HO CHI MINH CITY, UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

LOGIC DESIGN PROJECT (CO3091)

Topic

Internet of Things

A MINI HOME AUTOMATION SYSTEM

Advisor:	Le Trong Nhan
Students:	Nguyen Le Gia Nghi - 1952868 - CC04
	Doan Viet Tu - 1952521 - CC04
	Tran Trung Hieu - 1952684 - CC04

HO CHI MINH CITY, DECEMBER 2021

Contents

1 Introduction

1.1 Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

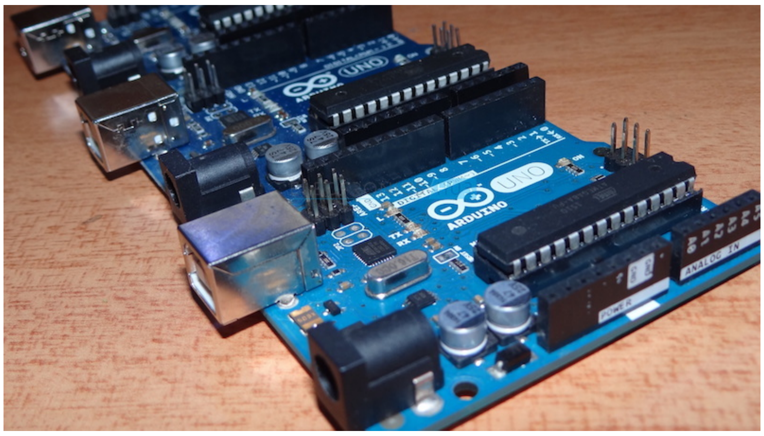
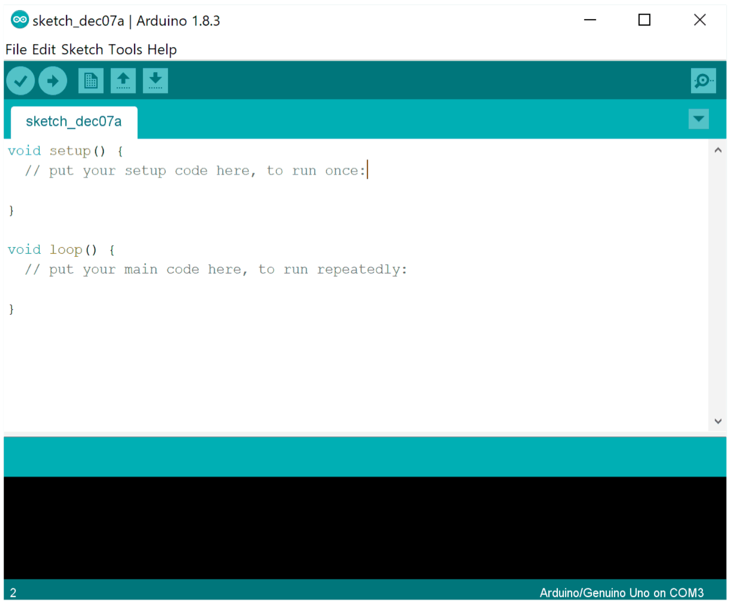


Figure 1. An Arduino board.

Arduino programming language (based on Wiring): Arduino programming language can be divided in three main parts: 1) Functions: for controlling the Arduino board and performing computations. 2) Variables: Arduino data types and constants. 3) Structure: the elements of Arduino (C++) code. Wiring: Wiring is an open-source programming framework for microcontrollers. Wiring allows writing cross-platform software to control devices attached to a wide range of microcontroller boards to create all kinds of creative coding, interactive objects, spaces or physical experiences. The framework is thoughtfully created with designers and artists in mind to encourage a community where beginners through experts from around the world share ideas, knowledge and their collective experience.

Arduino software (IDE) (based on Processing): The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board. Processing: Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology.


sketch_dec07a | Arduino 1.8.3
File Edit Sketch Tools Help
sketch_dec07a
2
Arduino/Genuino Uno on COM3

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 2. The workplace of Arduino software (IDE).

Arduino physical boards: Arduino boards are generally based on microcontrollers from Atmel Corporation like 8, 16 or 32 bit AVR architecture based microcontrollers. The important feature of the Arduino boards is the standard connectors. Using these connectors, we can connect the Arduino board to other devices like LEDs or add-on modules called Shields. There are many types of Arduino boards available in the market: Arduino UNO, Arduino Mega, Arduino Nano, Arduino Micro, Arduino Lilypad, ...

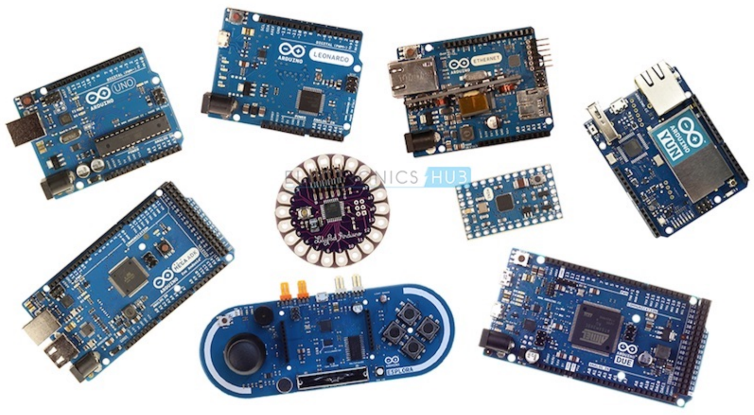


Figure 3. Some common types of Arduino boards.

Advantages of Arduino over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than 50

- Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

1.1.1 Arduino Uno

Arduino Uno is a very valuable addition in electronics that consists of a USB interface, 14 digital I/O pins (of which 6 Pins are used for PWM), 6 analog pins and an Atmega328 microcontroller. It also supports 3 communication protocols named Serial, I2C and SPI protocol. In this project, we are going to use the **Arduino Uno**. A few main features of Arduino Uno are shown in the below figure:

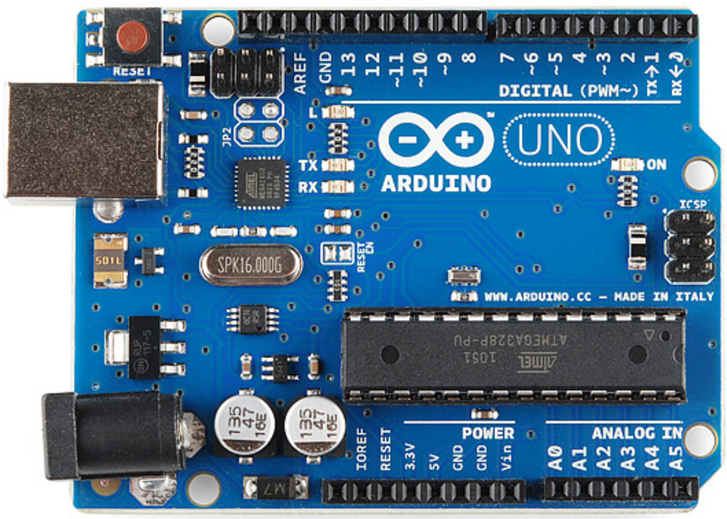
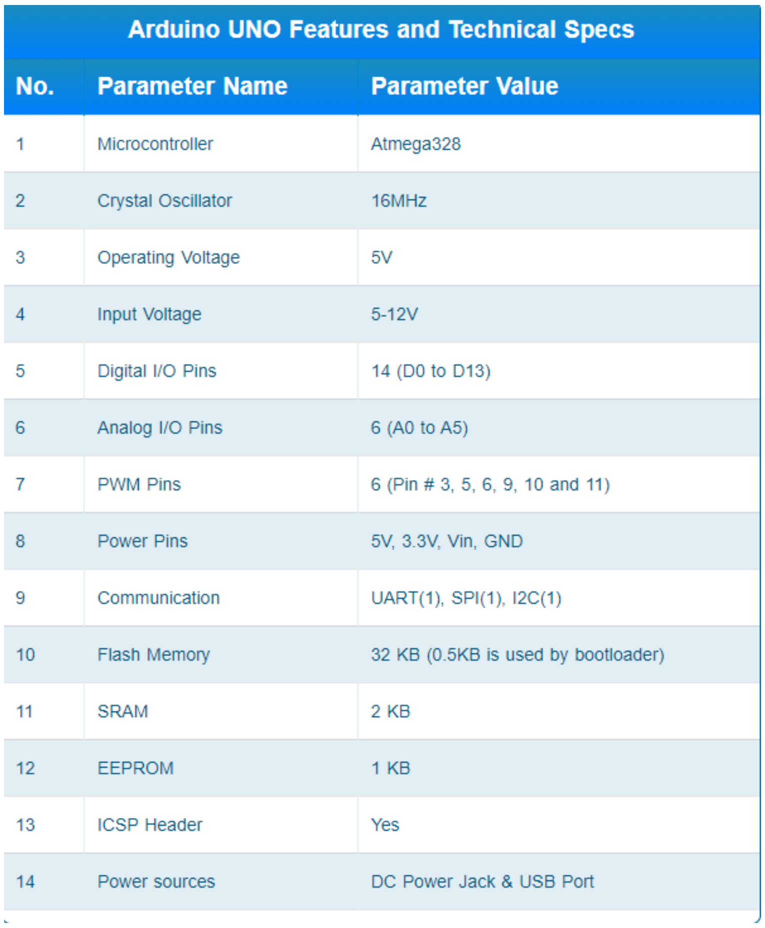


Figure 4. An Arduino Uno board.



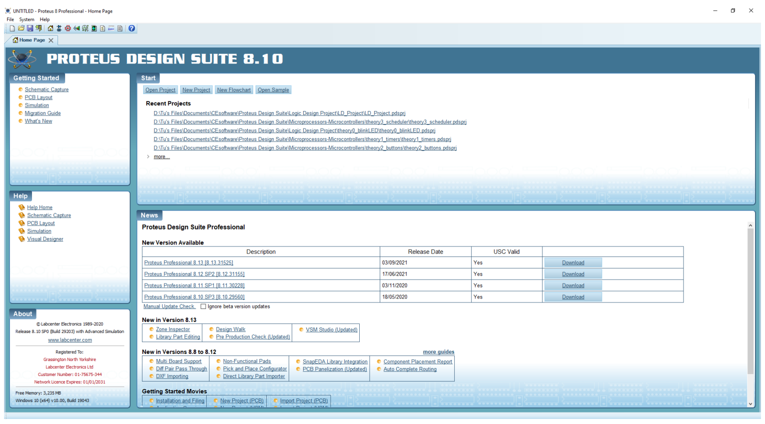
Arduino UNO Features and Technical Specs		
No.	Parameter Name	Parameter Value
1	Microcontroller	Atmega328
2	Crystal Oscillator	16MHz
3	Operating Voltage	5V
4	Input Voltage	5-12V
5	Digital I/O Pins	14 (D0 to D13)
6	Analog I/O Pins	6 (A0 to A5)
7	PWM Pins	6 (Pin # 3, 5, 6, 9, 10 and 11)
8	Power Pins	5V, 3.3V, Vin, GND
9	Communication	UART(1), SPI(1), I2C(1)
10	Flash Memory	32 KB (0.5KB is used by bootloader)
11	SRAM	2 KB
12	EEPROM	1 KB
13	ICSP Header	Yes
14	Power sources	DC Power Jack & USB Port

Figure 5. The technical specifications of Arduino Uno.

1.2 Proteus

Proteus Design Suite (designed by Labcenter Electronics Ltd.) is a software tool set, mainly used for creating schematics, simulating Electronics and Embedded Circuits and designing PCB Layouts.

There are 2 main parts of Proteus: Proteus ISIS and Proteus ARES. Proteus ISIS is used by Engineering students and professionals to create schematics and simulations of different electronic circuits. Proteus ARES is used for designing PCB Layouts of electronic circuits. It also provides features related to the three-dimensional view of design in PCB.



Recent Projects

D:\Tu's Files\Documents\CEsoftware\Proteus Design Suite\Logic Design Project\LD_Project\LD_Project.pdsprj
D:\Tu's Files\Documents\CEsoftware\Proteus Design Suite\Microprocessors-Microcontrollers\theory3_scheduler\theory3_scheduler.pdsprj
D:\Tu's Files\Documents\CEsoftware\Proteus Design Suite\Logic Design Project\theory0_blinkLED\theory0_blinkLED.pdsprj
D:\Tu's Files\Documents\CEsoftware\Proteus Design Suite\Microprocessors-Microcontrollers\theory1_timers\theory1_timers.pdsprj
D:\Tu's Files\Documents\CEsoftware\Proteus Design Suite\Microprocessors-Microcontrollers\theory2_buttons\theory2_buttons.pdsprj
more...

© Labcenter Electronics 1989-2020
Release 8.10 SP0 (Build 29203) with Advanced Simulation
www.labcenter.com
Registered To:
Grassington North Yorkshire
Labcenter Electronics Ltd
Customer Number: 01-75675-344
Network Licence Expires: 01/01/2031
Free Memory: 3,235 MB
Windows 10 (x64) v10.00, Build 19043

Proteus Design Suite Professional

New Version Available

Description	Release Date	USC Valid	
Proteus Professional 8.13 [8.13.31525]	03/09/2021	Yes	Download
Proteus Professional 8.12 SP2 [8.12.31155]	17/06/2021	Yes	Download
Proteus Professional 8.11 SP1 [8.11.30228]	03/11/2020	Yes	Download
Proteus Professional 8.10 SP3 [8.10.29560]	18/05/2020	Yes	Download

Figure 6. First look of Proteus Design Suite.

Benefits of using Proteus:

- Designing of circuits on the proteus takes less time than practical construction of the circuit.
- The possibility of error is less in software simulation such as loose connection that takes a lot of time to find out connections problems in a practical circuit.
- Circuit simulations provide the main feature that some components of circuits are not practical then you can construct your circuit on proteus.
- There is zero possibility of burning and damaging of any electronic component in proteus.
- The electronic tools that are very expensive can easily get in proteus such as an oscilloscope.
- Using proteus you can find different parents of circuits such as current, a voltage value of any component and resistance at any instant which is very difficult in a practical circuit.

1.3 Libraries

The Arduino environment can be extended through the use of libraries, just like most programming platforms. **Arduino libraries** are files written in C or C++ (.c, .cpp) which provide your sketches with extra functionality (e.g. the ability to control an LED matrix, or read an encoder, etc.). They were introduced in Arduino 0004. There are three general types of Arduino Libraries: standard libraries, library manager libraries and user installed libraries.

Some of the most popular libraries include: **Servo, Firmata, Software Serial, EEPROM, SD, GSM** and **LiquidCrystal**.

Let's dig into those two libraries: **LiquidCrystal.h** and **DHT.h**

1) LiquidCrystal sensor library
[Description]: This library allows an Arduino board to control LiquidCrystal displays (**LCDs**) based on the Hitachi HD44780 (or a compatible) chipset, which is found on most text-based LCDs.
[Compatibility]: This library is compatible with all architectures so you should be able to use it

on all the Arduino boards.
[NOTE]: The library works with in either 4- or 8-bit mode (i.e. using 4 or 8 data lines in addition to the rs, enable, and, optionally, the rw control lines).
[EXAMPLES]:

- Autoscroll: Shift text right and left.
- Blink: Control of the block-style cursor.
- Cursor: Control of the underscore-style cursor.
- Display: Quickly blank the display without losing what's on it.
- Hello World: Displays "hello world!" and the seconds since reset.
- Scroll: Scroll text left and right.
- Serial Display: Accepts serial input, displays it.
- Set Cursor: Set the cursor position.
- Text Direction: Control which way text flows from the cursor.

2) DHT sensor library
[Description]: Arduino library for Temperature and Humidity sensors
[Author]: Adafruit
[Maintainer]: Adafruit
[Compatibility]: This library is compatible with all architectures so you should be able to use it on all the Arduino boards.
[NOTE]: We have two versions of the DHT sensor: DHT11 and DHT22, they look a bit similar and have the same pinout, but have different characteristics. In this project, we are going to use DHT11. Here are the specifications of **DHT11**:

- Ultra low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20-80% humidity readings with 5% accuracy
- Good for 0-50°C temperature readings ±2°C accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

1.4 First project on Arduino

1.4.1 Brief

[TOPIC]: BUTTON DEBOUNCING
[Description]: This project aims to toggle an LED using buttons/switches with the issue of debouncing solved.
[Board]: Arduino Uno
[Simulation]: Proteus Design Suite
[Timer tick]: 10 ms
[Theory]: In practice, all mechanical switch contacts bounce (that is, turn on and off, repeatedly, for short period of time) after the switch is closed or opened as shown in the figure below.

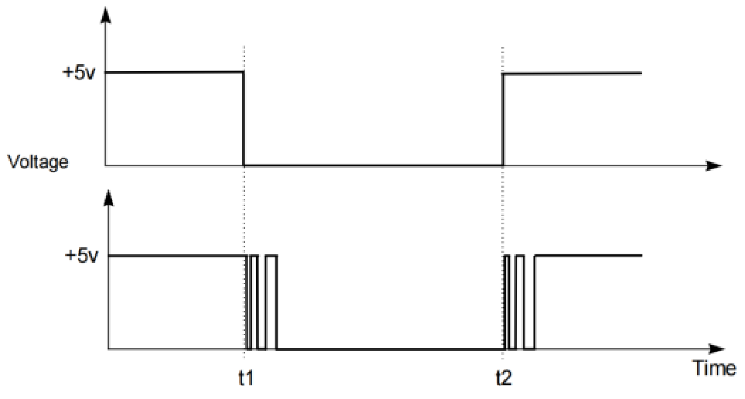


Figure 7.

Every system that uses any kind of mechanical switch must deal with the issue of debouncing. The key task is to make sure that one mechanical switch or button action is only read as one action by the MCU, even though the MCU will typically be fast enough to detect the unwanted switch bounces and treat them as separate events. Bouncing can be eliminated by special ICs or by RC circuitry, but in most cases debouncing is done in software because software is “free".

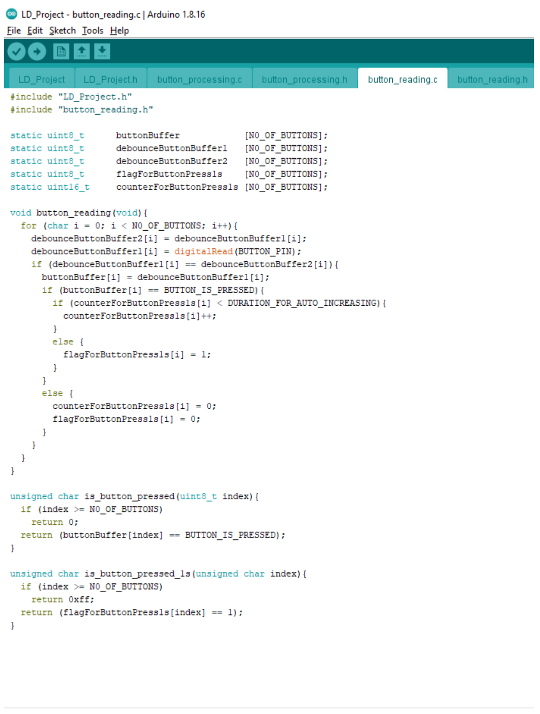
The relevant times we need to consider:

- Bounce time: most buttons seem to stop bouncing within 10ms.
- Button press time: the shortest time a user can press and release a button seems to be between 50 and 100ms.
- Response time: a user notices if the system response is 100ms after the button press, but not if it is 50ms after.

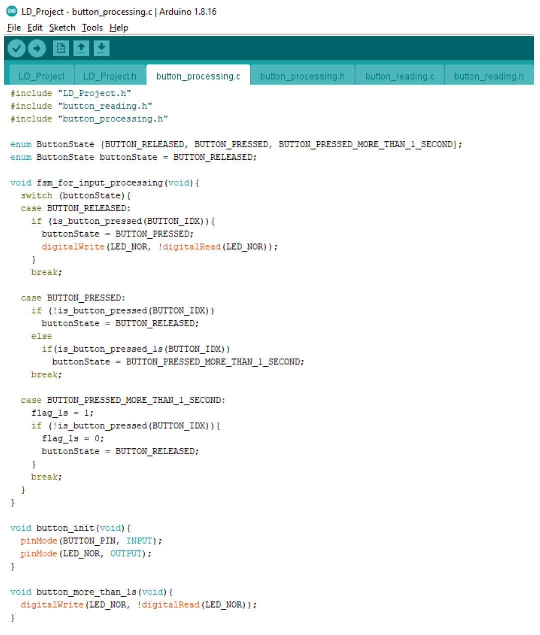
Combining all of these times, we can set a few goals

- Ignore all bouncing within 10ms
- Provide a response within 50ms of detecting a button push (or release)
- Be able to detect a 50ms push and a 50ms release

[Code explanation]:



```c
#include "LD_Project.h"
#include "button_reading.h"

static uint8_t      buttonBuffer            [NO_OF_BUTTONS];
static uint8_t      debounceButtonBuffer1   [NO_OF_BUTTONS];
static uint8_t      debounceButtonBuffer2   [NO_OF_BUTTONS];
static uint8_t      flagForButtonPress1s    [NO_OF_BUTTONS];
static uint16_t     counterForButtonPress1s [NO_OF_BUTTONS];

void button_reading(void){
  for (char i = 0; i < NO_OF_BUTTONS; i++){
    debounceButtonBuffer2[i] = debounceButtonBuffer1[i];
    debounceButtonBuffer1[i] = digitalRead(BUTTON_PIN);
    if (debounceButtonBuffer1[i] == debounceButtonBuffer2[i]){
      buttonBuffer[i] = debounceButtonBuffer1[i];
      if (buttonBuffer[i] == BUTTON_IS_PRESSED){
        if (counterForButtonPress1s[i] < DURATION_FOR_AUTO_INCREASING){
          counterForButtonPress1s[i]++;
        }
        else {
          flagForButtonPress1s[i] = 1;
        }
      }
      else {
        counterForButtonPress1s[i] = 0;
        flagForButtonPress1s[i] = 0;
      }
    }
  }
}

unsigned char is_button_pressed(uint8_t index){
  if (index >= NO_OF_BUTTONS)
    return 0;
  return (buttonBuffer[index] == BUTTON_IS_PRESSED);
}

unsigned char is_button_pressed_1s(unsigned char index){
  if (index >= NO_OF_BUTTONS)
    return 0xff;
  return (flagForButtonPress1s[index] == 1);
}
```



```c
#include "LD_Project.h"
#include "button_reading.h"
#include "button_processing.h"

enum ButtonState {BUTTON_RELEASED, BUTTON_PRESSED, BUTTON_PRESSED_MORE_THAN_1_SECOND};
enum ButtonState buttonState = BUTTON_RELEASED;

void fsm_for_input_processing(void){
  switch (buttonState){
  case BUTTON_RELEASED:
    if (is_button_pressed(BUTTON_IDX)){
      buttonState = BUTTON_PRESSED;
      digitalWrite(LED_NOR, !digitalRead(LED_NOR));
    }
    break;

  case BUTTON_PRESSED:
    if (!is_button_pressed(BUTTON_IDX))
      buttonState = BUTTON_RELEASED;
    else
      if(is_button_pressed_1s(BUTTON_IDX))
        buttonState = BUTTON_PRESSED_MORE_THAN_1_SECOND;
    break;

  case BUTTON_PRESSED_MORE_THAN_1_SECOND:
    flag_1s = 1;
    if (!is_button_pressed(BUTTON_IDX)){
      flag_1s = 0;
      buttonState = BUTTON_RELEASED;
    }
    break;
  }
}

void button_init(void){
  pinMode(BUTTON_PIN, INPUT);
  pinMode(LED_NOR, OUTPUT);
}

void button_more_than_1s(void){
  digitalWrite(LED_NOR, !digitalRead(LED_NOR));
}
```

Figure 8.Arduino code of button

The debouncing method here is to examine the keys (or buttons or switches) every N milliseconds, where N > 10ms (our specified button bounce upper limit) and N <= 50ms (our specified response time). We then have three possible outcomes every time we read a button:

- We read the button in the solid '0' state
- We read the button in the solid '1' state
- We read the button while it is bouncing (so we will get either a '0' or a '1'

Note that now we have not two but three button states: active (or pressed), inactive (or released), and indeterminate or invalid (in the middle of filtering, not yet filtered). In most cases we can treat the invalid state the same as the inactive state, since we care in most cases only about when we go active (from whatever state) and when we cease being active (to inactive or invalid). The function button_reading() must be called no more often than our debounce time (10ms).

1.4.2 System Architecture

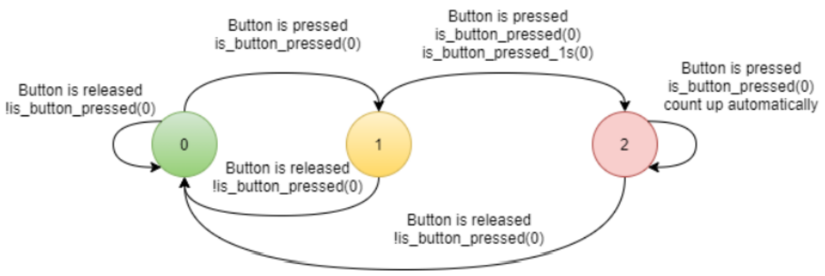


Figure 9.System of button

1.4.3 FSM

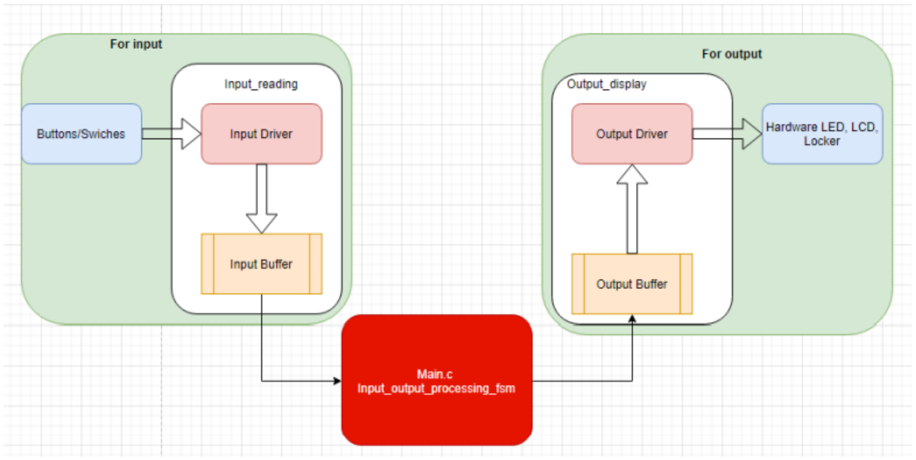


Figure 10.FSM of button

1.4.4 Demo

[Source]:
https://drive.google.com/file/d/1u8chx10xf5we2SG_iLlJ197xu-ItCKew/view?usp=sharing

2 Timer interrupt

2.1 Theory

Introduction

Timers are one of the most important features in modern micro-controllers. They allow us to measure how long something takes to execute, create non-blocking code, precisely control pin timing, and even run operating systems.

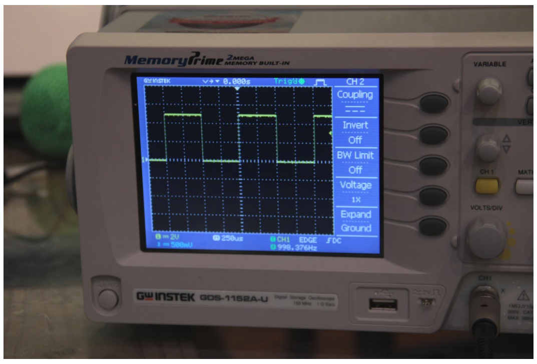
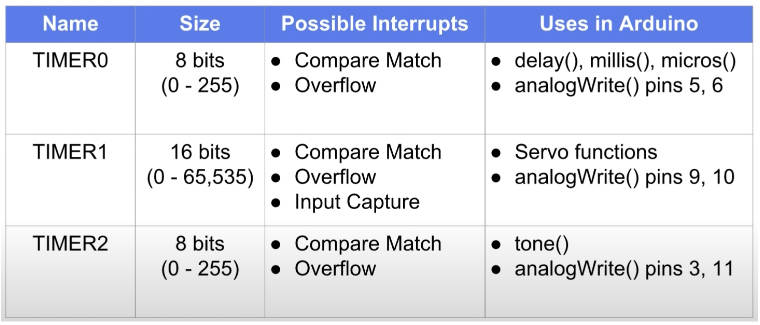


Figure 11.

Arduino Timers

The Arduino Uno has 3 timers: Timer0, Timer1 and Timer2. Timer interrupts allow you to perform a task at very specifically timed intervals regardless of what else is going on in your code.



Name	Size	Possible Interrupts	Uses in Arduino
TIMER0	8 bits (0 - 255)	• Compare Match • Overflow	• delay(), millis(), micros() • analogWrite() pins 5, 6
TIMER1	16 bits (0 - 65,535)	• Compare Match • Overflow • Input Capture	• Servo functions • analogWrite() pins 9, 10
TIMER2	8 bits (0 - 255)	• Compare Match • Overflow	• tone() • analogWrite() pins 3, 11

Figure 12. The difference between three timers in Arduino

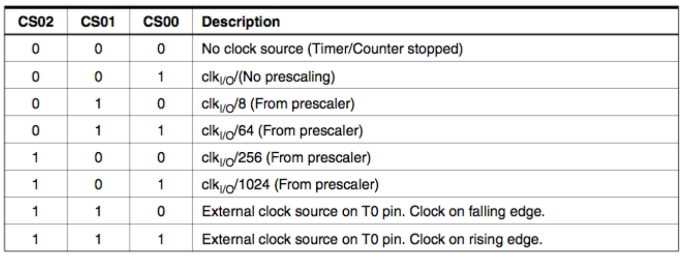
Normally when you write an Arduino sketch the Arduino performs all the commands encapsulated in the loop() function in the order that they are written, however, it's difficult to time events in the loop(). Some commands take longer than others to execute, some depend on conditional statements (if, while...) and some Arduino library functions (like digitalWrite or analogRead) are made up of many commands. Arduino timer interrupts allow you to momentarily pause the normal sequence of events taking place in the loop() function at precisely timed intervals, while you execute a separate set of commands. Once these commands are done the Arduino picks up again where it was in the loop().

Interrupts are useful for:

- Measuring an incoming signal at equally spaced intervals (constant sampling frequency)
- Calculating the time between two events
- Sending out a signal of a specific frequency
- Periodically checking for incoming serial data
- and much more...

To do interrupts, for now we will focus on the type called Clear Timer on Compare Match or CTC Mode. The main ideas presented here apply to the Mega and older boards as well, but the setup is a little different and the table below is specific to ATMEL 328/168.

Step 1: Prescalers and the Compare Match Register



CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{I/O}$/(No prescaling)
0	1	0	$\text{clk}_{I/O}$/8 (From prescaler)
0	1	1	$\text{clk}_{I/O}$/64 (From prescaler)
1	0	0	$\text{clk}_{I/O}$/256 (From prescaler)
1	0	1	$\text{clk}_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Figure 13.

Each of the timers has a counter that is incremented on each tick of the timer's clock. CTC timer interrupts are triggered when the counter reaches a specified value, stored in the compare match register. Once a timer counter reaches this value it will clear (reset to zero) on the next tick of the timer's clock, then it will continue to count up to the compare match value again. By choosing the compare match value and setting the speed at which the timer increments the counter, you can control the frequency of timer interrupts.

The Arduino clock runs at 16MHz, this is the fastest speed that the timers can increment their counters. At 16MHz each tick of the counter represents 1/16,000,000 of a second (63ns), so a counter will take 10/16,000,000 seconds to reach a value of 9 (counters are 0 indexed), and 100/16,000,000 seconds to reach a value of 99.

A prescaler dictates the speed of your timer according the the following equation:

(timer speed (Hz)) = (Arduino clock speed (16MHz)) / prescaler

Now you can calculate the interrupt frequency with the following equation:

interrupt frequency (Hz) = (Arduino clock speed 16,000,000Hz) / (prescaler * (compare match register + 1))
(the +1 is in there because the compare match register is zero indexed)

Rearranging the equation above, you can solve for the compare match register value that will give your desired interrupt frequency:

compare match register = [16,000,000Hz/ (prescaler * desired interrupt frequency)] - 1
(remember that when you use timers 0 and 2 this number must be less than 256, and less than 65536 for timer1)

Step 2: Structuring Timer Interrupts

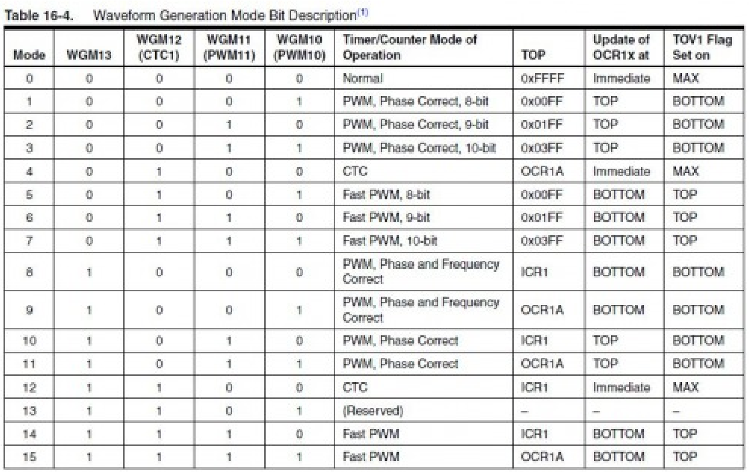


Table 16-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Figure 14.

Timer setup code is done inside the setup() function in an Arduino sketch.

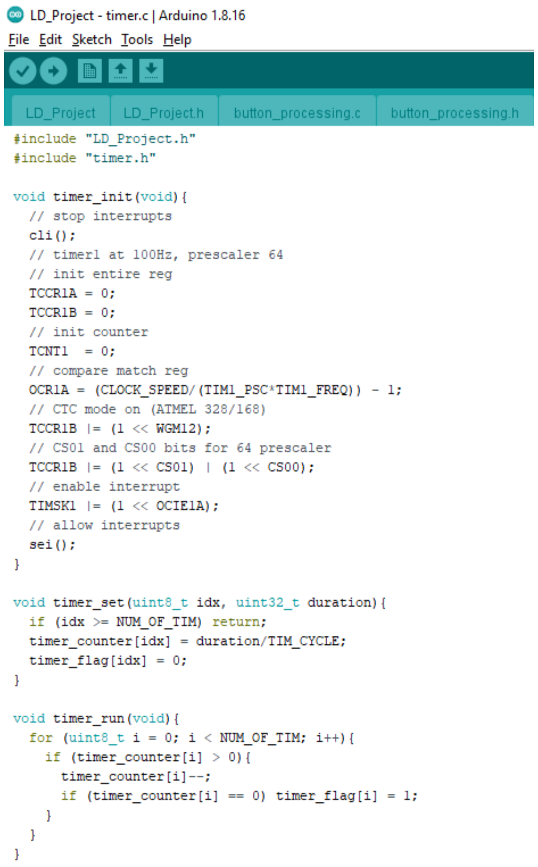
The commands you want to execute during these timer interrupts are located in the Arduino sketch encapsulated in the following:

ISR(TIMER0_COMPA_vect) //change the 0 to 1 for timer1 and 2 for timer2
//interrupt commands here

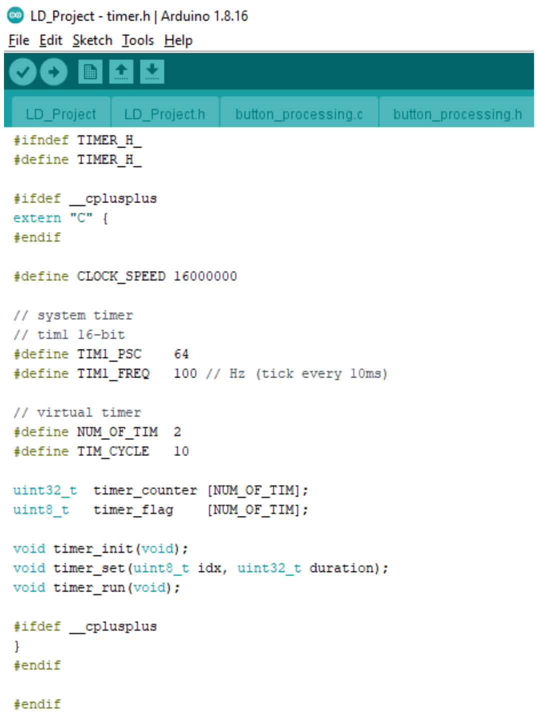
This bit of code should be located outside the setup() and loop() functions. Also, try to keep the interrupt routine as short as possible, especially if you are interrupting at a high frequency. It may even be worth addressing the ports/pins of the ATMEL chip directly instead of using the digitalWrite() and digitalRead() functions.

2.2 Implementation

2.2.1 Source code


LD_Project - timer.c | Arduino 1.8.16
File Edit Sketch Tools Help
LD_Project
LD_Project.h
button_processing.c
button_processing.h

```c
#include "LD_Project.h"
#include "timer.h"

void timer_init(void){
  // stop interrupts
  cli();
  // timer1 at 100Hz, prescaler 64
  // init entire reg
  TCCR1A = 0;
  TCCR1B = 0;
  // init counter
  TCNT1  = 0;
  // compare match reg
  OCR1A = (CLOCK_SPEED/(TIM1_PSC*TIM1_FREQ)) - 1;
  // CTC mode on (ATMEL 328/168)
  TCCR1B |= (1 << WGM12);
  // CS01 and CS00 bits for 64 prescaler
  TCCR1B |= (1 << CS01) | (1 << CS00);
  // enable interrupt
  TIMSK1 |= (1 << OCIE1A);
  // allow interrupts
  sei();
}

void timer_set(uint8_t idx, uint32_t duration){
  if (idx >= NUM_OF_TIM) return;
  timer_counter[idx] = duration/TIM_CYCLE;
  timer_flag[idx] = 0;
}

void timer_run(void){
  for (uint8_t i = 0; i < NUM_OF_TIM; i++){
    if (timer_counter[i] > 0){
      timer_counter[i]--;
      if (timer_counter[i] == 0) timer_flag[i] = 1;
    }
  }
}
```


LD_Project - timer.h | Arduino 1.8.16
File Edit Sketch Tools Help
LD_Project
LD_Project.h
button_processing.c
button_processing.h

```c
#ifndef TIMER_H_
#define TIMER_H_

#ifdef __cplusplus
extern "C" {
#endif

#define CLOCK_SPEED 16000000

// system timer
// tim1 16-bit
#define TIM1_PSC    64
#define TIM1_FREQ   100 // Hz (tick every 10ms)

// virtual timer
#define NUM_OF_TIM  2
#define TIM_CYCLE   10

uint32_t  timer_counter [NUM_OF_TIM];
uint8_t   timer_flag    [NUM_OF_TIM];

void timer_init(void);
void timer_set(uint8_t idx, uint32_t duration);
void timer_run(void);

#ifdef __cplusplus
}
#endif

#endif
```

Figure 15.

3 Indoor air monitoring

3.1 Theory

The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). It is fairly simple to use, but requires careful timing to grab data.

Since the DHT11 sensor resolution is 1, the values of the humidity and temperature are stored in two variables with type byte (8-bit unsigned).

The values of the humidity and temperature needs to be displayed on the LCD after the reading, between the DHT11 and the LCD we use the PCF8574.

The function "lcd.setCursor(,)" in the code highlight below is to set cursor position to index (0,0).

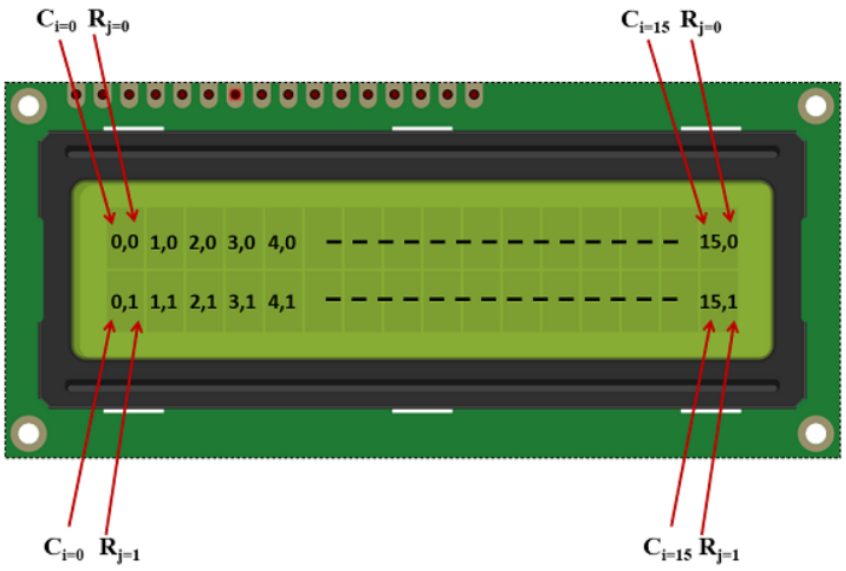


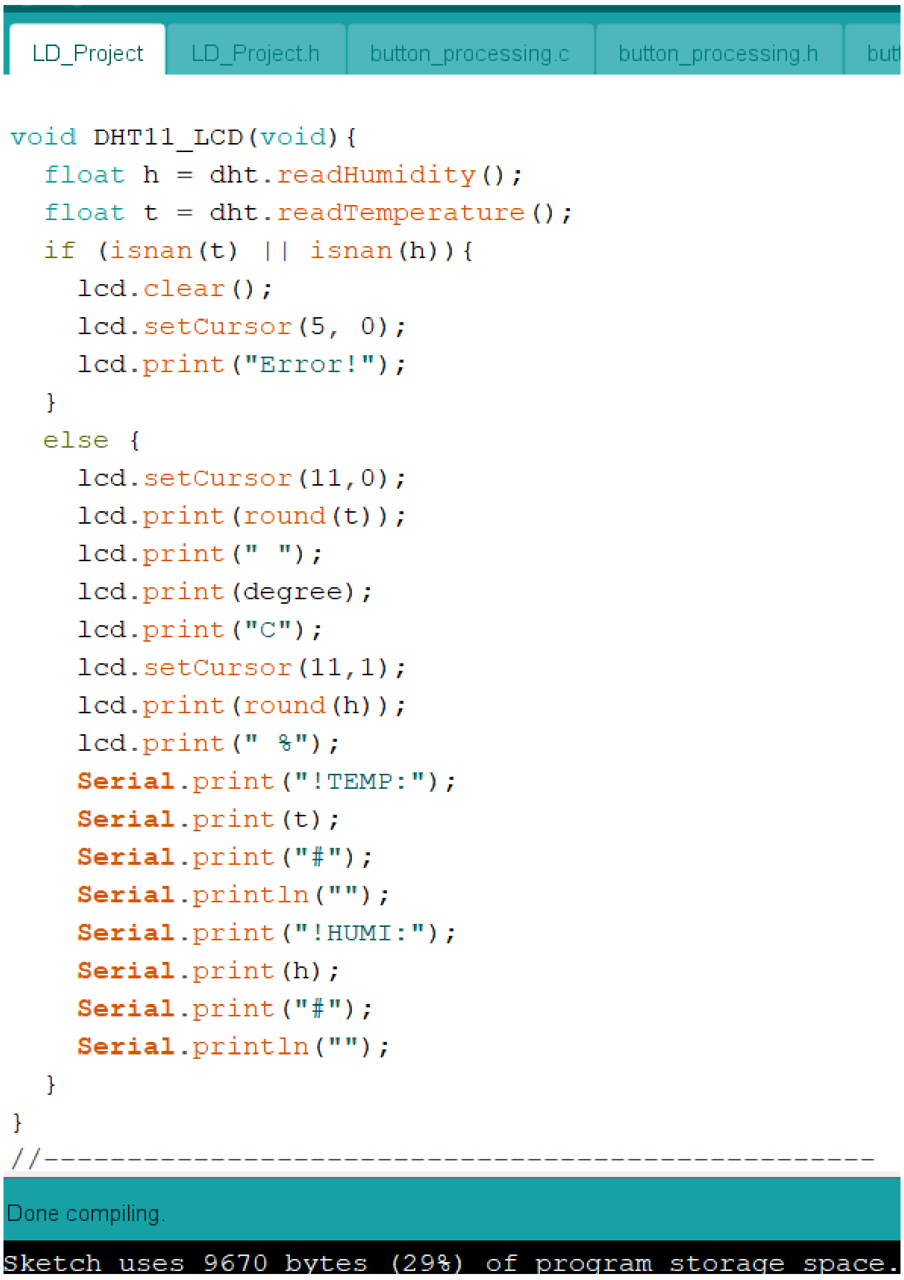
Figure 16. LCD

3.2 Implementation

3.2.1 Brief

[TOPIC]: INDOOR AIR MONITORING
[Description]: This feature aims to receive the data of temperature and humidity from the DHT11 sensor and then display on the LCD.
[Board]: Arduino Uno
[Simulation]: Proteus Design Suite

[Timer tick]: 10 ms
[Code highlight]:



```
void DHT11_LCD(void){
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(t) || isnan(h)){
    lcd.clear();
    lcd.setCursor(5, 0);
    lcd.print("Error!");
  }
  else {
    lcd.setCursor(11,0);
    lcd.print(round(t));
    lcd.print(" ");
    lcd.print(degree);
    lcd.print("C");
    lcd.setCursor(11,1);
    lcd.print(round(h));
    lcd.print(" %");
    Serial.print("!TEMP:");
    Serial.print(t);
    Serial.print("#");
    Serial.println("");
    Serial.print("!HUMI:");
    Serial.print(h);
    Serial.print("#");
    Serial.println("");
  }
}
//-----------------------------------------------------
```

```
Done compiling.
Sketch uses 9670 bytes (29%) of program storage space.
```

Figure 17. void DHT11_LCD()

3.2.2 System Architecture

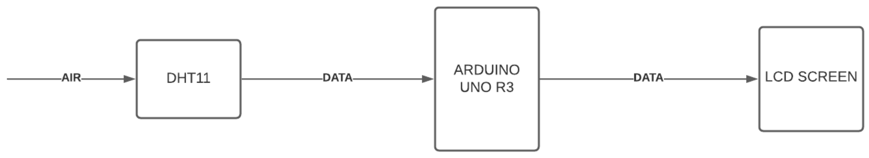


Figure 18.System of DHT11

3.2.3 FSM

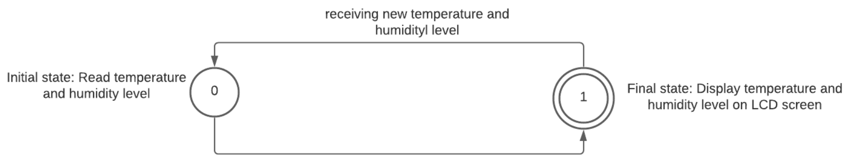

receiving new temperature and humidityl level
Initial state: Read temperature and humidity level
0
1
Final state: Display temperature and humidity level on LCD screen

Figure 19.FSM of DHT11

3.2.4 Demo

[Source]:
https://drive.google.com/file/d/1y-he7wXlH7N48s0mZEmDFXeB859PXcwX/view?usp=sharing

4 Simple automatic door

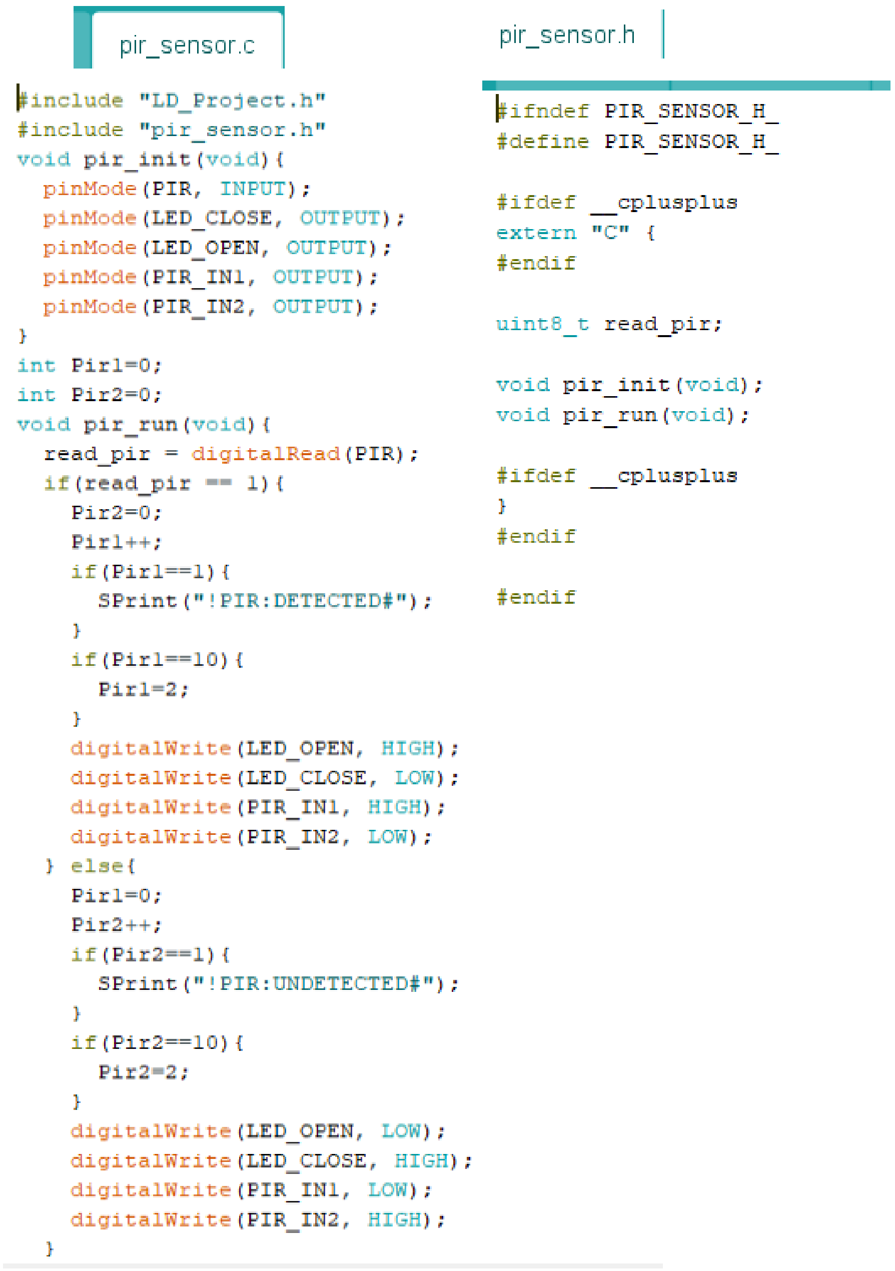
4.1 Theory

A passive infrared sensor (PIR sensor) is an electronic sensor that measures infrared (IR) light radiating from objects in its field of view. They are most often used in PIR-based motion detectors. PIR sensors are commonly used in security alarms and automatic lighting applications. PIR sensors detect general movement, but do not give information on who or what moved.

When an object, such as a person, passes in front of the background, such as a wall, the temperature at that point in the sensor's field of view will rise from room temperature to body temperature, and then back again. The sensor converts the resulting change in the incoming infrared radiation into a change in the output voltage, and this triggers the detection. Objects of similar temperature but different surface characteristics may also have a different infrared emission pattern, and thus moving them with respect to the background may trigger the detector as well.

4.2 Implementation

[TOPIC]: SIMPLE AUTOMATIC DOOR
[Description]: This feature aims to receive the signal of people entering the door (toggled by PIR sensor) to open or close the door automatically.
[Board]: Arduino Uno
[Simulation]: Proteus Design Suite
[Timer tick]: 10 ms
[Code highlight]:



```
#include "LD_Project.h"
#include "pir_sensor.h"
void pir_init(void){
  pinMode(PIR, INPUT);
  pinMode(LED_CLOSE, OUTPUT);
  pinMode(LED_OPEN, OUTPUT);
  pinMode(PIR_IN1, OUTPUT);
  pinMode(PIR_IN2, OUTPUT);
}
int Pir1=0;
int Pir2=0;
void pir_run(void){
  read_pir = digitalRead(PIR);
  if(read_pir == 1){
    Pir2=0;
    Pir1++;
    if(Pir1==1){
      SPrint("!PIR:DETECTED#");
    }
    if(Pir1==10){
      Pir1=2;
    }
    digitalWrite(LED_OPEN, HIGH);
    digitalWrite(LED_CLOSE, LOW);
    digitalWrite(PIR_IN1, HIGH);
    digitalWrite(PIR_IN2, LOW);
  } else{
    Pir1=0;
    Pir2++;
    if(Pir2==1){
      SPrint("!PIR:UNDETECTED#");
    }
    if(Pir2==10){
      Pir2=2;
    }
    digitalWrite(LED_OPEN, LOW);
    digitalWrite(LED_CLOSE, HIGH);
    digitalWrite(PIR_IN1, LOW);
    digitalWrite(PIR_IN2, HIGH);
  }
```

```
#ifndef PIR_SENSOR_H_
#define PIR_SENSOR_H_

#ifdef __cplusplus
extern "C" {
#endif

uint8_t read_pir;

void pir_init(void);
void pir_run(void);

#ifdef __cplusplus
}
#endif

#endif
```

Figure 20.PIR source code

4.2.1 System Architecture

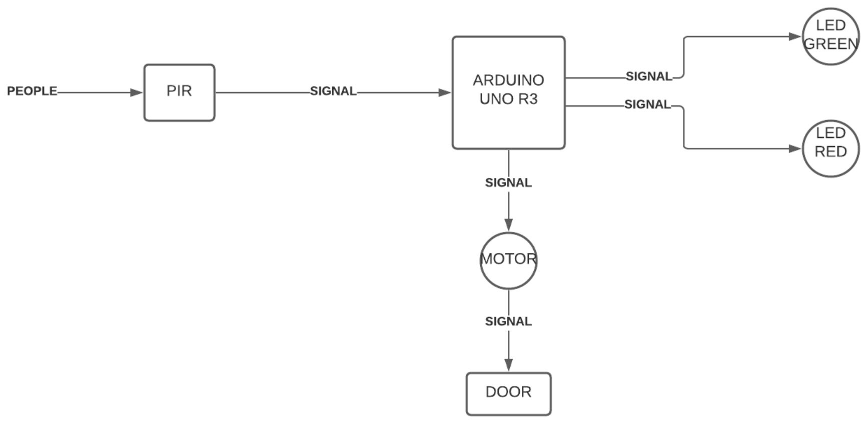


Figure 22.FSM of PIR

4.2.2 FSM

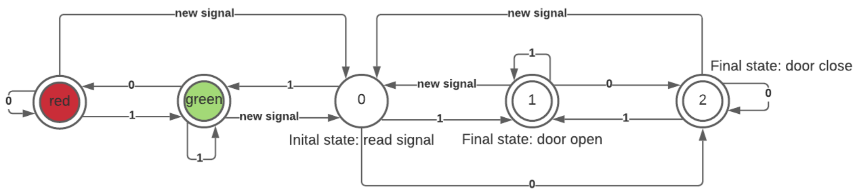


Figure 21.System of PIR

4.2.3 Demo

[Source]:
https://drive.google.com/file/d/19BQTGix81qcRxIpLGfo1kUx_6HuQU1Q6/view?usp=sharing

5 Detect soil moisture

5.1 Theory

The Soil Moisture Sensor measures soil moisture grace to the changes in electrical conductivity of the earth (soil resistance increases with drought).

The electrical resistance is measured between the two electrodes of the sensor.

A comparator activates a digital output when a adjutable threshold is exceeded.

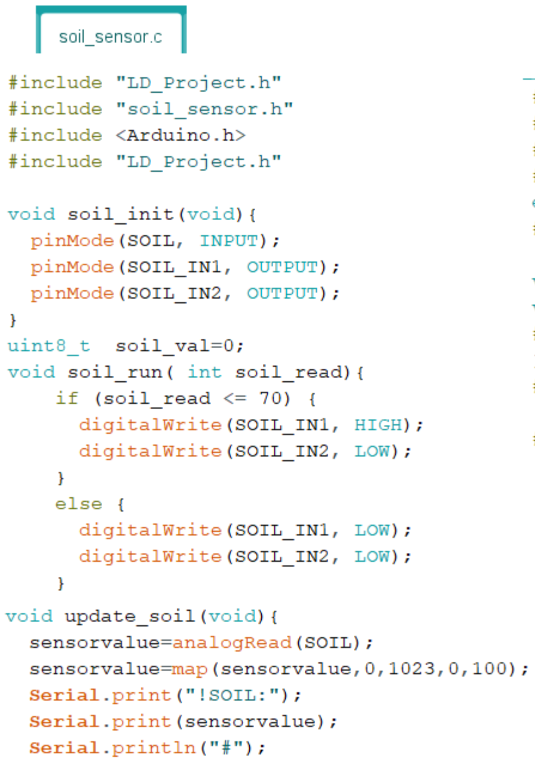
5.2 Implementation

5.2.1 Brief

[TOPIC]: DETECT SOIL MOISTURE
[Description]: This feature aims to read the soil moisture level from the sensor and adjust the pumping motor accordingly.

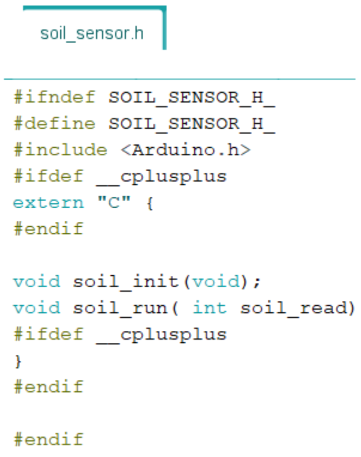
In details, the soil sensor will be read value between 10s duration, here because we use schematic in Proteus, we will use a "POT-HG" resistor to change the value of the Soil Sensor.Between 10s, The sensor will read the POT-HG resistor and then compare the condition to turn on or off the pumping machine. If the soil moisture is below 70%, we will open the pumping motor, otherwise it will be close.

[Board]: Arduino Uno
[Simulation]: Proteus Design Suite
[Timer tick]: 10 ms
[Code highlight]:



soil_sensor.c

```
#include "LD_Project.h"
#include "soil_sensor.h"
#include <Arduino.h>
#include "LD_Project.h"

void soil_init(void){
  pinMode(SOIL, INPUT);
  pinMode(SOIL_IN1, OUTPUT);
  pinMode(SOIL_IN2, OUTPUT);
}
uint8_t  soil_val=0;
void soil_run( int soil_read){
    if (soil_read <= 70) {
      digitalWrite(SOIL_IN1, HIGH);
      digitalWrite(SOIL_IN2, LOW);
    }
    else {
      digitalWrite(SOIL_IN1, LOW);
      digitalWrite(SOIL_IN2, LOW);
    }
void update_soil(void){
  sensorvalue=analogRead(SOIL);
  sensorvalue=map(sensorvalue,0,1023,0,100);
  Serial.print("!SOIL:");
  Serial.print(sensorvalue);
  Serial.println("#");
```



soil_sensor.h

```
#ifndef SOIL_SENSOR_H_
#define SOIL_SENSOR_H_
#include <Arduino.h>
#ifdef __cplusplus
extern "C" {
#endif

void soil_init(void);
void soil_run( int soil_read)
#ifdef __cplusplus
}
#endif

#endif
```

Figure 23.Soil Sensor source code

5.2.2 System Architecture

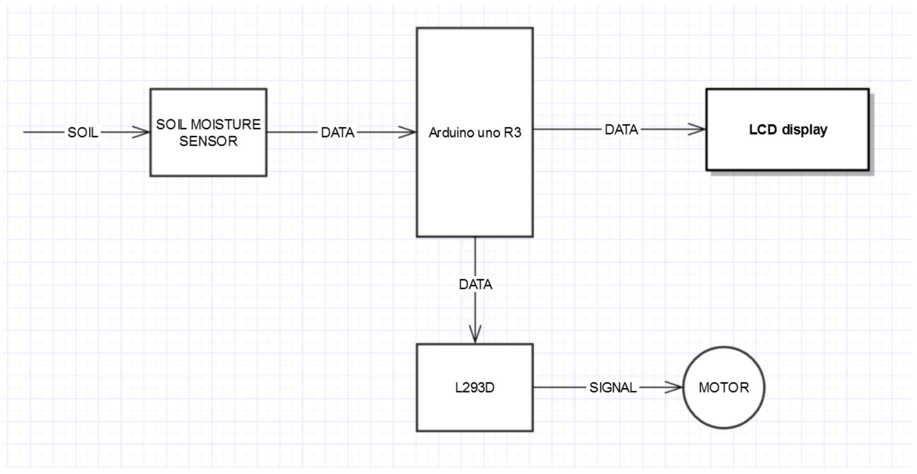


Figure 24.System of Soil sensor

5.2.3 FSM

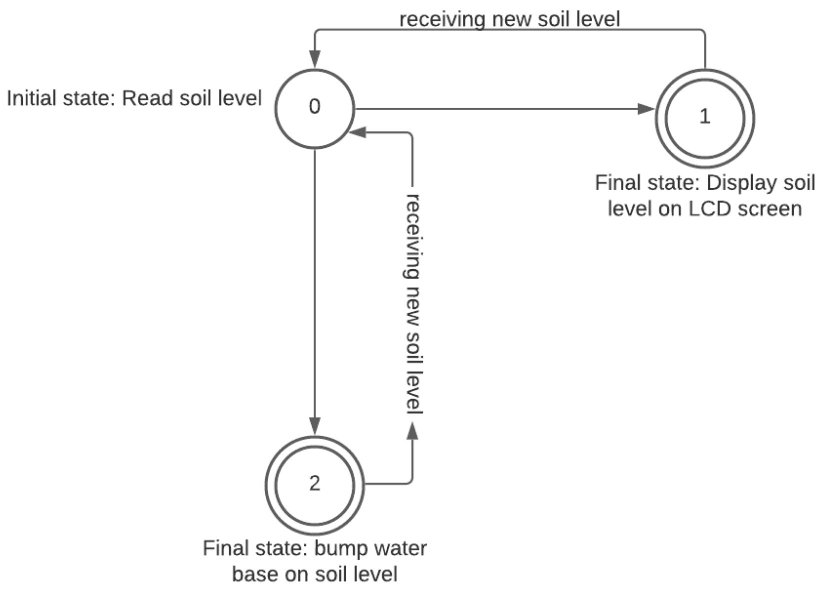

receiving new soil level
Initial state: Read soil level
0
1
Final state: Display soil level on LCD screen
receiving new soil level
2
Final state: bump water base on soil level

Figure 25.FSM soil

5.2.4 Demo

[Source]:
https://drive.google.com/file/d/1YYLPrjAWYsb3OGAFIwycLhrIA_KLQ5dK/view?usp=sharing

6 Detect gas density

6.1 Theory

This gas sensor uses a MQ-2 gas sensor to indicate if there is gas in the area. so the sensor has a threshold value (the threshold value is around the amount of smoke in there is a fire) in that if it goes above it starts alerting people in the vicinity. The alert just shows a red led if the density is high, yellow if medium and green if low or none.

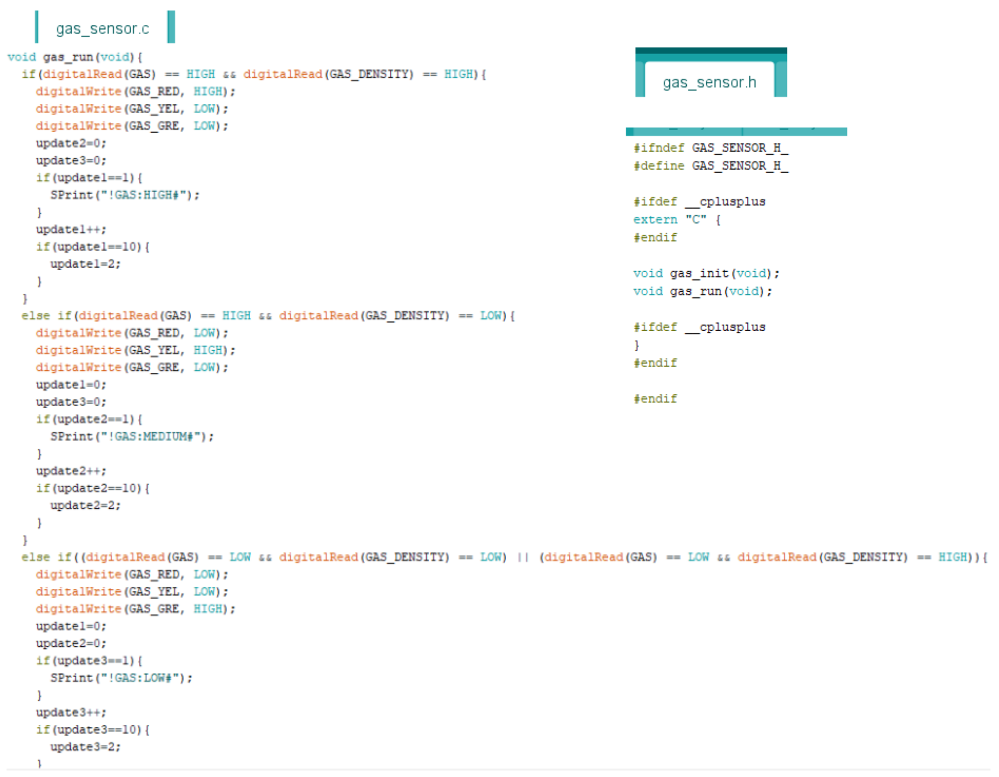
We use 2 gas sensor, 1 for the checking if the is gas or not, the second one to check the density of the gas.

Logic:
1. (Gas sensor=HIGH, Gas Density=HIGH) => Red Led Alert (HIGH)
2. (Gas sensor=HIGH, Gas Density=LOW) => Yellow Led Alert (MEDIUM)
3. (Gas sensor=LOW, Gas Density=LOW) => Green Led Alert (LOW)

6.2 Implementation

6.2.1 Brief

[TOPIC]: DETECT GAS DENSITY
[Description]: This feature aims to read the density from the sensor and alert us by the three LEDs with three different level of security.
[Board]: Arduino Uno
[Simulation]: Proteus Design Suite
[Timer tick]: 10 ms
[Code highlight]:



gas_sensor.c

```c
void gas_run(void){
  if(digitalRead(GAS) == HIGH && digitalRead(GAS_DENSITY) == HIGH){
    digitalWrite(GAS_RED, HIGH);
    digitalWrite(GAS_YEL, LOW);
    digitalWrite(GAS_GRE, LOW);
    update2=0;
    update3=0;
    if(update1==1){
      SPrint("!GAS:HIGH#");
    }
    update1++;
    if(update1==10){
      update1=2;
    }
  }
  else if(digitalRead(GAS) == HIGH && digitalRead(GAS_DENSITY) == LOW){
    digitalWrite(GAS_RED, LOW);
    digitalWrite(GAS_YEL, HIGH);
    digitalWrite(GAS_GRE, LOW);
    update1=0;
    update3=0;
    if(update2==1){
      SPrint("!GAS:MEDIUM#");
    }
    update2++;
    if(update2==10){
      update2=2;
    }
  }
  else if((digitalRead(GAS) == LOW && digitalRead(GAS_DENSITY) == LOW) || (digitalRead(GAS) == LOW && digitalRead(GAS_DENSITY) == HIGH)){
    digitalWrite(GAS_RED, LOW);
    digitalWrite(GAS_YEL, LOW);
    digitalWrite(GAS_GRE, HIGH);
    update1=0;
    update2=0;
    if(update3==1){
      SPrint("!GAS:LOW#");
    }
    update3++;
    if(update3==10){
      update3=2;
    }
```

gas_sensor.h

```c
#ifndef GAS_SENSOR_H_
#define GAS_SENSOR_H_

#ifdef __cplusplus
extern "C" {
#endif

void gas_init(void);
void gas_run(void);

#ifdef __cplusplus
}
#endif

#endif
```

Figure 26.Gas sensor source code

6.2.2 System Architecture

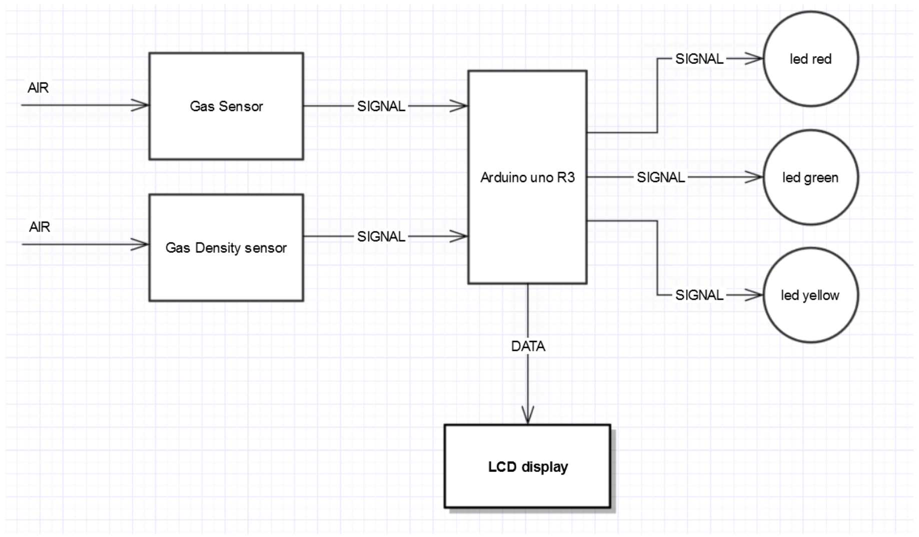


Figure 27.System of Gas sensor

6.2.3 FSM

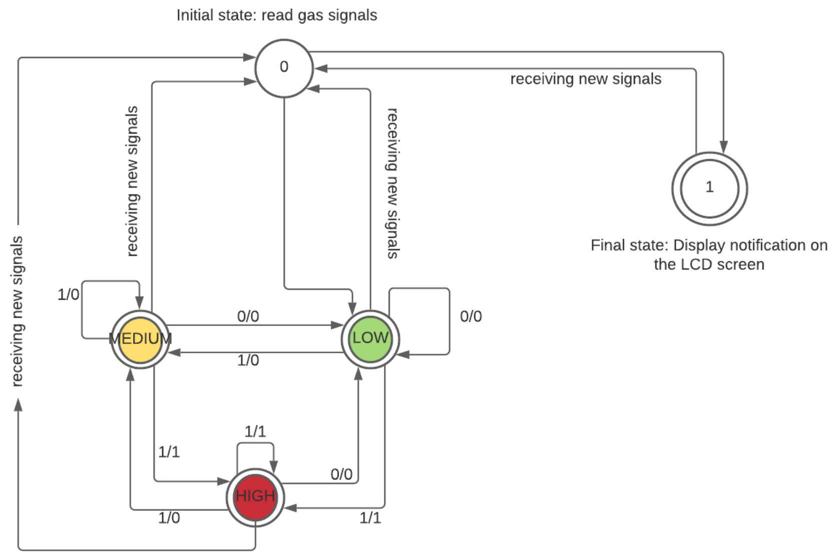


Figure 28.FSM of Gas sensor

6.2.4 Demo

[Source]:
https://drive.google.com/file/d/1XeTtLyJmRPkrlZlS6RKYP_k-cpCXb5to/view?usp=sharing

7 Server configuration

7.1 Introduction

This system is Gateway IoT development using Python. This system consists of two main viewing sections which are an schecmatic on Proteus of the smart home and an Adafruit IO server. The data (such as temperature and humidity) are displayed on the virtual terminal and Adafruit IO. Connect between block is wireless (Gateway - Proteus).

7.1.1 Create virtual ports

By using the application we can create 2 virtual port.
1. For Gateway (using Python/ Pycharm) (COM5).
2. Connect with COMPIM in Proteus (COM4).

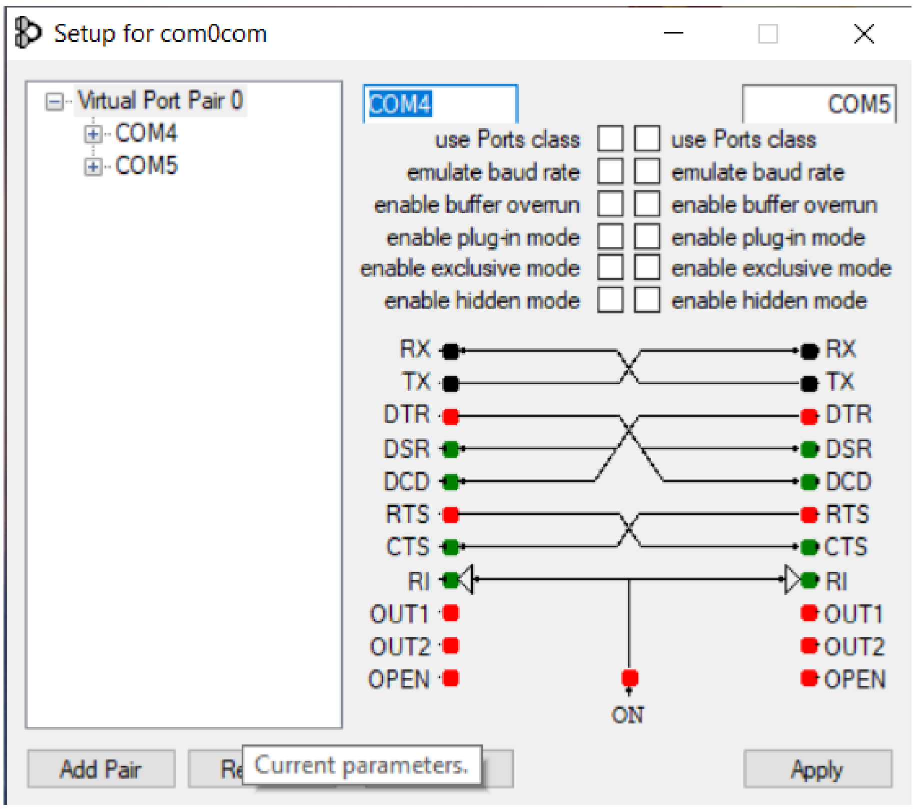


Figure 29.Com0Com

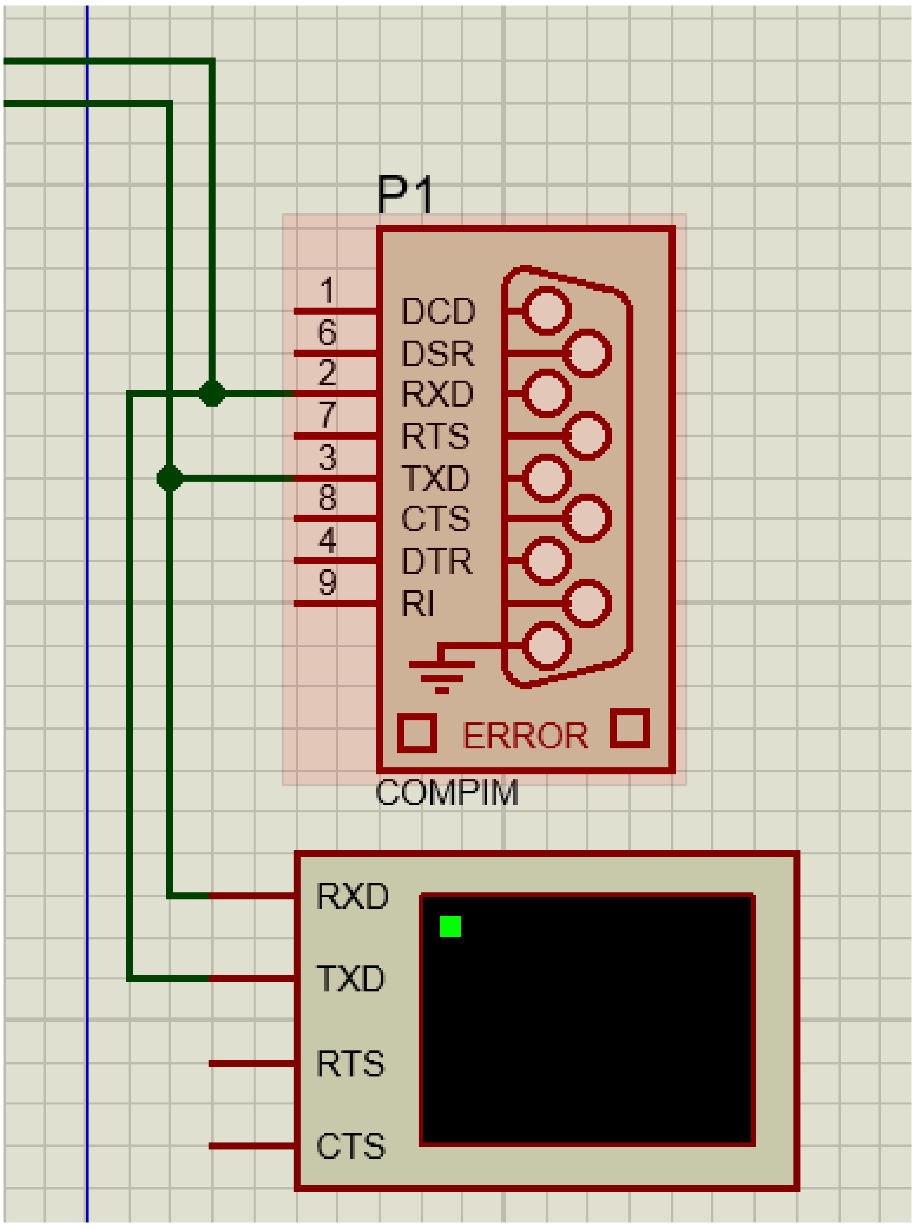
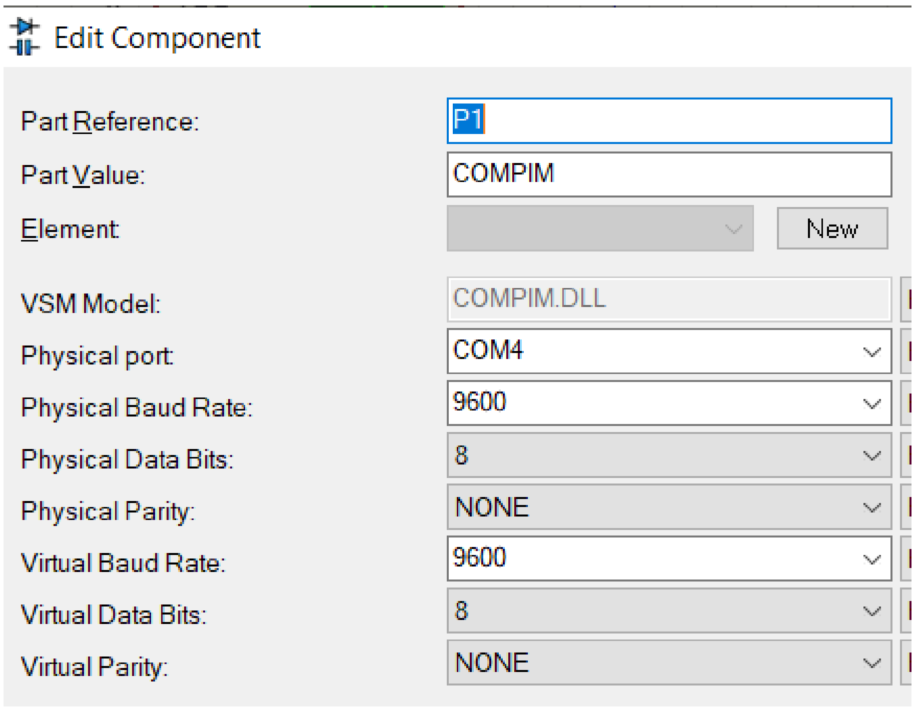


Figure 30.COMPIM


Edit Component

Part Reference:	P1
Part Value:	COMPIM
Element:	
VSM Model:	COMPIM.DLL
Physical port:	COM4
Physical Baud Rate:	9600
Physical Data Bits:	8
Physical Parity:	NONE
Virtual Baud Rate:	9600
Virtual Data Bits:	8
Virtual Parity:	NONE

Figure 31.COMPIM setting

7.1.2 Create dashboard

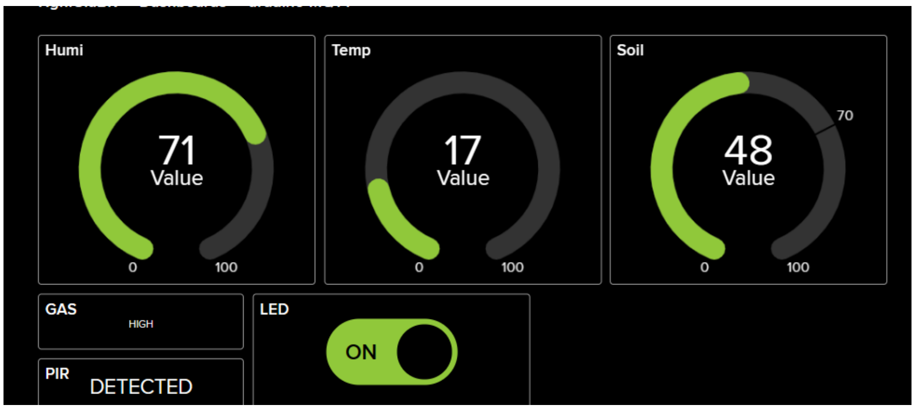


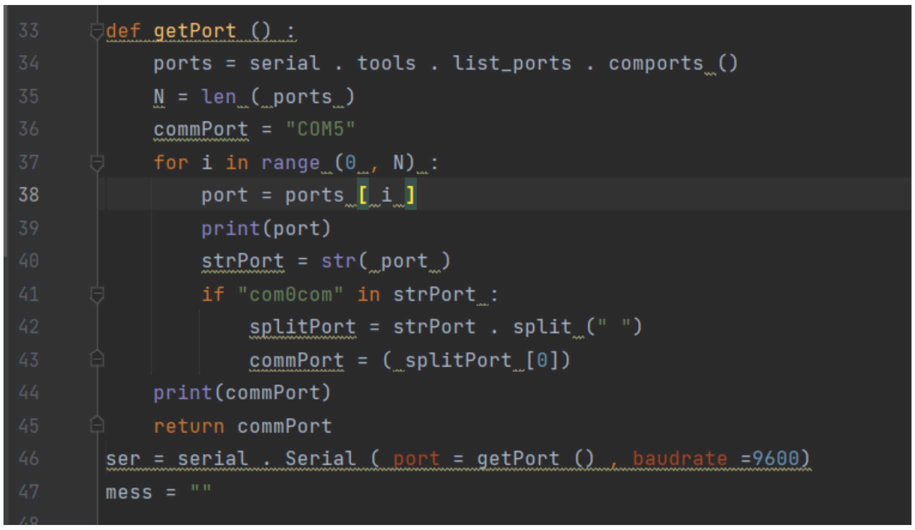
Figure 32.Adafruit Dashboard

Link for the Ardafruit :"https://io.adafruit.com/NghiGiaBK/dashboards/arduino-mqtt"

8 Gateway implementation

8.1 UART received data

As the data is received under a format, provide the source code to extract (or decode the data).

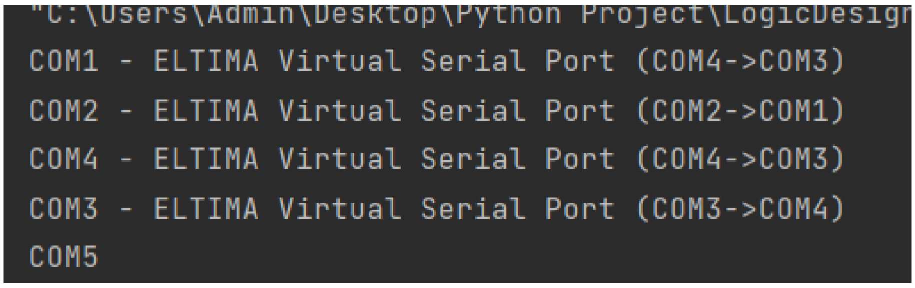


```
def getPort () :
    ports = serial . tools . list_ports . comports ()
    N = len ( ports )
    commPort = "COM5"
    for i in range (0 , N) :
        port = ports [ i ]
        print(port)
        strPort = str( port )
        if "com0com" in strPort :
            splitPort = strPort . split (" ")
            commPort = ( splitPort [0])
    print(commPort)
    return commPort
ser = serial . Serial ( port = getPort () , baudrate =9600)
mess = ""
```

Figure 33.Gateway connection

Here we set the gateway will be connected with the Port5 in com0com application which create for us 2 virtual ports, the baudrate will be 9600(as default).

The ("ports = serial.tools.listports.comports ()") will be collect all the ports available in the comupter.



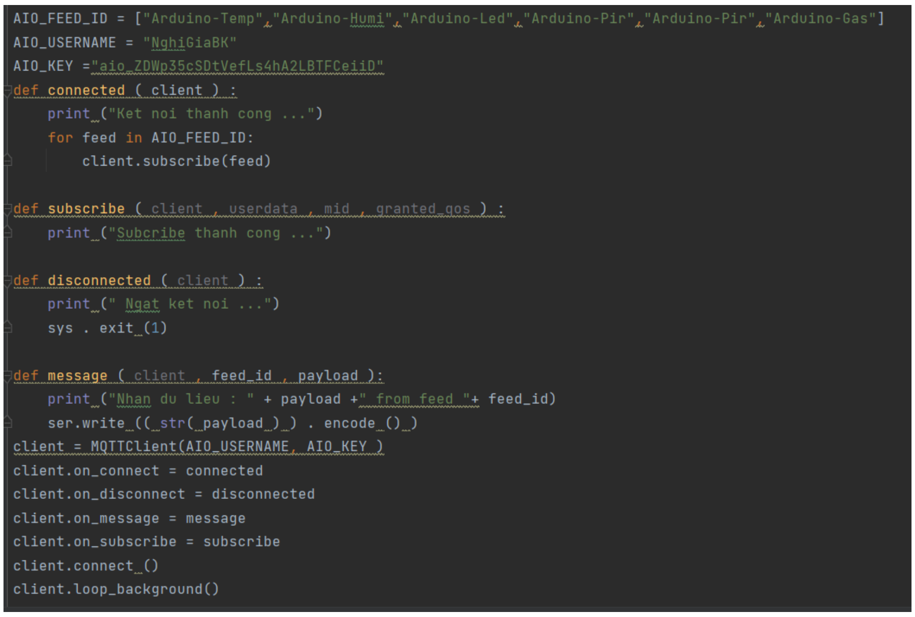
```
"C:\Users\Admin\Desktop\Python Project\LogicDesig
COM1 - ELTIMA Virtual Serial Port (COM4->COM3)
COM2 - ELTIMA Virtual Serial Port (COM2->COM1)
COM4 - ELTIMA Virtual Serial Port (COM4->COM3)
COM3 - ELTIMA Virtual Serial Port (COM3->COM4)
COM5
```

Figure 34.Ports available

The "for loop" will check for free port so that the gateway can connect with (here we will set it to Com5).

8.2 Data uploading process

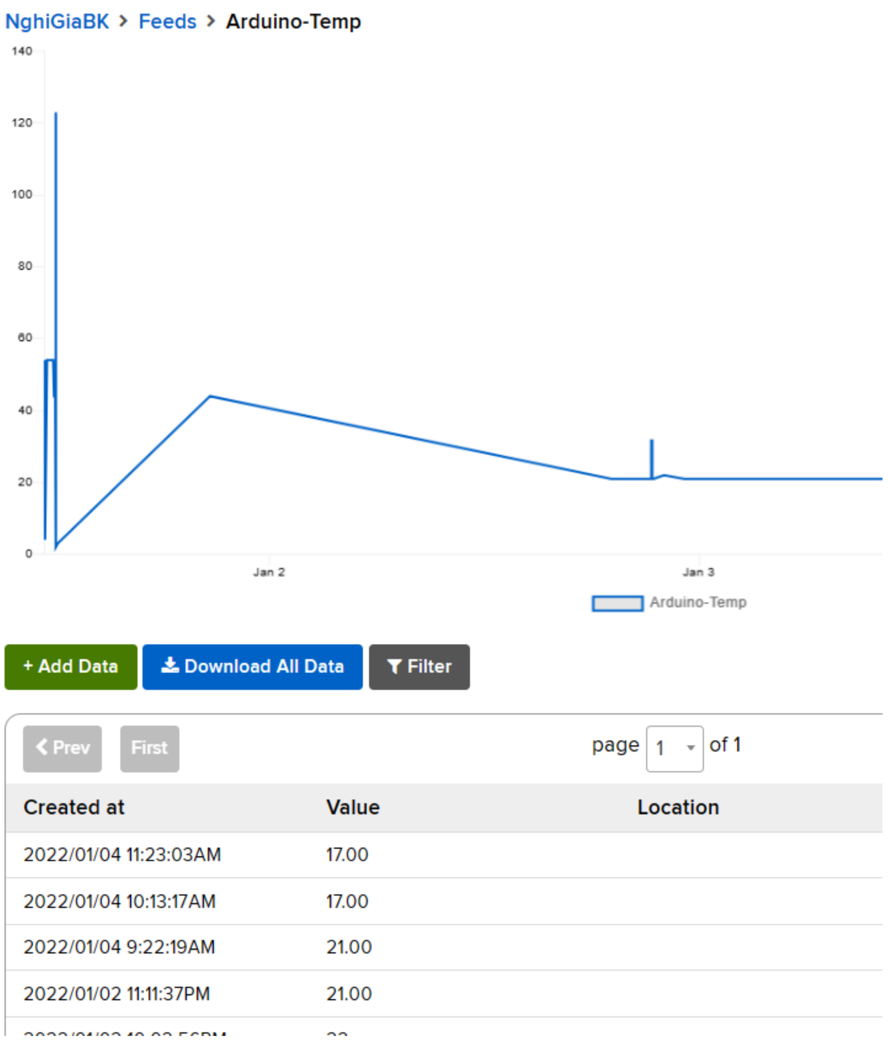
After the sensory data is received by the gateway, the source code to upload the data to MQTT server is presented here. In the case the data is enclosed a format (json string), please present your implemetation for this part.



```
AIO_FEED_ID = ["Arduino-Temp","Arduino-Humi","Arduino-Led","Arduino-Pir","Arduino-Pir","Arduino-Gas"]
AIO_USERNAME = "NghiGiaBK"
AIO_KEY ="aio_ZDWp35cSDtVefLs4hA2LBTFCeiiD"
def connected ( client ) :
    print ("Ket noi thanh cong ...")
    for feed in AIO_FEED_ID:
        client.subscribe(feed)

def subscribe ( client , userdata , mid , granted_qos ) :
    print ("Subcribe thanh cong ...")

def disconnected ( client ) :
    print (" Ngat ket noi ...")
    sys . exit (1)

def message ( client , feed_id , payload ):
    print ("Nhan du lieu : " + payload +" from feed "+ feed_id)
    ser.write (( str( payload ) ) . encode () )
client = MQTTClient(AIO_USERNAME, AIO_KEY )
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.on_subscribe = subscribe
client.connect ()
client.loop_background()
```

Figure 35.

First, if we need to upload some things to Adafruit server we need to create a Adafruit account, then receive AIO-Keys, then in gateway we will subscribe the account which we already create with the funtion ("connected", and "subscribe") Second, for each data we need a feed to store those data (like a box), here we have 6 different data so we will create 6 feeds for them.



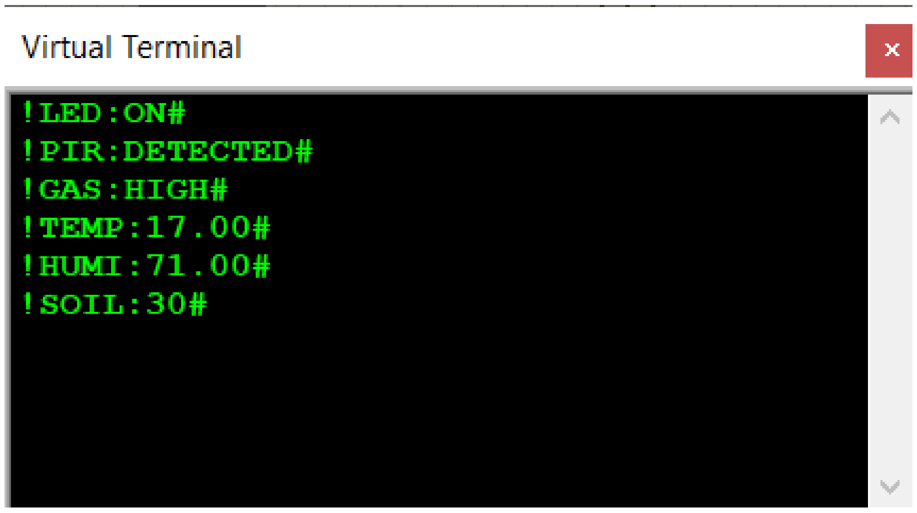
Created at	Value	Location
2022/01/04 11:23:03AM	17.00	
2022/01/04 10:13:17AM	17.00	
2022/01/04 9:22:19AM	21.00	
2022/01/02 11:11:37PM	21.00	

Figure 36.Example of a feed

Then, we need to read the data (funtion "message") which transfer from the server (feed or dashboard), and then print it to the terminal of python to check and then Serial write to the schematic of Proteus.
The ("ser.write ((str(payload)).encode())") will read the data (payload) and then send it to the serial connection which we already create.

8.3 Data received process

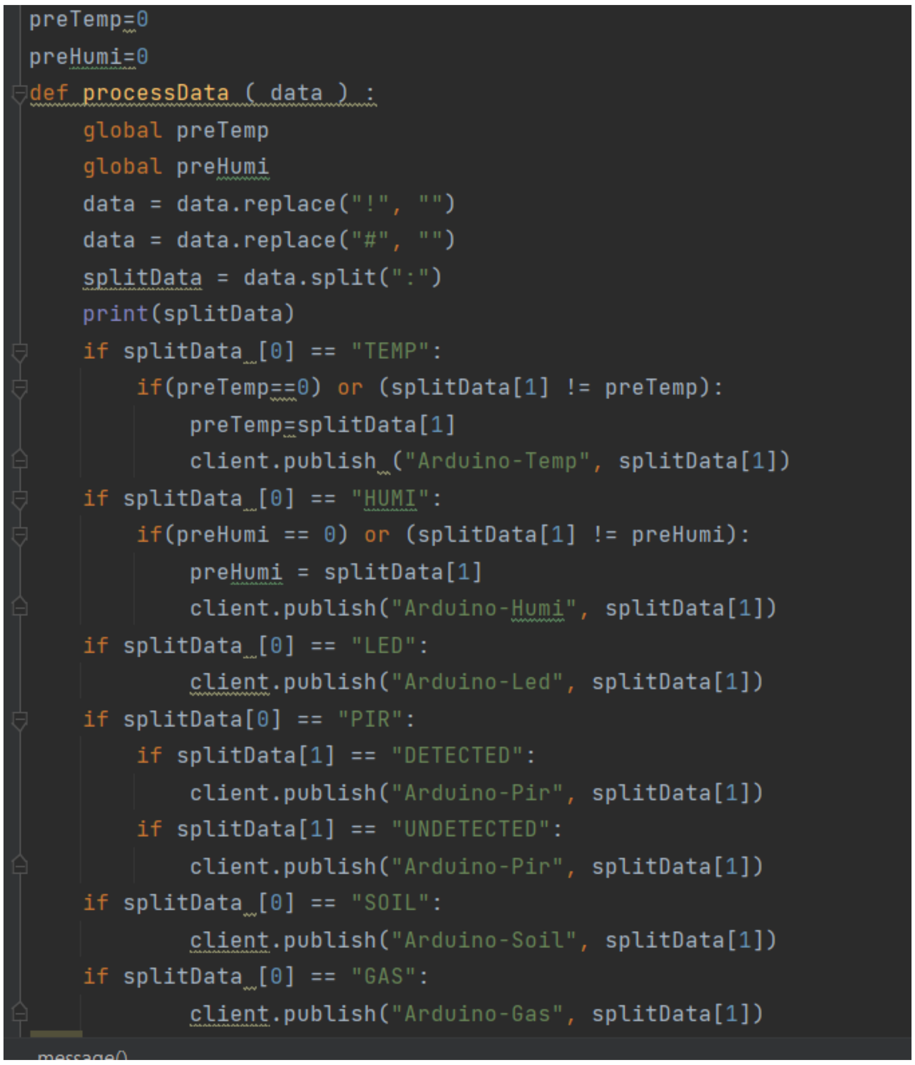
The format of frame data is : !NAME:VALUE



```
!LED:ON#
!PIR:DETECTED#
!GAS:HIGH#
!TEMP:17.00#
!HUMI:71.00#
!SOIL:30#
```

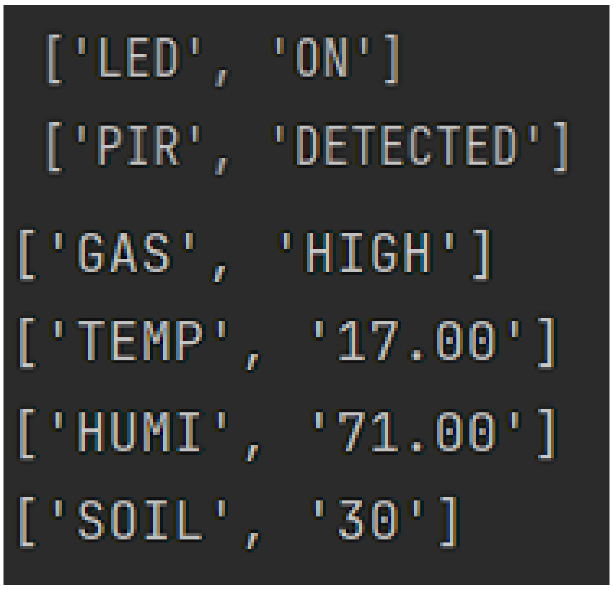
Figure 37. Virtual Terminal

Because the frame data is following a format, so when we receive a data, we need to split it to the Name and Value, the "Name" will be corresponding to the name of the data and also the feeds, it will publish the value to that feed



```
preTemp=0
preHumi=0
def processData ( data ) :
    global preTemp
    global preHumi
    data = data.replace("!", "")
    data = data.replace("#", "")
    splitData = data.split(":")
    print(splitData)
    if splitData [0] == "TEMP":
        if(preTemp==0) or (splitData[1] != preTemp):
            preTemp=splitData[1]
            client.publish ("Arduino-Temp", splitData[1])
    if splitData [0] == "HUMI":
        if(preHumi == 0) or (splitData[1] != preHumi):
            preHumi = splitData[1]
            client.publish("Arduino-Humi", splitData[1])
    if splitData [0] == "LED":
            client.publish("Arduino-Led", splitData[1])
    if splitData[0] == "PIR":
        if splitData[1] == "DETECTED":
            client.publish("Arduino-Pir", splitData[1])
        if splitData[1] == "UNDETECTED":
            client.publish("Arduino-Pir", splitData[1])
    if splitData [0] == "SOIL":
            client.publish("Arduino-Soil", splitData[1])
    if splitData [0] == "GAS":
            client.publish("Arduino-Gas", splitData[1])
```

Figure 38.Receive data

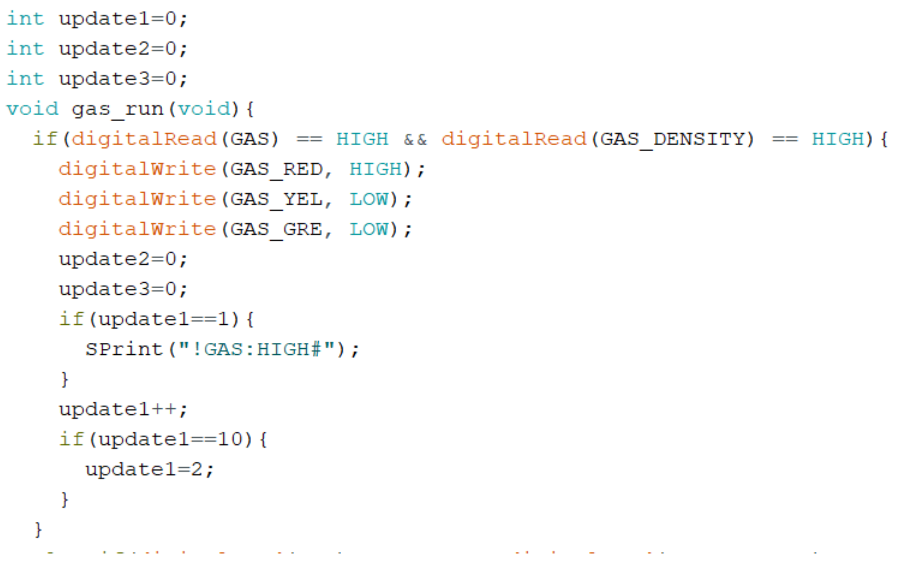


```
 ['LED', 'ON']
 ['PIR', 'DETECTED']
['GAS', 'HIGH']
['TEMP', '17.00']
['HUMI', '71.00']
['SOIL', '30']
```

Figure 39.Data after split

Because the Temp and Humidity will be update every 10 second, if we just upload to the server with the same date, it will overflow server memory and the CPU load, so we will create two integer "preTemp" and "preHumi" to check if the data is different or not between 10s duration.

Similarly, in Arduino, if we change some things to get new data, we must create a lock (just like mutex-lock in OS) to avoid the race condition (other parameters publish data) and it only publish one time until we change new conditions or new values.



```
int update1=0;
int update2=0;
int update3=0;
void gas_run(void){
  if(digitalRead(GAS) == HIGH && digitalRead(GAS_DENSITY) == HIGH){
    digitalWrite(GAS_RED, HIGH);
    digitalWrite(GAS_YEL, LOW);
    digitalWrite(GAS_GRE, LOW);
    update2=0;
    update3=0;
    if(update1==1){
      SPrint("!GAS:HIGH#");
    }
    update1++;
    if(update1==10){
      update1=2;
    }
  }
```

Figure 40.Avoid race condition in Arduino

9 Conclusion

9.1 Source code

https://drive.google.com/drive/folders/1mbsSoY6dhdBkLXItWRFM-6DNuYZ6jt6d?usp=sharing

References

[1] https://www.arduino.cc/en/

[2] https://www.electronicshub.org/arduino-introduction/

[3] https://learn.sparkfun.com/tutorials/what-is-an-arduino/all

[4] https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-uno.html

[5] https://www.theengineeringprojects.com/2020/01/introduction-to-proteus.html

[6] https://www.theengineeringknowledge.com/introduction-to-proteus/

[7] https://www.arduino.cc/reference/en/libraries/

[8] https://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use

[9] https://www.youtube.com/watch?v=2kr5A350H7E

[10] https://www.instructables.com/Arduino-Timer-Interrupts/

[11] https://en.wikipedia.org/wiki/Gas_detector

[12] https://en.wikipedia.org/wiki/Soil_moisture_sensor

[13] https://en.wikipedia.org/wiki/Passive_infrared_sensor

[14] https://create.arduino.cc/projecthub/ansh2919/

[15] https://www.theengineeringprojects.com/2013/05/how-to-use-virtual-terminal-in-proteus