INFORME DE LABORATORIO 3

Autores: *María Del Mar Arbeláez Sandoval, Julián Mauricio Sánchez Ceballos*

Laboratorio de Electrónica Digital 3
Departamento de Ingeniería Electrónica y de Telecomunicaciones
Universidad de Antioquia

Resumen

En este proyecto de laboratorio se implementó un sistema de control de acceso utilizando un teclado matricial y LEDs como indicadores de verificación. El objetivo principal desarrollado en la práctica fue crear un programa simple que gestione el acceso de usuario mediante un ID y una contraseña, aprovechando la gestión de los GPIOs de la Raspberry Pi Pico. El sistema incluye funciones como la validación de usuarios, el bloqueo de usuarios tras tres intentos incorrectos y la posibilidad de cambiar la contraseña. El escaneo del teclado se implementó mediante el uso de GPIOs, y para el control del flujo de acceso se empleó una máquina de estados la cual asegura la modularidad y la escalabilidad en el diseño del código. El proyecto demostró el uso de interrupciones, técnicas de antirrebote y temporizadores, conceptos esenciales para desarrollo de aplicaciones en sistemas embebidos. Este sistema desarrollado muestra como la gestión de GPIOs y el desarrollo modular de software puede aplicarse a diferentes aplicaciones embebidas, proporcionando una compresión clara sobre estos sistemas y las aplicaciones interactivas que se pueden diseñar.

Palabras clave: Control de Acceso, teclado matricial, Raspberry Pi Pico, LEDs, GPIOs, estructuras de datos, antirrebote, temporizadores, sistemas embebidos.

INTRODUCCIÓN

Para este laboratorio, se implementó un sistema de control de acceso por medio del uso de un teclado matricial y LEDs indicadores. Este laboratorio tiene como objetivo principal la estructuración de un programa simple que permita el control de acceso mediante una ID y una contraseña, para esto durante el desarrollo de la práctica se implementa la configuración debida para la lectura y manipulación de un teclado matricial, junto con el uso de estructura de datos para la gestión de los diferentes usuarios y contraseñas, el manejo de GPIOs para las señales luminosas implementadas mediante LEDs que indican el estado de verificación del usuario y el uso de timers para el manejo del flujo del programa y algunas funcionalidades implementadas.

Se tienen algunos detalles del programa como:

- Ingreso y validación de usuarios.
- Bloqueo de usuario luego de 3 intentos.
- Cambio de contraseña.

Se muestran en profundidad en el desarrollo de este informe.

ESQUEMÁTICO

El esquemático principal implementado en la práctica se muestra en la figura **0-1**, conectando de los pines 2 a 9.

La conexión de los tres LEDs indicadores se da de la siguiente manera: el LED rojo va conectado al pin 10, el LED verde al pin 11 y el amarillo al pin 12.

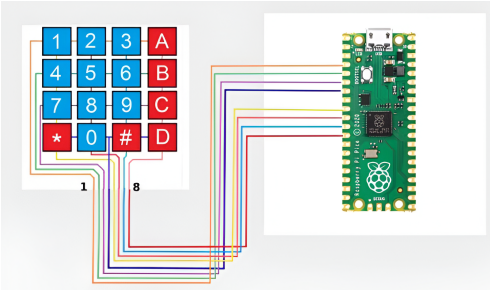

Figura **0-1**: Diagrama de conexión del teclado matricial.

SOFTWARE

Lectura del teclado matricial

La lectura del teclado matricial se basa en escanear filas y columnas para identificar la tecla presionada. A continuación, se presenta el funcionamiento detallado y la implementación.

Interrupciones

El siguiente fragmento de código muestra cómo se configura un *callback* para gestionar la interrupción generada por el teclado cuando se presiona una tecla:

```
   basicstyle
/**
 * @brief Funci n para gestionar la
 * interrupci n al presionar una tecla.
 */
void mat_callback(uint gpio, uint32_t events){
    if (reading == 0){
        reading = 1;
        gpio_pressed = (uint8_t)gpio;
    }
}
```

Este *callback* activa la bandera de lectura *reading* cuando se detecta un cambio en el estado del GPIO asociado.

Escaneo de Pines

El proceso de escaneo, tras notar un flanco de subida, busca cambiar las columnas como salida y las filas como entradas para identificar la tecla presionada:

```
   basicstyle
/**
 * @brief Configurar pines de entrada/salida.
 */
void set_dirs_mat(bool change){
    if (change == false){
        gpio_set_dir_in_masked(MAT_READ_PINS);
        gpio_set_dir_out_masked(MAT_WRITE_PINS);
    } else {
        gpio_set_dir_in_masked(MAT_WRITE_PINS);
        gpio_set_dir_out_masked(MAT_READ_PINS);
    }
}
```

Este código configura los pines GPIO para alternar entre escritura en las filas y lectura en las columnas, dependiendo de la fase del escaneo.

Debounce

La siguiente implementación realiza un *debounce*, eliminando los rebotes eléctricos que podrían causar lecturas erróneas:

```
   basicstyle
else if (!gpio_get(gpio_pressed)
        && reading == 3){
    start = time_us_32();
    reading = 4;
}
else if (reading == 4
        && ((time_us_32() - start)
        / 1000 >= 50)){
    if (gpio_get(gpio_pressed) == 1){
        reading = 3;
    } else {
        reading = 0;
        return true;
    }
}
```

Este código asegura que la tecla presionada se mantenga estable en bajo durante al menos 50 ms antes de confirmar la lectura.

Mapeo de teclas

Para traducir la fila y columna detectadas a un valor de tecla, se utiliza una tabla de mapeo:

```
   basicstyle
/**
 * @brief Diccionario para mapeo de teclas.
 */
char dic_mat[16] = {'1', '2', '3', 'A',
                    '4', '5', '6', 'B',
                    '7', '8', '9', 'C',
                    '*', '0', '#', 'D'};
uint8_t lectura = (gpio_get_all()
                & MAT_WRITE_PINS) >> 2;
if (lectura){
    uint8_t c = 0;
    while (!(lectura >> c & 1)){c++;}
    *value = dic_mat[c * 4 +
                gpio_pressed - 6];
}
```

Este código toma la lectura binaria de los pines GPIO y la traduce a un carácter o número en función de la tecla presionada.

Manejo de LEDs

En esta sección se describe la implementación de la librería para el control de LEDs en un sistema embebido utilizando la Raspberry Pi Pico. Se implementaron funciones que permiten encender, apagar, y hacer titilar (parpadear) los LEDs, gestionando el control de temporizadores para el titilado.

Inicialización de los LEDs

La inicialización de los pines asociados a los LEDs se realiza mediante la función `init_leds()`, que configura los pines GPIO como salidas y apaga los LEDs por defecto.

```
   basicstyle
/**
 * @brief Inicializa los pines asociados a los LEDs
 */
void init_leds(void) {
    gpio_init_mask(LEDS_PINS);
    gpio_set_dir_out_masked(LEDS_PINS);
    gpio_put_masked(LEDS_PINS, 0); // Apaga LEDs
}
```

Encendido de LEDs para señales de éxito y error

Las funciones `signal_success()` y `signal_error()` controlan los LEDs verde y rojo, respectivamente, para indicar estados de éxito o error. Ambas funciones cancelan cualquier titilado previo que esté activo.

```
   basicstyle
/**
 * @brief Se al de  xito  con LED verde
 */
void signal_success() {
    if (g_alarm_led_id != -1) {
        cancel_alarm(g_alarm_led_id); // Cancela alarma previa
        g_alarm_led_id = -1;
    }
    gpio_put_masked(LED_GREEN, LED_GREEN);
}

/**
 * @brief Se al de error con LED rojo
 */
void signal_error() {
    if (g_alarm_led_id != -1) {
        cancel_alarm(g_alarm_led_id); // Cancela alarma previa
        g_alarm_led_id = -1;
    }
    gpio_put_masked(LED_RED, LED_RED);
}
```

Control del LED amarillo

El LED amarillo puede estar en tres estados: apagado, encendido o titilando. La función `control_yellow_led()` permite seleccionar entre estos tres comportamientos:

```
   basicstyle
/**
 * @brief Controla el estado del LED amarillo
 * @param state 0: Apagado, 1: Encendido, 2: Titilando
 */
void control_yellow_led(uint8_t state) {
    if (state == 0) {
        cancel_alarm_if_exists();
        gpio_put_masked(LED_YELLOW, 0 << 12); // Apaga LED
    }
    else if (state == 1) {
        cancel_alarm_if_exists();
        gpio_put_masked(LED_YELLOW, LED_YELLOW); // Enciende LED
    }
    else if (state == 2) {
        cancel_alarm_if_exists();
        g_alarm_led_id = add_alarm_in_ms(500, blink_callback,
             NULL, false); // Configura titilado
    }
}
```

Callback de titilado

El callback `blink_callback()` se ejecuta periódicamente cuando el LED amarillo está titilando, alternando su estado cada 500 ms:

```
   basicstyle
/**
 * @brief Callback para titilar el LED amarillo
 * @param id ID del temporizador
 * @return 500*1000 (500ms)
 */
int64_t blink_callback(alarm_id_t id, void *user_data) {
    if (id != g_alarm_led_id) {
        return 0;  // No continuar si la alarma fue cancelada
    }
    g_led_state = !g_led_state;
    gpio_put_masked(LED_YELLOW, g_led_state << 12);
    return 500 * 1000;   // Repite cada 500ms
}
```

Apagado de LEDs

Finalmente, la función `turn_off_leds()` apaga todos los LEDs después de una breve espera de 2 segundos y cancela cualquier temporizador de titilado activo:

```
   basicstyle
/**
 * @brief Apaga todos los LEDs
 */
void turn_off_leds(void) {
    sleep_ms(2000);  // Espera 2 segundos
    gpio_put_masked(LEDS_PINS, 0x000 << 10); // Apaga LEDs
    if (g_alarm_led_id != -1) {
        cancel_alarm(g_alarm_led_id);  // Cancela alarma
        g_alarm_led_id = -1;
    }
}
```

Cancelación de alarmas

Para evitar que los LEDs sigan titilando si hay un cambio de estado, se implementa una función para cancelar las alarmas previas:

```
   basicstyle
/**
 * @brief Cancela la alarma si existe
 */
void cancel_alarm_if_exists() {
    if (g_alarm_led_id != -1) {
        cancel_alarm(g_alarm_led_id);
        g_alarm_led_id = -1;
    }
}
```

Este enfoque modular permite una fácil gestión de las señales visuales (LEDs) en el sistema embebido.

Máquina de Estados para el Control de Acceso

En esta sección se describe la implementación de una máquina de estados (FSM) que gestiona la entrada de ID y contraseña del usuario, así como la verificación de las credenciales. Esta máquina de estados también incluye la capacidad para cambiar la contraseña si el usuario lo solicita.

Inicialización de la FSM

La máquina de estados se inicializa mediante la función `fsm_init()`, la cual establece el estado inicial como `state_waiting`, donde el sistema espera la entrada del usuario. Se restablecen los índices de ID y contraseña para asegurar que no queden valores residuales de entradas previas.

```
   basicstyle
/**
 * @brief Inicializa la m quina de estados
 */
void fsm_init(void) {
    current_state = state_waiting; // Estado inicial
    id_index = 0;                  // Resetear  ndices
    pass_index = 0;
}
```

Ejecución de la FSM

La función `fsm_run()` es responsable de ejecutar la máquina de estados, llamando a la función correspondiente al estado actual.

```
   basicstyle
/**
 * @brief Ejecuta la m quina de estados
 */
void fsm_run(void) {
    current_state(); // Llama a la funci n del estado actual
}
```

Estado: Waiting (Esperando Entrada)

El primer estado, `state_waiting`, espera a que el usuario presione una tecla para iniciar el proceso de ingreso de ID. Si se presiona el asterisco (*), el sistema solicita el ingreso del ID. Si se presiona el símbolo de numeral (), el sistema permite el cambio de contraseña.

```
   basicstyle
/**
 * @brief Estado de espera que espera la entrada de una tecla
 */
static void state_waiting(void) {
    uint8_t key;
    control_yellow_led(1);              // LED amarillo listo
         para recibir ID

    while (!read_mat(&key));

    if (key == '*') {
        printf("Ingresa␣ID:␣");
        change_pass = false;
        current_state = state_entering_credentials;
        control_yellow_led(0);          // Apaga el LED amarillo
        id_index = 0;
        pass_index = 0;
    } else if (key == '#') {
        printf("Ingresa␣ID␣para␣el␣cambio␣de␣contrase a:␣");
        change_pass = true;
        current_state = state_entering_credentials;
        control_yellow_led(0);          // Apaga el LED amarillo
        id_index = 0;
        pass_index = 0;
    }
}
```

Estado: Entering Credentials (Ingreso del ID)

En este estado, el usuario ingresa su ID. La máquina de estados se encarga de recibir las teclas desde un teclado matricial y almacenar el ID en el buffer correspondiente.

```
   basicstyle
/**
 * @brief Estado en el que el usuario ingresa su ID
 */
static void state_entering_credentials(void) {
    uint8_t key;
    control_yellow_led(0);
    while (!read_mat(&key));

    if (id_index < 6) {
        user_id[id_index++] = key;
        printf("%c", key);

        if (id_index == 6) {
            user_id[id_index] = '\0'; // Cadena terminada
            printf("\nIngresa␣Contrase a:␣");
            current_state = state_entering_pass; // Cambio de
                 estado
        }
    }
}
```

Estado: Entering Password (Ingreso de Contraseña)

El estado `state_entering_pass` permite al usuario ingresar su contraseña. El LED amarillo titila durante este proceso. Si el tiempo de ingreso excede 10 segundos, se envía una señal de error, se apagan los LEDs, y se vuelve al estado de espera.

```
   basicstyle
/**
 * @brief Estado en el que el usuario ingresa su contrase a
 */
static void state_entering_pass(void) {
    uint8_t key;
    control_yellow_led(2); // LED amarillo titila indicando
         ingreso de contrase a
    uint32_t start_time = time_us_32();

    while (!read_mat(&key));
    if (pass_index < 4) {
        user_pass[pass_index++] = key;
        printf("%c", key);

        if (pass_index == 4) {
            user_pass[pass_index] = '\0'; // Cadena terminada
            if (time_us_32() - start_time > 10000 * 1000) {
                signal_error(2000);
                printf("\nTiempo␣de␣ingreso␣excedido\n");
                current_state = state_waiting;
                turn_off_leds();
            } else {
                current_state = state_verifying;
            }
        }
    }
}
```

Estado: Entering New Password (Ingreso de Nueva Contraseña)

Si el usuario solicitó cambiar la contraseña, este estado le permitirá ingresar la nueva contraseña. La fun-

ción `change_password()` se encarga de realizar el cambio cuando la nueva contraseña se ingresa correctamente.

```
   basicstyle
/**
 * @brief Estado para ingresar la nueva contrase a
 */
static void state_entering_new_pass(void) {
    uint8_t key;
    control_yellow_led(2); // LED amarillo titila indicando
         ingreso de nueva contrase a

    while (!read_mat(&key));

    if (pass_index < 4) {
        user_new_pass[pass_index++] = key;
        printf("%c", key);

        if (pass_index == 4) {
            user_new_pass[pass_index] = '\0';
            printf("\nNueva␣contrase a␣ingresada:␣%s\n",
                 user_new_pass);
            change_password(user_id, user_new_pass);
            printf("Contrase a␣cambiada␣exitosamente.\n");
            current_state = state_waiting;  // Vuelve al estado
                 de espera
        }
    }
}
```

Estado: Verifying (Verificación de Credenciales)

En este estado se verifican las credenciales ingresadas contra las almacenadas. Si son correctas, el acceso es concedido. Si el usuario solicitó cambiar la contraseña, se le permite ingresar una nueva. En caso contrario, si las credenciales son incorrectas, el acceso es denegado.

```
   basicstyle
/**
 * @brief Estado de verificaci n de credenciales
 */
static void state_verifying(void) {
    control_yellow_led(0); // Apaga el LED amarillo

    if (access_control(user_id, user_pass)) {
        printf("Acceso␣concedido\n");
        if (change_pass) {
            pass_index = 0;             // Reinicia el  ndice
                 de la contrase a
            printf("Cambio␣de␣contrase a␣solicitado.\n");
            printf("Ingresa␣nueva␣contrase a:␣");
            current_state = state_entering_new_pass;
        } else {
            current_state = state_waiting;
        }
        signal_success();
    } else {
        printf("Acceso␣denegado\n");
        signal_error(2000);
        current_state = state_waiting;
    }

    turn_off_leds();
    control_yellow_led(1);  // Prepara el LED para el estado de
         espera
}
```

Este enfoque modular y escalable de la máquina de estados garantiza un control claro sobre el flujo de entradas, proporcionando robustez y flexibilidad para gestionar tanto el acceso como la modificación de la contraseña.

IMPLEMENTACIÓN

La implementación de este sistema de control de acceso utilizando la Raspberry Pi Pico y un teclado matricial se realizó en varios pasos clave, abordando desde la configuración inicial hasta el desarrollo de la máquina de estados para el control de acceso.

Configuración del Entorno

Inicialmente, se configuraron los pines GPIO de la Raspberry Pi Pico para que pudieran manejar tanto el teclado matricial como los LEDs. Se utilizaron pines específicos para la lectura de las filas y columnas del teclado, así como para encender y apagar los LEDs indicadores. Los pines fueron configurados como entradas y salidas utilizando las funciones provistas por el SDK de Raspberry Pi Pico.

Manejo de Entradas

La lectura del teclado matricial se implementó mediante el escaneo de las filas y columnas, detectando las teclas presionadas a través de interrupciones y escaneo de pines. Se aplicó una técnica de debounce para evitar lecturas erróneas debido a rebotes eléctricos.

Control de LEDs

El sistema de LEDs se implementó para proporcionar señales visuales sobre el estado de verificación de las credenciales. Se implementaron funciones para encender, apagar y hacer titilar los LEDs en diferentes combinaciones, indicando éxito, error y la espera de una nueva entrada.

Máquina de Estados

El sistema de control de acceso fue implementado como una máquina de estados, la cual maneja el flujo entre diferentes estados: esperando entrada, ingresando ID, ingresando contraseña, verificando credenciales y cambiando la contraseña. Cada estado fue implementado como una función independiente que se ejecuta en el ciclo principal del sistema.

Pruebas y Depuración

Se realizaron pruebas exhaustivas para asegurar que la máquina de estados se ejecutara correctamente. Se verificó que el sistema gestionara correctamente el ingreso

de ID y contraseña, el bloqueo después de múltiples intentos fallidos, y la capacidad de cambiar la contraseña. Se resolvieron los problemas de rebote en la lectura de teclas mediante el uso adecuado del debounce.

CONCLUSIONES

- La implementación de este sistema de control de acceso proporcionó una valiosa experiencia en el manejo de recursos limitados en sistemas embebidos, como los GPIOs de la Raspberry Pi Pico. Si bien los GPIOs fueron suficientes para esta práctica, en futuros proyectos será necesario planificar mejor la distribución de estos recursos para optimizar su uso.
- El uso del SDK de Raspberry Pi Pico demostró ser fundamental para el desarrollo de este sistema, ya que ofrece una amplia gama de opciones para configurar y manejar los pines GPIO de manera eficiente. Las funciones modulares implementadas facilitaron el desarrollo, permitiendo mantener el código limpio y escalable.
- A través de esta práctica, se consolidaron los conocimientos sobre el uso de GPIOs y la implementación de máquinas de estados, lo que permitió crear un sistema funcional para el control de acceso. Además, se logró comprender cómo los sistemas embebidos pueden aplicarse en la vida real para diseñar soluciones interactivas y seguras.

BIBLIOGRAFÍA:

- Raspberry Pi Foundation. Raspberry Pi Pico documentation. Recuperado de https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html#raspberry-pi-pico-and-pico-h