UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

UNIDAD ACADÉMICA MULTIDISCIPLINARIA ZONA MEDIA UASLP

Nombre de la práctica: PRÁCTICA 14: Sistema de cruce peatonal

Gustavo Tello Rangel.
Clave del alumno: 325454

05/05/2025

Nombre del docente: Ing. Luis Jesús Padrón Padrón

Introducción
El sistema plantea una manera de elaborar un semáforo de doble vía, un elemento ampliamente utilizado en los sistemas de vialidad.

Desarrollo
Lo principal dentro de la planeación fue escoger la manera de mostrar los símbolos peatonales. Al final, me decanté por usar una matriz de LEDs de 8x8.
A partir de ahí, simplemente utilicé LEDs para representar los semáforos vehiculares, así como una plataforma de papel cascarón, además de alambre y soldadura.

Main.c

```
#include "pico/stdlib.h"

#include "hardware/gpio.h"

#include "hardware/timer.h"

#include "max7219.h"

#include <stdbool.h>

#include <string.h>


// Definición de pines

#define SEM1_RED 2

#define SEM1_YELLOW 3

#define SEM1_GREEN 4

#define SEM2_RED 5

#define SEM2_YELLOW 6

#define SEM2_GREEN 7

#define BTN_PEATONAL1 8

#define BTN_PEATONAL2 9

#define DIN1 10

#define CS1 11

#define CLK1 12

#define DIN2 13

#define CS2 14

#define CLK2 15
```

```
// Tiempos en milisegundos
#define VEHICLE_GREEN_TIME   7000   // 7 segundos (modificado de 10 a 7)
#define YELLOW_TIME          3000   // 3 segundos
#define PEDESTRIAN_GREEN     10000  // 10 segundos
#define BLINK_INTERVAL       500    // 500 ms
#define BUTTON_CHECK_INTERVAL 10    // 10 ms
#define COUNT_UPDATE_INTERVAL 1000  // 1 segundo para actualizar contador

typedef enum { VERDE, AMARILLO, ROJO } semaforo_estado;

// Variables de estado
volatile bool btn1_pressed = false;
volatile bool btn2_pressed = false;
int tiempo_restante = 7;  // Inicialmente 7 segundos (verde)
semaforo_estado estado_actual = VERDE;
bool semaforo1_activo = true;
bool paso_peatonal1 = false;
bool paso_peatonal2 = false;
bool mostrando_tiempo_matriz1 = false;  // Indica si la matriz 1 está mostrando tiempo
bool mostrando_tiempo_matriz2 = false;  // Indica si la matriz 2 está mostrando tiempo
absolute_time_t tiempo_actualizacion_contador;

// Variables para control peatonal
volatile bool ped1_request = false;
volatile bool ped2_request = false;
absolute_time_t pedestrian_start_time;
bool pedestrian_blink_state = false;
```

```
// Prototipos de funciones
void btn1_isr(uint gpio, uint32_t events);
void btn2_isr(uint gpio, uint32_t events);
void init_gpio();
void semaforo_vehicular(semaforo_estado estado, bool semaforo1);
void mostrar_matriz(uint8_t *patron, bool matriz1);
void mostrar_numero(int numero, bool matriz1);
void actualizar_peatonales();
void iniciar_cuenta_regresiva(bool matriz1);  // Corrección:
añadido parámetro bool matriz1
void pedestrian_green_phase(bool is_ped1);
void actualizar_cuenta_regresiva();

// Patrones para la matriz
uint8_t circulo[8] = {
    0b00111100,
    0b01111110,
    0b11111111,
    0b11111111,
    0b11111111,
    0b11111111,
    0b01111110,
    0b00111100
};

uint8_t flecha_der[8] = {
    0b00011000,
    0b00011000,
    0b01111110,
    0b00111100,
```

```
    0b00011000,
    0b00000000,
    0b00000000,
    0b00000000
};

uint8_t flecha_izq[8] = {
    0b00011000,
    0b00011000,
    0b00111100,
    0b01111110,
    0b00011000,
    0b00000000,
    0b00000000,
    0b00000000
};

uint8_t numeros[10][8] = {
    {0x3C, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x3C}, // 0
    {0x08, 0x18, 0x28, 0x08, 0x08, 0x08, 0x08, 0x3E}, // 1
    {0x3C, 0x42, 0x02, 0x04, 0x18, 0x20, 0x40, 0x7E}, // 2
    {0x3C, 0x42, 0x02, 0x1C, 0x02, 0x02, 0x42, 0x3C}, // 3
    {0x04, 0x0C, 0x14, 0x24, 0x44, 0x7E, 0x04, 0x04}, // 4
    {0x7E, 0x40, 0x40, 0x7C, 0x02, 0x02, 0x42, 0x3C}, // 5
    {0x3C, 0x42, 0x40, 0x7C, 0x42, 0x42, 0x42, 0x3C}, // 6
    {0x7E, 0x02, 0x04, 0x08, 0x10, 0x10, 0x10, 0x10}, // 7
    {0x3C, 0x42, 0x42, 0x3C, 0x42, 0x42, 0x42, 0x3C}, // 8
    {0x3C, 0x42, 0x42, 0x42, 0x3E, 0x02, 0x42, 0x3C}  // 9
};
```

```
void init_gpio() {
    // Configurar semáforos vehiculares
    for(int i = 2; i <= 7; i++) {
        gpio_init(i);
        gpio_set_dir(i, GPIO_OUT);
    }

    // Configurar botones
    gpio_init(BTN_PEATONAL1);
    gpio_init(BTN_PEATONAL2);
    gpio_set_dir(BTN_PEATONAL1, GPIO_IN);
    gpio_set_dir(BTN_PEATONAL2, GPIO_IN);
    gpio_pull_up(BTN_PEATONAL1);
    gpio_pull_up(BTN_PEATONAL2);

    // Configurar interrupciones
    gpio_set_irq_enabled_with_callback(BTN_PEATONAL1,
GPIO_IRQ_EDGE_FALL, true, &btn1_isr);
    gpio_set_irq_enabled_with_callback(BTN_PEATONAL2,
GPIO_IRQ_EDGE_FALL, true, &btn2_isr);
}

void check_pedestrian_buttons() {
    if (!gpio_get(BTN_PEATONAL1)) ped1_request = true;
    if (!gpio_get(BTN_PEATONAL2)) ped2_request = true;
}

void semaforo_vehicular(semaforo_estado estado, bool semaforo1) {
    // Apagar todos los LEDs
    for(int i = 2; i <= 7; i++) gpio_put(i, 0);
```

```
if(semaforo1) {
    switch(estado) {
        case VERDE:
            gpio_put(SEM1_GREEN, 1);
            gpio_put(SEM2_RED, 1);
            break;
        case AMARILLO:
            gpio_put(SEM1_YELLOW, 1);
            gpio_put(SEM2_RED, 1);
            break;
        case ROJO:
            gpio_put(SEM1_RED, 1);
            gpio_put(SEM2_GREEN, 1);
            break;
    }
} else {
    switch(estado) {
        case VERDE:
            gpio_put(SEM2_GREEN, 1);
            gpio_put(SEM1_RED, 1);
            break;
        case AMARILLO:
            gpio_put(SEM2_YELLOW, 1);
            gpio_put(SEM1_RED, 1);
            break;
        case ROJO:
            gpio_put(SEM2_RED, 1);
            gpio_put(SEM1_GREEN, 1);
            break;
    }
```

```
    }
}

void mostrar_matriz(uint8_t *patron, bool matriz1) {
    max7219_display(matriz1 ? CLK1 : CLK2,
                    matriz1 ? DIN1 : DIN2,
                    matriz1 ? CS1 : CS2,
                    patron);
}

void mostrar_numero(int numero, bool matriz1) {
    if(numero < 0 || numero > 9) return;
    mostrar_matriz(numeros[numero], matriz1);
}

void iniciar_cuenta_regresiva(bool matriz1) {  // Corrección: definición con parámetro bool matriz1
    // Iniciar la cuenta regresiva solo en la matriz correspondiente
    if (matriz1) {
        mostrando_tiempo_matriz1 = true;
        mostrar_numero(tiempo_restante, true);
    } else {
        mostrando_tiempo_matriz2 = true;
        mostrar_numero(tiempo_restante, false);
    }

    tiempo_actualizacion_contador = make_timeout_time_ms(COUNT_UPDATE_INTERVAL); // Actualizar cada segundo
}
```

```
void actualizar_cuenta_regresiva() {
    // Verificar si es tiempo de actualizar el contador
    if (time_reached(tiempo_actualizacion_contador)) {
        tiempo_actualizacion_contador =
make_timeout_time_ms(COUNT_UPDATE_INTERVAL);


        // Mostrar el tiempo restante actualizado solo en las
matrices que están mostrando tiempo
        if (mostrando_tiempo_matriz1) {
            mostrar_numero(tiempo_restante, true);
        }
        if (mostrando_tiempo_matriz2) {
            mostrar_numero(tiempo_restante, false);
        }
    }


    // La cuenta regresiva continúa hasta que termine el ciclo
completo
    // Si el tiempo llega a cero, resetear las matrices
correspondientes
    if (tiempo_restante <= 0) {
        if (mostrando_tiempo_matriz1) {
            mostrando_tiempo_matriz1 = false;
            mostrar_matriz(circulo, true);
        }
        if (mostrando_tiempo_matriz2) {
            mostrando_tiempo_matriz2 = false;
            mostrar_matriz(circulo, false);
        }
    }
}
```

```
void pedestrian_green_phase(bool is_ped1) {
    absolute_time_t start_time = get_absolute_time();
    absolute_time_t blink_time =
make_timeout_time_ms(BLINK_INTERVAL);
    pedestrian_blink_state = true;


    // Mostrar flecha estática durante toda la fase de paso
peatonal
    if(is_ped1) {
        mostrar_matriz(flecha_izq, true);
        mostrar_matriz(circulo, false);
    } else {
        mostrar_matriz(flecha_der, false);
        mostrar_matriz(circulo, true);
    }


    int tiempo_peatonal = PEDESTRIAN_GREEN / 1000;
    absolute_time_t tiempo_actualizacion =
make_timeout_time_ms(COUNT_UPDATE_INTERVAL);


    while(absolute_time_diff_us(start_time, get_absolute_time()) <
PEDESTRIAN_GREEN * 1000) {
        check_pedestrian_buttons();


        // Actualizar la cuenta regresiva para la otra matriz (la
que no muestra la flecha)
        if (time_reached(tiempo_actualizacion)) {
            tiempo_actualizacion =
make_timeout_time_ms(COUNT_UPDATE_INTERVAL);
            tiempo_peatonal--;


            // Solo mostrar tiempo en la matriz que no tiene la
flecha
```

```
            if (is_ped1) {
                mostrar_numero(tiempo_peatonal > 0 ?
tiempo_peatonal : 0, false);
            } else {
                mostrar_numero(tiempo_peatonal > 0 ?
tiempo_peatonal : 0, true);
            }
        }

        // Parpadeo de la flecha después de la mitad del tiempo
        if(absolute_time_diff_us(start_time, get_absolute_time())
> (PEDESTRIAN_GREEN/2) * 1000) {
            if(time_reached(blink_time)) {
                pedestrian_blink_state = !pedestrian_blink_state;
                blink_time = make_timeout_time_ms(BLINK_INTERVAL);

                if(is_ped1) {
                    if(pedestrian_blink_state) {
                        mostrar_matriz(flecha_izq, true);
                    } else {
                        mostrar_matriz(circulo, true);
                    }
                } else {
                    if(pedestrian_blink_state) {
                        mostrar_matriz(flecha_der, false);
                    } else {
                        mostrar_matriz(circulo, false);
                    }
                }
            }
        }
```

```
        sleep_ms(BUTTON_CHECK_INTERVAL);
    }

    // Restablecer
    if(is_ped1) {
        ped1_request = false;
        mostrar_matriz(circulo, true);
    } else {
        ped2_request = false;
        mostrar_matriz(circulo, false);
    }
}

void actualizar_peatonales() {
    // Primero actualizar las cuentas regresivas si están activas
    actualizar_cuenta_regresiva();

    // Para la matriz 1, si no está mostrando tiempo, mostrar el
patrón normal o procesar solicitud
    if (!mostrando_tiempo_matriz1) {
        if (!ped1_request) {
            mostrar_matriz(circulo, true);
        } else if (!semaforo1_activo && estado_actual == VERDE) {
            pedestrian_green_phase(true);
        }
    }

    // Para la matriz 2, si no está mostrando tiempo, mostrar el
patrón normal o procesar solicitud
    if (!mostrando_tiempo_matriz2) {
```

```
        if (!ped2_request) {
            mostrar_matriz(circulo, false);
        } else if (semaforo1_activo && estado_actual == VERDE) {
            pedestrian_green_phase(false);
        }
    }
}

// Implementación de las ISRs
void btn1_isr(uint gpio, uint32_t events) {
    ped1_request = true;
    iniciar_cuenta_regresiva(true); // Iniciar animación con
cuenta regresiva solo en matriz 1
}

void btn2_isr(uint gpio, uint32_t events) {
    ped2_request = true;
    iniciar_cuenta_regresiva(false); // Iniciar animación con
cuenta regresiva solo en matriz 2
}

int main() {
    stdio_init_all();
    init_gpio();

    // Inicializar matrices LED
    max7219_init(CLK1, DIN1, CS1);
    max7219_init(CLK2, DIN2, CS2);
    max7219_brightness(CLK1, DIN1, CS1, 5);
    max7219_brightness(CLK2, DIN2, CS2, 5);
```

```
    absolute_time_t next_update = make_timeout_time_ms(1000);

    while(1) {
        check_pedestrian_buttons();

        if(time_reached(next_update)) {
            next_update = make_timeout_time_ms(1000);

            tiempo_restante--;

            // Guardar el estado anterior para saber si cambiamos
de fase
            semaforo_estado estado_anterior = estado_actual;
            bool cambio_de_estado = false;

            if(tiempo_restante <= 0) {
                cambio_de_estado = true;
                switch(estado_actual) {
                    case VERDE:
                        estado_actual = AMARILLO;
                        tiempo_restante = YELLOW_TIME/1000;
                        break;
                    case AMARILLO:
                        estado_actual = ROJO;
                        semaforo1_activo = !semaforo1_activo;
                        tiempo_restante = VEHICLE_GREEN_TIME/1000;
                        estado_actual = VERDE;
                        break;
                    case ROJO:
                        // No debería llegar aquí con la lógica
actual
```

```
                        break;
                }
            }

            semaforo_vehicular(estado_actual, semaforo1_activo);

            // Actualizar matrices si están en modo cuenta
regresiva
            if (mostrando_tiempo_matriz1 ||
mostrando_tiempo_matriz2) {
                actualizar_cuenta_regresiva();

                // Comprobar si hay un cambio de semáforo activo
                if (cambio_de_estado && estado_anterior ==
AMARILLO && estado_actual == VERDE) {
                    // Solo terminamos la cuenta regresiva cuando
pasamos de amarillo a verde
                    // y cambia el semáforo activo
                    mostrando_tiempo_matriz1 = false;
                    mostrando_tiempo_matriz2 = false;
                    mostrar_matriz(circulo, true);
                    mostrar_matriz(circulo, false);
                }
            }
        }

        actualizar_peatonales();
        sleep_ms(10); // Pequeña pausa para no saturar el CPU
    }

    return 0;
}
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 3.12)

include($ENV{PICO_SDK_PATH}/external/pico_sdk_import.cmake)

project(semaforo_doble C CXX ASM)

set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)

pico_sdk_init()

add_executable(semaforo_doble
    main.c
    max7219.c
)

target_include_directories(semaforo_doble PRIVATE
${CMAKE_CURRENT_LIST_DIR})

pico_enable_stdio_usb(semaforo_doble 1)
pico_enable_stdio_uart(semaforo_doble 0)

pico_add_extra_outputs(semaforo_doble)

target_link_libraries(semaforo_doble
    pico_stdlib
    hardware_gpio
    hardware_spi
    hardware_timer
```

)

Max7219.c

```
#include "max7219.h"
#include "hardware/spi.h"
#include "pico/stdlib.h"

void max7219_send(uint clk_pin, uint din_pin, uint cs_pin, uint8_t
reg, uint8_t data) {
    gpio_put(cs_pin, 0);

    for (int i = 7; i >= 0; i--) {
        gpio_put(clk_pin, 0);
        gpio_put(din_pin, (reg >> i) & 1);
        gpio_put(clk_pin, 1);
    }

    for (int i = 7; i >= 0; i--) {
        gpio_put(clk_pin, 0);
        gpio_put(din_pin, (data >> i) & 1);
        gpio_put(clk_pin, 1);
    }

    gpio_put(cs_pin, 1);
}

void max7219_init(uint clk_pin, uint din_pin, uint cs_pin) {
    gpio_init(clk_pin);
    gpio_init(din_pin);
    gpio_init(cs_pin);
```

```
    gpio_set_dir(clk_pin, GPIO_OUT);
    gpio_set_dir(din_pin, GPIO_OUT);
    gpio_set_dir(cs_pin, GPIO_OUT);

    gpio_put(cs_pin, 1);

    // Configuración del MAX7219
    max7219_send(clk_pin, din_pin, cs_pin, 0x09, 0x00); // Decode mode: no decode
    max7219_send(clk_pin, din_pin, cs_pin, 0x0A, 0x01); // Intensity: medium
    max7219_send(clk_pin, din_pin, cs_pin, 0x0B, 0x07); // Scan limit: all digits
    max7219_send(clk_pin, din_pin, cs_pin, 0x0C, 0x01); // Shutdown: normal operation
    max7219_send(clk_pin, din_pin, cs_pin, 0x0F, 0x00); // Display test: off
}

void max7219_display(uint clk_pin, uint din_pin, uint cs_pin, uint8_t *buffer) {
    for (uint8_t i = 0; i < 8; i++) {
        max7219_send(clk_pin, din_pin, cs_pin, i + 1, buffer[i]);
    }
}

void max7219_brightness(uint clk_pin, uint din_pin, uint cs_pin, uint8_t brightness) {
    if (brightness > 15) brightness = 15;
    max7219_send(clk_pin, din_pin, cs_pin, 0x0A, brightness);
}
```

Max7219.h

```
#ifndef MAX7219_H
#define MAX7219_H

#include "pico/stdlib.h"

void max7219_init(uint clk_pin, uint din_pin, uint cs_pin);
void max7219_display(uint clk_pin, uint din_pin, uint cs_pin,
uint8_t *buffer);
void max7219_brightness(uint clk_pin, uint din_pin, uint cs_pin,
uint8_t brightness);

#endif
```

● **Semáforo Vehicular 1 (con LEDs individuales)**

Función	Pin GPIO Pico	Descripción
Luz roja	GPIO 2	LED rojo semáforo 1
Luz amarilla	GPIO 3	LED amarillo semáforo 1
Luz verde	GPIO 4	LED verde semáforo 1

● **Semáforo Vehicular 2**

Función	Pin GPIO Pico	Descripción
Luz roja	GPIO 5	LED rojo semáforo 2
Luz amarilla	GPIO 6	LED amarillo semáforo 2
Luz verde	GPIO 7	LED verde semáforo 2

🚶 **Botones Peatonales**

Botón Peatonal	Pin GPIO Pico	Descripción
Botón Peatonal 1	GPIO 8	Solicita cruce peatonal matriz 1
Botón Peatonal 2	GPIO 9	Solicita cruce peatonal matriz 2

□ Matriz 1 (lado izquierdo o para peatonal 1)

Señal MAX7219	Pin GPIO Pico
DIN (Data In)	GPIO 10
CS (Chip Select)	GPIO 11
CLK (Clock)	GPIO 12

 Matriz 2 (lado derecho o para peatonal 2)

Señal MAX7219	Pin GPIO Pico
DIN (Data In)	GPIO 13
CS (Chip Select)	GPIO 14
CLK (Clock)	GPIO 15

Conclusión

Tuve dificultades para hacer funcionar adecuadamente los botones. Al presionar cualquier botón, debería mostrarse una cuenta regresiva del tiempo restante para que el semáforo peatonal cambie a rojo, pero actualmente solo se muestra en una sola matriz.

Esperaba que fuera más fácil, aunque quizá debí haber utilizado pantallas OLED debido a la gran cantidad de librerías y tutoriales disponibles, en comparación con las matrices de LEDs.

También considero que el aspecto estético del proyecto podría mejorarse. En otras ocasiones he logrado trabajos visualmente más atractivos; en esta ocasión, la disposición de los cables no me terminó de convencer.