Semáforo Inteligente

Espinosa Tlatelpa Darina Jocelyn[0009-0006-5506-170X], Ramírez Martínez Ana Karen[0009-0004-2204-6285] , Romero Luna Gabriel[0009-0001-5255-2195] and Sampallo Amador Cesar Serafín[0009-0007-3059-2122]

1 Avenida San Claudio y 14 Sur, Colonia San Manuel, Edificio CCO4-211, Ciudad Universitaria, C.P. 72570, Puebla, Puebla. México.
darina.espinosa@alumno.buap.mx, ana.ramirezmart@alumno.buap.mx, gabriel.romerol@alumno.buap.mx , cesar.sampallo@alumno.buap.mx

Abstract. This project develops an ESP32-based traffic light system using a layered architecture to achieve clarity, modularity, and ease of maintenance. The presentation layer is a Tkinter GUI that issues HTTP GET requests to switch lights, start/stop the routine, and adjust timings. The business layer runs on the ESP32 with a web server (port 80) and logic that evolved from a blocking routine to a non-blocking state machine based on millis(), allowing the controller to remain responsive to the GUI while the sequence runs. The data/IO layer consolidates pin mappings, network parameters, and hardware access (GPIO → relays → lamps/LEDs). Prototyping began with relays and AC bulbs to emulate real operation, followed by a safer LED prototype on a breadboard. Tests confirmed connectivity, correct sequencing, relay activation (indicator LEDs and audible click), real-time remote control, and hot reconfiguration. The result is a synchronized, robust, and scalable system; future work includes an emergency mode and a local web interface for monitoring and control.

Keywords: Semáforo inteligente, Arquitectura en capas, ESP32, control de semáforos, interfaz, HTTP, servidor web, sincronización.

1 Introducción

En el contexto actual de la ingeniería electrónica y la programación, la automatización y el control de sistemas desempeñan un papel crucial en la resolución de problemas prácticos. La arquitectura en capas se ha consolidado como un enfoque fundamental para diseñar sistemas robustos, escalables y fáciles de mantener [1]. Este proyecto tiene como propósito explorar la relevancia de la arquitectura en capas mediante el desarrollo de un sistema de control de semáforos basado en el microcontrolador ESP32, destacando su capacidad para mejorar la claridad, la reutilización del código y la adaptabilidad del sistema.

La arquitectura en capas organiza el sistema en niveles diferenciados, cada uno con responsabilidades específicas, lo que facilita la modularidad y el mantenimiento del software [2]. En este proyecto, se implementaron tres capas principales:

La capa de presentación se encarga de la interacción física con el entorno, empleando componentes como relevadores y bombillas para manipular la corriente

AC y simular la operación de un semáforo real. En la capa de negocio, se implementa el cerebro del sistema, la lógica programada en C++ para el microcontrolador ESP32 que define las rutinas de tiempo y la sincronización entre semáforos. Finalmente, la capa de datos maneja la configuración del sistema, incluyendo los pines del microcontrolador y los parámetros de red para la conectividad.

El objetivo de este documento es detallar la metodología utilizada para la creación de cada una de estas capas, la implementación de los prototipos, las pruebas de funcionamiento y las resoluciones de los problemas encontrados, demostrando cómo una arquitectura de software bien definida permite construir un sistema funcional y adaptable, capaz de manejar múltiples semáforos de manera sincronizada y ser controlado a través de una interfaz gráfica de usuario.

2 Estado del arte

El desarrollo de semáforos inteligentes se ha consolidado como una estrategia fundamental para mejorar la eficiencia del tráfico vehicular y la seguridad vial en las ciudades inteligentes. Los sistemas modernos combinan microcontroladores, sensores y plataformas de visualización de datos, permitiendo optimizar la operación de los semáforos y adaptarla a las condiciones reales del tránsito.

Yépez Zambrano (2024) presenta un prototipo de semáforo inteligente basado en ESP32 y sensores, que ajusta los tiempos de los semáforos según la cantidad de vehículos detectados en tiempo real. Este sistema se conecta a Arduino Cloud, permitiendo la visualización y almacenamiento de datos mediante paneles interactivos. El estudio evidencia cómo la interacción entre sensores, semáforos y microcontroladores mejora la gestión del tráfico y genera información útil para la planificación urbana[5].

Martínez Torres et al. (2024) desarrollan un sistema de intervención de semáforos para servicios de emergencia, utilizando microcontroladores para cambiar la luz a verde y agilizar el paso de vehículos críticos. La investigación enfatiza la necesidad de un control dinámico y seguro del flujo vehicular, destacando que los sistemas embebidos pueden responder a eventos específicos sin comprometer la seguridad vial.[6]

Chuchuca Gómez (2024) describe un sistema de alarma residencial basado en ESP32 y monitoreo web, donde los sensores envían datos en tiempo real a una plataforma de visualización. Aunque se centra en seguridad residencial, su relevancia para los semáforos inteligentes radica en la eficacia del ESP32 en la adquisición de datos, procesamiento local y transmisión inalámbrica, y en la separación de funciones entre sensores, control y visualización, lo que constituye un ejemplo práctico de arquitectura modular en capas [7].

Complementando estos antecedentes, Tonato Chuquimarca y Sinche Maita (2022) realizan un análisis comparativo de arquitecturas de sistemas IoT, mostrando que no existe un estándar único y destacando las ventajas de la modularidad y escalabilidad que ofrece la arquitectura en capas. Esta aproximación permite separar claramente la capa de adquisición de datos, la capa de procesamiento y control y la capa de comunicación y visualización, facilitando el desarrollo de sistemas embebidos complejos y garantizando interoperabilidad, mantenimiento y expansión futura[8].

En conjunto, los estudios revisados evidencian tendencias comunes en sistemas inteligentes: el uso de ESP32 como núcleo de control, la integración de sensores para detección de eventos, el procesamiento en tiempo real y la visualización de datos en plataformas externas.

3 Metodología

En esta sección se describe el proceso sistemático seguido para el desarrollo de las capas que componen el sistema de control de semáforos inteligente. Se adoptó un enfoque iterativo que incluyó las siguientes etapas: diseño de prototipos, implementación de hardware y software, pruebas funcionales y optimización del sistema. Este proceso permitió validar cada componente de manera independiente antes de su integración, garantizando un sistema robusto y coordinado.

3.1 Capa de Presentación

Para la implementación de la interfaz gráfica se dieron los elementos visuales que permitieron la interacción entre el usuario y la computadora.

Para facilitar la interacción con el sistema, se desarrolló una interfaz gráfica de usuario (GUI) con los siguientes elementos funcionales:

- A) Imagen del semáforo con animación de encendido y apagado
- B) 1 Botón de control para cada lampara (verde, amarillo y rojo)
- C) 1 Botón de inicio de la rutina del semáforo
- D) 1 Botón de paro de la rutina del semáforo
- E) 1 Scale que sintonice los tiempos de la lámpara verde
- F) 1 Scale que sintonice los tiempos de la lámpara amarilla
- G) 1 Scale que sintonice los tiempos de la lámpara roja

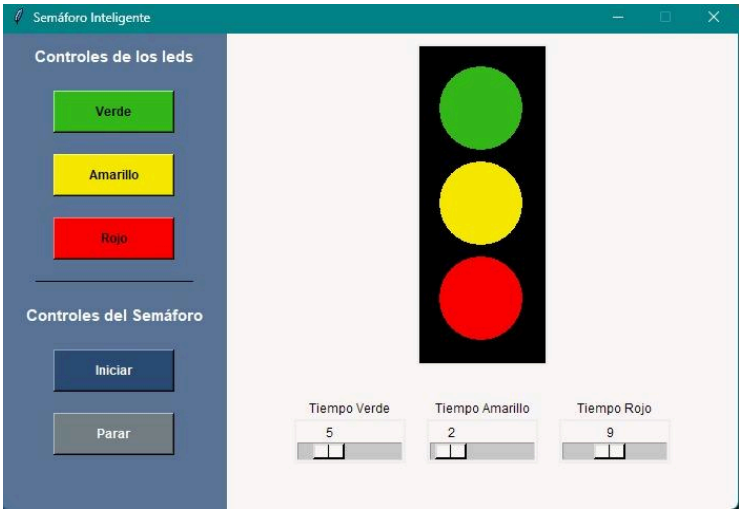


Fig. 1. Interfaz gráfica que controla el semáforo.

3.2 Capa de Negocio

La capa de negocio es el núcleo de nuestro sistema ya que en ella definimos la rutina que debe seguir cada semáforo a fin de lograr una correcta sincronización entre ellos. Para esta capa se utilizó el ide Arduino para la generación del código en C++ y su carga en la esp32.

La elaboración de esta capa se comenzó con la definición de la rutina del semáforo 1; que consiste en el siguiente comportamiento:

Tabla 1. Rutina base del semáforo.

Fase	Foco Verde	Foco Amarillo	Foco Rojo	Duración
1	**Enecender**	Apagar	Apagar	5 segundos
2	**Parpadear**	Apagado	Apagado	5 segundos por parpadeo
3	Apagar	**Encender**	Apagado	1.5 segundos
4	Apagado	**Apagar**	**Encender**	9.5 segundos

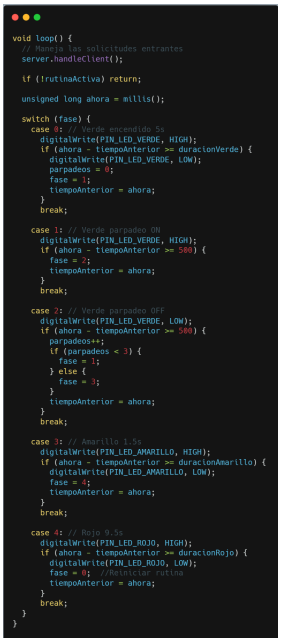
Posteriormente, la estrategia que seguimos para lograr la sincronización del semáforo 1 con el semáforo 2 fue encapsular la lógica del primer semáforo en la función “rutina” con 3 parámetros correspondiente a los pines que debe encender.

```
void rutina(int pinVerde, int pinAmarillo, int pinRojo){
  //Fase 1 Verde Encendido
  digitalWrite(pinVerde, HIGH);  // Enciende luz VERDE 5 segundos
  delay(5000);
  digitalWrite(pinVerde, LOW);  // Apaga luz VERDE

  //Fase 2 Verde Parpadeo
  digitalWrite(pinVerde, HIGH);  // Enciende luz VERDE 500 milisegundos
  delay(500);
  digitalWrite(pinVerde, LOW);  // Apaga luz VERDE 500 milisegundoso
  delay(500);

  digitalWrite(pinVerde, HIGH);  // Enciende luz VERDE 500 milisegundos
  delay(500);
  digitalWrite(pinVerde, LOW);  // Apaga luz VERDE 500 milisegundos
  delay(500);

  digitalWrite(pinVerde, HIGH);  // Enciende luz VERDE 500 milisegundos
  delay(500);
  digitalWrite(pinVerde, LOW);  // Apaga luz VERDE 500 milisegundos
  delay(500);

  //Fase 3 Amarillo
  digitalWrite(pinAmarillo, HIGH);  // Enciende luz AMARILLO 1.5 segundos
  delay(1500);
  digitalWrite(pinAmarillo, LOW);  // Apaga luz AMARILLO

  //Fase 4 Rojo Encendido
  digitalWrite(pinRojo, HIGH);  // Enciende luz ROJO 9.5 segundos

}
```

Fig. 2. Código que implementa la rutina del semáforo con delay().

De esta manera, logramos usar la misma función para encender tanto los pines del primer semáforo como los del segundo semáforo. Para lograr el efecto de cruce de semáforo llamamos a la función “rutina” con los pines del primer semáforo y posteriormente la volvemos a llamar con los pines del segundo semáforo en un ciclo.

Sin embargo, la incorporación de la interfaz gráfica trajo modificaciones al código. En lugar de la rutina secuencial que encadenaba los dos semáforos, se implementó una máquina de estados no bloqueante basada en millis() y un servidor HTTP estable en el puerto 80. Así, la app interfaz puede enviar GET a rutas claras por ejemplo, /led/<color>/(on|off) para accionar cada luz, /rutinaSemaforo/(on|off) para arrancar o detener la secuencia, y /configurar?verde=..&amarillo=..&rojo=.. para ajustar tiempos mientras la ESP32 sigue atendiendo solicitudes en paralelo gracias a server.handleClient(). Además, se movió la lógica de control a variables de estado (fase, parpadeos, rutinaActiva, duracion*).



```
void loop() {
  // Maneja las solicitudes entrantes
  server.handleClient();

  if (!rutinaActiva) return;

  unsigned long ahora = millis();

  switch (fase) {
    case 0: // Verde encendido 5s
      digitalWrite(PIN_LED_VERDE, HIGH);
      if (ahora - tiempoAnterior >= duracionVerde) {
        digitalWrite(PIN_LED_VERDE, LOW);
        parpadeos = 0;
        fase = 1;
        tiempoAnterior = ahora;
      }
      break;

    case 1: // Verde parpadeo ON
      digitalWrite(PIN_LED_VERDE, HIGH);
      if (ahora - tiempoAnterior >= 500) {
        fase = 2;
        tiempoAnterior = ahora;
      }
      break;

    case 2: // Verde parpadeo OFF
      digitalWrite(PIN_LED_VERDE, LOW);
      if (ahora - tiempoAnterior >= 500) {
        parpadeos++;
        if (parpadeos < 3) {
          fase = 1;
        } else {
          fase = 3;
        }
        tiempoAnterior = ahora;
      }
      break;

    case 3: // Amarillo 1.5s
      digitalWrite(PIN_LED_AMARILLO, HIGH);
      if (ahora - tiempoAnterior >= duracionAmarillo) {
        digitalWrite(PIN_LED_AMARILLO, LOW);
        fase = 4;
        tiempoAnterior = ahora;
      }
      break;

    case 4: // Rojo 9.5s
      digitalWrite(PIN_LED_ROJO, HIGH);
      if (ahora - tiempoAnterior >= duracionRojo) {
        digitalWrite(PIN_LED_ROJO, LOW);
        fase = 0;  // Reinicia rutina
        tiempoAnterior = ahora;
      }
      break;
  }
}
```

Fig. 3. Código que implementa la rutina del semáforo con máquina de estados.

3.3 Capa de Datos o de Acceso al Hardware

Prototipo 1: Semáforo con relevadores

La capa de datos no es solo "parámetros" en memoria: también incluye el hardware que materializa las órdenes. Los focos y los relevadores actúan como sumideros de datos (reciben señales ON/OFF) a través de los GPIO y producen el efecto físico de encender o cortar la carga. Por eso, en esta capa ubicamos el mapeo de pines, el módulo de relés , la alimentación y el GND común.

En la primera fase, se emplearon relevadores de 5V para controlar el flujo de corriente alterna (AC) hacia bombillas, que representaban las luces de los semáforos (verde, ámbar y rojo). Este diseño asegura una operación segura y eficiente, ya que los relevadores actúan como interruptores controlados por voltaje de corriente directa (DC) desde el microcontrolador ESP32 [3].

Los materiales utilizados para la primera implementación fueron:

- 1 Clavija
- 2m de cable dipolo calibre 14
- Estructura del semáforo
- 3 módulos de relevadores de 5V
- Cable macho-macho y macho-hrembra
- 1 arduino con cable
- 3 bombillas de corriente alterna (verde, rojo y amarillo)
- 3 sockets para las bombillas

Para el desarrollo de esta capa se utilizó un componente importante para el ensamblaje de cada componente, el ESP32.

Los pines que se utilizaron en el ESP32 para el control de la lógica del semáforo fueron:

- PIN 21 Relé verde
- PIN 22 Relé amarillo
- PIN 23 Relé rojo

Posteriormente se hizo la conexión de los puentes de alimentación de la ESP32 al Protoboard, se identificaron las entradas y salidas de los relevadores para seguir con el desarrollo del prototipo de la **Fig4**.

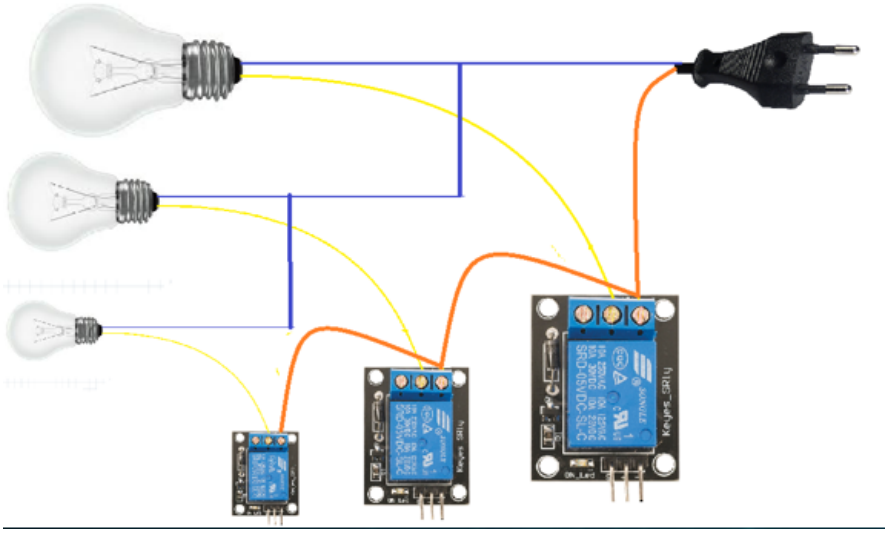


Fig. 4. Ilustra una guía de cableado para conectar relés y controlar varias bombillas mediante una fuente de alimentación de CA. Esta guía sirve como referencia práctica para ensamblar la capa de presentación de un sistema de control de semáforos. Esta configuración permite que el microcontrolador ESP32 (no mostrado) active o desactive los relés, controlando así el encendido y apagado de las bombillas. El diagrama sirve como guía paso a paso para garantizar las conexiones eléctricas correctas y la conformidad con las especificaciones de diseño del sistema.

Una vez terminado el ensamblado, se continuó con las pruebas con el código desarrollado, una vez verificado el funcionamiento de la ESP32 en Arduino y la conectividad, se cargó el código a la ESP32 para probar la rutina del código.

Prototipo 2: Semáforo adicional

Para la segunda implementación se añadió el "*Semáforo 2*" con nuevos componentes físicos y una nueva rutina. En el semáforo adicional se utilizaron LEDs en lugar de bombillas, lo que permitió una implementación más compacta y eficiente.

Los materiales utilizados para el "*Semáforo 2*" fueron:

- 1 Protoboard
- 3 leds (verde, amarillo y rojo)
- 3 resistencias de 300 ohms
- Alambre estañado
- Cable macho-macho y macho-hrembra

Los pines que se utilizaron en el ESP32 para el "*Semáforo 2*" fueron:

- PIN 15 Led verde
- PIN 16 Led amarillo
- PIN 17 Led rojo

Realizamos el ensamblado de los componentes, se añadieron las nuevas conexiones al ESP32 con los pines establecidos, con la nueva estructura se modificó el código de manera en que la rutina funcionara de manera sincronizada con ambos semáforos.

4 Pruebas experimentales

4.1 Conexión

Se comenzó probando la conexión a la ESP32, utilizando un código que permite al ESP32 conectarse a una red Wi-Fi e iniciar un servidor web en el puerto 80, esto para comprobar que podríamos continuar con la implementación de la lógica de negocio y la transmisión de órdenes remotas desde nuestra interfaz gráfica.

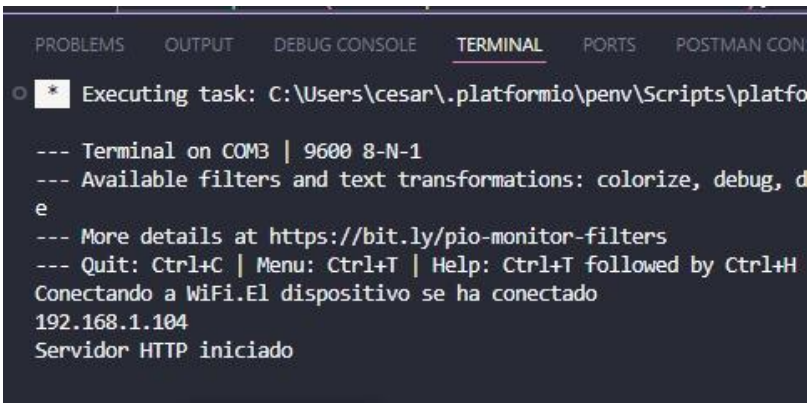


Fig. 5. Captura de la conexión exitosa a wifi de la esp32

4.2 Funcionamiento de los semáforos

Previo a la sincronización, se verificó el funcionamiento individual de cada semáforo. Con base en las reglas definidas en la capa de lógica, se comprobó que cada uno operará correctamente de forma independiente.

Una vez terminamos de armar el circuito de salida del primer semáforo probamos la rutina en la placa de relevadores antes de probarla con los focos.

Durante la prueba, se observó el encendido de los LEDs indicadores integrados en la placa de relevadores, lo cual confirmó la activación de los canales correspondientes. Asimismo, se escuchó el característico clic mecánico de los relevadores al cambiar de estado, lo que evidenció el correcto funcionamiento de la rutina de control desde el ESP32.

Este ejercicio nos facilitó integrar los focos al prototipo con menor riesgo de fallos eléctricos.

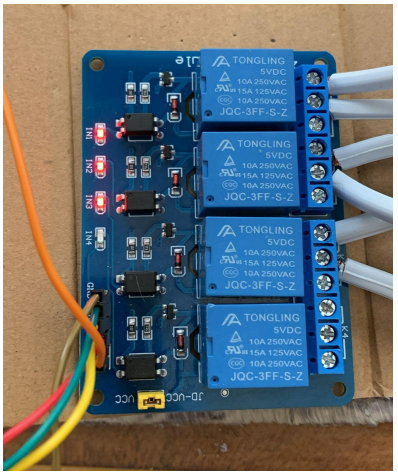


Fig. 6. LEDs indicadores integrados en la placa de relevadores encendidos tras completar satisfactoriamente la rutina.

Asimismo, una vez concluida la implementación del segundo semáforo con LEDs en el protoboard, se comprobó que encendieran en el orden correcto. Debido a que este montaje presentaba menores riesgos eléctricos, las pruebas se realizaron directamente con el prototipo completo.

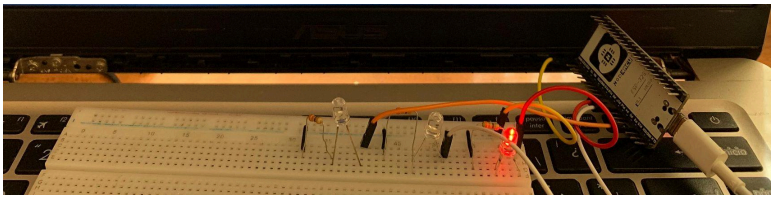


Fig. 7. Prueba de encendido del semáforo de LEDs.

4.3 Integración con la interfaz gráfica

Terminada la interfaz que controla al semáforo físico, pasamos a probar que las peticioneciones HTTP tuvieran las siguientes respuestas:

1. Encender físicamente los focos y leds de los semáforos.

2. Responder al cliente con un mensaje y el código HTTP 200.

NOTA: Para ello fue importante verificar que la PC y la ESP32 estuvieran en la misma red.

5 Resultados

Al finalizar la práctica contamos con 2 maquetas de semáforos realizadas con diferentes componentes electrónicos sincronizados entre sí y con la posibilidad de ser controlados mediante interfaces.

El flujo de la arquitectura resultante comienza con el usuario haciendo peticiones a través de los controladores (botones) y los controladores deslizantes (scales) de la interfaz; esto produce peticiones HTTP GET hacia la IP del ESP32.

En la capa de negocio, el WebServer (puerto 80) enruta cada solicitud con server.on(...) hacia sus handlers, los cuales actualizan el estado en RAM (duracion*, rutinaActiva, fase, parpadeos, tiempoAnterior) y disparan el control de fases del semáforo: una máquina de estados no bloqueante basada en millis() que decide la conmutación de cada color.

La capa de datos y acceso al hardware materializa la acción activando los pines que excitan el módulo de relés y, por su salida COM, energizan los focos; en paralelo, el servidor responde con código 200 OK. Gracias a server.handleClient() en el loop() y al uso de millis() (sin delay()), el sistema mantiene el servicio HTTP mientras corre la secuencia, permitiendo reconfigurar tiempos en caliente y arrancar/detener la rutina sin bloquear la interfaz.

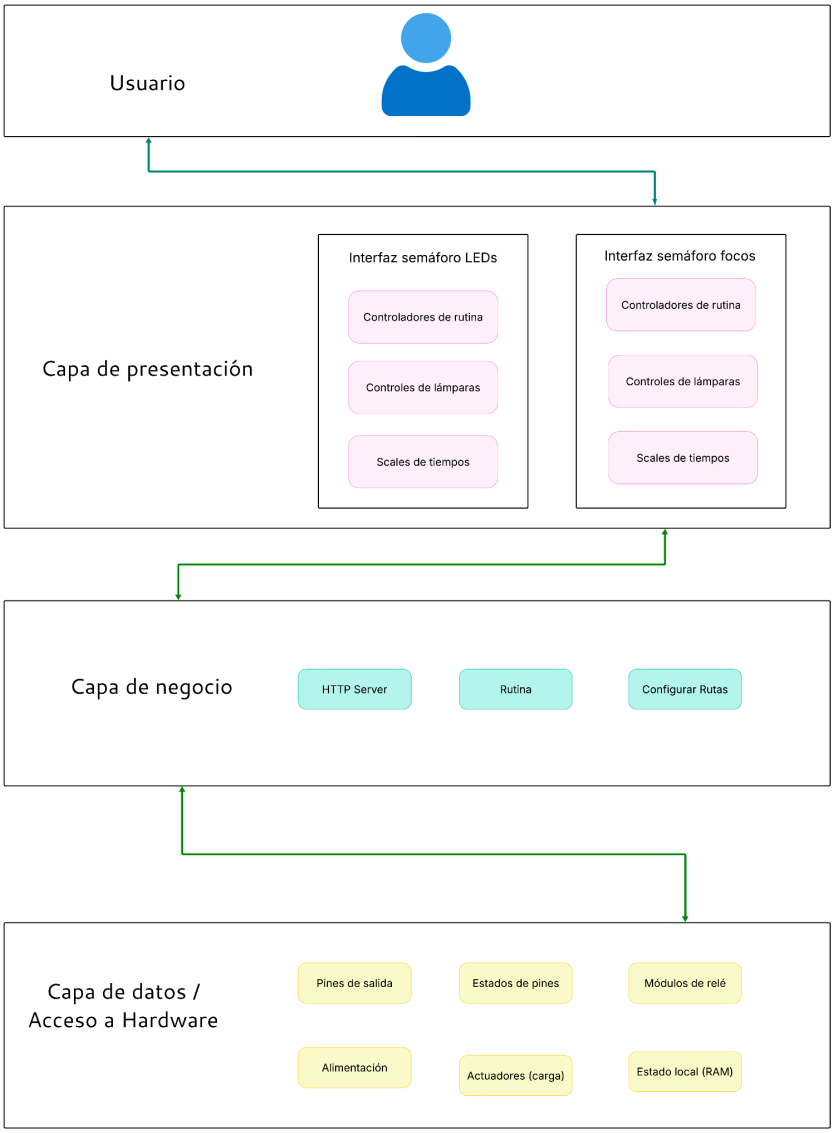


Fig. 8. Arquitectura del semáforo inteligente.

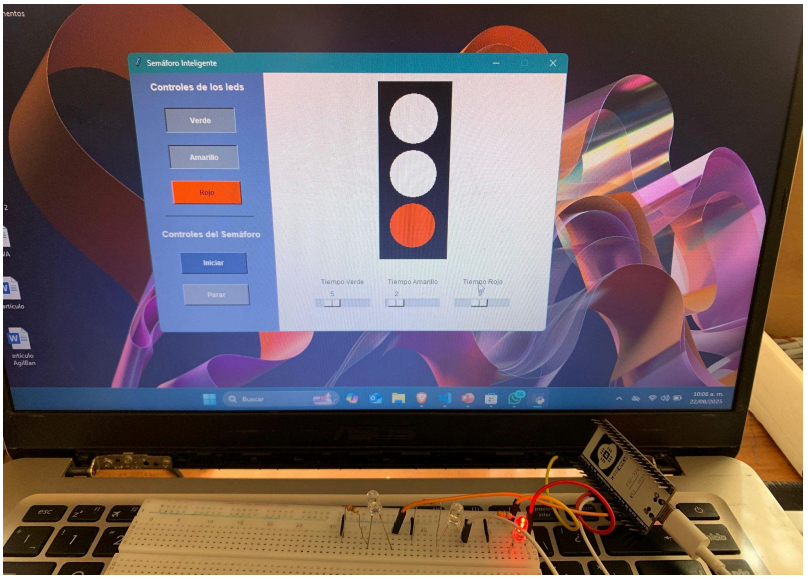


Fig. 9. Interfaz gráfica y semáforo de LEDs funcionando.

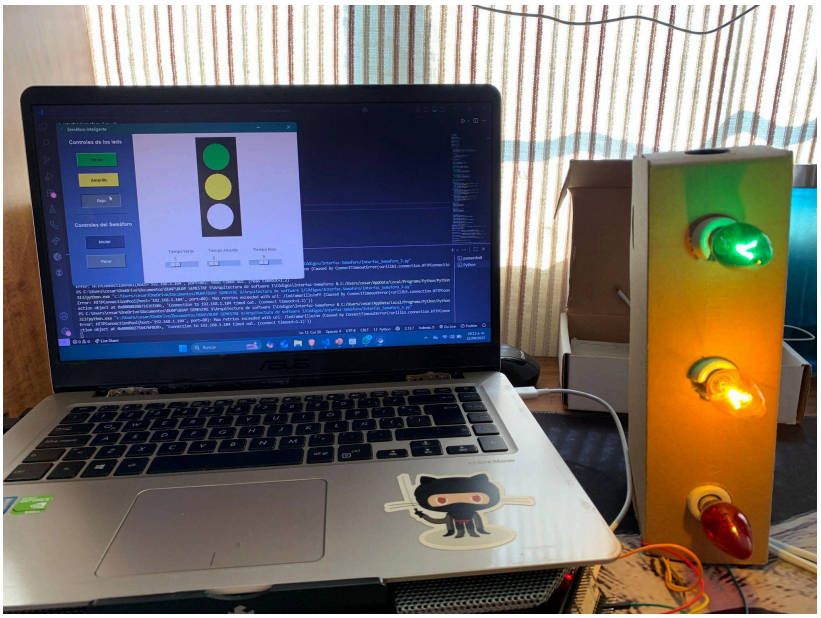


Fig. 10. Interfaz gráfica y semáforo de focos funcionando.

Fig. 11. Prototipo del semáforo de focos.

6 Conclusiones

La implementación de una arquitectura en capas en el desarrollo de un semáforo inteligente ha demostrado ser un método altamente eficaz para gestionar sistemas complejos. Esta estructura modular no solo permitió una clara delimitación de responsabilidades entre las capas, sino que también agilizó las etapas de desarrollo, depuración y expansión del proyecto. La capa de presentación confirmó la factibilidad de integrar componentes de alto voltaje (AC) con un microcontrolador de bajo voltaje (DC), estableciendo un puente sólido entre los dominios físico y digital. La capa de negocio validó la importancia de la programación modular, al habilitar la reutilización de funciones para sincronizar ambos semáforos con precisión, optimizando el rendimiento del sistema. Asimismo, la capa de datos demostró su valor al centralizar la configuración, facilitando la gestión y ajuste de parámetros críticos.

En retrospectiva, este enfoque superó las expectativas iniciales, proporcionando una base técnica robusta y escalable que respalda el éxito del proyecto. La implementación resultante no solo cumplió con los objetivos establecidos, sino que también sentó las bases para futuras innovaciones en sistemas de control embebidos.

7 Trabajos Futuros

Con base en los resultados obtenidos, se identifican varias oportunidades de mejora y expansión:

- *Modo de Emergencia***:** Incorporar un botón físico o digital que active un estado de emergencia, forzando a ambos semáforos a mostrar una luz roja fija durante un período predefinido, mejorando la seguridad en situaciones críticas (López, 2021).
- *Interfaz Web*: Desarrollar una interfaz web accesible a través de la red local, permitiendo el monitoreo remoto y el ajuste de parámetros en tiempo real, lo que ampliaría las capacidades del sistema hacia un entorno IoT [4].
- *Optimización de Energía:* Explorar técnicas de bajo consumo, como el uso de LEDs de alta eficiencia o modos de reposo para el ESP32, para alargar la vida útil del sistema en aplicaciones prolongadas.
- *Integración con Sensores:* Agregar sensores de tráfico para adaptar dinámicamente los tiempos de las luces según la densidad vehicular, incrementando la eficiencia del control.

Estas propuestas refuerzan el potencial del proyecto como una plataforma adaptable para futuras investigaciones y aplicaciones prácticas en ingeniería electrónica y de software.

References

1. García, J. (2020). *Arquitectura de Software: Fundamentos y Prácticas*. Recuperado de https://www.todoelectronica.com.mx/arquitectura-software.
2. Hernández, L. (2023). *Internet de las Cosas en México: Aplicaciones Prácticas*. Recuperado de https://www.electronica360.com.mx/iot-aplicaciones
3. López, M. (2021). *Metodologías Ágiles en Ingeniería de Software*. Recuperado de https://www.sistemas.unam.mx/metodologias-agiles
4. Hernández, L. (2023). *Internet de las Cosas en México: Aplicaciones Prácticas*. Recuperado de https://www.electronica360.com.mx/iot-aplicaciones
5. Yépez Zambrano, K. L. (2024). *Tecnologías de comunicaciones: prototipo para adquisición de datos de un sistema de semáforos para administración de tráfico vehicular en una ciudad inteligente*. Memorias del XXX Congreso Internacional Anual de la SOMIM.
6. Martínez Torres, R. E., Mendoza Razo, J. A., Lin Huang, L., Huerta Hernández, J. A., Ruis Coronado, C. B., Ledesma Elias, S., & Gomez Vega, A. G. (2024). *Sistema de intervención de semáforos para emergencias*. Tecnológico Nacional de México, Instituto Tecnológico de San Luis Potosí.
7. Chuchuca Gómez, A. D. (2024). *Implementación de un prototipo de sistema de alarma residencial basado en ESP32 y página web*. Quito: Escuela Politécnica Nacional.
8. Tonato Chuquimarca, C. E., & Sinche Maita, S. L. (2022). *Análisis comparativo entre arquitecturas de sistemas IoT*. Revista de Investigación en Tecnologías de la Información (RITI), 10(Esp. 21), 55–70. https://doi.org/10.36825/RITI.10.21.006