2/25/2022

Christopher Thomas

CST 231 – Digital System Design I

Lab 4

External Keypad

Christopher Thomas

Table of Contents

[Abstract 3](#_Toc96765376)

[Introduction 4](#_Toc96765377)

[Design 4](#_Toc96765378)

[Physical hardware design 4](#_Toc96765379)

[Synthesized hardware design 5](#_Toc96765380)

[Design Structure 5](#_Toc96765381)

[Modules 6](#_Toc96765382)

[Simulation and Testing 6](#_Toc96765383)

[Problems 6](#_Toc96765384)

[Results and Conclusion 7](#_Toc96765385)

[Appendix – A 8](#_Toc96765386)

[A-1 Pin Mappings Table 8](#_Toc96765387)

[A-2 RTL Diagrams 8](#_Toc96765388)

[A-2.a Lab04 8](#_Toc96765389)

[A-2.b clk\_divider 8](#_Toc96765390)

[A-2.c key\_scanner 8](#_Toc96765391)

[A-2.d key\_sequence\_gen 8](#_Toc96765392)

[A-2.e sequence\_check 8](#_Toc96765393)

[A-2.f timer 8](#_Toc96765394)

[A-2.g display 8](#_Toc96765395)

[A-3 State Machine Diagrams 8](#_Toc96765396)

[A-3.a key\_sequence\_gen 8](#_Toc96765397)

[A-3.b sequence\_check 8](#_Toc96765398)

[A-4 Module Code 8](#_Toc96765399)

[A-4.a Lab04 8](#_Toc96765400)

[A-4.b clk\_divider 8](#_Toc96765401)

[A-4.c key\_scanner 8](#_Toc96765402)

[A-4.d key\_sequence\_gen 8](#_Toc96765403)

[A-4.e sequence\_check 8](#_Toc96765404)

[A-4.f timer 8](#_Toc96765405)

[A-2.g display 8](#_Toc96765406)

[Table of Figures 9](#_Toc96765407)

[References 10](#_Toc96765408)

# Abstract

You will create a digital door lock system. This lock will use a 4x4 membrane keypad to enter a 4-digit code. The code will be checked against valid codes and the door either unlocked or an error LED will be flashed. The system should consist of a couple of LEDs. One will be lit to indicate (simulate a door is unlocking). The second will be lit to indicate an error. When either of these LEDs is lit it should remain in that state for a total of 5 seconds. The system should accept a 4-digit code from the keypad. This code will be entered in one digit at a time. The code should be displayed on the DE1-SoC 7-segment displays as it is being entered. To finish the code the ‘#’ key should be pressed. Once this is pressed the code should be checked against a list of known good codes. If found the door will be unlocked, otherwise the error will be shown. If the ‘\*’ key is entered at any point in the code entry. The code should be cleared. Also, once started if the code is not entered in within 15 seconds it should error out and reset. Break this out into its various modules using hierarchical design. This will require the implementation of a few state machines as well.

# Introduction

This lab is to create a digital lock system. This is intended to teach the participants how to include an external keypad to a system. This lab also makes use of two finite state machines to handle the sequence generation and valid code check. The intended outcome is to create a lock system that takes in a four hex code that could be implemented into a lock system.

Diagram

Description automatically generated

Figure : Top Level Diagram

# Design

## Physical hardware design

The physical hardware consists of a 4x4 membrane keypad connected to the GPIO of the device. The four columns are inputs, and the four rows are outputs. 1k ohm pull-up resistors were used as per the datasheet for keypad.

A picture containing text, electronics

Description automatically generated

Figure : Keypad Pull-up diagram

(Parallax Inc., 2011)

## Synthesized hardware design

### Design Structure

Diagram

Description automatically generated

Figure : Module Flow Diagram

The clk\_divider module divides the 50MHz clock down to 20Hz to negate debounce from the keypad. That clock signal is then used to synchronize the rest of the modules that have a clk signal input. The module key\_scanner rotates through the rows checking each row and column to see if a key has been pressed. Module key\_sequence\_gen receives they key from key\_scanner and adds that to the sequence. Once a full sequence is created it is sent to the sequence\_check module which checks if the sequence is a valid sequence. If it is it will unlock via an LED, if not will show an error occurred via an LED.

### Modules

#### Module: clk\_divider

The clk\_divider module divides the 50MHz default signal from the device to the 20Hz we need it to be for this application. This is achived by making a clk\_out output that only inverts its value after the onboard clock signal has inverted a certain number of times to slow the clock to the desired speed.

#### Module: key\_scanner

The key\_scanner module makes use the divided clk signal as an input, as well as the inputs for the columns. This module also consists of three outputs. The first being row which the value is rotated with the clock so that we look through the rows one by one continuously. The second being a four-bit value that is the hex value of the key that is pressed which gets updated each time a key is pressed. And finally, the key\_pressed input which is set high whenever a key press is detected and sent out to the timer module.

#### Module: timer

The timer module consists of a parameter called LIMIT that is defaulted to the value 300, three inputs clk, enable, and reset. The input clk is the divided clock signal from the clk\_divider module. The enable input is the key\_pressed output from the key\_scanner input which indicates that the timer should start. The final input is reset which if high resets the timer. The only output of the module is time\_up which goes high and acts as a reset for the system and is set high when the count has reached the LIMIT value which can be parameterized so that you can get the desired time.

#### Module: key\_sequence\_gen

The key\_sequence\_gen module has four inputs and four outputs. The inputs consist of clk, key\_pressed, times\_up, and key. The input clk is the divided 20Hz clock that the system is running off of. Times\_up is an input coming from the timer module that will go high after 15 seconds which is there so if you do not finish a full sequence within 15 seconds it will reset the process. Key\_pressed is an input coming from the key\_scanner module which activates the FSM to move from the idle state and start bringing in values and creating the sequence. Key is the four-bit hex value of the key that was pressed which will get loaded into sequence. The outputs consist of sequence, kp, temp, and new\_seq. Sequence is a sixteen-bit register that will store all four hex values to create the sequence that will be checked in the sequence\_check module. Kp is an output that starts the timer when the first key is pressed. Temp is a temporary register to store the sequence in until it is full built and is ready to be output. New\_seq is and output for the sequence\_check module to let it know that it has a new sequence to check for validity.

#### Module: sequence\_check

The sequence\_check module has four inputs and three outputs. The inputs consist of clk, times\_up, sequence, and new\_seq. Clk is the divided clock that all modules are synchronized to. Times\_up is an input coming from the timer which is used to control the error and unlock LED’s so that they only stay on for 5 seconds. Sequence is the sequence to be checked for validity that was build in the key\_sequence\_gen module. New\_seq is another input from the key\_sequence\_gen module that lets the FSM know that there is a new sequence to be checked and to have it move to the check state. The outputs consist of unlock, error, and start\_time. Unlock is the LED designated as the LED that will light up for 5 seconds when a valid sequence is inserted. Error is the LED designated to light up for 5 seconds when an invalid sequence is inserted, or the time runs out. Start\_time is the output to the timer to start the 5 second count once the appropriate LED is turned on.

#### Module: display

The display module has one input called hex and one output called display. The input is a hex value that determines what hex value should be displayed. A case statement then sets the value to be put into the display output so that the seven-segment display shows the proper symbol.

# Simulation and Testing

Five modules were simulated for this system. The timer module was tested by simulating a high enable and waiting for 300 clock cycles to verify that time\_up goes high. The display module was tested by simulating different hex value into the hex value and verifying that the display output matches the case statement. The key\_sequence\_gen module was tested by giving temp four different hex values and making sure that it was then properly sent into the sequence register. The key\_scanner module was tested by giving it different rows and columns combinations and verifying that key is getting the correct value for the case. The sequence\_check module was tested by giving it different valid and invalid sequence values and making sure we enter the correct state.

# Problems

There were two key issues that were encountered. The first issue was that in the key\_scanner module the case state for count was being ran by an always block that was triggered by count. This meant it was not synchronized to the clock and thus creating a delay that made the wrong key to be shown when pressed. This was solved by changing the always block to trigger off of the posedge of the clk. The second issue was in the FSM for sequence\_check. There was no Idle state that waited for a new sequence before checking to see if it was valid. So once a valid or invalid sequence was input it would just continuously go to the unlock or error state. This was solved by creating an input from the key\_sequence\_gen module that would let it know there was a new sequence to check, and it would not leave the Idle state until it received a high from that input.

# Results and Conclusion

This section is used to discuss the lab in its entirety. Discuss any problems that occurred during the lab (i.e. design issues). Any significant findings during the lab should also be discussed. Any major concerns that were realized during the lab should be discussed here. If anything should be done differently the next time, it should also be talked about. The conclusion should summarize the lab and detail what has been accomplished. A good conclusion should shore up any unanswered questions that were brought up during the lab.

The final design of this system had a total of seven modules. Two different versions of the timer were created one that would count to 15 seconds and one for 5 seconds. The 15 second timer was the time given to input the entire sequence to be check and the 5 second timer was for the duration of the LED’s to be on. The key\_scanner module would register a key press on the external keypad. The key\_sequence\_gen would receive a hex value from the key\_scanner module and build a sequence of four key presses. The sequence\_check module would check if the sequence was valid and enter either an error or unlock state. A display module was created to handle displaying what keys have been pressed so far on the onboard seven-segment displays. A key thing to this lab was having all the modules running on the same clock so everything was properly synchronized. All this creates a system that lets a user input a 4 key code into the system and will either unlock or stay locked depending on if the code was correct.

# Appendix – A

## A-1 Pin Mappings Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **FPGA Pin No.** | **Description** | **I/O Standard** |
| HEX0[0] | PIN\_AE26 | First Hex Display | 3.3-V LVTTL |
| HEX0[1] | PIN\_AE27 | First Hex Display | 3.3-V LVTTL |
| HEX0[2] | PIN\_AE28 | First Hex Display | 3.3-V LVTTL |
| HEX0[3] | PIN\_AG27 | First Hex Display | 3.3-V LVTTL |
| HEX0[4] | PIN\_AF28 | First Hex Display | 3.3-V LVTTL |
| HEX0[5] | PIN\_AG28 | First Hex Display | 3.3-V LVTTL |
| HEX0[6] | PIN\_AH28 | First Hex Display | 3.3-V LVTTL |
| HEX1[0] | PIN\_AJ29 | Second Hex Display | 3.3-V LVTTL |
| HEX1[1] | PIN\_AH29 | Second Hex Display | 3.3-V LVTTL |
| HEX1[2] | PIN\_AH30 | Second Hex Display | 3.3-V LVTTL |
| HEX1[3] | PIN\_AG30 | Second Hex Display | 3.3-V LVTTL |
| HEX1[4] | PIN\_AF29 | Second Hex Display | 3.3-V LVTTL |
| HEX1[5] | PIN\_AF30 | Second Hex Display | 3.3-V LVTTL |
| HEX1[6] | PIN\_AD27 | Second Hex Display | 3.3-V LVTTL |
| HEX2[0] | PIN\_AB23 | Third Hex Display | 3.3-V LVTTL |
| HEX2[1] | PIN\_AE29 | Third Hex Display | 3.3-V LVTTL |
| HEX2[2] | PIN\_AD29 | Third Hex Display | 3.3-V LVTTL |
| HEX2[3] | PIN\_AC28 | Third Hex Display | 3.3-V LVTTL |
| HEX2[4] | PIN\_AD30 | Third Hex Display | 3.3-V LVTTL |
| HEX2[5] | PIN\_AC29 | Third Hex Display | 3.3-V LVTTL |
| HEX2[6] | PIN\_AC30 | Third Hex Display | 3.3-V LVTTL |
| HEX3[0] | PIN\_AD26 | Fourth Hex Display | 3.3-V LVTTL |
| HEX3[1] | PIN\_AC27 | Fourth Hex Display | 3.3-V LVTTL |
| HEX3[2] | PIN\_AD25 | Fourth Hex Display | 3.3-V LVTTL |
| HEX3[3] | PIN\_AC25 | Fourth Hex Display | 3.3-V LVTTL |
| HEX3[4] | PIN\_AB28 | Fourth Hex Display | 3.3-V LVTTL |
| HEX3[5] | PIN\_AB25 | Fourth Hex Display | 3.3-V LVTTL |
| HEX3[6] | PIN\_AB22 | Fourth Hex Display | 3.3-V LVTTL |
| Rows[3] | PIN\_AC18 | GPIO Pin 0 | 3.3-V LVTTL |
| Rows[2] | PIN\_Y17 | GPIO Pin 1 | 3.3-V LVTTL |
| Rows[1] | PIN\_AD17 | GPIO Pin 2 | 3.3-V LVTTL |
| Rows[0] | PIN\_Y18 | GPIO Pin 3 | 3.3-V LVTTL |
| Columns[3] | PIN\_AK16 | GPIO Pin 4 | 3.3-V LVTTL |
| Columns[2] | PIN\_AK18 | GPIO Pin 5 | 3.3-V LVTTL |
| Columns[1] | PIN\_AK19 | GPIO Pin 6 | 3.3-V LVTTL |
| Columns[0] | PIN\_AJ19 | GPIO Pin 7 | 3.3-V LVTTL |
| LEDR[0] | PIN\_V16 | LED 0 | 3.3-V LVTTL |
| LEDR[1] | PIN\_W16 | LED 1 | 3.3-V LVTTL |

## A-2 RTL Diagrams

### A-2.a Lab04

Diagram, schematic

Description automatically generated

### A-2.b clk\_divider

Diagram, schematic

Description automatically generated

### A-2.c key\_scanner

Diagram, schematic

Description automatically generated

### A-2.d key\_sequence\_gen

Diagram, schematic

Description automatically generated

### A-2.e sequence\_check

Diagram

Description automatically generated

### A-2.f timer

Diagram

Description automatically generated

### A-2.g display

Diagram

Description automatically generated

## A-3 State Machine Diagrams

### A-3.a key\_sequence\_gen

Diagram

Description automatically generated

### A-3.b sequence\_check

Diagram

Description automatically generated

## A-4 Module Code

### A-4.a Lab04

Text

Description automatically generated

### A-4.b clk\_divider

Text

Description automatically generated

### A-4.c key\_scanner

module key\_scanner(

input clk\_20Hz,

input [3:0] columns,

output reg [3:0] rows,

output reg [3:0] key,

output reg key\_pressed

);

reg [1:0] count;

//init block for testbench

initial

begin

rows <= 4'b1111;

key <= 4'bzzzz;

count <= 0;

key\_pressed <= 0;

end

always @(posedge clk\_20Hz) //increase count on each posedge of clock

begin

count <= count + 1;

end

always @(posedge clk\_20Hz) //when count changes enter case statement

begin

case (count)

2'b00:

begin

case (columns) //check to see if columns is any of these values

4'b0111:

begin

key\_pressed <= 1;

key <= 4'b0001;

end

4'b1011:

begin

key\_pressed <= 1;

key <= 4'b0010;

end

4'b1101:

begin

key\_pressed <= 1;

key <= 4'b0011; //if so set key to specified value

end

4'b1110:

begin

key\_pressed <= 1;

key <= 4'b1010;

end

default:

begin

key\_pressed <= 0;

key <= 4'b0000;

end

endcase

rows <= 4'b1011;

end

2'b01:

begin

case (columns)

4'b0111:

begin

key\_pressed <= 1;

key <= 4'b0100;

end

4'b1011:

begin

key\_pressed <= 1;

key <= 4'b0101;

end

4'b1101:

begin

key\_pressed <= 1;

key <= 4'b0110;

end

4'b1110:

begin

key\_pressed <= 1;

key <= 4'b1011;

end

default:

begin

key\_pressed <= 0;

key <= 4'b0000;

end

endcase

rows <= 4'b1101;

end

2'b10:

begin

case (columns)

4'b0111:

begin

key\_pressed <= 1;

key <= 4'b0111;

end

4'b1011:

begin

key\_pressed <= 1;

key <= 4'b1000;

end

4'b1101:

begin

key\_pressed <= 1;

key <= 4'b1001;

end

4'b1110:

begin

key\_pressed <= 1;

key <= 4'b1100;

end

default:

begin

key\_pressed <= 0;

key <= 4'b0000;

end

endcase

rows <= 4'b1110;

end

2'b11:

begin

case (columns)

4'b0111:

begin

key\_pressed <= 1;

key <= 4'b1110;

end

4'b1011:

begin

key\_pressed <= 1;

key <= 4'b0000;

end

4'b1101:

begin

key\_pressed <= 1;

key <= 4'b1111;

end

4'b1110:

begin

key\_pressed <= 1;

key <= 4'b1101;

end

default:

begin

key\_pressed <= 0;

key <= 4'b0000;

end

endcase

rows <= 4'b0111;

end

default:

begin

key <= 4'bzzzz;

key\_pressed <= 0;

end

endcase

end

endmodule

### A-4.d key\_sequence\_gen

module key\_sequence\_gen(

input clk,

input key\_pressed,

input times\_up,

input [3:0] key,

output reg reset,

output reg [15:0] sequence,

output reg kp,

output reg [15:0] temp,

output reg new\_seq

);

reg [2:0] state;

/\* Creation of states\*/

parameter Idle = 3'b000,

First = 3'b001,

Second = 3'b010,

Third = 3'b011,

Fourth = 3'b101,

Fifth = 3'b110,

Output = 3'b111;

/\*Initial block for testbench\*/

initial

begin

reset <= 0;

sequence <= 0;

kp <= 0;

state <= Idle;

temp <= 0;

end

always @(posedge clk)

begin

if(times\_up == 1) //check for reset

begin

sequence <= 0;

kp <= 0;

temp <= 0;

state = Output;

end

case(state)

Idle:

begin

new\_seq <= 0;

temp <= 16'b1111111111111111;

if(key\_pressed) //when key pressed insert value into sequence and move states

begin

kp <= 1;

state <= First;

end

else

begin

state <= Idle;

end

end

First:

begin

if(key\_pressed) //when key pressed insert value into sequence and move states

begin

if(key == 4'b1111)

begin

state <= Output;

end

else if(key == 4'b1110)

begin

state <= Idle;

end

else

begin

temp[15:12] <= key;

state <= Second;

end

end

end

Second:

begin

if(key\_pressed) //when key pressed insert value into sequence and move states

begin

if(key == 4'b1111)

begin

state <= Output;

end

else if(key == 4'b1110)

begin

state <= Idle;

end

else

begin

temp[11:8] <= key;

state <= Third;

end

end

end

Third:

begin

if(key\_pressed) //when key pressed insert value into sequence and move states

begin

if(key == 4'b1111)

begin

state <= Output;

end

else if(key == 4'b1110)

begin

state <= Idle;

end

else

begin

temp[7:4] <= key;

state <= Fourth;

end

end

end

Fourth:

begin

if(key\_pressed) //when key pressed insert value into sequence and move states

begin

if(key == 4'b1111)

begin

state <= Output;

end

else if(key == 4'b1110)

begin

state <= Idle;

end

else

begin

temp[3:0] <= key;

state <= Fifth;

end

end

end

Fifth:

begin

if(key\_pressed)

begin

if(key == 4'b1111)

begin

state <= Output;

end

else if(key == 4'b1110)

begin

state <= Idle;

end

else

state <= Output;

end

end

Output: //set sequence to temp annd return to Idle state

begin

sequence <= temp;

new\_seq <= 1;

kp <= 0;

state <= Idle;

end

endcase

end

endmodule

### A-4.e sequence\_check

module sequence\_check(

input clk,

input times\_up,

input [15:0] sequence,

input new\_seq,

output reg unlock,

output reg error,

output reg start\_time,

output reg reset

);

reg [1:0] state;

/\* Creation of states\*/

parameter Idle = 2'b00,

Check = 2'b01,

Unlock = 2'b10,

Error = 2'b11;

/\*Initial block for testbench\*/

initial

begin

unlock <= 0;

error <= 0;

start\_time <= 0;

reset <= 0;

end

always @(posedge clk)

begin

case(state)

Idle:

begin

unlock <= 0;

error <= 0;

if(new\_seq)

begin

state <= Check;

end

else

begin

state <= Idle;

end

end

Check:

case(sequence)

16'h1234 : state <= Unlock;

16'hABCD : state <= Unlock;

16'h1122 : state <= Unlock;

16'h1010 : state <= Unlock;

16'h123A : state <= Unlock;

16'h2580 : state <= Unlock;

default : state <= Error;

endcase

Unlock:

begin

start\_time <= 1;

unlock <= 1;

if(times\_up == 1)

begin

unlock <= 0;

start\_time <= 0;

state <= Idle;

end

else

begin

unlock <= 1;

end

end

Error:

begin

start\_time <= 1;

error <= 1;

if(times\_up == 1)

begin

error <= 0;

start\_time <= 0;

state <= Idle;

end

else

begin

error <= 1;

end

end

endcase

end

endmodule

### A-4.f timer

Text

Description automatically generated

### A-2.g display

A picture containing table

Description automatically generated

# Table of Figures

[Figure 1: Top Level Diagram 3](#_Toc96764961)

[Figure 2: Keypad Pull-up diagram 4](#_Toc96764962)

[Figure 3: Module Flow Diagram 4](#_Toc96764963)

# References

Parallax Inc. (2011, 12 16). *www.parallax.com.* Retrieved from www.parallax.com: https://www.parallax.com/