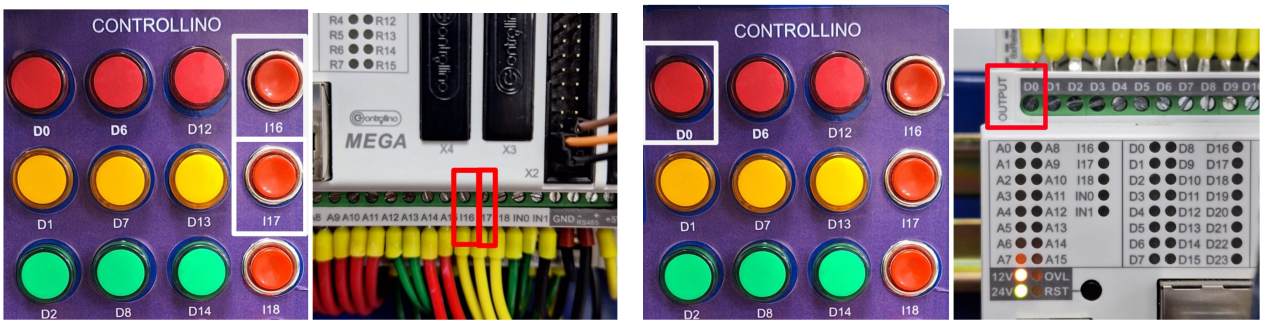
Práctica 5: Control básico de salidas digitales y aplicación avanzada con FSM

Pablo Andrés Bermeo Garcia
Facultad de Ingeniería
Departamento de Telecomunicaciones
Universidad de Cuenca
pablo.bermeo@ucuenca.edu.ec

Tyrone Miguel Novillo Bravo
Facultad de Ingeniería
Departamento de Telecomunicaciones
Universidad de Cuenca
tyrone.novillo@ucuenca.edu.ec

I. Configuración del Hardware



(a) (b)

Figura 1: Se presentan las correspondencias entre lo botones del tablero y sus entradas con respecto al Controllino. Tomado de [1].

En la Figura 1 se presenta la configuración de Hardaware que se empleó durante la práctica. En este caso no se requirió de ningún cableado adicional, pues el tablero ya implementadas todas las uniones. Únicamente se procede a recalcar que se emplearán las entradas del Controllino I16 e I17 para mapear los bótones del tablero, la salida D0 se empla para encender el LED primero rojo de la matriz de luces.

II. Práctica guiada

Para familizarizarse con el empleo de lo bótones y los LED del tablero se proveyó un código básico en la guía de la práctica. En este código guía se resaltan algunas de sus secciones a fin de explicar la lógica detrás de su funcionamiento:

- Empleo de las entradas y salidas propias del Controllino: únicamente se definen los pines que se emplearán para las entradas y salidas del controllino. Note que los nombres de las variables delatan que se pretende generar una lógica de encedido y apagado a través de dos botones: el botón superior apaga el led; el inferior, lo enciende.

```
#include <Controllino.h>
const int led             = CONTROLLINO_D0;
const int boton_encendido = CONTROLLINO_I16;
const int boton_apagado   = CONTROLLINO_I17;
```

- Lógica de encendido y apagado del LED: En este sentido, únicamente se detecta: En esta sección únicamente se verifica el estado de los botones para decidir el próximo estado del LED. En el condicional se considera si el boton para encender ha sido activado y su contrario, no; en este caso, se procede al encendido. En cambio, si únicamente el botón para apagar se se halla activado, se procede a la tarea obvia consiguiente.

```
if (digitalRead(boton_encendido) == HIGH && digitalRead(boton_apagado)==LOW) {
    digitalWrite(led, HIGH);
}
```

```
4
  if (digitalRead(boton_apagado) == HIGH) {
6     digitalWrite(led, LOW);
  }
8
  delay(10);
```

El resultado de esta sección fue exitoso: el LED se prendía ya apagaba una vez que se presionaban los botones correspondientes.

III. Retos a cumplir en el laboratorio

III-A. Parte A

Reto: Utilizar los tres botones del tablero de pruebas para controlar el patrón de encendido de los LED ubicados en forma de matriz 3x3. Tanto los botones como los LED ya se encuentran conectados directamente al Controllino Mega.[1]

- ***Botón 1: Encendido en espiral normal.***
- ***Botón 2: Encendido en espiral inverso.***
- ***Botón 3: Reinicia y apaga todos los LEDs***

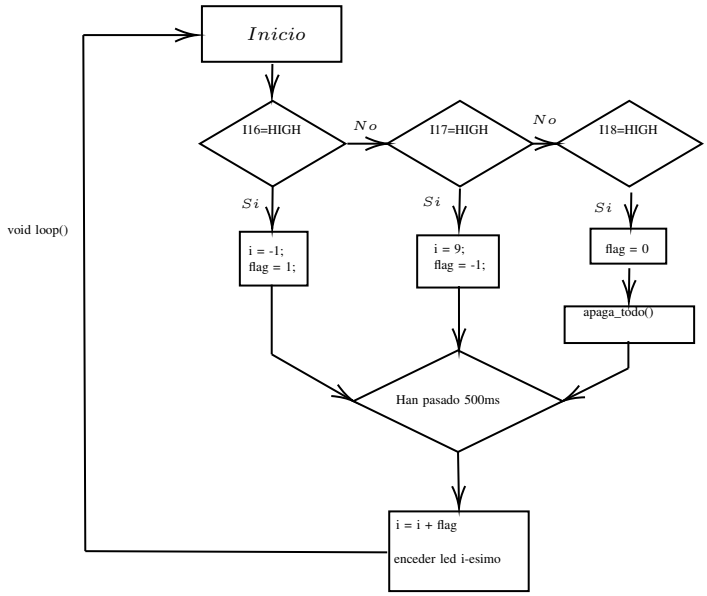


Figura 2: Diagrama de flujo de la solución proupuesta para el reto A.

Solución propuesta: La implementación de la solución se basa en tres pilares: uso de punteros para recorrer los LEDS, bandera para determinar el comportamiento concurrente , y empleo de un índice que aumente o desciende para dictaminar el LED a ser encedido o apagado.

- **Uso de banderas para determinar el comportamiento concurrente:** Se empleó una bandera (*flag*) para determinar si el estado correspondía a una visión en recorrido horario, antihorario o matriz apagada de los LEDS. La lógica consta de que si el botón superior (I16) es aplastado, la bandera toma un valor de 1; si es el medio (I17), la bandera es -1, y en caso de ser el tercero (I18), se asigna el valor de 0.
  Además el valor de la bandera se empleó para incrementar (o decrementar) el valor de índice que recorre los LEDS. Estas lógicas se pueden visualizar en el esquema de la programación presentado en la Figura 2.
  Las decisiones y asiginación de los valores para la bandera se pueden visualizar en la siguiente codificación:

```
1 //Horario
  if (digitalRead(CONTROLLINO_I16) == HIGH) {
3 apaga_todo();
  i = -1;
5 flag = 1;
```

```
t_previo = 0;
}
//Antihorario
if (digitalRead(CONTROLLINO_I17) == HIGH) {
apaga_todo();
i = 9;
flag = -1;
t_previo = 0;
}
//Apagar
if (digitalRead(CONTROLLINO_I18) == HIGH) {
apaga_todo();
flag = 0;
t_previo = 0;
}
```

- **Empleo de puntero para enceder los LEDs:** En la interrupción no bloqueante se aprovecha para recorrer el vector que contiene los pines, ordenados *a priori* para que sigan una espiral, de los LEDS en la matriz. La forma de encender los LEDS es a partir del empleo de punteros, pues se toma el valor de memoria a partir de **(led + i)*. Donde led se define como *int *led = LEDS*. Es decir que led es el espacio de memoria de algun elemento de la matriz. Esto se presenta en la siguiente codificación:

```
int LEDS[9] = {CONTROLLINO_D0,CONTROLLINO_D6,CONTROLLINO_D12,CONTROLLINO_D13,CONTROLLINO_D14,
    CONTROLLINO_D8,CONTROLLINO_D2,CONTROLLINO_D1,CONTROLLINO_D7};
int* led = LEDS;

(...)

if(t_actual - t_previo >= intervalo && flag != 0){

    digitalWrite(*(led + i), LOW);
    t_previo = t_actual;

    //Aumentar el corredor de memoria
    i = i + flag;

    if(i < 0 && flag == -1){
      i = 8;
    }

    //Reiniciar el corredor de memoria
    if(i > 8 && flag == 1){
      i = 0;
    }
    digitalWrite(*(led + i), HIGH);
}
```

III-B. Parte B

Reto:Diseñar un sistema que controle dos semáforos (Semáforo A y Semáforo B) ubicados en una intersección perpendicular (Figura 18), usando el enfoque de máquina de estados finita (FSM). El sistema debe simular el comportamiento simple de los semáforos, de manera que nunca haya luz verde simultánea para ambos sentidos, y se respeten los tiempos estándar de duración de cada luz.[1]

La solución planteada para el reto se basó en cuatro pilares fundamentales: uso de estrucutras para la definición de estados, y los leds correspondientes a las luminarias del semáforo, y definición de cuatro estados a partir de *switch*. En este sentido, se planteó un orden de estados que siguen la lógica de dos semáforos en intersección. A continuación se realiza una descripción de los estados:

- **Uso de estructuras para enlazar los LEDs:** Para este objetivo se empleo una estructura con tres atributos: amarillo, azul y rojo que refieren a los LEDs de cada semáforo. Esto se observa en la siguiente codificación:

```
//Creacion de estructura Semaforo
struct Semaforo {
  int verde;
  int amarillo;
```

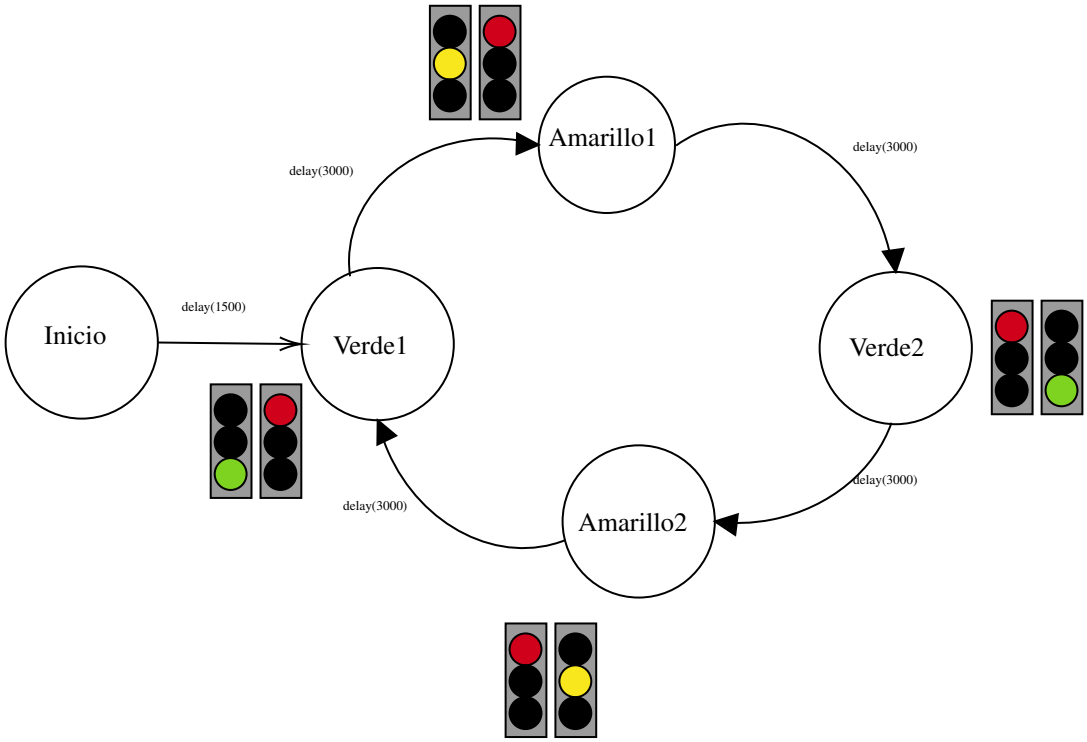


Figura 3: Descripción e ilustración de los estados empleados para la realización de la práctica.

```
  int rojo;
};

// Creacion de semaforos mediante estructura semaforo
Semaforo Semaforo1 = {CONTROLLINO_D2, CONTROLLINO_D1, CONTROLLINO_D0};
Semaforo Semaforo2 = {CONTROLLINO_D8, CONTROLLINO_D7, CONTROLLINO_D6};
```

Definición de los estados a emplear: Se empleo la estrucura *typedef* para englobar a los estados a emplear. A continuación se realiza una descripción de los estados empleados en la práctica:

- Estado *Verde1*: Este estado permite el avance de uno de los carriles y detiene el flujo de su adyancente.
- Estado *Amarillo1*: Este estado empieza el freno del carril concurrente y detiene el flujo de siguiente.
- Estado *Verde2*: Este estado permite el flujo del carril adyacente y frena el carril de observación anterior.
- Estado *Amarillo2*: Este estado empieza el freno del carril adyacente y detiene el flujo del carril de observación anterior.

```
typedef enum{
INICIO,
VERDE_1,
AMARILLO_1,
VERDE_2,
AMARILLO_2,
}estados_t;

estados_t estado_actual;
```

- **Transición de estados mediante *switch*:** Los estados reciben una transición a partir de una estructura switch que decide en función de la variable *estado_actual*. Esto se presenta en la siguiente codificacion:

```
switch(estado_actual){
case INICIO:
    estadoAmarillo = (estadoAmarillo == LOW) ? HIGH : LOW;
    digitalWrite(Semaforo1.amarillo, estadoAmarillo);
    digitalWrite(Semaforo2.amarillo, estadoAmarillo);
    duracion = 500;
```

```
7       break;
  case VERDE_1:
9       digitalWrite(Semaforo1.verde, ENCENDIDO);
        digitalWrite(Semaforo2.rojo, ENCENDIDO);
11      duracion = 5000;
        break;
13      case AMARILLO_1:
        digitalWrite(Semaforo1.amarillo, ENCENDIDO);
15      digitalWrite(Semaforo2.rojo, ENCENDIDO);
        duracion = 5000;
17      break;
  case VERDE_2:
19      digitalWrite(Semaforo1.rojo, ENCENDIDO);
        digitalWrite(Semaforo2.verde, ENCENDIDO);
21      duracion = 5000;
        break;
23  case AMARILLO_2:
        digitalWrite(Semaforo2.amarillo, ENCENDIDO);
25      digitalWrite(Semaforo1.rojo, ENCENDIDO);
        duracion = 5000;
27      break;
  default:
29      break;
  }
31
```

IV. Conclusiones

El empleo de estructuras a partir de *struct* permite el manejo de variables de una forma rápida. Este tipo de variable permite la asignación de atributos a las variables con el tipo correspondiente. En este sentido considere además que las distintas variables que correspondan a la estructura pueden ser caracterizadas a partir de esos atributos. En el caso de esta práctica los dos semáforos se usaron de manera comprensible y *legible* a nivel de codificación.

El uso de los botones implementados en la planta no requiere el uso de estructuras antirebote pues se hayan ya implementadas. En el transcurso de la práctica el uso de codificación simples relacionadas al uso de botones (flanco de subida o detección de estado) y no se detectaron problemas de rebotes. No obstante, no se debe descartar el uso de retardos que permitan bloquear de manera segura cualquier efecto no deseado.

La estructuras de máquina de estado permite realizar un ordenamiento de las estructuras presentes en el código. Este ordenamiento es, además, importante debido a las distintas entradas y comportamientos que se pueden hallar. Por ejemplo, en el caso de la práctica concurrente los estados corresponden a simples encendidos de LEDs. En cambio, estructuras más complejas podrían ser manejados facilmente a través de la definición de estados.

Referencias

[1] Henry Maldonado. *Tableros: Manual de usuario*. Ver. 1. Guía de práctica. Universidad de Cuenca, Departamento de Ingeniería Eléctrica, Electrónica y Telecomunicaciones. Mayo de 2025.

[2] Katsuhiko Ogata. *Ingeniería de control moderna 5 ed.* es. Old Tappan, NJ: Prentice Hall, 2010.

```
// Pr ctica 2 parte a: Encendido y apagado de LEDs mediante dos botones
#include <Controllino.h>  // Librer a para hardware Controlino

// ------------------------------------------------------------------------
// Constantes y variables globales
// ------------------------------------------------------------------------
const unsigned long INTERVALO = 500;  // Intervalo en ms para el cambio de LED
unsigned long t_previo      = 0;      //  ltimo  instante registrado
unsigned long t_actual      = 0;      // Tiempo actual desde millis()

int i    = 0;    //  ndice  del LED activo
int flag = 0;    // Direcci n de secuencia:  1 = adelante, -1 = atr s, 0 = detenido

// Arreglo con pines de los 9 LEDs y puntero para recorrerlos
int LEDS[9] = {
  CONTROLLINO_D0, CONTROLLINO_D6, CONTROLLINO_D12,
  CONTROLLINO_D13, CONTROLLINO_D14, CONTROLLINO_D8,
  CONTROLLINO_D2, CONTROLLINO_D1,  CONTROLLINO_D7
};
int* led = LEDS;  // Puntero al primer elemento de 'LEDS'

// ------------------------------------------------------------------------
// Funci n de setup: configura pines y comunicaci n serial
// ------------------------------------------------------------------------
void setup() {
  // Configurar cada pin de LED como salida
  for (int j = 0; j < 9; j++) {
    pinMode( *(led + j), OUTPUT );
  }

  // Configurar botones como entradas
  pinMode(CONTROLLINO_I16, INPUT);  // Bot n: inicia secuencia hacia adelante
  pinMode(CONTROLLINO_I17, INPUT);  // Bot n: inicia secuencia hacia atr s
  pinMode(CONTROLLINO_I18, INPUT);  // Bot n: reset de secuencia

  Serial.begin(9600);  // Inicializa comunicaci n serial (opcional)
}

// ------------------------------------------------------------------------
// Funci n para apagar todos los LEDs del tablero
// ------------------------------------------------------------------------
void apaga_todo() {
  for (int j = 0; j < 9; j++) {
    digitalWrite(*(led + j), LOW);
  }
}

// ------------------------------------------------------------------------
// Bucle principal: lectura de botones y control de secuencia de LEDs
// ------------------------------------------------------------------------
void loop() {
  // --- Lectura de botones ---
  if (digitalRead(CONTROLLINO_I16) == HIGH) {
    // Bot n 16 presionado: secuencia hacia adelante
    apaga_todo();
    i       = -1;   // Se iniciar  en LED 0 tras incrementar
    flag    =  1;   // Direcci n positiva
    t_previo = 0;   // Reinicia temporizador
  }

  if (digitalRead(CONTROLLINO_I17) == HIGH) {
    // Bot n 17 presionado: secuencia hacia atr s
    apaga_todo();
    i       =  9;   // Se iniciar  en LED 8 tras decrementar
    flag    = -1;   // Direcci n negativa
    t_previo = 0;
  }

  if (digitalRead(CONTROLLINO_I18) == HIGH) {
    // Bot n 18 presionado: detiene y resetea secuencia
    apaga_todo();
    flag    =  0;   // Secuencia detenida
```

```
      t_previo = 0;
74  }

76  delay(10);  // Peque o retardo para evitar rebotes de los botones

78  // --- Control de temporizaci n no bloqueante ---
    t_actual = millis();
80  if (flag != 0 && (t_actual - t_previo >= INTERVALO)) {
      // Apaga el LED actual
82    digitalWrite(*(led + i), LOW);

84    // Actualiza el tiempo de referencia
      t_previo = t_actual;
86
      // Avanza o retrocede el  ndice  seg n 'flag'
88    i += flag;

90    // Ajuste circular del  ndice
      if (i < 0 && flag == -1) {
92      i = 8;    // Salta al  ltimo  LED
      }
94    else if (i > 8 && flag == 1) {
        i = 0;    // Salta al primer LED
96    }

98    // Enciende el siguiente LED
      digitalWrite(*(led + i), HIGH);
100 }
}
```

Code Listing 1: Código correspondiente a la práctica 5 parte a

V-A. Código de parte b

```
1  // Pr ctica 2 parte b: Semaforo con maquina de estados
   #include <Controllino.h>
3
   // Definiciones para estado de salida
5  #define ENCENDIDO HIGH  // Nivel alto: LED encendido
   #define APAGADO   LOW   // Nivel bajo: LED apagado
7
   // Variables para temporizaci n
9  unsigned long t_previo    = 0;  //  ltimo  instante registrado (ms)
   unsigned long t_actual    = 0;  // Tiempo actual (ms)
11 int           duracion    = 0;  // Duraci n del estado actual (ms)

13 // Variables para fase de inicio
   int cont_inicio     = 0;         // Contador de parpadeos inicial
15 int estadoAmarillo  = LOW;       // Estado actual del amarillo en INICIO

17 // Estructura para representar un sem foro (verde, amarillo, rojo)
   struct Semaforo {
19   int verde;
     int amarillo;
21   int rojo;
   };
23
   // Instancias de dos sem foros con pines asignados
25 Semaforo Semaforo1 = { CONTROLLINO_D2, CONTROLLINO_D1, CONTROLLINO_D0 };
   Semaforo Semaforo2 = { CONTROLLINO_D8, CONTROLLINO_D7, CONTROLLINO_D6 };
27
   // Definici n de estados de la m quina
29 typedef enum {
     INICIO,
31   VERDE_1,
     AMARILLO_1,
33   VERDE_2,
     AMARILLO_2
35 } estados_t;

37 estados_t estado_actual;  // Estado actual de la m quina

39 // ------------------------------------------------------------------------
```

```
// setup(): Configura pines y arranca en el estado INICIO
// ------------------------------------------------------------------------
void setup() {
  Serial.begin(9600);  // Inicializa comunicaci n serial (opcional)

  // Configura pines de Semaforo1 como salida
  pinMode(Semaforo1.verde, OUTPUT);
  pinMode(Semaforo1.amarillo, OUTPUT);
  pinMode(Semaforo1.rojo, OUTPUT);

  // Configura pines de Semaforo2 como salida
  pinMode(Semaforo2.verde, OUTPUT);
  pinMode(Semaforo2.amarillo, OUTPUT);
  pinMode(Semaforo2.rojo, OUTPUT);

  // Inicia en el estado INICIO
  cambiar_estado(INICIO);
}

// ------------------------------------------------------------------------
// loop(): Control de la m quina de estados seg n temporizaci n
// ------------------------------------------------------------------------
void loop() {
  t_actual = millis();  // Actualiza tiempo actual

  // Si ha transcurrido la duraci n del estado, pasamos al siguiente
  if (t_actual - t_previo >= duracion) {
    switch (estado_actual) {
      case INICIO:
        // Parpadea amarillo 5 veces antes de pasar a VERDE_1
        if (cont_inicio >= 5) {
          cambiar_estado(VERDE_1);
        } else {
          cont_inicio++;
          cambiar_estado(INICIO);
        }
        break;

      case VERDE_1:
        cambiar_estado(AMARILLO_1);
        break;

      case AMARILLO_1:
        cambiar_estado(VERDE_2);
        break;

      case VERDE_2:
        cambiar_estado(AMARILLO_2);
        break;

      case AMARILLO_2:
        cambiar_estado(VERDE_1);
        break;
    }
  }
}

// ------------------------------------------------------------------------
// cambiar_estado(): Apaga sem foros, actualiza estado y configura LEDs/duraci n
// ------------------------------------------------------------------------
void cambiar_estado(estados_t estado_nuevo) {
  apagarSemaforos();       // Apaga todos los LEDs primero
  t_previo = t_actual;     // Reinicia temporizador
  estado_actual = estado_nuevo;

  switch (estado_actual) {
    case INICIO:
      // Alterna amarillo en ambos sem foros
      estadoAmarillo = (estadoAmarillo == LOW) ? HIGH : LOW;
      digitalWrite(Semaforo1.amarillo, estadoAmarillo);
      digitalWrite(Semaforo2.amarillo, estadoAmarillo);
      duracion = 500;  // 500 ms para parpadeo inicial
      break;

```

```
      case VERDE_1:
        // Semaforo1 en verde, Semaforo2 en rojo
        digitalWrite(Semaforo1.verde, ENCENDIDO);
        digitalWrite(Semaforo2.rojo, ENCENDIDO);
        duracion = 5000;  // 5 s
        break;

      case AMARILLO_1:
        // Semaforo1 en amarillo, Semaforo2 en rojo
        digitalWrite(Semaforo1.amarillo, ENCENDIDO);
        digitalWrite(Semaforo2.rojo, ENCENDIDO);
        duracion = 5000;
        break;

      case VERDE_2:
        // Semaforo1 en rojo, Semaforo2 en verde
        digitalWrite(Semaforo1.rojo, ENCENDIDO);
        digitalWrite(Semaforo2.verde, ENCENDIDO);
        duracion = 5000;
        break;

      case AMARILLO_2:
        // Semaforo1 en rojo, Semaforo2 en amarillo
        digitalWrite(Semaforo1.rojo, ENCENDIDO);
        digitalWrite(Semaforo2.amarillo, ENCENDIDO);
        duracion = 5000;
        break;

      default:
        // Sin acci n para estados no definidos
        break;
    }
  }

  // ------------------------------------------------------------------------
  // apagarSemaforos(): Apaga todos los LEDs de ambos sem foros
  // ------------------------------------------------------------------------
  void apagarSemaforos() {
    // Semaforo1
    digitalWrite(Semaforo1.verde,    APAGADO);
    digitalWrite(Semaforo1.amarillo, APAGADO);
    digitalWrite(Semaforo1.rojo,     APAGADO);

    // Semaforo2
    digitalWrite(Semaforo2.verde,    APAGADO);
    digitalWrite(Semaforo2.amarillo, APAGADO);
    digitalWrite(Semaforo2.rojo,     APAGADO);
  }
```

Code Listing 2: Código correspondiente a la práctica 5 parte b

El siguiente enlace corresponde al repositorio que presenta código, archivos, documentación y evidencia del funcionamiento.

Figura 4: Repositorio de prácticas