Standard Embedded Diploma - Final Project

Project Title:

Dual Microcontroller-Based Door Locker Security System Using Password Authentication

Objective:

The objective of this project is to develop a secure and efficient smart door control system. It uses two microcontrollers, HMI_ECU and Control_ECU, which communicate via UART. The system implements password authentication, stores data in external EEPROM, and integrates additional components such as a PIR sensor, H-bridge motor control, and a buzzer for enhanced functionality.

Project Overview:

This smart door control system is designed with two microcontrollers, one acting as the Human-Machine Interface (HMI_ECU) and the other as the Control Unit (Control_ECU). Users interact with the system using an LCD and keypad to enter passwords, which are verified and stored using external EEPROM. The door is controlled by an H-bridge circuit connected to a motor, and the system includes a PIR sensor for motion detection, a buzzer for alarms, and password-protected access to system options like door unlocking and password changing.

Features:

1. **Password Protection**: Users can set and verify a password stored in external EEPROM.
2. **LCD and Keypad Interface**: Allows easy interaction for entering and managing passwords.
3. **UART Communication**: HMI_ECU sends and receives data to and from Control_ECU via UART.
4. **EEPROM Storage**: Passwords and system data are stored securely in an external EEPROM.
5. **Motorized Door Control**: The door is unlocked/locked using a motor driven by an H-bridge.
6. **Buzzer Alert**: The buzzer is activated for failed password attempts and system alerts.
7. **PIR Motion Sensor**: Detects motion to trigger door operations.
8. **Password Change Option**: Users can change the password after verification.
9. **Security Lock**: System locks for one minute if the password is entered incorrectly three times consecutively.

Hardware Components:

- **HMI_ECU Connections:**
    1. **LCD (8-bit mode)**
        - **RS** pin connected to **PC0**
        - **E (Enable)** pin connected to **PC1**
        - **Data Pins (D0-D7)** connected to **Port A** (PA0 to PA7)
    2. **Keypad (4x4)**
        - **Rows** connected to: **PB0, PB1, PB2, PB3**
        - **Columns** connected to: **PB4**, **PB5** ,**PB6** ,**PB7**
    3. **UART Communication**
        - **TXD (Transmit Data)** pin connected to **RXD** of Control_ECU.
        - **RXD (Receive Data)** pin connected to **TXD** of Control_ECU.
- **Control_ECU Connections:**
    1. **External EEPROM (I2C Communication)**
        - **SCL** (Serial Clock Line) connected to **PC0**
        - **SDA** (Serial Data Line) connected to **PC1**
    2. **Buzzer**
        - Connected to **PC7**
    3. **H-bridge Motor Driver**
        - **Input 1** connected to **PD6**
        - **Input 2** connected to **PD7**
        - **Enable1:** PB3/OC0
    4. **Motor (for Door Control)**
        - Connected to the H-bridge motor driver.
    5. **PIR Motion Sensor**
        - Connected to **PC2**

Operation Steps

Step1 – Create a System Password

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

- 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).
- Ask the user to re-enter the same password for confirmation by display this message “Please 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).

- **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:

Step3 - Open Door +

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

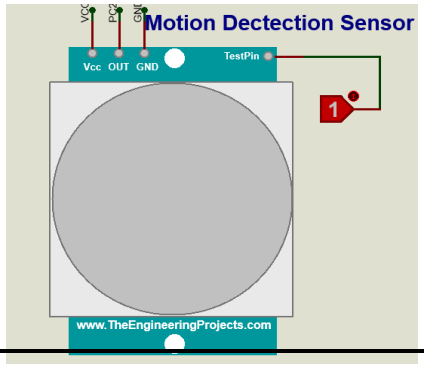
- Enter the password then press **enter** button (choose any button in the keypad as enter button).

- **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 as the PIR sensor Detected Their Motion and Display “Wait for people to Enter”.



-
- **rotates** motor for 15-seconds **A-CW** and display a message on the screen “Door is Locking” when the PIR is No longer Detect Motion

Step 4 - Change Password -

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

-

- 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 repeat Step 1.

Step 5

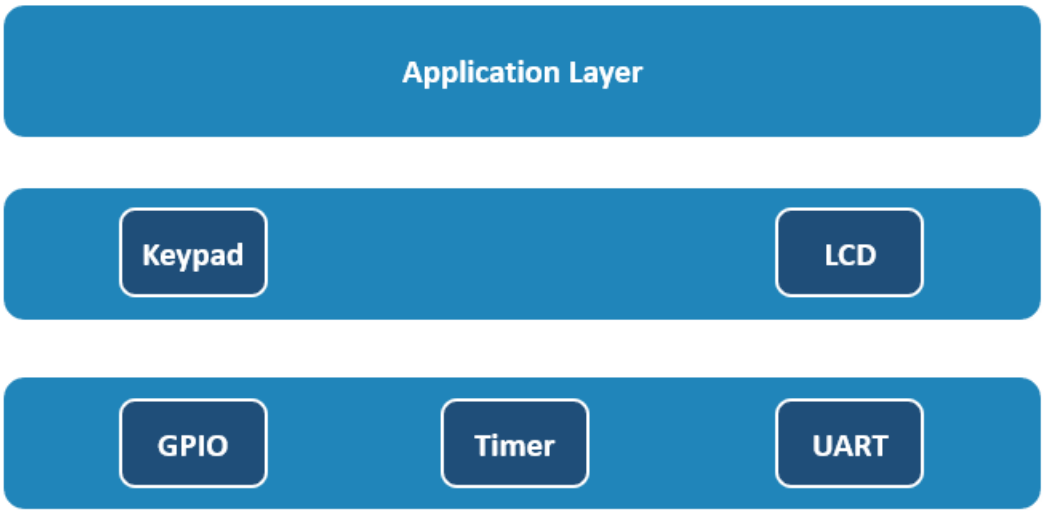
- if the two passwords are **unmatched** at step 3 (+ : Open Door) or step 4 (- : Change Password)
- Ask the user one more time for the password.
- The LCD should display “Please Enter Password” like that:

- 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.

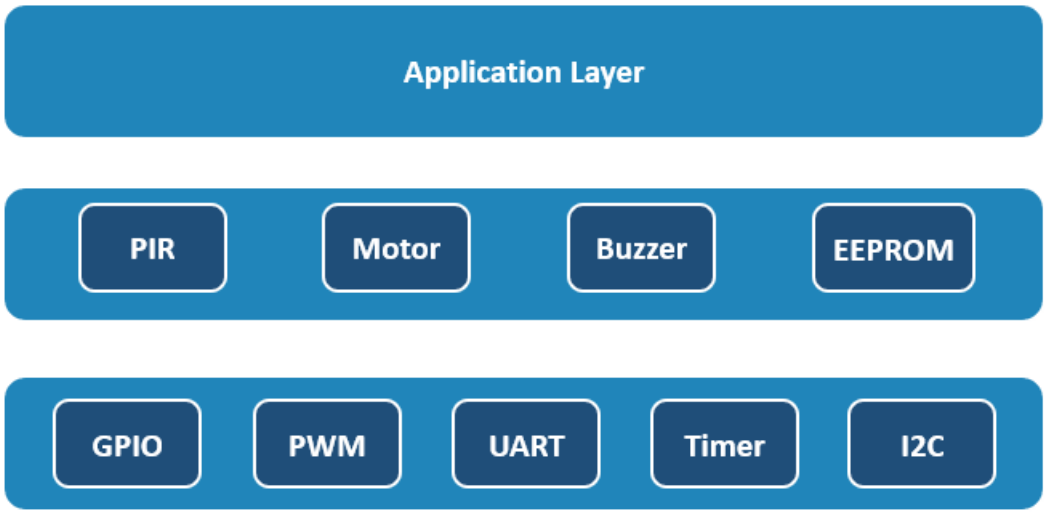
- if two passwords are matched then open the door or change the password in steps 3 and 4.
    - If the two passwords are not matched for the third consecutive time, then:
        - Activate Buzzer for 1-minute.
        - Display error message on LCD for 1 minute.
        - System should be locked no inputs from Keypad will be accepted during this time period.
        - Go to Step 2 the main options again

System Requirements:

1. **System Frequency:** 8 MHz
2. **Microcontroller:** ATmega32.
3. **The Project should be implemented using the below layered model architecture:**
    a. **The HMI_ECU**



    b. **The Control_ECU**



Drivers Requirements:

GPIO Driver Requirements

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

UART Driver Requirements

- Use the Same UART driver implemented in the course.
- Same driver should be used in the two ECUs.
- 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.**
- The function declaration should be:

  void UART_init(const UART_ConfigType * Config_Ptr)

- The **UART_ConfigType** structure should be declared like that:

  typedef struct {

  UART_BitDataType bit_data;

  UART_ParityType parity;

  UART_StopBitType stop-bit;

  UART_BaudRateType baud-rate;

  }UART_ConfigType;

  Note: The **UART_BitDateType** ,**UART_ParityType** , **UART_StopBitType** , and **UART_BaudRateType** are types defined as uint8/uint16/uint32 or enum

LCD Driver Requirements

- Use a 2x16 LCD.
- Use the Same LCD driver implemented in the course with 8-bits data mode.
- Connect the LCD control and data bus pins as mentioned in Hardware Component
- LCD should be connected to the HMI_ECU.

Keypad Driver Requirements

- Use a 4x4 Keypad.
- Connect the Keypad as mentioned in Hardware Component
- Keypad should be connected to the HMI_ECU.

I2C Driver Requirements

- Use the Same I2C driver implemented in the course.
- I2C driver will be used in the **CONTROL_ECU** to communicate with the external EEPROM
- 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.**
- The function declaration should be:

  void TWI_init(const TWI_ConfigType * Config_Ptr)

- The **TWI_ConfigType** structure should be declared like that:

  typedef struct {

  TWI_AddressType address;

  TWI_BaudRateType bit_rate;

  }TWI_ConfigType;

  Note: The **TWI_AddressType and TWI_BaudRateType** are types defined as uint8/uint16/uint32 or enum.

PWM Driver Requirements

The driver uses **Timer0** in **PWM mode** and operates as follows:

PWM Function:

- **void PWM_Timer0_Start(uint8 duty_cycle)**
  - Initializes Timer0 in PWM mode and sets the required duty cycle.
  - Prescaler: F_CPU/64
  - Non-inverting mode
  - The function configures OC0 as the output pin.
  - **Parameters**:
    - **duty_cycle:** Percentage (0 to 100%) representing the PWM duty cycle

Timer Driver Requirements

- Same driver should be used in the two ECUs.
- 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.
- Implement a full Timer driver for TIMER0, TIMER1, TIMER2 with the dynamic configuration technique.
- The Timer Driver should be designed using the Interrupts with the callback's technique.

- The Timer Driver should support both normal and compare modes and it should be configured through the configuration structure passed to its initialization function.
- The Timer Driver has **3 functions and Six ISR's** for Normal and Compare interrupts:
    - **void Timer_init(const Timer_ConfigType * Config_Ptr)**
        - Description: Function to initialize the Timer driver
        - Inputs: pointer to the configuration structure with type Timer_ConfigType.
        - Return: None
    - **void Timer_deInit(Timer_ID_Type timer_type);**
        - Description**:** Function to disable the Timer via Timer_ID.
        - Inputs: Timer_ID
        - Return: None
    - **void Timer_setCallBack(void(*a_ptr)(void), Timer_ID_Type a_timer_ID);**
        - Description: Function to set the Call Back function address to the required Timer.
        - Inputs: pointer to Call Back function and Timer Id you want to set The Callback to it.
        - Return: None
- The **Timer_ConfigType** structure should be declared like that:

```
typedef struct
{
        uint16 timer_InitialValue;
        uint16 timer_compare_MatchValue;     /*it will be used in compare mode only*/
        Timer_ID_Type  timer_ID;
        Timer_ClockType timer_clock;
        Timer_ModeType  timer_mode;
}Timer_ConfigType;
```

Note: The **Timer_ID_Type, Timer_ClockType and Timer_ModeType** are types defined as uint8/uint16/uint32 or enum.

Buzzer Driver Requirements

- Use the Same Buzzer driver implemented in the previous projects.
- The Buzzer should be Connected to the **Control_ECU**.

PIR Driver Requirements

- Use a PIR sensor to detect motion near the door. When the sensor detects movement, it will keep the door held open.
- PIR is connected to **CONTROL_ECU**
- The PIR Driver contain Two functions as following
    - **void PIR_init(void)**
        - Description: Function to initialize the PIR driver
        - Inputs: None
        - Return: None
    - **uint8 PIR_getState(void);**
        - Description: Function to return PIR State
        - Inputs: None
        - Return: uint8

DC_Motor Driver Requirements

- Use the Same **DC_Motor** driver implemented in the Smart Home Project.
- Motor should always run with the maximum speed using **Timer0 PWM.**
- Motor should be connected to the **CONTROL_ECU.**
- Connect the Motor pins mentioned in Hardware Component

EEPROM Driver Requirements

- Use the Same **external EEPROM** driver controller by the I2C.
- EEPROM should be connected to the **CONTROL_ECU**

Video Reference:

https://youtu.be/X5EwIRfGAAY

How to add PIR sensor library to proteus:

https://youtu.be/bhT90bM-Vdw

PIR sensor library download link:

https://www.mediafire.com/file/vjffcf868ea2fbi/PIR_Sensor_lib.zip/file

Thank You
Edges For Training Team