LAB : EC Design Problem - Vertical Smart Farm

Embedded Controller Lab Report

Date : 2023. 12. 19

Author : 21900427 양두원

Partner : 22100224 김하은

Introduction

In this lab, We create a innovative smart farm that controls the temperature and humidity of multiple layers of substrates through vertical pulley movement using a single DC motor. In this project, as it is a prototype, it has been designed to manipulate only one layer. This vertical smart farm utilizes a Bluetooth module (HC-06) for wireless communication, adopting the USART1 method. It is equipped with a light sensor to activate Plant LEDs when there is no external light. The system also adjusts a fan based on temperature and humidity levels, and it has a mechanism that vertically moves in a pulley style to ensure the roots can maintain moisture by reaching the water tank. This vertical smart farm is designed with three modes: a Service Mode for changing the water tank, a Manual Mode for individual control of each sensor, and an Auto Mode that automatically regulates the plant's temperature, humidity, and light intensity.

Overview

Requirement

You must use at least 3 sensors: including 1 analog sensor and 1 digital sensor.

- You can choose sensors that were not used in the course LABs.
- You will get higher scores if you use more sensors.
- You must use at least 2 actuators.
- You must use at least 1 wireless communication.
- You must use at least 1 timer interrupt IRQ.
- You will get a higher score if you use more numbers of MCUs, sensors, and actuators.

Hardware

Item	Model/Description	Qty
MCU	NUCLEO -F411RE	1
MCU	Arduino-uno	1
Analog Sensor	DHT11 Temperature-Humidity	2

Item	Model/Description	Qty
Analog Sensor	Photoresistor	2
Digital Sensor	Button	1
Actuator	DC motor	2
Actuator	Servo motor	1
Display	Computer Monitor	1
Communication	Bluetooth (HC-06)	1
Others	LED	3
	Resistor	3
	Plant LED	1
	Mini Breadboard	4

Software

- Keil uVision, CMSIS, EC_HAL library
- Arduino, DHT11 library

Problem

Problem Description

Mode 1 : Service mode

We can notice service mode with the yellow LED.

The service mode is a mode where a person can come in for maintenance. It is activated with an odd number of button press. Both operations based on sensor values and operations due to command inputs come to a halt. Upon pressing the button, the plant plate is immediately raised, allowing the replacement of the water container beneath the plant for maintenance purposes. When in service mode, pressing the button once more causes the previously raised plant plate to return to its original position, and the service mode is terminated. Then, the plant LED turns on. At this point, you can choose between automatic mode and manual mode.

Mode 2 : Manual mode

If 'M' or 'm' is input via Blue-tooth, manual mode is initiated. Manual mode allows remote control even when there is no one present in the area.

- 'U' : Elevator goes up which means lift up the plant plate for 1.3 seconds.
- 'D' : Elevator goes down which means let down the plant plate for 1.3 seconds.
- 'R' : Turn off the Plant LED
- 'L' : Turn on the Plant LED

- 'F' : Operate the fan
- 'N' : Stop operating the fan
- 'O' : Open the ceiling
- 'C' : Close the ceiling

Mode 3 : Automatic mode

Automatic mode operates automatically using sensors to assess the environmental conditions and create an optimal environment for the plant.

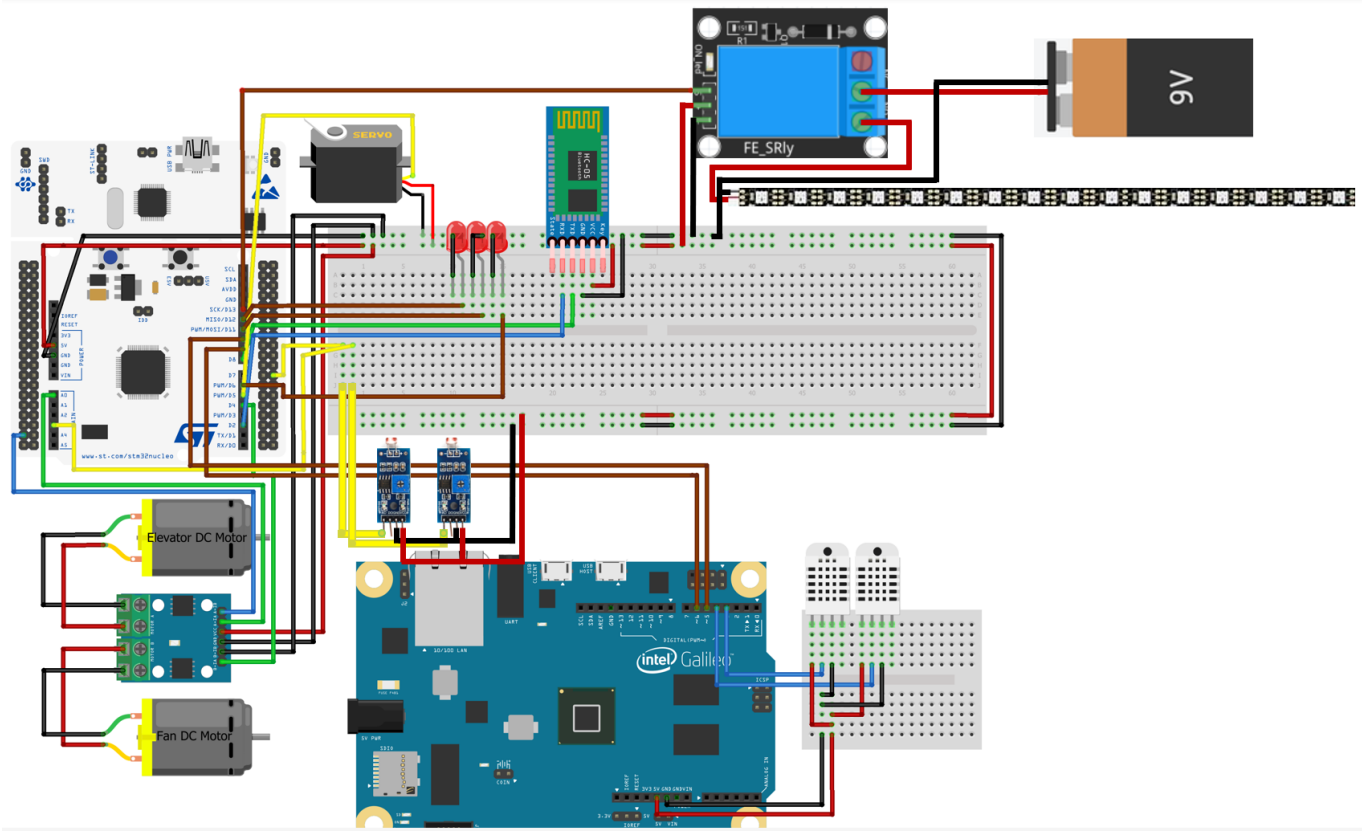
The humidity sensor measures the moisture content in the soil connected to the fabric. When the humidity drops below 50%, the plant plate goes down into the water below to immerse the soil and then raised. To prevent the plant tips from burning due to light, a temperature sensor is attached at the highest position when the plant is at its peak. When the temperature rises above 20 degrees Celsius, the fan is activated to lower the internal temperature. If it falls below 20 degrees Celsius, the fan operation is halted. Additionally, using a light sensor on the ceiling, when the sunlight is absent, the ceiling is closed, the plant LED is turned on. When sunlight is available, the ceiling is opened, and the plant LED is turned off.

MCU Configuration

Type	Port - Pin	Configuration
System Clock		PLL 84MHz
Digital Out: LED	PA6, PA7, PB10	Pull-Up, Medium
Digital Out: Plant LED	PA5	Pull-Up, Medium
Digital Input: Button	PC13	Pull-Up
Digital Out: Direction Pin (Elevator)	PC2	Push-Pull
PWM Pin (Elevator)	TIM2, PA0	AF, Push-Pull, Pull-Up, Fast, 1 [kHz]
Digital Out: Direction Pin (Fan)	Ground	
PWM Pin (Fan)	TIM2, PB5	AF, Push-Pull, Pull-Up, Fast, 1 [kHz]
Timer Interrupt	TIM5	1msec, Timer Interrupt of 1sec

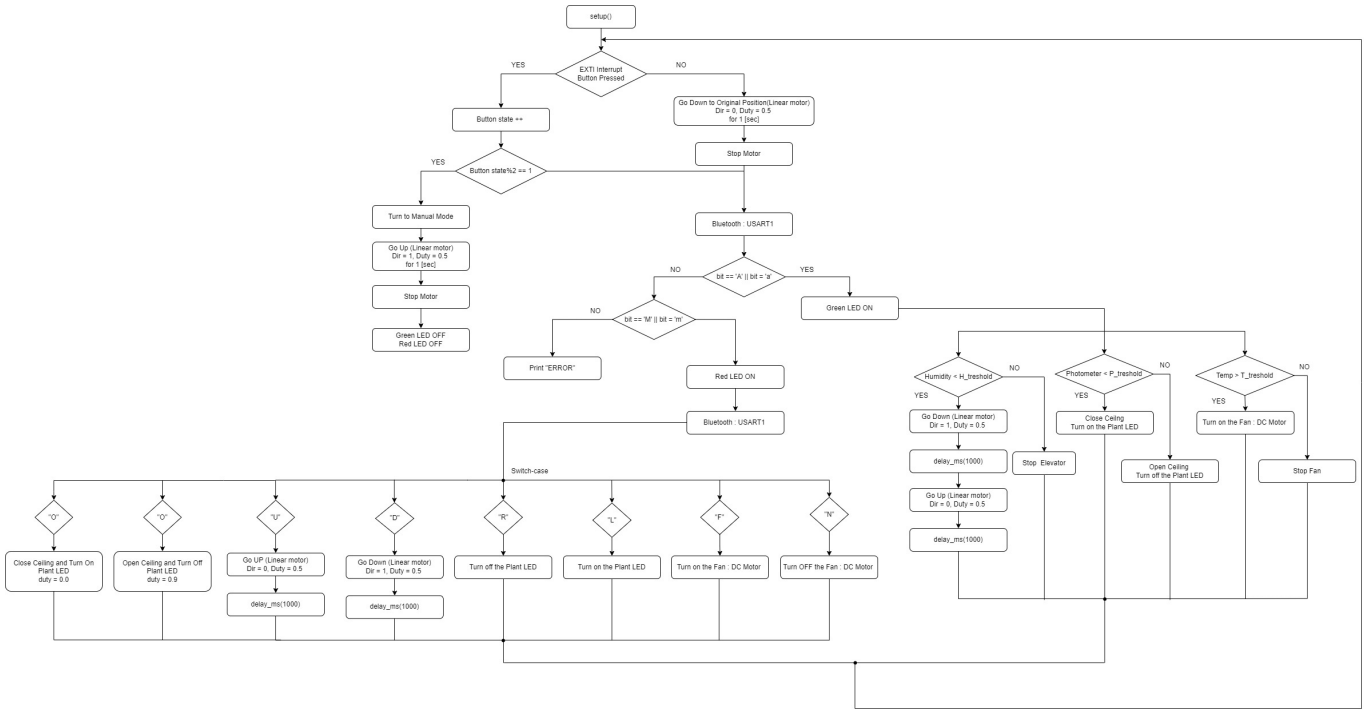
Type	Port - Pin	Configuration
USART : Bluetooth	TX : PA9, RX : PA10	Baud rate : 9600
PWM Pin (Servo motor)	TIM3, PB4	AF, Push-Pull, Pull-UDIp, Fast, PWM period 20[msec]
ADC (Photoresistor)	PB0: ADC1_CH8 (1st channel) PB1:ADC1_CH9 (2nd channel)	ADC Clock Prescaler /8 12-bit resolution, right alignment Continuous Conversion mode Scan mode: Two channels in regular group External Trigger (Timer3 Trigger) @ 1kHz Trigger Detection on Rising Edge
Digital Input (Temp flag)	PC7	
Digital Input (Humid flag)	PB6	

Circuit Diagram



Algorithm

Logic Design Flow Chart



Code

- Define

  Define the pins using the sensor's name.

```
#include "ecSTM32F411.h"
#define END_CHAR  13
#define MAX_BUF  100

// Motor for elevator
#define PWM_PIN PA_0
#define DIR_PIN 2

// Motor for fan
#define FAN_PWM_PIN PA_1

// LED for Mode
#define RED_LED    8
#define GRN_LED    6
#define PLT_LED   10 // PB10
#define YLW_LED    5 // PA5

// Bluetooth

// Temp and Humi
#define TEMP_PIN 7 // GPIOC
#define HUMI_PIN 6 // GPIOB

// Servo motor for window
#define SERVO_PWM_PIN PB_4
```

- Parameter

  Declared variables for each sensor to facilitate their use.

```
//--------------------PWM and Button-----------------//
static volatile uint32_t pwm_period = 1;
static volatile float    duty = 0;
static volatile float    FAN_duty = 1.0;
static volatile uint32_t Button = 0;
static volatile uint32_t Button_state = 0;
static volatile uint32_t dir = 0;
static volatile uint8_t  Mode = 0;
static volatile uint8_t  cmd = 0;
static volatile uint32_t servo_pwm_period = 20; // 20 [msec]

//-------------------USART1 Parameter----------------//
static volatile uint8_t bReceived = 0;
static volatile uint8_t BT_Data   = 0;

static volatile uint8_t buffer[MAX_BUF]    = {0, };
static volatile uint8_t BT_string[MAX_BUF] = {0, };

static volatile uint8_t  idx = 0;

static volatile char s1[40];

//-------------------USART2 Parameter----------------//
static volatile uint8_t cReceived = 0;
static volatile uint8_t CT_Data   = 0;

static volatile uint8_t buf[MAX_BUF]    = {0, };
static volatile uint8_t CT_string[MAX_BUF] = {0, };

static volatile uint8_t  i = 0;


//-------------Timer Interrupt parameter------------//
static volatile uint16_t A_cnt = 0;
static volatile uint16_t M_cnt = 0;
static volatile uint16_t count = 0;

//-------------------Light parameter----------------//
static volatile uint32_t Light_value  = 0;
static volatile uint32_t value1 = 0;
static volatile uint32_t value2 = 0;
static volatile uint32_t flag   = 0;
static volatile int sequence[2] = {8,9};


//---------Temperature and Humidity parameter-------//
static volatile unsigned int Temp_value = 0;
static volatile unsigned int Humi_value = 1;
```

- Main

  To ensure proper Blue-tooth connection at the start of the program, print 'Connected.' Additionally, print button instructions for user guidance in using the system. At the initial startup of the system, turn on the plant LED. To quickly update and reflect the values received from the sensors, we did not include other commands in the While loop.

```
int main(void)
{
    // Initialiization ------------------------------------------------------
---
    setup();
    USART1_write("Connected\r\n", 11);
    USART1_write("Press Button Once : Service Version\r\n", 37);
    USART1_write("Press Button Twice : Operating Version\r\n",40);
    GPIO_write(GPIOB, PLT_LED, HIGH);

    // Inifinite Loop -------------------------------------------------------
----
    while(1){}
}
```

- Initialization

  Call MCU_init function for the setup.

```
void setup(void)
{
    MCU_init();
}
```

  - MCU Initialization

    Initialize the timer and systick as 84[MHz] and 1[kHz]. Initialize the bluetooth pin by UART1_init and select the baud-rate. The plant LED and all the LEDs used are initialized as digital output. Button is initialized as input. We called initialization functions for motor, photo-resistor, and DHT11.

```
 void MCU_init(void){
    RCC_PLL_init();
    SysTick_init(1);        // priority = 8

    UART2_init();
    UART2_baud(BAUD_9600);

    UART1_init();
    UART1_baud(BAUD_9600);
```

```
    // Plant LED
    GPIO_init(GPIOB, PLT_LED, OUTPUT);
    GPIO_pupd(GPIOB, PLT_LED, EC_PU);
    GPIO_ospeed(GPIOB, PLT_LED, EC_MEDIUM);

    // INPUT : Button pin Initialization
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 9);  // priority = 9

    // Red LED
    GPIO_init(GPIOA, RED_LED, OUTPUT);
    GPIO_pupd(GPIOA, RED_LED, EC_PU);
    GPIO_ospeed(GPIOA, RED_LED, EC_MEDIUM);

    // Green LED
    GPIO_init(GPIOA, GRN_LED, OUTPUT);
    GPIO_pupd(GPIOA, GRN_LED, EC_PU);
    GPIO_ospeed(GPIOA, GRN_LED, EC_MEDIUM);

    // Yellow LED
    GPIO_init(GPIOA, YLW_LED, OUTPUT);
    GPIO_pupd(GPIOA, YLW_LED, EC_PU);
    GPIO_ospeed(GPIOA, YLW_LED, EC_MEDIUM);

    MOTOR_init();

    Light_init();

    TIM_UI_init(TIM5, 1);

}
```

- Motor Initialization

Initialize the direction pin as a digital output for DC motor which used for elevator. Set the initial direction to down direction. Apply PWM with 1[kHz] to the PWM pin of elevator DC motor. For DC motor of fan, fixed the direction pin to the ground since it will turn in single way. Similar to elevator, apply PWM with 1[kHz] to the PWM pin of fan's DC motor. For the servo motor, apply PWM with 50[Hz]. Apply 0 duty PWM for elevator not to move and apply 1.0 duty to fan. Apply 0.025 for servo motor, so it stays in the initial point.

```
void MOTOR_init(void){
    // OUTPUT : Direction pin Initialization
  GPIO_init(GPIOC, DIR_PIN, OUTPUT);
  GPIO_otype(GPIOC, DIR_PIN, EC_PUSH_PULL);
  GPIO_write(GPIOC, DIR_PIN, LOW);              // DIR = 0 : DOWN / DIR
= 1 : UP

    // PWM for elevator motor
```

```
    PWM_init(PWM_PIN);
    PWM_period(PWM_PIN, pwm_period);   // 1 msec PWM period

    // Direction for fan <- Connected to the 5[V]

    // PWM for fan motor
    PWM_init(FAN_PWM_PIN);
    PWM_period(FAN_PWM_PIN, pwm_period);

    // PWM for Servo motor
    PWM_init(SERVO_PWM_PIN);
    PWM_period(SERVO_PWM_PIN, servo_pwm_period);

    duty = 0.0;
    FAN_duty = 1.0;
    PWM_duty(PWM_PIN, duty);
    PWM_duty(FAN_PWM_PIN, FAN_duty);
    PWM_duty(SERVO_PWM_PIN,(float)0.025);
}
```

- Photo-resistor Initialization

  Initialize photo-resistors which connected to PB0 and PB1. PB0 is a first channel and PB1 is a second channel.

```
void Light_init(void){
    ADC_init(PB_0);
    ADC_init(PB_1);

    ADC_sequence(sequence, 2);
}
```

- DHT11 Temperature-Humidity Initialization

  Initialize the pins for temperature and humidity as a digital input because it converted as a digital signal from Arduino.

```
void TempHumid_init(void){
    GPIO_init(GPIOC, TEMP_PIN, INPUT);
    GPIO_init(GPIOB, HUMI_PIN, INPUT);
}
```

- Functions

  - EXTI15_10_IRQHandler

    External interrupt happens when the button pressed. When the button pressed, we gave 500 [ms] delay to prevent multiple recognitions from a single button press. Each time the button is

pressed, the value of the Button variable increases by 1. Then, functions that related to the button is implemented.

```
//EXTI for Pin 13
void EXTI15_10_IRQHandler(void) {
    if ((EXTI->PR & EXTI_PR_PR13) == EXTI_PR_PR13) {
        delay_ms(500);
        Button++;
        Button_pressed();
        Button_LED();
        clear_pending_EXTI(BUTTON_PIN);
    }
}
```

- Button_pressed

This function is responsible for selecting and deselecting the service mode. Store the remainder of the button variable divided by 2 in variable 'Button_state'.

If the Button_state is 1, it is a service mode. Print the "Service mode" on the monitor and lift up the plant plate by giving DC motor direction 1 and PWM duty 0.2 for 2 seconds. After 2 seconds, stop the motor. If the Button_stat is 0, print "Auto or Manual (A or M)" and let down the plant plate by giving PWM duty 0.8 for 2 seconds as well. Since the direction already set to 0 in Motor_Stop function, there is no need to set it to 0 again.

```
// Service Ver. or Mode Ver.
void Button_pressed(void){
    Button_state = Button%2 ;

    if (Button_state == 1){  // Service Version
        USART1_write("Service Mode.\r\n",15);
        GPIO_write(GPIOC, DIR_PIN, Button_state);   // Direction : UP
        PWM_duty(PWM_PIN, (float)0.2);
        delay_ms(2000);
        Motor_Stop(PWM_PIN, DIR_PIN);

    } else if (Button_state == 0){  // Mode Version
        USART1_write("Auto or Manual. (A or M)\r\n",26);
        PWM_duty(PWM_PIN, (float)0.8); // Direction : DOWN
        delay_ms(2000);
        Motor_Stop(PWM_PIN, DIR_PIN);
    }
}
```

- Button_LED

If the Button_state is 1 which means service mode, turn on only the yellow LED. Otherwise, if the Button_state is 0 which means not a service mode, turn all of the LEDs except yellow one.

```
void Button_LED(void){
    if (Button_state == 1){  // Service Version
        GPIO_write(GPIOA, RED_LED, LOW);   // LED OFF
        GPIO_write(GPIOA, GRN_LED, LOW);
        GPIO_write(GPIOA, YLW_LED, HIGH);
        GPIO_write(GPIOB, PLT_LED, LOW);

    }else if (Button_state == 0){  // Mode Version
        GPIO_write(GPIOA, RED_LED, HIGH);   // LED ON
        GPIO_write(GPIOA, GRN_LED, HIGH);       // LED OFF
        GPIO_write(GPIOA, YLW_LED, LOW);
        GPIO_write(GPIOB, PLT_LED, HIGH);
    }
}
```

- Motor_Stop

  This function is to stop the DC motor for elevator. Set the direction of the motor as 0 and give 0 PWM duty ratio, so the DC motor stops.

```
// Stop the Motor
void Motor_Stop(PinName_t PWM_pinName, int DIR_pin){
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(PWM_pinName, &port, &pin);

    GPIO_write(GPIOC, DIR_pin, LOW);
    PWM_duty(PWM_pinName, (float) 0.0);
}
```

- Mode_select

  This function is to select the mode and save to the variable 'Mode' that will used in other functions. If the blue-tooth input is a 'M' or 'm', print out "Manual Mode" and save 'M' to variable 'Mode'. If it is a 'A' or 'a', print out "Automatic mode" and save 'A' to variable 'Mode'.

```
// Select the Mode
void Mode_select(uint8_t MOD){
    if ((MOD == 'M') ||(MOD == 'm')){
        USART1_write("Manual Mode\r\n", 13);
        Mode = 'M';
    }else if((MOD == 'A') ||(MOD == 'a')){
        USART1_write("Automatic Mode\r\n", 16);
        Mode = 'A';
    }
}
```

- Manual_cmd

  This function is used when in manual mode. It is executed based on the input command and print out the command.

  'U' : Elevator goes up which means lift up the plant plate for 1.3 seconds.

  'D' : Elevator goes down which means let down the plant plate for 1.3 seconds.

  'R' : Turn off the Plant LED

  'L' : Turn on the Plant LED

  'F' : Operate the fan

  'N' : Stop operating the fan

  'O' : Open the ceiling

  'C' : Close the ceiling

```
void Manual_cmd(uint8_t command){
    switch(command){
        case 'U' :  // Elevator Go Up
            {
                USART1_write("Take from the water.\r\n",22);
                dir = 1;
                duty = (float) 0.2;
                GPIO_write(GPIOC, DIR_PIN, dir);   // Direction : UP
                PWM_duty(PWM_PIN, duty);
                delay_ms(1300);
                Motor_Stop(PWM_PIN, DIR_PIN);

                break;
            }
        case 'D' :  // Elevator Go Down
            {
                USART1_write("Put under water.\r\n", 18);
                dir = 0;
                duty = (float) 0.8;
                GPIO_write(GPIOC, DIR_PIN, dir);   // Direction : DOWN
                PWM_duty(PWM_PIN, duty);
                delay_ms(1300);
                Motor_Stop(PWM_PIN, DIR_PIN);

                break;
            }
        case 'R' :  // Turn Off the Plant LED
            {
                USART1_write("Plant LED OFF.\r\n",16);
                GPIO_write(GPIOB, PLT_LED, LOW) ;

                break;
```

```
            }
        case 'L' :  // Turn On the Plant LED
            {
                USART1_write("Plant LED ON.\r\n",15);
                GPIO_write(GPIOB, PLT_LED, HIGH);

                break;
            }
        case 'F' :  // Operate the Fan
            {
                USART1_write("Fan ON.\r\n",9);
                PWM_duty(FAN_PWM_PIN, (float)0.0);
                break;
            }
        case 'N' :  // Pause the Fan
            {
                USART1_write("Fan OFF.\r\n",10);
                PWM_duty(FAN_PWM_PIN, (float)0.8);
                break;
            }
        case 'O' :
            {
                USART1_write("Ceiling Open.\r\n",15);

                PWM_duty(SERVO_PWM_PIN, (float)0.95);
                break;
            }
        case 'C' :
            {
                USART1_write("Ceiling Closed.\r\n",17);
                PWM_duty(SERVO_PWM_PIN, (float)0.0);
                break;
            }
    }
}
```

- Manual_mode

  For the manual mode, turn on the red LED and turn off the green LED and Yellow LED.

```
void Manual_mode(void){
    GPIO_write(GPIOA, RED_LED, HIGH);   // LED ON
    GPIO_write(GPIOA, GRN_LED, LOW);        // LED OFF
    GPIO_write(GPIOA, YLW_LED, LOW);
}
```

- Automatic_mode

  For the automatic mode, turn on the green LED and turn off the red LED and yellow LED.

If Light_value is above 1000, it indicates no incoming light. In this case, turn on the plant LED and close the ceiling. Conversely, if Light_value is 1000 or below, it indicates incoming light. In this case, turn off the plant LED and open the ceiling. If Temp_value is 1, it indicates a temperature of 20 degrees or higher. In this case, activate the fan. Conversely, if Temp_value is 0, it indicates a temperature of 20 degrees or lower. In this case, do not activate the fan.

```
void Automatic_mode(void){
    GPIO_write(GPIOA, RED_LED, LOW);    // LED OFF
    GPIO_write(GPIOA, GRN_LED, HIGH);       // LED ON
    GPIO_write(GPIOA, YLW_LED, LOW);

    // Light Sensor
    if (Light_value > 1000){
        USART1_write("The sun sets.\r\n",15);
        GPIO_write(GPIOB, PLT_LED, HIGH);  // Turn on the Plant Light
        PWM_duty(SERVO_PWM_PIN,(float)0.0);
    }else{
        USART1_write("The sun rises.\r\n",16);
        PWM_duty(FAN_PWM_PIN, (float)0.0);
        GPIO_write(GPIOB, PLT_LED, LOW);   // Turn off the Plant Light
        PWM_duty(SERVO_PWM_PIN,(float)0.95);
    }

    // Temperature
    if (Temp_value == 1){    // higher than 20 degree --> fan
        PWM_duty(FAN_PWM_PIN, (float)0.0);
        USART1_write("Turn on the Fan.\r\n",18);
    }else{
        PWM_duty(FAN_PWM_PIN, (float)0.8);
    }
    // Humidity
}
```

- Humid_check

If 'Humi_value' is 0, indicating humidity is 50% or lower, lower and then raise the elevator. This is for dipping the plant plate into the water. When 'Humi_value' is 1, indicating humidity is 50% or higher, there is no need to provide water, so it remains stationary.

```
void Humid_check(void){
    if (Humi_value == 0){    // Humidity is lower than 50 %  --> water
(elevator DC motor)

        USART1_write("Humidity is lower than 50% : Give water\r\n",41);
        // DOWN
        dir = 0;
        duty = (float) 0.8;
        GPIO_write(GPIOC, DIR_PIN, dir);   // Direction : DOWN
        PWM_duty(PWM_PIN, duty);
        delay_ms(1000);
```

```
            Motor_Stop(PWM_PIN, DIR_PIN);
            delay_ms(1000);
            // UP
            dir = 1;
            duty = (float) 0.2;
            GPIO_write(GPIOC, DIR_PIN, dir);    // Direction : UP
            PWM_duty(PWM_PIN, duty);
            delay_ms(1000);
            Motor_Stop(PWM_PIN, DIR_PIN);

        }else if (Humi_value == 1){    // Humidity is higher than 50% -->
    open window (Servo)
            Motor_Stop(PWM_PIN, DIR_PIN);
        }
    }
```

- TIM5_IRQHandler

To update and apply sensor values or execute input commands, a timer is used. The timer updates every 1 [ms].

When the 'Mode' is 'M', execute Manual_mode function every 1 second. When the 'Mode' is 'A', execute Automatic_mode function and Humid_check function every 1 second. Also, every 1 second, update the photo-resistors value by storing average of two photo-resistors' value to the 'Light_value'. Then, update the 'Temp_value' and 'Humi_value' as well by read the assigned pin value.

```
void TIM5_IRQHandler(void){
    if(is_UIF(TIM5)){           // Check UIF(update interrupt flag)

        // Manual Mode
        if (Mode == 'M'){
            M_cnt++;
            if (M_cnt > 1000){
                Manual_mode();
                M_cnt = 0;
            }
        }

        // Automation Mode
        if(Mode == 'A'){
            A_cnt++;
            if (A_cnt > 1000){
                Automatic_mode();
                Humid_check();
                A_cnt = 0;
            }
        }

        count++;
        if (count > 1000){
```

```
            Light_value = (value1 + value2) / 2 ;
            Temp_value = GPIO_read(GPIOC, TEMP_PIN);    // 1 or 0
            Humi_value = GPIO_read(GPIOB, HUMI_PIN);    // 1 or 0
            count = 0;
        }
    }
    clear_UIF(TIM5);        // Clear UI flag by writing 0
}
```

- USART1_IRQHandler

This function is to type to the blue-tooth. Store the character to the buffer until the Enter key pressed. When the Enter key pressed copy the buffer to the 'BT_string' the reset the buffer. The first character of the 'BT_string' stored as 'cmd'. The 'cmd' uses as the input for functions Mode_selection and Manual_cmd.

```
void USART1_IRQHandler(void){               // USART2 RX Interrupt :
Recommended
    if(is_USART1_RXNE()){
        BT_Data = USART1_read();        // RX from UART2 (PC)
        USART_write(USART1, &BT_Data, 1);

        // Creates a String from serial character receive
        if(BT_Data != END_CHAR && (idx < MAX_BUF)){
            buffer[idx] = BT_Data;
            idx++;

        }else if (BT_Data== END_CHAR) {
// if(Enter)
            USART_write(USART1, "\r\n", 2);
            bReceived = 1;
            memset(BT_string, 0, sizeof(char) * MAX_BUF);       //
reset PC_string;
            memcpy(BT_string, buffer, sizeof(char) * idx);      // copy
to PC_string;
            memset(buffer, 0, sizeof(char) * MAX_BUF);          //
reset buffer
            idx = 0;
            cmd = BT_string[0];

            Mode_select(cmd);
            Manual_cmd(cmd);

        }else{
//  if(idx >= MAX_BUF)
            idx = 0;
            memset(BT_string, 0, sizeof(char) * MAX_BUF);       //
reset PC_string;
            memset(buffer, 0, sizeof(char) * MAX_BUF);
// reset buffer
        }
```

```
    }
}
```

- ADC_IRQHandler

  This function is for read the value of photo-resistors. If the flag is '0', read the first photo-resistor's value. If the flag is '1', read the second photo-resistor's value. Toggle the flag every time finishing the sequence.

```
void ADC_IRQHandler(void){
    if(is_ADC_OVR())
        clear_ADC_OVR();

    if(is_ADC_EOC()){       // after finishing sequence
        if (flag==0)
            value1 = ADC_read();
        else if (flag==1)
            value2 = ADC_read();
        flag = !flag;       // flag toggle

    }
}
```

- Arduino : DHT11 Temperature Humidity

  Arduino is used for getting values of temperature and humidity from DHT11. Read both temperature and humidity from both of two DHT11 sensor. Then, uses the first sensor's temperature value and the second sensor's humidity value because of the difference in positions. To check the changes in values, print it out to Serial.

  If the 'temp' value exceeds 20, the output of the pin number 10 is '1'. Otherwise, the output of the pin number 10 is '0'. If 'humi' value exceed 50, the output of the pin number 11 is '1'. Otherwise, the output of the pin number 11 is '0'.

```
#include <DHT11.h>  //아두이노 온습도센서 DHT11을 사용하기위해 위에서 설치해두었
던 라이브러리를 불러옵니다.
#include <SoftwareSerial.h>                // Serial 통신을 하기 위해 선언

char buf1[30], buf2[30];
SoftwareSerial BTSerial(blueTx, blueRx);            // HC-06모듈 7=TXD ,
8=RXD 핀 선언
DHT11 sensor1(3); /*불러온 라이브러리 안에 몇번 PIN에서 데이터값이 나오는*/
DHT11 sensor2(4);
void setup() {
  Serial.begin(9600); /*온습도값을 PC모니터로 확인하기위해 시리얼 통신을*/
  BTSerial.begin(9600);                    // HC-06 모듈 통신 선언 (보드레이
트 9600)

  pinMode(10, OUTPUT);
```

```
  pinMode(11, OUTPUT);
}

void loop() {

  float temp, humi, temp1, temp2, humi1, humi2;
  int result = sensor1.read(humi, temp); /* DHT.h 함수안에 dht11이라는 메소드
를 사용해서*/ // red line

  temp1 = temp;
  humi1 = humi;

  result = sensor2.read(humi, temp); /* DHT.h 함수안에 dht11이라는 메소드를 사
용해서*/

  temp2 = temp;
  humi2 = humi;

  temp = temp1;
  humi = humi2;

  String msg3 = "Temp: ";
  msg3 += temp;
  msg3 += "[C]\t\n";

  String msg4 = "Humi: ";
  msg4 += humi;
  msg4 += "[%]\n";

  msg3.toCharArray(buf1, 30);
  msg4.toCharArray(buf2, 30);

  Serial.write(buf1);
  Serial.write(buf2);

  int temp_out, humi_out;
  if (temp > 20){
    temp_out = 1;
  }else{
    temp_out = 0;
  }

  if (humi > 50){
    humi_out = 1;
  }else{
    humi_out = 0;
  }

  digitalWrite(10, temp_out);
  digitalWrite(11, humi_out);

  delay(1000); /* 일반적인 딜레이 값이 아니라 DHT11에서 권장하는*/
}
```

Results and Demo

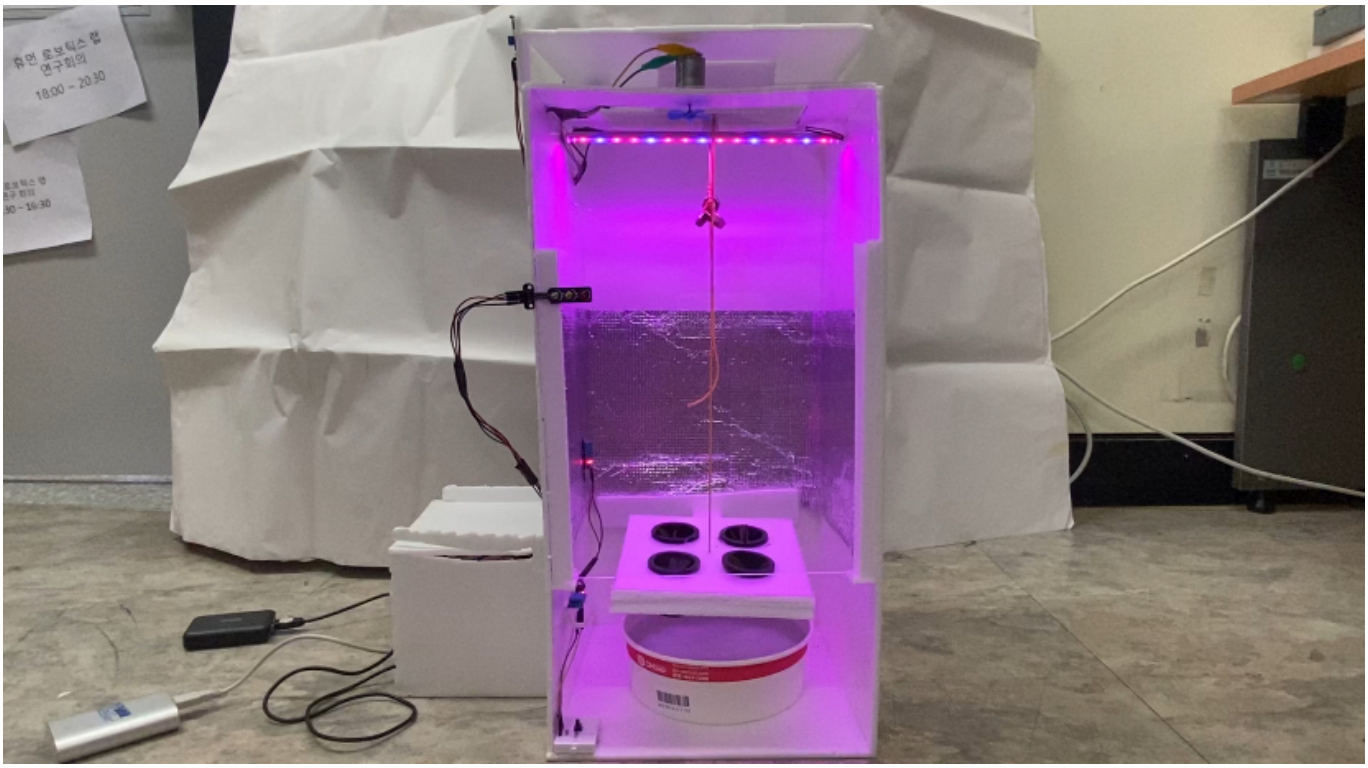
In this project, a vertical smart farm was designed and manufactured to operate in three modes - service mode, manual mode, and auto mode. RED, YELLOW, and GREEN LEDs indicate the current mode (RED for manual mode, YELLOW for service mode, and GREEN for auto mode).

As the button is pressed, it switches to service mode. In this mode, the plant LED turns off, and the YELLOW LED lights up. Additionally, the elevator operates upwards, allowing for the replacement of the plant water tank.

Pressing the button again enters a standby mode, where pressing the 'M' key switches to manual mode. In manual mode, the RED LED lights up, allowing for manual control of the elevator, Plane LED, fan, and ceiling.

Finally, in auto mode, the Green LED lights up. When it's bright around, the Plant LED is turned off and the ceiling opens to receive natural light. Conversely, when it gets dark, the ceiling closes, and the Plant LED turns on to maintain a constant light level. The heat generated by the LEDs can affect the plants, potentially causing tip burn. To prevent this, a temperature sensor is used to activate the fan when the temperature exceeds 25 degrees. Furthermore, if the humidity near the roots falls below 50%, the elevator descends to supply water to the plants.

Experiment images :



Demo Video : https://www.youtube.com/watch?v=ZO9ZyAyX1lM

Reference

RM0383 Reference manual

Troubleshooting

- If there is an interrupt within an interrupt, execution may not occur based on priority.
    - Besides external interrupt and timer interrupt, receiving sensor values is also interrupt-driven. So the priority should be made properly. The first priority is SysTick since we use delay functions which makes nothing execute for the time. Even if the data update is fast, it is ineffective if the sensor values are not available, so the sensors were given the second priority. To operate the system without errors, the timer was set as the third priority. Then, the last one is external interrupt which happens with button.
- The method of retrieving analog values from the DHT11 sensor was not working.
    - To retrieve the analog output value from the DHT11 sensor, a start signal is sent and 16 data values are sequentially received and processed. However, there are issues with values not being retrieved due to time delays caused by integrating with other sensors. To resolve it, the Arduino Uno and an open-source library were used to retrieve values. If specific temperature and humidity conditions are met, a flag is passed to the STM board.