PMOD Combination Lock

Jared Woelfel – 01570415 – 16 December 2020

Introduction

The goal of the design was to create a 4-digit combination lock using the 16-key PMOD keypad from Digilent in combination with the Artix-7 XC7A35T Basys-3 FPGA. The output of the lock's state was to be displayed on the Basys-3 board's 7-segment display. The lock needed to display the unlocked state if and only if 4 specific (predefined) digits were pressed in sequential order on the keypad. Any other sequence of button presses should result in the locked state.

Criteria

The design must interface the Digilent PMOD KYPD with the Basys-3 board. It must be able to read the output of the keypad without errors caused by bouncing. It must then compare the output of the keypad to the respective digit of the 4-digit predefined combination. Once the correct combination of values are entered in the correct sequence, the 7-segment display should display some indication that the lock is now in an unlocked state. Any mistake in the sequence of input keys should result in the lock going back to a completely locked state. Any button press while in the unlocked state should result in the lock going back to a completely locked state.

Solution

The 16-key Digilent KYPD was connected to the Basys-3 according to the PMOD port pin diagram in Figure 1 [3] and the KYPD pin diagram in Figure 2 and associated pinout table in Table 1 [2]. The design for the combination lock was created according to the state diagram as seen in Figure 3 to ensure that all elements of the criteria were met. The Verilog code used to implement the combination lock was developed with assistance from resources on the Digilent website for the 7-segment display controller (Figure 4) and keypad decoder (Figure 5) [2]. The main module was developed independently and relied on a series of conditional statements and case statements to update the lock state based on the keypad input (Figure 6). Debouncing was achieved by stating that if the correct button was pressed multiple times the lock should remain in the same state. The lock state was displayed on the 7-segment display as a range of values from 0 to 4. 0 represented a completely locked state while 4 represented a completely unlocked state. 1, 2, and 3 represented intermediate lock states. Once the lock was in state 4, any button press sent the lock back to state 0.

Proof of Concept

The Basys-3 FPGA implementation was tested by sequentially entering the correct predefined code of "71B8" to the keypad. Photos of the FPGA and 7-segment display were taken in sequential order between each key press and recorded in Figures 7 through 11. The lock state went from 0 (locked) to 4 (unlocked) in sequential order. Any mistake in the sequence resulted in the lock state returning to 0 as seen in the video included with this report submission. Additionally, any key press besides 8 while the lock was in state 4 (unlocked) sent the lock to state 0 (locked). The bouncing issue was solved using the case statements in the Verilog code and resulted in each intermediate lock state being stable.

Conclusion

There are several improvements that could be made to this design. One of the most influential improvements would be to edit the debouncing method. The decoder's built-in debouncing was not effective, so another debouncing method built into the case statements of the combination lock was used to ensure that each lock state was stable. Switching to a delay debouncing method built into the decoder would allow the lock to reset even if the sequentially correct button was pressed twice. Additionally, it would allow for a password programming function to be built in if the user wanted to reset the password on-device instead of inside the Verilog code (case statements made this impossible but were the most stable debouncing method). Another possible improvement for practical application would be to reduce the display to only outputting a 0 (locked) or 1 (unlocked) on the 7-segment display. This would prevent a would-be intruder from determining the passcode by seeing which values cause the lock state to tick from 0 to 1, 1 to 2, and so forth. This could easily be implemented by creating an additional variable that is fed to the 7-segment controller module and is 0 unless the solvestate variable becomes 4 and then it the variable would switch to 1. This variable/signal could also be output to a different PMOD port to activate any peripheral external lock.

Reference

[1] O. Hermes, Ed., *Creately*. Bellum Publishing, 2012.

[2] "Pmod KYPD," *Digilentinc.com*. [Online]. Available: https://reference.digilentinc.com/reference/pmod/pmodkypd/start. [Accessed: 17-Dec-2020].

[3] S. Bobrowicz, "Basys 3 Reference," *Digilentinc.com*. [Online]. Available: https://reference.digilentinc.com/basys3/refmanual. [Accessed: 17-Dec-2020].

Results

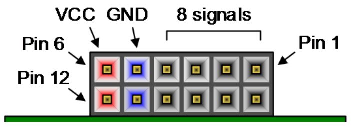

Figure 1: Basys-3 PMOD Port Pinout Diagram

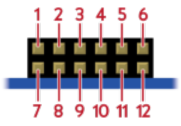

Figure 2: Digilent 16-Key KYPD Pinout Diagram

Table 1: Digilent 16-Key KYPD Pinout Diagram Label Table

Pin 1	COL4
Pin 2	COL3
Pin 3	COL2
Pin 4	COL1
Pin 5	GND
Pin 6	VCC
Pin 7	ROW4
Pin 8	ROW3
Pin 9	ROW2
Pin 10	ROW1
Pin 11	GND
Pin 12	VCC

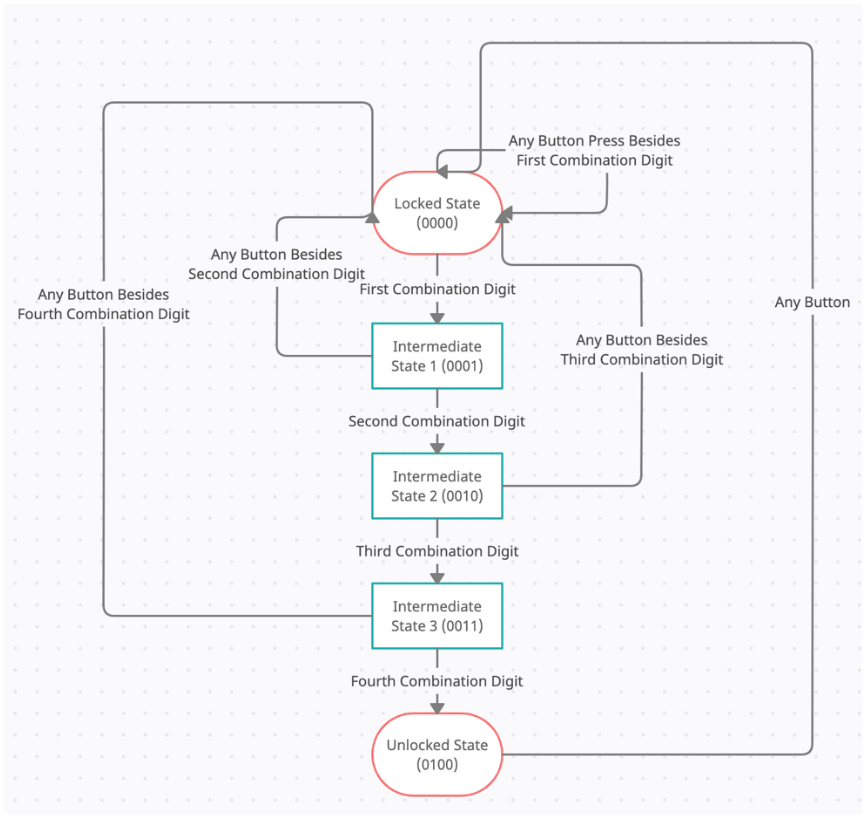
Any Button Press Besides First Combination Digit
Locked State (0000)
Any Button Besides Second Combination Digit
First Combination Digit
Any Button Besides Fourth Combination Digit
Any Button
Intermediate State 1 (0001)
Any Button Besides Third Combination Digit
Second Combination Digit
Intermediate State 2 (0010)
Third Combination Digit
Intermediate State 3 (0011)
Fourth Combination Digit
Unlocked State (0100)

Figure 3: Combination Lock Design State Diagram (created using [1])

7 Segment Display Controller Verilog Module:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////
// Company: Digilent Inc 2011
// Engineer: Michelle Yu
```

```
//Josh Sackos
// Create Date:    07/23/2012
//
// Module Name:    DisplayController
// Project Name:   PmodKYPD_Demo
// Target Devices: Basys3
// Tool versions:  Xilinx ISE 14.1
// Description: This file defines a DisplayController that controls the seven
segment display that works with
//                        the output of the Decoder.
//
// Revision History:
// Revision 0.01 - File Created (Michelle Yu)
// Revision 0.02 - Converted from VHDL to Verilog (Josh Sackos)
////////////////////////////////////////////////////////////////////////////////
/////////////////////////////

//
================================================================================
=================
//                       Define Module
//
================================================================================
=================
module DisplayController(
DispVal,
anode,
segOut
);

//
================================================================================
=================
//                       Port Declarations
//
================================================================================
=================

input [3:0] DispVal;                // Output from the Decoder
output [3:0] anode;                 // Controls the display digits
output [6:0] segOut;                // Controls which digit to display

//
================================================================================
=================
//                       Parameters, Regsiters, and Wires
//
================================================================================
=================

// Output wires and registers
wire [3:0] anode;
reg [6:0] segOut;

//
================================================================================
=================
//
      Implementation
//
==============================================================================
=================

// only display the rightmost digit
assign anode = 4'b1110;

//------------------------------
//             Segment Decoder
// Determines cathode pattern
//   to display digit on SSD
//------------------------------
always @(DispVal) begin
case (DispVal)

4'h0 : segOut <= 7'b1000000;  // 0
4'h1 : segOut <= 7'b1111001;  // 1
4'h2 : segOut <= 7'b0100100;  // 2
4'h3 : segOut <= 7'b0110000;  // 3
4'h4 : segOut <= 7'b0011001;  // 4
4'h5 : segOut <= 7'b0010010;  // 5
4'h6 : segOut <= 7'b0000010;  // 6
4'h7 : segOut <= 7'b1111000;  // 7
4'h8 : segOut <= 7'b0000000;  // 8
4'h9 : segOut <= 7'b0010000;  // 9
4'hA : segOut <= 7'b0001000;  // A
4'hB : segOut <= 7'b0000011;  // B
4'hC : segOut <= 7'b1000110;  // C
4'hD : segOut <= 7'b0100001;  // D
4'hE : segOut <= 7'b0000110;  // E
4'hF : segOut <= 7'b0001110;  // F

default : segOut <= 7'b0111111;

endcase
end

endmodule
```

Figure 4: 7-Segment Display Controller Verilog Code (borrowed from [2])

Keypad Decoder Verilog Module:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
/////////////////////////////
// Company: Digilent Inc 2011
// Engineer: Michelle Yu
//                         Josh Sackos
// Create Date:    07/23/2012
//
// Module Name:    Decoder
// Project Name:   PmodKYPD_Demo
// Target Devices: Basys3
// Tool versions:  Xilinx ISE 14.1
// Description: This file defines a component Decoder for the demo project
PmodKYPD. The Decoder scans
```

```
// each column by asserting a low to the pin corresponding to the column at
1KHz. After a
//column is asserted low, each row pin is checked. When a row pin is detected
to be low,
//the key that was pressed could be determined.
//
// Revision History:
// Revision 0.01 - File Created (Michelle Yu)
// Revision 0.02 - Converted from VHDL to Verilog (Josh Sackos)
//////////////////////////////////////////////////////////////////////////////
/////////////////////////////

//
==============================================================================
=================
//                                 Define Module
//
==============================================================================
=================
module Decoder(
clk,
Row,
Col,
DecodeOut
);

//
==============================================================================
=================
//                           Port Declarations
//
==============================================================================
=================
input clk;                                // 100MHz onboard clock
input [3:0] Row;                    // Rows on KYPD
output [3:0] Col;             // Columns on KYPD
output [3:0] DecodeOut; // Output data

//
==============================================================================
=================
//                     Parameters, Regsiters, and Wires
//
==============================================================================
=================

// Output wires and registers
reg [3:0] Col;
reg [3:0] DecodeOut;

// Count register
reg [19:0] sclk;

//
==============================================================================
=================
//                                 Implementation
```

```
//
================================================================================
=================

always @(posedge clk) begin

// 1ms
if (sclk == 20'b00011000011010100000) begin
//C1
Col <= 4'b0111;
sclk <= sclk + 1'b1;
end

// check row pins
else if(sclk == 20'b00011000011010101000) begin
//R1
if (Row == 4'b0111) begin
DecodeOut <= 4'b0001;         //1
end
//R2
else if(Row == 4'b1011) begin
DecodeOut <= 4'b0100;         //4
end
//R3
else if(Row == 4'b1101) begin
DecodeOut <= 4'b0111;         //7
end
//R4
else if(Row == 4'b1110) begin
DecodeOut <= 4'b0000;         //0
end
sclk <= sclk + 1'b1;
end

// 2ms
else if(sclk == 20'b00110000110101000000) begin
//C2
Col<= 4'b1011;
sclk <= sclk + 1'b1;
end

// check row pins
else if(sclk == 20'b00110000110101001000) begin
//R1
if (Row == 4'b0111) begin
DecodeOut <= 4'b0010;         //2
end
//R2
else if(Row == 4'b1011) begin
DecodeOut <= 4'b0101;         //5
end
//R3
else if(Row == 4'b1101) begin
DecodeOut <= 4'b1000;         //8
end
//R4
else if(Row == 4'b1110) begin
DecodeOut <= 4'b1111;         //F
```

```
end
sclk <= sclk + 1'b1;
end

//3ms
else if(sclk == 20'b01001001001111100000) begin
//C3
Col<= 4'b1101;
sclk <= sclk + 1'b1;
end

// check row pins
else if(sclk == 20'b01001001001111101000) begin
//R1
if(Row == 4'b0111) begin
DecodeOut <= 4'b0011;         //3
end
//R2
else if(Row == 4'b1011) begin
DecodeOut <= 4'b0110;         //6
end
//R3
else if(Row == 4'b1101) begin
DecodeOut <= 4'b1001;         //9
end
//R4
else if(Row == 4'b1110) begin
DecodeOut <= 4'b1110;         //E
end

sclk <= sclk + 1'b1;
end

//4ms
else if(sclk == 20'b01100001101010000000) begin
//C4
Col<= 4'b1110;
sclk <= sclk + 1'b1;
end

// Check row pins
else if(sclk == 20'b01100001101010001000) begin
//R1
if(Row == 4'b0111) begin
DecodeOut <= 4'b1010; //A
end
//R2
else if(Row == 4'b1011) begin
DecodeOut <= 4'b1011; //B
end
//R3
else if(Row == 4'b1101) begin
DecodeOut <= 4'b1100; //C
end
//R4
else if(Row == 4'b1110) begin
DecodeOut <= 4'b1101; //D
end
```

```
sclk <= 20'b00000000000000000000;
end

// Otherwise increment
else begin
sclk <= sclk + 1'b1;
end

end

endmodule
```

Figure 5: Digilent 16-Key KYPD Decoder Verilog Code (borrowed from [2])

Combination Lock Verilog Module:

```
timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
/////////////////////////////
// Company: Digilent Inc 2011
// Engineer: Michelle Yu
//                        Josh Sackos
// Create Date:    17:05:39 08/23/2011
//
// Module Name:    PmodKYPD
// Project Name:   PmodKYPD_Demo
// Target Devices: Basys3
// Tool versions:  Xilinx ISE 14.1
// Description: This file defines a project that outputs the key pressed on
the PmodKYPD to the
//                              seven segment display.
//
// Revision History:
//                                   Revision 0.01 - File Created (Michelle
Yu)
//                                         Revision 0.01 - Converted from VHDL
to Verilog (Josh Sackos)
//////////////////////////////////////////////////////////////////////////////
/////////////////////////////

//
==============================================================================
=================
//
      Define Module
//
==============================================================================
=================
module PmodKYPD(
clk,
JB,
an,
seg
);


//
==============================================================================
=================
//                                                                  Port
Declarations
//
============================================================================
=================
input clk;                          // 100Mhz onboard clock
inout [7:0] JB;               // Port JB on Basys3, JB[3:0] is Columns,
JB[10:7] is rows
output [3:0] an;              // Anodes on seven segment display
output [6:0] seg;             // Cathodes on seven segment display

//
============================================================================
=================
//                                                      Parameters, Regsiters,
and Wires
//
============================================================================
=================

// Output wires
wire [3:0] an;
wire [6:0] seg;
wire [3:0] Decode;

//Intermediate Parameters
reg [3:0] solvestate = 4'b0000;
reg [31:0] counter = 0;


//
============================================================================
=================
//
      Implementation
//
============================================================================
=================

//-------------------------------------------------
//                                  Decoder
//-------------------------------------------------



Decoder C0(
.clk(clk),
.Row(JB[7:4]),
.Col(JB[3:0]),
.DecodeOut(Decode)
); //runs code to decode input from 16 button keypad

always @(posedge clk) begin//create slower clock for updating the solvestate
at a lower sampling frequency
if (counter < 32'h007FFFFF)
begin
counter <= counter +1;
end
```

```
else if (counter == 32'h007FFFFF)//only update solvestate on a specific
interval defined by the counter
begin
counter <= 0;
if (solvestate == 4'b0000)//if solvestate is 0, check for a button press of 7
to send solvestate to 1. otherwise, maintain solvestate = 0.
begin
case (Decode)

4'h0 : solvestate <= 4'b0000;  // 0
4'h1 : solvestate <= 4'b0000;  // 1
4'h2 : solvestate <= 4'b0000;  // 2
4'h3 : solvestate <= 4'b0000;  // 3
4'h4 : solvestate <= 4'b0000;  // 4
4'h5 : solvestate <= 4'b0000;  // 5
4'h6 : solvestate <= 4'b0000;  // 6
4'h7 : solvestate <= 4'b0001;  // 7
4'h8 : solvestate <= 4'b0000;  // 8
4'h9 : solvestate <= 4'b0000;  // 9
4'hA : solvestate <= 4'b0000;     // A
4'hB : solvestate <= 4'b0000;    // B
4'hC : solvestate <= 4'b0000;    // C
4'hD : solvestate <= 4'b0000;    // D
4'hE : solvestate <= 4'b0000;    // E
4'hF : solvestate <= 4'b0000;    // F

default : solvestate <= 4'b0000;

endcase
end
else if (solvestate == 4'b0001)//if solvestate is 1, check for a button press
of 1 to send solvestate to 2.
//if button 7 is bouncing, maintain state at solvestate = 1. if any other
button is pressed, send solvestate back to 0.
begin
case (Decode)

4'h0 : solvestate <= 4'b0000;  // 0
4'h1 : solvestate <= 4'b0010;  // 1
4'h2 : solvestate <= 4'b0000;  // 2
4'h3 : solvestate <= 4'b0000;  // 3
4'h4 : solvestate <= 4'b0000;  // 4
4'h5 : solvestate <= 4'b0000;  // 5
4'h6 : solvestate <= 4'b0000;  // 6
4'h7 : solvestate <= 4'b0001;  // 7
4'h8 : solvestate <= 4'b0000;  // 8
4'h9 : solvestate <= 4'b0000;  // 9
4'hA : solvestate <= 4'b0000;     // A
4'hB : solvestate <= 4'b0000;    // B
4'hC : solvestate <= 4'b0000;    // C
4'hD : solvestate <= 4'b0000;    // D
4'hE : solvestate <= 4'b0000;    // E
4'hF : solvestate <= 4'b0000;    // F

default : solvestate <= 4'b0001;

endcase
```

```
end
else if (solvestate == 4'b0010)//if solvestate is 2, check for a button press
of B to send solvestate to 3.
//if button 1 is bouncing, maintain state at solvestate = 2. if any other
button is pressed, send solvestate back to 0.
begin
case (Decode)

4'h0 : solvestate <= 4'b0000;  // 0
4'h1 : solvestate <= 4'b0010;  // 1
4'h2 : solvestate <= 4'b0000;  // 2
4'h3 : solvestate <= 4'b0000;  // 3
4'h4 : solvestate <= 4'b0000;  // 4
4'h5 : solvestate <= 4'b0000;  // 5
4'h6 : solvestate <= 4'b0000;  // 6
4'h7 : solvestate <= 4'b0000;  // 7
4'h8 : solvestate <= 4'b0000;  // 8
4'h9 : solvestate <= 4'b0000;  // 9
4'hA : solvestate <= 4'b0000;     // A
4'hB : solvestate <= 4'b0011;    // B
4'hC : solvestate <= 4'b0000;    // C
4'hD : solvestate <= 4'b0000;    // D
4'hE : solvestate <= 4'b0000;    // E
4'hF : solvestate <= 4'b0000;    // F

default : solvestate <= 4'b0010;

endcase
end
else if (solvestate == 4'b0011)//if solvestate is 3, check for a button press
of 8 to send solvestate to 4.
//if button B is bouncing, maintain state at solvestate = 3. if any other
button is pressed, send solvestate back to 0.
begin
case (Decode)

4'h0 : solvestate <= 4'b0000;  // 0
4'h1 : solvestate <= 4'b0000;  // 1
4'h2 : solvestate <= 4'b0000;  // 2
4'h3 : solvestate <= 4'b0000;  // 3
4'h4 : solvestate <= 4'b0000;  // 4
4'h5 : solvestate <= 4'b0000;  // 5
4'h6 : solvestate <= 4'b0000;  // 6
4'h7 : solvestate <= 4'b0000;  // 7
4'h8 : solvestate <= 4'b0100;  // 8
4'h9 : solvestate <= 4'b0000;  // 9
4'hA : solvestate <= 4'b0000;     // A
4'hB : solvestate <= 4'b0011;    // B
4'hC : solvestate <= 4'b0000;    // C
4'hD : solvestate <= 4'b0000;    // D
4'hE : solvestate <= 4'b0000;    // E
4'hF : solvestate <= 4'b0000;    // F

default : solvestate <= 4'b0011;

endcase
end
```

```
else if (solvestate == 4'b0100)//if solvestate is 4, maintain solvestate
value if button 8 is bouncing.
//if any other button is pressed, send solvestate back to 0.
begin
case (Decode)

4'h0 : solvestate <= 4'b0000;  // 0
4'h1 : solvestate <= 4'b0000;  // 1
4'h2 : solvestate <= 4'b0000;  // 2
4'h3 : solvestate <= 4'b0000;  // 3
4'h4 : solvestate <= 4'b0000;  // 4
4'h5 : solvestate <= 4'b0000;  // 5
4'h6 : solvestate <= 4'b0000;  // 6
4'h7 : solvestate <= 4'b0000;  // 7
4'h8 : solvestate <= 4'b0100;  // 8
4'h9 : solvestate <= 4'b0000;  // 9
4'hA : solvestate <= 4'b0000;     // A
4'hB : solvestate <= 4'b0000;    // B
4'hC : solvestate <= 4'b0000;    // C
4'hD : solvestate <= 4'b0000;    // D
4'hE : solvestate <= 4'b0000;    // E
4'hF : solvestate <= 4'b0000;    // F

default : solvestate <= 4'b0100;

endcase
end
end

end
//------------------------------------------------
//          Seven Segment Display Controller
//------------------------------------------------

DisplayController C1(
.DispVal(solvestate),
.anode(an),
.segOut(seg)
);//runs code to display solvestate on the 7 segment display.

endmodule
```

Figure 6: Combination Lock Verilog Code

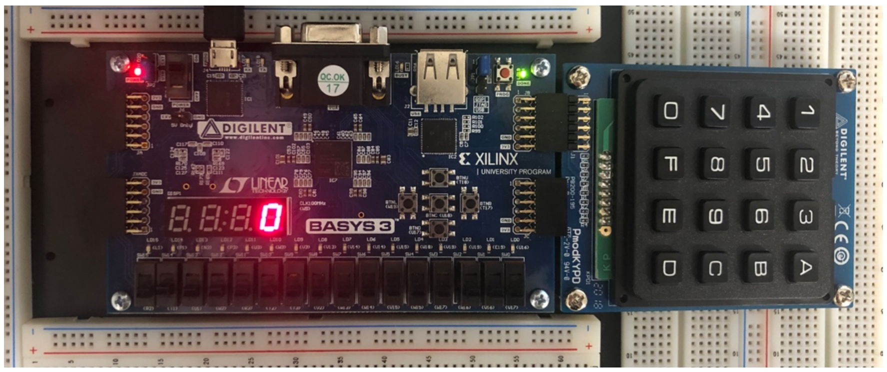

Figure 7: Combination Lock Output for Boot-up, Incorrect Sequence, and Any Button Press in State 4

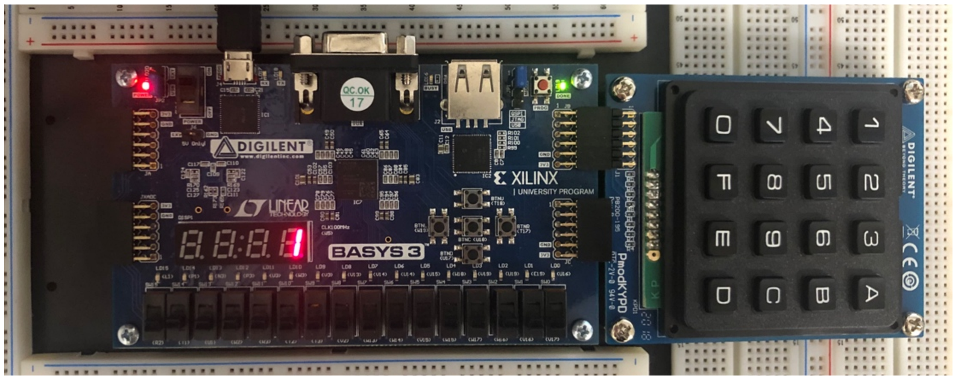
Figure 8: Combination Lock Output After Key 7 Pressed

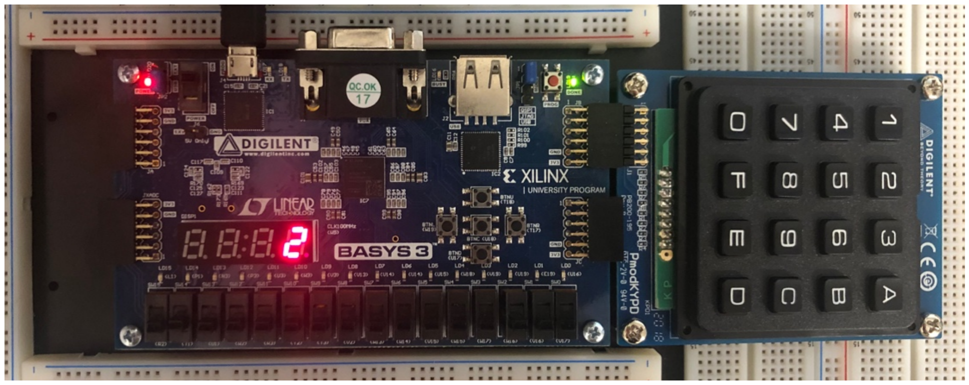
Figure 9: Combination Lock Output After Key 1 Pressed In State 1

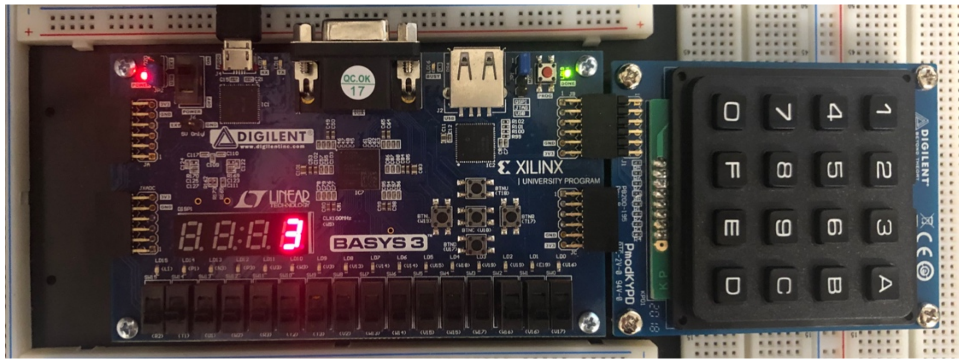
Figure 10: Combination Lock Output After Key B Pressed In State 2

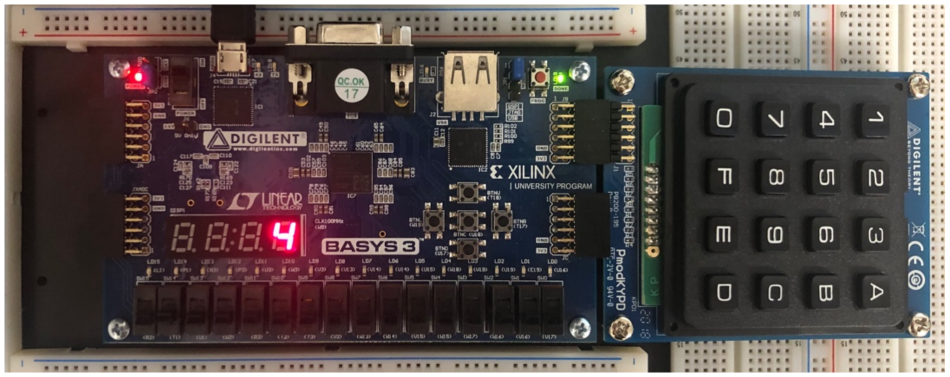
Figure 11: Combination Lock Output After Key 8 Pressed In State 3