EE 304 Embedded System

Project

Final Report

Mehmet Anıl İRFANOĞLU

Ziya DENİZ

Instructor

Kasım Taşdemir

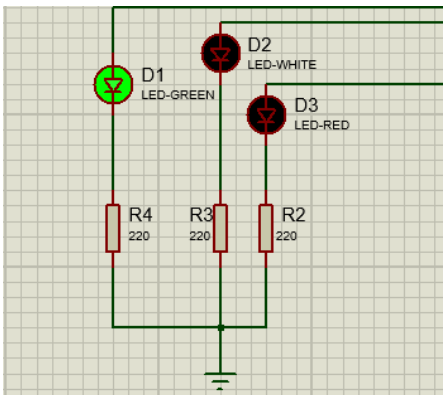
Objective:

Our smart business lighting project aims to automatically optimize the lighting system of businesses. It takes advantage of the smart device's saving mode function, turning the device off when not in use. It interacts with other lights such as daylight and provides the necessary amount of brightness to illuminate the environment. The brightness of the environment stays the same amount all the time, thus saving energy. According to research, this system saves up to 45%. Due to today's increasing energy prices, it will provide serious savings to businesses.

How Project Works:

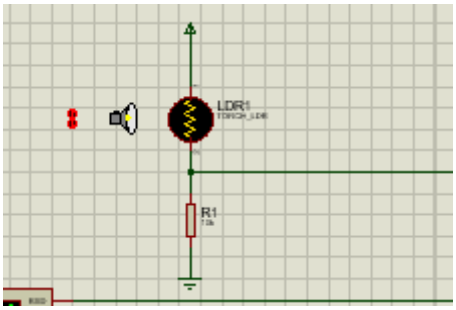
The main component of our project is the LDR sensor. The sensor measures the amount of the ambient brightness. To use this, we activated PA1 in Cube MX and changed the mode as independent. Because we want to receive data continuously, we enabled Continuous Conversion mode. The Rank is set as 1 and a sampling time to 55.5. We chose this length to ensure that the data was valid; if we chose a longer length, the data might not be correct. We changed the voltage of the lamp from the sensor data and kept the ambient brightness at the same level. We put a flame sensor in case the lamp explodes and causes a fire. When the flame sensor alarms, it makes a buzzer sound to attract people's attention. We used a bluetooth module and developed an android mobile application in order to connect with IoT and make the system more useful. Thanks to this app, users can enter the application from their phones instead of pressing the on/off button and turn it on/off from there.

Used Materials List:



1. GPIO

The purpose of the using leds are visualization of the savings rate. The red led means the lamp is using very much power. Other lamps are respectively white and green. Green represents most economic times.

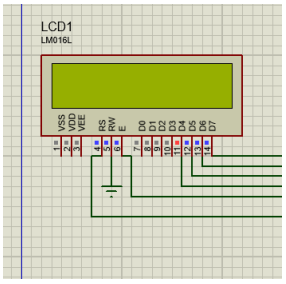


2. ADC & Sensor

We used a light sensor and the data we get from the light sensor is analog. After doing the necessary processes in our code, we put the data we received from here into the use of our system.

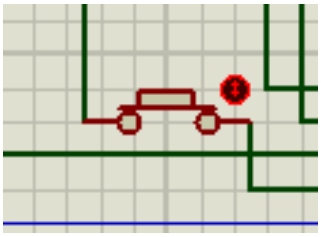
3. INTERRUPT

We connected these components to the timer for USART, Bluetooth and ADCto work, and this timer periodically invokes the interrupt necessary for these components to work.



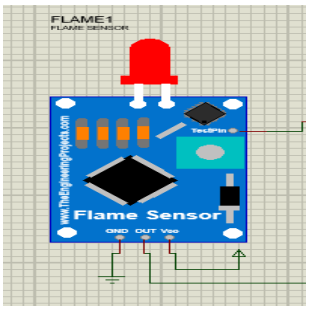
4. DISPLAY

After making the value we received from ADC user-friendly, we used LCD Display to printing ambient brightness.



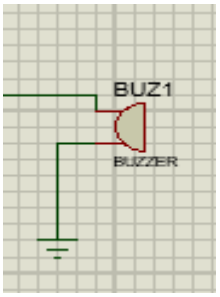
5. Timer & Button

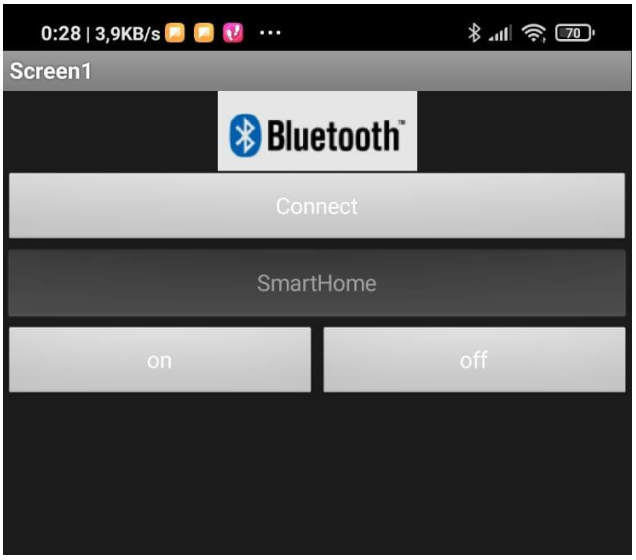
In our project, we added a timer so that the USART and light sensor, that is, the ADC, can be used. With the timer we used for USART, we ensured that the data we obtained was periodically transmitted to the terminal. With the timer we added for the ADC, instead of using the light sensor all the time, we made the sensor work at certain intervals.



6. Flame Sensor & Buzzer

We have integrated a flame warning system in our project to minimize the loss of life and property in a fire. When a flame is detected with the flame sensor, the buzzer gives a warning.

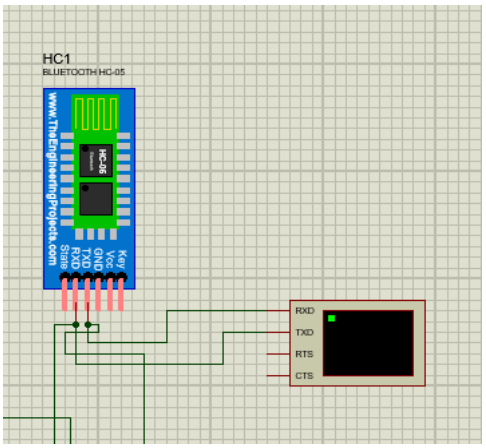




7. Mobile Application

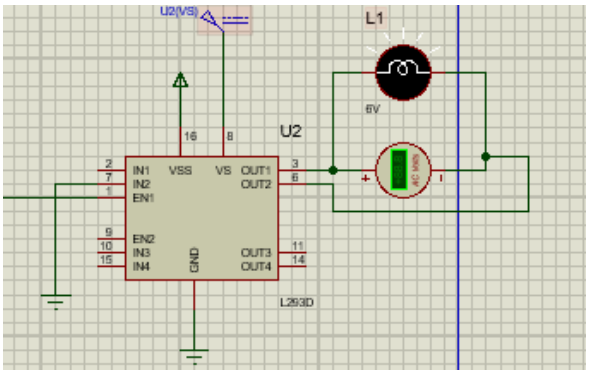
We developed a mobile application to turn our project into an IoT application and make it more useful. Thanks to this application, we can connect to the system from our phone

and perform the on and off operation. Users do not need to get up and press the button to activate or deactivate the system.



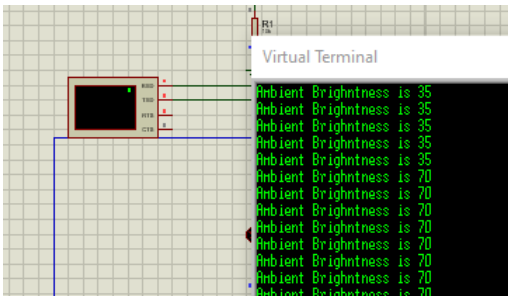
8. Bluetooth Module

To turn our concept into an IoT application and make it more helpful, we created a smartphone application to connect this app to our project. We put in a bluetooth driver. We may connect to the system from our phone and turn it on and off with the help of this driver. To activate or deactivate the system, users do not need to get up and click the button.



9. Motor Driver & Lamp

Thanks to the Motor Driver, we adjusted the voltage with the PWM signal and kept the ambient brightness constant. This motor driver increases or decreases the brightness of the lamp with the information from the sensor.



10. Communication

The data obtained with the light sensor and analyzed by the code we wrote was transferred to the terminal via USART communication.

11. Smart Algorithm

With the algorithm we will add according to the light intensity in the system, how much the user saves will be calculated. It will calculate the amount of time savings spent in areas with different light intensities. We have clock information, ambient brightness and voltage information given by the motor driver. Thanks to the algorithm developed for this information, users will be provided with a monthly annual reporting feature.

CubeMX:

We made the necessary port expansions in STMCube. These ports include GPIO, ADC, Timer, Interrupt and USART ports. After making the necessary period calculations and making the changes that affect the creation of the project, we generated our project.

Led: We have three leds so we open PB0, PB1, PB2. Because these are output devices, we set the GPIO modes to output-push-pull. We set labels respectively low_led, middle_led and high_led.

```
    // green lamp is on
    //lcd digits are setted
    if(ligthCopy>0 && ligthCopy<=33){
      GPIOB->ODR=(9 |firstDigit<<4 | (secondDigit<<8) |(thirdDigit<<12));
      TIM1->CCR4=(1010-pwmValue);

      // white lamp is on
      //lcd digits are setted
    }if(ligthCopy>33 && ligthCopy<=67){
      GPIOB->ODR=(10 |firstDigit<<4 | (secondDigit<<8) |(thirdDigit<<12));
      TIM1->CCR4=(1010-pwmValue);

    // red lamp is on
      //lcd digits are setted
    }if(ligthCopy>67 && ligthCopy<=101){
      GPIOB->ODR=(12 |firstDigit<<4 | (secondDigit<<8) |(thirdDigit<<12));
      TIM1->CCR4=(1010-pwmValue);
    }
```

UART: The UART bound rate was set at 9600. The word length was set to 8 bits. We have cross-connected PA9 and PA10 to UART because we used asynchronous mode.

```
// usart interrupt is triggered
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(htim-> Instance == TIM2){
    if(ligthCopy>0 && ligthCopy<=33){
      sprintf(uart_msg,"Ambient Brighntness is %d  \r\n", ligthCopy);
      HAL_UART_Transmit(&huart1, (uint8_t*)uart_msg, strlen(uart_msg),10);

    } if(ligthCopy>33 && ligthCopy<=67){
      sprintf(uart_msg,"Ambient Brighntness is %d  \r\n", ligthCopy);
      HAL_UART_Transmit(&huart1, (uint8_t*)uart_msg, strlen(uart_msg),10);

    }if(ligthCopy>67 && ligthCopy<=101){
      sprintf(uart_msg,"Ambient Brighntness is %d  \r\n", ligthCopy);
      HAL_UART_Transmit(&huart1, (uint8_t*)uart_msg, strlen(uart_msg),10);
    }
  }
}
```

ADC: The current light level of the environment is measured with the ADC, that is, the sensor, and the data obtained from this is used to adjust the brightness of the lamp in the next stage.

```
//this method is used to transform adc raw value to user friendly format
float mapValue(float adc_val, float min_Adc, float max_Adc, float min_Map, float max_Map){
  return adc_val*((max_Map-min_Map)/(max_Adc-min_Adc));
}
```

Display: We enabled the ADC in NVIC settings in cubemx. It has a printing value between 1-100. We show the current light density via a 7-segment LCD display.

```
    //LCD screen display
    //calculating digit values
    if(ligthCopy<0){
    Error_Handler();
    }
    ligthCopy=amountOfLigth;
    firstDigit=amountOfLigth%10;
    amountOfLigth=amountOfLigth/10;
    secondDigit=amountOfLigth%10;
    thirdDigit=amountOfLigth/10;
```

TIMER & INTERRUPT: We opened Tım1's channel 4 as CH4 and set it as an internal clock. Prescaler value is 7 and counter period is 999 from this formula. We controlled the brightness of the lamp with the EN1 pin on the motor. EN1 is connected to PA11.

```
clock/frequency.period -1 = prescaler value

8000000/1000*1000-1 = 7

We took prescaler value as 7.

We also used TIM2 as an interrupt. We chose Prescaler as 1999. The
counter period will repeat at 999. 4Hz, that is, every 250 micro seconds
and write the values.
```

ADC interrupt below.

```
// adc interrupt triggered
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
      // this method transforms light sensor's raw data
     adc_val_ldr=HAL_ADC_GetValue(&hadc1);
}
```

UART interrupt below.

```
// usart interrupt is triggered
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(htim-> Instance == TIM2){
    if(ligthCopy>0 && ligthCopy<=33){
      sprintf(uart_msg,"Ambient Brighntness is %d  \r\n", ligthCopy);
      HAL_UART_Transmit(&huart1, (uint8_t*)uart_msg, strlen(uart_msg),10);

    } if(ligthCopy>33 && ligthCopy<=67){
      sprintf(uart_msg,"Ambient Brighntness is %d  \r\n", ligthCopy);
      HAL_UART_Transmit(&huart1, (uint8_t*)uart_msg, strlen(uart_msg),10);

    }if(ligthCopy>67 && ligthCopy<=101){
      sprintf(uart_msg,"Ambient Brighntness is %d  \r\n", ligthCopy);
      HAL_UART_Transmit(&huart1, (uint8_t*)uart_msg, strlen(uart_msg),10);
    }
  }
}
```

KeilMDK:

In this part, we will talk about the things we do in general. Interrupt, Timer and ADC codes were automatically added to our project by STMCube. The next thing we had to do was to read data from the ADC at regular intervals. The reason for this was that when data was read from the ADC with polling, the system was seriously slowing down and reducing performance. That's why we created a Timer. This timer we created was invoking the interrupt with a certain period and we were reading the value given by the ADC, that is, the light sensor, in the callback of this interrupt. After the interrupt worked, we had light data, but this data was not user-friendly. For this reason, we have written a method to process this received data and make it user-friendly. After this data was converted to user friendly, we tried to determine the range in which the light intensity fell. For this, we have determined 3 intervals. These three ranges are high intensity, medium intensity and low intensity. If the light intensity is less than 33, the red LED lights up and we enter the high intensity area. If it falls between 33 and 67, the white led turns on and we enter the medium intensity area. If the light intensity is greater than 67, the green led will turn on and we will fall into the low intensity area. After determining which range the system will enter, this information is transferred to the 7-segment LCD and the user is provided to see this data physically. Then this data is sent to the device connected with USART so that it can be transferred to digital media. Since we connect the USART to a timer, the current light intensity is periodically transferred to the terminal and the user is enabled to see this data via the terminal.

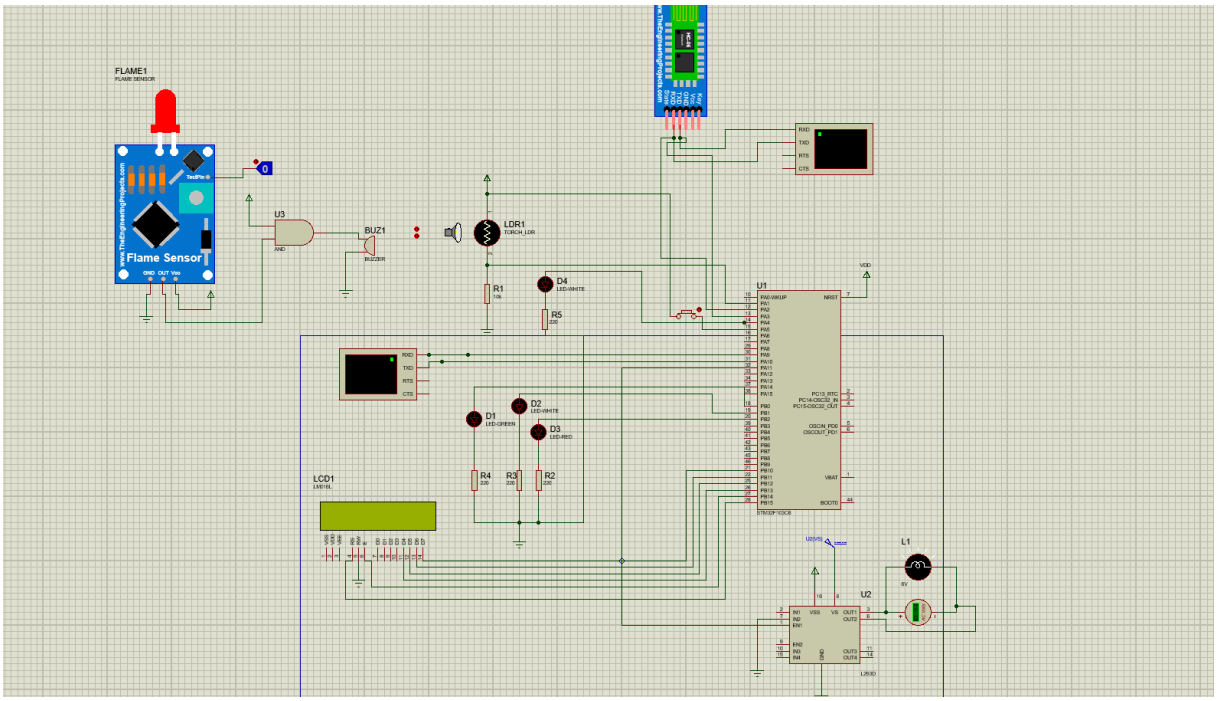
Proteus:

After compiling the code we wrote in Keil, we transferred it to proteus. As a result of our research on the internet, we downloaded and added the libraries related to the components that should be in our system. First of all, we added the light sensor and provided the necessary electricity supply, and also established the necessary port connection to receive ADC input. After we managed to get input from the ADC, we made the LED connections to the necessary ports and checked whether the system was in the required light intensity range. After successfully completing this step, we transferred the data we obtained using USART to the terminal. After solving the errors we received in this step, we conducted research on the internet on how to use 7-segment LCDs. After learning to use LCD, we added it to our project. We used three 7-segment LCD screens, but these screens were consuming all ports. We switched to the LCD screen to add new features. We also add an ADC and a lamp that changes the brightness. One day, we came across the news that dealt with the issue as a house fire. At the bottom of the news was a report saying that most home fires were caused by lamps. To avoid such a situation, we add a flame sensor and buzzer. When the sensors detect a flame, the buzzer sounds for warning. Thanks to this warning, we create a safe environment. Then we add a Bluetooth module to turn the whole system on and off. Users can turn the system on and off from their mobile applications or a button placed as an illuminated button.

Challenges:

The biggest problem we had was when we gathered the parts that we had used separately in a single project, and we got different errors from each other. We got errors on many of the light sensors we tried to use in our system. We finally found the right light sensor for our system, but this time we started to have problems in other parts of the project. While trying to show our data on the LCD, we used a 7-segment LCD and we had trouble installing this LCD, that is, the cable connections. It took us a long time to learn exactly how to send the output we want to these LCDs. After receiving many errors from the Proteus version, we decided to switch to the proteus version used in the course last year. After switching to the Proteus version, most of the problems were gone, and we solved the remaining problems via googling them.

Project Screenshot:



Features Table:

Module/Feature	Some of the possible types	We used these types:
GPIO	• ⭐Digital Output • ⭐Digital Input • ⭐Other	• ⭐Digital Output ○ Push/Pull x12 •

Communication	• UART ⭐,
	• SPI ⭐⭐,
	• I2C⭐⭐,
	• CAN⭐⭐⭐, Others
	• Using multiple devices at the same communication bus ⭐⭐⭐
		• UART ⭐,
		  ○ Interrupt
		• UART ⭐,
		  ○ Interrupt for bluetooth

Watchdog timer	⭐	
Interactivity (Leds, buttons, switches, touch etc.)	⭐⭐	• Leds x4 • Buzzer • LCD Display • Resistance 5x
Using sensors	Single ⭐, few ⭐⭐, many or advanced	• Sensor x2 type

	one ⭐⭐⭐⭐	○ Light Sensor (Analog) ○ Flame Sensor (Logical)
Actuators	• Motors, • ..	
Timers	Systick⭐, Advanced-basic Timers ⭐⭐, RTC alarm ⭐⭐	• TIM1 ○ Periodical interrupt generation • TIM2 ○ Periodical interrupt generation • Systick timer • RTC
Usage of polling	❌❌❌	
Usage of Interrupts	No interrupt ❌❌❌, Single ⭐, few ⭐⭐, many with different priorities ⭐⭐⭐	• ADC1_read_IT • Interrupt Button • Bluetooth Interrupt • USART_Transmit
Error handling	No error handling ❌ , few ⭐⭐, full ⭐⭐⭐	• In six places: ○ HAl_OK for ADC config ○ HAL_Base_Init checking for whether initialization setted correctly ○ HAL_TIM_ConfigClockSource whether clock setted correctly ○ HAL_UART_Init whether uart setted correctly ○ HAL_TIMEx_MasterConfigSynchronization whether synchronization setted correctly ○ To control get value of amount of light from sensor
Analog-digital Converter	ADC ⭐⭐, DAC⭐⭐	• ADC1 ○ Light Sensor input
Advanced Things that no code is provided during the course such as DAC, CAN etc.	extra ⭐⭐⭐	
Power saving	Sleep - standby - wakeup ⭐⭐⭐	

DMA	⭐⭐⭐	
Ethernet-internet-wifi	⭐⭐⭐	
Writing own driver library for a peripheral	⭐⭐⭐⭐⭐	
bluetooth	⭐⭐⭐⭐	• Bluetooth ○ With the bluetooth module users can connect the system with their phones. From their phone users can open and close the whole system.
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)	Via Bluetooth we can use our device from the phone.

Price List:

Stm32 223.01 ₺

LCD 44₺

Led 0.5*4=2 ₺

Lamp 4₺

Button 0.58

Console 42.43*2=85₺

LDR 30.80 ₺

Bluetooth Module 60₺

Buzzer 4₺

Flame Sensor 14₺

You can see different price results due to high inflation. In an example STM board was 223₺ but now 239₺.

Total Price is:462,39