Universidad de Costa Rica

Laboratorio 3: Arduino: GPIO,ADC y comunicaciones

IE-0624 Laboratorio de Microcontroladores

Prof. MSc. Marco Villalta

Isaac Rojas Hernández
B76693
Amy Herrera Mora
B53473

6 de mayo de 2023

Índice

1. Introducción

En el presente documento se describe todo el trabajado realizado para el segundo laboratorio del curso Laboratorio de microcontroladores. En él se desea desarrollar un voltímetro de cuatro canales que se base en el microcontrolador Arduino UNO. El sistema como totalidad debe tener la capacidad de recibir cuatro tensiones en el rango [-24,24]V en DC o en AC y mostrar los valores en una pantalla LCD PCD8544. Adicionalmente, se tendrá que comunicar con una PC para poder guardar la data generada en un archivo de texto plano. Para su realización, se siguieron requerimientos, como por ejemplo: medir los valores de los voltajes al mismo tiempo, disenar un circuito que pueda condicionar las tensiones de entrada de [-24,24]V a un rango de voltaje que el ADC del microcontrolador pueda manejar, o sea, [0, 5]V. Se implementaron dos switches tanto para el modo de medición DC o AC, como para el control de la comunicación serial. Además, se buscaba que los datos obtenidos serán enviados al PC mediante el puerto serial con el bloque USART para leer dicho puerto y exportarlos en formato CSV.

Como principales conclusiones se obtuvieron que el ARDUINO UNO es un microcontrolador con una arquitectura mucho mas extendible que otros microcontroladores y con mayor cantidad de pines que permite un desarrollo mas sencillo de programas y un uso de periféricos más amplio. También que el hecho de que un Arduino posea salidas y entradas tanto digitales como analógicas permite una combinación digital-analógica que permite un mayor control y flexibilidad en la realización de proyectos porque se pueden utilizar ambos tipos de salidas para conectar y controlar una amplia variedad de componentes electrónicos y dispositivos. Finalmente, que el puerto serial de Arduino es una herramienta muy útil para la comunicación, el control, la depuración y la programación de proyectos electrónicos. Este puerto permite muchas mas conexiones con mas dispositivos y este hecho convierte a la plataforma Arduino como una herramienta muy flexible y versatil.

2. Nota teórica

2.1. ARDUINO UNO

Para este laboratorio se utiliza el microcontrolador Arduino UNO. Sus principales características son [1]:

- Es un microcontrolador AVR de 8 bits y con arquitectura RISC/Harvard.
- Posee 23 GPIOS, interrupciones, timer/counters de 8 y 16 bits.
- Tiene 8 canales PWM, 6 canales 10-bits ADC y comparador analógico
- Bloque USART Y Maestro/Esclavo SPI.
- TWI (2-wire), DBG, BTLDR.

En las figuras (1) y (2) es posible apreciar tanto el esquemático, con sus pines etiquetados, como el diagrama de bloques del microcontrolador.

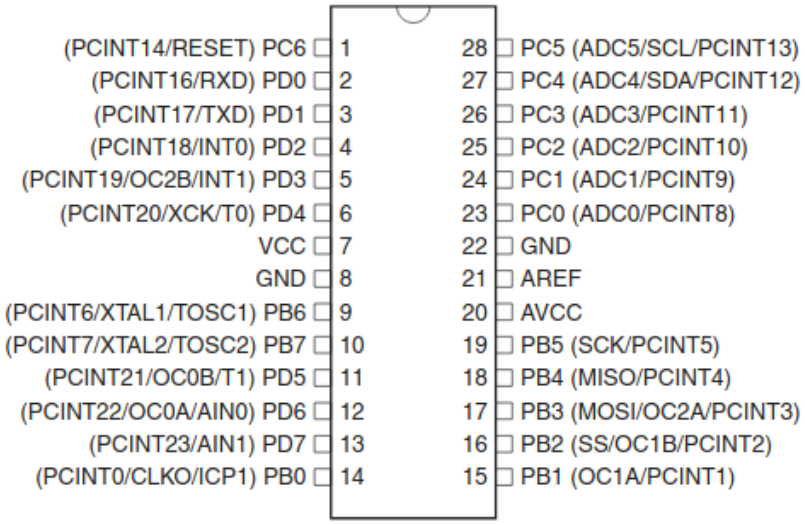


Figura 1: Esquema del Arduino UNO [1]

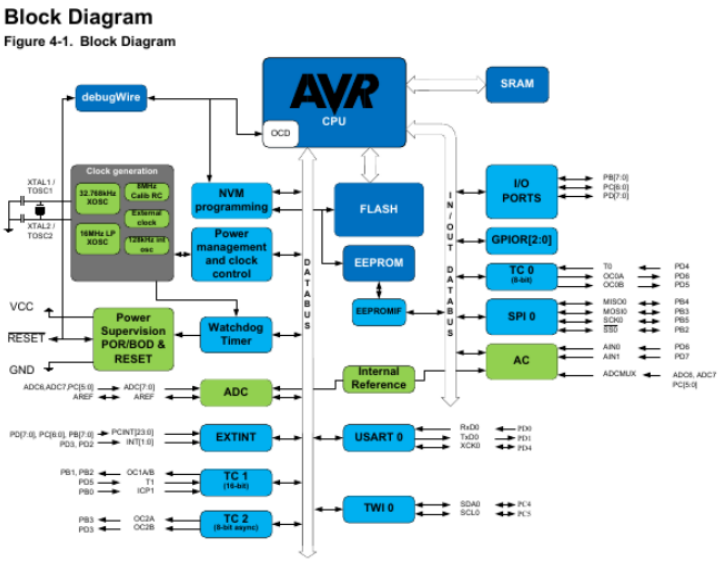


Figura 2: Diagrama de bloques del Arduino UNO[?]

La especificaciones eléctricas vienen dadas por la tabla (1) a continuación. [1].

Especificaciones eléctricas	**Valor**
Temperatura de operación	$-40°C$ a $+85°C$
Máxima tensión de operación	5.5 V
DC por I/O Pin	20 mA
DC en pines Vcc y GND	100 mA

Tabla 1: Especificaciones Eléctricas para el microcontrolador

A continuación se describirá el uso de los periféricos de interés para la elaboración de este laboratorio.

2.1.1. Estructura de un Sketch

La estructura general para realizar un programa que se pueda ejecutar en el Arduino consta de tres fundamentales. En primer lugar, la declaración de las variables globales y funciones propias que se requieran. En segundo lugar, la configuración e inicialización de los periféricos en una función llamada void setup(). Finalmente, se tiene la función void loop() que es la encargada de la ejecucion del programa en si [1]. Como ejemplo se tiene el Arduino IDE en la figura [3]. Como se observa, al abrir un archivo nuevo, por defecto ya viene con la función setup() y el loop() para comenzar el código deseado.

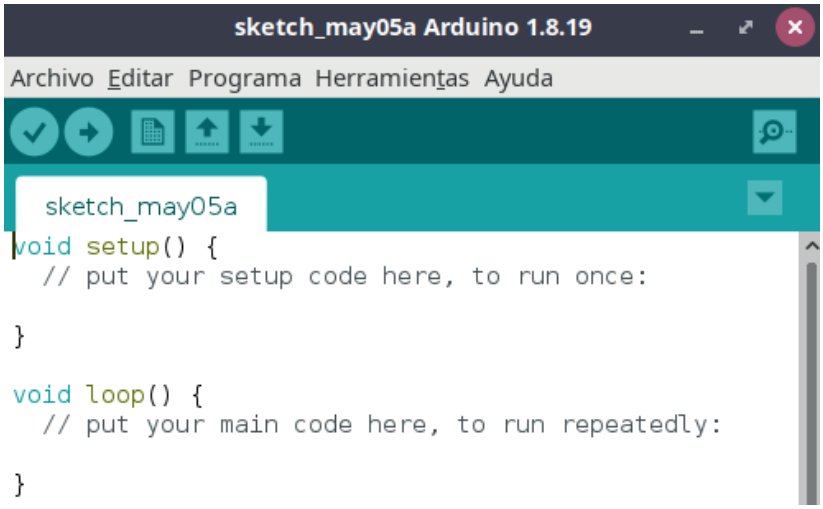


Figura 3: Ejemplo del Arduino IDE.

2.1.2. Entradas y Salidas Digitales

Para poder utilizar estas entradas y salidas digitales se puede por medio de funciones como: pinMode(), la cual se encarga de configurar el pin digital donde el primer parámetro es el pin y el segundo si es INPUT o OUTPUT. digitalWrite(), la cual pone en alto o bajo el pin de interés como primer parámetro y el estado como segundo. digitalRead(), la cual recibe como primer parametro el pin que se quiere leer para retornar ese valor. Las tensiones utilizadas estan entre 0 y 5V. La corriente maxima que pueden soportar es de 20 mA y de 100 mA en total todos los pines de una vez. [1].

Con ayuda del siguiente código se ejemplifica la configuración respectiva:

```
1  void setup(){
2    Serial.begin(115200);
3    //Pines de tensi n
4    pinMode(volt1, INPUT);
5    pinMode(volt2, INPUT);
6    }
7    void loop(){
8    //Leer el modo en el que se est n leyendo las tensiones
9    int mode = digitalRead(switch_AC_DC);
10   int button = digitalRead(switch_CSV);
11   }
```

2.1.3. Entradas y Salidas Analógicas

Además de las salidas y entradas digitales, se tienen las analógicas. Para este caso, las funciones que configuran son: analogWrite(), que se encarga de enviar un valor tipo byte para representar una señal PWM a pin digital que esta configurado como OUTPUT. Luego, se tiene analogRead(), la cual devuelve un valor leído del pin analógico configurado como INPUT. Dicho valor es proporcional a la entrada analógica con respecto a la tensión de referencia en un rango , o sea, 6 canales de 10 bits. Los voltajes que soportan están entre [0,5]V. [1].

Con ayuda del siguiente código se explica la especificación para los pines con este registro:

```
if (mode == LOW){
    display.setCursor(0, 0);
    display.println("Lectura en DC");
    //Tension 1
    v1 = analogRead(volt1) * (5.0 / 1023.0);
    if (v1 < 0 || v1 > 4.9){
      digitalWrite(led_one, HIGH);
      display.print("V1:");
      display.println("ERROR");
    }
```

2.1.4. Comunicaciones: USART

La mayoría de las comunicaciones en un MCU son en serie, y la transmisión se realiza bit a bit. Para poder lograr una comunicación existen algunos estándares y protocolos. Por ejemplo, esta el bloque USART que es mas rapido porque utiliza el clock y puede enviar mayor cantidad de datos que la otra transmisión. [1].

2.1.5. Comunicaciones entre Arduino, PC, y SimulIDE

Las comunicaciones entre un Arduino y la PC se realizan a una velocidad o baud rate. Para este laboratorio se utiliza un baud rate de 9600. Un dato importante que es para crear comunicación entre dos dispositivos, este parámetro y otros, como por ejemplo start bit, la paridad etc, deben ser exactamente iguales. Para este caso, en el Arduino se utilizan los pines digitales 0 (RX) y 1 (TX). [1]. Dentro de las múltiples funciones que se pueden utilizar están: Serial.begin(), la cual es indispensable para abrir la comunicación ya que tiene como primer parámetro la velocidad. Seguidamente, se tiene Serial.print() y Serial.println(), las cuales envían un dato de cualquier tipo y devuelve la cantidad de bytes enviados. La diferencia entre ambas es que una agrega el cambio de linea como usualmente se conoce. Luego, se encuentra Serial.end(), la cual se encarga de cerrar la comunicación. [1]. Con ayuda del siguiente código se explican parte de estas funciones:

```
if (button == HIGH && mode == LOW){
    //Tension 1
    v1 = analogRead(volt1) * (5.0 / 1023.0);
    if (v1 < 0 || v1 > 4.9){
      Serial.println("ERROR");
    }
    else{
      v1 = ((v1 - 0)/(5 - 0))*(25-(-25)) + -25+0.1;
      Serial.println(v1);
    }
    }
```

Adicionalmente, se puede crear comunicación entre la PC y el simulador utilizado SimulIDE. Para esto, basta con utilizar la configuración de Arduino y abrir el Serial Port. Luego, se le debe poner un nombre al puerto y con esto se permite abrir o cerrar para establecer la comunicación. En la figura [4] se ejemplifica este puerto.

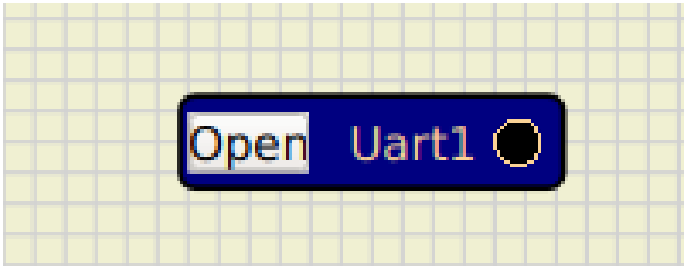


Figura 4: Serial Port.

2.1.6. LCD PCD8544

El PCD8544 es un controlador para pantalla LCD que es monocromática y utilizada comúnmente en pantallas de tipo Nokia 5110. Esta tiene dimensiones 84x48 píxeles y retroiluminación LED. El PCD8544 usa una interfaz serial para poder comunicarse con el microcontrolador. La comunicación se realiza en modo maestro-esclavo, donde el microcontrolador es el maestro y la pantalla el esclavo. Además, el PCD8544 puede manejar hasta 48 caracteres y mostrar gráficos y figuras simples. Tiene la capacidad de realizar desplazamientos y ofrecer una variedad de opciones de configuración para el usuario, como por ejemplo, ajustar el contraste, cambiar el modo de visualización y configurar la dirección de la pantalla. [2]. El esquema para esta pantalla se puede observar en la figura (5).

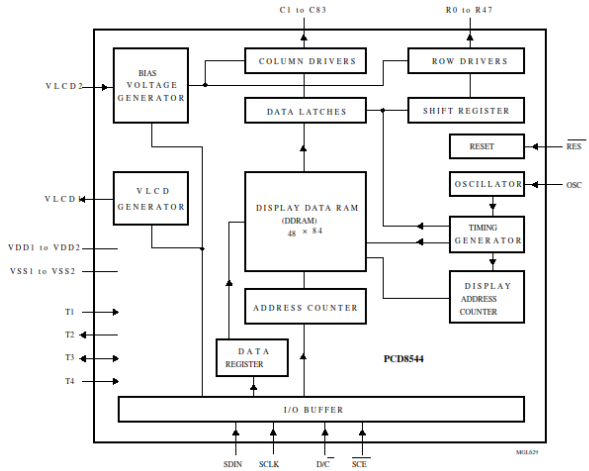


Figura 5: Diagrama para el LCD PCD8544 [2]

2.2. Diseño del circuito

El diseño del circuito fue desarrollado mediante la división en bloques que funcionan en conjunto para lograr el objetivo principal del proyecto. Se puede apreciar en la figura (6) que se dividió el diseño en diferentes partes, lo que permitió simplificar el problema y verificar que cada bloque funciona correctamente.

El primer bloque del diseño es el circuito procesador de tensiones, encargado de acondicionar la entrada para que sea compatible con la entrada analógica del Arduino. Este bloque se utilizó amplificadores operacionales y circuitos de protección para asegurar una medición precisa y estable.

El siguiente bloque es el switch de modo de operación (AC o DC), el cual permite seleccionar el modo de operación deseado. En modo AC, el circuito es capaz de medir señales de corriente alterna, mientras que en modo DC, el circuito mide señales de corriente continua. El tercer bloque es el switch para encender la transmisión serial de los datos capturados.

El cuarto bloque es el circuito de LEDs de alerta, que se encienden en caso de que se sobrepase alguno de los límites de voltaje preestablecido. Por último, el bloque que muestra los datos capturados se compone de una pantalla que muestra en tiempo real los valores medidos de la señal de entrada. Esta pantalla permite visualizar los resultados obtenidos de manera clara y sencilla.

Cada bloque se diseñó cuidadosamente para garantizar el correcto funcionamiento del circuito en su conjunto y asegurar una operación segura y confiable.

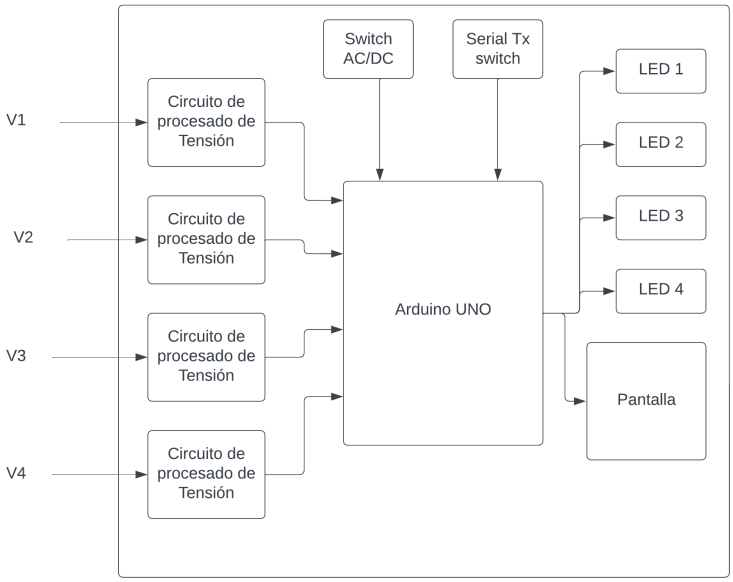


Figura 6: Diagrama de bloques para el voltímetro (Creación propia)

2.2.1. Convertidor de tensión para del Rango Arduino UNO

Debido a las limitaciones presentes en las entradas del arduino uno y las especificaciones del proyecto, se diseñó un circuito que fuese capaz de correr las tensiones de un rango de [-25; 25]V a un rango de [0; 5]V. Para ello se utilizó el esquemático mostrado en la Figura (7)

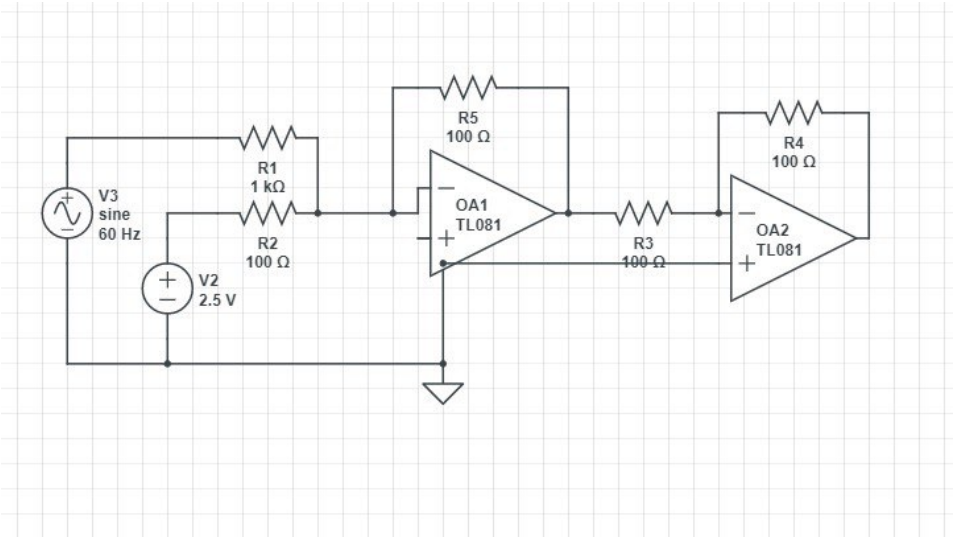


Figura 7: Circuito convertidor de rango de tensión para la entrada al Arduino uno.

El circuito procesa la señal de la entrada independientemente de si es AC o DC. El diseño consta de dos partes principales que al juntarlas dan como resultado el objetivo de diseño requerido. La primera parte es el sumador, el cual toma la tensión de entrada y le añade un valor extra $V_{in2} = 2{,}5V$, para que la señal se desplace. La salida del sumador es la salida del amplificador operacional TL081, de la izquierda del circuito y la ecuación que lo determina está dada por:

$$V_{o1} = -\left(V_{in}\frac{R_5}{R_1} + V_{in2}\frac{R_5}{R_2}\right) \tag{1}$$

De dicha ecuación, se determinó que para poder mover por completo el rango de la tensión de entrada con una relación equivalente a $V_{in} = -25 \rightarrow V_{out} = 0$ y $V_{in} = 25 \rightarrow V_{out} = 5$, es necesario dividir dicha tensión por un factor de diez y sumarle los 2.5 para desplazarla. De dicho análisis se encontró que:

$$R_1 = 10R_5 \tag{2}$$

$$R_5 = R_2 \tag{3}$$

Así se culminó de forma correcta el objetivo de la primera parte del circuito, además para ello se utilizó los siguientes resistores: $R_2 = R_5 = 100\Omega$ y $R_1 = 1000\Omega$. La salida que se obtuvo de esta configuración es inversa o negativa, ya que es un sumador inversor, así que se diseñó la segunda parte del circuito para que cumpla con este objetivo.

La configuración del segundo amplificador operacional es inversora, para que se corrija al valor real de la primera configuración. La topología sencilla da como resultado la siguiente ecuación:

$$V_{out} = -V_{o1}\frac{R_4}{R_3} \tag{4}$$

Como la señal ya se convirtió a un rango deseado para el Arduino por medio de sumador, el diseño de las resistencias debe cumplir que:

$$R_4 = R_3 \tag{5}$$

Por lo que por temas de diseño, se escogió un valor de 100Ω, para ambas. De esta forma se finalizó el diseño que corrige el rango de tensión especificado para este laboratorio a un rango que sea legible para el microcontrolador.

2.2.2. Switches

Para el desarrollo del diseño de este laboratorio 3 se requirió el uso de dos switches. En primer lugar, se encuentra el switch superior de la figura 10 que se conecta al pin 7. Este se encarga de la configuración para el modo de medición. Mientras este abierto, se obtendrán tensiones en DC, mientras que cuando se cierre estas serán en AC. Por otro lado, el switch superior conectado al pin 12 se encarga de controlar la transmisión serial para que los datos obtenidos se envíen hacia la computadora utilizando el puerto serial con el bloque USART. En la figura 10 se observan estos.

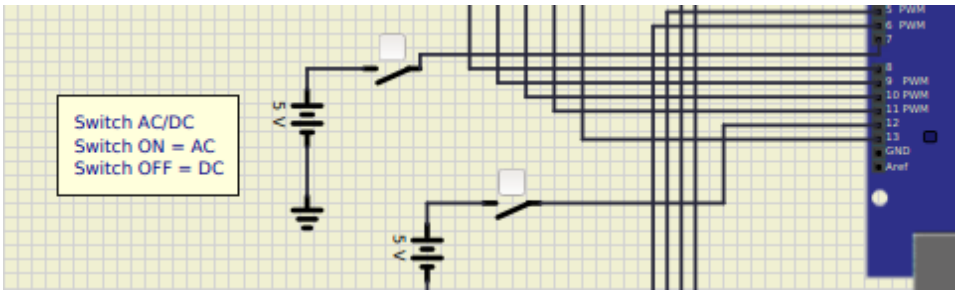


Figura 8: Switches utilizados en el diseño del circuito para el voltímetro.

2.2.3. LEDs de Alerta

Debido a que se trabajó con un rango específico de tensiones, se creyó necesario el diseño de un sistema que alerte a los usuarios cuando se encuentren fuera de dicho rango. Se escogió el LED rojo para la creación de las alertas y se diseñó un circuito con 4 LEDs uno por cada una de las entradas de tensión.

La hoja del fabricante, ubicada en el anexo 3, determina que para una operación normal la tensión de operación $V_f = 2{,}0V$ para una corriente máxima de $20mA$, dado que la salida digital del Arduino tiene una magnitud $V_{cc} = 5V$, entonces la resistencia de protección necesaria está determinada por:

$$R_{protection} = \frac{V_{cc} - V_f}{I_{cc}} = \frac{5-2}{20mA} = 150\Omega \tag{6}$$

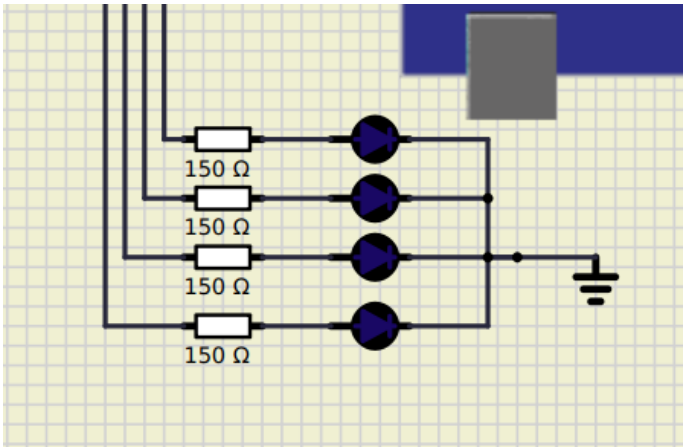


Figura 9: LEDs de alerta para el voltímetro.

2.2.4. Arduino UNO

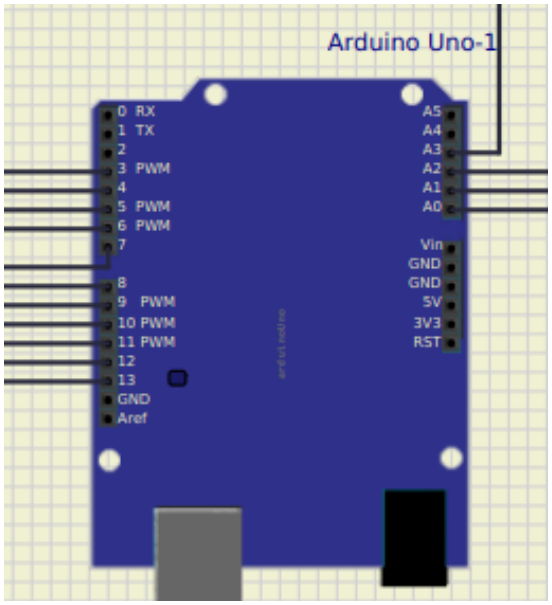


Figura 10: Esquemático completo para el voltímetro.

2.2.5. Circuito Completo

EL diseño final del circuito se puede apreciar en la figura (), donde además de lo diseñado anteriormente, se añadió una batería a la salida de cada una de las tensiones procesadas de $0{,}1V$, esto se tomó como una decisión de diseño para poder diferenciar cuando el usuario sobrepasaba los límites puestos para cada uno de los circuitos, la tensión perdida, luego se añadió en el código para tener el valor correcto.

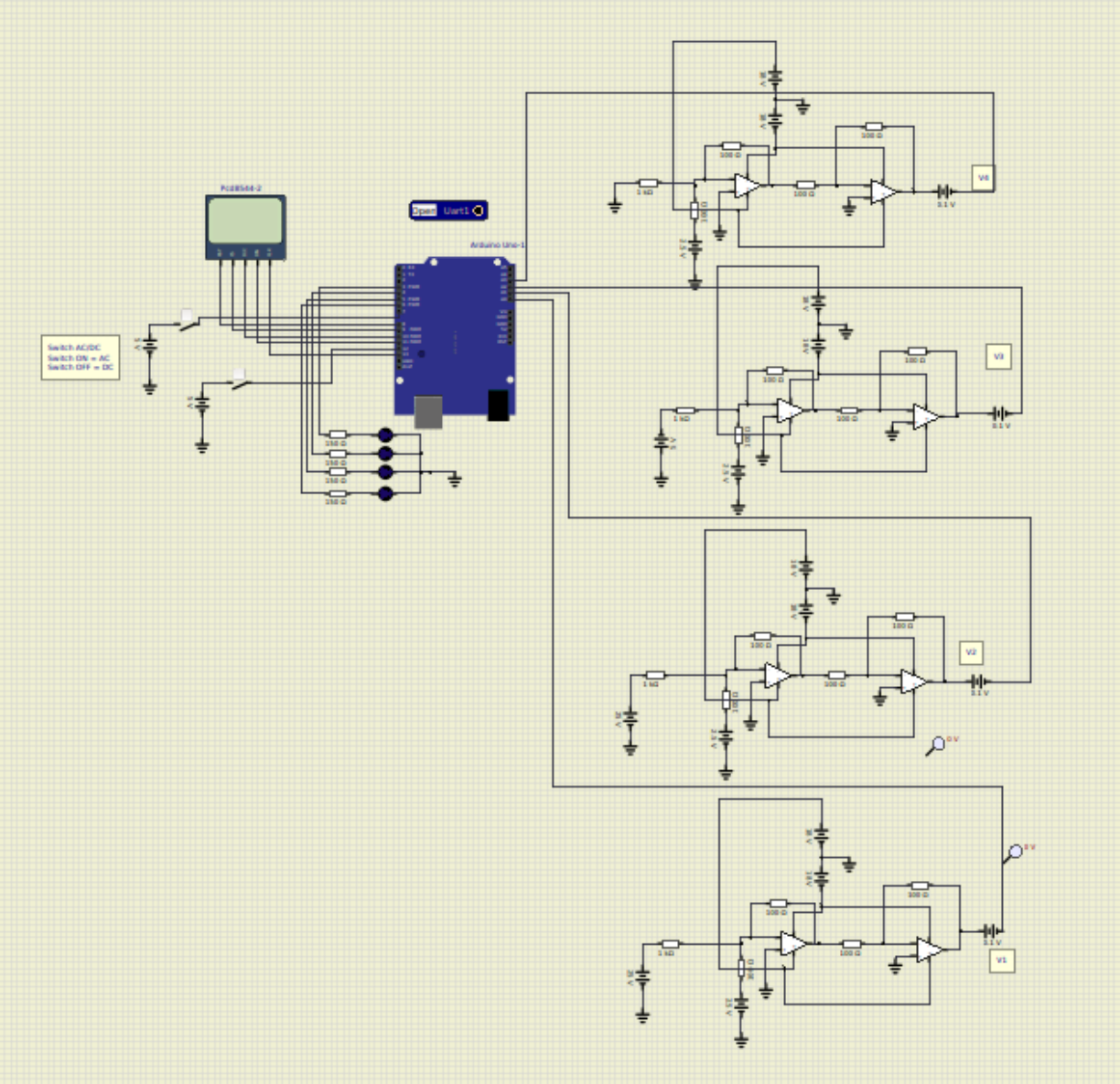


Figura 11: Esquemático completo del voltímetro.

Los costos de diseñar un circuito como el anterior físicamente están dados en la siguiente tabla:

Componente	**Cantidad**	**Precio (colones)**
Resistencias de 100Ω	20	500
Resistencias de 150Ω	4	280
LEDs de colores	4	240
LCD PCD8544	1	22 000
Switches	2	2200
Microcontrolador Arduino UNO	1	15 950
Total		41 170

Tabla 2: Tabla de costos de los componentes empleados para el circuito.

3. Desarrollo/Análisis de resultados

Para poder desarrollar el programa que contenga todas las funcionalidades descritas en el enunciado del laboratorio, fue necesario seguir el diagrama de flujo mostrado en la figura (12).

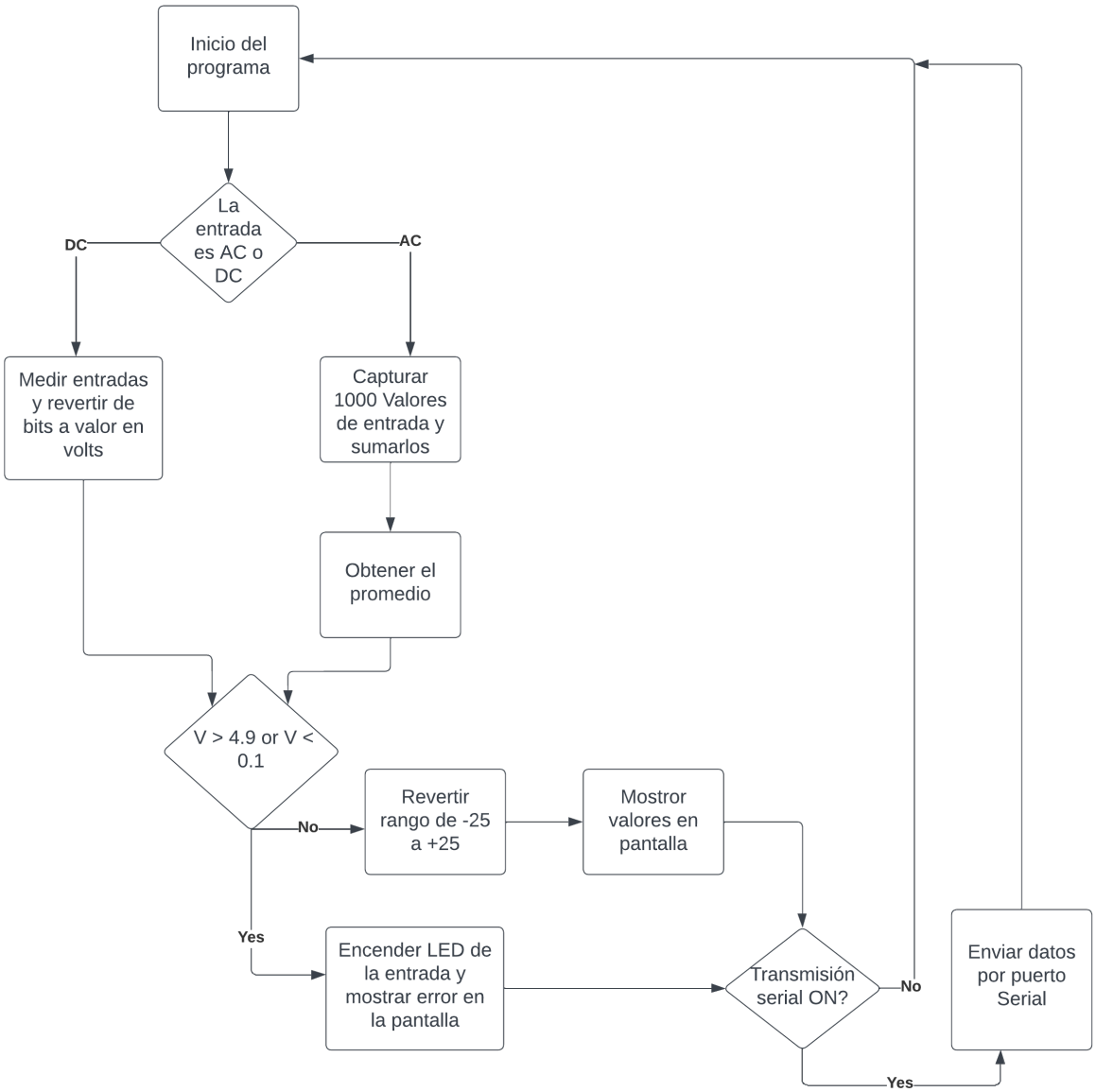


Figura 12: Diagrama de bloques del programa del voltímetro con Arduino UNO.

El programa como estándar en Arduino debe configurar inicialmente todo los puertos que vaya a utilizar. La asignación de cada puerto es bastante intuitiva y permitió que el manejo del código se realizara de forma sencilla y llevadera. Un ejemplo de definición de pines a constantes globales es el siguiente:

```
1 //Pines anal gicos de entradas de tensi n
2 int volt1 = A0;
3 int volt2 = A1;
```

```
int volt3 = A2;
int volt4 = A3;
```

Una vez se definieron todas las constantes globales, se debe configurar su modo de operación, además de instanciar todo lo que sea necesario para el funcionamiento adecuado del programa. La función setup utilizada se define a continuación:

```
void setup(){
  Serial.begin(115200);
  //Pines de tension
  pinMode(volt1, INPUT);
  pinMode(volt2, INPUT);
  pinMode(volt3, INPUT);
  pinMode(volt4, INPUT);
  // Pines para los switches
  pinMode(switch_AC_DC, INPUT);
  pinMode(switch_CSV, INPUT);
  // Pines de los leds
  pinMode(led_one, OUTPUT);
  pinMode(led_two, OUTPUT);
  pinMode(led_three, OUTPUT);
  pinMode(led_four, OUTPUT);
  // Inicializar la pantalla
  display.begin();
  display.setContrast(50); // Ajustar el contraste de la pantalla (0-127)

  // Limpiar la pantalla
  display.clearDisplay();

}
```

El programa que corre de forma cíclica cuenta con la estructura principal mostrada en el diagrama de flujo de la figura (12). Inicialmente se lee la entrada del switch de modo de operación que, en caso de estar en alto, se encuentra operando para tensiones en AC, mientras que si se lee bajo, significa que está operando con tensiones DC. La operación DC es igual a la AC, pero debido a que los valores que entran no están oscilando en una forma de onda, entonces únicamente debe procesarse el primer dato que obtiene la entrada del Arduino. Para esto se debe pasar ese valor recibido a un valor de tensión, ya que el Arduino devuelve un valor en bits y no un número como tal. En este punto es donde se verifica que el valor obtenido no esté por encima de 4.90V o menor a 0.1V. En caso de que alguno de estos dos se supere, el arduino enciende el LED de alerta y manda una señal de error por medio de la pantalla.

Si no se sobrepase ninguno de los valores del rango preestablecido hay que transformar el valor obtenido, ya que el circuito de la entrada lo modificó, para lograr esto se implementó la siguiente fórmula:

$$V_{out} = \frac{V_{in} - V_{minin}}{V_{maxin} - V_{minin}} \cdot (V_{maxout} - V_{minout}) + V_{minout} \tag{7}$$

Esta formula devuelve el valor de entrada leído de 0 a 5V a un valor en el rango original de -25 a 25V, donde V_{out} es la tensión original recibida, V_{in} es la tensión transformada recibida por el Arduino, V_{minin} y V_{maxin} son las tensiones de 0 a 5 respectivamente y por último V_{minout} y V_{maxout} son -25 y 25V respectivamente. El código que implementa dicha funcionalidad es el siguiente:

```
v1 = analogRead(volt1) * (5.0 / 1023.0);
    if (v1 < 0 || v1 > 4.9){
      digitalWrite(led_one, HIGH);
```

```
      display.print("V1:");
      display.println("ERROR");
    }
    else{
      digitalWrite(led_one, LOW);
      v1 = ((v1 - 0)/(5 - 0))*(25-(-25)) + -25+0.1;
      display.print("V1:");
      display.println(v1);
    }
```

Como se pudo ver, además de los cálculos descritos anteriormente en este proceso también se va actualizando la pantalla para siempre mostrar los valores más recientes en ella.

En caso de que la lectura sea en AC, el cambio que presenta el procesamiento de los datos es que no se puede tomar únicamente el primer valor leído, sino que se deben tomar una cantidad significativa de datos, en este caso se tomaron 1000 valores de muestra, y promediarlos. Luego de realizar este proceso, se multiplica por $\sqrt{2}$, para obtener el valor Vrms y este es el que se muestra por medio de las pantallas.

```
for (int i = 0; i < 1000; i++){
      v1 =+ analogRead(volt1) * (5.0 / 1023.0);
    }
    v1 = v1/1000;
    if (v1 < 0 || v1 > 4.9){
      digitalWrite(led_one, HIGH);
      display.print("V1:");
      display.println("ERROR");
    }
    else{
      digitalWrite(led_one, LOW);
      v1 = ((v1 - 0)/(5 - 0))*(25-(-25)) + -25+0.1;
      v1 = v1*sqrt(2);
      display.print("V1:");
      display.println(v1);
    }
```

Por último el algoritmo revisa si se encuentra activa la comunicación serial, para poder enviar los datos por medio de dicho bus y procesarlos por medio del script de python y convertirlo a un archivo CSV.

Para la parte de la comunicación con el Serial Port, se revisa que el switch de envío este activado y asimismo el modo de medición para que se exporten ya sea en DC o AC. Con este control no importa si el puerto esta abierto o cerrado, los datos simplemente no se van a pasar. Luego, para continuar con esta parte, se escribieron dos scripts adicionales para correr en dos ventanas separadas de la terminal al mismo tiempo. El primer archivo fue un virtualports.sh, en el cual se configuran puertos virtuales para obtener los datos desde el Arduino y poder leerlos con Python. En la figura (13) se observa este archivo. Cabe destacar que el puerto en la simulación corresponde a /tmp/ttyS0, mientras que el puerto que se lee desde Python en terminal es /tmp/ttyS1.

```
#!/bin/sh
socat PTY,link=/tmp/ttyS0,raw,echo=0 PTY,link=/tmp/ttyS1,raw,echo=0
```

Figura 13: Archivo para los puertos virtuales.

Por otra parte, el archivo en Python llamado datasaver.py contiene los mismos parámetros que el código en Arduino para poder comunicarse. El siguiente código es un fragmento de este script.

```
#!/usr/bin/python3
import serial
import time
import csv

fileName = "output.csv"
encabezado = "Tension1; Tension2; Tension3; Tension4\n"

ser = serial.Serial(
    port='/tmp/ttyS1',\
    baudrate=9600,\
    parity=serial.PARITY_NONE,\
    stopbits=serial.STOPBITS_ONE,\
    bytesize=serial.EIGHTBITS,\
    timeout=20\
    )
```

Finalmente, los datos se exportan en formato CSV después de obtener 100 muestras gracias a Socat. Este valor se puede ajustar a las necesidades que se requieran. El código se muestra a continuación:

```
while True:
    linea_serie = ser.readline().decode().strip()
    valores.extend(linea_serie.split(','))
    print(valores)

    if len(valores) >= 4:
        while len(valores) >= 4:
            linea = ";".join([str(valor) for valor in valores[:4]])
            file.write(linea)
            file.write("\n")
            valores = valores[4:]
            contador += 4

    contador_datos += 1  # Incrementar el contador de datos recibidos
    if contador_datos >= num_datos:
        break
```

Como parte del análisis de los resultados, se realizaron varias pruebas tanto en DC como en AC y se mostraran dos de ellas. En primer lugar, se muestra en la figura (14) que efectivamente los datos generados aparecen en la pantalla de la simulación. Se observa que como los valores están dentro del rango deseado, ninguno de los LEDs esta encendido.

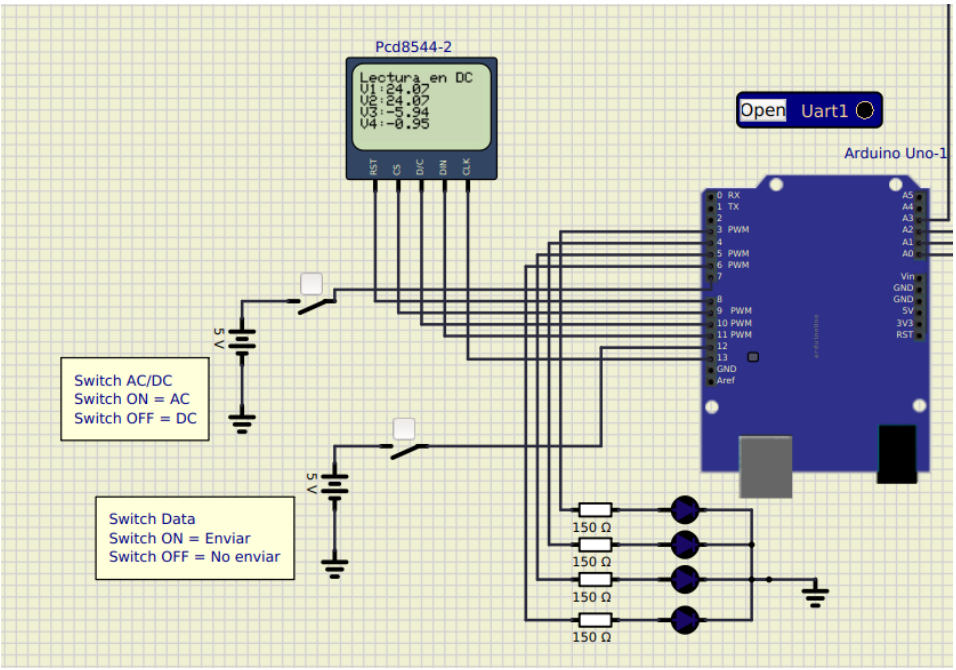


Figura 14: Prueba en DC.

Por otro lado, con esta misma prueba se muestra en la figura (15) que se conecta el switch pero se abre de nuevo, entonces no aparecen mas valores de tensiones en el serial monitor del simulador. Con esto se evidencian los objetivos alcanzados para los requerimientos especificados en este laboratorio.

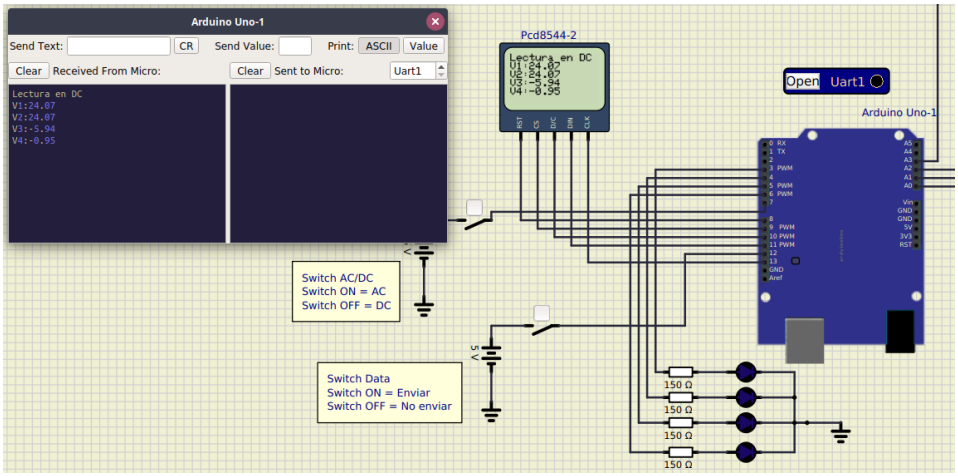


Figura 15: Muestra del Serial Port.

Para las pruebas en AC, se conectaron las fuentes AC en vez de las fuentes en DC como se puede apreciar en la figura (16) y además el switch AC debe estár activado.

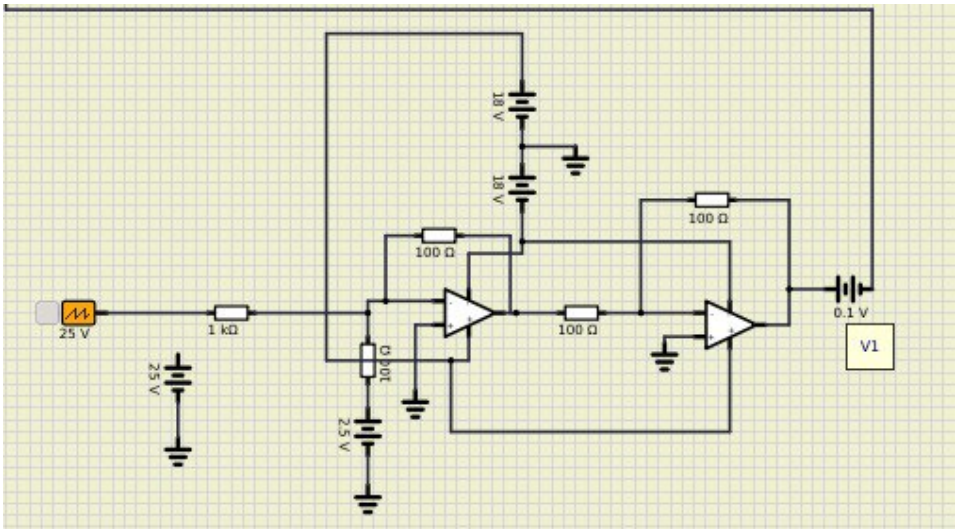


Figura 16: Configuración AC

Para una configuración con entradas de 25V, 5V, 86V, 15V, se tienen los siguientes resultados:

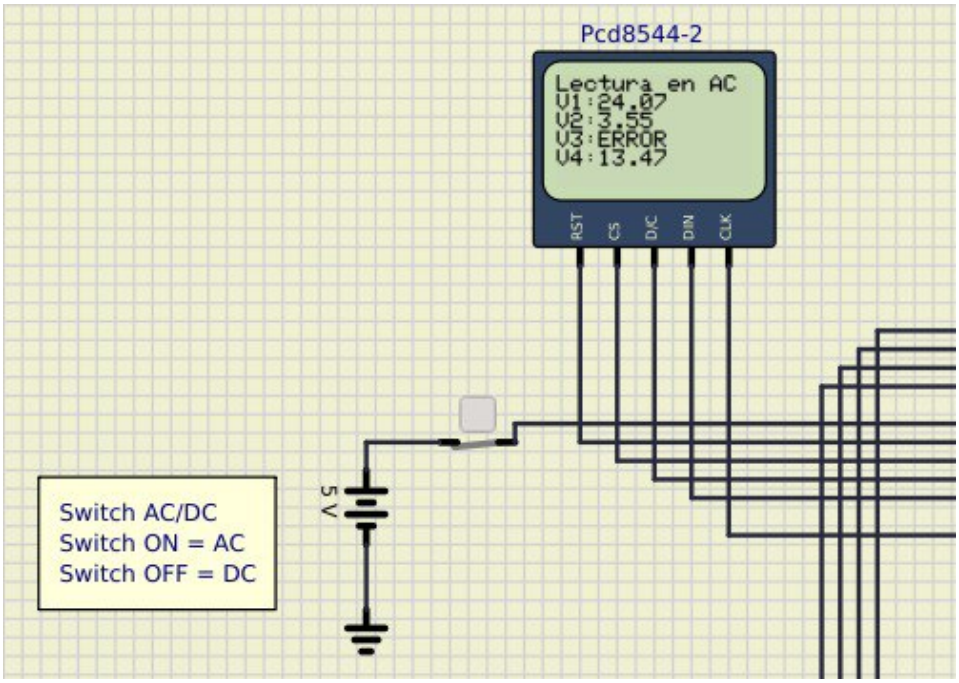


Figura 17: Resultados de las pruebas AC AC

De dichos resultados se puede apreciar que a pesar de que tienen un porcentaje de error mayor a las lecturas en DC, esto se debe por los constantes cambios en la entrada debido a la onda, además de el método utilizado para capturarlas. Es necesario mencionar, que durante el desarrollo del presente laboratio el simulador no generaba los mejores resultados en la configuración AC, apesar de que el circuito se diseñó pensando específicamente en la necesidad AC que posee mayor complejidad. La configuración exacta del amplificador operacional con respecto al modelo que se esocogió durante el diseño, no fue posible, así que se tomaron diferentes medidas para aproximar su comportamiento. Incluso un modelo ideal debió haber cumplido con las especificaciones de diseño, pero de la misma

forma, no se logró implementar de forma exacta. A pesar de ello se toman los resultados como resultados aceptables por la cercania con los medidos, aún con un algoritmo que simplifica la medición de los datos.

4. Conclusiones y recomendaciones

- Como primera conclusión se tiene que el ARDUINO UNO es un microcontrolador con una arquitectura mucho mas extendible que otros microcontroladores y con mayor cantidad de pines que permite un desarrollo mas sencillo de programas y un uso de periféricos más amplio.
- Por otra parte, el hecho de que un Arduino posea salidas y entradas tanto digitales como analógicas permite una combinación digital-analógica que permite un mayor control y flexibilidad en la realización de proyectos porque se pueden utilizar ambos tipos de salidas para conectar y controlar una amplia variedad de componentes electrónicos y dispositivos.
- Además, el puerto serial de Arduino es una herramienta muy útil para la comunicación, el control, la depuración y la programación de proyectos electrónicos. Este puerto permite muchas mas conexiones con mas dispositivos y este hecho convierte a la plataforma Arduino como una herramienta muy flexible y versatil. A pesar que se tenga un algoritmo con un funcionamiento adecuado, es necesario revisar que una implementación similar se desarrolle de forma adecuada, ya que al trabajar con hardware hay más que una ejecución lineal de código.
- Adicionalmente, el diseño del circuito como primera parte es la base de las implementaciones con microcontroladores, por ende, diseñar inicialmente el esquemático ayuda a que el desarrollo del programa sea más fácil.
- Como parte de las recomendaciones es importante mencionar que el uso de la hoja de datos del fabricante del microcontrolador es esencial para el desarrollo de cualquier programa cuando se desea realizar un circuito con uno de estos dispositivos.
- También, la investigación de distintas aplicaciones para el microcontroladores puede dar un mejor entendimiento a la hora de programar el firmware del mismo.
- Es necesario buscar fuentes externas a la hoja de datos del microcontrolador, como por ejemplo, la comunicación Arduino-PC-SimulIDE, ya que la documentación puede no ser tan explícita con respecto al funcionamiento o el manejo de ellas.
- Finalmente, la compilación y el testing por etapas en el desarrollo podría ayudar a detectar problemas en el desarrollo temprano de este.

5. GIT

El código fuente del presente laboratorio 2 se encuentra disponible en el siguiente enlace: https://github.com/mimiherrera/IE0624-Lab-de-Microcontroladores/tree/Lab3/TercerLaboratorio

Bibliografía

1. Atmel Corporation, *8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash*, 2016. [Online]. Disponible en: https://html.alldatasheet.com/html-pdf/392243/ATMEL/ATMEGA328/152/1/ATMEGA328.html. [Accesado: Abril 25, 2023].

2. Texas Instruments *PCD8544 48 x 84 pixels matrix LCD controller/driver*, 1999. [Online]. Disponible en: https://www.alldatasheet.com/datasheet-pdf/pdf/18170/PHILIPS/PCD8544.html. [Accesado en: Abril 27, 2023].

3. J. Christoffersen, *Switch Bounce: How to Deal with It, All About Circuits*, Sep. 03, 2015. [Online]. Disponible en: https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/. [Accesado el: Mar. 27, 2023].

6. Apéndices

Apéndice 1: Hoja de datos del microcontrolador

Features

- **ATMega328P** Processor
  - **Memory**
    - AVR CPU at up to 16 MHz
    - 32KB Flash
    - 2KB SRAM
    - 1KB EEPROM
  - **Security**
    - Power On Reset (POR)
    - Brown Out Detection (BOD)
  - **Peripherals**
    - 2x 8-bit Timer/Counter with a dedicated period register and compare channels
    - 1x 16-bit Timer/Counter with a dedicated period register, input capture and compare channels
    - 1x USART with fractional baud rate generator and start-of-frame detection
    - 1x controller/peripheral Serial Peripheral Interface (SPI)
    - 1x Dual mode controller/peripheral I2C
    - 1x Analog Comparator (AC) with a scalable reference input
    - Watchdog Timer with separate on-chip oscillator
    - Six PWM channels
    - Interrupt and wake-up on pin change
  - **ATMega16U2 Processor**
    - 8-bit AVR® RISC-based microcontroller
  - **Memory**
    - 16 KB ISP Flash
    - 512B EEPROM
    - 512B SRAM
    - debugWIRE interface for on-chip debugging and programming
  - **Power**
    - 2.7-5.5 volts

Apéndice 2: Hoja de datos del Amplificador Operacional TL081

Table of Contents

4 Revision History

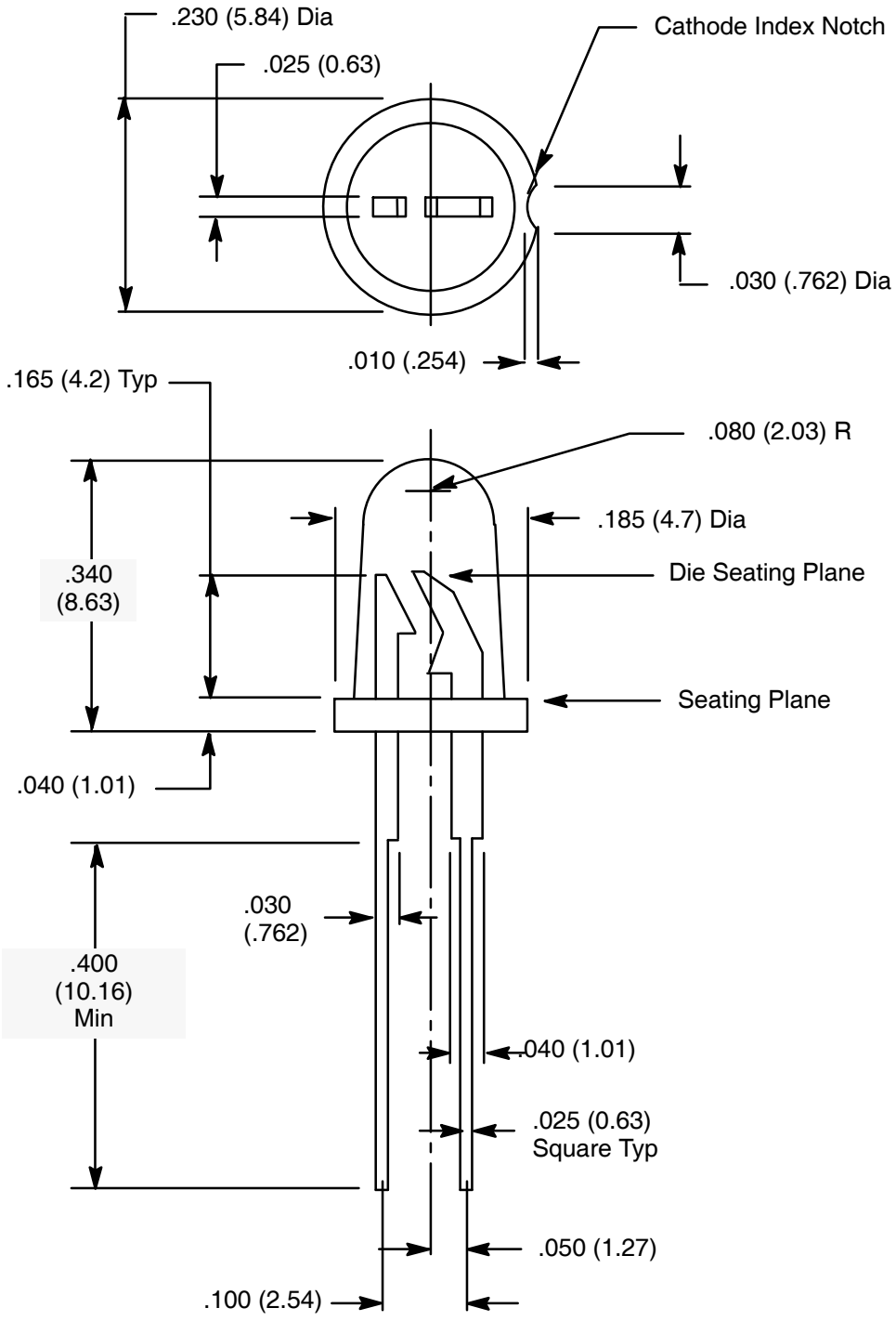
NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision L (July 2021) to Revision M (December 2021)	Page
• Corrected DCK pinout diagram and table in *Pin Configurations and Functions* section	4

Changes from Revision K (June 2021) to Revision L (July 2021)	Page
• Deleted preview note from TL081H SOIC (8), SOT-23 (5), and SC70 (5) packages throughout the data sheet	1

Changes from Revision J (November 2020) to Revision K (June 2021)	Page
• Deleted VSSOP (8) package references throughout data sheet	1
• Deleted preview note from TL082H SOIC (8), SOT-23 (8), and TSSOP (8) packages throughout the data sheet	1
• Added DBV, DCK, and D packages to TL081H in *Pin Configuration and Functions* section	4
• Added ESD information for TL082H	10
• Added D, DCK, and DBV package thermal information in Thermal Information for Single Channel: TL081H section	11
• Added D, DDF, and PW package thermal information in Thermal Information for Dual Channel: TL082H section	11
• Added I_B and I_{OS} specification for single channel DCK and DBV package	13
• Added I_Q spec for TL081H and TL082H	13
• Removed *Related Links* section from *Device and Documentation Support* section	36

Apéndice 3: Hoja de datos LED Rojo


.230 (5.84) Dia
.025 (0.63)
Cathode Index Notch
.030 (.762) Dia
.010 (.254)
.165 (4.2) Typ
.080 (2.03) R
.185 (4.7) Dia
.340
(8.63)
Die Seating Plane
Seating Plane
.040 (1.01)
.030
(.762)
.400
(10.16)
Min
.040 (1.01)
.025 (0.63)
Square Typ
.050 (1.27)
.100 (2.54)