ESE 570 Final Project: DRAM 4x4 Memory

Celine Lee

April 2020

1 Introduction

The purpose of this project is to design and build a memory cell array of 4 words with 4 bits each. A user interacting with the design should be able to provide an address, an instruction (read or write), and 4-bit data (depending on read/write). If the user selects write, the input data will be written to the word at the address specified. If the user selects read, the data stored in the word at the address specified is read out onto the output registers. Each bit cell is a 3T DRAM cell, and as such, the circuit is designed to also periodically refresh the data in each word.

2 Overall Design

The final schematic is pictured below:

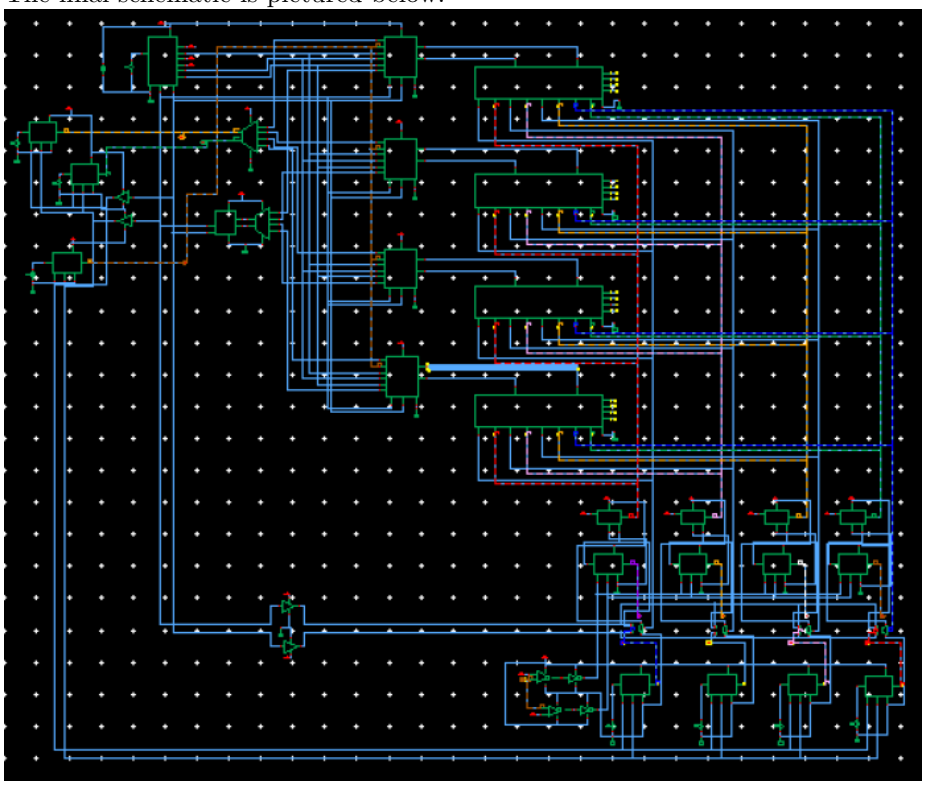

Figure: Final schematic for entire design

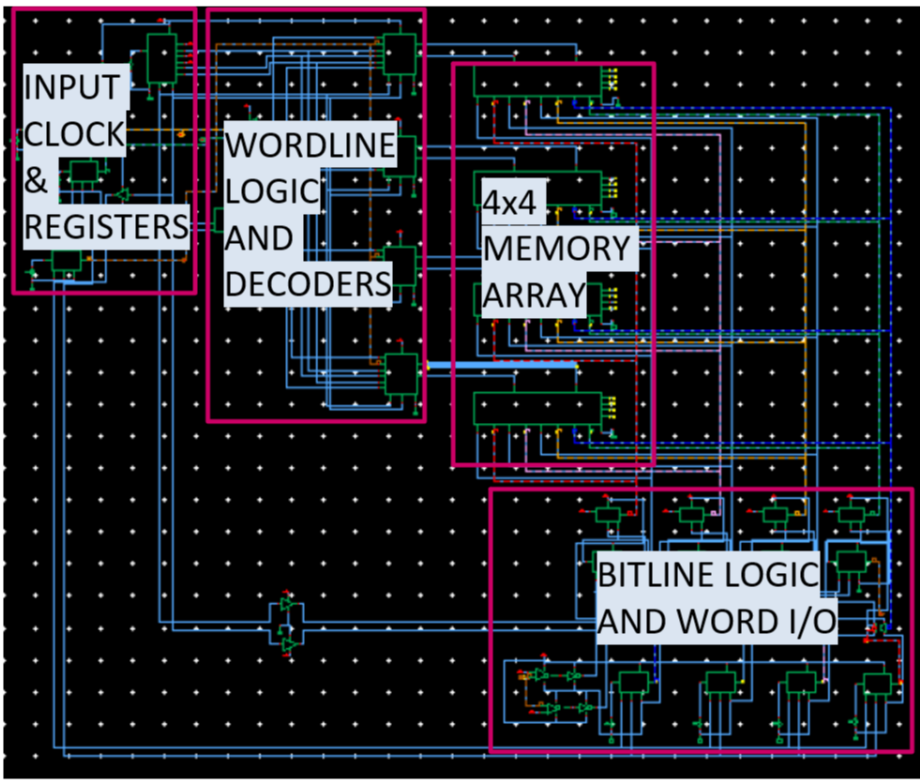

Figure: Labeled schematic for entire design

Users provide input by means of a 2-bit address, a 4-bit data bus, a write-enable (WE) bit, and a single clock signal. In return, the user receives output from the output registers, which, if the user selects to read (WE=0), reads out the data at the address specified.

To perform the memory logic, the entire design is composed of several sub-components, the major ones of which are the bit cell, the address decoder, the refresh circuitry, the registers, and the clock generator. The operation and design of these will be specified in the next section: 'Building Components.'

The entire design operates with a period of $8ns$, a frequency of $125MHz$.

2.1 Verification of Overall Design

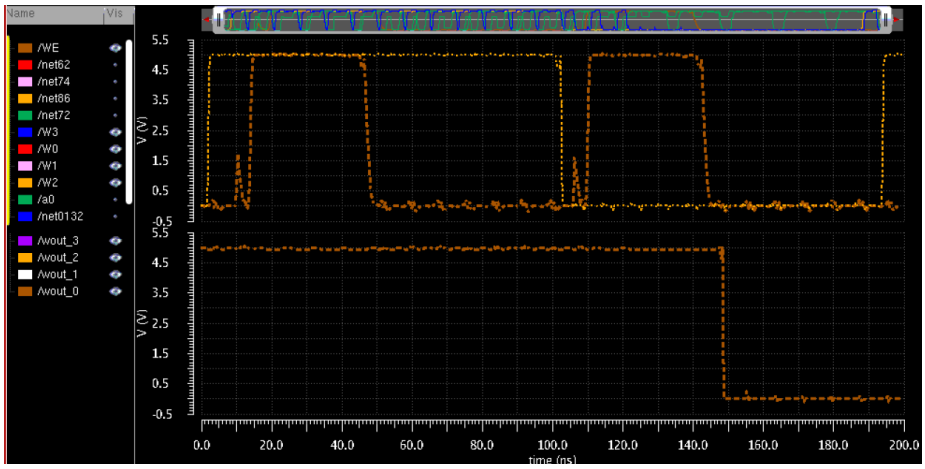

Figure. Output of entire (extracted) design

The test case we use to verify our logic is: (write 1111 to each word, read from each word, read from each word, write 0000 to each word, read from each word, read from each word). We read twice ensure that the first read did not corrupt the data. Also note that the first clock cycle is for setup, so it is invalid, but setup only needs to be done once. We can see from the output in the figure that during the first two read, the outputs are stable at high and during the last two reads, the outputs are stable at low. This is consistent with expectations. Also, the register does not oscillate during write operations– it hold whatever the last value it held was.

To look a little bit under the layers, we can also look at the read bit lines and probed data throughout the same time propagation:

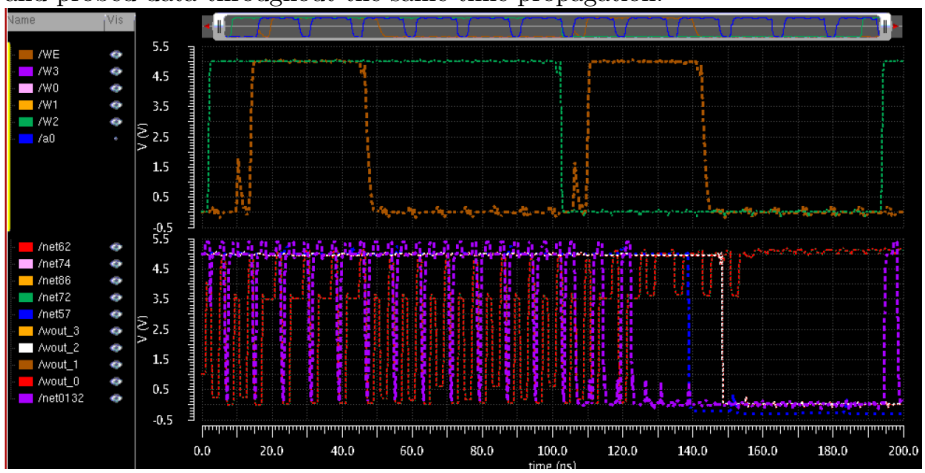

Figure. Probed output of entire (extracted) design

The red signal is the RBL throughout the process, as it continues to precharge to V_{dd} at the beginning of every clock cycle, and adjusts to the read action when

needed. The purple line is the output written to the registers, the inverted RBL value. The blue line is the probed data value held in the cells, and the white line is the word out from the registers to the users.

3 Building Components

3.1 3T DRAM Bit Cell

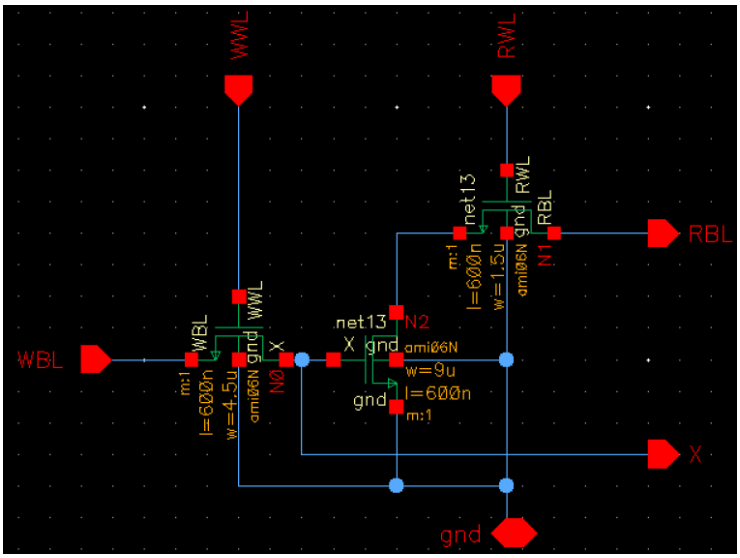

Figure. The 3T DRAM bit cell

The 3T dram bit cell works by storing charge on the parasitic capacitance between the leftmost and the middle transistor. To write to the cell, one would put the desired input data on WBL (Write Bit Line) and then raise the WWL (Write Word Line). With the leftmost transistor on, data now can flow between the data storage capacitance and the driving input. Note that if a '1' is being written to the storage point, the voltage held there will actually be $V_{WWL} - V_{th}$. To read from the cell, we use the right two transistors. If a '1' is being stored (atually a voltage of $V_{WWL} - V_{th}$), the middle transistor will be on, and when the RWL (Read Word Line) is risen, the rightmost transistor will also be on, allowing RBL (Read Bit Line) to start discharging to ground. On the other hand, if a '0' is being stored, the middle transistor will be off, and when the RWL rises, there is no actively driving voltage source affecting the RBL, so the RBL remains at whatever charge it previously held. (This previously-held charge will be V_{dd}, as set up by the precharge ciruitry explained in the next subsection.)

The transistor is sized at $W = 4.5uM$ for the leftmost transistor, $W = 9uM$ for the middle transistor, and $W = 1.5uM$ for the rightmost transistor. These sizings were all started at minimum $1.5uM$, and sized up as the remainder of

the circuit was designed in order to provide sufficient driving force. Since the clock inputs are rather short (around $1ns$ each), the word lines follow suit and are also only on for less than $1ns$ at a time, so the transistor needs to be large enough for the information to fully pass through.

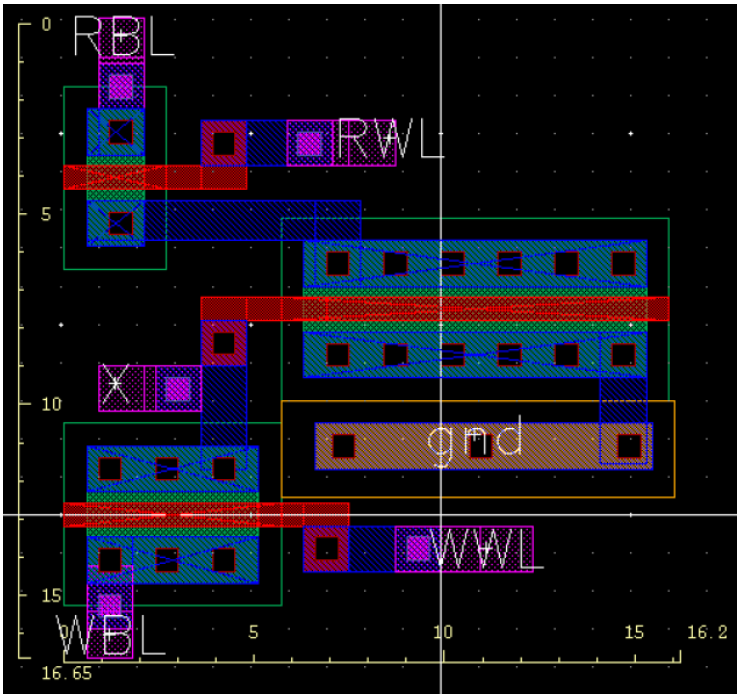

Figure. Bit cell layout

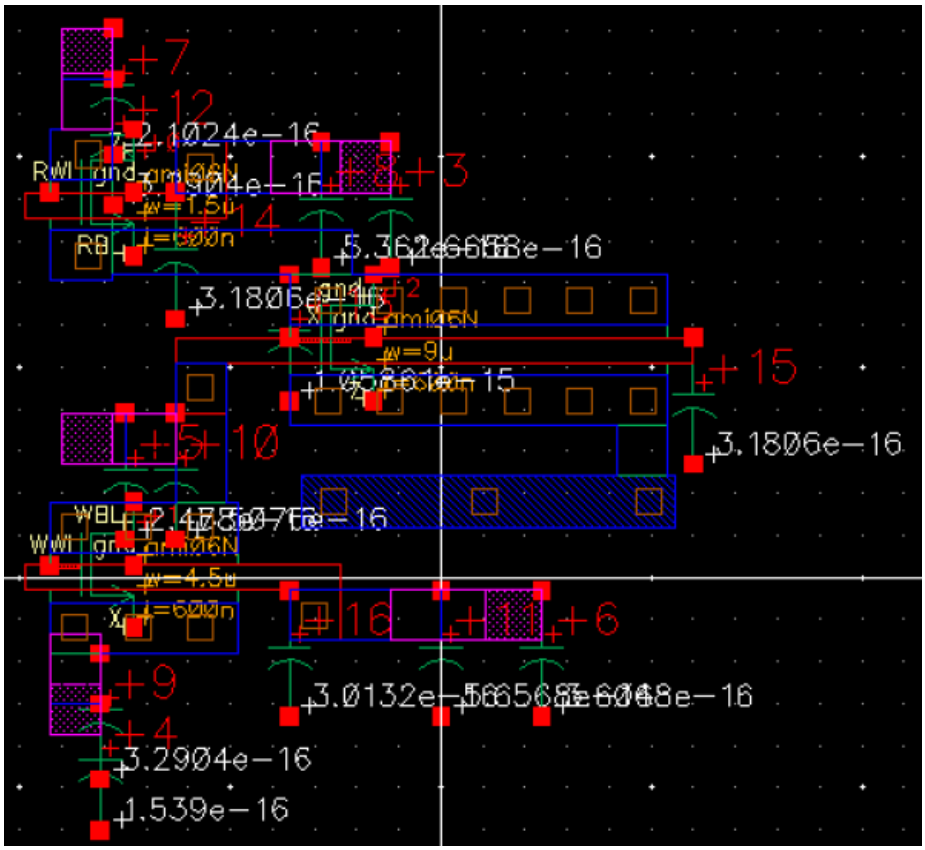

Figure. Bit cell extracted layout

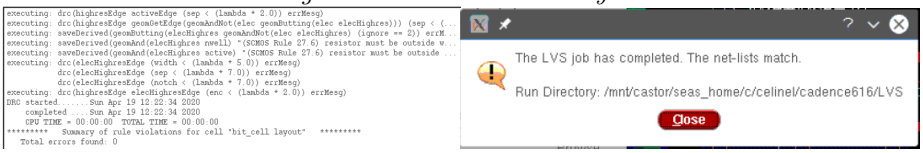

Figure. DRC and LVS passed

We can see simulation outputs to validate the bit cell performance:

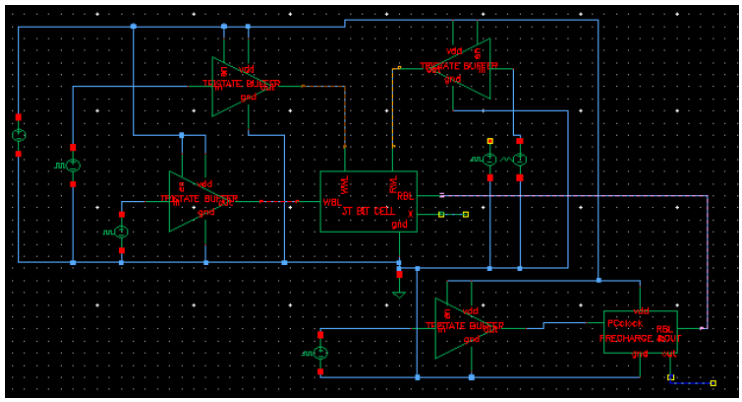

Figure. Test bit cell schematic. We use buffers to simulate imperfect input signals

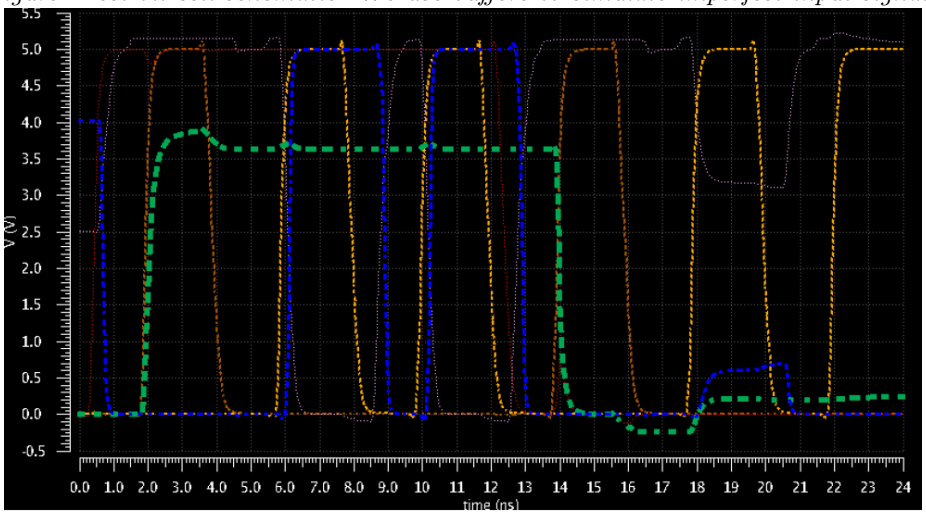

Figure. Output of test. We probe the X spot to determine success

The orange line is the WWL, the yellow the RWL, and the red the WBL. The green dashed line is the probed storage capacitor. (Note that though I created pins for this, the pins are purely used for test probing purposes and are never connected to any input/output directly.) We can see that the logic high is written to the cell when the WWL raises and the WBL is high, and that it is unaffected when the RWL raises. We perform read twice after the write operation, the first to read the data, and the second to ensure that the first read did not corrupt the data. We also see that the logic low is overwritten to the cell when the WWL raises and the WBL is low. The RBL is lightly traced in pink, and we can see it react when the RWL raises: when reading a stored high, it drops in voltage all the way to 0, and when reading a stored low, the voltage does not drop all the way to 0. I find that by inverting the read from the RWL, the signal is more informative, so that is traced in dark blue. We see that during the first two reads, it raises to high, and in the last two reads, it

drops to low, which is in line with the data that was stored in the cell (green).

In the corner of the schematic is the pre-charge circuitry, used to pre-charge the Read Bit Line to high before the read operation is executed. Its operation will be outlined in the next section.

3.2 Precharge Circuitry

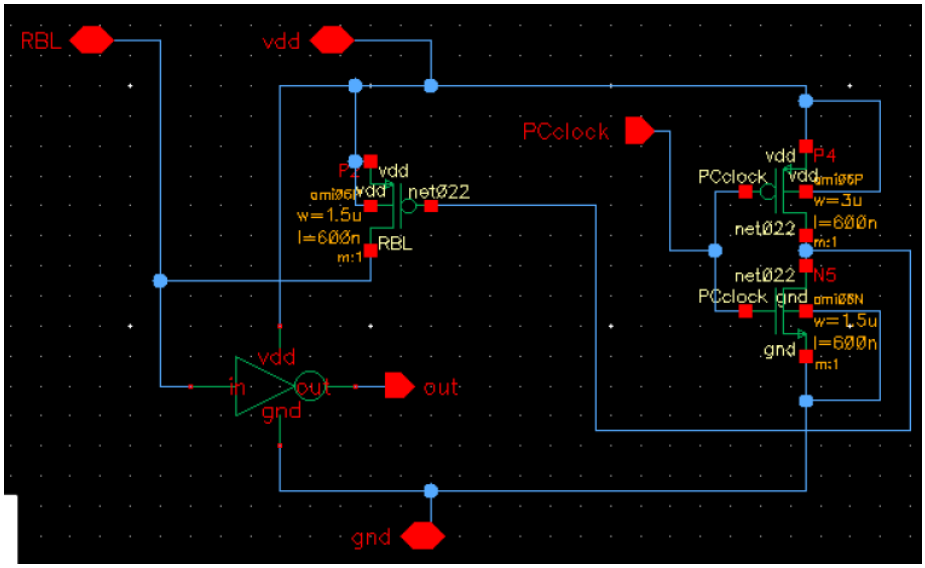

Figure. Precharge and RBL reader circuitry

The purpose of the pre-charge circuitry is to pre-charge the bit lines up to V_{dd} right before the read operation. This needs to be done because if the bit cell is storing a 0, no change to the RBL will occur, and if the bit cell is storing a 1, the RBL will head toward 0. So providing the RBL a pre-charge of V_{dd} allows the RBL to move to the NOT of the data stored in the cell.

The pre-charge is performed by setting a PMOS to be the gatekeeper between the V_{dd} and the RBL. When the PCclock is high, the PMOS turns on, and RBL is charged up. When the PCclock goes back low, the PMOS turns off, and the RBL is cut off from V_{dd}, but it still holds (temporarily) that pre-charged value.

An added element to the pre-charge circuitry is an inverter put on the RBL, in order to interpret the data on the RBL from a read operation. Initially, this circuitry also included the sense amplifier, but as I performed testing, I found that I didn't even need the sense amplifier because the bitline capacitance for these 4 words was rather low. Therefore, the sense amplifier was removed to conserve space/power, and a simple inverter was put in its place.

3.3 4 Bit Word

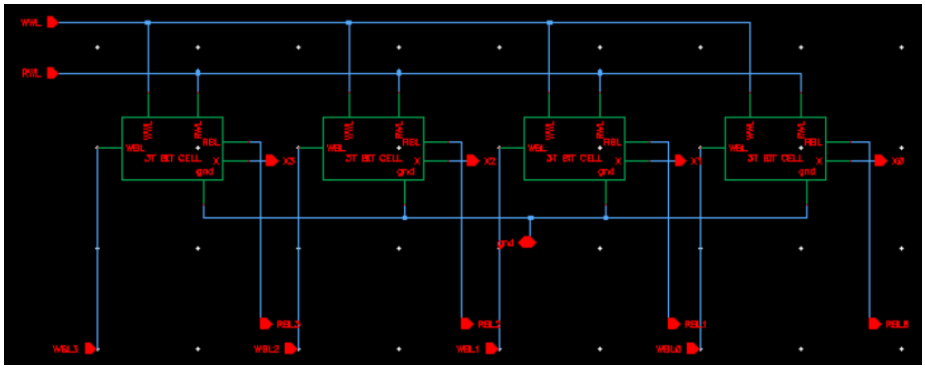

Figure. A word

The 4-bit word block is created simply to make the overall design more modular and easier to debug. The 4-bit word is 4 3T bit cells hooked up with one WWL, one RWL, and each of their respective bit lines as well as a pin to allow probing of the stored data, for testing purposes.

3.4 Tri-state Buffer

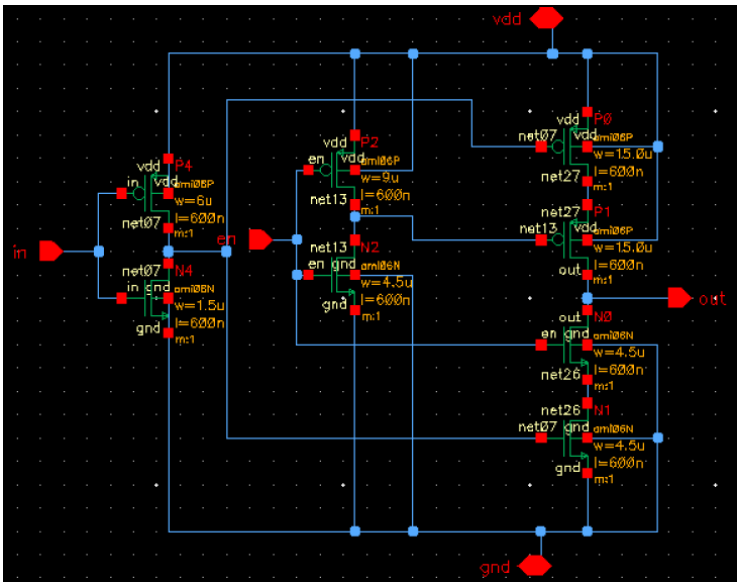

Figure. The tri-state buffer

The tri-state buffer is used extensively throughout this design because of its mux-like ability to allow or disallow certain signals through, based on an enable input. The design is a sort of extended buffer, in that the input is inverted and fed to a slightly-more-complex inverter. This second "slightly-more-complex inverter" has an NMOS and a PMOS that only turn on if the enable is high. Therefore, so long as the enable is high, the output is exactly what the input

is, and if the enable is low, the output is cut off from the input all together.

The driving ability of this tri-state buffer is critical because it takes in signals that are being fed to many inputs, and it feeds out signals that are also fed to many outputs. Oftentimes, the tri-state buffer is used to pass clock signals through, so timing is sensitive. Therefore, we provide the output a lot of driving force, sizing the output PMOS's to $15\mu M$ and the output NMOS's to $4.5\mu M$. The input transistors are sized a little smaller, but still notably up in order to make the buffer quick. The enable inverter is given a $9\mu M$ PMOS and $4.5\mu M$ NMOS, and the input inverter is given a $6\mu M$ PMOS, and $1.5\mu M$ NMOS.

We test the inverter to see its performance:

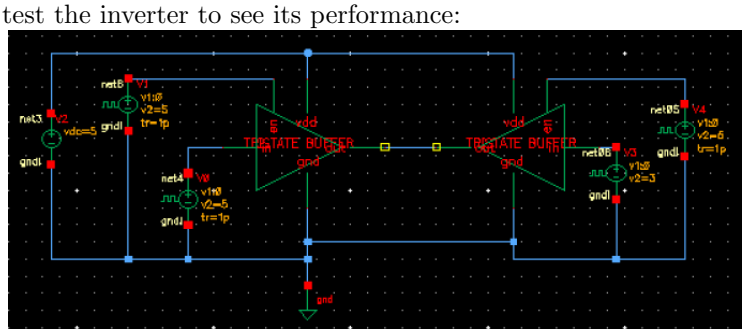

Figure. The testing schematic for the tri-state buffer

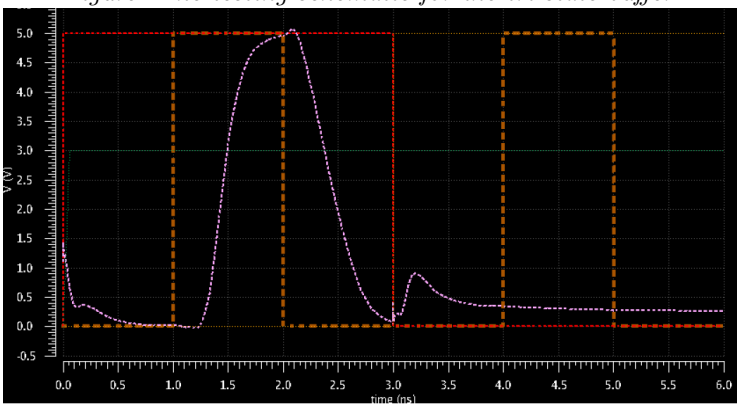

Figure. The testing output for the tri-state buffer

I set up the testing schematic to have two tri-state buffers facing each other, and see how it would perform in isolating and driving inputs. The red line is the enable for the left buffer, and the orange is the input to the left buffer. We can see that when the enable is high, the output follows the input. But when the enable is low, the output remains at what it was when the enable was last high.

3.5 Register

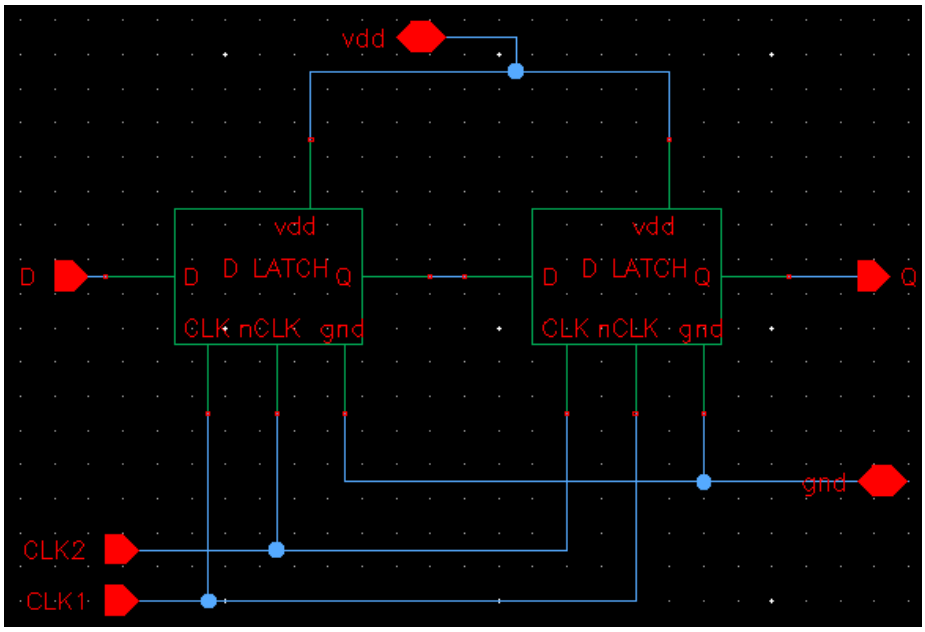

Figure. The register

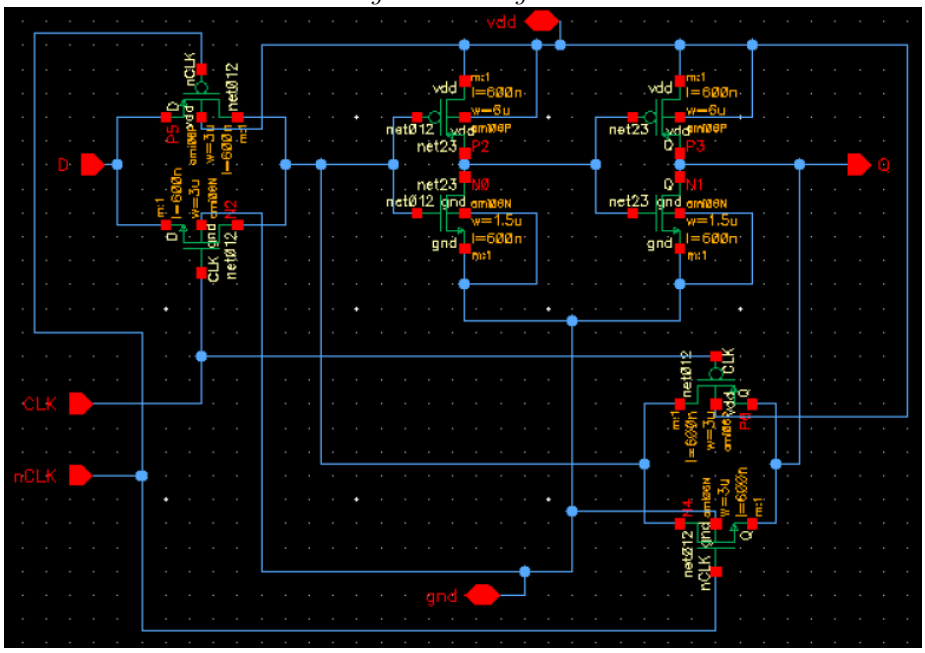

Figure. The static D-latch

The register works by cascading two positive-triggered d-latches that have opposite clocks. When the first clock rises, the first latch takes in the data. Then

the first clock falls, and the data is held in the first latch. This output is then fed into the second latch, which takes the data once its clock rises, and holds it when the clock goes back down. The result is a register that only changes input on rising edge of the (second) input clock.

The latch works by having the clock open a gate between the input and the feedback loop that holds the data. When the clock is off, the feedback loop remains a closed loop. When the clock is on, the feedback loop takes the input.

It is important that the latches/register react quickly enough to store data within the short clock cycle it is given, so we size up the PMOS for the feedback inverters to $W = 6\mu$ and the clock transistors to $W = 3\mu$.

We can verify the register:

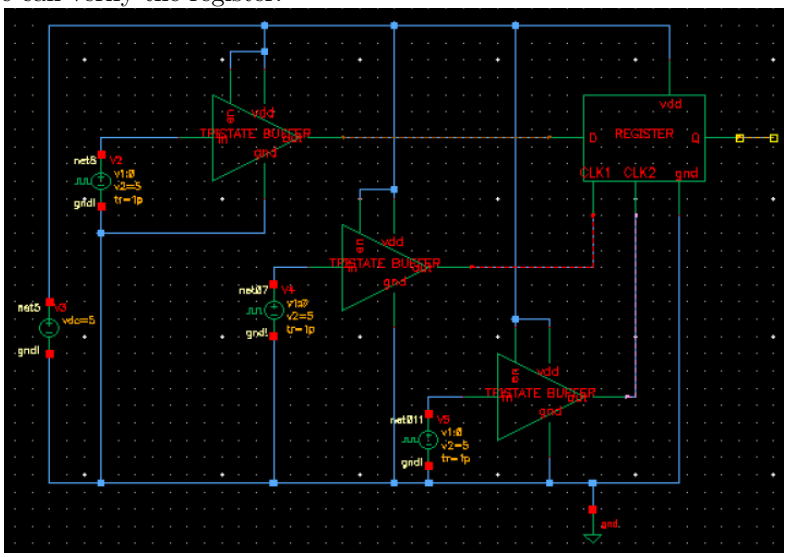

Figure. Register verification schematic.

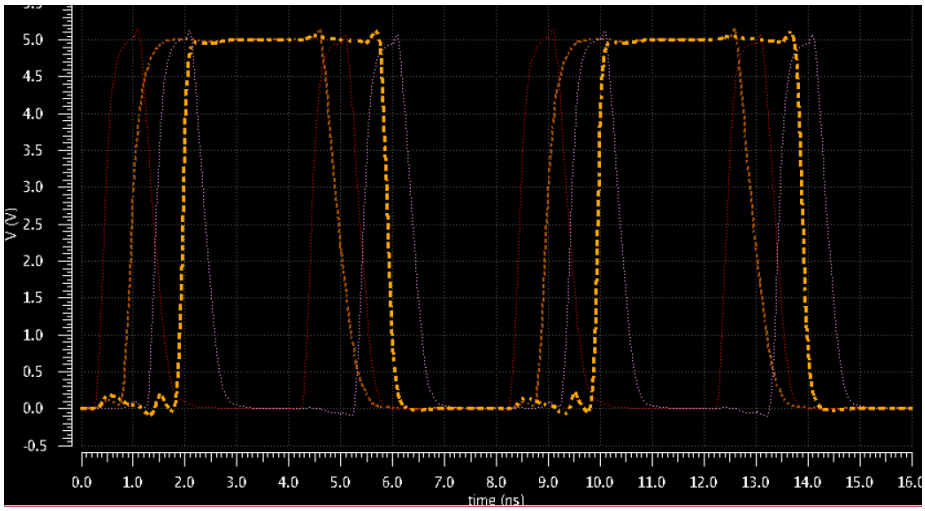

Figure. Register verification output.

See that the output (yellow line) follows the input (orange line) a little after the rising edge of the second clock.

3.6 Decoder

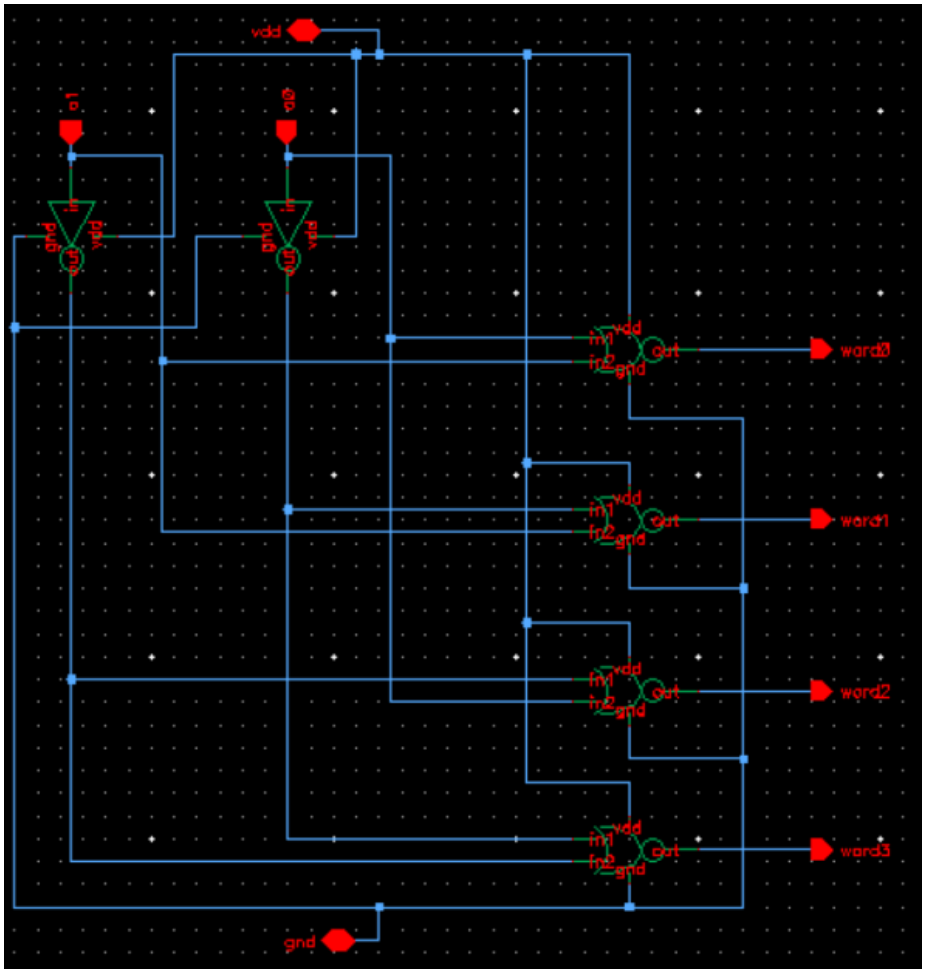

Figure. The 2-4 decoder

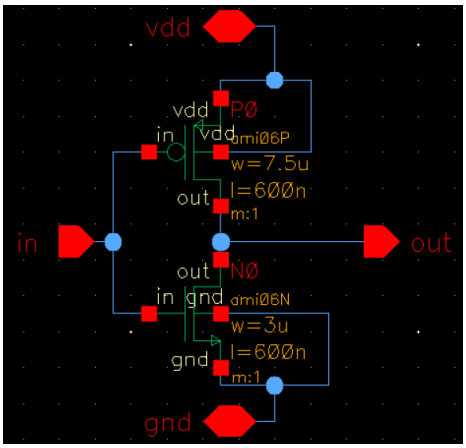

Figure. The inverter

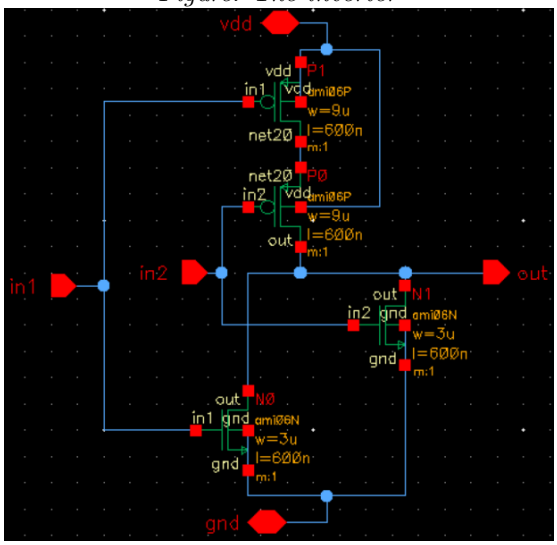

Figure. NOR gate

The decoder raises exactly one of its four output word lines based on the 2-address input, in a 1-1 mapping of combination to word line. (e.g. word0 = (a0=0, a1=0), word1 = (a0=1, a1=0), etc.) It does this using a combination of gates: inverters, and 2NORs.

The timing for this component is not as important because decoders only act once per clock cycle, and they are often done well before the outputs are used. Therefore, the sizing of the gates is really for the benefit of wherever else the gates are used in the design, always to make things faster, as the PMOS's are sized up to 7.5μ for the inverter and 9μ for the NOR, and the NMOS's to 3μ for both.

We verify the decoder:

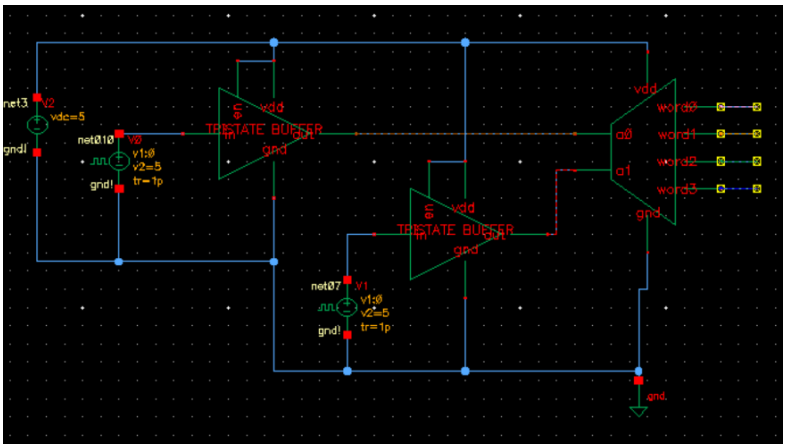

Figure. Decoder verification schematic.

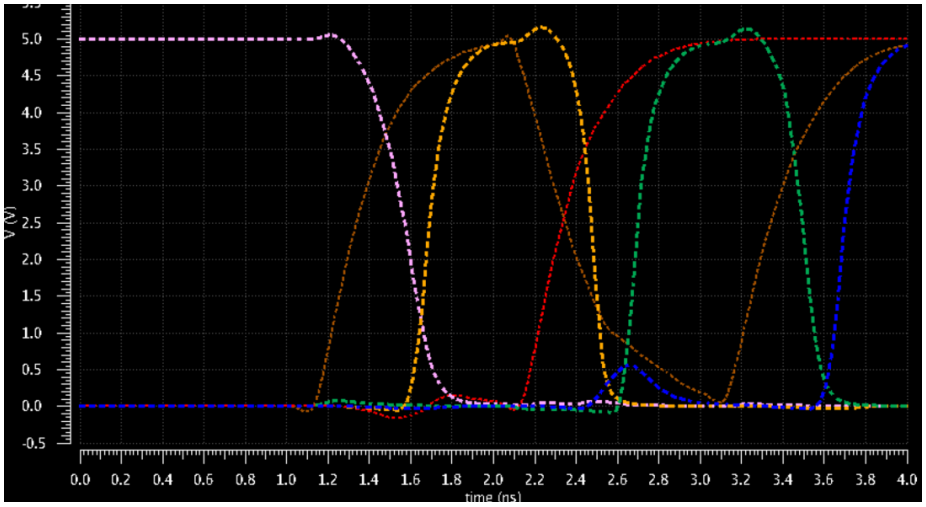

Figure. Decoder verification output.

We see that the inputs (a1=red, a0=orange) create, in order, combinations: 00, 01, 10, 11. Subsequently, each of one word line raises (word0=pink, word1=yellow, word2=green, word3=blue).

3.7 Non-overlapping Clocks

The clock generator is one of the most critical pieces of this entire schematic. Creation of non-overlapping clocks for each of the steps that occur in the process writing to/reading from memory is necessary in order to ensure proper operation without data corruption. The clock generator takes exactly one clock input, a 50%-duty cycle, 8ns-period (125MHz) clock. An 8ns period was selected after building the registers, bit cells, bit columns, and tri-state buffer. In each of these subcomponents' respective test schematics, imperfect input signals were

created by using the tri-state buffer and loading the internal signal buses with equivalent expected capacitive loads. By experimenting with a range of transistor sizings around some baseline, I found an optimal timing for this project at approximately $1ns$ assigned to each sub-operation to complete without data upset. Since I needed 6 distinct $1ns$-pulse clocks, with a little bit of time extra for settling, a $8ns$ clock was found optimal. (An added fact is that 8 is the nearest power of 2 higher than 6, and my initial clock generator design was a series of clock dividers (into two).)

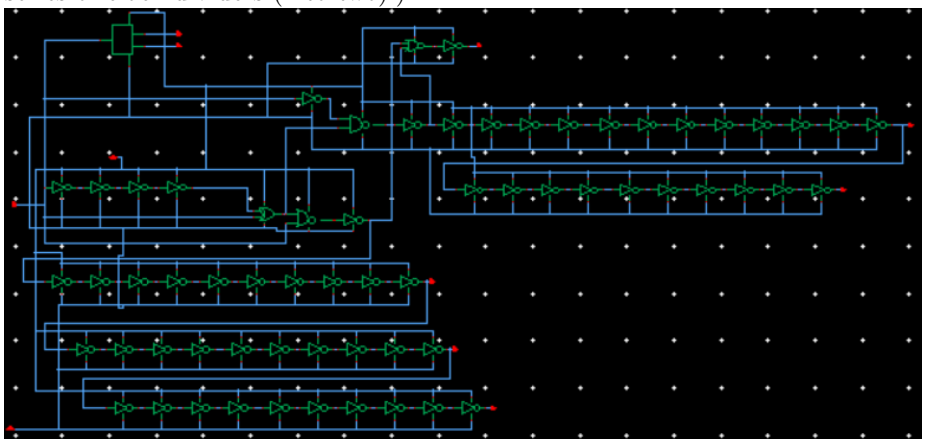

Figure. The sub-clock generator (clock divider)

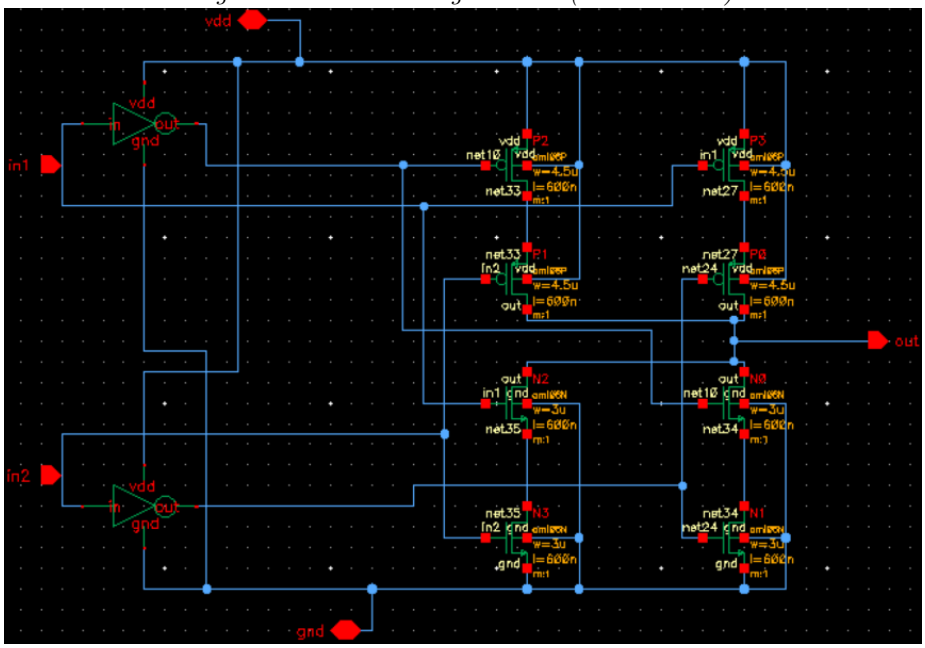

Figure. XOR gate

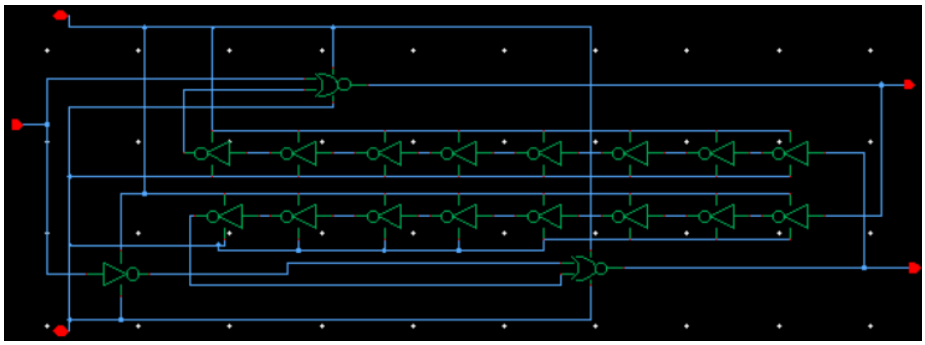

Figure. Non-overlapping clock generator of period 8ns

First, the clock creates a $1ns$ pulse by XORing itself with a delayed version of itself. The number of inverters used to create the delay here was extensively experimented with and modified as the clock cycle was determined. Then this $1ns$ pulse goes through a series of different delays to create the subsequent clocks. To get the clock pulses on the second half of the clock cycle (when it is at logic 0), a $8ns - to - 4ns$ non-overlapping clock generator and perform the same steps in order to create a series of $1ns$ pulses.

Timing here is of the most importance. The clock outputs must be given enough drive, and must rise/fall as close to sharply as possible in order to allow its (many) outputs to react as designed. Therefore, the XOR gate and inverter gate are sized up. The inverter has been described by a previous section. In the XOR, the NMOS's are given $W = 3\mu$ and the PMOS's are given $W = 4.5\mu$.

We test the clock generator by putting an approximately-equivalent load on each output, to make sure it has enough drive to create clock pulses:

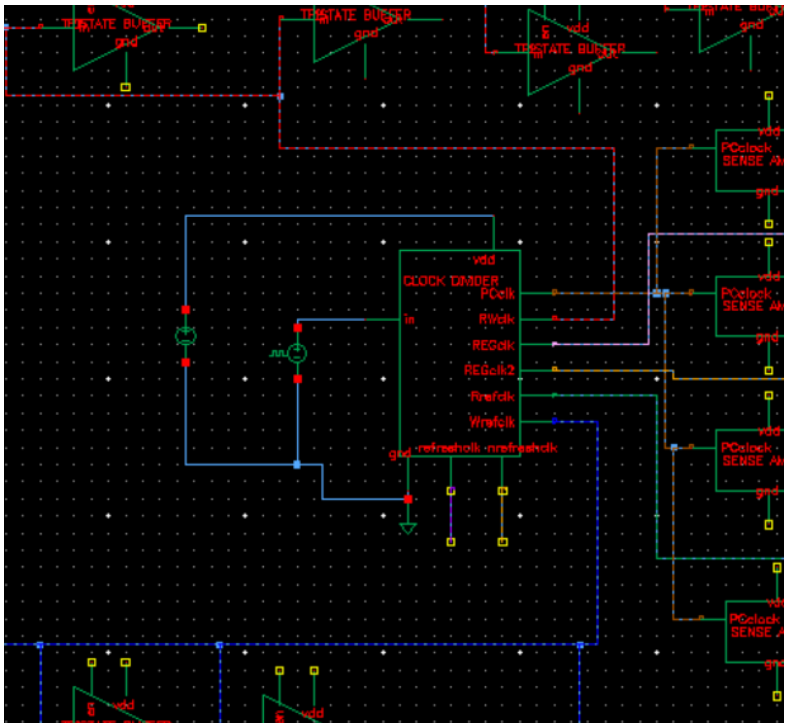

Figure. Clock divider verification schematic.

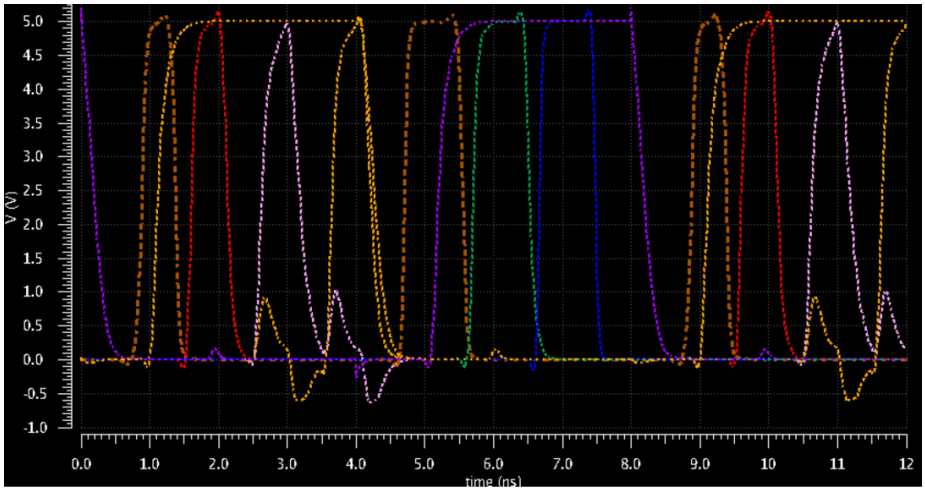

Figure. Clock divider output.

Figure. Expected clock divider, 1 period.

We can see that the output of the clock generator successfully creates the 6 non-overlapping $\sim 1ns$ pulses and 2 non-overlapping $\sim 4ns$ pulses that are used throughout the model.

3.8 Counter

The purpose of the counter is to select one word on each clock cycle to have its data refreshed.

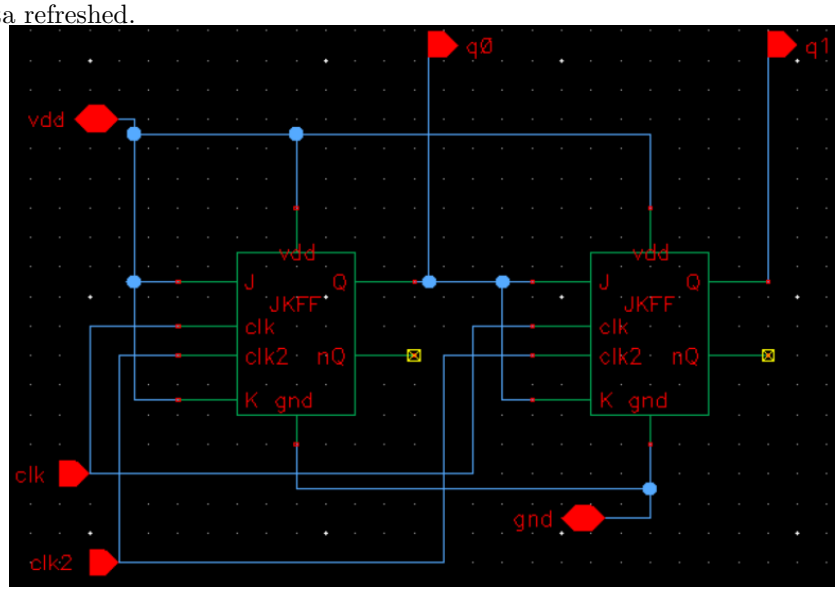

Figure. The 2-bit counter

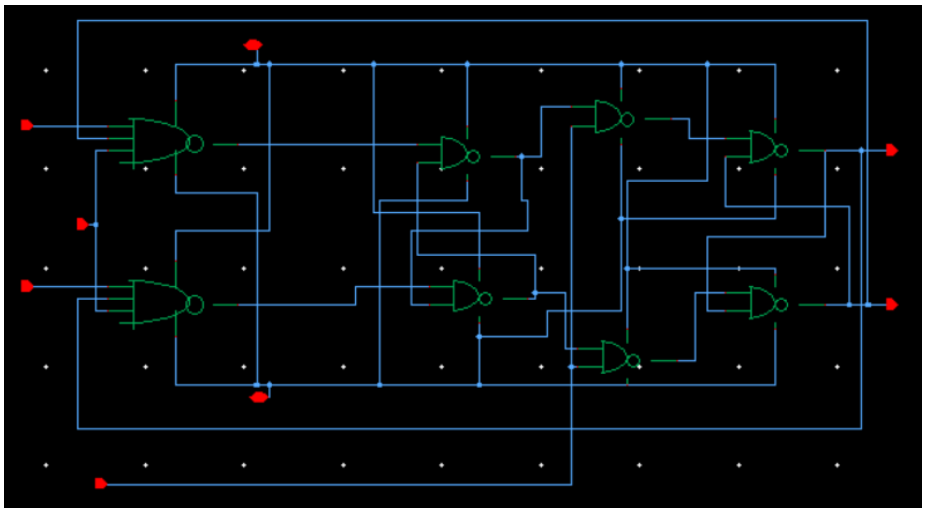

Figure. The JK Flip-Flop

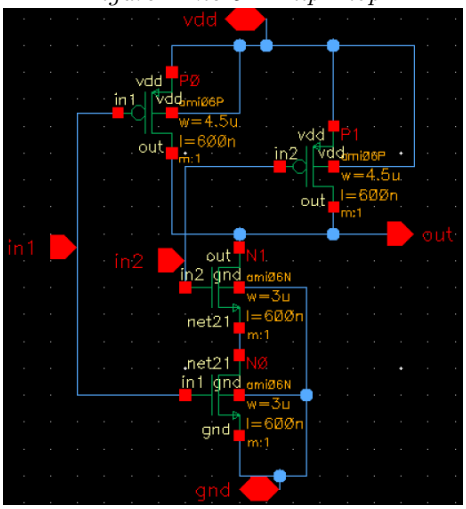

Figure. 2NAND gate

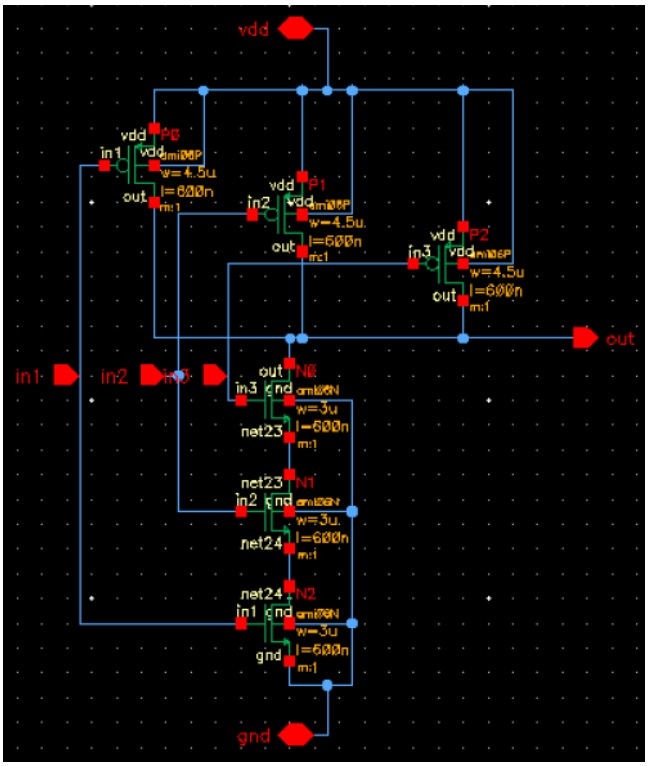

Figure. 3NAND gate

The counter works by cascading two JK Flip-Flops into one another, like the register, with two opposite clocks. Each JK Flip Flop takes in two inputs: J and K, and two clocks. A "master latch" takes the inputs and on clock 1 high, holds the and of the input with a feedback. The feedback is produced by the crossed 2NAND gates. A "slave latch" does the same thing, but with the opposite clock. Since the logic feeding into this master latch is a 3NAND that has one input hard-wired to V_{dd}, the result is a flip-flop that switches output on every clock tick. This output is used as the input to the second J-K Flip Flop, so the second J-K flip flop only changes input on every other clock-tick. With a 2-bit address, this is equivalent to a +1 counter.

Timing of the counter, like the decoder, is not as important because it has an entire clock cycle to switch, and even if it doesn't switch, the 3T dram cells do not need to be refreshed nearly as often as they are in this design, so it is OK. Therefore, we default the sizing to whatever sizings were needed by these gates elsewhere in the design. PMOS's are given $W = 4.5\mu$ and NMOS's $W = 3\mu$.

We see the testing operating:

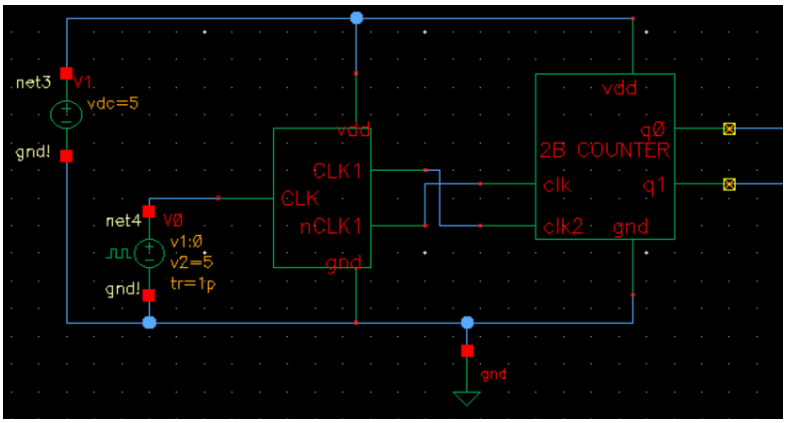

Figure. Counter verification schematic.

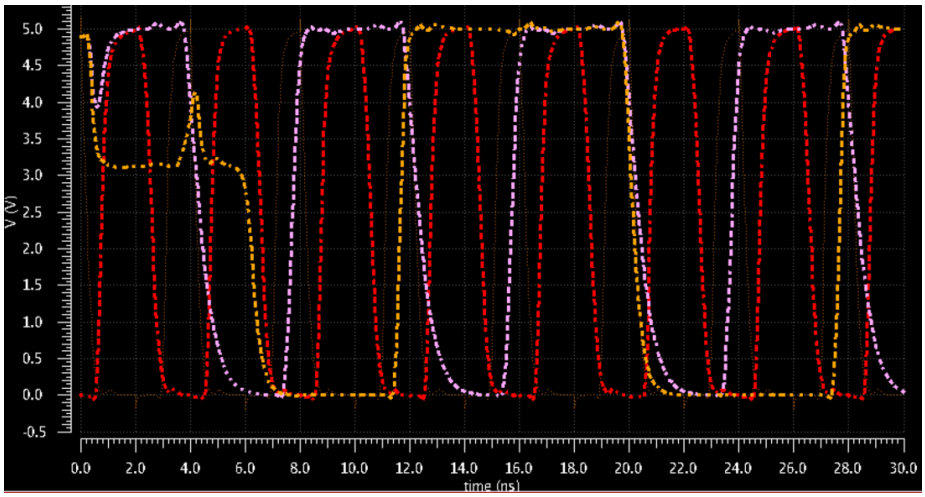

Figure. Counter output.

The pink and yellow lines are the output. We see that the pink goes up every clock cycle and the yellow every other.

3.9 Bitline Logic

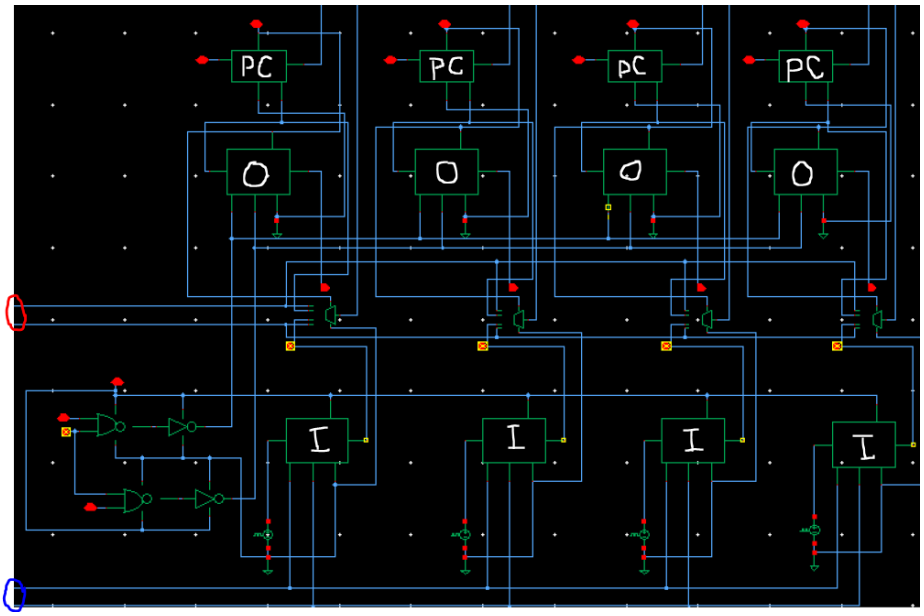

Figure. Bitline logic

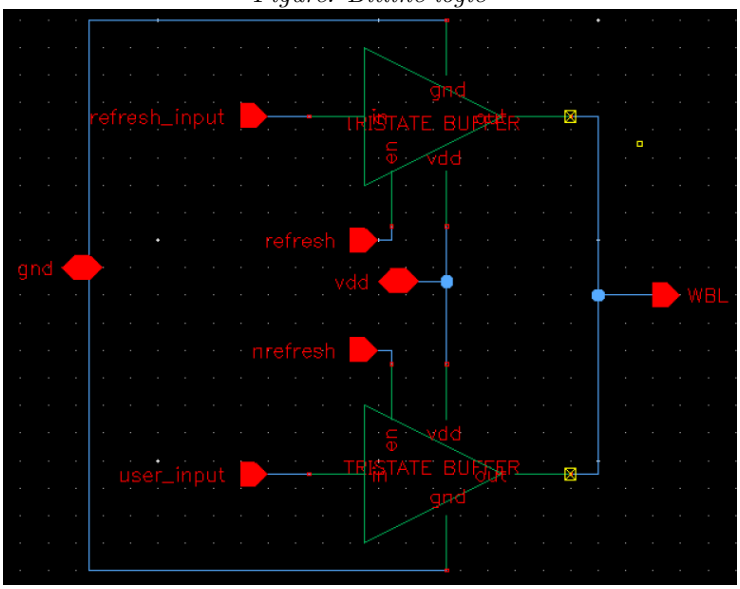

Figure. Write bitline logic multiplexer

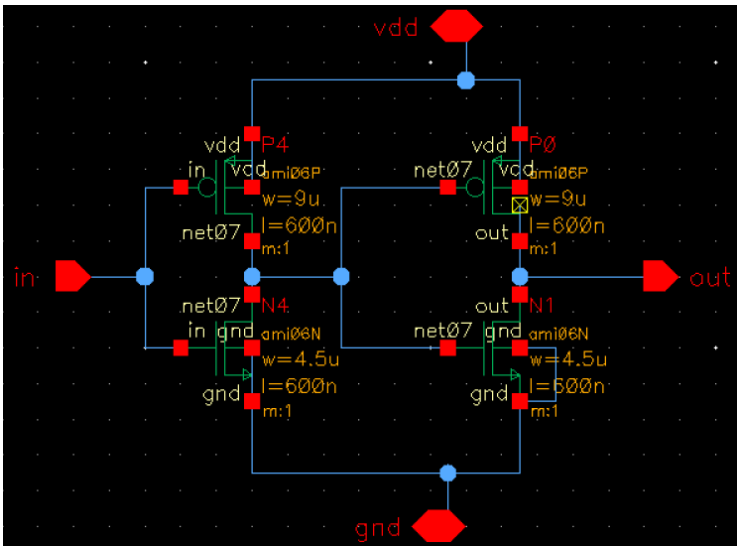

Figure. Buffer

The responsibility of the bitline logic is to (1-PC) pre-charge the bit lines, (2-O) provide output registers for the output to be held on, (3) provide logic for the WBL input, and (4-I) provide registers for the user input data. The pre-charge and register circuitry has already been detailed previously. Note the red circle on the left side of the first image in this section points out the refresh and not-refresh clocks, and the blue circle the register clocks.

A special multiplexer-type logic is used to determine whether to put the user input or the refresh data on the word bit line. The refresh input is the read-out from the pre-charge and output circuit, and goes on the WBL if the refresh clock is high. The user input is from the input data registers, and is passed through if we are in the not-refresh half of the clock cycle.

Also, the output registers have logic on the clocks determining that so long as the current action is not READ, the registers will not switch.

Since clock timing is used for every component of this subsection, I point out the usage of a buffer, sized to 9μ for the PMOS's and 4.5μ for the NMOS's, to restore the clock signal and reduce its capacitive load into intermediate chunks.

3.10 Wordline Logic

I designed a word line logic block in order to determine which word line (read or write, and which of the four words) to raise at any given time.

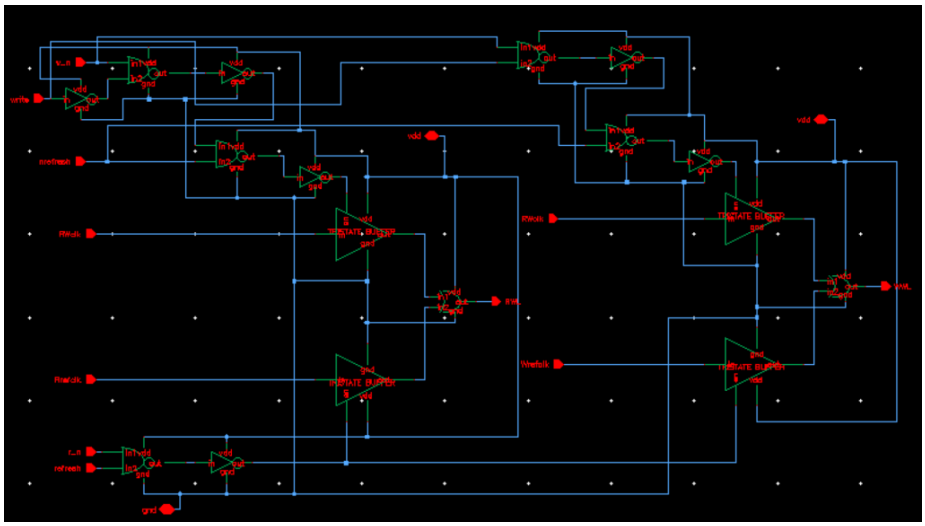

Figure. Wordline logic

This is designed such that every word gets one wordline logic block. The gate are arranged such that:
RWL ← (nrefresh & word & !WE & RWclk) OR (refresh & refreshword & Rrefclk)
WWL ← (nrefresh & word & WE & RWclk) OR (refresh & refreshword & Wrefclk)

We can verify the logic of the wordline block:

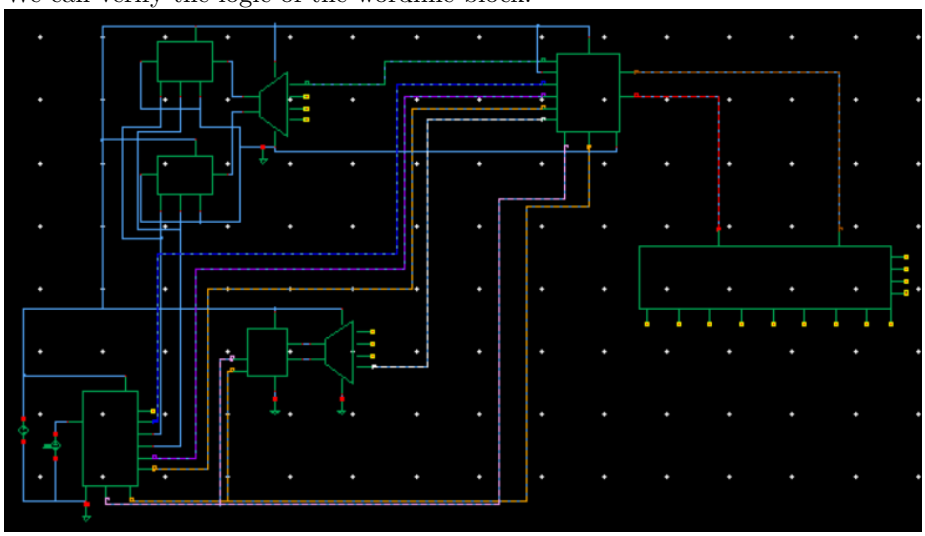

Figure. Verify wordline block schematic

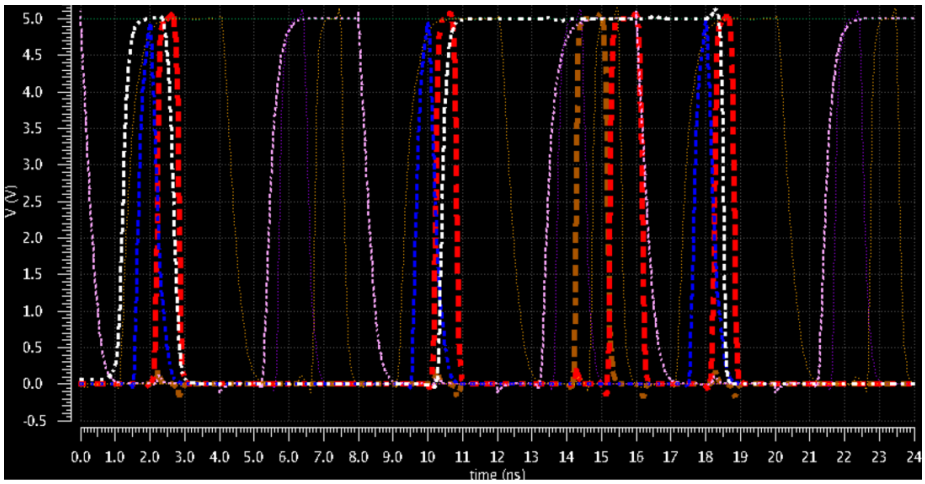

Figure. Verify wordline output

The red thicker line is the WWL, and the brown thicker line is the RWL. The setup is geared so that the *wordn* line of the input is always high, but the line telling it to refresh this word is based on the 2B counter. The appropriate outputs show, as the WWL raises right after the RWclk pulse (blue) during every not-refresh section, and in the refresh section (light pink), when it is this word's turn, the RWL pulses then the WWL pulses.

The wordline combined control is pictured below. User-input addresses are passed through a 2-4 decoder to determine which word line to raise. This information is passed to the wordline logic block. A 2-bit counter also outputs one address on each clock cycle to refresh the data of, and that information is also passed to the wordline logic block. The output clocks of the clock generator on the top left are also passed to the wordline logic block. All of these inputs together in the wordline logic blocks, on the right in the figure, determine which word lines to raise (or not raise) on which words at any given time.

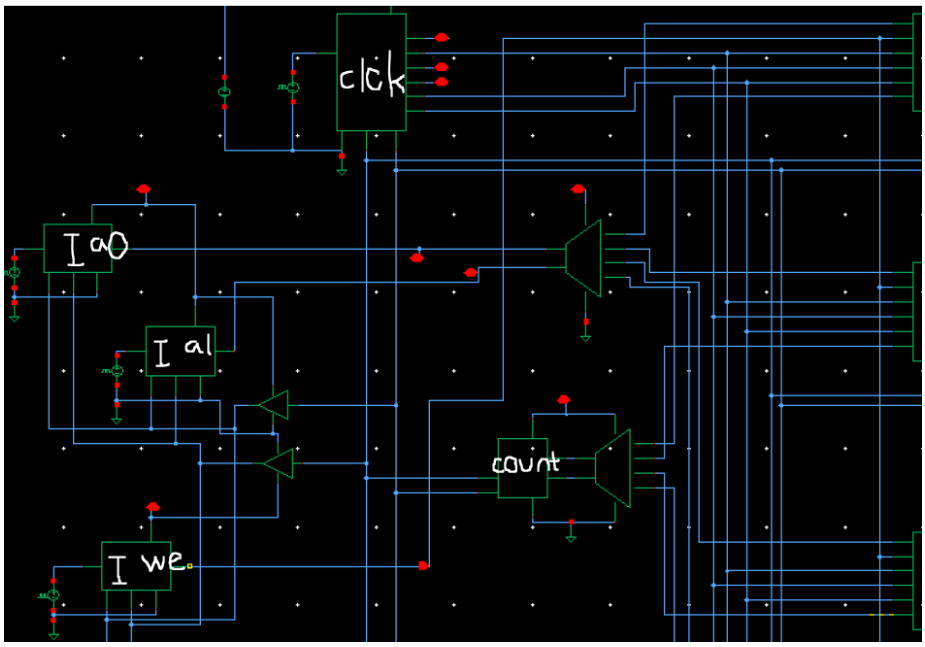

Figure. Wordline logic setup

4 Summary of Design Metrics

To calculate a figure of merit, we estimate with the equation:

$$FOM = 16 * BitcellArea \cdot Power \cdot Delay^2$$

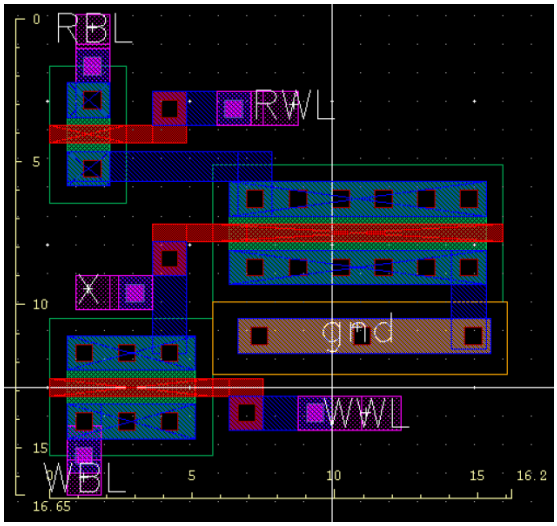

Figure. Layout for the cell. Square area: 16.65^2

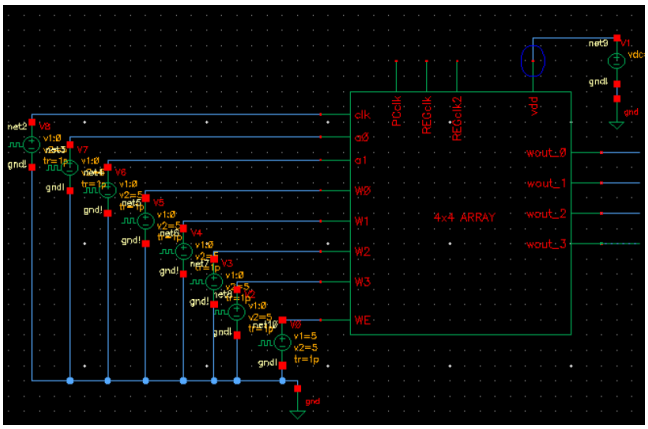

Figure. Test schematic used to measure current (power)

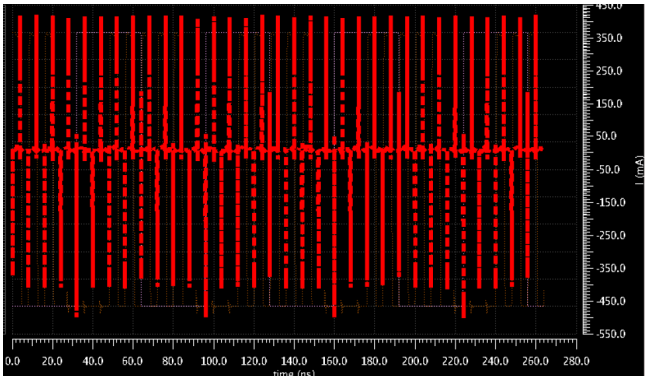

Figure. Current propagation over 4 rounds of writing 0's, then writing 1's.

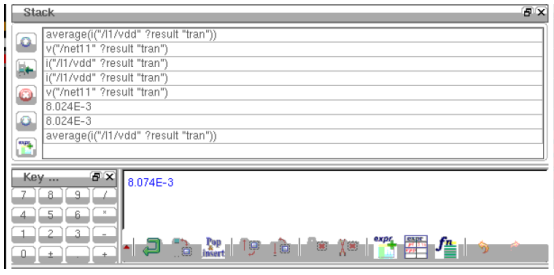

Figure. Average current over the above test case: 8.074mA

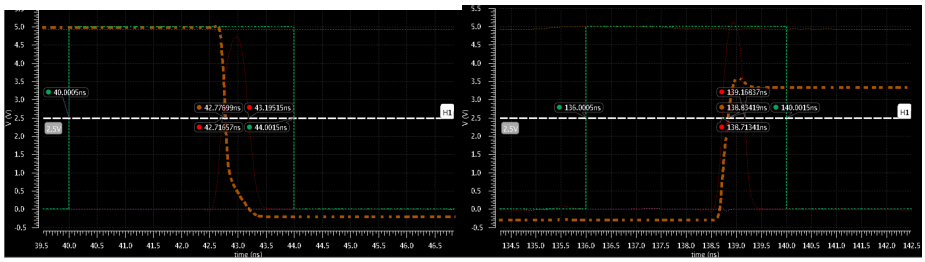

Figure. Delay of writing 0 and 1, respectively, to cell.

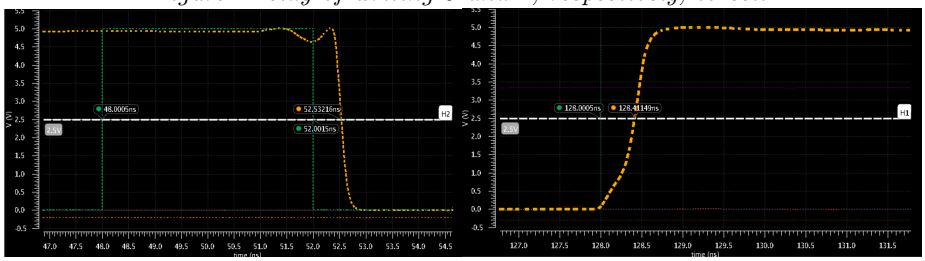

Figure. Delay of reading 0 and 1, respectively, from cell.

The worst delay is reading 0 from cell, with a delay of $4.5316ns$. (Note that the read delay of 1 from the cell appears much shorter because the register clocks are enabled during the entire read operation, and the wout signal is directly affected by the RBL that always drives up in the pre-charge phase.)

$$\begin{aligned} FOM &= 16 * ((16.65\mu)^2) * 8.043mA * (4.5316n)^2 \\ &= 7.326e - 28 \end{aligned}$$