Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica
Ingeniería en Computación

Proyecto 1
“Interfaz Física y Lenguajes de Programación”

Integrantes:
- Castelan Ramos Carlos
- Corona Nava Pedro Jair
- Mendoza de los Santos Lirio Aketzalli
- Ortiz Camacho Jessica Elizabeth

Materia: Fundamentos de Sistemas Embebidos
Grupo: 04
Semestre: 2024-1

Fecha de entrega: 12 de octubre 2023

Proyecto 1 “Interfaz Gráfica”

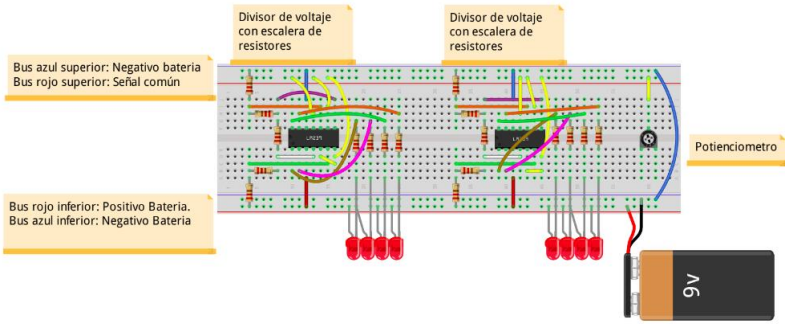
Desarrollo.

Interfaz Física.

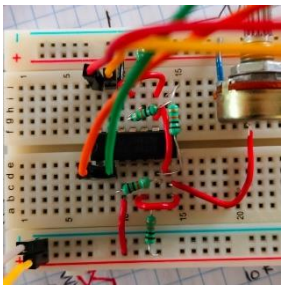
La construcción de la interfaz física contempló el uso de diferentes dispositivos, desde comparadores, optoacopladores y elementos básicos como cable, jumpers, leds, capacitores, resistencias y batería. Todo eso con la intención de generar una interfaz capaz de recibir y percibir valores del medio continúo para poder discretizarlos e interpretarlos posteriormente.

Se llevaron a cabo tres etapas:

1. Etapa 1: Se probó el correcto funcionamiento de los comparadores de forma individual, generando dos complejos de luces indicadores de corriente mediante el uso de leds.

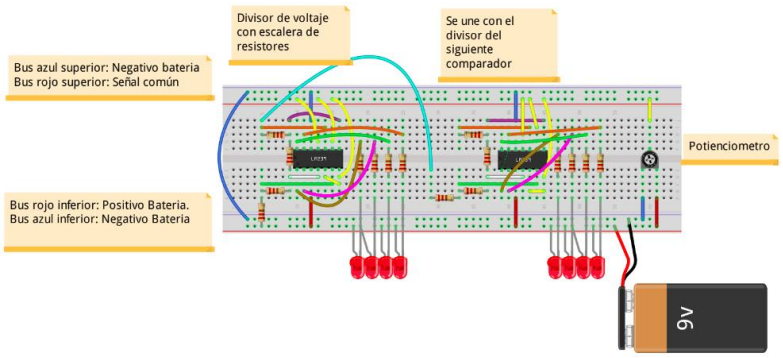


Diseño Etapa 1 en Fritzing



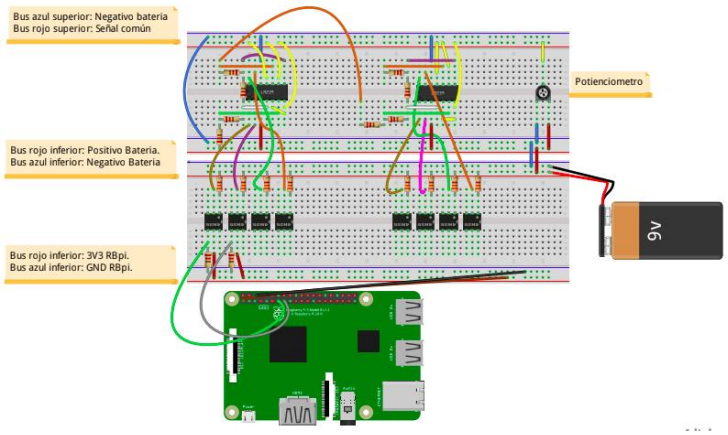
Alambrado Etapa 1

2. Etapa 2: Se realizó la conexión en serie de los dos complejos para obtener uno formado por 8 comparadores.

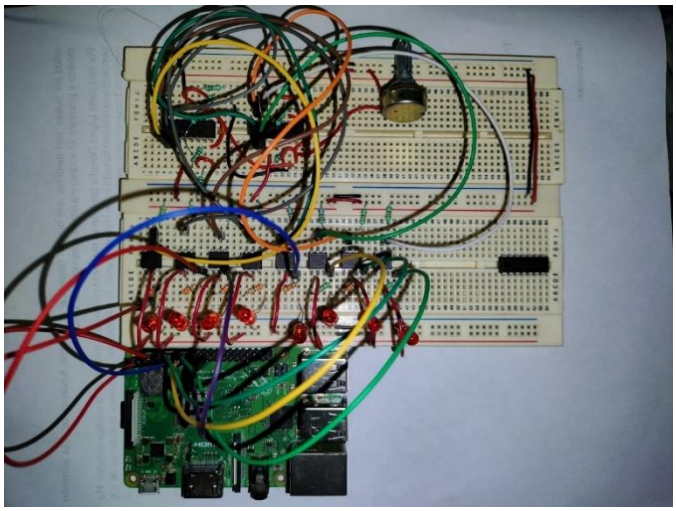


Diseño Etapa 2 en Fritzing

3. Etapa 3: Integración de los optoacopladores al circuito para funcionar como una conexión eléctricamente aislada entre el circuito de los comparadores con el circuito de control que es la raspberrypi.



Diseño Etapa 3 en Fritzing



Alambrado Etapa 3

Configuración de pines.

La configuración de pines se llevó a cabo para los GPIO del 0 al 7 en la cual se hizo uso de los pines 27, 22, 23, 24, 5, 6, 13 y respectivamente. Esto se realizó a partir de códigos de configuración en diferentes lenguajes de programación. La actividad se llevó a cabo con la intención de obtener los diferentes tiempos que tardan cada uno de los lenguajes para la configuración de los pines, esto para identificar las características de los lenguajes de programación, siendo unos compilados y otros interpretados.

Los códigos contienen tres funciones (config, valor, cierre):

a) **config:** Configura el uso de los pines GPIO mencionados anteriormente.
b) **valor:** Lee los valores que los pines reciben del medio continúo.
c) **cierre:** Termina la ejecución del programa y concluye el uso de los pines.

Cada uno de los programas toma la lectura de 10,000 iteraciones de los pines definidos tomadas en un tiempo **n**, es entonces que a partir de estos resultados generamos códigos en python para el análisis de los datos.

La primera parte de gráficas representa el cambio del crecimiento de tiempos en contra del número de iteraciones tomadas, en la cual se espera como resultado una gráfica cuadrática, ya que los tiempos deben crecer de manera constante.

Para esto únicamente se hizo uso del archivo generados en los programas del eje x, llamados SHx.txt, PYx.txt, Cx.txt y Cppx.txt los cuales son tratados en el programa itera.py.

La segunda parte de gráficas representa un histograma de frecuencias sobre el cálculo de Δt_n:

Amplitud	Tiempo	Δt_n
0	t_0	..
0	t_1	$\Delta t_0 = t_1 - t_0$
0	....	...
0	t_n	$\Delta t_n = t_{n+1} - t_n$

Tabla del cálculo de Δt_n

Para esto se hizo uso de un programa llamado resta.py que obtiene el cáculo de Δt a partir de los tiempos de eje x y los resultados se almcenan en otro archivo, llamados SHxRes.txt, PYxRes.txt, CxRes.txt y CppxRes.txt, en base a estos resultados generamos el histograma de frecuencias de las Δt haciendo uso de histo4.py

- Shell

  Código:

  La primera línea que utilizamos en el código es conocido como shebang (o sha-bang) es uno de los más comúnmente utilizados e indica el intérprete que se utilizará para la ejecución del script. Posteriormente, colocamos los GPIO que usamos junto con su identificador, dado que son 8 los numeramos del 0 al 7.

```
#!/bin/bash

# GPIO utilizados para generar numero
GPIO_0=27
GPIO_1=22
GPIO_2=23
GPIO_3=24
GPIO_4=5
GPIO_5=6
GPIO_6=13
GPIO_7=25
```

  Enseguida se verifica que el valor ingresa al ejecutar contenga uno de los tres comandos válidos disponibles, o de lo contrario mandará mensajes indicando que el argumento es inválido y no se podrá seguir ejecutando.

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

Después se realizan los procesos según el argumento proporcionado, para el caso de config se realizará la configuración de los GPIO que elegimos, se nos mostrará mensajes de confirmación de una correcta exportación correcta para poder validar que el proceso se realiza con éxito.

```
# Configurar GPIO como entradas

if [ "$1" == "config" ]; then
        #GPIO 27
        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"

              #GPIO 22
              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"

              #GPIO 23
              echo "Exportando GPIO2 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"

              #GPIO 24
              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"

              #GPIO 5
              echo "Exportando GPIO4 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"

              #GPIO 6
              echo "Exportando GPIO 5 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"

              #GPIO 13
              echo "Exportando GPIO6 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"

              #GPIO 25
              echo "Exportando GPIO 7 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
```

Para el argumento cerrar se realiza el cierre de cada uno de los GPIO usados, por lo que des exportamos cada uno de ellos.

```
#Cierre de los puertos GPIO
if [ "$1" == "cerrar" ]; then
        #GPIO 27
        echo "cerrando el GPIO $GPIO_0"
        echo $GPIO_0 >> "/sys/class/gpio/unexport"

        #GPIO 22
        echo "cerrando el GPIO $GPIO_1"
        echo $GPIO_1 >> "/sys/class/gpio/unexport"

        #GPIO 23
        echo "cerrando el GPIO $GPIO_2"
        echo $GPIO_2 >> "/sys/class/gpio/unexport"

        #GPIO 24
        echo "cerrando el GPIO $GPIO_3"
        echo $GPIO_3 >> "/sys/class/gpio/unexport"

        #GPIO 5
        echo "cerrando el GPIO $GPIO_4"
        echo $GPIO_4 >> "/sys/class/gpio/unexport"

        #GPIO 6
        echo "cerrando el GPIO $GPIO_5"
        echo $GPIO_5 >> "/sys/class/gpio/unexport"

        #GPIO 13
        echo "cerrando el GPIO $GPIO_6"
        echo $GPIO_6 >> "/sys/class/gpio/unexport"

        #GPIO 25
        echo "cerrando el GPIO $GPIO_7"
        echo $GPIO_7 >> "/sys/class/gpio/unexport"

fi
```

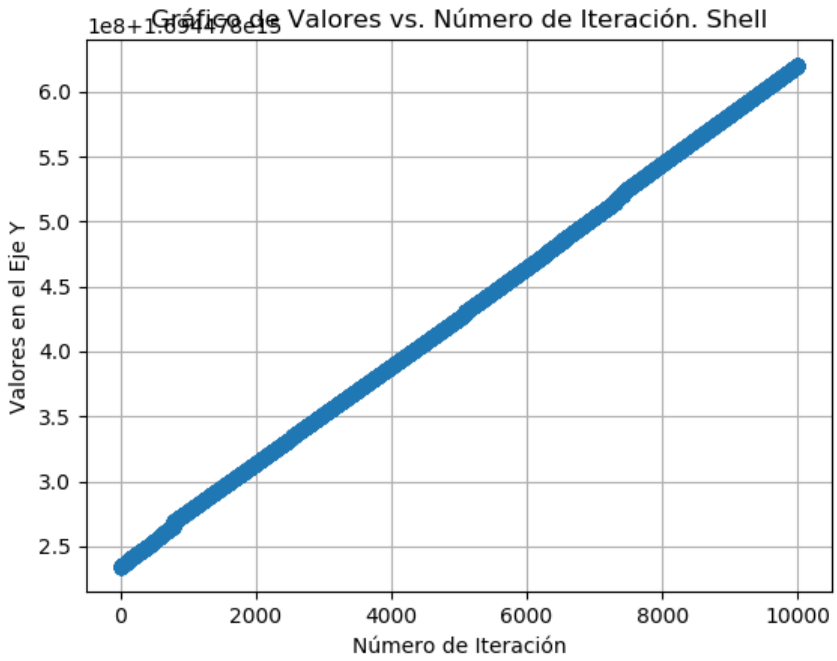
Por último, para el argumento valor se realiza la obtención de dos valores, para ello primero eliminamos el archivo x.txt en caso de que ya exista y con un ciclo realizamos las 10k iteraciones para cada uno de los GPIO, para ello se usaron variables bit0 a bit7. Con estos datos realizamos la suma en número.

```
#Obtención de valores
if [ "$1" == "valor" ]; then
        rm x.txt        #En caso de ya tener un archivo creado
        t0=$(date +%s%N)
        i=0
        while [ $i -le 10000 ]  #ciclo para realizar las iteraciones
        do                      #para la obtención de cada GPIO
        bit0=$(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")
        let numero=bit0+bit1+bit2+bit3+bit4+bit5+bit6+bit7 #suma de los 8 bits  de valores obtenidos
```

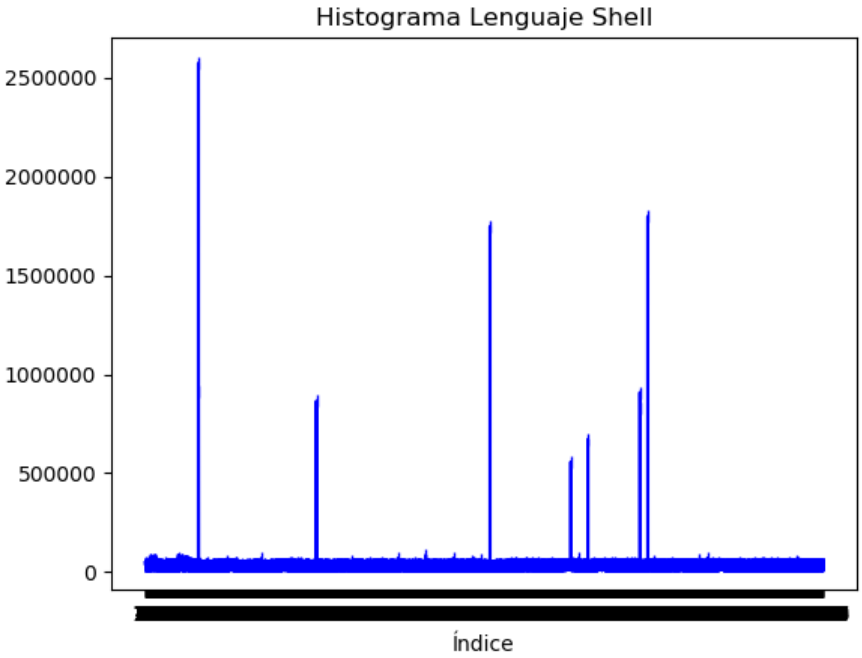
Después de esto se puede realizar el cálculo del tiempo y poder guardar el dato del tiempo en el archivo x.txt y la suma de valores en el archivo y.txt

```
        #Si quisieramos observar cada uno de estos valores individualmente junto con su suma
        #echo $bit0
        #echo $bit1
        #echo $bit2
        #echo $bit3
        #echo $bit4
        #echo $bit5
        #echo $bit6
        #echo $bit7
        #echo $numero

        #Calculo del tiempo
        t=$(date +%s%6N)
        #echo $t

        #Guarda el tiempo y la suma obtenida en dos archivos diferentes.
        echo -e "$numero" >> y.txt
        echo -e "$t" >> x.txt
        ((i++))
        done
fi
```

Gráfica de Tiempos vs Iteraciones:



Histograma de frecuencias de Δt :



- Python

Código:

Para comenzar con el programa debemos importar los modulos que nos será utiles para obtener el resultado esperado. En este caso son dos:

- import RPi.GPIO as GPIO: Importa la biblioteca RPi.GPIO y la renombramos como "GPIO" para facilitar su uso.
- import time: Importa el módulo "time" que se utiliza para trabajar con temporizadores y obtener marcas de tiempo.

```
import RPi.GPIO as GPIO
import time
```

Posteriormente Se definen ocho pines GPIO utilizando números para identificar cada pin. Estos pines se asignan a las variables GPIO_0 a GPIO_7.

```
# Definición de pines GPIO utilizados
GPIO_0 = 27
GPIO_1 = 22
GPIO_2 = 23
GPIO_3 = 24
GPIO_4 = 5
GPIO_5 = 6
GPIO_6 = 13
GPIO_7 = 25
```

Una vez teniendo los pines generamos la función "configurar_pines()" que se utiliza para configurar todos los pines GPIO definidos en el paso anterior. Además establece el modo de la Raspberry Pi en el modo BCM y configura cada pin como una entrada.

```
# Configuración de pines GPIO
def configurar_pines():
    GPIO.setmode(GPIO.BCM)  # Establece el modo de pines en BCM
    pines = [GPIO_0, GPIO_1, GPIO_2, GPIO_3, GPIO_4, GPIO_5, GPIO_6, GPIO_7]

    for pin in pines:
        GPIO.setup(pin, GPIO.IN)  # Configura el pin como entrada
```

Ahora lo que nos compete es el cierre y limpieza de pines GPIO: para ello empleamos la función "cerrar_pines()" que se se utiliza para limpiar y cerrar todos los pines GPIO configurados. Esto lo hacemos para garantizar que los pines no queden en un estado indefinido.

```
# Cierre y limpieza de pines GPIO
def cerrar_pines():
    GPIO.cleanup()  # Limpia y cierra los pines configurados
```

La función leer_valores() está diseñada para leer los valores de los pines GPIO y registrarlos en dos archivos de texto, 'x.txt' y 'y.txt'. Esto se hace en un bucle que se ejecuta 10,000 veces. En cada iteración, se leen los estados de los ocho pines GPIO definidos, se suma el número de pines en alto y se registra el valor resultante junto con una marca de tiempo en 'x.txt' y el número de pines en alto en 'y.txt'. Luego, el programa espera una pequeña fracción de segundo (1 microsegundo) antes de la siguiente lectura.

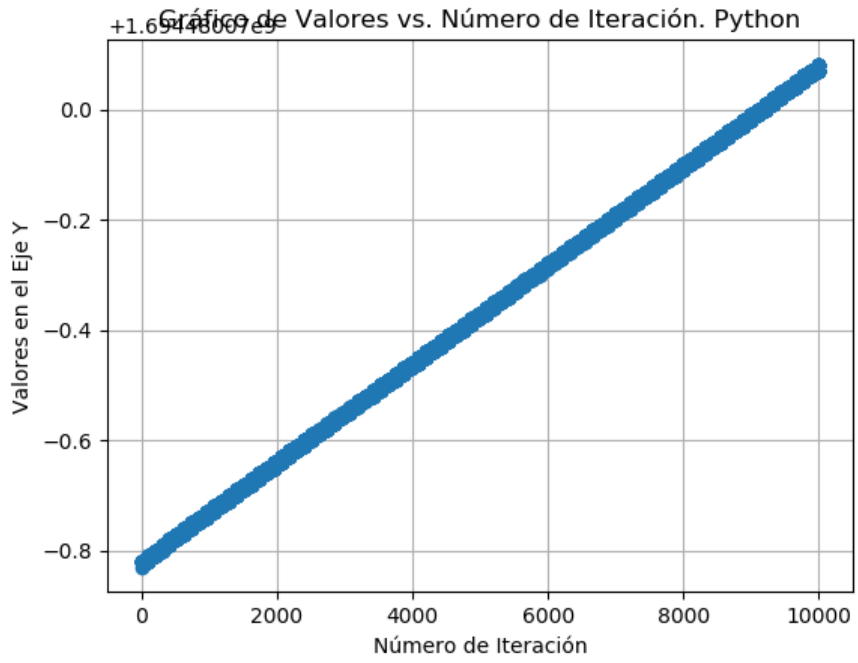
```
# Lectura de valores de pines GPIO y registro
def leer_valores():
    try:
        with open('x.txt', 'w') as x_file, open('y.txt', 'w') as y_file:
            for _ in range(10000):  # Realiza 10,000 lecturas
                valores = [GPIO.input(pin) for pin in [GPIO_0, GPIO_1, GPIO_2, GPIO_3, GPIO_4, GPIO_5, GPIO_6, GPIO_7]]
                numero = sum(valores)  # Suma de valores en alto (1)
                timestamp = time.time()  # Obtiene la marca de tiempo actual
                x_file.write('%s\n' % timestamp)  # Escribe la marca de tiempo en 'x.txt'
                y_file.write(f'{numero}\n')  # Escribe el número de pines en alto en 'y.txt'
                time.sleep(0.000001)  # Espera 1 microsegundo
    except KeyboardInterrupt:
        pass
```

Por último tenemos el control princpial del progama.La sección "if __name__ == '__main__':" verifica si el script se está ejecutando como un programa principal. Luego, analiza los argumentos de la línea de comandos utilizando sys.argv. Dependiendo del argumento proporcionado, se realizará una de las siguientes acciones:

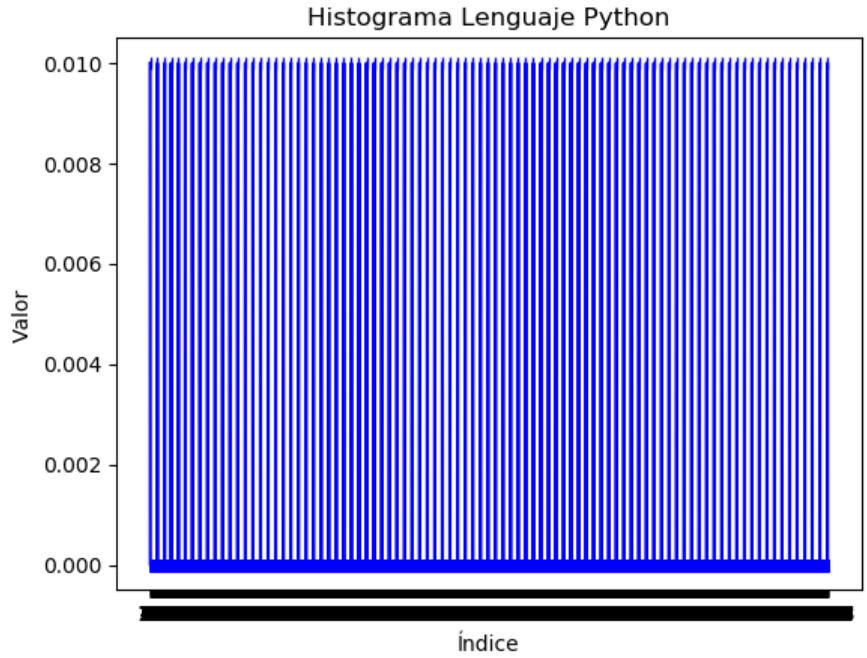
- Si se ejecuta python programa.py config, se configurarán los pines GPIO.
- Si se ejecuta python programa.py cerrar, se cerrarán y limpiarán los pines GPIO.
- Si se ejecuta python programa.py valor, se iniciará la lectura y registro de valores de los pines GPIO.
- Si se proporciona un comando no reconocido, se mostrará un mensaje de error.

```
if __name__ == '__main__':
    import sys

    if len(sys.argv) != 2:
        print("Uso: python programa.py [config|cerrar|valor]")
        sys.exit(2)

    if sys.argv[1] == 'config':
        configurar_pines()  # Configura los pines GPIO
    elif sys.argv[1] == 'cerrar':
        cerrar_pines()  # Limpia y cierra los pines GPIO
    elif sys.argv[1] == 'valor':
        leer_valores()  # Lee valores de los pines y registra
    else:
        print("Comando no reconocido. Los comandos válidos son: config, cerrar, valor")
        sys.exit(2)
```

Gráfica de Tiempos vs Iteraciones:



Histograma de frecuencias de Δt :



- C

Código:

Para implementar el código en C, utilizamos la biblioteca WiringPi, esta biblioteca está diseñada para Raspberry Pi con Raspbian de 32-bits; permite la configuración y acceso a los pines GPIO.

El código realizado sigue la misma lógica que el código de Shell, la primera parte corresponde a la declaración de constantes del procesador (los puertos GPIO) colocándoles un alias para identificarlos con facilidad.

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

```c
#define GPIO_0 27
#define GPIO_1 22
#define GPIO_2 23
#define GPIO_3 24
#define GPIO_4 5
#define GPIO_5 6
#define GPIO_6 13
#define GPIO_7 25
```

A continuación, definimos tres funciones, las cuales se encargan de configurar los pines, cerrarlos y leer los valores recibidos.

```c
void configurarPines() {
    // Se inicializa wiringPi y se indica el uso de los pines de forma directa
    wiringPiSetupGpio();
    // Configurar los pines GPIO como entradas
    pinMode(GPIO_0, INPUT);
    pinMode(GPIO_1, INPUT);
    pinMode(GPIO_2, INPUT);
    pinMode(GPIO_3, INPUT);
    pinMode(GPIO_4, INPUT);
    pinMode(GPIO_5, INPUT);
    pinMode(GPIO_6, INPUT);
    pinMode(GPIO_7, INPUT);
    printf("Configuración de pines realizada.\n");
}
```

En el caso de la biblioteca utilizada no es necesario cerrar los puertos, por lo que la función encargada de esta parte únicamente imprime un mensaje en la consola.

```c
void cerrarPines() {
    // No es necesario en wiringPi
    printf("Pines GPIO cerrados y limpiados.\n");
}
```

Una vez más, al leer los valores se realizan 10k iteraciones, y en cada una se guarda el tiempo tras realizar la suma de todos los valores. Para el tiempo, tuvimos que utilizar la biblioteca “time.h”, para utilizar la función “clock_gettime” indicando que la respuesta debe ser guaradada en una estructura “timespec”, esto para obtener el tiempo en segundos y nanosegundos.

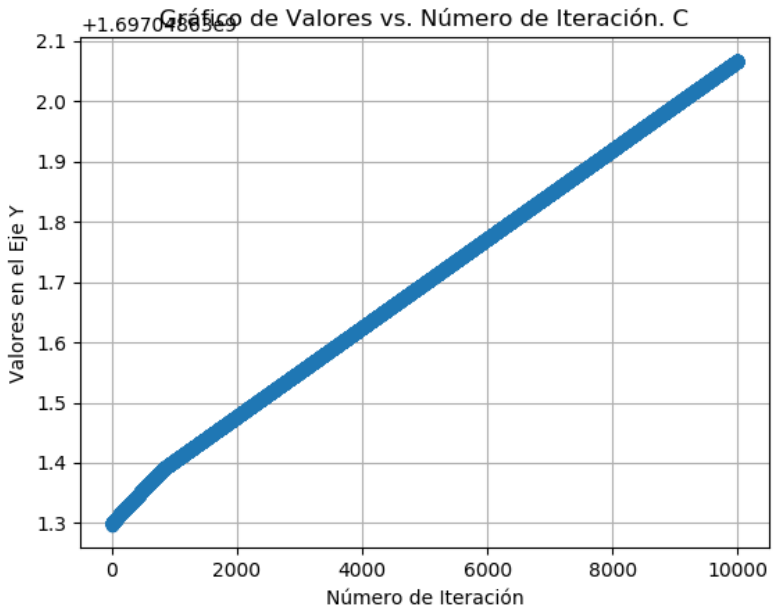
Una vez obtenido el tiempo se realiza el casteo de los segundos y nano segundos, con el fin de obtener un valor decimal en lugar de un entero. Posteriormente, guardamos los tiempos y la suma de valores en los archivos correspondientes, y se cierran ambos archivos al terminar.

```c
void leerValores() {
    FILE *xFile = fopen("x.txt", "w"); // Abrir archivos para escritura
    FILE *yFile = fopen("y.txt", "w");

    for (int i = 0; i < 10000; i++) { // Se realizan 10000 iteraciones
        int valores[8] = { //Lectura de los pines declarados al inicio ¿
            digitalRead(GPIO_0), digitalRead(GPIO_1), digitalRead(GPIO_2), digitalRead(GPIO_3),
            digitalRead(GPIO_4), digitalRead(GPIO_5), digitalRead(GPIO_6), digitalRead(GPIO_7)
        };

        int numero = 0;
        for (int j = 0; j < 8; j++) { //Se recorre el arreglo de valores leídos y se suman entre sí
            numero += valores[j];
        }
        struct timespec ts;
        clock_gettime(CLOCK_REALTIME, &ts); //Se obtiene el tiempo en este puento del programa
        // Se calcula el tiempo en segundos con milisegundos
        double timestamp_seconds = (double)ts.tv_sec + (double)ts.tv_nsec / 1000000000.0;
        fprintf(xFile, "%.6f\n", timestamp_seconds); //Se guarda el tiempo en el archivo "x"

        fprintf(yFile, "%d\n", numero); // Se guarda el resultado de la suma en el archivo "y"
        usleep(1); // Esperar 1 microsegundo
    }
    // Cerrar los archivos
    fclose(xFile);
    fclose(yFile);
}
```

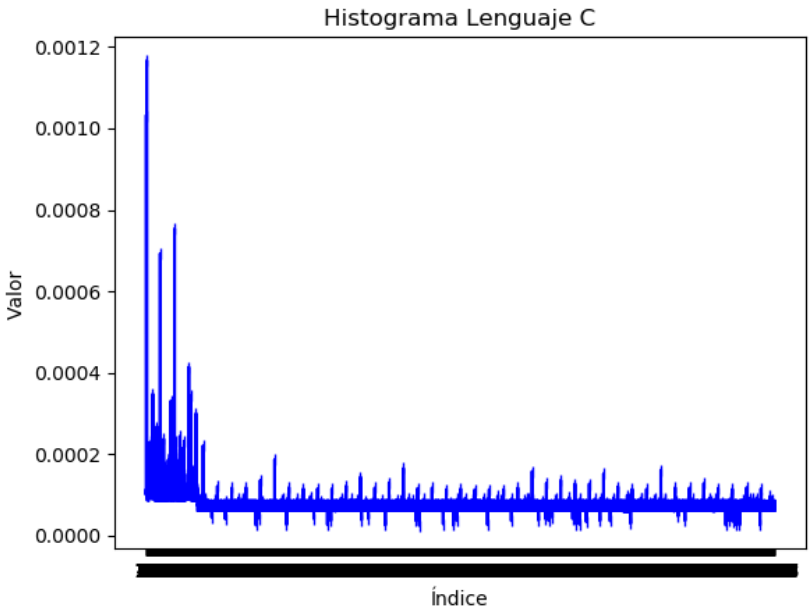
Finalmente, elaboramos una función main() que se encarga de validar los argumentos recibidos, y de llamar a la función correspondiente al comando válido recibido.

```c
int main(int argc, char *argv[]) {
    if (argc != 2) {  //Más de un comando o ningún comando recibido
        printf("Uso: %s [config|cerrar|valor]\n", argv[0]);
        exit(2);
    }

    char *command = argv[1]; //Obtenemos el comando recibido

    if (strcmp(command, "config") == 0) { // Comando de configuración
        configurarPines();
    } else if (strcmp(command, "cerrar") == 0) { // Comando de cierre
        cerrarPines();
    } else if (strcmp(command, "valor") == 0) { // Comando para obtención de valores
        leerValores();
    } else {
        printf("Comando no reconocido. Los comandos válidos son: config, cerrar, valor\n");
        exit(2);
    }

    return 0;
}
```

Gráfica de Tiempos vs Iteraciones:



Histograma de frecuencias de Δt :



- C++

Código:

El código en C++ es básicamente igual al explicado anteriormente realizado en C, sólo la sintaxis de algunas instrucciones es diferente.

```cpp
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <wiringPi.h>
```

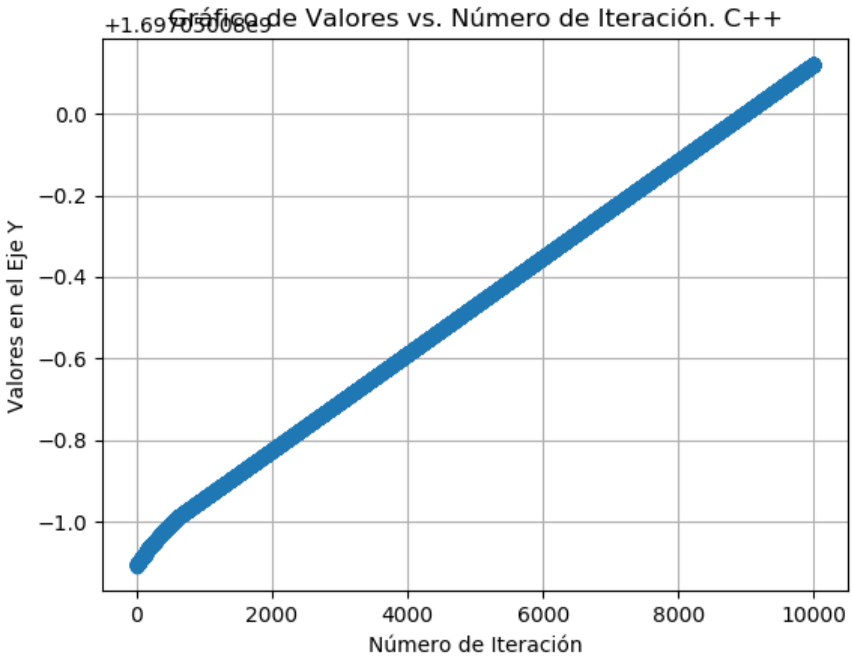
```cpp
#define GPIO_0 27
#define GPIO_1 22
#define GPIO_2 23
#define GPIO_3 24
#define GPIO_4 5
#define GPIO_5 6
#define GPIO_6 13
#define GPIO_7 25
```

```cpp
void configurarPines() {
    // Se inicializa wiringPi y se indica el uso de los pines de forma directa
    wiringPiSetupGpio();
    // Configurar los pines GPIO como entradas
    pinMode(GPIO_0, INPUT);
    pinMode(GPIO_1, INPUT);
    pinMode(GPIO_2, INPUT);
    pinMode(GPIO_3, INPUT);
    pinMode(GPIO_4, INPUT);
    pinMode(GPIO_5, INPUT);
    pinMode(GPIO_6, INPUT);
    pinMode(GPIO_7, INPUT);
    std::cout << "Configuración de pines realizada." << std::endl;
}

void cerrarPines() {
    // No es necesario en wiringPi
    std::cout << "Pines GPIO cerrados y limpiados." << std::endl;
}
```

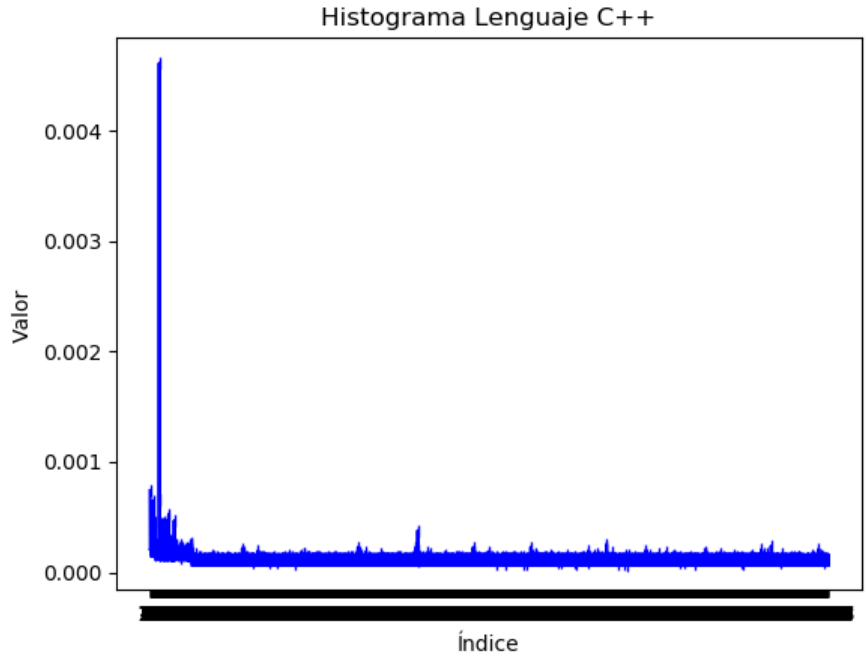
```cpp
void leerValores() {
    std::ofstream xFile("x.txt"); // Abrir archivos para escritura
    std::ofstream yFile("y.txt");

    for (int i = 0; i < 10000; i++) { // Se realizan 10000 iteraciones
        int valores[8] = { //Lectura de los pines declarados al inicio
            digitalRead(GPIO_0), digitalRead(GPIO_1), digitalRead(GPIO_2), digitalRead(GPIO_3),
            digitalRead(GPIO_4), digitalRead(GPIO_5), digitalRead(GPIO_6), digitalRead(GPIO_7)
        };

        int numero = 0;
        for (int j = 0; j < 8; j++) { //Se recorre el arreglo de valores leídos y se suman entre sí
            numero += valores[j];
        }

        timespec ts;
        clock_gettime(CLOCK_REALTIME, &ts); //Se obtiene el tiempo en este puento del programa
        // Se calcula el tiempo en segundos con milisegundos
        double timestamp_seconds = static_cast<double>(ts.tv_sec) + static_cast<double>(ts.tv_nsec) / 1000000000.0;
        xFile << std::fixed << std::setprecision(6) << timestamp_seconds << std::endl; //Se guarda el tiempo en el archivo "x"

        yFile << numero << std::endl; // Se guarda el resultado de la suma en el archivo "y"
        usleep(1); // Esperar 1 microsegundo
    }
    // Cerrar los archivos
    xFile.close();
    yFile.close();
}
```

```cpp
int main(int argc, char *argv[]) {
    if (argc != 2) { //Más de un comando o ningún comando recibido
        std::cout << "Uso: " << argv[0] << " [config|cerrar|valor]" << std::endl;
        return 2;
    }

    std::string command = argv[1]; //Obtenemos el comando recibido

    if (command == "config") { // Comando de configuración
        configurarPines();
    } else if (command == "cerrar") { // Comando de cierre
        cerrarPines();
    } else if (command == "valor") { // Comando para obtención de valores
        leerValores();
    } else {
        std::cout << "Comando no reconocido. Los comandos válidos son: config, cerrar, valor" << std::endl;
        return 2;
    }

    return 0;
}
```

Gráfica de Tiempos vs Iteraciones:



Histograma de frecuencias de Δt :



NOTA:

Las conclusiones son presentadas de manera individual al profesor.

Referencias

- WiringPi. GPIO Interface library for the Raspberry Pi. Recuperado de http://wiringpi.com/
- Free Software Foundation, Inc. *Getting the Time.* En *The GNU C Library.* Recuperado de **https://www.gnu.org/software/libc/manual/html_node/Getting-the-Time.html**
- Strickland, J. R., & Strickland, J. R. (2018). Meet WiringPi. *Raspberry Pi for Arduino Users: Building IoT and Network Applications and Devices*, 179-211.