Final Design Documentation: Morse Code Decoder

James Plummer
Kevin William

The Morse Code Decoder is a system that is able to accept messages inputted by the user and translate them alphanumerically. In order to create this system, we began by creating finite state machines (FSM). In our initial approach, we made an FSMs to model the conditions of the dot, dash, space between the dash, space between words, and the reset. In our original model, we created six states due to having a space between dots and a space between dashes. We later changed this design by combining these two spaces, as we realized that the program would get stuck in between states while running. After making this modification, we utilized this FSM for our 1st Milestone: to implement a single button for the dot/dash/space/word and use the LED patterns. The second FSM was made to model the conditions of the alphanumerical system, running through each output based on the corresponding input. We then utilized this FSM for our 2nd Milestone: to manually or automatically cycle through all ASCII characters on the seven-segment display of our BASY-3 board. These FSMs are displayed below.

The Morse Code Decoder is a system that is able to accept messages inputted by the user and translate them alphanumerically. In order to create this system, we began by creating finite state machines (FSM). There are many revisions in this stage of our project's process; the initial FSM modeled six states. These states consisted of reset, dot, dash, space between dots, space between dashes, and space between words. This FSM can be seen in the image below.

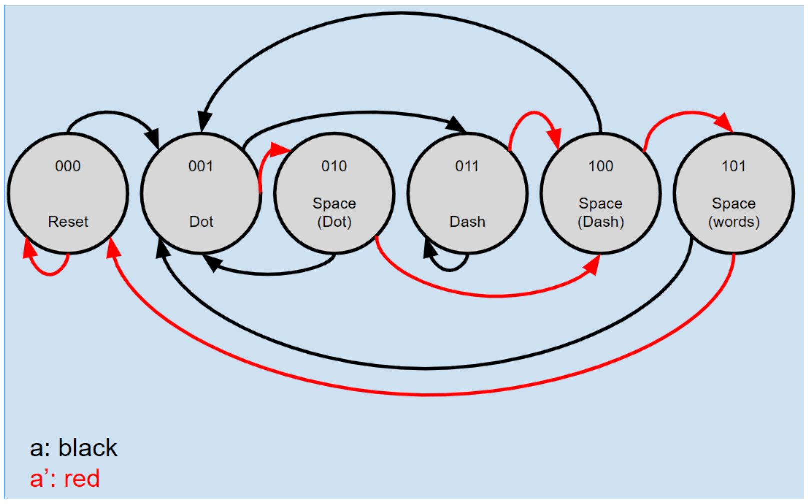

Figure 1

In the next version of our FSM, we modified the six states and simplified them to five. This change was made because we wanted to prevent the program from getting stuck in a loop because of the “space” states- so we combined them. This change can be seen in the image below.

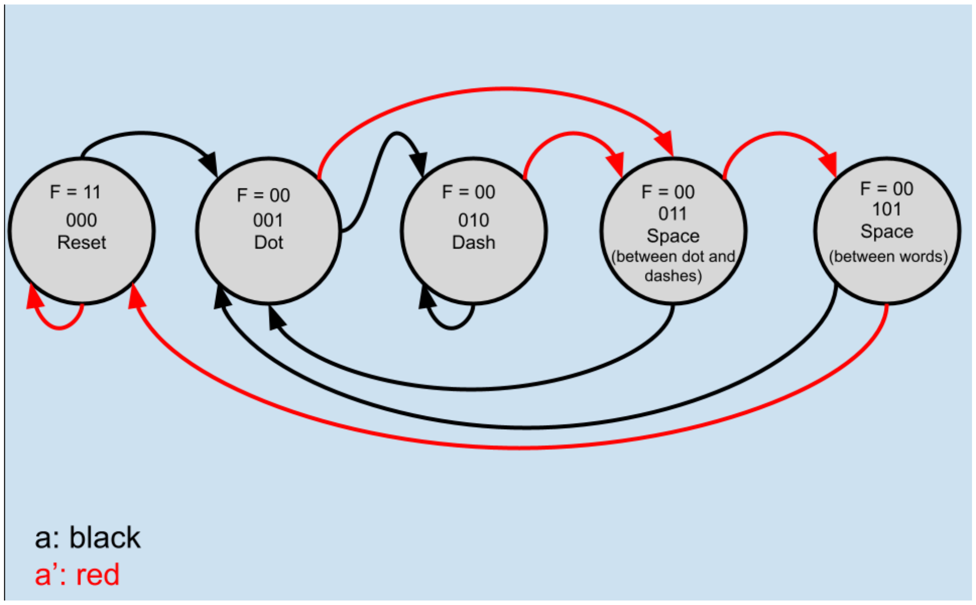

Figure 2

*** In our most recent FSM, we realized that in this model, in order to output a dash, the path would first pass through the dot stage, creating many complications as displayed in Figure 4.

Next, we created a corresponding truth table for each of the FSMs to list all the possible combinations of the inputs. For the first truth table, we listed all combinations to output the F's; F(dot), F(dash), F(space), F(word). In the second truth table, we listed all the combinations to produce all the alphanumerics. In this FSM, if a, then the combination stayed on the same alphanumeric, if a', then the combination moved to the next state; the next alphanumeric. From this table, we then created K-maps to get boolean expressions. These truth tables and their corresponding expressions are displayed below.

a	Q2	Q1	Q0		D2	D3	D0	F(dot)	F(dash)	F(space)	F(word)		D2 = (a')(Q1)(Q0)
0	0	0	0		0	0	0	0	0	0	1		D1 = (Q1')(Q0)+(Q1)(Q0')
0	0	0	1		0	1	1	1	0	0	0		D0 = (a')(Q1')(Q0) + (a')(Q1)(Q0')
0	0	1	0		0	1	1	0	1	0	0		F(dot) = (Q1')(Q0)
0	0	1	1		1	0	0	0	0	1	0		F(dash) =(Q1)(Q0')
0	1	0	0		0	0	0	0	0	0	1		F(space) = (Q1)(Q0)
													F(word) = (Q1')(Q0')
1	0	0	0		0	0	1	0	0	0	1		

1	0	0	1		0	1	0	1	0	0	0	
1	0	1	0		0	1	0	0	1	0	0	
1	0	1	1		0	0	1	0	0	1	0	
1	1	0	0		0	0	1	0	0	0	1	

Figure 3

After going to Lab on Friday, December 9th, 2022 we realized that we had messed up the truth table above, so we need to fix it and see how it affects the rest of the project. We realized that the FSM we were using had an additional output that was not necessary and created more of an issue. Instead, we implemented checks for dots and dashes, by having the user let go of the button before we accept the dot or the dash, this got rid of the issue of the dash only going after a dot.

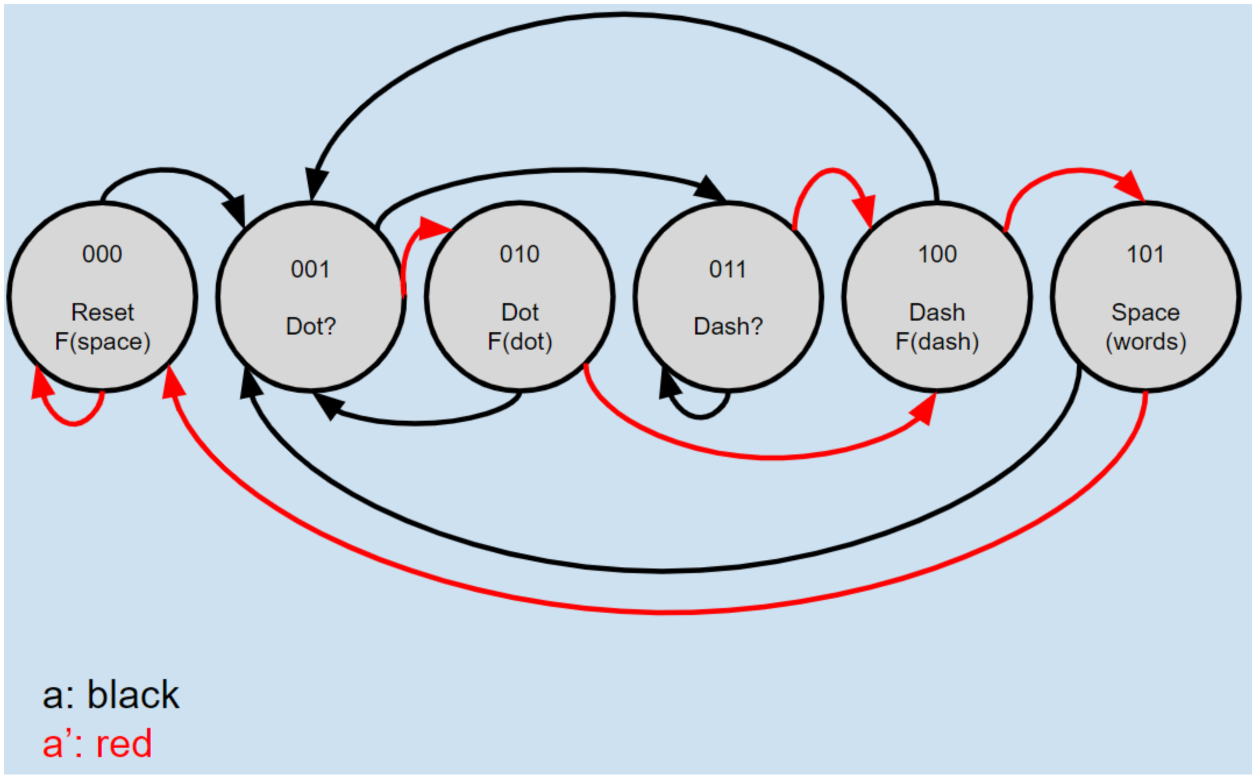

Figure 4: Updated FSM

This is our revised FSM as you can see the outputs we are going to be using have changed to F(space), F(dot), and F(dash), these will be the inputs of the next FSM where we roughly follow this image below:

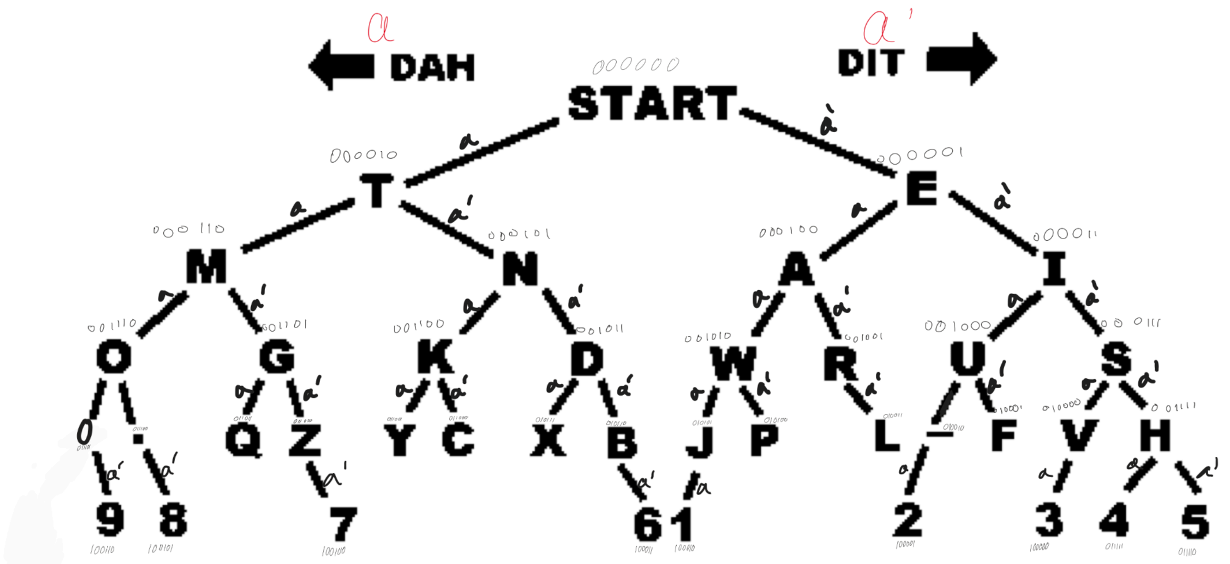

Figure 5: Morse code tree

We are currently working on the Truth Table for this FSM, there are 3 inputs and 6 bits for the registrar, the seven-segment display will not turn on until Fspace has been turned on after that whatever letter the user has entered will be displayed. The letter is built when either dot or dash is one, the register will look at the current value and then change based on the dot or dash.

	Fsp ace	Fda sh	Fdot	Q5	Q4	Q3	Q2	Q1	Q0		D5	D4	D3	D2	D1	D0	**Next**
0	0	0	0	0	0	0	0	0	0	Spa ce	0	0	0	0	0	0	N/a
1	0	0	0	0	0	0	0	0	1	**E**	0	0	0	0	0	1	N/a
2	0	0	0	0	0	0	0	1	0	**T**	0	0	0	0	1	0	N/a
3	0	0	0	0	0	0	0	1	1	**I**	0	0	0	0	1	1	N/a
4	0	0	0	0	0	0	1	0	0	**A**	0	0	0	1	0	0	N/a
5	0	0	0	0	0	0	1	0	1	**N**	0	0	0	1	0	1	N/a
6	0	0	0	0	0	0	1	1	0	**M**	0	0	0	1	1	0	N/a
7	0	0	0	0	0	0	1	1	1	**S**	0	0	0	1	1	1	N/a
8	0	0	0	0	0	1	0	0	0	**U**	0	0	1	0	0	0	N/a
9	0	0	0	0	0	1	0	0	1	**R**	0	0	1	0	0	1	N/a
10	0	0	0	0	0	1	0	1	0	**W**	0	0	1	0	1	0	N/a
11	0	0	0	0	0	1	0	1	1	**D**	0	0	1	0	1	1	N/a
12	0	0	0	0	0	1	1	0	0	**K**	0	0	1	1	0	0	N/a
13	0	0	0	0	0	1	1	0	1	**G**	0	0	1	1	0	1	N/a

14	0	0	0	0	0	1	1	1	0	**O**	0	0	1	1	1	0	N/a
15	0	0	0	0	0	1	1	1	1	**H**	0	0	1	1	1	1	N/a
16	0	0	0	0	1	0	0	0	0	**V**	0	1	0	0	0	0	N/a
17	0	0	0	0	1	0	0	0	1	**F**	0	1	0	0	0	1	N/a
18	0	0	0	0	1	0	0	1	0	**-**	0	1	0	0	1	0	N/a
19	0	0	0	0	1	0	0	1	1	**L**	0	1	0	0	1	1	N/a
20	0	0	0	0	1	0	1	0	0	**P**	0	1	0	1	0	0	N/a
21	0	0	0	0	1	0	1	0	1	**J**	0	1	0	1	0	1	N/a
22	0	0	0	0	1	0	1	1	0	**B**	0	1	0	1	1	0	N/a
23	0	0	0	0	1	0	1	1	1	**X**	0	1	0	1	1	1	N/a
24	0	0	0	0	1	1	0	0	0	**C**	0	1	1	0	0	0	N/a
25	0	0	0	0	1	1	0	0	1	**Y**	0	1	1	0	0	1	N/a
26	0	0	0	0	1	1	0	1	0	**Z**	0	1	1	0	1	0	N/a
27	0	0	0	0	1	1	0	1	1	**Q**	0	1	1	0	1	1	N/a
28	0	0	0	0	1	1	1	0	0	**.**	0	1	1	1	0	0	N/a
29	0	0	0	0	1	1	1	0	1	**0**	0	1	1	1	0	1	N/a
30	0	0	0	0	1	1	1	1	0	**5**	0	1	1	1	1	0	N/a
31	0	0	0	0	1	1	1	1	1	**4**	0	1	1	1	1	1	N/a
32	0	0	0	1	0	0	0	0	0	**3**	1	0	0	0	0	0	N/a
33	0	0	0	1	0	0	0	0	1	**2**	1	0	0	0	0	1	N/a
34	0	0	0	1	0	0	0	1	0	**1**	1	0	0	0	1	0	N/a
35	0	0	0	1	0	0	0	1	1	**6**	1	0	0	0	1	1	N/a
36	0	0	0	1	0	0	1	0	0	**7**	1	0	0	1	0	0	N/a
37	0	0	0	1	0	0	1	0	1	**8**	1	0	0	1	0	1	N/a
38	0	0	0	1	0	0	1	1	0	**9**	1	0	0	1	1	0	N/a
39	0	0	1	0	0	0	0	0	0	**Space**	0	0	0	0	0	1	E
40	0	0	1	0	0	0	0	0	1	**E**	0	0	0	0	1	1	I
41	0	0	1	0	0	0	0	1	0	**T**	0	0	0	1	0	1	N
42	0	0	1	0	0	0	0	1	1	**I**	0	0	0	1	1	1	S
43	0	0	1	0	0	0	1	0	0	**A**	0	0	1	0	0	1	R
44	0	0	1	0	0	0	1	0	1	**N**	0	0	1	0	1	1	D
45	0	0	1	0	0	0	1	1	0	**M**	0	0	1	1	0	1	G
46	0	0	1	0	0	0	1	1	1	**S**	0	0	1	1	1	1	H
47	0	0	1	0	0	1	0	0	0	**U**	0	1	0	0	0	1	F

48	0	0	1	0	0	1	0	0	1	**R**	0	1	0	0	1	1	L
49	0	0	1	0	0	1	0	1	0	**W**	0	1	0	1	0	0	P
50	0	0	1	0	0	1	0	1	1	**D**	0	1	0	1	1	0	B
51	0	0	1	0	0	1	1	0	0	**K**	0	1	1	0	0	0	C
52	0	0	1	0	0	1	1	0	1	**G**	0	1	1	0	1	0	Z
53	0	0	1	0	0	1	1	1	0	**O**	0	1	1	1	0	0	**.**
54	0	0	1	0	0	1	1	1	1	**H**	0	1	1	1	1	0	5
55	0	0	1	0	1	0	0	0	0	**V**	0	1	0	0	0	0	V
56	0	0	1	0	1	0	0	0	1	**F**	0	1	0	0	0	1	F
57	0	0	1	0	1	0	0	1	0	**-**	0	1	0	0	1	0	-
58	0	0	1	0	1	0	0	1	1	**L**	0	1	0	0	1	1	L
59	0	0	1	0	1	0	1	0	0	**P**	0	1	0	1	0	0	P
60	0	0	1	0	1	0	1	0	1	**J**	0	1	0	1	0	1	J
61	0	0	1	0	1	0	1	1	0	**B**	1	0	0	0	1	1	6
62	0	0	1	0	1	0	1	1	1	**X**	0	1	0	1	1	1	X
63	0	0	1	0	1	1	0	0	0	**C**	0	1	1	0	0	0	C
64	0	0	1	0	1	1	0	0	1	**Y**	0	1	1	0	0	1	Y
65	0	0	1	0	1	1	0	1	0	**Z**	1	0	0	1	0	0	7
66	0	0	1	0	1	1	0	1	1	**Q**	0	1	1	0	1	1	Q
67	0	0	1	0	1	1	1	0	0	**.**	1	0	0	1	0	1	8
68	0	0	1	0	1	1	1	0	1	**0**	1	0	0	1	1	0	9
69	0	0	1	0	1	1	1	1	0	**5**	0	1	1	1	1	0	5
70	0	0	1	0	1	1	1	1	1	**4**	0	1	1	1	1	1	4
71	0	0	1	1	0	0	0	0	0	**3**	1	0	0	0	0	0	3
72	0	0	1	1	0	0	0	0	1	**2**	1	0	0	0	0	1	2
73	0	0	1	1	0	0	0	1	0	**1**	1	0	0	0	1	0	1
74	0	0	1	1	0	0	0	1	1	**6**	1	0	0	0	1	1	6
75	0	0	1	1	0	0	1	0	0	**7**	1	0	0	1	0	0	7
76	0	0	1	1	0	0	1	0	1	**8**	1	0	0	1	0	1	8
77	0	0	1	1	0	0	1	1	0	**9**	1	0	0	1	1	0	9

Figure 6: Morse Code Truth Table

Above is a portion of our morse code truth table, this takes three inputs F(dot), F(dash), and F(space), Only one of these inputs can be turned on at any given time so we do not have to worry about when two are turned on at the same time- which was a problem in our initial design. If they are all turned off, the value in the register will stay the same. If F(dot) is turned on, the register will change the value if the corresponding letter has a further letter using a dot. For example, if the register was on "E" and received a dot, the registrar would then move to "I"(Line 40). The same thing happens for F(dash) but goes the other path down the Morse code tree.

We used this truth table to calculate the algebraic formulas of each digit of the seven segment display. These formulas were calculated using a kmap solver from https://www.charlie-coleman.com/experiments/kmap/ The formulas are sent straight to the seven-segment display. When F(space) is off the display will be off, as soon as the Fspace turns on the register will send whatever letter it is on to the display.

Testing

Troubleshooting and testing for our project can get a little complicated but it boils down to this: we are going to have to plug in every value of morse code at least once to make sure that the machine prints the correct letter. After all the letters have been tested, we will test multiple letters back to back, for example: spelling out all of our roommate's names and a bunch of words or numbers to make sure no final letter or number like 7 or Y prints a different letter if another dot or dash has been pressed.

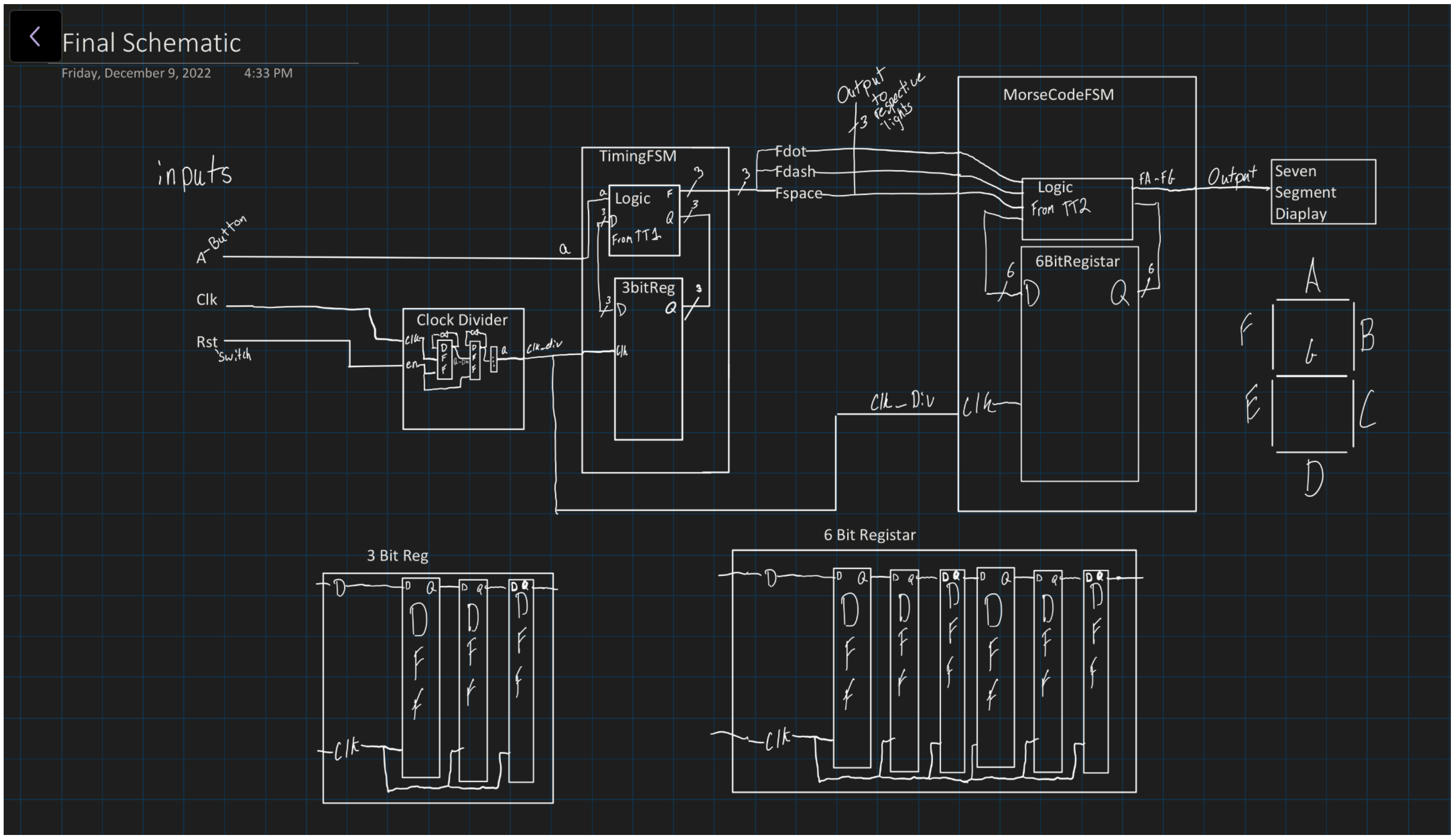

Figure 7