# Final Project

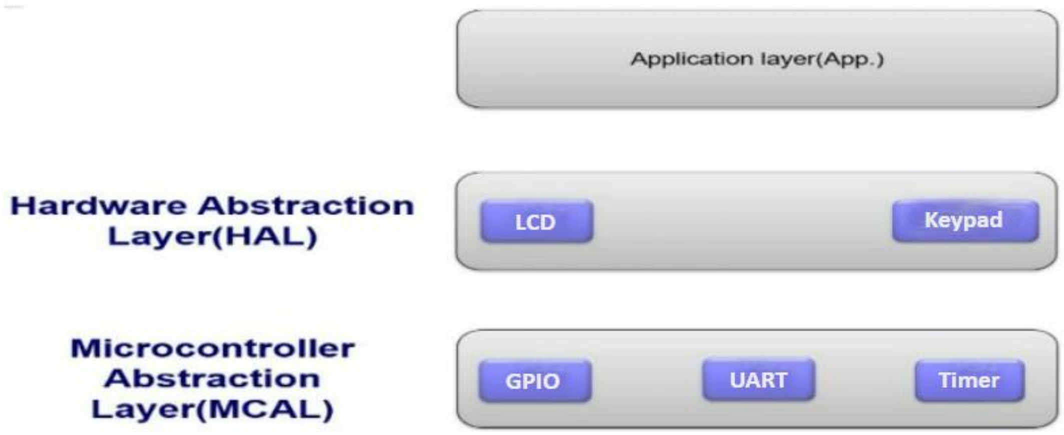
## System Requirements

Implement the **Door Locker Security System** to unlock a door using a password.

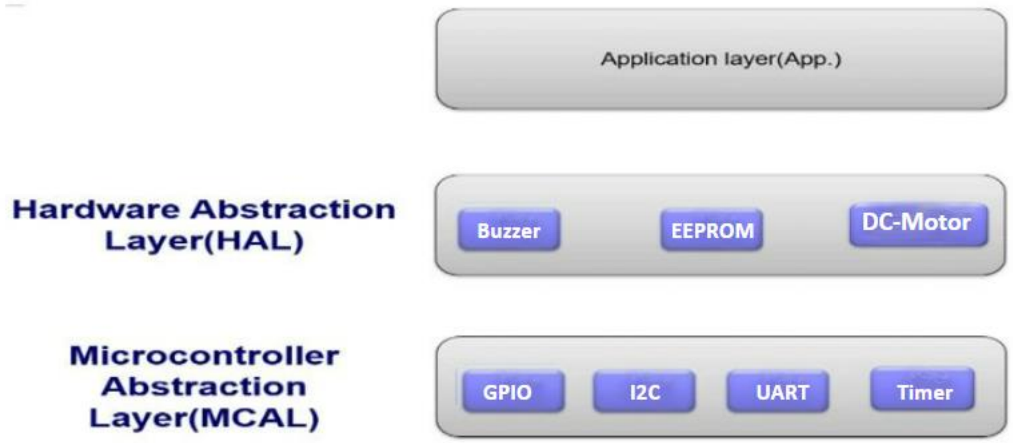
with the specifications listed below:

1) Use two **ATmega32** Microcontrollers with frequency **8Mhz**.
2) The project should be design and implemented based on the layered architecture model as follow:

**𝓜c1 → HMI_ECU (Human Machine Interface)** with 2x16 LCD and 4x4 keypad.


Application layer(App.)
Hardware Abstraction Layer(HAL)
LCD
Keypad
Microcontroller Abstraction Layer(MCAL)
GPIO
UART
Timer

**𝓜c2 → Control_ECU** with EEPROM, Buzzer, and Dc-Motor.


Application layer(App.)
Hardware Abstraction Layer(HAL)
Buzzer
EEPROM
DC-Motor
Microcontroller Abstraction Layer(MCAL)
GPIO
I2C
UART
Timer

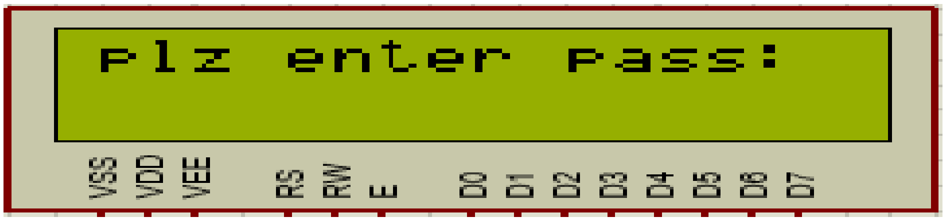
3) **HMI_ECU** is just responsible interaction with the user just take inputs through keypad and display messages on the LCD.

4) **CONTROL_ECU** is responsible for all the processing and decisions in the system like password checking, open the door and activate the system alarm.

5) **System Sequence:**

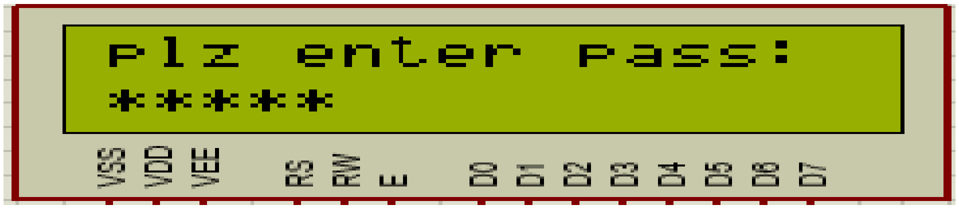
**Step1 – Create a System Password**

- The LCD should display "Please Enter Password" like that:



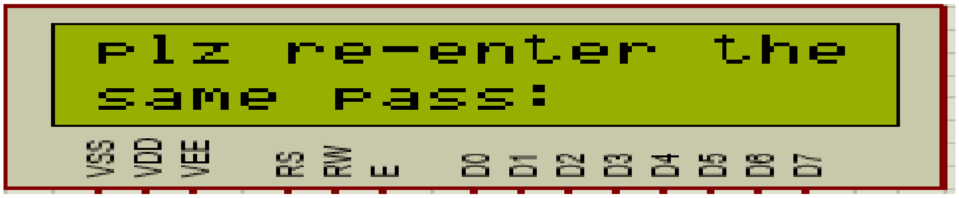
plz enter pass:

- Enter a password consists of 5 numbers, Display * in the screen for each number.



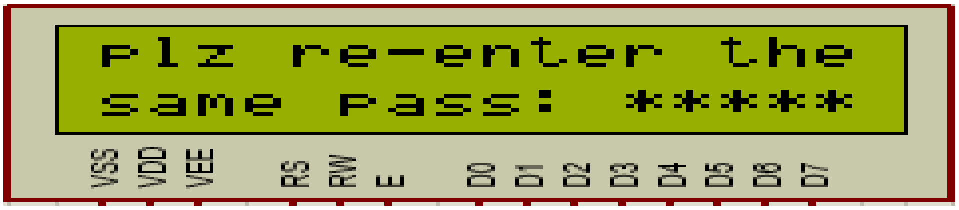
plz enter pass:
*****

- Press **enter** button (choose any button in the keypad as enter button).
- Ask the user to renter the same password for confirmation by display this message "Please re-enter the same Pass":



plz re-enter the
same pass:

- Enter a password consists of 5 numbers, Display * in the screen for each number.
- Press **enter** button (choose any button in the keypad as enter button).



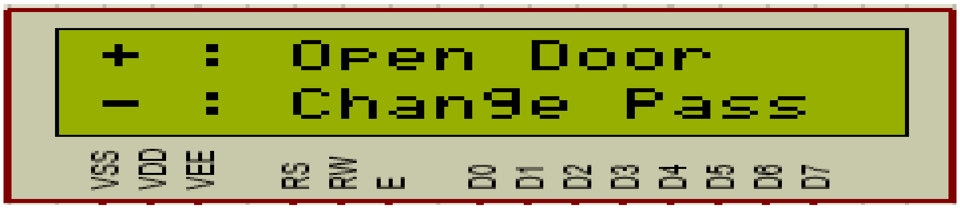
plz re-enter the
same pass: *****

- **HMI_ECU** should send the two passwords to the **Control_ECU** through the **UART**.

- If the two passwords are **matched** then the system has a password now and save it inside the **EEPORM** and go to **Step 2**.
- If the two passwords are **unmatched** then repeat **step 1** again.

**Step2 - Main Options**

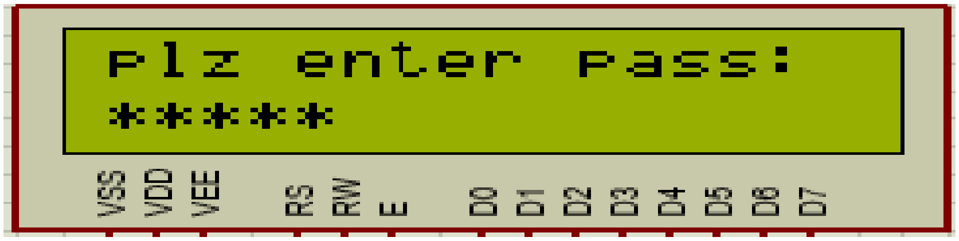
- The LCD will always display the main system option:



```
+ : Open Door
- : Change Pass
```

**Step3 - Open Door +**

- The LCD should display “Please Enter Password” like that:

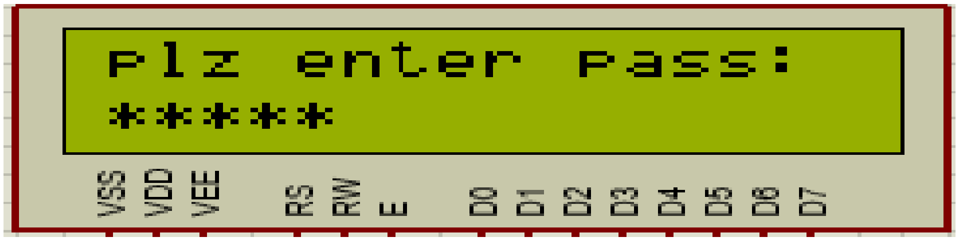


```
plz enter pass:
*****
```

- Enter the password then press **enter** button (choose any button in the keypad as enter button).
- **HMI_ECU** should send the Password to the **Control_ECU** and it should compare it with the one saved in the **EEPROM**.
- if two passwords are **matched**:
  - **rotates** motor for 15-seconds **CW** and display a message on the screen “Door is Unlocking”
  - hold the motor for 3-seconds.
  - **rotates** motor for 15-seconds **A-CW** and display a message on the screen “Door is Locking”

## Step 4 - Change Password -

- The LCD should display “Please Enter Password” like that:

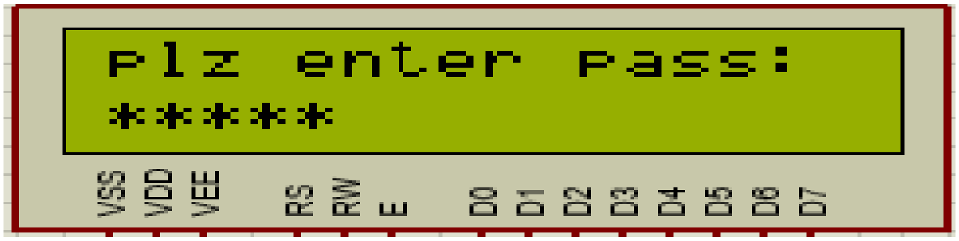


plz enter pass:
*****

- Enter the password then press **enter** button (choose any button in the keypad as enter button).
- **HMI_ECU** should send the Password to the **Control_ECU** and it should compare it with the one saved in the **EEPROM**.
- if two passwords are matched:
  - Repeat Step 1.

## Step 5

- if the two passwords are **unmatched** at step 3 (+ : Open Door) or step 4 (- : Change Password)
- Ask the user one more time for the password.
- The LCD should display “Please Enter Password” like that:



plz enter pass:
*****

- Enter the password then press **enter** button (choose any button in the keypad as enter button).
- **HMI_ECU** should send the password to the **Control_ECU** and it should compare it with the one saved in the **EEPROM**.
- if two passwords are matched then open the door or change the password in steps 3 and 4.
- If the two passwords are **not matched** again then ask the user **<u>one last time</u>** for the password.
- if two passwords are matched then open the door or change the password in steps 3 and 4.

- If the two passwords are not matched for the **third consecutive** time, then:
  - Activate Buzzer for 1-minute.
  - Display error message on LCD for 1 minute.
  - System should be locked no inputs from Keypad will be accepted during this time period.
  - Go to Step 2 the main options again.

## GPIO Driver Requirements

1. Use the Same GPIO driver implemented in the course.
2. Same driver should be used in the two ECUs.

## LCD Driver Requirements

1. Use a 2x16 LCD.
2. Use the Same LCD driver implemented in the course with 8-bits or 4-bits data mode.
3. Connect the LCD control and data bus pins to any pins of your choice in the MCU.
4. LCD should be connected to the **HMI_ECU**.

## Keypad Driver Requirements

1. Use a **4x4 Keypad**.
2. Connect the Keypad pins to any pins of your choice in the MCU.
3. Keypad should be connected to the **HMI_ECU**.

## DC_Motor Driver Requirements

1. Use the Same **DC_Motor** driver implemented in the fan controller project.
2. Motor should always run with the maximum speed using **Timer0 PWM**.
3. Motor should be connected to the **CONTROL_ECU**.
4. Connect the Motor pins to any pins of your choice in the MCU.

## EEPROM Driver Requirements

1. Use the Same **external EEPROM** driver controller by the I2C.
2. EEPROM should be connected to the **CONTROL _ECU**.

## I2C Driver Requirements

1. Use the Same I2C driver implemented in the course.
2. I2C driver will be used in the **CONTROL_ECU** to communicate with the external EEPROM.
3. You need to modify the **TWI_init** function implemented in the I2C session to take a pointer to the configuration structure with type **TWI_ConfigType**.
4. The function declaration should be:

   **void TWI_init(const TWI_ConfigType * Config_Ptr)**

**5.** The TWI_ConfigType structure should be declared like that:
**typedef struct{**

**TWI_Address address;**

**TWI_BaudRate bit_rate;**

**}TWI_ConfigType;**

The **TWI_Address** and **TWI_BaudRate** are types defined as uint8/uint16/uint32 or enum.

6. The **CONTROL_ECU** Microcontroller should be act as Master with **device address 10** and the used **baud rate** should be **400K Bits/Sec**.

## UART Driver Requirements

1. Use the Same UART driver implemented in the course.
2. Same driver should be used in the two ECUs.
3. You need to modify the **UART_init** function implemented in the UART session to take a pointer to the configuration structure with type **UART_ConfigType**.
4. The function declaration should be:

**void UART_init(const UART_ConfigType * Config_Ptr)**

**5.** The **UART_ConfigType** structure should be declared like that:
**typedef struct{**

**UART_BitData bit_data;**

**UART_Parity parity;**

**UART_StopBit stop-bit;**

**UART_BaudRate baud-rate;**

**}UART_ConfigType;**

The **UART_BitData**, **UART_Parity**, **UART_StopBit**, and **UART_BaudRate** are types defined as **uint8/uint16/uint32** or **enum**.

6. The UART Frame should be in the below format:
   - **Date Length:** 8-Bits Data
   - **Parity Type:** Even Parity
   - **Stop Bits:** 1-Stop Bit

## Timer Driver Requirements

1. Same driver should be used in the two ECUs.
2. In the **HMI_ECU** to count the displaying messages time on the LCD while opening/closing the door. In the **CONTROL_ECU** to count the time for controlling the motor.
3. Implement a full Timer driver for **TIMER1** with the configuration technique.
4. The Timer1 Driver should be designed using the Interrupts with the callback's technique.
5. The Timer1 Driver should support both **normal** and **compare** modes and it should be configured through the configuration structure passed to the init function.
6. The Timer Driver has 3 functions and two ISR's for Normal and Compare interrupts:

   a. **void Timer1_init(const Timer1_ConfigType * Config_Ptr)**

   - **Description**
     - ➢ Function to initialize the Timer driver
   - **Inputs: pointer to the configuration structure with type Timer1_ConfigType.**
   - **Return: None**

   **b. void Timer1_deInit(void)**

   - **Description**
     - ➢ Function to disable the Timer1.
   - **Inputs: None**
   - **Return: None**

   **c. void Timer1_setCallBack(void(*a_ptr)(void));**

   - **Description**
     - ➢ Function to set the Call Back function address.
   - **Inputs: pointer to Call Back function.**
   - **Return: None**

4. The **Timer1_ConfigType** structure should be declared like that:

```
typedef struct {
    uint16 initial_value;
    uint16 compare_value; // it will be used in compare mode only.
    Timer1_Prescaler prescaler;
```

**Timer1_Mode mode;**

**} Timer1_ConfigType;**

The **Timer1_Prescaler** and **Timer1_Mode** are types defined as **uint8** or **enum**.

## Buzzer Driver Requirements

1. Implement a full **Buzzer** driver.
2. Buzzer should be connected to the **CONTROL_ECU**.
3. Connect the **Buzzer** pin to any pins of your choice in the MCU.
4. The buzzer pin should be chosen by **static configurations**.
5. The Buzzer Driver has 3 functions:
   a. **void Buzzer_init()**
      - **Description**
        - ➢ Setup the direction for the buzzer pin as output pin through the GPIO driver.
        - ➢ Turn off the buzzer through the GPIO.
      - **Inputs: None**
      - **Return: None**

   b. **void Buzzer_on(void)**
      - **Description**
        - ➢ Function to enable the Buzzer through the GPIO.
      - **Inputs: None**
      - **Return: None**

   c. **void Buzzer_off(void)**
      - **Description**
        - ➢ Function to disable the Buzzer through the GPIO.
      - **Inputs: None**
      - **Return: No**

**Thank You & Good Luck**
**Eng/Mohamed Tarek**