
INTERFACES DE CONTROL Y VISUALIZACIÓN DE DATOS

3.1 INTRODUCCIÓN: AL FIN, EL SILICIO COBRA VIDA

Hasta este punto del manual, hemos dedicado nuestro tiempo a construir los cimientos de la arquitectura mental de un programador. Pasamos por el abismo de las variables, desarmamos los motores de los bucles y firmamos los contratos estrictos de las funciones. Todo ese conocimiento en lenguaje C era vital, pero sabemos que hasta ahora se sentía un poco “abstracto”, viviendo únicamente dentro de la memoria RAM de nuestra Raspberry Pi Pico W o mostrándose como simples líneas de texto en la terminal de la computadora.

Eso se termina hoy. Bienvenidos, al fin, a la etapa de los proyectos reales.

En el mundo de la automatización industrial y el desarrollo de tecnologías como el IoT, un microcontrolador aislado no sirve de nada si no puede comunicarse con su entorno. Un chip necesita “ojos” para entender el mundo (sensores), “músculos” para modificar la realidad (actuadores como relés o motores) y, fundamentalmente, canales para hablar con el ser humano.

De eso se trata esta unidad. Vamos a meter las manos en el hardware para diseñar interfaces que nos permitan tomar el control físico del sistema y visualizar la información crítica en tiempo real.

A partir de los siguientes capítulos, dejaremos de lado los ejemplos de laboratorio para empezar a resolver desafíos del mundo real.

Prepara tu protoboard, tus cables y componentes. A partir de este momento, cada línea de código en C que escribas se convertirá en una luz que se enciende, un motor que arranca o una pantalla que reporta el estado de un proceso productivo. ¡Manos a la obra!

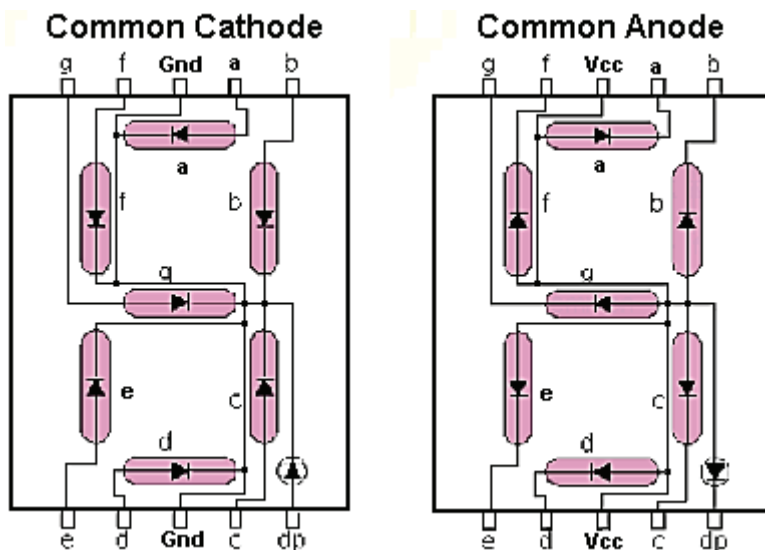
3.2 DISPLAYS DE 7 SEGMENTOS Y LÓGICA DE CONTROL

3.2.1 Estructura Básica

Un display de 7 segmentos es un componente electrónico compuesto por siete u ocho segmentos individuales que pueden encenderse o apagarse de forma independiente. Estos segmentos, generalmente en forma de líneas, se combinan para formar números del 0 al 9 y, en algunos casos, letras.

Tipos de Display de 7 Segmentos

- **Ánodo común:** Todos los ánodos (terminales positivos) están conectados a un punto común y al positivo de la fuente. Para encender un segmento, se conecta su cátodo (terminal negativo) a tierra.
- **Cátodo común:** Todos los cátodos están conectados a un punto común y al negativo de la fuente. Para encender un segmento, se aplica una tensión positiva a su ánodo.



Control del Display

Para controlar un display de 7 segmentos, se utiliza un microcontrolador como el Raspberry Pi Pico W. Se asignan los pines de salida del microcontrolador a cada segmento del display. Encendiendo y apagando los segmentos en la secuencia correcta, se pueden mostrar diferentes números y caracteres.

3.2.2 Aplicaciones

Los displays de 7 segmentos son ampliamente utilizados en:

- Relojes digitales: Para mostrar la hora y minutos.
- Contadores: Para visualizar números en ascenso o descenso.
- Medidores: Para mostrar valores numéricos de diversas magnitudes físicas (temperatura, voltaje, etc.).
- Instrumentos musicales: Para mostrar parámetros como el tempo o la frecuencia.
- Equipos electrónicos en general: Para mostrar cualquier tipo de información numérica.

3.2.3 Ventajas

- Bajo costo: Son componentes económicos y fáciles de encontrar.
- Sencillez: Su funcionamiento es relativamente sencillo de entender y programar.
- Visualización clara: Los números son fáciles de leer y distinguir.
- Bajo consumo de energía: Consumen poca energía, lo que los hace ideales para proyectos alimentados por batería.

3.2.4 Desventajas

- Limitación de caracteres: Solo pueden mostrar números y algunos caracteres limitados.
- Resolución limitada: La resolución es baja en comparación con otros tipos de pantallas.

3.2.5 Resumen

Los displays de 7 segmentos son dispositivos versátiles y económicos que permiten visualizar información numérica de forma sencilla. Son ideales para proyectos de electrónica donde se requiere una visualización básica de datos. Para controlar un display de 7 segmentos se configuran los pines de salida del microcontrolador para encender y apagar los segmentos correspondientes.

3.2.6 Práctica: manejo de 1 display de 7 segmentos en Raspberry Pi Pico W

Hardware necesario:

- Raspberry Pi Pico W: La placa de desarrollo.
- Display de 7 segmentos: De ánodo o cátodo comunes.
- Resistencias: Para limitar la corriente que pasa por los segmentos del display.
- Cables: Para conectar el display a la Pico W.
- Protoboard (opcional): Facilita la conexión de los componentes.

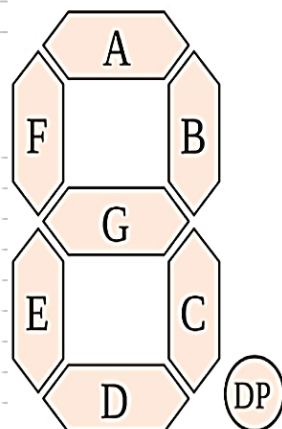
Conexión del display:

1. **Identificar los pines:** Cada segmento del display y el ánodo común (o cátodo común) tendrán un pin correspondiente.
2. **Conectar las resistencias:** Conecta una resistencia en serie entre cada segmento y el pin correspondiente en la Raspberry Pi Pico W. El valor de la resistencia dependerá del display y de la corriente máxima permitida.
3. **Conectar el común:** Conecta el ánodo o cátodo comunes a la fuente de alimentación (3.3v) o a tierra, según el tipo de display.

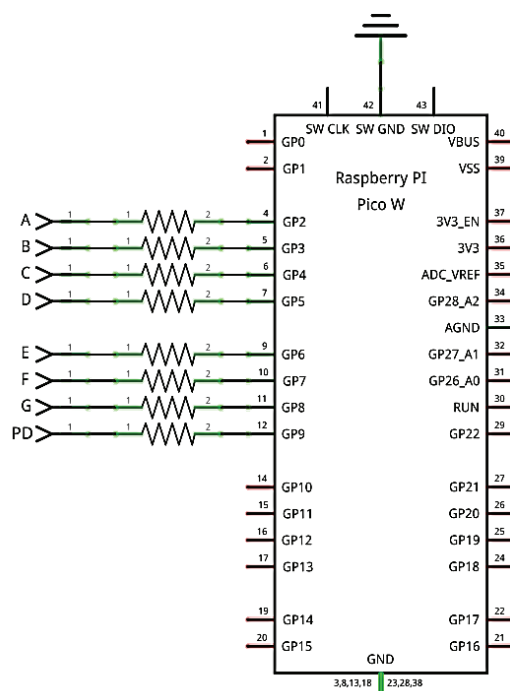
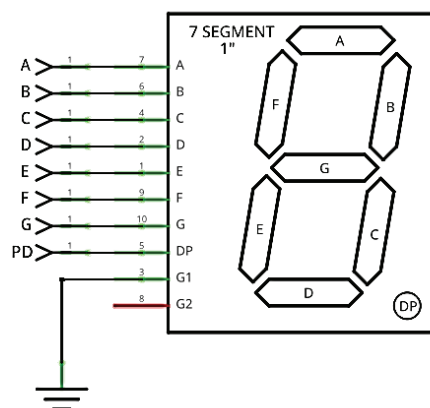
A continuación se muestra como quedaría la combinación de unos y ceros que deberíamos colocar en los bits para encender los leds de manera que formen los dígitos del 0 al 9.

Suponte que el bit 0 controla el segmento A del display, el bit 1 controla el segmento B y así sucesivamente hasta el bit 7 que controla el led del punto decimal. Siguiendo esta lógica, para mostrar el número 0 deberíamos encender los segmentos (A, B, C, D, E y F) y dejar apagados (G y el Punto decimal). Entonces el byte que guarda la combinación correspondiente al 0 quedaría así: 0b00111111 o lo que es lo mismo 0x3f en hexadecimal. Haciendo este ejercicio podremos obtener las combinaciones que forman todos los dígitos del 0 al 9 e incluso algunas letras y símbolos.

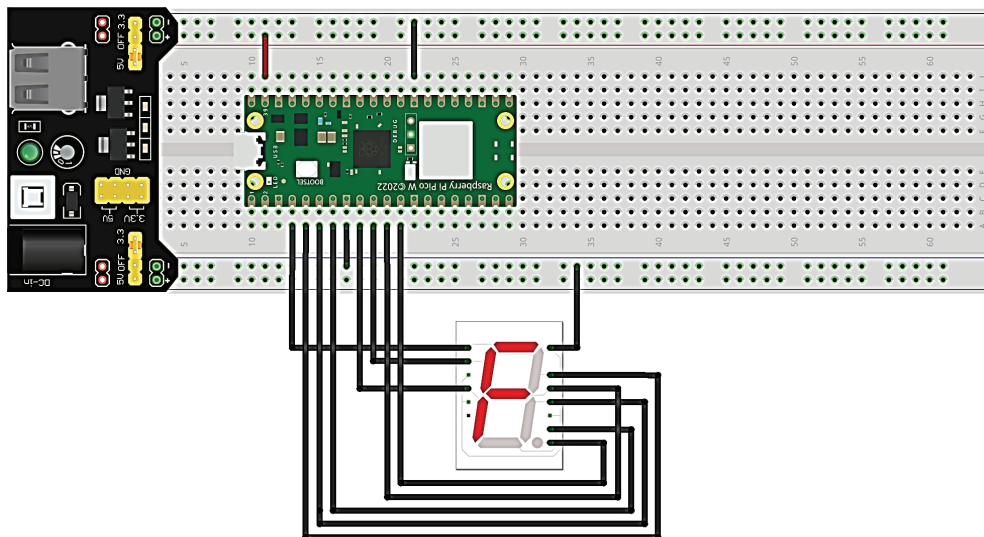
Byte									HEX
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
	DP	Segmento G	Segmento F	Segmento E	Segmento D	Segmento C	Segmento B	Segmento A	
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	1	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	1	1	1	1	0x69



Esquemático



fritzing



fritzing

Código en C

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

#define FIRST_GPIO 2

/* Este Array convierte un número del 0 al 9 en un patrón de bits para
enviar a los GPIO.*/

int bits[10] = {
    0x3f, // 0
    0x06, // 1
    0x5b, // 2
    0x4f, // 3
    0x66, // 4
    0x6d, // 5
    0x7d, // 6
    0x07, // 7
    0x7f, // 8
    0x67  // 9
};
```

```

int main() {
    stdio_init_all();
    printf("Hola 7segmentos\n");

    // Podríamos usar gpio_set_dir_out_masked() aquí
    for (int gpio = FIRST_GPIO; gpio < FIRST_GPIO + 7; gpio++) {
        gpio_init(gpio);
        gpio_set_dir(gpio, GPIO_OUT);
        // Nuestro mapa de bits anterior tiene un bit configurado a 1
donde
        //necesitamos que se encienda un LED.
    }

    int val = 0;
    while (true) {
        // Hacemos un corrimiento de la máscara según el pin desde el que
iniciamos.
        int32_t mask = bits[val] << FIRST_GPIO;

        /*¡Configure todos nuestros GPIO de una sola vez!
Si algo más usa GPIO, es posible que queramos usar gpio_put_
masked()*/
        gpio_set_mask(mask);
        sleep_ms(250);
        gpio_clr_mask(mask);

        val++;
        if(val>=10){
            val=0;
        }
    }
}

```

Explicación del Código

```

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

```

- Inclusión de bibliotecas:** Estas líneas incluyen las bibliotecas necesarias para realizar operaciones de entrada/salida estándar (stdio.h), para

inicializar la placa Raspberry Pi Pico (pico/stdlib.h) y para controlar los pines GPIO (hardware/gpio.h).

- **Definición de constantes:** Se define una constante `FIRST_GPIO` con el valor 2. Esta constante se utilizará como punto de partida para asignar los pines GPIO a los segmentos del display. Como comenzamos a conectar el display a partir del pin 2 con esta constante solucionamos ese corrimiento.

```
int bits[10] = {
    0x3f, // combinación de bits para el número 0
    0x06, // " 1
    0x5b, // " 2
    0x4f, // " 3
    0x66, // " 4
    0x6d, // " 5
    0x7d, // " 6
    0x07, // " 7
    0x7f, // " 8
    0x67  // " 9
};
```

- **Array de patrones:** Este array contiene los patrones de bits correspondientes a cada dígito del 0 al 9. Estos son los patrones que creamos en el ejemplo anterior. Este es un método muy interesante, si notas, el primer elemento del array es el elemento 0 (recuerdas lo de la unidad 0?), y “casualmente” allí alojamos el byte con la combinación para el número 0.”**¡Vualá! tenemos una correspondencia**”. La posición dentro del array se corresponde con el número almacenado (o su combinación). Entonces cuando queramos mostrar un 0 llamamos a `bit[0]` y para el 1 `bit[1]` y así. ¿No es genial?

```
stdio_init_all();
printf("Hola 7 segmentos\n");
```

- **Inicialización:** Se inicializa la salida estándar para poder imprimir mensajes en la consola.

```
for (int gpio = FIRST_GPIO; gpio < FIRST_GPIO + 7; gpio++) {
    gpio_init(gpio);
    gpio_set_dir(gpio, GPIO_OUT);
}
```


➤ **Configuración de los GPIO:** Este bucle recorre los pines GPIO desde FIRST_GPIO hasta FIRST_GPIO+7 (un total de 8 pines). Para cada pin:

- gpio_init(gpio);: Inicializa el pin como un GPIO.
- gpio_set_dir(gpio, GPIO_OUT);: Configura el pin como salida.

```
int val = 0;
while (true) {
    ...
}
```

➤ **Bucle principal:** Este bucle se ejecuta infinitamente.

```
int32_t mask = bits[val] << FIRST_GPIO;
```

➤ **Creación de una máscara:** Se crea una máscara de bits a partir del patrón correspondiente al valor actual de val. El operador << desplaza los bits del patrón hacia la izquierda para alinearlos con los pines GPIO correctos.

```
gpio_set_mask(mask);
sleep_ms(250);
gpio_clr_mask(mask);
```

➤ **Control de los LEDs:**

- gpio_set_mask(mask);: Enciende los LEDs correspondientes al patrón actual.
- sleep_ms(250);: Espera 250 milisegundos.
- gpio_clr_mask(mask);: Apaga todos los LEDs.

```
val++;
if(val>=10){
    val=0;
}
```

➤ **Incremento del valor:** Se incrementa el valor de val para mostrar el siguiente dígito en el display. El if se asegura que val siempre esté entre 0 y 9.

En resumen, este código configura los pines GPIO de la Raspberry Pi Pico W como salidas, crea patrones de bits para representar los dígitos del 0 al 9 y luego muestra estos dígitos secuencialmente en el display de 7 segmentos.

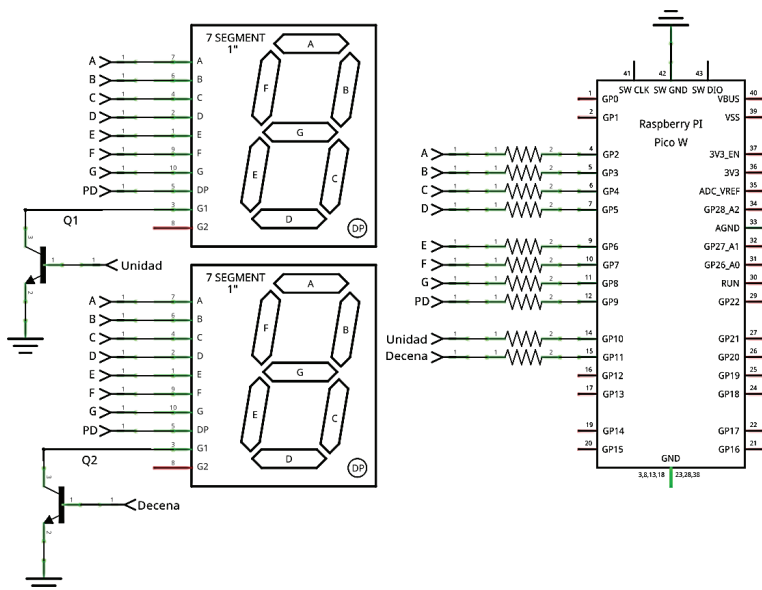
Consideraciones adicionales

- **Multiplexación:** Si se necesitan múltiples displays, se puede utilizar la técnica de multiplexación para controlarlos con un número limitado de pines.
- **Velocidad:** La velocidad a la que se pueden actualizar los displays depende de la frecuencia de reloj del microcontrolador y de la complejidad de los cálculos.
- **Brillo:** Se puede ajustar el brillo ajustando el valor de las resistencias o utilizando PWM (modulación por ancho de pulso).
- **Librerías existentes:** Existen librerías que facilitan el control de displays de 7 segmentos, como la librería `seven_segment.h`.

3.2.7 Práctica: Manejo de 2 o más display de 7 segmentos (multiplexación)

¿Qué es la multiplexación? Es una técnica que permite controlar múltiples dispositivos utilizando un número menor de pines. En este caso, utilizaremos los mismos pines GPIO para controlar ambos displays, pero los encenderemos y apagaremos alternadamente a una velocidad lo suficientemente rápida para que el ojo humano perciba ambos displays encendidos simultáneamente.

Esquemático



Código:

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

#define FIRST_GPIO 2
#define DISPLAY_UNIDAD 10
#define DISPLAY_DECENA 11

/* Este Array convierte un número del 0 al 9 en un patrón de bits para
enviar a los GPIO.*/
int bits[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f,
0x67};

int main() {
    stdio_init_all();
    printf("Hola 7segmentos\n");

    for (int gpio = FIRST_GPIO; gpio < FIRST_GPIO + 8; gpio++) {
        gpio_init(gpio);
        gpio_set_dir(gpio, GPIO_OUT);
    }

    // Inicializar los pines de habilitación de los displays
    gpio_init(DISPLAY_UNIDAD);
    gpio_set_dir(DISPLAY_UNIDAD, GPIO_OUT);
    gpio_init(DISPLAY_DECENA);
    gpio_set_dir(DISPLAY_DECENA, GPIO_OUT);

    int contador = 15;
    int unidad=0, decena=0;
    while (true) {
        printf("Bucle infinito: Contador=%u\n",contador);

        //Separamos las unidades de las decenas
        unidad = contador % 10;
        //El operador % (módulo) devuelve el resto de la división.
        decena = contador / 10;
        //Dividimos contador entre 10 para eliminar la cifra de las
```

```
unidades.  
    // Hacemos un corrimiento de la máscara según el pin de inicio.  
    int32_t mask_unidad = bits[unidad] << FIRST_GPIO;  
    int32_t mask_decena = bits[decena] << FIRST_GPIO;  
  
    for(int tiempo=0;tiempo<=1000;tiempo+=10){  
        // Encender el primer display y mostrar un dígito  
        gpio_put(DISPLAY_UNIDAD, 1);  
        gpio_put(DISPLAY_DECENA, 0);  
  
        //¡Configura todos nuestros GPIO de una sola vez!  
        gpio_set_mask(mask_unidad);  
        sleep_ms(5);  
        gpio_clr_mask(mask_unidad);  
  
        // Encender el segundo display y mostrar otro dígito  
        gpio_put(DISPLAY_UNIDAD, 0);  
        gpio_put(DISPLAY_DECENA, 1);  
  
        //¡Configura todos nuestros GPIO de una sola vez!  
        gpio_set_mask(mask_decena);  
        sleep_ms(5);  
        gpio_clr_mask(mask_decena);  
  
        gpio_put(DISPLAY_DECENA, 0);  
    }  
    contador++;  
    if(contador>=100){  
        contador=0;  
    }  
}  
}
```

Explicación de los cambios:

1. **Pines adicionales:** Se definen dos pines adicionales, DISPLAY_UNIDAD y DISPLAY_DECENA, para controlar la habilitación de cada display.
2. **Inicialización de los pines:** Se inicializan estos pines como salidas.
3. **Bucle principal:**

- Se alterna entre encender y apagar los pines de habilitación de cada display.
- Cuando un display está habilitado, se muestran los dígitos correspondientes en ese display.
- El tiempo de espera entre cada actualización (5 ms en este ejemplo) es crucial para lograr una buena visualización. Si el tiempo es demasiado corto, los displays parpadearán. Si es demasiado largo, la actualización será lenta.

Consideraciones adicionales:

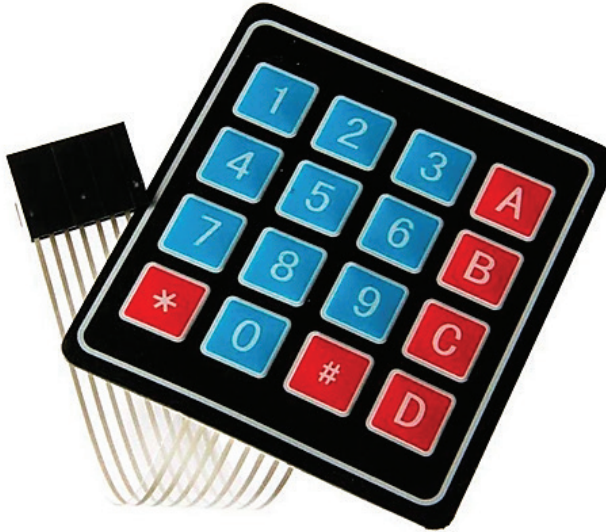
- **Frecuencia de actualización:** La frecuencia de actualización ideal dependerá de la persistencia de la visión humana y de la velocidad de respuesta del display.
- **Brillo:** Para aumentar el brillo aparente de los displays, se puede reducir el tiempo de apagado de cada display.
- **Consumo de corriente:** La multiplexación ayuda a reducir el consumo de corriente, ya que solo una parte de los segmentos está encendida en cada momento.
- **Sincronización:** Si se utilizan varios displays, es importante sincronizar la actualización para evitar parpadeos o distorsiones.

Personalización:

- **Números a mostrar:** Puedes modificar los valores en el array bits o agregar más elementos para mostrar diferentes números o caracteres.
- **Patrones de visualización:** Puedes crear patrones más complejos variando la secuencia de encendido y apagado de los displays.
- **Botones o interruptores:** Puedes agregar botones o interruptores para controlar el contenido de los displays.

En resumen, esta modificación permite controlar dos displays de 7 segmentos utilizando un número menor de pines GPIO, gracias a la técnica de multiplexación. Al ajustar los parámetros de tiempo y los patrones de visualización, puedes crear diferentes efectos y animaciones.

3.3 TECLADO MATRICIAL



3.3.1 Introducción

Un teclado matricial es un arreglo de botones organizado en filas y columnas, que permite controlar múltiples botones utilizando un número reducido de pines de entrada/salida. En este módulo, explicaremos cómo manejar un teclado matricial 4x4 en nuestra Raspberry Pi Pico.

3.3.2 Estructura y funcionamiento de un teclado matricial

Un teclado matricial 4x4 tiene 4 filas y 4 columnas, lo que nos permite controlar 16 botones utilizando solo 8 pines. Cada botón se encuentra en la intersección de una fila y una columna.

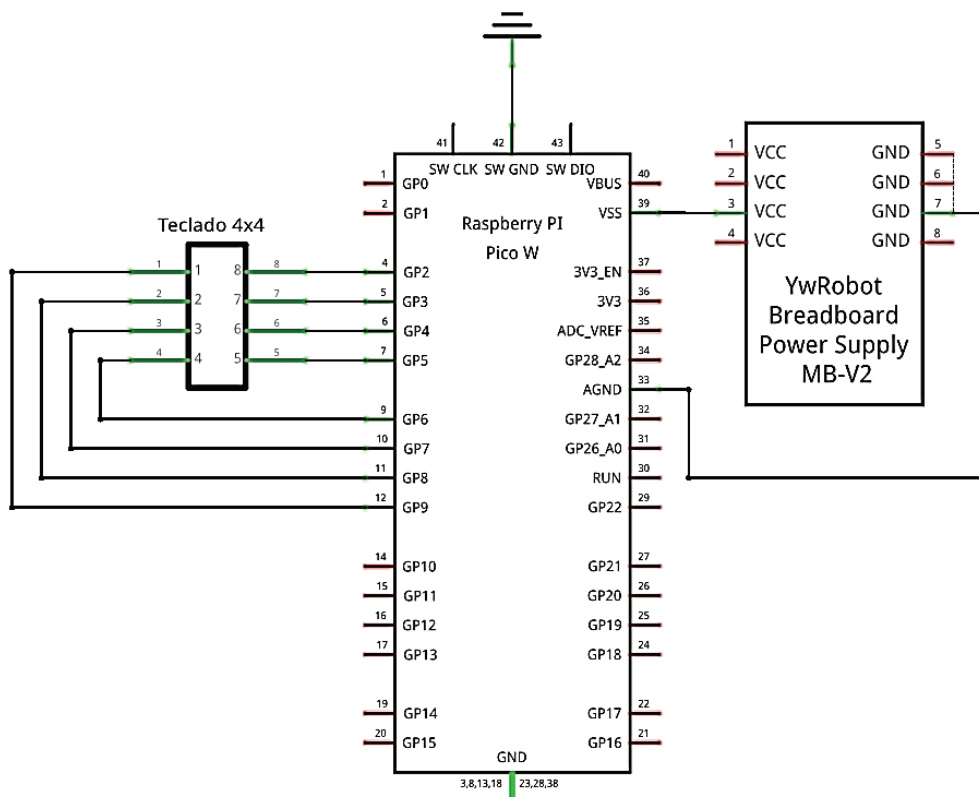
Funcionamiento:

1. **Escaneado de filas:** Se configura una fila como salida y el resto como entradas.
2. **Lectura de columnas:** Se leen los valores de las columnas. Si una tecla en esa fila está presionada, el valor de la columna correspondiente será bajo.

3. **Identificación de la tecla:** Combinando el número de fila y columna, se puede identificar la tecla presionada.

Resistencias Pull-up

Para detectar correctamente las pulsaciones de las teclas, es necesario conectar resistencias pull-up en cada fila. Estas resistencias mantienen los pines de entrada en alto cuando ninguna tecla está presionada. Al presionar una tecla, se crea un cortocircuito a tierra, lo que hace que el pin correspondiente se ponga bajo. En varios microcontroladores, como el RP2040, estas resistencias (pull-up o pull-down) vienen incluidas dentro de la circuitería interna del propio pin, con lo que se puede prescindir de agregarlas en el circuito físico, sólo hay que activar dichas resistencias por medio de configuración.



fritzing

3.3.3 Aplicaciones

Los teclados matriciales se utilizan en una amplia variedad de aplicaciones, como:

- Controladores remotos.
- Calculadoras.
- Juegos electrónicos.
- Interfaces de usuario para dispositivos embebidos.

3.3.4 Ventajas

- Reducción de pines: Permite controlar muchos botones con pocos pines.
- Simplicidad: El concepto es relativamente fácil de entender e implementar.
- Costo: Los teclados matriciales son económicos y fáciles de conseguir.

3.3.5 Desventajas

- Rebotes: Al presionar una tecla, puede ocurrir un rebote eléctrico que puede ser detectado como múltiples pulsaciones. Se requiere software adicional para filtrar estos rebotes.
- Límite de teclas: El número de teclas está limitado por el tamaño de la matriz.

3.3.6 Resumen

Los teclados matriciales son una forma eficiente de controlar múltiples botones utilizando un microcontrolador. Al comprender su funcionamiento y las técnicas de escaneo, podemos implementar interfaces de usuario intuitivas en nuestros proyectos.

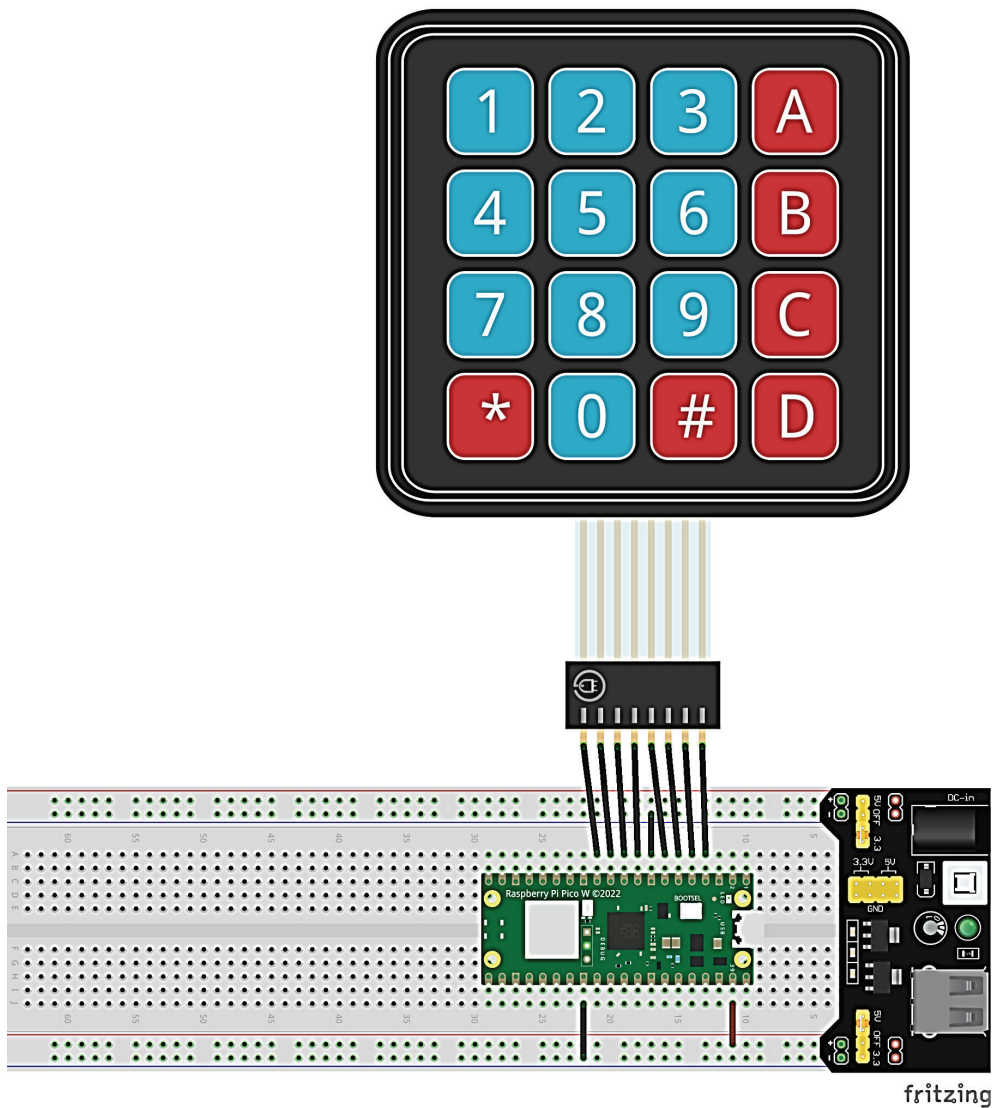
3.3.7 Práctica: manejo de teclado matricial 4x4

Hardware Necesario

- Raspberry Pi Pico W.
- Teclado matricial 4x4.
- Cables de conexión.

Conexión del teclado

1. **Conectar las columnas:** Conecta las columnas del teclado a los pines de salida de la Raspberry Pi Pico.
2. **Conectar las filas:** Conecta las filas del teclado a los pines de entrada de la Raspberry Pi Pico W.



Código en C

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

#define FILAS 4
#define COLUMNAS 4

const char teclas[FILAS][COLUMNAS] = {
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'},
};

int pinesFilas[FILAS]={2,3,4,5}; //Número de GPIO al que está conectada
cada fila
int pinesColumnas[COLUMNAS]={6,7,8,9}; //Número de GPIO al que está
conectada cada columna
char tecla_pulsada=0;

char leer_teclado(){
    char tecla;
    for (int c = 0; c < COLUMNAS; c++) {
        gpio_put(pinesColumnas[c], 0);

        for (int f = 0; f < FILAS; f++) {
            if (!gpio_get(pinesFilas[f])) {          // Tecla presionada
                tecla = teclas[f][c];
                gpio_put(pinesColumnas[c], 1);
                sleep_ms(200); // Retraso para eliminar rebote
                return tecla;
            }
        }

        gpio_put(pinesColumnas[c], 1); // Sube la columna
    }
    return tecla=0; // Ninguna tecla presionada
}

int main(){
    stdio_init_all();
```

```
for (int i=0; i<FILAS; i++) {
    gpio_init(pinesFilas[i]); //Inicializamos el GPIO de cada fila
    gpio_set_dir(pinesFilas[i], GPIO_IN); //Configuramos cada pin como
    entrada
    gpio_pull_up(pinesFilas[i]); //Activamos la resistencia PULLUP de
    cada pin
}

for (int i=0; i<COLUMNAS; i++) {
    gpio_init(pinesColumnas[i]); //Inicializamos el GPIO de cada
    columna
    gpio_set_dir(pinesColumnas[i], GPIO_OUT); //Configuramos cada pin
    como salida
}

while (true) {
    tecla_pulsada=leer_teclado(); //Leemos el teclado y guardamos el
    valor en la variable tecla_pulsada.
    if(tecla_pulsada !=0){ //Si el valor es distinto de 0,
    entonces se pulso una tecla.
        printf("Tecla Pulsada = %c\n",tecla_pulsada); //Imprimimos la
        tecla pulsada por consola
        tecla_pulsada=0; //Volvemos la variable a 0 para realizar otra
        lectura
    }
}
}
```

Explicación del código

- **Definición de constantes:** Se definen las constantes FILAS y COLUMNAS para representar el número de filas y columnas del teclado.
- **Función leer_teclado():** Esta función se encarga de escanear el teclado.
 - Recorre cada fila, configurando la columna actual como salida y el resto como entrada.
 - Lee el valor de cada fila. Si una tecla está presionada, el valor de la columna será bajo.

3.3.8 Mapeando el hardware: Arrays de dos dimensiones (matrices)

Hasta ahora estuvimos usando arrays de una sola dimensión, que imaginas como una fila de casilleros numerados. Sin embargo, para resolver problemas de hardware como el teclado matricial 4x4, el lenguaje C nos permite dar un paso más allá y crear **arrays de dos dimensiones**, comúnmente llamados **matrices** o tablas.

Si un array común es una línea de casilleros, una matriz es una **cuadrícula con filas y columnas**.

3.3.8.1 ¿CÓMO SE DECLARAN EN C?

Para definir una matriz, en lugar de usar un solo par de corchetes, usamos dos: el primero indica la cantidad de **filas** y el segundo la cantidad de **columnas**.

```
// Declaramos una matriz de 4 filas por 4 columnas para los caracteres
del teclado
char teclado[4][4] = {
    {'1', '2', '3', 'A'}, // Fila 0
    {'4', '5', '6', 'B'}, // Fila 1
    {'7', '8', '9', 'C'}, // Fila 2
    {'*', '0', '#', 'D'}  // Fila 3
};
```

3.3.8.2 ¿CÓMO TRABAJA EL MICROCONTROLADOR CON ESTO?

Para el procesador de la Pico, la memoria sigue siendo una línea continua de bytes, pero el compilador de C hace la matemática por nosotros. Cuando quieres leer qué carácter hay en una coordenada específica, le pasas ambos índices: `teclado[fila][columna]`.

- Si el programa detecta una pulsación en la **Fila 0** y la **Columna 2**, buscará `teclado[0][2]` y el microcontrolador te devolverá el carácter '3'.
- Si la pulsación ocurre en la **Fila 3** y la **Columna 0**, buscará `teclado[3][0]` y obtendrás el carácter '*'.

Recuerda siempre que, al igual que en los arrays simples, **en C los índices empiezan a contar desde el cero (0)**. Por lo tanto, una matriz de 4x4 tiene filas que van de la 0 a la 3, y columnas que van de la 0 a la 3.

3.4 PANTALLA LCD HD44780 (CONEXIÓN PARALELA)

3.4.1 Introducción

La pantalla LCD (Display de Cristal Líquido) HD44780 es uno de los componentes más utilizados en proyectos de electrónica debido a su bajo costo y facilidad de uso. Este módulo permite mostrar texto y caracteres especiales en una matriz de puntos. En este módulo, exploraremos cómo conectar y controlar una pantalla LCD HD44780 en modo paralelo con nuestra Raspberry Pi Pico.

3.4.2 Estructura y funcionamiento

La pantalla LCD HD44780 se compone de:

- **Cristales líquidos:** Estos materiales cambian su orientación cuando se les aplica un voltaje, permitiendo controlar la transmisión de luz.
- **Electrodos:** Se utilizan para aplicar voltajes a los cristales líquidos y formar los segmentos que componen los caracteres.
- **Controlador HD44780:** Este chip integrado controla la operación de la pantalla, interpretando comandos y generando las señales necesarias para mostrar la información.

