University Of Pennsylvania

School of Engineering and Applied Science

Department of Electrical and Systems Engineering

ESE 3700 Spring 2025

Circuit-Level Modeling, Design, And Optimization For Digital Systems

---

Project 2: Memory Design

Mohamed Elsheshtawy 17363793

Contents

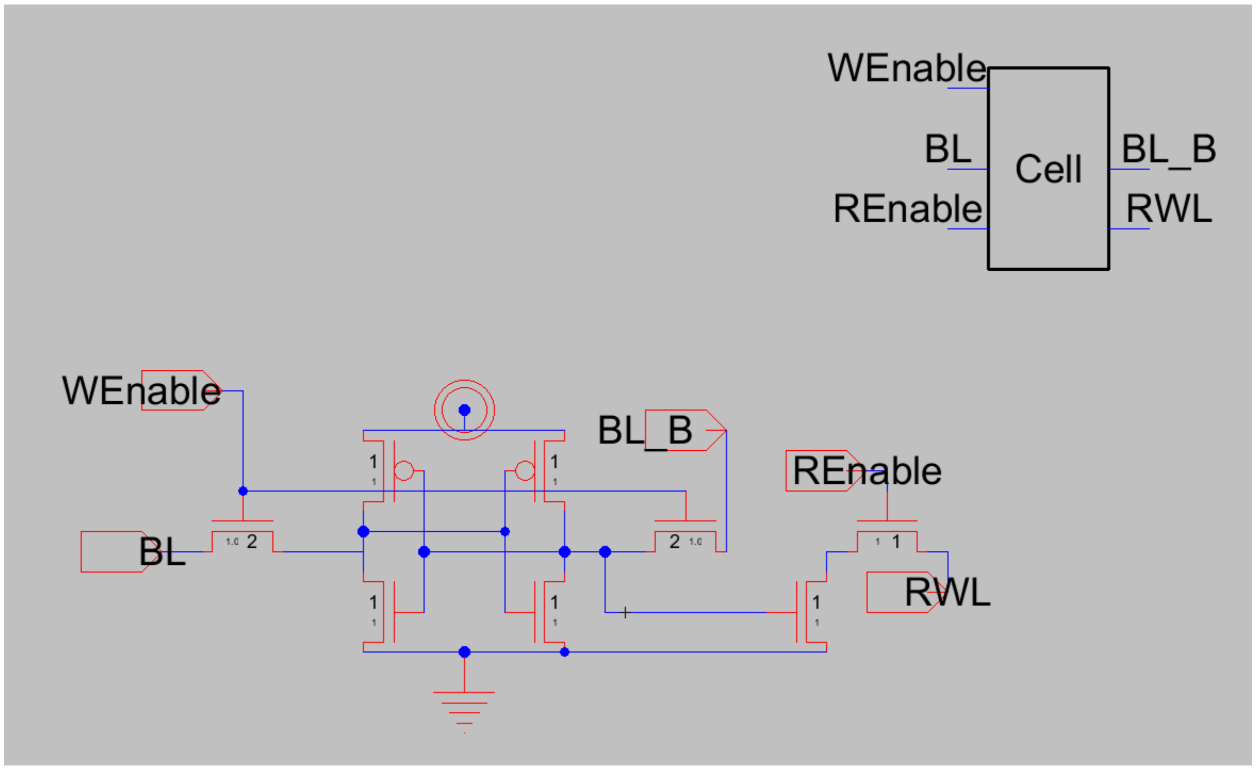
1- Design procedures

In order to make any design, we need to treat as a function. To address a function we need to know the functionality, inputs and outputs. This is the high level abstraction. Then we need to divide the design into modules(blocks) and define their relationship with each other.

The function is to store data in a word format (4bits) and be able to retrieve it. The size of the memory should be 16 words and can be address by an address bus of 4bits size.

To create this I needed Decoders, Clock, Memory(Memory-> Words-> cells), Buffers, Inverters, 2-input AND gates and 4-input OR.

1.1- Cells

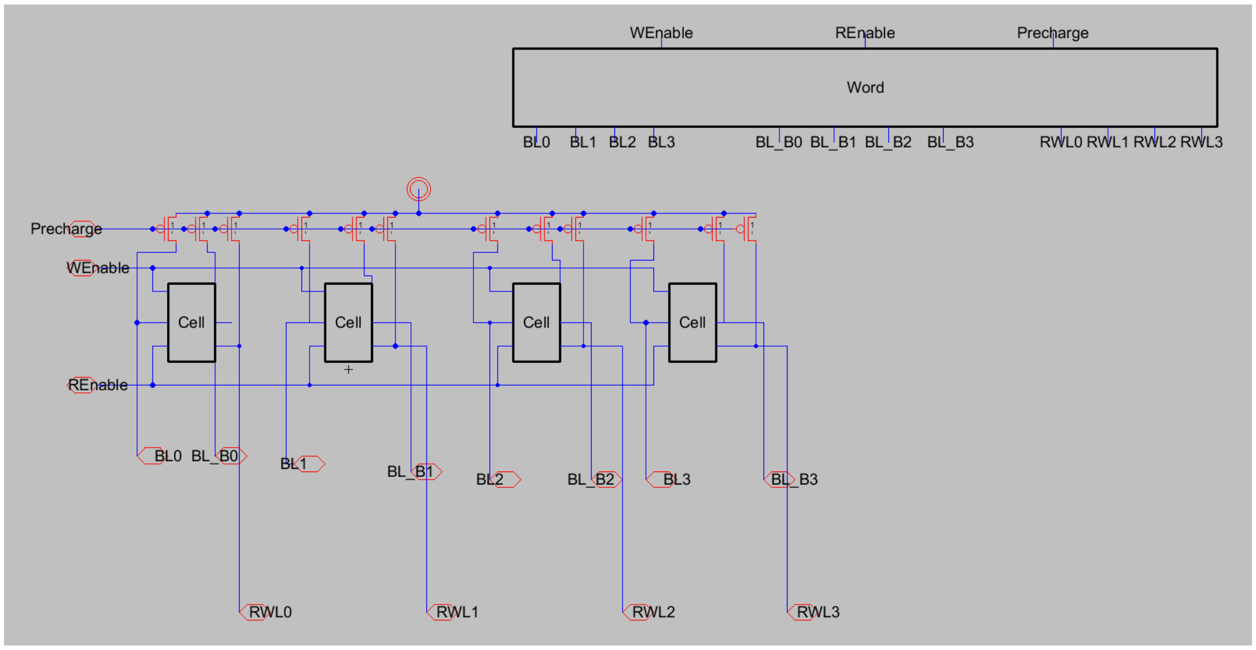


I used 8T SRAM cells to mitigate the need for sensing amps and also the strict sizing requirements for reading and writing. With this topology, reading is fast and more robust and don’t introduce any noise or errors in my data. This is because the reading route is decoupled from the writing route.

As we can see the size of the cell is 10(width). If I used the 6T it would have been more because I would have sized the NMOS inside the cell at width of 4 at least so that reading don't affect my cell. Another thing is that this is way either to interface with since the reading line is different from the writing line.

The only downside it this will use a little more power and also I will need to precharge the RWL(Read line).

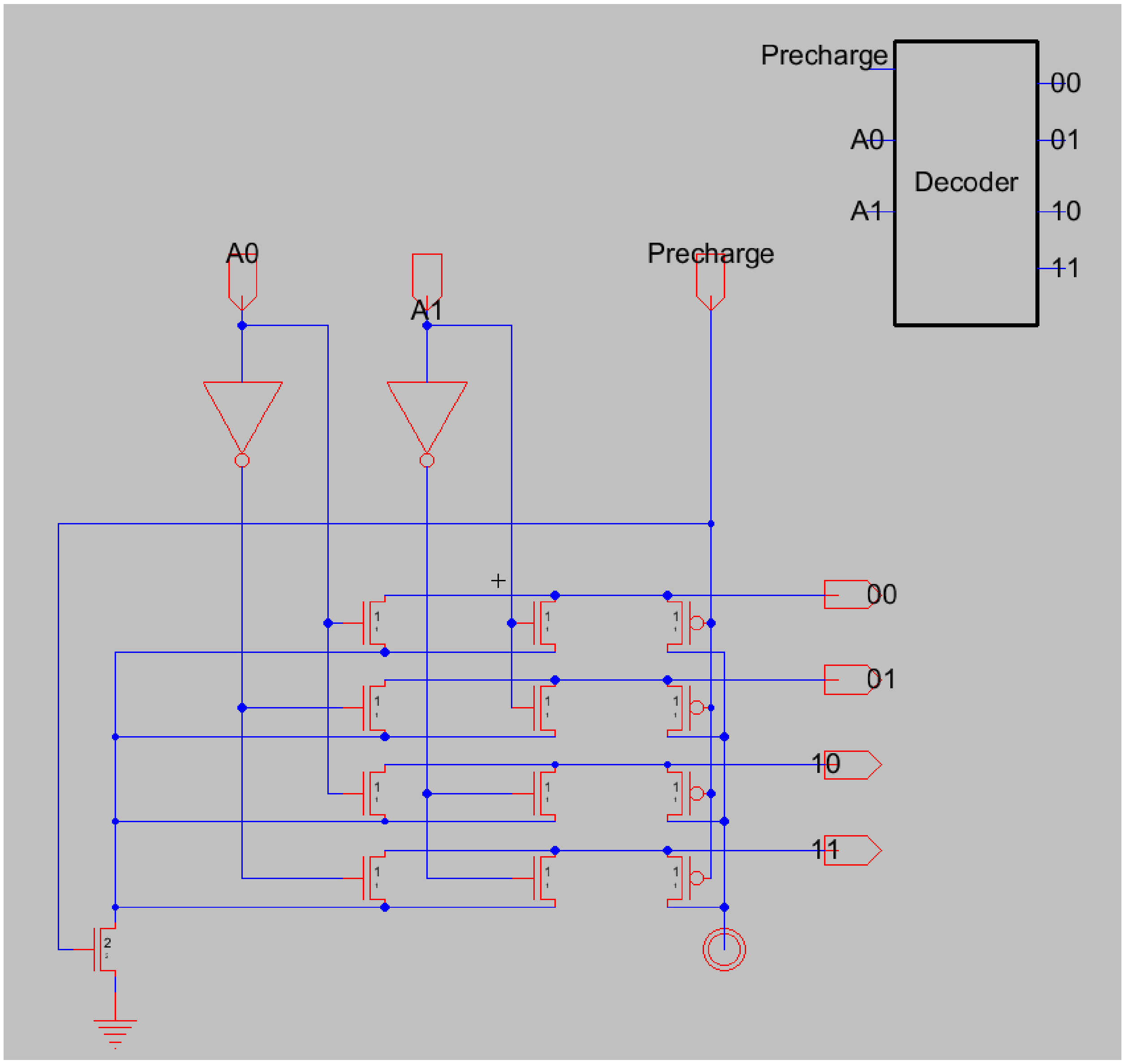
1.2- Word Design



The word is just 4 cells in parallel as. Also the write/Read enable works simultaneously for all 4 cells since we will be reading and writing word at a time. The RWL(Read Line), BL(bit line), BL_B(Bit line bar) are all connected to precharging transistors. Also we have access to them to be able to, of course, read and write.

The size of the word is 12(precharging transistors) + 4*10(Cell size) = 52

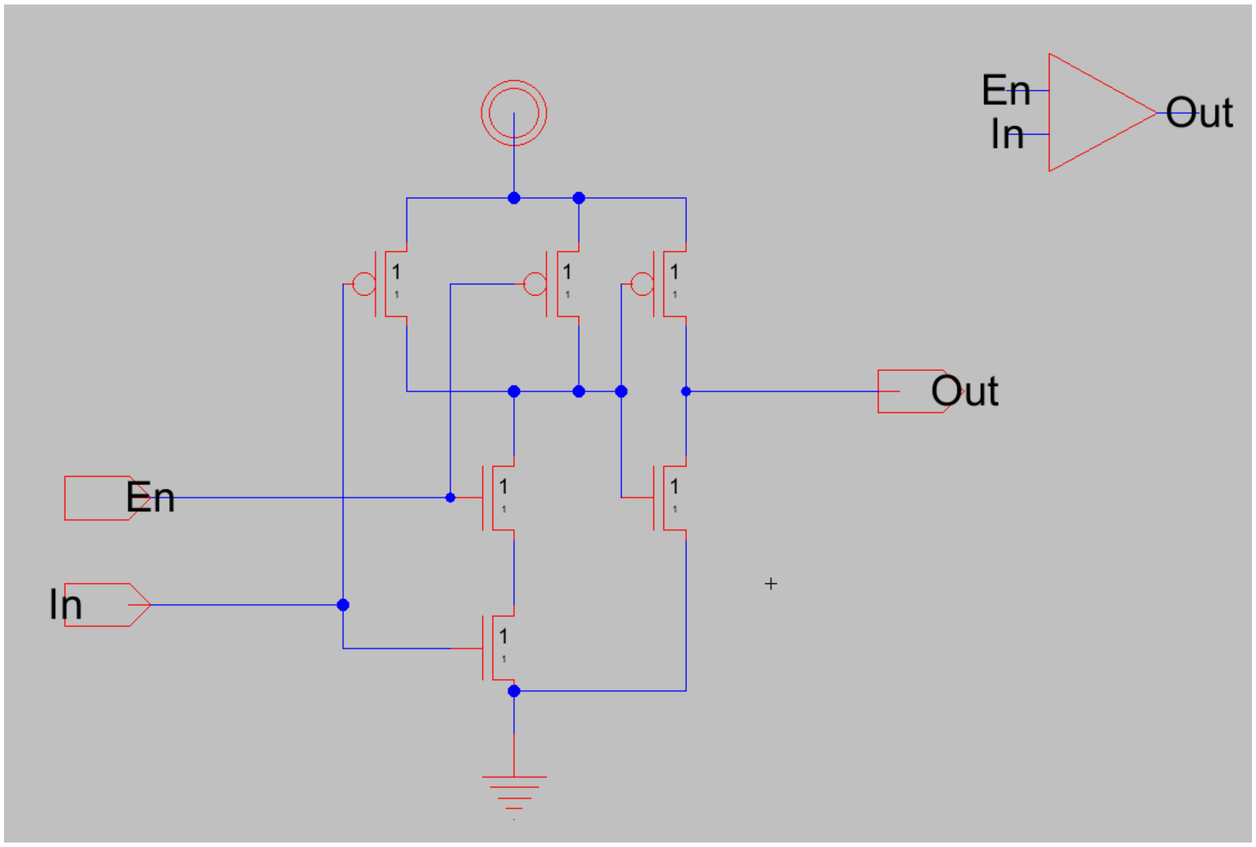
1.3- Decoder Design

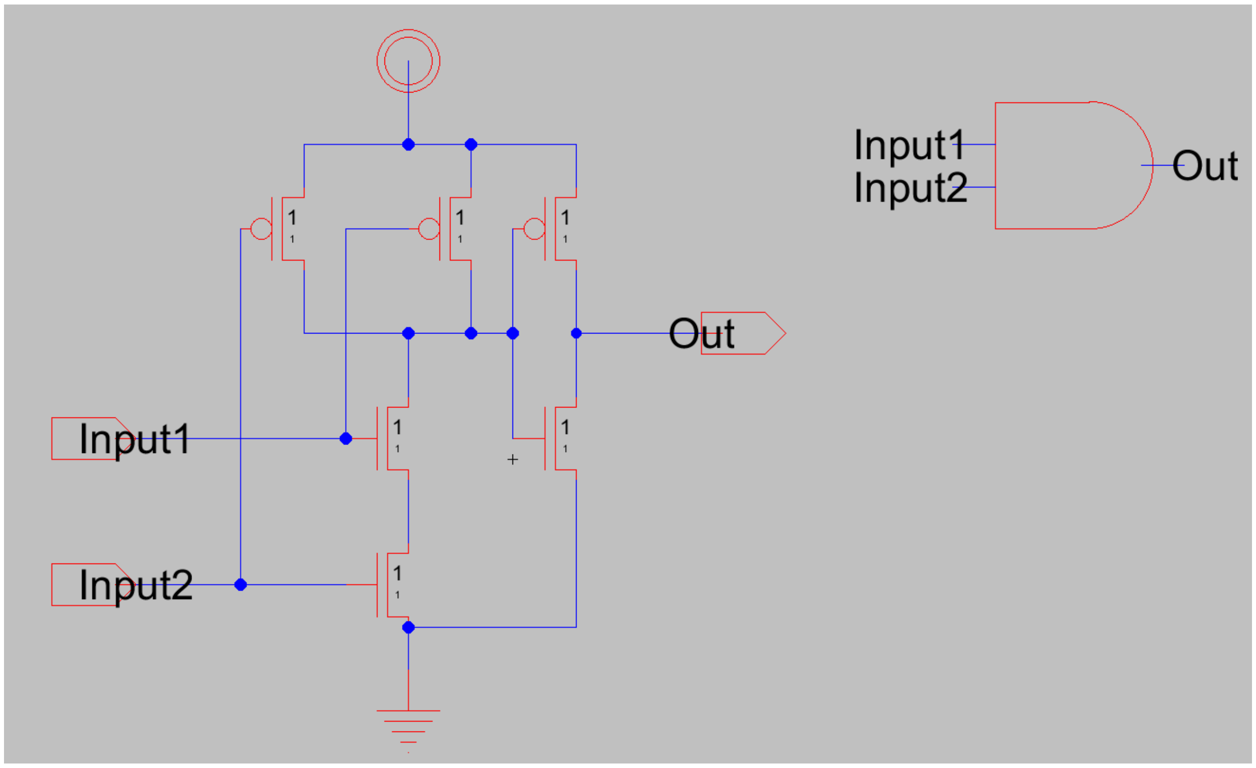


I made a decoder using domino logic where I precharge lines then evaluate. This design uses small space but its problem is that it needs buffers at the output. However the good thing is that the decoder where going to drive a big load, so the use of buffers was inevitable.

The size of the decoder is 2*2(not gate)+ 14 = 18

1.4- Enable buffer, AND gate

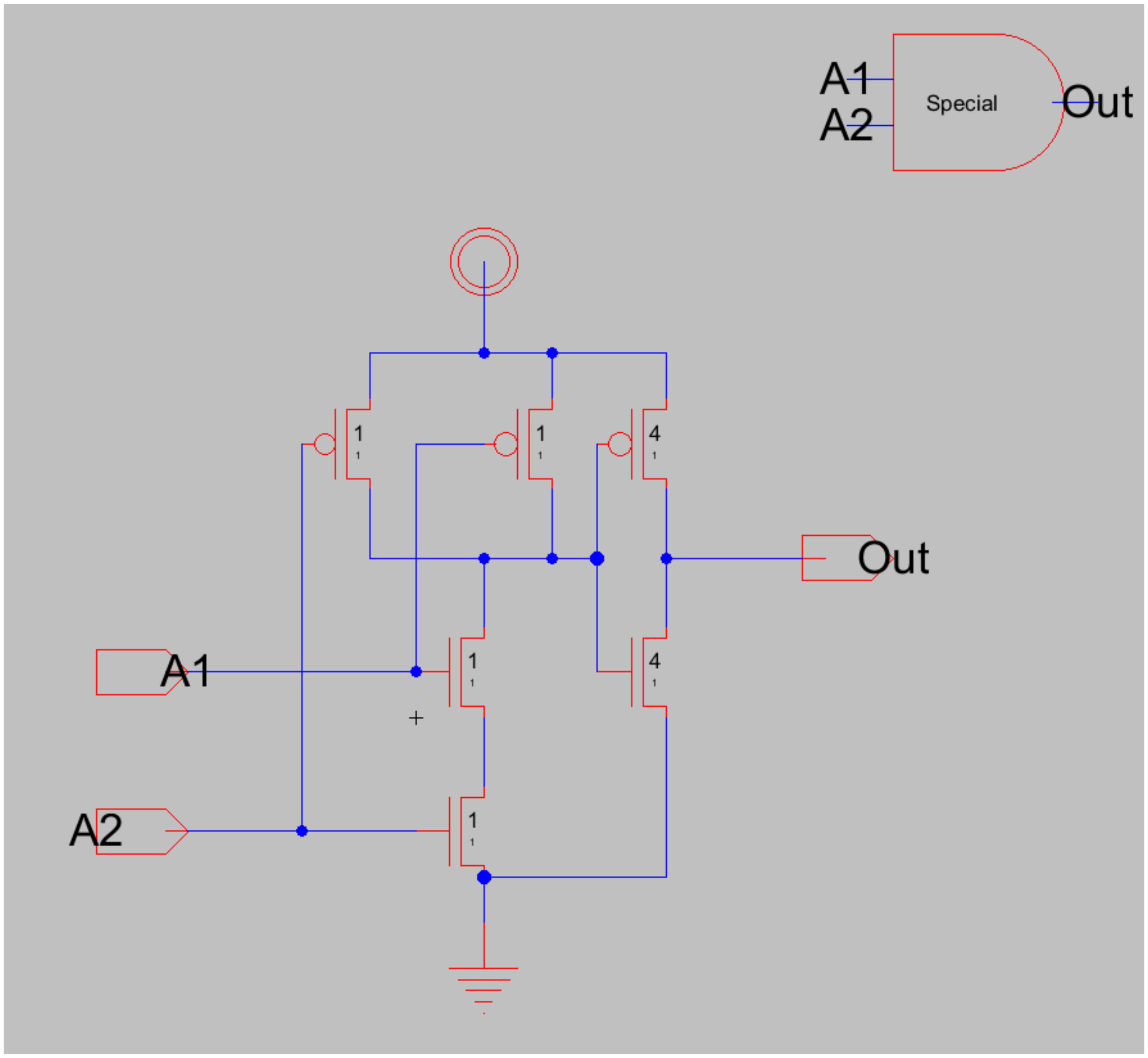




I used AND in the buffer design because it used the least amount of size while doing the work. Any other type that uses less transistors needed to be buffered because it was pass transistor logic.
Unfortunately I realized this after making the design and I used AND gates and enable buffers in the design but they are the same thing and same size.

The size of this is 6.

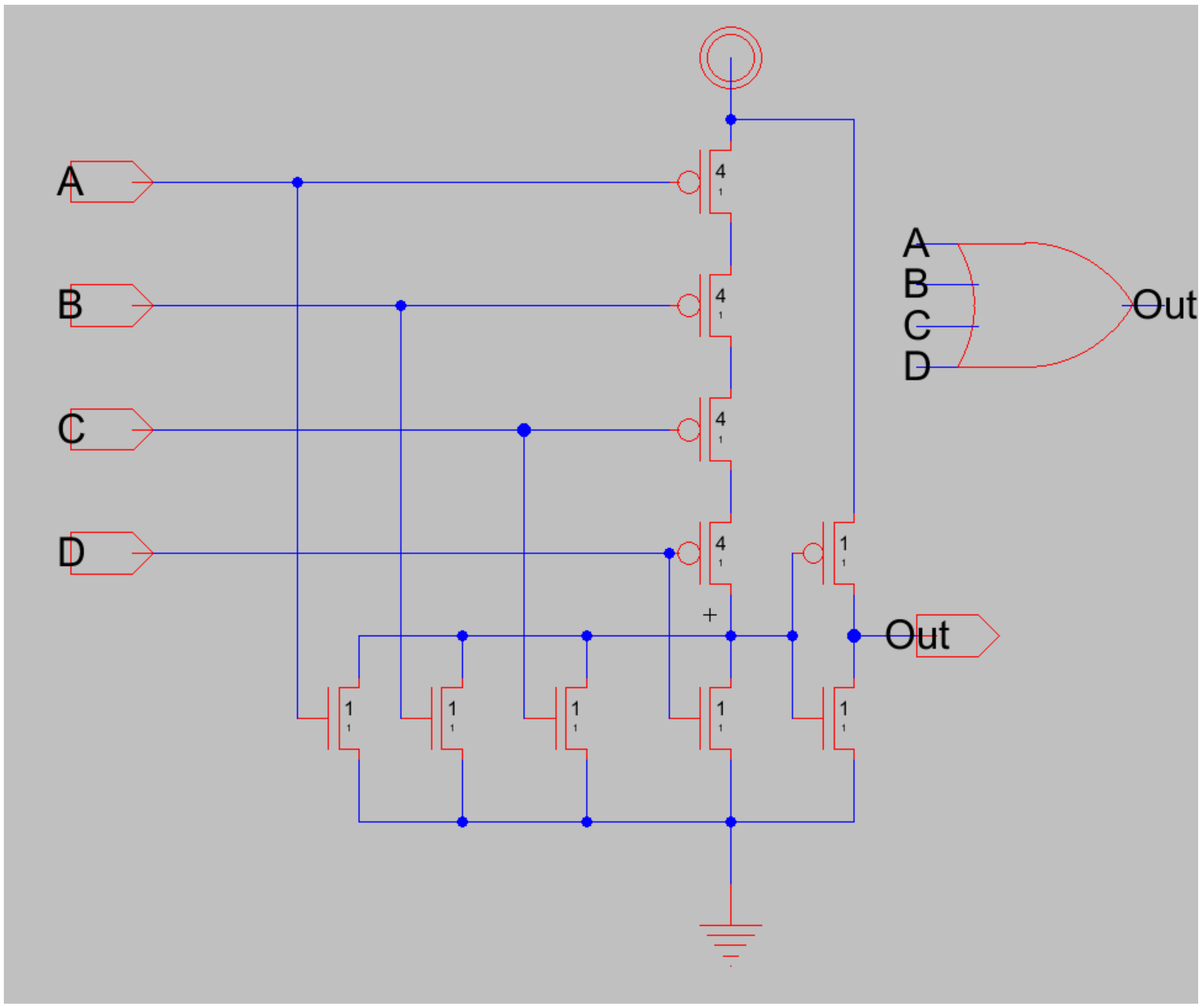
1.5- High load AND



This special AND gate was created to drive high loads as it will cut the load size by ¼ because of its low driving resistance.

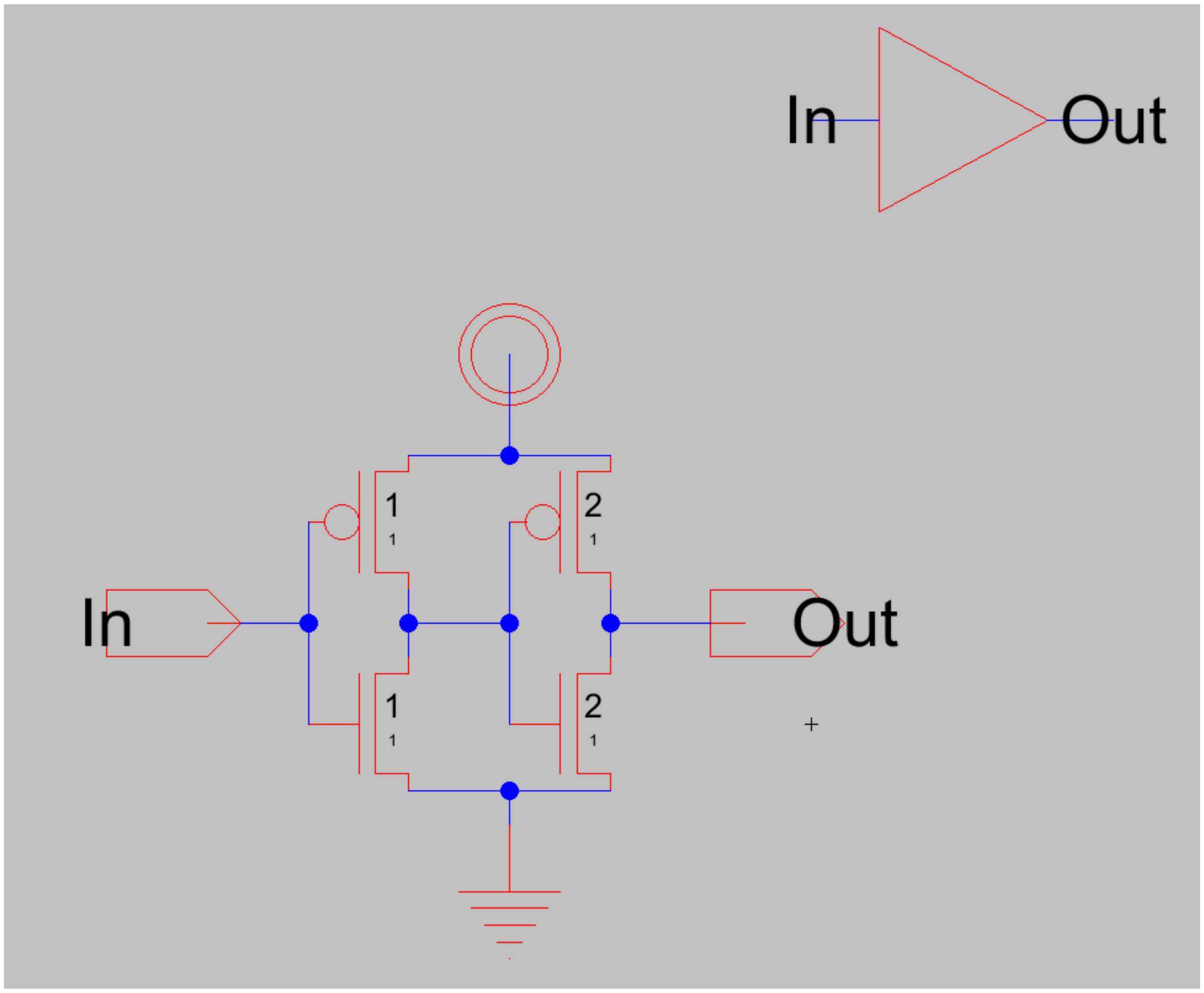
Size = 12

1.6- 4 input OR



Size = 22

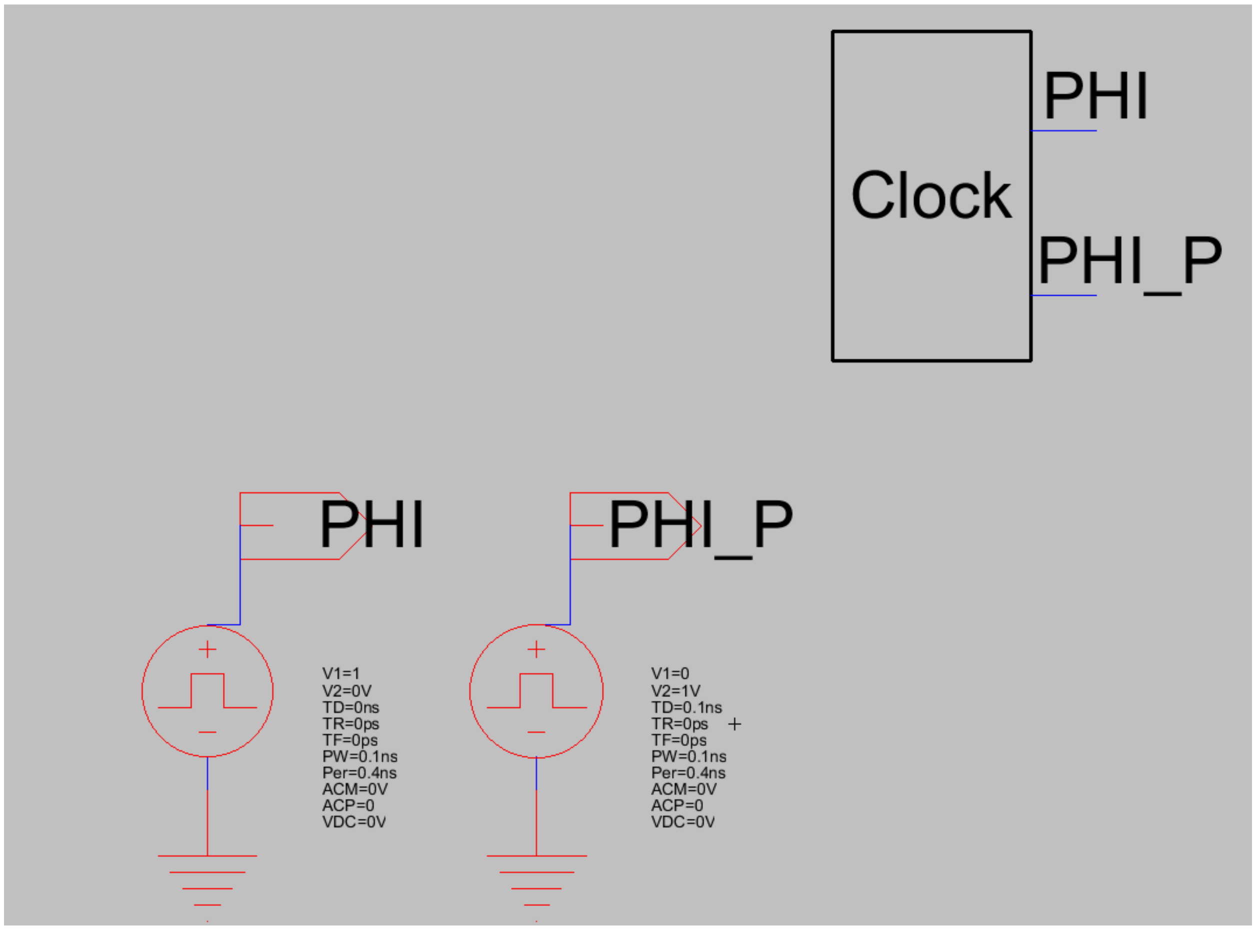
1.7- Buffers



This is for driving high loads.

Size = 6

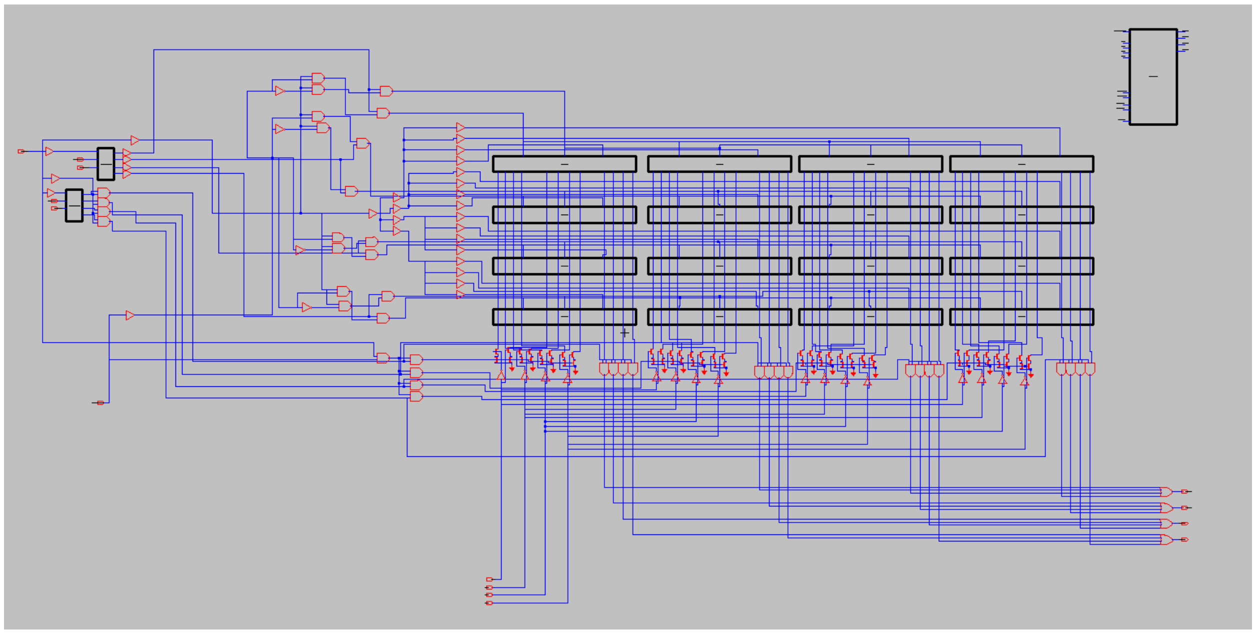
1.8- Clock



This is clock PHI and the other one is just for sophisticated uses like putting a latch at the end of the circuit to save data. I will used the latches at very high speeds to verify the design only.

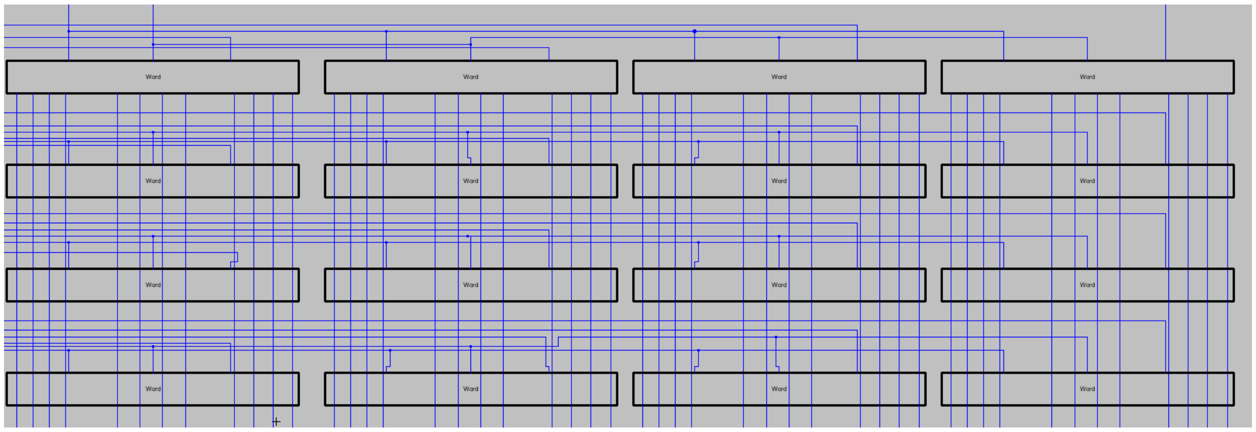
2- SRAM Design

Overall design is as follows:



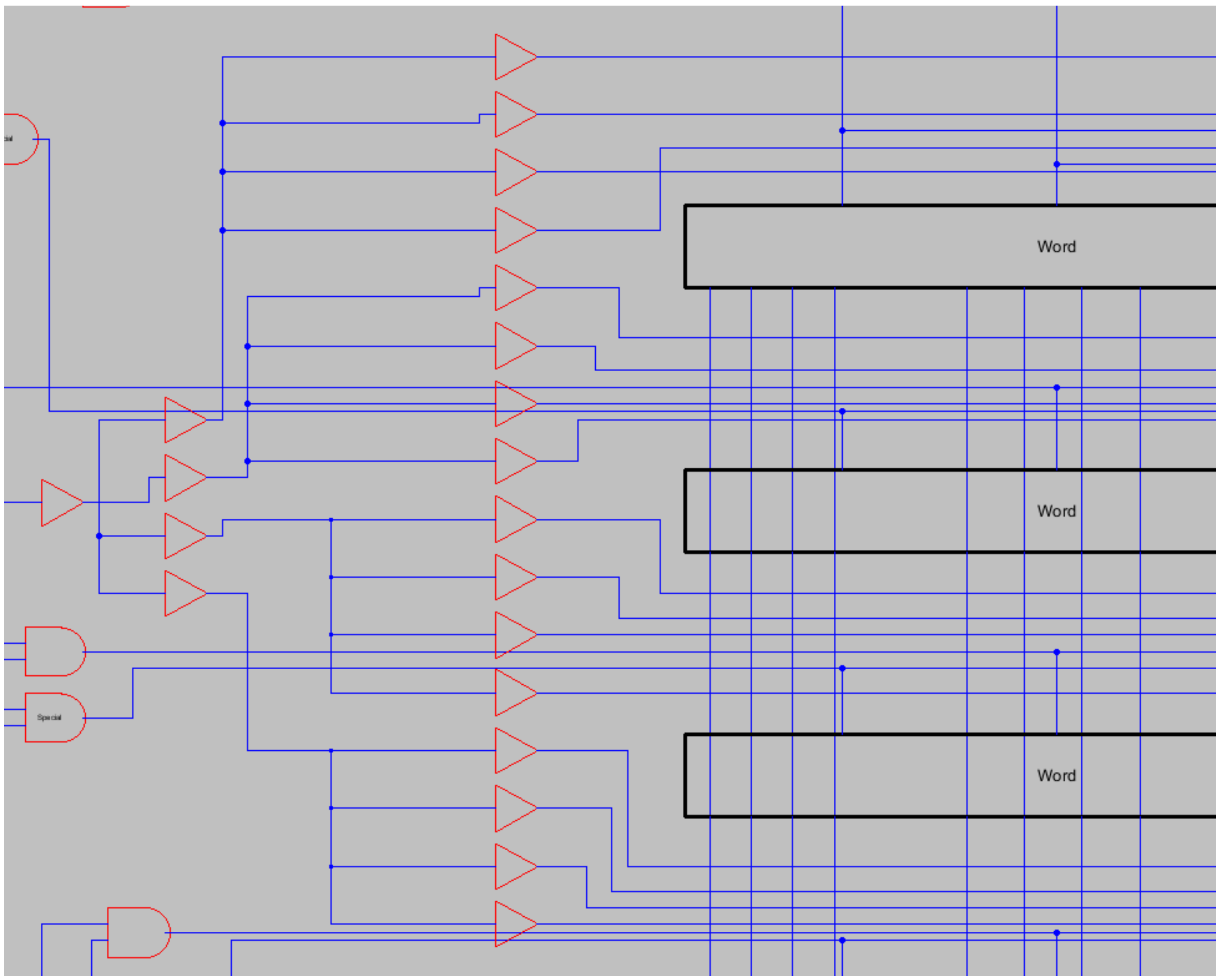
Of course, this is so complicated so I will tackle it part by part.

2.1- Memory



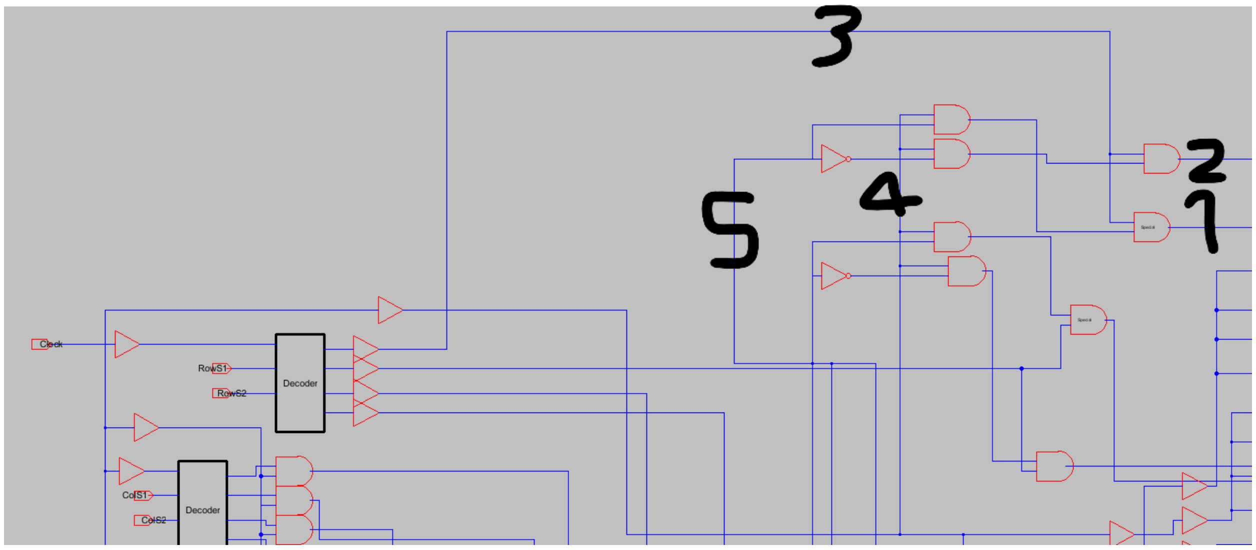
First part is the connection between words to each other. The words BL, BL_B, RWL are all connected together vertically. This is valid because only one row will be ON at a time due to the wordline enable and readline enable pins.

2.2- Precharging pins



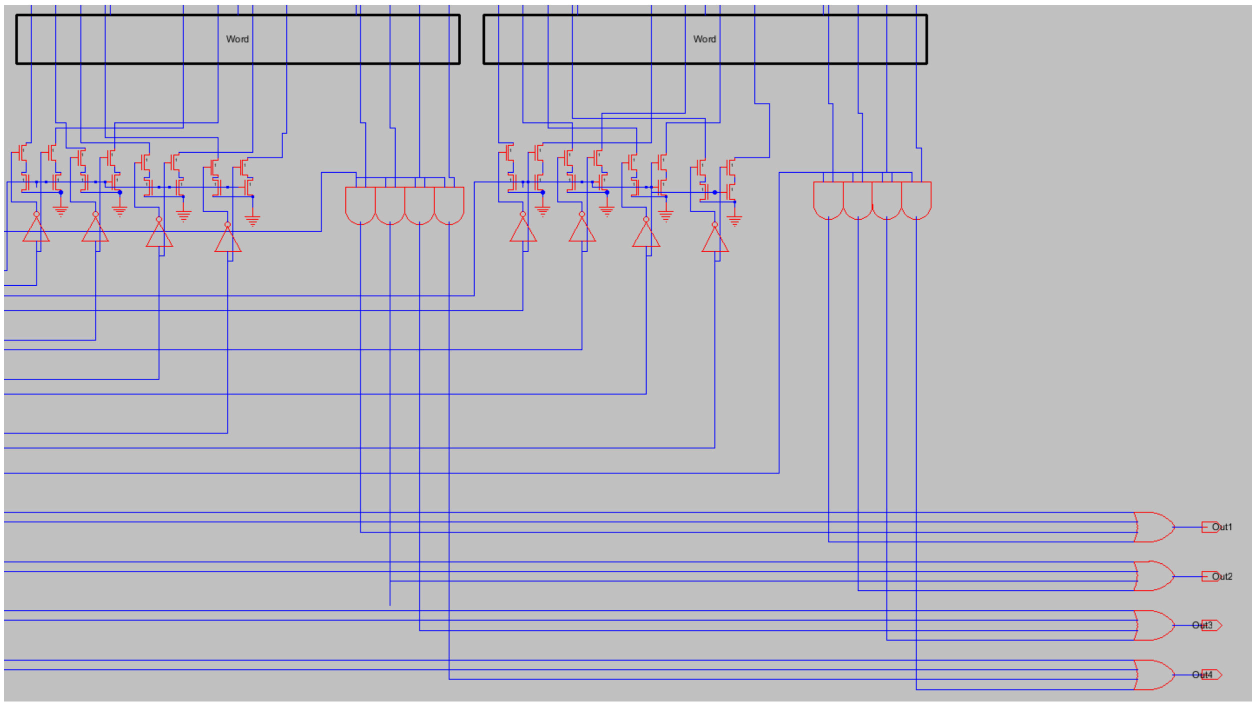
Here the first buffer is connected to the clock. I did a clock tree like this to reduce the delay. There is 16 buffers at the last stage each one driving the precharge pin of each word

2.3- Word bitline and read bitline



Line 1 is connected Word Enable of first row. Line 2 is connected to Read Enable of first row. Line 3 is the decoder and it is connected to last stage so that if it is zero then the first row doesn't work at all. Line 4 is clock is connected to the second row such that it prevents word and read bitline from working while we are precharging the lines. Line 5 is Read/Write input it works such that if 1 then write if 0 then read.

2.4- RWL lines.

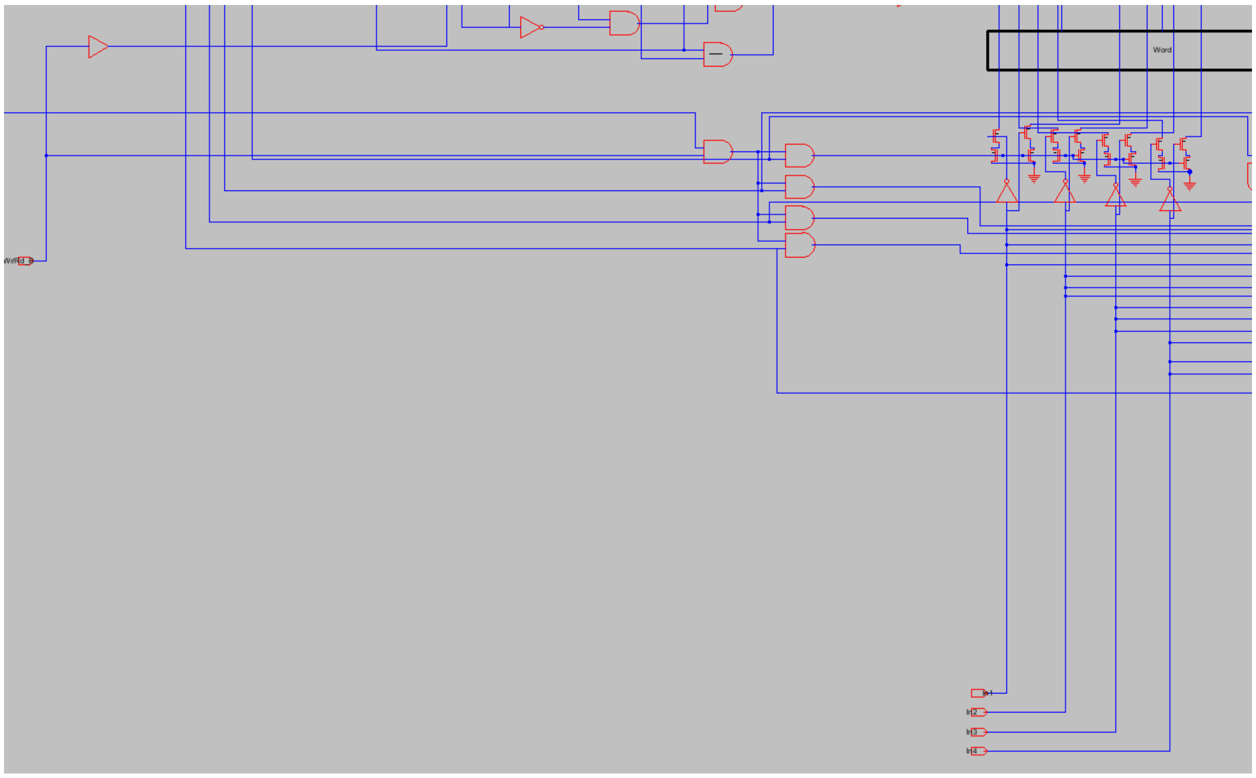


As we can see the RWL are ANDed with other signal (Which is the second decoder signal-column decoder), a signal that turns a column on or off such that if it is off then it will output zero and if it is ON, then the AND works as a buffer. After that RWL1(same case for RWL2,3,4) of each column of words are feed into one OR to detect if there is a one signal or not if not then It will output zero. This works before if a column isn't working it will always output zero and if it is working it might output zero or 1 and the OR will reflect this.

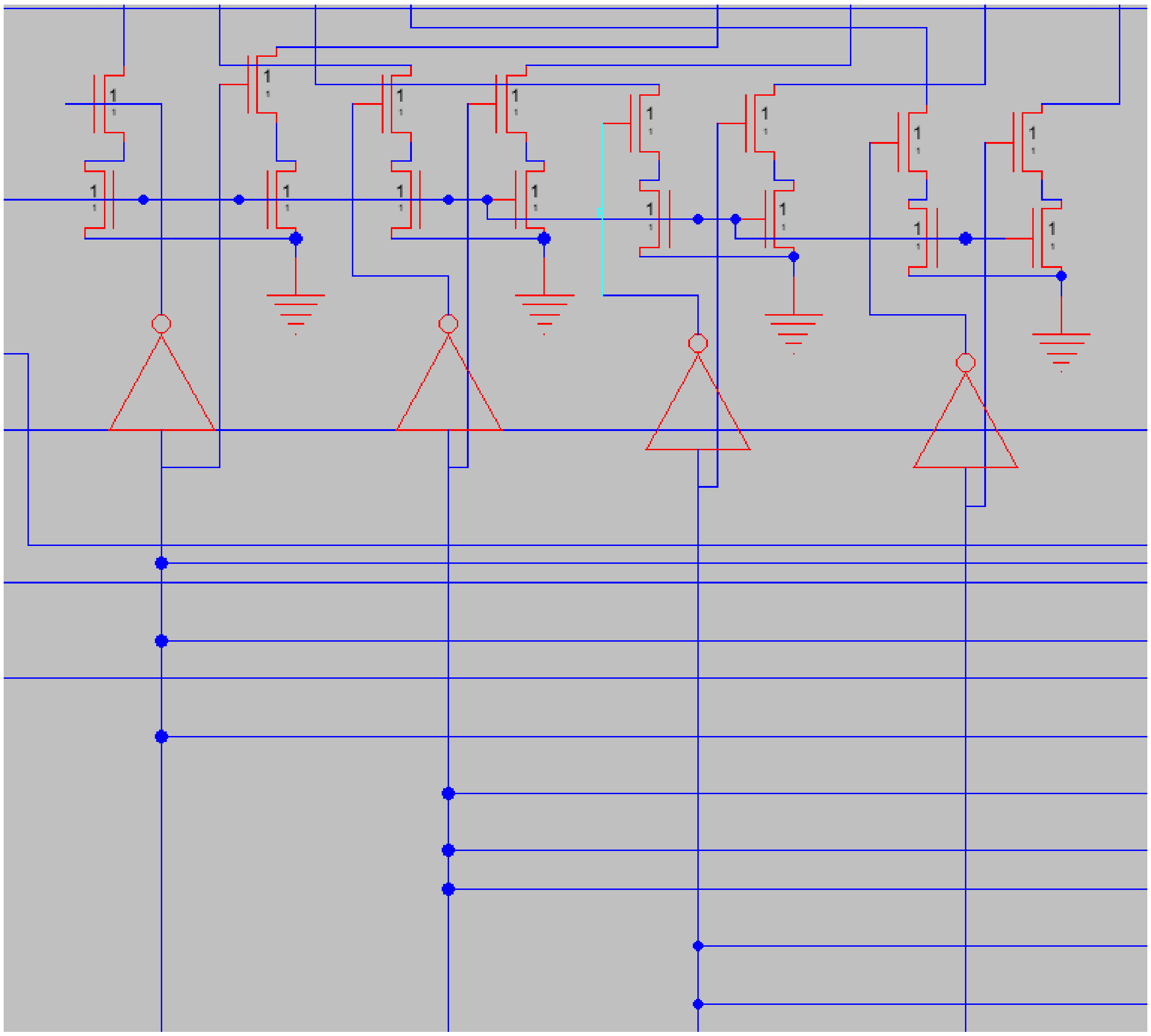
2.5- BL and BL_B

This is bit complicated so I will divide it into two.

2.5.1- Inputs



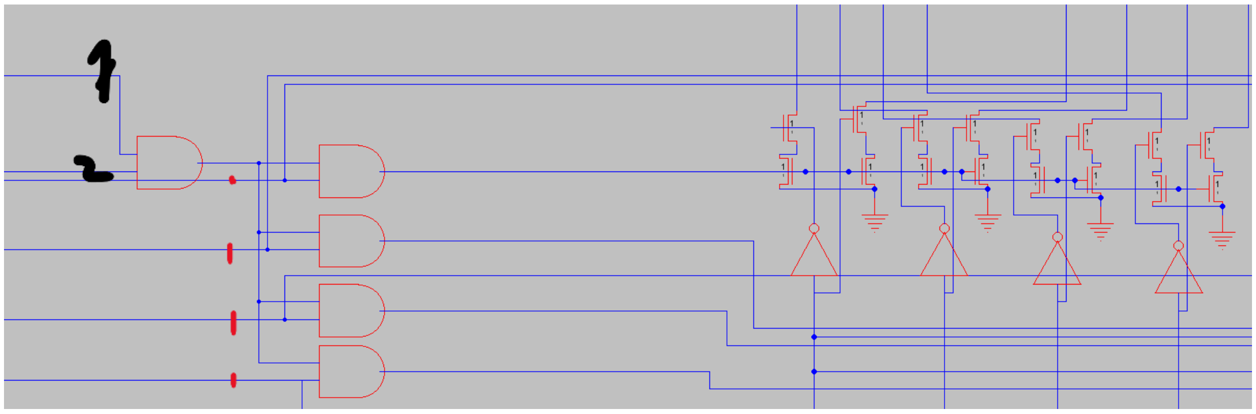
This is the overall architecture. We 5 main signals. Four signals are the inputs which are located down to the right. And the other signal is the signals ANDed with each other and connected to NMOSs the has there source on ground.



Each input is connected to a NOT gate and a normal line such that it turns one NMOS on and the other off so that we write on the SRAM. But as we can see each input line has 3 nodes attached to it. Those nodes are feeding all the other 3 columns of words. I didn't use the decoder here or anything because there is evaluation NMOS just above the inverters. There gate are the ones that decide if we are going to overwrite this column or not.

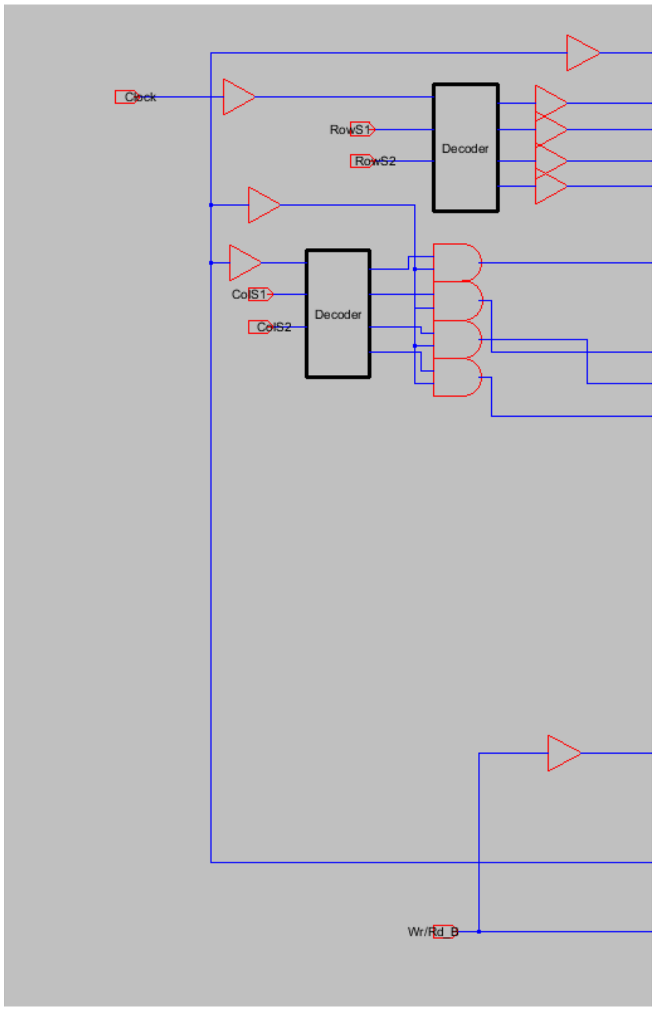
This takes us to the other signal

2.5.2- Decoder, clock and write signals



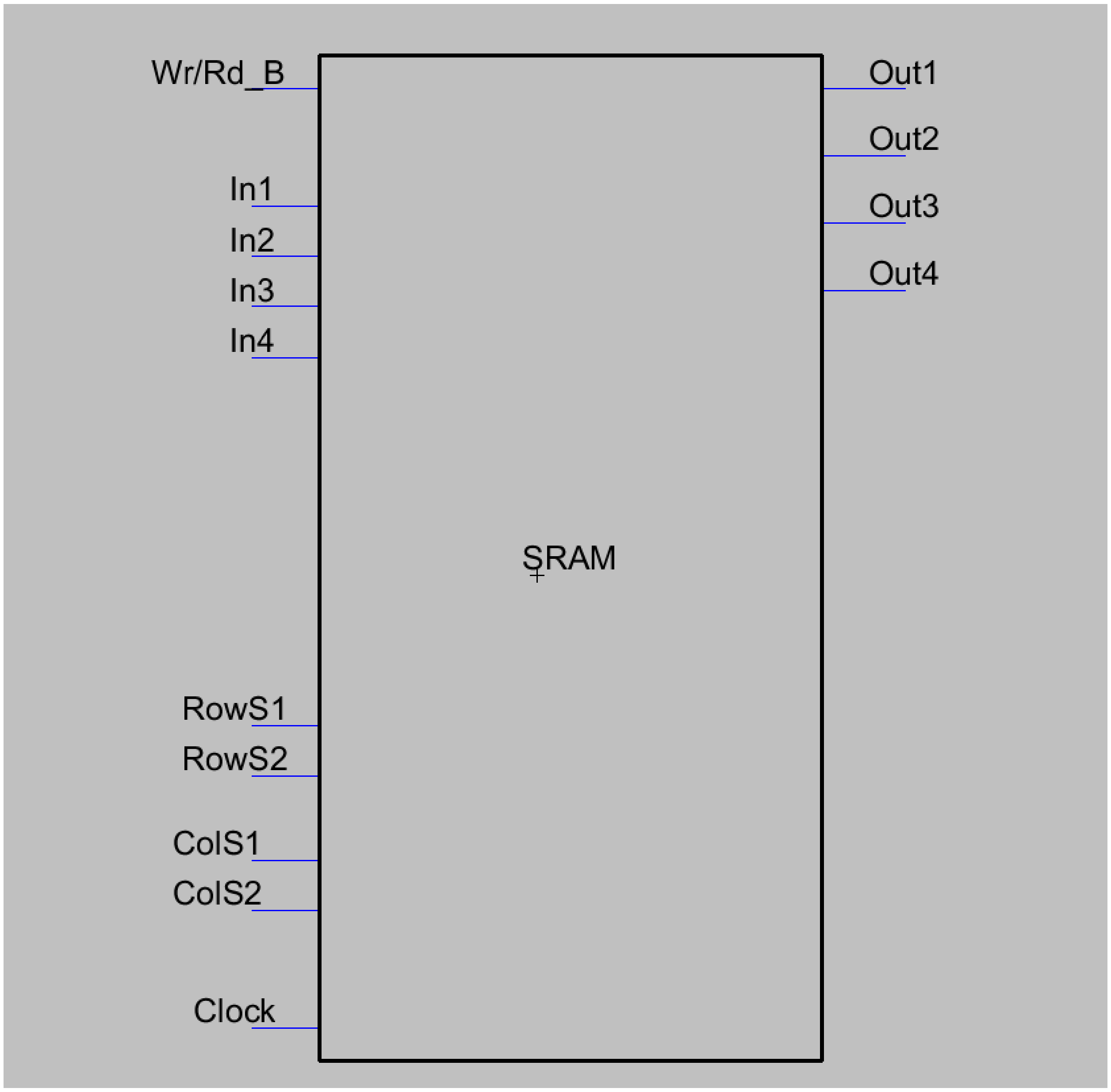
The lines that has red dash is the decoder lines. Those lines are the one which dictate which column will be working. Also, those lines are connected at the output side (RWL). Then, line 1 is just clock and line 2 is read and write signal.

2.6- Decoders



The write decoders has enable signal from the clock. This is because the decoder uses domino logic so it is high while precharging, which can result in writing on a word by mistake. However in the upper decoder I didn't need this because in the precharging if all Word and read enable became high without the second decoder then it is useless. Also I already protected my Word enable and read enable by introducing a stage that needs clock to be high to work(part 2.3).

2.7- Package



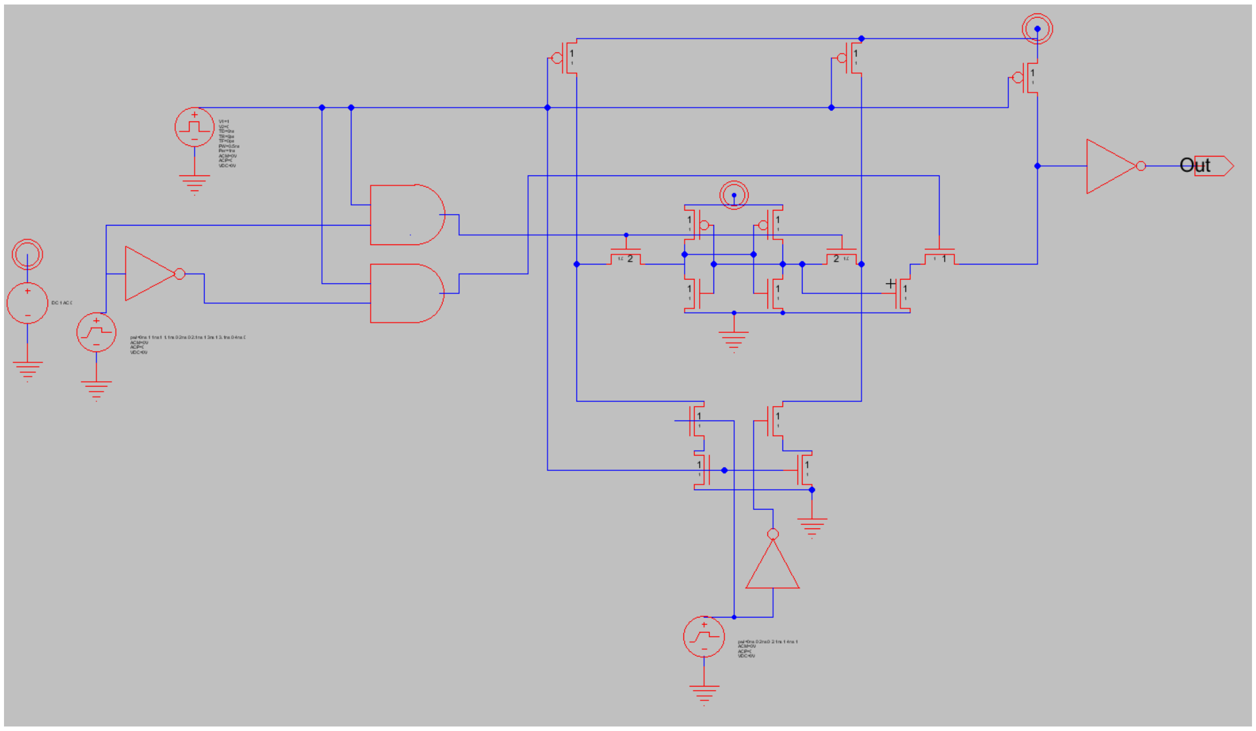
All of this gets abstracted into one cell. Here the signals needed are input data row and coloumn select (address). It is addressed such that address increases to the right and down with top left cell being 0000 and bottom right cell being 1111.

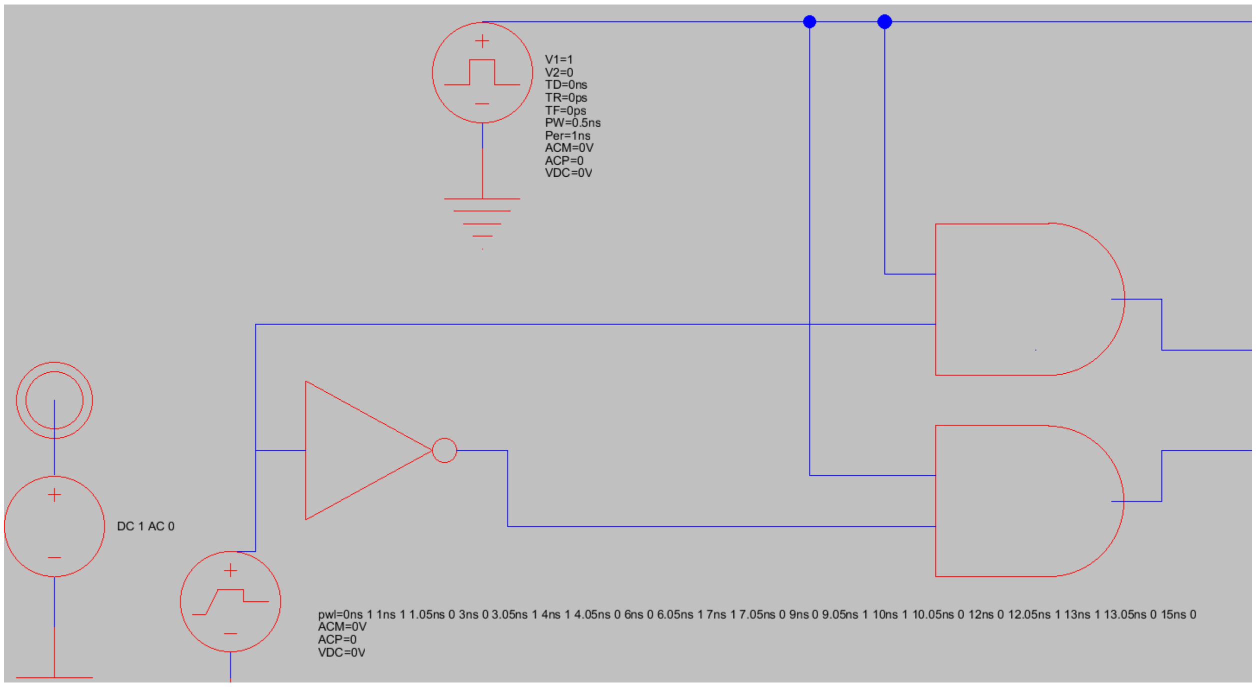
Also, This design needs one clock signal only.

Of course, The write is 1 and the read is 0.

3- Testing

3.1- Cell testing



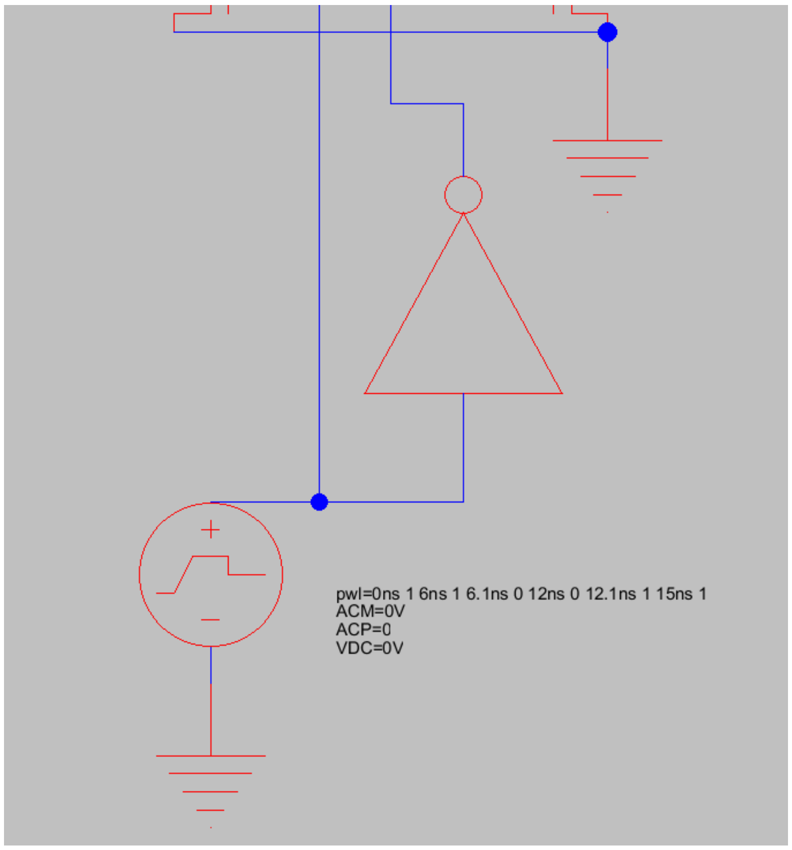


V1=1
V2=0
TD=0ns
TR=0ps
TF=0ps
PW=0.5ns
Per=1ns
ACM=0V
ACP=0
VDC=0V

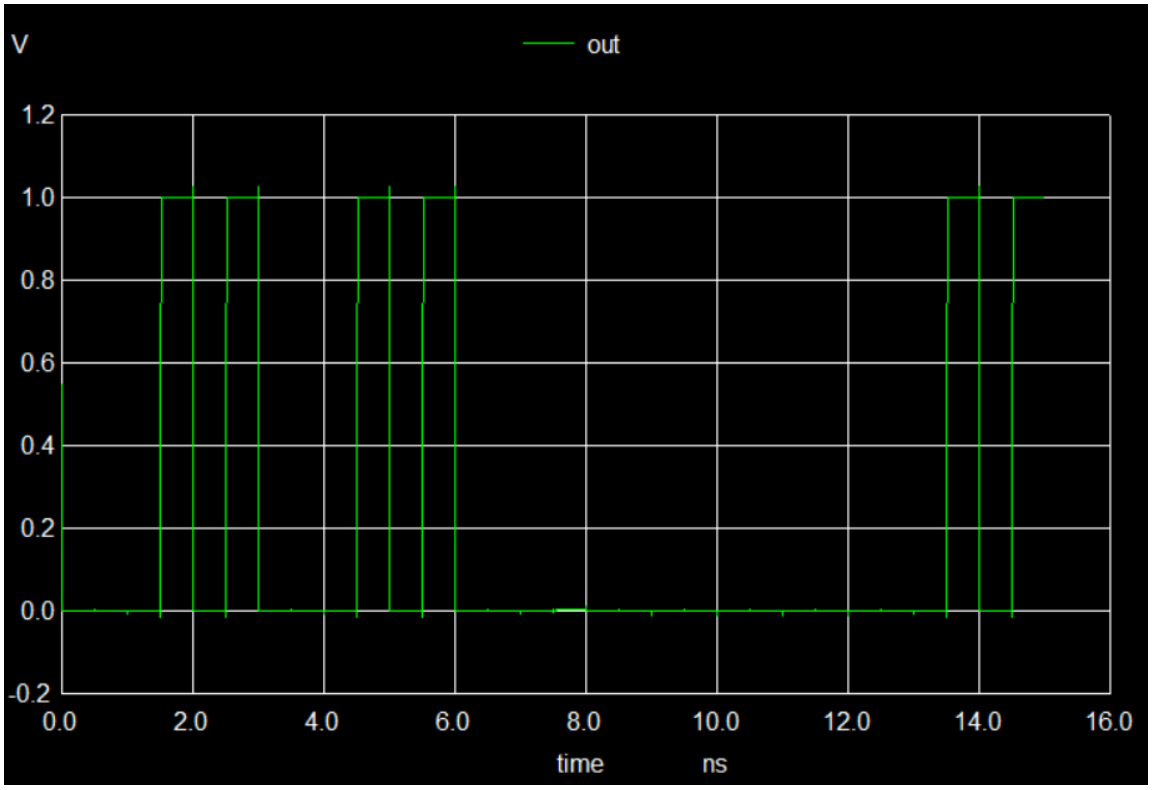
DC 1 AC 0

pwl=0ns 1 1ns 1 1.05ns 0 3ns 0 3.05ns 1 4ns 1 4.05ns 0 6ns 0 6.05ns 1 7ns 1 7.05ns 0 9ns 0 9.05ns 1 10ns 1 10.05ns 0 12ns 0 12.05ns 1 13ns 1 13.05ns 0 15ns 0
ACM=0V
ACP=0
VDC=0V

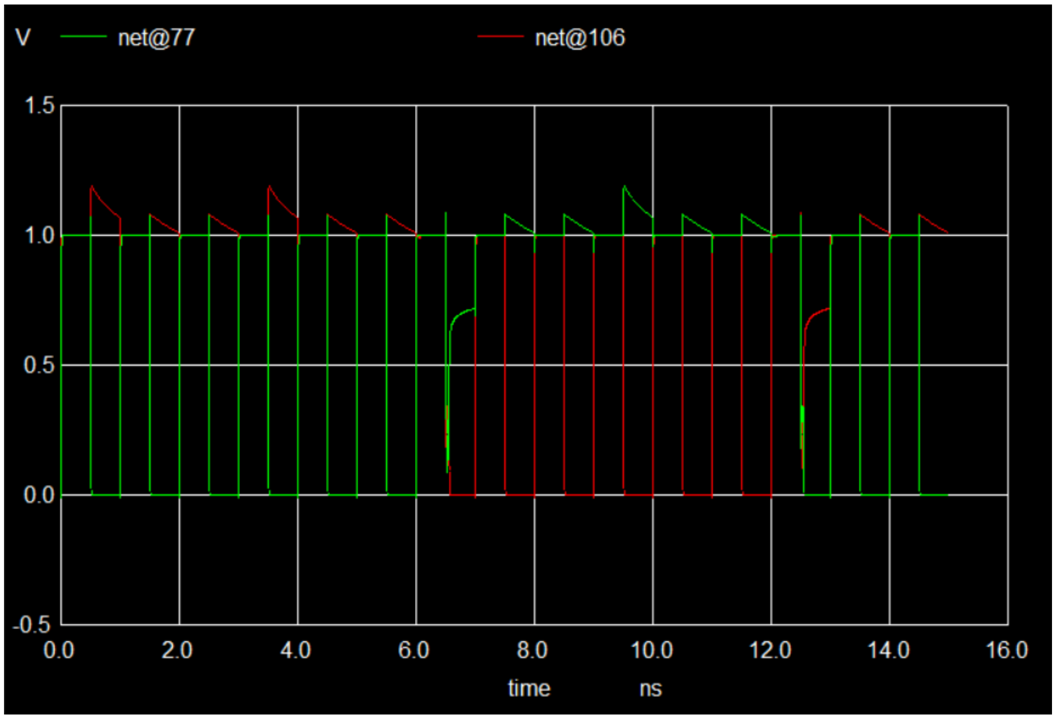
As we can see I want to read or write only when the clock is high and precharge when the clock is low. Also, I used the VPWL to test the same way it was said on ED discussion.



This is the input for the first 6 cycles it is 1 and the next six it is 0 and the last three it is zero.

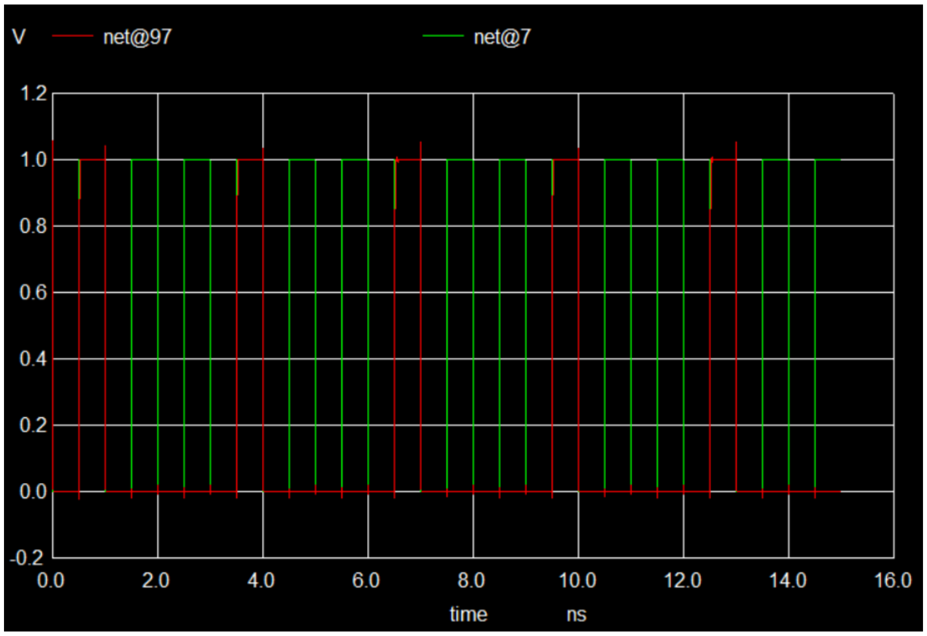


The output is as expected two ones then write then two ones then write 0 then two zeros then write 1 then two ones.



(note: this is inverted logic for the test of cell only (output has inverter on it))

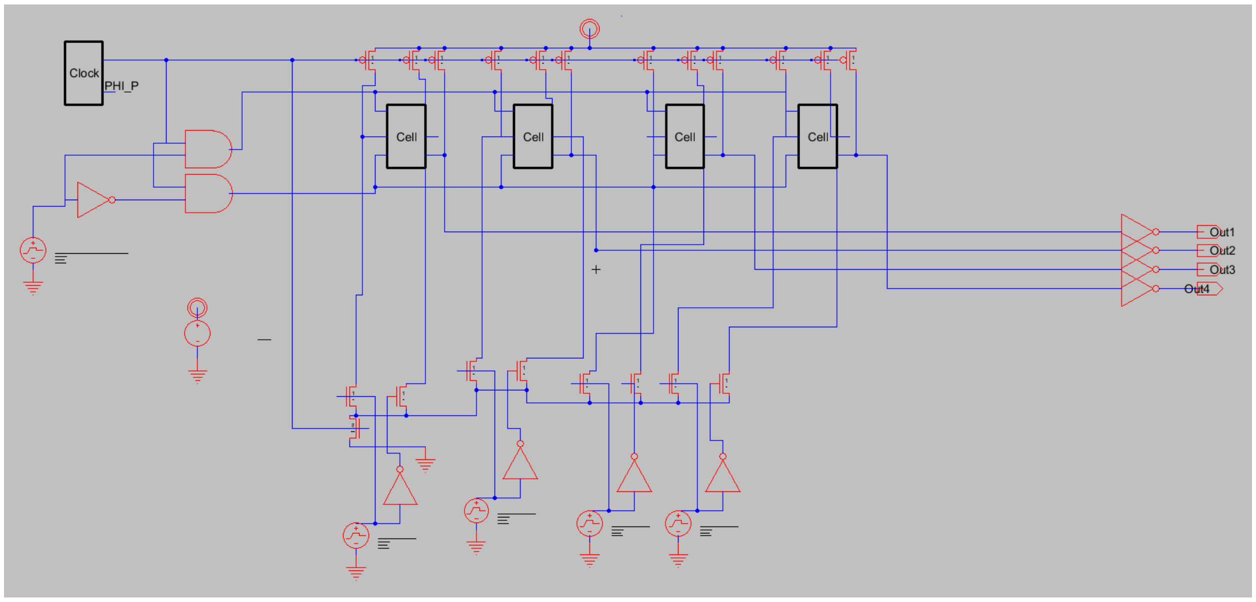
net@77 is BL and net@106 is BL_B. As we can see BL is always zero at the writing time except at seventh cycle it raised to nearly 0.7 to charge the cell and invert it. Then this happened to BL_B at the thirteenth cycle were it inverted again.

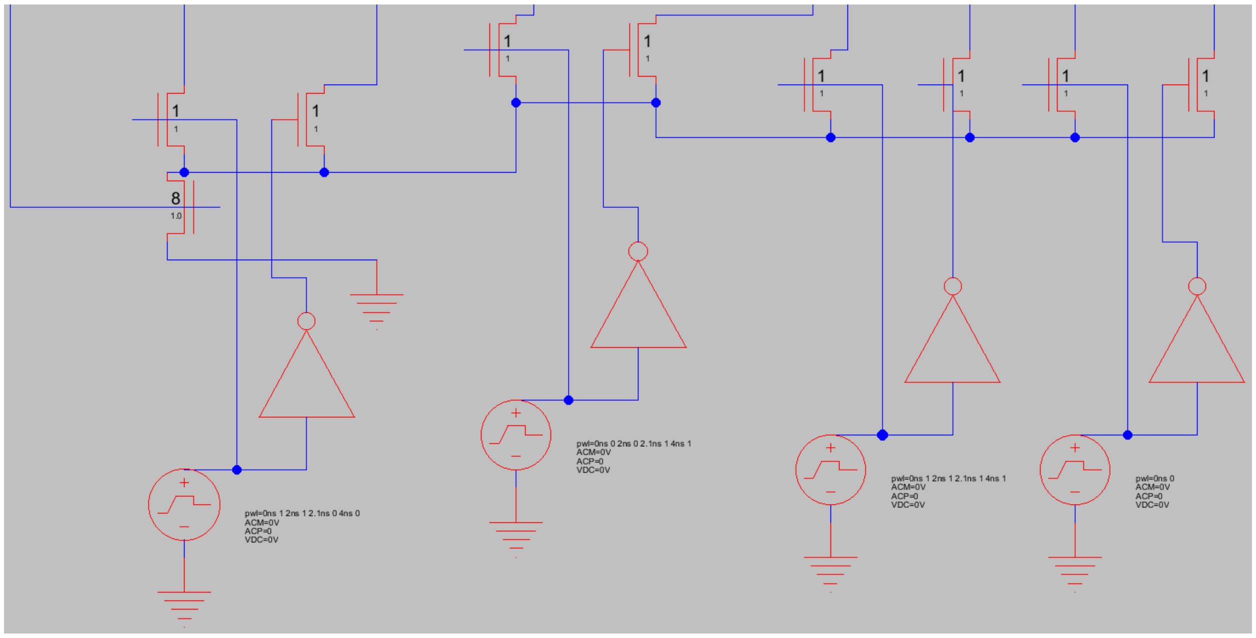


net@7 is clock and net@97 is the write enable line.

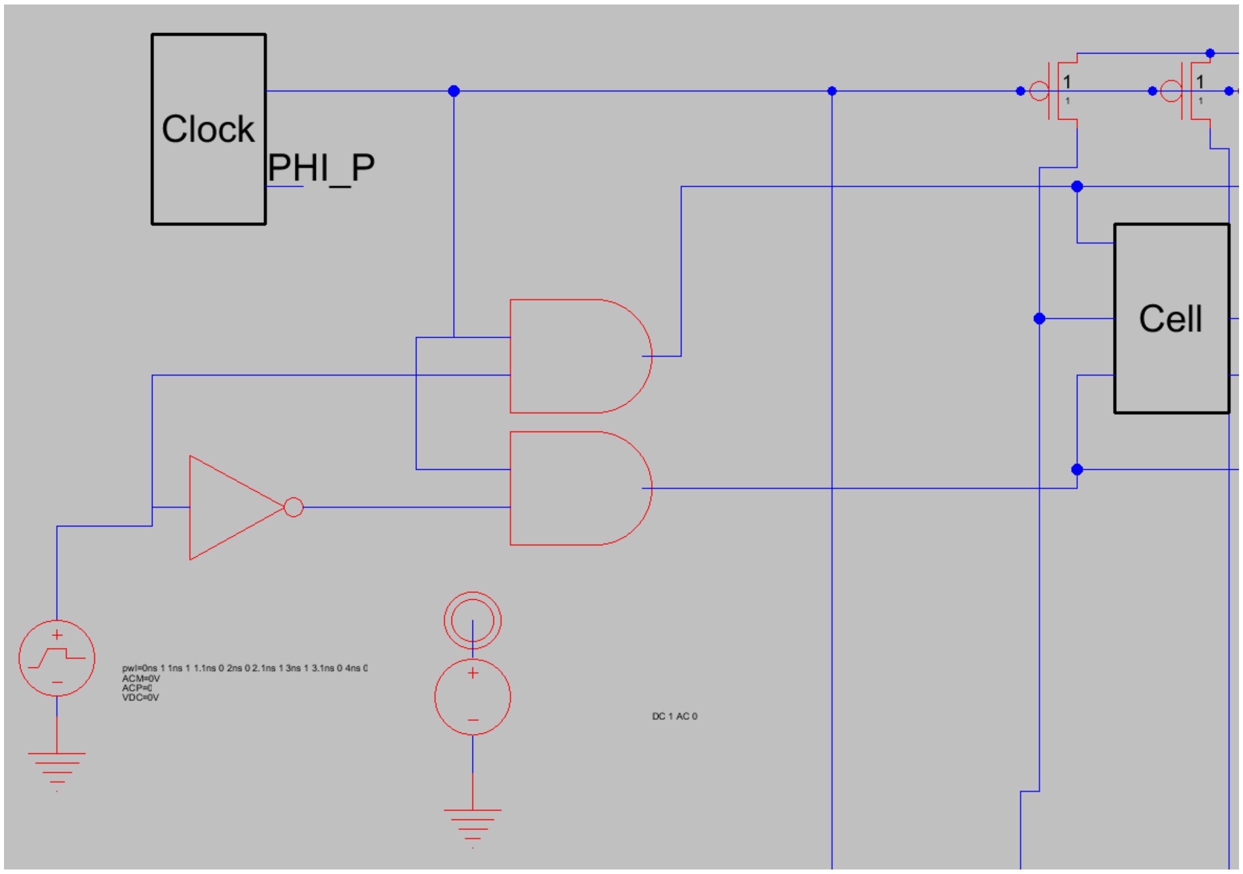
This is the best way to test a cell because we first try writing then we read twice to check if the first reading will change the stored value in the cell. After that we write the same value to check if really writing is working or not. Because the cell might have started in 1. Then we read twice to check again if the cell will change or not then we switch to zero to check if we can write 0 after 1 then read twice to check if reading will change anything. After that we write 0 again to check if we can write 0 after zero and read for the same reasons listed above. After that we write 1 and read to check if we can write 1 after zero and read for the same reasons above.

3.2- Word Testing



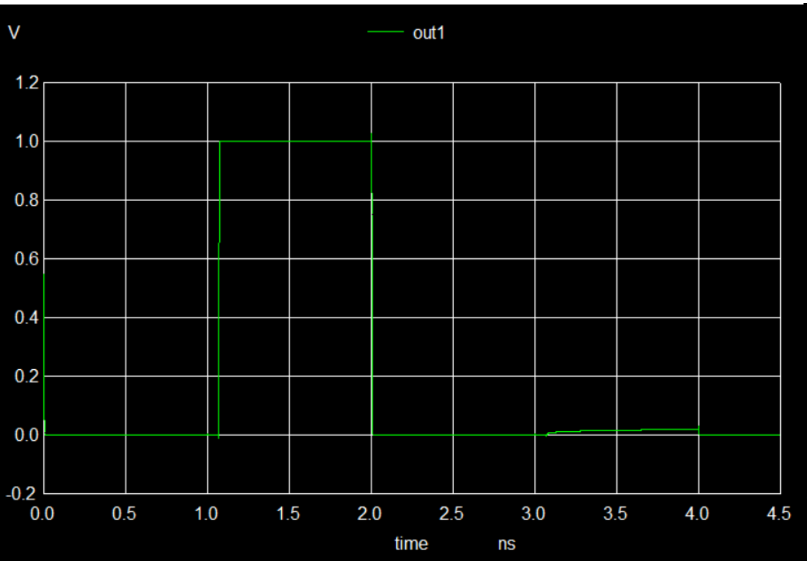

1
8
pwl=0ns 1 2ns 1 2.1ns 0 4ns 0
ACM=0V
ACP=0
VDC=0V
pwl=0ns 0 2ns 0 2.1ns 1 4ns 1
ACM=0V
ACP=0
VDC=0V
pwl=0ns 1 2ns 1 2.1ns 1 4ns 1
ACM=0V
ACP=0
VDC=0V
pwl=0ns 0
ACM=0V
ACP=0
VDC=0V

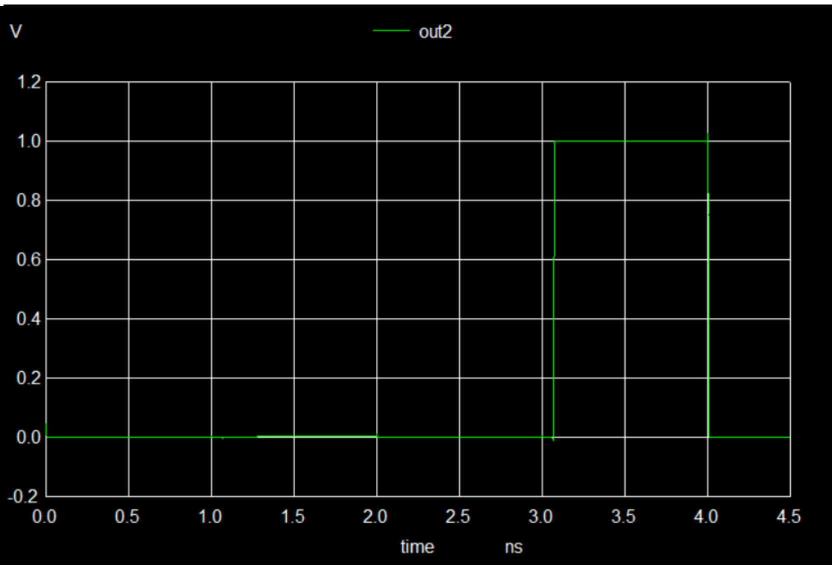
The input for first bit is 1 then 0 and second one is 0 then 1 and the third bit is always 1 and the fourth bit is always 0. This way I am testing all inputs transitions for each cell. Also, I don't need to test all 16 inputs transitions because the cells are parallel, so if one is working then the other will suffer the same environment exactly.

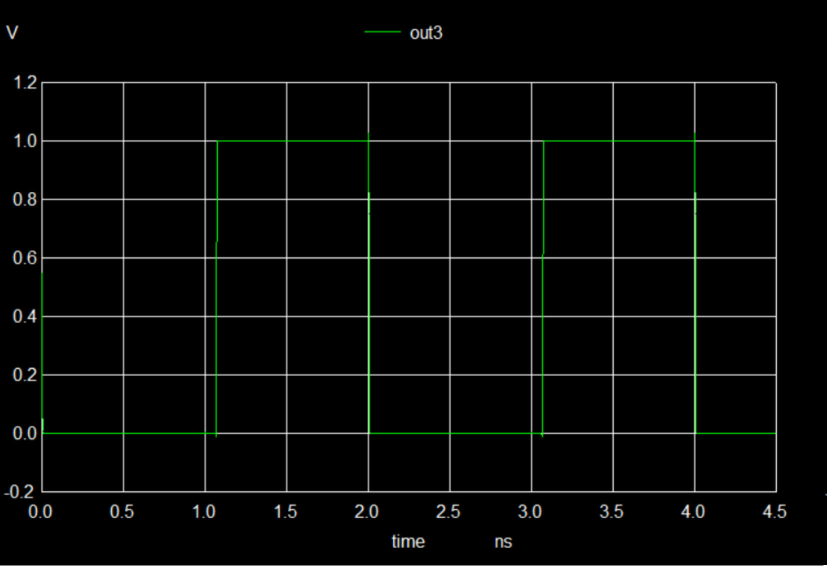


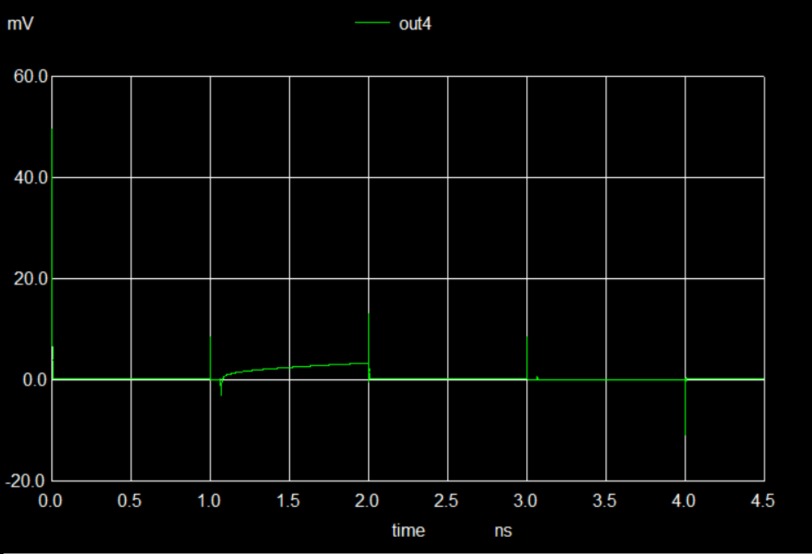
Regarding, the read and write signal it is the same as the cell one. Here I use my clock block too. The word line and the read line is driving for all cells simultaneously.

The results:



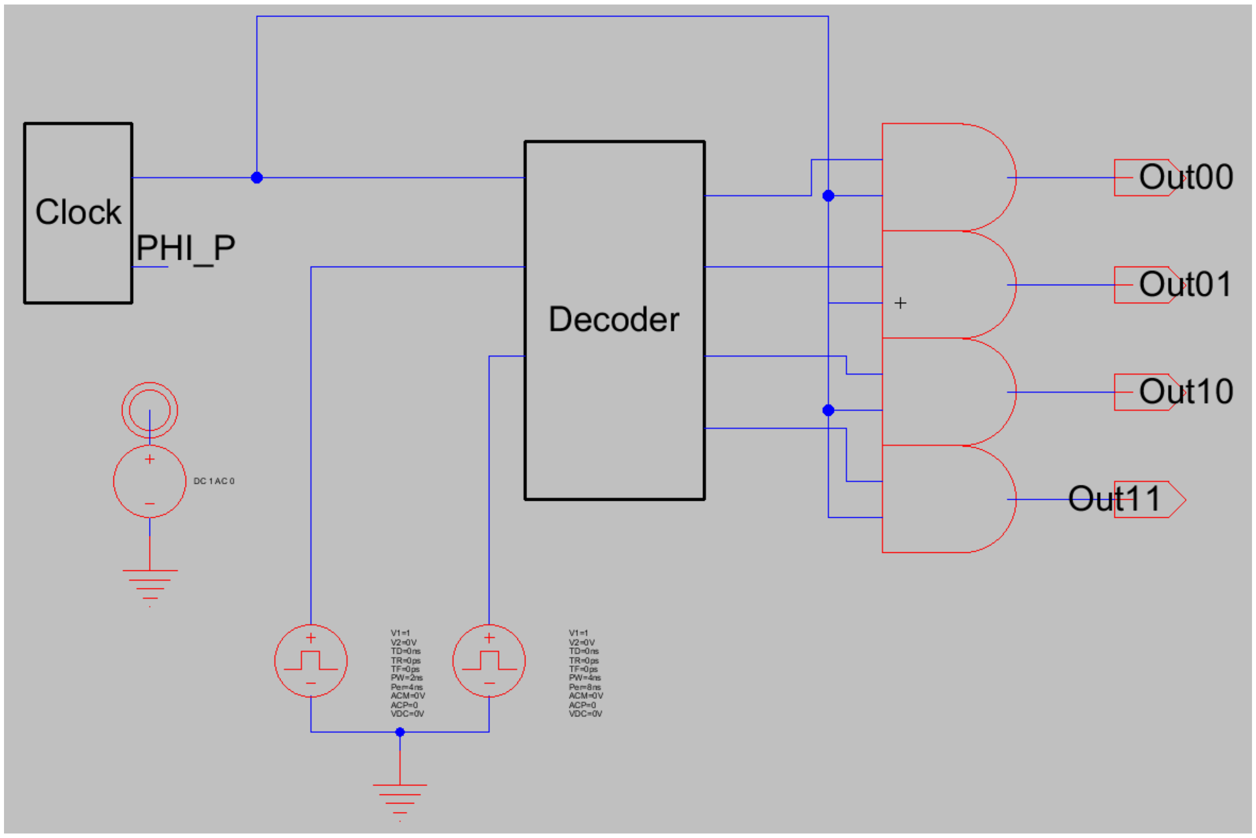






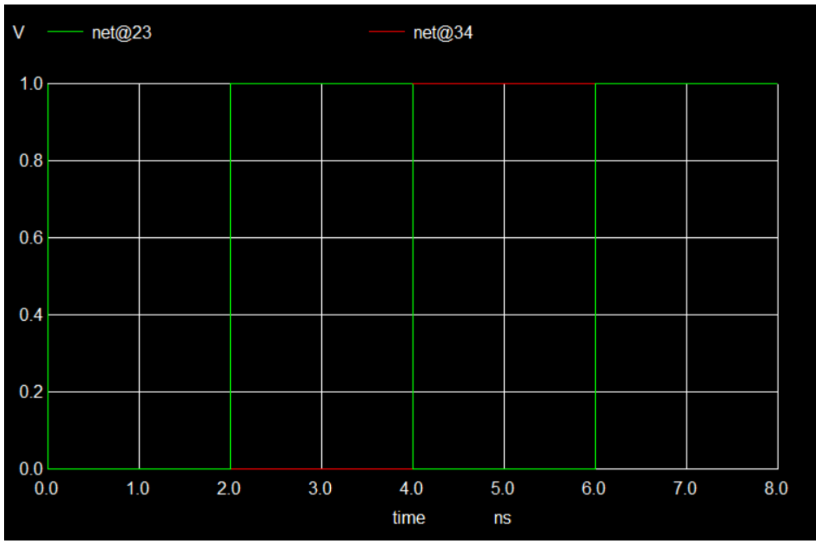
As we can see the results are right.

3.3- Decoder



The clock period is 2ns. I am inputting 00 then 01 then 10 then 11 and I am and-ing the output with the clock so that the precharging doesn't distort the outputs. Also, I used pass logic, so I need the AND to act as an enable buffer.

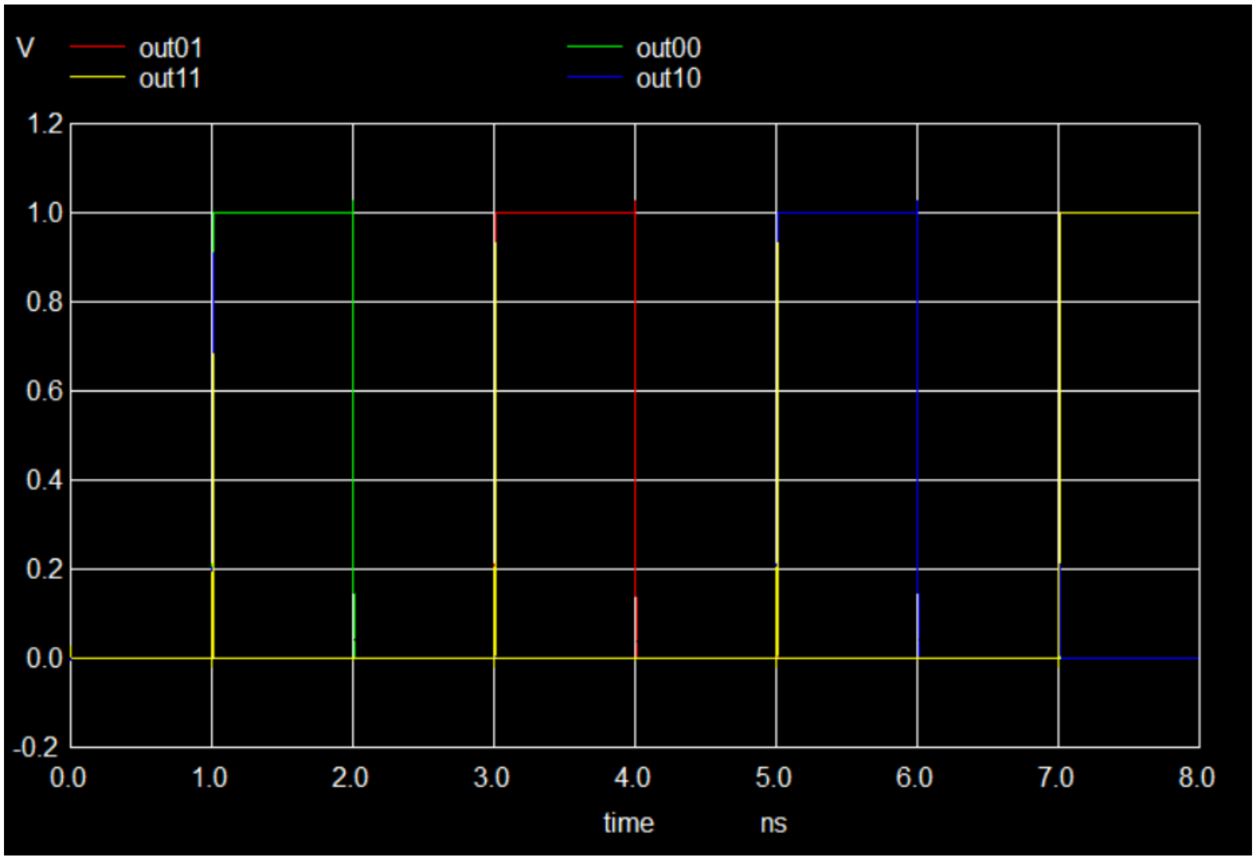
The inputs:



net@23 is A0 (first input)

net@34 is A1 (second input)

The output:



The output follows the input right. In the evaluation cycle everything is going good.

3.4- SRAM testing.

I already tested the schematic before abstraction and verified it is working and as you know it is huge so I will test on the abstraction in the report, because it will be easier to show and read from screenshots (I don't want to increase the length of this report without any reason and make your life harder by testing on the unabstracted one).

I will do the following 3 times:

1- two words in different columns and different rows.
2- Two words in different columns only but same row.
3- two words in different rows but same columns.

I will test writing this inputs on the two words 1010 and 0110 then switch them. I will write on both two words first, then read both, then write on both, then read them again.

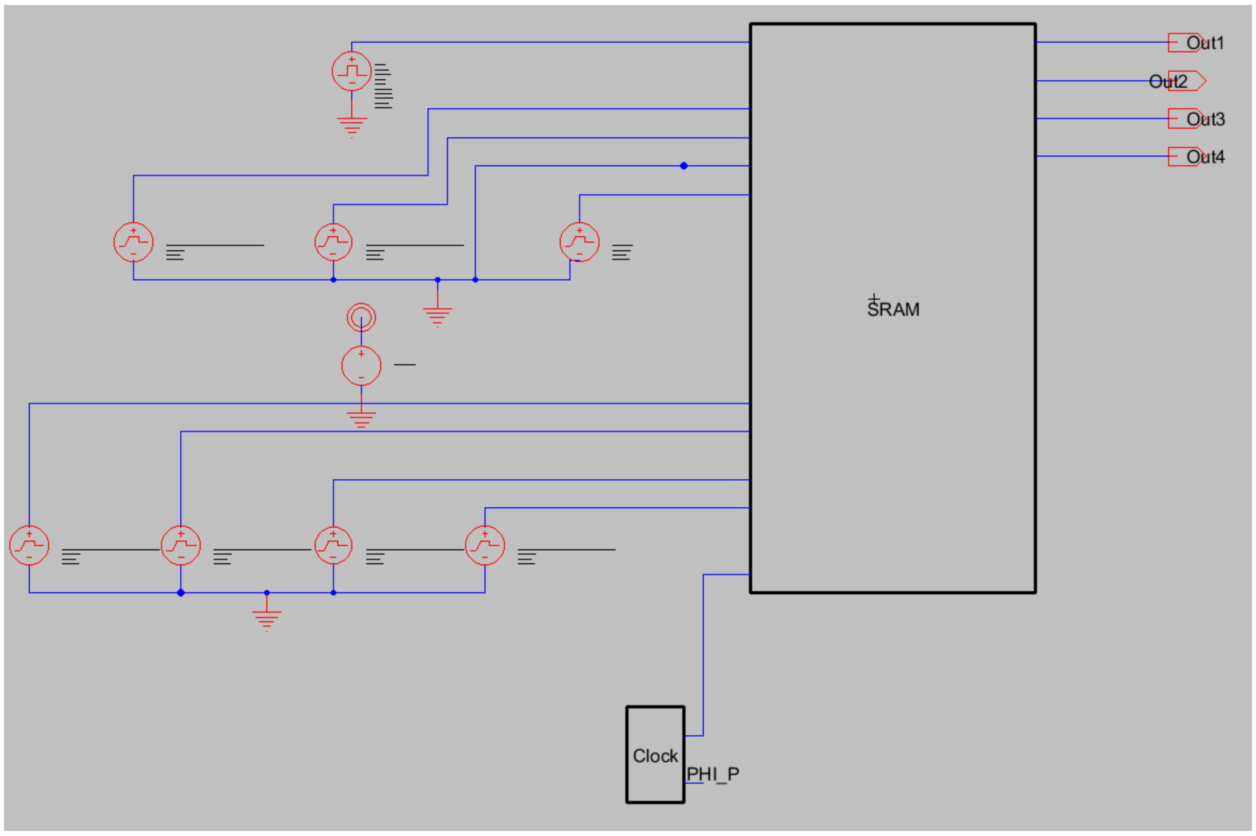
Why this is the best way of testing the circuit?

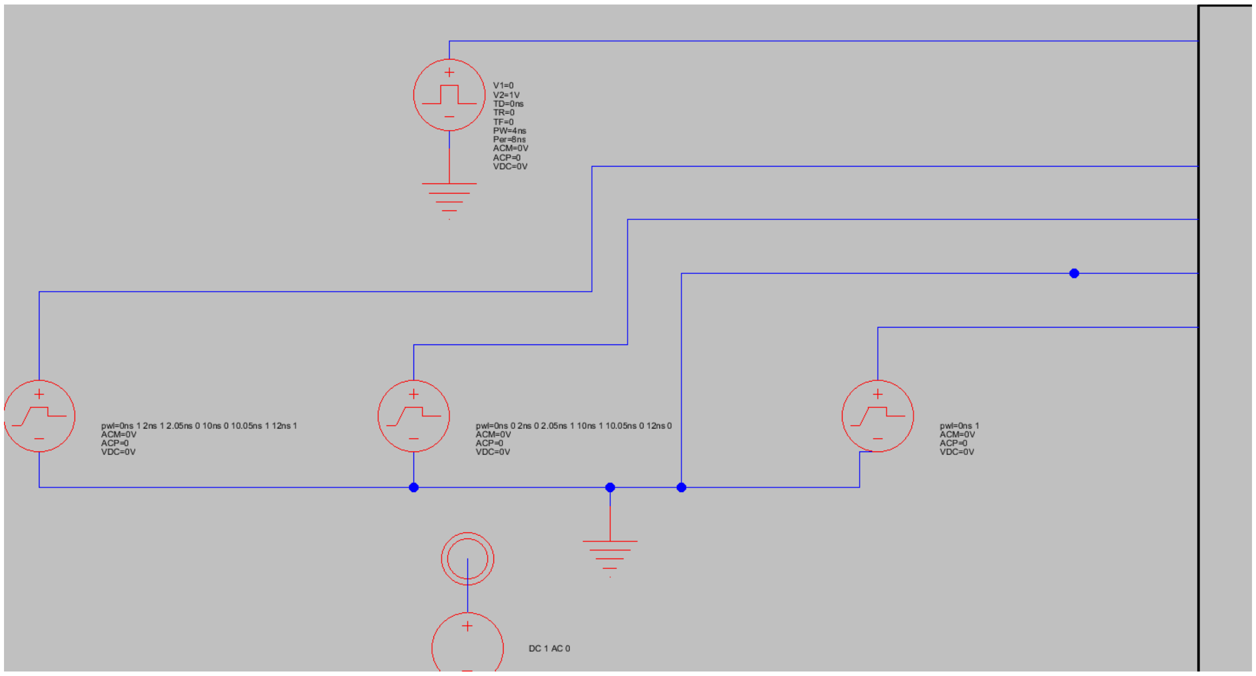
1- I will write on two different words then read this will tell me if writing different data on two words would affect the other or not.
2- I will read both after each other this will tell me if reading is working properly if I am reading two words after another
3- Also the three cases tested is the general cases that can happen (The possible relation between any two words)
4- The data inputs test changing inputs from 1->0 and 0->1 and 1->1 and 0->0.

I will use a clock speed of 500MHz, which is the minimum to also verify that it is working on the minimum.

In the next section, I will test the maximum speed this circuit can operate on, but here I want to verify the circuit connections and functionality. If the circuit is functional then I will just target one word and see its response to increasing the speed since all of them are identical when it comes to access time.

The circuit being tested:



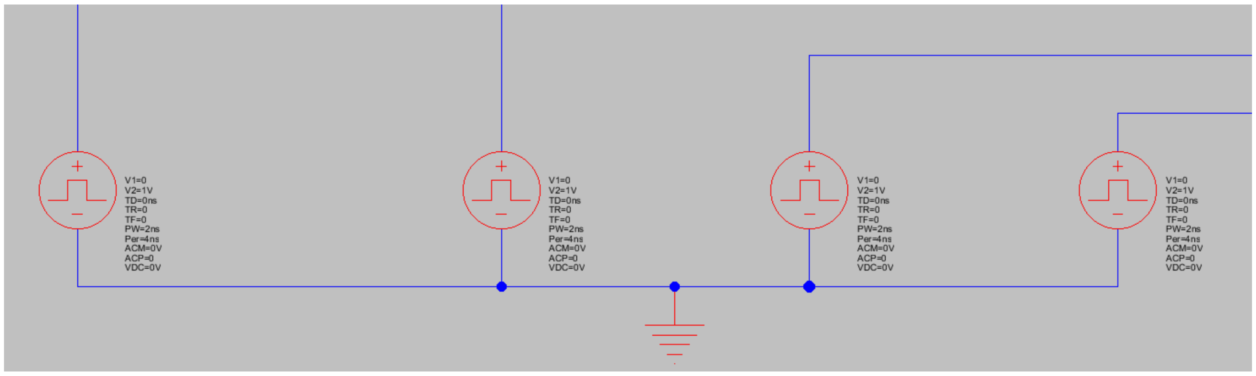


The upper one is just a square wave such that for the first two cycles it is 1 and the second two cycles it is 0 (read) then third two cycles 1(write) fourth two cycles 0 (read) again.

Regarding the input, it is as said above.

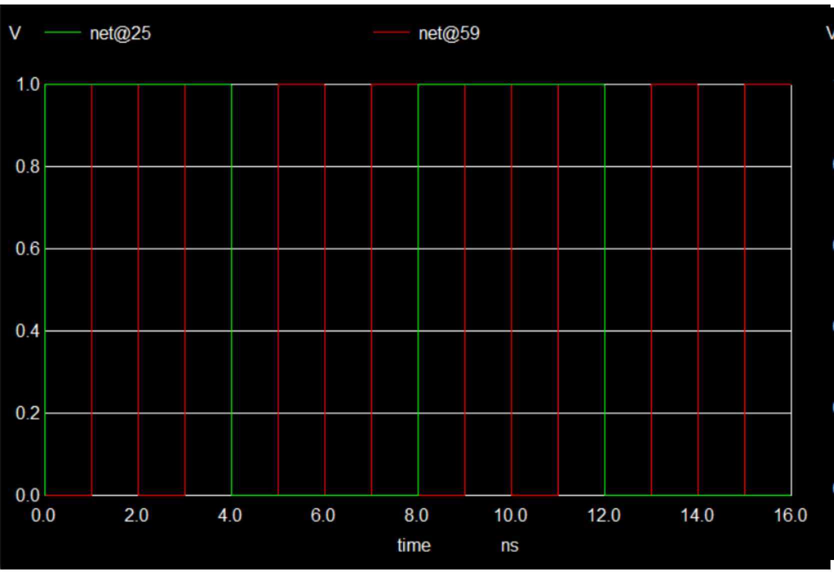
3.4.1- 1st case

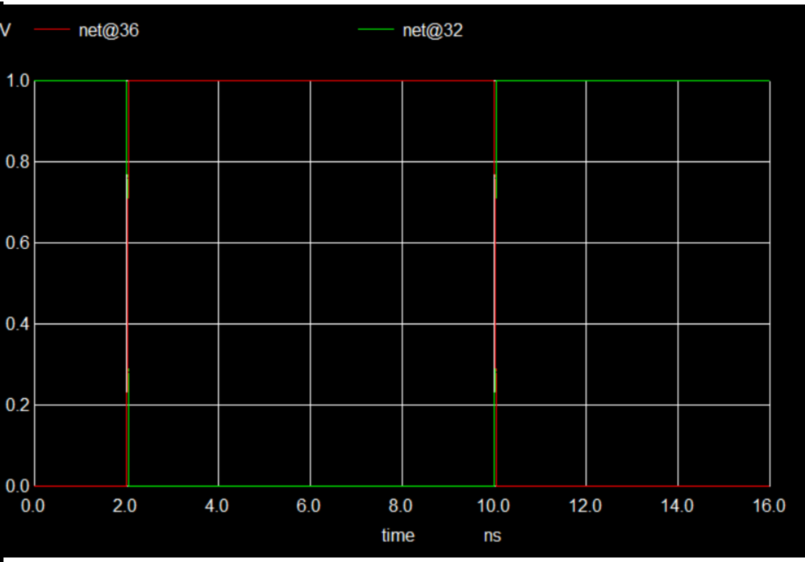
Here is the address: (0000 – 1111) (note: there is oscillations on Out pins due to precharging which can be eliminated by and gate with the not of W/R signal(useless). If this is going to be in a real circuit we will only read output pins when we request a read, so outputs at any other time except reading time is irrelevant.(but I will put it in speed test)



The first cycle is 1111 second cycle is 0000 ....etc first two is writing second two is reading ...etc

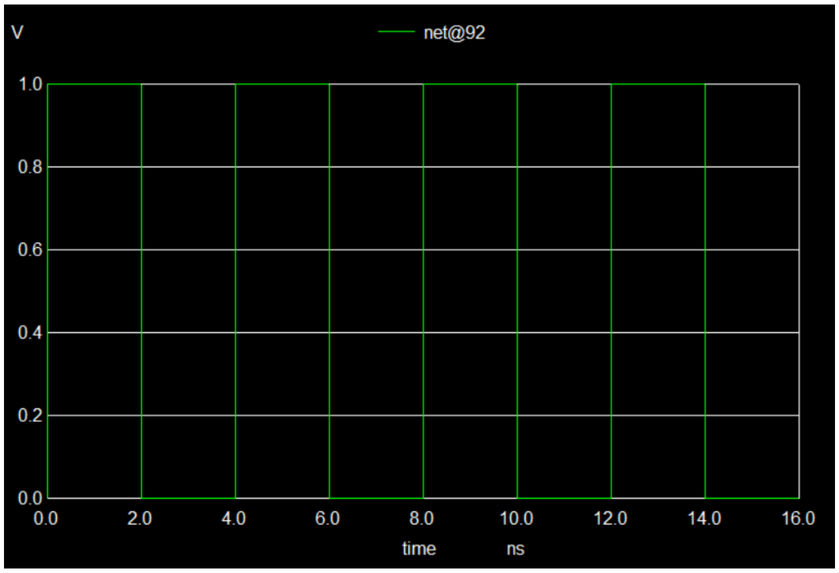
net@25 is W/R_B Enable and net@59 is clock.



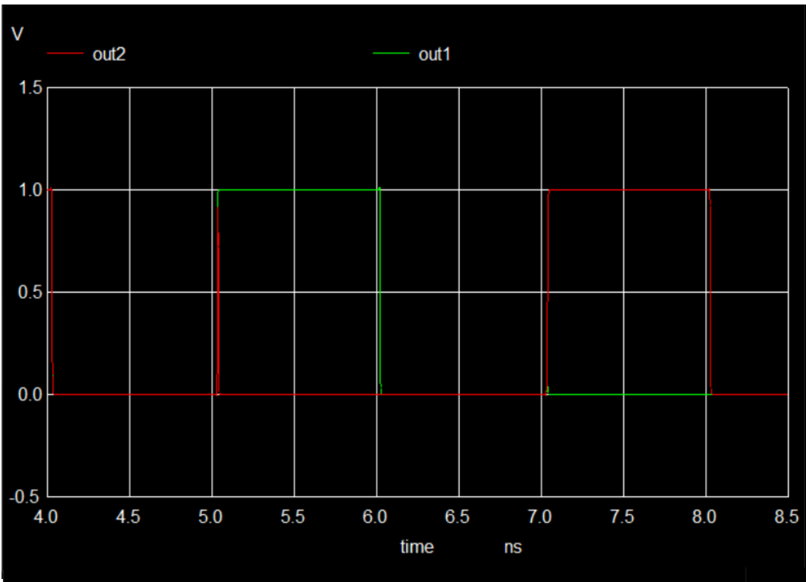


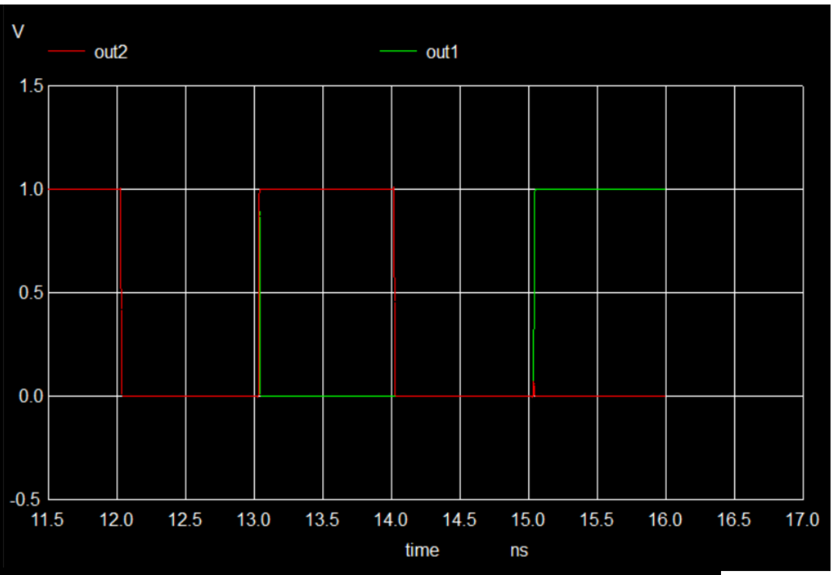
net@32 is the first input in first writing cycle it is 1 and second writing cycle is 0 then two reading cycles then 0 for a cycle then 1 for another writing cycle and then two reading cycles. net@36 is second input and no need to write specifications it is the same as net@32 but inverted. For the third and fourth input they are ground and VDD respectively.

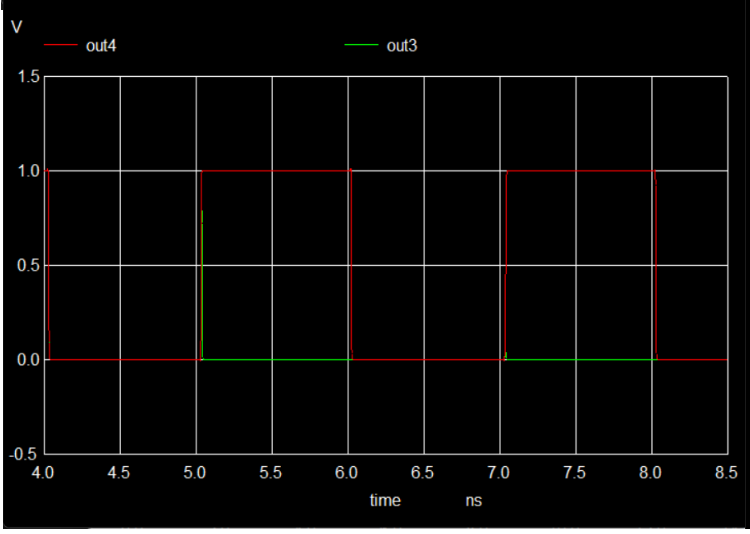
net@92 is the first address bit and the other address bits are the same since we are addressing 0000 and 1111 and it is changing each cycle to alternate between them and measure if they have effects on each other or not.

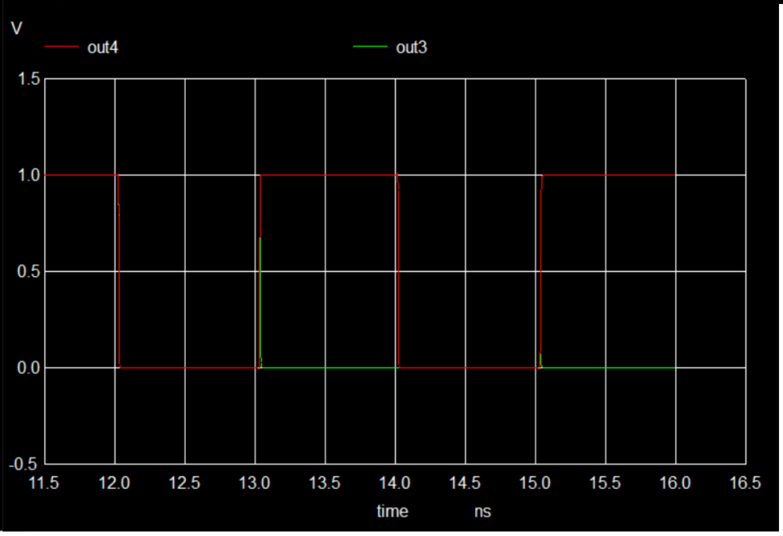


The output: Reading of 1111 is at 5-6ns and 13-14ns. Reading of 0000 is at 7-8 and 15-16ns





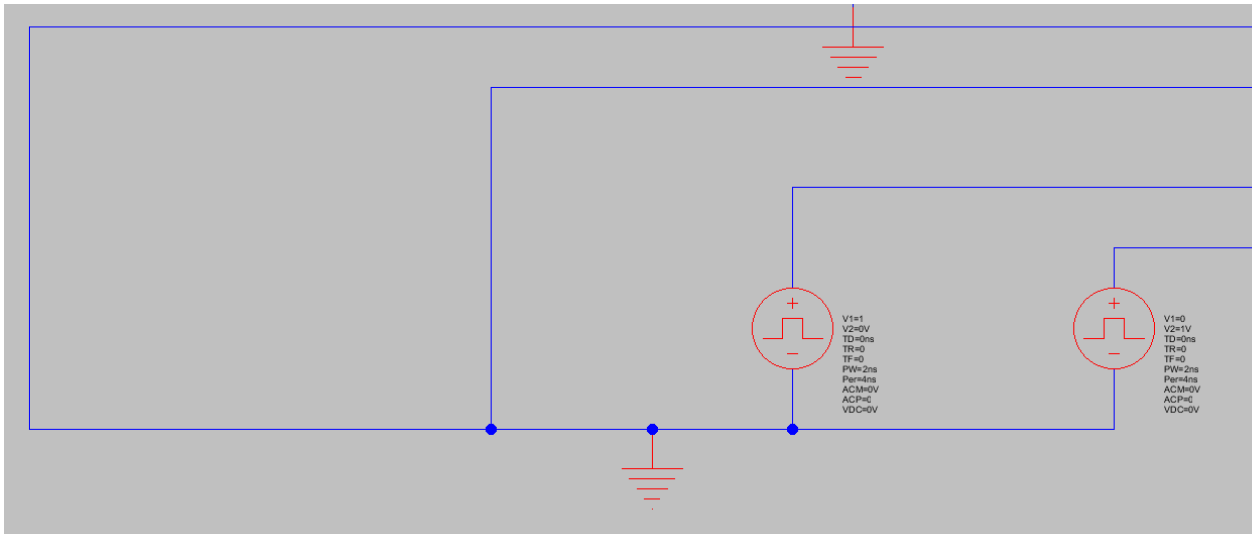




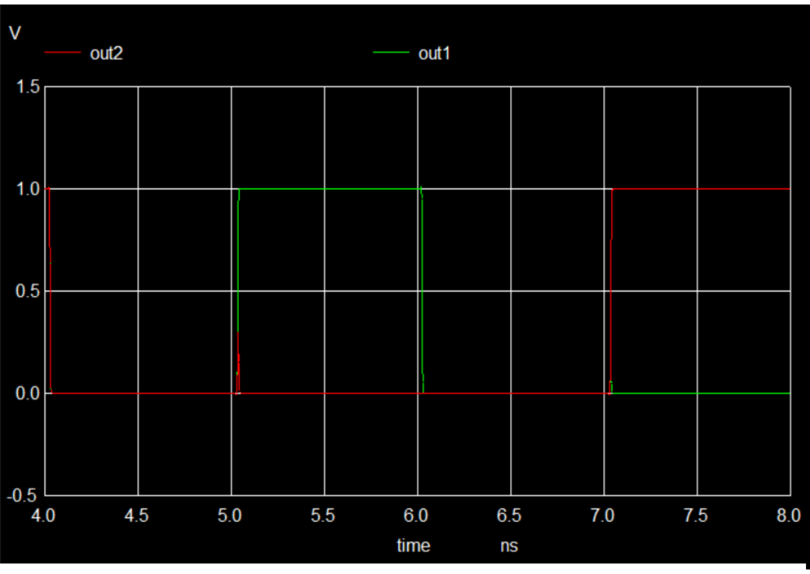
3.4.2- 2nd case

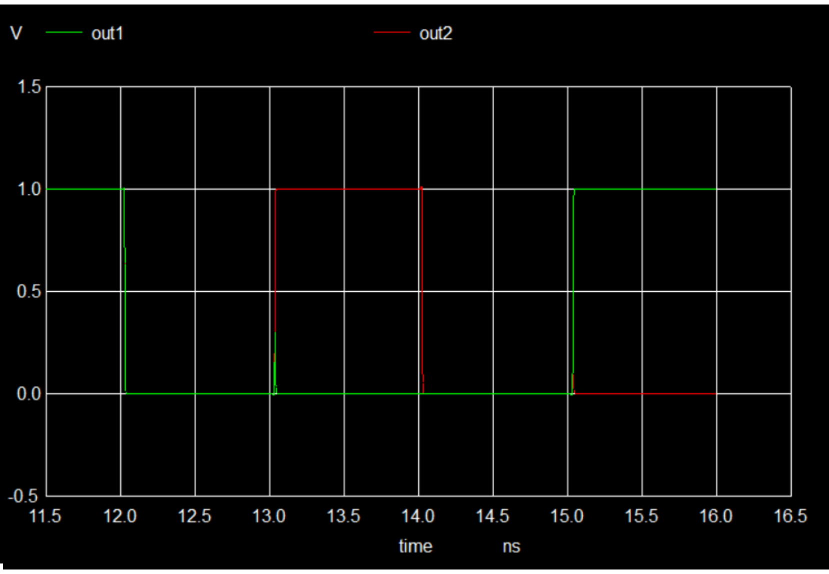
We will test two words that are beside each other(same row different column) (0001 – 0010)

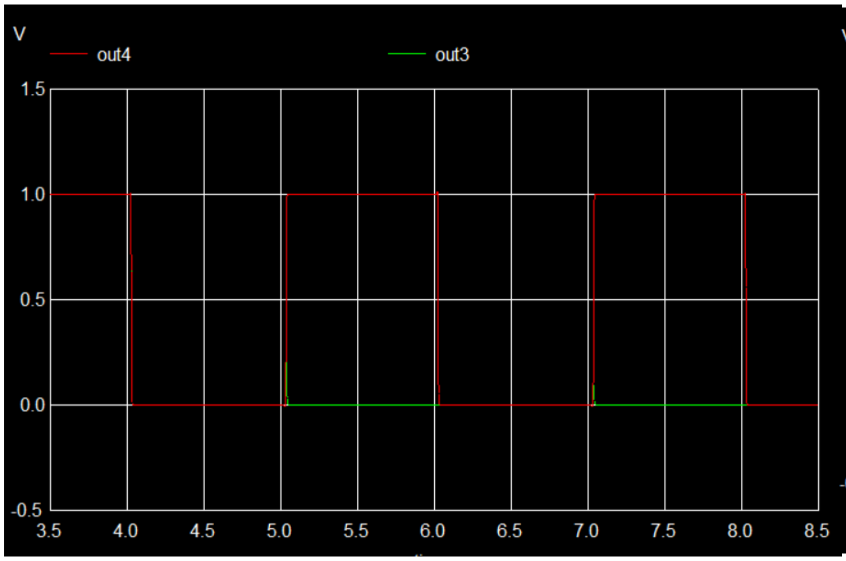
The address

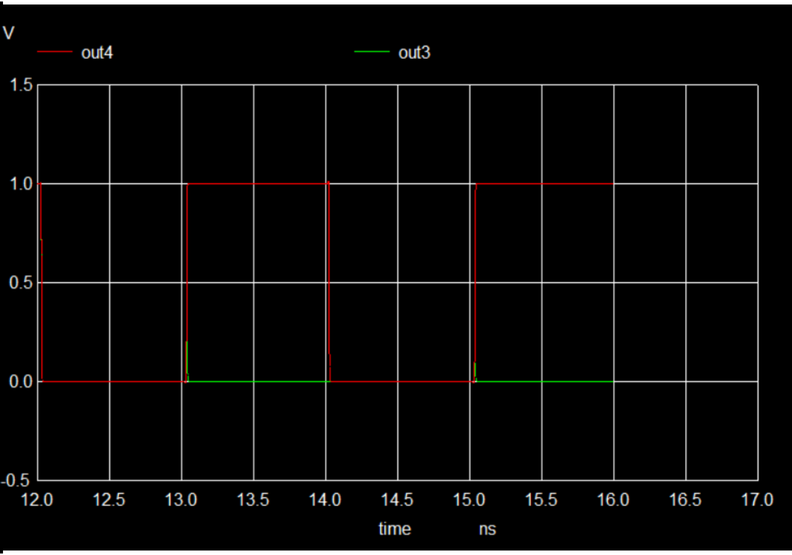


The outputs:





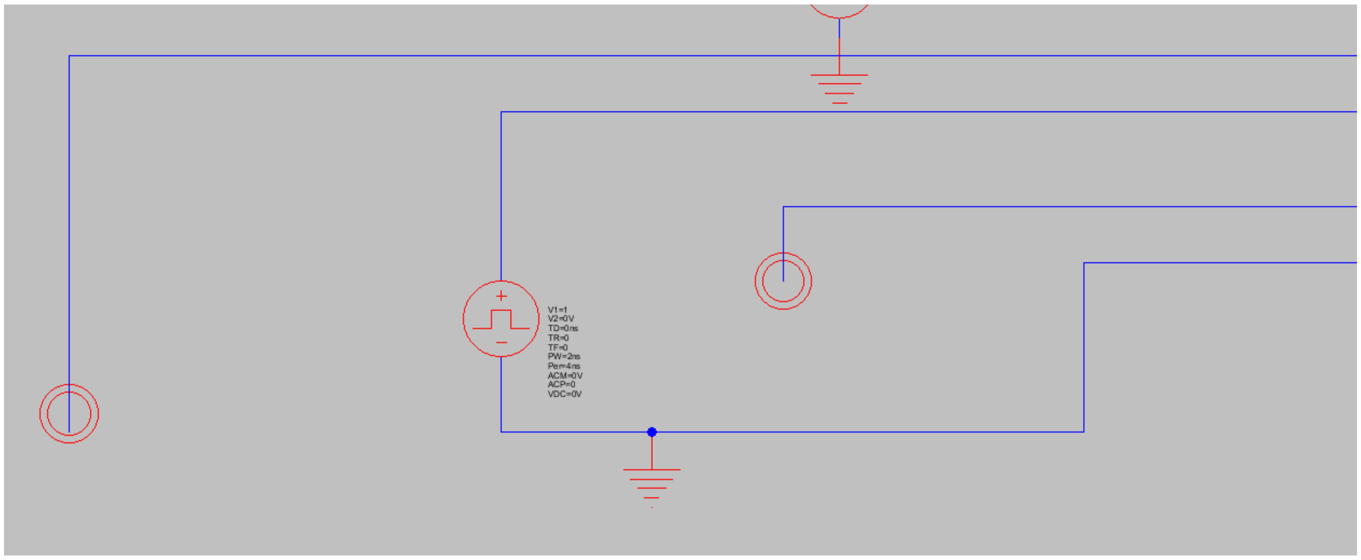




3.4.3- 3rd case

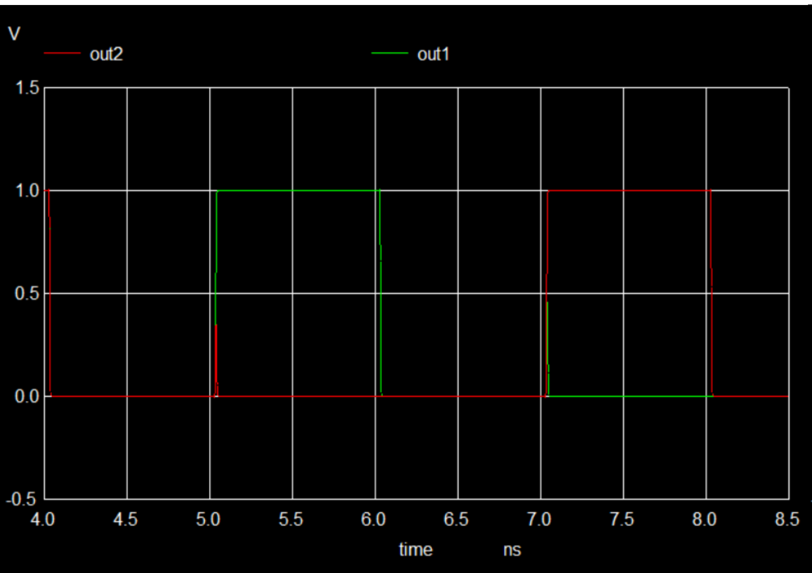
We will test two words that are beside each other(same row different column) (1010 – 1110)

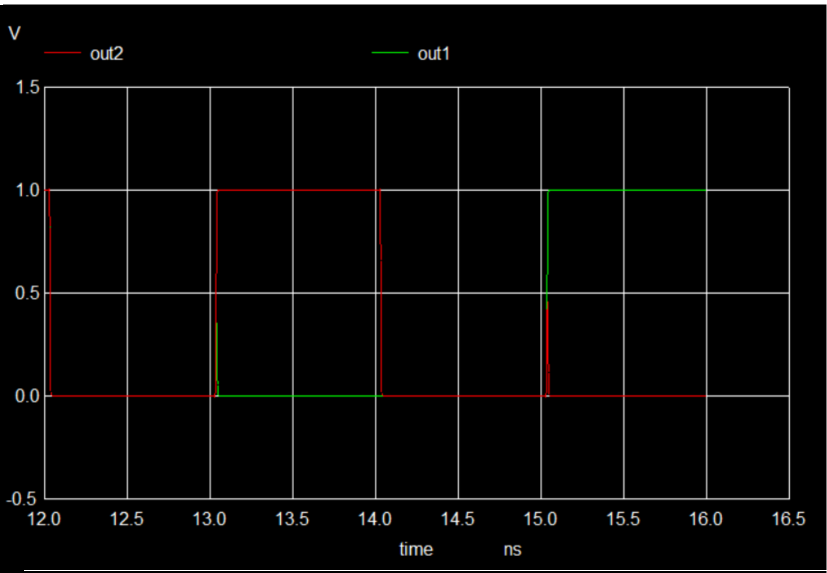
The address:

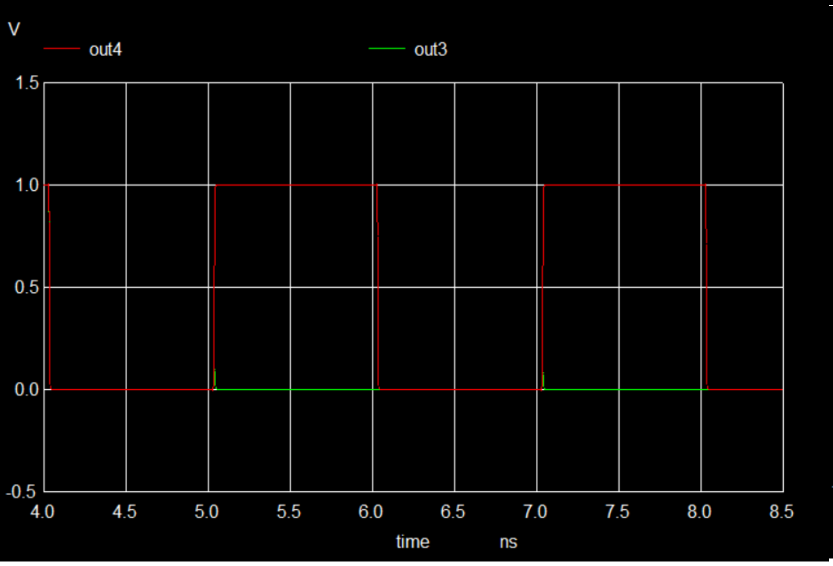


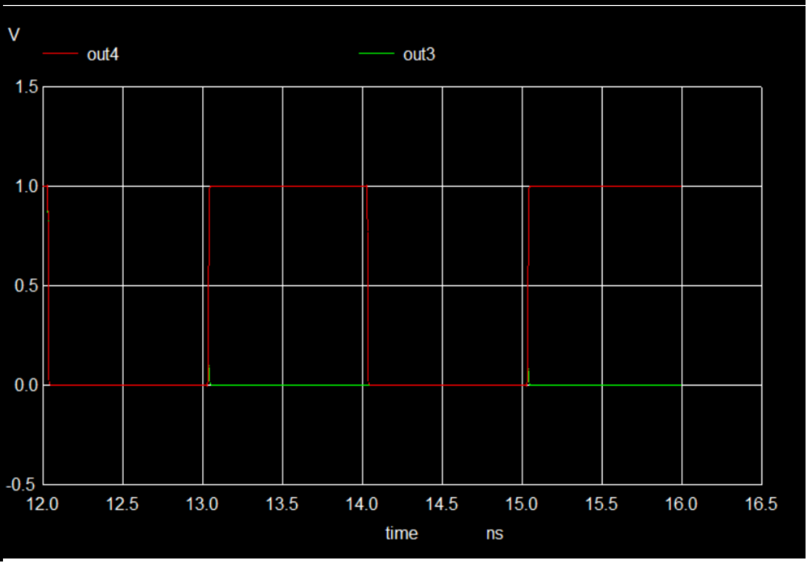
The outputs

Same case as the one above it is working









4- Performance

I will be using the packed and the unpacked version of the SRAM to find the optimal delay. Also, I will be filtering the output using two stages of AND in the delay and power calculations. This is an equivalent of a latch because if I used a latch I will do something similar to extract the data. It is irrelevant to do this here but I will do it to have a better looking output.

4.1- Area

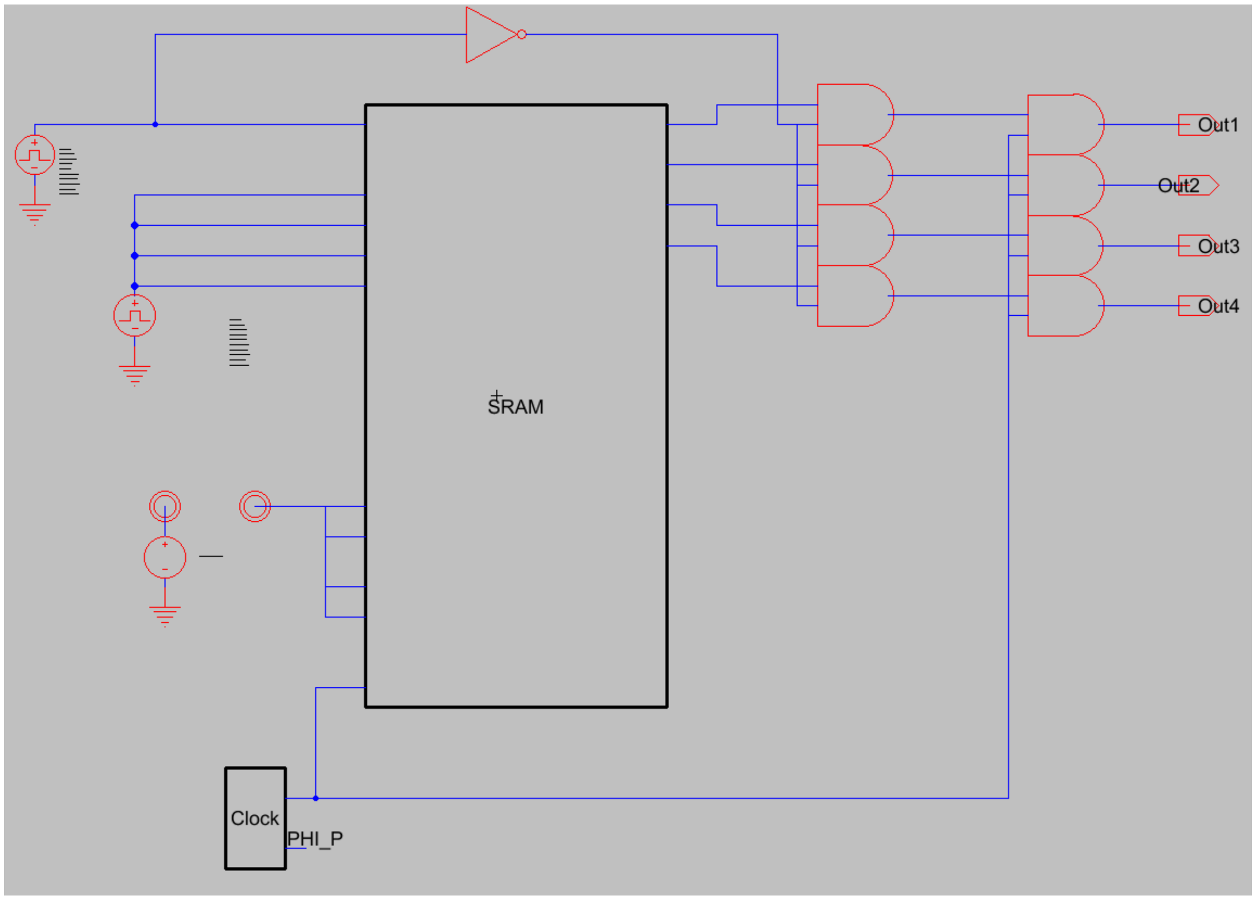
As we have seen in part 1.1 the area is 10.

4.2- Delay

4.2.1- Reading time

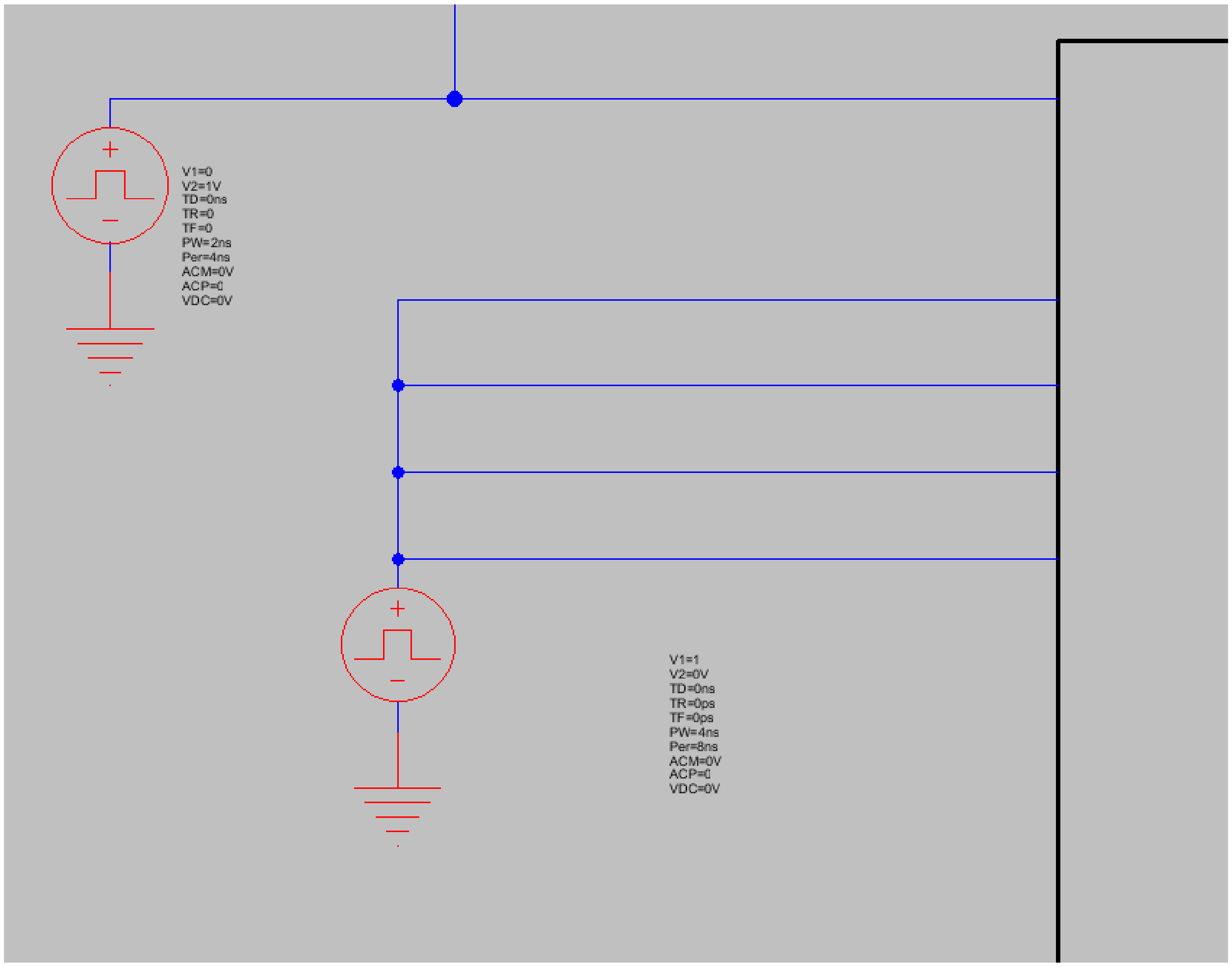
In the reading time, I will put the address as 1111 since all addresses are symmetric, so it doesn't matter, which address I chose. Also, I will initialize my cell to all 0000 and try to read from it then initialize all of it to 1111 and try to read it. It doesn't matter, which cell I am reading from because all cells in a word are symmetric.

The circuit:



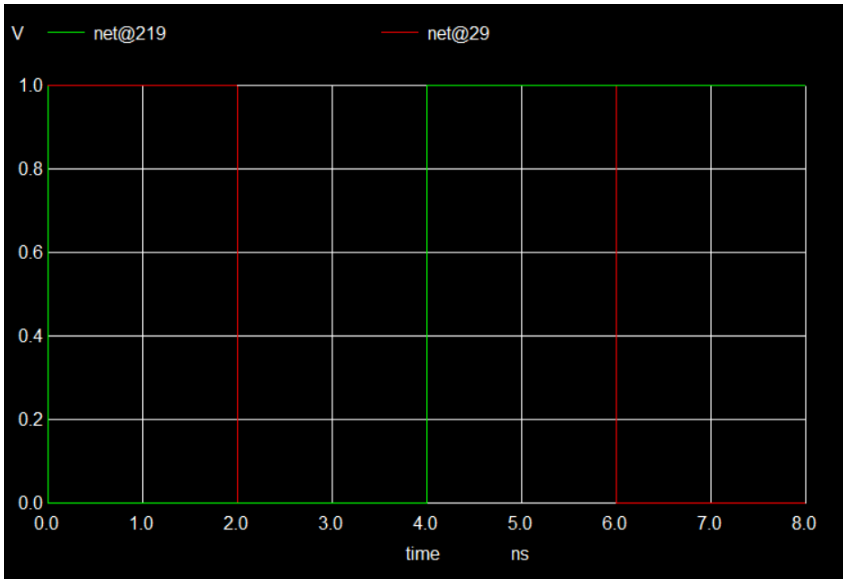
The input: (I will write in the first cycle then read in the second cycle then write in the third cycle then read in the fourth cycle)

I will write 0000 then 1111

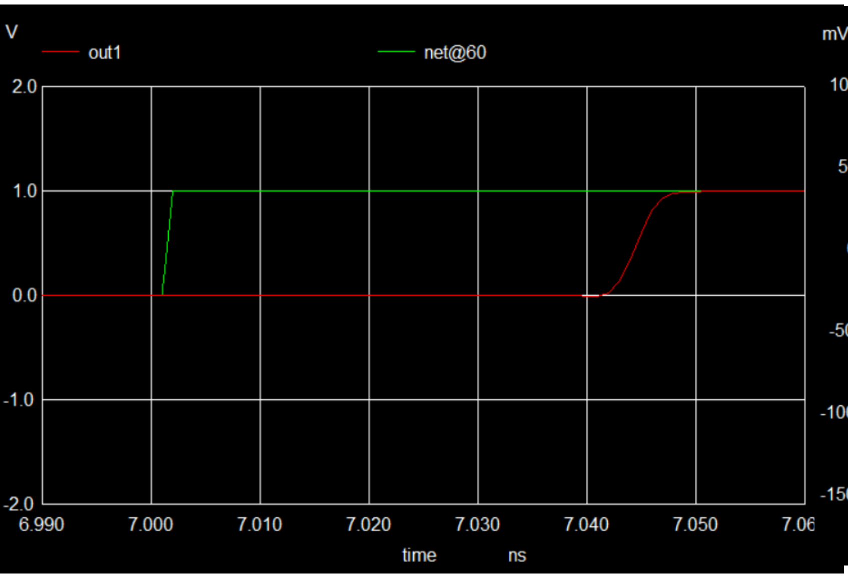


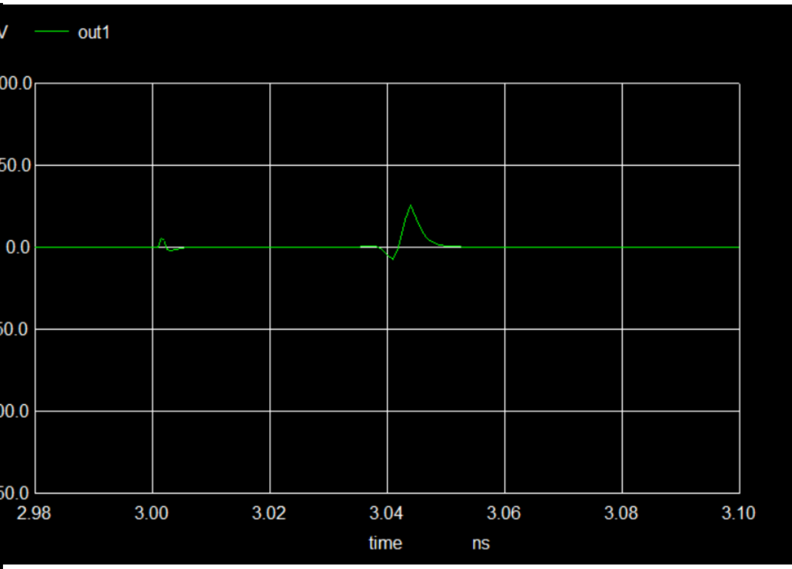
net@219 is the inputs

net@29 is the write/read signal.



The outputs: net@60 is clock





The reading time for 1 is 0.05ns. As we can see some perturbation happened in reading time for 0 and the output became stable after 0.053ns.

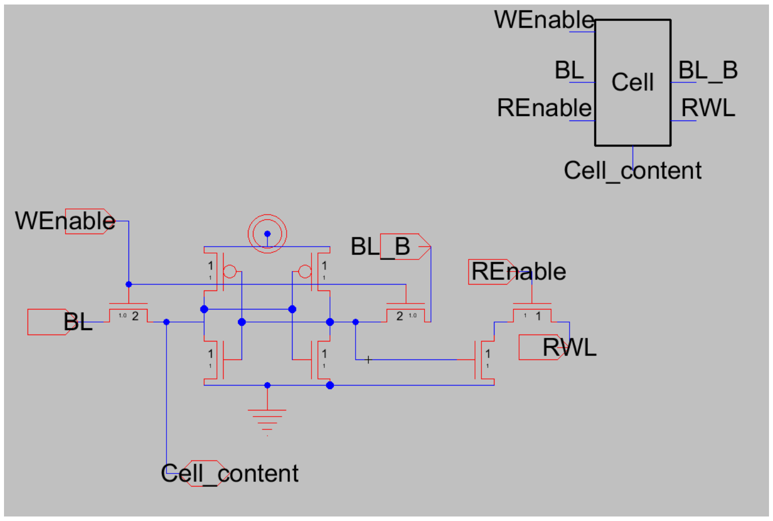
As a result the reading time is 0.055ns.

4.2.2- Writing time

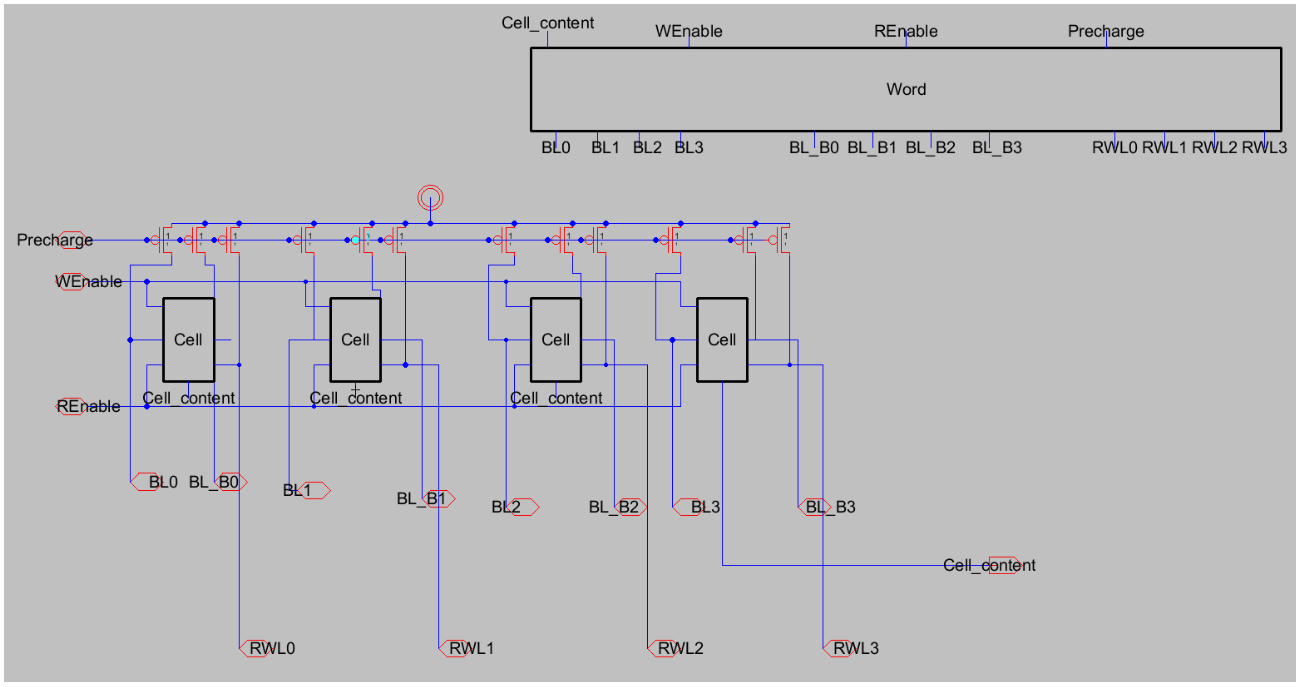
In the writing time, It will be tedious to replace the SRAM with individual cells (unpacked) in order to read the input. As a result, I will create a new library and edit the single cell schematic such that there is a pin out of the heart of one of the cells. It will become clear with pictures.

Regarding the inputs I will try writing 1 after 0 and 0 after 1 because those are the worst cases. Also, I will write on address 1111, because, as reading, all addresses are symmetric.

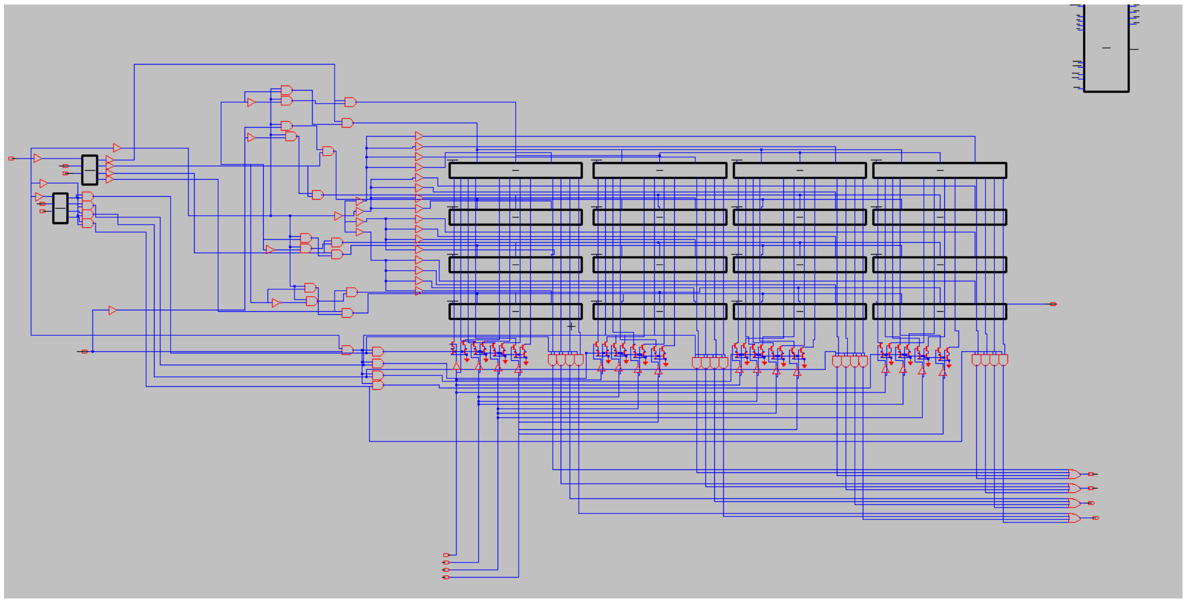
Cell:

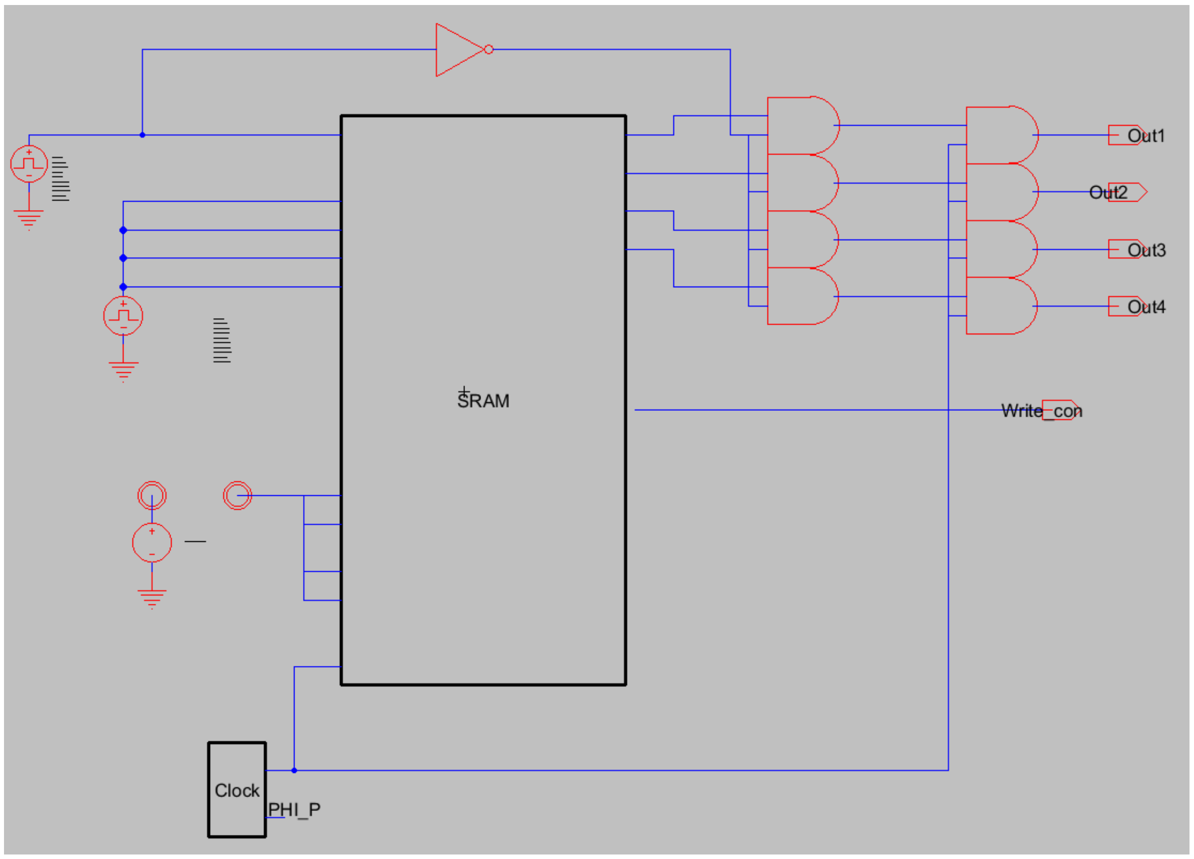


Word: (I used one cell as they are all symmetric)



SRAM:

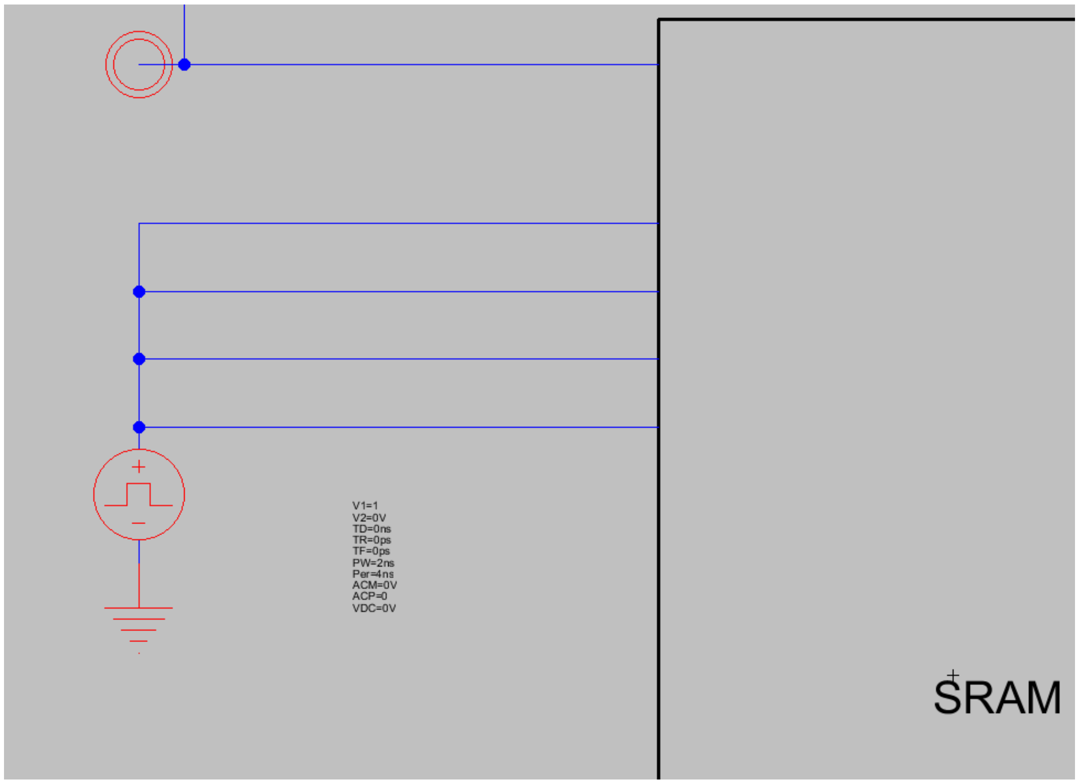




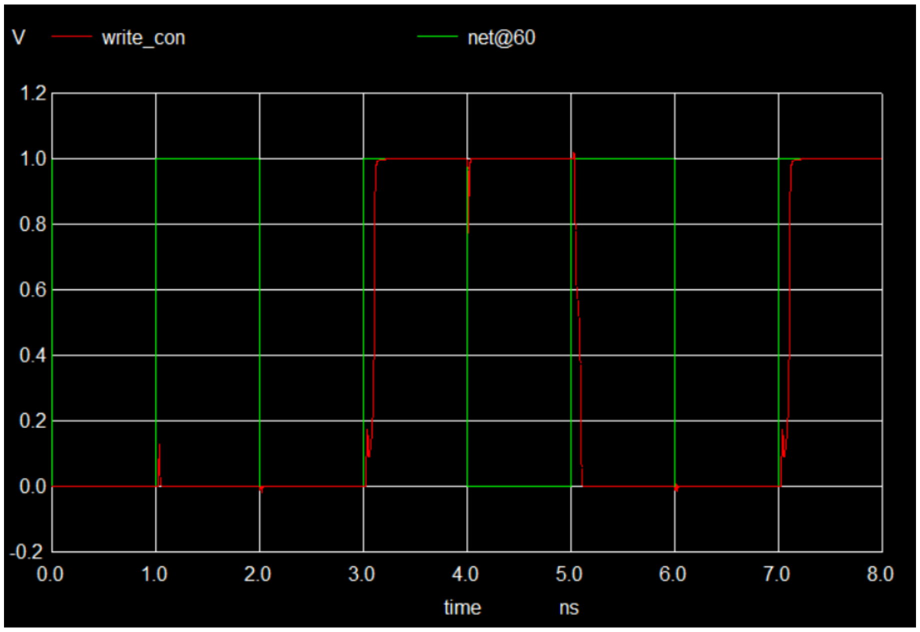
Here is a write con pin. This pin will tell us if the last cell in 1111 word has been written or not. Using it we will be able to measure the writing time.

The input: It will be all writing.

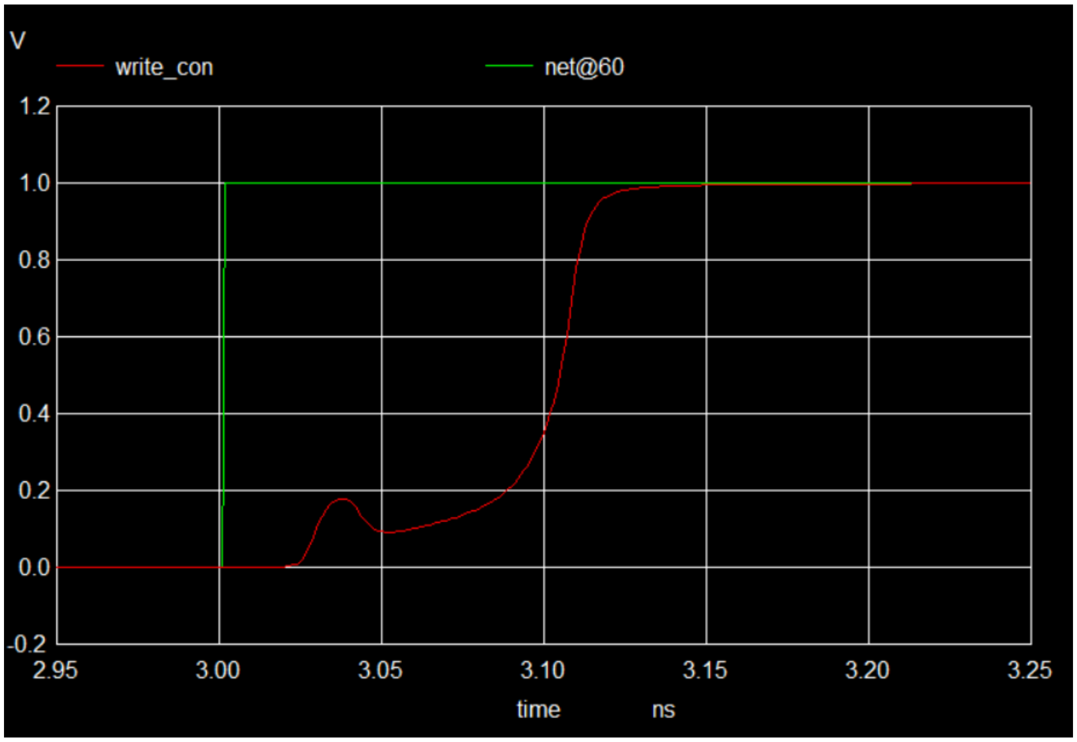
I will write 0 in the first cycle as an initialization. In the second cycle I will write 1 and measure the 0 to 1 writing cycle. Then the third cycle I will write 0 to measure the 1 to 0 writing cycle.



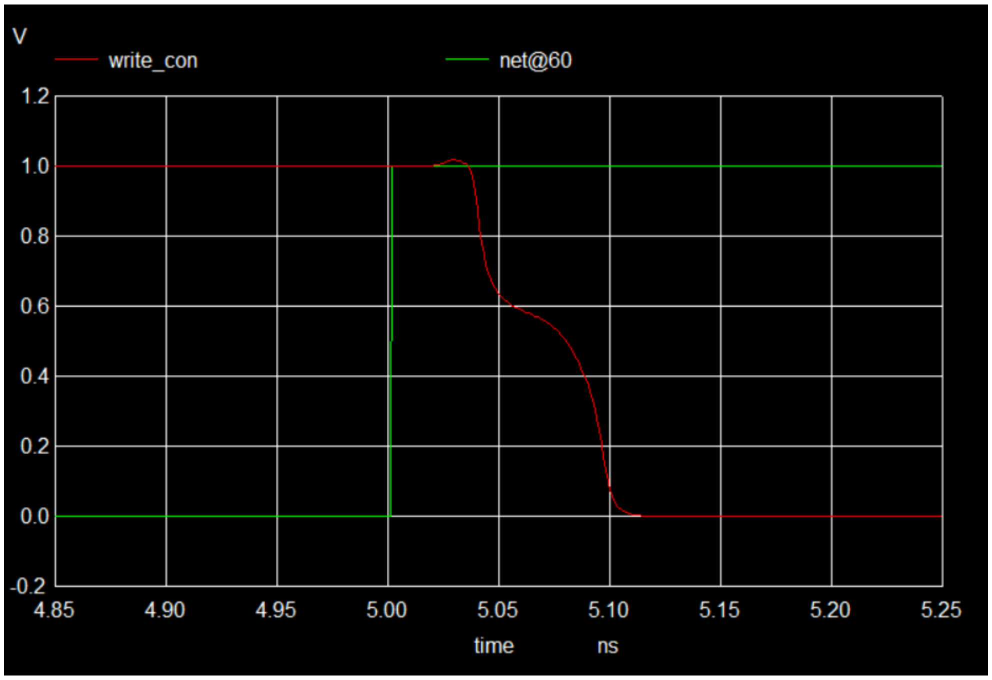
The output:



If we zoom in for the 0 to 1 transition: It take 0.13ns to rise (net@60 is clock)



If we zoom in for the 1 to 0 transition: it take 0.1ns

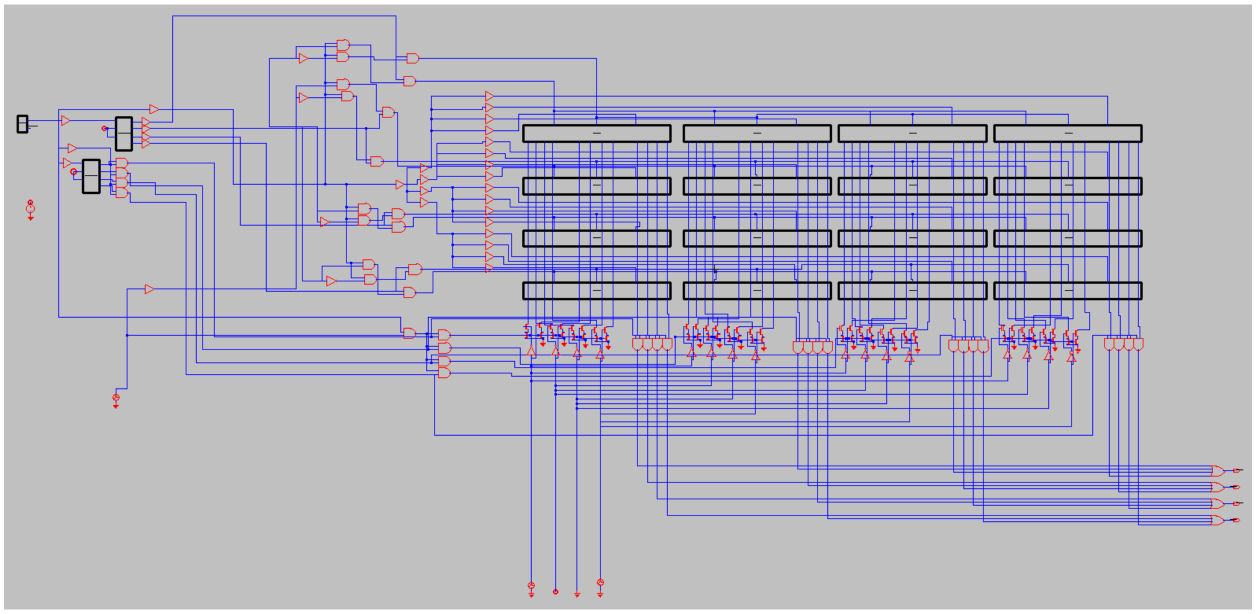


As a result our **delay is 0.13ns** (maximum of reading and writing))

4.2.3- Precharging delay

We precharge when the clock is low and evaluate when the clock is high. We found out that we need a high period of 0.13ns to write or read properly. Now we need to calculate the worst precharging time.

The circuit I will use:



Why I chose this version?

Because we need to precharge the following: Read line, write line and decoder.
Using this view I will be able to plot all those four signals and find the worst case of the precharging.

I will use a clock period of 2ns just to have space and be able to read properly.

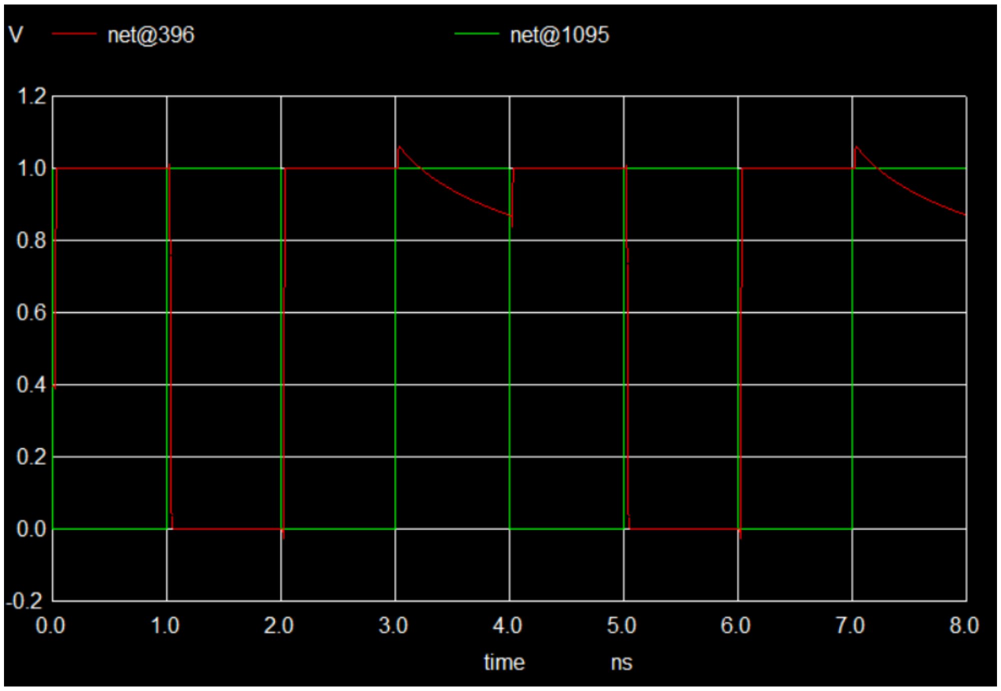
Since any bitline or read line is shared between all words vertically. I will write zeros on the 1111 word so that I drive the BL to zero then I will inspect how long it takes it to go to 1 in precharging. In the next cycle I will read the 0, so that I drive the read line to 0 and see how long it takes it to get charged.

Regarding the decoder, it won't matter the case because it will always be faster than the others because its load is very small (but I will measure it anyways).

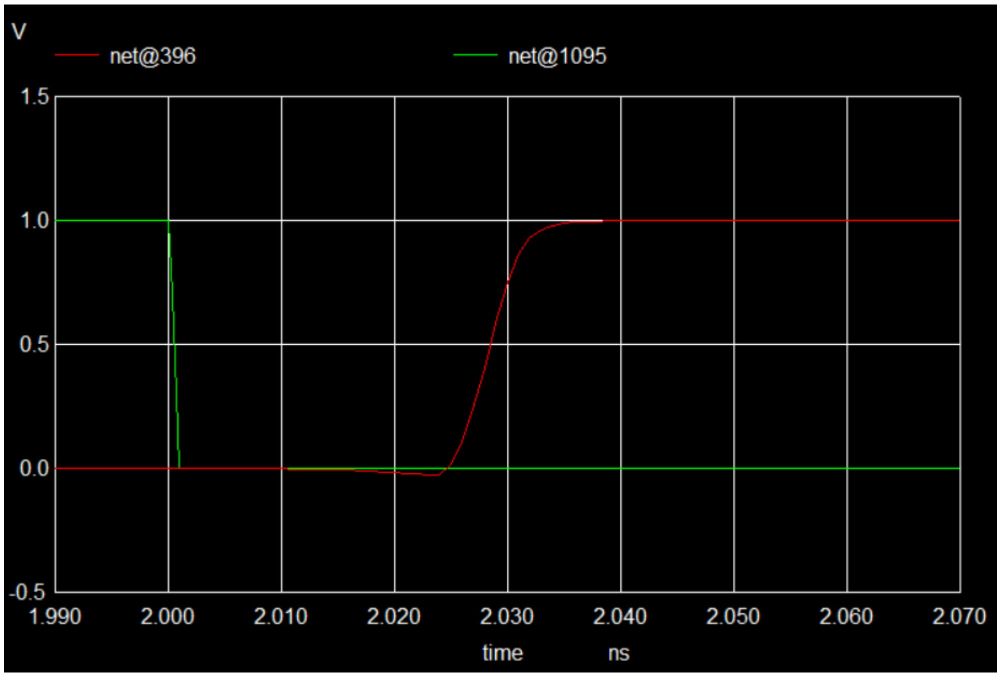
<u>The inputs:</u> Write in the first cycle, Read in the second cycle. I will write on 1111 address and I will write all zeros. The reason is above.

The output: net@1095 is the clock and net@396 is the bitline

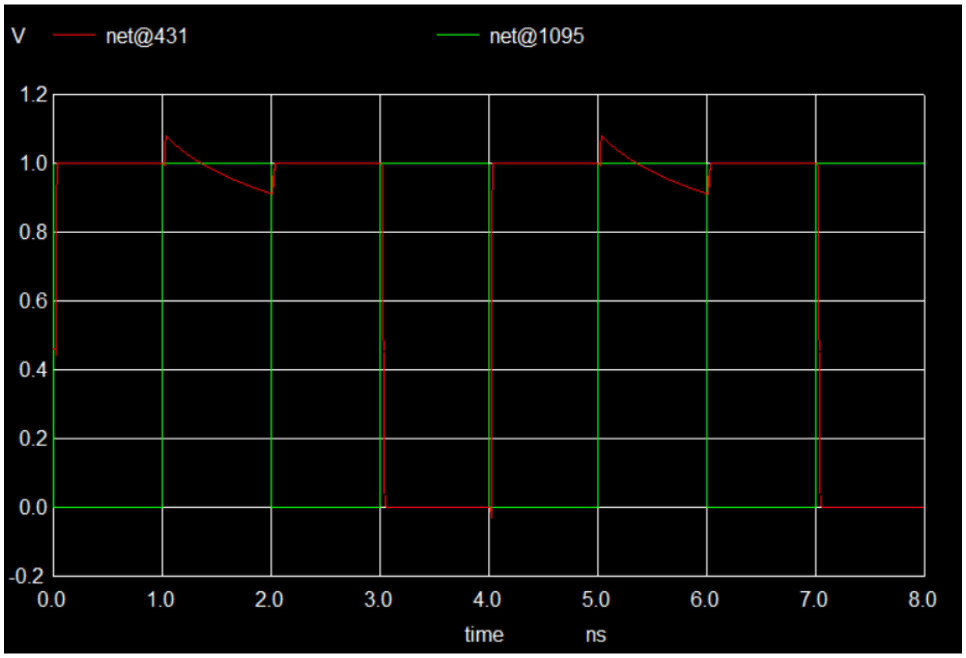
As we can see I performed a write operation in the second half of the first cycle to drive the bitline to zero. Then It is precharged. Lets zoom in



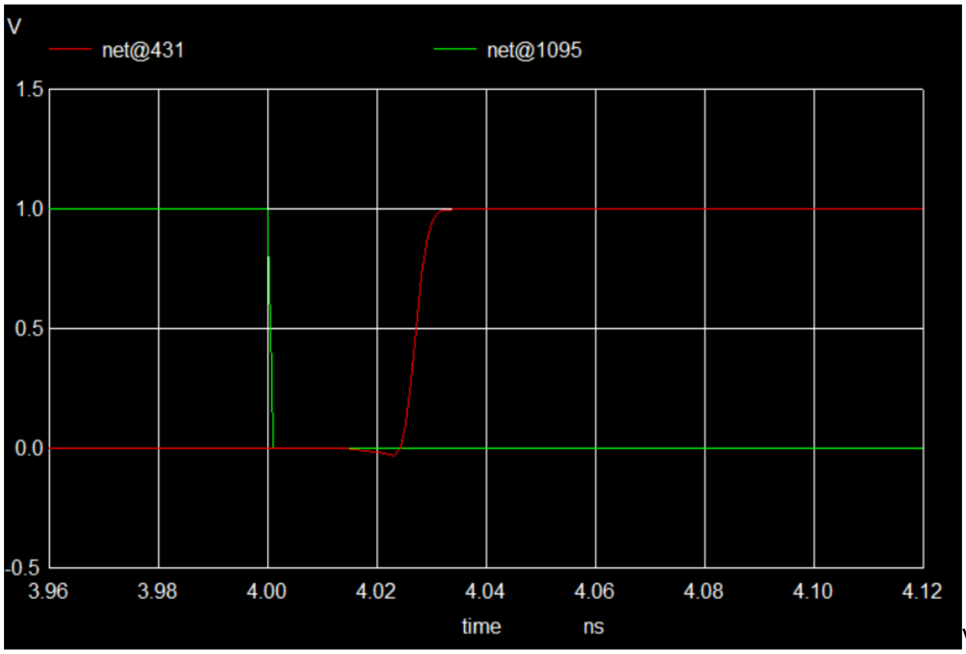
Precharging the bitline took 0.04ns.



Now regarding the readline. In the second half of the second cycle I did a reading operation. In the third cycle is is getting precharged. Lets zoom in. (clock is net@1095 and net@431 is read line)

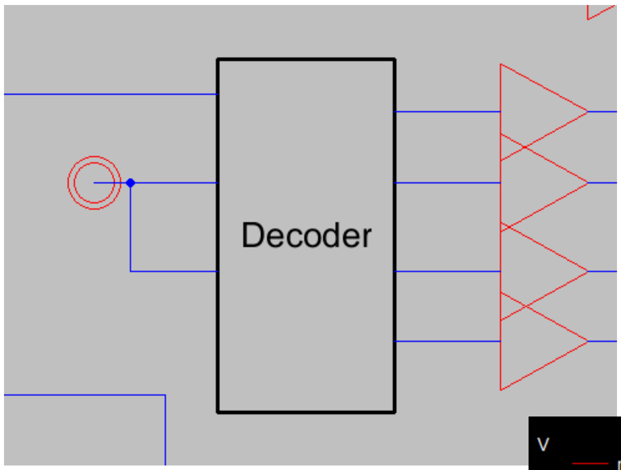


As we can see precharging nearly took 0.04ns also.

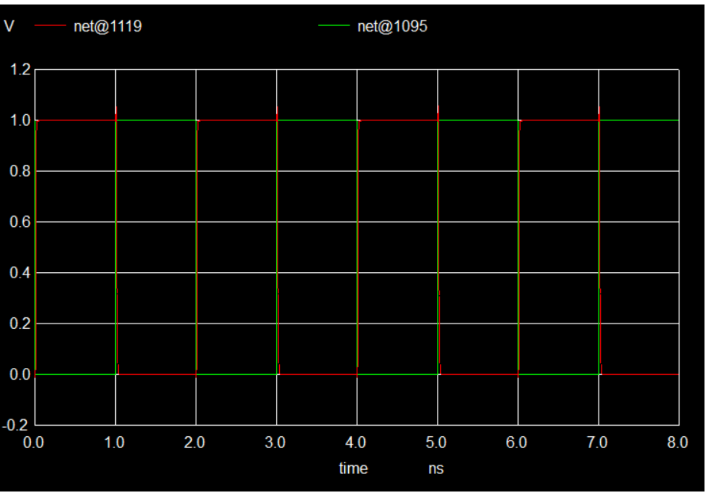


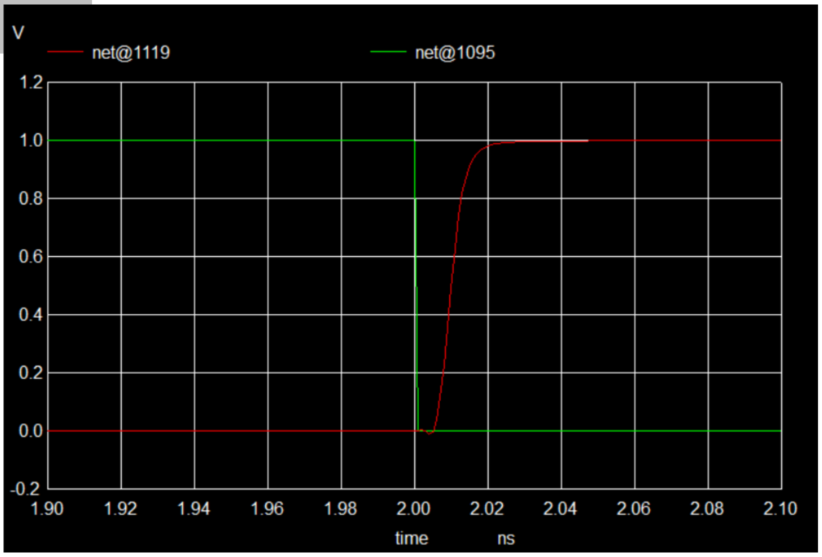
\

Now for the decoder.



I will measure the first signal 00 of the decoder since it will be driven to zero in the second half of the first cycle.





It also too 0.04ns

As a result, the delay for precharging is 0.04ns.

4.2.4- Total delay

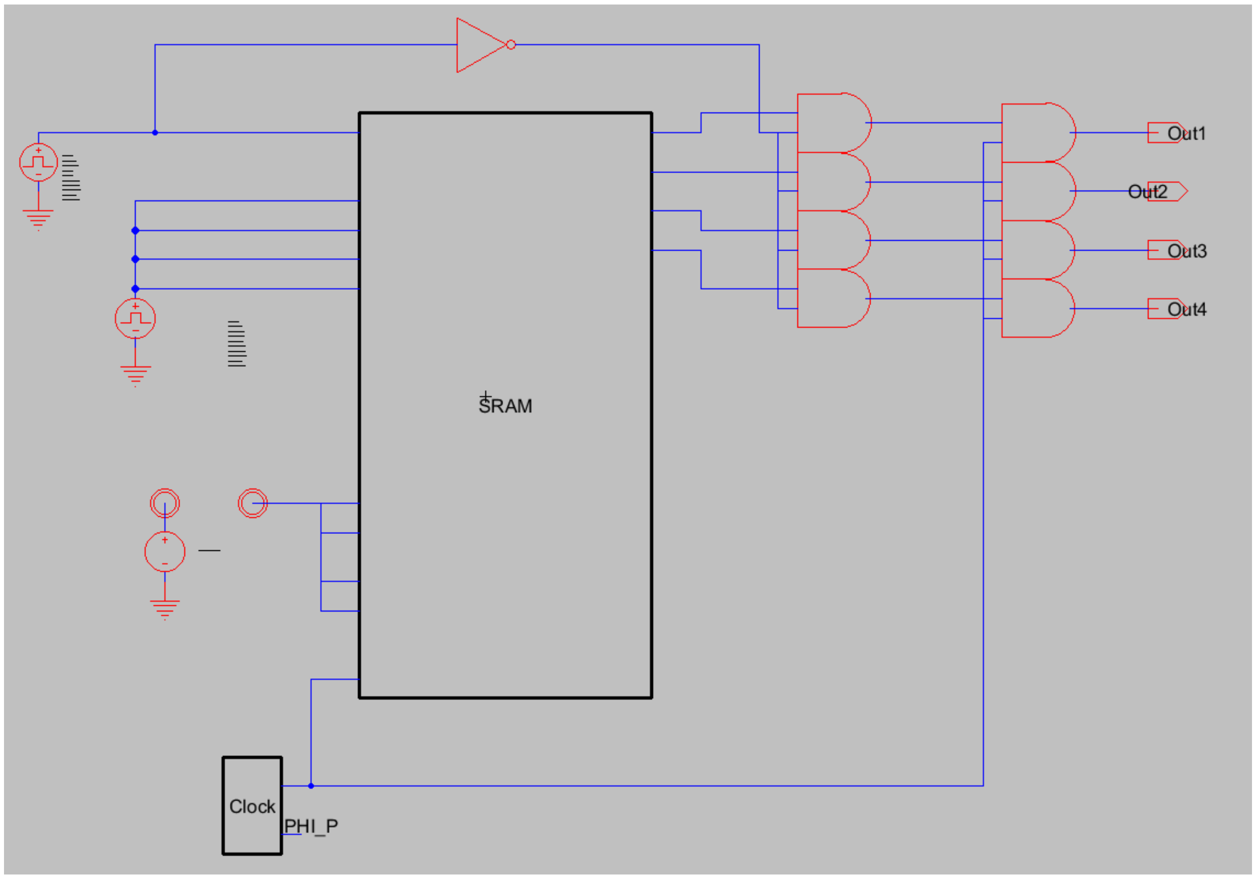
The precharging delay is 0.04ns and the maximum delay for writing/reading is 0.13ns. In total the **clock period is 0.17ns**

4.2.5- Testing the fastest speed of clock

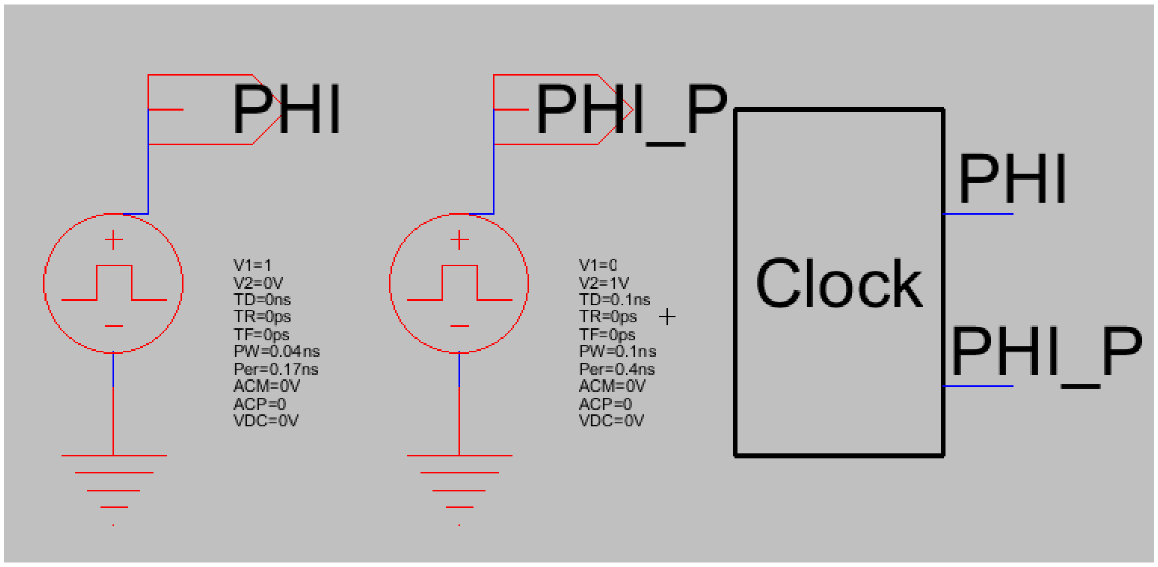
In this section, I will use the values I already got from the delay calculations to set the clock speed and see if it will work right or will there be any problems.

For this test, I will test writing this inputs on the 1111 word then 0000 word :1010 and 0110. The data inputs test changing inputs from 1->0 and 0->1 and 1->1 and 0->0. This will prove if the delay is calculated right or wrong because the values will switch in all permutations possible. Also, I will test on two words to make sure switching from a word to another is working right.

The circuit used: I will use this circuit because I just need to see the outputs and I don't care about the inner functionality. Here we treat it as a black box.



The clock: (Phi_B is useless I made it because I had another design in mind but I didn't remove it because I wanted to make all the report consistent and repeating each simulation just because I removed it wasn't realistic at all but I will remove it in power calc)

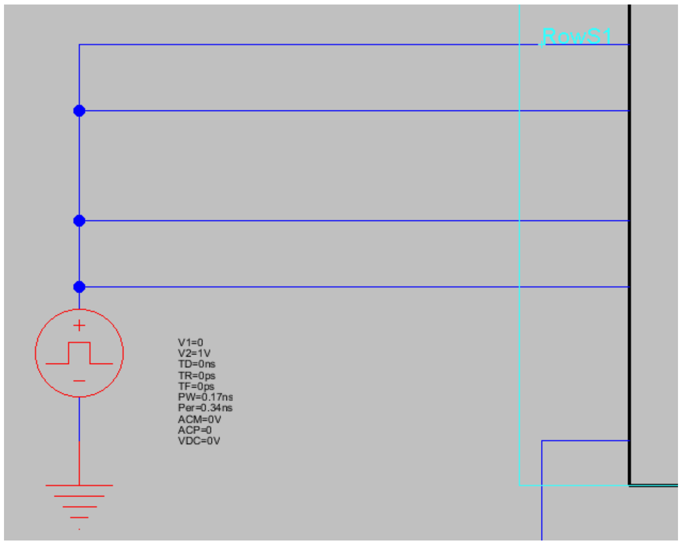


The inputs:

I will do a write 1010 on 1111 word then on 0000 word. After that I will read 1111 word then 0000 word. Then I will write 0110 on 1111 word then on 0000 word. After that I will read bot.

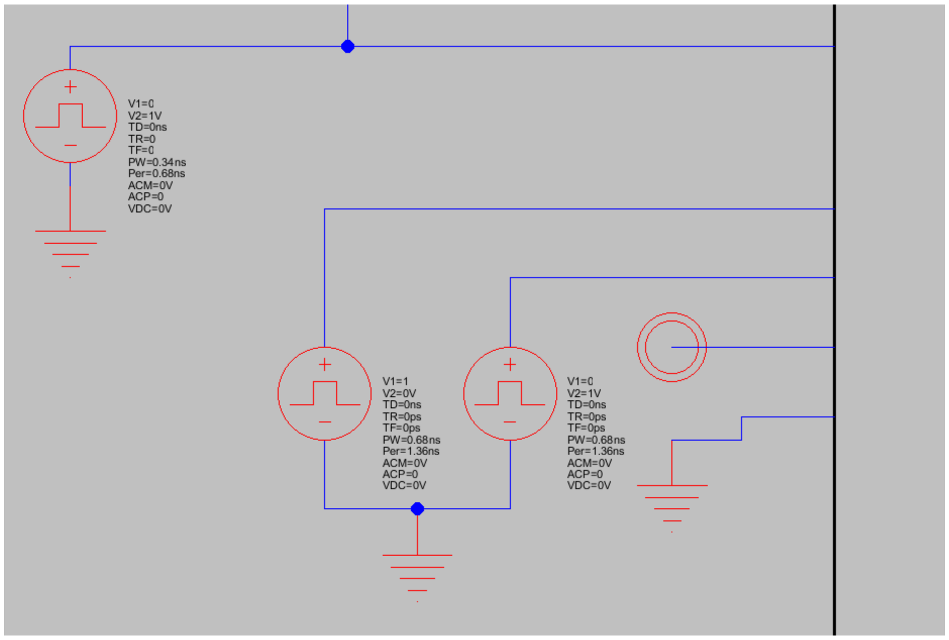
The number of cycles needed for this are 8.

The address:



```
V1=0
V2=1V
TD=0ns
TR=0ps
TF=0ps
PW=0.17ns
Per=0.34ns
ACM=0V
ACP=0
VDC=0V
```

The inputs:



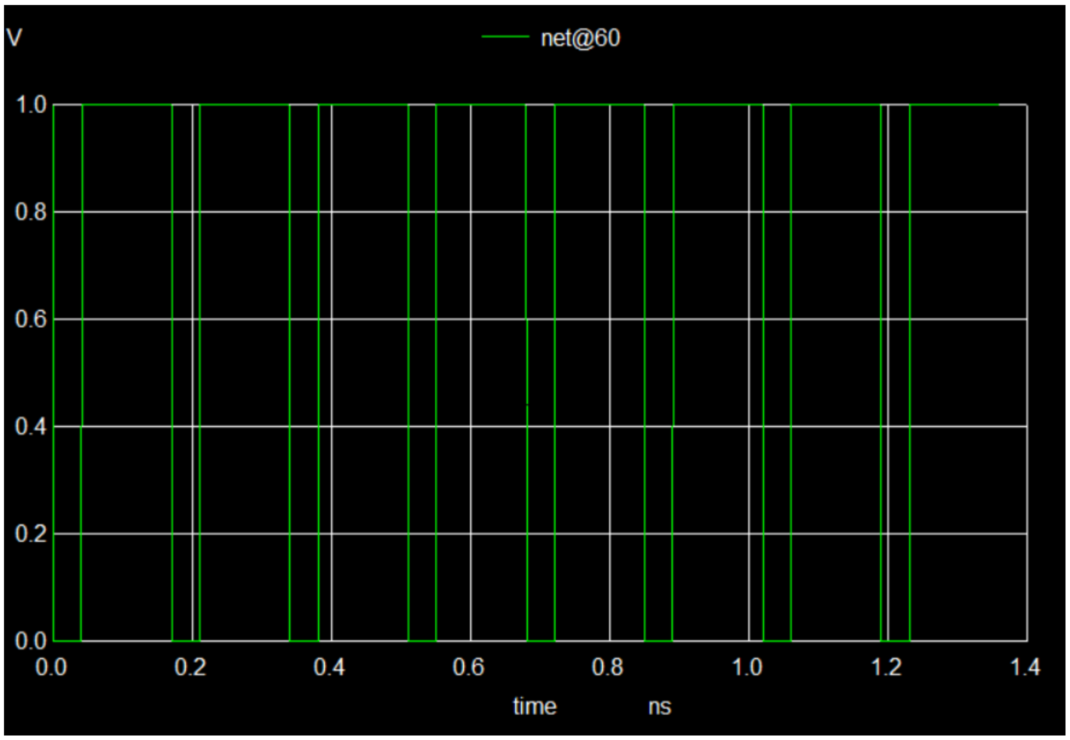
```
V1=0
V2=1V
TD=0ns
TR=0
TF=0
PW=0.34ns
Per=0.68ns
ACM=0V
ACP=0
VDC=0V
```

```
V1=1
V2=0V
TD=0ns
TR=0ps
TF=0ps
PW=0.68ns
Per=1.36ns
ACM=0V
ACP=0
VDC=0V
```

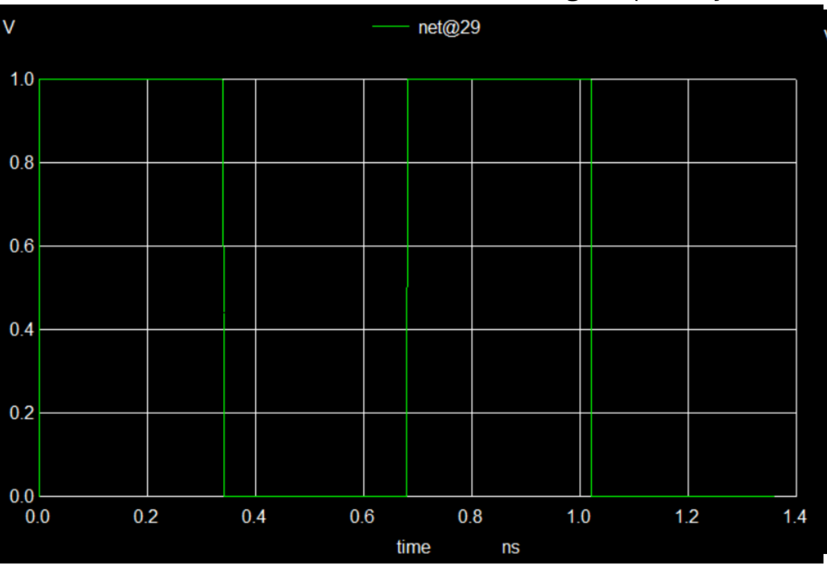
```
V1=0
V2=1V
TD=0ns
TR=0ps
TF=0ps
PW=0.68ns
Per=1.36ns
ACM=0V
ACP=0
VDC=0V
```

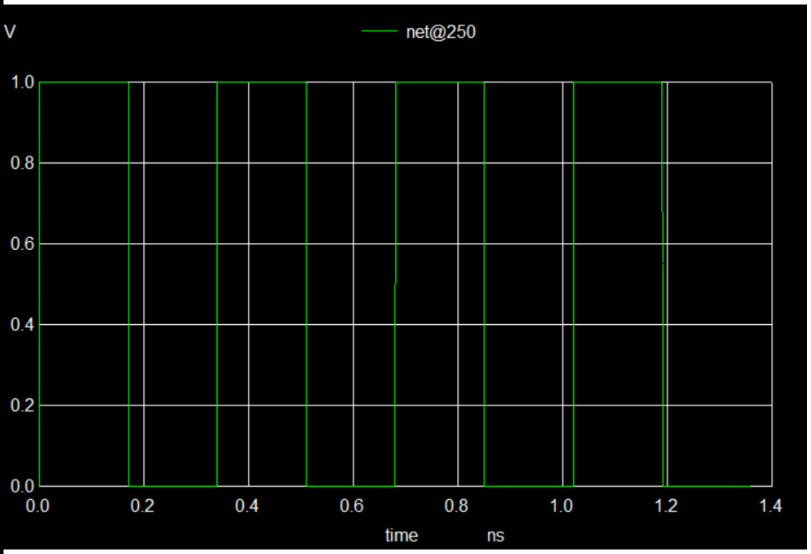
The outputs:

This is the clock (net@60)



net@29 is the write/read signal (two cycles of writing and two cycles of reading)



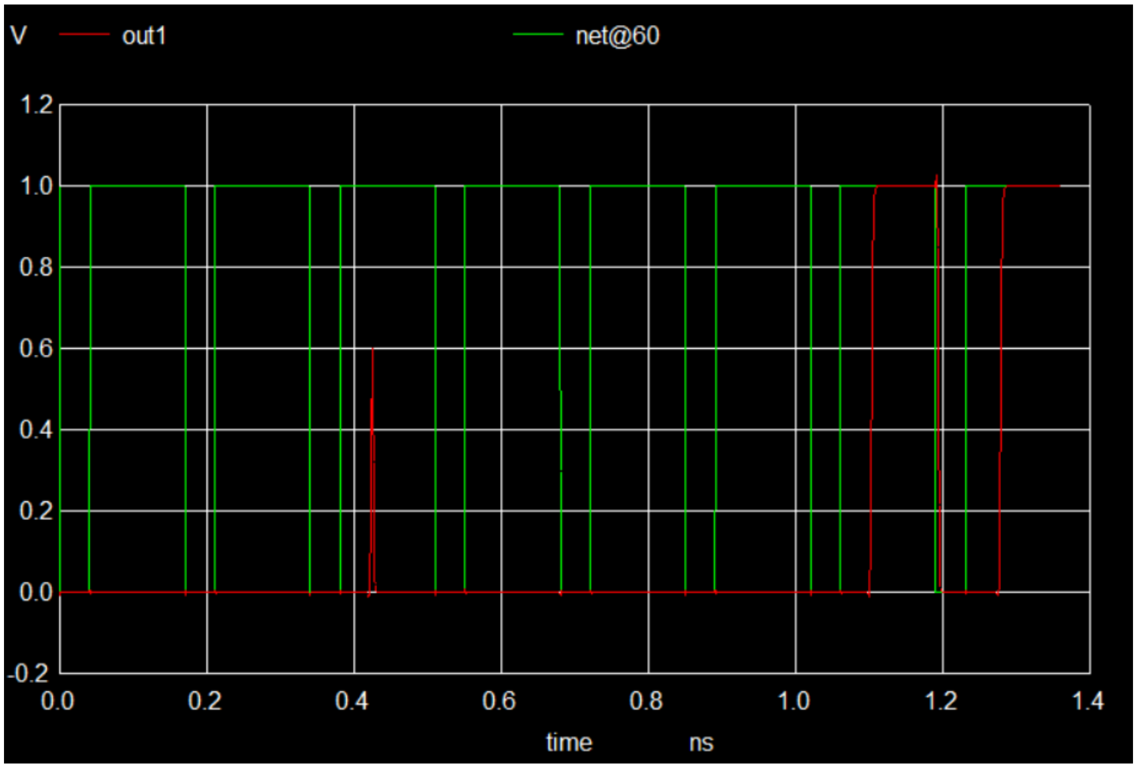


Net@250 is the address. It is one 1111 word for one cycle then 0000 for the other cycle.

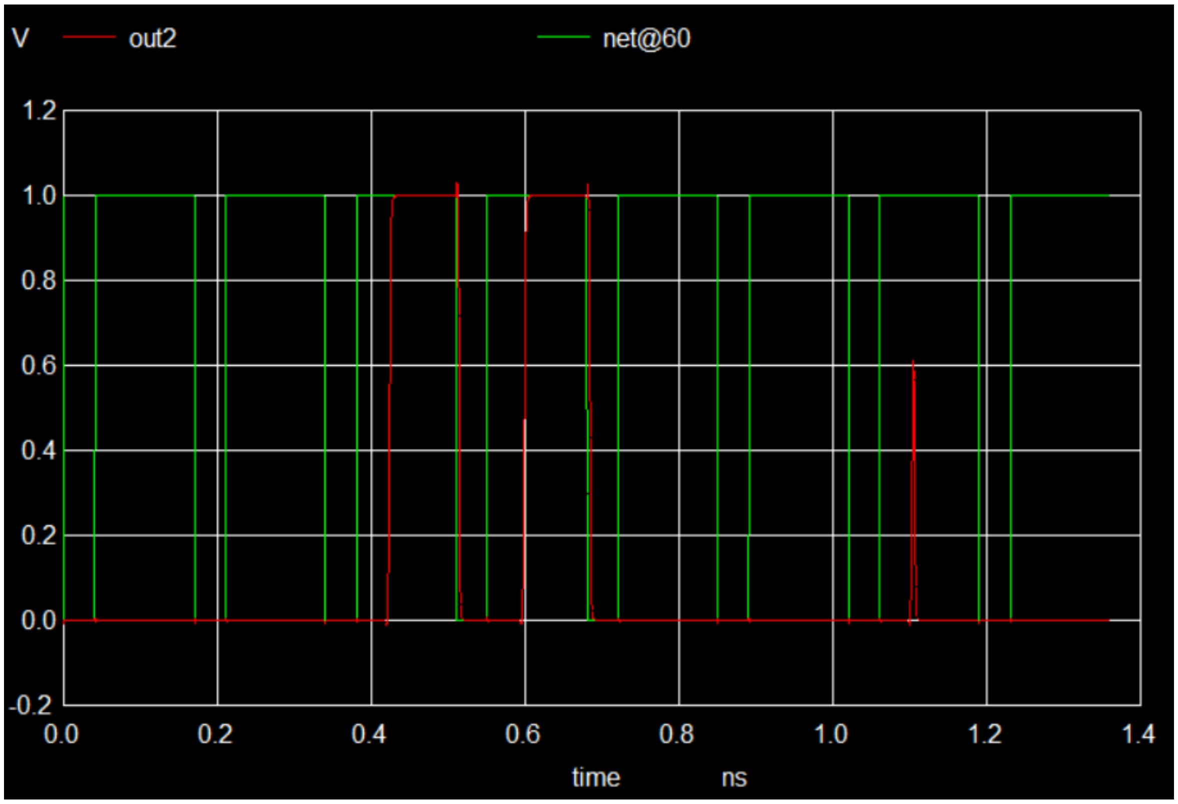
Now lets take a look at the output signal for each output while comparing it to the clock.

It is expected for out1 to be two zeros in the third and fourth cycle. Then two ones at seventh and eighth cycle. Out2 is contrast of out1 and out3 is all ones and out4 is all zeros.

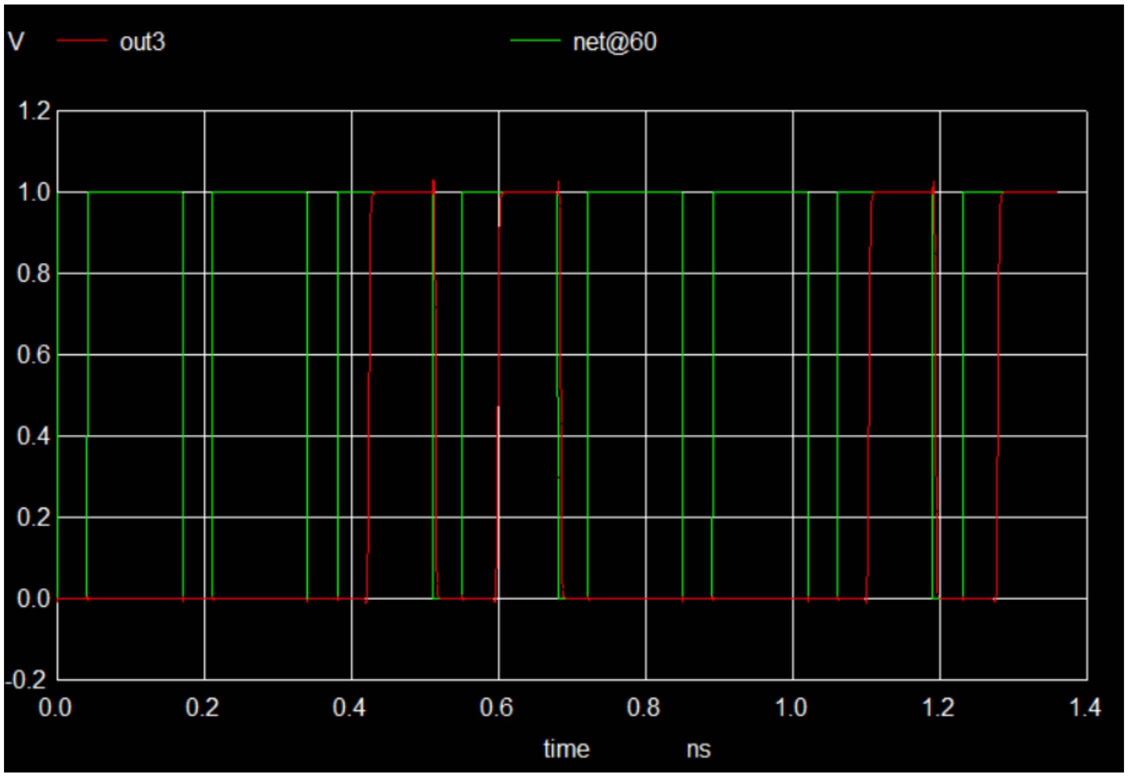
Out1:


V
out1
net@60
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
time ns

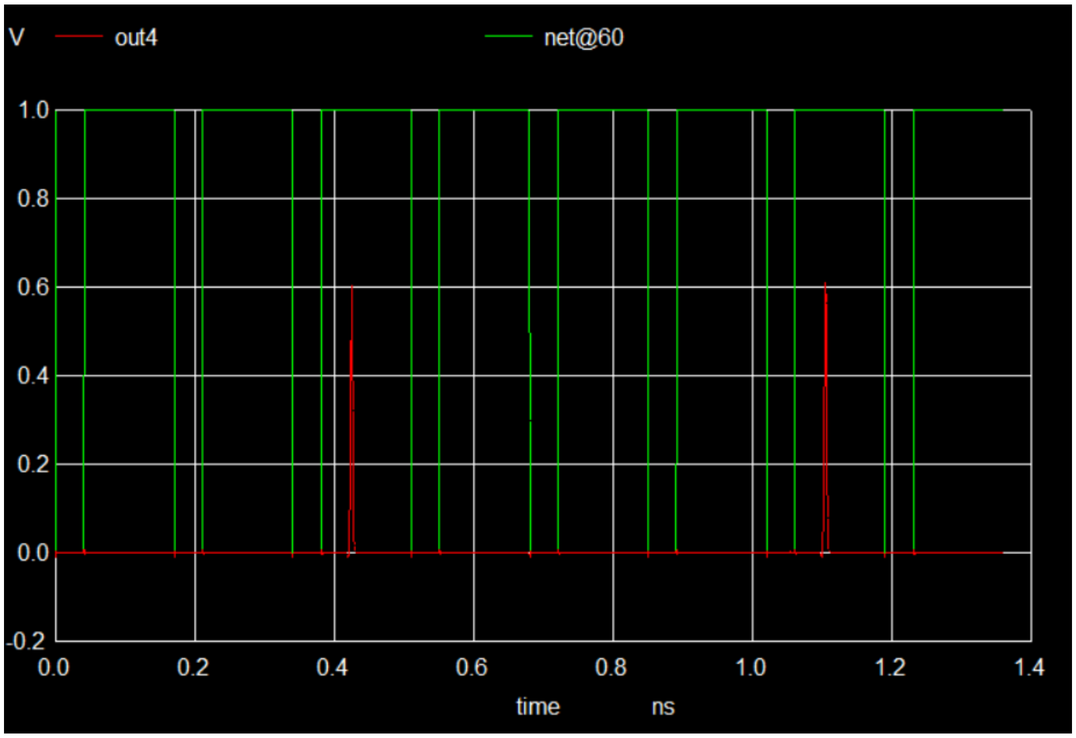
Out2:


V
out2
net@60
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
time ns

Out3:


V
out3
net@60
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
time
ns

Out4:


V
out4
net@60
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
time
ns

As we can see all the outputs are right. There is a small spike when we read zero of 1111 word but this is normal due to the speed of the circuit and it quickly settle so there is no problem.

Fastest clock frequency is 5.8KHz **Fastest Period is 0.17ns**

4.3- Power

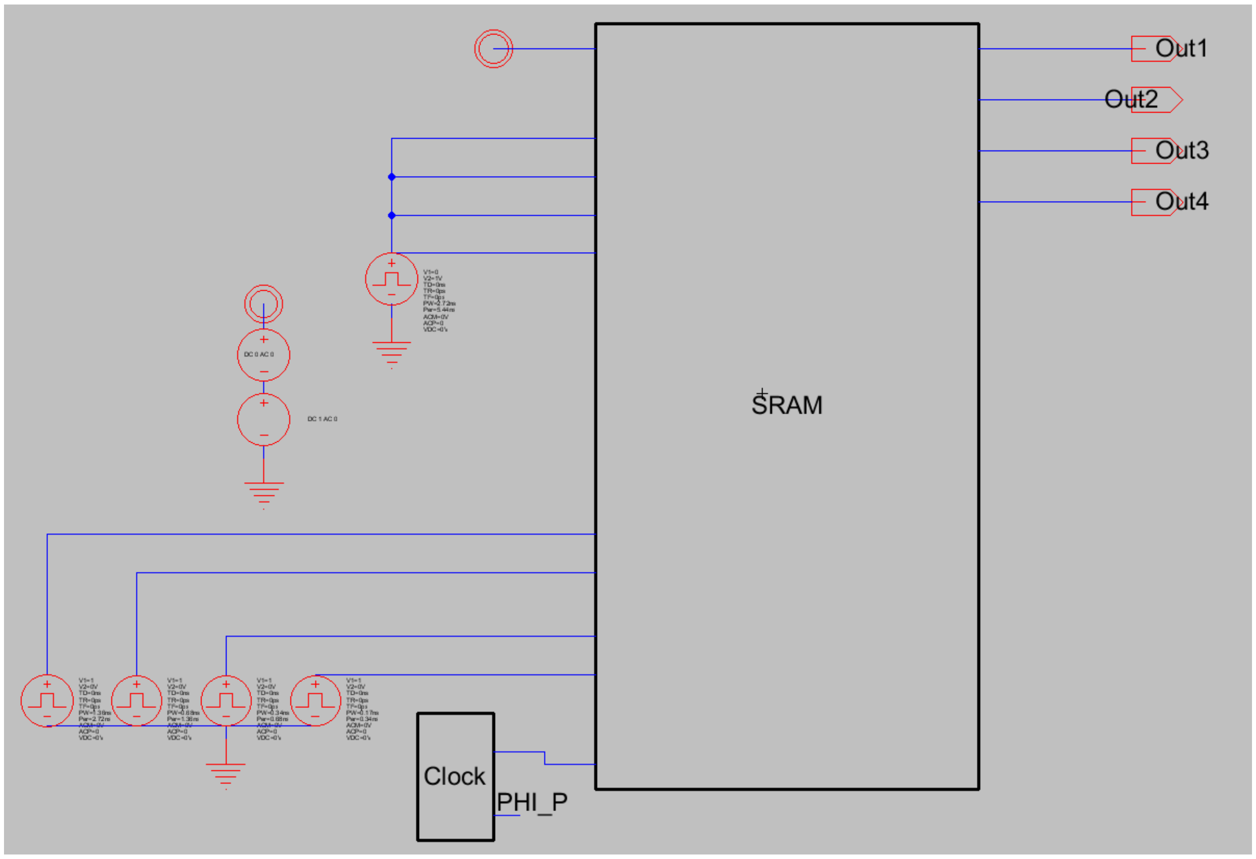
In this circuit, I need to write 0000 on all 16 words then write 1111 on all words 4 times in a row. This operation needs 32*4 clock cycles = 32*4*0.17ns = 21.76ns

For the first 16 cycles the input is 0000 for the second 16 cycles the input is 1111 then repeat.

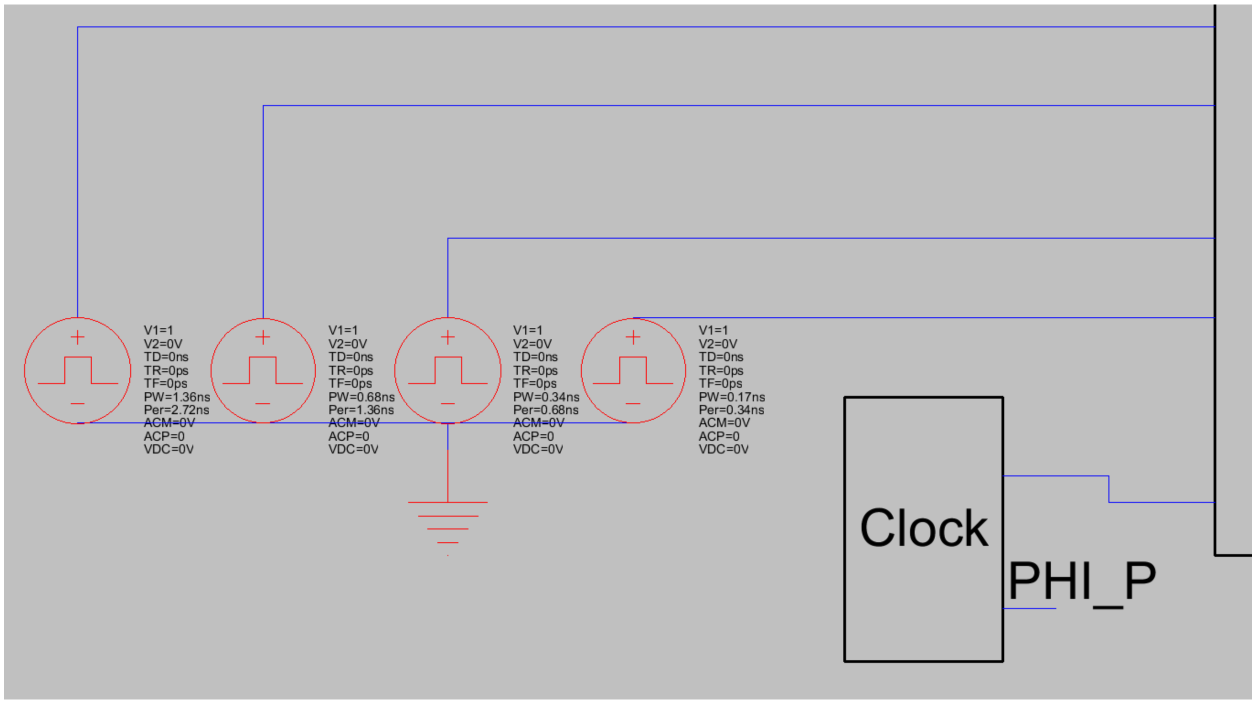
For Write/Read signal it will be always 1 since we are always writing.

For the address it will increase as a binary number 0000->0001->0010->0011-> etc...

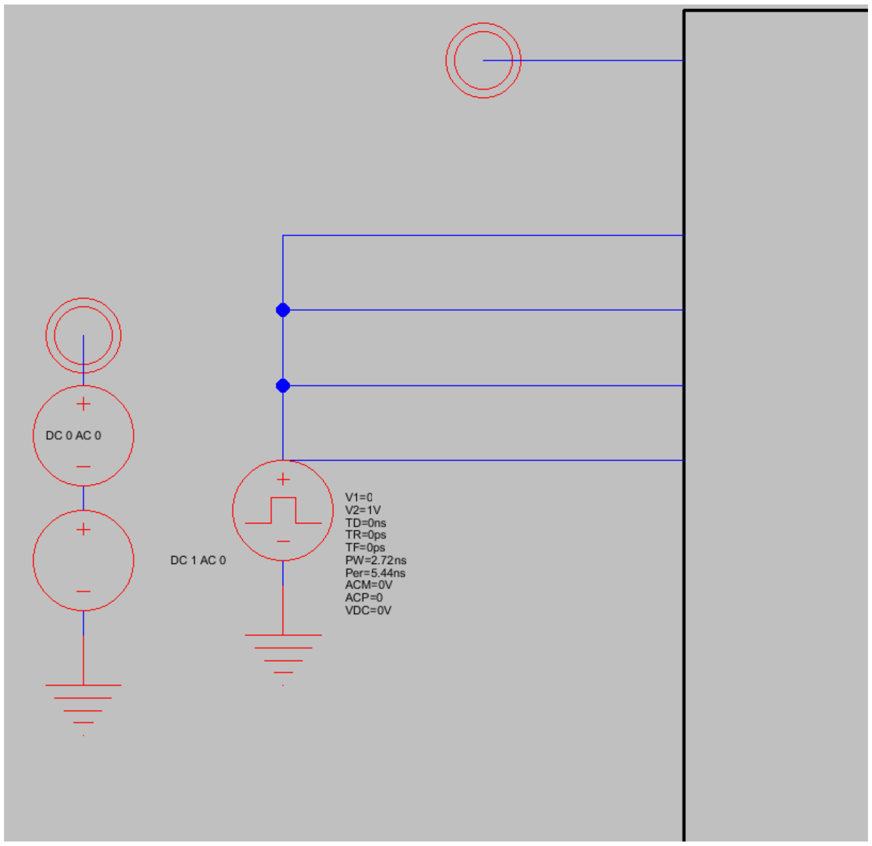
The circuit used:



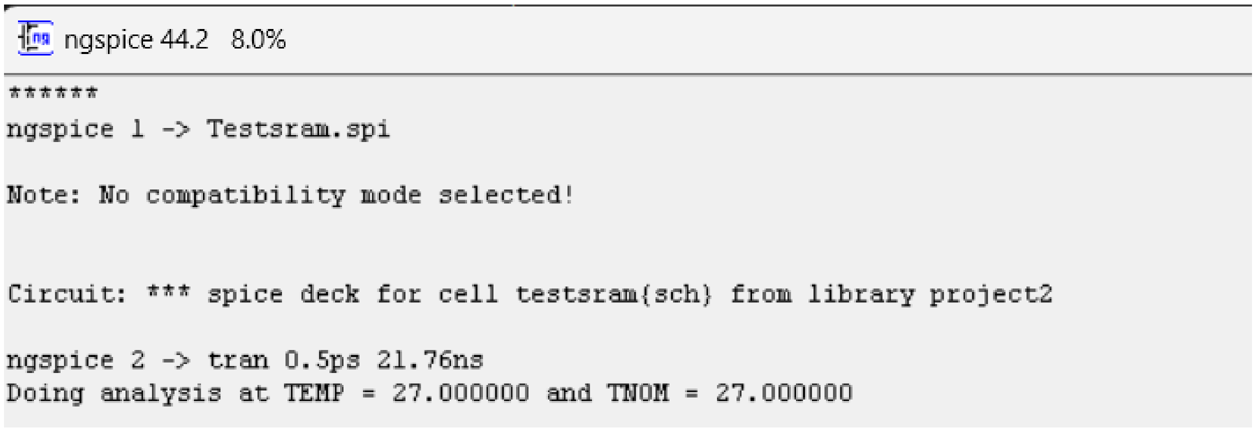
The address:



The inputs:



As we can see one cycle of the inputs is 5.44ns. Half of it is 0000 for 2.72ns and the other half is 1111 for another 2.72ns. This should be repeated 4 times.



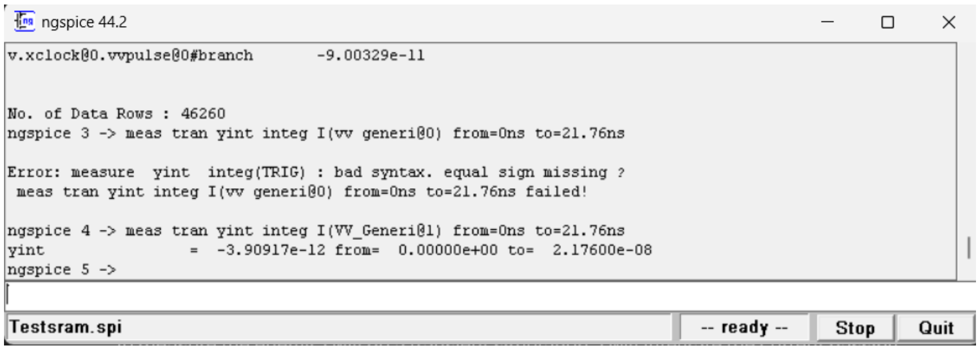
```
ngspice 44.2   8.0%
******
ngspice 1 -> Testsram.spi

Note: No compatibility mode selected!


Circuit: *** spice deck for cell testsram{sch} from library project2

ngspice 2 -> tran 0.5ps 21.76ns
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000
```

To measure the energy, I will do a transient simulation. I will integrate the current passing through the VV_Generi@1 battery. After that I will divide this current by 21.76ns to get the average current per second. Then multiply it to the voltage, which is 1V. This way we has the power of the circuit.



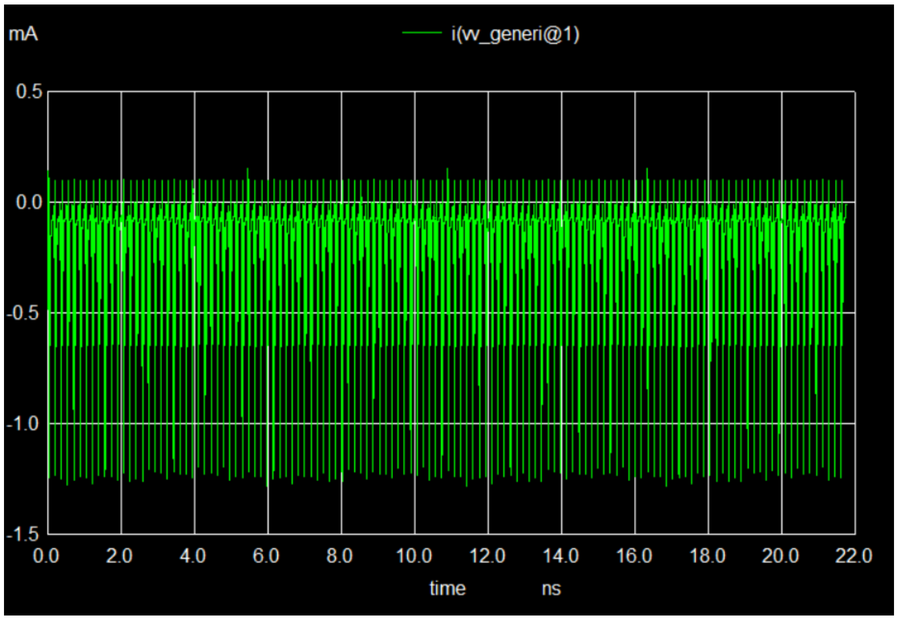
```
ngspice 44.2
v.xclock@0.vvpulse@0#branch       -9.00329e-11


No. of Data Rows : 46260
ngspice 3 -> meas tran yint integ I(vv generi@0) from=0ns to=21.76ns

Error: measure  yint  integ(TRIG) : bad syntax. equal sign missing ?
 meas tran yint integ I(vv generi@0) from=0ns to=21.76ns failed!

ngspice 4 -> meas tran yint integ I(VV_Generi@1) from=0ns to=21.76ns
yint                =  -3.90917e-12 from=  0.00000e+00 to=  2.17600e-08
ngspice 5 ->

Testsram.spi                                 -- ready --   Stop   Quit
```



As we can see the integration of the current is 3.9*10^(-12)A

The average of the current is 3.9*10^(-12)/(21.76* 10^(-9)) = 0.179mA

Power = 0.179mW

4.4- FOM calculation

FOM calculation = 60 ∗ BitcellArea · Power · Delay^2 .

Bitcellarea	10
Delay (Shortest CLK period)	0.17ns
Power	0.179mW
FOM	3.1*10^(-21)

5- Certification:

I, Mohamed Elsheshtawy, certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this project.