4/17/2017

Project Report: Home Security System

ELEX 7660: Digital System Design

Group 6: Cole Raschpichler & Paul Wiebe

Table of Contents

Table of Figures

Introduction & Objectives

The project that will be described in this report is a Home Security System that was designed, built, and tested using an Altera DE0-Nano FPGA board, along with several hardware components including optical sensors and magnetic sensors, as well as components that were used in previous labs, which include the 16-digit keypad, speaker, and OLED display. Code that was written in the previous labs to implement these components are used in this project as well.

The motivation behind picking a project of this caliber was to further working knowledge of an FPGA system and System Verilog HDL, as well as to be further knowledgeable with the software used to design FPGA programs, which in this case included Quartus Prime, and it's subprograms, Qsys and Nios. A more complex project was avoided so as to not be overwhelmed and overworked.

The objectives behind this project was to simulate as close as possible a Home Security System. These included:

- Implementation of different modes, including disarmed mode, home mode, armed mode, and alarmed mode
- A keypad to select which mode to be in, as well as to enter the passcode to arm or disarm the system
- An OLED display to show which mode the system is in, to show which sensors are triggered, and to show what the code being entered is
- A speaker for the alarm when the system has been breached

The system has eight sensors. Five magnetic sensors were used for doors and windows spread across three floors, while three optical sensors were used as motion sensors, one for each floor.

The difference between home mode and armed mode is that when in armed mode, any sensor will trigger the alarm. This is similar to when no one is home, the system is armed, and someone breaks into the house. In home mode, the motion sensors are deactivated, and only the triggering of the door and window sensors will trigger the alarm. This is analogous to when the homeowner is at home, they still want their house to be secure, and be alerted if someone breaks into their home, and not trigger the alarm while moving around the house.

Achievements

Upon working through the project, most of the initial objectives of the project were satisfied. The standard features of a typical home security system were successfully achieved, namely the four modes of operation: disarmed mode, home mode, armed mode, and alarmed mode.

Disarmed mode consisted of the system being inactive, where sensors could be triggered without anything happening. Home mode consisted of the window and door sensors being active, but not the motion sensors. Armed mode consisted of all sensors being active. Alarmed mode was triggered when the system was breached in either home mode or armed mode. In order to maintain alarmed mode when a sensor is triggered, the sensor variable was latched, and only unlatched when the system was disarmed.

Using code and knowledge from lab 2, the keypad was implemented to be able to select one of the four modes described above, as well as to be able to enter a passcode to enter one of these four modes. The layout of the keypad was implemented as shown in the figure below:

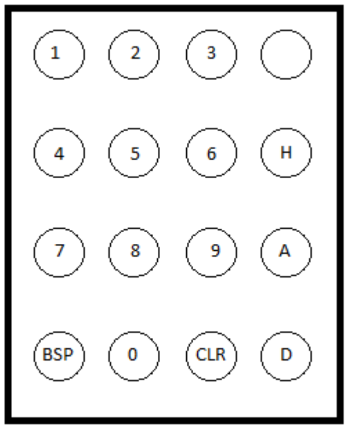

Figure 1: Keypad operation layout.

Keypad Symbol	Operation	Keypad Symbol	Operation
1	Enter Decimal Value 1	9	Enter Decimal Value 9
2	Enter Decimal Value 2	10	Enter Decimal Value 10
3	Enter Decimal Value 3	BSP	Backspace a Character
4	Enter Decimal Value 4	CLR	Clear Code
5	Enter Decimal Value 5	H	Enter Home Mode
6	Enter Decimal Value 6	A	Enter Armed Mode
7	Enter Decimal Value 7	D	Enter Disarmed Mode
8	Enter Decimal Value 8		

Table 1: Keypad operation description for each key.

Entering of the code into the keypad allowed the user the option to be able to backspace the four-digit code by one character, or the option to clear the code entirely. In order to go from one mode to the next, the user had to press either the disarmed, home, or armed button on the keypad before entering the code, otherwise entering of the code would not be allowed. Note also that the user can go from home mode to armed mode, but not from armed mode to home mode, so as to better simulate the operation of home security systems.

The most challenging aspect of this project was getting the OLED display to work properly. Note that the OLED display from lab 4 was used, and not the LCD display mentioned in the project proposal. The template used for the OLED display is pictured below:

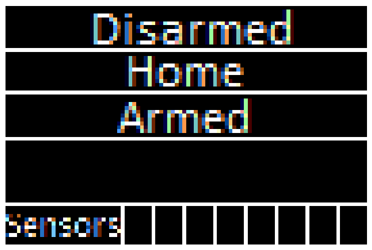

Figure 2: Template on the OLED display.

Based on the code from lab 4, printing the image onto the OLED display was simple. However, updating the image over time as keys were pressed became a challenge. Eventually, it was figured out that through Qsys that a parallel I/O port had to be implemented. Once that was completed, a 32-bit variable was sent through the top-level system verilog file to Nios, where it used the bit values to change the OLED display to show which mode the system was in, what the code being entered is, and what sensors are triggered. These changes to the OLED display were done by using a mask which overlaid the template seen in Figure 2. This mask was built up by using characters and images from the prototype image seen in Figure 3. By specifying x and y position on the display, x and y size of image, and the location of the image to be used in the prototype image, the display was able to show the correct information at the right time.

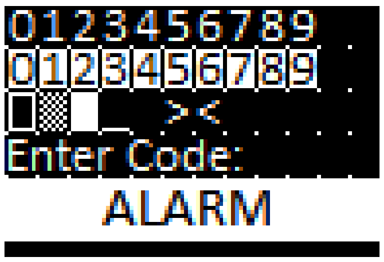

Figure 3: Prototype for Display Characters

Due to the amount of time it took to get the OLED display working, the original plan of including data logging was not implemented. Instead, debugging of the system was done to ensure proper operation.

Operation

Operation of the system is very similar to that of a standard security system. In the code, the default start up mode is disarmed mode, and the passcode is 1234.

In order to get from one mode to the other, the operation involves pressing the key of the desired mode the user would like to be in, then entering the code. For example, if the user is in disarmed mode, and the user presses either the home key or the armed key, the OLED display will prompt the user to enter the passcode. The passcode must be entered correctly in order to get to the desired mode. The OLED display will have arrows on either side of the mode currently in. When the system has moved to the desired mode, the arrows will move to that mode. Note that while it is possible to go from home mode to armed mode, it is not possible to go from armed mode to home mode. The user must go to disarmed mode first if home mode is desired after being in armed mode.

If a sensor is triggered that sets off the alarm, the OLED display will show that the system is in alarmed mode. In order to disarm the system, the user must press the disarmed key, enter the code, then the system will be in disarmed mode, and the alarm will shut off.

Note also that if attempting to transition from disarmed mode to either home mode or armed mode, the system will not arm if a sensor is triggered while entering the passcode. All sensors must be cleared, with the exception of the motion sensors when entering home mode.

Entering of the passcode has a backspace and clear option in the event that the code was entered wrong on the keypad. If the code is wrong after entering four digits, the program will detect that it does not match the passcode desired, and it will clear the code automatically.

The OLED display shows which sensors are triggered if they are triggered.

Resources

In order to use the image that was shown on the OLED display, the image had to be converted into a series of hex values. Due to unknown knowledge of how to do this, the following resource was used to convert images to hex values:

https://www.mathworks.com/matlabcentral/fileexchange/34713-imagetohex--hextoimage-cconversion

Conclusion & Recommendations

The project ended up being more work than initially thought, as there was a lot of trouble editing the image on the OLED display. This issue took up a lot of time. The other aspects of the project took up a little more than the expected amount of time; however, it proved to be a valuable learning experience on digital hardware systems. Due to the limited knowledge of FPGA's and System Verilog, this project was a suitable option to further learning about HDL and FPGA's.

Should this project be repeated or expanded upon, recommendations include, but are not limited to:

- Adding a data logging module that will log when the system has been breached, including details on when it happened, where it happened, etc. An SD card module could be used for this, and/or writing it in real time to a computer.
- Using a bigger display to show more information, and making it update faster.
- Implementing a delay from arming the system from disarmed/home mode to armed mode, so as to simulate a person leaving their house after arming the system.
- Implementing a delay after walking into the house and disarming the system, before the alarm goes off.

Appendix A: Code

The following section of the appendix includes all the code used in this project, including the System Verilog files and the C files. Files that were left out include the header files for the image, and the QSF pins file.

SV Module: project_top.sv

```
// File Name: project_top.sv
// Purpose: Top-level module for project.
// Authors: Cole Raschpichler & Paul Wiebe
// Date: April 7th, 2017

module project_top ( input logic CLOCK_50, temp_input,
                            input logic [1:0] KEY,
                            input logic [7:0] sensor,
                            output logic [3:0] kpc,  // column select, active-low
                            (* altera_attribute = "-name WEAK_PULL_UP_RESISTOR ON" *)
                            input logic  [3:0] kpr,  // rows, active-low w/ pull-ups
                            output logic rgb_din, rgb_clk, rgb_cs, rgb_dc, rgb_res,
                            output logic led_bar[10:0],
                            output logic spkr,
                            output logic [7:0] LED,

                            output logic [12:0] DRAM_ADDR,   //  dram.addr
                            output logic [1:0]  DRAM_BA,     //      .ba
                            output logic        DRAM_CAS_N,  //      .cas_n
                            output logic        DRAM_CKE,    //      .cke
                            output logic        DRAM_CS_N,   //      .cs_n
                            inout  logic [15:0] DRAM_DQ,     //      .dq
                            output logic [1:0]  DRAM_DQM,    //      .dqm
                            output logic        DRAM_RAS_N,  //      .ras_n
                            output logic        DRAM_WE_N,   //      .we_n
                            output logic DRAM_CLK ) ;

   logic clk;                    // 2kHz clock for keypad scanning
    logic clk_1sec;
   logic [4:0] num;              // value of pressed key
    logic valid;
    logic pass_valid, pass_enable;
    logic [15:0] passcode;
    logic [3:0] mode_send;
    logic [15:0] temp_send;
    logic [10:0] led_bar_send;

    colseq colseq_0(.*);
    kpdecode kpdecode_0(.*);
    mode_sensor mode_sensor_0(.*);
    alarm_generator alarm_generator_0(.*);
    passcode passcode_0(.*);

   lab5 u0
     (
      .clock_50_clk  (CLOCK_50),     //  clock_50.clk
      .reset_reset   (reset_n),          //  reset.reset

      .sdram_clk_clk (DRAM_CLK),     // sdram_clk.clk
      .sdram_addr    (DRAM_ADDR),    //     sdram.addr
      .sdram_ba      (DRAM_BA),      //          .ba
      .sdram_cas_n   (DRAM_CAS_N),   //          .cas_n
      .sdram_cke     (DRAM_CKE),     //          .cke
      .sdram_cs_n    (DRAM_CS_N),    //          .cs_n
      .sdram_dq       (DRAM_DQ),      //           .dq
      .sdram_dqm      (DRAM_DQM),     //           .dqm
      .sdram_ras_n    (DRAM_RAS_N),   //           .ras_n
      .sdram_we_n     (DRAM_WE_N),    //           .we_n

        .spi_sclk (rgb_clk),
        .spi_mosi (rgb_din),
        .spi_csn (rgb_cs),
        .spi_dcn (rgb_dc),
        .spi_resetn (rgb_res),
        .pio_export({mode_send, temp_send, 1'b0, 1'b0, 1'b0, temp_input,
led_bar_send[7:0]})
      );


    logic reset_n;
   assign reset_n = KEY[0];
    logic [15:0] i;

    initial begin
        i = 4'h0000;
        clk = 1'b0;
    end

    always_ff@( posedge CLOCK_50 ) begin

        i <= i + 1'b1;


        if ( i >= 12500 )   begin
            clk <= ~clk;
            i <= 0;
        end

    end

endmodule
```

SV Module: mode_sensor.sv

```
// File Name: mode_sensor.sv
// Purpose: Drives the mode of the system; determines which sensor is on.
// Authors: Cole Raschpichler & Paul Wiebe
// Date: April 7th, 2017

module mode_sensor ( input logic [7:0] sensor,
                            input logic clk, clk_1sec, valid, pass_valid,
                            input logic [4:0] num,
                            input logic [15:0] passcode,
                            output logic led_bar[10:0],
                            output logic [10:0] led_bar_send,
                            output logic spkr, pass_enable,
                            output logic [3:0] mode_send );

    enum logic [3:0] { disarmed, armed, home, alarmed, disarmed_Req_armed,
disarmed_Req_home, home_Req_disarmed, home_Req_armed, armed_Req_disarmed,
alarmed_Req_disarmed }
        state;

    logic [3:0] mode, mode_next;

    initial begin
        mode = disarmed;
    end

    always_comb begin
        mode_send = mode;
        led_bar_send[0] = led_bar[0];
        led_bar_send[1] = led_bar[1];
        led_bar_send[2] = led_bar[2];
        led_bar_send[3] = led_bar[3];
        led_bar_send[4] = led_bar[4];
        led_bar_send[5] = led_bar[5];
        led_bar_send[6] = led_bar[6];
        led_bar_send[7] = led_bar[7];
        led_bar_send[8] = led_bar[8];
        led_bar_send[9] = led_bar[9];
        led_bar_send[10] = led_bar[10];

        led_bar[0] = !sensor[0] || ( led_bar[0] && (( mode == armed ) || ( mode ==
alarmed )));
        led_bar[1] = !sensor[1] || ( led_bar[1] && (( mode == armed ) || ( mode ==
alarmed )));
        led_bar[2] = !sensor[2] || ( led_bar[2] && (( mode == armed ) || ( mode ==
alarmed )));

        led_bar[3] = !sensor[3] || ( led_bar[3] && (( mode == armed ) || ( mode ==
home ) || ( mode == alarmed )));
        led_bar[4] = !sensor[4] || ( led_bar[4] && (( mode == armed ) || ( mode ==
home ) || ( mode == alarmed )));
        led_bar[5] = !sensor[5] || ( led_bar[5] && (( mode == armed ) || ( mode ==
home ) || ( mode == alarmed )));
        led_bar[6] = !sensor[6] || ( led_bar[6] && (( mode == armed ) || ( mode ==
home ) || ( mode == alarmed )));
        led_bar[7] = !sensor[7] || ( led_bar[7] && (( mode == armed ) || ( mode ==
home ) || ( mode == alarmed )));

    spkr = clk && clk_1sec && ((mode == alarmed)|| (mode == alarmed_Req_disarmed));

        unique case( mode )
            disarmed:   begin
            led_bar[8] = 1'b0;
            led_bar[9] = 1'b0;
            led_bar[10] = 1'b0;
            pass_enable = 1'b0;
            end

            disarmed_Req_home:  begin
            led_bar[8] = 1'b0;
            led_bar[9] = 1'b1;
            led_bar[10] = 1'b0;
            pass_enable = 1'b1;
            end

            disarmed_Req_armed: begin
            led_bar[8] = 1'b0;
            led_bar[9] = 1'b1;
            led_bar[10] = 1'b0;
            pass_enable = 1'b1;
            end

            home:   begin
            led_bar[8] = 1'b1;
            led_bar[9] = 1'b0;
            led_bar[10] = 1'b0;
            pass_enable = 1'b0;
            end

            home_Req_disarmed:  begin
            led_bar[8] = 1'b0;
            led_bar[9] = 1'b1;
            led_bar[10] = 1'b0;
            pass_enable = 1'b1;
            end

            home_Req_armed: begin
            led_bar[8] = 1'b0;
            led_bar[9] = 1'b1;
            led_bar[10] = 1'b0;
            pass_enable = 1'b1;
            end

            armed:  begin
            led_bar[8] = 1'b1;
            led_bar[9] = 1'b1;
            led_bar[10] = 1'b0;
            pass_enable = 1'b0;
            end

            armed_Req_disarmed: begin
            led_bar[8] = 1'b0;
            led_bar[9] = 1'b1;
            led_bar[10] = 1'b0;
            pass_enable = 1'b1;
            end

            alarmed:    begin
            led_bar[8] = 1'b0;
            led_bar[9] = 1'b0;
            led_bar[10] = 1'b1;
            pass_enable = 1'b0;
            end

            alarmed_Req_disarmed:   begin
            led_bar[8] = 1'b0;
            led_bar[9] = 1'b1;
            led_bar[10] = 1'b0;
            pass_enable = 1'b1;
            end

        endcase
    end

always_comb begin

    unique case (mode)
        disarmed:begin if((num == 5'b1_1011) && valid && ( sensor ==? 8'b1111_1xxx ))
                        mode_next = disarmed_Req_home;
                    else if ((num == 5'b1_1100) && valid && ( sensor ==? 8'b1111_1111
))
                        mode_next = disarmed_Req_armed;
                    else
                        mode_next = disarmed;
                    end

        disarmed_Req_home:
                    begin if((num == 5'b1_1101) && valid)
                        mode_next = disarmed;
                    else if( pass_valid && ( sensor ==? 8'b1111_1xxx ))
```

```
                        mode_next = home;
                    else
                        mode_next = disarmed_Req_home;
                    end

        disarmed_Req_armed:
                    begin if((num == 5'b1_1101) && valid)
                        mode_next = disarmed;
                    else if( pass_valid && ( sensor == 8'b1111_1111 ))
                        mode_next = armed;
                    else
                        mode_next = disarmed_Req_armed;
                    end

        home:       begin if((num == 5'b1_1101) && valid)
                        mode_next = home_Req_disarmed;
                    else if((num == 5'b1_1100) && valid && ( sensor ==? 8'b1111_1111
))
                        mode_next = home_Req_armed;
                    else if (sensor !=? 8'b1111_1xxx)
                        mode_next = alarmed;
                    else
                        mode_next = home;
                    end

        home_Req_disarmed:
                    begin if((num == 5'b1_1011) && valid)
                        mode_next = home;
                    else if (sensor !=? 8'b1111_1xxx)
                        mode_next = alarmed;
                    else if( pass_valid )
                        mode_next = disarmed;
                    else
                        mode_next = home_Req_disarmed;
                    end

        home_Req_armed:
                    begin if((num == 5'b1_1011) && valid)
                        mode_next = home;
                    else if (sensor !=? 8'b1111_1xxx)
                        mode_next = alarmed;
                    else if( pass_valid && ( sensor == 8'b1111_1111 ))
                        mode_next = armed;
                    else
                        mode_next = home_Req_armed;
                    end

        armed:  begin if((num == 5'b1_1101) && valid)
                        mode_next = armed_Req_disarmed;
                    else if (sensor != 8'b1111_1111)
                        mode_next = alarmed;
                    else
                        mode_next = armed;
                    end

        armed_Req_disarmed:
                    begin if((num == 5'b1_1100) && valid)
                        mode_next = armed;
                    else if (sensor != 8'b1111_1111)
                        mode_next = alarmed;
                    else if( pass_valid)
                        mode_next = disarmed;
                    else
```

```
                        mode_next = armed_Req_disarmed;
                    end

        alarmed:    begin if((num == 5'b1_1101) && valid)
                        mode_next = alarmed_Req_disarmed;
                    else
                        mode_next = alarmed;
                    end

        alarmed_Req_disarmed:
                    begin if( pass_valid)
                        mode_next = disarmed;
                    else
                        mode_next = alarmed_Req_disarmed;
                    end


    endcase
end

always_ff@( posedge clk ) begin
    mode <= mode_next;
end

endmodule
```

SV Module: spimaster.sv

```
// File Name: spimaster.sv
// Purpose: SPI master to drive OLED display.
// Authors: Cole Raschpichler & Paul Wiebe
// Date: April 7th, 2017

module spimaster
  #( N=16 )
   ( input logic [31:0] writedata, // Avalon MM bus
     output logic [31:0] readdata,
     input logic read, write,
      input logic temp_input,

     output logic sclk, mosi, csn, dcn, resetn, // SPI master

     input logic reset, clk ) ;

   logic [31:0] data, data_next ;     // data register
   logic [4:0] i, i_next, j, j_next ; // bit/delay counters

   // output FF inputs
   logic sclk_next, mosi_next, csn_next, dcn_next ;

   // registers
   always_ff@(posedge clk) begin
      data <= data_next ;       // data register

      sclk <= sclk_next ;       // SPI port

      mosi = mosi_next;
      dcn = dcn_next;
      csn = csn_next;

      i <= i_next ;             // bit/clock counts
      j <= j_next ;
   end
   // connect master reset to SPI reset
   assign resetn = !reset ;

   // status register: idle is LS bit
   assign readdata = {31'b0,csn} ;

   // combinational logic
   always_comb begin
      data_next = write ? writedata : data ;

      // i=7...0
      i_next = write ? 5'd7 : ( j==N/2-1 && sclk ) ? i-1 : i ;
      // sclk=0..1
      sclk_next = write ? '0 : ( j==N/2-1 ) ? ~sclk : sclk ;
      // j=0...N/2-1
      j_next = write ? '0 : ( j==N/2-1 ) ? '0 : j+1 ;

      mosi_next = data_next[ i_next ] ;
      csn_next = !resetn ? 1 : write ? '0 : ( !i && j==N/2-1 && sclk ) ? 5'd1 : csn ;
      dcn_next = ~data_next[ 8 ] ;
   end

endmodule
```

SV Module: passcode.sv

```
// File Name: passcode.sv
// Purpose: Passcode to get from one mode to the next.
// Authors: Cole Raschpichler & Paul Wiebe
// Date: April 7th, 2017

module passcode ( input logic [4:0] num,
                        input logic valid, pass_enable, clk,
                        output logic [15:0] passcode, temp_send,
                        output logic pass_valid );

    logic [15:0] password = 16'b0001_0010_0011_0100;
    logic [15:0] temp = 16'b1111_1111_1111_1111;


    always_ff@( posedge valid ) begin
        if( num[3:0] <= 4'b1001) begin
            temp = (temp << 3'b100) | num[3:0];
        end

        else if (num[3:0] == 4'b1110) begin
            temp = ((temp >> 3'b100) | (16'b1111_0000_0000_0000));
        end

        else if (num[3:0] == 4'b1111) begin
            temp = 16'b1111_1111_1111_1111;
        end

        if (temp == password) begin
            pass_valid = 1'b1;
            temp = 16'b1111_1111_1111_1111;
        end

        else if (temp[15:12] != 4'b1111) begin
            temp = 16'b1111_1111_1111_1111;
            pass_valid = 1'b0;
        end
        else begin
            pass_valid = 1'b0;
        end
        temp_send = temp;
    end

endmodule
```

SV Module: alarm_generator.sv

```
// File Name: alarm_generator.sv
// Purpose: Generates a single tone alarm through the speaker.
// Authors: Cole Raschpichler & Paul Wiebe
// Date: April 7th, 2017

module alarm_generator ( input logic clk,
                                  output logic clk_1sec );

    logic [32:0] count;

    always_ff @(posedge clk) begin
        count <= count + 1;

        if ( count >= 100 ) begin
            clk_1sec <= ~clk_1sec;
            count <= 0 ;
        end

    end

endmodule
```

SV Module: kpdecode.sv

```
// File Name: kpdecode.sv
// Purpose: Converts two 4-bit inputs from the keypad (row and column) into
//              a 4-bit number for the 7-segment LED display, and a single bit
//              output to indicate a key has been pressed on the keypad.
// Authors: Cole Raschpichler & Paul Wiebe
// Date: April 7th, 2017

module kpdecode ( input logic [3:0] kpc,
                       input logic clk,
                       input logic [3:0] kpr,
                       output logic [4:0] num,
                        output logic valid );

    logic ready = 1'b1;

    always_comb begin
        case (kpc)
            7: case (kpr)                       // Column 3. 1, 4, 7, or E.
                    7:  num = 5'b1_0001;     // Row 3. Output is 1.
                    11: num = 5'b1_0100;     // Row 2. Output is 4.
                    13: num = 5'b1_0111;     // Row 1. Output is 7.
                    14: num = 5'b1_1110;     // Row 0. Output is E.
                    default: num = 5'b0_0000;
               endcase
            11: case (kpr)                      // Column 2. 2, 5, 8, or 0.
                    7:  num = 5'b1_0010;     // Row 3. Output is 2.
                    11: num = 5'b1_0101;     // Row 2. Output is 5.
                    13: num = 5'b1_1000;     // Row 1. Output is 8.
                    14: num = 5'b1_0000;     // Row 0. Output is 0.
                    default: num = 5'b0_0000;
                endcase
            13: case (kpr)                  // Column 1. 3, 6, 9, or F.
                    7:  num = 5'b1_0011;    // Row 3. Output is 3.
                    11: num = 5'b1_0110;    // Row 2. Output is 6.
                    13: num = 5'b1_1001;    // Row 1. Output is 9.
                    14: num = 5'b1_1111;    // Row 0. Output is F.
                    default: num = 5'b0_0000;
                endcase
            14: case (kpr)                  // Column 0. A, B, C, or D.
                    7:  num = 5'b1_1010;    // Row 3. Output is A.
                    11: num = 5'b1_1011;    // Row 2. Output is B.
                    13: num = 5'b1_1100;    // Row 1. Output is C.
                    14: num = 5'b1_1101;    // Row 0. Output is D.
                    default: num = 5'b0_0000;
                endcase
            default: num = 5'b0_0000;
        endcase
    end

    always_ff@( posedge clk ) begin

        if ( num[4] && ready ) begin
            valid <= 1'b1;
            ready <= 1'b0;
        end

        else if ( !num[4] ) ready <= 1'b1;

        else valid <= 1'b0;

    end

endmodule
```

SV Module: colseq.sv

```
// File Name: colseq.sv
// Purpose: Polls a 4-bit row value from the keypad and switches
//              from each 4-bit column value from the keypad. When a
//              key has been pressed the column value remains constant.
//              The column value is switched on the positive edge of the
//              clock cycle, and is switched to the first column on reset.
// Authors: Cole Raschpichler & Paul Wiebe
// Date: April 7th, 2017

module colseq (input logic [3:0] kpr,
                    output logic [3:0] kpc,
                    input logic clk, reset_n);

    enum logic { scan, hold }
        state;

    initial begin
        kpc = 4'b0111;
    end

        always_ff@(posedge clk) begin
            if ( ! reset_n ) kpc <= 4'b0111;
            else if ( state == scan )  begin
            case (kpc)
                7: kpc = 4'b1011;
                11: kpc = 4'b1101;
                13: kpc = 4'b1110;
                14: kpc = 4'b0111;
            endcase
        end
    end

    always_comb begin
        if ( (! reset_n) || ( kpr == 15) ) state = scan;
        else                         state = hold;
    end

endmodule
```

C File: Display.c

```
// File Name: Display.c
// Purpose: Initialize SSD1331 and fill
//          framebuffer an image
// Author: Cole Raschpichler & Paul Wiebe
// Date: April 7th, 2017

#include <unistd.h>             /* for usleep */

#include "altera_avalon_pio_regs.h"

#include <io.h>

#include "background.h"     /* image to write to display */
#include "mask.h"
#include "prototype.h"
#include <stdio.h>

enum logic { disarmed, armed, home, alarmed, disarmed_Req_armed, disarmed_Req_home,
home_Req_disarmed, home_Req_armed, armed_Req_disarmed, alarmed_Req_disarmed }
        state;

int fill(int enable, int x_coord, int y_coord, int x_size, int y_size, int x_proto,
int y_proto);


int initdata[38] = { 0xAE, 0x81, 0xFF, 0x82, 0xFF, 0x83, 0xFF,
             0x87, 0x06, 0x8A, 0x64, 0x8B, 0x78, 0x8C,
             0x64, 0xA0, 0x73, 0xA1, 0x00, 0xA2, 0x00,
             0xA4, 0xA8, 0x3F, 0xAD, 0x8E, 0xB0, 0x00,
             0xB1, 0x31, 0xB3, 0xF0, 0xBB, 0x3A, 0xBE,
             0x3E, 0x2E, 0xAF } ;

#if 1   // WAS CHANGED FROM 0 TO 1 FOR LAB

#include "system.h"     /* peripheral base addresses */
#define SPIWRITE(x) (*(int*)SPIMASTER_0_BASE) = (x)
#define SPIREAD (*(int*)SPIMASTER_0_BASE)
#define LEDSET(x) (*(int*)PIO_BASE) = (x)

#else

int nv ;
#include <stdio.h>
#define SPIWRITE(x) (printf("%03x%s",x,++nv%16?" ":"\n"))
#define SPIREAD  1
#define LEDSET(x)

#endif
```

```
int main()
{
   // controller initialization

   int i, j, k;//, *address = PIO_BASE;
   int *data = (int *) PIO_0_BASE;
   int mode;
   int temp;
   int num[4];
   int indicator;
   int hold;

   for ( i=0 ; i<38 ; i++ ) {
      SPIWRITE( initdata[i] + 0x100 ) ;
      while ( ( SPIREAD & 1 ) == 0 );
   }



while(1)
{

    hold = ((*data&(0x00000F00)) >> 8 );

    if (hold)
    {
        while(hold)
            hold = ((*data&(0x00000F00)) >> 8 );

        usleep(2000000);

        for ( i=0 ; i<38 ; i++ ) {
           SPIWRITE( initdata[i] + 0x100 ) ;
           while ( ( SPIREAD & 1 ) == 0 );
        }
    }
    while(hold)
        hold = ((*data&(0x00000F00)) >> 8 );

    //   printf("%i, ", *data);

   mode = (((*data)&(0xF0000000)) >> 28 );
   if(mode <= 3)
       indicator = mode;
   temp = ((*data&(0x0FFFF000)) >> 12 );

   printf("%x, %x, %x, %x      ", temp, mode, indicator, hold);



   for ( j=0 ; j<96 ; j++ )
      for ( i=0 ; i<64 ; i++ )
      {
     SPIWRITE( (((background[i][j] | mask[i][j]) * (1 + (15*(0)))) >> 8) & 0xFF )   ;
/* MS byte */
     while ( ( SPIREAD & 1 ) == 0 );
     SPIWRITE( ((background[i][j] | mask[i][j]) * (1 + (15*(0))) & 0xFF)) ; /* LS byte
*/
     while ( ( SPIREAD & 1 ) == 0 );
      }
   // display sensor numbers
   fill(((*data)&(0x00000001)), 53, 32, 10, 7, 0, 0);
   fill(((*data)&(0x00000002)), 53, 40, 10, 7, 0, 1);
   fill(((*data)&(0x00000004)), 53, 48, 10, 7, 0, 2);
   fill(((*data)&(0x00000008)), 53, 56, 10, 7, 0, 3);
   fill(((*data)&(0x00000010)), 53, 64, 10, 7, 0, 4);
   fill(((*data)&(0x00000020)), 53, 72, 10, 7, 0, 5);
   fill(((*data)&(0x00000040)), 53, 80, 10, 7, 0, 6);
   fill(((*data)&(0x00000080)), 53, 88, 10, 7, 0, 7);


   if(mode > 3)
   {
       // display "enter Code" and code
       fill((mode > 3), 39, 2, 10, 63, 3, 0);
       num[3] = (temp & 0xF000) >> 12;
       num[2] = (temp & 0x0F00) >> 8;
       num[1] = (temp & 0x00F0) >> 4;
       num[0] = (temp & 0x000F);
       fill((mode > 3) && (num[3] != 0xF), 39, 63, 10, 7, 0, num[3]);
       fill((mode > 3) && (num[2] != 0xF), 39, 71, 10, 7, 0, num[2]);
       fill((mode > 3) && (num[1] != 0xF), 39, 79, 10, 7, 0, num[1]);
       fill((mode > 3) && (num[0] != 0xF), 39, 87, 10, 7, 0, num[0]);
   }
   else
   {
       // display if alarmed
       fill((indicator == 3), 36, 1, 16, 94, 4, 0);     // alarmed
   }

   // display which mode currently in
   fill((indicator == 0), 2, 3, 10, 7, 2, 5);       // disarm
   fill((indicator == 0), 2, 88, 10, 7, 2, 6);      // disarm
   fill((indicator == 2), 13, 3, 10, 7, 2, 5);      // armed
   fill((indicator == 2), 13, 88, 10, 7, 2, 6);     // armed
   fill((indicator == 1), 24, 3, 10, 7, 2, 5);      // home
   fill((indicator == 1), 24, 88, 10, 7, 2, 6);     // home



 //  usleep(1000);

}
   return 0;
}

int fill(int enable, int x_coord, int y_coord, int x_size, int y_size, int x_proto,
int y_proto)
{
    int n, m;
    if (enable)
        for (n = 0 ; n < x_size; n++)
                for ( m = 0 ; m < y_size; m++)
                    mask[x_coord + n][y_coord + m] = prototype[11*x_proto + 0 +
n][8*y_proto + 1 + m];
    else
        for (n = 0 ; n < x_size; n++)
                for ( m = 0 ; m < y_size; m++)
                    mask[x_coord + n][y_coord + m] = 0;
    return(1);
}
```

Appendix B: Demonstration Images

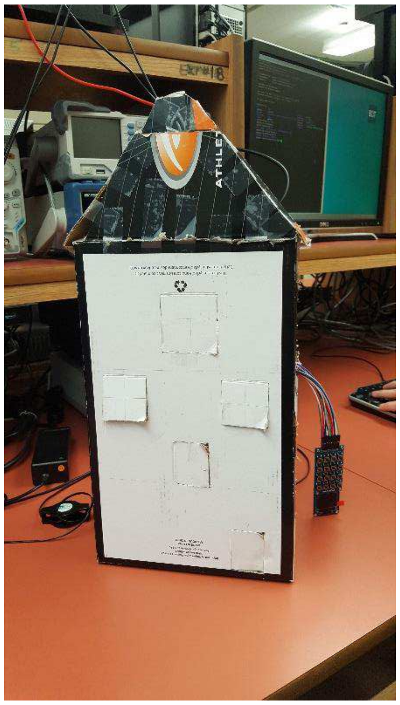
Figure 4: Front of makeshift house.

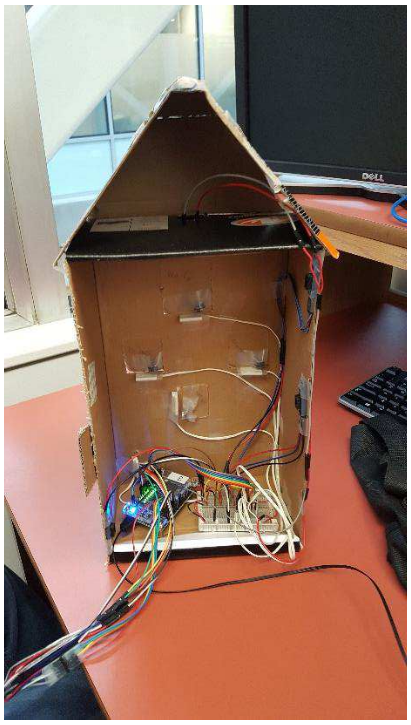
Figure 5: Back of makeshift house.

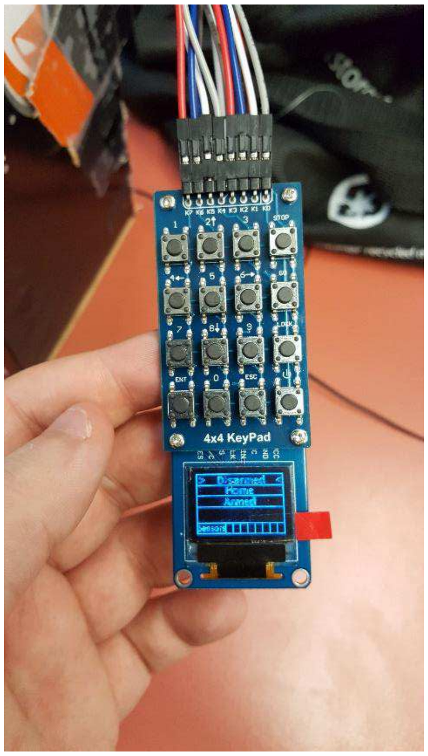

Figure 6: OLED display in disarmed mode.

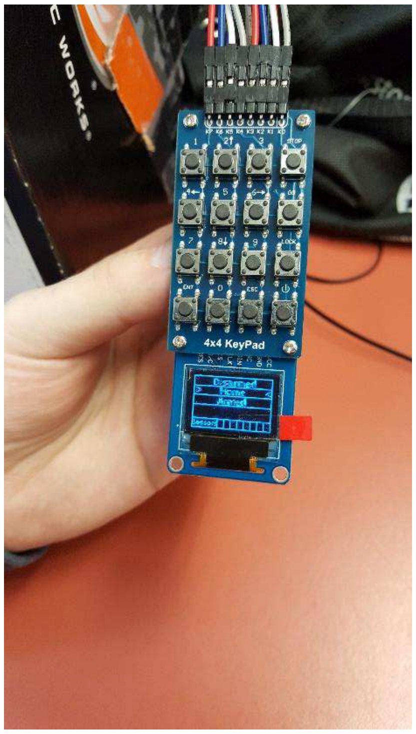

Figure 7: OLED display in home mode.

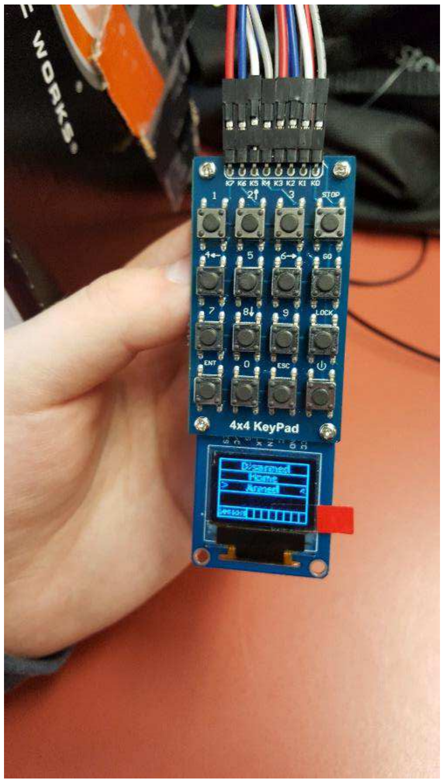

Figure 8: OLED display in armed mode.

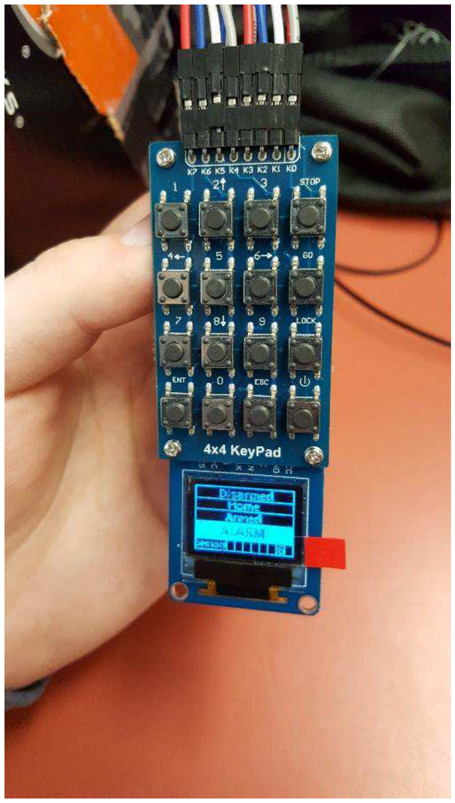

Figure 9: OLED display in alarm mode.

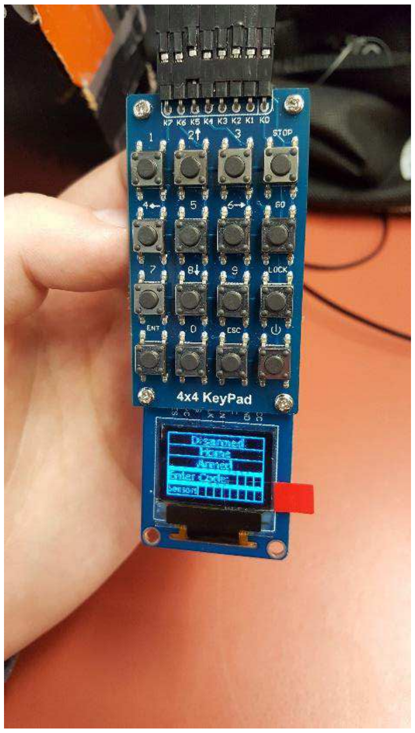

Figure 10: OLED display asking for code to be entered.

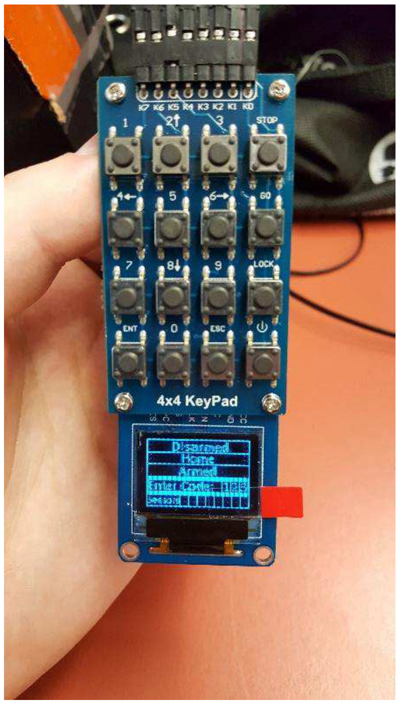

Figure 11: OLED display showing digits 123 entered into the system.

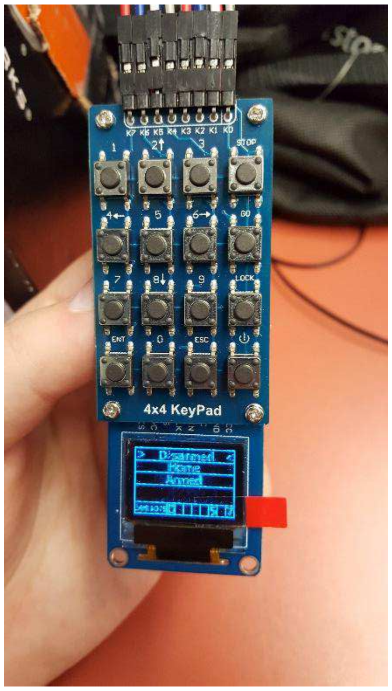

Figure 12: OLED display showing three sensors triggered in disarmed mode.

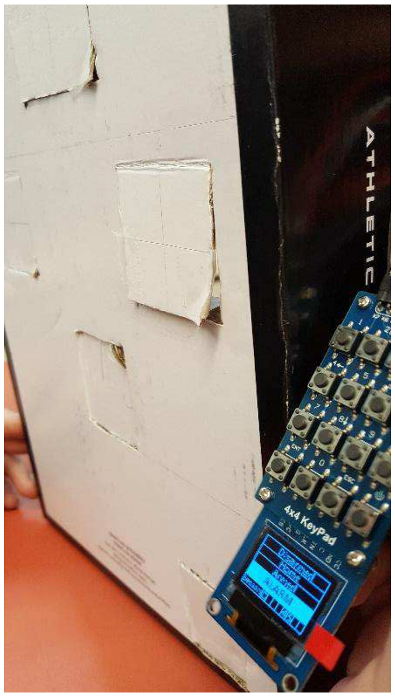

Figure 13: OLED display showing three latched sensors in alarm mode, with the makeshift windows closed to illustrate the latch.