Práctica 6: Diseño de interfaz gráfica para el control de salidas en Controllino

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 la pantalla HMI Stone incrustada en el tablero, y los LEDs amarillo (D1) y rojo (D0) de la matriz luminosa.

II. Práctica guiada

A fin de familiarizarse con la conexión y comunicación entre HMI Stone y Controllino, se suministró un código de ejemplo. La tarea del código de ejemplo corresponde a la captura de una valor proveniente una caja de texto y su refejo en uno de los LEDs del Controllino. En concreto, el texto correspondía a un valor de PWM y su reflejo se visualiza en el nivel de brillo de uno de los LEDs.

Del código de ejemplo se resaltan secciones en la que se visualizan la comunicación de Controllino con las varibles presentadas en la pantalla. Para esto, visualice en la codificación conjunta la línea propia de *void setup()* en la que se declara la existencia de una variable de tipo *spin_box* y nombre *spin_box1*. Luego, en la zona del *void_loop()* se halla la instrucción correspondiente *dutyCyclePercent=HMI_get_value* que lee el contenido de *spin_box1* y lo transmite al Controllino.

```
#include "Stone_HMI_Define.h"
(...)
void setup(){
    (...)
```

```
    Stone_HMI_Set_Value("spin_box", "spin_box1", NULL, 0);  // Pone en 0 el valor del spin box en el HMI.
}
(...)
dutyCyclePercent=HMI_get_value("spin_box", "spin_box1"); // Obtiene el valor del spin_box1
```

El resultado final de la práctica guiada fue la variación del brillo del LED D0 a partir de un valor ingresado en la *spin box* que se visualiza en el apartado (a) de la Figura 1.

III. Retos a cumplir en el laboratorio

Reto: Ampliar el programa actual para controlar dos LEDs de forma independiente, utilizando elementos de la interfaz gráfica (spin boxes) para regular el brillo de cada LED y botones físicos del tablero para encender y apagar cada uno de ellos.[1]

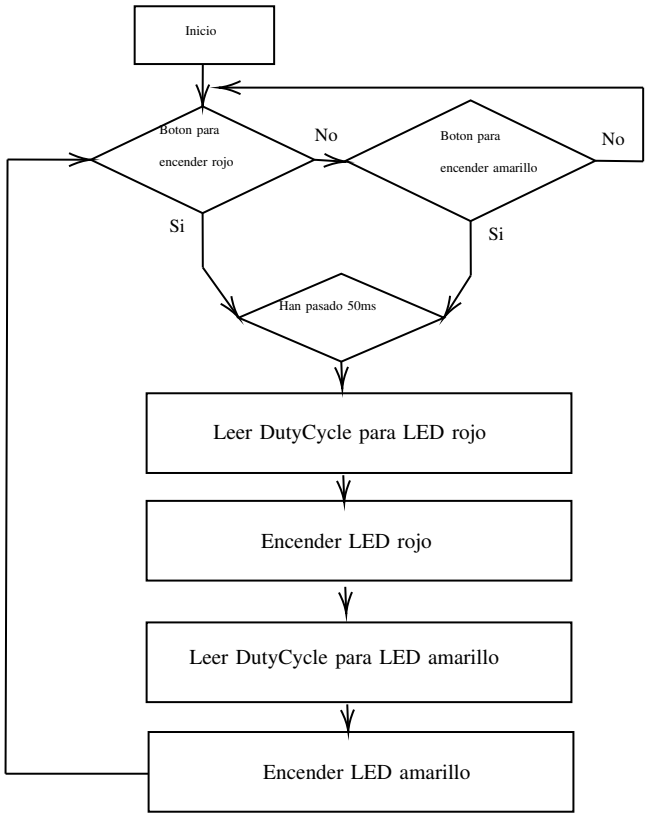

Figura 2: Diagrama de flujo de la solución proupuesta para el reto.

Solución propuesta: La solución propuesta se basa es tres pilares fundamentales: Detección de flanco en los botones I16 y e I17 de la planta, lectura de los valres en los *spinbox*, y negación del estado anterior del boton para apagarlo en caso de estar encendio y encenderlo si está apagado. A continuación se describe cada función junto con su codificación:

- **Detección de flancos en los botones y apagado/encendido de los LEDs:** Lo primero a realizar está relacionado a los flancos de los botones. Esto se hace ya que la detección única de pulso en alto o bajo puede causar problemas. Para generar esta detección se empleó la instrucción condicional junto con dos variables: una que detecta el estado actual del boton y otra que almacena un estado anterior a fin de comparar y detectar cambios. En la codificación estas variables se pueden visualizar como *estadoBoton_rojo* y *estadoAnterior_rojo*. En este sentido, el resultado de la comparación tiene un efecto sobre el encendido de uno de los LEDs (LED rojo, en este caso). La instrucción que se encuentra dentro del condicional permite generar el apagado en caso de encendido y su acción inversa.

```
    (...)
int estadoBoton_rojo = HIGH;
int estadoAnterior_rojo = HIGH;
    (...)
```

```
estadoBoton_rojo = digitalRead(CONTROLLINO_I16);
if (estadoAnterior_rojo == HIGH && estadoBoton_rojo == LOW) {
    led_1_encendido = (led_1_encendido == 0) ? 1 : 0;  // Alternar estado
}

  (...)

estadoAnterior_rojo = estadoBoton_rojo;
```

Esta lógica puede verse ilustrada en la Figura 2. Note como los valores PWM se reflejarán en los LEDs únicamente en caso de que se encuentren encendidos.

- **Léctura de los valores de *duty cycle* a partir de los *spinbox*:**En las siguientes líneas de código se visualiza la captura de datos de las *spinbox* a partir de la función *HMI_get_value*. Dados estos valores se procede a iluminar los LEDs con la intensidad que dicte el valor correspondiente.

```
    (...)
dutyCyclePercent_rojo = HMI_get_value("spin_box", "spin_box1");
dutyCyclePercent_amarillo = HMI_get_value("spin_box", "spin_box2");
    (...)
if (dutyCyclePercent_rojo >= 0 && dutyCyclePercent_rojo <= 100 && led_1_encendido == 1) {
      pwmValue_rojo = map(dutyCyclePercent_rojo, 0, 100, 0, 255);  // Convertir a valor PWM
      analogWrite(led_rojo, pwmValue_rojo);
      (...)
```

IV. HMI

A fin de controlar el ciclo de trabajo de los LEDs amarillo y rojo, se ha implementado en la pantalla Stone la interfaz humano máquina que muestra la Figura 3

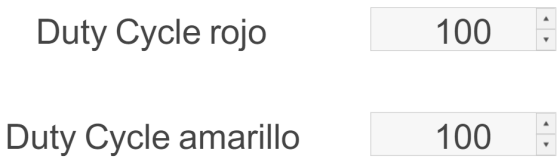

Figura 3: HMI implementado para control de duty cycle

Esta HMI fue diseñada en el software del mismo desarrollador de la pantalla táctil implementada en el tablero. La interfaz está conformada de dos labels que sirven para indicar el LED a modificar y dos objetos del tipo spin-box. El spin box que controla al LED rojo es `spin_box1` y para el amarillo es `spin_box2`. Ambos objetos están seteados como valores enteros con valor mínimo 0 y valor máximo 100 como muestra la Figura 4.

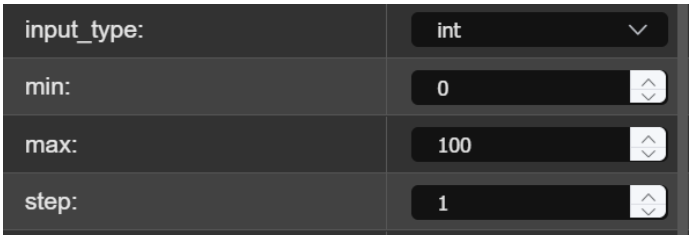

Figura 4: Configuración de los objetos del tipo spin box desde el software STONE Designer GUI.

Una vez cargada la interfaz a la pantalla, el Controllino es capaz de comunicarse mediante las librerías proporcionadas para la práctica. Ahora bien, para lograr intercambio de datos entre la pantalla táctil (HMI) con el programa de Controllino, se deben importar las siguientes librerías:

```
#include "Stone_HMI_Define.h"        // Definiciones para la pantalla tactil HMI Stone
#include "Procesar_HMI.h"            // Funciones de procesamiento para comunicacion con HMI
```

Luego, en la función `setup()`, se habilita la comunicación serial con el HMI a una velocidad de 115200 baudios, velocidad definida en la creación del proyecto. Luego, se inicializa la pantalla y se setea los valores de los objetos spin box en cero. Cabe mencionar que a dichos objetos se los identifica por el identificador o nombre con los que se configuró en el sotfware Designer GUI.

```
void setup() {
  Serial2.begin(115200);             // Comunicacion serial con la pantalla HMI

  HMI_init();                        // Inicializa sistema de comunicacion con la HMI

  // Inicializa los spin boxes de la HMI en cero
  Stone_HMI_Set_Value("spin_box", "spin_box1", NULL, 0);
  Stone_HMI_Set_Value("spin_box", "spin_box2", NULL, 0);
}
```

La función `HMI_get_value(widget_type, widget_id)` es la interfaz con la pantalla táctil: solicita el valor numérico actual de un control (aquí, dos “spin boxes”). Devuelve un entero que refleja el porcentaje que el usuario ha seleccionado (0–100). Luego se ejecutan la siguiente lógica:

- Se comprueba que el porcentaje esté entre 0 y 100 y que el LED esté activo (led_1_encendido == 1).
- La función `map(x, in_min, in_max, out_min, out_max)` convierte el rango de entrada (0–100 %) al rango de salida 0–255, que es el valor válido para analogWrite.
- `analogWrite(pin, value)` genera en el pin una señal PWM de ciclo de trabajo proporcional a value.
- Si el valor no es válido o el LED está apagado, `analogWrite(pin, 0)` lo apaga por completo.

```
// Ejecutar cada 50 milisegundos
  if (t_actual - t_previo >= 50) {
    t_previo = t_actual;

    // Obtener valores actualizados desde la HMI
    dutyCyclePercent_rojo = HMI_get_value("spin_box", "spin_box1");
    dutyCyclePercent_amarillo = HMI_get_value("spin_box", "spin_box2");

    // Control LED rojo
    if (dutyCyclePercent_rojo >= 0 && dutyCyclePercent_rojo <= 100 && led_1_encendido == 1) {
      pwmValue_rojo = map(dutyCyclePercent_rojo, 0, 100, 0, 255);  // Convertir a valor PWM
      analogWrite(led_rojo, pwmValue_rojo);  // Aplicar PWM

    } else {
      analogWrite(led_rojo, 0);  // Apagar si el valor no es valido o el LED esta apagado
    }

    // Control LED amarillo
    if (dutyCyclePercent_amarillo >= 0 && dutyCyclePercent_amarillo <= 100 && led_2_encendido == 1) {
      pwmValue_amarillo = map(dutyCyclePercent_amarillo, 0, 100, 0, 255);
      analogWrite(led_amarillo, pwmValue_amarillo);

    } else {
      analogWrite(led_amarillo, 0);  // Apagar si el valor no es valido o el LED esta apagado
    }
  }
```

V. Conclusiones

Es necesaria la implementación de detecciones de flanco a fin de evitar detecciones continuas de errores. A la hora de realizar el trabajo sucedió que un contacto continuo con uno de los botones provocaba una detecciónо ambigua y por lo tanto fluctuante del estado del boton: el LED correspondiente se encendía y apagaba continuamente. En cambio, la detección de

flanco compara estados anteriores y por lo tanto un contaco contínuo no generó ningún tipo de problema.

Existe cierto retardo entre la comunicación de la pantalla y el Controllino. Cuando se realizaba la interacción con los pusadores de la planta se observó que existía cierto tiempo entre la actuación del botón y su interacción. Esto, según la explicación del docente técnico, se debe a un retraso definido por el fabricante del HMI Stone y por lo tanto es normal. Cabe recalcar que este aspecto no tuvo mayor efecto en las detecciones de flanco.

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.

VI. Anexos

```
#include <Controllino.h>              // Librer a para controlar pines del PLC CONTROLLINO
#include "Stone_HMI_Define.h"         // Definiciones para la pantalla t ctil HMI Stone
#include "Procesar_HMI.h"             // Funciones de procesamiento para comunicaci n con HMI

// Definici n de pines de salida para los LEDs
const int led_rojo = CONTROLLINO_D0;
const int led_amarillo = CONTROLLINO_D1;

// Variables de PWM para controlar brillo
int pwmValue_rojo = 100;
int pwmValue_amarillo = 100;

// Ciclos de trabajo expresados en porcentaje (0-100%)
float dutyCyclePercent_rojo = 100;
float dutyCyclePercent_amarillo = 100;

// Banderas para indicar si los LEDs est n encendidos o apagados
int led_1_encendido = 0;
int led_2_encendido = 0;

// Variables de temporizaci n para control de actualizaci n
unsigned long t_previo = 0;
unsigned long t_actual = 0;

// Variables para detectar flancos de bajada en los botones
int estadoBoton_rojo = HIGH;
int estadoAnterior_rojo = HIGH;
int estadoBoton_amarillo = HIGH;
int estadoAnterior_amarillo = HIGH;

void setup() {
  Serial.begin(115200);               // Comunicaci n serial con el PC
  Serial2.begin(115200);              // Comunicaci n serial con la pantalla HMI

  pinMode(led_rojo, OUTPUT);          // LED rojo como salida
  pinMode(led_amarillo, OUTPUT);      // LED amarillo como salida

  HMI_init();                         // Inicializa sistema de comunicaci n con la HMI

  // Inicializa los spin boxes de la HMI en cero
  Stone_HMI_Set_Value("spin_box", "spin_box1", NULL, 0);
  Stone_HMI_Set_Value("spin_box", "spin_box2", NULL, 0);
}

void loop() {
  // Lectura de botones conectados a entradas digitales
  estadoBoton_rojo = digitalRead(CONTROLLINO_I16);
  estadoBoton_amarillo = digitalRead(CONTROLLINO_I17);

  // Detecci n de flanco de bajada (pulsaci n) en bot n rojo
  if (estadoAnterior_rojo == HIGH && estadoBoton_rojo == LOW) {
    led_1_encendido = (led_1_encendido == 0) ? 1 : 0;  // Alternar estado
    t_previo = 0;  // Reiniciar temporizador
  }

  // Detecci n de flanco de bajada (pulsaci n) en bot n amarillo
  if (estadoAnterior_amarillo == HIGH && estadoBoton_amarillo == LOW) {
    led_2_encendido = (led_2_encendido == 0) ? 1 : 0;  // Alternar estado
    t_previo = 0;  // Reiniciar temporizador
  }

  // Guardar estado actual para comparaci n futura
  estadoAnterior_rojo = estadoBoton_rojo;
  estadoAnterior_amarillo = estadoBoton_amarillo;

  t_actual = millis();  // Actualizar tiempo

  // Ejecutar cada 50 milisegundos
  if (t_actual - t_previo >= 50) {
    t_previo = t_actual;

    // Obtener valores actualizados desde la HMI
```

```
    dutyCyclePercent_rojo = HMI_get_value("spin_box", "spin_box1");
    dutyCyclePercent_amarillo = HMI_get_value("spin_box", "spin_box2");

    // Control LED rojo
    if (dutyCyclePercent_rojo >= 0 && dutyCyclePercent_rojo <= 100 && led_1_encendido == 1) {
      pwmValue_rojo = map(dutyCyclePercent_rojo, 0, 100, 0, 255);  // Convertir a valor PWM
      analogWrite(led_rojo, pwmValue_rojo);  // Aplicar PWM

      Serial.print(dutyCyclePercent_rojo);
      Serial.print(" -> PWM value: ");
      Serial.println(pwmValue_rojo);
    } else {
      Serial.println("Ingresa un valor entre 0 y 100.");
      analogWrite(led_rojo, 0);  // Apagar si el valor no es v lido o el LED est  apagado
    }

    // Control LED amarillo
    if (dutyCyclePercent_amarillo >= 0 && dutyCyclePercent_amarillo <= 100 && led_2_encendido == 1) {
      pwmValue_amarillo = map(dutyCyclePercent_amarillo, 0, 100, 0, 255);
      analogWrite(led_amarillo, pwmValue_amarillo);

      Serial.print(dutyCyclePercent_amarillo);
      Serial.print(" -> PWM value: ");
      Serial.println(pwmValue_amarillo);
    } else {
      Serial.println("Ingresa un valor entre 0 y 100.");
      analogWrite(led_amarillo, 0);  // Apagar si el valor no es v lido o el LED est  apagado
    }
  }
}
```

Code Listing 1: Código correspondiente a la práctica 6

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

Figura 5: Repositorio de prácticas