UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA
DEPARTAMENTO DE INGENIERÍA EN COMPUTACIÓN

PRACTICAS

LABORATORIO DE MICROCOMPUTADORAS

BASADAS EN LA PLATAFORMA RASPBERRY PI

RUBÉN ANAYA GARCÍA
MOISES MELENDEZ REYES
ANTONIO SALVA CALLEJA
JOSE ANTONIO ARREDONDO GARZA
ANGELICA QUIÑONES JUAREZ
DIANA CRUZ HERNANDEZ
LUIS SERGIO DURÁN ARENAS
ROMÁN OSORIO COMPARAN
ALBERTO TEMPLOS CARBAJAL

CIUDAD UNIVERSITARIA
PE108223

Este material y la infraestructura generada fueron obtenidos a través del apoyo de **DGAPA** al proyecto **PAPIME PE108223**
"*Prácticas de Laboratorio de microcomputadoras basadas en las plataformas Raspberry Pi***"**

Introducción

Se ha diseñado este material para uso en el laboratorio de microcomputadoras, con el objetivo de reforzar los conocimientos adquiridos en la teoría de la materia “Microcomputadoras”, se realizarán un conjunto de 12 prácticas. En cada una de ellas se presenta información introductoria, sugerida como inducción al tema a tratar; así como las actividades a propuestas y las complementarias que reforzarán la práctica.

El marco teórico, así como las actividades podrán ser aplicables a cualesquiera de las versiones de Raspberry Pi existentes actualmente. Nos enfocaremos en las versiones más empleadas como Raspberry Pi 3, Raspberry Pi 4, y Raspberry Pi 5 así como de la Raspberry Pi Pico, los ejercicios aquí propuestos inducen y llevan al estudiante a un mejor entendimiento de estos dispositivos y a un mayor aprovechamiento de estos.

Se realizarán dos prácticas que nos permitirá aprender la programación en ensamblador de los procesadores ARM, empleando directamente la Raspberry Pi, se empleará el Entorno de Desarrollo Integrado (Code::Blocks) para la edición, simulación y ejecución de los ejercicios requeridos. En estas prácticas se familiarizará con el conjunto de instrucciones y los diferentes modos de direccionamiento disponibles en estas arquitecturas.

A partir de la práctica tres y subsecuentes será empleada la versión Raspberry Pi Pico, se ha adoptado la programación en Python, desarrollada para uso en plataformas con microcontroladores, conocida como Micropython. En esta práctica se realizarán actividades empleando las terminales GPIO en la modalidad de salida.

La práctica cuatro, usará programación de los GPIO en las configuraciones de entrada y salida.

En la práctica cinco se controlarán motores de corriente directa, motores a pasos y servomotores, además se entenderá la importancia del uso de drivers de potencia.

La práctica seis mostrará al alumno las funcionalidades del convertidor analógico-digital como parte de los recursos del microcontrolador, además de ampliar las posibilidades de aplicación de este recurso y se realizarán acciones de control por medio de la modulación de ancho de pulso conocida como PWM.

En la práctica siete realizará la comunicación serie en la modalidad asíncrona, el estudiante controlará acciones desde y hacia la Raspberry Pi Pico, de manera alámbrica o inalámbrica a través de la UART.

La práctica ocho estudiará la comunicación serie síncrona a través del protocolo SPI.

En la práctica nueve se realizarán programas en los cuales se experimentará con dispositivos que sean controlados por medio de un mínimo de terminales; conocido como One Wire.

La práctica diez se ha enfocado en el aprendizaje de la comunicación serie bajo el protocolo I2C, con el cual se aprenderá la versatilidad de este tipo de comunicación.

Por último, las practicas 11 y 12 tendrán como objetivo el aprendizaje de funciones avanzadas de la plataforma Raspberry Pi Pico; la primera de ellas fomentará el control de sistemas por medio de

interrupciones y la programación de hilos(threads). En la última práctica, hará uso de la comunicación WiFi, conectarse a la red y adquirir la habilidad de realizar control de tipo IOT.

Además de la realización de las practicas, el estudiante deberá entregar un proyecto final; en el que plasme todo lo aprendido en el curso, este será el entregable como evidencia requerida para los procesos de acreditación.

Al concluir las prácticas y el proyecto final, el estudiante habrá experimentado con todos y cada uno de los recursos contenidos en una microcomputadora y un microcontrolador, empleando plataformas de última generación que a nivel nacional e internacional son utilizados en el diseño y construcción de sistemas digitales, aplicables en una gran variedad de áreas de la tecnología y para efectos didácticos: en la academia.

El manual contiene las siguientes:

Practicas

	Contenido
Práctica No. 1	Introducción de las arquitecturas ARM empleando Raspberry Pi.
Práctica No. 2	Programación en ensamblador; direccionamiento indirecto.
Práctica No. 3	Introducción a la plataforma Raspberry Pi Pico, GPIO como salida; programación en Micropython.
Práctica No. 4	GPIO como Entrada/Salida.
Práctica No. 5	Control de actuadores con GPIO.
Práctica No. 6	Convertidor Analógico Digital, control PWM.
Práctica No. 7	Comunicación Serie Asíncrona UART.
Práctica No. 8	Comunicación Serie Síncrona SPI.
Practica No. 9	Comunicación One Wire.
Práctica No. 10	Comunicación Serie Síncrona I2C.
Practica No. 11	Programación de Interrupciones y Threads.
Práctica No. 12	Uso y aplicaciones WiFi.
Evaluación	Proyecto Final.

Las actividades propuestas en este manual, buscan contribuir con los criterios de desempeño correspondientes a:

- *A2-CD3 (Desarrollo de proyectos que satisfacen las necesidades especificadas).*
- *A3-CD3 (presenta conclusiones con base en su formación ingenieril).*
- *A4-CD1(Elabora trabajos y reportes técnicos relacionados con sus asignaturas).*

Cumpliendo con el objetivo general de la asignatura de nombre: "*Microcomputadoras"*:

Objetivo.

El alumno aprenderá y aplicará los conocimientos de la teoría y funcionamiento de los microprocesadores y su interconexión con diferentes circuitos periféricos para la construcción y funcionamiento de microcomputadoras. Diseñará y construirá aplicaciones utilizando microprocesadores y sus periféricos para diferentes sistemas, simulando aplicaciones industriales en tiempo real, así como aplicaciones científicas.

Cubriendo en su totalidad el temario del curso; el cual consta de:

Temario.

1. *Conceptos básicos*
2. *Conjunto de instrucciones*
3. *Modos de direccionamiento*
4. *Señales de control y diseño de un sistema con microprocesadores*
5. *Periféricos e interfaces con microprocesadores*
6. *Técnicas de diseño de sistemas con microprocesadores*
7. *Características generales de microprocesadores de 16 y 32 bits*

Laboratorio de Microcomputadoras
Practica No. 1
Introducción a las arquitecturas ARM y Raspberry Pi

Objetivo. Aprender la estructura de los procesadores con arquitectura ARM, utilizar la plataforma Raspberry Pi, los entornos para programar; desarrollar algoritmos con las instrucciones en lenguaje ensamblador, controlar directamente los recursos del microprocesador; editar, compilar, ensamblar, simular y ejecutar programas en Raspberry Pi.

1. Introducción

Las plataformas Raspberry Pi están diseñadas con diferentes versiones de procesadores con arquitecturas ARM; en todos los casos es aplicable a su marco teórico de esta tecnología.

a. Registros

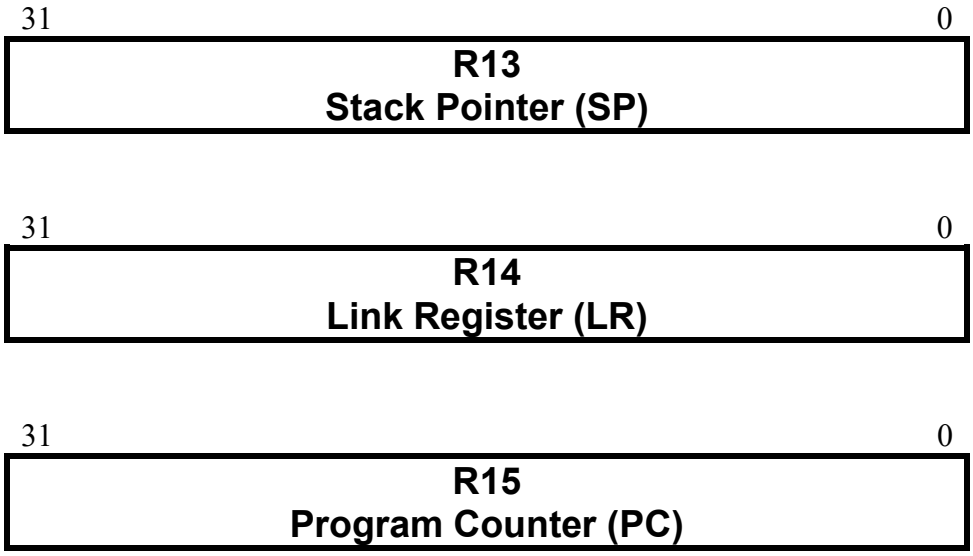
Tiene *16* registros de *32* bits identificados como *R0, R1, R2, … R15* y un *registro de banderas*; los registros R0 a R12 son registros de propósito general; mientras que R13, R14 y R15 son de propósito específico.

Registros de propósito general:

31 0

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12

Registros de propósito específico:



Registro de Banderas Estado/Control **CPSR**

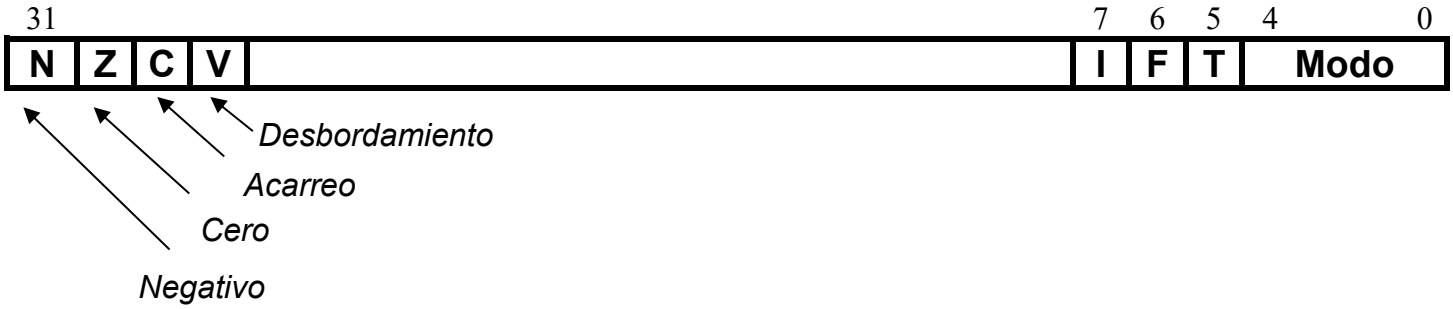


Figura 1-1. Registros ARM

b. Mapa de Memoria

Las arquitecturas ARM tienen un bus de direcciones de 32 bits; por lo que podrán direccionar hasta 4 GB, de 8 bits de espacio en memoria.

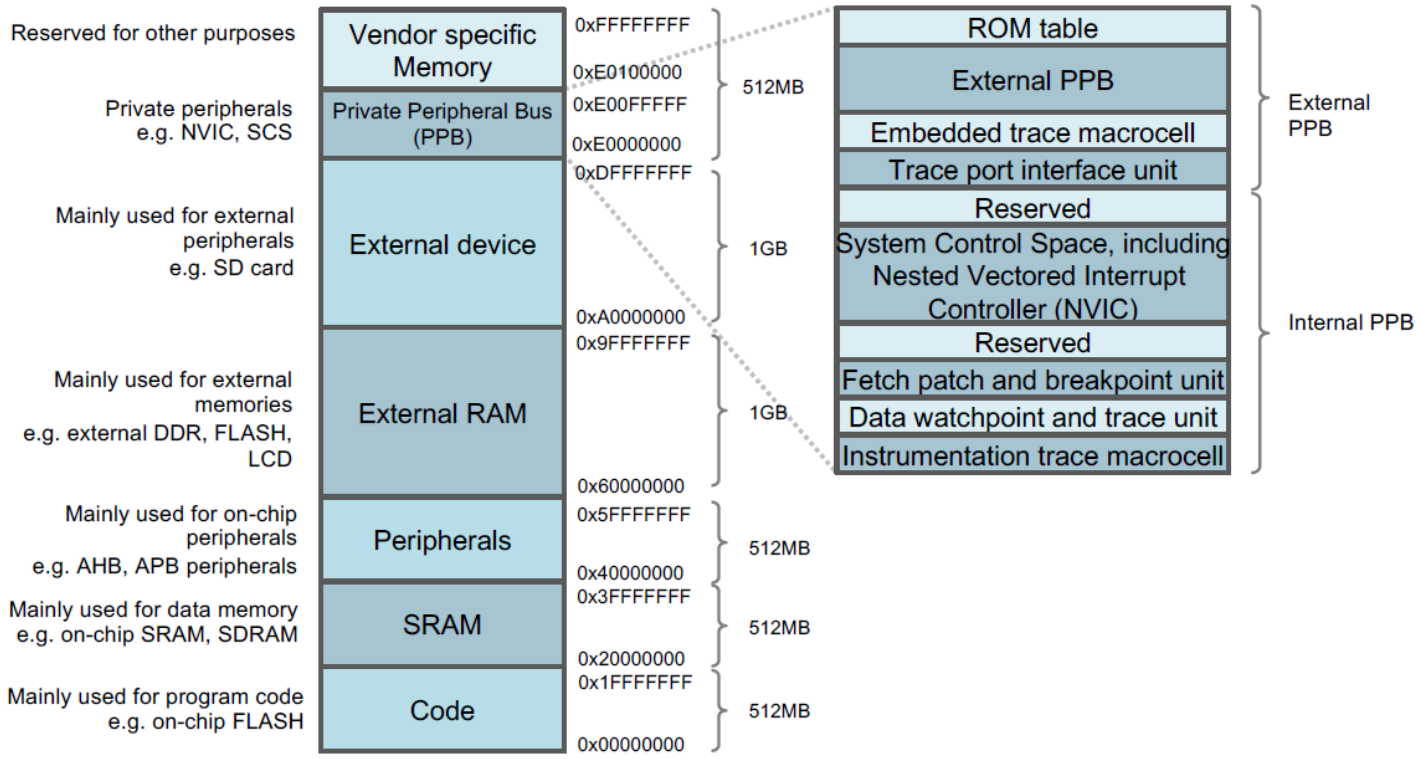


Figura 1-2. Mapa de memoria

c. Conjunto de instrucciones

A continuación, se muestra un extracto del conjunto de instrucciones; *consultar la documentación oficial para mayor información*.

Mnemonic	Instruction	Sintaxis	Action
ADC	Add with carry	*ADC{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Rn + Op2 + Carry
ADD	Add	*ADD{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Rn + Op2
AND	AND	*AND{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Rn AND Op2
B	Branch	*B{L}{<cond>} <target_address>*	R15 = address
BIC	Bit Clear	*BIC{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Rn AND NOT Op2
BL	Branch with Link	*B{L}{<cond>} <target_address>*	R14 = R15, R15 := address
BX	Branch and Exchange	*BX{<cond>} <Rm>*	R15 = Rn, T bit = Rn[0]
CMN	Compare Negative	*CMN{<cond>} <Rn>, <shifter_operand>*	CPSR flags = Rn + Op2
CMP	Compare	*CMP{<cond>} <Rn>, <shifter_operand>*	CPSR flags = Rn - Op2
EOR	Exclusive OR	*EOR{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = (Rn AND NOT Op2) OR (op2 AND NOT Rn)
LDM	Load multiple registers	*LDM{<cond>}<addressing_mode> <Rn>{!}, <registers>*	Stack manipulation (Pop)
LDR	Load register from memory	*LDR{<cond>} <Rd>, <addressing_mode>*	Rd = (address)
MCR	Move CPU register to coprocessor register	*MOV{<cond>}{S} <Rd>, <shifter_operand>*	cRn = rRn {<op>cRm}
MLA	Multiply Accumulate	*MRS{<cond>} <Rd>, CPSR*	Rd = (Rm * Rs) + Rn
MOV	Move register or constant	*MRS{<cond>} <Rd>, SPSR*	Rd = Op2
MRC	Move from coprocessor register to CPU register	*MSR{<cond>} CPSR_<fields>, #<immediate>*	Rn = cRn {<op>cRm}
MRS	Move PSR status/flags to register	*MSR{<cond>} CPSR_<fields>, <Rm>*	Rn = PSR
MSR	Move register to PSR status/flags	*MSR{<cond>} SPSR_<fields>, #<immediate>*	PSR = Rm
MUL	Multiply	*MUL{<cond>}{S} <Rd>, <Rm>, <Rs>*	Rd = Rm * Rs
MVN	Move negative register	*MVN{<cond>}{S} <Rd>, <shifter_operand>*	Rd = 0xFFFFFFFF EOR Op2
ORR	OR	*ORR{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Rn OR Op2
RSB	Reverse Subtract	*RSB{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Op2 - Rn
RSC	Reverse Subtract with Carry	*RSC{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Op2- Rn - 1 + Carry
SBC	Subtract with Carry	*SBC{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Rn - Op2 - 1 + Carry
STM	Store Multiple	*STM{<cond>}<addressing_mode> <Rn>{!}, <registers>*	Stack manipulation (Push)
STR	Store register to memory	*STR{<cond>} <Rd>, <addressing_mode>*	<address> = Rd
SUB	Subtract	*SUB{<cond>}{S} <Rd>, <Rn>, <shifter_operand>*	Rd = Rn - Op2
SWI	Software Interrupt	*SWI{<cond>} <immed_24>*	OS call
SWP	Swap register with memory	*SWP{<cond>} <Rd>, <Rm>, [<Rn>]*	Rd = [Rn], [Rn] := Rm
TEQ	Test bitwise equality	*TEQ{<cond>} <Rn>, <shifter_operand>*	CPSR flags = Rn EOR Op2
TST	Test bits	*TST{<cond>} <Rn>, <shifter_operand>*	CPSR flags = Rn AND Op2

Tabla 1-1. Resumen de instrucciones

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Procesamiento de datos turno inmediato	cond [1]				0	0	0	opcode				S	Rn				Rd				Cantidad de cambio				Cambio		0	Rm			
Instrucciones varias	cond [1]				0	0	0	1	0	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	X	X	X	X
Desplazamiento del registro de procesamiento de datos [2]	cond [1]				0	0	0	opcode				S	Rn				Rd				Rs				0	Cambio		1	Rm			
Instrucciones varias	cond [1]				0	0	0	1	0	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	0	X	X	1	X	X	X	X
Multiplicación, carga/guardado extra	cond [1]				0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	1	X	X	X	X
Procesamiento inmediato de datos [2]	cond [1]				0	0	1	opcode				S	Rn				Rd				Giro				Inmediato						
Instrucción indefinida [3]	cond [1]				0	0	1	0	0	X	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Movimiento inmediato al registro de estado	cond [1]				0	0	1	1	0	R	1	0	Máscara				SBO				Giro				Inmediato						
Cargar/guardar offset inmediato	cond [1]				0	1	0	P	U	B	W	L	Rn				Rd				Inmediato										
Cargar/guardar registro de offset	cond [1]				0	1	1	P	U	B	W	L	Rn				Rd				Cantidad de cambio				Giro		0	Rm			
Instrucción indefinida	cond [1]				0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X
Instrucción indefinida [4,7]	1	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Carga/guardado múltiple	cond [1]				1	0	0	P	U	S	W	L	Rn				Lista de registro														
Instrucción indefinida [4]	1	1	1	1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Rama y rama con enlace	cond [1]				1	0	1	L	Offset de 24 bits																							
Rama y rama con enlace y cambio a Thumb [4]	1	1	1	1	1	0	1	H	Offset de 24 bits																							
Carga/guardado de coprocesador y transferencia de doble registro	cond [5]				1	1	0	P	U	N	W	L	Rn				CRd				cp_num				Offset de 8 bits						
Procesamiento de datos del coprocesador	cond [5]				1	1	1	0	opcode1				CRn				CRd				cp_num				opcode2			0	CRm		
Transferencia de registro del coprocesador	cond [5]				1	1	1	0	opcode1			L	CRn				Rd				cp_num				opcode3			1	CRm		
Interrupción de software	cond [5]				1	1	1	1	Número SWI																							
Instrucción indefinida [4]	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabla 1-2. Código de las instrucciones

Por facilidad, las instrucciones se pueden dividir en los siguientes grupos:

C.1. PROCESAMIENTO DE DATOS:

a. Mover datos: *MOV, MVN, MOVT.*
b. Operaciones aritméticas: *ADC, ADD, RSB, RSC, SBC, SUB, MLA, MUL.*
c. Operaciones lógicas: *AND, ORR, EOR, BIC.*
d. Comparaciones: *CMN, CMP, TEQ, TST.*
e. Control de flujo
    i. Incondicionales: *B, BL, BX, BLX.*
    ii. Condicionales: *ver tabla 1-4*.

En general, para este grupo de instrucciones, la sintaxis es:

MNEMONICO {<cond>} {S} <Rd>, <Rn>, <shifter_operand>

Donde:

MNEMONICO	@	*Instrucción en ensamblador.*
{*<cond>*}	@	*Es opcional, aplica en caso de ejecutar con la condición especificada.*
{*S*}	@	*Opcional en caso de requerir actualizar las banderas.*
*<**Rd**>*	@	*Registro destino de la instrucción.*
*<**Rn**>*	@	*Primer operador.*
<shifter_operand>	@	*Segundo operador; puede ser: un dato inmediato, el contenido de un registro o el corrimiento lógico/aritmético de un dato/registro.*

Operador de desplazamiento (<*shifter_operand*>)	Sintaxis
Inmediato	#inmediato
Registro	Rm
Corrimiento lógico a la izquierda a través de un valor inmediato	Rm, LSL #shift_inm
Corrimiento lógico a la izquierda a través de un registro	Rm, LSL Rs
Corrimiento lógico a la derecha a través de un valor inmediato	Rm, LSR #shift_inm
Corrimiento lógico a la derecha a través de un registro	Rm, LSR Rs
Corrimiento aritmético a la derecha a través de un valor inmediato	Rm, ASR #shift_inm
Corrimiento aritmético a la derecha a través de un valor inmediato	Rm, ASR Rs
Rotación a la derecha a través de un valor inmediato	Rm, ROR #shift_inm
Rotación a la derecha a través de un valor inmediato	Rm, ROR Rs

Tabla 1-3. Parámetros de desplazamiento

Ejemplos:

MOV R1, #0XAA	*@ R1 = 0X000000AA*
MOV R0, #0X1234	*@ R0 = 0X00001234*
MOVT R0 ,#0XABCD	*@ R0 = 0XABCD*****
MOV R2, R3	*@ R2 = R3*
MOV R0, R2, LSL #2	*@R0= R2 << 2*
MOV R0, R1, ASR #3	*@R0= R1 >> 3*
MOV R0, R2, ROR #2	*@R0 = R2; 2 rotaciones a la derecha de R2, resultado en R2*
ADD R1, R2, #8	*@R1 = R2 + 8*
ADDS R1, R2, #0X10	*@R1 = R2 + 0X10; actualiza el registro de banderas CPSR*
ADC R1, R2, #0XFF	*@R1 = R2 + 0XFF + C*
ADD R0, R2, R1, LSL #2	*@R0 = R2 + (R1*4)*
ADD R2, R0, R0, LSL #3	*@R2 = 9 * R0*

C.2. TRANSFERENCIA DE DATOS ENTRE REGISTROS Y MEMORIA:

i. Carga: *LDR, LDRB, LDRH.*
ii. Almacenamiento: *STR, STRB, STRH.*

C.2.I. CARGA

LDR R*DESTINO*, [*Memoria_32*]	@	*Carga el contenido de la memoria al R*DESTINO*; es común que la dirección a acceder esté especificada previamente en un registro.*

Ejemplo:

LDR R0, =DATO1	@	*Carga en R0 la dirección de DATO1.*
LDR R1, [R0]	@	*Carga en R1 el contenido de la dirección de memoria indicada por R0*
LDR R2,0x12345678	@	*Carga el contenido de la dirección 0X12345678 a R2.*

C.2.II. ALMACENAMIENTO

STR R*FUENTE*, *Memoria_32*	@	*Almacena el contenido del R*FUENTE *en memoria; es común que la dirección a acceder esté especificada previamente en un registro.*

LDR R0, =DATO1	@	*Carga en R0 la dirección de DATO1.*
STR R1, [R0]	@	*Almacena el contenido de R1 en la dirección de memoria indicada por R0.*
STR R2,0x12345678	@	*Almacena el contenido de R2 en la dirección de memoria 0X12345678.*

Para los dos últimos ejemplos tanto para *LDR* como *STR*, se requiere tener identificado la dirección a acceder.

C.3. CONTROL DE FLUJO

a. Incondicionales: *B, BL, BX, BLX.*
b. Condicionales: *BEQ, BNE, BCS, BCC, BMI, BPL, BHI, BLE, BGE, BLT, BLE, etc*.

C.3.B. INSTRUCCIONES CONDICIONALES.

Sintaxis

B {L} {<cond>} <target_addres>

Donde:

B	@	*Instrucción de brinco.*
{L}	@	*BL para llamado a subrutina.*
{<***cond***>}	@	*Es opcional, aplica en caso de ejecutar con la condición especificada.*
<***target_address***>	@	*Dirección de salto; se especifica con una etiqueta.*

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

Tabla 1-4. Instrucciones condicionales

Ejemplos:

```
INICIO:   MOV R0,#1
          …
          B INICIO          @Incondicional

          CMP R0,#7
          BEQ IGUAL         @Condicional
          BNE DIF
IGUAL:    B IGUAL           @Incondicional
DIF:      B DIF             @Incondicional
```

Llamados a subrutinas

```
          BL SUB_RUT        @Llamado a subrutina; LR← PC, PC ← SUB_RUT
          …..
SUB_RUT:  MOV R0,2          @Inicio de subrutina
          ….
          MOV PC,LR         @Retorno de subrutina; PC ← LR
```

d. Formato del programa en ensamblador.

El programa deberá integrar:

- ***Instrucciones o mnemónicos****; especificadas por el conjunto de instrucciones ARM; pueden ser escritas en mayúsculas o minúsculas de manera libre.*
- ***Directivas****; configuraciones o definiciones dentro del programa, soportadas por el ensamblador.*
- ***Etiquetas****; referencias a instrucciones o direcciones de memoria, terminar con* ***:*** *(dos puntos), ejemplo: etiqueta1****:*** *; deben conservar el formato original (mayúsculas o minúsculas).*
- ***Comentarios****; se inicia con* ***@****, requerida para documentación del programa.*

Debe delimitar dos bloques del programa:

- ***Área de código o programa**; indicada por la directiva **.text**.*
  - *Al inicio del área de código **(.text)**, usar la directiva global para declarar el inicio del mismo **global _start; o global main en Code::Blocks**.*
  - *Agregar la etiqueta de inicio del código; **_start**, o main en Code::Blocks.*
- ***Área de datos**; especificada por la directiva **.data**.*

```
                .text           @Inicio de código de programa
                .global _start  @Define _start como global para ser visible por el linker
_start:                         @Punto de entrada para el linker
                Instrucción 1
etiqueta1:      Instrucción 2
                ….

etiqueta n:     Instrucción n

                .data           @inicio de área reservada para declarar datos
DATO1:          .word 0         @ Reserva espacio para DATO1 y asigna el valor 0
```

Como puede consultar en la documentación adicional, el formato del programa empleando Code::Blocks, cambia ***_start*** por ***main***, dado que se compila el programa usando un proyecto en C.

```
                .text           @Inicio de código de programa
                .global main    @Define main como global para ser visible por el linker
main:                           @Punto de entrada para el linker
                Instrucción 1
etiqueta1:      Instrucción 2
                ….

etiqueta n:     Instrucción n

                .data           @inicio de área reservada para declarar datos
DATO1:          .word  0        @ Reserva espacio para DATO1 y asigna el valor 0
```

2. Requerimientos

2.1. Software.

- *GNU GCC Compiler*

2.2. Editor (opciones).

- *Code::Blocks*
- *Vi*
- *Nano*
- *Geanny*
- *CPUlator*

2.3. Hardware (opciones).

- *Raspberry Pi 3+*
- *Raspberry Pi 4*
- *Raspberry Pi 400*
- *Raspberry Pi 5*
- *Simulado en línea CPUlator*

3. Desarrollo

Realizar las actividades solicitadas.

Actividad 1. Seguir el procedimiento indicado en el apartado ***cuarto*** de manual de tutoriales, escribir, comentar y ensamblar y ejecutar el siguiente programa; explicar que hace:

```
        .global _start
_start:  mov r1, #0x19
        mov r2, #53
        add r3, r2,r1
        mov r0,r3
        mov r7,#1
        svc 0
```

Figura 1-3. Código de ejemplo; actividad 1

Actividad 2. Seguir el procedimiento indicado en el apartado ***cuarto*** de manual de tutoriales, escribir, comentar, ensamblar y ejecutar el siguiente programa; explicar que hace.

```
        .text
        .global _start
_start: mov r0,#5
        mov r1,#0x01
        subs r3,r0,r1
        beq igual
        bne diferente
igual:  mov r0,#1
        ldr r1,=texto1
        mov r2,#30
        mov r7,#4
        svc 0
        b fin
diferente: mov r0,#1
        ldr  r1,=texto2
        mov r2,#33
        mov r7,#4
        svc 0
fin:    mov r0,r3
       mov r7,#1
       svc 0
        .data
texto1: .ascii "Datos iguales ... resultado = "
texto2: .ascii "Datos diferentes ... resultado = "
```

Figura 1-4. Código de ejemplo; actividad 2

Actividad 3. Empleando el IDE Code::Block, seleccionar 10 instrucciones, formalizar un programa; comprobar el funcionamiento (agregar las directivas correspondientes).

a. Reportar el resultado esperado y el obtenido.

Recordar cambiar ***_start*** por ***main*** como inicio del llamado al programa.

```
    .global main
main:
```

Figura 1-5. Modificación Code::Blocks

Actividad 4. Tomando como base el programa de la actividad 1, para que obtenga el promedio de dos números de 8 bits; utilizar Code::Blocks para todo el proceso.

$$Promedio = \frac{DATO1 + DATO2}{2}$$

Actividad 5. Emplear el IDE Code::Block, escribir, comentar, compilar y ejecutar el siguiente programa.

```
        .text
        .global main
main:   mov R0,#0
        mov r1,#9
        mov r2,#0
loop1:  add r0,r0,#1
        cmp r1,r0
        bne loop1
loop2:  add r0,r0,#-1
        cmp r2,r0
        beq loop1
        b loop2
```

Figura 1-6. Código de ejemplo; actividad 5

Actividad 6. Realizar un programa que inicie activando el bit menos significativo de un registro y recorra de posición hacia el bit más significativo (solo un bit estará activado); usar el IDE Code:Blocks.

31	30		2	1	0
0	0	…	0	0	1
0	0	…	0	1	0
⋮	⋮	⋮	⋮	⋮	⋮
0	1	…	0	0	0
1	0	…	0	0	0

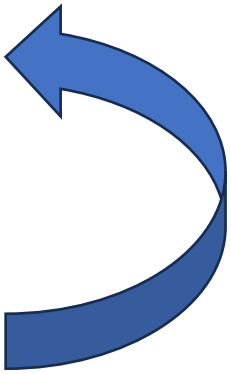


Figura 1-7. Secuencia actividad 6

Actividad 7. Escribir un programa que realice la suma de dos números de 32 bits y almacene el resultado en memoria empleando las direcciones que considere el resultado del acarreo en caso de existir.

	DATO1_32_BITS
+	*DATO2_32_BITS*
ESTADO ACARREO	***RESULTADO_32BITS***

Actividad 8. Escribir un programa que realice la suma de dos números de 64 bits y almacene el resultado en memoria empleando las direcciones que considere el resultado del acarreo en caso de existir.

	DATO1_64_BITS_H	*DATO1_64_BITS_L*
+	*DATO2_64_BITS_H*	*DATO2_64_BITS_L*
ESTADO DEL ACARREO	***RESULTADO_64_BITS_H***	***RESULTADO_64_BITS_L***

Actividad 9. Realizar un programa que obtenga el factorial de un número de 8 bits.

$$Resultado = n!$$

Actividad 10. Implementar con instrucciones en lenguaje ensamblador la sentencia:

---

int j = 0;

for (i = 0; i ≤ 50 ; i++) { j = j+2: }

Laboratorio de Microcomputadoras
Practica No. 2

Programación en ensamblador, direccionamiento indirecto

Objetivo. Programar las variantes del modo de direccionamiento indirecto existentes para los procesadores ARM.

1. Introducción

El acceso a memoria se realiza utilizando las instrucciones de carga (*ldr*) y almacenamiento (*str*); la forma general de estas instrucciones es:

LDR Reg_destino, [Dir_Men_32]

STR Reg_fuente, [Dir_Mem_32]

Donde:

Reg_destino:	Será el registro donde será almacenado el dato requeridor; este podrá ser: • La dirección base de la ubicación del dato. • El contenido de memoria**.**
Reg_fuente:	El registro cuyo contenido será transferido a memoria.
[Dir_Mem_32]:	Dirección de memoria en el que se lee o escribe el dato.

Ejemplo:

LDR R0, =DATO	@	Carga en R0 la dirección asignada a la variable dato.
LDR R0,=abcd1234	@	Carga en R0 el valor 0xabcd1234; podrá ser usado como una constante que especifica la dirección de memoria.
LDR R0, [R0]	@	Carga el contenido de la dirección de memoria indicada por R0 en R0.
STR R1, [R0]	@	Transfiere el contenido el registro R1 a la dirección de memoria indicada por R0.

Al usar ***LDR*** y ***STR***, se podrá operar con la memoria que se indica con el valor contenido en el registro base, así como un desplazamiento.

El desplazamiento puede ser incrementado o decrementado, considerando tres posibles parámetros, actualizando el registro base antes o después de obtener la dirección de memoria a operar.

Formato general:

LDR Registro_destino, [Registro_base, Opciones_de_desplazamiento]

STR Registro_fuente, [Registro_base, Opciones_de_desplazamiento]

Donde:

Registro_destino:	Registro en el que será almacenado el dato de memoria.
Registro_fuente:	Registro cuyo contenido será transferido a memoria.
Registro_base:	Contiene el valor considerado como base (apuntador); debe ser cargado previamente.
Opciones_de_desplazamiento:	Es el desplazamiento que se aplicará a la dirección base; puede ser:

• *Sin desplazamiento:*	LDR R0, [R1]
• *Desplazamiento con valor inmediato:*	LDR R0, [R1, #4]
• *Desplazamiento a través de un registro:*	LDR R0, [R1, R2]
• *Desplazamiento de un registro:*	LDR R0, [R1, R2 LSL #2]

a. Actualización del registro base.

Considerando la forma de operación de este tipo de direccionamiento, es posible actualizar el valor del registro base previo o posterior a la ejecución de la instrucción.

a.1. Auto indexado (pre indexado).

LDR R0, [R1, #4]!	@	Carga al registro R0 el contenido de la dirección de memoria conformada por ***[R1 + 4]***; el registro será actualizado con: ***R1 = R1 + 4***.

a.2. Post indexado.

LDR R0, R1, #4	@	Carga al registro R0 el contenido de la dirección de memoria conformada por ***[R1]***; posteriormente el registro será actualizado con: ***R1 = R1 + 4***.

2. Requerimientos

2.1 Software.

- *GNU GCC Compiler*

2.2 Editor (opciones).

- *Code::Blocks*
- *Vi*
- *Nano*
- *Geanny*
- *CPUlator*

2.3 Hardware (opciones).

- *Raspberry Pi 3+*
- *Raspberry Pi 4*
- *Raspberry Pi 400*
- *Raspberry Pi 5*
- *Simular en línea CPUlator*

3 Desarrollo.

Realizar las actividades solicitadas; ocupará el IDE Code::Blocks.

Actividad 1. Escribir, comentar, compilar y comprobar el funcionamiento del siguiente programa.

```
    .data
i:  .skip 64

    .text
    .global main
main:
    ldr r1,=i
    mov r2,#0
loop: cmp r2,#16
    beq fin
    add r3,r1,r2,LSL #2
    str r2,[r3]
    add r2,r2,#1
    b loop

fin: b fin
```

Figura 2-1. Código de ejemplo: actividad 1

Actividad 2. Modificar el programa de la actividad 1, para usar el direccionamiento indexado de su preferencia con el doble de datos.

Actividad 3. Realizar un programa almacene en memoria un arreglo de datos de 32 bits con 16 elementos; una vez transferidos, realizar la copia en sentido inverso en otro arreglo.

A = [dato1, dato2, dato3, dato4, ……, dato15, dato16] @Original

B = [dato16, dato15, dato14, dato13, ……, dato2, dato1] @Copia

---

Actividad 4. Realizar un programa que forme un arreglo de 20 elementos, con el siguiente criterio:

A = [i, 2i, 4i, 8i, 16i, ….ni]

Donde i es un número considerado como valor inicial.

a. Enviar a memoria cada uno de ellos;
b. Sumar y almacenar en memoria el resultado.

Actividad 5. Realizar un programa que multiplique dos matrices de 2x2; los datos podrán ser de 8 bits.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} I & J \\ K & L \end{bmatrix}$$

Actividad 6. Realizar un programa que encuentre el número con valor mayor en un arreglo de 20 elementos que serán almacenados en memoria; para lo cual:

a. Indicar cual fue el valor mayor.
b. Ubicar la dirección donde se encontró este número.
c. Usar las direcciones que requiera para cumplir lo solicitado.

Actividad 7. Realizar un programa que ordene de manera ascendente un arreglo de 32 elementos de 32 bits; deberá:

a. Mantener el arreglo original.
b. Generar otro arreglo con el ordenamiento del original.

Arreglo original.

A[0]	*A[1]*	*A[2]*		*A[30]*	*A[31]*

Arreglo ordenado.

Menor A[x]		*Mayor A[y]*

Laboratorio de Microcomputadoras
Practica No. 3
Introducción a la plataforma Raspberry Pi Pico
GPIO de Salida; programación en Micropython

Objetivo. Conocer los recursos de la plataforma con el microcontrolador RP2040 contenida en la plataforma Raspberry Pi Pico; desarrollar algoritmos mediante la programación en Micropyton utilizando el IDE Thonny para generar salidas a través de las terminales GPIO.

1. Introducción

La fundación Raspberry Pi, ha lanzado plataformas en la versión microcontrolador, en la actualidad están disponibles: Raspberry Pi Pico y Raspberry Pi Pico 2, en ambas casos dispone de versiones con conexión WiFi denotadas como W, así como con pines (headers) soldados o por soldar; entre sus características se encuentran:

Versión	Microcontrolador	Características
Raspberry Pi Pico	RP2040	Dual Core ARM Cortex M0+@133MHz 254 KB de SRAM y 2MB de FLASH USB 1.1, 26 GPIO con multifunción, 3 ADC de 12bits, 2 SPI, 2 I2C, 2 UART, 24 PWM, 2 TIMERS, 1 sensor de temperatura integrado.
Raspberry Pi Pico W	RP2040	Dual Core ARM Cortex M0+@133MHz Wireless (802,11n) @2.4 GH z WPA 3, Bluetooth 5.2 (clásico y BLE) 254 KB de SRAM y 2MB de FLASH USB 1.1, 26 GPIO con multifunción, 3 ADC de 12bits, 2 SPI, 2 I2C, 2 UART, 24 PWM, 2 TIMERS, 1 sensor de temperatura integrado.
Raspberry Pi Pico 2	RP2350	Dual Cortex-M33 @150MHz 520 KB de SRAM y 4MB de FLASH USB 1.1, 26 GPIO con multifunción, 3 ADC de 12bits, 2 SPI, 2 I2C, 2 UART, 24 PWM, 2 TIMERS, 1 sensor de temperatura integrado.
Raspberry Pi Pico 2W	RP2350	Dual Cortex-M33 @150MHz Wireless (802,11n) @2.4 GH z WPA 3, Bluetooth 5.2 (clásico y BLE) 520 KB de SRAM y 4MB de FLASH USB 1.1, 26 GPIO con multifunción, 3 ADC de 12bits, 2 SPI, 2 I2C, 2 UART, 24 PWM, 2 TIMERS, 1 sensor de temperatura integrado.

Tabla 3-1. Raspberry Pi Pico/Pico W

Para realizar la practica y las subsecuentes, se utilizará el entorno THONNY y la programación en Micropython. Previamente le ha sido instalado el intérprete de Micropython a la plataforma Raspberry Pi Pico; para mayor información, consultar la documentación oficial o apéndice respectivo.

Conectar usando el cable USB entre la tarjeta y la PC; al ejecutar THONNY, mostrará la siguiente pantalla:

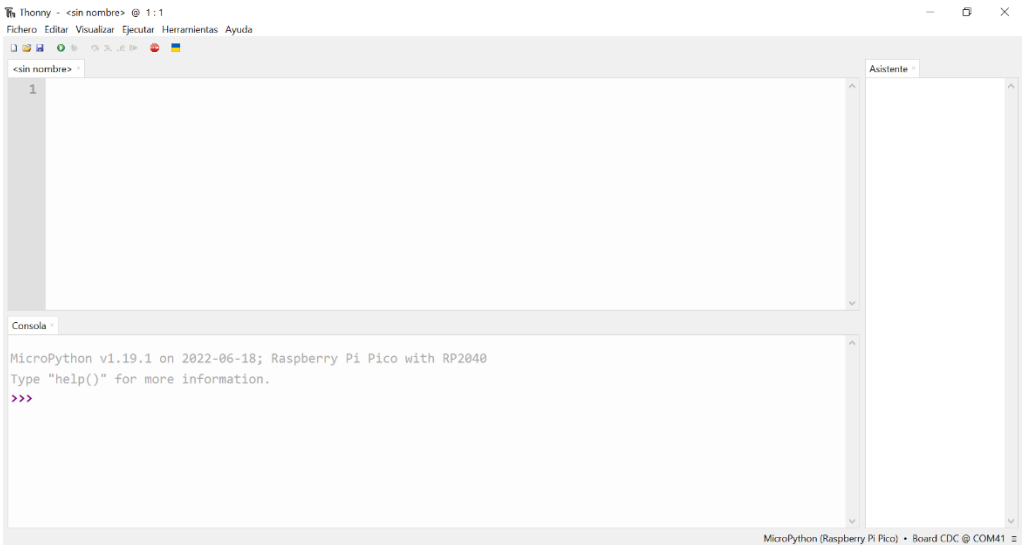


Figura 3-1. IDE Thonny; conexión exitosa

En caso de haber seguido las instrucciones dadas, aparecerá la conexión exitosa indicando el reconocimiento de la plataforma, el puerto COM habilitado (será diferente para cada caso); la consola queda lista para escribir instrucciones.

En caso de haber omitido conectar Raspberry Pi Pico, una vez conectado bastará con presional el icono rojo 

*(**Detener/reiniciar**)* y se restablecerá la conexión, desplegando la pantalla de la figura 3-1.

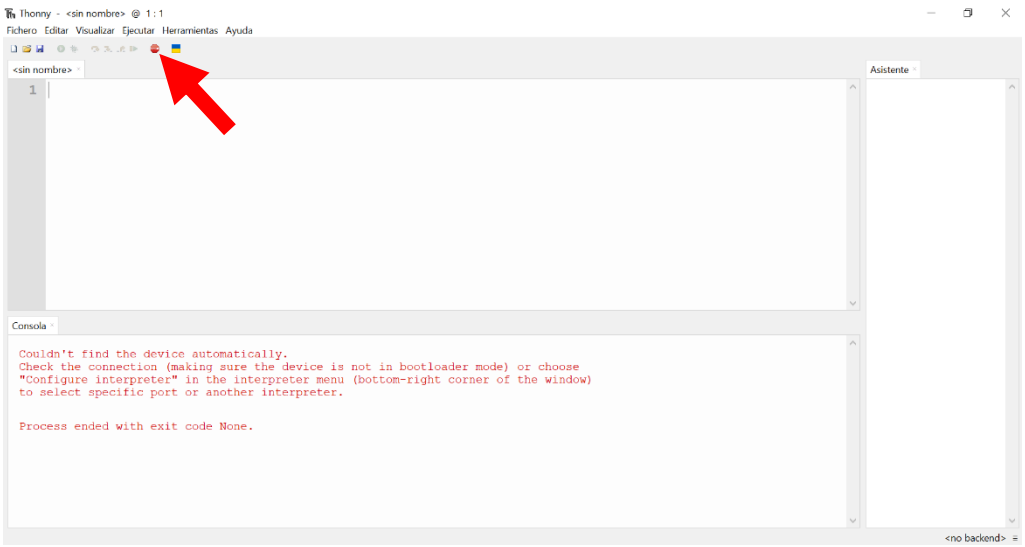


Figura 3-2. IDE Thonny; conexión fallida

2. Requerimientos

2.1 Software.

- *Micropython*

2.2 Editor.

- *Thonny*

2.3. Hardware.

- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 1 para Raspberry Pi Pico*
- *8 Leds*
- *8 Resistencias de 220 Ω*

La asignación corresponde a las terminales GPIO0 a GPIO7 de acuerdo a la tabla 3-2; el circuito implementado se muestra en el esquemático de la figura 3-3.

GPIO0	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7
LED VERDE	LED AMARILLO	LED ROJO	LED AZUL	LED AZUL	LED ROJO	LED AMARILLO	LED VERDE

Tabla 3-2. Asignación de GPIO

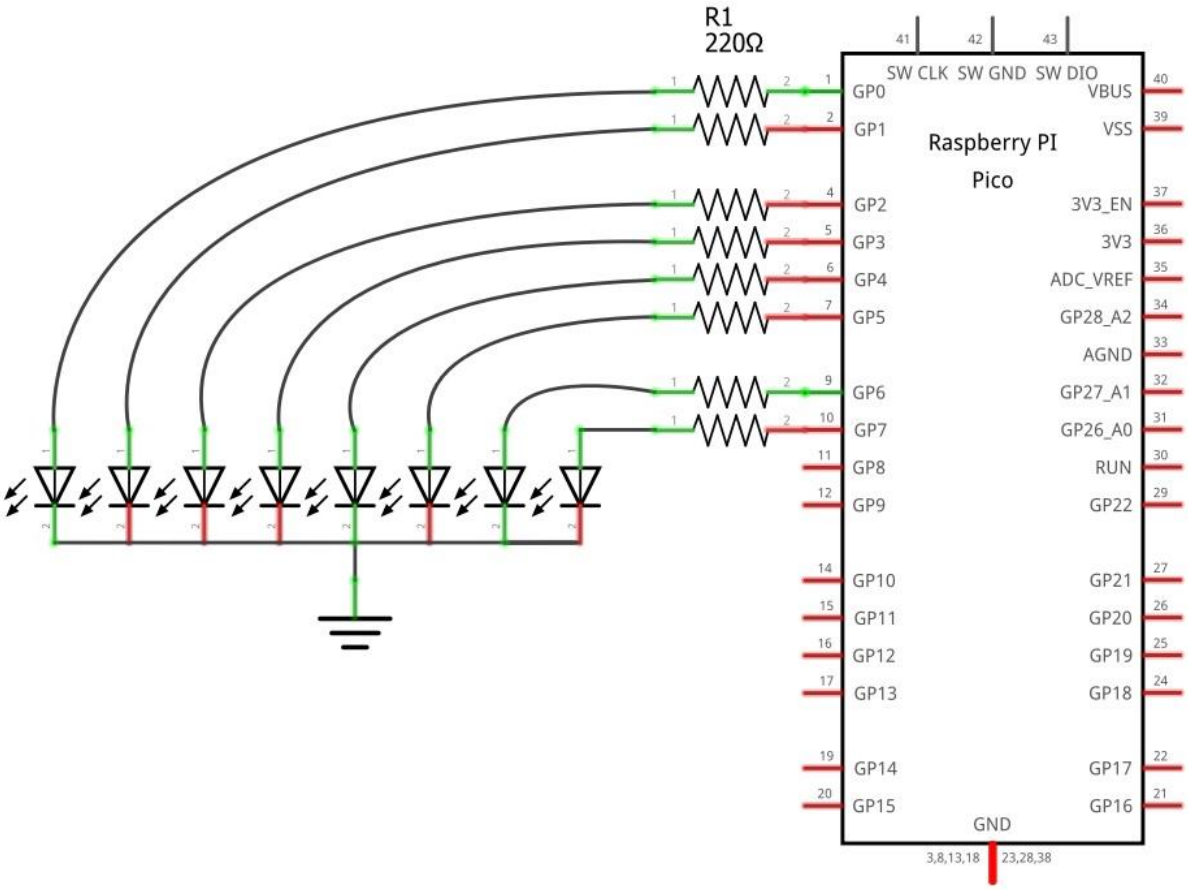


Figura 3-3. Esquemático del módulo de salida GPIO0 a GPIO7

3. Desarrollo

Realizar las actividades solicitadas, mostrar el funcionamiento de cada una de ellas.

Actividad 1.

a. Usando el IDE Thonny; escribir el siguiente código y comentar cada instrucción:

```
import machine
import utime

LED = machine.Pin(0,machine.Pin.OUT)
while True:
    LED.value(1)
    utime.sleep_ms(500)
    LED.value(0)
    utime.sleep_ms(500)
```

Figura 3-4. Código de prueba

Del circuito de la figura 3-3, se muestra solo la salida correspondiente a GPIO0, quedando de la siguiente manera:

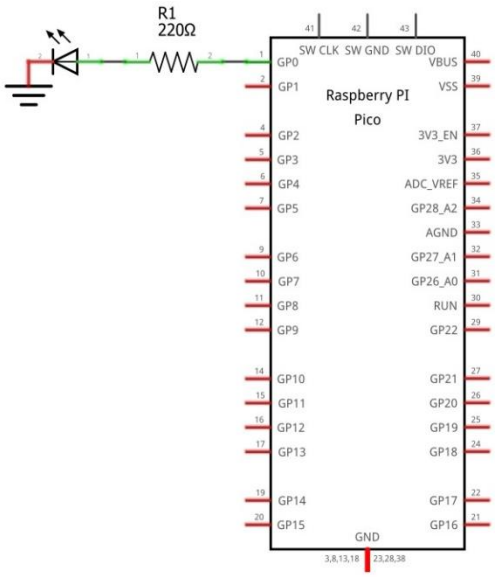


Figura 3-5. Circuito actividad 1

El programa queda de la siguiente manera; cuidar la indentación correspondiente a la programación en Python.

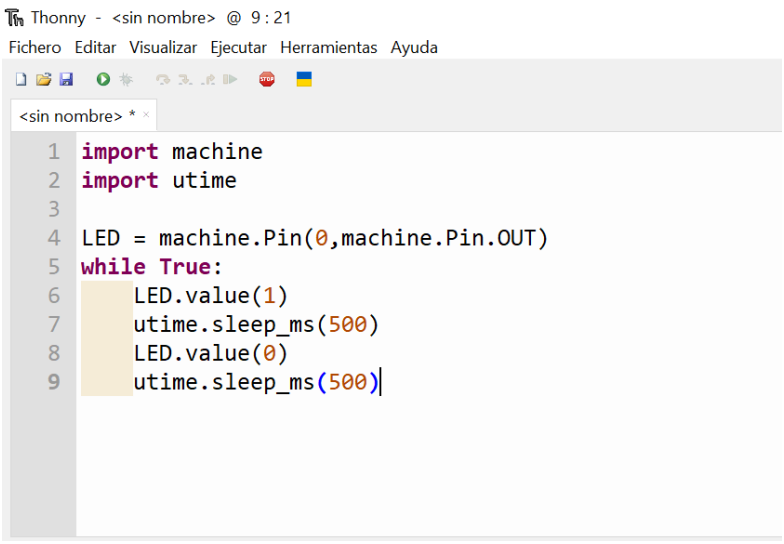
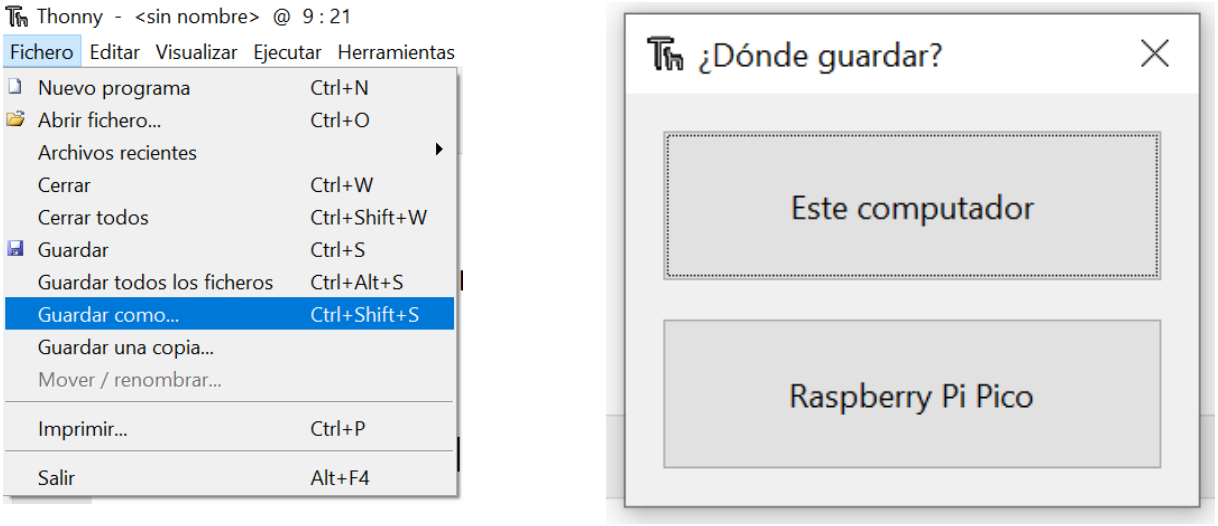


Figura 3-6. Código Micropython en Thonny

b. Guardar el programa en la computadora.



a b

Figura 3-7. Procedimiento para guardar un programa

c. Ejecutar el programa

Ubicarse en el programa recién guardado y ejecutar presionando el icono  *Ejecutar el scrip actual.*

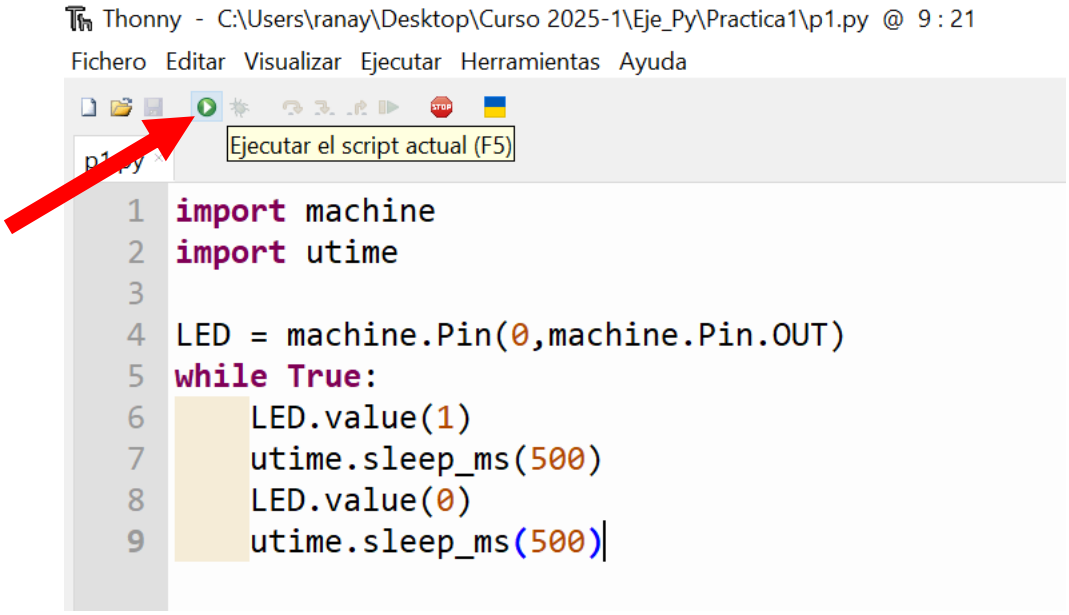


Figura 3-8. Ejecución

d. Detener ejecución

Para detener la ejecución del programa, presionar el icono  *stop*.

Figura 3-9. Detener ejecución

El IDE y la Raspberry Pi Pico, quedarán listos para recibir nuevos comandos.

Figura 3-10. Detener ejecución y en espera

Actividad 2. Realizar las modificaciones necesarias para que los tiempos sean más pequeños y más grandes que el código anterior.

Actividad 3. Realizar las modificaciones requeridas para que el efecto se muestre en otro GPIO (podrá usar GPIO1 al GPIO 7); usar retardos a conveniencia.

Actividad 4. Realizar un programa que genere las siguientes salidas:

GPIO2	GPIO3	
1	0	@85 ms entre estados
0	1	

Tabla 3-3. Control; actividad 4

Actividad 5. Realizar un programa que prenda y apague las GPIO0 al GPIO7; el circuito corresponde en su totalidad al mostrado en la figura 3.3, usar los retardos a elección.

GPI00	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7
0	0	0	0	0	0	0	0
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Tabla 3-4. Control; actividad 5

Actividad 6. Repetir la acción anterior 10 veces; mantener apagado 2 segundos y repetir acción.

GPI00	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7	
0	0	0	0	0	0	0	0	
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	*10 repeticiones (200 ms)*
0	0	0	0	0	0	0	0	
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	*2 segundos*

Tabla 3-5. Control; actividad 6

Actividad 7. Usando los GPIO0 al GPIO7; realizar un programa que genere la siguiente secuencia; usar retardos de 100ms.

GPI00	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7
1	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0
0	0	**1**	0	0	0	0	0
0	0	0	**1**	0	0	0	0
0	0	0	0	**1**	0	0	0
0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	**1**

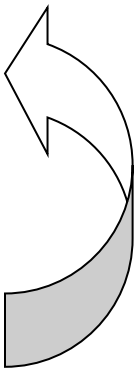


Tabla 3-6. Control; actividad 7

Actividad 8. Usando los GPIO0 al GPIO7; realizar un programa que genere la siguiente secuencia; usar retardos de 100ms.

GPI00	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7
1	0	0	0	0	0	0	0
1	**1**	0	0	0	0	0	0
1	**1**	**1**	0	0	0	0	0
1	**1**	**1**	**1**	0	0	0	0
1	**1**	**1**	**1**	**1**	0	0	0
1	**1**	**1**	**1**	**1**	**1**	0	0
1	**1**	**1**	**1**	**1**	**1**	**1**	0
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**

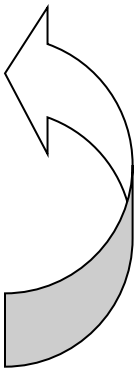


Tabla 3-7. Control; actividad 8

Actividad 9. Usando los GPIO0 al GPIO7; realizar un programa que genere un contador de 8 bits de forma ascendente; usar retardos de 50ms.

GPI00	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1

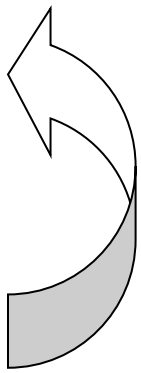


Tabla 3-8. Control; actividad 9

Actividad 10. Usando las salidas correspondientes a los colores de los leds; realizar un programa que controle el funcionamiento de dos semáforos, de acuerdo a la siguiente tabla.

Estado	Salida	Requerimiento
1	V1, R2	5 segundos
2	V1, R2	5 intermitentes de 200 ms, en V1
3	A1, R2	3 segundos
4	R1, V2	5 segundos
5	R1, V2	5 intermitentes de 200 ms, en V2
6	R1, A2	3 segundos

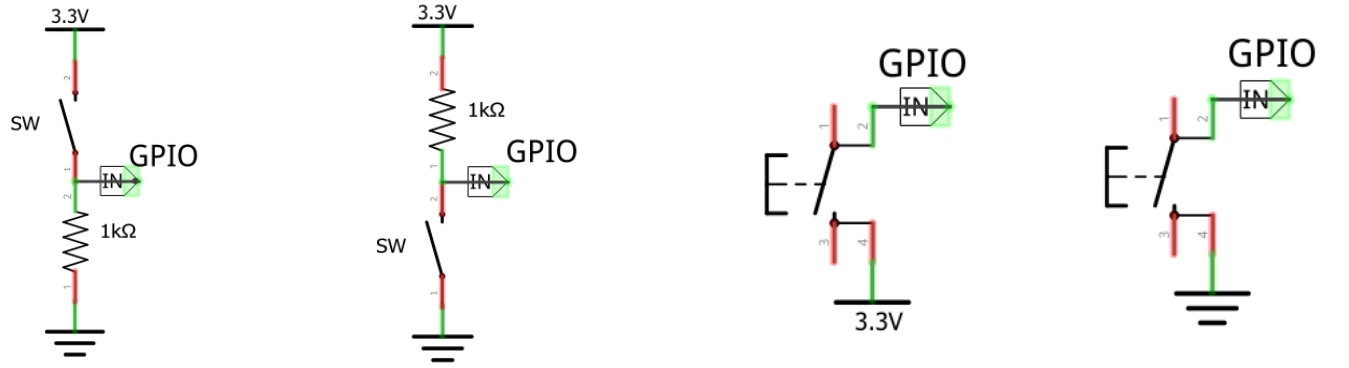
Tabla 3-9. Control; actividad 10

Laboratorio de Microcomputadoras
Practica No. 4
GPIO como entrada y salida

Objetivo. Realizar control de acciones mediante las terminales de Raspberry Pi Pico por medio de las funciones GPIO en la modalidad de entrada y salida.

1. Introducción

Las 26 terminales disponibles en Raspberry Pi Pico, asignados como GPIO0 a GPIO28, pueden se utilizadas como entrada o salida, bastará con la definición respectiva; así mismo cuando son utilizados push button, es posible habilitar resistencias internas para permitir reconocer flancos; estas resistencias pueden habilitarse como pull up o pull down.



(a) Interruptor con resistencia externa; pull down

(b) Interruptor con resistencia externa; pull up

(c) Interruptor con resistencia interna; pull down

(d) Interruptor con resistencia interna; pull up

Figura 4-1. Configuración de interruptores y push button

En micropython, la creación del objeto será:

sw1 = Pin(GPIO, Pin.IN, Pin.Configuración)

Instancia u objeto ***sw1***

Clase ***Pin***

GPIO *a usar* ***(0, 1, …, 28)***

Define como entrada ***Pin.IN***

Configuración ***PULL UP*** ***PULL DOWN***

Ejemplo:

sw8=Pin(8, Pin.IN) # Configura el **GPIO8** como entrada asignando a sw8.

sw11=Pin(11, Pin.IN, Pin.PULL_UP) # Configura **GPIO11** como entrada, habilita resistencia en la configuración **PULL_UP**.

2. Requerimientos

2.1. Software.

- *Micropython*

2.2. Editor.

- *Thonny*

2.3. Hardware.

- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 1 para Raspberry Pi Pico*
- *8 Leds*
- *8 Resistencias de 220 Ω*
- *3 Dip switch (3)*
- *Resistencias de 1 KΩ*
- *2 push button*
- *1 zumbador*

Distribución de terminales.

GPIO 8	GPIO9	GPIO10	GPI011	GPIO12
SW1_1	SW1_2	SW1_3	PUSH BUTTON PULL_UP	PUSH BUTTON PULL_DOWN

(a)

GPIO0	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7	GPIO22
LED VERDE	LED AMARILLO	LED ROJO	LED AZUL	LED AZUL	LED ROJO	LED AMARILLO	LED VERDE	BUZZER ACTIVO

(b)

Tabla 4-1. Asignación de entradas (a) y salidas (b)

El circuito se muestra en la figura 4-2.

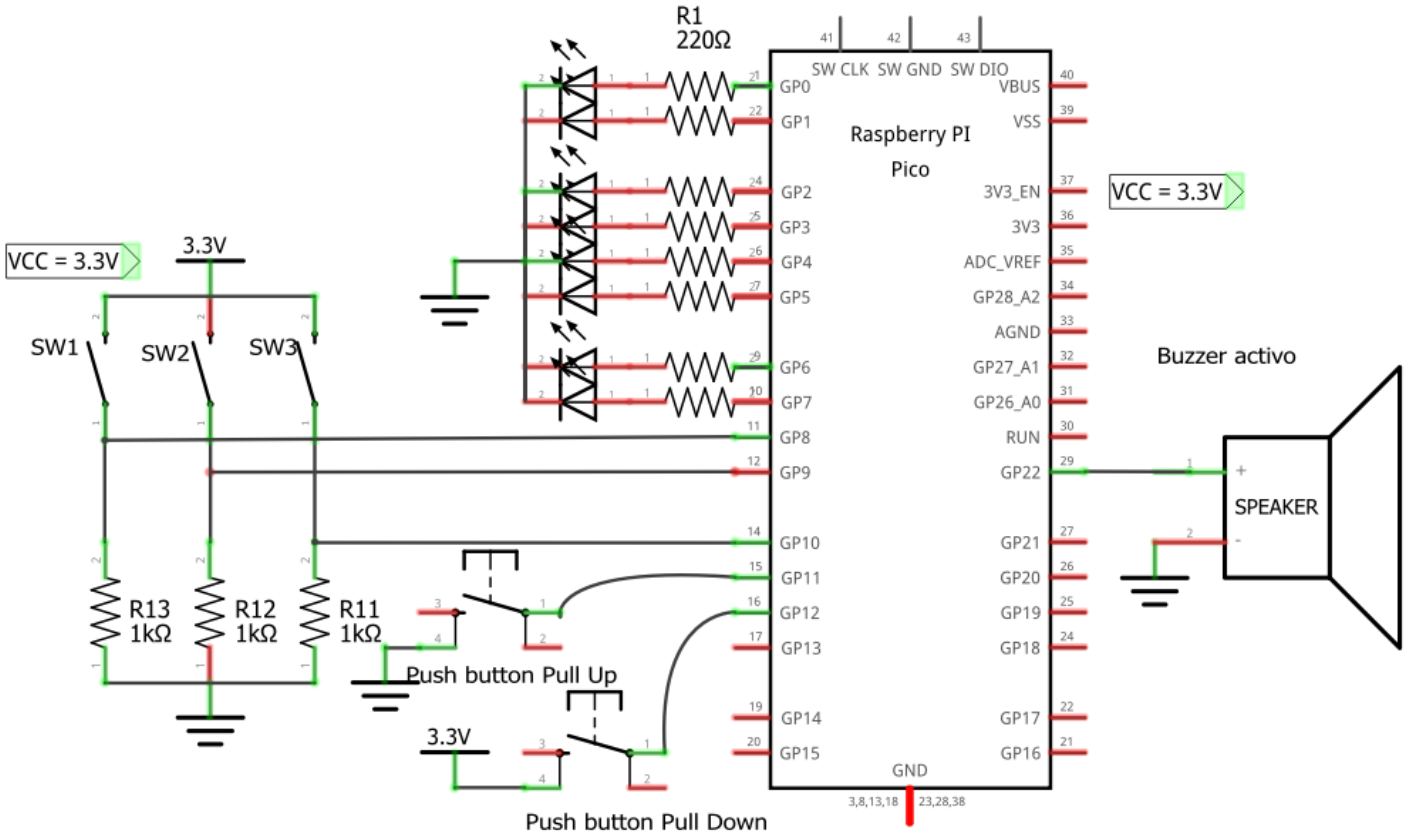


Figura 4-2. Esquemático entradas y salidas

3. Desarrollo.

Realizar las actividades solicitadas.

Actividad 1. Escribir el programa de la figura 4.4, comentar cada línea de código, reportar que hace y ejecutarlo; el circuito a emplear es el siguiente:

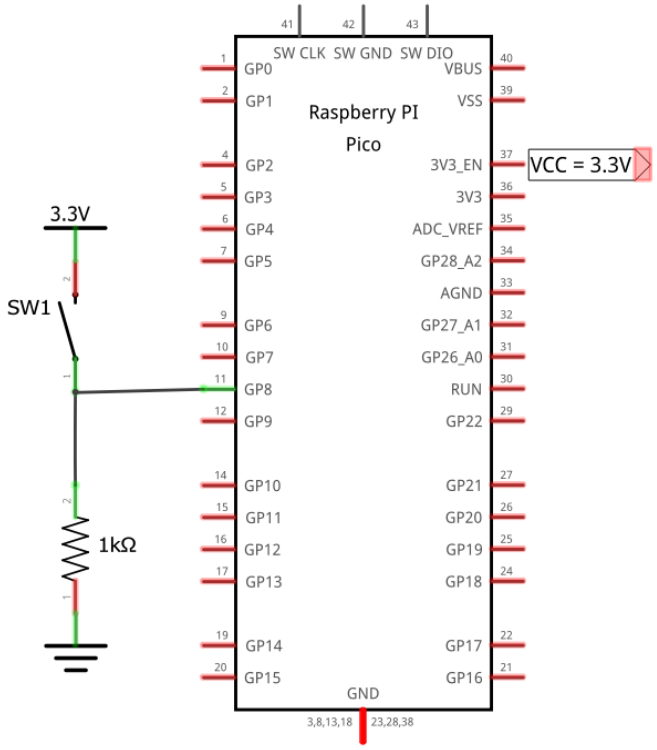
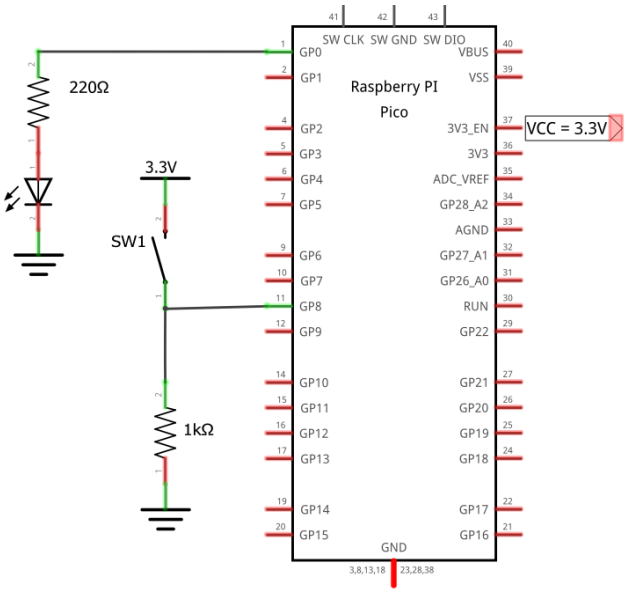


Figura 4-3. Entrada digital usando un interruptor

```
from machine import Pin
import time

sw1_1 = Pin(8 ,Pin.IN)

while True:
    if sw1_1.value() == 1:
        print("Interruptor cerrado, '1'")
        time.sleep(0.5)
    else:
        print("Interruptor abierto, '0'")
        time.sleep(0.5)
```

Figura 4-4. Código de prueba; actividad 1

Actividad 2. Realizar las modificaciones necesarias para que genere las acciones mostradas en la figura 4-5.



(a) Esquemático

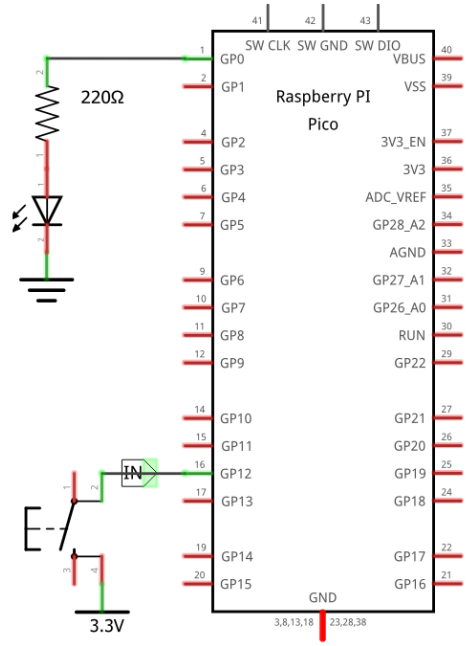
Entradas | Salidas

GPIO8 SW1_1	GPIO0 LED VERDE	Consola de Thonny
'0'	0	Interruptor abierto, '0' LED VERDE en GPIO0 apagado
'1'	1	Interruptor cerrado, '1' LED VERDE en GPIO0 encendido

(b) Tabla

Figura 4-5. Esquemático y tabla de la actividad 2

Actividad 3. Escribir el programa de la figura 4-6, comentar cada línea de código, describir que hace y ejecutarlo; en el reporte agregar la tabla de control.




```
from machine import Pin
import time

LED = Pin(0,Pin.OUT)
push_S2 = Pin(12,Pin.IN ,Pin.PULL_DOWN)

while True:

    if push_S2.value() == 0:
        LED.value(0)
        print("Push Button S2 en espera; ... lectura: '1'")
        time.sleep(0.5)
    else :
        LED.value(1)
        print("Push Button S2 presionado; ... lectura: '0'")
        time.sleep(0.5)
```

Figura 4-6. Circuito y código de prueba; actividad 3

Actividad 4. Usando el push button S1, tal como se ha mencionado previamente está conectado para operar en la configuración *PULL_UP*; modificar el estado de la GPIO4, GPIO5, GPIO6 y GPIO7 de acuerdo a la entrada indicada en la tabla 4.2.

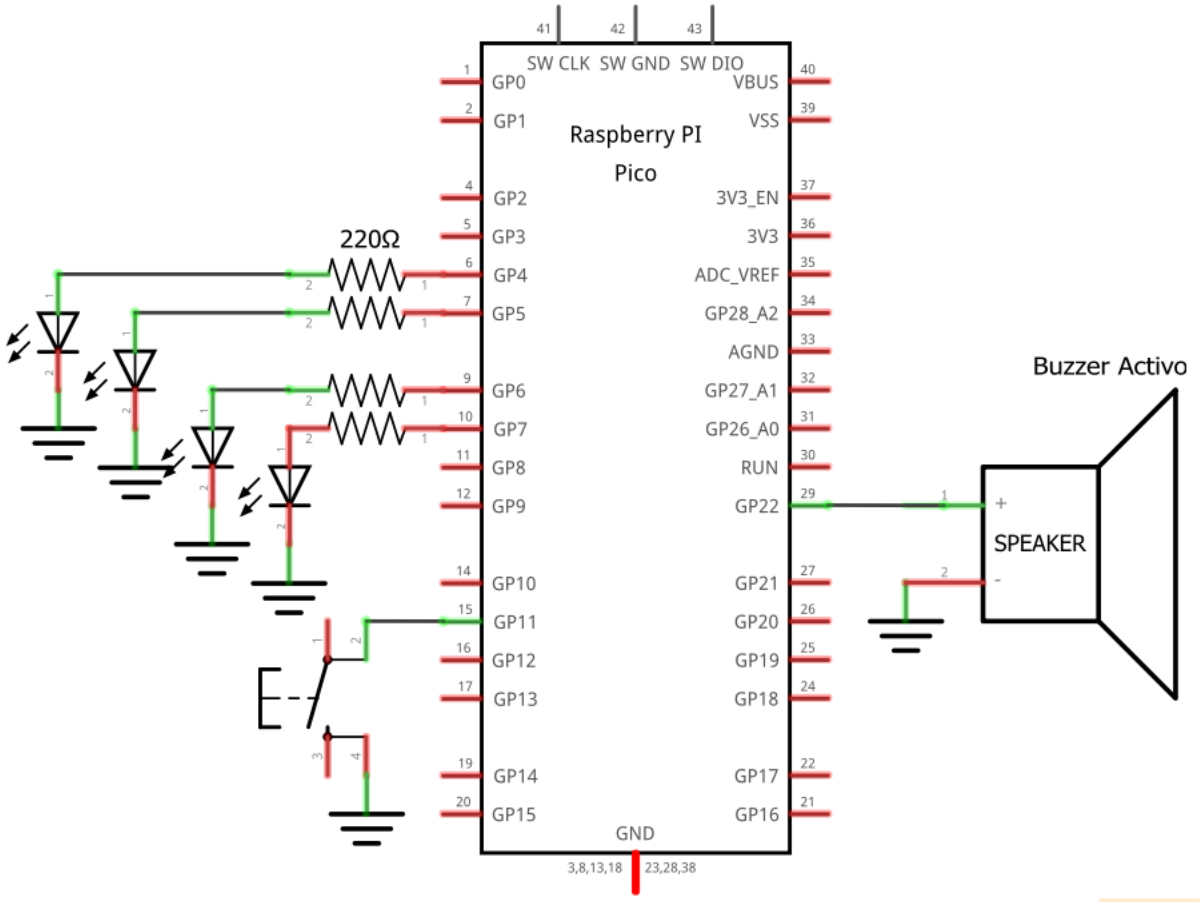


Figura 4-7. Esquemático; actividad 4

Entrada	**Salidas**					
GPIO11 S1	**GPIO4** LED4	**GPIO5** LED5	**GPIO6** LED6	**GPIO7** LED7	**GPIO22** ZUMBADOR	Consola de Thonny
'1'	0	0	0	0	0	Push button S1 liberado, '0' Salidas en bajo
'0'	1	1	1	1	1	Push button S1 presionado, '0' Salidas en alto

Tabla 4-2. Control GPIO4, GPIO5, GPIO6, GPIO7 a través de S1 (Pull_Up)

Actividad 5. Realizar el programa que genere las acciones solicitadas en la tabla 4-3; el circuito utilizado corresponde al indicado en la figura 4-2.

Entradas						**Salidas**								
GPIO						**GPIO**								
12 **S2**	11 S1	10 SW1_1	9 SW1_2	8 SW1_3		7 LED	6 LED	5 LED	4 LED	3 LED	2 LED	1 LED	0 LED	22 BUZZ
0	1	0	0	0		0	0	0	0	0	0	0	0	0
0	1	0	0	1		1	1	1	1	1	1	1	1	0
0	1	0	1	0		1	0	1	0	1	0	1	0	0
0	1	0	1	1		0	1	0	1	0	1	0	1	0
0	1	1	0	0		1 0 ⋱ 0	0 1 ⋱ 0	0 0 ⋱ 0	0 0 ⋱ 0	0 0 ⋱ 0	0 0 ⋱ 0	0 0 ⋱ 0	0 0 ⋱ 1	0 0 0 |
0	1	1	0	1		0 0 ⋱ 1	0 0 ⋱ 0	0 0 ⋱ 0	0 0 ⋱ 0	0 0 ⋱ 0	0 0 ⋱ 0	0 1 ⋱ 0	1 0 ⋱ 0	0 0 0 1
0	1	1	1	0		1 0 ⋱ 0 0 ⋱ 1	0 1 ⋱ 0 0 ⋱ 0	0 0 ⋱ 0 0 ⋱ 0	0 0 ⋱ 0 0 ⋱ 0	0 0 ⋱ 0 0 ⋱ 0	0 0 ⋱ 0 0 ⋱ 0	0 0 ⋱ 0 1 ⋱ 0	0 0 ⋱ 1 1 0 ⋱	0 0 ⋱ 0 0 0 1
0	1	1	1	1		1 0 0 0 0 0 1	0 1 0 0 0 1 0	0 0 1 0 1 0 0	0 0 0 1 0 0 0	0 0 0 1 0 0 0	0 0 1 0 1 0 0	0 1 0 0 0 1 0	1 0 0 0 0 0 1	0 0 0 0 0 0 1
0	0	1	1	1		Contador ascendente 0 – 9 (GPI03, GPIO4, GPIO5, GPIO6)								0
1	1	1	1	1		Contador descendente 0 – 9 (GPI03, GPIO4, GPIO5, GPIO6)								0
1	0	1	1	1		Notas musicales								1

Tabla 4-3. Acciones de control

Laboratorio de Microcomputadoras
Practica No. 5
Control de actuadores con GPIO

Objetivo. Reforzará las habilidades para programar y configurar las funciones GPIO para controlar motores de corriente directa, motores de pasos y servomotores a través del microcontrolador Raspberry Pi Pico. Estudiar la importancia de los amplificadores de potencia.

1. Introducción

Entre los actuadores más empleados se encuentran:

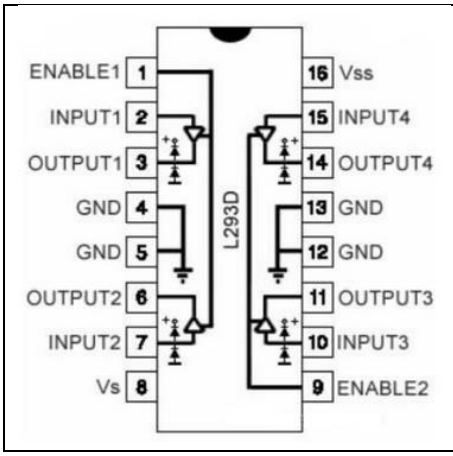
a. Motores de corriente directa
b. Motores a pasos
c. Servomotores

En cualquiera de los anteriores se genera un campo magnético producido por la circulación de corriente por sus devanados creando fuerzas de atracción y repulsión.

Un microcontrolador no otorga la corriente requerida para producir el movimiento de rotación en los motores, por lo que se hace indispensable el uso de un amplificador de corriente, que puede ser desde un solo transistor o un arreglo de cuatro transistores o contar con un driver de potencia disponible como el L2393, L298, TB6612, entre otros; la mayoría de ellos funcionando de manera parecida.

Motores de Corriente Directa

Para la práctica se emplea el driver L293, que tiene el siguiente encapsulado:



Terminal	Función
ENABLE1, ENABLE2	Habilitadores (Izq, Der)
INPUT1,INPUT2, INPUT3,INPUT4	Señales de control
OUTPUT1, OUTPUT2, OUTPUT3,UOUTUT4	Salidas, conexión a motores
GND	0 Volts
VSS	5 Volts
VS	Tensión del motor, puede ser desde 0.2 V a 32 V

Figura 5-1. Driver L293

Motores a pasos

Existen dos tipos de motores a pasos, los unipolares y los bipolares.

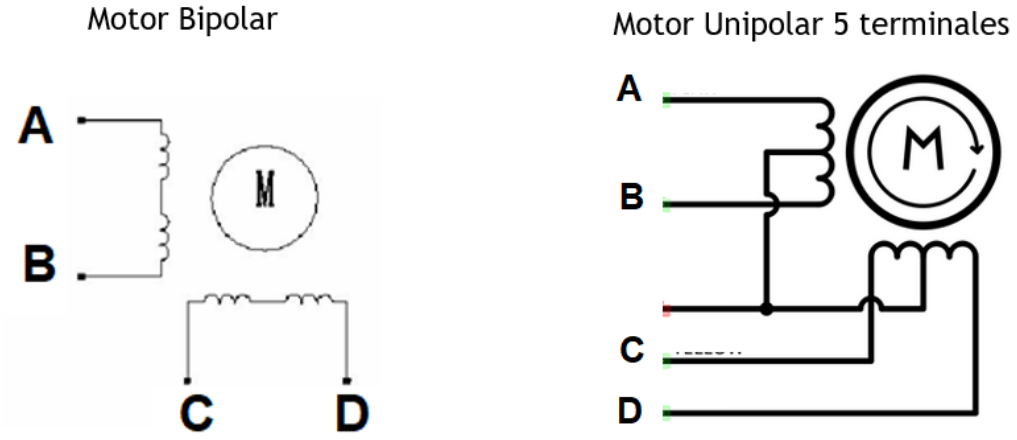


Figura 5-2. Motores a pasos

Para los motores bipolares se emplea regularmente dos puentes H para controlarlos, en este caso el L293D, mientras que los motores unipolares se controla cada bobina de manera independiente; es recomendable el empleo del driver ULN2003A.

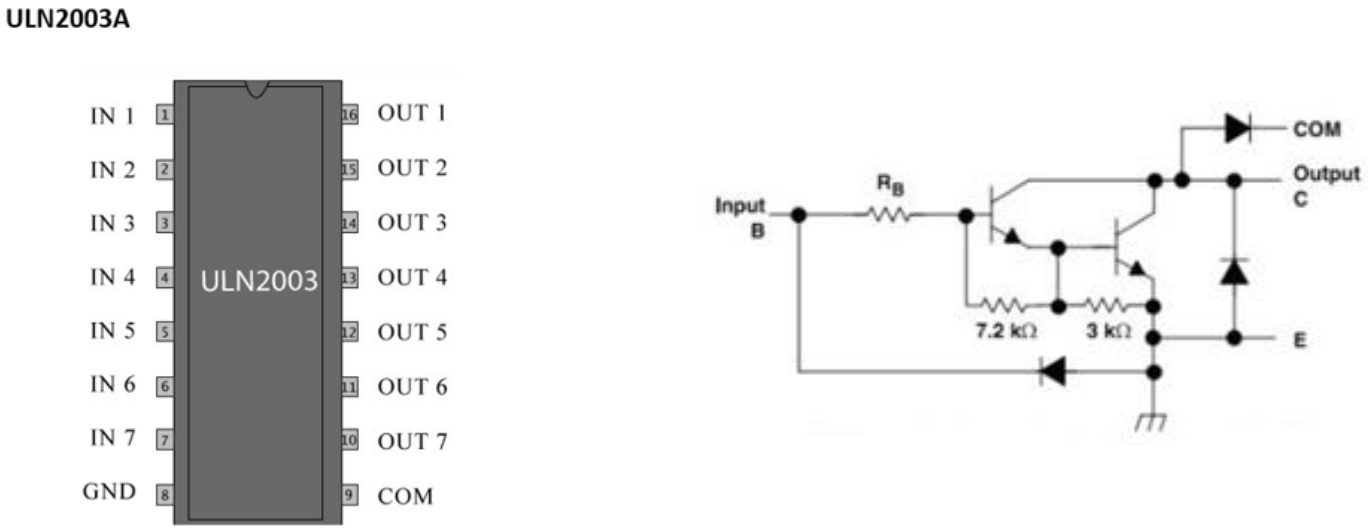


Figura 5-3. Driver UNL2003

El control de paso unipolares se realiza mediante tres técnicas:

- Pasos completos
- Medio paso
- Oleada

La técnica recomendada en esta práctica es la de pasos completos; que se describe en la figura 5-4.

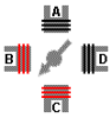
PASO	Bobina A	Bobina B	Bobina C	Bobina D	EFECTO
1	ON	ON	OFF	OFF	
2	OFF	ON	ON	OFF	
3	OFF	OFF	ON	ON	
4	ON	OFF	OFF	ON	

Figura 5-4. Secuencia de pasos completos para motores unipolares

Servo motores

El servo motor contiene en su encapsulado los mecanismos, que le periten funcionar sin requerir elementos externos; integra el sistema de control que permiten colocar en la posición deseada, el driver de potencia que amplifica la corriente del microcontrolador y el sistema de reducción en base de engranes para incrementar el torque.

Estructura interna

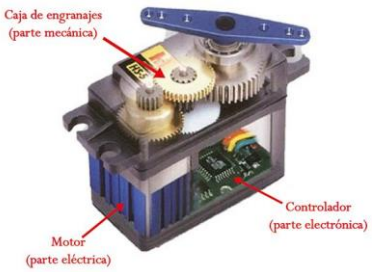


Diagrama a bloques

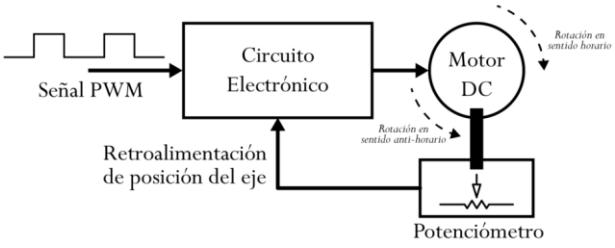


Figura 5-5. Descripción de servomotor

Como se puede ver en la figura 5-6, tiene un cable con tres terminales:

Servomotor

Terminal	
Café (1)	0 Volts
Roja (2)	5 Volts
Naranja (3)	Señal de control PWM

Figura 5-6. Servomotor y sus terminales

Para controlar la posición del cursor se ingresa un pulso en la terminal de entrada al servomotor, debe generar la señal PWM con periodo de 20 ms; esta señal debe modular el pulso en alto para que se encuentre en un tiempo comprendido entre 0.5 mS a 2.5 ms. La posición de 0º a 180º, como se muestra a continuación.

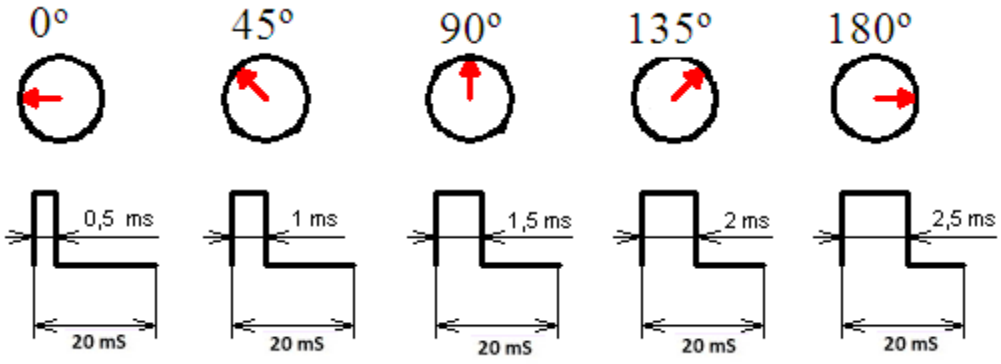


Figura 5-7. Pulsos para control de servomotor.

Material a utilizar para la práctica:

a b c

Figura 5-8. Moto-reductor, motor a pasos y servomotor

2. Requerimientos

2.1. Software.

- *Micropython*

2.2. Editor.

- *Thonny*

2.3. Hardware.

- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 1 para Raspberry Pi Pico*
- *1 L293D*
- *1 ULN2003*
- *1 Servomotor*
- *2 Motores de CD*
- *1 Motor a pasos*

Distribución de terminales.

Motores de corriente directa

Motor 1				Motor 2		
GPIO0	GPIO1	GPIO20		GPIO2	GPIO3	GPIO19
DIR1_M1	DIR2_M1	EN_M1		DIR1_M2	DIR2_M2	EN_M2

Motor a pasos

GPIO7	GPIO6	GPIO5	GPIO5
BOBINA A	BOBINA B	BOBINA C	BOBINA D

Servo Motor

GPIO21
Vo

Interruptores

GPIO 8	GPIO9	GPIO10	GPI011	GPIO12
SW1_1	SW1_2	SW1_3	PUSH BUTTON PULL_UP	PUSH BUTTON PULL_DOWN

Tabla 5-1. Asignación de GPIO; practica 5

El circuito es:

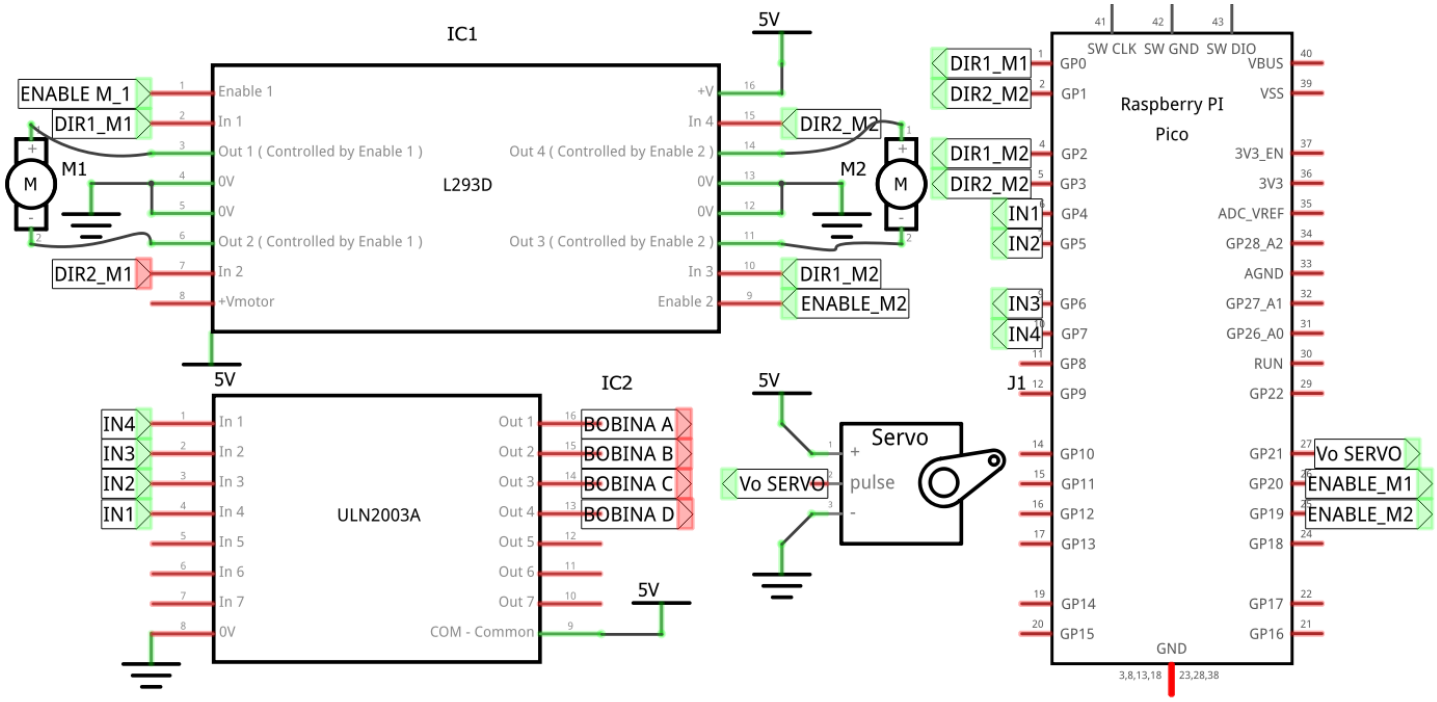


Figura 5-9. Módulo de actuadores

3. Desarrollo

Realizar las actividades solicitadas.

Actividad 1. De acuerdo a la distribución indicada en el circuito de la figura 5-10; realizar las acciones de control indicadas en la tabla 5-2. Conectar la fuente externa con 5V.

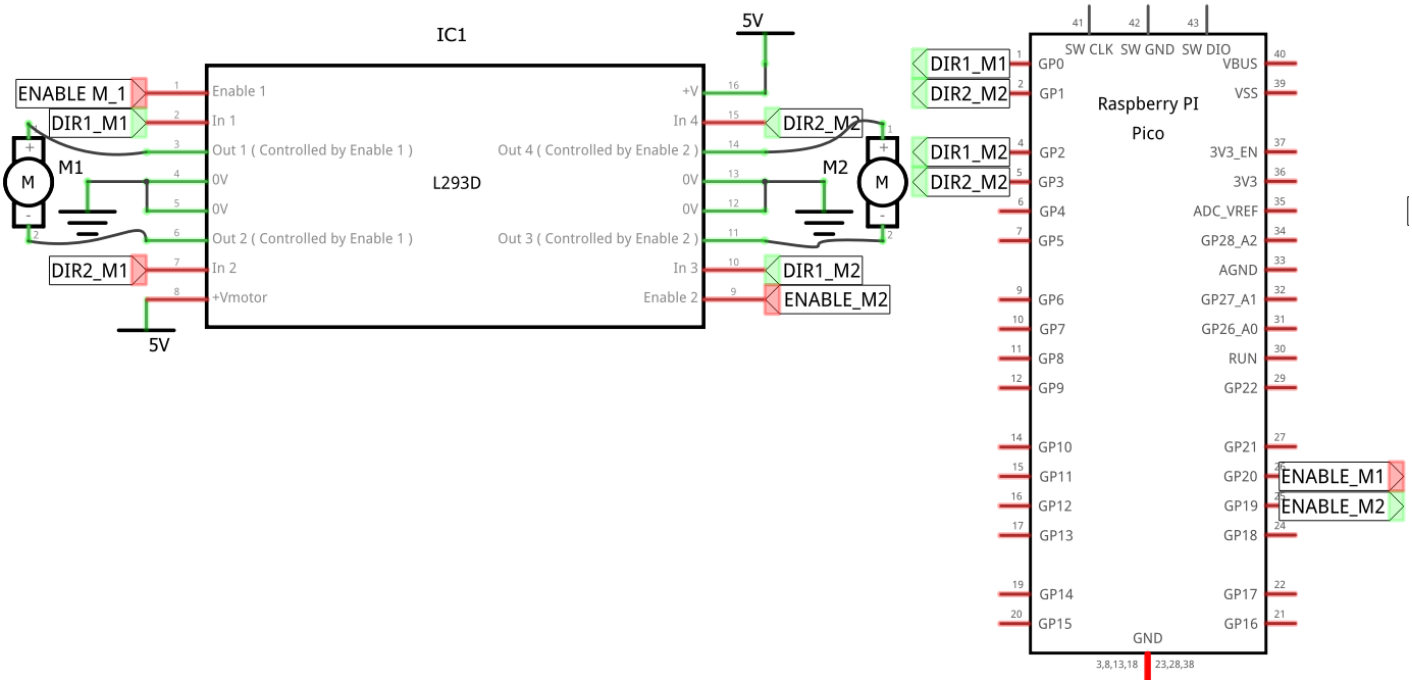


Figura 5-10. Control de motores de corriente directa

Entradas			Acciones	
SW1_1 ***GPIO10***	**SW1_2** *GPIO9*	**SW1_3** *GPIO8*	**MOTOR 1** ***GPIO0 – GPIO1-GPIO20***	**MOTOR 2** *GPIO2 – GPIO3-GPIO19*
0	0	0	**PARO**	PARO
0	0	1	**GIRA HORARIO**	PARO
0	1	0	**PARO**	GIRA HORARIO
0	1	1	**GIRA HORARIO**	GIRA HORARIO
1	0	0	**GIRA ANTIHORARIO**	GIRA HORARIO
1	0	1	**GIRA HORARIO**	GIRA ANTIHORARIO
1	1	0	**GIRA ANTIHORARIO**	GIRA ANTIHORARIO
1	1	1	**PARO**	PARO

Tabla 5-2. Controlador; actividad 1

Actividad 2. De acuerdo a la distribución indicada en el circuito de la figura 5-11; realizar las acciones de control indicadas en la tabla 5-3, considerar la secuencia indicada en la figura 5-4. Conectar la fuente externa con 5V.

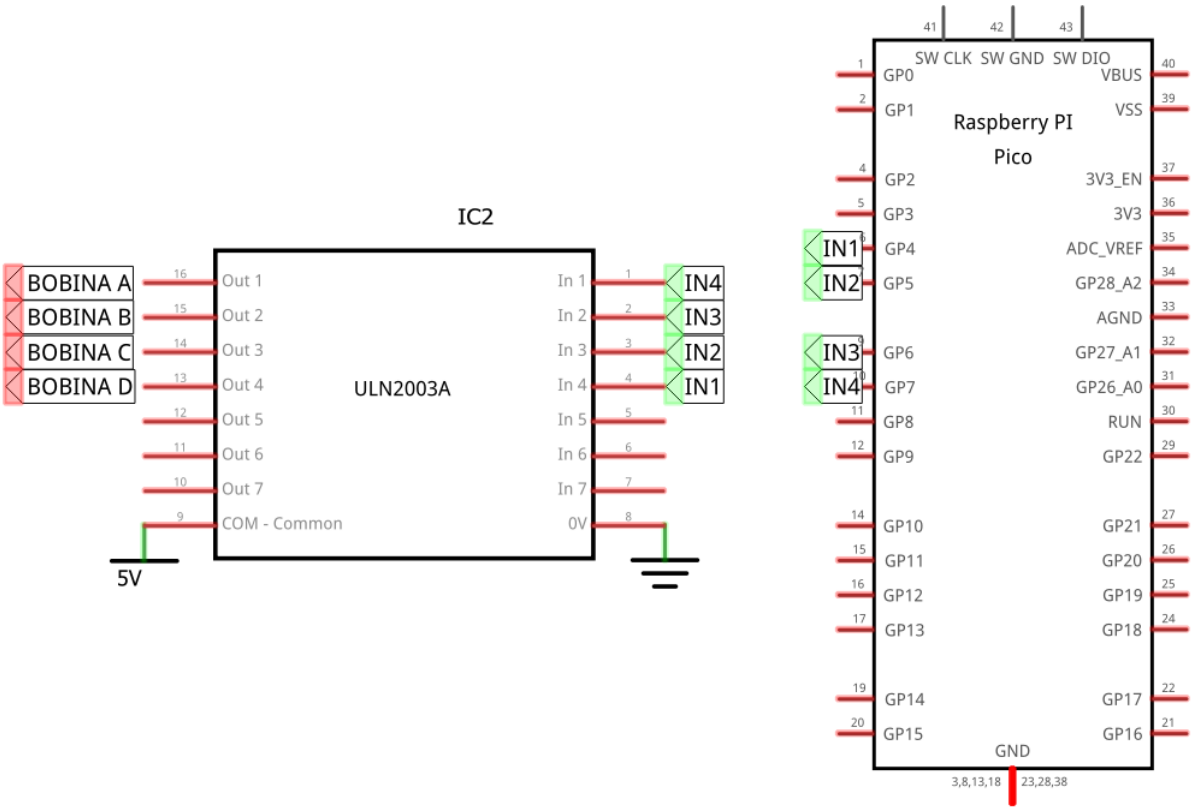


Figura 5-11. Esquemático; actividad 2

Entradas			Acciones
SW1_1 ***GPIO10***	**SW1_2** *GPIO9*	**SW1_3** *GPIO8*	**Motor a pasos** ***GPIO7 – GPIO6-GPIO5 – GPIO4***
0	0	0	**PARO**
0	0	1	**GIRA HORARIO**
0	1	0	**GIRA SENTIDO ANTIHORARIO**
0	1	1	**GIRA 90º SENTIDO HORARIO cada 2 seg**
1	0	0	**GIRA 180º SENTIDO ANTIHORARIO cada 3 seg**
1	0	1	**GIRA 360º SENTIDO HORARIO cada 4 seg ***
1	1	0	**GIRA 5 REVOLUCIONES SENTIDO HORARIO ***
1	1	1	**GIRAR 10 REVOLUCIONES SENTIDO ANTIHORARIO ***

Tabla 5-2. Controlador; actividad 2

* *Para los tres últimos casos, cada que se concluya una revolución en el motor a pasos, activar el Zumbador conectado en el GPIO22 durante 300 ms.*

Actividad 3. De acuerdo a la distribución indicada en el circuito de la figura 5-12; realizar las acciones de control indicadas en la tabla 5-4, considerar la información indicada en la figura 5-7. Conectar la fuente externa con 5V.

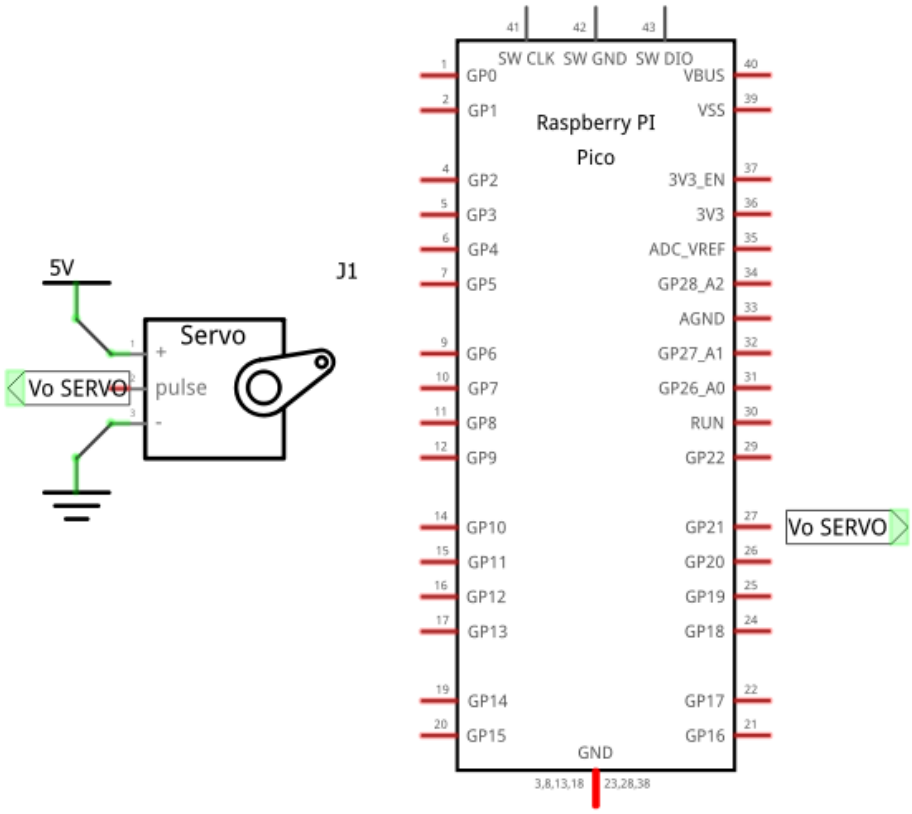


Figura 5-12. Esquemático control del servomotor

Entradas					Acciones
S2 *GPIO12*	S1 *GPIO11*	SW1_1 *GPIO10*	SW1_2 *GPIO9*	SW1_3 *GPIO8*	**SERVOMOTOR *GPIO21***
0	1	0	0	0	**PARO**
0	1	0	0	1	**SERVO A 0º**
0	1	0	1	0	**SERVO A 45º**
0	1	1	0	0	**SERVO A 90º**
0	0	0	0	0	**SERVO A 135º**
1	1	0	0	0	**SERVO A 180º**

Tabla 5-3. Controlador; actividad 3

Actividad 4. Realizar un programa, de manera que controle la posición del servo de forma automática, iniciando en la posición de 0º hasta 180º y viceversa.

Laboratorio de Microcomputadoras
Practica No. 6
Convertidor Analógico Digital, control PWM

Objetivo. Conocer el funcionamiento del ADC (convertidor analógico digital) para realizar aplicaciones que requieran procesar señales en el mundo continuo, convertir señales provenientes de sensores; aprender el funcionamiento del control PWM.

1 Introducción

a. Convertidor Analógico Digital.

El convertidor AD se encarga de convertir una señal continua a una discreta; mediante técnicas de discretización; entre las más conocidas están las de aproximaciones sucesivas.

Raspberry Pi Pico dispone al usuario de 3 terminales para ingresar señales analógicas, designadas como ADC0, ADC1 y ADC2, ubicados como función alterna en los GPIO26, GPIO27 y GPIO28 respectivamente; además del canal interno referenciado como ADC4, el cual está asignado para un sensor de temperatura interno. La resolución del convertidor es de 12 bits; el método utilizado en Micropython genera un resultado de 16 bits. La tensión de entrada deberá estar entre 0 y 3.3 Volts.

Los requerimientos para la realización de la práctica se muestran en el siguiente diagrama:

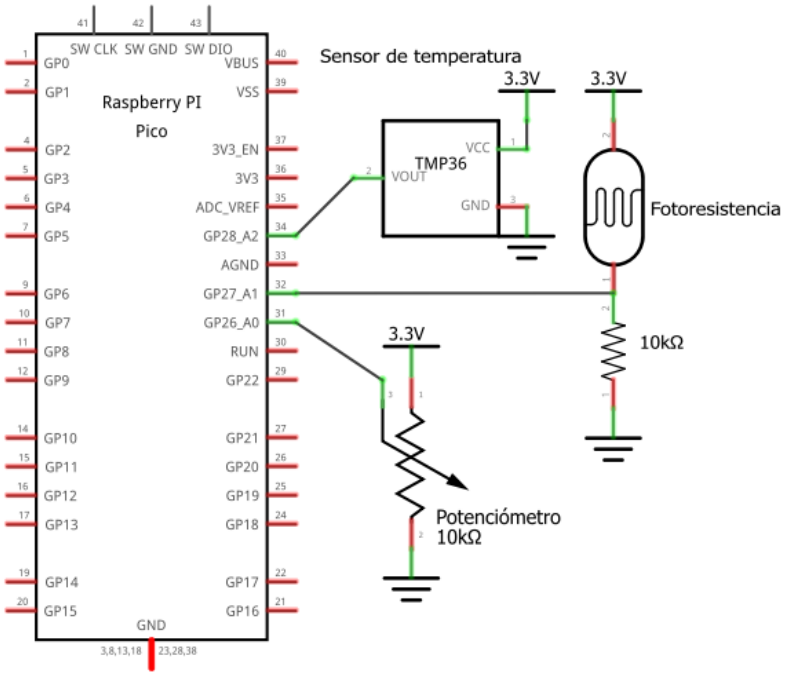


Figura 6-1. Circuito esquemático; practica 6

Dispositivo analógico			
Potenciómetro	Foto-resistencia	TMP 36	Sensor Temp. interno
ADC0	ADC1	ADC2	ADC4
Canal			

Tabla 6-1. Asignación de terminales a entradas analógicas

Los recursos de software requeridos para uso del convertidor AD son:

from machine import ADC	**#**	***Importar la clase ADC del módulo machine***
canal = ADC (GPIO)	#	*Crear un objeto de la clase ADC de nombre canal, ubicado en el pin GPIO indicado; este podría ser: 26, 27, 28 o 29, también es reconocido: ADC0, ADC1, ADC2 o ADC4, o en su defecto como: 0, 1, 2 o 4; de manera indistinta.*
valor = canal.read_u16()	#	*Utiliza el método read_u16() para obtener el resultado de la conversión A/D del canal previamente definido, a la variable valor; el resultado será en formato de 16 bits (0 – 65535).*

b. Modulación de Ancho de Pulso PWM.

El módulo se encarga de generar el control de modulación de ancho de pulso conocido como PWM; este consiste en la generación de un tren de pulsos con periodo constante y lo que varia es el tiempo de la señal en alto y bajo, obteniendo un ciclo de trabajo variable D (duty cycle). La técnica PWM es muy utilizada en diversos procesos, se controla el flujo de corriente que circulará a través de un dispositivo, con lo que se considerará un promedio de tensión presente en un instante dado; es ampliamente empleado en control de motores para implementar algoritmos de tipo PID, difusos, redes neuronales, algoritmos genéticos, inteligencia artificial, entre los más difundidos.

La Raspberry Pico tiene 8 canales disponibles para la función PWM; es posible asignar a cualquier GPIO deseado.

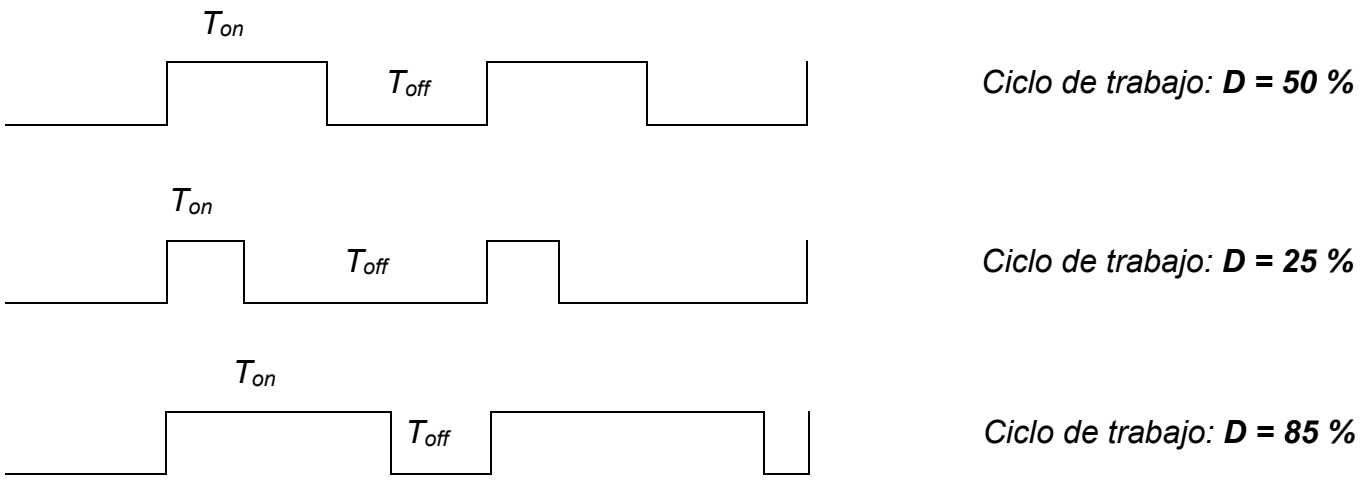


Figura 6-2. Control PWM

Los recursos de software requeridos para uso del control PWM son:

from machine import PWM	**#**	***Importar la clase PWM del módulo machine***
pwm1 = PWM (Pin(#GPIO))	#	*Crear un objeto de la clase PWM , ubicado en el pin GPIO indicado (0 – 28).*
pwm1.freq(valor)	#	*Configura el valor de la frecuencia deseada PWM.*
pwm1.duty_u16(duty)	#	*Define el ciclo de trabajo; el duty podrá estar entre o y 65535.*

2 Requerimientos

2.1. Software.

- *Micropython*

2.2. Editor.

- *Thonny*

2.3. Hardware.

- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 1 para Raspberry Pi Pico*
- *Sensor de temperatura interno*
- *Potenciómetro 10 KΩ*
- *Foto-resistencia*
- *Sensor de temperatura TMP36*

3 Desarrollo.

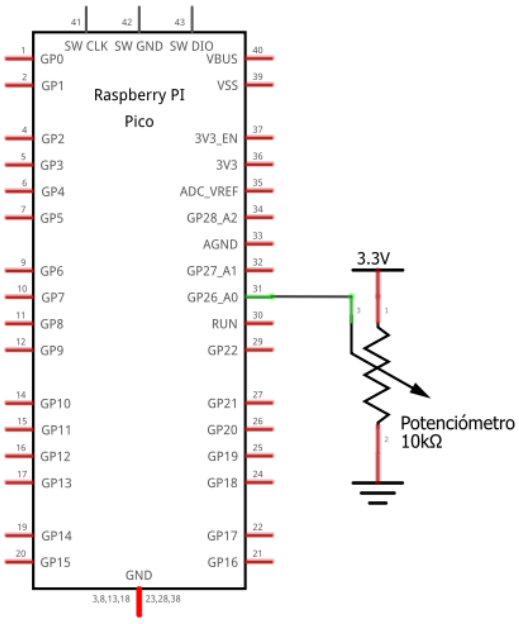
Realizar las actividades solicitadas.

Actividad 1. Comentar el siguiente programa, e indicar que hace:

```
from machine import ADC, Pin
import time

Sensor = ADC(4)

while True:

    Valor = Sensor.read_u16()*(3.3/65535)
    Temp = 27-(Valor-0.706)/0.001721
    print(Temp)
```

Figura 6.3. Código de prueba

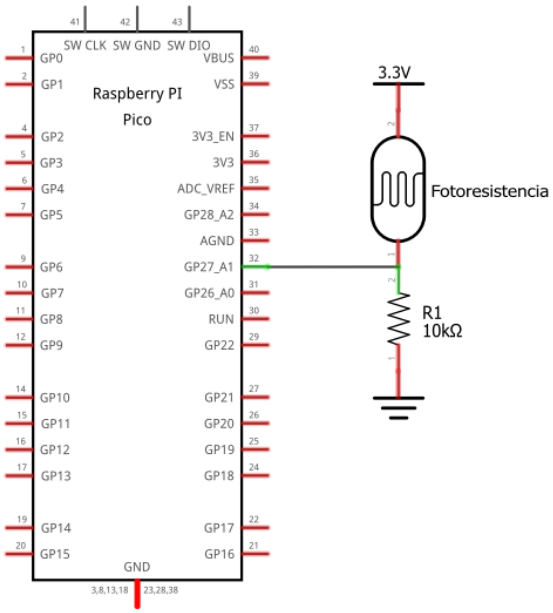
Actividad 2. Realizar un programa que despliegue en la consola de Thonny el resultado de la conversión y el voltaje generado por el potenciómetro; actualizar cada segundo.



Conversión = 65535 ; Voltaje = 3.30 Volts
Conversión = 0 ; Voltaje = 0 Volts

Figura 6-4. Circuito y ejecución; actividad 2

Actividad 3. Empleando el divisor de voltaje generado por la foto-resistencia; tomar la lectura de voltaje, registrar el máximo y mínimo obtenido, fijar un valor de referencia. Mientras la lectura es menor a la referencia, mantenga en bajo el GPIO7 y cuando sobrepase el valor configurado lo ponga en alto.



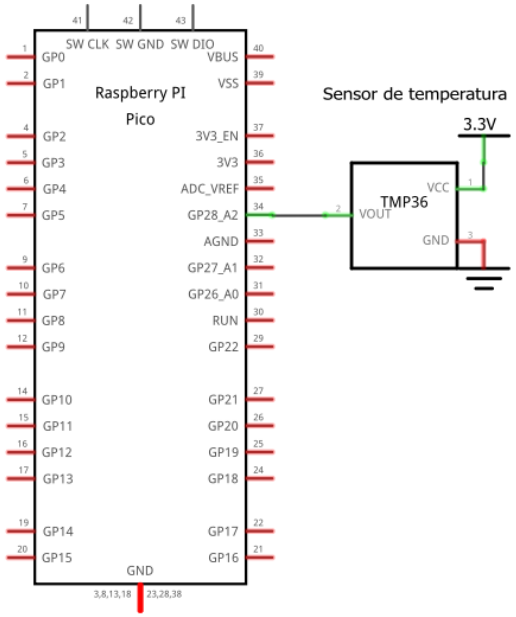
Lectura de la foto-resistencia:

V_{MIN} =
V_{MAX} =
V_{REF} =

$V_{Foto\_resistencia} < V_{REF}$	$V_{Foto\_resistencia} > V_{REF}$
GPIO7 = '0'	**GPIO7 = '1'**

Figura 6-5. Circuito de foto-resistencia; actividad 3

Actividad 4. Realizar un programa que muestre el valor de temperatura tanto en ºC como ºF, en la consola de Thonny del sensor interno y del sensor TMP36; indicando cual de ellos tiene valor mayor al otro.



Temperatura interna = 19ºC; 66ºF
Temperatura externa TMP36 = 20ºC; 68ºF
El sensor TMP36 tiene el valor mayor de temperatura

Figura 6-6. Circuito sensor TMP36 y sensor interno; actividad 4

Actividad 5. Escribir el siguiente programa, comentar e indicar que hace:

```
from machine import Pin, PWM
import time
pwm=PWM(Pin(1))
pwm.freq(1000)

while True:
    for duty in range(0, 65535,500):
        pwm.duty_u16(duty)
        time.sleep(0.05)
    pwm.duty_u16(0)
    time.sleep(2)
```

Figura 6-7. Código; actividad 5

Actividad 6. Modificar el programa para que incremente y decremente de manera automática el valor de PWM en la terminal GPIO0 y refleje el valor inverso en el pin GPIO1.

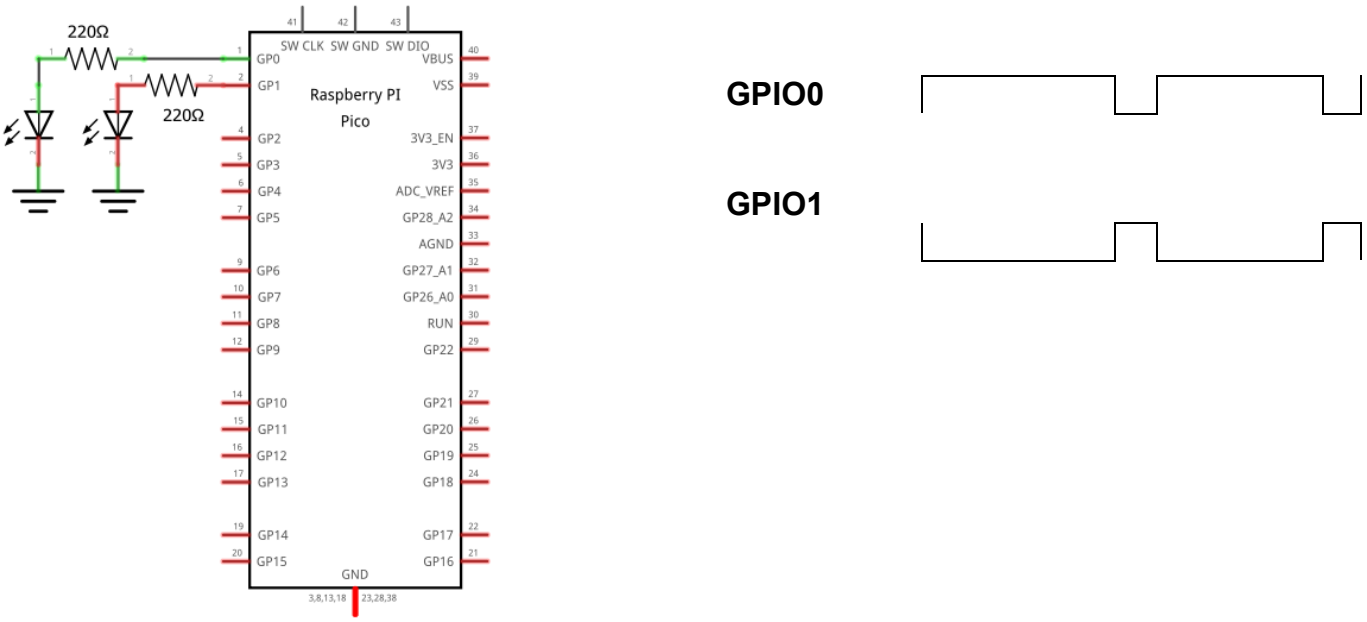


Figura 6-8. Circuito y oscilograma requerido; actividad 6

Actividad 7. Realizar las modificaciones necesarias para realizar el control anterior pero ahora a través de la selal analógica proveniente del potenciómetro.

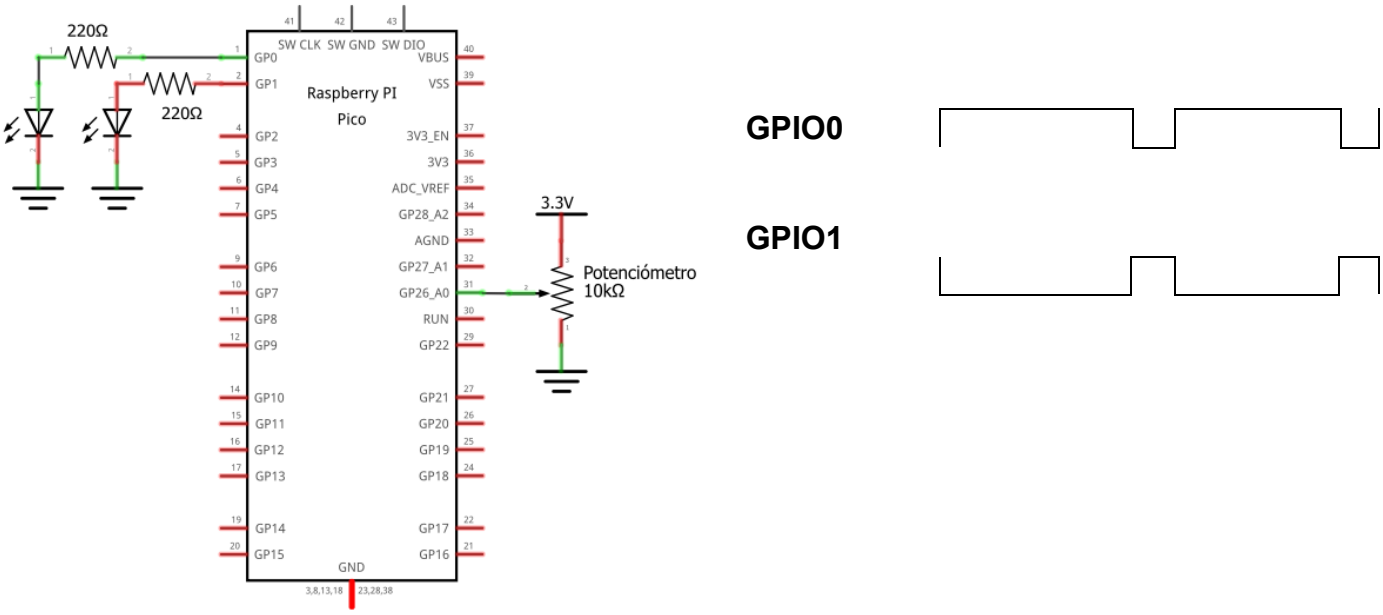


Figura 6.9. Control PWM por medio de entrada analógica; actividad 7

Laboratorio de Microcomputadoras
Practica No. 7
Comunicación Serie Asíncrona UART

Objetivo. Aprender el funcionamiento de la comunicación serial en la modalidad asíncrona para la transferencia de información entre diferentes dispositivos por medios alámbricos e inalámbricos.

1. Introducción

La comunicación serie asíncrona se establece entre un dispositivo fuente y un destino, utilizando dos líneas, de tipo unidireccional o bidireccional; se debe configurar la razón de transferencia (*Baud*), en ambos dispositivos. Las microcomputadoras y microcontroladores disponen de un módulo encargado de este formato de comunicación, conocidos como **UART o USART**.

La comunicación asíncrona requiere de solo dos vías:

UART		
Tx	**Transmisor**	**Línea de transmisión de datos.**
Rx	Receptor	Línea de recepción de datos.

En caso de comunicación alámbrica, se requiere **GND** en ambos dispositivos con conexión cruzada (**Tx1-Rx2 y Rx2-Tx1**); también es posible comunicación inalámbrica a través de diversos medios (*RC, Bluetooth, Infrarroja, WiFi*).

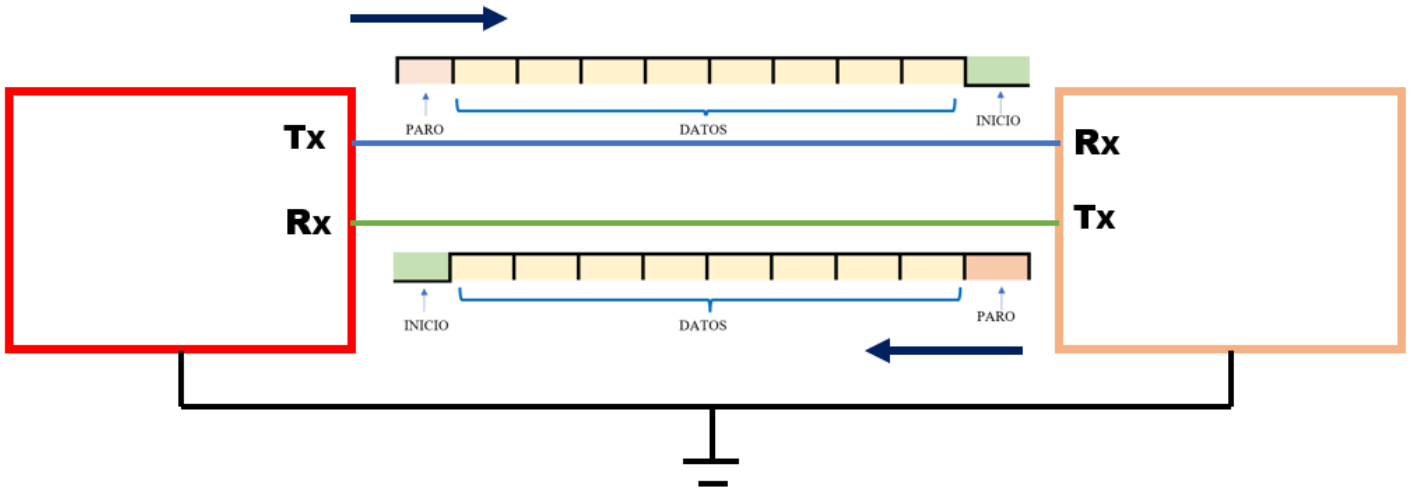


Figura 7-1. Comunicación asíncrona UART

El protocolo se define con bits de ***datos*** y bits de ***control*** (*inicio, paro, paridad*); el estándar consta de 1 bit de inicio, 8 bits de datos y 1 bit de paro, con lo que se define el frame (marco) del dato a transmitir o recibir.

Raspberry Pi Pico

Tiene dos puertos serie asíncrono UART, denominados UART0 y UART1; asignados como función alterna de los GPIO:

UART0			
UART0 TX	GPIO0	GPIO12	GPIO16
UART0 RX	GPIO1	GPIO13	GPIO17

UART1		
UART1 TX	GPIO4	GPIO8
UART1 RX	GPIO5	GPIO9

Tabla 7-1. Distribución de puertos UART

Los recursos de software requeridos para uso del módulo UART son:

from machine import UART, Pin	#	*Importar la clase UART del módulo machine*
uart = UART(Num_uart,baudrate=baud,tx=Pin(GP_Tx), rx=Pin(GP_Rx))		
	#	*Creación de la instancia UART* *uart – Nombre del objeto* *Num_uart – es el número de UART a usar 0/1* *Baud – Configuración de baudaje* *GP_Tx – GPIO a utilizar como Tx* *GP_Rx – GPIO a utilizar como Rx*
uart.any()	#	*Espera la recepción de datos*
dato= uart.read()	#	*Asignación del dato recibido*
uart.write	#	*Transmisión de datos*

2 Requerimientos

2.1. Software.

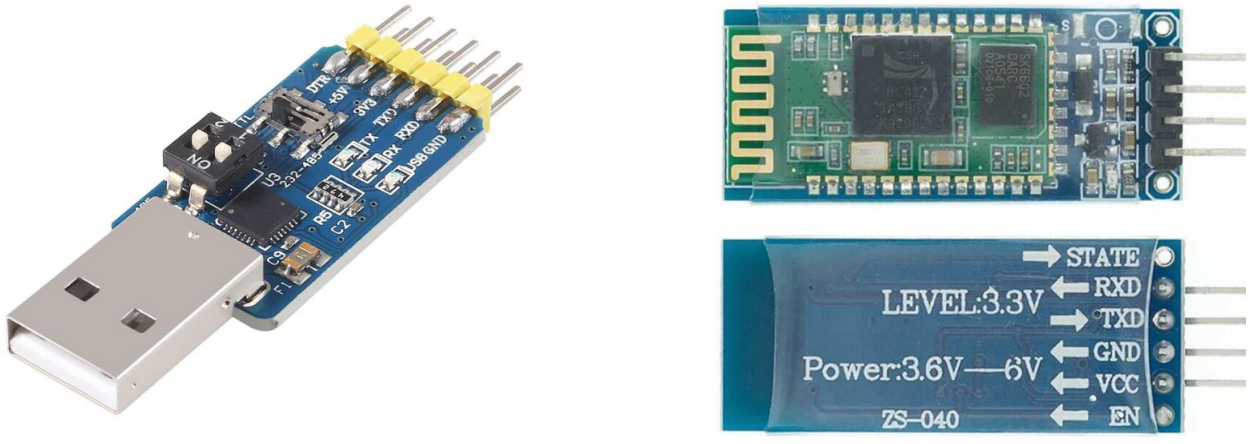
- *Micropython*

2.2. Editor.

- *Thonny*

2.3. Hardware.

- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 1 para Raspberry Pi Pico*
- *Sensor de temperatura interno*
- *8 Leds*
- *8 Resistencias de 220 Ω*
- *Módulo convertidor USB-TTL CP2102*
- *Módulo Bluetooth HC05 o HC06*



a. Convertidor USB – TTL CP2102 b. Módulo Bluetooth HC05/HC06

Figura 7-2. Módulos de comunicación serie asíncrona UART

3. Desarrollo.

Realizar las actividades solicitadas, en ellas se realizará la comunicación serie asíncrona, será empleada el formato REPL usando la interfaz por default; así como las siguientes interfaces.

Actividad 1. Escribir el siguiente programa:

```
import select
import sys
import time
import machine

poll_obj = select.poll()
poll_obj.register(sys.stdin,1)
sys.stdout.write("Esperando recepción de datos \n")
print("Teclea un caractér y luego <enter>")
while True:
    if poll_obj.poll(0):
        ch = sys.stdin.read(1)
        sys.stdout.write("Dato recibido \n")
        print("Hola UNAM")
    time.sleep(0.1)
```

Figura 7-3. Código de ejemplo; actividad 1

a. *Comentar el código.*
b. *Indicar que hace.*
c. *Ejecutar.*
d. *Ingresar datos desde la consola de Thonny.*
e. *Explicar la diferencia entre:*
    a. *sys.stdout.write("Esperando recepción de datos \n") .*
    b. *print("Teclea un caracter y luego <enter>").*

Actividad 2. Para este ejercicio, realizar los pasos siguientes:

a. Ubicar el COM de conexión de la Raspberry Pi.

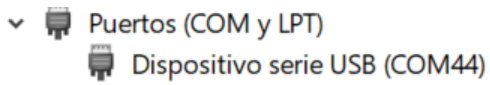


Figura 7-4. Información del Administrador de Dispositivos

b. Abrir la terminal de su preferencia.

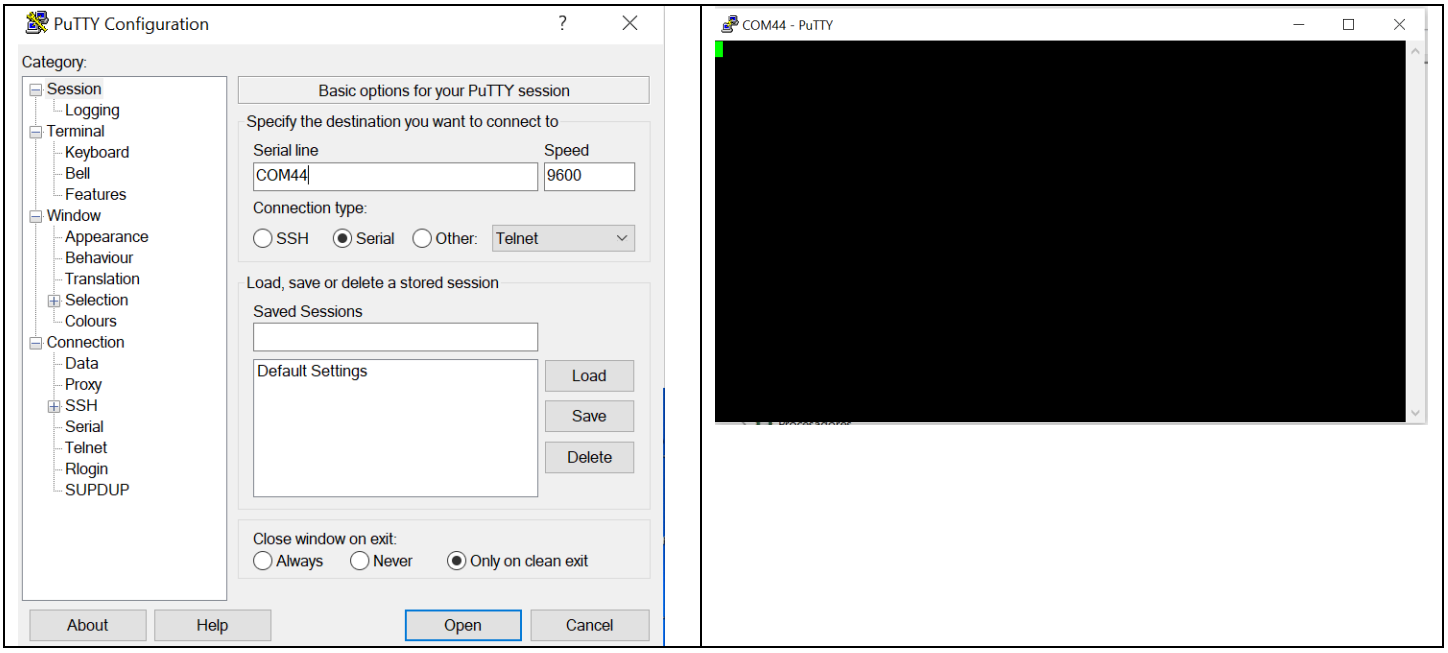


Figura 7-5. Configuración de Putty

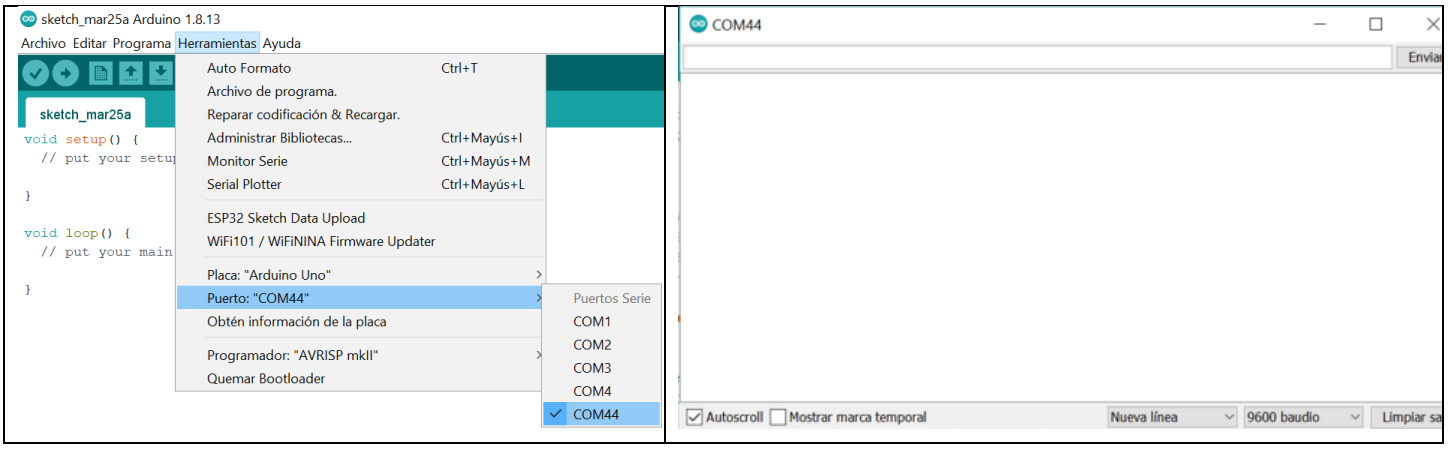


Figura 7-6. Configuración en monitor de Arduino

c. Explicar si existe alguna diferencia en la ejecución y cuál sería el motivo.

Actividad 3. Utilizando el formato de la actividad 1, realizar un programa que reciba comandos a través del puerto serie y ejecute las acciones indicadas.

Dato recibido	Acción
'0'	GPIO25 = OFF
'1'	GPIO25 = ON

Tabla 7-2. Control; actividad 3

Actividad 4. Realizar el siguiente procedimiento:

a. Escribir el siguiente programa.
b. Comentar el código e indicar que hace.
c. Conectar el módulo CP2102.
d. Ubicar el puerto asignado.
e. Comprobar el funcionamiento

```
from machine import Pin,UART
import time

uart = UART(0, baudrate=9600, tx=Pin(16), rx=Pin(17))
uart.init(bits=8, parity=None, stop=1)
led = Pin(25, Pin.OUT)
uart.write('Inicia Comunicacion Serie')
while True:
    if uart.any() > 0:
        data = uart.read()
        uart.write(data)
        led.toggle()
    time.sleep(1)
```

Figura 7-7. Código de ejemplo; actividad 4

Actividad 5. Siguiendo el procedimiento de la actividad No. 4; realizar un programa que ejecute las acciones indicadas.

Dato UART	Acción
'1'	Mostrar el voltaje del Potenciómetro
'2'	Mostrar el resultado de la conversión AD de la foto-resistencia en hexadecimal.
'3'	Mostrar la temperatura del sensor TMP36 en ºC, ºF y ºK.
'4'	Reproducir la nota DO en el zumbador
'5'	Prender y apagar 5 veces GPIO0 al GPIO7
'6'	Corrimiento a la derecha de los 8 leds GPIO0 – GPIO7
'7'	Corrimiento a la izquierda de los 8 leds GPIO0 – GPIO7

Tabla 7-3. Tabla de control; actividad 5

Actividad 6. Conectando el módulo bluetooth, realizar un programa que ejecute las acciones indicadas en la tabla 7-4.

Dato recibido	Acción
'A'	GPIO0 = ON
'T'	GPIO1 = ON
'D'	GPIO2 =ON
'I'	GPIO3 =ON
S	GPIO0:GPIO3 = OFF

Tabla 7-4. Control; actividad 6

Nota: Descargar la aplicación de control bluetooth de la página del Laboratorio de Microcomputadoras.

Actividad 7. Realizar un programa que controle el funcionamiento de los motores de Corriente Directa de la practica 5, por medio de la comunicación serie empleando el módulo Bluetooth.

Comando Puerto Serie	ACCION	
	MOTOR 1	MOTOR 2
"S"	PARO	PARO
"A"	HORARIO	HORARIO
"T"	ANTI-HORARIO	ANTI-HORARIO
"D"	HORARIO	ANTI-HORARIO
"I"	ANTI-HORARIO	HORARIO

Tabla 7-5. Control UART; actividad 7

Nota: Descargar la aplicación de control bluetooth de la página del Laboratorio de Microcomputadoras.

Laboratorio de Microcomputadoras
Practica No. 8
Comunicación Serie SPI

Objetivo. Aprender el funcionamiento de la comunicación SPI; realizar comunicación entre diferentes componentes por medio de la comunicación serie síncrona en la modalidad SPI, estudiar librerías para controlar pantallas SPI.

1 Introducción

La comunicación SPI corresponde a un formato síncrono; permite la transferencia de información entre dispositivos, pudiendo ser entre una microcomputadora o microcontrolador y uno o más periféricos; este protocolo requiere generalmente cuatro señales de control:

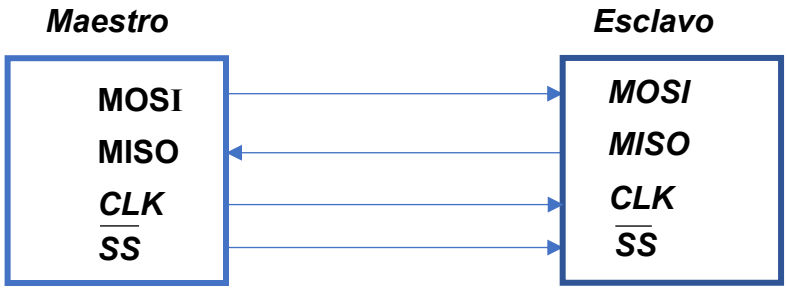


Figura 8-1. Comunicación serie asíncrona SPI

SPI		
MOSI	Master Output Slave Input	Transferencia de datos, iniciada por el maestro.
MISO	Master Input Slave Output	Datos solicitados al esclavo.
CLK	Clock	Señal de reloj que sincroniza el reconocimiento la información solicitada.
$\overline{SS}$	Slave Select	Selección del esclavo; por lo regular activada en bajo.

De acuerdo al fabricante podría asignarle un nombre distinto, pero la función seguirá siendo la misma; de igual manera podría contar con más terminales para controlarlo.

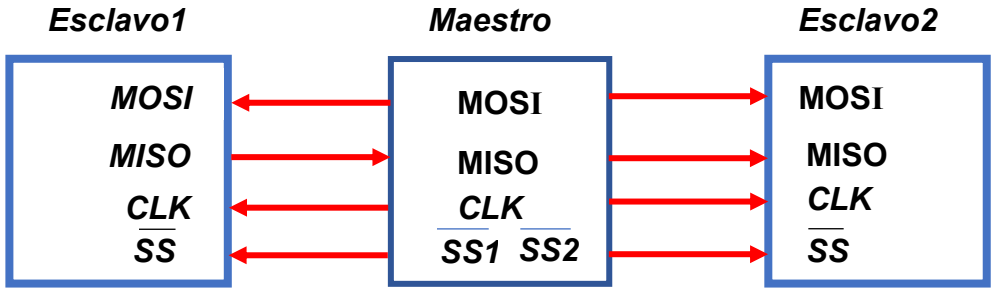


Figura 8-2. Comunicación serie asíncrona SPI, un maestro y dos esclavos

SPI en Raspberry Pi Pico

Tiene dos puertos serie síncronos SPI, designados como SPI0 y SPI1; la función alterna de los GPIO queda de la siguiente manera:

SPI0			
SPI0 RX	GPIO0	GPIO4	GPIO16
SPI0 CSn	GPIO1	GPIO5	GPIO17
SPI0 SCK	GPIO2	GPIO6	GPIO18
SPI0 TX	GPIO3	GPIO7	GPIO19

SPI1		
SPI1 RX	GPIO8	GPIO12
SPI1 CSn	GPIO9	GPIO13
SPI1 SCK	GPIO10	GPIO14
SPI1 TX	GPIO11	GPIO15

Tabla 8-1. Distribución de puertos SPI

A continuación se presentan los métodos y funciones disponibles para emplear el SPI; es altamente recomendable consultar la documentación oficial de Micropython para obtener mayor información. La comunidad alrededor del mundo, ha aportado librerías de dispositivos SPI, lo que ha facilitado la implementación de aplicaciones con ellos.

from machine import SPI, Pin	#	*Importar la clase SPI del módulo machine*
spi = SPI (Num_spi,baudrate=baud, polarity = Conf, sck =Pin(GP_sck), mosi=Pin(GP_mosi))		
	#	*Creación de la instancia SPI* *spi – Nombre del objeto* *Num_spi – Número de SPI a usar 0/1* *Baud – Configuración de la tasa de transferencia* *Conf – Definición de la polaridad* *GP_sck – GPIO a utilizar como sck* *GP_mosi – GPIO a utilizar como mosi*
cs = Pin(GP_cs, Pin.OUT)	#	*Define el pin GPIO_cs, como selector del periférico externo (esclavo)*
spi.write(datos_a_transmitir)	#	*Transfiere información a un periférico externo a través del SPI*
datos_recibidos= spi.read(dat)	#	*Recepción de datos del periférico externo SPI*

2 Requerimientos

2.1. Software.

- *Micropython*

2.2. Editor.

- *Thonny*

2.3. Hardware.

- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 2 para Raspberry Pi Pico*
- *Sensor de temperatura interno*
- *Display de 8 dígitos MAX7219 SPI*
- *Matriz 4 segmentos MAX7219 SPI*
- *Display TFT ST7735 SPI*

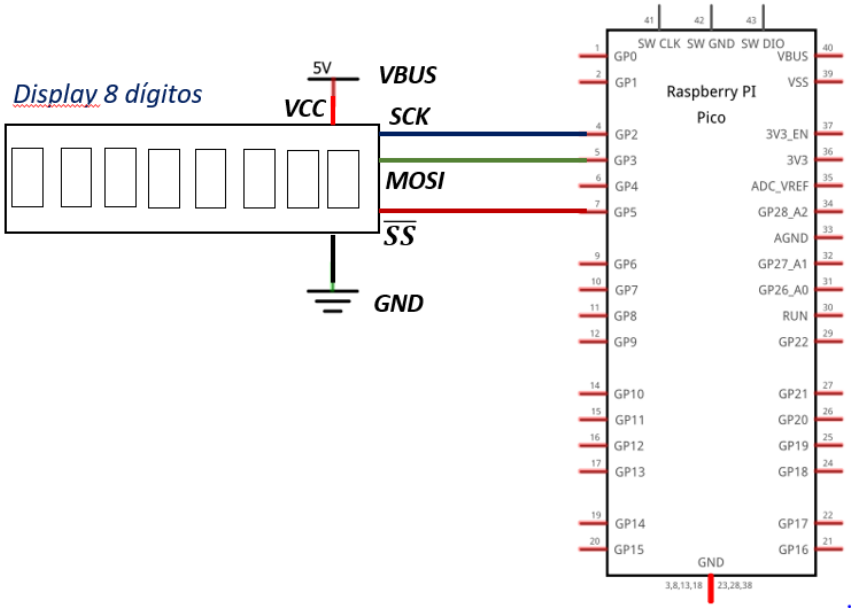
a. Display MAX7219 SPI b. Matriz MAX7219 SPI c. Display TFT ST7735 SPI

Figura 8-3. Display; comunicación SPI

3 Desarrollo

Realizar las actividades solicitadas, investigar el funcionamiento de los tres módulos utilizados en la práctica; estudiar las librerías empleadas.

Actividad 1. Escribir el siguiente programa, cerciorarse que el módulo Display de MAX7219 de 8 dígitos, se encuentre conectado en la posición correcta; investigar el funcionamiento de la librería utilizada; comentar el código.




```
from machine import Pin, SPI
import max7219_8digit
import time

spi = SPI(0, baudrate=10000000, polarity=1, phase=0, sck=Pin(2), mosi=Pin(3))
ss = Pin(5, Pin.OUT)
display = max7219_8digit.Display(spi, ss)

display.write_to_buffer("01234567")
display.display()
time.sleep(1)
```

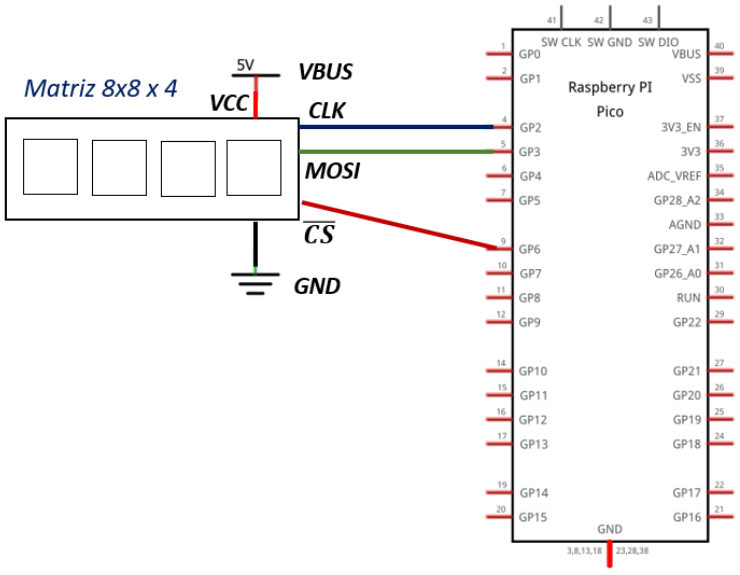
Figura 8-4. Código ejemplo; actividad 1

Actividad 2. Realizar el control indicado en la tabla.

Señal de control	Estado	Display de 8 dígitos
GPIO12	Activado	Inicia cuenta decimal ascendente (incremento cada 0.5 seg.)
GPIO13	Activado	Inicia cuenta decimal descendente (decremento cada 0.5 seg.)

Tabla 8-2. Señales de control; actividad 2

Actividad 3. Escribir el siguiente programa, cerciorarse que el módulo Matriz MAX7219 de 4 elementos matriciales de 8x8, se encuentre conectado en la posición correcta; investigar el funcionamiento de la librería max7219.py; comentar el código.




```
import max7219
from machine import Pin, SPI
from time import sleep

num_display = 1

spi = SPI(0, baudrate=10000000, polarity=1, phase=0, sck=Pin(2), mosi=Pin(3))
cs_pin = Pin(6, Pin.OUT)

display = max7219.Matrix8x8(spi, cs_pin, num_display)

display.fill(0)
display.text('0',0,1,1)
display.show()
sleep(3)
```

Figura 8-5. Código de ejemplo; actividad 3

Actividad 4. Usando la matriz de leds SPI, desplegar los siguientes mensajes, usar intervalos de tiempo de 2 segundos entre cada uno de ellos (puede ser texto fijo o con desplazamiento).

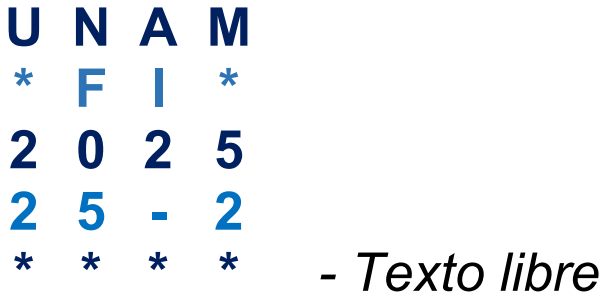
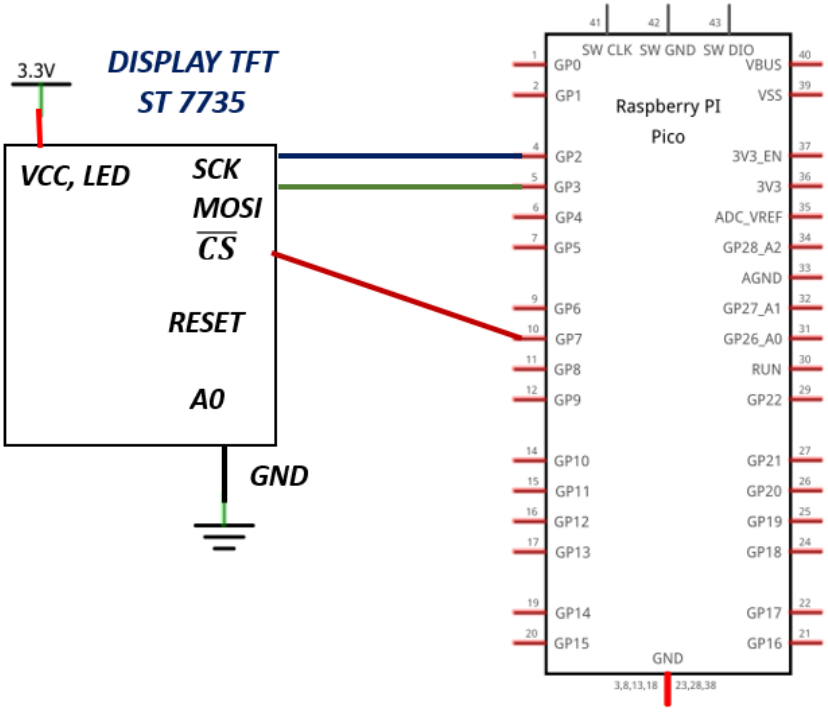


Figura 8-6. Actividad 4

Actividad 5. Escribir el siguiente programa, cerciorarse que el módulo Display TFT ST7735, se encuentre conectado en la posición correcta; investigar el funcionamiento de las librerías usadas; comentar el programa.



```
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin

spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

tft.rotation(2)

tft.fill(TFT.WHITE)
tft.text((10, 10), "MICROS", TFT.RED, sysfont, 2, nowrap=True)
tft.text((25, 30), "FI", TFT.GREEN, sysfont, 2, nowrap=True)
```

Figura 8-7. Código de ejemplo; actividad 5

Actividad 6. Realizar un programa que muestre los nombres de cada integrante de su equipo; elegir el formato que desee, así como el color para cada campo.

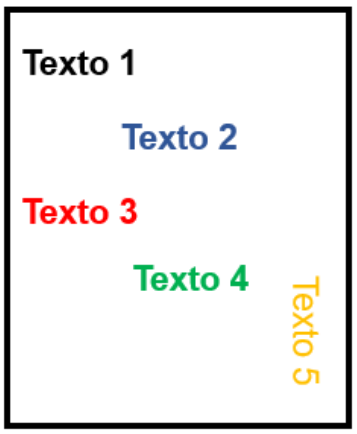
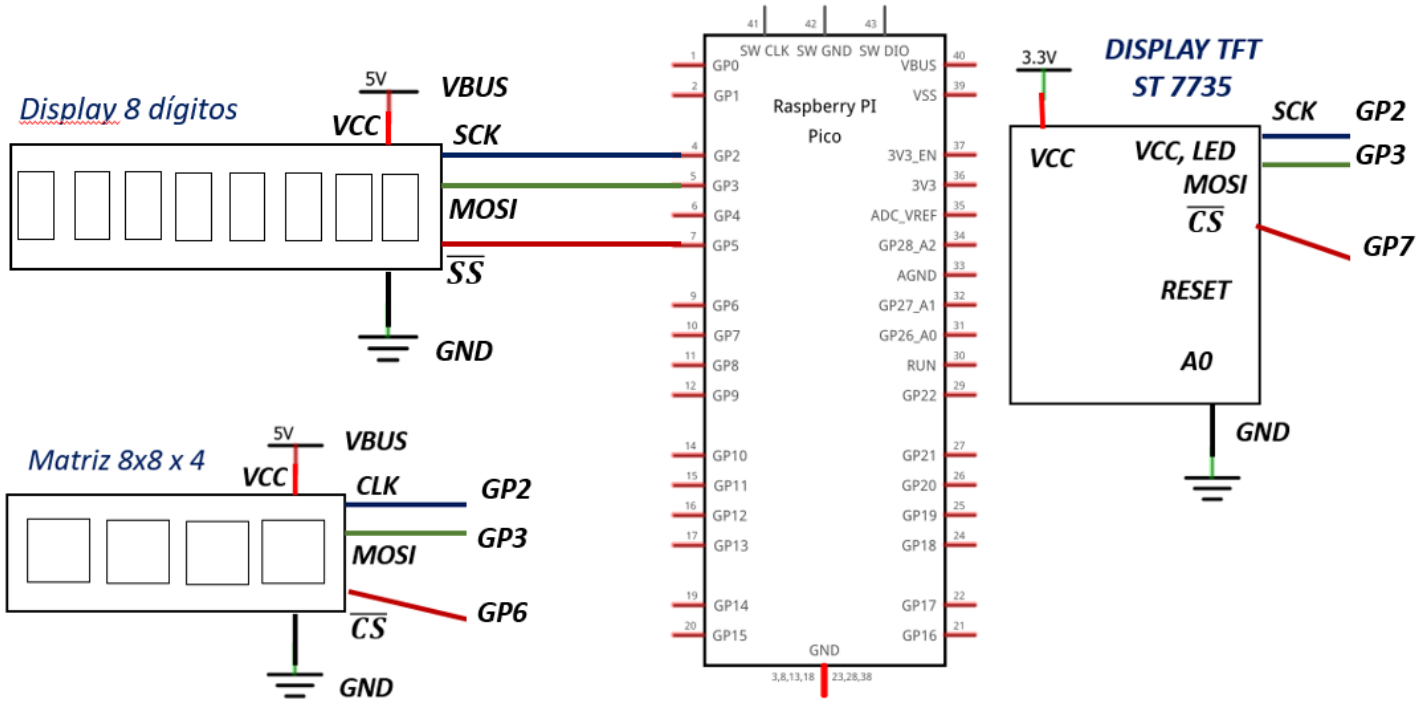


Figura 8-8. Despliegue propuesto; actividad 6

Actividad 7. Realizar un programa que muestre el valor del contador generado por dos entradas digitales, tal como se muestra en la tabla 8-3; el contador debe mostrarse en los tres dispositivos utilizados en esta práctica (Display TFT, Matriz 4x8x8 y el Display de 8 dígitos) al mismo tiempo.



Control	Estado	Acción
GPIO12 (Pull_Up)	Cerrado	Inicia/Reinicia cuenta
GPIO13 (Pull_Up)	Cerrado	Detiene cuenta

Display TFT
Matriz
Display de 8 dígitos

Tabla 8-3. Acciones de control; actividad 7

Laboratorio de Microcomputadoras
Practica No. 9
Comunicación One Wire

Objetivo. Conocer diversos protocolos de comunicación One Wire para el control de dispositivos a través de la plataforma Raspberry Pi Pico.

1 Introducción

Existe una gran variedad de dispositivos que transmiten o reciben información usando protocolos específicos, por lo regular emplean una línea de datos, en ocasiones requieren la señal de reloj, algunos módulos de uso común serán empleados en esta práctica; entre los que destacan:

a. Módulo de leds RGB

Los Neopixel son leds de tipo RGB que se controlan de forma serial; es posible emplear módulos de un solo led, 8, 16 más, pueden tener distribución lineal, circular o matricial. Los leds pueden ser activados de manera individual, el módulo contiene un chip que se encarga de las acciones de control.

Además de Vcc y GND tiene las señales DIN y DOUT que permite la conexión en cascada.

Figura 9-1. Módulos Neopixel

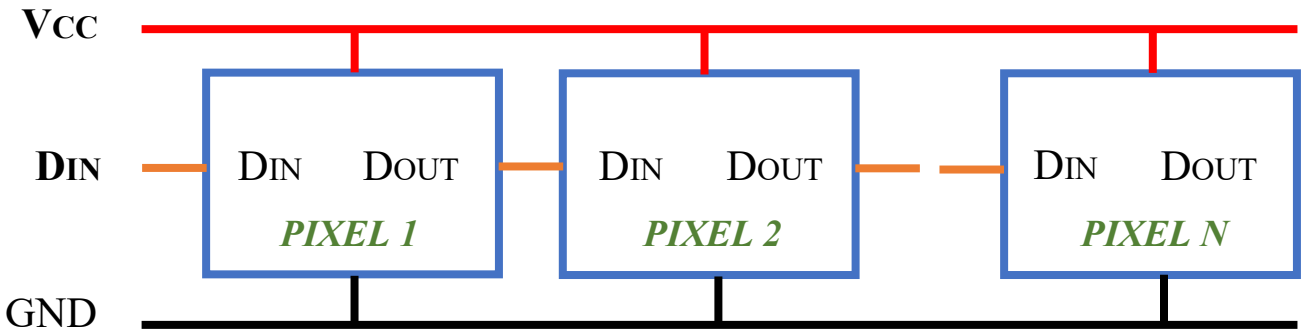
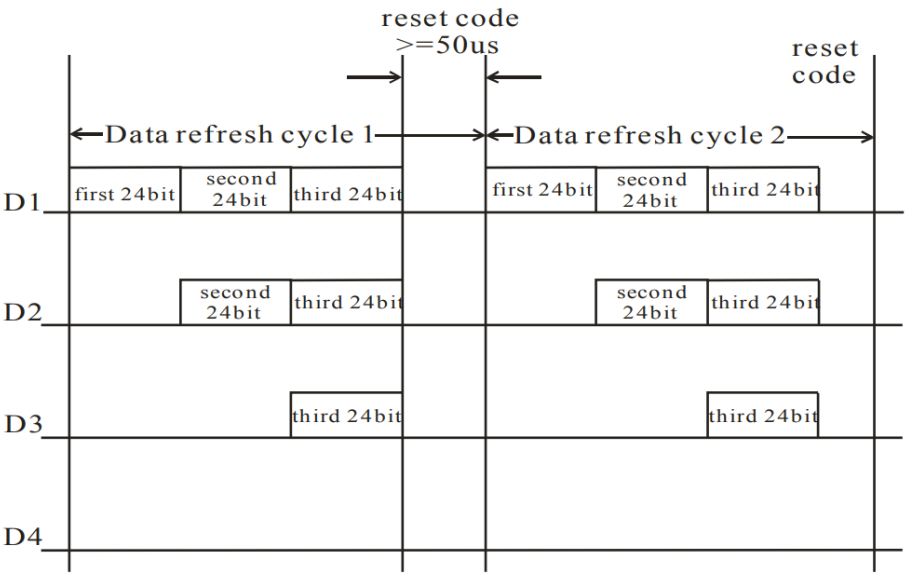


Figura 9-2. Conexión en cascada



G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Figura 9-3. Protocolo control Neopixel

b. Módulo de 4 dígitos TM1637

El TM1637 es un módulo de 4 display de 7 segmentos que utiliza comunicación serie para ser controlados. La activación la realiza con dos líneas: una de datos DIO y otra de reloj CLK; este tipo de interfaz se conoce como Two-wire. Se rige bajo un protocolo parecido al I2C, solo que no tiene dirección asociada.

CLK Señal de reloj
DIO Entrada y salida de datos
GND GND
VCC 3.3V – 5V

Figura 9-4. Display de 4 dígitos TM1637

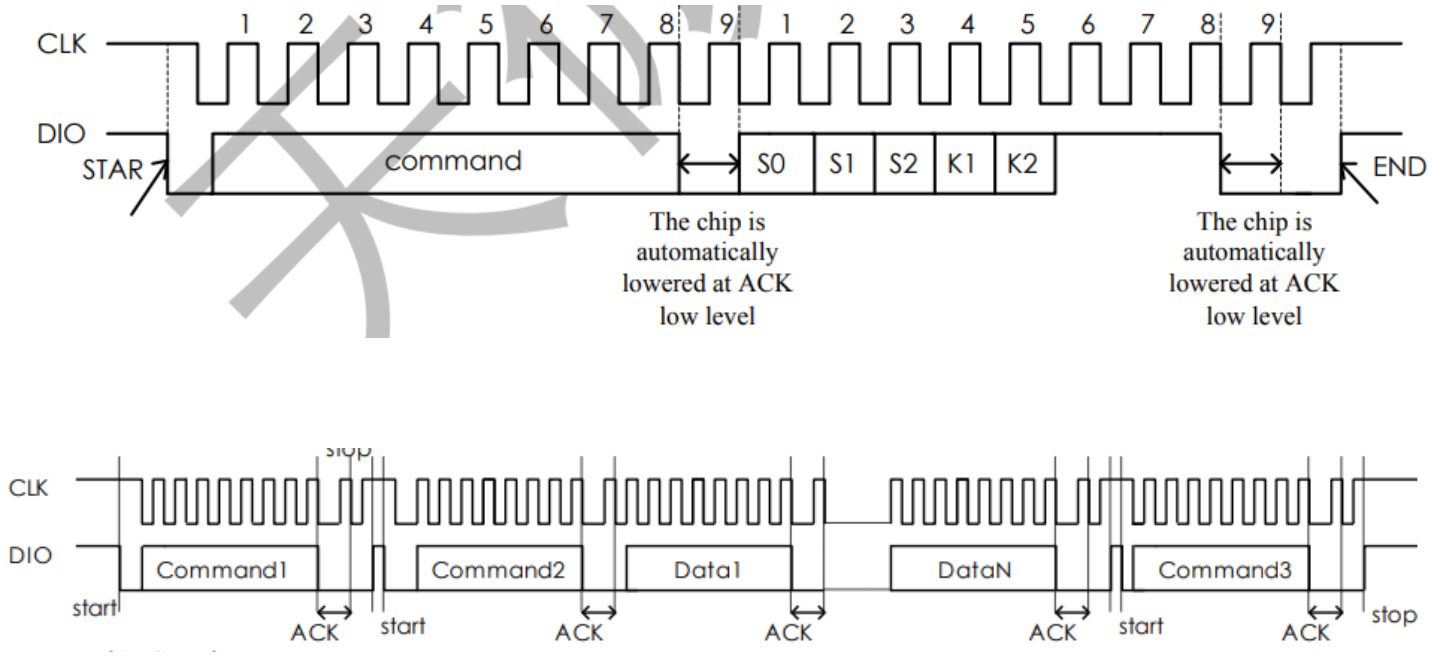


Figura 9-5. Protocolo del Display TM1637

c. Sensor digital de temperatura DS18B20

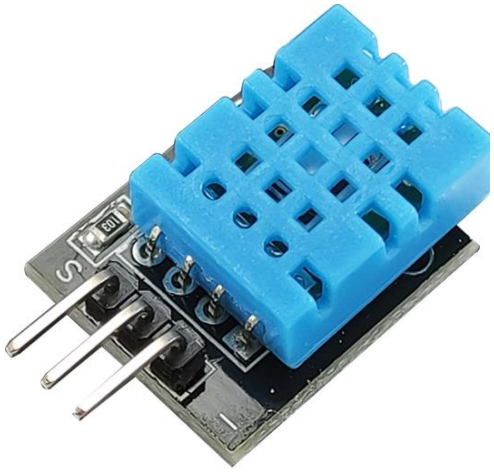
Es un sensor de temperatura que utiliza el protocolo 1-wire para recibir comandos y atender peticiones; emplea un bit de datos DQ, es posible conectar más de un sensor por el mismo bus, es capaz de tomar temperaturas entre los rangos de -55 °C hasta 125 °C, con resolución hasta 12 bits.

Este tipo de sensores cuenta con diferentes versiones, cada uno de los sensores tendrá una dirección única de 64 bits que se ha establecido desde el fabricante; será de gran utilidad para identificar el dispositivo.

Figura 9-6. Versiones disponibles, sensor de temperatura DS18B20

d. Sensor DTH11

El DTH11 es un sensor de temperatura y humedad emplea un protocolode tipo Single Wire, este comienza cuando el microcontrolador envía una señal de inicio (start), el DHT11 se activará y transmitirá 40 bits, estos corresponden a 16 bits de humedad (RH), 16 bits de temperatura (T), seguidos de 8 bits de cheksum, una vez enviado, pasará al estado de reposo hasta recibir otra petición.



Terminales:

S	*Señal de salida*
VCC	*3.3 – 5.0 Volts*
-	*GND*

Figura 9-7. Módulo DHT11

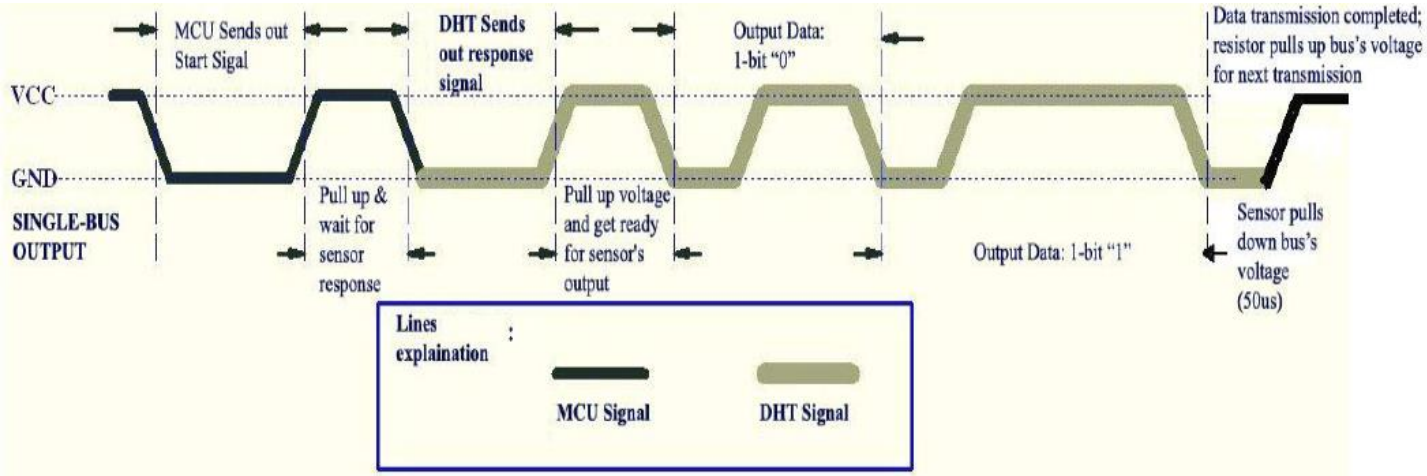


Figura 9-8. Protocolo DHT11; imagen tomada de DHT11 Technical Data Sheet

2 Requerimientos

2.1 Software.

- *Micropython*

2.2. Editor.

- *Thonny*

2.3 Hardware.

- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 2 para Raspberry Pi Pico*
- *Módulo leds RGB Neopixel (8)*
- *Display TM1637*
- *Sensor de temperatura DS18B20*
- *Sensor de temperatura y humedad DHT11*

a. Barra de leds RGB Neopixel

b. Display 4 dígitos TM1637

c. Sensor de temperatura y humedad DTH11

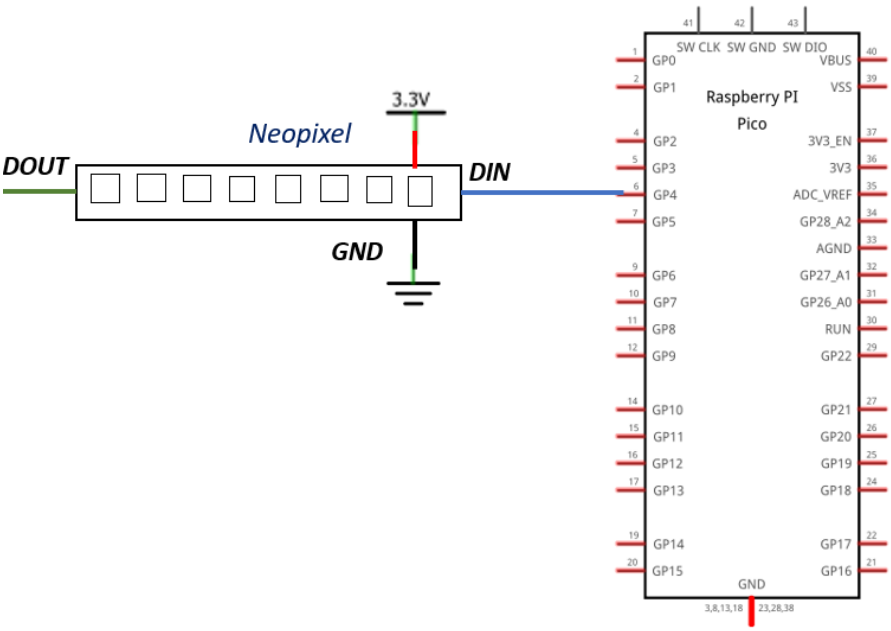
d. Sensor de temperatura DS18B20

Figura 9-9. Módulos One/Two – Wire

3 Desarrollo

Realizar las actividades solicitadas, investigar el funcionamiento de los dispositivos empleados; estudiar las librerías importadas.

Actividad 1. Escribir, comentar e indicar el funcionamiento del siguiente programa; estudiar la librería usada.




```
import time
from neopixel import Neopixel

pixels = Neopixel(8,0,4,"GRB")
brightness = 0.1

red=(255,0,0)
black=(0,0,0)

while True:
    pixels.set_pixel(0,red)
    pixels.show()
    time.sleep(1)
    pixels.set_pixel(0,black)
    pixels.show()
    time.sleep(1)
```

Figura 9-10. Código de ejemplo; actividad 1

Actividad 2. Utilizando la tira de 8 leds RGB; generar el efecto mostrado en la figura 9-11, en forma secuencial con tres distintos colores.

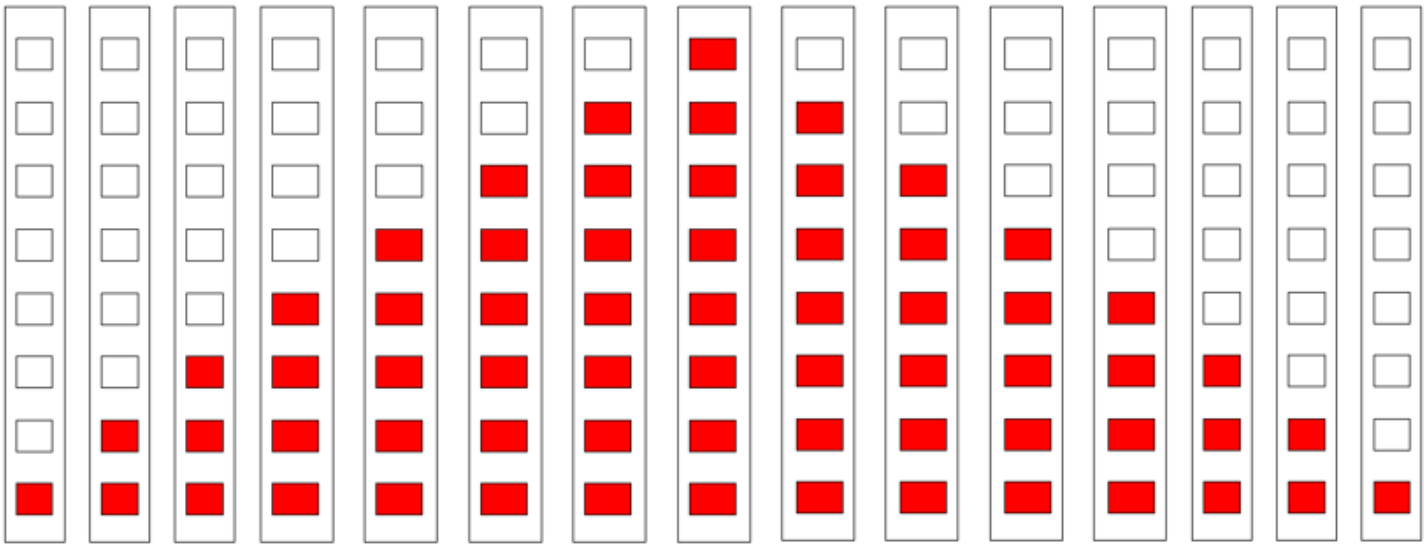
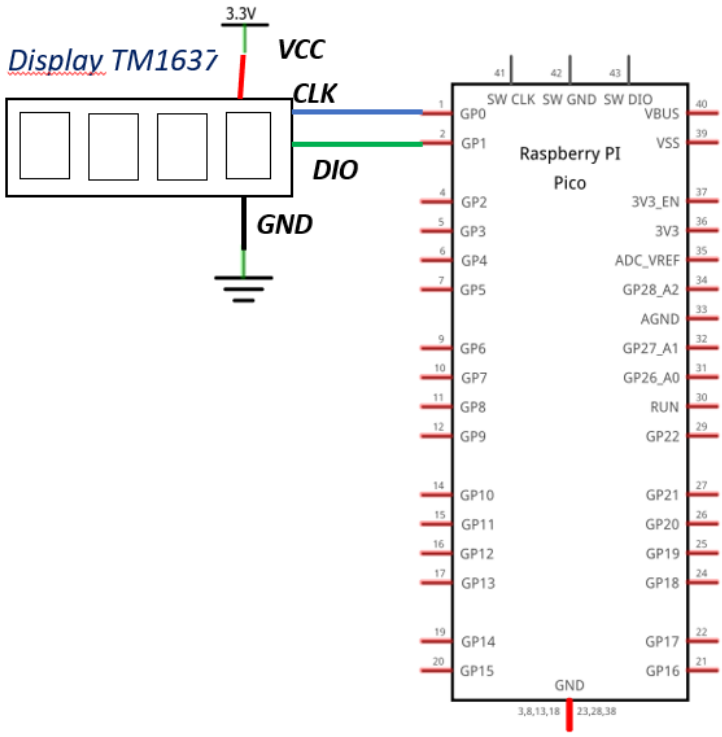


Figura 9-11. Muestra de secuencia a generar; actividad 2

Actividad 3. Escribir, comentar e indicar el funcionamiento del siguiente programa; estudiar la librería usada.



```
import tm1637
from machine import Pin
from utime import sleep
tm=tm1637.TM1637(clk=Pin(0),dio=Pin(1))
Sec=00
Min=00
while True:
    tm.numbers(Min,Sec,colon=True)
    sleep(0.5)
    tm.numbers(Min,Sec,colon=False)
    sleep(0.5)
    Sec=Sec+1
    if Sec==60:
        Min=Min+1
        Sec=0
        if Min==60:
            Min=0
```

Figura 9-12. Código de ejemplo y esquemático; actividad 3

Actividad 4. Empleando el display TM1637; realizar un programa que muestre un contador descendente, iniciando en 20 y cuando corresponda el valor 0, active un sonido de 1 segundo de duración en el zumbador de la tarjeta (GPIO17).

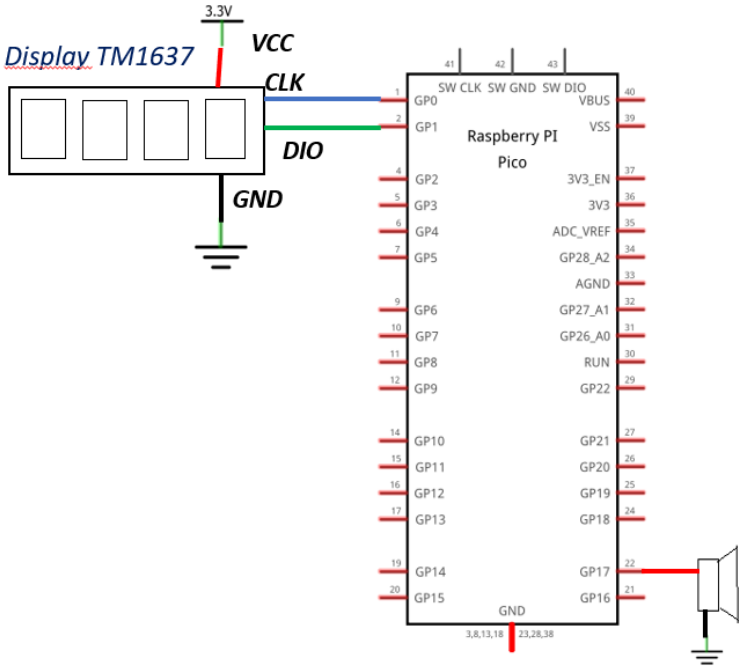
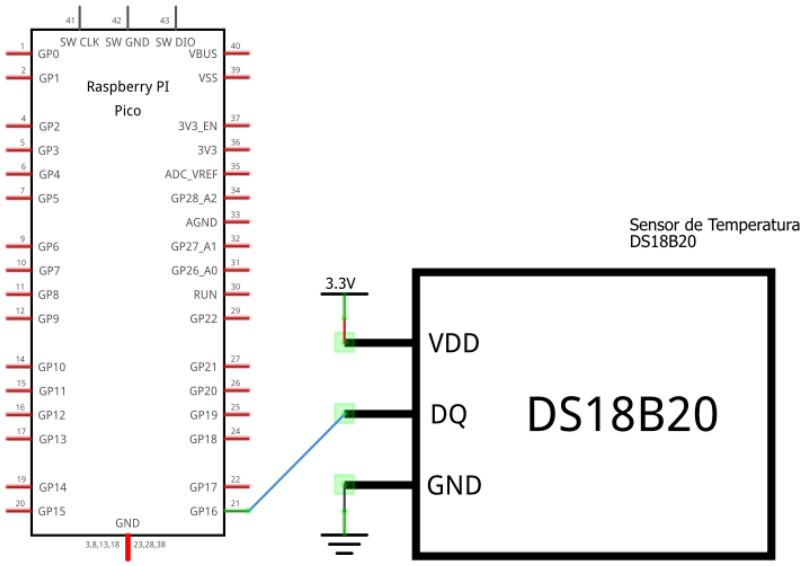


Figura 9-13. Esquemático de la actividad 4

Actividad 5. Escribir, comentar e indicar el funcionamiento del siguiente programa.




```
import machine, onewire, ds18x20, time

ds_pin = machine.Pin(16)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))

roms = ds_sensor.scan()
print("Sensor detectado", roms)

while True:
  ds_sensor.convert_temp()
  time.sleep_ms(750)
  for rom in roms:
    print(rom)
    tempC = ds_sensor.read_temp(rom)
    print('temperatura (ºC):', "{:.2f}".format(tempC))
    print()
  time.sleep(2)
```

Figura 9-14. Código de ejemplo y esquemático; actividad 4

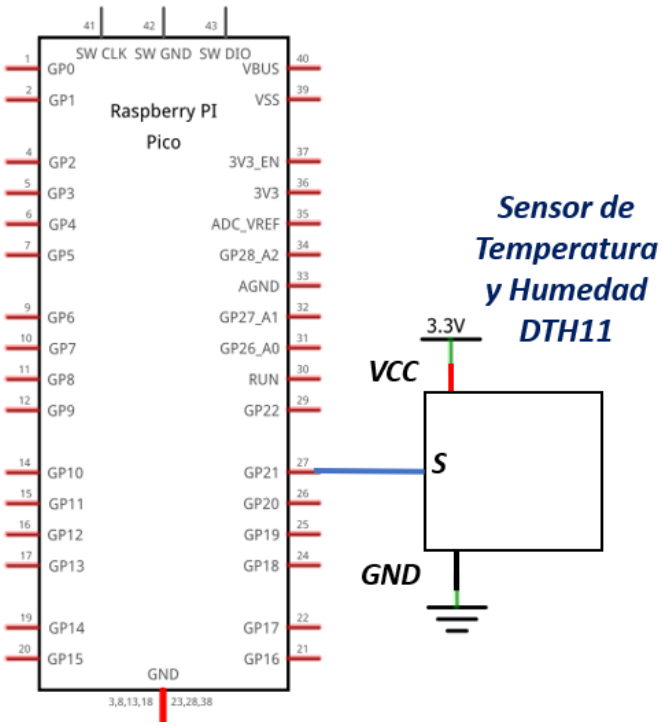
Actividad 6. Realizar las modificaciones necesarias al programa anterior, para mostrar también la temperatura en ºF.

```
bytearray(b'(\x0f\x95D\xd4\xe1<\x9a')
temperature (°C): 23.56
temperature (°F): 74.41

bytearray(b'(\x0f\x95D\xd4\xe1<\x9a')
temperature (°C): 23.56
temperature (°F): 74.41
```

Figura 9-15. Temperatura con DS18B20 en ºC y ºF

Nota. El identificador del sensor puede variar.

Actividad 7. Escribir, comentar e indicar el funcionamiento del siguiente programa; estudiar la librería usada.



```
import dht
from machine import Pin
import time

Sensor = dht.DHT11(Pin(21))

while True:

    Sensor.measure()
    temp = Sensor.temperature()
    hum = Sensor.humidity()
    time.sleep(2)

    print(f"Temperatura: {temp}°C Humedad: {hum}%")
```

Figura 9-16. Código de ejemplo y esquemático; actividad 5

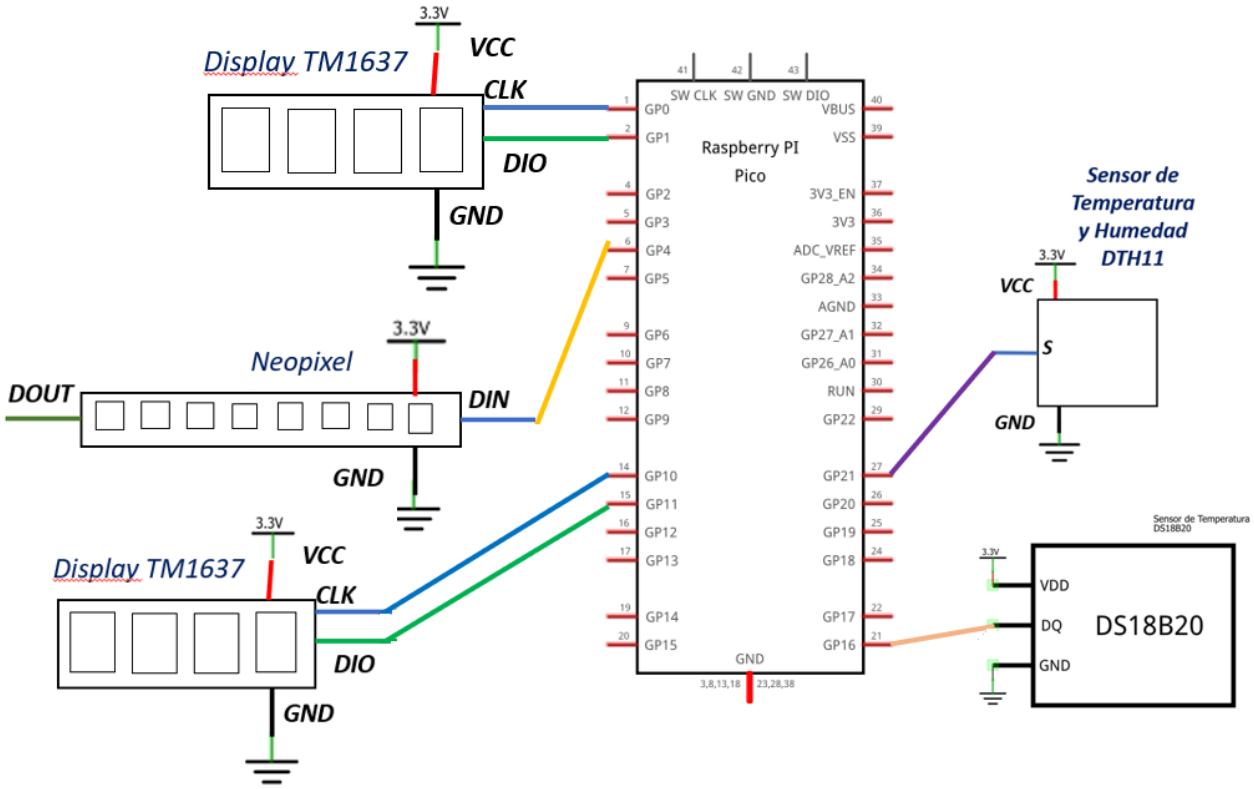
Nota. Registrar la temperatura leída.

Actividad 8. Utilizando el sensor DTH11, modificar el programa anterior, para que indique el momento en que se ha incrementado la temperatura de referencia, en la cercanía del sensor en razón de 5 ºC (*temperatura tomada de la actividad anterior*), como se indica en la tabla 9-1.

Temperatura DTH11	LED (GPIO20)	Zumbador (GPIO17)
Temp _actual ≤ ***Temp_ref*** **+ 5ºC**	0	0
Temp _actual > ***Temp_ref*** **+ 5ºC**	1	1

Tabla 9-1. Control; actividad 8

Actividad 9. Empleando los cuatro módulos usados en esta práctica, realizar un programa que controle el funcionamiento de dos semáforos con el comportamiento descrito en la *actividad 10 de la practica 3*, al cabo de cada secuencia completa, mostrar el valor de la temperatura de cada uno de los sensores en un display TM1637 distinto; tal como se muestra en la tabla.



Semáforo			Display TM1637		Display TM1637	
V1	A1	R1	CLK	DIO	CLK	DIO
V2	A2	R2	GPIO0	GPIO1	GPIO10	GPIO11

Barra leds Neopixel	Temperatura DS18B20	Temperatura DHT11
GPIO4	GPIO16	GPIO21

Figura 9-16. Especificaciones y esquemático; actividad 9

Laboratorio de Microcomputadoras
Practica No. 10
Comunicación Serie Síncrona I2C

Objetivo. Aprender la teoría y aplicación del protocolo I2C, controlar módulos de diversos tipos con comunicación I2C.

1. Introducción

El bus I2C o I^2C *(Inter-Integrated Circuit*) fué desarrollado por Philips, su propósito original fue conectar un controlador de un televisor a otros periféricos. Se ha adoptado como protocolo de comunicación serial, permite transferencia de datos entre un microcontrolador y dispositivos externos. La tasa de transferencia estándar es de 100Kbps y la más alta puede alcanzar los 3.4 Mbps.

El protocolo I2C requiere de solo dos líneas:

I2C		
SDA	**Serial Data Line**	**Línea de datos bidireccional**.
SCL	Serial Clock Line	Señal de reloj o línea de sincronización.

Los dispositivos conectados a estas líneas son de tipo "*colector abierto*", por lo tanto debe de conectar resistencias de *pull-up* de 10 KΩ, así como tener tierras comunes para establecer las mismas referencias entre todos los dispositivos.

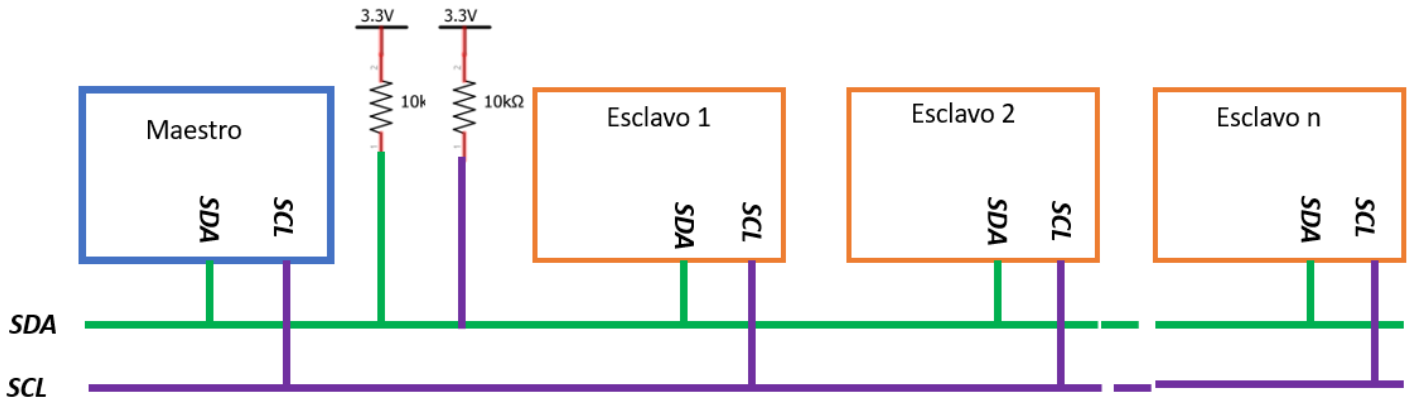


Figura 10-1. Comunicación I2C

Se puede establecer comunicación entre un maestro y un esclavo o varios esclavos, así como de tipo multi-maestro. Cada uno de los dispositivos tiene una dirección única, en general el protocolo consta de las siguientes etapas:

1. *Señal de inicio **START***
2. *Selección del dispositivo esclavo*
3. *Indica acción a realizar lectura o escritura*
4. *Respuesta del esclavo*
5. *Transferencia de datos*
6. *Señal de paro **STOP***

a. Transferencia de datos del maestro al esclavo

La transferencia de datos al esclavo se describe en la figura 10.2

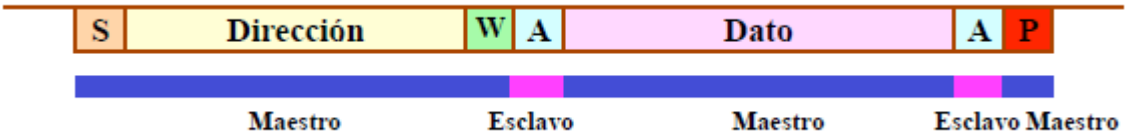


Figura 10-2. Protocolo I2C Maestro-Esclavo

El algoritmo a para transferir información del maestro al esclavo:

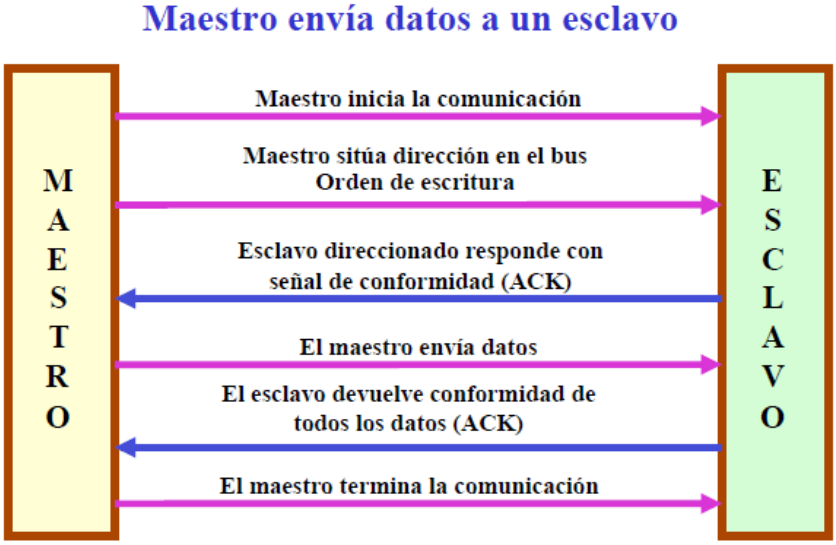


Figura 10-3. Algoritmo I2C.

En caso de transferir mayor información del maestro al esclavo, entonces el protocolo a seguir será:

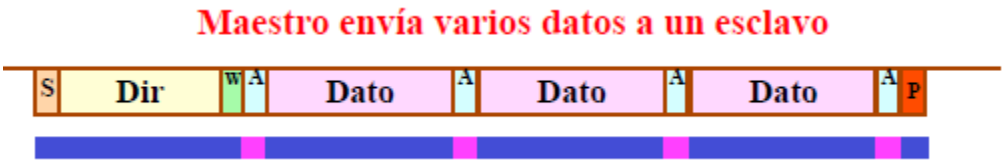


Figura 10-4. Protocolo I2C Maestro – Esclavo; transferencia de varios datos

b. Lectura de datos del esclavo

El protocolo para esta actividad se describe en las figuras 10-5 y 10-6.

Figura 10-5. Solicitud de datos del maestro

En situación de peticiones del maestro a un esclavo.

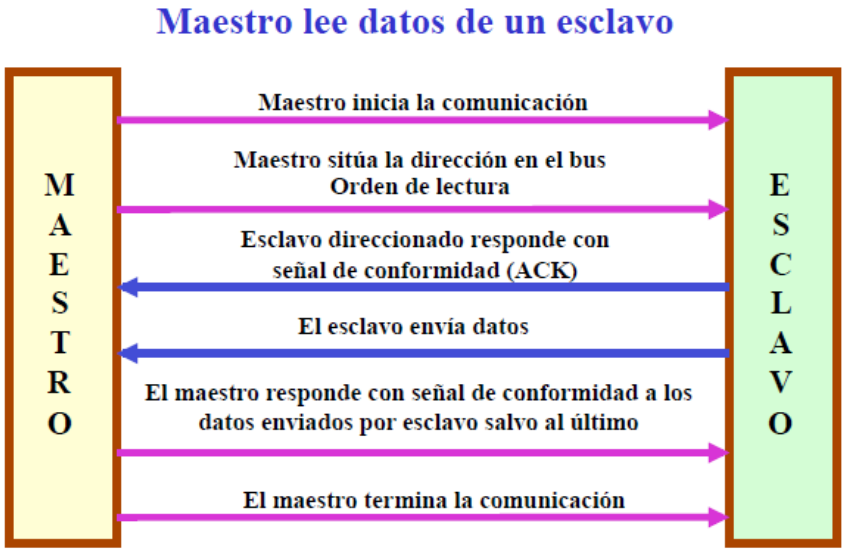


Figura 10-6. Algoritmo para lectura de datos del maestro

c. I2C en Raspberry Pi Pico

Tiene dos puertos serie asíncrono I2C, denominados I2C0 e I2C1; asignados como función alterna de los GPIO:

I2C0						
I2C0 SDA	GPIO0	GPIO4	GPIO8	GPIO12	GPIO16	GPIO20
I2C0 SCL	GPIO1	GPIO5	GPIO9	GPIO13	GPIO17	GPIO21

I2C1						
I2C1 SDA	GPIO2	GPIO6	GPIO10	GPIO14	GPIO18	GPIO26
I2C1 SCL	GPIO3	GPIO7	GPIO11	GPIO15	GPIO19	GPIO27

A continuación se presentan los métodos y funciones disponibles para emplear el I2C; es altamente recomendable consultar la documentación oficial de Micropython para obtener mayor información.

La comunidad alrededor de Micropython ha aportado librerías de dispositivos I2C, lo que ha facilitado la implementación de aplicaciones con estos; también se sugiere estudiarlas a detalle.

from machine import I2C, Pin	#	*Importar la clase UART del módulo machine*
i2c = I2C (Num_i2c, scl=Pin(GP_scl), sda=Pin(GP_sda), freq = frecuencia)		
	#	*Creación de la instancia I2C* *i2c – Nombre del objeto* *Num_i2c – es el número de I2C a usar 0/1* *GP_scl – Selección del pin GPIO como señal de reloj* *GP_sda – Selección del pin GPIO como señal de datos* *freq – Configuración de frecuencia*
i2c.scan()	#	*Busca los dispositivos conectados al bus I2C*
i2c.start()	#	*Inicia protocolo de comunicación*
i2c.stop()	#	*Termina comunicación*
i2c.writeto(dir, datos)	#	*Transmisión de datos al dispositivo externo indicado*
I2c. readfrom(dir,datos)	#	*Lectura de datos del dispositivo externo indicado*
i2c.readfrom_mem(dir, mem,dat)	#	*Lectura de datos de la dirección de memoria del dispositivo externo*
i2c.writeto_mem(dir, mem, dat)	#	*Escritura de datos a la dirección de memoria del dispositivo externo*

En esta práctica se han conectado los distintos módulos al I2C0, en las terminales GPIO8 y GPIO9.

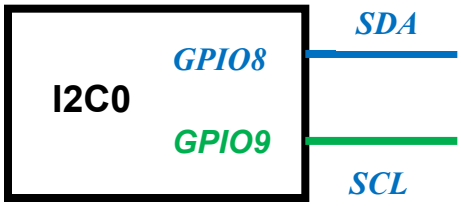


Figura 10-7. Asignación de terminales I2C

2 Requerimientos

2.1 Software.

- *Micropython*

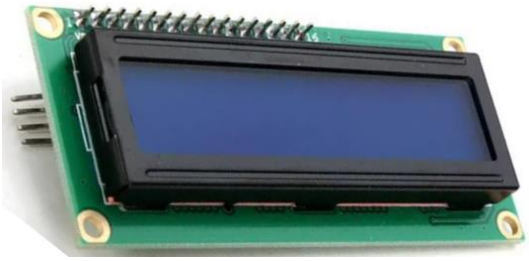
2.2. Editor.

- *Thonny*

2.3. Hardware.

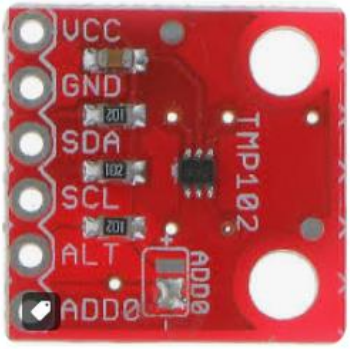
- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 2 para Raspberry Pi Pico*
- *CI I2C PCF8574*
- *LCD I2C 16x2*
- *Display Oled*
- *Sensor de temperatura TMP102*
- *Sensor de temperatura y humedad AHT10*
- *Módulo inercial IMU MPU6050*

a. PCF8574

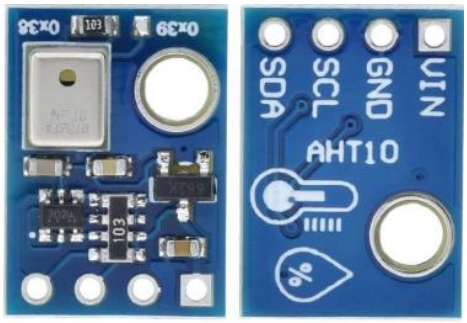


b. LCD 16x2 I2C

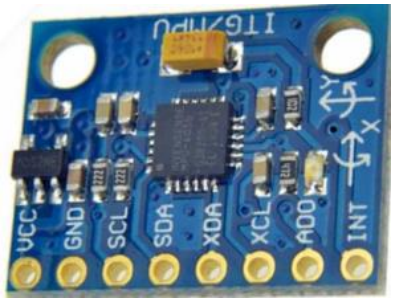
c. Display OLED SSD1306



d. Sensor de temperatura TMP102



e. Sensor de temperatura y humedad AHT10



f. Acelerómetro – giroscopio MPU6050

Figura 10-8. Módulos I2C; practica 10

3 Desarrollo

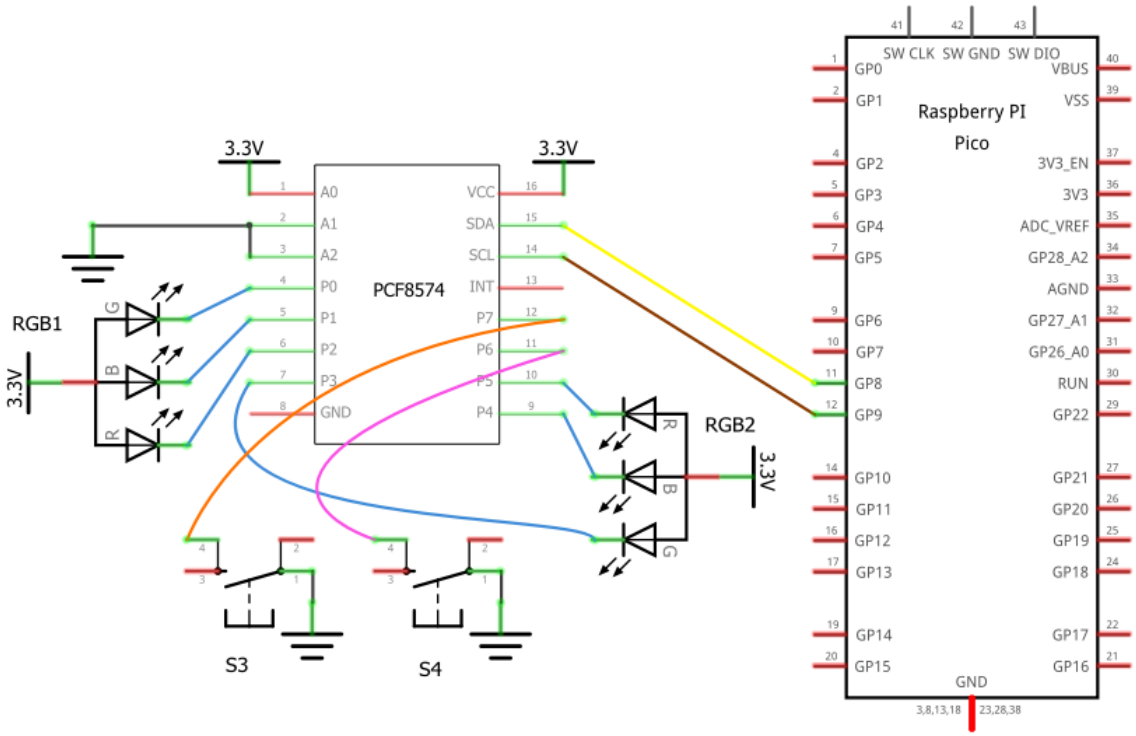
Realizar las actividades solicitadas; considerar los módulos ocupados en cada una de ellos, consultar la documentación oficial para obtener mayor información del funcionamiento de estos, analizar las librerías empleadas.

Actividad 1. Escribir el siguiente código, comentar; reportar que hace.

```
from machine import Pin, I2C

sda = Pin(8)
scl = Pin(9)
i2c = I2C(0, scl=scl, sda=sda)

devices = i2c.scan()
if devices:
    for d in devices:
        print(hex(d))
```

Figura 10-9. Código de prueba; actividad 1

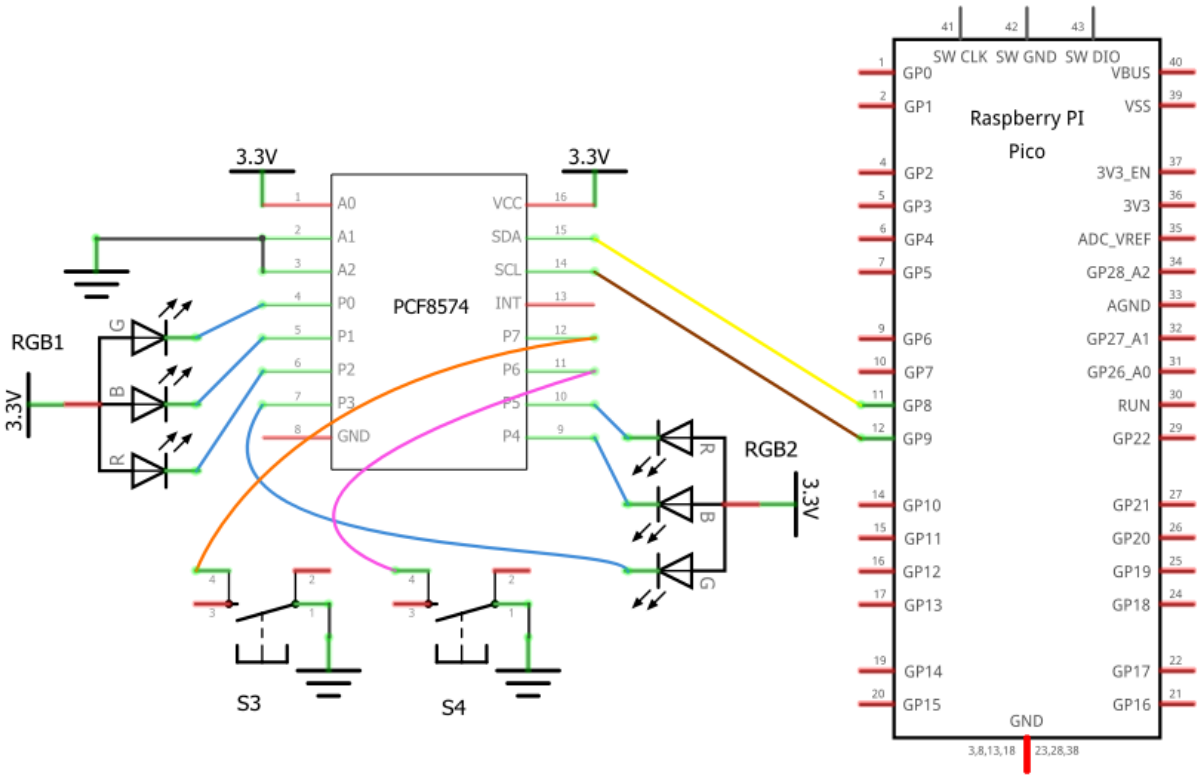
Actividad 2. Escribir, comentar e indicar el funcionamiento del siguiente código, será empleada la comunicación serie I2C usando el circuito PCF8574; estudiar la librería empleada.




```
import pcf8574
from machine import I2C, Pin
import time

i2c = I2C (0,scl=Pin(9), sda=Pin(8))
pcf = pcf8574.PCF8574(i2c, 0x39)

while True:
    pcf.port = 0x3F
    print(pcf.port)
    time.sleep(0.8)

    pcf.port = 0x3E
    print(pcf.port)
    time.sleep(0.8)
```

Figura 10-10. Código de ejemplo; actividad 2

Actividad 3. Utilizando el circuito anterior, generar la secuencia mostrada en la tabla 10-1; considerar comunicación I2C.



P5	P4	P3	P2	P1	P0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0

0 – OFF
1 - ON

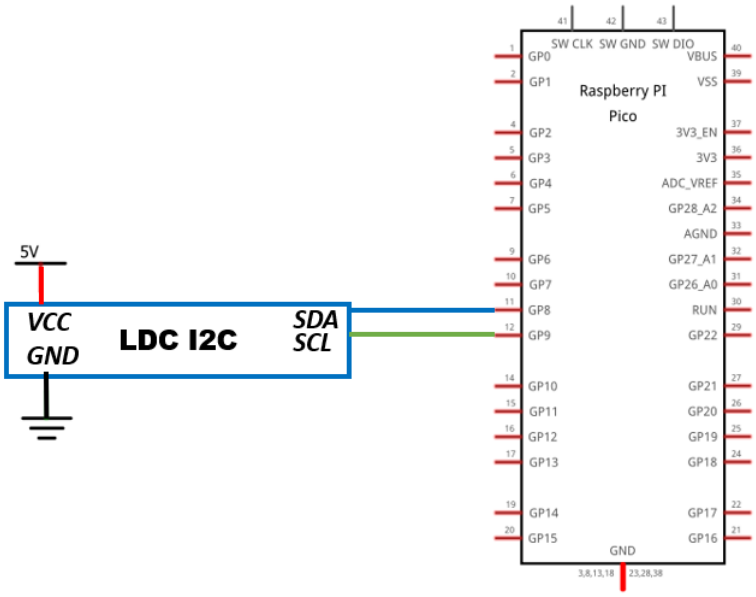
Tabla 10-1. Control; actividad 3

Actividad 4. Escribir el siguiente código; comentar y explicar su funcionamiento; se utilizará circuito de la actividad previa.

```
import pcf8574
from machine import I2C, Pin
import time

i2c = I2C (0,scl=Pin(9), sda=Pin(8))
pcf = pcf8574.PCF8574(i2c, 0x39)
ON = 0
OFF = 1
pcf.port = 0x3F

pcf.pin (6, 1)  #Entrada P6
print("iniciamos")
while True:
    if (pcf.pin(6)==0):
        pcf.pin(0,ON)
        print("Alto")
        pcf.pin (6, 1)
    else:
        pcf.pin(0,OFF)
        print("bajo")
        pcf.pin (6, 1)
```

Figura 10-11. Código; actividad 4

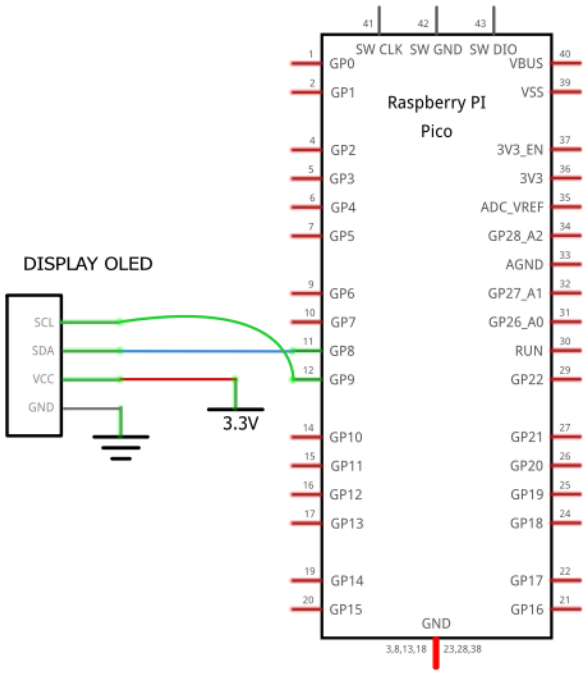
Actividad 5. Escribir el siguiente código, comentar e indicar que hace, revisar las librerías empleadas.




```
from machine import I2C, Pin
import time
from esp8266_i2c_lcd import I2cLcd

DEFAULT_I2C_ADDR = 0x27
i2c = I2C(0,scl=Pin(9), sda=Pin(8), freq=200000)
lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)

lcd.putstr("UNAM!\nFI")
time.sleep(3)
lcd.clear()

lcd.move_to(3, 0)
lcd.putstr("Laboratorio")
lcd.move_to(0, 1)
lcd.putstr("* M I C R O S *")
time.sleep(1)
```

Figura 10-12. Código ejemplo; LCD I2C

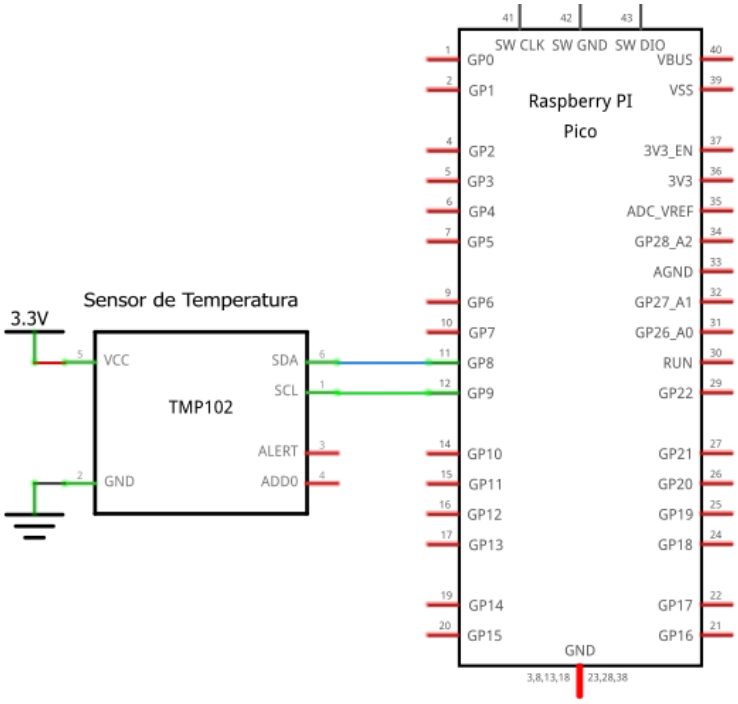
Actividad 6. Escribir el siguiente código, comentar e indicar que hace, revisar la librería utilizada.




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

i2c = I2C(0, scl=Pin(9),sda=Pin(8),freq=400000)
oled = SSD1306_I2C(128,64,i2c)

devices = i2c.scan()
if devices:
    for d in devices:
        print("I2C Adress: " + hex(d))

oled.fill(0)
oled.text("Microcomputadoras",1,6,1)
oled.text("Practica I2C",3,30,1)
oled.show()
print("UNAM FI")
```

Figura 10-13. Código de prueba; actividad 6

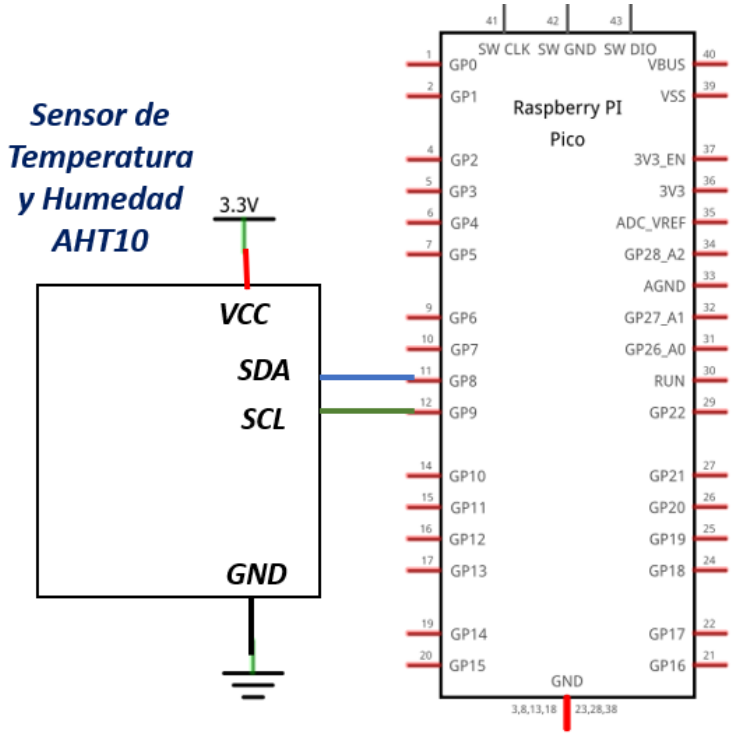
Actividad 7. Escribir, comentar e indicar que hace el siguiente código, este usa la comunicación I2C a través del módulo TMP102 (sensor de temperatura).




```
from machine import Pin, I2C
import time

i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=100000)
addr = i2c.scan()
print( "address is :" + str(addr) )

for i in range(100):
        data = []
        data = i2c.readfrom(0x48, 2)
        intdata = int.from_bytes(data, 'big')
        tmp = intdata >> 4
        print(tmp * 0.0625)
        time.sleep(1)
```

Figura 10-14. Código de ejemplo; actividad 7

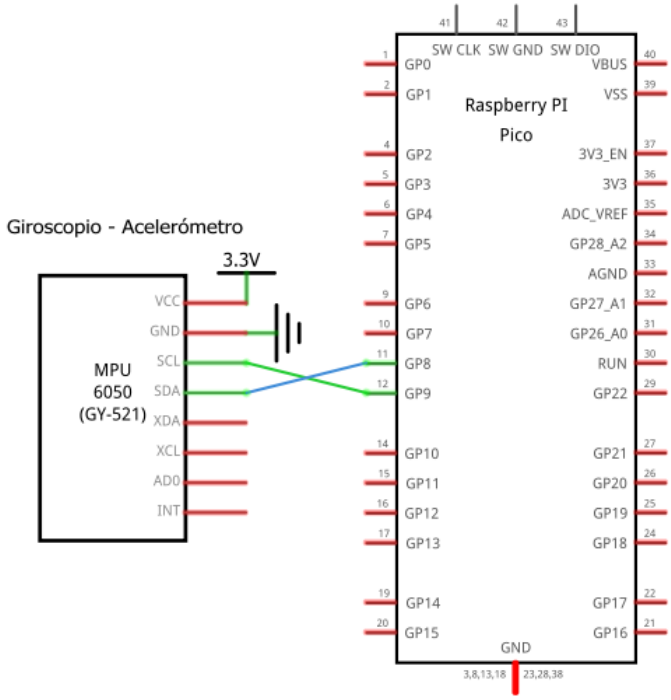
Actividad 8. Escribir, comentar e indicar que hace el siguiente código, este usa la comunicación I2C a través del módulo AHT10 (sensor de temperatura y humedad); estudiar la librería empleada.




```
import utime
from machine import Pin, I2C
import ahtx0

i2c =I2C(0,sda=Pin(8),scl=Pin(9),freq=400000)
sensor = ahtx0.AHT10(i2c)

while True:
    print("\nTemperature: %0.2f C" % sensor.temperature)
    print("Humidity: %0.2f %%" % sensor.relative_humidity)
    utime.sleep(5)
```

Figura 10-15. Código de ejemplo, actividad 8

Actividad 9. MPU6050 Acelerómetro giroscopio.




```
from imu import MPU6050
from machine import Pin,I2C
import utime

i2c =I2C(0,sda=Pin(8),scl=Pin(9),freq=400000)
imu = MPU6050(i2c)

while True:
    ax =round(imu.accel.x,2)
    print(ax)
    utime.sleep_ms(200)
```

Figura 10-16. Programa de ejemplo; actividad 9

Actividad 10. Realizar un programa, que genere las acciones mostradas en la tabla 10-2.

Condición	Acción	
$-45° \leq ax \leq 45°$	LED (*GPIO18*) = ***OFF***	BUZZER (*GPIO17*) = ***OFF***
$-46° \geq ax \geq 46°$	LED (*GPIO18*) = ***ON***	BUZZER (*GPIO17*) = ***ON***

Tabla 10-2. Control; actividad 10

Laboratorio de Microcomputadoras
Practica No. 11
Programación de Interrupciones y Threads

Objetivo. Entender y aplicar la programación mediante interrupciones; así como la realización de programas en los que se utilicen diferentes núcleos, optimizar la ejecución de programación en paralelo.

1 Introducción

El microcontrolador RP2040 permite la ejecución de programas en forma paralela, mediante el uso de interrupciones (***isr)*** y programación de hilos (***threads***); cada una de ellas con características especiales.

I. Interrupciones

Una interrupción es la solicitud al procesador para suspender la ejecución del programa, por lo que se procederá a atender esta petición, se almacenará en la pila (*stack*) el entorno del procesador, que consta de la dirección de la instrucción que continuaba en la cola de ejecución y el contenido de los registros internos, así como del registro de banderas; el control del programa pasará a través del vector de interrupción a la rutina o función de interrupción.

Una vez que interrupción ha sido atendida el control regresará a las condiciones previas a la petición de interrupción.

Para que un procesador atienda una petición de interrupción se requiere:

1. *Habilitar interrupciones particulares.*
2. *Habilitar interrupciones generales.*
3. *Configurar el vector de interrupción (aplica en programación en ensamblador).*
4. *Definir la rutina de interrupción.*

En Micropython se proporcionan métodos encargados de configurar y atender las interrupciones.

from machine import Pin	#	*Importar la clase Pin para uso de interrupciones digitales*
fuente=machine.Pin(Num_GPIO, machine.Pin.IN, machine. Pin.PULL_UP |Pin. PULL_DOWN)		
	#	*Declaración de las características del objeto fuente*
fuente.**irq=(handler =** *Nombre_Funcion_isr*, **trigger =** *Evento_que_dispara_la_peticion_de_isr***)**		
	#	*Evento_que_dispara_la_petición_de_isr* *Pin.IRQ_FALLING – Interrupción con flancos de bajada* *Pin.IRQ_RISING – Interrupción con flancos de subida* *Pin.IRQ_LOW_LEVEL – Interrupción en nivel bajo '0'* *Pin.IRQ_HIGH_LEVEL – Interrupción en nivel alto '1'*
machine.disable_irq()	#	*Deshabilita interrupción*
machine.enable_irq(state)	#	*Habilita interrupción*
def función_isr(pin): **global var**	#	*Define la función de interrupción de tipo pin; de nombre función_isr; la(s) variables deben ser globales.*

Un programa que utilice control con interrupciones digitales, debe tener las configuraciones descritas previamente, identificarlas en el programa de la figura 11-1.

```
import machine

SW1 = machine.Pin (Num_GPIO,machine.Pin.IN, machine.Pin.PULL_DOWN) #Define instancia SW1

def funcion_isr (pin): # Función de Interrupción
    global VARIABLE    #Declarar variable de control de tipo GLOBAL


SW1.irq (trigger = machine.Pin.IRQ_RISING, handler = funcion_isr) #Configura interrupción

while True:
            # Programa principal
```

Figura 11-1. Plantilla de programa con interrupciones en Micropython

II. Programación por Hilos

Raspberry Pi Pico dispone de dos núcleos (*Cores*), regularmente todas las aplicaciones son ejecutadas en un solo núcleo; habilitar los dos núcleos nos permitirá ejecutar dos procesos al mismo tiempo ejecutándose en forma paralela, a este tipo de procesamiento se le conoce como programación por hilos (thread).

En Micropython se requiere agregar la librería para manejo de hilos, incluir la función que correrá en el segundo núcleo e invocar la ejecución en el hilo principal. En ocasiones se requiere sincronizar el funcionamiento de los dos núcleos por lo que se crea un control por semáforos. Los semáforos son objetos que permiten detener la ejecución del programa hasta que el semáforo es liberado, permitiendo un control en ejecuciones multihilo.

import _thread	#	*Importa la librería _thread para utilizar procesamiento de hilos*
_thread.start_new_thread*(nom_función_del_2º_hilo, (argumentos))*		
	#	*Nom_funcion_del_2º_hilo – Corresponde al nombre dado Argumentos – en caso de requerir si no iría vacio*
*sincro***=_thread.allocate_look()**	#	*Creación del semáforo de bloqueo para sincronizar*
*sincro***.acquire()**	#	*Adquisición del bloqueo del semáforo*
*sincro***.release()**	#	*Libera el bloqueo del semáforo.*

2 Requerimientos

2.1. Software.

- *Micropython*

2.2. Editor.

- *Thonny*

2.3. Hardware.

- *Raspberry Pi Pico o Raspberry Pi Pico W*
- *Plataforma embebida 2 para Raspberry Pi Pico*
- *Barra leds RGB Neopixel*
- *2 Displays TM1637*
- *2 Push button*
- *Leds*

3 Desarrollo

Realizar las actividades solicitadas; investigar el funcionamiento de las interrupciones y la programación de hilos, para detallar sus características de uso.

Actividad 1. Escribir el siguiente código, comentar, indicar que hace y explicar su funcionamiento.

```
from machine import Pin
import utime

S1 = Pin(12,Pin.IN, Pin.PULL_UP)

def FuncISR_S1(pin):
    print("Interrupción detectada")
    utime.sleep(1)

S1.irq(trigger =Pin.IRQ_FALLING, handler=FuncISR_S1)

print("! ... Esperando Interrupción !")

while True:
    pass
```

Figura 11-2. Código de ejemplo; actividad 1

Actividad 2. Realizar las modificaciones necesarias al programa de la actividad 1, de manera que, al reconocer la interrupción, provocada por un flanco de bajada ocurrida en S1 (GPIO12); el programa principal genere una señal cuadrada con frecuencia de 1 Hz cada que sea detectada y sea visible en la terminal GPIO20.

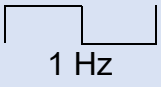
Evento	Acción (GPIO20)
S1 (GPIO12) 	

Tabla 11-1. Control; actividad 2

Actividad 3. Realizar un programa en el que se activen dos interrupciones, controladas por las entradas S1 y S2; en el momento de alguna de estas interrupciones sea requerida, genere las acciones solicitadas en la tabla 11-2.

Evento		Acciones	
S1 (GPIO12)	Pulso de bajada	LED ON (GPIO18)	TM1637 (GP0) Cuenta Interrupciones S1
S2 (GPIO13)	Pulso de bajada	LED ON (GPIO19)	TM1637 (GP10) Cuenta Interrupciones S2

Tabla 11-2. Control; actividad 3

Actividad 4. Escribir el siguiente código, comentar, indicar que hace y explicar su funcionamiento.

```
from machine import Pin
import utime
import _thread

led1 = Pin(18, Pin.OUT)
led2 = Pin(20, Pin.OUT)

def led2_thread():
    while True:
        print("Este es un mensaje del segundo nucleo")
        led2.toggle()
        utime.sleep(0.2)

_thread.start_new_thread(led2_thread, ())

while True:
    led1.toggle()
    utime.sleep(0.25)
```

Figura 11-3. Código de ejemplo; actividad 4

Actividad 5. Realizar un programa, que sea controlado con interrupciones, serán activadas cada que se detecten flancos de bajada en S1 y S2; cuando sean reconocidas generar las acciones solicitadas en la tabla 11-4; considerar que estas serán ejecutadas en diferentes núcleos.

Evento		Acción		
Interrupción	Hilo (Thread)	LED	Estado	Hilo (Thread)
S1	1	GPIO18	Toggle	1
S2	1	GPIO19	Toggle	2

Tabla 11-4. Acciones de control; actividad 5

Actividad 6. Realizar un programa que controle el flujo de automóviles y permita el paso de los peatones en un semáforo de CU en la UNAM. Para las luces del semáforo se empleará la barra de leds RGB Neopixel; de acuerdo a lo siguiente:

Automóviles		
LED1	LED2	LED3
ROJO	AMARILLO	VERDE

Peatón		
LED4	LED5	LED6
ROJO	AMARILLO	VERDE

Características de diseño.

a. Funcionamiento normal para flujo de automóviles; se controla con un núcleo.
b. El peatón cuenta con un botón de cada lado de la avenida (petición de paso); cuando exista la solicitud de paso del peatón, se activará una interrupción para avisar al primer núcleo que hay una solicitud de paso.
c. Una vez que el semáforo está en rojo, cederá el paso al peatón durante 10 segundos.
d. La rutina del paso anterior (semáforo del peatón), se ejecutará en el segundo núcleo.
e. El tiempo para paso de peatón será mostrada en el display TM1637 y hará sonar un zumbador.

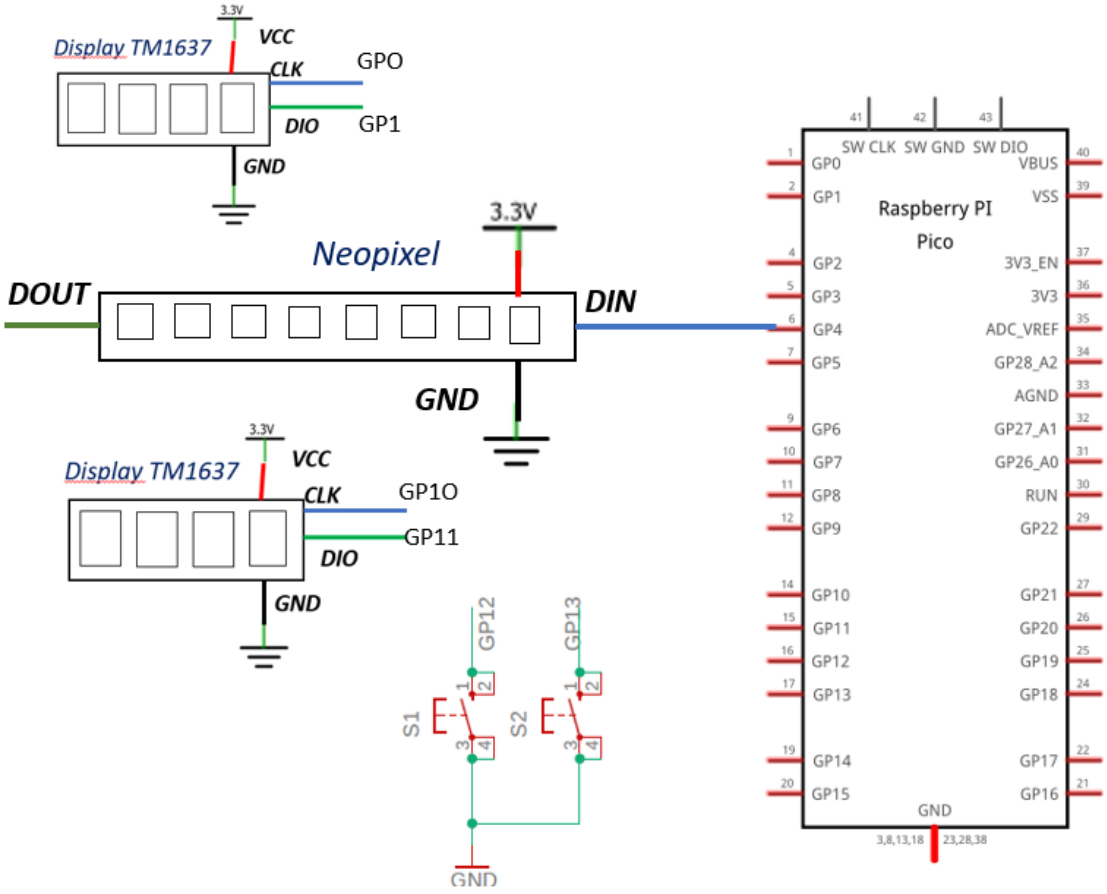


Figura 11-5. Circuito; actividad 6

Laboratorio de Microcomputadoras
Practica No. 12
Uso y aplicaciones de WiFi

Objetivo. Introducir al estudiante en el control de sistemas empleando la comunicación WiFi, para la realización de aplicaciones de tipo IOT.

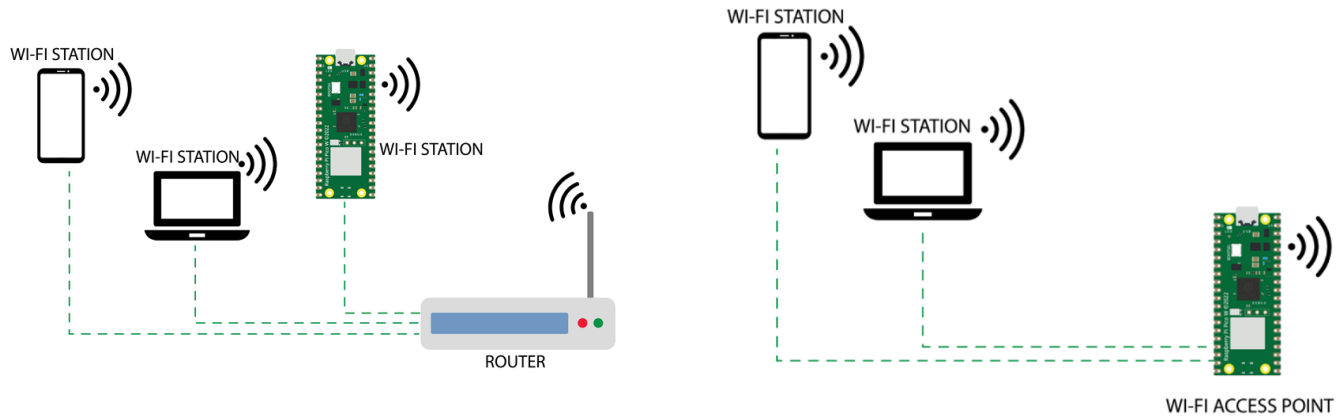
1 Introducción

Las versiones Raspberry Pi Pico W y Raspberry Pi Pico 2W contienen un módulo WiFi de 2.4 GHz con protocolo 802.11n, lo que permite conectarse a internet y adentrarse al mundo IOT.

Es posible implementar alguno de los tipos de funcionamiento permitidos, con las características que demanda cada una de ellas, como:

*A. Modo Punto de Acceso o **AP** (Access Point).*

*B. Modo Estación o **STA** (Station).*



a. Modo Estación

b. Modo Access Point

Figura 12-1. Modos de trabajo WiFi; Raspberry Pi Pico W

Los recursos de software requeridos para uso del módulo UART son:

import network	#	*Importar la librería network; proporciona herramientas de conexión y configuración.*
Import sockets	#	*Importar la librería sockets*

2 Requerimientos

2.1. Software.

- *Micropython*

2.2. Editor.

- *Thonny*

2.3. Hardware.

- *Raspberry Pi Pico W*
- *Plataforma embebida 1 o 2 para Raspberry Pi Pico*
- *2 Push button*
- *Leds*
- *Interruptores*
- *Potenciómetro*

3 Desarrollo

Realizar las actividades solicitadas; es altamente recomendable haber capturado los programas como actividad previa a la sesión.

Actividad 1. Escribir el siguiente código; con ayuda de la documentación oficial comentar que hace cada línea e indicar que hace; Reportar mediante captura de pantalla la ejecución del mismo.

```
import network

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
networks = wlan.scan()

print("Redes cercanas:")
for network_info in networks:
    print(network_info)
```

Figura 12-2. Código; actividad 1

Actividad 2. Para el siguiente código, identificar los parámetros faltantes. Complementar con los datos indicados, escribir el código; con ayuda de la documentación oficial comentar que hace cada línea, e indicar que hace el programa; reportar mediante captura de pantalla la ejecución del mismo.

```
import network
from time import sleep

ssid = 'RED_WIFI'
password = 'CONTRASEÑA'
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)
connection_timeout = 10

while connection_timeout > 0:
    if wlan.status() >= 3:
        break
    connection_timeout -= 1
    print('Espera conexión WIFI...')
    sleep(1)

if wlan.status() != 3:
    raise RuntimeError('Error en conexión')
else:
    print('Conexión establecida')
    network_info = wlan.ifconfig()
    print('IP address:', network_info[0])
```

Figura 12-3. Código; actividad 2

Actividad 3. Escribir el siguiente código (pagina HTLM), ejecutar y dar el formato de su preferencia, incluir el o los nombres de los integrantes de su equipo.

```
<html>
<head>
    <title>Control de un LED GPIO 2...</title>
</head>
<body>
    <h1>Control de LEDs</h1>
    <p><a href="/?led1=on"><button>LED 1 ON</button></a>
       <a href="/?led1=off"><button>LED 1 OFF</button></a></p>
</body>
</html>
```

Figura 12-4. Página HTLM; actividad 3

Actividad 4. Escribir el siguiente programa, incrustando las modificaciones realizadas en la actividad anterior, comentar el código y comprobar su funcionamiento.

```
import network
import socket
import machine
import time

led1 = machine.Pin(2, machine.Pin.OUT)

ssid = 'RED_WIFI'
password = 'Contraseña_WIFI'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

print("Conectando a Wi-Fi...")
while not wlan.isconnected():
    time.sleep(1)
    print(".", end="")

print("\nConectado a Wi-Fi!")
print("Dirección IP:", wlan.ifconfig()[0])
```

```
def web_page():
    html = """
    <html>
    <head>
        <title>Control de un LED GPIO 2...</title>
    </head>
    <body>
        <h1>Control de LEDs</h1>
        <p><a href="/?led1=on"><button>LED 1 ON</button></a>
           <a href="/?led1=off"><button>LED 1 OFF</button></a></p>
    </body>
    </html>
    """
    return html
```

```
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
s = socket.socket()
s.bind(addr)
s.listen(5)
print("Servidor web escuchando en", addr)

while True:
    cl, addr = s.accept()
    print('Conexión de', addr)
    request = cl.recv(1024)
    request = str(request)

    if "/?led1=on" in request:
        led1.value(1)
    if "/?led1=off" in request:
        led1.value(0)
    response = web_page()
    cl.send('HTTP/1.1 200 OK\n')
    cl.send('Content-Type: text/html\n')
    cl.send('Connection: close\n\n')
    cl.sendall(response)
    cl.close()
```

Figura 12-4 Código actividad 4

Actividad 5. Realizar el control (Web Server) de 4 salidas GPIO (0, 1, 2 y 3), de manera que contenga cuatro botones de apagado y encendido de estas señales; la página Web queda a criterio del estudiante.

Actividad 6. Realizar un control (Web Server), de manera que controle, el funcionamiento de 4 GPIO como salida, 2 entrada digitales, 1 entrada analógica y se permita controlar la salida PWM; elegir los recursos que sean pertinentes en cada caso, para ver el funcionamiento de esta aplicación.

Referencias

https://thonny.org/

https://www.raspberrypi.com/

https://micropython.org/

https://cpulator.01xz.net/

https://www.raspberrypi.com/documentation/

https://developer.arm.com/documentation