Raspberry pi pico W - I2C

- El desarrollo a continuación de Hizo en C/C++ usando el editor de código Visual Studio Code → Para esto se necesita configurar todo el entorno de desarrollo: How to Set Up Visual Studio Code to Program the Pi Pico (Windows)

Conexión I2C : VIM3 - Raspberry pi pico W

- Vea el esquema : conexion_i2c_vim3_rppw.excalidraw , estos son los pines necesarios que se usan en el desarrollo, puede dirigirse al Datasheet y definir otros siempre y cuando tengan la función de I2C.

Configuración I2C

- La Raspberry Pi Pico W al contrario de la VIM3 , no requiere de la manipulacón de archivos de configuración, dado que aquí la programación se hace Bare Metal. Esto quiere decir que no tenemos montado un sistema operativo como Linux o alguna distro de este, por lo que el código debe subirse compilado al dispositivo y este se encarga de ejecutarlo.
- La configuración I2C como esclavo de hace con la ayuda que nos brindan las API del Pico C SDK.
- La API usada es pico_i2c_slave, La cual nos provee una interfaz de esclavo I2C que se encarga de manejar las interrupciones. Recomiendo ir y leer las funciones que nos brinda para entender el código implementado.

Código implementado para la Prueba de comunicación

- Con el código a continuación corriendo en la Raspberry pi pico, solo queda hacer la prueba de comunicación desde la VIM3 como maestro hacia la Pi pico como esclavo.
- Vea el código implementado en la VIM3, el cual se encargará de establecer la comunicación, enviar data, solicitar data al Esclavo además de imprimir en pantalla el flujo para validar el funcionamiento: VIM3 - I2C > Código implementado para la prueba de comunicación

```c
// Inclusión de librerías necesarias -> api pico : High level APIs

#include <hardware/i2c.h>
#include <pico/i2c_slave.h>
#include <pico/stdlib.h>
#include <stdio.h>
#include <string.h>

//Defino el address de la pico como esclavo
static const uint I2C_SLAVE_ADDRESS =  0X17;
static const uint I2C_BAUDRATE      =  100000; // 100kHz
static const uint I2C_SLAVE_SDA_PIN = PICO_DEFAULT_I2C_SDA_PIN; // 4
static const uint I2C_SLAVE_SCL_PIN = PICO_DEFAULT_I2C_SCL_PIN; // 5
// Defino una estructura para manejar los estados de la I2C en modo esclavo
static struct
{
    uint8_t mem[256]; // Un buffer de 256 Bytes
    uint8_t mem_address; // Dirección en la que va al momento de escribir o leer
    bool mem_address_written; // Bandera para indicar que está escribiendo
} context;

// Definición del handler, el cual es llamado desde el I2C_ISR el cual debe completarse lo más rapido posible.Esta función es la que como desarrollador debemos modificar según nuestras necesidades, la librería nos brinda estos eventos para manejar el esclavo de una manera muy cómoda.
static void i2c_slave_handler(i2c_inst_t *i2c, i2c_slave_event_t event) {
    switch (event) {

    case I2C_SLAVE_RECEIVE: // El maestro ha escrito algo de información
        if (!context.mem_address_written) {
            // Las escrituras siempre comienzan con la dirección de memoria
            context.mem_address = i2c_read_byte_raw(i2c);
```

```
            context.mem_address_written = true;
        } else {
            // Guarda en la memoria
            context.mem[context.mem_address] =
i2c_read_byte_raw(i2c);
            context.mem_address++;
        }
        break;

    case I2C_SLAVE_REQUEST: // El maestro solicita data
        // carga la data desde la memoria/buffer
        // El mem_address es enviado por el maestro en una llamada
anterior
        i2c_write_byte_raw(i2c, context.mem[context.mem_address]);
        context.mem_address++;
        break;

    case I2C_SLAVE_FINISH: // master ha enviado la señal stop /
restart
        context.mem_address_written = false;
        break;

    default:
        break;
    }
}

// Configuración del esclavo
static void setup_slave(){

    // Inicializo el GPIO SDA
    gpio_init(I2C_SLAVE_SDA_PIN);
    // Seteo el GPIO en modo I2C
    gpio_set_function(I2C_SLAVE_SDA_PIN, GPIO_FUNC_I2C);
    // Activo el GPIO con resistencia en PULL UP
    gpio_pull_up(I2C_SLAVE_SDA_PIN);

    // Inicializo el GPIO DEL SCL
    gpio_init(I2C_SLAVE_SCL_PIN);
    // Seteo el GPIO en modo I2C
```

```
    gpio_set_function(I2C_SLAVE_SCL_PIN, GPIO_FUNC_I2C);
    // Activo el GPIO con resistencia en PULL UP
    gpio_pull_up(I2C_SLAVE_SCL_PIN);

    // Inicializo el canal I2C con su respectiva velocidad de
transferencia
    i2c_init(i2c0, I2C_BAUDRATE);

    // Inicio el manejador del esclavo y le asigno su dirección de
memoria
    i2c_slave_init(i2c0, I2C_SLAVE_ADDRESS, &i2c_slave_handler);
}


int main(){
    stdio_init_all();
    puts("\nI2C slave first test");
    setup_slave();

    while (true)
    {
        tight_loop_contents();
    }

}
```