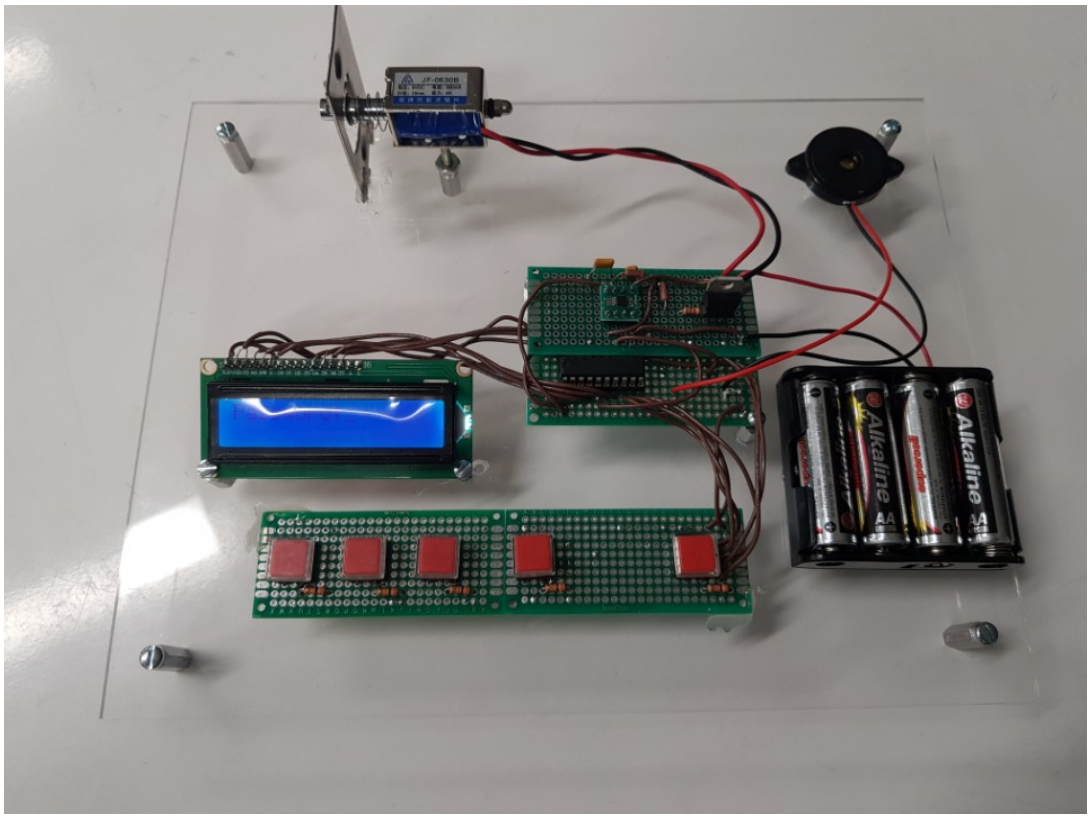
EE53 Project

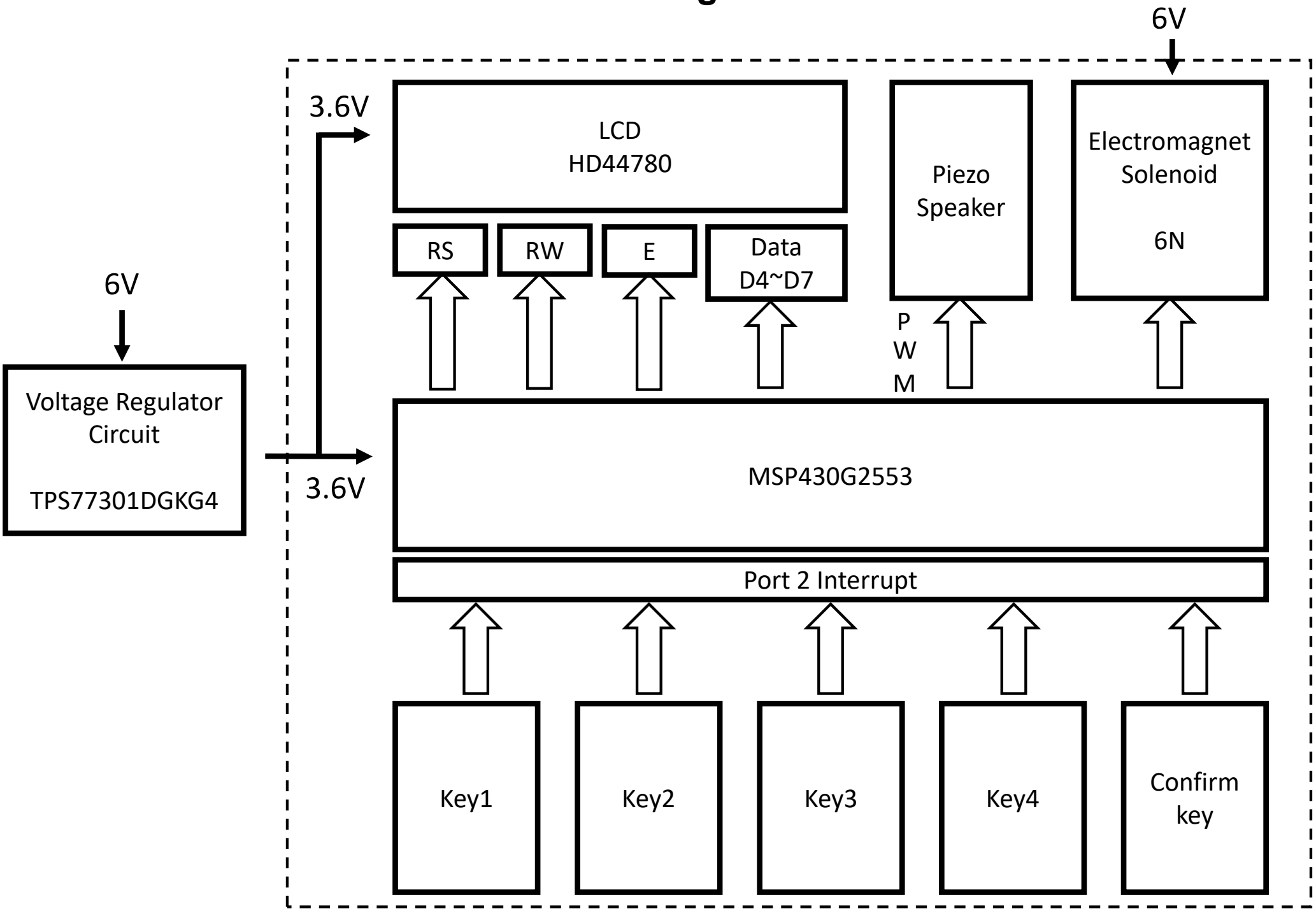


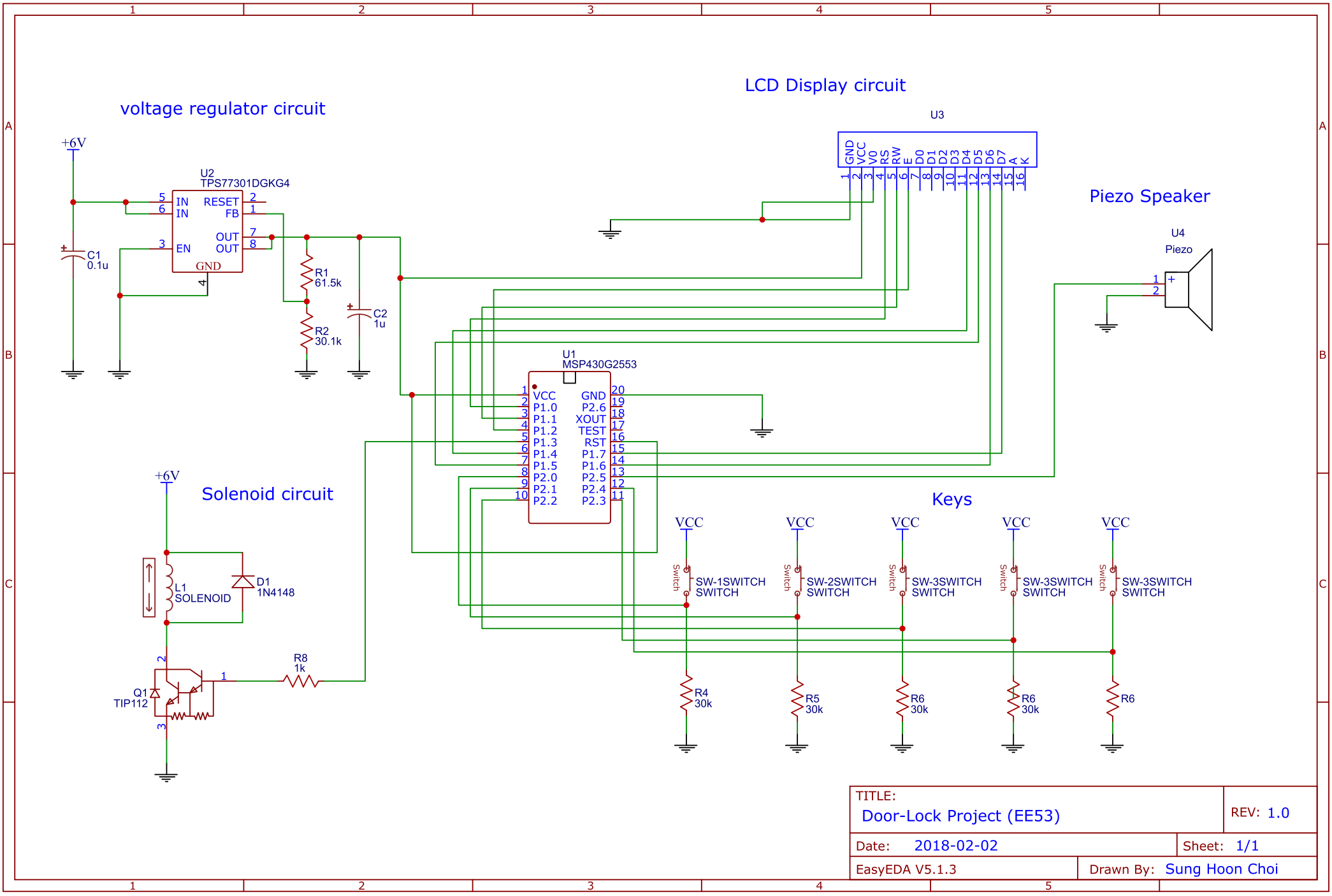
Caltech

Sung Hoon Choi

Table of Contents

Block Diagram




voltage regulator circuit
LCD Display circuit
Piezo Speaker
Solenoid circuit
Keys
+6V
U2
TPS77301DGKG4
IN
IN
EN
RESET
FB
OUT
OUT
GND
C1
0.1u
R1
61.5k
R2
30.1k
C2
1u
U3
GND
VCC
V0
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
A
K
U4
Piezo
U1
MSP430G2553
VCC
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P2.0
P2.1
P2.2
GND
P2.6
XOUT
TEST
RST
P1.7
P1.6
P2.5
P2.4
P2.3
L1
SOLENOID
D1
1N4148
Q1
TIP112
R8
1k
VCC
Switch
SW-1SWITCH
SWITCH
SW-2SWITCH
SW-3SWITCH
R4
30k
R5
30k
R6
30k
TITLE:
Door-Lock Project (EE53)
REV: 1.0
Date: 2018-02-02
Sheet: 1/1
EasyEDA V5.1.3
Drawn By: Sung Hoon Choi

```
//**************************************************************************//
//                                                                          //
//                              lcd_4bit.c                                  //
//                         EE53 Door-Lock Project                           //
//                             Sunghoon Choi                                //
//                                                                          //
//**************************************************************************//
// Description:
//      This file contains the functions necessary for driving the LCD (HD44780)
//      in 4 bit mode.
//
// Table of Contents:
//      init_4bit_lcd           -   Initialize the LCD settings
//      toggle_en_lcd           -   Toggle EN to enable the trigger.
//      send_command_4bit_lcd   -   Send a command to LCD in 4 bit mode
//      send_data_4bit_lcd      -   Send data to LCD in 4 bit mode
//      send_string_4bit_lcd    -   Send string to LCD for display in 4 bit mode
//
// Revision History:
//      02/05/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      02/25/2018  Sunghoon Choi   Errors fixed
//      03/12/2018  Sunghoon Choi   Comments updated

#include "lcd_4bit.h"               //include the header file for LCD in 4-bit mode.
#include "general.h"                //include the header file for general functions and
                                    //constants



// init_4bit_lcd()
//
// Description:
//      Initialize the LCD settings in 4 bit mode.
// Operation:
//      The initialization function first sends the commands 0x33 and 0x32 consecutively
//      to make the LCD run in 4-bit mode. Then, it sends 0x0F to activate the blinking
//      cursor, 0x01 to clear the screen, 0x02 to reset the cursor's position back to
//      its home position(top left). While the system also sends 0x80 to move the cursor
//      back to (row1, column1), which is the top left position, one of the commands
//      0x02 or 0x80 can be removed without causing problems since they actually do the same
//      task.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/06/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      03/12/2018  Sunghoon Choi   Comments updated
void init_4bit_lcd()
{
    send_command_4bit_lcd(0x33);        //Commands 0x33 and 0x32 are sent consecutively
    send_command_4bit_lcd(0x32);        //to operate the LCD in 4 bit mode.
    send_command_4bit_lcd(0x28);        //Operate in 2 lines mode, 5x7 pixels.
    send_command_4bit_lcd(0x0F);        //Turn on the blinking cursor.
    send_command_4bit_lcd(0x01);        //Clear the screen.
    send_command_4bit_lcd(0x02);        //Reset the cursor's position back to its
                                        //home position.(top left)
    send_command_4bit_lcd(0x80);        //Set the cursor's position to (row1,column1).
                                        //In fact, the commands of 0x02 and 0x80 do
                                        //the same task. Thus, one of them can be
                                        //removed without causing problems.
}

// toggle_en_lcd
//
// Description:
```

```
//      Toggle the EN bit to enable the trigger.
//      Toggling EN bit is required between sending the higher nibbles and lower nibbles
//      to enable trigger in 4-bit mode.
// Operation:
//      It sets the EN pin of LCD.
//      Then, it calls the delay function for 10 miliseconds of delay.
//      After the delay, it resets the EN pin of LCD.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/06/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      03/12/2018  Sunghoon Choi   Comments updated
void toggle_en_lcd()
{
    LCD_PORT |= LCD_EN_PIN;             //Set the EN bit
    delay_ms(10);                       //Delay 10 miliseconds
    LCD_PORT &= ~LCD_EN_PIN;            //Reset the EN bit

}

// send_command_4bit_lcd
//
// Description:
//      Send the desired command to LCD in 4-bit mode.
// Operation:
//      It sends the command by sending the upper nibble and lower nibble separately.
//      Notice that in our project, the LCD uses D4~D7 since it operates in 4-bit mode.
//      First, clear (D4~D7) of LCD.
//      Then, reset RS to operate in "send command" mode and reset RW for "write" mode.
//      Next, extract the higher nibble from the input command and send the
//      higher nibble to LCD. Then, give the enable trigger to get ready to send the
//      lower nibble. Clear D4~D7. Extract the lower nibble from the input command and
//      send the lower nibble to LCD. Give the enable trigger again for next commands.
// Argument:
//      cmd        - type       : char
//                 - description: desired command to be sent to the LCD.
// Return Value:
//      None
// Revision Histories:
//      02/06/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      03/12/2018  Sunghoon Choi   Comments updated
void send_command_4bit_lcd(char cmd)
{

    LCD_PORT &= 0x0F;                   //Clear LCD (D4~D7)
    LCD_PORT &= ~LCD_RS_PIN;            //RS=0 (command mode)
    LCD_PORT &= ~LCD_RW_PIN;            //RW=0 (write)
    LCD_PORT |= (cmd & 0xF0);           //send the upper nibble of the command
                                        //(4 bit mode)

    toggle_en_lcd();                    //Give enable trigger.

    LCD_PORT &= 0x0F;                   //Clear LCD (D4~D7)
    LCD_PORT |= ((cmd & 0x0F)<<4);      //send the lower nibble of the command
                                        //(4 bit mode)

    toggle_en_lcd();                    //Give enable trigger.

}


// send_data_4bit_lcd
//
// Description:
```

```
//      Send the desired data to LCD in 4-bit mode.
// Operation:
//      It sends data by sending the upper nibble and lower nibble separately.
//      Notice that in our project, the LCD uses D4~D7 since it operates in 4-bit mode.
//      First, clear (D4~D7) of LCD.
//      Then, set RS to operate in "send data" mode and reset RW for "write" mode.
//      Next, extract the higher nibble from the input data and send the
//      higher nibble to LCD. Then, give the enable trigger to get ready to send the
//      lower nibble. Clear D4~D7. Extract the lower nibble from the input data and
//      send the lower nibble to LCD. Give the enable trigger again for next commands.
// Argument:
//      data       - type       : char
//                 - description: desired data to be sent to the LCD.
// Return Value:
//      None
// Revision Histories:
//      02/06/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      03/12/2018  Sunghoon Choi   Comments updated
void send_data_4bit_lcd(char data)
{
    LCD_PORT &= 0x0F;                   //Clear LCD (D4~D7)
    LCD_PORT |=  LCD_RS_PIN;            //RS=1 (data mode)
    LCD_PORT &= ~LCD_RW_PIN;            //RW=0 (write)
    LCD_PORT |= (data & 0xF0);          //send the upper nibble of data first
                                        //(4 bit mode)
    toggle_en_lcd();                    //Give enable trigger.

    LCD_PORT &= 0x0F;                   //Clear LCD (D4~D7)
    LCD_PORT |= ((data & 0x0F)<<4);     //send the lower nibble of the command
                                        //(4 bit mode)

    toggle_en_lcd();                    //Give enable trigger.
}



// send_string_4bit_lcd
//
// Description:
//      Send the desired string to LCD in 4-bit mode
// Operation:
//      It accepts an array of characters as an argument to be printed on LCD.
//      By iterating through each element(character) of the array(string), it calls
//      send_data_4bit_lcd to send each character to LCD.
// Argument:
//      str        - type       : character array
//                 - description: desired string to be sent to the LCD.
// Return Value:
//      None
// Revision Histories:
//      02/06/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      03/12/2018  Sunghoon Choi   Comments updated
void send_string_4bit_lcd(char * str)
{
    while(*str !=0)                     //iterate through every element(character) of the
                                        //argument array (string).
    {
        send_data_4bit_lcd(*str++);     //Send one character to LCD per every function call
    }
}
```

```
//***********************************************************************************//
//                                                                                   //
//                                  lcd_4bit.h                                       //
//                            EE53 Door-Lock Project                                 //
//                                Sunghoon Choi                                      //
//                                                                                   //
//***********************************************************************************//
// Description:
//      This file contains definitions for lcd_4bit.c
// Revision History:
//      02/05/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      02/24/2018  Sunghoon Choi   Errors fixed
//      03/12/2018  Sunghoon Choi   Comments updated

#ifndef LCD_4BIT_H              //Preprocessor to prevent multiple inclusion
#define LCD_4BIT_H


#include <msp430.h>             //include the functions and constants for TI MSP430

#define LCD_PORT P1OUT          //PORT1 is used for LCD.
#define LCD_RS_PIN BIT0         //PORT1 BIT0: RS
#define LCD_RW_PIN BIT1         //PORT1 BIT1: RW
#define LCD_EN_PIN BIT2         //PORT1 BIT2: EN
                                //PORT1 BIT4~7: D4~D7

void init_4bit_lcd();                   //Initializes the LCD in 4-bit mode
void send_command_4bit_lcd(char cmd);   //Send a command to LCD in 4-bit mode
void send_data_4bit_lcd(char data);     //Send data to LCD in 4-bit mode
void send_string_4bit_lcd(char *str);   //Send string to LCD in 4-bit mode
void toggle_en_lcd();                   //Toggle EN t enable trigger

#endif /* LCD_4BIT_H */
```

```
//*****************************************************************************//
//                                                                             //
//                                  sound.c                                    //
//                           EE53 Door-Lock Project                            //
//                               Sunghoon Choi                                 //
//                                                                             //
//*****************************************************************************//
// Description:
//      This file contains the functions necessary for operating a piezo speaker to
//      generate sounds for the door lock system.
//
// Table of Contents:
//      gen_single_note         -   Generate a sound of desired pitch and duration
//                                  through PWM.
//      gen_warning_sound       -   Generate a warning sound for the case when
//                                  the password does not match
//      gen_open_sound          -   Generate a welcoming sound for the case when
//                                  the password matches and the door opens.
//
// Revision History:
//      02/17/2018  Sunghoon Choi   Created
//      02/20/2018  Sunghoon Choi   Initial Compilation
//      02/24/2018  Sunghoon Choi   Errors fixed
//      03/13/2018  Sunghoon Choi   Comments updated

#include <msp430.h>     //include the functions and constants for TI MSP430
#include "general.h"    //include the header file of general functions and constants
#include "sound.h"      //include the header file for sound generating functions



// gen_single_note()
//
// Description:
//      Generate a single note (sound) with desired pitch and duration with a piezo speaker
//      through PWM control.
// Operation:
//      It uses Pulse Width Modulation to change the frequency of the generated sound.
//      For the maximum amplitude, the duty cycle is set to 50%.
//      After activating the piezo speaker, it delays the desired time in microseconds.
//      Then, it deactivates the piezo speaker and again delays the desired time in
//      microseconds. Let "pitch" be the argument for the amount of time delay.
//
//      Waveform: ____----____----
//      ----  :  ON state  - stay ON state for "pitch" microseconds (50%)
//      ____  :  OFF state - stay OFF state for "pitch"  microseconds (50%)
//
//      Frequency of the wave = 100000/(2*"pitch")
//
//      By repeating this waveform cycle several times through a For loop,
//      it controls the duration of the sound.
//
// Argument:
//      pitch      - type       : Integer
//                 - description: Desired pitch of the sound.
//                                The frequency of the sound is 1/(2*pitch)
//      duration   - type       : Integer
//                 - description: Desired time duration for each note (each sound)
// Return Value:
//      None
// Revision Histories:
//      02/17/2018  Sunghoon Choi   Created
//      02/20/2018  Sunghoon Choi   Initial Compilation
//      03/13/2018  Sunghoon Choi   Comments updated
void gen_single_note(int pitch, int duration)
{
    int i;
    for(i = 1; i<=duration; i++)        //Repeats each waveform cycle for the desired
```

```
                                        //duration of the sound.
    {
        P2OUT |= BIT5;                  //Waveform: ON state
        delay_us(pitch);                //ON state for "pitch" microseconds
        P2OUT &= ~BIT5;                 //Waveform: OFF state
        delay_us(pitch);                //OFF state for "pitch" microseconds
                                        //Duty Cycle = 50%
    }
}


// gen_warning_sound()
//
// Description:
//      Generate a warning sound for the case when the password does not match.
// Operation:
//      It generates a pair of notes twice by calling the gen_single_note four times.
//      The frequencies of the warning sound is
//      "555Hz-714Hz-555Hz-714Hz"
//      gen_single_note(90,200) - Frequency = 100000/(2*90) = 555Hz
//      gen_single_note(70,200) - Frequency = 100000/(2*70) = 714Hz
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/17/2018  Sunghoon Choi   Created
//      02/20/2018  Sunghoon Choi   Initial Compilation
//      03/13/2018  Sunghoon Choi   Comments updated
void gen_warning_sound()
{
    gen_single_note(90,200);        //100000/(2*90) = 555Hz
    gen_single_note(70,200);        //100000/(2*70) = 714Hz
    gen_single_note(90,200);        //100000/(2*90) = 555Hz
    gen_single_note(70,200);        //100000/(2*70) = 714Hz


}

// gen_open_sound()
//
// Description:
//      Generate a welcoming sound for the case when the password matches and the door
//      opens.
// Operation:
//      It generates three notes (sounds) in series as a welcoming message.
//      "500Hz-555Hz-625Hz"
//      gen_single_note(100,300) - Frequency = 100000/(2*100) = 500Hz
//      gen_single_note(90,300)  - Frequency = 100000/(2*90)  = 555Hz
//      gen_single_note(80,300)  - Frequency = 100000/(2*80)  = 625Hz
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/17/2018  Sunghoon Choi   Created
//      02/20/2018  Sunghoon Choi   Initial Compilation
//      03/13/2018  Sunghoon Choi   Comments updated
void gen_open_sound()
{
    gen_single_note(100,300);       //100000/(2*100) = 500Hz
    gen_single_note(90,300);        //100000/(2*90)  = 555Hz
    gen_single_note(80,300);        //100000/(2*80)  = 625Hz
}
```

```
//*****************************************************************************//
//                                                                             //
//                                  sound.h                                    //
//                           EE53 Door-Lock Project                            //
//                               Sunghoon Choi                                 //
//                                                                             //
//*****************************************************************************//
// Description:
//      This file contains definitions for sound.c
// Revision History:
//      02/17/2018  Sunghoon Choi   Created
//      02/20/2018  Sunghoon Choi   Initial Compilation
//      02/24/2018  Sunghoon Choi   Errors fixed
//      03/13/2018  Sunghoon Choi   Comments updated

#ifndef SOUND_H_                //Preprocessor to prevent multiple inclusion
#define SOUND_H_

void gen_single_note(int pitch, int duration); //Generate a sound of desired pitch and
                                               //duration through PWM.
void gen_warning_sound(void);                  //Generate a warning sound for the case when
                                               //the password does not match.
void gen_open_sound(void);                     //Generate a welcoming sound for the case
                                               //when the password matches and the door
                                               //opens.

#endif /* SOUND_H_ */
```

```
//******************************************************************************//
//                                                                              //
//                                  general.c                                   //
//                           EE53 Door-Lock Project                             //
//                               Sunghoon Choi                                  //
//                                                                              //
//******************************************************************************//
// Description:
//      This file contains general functions necessary for the operation of the
//      door-lock system.
// Table of Contents:
//      check_pw            -   Check if the password matches
//      delay_ms            -   Delay the desired time in miliseconds.
//      delay_us            -   Delay the desired time in microseconds.
//
// Revision History:
//      02/08/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      02/22/2018  Sunghoon Choi   Errors fixed
//      03/06/2018  Sunghoon Choi   Comments updated


#include <msp430.h>         //include the functions and constants for TI MSP430
#include "lcd_4bit.h"       //include the header file for LCD in 4-bit mode.
#include "fsm.h"            //include the header file for finite state machine
#include "general.h"        //include the header file for general functions and constants


// check_pw()
//
// Description:
//      Check if the password matches.
// Operation:
//      InputPW is an array that contains the 4 integers typed by the user.
//      CorrectPW is an array that contains the 4 integers of the correct password.
//      check_pw compares each element of the two arrays and returns 1 if all elements
//      match and returns 0 otherwise.
// Argument:
//      None
// Return Value:
//      1   -   If the password matches.
//      0   -   If the password does not match.
// Revision Histories:
//      02/09/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      03/06/2018  Sunghoon Choi   Comments updated
int check_pw()
{
    if(CorrectPW[0]==InputPW[0] && CorrectPW[1]==InputPW[1] && CorrectPW[2]==InputPW[2] &&
    CorrectPW[3]==InputPW[3])
    {
        return 1;           //If every element of InputPW and CorrectPW matches, return 1.
    }
    else
    {
        return 0;           //Otherwise, return 0.
    }
}

// delay_ms
//
// Description:
//      Delay time in miliseconds.
//      Delay functions are used in various situations, such as driving the LCD and
//      generating sounds by PWM control.
// Operation:
//      Delay miliseconds using the delay_cycles function of MSP430.
//      MSP430G2553 operates with 1MHz clock in default.
```

```
//      (1/1000000)*1000 = 1/1000 sec = 1 milisecond.
// Argument:
//      milisecond - type       : integer
//                 - description: argument to set the amount of time delay in miliseconds.
// Return Value:
//      None
// Revision Histories:
//      02/09/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      03/06/2018  Sunghoon Choi   Comments updated
void delay_ms(int milisecond)
{
    int i;
    for(i=1; i<=milisecond; i++)
    {
    __delay_cycles(1000);              //MSP430G2553 operates with 1MHz clock in default.
                                       //(1/1000000)*1000 = 1/1000 sec = 1 milisecond.
    }

}


// delay_us
//
// Description:
//      Delay time in microseconds.
//      Delay functions are used in various situations, such as driving the LCD and
//      generating sounds by PWM control.
// Operation:
//      Delay microseconds using the delay_cycles function of MSP430.
//      MSP430G2553 operates with 1MHz clock in default.
//      (1/1000000)*1 = 10^(-6) sec = 1 microsecond.
// Argument:
//      microsecond - type      : integer
//                  - description: argument to set the amount of time delay in microseconds.
// Return Value:
//      None
// Revision Histories:
//      02/09/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      03/06/2018  Sunghoon Choi   Comments updated

void delay_us(int microsecond)
{
    int i;
    for(i=1; i<=microsecond; i++)
    {
        __delay_cycles(1);             //MSP430G2553 operates with 1MHz clock in default.
                                       //(1/1000000)*1 = 10^(-6) sec = 1 microsecond.
    }
}
```

```
//*****************************************************************************//
//                                                                             //
//                                general.h                                    //
//                           EE53 Door-Lock Project                            //
//                               Sunghoon Choi                                 //
//                                                                             //
//*****************************************************************************//
// Description:
//      This file contains definitions for fsm.c
// Revision History:
//      02/08/2018  Sunghoon Choi   Created
//      02/13/2018  Sunghoon Choi   Initial Compilation
//      02/22/2018  Sunghoon Choi   Errors fixed
//      03/06/2018  Sunghoon Choi   Comments updated

#ifndef GENERAL_H_              //Preprocessor to prevent multiple inclusion
#define GENERAL_H_


#include <msp430.h>             //include the functions and constants for TI MSP430
#include "lcd_4bit.h"           //include the header file for LCD in 4-bit mode.


#define DEFAULT_PW_DIGIT1    3  //First digit of the default password of the door lock
#define DEFAULT_PW_DIGIT2    4  //Second digit of the default password of the door lock
#define DEFAULT_PW_DIGIT3    1  //Third digit of the default password of the door lock
#define DEFAULT_PW_DIGIT4    2  //Fourth digit of the default password of the door lock

#define DEFAULT_INPUT_DIGIT1 5  //First digit of the input buffer in default
#define DEFAULT_INPUT_DIGIT2 5  //Second digit of the input buffer in default
#define DEFAULT_INPUT_DIGIT3 5  //Third digit of the input buffer in default
#define DEFAULT_INPUT_DIGIT4 5  //Fourth digit of the input buffer in default



extern int CorrectPW[4];        //An array(buffer) that contains the correct password
extern int InputPW[4];          //An array(buffer) that contains the password typedef
                                //by the user
extern int CorrectPWIndex;      //Index used for taking values from CorrectPW buffer
extern int InputPWIndex;        //Index used for taking values from InputPW buffer

int check_pw();                 //Checks if the password matches
void delay_ms(int milisecond);  //Delays time in miliseconds.
void delay_us(int microsecond); //Delays time in microseconds.


#endif /* GENERAL_H_ */
```

```
//*****************************************************************************//
//                                                                             //
//                              fsm_transition.c                               //
//                           EE53 Door-Lock Project                            //
//                               Sunghoon Choi                                 //
//                                                                             //
//*****************************************************************************//
// Description:
//      This file contains transition functions necessary for the finite state machine of
//      the door-lock system.
// Table of Contents:
//      FSM_save_digit_one      -   save the input from Key1 in the input array
//      FSM_save_digit_two      -   save the input from Key2 in the input array
//      FSM_save_digit_three    -   save the input from Key3 in the input array
//      FSM_save_digit_four     -   save the input from Key4 in the input array
//      FSM_return_to_start     -   Clear the input array and clear the LCD
//      FSM_do_nothing          -   Do nothing. Necessary for the finite state machine
//      FSM_print_reset_msg     -   Clear the LCD, print reset messeage on LCD,
//                                  and clear the correct PW array
//      FSM_open_door           -   activate the solenoid, generate welcoming sound,
//                                  and print the welcoming message on LCD
//      FSM_close_door          -   Deactivate the solenoid and call FSM_return_to_start
//      FSM_reset_digit_one     -   save the input from Key1 in the reset PW array
//      FSM_reset_digit_two     -   save the input from Key2 in the reset PW array
//      FSM_reset_digit_three   -   save the input from Key3 in the reset PW array
//      FSM_reset_digit_four    -   save the input from Key4 in the reset PW array
//      FSM_alert_wrong_pw      -   Call FSM_return_to_start and generate an alarm sound
//
// Revision History:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      02/27/2018  Sunghoon Choi   Errors fixed
//      03/02/2018  Sunghoon Choi   Comments updated

#include "fsm.h"            //include the header file for finite state machine
#include "general.h"        //include the header file of general functions
#include "sound.h"          //include the header file of audio generating functions


// FSM_save_digit_one
//
// Description:
//      Save the input data from Key1 to accept the password typed by the user.
// Operation:
//      Print the digit '1' on the LCD display by calling send_data_4bit_lcd('1') and
//      store the integer value 1 in the input password array, InputPW. Increment the
//      array's index to wait for the next input digit.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_save_digit_one (void)
{
   send_data_4bit_lcd('1');     //Display '1' on LCD using the 4-bit mode.
   InputPW[InputPWIndex]=1;     //Save the integer data 1 in the input password buffer.
                                //Input password buffer will be compared with
                                //Correct password buffer to figure out whether the
                                //password matches or not.
   InputPWIndex++;              //Increment the index of the buffer to accept next input.
}


// FSM_save_digit_two
//
```

```
// Description:
//      Save the input data from Key2 to accept the password typed by the user.
// Operation:
//      Print the digit '2' on the LCD display by calling send_data_4bit_lcd('2') and
//      store the integer value 2 in the input password array, InputPW. Increment the
//      array's index to wait for the next input digit.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_save_digit_two (void)
{
    send_data_4bit_lcd('2');    //Display '2' on LCD using the 4-bit mode.
    InputPW[InputPWIndex]=2;    //Save the integer data 2 in the input password buffer.
                                //Input password buffer will be compared with
                                //Correct password buffer to figure out whether the
                                //password matches or not.
    InputPWIndex++;             //Increment the index of the buffer to accept next input.
}


// FSM_save_digit_three
//
// Description:
//      Save the input data from Key3 to accept the password typed by the user.
// Operation:
//      Print the digit '3' on the LCD display by calling send_data_4bit_lcd('3') and
//      store the integer value 3 in the input password array, InputPW. Increment the
//      array's index to wait for the next input digit.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_save_digit_three (void)
{
    send_data_4bit_lcd('3');    //Display '3' on LCD using the 4-bit mode.
    InputPW[InputPWIndex]=3;    //Save the integer data 3 in the input password buffer.
                                //Input password buffer will be compared with
                                //Correct password buffer to figure out whether the
                                //password matches or not.
    InputPWIndex++;             //Increment the index of the buffer to accept next input.
}


// FSM_save_digit_four
//
// Description:
//      Save the input data from Key4 to accept the password typed by the user.
// Operation:
//      Print the digit '4' on the LCD display by calling send_data_4bit_lcd('4') and
//      store the integer value 4 in the input password array, InputPW. Increment the
//      array's index to wait for the next input digit.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
```

```
void FSM_save_digit_four (void)
{
    send_data_4bit_lcd('4');    //Display '4' on LCD using the 4-bit mode.
    InputPW[InputPWIndex]= 4;   //Save the integer data 4 in the input password buffer.
                                //Input password buffer will be compared with
                                //Correct password buffer to figure out whether the
                                //password matches or not.

    InputPWIndex++;             //Increment the index of the buffer to accept next input.
}


// FSM_return_to_start
//
// Description:
//      Clear the LCD, clear the input password buffer, and show "Password: " on LCD
//      After this function is called, the state is back to the initial state and ready
//      to accept the password from the first digit.
// Operation:
//      Send the 4-bit mode command to LCD to clear the LCD.
//      Print "Password: " on the LCD by calling the send_string_4bit_lcd function.
//      Clear the input password array and reset the index to accept and store
//      the key inputs anew.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_return_to_start (void)
{
    send_command_4bit_lcd(0x01);        //0x01 is the 4-bit command to clear the LCD.
    send_string_4bit_lcd("Password:");  //Display "Password: " to notify the user to
                                        //type the password.
    int i;
    for(i = 1 ; i<= 4; i++)
    {
        InputPW[i] = 0;                 //Clear every element of the input password buffer.
    }
    InputPWIndex = 0;                   //Reset the index of the array to accept the
                                        //input keys anew.
}


// FSM_do_nothing
//
// Description:
//      Do nothing.
//      This function is necessary for the operation of the finite state machine.
// Operation:
//      Do nothing.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_do_nothing (void)
{
                                        //Do nothing
}


// FSM_print_reset_msg
```

```
//
// Description:
//      Print the reset message on the LCD to notify the user that the door-lock is now
//      in reset mode and ready to accept a new password.
// Operation:
//      Clear the Correct Password buffer to accept the new password.
//      Clear the LCD and print the string of "reset: " by calling
//      send_string_4bit_lcd.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_print_reset_msg (void)
{
    int i;
    for(i = 1 ; i<= 4; i++)
    {
        CorrectPW[i] = 0;           //clear the correct PW data to accept the new
                                    //password
    }
    CorrectPWIndex = 0;             //Initialize the index

    send_command_4bit_lcd(0x01);    //0x01 is the 4-bit command to clear the LCD.
    send_string_4bit_lcd("reset:"); //Print "reset: " on LCD to notify the user to
                                    //type the desired new password.
}


// FSM_open_door
//
// Description:
//      Open the door by activating the solenoid and print the welcome message on the LCD.
//      Generate a welcoming sound to notify the user that the password matches and the
//      door opens.
// Operation:
//      Set BIT3 of Port1 to activate the solenoid. Clear the LCD by sending a 4-bit
//      command(0x01) and print "welcome" by calling send_string_4bit_lcd.
//      Generate a welcoming sound by calling gen_open_sound.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_open_door (void)
{
    P1OUT |= BIT3;                      //Set BIT3 of Port1 to activate the solenoid
    send_command_4bit_lcd(0x01);        //0x01 is the 4-bit command to clear the LCD.
    send_string_4bit_lcd("welcome");    //Print "welcome" on the LCD.
    gen_open_sound();                   //Generate a welcoming sound to notify the
    user
                                        //that the password matches and the door opens.
}


// FSM_close_door
//
// Description:
//      Close the door by deactivating the solenoid and return to the initial state.
// Operation:
//      Reset BIT3 of Port1 to deactivate the solenoid.
//      Call FSM_return_to_start to print "Password: " on LCD and clear the input buffer.
```

```
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_close_door (void)
{

    P1OUT &= ~BIT3;                     //Reset BIT3 of Port1 to deactivate the solenoid.
    FSM_return_to_start();              //Clear the input buffer and print "Password: "
                                        //on LCD to notify the user to type the password.
    InputPWIndex = 0;                   //Reset the index of input buffer to accept the
                                        //passowrd anew.
}


// FSM_reset_digit_one
//
// Description:
//      Reset the password by storing the input from Key1.
//      This function only resets the password when the input comes from Key1.
//      Display the corresponding number (1) on LCD for the user.
// Operation:
//      Store the integer value 1 in the Correct password buffer.
//      Send the data of '1' to the LCD using the 4-bit mode.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_reset_digit_one (void)
{
   send_data_4bit_lcd('1');          //Display '1' on LCD by sending '1' using 4-bit mode.
   CorrectPW[CorrectPWIndex]=1;      //Store the integer value 1 in the Correct Password
                                     //buffer to save the information of the new password.

   CorrectPWIndex++;                 //Increment the index to accept next digit of the
                                     //new password.
}


// FSM_reset_digit_two
//
// Description:
//      Reset the password by storing the input from Key2.
//      This function only resets the password when the input comes from Key2.
//      Display the corresponding number (2) on LCD for the user.
// Operation:
//      Store the integer value 2 in the Correct password buffer.
//      Send the data of '2' to the LCD using the 4-bit mode.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_reset_digit_two (void)
{
   send_data_4bit_lcd('2');          //Display '2' on LCD by sending '2' using 4-bit mode.
   CorrectPW[CorrectPWIndex]=2;      //Store the integer value 2 in the Correct Password
```

```
                                    //buffer to save the information of the new password.

   CorrectPWIndex++;                //Increment the index to accept next digit of the
                                    //new password.
}


// FSM_reset_digit_three
//
// Description:
//      Reset the password by storing the input from Key3.
//      This function only resets the password when the input comes from Key3.
//      Display the corresponding number (3) on LCD for the user.
// Operation:
//      Store the integer value 3 in the Correct password buffer.
//      Send the data of '3' to the LCD using the 4-bit mode.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_reset_digit_three (void)
{
   send_data_4bit_lcd('3');         //Display '3' on LCD by sending '3' using 4-bit mode.
   CorrectPW[CorrectPWIndex]=3;     //Store the integer value 3 in the Correct Password
                                    //buffer to save the information of the new password.

   CorrectPWIndex++;                //Increment the index to accept next digit of the
                                    //new password.
}


// FSM_reset_digit_four
//
// Description:
//      Reset the password by storing the input from Key4.
//      This function only resets the password when the input comes from Key4.
//      Display the corresponding number (4) on LCD for the user.
// Operation:
//      Store the integer value 4 in the Correct password buffer.
//      Send the data of '4' to the LCD using the 4-bit mode.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_reset_digit_four (void)
{
   send_data_4bit_lcd('4');         //Display '4' on LCD by sending '4' using 4-bit mode.
   CorrectPW[CorrectPWIndex]=4;     //Store the integer value 4 in the Correct Password
                                    //buffer to save the information of the new password.
   CorrectPWIndex++;                //Increment the index to accept next digit of the
                                    //new password.
}

// FSM_alert_wrong_pw
//
// Description:
//      Generate an alarm sound and go back to the initial state.
//      This function is called when the user types an incorrect password and
//      presses the confirm key.
// Operation:
```

```c
//      Generate an alarm sound by calling gen_warning_sound.
//      Clear the LCD, print "Password: ", and initialize the input buffer to receive
//      the input anew by calling FSM_return_to_start.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_alert_wrong_pw (void)
{
    FSM_return_to_start();          //Initialize the LCD and input buffer
    gen_warning_sound();            //Generate an alarm sound to notify the user
                                    //that the password does not match.
}
```

```
//******************************************************************************//
//                                                                              //
//                                  fsm.c                                       //
//                           EE53 Door-Lock Project                             //
//                               Sunghoon Choi                                  //
//                                                                              //
//******************************************************************************//
// Description:
//      This file contains state table and event functions necessary for the finite state
//      machine of the door-lock system.
// Table of Contents:
//      FSM_Init                -   Set the current state to the initial state.
//
//      FSM_EVENT_KEY1          -   Do the action corresponding to the current state
//                                  and the event of key1 being pressed.
//      FSM_EVENT_KEY2          -   Do the action corresponding to the current state
//                                  and the event of key2 being pressed.
//      FSM_EVENT_KEY3          -   Do the action corresponding to the current state
//                                  and the event of key3 being pressed.
//      FSM_EVENT_KEY4          -   Do the action corresponding to the current state
//                                  and the event of key4 being pressed.
//      FSM_EVENT_KEY_CONFIRM   -   Do the action corresponding to the current state
//                                  and the event of confirm key being pressed.
//      FSM_EVENT_KEY_RESET     -   Do the action corresponding to the current state
//                                  and the event of reset key (pressing key 1 and key3
//                                  at the same time for 1 second) being pressed.
//      FSM_EVENT_PW_MATCH      -   Do the action corresponding to the current state
//                                  and the event when the user password matches.
//      FSM_EVENT_PW_UNMATCH    -   Do the action corresponding to the current state
//                                  and the event when the user password does not match.
// Revision History:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      02/24/2018  Sunghoon Choi   Errors fixed
//      03/03/2018  Sunghoon Choi   Comments updated

#include "fsm.h"            //include the header file for finite state machine
#include "general.h"        //include the header file of general functions


int  ActState;              //Global variable that indicates the current state of the FSM.


// StateTable
//
// Description:
//  A table that contains the information of functions and next states corresponding to
//  the current state and the event.
//  Each row corresponds to each current state. A pair of columns correspond to the
//  function to be called and the next state. This pair is determined by the event.
//  For example, (FSM_save_digit_one, ST_ACCEPT_SECOND_DIGIT) is a pair.
//  Please notice that the rows and columns are messed when printed on paper since the
//  length of each row exceeds the length of an A4 paper sheet.

const FSM_STATE_TABLE StateTable [NR_STATES][NR_EVENTS] =
{
  FSM_save_digit_one, ST_ACCEPT_SECOND_DIGIT, FSM_save_digit_two, ST_ACCEPT_SECOND_DIGIT,
  FSM_save_digit_three, ST_ACCEPT_SECOND_DIGIT, FSM_save_digit_four, ST_ACCEPT_SECOND_DIGIT,
  FSM_return_to_start, ST_ACCEPT_FIRST_DIGIT, FSM_do_nothing, ST_ACCEPT_FIRST_DIGIT,
  FSM_do_nothing, ST_ACCEPT_FIRST_DIGIT, FSM_print_reset_msg, ST_RESET_FIRST_DIGIT,
  FSM_save_digit_one, ST_ACCEPT_THIRD_DIGIT, FSM_save_digit_two, ST_ACCEPT_THIRD_DIGIT,
  FSM_save_digit_three, ST_ACCEPT_THIRD_DIGIT, FSM_save_digit_four, ST_ACCEPT_THIRD_DIGIT,
  FSM_return_to_start, ST_ACCEPT_FIRST_DIGIT, FSM_do_nothing, ST_ACCEPT_SECOND_DIGIT,
  FSM_do_nothing, ST_ACCEPT_SECOND_DIGIT, FSM_print_reset_msg, ST_RESET_FIRST_DIGIT,
  FSM_save_digit_one, ST_ACCEPT_FOURTH_DIGIT, FSM_save_digit_two, ST_ACCEPT_FOURTH_DIGIT,
  FSM_save_digit_three, ST_ACCEPT_FOURTH_DIGIT, FSM_save_digit_four, ST_ACCEPT_FOURTH_DIGIT,
  FSM_return_to_start, ST_ACCEPT_FIRST_DIGIT, FSM_do_nothing, ST_ACCEPT_THIRD_DIGIT,
  FSM_do_nothing, ST_ACCEPT_THIRD_DIGIT, FSM_print_reset_msg, ST_RESET_FIRST_DIGIT,
```

```
  FSM_save_digit_one, ST_ACCEPT_CONFIRM, FSM_save_digit_two, ST_ACCEPT_CONFIRM,
  FSM_save_digit_three, ST_ACCEPT_CONFIRM, FSM_save_digit_four, ST_ACCEPT_CONFIRM,
  FSM_return_to_start, ST_ACCEPT_FIRST_DIGIT, FSM_do_nothing, ST_ACCEPT_FOURTH_DIGIT,
  FSM_do_nothing, ST_ACCEPT_FOURTH_DIGIT, FSM_print_reset_msg, ST_RESET_FIRST_DIGIT,
  FSM_do_nothing, ST_ACCEPT_CONFIRM, FSM_do_nothing, ST_ACCEPT_CONFIRM, FSM_do_nothing,
  ST_ACCEPT_CONFIRM, FSM_do_nothing, ST_ACCEPT_CONFIRM, FSM_do_nothing, ST_CHECK_PASSWORD,
  FSM_do_nothing, ST_ACCEPT_CONFIRM, FSM_do_nothing, ST_ACCEPT_CONFIRM, FSM_print_reset_msg,
  ST_RESET_FIRST_DIGIT,
  FSM_do_nothing, ST_CHECK_PASSWORD, FSM_do_nothing, ST_CHECK_PASSWORD, FSM_do_nothing,
  ST_CHECK_PASSWORD, FSM_do_nothing, ST_CHECK_PASSWORD, FSM_do_nothing, ST_CHECK_PASSWORD,
  FSM_open_door, ST_DOOR_OPENED, FSM_alert_wrong_pw, ST_ACCEPT_FIRST_DIGIT, FSM_do_nothing,
  ST_CHECK_PASSWORD,
  FSM_do_nothing, ST_DOOR_OPENED, FSM_do_nothing, ST_DOOR_OPENED, FSM_do_nothing,
  ST_DOOR_OPENED, FSM_do_nothing, ST_DOOR_OPENED, FSM_close_door, ST_ACCEPT_FIRST_DIGIT,
  FSM_do_nothing, ST_DOOR_OPENED, FSM_do_nothing, ST_DOOR_OPENED, FSM_do_nothing,
  ST_DOOR_OPENED,
  FSM_reset_digit_one, ST_RESET_SECOND_DIGIT, FSM_reset_digit_two, ST_RESET_SECOND_DIGIT,
  FSM_reset_digit_three, ST_RESET_SECOND_DIGIT, FSM_reset_digit_four, ST_RESET_SECOND_DIGIT,
  FSM_print_reset_msg, ST_RESET_FIRST_DIGIT, FSM_do_nothing, ST_RESET_FIRST_DIGIT,
  FSM_do_nothing, ST_RESET_FIRST_DIGIT, FSM_do_nothing, ST_RESET_FIRST_DIGIT,
  FSM_reset_digit_one, ST_RESET_THIRD_DIGIT, FSM_reset_digit_two, ST_RESET_THIRD_DIGIT,
  FSM_reset_digit_three, ST_RESET_THIRD_DIGIT, FSM_reset_digit_four, ST_RESET_THIRD_DIGIT,
  FSM_print_reset_msg, ST_RESET_FIRST_DIGIT, FSM_do_nothing, ST_RESET_SECOND_DIGIT,
  FSM_do_nothing, ST_RESET_SECOND_DIGIT, FSM_do_nothing, ST_RESET_SECOND_DIGIT,
  FSM_reset_digit_one, ST_RESET_FOURTH_DIGIT, FSM_reset_digit_two, ST_RESET_FOURTH_DIGIT,
  FSM_reset_digit_three, ST_RESET_FOURTH_DIGIT, FSM_reset_digit_four, ST_RESET_FOURTH_DIGIT,
  FSM_print_reset_msg, ST_RESET_FIRST_DIGIT, FSM_do_nothing, ST_RESET_THIRD_DIGIT,
  FSM_do_nothing, ST_RESET_THIRD_DIGIT, FSM_do_nothing, ST_RESET_THIRD_DIGIT,
  FSM_reset_digit_one, ST_RESET_CONFIRM, FSM_reset_digit_two, ST_RESET_CONFIRM,
  FSM_reset_digit_three, ST_RESET_CONFIRM, FSM_reset_digit_four, ST_RESET_CONFIRM,
  FSM_print_reset_msg, ST_RESET_FIRST_DIGIT, FSM_do_nothing, ST_RESET_FOURTH_DIGIT,
  FSM_do_nothing, ST_RESET_FOURTH_DIGIT, FSM_do_nothing, ST_RESET_FOURTH_DIGIT,
  FSM_do_nothing, ST_RESET_CONFIRM, FSM_do_nothing, ST_RESET_CONFIRM, FSM_do_nothing,
  ST_RESET_CONFIRM, FSM_do_nothing, ST_RESET_CONFIRM, FSM_return_to_start,
  ST_ACCEPT_FIRST_DIGIT, FSM_do_nothing, ST_RESET_CONFIRM, FSM_do_nothing, ST_RESET_CONFIRM,
  FSM_do_nothing, ST_RESET_CONFIRM


};


// FSM_Init
//
// Description:
//      Set the current state to the initial state.
// Operation:
//      Set ActState to ST_ACCEPT_FIRST_DIGIT
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_Init (void)
{
  ActState = ST_ACCEPT_FIRST_DIGIT;     //Set the state to the initial state.
}


// FSM_EVENT_KEY1
//
// Description:
//      Do the action corresponding to (current state, EVENT_KEY1) and move to next state
//      using the StateTable. EVENT_KEY1 is the event when key1 is pressed.
// Operation:
//      Call the action function corresponding to the current state and EVENT_KEY1 inside
```

```
//      the StateTable structure. Update ActState to the NextState corresponding to the
//      current state and EVENT_KEY1 inside the StateTable structure.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_EVENT_KEY1 (void)
{

    StateTable[ActState][EVENT_KEY1].ptrFunct();            //call the action function
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY1.

    ActState = StateTable[ActState][EVENT_KEY1].NextState;  //Proceed to the next state
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY1.
}


// FSM_EVENT_KEY2
//
// Description:
//      Do the action corresponding to (current state, EVENT_KEY2) and move to next state
//      using the StateTable. EVENT_KEY2 is the event when key2 is pressed.
// Operation:
//      Call the action function corresponding to the current state and EVENT_KEY2 inside
//      the StateTable structure. Update ActState to the NextState corresponding to the
//      current state and EVENT_KEY2 inside the StateTable structure.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_EVENT_KEY2 (void)
{

    StateTable[ActState][EVENT_KEY2].ptrFunct();            //call the action function
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY2.

    ActState = StateTable[ActState][EVENT_KEY2].NextState;  //Proceed to the next state
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY2.
}


// FSM_EVENT_KEY3
//
// Description:
//      Do the action corresponding to (current state, EVENT_KEY3) and move to next state
//      using the StateTable. EVENT_KEY3 is the event when key3 is pressed.
// Operation:
//      Call the action function corresponding to the current state and EVENT_KEY3 inside
//      the StateTable structure. Update ActState to the NextState corresponding to the
//      current state and EVENT_KEY3 inside the StateTable structure.
// Argument:
//      None
```

```
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_EVENT_KEY3 (void)
{

    StateTable[ActState][EVENT_KEY3].ptrFunct();            //call the action function
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY3.

    ActState = StateTable[ActState][EVENT_KEY3].NextState;  //Proceed to the next state
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY3.
}


// FSM_EVENT_KEY4
//
// Description:
//      Do the action corresponding to (current state, EVENT_KEY4) and move to next state
//      using the StateTable. EVENT_KEY4 is the event when key4 is pressed.
// Operation:
//      Call the action function corresponding to the current state and EVENT_KEY3 inside
//      the StateTable structure. Update ActState to the NextState corresponding to the
//      current state and EVENT_KEY4 inside the StateTable structure.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_EVENT_KEY4 (void)
{

    StateTable[ActState][EVENT_KEY4].ptrFunct();            //call the action function
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY4.

    ActState = StateTable[ActState][EVENT_KEY4].NextState;  //Proceed to the next state
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY4.
}


// FSM_EVENT_KEY_CONFIRM
//
// Description:
//      Do the action corresponding to (current state, EVENT_KEY_CONFIRM) and move to
//      next state using the StateTable. EVENT_KEY_CONFIRM is the event when the
//      confirm key is pressed.
// Operation:
//      Call the action function corresponding to the current state and EVENT_KEY_CONFIRM
//      inside the StateTable structure. Update ActState to the NextState corresponding to
//      the current state and EVENT_KEY_CONFIRM inside the StateTable structure.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
```

```
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_EVENT_KEY_CONFIRM (void)
{

    StateTable[ActState][EVENT_KEY_CONFIRM].ptrFunct();     //call the action function
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY_CONFIRM.

    ActState = StateTable[ActState][EVENT_KEY_CONFIRM].NextState;
                                                            //Proceed to the next state
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY_CONFIRM.
}


// FSM_EVENT_PW_MATCH
//
// Description:
//      Do the action corresponding to (current state, EVENT_PW_MATCH) and move to
//      next state using the StateTable. EVENT_PW_MATCH is the event when the
//      the user password matches.
// Operation:
//      Call the action function corresponding to the current state and EVENT_PW_MATCH
//      inside the StateTable structure. Update ActState to the NextState corresponding to
//      the current state and EVENT_PW_MATCH inside the StateTable structure.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_EVENT_PW_MATCH (void)
{

    StateTable[ActState][EVENT_PW_MATCH].ptrFunct();        //call the action function
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_PW_MATCH.

    ActState = StateTable[ActState][EVENT_PW_MATCH].NextState;
                                                            //Proceed to the next state
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_PW_MATCH.
}


// FSM_EVENT_PW_UNMATCH
//
// Description:
//      Do the action corresponding to (current state, EVENT_PW_UNMATCH) and move to
//      next state using the StateTable. EVENT_PW_UNMATCH is the event when the
//      the user password does not match.
// Operation:
//      Call the action function corresponding to the current state and EVENT_PW_UNMATCH
//      inside the StateTable structure. Update ActState to the NextState corresponding to
//      the current state and EVENT_PW_UNMATCH inside the StateTable structure.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
```

```
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_EVENT_PW_UNMATCH (void)
{

    StateTable[ActState][EVENT_PW_UNMATCH].ptrFunct();      //call the action function
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_PW_UNMATCH.

    ActState = StateTable[ActState][EVENT_PW_UNMATCH].NextState;
                                                            //Proceed to the next state
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_PW_UNMATCH.
}


// FSM_EVENT_KEY_RESET
//
// Description:
//      Do the action corresponding to (current state, EVENT_KEY_RESET) and move to
//      next state using the StateTable. EVENT_KEY_RESET is the event when the
//      the reset key (pressing key1 and key3 at the same time for 1 second) is pressed.
// Operation:
//      Call the action function corresponding to the current state and EVENT_KEY_RESET
//      inside the StateTable structure. Update ActState to the NextState corresponding to
//      the current state and EVENT_KEY_RESET inside the StateTable structure.
// Argument:
//      None
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/02/2018  Sunghoon Choi   Comments updated
void FSM_EVENT_KEY_RESET (void)
{

    StateTable[ActState][EVENT_KEY_RESET].ptrFunct();       //call the action function
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY_RESET.

    ActState = StateTable[ActState][EVENT_KEY_RESET].NextState;
                                                            //Proceed to the next state
                                                            //corresponding to the
                                                            //current state and
                                                            //EVENT_KEY_RESET.
}
```

```
//******************************************************************************//
//                                                                              //
//                                  fsm.h                                       //
//                          EE53 Door-Lock Project                              //
//                              Sunghoon Choi                                   //
//                                                                              //
//******************************************************************************//
// Description:
//      This file contains definitions for fsm.c
// Revision History:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      02/24/2018  Sunghoon Choi   Errors fixed
//      03/06/2018  Sunghoon Choi   Comments updated

#ifndef FSM_H               //Preprocessor to prevent multiple inclusion
#define FSM_H


// FSM_STATE_TABLE
//
// Description:
//  A structure to contain the information of functions and next states corresponding to
//  the current state and the event.

typedef struct
{
  void (*ptrFunct) (void);  //Function to be called according to current state
                            //and event
  int NextState;            //Next state to proceed according to current state
                            //and event
} FSM_STATE_TABLE;

extern int ActState;        //Global variable that indicates the current state of the FSM.


#define NR_EVENTS 8                 //Total number of events
#define EVENT_KEY1  0               //Event when key1 is pressed
#define EVENT_KEY2  1               //Event when key2 is pressed
#define EVENT_KEY3  2               //Event when key3 is pressed
#define EVENT_KEY4  3               //Event when key4 is pressed
#define EVENT_KEY_CONFIRM  4        //Event when confirm key is pressed
#define EVENT_PW_MATCH  5           //Event when password matches
#define EVENT_PW_UNMATCH  6         //Event when password does not match
#define EVENT_KEY_RESET  7          //Event when the reset key combo is pressed

#define NR_STATES 12                //Total number of states
#define ST_ACCEPT_FIRST_DIGIT  0    //State waiting for the first digit of password
#define ST_ACCEPT_SECOND_DIGIT  1   //State waiting for the second digit of password
#define ST_ACCEPT_THIRD_DIGIT  2    //State waiting for the third digit of password
#define ST_ACCEPT_FOURTH_DIGIT  3   //State waiting for the fourth digit of password
#define ST_ACCEPT_CONFIRM  4        //State waiting for the confirm key being pressed
#define ST_CHECK_PASSWORD  5        //State waiting for the validification of password
#define ST_DOOR_OPENED  6           //State when the door is opened
#define ST_RESET_FIRST_DIGIT  7     //State waiting for the first digit of password to be
                                    //reset
#define ST_RESET_SECOND_DIGIT  8    //State waiting for the second digit of password to be
                                    //reset
#define ST_RESET_THIRD_DIGIT  9     //State waiting for the third digit of password to be
                                    //reset
#define ST_RESET_FOURTH_DIGIT  10   //State waiting for the fourth digit of password to be
                                    //reset
#define ST_RESET_CONFIRM  11        //State waiting for the confirm key to be pressed to
                                    //finish resetting the password
```

```
void FSM_Init (void);                   // Initialize the finite state machine


void FSM_EVENT_KEY1 (void);             //Do the action corresponding to
                                        //(current state, EVENT_KEY1) and move to next state
                                        //using the StateTable.
                                        //EVENT_KEY1 is the event when key1 is pressed.


void FSM_EVENT_KEY2 (void);             //Do the action corresponding to
                                        //(current state, EVENT_KEY2) and move to next state
                                        //using the StateTable.
                                        //EVENT_KEY2 is the event when key2 is pressed.


void FSM_EVENT_KEY3 (void);             //Do the action corresponding to
                                        //(current state, EVENT_KEY3) and move to next state
                                        //using the StateTable.
                                        //EVENT_KEY3 is the event when key3 is pressed.


void FSM_EVENT_KEY4 (void);             //Do the action corresponding to
                                        //(current state, EVENT_KEY4) and move to next state
                                        //using the StateTable.
                                        //EVENT_KEY4 is the event when key4 is pressed.


void FSM_EVENT_KEY_CONFIRM (void);      //Do the action corresponding to
                                        //(current state, EVENT_KEY_CONFIRM) and move to next
                                        //state using the StateTable.
                                        //EVENT_KEY_CONFIRM is the event when confirm key
                                        //is pressed.


void FSM_EVENT_PW_MATCH (void);         //Do the action corresponding to
                                        //(current state, EVENT_PW_MATCH) and move to next
                                        //state using the StateTable.
                                        //EVENT_PW_MATCH is the event when the password matches


void FSM_EVENT_PW_UNMATCH (void);       //Do the action corresponding to
                                        //(current state, EVENT_PW_UNMATCH) and move to next
                                        //state using the StateTable.
                                        //EVENT_PW_UNMATCH is the event when the password
                                        //does not match.


void FSM_EVENT_KEY_RESET (void);        //Do the action corresponding to
                                        //(current state, EVENT_KEY_RESET) and move to next
                                        //state using the StateTable.
                                        //EVENT_KEY_RESET is the event when the reset key combo
                                        //is pressed


void FSM_save_digit_one (void);         //Save the input data from Key1 to accept the password
                                        //typed by the user.


void FSM_save_digit_two (void);         //Save the input data from Key2 to accept the password
                                        //typed by the user.


void FSM_save_digit_three (void);       //Save the input data from Key3 to accept the password
                                        //typed by the user.


void FSM_save_digit_four (void);        //Save the input data from Key4 to accept the password
                                        //typed by the user.
```

```
void FSM_return_to_start (void);    //Go back to the initial state after clearing the LCD
                                    //and the input buffer.

void FSM_do_nothing (void);         //Do nothing. Necessary for finite state machine.

void FSM_print_reset_msg (void);    //Print "reset: " on LCD to notify the user that
                                    //the door lock is now in reset mode.

void FSM_open_door (void);          //Open the door by activating the solenoid.

void FSM_close_door (void);         //Close the door by deactivating the solenoid.
                                    //Return to the initial state as well.

void FSM_reset_digit_one (void);    //Reset the password by storing the input from Key1.
                                    //This function only resets the password when the input
                                    //comes from Key1.
                                    //Display the corresponding number (1) on LCD.

void FSM_reset_digit_two (void);    //Reset the password by storing the input from Key2.
                                    //This function only resets the password when the input
                                    //comes from Key2.
                                    //Display the corresponding number (2) on LCD.

void FSM_reset_digit_three (void);  //Reset the password by storing the input from Key3.
                                    //This function only resets the password when the input
                                    //comes from Key3.
                                    //Display the corresponding number (3) on LCD.

void FSM_reset_digit_four (void);   //Reset the password by storing the input from Key4.
                                    //This function only resets the password when the input
                                    //comes from Key4.
                                    //Display the corresponding number (4) on LCD.

void FSM_alert_wrong_pw (void);     //Generate an alarm sound and go back to the initial
                                    //state.
                                    //This function is called when the user types an
                                    //incorrect password and presses the confirm key.


#endif /* FSM_H */
```

```
//******************************************************************************//
//                                                                              //
//                                  main.c                                      //
//                          EE53 Door-Lock Project                              //
//                              Sunghoon Choi                                   //
//                                                                              //
//******************************************************************************//
// Description:
//      This file contains the main loop for the door lock system.
//      After initializing the hardware settings of the system and resetting the password
//      to its default value, it enters the infinite loop and goes through the Finite State
//      Machine for the operation of the door lock system.
// Table of Contents:
//      CorrectPW[4]        -   The array that contains the correct 4-digit password
//      InputPW[4]          -   The array that contains the password typed by user
//      CorrectPWIndex      -   The index used for iterating through CorrectPW array
//      InputPWIndex        -   The index used for iterating through InputPW array
//      main()              -   The main function that initializes the hardware
//                              settings and runs the infinite while loop to
//                              operate the door lock system.
// Revision History:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      02/27/2018  Sunghoon Choi   Errors fixed
//      03/13/2018  Sunghoon Choi   Comments updated


#include <msp430.h>         //include the functions and constants for TI MSP430
#include "lcd_4bit.h"       //include the header file for LCD in 4-bit mode.
#include "fsm.h"            //include the header file for finite state machine
#include "general.h"        //include the header file for general functions and constants
#include "sound.h"          //include the header file for sound generating functions

int CorrectPW[4]
 = {DEFAULT_PW_DIGIT1, DEFAULT_PW_DIGIT2,DEFAULT_PW_DIGIT3,DEFAULT_PW_DIGIT4};
                            //Array that contains the correct 4-digit password.
                            //When the system's power is reset, the password is also reset
                            //to its default 4-digit value.


 int InputPW[4]
 = {DEFAULT_INPUT_DIGIT1,DEFAULT_INPUT_DIGIT2,DEFAULT_INPUT_DIGIT3,DEFAULT_INPUT_DIGIT4};
                            //Array that contains the password typed by the user.
                            //When the system's power is reset, the contents of the array
                            //are set to their default values. To prevent unexpected
                            //accidents, the default values are set to the values that
                            //can never be the correct password.


 int CorrectPWIndex = 0;    //The index used for iterating through CorrectPW array

 int InputPWIndex = 0;      //The index used for iterating through InputPW array

// main()
//
// Description:
//      It initializes the hardware settings of the door-lock system and runs an infinite
//      loop to operate the door-lock. Hardware settings include watchdog timer setting,
//      pin directions, interrupt settings, enabling interrupts, and LCD settings. After
//      resetting the password to its default value, it goes into the infinite loop
//      to use the finite state machine to operate the door lock.
// Operation:
//      It first stops the watch dog timer.
//      Then, it sets the directions of each pins of port 1 and port 2.
//      The port/pin setup looks like:
//
//      Port1------------------------------
//              Pin0: LCD (RS), OUTPUT
```

```
//              Pin1: LCD (RW), OUTPUT
//              Pin2: LCD (E) , OUTPUT
//              Pin3: Solenoid, OUTPUT
//              Pin4: LCD (D4), OUTPUT
//              Pin5: LCD (D5), OUTPUT
//              Pin6: LCD (D6), OUTPUT
//              Pin7: LCD (D7), OUTPUT
//      ---------------------------------
//
//      Port2-----------------------------
//              Pin0: Key1, INPUT
//              Pin1: Key2, INPUT
//              Pin2: Key3, INPUT
//              Pin3: Key4, INPUT
//              Pin4: Confirm key, INPUT
//              Pin5: Piezo speaker, OUTPUT
//      ---------------------------------
//
//      Then, it enables the interrupts for Port2 (keys) and set the interrupt mode to
//      "Low to High Transition trigger". It resets the interrupt flags for Port2.
//      Next, it enables the general interrupts by calling __bis_SR_register(GIE).
//      Then, it initializes the LCD settings in 4-bit mode and display the string
//      "Password: " on the LCD to accept passwords from user.
//      Finally, it enters the infinite loop to iterate through the Finite State Machine.
//      While most part of the state machine is handled by the port interrupt, the task of
//      checking whether the password matches is handled inside the infinte loop. When
//      the current state is ST_CHECK_PASSWORD (when the user has typed all 4 digits and
//      pressed the confirm key), it calls FSM_EVENT_PW_MATCH if the password matches
//      and calls FSM_EVENT_PW_UNMATCH otherwise.
// Argument:
//      NONE
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/13/2018  Sunghoon Choi   Comments updated
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;       // Stop the watchdog timer

    //Setting Port1 directions
    P1DIR |= BIT0;      //LCD (RS), OUTPUT
    P1DIR |= BIT1;      //LCD (RW), OUTPUT
    P1DIR |= BIT2;      //LCD (E) , OUTPUT
    P1DIR |= BIT3;      //Solenoid, OUTPUT
    P1DIR |= BIT4;      //LCD (D4), OUTPUT
    P1DIR |= BIT5;      //LCD (D5), OUTPUT
    P1DIR |= BIT6;      //LCD (D6), OUTPUT
    P1DIR |= BIT7;      //LCD (D7), OUTPUT

    //Setting Port2 directions
    P2DIR &= ~BIT0;     //Key1, INPUT
    P2DIR &= ~BIT1;     //Key2, INPUT
    P2DIR &= ~BIT2;     //Key3, INPUT
    P2DIR &= ~BIT3;     //Key4, INPUT
    P2DIR &= ~BIT4;     //Confirm key, INPUT
    P2DIR |= BIT5;      //Piezo speaker, OUTPUT

    P1OUT = 0;          //Resetting Port 1 outputs
    P2IN = 0;           //Resetting Port 2 inputs
    P2OUT = 0;          //Resetting Port 2 outputs


    P2IE  |= BIT0|BIT1|BIT2|BIT3|BIT4;          //Enable interrupts for the keys
                                                //(key1~confirm key)

    P2IES &= ~(BIT0|BIT1|BIT2|BIT3|BIT4);       //The interrupts of the keys are
```

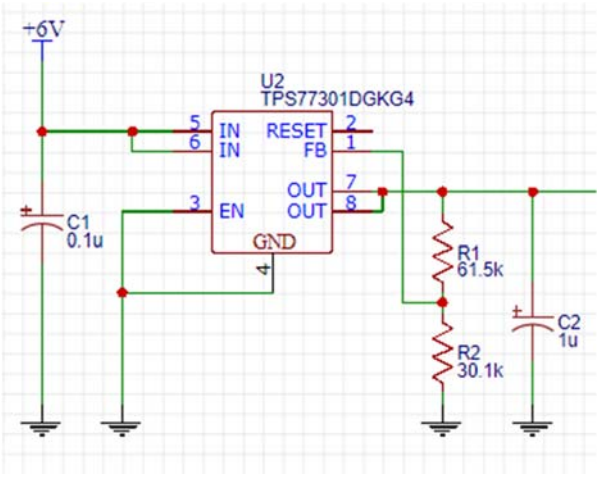
```
                                            //set to be triggered at
                                            //"Low to High Transition"

    P2IFG &= ~(BIT0|BIT1|BIT2|BIT3|BIT4);   //Reset the interrupt flags for the keys.
                                            //If the flags are set, interrupts cannot
                                            //be triggered.

    __bis_SR_register(GIE);                 //Enable interrupts in general.

    init_4bit_lcd();                        //Initialize the LCD in 4-bit mode

    send_string_4bit_lcd("Password:");      //Send and display "Password: " on LCD
                                            //when the system is rebooted.

    //*************************************************************************//
    //Initialization complete.
    //Inifinite loop starts.

    while(1)                                //Infinite loop
    {

        if(ActState == ST_CHECK_PASSWORD)   //When current state is ST_CHECK_PASSWORD
                                            //the while loop checks if the password
                                            //matches
        {
            if(check_pw())                  //If the password matches
            {
                FSM_EVENT_PW_MATCH();       //Call FSM_EVENT_PW_MATCH to open the door
                                            //and proceed to the next state
                                            //(ST_DOOR_OPENED)

            }
            else                            //If the password does not match
            {
                FSM_EVENT_PW_UNMATCH();     //Call FSM_EVENT_PW_UNMATCH to generate
                                            //an alarming sound and proceed to the
                                            //next state (ST_ACCEPT_FIRST_DIGIT)
            }
        }


    }
}

// port2_ISR
//
// Description:
//      It is the interrupt service routine for Port 2. (key1~4, confirm key)
//      Whenever there is a Low to High transition in Port2 input, this service routine
//      is called to identify the source of interrupt and handle the event.
// Operation:
//      It uses a Switch statement to identify the source of interrupt and call the
//      corresponding function to proceed. After calling the function, it resets the
//      corresponding interrupt flag to notify the system that the current interrupt has
//      been handled. After calling the function, proceeding to next state, and resetting
//      the interrupt flag, it calls the time delay function to delay 200 miliseconds.
//      200ms of delays after each event handling debounces the keys.
// Argument:
//      NONE
// Return Value:
//      None
// Revision Histories:
//      02/05/2018  Sunghoon Choi   Created
//      02/15/2018  Sunghoon Choi   Initial Compilation
//      03/13/2018  Sunghoon Choi   Comments updated
#pragma vector=PORT2_VECTOR
__interrupt void port2_ISR(void)
{
```

```
    switch(P2IN) {                      //identifies the source of interrupt

        case BIT0:                      //If key 1 is pressed
            FSM_EVENT_KEY1();           //Handle the event of key 1 being pressed
            P2IFG &= ~BIT0;             //Clear key 1's interrupt flag
            delay_ms(200);              //Debounce the key
            break;
        case BIT1:                      //If key 2 is pressed
            FSM_EVENT_KEY2();           //Handle the event of key 2 being pressed
            P2IFG &= ~BIT1;             //Clear key 2's interrupt flag
            delay_ms(200);              //Debounce the key
            break;
        case BIT2:                      //If key 3 is pressed
            FSM_EVENT_KEY3();           //Handle the event of key 3 being pressed
            P2IFG &= ~BIT2;             //Clear key 3's interrupt flag
            delay_ms(200);              //Debounce the key
            break;
        case BIT3:                      //If key 4 is pressed
            FSM_EVENT_KEY4();           //Handle the event of key 4 being pressed
            P2IFG &= ~BIT3;             //Clear key 4's interrupt flag
            delay_ms(200);              //Debounce the key
            break;
        case BIT4:                      //If confirm key is pressed
            FSM_EVENT_KEY_CONFIRM();    //Handle the event of confirm key being pressed
            P2IFG &= ~BIT4;             //Clear confirm key's interrupt flag
            delay_ms(200);              //Debounce the key
            break;
        case (BIT0+BIT3):               //If key 1 and key 3 are pressed simultaneously
            FSM_EVENT_KEY_RESET();      //Handle the event to reset the password
            P2IFG &= ~(BIT0+BIT4);      //Clear the interrupt flags
            delay_ms(200);              //Debounce the key
        default:
            P2IFG = 0;                  //Clear all interrupt flags of port 2 (keys)
    }

}
```

Technical Descriptions

Voltage Regulator

MSP430G2553, the processor of the door lock, is driven by 3.6V VCC. For a stable voltage supply, TPS77301DGKG4 LDO voltage regulator was used to generate 3.6V from the 6V source.



Peripheral settings above allow TPS77301DGKG4 to generate a constant 3.6V voltage supply.

HD44780 LCD Display

HD44780 LCD gets 3.6V power supply from the output of TPS773 voltage regulator circuit. While the Contrast (brightness) of LCD could be controlled by the input voltage on V0 pin, I directly connected V0 pin to ground to maximize the Contrast.

RS pin determines if it is in data transmission mode or command transmission mode. RS pin should be HIGH (RS=1) when sending data to LCD and LOW (RS=0) when sending commands.

RW pin determines Read(RW=1)/Write(RW=0) mode. In this project, only Write is used to transmit data and commands to LCD.

EN pin is the ENABLE TRIGGER pin. When communicating LCD in 4-bit mode, EN bit must be toggled between nibbles to enable triggering to allow successful data/command transmission.

To operate HD44780 in 4-bit mode, a certain series of commands must be transmitted .Notice that in 4-bit mode, we only use D4, D5, D6, D7 for data/command transmission. See next page for the necessary commands for operation of HD44780 in 4-bit mode.

Initialization for 4-bit operation

The module powers up in 8-bit mode. The initial start-up instructions are sent in 8-bit mode, with the lower four bits (which are not connected) of each instruction as don't cares.

<POWER ON>

<Wait at least 15ms>

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	n/c	n/c	n/c	n/c

<Wait at least 4.1ms>

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	n/c	n/c	n/c	n/c

<Wait at least 100us>

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	n/c	n/c	n/c	n/c

<Wait 4.1ms>

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	0	n/c	n/c	n/c	n/c

After the fourth instruction shown above, which switches the module to 4-bit operation, the control bytes are sent on consecutive enable cycles (no delay is required between nibbles). The most significant nibble is sent first, followed immediately by the least significant nibble. See the next page for details.

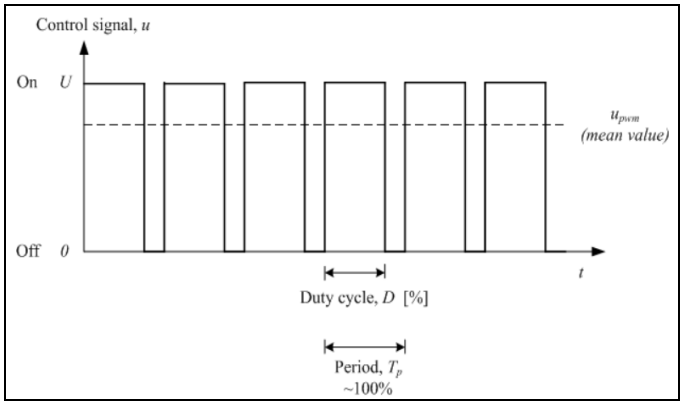
Commands to be sent for operation of HD44780 in 4bit mode

6N Solenoid

This project uses a 6N solenoid (6V 0.8A) to open/close the door. Since MSP430G2553 can't draw enough current for the solenoid, a TIP112 Darlington transistor was used in common-emitter mode for driving the solenoid. A diode was attached in parallel with the solenoid to protect it from reverse voltage spikes.

Piezo Speaker

The piezo speaker in this project is used to generate an alarming sound when the password does not match, and a welcoming sound when the password matches. Piezo speaker is driven by a Pulse Width Modulation (PWM) control since it needs to vibrate to generate sounds. For the maximum amplitude of sound, duty cycle was set to 50%.



The frequency of the waveform in 50% duty cycle can be calculated as:

$$Frequency = \frac{1}{2 * PulseWidth}$$

For the door-lock system, alarm sound and welcoming sound are generated by a series of notes of different pitches. The frequencies of the generated sounds are:

Sound Type	Frequency
Alarm Sound	555Hz - 714Hz - 555Hz - 714Hz (about 500ms for each note)
Welcoming Sound	500Hz – 555Hz – 625Hz (about 700ms for each note)

Keys (Buttons)

Five keys are attached to the door-lock: Key1, Key2, Key3, Key4, and Confirm Key. The system accepts 4 digit password from the user through the keys which are connected to Port 2. MSP430G2553 handles the inputs by Port 2 Interrupt Service Routine. While the system is designed to accept one key at a time, pressing Key1 and Key3 at the same time allows the system to enter “reset password” mode. 200ms of time delay is implemented after each interrupt handling to debounce the keys.

Finite State Machine Table (page 1)

	EVENT_KEY1		EVENT_KEY2		EVENT_KEY3	
	Function	**Next State**	**Function**	**Next State**	**Function**	**Next State**
ST_ACCEPT_FIRST_DIGIT	save_digit_one	ST_ACCEPT_SECOND_DIGIT	save_digit_two	ST_ACCEPT_SECOND_DIGIT	save_digit_three	ST_ACCEPT_SECOND_DIGIT
ST_ACCEPT_SECOND_DIGIT	save_digit_one	ST_ACCEPT_THIRD_DIGIT	save_digit_two	ST_ACCEPT_THIRD_DIGIT	save_digit_three	ST_ACCEPT_THIRD_DIGIT
ST_ACCEPT_THIRD_DIGIT	save_digit_one	ST_ACCEPT_FOURTH_DIGIT	save_digit_two	ST_ACCEPT_FOURTH_DIGIT	save_digit_three	ST_ACCEPT_FOURTH_DIGIT
ST_ACCEPT_FOURTH_DIGIT	save_digit_one	ST_ACCEPT_CONFIRM	save_digit_two	ST_ACCEPT_CONFIRM	save_digit_three	ST_ACCEPT_CONFIRM
ST_ACCEPT_CONFIRM	do_nothing	ST_ACCEPT_CONFIRM	do_nothing	ST_ACCEPT_CONFIRM	do_nothing	ST_ACCEPT_CONFIRM
ST_CHECK_PASSWORD	do_nothing	ST_CHECK_PASSWORD	do_nothing	ST_CHECK_PASSWORD	do_nothing	ST_CHECK_PASSWORD
ST_DOOR_OPENED	do_nothing	ST_DOOR_OPENED	do_nothing	ST_DOOR_OPENED	do_nothing	ST_DOOR_OPENED
ST_RESET_FIRST_DIGIT	reset_digit_one	ST_RESET_SECOND_DIGIT	reset_digit_two	ST_RESET_SECOND_DIGIT	reset_digit_three	ST_RESET_SECOND_DIGIT
ST_RESET_SECOND_DIGIT	reset_digit_one	ST_RESET_THIRD_DIGIT	reset_digit_two	ST_RESET_THIRD_DIGIT	reset_digit_three	ST_RESET_THIRD_DIGIT
ST_RESET_THIRD_DIGIT	reset_digit_one	ST_RESET_FOURTH_DIGIT	reset_digit_two	ST_RESET_FOURTH_DIGIT	reset_digit_three	ST_RESET_FOURTH_DIGIT
ST_RESET_FOURTH_DIGIT	reset_digit_one	ST_RESET_CONFIRM	reset_digit_two	ST_RESET_CONFIRM	reset_digit_three	ST_RESET_CONFIRM
ST_RESET_CONFIRM	do_nothing	ST_RESET_CONFIRM	do_nothing	ST_RESET_CONFIRM	do_nothing	ST_RESET_CONFIRM

	EVENT_KEY4		EVENT_KEY_CONFIRM		EVENT_PW_MATCH	
	Function	**Next State**	**Function**	**Next State**	**Function**	**Next State**
ST_ACCEPT_FIRST_DIGIT	save_digit_four	ST_ACCEPT_SECOND_DIGIT	return_to_start	ST_ACCEPT_FIRST_DIGIT	do_nothing	ST_ACCEPT_FIRST_DIGIT
ST_ACCEPT_SECOND_DIGIT	save_digit_four	ST_ACCEPT_THIRD_DIGIT	return_to_start	ST_ACCEPT_FIRST_DIGIT	do_nothing	ST_ACCEPT_SECOND_DIGIT
ST_ACCEPT_THIRD_DIGIT	save_digit_four	ST_ACCEPT_FOURTH_DIGIT	return_to_start	ST_ACCEPT_FIRST_DIGIT	do_nothing	ST_ACCEPT_THIRD_DIGIT
ST_ACCEPT_FOURTH_DIGIT	save_digit_four	ST_ACCEPT_CONFIRM	return_to_start	ST_ACCEPT_FIRST_DIGIT	do_nothing	ST_ACCEPT_FOURTH_DIGIT
ST_ACCEPT_CONFIRM	do_nothing	ST_ACCEPT_CONFIRM	do_nothing	ST_CHECK_PASSWORD	do_nothing	ST_ACCEPT_CONFIRM
ST_CHECK_PASSWORD	do_nothing	ST_CHECK_PASSWORD	do_nothing	ST_CHECK_PASSWORD	open_door	ST_DOOR_OPENED
ST_DOOR_OPENED	do_nothing	ST_DOOR_OPENED	close_door	ST_ACCEPT_FIRST_DIGIT	do_nothing	ST_DOOR_OPENED
ST_RESET_FIRST_DIGIT	reset_digit_four	ST_RESET_SECOND_DIGIT	print_reset_msg	ST_RESET_FIRST_DIGIT	do_nothing	ST_RESET_FIRST_DIGIT
ST_RESET_SECOND_DIGIT	reset_digit_four	ST_RESET_THIRD_DIGIT	print_reset_msg	ST_RESET_FIRST_DIGIT	do_nothing	ST_RESET_SECOND_DIGIT
ST_RESET_THIRD_DIGIT	reset_digit_four	ST_RESET_FOURTH_DIGIT	print_reset_msg	ST_RESET_FIRST_DIGIT	do_nothing	ST_RESET_THIRD_DIGIT
ST_RESET_FOURTH_DIGIT	reset_digit_four	ST_RESET_CONFIRM	print_reset_msg	ST_RESET_FIRST_DIGIT	do_nothing	ST_RESET_FOURTH_DIGIT
ST_RESET_CONFIRM	do_nothing	ST_RESET_CONFIRM	return_to_start	ST_ACCEPT_FIRST_DIGIT	do_nothing	ST_RESET_CONFIRM

Finite State Machine Table (page 2)

	EVENT_PW_UNMATCH		EVENT_KEY_RESET	
	Function	**Next State**	**Function**	**Next State**
ST_ACCEPT_FIRST_DIGIT	do_nothing	ST_ACCEPT_FIRST_DIGIT	print_reset_msg	ST_RESET_FIRST_DIGIT
ST_ACCEPT_SECOND_DIGIT	do_nothing	ST_ACCEPT_SECOND_DIGIT	print_reset_msg	ST_RESET_FIRST_DIGIT
ST_ACCEPT_THIRD_DIGIT	do_nothing	ST_ACCEPT_THIRD_DIGIT	print_reset_msg	ST_RESET_FIRST_DIGIT
ST_ACCEPT_FOURTH_DIGIT	do_nothing	ST_ACCEPT_FOURTH_DIGIT	print_reset_msg	ST_RESET_FIRST_DIGIT
ST_ACCEPT_CONFIRM	do_nothing	ST_ACCEPT_CONFIRM	print_reset_msg	ST_RESET_FIRST_DIGIT
ST_CHECK_PASSWORD	alert_wrong_pw	ST_ACCEPT_FIRST_DIGIT	do_nothing	ST_CHECK_PASSWORD
ST_DOOR_OPENED	do_nothing	ST_DOOR_OPENED	do_nothing	ST_DOOR_OPENED
ST_RESET_FIRST_DIGIT	do_nothing	ST_RESET_FIRST_DIGIT	do_nothing	ST_RESET_FIRST_DIGIT
ST_RESET_SECOND_DIGIT	do_nothing	ST_RESET_SECOND_DIGIT	do_nothing	ST_RESET_SECOND_DIGIT
ST_RESET_THIRD_DIGIT	do_nothing	ST_RESET_THIRD_DIGIT	do_nothing	ST_RESET_THIRD_DIGIT
ST_RESET_FOURTH_DIGIT	do_nothing	ST_RESET_FOURTH_DIGIT	do_nothing	ST_RESET_FOURTH_DIGIT
ST_RESET_CONFIRM	do_nothing	ST_RESET_CONFIRM	do_nothing	ST_RESET_CONFIRM

User Manual

SH-DL-01

Initial Version: 03/02/2018

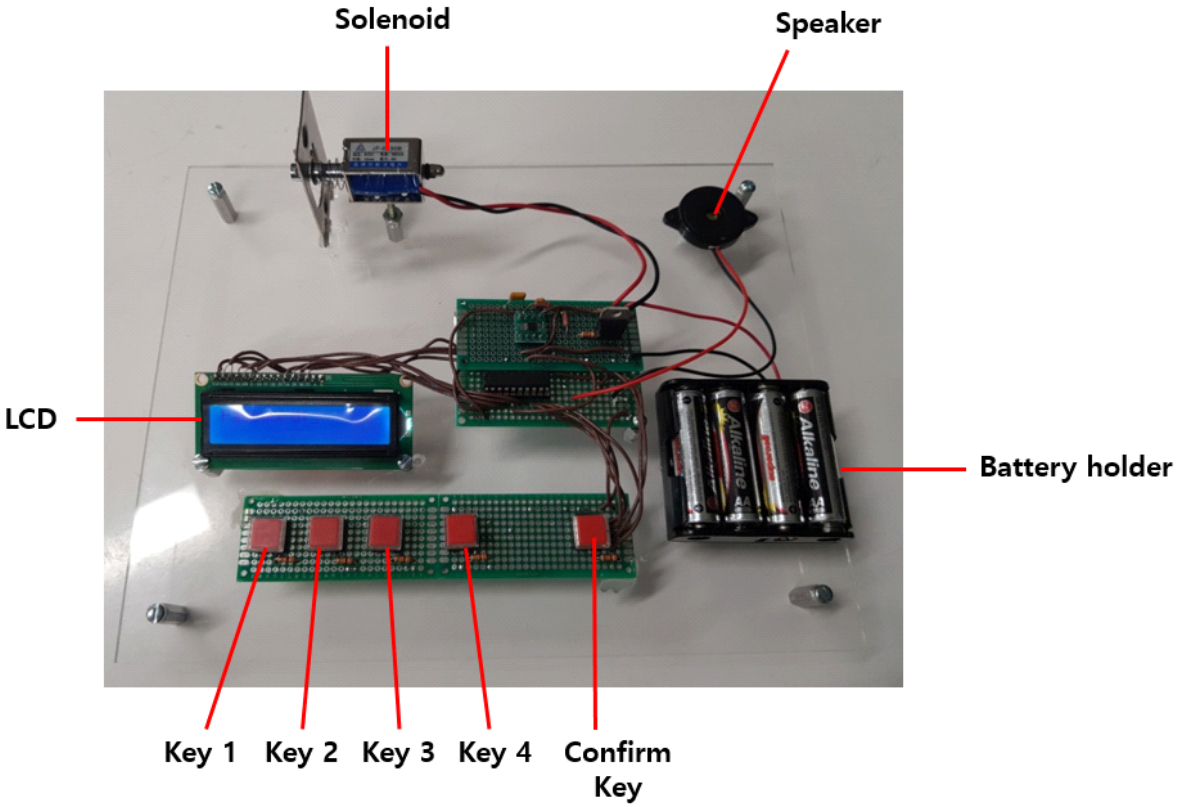
Final Revision: 03/13/2018

Power Supply

- SH-DL-01 requires 4 x AA batteries for the power supply.

> ⚠ **For Your Safety**
>
> **Please do not mix old and new batteries.**

Your SH-DL-01



How to reset the password

- Keep Key1 and Key3 pressed at the same time for 1 second
- “Reset: ” will show on the LCD display
- Type your desired new 4-digit password
- Press the Confirm Key

How to open the door

- Type the 4-digit password
- Press the Confirm Key
  - If you typed a wrong digit by mistake, you can press the confirm key to clear the password on the LCD display.

When the password does not match

- SH-DL-01 will generate an alarm sound
- The door will stay closed.
- The LCD display will be cleared and be ready to receive new input.

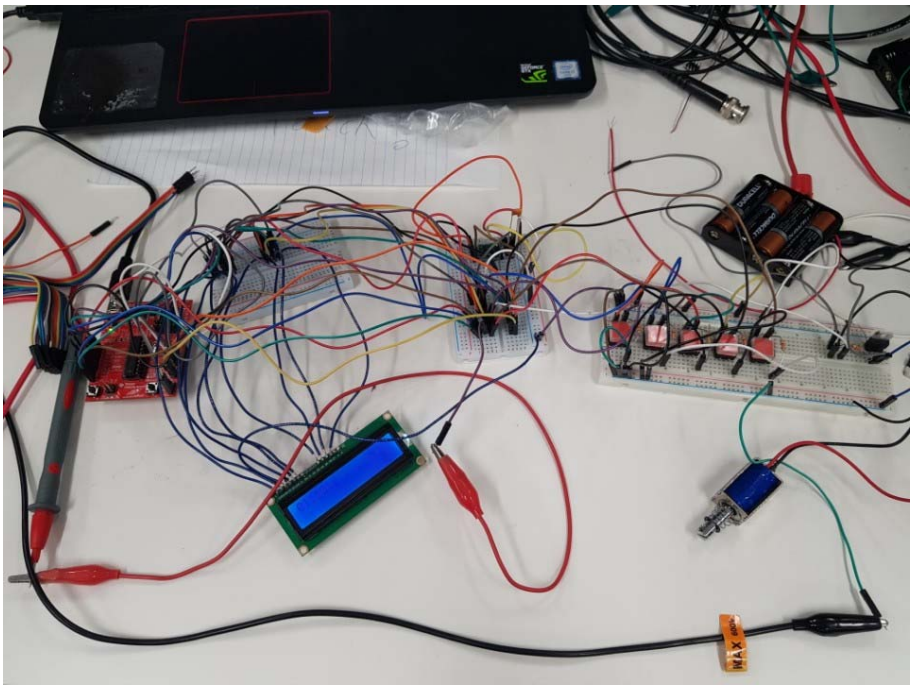
When the password matches

- SH-DL-01 will generate a welcoming sound
- The door will open
- The LCD display will show “Welcome”

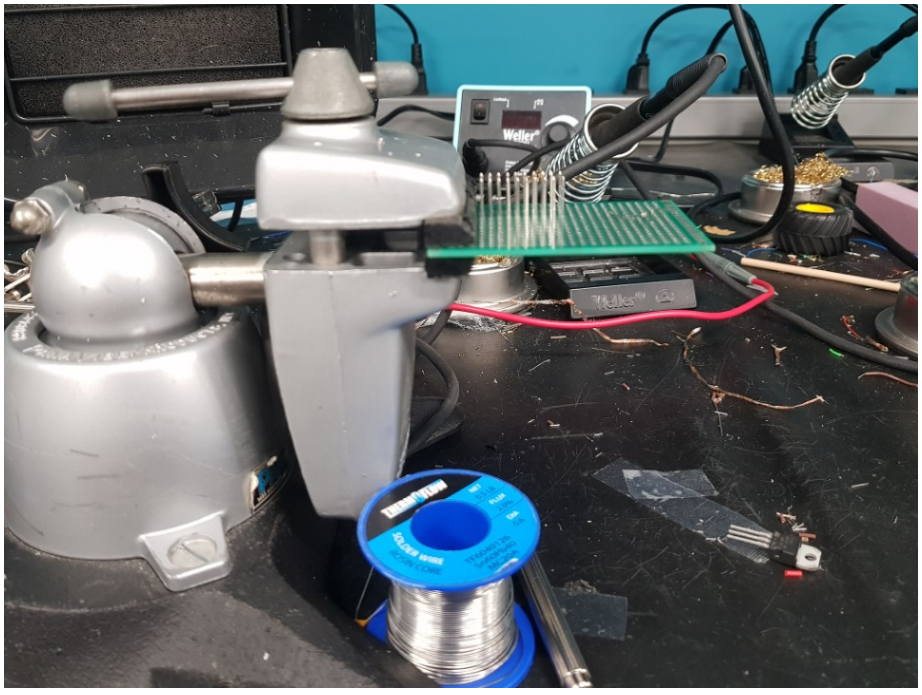
How to close the door

- When the door is open, press the Confirm Key to close the door.
- LCD display will again show “Password: ”, waiting for the user to type the password.

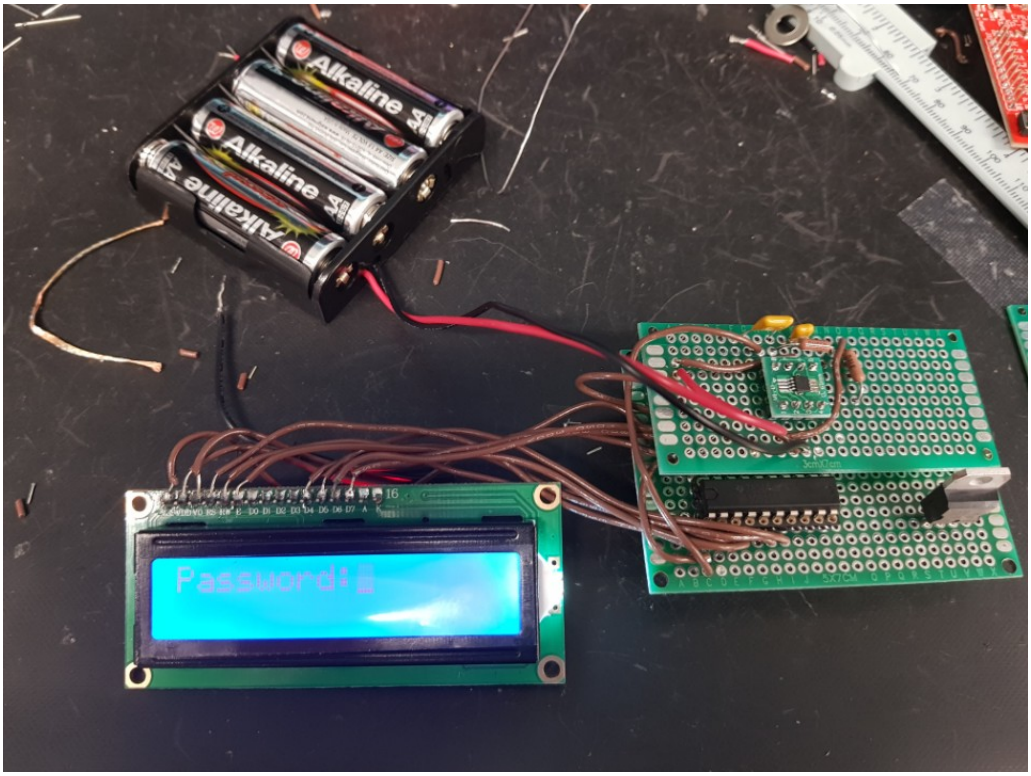
[Appendix]
Product Making Process



Prototyping with breadboard and MSP430 evaluation board

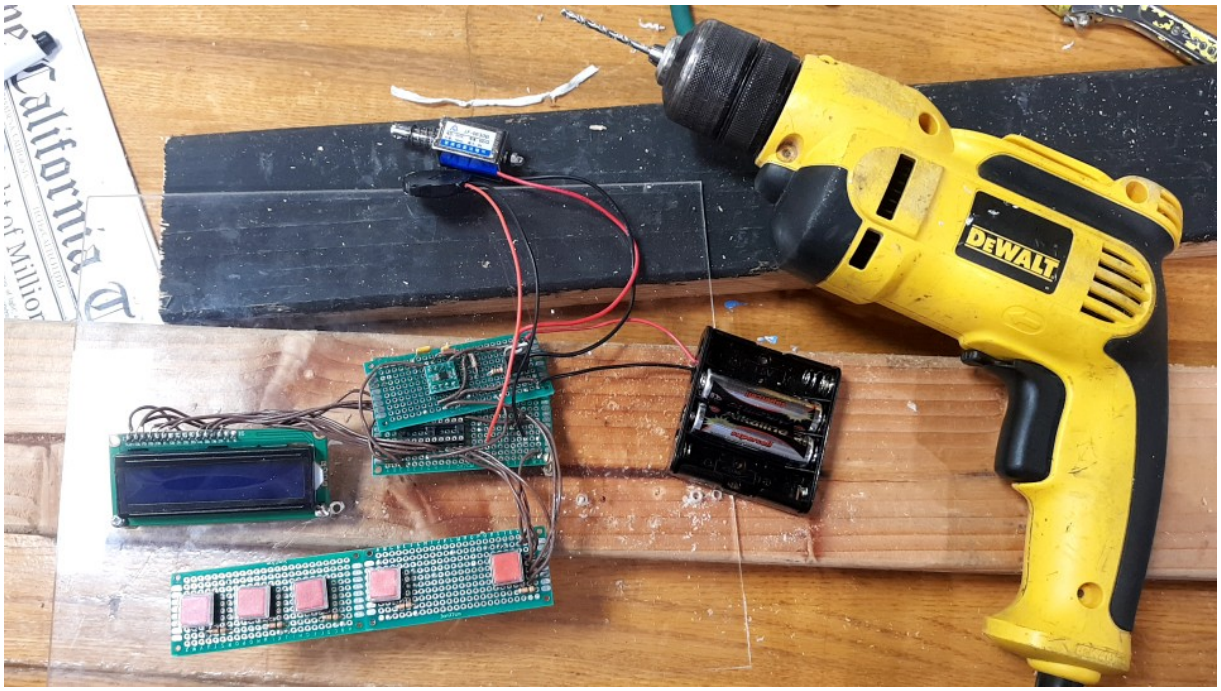


Soldering on the grid board

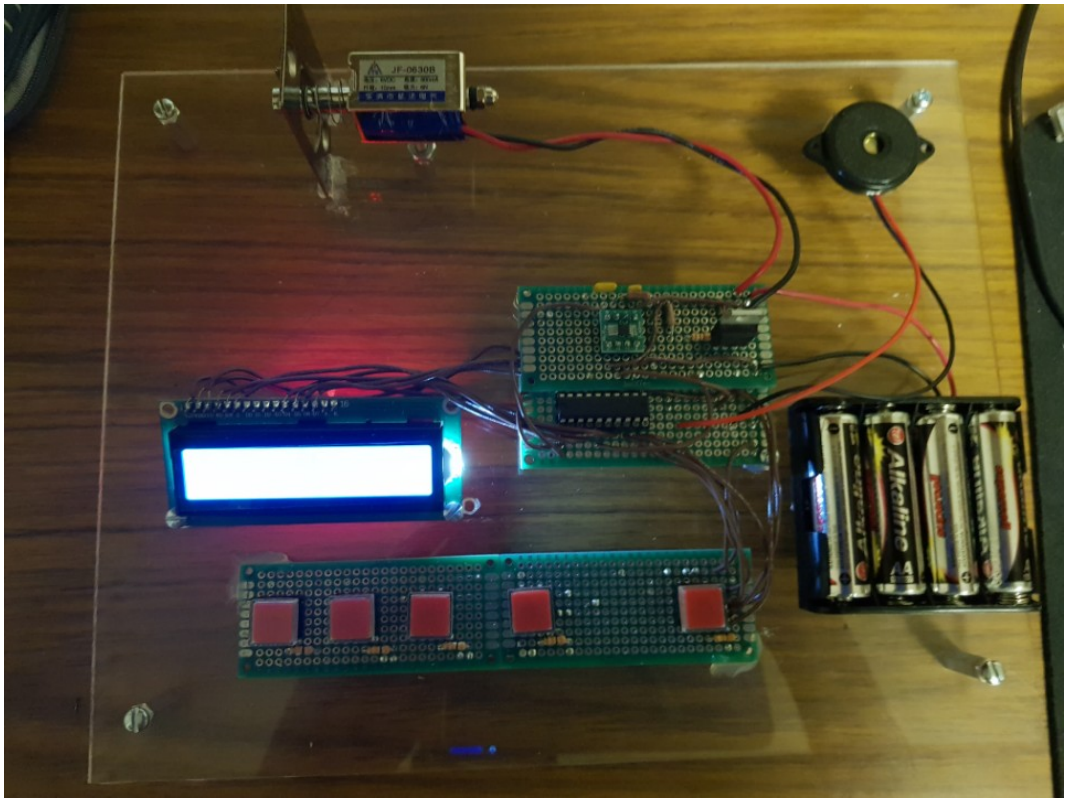


LCD attached to the main circuit

Cutting the acrylic glass to make a platform



Drilling the glass to mount the system



Making Complete