Stephen More

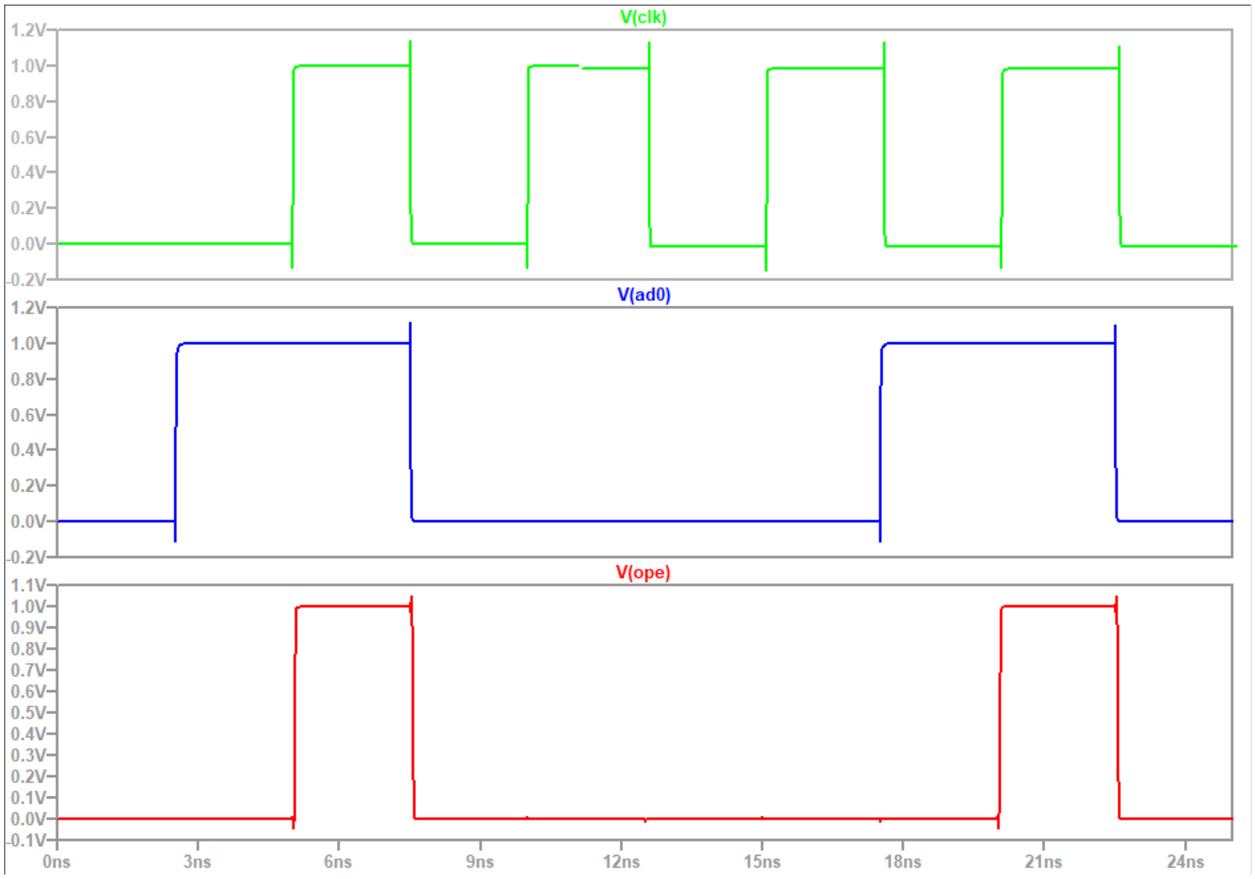
ECE 471 Final Project SRAM

SRAM Functional Block Walkthrough:

Note: Electric wasn't letting me change transistor sizes on diagram so schematic widths are not to scale

A. Decoder

testDecoder.cir output:



decModel subckt:

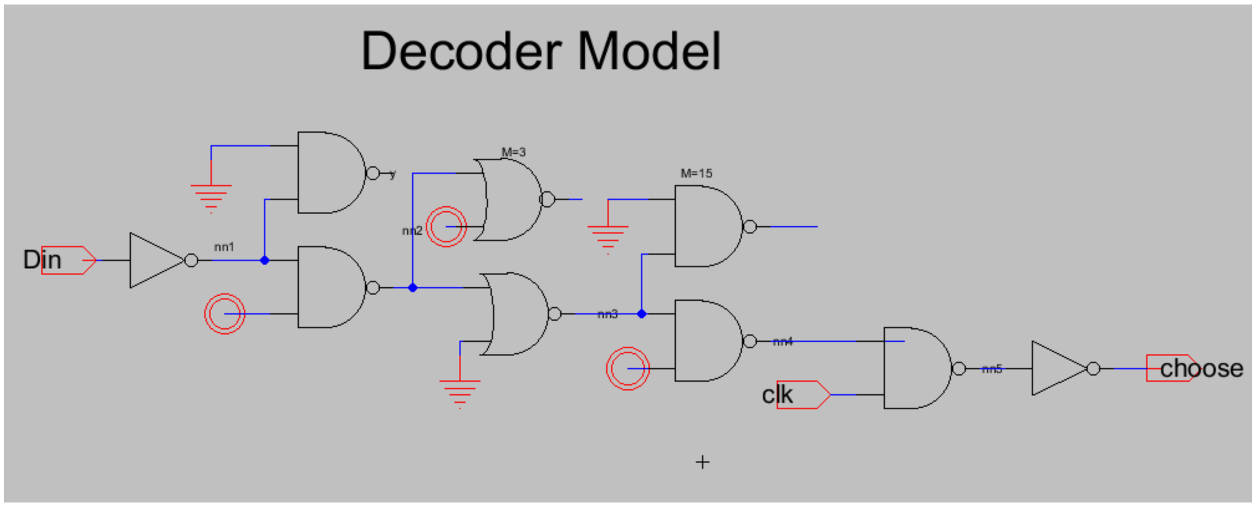
```
.subckt decModel cho din clk size=100 beta=2
Xin1 nn1 din     inv
Xpt2 nn2 nn1 vdd nnd2
Xdu2 dd1 nn1 gnd nnd2
Xpt3 nn3 nn2 gnd nor2
Xdu3 dd2 nn2 vdd nor2 size='3*size'
Xpt4 nn4 nn3 vdd nnd2
Xdu4 dd3 nn3 gnd nnd2 size='15*size'
*Xin5 nn5 nn4     inv
*commented out above inv because it buffered my input to be behind the example waveform
Xpt5 nn5 nn4 clk nnd2
Xin6 cho nn5     inv
.ends decModel
```

Decoder Description:

The decoder will have 8 inputs and will select a single address from 256 possibilities on the output. The decoder is for selecting the correct SRAM row to write to. The output will go to an access pin on a group of 8 SRAM cells representing a byte. The decoder model for the schematic below shows the worst possible case in which the input is forced to go through the maximum amount of logic, creating the maximum possible delay for the decoding of a single 8-bit address. This means that if the design can perform under worst conditions, all faster conditions will also perform correctly.

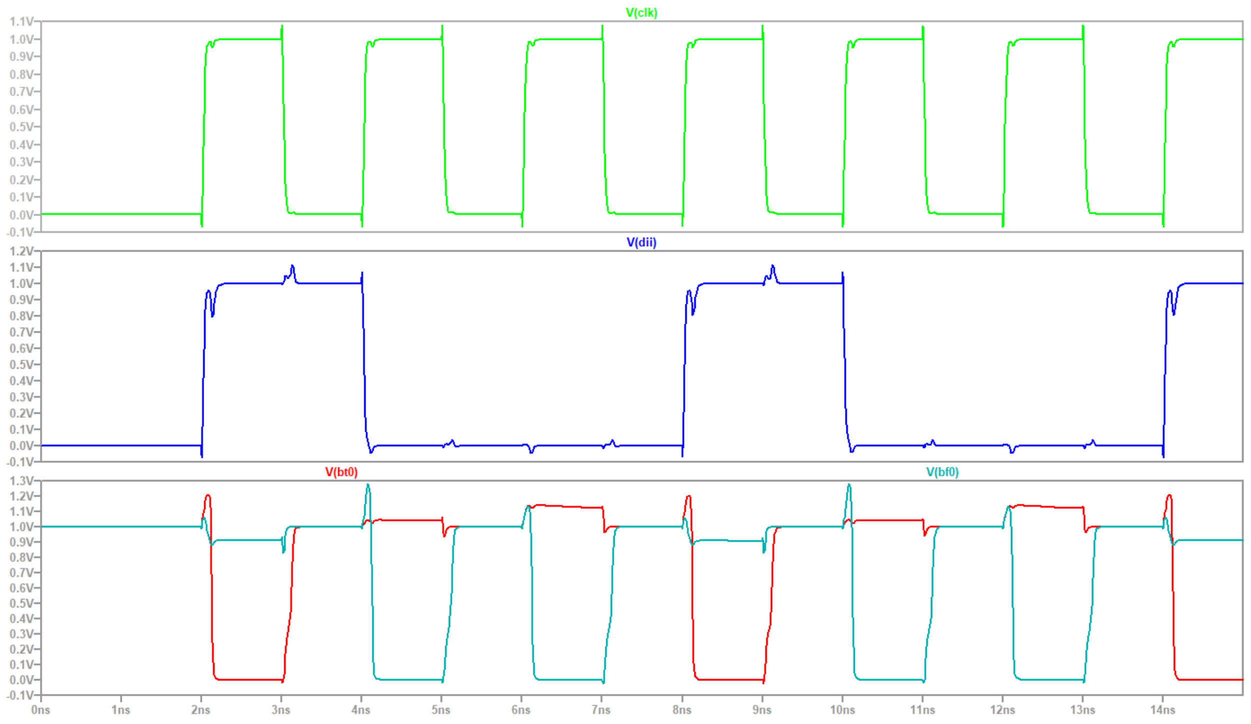
The way in which my decoder model deviates from the presented version is the removal of an inverter between “nn4” and the input of the nand2 outputting to “nn5”. The reason I removed this block is because while testing I was able to output a very similar waveform to the reference, however, it was shifted to the right, signaling to me that there was an extra stage that was not needed. From there I used trial and error to find the right inverter to remove. I kept my sizing consistent from the in-class reference.

Schematic:



B. Write Block

testWrite.cir output:



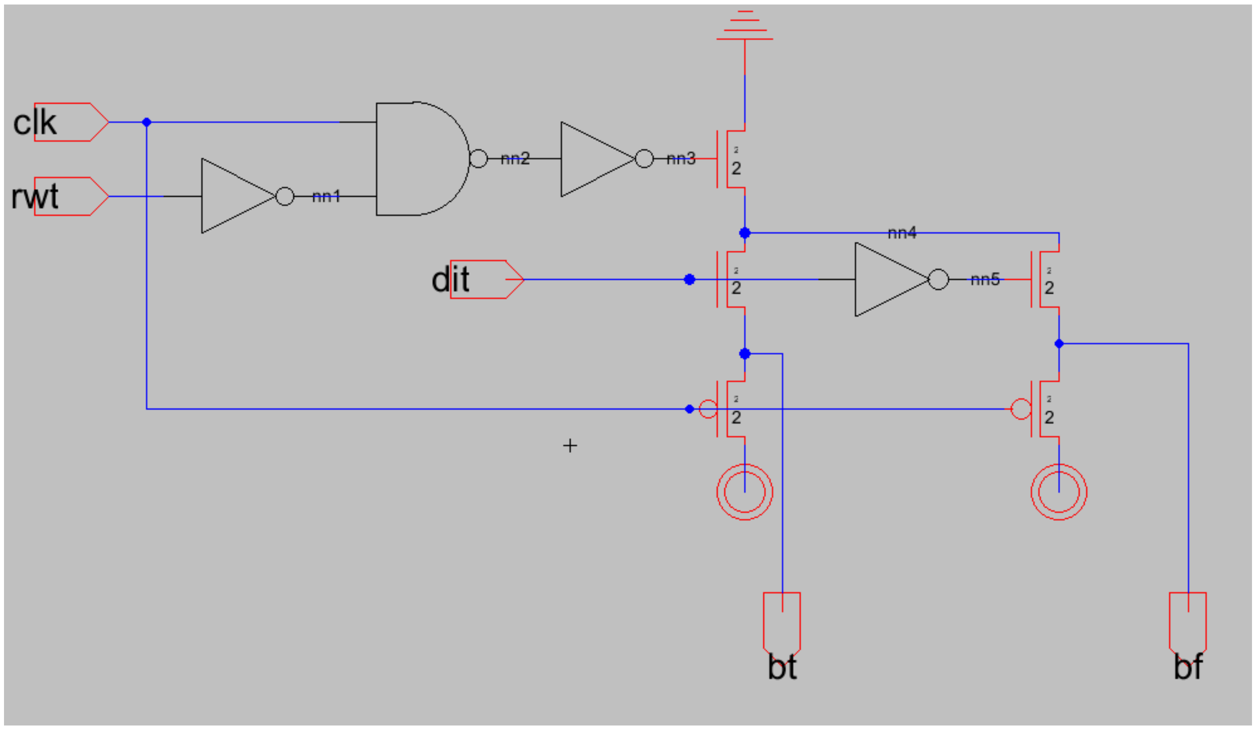
Write1 subckt:

```
.subckt write1 btt bff dit rwt clk size=30 beta=2
Xin1 nn1 rwt     inv
Xnd2 nn2 nn1 clk nnd2
Xin2 nn3 nn2     inv
Xnn2 nn4 nn3 gnd nn
Xnn3 btt dit nn4 nn
Xin3 nn5 dit     inv
Xnn4 bff nn5 nn4 nn
Xpp4 btt clk vdd pp
Xpp5 bff clk vdd pp
.ends write1
```

Write Desc:

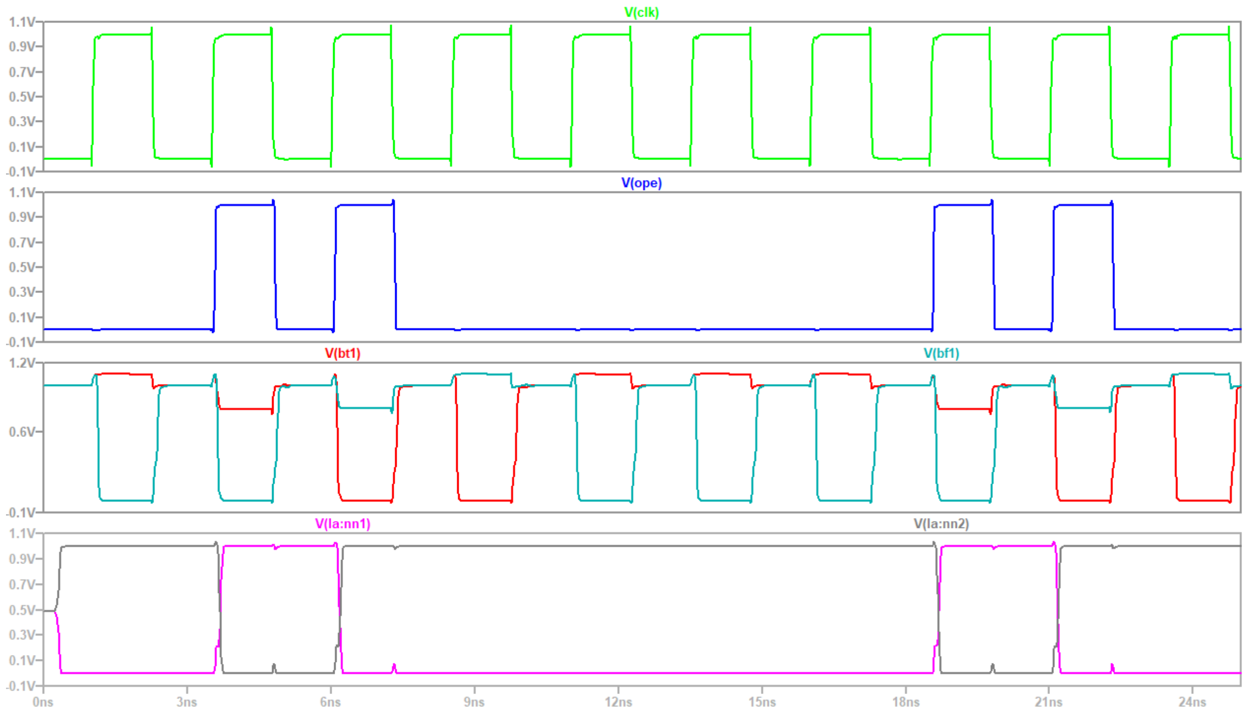
The writeline takes in input ports clk,rwt, and dit, and outputs low and high signals to the SRAM bitlines, bt and bf. If rwt is high, the write block is off, and if rwt is low the bitlines can be written to. When rwt is high, the write circuit provides high impedance to both bt and bf which allows the sense amp in the read to be the only circuit pulling from the lines. When rwt is low, the dit input line is able to pull bt low or high, giving it the desired value, and the inverter then provides the inverse to the bf.

I used the same default sizing provided by reference, I would have normally changed the pp to be double the width of nn but since it worked from the first try I didn't change anything.


clk
rwt
nn1
nn2
nn3
nn4
dit
nn5
2
2
2
2
2
2
bt
bf

C. Memory Cell

testMem.cir output:



Mem1 subckt:

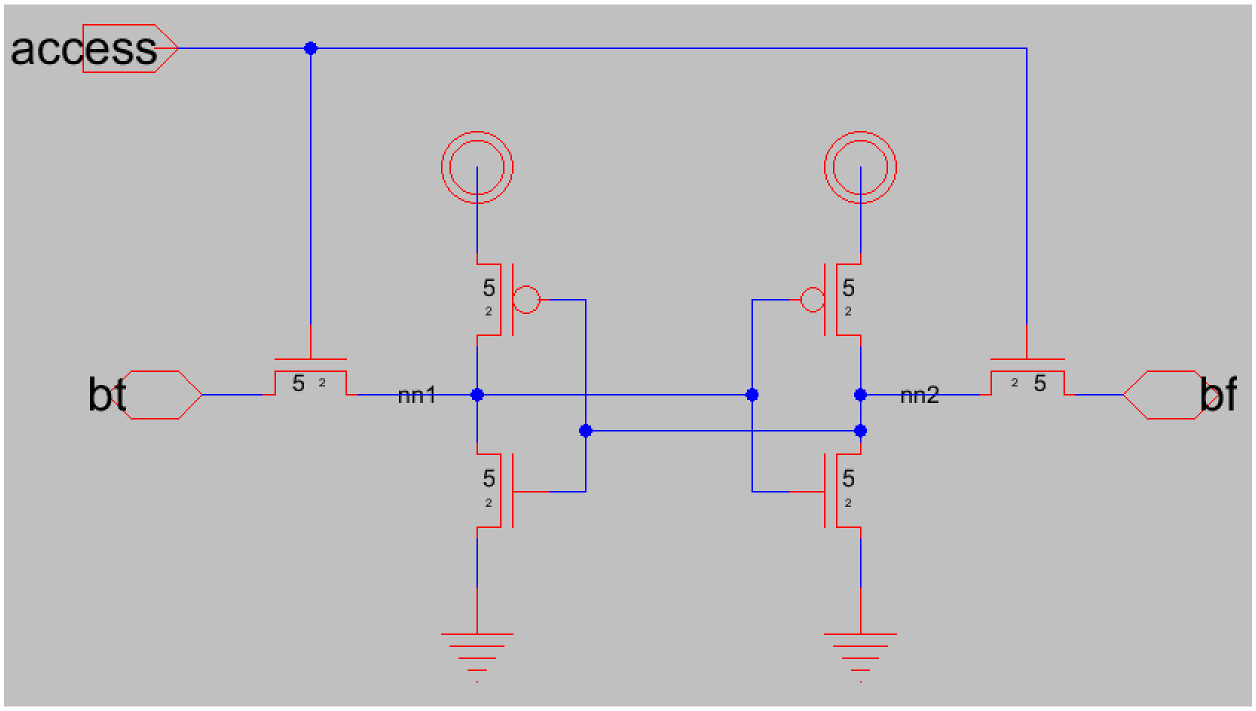
```
.subckt mem1 btt bff ope wid=5
Xnn1 btt ope nn1 nn ww='wid'
Xnn2 bff ope nn2 nn ww='wid'
Xpp3 vdd nn2 nn1 pp ww='2*wid'
Xnn3 nn1 nn2 gnd nn ww='wid'
Xpp4 vdd nn1 nn2 pp ww='2*wid'
Xnn4 nn2 nn1 gnd nn ww='wid'
.end mem1
```

Memory Cell Description:

The memory cell is the fundamental block of the SRAM which holds data. A single cell holds 1 bit. A cell has access, bt and bf ports. If access is high, the internal nodes are exposed to the bit lines bf and bt. The port bt is the value which is desired to put into the cell and bf is the inverse of that value.

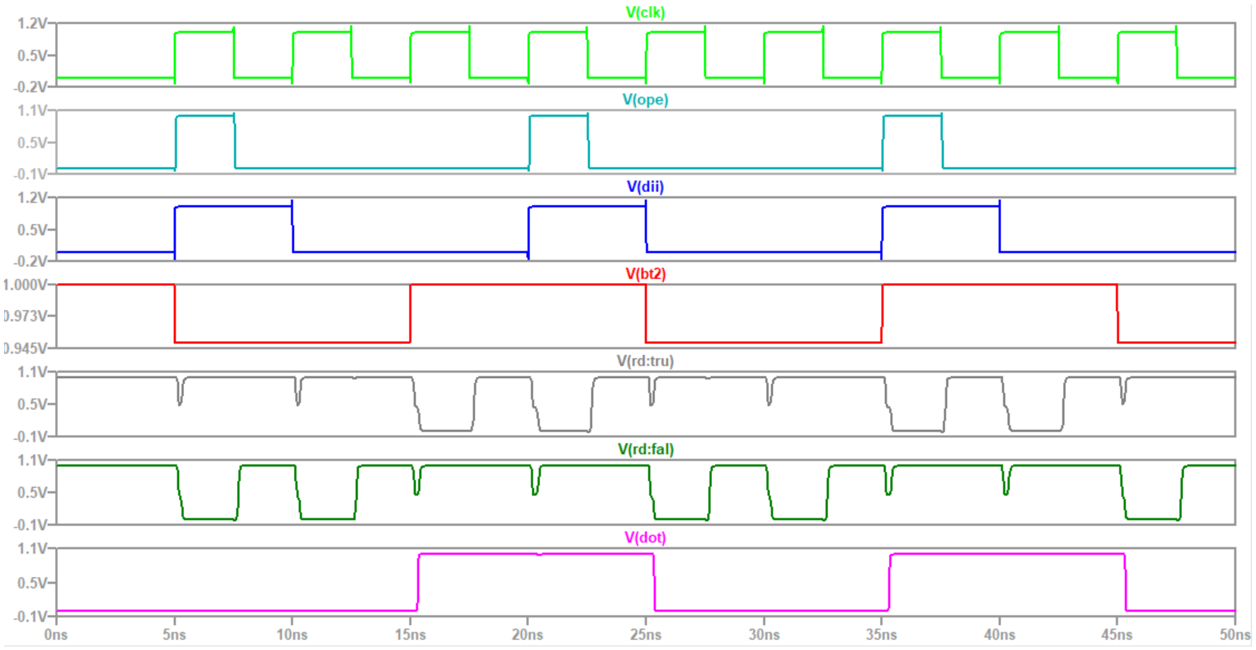
The sizing was done to maintain a rough ratio of pull ups being twice the size of pull downs, to compensate for electron mobility vs hole mobility.

Schematic:



D. Read Block

testRead.cir output:



Read1 subckt:

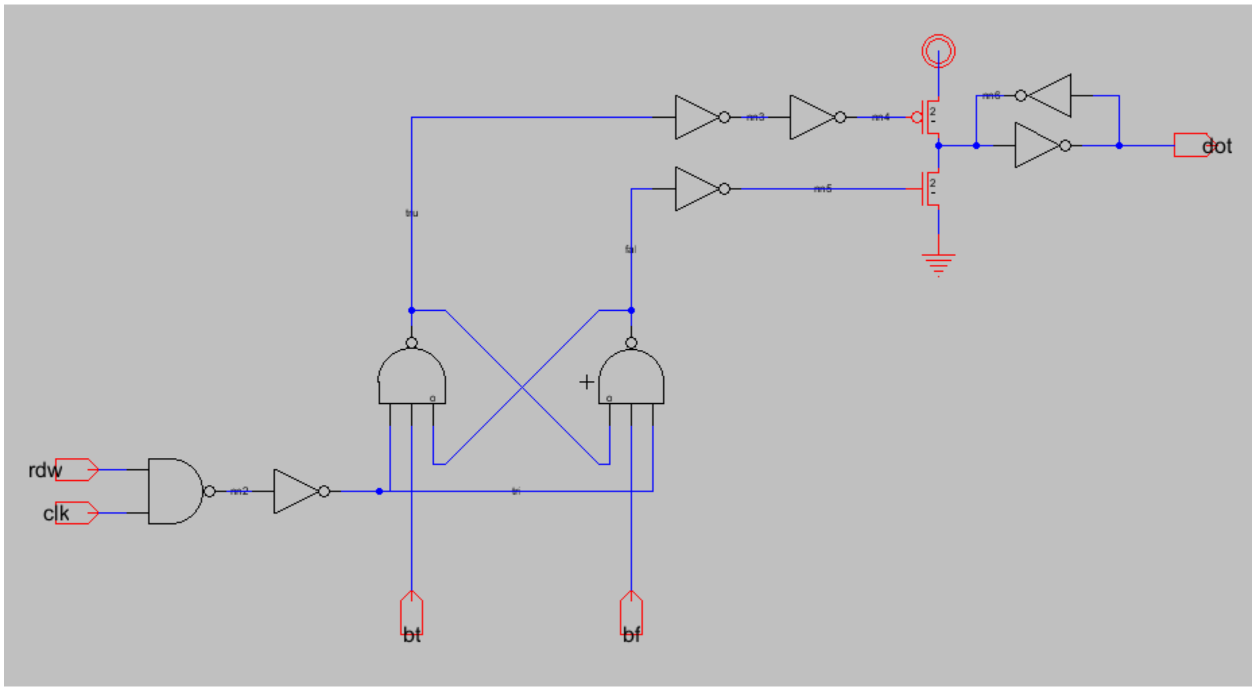
```
.subckt read1 btt bff dot rwt clk size=10
Xnd1 nn1 rwt clk nnd2 size='size'
Xin1 nn2 nn1     inv  size='size'
Xse1 fal tru bff btt  nn2 senseAmp size='5*size'
Xin2 nn3 fal     inv  size='2*size'
Xin3 nn4 nn3     inv  size='2*size'
Xin4 nn5 tru     inv  size='2*size'
Xpp5 nn6 nn4 vdd pp   ww='2*size'
Xnn5 gnd nn5 nn6 nn   ww='size'
Xin5 nn6 dot     inv  size='size'
Xin6 dot nn6     inv  size='2*size'
.ends read1
```

Read Description:

The read block's purpose is to take in rdw, clk, bt, bf and output the value stored in the SRAM cell to dot. The rdw acts as an enable with high being read enable, and low being disabled. The read block senses bt and bf and compares the two with an analog sense amp. If bt is higher than bf the output will be high, and if bf is higher than bt, the output will be a low.

The sizing was determined iteratively, starting with size 30, my output was out of wack, so I stepped it down by 10 until I was able to get the dot output similar to the reference. The sense amp I also had to perform the same iterative process going up and down by 10, starting at 40. I played around a lot with the inverter sizing as sometimes I would see a spike on the dot output, when it should be asserted high. I looked at the signals which I wanted to drive with more strength and sized accordingly.

Schematic:



E. testBuffer

readSub subckt:

```
*read sub is dut (seems like changing sizes doesnt do anything to output)
*sized according to reference sense amp provided
.subckt readSub btt bff set rst rdw clk size=40 beta=2
Xnd1 nn1 rdw clk nnd2     size = 'size'
Xin1 tri nn1     inv      size = 'size'
*Xsa1 set rst btt bff trg senseAmp size = '20'
Xnd3 set tri btt rst nnd3 size='size'
Xnd4 rst tri bff set nnd3 size='size'
.ends readSub
```

readSub Description:

The readSub is equivalent to the lower half of the above read circuit, in which the outputs are the outputs of the above sense amp. It was broken down in this way so that the upper half of the read can be multiplied, as would be the case for the 32x8 cell design I chose.

The sizing stayed consistent with my original read circuit except I switched from 50 to the default sizing provided by reference.

readCollect8 subckt:

```
.subckt readcollect8 nnn st7 rs7 st6 rs6 st5 rs5 st4 rs4 st3 rs3 st2 rs2 st1 rs1 st0 rs0 size=30 beta=2
Xnd7 aa7 st7 st6 nnd2
Xnd6 aa6 st5 st4 nnd2
Xnd5 aa5 st3 st2 nnd2
Xnd4 aa4 st1 st0 nnd2
Xin7 hi7 aa7     inv
Xin6 hi6 aa6     inv
Xin5 hi5 aa5     inv
Xin4 hi4 aa4     inv
Xpp7 dot hi7 vdd pp ww='2*size'
Xpp6 dot hi6 vdd pp ww='2*size'
Xpp5 dot hi5 vdd pp ww='2*size'
Xpp4 dot hi4 vdd pp ww='2*size'
Xnd3 io3 rs7 rs6 nnd2
Xnd2 io2 rs5 rs4 nnd2
Xnd1 io1 rs3 rs2 nnd2
Xnd0 io0 rs1 rs0 nnd2
Xnn3 dot io3 gnd nn
Xnn2 dot io2 gnd nn
Xnn1 dot io1 gnd nn
Xnn0 dot io0 gnd nn
*output state holder
Xinx dot nnn     inv
Xiny nnn dot     inv
.ends readcollect8
```

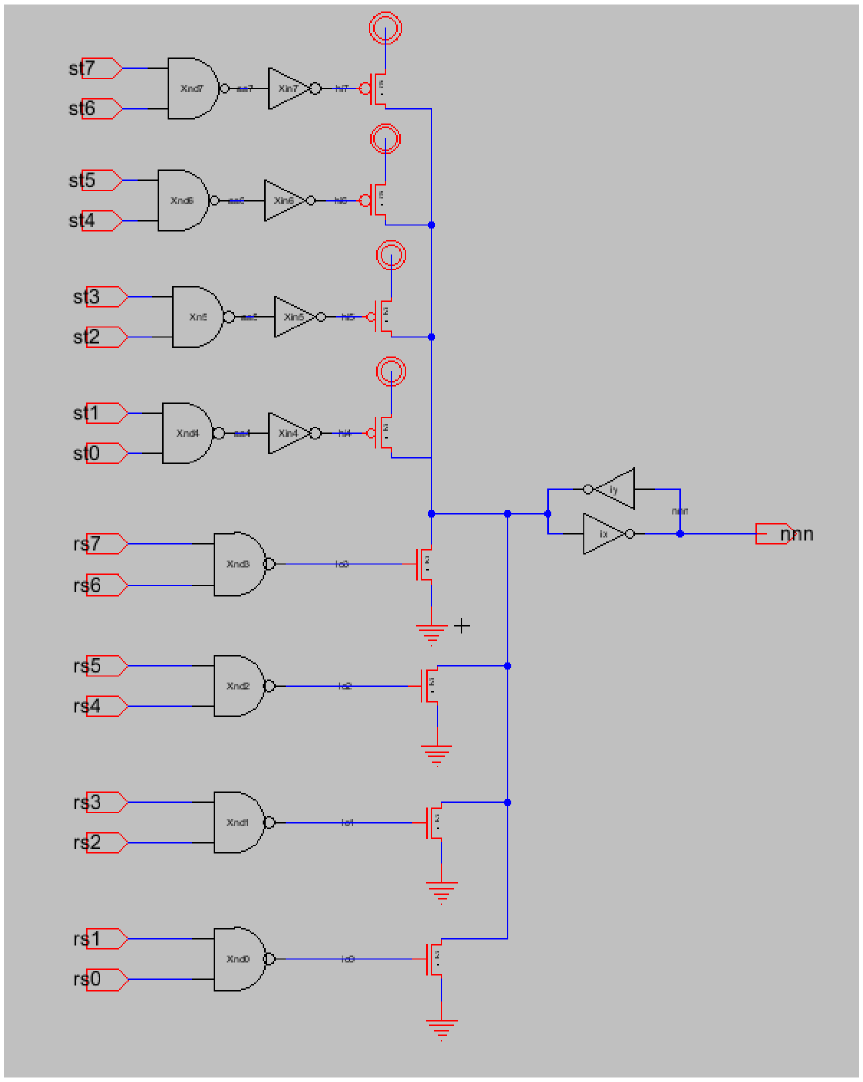
readCollect8 description:

readCollect8 is equivalent to the top half of the above read circuit, replicated 8 times for 8 different reset and set signals. In my test buffer I chose to do some hand waving and only assert a single reset and set line out of 8 to show a worst possible timing for a single line. Using the single line, results can be extrapolated for all of them.

For my test, I am putting signals into st7 and rs7, which merge on hi7 and io3 respectively.

I switched the output from dot to nnn because when I was first testing, I had the correct output at dot, however, it was inverted from what it should be. So as a quick fix I used the state holder at the output to invert it.

Schematic:



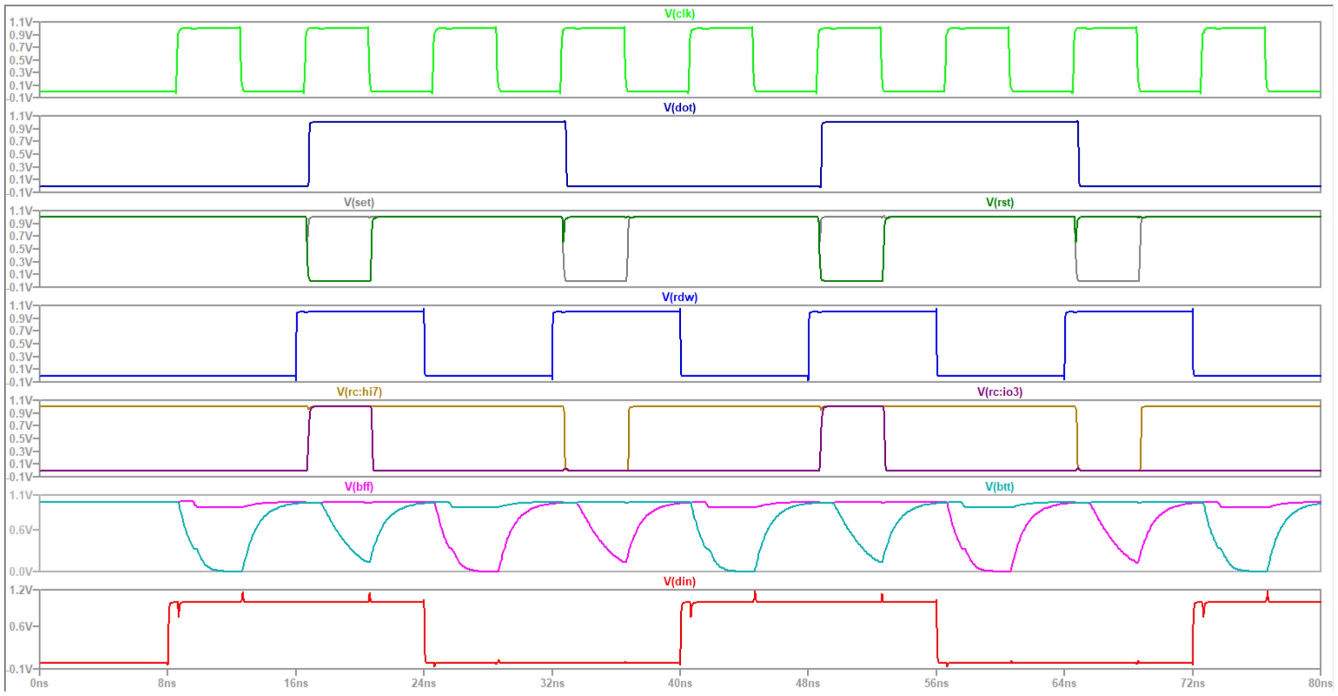
testBuffer description:

Tested Parameters:

```
********begin: topLevel*****
.param per = 8ns
.param dataLead=500ps
.param lw=1358
.param wirew=12
```

I got my length from layout of 32x8 SRAM cells (Shown in layout section). The wire width was just one I started with and I ended up getting decent results, so I kept it the same. I was able to get the correct output with 7nS clock frequency, however, hysteresis prevented me from going that low because I saw that the btt and bff voltages were separated by around 10mV.

SRAM testBuffer output:

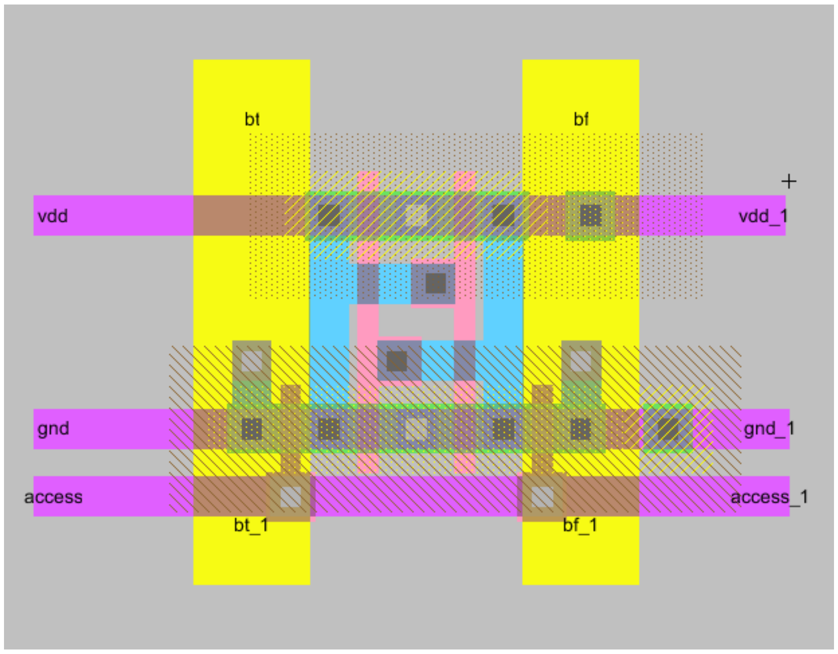


Electric Layout:

DRC: PASSED

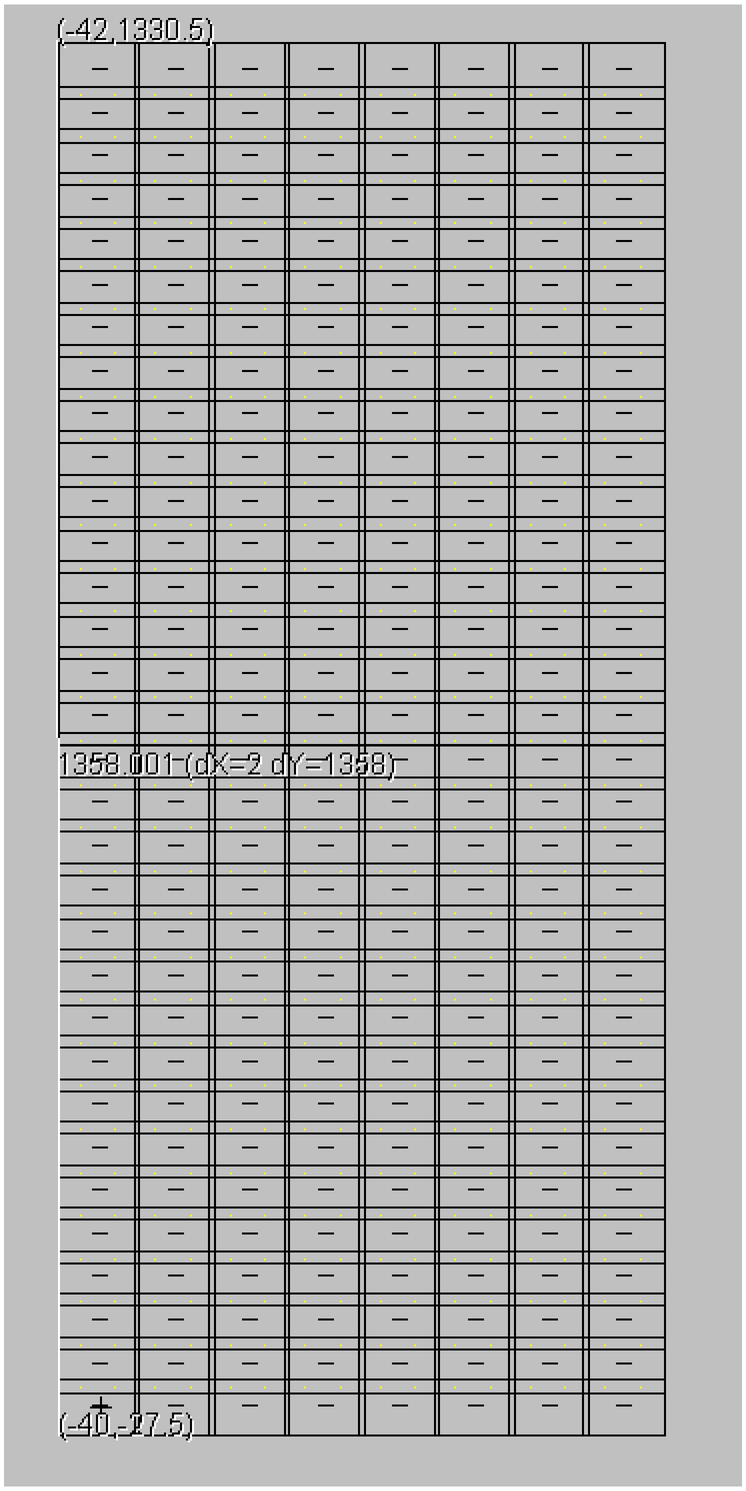
NCC/LVS: PASSED

Final Layout cell used cell name: “Stephen_More_1bit_SRAM_v3”



Cellname: “Stephen_More_32_by_8_FINAL”

32x8 Layout:

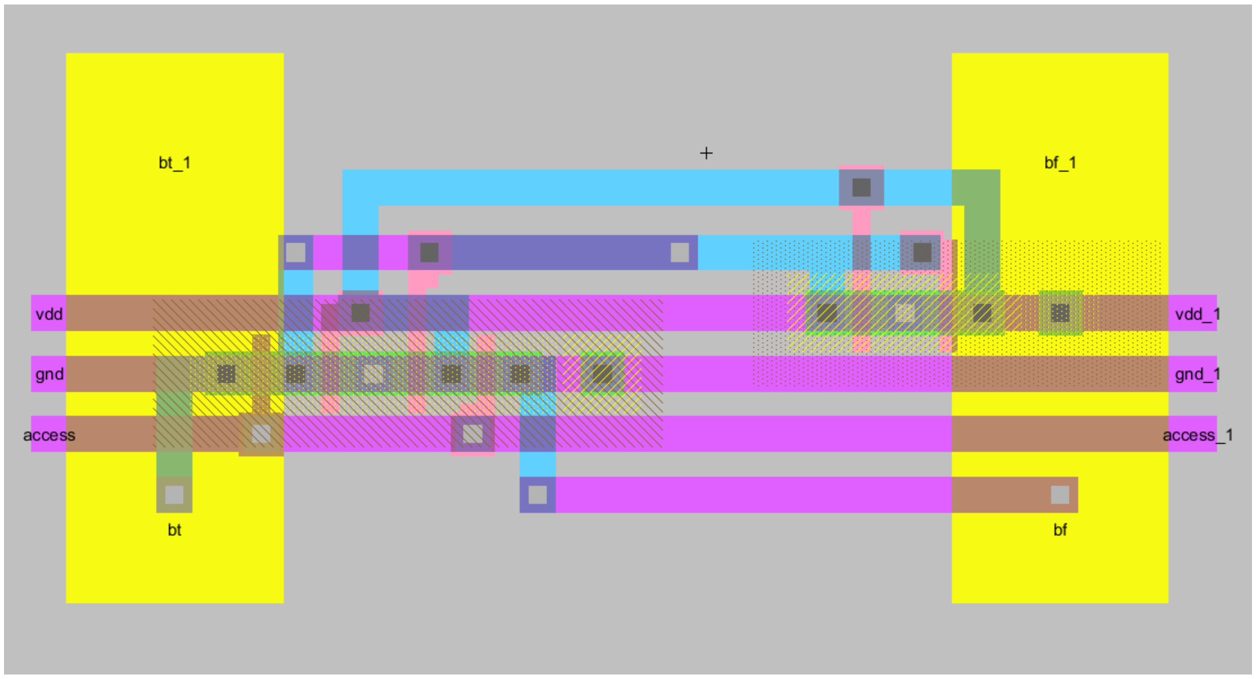


Layout Description:

To represent the whole 2048 bits, the layout above would be replicated 8 times horizontally.

I tried a few different iterations of layout, in the “Stephen_More_Final_Project_471” library, and the third iteration was the best density, as well as organization that I could manage. I got the idea for the layout from google images when I looked up SRAM layouts.

Unused “Stephen_More_1bit_SRAM_v4”:


bt_1
bf_1
vdd
vdd_1
gnd
gnd_1
access
access_1
bt
bf