Homework #3 v20180411

(Deadline: 11:59PM PDT, May 18, 2018)

Name (Last, First):

Student Id #:

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

A summation of weighted versions of multiple inputs is a very common building block for deep learning, FIR filters, soft decoders, etc. An example equation can be expressed as follows.

$$out[r-1:0] = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

where w_i are the multi-bit weights, and x_i are the multi-bit inputs. Such an expression when implemented as hardware would require five multiplications and four 2-input additions. As indicated in lecture, each multiplier also includes at least one addition, hence net 9 multi-bit adders are needed. Recall that additions have longer delay because the carry needs to be propagated and calculated through to the MSB of each word. Instead, hardware accelerates the operation using a method called Carry-Save where each instead of calculating the final result of each operation, we keep the binary numbers as 2 *n*-bit words as the intermediate result. The outputs (*carry* and *sum*) of a row of 1-bit Full Adder (FA) are kept separate. The full summation is done only when the final binary result, out, needs to be calculated by adding the *sum* word and *carry* word with full carry propagation. This idea is the same as an array multiplier we discussed in lecture where each partial product (PP) is not fully added and instead the *carry* and *sum* bits are added to the next partial-product.

(a) Assuming 8-bit unsigned integers as weights and inputs, the final result, *out*, can be a wide word. How many bits, *r*, are needed? How many bits wide does the final adder need?

(b) Assume that each 1-bit FA has unit delay, t_{FA}, implement the multiplication of one of the terms, w_ix_i, where both are 8-bit unsigned integers. For this design, minimize the number of unit delays from the x_i input to the multiplier output. As an added constraint, *carry* can only be propagated using FAs (using carry propagate adders as opposed to faster adders such as carry look-ahead adder).

(c) Similar to (b), design the circuit that implements the equation above while minimizing the number of t_{FA} delays from the inputs, $x_i[7:0]$, up to the input to the final adder. How many unit delays is needed? You do not need to draw an entire design; showing the appropriate design of the critical path is sufficient. You should assume that each multiplication does not compute a propagate addition but you can use the design from (b).

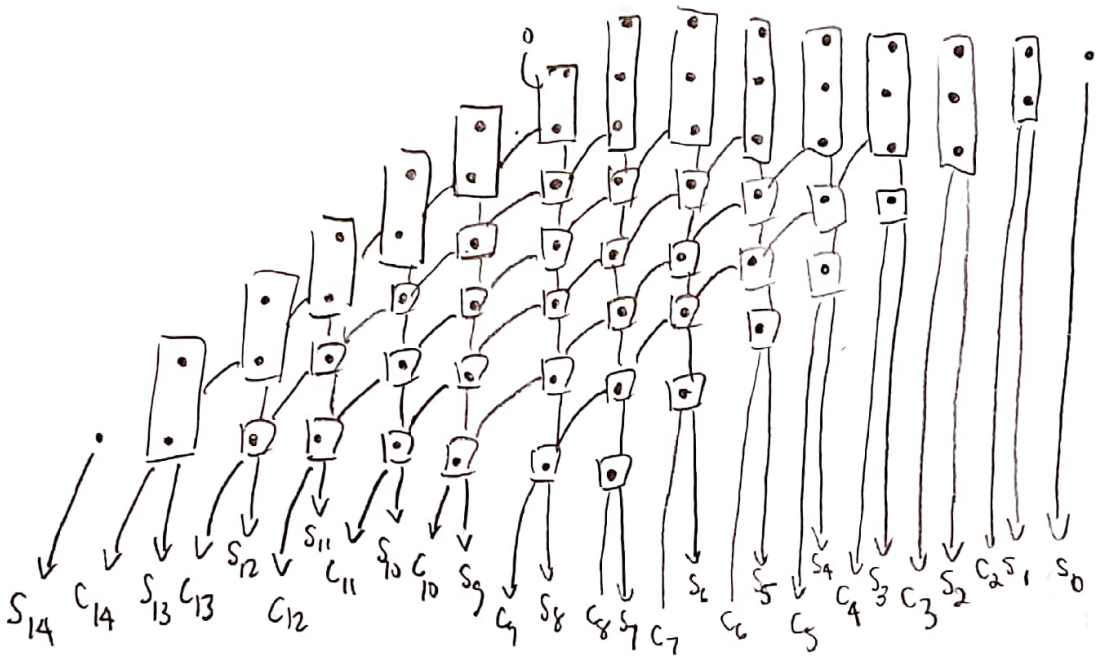
(a) 19 bits are needed for r (The Final output)

18 bits would be the width of the final adder

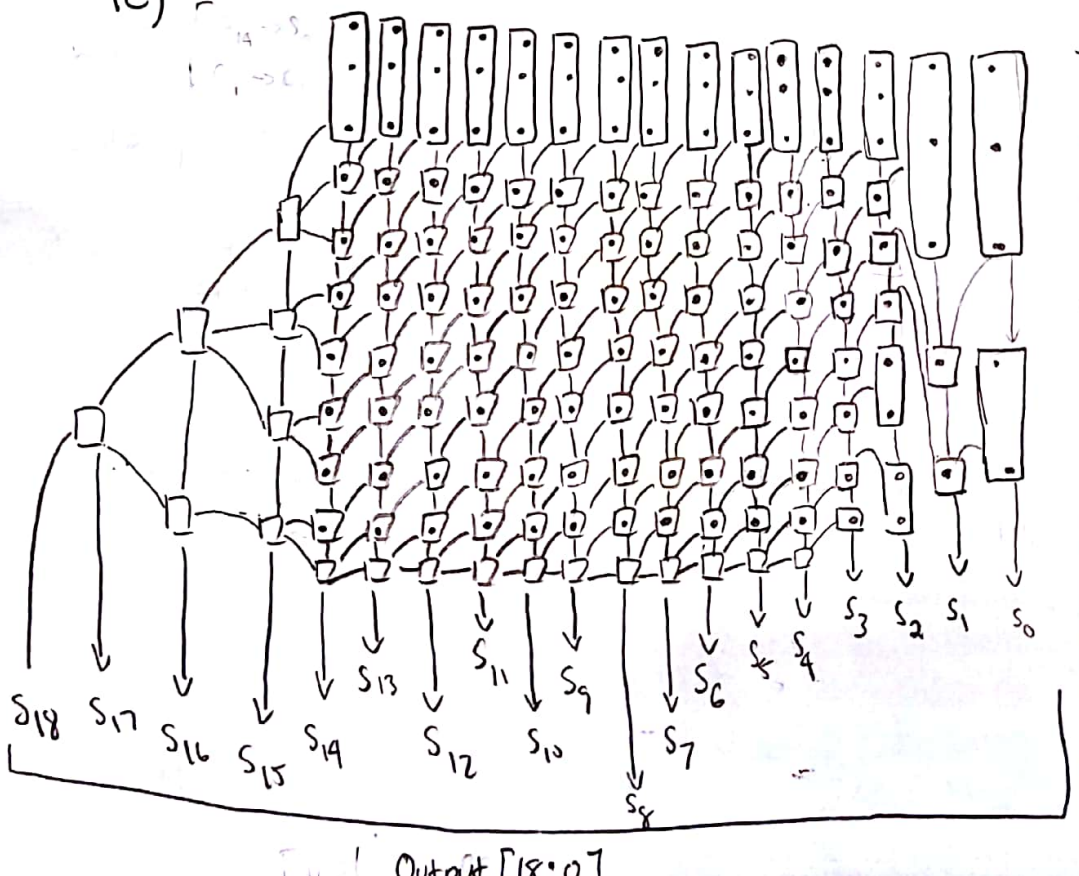
(IMPORTANT: Keep this page in submission even if left unused)

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

1B)



1C)



Output [18:0]

6 unit delays are needed for the multiplication of an 8 bit word with an 8 bit word.

22 unit delays are needed to add 5 16 bit words.

28 unit delays are needed for the multiplication as well as the final addition of 5 16 bit words.

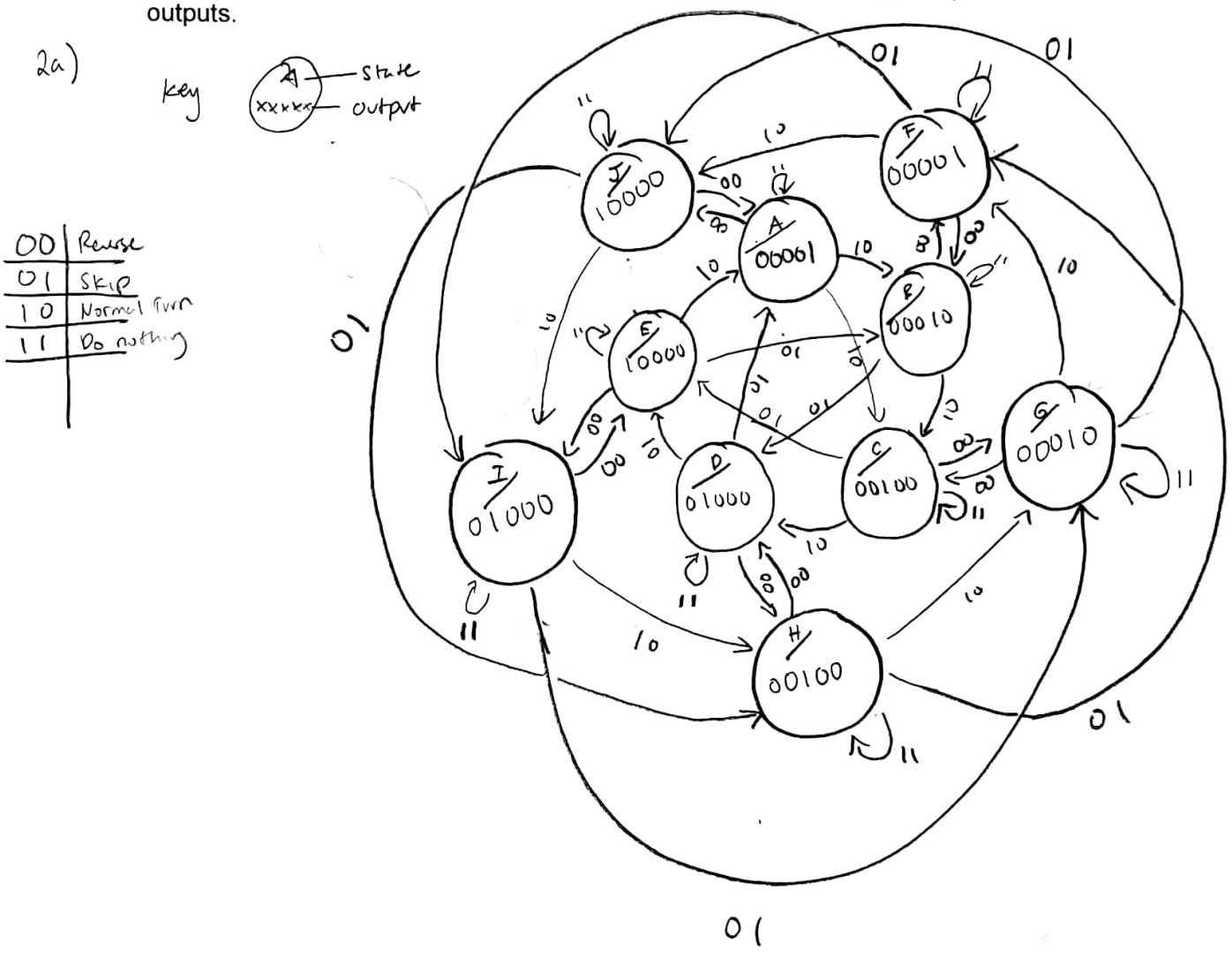
Problem #2

You are to design a **Moore** finite state machine that keeps track of whose turn it is in a game of Uno with 5 players. Signals, *turn[4:0]*, are one-hot encoded outputs that indicate the turn for players 4 to 0 respectively. The inputs are *no_action* (indicating that the respective player is still drawing a card or deciding on what to do}, *normal_turn* (where a normal card including wilds or draw-2 is played), *skip* (where a skip card skips over the next person in the direction of play), or *reverse* (where the reverse card flips the direction of play). The game is initialized to begin with player 0. The default direct of play moves from player 0 to player 1 in increasing order.

(a) Draw a state diagram for the design.

(b) Show the state assignment for each of the states to minimize logic to produce the turn outputs.

2a)



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

2b)

State Assignment

A	000001
B	000010
C	000100
D	001000
E	010000
F	100001
G	100010
H	100100
I	101000
J	110000

TO minimize the logic for state assignment, we used an extra bit to represent all 10 states

Problem #3

For design assignment #3, there are several modules that needs to be implemented. This problem implements two of them in a specific way. There are of course many other ways to implement these functions.

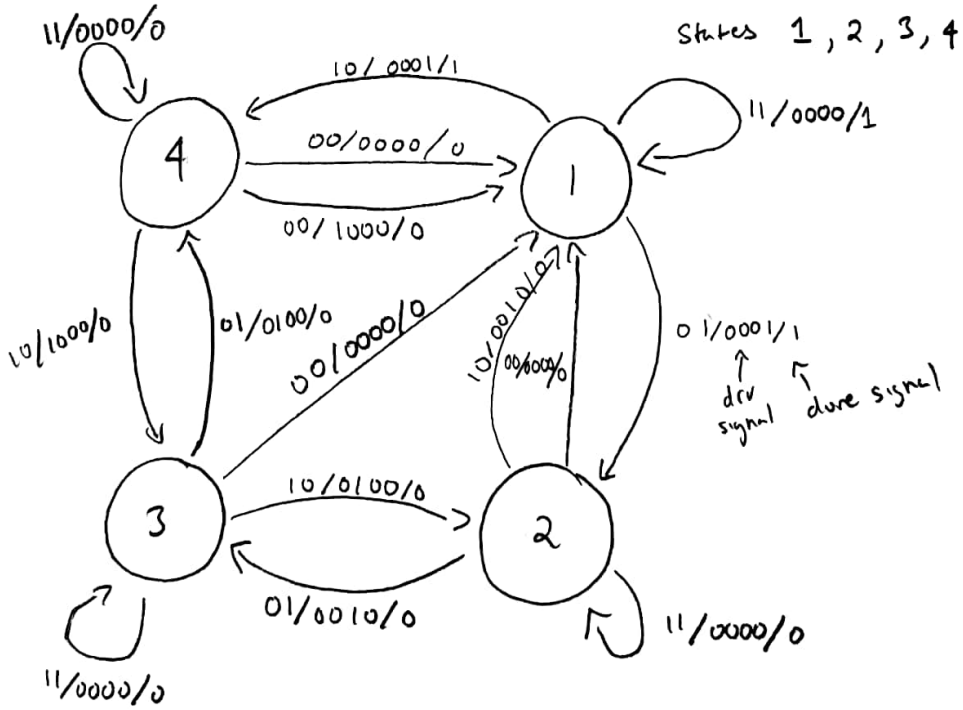
(a) Four signals control the stepper motor, *motor_drv[3:0]*. These four signals are pulsed in a specific sequence in order to move the motor and also to rotate in the desired direction. Note that each pulse results in one step of the motor. Stepping this motor out of order results in improper rotation. Design a **Mealy** finite state machine that controls the sequencing of these outputs. Asserting a *reset* input initializes the motor with *motor_drv[3:0]= 4'b0000*. Inputs, *forward* and *reverse*, indicate the direction of rotation. Asserting *forward* or *reverse* for 1 cycle rotates the motor 1 step in the desired direction by pulsing the appropriate pulse in the right ordering. From the initial state, the first forward or reverse pulse should be *motor_drv[3:0] = 4'b0001*. Note that *forward* and *reverse* can stay asserted for multiple cycles to continuously step for the # of cycles in the duration. If neither *forward* or *reverse* are asserted, the output should not generate a pulse, *motor_drv[3:0] = 4'b0000*. The next input should pulse the appropriate *motor_drv* signal. Forward and reverse cannot both be asserted simultaneously. Each tim e*motor_drv[3:0]=4'b0001*, a *done* signal needs to be asserted to indicate that the motor reached its initial state (this *done* signal helps with sanity checking the motor drive). Draw the FSM state diagram for this state machine.

(b) Every 6 or 7 steps corresponds to one character position. Each step corresponds to a pulse in (a). Every 4th character position corresponds to 7 steps instead of 6. This modulo functionality can be implemented using a simple state machine. A *reset* initializes the state machine assuming that *pos[4:0]=5'b00000*. The FSM takes two 1-bit inputs, *inc* and *dec*, to indicate an increment or decrement of the position respectively. Note that the absolute position is not known except when *reset*. The FSM outputs a signal, *mod3*, to indicate when *pos[4:0]* mod 4 = 3 is reached. Draw the state diagram of a **Moore** FSM to achieve this.

3a)

Input

00	Reset
01	Forward
10	Reverse
11	Else



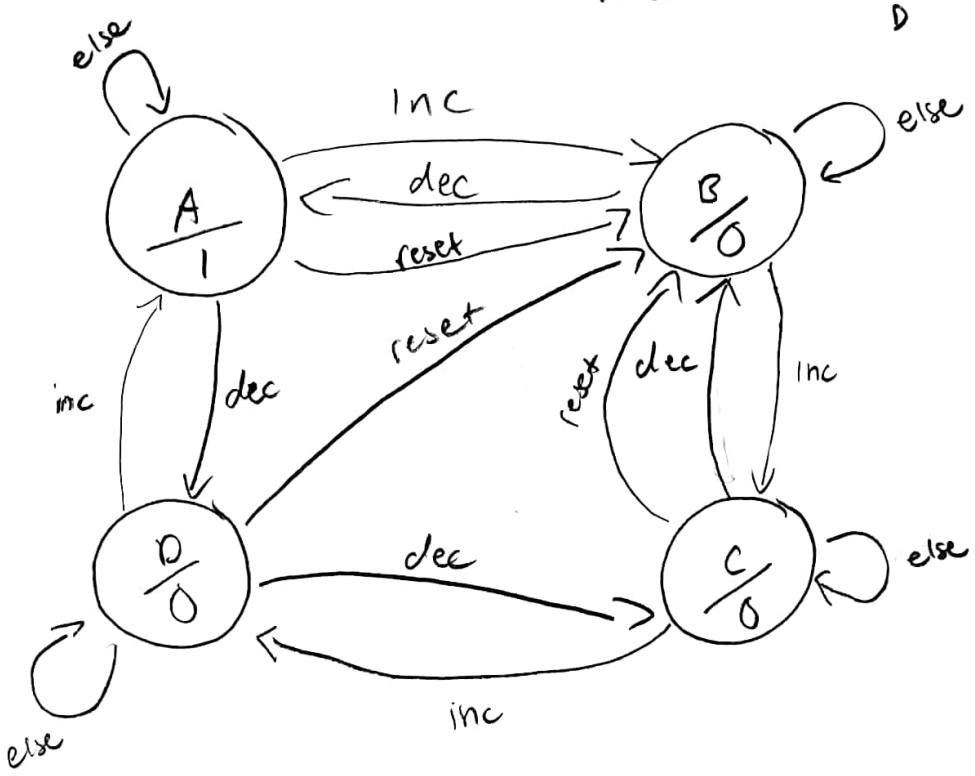
(IMPORTANT: Keep this page in submission even if left unused)

3b)

Inputs

00	Inc
01	dec
10	reset
11	Else

States A B C D



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

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

Problem #4

A state transition table is shown below.

st	n	d	q	acc	nxst
A	1	0	0	0	B
A	0	1	0	0	C
A	0	0	1	0	D
B	1	0	0	0	C
B	0	1	0	0	E
B	0	0	1	0	G
C	1	0	0	0	E
C	0	1	0	0	F
C	0	0	1	1	A
D	1	0	0	0	G
D	0	1	0	1	A
D	0	0	1	1	E
E	1	0	0	0	F
E	0	1	0	0	D
E	0	0	1	1	B
F	1	0	0	0	D
F	0	1	0	0	G
F	0	0	1	1	C
G	1	0	0	1	A
G	0	1	0	1	B
G	0	0	1	1	E

(a) Is this a Mealy or Moore FSM?

(b) Draw the state diagram for this.

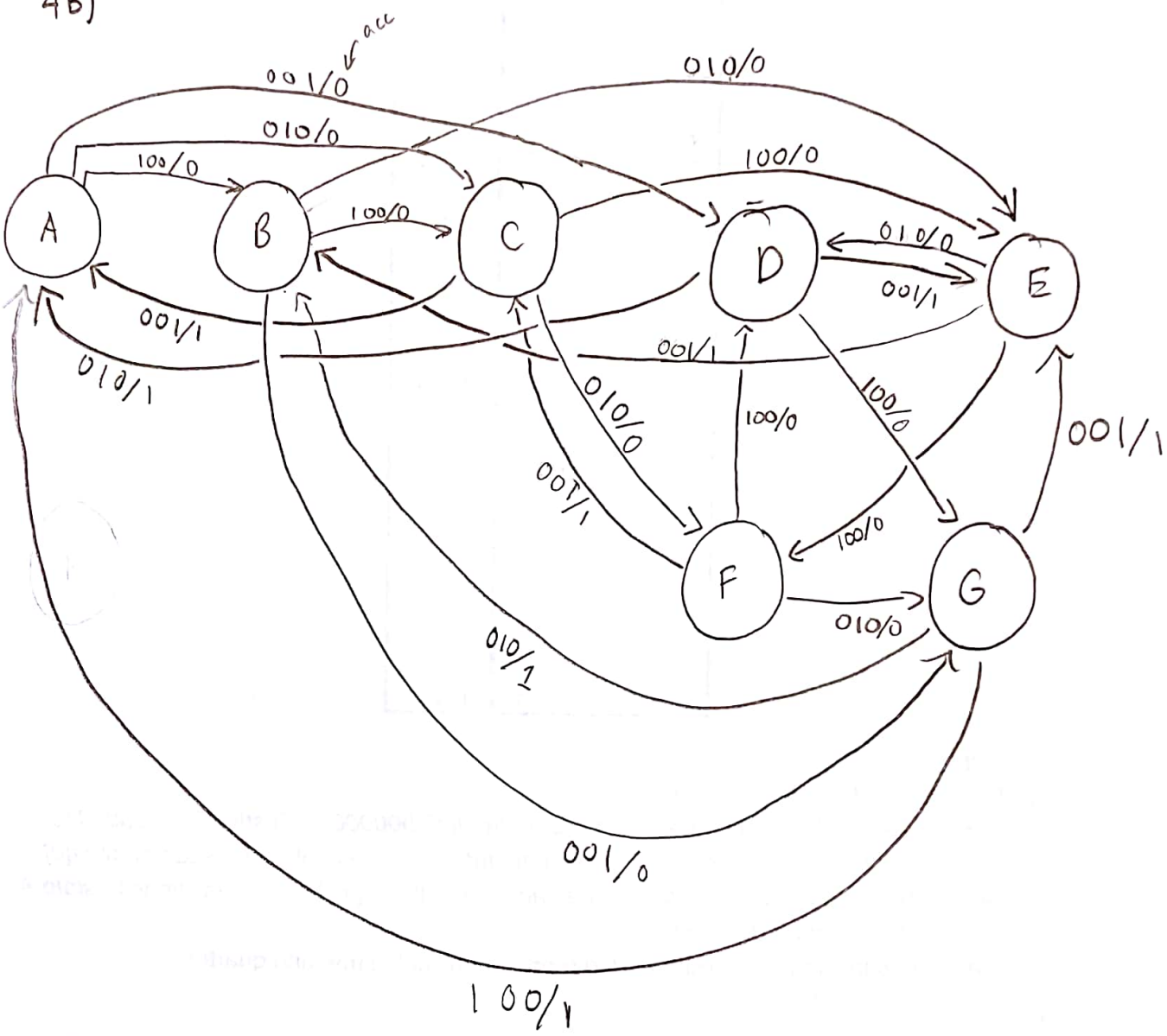
(c) If the states are one-hot encoded where *A:st[6:0]=7'b0000001*, *B:st[6:0]=7'b0000010*, *C:st[6:0]=7'b0000100*, and so on. Note that *state A* is essentially the assertion of *st[0]* etc. Write the logic for the output signal, *acc*. Write the logic for the transitions to *state A* (*nx_st[0]*), and *state B* (*nx_st[1]*).

(d) What does this logic do? Hint: *n, d, q* represent nickel, dime, and quarter.

4a) Mealy FSM

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

4b)



4c) Logic for transition to State A

$(st[2] \wedge \bar{n} \wedge d \wedge q) \vee (st[3] \wedge \bar{n} \wedge d \wedge \bar{q}) \vee (st[6] \wedge n \wedge \bar{d} \wedge \bar{q})$

Logic for transition to state B

$(st[6] \wedge \bar{n} \wedge d \wedge \bar{q}) \vee (st[4] \wedge \bar{n} \wedge \bar{d} \wedge q) \vee (st[0] \wedge n \wedge \bar{d} \wedge \bar{q})$

Logic for acc

$(st[2] \wedge \bar{n} \wedge \bar{d} \wedge q) \vee (\bar{n} \wedge d \wedge \bar{q} \wedge st[3]) \vee (st[3] \wedge \bar{n} \wedge \bar{d} \wedge q) \vee (st[4] \wedge \bar{n} \wedge \bar{d} \wedge q) \vee (st[5] \wedge \bar{n} \wedge \bar{d} \wedge q) \vee (st[6] \wedge n \wedge \bar{d} \wedge \bar{q}) \vee (st[6] \wedge \bar{n} \wedge d \wedge \bar{q}) \vee (st[6] \wedge \bar{n} \wedge d \wedge q)$

4d) The output is a 1 if the total adds up to be greater than 35 ¢ based off previous inputs. Else, the output is a 0.

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

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