Harsh Patel

RUID: 186000920

December 9, 2019

1011000101100010011000 0 11 000

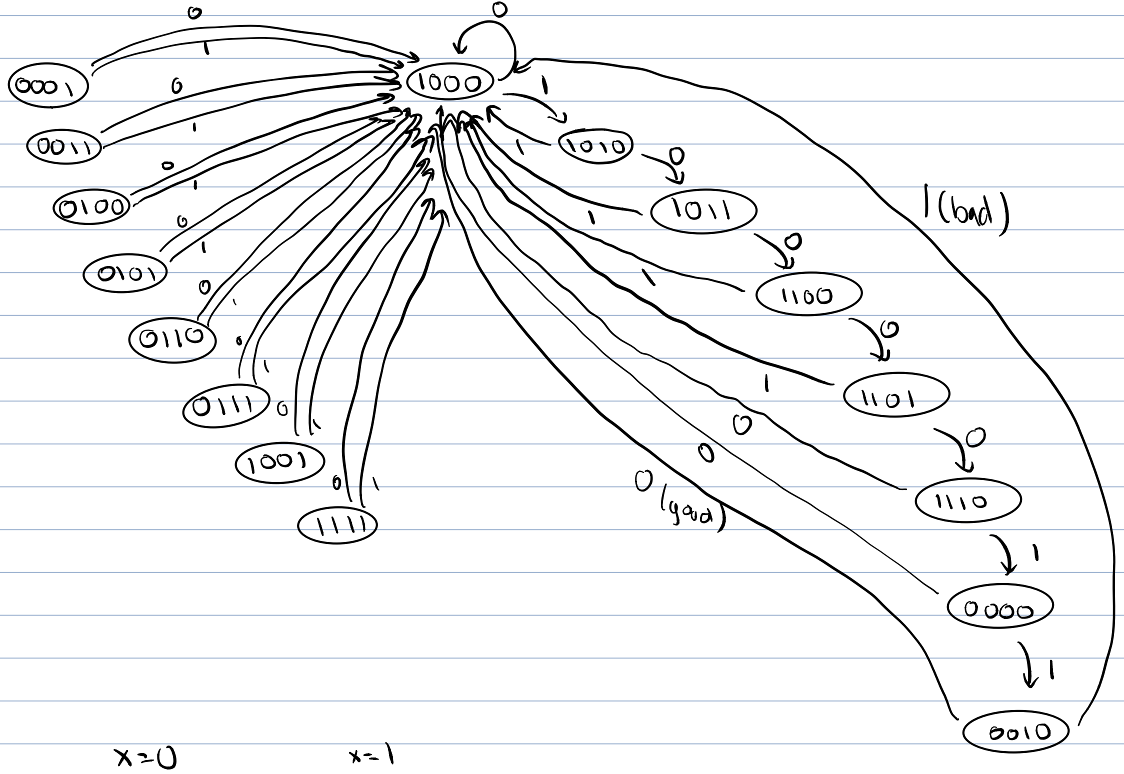
Harsh Patel RUID 186000920 → binary → 1011 00010 1100010011000011000

← least 12 significant bits

A	B	C	D
~~0~~	~~0~~	~~0~~	~~0~~
~~0~~	~~0~~	~~0~~	~~1~~
~~0~~	~~0~~	~~1~~	~~0~~
~~0~~	~~0~~	~~1~~	~~1~~
~~0~~	~~1~~	~~0~~	~~0~~
~~0~~	~~1~~	~~0~~	~~1~~
~~0~~	~~1~~	~~1~~	~~0~~
~~0~~	~~1~~	~~1~~	~~1~~
~~1~~	~~0~~	~~0~~	~~0~~
~~1~~	~~0~~	~~0~~	~~1~~
~~1~~	~~0~~	~~1~~	~~0~~
~~1~~	~~0~~	~~1~~	~~1~~
~~1~~	~~1~~	~~0~~	~~0~~
~~1~~	~~1~~	~~0~~	~~1~~
~~1~~	~~1~~	~~1~~	~~0~~
~~1~~	~~1~~	~~1~~	~~1~~

011000011000

start state



3.

				x=0				x=1			
q_3	q_2	q_1	q_0	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
1	0	0	0	1	0	0	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	0	0
1	0	1	1	1	1	0	0	1	0	0	0
1	1	0	0	1	1	0	1	1	0	0	0
1	1	0	1	1	1	1	0	1	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0
0	0	1	0	1	0	0	0	1	0	0	0
0	0	0	1	1	0	0	0	1	0	0	0
0	0	1	1	1	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0	1	0	0	0
0	1	0	1	1	0	0	0	1	0	0	0
0	1	1	0	1	0	0	0	1	0	0	0
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	0

4.

Q_P	Q_N	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

x = 0

5.

$q_3 q_2 q_1 q_0$	$Q_3 Q_2 Q_1 Q_0$	$J_3 K_3$	$J_2 K_2$	$J_1 K_1$	$J_0 K_0$
1 0 0 0	1 0 0 0	- 0	0 -	0 -	0 -
1 0 1 0	1 0 1 1	- 0	0 -	- 0	1 -
1 0 1 1	1 1 0 0	- 0	1 -	- 1	- 1
1 1 0 0	1 1 0 1	- 0	- 0	0 -	1 -
1 1 0 1	1 1 1 0	- 0	- 0	1 -	- 1
1 1 1 0	1 0 0 0	- 0	- 1	- 1	0 -
0 0 0 0	1 0 0 0	1 -	0 -	0 -	0 -
0 0 1 0	1 0 0 0	1 -	0 -	- 1	0 -
0 0 0 1	1 0 0 0	1 -	0 -	0 -	- 1
0 0 1 1	1 0 0 0	1 -	0 -	- 1	- 1
0 1 0 0	1 0 0 0	1 -	- 1	0 -	0 -
0 1 0 1	1 0 0 0	1 -	- 1	0 -	- 1
0 1 1 0	1 0 0 0	1 -	- 1	- 1	0 -
0 1 1 1	1 0 0 0	1 -	- 1	- 1	- 1
1 0 0 1	1 0 0 0	- 0	0 -	0 -	- 1
1 1 1 1	1 0 0 0	- 0	- 1	- 1	- 1

x = 1

6.

$q_3 q_2 q_1 q_0$	$Q_3 Q_2 Q_1 Q_0$	$J_3 K_3$	$J_2 K_2$	$J_1 K_1$	$J_0 K_0$
1 0 0 0	1 0 1 0	- 0	0 -	1 -	0 -
1 0 1 0	1 0 0 0	- 0	0 -	- 1	0 -
1 0 1 1	1 0 0 0	- 0	0 -	- 1	- 1
1 1 0 0	1 0 0 0	- 0	- 1	0 -	0 -
1 1 0 1	1 0 0 0	- 0	- 1	0 -	- 1
1 1 1 0	0 0 0 0	- 1	- 1	- 1	0 -
0 0 0 0	0 0 1 0	0 -	0 -	1 -	0 -
0 0 1 0	1 0 0 0	1 -	0 -	- 1	0 -
0 0 0 1	1 0 0 0	1 -	0 -	0 -	- 1
0 0 1 1	1 0 0 0	1 -	0 -	- 1	- 1
0 1 0 0	1 0 0 0	1 -	- 1	0 -	0 -
0 1 0 1	1 0 0 0	1 -	- 1	0 -	- 1
0 1 1 0	1 0 0 0	1 -	- 1	- 1	0 -
0 1 1 1	1 0 0 0	1 -	- 1	- 1	- 1
1 0 0 1	1 0 0 0	- 0	0 -	0 -	- 1
1 1 1 1	1 0 0 0	- 0	- 1	- 1	- 1

x = 0

7.

$q_3 q_2 q_1 q_0$	$Q_3 Q_2 Q_1 Q_0$	R	G
1 0 0 0	1 0 0 0	1	0
1 0 1 0	1 0 1 1	0	0
1 0 1 1	1 1 0 0	0	0
1 1 0 0	1 1 0 1	0	0
1 1 0 1	1 1 1 0	0	0
1 1 1 0	1 0 0 0	1	0
0 0 0 0	1 0 0 0	1	0
0 0 1 0	1 0 0 0	0	1
0 0 0 1	1 0 0 0	1	0
0 0 1 1	1 0 0 0	1	0
0 1 0 0	1 0 0 0	1	0
0 1 0 1	1 0 0 0	1	0
0 1 1 0	1 0 0 0	1	0
0 1 1 1	1 0 0 0	1	0
1 0 0 1	1 0 0 0	1	0
1 1 1 1	1 0 0 0	1	0

x = 1

8.

$q_3 q_2 q_1 q_0$	$Q_3 Q_2 Q_1 Q_0$	R	G
1 0 0 0	1 0 1 0	0	0
1 0 1 0	1 0 0 0	1	0
1 0 1 1	1 0 0 0	1	0
1 1 0 0	1 0 0 0	1	0
1 1 0 1	1 0 0 0	1	0
1 1 1 0	0 0 0 0	0	0
0 0 0 0	0 0 1 0	0	0
0 0 1 0	1 0 0 0	1	0
0 0 0 1	1 0 0 0	1	0
0 0 1 1	1 0 0 0	1	0
0 1 0 0	1 0 0 0	1	0
0 1 0 1	1 0 0 0	1	0
0 1 1 0	1 0 0 0	1	0
0 1 1 1	1 0 0 0	1	0
1 0 0 1	1 0 0 0	1	0
1 1 1 1	1 0 0 0	1	0

9.

Red

$q_1q_0 \backslash q_3q_2$	00	01	11	10
00	1	1		1
01	1	1		1
11		1	1	
10		1	1	

x = 0

$q_1q_0 \backslash q_3q_2$	00	01	11	10
00		1	1	
01	1	1	1	1
11	1	1	1	1
10	1	1		1

x = 1

$$= x'q_3'q_1' + x'q_2q_1 + x'q_3q_1' + xq_2q_1'q_0' + xq_0 + xq_3'q_1q_0' + xq_3q_2'q_1q_0'$$

Green

$q_1q_0 \backslash q_3q_2$	00	01	11	10
00				
01				
11				
10	1			

x = 0

$q_1q_0 \backslash q_3q_2$	00	01	11	10
00				
01				
11				
10				

x = 1

$$= x'q_3'q_2'q_1q_0'$$

10.

$J_3 =$

q_1q_0 \ q_3q_2	00	01	11	10
00	1	1	–	–
01	1	1	–	–
11	1	1	–	–
10	1	1	–	–

$x=0$

q_1q_0 \ q_3q_2	00	01	11	10
00	0	1	–	–
01	1	1	–	–
11	1	1	–	–
10	1	1	–	–

$x=1$

$= x' + q_0 + q_1 + q_2$

$K_3 =$

q_1q_0 \ q_3q_2	00	01	11	10
00	–	–	0	0
01	–	–	0	0
11	–	–	0	0
10	–	–	0	0

$x=0$

q_1q_0 \ q_3q_2	00	01	11	10
00	–	–	0	0
01	–	–	0	0
11	–	–	0	0
10	–	–	1	0

$x=1$

$= x q_2 q_1 q_0'$

$J_2 =$

q_1q_0 \ q_3q_2	00	01	11	10
00	0	–	–	0
01	0	–	–	0
11	0	–	–	1
10	0	–	–	0

$x=0$

q_1q_0 \ q_3q_2	00	01	11	10
00	0	–	–	0
01	0	–	–	0
11	0	–	–	0
10	0	–	–	0

$x=1$

$= x' q_3 q_1 q_0$

$K_2 =$

q_1q_0 \ q_3q_2	00	01	11	10
00	–	1	0	–
01	–	1	0	–
11	–	1	1	–
10	–	1	1	–

$x=0$

q_1q_0 \ q_3q_2	00	01	11	10
00	–	1	1	–
01	–	1	1	–
11	–	1	1	–
10	–	1	1	–

$x=1$

$= q_3' + q_1 + x$

$J_1 =$

q_1q_0 \ q_3q_2	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	–	–	–	–
10	–	–	–	–

$x=0$

q_1q_0 \ q_3q_2	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	–	–	–	–
10	–	–	–	–

$x=1$

$= x q_2' q_0' + x' q_3 q_2 q_0$

$K_1 =$

q_1q_0 \ q_3q_2	00	01	11	10
00	–	–	–	–
01	–	–	–	–
11	1	1	1	1
10	1	1	1	0

$x=0$

q_1q_0 \ q_3q_2	00	01	11	10
00	–	–	–	–
01	–	–	–	–
11	1	1	1	1
10	1	1	1	1

$x=1$

$= q_3' + q_0 + q_2 + x$

$J_0 =$

$q_1q_0 \backslash q_3q_2$	00	01	11	10
00	0	0	1	0
01	-	-	-	-
11	-	-	-	-
10	0	1	0	1

x=0

$q_1q_0 \backslash q_3q_2$	00	01	11	10
00	0	0	0	0
01	-	-	-	-
11	-	-	-	-
10	0	0	0	0

x=1

$= X'q_3'q_2q_1 + X'q_3q_2'q_1 + X'q_3q_2q_1'$

$K_0 =$

$q_1q_0 \backslash q_3q_2$	00	01	11	10
00	-	-	-	-
01	1	1	1	1
11	1	1	1	1
10	-	-	-	-

x=0

$q_1q_0 \backslash q_3q_2$	00	01	11	10
00	-	-	-	-
01	1	1	1	1
11	1	1	1	1
10	-	-	-	-

x=1

$= 1$ (all valid)

11)

$J_3 = x' + q_0 + q_1 + q_2$

$K_3 = Xq_2q_1q_0'$

$J_2 = x'q_3q_1q_0$

$K_2 = q_3' + q_1 + X$

$J_1 = Xq_2'q_0' + X'q_3q_2q_0$

$K_1 = q_3' + q_0 + q_2 + X$

$J_0 = X'q_3'q_2q_1 + X'q_3q_2'q_1 + X'q_3q_2q_1'$

$K_0 = 1$ (all valid)

12. Conclusion:

This project was really interesting as it required us to create a combination lock that was unique in the sense that it only opens with our unique RUID. The design was pretty straightforward. Given the RUID decimal to binary conversion, we were meant to use the 16 least significant bits of the binary RUID to open the combination lock. The start state was given by the 4 LSB. Mine happened to be 1000 for my start state. From here, we can look at the right side of the state diagram shown. The right side of the start state shows all the “good” states which will be used in the process to unlock the combination given the correct binary inputs. For example, the first of the 8 binary digits needed to unlock the combination for me was a 1. If there was an input of 1, we know that we can move onto the next good state which would be 1010 (since an input of one skips a state) but if there was an input of 0, you have to rewrap back to the start state. As you can notice, my design bypasses the use of a “lockout state”. What my state machine does instead is that whenever it encounters a bad input, it just starts back again at the start state and therefore not needing a lockout state. The only way for my state machine to succeed and give a green light is if it goes through the entire combination of bits correctly in the correct order. The left side of the start state in my diagram is composed of all of the bad/unused states. Regardless of a 0 or 1 input from these states, it will cause an error which will make it start back at the start state.

The first issue I had with this problem started with understanding how my specific RUID would be incorporated into the state machine but after a couple minutes of reading and rereading the directions, I finally understood that the converted ruid would give me the correct bits to unlock the machine. The state diagram was relatively simple to create so that was not much of an issue. Another trivial difficulty I had while completing this project was simply how tedious it was. After creating the state diagram in step 2, the work left to do was not necessarily “hard” but it was tedious. Having so many different truth tables and Karnaugh maps was time consuming and left a lot of room for error since I noticed it was really easy to mix up bits if you were not paying close attention and concentrating. This was just a matter of really being focused on the task and making sure you are doing the correct thing at the correct time.

One way to make the project better would be to make it more like a real life combination lock in regards to the red and green signals. In real life, locks do not alert the user right away when they make an error in one input. Real combination locks wait until the whole input is done and then give a red or green light as to not give an intruder any idea of which part of the combination is incorrect...making the combination lock effective against people who do not know the correct combination. Our machine flashes a red signal every time an incorrect bit is input which is unrealistic because with trial and error, it would be really easy for an intruder to figure out the correct combination of bits with the red signal outputs.