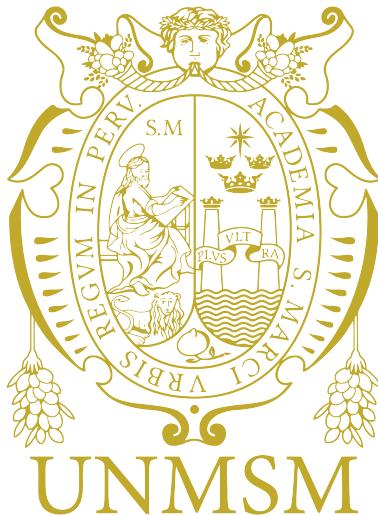


UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Facultad de Ingeniería de Sistemas e Informática

E.A.P. de Ingeniería de Sistemas



“Hydro ConCiencia”

Sistema de riego inteligente usando los microcontroladores ESP32 y

dsPIC33FJ32MC204

Docente:

Fermin Perez, Felix Armando

Curso:

Arquitectura de Computadoras

Grupo3:

- 22200193 - Cartagena Valera Brush, Davis Leonardo
- 22200134 - Rivera Llancari Aldair Alejandro
- 22200182 - Sernaque Cobeñas José Manuel
- 22200031 - Said William Najarro Llacza
- 22200113 - Huamani Quispe José Fernando
- 22200199 - Jhair Roussell Melendez Blas

Lima - Perú

2024

Índice

1	Introducción	4
2	Justificación	4
3	Alcance	5
4	Objetivos	5
4.1	Objetivo General	5
4.2	Objetivos Específicos	5
5	Investigación Previa	5
5.1	<i>Sobre el problema</i>	5
5.2	<i>Smart cities y fundamentos</i>	5
5.3	<i>Plantas y Macetas</i>	6
6	Materiales	7
6.1	dsPIC33FJ32MC204	7
6.2	ESP32 con entrada tipo C	7
6.3	Programador Pickit 3	7
6.4	Protoboard	7
6.5	Leds	7
6.6	Resistencias	8
6.7	Buzzer	8
6.8	Módulo I2C	8
6.9	Módulo MB-102	8
6.10	Sensor de humedad	9
6.11	Módulo relay 2CH 5VDC	9
6.12	Mini bomba de agua 5V	9
6.13	Jumper 20cm y 30cm	9
6.14	Broche	9
6.15	Condensador 16v y 10 miliamperios	10
6.16	Batería de 9V	10
7	Procedimiento	10
7.1	Discusión del problema y posible solución	10

7.2	Uso de Wokwi, Proteus y bocetos	10
7.3	Establecimiento de comunicación entre ESP-32 y dsPIC33FJ32MC204	11
7.4	Lectura de valores del sensor de humedad	12
7.5	Uso del PWM	13
7.6	Implementación del servidor web	14
7.7	Juego de luces Led	14
7.8	Ensamblaje del circuito	15
8	Conclusiones	15
9	Referencias	17
10	Anexos	17
10.1	Anexo 1: Evidencia de Reunión Virtual	17
10.2	Anexo 2: Evidencia de Reunión Presencial	17
10.3	Anexo 3: Código completo del dsPIC33FJ32MC204	17
10.4	Anexo 4: Código completo del ESP32	27
10.5	Anexo 5: Código del servidor web	35

Índice de figuras

1	dsPIC33FJ32MC204	7
2	ESP-32 con entrada tipo C	7
3	Programador Pickit 3	7
4	Protoboard	7
5	Resistencias	8
6	Buzzer	8
7	Módulo I2C	8
8	Módulo MB-102	8
9	Sensor de humedad de suelo	9
10	Módulo relay 2CH 5VDC	9
11	Mini bomba de agua 5V	9
12	Jumpers 20cm y 30cm	9
13	Broche para Batería	10
14	Condensador 16v y 10 miliamperios	10
15	Batería de 9V	10
16	Simulación en Proteus	11
17	Esquema del proyecto	11
18	Código de comunicación UART	12
19	Código de Interrupciones del dsPIC33FJ32MC204	12
20	Código de lectura de sensor de humedad	12
21	Algoritmo de quicksort	13
22	Suavizado mediante filtro de mediana	13
23	Estructura Color	13
24	Inicialización del PWM	13
25	Uso del PWM	14
26	Fragmento de código Django	14
27	Interfaz del servidor web	14
28	Simulación en Wokwi	15
29	Primer ensamblaje de los led	15
30	Reunión virtual	17
31	Reunión presencial	17

1. Introducción

El concepto de “smart city” representa un paradigma moderno en lo que respecta a la planificación urbana, en la cual se integran diversas tecnologías para mejorar la calidad de vida de la población. En relación a esto, una de sus seis dimensiones fundamentales es la denominada “smart environment.^º traducido como “entorno inteligente”, la cual se relaciona con la calidad del espacio o medio ambiente en el que se desarrolla la ciudad. En este sentido, el enfoque hacia la gestión eficiente del agua emerge como un desafío primordial, considerando este recurso como vital y crítico para la subsistencia de los ecosistemas urbanos y la calidad de vida de sus habitantes. En las áreas urbanas, donde el espacio verde a menudo es muy limitado, mantener la vitalidad de las plantas se convierte en una tarea crucial para garantizar la armonía entre la vida urbana y la naturaleza.

Las plantas, al igual que los seres humanos, dependen de un suministro de este recurso hídrico para su crecimiento y desarrollo óptimos. Sin embargo, el uso excesivo o inadecuado del agua al momento de la irrigación puede resultar en un desperdicio y degradación ambiental. Un sistema de riego inteligente, al adaptarse a las necesidades específicas de humedad del suelo, proporciona a las plantas la cantidad justa de agua en el momento adecuado. Esto no solo fomenta la conservación del recurso hídrico, sino que también contribuye a la belleza y la vitalidad de los espacios verdes urbanos.

En este contexto, se plantea la utilización en conjunto de los microcontroladores ESP32 y dsPIC33FJ32MC204 para la creación de un sistema de riego inteligente facilitando la toma de decisiones informadas al procesar datos en tiempo real provenientes de un sensor de humedad. De esta forma, no solo busca optimizar el uso del agua, un recurso vital, sino que también demuestra cómo la tecnología puede desempeñar un papel crucial en la gestión eficiente de nuestros entornos naturales.

En el presente informe se proporciona la justificación, alcance y objetivos del proyecto desarrollado. De igual manera, se describe a detalle los materiales y recursos usados así como también el procedimiento y las distintas pruebas realizadas hasta su adecuada implementación.

2. Justificación

El desarrollo e implementación de un sistema de riego inteligente para plantas domésticas usando los microcontroladores ESP32 y dsPIC33FJ32MC204 resulta importante y adecuado. Uno de los aspectos por lo cual se afirma lo anterior es que simplifica la vida de las personas, esto debido a que las libera de la preocupación y revisión constante sobre el estado de humedad de la tierra de sus plantas. Asimismo, se evita el desperdicio de agua asociado con métodos de riego tradicionales.

3. Alcance

El alcance de este proyecto se centra en el diseño, desarrollo y la realización de distintas pruebas de un sistema de riego inteligente para las plantas domésticas. De tal forma, que su operabilidad radica en los distintos hogares de las personas.

4. Objetivos

4.1. Objetivo General

Diseñar e implementar un sistema de riego inteligente basado en los microcontroladores ESP32 y dsPIC33FJ32MC204, que permita optimizar el uso del agua adecuado para el crecimiento y desarrollo de las plantas domésticas en una maceta.

4.2. Objetivos Específicos

- Utilizar adecuadamente el canal de comunicación UART entre el dsPIC33FJ32MC204 y uno de los ESP32.
- Realizar un juego de luces con varios leds basado en una música para que el usuario pueda activarlo.
- Implementar un programa de riego por comandos de voz.
- Diseñar un servidor web para visualizar la humedad leída por el sensor en tiempo real.

5. Investigación Previa

5.1. Sobre el problema

Todo surge con la necesidad creada por uno de tener un ambiente verde en su hogar, esto crea un problema que necesita ser resuelto; debido a la poca disponibilidad o interés que se pierde cuando las personas lo consiguen.

Esto surge en nosotros como una problemática, debido a la interacción mediante redes sociales o en la conversa con otras personas sobre la conservación de sus plantas y al ver que concluimos en lo mismo.

Por ello, para empezar a desarrollar este proyecto decidimos informarnos bien en los fundamentos en que se basa esta solución a un problema poco tomado en cuenta.

5.2. Smart cities y fundamentos

En general las literaturas que hablan sobre las ciudades inteligentes o “*smart cities*”, señalan que pueden tener múltiples definiciones, entre ellas, las que se pueden destacar son las siguientes:

- Ciudades que implementan aspectos tecnológicos en ámbito general como la infraestructura, acceso a servicios, etc.
- Ciudades que tienen desarrollo sostenible, o sea, buscar causar un menor impacto negativo en el progreso de la ciudad.

- Ciudades que progresan con una población comprometida con el avance; o sea, buscan una mayor participación de la población con el gobierno, así como con ellos mismos como en la innovación, educación, etc.

Obtenidas de Kozłowski y Suwar, 2021.

Estas definiciones son una interpretación nuestra acerca de las ciudades inteligentes para darle un fundamento al proyecto. Con esto mencionado, también podemos señalar una de las dimensiones que va a sustentar nuestra idea, esta es, según el artículo de Wojciech Kozłowski, Kacper Suwar la dimensión “*smart environment*” que nos dice que la polución, no un sector en específico, busca la conservación de espacios naturales o “*green areas*”.

Esto aplicado en nuestro caso busca la existencia de al menos una pequeña área verde en tu hogar, sin comprometer una gran cantidad de espacio.

Tierrogro, un supermercado agropecuario con un equipo que está capacitado como veterinarios, agrónomos, ingenieros mecánicos y personal experto en cada área, afirma que, generalmente, “las plantas suelen requerir de un 5 a un 10 % de agua de acuerdo a la capacidad de la maceta. Sin embargo, esto varía de acuerdo al lugar en la que está sembrada, el clima y el tipo de planta. Es crucial comprender estas variaciones para asegurar un suministro óptimo de agua que satisfaga las necesidades específicas de cada planta” (Tierragro, 2022). En el contexto de un sistema de riego inteligente basado en el microcontrolador ESP32, esta información cobra especial relevancia. La adaptabilidad del sistema a las condiciones específicas de humedad del suelo permite no solo optimizar el consumo de agua, sino también garantizar un entorno propicio para el crecimiento y desarrollo óptimos de las plantas en diferentes tipos de macetas y condiciones ambientales.

5.3. *Plantas y Macetas*

En una ciudad inteligente, la importancia de las plantas va más allá de la estética verde; son elementos vitales que contribuyen a un entorno sostenible y saludable. En este contexto, las plantas desempeñan un papel crucial al interactuar con tecnologías como el riego inteligente, asegurando no solo su propia vitalidad, sino también la eficiencia en el uso del agua y la preservación de espacios verdes urbanos.

En ese sentido, en un artículo publicado por

Este enfoque personalizado en el suministro hídrico, basado en las características individuales de las plantas y las macetas, se alinea con la visión de una ”Smart Environment.” en el contexto de las ciudades inteligentes. No solo contribuye a la eficiencia en el uso del recurso hídrico, sino que también promueve la preservación y prosperidad de las áreas verdes urbanas, elementos fundamentales para el bienestar de los habitantes y la sostenibilidad de los entornos naturales dentro de las ciudades.

6. Materiales

6.1. dsPIC33FJ32MC204

Es un microcontrolador perteneciente a la familia dsPIC de la empresa Microchip (fig 1). Posee capacidades de procesamiento digital de señales (DSP) y control de motor.



Figura 1: dsPIC33FJ32MC204

6.2. ESP32 con entrada tipo C

Este dispositivo actúa como el cerebro del sistema. El ESP32 es un microcontrolador de bajo consumo de energía con conectividad inalámbrica, lo que permite la programación y control remoto del regador automático (fig 2).

Su capacidad para manejar múltiples tareas y comunicarse con otros dispositivos lo convierte en una elección ideal.



Figura 2: ESP-32 con entrada tipo C

6.3. Programador Pickit 3

Permite la depuración y programación del microcontrolador dsPIC33FJ32MC204.



Figura 3: Programador Pickit 3

6.4. Protoboard

La protoboard es una herramienta esencial durante la fase de prototipado. Permite conectar de manera temporal y flexible los diferentes componentes del circuito sin necesidad de soldaduras. Esto facilita las pruebas y modificaciones en el diseño del regador automático.

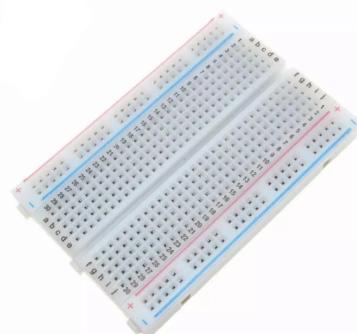


Figura 4: Protoboard

6.5. Leds

Es un diodo particular que emite luz al ser atravesado por una corriente eléctrica. Posee

un terminal positivo y otro negativo denominados ánodo y cátodo, respectivamente. La terminal más larga corresponde al positivo y la más corta al negativo.

6.6. Resistencias

Es un componente diseñado para limitar el flujo de la corriente eléctrica. Es posible su uso para proteger los leds de corrientes excesivas y de esa forma evitar que se quemen.



Figura 5: Resistencias

6.7. Buzzer

Es un componente electroacústico que produce un sonido o zumbido continuo o intermitente de un mismo tono. Su uso en el proyecto radicará en el acompañamiento del juego de luces led.



Figura 6: Buzzer

6.8. Módulo I2C

Este módulo simplifica la conexión de dispositivos al microcontrolador utilizando solo dos cables para la comunicación. Se puede usar para conectar eficientemente sensores y pantallas al ESP32, facilitando la expansión del sistema.

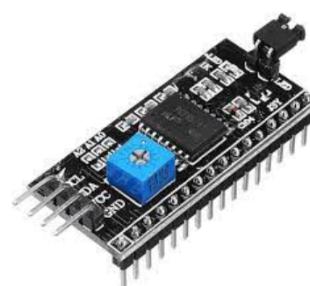


Figura 7: Módulo I2C

6.9. Módulo MB-102

La placa de distribución de energía MB-102 proporciona conexiones seguras y ordenadas para la alimentación de los componentes. Facilita la organización del cableado y asegura un suministro eléctrico estable para el regador automático.



Figura 8: Módulo MB-102

6.10. Sensor de humedad

Sensor de humedad del suelo Este sensor detecta la humedad del suelo, permitiendo al ESP32 determinar cuándo es necesario regar las plantas. Proporciona información crítica para automatizar el proceso de riego y garantizar un cuidado adecuado de las plantas.

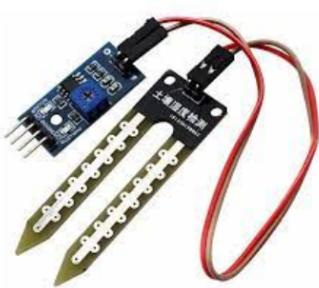


Figura 9: Sensor de humedad de suelo

6.11. Módulo relay 2CH 5VDC

Módulo relay 2CH 5VDC El módulo relay actúa como un interruptor controlado por el ESP32. Permite encender y apagar dispositivos de mayor voltaje, como la bomba de agua. Esto es esencial para controlar el suministro de agua de manera automática.



Figura 10: Módulo relay 2CH 5VDC

6.12. Mini bomba de agua 5V

Esta bomba suministra agua a las plantas cuando se activa mediante el módulo relay. La baja tensión (5V) la hace adecuada para el proyecto, y su control permite un riego eficiente y preciso.



Figura 11: Mini bomba de agua 5V

6.13. Jumper 20cm y 30cm

Los jumpers son cables que conectan los distintos componentes del circuito. Los de diferentes longitudes ayudan a organizar y mantener el orden en la protoboard durante el desarrollo del proyecto.



Figura 12: Jumpers 20cm y 30cm

6.14. Broche

Adaptador para suministrar energías de las pilas



Figura 13: Broche para Batería



Figura 15: Batería de 9V

6.15. Condensador 16v y 10 miliamperios

Componente que almacena y libera energía eléctrica de manera controlada. Puede utilizarse para estabilizar la alimentación y reducir el ruido en el sistema.



Figura 14: Condensador 16v y 10 miliamperios

6.16. Batería de 9V

Fuente de energía portátil para el regador automático. La batería asegura que el sistema pueda funcionar incluso en lugares sin acceso a una toma de corriente.

7. Procedimiento

7.1. Discusión del problema y posible solución

Se procede a debatir cuál sería la solución al problema planteado relacionado a la repartición de agua y qué deberíamos hacer para empezar a desarrollarlo. Para eso planteamos lo siguiente:

- Hacer un esquema del circuito.
- Identificar los materiales a utilizar.
- Ensamblar el circuito.
- Ejecutar el circuito simple.
- Realizar las pruebas y verificar el correcto funcionamiento.
- Proponer adicionales al proyecto.
- Concluir con el desarrollo.

7.2. Uso de Wokwi, Proteus y bocetos

Una vez definido el problema y la solución por ejecutar, resulta esencial validar la viabilidad

y funcionalidad del proyecto de manera sencilla antes de realizar la implementación física. Debido a esto, se requiere el uso de distintas herramientas que permitan comprobar si este se ejecutará de forma correcta.

La respuesta a esta necesidad recae en el uso de simuladores, entre los más usados se encuentran Wokwi para el ESP32 y Proteus para el dsPIC33FJ32MC204. De tal forma que sea posible materializar lo que sería el circuito junto a la lógica que lo acompaña.

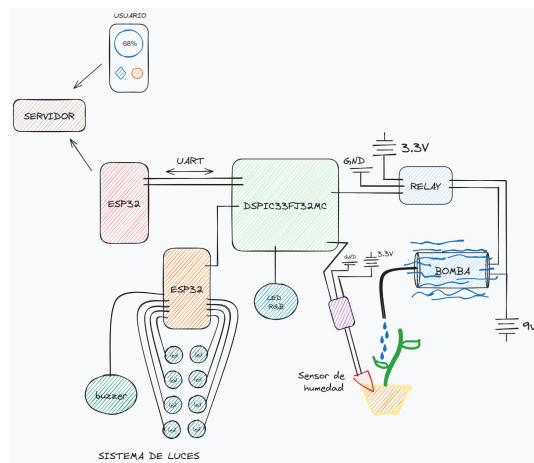


Figura 17: Esquema del proyecto

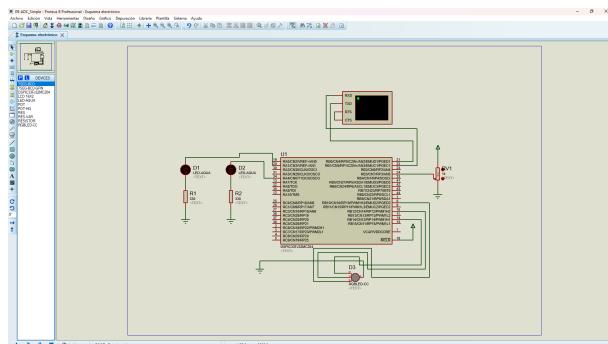


Figura 16: Simulación en Proteus

Asimismo, se diseñó un esquema general del circuito para que se pueda visualizar mejor la forma en que los distintos componentes interactúan dentro de esta y así tener una visión más general de los pasos para la realización del proyecto.

7.3. Establecimiento de comunicación entre ESP-32 y ds-PIC33FJ32MC204

El protocolo de comunicación utilizado entre estos dos microcontroladores es el UART (Universal Asynchronous Receiver/Transmitter), en el cual los datos se transfieren de manera serial, es decir, un bit a la vez. Este consta de dos líneas de comunicación: una para transmisión (TX) y otra para recepción (RX). De igual manera, es importante tener en cuenta que para la comunicación entre el ESP32 y el dsPIC33FJ32MC204 la línea de transmisión de un microcontrolador debe ir conectada a la línea de recepción del otro y viceversa.

```

void pin_config(void) {
    AD1PCFGL = 0xffff;
    AD1CHS0 = 0x0005;
    RPINR18 = 0x0000;
    RPOR0 = 0x0300;
    TRISAbits.TRISA1 = 0;
    TRISAbits.TRISA4 = 0;
}

void uart_start(void) {
    U1BRG = BRGVAL;
    U1MODEbits.UARTEN = 1;
    U1STA_bits.UTXEN = 1;
    IFS0bits.U1RXIF = 0;
    IEC0bits.U1RXIE = 1;
}

```

Figura 18: Código de comunicación UART

En la figura 18 se observa la configuración de los pines del dsPIC33FJ32MC204 RP0 como RX y el RP1 como TX.

```

void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void) {
    char data = U1RXREG; // Leer el dato recibido
    switch (data) {
        case '1':
            enable_light_show(); // Habilitar el espectáculo de luces
            break;
        case '2':
            disable_light_show(); // Deshabilitar el espectáculo de luces
            break;
        case '3':
            execute_plant_watering(); // Ejecutar el riego de la planta
            break;
        case '4':
            disable_plant_watering(); // Deshabilitar el riego de la planta
            break;
        case 'r':
            set_colors(P_RED); // Establecer el color rojo
            break;
        case 'g':
            set_colors(P_GREEN); // Establecer el color verde
            break;
        case 'b':
            set_colors(P_BLUE); // Establecer el color azul
            break;
    }
    IFS0bits.U1RXIF = 0; // Limpiar la bandera de interrupción de recepción
}

```

Figura 19: Código de Interrupciones del dsPIC33FJ32MC204

Además, se utiliza una interrupción para la recepción de datos provenientes de este canal de comunicación en la cual se lee el dato recibido ubicado en un registro y se guarda en una

variable tipo char.

7.4. Lectura de valores del sensor de humedad

Los valores leídos por el sensor de humedad son realizados a través de un ADC los cuales son procesados y convertidos en porcentaje para posteriormente ser enviados.

```

void loop(void) {
    while (1) {
        adc_prepare();
        int32_t humidity_raw_value = adc_read();
        int32_t humidity_value = process_data(humidity_raw_value);
        int8_t humidity_percentage = percentage(humidity_value);
        send_percentage(humidity_percentage);
        delay_ms(3000);
    }
}

```

Figura 20: Código de lectura de sensor de humedad

Cabe resaltar que para las lecturas del sensor de humedad se declaró un arreglo de tamaño 5 para el buffer de datos en el cual se almacenan los últimos cinco valores obtenidos del sensor. Estos valores se ordenan a través del algoritmo de quicksort, el cual se basa en la lógica “divide y vencerás”. Lo que hace es partir el arreglo en dos subarreglos en base a un pivote elegido, a la izquierda del pivote se encuentran valores menores que este y a su derecha valores mayores.

Posteriormente, cada subarreglo se ordena, acelerando la ejecución.

```

1 void swap(int32_t *a, int32_t *b) { /
2     int t = *a;
3     *a = *b;
4     *b = t;
5 }
6
7 int32_t partition(int32_t array[], int32_t low, int32_t high) {
8     int pivot = array[high];
9     int i = (low - 1);
10
11    for (int j = low; j <= high - 1; j++) {
12        if (array[j] < pivot) {
13            i++;
14            swap(&array[i], &array[j]);
15        }
16    }
17    swap(&array[i + 1], &array[high]);
18    return (i + 1);
19 }
20
21 void quick_sort(int32_t array[], int32_t low, int32_t high) {
22     if (low < high) {
23         int pi = partition(array, low, high);
24         quick_sort(array, low, pi - 1);
25         quick_sort(array, pi + 1, high);
26     }
27 }

```

Figura 21: Algoritmo de quicksort

Una vez ordenado estos valores, se procede a aplicar el filtro de suavizado por mediana. El propósito de aplicar este filtro es ignorar las anomalías que se puedan presentar como valores muy bajos o altos.

```

1 int32_t process_data(int32_t value) {
2     data_buffer[data_buffer_index] = value;
3     data_buffer_index = (data_buffer_index + 1) % DATA_SIZE;
4     return median_filter();
5 }
6
7 int32_t median_filter(void) {
8     quick_sort(data_buffer, 0, DATA_SIZE - 1);
9     if (DATA_SIZE % 2 == 0) {
10         return (data_buffer[DATA_SIZE / 2] + data_buffer[DATA_SIZE / 2 - 1]) / 2;
11     } else {
12         return data_buffer[DATA_SIZE / 2];
13     }
14 }

```

Figura 22: Suavizado mediante filtro de mediana

7.5. Uso del PWM

Para el uso del módulo PWM (modulación por ancho de pulsos) se utilizó un led RGB. En primer lugar, se define una estructura “Color” que contiene los componentes de intensidad del rojo, verde y azul. Después, se definen cons-

tantes de esta para mostrar colores específicos mediante valores predefinidos para cada componente.

```

1 typedef struct {
2     int red;
3     int green;
4     int blue;
5 } Color;
6
7 const Color P_BLUE = {0, 0, 255};
8 const Color P_GREEN = {0, 255, 0};
9 const Color P_RED = {255, 0, 0};

```

Figura 23: Estructura Color

Se inicializa el módulo PWM configurando sus parámetros como por ejemplo el periodo que determina la frecuencia de la señal PWM, es decir, cuántas veces se repite el ciclo de trabajo de por segundo.

```

1 void init_pwm(void) {
2     PTCONbits.PTEN = 0b00;
3     PTCONbits.PTCKPS = 0b00;
4     PTCONbits.PTOPS = 0b00;
5     PTPER = 999;
6     PWM1CON2bits.IUE = 1;
7     PWM1CON1bits.PMOD1 = 0;
8     PWM1CON1bits.PMOD2 = 0;
9     PWM1CON1bits.PMOD3 = 0;
10
11    PWM1CON1bits.PEN1H = 1;
12    PWM1CON1bits.PEN2H = 1;
13    PWM1CON1bits.PEN3H = 1;
14
15    PTCONbits.PTEN = 1;
16    P1DC1 = 0;
17    P1DC2 = 0;
18    P1DC3 = 0;
19 }

```

Figura 24: Inicialización del PWM

Lo que realiza el PWM es realizar una conversión por la cual se muestran estos colores en el led.

```

1 void set_colors(Color color) {
2     int _red = (color.red * 2000) / (255);
3     int _green = (color.green * 2000) / 255;
4     int _blue = (color.blue * 2000) / 255;
5
6     P1DC1 = _red; // Canal 1 (Rojo)
7     P1DC2 = _green; // Canal 2 (Verde)
8     P1DC3 = _blue; // Canal 3 (Azul)
9 }

```

Figura 25: Uso del PWM

```

1 from django.shortcuts import render
2 from django.http import HttpResponse
3 from django.views.decorators.csrf import csrf_exempt
4 import re
5
6 percenHum = 0
7 regarForzado = False
8 encenderLuces = False
9 solicitudRegado = {
10     'pendiente': False,
11     'modo': '',
12     'duracion': 0,
13     'timer': ''
14 }
15 comando = {
16     'recibido': False,
17     'contenido': '',
18     'respuesta': {
19         'recibida': False,
20         'valido': '',
21     }
22 }

```

Figura 26: Fragmento de código Django

7.6. Implementación del servidor web

La implementación de una aplicación externa ocasiona que busquemos una forma para almacenar datos en un servidor externo, esto es para que se pueda recopilar datos como la humedad detectada. El servidor del proyecto fue desarrollado en Django para poder acceder a las solicitudes admitidas. Estas son presentadas por el cliente tanto en el móvil como en la computadora y también como los dirigidos en ESP32. El frontend y la configuración del cliente ha sido realizado en HTML, CSS y Javascript. Asimismo, este servidor permite el ingreso de solicitudes tanto por audio como por escrito para la programación del encendido del sistema de riego. Como por ejemplo:

- Encender el sistema en 5 segundos
- Encender el sistema en 10 minutos
- Activar el riego en 5 minutos.

De igual manera, se diseñó una interfaz simple y amigable para la visualización de la humedad leída por el sensor así como también la programación de riego y el encendido del juego de luces de manera manual o por voz.

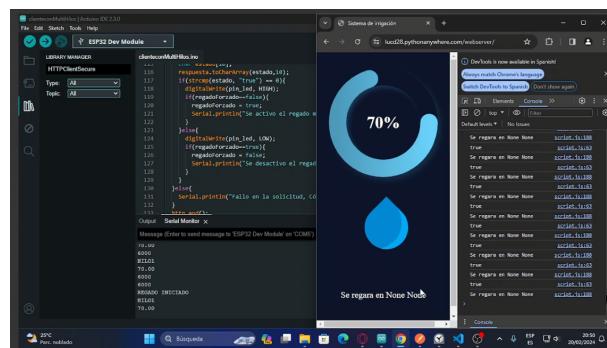


Figura 27: Interfaz del servidor web

7.7. Juego de luces Led

Se utilizaron diversos leds conectados a un ESP32 para la realización de un juego de luces acompañados de un buzzer o zumbador con el ritmo de una pieza musical que el usuario pueda activar. Inicialmente, se empleó Wokwi

para desarrollar el código y simular su funcionamiento.

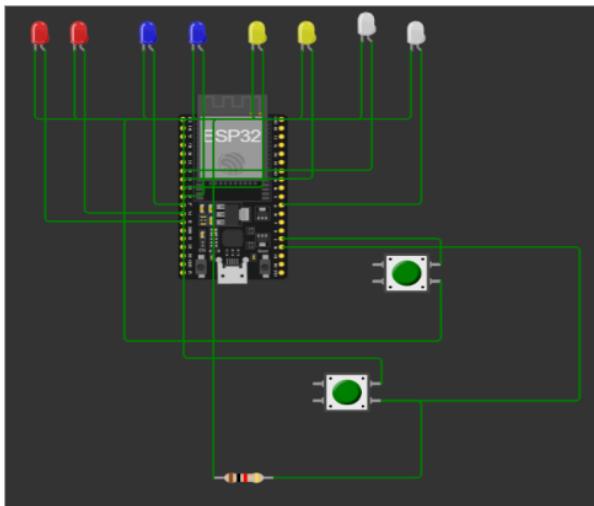


Figura 28: Simulación en Wokwi

Posteriormente, se construyó el circuito simulado en un protoboard con el ESP32 y los componentes requeridos.

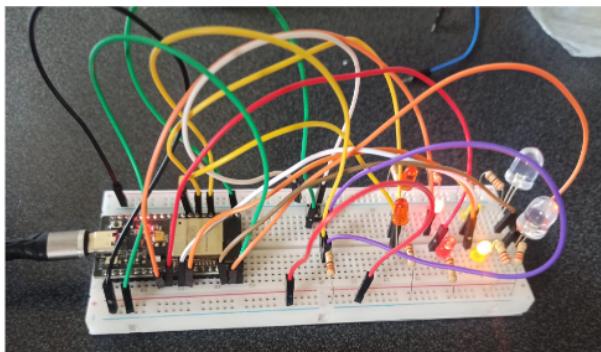


Figura 29: Primer ensamblaje de los led

Para su funcionamiento, se emplearon tareas independientes que se ejecutan en el núcleo 1 del ESP32, también conocidas como hilos o “*threads*”, de tal forma que sea posible controlar diferentes patrones de iluminación. Es decir, que se pueden controlar diversos leds de manera simultánea para crear los efectos visuales necesarios.

7.8. Ensamblaje del circuito

Una vez realizado los pasos anteriores se procede a ensamblar todo el circuito propuesto en los distintos protoboards. Para la implementación del código se usa la IDE de Arduino para el ESP32 y, por su parte, MPLAB para el dsPIC33FJ32MC204 con ayuda de un programador Pickit3.

8. Conclusiones

La colaboración entre los microcontroladores dsPIC33FJ32MC204 y ESP32 en este proyecto resultó fundamental para aprovechar las fortalezas de ambos. La combinación de la capacidad de procesamiento digital de señales del dsPIC33FJ32MC204 con la conectividad inalámbrica y la versatilidad del ESP32 permitió desarrollar un sistema de riego inteligente adecuado. A través de esta colaboración se ha demostrado la importancia de utilizar múltiples dispositivos en arquitecturas integradas para aprovechar al máximo sus capacidades y ofrecer soluciones más completas.

El desarrollo e implementación del sistema de riego inteligente basado en los microcontroladores ESP32 y dsPIC33FJ32MC304 ha demostrado ser una solución eficaz y prometedora para la gestión eficiente del agua en entornos domésticos. La integración de tecnología, como el sensor de humedad del suelo permite una adaptación precisa a las necesidades de las plantas, optimizando el uso del agua y fomen-

tando la conservación de este recurso vital.

La elección de los materiales y su ensamblado, seguido por la realización del procedimiento descrito, ha demostrado la viabilidad técnica del sistema. Las pruebas realizadas durante el desarrollo han permitido identificar y resolver desafíos, mejorando la robustez y eficacia del sistema.

9. Referencias

- Kozłowski, W., & Suwar, K. (2021). Smart city: definitions, dimensions, and initiatives.
- Tierragro. (2022, noviembre). Riego de plantas: Recomendaciones para regar las plantas [Acceso: 2024-2-29]. <https://www.tierragro.com/blogs/news/riego-plantas>

10. Anexos

10.1. Anexo 1: Evidencia de Reunión Virtual

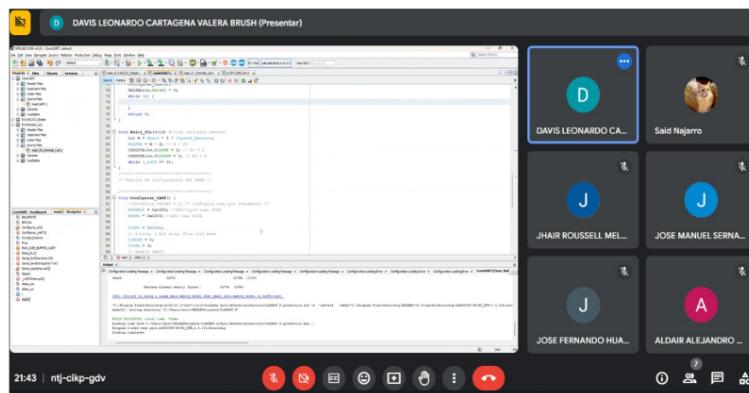


Figura 30: Reunión virtual

10.2. Anexo 2: Evidencia de Reunión Presencial



Figura 31: Reunión presencial

10.3. Anexo 3: Código completo del dsPIC33FJ32MC204

```

#include "xc.h"
#include <libpic30.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#pragma config PWMPIN = ON
#pragma config HPOL = ON
#pragma config LPOL = ON
#pragma config ALTI2C = OFF
#pragma config FPWRT = PWR128
#pragma config WDTPOST = PS32768
#pragma config WINDIS = OFF
#pragma config FWDTEN = OFF
#pragma config POSCMD = HS
#pragma config OSCIOFNC = OFF
#pragma config IOL1WAY = ON
#pragma config FCKSM = CSDCMD
#pragma config FNOSC = PRIPLL
#pragma config IESO = OFF
typedef struct {
    int red;
    int green;
    int blue;
} Color;

const uint8_t MAX_SPEED_CLOCK_MHZ = 40;
const uint8_t CRYSTALVALUEMHZ = 20;
const int32_t FREQ = 40 * 1000000;
const int32_t BAUDRATE = 9600;
const int32_t BRGVAL = ((40 * 1000000 / 9600) / 16) - 1;
const int32_t DATA_SIZE = 5;

```

```

const Color P_BLUE = {0, 0, 255};
const Color P_GREEN = {0, 255, 0};
const Color P_RED = {255, 0, 0};

int32_t data_buffer[5];
int32_t data_buffer_index = 0;
int32_t min_read = 0;
int32_t max_read = 1024;
void delay_ms(int32_t x);
void clock_pll(void);
void setup(void);
void pin_config(void);
void adc_config(void);
void adc_start(void);
void uart_start(void);
void loop(void);
void adc_prepare(void);
int32_t adc_read(void);
int32_t process_data(int32_t value);
void prepare_data_buffer(void);
int32_t median_filter(void);
void swap(int32_t *a, int32_t *b);
int32_t partition(int32_t arr[], int32_t low, int32_t high);
void quick_sort(int32_t arr[], int32_t low, int32_t high);
int8_t percentage(int32_t value);
void send_percentage(int8_t percentage);
void send_string(char *string);
void send_char(char character);
void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void);
void enable_light_show(void);
void disable_light_show(void);
void execute_plant_watering(void);
void init_pwm(void);

```

```

void set_colors(Color color);
void disable_plant_watering(void);

int main(void) {
    setup();
    loop();
}

void delay_ms(int32_t x) {
    __delay32((FREQ / 1000) * x);
}

void setup(void) {
    clock_pll();
    pin_config();
    adc_config();
    adc_start();
    uart_start();
    init_pwm();
    prepare_data_buffer();
    set_colors(P_GREEN);
}

void clock_pll(void) {
    int M = MAX_SPEED_CLOCK_MHZ * 8 / CRYSTAL_VALUE_MHZ;
    PLLFBD = M - 2;
    CLKDIVbits.PLLPRE = 0;
    CLKDIVbits.PLLPOST = 0;
    while (LOCK == 0) {
    }
}

void pin_config(void) {

```

```
AD1PCFGL = 0xffff ;  
AD1CHS0 = 0x0005 ;  
RPINR18 = 0x0000 ;  
RPOR0 = 0x0300 ;  
TRISAbits.TRISA1 = 0;  
TRISAbits.TRISA4 = 0;  
}
```

```
void adc_config(void) {  
    AD1CON1 = 0x00E4 ;  
    AD1CON2 = 0x003C ;  
    AD1CON3 = 0x0353 ;  
    AD1PCFGL = 0xFFDF ;  
}
```

```
void adc_start(void) {  
    AD1CON1bits.ADON = 1;  
}
```

```
void uart_start(void) {  
    U1BRG = BRGVAL;  
    U1MODEbits.UARTEN = 1;  
    U1STAbits.UTXEN = 1;  
    IFS0bits.U1RXIF = 0;  
    IEC0bits.U1RXIE = 1;  
}
```

```
void init_pwm(void) {  
    PTCONbits.PTEN = 0b00 ;  
    PTCONbits.PTCKPS = 0b00 ;  
    PTCONbits.PTOPS = 0b00 ;  
    PTPER = 999 ;  
    PWM1CON2bits.IUE = 1 ;
```

```

PWM1CON1bits.PMOD1 = 0;
PWM1CON1bits.PMOD2 = 0;
PWM1CON1bits.PMOD3 = 0;
PWM1CON1bits.PEN1H = 1;
PWM1CON1bits.PEN2H = 1;
PWM1CON1bits.PEN3H = 1;
PTCONbits.PTEN = 1;
P1DC1 = 0;
P1DC2 = 0;
P1DC3 = 0;
}

void prepare_data_buffer(void) {
    for (int32_t i = 0; i < DATA_SIZE; i++) {
        adc_prepare();
        int32_t humidity_raw_value = adc_read();
        data_buffer[i] = humidity_raw_value;
    }
}

void loop(void) {
    while (1) {
        adc_prepare();
        int32_t humidity_raw_value = adc_read();
        int32_t humidity_value = process_data(humidity_raw_value);
        int8_t humidity_percentage = percentage(humidity_value);
        send_percentage(humidity_percentage);
        delay_ms(3000);
    }
}

void adc_prepare(void) {
    while (!IFS0bits.AD1IF) {

```

```

    }

IFS0bits.AD1IF = 0;

}

int32_t adc_read(void) {
    int32_t value = ADC1BUF0;
    return value;
}

int32_t process_data(int32_t value) {
    data_buffer[data_buffer_index] = value;
    data_buffer_index = (data_buffer_index + 1) % DATA_SIZE;
    return median_filter();
}

int32_t median_filter(void) {
    quick_sort(data_buffer, 0, DATA_SIZE - 1);
    if (DATA_SIZE % 2 == 0) {
        return (data_buffer[DATA_SIZE / 2] + data_buffer[DATA_SIZE / 2 - 1]) / 2;
    } else {
        return data_buffer[DATA_SIZE / 2];
    }
}

void swap(int32_t *a, int32_t *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int32_t partition(int32_t array[], int32_t low, int32_t high) {
    int pivot = array[high];
    int i = (low - 1);
}

```

```

for (int j = low; j <= high - 1; j++) {
    if (array[j] < pivot) {
        i++;
        swap(&array[i], &array[j]);
    }
    swap(&array[i + 1], &array[high]);
    return (i + 1);
}

void quick_sort(int32_t array[], int32_t low, int32_t high) {
    if (low < high) {
        int pi = partition(array, low, high);
        quick_sort(array, low, pi - 1);
        quick_sort(array, pi + 1, high);
    }
}

int8_t percentage(int32_t value) {
    return 100 - ((value - min_read) * 100 / (max_read - min_read));
}

void send_percentage(int8_t percentage) {
    char buffer[10];
    sprintf(buffer, "%d\r", percentage);
    send_string(buffer);
}

void send_string(char *string) {
    while (*string) {
        send_char(*string++);
    }
}

```

```
}

void send_char(char character) {
    while (U1STAbits.UTXBF == 1) {
        }
    U1TXREG = character;
}

void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void) {
    char data = U1RXREG;
    switch (data) {
        case '1':
            enable_light_show();
            break;
        case '2':
            disable_light_show();
            break;
        case '3':
            execute_plant_watering();
            break;
        case '4':
            disable_plant_watering();
            break;
        case 'r':
            set_colors(P_RED);
            break;
        case 'g':
            set_colors(P_GREEN);
            break;
        case 'b':
            set_colors(P_BLUE);
            break;
    }
}
```

```

IFS0bits.U1RXIF = 0;
}

void enable_light_show(void) {
    if (LATAbits.LATA1 == 0) {
        LATAbits.LATA1 = 1;
    }
}

void disable_light_show(void) {
    if (LATAbits.LATA1 == 1) {
        LATAbits.LATA1 = 0;
    }
}

void execute_plant_watering(void) {
    if (LATAbits.LATA4 == 0) {
        LATAbits.LATA4 = 1;
    }
}

void set_colors(Color color) {
    int _red = (color.red * 2000) / (255);
    int _green = (color.green * 2000) / 255;
    int _blue = (color.blue * 2000) / 255;

    P1DC1 = _red;
    P1DC2 = _green;
    P1DC3 = _blue;
}

void disable_plant_watering(void) {
    if (LATAbits.LATA4 == 1) {
}

```

```

LATAbits.LATA4 = 0;
}
}

```

10.4. Anexo 4: Código completo del ESP32

```

#include <HTTPClient.h>
#include <WiFi.h>
#include <atomic>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <stdio.h>
#include <string.h>

const char *ssid = "";
const char *password = "";

const String host = "https://lucd28.pythonanywhere.com/webserver";
const int port = 80;

const uint8_t pin_rx = 16;
const uint8_t pin_tx = 17;
std::atomic<double> humedad(0.0);
TaskHandle_t Tarea1;

void setup() {
    Serial.begin(9600);

    Serial1.begin(9600, SERIAL_8N1, pin_rx, pin_tx);

    Serial.println("Conectando a WiFi");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Conectando . . .");
    }
    Serial.println("Conectado a la red WiFi");

    xTaskCreatePinnedToCore(loop1, "Tarea_1", 18000, NULL, 14, &Tarea1, 1);
    Serial.println("CREADOS");
}

```

```

}

void loop() {
    // Lectura de datos del sensor
    char message[15];
    if (!Serial1.available()) {
        Serial1.println('r');
    }
    if (Serial1.available()) {
        Serial1.readBytesUntil('\r', message, sizeof(message));
        Serial.print(message);
        Serial.print(" -- ");
        int a = atoi(message);
        a += 30;
        Serial.println(a);

        humedad = a;
        updateHumedad();
    }
    if (a < 60) {
        Serial1.println('3');
        Serial1.println('4');
    } else {
        Serial1.println('4');
    }
}
if (Serial.available()) {
    char send = Serial.read();
    Serial1.println(send);
}
}

```

```

void loop1(void *parameter) {
    long espera = -1;
    unsigned long inicio = 0;
    char modo[100];

    while (true) {
        ObtenerComando(&espera, &inicio, modo);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}

```

```

strcpy(modo, " ");
}

void updateHumedad() {
    HTTPClient cHumedad;
    Serial.println(humedad.load());
    char buffer[40];
    sprintf(buffer, "%f", humedad.load());
    String url_h = host + "/humedad?valor=" + buffer;
    cHumedad.begin(String(url_h));
    cHumedad.addHeader("Content-Type", "text/plain");
    int rpt = cHumedad.POST("valor=" + String(humedad, 1));
    while (rpt != 200) {
        rpt = cHumedad.POST("valor=" + String(humedad, 1));
    }
    if (rpt > 0) {
        String response = cHumedad.getString();
    }
    cHumedad.end();
}

void ObtenerComando(long *espera, unsigned long *inicio, char modo[]) {
    const String host2 = "https://lucd28.pythonanywhere.com/webserver";
    char parametros[5][30];
    HTTPClient http;
    String url = host2 + "/rcomando";
    http.begin(url.c_str());
    http.addHeader("Content-Type", "text/plain");
    int Codigohttp = http.GET();
    while (Codigohttp != 200) {
        Codigohttp = http.GET();
    }

    if (Codigohttp > 0) {
        String respuesta = http.getString();
        char comando[80];
        respuesta.toCharArray(comando, 80);

        if (strcmp(comando, "Comando-no-recibido") != 0) {

```

```

Serial.println("COMANDO");
Serial.println(comando);
bool flag = evaluarComando(comando, modo, parametros);

if (flag == true) {
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    programarRegado(espera, inicio, parametros);
    Serial1.println('b');
    while (millis() - (*inicio) <= (*espera)) {
        Serial.println("ESPERANDO");
    }
    Serial1.println('g');
    Serial.println(*espera));
    Serial.println("REGADO-INICIADO");
    vTaskDelay(1000 / portTICK_PERIOD_MS);

    if (strcmp(modo, "regaren") == 0) {
        iniciarRegadoProgramado();
    } else if (strcmp(modo, "lucesen") == 0) {
        iniciarEncendidoProgramado();
    } else if (strcmp(modo, "apagarlucesInmediato") == 0) {
        apagar_leds();
    } else if (strcmp(modo, "apagarregadoInmediato") == 0) {
        apagar_relay();
    } else if (strcmp(modo, "regarInmediato") == 0) {
        activar_relay();
    } else if (strcmp(modo, "lucesInmediato") == 0) {
        encender_leds();
    }

    completarProgramacion();
}

else {
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    denegarSolicitud();
}
}

strcpy(modo, "");
strcpy(comando, "");

```

```

} else {
    Serial.println("ERROR-EN-obtenercomando");
    Serial.println("Fallo-en-la-solicitud,-Codigo-de-estado:-" + Códigohttp);
}

strcpy(parametros[0], "");
strcpy(parametros[1], "");
strcpy(parametros[2], "");
strcpy(modo, "");
(*espera) = -1;

http.end();
}

bool evaluarComando(char *comando, char modo[], char parametros[5][30]) {
    if (comando[0] == '&') {
        int longitud = strlen(comando);

        for (int i = 0; i < longitud; i++) {
            comando[i] = comando[i + 1];
        }
    }

    Serial.println("Comando-evaluado");

    int indice = 0;
    char *palabra = strtok(comando, " ,");

    while (palabra != NULL) {
        strcpy(parametros[indice], palabra);
        indice++;
        palabra = strtok(NULL, " ,");
    }

    if (strcmp(parametros[1], "None") == 0) {
        return false;
    }

    strncpy(modo, parametros[0], sizeof(modo) - 1);
}

```

```

modo[sizeof(modo) - 1] = '\0';

return true;
}

void programarRegado(long *espera, unsigned long *inicio, char parametros[5][30]) {
    HTTPClient cliente;
    String url = "https://lucd28.pythonanywhere.com/webserver/rptcomando?estado=valido";
    cliente.begin(url.c_str());
    int httpResponseCode = cliente.POST("estado=valido");
    while (httpResponseCode != 200) {
        httpResponseCode = cliente.POST("estado=valido");
    }

    if (httpResponseCode > 0) {
        if ((strcmp(parametros[0], "regaren") == 0) || (strcmp(parametros[0], "lucesen") ==
            (*inicio) = millis() / 1000;
            (*espera) = std::stoi(parametros[1]));
        if (strcmp(parametros[2], "seg") == 0 || strcmp(parametros[2], "segundos") ==
            (*espera) *= 1000;
        } else if (strcmp(parametros[2], "min") == 0 || strcmp(parametros[2], "minutos") ==
            (*espera) *= 60000;
        } else if (strcmp(parametros[2], "horas") == 0) {
            (*espera) *= 3600000;
        }
        if ((*espera) > 10000) {
            (*espera) = 10000;
        } else if (((*espera) <= 10000) && ((*espera) > 4000)) {
            (*espera) = 4000;
        }
    } else if ((strcmp(parametros[0], "lucesInmediato") == 0) || (strcmp(parametros[0],
        (*inicio) = millis();
        (*espera) = 0;
    }
} else {
    Serial.println("error en programar regado");
    Serial.println("Fallo en la solicitud , -Codigo de estado : -" + httpResponseCode);
}

cliente.end();

```

```

}

void denegarSolicitud() {
    HTTPClient cliente;
    String url = "https://lucd28.pythonanywhere.com/webserver/rptcomando?estado=invalido";
    cliente.begin(url.c_str());
    cliente.addHeader("Content-Type", "text/plain");

    int httpResponseCode = cliente.POST("estado=invalido");
    while (httpResponseCode != 200) {
        httpResponseCode = cliente.POST("estado=invalido");
    }
    if (httpResponseCode > 0) {
    } else {
        Serial.println("ERROR EN denegarSolicitud");
        Serial.println("Fallo en la solicitud , - Código de estado : -" + httpResponseCode);
    }

    cliente.end();
}

void completarProgramacion() {
    HTTPClient http;
    String url = "https://lucd28.pythonanywhere.com/webserver/riegoprog?pendiente=False";
    http.begin(url.c_str());
    http.addHeader("Content-Type", "text/plain");
    int httpResponseCode = http.POST("pendiente=False");
    while (httpResponseCode != 200) {
        httpResponseCode = http.POST("pendiente=False");
    }

    if (httpResponseCode > 0) {
    } else {
        Serial.println("ERROR EN completarProgramacion");
        Serial.println("Fallo en la solicitud , - Código de estado : -" + httpResponseCode);
    }
    http.end();
}

void iniciarRegadoProgramado() {

```

```

HTTPClient http;
String url = "https://lucd28.pythonanywhere.com/webserver/regar?state=1";
http.begin(url.c_str());
http.addHeader("Content-Type", "text/plain");
int httpResponseCode = http.POST("state=1");
while (httpResponseCode != 200) {
    httpResponseCode = http.POST("state=1");
}

if (httpResponseCode > 0) {
    activar_relay();
}

} else {
    Serial.println("ERROR-EN-iniciarRegadoProgramado");
    Serial.println("Fallo-en-la-solicitud,-Codigo-de-estado:-" + httpResponseCode);
}

http.end();
}

void iniciarEncendidoProgramado() {
    HTTPClient http;
    String url = "https://lucd28.pythonanywhere.com/webserver/encender?state=1";
    http.begin(url.c_str());
    http.addHeader("Content-Type", "text/plain");
    int httpResponseCode = http.POST("state=1");
    while (httpResponseCode != 200) {
        httpResponseCode = http.POST("state=1");
    }

    if (httpResponseCode > 0) {
        encender_leds();
    }

} else {
    Serial.println("ERROR-EN-iniciarEncendidoProgramado");
    Serial.println("Fallo-en-la-solicitud,-Codigo-de-estado:-" + httpResponseCode);
}

http.end();
}

```

```
void activar_relay() {
    Serial1.println('3');
}

void encender_leds() {
    Serial1.println('1');
}

void apagar_leds() {
    Serial1.println('2');
}

void apagar_relay() {
    Serial1.println('4');
}
```

10.5. Anexo 5: Código del servidor web

Puesto que el servidor contiene código muy extenso, se prefirió crear un repositorio en GitHub para su visualización. El enlace es el siguiente: <https://github.com/lucad-28/ServidorRiego/tree/main/webserver>