2022-2023 2nd Semester, EE304, Embedded Systems Final Project

Instructor

Abdulkadir KÖSE

Subject

Energy Consumption Detector

Submission Date

23.06.2023

Submitted by

Zehra Moğulkoç 110510223

Beyzanur Yüce 2011051071

Dilek Taylı 110510228

Group 16

I. Objective

The purpose of the Energy Consumption Detector is to prevent unnecessary waste and contribute to sustainability by keeping the amount of energy used at the optimum level in every environment where light and heat energy is used (home, workplace, institutions, etc.). Since it is thought that the system's self-management of energy may pose a danger, it is aimed to inform the user, offer suggestions, and make calculations about energy saving with a mobile application. In addition, when extraordinary values are reached in terms of heat and light, it is expected to detect an emergency and notify the user by giving a red alarm.

II. Features

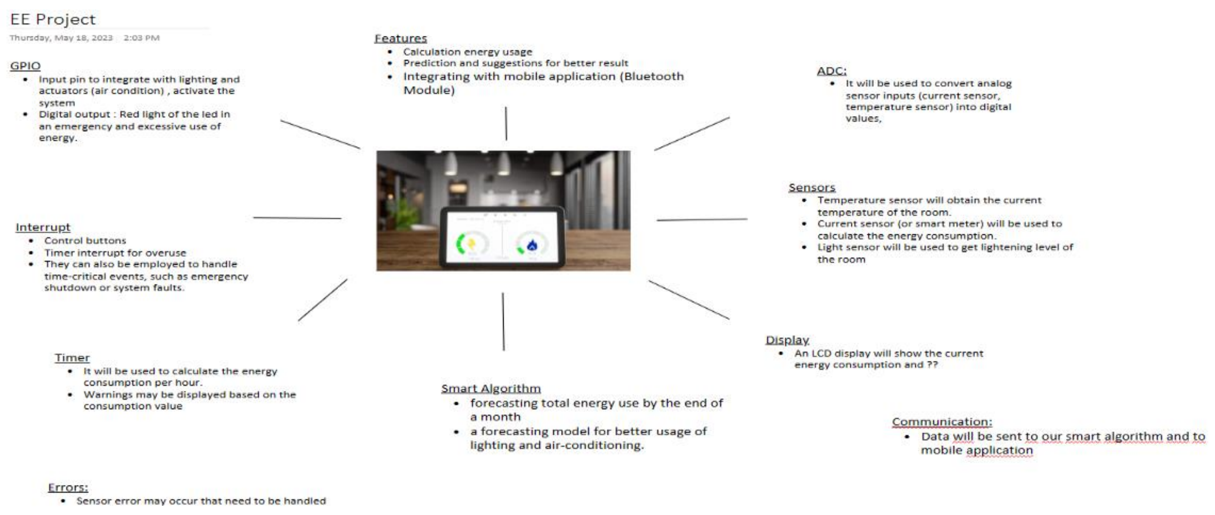
1) Real-time Energy Monitoring: The system has sensors that allow you to monitor energy consumption and energy usage data in various areas.

2) Calculation of Energy Usage: The system provides algorithms to calculate energy usage accurately and the data is provided by the sensors. Total energy consumption will be calculated by using a current sensor and warnings will be displayed according to the values coming from the temperature sensor and light sensor.

   - With a smart algorithm, the system will be able to predict and suggest better results.

3) It makes some predictions on the energy consumption for different time intervals based on given data and makes suggestions accordingly.

4) With a Bluetooth module, the system is integrated with mobile applications. So, the users will be able to monitor/see the energy usage remotely. In the mobile application, additional features such as energy-saving tips can be added.



EE Project

Thursday, May 18, 2023 2:03 PM

GPIO
- Input pin to integrate with lighting and actuators (air condition) , activate the system
- Digital output : Red light of the led in an emergency and excessive use of energy.

Features
- Calculation energy usage
- Prediction and suggestions for better result
- Integrating with mobile application (Bluetooth Module)

ADC:
- It will be used to convert analog sensor inputs (current sensor, temperature sensor) into digital values,

Interrupt
- Control buttons
- Timer interrupt for overuse
- They can also be employed to handle time-critical events, such as emergency shutdown or system faults.

Sensors
- Temperature sensor will obtain the current temperature of the room.
- Current sensor (or smart meter) will be used to calculate the energy consumption.
- Light sensor will be used to get lightening level of the room

Timer
- It will be used to calculate the energy consumption per hour.
- Warnings may be displayed based on the consumption value

Smart Algorithm
- forecasting total energy use by the end of a month
- a forecasting model for better usage of lighting and air-conditioning.

Display
- An LCD display will show the current energy consumption and ??

Communication:
- Data will be sent to our smart algorithm and to mobile application

Errors:
- Sensor error may occur that need to be handled

Figure1: Design of the project

III. Components

- **Sensors with ADC**

  In order to get the current temperature, LM35 temperature sensor is being used with Analog to Digital Converter. The temperature value is read from the sensor with the aid of ADC pin and then the obtained value is converted to Celsius with the aid of following equations:

  $$Voltage = ((ADC\ value \times 3.3))/4095$$
  $$Temperature = Voltage \times 100$$

  In addition to LM35, a light sensor (LDR) is used to detect the light intensity in the environment. Similar to LM35, an analog value is read from the sensor and with the aid of an ADC, it is converted to digital.

- **Display**

  A 16*2 LCD display is used for monitoring the values read from the sensors after the ADC conversion.

- **GPIO**

  We used some GPIO output pins to connect leds to the circuit which will be turned on in case of emergencies or after some specific changes.

- **Motor Driver and Lamp**

  Light intensity of the environment is measured with a lamp connected to the system. When the lamp uses a volt above 4.5, it will be considered as an emergency and the red LED will warn. The ADC value of 4.5 volt is calculated by:

  $$\frac{4.5}{3.3} \times 2^{12} + \frac{1}{2} = 5585.5$$

- **Communication**

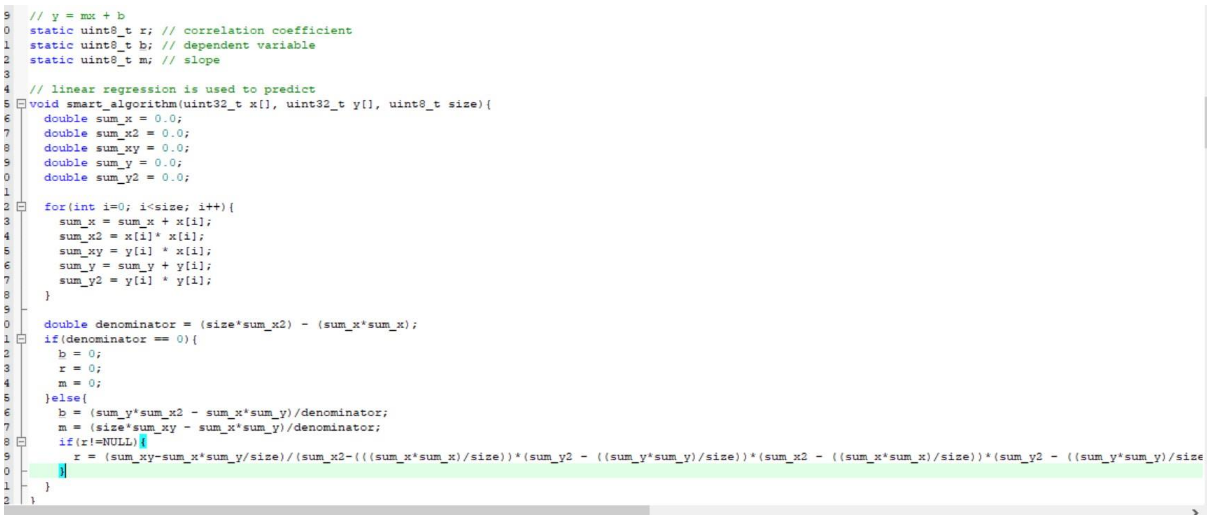
  The data obtained from the sensors are transferred to the virtual terminal via USART communication.

- **DMA**

  DMA is used in conjunction with SPI (Serial Peripheral Interface) to send temperature values from a peripheral device to memory.
- **Timer**

  We used TIM1 for doing the DMA operation on a regular basis.
- **Interrupt**
- We used interrupts while making predictions in order to do other jobs during the process.
- **Mobile Application**
- We developed a mobile application to control our system and enhance user experience. The user will be able to connect the system and on/off the system.

- **Bluetooth Module**
  To make the project more helpful, we created a mobile application and connected it to our project. We put in a Bluetooth module. We may connect to the system from our phone and turn it on and off with the help of this Bluetooth module. The Bluetooth module we used is Bluetooth HC-05. In order to use this module in proteus, we added its library in Keil library.
- **Smart Algorithm**
  To make the project more smart, we implemented a linear regression. With the help of the algorithm we will be able to make predictions about the future energy consumption. Additionally, this predicted value is sent to the mobile application.

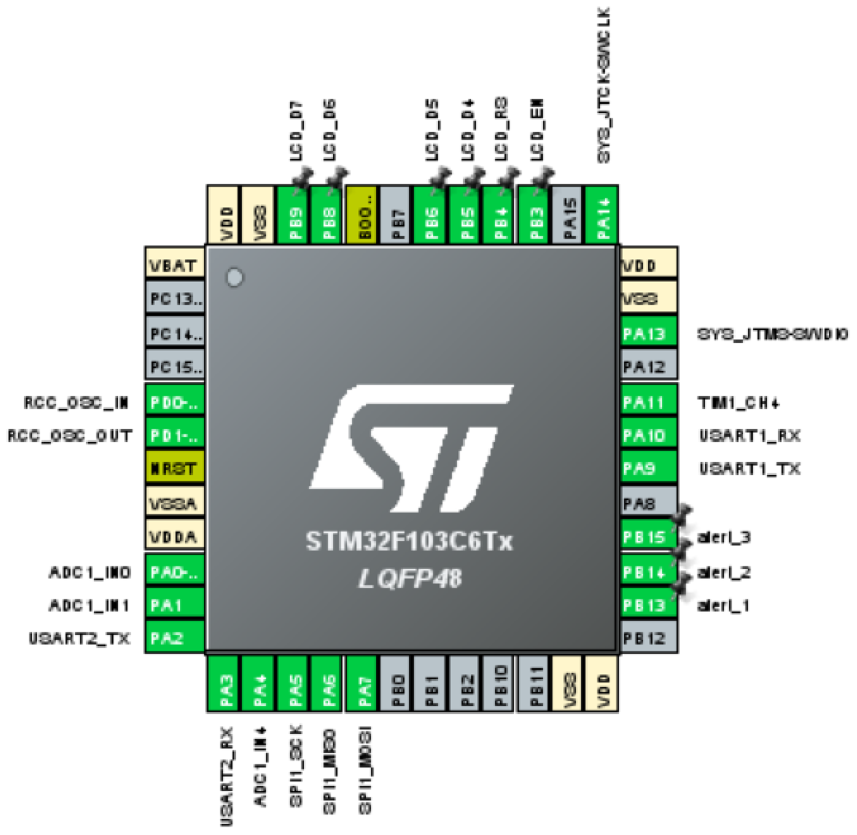


```
// y = mx + b
static uint8_t r; // correlation coefficient
static uint8_t b; // dependent variable
static uint8_t m; // slope

// linear regression is used to predict
void smart_algorithm(uint32_t x[], uint32_t y[], uint8_t size){
  double sum_x = 0.0;
  double sum_x2 = 0.0;
  double sum_xy = 0.0;
  double sum_y = 0.0;
  double sum_y2 = 0.0;

  for(int i=0; i<size; i++){
    sum_x = sum_x + x[i];
    sum_x2 = x[i]* x[i];
    sum_xy = y[i] * x[i];
    sum_y = sum_y + y[i];
    sum_y2 = y[i] * y[i];
  }

  double denominator = (size*sum_x2) - (sum_x*sum_x);
  if(denominator == 0){
    b = 0;
    r = 0;
    m = 0;
  }else{
    b = (sum_y*sum_x2 - sum_x*sum_y)/denominator;
    m = (size*sum_xy - sum_x*sum_y)/denominator;
    if(r!=NULL){
      r = (sum_xy-sum_x*sum_y/size)/(sum_x2-(((sum_x*sum_x)/size))*(sum_y2 - ((sum_y*sum_y)/size))*(sum_x2 - ((sum_x*sum_x)/size))*(sum_y2 - ((sum_y*sum_y)/size
    }
  }
}
```

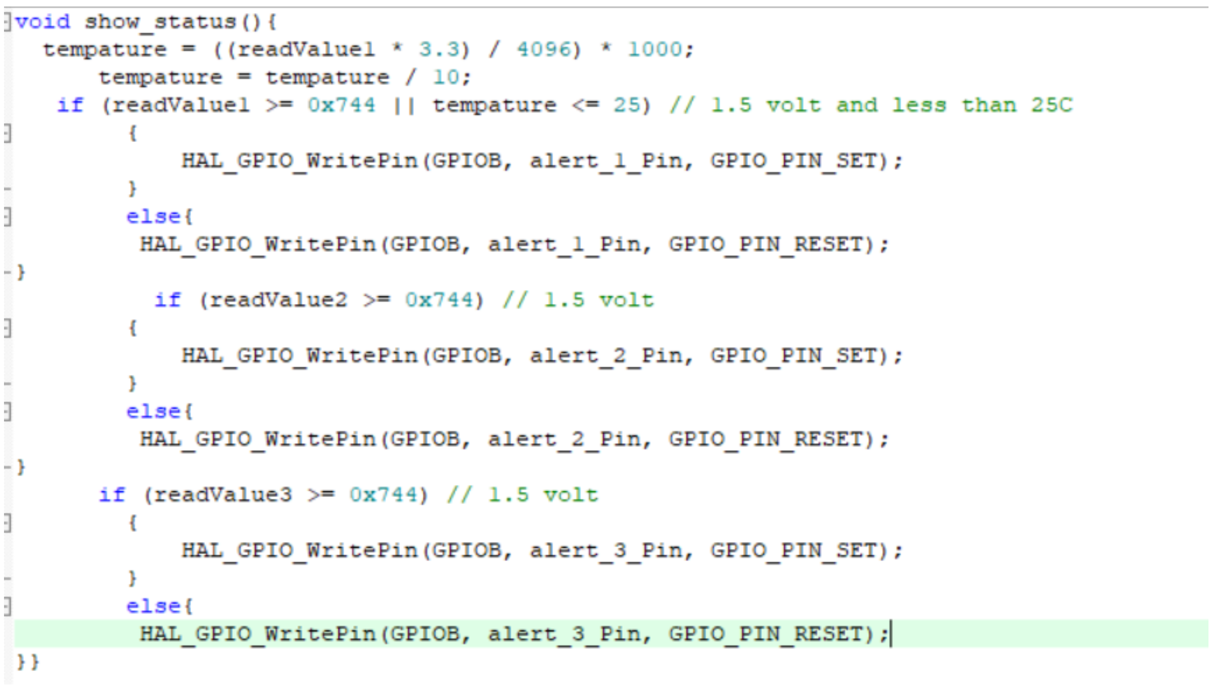
III. Implementation

STM32 microcontroller, including ADC reading, LCD display, smart algorithm for prediction, Bluetooth module for mobile application connection, and LED control. It combines multiple peripheral configurations and functions to achieve the desired functionality. Simulation is used on proteus.

CubeMX

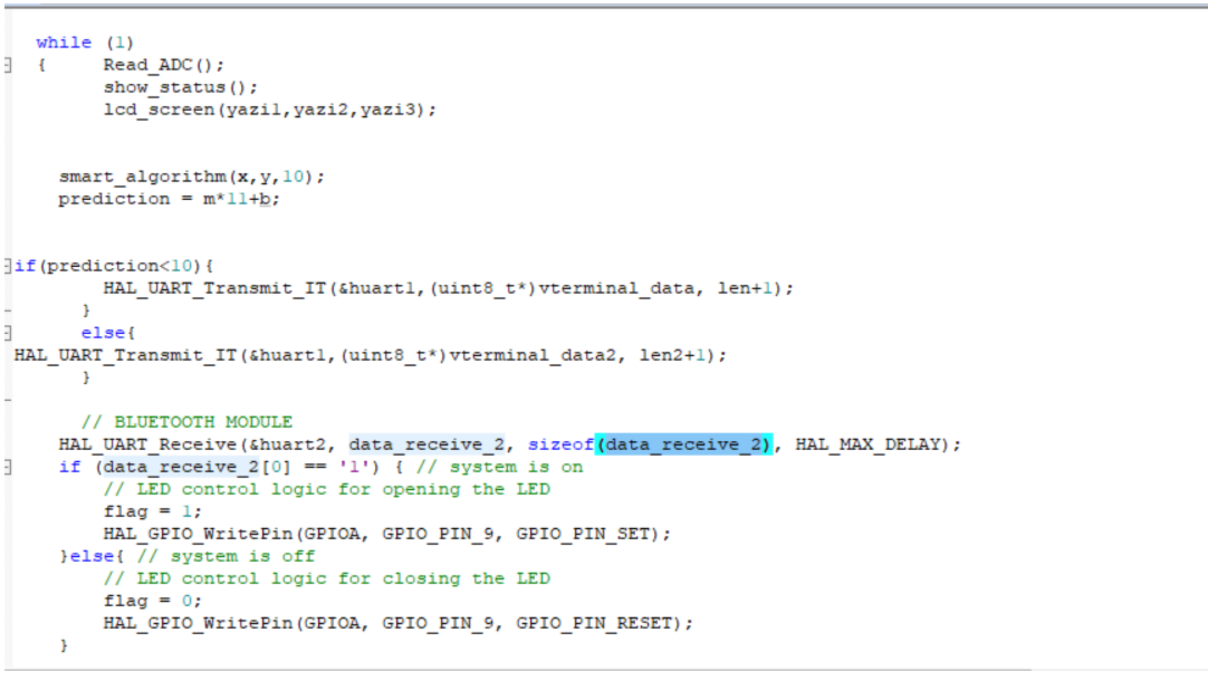


Leds: PB13, PB14 and PB15 pins are set as GPIO Output pins for led connections.



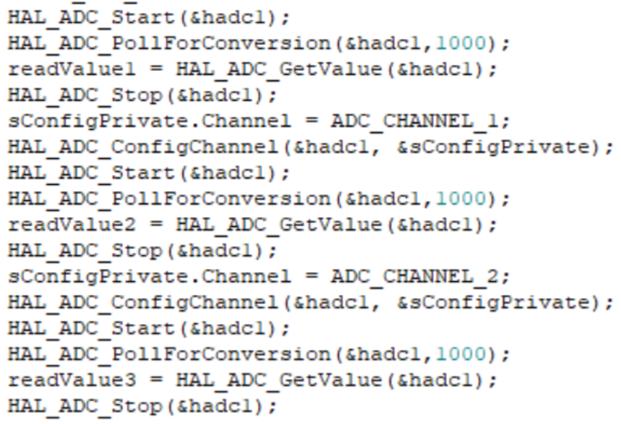
```
void show_status(){
  tempature = ((readValue1 * 3.3) / 4096) * 1000;
      tempature = tempature / 10;
   if (readValue1 >= 0x744 || tempature <= 25) // 1.5 volt and less than 25C
          {
              HAL_GPIO_WritePin(GPIOB, alert_1_Pin, GPIO_PIN_SET);
          }
          else{
           HAL_GPIO_WritePin(GPIOB, alert_1_Pin, GPIO_PIN_RESET);
}
            if (readValue2 >= 0x744) // 1.5 volt
          {
              HAL_GPIO_WritePin(GPIOB, alert_2_Pin, GPIO_PIN_SET);
          }
          else{
           HAL_GPIO_WritePin(GPIOB, alert_2_Pin, GPIO_PIN_RESET);
}
        if (readValue3 >= 0x744) // 1.5 volt
          {
              HAL_GPIO_WritePin(GPIOB, alert_3_Pin, GPIO_PIN_SET);
          }
          else{
           HAL_GPIO_WritePin(GPIOB, alert_3_Pin, GPIO_PIN_RESET);
}}
```

UART: The UART baud rate is set to 9600 and the word length is set to 8 bits. PA9 and PA10 pins are cross connected to UART1 because of the asynchronous mode and PA1 and PA2 pins are cross-connected to UART2. UART1 is used for sensor data while UART2 is used for the Bluetooth module.



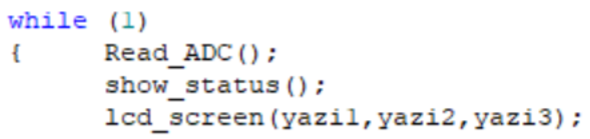
```
  while (1)
  {     Read_ADC();
        show_status();
        lcd_screen(yazi1,yazi2,yazi3);


    smart_algorithm(x,y,10);
    prediction = m*11+b;


if(prediction<10){
        HAL_UART_Transmit_IT(&huart1,(uint8_t*)vterminal_data, len+1);
      }
      else{
HAL_UART_Transmit_IT(&huart1,(uint8_t*)vterminal_data2, len2+1);
      }

      // BLUETOOTH MODULE
    HAL_UART_Receive(&huart2, data_receive_2, sizeof(data_receive_2), HAL_MAX_DELAY);
    if (data_receive_2[0] == '1') { // system is on
        // LED control logic for opening the LED
        flag = 1;
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
    }else{ // system is off
        // LED control logic for closing the LED
        flag = 0;
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
    }
```

ADC: All data from the sensors is gathered with the aid of ADC. We used pins PA0, PA1 and PA2 with ADC1.



```
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1,1000);
readValue1 = HAL_ADC_GetValue(&hadc1);
HAL_ADC_Stop(&hadc1);
sConfigPrivate.Channel = ADC_CHANNEL_1;
HAL_ADC_ConfigChannel(&hadc1, &sConfigPrivate);
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1,1000);
readValue2 = HAL_ADC_GetValue(&hadc1);
HAL_ADC_Stop(&hadc1);
sConfigPrivate.Channel = ADC_CHANNEL_2;
HAL_ADC_ConfigChannel(&hadc1, &sConfigPrivate);
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1,1000);
readValue3 = HAL_ADC_GetValue(&hadc1);
HAL_ADC_Stop(&hadc1);
```

LCD Display: ADC is activated in NVIC settings in CubeMX. It has a printing value between 1-100. We monitor the sensor data via a 7-segment LCD display.



```
while (1)
{       Read_ADC();
        show_status();
        lcd_screen(yazi1,yazi2,yazi3);
```

Keil MDK

The code includes several header files, such as "main.h," "stdio.h," "LCD.h," "stdlib.h," and "math.h," which provide necessary definitions, function prototypes, and standard library functions. We used an external library for defining LCD functions.We also used different types of private variables, such as flag, readValue1, readValue2, readValue3, sConfigPrivate, and temperature to store ADC readings, configuration settings, and temperature values. The code declares several function prototypes, including SystemClock_Config() for system clock configuration, and various functions for initializing peripherals such as GPIO, ADC, UART, SPI, and timers. Some of the functions are explained below:

Read_ADC(): This function reads the ADC values from three different channels (0, 1, and 2) using the HAL library. It configures the ADC channel, starts the ADC conversion, waits for the conversion to complete, and stores the ADC readings in the respective variables.

show_status(): This function calculates the temperature value based on the ADC readings and compares it with predefined thresholds. It sets or resets GPIO pins accordingly to indicate temperature conditions.

lcd_screen(): This function displays the ADC values on an LCD screen. It initializes the LCD, prepares the ADC readings as strings, and uses the lcd_print() function to display the values on specific lines of the LCD.

smart_algorithm(): This function implements a simple linear regression algorithm to predict a value based on given input arrays x and y. It calculates the slope (m) and intercept (b) of the regression line and stores them in corresponding variables.

The program enters an infinite loop where it repeatedly reads ADC values, displays them on the LCD, performs the smart algorithm for prediction, and controls LEDs based on certain conditions. It also includes UART communication for receiving commands from an external device.

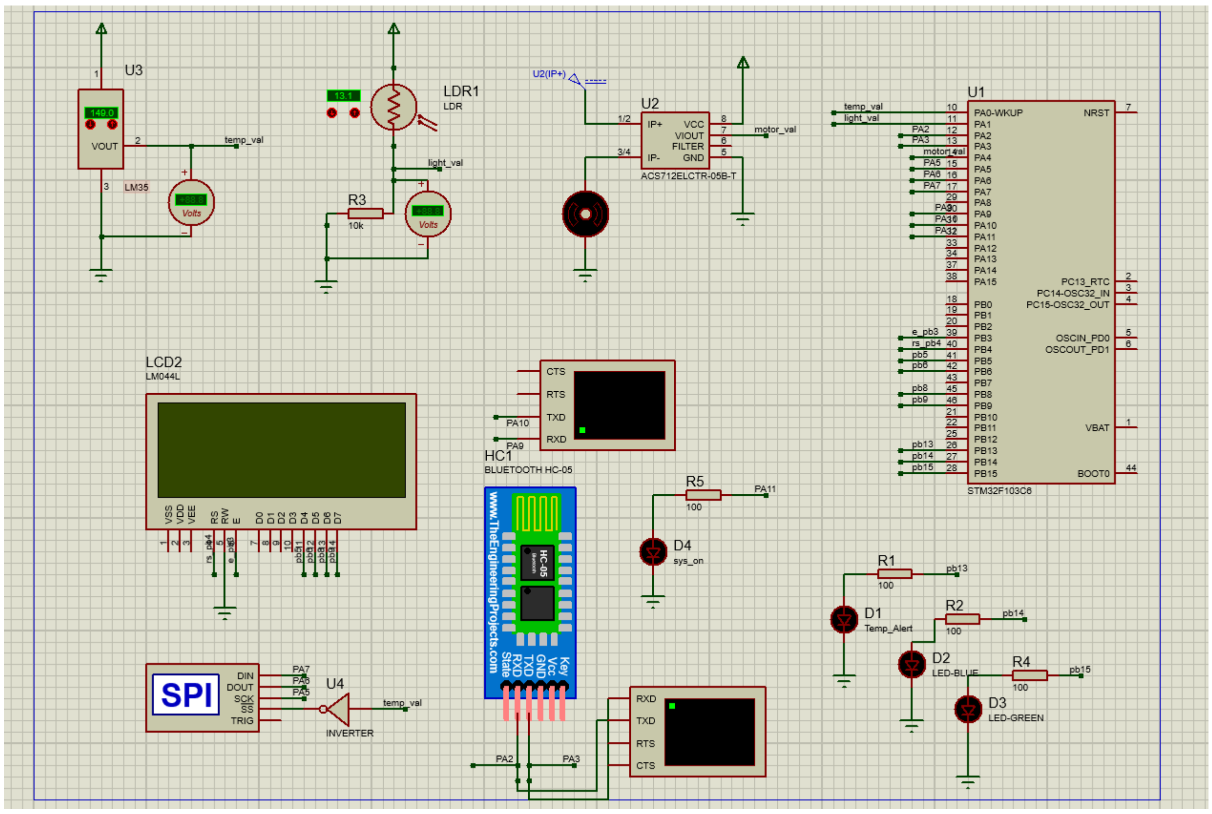
Proteus

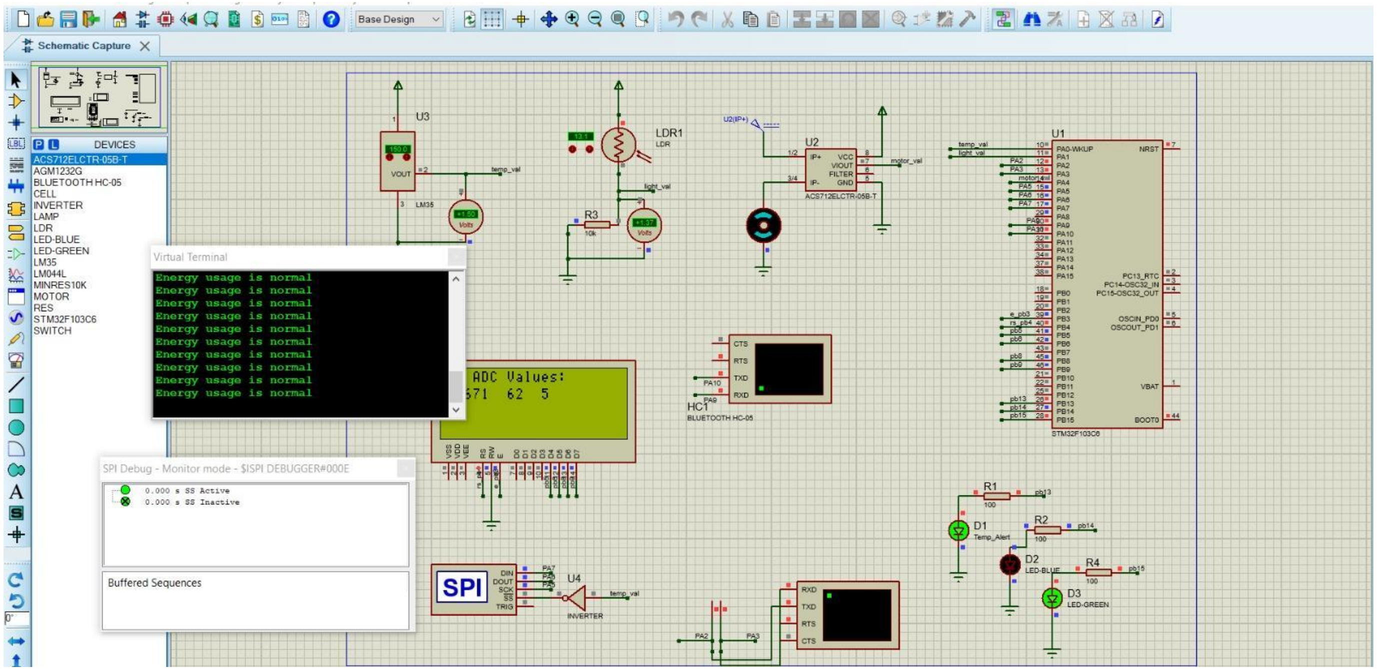
After completing the configuration on CubeMX and writing the project function in KeilMDK, we started to work on our Proteus design. First of all, we added our sensors and made the ADC connections and all other necessary connections and we also added a virtual terminal to see if we can get the data correctly. At this point we added a motor to detect the current passing through it in order to calculate the energy consumption. After that, we added the leds which work synchronously with the sensors. Then we added the Bluetooth module which allows us to communicate with our application and connect it to the terminal to see the output appropriately. We also used SPI for DMA which we aim to pass the temperature value.

V. Challenges

One of the biggest problems that we had was sending the data correctly to the LCD display and we also were not able to send our temperature data to the virtual terminal via UART appropriately. We were able to solve the issue after switching to Proteus 8.11. Another issue we had was Bluetooth connection between mobile application and microcontroller. We needed to add serial ports separately. We managed to get the ADC values as an interrupt in Keil, but unfortunately, the callback function did not work in proteus. We changed versions for this, we used different methods, and we got help from TAs, but unfortunately, we could not solve this problem.

VI. Project Screenshots



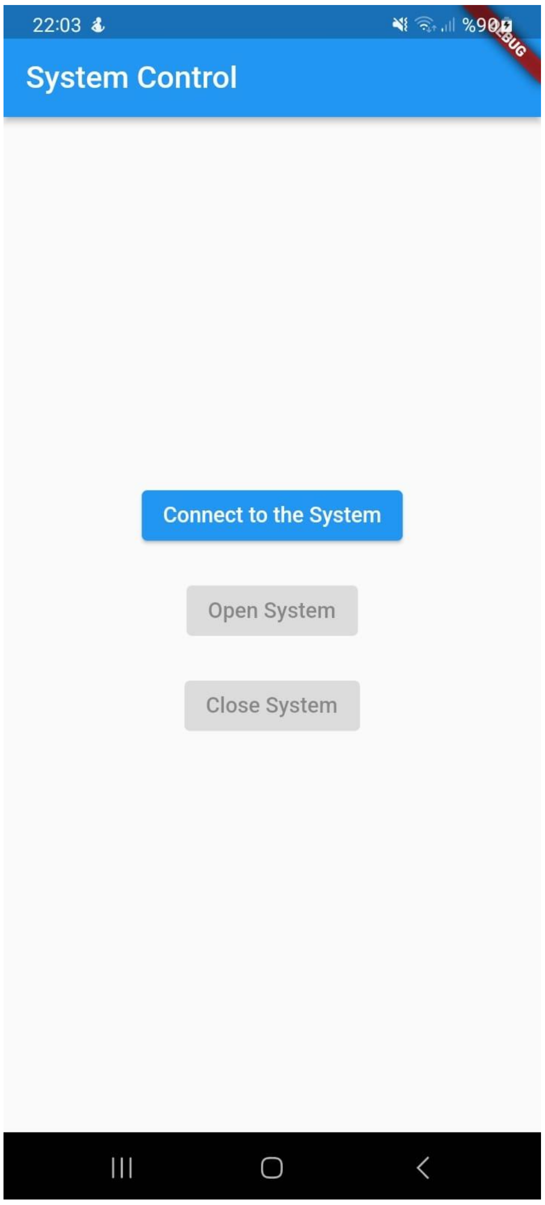


```
Energy usage is normal
Energy usage is normal
Energy usage is normal
Energy usage is normal
Energy usage is normal
Energy usage is normal
Energy usage is normal
Energy usage is normal
Energy usage is normal
Energy usage is normal
```

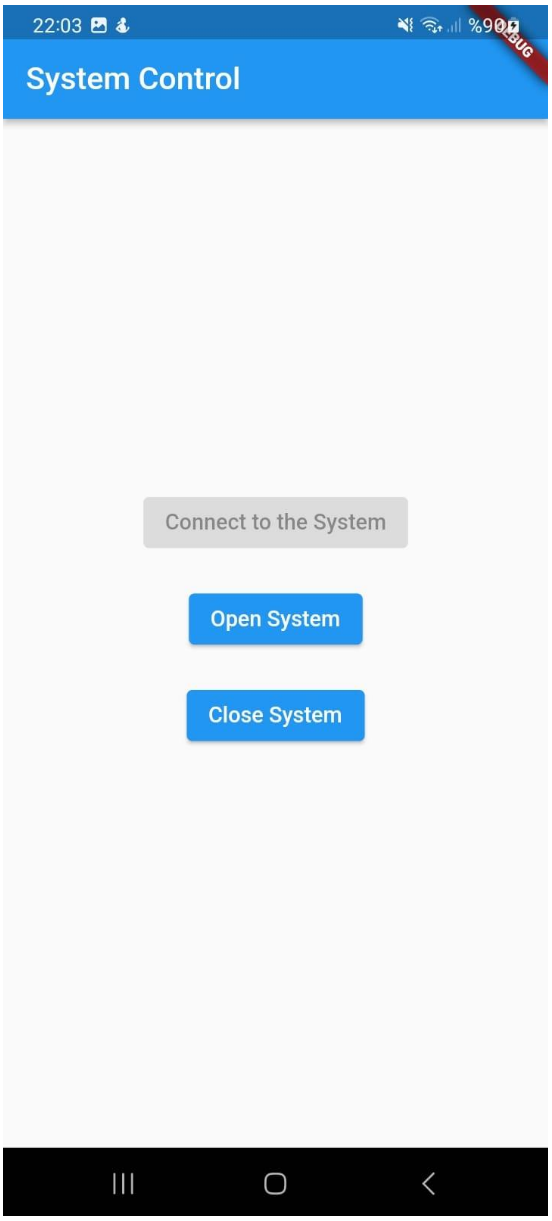
```
 ADC Values:
571  62  5
```

```
0.000 s SS Active
0.000 s SS Inactive
```

VII. Mobile Application Screenshot



The user will be able to connect to the system by pressing the active



After connecting to the system, the user can choose to turn on or turn off the

VIII. Demo Video

https://drive.google.com/drive/folders/14t-qJGQDgcDZOGvBD3nRhErHlM4KXlHO?usp=sharing

IX. Features table

Module/Feature	Types	Used types
GPIO	● Digital Output ● Digital Input ● Other	● Digital Output: Push/Pull Alert Led
Communication	●UART ● SPI ● I2C ● CAN, Others ● Using multiple devices at the same communication bus	● UART: - Interrupt for inputs and Bluetooth (HC-05) ● SPI: - Sending the temperature value from peripheral to memory
Watchdog Timer		
Interactivity (Leds, buttons, switches, touch etc.)		● Leds ● LCD Screen ● Virtual Terminal ● Bluetooth Module
Using Sensors	● Single ● Few ● Many or advanced one	● ACS712- Current sensor ● LM35- Temperature sensor ● LDR – Light sensor
Actuators	● Motors ● ..	● DC Motor for air conditioning
Timers	● Systick ● Advanced-basic Timers ● RTC alarm	Timer for DMA

Usage of polling		Taking ADC values
Usage of Interrupts	● No interrupt ● Single ● Few ● Many with different priorities	● Few interrupts
Error handling	● No error handling ● Few ● Full	● Few
Advanced Things that no code is provided during the course such as extra DAC, CAN etc.	extra	
Power saving	Sleep - standby - wakeup	
DMA		● Sending the temperature value from peripheral to memory
Ethernet-internet-Wi-Fi		
Writing own driver library for a peripheral		
Bluetooth		X
PCB	● External electronics design ● Using a different board ● Using MCU unit on your own design without the development board	
Usage of advanced tools e.g., Matlab, CubeAI etc. (Matlab code should run on MCU)		

Real time OS		
IoT	Making it work with: ● Node-red, Blynk etc. ● Mobile device interaction ● Time series database (InfluxDB)	● Mobile Device interaction