Homework #4 v20180516

(Deadline: 11:59PM PDT, Wednesday June 6, 2018)

Name (Last, First): YANG RYAN

Student Id #: 404 904 494

INSTRUCTIONS

This homework is to be done individually. You may use any tools or refer to published papers or books, but may not seek help from any other person or consult solutions to prior exams or homeworks from this or other courses (including those outside UCLA). You're allowed to make use of tools such as Logisim, WolframAlpha (which has terrific support for boolean logic) etc.

You must submit all sheets in this file based on the procedure below. Because of the grading methodology, it is much easier if you print the document and answer your questions in the space provided in this problem set. It can be even easier if you answer in electronic form and then download the PDF. Answers written on sheets other that the provided space will not be looked at or graded. Please write clearly and neatly - if we cannot easily decipher what you have written, you will get zero credit

SUBMISSION PROCEDURE: You need to submit your solution online at Gradescope (https://gradescope.com/). Please see the following guide from Gradescope for submitting homework. You'd need to upload a PDF and mark where each question is answered.
http://gradescope-static-assets.s3-us-west-2.amazonaws.com/help/submitting_hw_guide.pdf

Problem #1

The following is pseudo-code for an algorithm (Newton's method) for calculating square root, *y=root(x)*. The input, *x*, and output, *y*, are floating point numbers. The computation is triggered with a signal, *go=1'b1*, for 1 cycle that loads the input to a register that holds *x*. The computation completes by asserting a signal, *done=1'b1*, for 1 cycle. Implement this code as a controller and datapath.

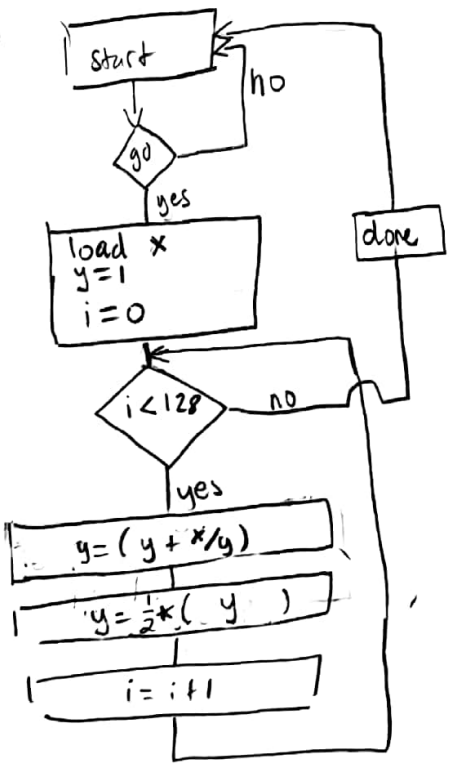
```
initial
        y = 1
        i = 0
while (i<128) {
        y = 0.5*(y + x/y)
        i++
}
```

(a) Draw a flow chart for this algorithm.

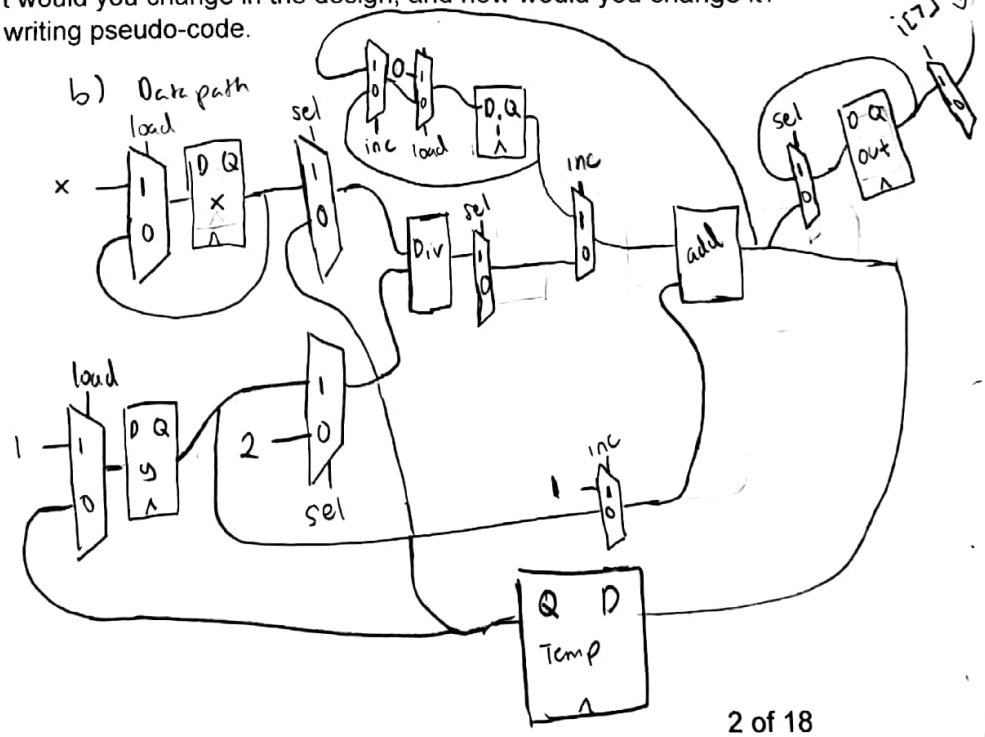
(b) Draw the controller FSM state diagram and the datapath using building blocks that are available below. Be sure to indicate the outputs that control the datapath and inputs from the datapath. Write any boolean logic needed for those control signals of the datapath. For the datapath, you can assume available to you the following building blocks: multiplexers (any number of inputs), any number of registers, one magic adder that can add either floating point or integer, and one magic divider. The variable, *y*, is the iterated solution to the algorithm. Use as few states as possible in your controller. You don't need to do any checks on the bounds of the floating point number. Assume a *START* state where the FSM is initialized. Upon completion, return to the *START* state.

(a) If instead of iterating for 128 times, we are looking for a certain percent accuracy, for instance, 1%. What part would you change in the design, and how would you change it? Show the change in by writing pseudo-code.

a)



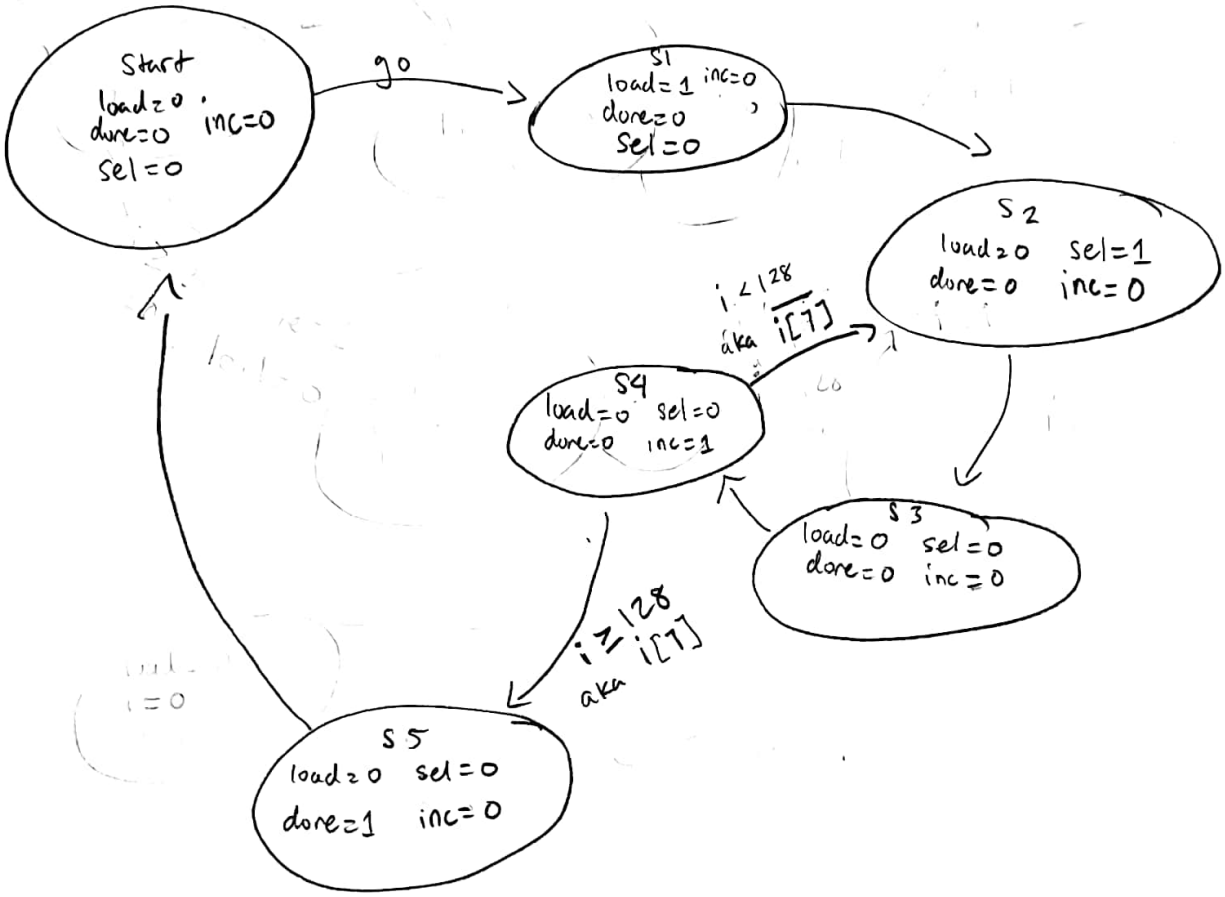
b) Data path



(IMPORTANT: Keep this page in submission even if left unused)

b)

State diagram



c) $\frac{|\text{actual value} - \text{value}|}{\text{actual value}}$ = percent accuracy

$$\frac{|y-\sqrt{x}|}{\sqrt{x}} \leq .01 \Rightarrow |y-\sqrt{x}| \leq .01\sqrt{x} \quad \text{or} \quad |y-\sqrt{x}| \geq -.01\sqrt{x}$$

$$y \leq .01\sqrt{x}+\sqrt{x} \qquad y \geq 0.99\sqrt{x}$$

$$y \leq 1.01\sqrt{x} \qquad y \geq 0.99\sqrt{x}$$

$$y^2 \leq (1.01)^2 x \qquad y^2 \geq (0.99^2)\, x$$

↓ continued on back

(IMPORTANT: Keep this page in submission even if left unused)

c)

Pseudo code

```
y = 1;
while( y^2 > (1.01)^2 * x && y^2 < (0.99)^2 * x )
    y = 0.5*(y + x/y);
```

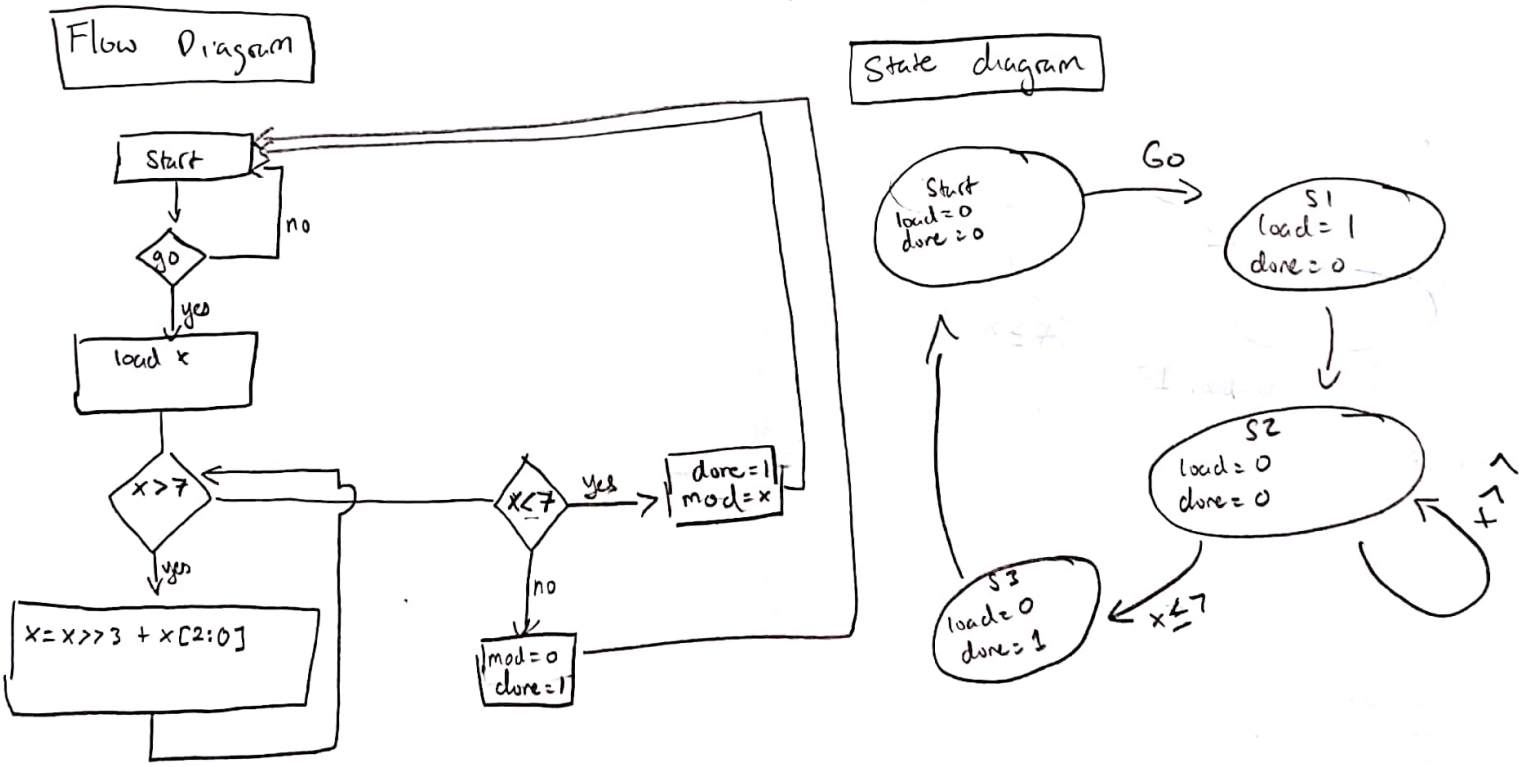
3

Problem #2

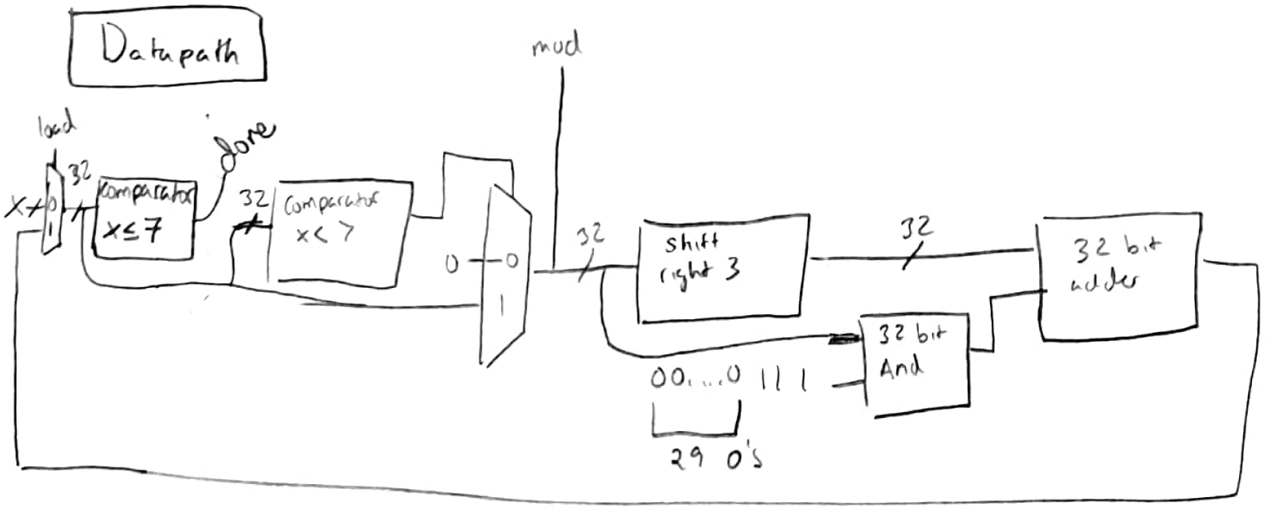
The following is pseudo-code for calculating modulo-7 in a binary (fixed-point) system. The input, *x[31:0]*, and output, *mod*, are 32-bit integers. The computation is triggered with a signal, *go=1'b1*, for 1 cycle that loads the input word to register *x*. The computation completes by asserting a signal, *done=1'b1*, for 1 cycle.

```
while (x > 7) {
        x = x >> 3 + x[2:0]
}
mod = (x < 7) ? x : 0;
```

Implement this code as a controller and datapath. In your implementation. Similar to problem #1, first draw a flow diagram. Then draw a datapath and a controller FSM state diagram. Write any boolean expressions for the logic for any control signals to the datapath. For the datapath, as an added constraint, use as few hardware blocks or gates as possible. You are not constrained by the number of states. Indicate what building blocks or logic gates you need. Assume a *START* state where the FSM is waiting for the *go=1'b1*. Upon completion, return to the *START* state.The *mod* output of the previous computation should be asserted and maintained in the *START* state until the next computation starts.



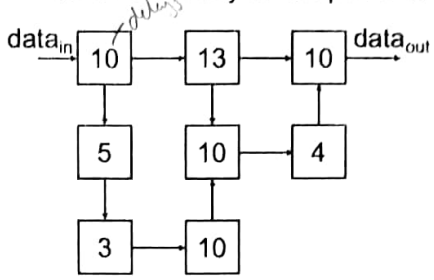
(IMPORTANT: Keep this page in submission even if left unused)



(IMPORTANT: Keep this page in submission even if left unused)

Problem #3

The following shows delays of blocks of logic that are atomic (cannot be further divided). If $data_{in}$ are from registers and $data_{out}$ are stored by registers. The registers have t_{dCLK-Q} = 4, t_{setup} =2 and t_{hold} = -1. Note that there is no contamination delay in the problem.



(a) What is the cycle time of this logic?

(b) If we are to minimize the cycle time, show on the figure below where you would insert registers. How many would you insert (show each register with a vertical line through the arrows)?

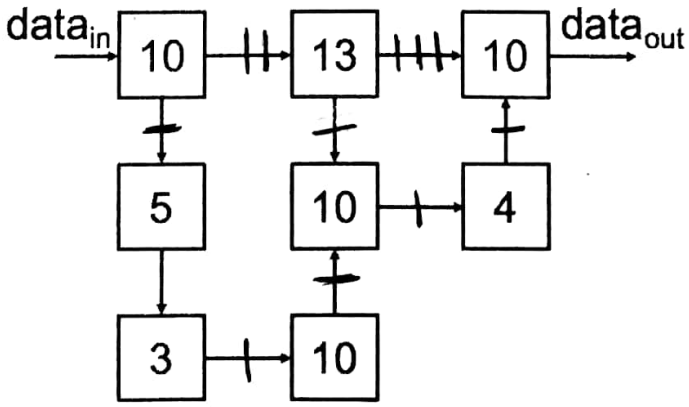
(c) What is the cycle time of the design in (b)? What is the latency in (b)?

(d) Note that the design in (b) requires a lot of registers in order to minimize the cycle time. Meanwhile by relaxing the cycle time slightly to be <22, one can come to a design that uses far fewer registers without sacrificing much cycle time. Show on the figure below where you would insert the registers. How many would you insert?

(e) What is the cycle time of the design in (d)? What is the latency in (d)?

a) $T_{clk} \geq t_{pclk->Q} + t_{pLogic} + t_{setup} = 4 + 2 + 52 = 58$

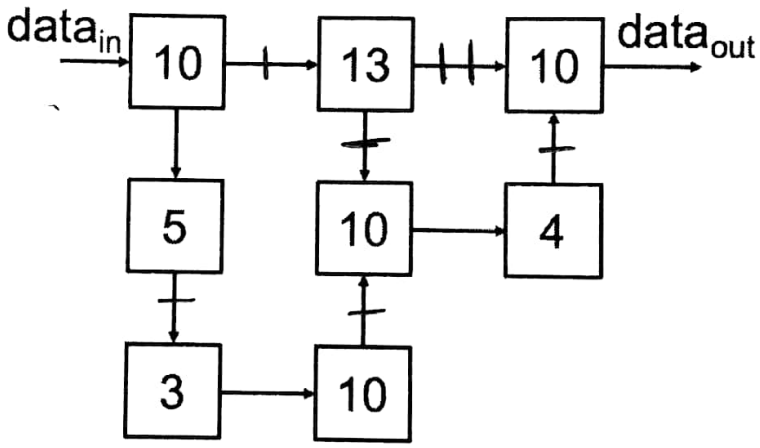
Figure provided to answer (b)



c) 19 = cycle time latency = 114

(IMPORTANT: Keep this page in submission even if left unused)

Figure provided to answer (d)

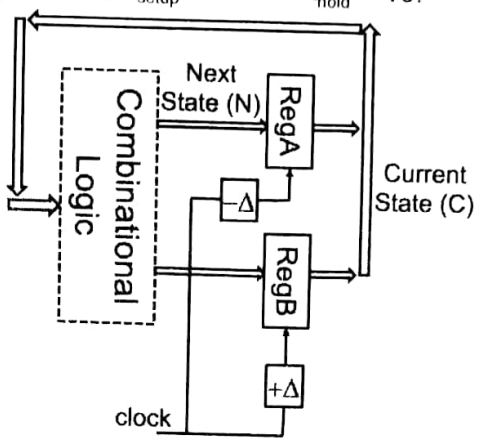


e) cycle time = 21
Latency = 84

(IMPORTANT: Keep this page in submission even if left unused)

Problem #4

The following shows a conceptual state machine. The registers have the following characteristics: $(t_{cCLK\text{-}Q}, t_{dCLK\text{-}Q}) = (5,40)$, $t_{setup} = 20$ and $t_{hold} = 10$.



(a) If the combinational logic has delay of $(t_{cCN}, t_{dCN}) = (10,200)$, and clock skew of $\Delta=0$, what is the minimum cycle time?
(b) With the same delay as in (a), is there a hold time violation?
(c) What is the amount of clock skew, Δ, such that there would be a hold time violation?
(d) Assuming the Δ in (c), what is the new minimum cycle time?

a) $t_{clk} \geq t_{pClk\to Q} + t_{pLogic} + t_{setup}$ $\quad t_{hold} \leq t_{cClk\to Q} + t_{cLogic}$

$t_{clk} \geq 40 + 200 + 20$

$t_{clk} \geq 260$

b) $t_{hold} \leq t_{cClk\to Q} + t_{cLogic}$

$10 \leq 5 + 10$

$10 \leq 15$

No there is not

c) $t_{hold} \leq t_{cClk\to Q} + t_{cLogic} - 2\Delta$ $\quad = 10 \leq 15 - 2\Delta$

$= -5 \leq -2\Delta$

$5 \leq 2\Delta$

$\frac{5}{2} \leq \Delta$

Skew would be any value greater than 2.5

d) $t_{clk} \leq t_{Clk\to Q} + t_{pLogic} + t_{setup}$

minimum cycle time = anything greater than 265

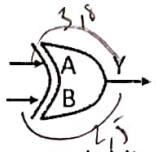
(IMPORTANT: Keep this page in submission even if left unused)

(IMPORTANT: Keep this page in submission even if left unused)

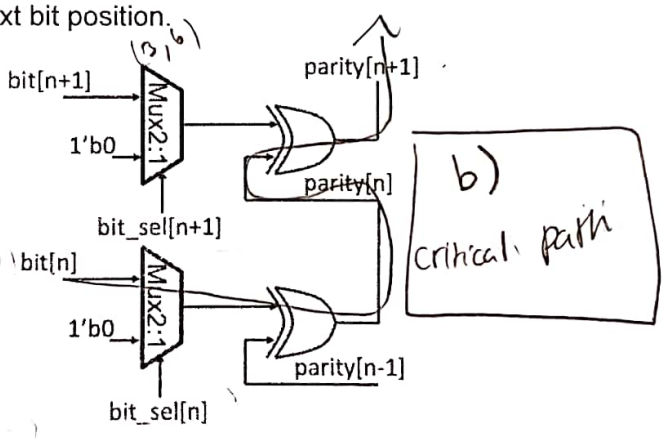
Problem #5

Hamming code is a way to correct errors when storing and/or transmitting information. This problem does not require you to know the details of a Hamming code. With a 12-bit word of data, *d[11:0]*, typically an additional 4 bits, *p[3:0]*, of information is appended to be able to correct for any errors in the resulting 16 bits. Each of the *p[3:0]* ensures parity for 8 of the 16 bits. The 8 bits for each *p[3:0]* are a carefully chosen combination of the 16 bits. Each of the 4 parity is calculated by XORing 16-bits together, and 8 of those bits are forced to 1'b0. By choosing which of the 8-bits to force to 1'b0, one can adjust and program the Hamming code.

Parity is checked by XORing 16 inputs together (8 of the inputs are selected to be 1'b0). The XOR gate you are to use is shown below. The propagation delay from A-Y, ($t_{cA\text{-}Y}$, $t_{dA\text{-}Y}$) = (3, 8), and delay from B-to-Y, ($t_{cB\text{-}Y}$, $t_{dB\text{-}Y}$) = (2, 5). Notice that the two inputs are not symmetric. The MUX that selects between the 1'b0 input versus the data bit has propagation delay from M-Y, ($t_{cA\text{-}Y}$, $t_{dA\text{-}Y}$) = (3, 6).



A bit-cell style design is shown below where each bit position calculates the parity with the input and passes the result to the next bit position.



(a) To minimize the critical path delay of the entire parity checker, which input of the XOR (A or B) would you assign the MUX output? A

(b) What is the critical path of this circuit, show on a logic gate diagram? It goes through 1 mux & 16 xor's

(c) What is the delay of this circuit (contamination and propagation)?

contamination = 6

propagation = 6+8+15(5) = 89

Assume that the inputs are from flip-flops (DFF) and the parity output is an input to a flip-flop. The flip-flop (DFF) has the following characteristics: ($t_{cCLK\text{-}Q}$,$t_{dCLK\text{-}Q}$) = (3, 4), t_{setup} =2 and t_{hold} = 1.

(d) What is the minimum cycle time?

minimum cycle time = 4 + 2 + 89 = 95

To speed up the parity calculation, we can pipeline the design so that the calculation is performed in two cycles. Each cycle would compute the parity for only 8 bits so the cycle time is shorter.

(e) Show the change in the design needed to pipeline the design.

(f) The pipelining enables higher throughput. What is the new cycle time?

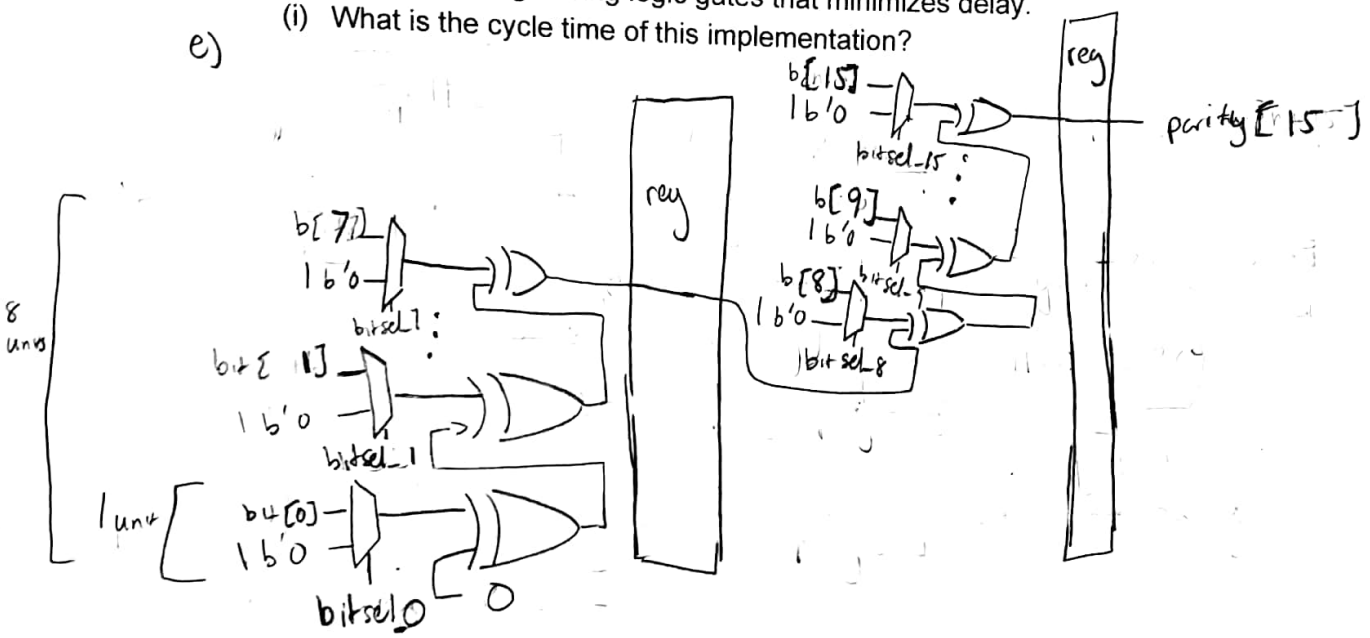
(g) With the cycle time in (f), what is the latency for computing the parity?

As an alternative to pipelining, the logic can be restructured so that multiple bits are processed in parallel in a single cycle instead of serially

(h) Draw the design using logic gates that minimizes delay.

(i) What is the cycle time of this implementation?

e)



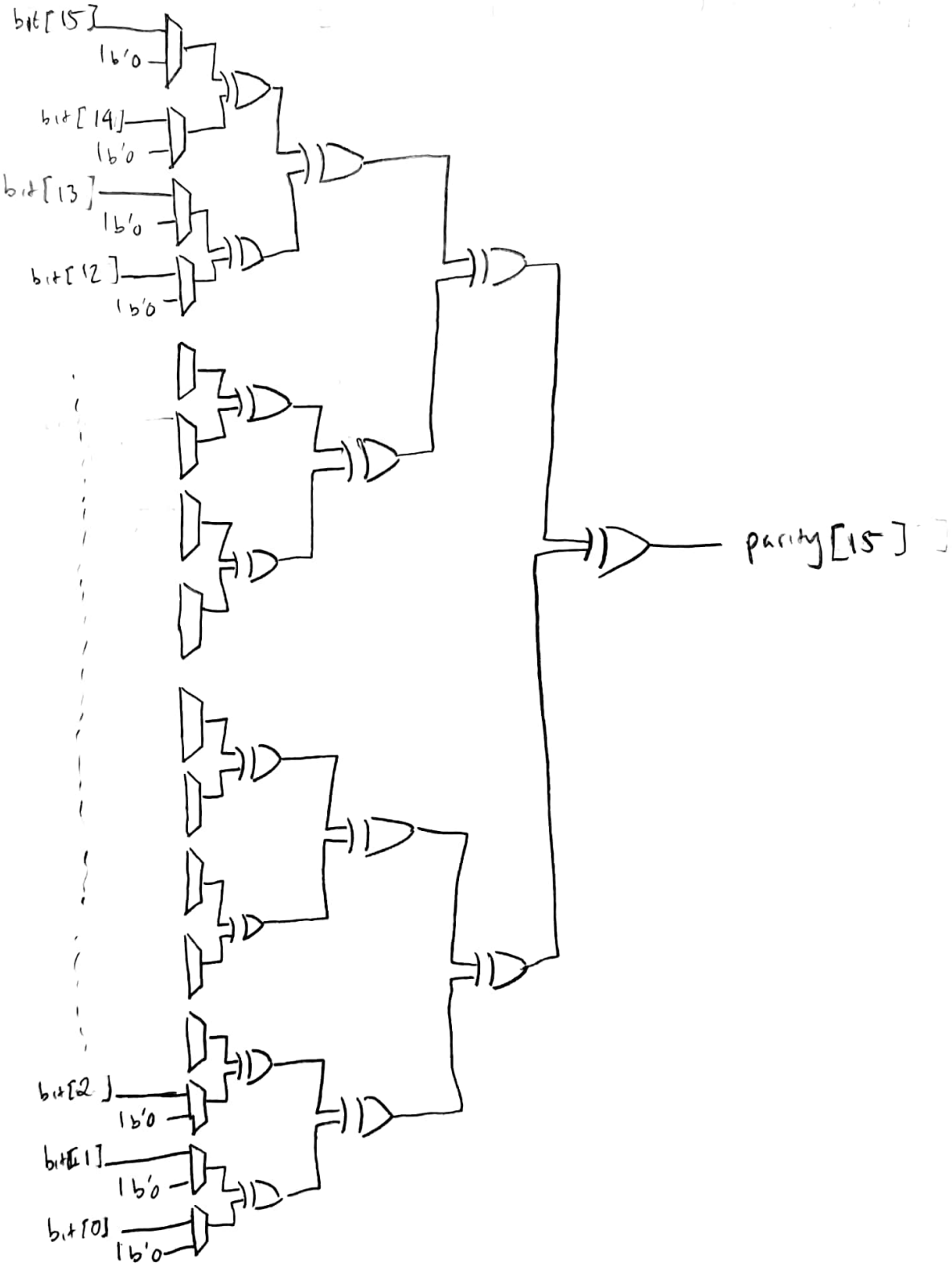
f) Cycle Time = $T_{clk} \geq T_{setup} + T_{dclk \to q} + T_{d\ logic}$

$4 + 2 + 8 + 7(5) + 6 = \boxed{55}$

g) Latency time = 2 x Cycle = $\boxed{110}$

(IMPORTANT: Keep this page in submission even if left unused)

h)



(IMPORTANT: Keep this page in submission even if left unused)

i) Cycle time = $T_{clk} \geq t_{setup} + t_{d\,logic} + t_{d\,clk->Q}$

$= T_{clk} \geq 2 + 4 + 6 + 8(4)$

$= T_{clk} \geq 44$

(IMPORTANT: Keep this page in submission even if left unused)