

IoT Plant Waterer

E155 Final Project

December 1, 2020

Nico Perez Vergel and Naiche Whyte-Aguayo

1. Introduction

In the early 1980's, the advent of the internet ushered in a new level of connectivity possible for humanity. What began as a network of research computers on university campuses, now permeates many aspects of our lives. One such development is the Internet of Things, which involves household objects connecting to the network.

Inspired by this trend, we created an automatic plant watering system connected through the internet for our final project. The objective was to have a device that would not only automatically water a plant based on soil moisture levels, but also update a website with pertinent information such as the soil moisture level and the parameters it operates on.

2. Design

In order to accomplish these objectives, our design consisted of three separate modules: a sensor module to measure the soils moisture, a pump module for delivering water to the plant, and an IoT module to host a website for user inputs. Every module is controlled by the STM32F401RE ARM Cortex M4 microcontroller.

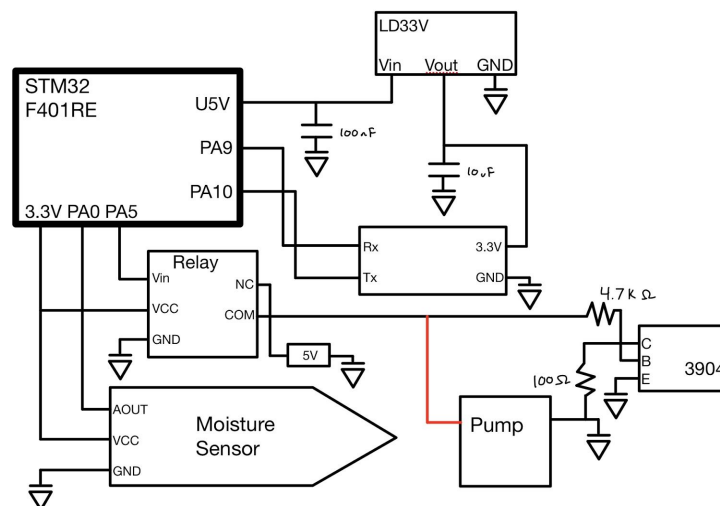


Figure 1. Overall Schematic

2.1 Sensor

The sensing is very simple, it consists solely of a capacitive moisture sensor. Capacitive moisture sensors are more expensive than resistive sensors but they last longer because of their resistance to corrosion. Resistive sensors pass DC current through two probes in the soil and measure the conductivity between them, while capacitive (also known as frequency domain) sensors measure the charge time of a capacitor using the soil as a dielectric medium.[1] The sensor contains an onboard voltage regulator, so it can be powered using either 3.3V or 5V. The output is an analog voltage that scales linearly with the moisture level.

In our circuit, the sensor is powered by 3.3V and the analog voltage is passed to the microcontroller's ADC through PA0. The analog voltage was calibrated by measuring the "moisture level" of the sensor when completely dry and then the level when submerged in water. Then, the moisture level was transformed so that a dry reading was 0% and fully wet was 100% moisture. Any ADC measurement between these two values produces a value between 0 and 100% that scales linearly.

2.2 Pump

The pump consists of a 5V DC motor pre-packaged into a plastic enclosure, with an inlet, outlet, and waterproofed power wires. Since the microcontroller contains multiple 5V pins, the primary concerns were that the motor would draw more current than the 25 mA a GPIO pin can safely output. While datasheets and online sources indicated that the current draw of the pump was a safe 21 mA, corroborated using a multimeter and 9V battery, we decided that this was close enough to the maximum to warrant extra design, especially when combined with the power loss due to supplying 3.3V instead of 5V. Instead, the motor is controlled using a 5V relay connected to a transistor current amplification circuit.

The relay works as a mechanical switch, physically connecting or disconnecting its internal circuit depending on the configuration of NC, NO pins and Vin pins. If COM and NO (normally open) pins are used, the switch is open and the circuit is active when the relay receives a high signal. The NC (normally closed) pin acts the same but in reverse. Our Vin pin is connected to PA5, which is set by the microcontroller logic if the moisture level is below a user-settable threshold. We decided to use NO so that the PA5 LED light would signal when the relay's circuit is closed. Initially, our NO and COM pins were wired to transfer 5V from the microcontroller to the transistor circuit when Vin was high, but this was changed to transfer power from a battery array to lower the overall current load of the microcontroller and improve ESP reliability.

The transistor current amplification is designed to have a voltage drop of 1 V between the collector and base pins. This helped us simplify current calculations based on the datasheet. We used a 4.7KOhm resistor on the base pin and a 100 Ohm resistor on the collector pin. The motor is connected to the collector pin, which is designed to provide 50 mA to the pump. This was the closest current measurement in the datasheet to the 21 mA our motor would draw, and we decided if it was too much for the pump we could add resistors to lower the current later in the design process.

Finally, the pump is submerged in a water container and provides water to the plant with a plastic tube connected to the outlet.



Figure 2. Pump and Sensor

2.3 ESP

The other modules were designed around the needs of the IoT module, which was the most sensitive. Initial tests, as well as experience with the ESP 8266 1-S chip from lab 7, showed that our biggest problem would be the reliability of the module. Research on forums showed that some of these problems came from the power needs of the ESP, which is reportedly very sensitive to drops in current and to voltage overloads. [3] Moreover, the ESP draws an average of 200-300 mA, with more potentially being drawn on startup. Both the soil

moisture sensor and relay use 3.3V to simplify the breadboard circuit because the ESP needs 3.3V. For example 0.1 V over the datasheet maximum of 3.6V is harmful to the chip. [4] The research on current stability was confirmed in initial tests, which showed that the ESP stopped working when the pump was added to the circuit, presumably because the increased current draw on the microcontroller negatively affected the current stability of the ESP. The first solution we tried was to bypass the microcontroller voltage regulator and use our own, which would hopefully isolate the ESP power from microcontroller current fluctuations. This was accomplished using the circuit shown in Figure X, which takes 5V from the microcontroller U5V pin and branches off from the microcontroller circuit before the onboard regulator. We fed the pins output to an LD33V linear voltage regulator circuit containing stabilizing capacitors. However, this was unable to power the ESP. The roundabout solution we found was to power the pump using batteries instead of using the microcontroller, freeing the microcontroller from one of its current supply duties. However, this isn't an optimal solution because the microcontroller is still reading analog voltages from the sensor and providing a 3.3V signal to the relay, as well as acting as the overall system master. Ideally, a separate power supply would be entirely dedicated to the ESP and nothing else.

This may be accomplished with a homemade makeshift power supply consisting of a 12V 1 A wall power supply being fed to an LC805CV 5V linear voltage regulator, which is then fed through the LD33V 3.3V linear regulator. This would be necessary because the absolute maximum power dissipation of the LD33V is 12W, the same amount supplied by the wall power supply.

3. Microcontroller Code

3.1 Sensor Code

Each of the modules are controlled by a single ARM Cortex M4 microcontroller. The sensing module's control is fairly simple. The Moisture.c library contains 3 functions that set the global variables Moisture Threshold, Water Time, and Probe Interval. There are 2 separate functions to water and stop watering, using TIM3 to control PA0, which in turn controls the relay Vin. Finally, we use a probe function to read and convert the moisture sensor analog voltage to a scaled percentage moisture. This function also attempts to control measurement error by throwing out analog measurements of 2048. During testing, this erroneous measurement occurred, so we filter it out. The ADC itself takes a single measurement when called by the probe() function.

3.2 ESP USART and DMA

The ESP codebase retained the starter code from E155 Lab 7, with the addition of DMA. We use DMA to communicate with the Tx buffer so that we could simply pass pointers of strings that contained AT commands or the html page. However, we found that Rx did not work as well with DMA since it requires that a string's length is known before communication. This was not possible to implement since ESP replies are not always deterministic. Therefore, to implement DMA for Rx, we would need to poll or add interrupts which defeats the purpose. Hence, we retained the ring buffer implementation from Lab 7's starter code.

We expected DMA to substantially help with ESP reliability. In regards to speed, DMA gave a noticeable improvement when loading websites and replying to other GET requests. However, the ESP is still sometimes unreliable because webpages can get stuck loading.

3.3 Baremetal Timer Master

We use timer interrupts to handle parameters like the frequency at which the moisture sensor is probed and how long the pump is on. The interrupt also handles all the logic to turn on the pump and turn it off.

3.4 Parsing

To parse get requests, we created a struct called GET_Request that stores information about the request. For example, the struct stores booleans that serve as flags to indicate if the GET request is for loading the website or updating parameters.

When the ESP receives a get request, the main loop sends the request into a function for parsing. The function then looks for "GET" in the string and sets the get flag in GET_Request. Then it looks for either "fav" or the param request and sets their respective flags. If it is a param request, it parses the line to extract the value for updating. Once it is done parsing, the main loop checks the flags to update the values or serve the webpage.

We also had to parse the webpage at startup. The webpage is initialized with "xxx" wherever there is a value that needs to be updated. Therefore, we parse the webpage to find every instance of "xxx" and set pointers in GET_Request to fill in those values.

3.5 Javascript

There are two javascript functions. The first handles button clicks whenever the user wants to update values. This function will extract the value and create a GET request with it. It then sends the request as a promise and notifies the user when the value is updated. An

example of the request would contain the line "WT: 5". The microprocessor would then find this line and set one of the values in the GET_Request to 5.

The second function sends a similar request but waits for a response to update the last measured value from the moisture sensor. Additionally we wanted this function to execute in the background repeatedly so we used setInterval() from the JavaScript library. setInterval() takes in a function and the amount of time in milliseconds between every execution. Therefore, we simply created the function inside of setInterval and set it to repeat every few seconds.

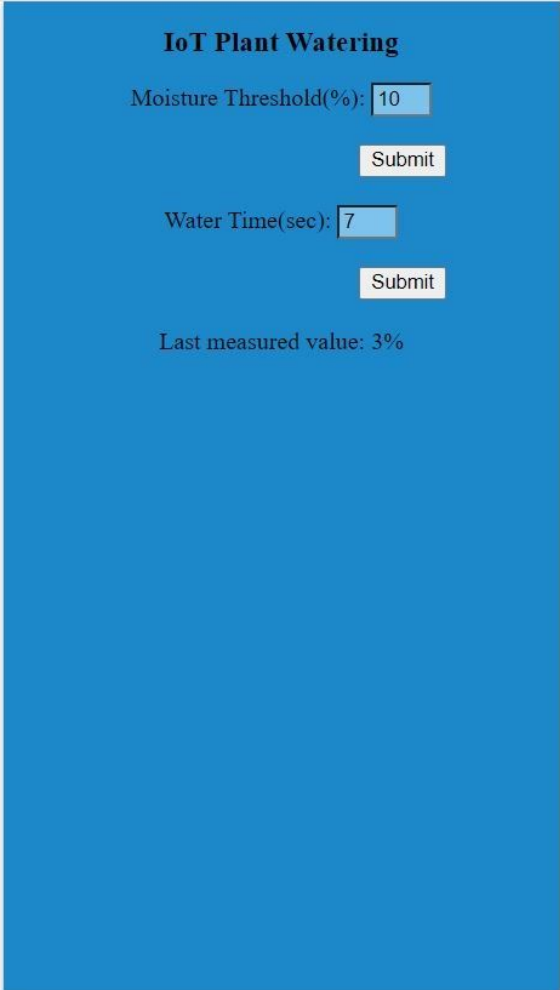
4. Results

This project was largely a success. The core functionality of an automatic plant waterer that also displays a webpage with user-settable parameters and information was achieved. There were several areas for improvement of the design. First, we clearly struggled with the reliability of the ESP and webpage setup. We tried to improve this with a multifaceted approach that addressed both hardware and software. First, we stabilized the power supply using a separate voltage regulator and a transistor current amplifier to take the load off the microcontroller. This was not fully successful, and would have benefitted from a separate power supply. The constraints of this semester made access to a separate power supply difficult (on campus we could've just used the power supplies readily available in the lab), but one idea we had was to cut open a wall power supply and pass it through two voltage regulators to step down to the correct voltage. Software wise, we changed the firmware instructions to utilize the buffer and implemented DMA transfer of instructions directly from memory. We hoped this would speed up the communication and lead to less "busy p" hangs. Aside from speed, we also changed the webpage updating design from a continuous transfer of HTML data in a while loop, to loading the entire webpage on startup and then using Javascript functions to handle get requests and parameter updating, saving the amount of characters transferred.

Secondarily, we also went through many designs for powering the pump. Once the webpage was functional, we discovered that occasionally the power draw of the motor would interfere with the ESP, so we tried passing a 9V battery through an L7805CV 5V linear regulator. However, this seemed to disrupt the timer interrupt functioning, probably due to current overloading. Eventually, we switched back to the microcontroller power source, which worked better after the DMA implementation.

Finally, we tried unsuccessfully to implement a "Time Elapsed Since Last Water" value to the website. The initial rudimentary code simply incremented a global variable in the timer interrupt if the pump was not watered and cleared it if it was, then multiplied this by the probe

time interval. This simple method gave the time elapsed in seconds, which was then added to the website with the same automatic update implementation the last measured value used. However, this caused the webpage to be extremely unreliable, so in the interest of preserving core functionality we removed it.



The image shows a screenshot of a web interface titled "IoT Plant Watering" on a blue background. It features two input fields for configuration: "Moisture Threshold(%)" with a value of 10 and "Water Time(sec)" with a value of 7. Each input field is followed by a "Submit" button. Below these fields, the text "Last measured value: 3%" is displayed.

5. References

¹ "The Soil Water Compendium." *Edaphic Scientific*,

² www.edaphic.com.au/soil-water-compendium/.

³ Walterscheidt, Udo. "Details." *Building a -Reliable- IoT Device Using the ESP8266*, 2 Mar.

⁴ 2015, hackaday.io/project/4493-building-a-reliable-iot-device-using-the-esp8266/details.

6. Code base:

https://github.com/nvergel/E155_IOT_Plant_Watering

7. Appendices

Code Appendices

Src Folder

Main.c

```
#include "STM32F401RE.h"
#include "main.h"
#include <string.h>
#include "UARTRingBuffer.h"
#include "Moisture_Sensor.h"

#define initial_probe_interval 15
//initial html data
uint8_t htmlPage[] =
"<!DOCTYPE html>\
<meta name=\"viewport\" content=\"width=device-width,\
initial-scale=1.0\">\
<title>IoT Plant Watering</title>\
<style> html{height: 100%;} body {text-align: center; margin-top: 0;\
height: 100%;}\
    input {width:1.5rem;}\
    div {background-color: #1c87c9; width: 50%; margin-left: 25%;\
height: 100%; top: 10%;\
        box-shadow: 0 1px 6px rgba(0, 0, 0, 0.12), 0 1px 4px rgba(0,\
0, 0, 0.24);}\
    button {margin-left: 10rem; width:revert;}\
    input {background-color: #7ec3ed; width: 2rem}\
    h3 {margin: 0; padding: 1rem}\
    p {display: inline}\
</style>\
<div>\
<h3>IoT Plant Watering</h3>\
    <label for=\"MT\">Moisture Threshold(%):</label>\
    <input type=\"number\" id=\"MT\" name=\"MT\" min=\"1\" max=\"255\"
value=xxx >\
    <br><br><button
onclick=\"updateData('MT')\">Submit</button><br><br>\
    <label for=\"WT\">Water Time(sec):</label>\
```

```

    <input type="number" id="WT" name="WT" min="1" max="255"
value=xxx >\
    <br><br><button onclick="updateData('WT')">Submit</button>\
    <br><br><p>Last measured value: <p id="LMV">xxx </p>%</p>\
</div>\
<script>\
function updateData(input) {\
    const params = new URLSearchParams();\
    params.append(input, document.getElementById(input).value);\
    fetch(new Request('', {headers: params})).then(window.alert("Value
successfully updated"));
}\
const updateParams = new URLSearchParams();\
updateParams.append("LMV", "");\
setInterval(function(){\
    fetch(new Request('', {headers: updateParams})).then(response =>
response.text())\
    .then(text => {if (text.length < 4)
document.getElementById("LMV").innerText = text});\
}, 10000);\
</script>\r\n";

```

```

/** Initialize the ESP and print out IP address to terminal
*/

```

```

void initESP8266(USART_TypeDef * ESP_USART, USART_TypeDef * TERM_USART){
    // Disable echo
    sendData("ATE1\r\n", 6, ESP_USART);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);

    // Enable multiple connections
    sendData("AT+CIPMUX=1\r\n", 13, ESP_USART);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);

    // Create TCP server on port 80
    sendData("AT+CIPSERVER=1,80\r\n", 19, ESP_USART);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);

    // Change to mode 3 (AP + station )
    sendData("AT+CWMODE=3\r\n", 13, ESP_USART);
}

```

```

delay_millis(DELAY_TIM, CMD_DELAY_MS);

// Connect WiFi
uint8_t connect_cmd[128] = "";
sprintf(connect_cmd, "AT+CWJAP=\"%s\", \"%s\"\r\n", SSID, PASSWORD);
uint16_t cmdLen = strlen(connect_cmd);
sendData(connect_cmd, cmdLen, ESP_USART);

// Wait for connection
delay_millis(DELAY_TIM, 10000);
sendData("AT+CIFSR\r\n", 10, ESP_USART);
}

/** Map USART1 IRQ handler to our custom ISR for ESP Rx
 */
void USART1_IRQHandler(){
    USART_TypeDef * ESP_USART = id2Port(ESP_USART_ID);
    usart_ISR(ESP_USART);
}

/** TIM3 handles probing moisture sensor and watering plant
 */
void TIM5_IRQHandler() {
    if (pumpOn) {
        stopWaterPlant();
    } else {
        probe();
    }
    TIM5->SR &= ~(0x1); // Clear UIF
}

uint8_t parseValue(uint8_t *buffer, uint32_t i){
    uint8_t value = 0;
    while (buffer[i] != '\r') {
        value = value*10 + buffer[i] - 48;
        ++i;
    }
    return value;
}

```

```

void parseRequest(uint8_t *buffer, GET_Request *get_request){
    uint32_t bufferlength = strlen(buffer);
    uint8_t char1 = buffer[0];
    uint8_t char2 = buffer[1];
    uint8_t char3 = buffer[2];
    uint32_t i = 3;
    // i always ahead of char3, buffer[i] = char4

    while (i < bufferlength-2) {
        if (char1 == 'G' && char2 == 'E' && char3 == 'T') {
            get_request->GET = 1;
            break;
        }
        char1 = char2;
        char2 = char3;
        char3 = buffer[i];
        ++i;
    }
    while (i < bufferlength-2 && get_request->GET) {
        if (char1 == 'f' && char2 == 'a' && char3 == 'v') {
            get_request->FAV = 1;
        }
        // Search for moisture threshold
        else if (char1 == 'M' && char2 == 'T' && char3 == ':') {
            get_request->MT = 1;
            get_request->MT_val = parseValue(buffer, i+1);
        }
        // Search for water time
        else if (char1 == 'W' && char2 == 'T' && char3 == ':') {
            get_request->WT = 1;
            get_request->WT_val = parseValue(buffer, i+1);
        }
        // Search for water time
        else if (char1 == 'L' && char2 == 'M' && char3 == 'V') {
            get_request->LMV = 1;
        }
        char1 = char2;
        char2 = char3;
        char3 = buffer[i];
        ++i;
    }
}

```

```

    }
}

void updateVal(uint8_t* htmlPos, uint8_t* val) {
    for (uint8_t i = 0; i < 5; ++i) {
        htmlPos[i] = val[i];
    }
}

int main(void) {
    // Configure flash latency and set clock to run at 84 MHz
    configureFlash();
    configureClock();

    // Enable GPIOA clock
    RCC->AHB1ENR.GPIOAEN = 1;
    //set PA5 "WATER_PUMP" to output mode
    pinMode(GPIOA, WATER_PUMP, GPIO_OUTPUT);

    // Initialize timer
    RCC->APB1ENR |= (1 << 0); // TIM2_EN
    RCC->APB1ENR |= (1 << 3); // TIM5_EN
    initTIM(DELAY_TIM);
    setTimer(TIM5, initial_probe_interval); // Probe every minute

    // Configure ESP and Terminal UARTs
    USART_TypeDef * ESP_USART = initUSART(ESP_USART_ID, 115200);
    USART_TypeDef * TERM_USART = initUSART(TERM_USART_ID, 115200);

    initADC();

    initDMA();

    // Enable interrupts globally
    __enable_irq();

    // Configure interrupt for TIM3, USART1 and USART2
    *NVIC_ISER1 |= (1 << 18);
    *NVIC_ISER1 |= (1 << 5);
    // *NVIC_ISER1 |= (1 << 6);

```

```

ESP_USART->CR1.RXNEIE = 1;
TIM5->DIER |= 1;

// Initialize ring buffer
init_ring_buffer();
flush_buffer();

setWaterTime(7); // Initial water time set to 7 secs, can be changed
through website
setProbeInterval(initial_probe_interval); // Set probe interval

// Initialize moisture threshold to 10%
setMoistureThreshold(10);

// Initialize ESP
delay_millis(DELAY_TIM, 1000);
initESP8266(ESP_USART, TERM_USART);
delay_millis(DELAY_TIM, 500);

// Set up temporary buffers for requests
uint8_t volatile http_request[BUFFER_SIZE] = "";
uint8_t volatile temp_str[BUFFER_SIZE] = "";

GET_Request get_request;
get_request.htmlLen = strlen(htmlPage) + 1;
uint8_t cmd[25] = "";
// Send HTML
sprintf(cmd, "AT+CIPSENBUF=0,%d\r\n", get_request.htmlLen);
uint16_t cmdLen = strlen(cmd);
uint8_t j = 0;
for (uint16_t i = 0; j < 4; ++i) {
    if (htmlPage[i] == 'x' && htmlPage[i+1] == 'x' && htmlPage[i+2] ==
'x') {
        switch (j) {
            case 0:
                get_request.ptrMT = htmlPage+i;
            case 1:
                get_request.ptrWT = htmlPage+i;
            case 2:
                get_request.ptrLMV = htmlPage+i;

```

```

        }
        ++j;
    }
}

uint8_t paramHolder[7] = "";

// Replace xxx with \"num\"
sprintf(paramHolder, \"%d\" ", moistureThreshold);
updateVal(get_request.ptrMT, paramHolder);

// Replace xxx with \"num\"
sprintf(paramHolder, \"%d\" ", WATER_TIME_SECONDS);
updateVal(get_request.ptrWT, paramHolder);

// Replace xxx with num
sprintf(paramHolder, \"%d\" ", moisture);
updateVal(get_request.ptrLMV, paramHolder);

printData("Ready");

while(1) {
    memset(http_request, 0, BUFFER_SIZE);

    // Clear temp_str buffer
    get_request.GET = 0;
    get_request.FAV = 0;
    get_request.MT = 0;
    get_request.WT = 0;
    get_request.LMV = 0;

    //if (lowMoisture) {
        //waterPlant();
    //}

    // Loop through and read any data available in the buffer
    if (is_data_available()){

        do{

            memset(temp_str, 0, BUFFER_SIZE);

```



```

        readString(ESP_USART, temp_str); // Read in available
bytes
        strcat(http_request, temp_str); // Append to current
http_request string
        delay_millis(DELAY_TIM, CMD_DELAY_MS); // Delay
    } while(is_data_available()); // Check for end of transaction

// Echo received string to the terminal
printData(http_request);

parseRequest(http_request, &get_request);
//get request execution
if ( get_request.MT) {
    setMoistureThreshold(get_request.MT_val);
    sprintf(paramHolder, "\"%d\" ", moistureThreshold);
    updateVal(get_request.ptrMT, paramHolder);
    sendData("AT+CIPCLOSE=0\r\n", 15, ESP_USART);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    get_request.MT = 0;
} else if ( get_request.WT) {
    setWaterTime(get_request.WT_val);
    sprintf(paramHolder, "\"%d\" ", WATER_TIME_SECONDS);
    updateVal(get_request.ptrWT, paramHolder);
    sendData("AT+CIPCLOSE=0\r\n", 15, ESP_USART);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    get_request.WT = 0;
} else if (get_request.LMV) {
    sprintf(paramHolder, "%d", moisture);
    uint16_t paramLen = strlen(paramHolder);

    uint8_t cmd1[25] = "";
    sprintf(cmd1, "AT+CIPSENDERBUF=0,%d\r\n", paramLen);
    uint16_t cmd1Len = strlen(cmd1);
    sendData(cmd1, cmd1Len, ESP_USART);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    sendData(paramHolder, paramLen, ESP_USART);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    sendData("AT+CIPCLOSE=0\r\n", 15, ESP_USART);
} else if (get_request.GET && !get_request.FAV &&
!get_request.WT && !get_request.MT) {

```



```

    TIM5->DIER &= ~1;
    setTimer(TIM5, WATER_TIME_SECONDS);
    TIM5->DIER |= 1;

    pumpOn = 1;
}

void stopWaterPlant() {
    // End plant watering
    digitalWrite(GPIOA, WATER_PUMP, 0);
    //turn off timer interrupt
    TIM5->DIER &= ~1;
    //set interval
    setTimer(TIM5, PROBE_INTERVAL);
    //turn on interrupt
    TIM5->DIER |= 1;
    //rest pumpOn
    pumpOn = 0;
}

//functions for setting global vars
void setMoistureThreshold(uint8_t moisture) {
    moistureThreshold = moisture;
}

void setWaterTime(uint8_t waterTimeSeconds) {
    WATER_TIME_SECONDS = waterTimeSeconds;
}

void setProbeInterval(uint8_t probeInterval) {
    PROBE_INTERVAL = probeInterval;
    setTimer(TIM5, PROBE_INTERVAL);
}

```

DMA Library

```

// Standard library includes.
#include <stdint.h>
#include "STM32F401RE_DMA.h"
#include "STM32F401RE_RCC.h"
#include "STM32F401RE_USART.h"

```

```

#include "STM32F401RE_TIM.h"

void setupTx(DMA_STREAM* DMAx_STREAMy, uint8_t PL) {
    DMAx_STREAMy->CR.EN = 0;

    // Channel 4
    DMAx_STREAMy->CR.CHSEL = 4;

    // Set byte size
    DMAx_STREAMy->CR.PSIZE = 0;
    DMAx_STREAMy->CR.MSIZE = 0;

    // memory to peripheral
    DMAx_STREAMy->CR.DIR = 1;
    DMAx_STREAMy->CR.CIRC = 0;

    // Memory pointer increment
    DMAx_STREAMy->CR.MINC = 1;

    // High priority level
    DMAx_STREAMy->CR.PL = PL;
}

void initDMA() {

    RCC->AHB1ENR.DMA1EN = 1;
    RCC->AHB1ENR.DMA2EN = 1;

    // Set USART to DMAT
    USART1->CR3 = (0b10 << 6);
    USART2->CR3 = (0b10 << 6);

    setupTx(DMA1_STREAM6, 1);
    setupTx(DMA2_STREAM7, 2);

    DMA1_STREAM6->PAR = (uint32_t) &(USART2->DR);
    DMA2_STREAM7->PAR = (uint32_t) &(USART1->DR);
}

```

```

void printData(uint8_t* str) {
    DMA1_STREAM6->CR.EN = 0;
    DMA1->HIFCR.CTCIF6 = 1;
    uint16_t strLen = strlen(str);
    DMA1_STREAM6->M0AR = str;
    DMA1_STREAM6->NDTR = strLen;
    DMA1_STREAM6->CR.EN = 1;
    while(!DMA1->HISR.TCIF6);
}

void sendData(uint8_t* str, uint16_t strLen, USART_TypeDef* ESP_USART) {
    //disable stream 7
    DMA2_STREAM7->CR.EN = 0;
    //set clear transfer complete interrupt flag, clearing it
    DMA2->HIFCR.CTCIF7 = 1;

    //set base address from which data will be read/written
    DMA2_STREAM7->M0AR = str;
    //set number of data items to be transferred to cmdLen
    DMA2_STREAM7->NDTR = strLen;
    //enable stream 7
    DMA2_STREAM7->CR.EN = 1;
    //while stream 7 transmit complete interrupt flag is low
    while(!DMA2->HISR.TCIF7);
    delay_millis(TIM2, 30);

    uint8_t response[512] = "";
    readString(ESP_USART, response);
    printData(response);
}

```