Homework #3

(Deadline: 11:59PM PDT, Tuesday, Mar 3, 2020)

Name (Last, First): Arora, Gurbir
Student Id #: 105178554

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 [30 points]

You are to design a **Mealy** finite state machine by **drawing the state diagram** that implements a routing algorithm. The goal is to go from South to North on a conceptually infinite grid via a series of intersections. Assume that upon reset, *rst*, the "car" is on a South-most grid intersection. The "car" accepts 3-bit input and produces 3-bit output. The 3 input bits, *blked[2:0] = {blockLeft, blockStraight, blockRight}* indicates whether the directions to the left, straight, and right (from the perspective of the entry point) is blocked. For instance, {1,1,1} indicates all paths are blocked, {1,0,1} indicates only the straight path is open, etc. The 3 output bits, *heading[2:0] = {goLeft, goStraight, goRight}* indicate which direction to go, left, straight and right respectively, and *heading* is 1-hot encoded. The exception for the one-hot encoding is {0,0,0} which indicates *Wait* at the junction due to blockage. The goal is to move whenever a path exists to the North, East, or West, (so that any cars behind you does not get backed up), and to move North whenever possible (regardless of how much to the left or right the car moves on the grid). So, when North is blocked, you should move to the West if possible and if West is blocked to head East (Note that we prioritize the heading direction to constrain the design). When heading East or West, the "car" should try to return heading North as soon as possible. You should never head South and cannot head backwards. You should Wait until another direction is clear if desired directions are blocked. Build this FSM with as few states as possible and explain your thinking.

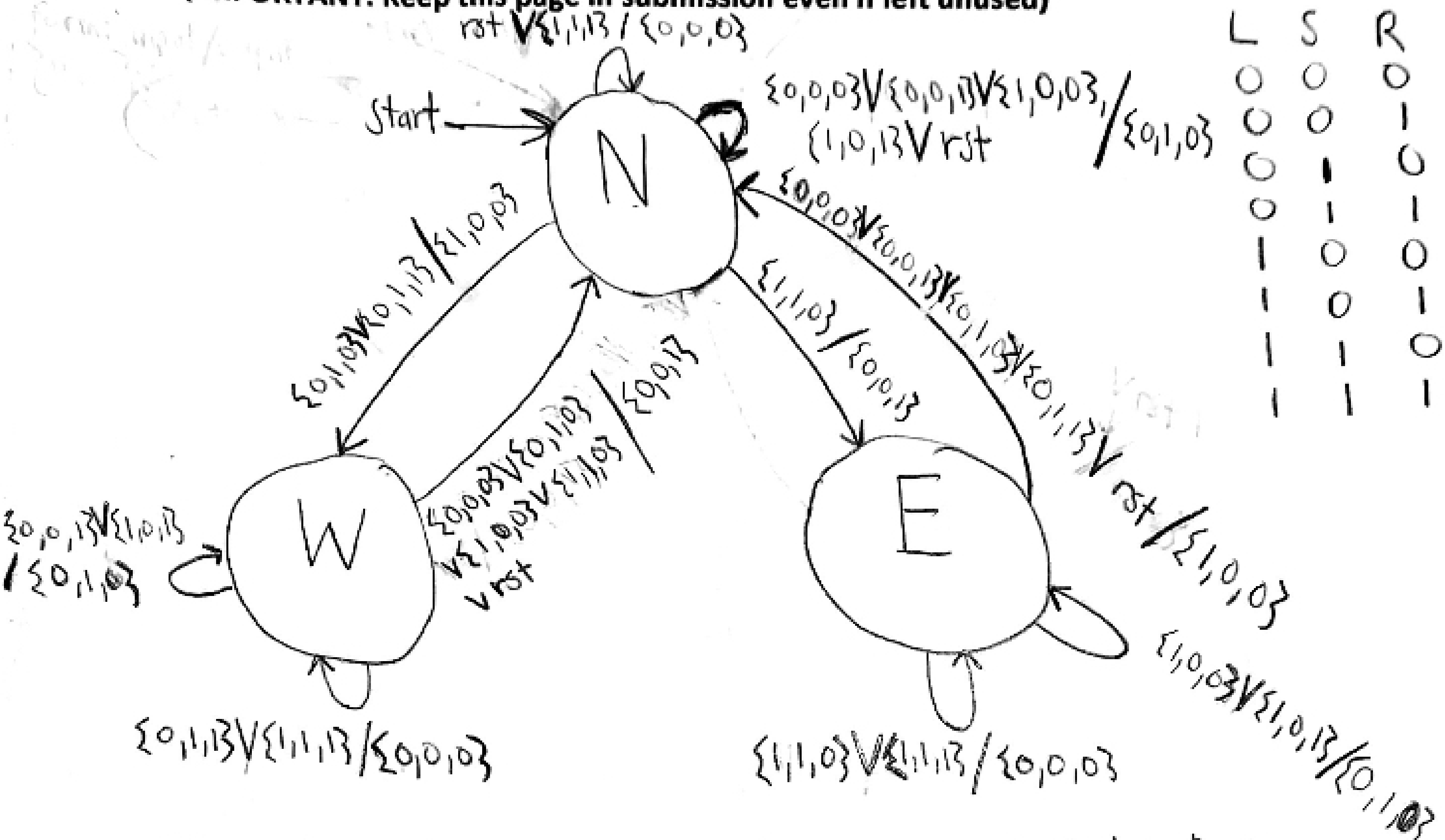
Answer the question in the space below.

- reset (rst) car is on a South-most grid intersection
- 3 bit input blked[2:0] = {bL, bS, bR}
- 3 bit output heading[2:0] = {gL, gS, gR} {0,0,0} = wait
- main goal is North, second West if N blked, 3rd East
- when going E or W, car should try returning to N
- never head S & never backwards
- wait till clear if all dir are blked

My design originally consisted of 3 states N, W, E, but I then realized I may need an initial state depending on what reset is supposed to do. If reset must put the "car" on a South-most grid intersection relative to the previous path, then a 4th state is needed. Upon reflection, I've realized a 4th state is unnecessary, as reset means the car only has to be facing North in order to be "South-most", as this is an infinite grid.

input= blked[2:0] → format: {blk[2:0]$_1$ V {blk[2:0]$_2$, ...} / {heading[2:0]}
output= heading[2:0] inputs / output

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



This design keeps the absolute direction as states & the inputs consist of the relative directions. The priority is to move North, then West, then East and if the only possible movement is South or backwards then the car will wait. If there's no possible movement, obviously the car will wait. So, whenever you're going East and you can go left, do that so you will be facing North. Same goes for when you're facing West and right is open

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

Problem #2 [35 points]

A state transition table is shown below (st: state; nx_st: next_state). The inputs are max, goU and goD; the output is op.

st	max	goU	goD	op	nx_st
ID	X	1	0	0	UPF
ID	X	0	1	0	DNF
ID	X	0	0	0	ID
UPF	0	X	X	1	OPU
UPF	1	X	X	1	OPD
DNF	0	X	X	1	OPD
DNF	1	X	X	1	OPU
OPU	X	0	0	0	ID
OPU	X	1	0	0	UPF
OPU	X	0	1	0	DNF
OPD	X	0	0	0	ID
OPD	X	1	0	0	UPF
OPD	X	0	1	0	DNF

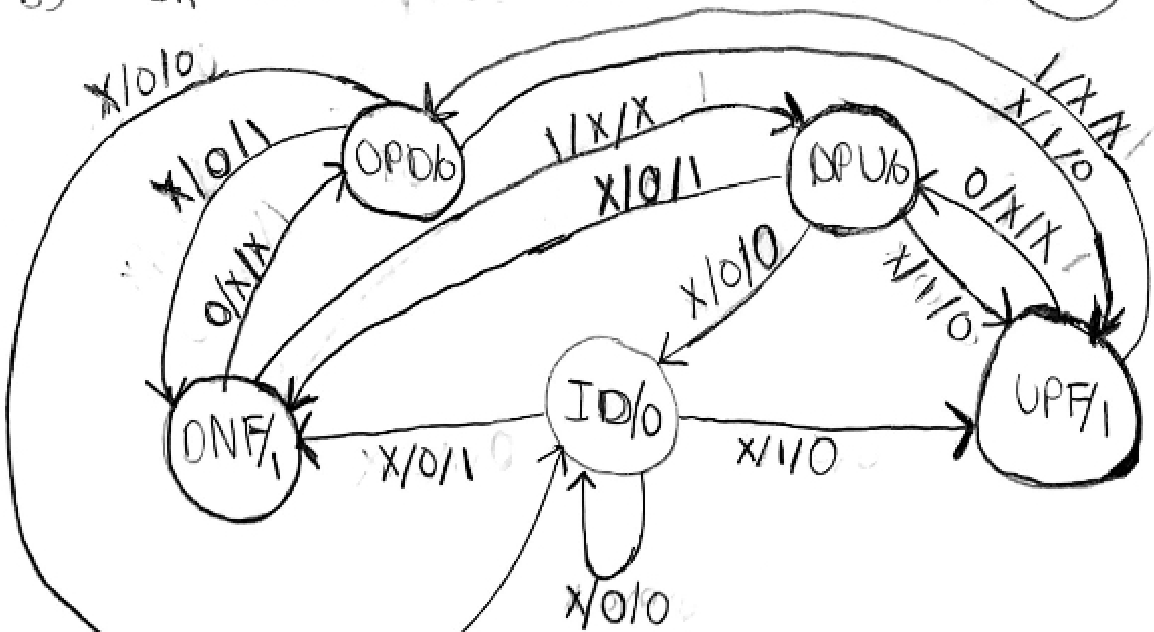
(a) Is this a Mealy or Moore FSM? [5 points] Moore, since only state is needed to get op.

(b) Draw the state diagram for this. [15 points]

(c) We encode states with one-hot encoding where *ID:st[4:0] = 5'b00001*, *UPF:st[4:0] = 5'b00010*, *DNF:st[5:0]=5'b00100*, *OPU:st[5:0]=5'b01000*, and *OPD:st[5:0]=5'b10000*. Note that *state ID* is essentially the assertion of *st[0]*, i.e., state=*ID* iff *st[0]*=1. Write the logic for the output signal, *op*. Write the logic for the transitions to *state ID* (*nx_st[0]*), and *state UPF* (*nx_st[1]*). [15 points]

Answer the question for all parts in the space below.

b) In form: max / goU / goD State in form: state/op



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

c) state=ID iff st[0], =UPF iff st[1], =DNF iff st[2], =OPU iff st[3], =OPD iff st[4]

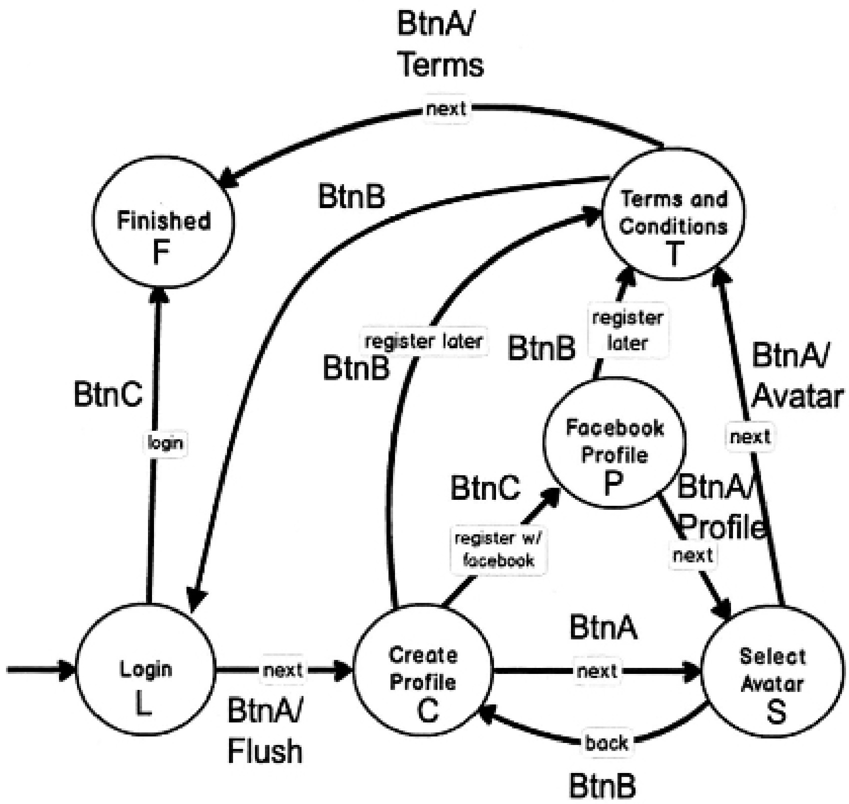
op = st[1] ∨ st[2]

nx_st[0] = (¬goU)∧(¬goD)

nx_st[1] = (goU)∧(¬goD)

Problem #3 [35 points]

State machines come in many forms. Software FSMs are extremely common. We can implement any software FSMs in hardware and vice versa. Below is such an example for a user interface. This interface for a registration page is implemented as a Mealy state machine. For the hardware implementation, we accept 3 button inputs *{BtnA, BtnB, BtnC}* and only one of the buttons are asserted to 1'b1 at one time or all are zeros. For inputs not shown on the arcs, the assumption should be that the state feeds back to itself. There are 4 outputs that are asserted to 1'b1 only on specific arcs *{Terms, Profile, Avatar, Flush}*. On all other arcs, the outputs are zeros.



(a) Draw the State Transition Table for this state machine. You can simplify the table size by noting that only one input can be asserted at any time. [15 points]

(b) If the 6 states are binary encoded into 3 bits, *state[2:0]*, {L,C,S,P,T,F} = {3'b000,3'b001,3'b010,3'b011,3'b100, 3'b101}, determine the logic for the output Profile, and Avatar based on input and/or state. [15 points]

(c) With the same assumption as (b), write the logical expression for the *next_state[0]* as a minimal sum-of-products. You can use symbols (e.g., BtnA, L, C, S). [5 points]

Answer the question for all parts in the space below.

6 states, 3 inputs (one hot)

input format: since only one button can be asserted at a time, stating BtnA implies {1,0,0}, BtnB implies {0,1,0}, BtnC implies {0,0,1}, while 0 implies {0,0,0}

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

a)

State	input	output	next State
Login (L)	BtnA=1'b1	1	C •
Login (L)	BtnB=1'b1	0	L
Login (L)	BtnC=1'b1	0	F •
Login (L)	0	0	L
Create Profile (C)	BtnA=1'b1	0	S
Create Profile (C)	BtnB	0	T
Create Profile (C)	BtnC	0	P •
Create Profile (C)	0	0	C •
Select Avatar (S)	BtnA	1	T
S	BtnB	0	C •
S	BtnC	0	S
S	0	0	S
Terms & Condition (T)	BtnA	1	F •
T	BtnB	0	L
T	BtnC	0	T
T	0	0	T
FaceBook Prof (P)	BtnA	1	S
P	BtnB	0	T
P	BtnC	0	P •
P	0	0	P •

★ Ava

★ Profile

b)

Profile = ¬state[2] ∧ state[1] ∧ state[0] ∧ BtnA

Avatar = ¬state[2] ∧ state[1] ∧ ¬state[0] ∧ BtnA

BtnA = {1,0,0} as input where BtnA = 1'b1, BtnB = 1'b0 BtnC = 1'b0

c) next_state[0] = C(¬BtnA)(¬BtnB)(¬BtnC) + P(¬BtnA)(¬BtnB)(¬BtnC) + L(BtnA) + L(BtnC) + C(BtnC) + S(BtnB) + T(BtnA) + P(BtnC)

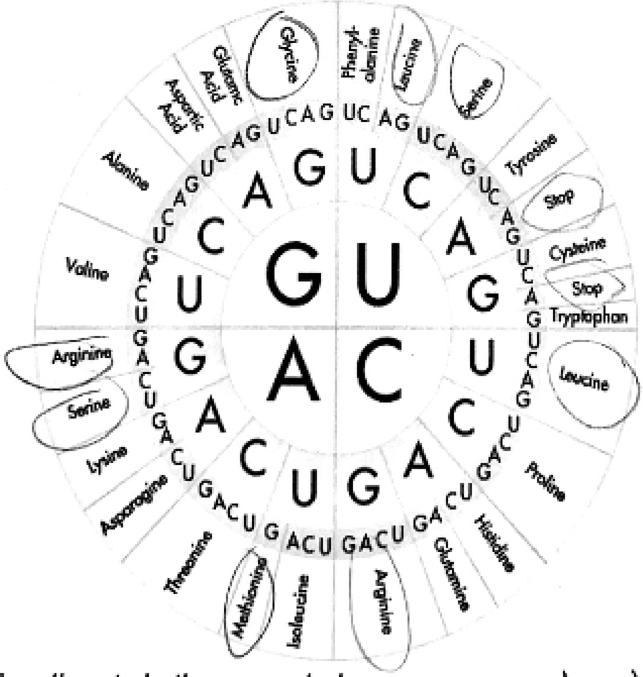
= C(¬BtnA)(¬BtnB) + P(¬BtnA)(¬BtnB) + S(BtnB) + T(BtnA) + L(BtnA) + L(BtnC)

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

[illegible]

[Extra] Problem #4 [+10 points]

Recall the nucleotides in Homework #1 and #2. The translation of a messenger RNA into amino acid is actually not done by looking at 3 nucleotides at once (as we had done in previously) but rather done sequentially where each nucleotide enters a machine. Again, the nucleotides are represented as 2-bit binary digits. G=2'b00, C=2'b11, A=2'b10, and U=2'b01. There are 23 possible amino acids can be identified and are represented as 5-bit binary. A graph of the composition of these amino acids is shown below. There are 2 special amino acids, Methionine and Stop. You are to design a **Moore FSM** by drawing the state transition diagram that can identify a subset of these amino acids. The state machine starts in an initial set of states that is looking for Methionine. After Methionine is detected, it cycles through more states to identify other amino acids. The machine returns to its initial set of states when it detects a Stop codon. Instead of identifying all 20 of the remaining amino acids, we will simplify the design to only detect *Glycine, Leucine, Serine, and Arginine* (note that many of these occur with multiple possible codes). The input to the machine is 2 bits indicating the nucleotide that is arriving every cycle. The outputs are 4 signals, *gly, leu, ser,* and *arg*, that are asserted when the proper sequence is detected. Design this as a Moore state machine.

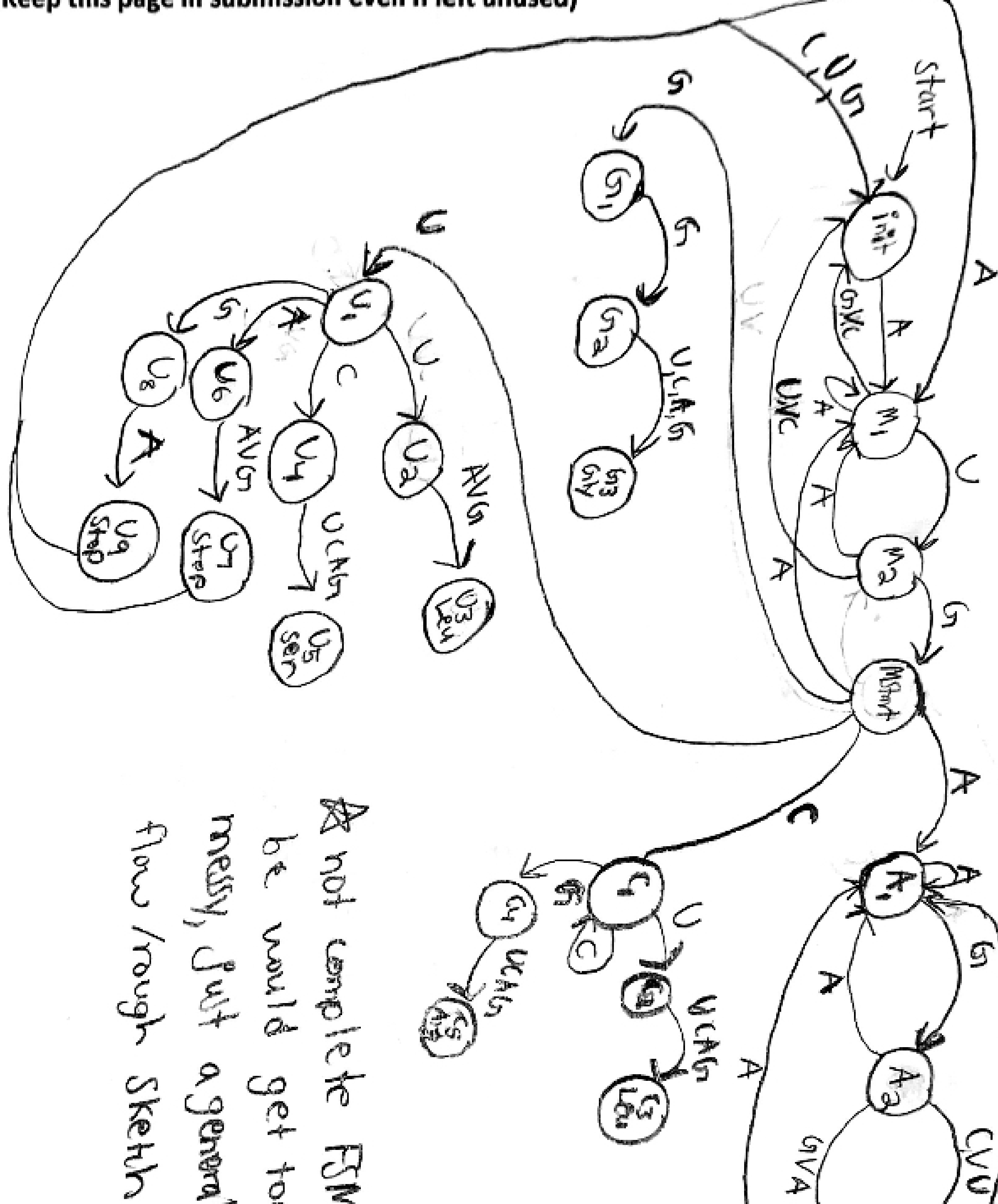


Answer the question for all parts in the space below.

input: 2bit nucleotide
Output: 4 sig: gly, leu, ser, arg

Stop: UAA, UAG, UGA
Methionine: AUG
Glycine: GGU, GGC, GGA, GGG
Leucine: CUU, CUC, CUA, CUG, UUA, UUG
Serine: UCU, UCC, UCA, UCG, AGC, AGU
Arginine: CGU, CGC, CGA, CGG, AGG, AGA

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



☆ not complete FSM
be would get too
messy, just a general
flow/rough sketch

Stop: UAA
UAG
UGA

Start: AUG

Glycine: GGU
GGC
GGA
GGG

Leucine: CUU
CUC
CUA
CUG
UUA
UUG

Serine: UCU
UCC
UCA
UCG
AGC
AGU

Arginine: CGU
CGC
CGA
CGG
AGG
AGA

Input: A= 2'b10, C=2'b11, G=2'b00, U=2'b01

Output: {gly, leu, ser, arg}, not dependent on input since Moore FSM

output not dependent on input only on state

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

State	Input	Output	Next State	State	Input	Output	Next State
Init	A	{0,0,0,0}	M_1	U_1	A	{0,0,0,0}	U_6
Init	C		Init	U_1	C		U_4
Init	G		Init	U_1	G		U_8
Init	U		Init	U_1	U		U_2
M_1	A	{0,0,0,0}	M_1	U_2	A	{0,0,0,0}	U_3
M_1	C		Init	U_2	C		C_1
M_1	G		Init	U_2	G		U_3
M_1	U		M_2	U_2	U		U_1
M_2	A	{0,0,0,0}	M_1	U_3	A	{0,1,0,0}	A_1
M_2	C		Init	U_3	C		C_1
M_2	G		MStart	U_3	G		G_1
M_2	U		Init	U_3	U		U_1
MStart	A	{0,0,0,0}	A_1	U_4	A	{0,0,0,0}	U_5
MStart	C		C_1	U_4	C		U_5
MStart	G		G_1	U_4	G		U_5
MStart	U		U_1	U_4	U		U_5
A_1	A	{0,0,0,0}	A_1	U_5	A	{0,0,1,0}	A_1
A_1	C		C_1	U_5	C		C_1
A_1	G		A_2	U_5	G		G_1
A_1	U		U_1	U_5	U		U_1
A_2	A	{0,0,0,0}	A_4	U_6	A	{0,0,0,0}	U_7
A_2	C		A_3	U_6	C		C_1
A_2	G		A_4	U_6	G		U_7
A_2	U		A_3	U_6	U		U_1
A_3	A	{0,0,1,0}	A_1	U_7	A	{0,0,0,0}	M_1
A_3	C		C_1	U_7	C		Init
A_3	G		G_1	U_7	G		Init
A_3	U		U_1	U_7	U		Init
A_4	A	{0,0,0,1}	A_1	U_8	A	{0,0,0,0}	U_9
A_4	C		C_1	U_8	C		C_1
A_4	G		G_1	U_8	G		G_1
A_4	U		U_1	U_8	U		U_1

State	Input	Output	Next State	State	Input	Output	Next State
U_9 U_9 U_9 U_9	A C G U	{0,0,0,0}	M_1 Init Init Init	C_4 C_4 C_4 C_4	A C G U	{0,0,0,0}	C_5 C_5 C_5 C_5
G_1 G_1 G_1 G_1	A C G U	{0,0,0,0}	A_1 C_1 G_2 U_1	C_5 C_5 C_5 C_5	A C G U	{0,0,0,1}	A_1 C_1 G_1 U_1
G_2 G_2 G_2 G_2	A C G U	{0,0,0,0}	G_3 G_3 G_3 G_3				
G_3 G_3 G_3 G_3	A C G U	{1,0,0,0}	A_1 C_1 G_1 U_1				
C_1 C_1 C_1 C_1	A C G U	{0,0,0,0}	A_1 C_1 C_4 C_2				
C_2 C_2 C_2 C_2	A C G U	{0,0,0,0}	C_3 C_3 C_3 C_3				
C_3 C_3 C_3 C_3	A C G U	{0,1,0,0}	A_1 C_1 G_1 U_1				