Homework #4

(Deadline: 11:59PM PDT, Wednesday June 7, 2017)

Name (Last, First): Miller, Joseph

Student Id #: 504744848

INSTRUCTIONS

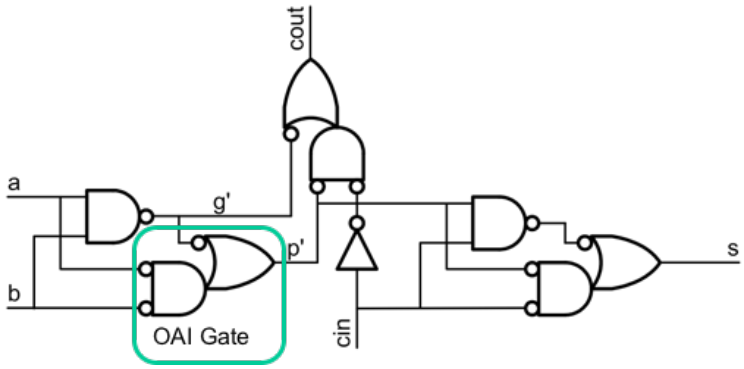
This homework is to be done individually. You may use any tools or refer to published papers or books, but may not seek help from any other person or consult solutions to prior exams or homeworks from this or other courses (including those outside UCLA). You're allowed to make use of tools such as Logisim, 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 logic for a Full-Adder (FA) (from the lecture notes) is shown in the figure below. Three logic gates are used in the implementation: INV, 2-input NAND, and OAI (OR-AND-INV – push the bubbles will reveal the reason for the name). INV delay is 1ns and NAND delay is 2ns for either input. The OAI gate has 2 possible delays, for the NAND input, the delay is 1.5ns (note the difference with a 2-input NAND logic gate), and for the OR input, the delay is 3ns.



(a) Determine the contamination delay (min) for t_{c_a2cout}, t_{c_a2s}.
(b) Determine the propagation delay (max) for t_{d_a2cout}, t_{d_a2s}.
(c) For a ripple carry adder of 4 bits, determine the contamination and propagation delay for $t_{c\_cin[0]2cout[3]}$ and $t_{d\_cin[0]2cout[3]}$ where cin[0] is the carry in of the 0^{th} bit and cout[3] is the carry out of the 3^{rd} bit.
(d) If the ripple carry adder of (c) is used in a datapath for one stage of a pipeline, what is the minimum cycle time? Assume that the DFF has a t_{setup} of 0.5ns and t_{d_CQ} of 1ns.
(e) If instead of the design in (d), each bit adder is in its own stage of a pipeline, what is the minimum cycle time.
(f) For the design in (e), what is the hold-time requirement to not violate timing. Note that the DFF has a t_{c_CQ} of 0.5ns.
Answer the question for all parts in the space below.

a) t(c_a2cout) = 2 + 1.5 = 3.5 ns

t(c_a2s) = 3+ 3 = 6 ns

b) t(d_a2cout) = 2 + 1.5 + 3 = 6.5 ns

t(d_a2s) = 2 + 1.5 + 2 + 1.5 = 7 ns

c) t(c_cin[0]2cout[3]) = t(d_cin[0]2cout[3]) = 4(1 + 3) = 16ns

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

d) t(cy) >= t(dCQ) + t(dMAX) + t(s) = 1 + 16 + 0.5 = 17.5 ns

e) t(cy) = 4*t(dCQ) + t(dMAX) + t(s) = 4 + 16 + 0.5 = 20.5 ns

f) t(h) <= 4*(0.5) + t(dMin) = 2 + 16 = 18 ns

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

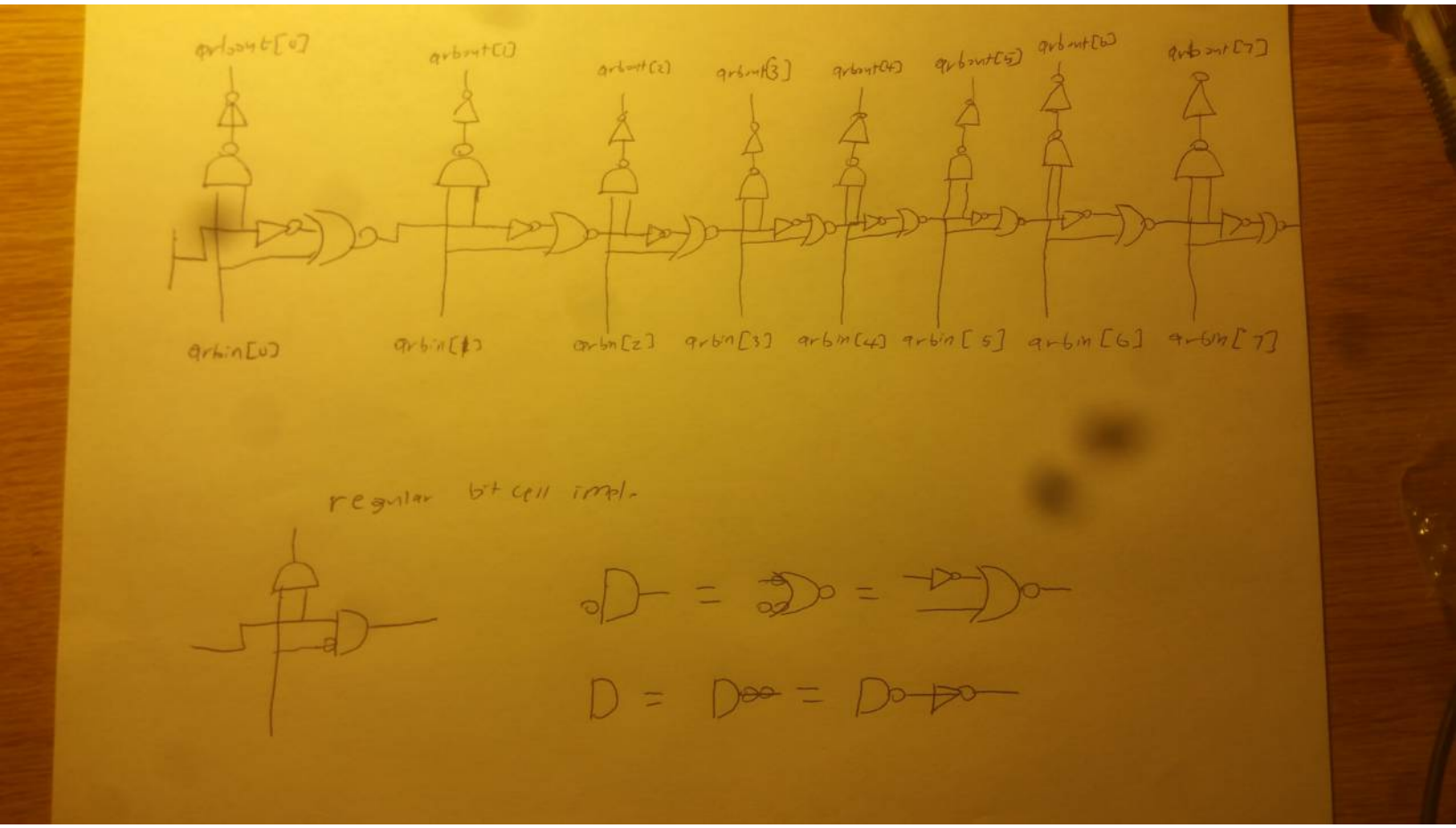
Problem #2

As taught previously, an arbiter for 8 input bits, *arbin*[7:0] outputs 8 "1-hot" output bits, *arbout*[7:0], that indicate the position of a 1 with highest priority. Bits with higher significance i.e. *arbin*[7] has priority over lower order inputs that are 1's. Use only INV, 2-input NAND and 2-input NOR logic gates in your design. The INV has a *delay* of 2, 2-input NAND has a *delay of 3* and the 2-input NOR has a *delay of 4*.

(a) Implement this function using a bit-cell approach (as shown in the notes). Show the design and determine the t_d and t_c for the bus, *arbout*[7:0]. Recall that t_d corresponds to the longest delay and t_c corresponds to the shortest delay.

(b) Repeat (a) but implement this function with a look-ahead approach (a wide multi-input AND function built as a tree of 2-input NANDs). Show the design and determine the t_d and t_c for the bus, *arbout*[7:0].
Answer the question for all parts in the space below.

a)



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

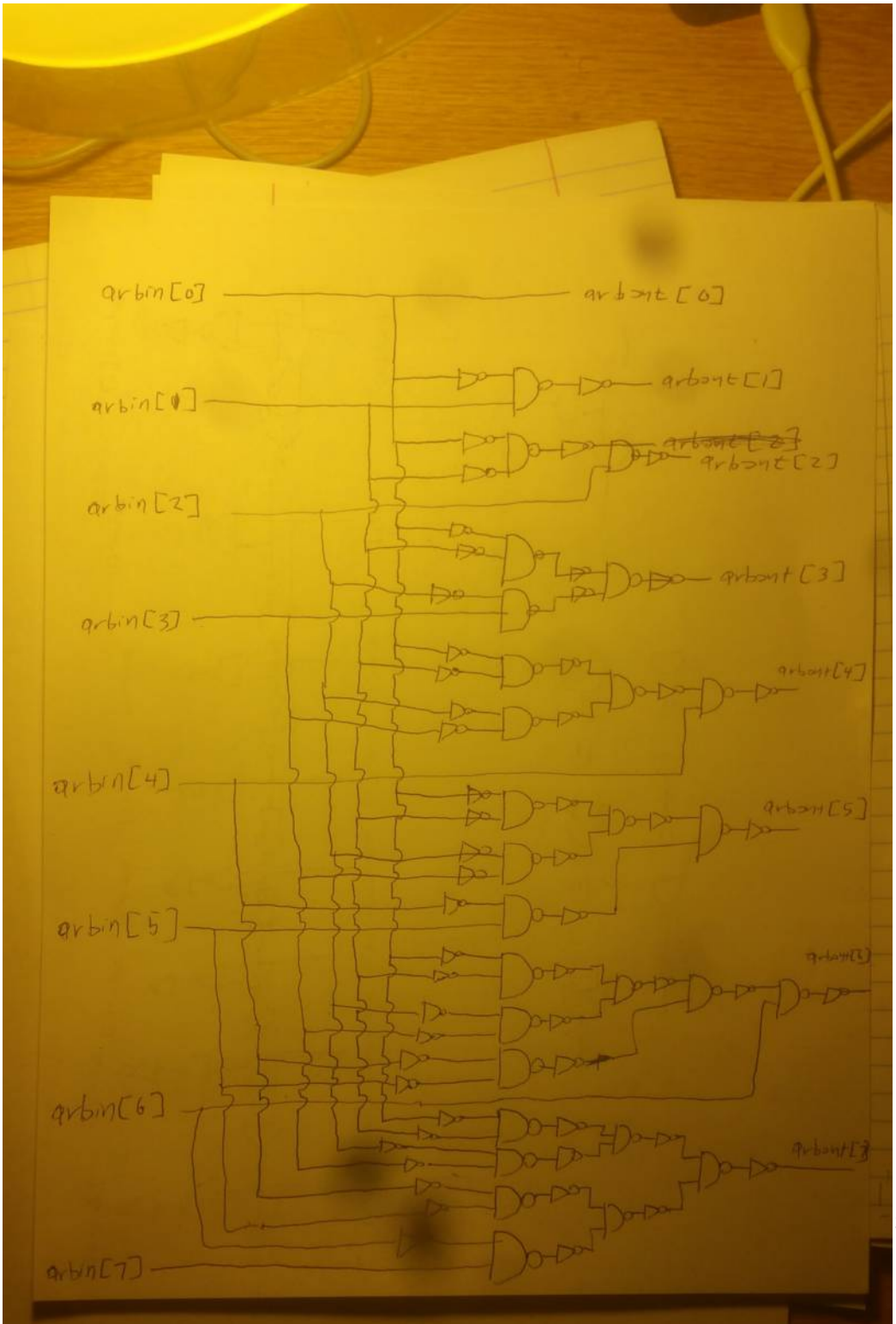
From arbin[0] the signal passes through 7 nors, and 6 inverters before reaching a nand and the final inverter therefore,

t(d) = 7(4) + 6(2) + 1(3) + 1(2) = 45 ns.

From arbin[0] the signal passes through the first nand then the inverter.

t(c) = 1(3) + 1(2) = 5 ns.

b)



going from arbin[0] to arbout[0] is the shortest delay as it passes through no gates.

t(c) = 0 ns.

going from arbin[0] to arbput[7] is the longest path that can be taken. It passes through 3 nands and 4 inverters therefore,

T(d) = 3(3) + 4(2) = 17 ns.

Problem #3

A standard positive-edge trigger Flip-Flop (DFF) is provided as a building block. The goal is to add logic (AND, OR, INV, or MUX) to the D, Q, CLK signals to implement additional functionality.

(a) Implement DFFR, where a RESET signal is added that implements a synchronous reset to the output when a clock edge arrives.

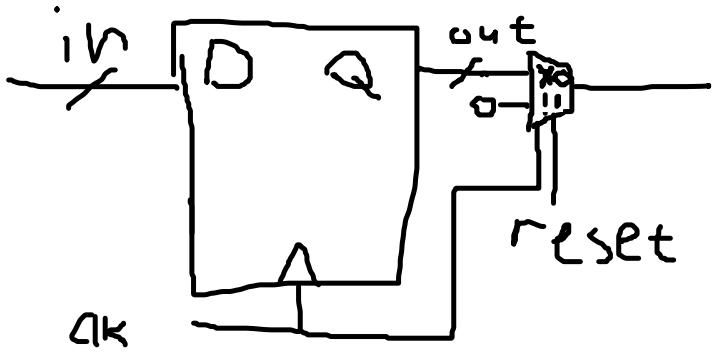
(b) Implement DFFS, where a SET signal is added that synchronously sets the output to 1'b1.

(c) Implement DFFCG, where a CLKSTOP signal is added that stops the clock to the Flip-Flop when CLKSTOP=1'b1.

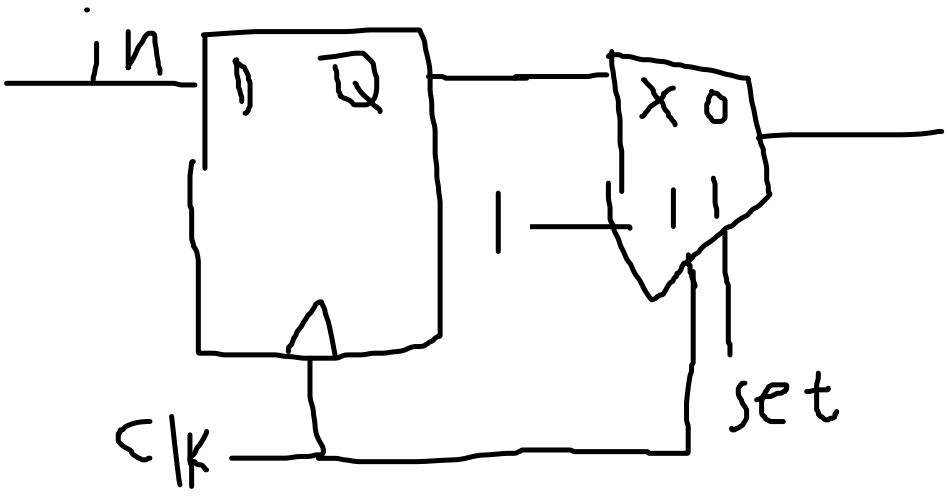
(d) Implement DFFRA, where a ARESET signal is added that implements an asynchronous reset to the output.

(e) What is the impact on t_{dCLK-Q}, t_{setup} and t_{hold} for the designs in (b), (c), and (d) as compared to the standard DFF in the problem description? Do they increase, decrease, or stay-the-same. *Answer the question for all parts in the space below.*

a)

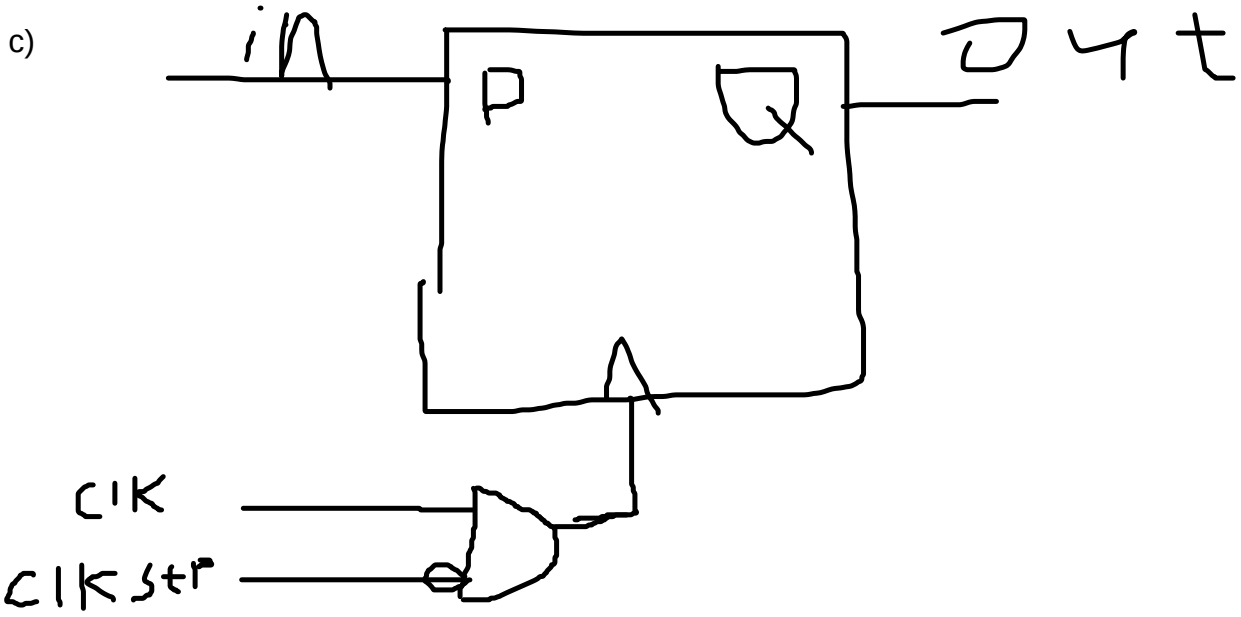


b)

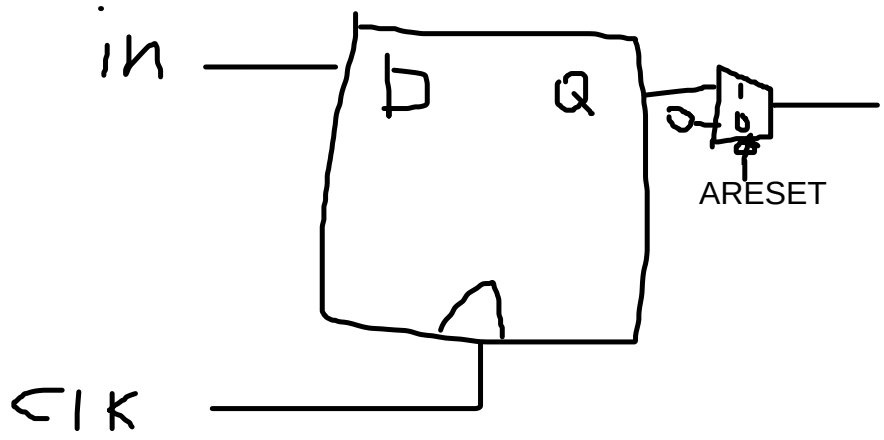


(IMPORTANT: Keep this page in submission even if left unused)

c)



d)



e) for b, t(dclk2Q) will remain unchanged. Since t(cmin) will go up with the addition of the multiplexer, t(h) will go up and t(s) will go down.

for c, there is a positive skew added by the and gate and inverter. This will cause t(dclk2Q) to increase, t(h) to decrease and t(s) to increase.

for d, t(dclk2Q) will remain unchanged. Since t(cmin) will go up with the addition of the multiplexer, t(h) will go up and t(s) will go down.

Problem #4

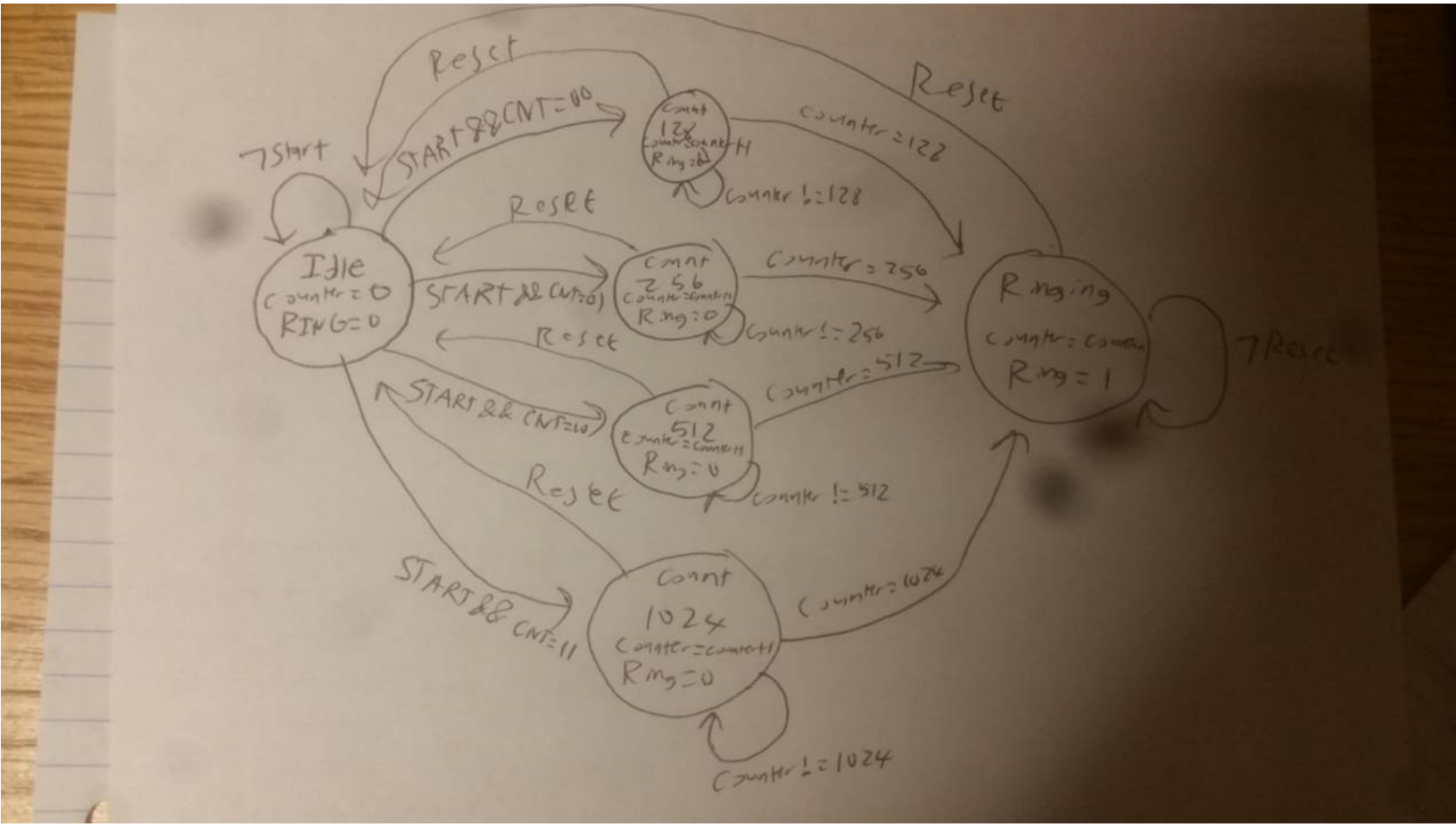
Design a programmable timer with the following description. The timer can be programmed to count differing amounts based on a 2-bit control input.

CNT[1:0]=2'b00, count 128
CNT[1:0]=2'b01, count 256
CNT[1:0]=2'b10, count 512
CNT[1:0]=2'b11, count 1024

An input signal, RESET, de-asserts the RING and zeros the counter. An input signal, START, initiates the counting. When the desired count is reached, an output signal, RING, is asserted and the counting stops. The signal RING stays high until the control state machine is RESET. The goal is to implement with a design that is partitioned into a controller and datapath where the primary element of the datapath is a 12-bit counter. You can choose if you want an up/down/load functions on your counter. You can use any other combinational building blocks such as multiplexers and comparators).

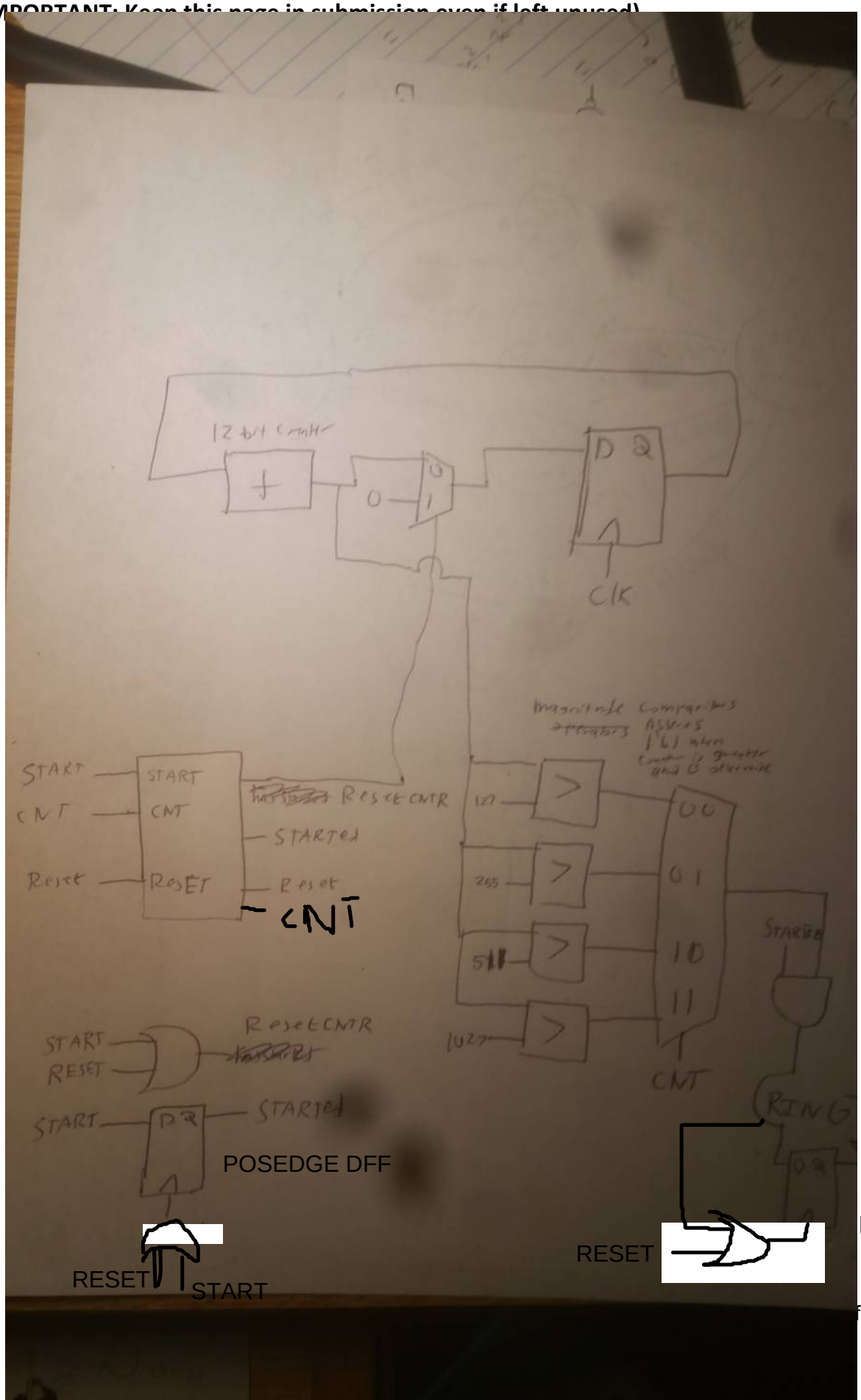
(a) Draw the input/output signals as well as the signals that interface between the controller and datapath.
(b) Draw the FSM for the controller.
(c) Show how signals of the controller is used within the datapath. Describe any logic with combinational building blocks.
Answer the question for all parts in the space below.

b)

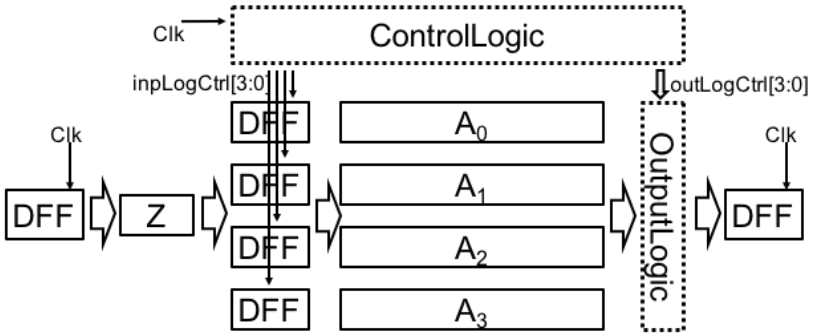


(IMPORTANT: Keep this page in submission even if left unused)

a) c)



Problem #5



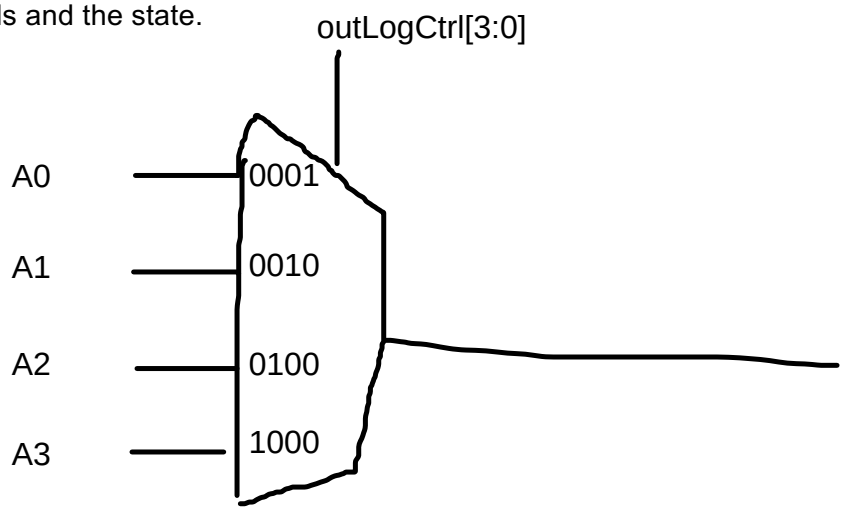
The speed of a pipeline is typically limited by the delay of the longest pipeline stage. The pipeline can be made faster by parallelizing a long delay path. For the diagram above, $delay_{A[0-3]}$ can be almost 4x longer than $delay_Z$ (the nominal delay per stage of a pipeline). In order to parallelize, block A is replicated 4 times into $A_{[3:0]}$. The DFF in the figure above refers to positive edge triggered D-type flipflops. The clock signal, *Clk*, is running at the full speed of the pipeline. Additional control signals, inpLogCtrl[3:0] and outLogCtrl[3:0], are needed. ControlLogic and OutputLogic are needed as well.

(a) What is the logic in OutputLogic that is needed to properly merge the 4 paths? Draw the logic gate(s) needed.

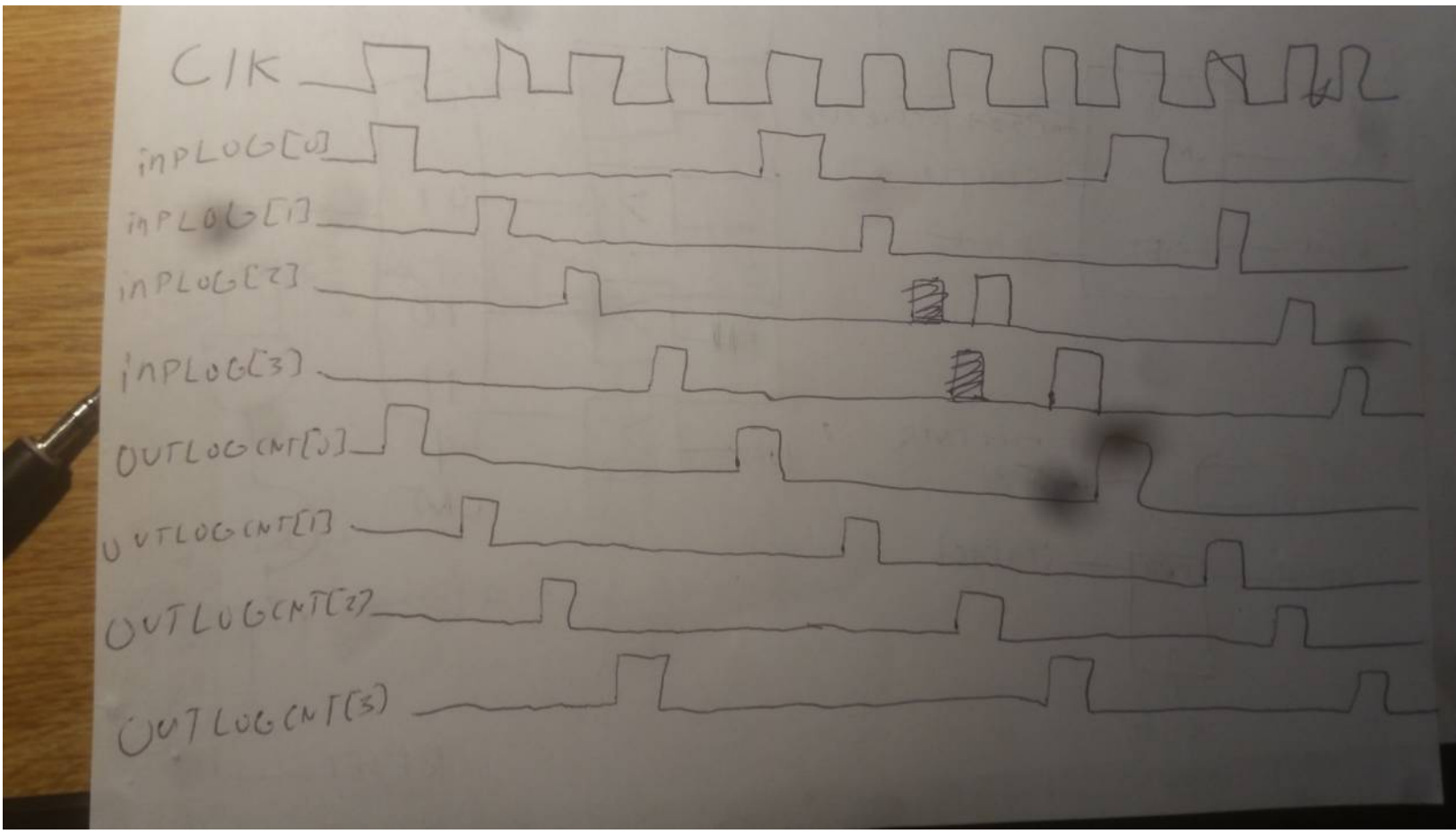
(b) What is the timing diagram for of the signals, inpLogCtrl[3:0] and outLogCtrl[3:0]? Hint: Draw these signals with respect to Clk to allow an input data sequence to be distributed (round-robin) to each path, $A_{[3:0]}$.

(c) Implement the FSM needed in ControlLogic. Hint: look at the timing diagram in (b). Consider a 1-hot encoded Moore FSM. You can also assume a Reset signal that initializes the control signals and the state.

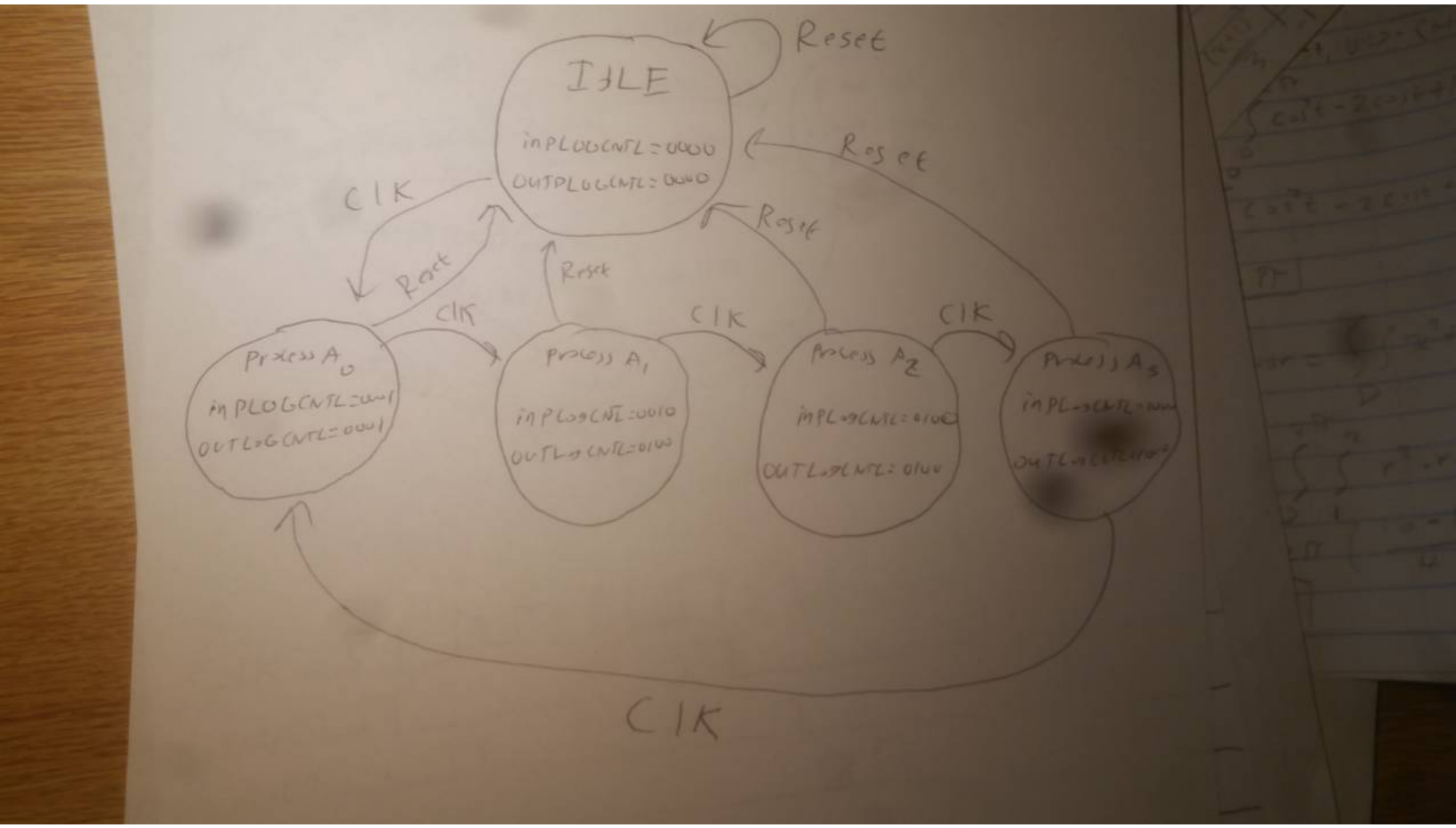
a)



b)


CLK
inPLOG[0]
inPLOG[1]
inPLOG[2]
inPLOG[3]
OUTLOGCNT[0]
OUTLOGCNT[1]
OUTLOGCNT[2]
OUTLOGCNT[3]

c)


IDLE
inPLOGCNTL=0000
OUTPLOGCNTL=0000
Reset
Reset
Reset
Reset
CLK
CLK
CLK
CLK
CLK
Process A0
inPLOGCNTL=0001
OUTLOGCNTL=0001
Process A1
inPLOGCNTL=0010
OUTLOGCNTL=0100
Process A2
inPLOGCNTL=0100
OUTLOGCNTL=0100
Process A3