Universidad Autónoma de San Luis Potosí

Facultad de Estudios Profesionales Zona Media

Práctica 14: sistema de cruce peatonal

Nombre del alumno: Sergio Adolfo Juárez Mendoza

Clave: 369376

Profesor: Ing. Jesús Padrón

Fecha de entrega: 5 de mayo del 2025

.

Introducción

El objetivo de esta práctica es diseñar y construir un sistema de cruce peatonal que funcione de forma sincronizada con semáforos vehiculares y peatonales. Esta simulación es importante en la industria del control de tráfico urbano, ya que permite entender cómo se implementan sistemas embebidos en aplicaciones reales.

Objetivo

poder recrear el funcionamiento de un sistema peatonal de dos semaforos peatonales y dos semaforos vehiculares al mismo tiempo de acompañar los semaforos peatonales con dos contadores

1 Material y Recursos Necesarios

1.1 Hardware

- Raspberry Pi Pico W
- 3 led verde
- 3 led rojos
- 2 led amarillo
- 2 displays de 7 segmentos
- 2 botones
- 12 resistencias de 240 ohmios
- Protoboard
- Cables de conexión calibre 22 AWG

1.2 Software y Librerías

- Visual Studio Code con soporte para Raspberry Pi Pico
- Librerías:
  - `pico/stdlib.h`
  - `hardware/adc.h`

Descripción del sistema

La maqueta consta de dos semáforos vehiculares (con LEDs rojo, amarillo y verde) y dos semáforos peatonales (con LEDs rojo y verde). El sistema incluye un botón para solicitar el cruce, el cual activa un contador regresivo de 10 segundos cuando el semáforo peatonal se pone en verde.

Funcionamiento

1. Al presionar el botón, el semáforo vehicular cambia de verde a amarillo por 2-3 segundos y luego a rojo.
2. Una vez que el semáforo vehicular está en rojo, el semáforo peatonal se pone en verde y comienza un contador regresivo de 10 segundos.
3. Cuando el conteo termina, el semáforo peatonal cambia a rojo y el vehicular a verde.

Desarrollo

creacion del codigo

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/timer.h"

// === Pines de botones ===
#define BOTON_1 11
#define BOTON_2 12

// === Sem foros peatonales ===
#define VERDE_P1 13
#define ROJO_P1  14
#define VERDE_P2 15
#define ROJO_P2  16

// === Sem foros vehiculares ===
#define VERDE_V1    17
#define AMARILLO_V1 18
#define ROJO_V1     19
#define VERDE_V2    20
#define AMARILLO_V2 21
#define ROJO_V2     22

// === Pines de segmentos del display ===
#define NUM_SEGMENTS 7
const uint SEGMENTS[NUM_SEGMENTS] = {2, 3, 4, 5, 6, 7, 8};

// === Control de displays ===
const uint DISPLAY_1_CTRL = 9;
```

```
const uint DISPLAY_2_CTRL = 10;

// Tabla de segmentos para n meros 0-9 ( nodo   com n)
const bool DIGITS[10][NUM_SEGMENTS] = {
    {1,1,1,1,1,1,0}, {0,1,1,0,0,0,0}, {1,1,0,1,1,0,1}, {1,1,1,1,0,0,1},
    {0,1,1,0,0,1,1}, {1,0,1,1,0,1,1}, {1,0,1,1,1,1,1}, {1,1,1,0,0,0,0},
    {1,1,1,1,1,1,1}, {1,1,1,1,0,1,1}
};

int counter = -1;
int active_display = 0;
int estado_semaforo[2] = {0, 2};
int tiempo_estado[2] = {0, 0};
int tiempo_cruce = 0;
bool parpadeo_verde[2] = {false, false};

void set_segments(int number) {
    for (int i = 0; i < NUM_SEGMENTS; i++) {
        gpio_put(SEGMENTS[i], DIGITS[number][i]);
    }
}

void clear_display() {
    for (int i = 0; i < NUM_SEGMENTS; i++) {
        gpio_put(SEGMENTS[i], 0);
    }
    gpio_put(DISPLAY_1_CTRL, 0);
    gpio_put(DISPLAY_2_CTRL, 0);
}

void init_gpio() {
    for (int i = 0; i < NUM_SEGMENTS; i++) {
        gpio_init(SEGMENTS[i]);
        gpio_set_dir(SEGMENTS[i], GPIO_OUT);
    }

    gpio_init(DISPLAY_1_CTRL);
    gpio_set_dir(DISPLAY_1_CTRL, GPIO_OUT);
    gpio_init(DISPLAY_2_CTRL);
    gpio_set_dir(DISPLAY_2_CTRL, GPIO_OUT);

    gpio_init(BOTON_1);
    gpio_set_dir(BOTON_1, GPIO_IN);
    gpio_pull_up(BOTON_1);

    gpio_init(BOTON_2);
    gpio_set_dir(BOTON_2, GPIO_IN);
    gpio_pull_up(BOTON_2);

    int leds[] = {VERDE_P1, ROJO_P1, VERDE_P2, ROJO_P2,
                  VERDE_V1, AMARILLO_V1, ROJO_V1,
                  VERDE_V2, AMARILLO_V2, ROJO_V2};
    for (int i = 0; i < 10; i++) {
        gpio_init(leds[i]);
```

```
        gpio_set_dir(leds[i], GPIO_OUT);
    }

    gpio_put(ROJO_P1, 1);
    gpio_put(ROJO_P2, 1);
    gpio_put(VERDE_V1, 1);
    gpio_put(ROJO_V2, 1);
}

void multiplex_display() {
    gpio_put(DISPLAY_1_CTRL, 0);
    gpio_put(DISPLAY_2_CTRL, 0);

    if (counter >= 0) {
        set_segments(counter);
        if (active_display == 1) {
            gpio_put(DISPLAY_1_CTRL, 1);
        } else if (active_display == 2) {
            gpio_put(DISPLAY_2_CTRL, 1);
        }
    }
}

void ciclo_semaforo(int cruce) {
    if (active_display == cruce) return;

    int verde = (cruce == 1) ? VERDE_V1 : VERDE_V2;
    int amarillo = (cruce == 1) ? AMARILLO_V1 : AMARILLO_V2;
    int rojo = (cruce == 1) ? ROJO_V1 : ROJO_V2;

    tiempo_estado[cruce - 1]++;

    switch (estado_semaforo[cruce - 1]) {
        case 0:
            if (tiempo_estado[cruce - 1] >= 650) {
                parpadeo_verde[cruce - 1] = true;
            }
            if (tiempo_estado[cruce - 1] >= 800) {
                gpio_put(verde, 0);
                gpio_put(amarillo, 1);
                estado_semaforo[cruce - 1] = 1;
                tiempo_estado[cruce - 1] = 0;
                parpadeo_verde[cruce - 1] = false;
            }
            if (parpadeo_verde[cruce - 1]) {
                gpio_put(verde, tiempo_estado[cruce - 1] % 20 < 10);
            }
            break;
        case 1:
            if (tiempo_estado[cruce - 1] >= 200) {
                gpio_put(amarillo, 0);
                gpio_put(rojo, 1);
                estado_semaforo[cruce - 1] = 2;
                tiempo_estado[cruce - 1] = 0;
```

```
            }
            break;
        case 2:
            if (tiempo_estado[cruce - 1] >= 800) {
                gpio_put(rojo, 0);
                gpio_put(verde, 1);
                estado_semaforo[cruce - 1] = 0;
                tiempo_estado[cruce - 1] = 0;
            }
            break;
    }
}

void forzar_rojo(int cruce) {
    int verde = (cruce == 1) ? VERDE_V1 : VERDE_V2;
    int amarillo = (cruce == 1) ? AMARILLO_V1 : AMARILLO_V2;
    int rojo = (cruce == 1) ? ROJO_V1 : ROJO_V2;

    gpio_put(verde, 0);
    gpio_put(amarillo, 0);
    gpio_put(rojo, 1);
    estado_semaforo[cruce - 1] = 2;
    tiempo_estado[cruce - 1] = 0;
}

void activar_cruce(int cruce) {
    int verde_p = (cruce == 1) ? VERDE_P1 : VERDE_P2;
    int rojo_p  = (cruce == 1) ? ROJO_P1  : ROJO_P2;

    forzar_rojo(cruce);
    gpio_put(rojo_p, 0);
    gpio_put(verde_p, 1);
    tiempo_cruce = 0;
}

void desactivar_cruce(int cruce) {
    int verde_v = (cruce == 1) ? VERDE_V1 : VERDE_V2;
    int rojo_v  = (cruce == 1) ? ROJO_V1  : ROJO_V2;
    int verde_p = (cruce == 1) ? VERDE_P1 : VERDE_P2;
    int rojo_p  = (cruce == 1) ? ROJO_P1  : ROJO_P2;

    gpio_put(verde_p, 0);
    gpio_put(rojo_p, 1);
    gpio_put(rojo_v, 0);
    gpio_put(verde_v, 1);
    estado_semaforo[cruce - 1] = 0;
    tiempo_estado[cruce - 1] = 0;
}

int main() {
    stdio_init_all();
    init_gpio();
    absolute_time_t last_time = get_absolute_time();

```

```
    while (true) {
        if (active_display == 0) {
            if (!gpio_get(BOTON_1)) {
                active_display = 1;
                counter = 9;
                activar_cruce(1);
            } else if (!gpio_get(BOTON_2)) {
                active_display = 2;
                counter = 9;
                activar_cruce(2);
            }
        }

        if (absolute_time_diff_us(last_time, get_absolute_time()) > 2000000) {
            if (counter > 0) {
                counter--;
            } else if (counter == 0) {
                desactivar_cruce(active_display);
                counter = -1;
                active_display = 0;
                clear_display();
            }
            last_time = get_absolute_time();
        }

        ciclo_semaforo(1);
        ciclo_semaforo(2);
        multiplex_display();
        sleep_ms(5);
    }
}
```

Listing 1: semaforo.c

2 Explicación del Código Fuente

El programa implementa un sistema de cruce peatonal controlado por una Raspberry Pi Pico. A continuación se describen las partes más importantes del código:

2.1 Inclusión de bibliotecas y definición de pines

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/timer.h"
```

Estas líneas incluyen las bibliotecas necesarias para trabajar con la Raspberry Pi Pico, los pines GPIO y temporizadores.

```
#define BOTON_1 11
#define BOTON_2 12
#define VERDE_P1 13
#define ROJO_P1  14
...
```

Se definen constantes con los números de los pines utilizados para los botones, semáforos peatonales, semáforos vehiculares y los segmentos del display de 7 segmentos.

2.2 Configuración del Display

```
const uint SEGMENTS[NUM_SEGMENTS] = {2, 3, 4, 5, 6, 7, 8};

const bool DIGITS[10][NUM_SEGMENTS] = {
    {1,1,1,1,1,1,0}, {0,1,1,0,0,0,0}, ...
};
```

Se utiliza una tabla de segmentos (‘DIGITS‘) que representa los números del 0 al 9 para un display de ánodo común. Cada número activa los pines necesarios para mostrar la cifra deseada.

2.3 Inicialización de Pines

```
void init_gpio() {
    ...
}
```

Esta función configura todos los pines como entrada o salida dependiendo de su uso (botones, LEDs, segmentos, control de displays).

2.4 Manejo del Display Multiplexado

```
void multiplex_display() {
    ...
}
```

El sistema utiliza multiplexado para alternar entre los dos displays de 7 segmentos y mostrar el número de cuenta regresiva.

2.5 Ciclo de Semáforo

```
void ciclo_semaforo(int cruce) {
    ...
}
```

Controla el ciclo de luces (verde → amarillo → rojo → verde) de los semáforos vehiculares, incluyendo el parpadeo del verde al finalizar su tiempo.

2.6 Activación y Desactivación del Cruce Peatonal

```
void activar_cruce(int cruce) { ... }
void desactivar_cruce(int cruce) { ... }
```

Estas funciones controlan cuándo un peatón puede cruzar (verde peatonal encendido, rojo vehicular encendido), y cuándo debe detenerse (rojo peatonal encendido, verde vehicular activado).

2.7 Función Principal

```
int main() {
    ...
}
```

El ‘main()‘ se encarga de:

- Inicializar los pines y la comunicación estándar.
- Detectar si un botón ha sido presionado.
- Iniciar una cuenta regresiva en el display correspondiente.
- Controlar el flujo del tráfico vehicular y peatonal.
- Multiplexar los displays de 7 segmentos.

Este código implementa un sistema simple pero funcional de cruce peatonal con control por botones, luces LED de semáforo y displays para cuenta regresiva.

creacion del CMakeLists

```
cmake_minimum_required(VERSION 3.13)

# Nombre del proyecto
project(Practica_14)

# Habilitar la Raspberry Pi Pico SDK
include(pico_sdk_import.cmake)
pico_sdk_init()

# Agregar el archivo fuente
add_executable(semaforo
    semaforo.c
)

# Agregar las bibliotecas est ndar de la Pico
target_link_libraries(semaforo
    pico_stdlib
    hardware_gpio
    hardware_timer
)

# Habilitar la salida USB para el puerto serie
pico_enable_stdio_usb(semaforo 1)
pico_enable_stdio_uart(semaforo 0)

# Crear el archivo binario UF2
pico_add_extra_outputs(semaforo)
```

Listing 2: CMakeLists

3 Archivo CMakeLists.txt

Para compilar el proyecto en la Raspberry Pi Pico, se utiliza el sistema de construcción `CMake`, que permite configurar y gestionar el proceso de compilación de manera sencilla y estructurada. A continuación se describe el contenido del archivo `CMakeLists.txt` utilizado:

Explicación del contenido

- **Versión mínima:** La línea `cmake_minimum_required(VERSION 3.13)` indica que se requiere al menos la versión 3.13 de CMake.
- **Nombre del proyecto:** Con `project(Practica_14)` se define el nombre interno del proyecto.
- **Importación e inicialización del SDK:**
  - `include(pico_sdk_import.cmake)` importa el SDK oficial de Raspberry Pi Pico.
  - `pico_sdk_init()` inicializa el entorno de desarrollo.

- **Archivo fuente y ejecutable:**
  - `add_executable(semaforo semaforo.c)` indica que se generará un ejecutable llamado `semaforo` a partir del archivo fuente `semaforo.c`.
- **Vinculación de bibliotecas:**
  - `pico_stdlib`: Biblioteca estándar del SDK.
  - `hardware_gpio`: Para manejo de entradas/salidas digitales.
  - `hardware_timer`: Para uso de temporizadores.
- **Configuración de salida estándar:**
  - Se habilita la salida por USB (`pico_enable_stdio_usb`) para funciones como `printf`.
  - Se desactiva la salida por UART.
- **Generación de salidas adicionales:**
  - `pico_add_extra_outputs` genera archivos `.uf2`, `.bin` y otros formatos necesarios para cargar el programa en la placa.

4 Problemas

Durante el diseño e implementación del sistema de cruce peatonal con la Raspberry Pi Pico se enfrentaron varios desafíos técnicos. Estos problemas surgieron tanto en la lógica de control como en el manejo de hardware, afectando el funcionamiento esperado del sistema. A continuación se describen los problemas más relevantes encontrados y las soluciones aplicadas.

4.1 Sincronización Incorrecta de Semáforos Vehiculares

Uno de los problemas más críticos detectados fue la sincronización incorrecta de los semáforos vehiculares correspondientes a los dos cruces independientes. Debido a una programación deficiente en la gestión de los estados de los semáforos, en determinados momentos ambos semáforos vehiculares se encontraban en verde al mismo tiempo, lo que representa un comportamiento inaceptable en un sistema de tráfico real, ya que podría causar accidentes por permitir el paso simultáneo de vehículos en direcciones conflictivas.

Causa: Este problema se debió a que los estados de los semáforos se manejaban de forma independiente, sin ninguna coordinación centralizada. La lógica de control permitía que ambos semáforos iniciaran o mantuvieran el estado de luz verde si no había una petición activa de cruce peatonal, provocando conflictos de tráfico.

Solución: Para resolver esta situación, se implementó una verificación adicional en la función que gestiona el cambio de estados del semáforo. Antes de permitir que un semáforo pase a verde, se valida que el otro cruce no esté en uso ni en transición. Además, se forzó el

encendido del semáforo rojo en el cruce opuesto al activar un cruce peatonal. Esto asegura que sólo un semáforo pueda estar en verde al mismo tiempo, manteniendo la exclusividad del paso vehicular.

4.2 Problemas con la Multiplexación de los Displays

Otro inconveniente importante se presentó en la visualización de los números en los displays de 7 segmentos. Estos displays están multiplexados, lo que significa que se comparten los mismos pines de segmentos para ambos, y se activan alternadamente a alta velocidad para simular que ambos están encendidos de manera simultánea.

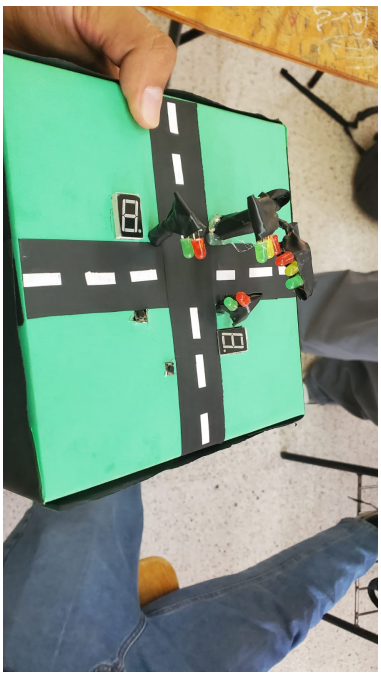
Causa: El problema surgió porque el tiempo de activación de cada display no estaba bien ajustado. Además, la rutina de multiplexación no se ejecutaba con la frecuencia suficiente, o bien se interrumpía por otras tareas como el control de semáforos. Esto provocaba parpadeos, visualización errónea de los dígitos o incluso que el número no se mostrara en absoluto.

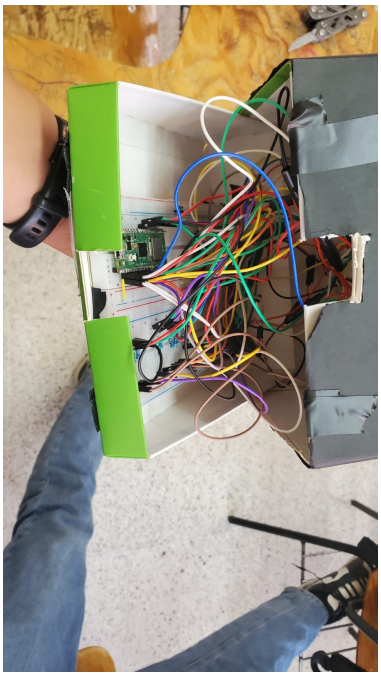
Solución: Se reestructuró el código para asegurar que la rutina de multiplexación se ejecute periódicamente con un retardo fijo (por ejemplo, cada 5 milisegundos), independientemente del resto de la lógica del sistema. También se revisó el orden de activación de los pines de control de cada display para garantizar que solo uno esté activo en un momento dado, evitando interferencias visuales.

4.3 Lecciones Aprendidas

Estos problemas resaltaron la importancia de una correcta planificación en sistemas embebidos con múltiples tareas concurrentes. La coordinación entre módulos —en este caso, semáforos, botones y displays— requiere una lógica clara y jerárquica que evite conflictos y errores de sincronización. Asimismo, la multiplexación efectiva de displays demanda un control preciso del tiempo, lo que en plataformas de tiempo real como la Raspberry Pi Pico requiere especial atención a la estructura del bucle principal y a los temporizadores utilizados.Finalmente, se espera 100 ms antes de repetir el ciclo, estabilizando la lectura y evitando saturación del monitor serial.

pruebas





Conclusion

La implementación del sistema de cruce peatonal utilizando la Raspberry Pi Pico permitió reforzar conocimientos en programación embebida, control de dispositivos a nivel de hardware y manejo de entradas/salidas digitales. Se logró integrar con éxito múltiples componentes como LEDs, displays de 7 segmentos, botones y lógica de temporización, para simular el comportamiento realista de un semáforo peatonal.

A pesar de los desafíos enfrentados —como la sincronización no deseada de los semáforos vehiculares y la complejidad en la multiplexación de los displays— fue posible identificar y corregir estos errores mediante un análisis cuidadoso del código y ajustes en la lógica de control.

Este proyecto sirvió como una valiosa experiencia práctica, permitiendo comprender la importancia del diseño estructurado del código, la planificación del comportamiento secuencial de un sistema embebido y la depuración en tiempo real. En general, se cumplió el objetivo de desarrollar un sistema funcional que responde adecuadamente a las señales de los usuarios simulando un entorno urbano de cruce peatonal.