

Proyecto 1.

Fredy Alberto Cuellar Torres

Marco Alfredo Loaiza Carrillo

Sistemas embebidos



Pontificia Universidad Javeriana
Facultad de Ingeniería
Marzo 19 2021

1. Abstract

Un sistema embebido es un sistema de cómputo compuesto por hardware y software diseñado para realizar funciones específicas o dedicadas que hacen parte de alguna aplicación o producto o de un sistema más grande.

La programación de estos sistemas se puede realizar en lenguajes interpretados o lenguajes compilados, los dos traducen el código escrito a código máquina o binario para que el sistema pueda entenderlo.

En el presente documento se muestra la comparación de tres lenguajes de programación (dos interpretados y uno compilado), realizando la misma tarea con el fin de evaluar el consumo de CPU y la rapidez de ejecución de cada lenguaje, para de esta manera poder determinar o seleccionar el lenguaje que mejor se adapte a un diseño de acuerdo con sus especificaciones y requerimientos.

An embedded system is a computer system made up of hardware and software designed to perform specific or dedicated functions that are part of some application or product of a larger system.

The programming of these systems can be done in interpreted languages or compiled languages, both of which translate the written code into machine or binary code so that the system can understand it.

This document shows the comparison of three programming languages (two interpreted and one compiled), performing the same task in order to evaluate the CPU consumption and the speed of execution of each language, in order to determine or select the language that best suits a design according to the specifications and requirements.

2. Introducción

El proyecto propone dos objetivos principales, el primero es comparar la frecuencia de conmutación de un pin de GPIO de una tarjeta de desarrollo raspberry pi, al programar la activación o desactivación (subida / bajada) del pin en tres lenguajes de programación diferentes(C/C++, Python o Bash).

El segundo objetivo es el de utilizar periféricos externos de la tarjeta para medir mediante un sensor one wire la variable de medida del sensor y posteriormente guardar esta información en un archivo .csv cada determinado tiempo.

Para el desarrollo del primer objetivo se requiere:

- Escribir y compilar un programa en C o C++, utilizando librerías de estos lenguajes que permitan subir o bajar el pin GPIO seleccionado.
- Escribir un programa en python3, que suba y baje un pin de GPIO en la tarjeta a la frecuencia más alta que le sea posible.
- Escribir un programa en bash, que suba y baje un pin de GPIO en la tarjeta a la frecuencia más alta que le sea posible.
- Comparar la frecuencia de conmutación del pin GPIO, entre los 3 programas utilizando un osciloscopio.

Para el desarrollo del segundo objetivo se requiere:

- Implementar una aplicación que lea los valores del sensor y los almacene anexando cada 10 segundos la medición en un archivo llamado: *AAAAMMDD TEMPERATURA.csv*.
- La primera columna del archivo corresponde a la fecha y hora actual en formato “AAAAMMDD HHMMSS” teniendo en cuenta el espacio indicado y la segunda columna corresponde a la temperatura leída desde el sensor.

A continuación se muestra el desarrollo del proyecto.

3. Arquitectura de Bloques General

En el presente proyecto se implementan 3 tipos de lenguajes de programación como lo son: C, Python 3 y bash, con el objetivo de bajar y subir un pin GPIO en la tarjeta raspberry Pi 2 Model B V1.1, se realiza la medición con un osciloscopio para comparar las frecuencias de cada una de las librerías y lenguajes utilizados en el desarrollo del proyecto, por último se utiliza un sensor de temperatura onewire (DS18B20) que por medio de una aplicación almacena los datos en un archivo AAAAMMDD TEMPERATURA.csv. cada 10 segundos.

Teniendo en cuenta el proyecto, se divide en dos fases, la primera corresponde a la subida y bajada de un pin GPIO y la segunda la adquisición de datos de un sensor de temperatura (DS18B20).

En la figura 1. se observa el diagrama de bloques para la fase la sección de configuración de GPIO.

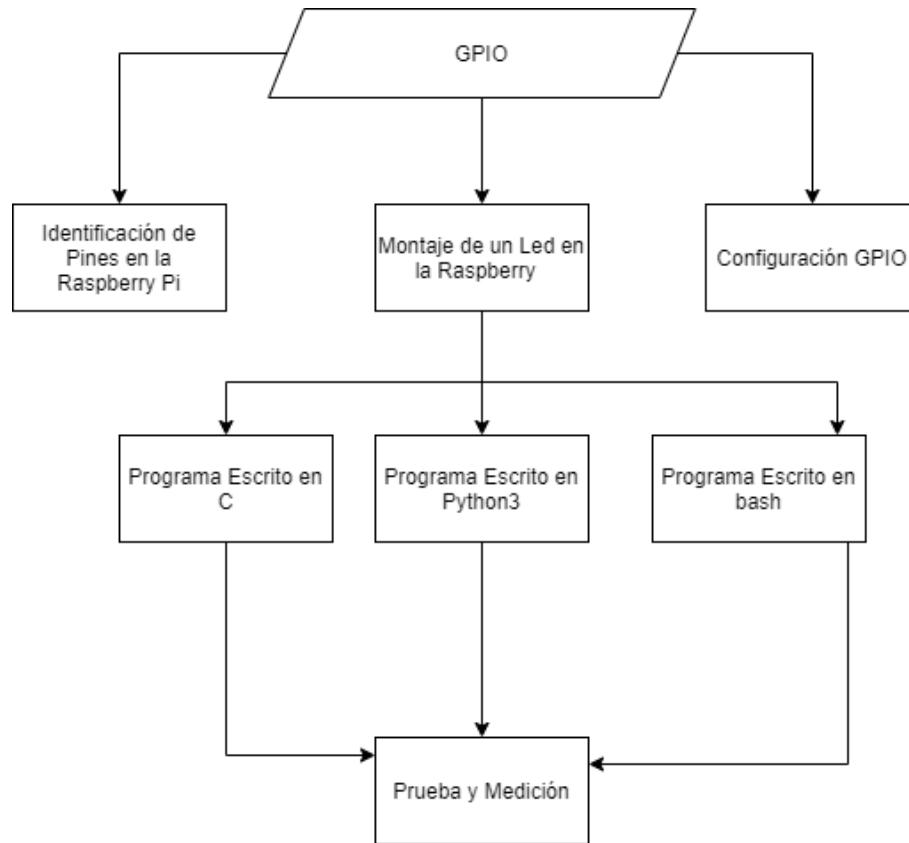


Figura 1: Diagrama de bloques GPIO

General Purpose Input Output (GPIO): Hace referencia a un sistema de entrada y salida de la tarjeta Raspberry Pi, de propósito general, estas I/O consisten o hacen referencia a una serie de pines o conexiones que se pueden usar como entradas o salidas para múltiples usos.

Identificación de pines en la Raspberry Pi: Teniendo en cuenta que el modelo a utilizar es (Raspberry Pi 2 Model B v1.1), se hace necesario realizar la identificación de pines para realizar el montaje correspondiente, para ello nos basamos en la guía que se puede observar en la Figura 2.

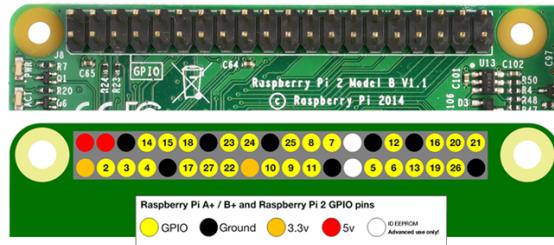


Figura 2: Pines GPIO Raspberry Pi2, tomado de <https://www.raspberrypi.org/documentation/usage/gpio/>

Montaje de un led en la Raspberry Pi: Para realizar el montaje de un led para comprobar la orden enviada al Gpio, se hace necesario indicar que el GPIO a trabajar es el 17 que cumple con las siguientes características:

pin físico 11
BCM 17 BCM pin 17
Wiring Pi pin 0

Teniendo en cuenta la anterior configuración, realizamos el siguiente montaje en la raspberry:

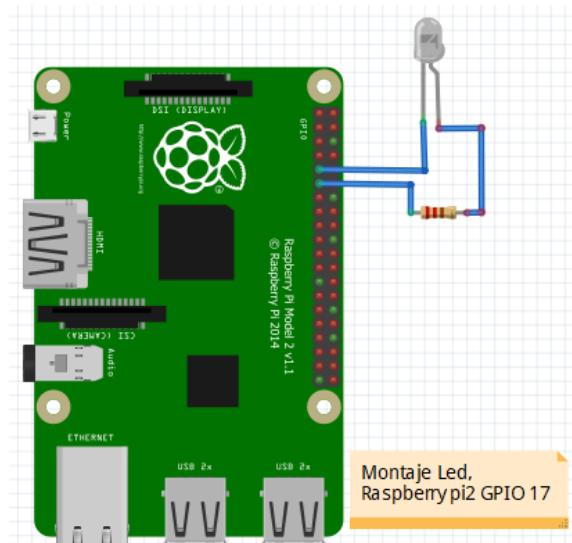


Figura 3: Conexión GPIO 17 Raspberry Pi2, elaboración propia

Configuración GPIO: Es importante conocer la configuración de cada uno de los GPIOs en la raspberry para ello y en la terminal de la tarjeta optamos por el siguiente comando:

Gpio readall

Programación escrito C, Python3, Bash: Para los pogramas de cada uno de ellos se utilizaron diferentes librerías que se podrán apreciar en el capítulo 5.

Prueba y medición: Para realizar la prueba de cada uno de los casos se verificará con el adecuado funcionamiento del Led y se realiza la medición en un osciloscopio para conocer la frecuencia de cada uno de los programas implementados y poder tomar conclusiones.

Para la segunda fase corresponde la adquisición de datos de un sensor de temperatura (DS18B20). y almacenarlos en un documento AAAAMMDD TEMPERATURA.csv. este registro se realiza cada 10 segundos

En la figura 4. Se observa el diagrama de bloques para la fase de configuración del sensor de temperatura (DS18B20).

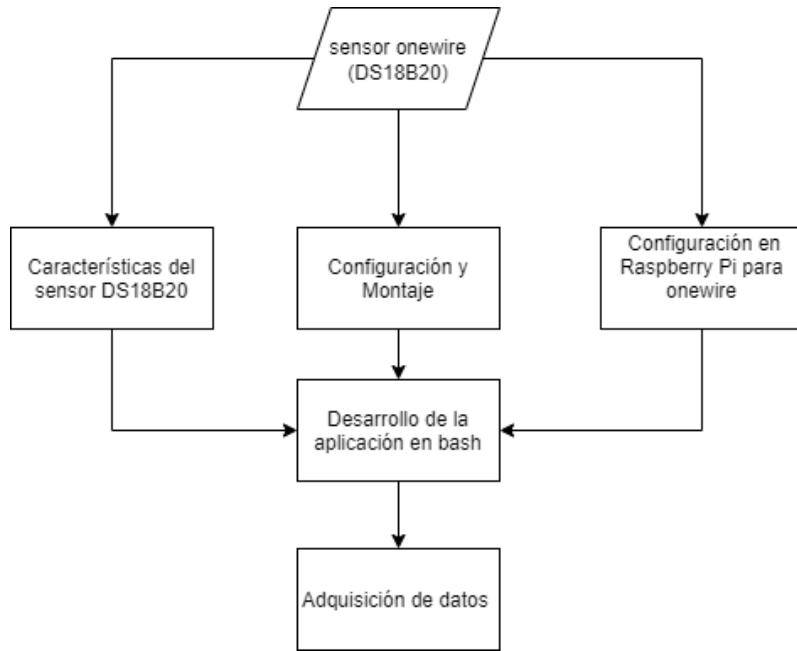


Figura 4: Diagrama sensor de temperatura DS18B20

Características del sensor DS18B20: El sensor DS18B20 se comunica con el protocolo de comunicación “One-Wire”, un protocolo de comunicación serial que utiliza un solo cable para transmitir las lecturas de temperatura a la raspberry.

Las especificaciones técnicas son:

- Rango de -55 ° C a 125 ° C
- Voltaje de operación de 3.0V a 5.0V
- Muestreo de 750 ms
- 0,5 ° C (9 bits); 0,25 ° C (10 bits); 0,125 ° C (11 bits); Resolución de 0.0625 ° C (12 bits)
- Dirección única de 64 bits
- Protocolo de comunicación de un cable

En la figura 5, se puede observar el sensor a utilizar



Figura 5: Sensor DS18B20, tomado de: <https://www.mactronica.com.co/>

Configuración y Montaje: Para la conexión del sensor se configura en el pin BCM 4, el cual cumple con las siguientes características:

pin físico 7	
BCM 4	BCM pin 4
	Wiring Pi pin 7

El montaje del sensor en la raspberry pi2 se puede observar en la figura 6

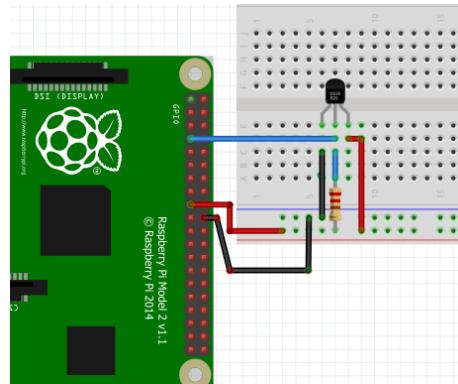


Figura 6: Conexión Sensor DS18B20, elaboración propia

Configuración en Raspberry Pi: Es importante tener en cuenta la configuración de la raspberry para ello se investigó el protocolo de W1-GPIO - One-Wire Interface, consultado en pinout.xyz.

Desarrollo de la aplicación en bash y adquisición de datos: Se desarrolló la aplicación en bash para adquirir los datos y almacenarlos en un archivo de tipo AAAAMMDD TEMPERATURA.csv. cada 10 segundos

4. Desarrollo de la Solución

A continuación se presenta el desarrollo del proyecto por objetivos, donde se incluyen los diagramas de flujo y código ejecutado en cada punto.

Inicialmente se observa la configuración de los pines de la tarjeta utilizando el comando readall para identificar los pines seleccionados con el uso de las diferentes librerías como se muestra en la figura 7.

Pi 3B													
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM			
		3.3v			1	2		5v					
2	8	SDA.1	IN	1	3	4		5v					
3	9	SCL.1	IN	1	5	6		0v					
4	7	GPIO. 7	IN	1	7	8	0	IN	TxD	15	14		
		0v			9	10	1	IN	RxD	16	15		
17	0	GPIO. 0	IN	0	11	12	0	IN	GPIO. 1	1	18		
27	2	GPIO. 2	IN	0	13	14		0v					
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4	23		
		3.3v			17	18	0	IN	GPIO. 5	5	24		
10	12	MOSI	IN	0	19	20		0v					
9	13	MISO	IN	0	21	22	0	IN	GPIO. 6	6	25		
11	14	SCLK	IN	0	23	24	1	IN	CE0	10	8		
		0v			25	26	1	IN	CE1	11	7		
0	30	SDA.0	IN	1	27	28	1	IN	SCL.0	31	1		
5	21	GPIO.21	IN	1	29	30		0v					
6	22	GPIO.22	IN	1	31	32	0	IN	GPIO.26	26	12		
13	23	GPIO.23	IN	0	33	34		0v					
19	24	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27	16		
26	25	GPIO.25	IN	0	37	38	0	IN	GPIO.28	28	20		
		0v			39	40	0	IN	GPIO.29	29	21		

Figura 7: Pines GPIO raspberry.

Observamos los nombres de las columnas donde aparece BCM, WPi, Name, Mode, V y Physical. Las columnas de Nombre, modo y V indican el estado del pin donde se puede observar el nombre del pin, su modo si está configurado como de entrada o salida y el voltaje indica 1 si está encendido (3.3V) o 0 si esta apagado (Tierra).

Las columnas BCM, Wpi y Physical hacen referencia al sistema de numeración de pines que se debe tener en cuenta al momento de la conexión y programación de la tarjeta siendo estas librerías posibles de cargar en los scripts.

4.1. Primer objetivo

Se crean los scripts de los tres lenguajes de programación, para los tres casos se implementa el mismo diagrama de flujo teniendo en cuenta que se busca realizar la

misma tarea y la lógica del código es la misma.

Se implementa un bucle infinito con un while (true), para que el programa ejecuta la acción hasta que el usuario interrumpe el programa con el comando Ctrl + C, donde se sube y se baja el pin seleccionado previamente de acuerdo a la librería o método seleccionado.

En todos los códigos se introduce un tiempo de espera, delay o sleep para poder verificar el correcto funcionamiento del programa añadiendo a la salida un led y una resistencia, estos permiten observar el cambio de estado "subida y bajada" prendiendo y apagando el led, en caso de querer realizar la verificación se debe quitar los comentarios del código para ejecutarlo.

A continuación, se presenta el diagrama de flujo del código a implementar.

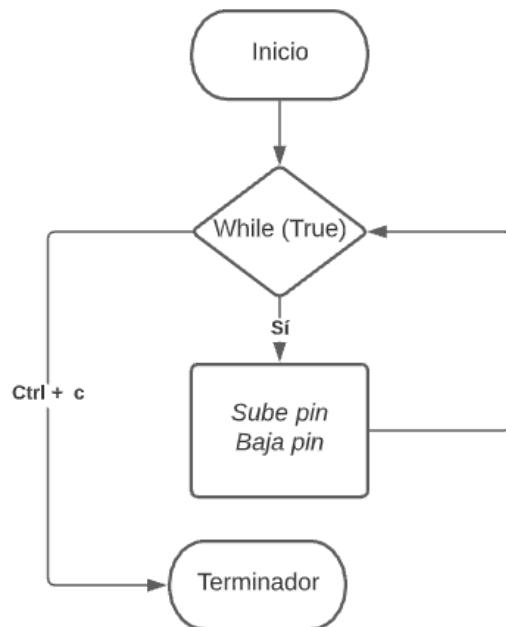


Figura 8: Diagrama de flujo programación.

4.1.1. C/C++

En el caso de C/C++ se implementa el código utilizando dos librerías diferentes WirinigPi y BCM2835.

```
1 #include <wiringPi.h>
2 int main (void)
3 {
4     wiringPiSetup();
5     pinMode(0, OUTPUT);
6     int i = 0;
7     for (i = 0;i<10;i++){
8         digitalWrite(0, HIGH);
9         delay(500);
10        digitalWrite(0, LOW);
11        delay(500);
12    }
13    return 0 ;
14 }
```

```
1 /*
2 Cgpio.c
3 Para compilar el script utilice el siguiente comando:
4 gcc -o nombredeejecutable cgpio.c -l bcm2835
5 Descomentar las lineas de Delay para pruebas con led.
6 /
7
8 #include <bcm2835.h>
9 #include <stdio.h>
10
11 #define LED RPI_GPIO_P1_11
12 #define Delay 500
13
14
15 int main(int argc, char **argv)
16 {
17     if (!bcm2835_init()) //Inicializa la libreria
18     return 1;
19
20     bcm2835_gpio_fsel(LED,BCM2835_GPIO_FSEL_OUTP); //Configura el pin como salida
```

```

21
22     while (1)
23     {
24         bcm2835_gpio_write(LED, HIGH);      // Sube el pin - prende el led
25         bcm2835_delay(Delay);           // espera un momento
26         bcm2835_gpio_write(LED, LOW);    // Baja el pin Apaga el led
27         bcm2835_delay(Delay);          // Espera un momento
28     }
29     bcm2835_close();
30     return 0;
31 }
```

4.1.2. Python

Para el script de python se utiliza la librería de RPi.GPIO, a continuación se presenta el código.

```

1  #! /bin/python3
2
3  #Script para levantar y bajar un pin con Python
4  #Para probar con led por favor descomente los comando time.sleep()
5
6  import RPi.GPIO as gpio
7  import time
8
9  gpio.setmode(gpio.BOARD)
10 gpio.setup(11, gpio.OUT)
11
12 print(" Ejecutando el script para finalizar presione Ctrl + c")
13
14 while True:
15     gpio.output(11, True)
16     time.sleep(0.5)
17
18     gpio.output(11, False)
19     time.sleep(0.5)
```

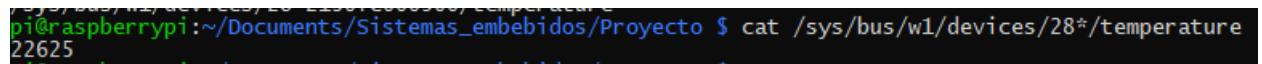
4.1.3. Bash

Finalmente se realiza el script en bash como se muestra a continuación.

```
1  #! /bin/bash
2
3  # Script para subir y bajar un pin.
4  # para la demostracion descomente los sleep y conecte la salida a un led.
5
6  gpio mode 0 out
7
8  echo "Ejecutando el script .... para detener presione Ctrl + c"
9
10 while true
11 do
12     gpio write 0 1
13     sleep 1
14     gpio write 0 0
15     sleep 1
16 done
```

4.2. Segundo objetivo

Para el desarrollo del segundo objetivo, se realiza la conexión del sensor de temperatura al pin correspondiente de one wire correspondiente al pin 7 físico o pin 4 BCM, se procede con la verificación de lectura de la variable dirigiéndose a la ruta `/sys/bus/w1/devices/28*/temperature` como se muestra en la siguiente figura.



```
b1@raspberrypi:~/Documents/Sistemas_embebidos/Proyecto $ cat /sys/bus/w1/devices/28*/temperature
22625
```

Figura 9: Lectura del valor entregado por el sensor de temperatura.

Verificando que el sensor entrega los datos se procede a realizar el programa que sobreesciba los datos capturados por el sensor cada 10 segundos como se muestra en el siguiente diagrama.

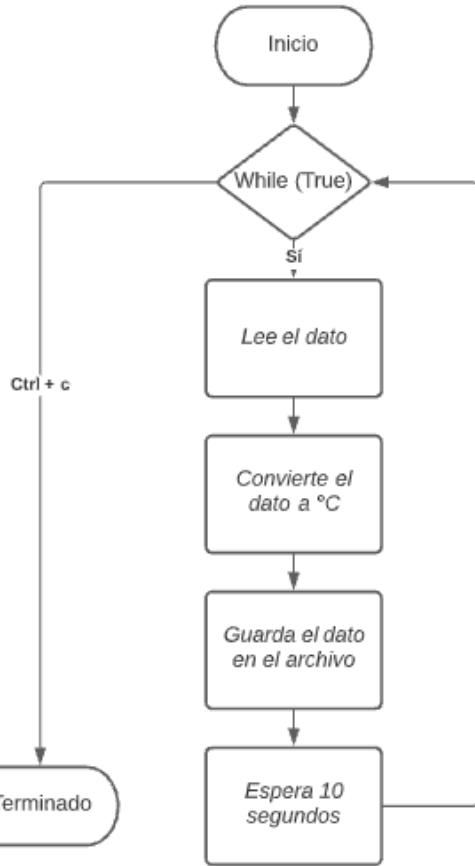


Figura 10: Diagrama de flujo de captura y guardado de informacion en el archivo.

El lenguaje de programación seleccionado para esta tarea es bash y se muestra el código a continuación.

```

1  #! /bin/bash
2
3  #El script captura la informacion del sensor
4  #transforma el valor de tempertura
5  #y guarda los valores junto a la fecha cada 10 segundos.
6
7
8  echo "Ejecutando el script .... para finalizar presione Ctrl + c "
9
10 while true
11 do

```

```

12      a=$(date +%Y%m%d' '%H%M%S)
13      b=$(cat /sys/bus/w1/devices/28-2130fc000900/temperature)
14      c=$((echo $((b/1000))'.'$((b%100))' °C')
15
16      echo $a';'$c >> $(date +%Y%m%d)_TEMPERATURA.csv
17      sleep 9.1
18
19      done

```

En el script anterior se crean tres variables auxiliares:

- a: guarda la informacion de la fecha y hora en que se ejecuta el comando.
- b: guarda el valor de temperatura entregado por el sensor.
- c: convierte el valor guardado en la variable b en °C

Posterior a esto se guarda la fecha-hora y el valor transformado en el archivo .csv con los nombres presentados en los requerimientos.

Finalmente se da un tiempo de espera de 9.1 segundos, teniendo en cuenta que el programa tiene un tiempo de ejecución, que en total tome 10 segundos y las lecturas sean correctas.

5. Pruebas

En el presente capitulo se realizan las respectivas pruebas de cada uno de los objetivos propuestos en el proyecto 1.

El protocolo de pruebas se diseña para demostrar el correcto funcionamiento de los scripts realizados, y poder medir las frecuencias de ejecución de cada uno de ellos, para finalmente poder realizar la comparación de tiempo de ejecución de los diferentes lenguajes.

Para esto se propone:

- Ejecutar los scripts utilizando como salida un led para comprobar que el pin este subiendo y bajando como se requiere.

- Ejecutar los scripts y medir con un osciloscopio las frecuencias de conmutación de los diferentes lenguajes.
- Ejecutar los scripts y observar el consumo de CPU sobre la tarjeta.
- Ejecutar el script para guardar la información que brinda el sensor cada 10 segundos y visualizar el archivo de salida.

Para el primer punto del protocolo de pruebas se realiza el montaje del led como se muestra en la figura 11.

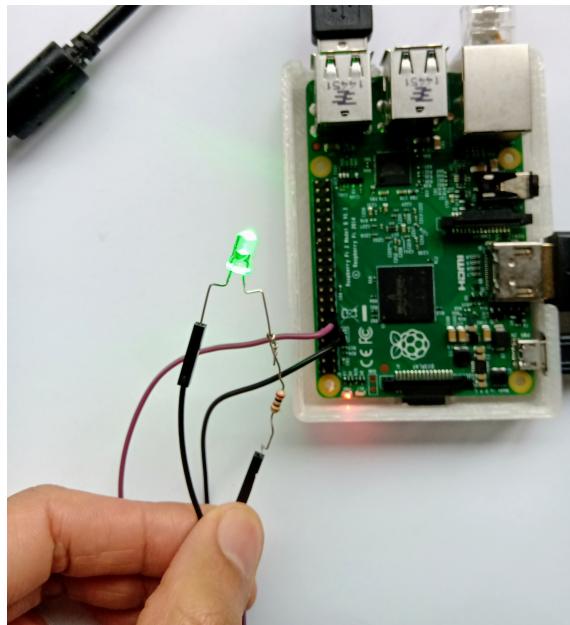


Figura 11: Montaje del circuito de prueba con Led.

En los diferentes programas se selecciona el mismo pin de salida (pin 11) para que el montaje funcione para todos los casos.

A continuación, se presentan los diferentes lenguajes programados.

Caso C/C++:

En el lenguaje de C/C++ se prueban dos scripts utilizando diferentes librerías.

El primero se realiza con la librería WiringPi y el segundo se realiza utilizando la librería BCM. Como estos scripts utilizan un lenguaje compilado, se realiza la compilación de cada uno de ellos con el comando gcc y posteriormente se ejecuta el binario con el código ya compilado.

En la Figura 12, se observa la prueba con la librería WiringPi.

The screenshot shows a terminal window titled "pi@raspberrypi: ~/proyecto_1". It displays the following text:

```
Archivo Editar Pestañas Ayuda  
pi@raspberrypi:~/proyecto_1$ gcc -Wall -o led led.c -lwiringPi  
pi@raspberrypi:~/proyecto_1$ sudo ./led  
pi@raspberrypi:~/proyecto_1$
```

To the left of the terminal, there is a code editor window showing a C file named "led.c". The code contains the following:led.c * led.py * led.sh * sensor.py * sensor2.py * AAAAMMDD_TEMPERATURA.csv *
1 #include <wiringPi.h>
2 int main (void)
3 {
4 wiringPiSetup();
5 pinMode(0, OUTPUT);
6 int i = 0;
7 for (i = 0;i<100;i++){
8 digitalWrite(0, HIGH);
9 delay(500);
10 digitalWrite(0, LOW);
11 delay(500);
12 }
13 return 0;
14 }
15

Figura 12: Programación en C con librería WiringPi

Con la librería BCM obtenemos la siguiente prueba:

The screenshot shows a terminal window titled "pi@raspberrypi: ~/Proyecto". It displays the following text:

```
Archivo Editar Pestañas Ayuda  
pi@raspberrypi:~/Proyecto$ gcc -o cgpio cgpio.c -l bcm2835  
pi@raspberrypi:~/Proyecto$ sudo ./cgpio
```

To the left of the terminal, there is a code editor window showing a C file named "cgpio.c". The code contains the following:cgpio.c *
11 #include <bcm2835.h>
12 #define LED GPIO_P1_11
13 #define Delay 500
14
15 int main(int argc, char **argv)
16 {
17 if (!bcm2835_init()) //Inicializa la libreria
18 return 1;
19 bcm2835_gpio_fsel(LED, BCM2835_GPIO_FSEL_OUTP);
20 while (1)
21 {
22 bcm2835_gpio_write(LED, HIGH);
23 bcm2835_delay(Delay);
24 bcm2835_gpio_write(LED, LOW);
25 bcm2835_delay(Delay);
26 }
27 bcm2835_close();
28 return 0;

Figura 13: Programación en C con librería BCM

Caso python3:

Para el caso de Python se escoge la librería de RPI.GPIO.

En la Figura 14, se observa el programa con su respectivo funcionamiento.

```

1 #! /bin/python3
2
3 import RPi.GPIO as gpio
4 import time
5
6 gpio.setmode(gpio.BOARD)
7 gpio.setup(11, gpio.OUT)
8
9 print( " Ejecutando el script para finalizar presione Ctrl + c")
10
11 while True:
12
13     gpio.output(11, True)
14     time.sleep(0.5)
15
16     gpio.output(11, False)
17     time.sleep(0.5)
18
19

```

Figura 14: Prueba y Montaje con programa en Python3

Caso bash:

En bash no es necesario utilizar librerías, se debe utilizar el comando gpio mode para seleccionar el modo de uso del pin. En la Figura 15, se observa el programa con su respectivo funcionamiento.

```

1 #!/bin/bash
2
3 gpio mode 0 out
4
5 echo "Ejecutando el script .....para detener presione Ctrl + c"
6
7 while true
8 do
9     gpio write 0 1
10    sleep 1
11    gpio write 0 0
12    sleep 1
13 done
14

```

Figura 15: Prueba y Montaje con programa en Bash.

Continuando con el protocolo de pruebas, durante la clase se realiza la medición con osciloscopio de las frecuencias de conmutación de cada uno de los lenguajes, y se toma la medida de consumo de CPU de cada uno de los scripts con el comando htop. Los resultados se presentan en el siguiente capítulo.

Sensor de temperatura DS18B20:

Para cumplir con el último punto del protocolo de pruebas, se realiza el montaje del sensor sobre la raspberry como se muestra en la figura 16, se configura el pin de entrada de datos 1-WIRE correspondiente al pin7 de la tarjeta, se habilita el módulo de comunicación del sensor one wire mediante el comando modprobe, se verifica la conexión del sensor en la ruta /sys/bus/w1/devices como se muestra a continuación.

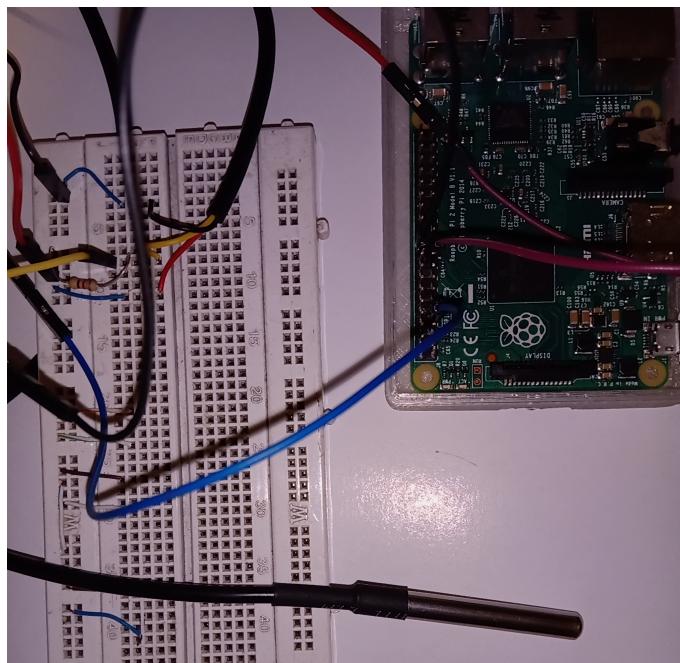


Figura 16: Conexión del sensor DS18B20 a la Raspberry pi2.

```
pi@raspberrypi: /  
Archivo Editar Pestañas Ayuda  
pi@raspberrypi:/ $ dtoverlay=w1-gpio  
pi@raspberrypi:/ $ sudo modprobe w1-gpio  
pi@raspberrypi:/ $ ls /sys/bus/w1/devices/  
28-3c01d0756794 w1_bus_master1  
pi@raspberrypi:/ $ |
```

Figura 17: Configuración e identificación del sensor

Finalizada la configuración del sensor, se dispone a realizar la aplicación por bash y se ejecuta, tal como se observa en la siguiente figura

```
sensorDS18B20.sh x
1 #!/bin/bash
2
3 echo "Ejecutando el script ..... para finalizar presiones Ctrl + c"
4
5 while true
6
7 do
8     a=$(date +%Y%m%d' '%H%M%S)
9     b=$(cat /sys/bus/w1/devices/28-3c01d0756794/temperature)
10    c=$((b/100)).$((b%100)) ^C'
11
12 echo $a';$c >> $(date +%Y%m%d)_TEMPERATURE.csv
13 sleep 9.1
14
15 done
```

Archivo Editar Pestañas Ayuda
pi@raspberrypi:~/proyecto_1 \$. sensorDS18B20.sh
Ejecutando el script para finalizar presiones Ctrl + c

Figura 18: sensor DS18B20 en Bash

Finalmente se observa el archivo 202103_11_TEMPERATURE.csv en donde se almacena los datos cada 10 segundos cumpliendo con los requerimientos planteados en los objetivos.

	sensorDS18B20.sh	20210311_TEMPERATURE.csv
1		20210311 205212;19.12 °C
2		20210311 205222;19.75 °C
3		20210311 205232;19.75 °C
4		20210311 205446;19.87 °C
5		20210311 205456;19.12 °C
6		20210311 205506;19.12 °C
7		20210311 205516;19.12 °C
8		20210311 205526;19.12 °C
9		20210311 205536;19.12 °C
10		20210311 205546;19.12 °C
11		20210311 205556;19.12 °C
12		20210311 205606;19.12 °C
13		20210311 205616;19.50 °C
14		20210311 205626;19.50 °C
15		20210311 205636;19.12 °C
16		20210311 205646;19.12 °C
17		20210311 205656;19.12 °C
18		20210311 205707;19.50 °C
19		

Figura 19: Archivo de almacenamiento de datos del sensor

6. Resultados y Análisis

Las mediciones de consumo de CPU de cada uno de los lenguajes se realizan ejecutando los scripts y visualizando en una segunda ventana el monitoreo de carga de la tarjeta utilizando el comando htop.

Las mediciones de frecuencia de conmutación se toman ejecutando los scripts y visualizando en el osciloscopio el tiempo que le toma al pin en completar un ciclo de subida y bajada como se muestra a continuación.

Resultados C/C++:

Libreria WiringPi:

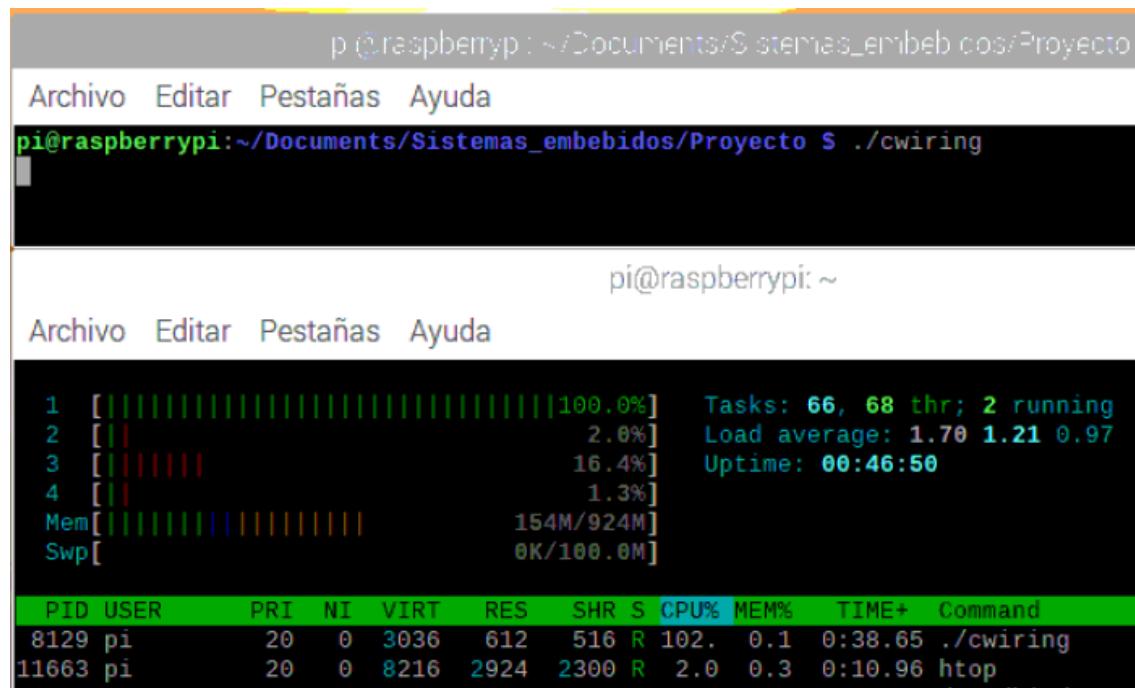
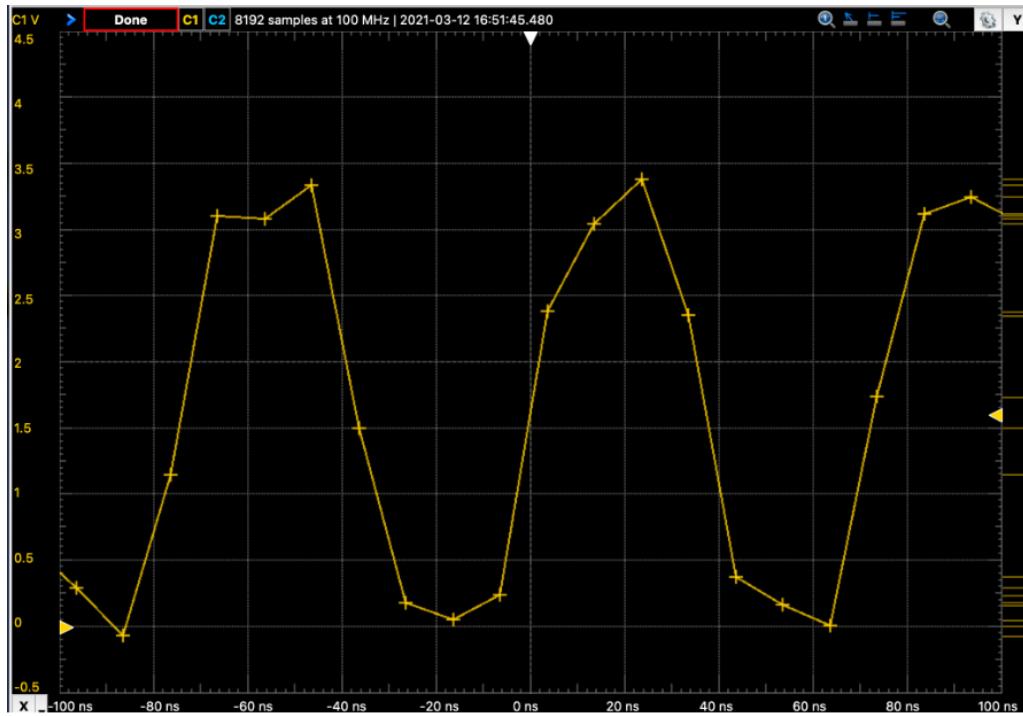


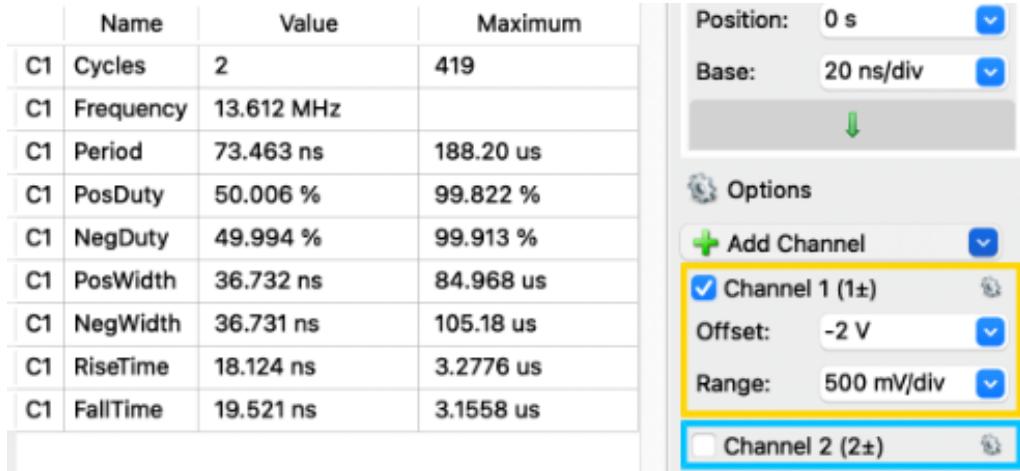
Figura 20: Consumo CPU script C/C++ utilizando librería WiringPi

Evaluando el consumo de recursos la acción de subir y bajar un pin de la tarjeta consume CPU del 98% al 102%, como se observa en la figura 20 la ejecución del script consume en su totalidad uno de los cuatro nucleos presentes en la tarjeta.

Ahora se presentan los tiempos de conmutación del script de C/C++ utilizando la librería de WiringPi.



(a) Gráfica

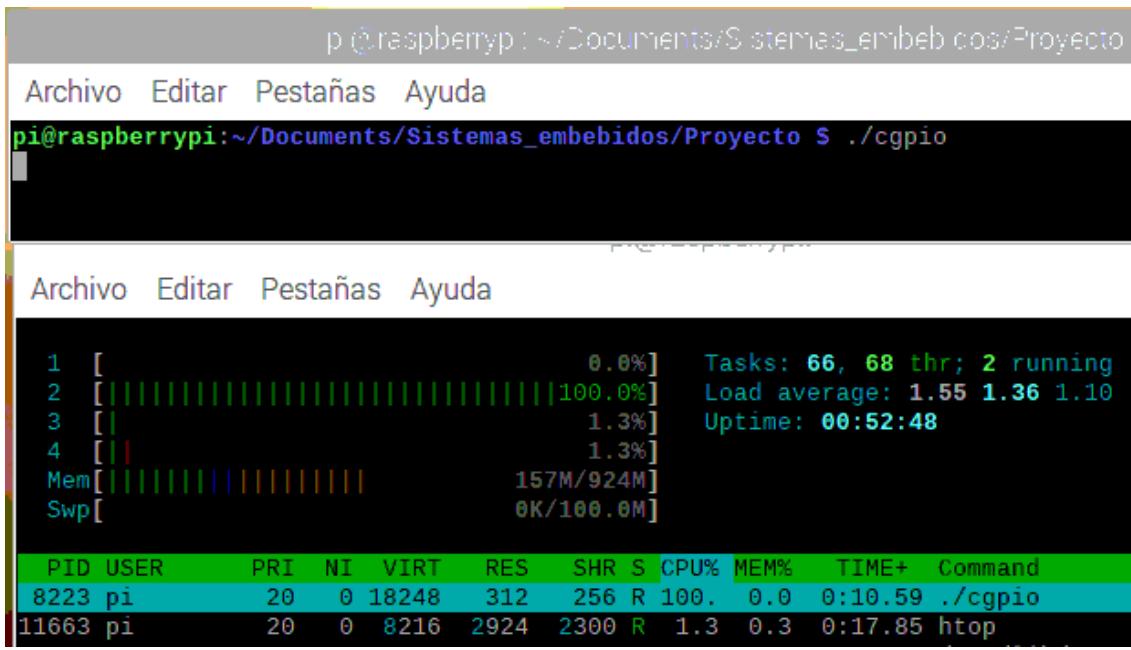


(b) Información de gráfica

Figura 21: Frecuencia de conmutación libreria WiringPi

De la Figura 21 podemos observar que la frecuencia de conmutación de la señal es de 13.612 MHz con un periodo de 73 ns, dada la alta frecuencia de cambio la señal no se ve estable en sus cambios, presentando picos de voltaje en un rango de 3 a 3.5 Voltios y tomando como baja la señal de 0 a 0.5V, sin embargo, el ciclo útil de la señal parece estar en un buen rango de aproximadamente el 50 %.

Libreria BCM:



The screenshot shows a terminal window with the following content:

```
p@raspberrypi:~/Documents/Sistemas_embebidos/Proyecto
Archivo Editar Pestañas Ayuda
pi@raspberrypi:~/Documents/Sistemas_embebidos/Proyecto $ ./cgpio
```

Below the terminal prompt, there is a system status summary:

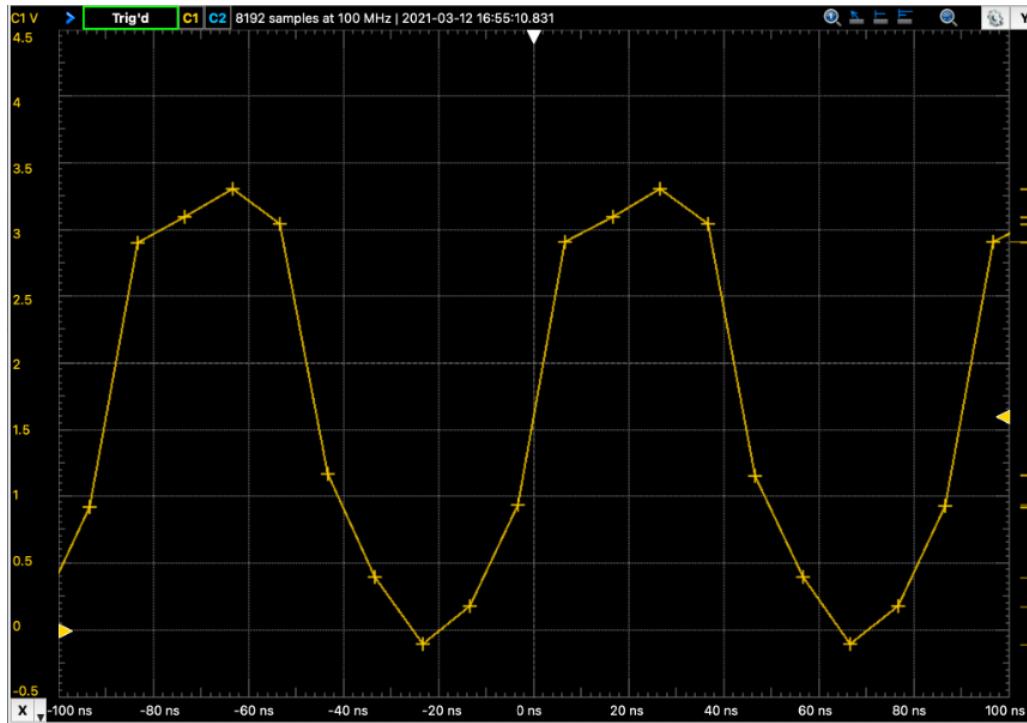
```
1 [          0.0%] Tasks: 66, 68 thr; 2 running
2 [|||||||||||||||||100.0%] Load average: 1.55 1.36 1.10
3 [|        1.3%] Uptime: 00:52:48
4 [||       1.3%]
Mem[||||||| 157M/924M]
Swp[          0K/100.0M]
```

Finally, an htop-style process list is shown:

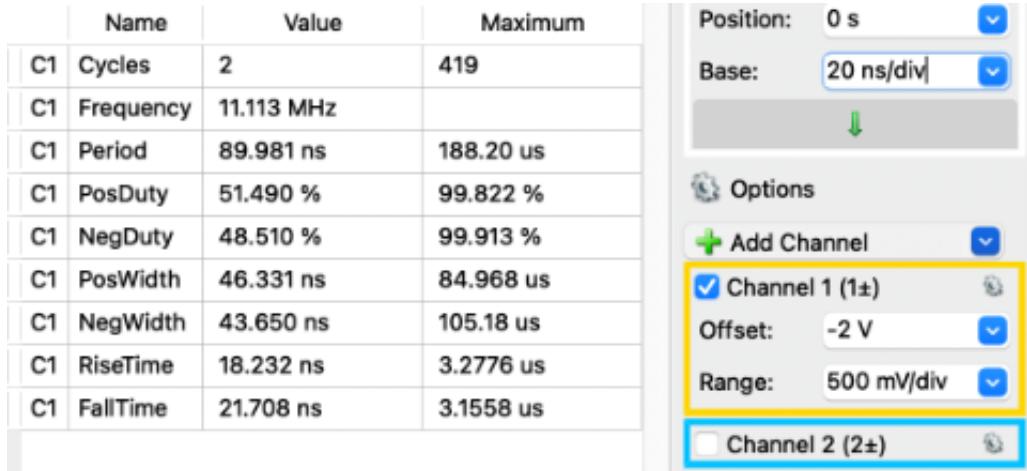
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
8223	pi	20	0	18248	312	256	R	100.	0.0	0:10.59	./cgpio
11663	pi	20	0	8216	2924	2300	R	1.3	0.3	0:17.85	htop

Figura 22: Consumo CPU script C/C++ utilizando libreria BCM

Continuando con el análisis del lenguaje C/C++ y la librería BCM, en cuanto a consumo se observa el mismo comportamiento que con la librería de WiringPi consumiendo entre el 97 % y el 100 % de CPU ocupando totalmente uno de los 4 núcleos presentes en la tarjeta.



(a) Gráfica



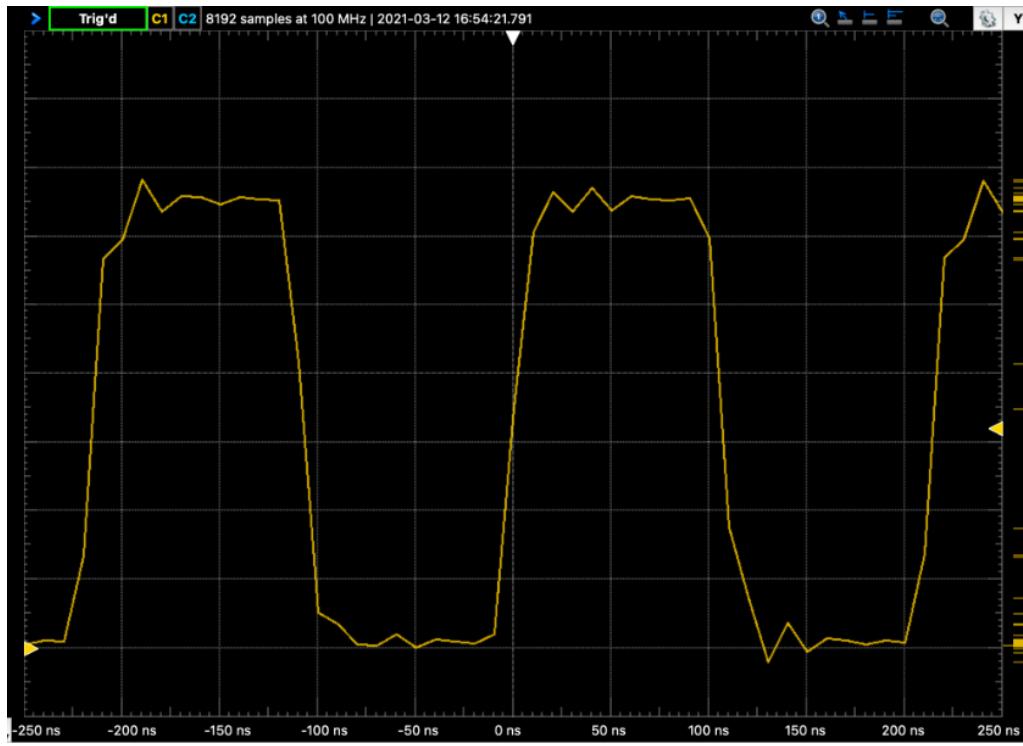
(b) Información de gráfica

Figura 23: Frecuencia de conmutación libreria BCM

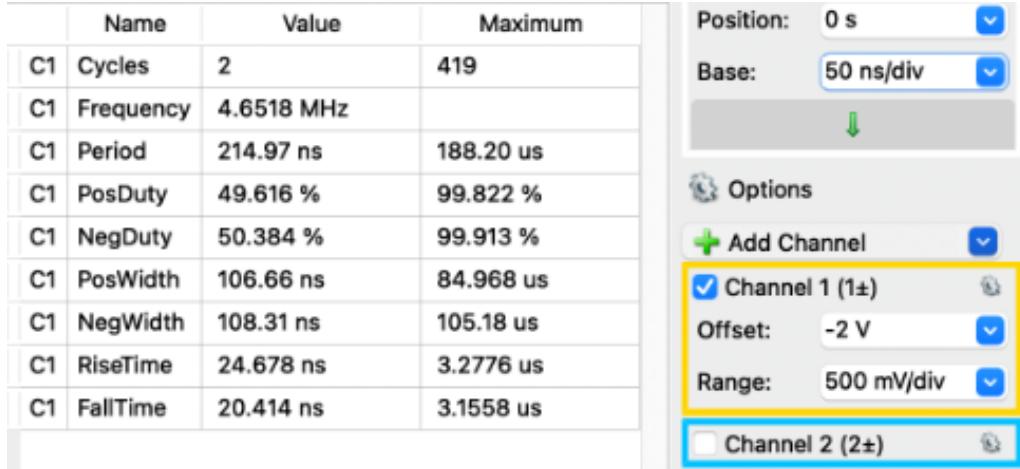
En cuanto a la frecuencia de conmutación se observa una señal que igual a la anterior presenta picos en los niveles alto y bajo entre el rango de 3 a 3.5V y 0 a 0.5V respectivamente, la periodicidad de la señal se observa un poco más estable esto consecuencia de que la frecuencia disminuye con relación a la librería anterior, con un valor de 89.9 ns correspondiente a 11.113 MHz y con un ciclo útil positivo del 51.49 %.

Libreria Piggpio:

Durante la evaluación de los scripts, se presenta esta librería adicional de la cual se obtiene la siguiente información.



(a) Gráfica



(b) Información de gráfica

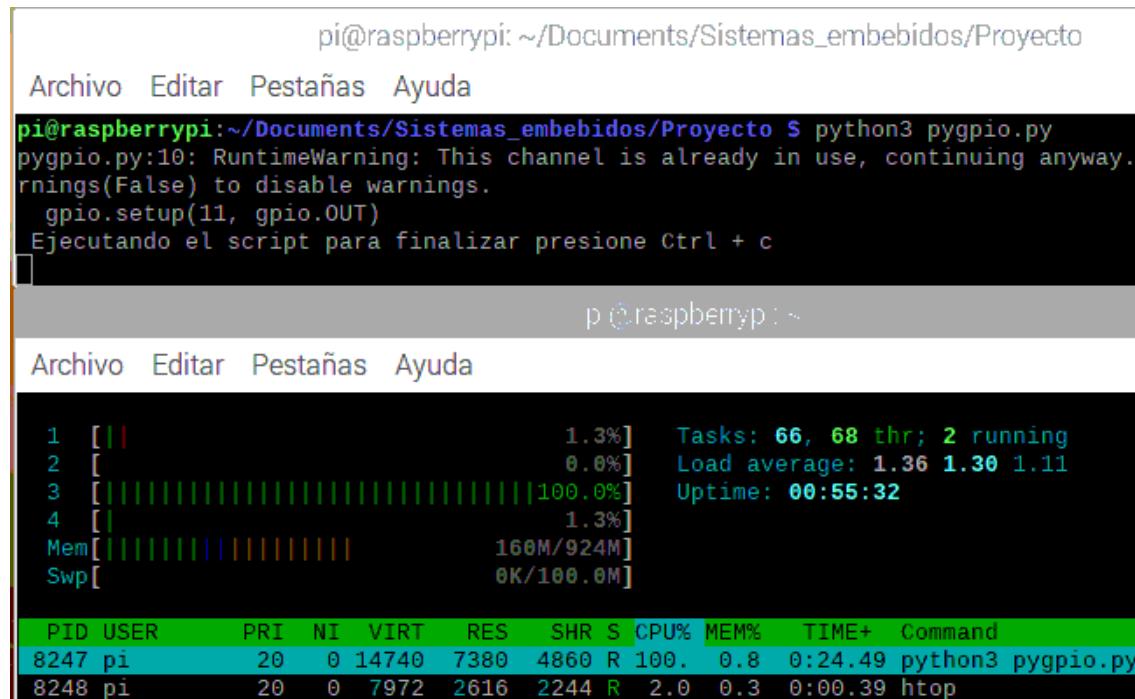
Figura 24: Frecuencia de conmutación libreria Piggpio

Como se observa en la figura 24, esta librería tiene un menor valor de frecuencia de conmutacion correspondiente a 4.6 MHz. Consecuencia de esta reducción, la señal se ve mas estable en sus niveles alto y bajo, en su periodicidad de 214.97 ns su ciclo

útil es de aproximadamente el 50 %.

Resultados Python

Cambiando de lenguaje obtenemos los siguientes datos.



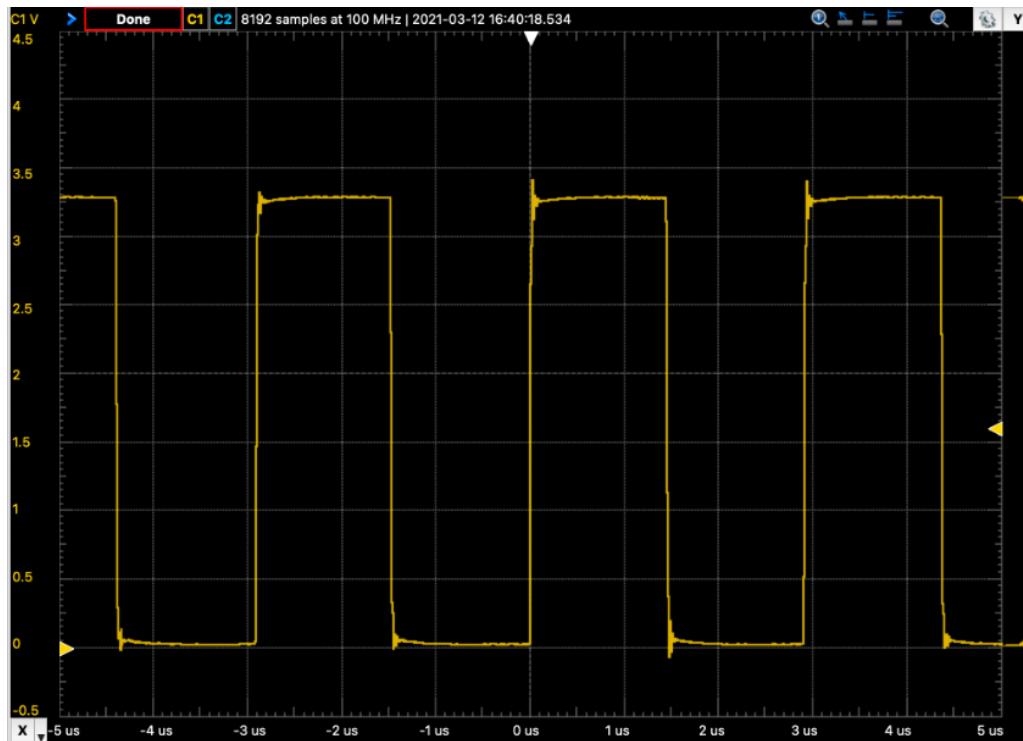
The terminal window shows the command `python3 pygpio.py` being run, which outputs a `RuntimeWarning` about a channel already in use. It also displays a message to press Ctrl + c to exit. Below the terminal is an `htop` process list showing two processes: `python3 pygpio.py` and `htop`. The `htop` output shows CPU usage for four cores, with core 3 at 100.0% and others at 1.3%. Memory usage is shown as 160M/924M and Swap as 0K/100.0M.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
8247	pi	20	0	14740	7380	4860	R	100.	0.8	0:24.49	python3 pygpio.py
8248	pi	20	0	7972	2616	2244	R	2.0	0.3	0:00.39	htop

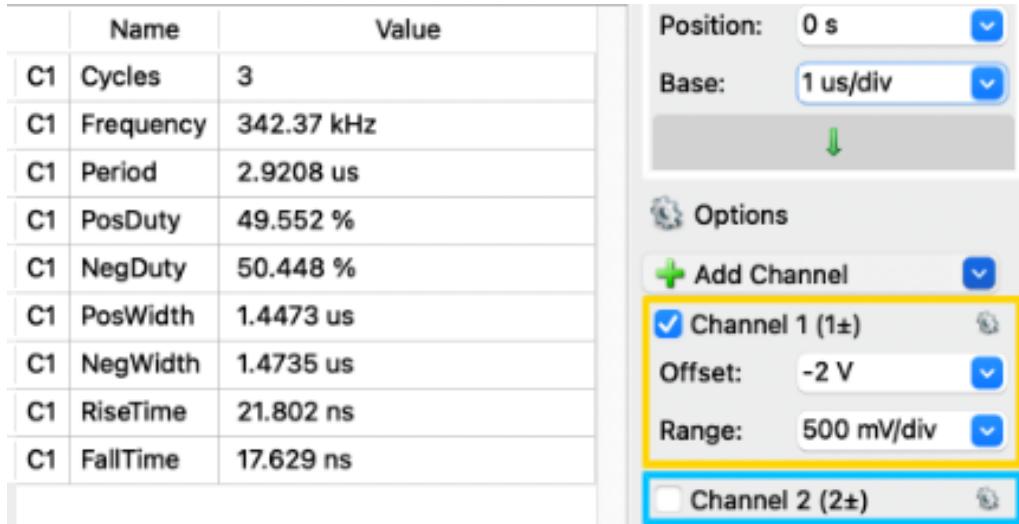
Figura 25: Consumo CPU script Python utilizando libreria RPI.gpio

En cuanto a consumo de recursos, se comporta igual que al lenguaje C/C++ con un valor aproximado del 100 % ocupando igualmente en su totalidad uno de los 4 núcleos de la tarjeta.

En cuanto a la frecuencia de comutación se observa una reducción considerable comparado con el lenguaje presentado anteriormente a un valor de 342.37 kHz correspondiente a un periodo de 2.92 us con un ciclo útil aproximado al 50 %, como se observa en la figura 26.



(a) Gráfica

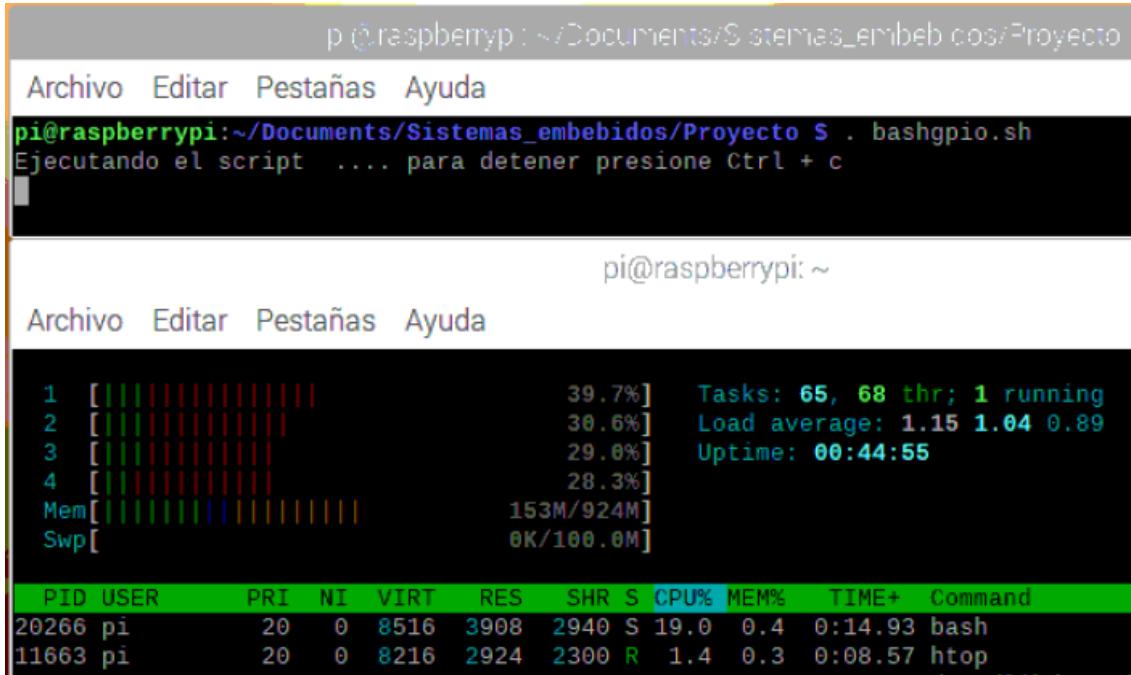


(b) Información de gráfica

Figura 26: Frecuencia de commutación Python libreria RPI.gpio

Resultados Bash

Finalmente se evalua el comportamiento del script escrito en Bash.



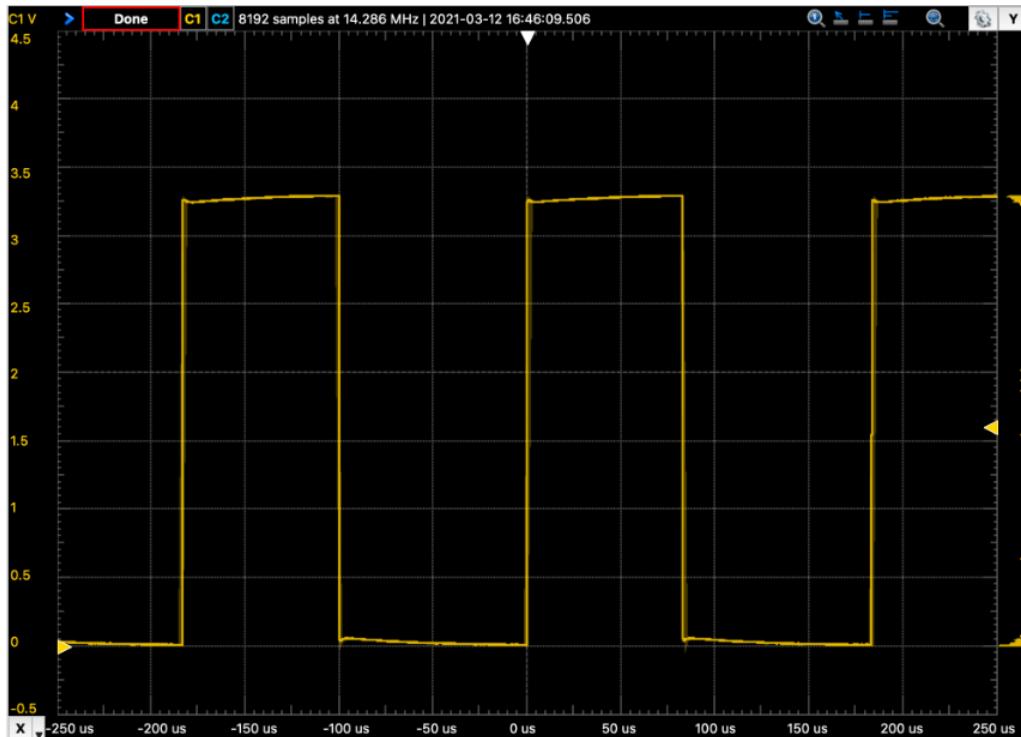
The screenshot shows a terminal window with the following content:

```
p @raspberrypi:~/Documents/Sistemas_embebidos/Proyecto
Archivo Editar Pestañas Ayuda
pi@raspberrypi:~/Documents/Sistemas_embebidos/Proyecto $ . bashgpio.sh
Ejecutando el script .... para detener presione Ctrl + c
[ 1 [ 2 [ 3 [ 4 [ Mem[ Swp[ 39.7%] Tasks: 65, 68 thr; 1 running
30.6%] Load average: 1.15 1.04 0.89
29.0%] Uptime: 00:44:55
28.3%] 153M/924M]
0K/100.0M]

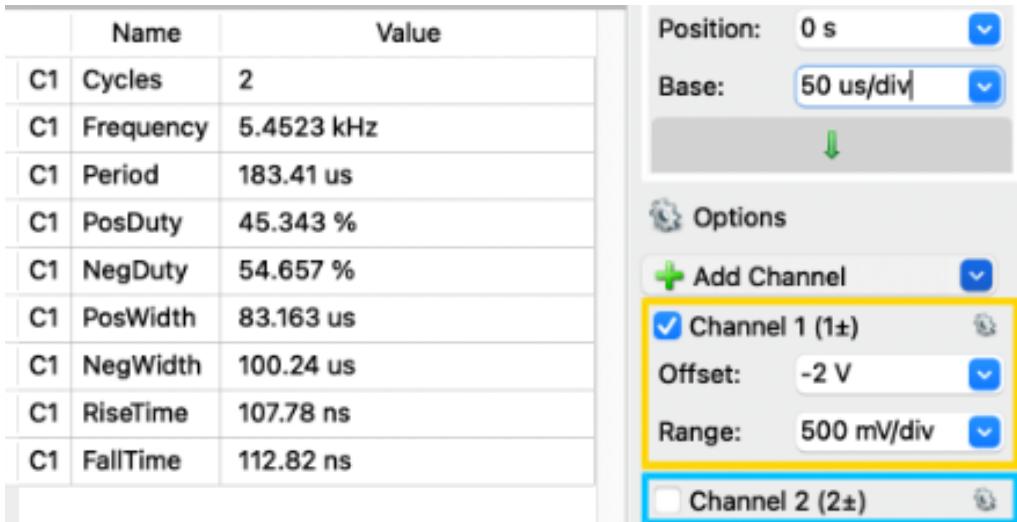
PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM% TIME+  Command
20266 pi        20   0 8516 3908 2940 S 19.0  0.4  0:14.93 bash
11663 pi        20   0 8216 2924 2300 R  1.4  0.3  0:08.57 htop
```

Figura 27: Consumo CPU script Bash.

En cuanto a consumo de CPU se observa una gran mejora con respecto a los otros lenguajes consumiendo en su tarea un 20% de recursos de CPU.



(a) Gráfica



(b) Información de gráfica

Figura 28: Frecuencia de conmutación Bash

En su respuesta de conmutación se encuentra una reducción en la frecuencia equivalente a 5.4 kHz siendo la más baja de los 3 lenguajes y diferentes librerías probadas, con un periodo de 183 us y un ciclo útil negativo del 54.6 % es la solución que más tarda en ejecutar el cambio de un alto a un bajo.

En la siguiente tabla podemos observar los diferentes lenguajes de programación con su respectiva frecuencia

Lenguaje de Programación	Frecuencia
C/C++, Librería Wiring Pi	13.612 Mhz
C/C++, Librería BCM	11.113 Mhz
C/C++, Librería Piggpio	4.6 Mhz
Python	342.37 KHz
Bash	5.4 KHz

Cuadro 1: Frecuencias de trabajo de cada lenguaje de programación utilizado.

7. Conclusiones

Al observar el consumo de CPU para la subida y bajada de un GPIO en la tarjeta Raspberry Pi, nos damos cuenta que en el lenguaje C/C++, con las librerías Wiring Pi, BCM, Piggio y en Python el consumo de CPU es de aproximadamente del 98 % al 100 %, pero cuando analizamos el rendimiento en Bash nos encontramos que tan solo consume en su tarea un 20 %, lo que es realmente significativo y la más optima, en cuanto a consumo de recursos, para elegir la configuración de un GPIO.

Teniendo en cuenta las frecuencias en cada uno de los lenguajes de programación, el lenguaje más rápido es C/C++ con un valor de frecuencia del orden de mega hertz, seguido de Python y por ultimo Bash con ordenes de kilo Hertz.

Esto nos permite concluir:

- Cuando se requiera diseñar en alto nivel o en donde trabajar con periféricos demande un comportamiento rápido del sistema, se puede utilizar el lenguaje C/C++ con las librerías de WiringPi o BCM.
- Cuando se requiera una solución de fácil prototipado y no demande comportamientos tan rápidos, la opción más favorable es Python.
- Finalmente para realizar pruebas rápidas y confiables sobre el código se puede utilizar Bash ya que se puede codificar el mismo programa con pocas líneas de código y su sistema operativo permite supervisar las tareas que se están ejecutando.

8. Anexos

Los scripts ejecutados se encuentran anexos en el siguiente repositorio de github:
https://github.com/Fredycuellar/Proyecto1_Sistemas_Embebidos