An Overview of Computer Architecture adventure into the imaginary world of Harry Potter Part 2 - SOLUTIONS!

Anas Shrinah

November 2, 2022

In our last adventure, we successfully unlocked the door of vault 713 in Gringotts Wizarding Bank. In the vault, there is a ruby-red stone called the Philosopher's Stone, fig. 1. A legendary substance with magical powers. It is believed that the Stone can transform any metal into gold. It also produces the Elixir of Life, which will make the drinker immortal! This stone is sought after by Lord Voldemort (The most powerful and dangerous Dark Wizard of all time). Sorry, I meant You-Know-Who. I almost forgot that we are not supposed to say his name.

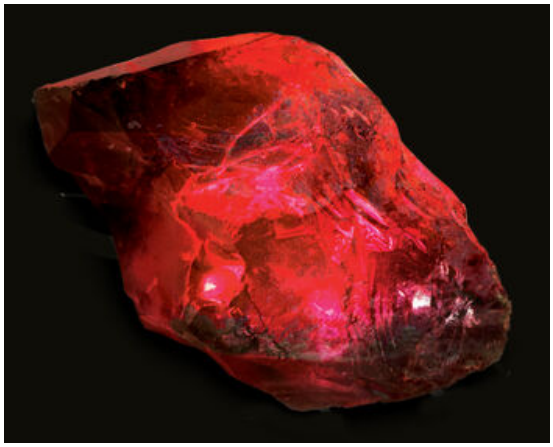

Figure 1: Philosopher's Stone [1].

Our mission today is to protect the Stone by changing the logical circuit of the lock. Last time we solved the puzzle of the new secret code. Of course, we can use any four bits pattern as the new code, but the new secret code from the riddle in vault 713 is magical. It is said that code is extra hard, even might be impossible, for You-Know-Who to crack it. Now it is time to design a circuit that will only unlock the door upon receiving your new secret code.

1 State Transition Diagram

The specification of our lock is simple. The door should be unlocked after receiving the correct input sequence. Every time a wrong input is received, the circuit goes back to the initial state. So it will be ready to receive the next sequence of inputs from the beginning.

The first step is to draw a state transition diagram that captures the specification of the new lock.

1. How many states does our FSM have?
   - We need four states to code the receive the four bits of the secret code 1001.

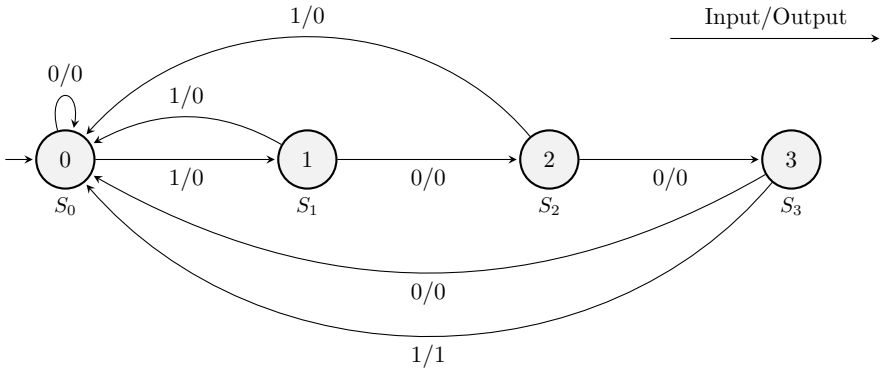

Figure 2: The state transition diagram to implement the new secret code 1001.

2. How many bits do we need to represent all states?
   - We have four states. Hence we need two bits.
3. What does the output represent?
   - The output represents the state of the lock: locked "0" or "1" unlocked.
4. How many outgoing transitions does each state have?
   - Each state has two outgoing transitions. One transition to capture the input of "0" and another transition to capture the input of "1".
5. Why does every state have an equal number of outgoing transitions?
   - Each state has an equal number of outgoing transitions because we can enter "0" or "1" in each state.
6. How many transitions does our FSM have?
   - Our new FSM has two transition for each state, and we have four states. Thus, we have total of eight transitions.
7. How many incoming transitions does the initial state have?
   - Five transitions goes back to the initial state. We need a transition for wrong input from each state to go back to the initial state to restart the process. We also need on transition to go back from the last state to the initial state upon receiving the full correct code.
8. Why does the initial state have at lease an incoming transition from every other state?
   - So we can restart the process of checking the code after receiving wrong input.
9. Why does the last state have no self lopping transition?
   - Because from the last state we always have to go back to the initial state regardless of the input.

With the state transition diagram, now we can easily make the state transition table by recording all transitions.

2 State Transition Table

State transition table is a tabular representation of the state transition diagram. We need it to make the next steps easier. Remember, the rows of state transition table are the transitions of the state transition diagram. The columns of the state transition table are binary codding of the current state, input, next state and the output. How many rows does the new state transition table have?

Good job preparing your state transition table. Now we can find the logical formulas of the next state bits and the output as functions to the current state bits and the input. Use the state transition table to find the Disjunctive Normal Form (DNF) formulas of the output and the next state bits. Next, we will use Karnaugh maps to find optimised formulas.

S1	S0	Input	S1_next	S0_next	Output
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	1

Table 1: The transition table of the FSM of the new secret code.

3 Karnaugh Maps

K-map is a great method to simplify boolean algebra expressions. Use K-map to find simplified formulas for the output and the next state bits. Discuss and answer the following questions with your group.

S1S0/Input	0	1
00	0	0
01	0	0
11	0	1
10	0	0

Table 2: Karnaugh map of the output

S1S0/Input	0	1
00	0	1
01	0	0
11	0	0
10	1	0

Table 3: Karnaugh map of the S0' bit of the next state.

S1S0/Input	0	1
00	0	0
01	1	0
11	0	0
10	1	0

Table 4: Karnaugh map of the S1' bit of the next state.

1. How many K-maps do we need to simplify the formulas of the output and the next state bits?
   - We need one K-map to optimise the logical function of the output and two logical functions for each bit of the next state.
2. How many variables are there in each one of our K-maps?
   - There are three variables in each K-map. Two variables of the current state bits and one variable for the input.
3. Compare the formulas from the K-map and the ones derived directly from the state transition table.
   - The formulas derived directly from the transition table are as follows:
     - $Out = Input \wedge S_0 \wedge S_1$

  - $S_0' = \neg S_1 \wedge \neg S_0 \wedge Input \vee S_1 \wedge \neg S_0 \wedge \neg Input$
  - $S_1' = \neg S_1 \wedge S_0 \wedge \neg Input \vee S_1 \wedge \neg S_0 \wedge \neg Input$
- The formulas derived from K-maps:
  - $Out = Input \wedge S_0 \wedge S_1$
  - $S_0' = \neg S_1 \wedge \neg S_0 \wedge Input \vee S_1 \wedge \neg S_0 \wedge \neg Input$
  - $S_1' = \neg S_1 \wedge S_0 \wedge \neg Input \vee S_1 \wedge \neg S_0 \wedge \neg Input$
- There is no difference between the formulas produced directly from the transition table and the K-maps! The reason the example is so simple that the logic formulas produced are already simple and do not require further simplification.

4 Circuit implementation in Logisim

Well done finding the formulas for the output and the next state bits. As a team, implement these formulas in Logisim. You can use any logic gate with as many inputs as Logisim provides. Download and use the lock circuit template `Lock_FSM_template.circ`.

Discuss with your team at least two input sequences to test if your lock circuit works properly.

Good job! As an extra task, try to reimplement your design using two-input NAND gates only. Re-test your design to ensure the behaviour of your design did not change in the new implementation.

This brings us to the conclusion of our adventure. Hope it was fun and you got a chance to practice your computer architecture skills.

References

[1] Harry Potter Wiki. Philosopher's stone. https://harrypotter.fandom.com/wiki/Philosopher%27s_Stone, 2020. [Online; accessed 23-November-2020].