ES-ESP5200 Modern Embedded Systems programming Coursework 2020

Design, develop, test and evaluate an
Embedded Control System for a Real-Time application

Final Report

Name: **M. Singh**

The embedded real-time application I have implemented is **DLS:**

A door-lock system utilizing a keypad, liquid crystal display (LCD), servo motor, a push-button and eight light emitting diodes (LEDs) that can grant or deny a user access. In addition to that, the system can be put into lockdown mode, in which the user must wait a certain amount of time before utilizing the system again.

1. Introduction

This project was finalised through two development iterations; version 0.1 and version 0.2. This document aims to describe the development for version 0.2. However, necessary descriptions of v0.1 is included where appropriate to better describe the overall development of DLS v0.2.

See Figure 1 for a use case diagram for DLS v0.2 and Figure 2 for a block diagram showing the main components. DLS is a door-lock system that takes password input from the user (keypad), assesses input and grants access if input matches predefined password. The system will deny access if input password does not match predefined password. Furthermore, DLS v0.2 has more features by adding two more devices, such as lockdown mode (triggered by push-button) and rotation of servo motor to signalise a door opening in case of granted access. In conjunction with LEDs, the LCD shows necessary information to the user.

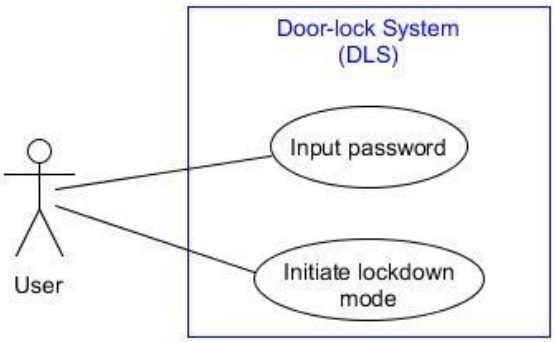

Figure 1: Use case diagram for DLS

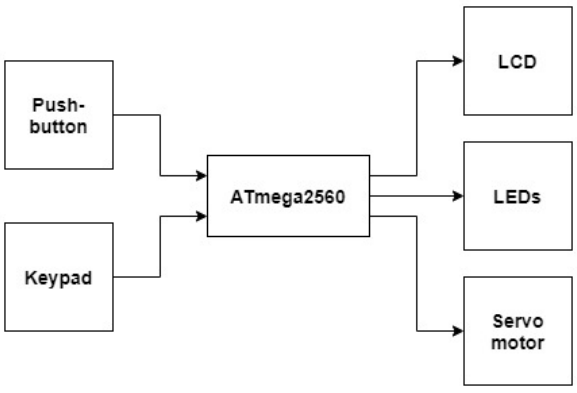

Figure 2: Block diagram of main components in DLS

The user has 15 seconds of time to input full password from first keypress and in lockdown mode, the system is suspended from normal operation for 60 seconds of time.

The requirements for DLS are shown in table 1. R01 relates to system function if access is granted to user while R02 relates to system function if access is denied. R03 defines system function for push-button press.

Table 1: Requirements for DLS

Requirement ID	Description
R01A	The system shall grant access if input password matches predefined password.
R01B	The system shall give the user 15 seconds of time to input full password from first keypress.
R01C	In case of access granted, the servo motor shall spin, LEDs shall light accordingly and LCD shall display “Access granted”
R02A	The system shall deny access if input password does not match predefined password.
R02B	In case of access denied, LEDs shall light accordingly and LCD shall display “Access denied”
R03A	The system shall go into lockdown mode if push-button is pressed.
R03B	The system shall stay in lockdown mode for 60 seconds of time from button press.

2. Design Documentation

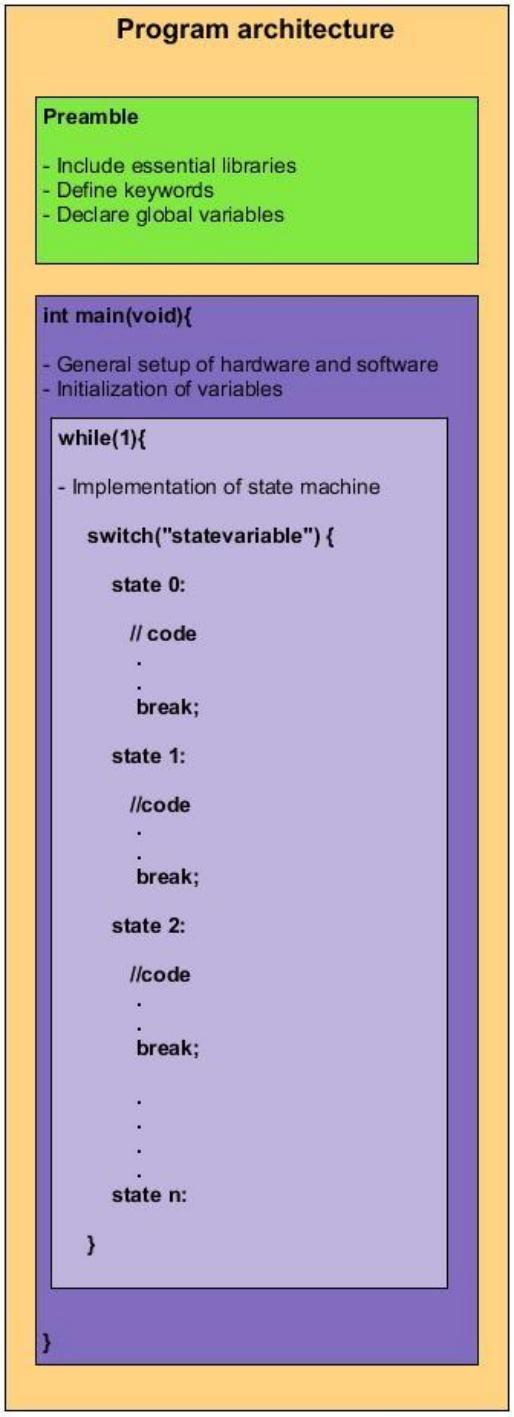

Figure 3: Overview of program arch. of DLS

The software for DLS is written in Embedded C and utilizes a finite state machine. For a general overview of the program architecture, see Figure 3. The first section is the preamble section. In this section libraries/header files are included, keywords defined, and global variables declared. Such variables sometimes must be declared 'volatile' since we do not want the compiler to include them in the optimization process., especially if the variables are to be utilized by interrupt service routines.

The second section is the method "int main(void)" section. It is in this section we carry out necessary setups of hardware and software to be utilized by the program, such as ports and interrupts. Furthermore, the sub-section begun by "while(1)" is of special interest since it is in this sub-section the main code is executed. It is worth noting that the CPU frequency is set to 1MHz. In looser terms, the program is executed 1 million times a second. It is almost as if all code is run concurrently due to the high frequency, when in reality it is run sequentially.

In addition to that, the state machine is implemented inside of the "while(1)" sub-section. A statevariable, meaning a variable that is assessed to determine the next state, is used to configure the transition between one state and another.

The state machine is further detailed in Figure 4. It is recommended to look at this diagram in conjunction with sequence diagrams (Figure 6-10). State n translates to case n in the switch statement inside while(1) sub-section. State 0 is the standby mode, meaning the default state the system functions in. It is in this state that the system asks for user input. See Figure 6. In state 0, the program will wait inside the for loop until a keypress has been registered. The keypress will then be assigned to index i in the "input" array via method "AssignSV" (assign switchvalue). This method will not return anything as long as a switch is not pressed because of a while loop that will keep looping. "AssignSV" method utilizes four sub-routines to detect a keypress. It does so by sending a high level of logic to all rows, then it sends a low signal to the row it is checking, A column pin will return low signal if a key has been pressed in a particular row and column. This way, the "AssignSV" method returns the keypress to the respective input[i]. It is worth noting that before a keypress is returned to the stated array that holds the input keypresses, the program checks if any interruptflags have been set. "_intrFlag" is a flag that is set when TIMER1 times out. TIMER1 starts when the first keypress is pressed, as can be seen in Figure 6. The timer counts to 1 second and calls ISR(TIMER1_COMPA_vect) that updates a 15 second countdown on the LCD. This way the user has 15 seconds to input the password from the first keypress. If the timer reaches 0, the "_intrFlag" is set and the rest of the for-loop is discarded. Do also note that the for-loop will only let the system assign a keypress to the input array if "_intrFlag_EG" is false as well. This flag is set when the external interrupt INT4 is true.

This interrupt is connected to the push button shown in Figure 4. Once this push button is pushed, the ISR(INT4_vect) is called. This interrupt service routine checks if global variable "_timer1" is smaller than the timeout time of 15 seconds, meaning if TIMER1 is running. If it is, then it resets timer 1. After that, the interrupt service routine checks if the global variable "_timer3" is equal to the predefined lockdown time. If it is, then the LCD displays "Lockdown mode", TIMER3 starts counting to 1 and the stated "_intrFlag_EG" is set.

Once the external interrupt flag is set, the for-loop runs empty, meaning it does not execute anything and the program exits the loop after its last run. After that, TIMER1 is reset as a extra measure. Then the program determines what state the system should transition to. Since the external interrupt flag is set, it will choose to go to state 2 which is the lockdown mode state. Just before transitioning, the program will clear the flag.

After transitioning to state 2, the program is suspended from any other function other than checking if global variable "_timer" is equal to 0, since it decrements from "lockdown_time" (60 s) to 0 through the ISR(TIMER3_COMPA_vect) that gets called every second. This means that any external keypress or other input is not registered. After the countdown is finished, the program determines transitioning to state 0 which is the standby state and sets the global variable "_timer3" to its original value again.

Returning back to state 0, if the password is input within the 15s timelimit and no push-button is pressed, then the program determines at the end of state 0 that it should transition to state 1 in order to check the input password against the predefined password. In state 1, only one method is called; "CheckPassword((*p_passw), (*p_input))". This method takes two arguments, and both are pointers. The former argument is a pointer to the array that holds the predefined password while the latter is a pointer to the array that holds the value for the input keypresses.

The way this method checks if the input password is correct, is by see if each digit pressed corresponds to respective digit in passw array as well, i.e. input0 should match passw0 and so on. For each digit that matches, a variable increments, signifying the correct match of input digit and password digit. If the number of correct digits is matches with the length of predefined password, then access is granted. Otherwise it is not. If access is granted then the program transitions into state 3. In state 3, the program does two things. The first is to display "Access granted" on the LCD and flash the blue LEDs. The other is to spin the servo motor to indicate that a door has been opened. TIMER4 is used for pulse with modulation (PWM) signal that is sent to the servo motor.

In case the program denies access to the user, the program transitions to state 4. In state 4, the LCD displays "Access denied" and flashed red LEDs to indicate no access. After that, the program returns to state 0 where it remains in standby.

Furthermore, a state 5 has been created but not developed to carry out any function. This state is reserved for future use and development.

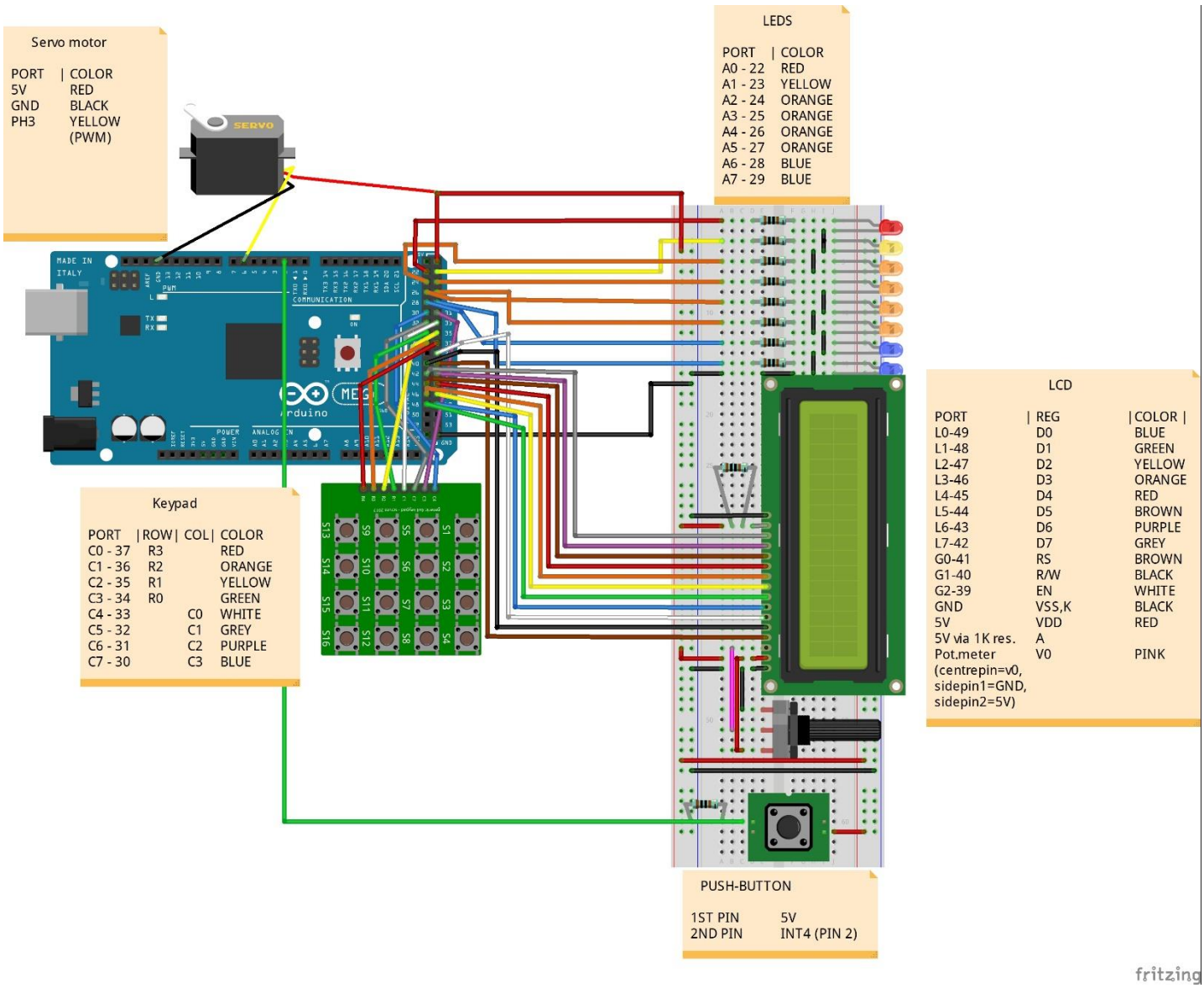

Figure 4: Hardware connectivity diagram

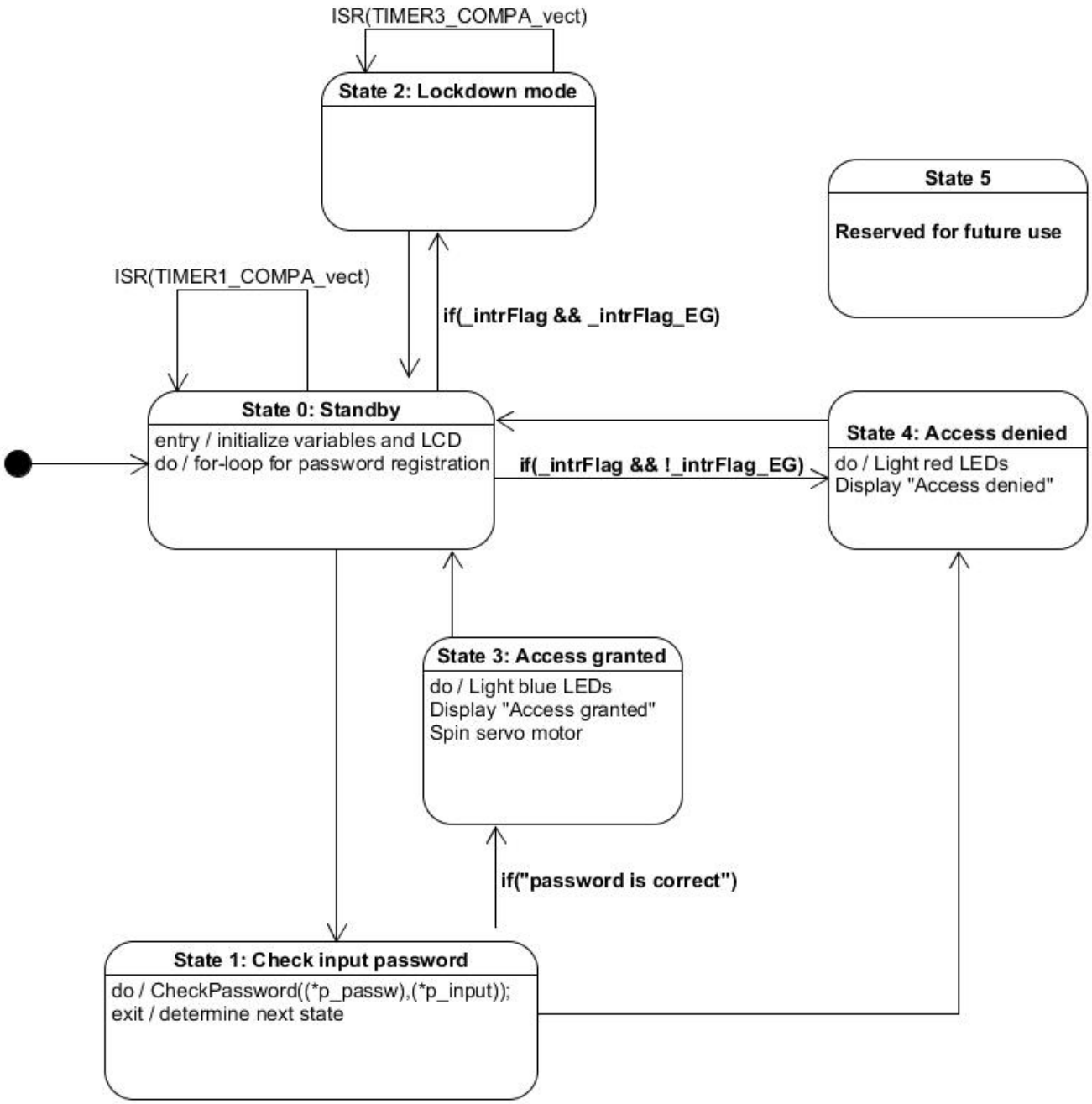

Figure 5: State machine diagram of DLS

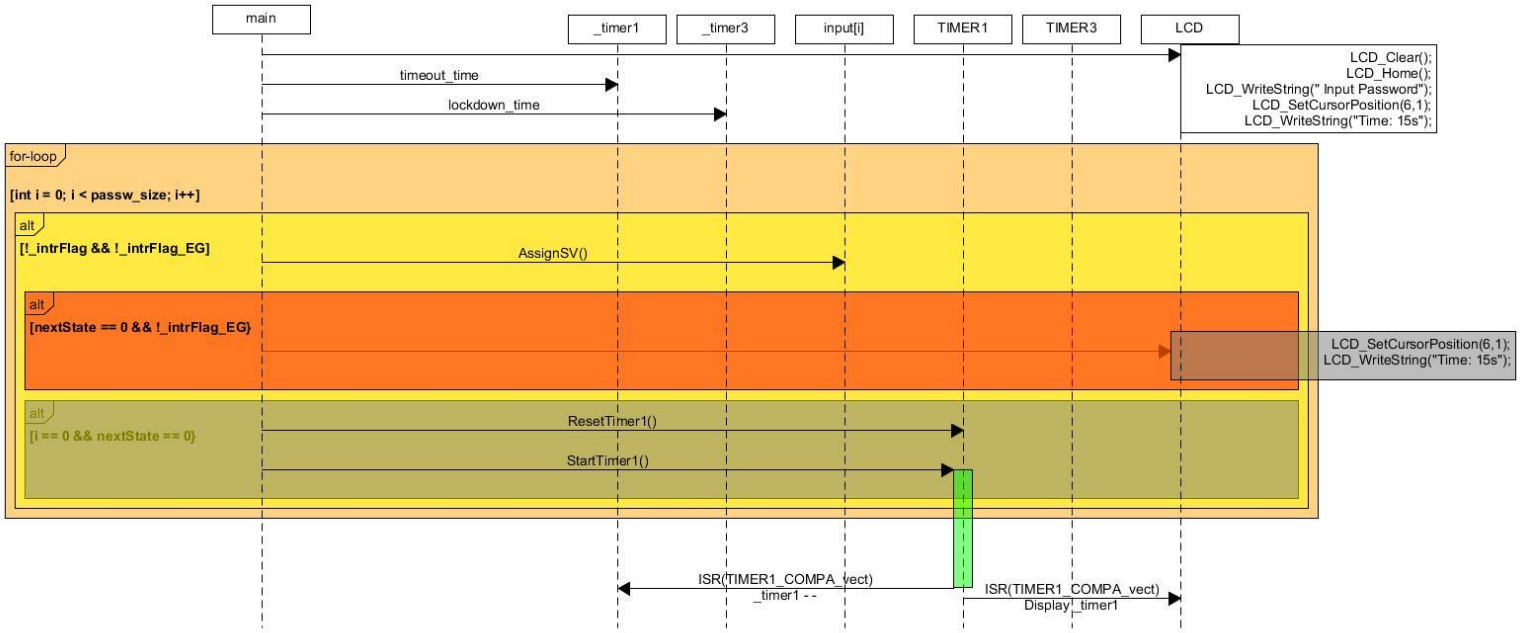

Figure 6: Sequence diagram for for-loop in state 0

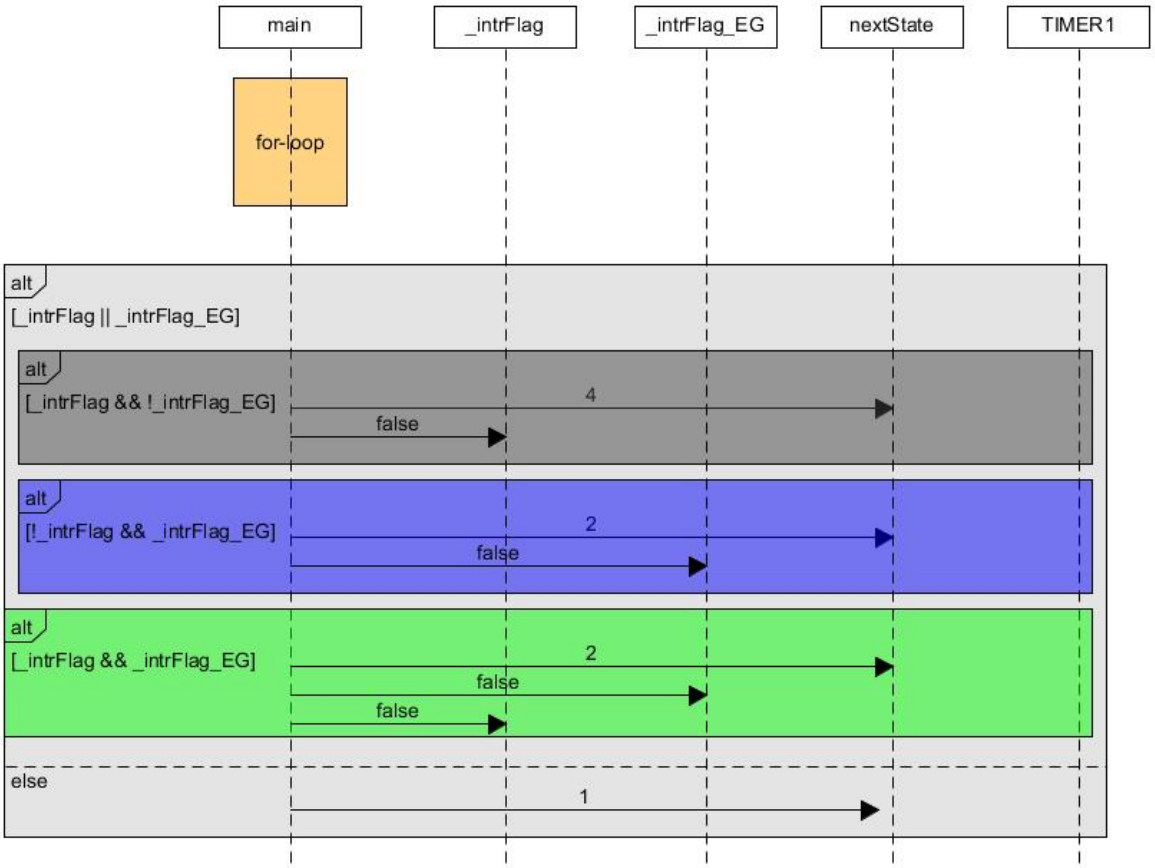

Figure 7: Sequence diagram for state 0

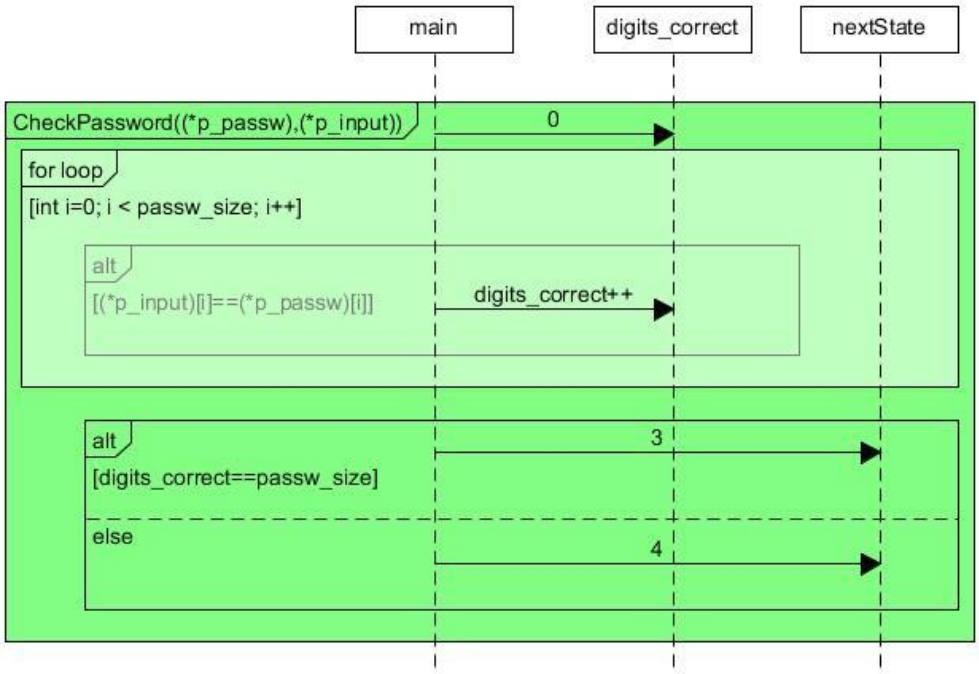

Figure 8: Sequence diagram for state 1

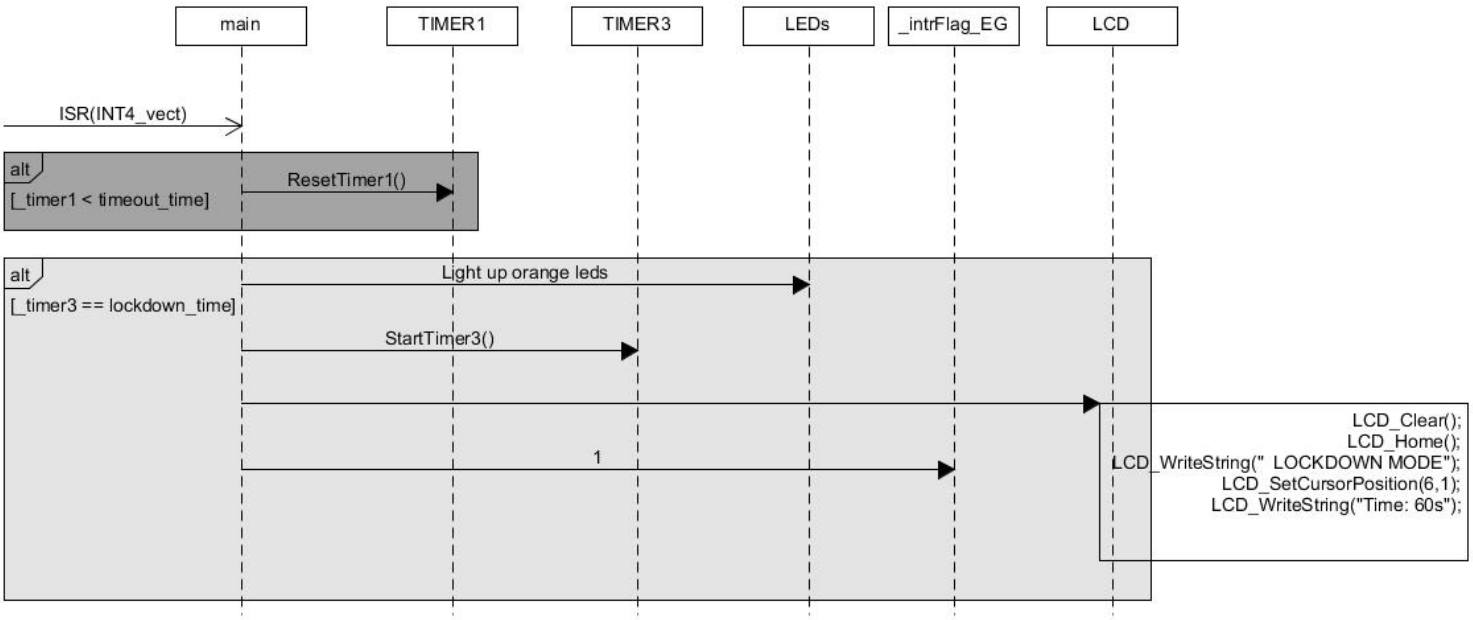

Figure 9: Sequence diagram for external interrupt ISR 4

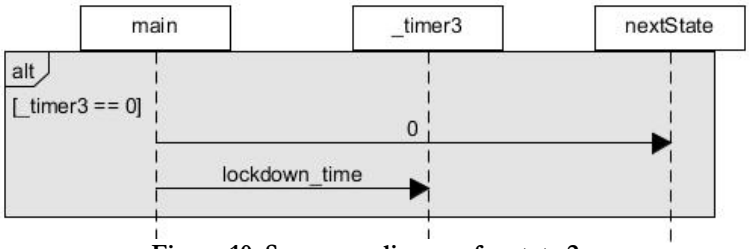

Figure 10: Sequence diagram for state 2

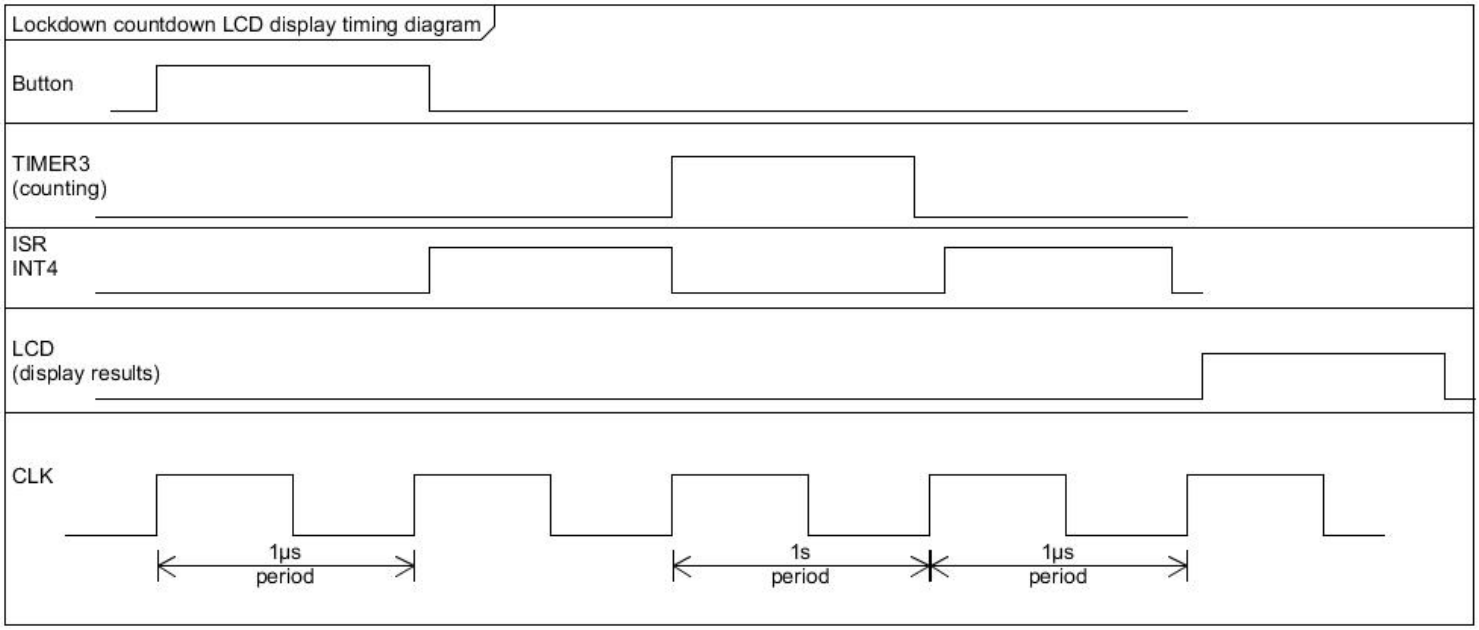

Figure 11: Timing diagram for displaying lockdown count on LCD

3. Test methodology and detailed analysis of results

The testing of DLS was done incrementally. This means that after a base feature, e.g. registering input from keypad was implemented, appropriate tests were carried out to ensure that the functionality is as desired. See table 2 for the tests that were carried out for each component of the system.

Table 2: Test descriptions of all major components

Device	Test Description
1. LEDs	1. Connected LEDs as shown in Figure 4. 2. Wrote a simple program that turned on and off LEDS using a delay of 1 second in while-loop.
2. Keypad	1. Connected keypad as shown in Figure 4. 2. Set up necessary code in order to detect keypress in one column and one row. 3. Expanded code to include four columns and four rows. 4. Each keypress light a different combination of LEDs,
3. LCD	1. Connected LCD as shown in Figure 4. 2. Started from samplecode provided by R.A. (All credit to him) with basic functionality 3. Displayed custom messages
4.Push-button	1. Connected push-button as shown in Figure 4. 2. Wrote simple program that turned all LEDs on when button was pressed
5. Servo motor	1. Connected servo motor as shown in Figure 4. 2. Programmed TIMER4 with correct settings 3. Wrote a simple program that spins motor some degrees positive, then back to 0 degrees.

However, it is worth noting that these tests were carried out while other devices were successfully connected and tested to the microcontroller to ensure as close context as possible during testing. It is not sufficient to test components individually, especially when a combination of device connectivity is essential for desired system function. The tests carried out were mostly successful after first or second attempt.

Moreover, a range of test cases were tested in order to ensure that use cases in Figure 1 were successfully implemented. This led to correct function of system within system use-context. Furthermore, the system can be further tested for test cases that indirectly affect normal operation of the system, such as push-button presses when least expected and keypad presses in special timeframes.

4. Critical evaluation and conclusion

The developed system is a fine prototype that is fully functional in terms of use cases depicted in Figure 1 and requirements given in table 1. The development method was iterational, making it easier to add or subtract functionality at will and troubleshooting problems related to each hardware component. Furthermore, testing of system ensured proper functionality. The developed design fully sports the design requirements as well as functional requirements set. It can function as intended without failure given proper operational environment and matches very well with the intended design. The developed system is well suited as a proof-of-concept prototype for its purpose.

However, there are still several things that has great potential for improvement. First of all, the prototype is prone to hacking by various number of methods. As all connections (wires) are exposed, it is not a difficult task to gain access by pulling or pushing correct cables at peculiar times. Furthermore, to ensure system security, a more robust and secure algorithm for checking password should be implemented. It is also favorable to design a better protoboard, perhaps PCB, that can be mounted in a casing that shields the system from hackers and environmental wear and tear. It would also be beneficial to design a better power deliver subsystem that makes the overall system more portable to use. Portability can in turn open more doors of applicability. In addition to this, it is possible to add other forms of secure access protocols and devices such as RFID or biometrics like fingerprint scanning.

It is also possible to further improve the prototype in terms of hardware. Laser cutting a framework of a small house with a door that sports flexible hinges, and tying a thread to the door and servo motor could prove a better overall contextual prototype.

I have learned a lot through this assignment. It has helped me to fathom how electronics and embedded systems are rooted in almost all technology present and a mastery in this field will prove a great asset for future technological advancements. I have learnt how timers, PWM, interrupts, registers, microprocessors and microcontrollers work. In addition to that, I have learnt more about computer architecture like CPUs, which is the heart of any microprocessor or controller.

5. Program listings

Preamble

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 1000000L
#include <util/delay.h>
#include <stdbool.h>

// Defining our password size: (PS! Remember to set password of correct length
in "passw" array as well)
#define    passw_size 4
// Defining lockdown time in seconds
#define lockdown_time 60
// Defining timeout time in seconds
#define timeout_time 15

// Variable to hold state value.
volatile int nextState = 0;

// Variable that gets true if timer interrupt occured
volatile bool _intrFlag = false;

// Variable that gets true if push button interrupt occured (i.e. lockdown mode
for 1 min).
volatile bool _intrFlag_EG = false;

// Variable to be indirectly used in conjunction with TIMER1, to display timer
on LCD
volatile int _timer1 = timeout_time; // The value is 15 seconds so users have
enough time to input password.

// Variable to be indirectly used in conjunction with TIMER3, to display timer
on LCD
volatile int _timer3 = lockdown_time; // The value is 60 seconds for lockdown
mode.


#include "BorrowedMisc_code.h" // ATTENTION: ALL CODE IN THIS HEADER FILE IS
ORIGINALLY WRITTEN BY R. ANTHONY. ALL CREDITS GO TO HIM.

// This library is where I've stored the declaration of my methods and keywords:
#include "Singh_library.h"

// Array to hold our predefined password. P1 means switch 1 (upper leftmost
switch on keypad), P2 means switch 2 and so on.
 unsigned char passw[passw_size] = {P1, P1, P1, P1};

// Array to hold our input (switch presses, e.g. S1, S2 and so forth)
unsigned char input[passw_size];
```

Main.c

```
/* ATTENTION: THIS CODE INCLUDES SAMPLE CODE NOT AUTHORED BY ME.
 * ALL CREDITS FOR THAT CODE TO ORIGINAL AUTHOR RICHARD ANTHONY (R.A.)
 * I HAVE ONLY USED AND MODIFIED CODE WRITTEN BY HIM TO BETTER SUIT MY PROJECT.
 * THE BORROWED CODE CAN BE FOUND IN "BorrowedMisc_code.h".
 */

int main(void)
{
	// Initial (general) Setup. Consists of sub-setups.
	IS();

	// Filling the input array with null=0xFF values
	for (int i = 0; i < passw_size; i++){input[i]=null;}

	// Declaring a pointer to our input array. This pointer can be used to pass
in functions instead of whole array.
	unsigned char (*p_input)[passw_size] = &input;

	// Declaring a pointer to our password array. This pointer can be used to
pass in functions instead of whole array.
	unsigned char  (*p_passw)[passw_size] = &passw;
```

```
 while (1)
    {
            // Implementing state machine:
            switch (nextState)
            {
                  case 0: /* STATE 0: THIS STATE SHOWS "INPUT PASSWORD" ON LCD
AND REGISTERS THE KEY PRESSES TO ARRAY "input". */

/* In order to write to the screen, we first clear the screen,
then set the cursor to left, upper corner and write our sentence*/

                  // Clearing screen:
                  LCD_Clear();
                  // Setting the cursor:
                  LCD_Home();

                  // Write to the top row: "Input Password:"
                  LCD_WriteString(" Input Password");
                  LCD_SetCursorPosition( 6 /*0 - 40 */,  1 /*0 for top row, 1 for
bottom row*/);
                  LCD_WriteString("Time: 15s");

                  // Resetting _timer1 and _timer3:
                  _timer1 = timeout_time;
                  _timer3 = lockdown_time;

/* In order to register the key presses to an array, we start with a for loop.
Inside the for-loop, before a switch press is parsed into our "input" array,
we check whether there has been any interrupts signaling that 15 seconds has
passed yet. If there are no flags, then we proceed with registering switch press
0-3
to their respective places in array "input".
To see how AssignSV works, see its implementation in the header file
"Singh_library.h".

After that, program checks whether the statevalue is the same, i.e. if a decision
to change states has been made or not. If the statevalue is the same (nextState
= 0) stars get printed to LCD for each digit of the password stored in "input"
array. If not, then it proceeds to the next if-statement. That is, if for-loop
variable 'i' and the statevalue 'nextState' is equal to 0 (i.e. first switch
press and we are to stay in state 0), then TIMER1 starts. Before it is started,
it is reset, just to be sure that nothing is faulty. This is timer 1, a 16-bit
timer that needs no overflow counting in order to count to 1 or 15 seconds. With
timer0, overflow counting and remainder 'ticks' are necessary. These things mess
up the math since they incl. decimal points for counting to 1second. Therefore
I used timer 1. The way I have chosen to apply this timer is to make it count 1
second, display that second on the LCD (as countdown) and issue a timeout after
15 seconds. See .h file "Singh_library.h" for the timer's declaration and
implementation. A interrupt service routine has been made for the timer. The ISR
is called whenever timer counts to 1 (in CTC mode). See the bottom of this file
for ISRs.*/
```

```
for (int i = 0; i < passw_size; i++)
{

      if (!_intrFlag && !_intrFlag_EG){
            input[i]=AssignSV();
// Program waits (while-loop) in AssignSV() until a switch press has been
detected.
// This function returns that switch press to input[i].


            if (nextState==0 && !_intrFlag_EG){
                  LCD_SetCursorPosition( i+1 /*0 - 40 */,  1 /*0 for top row, 1
for bottom row*/);
// PS! CODE BORROWED FROM R.A. ALL CREDITS TO HIM.
                  LCD_WriteString("*");
            }

            if (i==0 && nextState==0){

                  ResetTimer1();

                  StartTimer1();
// ISR for TIMER1 is responsible for updating LCD with timer countdown.

            }
      }
}

/* The only way to get out of for-loop is if the 15second interrupt happens, or
4 digits were pressed (doesn't matter if they're correct or not). After the for-
loop, the timeout timer (TIMER1) is reset. After that, the program checks if the
interruptflag for the 15s timer and/or for the push button (lockdown mode)
      is set or not. If not set, i.e. input timelimit was not reached or push
button not pushed, the program is to proceed to next state (checking the
password).

The following nested if-statements checks whether any interrupt flags have been
set. "_intrFlag" is an interrupt flag for the 15 second timeout that happens if
user doesn’t input password. “_intrFlag_EG” is an interrupt flag set by the ISR
for INT4 for the pushbutton, meaning push-button has been pressed.*/
```

```
ResetTimer1();

if (_intrFlag || _intrFlag_EG){

      if(_intrFlag && !_intrFlag_EG){
// If the timeout flag is set but not the push button flag, we deny access by
going to state 4 next, clear the flag
// Deny access:
            nextState = 4;

// Clear interrupt flag
            _intrFlag = false;

      } else if(!_intrFlag && _intrFlag_EG){
//  If the push button flag is set but not the timeout flag, we go into
lockdown(State2) and clear the flag.

// Go into Lockdown mode for 1 minute:
                  nextState = 2;

// Clear interrupt flag
                  _intrFlag_EG = false;

                  } else if (_intrFlag && _intrFlag_EG){
// In the odd case that timeout flag and push button flag is set, we go into
lockdown as previous if-statement shows

// Go into Lockdown mode for 1 minute:
                        nextState = 2;

// Clear interrupt flag
                        _intrFlag_EG = false;
// Clear interrupt flag
                        _intrFlag = false;
                  }

}     else{
// if no interrupts have been triggered at all, we go on to check the input
password in state 1

// Go to state 1 to check input password
            nextState = 1;

      }

break; /* END OF STATE 0 */
```

```
case 1: /* The purpose of this state is to check the input on keypad against
predefined password in array "passw". After password is checked, system goes to
state 3 or 4, depending if access was given or not.*/

CheckPassword((*p_passw), (*p_input));

break; /* END OF STATE 1 */
```

```
case 2: // The purpose of this state is to set the system in lockdown mode for
1 minute and then restart program (state 0)-----------------------------

/* ISR for TIMER3 gets called for every 1 sec after button is pressed. Push
button is connected to INT4 (PE4) and once button is pressed, the interrupt
service routine for INT4 gets called. The ISR for INT4 starts timer3 if global
variable _timer3=60. ISR for timer 3 takes  care of displaying timer on LCD.
Keep in mind that the whole period we are in lockdown mode, we are actually in
this state (state 2). The only thing that is happening in this state is ISR for
timer3 is called every sec, and we are executing the code below. The code below
checks whether global variable _timer3 is equal to 0, since it decrements with
1 for every ISR TIMER3 call. That is why _timer3 variable is initialised at 60
before main function. If _timer3 is equal to 0, meaning lockdown is done, we
restart program (state 0) and reset _timer3 to 60. */

// if the counter has reached 60 to 0
if (_timer3 == 0){
// Start program again, that is state 0
      nextState = 0;

// Set timer3 to 60 again
      _timer3=lockdown_time;
}

break;
```

```
case 3: // STATE 3: ACCESS GRANTED-----------------------------------------------
-----------------------------------------

AccessGranted();

// Checking which state to go to:
if (_intrFlag_EG) // If push button flag is set
{
// Go to lockdown state
      nextState = 2;

// Clear interrupt flag
      _intrFlag_EG = false;

      break;

}

// "Open door" by showing that servo motor is turning.
SpinMotor();

// Go to state 0
nextState = 0;

break;
```

```
case 4: // STATE 4: ACCESS DENIED-------------------------------------------
-----------------------------------------

AccessDenied();

// Checking which state to go to:
if (_intrFlag_EG) // If push button flag is set
{
// Go to lockdown state
      nextState = 2;

// Clear interrupt flag
      _intrFlag_EG = false;

      break;

}


// Go to state 0
nextState = 0;

break;



case 5: // Reserved for future use


// Start program again, that is state 0
nextState = 0;
break;


default:

// Start program again, that is state 0
nextState = 0;
break;

      }
   }
}
```

```
// Interrupt Service Routine for timer1, channel A (Timeout timer)
ISR(TIMER1_COMPA_vect){
/* The purpose of this interrupt service routine is to hold count of time,
display countdown on LCD and trigger timeout if 15 secs has been counted */

      // A second has passed because ISR has now been called;
      // Decrement the "timer value":
      _timer1--;

      // Setting the cursor for writing remaining time
      LCD_SetCursorPosition( 6 /*0 - 40 */,  1 /*0 for top row, 1 for bottom
row*/);
// PS! CODE BY R.A. ALL CREDITS GO TO HIM.

// A switch case for the timer. Displays countdown on LCD.
// When timer is 0, timeout happens and timer is reset.

      switch (_timer1)
      {
            case 14:

            LCD_WriteString("Time: 14s");
            PORTA = 0x0e;

            break;

            case 13:

            LCD_WriteString("Time: 13s");
            PORTA = 0x0d;

            break;

            case 12:

            LCD_WriteString("Time: 12s");
            PORTA = 0x0c;

            break;

            case 11:

            LCD_WriteString("Time: 11s");
            PORTA = 0x0b;

            break;

            case 10:

            LCD_WriteString("Time: 10s");
            PORTA = 0x0a;

            break;
```

```
case 9:

LCD_WriteString("Time: 9s ");
PORTA = 0x09;

break;

case 8:

LCD_WriteString("Time: 8s ");
PORTA = 0x08;

break;

case 7:

LCD_WriteString("Time: 7s ");
PORTA = 0x07;

break;

case 6:

LCD_WriteString("Time: 6s ");
PORTA = 0x06;

break;

case 5:

LCD_WriteString("Time: 5s ");
PORTA = 0x05;

break;

case 4:

LCD_WriteString("Time: 4s ");
PORTA = 0x04;

break;

case 3:

LCD_WriteString("Time: 3s ");
PORTA = 0x03;

break;

case 2:

LCD_WriteString("Time: 2s ");
```

```
PORTA = 0x02;

break;

case 1:

LCD_WriteString("Time: 1s ");
PORTA = 0x01;

break;

case 0:

// Set timeout flag since timeout has occured
_intrFlag=true;

// Clearing screen:
LCD_Clear();
// Setting the cursor:
LCD_Home();

// Write to the top row: "Input Password:"
LCD_WriteString("    Timeout!");
PORTA=0x3C;
_delay_ms(1000); // hold screen for 1 second
PORTA=0x00;

// Stop Timer1
ResetTimer1();
```

```
            // Reset _timer1 global variable
            _timer1 = timeout_time;

            break;

            default:
            // Clearing screen:
            LCD_Clear();
            // Setting the cursor:
            LCD_Home();

            // Write to the top row: "Input Password:"
            LCD_WriteString("FAULT. SEE ISR.");



            break;


        }


}
```

```
// Interrupt Service Routine for timer3, channel A (Lockdown mode timer)
ISR(TIMER3_COMPA_vect)
{
	/* The purpose of this interrupt service routine is to hold count of
lockdown time, display countdown on LCD and restart program after countdown is
done
	*/

	// A second has passed because ISR has now been called;
	// Decrement the "timer value":
	_timer3--;

	// Setting the cursor for writing remaining time
	LCD_SetCursorPosition( 6 /*0 - 40 */,  1 /*0 for top row, 1 for bottom
row*/);
// PS! CODE BY R.A. ALL CREDITS GO TO HIM.

	// A switch case for the timer. Displays countdown on LCD.
	// When timer is 0, timeout happens and timer is reset.

	switch (_timer3)
	{
		case 60:

		LCD_WriteString("Time: 60s");


		break;

		case 59:

		LCD_WriteString("Time: 59s");


		break;

		case 58:

		LCD_WriteString("Time: 58s");


		break;

		case 57:

		LCD_WriteString("Time: 57s");


		break;

		case 56:

		LCD_WriteString("Time: 56s");
```

```
break;
case 55:
LCD_WriteString("Time: 55s");

break;
case 54:
LCD_WriteString("Time: 54s");

break;
case 53:
LCD_WriteString("Time: 53s");

break;
case 52:
LCD_WriteString("Time: 52s");

break;
case 51:
LCD_WriteString("Time: 51s");

break;
case 50:
LCD_WriteString("Time: 50s");

break;
case 49:
LCD_WriteString("Time: 49s");

break;
```

```
case 48:
LCD_WriteString("Time: 48s");

break;
case 47:
LCD_WriteString("Time: 47s");

break;
case 46:
LCD_WriteString("Time: 46s");

break;
case 45:
LCD_WriteString("Time: 45s");

break;
case 44:
LCD_WriteString("Time: 44s");

break;
case 43:
LCD_WriteString("Time: 43s");

break;
case 42:
LCD_WriteString("Time: 42s");

break;
case 41:
LCD_WriteString("Time: 41s");
```

```
break;
case 40:
LCD_WriteString("Time: 40s");
break;
case 39:
LCD_WriteString("Time: 39s ");
break;
case 38:
LCD_WriteString("Time: 38s ");
break;
case 37:
LCD_WriteString("Time: 37s ");
break;
case 36:
LCD_WriteString("Time: 36s ");
break;
case 35:
LCD_WriteString("Time: 35s ");
break;
case 34:
LCD_WriteString("Time: 34s ");
break;
case 33:
```

```
LCD_WriteString("Time: 33s ");

break;

case 32:

LCD_WriteString("Time: 32s ");

break;

case 31:

LCD_WriteString("Time: 31s ");

break;

case 30:

LCD_WriteString("Time: 30s");

break;

case 29:

LCD_WriteString("Time: 29s");

break;

case 28:

LCD_WriteString("Time: 28s");

break;

case 27:

LCD_WriteString("Time: 27s");

break;

case 26:

LCD_WriteString("Time: 26s");
```

```
break;
case 25:
LCD_WriteString("Time: 25s");

break;
case 24:
LCD_WriteString("Time: 24s");

break;
case 23:
LCD_WriteString("Time: 23s");

break;
case 22:
LCD_WriteString("Time: 22s");

break;
case 21:
LCD_WriteString("Time: 21s");

break;
case 20:
LCD_WriteString("Time: 20s");

break;
case 19:
LCD_WriteString("Time: 19s");

break;
case 18:
```

```
LCD_WriteString("Time: 18s");

break;

case 17:

LCD_WriteString("Time: 17s");

break;

case 16:

LCD_WriteString("Time: 16s");

break;

case 15:

LCD_WriteString("Time: 15s");

break;

case 14:

LCD_WriteString("Time: 14s");

break;

case 13:

LCD_WriteString("Time: 13s");

break;

case 12:

LCD_WriteString("Time: 12s");

break;

case 11:

LCD_WriteString("Time: 11s");

break;
```

```
case 10:
LCD_WriteString("Time: 10s");

break;
case 9:
LCD_WriteString("Time: 9s ");

break;
case 8:
LCD_WriteString("Time: 8s ");

break;
case 7:
LCD_WriteString("Time: 7s ");

break;
case 6:
LCD_WriteString("Time: 6s ");

break;
case 5:
LCD_WriteString("Time: 5s ");

break;
case 4:
LCD_WriteString("Time: 4s ");

break;
case 3:
LCD_WriteString("Time: 3s ");
```

```
            break;

            case 2:

            LCD_WriteString("Time: 2s ");


            break;

            case 1:

            LCD_WriteString("Time: 1s ");


            break;

            case 0:

            // Turning off all LEDs that were set because of push button press
(ISR for INT4)
            PORTA = 0;

            // Resetting TIMER3
            ResetTimer3();


            break;

            default:
            // Clearing screen:
            LCD_Clear();
            // Setting the cursor:
            LCD_Home();

            // Write to the top row: "Input Password:"
            LCD_WriteString("FAULT. SEE ISR3.");


            break;
      }
}
```

```
// Interrupt Service Routine for push button (INT4), that sets system in lockdown
mode for 1 minute
ISR(INT4_vect){

	if (_timer1<timeout_time) // if the user pushes push-button when typing
password
	{
		// Reset timeout timer
		/* Turn of Timer1 (this is valid for the situation when user starts
	pressing switches to input passcode and then pushes red push-button for
	lockdown)*/
		ResetTimer1();
	}

	if(_timer3==lockdown_time){

		// Light up orange leds
		PORTA=0x3C;

		// Start Timer3 to be used for lockdown mode
		StartTimer3();

		//Write to the LCD that system is in lockdown mode
		// Clearing screen:
		LCD_Clear();
		// Setting the cursor:
		LCD_Home();
		LCD_WriteString("  LOCKDOWN MODE");
		// Setting the cursor for writing remaining time
		LCD_SetCursorPosition( 6 /*0 - 40 */,  1 /*0 for top row, 1 for bottom
row*/); // PS! CODE BY R.A. ALL CREDITS GO TO HIM.
		LCD_WriteString("Time: 60s");

		// Push button has now been set so we set the flag for push button
		_intrFlag_EG = 1;
	}
}
```

Singh_library.h

```
#ifndef  SINGH_LIBRARY_H_
#define  SINGH_LIBRARY_H_

// Keyword for no switch pressed:
#define    null  0xFF
// Keywords for different switchpresses:
//                     c-76543210
#define    S1          0b11100111
#define    S2          0b11010111
#define    S3          0b10110111
#define    S4          0b01110111
#define    S5          0b11101011
#define    S6          0b11011011
#define    S7          0b10111011
#define    S8          0b01111011
#define    S9          0b11101101
#define    S10         0b11011101
#define    S11         0b10111101
#define    S12         0b01111101
#define    S13         0b11101110
#define    S14         0b11011110
#define    S15         0b10111110
#define    S16         0b01111110


// Defining keywords for password:
#define    P1          0x01
#define    P2          0x02
#define    P3          0x03
#define    P4          0x04
#define    P5          0x05
#define    P6          0x06
#define    P7          0x07
#define    P8          0x08
#define    P9          0x09
#define    P10         0x0A
#define    P11         0x0B
#define    P12         0x0C
#define    P13         0x0D
#define    P14         0x0E
#define    P15         0x0F
#define    P16         0x10
```

```
/*-------------------------------------------------------|
Methods used setting up stuff; ports, interrupts etc:
-------------------------------------------------------*/
void IS(void); // Initial Setup. Calls PS() and LCDS().

void PS(void); // Setup for ports

void LCDS(void); // Setup for LCD. PS! CODE IN HERE WRITTEN BY R.A. ALL CREDITS
GO TO HIM.



/*-------------------------------------------------------|
Methods for detecting switch presses:
-------------------------------------------------------*/
unsigned char DetectSwitchPress();// Container method for detecting switch press.

unsigned char Detect0(unsigned char, unsigned char);// Method for detecting if
a switch in row 0 has been pressed
unsigned char Detect1(unsigned char, unsigned char);// Method for detecting if
a switch in row 1 has been pressed
unsigned char Detect2(unsigned char, unsigned char);// Method for detecting if
a switch in row 2 has been pressed
unsigned char Detect3(unsigned char, unsigned char);// Method for detecting if
a switch in row 3 has been pressed



/*-------------------------------------------------------|
Methods used to store each switch press to its designated variable:
-------------------------------------------------------*/
unsigned char AssignSV();
```

```
/*-------------------------------------------------------|
Methods used to initialise and start timers:
-------------------------------------------------------*/
void StartTimer1(void); // Used for 15s countdown during password input on switchpad
void StartTimer3(void); // Used for 60s countdown during lockdown mode of system. Stated mode is triggered by red push button next to LCD
void StartTimer4(void); // Used for PWM - servo motor. Motor spins when access is granted (correct password input)

/*-------------------------------------------------------|
Methods used to stop and clear timers:
-------------------------------------------------------*/
void ResetTimer1(void);
void ResetTimer3(void);
void ResetTimer4(void);

/*-------------------------------------------------------|
Methods used to check the input password from keypad
-------------------------------------------------------*/
void CheckPassword(unsigned char (*p_passw)[], unsigned char (*p_input)[]);


/*-------------------------------------------------------|
Methods used if access is granted:
-------------------------------------------------------*/
void AccessGranted(void);
void SpinMotor(void);



/*-------------------------------------------------------|
Methods used if access is not granted:
-------------------------------------------------------*/
void AccessDenied(void);
```

```
/*-------------------------------------------------------|
FUNCTION IMPLEMENTATIONS HERE ON OUT
-------------------------------------------------------*/

void IS(){

	PS(); // General Port Setup for Keypad, Servo motor and Interrupt 4 (for ext. intr.)
	LCDS(); // LCD Setup. PS! THIS METHOD INCLUDES CODE ORIGINALLY WRITTEN BY R.A. ALL CREDIT GOES TO HIM.

	sei(); // Enable global interrupts.

}

void PS(){

	// Rows on the keypad are outputs while columns are inputs (will be used to detect logic level and determine which switch is pressed)
	// Set bits C3 downto C0 as output (rows R3-R0) and C7 downto C0 as input (columns C3-C0)
	DDRC = 0b0001111;
	// Set port C as high; all rows will begin with having high level, the columns need to be pulled up to high level before any switch is pressed.
	PORTC= 0xFF;

	// PortA is for the LEDs. Their data direction register is set high (output) and the LEDs will initially be off.
	DDRA = 0xFF; // Setting PortA as output
	PORTA = 0x00; // Output signal is initially low (0)

	// PortH3 - 6 is for PWM for servo motor. This port is connected to OC4A and can be used for PWM signaling.
	// Once access is granted, servo motor will spin to indicate door opening.
	DDRH = (1 << PORTH3); // Setting PH3 to output
	//PORTB = (0 << PORTB7); // Output signal is initially low (0)

	// Using INT4 (PE4) as ext. intr. for push button that sets system in lockdown mode
	DDRE = (0 << PORTE4); // Setting PE4 as input
	PORTE = (1 << PORTE4); // Internal pull-up

	EIMSK = 0x00; // Clearing external interrupt mask register before setting EICRB
	EICRB = (1<<ISC41) | (1<<ISC40); // a rising edge (low to high) will trigger an interrupt

	EIMSK = (1<<INT4); // Enabling ext. intr. request for intr. 4

}
```

```
// PS! CODE IN LCDS() CONCISTS OF SUB-ROUTINES WRITTEN BY R.A. ALL CREDITS TO
HIM.
void LCDS(){

      // LCD Setup; 2 line mode and 5*8 pixel (small font)
      LCD_Initialise(true, false); // from BorrowedMisc_code.h, see crediting in
preamble.

      LCD_ShiftDisplay(false /*true = ON false = OFF*/, true /*true = shift right,
false = shift left*/);
      LCD_Display_ON_OFF(true  /*Display  ON*/,  false  /*Cursor  OFF*/,  false
/*Cursor Position OFF*/);

      // Clearing screen:
      LCD_Clear();
      // Setting the cursor:
      LCD_Home();

      // Write to the top row: "Input Password:"
      LCD_WriteString(" Input Password");

}
```

```
unsigned char DetectSwitchPress(){
      /* We need two variables; one to hold value for which row we will be giving
a high level,
       * and one to temporarily hold value for switch press:
      */

      unsigned char R;
      unsigned char _SV=null;

// As long as no switch is pressed, the program will keep waiting for switch
press,
// except if an interrupt flag for timeout is set:
while (_SV==null)
{

      // Method for detecting switch press in row 0:
      _SV = Detect0(R, _SV);

      // if a key has been pressed or the interrupt flag for timeout has been
set, return the switch value _SV:
      if (_SV!=null || _intrFlag || _intrFlag_EG){return _SV;}

      // Method for detecting switch press in row 1:
      _SV = Detect1(R, _SV);

      // if a key has been pressed or the interrupt flag for timeout has been
set, return the switch value _SV:
      if (_SV!=null || _intrFlag || _intrFlag_EG){return _SV;}

      // Method for detecting switch press in row 2:
      _SV = Detect2(R, _SV);

      // if a key has been pressed or the interrupt flag for timeout has been
set, return the switch value _SV:
      if (_SV!=null || _intrFlag || _intrFlag_EG){return _SV;}

      // Method for detecting switch press in row 3:
      _SV = Detect3(R, _SV);

      // if a key has been pressed or the interrupt flag for timeout has been
set, return the switch value _SV:
      if (_SV!=null || _intrFlag || _intrFlag_EG){return _SV;}

}

}
```

```
unsigned char Detect0(unsigned char R, unsigned char _SV){
	// The purpose of this subroutine is to determine whether a switch in row
0 has been pressed.

	R = 0x01; // This is the value for switch S1 at row 0

	/*Configuring port C with all high levels.
	* Then only row 0 (bit C3) is sent a low level.
	* This way the column connected to row 0 through the switch press will also
have a low logic level.
	* Based on that we can detect which key is pressed
	*/

	PORTC = 0xFF;
	PORTC &= 0b11110111; // row 0 is given low level

	/*After giving row 0 logic 0, we start looking at which columns
	* (these are inputs to MCU) also have a logic 0 present
	*/
	switch(PINC){

		case S1:   // This means that row 0 has a low, and column 0 has a
low. This means that S1 has been pressed
			_SV = R;		// The temporary switch value is given value
of 0x01 which is the hex value for switch S1.
			break;

		case S2:
			_SV = R + 1;	// S2
			break;

		case S3:
			_SV = R + 2;	// S3
			break;

		case S4:
			_SV = R + 3;	// S4
			break;

		default:
			_SV = null; // No key is pressed in this row
			break;
	}

	return _SV;
}
```

```
unsigned char Detect1(unsigned char R, unsigned char _SV){

	// Scanning row 1	(bit C2)
	R = 0x05; // This is the value for switch S5 at row 1

	PORTC = 0xFF;
	PORTC &= 0b11111011;
	// Scanning columns
	switch(PINC){

		case S5:
		_SV = R;		// S5
		break;

		case S6:
		_SV = R + 1;		// S6
		break;

		case S7:
		_SV = R + 2;		// S7
		break;

		case S8:
		_SV = R + 3;		// S8
		break;

		default:
		_SV = null;
		break;
	}

	return _SV;
}
```

```
unsigned char Detect2(unsigned char R, unsigned char _SV){

	R = 0x09; // This is the value for switch S1 at row 2

	PORTC = 0xFF;
	PORTC &= 0b11111101; // row 2 is given low level


	switch(PINC){

		case S9:
			_SV = R;
			break;

		case S10:
			_SV = R + 1;
			break;

		case S11:
			_SV = R + 2;
			break;

		case S12:
			_SV = R + 3;
			break;

		default:
			_SV = null; // No key is pressed in this row
			break;
	}

	return _SV;
}
```

```
unsigned char Detect3(unsigned char R, unsigned char _SV){


	R = 0x0D;

	PORTC = 0xFF;
	PORTC &= 0b11111110;


	switch(PINC){

		case S13:
			_SV = R;
			break;

		case S14:
			_SV = R + 1;
			break;

		case S15:
			_SV = R + 2;
			break;

		case S16:
			_SV = R + 3;
			break;

		default:
			_SV = null; // No key is pressed in this row
			break;
	}

	return _SV;
}
```

```
unsigned char AssignSV(){

      unsigned char SV = DetectSwitchPress();
      DebounceDelay();
// PS! THIS METHOD BORROWED FROM R.A. ALL CREDITS GO TO HIM.

      /* After returning from "DetectSwitchPress()" the program checks whether
       the interruptflag for timeout or lockdown mode has been set or not. If it
has been set, the switch value
       that is returned and given to "input[i]" will be equal to null (oxFF).
      */
      if (_intrFlag || _intrFlag_EG)
      {
            SV=null;

            return SV;
      }

      return SV;
}
```

```
/* Timer 1 is used as timeout timer.
It counts to 1 sec (CTC mode) and triggers ISR
I am applying it to 15 sec so that user has enough time to input password before
timeout
*/
void StartTimer1(void){
	// TCCRnA= COMnA2 | COMnA0 | COMnB1	| COMnB0 | COMnC1 | COMnC0 | WGMn1
| WGMn0 |
	// In order to have normal port operation, all "comms" are cleared.
	// In order to clear timer on compare match (CTC), last two bits need to
be 0
	TCCR1A = 0x00;

	// TCCR3B= ICNCn | ICESn |	X	| WGMn3 | WGMn2 | CSn2 | CSn1 | CSn0 |
	// In order to have CTC, WGMn2 needs to be set.
	// In order to have 1024 prescaling, CS bits have to be 101
	//TCCR1B = 0b00001101;// CTC mode, use 1024 prescaler
	TCCR1B = 0b00001011;  // CTC mode, use 64 prescaler

	TCCR1C = 0x00; // not done anything with

	// System clock is 1 MHz, timer "clock" is 1MHz/64= 15625Hz
	// to get 1 seconds in total, we need to count to 15625 in decimal
	// 15625 is 0x3D09 in hex
	OCR1AH = 0x3D; // Each of these store half of the total time
	OCR1AL = 0x09;


	TCNT1H = 0x00;   // Timer/Counter count/value registers (16 bit) TCNT1H and
TCNT1L
	TCNT1L = 0x00;

	// TIMSK1= X | X |	ICIE1	| X | OCIE1C | OCIE1B | OCIE1A | TOIE1 |
	// In order to have an interrupt generated by output compare match on
channel A, OCIE1A needs to be set.
	TIMSK1 = 0b00000010;

}
```

```
void ResetTimer1(void){
	// TCCR1A= COM1A1 | COM1A0 | COM1B1	| COM1B0 | COM1C1 | COM1C0 | WGM11
| WGM10 |
	// In order to have normal port operation, all "comms" are cleared.
	// In order to clear timer on compare match (CTC), last two bits need to
be 0
	TCCR1A = 0x00;

	// TCCR1B= ICNC1 | ICES1 |	X	| WGM13 | WGM12 | CS12 | CS11 | CS10 |
	// In order to have CTC, WGM12 needs to be set.
	// In order to have 1024 prescaling, CS bits have to be 101
	TCCR1B = 0b00000000;	// CTC mode, use 1024 prescaler


	TCCR1C = 0x00; // not done anything with

	// System clock is 1 MHz, timer "clock" is 976.5625=15625/16 Hz
	// to get 15 seconds in total, we need to count to 15*976=14648.4375~14648
in decimal
	// 14648 is 0x3938 in hex
	OCR1AH = 0x00; // Each of these store half of the total time
	OCR1AL = 0x00;

	OCR1BH = 0x00;
	OCR1BL = 0x00;

	TCNT1H = 0b00000000;	// Timer/Counter count/value registers (16 bit)
TCNT1H and TCNT1L
	TCNT1L = 0b00000000;

	// TIMSK1= X | X |	ICIE1	| X | OCIE1C | OCIE1B | OCIE1A | TOIE1 |
	// In order to have an interrupt generated by output compare match on
channel A, OCIE1A needs to be set.
	TIMSK1 = 0b00000000;



}
```

```
/* Timer 3 is used as Lockdown timer.
It counts to 1 sec (CTC mode) and triggers ISR
I am applying it to 60 sec (lockdown time)
*/
void StartTimer3(void){
	// TCCR3A= COM3A2 | COM3A0 | COM3B1    | COM3B0 | COM3C1 | COM3C0 | WGM31
| WGM30 |
	// In order to have normal port operation, all "comms" are cleared.
	// In order to clear timer on compare match (CTC), last two bits need to
be 0
	TCCR3A = 0x00;

	// TCCR3B= ICNC3 | ICES3 |   X    | WGM33 | WGM32 | CS32 | CS31 | CS30 |
	// In order to have CTC, WGM32 needs to be set.
	// In order to have 1024 prescaling, CS bits have to be 101
	//TCCR3B = 0b00001101; // CTC mode, use 1024 prescaler
	TCCR3B = 0b00001011; // CTC mode, use 64 prescaler


	TCCR3C = 0x00; // not done anything with

	// System clock is 1 MHz, timer "clock" is 1MHz/64= 15625Hz
	// to get 1 seconds in total, we need to count to 15625 in decimal
	// 15625 is 0x3D09 in hex
	OCR3AH = 0x3D; // Each of these store "half" of the total time
	OCR3AL = 0x09;


	TCNT3H = 0x00;   // Timer/Counter count/value registers (16 bit) TCNT1H and
TCNT1L
	TCNT3L = 0x00;

	// TIMSK3= X | X |   ICIE3  | X | OCIE3C | OCIE3B | OCIE3A | TOIE3 |
	// In order to have interrupt generated by output compare match on channel
A, OCIE3A is set.
	TIMSK3 = 0b00000010;

}
```

```
void ResetTimer3(void){
	// TCCR1A= COM1A1 | COM1A0 | COM1B1	| COM1B0 | COM1C1 | COM1C0 | WGM11
| WGM10 |
	// In order to have normal port operation, all "comms" are cleared.
	// In order to clear timer on compare match (CTC), last two bits need to
be 0
	TCCR3A = 0x00;

	// TCCR1B= ICNC1 | ICES1 |	X	| WGM13 | WGM12 | CS12 | CS11 | CS10 |
	// In order to have CTC, WGM12 needs to be set.
	// In order to have 1024 prescaling, CS bits have to be 101
	TCCR3B = 0b00000000;	// CTC mode, use 1024 prescaler


	TCCR3C = 0x00; // not done anything with

	// System clock is 1 MHz, timer "clock" is 976.5625=15625/16 Hz
	// to get 15 seconds in total, we need to count to 15*976=14648.4375~14648
in decimal
	// 14648 is 0x3938 in hex
	OCR3AH = 0x00; // Each of these store half of the total time
	OCR3AL = 0x00;

	TCNT3H = 0b00000000;	// Timer/Counter count/value registers (16 bit)
TCNT1H and TCNT1L
	TCNT3L = 0b00000000;

	// TIMSK1= X | X |	ICIE1	| X | OCIE1C | OCIE1B | OCIE1A | TOIE1 |
	// In order to have an interrupt generated by output compare match on
channel A, OCIE1A needs to be set.
	TIMSK3 = 0b00000000;



}
```

```
/* Timer 4 is used for fast PWM to servo motor.
Global variable DT (dutycycle) determines the dutycycle.
https://www.electronicoscaldas.com/datasheet/MG995_Tower-Pro.pdf
Datasheet for servo motor gives that PWM period has to be 20ms (50 Hz freq)
We will not be using any prescaler.
*/
void StartTimer4(void){
	// TCCR4A= COM4A1 | COM4A0 | COM4B1	| COM4B0 | COM4C1 | COM4C0 | WGM41
| WGM40 |
	// We use channel A, so COM4A1 is set. A0 is 0 (non-inverting mode).We will
be using Fast PWM with ICR4 as TOP
	// Therefore WGM is set to 1 1 1 0 (mode 14)
	TCCR4A = 0b10000010;

	// TCCR4B= ICNC4 | ICES4 |	X	| WGM43 | WGM42 | CS42 | CS41 | CS40 |
	// In order to have fast PWM, we will be using mode 14,
	// so WGM has to be set 1 1 1 0
	// We will not be using any prescaler (N=1) so CS = 0 0 1
	TCCR4B = 0b00011001;

	// ICR4 is calc. by ICR4= (F_CPU/(F_PWM*N)) - 1
	// F_CPU=10^6, F_PWM=50, N=1 --> ICR4 = 19999
	ICR4 = 19999; // Period 20ms



}


void ResetTimer4(void){
	TCCR4A = 0;
	TCCR4B = 0;
	ICR4 = 0;
}
```

```
void CheckPassword(unsigned char (*p_passw)[], unsigned char (*p_input)[]){
	/* The purpose of this sub-routine is to check whether the input password matches the predefined
	password in "passw" array. The way it checks this is to see if each digit pressed corresponds
	to respective digit in passw array as well, i.e. input0 should match passw0 and so on.
	For each digit that matches, a variable increments, signifying the correct match of input digit and
	password digit. If the number of correct digits is matches with the length of predefined password,
	then access is granted. Otherwise it is not.
	*/

	int digits_correct=0;

	for (int i = 0; i < passw_size; i++)
	{
		if((*p_input)[i]==(*p_passw)[i]){
			digits_correct++;
		}
	}


	if (digits_correct==passw_size) // We grant access (next state is 3)
	{
		// Access granted:
		nextState = 3;
		//AccessGranted();

	} else{ // We do not grant access (next state is 4)

		// Access not granted:
		nextState = 4;
		//AccessDenied();

	}

}
```

```
void AccessGranted(void){
	//Write to the LCD that Access is granted on top row
	// Clearing screen:
	LCD_Clear();
	// Setting the cursor:
	LCD_Home();
	LCD_WriteString(" Access granted");


	// Flash blue lights, signaling access granted
	PORTA=0xC0;
	_delay_ms(200);
	PORTA=0x00;
	_delay_ms(200);
	PORTA=0xC0;
	_delay_ms(200);
	PORTA=0x00;
	_delay_ms(200);
	PORTA=0xC0;
	_delay_ms(200);
	PORTA=0x00;
	_delay_ms(200);
	PORTA=0xC0;
	_delay_ms(200);
	PORTA=0x00;
}
```

```
void SpinMotor(void){

	//Write to the LCD that Access is granted on top row
	// Clearing screen:
	LCD_Clear();
	// Setting the cursor:
	LCD_Home();
	LCD_WriteString("   Door open");

	// Starting Timer4 for PWM
	StartTimer4();

	// Spin 180 deg
	OCR4A = 950;

	// Let it spin for 2 sec
	_delay_ms(1000);

	//Spin back to 0
	OCR4A = 0;

	// Let it spin for 2 sec
	_delay_ms(1000);

	// Stop timer4
	ResetTimer4();

}
```

```
void AccessDenied(void){
      //Write to the LCD that Access is denied on top row
      // Clearing screen:
      LCD_Clear();
      // Setting the cursor:
      LCD_Home();
      LCD_WriteString(" Access denied");
      DebounceDelay();
// a little delay (same delay as for switch bouncing).PS! CODE BY R.A. ALL
CREDITS TO HIM.

      // Flash red lights, signaling access denied
      PORTA=0x03;
      _delay_ms(200);
      PORTA=0x00;
      _delay_ms(200);
      PORTA=0x03;
      _delay_ms(200);
      PORTA=0x00;
      _delay_ms(200);
      PORTA=0x03;
      _delay_ms(200);
      PORTA=0x00;
      _delay_ms(200);
      PORTA=0x03;
      _delay_ms(200);
      PORTA=0x00;

}



#endif
```

BorrowedMisc_code.h

```
/*
 * BorrowedMisc_code.h


 * Original Author: Richard Anthony
 * ------------------------------------------------------------------------
--------------------------------
 * ATTENTION: ALL CODE IN THIS FILE IS BORROWED FROM ORIGINAL AUTHOR RICHARD
ANTHONY. ALL CREDIT GOES TO HIM.|
 * I'VE ONLY USED HIS MATERIAL TO BETTER SUIT MY CONFIG (PORTS ETC).
 *------------------------------------------------------------------------
--------------------------------
 */

#include <stdbool.h>
#include <string.h>
#define LCD_DisplayWidth_CHARS 16



#ifndef BORROWEDMISC_CODE_H_
#define BORROWEDMISC_CODE_H_

void DebounceDelay()  // This delay is needed because after pressing a key, the
mechanical switch mechanism tends
// to bounce, possibly leading to many key presses being falsely detected. The
debounce delay
// makes the program wait long enough for the bouncing to stop before reading
the keypad again.
{
      for(int i = 0; i < 50; i++)
      {
            for(int j = 0; j < 255; j++);
      }
}
```

```
//------------------------------------------------------------------------
----------------------
// The following methods are from Richard Anthony's "LCD_LibraryFunctions2560.h".
All credit goes to him.
// I've just changed and/or used the implementation of some methods to suit the
ports and configurations I have.
//------------------------------------------------------------------------
----------------------


//Function declarations
// *** 'Private' Functions accessed by other member functions - do not call these
direct from application code ***
void LCD_Write_CommandOrData(bool bCommand /*true = Command, false = Data*/,
unsigned char DataOrCommand_Value);
void LCD_Wait();
void LCD_Enable();
void LCD_Disable();
// *** END of 'Private' Functions accessed by other member functions - do not
call these direct from application code ***

// *** USER functions
void LCD_Initialise(bool bTwoLine/*false = 1 line mode, true =  2 line mode*/,
bool bLargeFont/*false = 5*8pixels, true = 5*11 pixels*/);
void LCD_Display_ON_OFF(bool bDisplayON /*true = ON, false = OFF*/, bool
bCursorON, bool bCursorPositionON); // Turn the LCD display ON / OFF
void LCD_Clear();
void LCD_Home();
void LCD_WriteChar(unsigned char cValue);
void LCD_ShiftDisplay(bool bShiftDisplayON /*true = On false = OFF*/, bool
bDirectionRight /*true = shift right, false = shift left*/);
void LCD_SetCursorPosition(unsigned char iColumnPosition /*0 - 40 */, unsigned
char iRowPosition /*0 for top row, 1 for bottom row*/);
void LCD_WriteString(char Text[]);
// *** END of USER functions
```

```
// Function implementations
// *** 'Private' Functions accessed by other member functions - do not call these
direct from application code ***
void LCD_Write_CommandOrData(bool bCommand /*true = Command, false = Data*/,
unsigned char DataOrCommand_Value)
{
	LCD_Wait();// Wait if LCD device is busy

	// The access sequence is as follows:
	// 1. Set command lines as necessary:
	if(true == bCommand)
	{	// Register Select LOW, and R/W direction LOW
		PORTG &= 0b11111100;  // Clear Read(H)/Write(L) (PortG bit1),Clear
Register Select for command mode (PortG bit0)
	}
	else /*Data*/
	{	// Register Select HIGH, and R/W direction LOW
		PORTG |= 0b00000001;  // Set Register Select HIGH for data mode
(PortG bit0)
		PORTG &= 0b11111101;  // Clear Read(H)/Write(L) (PortG bit1)
	}
	// 2. Set Enable High
	// 3. Write data or command value to PORTL
	// 4. Set Enable Low
	LCD_Enable();
	DDRL = 0xFF;				// Configure PortL direction for
Output
	PORTL = DataOrCommand_Value;	// Write combined command value to port
L
	LCD_Disable();
}

void LCD_Wait()		// Check if the LCD device is busy, if so wait
{				// Busy flag is mapped to data bit 6, so read as port
A bit 6
	PORTG &= 0b11111110;		// Clear Register Select for command mode
(PortG bit0)
	PORTG |= 0b00000010;		// Set Read(H)/Write(L) (PortG bit1)
	DDRL = 0x00;			// Configure PortL direction for Input
(so busy flag can be read)

	unsigned char PINA_value = 0;
	while(PINA_value & 0b10000000);  // Wait here until busy flag is cleared
	{
		LCD_Enable();
		PINA_value = PINL;
		LCD_Disable();
	}
}

void LCD_Enable()
{
	PORTG |= 0b00000100;  // Set LCD Enable (PortG bit2)
```

```
}

void LCD_Disable()
{
	PORTG &= 0b11111011;	// Clear LCD Enable (PortG bit2)
}
// *** END of 'Private' Functions accessed by other member functions - do not call these direct from application code ***

// *** USER functions
void LCD_Initialise(bool bTwoLine/*false = 1 line mode, true =  2 line mode*/, bool bLargeFont/*false = 5*8pixels, true = 5*11 pixels*/)
{	// Note, in 2-line mode must use 5*8 pixels font

	// Set Port L and Port G for output
	DDRG = 0xFF;		// Configure PortG direction for Output
	PORTG = 0x00;		// Clear port G
	DDRL = 0xFF;		// Configure PortL direction for Output
	PORTL = 0x00;		// Clear port L

	unsigned char Command_value = 0b00110000;	// bit 5 'Function Set' command, bit 4 High sets 8-bit interface mode
	if(true == bTwoLine)
	{
		Command_value |= 0b00001000;		// bit 3 high = 2-line mode (low = 1 line mode)
	}
	else
	{	// One-line mode
		if(true == bLargeFont)
		{	// Large font (nested because can only use large font in one-line mode)
			Command_value |= 0b00000100;		// bit 2 high = large font mode 5*11 pixels (low = small font 5*8pixels)
		}
	}

	LCD_Write_CommandOrData(true	/*true = Command, false = Data*/, Command_value);
}

void LCD_Display_ON_OFF(bool bDisplayON /*true = ON, false = OFF*/, bool bCursorON, bool bCursorPositionON) // Turn the LCD display ON / OFF
{
	if(true == bDisplayON)
	{
		if(true == bCursorON)
		{
			if(true == bCursorPositionON)
			{
				// 'Display ON/OFF' function command, Display ON, Cursor ON, Cursor POSITION ON
```

```
                        LCD_Write_CommandOrData(true /*true = Command, false =
Data*/, 0b00001111);
                }
                else
                {
                        // 'Display ON/OFF' function command, Display ON, Cursor
ON, Cursor POSITION OFF
                        LCD_Write_CommandOrData(true /*true = Command, false =
Data*/, 0b00001110);
                }
            }
            else /*Cursor OFF*/
            {
                if(true == bCursorPositionON)
                {
                        // 'Display ON/OFF' function command, Display ON, Cursor
OFF, Cursor POSITION ON
                        LCD_Write_CommandOrData(true /*true = Command, false =
Data*/, 0b00001101);
                }
                else
                {
                        // 'Display ON/OFF' function command, Display ON, Cursor
OFF, Cursor POSITION OFF
                        LCD_Write_CommandOrData(true /*true = Command, false =
Data*/, 0b00001100);
                }
            }
      }
      else
      {
            // 'Display ON/OFF' function command, Display OFF, Cursor OFF, Cursor
POSITION OFF
            LCD_Write_CommandOrData(true /*true = Command, false = Data*/,
0b00001000);
      }
}

void LCD_Clear()            // Clear the LCD display
{
      LCD_Write_CommandOrData(true /*true = Command, false = Data*/, 0b00000001);
      _delay_ms(2);    // Enforce delay for specific LCD operations to complete
}

void LCD_Home()             // Set the cursor to the 'home' position
{
      LCD_Write_CommandOrData(true /*true = Command, false = Data*/, 0b00000010);
      _delay_ms(2);    // Enforce delay for specific LCD operations to complete
}

void LCD_WriteChar(unsigned char cValue)
{    // Write character in cValue to the display at current cursor position
(position is incremented after write)
```

```
	LCD_Write_CommandOrData(false /*true = Command, false = Data*/, cValue);
}

void LCD_ShiftDisplay(bool bShiftDisplayON /*true = On false = OFF*/, bool
bDirectionRight /*true = shift right, false = shift left*/)
{
	if(true == bShiftDisplayON)
	{
		if(true == bDirectionRight)
		{
			LCD_Write_CommandOrData(true /*true = Command, false = Data*/,
0b00000101);
		}
		else /*shift display left*/
		{
			LCD_Write_CommandOrData(true /*true = Command, false = Data*/,
0b00000111);
		}
	}
	else /*ShiftDisplay is OFF*/
	{
		LCD_Write_CommandOrData(true /*true = Command, false = Data*/,
0b00000100);
	}
}

void LCD_SetCursorPosition(unsigned char iColumnPosition /*0 - 40 */, unsigned
char iRowPosition /*0 for top row, 1 for bottom row*/)
{
	// Cursor position is achieved by repeatedly shifting from the home
position.
	// In two-line mode, the beginning of the second line is the 41st position
(from home position)
	unsigned char iTargetPosition = (40 * iRowPosition) + iColumnPosition;
	LCD_Home();
	for(unsigned char iPos = 0; iPos < iTargetPosition; iPos++)
	{
		LCD_Write_CommandOrData(true /*true = Command, false = Data*/,
0b00010100); // Shift cursor left one place
	}
}

void LCD_WriteString(char Text[])
{
	for(unsigned char iIndex = 0; iIndex < strlen(Text); iIndex++)
	{
		LCD_WriteChar(Text[iIndex]);
	}
}
// *** END of USER functions
```

```
#endif /* BORROWEDMISC_CODE_H_ */
```