Andrew Mattheisen amattheisen@yahoo.com 2134514711
Zhiyang Ong zhiyang@ieee.org 6004919412

EE577B - SRAM PART 2 Report

Andrew Mattheisen amattheisen@yahoo.com 2134514711
Zhiyang Ong zhiyang@ieee.org 6004919412

EE577B - SRAM PART 2 Report

TABLE OF CONTENTS

1 Introduction

In the reference by Kang and Leblebici (2003), Figure 10.3 gives the basic architecture design for arrays of RAM cells. Various architectural designs for SRAMs were considered by varying the dimensions of the SRAM block, or array of SRAM cells, which would in turn affect the design of the column and row decoders. The dimensions for such SRAM blocks were varied to achieve different aspect ratios that consequently result in different SRAM architecture designs. Since most chips are packages in squares, the area of the SRAM is determined to be the square of the longer of two horizontal dimensions. It is assumed that for higher aspect ratios, either the row or column decoder would be considerably fast, while the other would be very slow. Our design validates this assumption that SRAM designs with larger aspect ratios would have longer delay and much greater area.

2 Results of Architectural evaluation

The design objective chosen was to minimize delay at the expense of incurring more area usage.

The dependency of the area and performance on the dimensions of the SRAM block indicate that the SRAM needs to be designed so that both the column and row decoder are reasonably fast to avoid having either become the bottleneck in the SRAM design. This requires the aspect ratio to be minimum so that the parasitic capacitance and resistance from the wire routing in the decoders or SRAM block would not dominate the delay of the SRAM design. Based on this assumption, the authors attempted to estimate the delay of the SRAM using the method of logical effort (Sutherland et al. 1999 and Weste et al. 2005), and estimate the area using the following metric function; see Equation (1) for the metric function.

$$\text{Area } (\lambda^2) = (6 \times \Sigma_{\text{all transistors}} \text{ WL}) + 360 \times (\text{number of transistors}) \quad (1)$$

The Lyon-Schediwy decoder is used for decoding in 8 designs (Sutherland et al. 1999). The dimensions for these eight designs are given as follows in the product form of rows and columns: 1024×32, 521×64, 256×128, 128×256, 64×512, 32×1024,16×2048,1×32768. The delays in nanoseconds (ns) and their area usage in λ^2 are given in the first eight rows of Table 1. The area for the SRAM design with the 1×32768 SRAM block is much larger than the rest, since it does not have the row decoder and relies only the column decoder for all the decoding.

Additionally, the designs for using multiple-stage decoders, which include the use of a predecoder, are used to evaluate the following dimensions (given in λ) for the SRAM block design: 256×128, 64×512,128×256, and 128*256. The schematics for such architectural designs are given in Appendix A.

These calculations are semi-automated using MATLAB scripts; see [sram_design.m] for the main MATLAB script. It determines the path delay for the critical path of the SRAM, which is the path connecting the input flip-flops to the output flip-flops during a read operation. This requires determining information concerning the logical effort and parasitic for each stage in the critical path, and the main script would call other functions to do so. Upon completing execution of the MATLAB script, a plot of delay versus area is obtained. The delay is given in ns, since a pair of MATLAB and HSPICE scripts were used to calibrate the process technology to determine the value of τ in ns.

Consequently, the designs that yield delays much smaller than the rest include 256 λ $\times$128 λ and 128λ $\times$ 256λ. These designs have the same aspect ratio, which is much smaller than other architectural designs. This is partially due to the smaller delay through the parasitic capacitances and resistances in the long wires interconnecting the modules, and larger aspect ratios indicate that either the row or column decoder would have a large number of outputs, which imply a larger loading effect at the output logic of the decoder stage since the decoder has to drive more load at the output.

Therefore, the selected SRAM design uses the Lyon-Schediwy decoder for its row and column decoders, and a SRAM block of 128λ $\times$ 256λ, since it has the minimum delay, and has one of the smallest area among the considered SRAM designs.

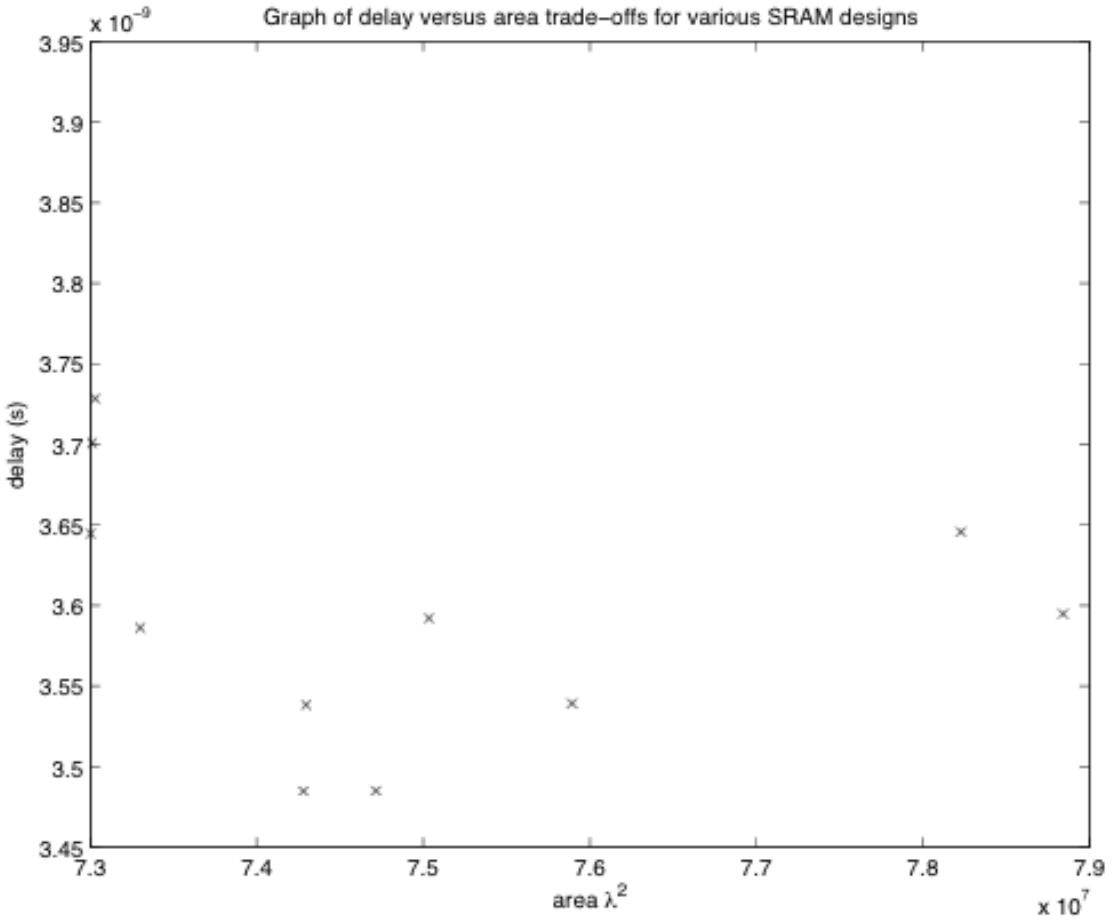


Figure 1 Plot of delay versus area trade-off for various SRAM designs

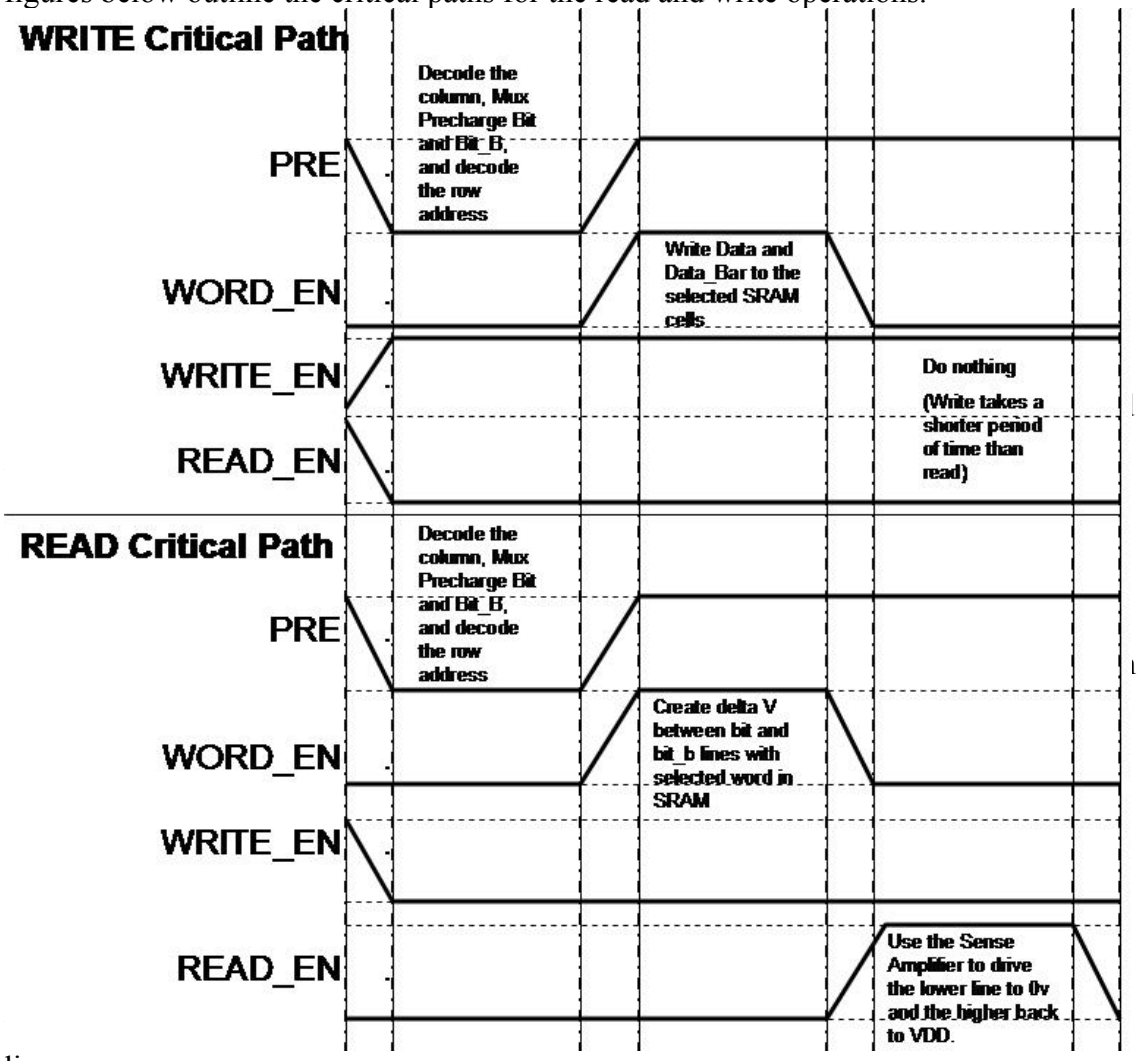
Table 1 Table of values for delay and their area trade-off for various SRAM designs

Delay through the SRAM design (ns)	Area of SRAM design (λ^2)
3.59487825072219	78841504
3.53925202627758	75893431
3.48504501879838	74716366
3.48487456032927	74280421
3.53846647730621	74296060
3.59209429014602	75033619
3.64573715655573	78228778
3.86033490178664	1073741824
3.72839638752604	73030613.6574197
3.70087624882535	73010855.4854321
3.64467501372661	73003264
3.58626598225101	73300461.8924385

For each row, the corresponding SRAM designs are given as follows: row 1 uses a 1024×32 SRAM macro block and a 10-bit Lyon-Schediwy row decoder; row 2 uses a 521×64 macro block and a 9-bit Lyon-Schediwy row decoder; row 3 uses a 256×128 macro block and a 8-bit Lyon-Schediwy row decoder; row 4 uses a 128×256 macro block and a 7-bit Lyon-Schediwy row decoder; row 5 uses a 64×512 macro block and a 6-bit Lyon-Schediwy row decoder; row 6 uses a 32×1024 macro block and a 5-bit Lyon-Schediwy row decoder; row 7 uses a 16×2048 macro block and a 4-bit Lyon-Schediwy row decoder; row 8 uses a 1×32768 macro block and a 15-bit Lyon-Schediwy column decoder. Rows 9 till 12 are discussed in Appendix A.

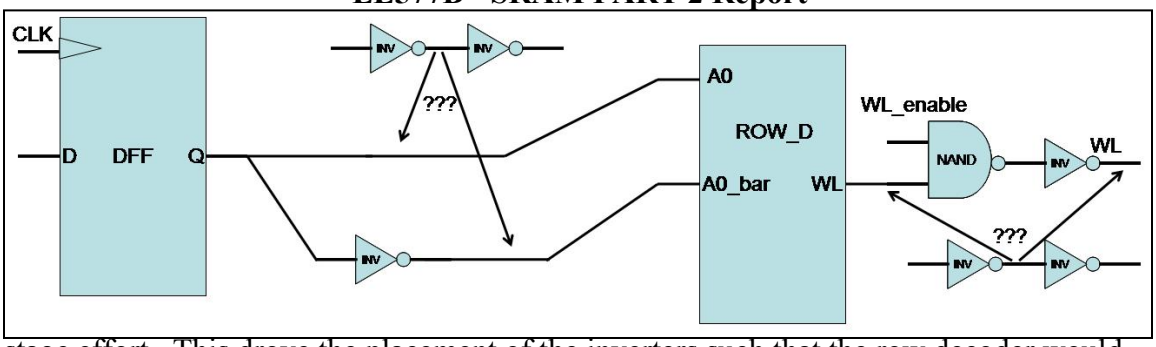
3 Delay Estimation

The delay of the SRAM is based on the speed at which it can read and write. The read or write operation may be faster, but the delay of the SRAM is the slower of the two. The figures below outline the critical paths for the read and write operations.



line.

The above paragraphs are summarized by Figure 2 below.



stage effort. This drove the placement of the inverters such that the row decoder would be at least as large as the smallest possible LS decoder. The results of the MATLAB script indicate that on the optimal path, there should be 3 inverters between the FF and the LS decoder. This allows the optimal path to contain the FF, three inverters, the LS decoder, a NAND gate, and a final inverter. This optimizes the delay on the path to the A0_bar input of the row decoder from the flip flop!

Unfortunately, the path through the A0 input of the LS row decoder must have an even number of stages, and therefore will be suboptimal. An extra inverter will be added to this path, causing an extra delay of pinv on the path to the A0 input. This makes the path through the A0 input of the row decoder the critical path. Figure 3 shows the gates arranged between the FF and the WL to get the optimal number of stages.

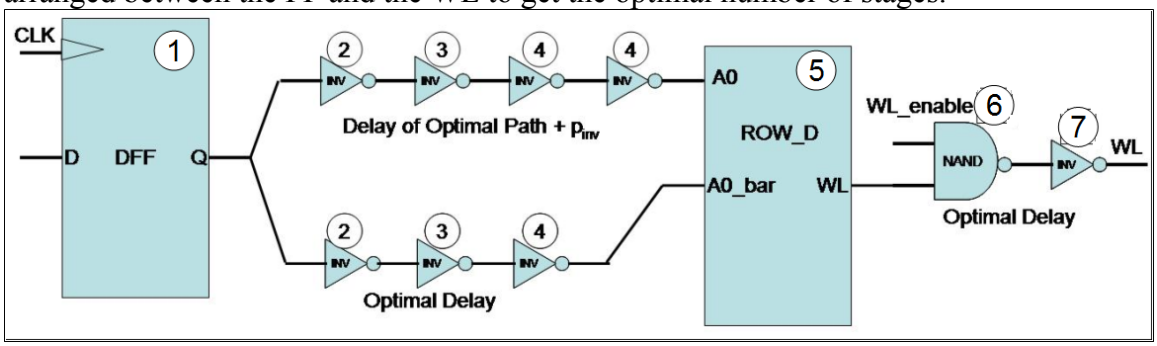


Figure 3 - Critical Path Optimization for WL driver

The MATLAB script Calcs_A.m determines the optimal widths of the NMOS and PMOS transistors in the gates on this path. Table 2 below shows the widths of the gates calculated to minimize the delay on the optimal path. The Circles in Figure 3 correspond to the numbering in Table 2.

Table 2 - Transistor Sizes

(1) FF
WNgate1 = 2
WPgate1 = 4
(2) Branched inverter (bi=2)
WNgate2 = 2.8
WPgate2 = 8.5
(3) Inverter
WNgate3 = 10.8
WPgate3 = 32.4
(4) Inverter
WNgate4 = 40.9
WPgate4 = 122.7
(5) 7 bit LS Decoder
WNgate5 = 1.4
WPgate5 = 4.2
(6) AND gate
WNgate6 = 8.5
WPgate6 = 12.7
(7) Inverter
WNgate7 = 16
WPgate7 = 48.1

The Matlab script also computes the theoretical delays using logical effort and the RC pi-model for the capacitance of the load. The gate capacitance was much much greater than the parasitic capacitance of the lines running between the PMOS and NMOS for each input signal to the LS decoder, and therefore the parasitics of lines internal to the LS decoder were ignored. The delays calculated do not include the entire clock to Q internal delay of the flip flop, which would make them slightly optimistic. However, the delay of the pi model was added directly to the delay of the driving circuit assuming the load was a lumped capacitance, which counts the delay of inverter 7 twice, and therefore makes our final estimate slightly pessimistic. Table 3 below summarizes the delay calculations for the WL.

Table 3 – Delay of Sub-Optimal Path from DFF to WL

Delay of diver – WL as Lumped Cap	WL Delay using Pi Model	Total Delay
6.542e-10 seconds	5.968e-11 seconds	7.139e-10 seconds

3.2 Delay Estimation CLK to Bit Precharge (Not on Critical Path)

A PMOS is used to precharge each BIT and BIT_B line of the SRAM Macro selected by the column decoder. Therefore, once the column decoder has selected the appropriate SRAM macro, the delay to precharge the bit and bit_b lines can be calculated using a pi model as shown in Figure 4 below.

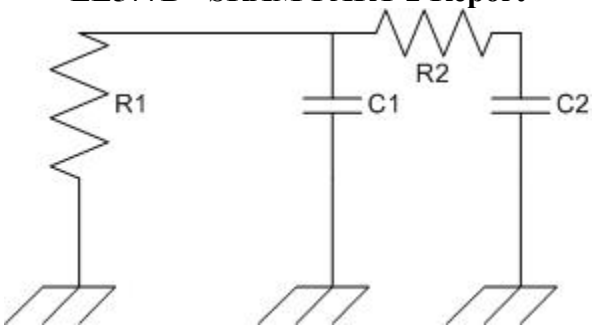


Figure 4 - Pi Model with Driving Transistor Resistance

R1 is the effective resistance of the PMOS transistor. R2 is the resistance of the bit line. C1 and C2 are each ½ of the capacitance of the bit line. Each bit line runs the length of 128 rows of SRAM cells. The detailed calculations for the bit line wire capacitance and the additional bit line capacitance due to the 128 NMOS diffusions connected to each line are shown in the matlab file Calcs_A.m.

The theoretical delay to precharge the bit and bit_bar lines was calculated as 1.636e-11 seconds (see matlab script Calcs_A.m for details). Since this is far less than the delay from the address bits to the WL, precharge delay is not on the critical path.

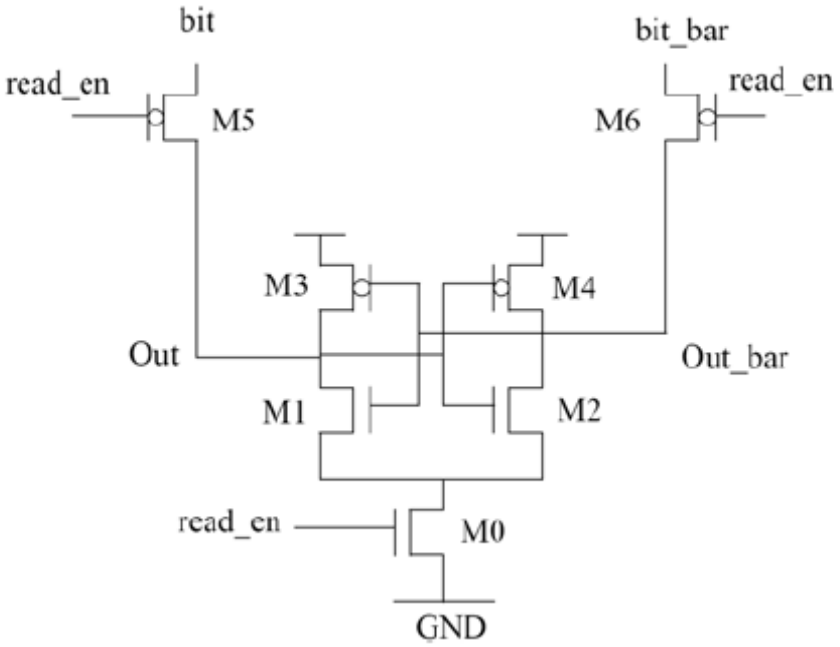
3.3 Delay Estimation WL to Bit line voltage drop of delta V (Part of Critical Path)

The delay to drop the Bit line by 0.15V is approximated by calculating the bit line delay, which is the delay to drop the signal 0.63*VDD, and multiplying by a factor of 0.15/(0.63*VDD) = 0.132. This assumes that the voltage drops linearly with time initially for the duration of the first 0.15V drop, which is reasonable to expect from an RC circuit such as this one.

The generic circuit shown in Figure 4 above can be used to calculate the delay of the WL to Bit line. The value calculated will then be multiplied by 0.132 (see previous paragraph) to obtain the delay to drop bit line by 0.15V. This calculation is performed by the Calcs_A.m script, and the theoretical delay of the WL to Bit dropping 0.15V is 2.165e-12 seconds.

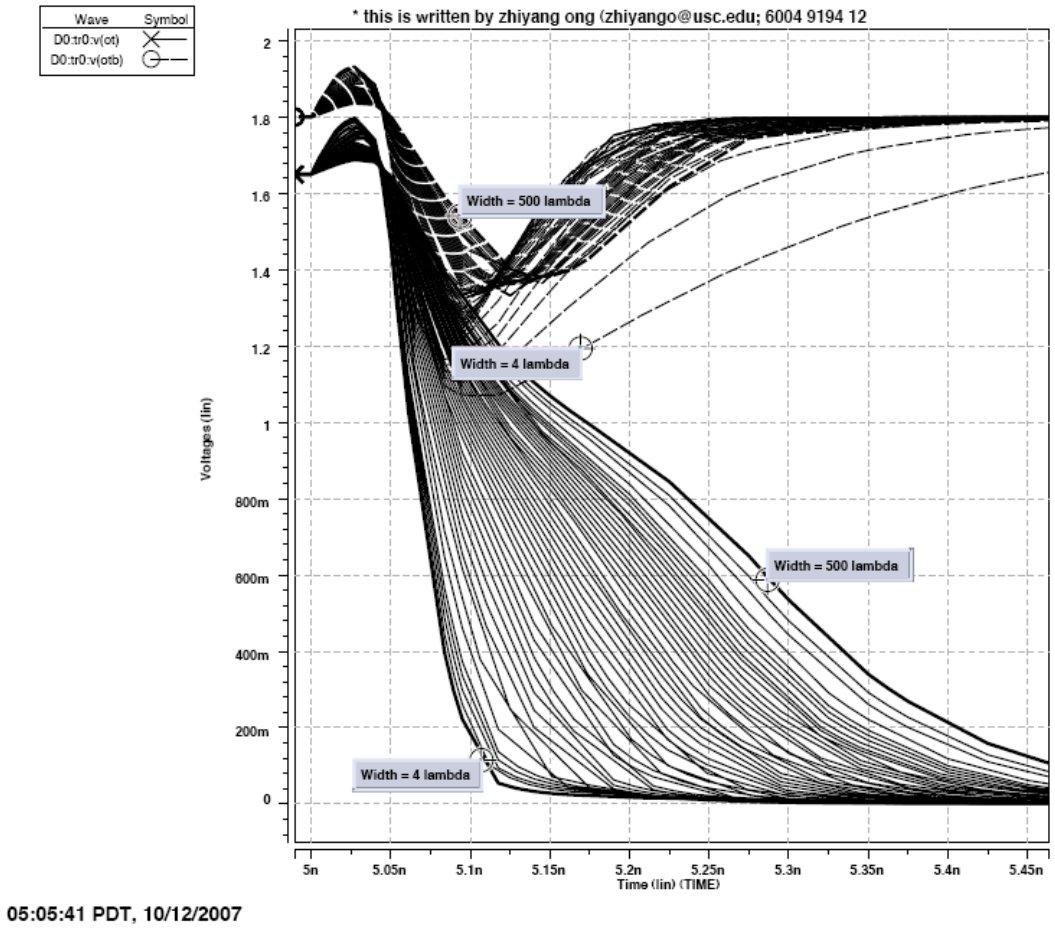
3.4 Delay Estimation of Sense Amp (Part of Critical Path)

The schematic of the sense amplifier circuit is shown below:

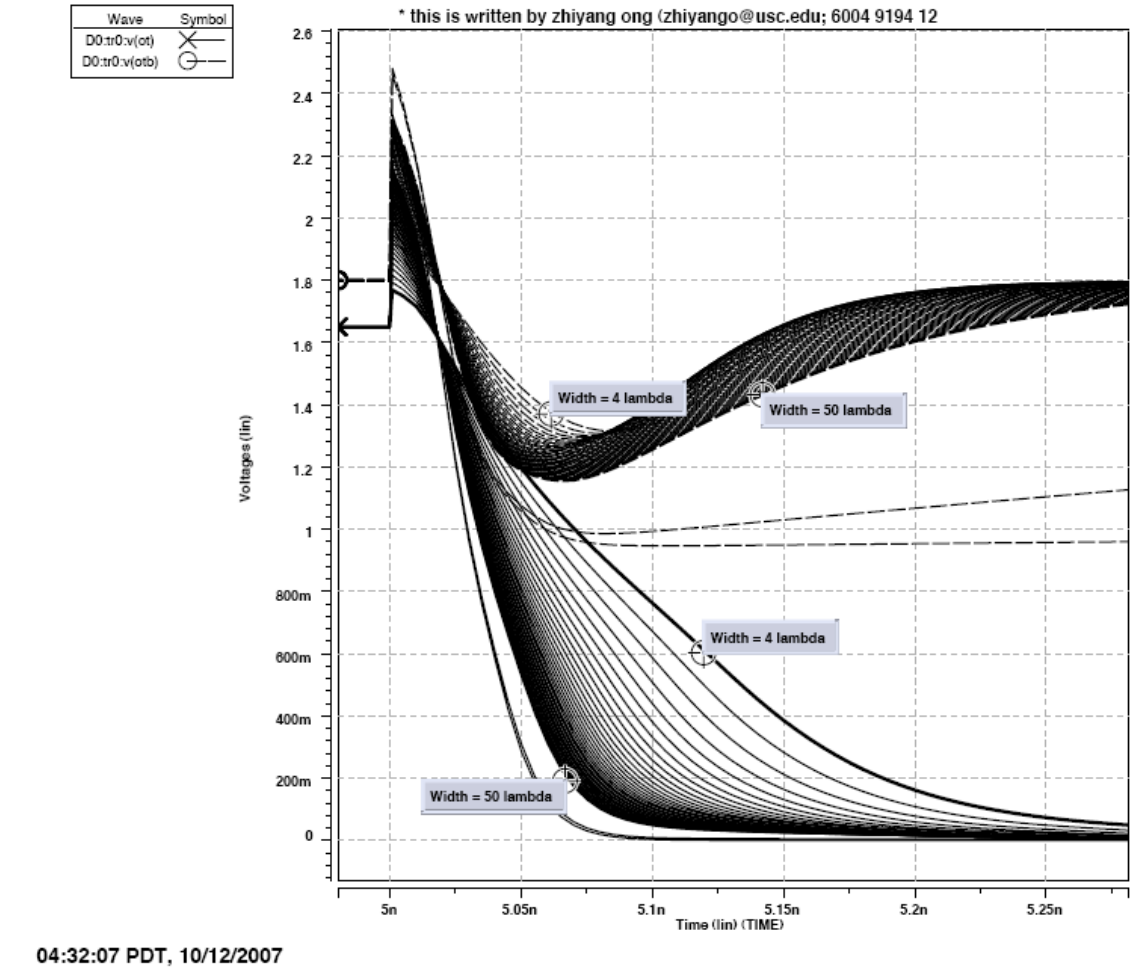


The delay of the sense amp is the delay from the rising edge of the read_en signal to the falling edge of either Out or Out_bar. For calculations, we will assume that Out is the falling node. Therefore at T=0, the voltage of Out is VDD-0.15V and the voltage of Out_bar is VDD. The delay for the sense amp is equal to the time it takes for the NMOS with drain attached to OUT to pull out to ground while fighting the PMOS. We modeled the sense amp in spice and using the sweep command for various PMOS and NMOS sizes and graphed the voltages at Out and Out_b over time (see the two graphs below).

EE577B - SRAM PART 2 Report



Sizing M3 and M4 in Sense Amp



Sizing M1 and M2 in Sense Amp

From the first graph above, it is clear that using the smallest possible PMOS is best. Therefore we will use PMOS transistors of width 4 lambda. Using the 4 lambda wide PMOS transistors for M3 and M4, the size of NMOS transistors M1 and M2 was then varied. The second graph clearly shows that increasing the NMOS width higher than 50*lambda does not reduce the delay of the sense amplifier drastically (the two lines to the left of the line for width =50 lambda are for 500 lambda and 5000 lambda), therefore we will use NMOS transistors M1 and M2 of width 50 lambda and expect the delay of the sense amp to be about 2.5e-11s.

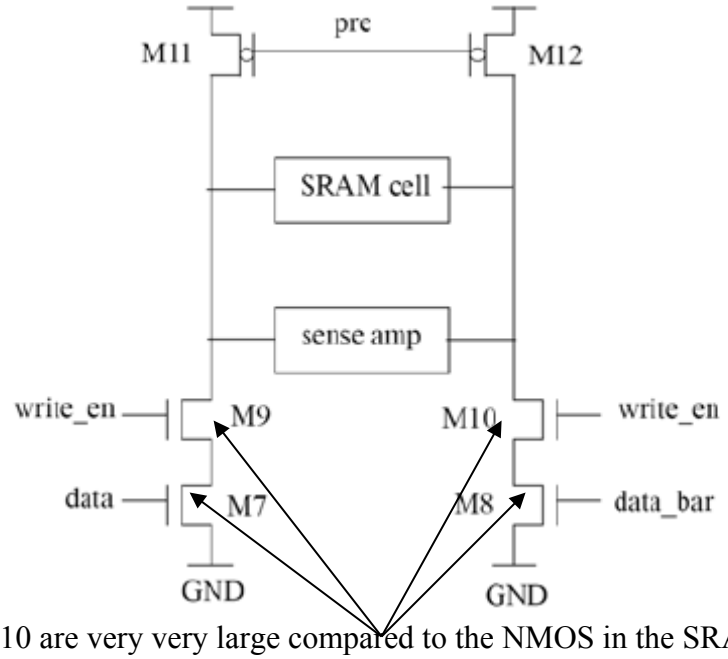
3.5 Total Critical Path Delay for Read

The total critical path delay for read is the sum of:

Delay from CLK to WL:	7.14e-10s from above
Delay from WL to Bit falling .15V:	2.15e-12s from above
Delay from read_en to Out:	2.5e-11s from above
The setup time for the output FF	200e-12s assumed setup time for FF
TOTAL	**9.4115e-10s**

3.6 Comparing Write Delay to Read Delay

The write delay will have the same delay for the row decoder and precharge circuit, but will be faster than the read circuit because it uses large NMOS transistors (attached to Write_En, Data, and Data_bar to pull down the bit or bit bar line as shown below, and this can happen in parallel with the row address decoding. Therefore, detailed calculations of the Write path delay are not necessary.



(M7-M10 are very very large compared to the NMOS in the SRAM cell)

4 Floorplanning

4.1 Dimentions

EE577B - SRAM PART 2 Report

Total Height (Entire SRAM) = 64*(40 * lambda)*3+ 110*lambda = 7790 lambda

Total Width (Entire SRAM) = 32*(40 * lambda)*6+ 1024*lambda = 8704 lambda

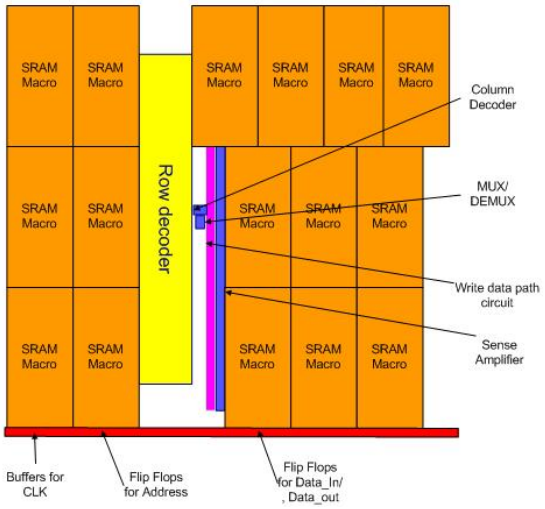
Column decoder = (2*3*10*lambda+w*(1+2+4)*4*lambda)*(2^3*8*lambda)=228 lambda * 64 lambda

Column MUX/DEMUX = 8*10 lambda * 40 lambda = 80 lambda * 40 lambda

Sense amplifier = 32*2*(36*2+4) lambda * 40 lambda = 4864 lambda * 40 lambda

Write data path circuit = 32*2(34*2+4) lambda * 40 lambda = 4608 lambda * 40 lambda

SRAM Macro = 32*(40*lambda) * 64*(40*lambda) = 1280*lambda * 2560*lambda

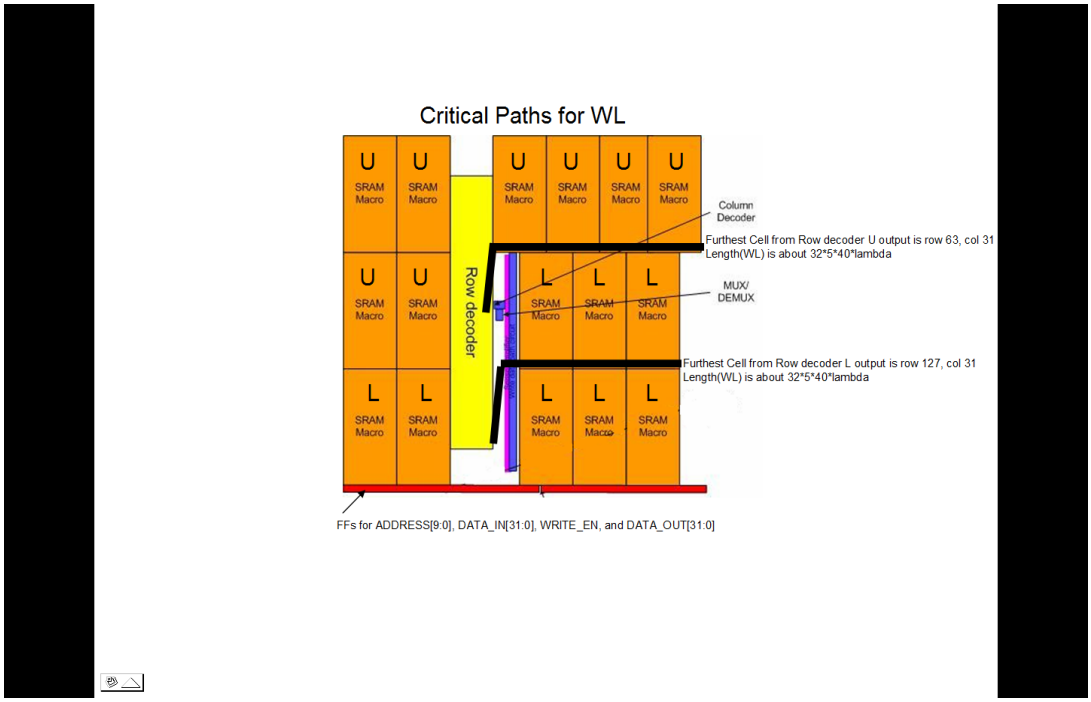


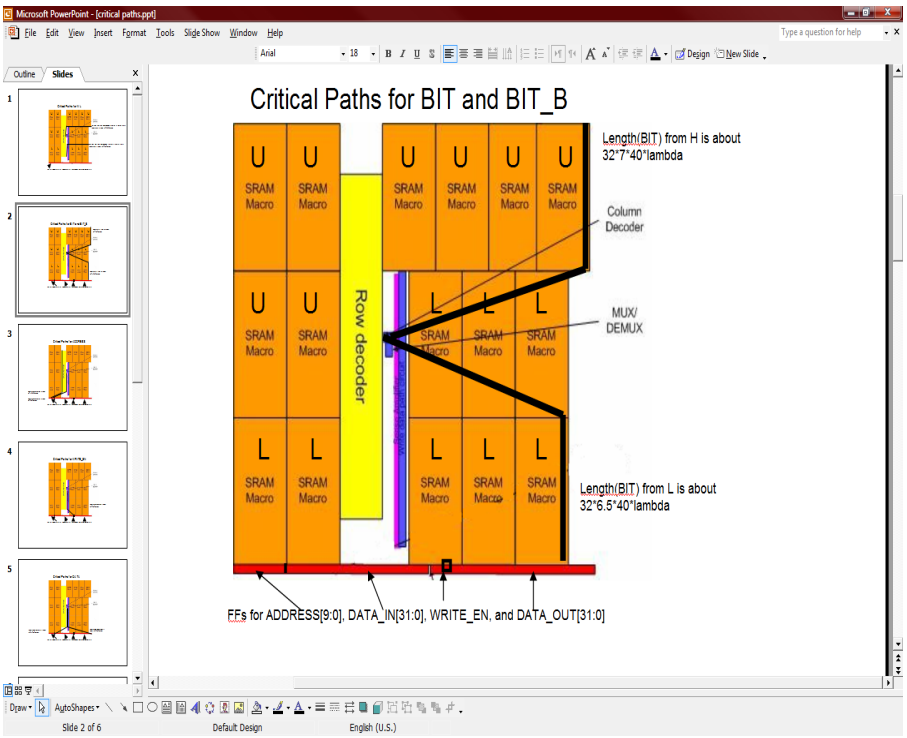
From Cadence Virtuoso, the area of a flip-flop is 10.85u*10.9u = about (10.9u)^2
Cascade 75 flip-flops for 75 signals together => (75*10.9u)*10.9u
Input/Output buffers: 8284 * 109 lambda square =
Clock buffer = 10.9u * 15.2u = 109 lambda * 152 lambda
All buffers = (152+75*109) * 109 lambda^2 = 8327 lambda * 109 lambda

w is about 6
Row decoder = (2*7*10*lambda+w*(1+2+4+16+32+64+128)*4*lambda)*(2^7*8*lambda)=6068 lambda * 1024 lambda

4.2

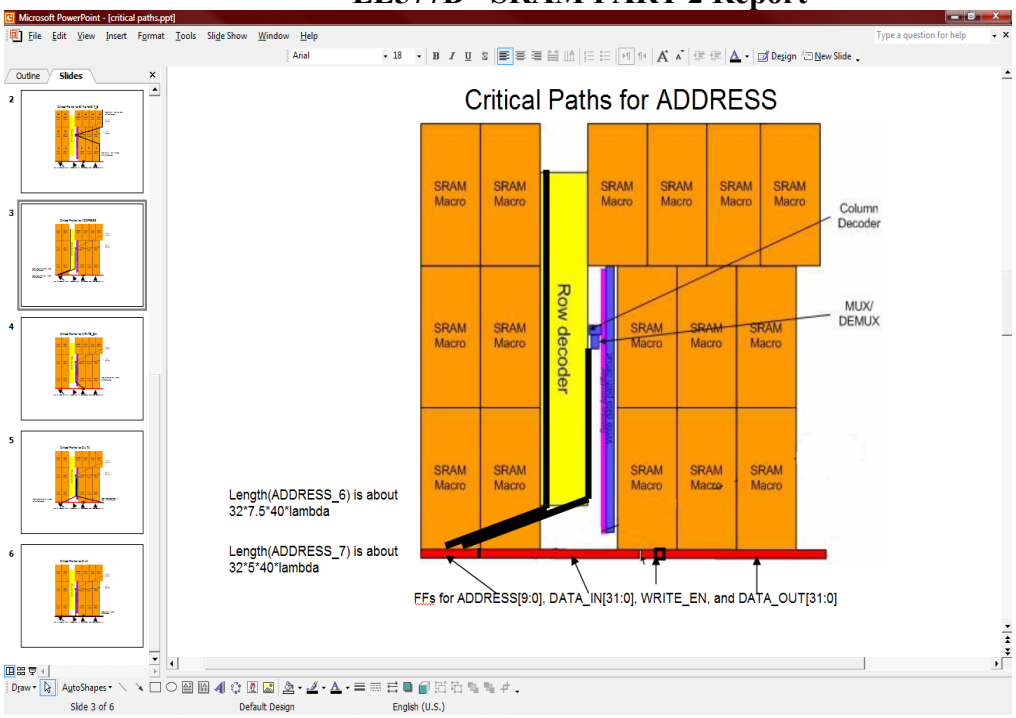
Critical Path Lengths

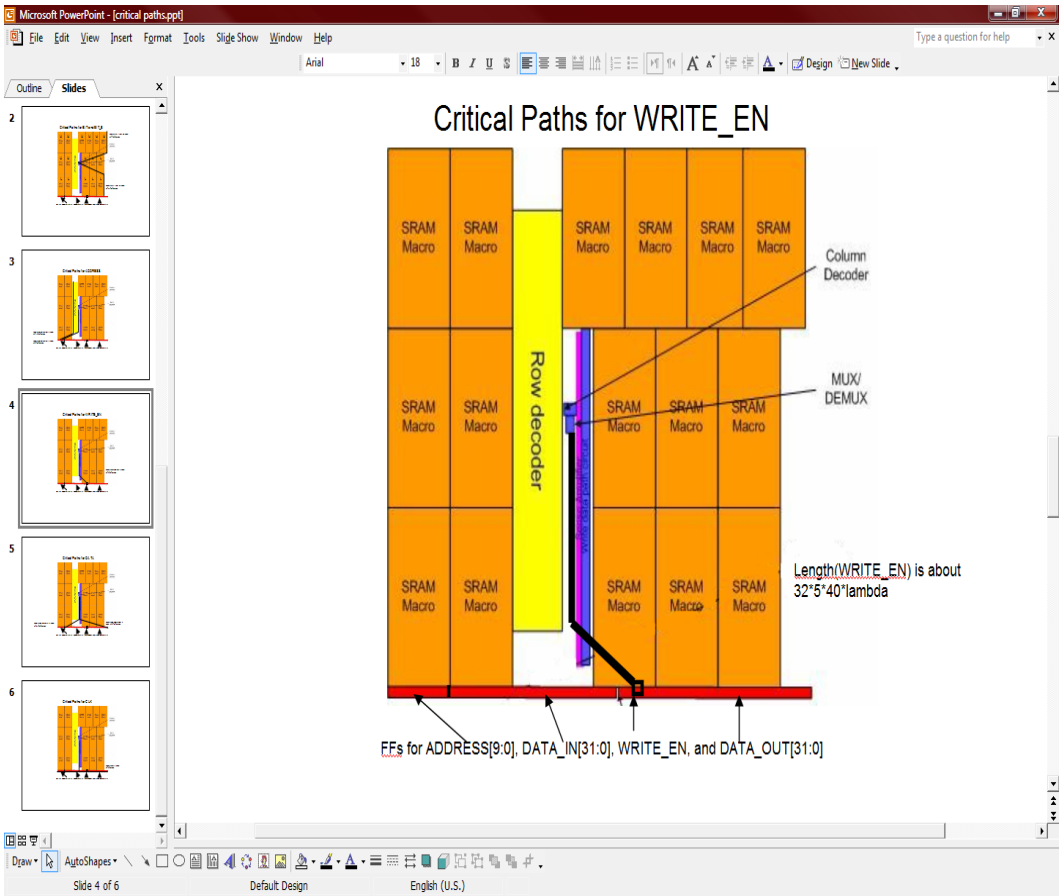


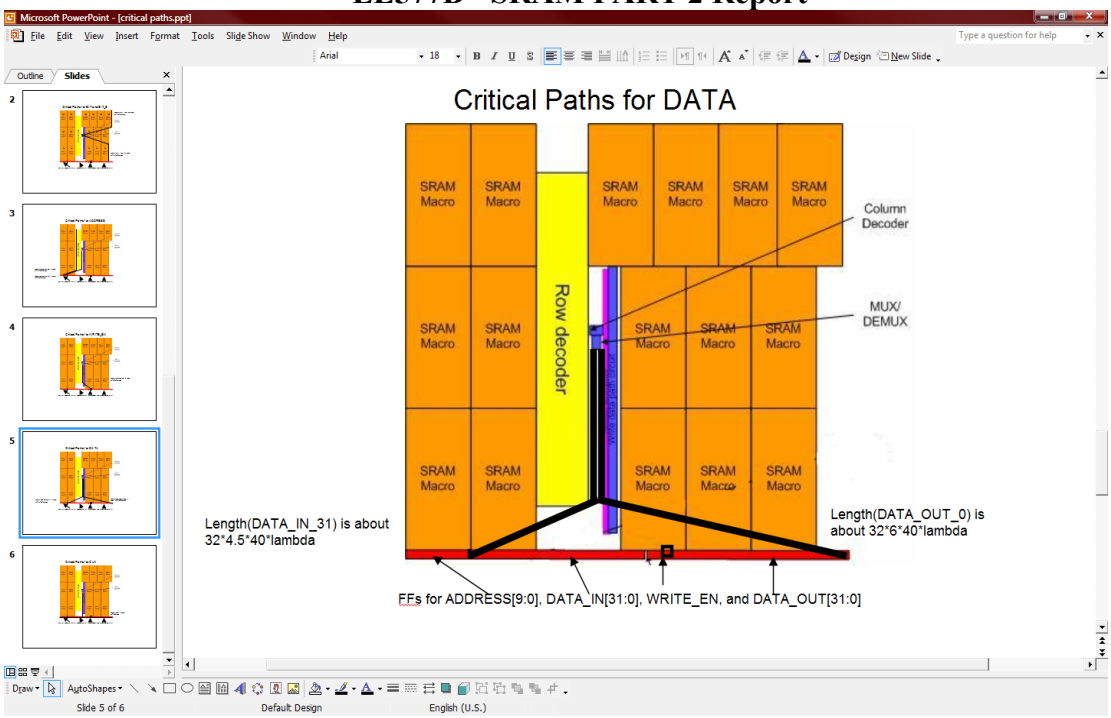


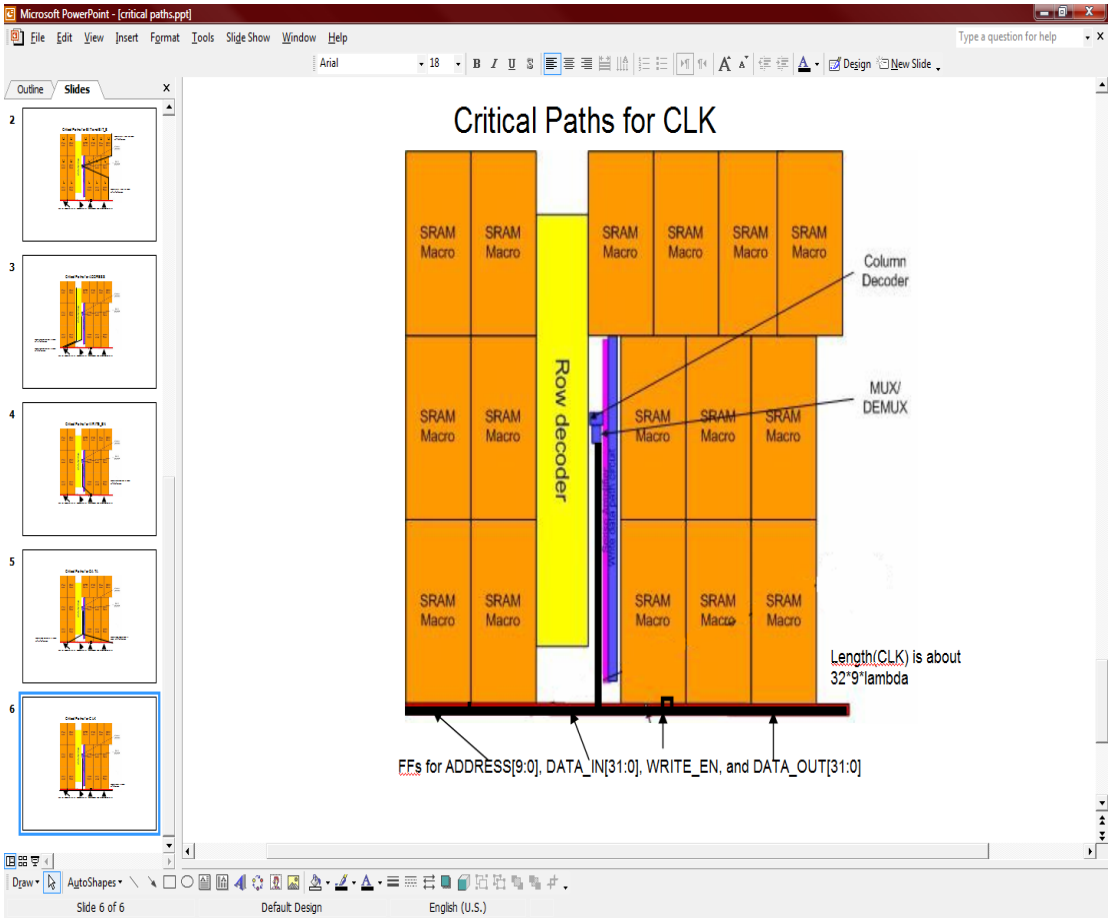
Andrew Mattheisen amattheisen@yahoo.com 2134514711
Zhiyang Ong zhiyang@ieee.org 6004919412

EE577B - SRAM PART 2 Report






Microsoft PowerPoint - [critical paths.ppt]
Critical Paths for DATA
SRAM Macro
Row decoder
Column Decoder
MUX/ DEMUX
Length(DATA_IN_31) is about 32*4.5*40*lambda
Length(DATA_OUT_0) is about 32*6*40*lambda
FFs for ADDRESS[9:0], DATA_IN[31:0], WRITE_EN, and DATA_OUT[31:0]
Slide 5 of 6
Default Design
English (U.S.)


Microsoft PowerPoint - [critical paths.ppt]
Critical Paths for CLK
SRAM Macro
Row decoder
Column Decoder
MUX/ DEMUX
Length(CLK) is about 32*9*lambda
FFs for ADDRESS[9:0], DATA_IN[31:0], WRITE_EN, and DATA_OUT[31:0]
Slide 6 of 6
Default Design
English (U.S.)

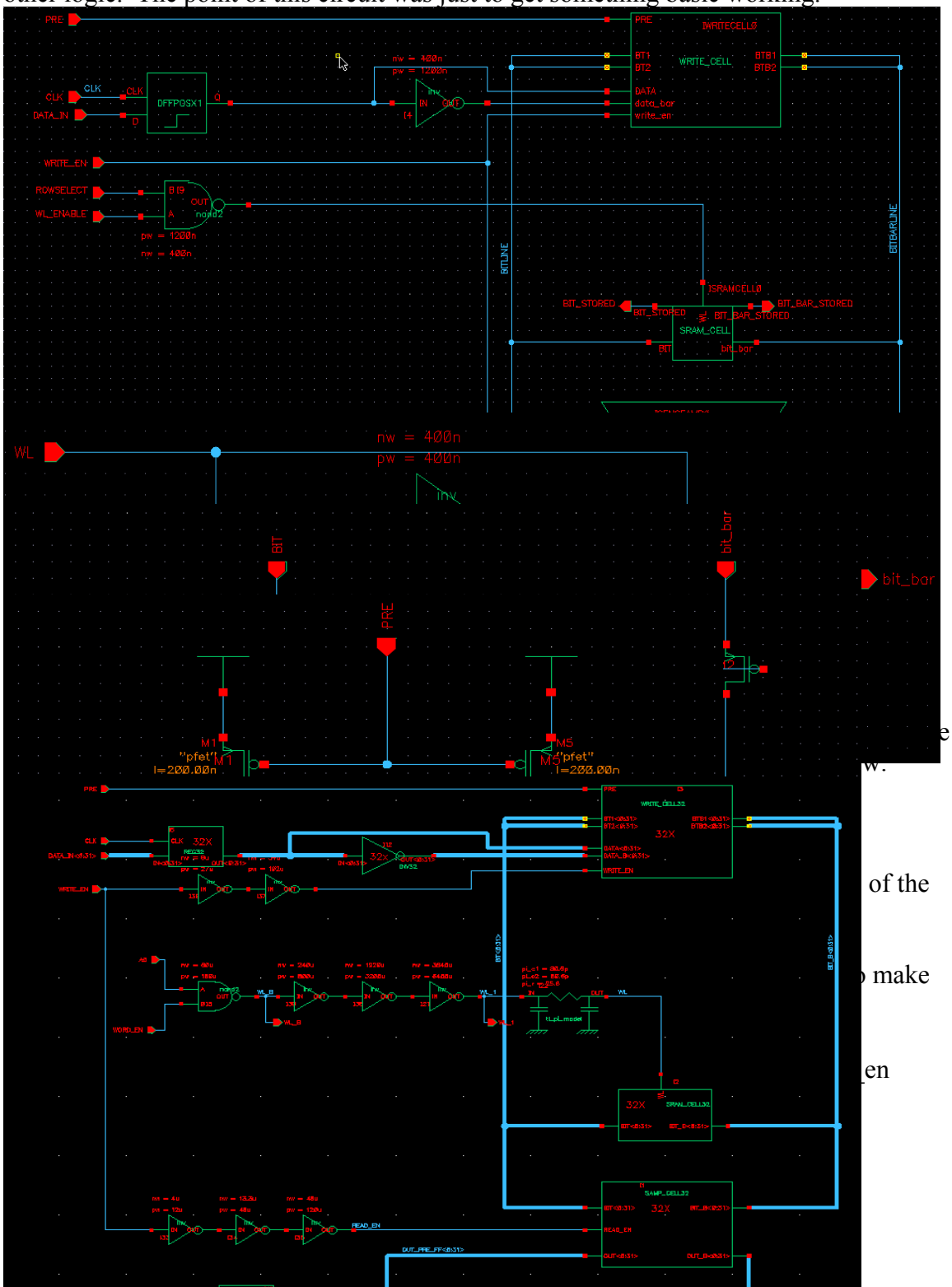
Andrew Mattheisen amattheisen@yahoo.com 2134514711
Zhiyang Ong zhiyang@ieee.org 6004919412

EE577B - SRAM PART 2 Report

5 Schematic Design Approach

The overall goal of the schematic design was to design a very simple circuit, simulate it and verify that it worked, then increase the complexity and simulate again until the entire circuit simulated correctly. A hierarchical approach to schematic design was used to keep the schematics compact and manageable.

Here is the first fully functional SRAM built for this project. The schematic contains a single bit of SRAM, a sense amp, a write circuit, a read circuit (sense amp), and minimal other logic. The point of this circuit was just to get something basic working.

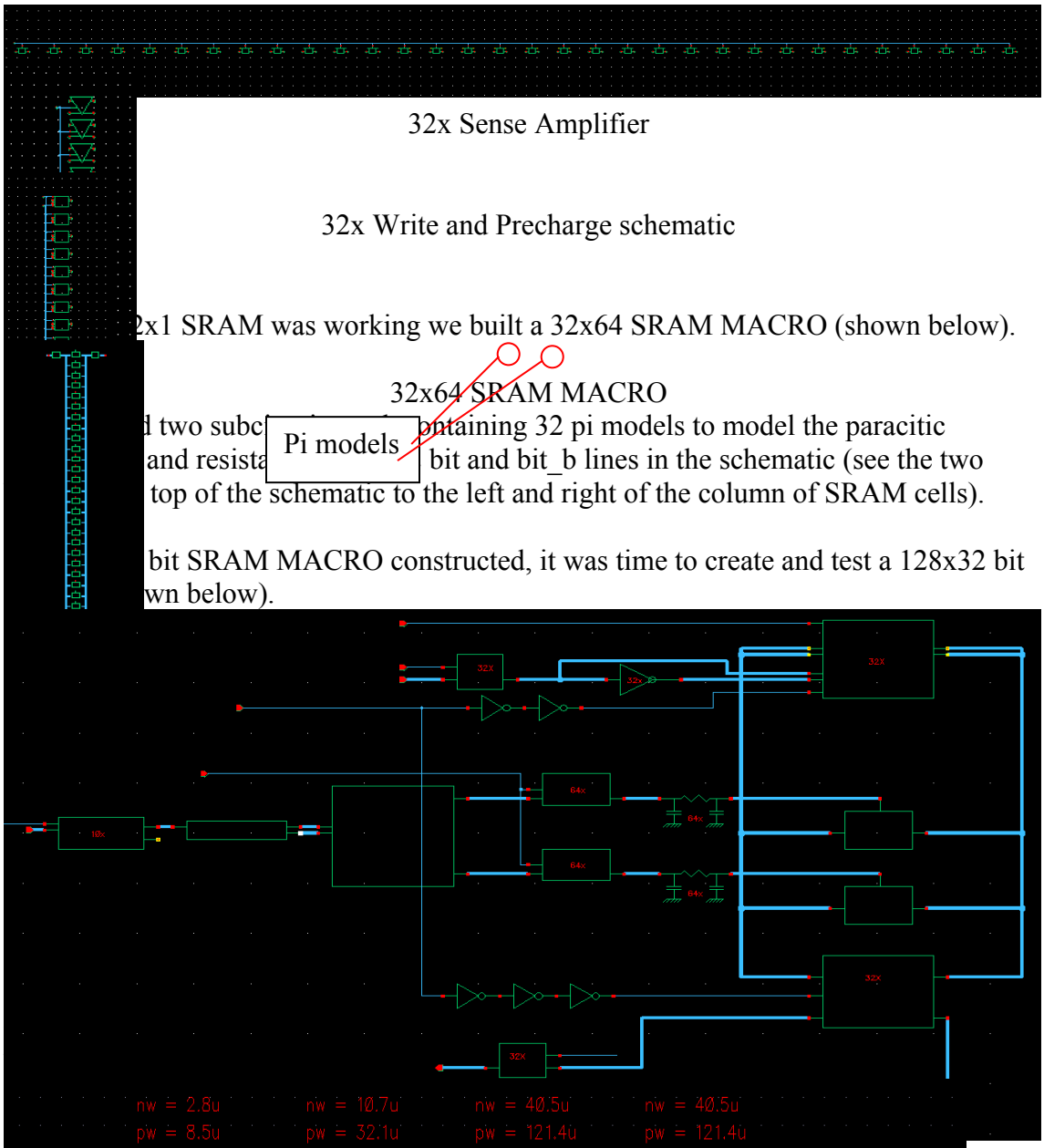


Andrew Mattheisen amattheisen@yahoo.com 2134514711
Zhiyang Ong zhiyang@ieee.org 6004919412

EE577B - SRAM PART 2 Report

- A group of 32 Write cells, all with the pre and write_en inputs tied to pre and write_en respectively.
- A group of 32 FFs (not shown below)
- A group of 32 inverters (not shown below)

To avid confusion "32x" was included on the symbol of cells containing 32 of something. A few examples of sub circuits for the 32x1 SRAM are provided below:



32x Sense Amplifier

32x Write and Precharge schematic

2x1 SRAM was working we built a 32x64 SRAM MACRO (shown below).

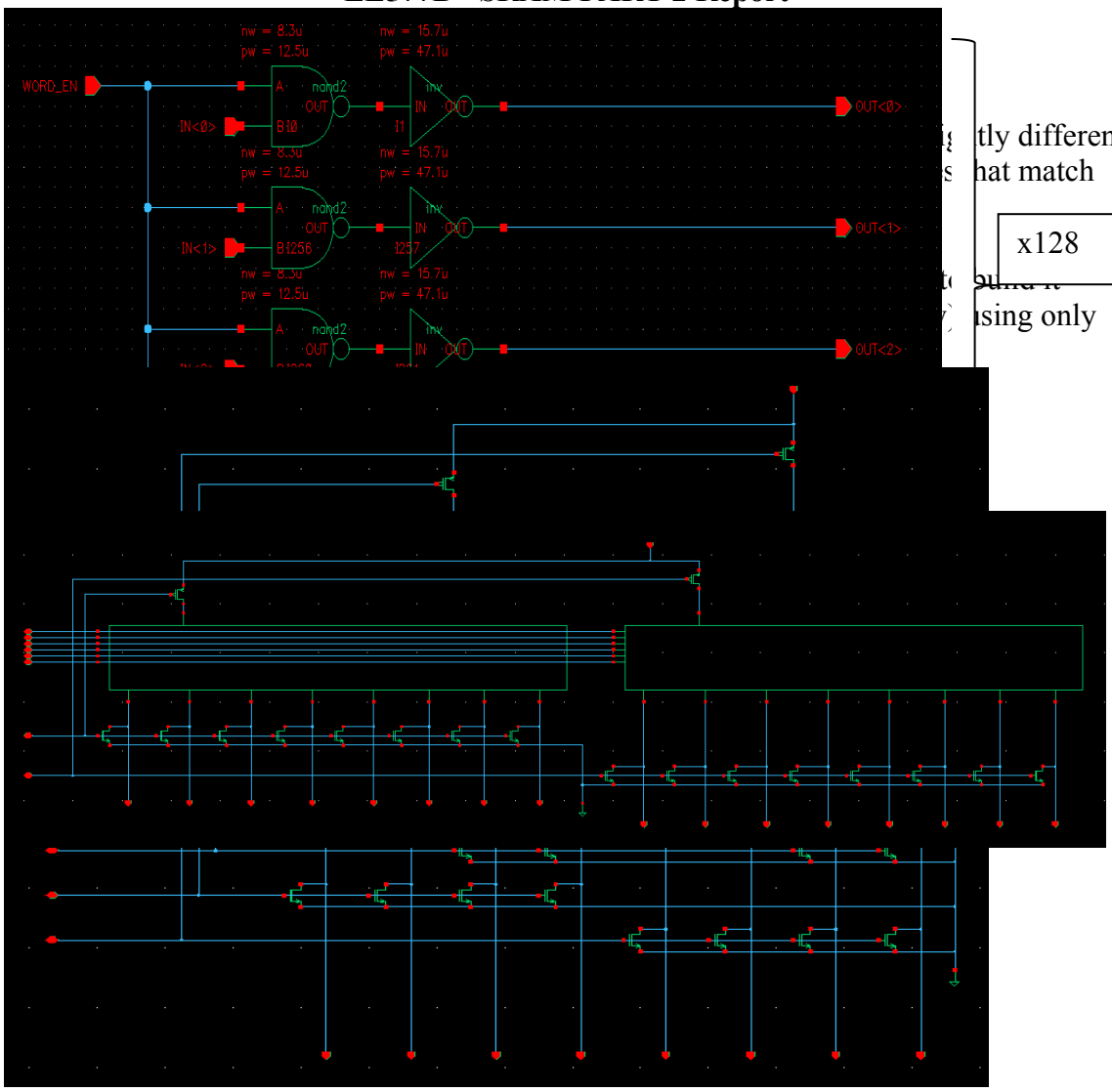
32x64 SRAM MACRO

d two subc ntaining 32 pi models to model the paracitic and resista bit and bit_b lines in the schematic (see the two top of the schematic to the left and right of the column of SRAM cells).

Pi models

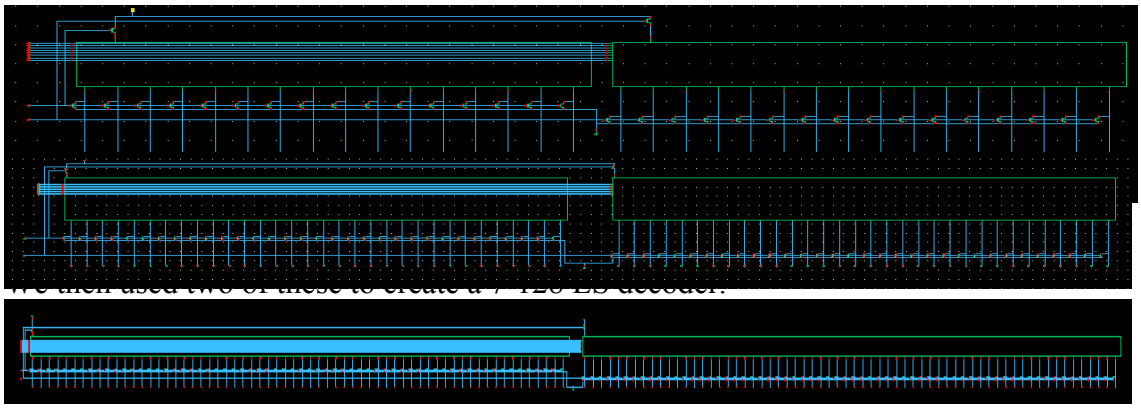
bit SRAM MACRO constructed, it was time to create and test a 128x32 bit wn below).

ifferent
natch



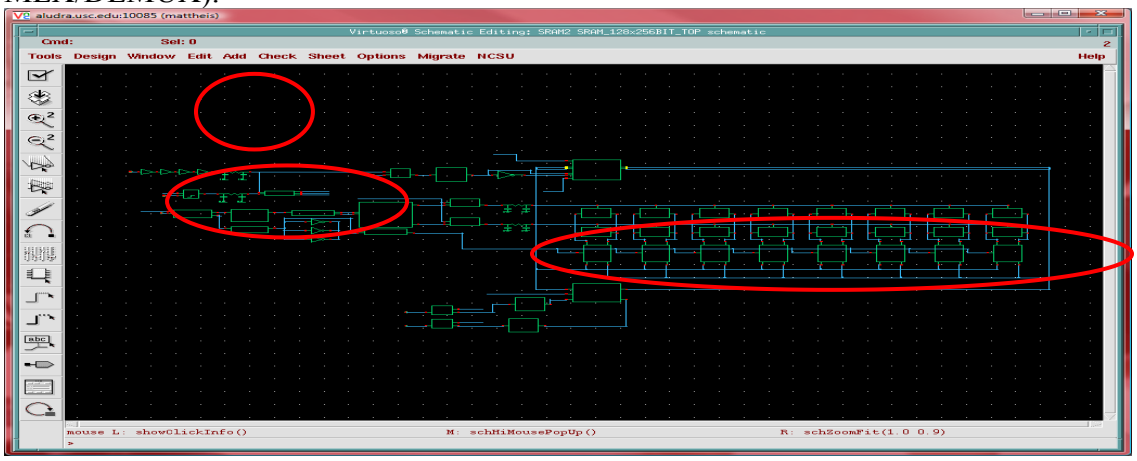
tly different
hat match
using only

We then used two of these to create a 5-32 LS decoder.



The hierarchical approach made the creation and debugging of the 7-128 LS decoder much easier.

Once the 128x32 SRAM was working, it was finally time to create the 128x256 SRAM. This required us to create a control signals, column decoder, and MUX/DEMUX (circled in figure below: from top left to bottom right – control circuit, column decoder, MEX/DEMUX).

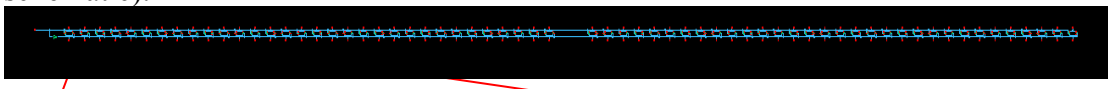


128x256 SRAM (TOP LEVEL CIRCUIT)

Since it was not on the critical path, we copied the 3-8 LS decoder used in the row decoder and left the transistor sizes the same. We used the required the same inverter chain that drives the inverters to provide the address bits and their complements to the 3-8 LS decoder.

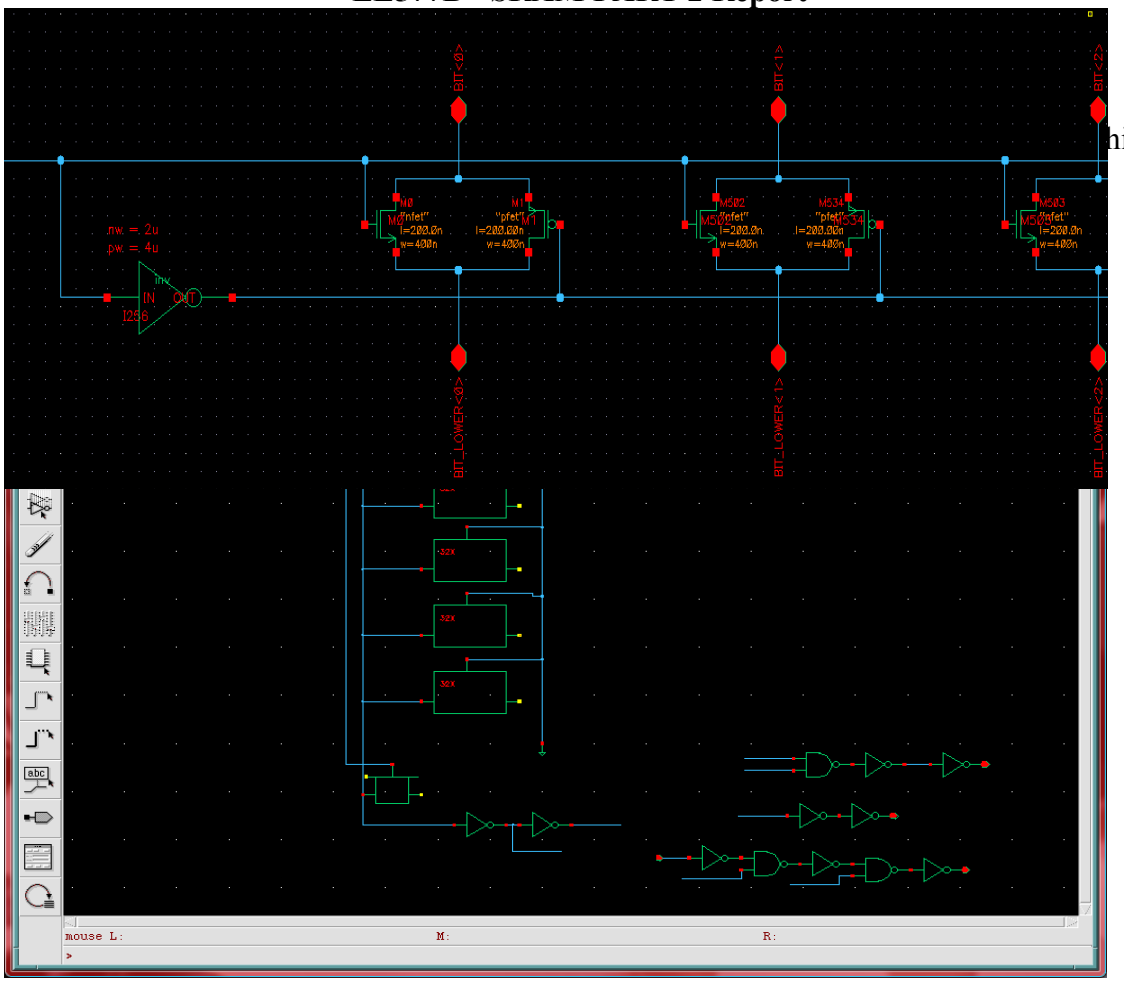
We arranged 8 columns of 128x32 SRAM cells where before there had only been one, and we added the MUX/DEMUX logic below each column.

The MUX/DEMUX cell itself was just 32 transmission gates all controlled by a single input (which we wired to a single output of the column decoder in the top level schematic).



Andrew Mattheisen amattheisen@yahoo.com 2134514711
Zhiyang Ong zhiyang@ieee.org 6004919412

EE577B - SRAM PART 2 Report

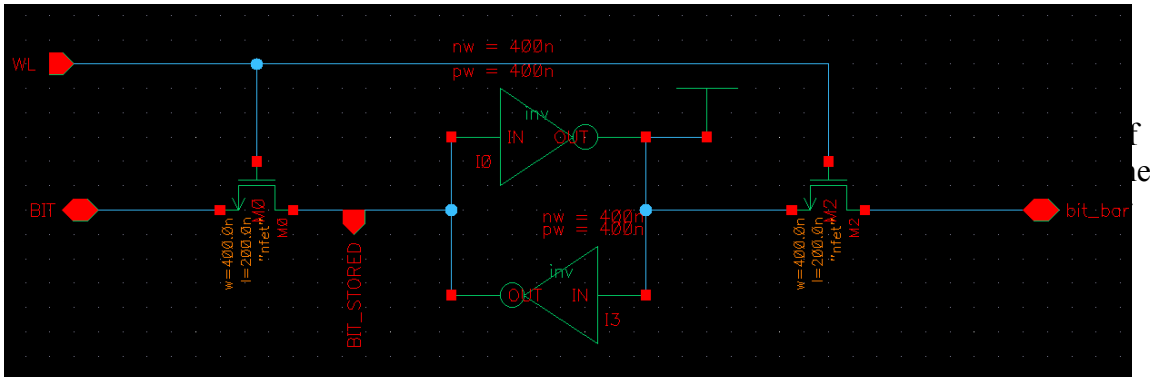


his

Control Signal Generator

The input to the inverter at the top right corner is the clock signal. The inverted clock signal drives the precharge PMOS transistor at the top center of the schematic and an entire dummy row of word lines. The PMOS in turn pulls the dummy bit line high. The small PMOS was used and sized to tune the delay to the desired value for the rise of the bit line in this schematic. The goal was about 1/3 of the clock period.

The inverted clock signal also drives the word line input of the dummy SRAM Cell in the bottom left of the schematic. Once this WL is high, it activeates the dummy SRAM cell, which has been hardwired to always contain a 0 at bit and a 1 at bit bar (see figure below).

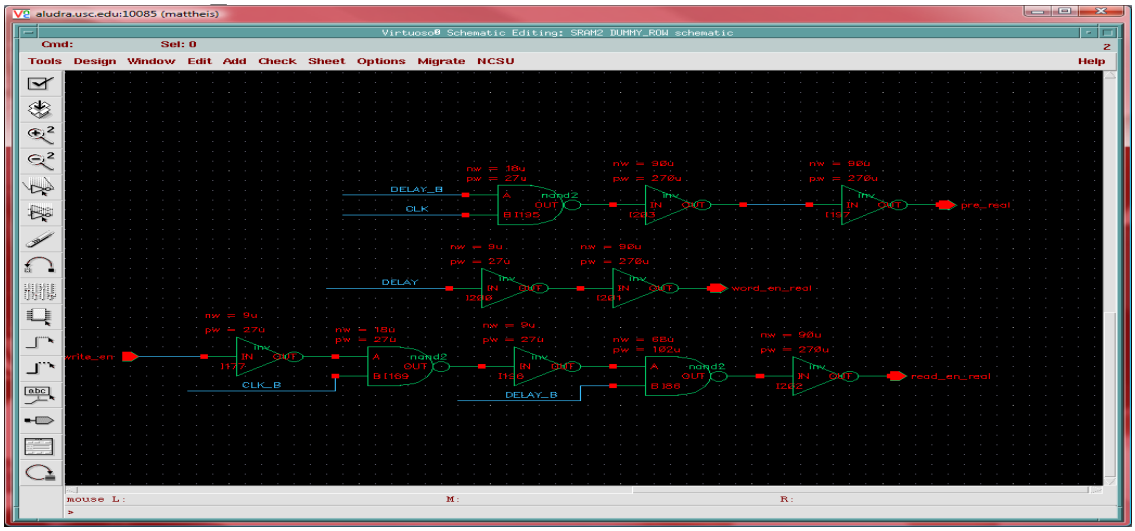


f

ne

Ultimately the waveform of the buffered dummy bit line rises after about 1/3 of the clock period and falls at about 2/3 of the clock signal. Thus it can be used to divide the clock period into three sections.

Here is a zoomed in view of the lower right hand corner of the control signal generation schematic. In this view, you can clearly see the logic used to generate the control signals. The buffered dummy bit line is called "DELAY" and the inverted dummy bit line is called "DELAY_B".



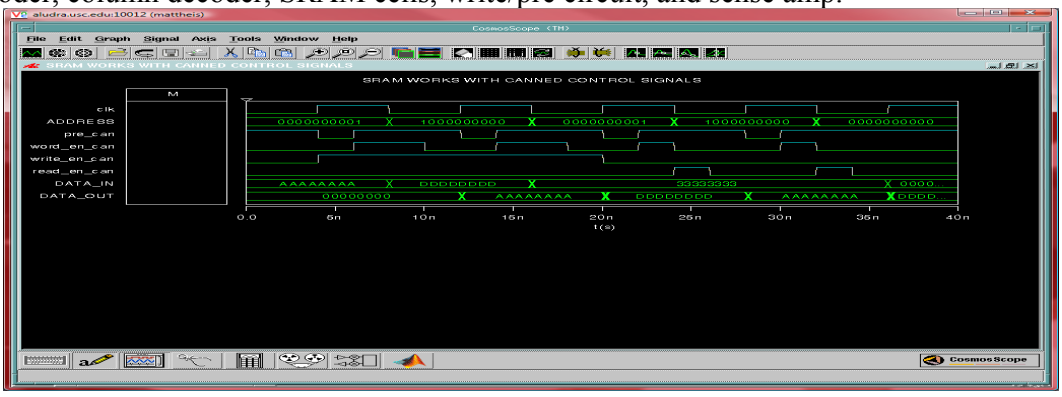
Logic for generation of control signals using dummy circuit delay

Note that the write_en signal does not require complex logic – It will always be the value of write_enable exerted by the write_en Flip Flop.

6 Simulation results

6.1 Functionality Using Canned Inputs

Just to ensure that the SRAM was fully functional, the following test bench exercises the row decoder, column decoder, SRAM cells, write/pre circuit, and sense amp.

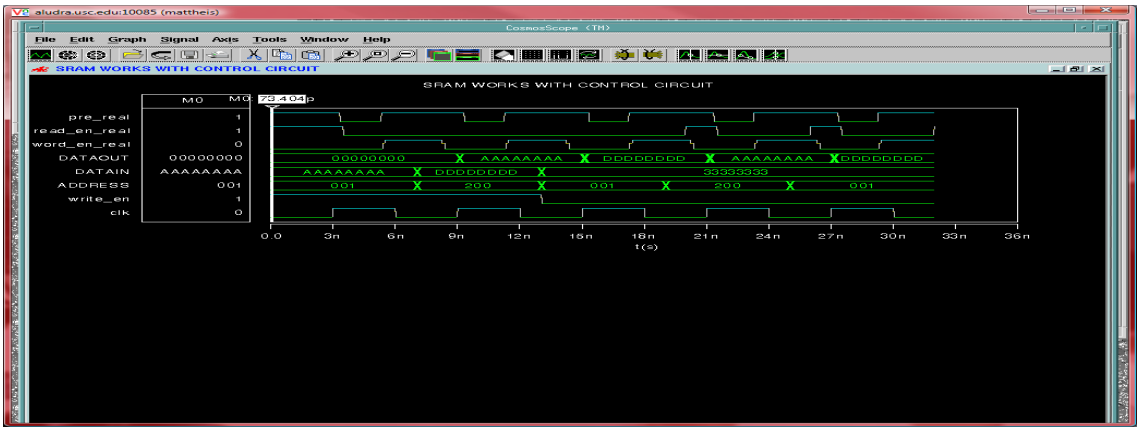


Simulation results of properly working SRAM with canned input signals

As can be seen above, the SRAM cell functions properly. The clock period above is 6ns.

6.2 Functionality Using Self Generated Control Signals

Next those signals were recreated using only CLK, Address, Data_in, and Write_en. Address, Data_in, and Write_en are each available at the output of DFFs and thus only change at the positive edge of CLK. To do this, a control circuit was created. To ensure that the circuit works for temperature and process variations, a dummy row and dummy column of SRAM cells were used to create any delay required for the control signals. Below is a waveform showing the successful results of these efforts.



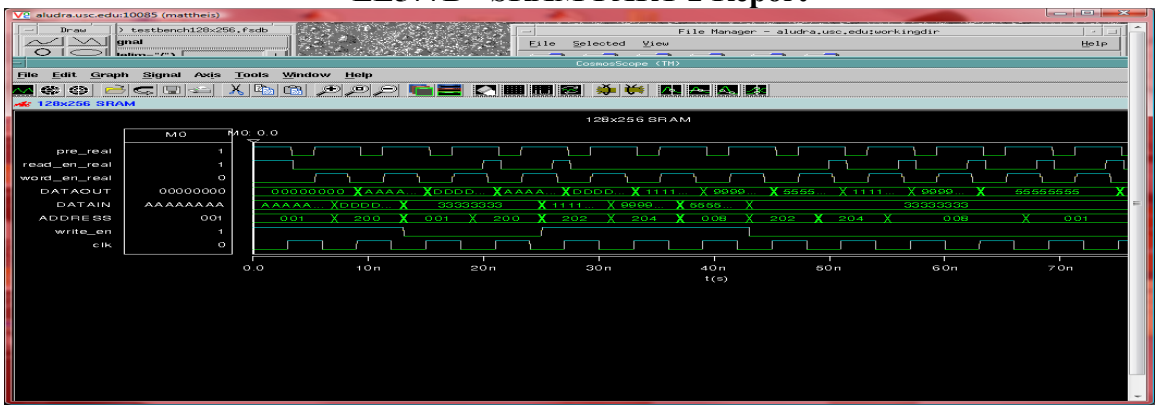
Simulation results of properly working SRAM with proper control logic

As can be seen above, the SRAM cell functions properly. The clock period above is 6ns.

6.3 Extended Functionality Testing

To ensure that the SRAM works in all cases, the test set was expanded to include 5 writes and 6 reads.

PosEdge	Operation	Data Out
1	Write A…A @001	XXX
2	Write D…D @200	XXX
3	Read @001	XXX
4	Read @200	Data from 001 is A…A
5	Write 1…1 @202	Data from 202 is D…D
6	Write 9…9 @204	XXX
7	Write 5…5 @008	XXX
8	Read @202	XXX
9	Read @204	Data from 202 is 1…1
10	Read @008	Data from 204 is 9…9
11	Read @008	Data from 008 is 5…5
12	XXX	Data from 008 is 5…5

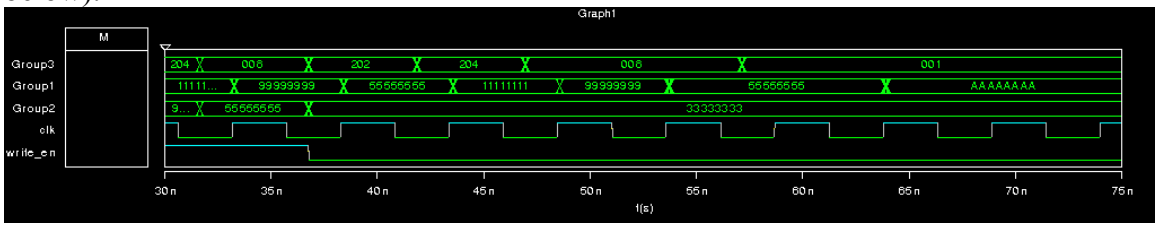


Simulation results – final SRAM test

As can be seen above, the SRAM cell functions properly for consecutive writes to different locations (switching both column and row), consecutive reads to different locations (switching both column and row), reads following writes, writes following reads, and multiple reads of the same location. The clock period used was 6ns, although as will be seen in the next section, the SRAM does function faster.

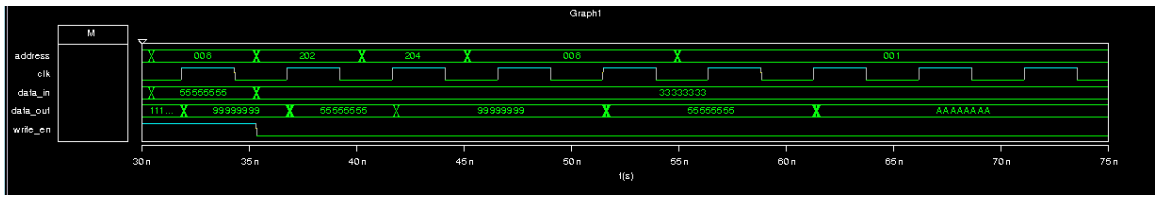
6.4 Max speed of the SRAM

We reduced the clock period until the SRAM stopped functioning correctly. The minimum clock period that the SRAM still functioned correctly was 5.1 ns (see the figure below).



SRAM works for T_{CLK}=5.1ns

The circuit shown working above with a clock period of 5.1 ns operates at 196.1 MHz. **Therefore the maximum speed of the SRAM at 25 °C and supply voltage V_{DD} of 1.8 V is 196.1 MHz.**



SRAM does not work for T_{CLK}=5.0ns

This delay is larger than our theoretical fastest SRAM because of the rough tuning of the control signal generating circuit.

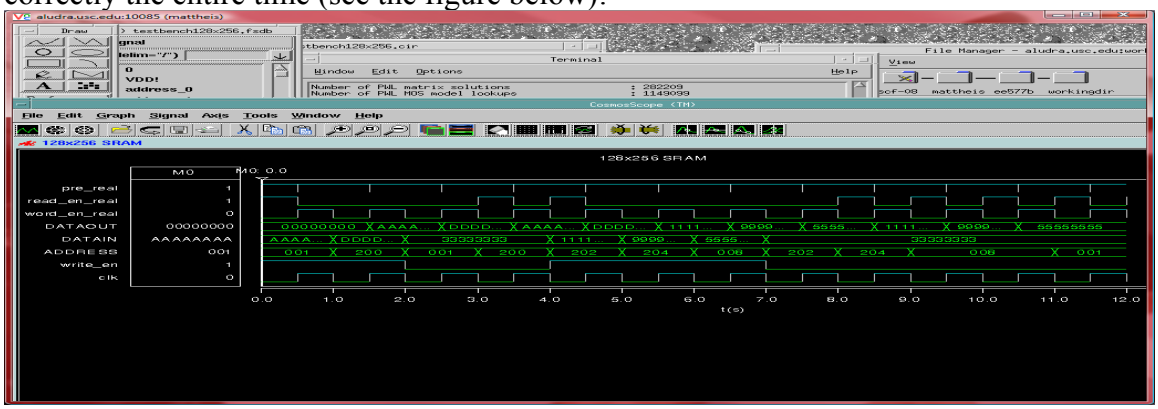
As you can see – a better control signal generator would spend less time with word line enabled and more time with the read enabled during the read clock cycles. If we had a lot

longer to work on this lab, the following could be tweaked to improve the performance of the SRAM:

- PMOS pull up transistor width increased to increase precharge time
- The ratio "Dummy bit line capacitance : number of dummy SRAM cells" reduced to decrease WORD_EN duty cycle and increase time that READ_EN is high.

6.5 Min speed of the SRAM

We tried to increase the clock period until the SRAM stopped functioning correctly… However, we got all the way to 1 second for the clock period and the SRAM functioned correctly the entire time (see the figure below)!



SRAM works for T_{CLK}=1 000 000 000ns (1 second)

Thus the SRAM works for all clock periods at or greater than 5.1ns. This is the result of using the dummy cells and good control circuit design!!!

6.6 PVT Variation Testing

Since the MOSIS transistor model library does not provide the options to simulate various process corners, we varied the input voltage in the model to achieve the effect of the process variations. The results of this testing are included below in Table 4.

We also varied the operating temperature to determine Temperatures effect on out SRAM's performance. The results of this testing are included below in Table 4.

7 Claimed specification

Below is a short table providing the nominal, fastest and slowest operating speeds of the SRAM design corresponding to various operating voltage and temperature conditions. The nominal speeds of the SRAM designs are selected to be the fastest speed of the SRAM design for different operating conditions.

Table 1 Table of Values for the Operating Conditions and Speeds of the SRAM Design.

Operating Conditions	Nominal Speed	Fastest Speed	Slowest Speed
T=25C, VDD=1.8V	196 MHz (T_{CLK}=5.1 ns)	196 MHz (T_{CLK}=5.1 ns)	N/A*
T=25C, VDD=1.5V	166 MHz (T_{CLK}= 6 ns)	166 MHz (T_{CLK}= 6 ns)	N/A*
T=25C, VDD=2.5V	263 MHz (T_{CLK}= 3.8 ns)	263 MHz (T_{CLK}= 3.8 ns)	N/A*
T=50C, VDD=1.8V	188 MHz (T_{CLK}= 5.3 ns)	188 MHz (T_{CLK}= 5.3 ns)	N/A*
T=75C, VDD=1.8V	172 MHz (T_{CLK}= 5.8 ns)	172 MHz (T_{CLK}= 5.8 ns)	N/A*

*The SRAM works at any speed slower than the fastest speed.

From simulating the SRAM design in Nanosim at V_{DD}=1.8V and at a temperature of 25°C for its maximum speed, the average power of the SRAM is 32.863 mW, and its RMS (root-mean-square) power is 50.4483 mW. It has a peak power radiation of 64.9256 mW. Also, the average and RMS capacitive currents for the SRAM design are -13.070890 mA and 21.996679 mA. Likewise, the average and RMS wasted currents for the SRAM design are -5.190366 m and 10.856116 mA. Thus, Nanosim yields the wasted current percentage as 28.422830%. The amount of wasted power is attributed to short-circuit power consumption, static power consumption, and leakage power consumption (Synopsys 2001).

Total power dissipation is measured by the aggregate of dynamic (or switching) power consumption, short-circuit power consumption, leakage power consumption, and static power consumption (Kang et al. 2003). This can be calculated as follows:

$$P_{total} = \alpha_T * C_{load} * (V_{DD})^2 * f_{clk} + V_{DD} *(I_{short\text{-}circuit} + I_{leakage} + I_{static})$$

However, since the utilization of this equation would require the determination of power consumption at every node, a circuit simulation tool is needed to perform the complex calculations. With Nanosim, this requires listing all possible nodes. Thus, for faster computation of power consumption, the authors chose to measure the power consumed for the entire SRAM design using Nanosim features, rather than performing manual estimations/calculations, or semi-automate that with nodal measurements of power consumption.

The total area for the SRAM design is the square of its longest dimension, which is 8704 λ by 8704 λ. This is given as 75.759 * 10^6 λ^2. The total amount of space used is given by the sum of the area for all modules in the SRAM, which is 59.947 *10^6 λ^2. Therefore, the leftover white space is 15.812 *10^6 λ^2.

8 Matlab Scripts

8.1 "Calcs_A.m"

The "Calcs_A.m" matlab script has been included below:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Matlab file created by Andrew Mattheisen and Zhiyang Ong                    %
% You can contact us at amattheisen@yahoo.com or zhiyang@ieee.org             %
% Andrew's student ID is 2134514711.  Zhiyang's student ID is                 %
% 6004919412                                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This matlab file determines the delay of the row decoder.                   %
% And several other good things (see disp lines in text for                   %
% explanation                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
disp('-----------------------------------------------------------------
--')
disp ('This script calculates gate sizes for WL driver')
disp('-----------------------------------------------------------------
--')
format long g
format compact
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GENERIC PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
m=10^-3;
u=10^-6;
n=10^-9;
p=10^-12;
f=10^-15;
a=10^-18;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TECHNOLOGY SPECIFIC PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lambda=.1*u;
CGate=2.396*f/u; % This is for an inverter gate, assumed to work for
                 %   the NMOSgate, from TA's solution to part 1
CNDiff=2.244*f/u;   % nmos drain or source capacitance from TA's
solution
                    % to part 1
CPDiff=1.679*f/u;   % pmos drain or source capacitance from TA's
solution
                    % to part 1
REffPmos = 7.8648*10^3*u; % from TA's solution to part 1, problem 1e
REffNmos = 2.5832*10^3*u; % from TA's solution to part 1, problem 1e
rsheetm2 = .08; % this is in ohms per square from tech file
pinv=.5499e-10; % determined by dettao.m
tao=.0778e-10; % determined by dettao.m
disp('-----------------------------------------------------------------
--')
disp('Calculate the width of the L-S gate of minimum size for 7 input
bits.')
disp('-----------------------------------------------------------------
--')
```

```
disp('')
NMOSW=4;
MINPMOSW=24;
TotalWidth=0.0;
for i=1:7
    TotalWidth=TotalWidth+NMOSW+MINPMOSW*2^(i-1);
end
TotalWidth
%side load capacitance of the minimum sized L-S decoder
Carea=TotalWidth*lambda^2*14*a/(u*u);  % wire area cap
Cfringe=TotalWidth*lambda*35*a/u; %   wire fringe cap
Cwire=Carea+Cfringe;
%  calculate input gate width of the L-S of minimum size for 7 input
bits
n=7; % number of input bits
GateWidth=NMOSW*2^(n-1)+MINPMOSW*2^(n-1);
GateWidth
%input capacitance of the minimum sized L-S decoder
Cout1=GateWidth*lambda*CGate;
disp('-------------------------------------------------------------------
-----')
disp('Note: The vertical wire capacitance is negligible compared to the
gate')
disp('capacitance of the decoder')
disp('-------------------------------------------------------------------
-----')
disp('')
disp('-------------------------------------------------------------------
--')
disp('Calculations for optimal gate sizing on entire path')
disp('-------------------------------------------------------------------
--')
disp('')
Cin3=6*CGate*10^-6;
NumCols=256;
Cout3=(4*lambda)*(40*lambda)*(NumCols+48)*14*a/(u*u) ...   % wire area
cap
+ 2*(40*lambda)*(NumCols+48)*35*a/u ... %                  + wire
fringe cap
+ 2*CGate*NumCols*4*lambda; %                           + SRAM pass gate
cap
%note the extra 48 in the area and fringe calcs is due to routing
distances


H3=Cout3/Cin3;
G3=(2^6)*(1+3*((1-(1/2^7))/(1-1/2)))/(1+3)*(5/4);
B3=2;
F3=H3*G3*B3;
Nhat3=round(log (F3)/log (4))
fhat3=F3^(1/Nhat3)
% gate capacitance ans size calculations
disp(' ')
disp('FF')
disp('----------------------')
C3gate1=Cin3/(1)
```

```
WN3gate1=round(C3gate1/CGate/3*10^6*10)/10
WP3gate1=round(C3gate1/CGate/3*2*10^6*10)/10
disp(' ')
disp('Branched inverter (bi=2)')
disp('-----------------------')
C3gate2=C3gate1*fhat3/(1*2)
WN3gate2=round(C3gate2/CGate/4*10^6*10)/10
WP3gate2=round(C3gate2/CGate/4*3*10^6*10)/10
disp(' ')
disp('Inverter')
disp('-----------------------')
C3gate3=C3gate2*(fhat3)/(1)
WN3gate3=round(C3gate3/CGate/4*10^6*10)/10
WP3gate3=round(C3gate3/CGate/4*3*10^6*10)/10
disp(' ')
disp('Inverter')
disp('-----------------------')
C3gate4=C3gate3*fhat3/(1)
WN3gate4=round(C3gate4/CGate/4*10^6*10)/10
WP3gate4=round(C3gate4/CGate/4*3*10^6*10)/10
disp('7 bit LS Decoder')
disp('-----------------------')
C3gate5=C3gate4*fhat3/(1)
WN3gate5=round(C3gate5/CGate/7*10^6*10/64)/10
WP3gate5=round(C3gate5/CGate/7*6*10^6*10/(2^7))/10
disp(' ')
disp('AND gate')
disp('-----------------------')
C3gate6=C3gate5*fhat3/((2^6)*(1+3*((1-(1/2^7))/(1-1/2)))/(1+3))
WN3gate6=round(C3gate6/CGate/5*2*10^6*10)/10
WP3gate6=round(C3gate6/CGate/5*3*10^6*10)/10
disp(' ')
disp('Inverter')
disp('-----------------------')
C3gate7=C3gate6*(fhat3)/(5/4)
WN3gate7=round(C3gate7/CGate/4*10^6*10)/10
WP3gate7=round(C3gate7/CGate/4*3*10^6*10)/10
disp(' ')
%final load - check that it matches Cout3
disp(' ')
disp('-----------------------')
C3load=C3gate7*fhat3
disp('Should be equal to')
Cout3
disp(' ')
disp('-----------------------------------------------------------------
--')
disp('Calculations for Delay for path 3')
disp('-----------------------------------------------------------------
--')
disp('')
%delay is the sum of the gh's +P
Delay3=fhat3*7+1 %+1 is for extra inverter in non-optimal path
DelayinSec=Delay3*tao+pinv*7+pinv %.65psec!!!
Delay3Pi=WN3gate7*REffNmos*(Cout3/2) ...
+(WN3gate7*REffNmos+rsheetm2*(40*lambda)*NumCols/(4*lambda))*(Cout3/2)
```

```
Delay3totalPi=DelayinSec+Delay3Pi
disp(' ')
disp('-----------------------------------------------------------------
--')
disp('Calculations for Pi models')
disp('-----------------------------------------------------------------
--')
disp('')

CpiWL=Cout3/2
Rpiwl=rsheetm2*(40*lambda)*NumCols/(4*lambda)

CpiBL=(4*lambda)*(40*lambda)*(32*7)*14*a/(u*u)/2 ...    % wire area cap
+ (40*lambda)*(32*7)*35*a/u    %                  + wire fringe cap
RpiBL=rsheetm2*(40*lambda)*(32*7)/(4*lambda)

CpiDATA_OUT=(4*lambda)*(40*lambda)*(32*6)*14*a/(u*u)/2 ...    % wire
area cap
+ (40*lambda)*(32*7)*35*a/u    %                  + wire fringe cap
RpiDATA_OUT=rsheetm2*(40*lambda)*(32*6)/(4*lambda)

CpiDATA_IN=(4*lambda)*(40*lambda)*(32*4.5)*14*a/(u*u)/2 ...    % wire
area cap
+ (40*lambda)*(32*7)*35*a/u    %                  + wire fringe cap
RpiDATA_IN=rsheetm2*(40*lambda)*(32*4.5)/(4*lambda)

CpiWRITE_EN=(4*lambda)*(40*lambda)*(32*5)*14*a/(u*u)/2 ...    % wire
area cap
+ (40*lambda)*(32*7)*35*a/u    %                  + wire fringe cap
RpiWRITE_EN=rsheetm2*(40*lambda)*(32*5)/(4*lambda)

CpiCLK=(4*lambda)*(40*lambda)*(32*9)*14*a/(u*u)/2 ...    % wire area cap
+ (40*lambda)*(32*7)*35*a/u    %                  + wire fringe cap
RpiCLK=rsheetm2*(40*lambda)*(32*9)/(4*lambda)

CpiROW_ADDRESS=(4*lambda)*(40*lambda)*(32*7.5)*14*a/(u*u)/2 ...    %
wire area cap
+ (40*lambda)*(32*7)*35*a/u    %                  + wire fringe cap
RpiROW_ADDRESS=rsheetm2*(40*lambda)*(32*7.5)/(4*lambda)

CpiCOL_ADDRESS=(4*lambda)*(40*lambda)*(32*5)*14*a/(u*u)/2 ...    % wire
area cap
+ (40*lambda)*(32*7)*35*a/u    %                  + wire fringe cap
RpiCOL_ADDRESS=rsheetm2*(40*lambda)*(32*5)/(4*lambda)



disp(' ')
disp('-----------------------------------------------------------------
--')
disp('Calculations for precharge control signal circuit')
disp('-----------------------------------------------------------------
--')
disp(' ')
Pre_load=3430.4*u*CGate
```

```
Pre_in=(.4+1.2)*u*CGate
HPre=Pre_load/Pre_in
FPre=HPre;
NhatPre=round(log(FPre)/log(4))
%D=gh+P
DPre=HPre*tao

disp(' ')
disp('-----------------------------------------------------------------
--')
disp('Calculations for precharge delay')
disp('-----------------------------------------------------------------
--')
disp(' ')
r1pre=50*lambda*REffPmos
r2pre=RpiBL
C1pre=CpiBL+(CNDiff*128*4*lambda)/2
C2pre=CpiBL+(CNDiff*128*4*lambda)/2
Delay_Pre=r1pre*C1pre+(r1pre+r2pre)*C2pre

disp(' ')
disp('-----------------------------------------------------------------
--')
disp('Calculations for WL rise to dropping Bit line by .15V')
disp('-----------------------------------------------------------------
--')
disp(' ')
r1bit=4*lambda*REffNmos
r2bit=RpiBL
C1bit=C1pre
C2bit=C2pre
Delay_bit=r1bit*C1bit+(r1bit+r2bit)*C2bit
Delay_bit_total=Delay_bit*.15/(.63*1.8)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%RESULTS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%{

%}
```

9

References

Sung--Mo Kang and Yusuf Leblebici, “CMOS Digital Integrated Circuits: Analysis and Design”, Third edition, McGraw--Hill, New York, NY, 2003

Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic, “Digital Integrated Circuits: A Design Perspective”, Second edition, Pearson Education, Upper Saddle River, NJ, 2003

Synopsys, NanoSim User Guide, Mountain View, CA, 2001

Ivan Sutherland, Bob Sproull, and David Harris, “Logical Effort: Designing Fast {CMOS} Circuits”, Morgan Kaufmann, San Francisco, CA, 1999

Neil H. E. Weste and David Harris, “CMOS VLSI Design: A Circuits and Systems Perspective”, Third edition, Pearson Education, Boston, MA, 2005

10 Appendix A

Figures 5 to 10 (below) represent the schematics for some of the decoder designs that were considered. The remaining considered decoder designs used the Lyon-Schediwy decoder. This is used to compared how cost-effective is the Lyon-Schediwy decoder, when compared to some alternatives since the Lyon-Schediwy decoder pushes the side-load parasitic capacitances and resistances of the word line to the outputs. With other decoder designs, iterative approaches and the method of logical effort were used to perform gate sizing to deal with side-load parasitic capacitances and resistances of the word line along the critical path. However, when buffer insertion and gate sizing were carried out, the authors had to use MATLAB scripts to semi-automate the process of estimating the delay-area trade-off of the SRAM architectural designs to empirically prove that a design is better than others.

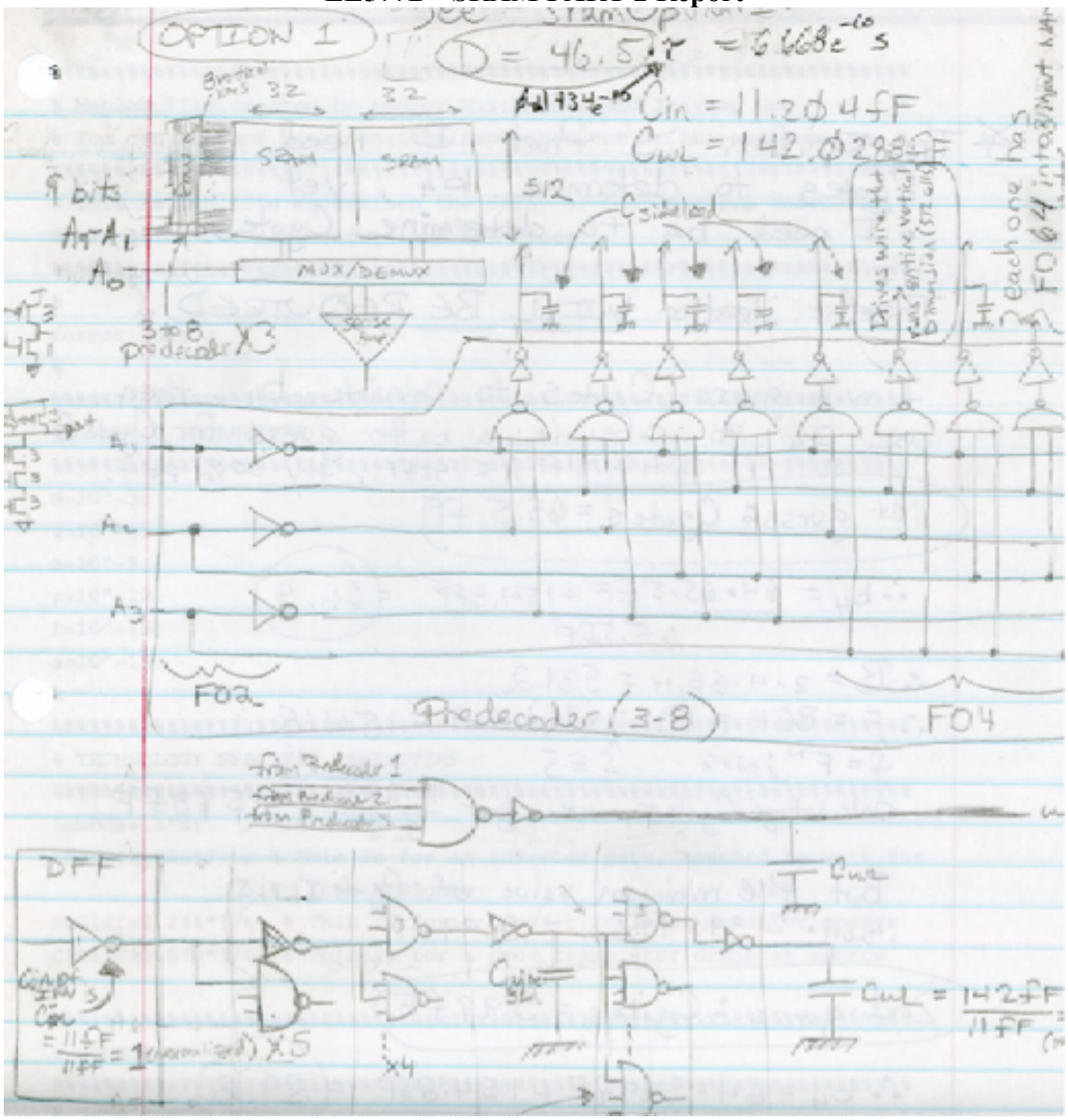


Figure 5 Parital schematic for the circuit of the 9th considered decoder

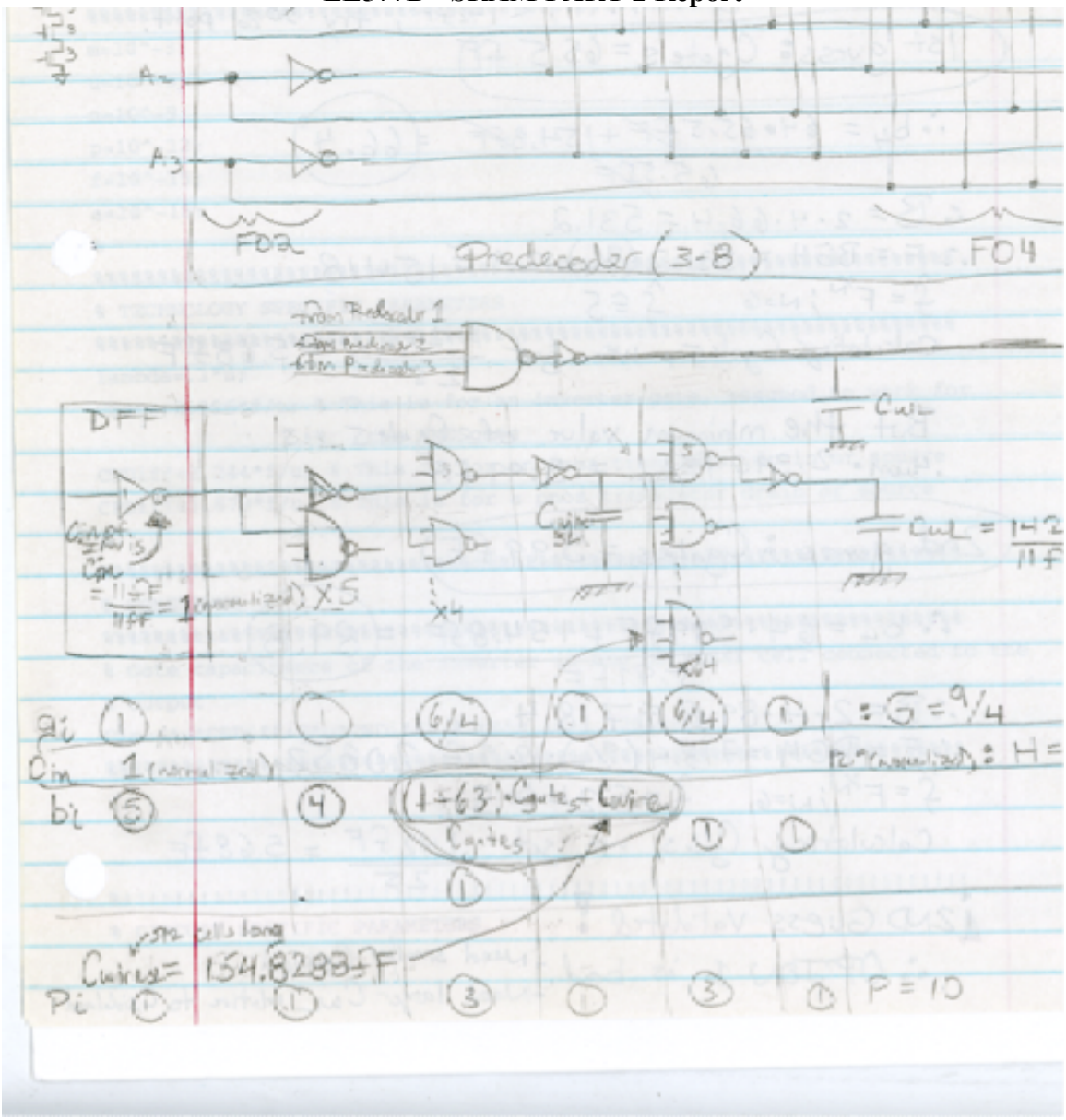


Figure 6 Remaining circuit for the 9th decoder design

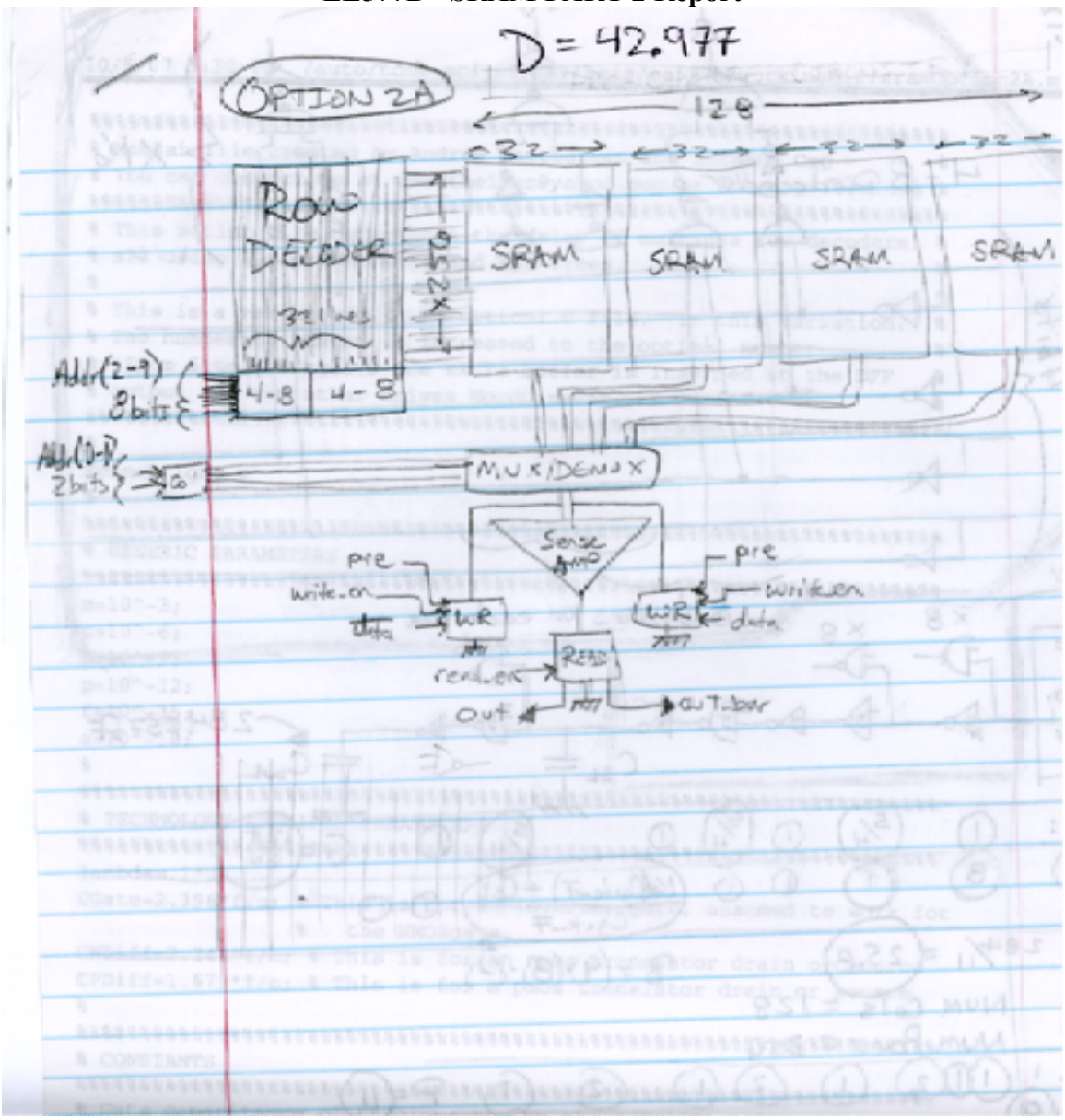


Figure 7 Partial schematic for the 10th considered decoder

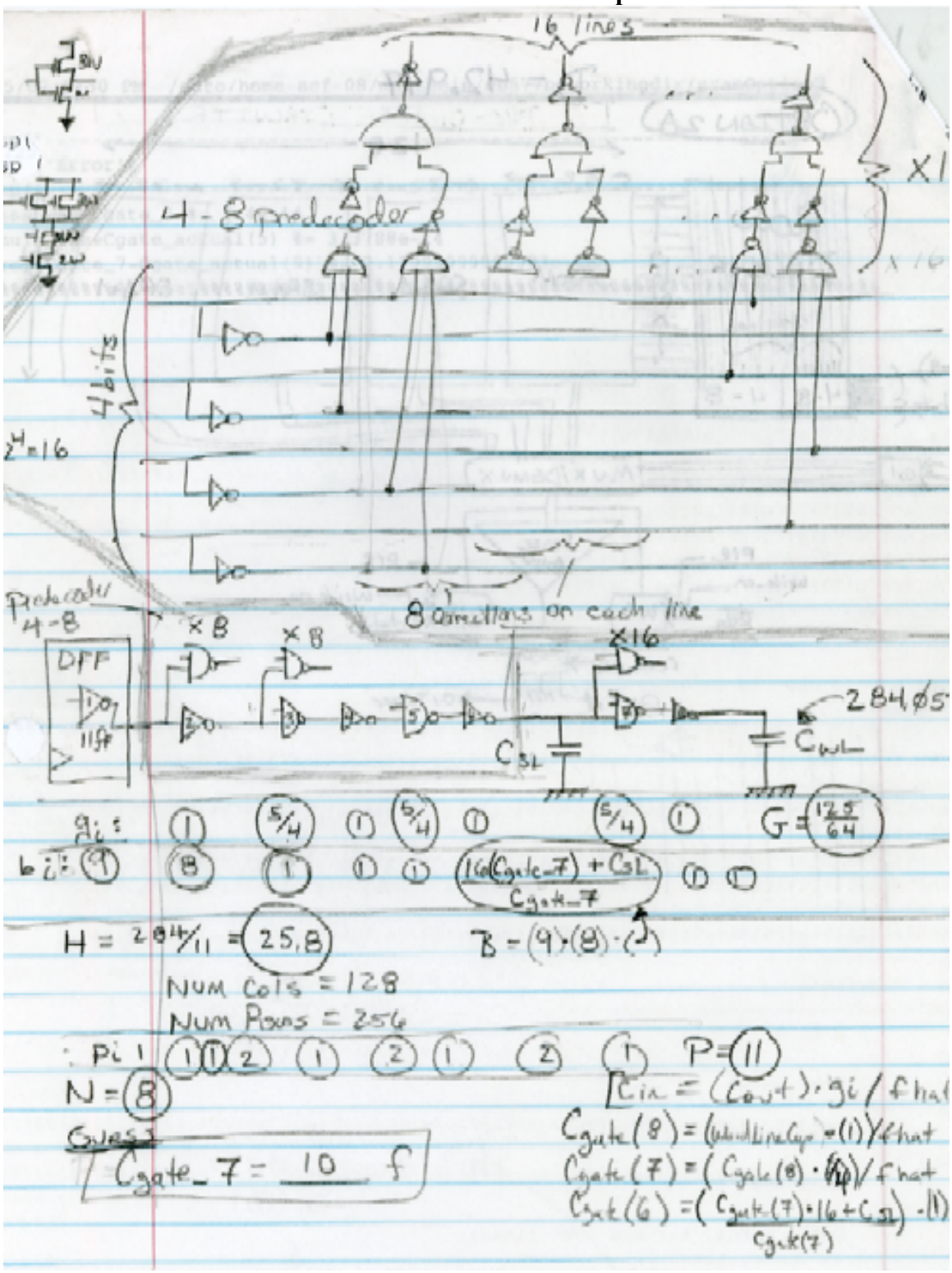


Figure 8 Remaining circuit for the 10th considered decoder

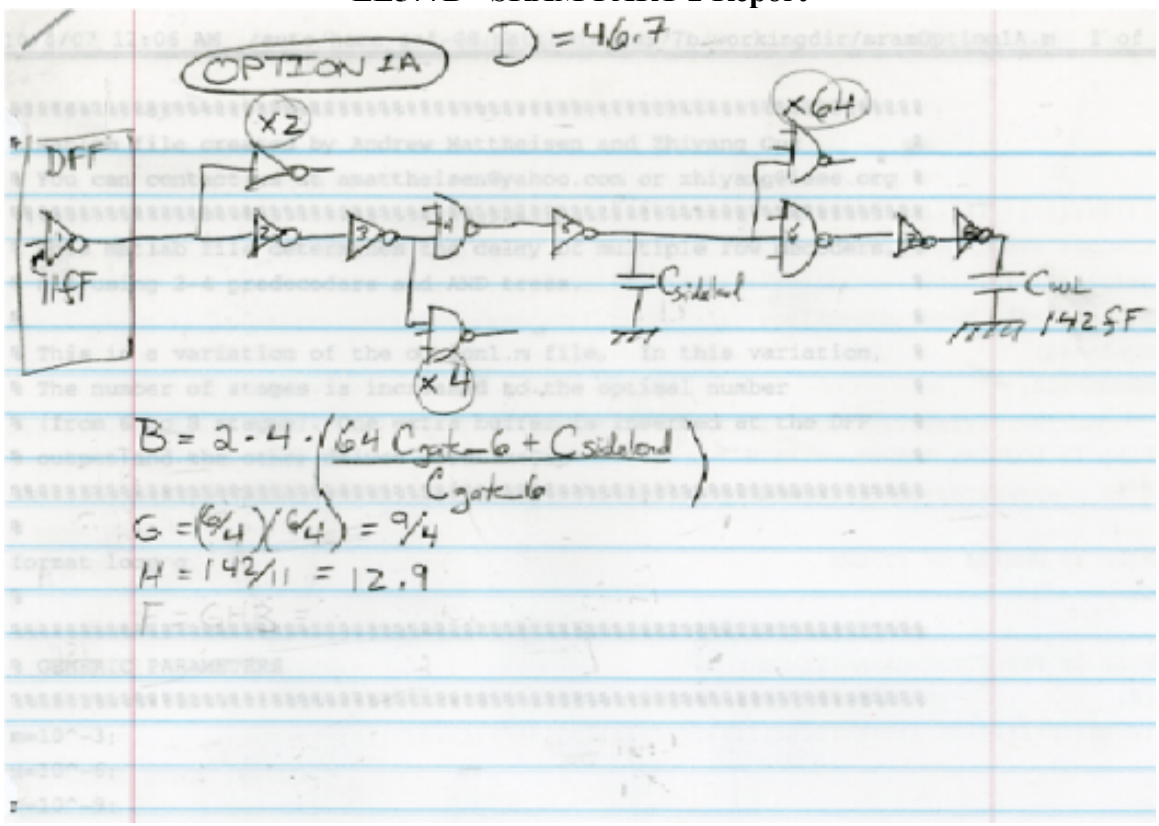


Figure 9 Schematic for the 11th considered decoder

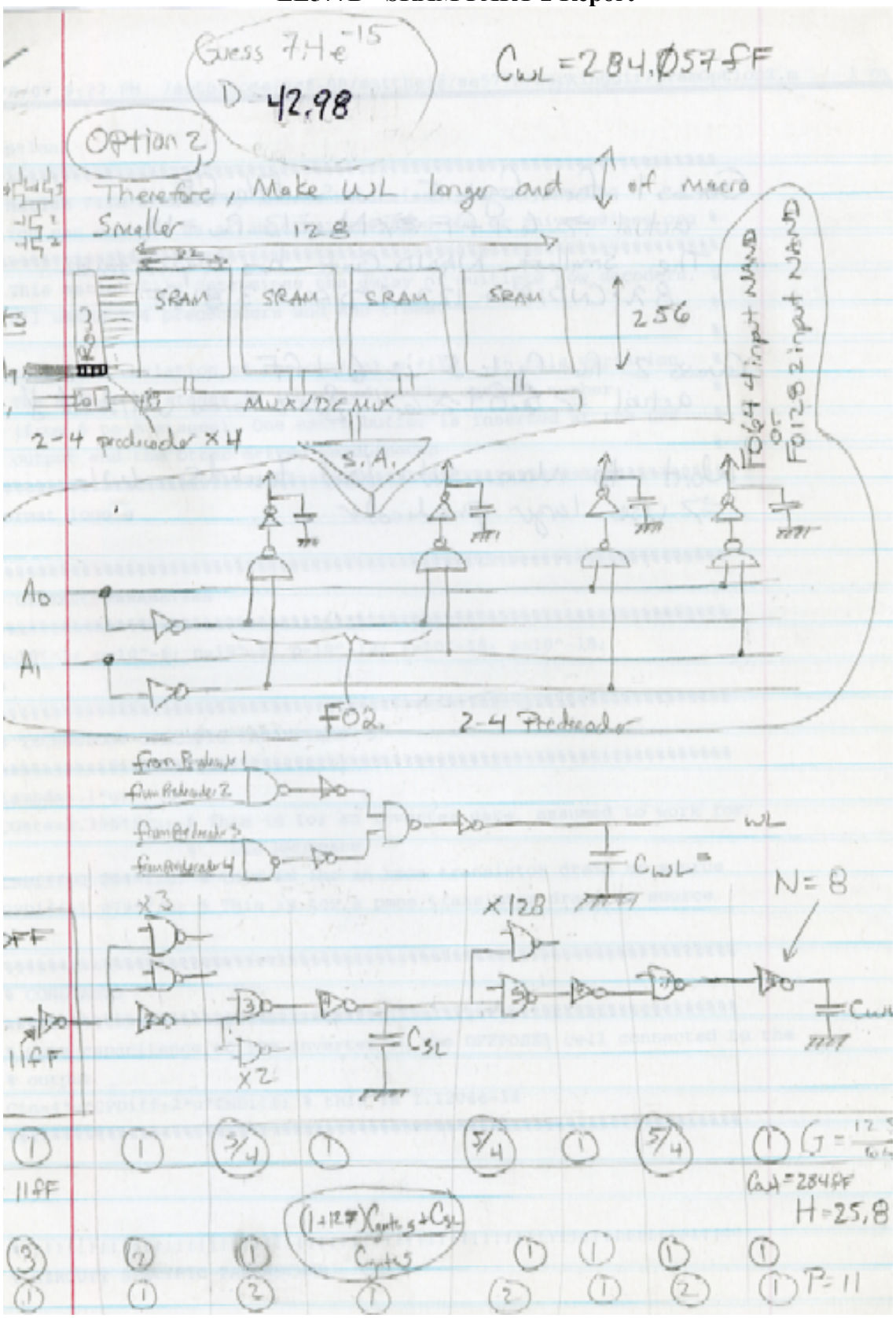


Figure 10 Schematic for the 12th considered decoder