UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

FUNDAMENTOS DE SISTEMAS EMBEBIDOS

Proyecto 1. Interfaz física

PROFESOR:	Gerardo Ariel Castillo García
GRUPO DE TEORIA:	3
NÚMERO DE PROYECTO:	1
ALUMNO(S):	Cruz Martínez Raúl
	Mendoza Bolaños Carlos Gabriel
	Villanueva Corona Miguel Angel
SEMESTRE:	2024-2
FECHA DE ENTREGA:	1 de abril de 2024

MATERIAL ELECTRÓNICO:

- 1 potenciómetro mayor a 100 kOhms
- 2 Comparadores encapsulado con 4 unidades
- 8 opto-asilador/acoplador con salida de transistor
- 8 LED rojos o verdes
- 10 resistores 330 Ohms
- 10 resistores 10k Ohms
- 2 protoboard
- 1 batería de 9V
- 1 Rasspberry pi 3B+ (proporcionadas por la Facultad de Ingeniería)
- 1 cargador para tarjeta embebida
- Jumpers

DESARROLLO

El desarrollo este proyecto tiene gran importancia la construcción del cableado físico de la interfaz en donde consta de 4 partes para un mejor manejo y entendimiento permitiendo visualizar de manera correcta el circuito, además de que nos ayudó a comprobar el funcionamiento en partes posteriores así también como detectar alguna posible falla en caso de que se presentaran.

Este circuito involucra componentes electrónicos mencionados anteriormente con la intención de generar, recibir y percibir valores con ayuda de la Raspberry de manera continua para que posteriormente se analice, comprenda e interpreta la información obtenida.

Las etapas realizadas son las siguientes:

- Interfaz física Parte 1

  Para la primera parte se realizó el cableado de los comparadores realizando divisores de voltaje con las resistencias y a su vez colocando leds los cuales nos ayudaron a comprobar la correcta conexión y funcionamiento de cada uno de los comparadores al ser manipulados por el potenciómetro.

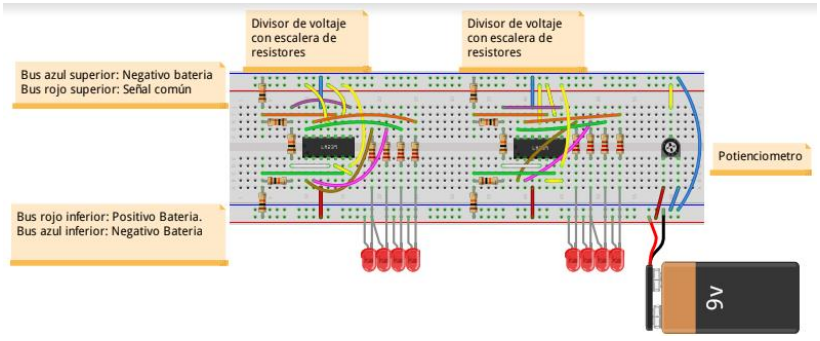


Fig. 1 Diseño Etapa 1

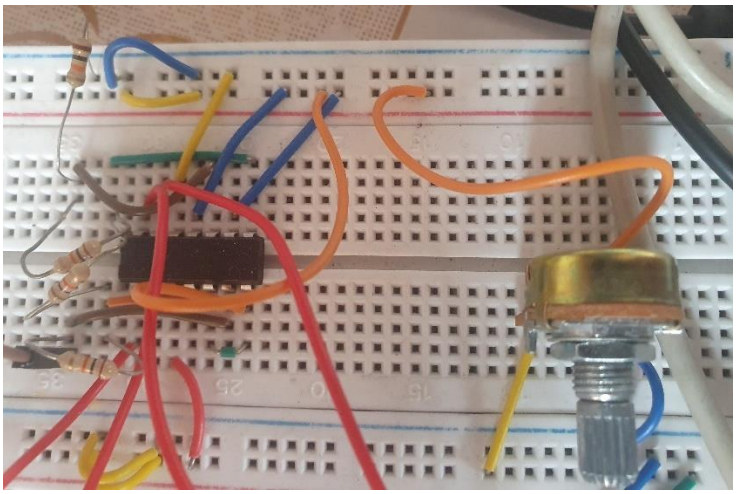
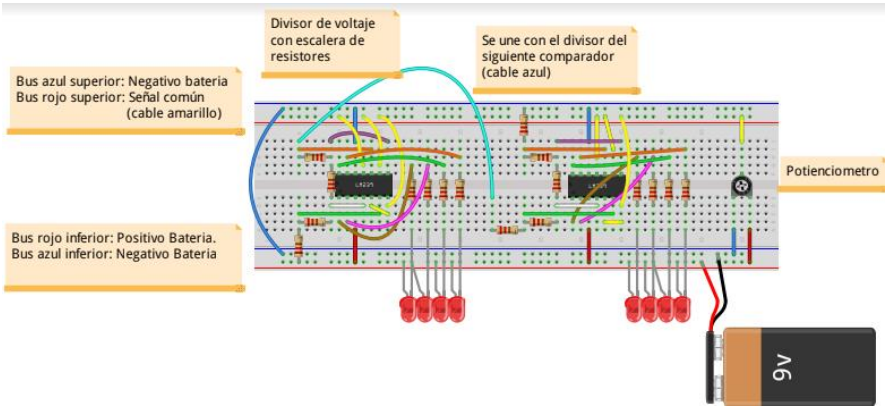


Fig. 2 Alambrado físico Etapa 1

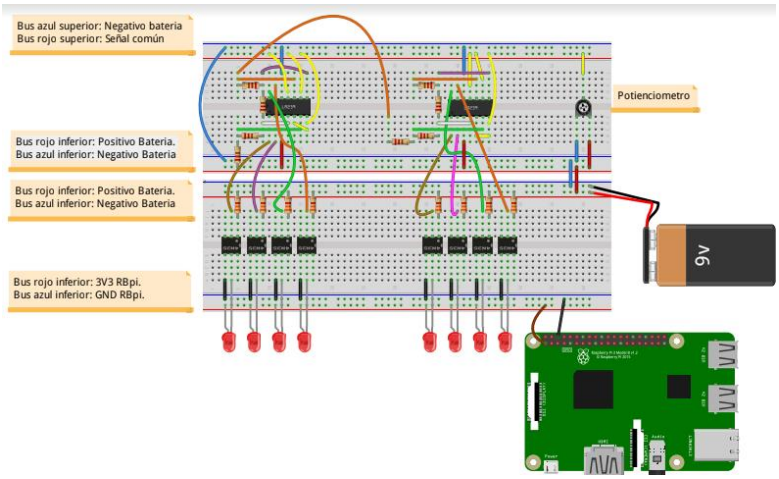
- Interfaz física Parte 2

Posteriormente a la primera etapa colocamos un jumper el cual conectaría ambos comparadores juntando ambas conexiones, de esta manera obtener un comparador de ocho, y conforme se fuera moviendo el potenciómetro se visualiza que los leds iban prendiendo de uno por uno hasta estar todos prendidos, permitiendo seguir con los pasos posteriores, justo al momento de conectar la Raspberry.



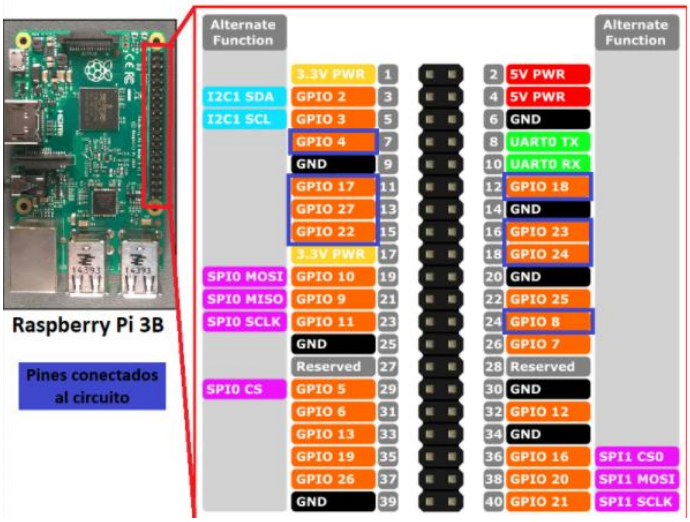
- Interfaz Fisica Parte 3

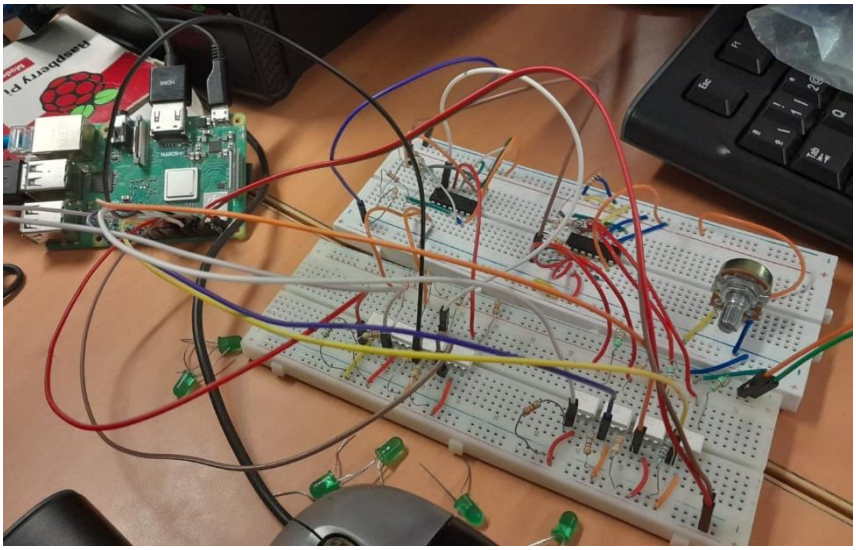
En esta parte se colocaron los optacopladores al circuito para aislar los circuitos realizadas conforme al realizado en conjunto con la Raspberry, verificando en el circuito su correcto funcionamiento de igual manera con respecto al circuito anterior con los leds



- Interfaz física Parte 4

Por último, para terminar con el apartado del hardware se sustituyeron cada uno de los leds por algunos pines de la Raspberry para que con ayuda del software se obtuvieran lecturas de cada uno de los GPIOs y almacenar la información dada. A continuación, se muestra la asignación de los pines de la Raspberry, así como la conexión completa.





Configuración

Como se vio en las imágenes anteriores se usaron diferentes pines de la Raspberry para realizar las lecturas realizando la configuración de la GPIO del 0 al 7 colocando los pines 27, 22, 17, 4, 18, 23, 24 y 8 en ese orden siendo de esta manera el numero 27 como el GPIO_0 y el pin 8 como el GPIO_7. Por lo que al definir los GPIO a usar, mediante uso de diferentes lenguajes de programación se configuraran cada uno para la obtención de tiempos teniendo definido la realización de 100 iteraciones conforme a los 8 pines, con la intención de definir y diferenciar que tipo de lenguaje de programación es eficiente para la obtención de tiempos en menor tiempo como objetivo principal.

Dentro de cada uno de los códigos realizados y usados, tendrán tres tipos de entradas, lo cuales son:

- config. Permite abrir y hacer uso de los GPIO mediante la configuración mencionada.
- valor. Lee los datos obtenidos de cada GPIO involucrado.
- cerrar. Termina de usar los GPIO y cierra los pines para su uso.

Teniendo presente lo anterior, cabe mencionar que cada uno de los lenguajes de programación involucrados generarán sus propios archivos de datos donde tendrá el tiempo calculado conforme a cada iteración realizadas y los valores registrados (tiempo.txt y valores.txt), recordando que serán 100 iteraciones.

El archivo de tiempo nos permitirá calcular con ayuda del programa delta.py, la diferencia entre tiempos, es decir, el tiempo resultante entre cada iteración permitiendo observar de manera grafica el tiempo transcurrido, por lo que se analizará para obtener una mayor visualización del lenguaje con mayor eficiencia.

Los lenguajes a ocupar son los siguientes:

- Shell de Bash
- Python

- C
- C++

A continuación, se muestran cada uno de los códigos realizados para la creación de este proyecto, así como cada de las gráficas generadas.

Cabe resaltar que esta asignación de pines deberá ser hecha en cada uno de los programas a manejar ya que el propósito de este proyecto es identificar cual de todos los lenguajes seleccionados realiza la tarea en menor tiempo y a su vez realizando graficas las cuales nos muestren las lecturas de dichos pines.

- Shell

Con base al código proporcionado en lenguaje Shell, se observa la asignación de GPIO dentro de la codificación, visualizando lo siguiente:

```
# GPIO utilizados para generar numero
GPIO_0=27
GPIO_1=22
GPIO_2=17
GPIO_3=4
GPIO_4=18
GPIO_5=23
GPIO_6=24
GPIO_7=8
```

Fig. 3 Asignación de pines dentro del código de Shell

Así mismo, se muestra la forma de conocer el tipo de entrada que se tendrá al momento de la ejecución del programa, en caso de que dicho parámetro no sea correcto.

```
# En caso que no se pase algun valor
if [ $# -ne 1 ]; then   # si no hay argumento
  echo "No hay comando"
  echo "los comandos a utilizar es config, valor, cerrar"
  exit 2     # Numero invalido de argumentos
fi
```

En caso contrario, que la entrada sea algunas de las aceptadas dentro del código, se ejecutara las siguientes líneas proporcionadas:

Cuando la entrada sea config, se ejecutará lo que se observa en la siguiente imagen, en donde se visualiza la exportación y escritura de las variables definidas como GPIO dentro de cada uno del directorio, asimismo un tiempo de espera en cada uno de los archivos de GPIO involucrados.

```
if [ "$1" == "config" ]; then
        echo "Exportando GPIO numero $GPIO_0"
        echo $GPIO_0 >> "/sys/class/gpio/export"
        sleep 1  # para asegurar que se exporto correctamene
        echo "in" >> "/sys/class/gpio/gpio$GPIO_0/direction"
        echo "Exportando GPIO numero $GPIO_1"
        echo $GPIO_1 >> "/sys/class/gpio/export"
        sleep 1  # para asegurar que se exporto correctamene
        echo "in" >> "/sys/class/gpio/gpio$GPIO_1/direction"
        echo "Exportando GPIO numero $GPIO_2"
        echo $GPIO_2 >> "/sys/class/gpio/export"
        sleep 1  # para asegurar que se exporto correctamene
        echo "in" >> "/sys/class/gpio/gpio$GPIO_2/direction"
        echo "Exportando GPIO numero $GPIO_3"
        echo $GPIO_3 >> "/sys/class/gpio/export"
        sleep 1  # para asegurar que se exporto correctamene
        echo "in" >> "/sys/class/gpio/gpio$GPIO_3/direction"
        echo "Exportando GPIO numero $GPIO_4"
        echo $GPIO_4 >> "/sys/class/gpio/export"
        sleep 1  # para asegurar que se exporto correctamene
        echo "in" >> "/sys/class/gpio/gpio$GPIO_4/direction"
        echo "Exportando GPIO numero $GPIO_5"
        echo $GPIO_5 >> "/sys/class/gpio/export"
        sleep 1  # para asegurar que se exporto correctamene
        echo "in" >> "/sys/class/gpio/gpio$GPIO_5/direction"
        echo "Exportando GPIO numero $GPIO_6"
        echo $GPIO_6 >> "/sys/class/gpio/export"
        sleep 1  # para asegurar que se exporto correctamene
        echo "in" >> "/sys/class/gpio/gpio$GPIO_6/direction"
        echo "Exportando GPIO numero $GPIO_7"
        echo $GPIO_7 >> "/sys/class/gpio/export"
        sleep 1  # para asegurar que se exporto correctamene
        echo "in" >> "/sys/class/gpio/gpio$GPIO_7/direction"

fi
```

Si la entrada es “cerrar”, de igual manera que la entrada config, se exporta e interactúa con el directorio para impedir hacer uso de los GPIOs, haciendo un uso similar al antes mencionado solo que sentido contrario.

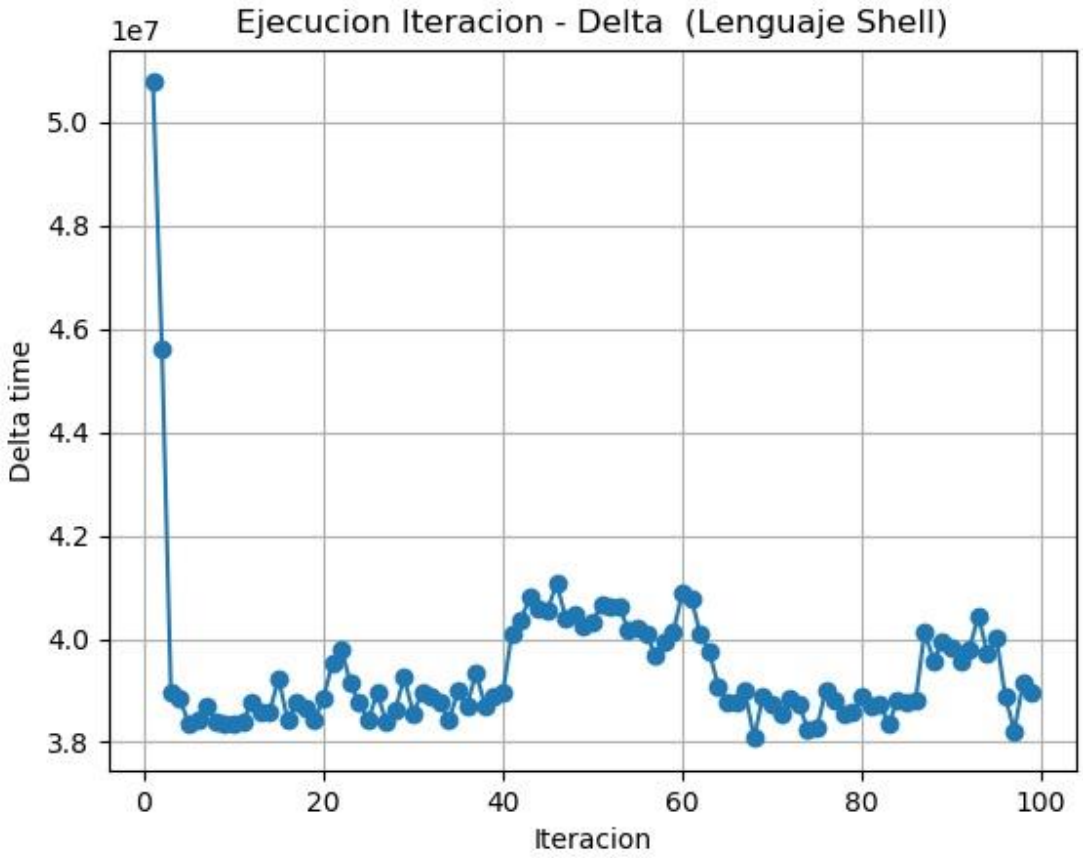
```
if [ "$1" == "cerrar" ]; then
        echo "cerrando el GPIO $GPIO_0"
        echo $GPIO_0 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_1"
        echo $GPIO_1 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_2"
        echo $GPIO_2 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_3"
        echo $GPIO_3 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_4"
        echo $GPIO_4 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_5"
        echo $GPIO_5 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_6"
        echo $GPIO_6 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_7"
        echo $GPIO_7 >> "/sys/class/gpio/unexport"
fi
```

Y si la entrada es “valor”, se realiza las 100 iteraciones con el propósito de recabar información sobre los tiempos de ejecución dentro del programa, por lo que se registrara los tiempos y los valores de cada iteración en archivos, comprobando previamente la existencia del archivo.

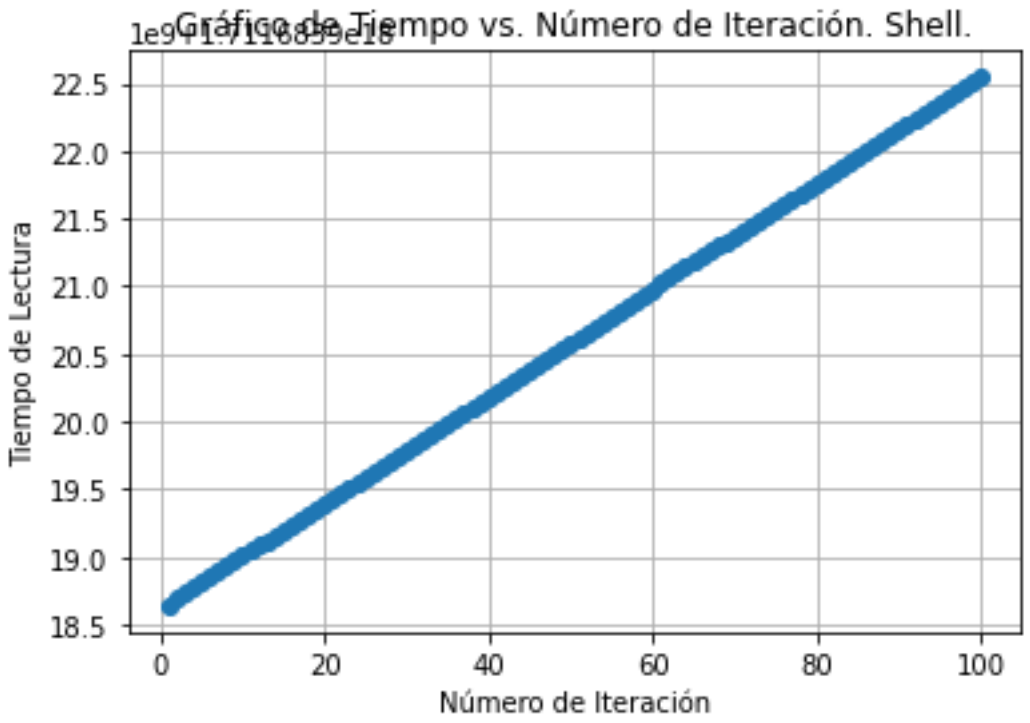
Por lo que al realizarse la iteración se identificará y se asignará la medición de cada GPIO dentro del bucle, haciendo uso de la línea de comando cat dentro del directorio observando en la siguiente imagen.

```
if [ "$1" == "valor" ]; then

        n=100 #Numeros de experimentos
        i=0
        #echo >tiempoShell.txt
        #echo >valorSShell.txt
        rm ./tiempoShell.txt
        rm ./valorShell.txt
        rm ./iteracion.txt
        while((i<n))
        do
                bit0=$(cat "/sys/class/gpio/gpio$GPIO_0/value")
                bit1=$(cat "/sys/class/gpio/gpio$GPIO_1/value")
                bit2=$(cat "/sys/class/gpio/gpio$GPIO_2/value")
                bit3=$(cat "/sys/class/gpio/gpio$GPIO_3/value")
                bit4=$(cat "/sys/class/gpio/gpio$GPIO_4/value")
                bit5=$(cat "/sys/class/gpio/gpio$GPIO_5/value")
                bit6=$(cat "/sys/class/gpio/gpio$GPIO_6/value")
                bit7=$(cat "/sys/class/gpio/gpio$GPIO_7/value")
                #bit1=3.3
                let numero=bit0+bit1+bit2+bit3+bit4+bit5+bit6+bit7
                let i=i+1
                t=$(date +%s%N)
                #Visualizacion de cada valor individual
                echo $bit0
                echo $bit1
                echo $bit2
                echo $bit3
                echo $bit4
                echo $bit5
                echo $bit6
                echo $bit7
                echo $i
                echo $t>>tiempoShell.txt
                echo $numero>>valorShell.txt
                echo $i>>iteracion.txt
        done
fi
```

Grafica entre Delta tiempo vs las iteraciones.



Gráfica entre el tiempo de lectura vs. la iteración.



- Python

Con respecto al lenguaje de programación Python, primeramente, se realizó la importación de bibliotecas para poder manipular el tiempo y las entradas proporcionadas en el teclado al momento de la ejecución del programa. Asimismo, definir mediante variables la asignación de los GPIOs involucrados.

```
import os
import time

# GPIO utilizados para generar numero
GPIO_0 = 27
GPIO_1 = 22
GPIO_2 = 17
GPIO_3 = 4
GPIO_4 = 18
GPIO_5 = 23
GPIO_6 = 24
GPIO_7 = 8
```

Posteriormente se debe verificar la entrada dada, ya que solamente es posible aceptar “config”, “valor” y “cerrar”, verificando la longitud del texto dado.

```
# Verificar si se pasó algún valor
if len(os.sys.argv) != 2:
    print("No hay comando")
    print("Los comandos a utilizar son config, valor, cerrar")
    os.sys.exit(2)  # Número inválido de argumentos
```

Si la entrada es “config”, se iterará dentro de una lista las GPIOs definidas para poder realizar la exportación y uso de cada uno de los pines, por lo que se creó y definió funciones para poder realizar un código más completo y eficiente, donde las funciones creadas fueron export_gpio y set_direction.

```
# Configurar GPIO como entradas
if os.sys.argv[1] == "config":
    for gpio in [GPIO_0, GPIO_1, GPIO_2, GPIO_3, GPIO_4, GPIO_5, GPIO_6, GPIO_7]:
        print(f"Exportando GPIO número {gpio}")
        export_gpio(gpio)
        time.sleep(1)  # para asegurar que se exportó correctamente
        print(f"Configurando GPIO número {gpio} como entrada")
        set_direction(gpio, "in")
```

En esta función export_gpio permite verificar si el directorio se encuentra abierto y creado, por lo que permitirá crear, si no existe, y modificar el directorio, por lo que realizará la escritura de cada uno de los GPIOs usados.

```
# Función para exportar un GPIO
def export_gpio(gpio):
    with open(f"/sys/class/gpio/export", 'w') as file:
        file.write(str(gpio))
```

En la función set_direction verificará y realizará la dirección de cada carpeta dentro del directorio, permitiendo ubicar y crear cada uno de los GPIOs a usar.

```
# Función para configurar la dirección de un GPIO
def set_direction(gpio, direction):
    with open(f"/sys/class/gpio/gpio{gpio}/direction", 'w') as file:
        file.write(direction)
```

Si la entrada es “cerrar”, iterará por la lista de los GPIOs a usar para que haga uso de la función unexport_gpio por cada GPIO.

```
# Cerrar GPIO
elif os.sys.argv[1] == "cerrar":
    for gpio in [GPIO_0, GPIO_1, GPIO_2, GPIO_3, GPIO_4, GPIO_5, GPIO_6, GPIO_7]:
        print(f"Cerrando el GPIO {gpio}")
        unexport_gpio(gpio)
```

Esta función permite escribir, desexportar y manipular el directorio de los GPIOs para poder inhabilitar los GPIOs para su uso.

```
# Función para desexportar un GPIO
def unexport_gpio(gpio):
    with open(f"/sys/class/gpio/unexport", 'w') as file:
        file.write(str(gpio))
```

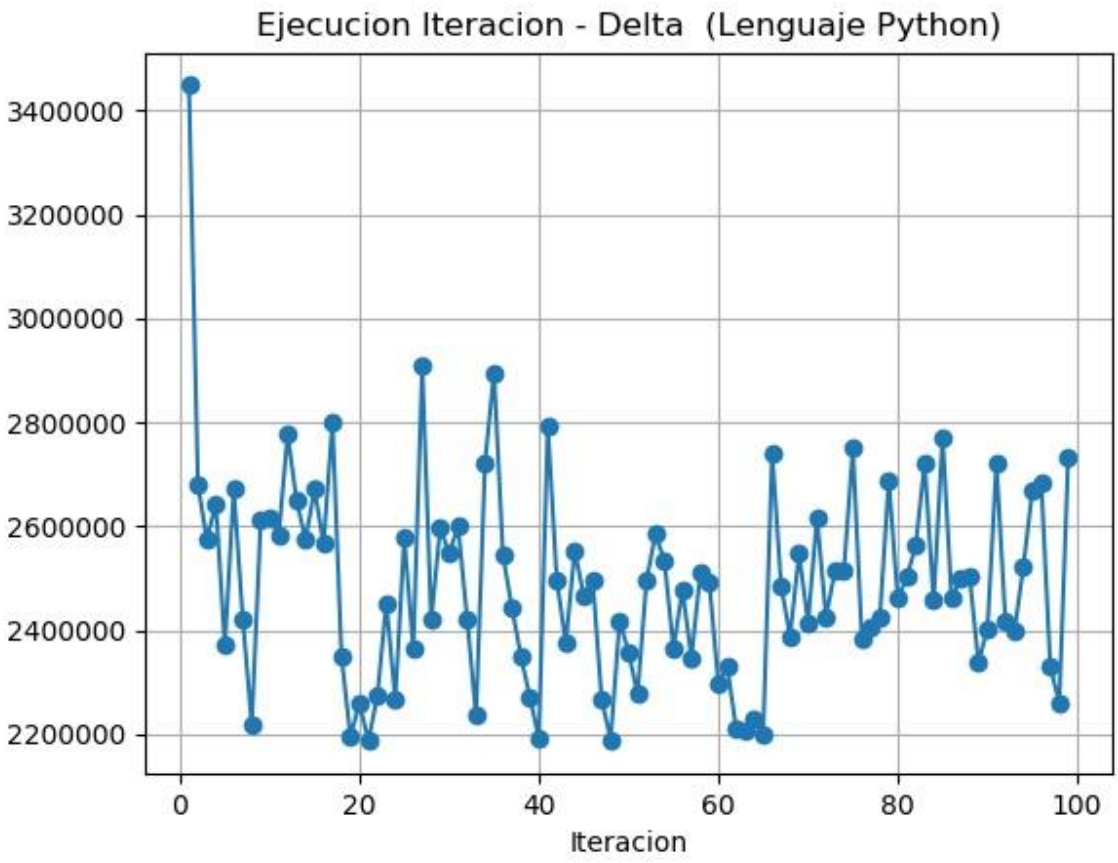
Y si la entrada llega ser “valor” realizará la creación de archivos donde contendrá los valores dados respecto a los tiempos y los valores dado en cada iteración. Para posteriormente, realizar las 100 iteraciones con el respectivo tiempo conforme a cada GPIO, manipulando los archivos para escribir lo solicitado.

```
# Leer valor de los GPIO
elif os.sys.argv[1] == "valor":
    n = 100  # Números de experimentos
    with open("tiempoPy.txt", 'w') as tiempo_file:
         tiempo_file.write("")
    with open("valorPy.txt", 'w') as valor_file:
          valor_file.write("")
    for i in range(n):
        bits = [read_gpio(gpio) for gpio in [GPIO_0, GPIO_1, GPIO_2, GPIO_3, GPIO_4, GPIO_5, GPIO_6, GPIO_7]]
        numero = sum(int(bit) for bit in bits)
        t = time.time_ns()
        # Visualización de cada valor individual
        for bit in bits:
            print(bit)
        print(i)
        with open("tiempoPy.txt", 'a') as tiempo_file:
            tiempo_file.write(f"{t}\n")
        with open("valorPy.txt", 'a') as valor_file:
            valor_file.write(f"{numero}\n")
```

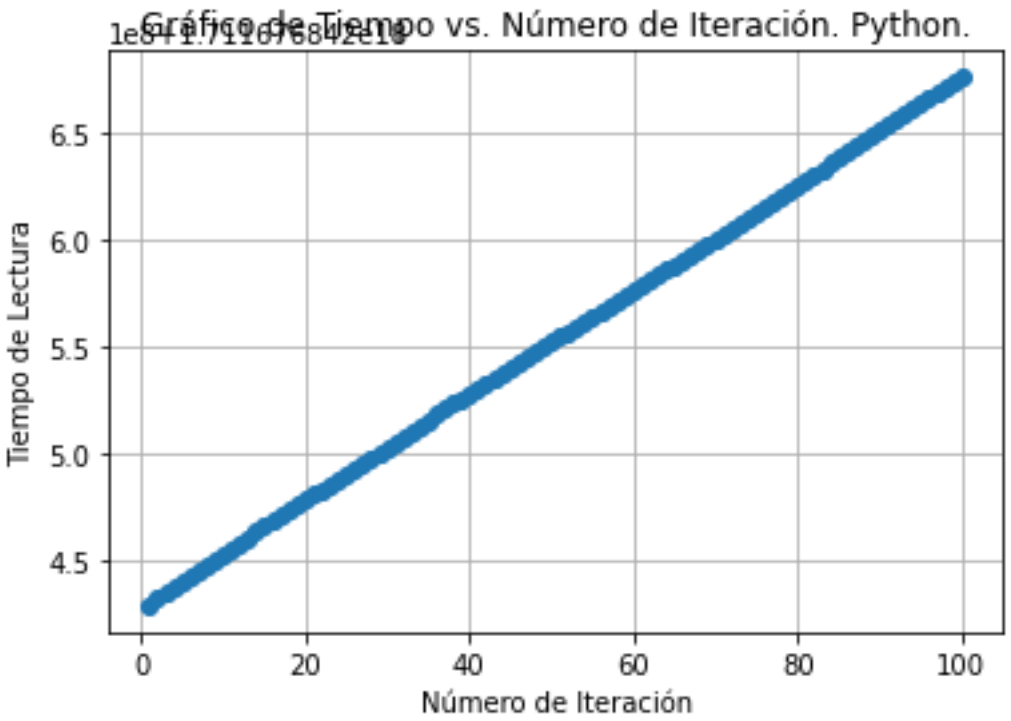
La función read_gpio dentro del elif de la entrada “valor” nos da la posibilidad de leer lo que registra cada pin de la Raspberry, dando la característica del flujo del voltaje y uso dentro del circuito, permitiendo registrar las lecturas deseadas.

```
# Función para leer el valor de un GPIO
def read_gpio(gpio):
    with open(f"/sys/class/gpio/gpio{gpio}/value", 'r') as file:
        return file.read().strip()
```

Grafica entre Delta tiempo vs las iteraciones.



Gráfica entre el tiempo de lectura vs. la iteración.



- C

Respecto al lenguaje de programación C, primeramente, se empieza con la declaración de las bibliotecas a usar durante el desarrollo de la codificación y asimismo, definir las variables respecto a los GPIOs a usar para poder manipular de manera eficiente dentro del código.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>

#define GPIO_0 27
#define GPIO_1 22
#define GPIO_2 17
#define GPIO_3 4
#define GPIO_4 18
#define GPIO_5 23
#define GPIO_6 24
#define GPIO_7 8
```

Posteriormente en la función main, se identificará la entrada proporcionada como entrada para manipular el flujo del código, por lo que, si no llega ser la entrada “config”, “valor” o “cerrar”, entonces se dará mensaje de error y notificando las entradas admitidas.

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("No hay comando\n");
        printf("Los comandos a utilizar son: config, valor, cerrar\n");
        exit(EXIT_FAILURE);
    }


} else {
    printf("Comando no reconocido\n");
    printf("Los comandos a utilizar son: config, valor, cerrar\n");
    exit(EXIT_FAILURE);
}
```

Si la entrada es “config” se realiza la llamada de la función export_gpio con cada parámetro respectivo, donde dicho parámetro es cada GPIO definido previamente.

```
if (strcmp(argv[1], "config") == 0) {
    export_gpio(GPIO_0);
    export_gpio(GPIO_1);
    export_gpio(GPIO_2);
    export_gpio(GPIO_3);
    export_gpio(GPIO_4);
    export_gpio(GPIO_5);
    export_gpio(GPIO_6);
    export_gpio(GPIO_7);
```

Al ser uso de la función export_gpio, esta función realizará un archivo con el directorio definido y escribirá en dicho archivo creado el pin a usar para la ejecución de la codificación, así como la ubicación donde se encontrará la asignación de cada uno de los GPIO para hacer uso de estos.

```c
void export_gpio(int gpio_num) {
    FILE *export_file;
    export_file = fopen("/sys/class/gpio/export", "w");
    if (export_file == NULL) {
        perror("Error al abrir el archivo de exportación");
        exit(EXIT_FAILURE);
    }
    fprintf(export_file, "%d\n", gpio_num);
    fclose(export_file);

    char direction_path[100];
    sprintf(direction_path, "/sys/class/gpio/gpio%d/direction", gpio_num);
    FILE *direction_file = fopen(direction_path, "w");
    if (direction_file == NULL) {
        perror("Error al abrir el archivo de dirección");
        exit(EXIT_FAILURE);
    }
    fprintf(direction_file, "in\n");
    fclose(direction_file);
}
```

En caso de que la entrada sea “cerrar”, se realizará la llamada de la función unexport_gpio con el parámetro de cada GPIO asignados anteriormente.

```c
else if (strcmp(argv[1], "cerrar") == 0) {
  unexport_gpio(GPIO_0);
  unexport_gpio(GPIO_1);
  unexport_gpio(GPIO_2);
  unexport_gpio(GPIO_3);
  unexport_gpio(GPIO_4);
  unexport_gpio(GPIO_5);
  unexport_gpio(GPIO_6);
  unexport_gpio(GPIO_7);
```

En la función unexport_gpio nos ubicaremos en la dirección definida para poder escribir en ella y poder inhabilitar el uso de los GPIO, definido dentro del directorio de unexport.

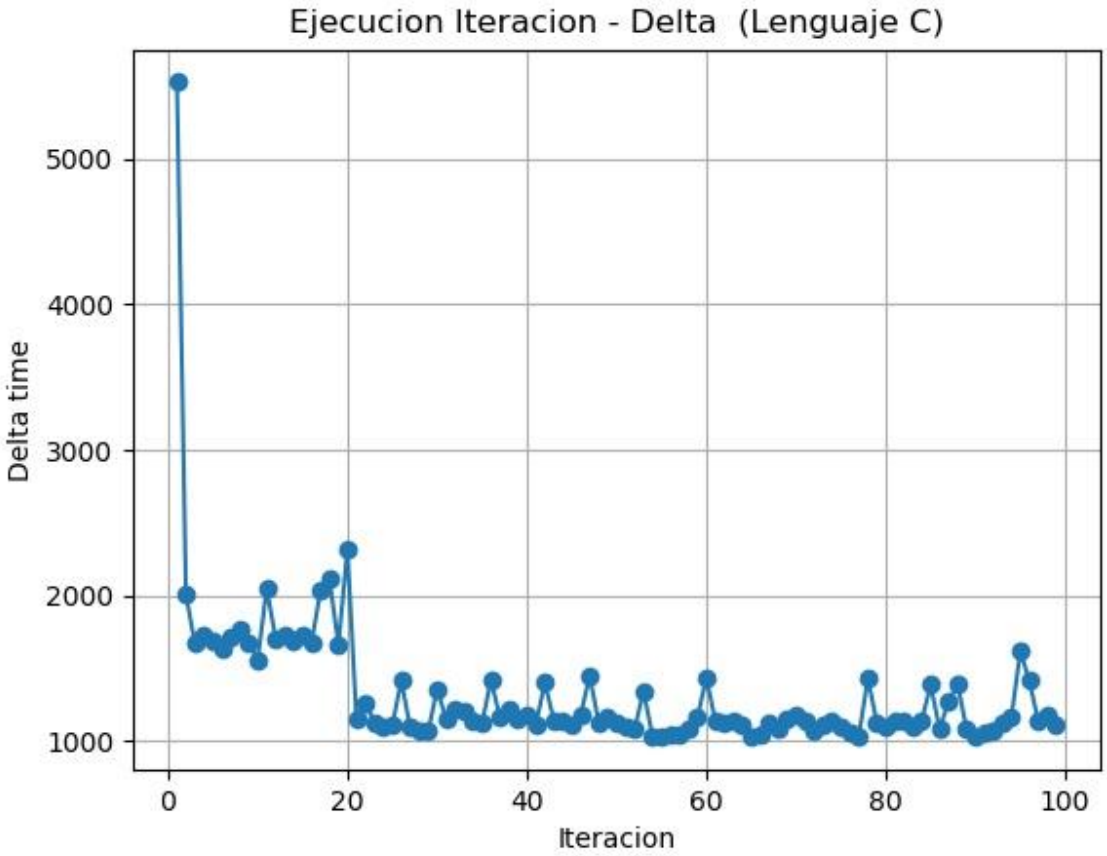
```c
void unexport_gpio(int gpio_num) {
    FILE *unexport_file;
    unexport_file = fopen("/sys/class/gpio/unexport", "w");
    if (unexport_file == NULL) {
        perror("Error al abrir el archivo de unexport");
        exit(EXIT_FAILURE);
    }
    fprintf(unexport_file, "%d\n", gpio_num);
    fclose(unexport_file);
}
```

Y si la entrada es “valor” se define un arreglo de los GPIO a usar para el uso de un ciclo for para sobrescribir dentro del directorio para la creación de cada carpeta de la GPIO y poder contener en ella las lecturas necesarias para el análisis. De la misma manera la creación y escritura de loa archivos tiempoC.txt y valorC.txt para el almacenamiento de los datos necesarios, agregando la toma del tiempo durante la ejecución, ya es necesario en conjunto de bibliotecas para poder realizar las tareas de medición dentro de la ejecución del ciclo de “valor”.

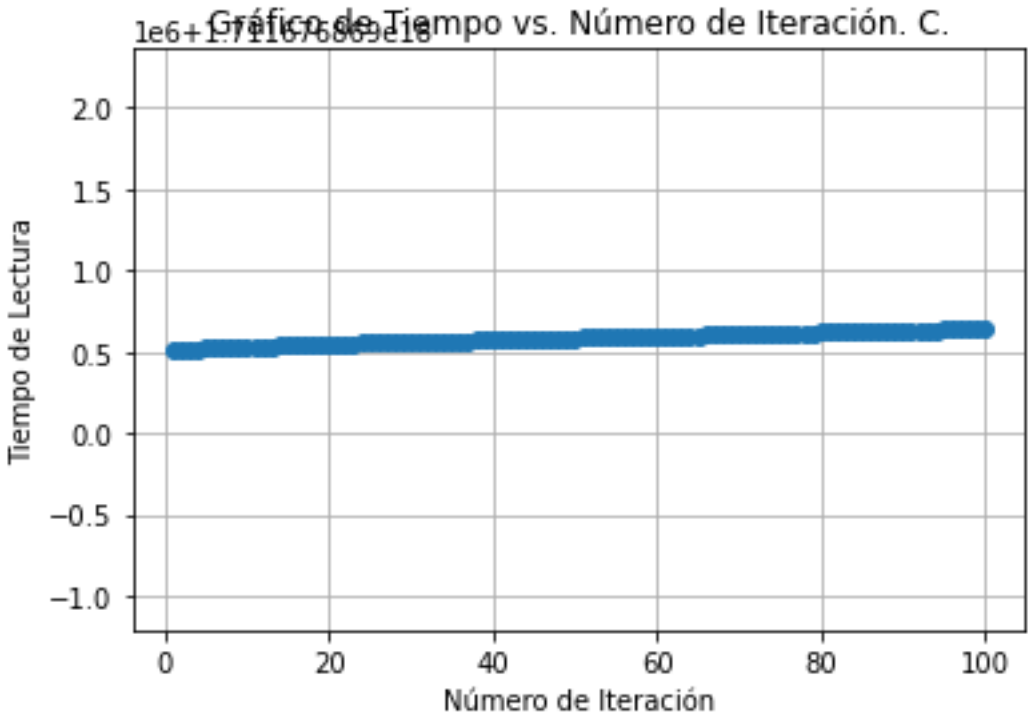
```c
    } else if (strcmp(argv[1], "valor") == 0) {
        int n = 100;
        int i;
        int GPIO[8]={27,22,17,4,18,23,24,8};
        FILE *tiempo_file = fopen("tiempoC.txt", "w");
        fclose(tiempo_file);
        FILE *valor_file = fopen("valorC.txt", "w");
        fclose(valor_file);


for (i = 0; i < n; i++) {
    FILE *gpio_files[8];
    int bit_values[8];
    char gpio_value_path[100];
    int j;

    for (j = 0; j < 8; j++) {
        sprintf(gpio_value_path, "/sys/class/gpio/gpio%d/value", GPIO[j]);
        //printf("%s\n",gpio_value_path);
        gpio_files[j] = fopen(gpio_value_path, "r");
        if (gpio_files[j] == NULL) {
            perror("Error al abrir el archivo GPIO value");
            exit(EXIT_FAILURE);
        }
        fscanf(gpio_files[j], "%d", &bit_values[j]);
        fclose(gpio_files[j]);
    }
```

```
int numero = 0;
for (j = 0; j < 8; j++) {
    numero += bit_values[j];
}

// Visualización de cada valor individual
for (j = 0; j < 8; j++) {
    printf("%d\n", bit_values[j]);
}
printf("%d\n", i);

struct timeval tv;
gettimeofday(&tv,NULL);

// Guardar tiempo actual en tiempo.txt
long long int t = tv.tv_sec*(long long int) 1000000000+tv.tv_usec;
FILE *tiempo_file = fopen("tiempoC.txt", "a");
if (tiempo_file == NULL) {
    perror("Error al abrir el archivo tiempo.txt");
    exit(EXIT_FAILURE);
}
fprintf(tiempo_file, "%lld\n", t);
fclose(tiempo_file);

// Guardar el número en valor.txt
FILE *valor_file = fopen("valorC.txt", "a");
if (valor_file == NULL) {
    perror("Error al abrir el archivo valor.txt");
    exit(EXIT_FAILURE);
}
fprintf(valor_file, "%d\n", numero);
fclose(valor_file);
```

Grafica entre Delta tiempo vs las iteraciones.



Gráfica entre el tiempo de lectura vs. la iteración.



- C++

Para la programación en lenguaje C++ se inicia con la definición de las bibliotecas a usar, por lo que se observa a continuación una imagen de las usadas dentro del código.

```
#include <fstream>
#include <iostream>
#include <string>
#include <chrono>
#include <thread>
#include <vector>
#include <unistd.h> // Para la función sleep
```

En la función main se verifica el tipo de entrada proporcionada, por lo que se verificará lo ingresado y posteriormente, definiendo los pines a usar durante la ejecución del código.

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        std::cerr << "Uso: " << argv[0] << " [config|cerrar|valor]" << std::endl;
        return 1;
    }

    std::string command = argv[1];
    const std::vector<int> gpios = {27, 22, 17, 4, 18, 23, 24,8};
```

Cuando la entrada dada es “config” se realiza un ciclo for para iterar por el arreglo definido previamente y poder hacer uso de la función exportAndConfigureGPIO para el uso de los GPIO.

```
    if (command == "config") {
        //const int gpios[] = {27, 22, 17, 4, 18, 23, 24, 25};
        for (int gpio : gpios) {
            std::cout << "Exportando GPIO número " << gpio << std::endl;
            exportAndConfigureGPIO(gpio);
        }
    }
```

Dentro de la función exportAndConfigureGPIO se manipulará el directorio dado para poder hacer uso de los GPIO y poder escribir en ellos para la lectura de los datos dado en cada GPIO.

```
// Función para exportar un GPIO y configurarlo como entrada
void exportAndConfigureGPIO(int gpio) {
    std::ofstream file;
    file.open("/sys/class/gpio/export");
    if (file.is_open()) {
        file << gpio;
        file.close();
        sleep(1); // Esperar 1 segundo para asegurar que se exportó correctamente
        file.open("/sys/class/gpio/gpio" + std::to_string(gpio) + "/direction");
        if (file.is_open()) {
            file << "in";
            file.close();
        } else {
            std::cerr << "No se pudo configurar la dirección del GPIO número " << gpio << std::endl;
        }
    } else {
        std::cerr << "No se pudo exportar el GPIO número " << gpio << std::endl;
    }
```

Si llegara ser que la entrada sea “cerrar”, entonces ejecutará un ciclo for para poder iterar sobre cada una de las GPIO dentro de la lista definida previamente y así realizar la llamada de la función unexportGPIO.

```
if (command == "cerrar") {
    for (int gpio : gpios) {
        std::cout << "Cerrando el GPIO " << gpio << std::endl;
        unexportGPIO(gpio);
    }
```

La función unexportGPIO comprueba el directorio dado para escribir en él para poder declarar que dichos pines usadas ya no serán usados por lo que serán inhabilitados para la manipulación de los GPIOS.

```
// Función para desexportar un GPIO
void unexportGPIO(int gpio) {
    std::ofstream file;
    file.open("/sys/class/gpio/unexport");
    if (file.is_open()) {
        file << gpio;
        file.close();
    } else {
        std::cerr << "No se pudo cerrar el GPIO número " << gpio << std::endl;
    }
}
```

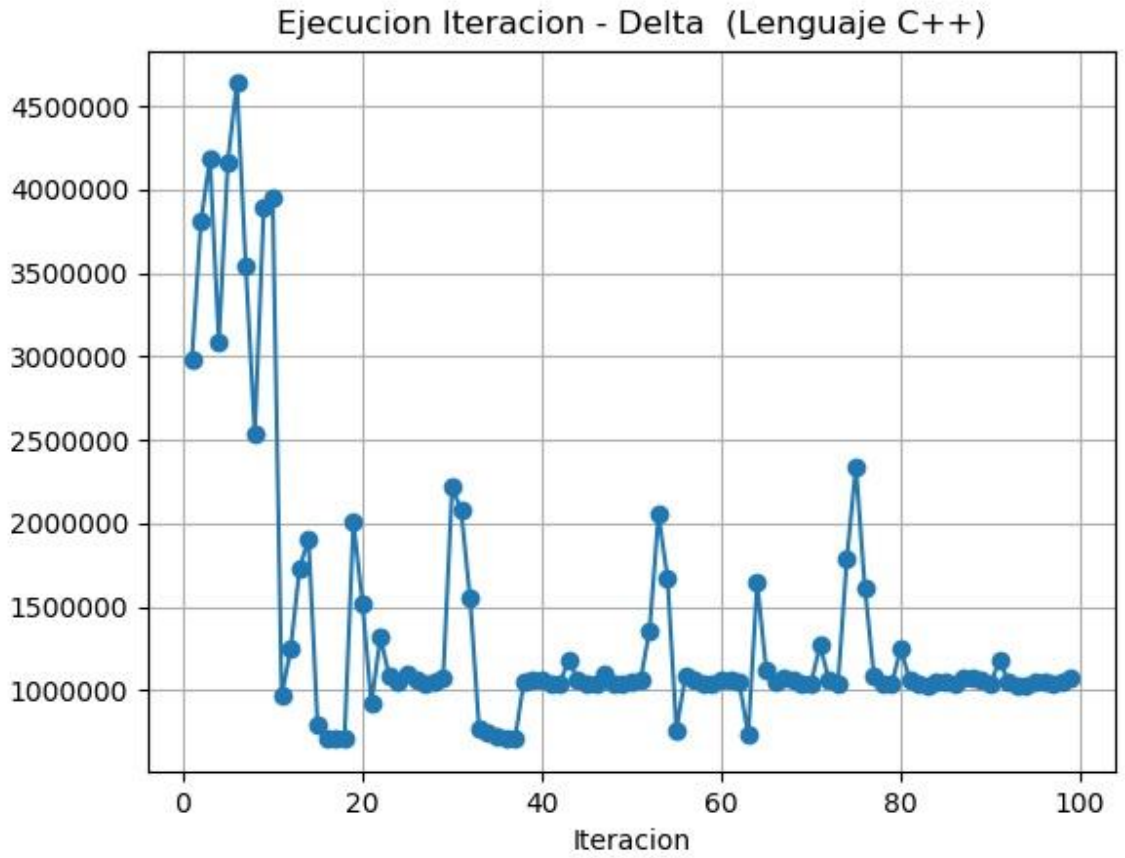
Si la entrada es “valor” se realizar una serie de línea donde creará los archivos de texto donde contendrá los datos leídos sobre los tiempos correspondiente de ejecución como los valores dado por cada GPIO, posteriormente se iterará 100 veces para realizar la ejecución de lecturas con ayuda de la función readGPIO.

```
} else if (command == "valor") {
    const int n = 101; // Número de experimentos
    std::ofstream tiempo_file("tiempoCm.txt", std::ofstream::out|std::ofstream::trunc);
    std::ofstream valor_file("valorCm.txt", std::ofstream::out|std::ofstream::trunc);

    for (int i = 0; i < n; ++i) {
        int numero = 0;
        for (int gpio : gpios) {
            int bit = readGPIO(gpio);
            std::cout << bit << std::endl;
            numero += bit;
        }

        auto t = std::chrono::high_resolution_clock::now();
        auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(t.time_since_epoch()).count();

        std::cout << i << std::endl;
        tiempo_file.open("tiempoCm.txt",std::ofstream::out|std::ofstream::app);
        valor_file.open("valorCm.txt",std::ofstream::out|std::ofstream::app);
        if (tiempo_file.is_open() && valor_file.is_open()) {
            tiempo_file << duration << std::endl;
            valor_file << numero << std::endl;
        }

        tiempo_file.close();
        valor_file.close();
    }
```

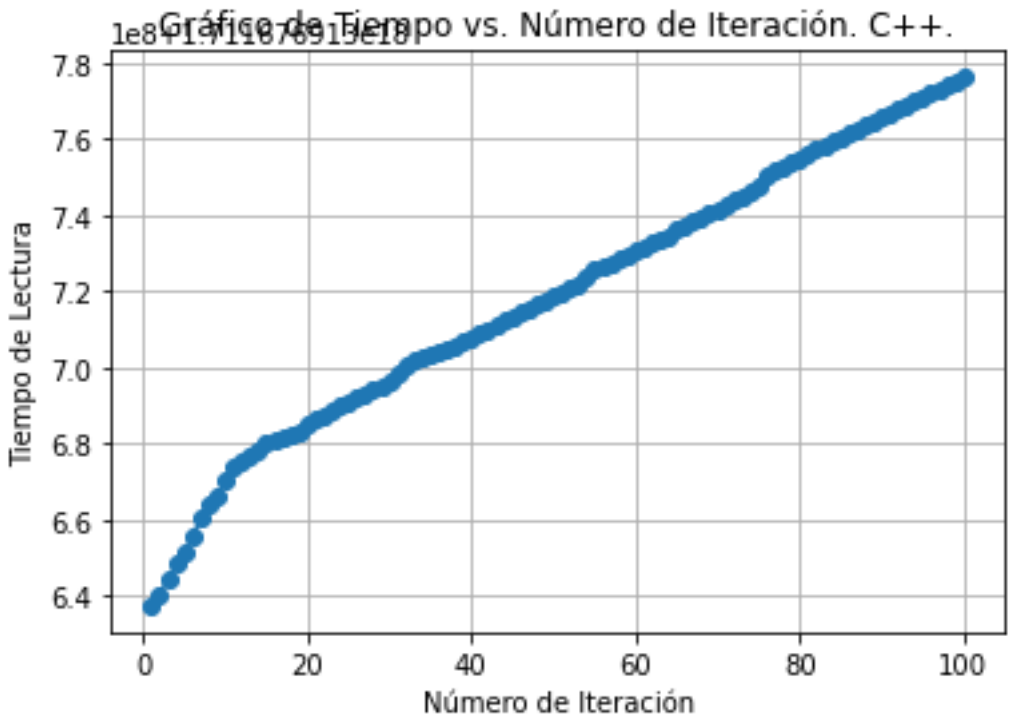
Esta función readGPIO nos da la posibilidad de posicionar cada GPIO a utilizar dentro del directorio y así poder realizar las lecturas deseada y escribir dentro de cada pin sobre su ubicación.

```
// Función para leer el valor de un GPIO
int readGPIO(int gpio) {
    std::ifstream file;
    file.open("/sys/class/gpio/gpio" + std::to_string(gpio) + "/value");
    if (file.is_open()) {
        int value;
        file >> value;
        file.close();
        return value;
    } else {
        std::cerr << "No se pudo leer el valor del GPIO número " << gpio << std::endl;
        return -1;
    }
}
```

Grafica entre Delta tiempo vs las iteraciones.



Gráfica entre el tiempo de lectura vs. la iteración.



Conclusión.

Cruz Martínez Raúl

El desarrollo de este proyecto, contó de dos partes principales: la construcción de una interfaz física que vimos representada con un circuito compuesto por comparadores, optoacopladores y LEDs, y la segunda fue la comunicación de esta interfaz física con la RaspberryPi a través del Convertidor Analógico Digital, el cual discretizaba los valores continuos de voltaje proporcionado por la interfaz.

En la primera parte, gracias a que los diagramas estaban bien representados, no existió problema alguno para la construcción de la interfaz; de la misma manera, conectar el circuito a los GPIO de la RaspberryPi no tuvo complicación, de modo que al hacerlo pudimos ejecutar las primeras pruebas del DAC con el código de Shell.

La siguiente parte del proyecto fue la de realizar las lecturas de la interfaz con 4 lenguajes de programación diferentes: programación en Shell, Python, C++ y C, este último como lenguaje propuesto. Con Shell, Python y C++ no hubo problema alguno, pues bastó con instalar las bibliotecas y dependencias necesarias para cada lenguaje. Para el lenguaje propuesto, inicialmente se tenía contemplado que fuera Fortran 90, pero, al trabajar dicho lenguaje, nos percatamos de que no existen las dependencias necesarias para poder usar este lenguaje de programación para el fin que necesitábamos, esto nos llevó a cambiar al lenguaje C, en donde no existieron problemas para su utilización, principalmente porque al trabajar con Linux, que es un Sistema Operativo construido en C, ya contenía la mayoría de bibliotecas y dependencias necesarias.

La parte del análisis del desempeño, fue la que requirió más tiempo, ya que adaptar el funcionamiento del código de Shell a otros lenguajes de programación, implicó entender bien el funcionamiento de cada lenguaje. De forma general, observamos que el lenguaje que presenta el mejor desempeño es el lenguaje C, ya que, el tiempo transcurrido entre cada iteración (delta) es menor y también mantiene un ritmo o frecuencia constante después de las 20 iteraciones, en cuanto a C++, se tiene la segunda mejor optimización, aunque en cada iteración tiene un cambio relativamente constante y predecible, cada 20 iteraciones se observa un pico o retraso en la lectura; finalmente, Python y Shell tienen variaciones más notables, así como una demora mayor entre cada lectura, lo que hace que sus gráficas sean más desordenadas y poco predecibles.

Para finalizar, he de decir que la elaboración de este proyecto, fue de mucha utilidad para observar cómo es posible la interconexión de sistemas, pero, la parte más importante fue la de observar los factores que intervienen en la obtención de la información, donde lo más destacable fue lo siguiente: elegir un lenguaje de programación más óptimo es una parte crucial para la realización de tareas repetitivas, como lo es el lenguaje C, pero, no sólo de este elemento depende el desempeño del sistema, sino de otros factores como la temperatura de la RaspberryPi. Dicho lo anterior, considero que se han cumplido los objetivos del presente proyecto.

Mendoza Bolaños Carlos Gabriel.

Para el desarrollo de este proyecto nos permitió observar la complejidad que puede llegar ser un sistema para realizar diversas tareas en donde el tiempo, la eficiencia y el resultado es primordial dentro del mercado laboral, por lo que al estar en contacto en sistemas y sus configuraciones permiten observar los diversos factores que se deben tomar en cuenta para la elaboración y mantenimiento de un sistema embebido.

Por lo que este proyecto con base a la codificación en Shell nos dio la posibilidad de interactuar y manipular GPIO de una tarjeta de desarrollo como es la Raspberry PI 3, donde se configuro para hacer uso de sus componentes e incorporar circuitos para la realización de las lecturas. Cabe mencionar que no se encontró algún problema en la construcción ni el armado del circuito, teniendo con el éxito los valores y funcionamiento de los leds, comparadores y optoacopladores, por lo que se desarrollo de manera continua la función y los resultados deseados.

Con respecto al análisis y comparación dada, cabe resalta que primeramente se tenia pensado usar el lenguaje de programación Fortran, sin embargo, no obtuvimos un resultado éxito ni manipulación del código para la interacción de los GPIOs, por lo que se deseo cambiar por el lenguaje C por la facilidad de manipulación y mayor conocimiento del lenguaje. Al definir los lenguajes de programación a usar, como fueron Shell, Python, C y C++ nos percatamos que cada uno al desarrollar la codificación presente cierta complejidad y relación en la formación de las funciones con respecto al main, sin embargo, se posee conocimientos y así mismo con ayuda de instigación se formuló cada uno de los códigos con base al proporcionado en Shell, por lo que se presentaron algunos inconvenientes con la agregación algunas funciones para facilitar el flujo del código.

En general, al analizar y comprender los resultados dados durante 100 iteraciones se visualiza que el resultado mas optimo y eficiente es el lenguaje C teniendo unas diferencias mínimas en cada iteración y además de ser aquel lenguaje con una ejecución más rápido con la misma tarea en todos los lenguajes involucrados. Después el lenguaje más constante es el lenguaje C++, sin embargo, se visualiza al inicializar la ejecución le toma más tiempo, pero con el pasar de las iteraciones toma un valor más constante y mínima diferencia de tiempos; donde los lenguajes C++ y C son un tipo de lenguaje de programación compilado, en donde verifica todo el programa para posteriormente construir un ejecutable con el contenido del código.

En caso contrario del lenguaje Python, se observa que requiere menor tiempo de inicialización, pero al pasar de las iteraciones este es muy variable y no posee algún constante que puede interpretar un patrón, además de que sus diferencias de tiempo ente iteración son mayores. Y con respecto al lenguaje Shell requiere mayor tiempo para la ejecución de las lecturas y mayor distanciamiento de tiempo entre interacción, siendo el menor eficiente de los cuatro lenguajes. Estos lenguajes son de tipo interpretados, por lo que nos dice que se va ejecutando conforme se va desarrollando cada línea del código.

Todo este desarrollo y análisis nos da la entrada a la automatización y simplificación de tareas de nuestra vida cotidiana, así como en la implementación y creación de nuevas tecnologías en productos como en la realización de tareas más complejas.

Villanueva Corona Miguel Angel

En el transcurso de este proyecto, hemos tenido la oportunidad de sumergirnos profundamente en lo que implica la construcción de un sistema embebido. Nuestro trabajo comenzó con la fase de hardware, donde minuciosamente ensamblamos un circuito utilizando los componentes especificados. Este meticuloso proceso fue esencial, ya que estableció la base para que, al suministrar una tensión específica mediante una batería, la Raspberry Pi pudiera interpretar con precisión las señales de voltaje en sus pines. Esto fue esencial para avanzar hacia la etapa de programación, donde nuestra tarea era determinar cuál entre los distintos lenguajes de programación disponibles se alineaba óptimamente con nuestras metas y requisitos.

La decisión sobre qué lenguaje de programación adoptar para este proyecto no fue a simple vista sencillo. Requirió una evaluación detallada de las capacidades de procesamiento y la eficiencia en el manejo de los recursos, donde C y C++ destacaron por su rendimiento superior. Estos lenguajes ofrecen un control granular sobre el hardware, lo que es vital en sistemas embebidos donde cada ciclo de procesador y cada byte de memoria son preciosos. Por otro lado, Python se presentó como una muy buena solución para aquellos aspectos del proyecto donde la velocidad de desarrollo y la facilidad de uso eran primordiales. Su sintaxis intuitiva y su colección de bibliotecas simplifican la implementación de funcionalidades complejas, aunque con ciertas desventajas en términos de eficiencia.

En cuanto a Bash, aunque es una herramienta poderosa para la automatización de tareas y la gestión de configuraciones del sistema, su aplicación en el contexto de sistemas embebidos es

limitada. Bash es ideal para scripts de inicialización y tareas de mantenimiento, pero no está diseñado para el desarrollo de aplicaciones embebidas que requieren interacciones a bajo nivel con el hardware o tiempos de respuesta críticos.

En resumen, la elección del lenguaje de programación adecuado es un ejercicio de equilibrio entre múltiples factores, incluyendo, la eficiencia, la velocidad de desarrollo, la facilidad de mantenimiento y la escalabilidad. La decisión final debe estar alineada con los objetivos y necesidades del proyecto y las capacidades del equipo de desarrollo, asegurando que el sistema embebido no solo cumpla con los requisitos actuales, sino que también sea robusto y adaptable para las demandas futuras.

Bibliografía.

- GNU Manuals online - GNU Project - Free Software Foundation. (s. f.). https://www.gnu.org/manual/manual.html
- How to control the Raspberry Pi GPIO using C. (2020, septiembre 7). LEARN @ CIRCUITROCKS. https://learn.circuit.rocks/how-to-control-the-raspberry-pi-gpio-using-c
- (S/f). Digikey.com, de https://www.digikey.com/es/maker/tutorials/2019/how-to-use-gpio-on-the-raspberry-pi-with-c
- Llamas, L. (2023, septiembre 5). Qué son y cómo usar los GPIO en el ESP32. Luis Llamas. https://www.luisllamas.es/esp32-gpio/
- Usando los GPIO con Python – Prometec. (s. f.). https://www.prometec.net/usando-los-gpio-con-python/

ANEXO.

En este anexo encontraras todos los códigos realizados y usados para la realización de este proyecto.

- Shell

```
#!/bin/bash
# GPIO utilizados para generar numero
GPIO_0=27
GPIO_1=22
GPIO_2=17
GPIO_3=4
GPIO_4=18
GPIO_5=23
GPIO_6=24
GPIO_7=8

# En caso que no se pase algun valor
if [ $# -ne 1 ]; then   # si no hay argumento
  echo "No hay comando"
  echo "los comandos a utilizar es config, valor, cerrar"
  exit 2    # Numero invalido de argumentos
fi

# Configurar GPIO como entradas
if [ "$1" == "config" ]; then
      echo "Exportando GPIO numero $GPIO_0"
      echo $GPIO_0 >> "/sys/class/gpio/export"
      sleep 1  # para asegurar que se exporto correctamene
      echo "in" >> "/sys/class/gpio/gpio$GPIO_0/direction"
      echo "Exportando GPIO numero $GPIO_1"
      echo $GPIO_1 >> "/sys/class/gpio/export"
      sleep 1  # para asegurar que se exporto correctamene
      echo "in" >> "/sys/class/gpio/gpio$GPIO_1/direction"
      echo "Exportando GPIO numero $GPIO_2"
      echo $GPIO_2 >> "/sys/class/gpio/export"
      sleep 1  # para asegurar que se exporto correctamene
      echo "in" >> "/sys/class/gpio/gpio$GPIO_2/direction"
            echo "Exportando GPIO numero $GPIO_3"
      echo $GPIO_3 >> "/sys/class/gpio/export"
      sleep 1  # para asegurar que se exporto correctamene
      echo "in" >> "/sys/class/gpio/gpio$GPIO_3/direction"
            echo "Exportando GPIO numero $GPIO_4"
      echo $GPIO_4 >> "/sys/class/gpio/export"
      sleep 1  # para asegurar que se exporto correctamene
      echo "in" >> "/sys/class/gpio/gpio$GPIO_4/direction"
            echo "Exportando GPIO numero $GPIO_5"
```

```
	echo $GPIO_5 >> "/sys/class/gpio/export"
	sleep 1  # para asegurar que se exporto correctamene
	echo "in" >> "/sys/class/gpio/gpio$GPIO_5/direction"
		echo "Exportando GPIO numero $GPIO_6"
	echo $GPIO_6 >> "/sys/class/gpio/export"
	sleep 1  # para asegurar que se exporto correctamene
	echo "in" >> "/sys/class/gpio/gpio$GPIO_6/direction"
		echo "Exportando GPIO numero $GPIO_7"
	echo $GPIO_7 >> "/sys/class/gpio/export"
	sleep 1  # para asegurar que se exporto correctamene
	echo "in" >> "/sys/class/gpio/gpio$GPIO_7/direction"
fi

if [ "$1" == "cerrar" ]; then
	echo "cerrando el GPIO $GPIO_0"
	echo $GPIO_0 >> "/sys/class/gpio/unexport"
	echo "cerrando el GPIO $GPIO_1"
	echo $GPIO_1 >> "/sys/class/gpio/unexport"
		echo "cerrando el GPIO $GPIO_2"
	echo $GPIO_2 >> "/sys/class/gpio/unexport"
		echo "cerrando el GPIO $GPIO_3"
	echo $GPIO_3 >> "/sys/class/gpio/unexport"
		echo "cerrando el GPIO $GPIO_4"
	echo $GPIO_4 >> "/sys/class/gpio/unexport"
		echo "cerrando el GPIO $GPIO_5"
	echo $GPIO_5 >> "/sys/class/gpio/unexport"
		echo "cerrando el GPIO $GPIO_6"
	echo $GPIO_6 >> "/sys/class/gpio/unexport"
		echo "cerrando el GPIO $GPIO_7"
	echo $GPIO_7 >> "/sys/class/gpio/unexport"
fi
if [ "$1" == "valor" ]; then
	n=100 #Numeros de experimentos
	i=0
		#echo >tiempoShell.txt
		#echo >valorSShell.txt
		rm ./tiempoShell.txt
		rm ./valorShell.txt
	while((i<n))
	do
		bit0=$(cat "/sys/class/gpio/gpio$GPIO_0/value")
		bit1=$(cat "/sys/class/gpio/gpio$GPIO_1/value")
		bit2=$(cat "/sys/class/gpio/gpio$GPIO_2/value")
		bit3=$(cat "/sys/class/gpio/gpio$GPIO_3/value")
		bit4=$(cat "/sys/class/gpio/gpio$GPIO_4/value")
			bit5=$(cat "/sys/class/gpio/gpio$GPIO_5/value")
		bit6=$(cat "/sys/class/gpio/gpio$GPIO_6/value")
		bit7=$(cat "/sys/class/gpio/gpio$GPIO_7/value")
```

```
        #bit1=3.3
        let numero=bit0+bit1+bit2+bit3+bit4+bit5+bit6+bit7
        let i=i+1
        t=$(date +%s%N)
                #Visualizacion de cada valor individual
        echo $bit0
        echo $bit1
        echo $bit2
        echo $bit3
        echo $bit4
        echo $bit5
        echo $bit6
        echo $bit7
        echo $i
        echo $t>>tiempoShell.txt
        echo $numero>>valorShell.txt
    done
fi
```

- Python

```
        import os
        import time

        # GPIO utilizados para generar numero
        GPIO_0 = 27
        GPIO_1 = 22
        GPIO_2 = 17
        GPIO_3 = 4
        GPIO_4 = 18
        GPIO_5 = 23
        GPIO_6 = 24
        GPIO_7 = 8

        # Función para exportar un GPIO
        def export_gpio(gpio):
          with open(f"/sys/class/gpio/export", 'w') as file:
            file.write(str(gpio))

        # Función para configurar la dirección de un GPIO
        def set_direction(gpio, direction):
          with open(f"/sys/class/gpio/gpio{gpio}/direction", 'w') as file:
            file.write(direction)

        # Función para desexportar un GPIO
        def unexport_gpio(gpio):
          with open(f"/sys/class/gpio/unexport", 'w') as file:
            file.write(str(gpio))
```

```
# Función para leer el valor de un GPIO
def read_gpio(gpio):
  with open(f"/sys/class/gpio/gpio{gpio}/value", 'r') as file:
    return file.read().strip()

# Verificar si se pasó algún valor
if len(os.sys.argv) != 2:
  print("No hay comando")
  print("Los comandos a utilizar son config, valor, cerrar")
  os.sys.exit(2)  # Número inválido de argumentos

# Configurar GPIO como entradas
if os.sys.argv[1] == "config":
  for gpio in [GPIO_0, GPIO_1, GPIO_2, GPIO_3, GPIO_4, GPIO_5, GPIO_6, GPIO_7]:
    print(f"Exportando GPIO número {gpio}")
    export_gpio(gpio)
    time.sleep(1)  # para asegurar que se exportó correctamente
    print(f"Configurando GPIO número {gpio} como entrada")
    set_direction(gpio, "in")

# Cerrar GPIO
elif os.sys.argv[1] == "cerrar":
  for gpio in [GPIO_0, GPIO_1, GPIO_2, GPIO_3, GPIO_4, GPIO_5, GPIO_6, GPIO_7]:
    print(f"Cerrando el GPIO {gpio}")
    unexport_gpio(gpio)

# Leer valor de los GPIO
elif os.sys.argv[1] == "valor":
  n = 100  # Números de experimentos
  with open("tiempoPy.txt", 'w') as tiempo_file:
     tiempo_file.write("")
  with open("valorPy.txt", 'w') as valor_file:
      valor_file.write("")
  for i in range(n):
    bits = [read_gpio(gpio) for gpio in [GPIO_0, GPIO_1, GPIO_2, GPIO_3, GPIO_4,
GPIO_5, GPIO_6, GPIO_7]]
    numero = sum(int(bit) for bit in bits)
    t = time.time_ns()
    # Visualización de cada valor individual
    for bit in bits:
      print(bit)
    print(i)
    with open("tiempoPy.txt", 'a') as tiempo_file:
      tiempo_file.write(f"{t}\n")
    with open("valorPy.txt", 'a') as valor_file:
      valor_file.write(f"{numero}\n")
```

- C

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>

#define GPIO_0 27
#define GPIO_1 22
#define GPIO_2 17
#define GPIO_3 4
#define GPIO_4 18
#define GPIO_5 23
#define GPIO_6 24
#define GPIO_7 8

void export_gpio(int gpio_num) {
   FILE *export_file;
   export_file = fopen("/sys/class/gpio/export", "w");
   if (export_file == NULL) {
      perror("Error al abrir el archivo de exportación");
      exit(EXIT_FAILURE);
   }
   fprintf(export_file, "%d\n", gpio_num);
   fclose(export_file);

   char direction_path[100];
   sprintf(direction_path, "/sys/class/gpio/gpio%d/direction", gpio_num);
   FILE *direction_file = fopen(direction_path, "w");
   if (direction_file == NULL) {
      perror("Error al abrir el archivo de dirección");
      exit(EXIT_FAILURE);
   }
   fprintf(direction_file, "in\n");
   fclose(direction_file);
}

void unexport_gpio(int gpio_num) {
   FILE *unexport_file;
   unexport_file = fopen("/sys/class/gpio/unexport", "w");
   if (unexport_file == NULL) {
      perror("Error al abrir el archivo de unexport");
      exit(EXIT_FAILURE);
   }
   fprintf(unexport_file, "%d\n", gpio_num);
   fclose(unexport_file);
}
```

```
int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("No hay comando\n");
    printf("Los comandos a utilizar son: config, valor, cerrar\n");
    exit(EXIT_FAILURE);
  }

  if (strcmp(argv[1], "config") == 0) {
    export_gpio(GPIO_0);
    export_gpio(GPIO_1);
    export_gpio(GPIO_2);
    export_gpio(GPIO_3);
    export_gpio(GPIO_4);
    export_gpio(GPIO_5);
    export_gpio(GPIO_6);
    export_gpio(GPIO_7);
  } else if (strcmp(argv[1], "cerrar") == 0) {
    unexport_gpio(GPIO_0);
    unexport_gpio(GPIO_1);
    unexport_gpio(GPIO_2);
    unexport_gpio(GPIO_3);
    unexport_gpio(GPIO_4);
    unexport_gpio(GPIO_5);
    unexport_gpio(GPIO_6);
    unexport_gpio(GPIO_7);
  } else if (strcmp(argv[1], "valor") == 0) {
    int n = 100;
    int i;
        int GPIO[8]={27,22,17,4,18,23,24,8};
    FILE *tiempo_file = fopen("tiempoC.txt", "w");
    fclose(tiempo_file);
    FILE *valor_file = fopen("valorC.txt", "w");
    fclose(valor_file);

    for (i = 0; i < n; i++) {
      FILE *gpio_files[8];
      int bit_values[8];
      char gpio_value_path[100];
      int j;

      for (j = 0; j < 8; j++) {
        sprintf(gpio_value_path, "/sys/class/gpio/gpio%d/value", GPIO[j]);
                //printf("%s\n",gpio_value_path);
        gpio_files[j] = fopen(gpio_value_path, "r");
        if (gpio_files[j] == NULL) {
          perror("Error al abrir el archivo GPIO value");
          exit(EXIT_FAILURE);
        }
```

```
            fscanf(gpio_files[j], "%d", &bit_values[j]);
            fclose(gpio_files[j]);
        }

        // Supongamos que bit1 siempre tiene el valor 3.3
        //bit_values[1] = 3.3;

        int numero = 0;
        for (j = 0; j < 8; j++) {
            numero += bit_values[j];
        }

        // Visualización de cada valor individual
        for (j = 0; j < 8; j++) {
            printf("%d\n", bit_values[j]);
        }
        printf("%d\n", i);

            struct timeval tv;
            gettimeofday(&tv,NULL);

        // Guardar tiempo actual en tiempo.txt
        long long int t = tv.tv_sec*(long long int) 1000000000+tv.tv_usec;
        FILE *tiempo_file = fopen("tiempoC.txt", "a");
        if (tiempo_file == NULL) {
            perror("Error al abrir el archivo tiempo.txt");
            exit(EXIT_FAILURE);
        }
        fprintf(tiempo_file, "%lld\n", t);
        fclose(tiempo_file);

        // Guardar el número en valor.txt
        FILE *valor_file = fopen("valorC.txt", "a");
        if (valor_file == NULL) {
            perror("Error al abrir el archivo valor.txt");
            exit(EXIT_FAILURE);
        }
        fprintf(valor_file, "%d\n", numero);
        fclose(valor_file);
      }
    } else {
      printf("Comando no reconocido\n");
      printf("Los comandos a utilizar son: config, valor, cerrar\n");
      exit(EXIT_FAILURE);
    }

    return 0;
}
```

- C++

```
#include <fstream>
#include <iostream>
#include <string>
#include <chrono>
#include <thread>
#include <vector>
#include <unistd.h> // Para la función sleep

// Función para exportar un GPIO y configurarlo como entrada
void exportAndConfigureGPIO(int gpio) {
  std::ofstream file;
  file.open("/sys/class/gpio/export");
  if (file.is_open()) {
    file << gpio;
    file.close();
    sleep(1); // Esperar 1 segundo para asegurar que se exportó correctamente
    file.open("/sys/class/gpio/gpio" + std::to_string(gpio) + "/direction");
    if (file.is_open()) {
      file << "in";
      file.close();
    } else {
      std::cerr << "No se pudo configurar la dirección del GPIO número " << gpio << std::endl;
    }
  } else {
    std::cerr << "No se pudo exportar el GPIO número " << gpio << std::endl;
  }
}
// Función para desexportar un GPIO
void unexportGPIO(int gpio) {
  std::ofstream file;
  file.open("/sys/class/gpio/unexport");
  if (file.is_open()) {
    file << gpio;
    file.close();
  } else {
    std::cerr << "No se pudo cerrar el GPIO número " << gpio << std::endl;
  }
}

// Función para leer el valor de un GPIO
int readGPIO(int gpio) {
  std::ifstream file;
  file.open("/sys/class/gpio/gpio" + std::to_string(gpio) + "/value");
  if (file.is_open()) {
```

```
        int value;
        file >> value;
        file.close();
        return value;
    } else {
        std::cerr << "No se pudo leer el valor del GPIO número " << gpio << std::endl;
        return -1;
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        std::cerr << "Uso: " << argv[0] << " [config|cerrar|valor]" << std::endl;
        return 1;
    }

    std::string command = argv[1];
    const std::vector<int> gpios = {27, 22, 17, 4, 18, 23, 24,8};

    if (command == "config") {
        //const int gpios[] = {27, 22, 17, 4, 18, 23, 24, 25};
        for (int gpio : gpios) {
            std::cout << "Exportando GPIO número " << gpio << std::endl;
            exportAndConfigureGPIO(gpio);
        }
    }
    if (command == "cerrar") {
        for (int gpio : gpios) {
            std::cout << "Cerrando el GPIO " << gpio << std::endl;
            unexportGPIO(gpio);
        }
    } else if (command == "valor") {
        const int n = 101; // Número de experimentos
        std::ofstream tiempo_file("tiempoCm.txt", std::ofstream::out|std::ofstream::trunc);
        std::ofstream valor_file("valorCm.txt", std::ofstream::out|std::ofstream::trunc);

        for (int i = 0; i < n; ++i) {
            int numero = 0;
            for (int gpio : gpios) {
                int bit = readGPIO(gpio);
                std::cout << bit << std::endl;
                numero += bit;
            }

            auto t = std::chrono::high_resolution_clock::now();
            auto                                                duration                                                =
std::chrono::duration_cast<std::chrono::nanoseconds>(t.time_since_epoch()).count();
```

```
        std::cout << i << std::endl;
            tiempo_file.open("tiempoCm.txt",std::ofstream::out|std::ofstream::app);
            valor_file.open("valorCm.txt",std::ofstream::out|std::ofstream::app);
        if (tiempo_file.is_open() && valor_file.is_open()) {
          tiempo_file << duration << std::endl;
          valor_file << numero << std::endl;
        }

            tiempo_file.close();
            valor_file.close();
      }
    }

    return 0;
}
```