NATIONAL AUTONOMOUS UNIVERSITY OF MEXICO

FACULTY OF ENGINEERING

FUNDAMENTALS OF EMBEDDED SYSTEMS

Project 1. Physical Interface

TEACHER:	Gerardo Ariel Castillo García
THEORY GROUP:	3
PROJECT NUMBER:	1
STUDENT(S):	Cruz Martinez Raul
	Mendoza Bolaños Carlos Gabriel
	Villanueva Corona Miguel Angel
SEMESTER:	2024-2
DELIVERY DATE:	April 1, 2024

ELECTRONIC MATERIAL:

- 1 potentiometer greater than 100 kOhms
- 2 Encapsulated comparators with 4 units
- 8 opto-isolator coupler with transistor output
- 8 red or green LEDs
- 10 resistors 330 Ohms
- 10 x 10k Ohm Resistors
- 2 breadboard
- 1 x 9V battery
- 1 Rasspberry pi 3B+ (provided by the Faculty of Engineering)
- 1 Embedded Card Charger
- Jumpers

The
development of this project is of great importance the construction of the physical wiring of the interface where it consists of 4 parts for a better management and understanding allowing to visualize correctly the circuit, in addition to helping us to check the operation in later parts as well as detect any possible failure in case they occur.

This circuit involves electronic components mentioned above with the intention of generating, receiving and perceiving values with the help of the Raspberry in a continuous way so that the information obtained is subsequently analyzed, understood and interpreted.

The steps carried out are as follows:

- Physical Interface Part 1

  For the first part, the wiring of the comparators was carried out, making voltage dividers with the resistors and at the same time placing LEDs which helped us to check the correct connection and operation of each of the comparators when manipulated by the potentiometer.

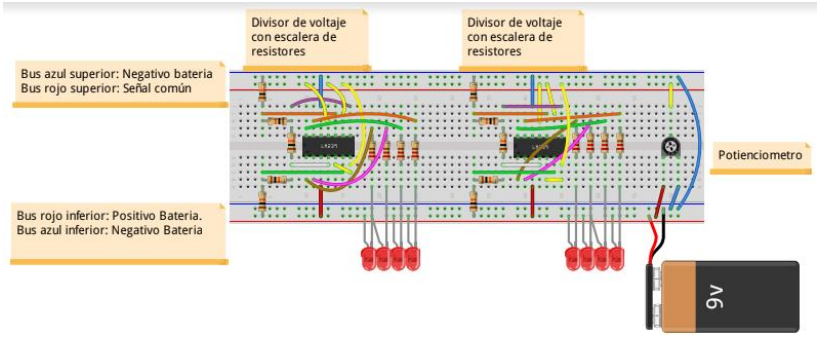


Fig. 1 Design Stage 1

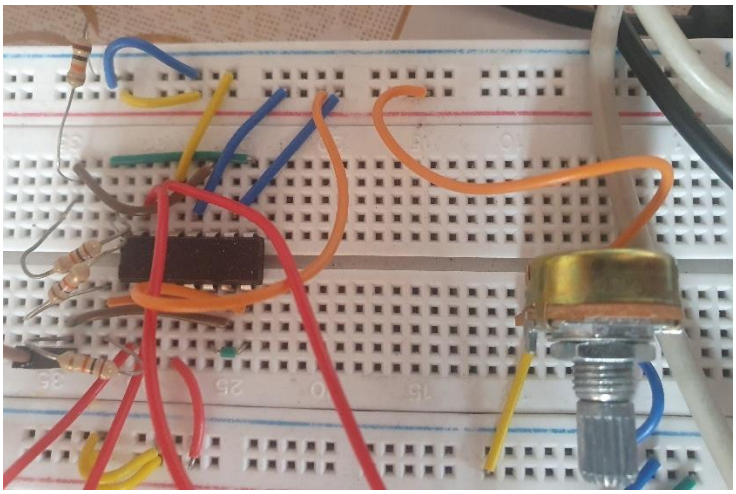
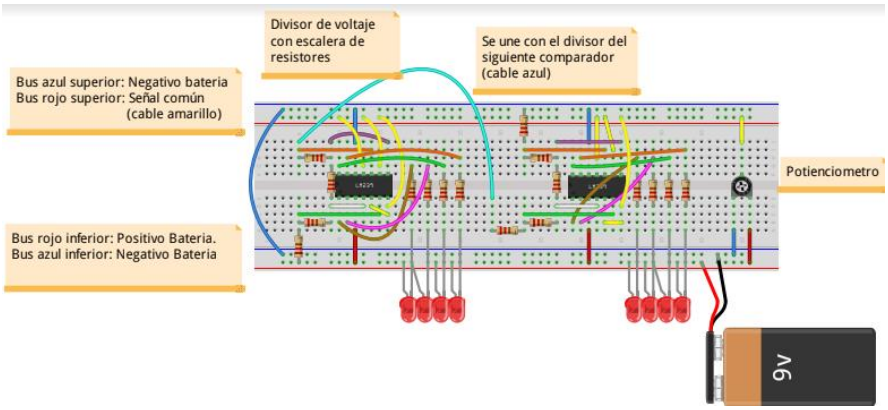


Fig. 2 Physical Wiring Stage 1

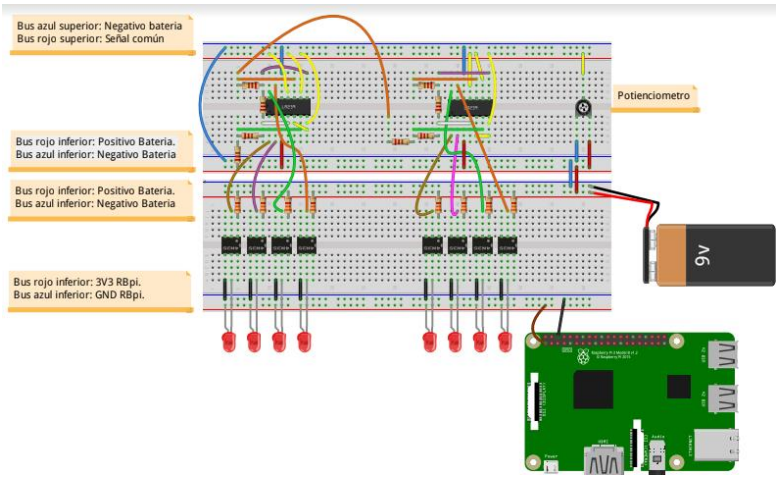
- Physical Interface Part 2

After the first stage we placed a jumper which would connect both comparators joining both connections, in this way obtain a comparator of eight, and as the potentiometer was moving it is visualized that the LEDs were turning on one by one until they were all on, allowing us to continue with the subsequent steps, just at the time of connecting the Raspberry.



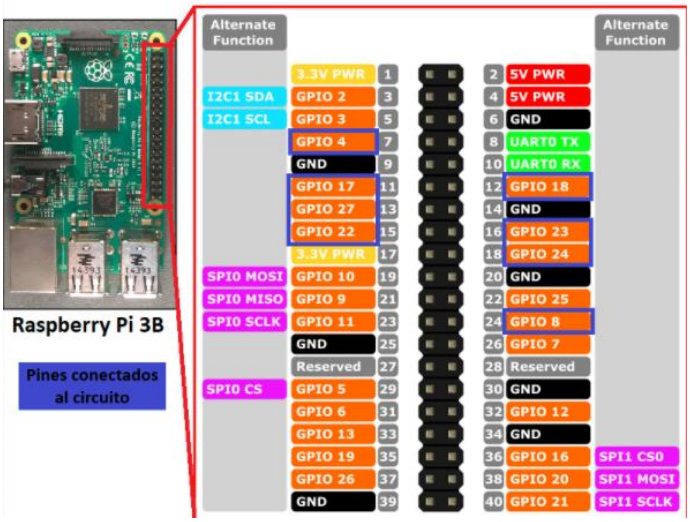
- Physical Interface Part 3

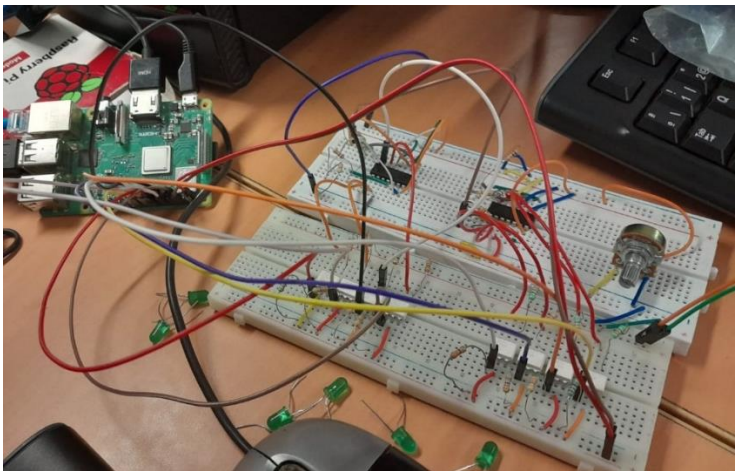
In this part, the opt-couplers were placed in the circuit to isolate the circuits made according to the one made in conjunction with the Raspberry, verifying in the circuit its correct operation in the same way with respect to the previous circuit with the LEDs



- Physical Interface Part 4

Finally, to finish with the hardware section, each of the LEDs was replaced by some pins of the Raspberry so that with the help of the software readings of each of the GPIOs were obtained and the given information was stored. Below is the assignment of the Raspberry's pins, as well as the full connection.





Configuration

As seen in the previous images, different pins of the Raspberry were used to make the readings by configuring the GPIO from 0 to 7 by placing pins 27, 22, 17, 4, 18, 23, 24 and 8 in that order, thus being the number 27 as the GPIO_0 and the pin 8 as the GPIO_7. Therefore, when defining the GPIOs to be used, through the use of different programming languages, each one will be configured to obtain times, having defined the performance of 100 iterations according to the 8 pins, with the intention of defining and differentiating what type of programming language is efficient for obtaining times in less time as the main objective.

Within each of the codes made and used, they will have three types of entries, which are:

- config. It allows you to open and make use of GPIOs using the above settings.
- value. It reads the data obtained from each GPIO involved.
- close. Finish using the GPIOs and close the pins for use.

Bearing in mind the above, it is worth mentioning that each of the programming languages involved will generate their own data files where you will have the time calculated according to each iteration carried out and the recorded values (tiempo.txt and valores.txt), remembering that there will be 100 iterations.

The time file will allow us to calculate with the help of the delta.py program, the difference between times, that is, the resulting time between each iteration, allowing us to observe graphically the elapsed time, so it will be analyzed to obtain a greater visualization of the language with greater efficiency.

The languages to be used are the following:

- Shell de Bash
- Python
- C
- C++

Below are each of the codes made for the creation of this project, as well as each of the graphs generated.

It should be noted that this assignment of pins must be done in each of the programs to be managed, since the purpose of this project is to identify which of all the selected languages performs the task in the shortest time and at the same time making graphs which show us the readings of these pins.

- Shell

Based on the code provided in Shell language, the GPIO mapping within the encoding is observed, displaying the following:

```
# 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 Mapping pins within the shell code

Likewise, it shows the way to know the type of input that will be available at the time of execution of the program, in case this parameter is not correct.

```
# 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
```

Otherwise, if the input is one of those accepted within the code, the following lines provided will be executed:

When the input is config, it will execute what is seen in the following image, where the export and write of the variables defined as GPIO within each of the directory is displayed, as well as a timeout in each of the GPIO files involved.

```
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 the entry is "close", in the same way as the config entry, it is exported and interacts with the directory to prevent the use of GPIOs, making a similar use to the aforementioned only in the opposite direction.

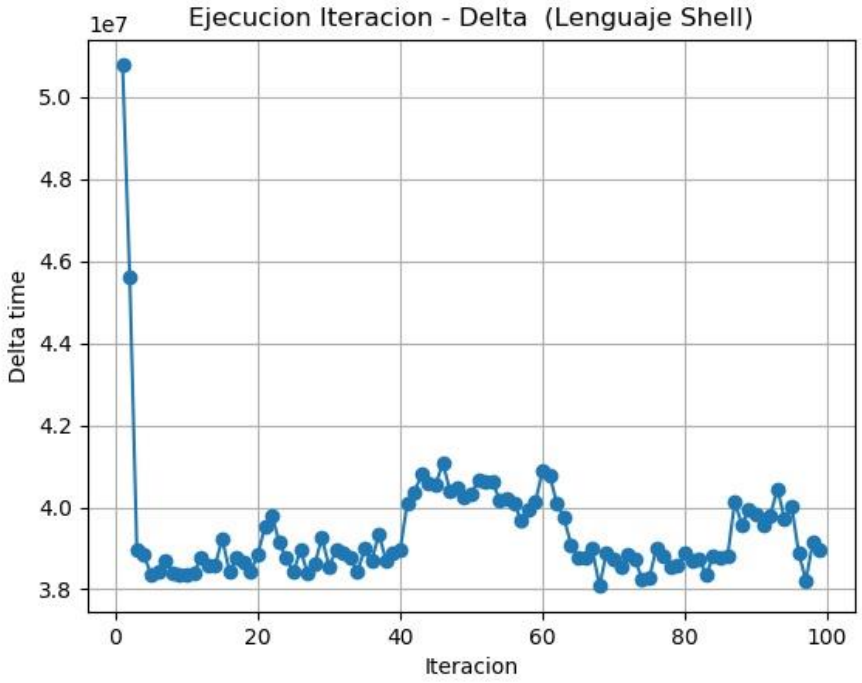
```
if [ "$1" == "cerrar" ]; then
        echo "cerrando el GPIO $GPIO_0"
        echo $GPIO_0 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_1"
        echo $GPIO_1 >> "/sys/class/gpio/unexport"
        echo "cerrando el GPIO $GPIO_2"
        echo $GPIO_2 >> "/sys/class/gpio/unexport"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

And if the entry is "value", the 100 iterations are carried out with the purpose of collecting information about the execution times within the program, so the times and values of each iteration will be recorded in files, previously checking the existence of the file.

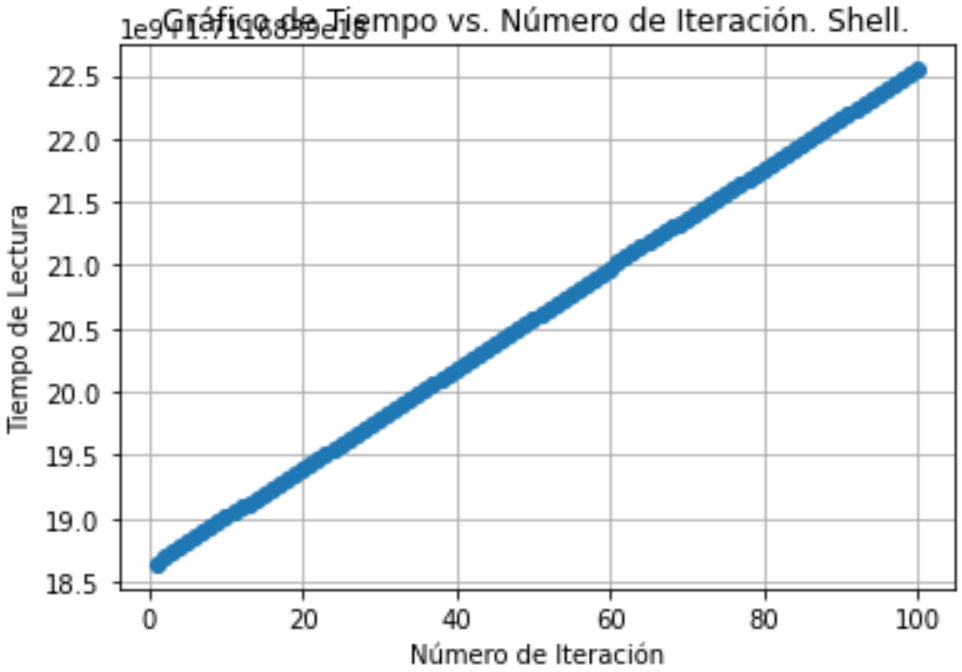
Therefore, when the iteration is carried out, the measurement of each GPIO within the loop will be identified and assigned, making use of the cat command line within the directory observed in the following image.

```
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
```

Graph between Delta time vs iterations.



Graph between read time vs. iteration.



➢ Python

Regarding the Python programming language, first, libraries were imported to be able to manipulate the time and inputs provided on the keyboard at the time of program execution. Also, define the assignment of the GPIOs involved by means of variables.

```
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
```

Subsequently, the given input must be verified, as it is only possible to accept "config", "value" and "close", checking the length of the given text.

```
# 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
```

If the entry is "config", the defined GPIOs will be iterated within a list to be able to export and use each of the pins, so functions were created and defined to be able to make a more complete and efficient code, where the functions created were export_gpio and 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")
```

This function allows export_gpio to check if the directory is open and created, so it will allow you to create, if it does not exist, and modify the directory, so it will write each of the GPIOs used.

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

In the function set_direction will verify and make the address of each folder within the directory, allowing you to locate and create each of the GPIOs to be used.

```
# 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)
```

If the entry is "close", it will iterate through the list of GPIOs to use so that it makes use of the unexport_gpio function for each 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)
```

This feature allows you to write, de-export, and manipulate the directory of GPIOs so that GPIOs can be disabled for use.

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

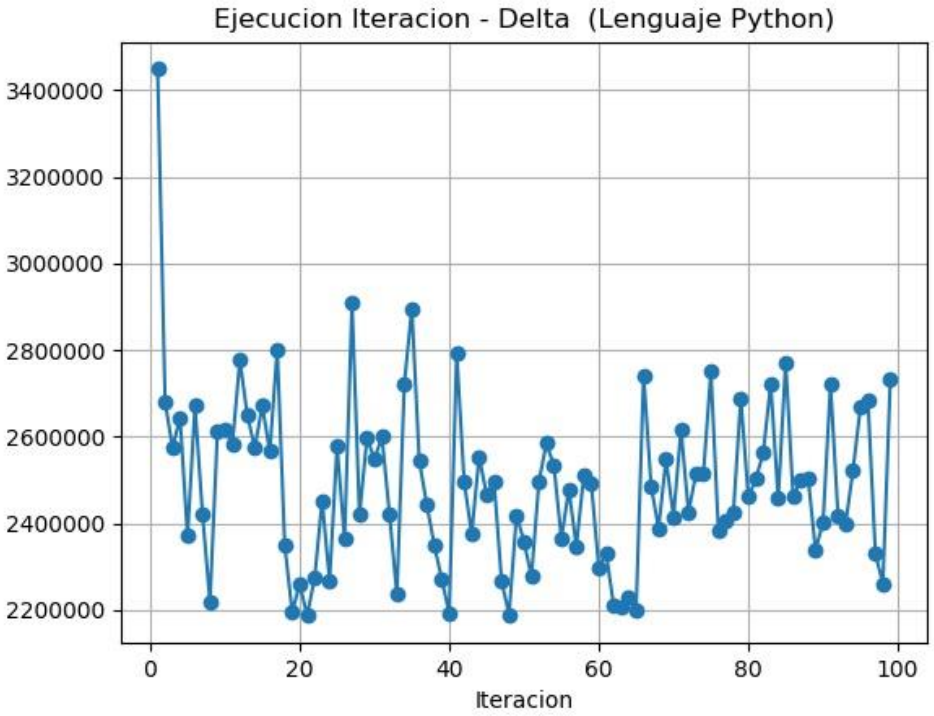
And if the input becomes "value" it will create files where it will contain the given values regarding the times and the values given in each iteration. Subsequently, perform the 100 iterations with the respective time according to each GPIO, manipulating the files to write what was requested.

```
# 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")
```

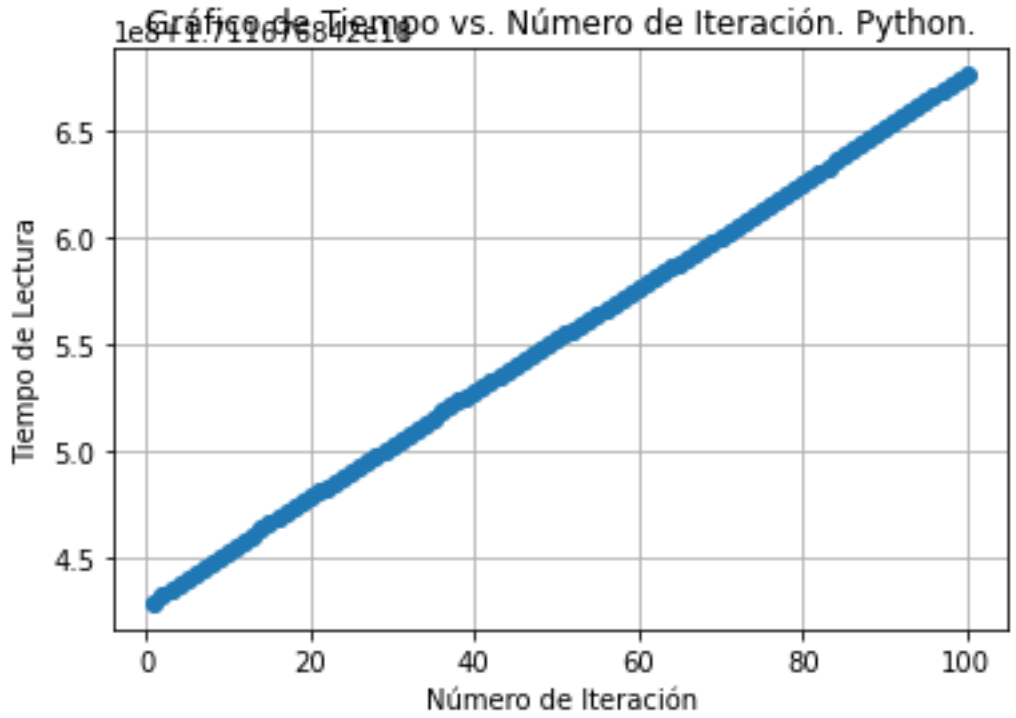
The read_gpio function within the elif of the "value" input gives us the possibility to read what each pin of the Raspberry registers, giving the characteristic of the voltage flow and usage within the circuit, allowing the desired readings to be recorded.

```
# 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()
```

Graph between Delta time vs iterations.



Graph between read time vs. iteration.



➢ C

Regarding the C programming language, first, we start with the declaration of the libraries to be used during the development of the coding and also, define the variables regarding the GPIOs to be used in order to manipulate efficiently within the code.

```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
```

Later in the main function, the input provided will be identified as input to manipulate the code flow, so if the input is not "config", "value" or "close", then an error message will be given and the supported inputs will be notified.

```c
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);
}
```

If the input is "config", the function is called export_gpio with each respective parameter, where that parameter is each previously defined GPIO.

```
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);
```

When using the export_gpio function, this function will make a file with the defined directory and write in the created file the pin to be used for the execution of the encoding, as well as the location where the assignment of each of the GPIOs to make use of them will be located.

```
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);
}
```

In case the input is "close", the unexport_gpio function will be called with the parameter of each GPIO assigned above.

```
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);
```

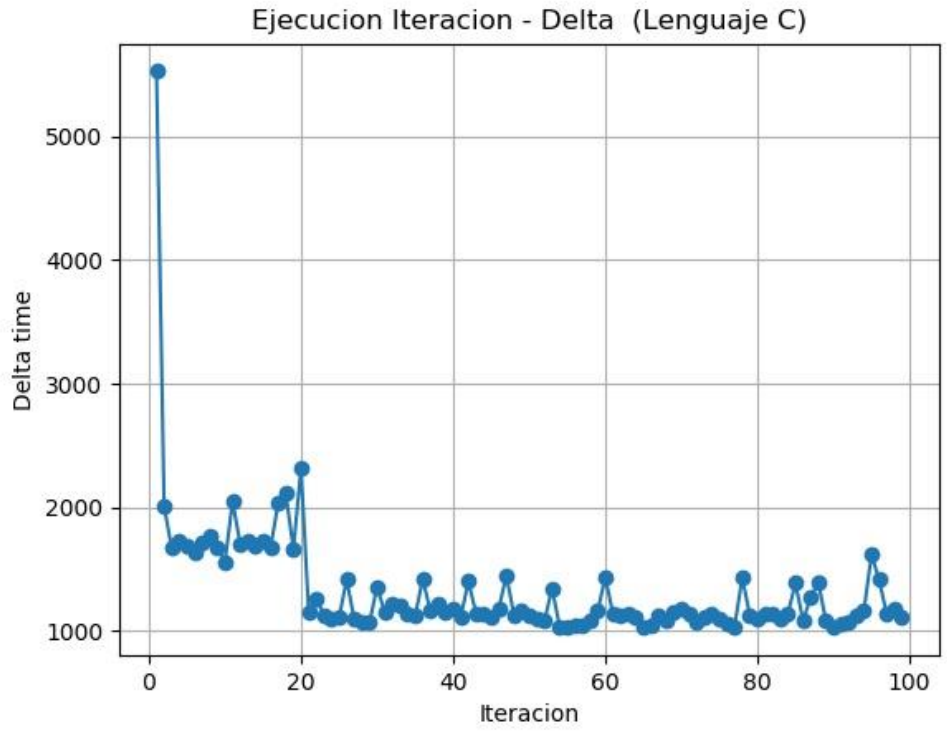
In the unexport_gpio function, we will be placed in the defined address to be able to write to it and be able to disable the use of GPIOs, defined within the unexport directory.

```
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);
}
```

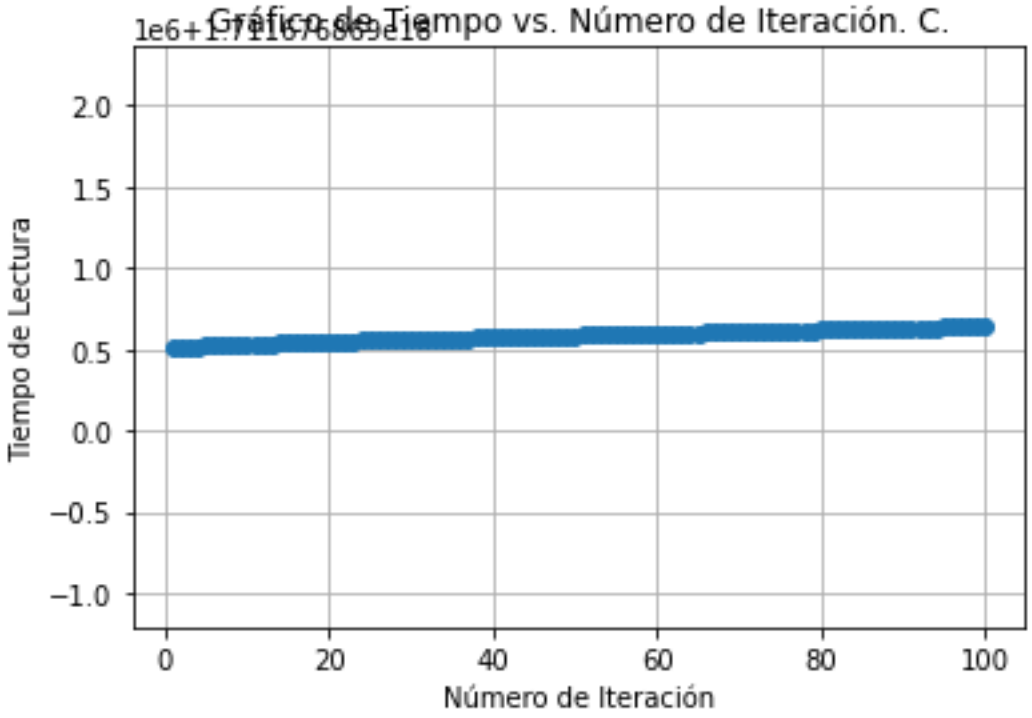
And if the entry is "value", an array of GPIOs is defined to be used for the use of a for cycle to overwrite within the directory for the creation of each folder of the GPIO and to be able to contain in it the readings necessary for the analysis. In the same way, the creation and writing of tiempoC.txt and valorC.txt files for the storage of the necessary data, adding the taking of time during execution, is already necessary in a set of libraries to be able to perform the measurement tasks within the execution of the "value" cycle.

```
    } 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);
```

Graph between Delta time vs iterations.



Graph between read time vs. iteration.



➢ C++

For programming in C++ language, it starts with the definition of the libraries to be used, so you can see below an image of those used within the code.

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

In the main function, the type of input provided is verified, so it will verify what has been entered and subsequently, defining the pins to be used during the execution of the code.

```
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};
```

When the given input is "config" a for cycle is performed to iterate through the previously defined array and be able to make use of the exportAndConfigureGPIO function for the use of GPIOs.

```
    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);
        }
    }
```

Within the exportAndConfigureGPIO function, the given directory will be manipulated to be able to make use of the GPIOs and be able to write to them to read the given data in each 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;
    }
```

If the input is "close", then it will run a for loop to be able to iterate over each of the GPIOs within the previously defined list and thus make the unexportGPIO function call.

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

The unexportGPIO function checks the directory given to write to it in order to declare that those used pins will no longer be used and will therefore be disabled for GPIO manipulation.

```
    // 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;
        }
    }
```

If the input is "value" a series of lines will be made where it will create the text files where it will contain the read data about the corresponding execution times as the values given by each GPIO,

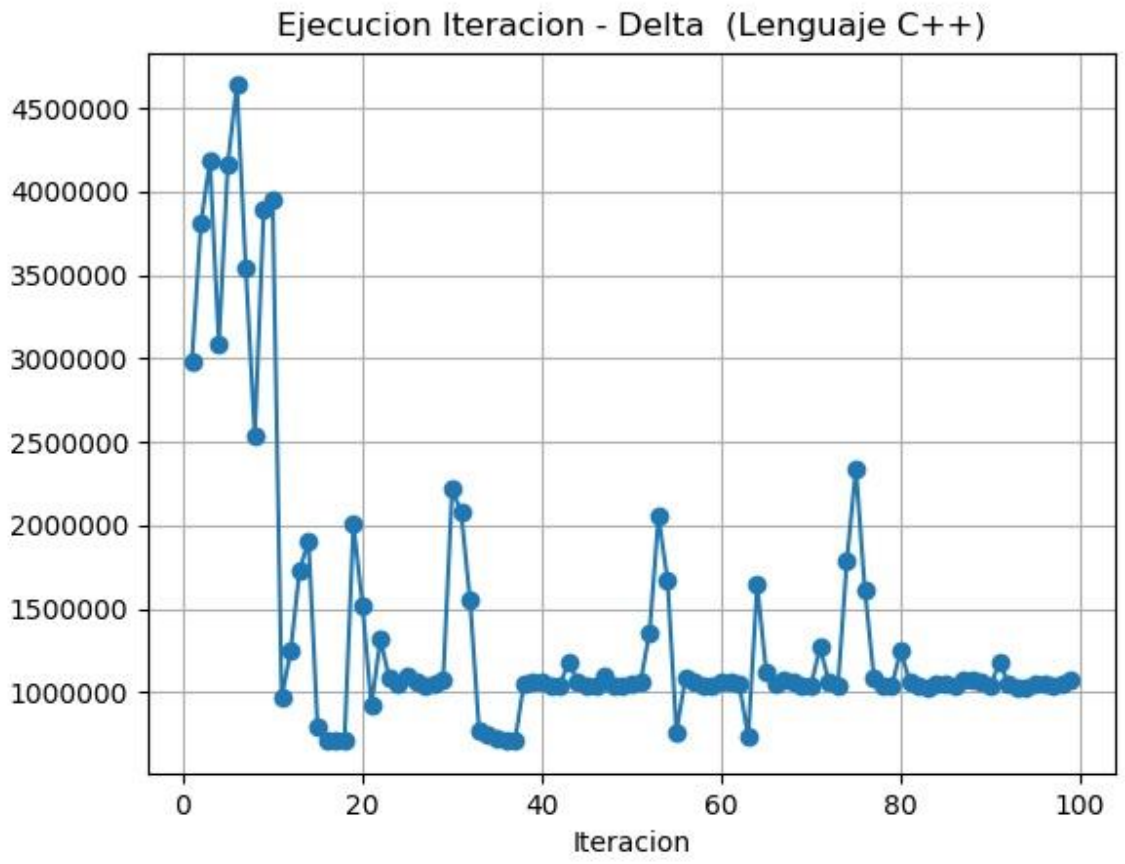
then it will be iterated 100 times to perform the execution of reads with the help of the readGPIO function.

```
} 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();
    }
```

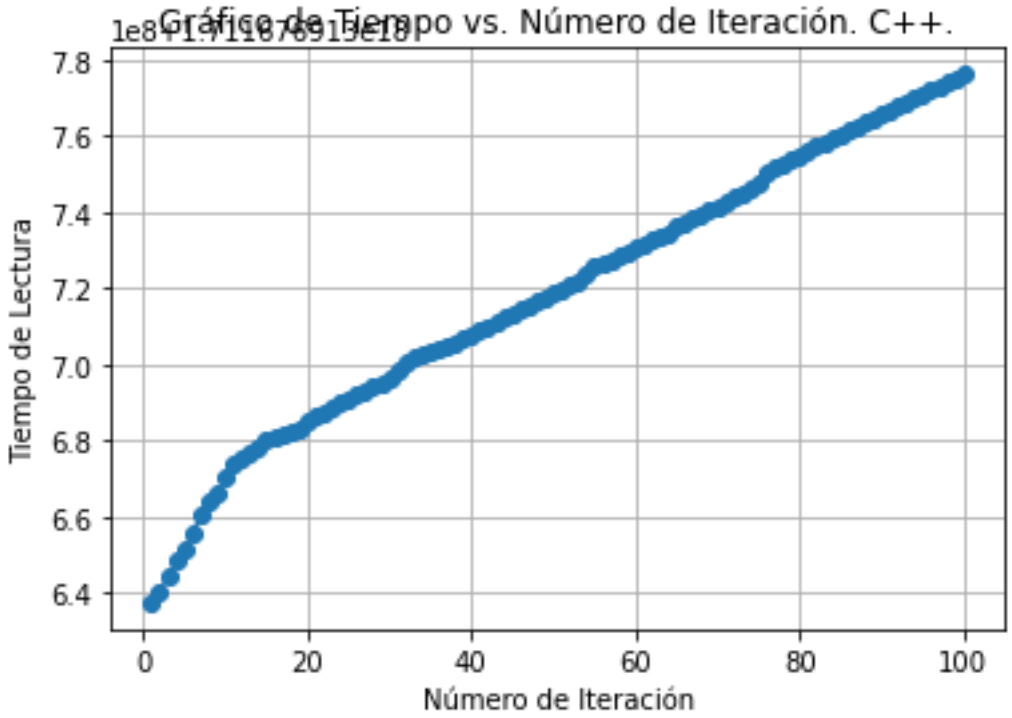
This readGPIO function gives us the possibility to position each GPIO to be used within the directory and thus be able to perform the desired readings and write inside each pin about its location.

```cpp
// 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;
    }
}
```

Graph between Delta time vs iterations.



Graph between read time vs. iteration.



Conclusion.

Cruz Martinez Raul

The development of this project had two main parts: the construction of a physical interface that we saw represented with a circuit composed of comparators, optocouplers and LEDs, and the second was the communication of this physical interface with the RaspberryPi through the Digital Analog Converter, which discretized the continuous voltage values provided by the interface.

In the first part, thanks to the fact that the diagrams were well represented, there was no problem for the construction of the interface; In the same way, connecting the circuit to the GPIOs of the RaspberryPi was uncomplicated, so in doing so we were able to run the first tests of the DAC with the Shell code.

The next part of the project was to read the interface with 4 different programming languages: Shell programming, Python, C++ and C, the latter as a proposed language. With Shell, Python and C++ there was no problem, as it was enough to install the necessary libraries and dependencies for each language. For the proposed language, it was initially contemplated that it would be Fortran 90, but, when working with this language, we realized that there are not the necessary dependencies to be able to use this programming language for the purpose we needed, this led us to change to the C language, where there were no problems for its use, mainly because working with Linux, which is an Operating System built in C, already contained most of the necessary libraries and dependencies.

The performance analysis part was the most time-consuming, since adapting the operation of the Shell code to other programming languages involved a good understanding of how each language works. In general, we observe that the language that presents the best performance is the C language, since the time elapsed between each iteration (delta) is shorter and also maintains a constant rhythm or frequency after the 20 iterations, as for C++, it has the second best optimization, although in each iteration it has a relatively constant and predictable change. every 20 iterations a peak or delay in reading is observed; finally, Python and Shell have more noticeable variations, as well as a longer delay between each read, making your graphics messier and more unpredictable.

To conclude, I have to say that the elaboration of this project was very useful to observe how the interconnection of systems is possible, but the most important part was to observe the factors that intervene in obtaining the information, where the most remarkable was the following: choosing a more optimal programming language is a crucial part for the performance of repetitive tasks. as is the C language, but, not only the performance of the system depends on this element, but also on other factors such as the temperature of the RaspberryPi. Having said that, I believe that the objectives of this project have been met.

Mendoza Bolaños, Carlos Gabriel.

For the development of this project, it allowed us to observe the complexity that a system can be to perform various tasks where time, efficiency and results are paramount within the labor market, so being in contact with systems and their configurations allows us to observe the various factors that must be taken into account for the development and maintenance of an embedded system.

So this project, based on Shell coding, gave us the possibility of interacting and manipulating GPIO of a development board such as the Raspberry PI 3, where it was configured to make use of its components and incorporate circuits to perform the readings. It is worth mentioning that no problem was found in the construction or assembly of the circuit, having with the success the values and operation of the LEDs, comparators and optocouplers, so the function and the desired results were continuously developed.

Regarding the analysis and comparison given, it should be noted that at first we had planned to use the Fortran programming language, however, we did not obtain a successful result or manipulation of the code for the interaction of the GPIOs, so we wanted to change for the C language due to the ease of manipulation and greater knowledge of the language. When defining the programming languages to be used, such as Shell, Python, C and C++ we realized that each one when developing the coding presents a certain complexity and relationship in the formation of the functions with respect to the main, however, there is knowledge and also with the help of instigation each of the codes was formulated based on the one provided in Shell, So there were some drawbacks with the aggregation of some functions to facilitate the flow of the code.

In general, when analyzing and understanding the results given during 100 iterations, it is visualized that the most optimal and efficient result is the C language, having minimal differences in each iteration and also being the language with the fastest execution with the same task in all the

languages involved. Then the most constant language is the C++ language, however, it is visualized when initializing the execution takes more time, but with the passing of the iterations it takes a more constant value and minimal time difference; where the C++ and C languages are a type of compiled programming language, where the entire program is verified to later build an executable with the content of the code.

On the other hand, the Python language requires less initialization time, but when it passes through iterations it is very variable and does not have any constant that can interpret a pattern, in addition to the fact that its time differences between iterations are greater. And with respect to the Shell language, it requires more time for the execution of the reads and greater distance of time between interactions, being the least efficient of the four languages. These languages are interpreted, so it tells us that it is executed as each line of code develops.

All this development and analysis gives us the entrance to the automation and simplification of tasks in our daily lives, as well as in the implementation and creation of new technologies in products as well as in the performance of more complex tasks.

Villanueva Corona Miguel Angel

Over the course of this project, we've had the opportunity to take a deep dive into what goes into building an embedded system. Our work started with the hardware phase, where we painstakingly assembled a circuit using the specified components. This meticulous process was essential, as it laid the foundation for the Raspberry Pi to accurately interpret the voltage signals on its pins by supplying a specific voltage by delivering a specific voltage. This was essential to move on to the programming stage, where our task was to determine which among the various programming languages available optimally aligned with our goals and requirements.

The decision on which programming language to adopt for this project was not at first glance straightforward. It required a detailed evaluation of processing capabilities and resource management efficiency, where C and C++ stood out for their superior performance. These languages offer granular control over the hardware, which is vital in embedded systems where every processor cycle and every byte of memory is precious. On the other hand, Python was presented as a very good solution for those aspects of the project where speed of development and ease of use were paramount. Its intuitive syntax and collection of libraries simplify the implementation of complex functionalities, although with certain disadvantages in terms of efficiency.

As for Bash, although it is a powerful tool for automating tasks and managing system configurations, its application in the context of embedded systems is limited. Bash is ideal for initialization scripts and maintenance tasks, but it is not designed for embedded application development that requires low-level interactions with hardware or critical response times.

In short, choosing the right programming language is an exercise in balancing multiple factors, including, efficiency, speed of development, ease of maintenance, and scalability. The final decision should be aligned with the goals and needs of the project and the capabilities of the development

team, ensuring that the embedded system not only meets current requirements, but is also robust and adaptable for future demands.

Bibliography.

- GNU Manuals online - GNU Project - Free Software Foundation. (n.d.). https://www.gnu.org/manual/manual.html
- How to control the Raspberry Pi GPIO using C. (2020, September 7). LEARN @ CIRCUITROCKS. https://learn.circuit.rocks/how-to-control-the-raspberry-pi-gpio-using-c
- (n.d.). Digikey.com, de https://www.digikey.com/es/maker/tutorials/2019/how-to-use-gpio-on-the-raspberry-pi-with-c
- Llamas, L. (2023, September 5). What GPIOs are and how to use them in the ESP32. Luis Llamas. https://www.luisllamas.es/esp32-gpio/
- Using GPIOs with Python – Prometec. (n.d.). https://www.prometec.net/usando-los-gpio-con-python/

ANNEX.

In this annex you will find all the codes made and used for the realization of this project.

- Shell

```
#!/bin/bash
# GPIOs used to generate number
GPIO_0=27
GPIO_1=22
GPIO_2=17
GPIO_3=4
GPIO_4=18
GPIO_5=23
GPIO_6=24
GPIO_7=8

# In case any value is not passed
if [ $# -ne 1 ]; then # if there is no argument
  echo "No Command"
  echo "The commands to use are config, value, close"
  exit 2 # Invalid number of arguments
fi

# Configure GPIO as inputs
if [ "$1" == "config" ]; then
      echo "Exporting GPIO number $GPIO_0"
      echo $GPIO_0 >> "/sys/class/gpio/export"
      sleep 1# to ensure it was exported correctly
      echo "in" >> "/sys/class/gpio/gpio$GPIO_0/direction"
      echo "Exporting GPIO number $GPIO_1"
      echo $GPIO_1 >> "/sys/class/gpio/export"
      sleep 1# to ensure it was exported correctly
      echo "in" >> "/sys/class/gpio/gpio$GPIO_1/direction"
      echo "Exporting GPIO number $GPIO_2"
      echo $GPIO_2 >> "/sys/class/gpio/export"
      sleep 1# to ensure it was exported correctly
      echo "in" >> "/sys/class/gpio/gpio$GPIO_2/direction"
            echo "Exporting GPIO number $GPIO_3"
      echo $GPIO_3 >> "/sys/class/gpio/export"
      sleep 1# to ensure it was exported correctly
      echo "in" >> "/sys/class/gpio/gpio$GPIO_3/direction"
            echo "Exporting GPIO number $GPIO_4"
      echo $GPIO_4 >> "/sys/class/gpio/export"
      sleep 1# to ensure it was exported correctly
      echo "in" >> "/sys/class/gpio/gpio$GPIO_4/direction"
            echo "Exporting GPIO number $GPIO_5"
      echo $GPIO_5 >> "/sys/class/gpio/export"
```

```
    sleep 1# to ensure it was exported correctly
    echo "in" >> "/sys/class/gpio/gpio$GPIO_5/direction"
        echo "Exporting GPIO number $GPIO_6"
    echo $GPIO_6 >> "/sys/class/gpio/export"
    sleep 1# to ensure it was exported correctly
    echo "in" >> "/sys/class/gpio/gpio$GPIO_6/direction"
        echo "Exporting GPIO number $GPIO_7"
    echo $GPIO_7 >> "/sys/class/gpio/export"
    sleep 1# to ensure it was exported correctly
    echo "in" >> "/sys/class/gpio/gpio$GPIO_7/direction"
fi

if [ "$1" == "close" ]; then
    echo "closing GPIO $GPIO_0"
    echo $GPIO_0 >> "/sys/class/gpio/unexport"
    echo "closing GPIO $GPIO_1"
    echo $GPIO_1 >> "/sys/class/gpio/unexport"
        echo "closing GPIO $GPIO_2"
    echo $GPIO_2 >> "/sys/class/gpio/unexport"
        echo "closing GPIO $GPIO_3"
    echo $GPIO_3 >> "/sys/class/gpio/unexport"
        echo "closing GPIO $GPIO_4"
    echo $GPIO_4 >> "/sys/class/gpio/unexport"
        echo "closing GPIO $GPIO_5"
    echo $GPIO_5 >> "/sys/class/gpio/unexport"
        echo "closing GPIO $GPIO_6"
    echo $GPIO_6 >> "/sys/class/gpio/unexport"
        echo "closing GPIO $GPIO_7"
    echo $GPIO_7 >> "/sys/class/gpio/unexport"
fi
if [ "$1" == "value" ]; then
    n=100 #Numeros of experiments
    i=0
        #echo >tiempoShell.txt
        #echo >valorSShell.txt
        RM ./tiempoShell.txt
        RM ./valorShell.txt
    while((i<n))
    do
        bit0=$(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 number=bit0+bit1+bit2+bit3+bit4+bit5+bit6+bit7
        let i=i+1
        t=$(date +%s%N)
                #Visualizacion of each individual value
        ECHO $bit 0
        ECHO $bit 1
        Echo $bit 2
        Echo $bit 3
        ECHO $bit 4
        Echo $bit 5
        Echo $bit 6
        Echo $bit 7
        echo $i
        echo $t>>tiempoShell.txt
        echo $numero>>valorShell.txt
    done
fi
```

- Python

```
        Imports OS
        import time

        # GPIOs used to generate number
        GPIO_0 = 27
        GPIO_1 = 22
        GPIO_2 = 17
        GPIO_3 = 4
        GPIO_4 = 18
        GPIO_5 = 23
        GPIO_6 = 24
        GPIO_7 = 8

        # Function to export a GPIO
        def export_gpio(GPIO):
          with open(f"/sys/class/gpio/export", 'w') as file:
            file.write(str(gpio))

        # Function to configure the address of a GPIO
        def set_direction(GPIO, direction):
          with open(f"/sys/class/gpio/gpio{gpio}/direction", 'w') as file:
            file.write(direction)

        # Function to de-export a GPIO
        def unexport_gpio(GPIO):
          with open(f"/sys/class/gpio/unexport", 'w') as file:
            file.write(str(gpio))
```

```
# Function to read the value of a GPIO
def read_gpio(GPIO):
  with open(f"/sys/class/gpio/gpio{gpio}/value", 'r') as file:
    return file.read().strip()

# Check if any values were passed
if len(os.sys.argv) != 2:
  print("No command")
  print("Commands to use are config, value, close")
  os.sys.exit(2) # Invalid number of arguments

# Configure GPIO as inputs
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"Exporting GPIO number {gpio}")
    export_gpio(GPIO)
    time.sleep(1) # to ensure that it was exported correctly
    print(f"Configuring GPIO number {gpio} as input")
    set_direction(gpio, "in")

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

# Read GPIO Value
elif os.sys.argv[1] == "value":
  n = 100# Numbers of experiments
  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]]
    number = sum(int(bit) for bit in bits)
    t = time.time_ns()
    # Display of each individual value
    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"{number}\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 opening export file");
      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 opening address file");
      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 opening unexport file");
      exit(EXIT_FAILURE);
   }
   fprintf(unexport_file, "%d\n", gpio_num);
   fclose(unexport_file);
}
```

```
int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("No command\n");
    printf("Commands to use are: config, value, close\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], "close") == 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], "value") == 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 opening GPIO value file");
          exit(EXIT_FAILURE);
        }
```

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

        Suppose bit1 always has the value 3.3
        bit_values[1] = 3.3;

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

        Display of each individual value
        for (j = 0; j < 8; j++) {
            printf("%d\n", bit_values[j]);
        }
        printf("%d\n", i);

            Struct Timeval TV;
            gettimeofday(&tv,NULL);

        Save Current Time in 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 opening tiempo.txt file");
            exit(EXIT_FAILURE);
        }
        fprintf(tiempo_file, "%lld\n", t);
        fclose(tiempo_file);

        Save the number in valor.txt
        FILE *valor_file = fopen("valorC.txt", "a");
        if (valor_file == NULL) {
            perror("Error opening valor.txt file");
            exit(EXIT_FAILURE);
        }
        fprintf(valor_file, "%d\n", number);
        fclose(valor_file);
      }
    } else {
      printf("Command not recognized\n");
      printf("Commands to use are: config, value, close\n");
      exit(EXIT_FAILURE);
    }

    return 0;
}
```

- C++

```
#include <fstream>
#include <iostream>
#include <string>
#include <chrono>
#include <thread>
#include <vector>
#include <unistd.h> // For the sleep function

Function to export a GPIO and configure it as input
void exportAndConfigureGPIO(int gpio) {
  std::ofstream file;
  file.open("/sys/class/gpio/export");
  if (file.is_open()) {
    file << gpio;
    file.close();
    sleep(1); Wait 1 second to ensure that it was exported correctly
    file.open("/sys/class/gpio/gpio" + std::to_string(gpio) + "/direction");
    if (file.is_open()) {
      file << "in";
      file.close();
    } else {
      std::close << "Could not configure GPIO number address " << gpio << std::endl;
    }
  } else {
    std::close << "Failed to export GPIO number " << gpio << std::endl;
  }
}
Function to de-export a GPIO
void unexportGPIO(int gpio) {
  std::ofstream file;
  file.open("/sys/class/gpio/unexport");
  if (file.is_open()) {
    file << gpio;
    file.close();
  } else {
    std::close << "Failed to close GPIO number " << gpio << std::endl;
  }
}

Function to read the value of a 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::close << "Could not read the value of GPIO number " << gpio << std::endl;
    return -1;
  }
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
    std::close << "Usage: " << argv[0] << " [config|close|value]" << 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 << "Exporting GPIO number" << gpio << std::endl;
      exportAndConfigureGPIO(gpio);
    }
  }
  if (command == "close") {
    for (int gpio : gpios) {
      std::cout << "Closing the GPIO" << gpio << std::endl;
      unexportGPIO(gpio);
    }
  } else if (command == "value") {
    const int n = 101; Number of experiments
    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 number = 0;
      for (int gpio : gpios) {
        int bit = readGPIO(gpio);
        std::cout << bit << std::endl;
        number += bit;
      }

      auto t = std::chrono::high_resolution_clock::now();
      auto duration = std::chrono::d
uration_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 << number << std::endl;
        }

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

    return 0;
}
```