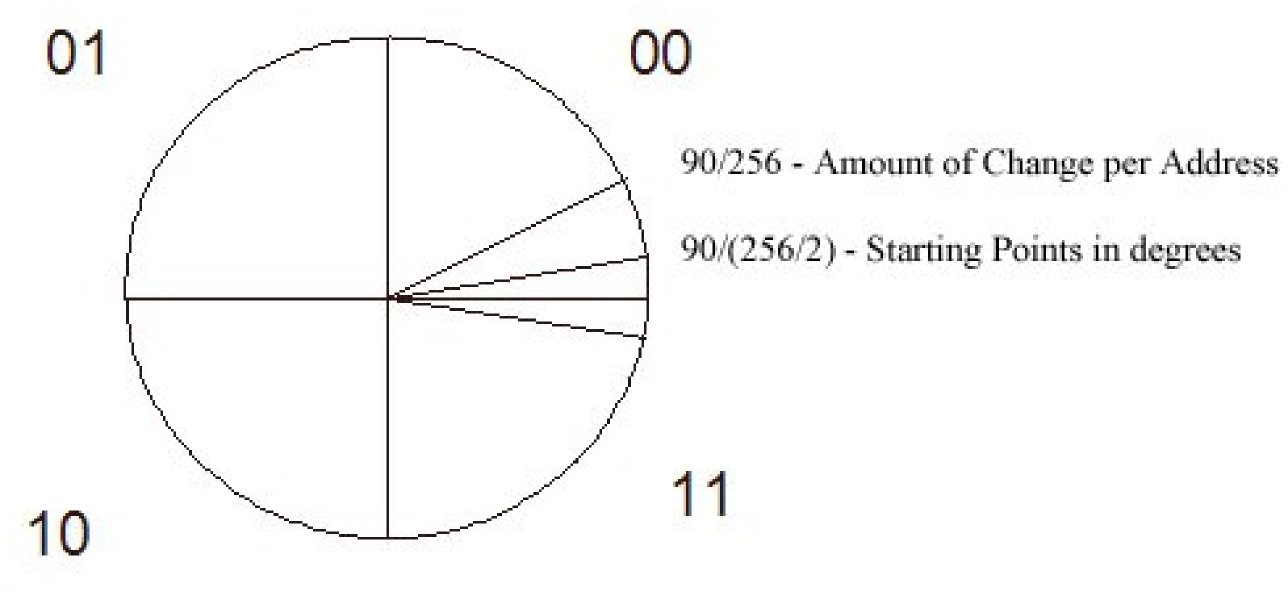

01
00
90/256 - Amount of Change per Address
90/(256/2) - Starting Points in degrees
11
10

Fig. 1.2 Quadrant Diagram

The ROM stores all of the values of quadrant 00, but nothing more. It is possible to only have 90 degrees of sine information because of the repetitive nature of a sine wave; you only need to change certain things to account for each quadrant. The ROM Pointer is used to decide which quadrant, and therefore which word line, should be accessed.

2 Specification Overview

A DDS system generates one or more frequency from a single reference frequency using digital logic. In this project, my objective was to design a 256 bit ROM which stores the amplitude values for each phase of DDS output waveform. A 256 bit ROM could be designed by using four 64 bit ROMs to both minimize the capacitance and also occupy less space on the chip.

Design Specifications

The different specifications that were given for the design of a ROM are as follows:

Name	Specification
Number of rows	256
Number of columns	9
Frequency	50 MHz
Voltage	0 to 5 V

Table 2.1 Design Specifications

3 ROM Overview

To implement our design specification, we had to look realistically at what can fit into a chip. The chip is about 900 microns by 900 microns square, so that directly impacted the design of our ROM. We needed to break the 256 rows of memory up into four 64 bit ROM blocks in order to fit; this also helps lower the capacitance on the lines as well.

To make a 64 bit ROM, we would need the 6 Least Significant Bits coming from the Accumulator. By using these 6 bits, we could make a tree decoder capable of accessing one of 64 different rows. Each time a signal is used, the output is split to either a high or low signal. After using 6 of these signals, the total output can encompass any one of 64 rows, or 2^6.

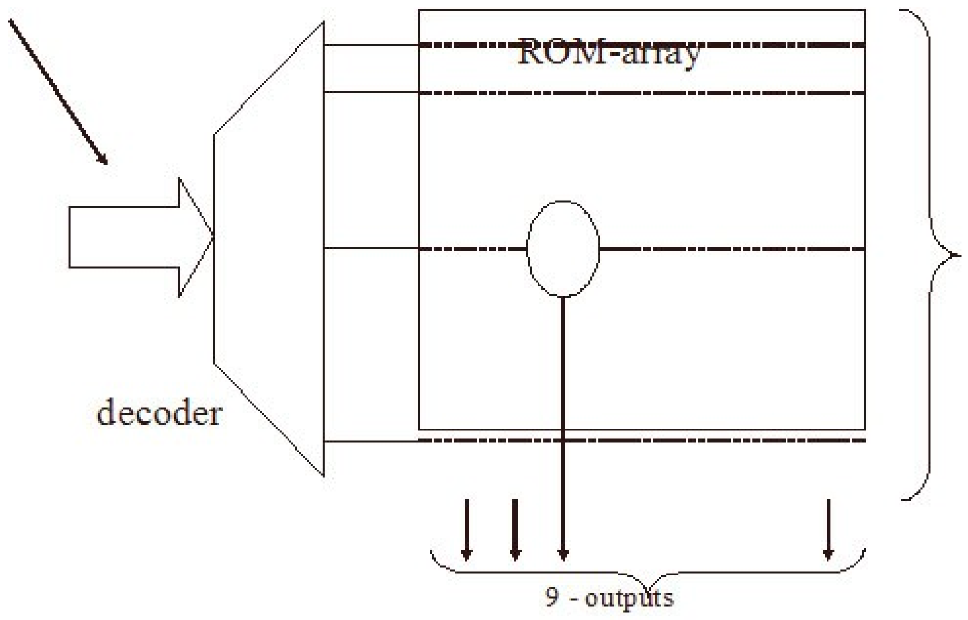
To link these four blocks of ROM together, a two stage Multiplexer was needed. The ROM's themselves would output 10 bits each, on every clock cycle, and these outputs would go into the first stage. ROM A and ROM B would go into one, ROM C and ROM D the other. The 7th least significant bit is used to determine which of these is passed along; the output is sent to the 2nd stage. Here either (A or B) or (C or D) is chosen, using the Most Significant Bit to make this choice. By using this method, one can accurately choose which of the four ROM's to access and which row in that ROM to access.

3.1 ROM Structure

The basic ROM structure consists of grid-like building blocks, enabling ROM's of any logical size to be created without any extra design work involved after the initial setup. These building blocks are broken into four categories; Tree Decoder, Memory Array, Sense Amplifiers and Multiplexers.

The best way to begin explaining the ROM is by describing the Memory Array. As our goal was a 256 ROM with a 9 bit output, we will start with the identical 64 bit ROM array. The Memory stored in these ROM's consists of 64 rows with 9 columns; each row is called a "Word" and each column a "Bit". Whatever is stored in these different memory cells can be accessed at any time and also need to be programmed to be accurate.

To access these cells a tree decoder is necessary. The tree decoder takes the 6 least significant bits from the accumulator and uses it to control 6 stages. Using these stages, we split the output 32 times to create a possibility of 64 different rows. This is how we can select an individual row, or alternatively, word line.



Once these word lines are activated, the data stored on the bit lines is accessible by the sense amplifier at the bottom of the ROM. These sense amp's are used just to increase the signal, purifying any distortion and re-centering it according to the voltage rails that are in use. In our case, this means the sense amp decides whether the signal bit is high or low and then outputs either 0V or 5V accordingly.

The output coming from these sense amps go directly into the multiplexers. As mentioned before, ROM A and B share a MUX while ROM C and D share the other. After the first stage, the two multiplexers both output to the third, which is controlled by the Most Significant Bit. This output is determined both by which ROM is selected and which word line in the ROM is selected; it is the final output of the ROM. From here, it goes to the latch and is latched before being sent to the DAC.

4 Schematics

4.1 64 bit Memory Units

Each 64 bit memory unit is made up of 5 different parts; Inputs, Tree Decoders, the Pull-Up Network, Memory Cells and the Sense Amplifiers. The Inputs take the signal from the accumulator and invert it and both signals are propagated through the tree decoder. This in turn controls the decoder and selects which row of memory to access. When accessed, the Pull-Up Network gives a high voltage to each bit line unless there is an NMOS circuit stored in one of the memory cells. If an NMOS exists, it pulls the bit line low. These bit lines are connected to the sense amplifiers; the sense amps reset the signal to either a pure low or a pure high, outputting to a later stage in the ROM.

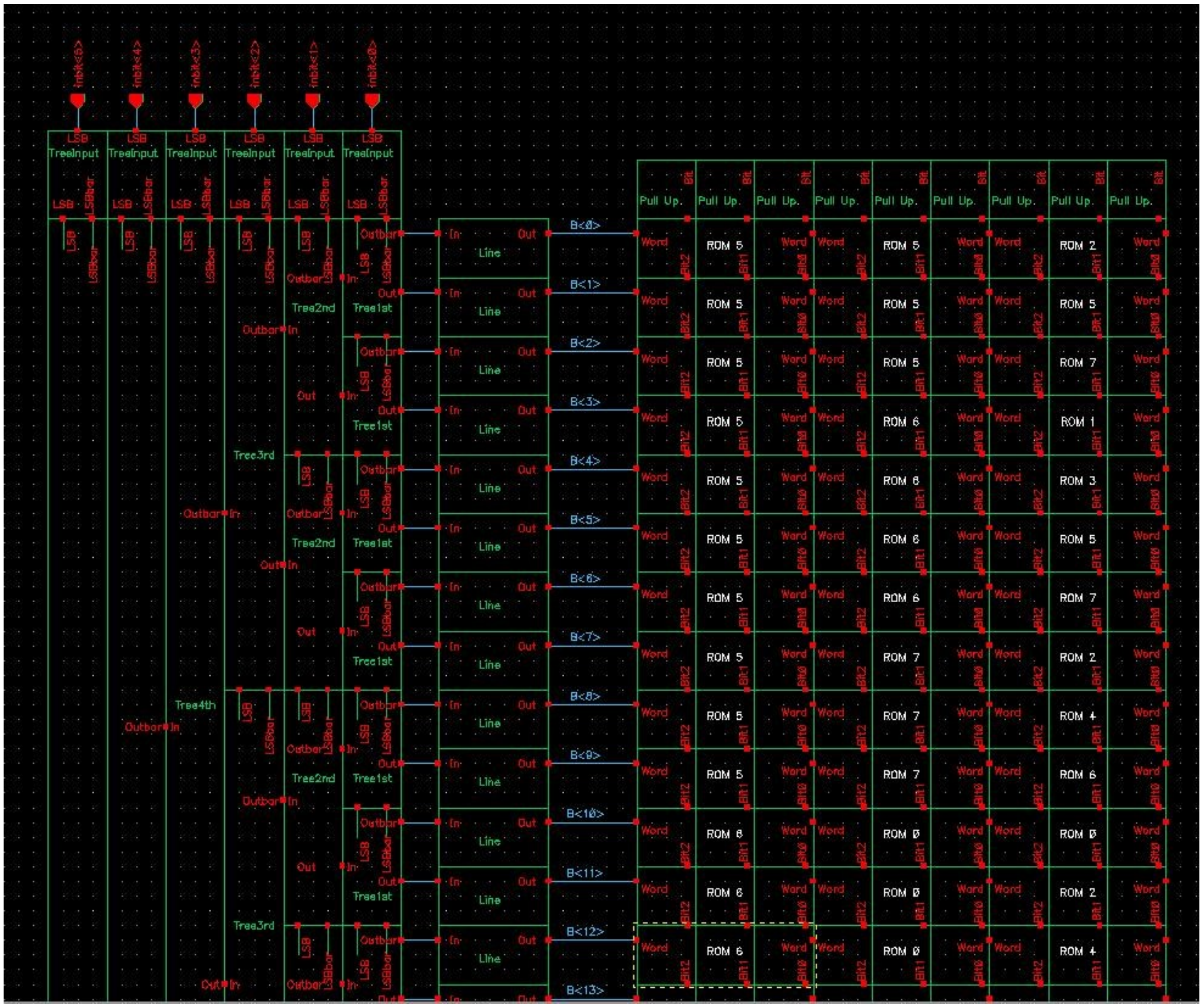


Fig. 4.1 64 Bit ROM Schematic

4.2 Tree Decoder Input

There are 7 inputs to the tree decoder; 6 signal lines and Vdd. Vdd is the input to the very first stage, giving it a steady 5 volt signal to propagate. The other 6 signal lines are the 6 least significant bits coming from the accumulator. These same signal lines go to all four 64 bit ROM's; all select the same row in memory. The two most significant bits are used to choose which ROM is selected; that is how we were able to fit a 256 bit ROM in a rather tight space.

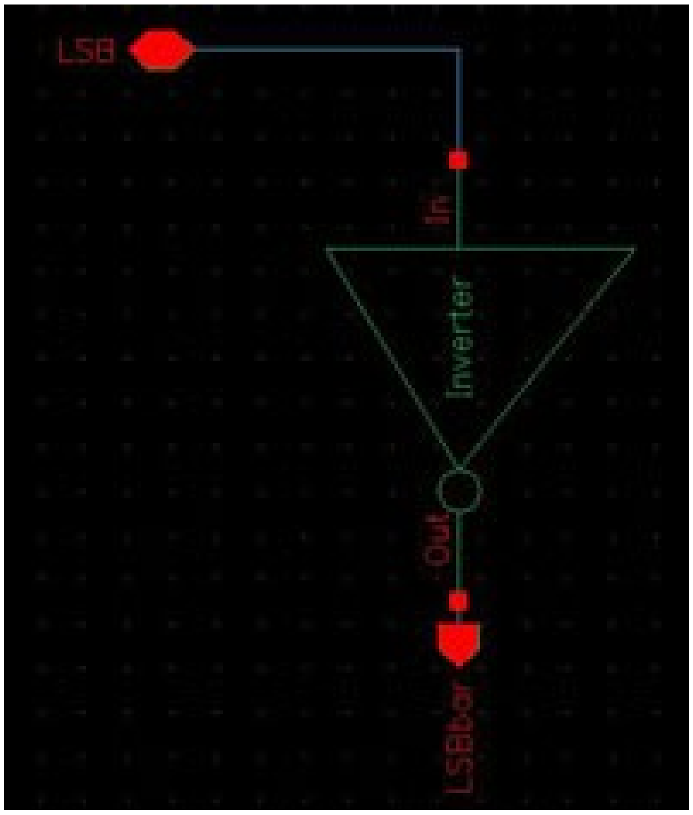


Fig. 4.2.1 Tree Decoder Input Schematic

The six signal lines are inverted and both the signal and its inversion are required for the tree decoder. They are both used in selecting which output is high; we originally had one signal, but this required us to use a complementary CMOS design. Using both signal lines allows us to ditch the PMOS and have an easier implementation.

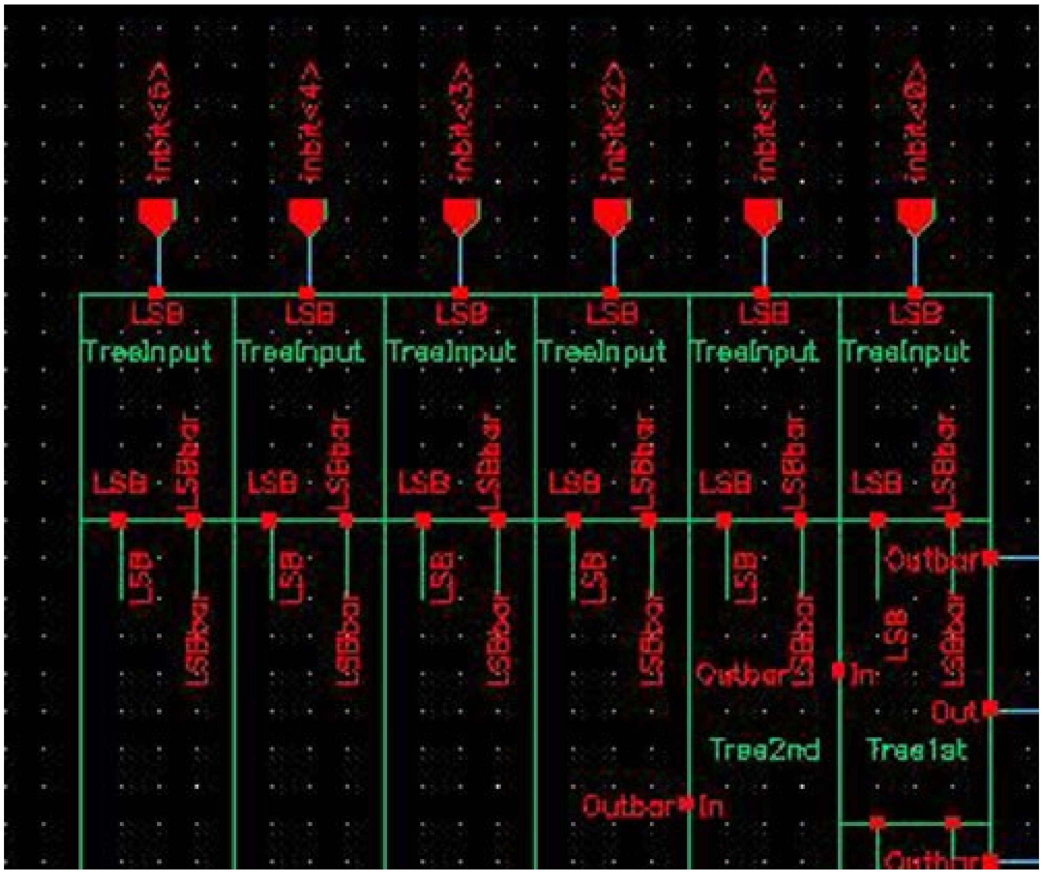


Fig. 4.2.2 Tree Decoder Input Schematic Overview

4.3 Tree Decoder

The Tree Decoder is the means by which a particular row in memory is selected. There are six stages to it; all are identical except for the third stage, which is designed to keep the output from floating. Each instance of the tree decoder has an input that is either high or low; if it is low, nothing will output but it works in the same way. When the input is high, the decoder looks to the signal bit to find out whether to make output a high or a low. This high signal is sent in a direct path through six levels of the tree decoder to its final destination; a row in memory.

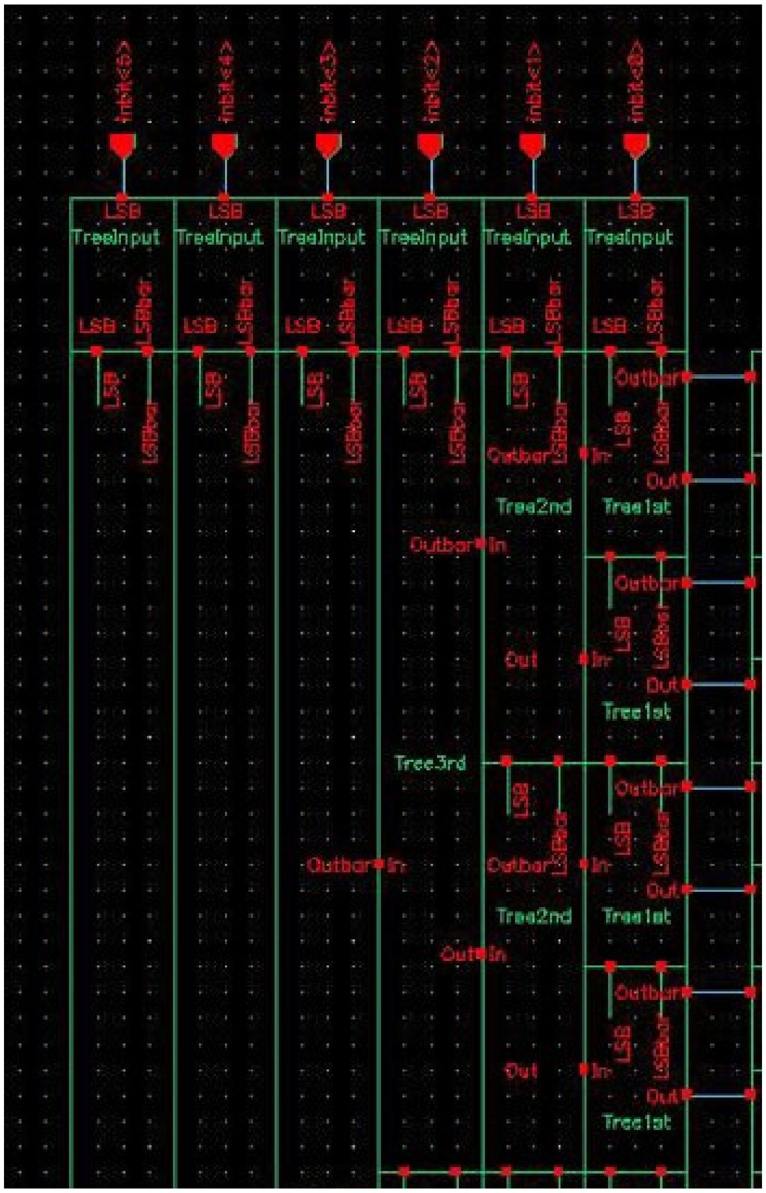


Fig. 4.3.1 Tree Decoder Schematic Overview

To propagate this high signal, two NMOS transistors are used in the circuit. They both are connected to the input, with their output connected to either Output or Output bar. To create the output, the signal line is connected to the gate of the NMOS transistor; when the signal is high it turns on, making the output high as well. If the signal is low, the NMOS turns off and the output is low. However, when the signal line is low, its inverse is high, and that is what is connected to the gate of the other NMOS. This is how Output bar is created; whenever the signal switches from high to low, one transistor turns on while the other turns off, with a negligible overlap between the two.

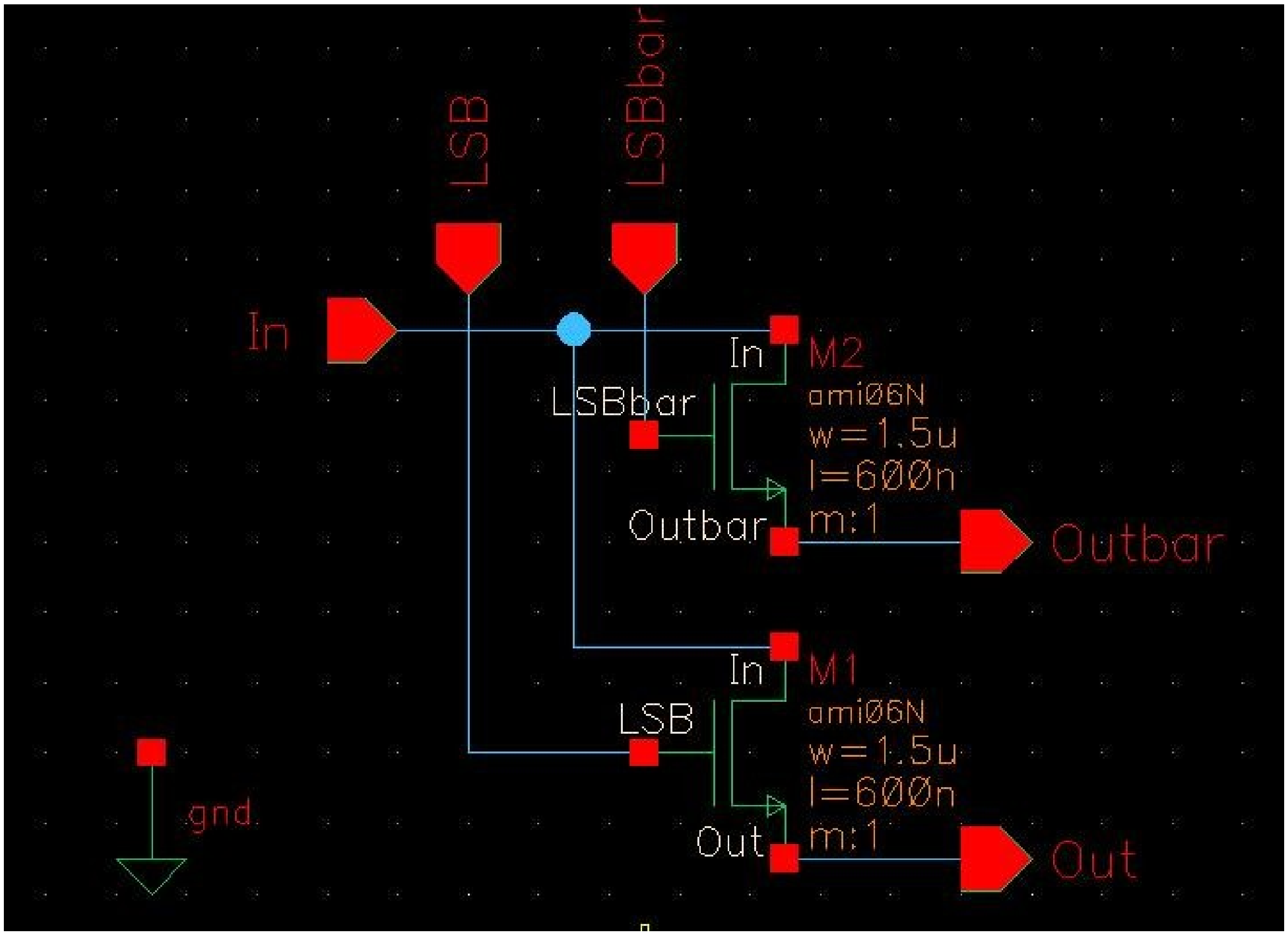


Fig. 4.3.2 Tree Decoder Schematic

For the third stage of the tree decoder, two extra NMOS transistors are added; one to the Output and one to the Output bar. They are both used to ground the outputs; their gates are connected to the signal that is opposite of the output, so whenever the output is low the transistors turn on. This guarantees that the output is not floating; it pulls it to ground and in doing helps lessen the dampening to the signal.

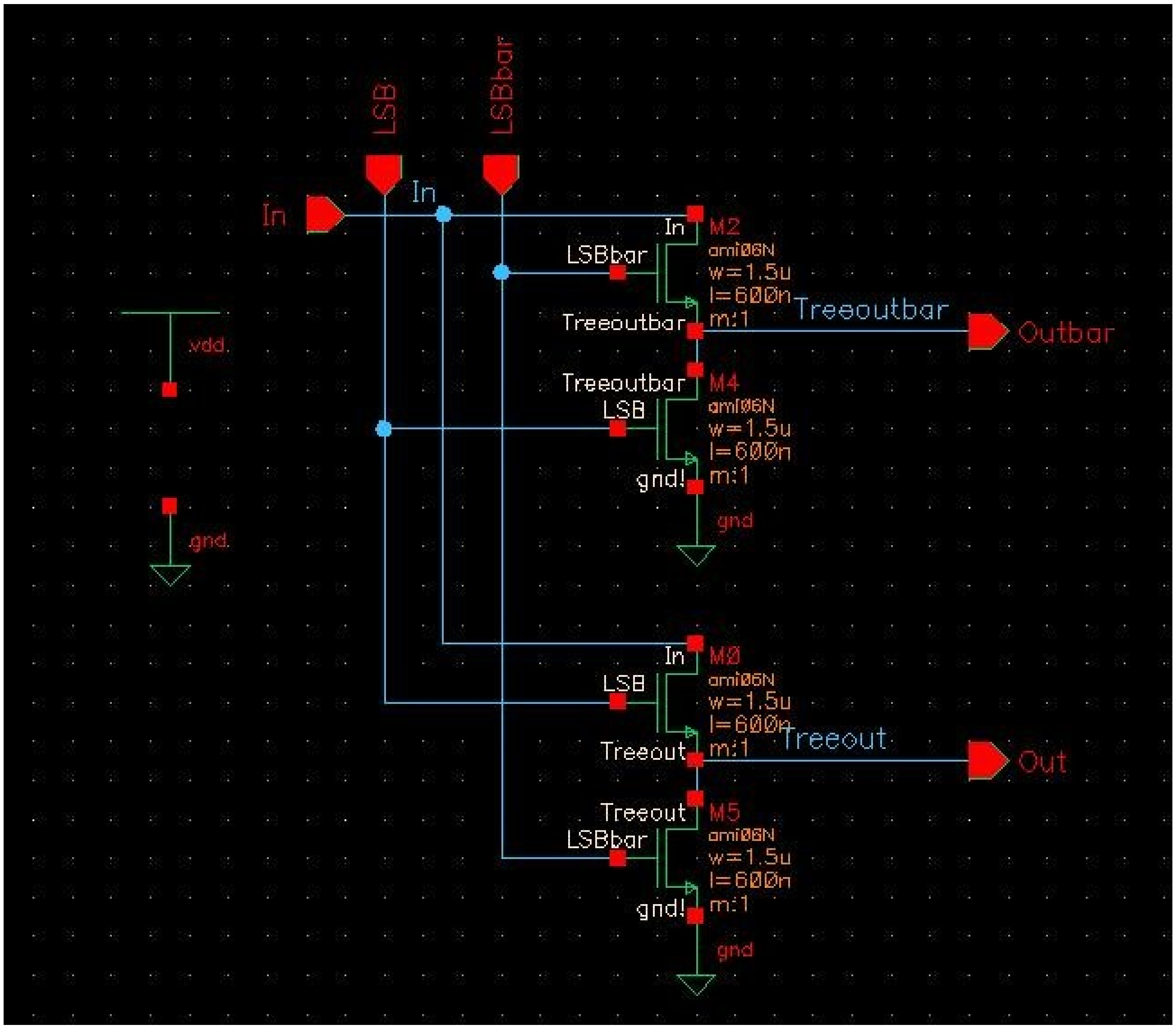


Fig. 4.3.3 Tree Decoder 3rd Stage Schematic

At the end of the Tree Decoder, a very important combination of devices is connected. To compensate for the possibility of a floating output, another NMOS transistor is attached to the output. The output line is attached to both its source and its gate, with the drain attached to ground. This requires that a voltage source remain high. Any errant signals would trigger the NMOS to turn on and give a path to ground, draining any voltage from the line. If it is a true positive, however, the signal will continue to the offset inverter.

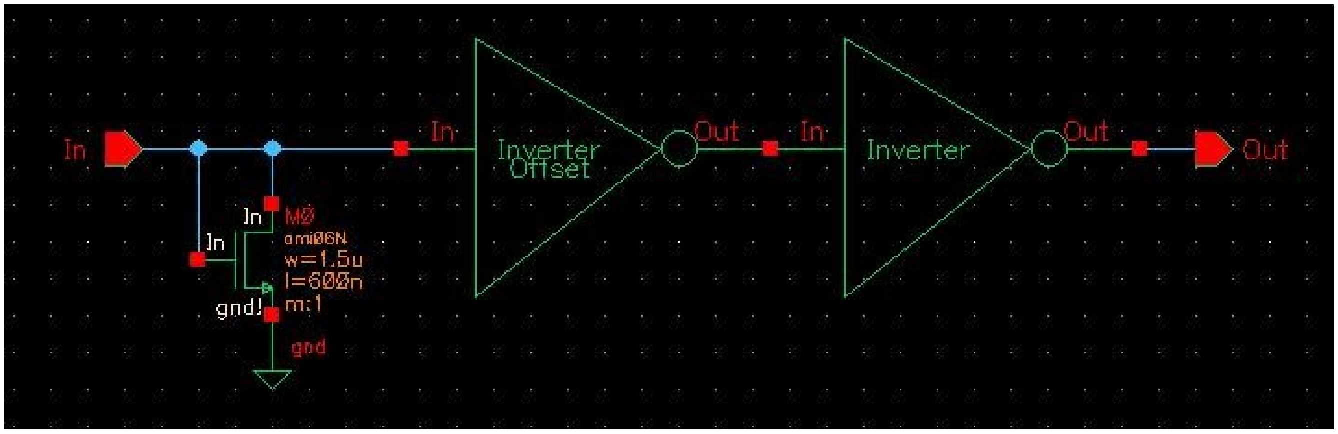


Fig. 4.3.4 Tree Line Schematic

The offset inverter was sized with its NMOS transistor six times larger than minimum sized, while the PMOS remained minimum sized. This gives it a lower threshold voltage, allowing it to turn on at a lower voltage level. This is important because the signal is dampened by the time it gets to this point; the voltage high level decreases from the 5 volts at Vdd to about 3.8 volts. Switching at a lower level correctly reflects the midpoint of the signal; the second inverter inverts the signal and resets the midpoint to 2.5 volts. The combination of these two allow the signal to go from 0 volts to 5 volts again, which is important for the sense amplifier.

4.4 The Memory Structure

The Memory Structure is what takes the information provided by the tree decoder and outputs the selected stored memory bits. Each row is called a word line and a properly working tree decoder only selects one word at a time. The number of output bits is equal to the number of columns; each output is connected to every memory cell in its column by the bit line. The way we implemented this is if no word is selected, the bit lines will all be high. There is a difference in trying to send a low or a high signal to the sense amplifiers-- we chose a high to be our norm, meaning that our memory has a structure of lines and bits without anything connecting between them. Whether the word line is chosen or not, the bit line doesn't connect to it and so doesn't care. To have a low output, however, requires an NMOS transistor in each memory cell. This transistor is connected to both the bit line and the word line, giving the bit line a path to ground. The word line is attached to the gate of the transistor; whenever the word line goes high, the gate is high and the NMOS turns on. This in turn pulls the Bit Line low, so the output seen at the sense amplifier is zero.

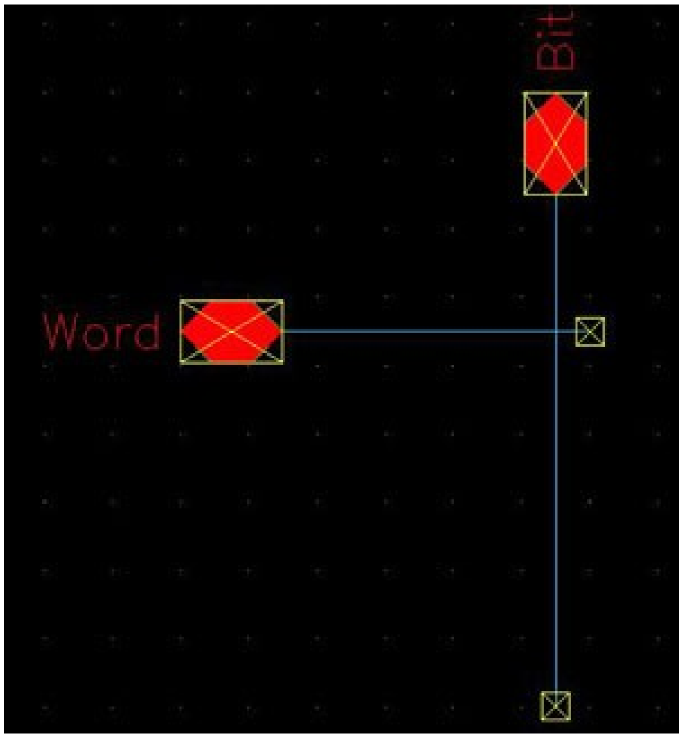


Fig. 4.4.1 Memory Cell One

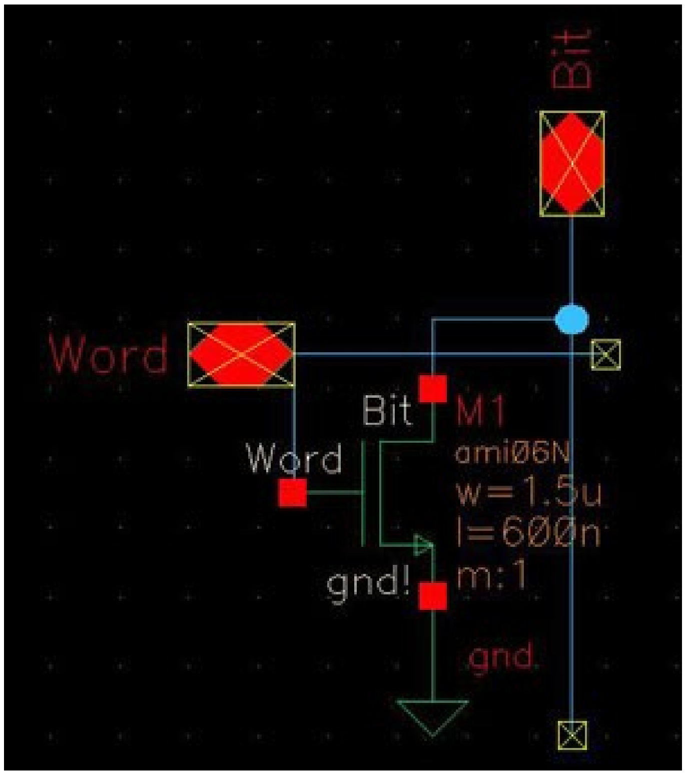


Fig. 4.4.2 Memory Cell Zero

4.5 Pull-Up Network

The Pull-Up Network is a simple PMOS transistor. Its source is Vdd, its drain is the bit line and its gate is connected to ground. This means that it is always on, but it is a necessary safeguard due to current considerations.

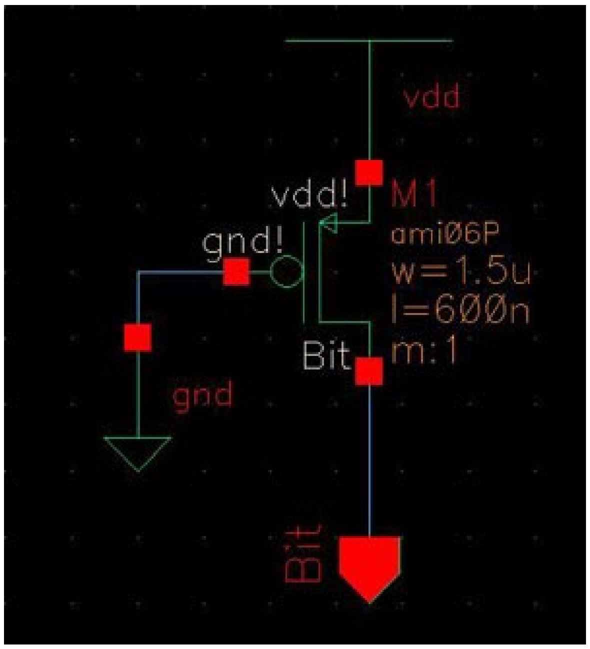


Fig. 4.5 Pull Up Network Schematic

4.6 Sense Amp

The sense amp works in the same way that the offset inverter in the tree line does. It consists of two minimum sized inverters in series, resetting the output to either a pure high or a pure low. The sense amp does have an offset inverter, but it wasn't necessary to change sizes; the switching midpoint is the same as it should be. The only reason for the sense amp, in our situation, was that the high signal was not a pure high. It dropped down by at least a volt, so to pull it back up and give a clean output, the inverters are there.

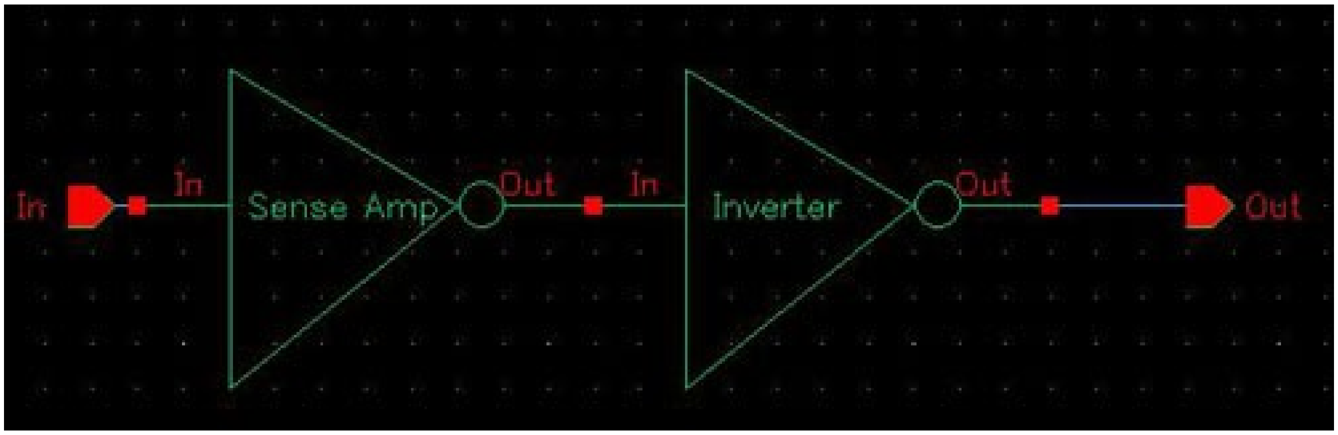


Fig. 4.6 Sense Amp Schematic

4.7 Multiplexers

The Multiplexers are in two stages, with each stage being identical. The two multiplexers in the first stage accept two sets of 9 inputs and output 9 bits to the second stage. The second stage chooses which of the first multiplexers to pass on to the output.

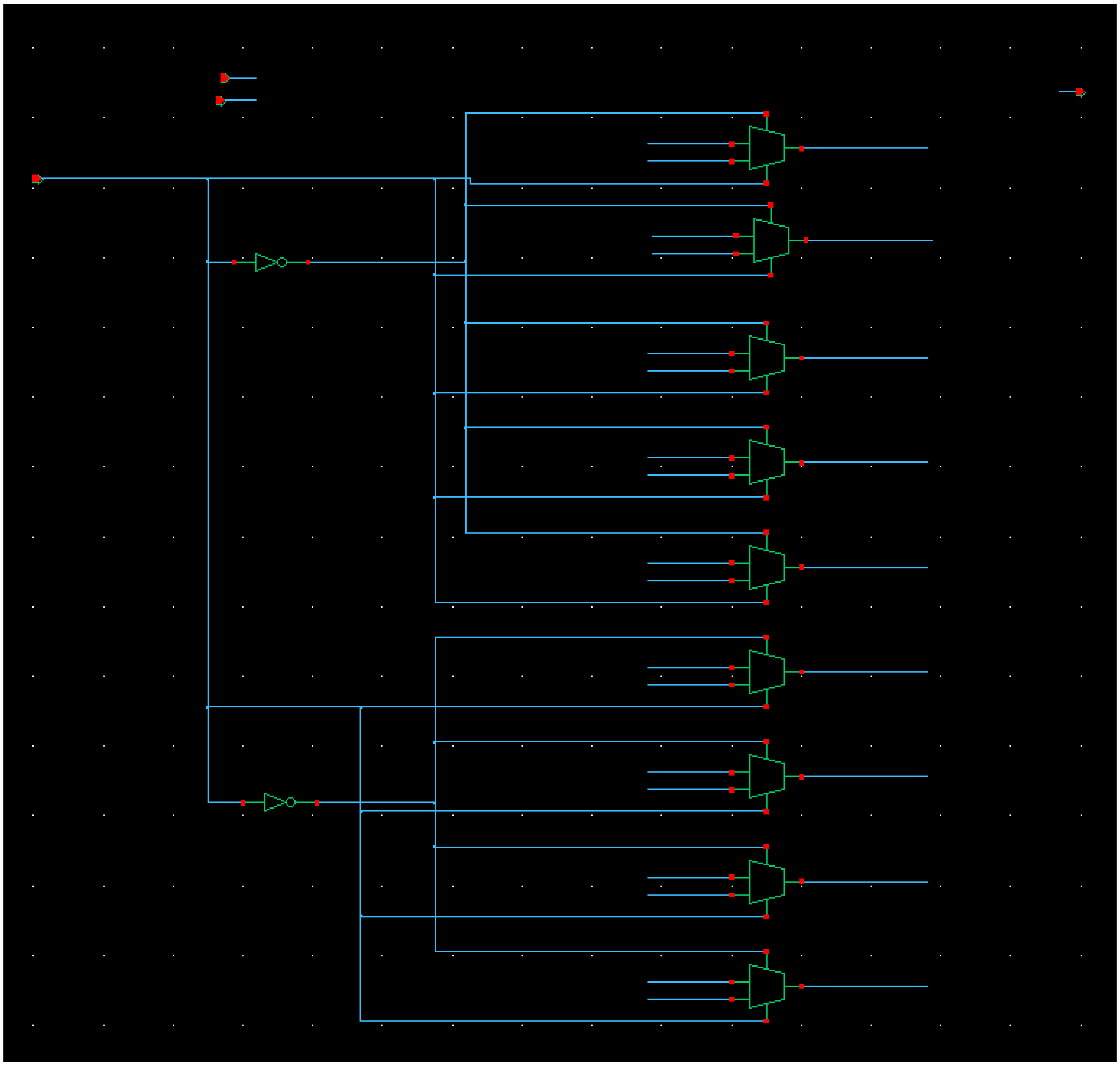


Fig. 4.7.1 Multiplexer Schematic

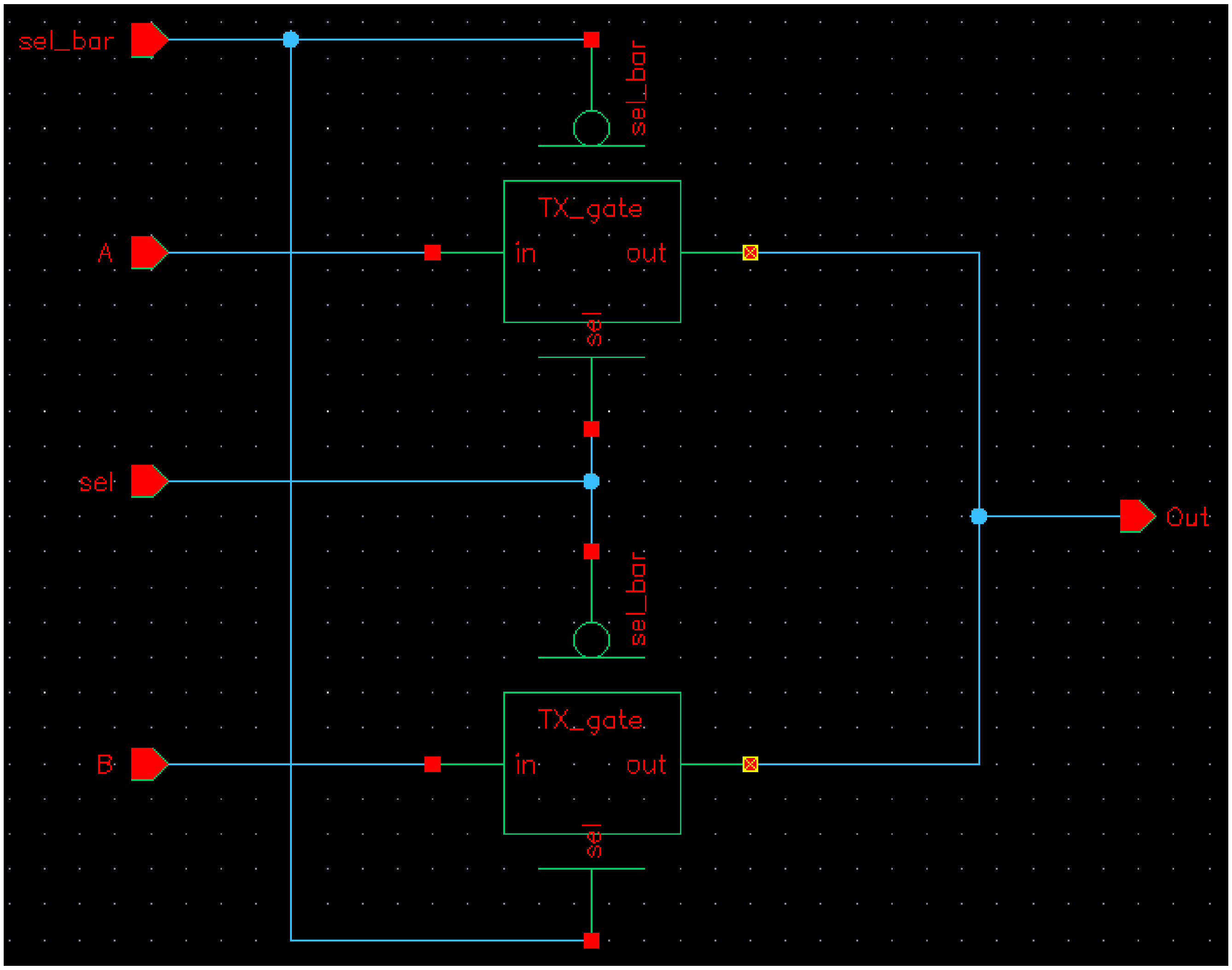


Fig. 4.7.2 Multiplexer 2:1 Schematic

4.8 Latch

The outputs from the multiplexer are latched using clocked D flip-flops. The edge triggered D flip-flop can be easily constructed from the RS flip-flop. One essential point about the D flip-flop is that when the clock input falls to logic 0 and the outputs can change state, the Q output always takes on the state of the D input at the moment of the clock edge. This circuit has two D latches in a master- slave configuration driven by a clock. The different logic circuits used in this design are 2 inverters, 4 AND gates and 4 NOR gates. The sizing of the transistors was done for each of this digital logic. Fig 2.5.1 shows the schematic view of the D flip-flop used for latching.

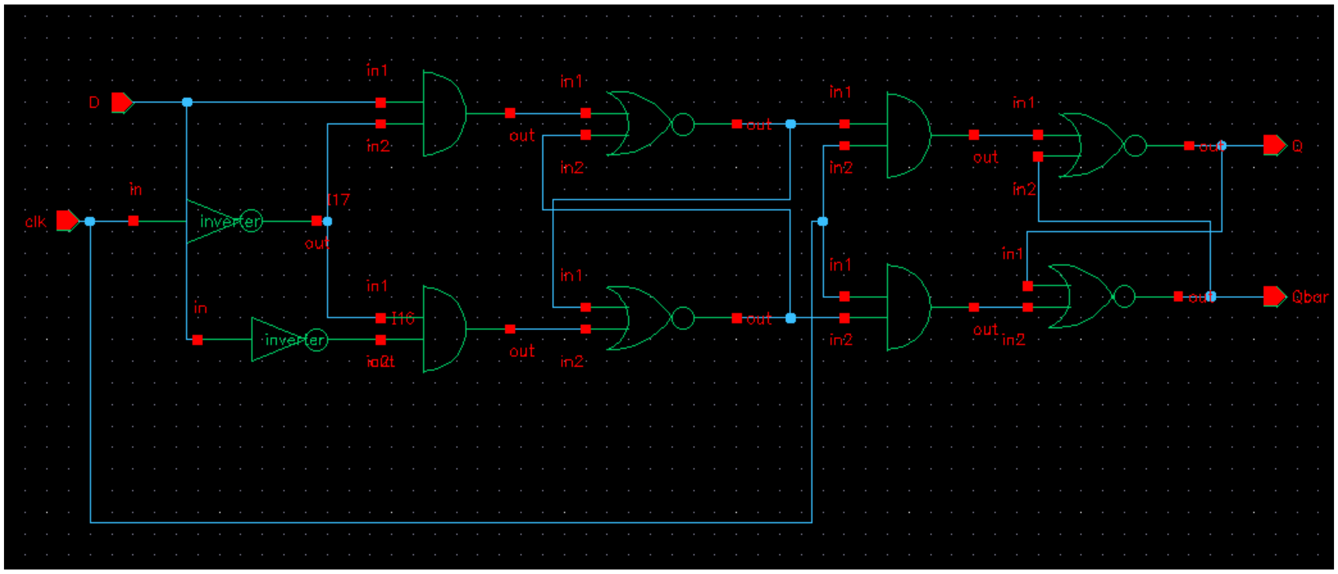


Fig. 4.8.1 Clocked D Flip Flop Schematic

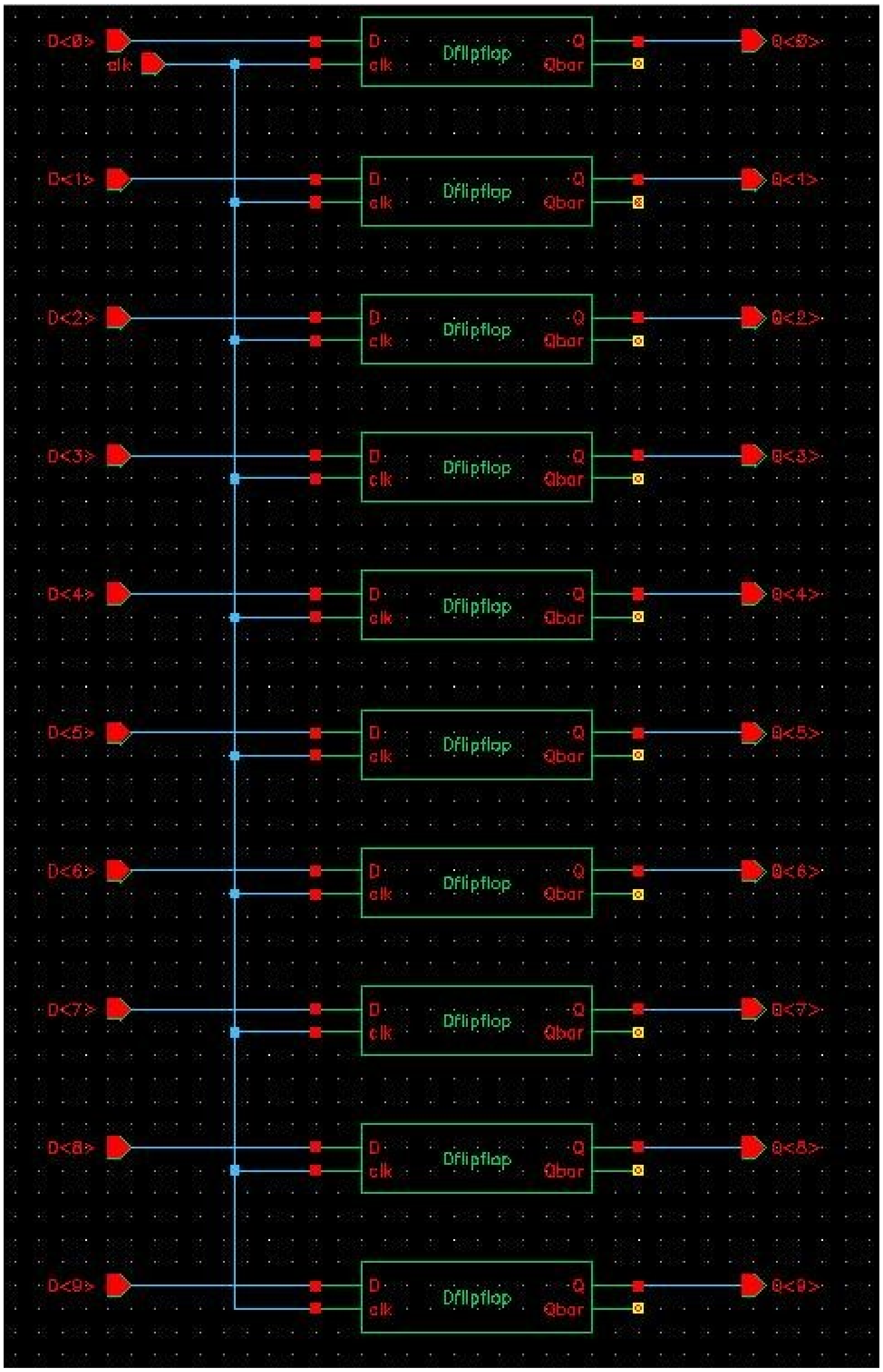


Fig. 4.8.2 Clocked D Flip Flop Schematic Overview

5 Layouts

5.1 64 bit Memory Units

The completed 64 bit Memory Units are tall and skinny; the limiting factor in sizing was the height of the memory cells. The height and width were the same, but they only needed to be 9 units wide whereas they were 64 units tall.

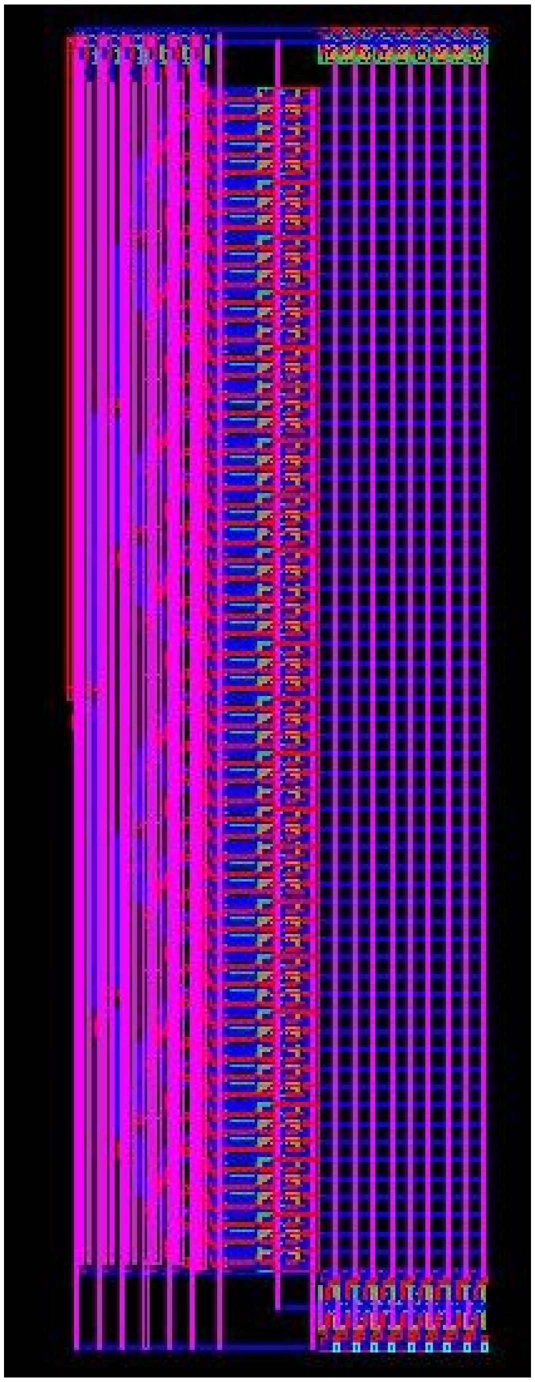


Fig. 5.1 64 Bit ROM Layout

5.2 Tree Decoder Input

The Tree Decoder Input would typically be a very simple layout, but in our application we wanted to fit several in a very tight space. It ended up fitting very nicely and is a very simple design.

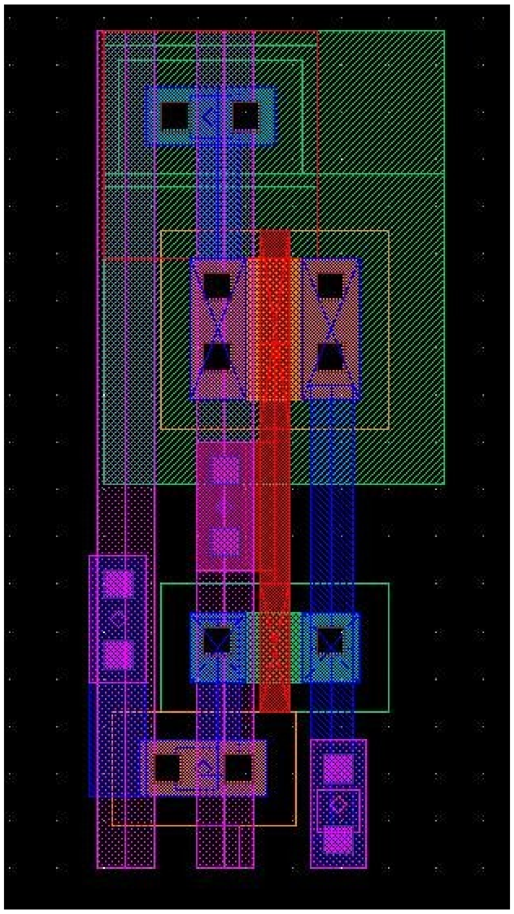


Fig. 5.2.1 Tree Decoder Input Layout

Notice that the input bit from the top attaches to the gate and then continues through as its own output; this is the signal, the output of the inverter is the signal bar. By using Metal 2 as we did, it allows each piece to be put in a grid while having all of the vertical metal 2 strips overlap. This comes into play when we lay out the tree decoder itself; by keeping the units very basic, arrays of tree decoders are very easily implemented.

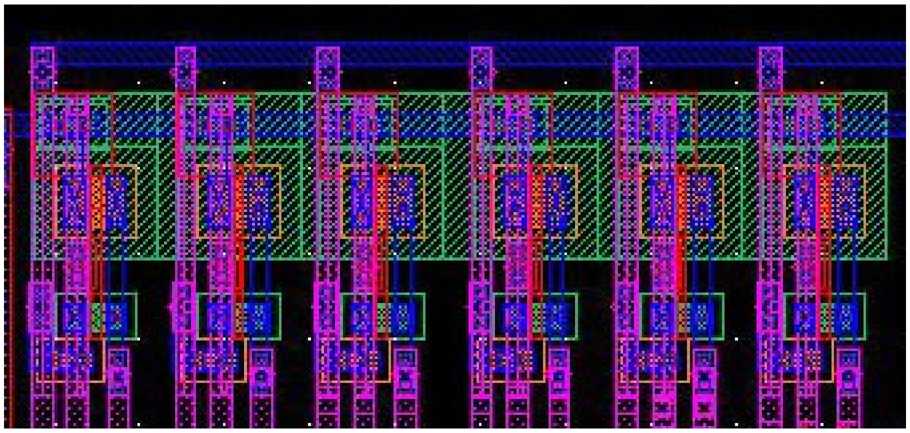


Fig. 5.2.2 Tree Decoder Input Layout Overview

5.3 Tree Decoder

The layout of the tree decoder becomes increasingly cluttered the more stages there are. The minimum-sized stage 6 are crammed next to each other as tight as they can fit; it slowly expands as you go to the left, with Stage 1 being the entire length of the ROM.

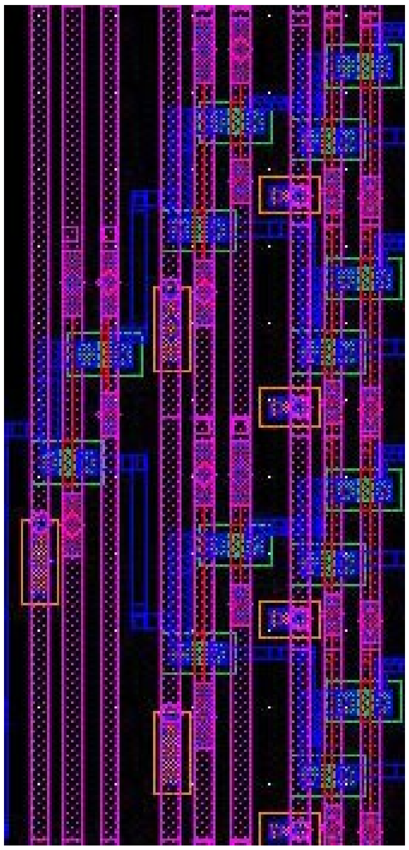


Fig. 5.3.1 Tree Decoder Layout Overview

The individual layouts for each stage are the same, with the only difference being the height involved. Stage 5 has to be twice as tall as stage 6, so the lengths of the connecting wires is longer. From stage 6 to 5 there is also a slight change as well; with more space to expand vertically, the units become less wide. Following will be examples of the stages with the exception of Stage 1 and 2. Both of these stages are identical to Stage 4 but their height makes them very hard to see.

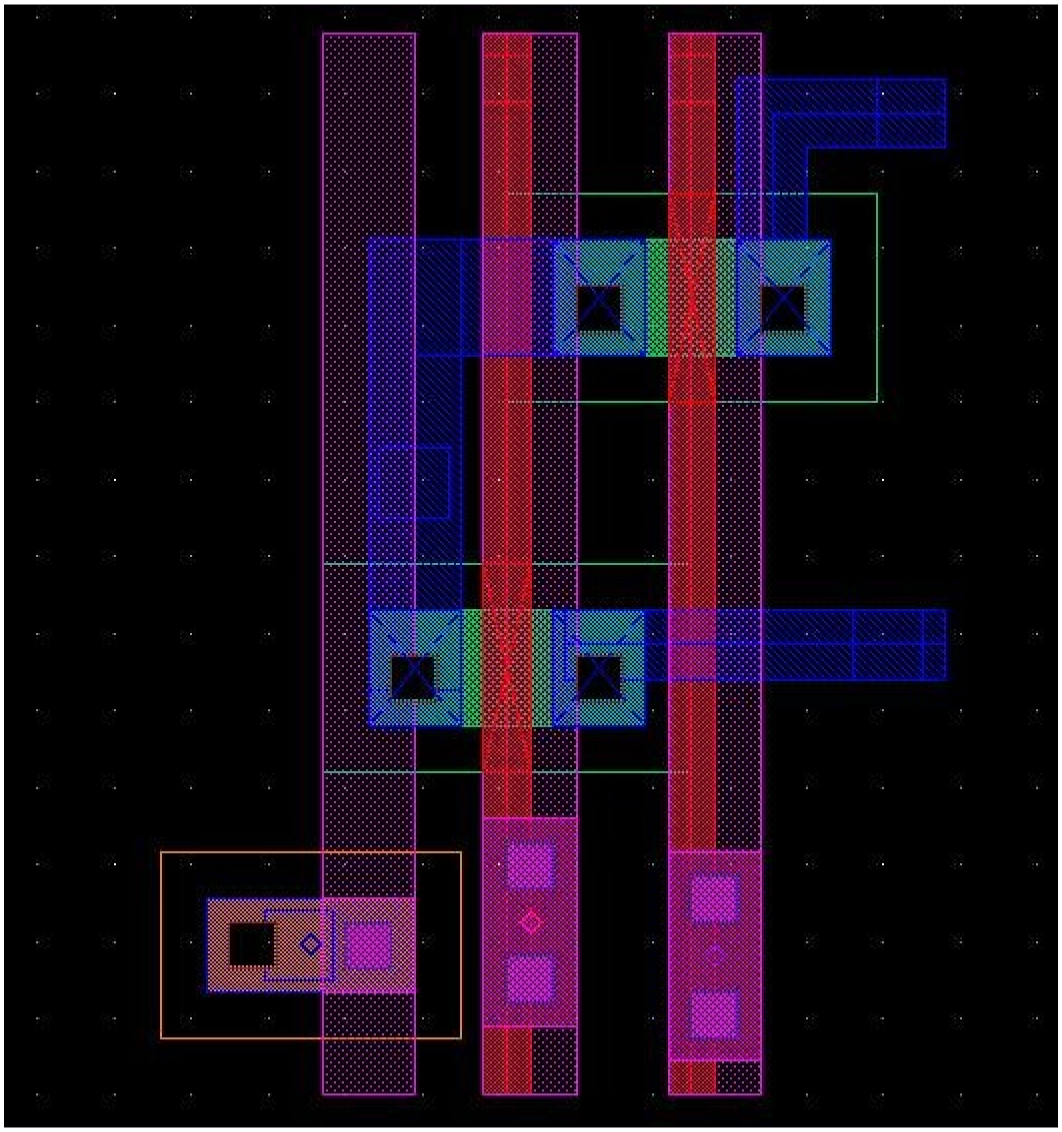


Fig. 5.3.2 Tree Decoder Stage 6 Layout

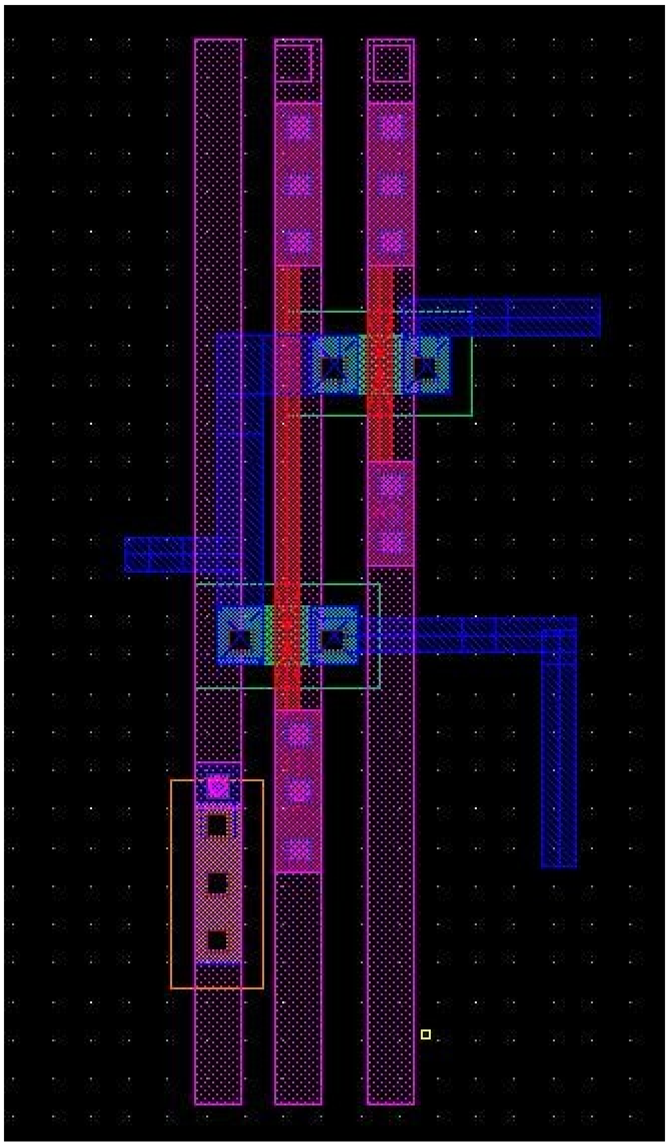


Fig. 5.3.3 Tree Decoder Stage 5 Layout

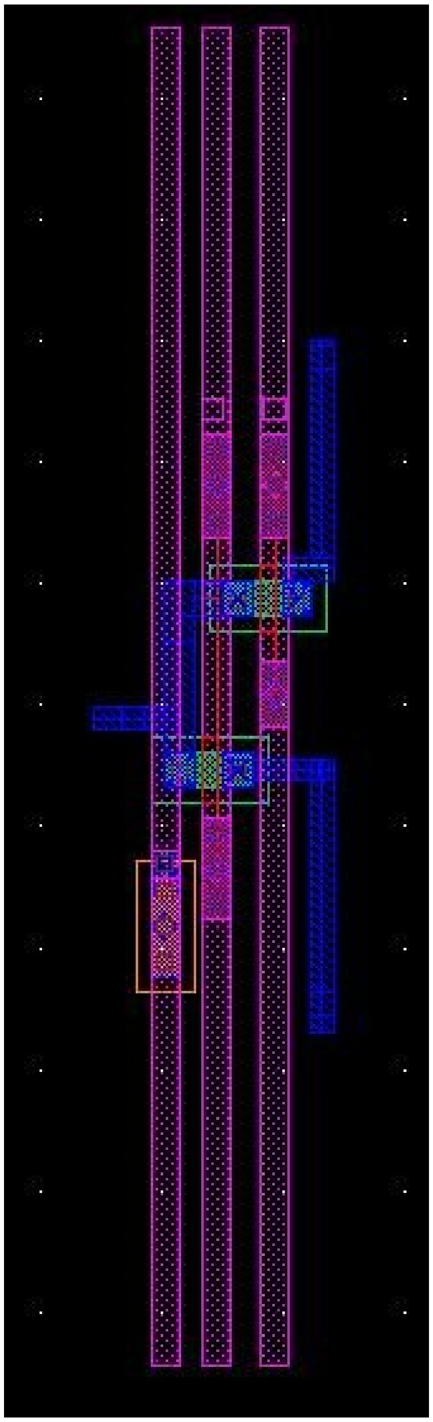


Fig. 5.3.4 Tree Decoder Stage 4 Layout

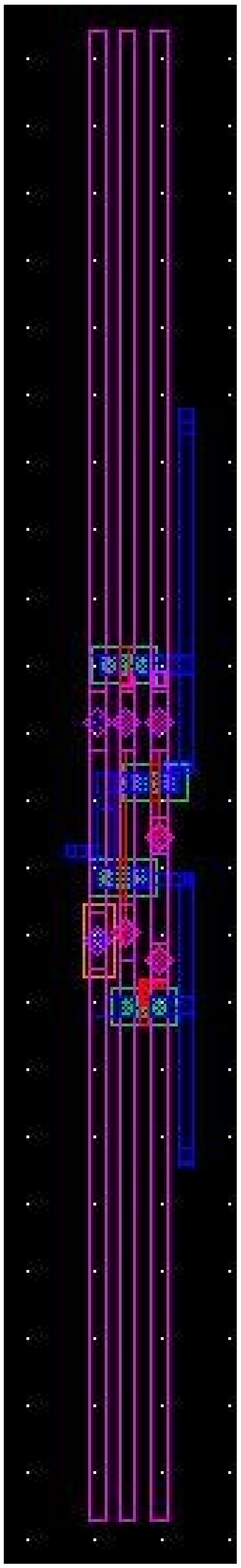
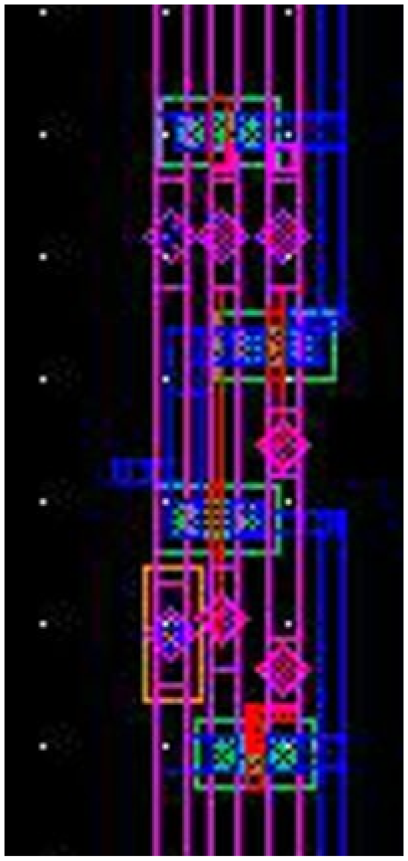



Fig. 5.3.5 Tree Decoder Stage 4 Layout

The Tree Decoder outputs each row into the Tree Line, as shown here:

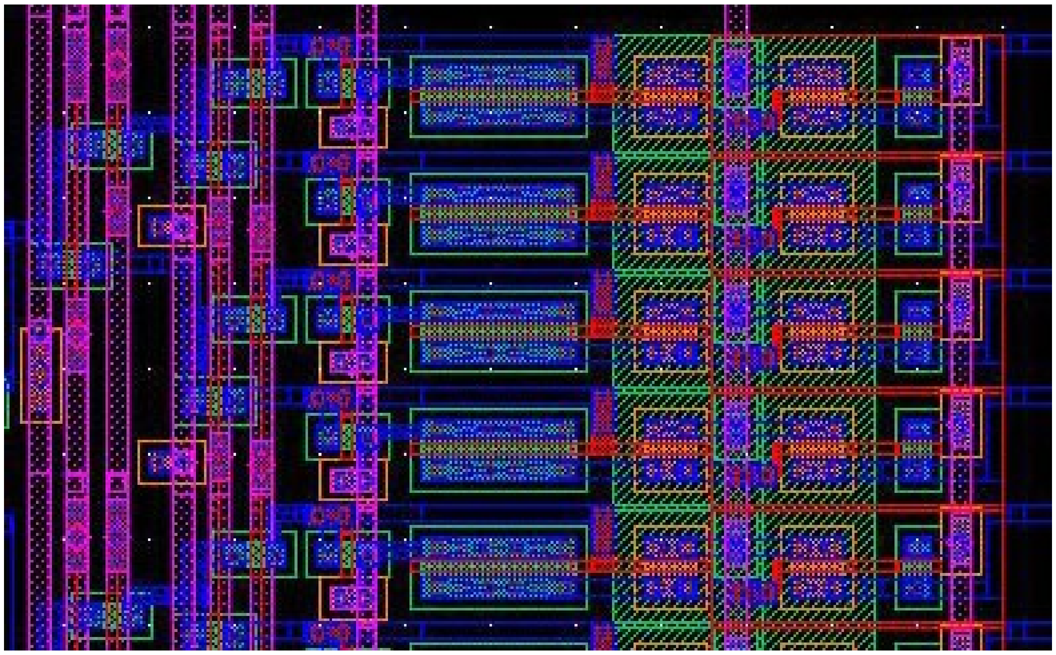


Fig. 5.3.6 Tree Line Layout Overview

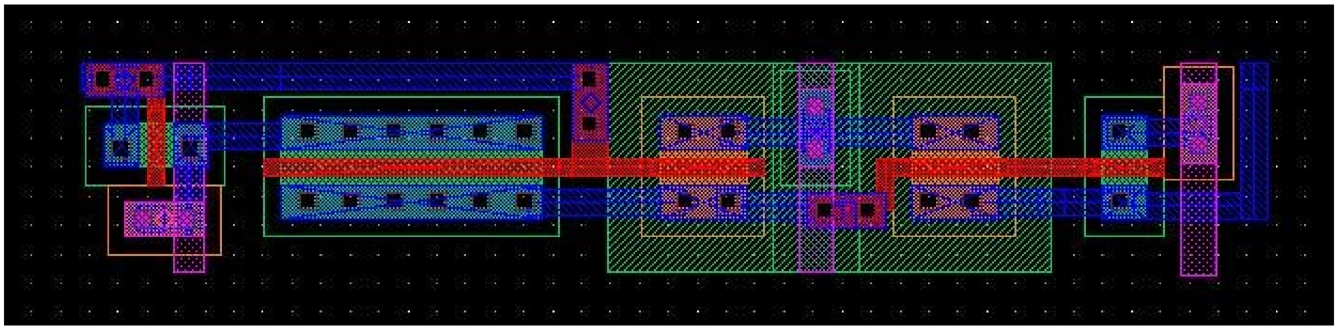


Fig. 5.3.7 Tree Line Layout

The hardest part in designing this was the height restrictions. Because the height was meant to be the minimum height of each memory cell, it forced the width to increase tremendously. The various components almost don't fit correctly, but they just manage to do so. Notice also the strips of metal 2 going vertically; these connect all of the grounds and Vdd's together without requiring any special wiring when you use the Tree Line in your layout. The same thing happened with the rest of the Tree Decoder; each signal bit, and its inversion, are automatically connected to the following decoder.

5.4 The Memory Structure

The Memory Structure was the basic building block around which the rest of the ROM functioned. With a network of bit lines and word lines, the Memory Structure looked like a grid.

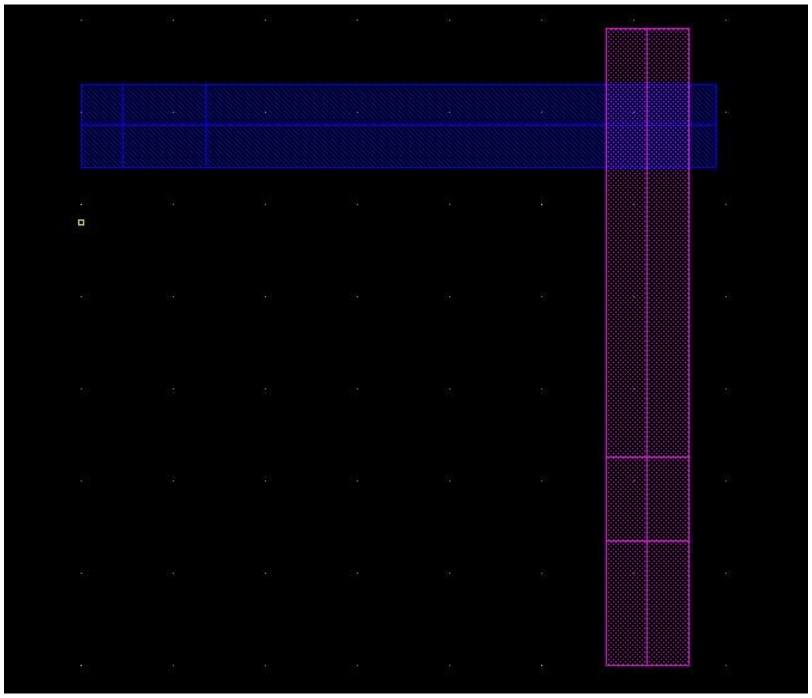


Fig. 5.4.1 Memory Structure Layout

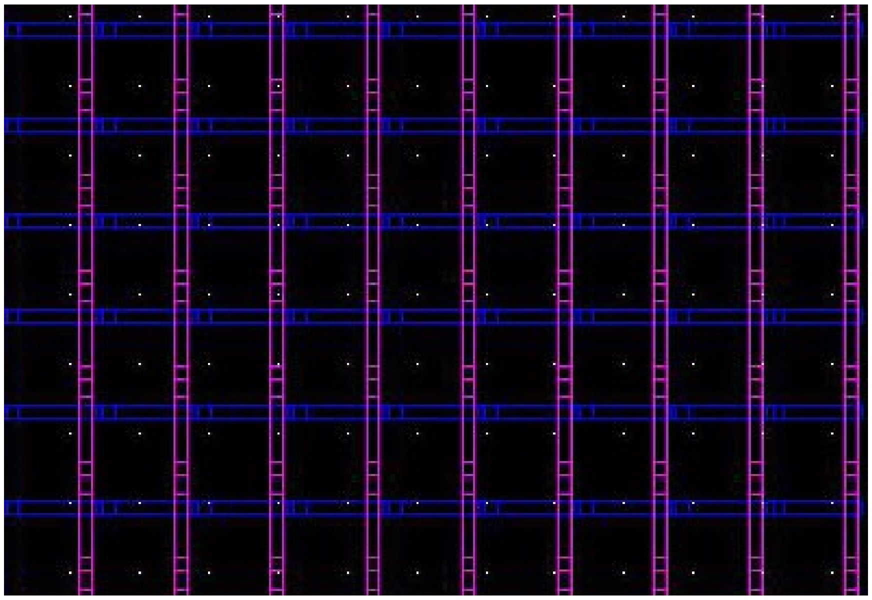


Fig. 5.4.2 Memory Structure Layout Overview

The stored zeros used the same physical structure but added an NMOS in the middle. This NMOS and it's properties are what controlled the overall size of the ROM-- there was no way to make it any smaller than we did.

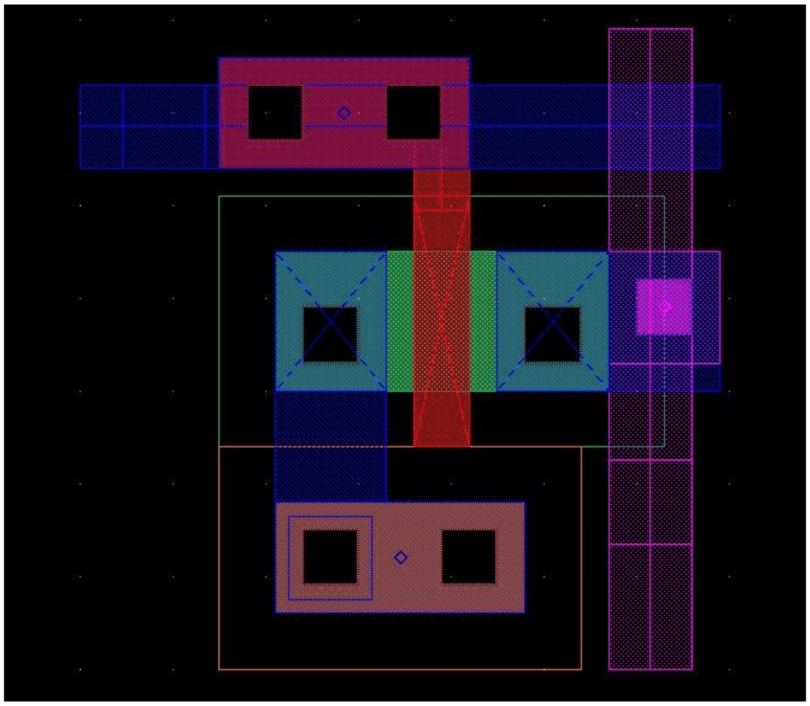


Fig. 5.4.3 Memory One Layout

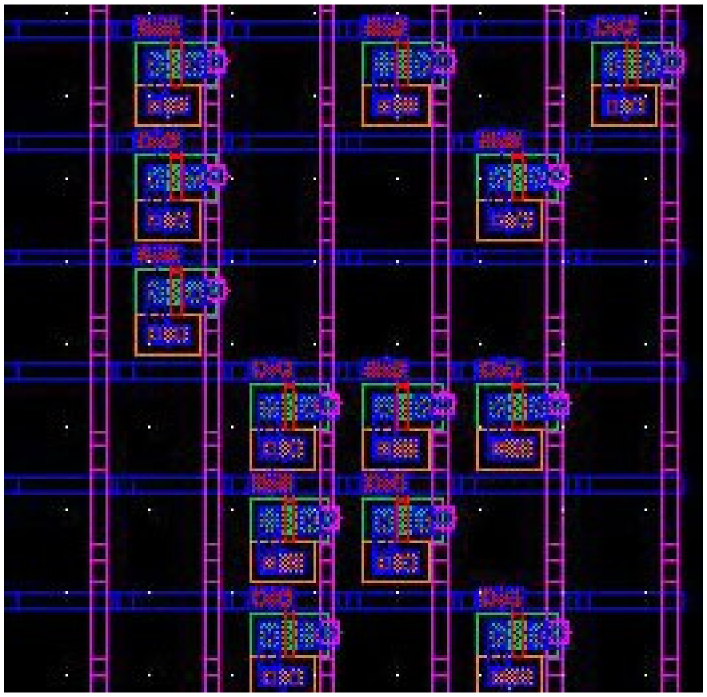


Fig. 5.4.4 Memory One Layout Overview

5.5 Pull-Up Network

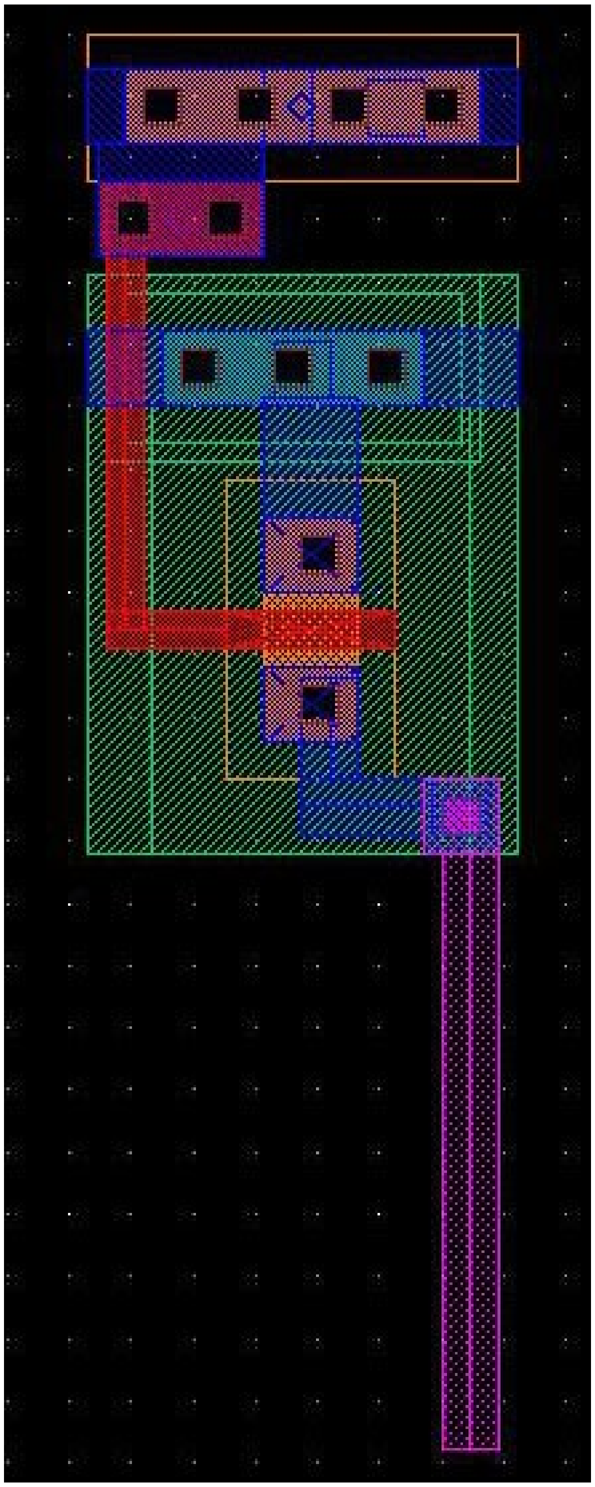


Fig. 5.5.1 Pull-Up Network Layout

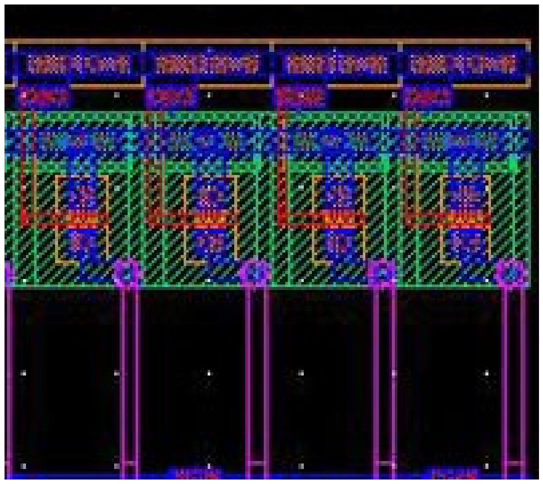


Fig. 5.5.2 Pull-Up Network Layout Overview

5.6 Sense Amplifiers

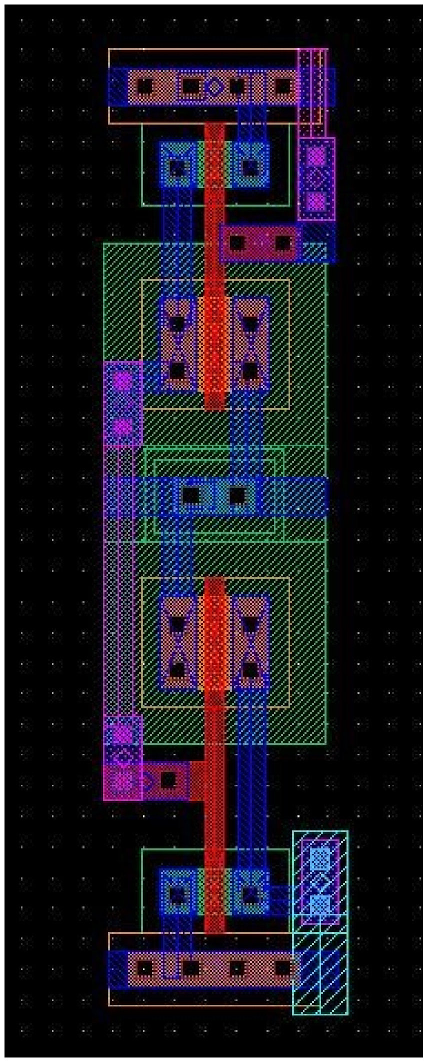


Fig. 5.6 Sense Amplifier Layout

5.7 Multiplexers

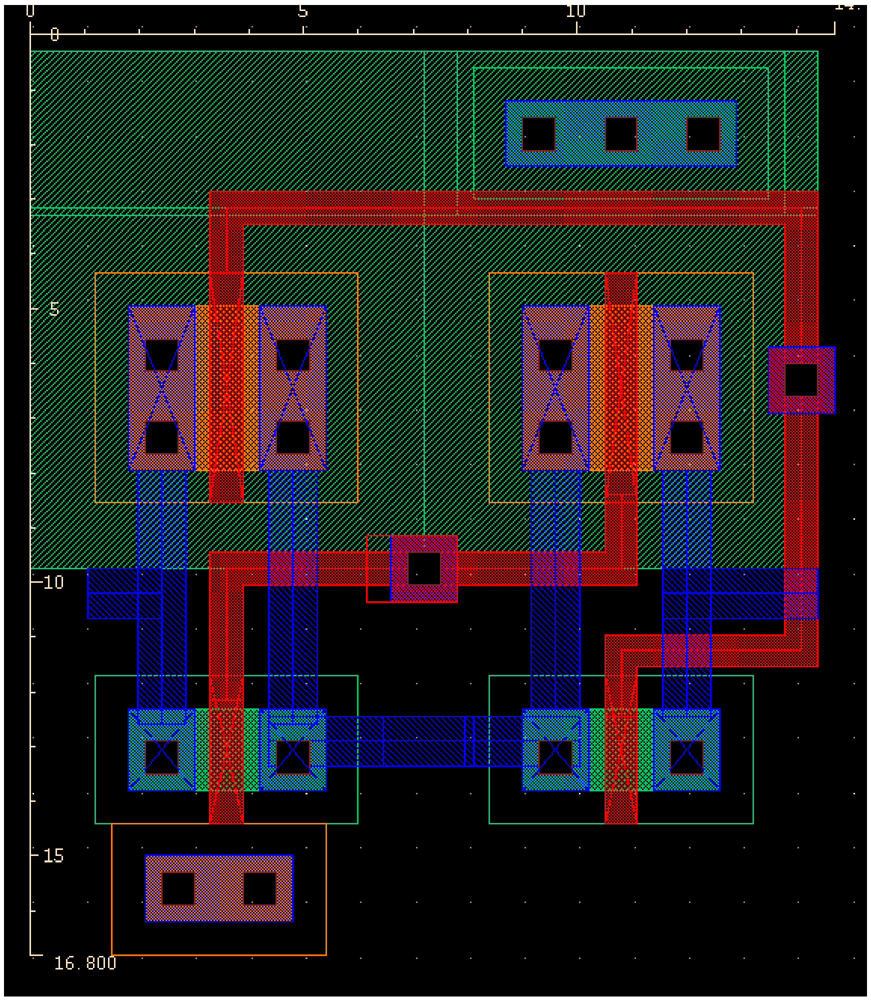


Fig. 5.7.1 Two to One MUX Layout

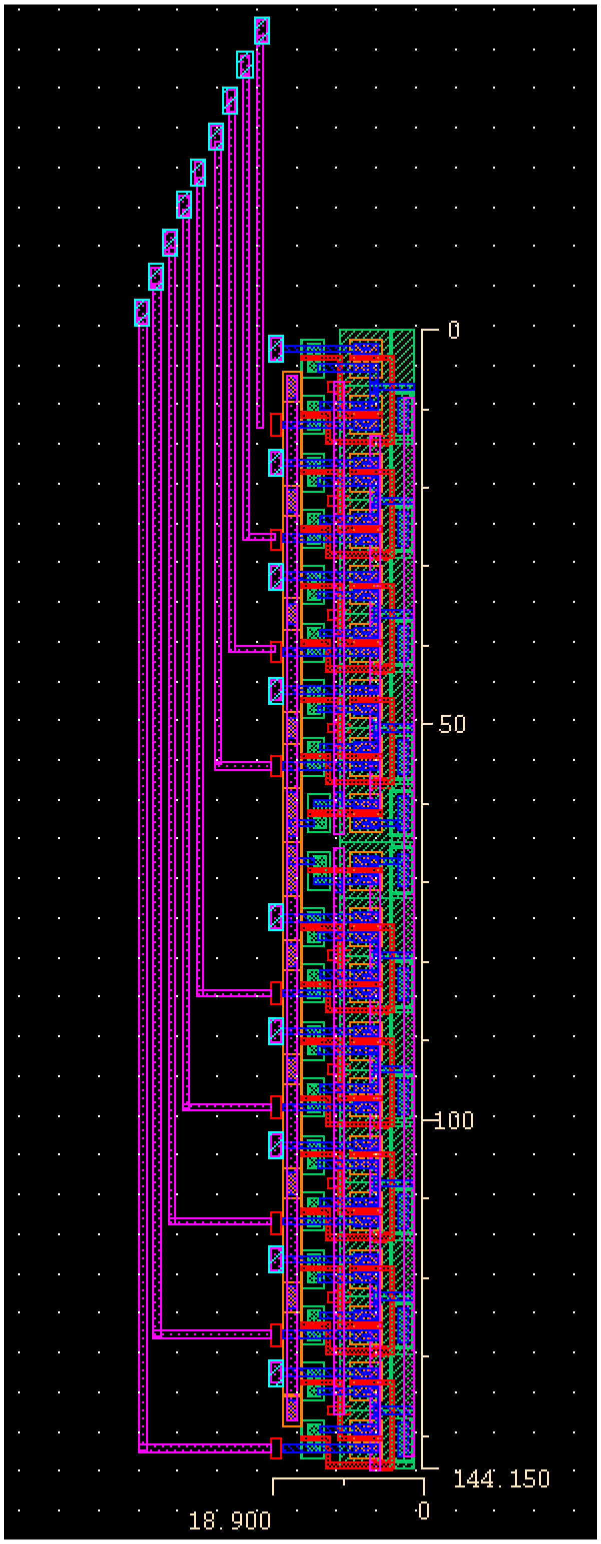


Fig. 5.7.2 Two to One MUX Layout Overview

5.8 Latch

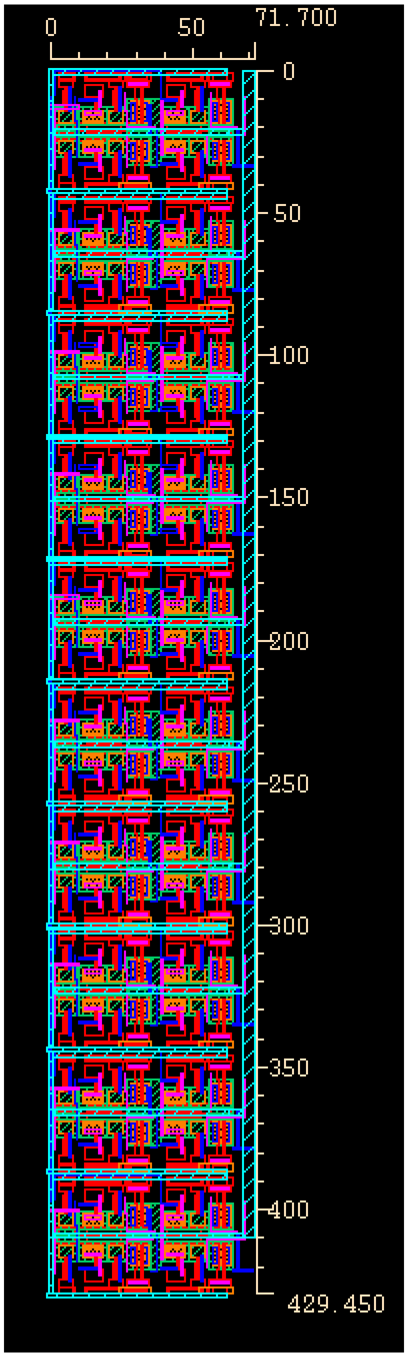


Fig. 5.8 Latch Layout

6 Sizing

To achieve minimum sizing, the limiting factor must be minimized. In this case, the limiting factor is the height of the individual memory cell; the smaller it is, the smaller the ROM will be. This is due to the fact that there are 64 rows of memory, compared to only 9 columns. The smallest the memory cell can be is dependent on the NMOS transistor that has to fit within it; there is no way to avoid overlapping cells unless everything fits inside the block of memory. Minimum sizing gives us Fig. 6.1.

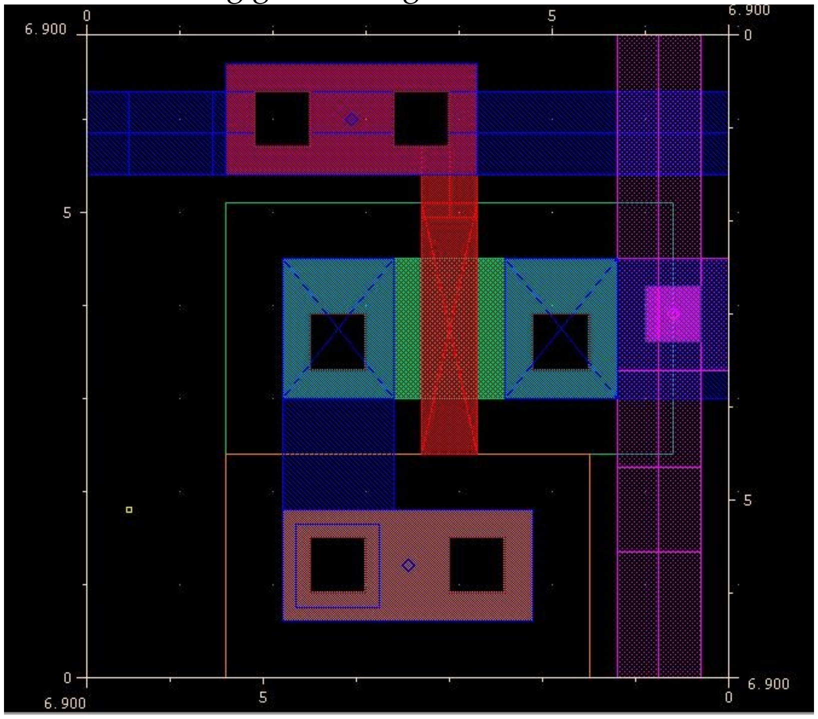


Fig. 6.1 Minimum Sized Memory Cell

This minimum sized cell forms the basis for everything else; to achieve minimum size, all of the neighboring components must fit within the height or width dimensions as well. The Pull-Up Network and the Sense Amp both need to be 6.9 micron wide in order to fit; conversely the 6th stage of the tree decoder and the tree line can only be 6.9 micron high.

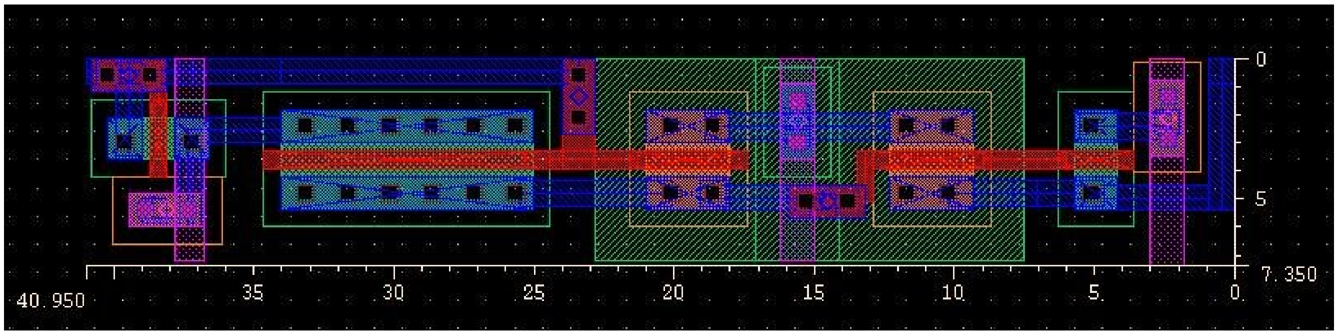


Fig. 6.2 Height Restricted Tree Line

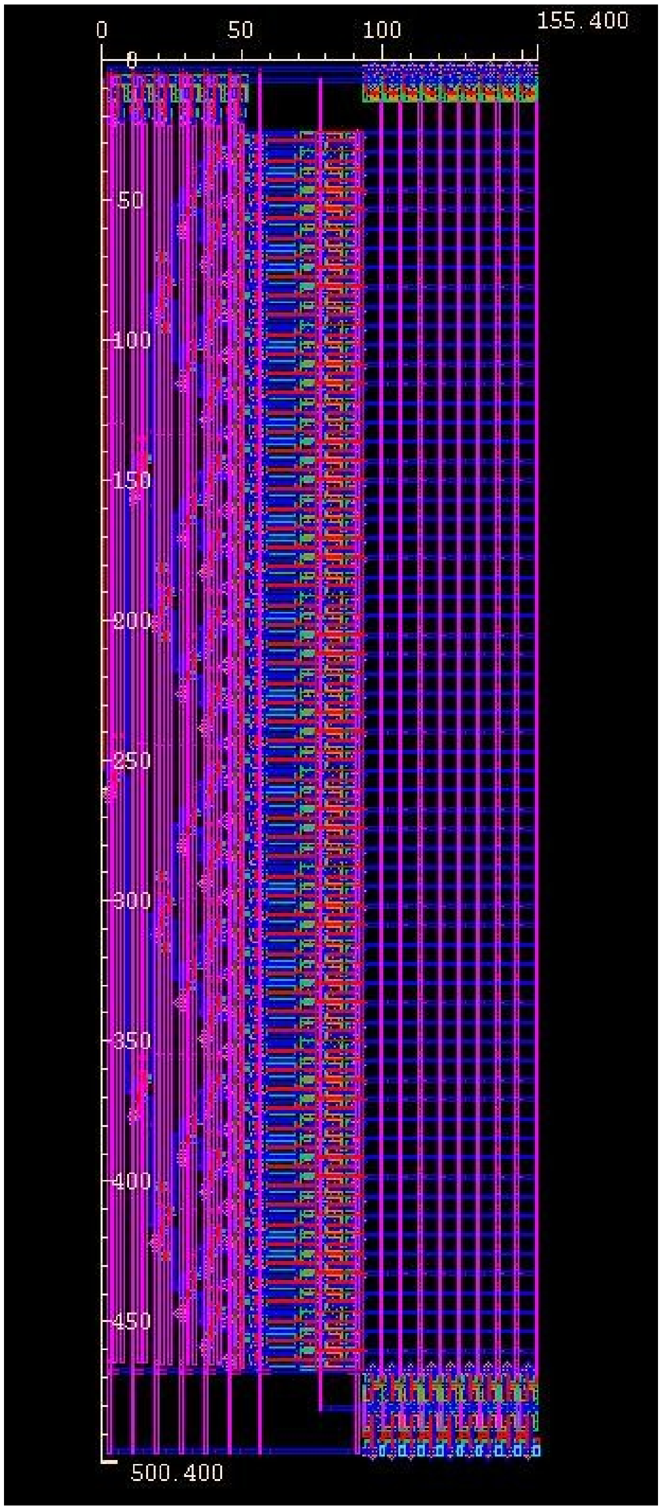


Fig. 6.3 Minimum Sized 64 bit ROM

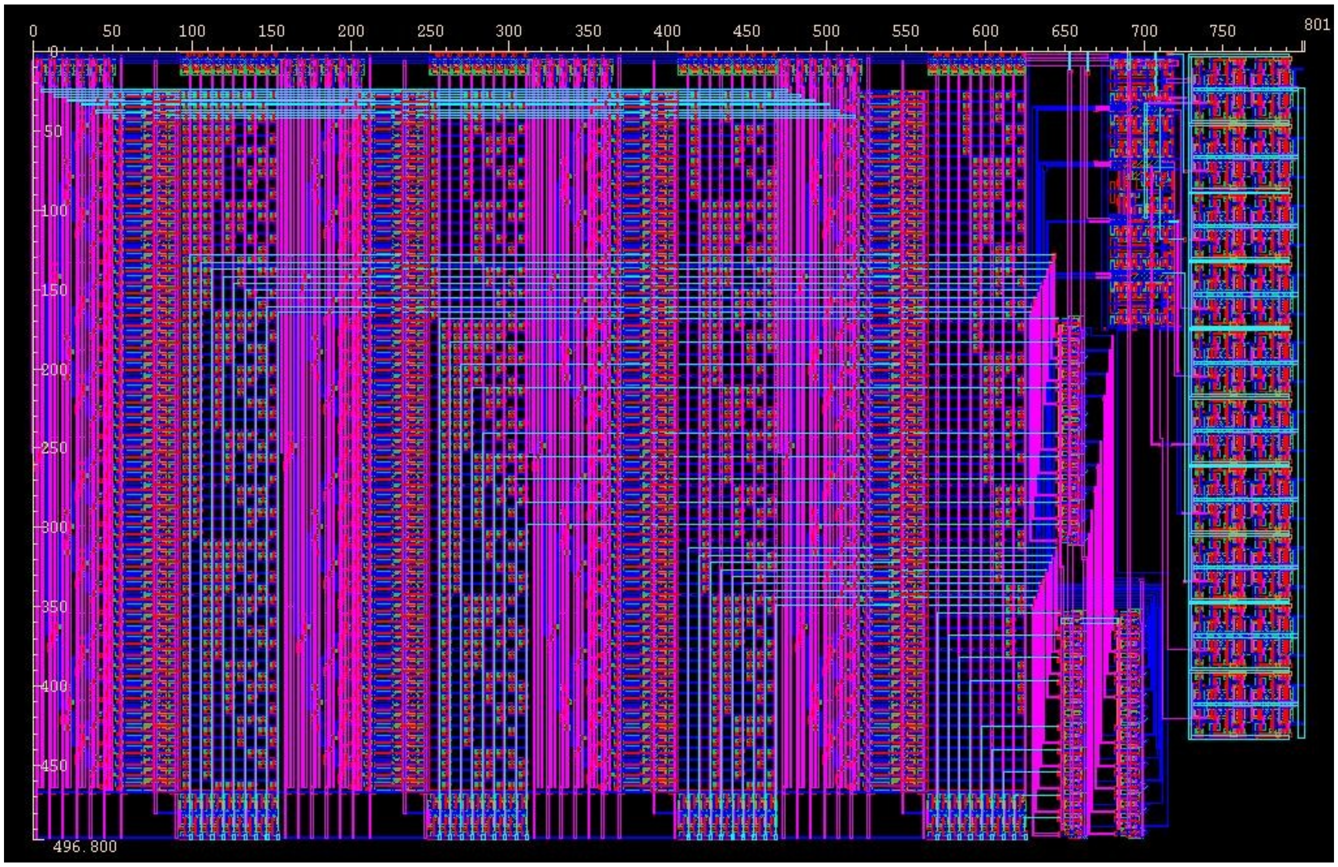


Fig. 6.4 Minimum Sized 256 bit ROM

7 Programming

To make the ROM easier to implement and change, we added some extra symbols to help in the programming. To do this, we grouped three bits together into a larger symbol, creating all of the values from 0 to 7. These look like Figure 7.1.

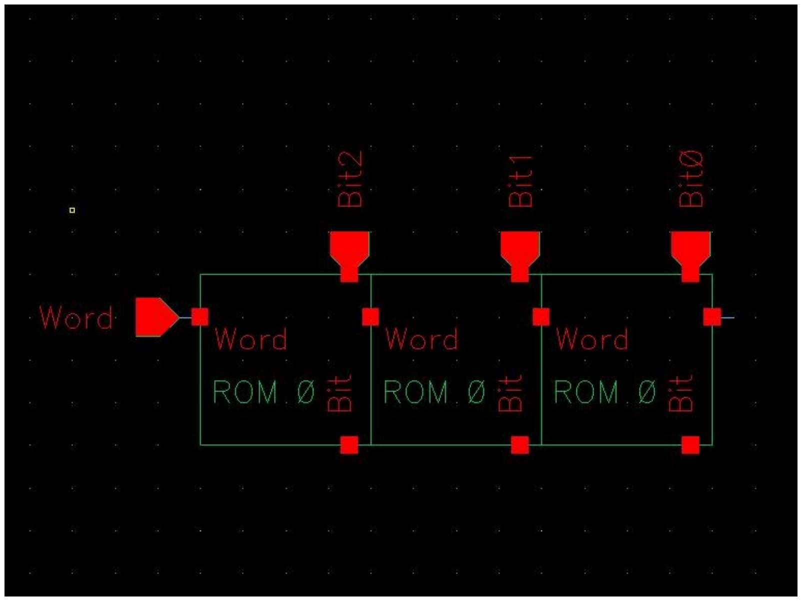


Fig. 7.1 ROM 0

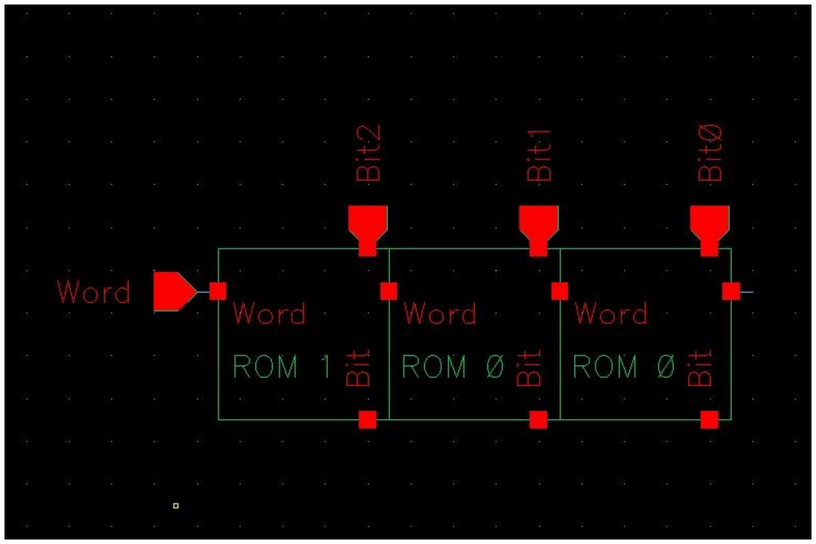


Fig. 7.2 ROM 4

This allows faster programming and quicker modifications. It also enables better troubleshooting as there are fewer cell blocks to identify as wrong. The easiest way to program, using this technique, is to create an array of 64 rows and 3 columns of ROM 0. This will completely fill up the ROM, then allowing you to modify each cell individually to change which value you would like. This method saves a lot of time because there is no longer a requirement to line things up correctly; it is already properly lined up.

8 Testing

Testing was done on every part of the ROM, troubleshooting and checking all components to make sure that it was fully functional. Only three tests will be documented here; for the rest, go to Anusha Ramanujam's "ROM for Direct Digital Synthesizer" report at the following web site:

http://www.eece.maine.edu/research/vlsi/2006/Ramanujam/ECE547_Anusha.pdf

The three tests shown here are testing the Tree Decoder and testing the Sense Amplifier. The Tree Decoder should switch as the inputs increment, thereby turning on one and only one row at a time while systematically increasing. The test is shown in Figure 8.1.

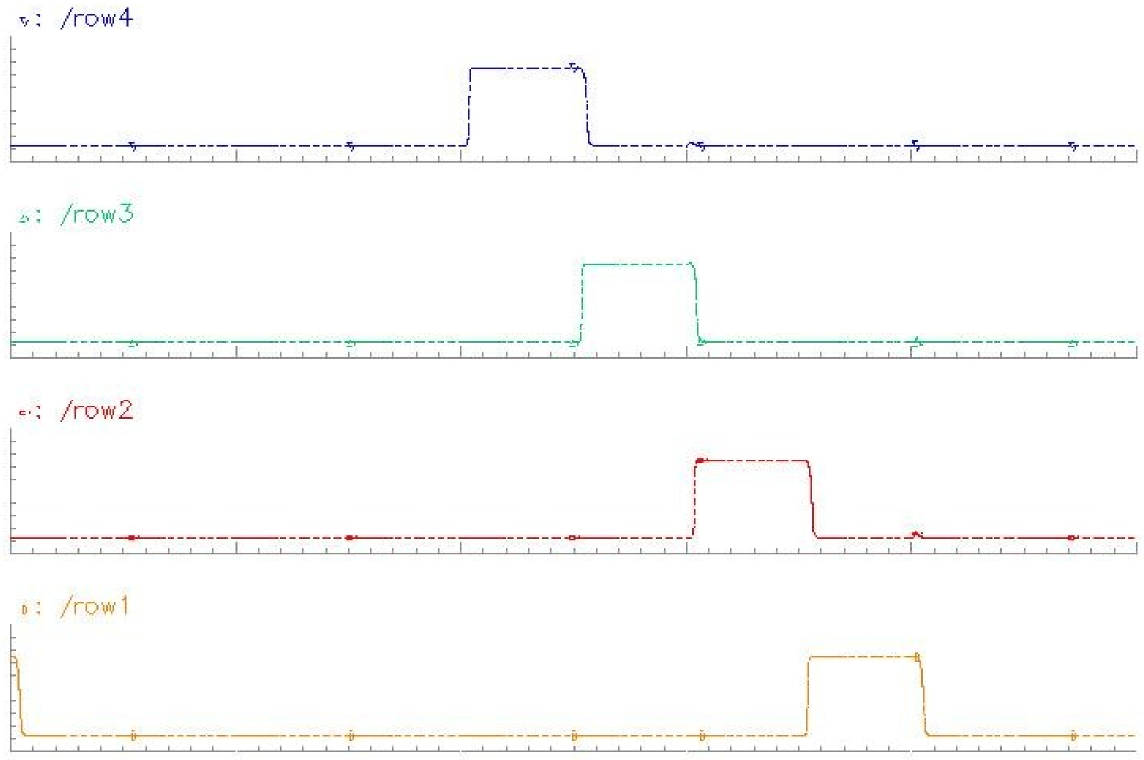


Fig. 8.1 Testing the Tree Decoder

Similarly, the Tree Line itself should work as described; you should see inputs coming in at a lower value than they should, but leaving at the desired value, as in Figure 8.2.

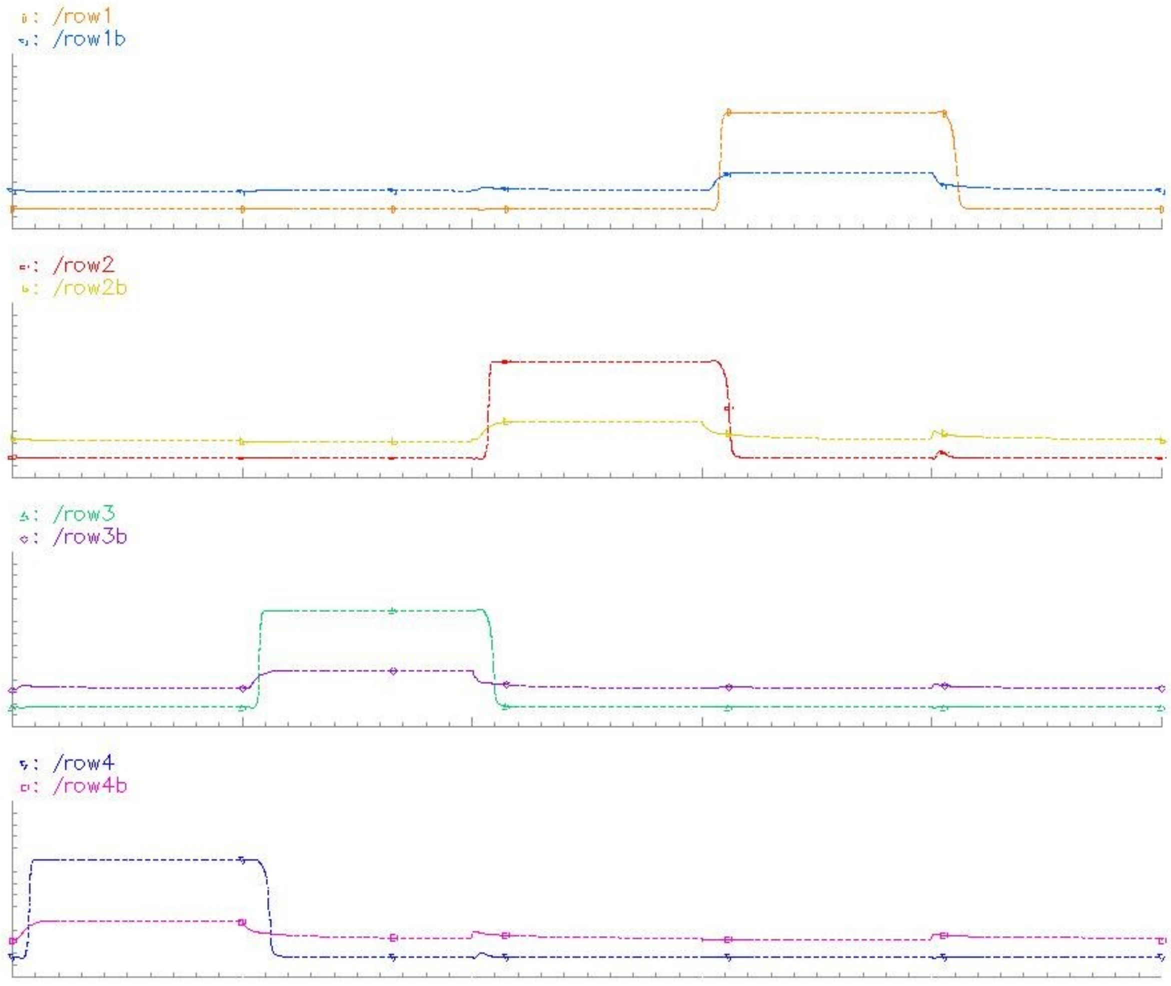


Fig. 8.2 Testing the Tree Line

And finally, to test the Sense Amplifier, the output of the ROM should travel to the Sense Amp and be brought back up to the expected value. This graph is taken from a larger test that was done, but this particular sense amp was the only one that changed, so the others were omitted.

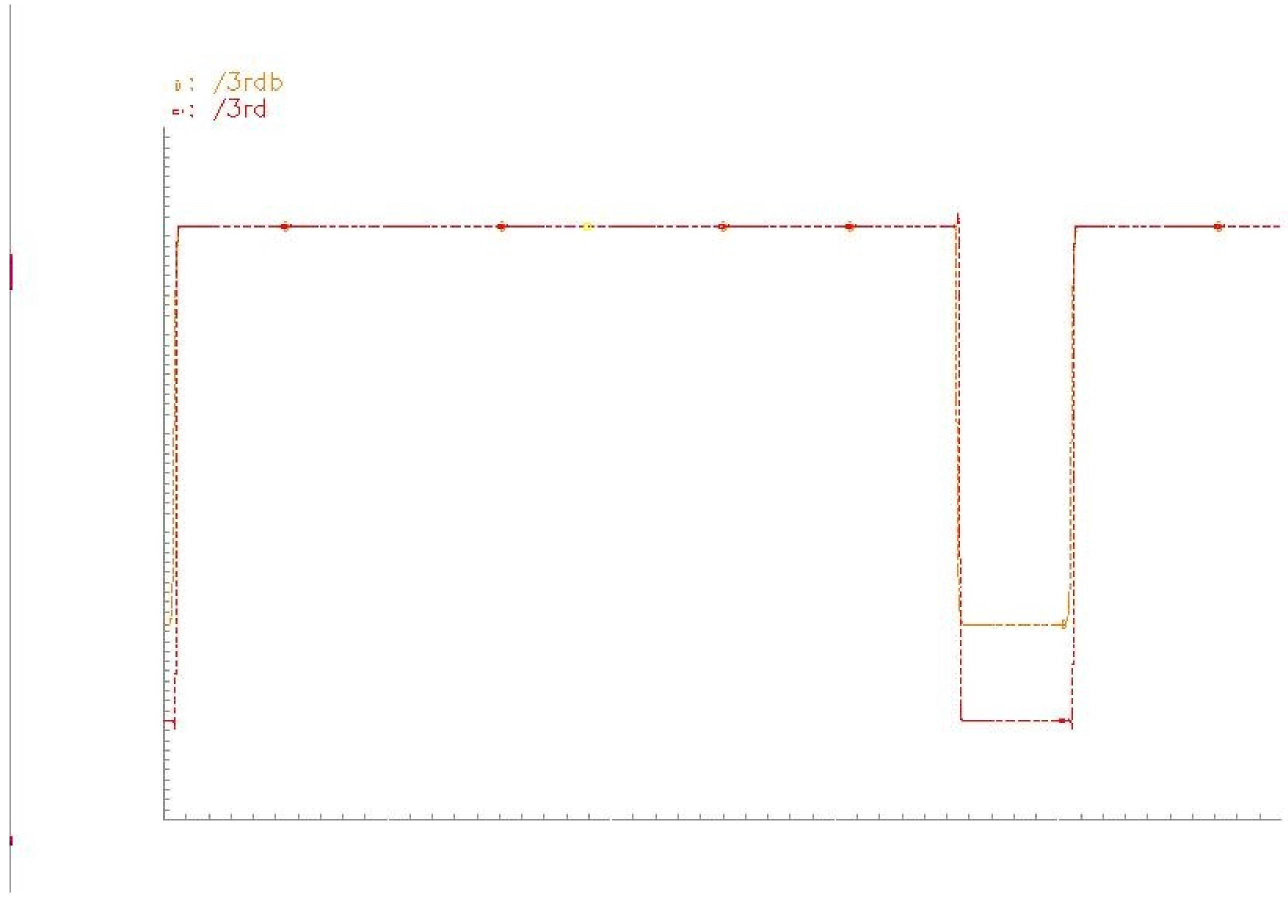


Fig. 8.3 Testing the Sense Amplifer

9 Complete Chip

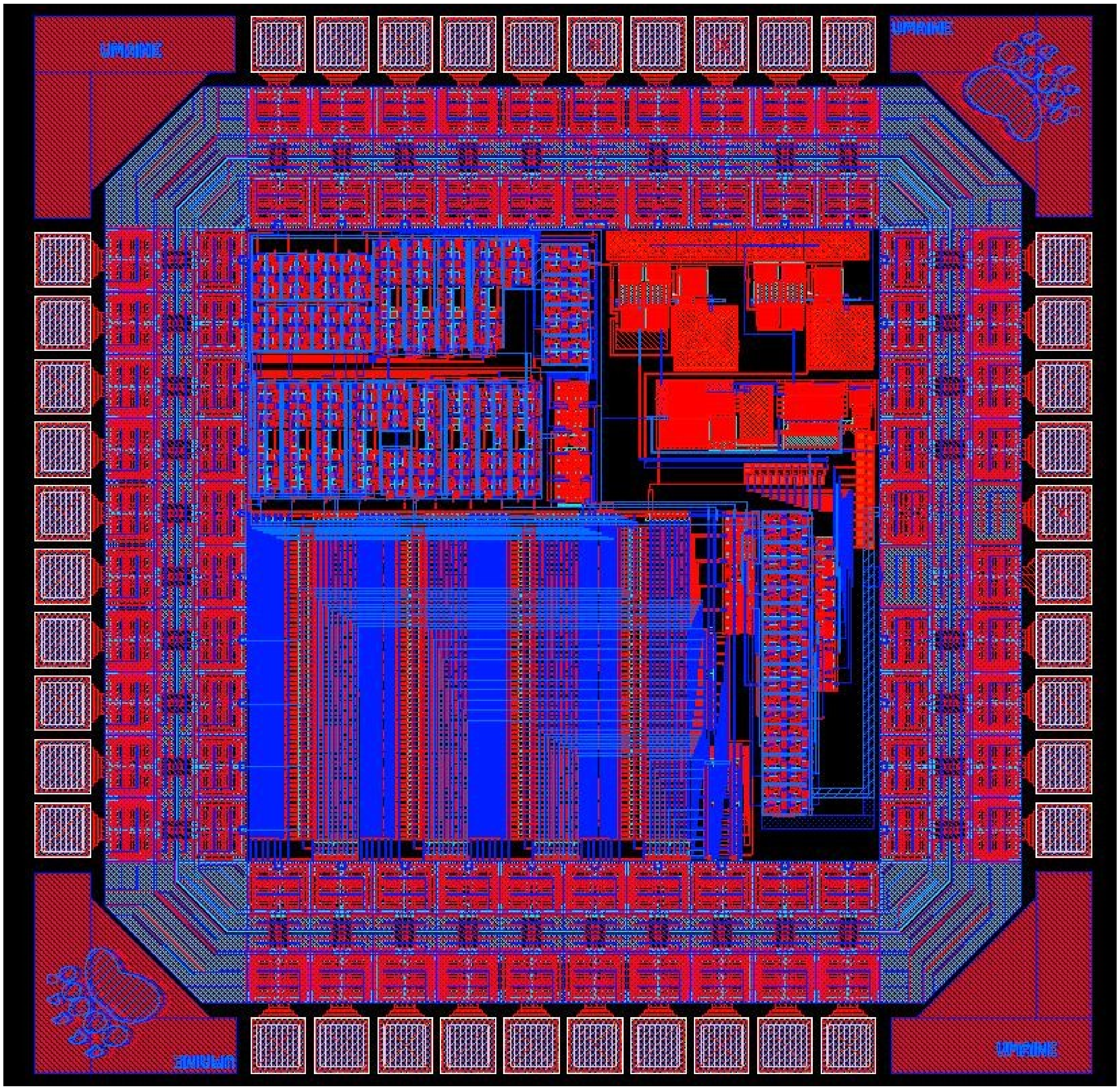


Fig. 9 Complete DDS Chip

PIN	PAD TYPE	NAME	DESCRIBTION
1	Padvdd	vdd	Vdd pin
2	Padio		
3	Padio		
4	Padio		
5	Padio		
6	padio		
7	Padio		
8	Padio		
9	Padio		
10	Padaref	out	Out pin
11	Padio	Clk	Clk pin
12	Padio		
13	Padio		
14	Padio	A4	4th input from ACC
15	Padio	A5	5th input from ACC
16	Padio	A6	6th input from ACC
17	Padio	A7	7th input from ACC
18	Padio	A8	8th input from ACC
19	Padio	A9	9th input from ACC
20	Padio	A10	10th input from ACC
21	Padio	A11	11h input from ACC
22	Padio	A3	3rd input from ACC
23	Padio	A2	2nd input from ACC
24	Padio	A1	1st input from ACC
25	Padio	A0	0 input from ACC
26	Padio		
27	Padio		
28	Padio		
29	Padio		
30	Padio		
31	Padio		
32	Padio		
33	Padio		
34	Padio		
35	Padio		
36	Padio		
37	Padio		
38	Padio		
39	Padio		
40	Padgnd	Gnd	Gnd pin

Fig. 9 Pin Assignments

Appendix

	ROM D	ROM C	ROM B	ROM A
0	000000010	011000101	101101010	111011001
1	000000101	011001000	101101101	111011010
2	000001000	011001011	101101111	111011011
3	000001011	011001110	101110001	111011100
4	000001110	011010001	101110011	111011101
5	000010001	011010011	101110101	111011110
6	000010100	011010110	101110111	111100000
7	000011000	011011001	101111010	111100001
8	000011011	011011100	101111100	111100010
9	000011110	011011111	101111110	111100011
10	000100001	011100010	110000000	111100100
11	000100100	011100100	110000010	111100101
12	000100111	011100111	110000100	111100110
13	000101010	011101010	110000110	111100111
14	000101101	011101101	110001000	111101000
15	000110001	011101111	110001010	111101001
16	000110100	011110010	110001100	111101001
17	000110111	011110101	110001110	111101010
18	000111010	011111000	110010000	111101011
19	000111101	011111011	110010010	111101100
20	001000000	011111101	110010100	111101101
21	001000011	100000000	110010110	111101110
22	001000110	100000011	110011000	111101111
23	001001001	100000101	110011010	111101111
24	001001101	100001000	110011011	111110000
25	001010000	100001011	110011101	111110001
26	001010011	100001101	110011111	111110010
27	001010110	100010000	110100001	111110010
28	001011001	100010011	110100011	111110011
29	001011100	100010101	110100100	111110100
30	001011111	100011000	110100110	111110100
31	001100010	100011011	110101000	111110101
32	001100101	100011101	110101010	111110101
33	001101000	100100000	110101011	111110110
34	001101011	100100010	110101101	111110111
35	001101110	100100101	110101111	111110111
36	001110001	100101000	110110001	111111000
37	001110101	100101010	110110010	111111000
38	001111000	100101101	110110100	111111001
39	001111011	100101111	110110101	111111001

40	001111110	100110010	110110111	111111010
41	010000001	100110100	110111001	111111010
42	010000100	100110111	110111010	111111011
43	010000111	100111001	110111100	111111011
44	010001010	100111100	110111101	111111011
45	010001101	100111110	110111111	111111100
46	010010000	101000001	111000000	111111100
47	010010011	101000011	111000010	111111100
48	010010110	101000101	111000011	111111101
49	010011001	101001000	111000101	111111101
50	010011100	101001010	111000110	111111101
51	010011111	101001101	111001000	111111101
52	010100010	101001111	111001001	111111110
53	010100101	101010001	111001011	111111110
54	010101000	101010100	111001100	111111110
55	010101011	101010110	111001101	111111110
56	010101110	101011000	111001111	111111110
57	010110001	101011011	111010000	111111111
58	010110100	101011101	111010001	111111111
59	010110110	101011111	111010011	111111111
60	010111001	101100001	111010100	111111111
61	010111100	101100100	111010101	111111111
62	010111111	101100110	111010110	111111111
63	011000010	101101000	111011000	111111111