UNIVERSIDAD DE LOS ANDES

ESCUELA DE INGENIERÍA ELÉCTRICA

LABORATORIO DE ELECTRONICA II

DEPARTAMENTO DE ELECTRÓNICA Y COMUNICACIONES

INTEGRANTES: TULIO MOLINA, CI: 24608773; JUAN ORTEGA, CI: 24879443

PROYECTO

**Planteamiento del Problema**

En orden de diseñar y construir un electrocardiógrafo totalmente funcional se construirá un circuito simplificado de este mismo, concibiéndose como un osciloscopio digital, este deberá cumplir con las siguientes características:

1. Tener tres escalas de voltaje para medir: 2Vpp, 5Vpp y 20Vpp
2. Trabajar entre las frecuencias de 1Hz y 2kHz.
3. Graficar la señal en un sistema computacional.

**Objetivos:**

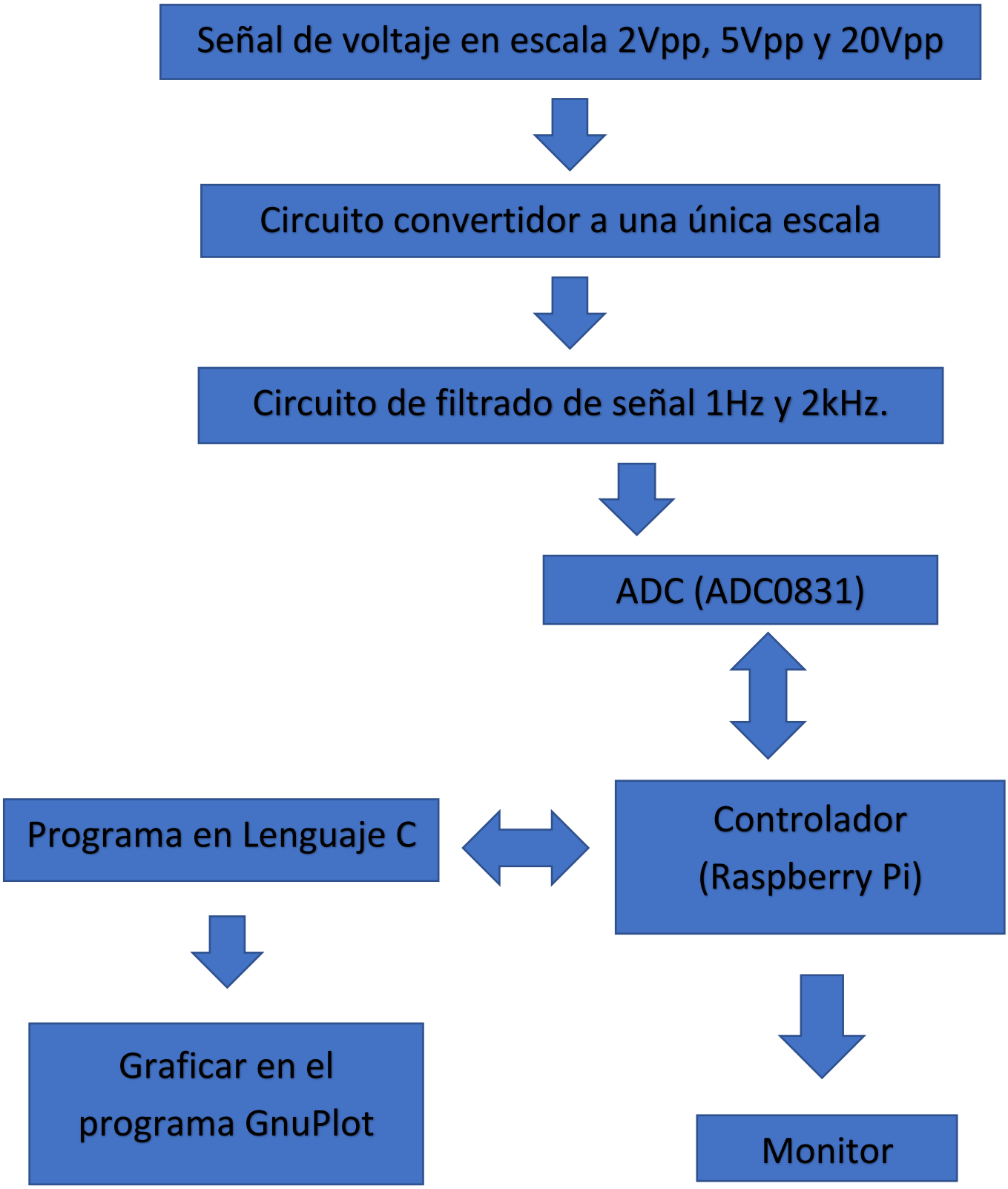
**Objetivo General:**

- Diseñar y construir un osciloscopio digital.

**Objetivos Específicos:**

- Diseñar un circuito capaz de recibir varias señales de voltaje de diferente escala y llevarlas a una escala especifica.
- Diseñar un filtro de frecuencias paso banda entre 1Hz y 2kHz.
- Implementar un ADC y diseñar una interfaz con un equipo programable.
- Diseñar un sistema computacional que grafique la señal de voltaje tomada.
- Implementar todos los circuitos anteriores en conjunto.
- Comprobar funcionamiento.

**Diagrama de bloques:**

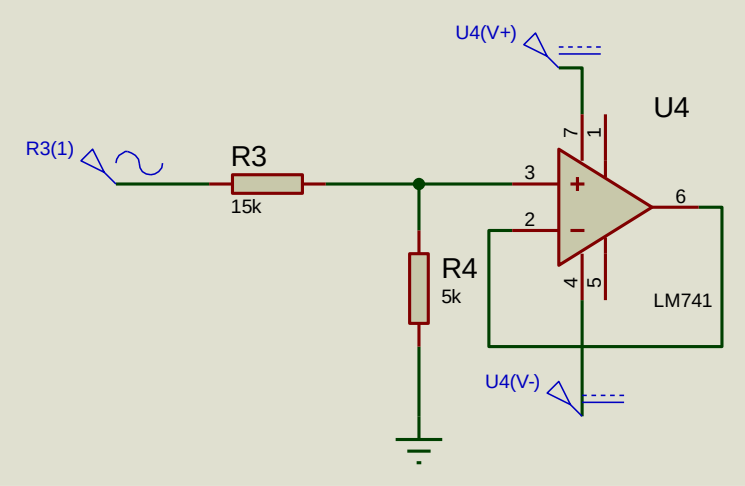


Descripción del proyecto

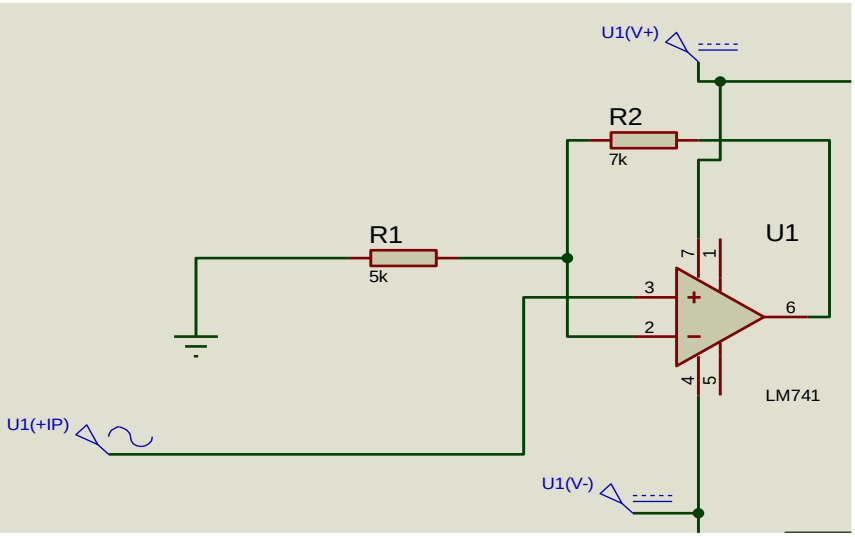
**Etapa 1, Acondicionamiento de la señal**

En esta etapa inicial, es recibida la señal de entrada, donde se debe elegir manualmente el rango de amplitud entre 2,5 V, 10 V, o 1 V. las señales de 2,5 V pasan directamente a la etapa de filtrado, las señales de 10 V deben ser atenuadas hasta 2,5 V y las de 1 V amplificadas a 2,5 V, debido a que el convertidor ADC admite señales de 0 a 5 V, por lo que se puede dividir esta etapa en dos  sub-etapas: atenuación y amplificación de la señal de entrada.

Para la atenuación se utiliza un divisor de voltaje de relación 0,25 V/V, en serie con un amplificador operacional configurado como seguidor de tensión, con el fin de hacer un acople correcto de impedancias entre el filtro y dicha sub-etapa.



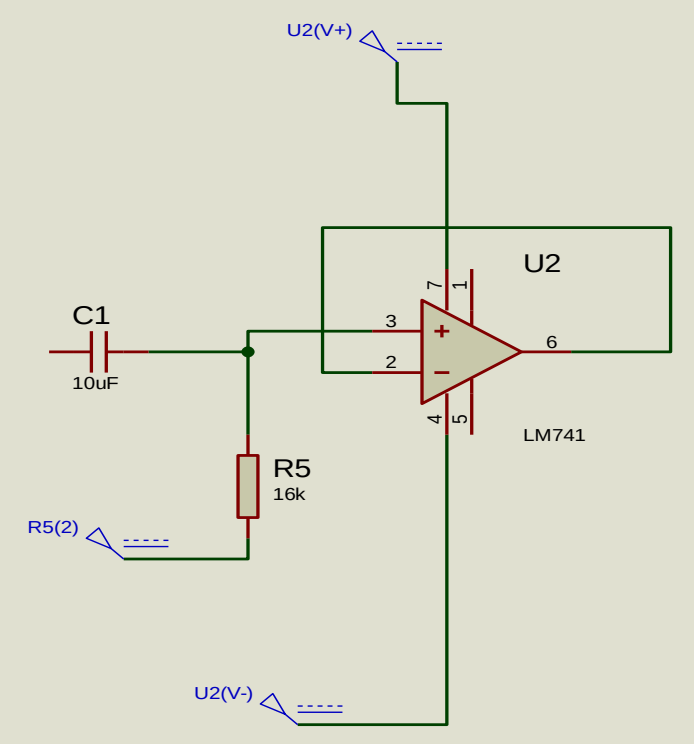

Respecto a la amplificación, se implementa un amplificador operacional no inversor con ganancia de 2,5 V/V.



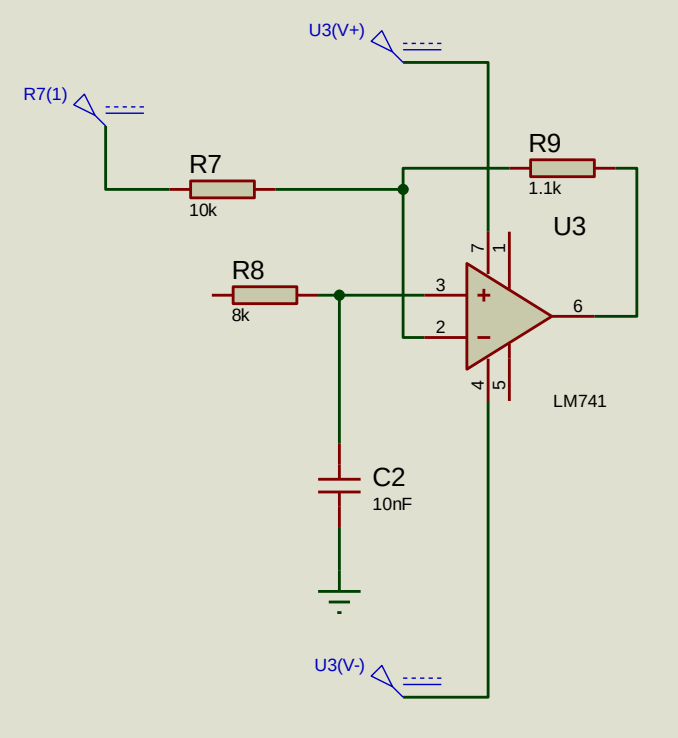
**Etapa 2, filtrado de la señal**

Esta etapa es de gran importancia ya que debido a la aplicación final de este proyecto es necesario eliminar el posible ruido y señales de otras frecuencias existentes en la señal de entrada, para así obtener una medición e imagen de calidad, por lo tanto, se puede dividir en dos sub-etapas más: filtro paso alto y filtro paso alto. Cabe destacar que se implementaron filtros activos Butterworth de primer orden, ya que estos presentan una respuesta bastante plana o estable en la banda de paso y una caída mayor a otros filtros en la frecuencia de corte, pero con un desfase variable en las bandas de frecuencia distintas. Cabe resaltar que como se explicó anteriormente, el ADC recibe señales de 0 a 5 V, por lo que la señal entrante de 5 V pico-pico debe ser desplazada 2,5 V positivamente, para obtener una onda de entrada al conversor en el rango de valores anteriormente descrito.

Para el filtro paso alto se utiliza una frecuencia de corte de 1 Hz, se establece el valor del capacitor en 10 uF y según los cálculos se obtiene la resistencia de 16 kΩ; este filtro no aporta ganancia.



Respecto al filtro paso bajo, la frecuencia de corte establecida es de 2 kHz, se define el valor de capacitancia de 10 nF y mediante los cálculos se obtienen las resistencias de 10 kΩ, 8 kΩ y 1,1 kΩ, esta última con el fin de obtener ganancia de voltaje, debido a que fue observado mediante las simulaciones y el mismo montaje práctico que toda la etapa de filtrado presenta una pequeña caída de voltaje por sus impedancias, la cual se compensa exitosamente con esta ganancia calculada de 1,1 V/V.




El desplazamiento de 2,5 V de la señal de salida de esta etapa que es conectada al ADC se hace colocando 2,5 V en serie con las resistencias R7 y R5 de los filtros paso bajo y paso alto respectivamente, ya que este offset no puede ser sumado a la señal que entra a los filtros debido a que dicha señal DC es eliminada por el filtro paso alto.

**Etapa 3, ADC:**

En está etapa interactúan un circuito integrado adc, el controlador y un programa en lenguaje C que es ejecutado por el controlador.

**ADC:**

Se utilizó el circuito integrado ADC0831 de la Texas Instruments, un convertidor analógico a digital de una resolución de 8 bits, utiliza aproximaciones sucesivas para la conversión y convierte voltajes de 0-1V o de 0-5v. El dibujo del circuito integrado está a continuación:

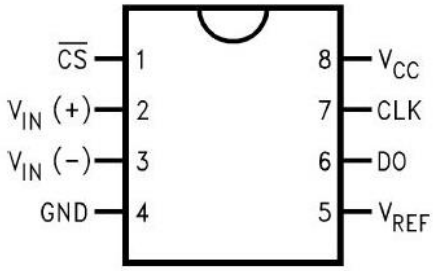
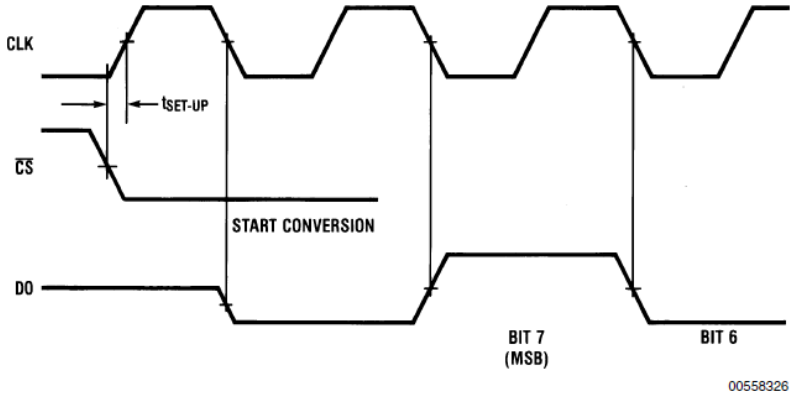


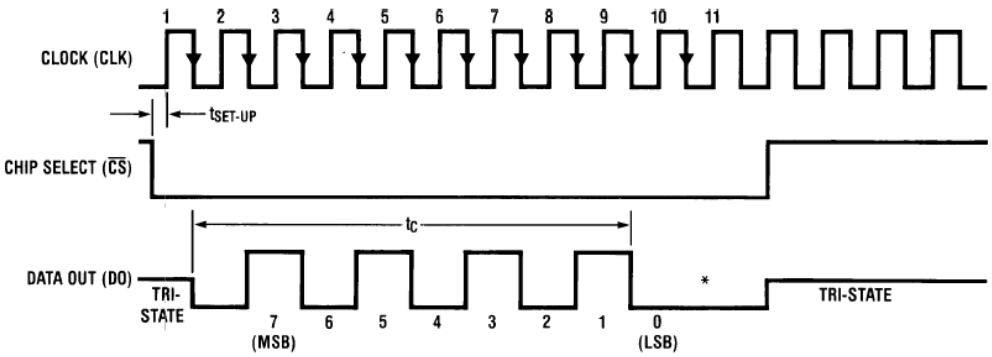

**Figure 4. ADC0831-N Single Differential Input PDIP Package (P) Top View**

El pin GND corresponde a tierra y el Vcc a la alimentación del circuito integrado, la recomendada es 5v. Los adc dan una salida digital que se corresponde a la proporción del voltaje medido con un voltaje fijo, este voltaje fijo es Vref, el cual debe estar entre 1 y 5 V con respecto a tierra. En los pines Vin(+) y Vin(-) se conecta la señal que se medirá, esta no puede superar al Vref ni tampoco 5V. Los pines $\overline{CS}$, CLK y Do son pines de control y dedicados a la transmisión del valor digital convertido, estos pines se explicarán a continuación del diagrama de tiempos del circuito integrado.

**ADC0831 Start Conversion Timing**



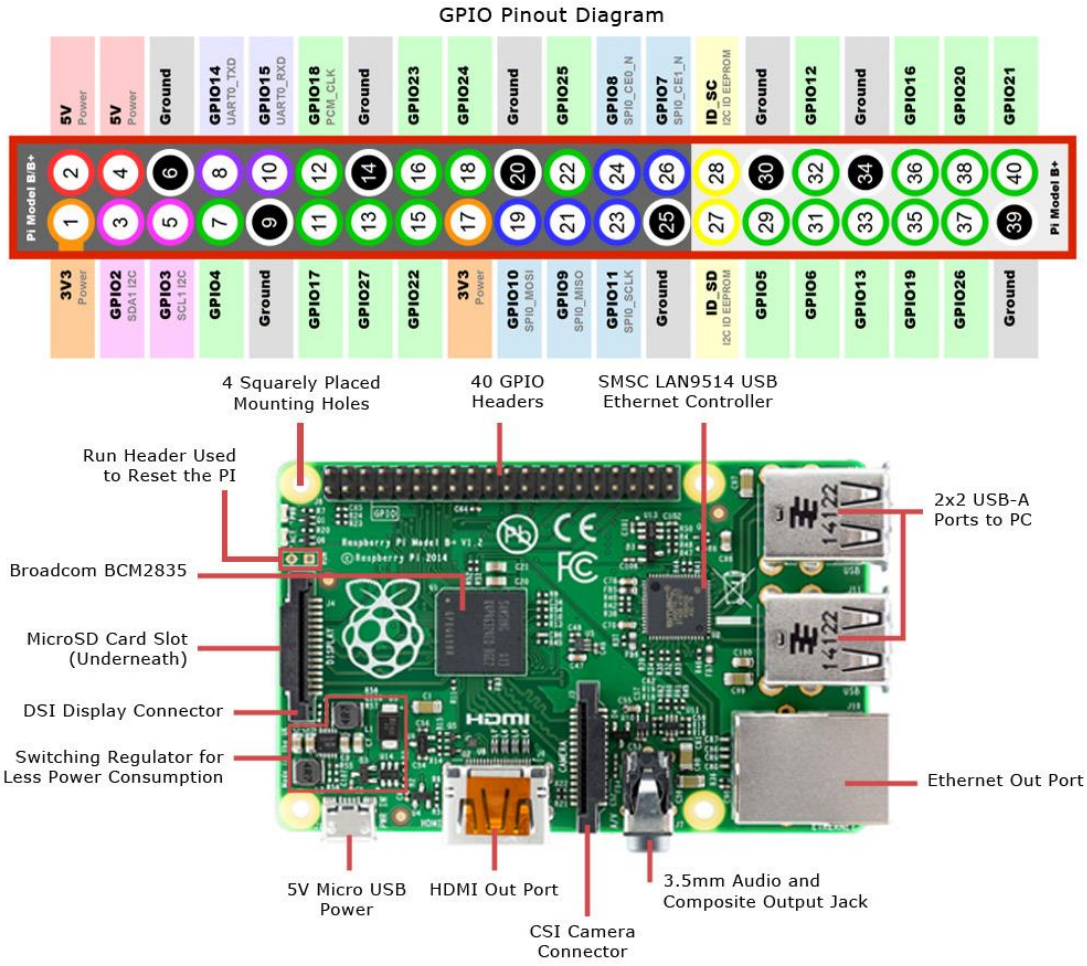

**ADC0831 Timing**



En la gráfica se puede observar el funcionamiento del adc en base a tiempo y sus pines de control y transmisión de datos. El pin $\overline{\mathrm{CS}}$ necesita siempre estar en un alto y al momento de ponerse en bajo se habilita al adc para realizar una conversión, debe estar en bajo durante el tiempo de completo de conversión. Este adc funciona con aproximaciones sucesivas y su salida digital es de forma serial, cada vez que realiza una aproximación él da el valor digital de ese bit por el pin DO. Al colocar el pin $\overline{\mathrm{CS}}$ en bajo se habilita para la conversión, se necesita esperar un tiempo de Tset-up para la estabilidad del adc, después se necesita que por el pin CLK mande un primer clock (tiene que estar inicialmente en bajo, y pasar a alto, seguidamente pasar de bajo a alto), en el flanco de bajada se habilitará el inicio de la conversión, después del flanco de bajada de un segundo clock se tiene el valor del bit mas significativo en el pin DO, continuamente con cada flanco de bajada del clock se actualizará el valor de DO con el valor del siguiente bit hasta llegar al bit menos significativo, por lo cual se necesitarían 8 clocks para transmitir los 8 bits del valor digital. En total se necesitan 10 ciclos de clock para toda la conversión, mas un tiempo de espera de Tset-up y medio clock de espera que necesita el adc al terminar de pasar el último bit. Al terminar una conversión el pin $\overline{\mathrm{CS}}$ debe devolverse a un alto y el CLK a un bajo hasta la próxima conversión.

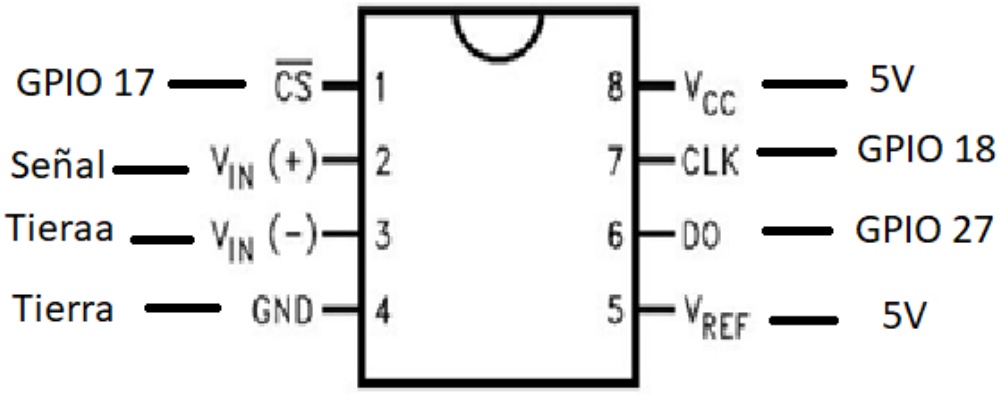
**Controlador:**

El controlador utilizado es un Raspberry Pi, este equipo es una computadora de reducido tamaño, el cual soporta un sistema operativo y en este caso tiene instalado una distribución Linux llamada Raspbian, tiene una variada cantidad de puertos como USB, HDMI y otros, además de 40 pines para interactuar con elementos externos. Estos pines se controlan mediante código escrito en algún lenguaje de programación que ejecute el raspberry y estos son los interactúan con el ADC.



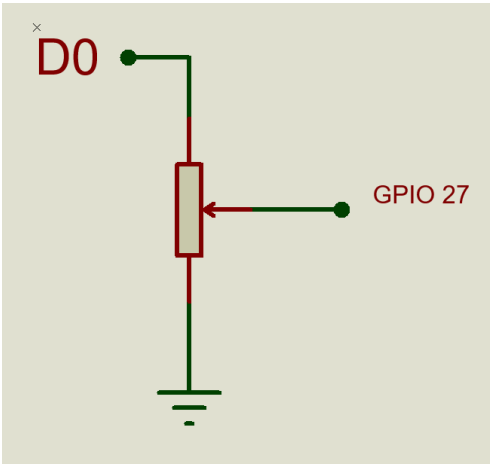

**Programa:**

El programa que ejecuta el raspberry y que interactúa con el ADC está escrito en lenguaje C, hace uso de los pines y para ello utiliza la librería *wiringPi*. Este programa utiliza instrucciones para mandar un pin a estado alto o bajo, con ello mantiene el pin $\overline{\text{CS}}$ en alto y lo coloca en bajo cuando se quiere realizar una conversión, de la misma forma se modula un clock intercambiando el estado de un pin entre alto y bajo cuando corresponde, además se lee de un pin si está recibiendo un alto o un bajo y así se toma la medida digital del ADC a través del pin DO. A continuación, se encuentran los pines de conexión entre el Raspberry y el ADC.




Conexión del ADC0831 con el Raspberry Pi3

**IMPORTANTE:** Los voltajes lógicos de los pines del Raspberry Pi3 son 0 y 3,3V, y los voltajes lógicos del ADC0831 son 0 y 5V. Los pines $\overline{CS}$ y CLK son salidas del raspberry y entradas para el ADC, los 3.3V del raspberry son recibidos como un alto por el ADC sin problemas. Por otro lado, el pin DO es una salida del ADC y una entrada para el raspberry, así que los 5V de salida del ADC entran en un puerto del raspberry que está capacitado para soportar un máximo de 3.3V, así que es necesario realizar un simple divisor de tensión con resistencias a la salida de D0 y conectar allí el pin del raspberry, ese divisor se muestra a continuación:




Utilizando un potenciómetro de 5kΩ donde se coloca una resistencia entre la pata del GPIO y la de tierra de 3,3kΩ se acoplan los voltajes correctamente. En el caso del circuito implementado se observó que el valor alto de salida del pin D0 era de 4V, por lo cual el valor que llegaba al GPIO era menor a 3,3V y no lo lograba medir exitosamente como un alto, así que se aumentó de 3,3kΩ a 4kΩ el valor de la resistencia entre la pata del GPIO y la de tierra. Se utiliza este valor de 5kΩ para

limitar la corriente del puerto D0, no se recomienda usar valores menores. Se pueden utilizar resistencias fijas con estos mismos valores en vez de un potenciómetro.

A continuación, se encuentra un código en C que realiza continuamente 100 conversiones con el menor tiempo posible entre una conversión y otra, imprime estas 100 medidas (en sus valores digitales, no su voltaje correspondiente) y espera 500ms para las siguientes 100 conversiones.

```
#include <stdio.h>    // Used for printf() statements
#include <wiringPi.h> // Include WiringPi library!

// Pin number declarations. We're using the Broadcom chip pin numbers.
const int PIN_CLK = 18; // CLK - Board pin 12 -- BCM pin 18
const int PIN_CS = 17; // CLK - Board pin 11 -- BCM pin 17
const int PIN_DO = 27; // CLK - Board pin 13 -- BCM pin 27

char conv(char dato,char medclk, char j)
{
        dato = 0;
        digitalWrite(PIN_CS, LOW);
        delayMicroseconds(1);
        digitalWrite(PIN_CLK, HIGH);
        delayMicroseconds(medclk);
        digitalWrite(PIN_CLK, LOW);
        delayMicroseconds(medclk);
        digitalWrite(PIN_CLK, HIGH);
        delayMicroseconds(medclk);
        digitalWrite(PIN_CLK, LOW);

        for(j = 0; j < 8; j++){
                delayMicroseconds(medclk);
                dato = dato << 1;
                if(digitalRead(PIN_DO))
                        dato = dato | 0x01;
                digitalWrite(PIN_CLK, HIGH);
                delayMicroseconds(medclk);
                digitalWrite(PIN_CLK, LOW);
        }
        delayMicroseconds(medclk);
        digitalWrite(PIN_CS, HIGH);
        return dato;
```

```
}

int main(void)
{
	// Setup stuff:
	wiringPiSetupGpio(); // Initialize wiringPi -- using Broadcom pin numbers

	//Declarando Pines
	pinMode(PIN_CLK, OUTPUT);
	pinMode(PIN_CS, OUTPUT);
	pinMode(PIN_DO, INPUT);

	//Poniendo condiciones iniciales
	digitalWrite(PIN_CLK, LOW);
	digitalWrite(PIN_CS, HIGH);

	char dato, j, vect[100], medclk = 1;
	int i;

	while(1){
		for(i = 0; i < 100; i++){
			vect[i] = conv(dato,medclk,j);
			delayMicroseconds(1);
		}
		for(i = 0; i < 1000; i++){
			printf("%d ",vect[i]);
		}
		printf("\n\n");
		delay(500);
	}

  return 0;
}
```

En el código, la función “conv” es la función que realiza una conversión y retorna el valor digital que entrega el ADC. El bucle infinito “while(1)” es el que toma 100 medidas y las imprime continuamenta, guardando el valor que entrega la función en el vector “vect[]”. Todo lo demás son declaraciones de librerías, asignaciones de pines y funciones de la librería *wiringPi*. El clock funciona a un ciclo de servicio del 50%.

Después de diferentes pruebas realizadas se determinaron cuáles son los tiempos mínimos para realizar una conversión exitosamente y el tiempo mínimo entre una conversión y la siguiente:

- Tset-up = 1us
- Tclock (Perido clock) = 2us
- Tiempo entre una conversión y otra = 1us

El tiempo que el clock pasa en alto o en bajo entonces es: 1us, este tiempo es declarado como "medclk**"** en el código.

Al sumar los tiempos, una conversión completa toma 21us y si se le suma 1us de tiempo entre una conversión y otra se tiene el periodo de muestreo, el cual sería 22us y esto da una frecuencia máxima de 45454Hz. Si se quiere reducir la frecuencia de muestreo se tiene que alargar el tiempo de espera entre una conversión y otra.

En la implementación completa del circuito se utiliza la frecuencia máxima de muestreo y para graficar se cambia solo el número de puntos que se tomarán (conversiones que se realizarán), de modo que se pueda observar la señal eficientemente, esto se explicará más adelante.

**Etapa 4, descripción y funcionamiento del código**

Se decide implementar el programa en lenguaje C, ya que aunque inicialmente se implementaría en Python, pues presenta facilidad mucho mayor a la hora de codificar, fueron hechas pruebas de velocidad de respuesta, haciendo que cada uno de los lenguajes haga un cambio de estado de alto a bajo en un bucle infinito, y se obtuvo que mientras Python tarda 5,4 us en ejecutar una iteración, C hace la misma operación en tan solo 66 ns, por lo que Python limita en gran medida la frecuencia de muestreo del osciloscopio; siendo observadas dichas mediciones en uno de los osciloscopios del laboratorio.

El programa se divide en 6 segmentos o bloques

1. Declaración de las librerias y pines de interacción entre el raspberry y el ADC, como se explica en la etapa anterior.
2. Creación de la función que ejecuta la conversión de una muestra, también explicado en la etapa anterior.
3. Inicio de la función principal (main) donde se configuran los pines anteriormente declarados, y se establecen las condiciones iniciales de pines de clock y $\overline{\mathrm{CS}}$ esperando para hacer el primer muestreo. Luego se calcula el tamaño máximo del vector donde se guarda la cantidad de puntos a graficar en cada iteración del programa, mediante la fórmula $num\_puntos = Muestras\_periodo * 4$ para una frecuencia de 3 Hz, siendo este el caso donde ocurre el valor máximo de puntos por gráfica, este proceso se explica en el siguiente párrafo. Además, se declaran las

variables y vectores donde almacenan los datos de voltaje “MuestrasVoltaje[num_puntos]” y tiempo.

```
int main(void)
{
    // Setup stuff:
    wiringPiSetupGpio(); // Initialize wiringPi -- using Broadcom pin numbers

    //Declarando Pines
    pinMode(PIN_CLK, OUTPUT);
    pinMode(PIN_CS, OUTPUT);
    pinMode(PIN_DO, INPUT);
    pinMode(PIN_BOTON, INPUT);

    //Poniendo condiciones iniciales
    digitalWrite(PIN_CLK, LOW);
    digitalWrite(PIN_CS, HIGH);

     //Calculando el número máximo de puntos que se pueden graficar
     //(num_puntos), el cual sería con le frecuencia de 3Hz,
    //esto debido a la formula que se utiliza. Se hace esto para que el vector
    // tenga
    //el máximo necesario, pero no siempre se usarán todas las casillas.
    int frec = 3;
    int num_puntos;
    double Muestras_periodo = (1/(float)frec)/0.000022;
    num_puntos = Muestras_periodo*4;
    //Variables
    char MuestrasVoltaje[num_puntos], dato, j, medclk = 1;
    register int i=0;
    int PeriodoMuestreo = 1;
    float ValoresTiempo[num_puntos],MuestrasVoltaje_2[num_puntos];
```

Luego se pregunta al usuario la frecuencia aproximada de la señal, en un rango de 1 Hz a 2 kHz, y en base a este valor se calcula el número de puntos que presenta cada gráfica usando primeramente la ecuación $Muestras\_periodo = \frac{1/Frec}{22\ us}$, donde la variable “Muestras_periodo” puntos por un periodo de la señal de entrada que el conversor es capaz de obtener, $22\ us$ es el periodo de la frecuencia de muestreo, y $1/\text{"Frec"}$ el periodo de la señal de entrada. A continuación, se aplica una estructura de decisión CASE, en la que se establece el número de puntos a ser muestreados por cada gráfica dependiendo de la frecuencia de la señal previamente solicitada, a esta variable se le denomina “num_puntos” y se calcula de de la siguiente manera: el

caso 1 para 1 Hz de frecuencia, es directamente “Muestras_periodo” que es aproximadamente 45000 puntos por gráfica, para el caso 2 para 2 Hz, “Muestras_periodo” es aproximadamente 22500, pero para presentar más puntos por gráfica se multiplica este valor por dos, finalmente para las demás frecuencias se multiplica por cuatro la cantidad de muestras por periodo, ya que a medida que la frecuencia de entrada aumenta, disminuye la resolución de la gráfica puesto que las muestras por periodo se van haciendo cada vez menores, obteniéndose tan solo 30 puntos por un periodo para una frecuencia de 2 kHz.

```
//Pidiendole al usuario la primera frecuencia
printf("La frecuencia debe estar ser (1 < frec < 2000). Que frecuencia
        usara: ");
scanf("%d",&frec);
while( (frec < 1) || (frec > 2000) ){
        printf("Introdujo un valor incorrecto. La frecuencia debe ser”)
                printf("(1 < frec < 2000). Que frecuencia usara: ");
        scanf("%d",&frec);
}
Muestras_periodo = (1/(float)frec)/0.000022;
// CASE
switch (frec){
        case 1: num_puntos = (long int) Muestras_periodo + 10;
                break;
        case 2: num_puntos = (long int) Muestras_periodo*2;
                break;
        default:num_puntos = (long int) Muestras_periodo*4;
                break;
        }
```

4. Posteriormente se procede a calcular los puntos a presentar en la primera gráfica, para ello se hacen 3 bucles FOR. En el primero se obtienen los puntos de voltaje en valores digitales de 0 a 255 invocando a la función de conversión en cada iteración, este bucle va desde 0 a la cantidad de puntos por gráfica. Cada uno de estos valores se guardan en “MuestrasVoltaje[]”.

   Para el segundo bucle se convierten los valores del vector “MuestrasVoltaje[]” de datos digitales a analógicos, y se guardan en el vector “MuestrasVoltaje_2[]”, estos valores analógicos si son usados para dibujar la gráfica. Para esto se aplica la siguiente formula $MuestrasVoltaje\_2[i] = \frac{MuestrasVoltaje[i]*5}{255} - 2{,}5$ donde “i” es la variable de iteración que va desde 0 a “num_puntos”, se divide entre 255 el valor de “MuestrasVoltaje[i]” para hacer la conversión antes descrita, y se resta 2,5

para eliminar la componente DC que se agrega a la señal justo antes de entrar al ADC.

El tercer bucle se encarga de generar el vector de tiempo, llamado "ValoresTiempo[]" que define la componente horizontal de cada punto de la gráfica donde dicho vector se define de la siguiente manera: $ValoresTiempo[i] = i * 44\ us$ siendo "i" la variable de iteración como en el bucle anterior, con el mismo rango de variación.

```
// TOMANDO PRIMERA MUESTRA
        for(i = 0; i < num_puntos; i++){
                MuestrasVoltaje[i] = conv(dato,medclk,j);
                delayMicroseconds(1);
        }

        for(i=0; i<num_puntos; i++){
                MuestrasVoltaje_2[i] = (MuestrasVoltaje[i]*5)/255.0f - 2.5;
        }

        for(i=0; i<num_puntos; i++){
                ValoresTiempo[i] = i*44;
        }
```

5. En este bloque de código se interactúa por primera vez con el programa de graficación llamado GNUPlot, el cual, por ser un programa, mas no una librería, no tiene funciones a las cuales invocar y pasarle directamente los vectores obtenidos anteriormente para trabajar con ellos; en cambio, dicho programa interactúa directamente con el command line del sistema operativo, y su modo de recibir los datos para generar las gráficas es mediante archivos de texto que contienen estos valores. Por lo tanto el código se encarga de generar un archivo .txt y escribir en una columna los valores de "MuestrasVoltaje_2[]" y los de "ValoresTiempo[]" en otra; luego se crea un vector "configGnuplot[]" con las configuraciones que serán también transmitidas al graficador, como título de ejes y escalas, así como el archivo .txt con los valores a graficar; a continuación se crea el archivo tipo popen también conocido como pipeline o tubería, y en este caso llamado "ventanaGnuplot", para ir desde el programa al command line y ejecutar cada una de los comandos establecidos en "configGnuplot[]" mediante un bucle FOR que imprime cada uno de dichos comandos llamando a "ventanaGnuplot" y pasándole cada valor de "configGnuplot" en cada iteración. Al final de este proceso se debe cerrar el archivo

de texto anteriormente abierto, ya que si se deja abierto se producen errores a la hora de escribir de nuevo en este archivo.

```
        /* se crea y se abre el archivo puntosGraficar.txt en modo escritura
 * para almacenar los valores de x y y que están declarados en los arreglos
 * valoresX y valoresY*/
FILE * archivoPuntos = fopen("puntosGraficar.txt", "w");

/*Guardar los puntos x,y en el archivo de texto creado y abierto
        previamente*/
for (i=0; i<num_puntos; i++){
        fprintf(archivoPuntos, "%.2f %.2f \n", ValoresTiempo[i],
                MuestrasVoltaje_2[i]);
}

/*lista de comandos para ejecutar y configurar la visualización que
        tendrán
* los puntos en la gráfica con gnuplot*/
char * configGnuplot[] = {"set title \"OSCILOSCOPIO DIGITAL\"",
                          "set ylabel \"----VOLTAJE (V)--->\"",
                             "set xlabel \"----TIME/DIV 22us--->\"",
                         "plot \"puntosGraficar.txt\" using 1:2 with lines",
                           "set yrange [-2.5:2.5]"
                          };

/*Se crea una archivo de tipo popen, es una tebería IPC que se usa, para
 * ejecutar gnuplot y enviarle el archivo a graficar*/
FILE * ventanaGnuplot = popen ("gnuplot -persist", "w");

// Executing gnuplot commands one by one
for (i=0; i<NUM_COMANDOS; i++){
        fprintf(ventanaGnuplot, "%s \n", configGnuplot[i]);
}

fflush(ventanaGnuplot);
fclose(archivoPuntos);
```

6. Entrando al “while(1)”, se establece un condicional IF al cual se entra cuando el usuario lo desee, presionando un botón en el protoboard, para de esta manera cambiar el valor de frecuencia “frec” aproximado de la señal de entrada, y si esto ocurre se procede a calcular las nuevas variables “Muestras_periodo” y posteriormente “num_puntos” de la misma manera que en el bloque tres. Luego de dicho IF, se repiten los bloques cuatro y cinco, con la única diferencia respecto al

bloque 5 que ahora solo será transmitido al command line a través de “ventanaGnuplot” (archivo popen) la configuración de graficar los valores guardados en el archivo .txt del vector “ventanaGnuplot”, es decir, solo la componente tres de este vector.

```
while(1){
        delay(3000);  //Refrescamiento grafica!!
        printf("NUEVA GRAFICA!! \n");

        if(digitalRead(PIN_BOTON)){ //Boton para cambiar la frecuencia
                // Frecuencia
                printf("La frecuencia debe estar ser (1 < frec < 2000).”)
                        printf("Que frecuencia usara: ");
                scanf("%d",&frec);
                while( (frec < 1) || (frec > 2000) ){
                        printf("Introdujo un valor incorrecto. La frecuencia”);
                        printf("debe ser (1 < frec < 2000). Que frecuencia”);
                        printf(" usara: ");
                        scanf("%d",&frec);
                }
                Muestras_periodo = (1/(float)frec)/0.000022;
                // CASE
                switch (frec){
                        case 1: num_puntos = (long int) Muestras_periodo +
                                        10;
                                break;
                        case 2: num_puntos = (long int)
                                        Muestras_periodo*2;
                                break;
                        default:num_puntos = (long int)
                                        Muestras_periodo*4;
                                break;
                           }
        }

        //TOMANDO OTRA MUESTRA
        for(i = 0; i < num_puntos; i++){
                MuestrasVoltaje[i] = conv(dato,medclk,j);
                delayMicroseconds(1);
        }

        for(i=0; i<num_puntos; i++){
```

```
                MuestrasVoltaje_2[i] = (MuestrasVoltaje[i]*5)/255.0f - 2.5;
        }

        for(i=0; i<num_puntos; i++){
                ValoresTiempo[i] = i*PeriodoMuestreo;
        }

        FILE * archivoPuntos = fopen("puntosGraficar.txt", "w");

        for (i=0; i<num_puntos; i++){
                        fprintf(archivoPuntos, "%.2f %.2f \n",
ValoresTiempo[i],
                        MuestrasVoltaje_2[i]);
        }

        fprintf(ventanaGnuplot, "%s \n", configGnuplot[3]);

        fflush(ventanaGnuplot);
        fclose(archivoPuntos);
}

   return 0;
}
```

**Cálculos**

**Divisor de voltaje de atenuación en etapa 1:**

$$\frac{Vo}{Ve} = 0{,}25 = \frac{R3}{R3 + R4}$$

Si R3 = 5 kΩ entonces R4 = 15 kΩ $\frac{Vo}{Ve} = 0{,}25$

**Amplificación de voltaje de etapa 1:**

Ganancia del amplificador como no inversor:

$$\frac{Vo}{Ve} = 1 + \frac{R2}{R1} = 2{,}5$$

Si R2 = 7 kΩ y R1 = 5 kΩ, la ganancia $\frac{Vo}{Ve} = 2{,}4 \approx 2{,}5$

**Filtro paso alto:**

$$w_c = 2 * \pi * Fc$$

Si Fc = 1 Hz entonces $w_c = 2\pi\ rad/s$

$$w_c = \frac{1}{R5 * C1}$$

Si C1 = 10 uF; entonces R5 ≈ 16 kΩ

**Filtro paso bajo:**

Ganancia de voltaje = 1,1 V/V

$$K = \frac{R9 + R7}{R7}$$

Si R9 = 1,1 kΩ y R7 = 10 kΩ entonces K = 1,1 V/V

Luego:

$$w_c = 2 * \pi * Fc$$

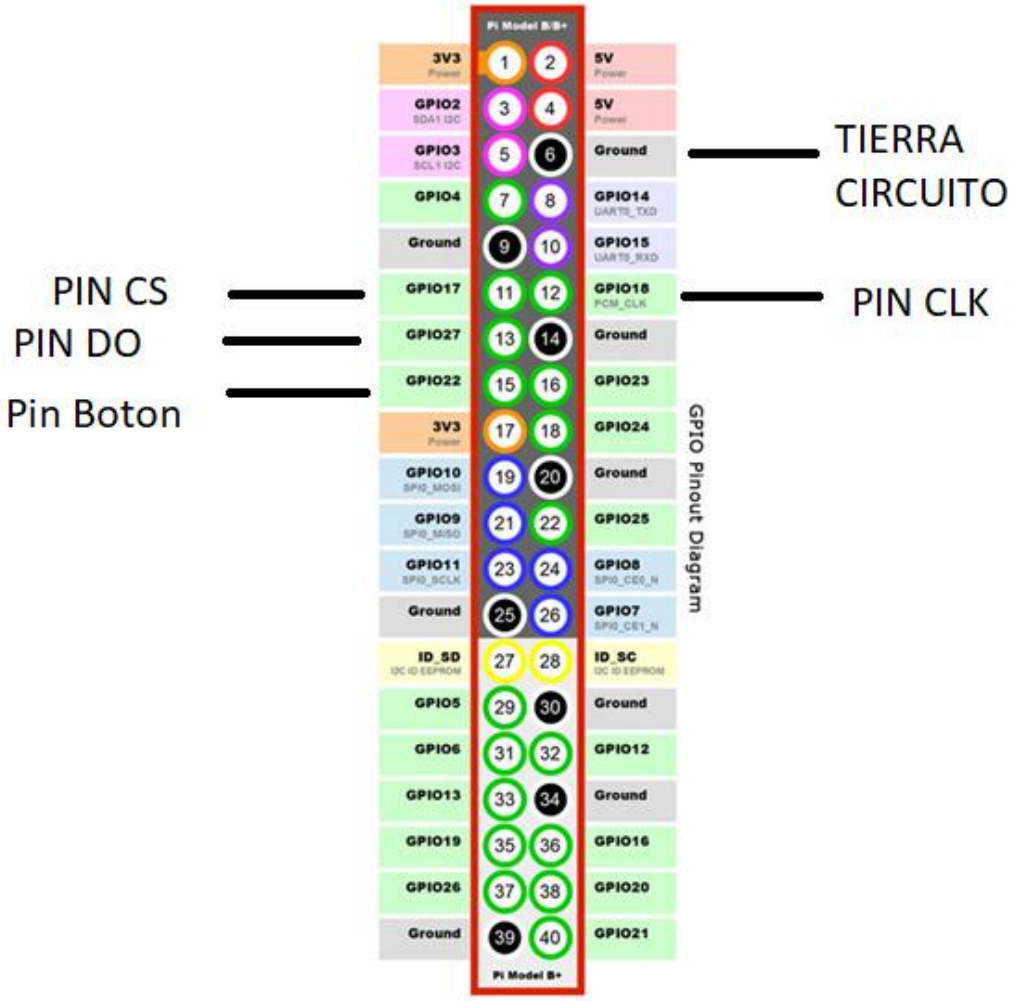
Si Fc = 2 kHz entonces $w_c = 12566{,}37\ rad/s$

$$w_c = \frac{1}{R8 * C2}$$

Si C1 = 10 nF; entonces R5 ≈ 8 kΩ

**Diagramas esquemáticos**

**Conexión Raspberry Pi 3:**



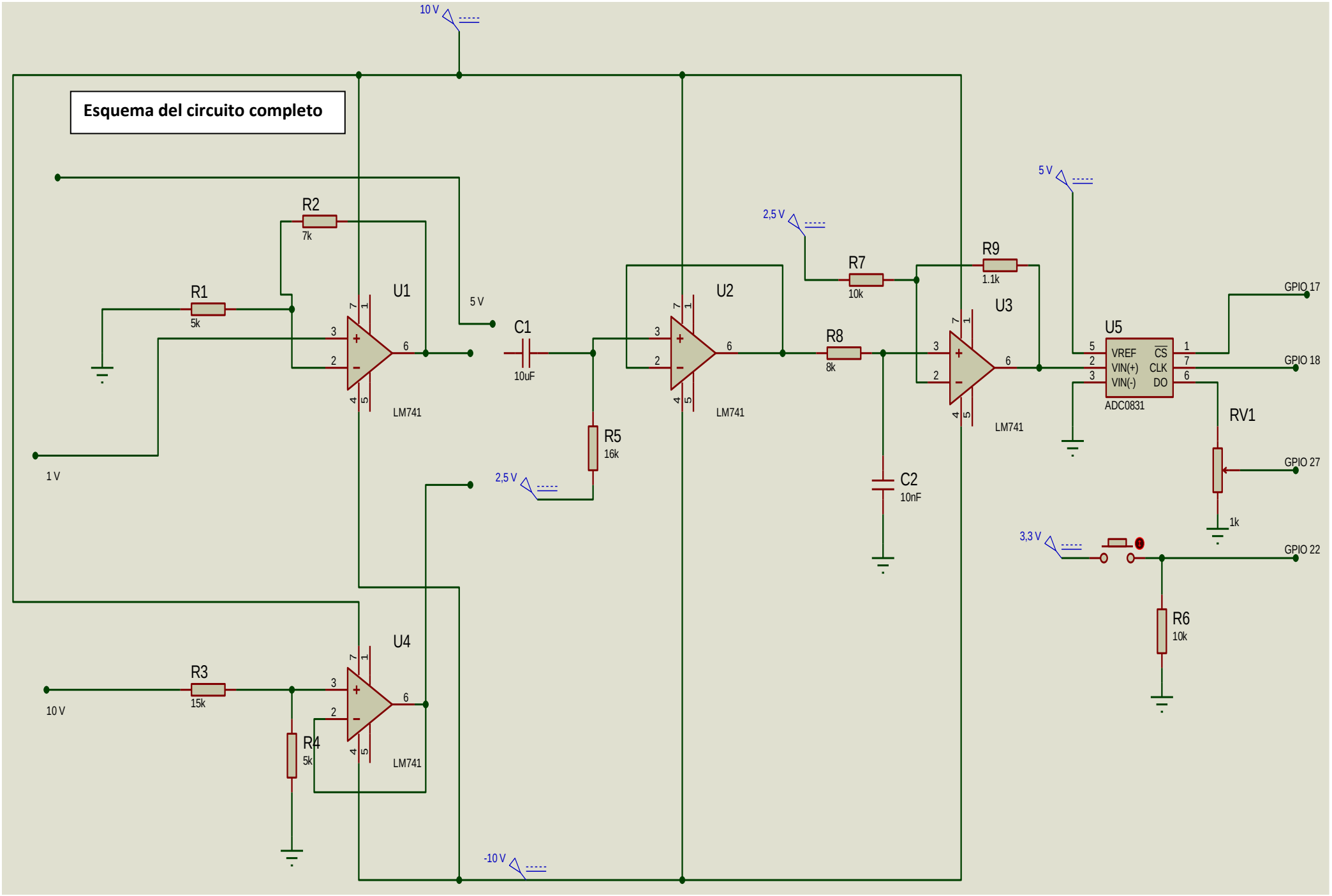
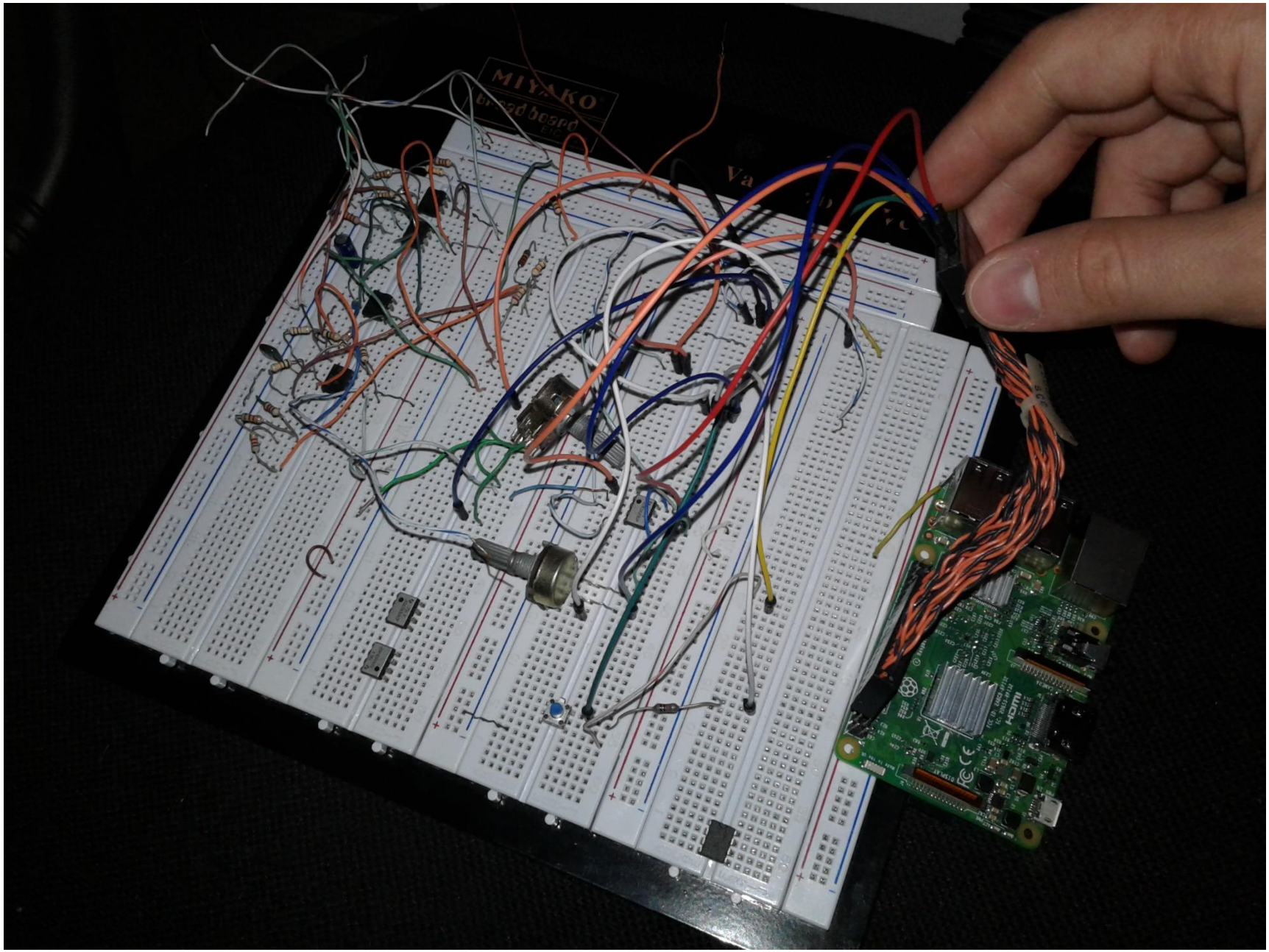
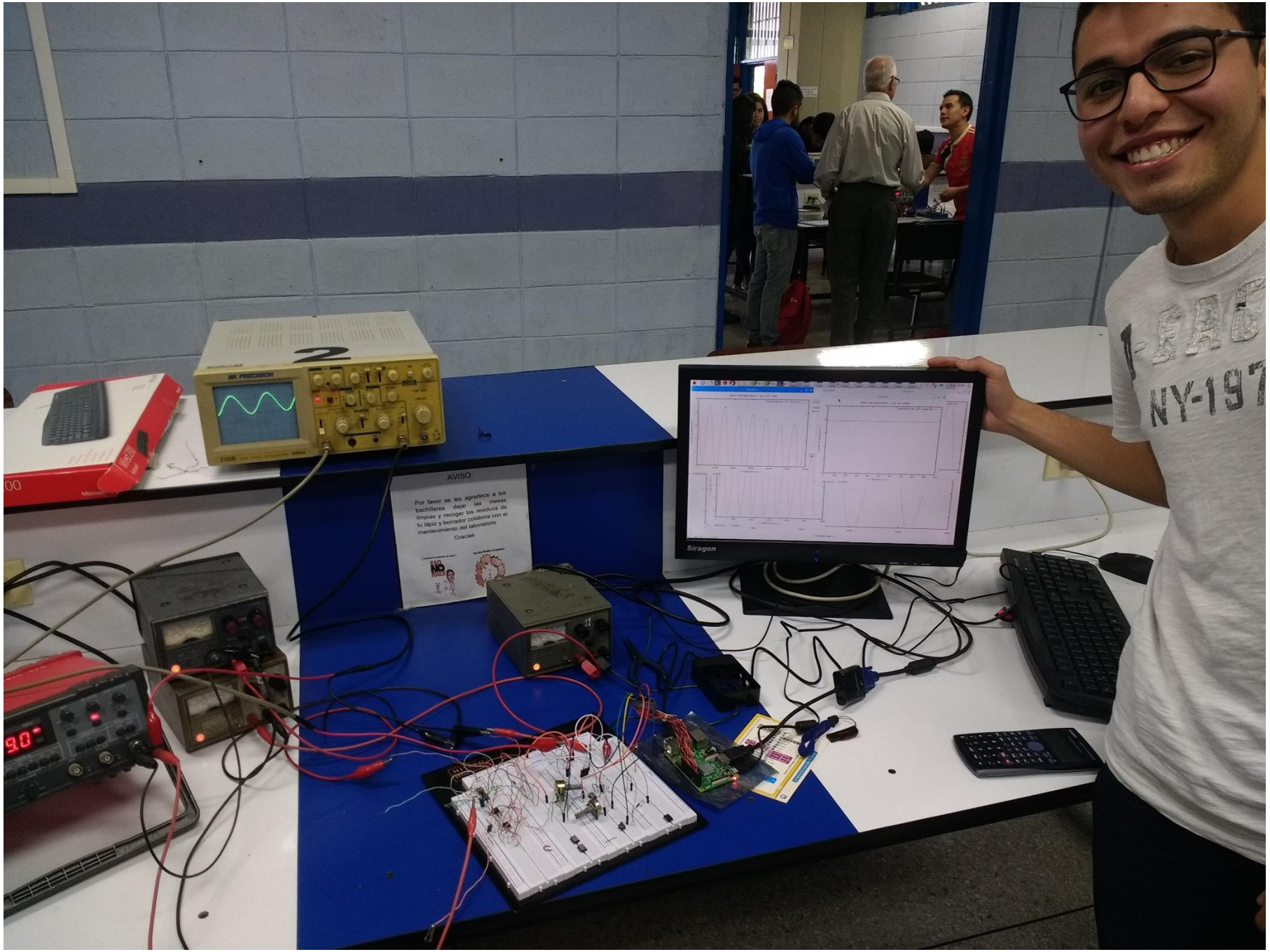
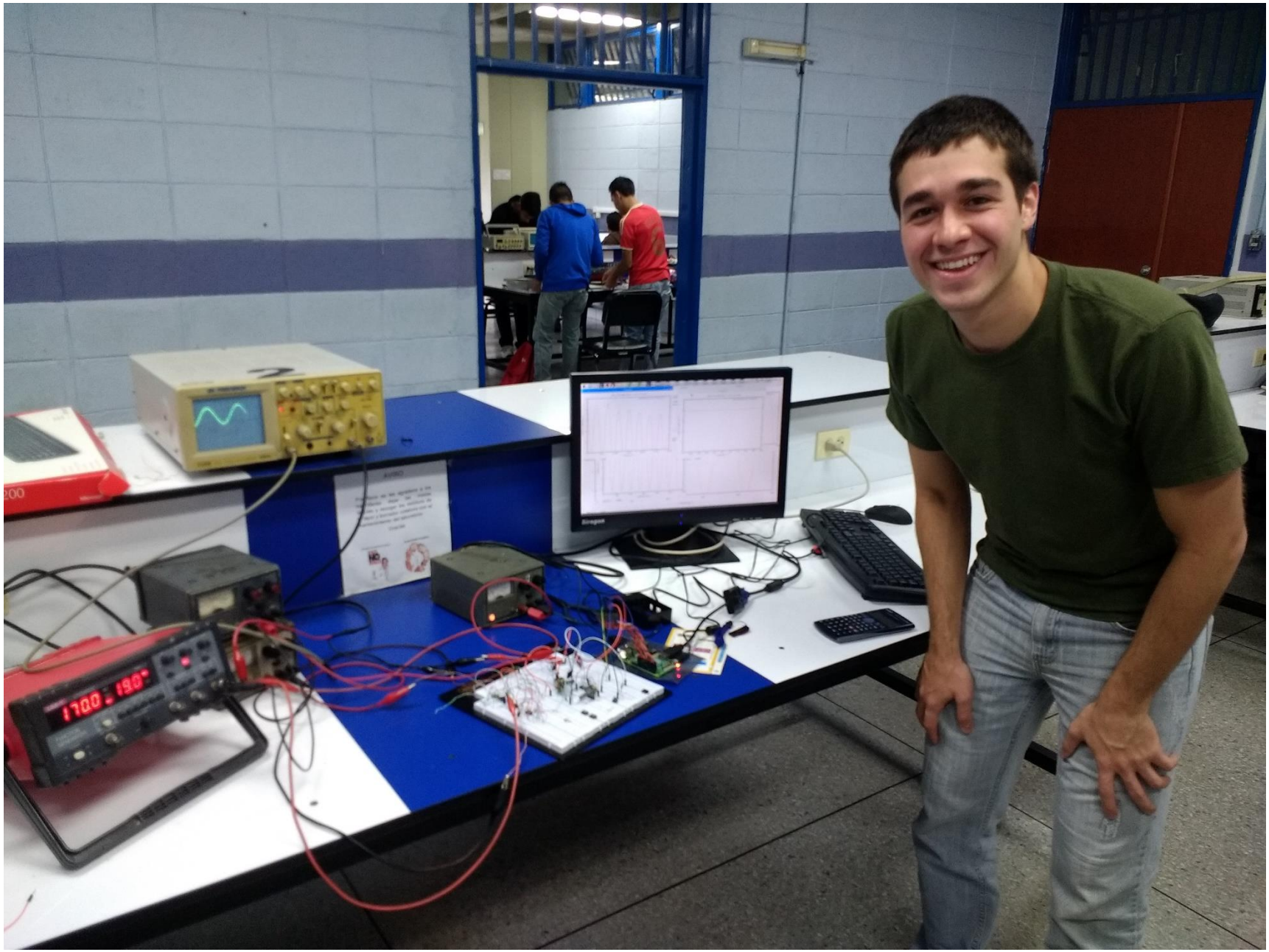

Esquema del circuito completo
10 V
5 V
2,5 V
3,3 V
1 V
-10 V
R1
5k
R2
7k
R3
15k
R4
5k
R5
16k
R6
10k
R7
10k
R8
8k
R9
1.1k
C1
10uF
C2
10nF
RV1
1k
U1
U2
U3
U4
LM741
U5
ADC0831
VREF
VIN(+)
VIN(-)
CS
CLK
DO
GPIO 17
GPIO 18
GPIO 27
GPIO 22

**Foto circuito Completo:**




AVISO
Por favor se les agradece a los bachilleres dejar las mesas limpias y recoger los residuos de tu lápiz y borrador colabora con el mantenimiento del laboratorio
Gracias
Siragon



**Análisis de Resultados:**

El comportamiento de cada etapa, el reductor, amplificador, filtro, adc y graficador fue probada por separado con sencillas pruebas para comprobar su funcionamiento, todas estas pruebas fueron exitosas. Al montar el circuito completo se hicieron cuatro pruebas:

1. Se conectó la entrada del adc directo a una fuerte de voltaje variable, se colocó una frecuencia de muestreo lenta y muchos puntos, logrando muestrear por alrededor de unos 3 segundos. Se comenzó a muestrear y se varió el valor del voltaje, al mostrarse la gráfica en efecto era como los valores esperados así que se pudo comprobar que a frecuencias lentas el sistema tiene la capacidad de muestrear eficientemente los valores de voltaje y graficarlos. A continuación, se anexan fotos:

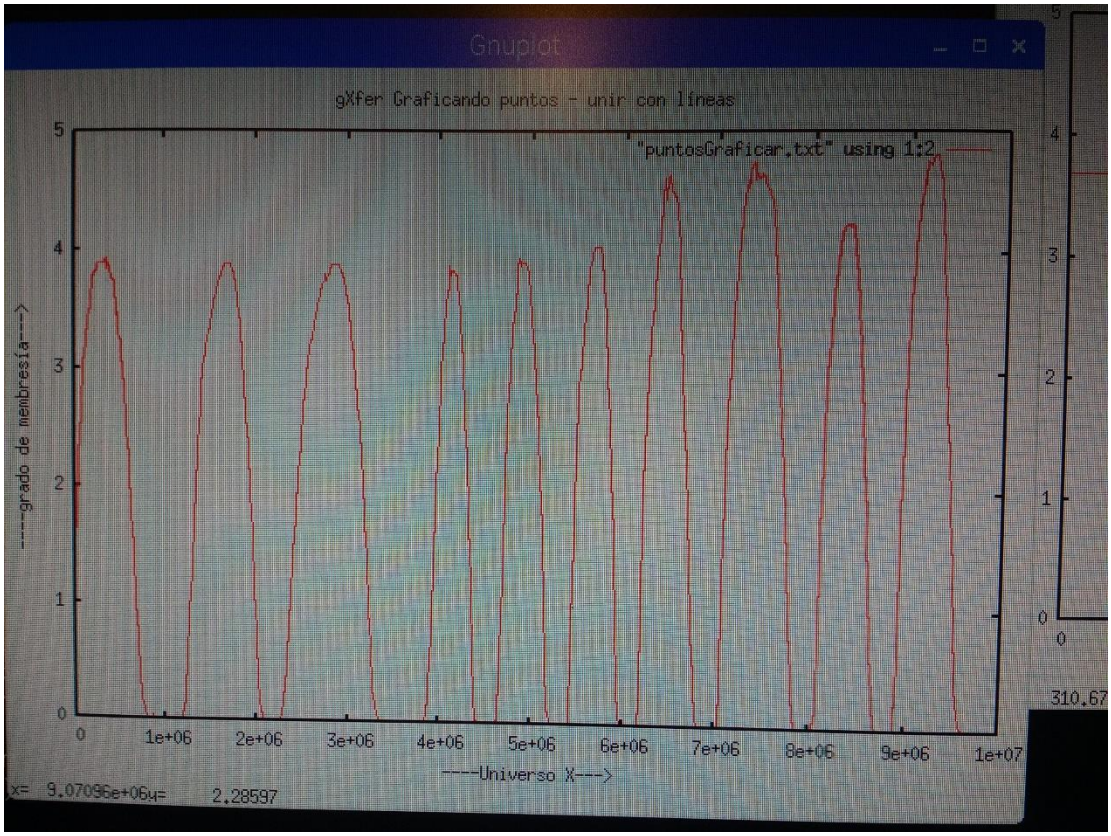



Foto 1: Variando el voltaje durante todo el muestreo.

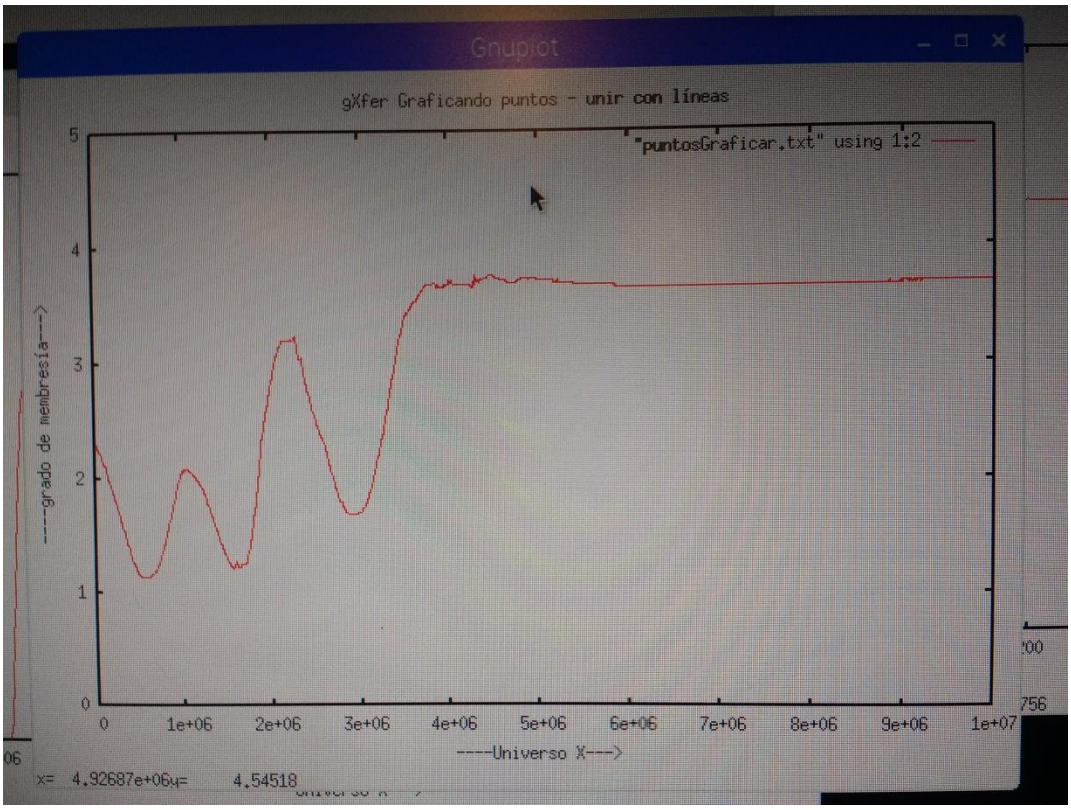



Foto 2: Variando el voltaje una parte del muestreo y dejándolo fijo otra.

2. Se conectó la entrada del adc directo a una fuerte de voltaje variable, se colocó la frecuencia de muestreo máxima y pocos puntos, con el objetivo de muestrear instantáneamente el voltaje. Se comenzó a muestrear y se producía una línea horizontal que cortaba en el valor de voltaje que se tenía, al variar el voltaje la línea de la gráfica subía y bajaba instantáneamente, lo que era el comportamiento esperado así que se pudo comprobar que a frecuencias altas el sistema tiene la capacidad de muestrear eficientemente los valores de voltaje y graficarlos.
3. Se conectó la entrada de amplitud de 20Vpp con una onda sinusoidal de esa amplitud, se puso a funcionar el circuito completo y la gráfica era una onda sinusoidal perfecta. Se midió el periodo de la onda a través de la gráfica de la misma manera que se haría en un osciloscopio analógico y al calcular la frecuencia esta era en efecto la frecuencia de la onda que se le introducía al circuito, con esto se comprobó el funcionamiento del circuito completo
4. Se conectó una onda sinusoidal con una determinada frecuencia y se puso a funcionar el circuito completo, se eligió en el programa la frecuencia que se colocó y esta se observaba perfectamente. Se comenzó a variar la frecuencia del generador de la onda sinusoidal y en la pantalla de la grafica se podía

observar como esta se comprimía y extendía, así que se comprobó el funcionamiento del circuito dependiendo de la frecuencia y su capacidad de mostrar eficientemente una onda de la frecuencia introducida.

**Conclusiones:**

Con el éxito de todas las pruebas descritas en el análisis de los resultados, se concluye que el proyecto planteado se realizó satisfactoriamente, logrando un funcionamiento perfecto. Todos los objetivos fueron cumplidos. Con este proyecto diseñado e implementado exitosamente se tiene un avance para diseñar e implementar un osciloscopio digital a futuro.

**Recomendaciones**

Recomendaciones para futuros proyectos que tengan características o etapas similares a este:

- Trabajar con lenguaje C, y no otro lenguaje como Python, esto debido a que este tipo de aplicaciones de muestreo a muy alta velocidad necesitan de un programa muy rápido y C resulta perfecto para estas aplicaciones.
- Desarrollar con mayor antelación el código del proyecto, ya que en esta etapa se necesita de mucho tiempo para corregir errores.
- Utilizar filtros de tipo bessel ya que este ofrece un desfase constante para todo el espectro de frecuencia así que la señal no se verá afectada.

Recomendaciones para mejoras de este proyecto a un futuro:

- Investigar a fondo los tipos de filtro más adecuados para estas aplicaciones. Investigar de los filtros tipo Besse primeramente.
- Colocar un comparador y un switch electrónico que elija la amplitud de la señal que se está utilizando, asi tener una sola entrada de señales al circuito y que este direccione eficientemente.
- Trabajar con mas escalas de amplitud.
- Colocar un segundo adc que mida con exactitud el valor de voltaje que se utiliza para trasladar las señales al momento de ser medidas por el adc y así evitar colocar un valor arbitrario.
- Mejorar en gran parte la programación.
- Poder introducir los valores de amplitud y frecuencia deseadas y que este las trabaje eficientemente.
- Que el equipo mismo pueda determinar la amplitud de la señal (dependiendo del canal que habilito).

- Que el equipo mismo realice un procesamiento de la data muestreada y adapte los puntos a graficar y los tiempos de muestreo.