UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS
PROYECTOS DIGITALES AVANZADOS

PRÁCTICA N°6:
PROTOCOLOS DE COMUNICACIÓN ALÁMBRICOS

Profesor:

Pedro Rene
Cabrera

Bachiller:

Claudia Rodríguez
C.I: 27.943.668

Barcelona, noviembre de 2024

1. OBJETIVO DE LA PRÁCTICA

1.1. Enviar y recibir un mensaje por cada uno de los protocolos de comunicación alambricos: UART, SPI e I2C. Ademas, mostrar en la pantalla OLED lo transmitido/recibido.

1.2. Realizar la asignación de pines correspondientes para cada uno de los protocolos, así mismo, realizar la conexión correcta dado que los protocolos I2C y UART requieren resistencias pullup con el objetivo de reducir el ruido en la transmisión de datos.

1.3. Inicializar el Raspberry Pi Pico W para que actúe como receptor / esclavo dependiendo del protocolo que se este implementando..

1.4. Otros agregados realizados fueron:

1.4.1. Con el protocolo SPI realizar una parábola indicándole los coeficientes.

1.4.2. Con el protocolo I2C y definido un maestro y esclavo, el esclavo al recibir el mensaje, lo devuelve al maestro con caracteres añadidos.

1.4.3. Con el protocolo UART, mejorar la transmisión haciendo que en la pantalla OLED se visualice mejor el mensaje enviado.

2. DESARROLLO

2.1. Planteamiento del problema

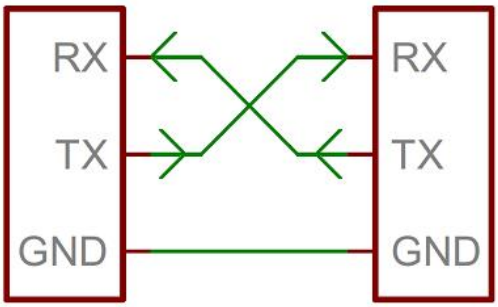
La transmisión de datos bit a bit entre dos Raspberry Pi Pico tiene varios factores a considerar. Uno de ellos es establecer un formato de codificación de caracteres compatible con ambos dispositivos y ademas convertirlo a bits. Además, es necesario comprender los protocolos de comunicación física, incluyendo los pines específicos para cada protocolo y las frecuencias de transmisión de datos. Al realizar la conexión física, se encontraron algunos problemas específicos en los protocolos I2C y UART. En el caso de I2C, la falta de una resistencia pull-up con un valor mínimo de 1kΩ puede causar errores de conexión. En el protocolo UART, fue la inversión de la conexión. Los códigos construidos están basados en programación: modular, funcional y orientada a objetos, debido a que cada código no se extendió a la más de 120 líneas, se considero más simple mencionar las bibliotecas y funciones utilizadas en cada .py.

2.2. Solución, bibliotecas y funciones utilizadas:

2.2.1. Solución UART: Primeramente, se tuvo que definir una función denominada **uart_pin()** con el objetivo que al ser invocada devuelva el objeto de tipo **uart** el cual va a ser utilizado conjunto a los comandos write() en el transmisor y read() y any() en el

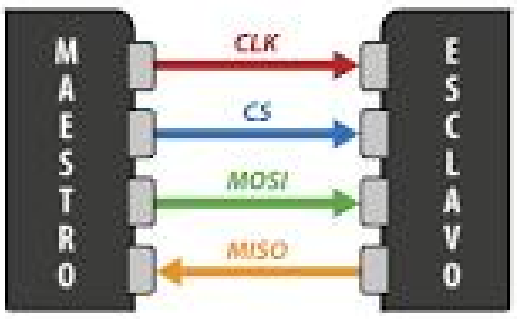
receptor. Ya una vez transmitido y recibido el mensaje se muestra en pantalla el contenido de dicho mensaje. No hizo falta el uso de alguna biblioteca externa que no se haya implementado en códigos anteriores.

- ✓ **Bibliotecas utilizadas:** machine, ssd1306 y time
- ✓ **Funciones del transmisor:** uart_pin(), crear_oled(), mostrar_oled(oled, message), enviar(oled, uart) y main()
- ✓ **Funciones del receptor:** uart_pin(), crear_oled(), mostrar_oled(oled, message), recibir(uart) y main()



En el código se designo a ambos picos los pines GPIO 12 como TX y el GPIO 13 como RX. Con respecto al cableado, se invierte la conexión como se muestra en la figura.

2.2.2. Solución SPI: Para el protocolo SPI se necesita una librería externa, tanto para el pico maestro (*spi_master.py)* como para el pico esclavo *(spi_slave.py).* El objetivo de dicho código era el envío del mensaje y ademas de ello, poder realizar una gráfica de segundo grado con los coeficientes enviados (parábola).



Para el Pico maestro los GPIOs utilizados fueron: CLK (18) CS (17), MOSI (19), y MISO (16)

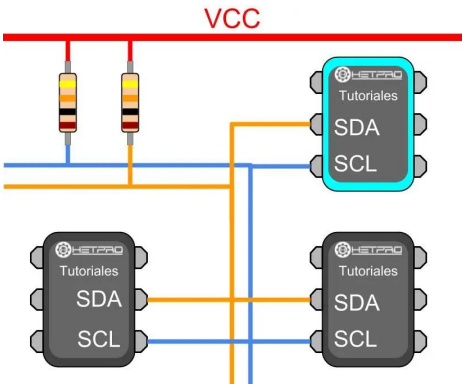
Para el Pico esclavo los GPIOs utilizados fueron: CLK (27) , CS (28), MOSI (26), y MISO (22)

- ✓ **Bibliotecas utilizadas:** math, array, spi_master, time, machine, ssd1306 y spi_slave
- ✓ **Funciones del transmisor:** crear_oled(), mostrar_oled(oled, message), transmission(oled) y main()

- ✓ **Funciones del receptor:** crear_oled(), mostrar_oled(oled, message), archivo(), spi_pin(), recibir(file, slave), graficar_parabola(oled, numeros) y main()

2.2.3. Solución I2C: Inicialmente, el protocolo I2C presentaba problemas para mostrar en la pantalla del esclavo la información recibida, para corregir el problema se colocaron los mismos GPIOS que se utilizaban en el maestro para utilizar la pantalla OLED, ya luego, se realizaba la transmisión y la visualización de manera correcta. Ademas de implementar la comunicación I2C, se modifica el mensaje original enviado de maestro a esclavo, añadiéndole caracteres al inicio de la cadena y se devuelve al maestro.

- ✓ **Bibliotecas utilizadas:** machine, ssd1306, time e i2cSlave
- ✓ **Funciones del transmisor y receptor:** i2c_transmision_pin(), crear_oled(), i2c_pin_recepcion(), mostrar_oled(oled, message), envio_i2c(message, i2c_trans), recepcion_i2c(s_i2c) y main()



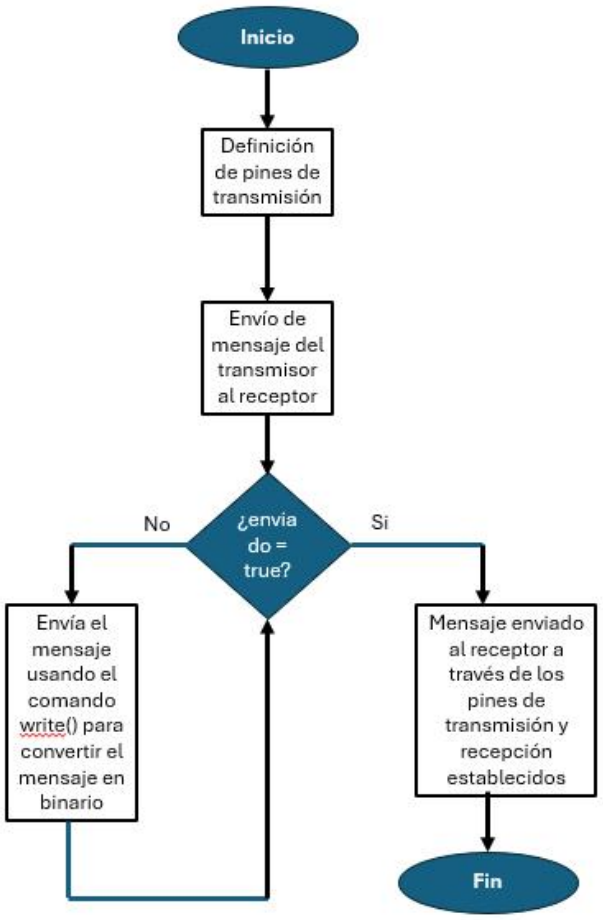
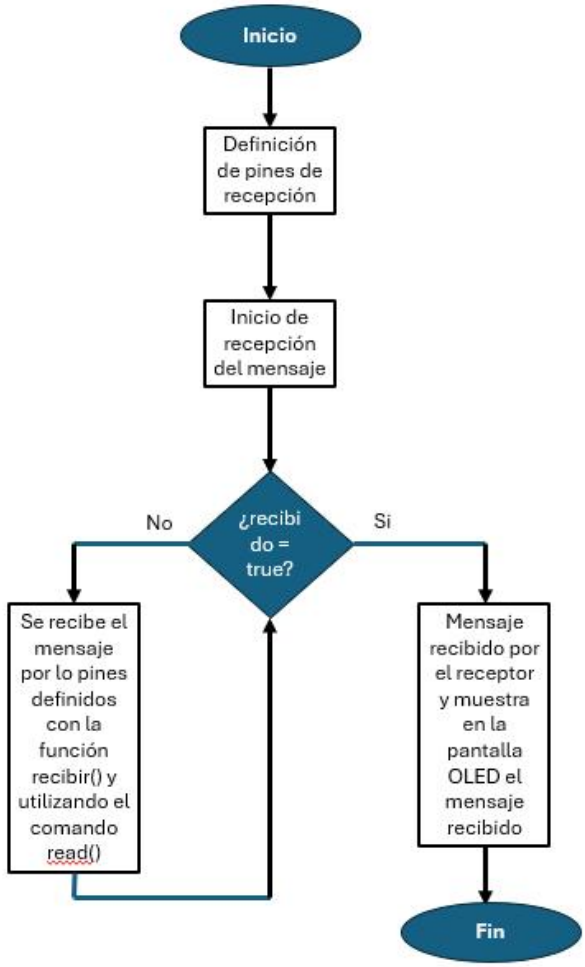
En ambos Picos, tanto maestro como esclavo se definieron los GPIOS 1 como línea SCL y GPIO 0 como SDA. Se utilizaron resistencias pull-up.

2.3. Tabla de comandos utilizados (en relación a los protocolos)

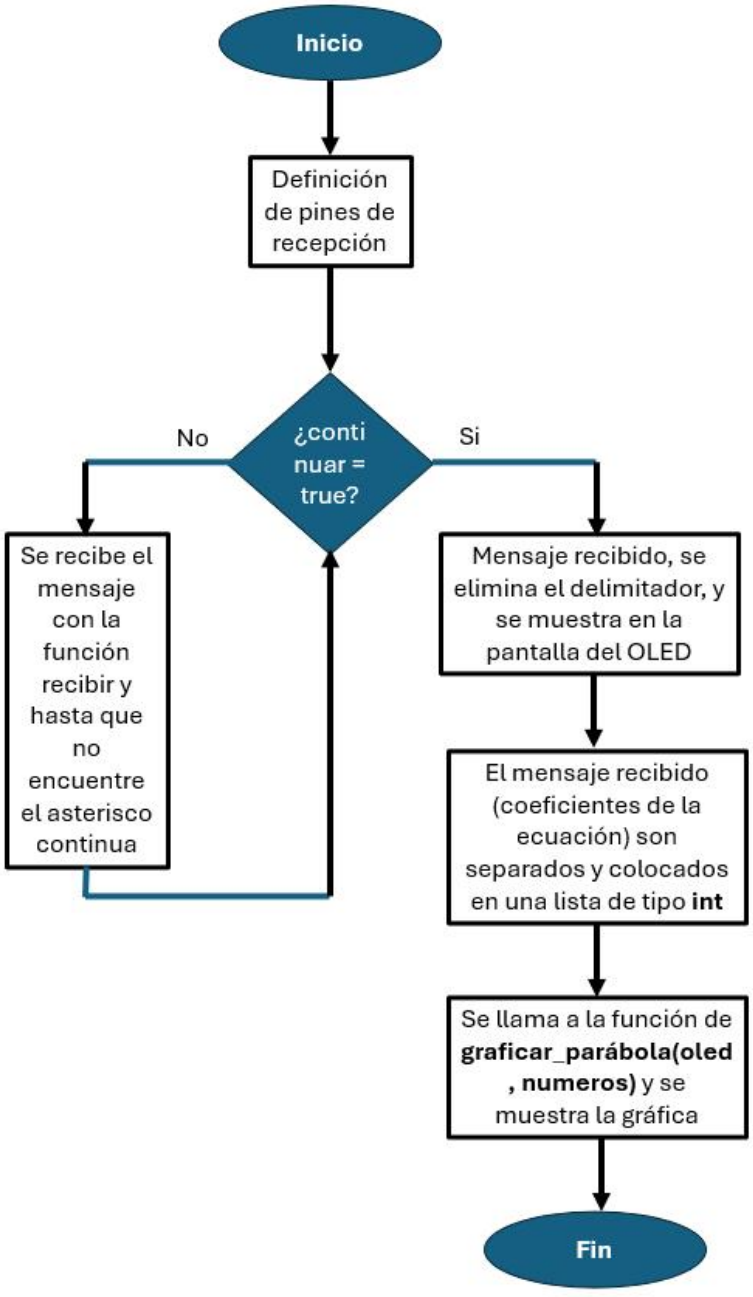
Comando	Función	Utilizado en:
read()	Se encarga de leer los datos del puerto serie.	UART
any()	Comprueba si hay datos para leer en el puerto.	UART
write(str)	Envia datos al puerto escogido.	UART
rx_words()	Extraido del spi_slave. Devuelve una referencia al array que contiene los datos que se van a recibir	SPI
tx_words()	Extraido del spi_slave. Devuelve una referencia al array que contiene los datos que se van a enviar	SPI
get()	Obtiene un dato especifico de los datos recibidos	I2C
writeto()	Se uso para enviar los datos al pico esclavo. writeto(direccion, datos)	I2C

2.4. Diagrama de flujo

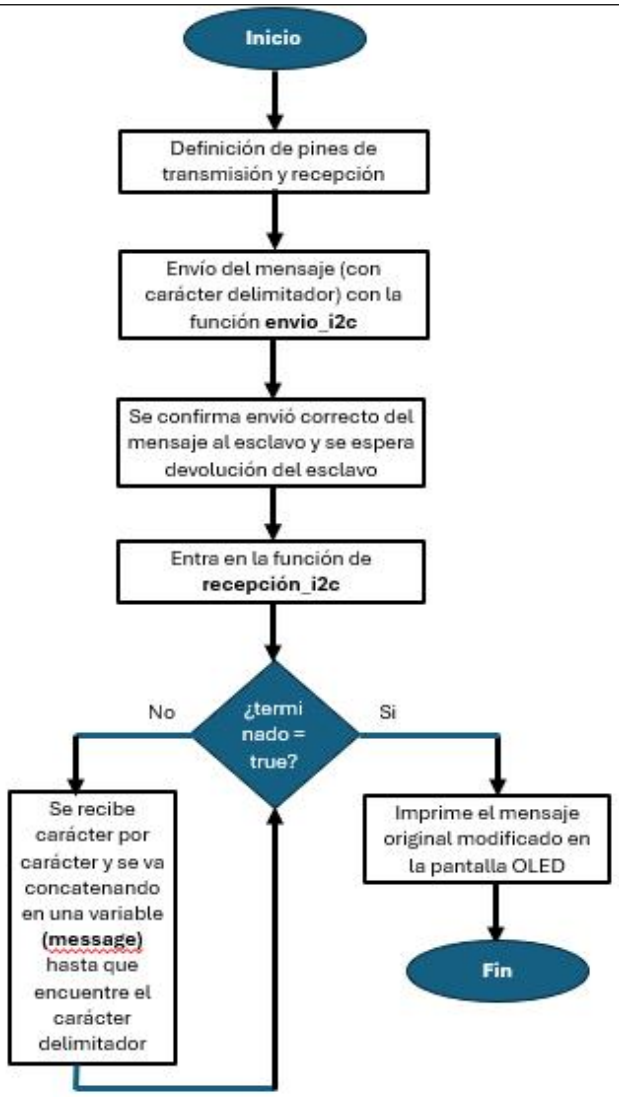
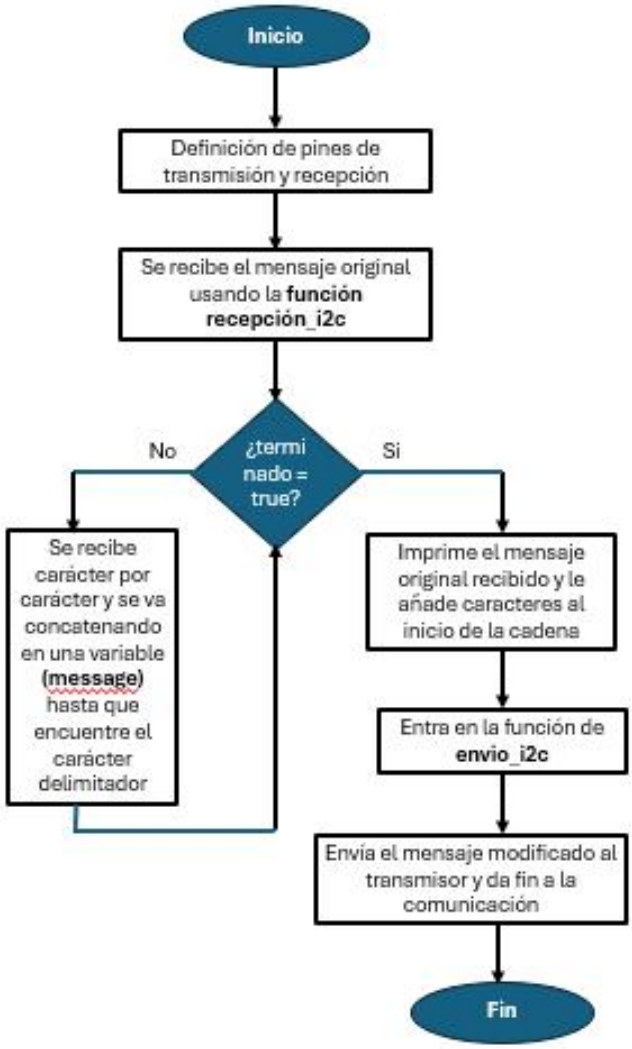
2.4.1. UART: Envío de mensaje desde Pico normal a Pico W

Transmisor	Receptor
	

2.4.2. SPI: Envío de mensaje desde Pico normal a Pico W

Transmisor	Receptor
	

2.4.3. I2C: Envío de mensaje desde Pico normal a Pico W y devolución de Pico W a Pico normal

Transmisor	Receptor
	

3. ANEXOS

3.1. Código

3.1.1. UART: Envío de mensaje desde Pico normal a Pico W

Transmisor	Receptor

from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
from time import sleep
 	
def **uart_pin():**
  uart = machine.UART(0, baudrate=9600, tx=machine.Pin(12), rx=machine.Pin(13))
  return uart
 	
def **crear_oled()**:
  i2c = I2C(1, scl=Pin(15), sda=Pin(14), freq=400000) ## <-- pendiente
  oled = SSD1306_I2C(128,64,i2c)
  return oled
 	
def **mostrar_oled(oled, message)**:
  oled.fill(0)
 	
  ancho_caracter = 7 # <-- tamaño en pixeles de un caracter
  max_columna = 120 # <-- long max de la pantalla
  fila = 0 # <-- primera linea
 	
  columna = 0 # <-- primera columna
 	
  palabras = message.split() # <-- string a lista
 	
  for palabra in palabras:
  ancho_palabra = len(palabra) * ancho_caracter
  if columna + ancho_palabra > max_columna:
  fila += 16
  columna = 0
 	
  if fila >= 50:
  oled.show()
  sleep(5)
  oled.fill(0)
  fila = 0
	from machine import Pin, I2C
	from ssd1306 import SSD1306_I2C
	from time import sleep
	
	def **uart_pin()**:
	  uart = machine.UART(0, baudrate=9600, tx=machine.Pin(12), rx=machine.Pin(13))
	  return uart
	
	def **crear_oled()**:
	  i2c = I2C(0, scl=Pin(17), sda=Pin(16), freq=400000) ## <-- pendiente
	  oled = SSD1306_I2C(128,64,i2c)
	  return oled
	
	def **mostrar_oled(oled, message):**
	  oled.fill(0)
	
	  ancho_caracter = 7 # <-- tamaño en pixeles de un caracter
	  max_columna = 120 # <-- long max de la pantalla
	  fila = 0 # <-- primera linea
	
	  columna = 0 # <-- primera columna
	
	  palabras = message.split() # <-- string a lista
	
	  for palabra in palabras:
	  ancho_palabra = len(palabra) * ancho_caracter
	  if columna + ancho_palabra > max_columna:
	  fila += 16
	  columna = 0
	
	  if fila >= 50:
	  oled.show()
	  sleep(5)
	  oled.fill(0)
	  fila = 0

```
            columna = 0

        oled.text(palabra, columna, fila)
        columna = columna + 7 # --> espacio entre palabras porsia

        columna += ancho_palabra + ancho_caracter

    oled.show()
    sleep(5)

def enviar(oled, uart):
    enviado = False
    while not enviado:
        message = input("Ingrese el mensaje a enviar: ")
        oled.fill(0)
        cadena = "El mensaje enviado fue: "
        mostrar_oled(oled, cadena)
        oled.show()
        sleep(2)
        mostrar_oled(oled, message)
        uart.write(message)
        print("Se ha enviado el mensaje: "+message+" a el receptor.")
        enviado = True
        sleep(5)

def main():
    uart = uart_pin()
    oled = crear_oled()
    enviar(oled, uart)
    cadena = "El mensaje fue recibido por el receptor. "
    mostrar_oled(oled, cadena)

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

```
            columna = 0

        oled.text(palabra, columna, fila)
        columna = columna + 7 # --> espacio entre palabras porsia

        columna += ancho_palabra + ancho_caracter

    oled.show()
    sleep(5)

def recibir(uart):
    recibido = False
    while not recibido:
        if uart.any():
            message = uart.read()
            cant = len(message)
            if len(message) > 0:
                recibido = True
                return message

def main():
    uart = uart_pin()
    oled = crear_oled()
    message = recibir(uart)
    print("Mensaje recibido:", message)
    oled.fill(0)
    cadena = "ESPERANDO MENSAJE"
    mostrar_oled(oled, cadena)
    oled.show()
    sleep(2.5)
    mostrar_oled(oled, message)
    cadena = "El mensaje fue visualizado en la pantalla OLED"
    mostrar_oled(oled, cadena)

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

3.1.2. SPI: Envío de mensaje desde Pico normal a Pico W

Transmisor	Receptor
import math import array as arr from spi_master import SPI_Master from time import sleep	import array as arr from spi_slave import SPI_Slave from machine import Pin, I2C from ssd1306 import SSD1306_I2C

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

def crear_oled():
    i2c = I2C(1, scl=Pin(15), sda=Pin(14), freq=400000) ## <-- pendiente
    oled = SSD1306_I2C(128,64,i2c)
    return oled

def mostrar_oled(oled, message):
    message = message.rstrip("*") ## <-- para eliminar el asterisco
    asterisco = False
    #print(message)
    if message == "*":
        asterisco = True
        oled.fill(0)
        oled.show()
        cadena = "FIN DEL MENSAJE"
        mostrar_oled(oled, cadena)

    while not asterisco:
        oled.fill(0)

        ancho_caracter = 7  # <-- tamaño en pixeles de un caracter
        max_columna = 120  # <-- long max de la pantalla
        fila = 0          # <-- primera linea

        columna = 0                # <-- primera columna

        # metodo split para convertir un string
        # en una lista
        palabras = message.split()

        for palabra in palabras:
            ancho_palabra = len(palabra) * ancho_caracter
            if columna + ancho_palabra > max_columna:
                fila += 16
                columna = 0
            if fila >= 50:
                oled.show()
                sleep(5)
                oled.fill(0)
                fila = 0
```

```
from time import sleep
import math

def crear_oled():
    i2c = I2C(0, scl=Pin(17), sda=Pin(16), freq=400000) ## <-- pendiente
    oled = SSD1306_I2C(128,64,i2c)
    return oled

def mostrar_oled(oled, message):
    espacio = False
    message = message.rstrip("*") ## <-- para eliminar el asterisco

    if message == "":
        oled.fill(0)
        oled.show()
        cadena = "FIN DEL MENSAJE"
        mostrar_oled(oled, cadena)
        espacio = True

    while not espacio:
        oled.fill(0)

        ancho_caracter = 7  # <-- tamaño en pixeles de un caracter
        max_columna = 120  # <-- long max de la pantalla
        fila = 0          # <-- primera linea

        columna = 0                # <-- primera columna

        # metodo split para convertir un string
        # en una lista
        palabras = message.split()

        for palabra in palabras:
            ancho_palabra = len(palabra) * ancho_caracter
            if columna + ancho_palabra > max_columna:
                fila += 16
                columna = 0

            if fila >= 50:
                oled.show()
                sleep(5)
                oled.fill(0)
```

```
            columna = 0

        oled.text(palabra, columna, fila)
        columna = columna + 7 # --> espacio entre palabras porsia

        columna += ancho_palabra + ancho_caracter
    asterisco = True
    oled.show()
    sleep(5)

def transmision(oled):
  SPI_WORDS = 4
  SPI_BYTES = SPI_WORDS * 4

  csv_file = "enviar_spi.csv"
  word_buffer = arr.array("I")

  with open(csv_file, "r") as file:
    csv_data = file.read()
    mostrar_oled(oled, csv_data)
    csv_bytes = csv_data.encode("utf-8")
    print(csv_bytes)
    for i in range(0, len(csv_bytes), SPI_BYTES):
      chunk = csv_bytes[i:i+SPI_BYTES]
      if len(chunk) < SPI_BYTES:
        chunk += b"\x00" * (SPI_BYTES - len(chunk))
      word_buffer.extend(arr.array("I", [int.from_bytes(chunk[j:j+4], "big") for j in range(0, SPI_BYTES, 4)]))

  master = SPI_Master(mosi_pin=19, miso_pin=16, sck_pin=18, csel_pin=17, spi_words=SPI_WORDS, F_SPI=1_000_000)

  for i in range(0, len(word_buffer), SPI_WORDS):
    block = word_buffer[i:i+SPI_WORDS]
    master.write(block)
    sleep(1)

  print("Transmision completada")
  cadena = "Transmision completada"
  mostrar_oled(oled, cadena)
```

```
            fila = 0
            columna = 0

        oled.text(palabra, columna, fila)
        columna = columna + 7 # --> espacio entre palabras porsia

        columna += ancho_palabra + ancho_caracter
    espacio = True
    oled.show()
    sleep(5)

def archivo():
  file_path = "recibir_spi.csv"
  file = open(file_path, "w")
  return file

def spi_pin():
  SPI_WORDS = 4
  word_buffer = arr.array("I", [0] * SPI_WORDS)
  slave = SPI_Slave(csel=28, mosi=26, sck=27, miso=22, spi_words=SPI_WORDS, F_PIO=10_000_000)
  return slave

def recibir(file, slave):
  read = slave.rx_words()
  write = slave.tx_words()
  data_bytes = bytearray()
  for word in read:
    data_bytes.extend(word.to_bytes(4, "big"))

  file.write(data)
  file.flush()
  for i, word in enumerate(read):
    write[i] = word
  slave.put_words()
  return data

def graficar_parabola(oled, numeros):
  oled_width  = 128
  oled_height = 64
  a = numeros[0]
  b = numeros[1]
  c = numeros[2]
  a = float(a/100)
```

```
def main():
    oled = crear_oled()
    oled.fill(0)
    oled.show()
    cadena = "MENSAJE A ENVIAR POR SPI"
    mostrar_oled(oled, cadena)
    transmision(oled)
    sleep(1.5)
    cadena = "FIN DEL MENSAJE"
    mostrar_oled(oled, cadena)

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

```
    b = float(b/100)
    c = float(c/100)

    while True:
        oled.fill(0)
        oled.hline(0, oled_height // 2, oled_width, 1)
        oled.vline(oled_width // 2, 0, oled_height, 1)
        for x in range(-64, 64):
            y = a * (x * x) + b * x + c
            pixel_y = int(oled_height // 2 - y * 10)
            if 0 <= pixel_y < oled_height:
                oled.pixel(x + (oled_width // 2), pixel_y, 1)
        oled.show()
        sleep(1)

def main():
    oled = crear_oled()
    oled.fill(0)
    oled.show()
    cadena = "MENSAJE A RECIBIR POR SPI"
    mostrar_oled(oled, cadena)
    slave = spi_pin()
    file = archivo()
    continuar = False
    mensaje = ""
    while not continuar:
        if slave.received():
            mensaje += recibir(file, slave)
            if "*" in mensaje:
                continuar = True
    mensaje = mensaje.split('*')[0]
    print("Mensaje recibido: ", mensaje)
    mostrar_oled(oled, mensaje)
    sleep(1.5)
    cadena = "FIN DEL MENSAJE"
    mostrar_oled(oled, cadena)
      mensaje = mensaje.split('/')
    numeros = []
    for i in range (len(mensaje)):
        numeros.append(int(mensaje[i]))
    print(numeros)
    graficar_parabola(oled, numeros)

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

3.1.3. I2C: Envío de mensaje desde Pico normal a Pico W y devuelve al transmisor con caracteres añadidos al inicio de la cadena

Transmisor	Receptor

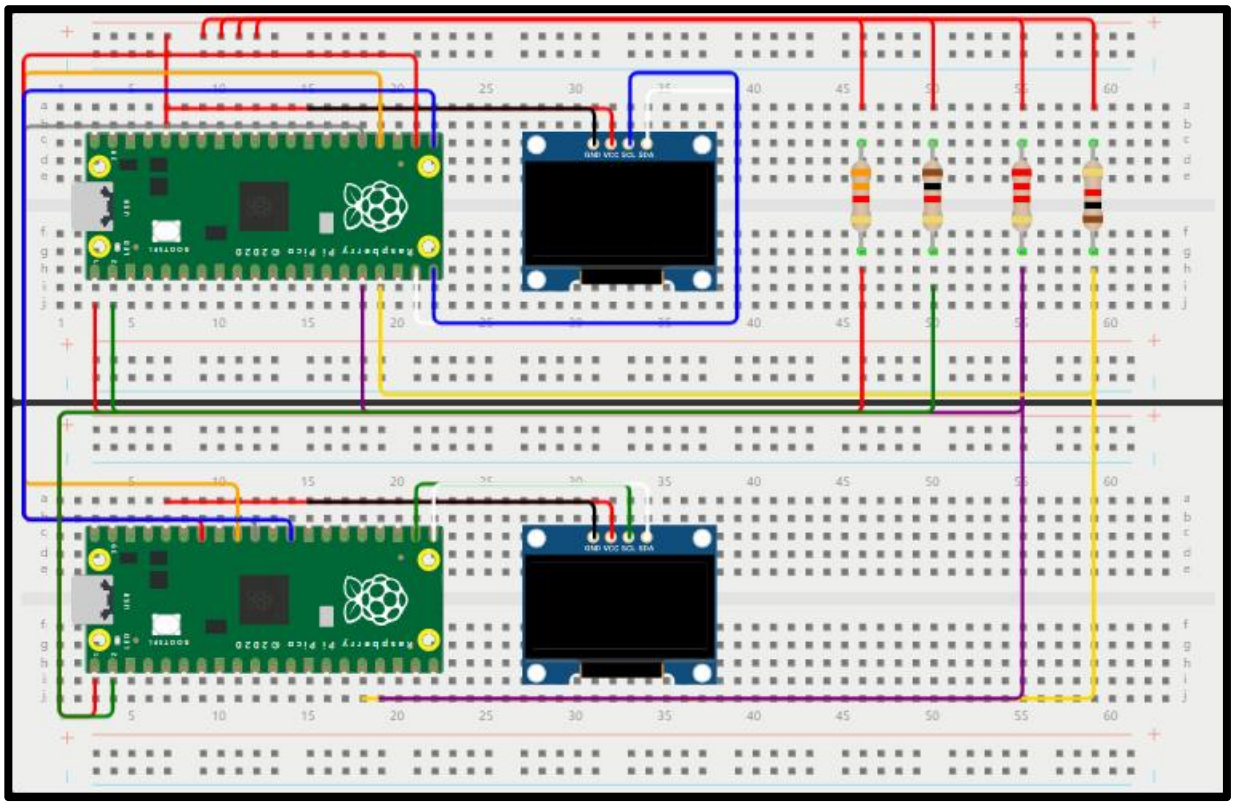
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
from time import sleep
from i2cSlave import i2c_slave
 	
def **i2c_transmision_pin():**
  i2c_trans = machine.I2C(0, scl=machine.Pin(1), sda=machine.Pin(0), freq=1000000)
  return i2c_trans
 	
def **i2c_pin_recepcion():**
  s_i2c = i2c_slave(0,sda=0,scl=1,slaveAddress=0x41)
  return s_i2c
 	
def **crear_oled():**
  i2c = I2C(1, scl=Pin(15), sda=Pin(14), freq=400000) ## <-- pendiente
  oled = SSD1306_I2C(128,64,i2c)
  return oled
 	
def **mostrar_oled(oled, message):**
  oled.fill(0)
  ancho_caracter = 7 # <-- tamaño en pixeles de un caracter
  max_columna = 120 # <-- long max de la pantalla
  fila = 0 # <-- primera linea
 	
  columna = 0 # <-- primera columna
 	
  palabras = message.split() # <-- string a lista
 	
  for palabra in palabras:
  ancho_palabra = len(palabra) * ancho_caracter
  if columna + ancho_palabra > max_columna:
  fila += 16
  columna = 0
	from machine import Pin, I2C
	from ssd1306 import SSD1306_I2C
	from time import sleep
	from i2cSlave import i2c_slave
	
	def **i2c_transmision_pin()**:
	  i2c_trans = machine.I2C(0, scl=machine.Pin(1), sda=machine.Pin(0), freq=1000000)
	  return i2c_trans
	
	def **i2c_pin_recepcion()**:
	  s_i2c = i2c_slave(0,sda=0,scl=1,slaveAddress=0x41)
	  return s_i2c
	
	def **crear_oled():**
	  i2c = I2C(1, scl=Pin(15), sda=Pin(14), freq=400000) ## <-- pendiente
	  oled = SSD1306_I2C(128,64,i2c)
	  return oled
	
	def **mostrar_oled(oled, message**):
	  oled.fill(0)
	  ancho_caracter = 7 # <-- tamaño en pixeles de un caracter
	  max_columna = 120 # <-- long max de la pantalla
	  fila = 0 # <-- primera linea
	  columna = 0 # <-- primera columna
	  palabras = message.split() # <-- string a lista
	
	  for palabra in palabras:
	  ancho_palabra = len(palabra) * ancho_caracter
	  if columna + ancho_palabra > max_columna:
	  fila += 16
	  columna = 0
	
	  if fila >= 50:
	  oled.show()

```
        if fila >= 50:
            oled.show()
            sleep(5)
            oled.fill(0)
            fila = 0
            columna = 0

        oled.text(palabra, columna, fila)
        columna = columna + 7 # --> espacio entre palabras porsia

        columna += ancho_palabra + ancho_caracter

    oled.show()
    sleep(5)

def envio_i2c(message, i2c_trans):
    device_address = 0x41 # <-- dir esclavo
    message = message + " *" #<-- caracter delimitador
    # <-- por seguridad, envia el mensaje por partes
    message_parts = [message[i:i+10] for i in range(0, len(message), 10)]
    for part in message_parts:
        message_bytes = bytearray(part, "utf-8")
        i2c_trans.writeto(device_address, message_bytes)
        sleep(0.05) #<-- obligatorio delay pq si no no envia bien

def recepcion_i2c(s_i2c):
    message = ""
    terminado = False
    while not terminado:
        data = s_i2c.get()
        char = chr(int(hex(data), 16))
        message += char
        if char == "*":
            terminado = True
            #print("Mensaje recibido:", message[:-1]) # Imprime el mensaje completo sin el '*'
            message = message.rstrip(" *")
            break # Sale del bucle
    return message

def main():
    message = "son las 9:16am y estamos en
            sleep(5)
            oled.fill(0)
            fila = 0
            columna = 0

        oled.text(palabra, columna, fila)
        columna = columna + 7 # --> espacio entre palabras porsia

        columna += ancho_palabra + ancho_caracter

    oled.show()
    sleep(5)

def recepcion_i2c(s_i2c):
    message = ""
    terminado = False
    while not terminado:
        data = s_i2c.get()
        char = chr(int(hex(data), 16))
        message += char
        if char == "*":
            terminado = True
            #print("Mensaje recibido:", message[:-1]) # Imprime el mensaje completo sin el '*'
            message = message.rstrip(" *")
            break # Sale del bucle
    return message

def envio_i2c(message, i2c_trans):
    device_address = 0x41 # <-- dir esclavo
    message = message + " *" #<-- caracter delimitador
    # <-- por seguridad, envia el mensaje por partes
    message_parts = [message[i:i+10] for i in range(0, len(message), 10)]
    for part in message_parts:
        message_bytes = bytearray(part, "utf-8")
        i2c_trans.writeto(device_address, message_bytes)
        sleep(0.05) #<-- obligatorio delay pq si no no envia bien

def main():
    oled = crear_oled()      # <-- objeto para oled
    oled.fill(0)
```

```
clase de proyectos digitales avanzados"
    i2c_trans = i2c_transmision_pin() # <-- objeto para transmitir
    oled = crear_oled()       # <-- objeto para oled
    oled.fill(0)
    oled.show()
    cadena = "MENSAJE I2C A ENVIAR: "
    print("MENSAJE ENVIADO POR I2C")
    print(message)
    mostrar_oled(oled, cadena)
    sleep(2)
    mostrar_oled(oled, message)
    sleep(2)

    envio_i2c(message, i2c_trans)
    cadena = "MENSAJE ENVIADO POR I2C"
    mostrar_oled(oled, cadena)
    print("Enviado")

    s_i2c = i2c_pin_recepcion()
    mensaje = recepcion_i2c(s_i2c)
    print(mensaje)
    mostrar_oled(oled, mensaje)

    s_i2c = i2c_pin_recepcion()
    mensaje = recepcion_i2c(s_i2c)
    print(mensaje)
    mostrar_oled(oled, mensaje)

    cadena = "FINALIZADO"

    mostrar_oled(oled, cadena)

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

```
    s_i2c = i2c_pin_recepcion()
    mensaje = recepcion_i2c(s_i2c)
    print("MENSAJE I2C A RECIBIR: ")
    print(mensaje)
    print("Recibido")

    oled.fill(0)
    oled.show()
    cadena = "MENSAJE I2C A RECIBIR: "
    mostrar_oled(oled, cadena)
    mostrar_oled(oled, mensaje)
    cadena = "MENSAJE RECIBIDO POR I2C"
    mostrar_oled(oled, cadena)

    # add añade al inicio de la cadena
    add = "123-56+"
    mensaje = str(mensaje)
    mensaje = add + mensaje

    cadena = "MENSAJE RECIBIDO POR EL RECEPTOR. GRACIAS "
    mostrar_oled(oled, cadena)

    i2c_respuesta = i2c_transmision_pin()

    envio_i2c(mensaje, i2c_respuesta)

    cadena = "FINALIZADO"

    mostrar_oled(oled, cadena)


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

3.2. Diagrama circuital



3.3. Bibliografía

Using I2C devices with Raspberry PI Pico and MicroPython. Enlace: https://peppe8o.com/using-i2c-devices-with-raspberry-pi-pico-and-micropython/

How to set up Raspberry Pi Pico as I2C slave. Enlace: https://python-academia.com/en/raspberry-pi-pico-slave/

Connecting 2 pico cards via SPI bus. Enlace: https://forums.raspberrypi.com/viewtopic.php?t=320599

Using UART between a Raspberry Pi Pico and Raspberry Pi 3b (Raspbian). Enlace: https://timhanewich.medium.com/using-uart-between-a-raspberry-pi-pico-and-raspberry-pi-3b-raspbian-71095d1b259f

Wokwi - World's most advanced ESP32 Simulator. Enlace: https://wokwi.com/