DISEÑO DE SISTEMAS BASADOS EN MICROPROCESADOR

➔ Práctica 8 Laboratorio:

Esta práctica es continuación de la anterior y su finalidad es controlar el estado del semáforo mediante envío y recepción de mensajes de otro microcontrolador (ESP32) mediante las UARTS. Estas ordenes las controlaremos vía Wi-Fi con un programa escrito en micropython para el ESP32.

Si el semáforo ya está haciendo la secuencia del botón, no aceptará ordenes del ESP32 hasta que la misma termine. Pero si el semáforo está atendiendo ordenes del ESP32, no aceptará la interrupción del botón.

■ El funcionamiento del semáforo debe ser el siguiente:

El paso de vehículos siempre estará en verde, y solo se pondrá en rojo cuando se pulse el botón. En este estado se enviará periódicamente el mensaje *rojo* mediante Bluetooth al dispositivo móvil conectado al emisor y la pantalla LCD estará en rojo mostrando el mensaje *Pulse Botón*. En este caso, tras 3 segundos, se pondrá en verde el paso de peatones y rojo el de vehículos, 15 segundos después parpadeará y, 3 segundos después, el sistema volverá a su estado inicial. La secuencia será:

- Pulsar botón del semáforo.
  - Se comprueba si existen vehículos cercanos al sensor ultrasónico (menos de 400 mm).
  - Si existen vehículos cercanos se esperan los 3 segundos.
  - Si no existen vehículos cercanos no se esperan los 3 segundos.
- Pasar el control de vehículos del verde al amarillo en la calle principal, se envia el mensaje *amarillo* al dispositivo móvil y la pantalla LCD se cambia a color azul y se muestra el mensaje *Espere Verde*.
- Esperar 3 segundos.
- Poner el semáforo de vehículos en rojo y el de peatones en verde, bajar la barrera, además se manda al dispositivo móvil la palabra *verde* y la pantalla LCD pasa a color verde mostrando el mensaje *Pase*.
- Espera 15 segundos.
- Poner semáforo de peatones en verde intermitente.
- Esperar 3 segundos.
- Poner semáforo de peatones en rojo y el principal en verde y subir la barrera.

- **El funcionamiento del semáforo debe ser el siguiente (modo ESP32):**

Esta tabla está sacada del enunciado de la práctica:

Semáforo vehículos	Comando	Acción
Verde	green	Bloquear el pulsador y mantener el estado actual
Rojo	green	Iniciar parpadeo de semáforo peatones. Seguidamente se pasa a verde y se bloquea el pulsador
Ámbar	green	No se pasa a rojo el semáforo de vehículos ni a verde el de peatones, directamente se vuelve a verde vehículos y se bloquea el pulsador
Verde	red	Misma acción que si se pulsara el botón (comprobar si coches cercanos, pasar a ámbar, ...). También se bloquea el pulsador
Rojo	red	Bloquear el pulsador y mantener el estado actual
Ámbar	red	Pasar el semáforo de vehículos a rojo tras el tiempo del ámbar y bloquear el pulsador

➔ Resultados:

➔ Código principal:

➢ Código de la Máquina de estados:

Esta es una de las nuevas características de la práctica, para poder implementar la funcionalidad del ESP32, hemos sacado el antiguo código del while que pertenecía a la máquina de estados y lo hemos metido en un método aparte.

```
void stateMachine()
{
  switch (modo)
  {
  case 1:
    counter_parpadeo = 0;
    if (cercanos == 1)
    {
      HAL_Delay(3000);
    }
    esp32_modo = 2;
    GPIOB->ODR &= ~GPIO_ODR_OD6_Msk;        //Apagar verde coches
    GPIOC->ODR |= GPIO_ODR_OD7_Msk;         //Encender amarillo coches
    HAL_UART_Transmit(&huart1, "Amarillo\n", 9, 1000); //Mandar Espere Verde
    HAL_UART_Transmit(&huart1, 1, 1, 1000);
    HAL_Delay(3000);
```

```
    modo = 2;
    break;
  case 2:
    GPIOC->ODR &= ~GPIO_ODR_OD7_Msk; //Apagar amarillo coches
    GPIOA->ODR |= GPIO_ODR_OD9_Msk;  //Encender rojo coches
    htim2.Instance->CCR1 = 25;
    GPIOA->ODR &= ~GPIO_ODR_OD7_Msk;                //Apagamos rojo peatones
    GPIOA->ODR |= GPIO_ODR_OD6_Msk;                 //Encender verde peatones
    HAL_UART_Transmit(&huart1, "Verde\n", 6, 1000); //Mandar Pase
    HAL_UART_Transmit(&huart1, 2, 1, 1000);
    HAL_Delay(15000);
    if (esp32_int == 1)
    {
      while (data_esp32[1] == '0')
      {
        printf(" \r\n"); //While Delay
      }
    }
    modo = 3;
    esp32_modo = 0;
    break;
  case 3:
    while (counter_parpadeo < 15)
    {
      GPIOA->ODR &= ~GPIO_ODR_OD6_Msk; //Apagar verde peatones
      HAL_Delay(100);
      GPIOA->ODR |= GPIO_ODR_OD6_Msk; //Encender verde peatones
      HAL_Delay(100);
      counter_parpadeo++;
    }
    modo = 4;
    esp32_modo = 0;
    break;
  case 4:
    GPIOA->ODR &= ~GPIO_ODR_OD6_Msk; //Apagar verde peatones
    GPIOA->ODR &= ~GPIO_ODR_OD9_Msk; //Apagar rojo coches
    GPIOB->ODR |= GPIO_ODR_OD6_Msk;  //Encendemos verde coches
    GPIOA->ODR |= GPIO_ODR_OD7_Msk;  //Encendemos rojo peatones
    htim2.Instance->CCR1 = 75;
    modo = 1;
    control = 0;
    pulsado = 0;
    cercanos = 0;
```

```
    times = 0;
    esp32_modo = 1;
    break;
  default:
    htim2.Instance->CCR1 = 75;
    HAL_Delay(1000);
    GPIOC->ODR &= ~GPIO_ODR_OD7_Msk; //Apagar Amarillo coches (Para ESP32)
    GPIOB->ODR |= GPIO_ODR_OD6_Msk;                //Encender Verde Coches
    GPIOA->ODR |= GPIO_ODR_OD7_Msk;                //Encender Rojo Peatones
    HAL_UART_Transmit(&huart1, "Rojo\n", 5, 1000); //Mandar Pulsar Botón
    esp32_modo = 1;
    break;
  }
}
```

Con respecto a la anterior práctica, hemos marcado en azul las nuevas instrucciones que contiene el código. Todas ellas se corresponden con el control de la máquina de estados con ordenes del microcontrolador ESP32.

- ➔ La variable `esp32_modo` nos sirve para saber en que estado se encuentra el semáforo de peatones:
  - **0:** el semáforo está en rojo.
  - **1:** el semáforo está en verde.
  - **2:** el semáforo está en ámbar.
- ➔ El while sirve para quedarnos bloqueados en la instrucción como se indica en la tabla de ordenes hasta que llegue una orden que indique el desbloqueo.
- ➔ Hemos añadido el modo default al while para meter la opción por defecto del semáforo dentro de la máquina de estados.

Código de la Máquina de estados:

```
while (1){
    HAL_UART_Receive_IT(&huart1, data_esp32, sizeof(data_esp32));
    if (esp32_int == 1){
      if (data_esp32[0] == '8'){
        __set_BASEPRI(1);
        if (data_esp32[1] == '0'){
          if (esp32_modo == 0){
            while (data_esp32[1] != '2'){
              printf(" \r\n");
            }
```

```
        }else if (esp32_modo == 1){
          ultrasonidos();
          modo = 1;
          for (int i = 0; i < 4 && data_esp32[1] != '1'; i++){
            stateMachine();
          }
        }else if (esp32_modo == 2){
          modo = 2;
          stateMachine();
        }
      }else if (data_esp32[1] == '1'){
        if (esp32_modo == 1){
          while (data_esp32[1] == '1'){
            printf(" \r\n");
          }
        }else if (esp32_modo == 0){
          modo = 3;
          stateMachine();
        }else if (esp32_modo == 2){
          modo = 4;
          stateMachine();
        }
      }else if (data_esp32[1] == '2'){
        modo = 0;
        stateMachine();
      }
      esp32_int = 0;
    }
  } else{
    __set_BASEPRI(8);
    modo = 0;
    stateMachine();
  }

  if (pulsado == 1){
    modo = 1;
```

```
      control = 1;
      counter_parpadeo = 0;
      ultrasonidos();
      while (control == 1){
        stateMachine();
      }
    }
}
```

Este código se corresponde con el control del semáforo con las ordenes del ESP32.

En primer lugar recibimos el código que viene del ESP32 mediante la activación de las interrupciones.

Él código que nos llega siempre tienen que ser 2 números:

- ➔ En primer lugar un 8, que se corresponde con nuestro grupo de laboratorio.
- ➔ En segundo lugar:
  - 0: Rojo.
  - 1: Verde.
  - 2: Restore.

A partir de estos códigos y el estado en el que se encuentra el semáforo mediante la variable antes explicada, realizamos comprobaciones mediante instrucciones de condición `if` y bloqueos con `while` y `for` dependiendo del funcionamiento deseado a partir de la tabla de instrucciones proporcionada.

- **Código del ESP32:**

```
import picoweb
import wifiConnect
from machine import UART

ip = wifiConnect.connect()
uart = UART(2, 9600)
app = picoweb.WebApp(__name__)
@app.route("/red/8")
def red(req, resp):
    uart.write('80')
    print(uart.read(2))
    yield from picoweb.start_response(resp, content_type="text/html")
    yield from resp.awrite("RED")
    yield from resp.aclose()
@app.route("/green/8")
```

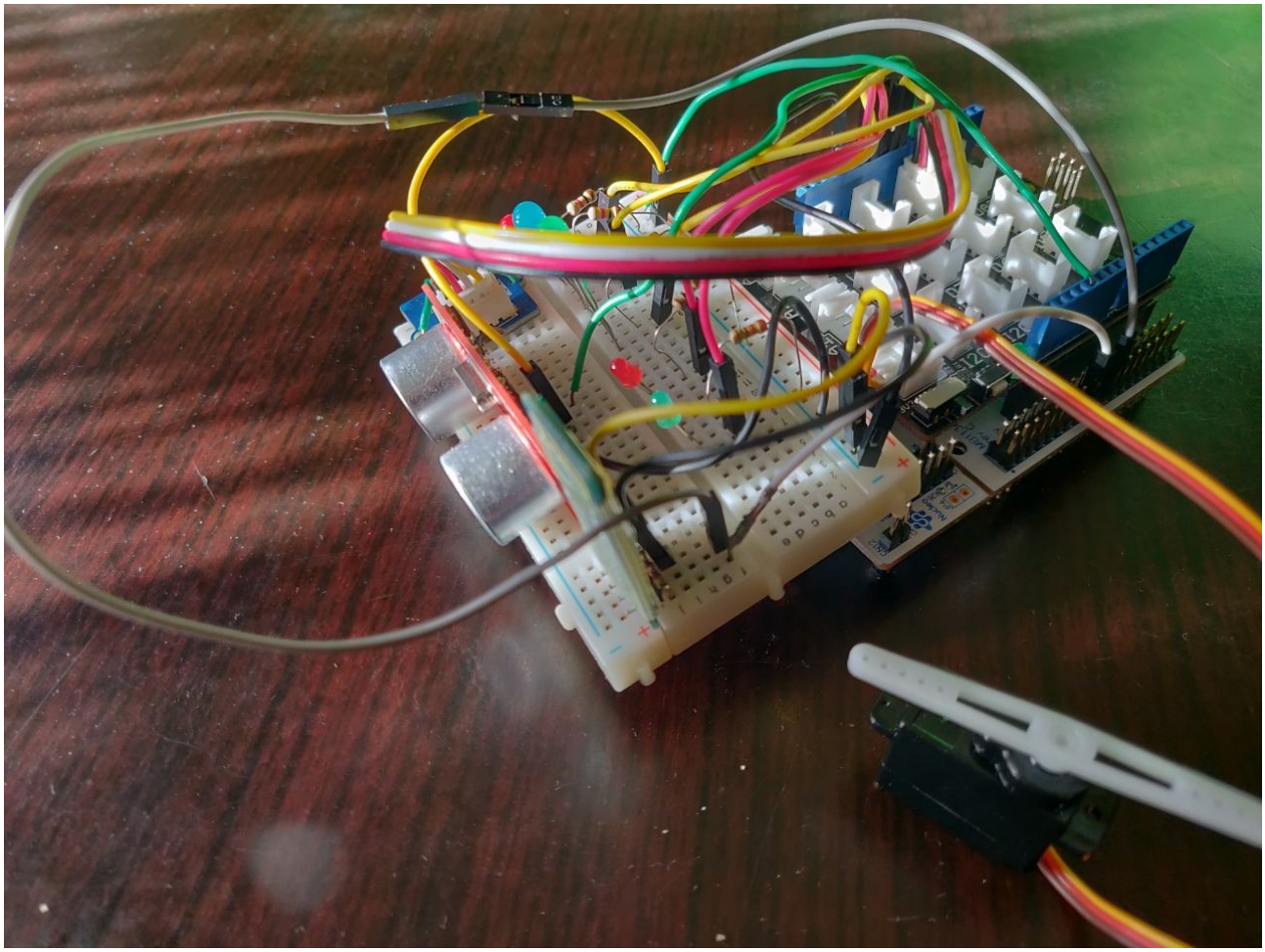
```
def green(req, resp):
    uart.write('81')
    print(uart.read(2))
    yield from picoweb.start_response(resp, content_type="text/html")
    yield from resp.awrite("GREEN")
    yield from resp.aclose()
    pass
@app.route("/restore/8")
def restore(req, resp):
    uart.write('82')
    print(uart.read(2))
    yield from picoweb.start_response(resp, content_type="text/html")
    yield from resp.awrite("RESTORE")
    yield from resp.aclose()
    pass
# if __name__ == '__main__':
app.run(debug=True, host = ip, port = 80)
```

Este código corresponde al utilizado para el envío de ordenes al semáforo con el microcontrolador ESP32.

En azul están marcadas las instrucciones correspondientes al uso de la UART con la que se envían los mensajes.

El resto de instrucciones se utilizan para poder realizar estas ordenes a través de la web.

➔ Imagen del circuito:



Este es el circuito que hemos diseñado para la práctica. Consta de 5 LEDs y un botón que representan la estructura del semáforo. El LED azul representa la luz ámbar y el botón que hemos utilizado ha sido el de la shell.

La configuración de los cables es la siguiente:

- Los cables amarillos son de alimentación para el semáforo de coches.
- Los cables rojos son de alimentación para el semáforo de peatones y para el sensor de ultrasonidos.
- El cable negro es el cable de tierra al que se conectan las resistencias de los LEDs y el sensor de ultrasonidos.
- El cable verde es el que controla la emisión y recepción de señales del sensor de ultrasonidos y otro da corriente al sensor de ultrasonido y al emisor Bluetooth.
- El cable blanco conecta el puerto TX de la placa principal con el RX del emisor Bluetooth.

- El cable gris conecta el RX del emisor Bluetooth con el RX de la placa Arduino (Como un puente para repartir desde el TX de la placa principal a los 2).

Nota: No podemos mostrar la conexión con el ESP32 porque no tenemos el módulo disponible. Tampoco podemos mostrar la conexión con el Arduino y el LCD por el mismo motivo.

➔ En el caso de utilizar el ESP32 para realizar la comunicación con la placa principal conectamos el puerto TX2 del ESP32 con el puerto RX configurado en la placa.