VIM3 - I2C

Configuración I2C en La VIM 3

Ver la documentación del I2C en la VIM3 → Enable I2C
Ver configuración del device-tree-overlay → Device Configurations
Ver el uso del I2C → I2C Usage
Ver → GPIO header pinout

- Según la documentación, para habilitar el I2C en la VIM3, debemos ir y editar el archivo en la ruta **/boot/env.txt** y agregar el nodo al **overlays node** si este no existe:

```txt
overlays = uart4 pwm_f wathdog i2c3
```

- Después de hacer un reboot, verá el nodo del dispositivo I2C.

```shell
$ ls /dev/i2c-3
/dev/i2c-3
```

- Esta configuración habilitará los GPIO para que se use I2C, en caso de necesitar usarlos para otra funcionalidad se debe remover el nodo I2C del Device Tree Overlay

Configuración Pines GPIO y Conexión con Raspberry pi pico W

- A continuación se muestran los pines que se eligieron en el proyecto, en caso de querer usar otros, vaya a la documentación.
- Ver conexión con la Raspberry pi pico W → conexion_i2c_vim3_rppw.excalidraw
- En caso de definir una interfaz i2c diferente para la VIM3 y decida usar otros pines, diríjase al Datasheet

PIN	Función	Descripción
21	GND	Tierra que debe de ir con la Tierra de la Pi pico
22	I2C_M3_SCL	Clock que debe ir al SCL de la Pi Pico
23	I2C_M3_SDA	Datos que debe ir al SDA de la Pi Pico

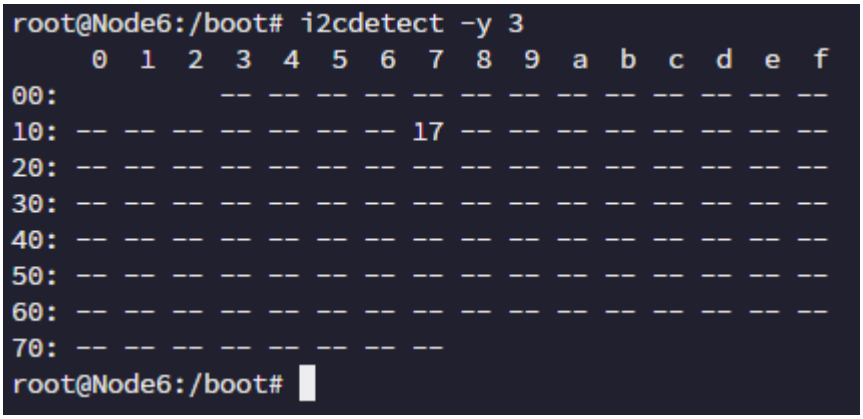
PIN	Función	Descripción
27	3.3V	Pin que alimenta a la Pi pico

Prueba de funcionamiento del I2C

- Una ves habilitado el I2C y configurada la Raspberry pi pico W - I2C y conectada a la VIM, se puede probar desde la VIM3 si esta detecta correctamente la Raspberry en su dirección de esclavo definida:
- En la consola de la VIM3 ingrese el siguiente comando, el cual detectará si hay un esclavo conectado y nos mostrará en que dirección se encuentra:

```shell
$ i2cdetect -y 3
```

- Al ejecutar el comando, tenemos una salida como la que vemos abajo, la cual nos mostrará la dirección donde se encuentra conectado un dispositivo a través de I2C, en este caso sale "17" la cual le definimos a la Raspberry pi pico.



Código implementado para la prueba de comunicación

- El código a continuación, establece una comunicación I2C y envía datos según la memoria definida en ambos lados de la conexión, también hace solicitud de datos al esclavo y todo esto con el fin de validar que se este escribiendo y leyendo la data correctamente, validándolo con los printf que hay en el código.
- Se dejan ejemplos del Código tanto en C como en Python y además el código que está implementado para enviar la data en un formato JSON:

C

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <string.h>
#include <stdint.h>


#define I2C_DEVICE "/dev/i2c-3" // Reemplaza con el número de I2C correcto habilitado en el device overlay tree de la VIM3, dado que la libreria va y busca el directorio y hace uso del dispositivo i2c que encuentra habilitado
#define I2C_SLAVE_ADDRESS 0x17  // Direcciónd el esclavo

int main() {
    int file;
    char buf[32];
    // Abrir el dispositivo I2C
    if ((file = open(I2C_DEVICE, O_RDWR)) < 0) {
        perror("Failed to open the i2c bus");
        exit(1);
    }

    // Configurar la dirección del esclavo
    if (ioctl(file, I2C_SLAVE, I2C_SLAVE_ADDRESS) < 0) {
        perror("Failed to acquire bus access and/or talk to slave");
        exit(1);
    }

    for (uint8_t mem_address = 0;; mem_address = (mem_address + 32) % 256) {
        char msg[32];
        snprintf(msg, sizeof(msg), "Hello, I2C slave! - 0x%02X", mem_address);
        uint8_t msg_len = strlen(msg);
        buf[0] = mem_address;
        memcpy(buf + 1, msg, msg_len);
```

```
        // Escribir mensaje en la dirección de memoria del esclavo
        if (write(file, buf, 1 + msg_len) != 1 + msg_len) {
            perror("Failed to write to the i2c bus");
            continue;
        }
        printf("Write at 0x%02X: '%s'\n", mem_address, msg);

        // Leer datos del esclavo
        if (write(file, buf, 1) != 1) {
            perror("Failed to set read address");
            continue;
        }

        // Leer parcialmente
        uint8_t split = 5;
        if (read(file, buf, split) != split) {
            perror("Failed to read from the i2c bus");
            continue;
        }
        buf[split] = '\0';
        printf("Read  at 0x%02X: '%s'\n", mem_address, buf);

        if (read(file, buf, msg_len - split) != msg_len - split) {
            perror("Failed to read remaining from the i2c bus");
            continue;
        }
        buf[msg_len - split] = '\0';
        printf("Read  at 0x%02X: '%s'\n", mem_address + split, buf);
        sleep(2); // Espera antes de la siguiente iteración
    }

    close(file);
    return 0;

}
```

PYTHON

```python
import smbus
import time

I2C_SLAVE_ADDRESS = 0x17
I2C_BUS = 3  # Reemplaza con el número de I2C correcto
"""Esto indica que estás usando el bus I2C número 3 en tu VIM3. La
VIM3 tiene varios buses I2C disponibles, y el número 3 corresponde a
uno de ellos. Si necesitas usar un bus diferente, deberías cambiar
este número y habilitarlo desde el Device Overlay Tree como se indica
en la guia."""


def main():
    # Inicializar el bus I2C
    bus = smbus.SMBus(I2C_BUS)
    mem_address = 0
    while True:
        msg = f"Hello, I2C slave! - 0x{mem_address:02X}"
        msg_bytes = list(msg.encode('ascii'))
        try:
            # Escribir mensaje en la dirección de memoria del esclavo
            bus.write_i2c_block_data(I2C_SLAVE_ADDRESS, mem_address,
msg_bytes)
            print(f"Write at 0x{mem_address:02X}: '{msg}'")
            # Leer datos del esclavo
            split = 5
            read_data = bus.read_i2c_block_data(I2C_SLAVE_ADDRESS,
mem_address, split)

            print(f"Read  at 0x{mem_address:02X}:
'{bytes(read_data).decode('ascii')}'")

            read_data = bus.read_i2c_block_data(I2C_SLAVE_ADDRESS,
mem_address + split, len(msg) - split)

            print(f"Read  at 0x{mem_address + split:02X}:
'{bytes(read_data).decode('ascii')}'")

        except IOError as e:
```

```
            print(f"Error de I/O: {e}")


        time.sleep(2)  # Espera antes de la siguiente iteración

        mem_address = (mem_address + 32) % 256

if __name__ == "__main__":
    main()
```

- Código para enviar data en formato JSON a través de I2C y solicitarla al esclavo para validar su funcionamiento:

PYTHON

```python
import smbus #Libreria para la comunicación I2C
import time  #Libreria para poner a dormir durante x tiempo
import json  #Libreria para convertir a formato JSON

I2C_SLAVE_ADDRESS = 0x17
I2C_BUS = 3  # Reemplaza con el número de I2C correcto


def send_dict_over_i2c(bus, address, data_dict):
    """
    Envía un diccionario sobre I2C dividiéndolo en chunks.

    Esta función toma un diccionario, lo convierte a JSON, y
    luego lo envía en chunks de 32 bytes sobre I2C a un dispositivo
esclavo.
    Después de enviar cada chunk, intenta leer los datos enviados
para validar
    la transmisión.

    Parámetros:
    bus (SMBus): Objeto SMBus inicializado para la comunicación I2C.
    address (int): Dirección I2C del dispositivo esclavo.
    data_dict (dict): Diccionario con los datos a enviar.

    Comportamiento:
    1. Convierte el diccionario a una cadena JSON.
    2. Codifica la cadena JSON a bytes UTF-8.
    3. Divide los bytes en chunks de 32 bytes.
    4. Envía cada chunk al dispositivo esclavo usando
write_i2c_block_data.
    5. Después de cada envío, intenta leer los datos del dispositivo
esclavo para validar.
    6. Imprime información sobre cada chunk enviado y leído.
    """

    # Convertir el diccionario a una cadena JSON
    json_str = json.dumps(data_dict)
    # Convertir la cadena JSON a bytes
    json_bytes = json_str.encode('utf-8') # aqui se puede hacer
```

```
encode de varías maneras
    print(json_bytes)
    # Dividir los bytes en chunks de 32 bytes (o menos para el último
chunk)
    chunk_size = 32
    for i in range(0, len(json_bytes), chunk_size):
        chunk = json_bytes[i:i+chunk_size]
        try:
            bus.write_i2c_block_data(address, i, list(chunk))
            print(f"Enviado chunk {i//chunk_size + 1}: {chunk}")
        except IOError as e:
            print(f"Error al enviar chunk {i//chunk_size + 1}: {e}")
        time.sleep(0.1)  # Pequeña pausa entre chunks


        # BLOQUE DE CÓDIGO PARA VALIDAR LA INFORMACIÓN GUARDADA EN LA
RASPBERRY PI PICO :D
        try:
            read_data = bus.read_i2c_block_data(I2C_SLAVE_ADDRESS, i,
chunk_size)
            read_data =  bytes(read_data).decode('utf-8') # Paso de
Bytes a utf-8
            if('}' in list(read_data)):

                print(f"Read  at 0x{i:02X}:
'{read_data[:read_data.index('}')+1]}'")
            else:
                print(f"Read  at 0x{i:02X}: '{read_data}'")


        except IOError as e:
            print(f"Error :  {e}")

def main():
    # Inicializar el bus I2C
    bus = smbus.SMBus(I2C_BUS)

    # Ejemplo json que se envía
    mensaje_dic = {
        "tipo": "Audio",
```

```
        "deteccion": "sirena",
        "fecha": "2023-11-24T13:37:00Z",
        "nivel_confianza": 0.95
    }

    while True:
        try:
            send_dict_over_i2c(bus, I2C_SLAVE_ADDRESS, mensaje_dic)
            print("Mensaje enviado correctamente")
        except Exception as e:
            print(f"Error al enviar el mensaje: {e}")

        time.sleep(5)  # Espera 5 segundos antes de enviar el
siguiente mensaje

if __name__ == "__main__":
```