University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS151/251A — B. Nikolic & S. Shao
Fall 2019 — 12/19/19

Final Exam

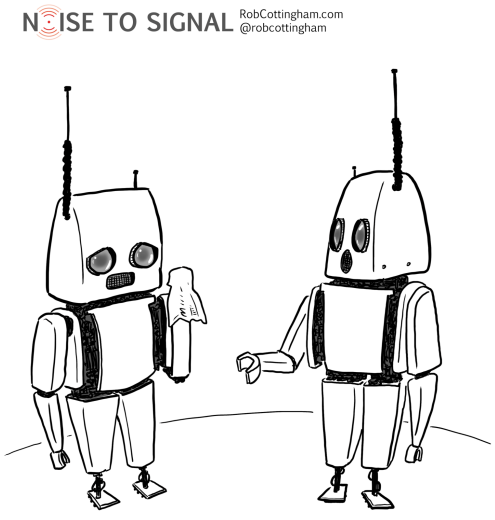
Name: _____

Student ID number: _____

Class (EECS151 or EECS251A): _____

This is a *closed-book* exam, but you are allowed three sheets of stapled notes. Write your Student ID number on all pages.

Problem	1	2	3	4	5	6	Total
Minutes	20	35	30	30	25	40	180
Max Points	20	40	30	30	20	40	180
Points							



I am sorry that my leaving causes you pain.
But the algorithm wants what the algorithm wants.

1. Finite State Machine(Midterm 1 Clobber 1) (20 mins, 20 points)

Cory's Building Manager asked you to help them design a Moore state machine for the elevator in Cory Hall that doesn't break all the time. Specifically, you received the following design specs:

- To simplify the problem, let's assume that there are only two floors, where we have an "Up" button on the first floor and a "Down" button on the second floor.
- There is also a sensor indicating which floor the elevator is currently at.
- If the elevator is moving, pressing the "Up" and "Down" buttons should have no effect.
- In addition, we only read the sensor when the elevator is moving.

We use the following symbols to represent the inputs, outputs, and the states of the elevator controller in our FSM design:

Inputs:

- U1, D2 (Buttons on each floor.)
- S (Sensor to tell whether elevator is at the floor: 0 for the 1st floor and 1 for the 2nd floor.)

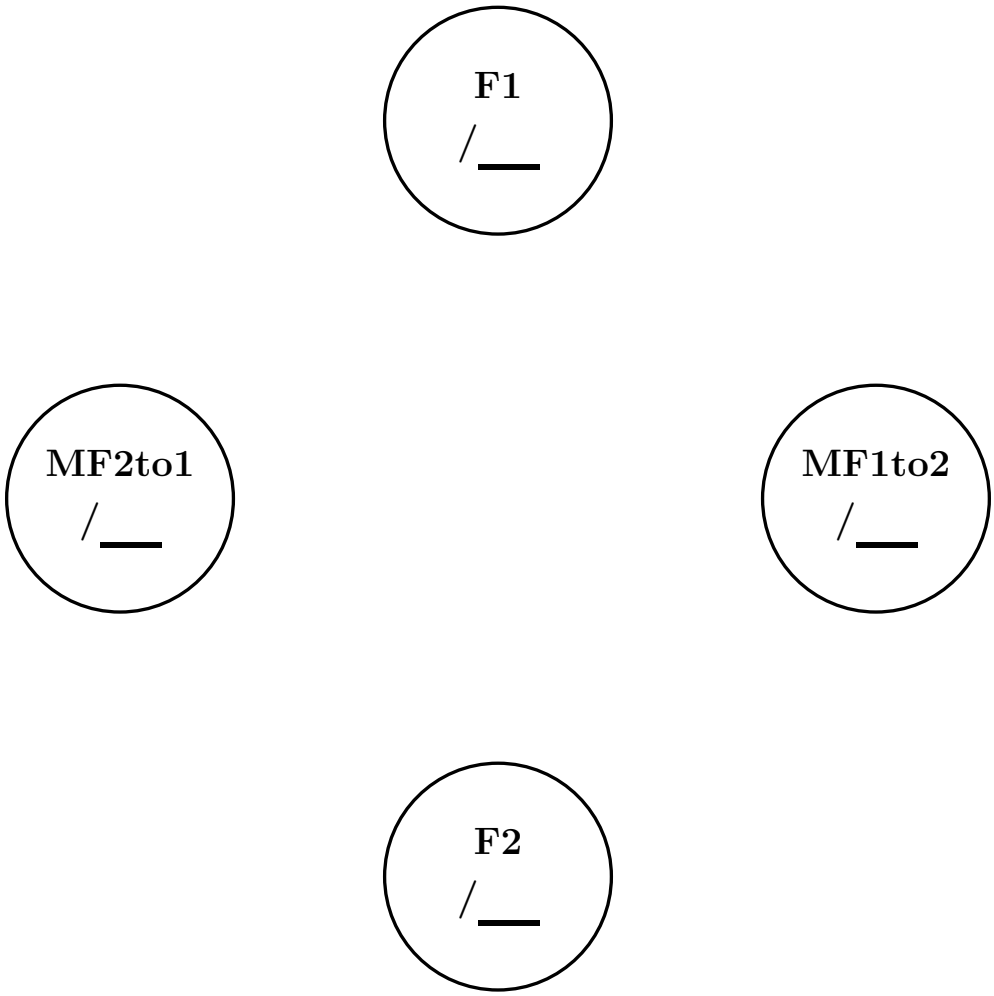
Outputs:

- Door Open (0) or Closed (1)

States:

- $F1$: At Floor 1.
- $MF1to2$: Moving from Floor 1 to Floor 2.
- $F2$: At Floor 2.
- $MF2to1$: Moving from Floor 2 to Floor 1.

(a) Complete the state transition diagram.



(b) Fill out the truth table for the next state and output logic.

Current State (CS)	U1	D2	S	Next State (NS)	Output
$F1$	0	0	X		
$F1$	0	1	X		
$F1$	1	0	X		
$F1$	1	1	X		
$MF1to2$	X	X	0		
$MF1$to2	X	X	1		
$F2$	0	0	X		
$F2$	0	1	X		
$F2$	1	0	X		
$F2$	1	1	X		
$MF2to1$	X	X	0		
$MF2to1$	X	X	1		

(c) The current state is one-hot encoded into 4 bits according to the table below:

Current State (CS)	CS[3:0]
`F1`	`4'b0001`
`MF1to2`	`4'b0010`
`F2`	`4'b0100`
`MF2to1`	`4'b1000`

Write the Boolean equations for the next state bits and output in **sum-of-products form** in terms of `CS[3:0], U1, D2, S`.

`NS[0]` = _____

`NS[1]` = _____

`NS[2]` = _____

`NS[3]` = _____

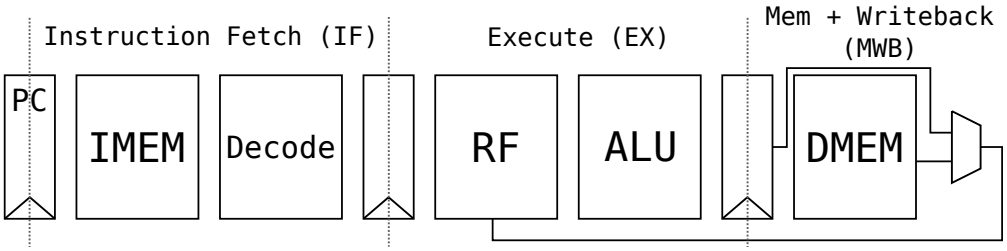
`Output` = _____

(d) Implement the logic for `Output` using NAND2 and NOR2 gates.

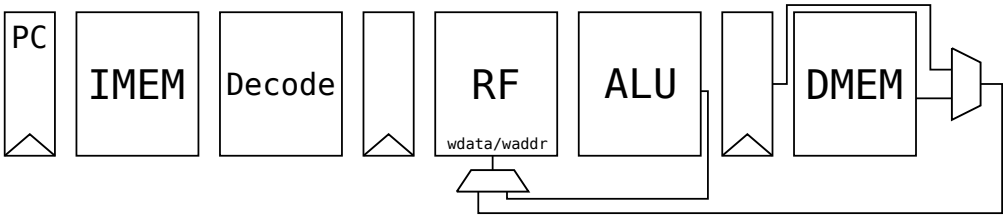
2. RISC-V and Microarchitecture (Midterm 1 Clobber 2) (35 mins, 40 points)

There are several ways to improve the performance of the basic RISC-V datapath presented in lecture. We'll explore 3 optimizations in this problem: 1) regfile writeback bypass from EX stage, 2) multicore processor, and 3) superscalar architecture.

Take a typical 3-stage RISC-V datapath split as shown in the diagram below. It has no forwarding paths, but the regfile can be written and read on the same cycle (as presented in lecture). The IMEM and DMEM are async read memories.



It seems wasteful to force all instructions to go through the memory stage even if they are arithmetic instructions that have finished computation in the 2nd stage. One proposal is to add a mux to the regfile `wdata` and `waddr` ports with the ALU output and `rd` from the EX stage, as shown below.



(a) **Circle** the hazard types that are introduced by this datapath modification.

Data Hazard Control Hazard Structural Hazard

Circle the instruction sequences that wouldn't execute correctly on the modified datapath.

```
addi x2, x3, 100        lw x1, 0(x2)        sw x1, 4(x2)
 xor x2, x2, x3       addi x3, x4, 100      sw x1, 8(x2)
```

(b) Let's say we add another write port to the regfile, so that the mux is no longer necessary. If both the MWB stage and EX stage want to write to the same address in the regfile, which stage's data should take precedence and why?

(c) To prepare for a multicore processor, design an arbiter that arbitrates two input interfaces (`data0_in, data1_in`) to one output interface (`data_out`). The arbiter should conform to this spec:

i. If only `data_in_valid[0]` is high, then `data_out` should be driven from `data0_in`.

ii. If only `data_in_valid[1]` is high, then `data_out` should be driven from `data1_in`.

iii. If `data_in_valid == 2'b11`, the arbiter should prioritize the interface that didn't send data most recently (round-robin). On reset, the arbiter should prioritize interface 0.

Fill the Verilog template below by writing your answers in the lines corresponding to the (1), (2), etc. numbers in the template.

```
module arbiter (
  input clk, rst,
  input [31:0] data0_in, data1_in,
  input [1:0] data_in_valid,
  output reg [31:0] data_out,
  output reg data_out_valid
);
  reg last_served;
  always @(posedge clk) begin
    if (rst)
      last_served <= (1);
    else if (&data_in_valid)
      last_served <= (2);
    else if (|data_in_valid)
      last_served <= (3);
  end

  always @(*) begin
    if (&data_in_valid) begin
      data_out            = (4);
      data_out_valid      = (5);
    end
    else if (|data_in_valid) begin
      data_out            = (6);
      data_out_valid      = (7);
    end
    else if (!(|data_in_valid)) begin
      data_out            = (8);
      data_out_valid      = (9);
    end
  end
endmodule
```

1. _____

2. _____

3. _____

4. _____

5. _____

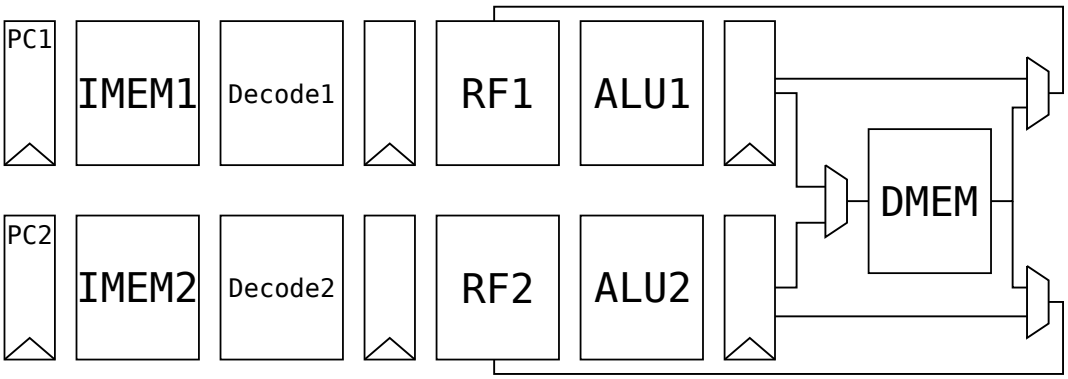
6. _____

7. _____

8. _____

9. _____

(d) Let's design a multi-core processor. To do this, we will take a standard 3-stage RISC-V datapath and duplicate it, where each copy has its own PC and regfile. Assume that each datapath has its own instruction memory, but they share a data memory that only has one R/W port, as shown below.

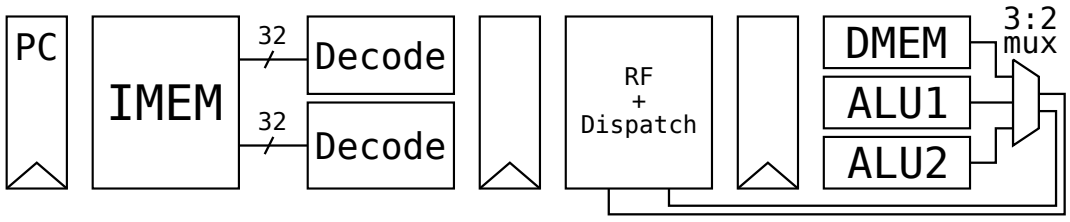


The data memory access is arbitrated in a round-robin fashion (starting with prioritizing core 1) just like the arbiter you designed. There are no forwarding paths, but you can write and read data to/from the regfile on the same cycle (as presented in lecture). Complete the pipeline diagram for the following assembly program executing on both datapaths at the same time.

```
1 addi x2, x1, 100
2 sb x1, 0(x2)
3 lw x1, 4(x2)
4 sw x1, 4(x2)
```

	Core 1			**Core 2**		
Cycle	IF	EX	MWB	IF	EX	MWB
0	addi			addi		
1	sb	addi		sb	addi	
2	lw	sb	addi	lw	sb	addi
3						
4						
5						
6						
7						
8						
9						

Let's now turn to a superscalar datapath that fetches **two** instructions per cycle from the IMEM, and has two execution units (ALUs) to double the instruction throughput. The 2 instructions in the RF stage are allowed to dispatch to the EX/M stage if their required execution unit is free and there are no data hazards. A diagram is shown below:



(e) Does the superscalar pipeline have any structural hazards? If so, write a small assembly snippet that exposes the hazard.

Circle your answer

Yes

No

(f) Fill in a pipeline diagram for the following program. The '1' and '2' suffixes are used to distinguish the 2 instruction slots in each pipeline stage.

```
1 ori x1, x2, 0x2
2 addi x1, x2, 100
3 xor x3, x4, x5
4 and x2, x4, x5
5 lw x3, 0(x3)
6 sw x2, 0(x3)
```

Cycle	IF 1	IF 2	RF 1	RF 2	EX/M 1	EX/M 2
0	ori	addi				
1	xor	and	ori	addi		
2						
3						
4						
5						
6						
7						
8						

(g) **(EECS251A Students Only)** Often the data bus widths of a cache and a backing memory don't match. We'll design a width converter that bridges a 8-bit ready-valid source to a 16-bit ready-valid sink.

You will use two FIFOs to perform this task. Assume the FIFOs produces `dout` on the *same cycle* that `rd_en` is asserted (this is also known as a first-word-fall-through FIFO). (i.e. you don't have to wait a cycle after `rd_en` is asserted to get the right `dout`).

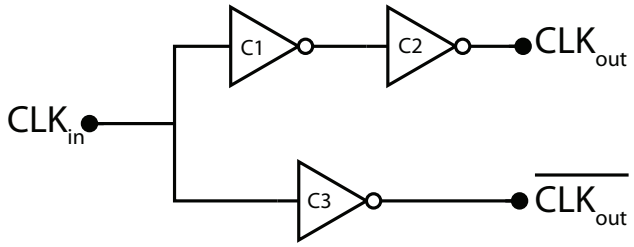
```
module width_converter (
    input clk, rst,
    input [7:0] data_in,
    input data_in_valid,
    output data_in_ready,

    output [15:0] data_out,
    output data_out_valid,
    input data_out_ready
);
  reg x_reg;
  wire f1_empty, f1_full;
  wire f2_empty, f2_full;
  always @(posedge clk) begin
    if (rst) x_reg      <= (1);
    else if ((2)) x_reg <= (3);
  end
  fifo f1 (
    .clk(clk), .rst(rst),
    .din(data_in),
    .wr_en((4)),
    .dout(data_out[7:0]),
    .rd_en((5)),
    .empty(f1_empty),
    .full(f1_full)
  );
  fifo f2 (
    .clk(clk), .rst(rst),
    .din(data_in),
    .wr_en((6)),
    .dout(data_out[15:8]),
    .rd_en((7)),
    .empty(f2_empty),
    .full(f2_full)
  );
  assign data_in_ready  = (8);
  assign data_out_valid = (9);
endmodule
```

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____
9. _____

3. Delay and Energy (Midterm 2 Clobber 1) (30 mins, 30 points)

(a) Circuit from the figure below takes the input clock, CLK_{in}, and generates two complementary clocks CLK_{out} and $\overline{CLK_{out}}$. CMOS inverters are symmetrically sized with input capacitances C1, C2 and C3. Drain capacitances are equal to gate capacitances.



If the total capacitance seen by the input CLK_{in} is 1pF and each of the loads on CLK_{out} and $\overline{CLK_{out}}$ is 10pF, find three equations that when solved for will give you the optimal sizing of C1, C2 and C3 such that the output clock phases are complementary with minimum delay.

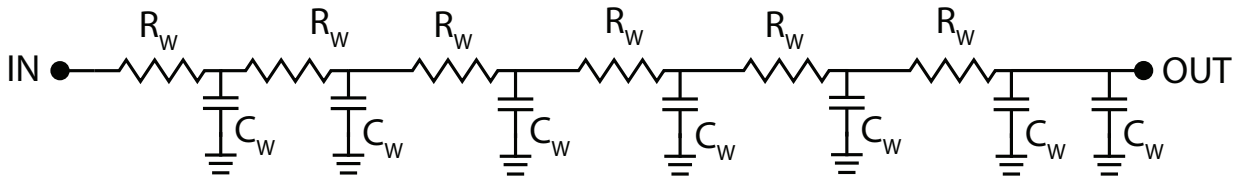
Equation1 :

Equation2 :

Equation3 :

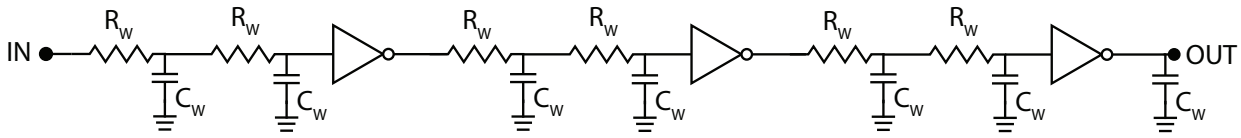
(b) In this question, we will investigate the delay due to a wire, and how it changes with inverters that buffer the signal along the path.

i. What is the propagation delay along this wire segment from IN to OUT? Express your answer in terms of circuit parameters, R_W and C_W.



t_p =

ii. Now we'd like to see how the delay scales as we buffer between these segments. First replace the inverters with their equivalent RC models. Given that the input and output capacitance of an inverter is $N \cdot C_{inv}$ and the output resistance of the inverter is $\frac{R_w}{N}$ (where N is the size of the inverters, each of them being the same size), express the propagation delay along the line in terms of circuit parameters. Then, derive an expression for the optimal size (N_{opt}) of the inverters that minimizes this delay.



t_p =

N_{opt} =

4. Arithmetic (Midterm 2 Clobber 2)(30 mins, 30 points)

In this problem we will analyze and compare a few different 16-bit adders. Inputs are A_{15}-A_0 and B_{15}-B_0, outputs are S_{15}-S_0. In all diagrams, white squares compute propagate and generate signals for each bit ($P_i = A_i + B_i$, $G_i = A_i B_i$). The complements of these signals are also available.

A ripple carry adder is shown in Figure 1. Each gray circle implements the generate equation $G_{i:k} = G_{j:k} + P_{j:k} \cdot G_{i:j-1}$ (for a ripple carry this would simplify to $G_{0:k} = G_k + P_k \cdot G_{0:k-1}$), where the indices i, j, k ($i \leq j \leq k$) correspond to the index of the input propagate and generate signals.

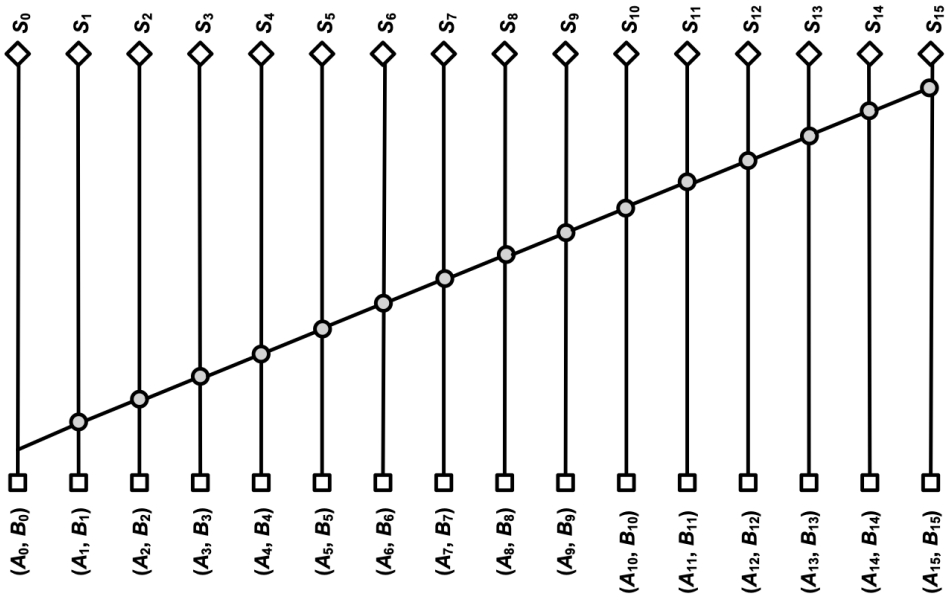


Figure 1: Ripple-carry Adder.

(a) You have been asked to implement this adder in CMOS and you realize that each of carry stage needs to be implemented as an inverting CMOS and-or-invert (AOI) gate ($\overline{A \cdot B + C}$), followed by an inverter to implement the correct logic.
Alternatively, you may be able to use two types of gates in alternating bit positions and reduce the number of gates in the critical path. **Draw and size** the CMOS logic gates that correspond to grey circles at **odd and even bit positions** in Figure 1.

Odd **Even**

(b) For each gate, what is the logical effort of the gate's input that's on the critical path.

LE_{odd} =

LE_{even} =

(c) What is the critical path delay for the adder in Figure 1, if the diamond boxes have the same input capacitance as the grey dots and all grey dots are sized the same? Assume the delay of the diamond and square boxes ares $t_{diamond}$ and t_{square}.

Delay =

(d) Now you would like to build an even faster adder. A diagram of a carry-lookahead adder is shown in Figure 2. Each white triangle is just a buffer. Grey dots perform the same logic operation as those in part a). black dots implement the function of grey dots, and, in addition, compute group propagate signals $P_{i:k} = P_{i:j-1}P_{j:k}$

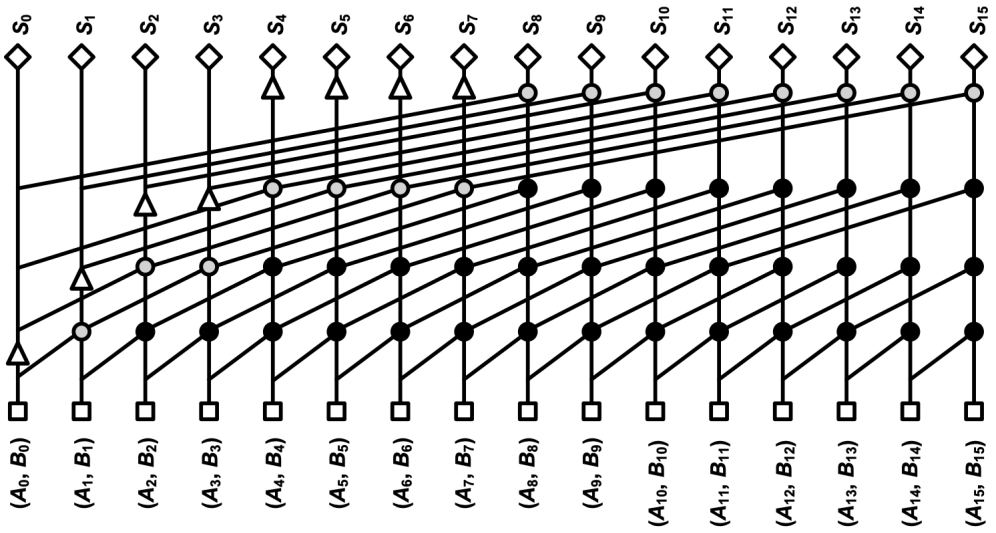


Figure 2: Carry-lookahead adder.

What is the radix of tree in Figure 2?

Radix =

(e) You realize again that you may be able to use two types of gates in alternating rows in Figure 2 to reduce the number of inverters in the critical path. Draw the CMOS logic gates that correspond to black circles at odd and even rows in Figure 2.

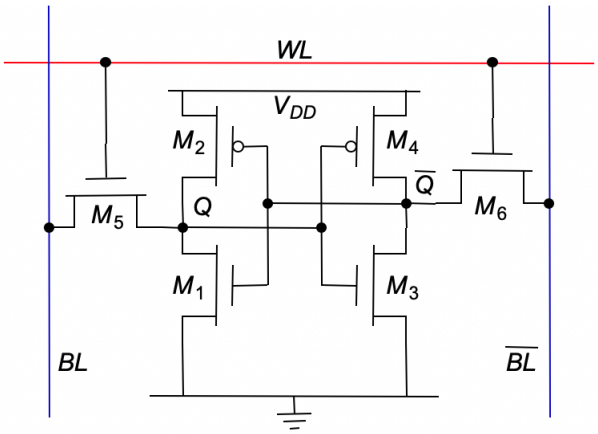
Odd **Even**

(f) What is the overall branching effort of the critical path of this tree (Assume all gates have the same input capacitance).

Branching Effort =

5. SRAM (25 mins, 20 points)

Consider a 256-word SRAM array where each word is 128 bits wide. The SRAM cells, shown in the figure below, are sized to be as small as possible but stable. SRAM cell access transistors are minimum length, and the cell area is $0.2\mu m$ x $0.2\mu m$. Gate capacitance equals drain capacitance Cg = Cd = $2fF/\mu m$. The decoder consists of a 4-bit predecoder and final row decoders. The final row decoder consists of a NAND gate and an inverter.



(a) Assuming that all the transistors have the same length and that during a read, the bitlines are precharged to V_{dd}. If the width of M5 is 30 nm, what are the right width combinations of M2 and M1 would guarantee both read and write stability?

A. $W_{M2} = 15nm$ and $W_{M1} = 45nm$

B. $W_{M2} = 45nm$ and $W_{M1} = 15nm$

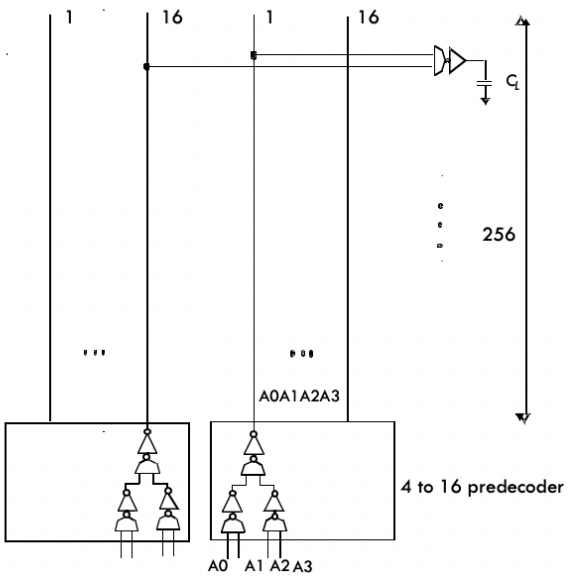
C. $W_{M2} = 30nm$ and $W_{M1} = 30nm$

Answer =

(b) If the wordline has the capacitance per unit length of $C_{WL} = 0.1fF/\mu m$, what is the total capacitive load of the row decoders?

C_{rd_load} =

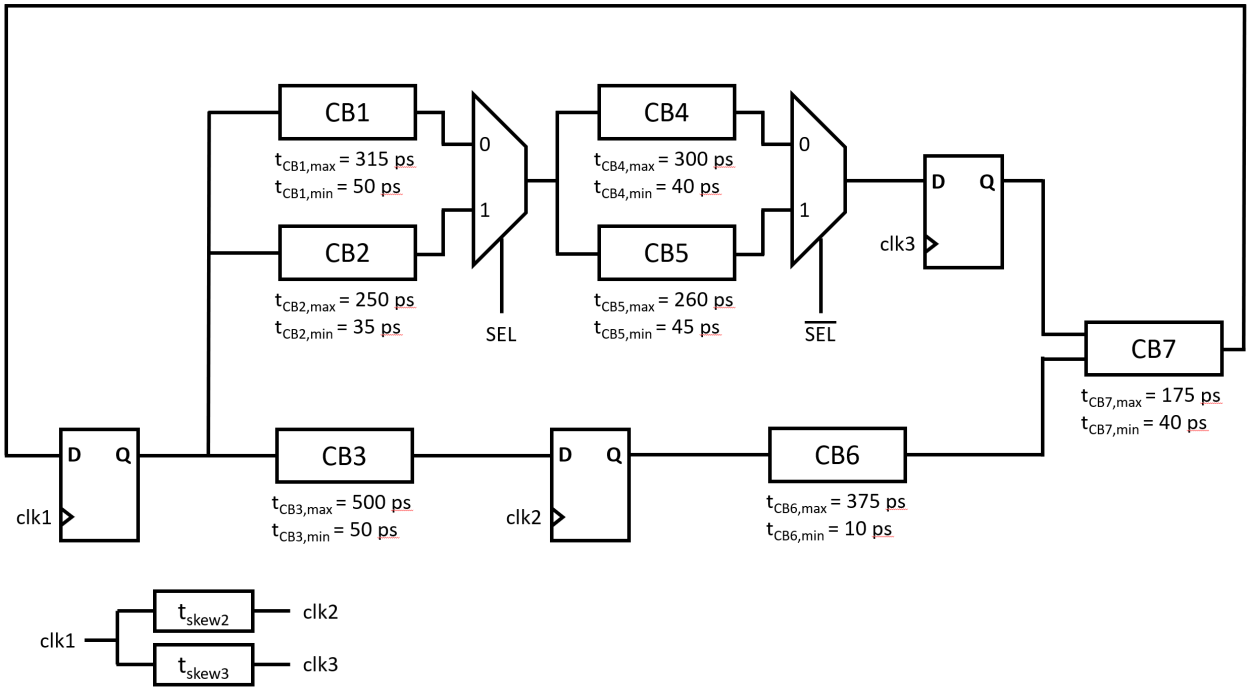
(c) The predecoder decodes 4 bits, and drives a metal line with $C_{metal} = 0.01fF/\mu m$, as shown in the figure below. What is the total capacitance load on the predecoder? Use C_{rd} to represent the capacitance of a row decoder.



$C_{predecoder\_load}$ =

6. Timing Analysis (40 mins, 40 points)

In this problem, you are asked to perform setup and hold timing analyses of the given circuit. Each flip flop is identical with a clock-to-Q delay of $t_{c-q} = 50ps$, setup time of $t_{setup} = 75ps$ and a hold time of $t_{hold} = 85ps$. The signal SEL (and $\overline{SEL}$) is an input to the circuit, and while its logic value is unknown to you, assume that it is constant during normal operation.



(a) Given that there is no skew between the clock signals, what is the minimum clock period this circuit can operate with?

$T_{clk,min}$ = ps

(b) If $t_{skew2} = -10ps$, $t_{skew3} = 10ps$, is there a timing violation in this circuit with a clock period of $T_{clk} = 725ps$? Denote your timing analysis in terms of Setup and Hold slacks, where a negative slack would mean a violation.

Setup Slack= ps

Hold Slack = ps

(c) Using the same skew values from part (b), but now also considering there is a **cycle-to-cycle jitter** in clk1, with a value of $t_{jitter} = 40ps$, is there a timing violation in this circuit with a clock period of $T_{clk} = 725ps$? Denote your timing analysis in terms of Setup and Hold slacks, where a negative slack would mean a violation.

Setup Slack= ps

Hold Slack = ps

(d) Considering there is no jitter, if you were free to set the values of t_{skew2} and t_{skew3} (including negative values), what would you set them as to operate the circuit with the smallest possible clock period while ensuring there are no timing violations? What is the minimum clock period this design can work with?

$t_{skew2} =$ ps

$t_{skew3} =$ ps

$T_{clk,min} =$ ps