Digital Design
Combinational Lock

Student Name: Juliet Chinyere Nkwor

Student ID: 2340240

Tutor: Steven Quigley

April 20, 2022

Contents

Table of Contents

Introduction

The aim of the assignment is to design a digital combination lock using VHDL and synchronous design techniques, and to implement the design on the FPGA board. The user interface will be simple and based on use of the push buttons, slider switches, LEDs and 7-segment display on the FPGA board.

There are some possible solutions and techniques applied to address certain hitches and code optimization. Such as debounce, random number generation, finite state machine and combination check.

This report will cover the following parts: Introduction and overview, Design organisation, Simulation results, Synthesis results, and Conclusions

Overview

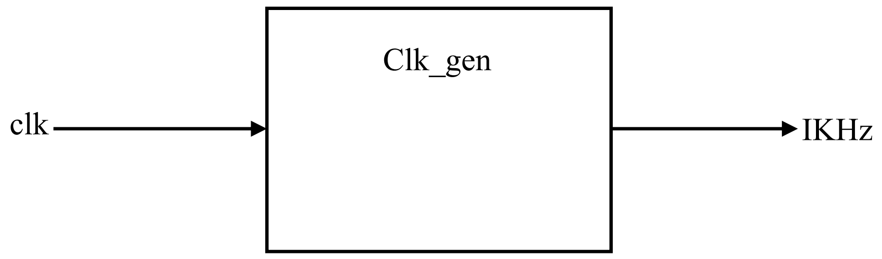
Various components of the project design will be discussed here.

Design Requirement

The Nexys A7 board with part number XC7A50T-1CSG324 produced by Digilent is used. This is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx. With its large, high-capacity FPGA, generous external memories, and collection of USB, Ethernet, and other ports, the Nexys A7 can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, a speaker amplifier, and several I/O devices allow the Nexys A7 to be used for a wide range of designs without needing any other components. The IDE is Vivado from Diligent and the language is VHDL.

Clock

The 100MHz clock generator included in Nexys A7 board can drive all the system. However, the standard speed of the clock is much higher than what the system requires. In this synchronization system design, I derived a 1 kHz signal from a 100 MHz clock. This is achieved by having a 100 MHz counter that will count to 100 thousand (binary 11000011010100000) then reset itself to zero and start all over again. The clock divider is important for multiplexing. During multiplexing, if we use the system clock of the board, which is 100 MHz, to carry out the multiplex, this is too fast compared to the persistence of the LEDs or switches, and all LEDs will look half-on, half-off all the time. The correct frequency to multiplex is about 1 kHz.



State Transition

Sequential logic systems are finite state (FSMs). As FSMs, they consist of a set of states, some inputs, some outputs, and a set of rules for moving from state to state. This FSM has a maximum of eight states: Idle, s0, s1, s2, s3, s4, s5, s6, s7. The system expects two buttons for each project and the value of the button determines what state the system moves to next. The system changes state from Idle to s0 to s1 to s2 to s3 to s4 to s5 to s6 to s7 as long as the input button is high (1). If button is low, and the system state is not changed. Figure 1 is the diagram for the FSM, but first here are a few notes about this diagram:

Debounce Algorithm

Because of the structure and the material of the button, when the button pressed for a moment, it cannot output steady signal, but a jitter of the electrical frequency of the unstable signal. In this project, a simple debouncing circuit is implemented in VHDL to generate only a single pulse when pressing a button on FPGA.

Random Code Generator Method

I used a very high-speed counter that starts counting at the full clock rate of the FPGA as soon as the system is powered up. When the user presses the BTNL button, the counter stops counting. The low order bits of the counter can be regarded as a random number.

The Multiplexer Module

Multiplexer (or Mux) is another word for a selector. It acts much like a railroad switch. This picture shows two possible source tracks that can be connected to a single destination track. The railroad switch controls via some external control which train gets to connect to the destination track. This exact same concept is used with a 2-1 Mux. Two inputs can connect to a single output. This module aims to allow the FSM to display different values on the first four 7-segments. A counter is introduced, which is sets to zero if it is greater than the required 7-

The outputs are the 7-segment LED lights(sseg), and anode (an) is used to enable one of the four 7-segment displays at a time.

Design organisation

Question 2: A Simple Digital Combination Lock

This simple digital combinational lock displays all LEDs for the if the user inputs the correct student ID. Else, the LEDs will be off.

State Diagram

The state diagram shows how the push button causes transitions between states:

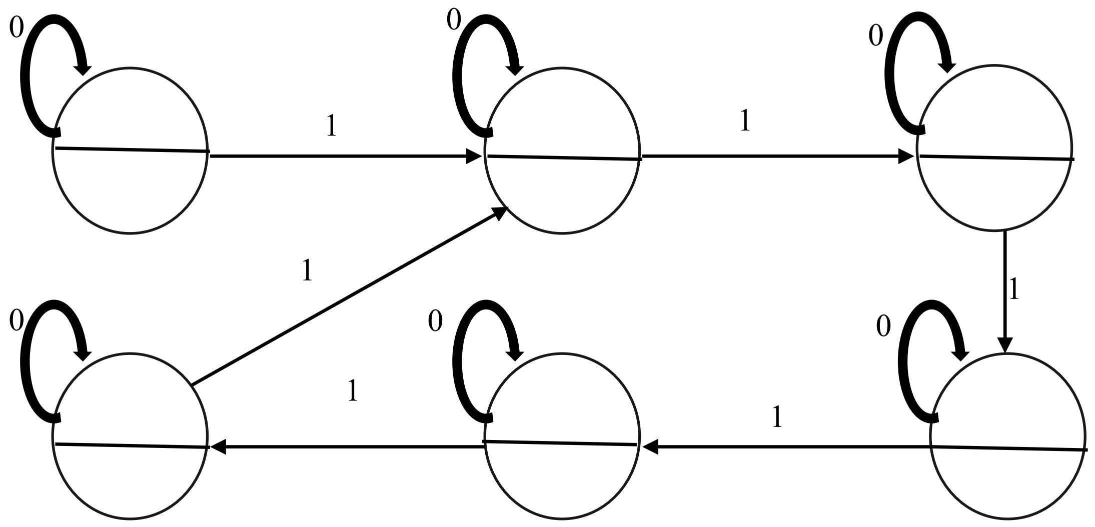


Fig 1

At turn-on or reset, we start in state Idle. If we receive BTNC = 1, we transition to s0 else we stay in state Idle. When in s0, if we receive BTNC = 1, we transition to s1 else we stay in s0. we stay in Idle state. When in s1, if we receive BTNC = 1, we transition to s2 else we stay in s1. When in s2, if we receive BTNC = 1, we transition to s3 else we stay in s2. . When in s3, if we receive BTNC = 1, we transition to s4 else we stay in s3. . When in s4, if we receive BTNC = 1, we transition to s0 else we stay in s4. we stay in Idle state.

State Table

The state table conveys the same information as the state diagram, but in tabular form. For each state that the device could be in, it lists what the next state will be for any possible value of the input.

Present State	Next State	
	BTNC= 1	BTNC = 0
Idle	s0	Idle
	BTNL = 1	BTNL = 0
s0	s1	s0

s1	s2	s1
s2	s3	s2
s3	s4	s3
s4	s0	s4

In this case, I identified five states, which I labelled as Idle, s0, s1, s2, s3. To avoid making bad choices while selecting the memory bits, I will allow the synthesis tool to decide on the state encoding.

Idle – No input on the system
S_0 - A button is light up to indicate that the system is ready.
S_1 – Read first input
S_2 - Read second input
S_3 - Read third input
s4 – Read four input and display outcome

In state S_4, all the LEDs will light up if the sequence matches else LEDs will remain off.

Code Design

There are six key parts of the code:

- Derive a 1 kHz enable signal from a 100 MHz clock by using counter to switch rapidly between the different display digits
- Define the state encoding in the declarations section
- When pressing buttons on FPGA, there are unpredictable bounces that are unwanted. I defined a VHDL code to debounce buttons on FPGA by only generating a single pulse with a period of the input clock when the button on FPGA is pressed, held long enough, and released.
- Use a clocked PROCESS to ensure that transitions happen synchronously
- Use a CASE statement to define the state transitions, as per the state table
- Light up all LEDs if input is correct.

Implementation Procedure

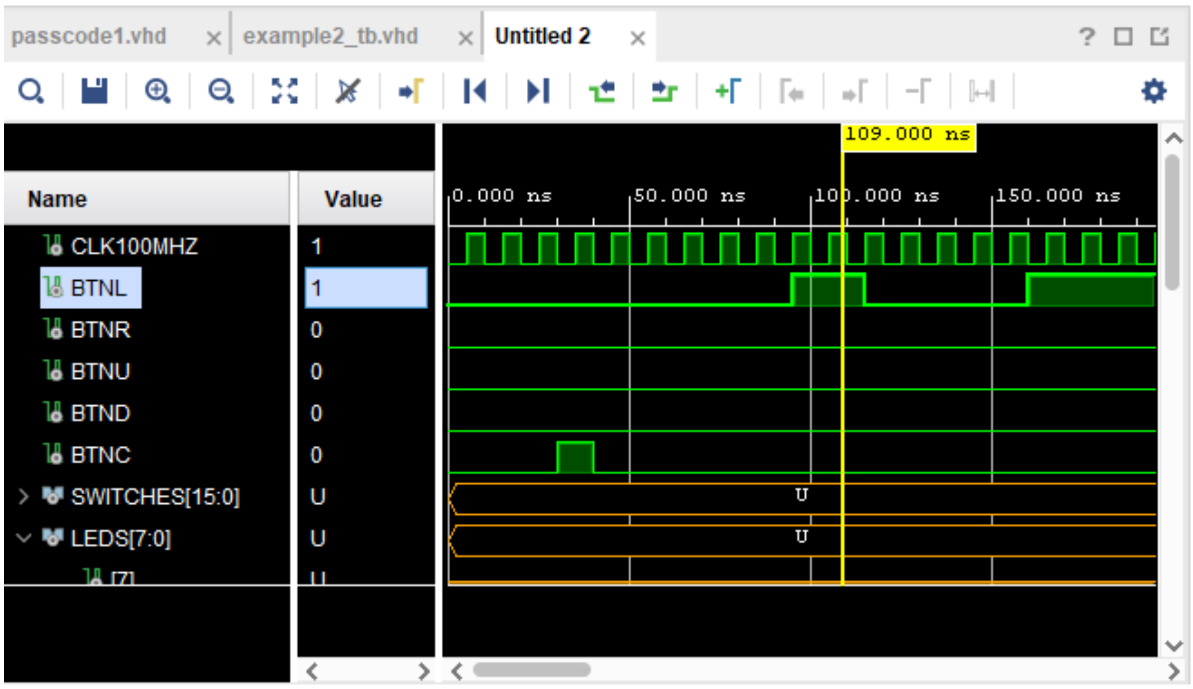
The passcode sequence is 0240.
Two buttons namely BTNC and BTNL are used to operate the system. The BTNC is used to start the system while the BTNL is used to accept or read input.
All the slider switches were used in entering the user input. The inputs are read in the following order.
Input I = SWITCHES(3 downto 0)
Input 2 = SWITCHES(7 downto 4)
Input 3 = SWITCHES(11 downto 8)
Input 4 = SWITCHES(15 downto 12

My user interface operates as follows:

1. The BTNC push button will be used to indicate that the user wants to enter a code sequence. To make my interface user friendly, a single LED is light up in this state.
2. User enters first input using the slider switches. The second button is pressed to read the input

- The user uses the slider switches to enter second, third and fourth input and uses the push button to accept input at every level.
- If the code sequence is correct, all the LEDs will light up else all the LEDs will be off.

Simulation

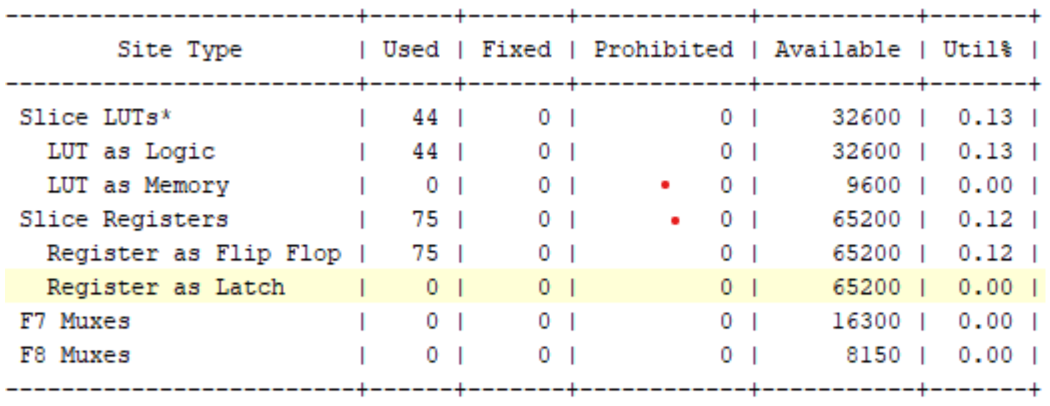


Name	Value
CLK100MHZ	1
BTNL	1
BTNR	0
BTNU	0
BTND	0
BTNC	0
SWITCHES[15:0]	U
LEDS[7:0]	U
[7]	U

Synthesis

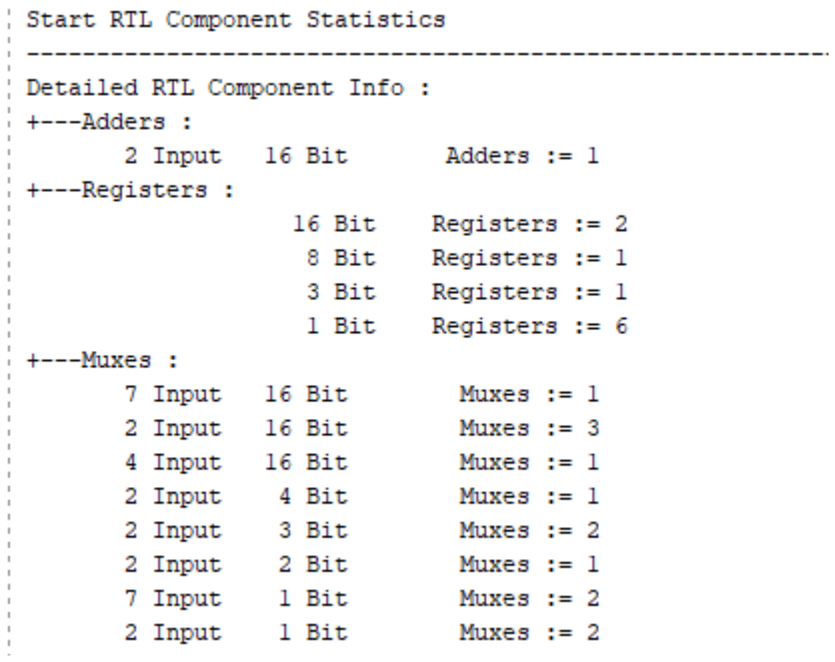
Utilization Report

We can see from this that our design used a tiny percentage of the available resources. I can conclude from this that our design would fit into a much cheaper chip of much lower capacity.



```
-------------------------+------+-------+------------+-----------+-------+
        Site Type        | Used | Fixed | Prohibited | Available | Util% |
-------------------------+------+-------+------------+-----------+-------+
 Slice LUTs*             |   44 |     0 |          0 |     32600 |  0.13 |
   LUT as Logic          |   44 |     0 |          0 |     32600 |  0.13 |
   LUT as Memory         |    0 |     0 |          0 |      9600 |  0.00 |
 Slice Registers         |   75 |     0 |          0 |     65200 |  0.12 |
   Register as Flip Flop |   75 |     0 |          0 |     65200 |  0.12 |
   Register as Latch     |    0 |     0 |          0 |     65200 |  0.00 |
 F7 Muxes                |    0 |     0 |          0 |     16300 |  0.00 |
 F8 Muxes                |    0 |     0 |          0 |      8150 |  0.00 |
-------------------------+------+-------+------------+-----------+-------+
```

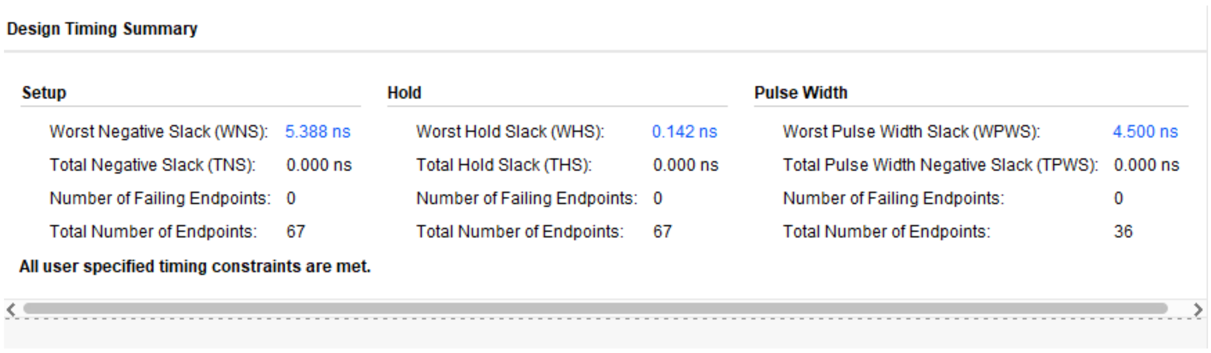
Synthesis Report



```
Start RTL Component Statistics
---------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
	   2 Input   16 Bit       Adders := 1
+---Registers :
	               16 Bit    Registers := 2
	                8 Bit    Registers := 1
	                3 Bit    Registers := 1
	                1 Bit    Registers := 6
+---Muxes :
	   7 Input   16 Bit        Muxes := 1
	   2 Input   16 Bit        Muxes := 3
	   4 Input   16 Bit        Muxes := 1
	   2 Input    4 Bit        Muxes := 1
	   2 Input    3 Bit        Muxes := 2
	   2 Input    2 Bit        Muxes := 1
	   7 Input    1 Bit        Muxes := 2
	   2 Input    1 Bit        Muxes := 2
```

Timing Summary

The important point here is that the number of failures reported is zero and all timing constraints have been met. This means that all combinational parts of our circuit respond within the 10 ns time frame for one clock period and have sufficient margin for the setup and hold times.such lower capacity.


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.388 ns	Worst Hold Slack (WHS):	0.142 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	67	Total Number of Endpoints:	67	Total Number of Endpoints:	36

All user specified timing constraints are met.

Question 3: Improving the user interface

The only difference between this and Question 2 implementation is that the user input, success, and the error message is now visualized to the user using the 7-segment display. The state diagram, state table, code design, the synthesis and simulation table remain the same.

Implementation Procedure

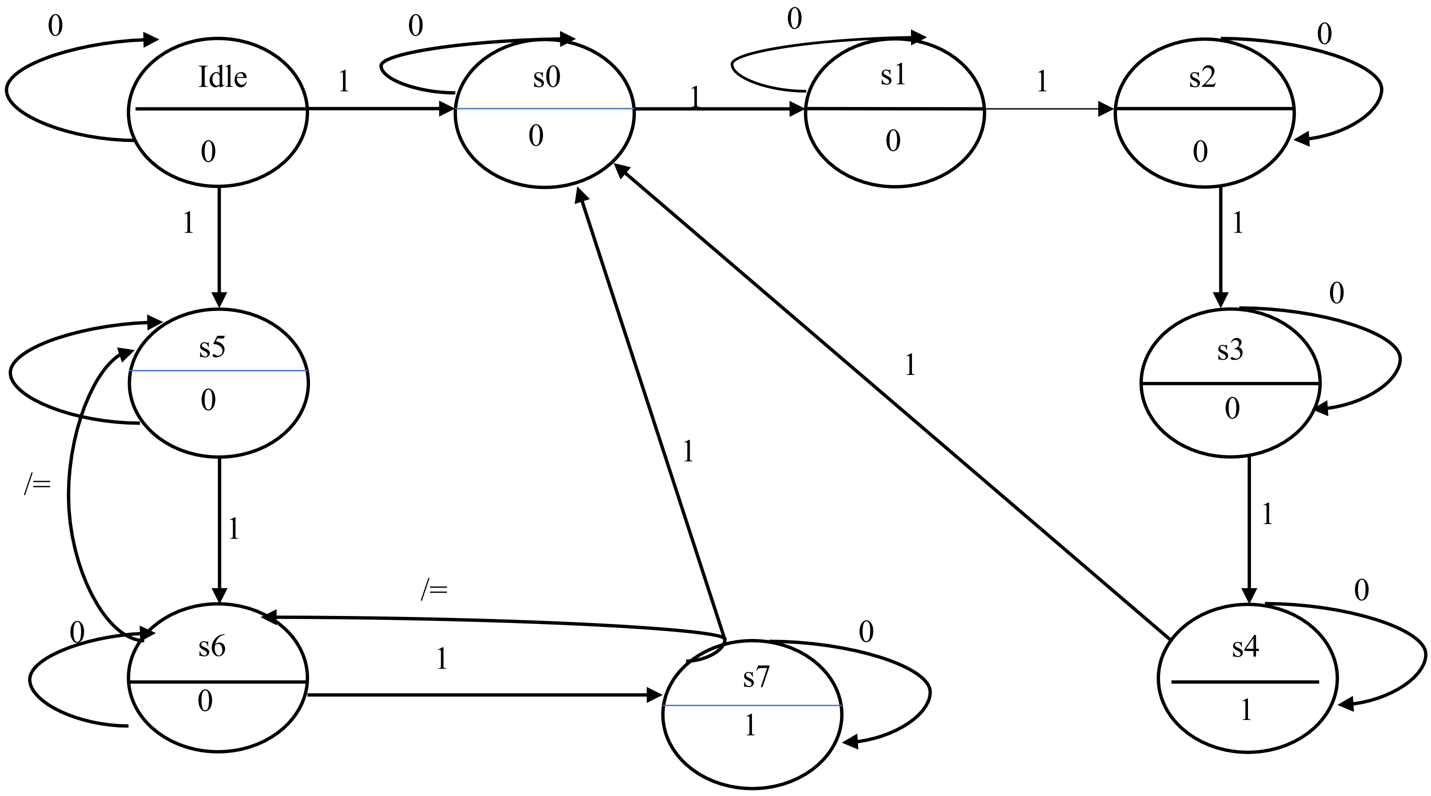
The following improvements was applied to the existing design in Question 2.

- The four digits entered by the user is displayed on the 4-digit 7-segment display.
- After the four digits have been entered, if the code entered is incorrect, the display should indicate to the user that an error has occurred by setting the display to read “Err”. The display of “Err” will alternate at one second intervals with display of the number entered.
- If the code entered is correct, the display will indicate to the user that the code is OK by setting the display to read “OH”.
- The display of “OK” should alternate at one second intervals with display of the number entered.

Question 4: An improved combination lock

In this section, we are expected to modify the existing combination lock design so that the user can change the passcode to a new passcode. We are expected to take appropriate measures to ensure that the user must correctly identify him or herself before a new passcode can be entered. The user can only have access to change their passcode if they can successfully identify their birth date (expressed as a 4-digit number). If the birth date is correctly entered, then the user can reset their passcode.

State Diagram



The implementation for the passcode change is as follows; At turn-on or reset, we start in state Idle. If we receive BTNB = 0, we stay in Idle state. If we receive BTNB = 1, we transition to s5. When we are in s5, if we receive BTNL=1, then we transition to s6. If BTNL = 0 then we stay in state s5. When we are in s6 state, if we receive BTNL=1, if the birth date is correct, we transition to s7, else we go back to s5. When in s7 if we receive BTNL = 1, if the new passcode is not 0, or equal to the old passcode, we transition to s0 to log in. else go back to s6 and try again.
The passcode login implementation remains the same. We start in state Idle. If we receive button = 0, we stay in Idle state. If, on the other hand, we receive BTNC = 1, we transition to s0. When we are in s0 state, if we receive BTNL=1, then we transition to s1. If BTNL = 0 then we stay in state s0. When we are in s1 state, if we receive BTNL=1, then we transition to s2, else we remain in s1. When we get to s2, if BTNL = 1, we transition to s3 else we stay in s2. When in s3 and BTNL = 1, we transition to s4 else we stay in s3. When in s4 and BTNL = 1, we transition to s0 else we stay in s4.

State Table

An additional requirement has been added to combination lock design so that the user can change the passcode to a new passcode. I also introduced some new state transition. For this design, I introduced three more states making the states a total of nine. Which are Idle, s0, s1, s2, s3, s4, s5, s6, s7. To avoid making bad choices while selecting the memory bits, I will allow the synthesis tool to decide on the state encoding.

Idle – No input on the system
s0 – A single button is light up to indicate that the system is ready.
s1 – Read first passcode input
s2 – Read second passcode input
s3 – Read third passcode input
s4 – Read fourth passcode input and display outcome
s5 – A single button is light up to indicate that the system is ready for password change.
s6 – Read birth date, display user input, show Err or OH depending on the outcome
s7 – Read new passcode, display user input, show Err or OH depending on the outcome

In state s7, if button is pressed, will go back to the s0 state, so that the user can log in with the new passcode.

Present State	Next State	
Idle	BTNC= 1	BTNC = 0
	s0	Idle
	BTNU = 1	BTNU = 0
	s5	Idle
	BTNL = 1	BTNL = 0
s0	s1	s0
s1	s2	s1
s2	s3	s2
s3	s4	s3
s4	s0	s4
s5	s6	s5
s6	s7	s6
s7	s0	s7

Implementation Procedure

The passcode sequence is 0240 and the birth date sequence is 1967.
BTNU button is introduced in the design to indicate that the user wishes to change their passcode. The combinational lock design from question 2 and 3 remains the same. If the user presses this BTNU button, the system will prompt them to enter their 4 digit birth date. The user can only have access to change their passcode if the birthday sequence is correct.

My user interface operates as follows:

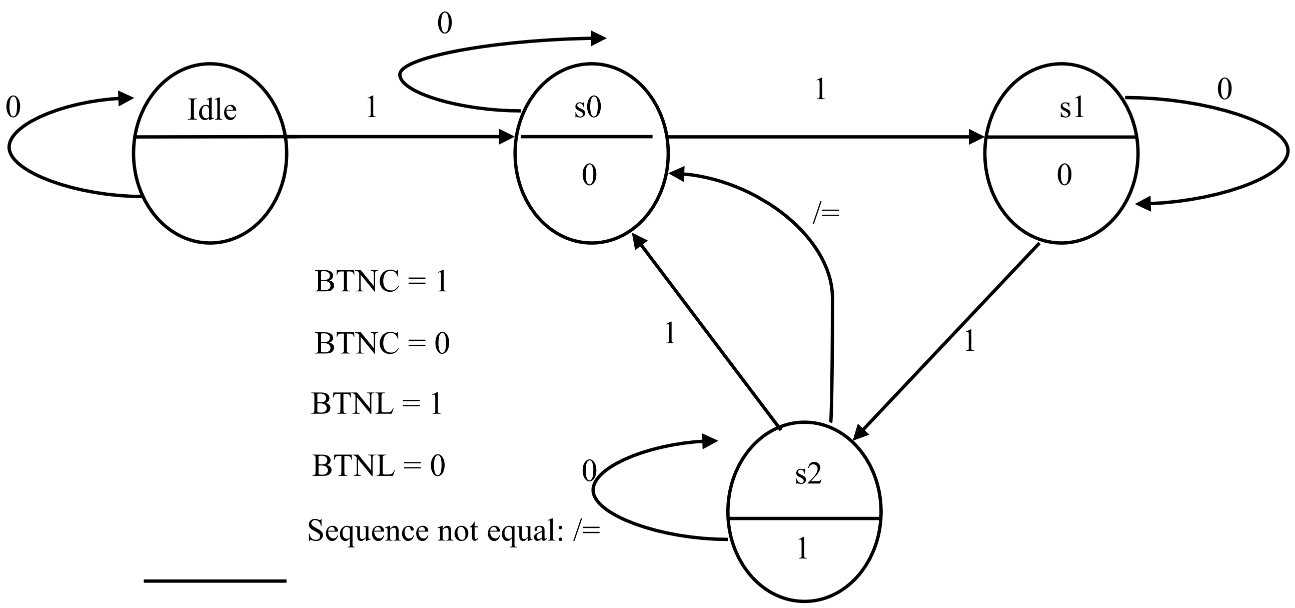
1. The user presses BTNU to indicate that the user wishes to change their passcode.
2. The user uses the slider switches to enter the four-digit birth date. Press the BTNL to read the input.

3. The inputted value is displayed on the 7-segment display. The system delays for about 2 seconds and display the error or success message depending on the outcome.
4. If the birth date sequence is correct, step 2 and 3 is repeated but this time, it is the new passcode.
5. Checks are applied to ensure that the new passcode sequence is correct and valid.
6. If the new passcode sequence is valid, the user is then allowed to login with their new passcode.

Question 5: Further improvements

To further improve the secure system, we are expected to modify the existing combination lock design so that it will request the user to type in two of the digits in the code sequence. The system will prompt the user to show the position of the two numbers that should be entered. Each time the user attempts to gain access, the system will make a different choice of which of the numbers in the code sequence must be entered. If the user successfully identifies the values in both position, access will be granted.

State Diagram



At turn-on or reset, we start in state Idle. At this stage, the random counter is already running. If we receive BTNR = 0, we stay in Idle state. If, on the other hand, we receive BTNB = 1, we stop the counter and transition to s0. When we are in s0, if we receive BTNL=1, then we transition to s1. If BTNL = 0 then we stay in state s0. When we are in s1 state, if we receive BTNL=1, then we transition to s2, else we remain in s1. When we get to s2, if BTNL = 1, we transition back to the s0 state else we remain in the s2 state.

State Table

With this recent code improvement, I now have only four states: Idle, s0, s1, s2. To avoid making bad choices while selecting the memory bits, I will allow the synthesis tool to decide on the state encoding.

Idle – No input on the system
s0 – A single LED is light up to indicate that the system is ready to accept code sequence. Display the first random number.
s1 – Read first user input. Display the second random number to the user
s2 – Read second user input, If sequence matches, give access and display result

Present State	Next State	
	BTNC= 1	BTNC = 0
Idle	Ready	Idle
	BTNL = 1	BTNL = 0
s0	s1	s0
s1	s2	s1
s2	s0	s2

Implementation Procedure

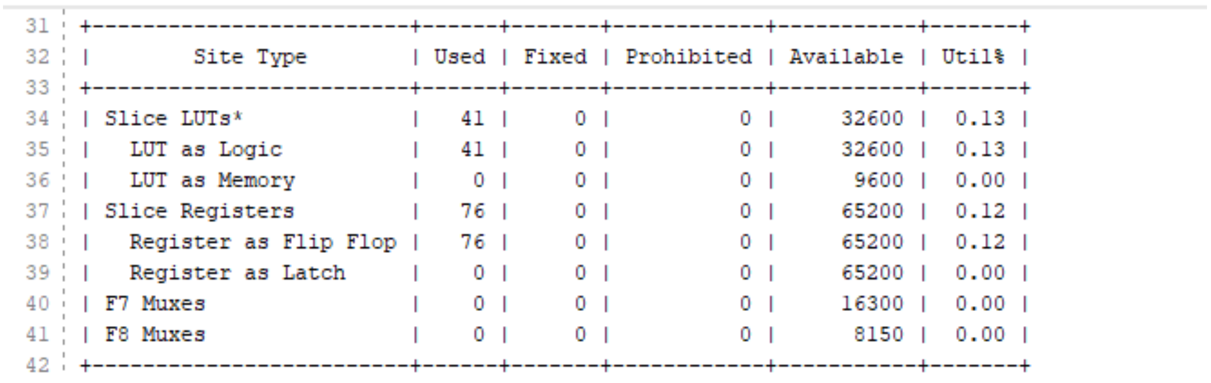
BTNR button is introduced in this design used to indicate that the user wishes to be prompted to enter two numbers from the code sequence. At the rising edge clock, a random number is generated using the method described above. As usual, the BTNL button will be used to read input and the user can only gain access to the system if they can identify the values in both position in the code sequence

The user interface should operate as follows:

1. The BTNR is introduced to indicate that the user wishes to be prompted to enter two numbers from the code sequence.
2. The system will choose a random number in the range 0 to 4 and display it to the user. The user enters the corresponding number from the code sequence using the slider switches and the second push button as normal.
3. The system will then choose a different random number in the range 0 to 4 and display it to the user. The user enters the corresponding number from the code sequence.
4. If the two digits are correct, then the system unlocks

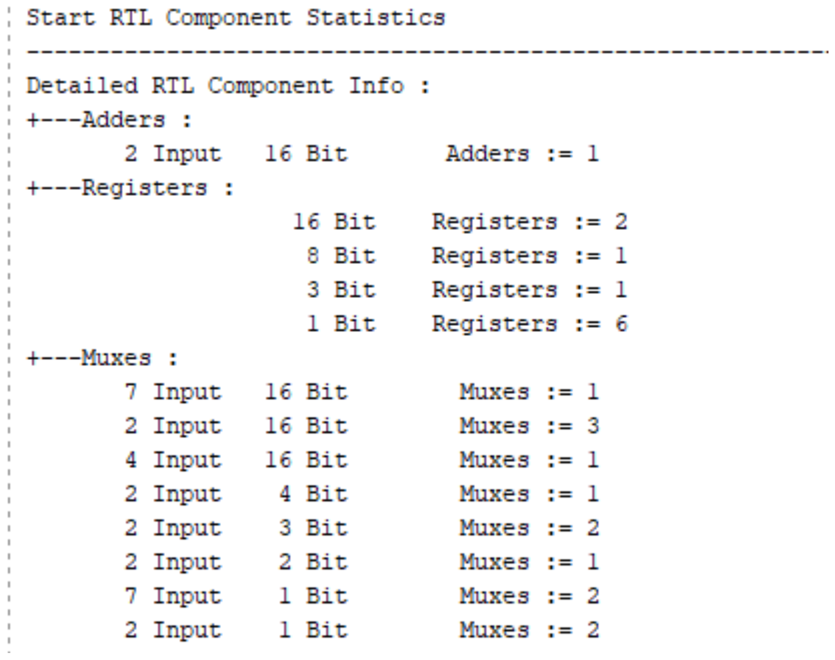
Synthesis

Utilization Report



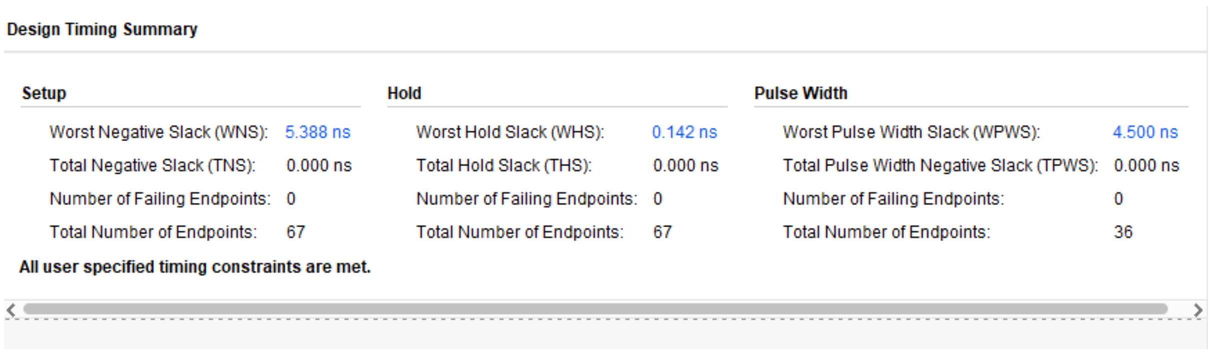
```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   41 |     0 |          0 |     32600 |  0.13 |
|   LUT as Logic          |   41 |     0 |          0 |     32600 |  0.13 |
|   LUT as Memory         |    0 |     0 |          0 |      9600 |  0.00 |
| Slice Registers         |   76 |     0 |          0 |     65200 |  0.12 |
|   Register as Flip Flop |   76 |     0 |          0 |     65200 |  0.12 |
|   Register as Latch     |    0 |     0 |          0 |     65200 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |      8150 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

Synthesis Report



```
Start RTL Component Statistics
---------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input   16 Bit       Adders := 1
+---Registers :
                    16 Bit    Registers := 2
                     8 Bit    Registers := 1
                     3 Bit    Registers := 1
                     1 Bit    Registers := 6
+---Muxes :
       7 Input   16 Bit        Muxes := 1
       2 Input   16 Bit        Muxes := 3
       4 Input   16 Bit        Muxes := 1
       2 Input    4 Bit        Muxes := 1
       2 Input    3 Bit        Muxes := 2
       2 Input    2 Bit        Muxes := 1
       7 Input    1 Bit        Muxes := 2
       2 Input    1 Bit        Muxes := 2
```

Timing Summary



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.388 ns	Worst Hold Slack (WHS):	0.142 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	67	Total Number of Endpoints:	67	Total Number of Endpoints:	36

All user specified timing constraints are met.

Conclusion

In this design, I learned a lot about the VHDL language design ideas and ISE software problems related to the troubleshooting methods. At the beginning of the design, I had just a simple understanding of the working principle of the finite synchronous state machine and try to solve the problem by step by step according to the requirement.

The code still requires a lot of improvements as I am still learning how to use some integral paths of the language to handle reusability and optimization. In the future, I hope to rewrite this code, to make it more readable, contained, reusable and robust.

I faced a couple of challenges, but the two major ones were the time and Vivado software installation on Ubuntu OS. Overall, the design meets all the requirements, and I am grateful for the opportunity.

References

[1] https://digilent.com/reference/programmable-logic/nexys-a7/start

[2] https://digilent.com/reference/vivado/getting_started/start

[3] https://www.fpga4student.com/2017/04/simple-debouncing-verilog-code-for.html

[4] https://www.allaboutcircuits.com/technical-articles/implementing-a-finite-state-machine-in-vhdl/