

Imt School

Project Documentation

Group Members: Mahmoud bahaa eldeen Ahmed Abdelghafar Moller maged

**Agenda:**

1. Introduction
2. System requirement
3. Architecture and Design

• system architecture

• flow chart diagram

4- Implementation Details

1. Hardware Connections
2. User Guide

• Getting started

•Trouble shooting

1. Test cases and Test results

#### Introduction

#### Project overview

#### Objective:

The objective of this project is to design and implement a comprehensive mobile-based smart home system that enhances the convenience, security, and energy efficiency of residential living spaces. This system enables up to 10 different users to remotely control LED lights and manage the safety of the entrance door using a mobile application.

Additionally, it incorporates smart lighting and air conditioning features that automatically adjust based on environmental conditions, utilizing light-dependent resistor (LDR) sensors for lighting control and LM35 sensors for temperature-based air conditioning management.

#### Key Features:

* + **User Authentication**: Each user is provided with a unique username and password, ensuring secure access to the system. A robust authentication mechanism is in place to prevent unauthorized access, with a security feature that triggers an alarm if a wrong password is entered three times consecutively.
  + **LED Light Control**: Users can control LED lights remotely via the mobile application, allowing them to turn lights on or off and adjust brightness levels. The system uses LDR sensors to automatically adjust the lighting based on ambient light conditions, promoting energy savings and enhancing user comfort.
  + **Safety Door Management**: The system includes functionality for remotely managing the safety door, providing users with the ability to lock or unlock the door through the mobile application. This feature enhances home security and provides convenience for users.
  + **Smart Air Conditioning**: Utilizing LM35 temperature sensors, the system intelligently manages air conditioning to maintain optimal indoor temperature levels. Users can set temperature preferences, and the system automatically adjusts the air conditioning based on real-time temperature readings, ensuring a comfortable living environment while optimizing energy consumption.
  + **Multi-User Support**: Designed to accommodate up to 10 users, the system allows multiple residents or authorized individuals to control and monitor the smart home system, ensuring flexibility and convenience for families or shared living spaces.

**2.System Requirements**

#### System Hardware Requirements Core Microprocessor: ATmega32

* + **Microcontroller**: ATmega32
  + **Operating Voltage**: 4.5-5.5V
  + **Clock Speed**: Up to 16 MHz
  + **Flash Memory**: 32 KB (which is sufficient for storing the firmware)
  + **SRAM**: 2 KB
  + **EEPROM**: 1 KB
  + **I/O Pins**: 32 programmable I/O lines
  + **Communication Interfaces**: UART, SPI, and I2C interfaces for peripheral communication
  + **Timers**: Three flexible Timer/Counters with compare modes
  + **Analog to Digital Converter (ADC)**: 10-bit ADC with up to 8 channels, allowing for sensor data acquisition
  + **PWM Channels**: 4 PWM channels for motor control or LED dimming

#### Additional Hardware Components

* + **LED Lights**: For the control of lighting within the smart home system. The number and specifications will depend on the application's requirements.
  + **LDR Sensor (Light-Dependent Resistor)**: To automatically adjust the lighting based on ambient light conditions. The system should be capable of reading analog values through the ADC of the ATmega32.
  + **LM35 Temperature Sensor**: For monitoring and controlling the air conditioning system based on the current temperature. The LM35 provides analog output proportional to the temperature, which can be read by the ATmega32's ADC.
  + **Servo Motor** An electronic lock mechanism controllable through one of the ATmega32's digital I/O pins.
  + **Buzzer or Alarm System**: For security alerts, operable through a digital I/O pin.
  + **Dc Motor:** used with Lm35

**System Architecture:**

# 3.Architecture and Design

We will use the layered Architecture that consists of 4 layers

-HAL Layer : Dc motor, led, LM35 Sensor, Ldr Sensor

-MCAL Layer : DIO , GI, Timers, UART, TWI

-Library Layer : STD\_TYPES, BIT\_MATH

-source Layer : application

**Program Flow Chart:**

# 4.Implementation Details

## 1-HAL Layer:

**-Led driver Functions:**

#### void HLED\_voidTurnOn(DIO\_PORT\_e A\_DIOPort,DIO\_PIN\_e A\_PINID)

Description:

This function turns on an LED connected to a specified port and pin on the microcontroller.

Parameters:

* + **A\_DIOPort**: The port where the LED is connected. This parameter is of type **DIO\_PORT\_e**, which is an enumeration listing all the digital I/O ports available on the microcontroller.
  + **A\_PINID**: The pin number on the specified port to which the LED is connected. This parameter is of type **DIO\_PIN\_e**, an enumeration of the pin identifiers.

Operation:

* + Sets the pin direction to output by calling **MDIO\_voidSetPinDirection**.
  + Sets the pin value to high (**DIO\_SET**), turning the LED on.

#### void HLED\_voidTurnOff(DIO\_PORT\_e A\_DIOPort,DIO\_PIN\_e A\_PINID)

Description:

This function turns off an LED connected to a specified port and pin on the microcontroller.

Parameters:

* + **A\_DIOPort**: The port where the LED is connected.
  + **A\_PINID**: The pin number on the specified port to which the LED is connected. Operation:
  + Sets the pin value to low (**DIO\_RESET**), turning the LED off.

#### void HLED\_voidToggleLED(DIO\_PORT\_e A\_DIOPort,DIO\_PIN\_e A\_PINID)

Description:

This function toggles the state of an LED (from on to off or off to on) connected to a specified port and pin on the microcontroller.

Parameters:

* + **A\_DIOPort**: The port where the LED is connected.
  + **A\_PINID**: The pin number on the specified port to which the LED is connected.

Operation:

* + Toggles the pin value, effectively changing the LED's state from on to off or vice versa.

## 2-MCAL Layer:

### -DIO Driver Functions:

#### MDIO\_voidSetPinDirection

Configures the direction (input or output) of a specified pin within a given port.

#### MDIO\_voidSetPinValue

Sets the output value (high or low) of a specified pin within a given port.

#### MDIO\_GetPinValue

Retrieves the current input value (high or low) from a specified pin within a given port.

#### MDIO\_voidTogglePinValue

Toggles the output value (from high to low or vice versa) of a specified pin within a given port.

#### MDIO\_voidSetPortDirection:

Sets the direction (input or output) for all pins within a specified port using a byte where each bit represents a pin's direction.

#### MDIO\_voidSetPortValue

Assigns output values (high or low) to all pins within a specified port using a byte where each bit represents a pin's value.

#### MDIO\_voidInit

Initializes the direction and value of all pins across all ports according to predefined macros, setting up the initial state of the GPIO interface.

### -Timers Driver Functions

#### void MTIMER1\_voidInit (void)

**Description:** Initializes Timer 1 with a predefined mode and settings. Specifically, it sets Timer 1 to Fast PWM mode with the top value in ICR1, non-inverting mode for OC1A output, and a prescaler value to start the timer.

#### Operation:

* + Configures Timer 1 for Fast PWM mode (Mode 14) by setting the appropriate bits in the TCCR1A and TCCR1B registers.
  + Sets non-inverting mode for the OC1A pin, meaning OC1A will be cleared on compare match and set at BOTTOM.
  + Initializes ICR1 (Input Capture Register 1) to a value of 19999, which defines the top value for the PWM cycle.
  + Sets OCR1A (Output Compare Register A for Timer 1) to 0, effectively starting with a duty cycle of 0%.
  + Configures the clock source for Timer 1 with a specific prescaler to start the timer.

#### void MTIMER1\_voidSetOCR1AValue (u16 A\_u16Value)

**Description:** Sets the value of OCR1A, which is used to determine the compare match value for generating PWM signals on OC1A pin in Fast PWM mode.

#### Parameters:

* + **A\_u16Value**: The value to be set in the OCR1A register.

#### Operation:

* + Directly assigns the provided value to OCR1A, affecting the PWM duty cycle for OC1A.

#### void MTIMER1\_voidInitSWICU()

**Description:** Initializes the Timer 1 for use with a Software Input Capture Unit (SWICU) by setting its clock source and prescaler.

#### Operation:

* + Similar to **MTIMER1\_voidInit**, this function sets up the clock source for Timer 1, essentially starting the timer with a specific prescaler. It's tailored for use cases involving software-based input capture.



1. **void MTIMER1\_voidSetTimerValue(u16 A\_u16Value) Description:** Sets the Timer 1's current count value to a specified value.

#### Parameters:

* + **A\_u16Value**: The value to be loaded into the TCNT1 register.

#### Operation:

* + Directly writes the provided value to the TCNT1 register, setting the current count value of Timer 1.

#### u16 MTIMER1\_u16ReadTimerValue(void)

**Description:** Reads and returns the current count value of Timer 1.

**Returns:** The current value of the TCNT1 register.

#### Operation:

* + Returns the value stored in TCNT1, which represents the current count value of Timer 1.

### -UART Driver Functions

#### void MUART\_voidInit(void)

**Description:** Initializes the UART module with predefined settings, including baud rate, frame format, and enabling the transmitter and receiver.

#### Operation:

* + Calculates the baud rate using a predefined equation and sets the UBRR (USART Baud Rate Register) low and high bytes to configure the baud rate.
  + Configures frame format (data bits, parity, stop bits) and other settings by manipulating the UCSRC (USART Control and Status Register C).
  + Enables the UART transmitter and receiver by setting bits in the UCSRB (USART Control and Status Register B).

#### void MUART\_voidSendByteSyncBlocking(u8 A\_u8DataByte)

**Description:** Sends a single byte over UART in a synchronous, blocking manner, waiting until the UART data register is empty before and after sending the byte.

#### Parameters:

* + **A\_u8DataByte**: The byte to send over UART.

#### Operation:

* + Waits until the transmit buffer (UDRE bit in UCSRA) is empty.
  + Sends the byte by writing it to UDR (USART Data Register).
  + Waits again for the transmit buffer to be empty to ensure byte is completely sent.

#### void MUART\_voidSendByteSyncNonBlocking(u8 A\_u8DataByte)

**Description:** Attempts to send a single byte over UART in a non-blocking manner, with a timeout mechanism to prevent indefinite waiting.

#### Parameters:

* + **A\_u8DataByte**: The byte to send over UART.

#### Operation:

* + Tries to send the byte if the transmit buffer is empty, with a timeout to exit if the condition is not met within a specified period.

1. **MUART\_voidSendStringSyncNonBlocking and MUART\_voidSendStringSyncBlocking Description:** Sends a null-terminated string over UART, character by character, using either the blocking or non-blocking byte transmission function.

#### Parameters:

* + **A\_pu8String**: Pointer to the null-terminated string to send.

#### Operation:

* + Iterates over each character in the string and sends it using the respective byte sending function until the null terminator is reached.

#### u8 MUART\_u8ReadByteSyncBlocking(void)

**Description:** Receives a single byte over UART in a synchronous, blocking manner, waiting until there is data to read.

**Returns:** The byte received from UART.

#### Operation:

* + Waits until the RXC (Receive Complete) bit in UCSRA is set, indicating data is available.
  + Returns the byte read from UDR.

#### MUART\_voidSendByteAsync and MUART\_u8ReadByteAsync

**Description:** Provides a basic framework for asynchronous byte transmission and reception, primarily intended for use with interrupt-based handling.

#### Operation:

* + **MUART\_voidSendByteAsync**: Sends a byte if the transmit buffer is empty.
  + **MUART\_u8ReadByteAsync**: Returns the byte in UDR, typically used within an RX complete interrupt service routine.

#### u8 MUART\_u8ReadByteSyncNonBlocking(void)

**Description:** Attempts to receive a single byte over UART in a non-blocking manner, with a timeout mechanism.

**Returns:** The byte received from UART or a predefined value if no data is received within the timeout period.

#### Operation:

* + Tries to read a byte from UDR if the RXC bit is set, with a timeout to exit if no data is received within a specified period.

1. **void MGI\_voidEnable(void)**

### -GI Driver Functions

**Description:** Enables global interrupts for the microcontroller by setting the I-bit (Interrupt Enable Bit) in the **SREG**.

#### Operation:

* + Sets the I-bit (usually bit 7) in the **SREG** register to enable global interrupts. This allows the microcontroller to respond to interrupt requests from various peripherals and external sources.

#### void MGI\_voidDisable(void)

**Description:** Disables global interrupts for the microcontroller by clearing the I-bit in the

#### SREG.

**Operation:**

* + Clears the I-bit in the **SREG** register to disable global interrupts. This prevents the microcontroller from responding to any interrupt requests, effectively making the system non-interruptible until global interrupts are re-enabled.

**-ADC Driver Functions**

**1-MADC\_voidInit:**

This function configures the ADC settings:

Sets the ADC reference voltage to AVCC with external capacitor at AREF pin by configuring ADMUX register bits.

Enables the ADC, its interrupt, and auto-trigger feature through the ADCSRA register.

Sets the ADC clock prescaler to a specified value for optimal ADC operation frequency.

Starts the ADC conversion by setting the ADSC bit in ADCSRA.

**2-MADC\_u16GetDigitalValue**

Retrieves the digital value from the ADC conversion process:

•Accepts an ADC\_Channels type, indicating the ADC channel to read from. The function checks if the channel is within a valid range.

•Masks the ADMUX register to clear previous channel selections and sets the new channel for the next conversion.

•Initiates a new ADC conversion, waits for the conversion to complete (this part is commented out, suggesting the function expects the conversion to be already initiated or handled externally), and then reads the result from the ADCLH register (note: ADCLH seems to be a placeholder for reading from both the ADCL and ADCH registers correctly).

•Returns the 10-bit ADC conversion result as a 16-bit unsigned integer.

### -Application layer

**User Database:** A predefined 2D array **Pre\_Defined\_Users** contains pairs of usernames and passwords for up to 10 users.

**Initialization:** Initializes the microcontroller's Digital I/O (MDIO), UART (MUART), and Timer 1 (MTIMER1) modules for further operations.

#### User Authentication Loop:

* **Username Input:** Prompts the user to enter a username via UART. The entered username is compared against the predefined list. If a match is

found, the system proceeds to request a password; otherwise, it notifies the user of the invalid username.

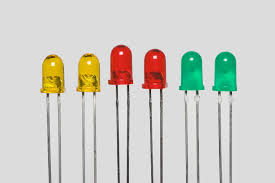
* **Password Verification:** After successfully entering a username, the user is prompted to enter the corresponding password. If the password matches the one associated with the username, the system grants access; otherwise, the user is given a limited number of attempts before triggering a security response (e.g., activating a siren).

**Control Commands:** Once authenticated, the user can control an LED and a door. The user sends '1' to toggle the LED and '0' to open/close the door. The door's opening and closing are simulated by adjusting the Output Compare Register (OCR1A) for Timer 1, which changes the PWM signal controlling a servo motor.

**Security Response:** If the user fails to authenticate after a certain number of attempts, a siren (an LED or buzzer connected to a digital pin) is activated for a short period as a security measure.

### Technical Details:

* + **UART Communication:** Uses synchronous blocking and non-blocking UART read/write functions to interact with the user through a serial terminal.
  + **PWM Control for Door:** Utilizes Timer 1 in PWM mode to generate signals for controlling a servo motor representing the door mechanism. The **MTIMER1\_voidSetOCR1AValue** function adjusts the PWM duty cycle, simulating the door's opening and closing.
  + **LED Control:** Toggles an LED connected to a specific pin/port based on user input, demonstrating simple digital output control.
  + **Security Mechanism:** Implements a simple security mechanism that limits password attempt failures and activates a siren upon repeated failures.



**Led connection:**

## Hardware connections

Connect the led to PORTA ,PIN0

#### Servo motor connection:

RED >VCC

Brown-->GND

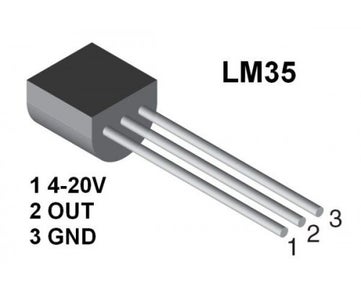
Orange--->PWM--->OCR1A >PD5

#### UART connection:

**Using USP to TTL**

TX--->PD0

RX--->PD1 GROUND--->GND

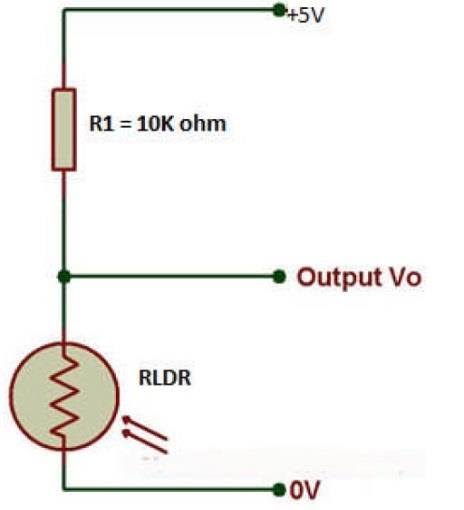


**LM35 Sensor:**

Left pin --🡪VCC

Center ---🡪Vo

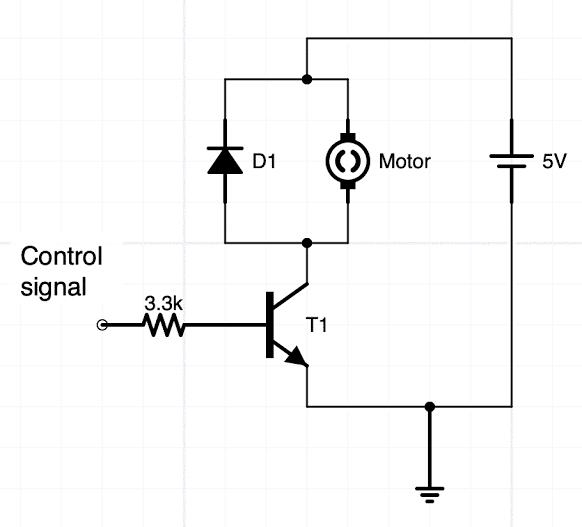
Right pin->GND

**LDR Sensor:**

Connect one pin to --🡪VCC

The other pin connected to -> resentence

Then ground

****Vo is the pin connected from resentence and LDRA

**DC motor:**

Connection with transistor

**System Requirements:**

1. **User Guide**

#### Getting Started

* + A device with a serial terminal application (e.g., PuTTY, Tera Term) installed.
  + An AVR microcontroller board with the application loaded.
  + The microcontroller board should be connected to a PC via a USB-to-serial adapter if necessary.

#### Connecting to the System:

* + Connect the microcontroller board to your PC.
  + Open your serial terminal application.
  + Configure the serial terminal with the correct COM port and set the baud rate as defined in the application (typically 9600, 19200, or 115200 bps).

#### Interacting with the System:

* + Once connected, reset the microcontroller board if necessary. You should see a prompt asking you to "Enter UserName: " in the serial terminal.
  + Type in your username and press Enter.
  + If prompted, enter the password associated with the username and press Enter.

#### Controlling the System:

* + Upon successful authentication, you'll receive instructions to press '1' to toggle the LED or '0' to open/close the door.
  + Send '1' or '0' as per the instructions to control the LED and the door mechanism, respectively.

#### Troubleshooting Issue 1: No response in the serial terminal

* **Check the physical connection** between the microcontroller board and your PC.
* **Ensure the serial terminal is configured** with the correct COM port and baud rate matching the application's settings.

#### Issue 2: Username or Password is not recognized

* **Double-check the username and password** you entered for typos. The credentials are case-sensitive.
* **Reset the microcontroller** if you exceed the password attempt limit and try again.

#### Issue 3: LED/Door does not respond to commands

* **Verify the hardware connections** of the LED and servo motor (for the door) to ensure they are properly connected to the specified pins on the microcontroller.
* **Check for any visible signs of damage** to the LED or servo motor.

#### Issue 4: System hangs or becomes unresponsive

* **Reset the microcontroller** to restart the application. This can resolve temporary glitches or issues.
* **Ensure there's no external interference** or issues with the power supply that might affect the microcontroller's operation.

#### Issue 5: Siren activates unexpectedly

* **This occurs after multiple failed login attempts.** Ensure you're entering the correct credentials.
* **If the issue persists without failed attempts**, inspect the logic handling the authentication and siren activation to ensure it's working as intended.

### 7.Test Cases and Test Results

Test Case 1: Successful User Login

* **Objective:** Verify the system allows access with correct username and password.

#### Procedure:

* + Enter a valid username from the predefined list.
  + Enter the corresponding valid password.
* **Expected Result:** The system displays a welcome message and prompts for control commands ('1' to toggle LED, '0' to control the door).

#### Actual Result:

Test Case 2: Invalid Username

* **Objective:** Confirm the system rejects access for an invalid username.

#### Procedure:

* + Enter an invalid username (not on the predefined list).
  + Observe the system's response.
* **Expected Result:** The system notifies the user that the username does not exist and prompts to try again.

#### Actual Result:

Test Case 3: Invalid Password

* **Objective:** Verify the system rejects access for a valid username with an incorrect password.

#### Procedure:

* + Enter a valid username.
  + Enter an incorrect password.
* **Expected Result:** The system notifies the user that the password is incorrect and allows for additional attempts.

#### Actual Result:

Test Case 4: Exceeded Password Attempts

* **Objective:** Ensure the system activates the security response after multiple failed password attempts.

#### Procedure:

* + Enter a valid username.
  + Enter an incorrect password three times.
* **Expected Result:** The system activates the siren (or security response) after the specified number of failed attempts.

#### Actual Result:

Test Case 5: Control LED

* **Objective:** Test the functionality to toggle the LED on and off.

#### Procedure:

* + Successfully log in with a valid username and password.
  + Send '1' to toggle the LED status.
* **Expected Result:** The LED toggles its state with each '1' received.

#### Actual Result:

Test Case 6: Control Door

* **Objective:** Verify the door mechanism (servo motor) opens/closes on command.

#### Procedure:

* + Successfully log in.
  + Send '0' to activate the door control mechanism.
* **Expected Result:** The door mechanism activates, simulating an open/close action with the servo motor.

#### Actual Result:

Test Case 7: Unrecognized Commands

* **Objective:** Ensure the system handles unrecognized commands gracefully.

#### Procedure:

* + Successfully log in.
  + Send a command other than '1' or '0'.
* **Expected Result:** The system ignores or notifies about the unrecognized command without crashing or hanging.

#### Actual Result:

Test Case 8: Logout/Session Termination

* **Objective:** Assess the system's ability to handle session termination correctly.

#### Procedure:

* + Successfully log in.
  + Perform an action to simulate session termination (if applicable).
* **Expected Result:** The system logs out the user, requiring re-authentication for further actions.