Laboratorio #3

Gabriel Fernando Araya Mora
B80525
gabriel.arayamora@ucr.ac.cr

Arduino: GPIO,ADC y comunicaciones

2021-10-05

Laboratorio de Microcontroladores
(IE-0624)

1. Introducción:

Para esta práctica de laboratorio se implementa un voltímetro digital por medio de un microcontrolador ATmega328 o propiamente con un Arduino Uno. Para hacer esto posible se diseña un divisor de tensión capaz de acondicionar las tensiones en el rango que se quieren medir a un rango que el Arduino entienda y sea seguro para el mismo, para lograr esto se hizo uso de componentes pasivos. Por medio de software se ajustan los valores con el fin de que a la salida se muestren valores con los menores porcentajes de error posibles. El Voltímetro es capaz de medir tensiones directas y alternas mediante el mismo puerto de entrada, ya que el divisor diseñado es lo suficientemente versátil para transformar ambas entradas. Además se tiene por medio del puerto serial que el Arduino se puede comunicar con una computadora para generar un archivo CSV usando Python, además del CSV para mostrar los datos se tiene una pantalla LCD PCD8544. Por último con el fin de eliminar el factor del "bouncing" del botón de entrada se utiliza un filtro pasivo hecho con una resistencia y un capacitor.

2. Nota Teórica:

2.1. ATmega328

El Atmega328 es un microcontrolador el cual cuenta con un procesador de tipo RISC (Reduced instruction set) avanzado de 8 bits. Este microcontrolador es lo suficientemente potente como para poder realizar complejas instrucciones en un único ciclo de reloj. Esto permite al programador, pensar en optimizar el consumo de potencia sin preocupaciones de perder rendimiento. Este MCU cuenta con la mayor cantidad de memoria de todos los que se han usado hasta el momento, por lo que es de esperar que para este proyecto no se tengan que cambiar implementaciones para reducir consumo de memoria.
En cuanto a los periféricos, este microcontrolador cuenta con 23 pines en entrada/salida completamente programables (GPIOs), los cuales están divididos en pines analógicos y digitales. Cuenta con tres contadores, 2 de 16 bits y 1 de 8 bit.

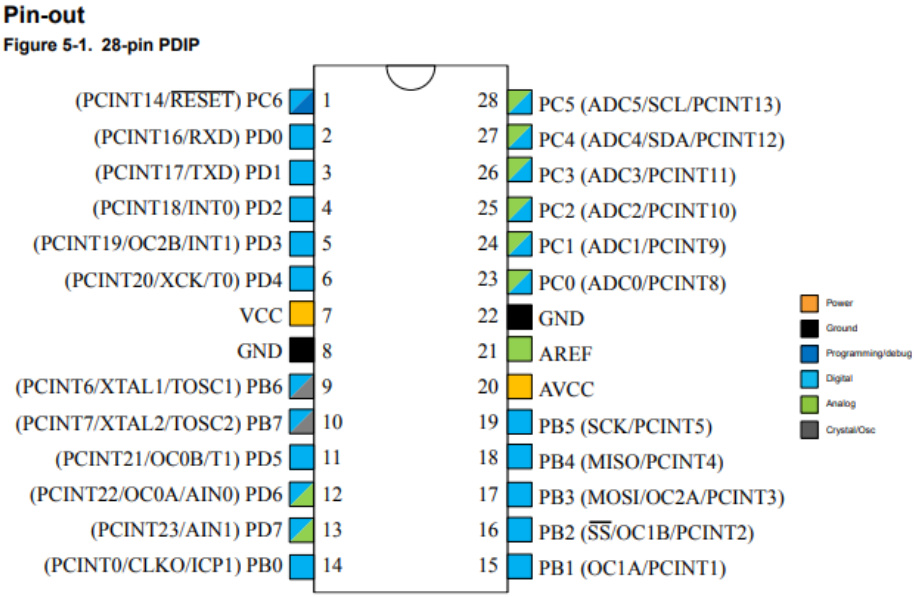


Figura 1: Esquema general del ATmega328. [Atmel(2016)]

2.1.1. Memoria y registros en el ATmega328

Este microcontrolador cuenta con una memoria flash de $32K$ Bytes, una EEPROM de $1k$ Bytes, y por último una SRAM de $1K$ Bytes. Este controlador además tiene 23 registros multi propósito.

2.1.2. ADC (Analog to Digital Converter)

El Arduino Uno cuenta con 6 pines analógicos, con los cuales se puede hacer una conversión analógica/digital con una precisión de 10 bits; sin embargo, tienen la limitación de solo poder transformar en un rango de 0 a VCC o alimentación, por lo que para este laboratorio, se alimenta el Arduino con 5V, para poder usar todo el rango disponible de conversión. También se puede usar una fuente interna del MCU como tensión de referencia de 1.1V, pero la misma no hace falta para la implementación deseada.

28.7. ADC Conversion Result

After the conversion is complete (ADCSRA.ADIF is set), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$\text{ADC} = \frac{V_{\text{IN}} \cdot 1024}{V_{\text{REF}}}$$

Figura 2: Descripción general del ADC [Atmel(2016)]

Basado en la fórmula anterior tomada de la hoja del fabricante, este es el valor que me daría a la salida el Arduino, por lo que si se quiere que en la salida se refleje un valor "real" se debe despejar el V_IN, de lo cual se obtiene:

$$V_{IN} = \frac{ADC \cdot V_{REF}}{1024} \tag{1}$$

NOTA: La entrada del Arduino está en un rango de 0 a 5V por lo que al usar la expresión anterior se obtiene exactamente esto, un rango de 0 a 5V lo cual no representa la tensión real que se está midiendo en el exterior, pero si representa la tensión de entrada.
Un ejemplo de la configuración de esto se muestra en seguida:

```
void loop(){

    //Hace la lectura analógica del pin en cuestión
    Var = analogRead(A0-A5);

    //Obtiene la tensión real
    Vreal = ((V1*5)/1023);

}
```

2.1.3. I/O Ports

Al igual que con el microcontrolador AT-Tiny usado en el laboratorio pasado, el ATmega328 cuenta con los registros DDxn y Portxn, y si bien es posible usar el Arduino de esta forma y programar los puertos a "pie", no es la convención ya que la misma plataforma cuenta con funciones que simplifican la programación increíblemente.

18.2.3. Switching Between Input and Output

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled {DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedance environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ({DDxn, PORTxn} = 0b00) or the output high state ({DDxn, PORTxn} = 0b11) as an intermediate step.

The following table summarizes the control signals for the pin value.

Table 18-1. Port Pin Configurations

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Figura 3: Descripción general del I/O [Atmel(2016)]

Un ejemplo de cómo se pueden encender o apagar pines se muestra en el siguiente código:

```
void setup(){
  //Configura los pines como salida
    pinMode(10,OUTPUT);
  //Configura los pines como entrada
    pinMode(11,INPUT);
}
void loop(){
    //Pone en alto el pin 10 configurado como entrada
    digitalWrite(10,HIGH);
}
```

Esto es especialmente útil, ya que se sabe que la corriente viaja de mayor potencial a menor potencial, y por ende se puede pensar en alguna configuración en que la corriente vaya de los pines hasta alguna tierra, pero esto ya depende del diseño físico. [Atmel(2016)]

2.1.4. Características eléctricas del ATMega328

Tomando la hoja de datos del microcontrolador, se obtienen las características eléctricas físicas del mismo. Esto es importante ya que permiten dimensionar resistencias, y componentes periféricos. Todo lo anterior con el fin de proteger los componentes y asegurar el buen funcionamiento del diseño.
Como se ve en la siguiente figura, cada pin del ATmega328 es capaz de entregar 40mA. Con el fin de proteger tanto los LEDs del diseño, y el microcontrolador, se utilizan resistencias de protección, las cuales se dimensionan en la sección siguiente. Además la corriente total que pueden entregar en total lo pines juntos es de 200mA según la hoja de datos, por lo que se debe cuidar que la suma de las corrientes cuando varios pines estén dando una salida no sobrepase los 200mA. [Atmel(2016)]
Además se muestra que la tensión máxima de operación es de 6.0V, sin embargo la alimentación del Arduino se hará con 5.0V con el fin de proteger y además para que todos los valores concuerden las formulas de conversión analógica digital.

32.1. Absolute Maximum Ratings

Table 32-1. Absolute Maximum Ratings

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0mA
DC Current V_{CC} and GND Pins	200.0mA

Figura 4: Características eléctricas del microcontrolador en cuestión. [Atmel(2016)]

2.2. Pantalla LCD PCD8544

Esto es una matriz de pixeles 48x84, típicamente se puede encontrar en el celular Nokia 5110, esta pantalla tiene la ventaja de que consume alrededor de 200μ A, lo que la hace muy versátil y buena para aplicaciones de bajo consumo energético. Para hacer uso de esta pantalla de forma fácil, rápida y eficaz se investigó y se llegó a conclusión de que la opción mas sencilla es usando estas dos librerías que se muestran a continuación junto con un ejemplo de cómo configurar la misma dentro de la plataforma Arduino y cómo conectarla eléctricamente.

```
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>
// Software SPI (slower updates, more flexible pin options):
    // pin 7 - Serial clock out (SCLK)
    // pin 6 - Serial data out (DIN)
    // pin 5 - Data/Command select (D/C)
    // pin 4 - LCD chip select (CS)
    // pin 3 - LCD reset (RST)
//Configuracion de pines para el display
Adafruit_PCD8544 display = Adafruit_PCD8544(7, 6, 5, 4, 3);
```

Esta última linea se encarga de crear una instancia de la clase **Adafruit_PCD8544** llamada display y la inicializa con los valores de los pines de salida del Arduino, en este caso los pines 7,6,5,4,3.
Como se muestra en el código anterior se usa el protocolo de comunicación SPI por software para poder explotar al máximo las capacidades de la librería y ahorrar la inclusión de la librería SPI. Si bien hacerlo de esta manera resulta más lento que hacerlo por Hardware, para la aplicación que se va a usar debería no haber diferencia.

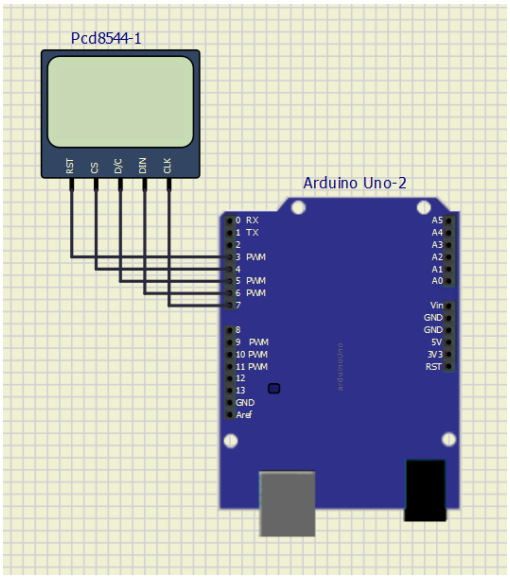


Figura 5: Ejemplo en hardware, del ejemplo en software que se dió anteriormente. (Creación Propia)

Las características eléctricas y parámetros físicos de la pantalla se resumen en la siguiente tabla:

The LCD 5110 display module uses SPI (Serial Peripheral Interface) communication protocol, the name of each connection pin is marked on the back of the LCD module as show below.

Table 1 – Pin Assignments of LCD 5110 Display Module

LCD	Arduino(Nano)	Function
1. VCC	5V	+VCC Power Supply from 2.7 – 5v
2. GND	GND	Ground
3. SCE	SS	SPI – Slave Select(SS), System Chip Enable when SCE=LOW
4. RST	RST	Reset the chip when RST=LOW
5. D/C	D8	Command(D/C=LOW)/Data(D/C=HIGH) mode
6. DN/MOSI	MOSI	SPI – Master Output, Slave Input (MOSI)
7. SCLK	SCK	SPI – Serial Clock (SCK)
8. LED	D7*	LED=HIGH, turn on back light LED

Figura 6: Características eléctricas de la pantalla en cuestión. [Atmel(2016)]

2.2.1. SPI

El Bus SPI (Serial Peripheral Interface) es un protocolo de comunicaciones, usado para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica).
Incluye una línea de reloj, dato entrante, dato saliente y un pin de chip select, que conecta o desconecta la operación del dispositivo con el que se desea comunicar. De esta forma, este estándar permite multiplexar las líneas de reloj.

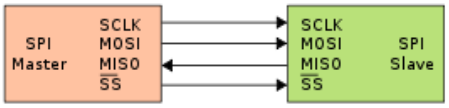


Figura 7: Diagrama general de la comunicación SPI. [SPI(2021)]

2.2.2. UART

UART por sus siglas en inglés (Asynchronous Receiver-Transmitter), Transmisor-Receptor Asíncrono es el dispositivo que controla los puertos y dispositivos en serie. Se encuentra integrado en la placa madre o en la tarjeta controladora del dispositivo. Se trata de un puerto de comunicación en paralelo, por lo que transfiere varios bits por ciclo de reloj.[**?**]

- Existen pines encargados de mover datos al otro lado.
- Existen pines encargados de recibir datos del otro extremo.
- Hay un pin de reloj que controla cuando se reciben señales y cuando se mandan.

Los puertos VGA viejos son ejemplos de UART. Los HDMI actuales son una evolución de UART.

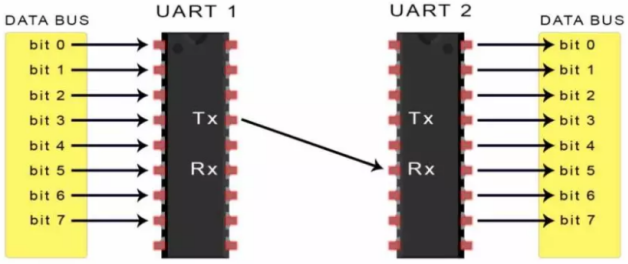


Figura 8: Esquema de funcionamiento UART[SPI(2021)]

2.3. Conversión Analógica/Digital

La conversión analógica/digital consiste en la transformación de señales analógicas (funciones continuas en el tiempo) en señales digitales (funciones discretas 1s y 0s), con el propósito de facilitar su procesamiento (codificación y compresión) y convertir la señal digital en una señal casi inmune al ruido y otras interferencias a las que son más vulnerables las señales analógicas.
Las señales analógicas en principio vienen de alguna fuente de información, ya sea tensión o de algún otro tipo, y tienen el inconveniente que pueden tomar cualquier valor, lo que no es deseado ya que las hace difíciles de trabajar. Es por esto que se debe transformar en una señal cuantizada, es decir que sea uniforme y con valores máximos y mínimos, sin intermedios.
Las señales digitales, al ser codificadas cuentan con sistemas y algoritmos de corrección de errores, con el fin de preservar el mensaje original y eliminar el ruido que puede introducir el canal de transmisión en este caso los cables y componentes pasivos del sistema. Además son de fácil procesamiento.
La conversión analógica digital más básica consiste del siguiente procedimiento:

1. **Muestreo:** Se debe leer la señal analógica y sacar pequeñas muestras de la misma, según el teorema de Nyquist, la frecuencia de muestreo está dada por el doble o más del doble de muestras que el ancho de banda de la señal en cuestión. Lo anterior con el fin de tener las muestras suficientes para reconstruir el mensaje de forma digital.

2. **Retención:** La retención se tiende a no estudiar, ya que consta de un modelo matemático que no siempre se aplica en la vida real, pero consta en guardar las muestras el tiempo suficiente para poder evaluarlas y saber si las mismas están bien tomadas o tienen algún error.

3. **Cuantificación:** Es este proceso se mide la tensión obtenida en cada una de las muestras y se decide si se toman como ceros o unos lógicos dependiendo de ese valor, ya en este punto se transforman los datos de forma analógica en datos binarios o digitales.

4. **Codificación:** La codificación consiste en asignarle una secuencia de bits única a cada uno de los símbolos disponibles.

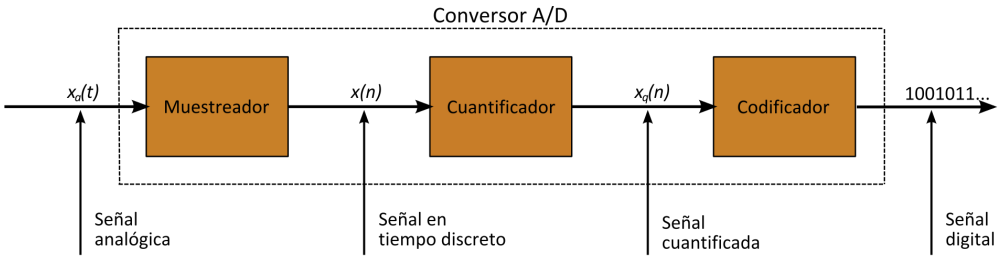


Figura 9: Diagrama General de un conversor analógico digital. (Creación Propia)

2.4. API Arduino:

El API (Application Programming Interface) de Arduino, es el conjunto de funciones, rutinas y subrutinas que facilitan la programación en el entorno de programación de Arduino. Estas se dividen en grandes bloques según su funcionalidad, entradas y salidas digitales, entradas y salidas analógicas, entradas y salidas avanzadas, tiempo, matemáticas, trigonometría, interrupciones y comunicación.

1. Entradas y salidas digitales:

   - **pinMode(numeropin, modo)**: Esta función recibe el numero de pin el que se quiere configurar, y el modo de cómo se desea configurar el pin dado. Aquí se designa si el pin se va a utilizar como entrada (INPUT) o salida (OUTPUT) y si se desea utilizar la resistencia de PULL UP, se usa el modo INPUT_PULLUP.
   - **digitalRead(numeropin)**: Esta función lee el estado de un pin en un momento dado. Este valor puede ser HIGH o LOW. Depende de la tensión medida podrá asignar un valor u otro. Para el rango de 0V a 1.5V el estado será LOW, para el rango 3.3V a 5V el estado será HIGH.
   - **digitalWrite(numeropin, valor)**: Permite asignar el valor HIGH o LOW al pin que se le pase como parámetro. Dependiendo de cómo se haya configurado ese pin con la función pinMode funcionará de una manera o de otra. Si se ha configurado el pin como OUTPUT se asigna un voltaje de 5V para el estado HIGH y 0V para el estado LOW. Si por el otro lado se ha configurado como INPUT, al asignar un valor HIGH a la entrada se habilita la resistencia de pull-up del pin y al asignar LOW dejará de funcionar dicha resistencia.

2. Entradas y salidas analógicas:
    - **analogReference(tipo)**: Esta función recibe un parámetro el cual sirve para configurar la tensión de referencia al medir un pin analógico. (Ver fórmula en la sección anterior)
    - **analogRead(numeropin)**: Al igual que la función digitalRead, esta función lee el estado del pin que se le pasa como parámetro, pero no le asigna un valor de alto o bajo, sino que esta lee el valor analógico en un rango de 0 a 1023 dependiendo del valor de tensión que se tenga en la entrada.
    - **analogWrite(numeropin, valor)**: Pone un valor analógico en ese pin, de 0 a 255, esta función genera una onda cuadrada.
3. Entradas y salidas avanzadas:
    - **tone()**: Esta función genera una onda cuadrada con un ciclo de trabajo del 50 %. Se utiliza para reproducir sonidos. Se ejecutará hasta que encuentre la sentencia noTone().
    - **noTone()**: Esta función detiene la función generadora de onda cuadrada tone().
4. Tiempo:
    - **millis()**: Esta función devuelve el tiempo en milisegundos desde que el Arduino comienza a ejecutar el programa que está cargado en la placa. El contador puede llegar a un valor máximo y causar un Overflow por lo que se reinicia a cero, esto ocurre pasados 50 días.
    - **micros()**: Igual que la función millis(), pero son micro segundos, el overflow ocurre aproximadamente a los 70 minutos.
    - **delay(milisegundos)**: Detiene la ejecución de un programa una cantidad de milisegundos.
    - **delayMicroseconds(microsegundos)**: Es igual que la función anterior pero con microsegundos.
5. Matemáticas:
    - **min(var1, var2)**: Devuelve el valor máximo entre dos valores (var1 y var2).
    - **max(var1, var2)**: Devuelve el valor mínimo entre dos valores (var1 y var2).
    - **abs(valor)**: Devuelve el valor absoluto del valor.
    - **sqrt(valor)**: Hace el cálculo de la raíz cuadrada de un número.
6. Trigonometría:
    - **sin(ang)**: Hace el cálculo del seno de un ángulo dado en radianes.
    - **cos(ang)**: Hace el cálculo del coseno de un ángulo dado en radianes.
    - **tan(ang)**: Hace el cálculo de la tangente de un ángulo dado en radianes.
7. Interrupciones:
    - **attachInterrupt(interrupcion, ISR, modo)**: Con esta función se indica que función se va a llamar cuando se produce una interrupción por los pines que permiten dicha interrupción. En Arduino UNO los pines digitales 2 y 3 permiten esta funcionalidad. Hay cuatro posibilidades de manejo para las interrupciones:
        - LOW: LLama a la interrupción cuando el pin se lee bajo.
        - CHANGE: LLama a la interrupción cuando el pin cambia de nivel.
        - RISING: LLama a la interrupción cuando el pin pase de 0 a 1 en flanco positivo.
        - FALLING: Llama a la función cuando el pin pasa de 1 a 0 en flanco negativo.
    - **noInterrupts()**: Desabilita por completo las interrupciones en el sistema.
8. Comunicación:
    - **Serial.begin(baud)**: Con esta función se inicia la comunicación serial con dispositivos fuera del Arduino. El parámetro baud dice los bits por segundo, lo típico son 9600.
    - **Serial.availabe()**: Deuelve el número de bytes disponibles para lectura en el puerto serial. Estos datos son almacenados en un buffer de 64 bytes. Su uso más común es para comprobar si se tienen datos para leer.
    - **Serial.read()**: Lee los datos en el puerto en serie.
    - **Serial.write()**: Escribe en el puente en serie, datos binarios.
    - **Serial.print()**: Escribe los datos en el puerto serial en forma de caracteres legibles para un ser humano.
    - **Serial.prinln()**: Es igual que Serial.print() pero con la diferencia que esta escribe un salto de línea al final de la secuencia de caracteres.

2.5. Divisores de Tensión:

Como uno de los requerimientos es que el voltímetro pueda medir tensiones en un rango de -12V a 12V y las mismas presentan dos problemas que hay que resolver, primero que los pines analógicos del Arduino solo pueden leer en un rango de 0V a 5V, y de la premisa anterior, solo puede medir tensiones positivas. Para resolver ambos problemas se tienen diversas alternativas, como lo sería un circuito con amplificadores operacionales capaz de obtener el valor absoluto de la señal, y configurarlo de manera que la salida se encuentre acotada en el rango deseado. Sin embargo, el simulador parece tener problemas con este circuito y en general los amplificadores operacionales, por lo que entonces se tomó la iniciativa de diseñar un divisor de tensión con dos fuentes, el diseño propuesto es capaz de resolver ambos problemas, con cierto margen de error pero es capaz de pasar de una tensión que varía entre -12V y 12V a una que va desde 0V a 5V.

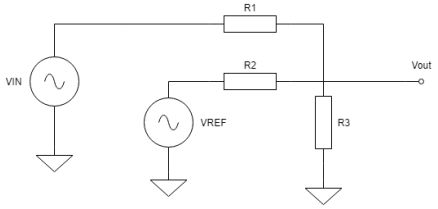


Figura 10: Diagrama General del divisor de tensión utilizado en el laboratorio. (Creación Propia)

El segundo problema se presenta al tratar de medir tensiones alternas, ya que la misma como lo dice su nombre oscila entre valores positivos y negativos de tensión, por lo que lo ideal sería implementar un rectificador de onda completa y obtener el valor RMS a partir de eso, pero de nuevo los puentes de diodos y los amplificadores operacionales no sirven en el simulador utilizado en el curso, por lo que se optó por usar el mismo divisor de tensión, ya que el efecto que tiene el mismo sobre una señal alterna, es acortarla en el rango deseado.
Para hacer el análisis de este circuito y obtener una expresión equivalente para V_out que sirva para poder diseñar las resistencias es necesario y por simplicidad utilizar el método de superposición, el cual consiste en apagar una fuente, obtener el aporte de la fuente restante a V_out, repetir con la fuente opuesta y el teorema dice que la suma de ambos aporte es igual a la tensión en cuestión.

$$V_{out} = V_1 \cdot K_1 + V_2 \cdot K_2 \tag{2}$$

Al apagar una de las fuentes, dos de las resistencias quedan en paralelo y al despejar el divisor de tensiones se obtiene:

$$V'_{out} = V_1 \cdot \frac{R_3//R_2}{R_1 + R_3//R_2} \tag{3}$$

Al hacer el procedimiento de nuevo pero invirtiendo las fuentes se obtiene:

$$V''_{out} = V_2 \cdot \frac{R_1//R_3}{R_1 + R_1//R_3} \tag{4}$$

Para una ecuación general:

$$V_{out} = V'_{out} + V''_{out} = V_1 \cdot \frac{R_3//R_2}{R_1 + R_3//R_2} + V_2 \cdot \frac{R_1//R_3}{R_1 + R_1//R_3} \tag{5}$$

Teniendo esto se puede pasar a la etapa de diseño, para la cual se conocen los valores de todas las tensiones, debido a que una es la entrada, y se conoce su rango, y la otra fuente puede ser una batería y se conoce su valor. Además se pueden colocar dos resistencias con valores típicos y despejar para la otra y así terminar con la primera etapa de diseño. Hechos los cálculos se obtiene la siguiente expresión para el diseño de R_2

$$R_2^2 + R_2(R_1//R_3) - \frac{V_1}{\left(V_{out} - \frac{V_2(R_1//R_3)}{R_1+R_1//R_3}\right) \cdot \frac{R_3}{R_1+R_3}} = 0 \tag{6}$$

Los valores diseñados se resumen en la siguiente tabla:

Componente	Valor
R_1	$6{,}35K\Omega$
R_2	$2{,}5K\Omega$
R_3	$5K\Omega$
V_1	$4{,}75V$

Cuadro 1: Resumen de los valores diseñados para el divisor de tensión. (Creación Propia)

En la siguiente figura se muestra cómo se configuran cada uno de los cuatro canales de entrada del voltímetro. Además se le agregó un LED de color con el fin de etiquetar cada canal para así saber diferenciarlos con la función de warning que notifica cuando el usuario mide una tensión más alta de 11.90 V o menor que -11.90V.

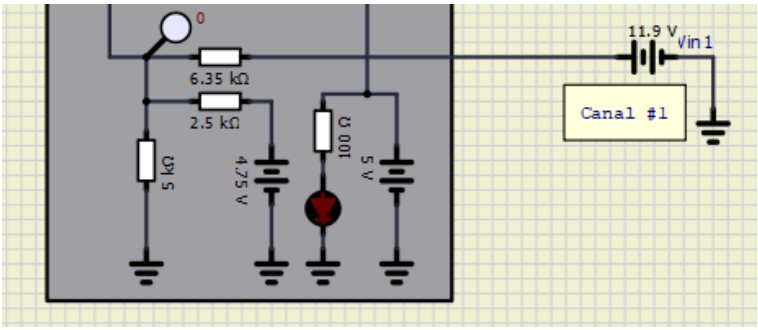


Figura 11: Implementación del divisor en el simulador del curso. (Creación Propia)

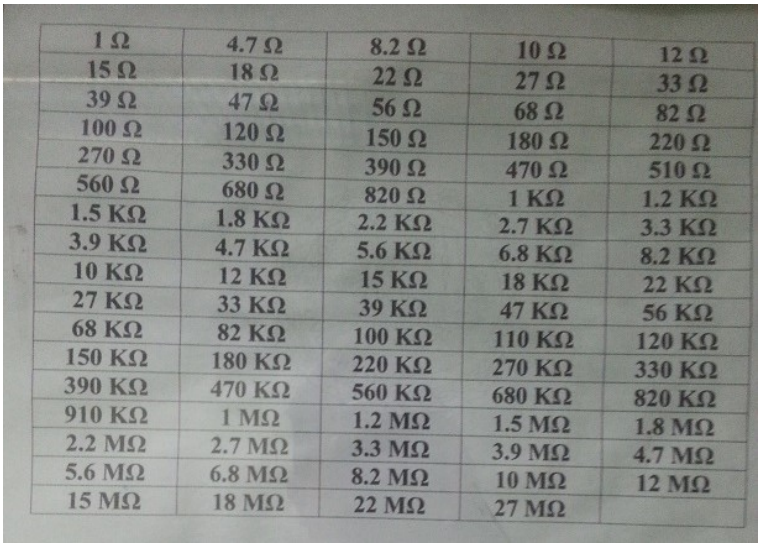
Cabe recalcar que si el usuario decide medir tensiones que se salen del rango estipulado, pueden dañar el microcontrolador, ya que el divisor de tensión en el canal no está diseñado para soportar más de $\|12V\|$, sin embargo, el divisor de tensión es lo suficientemente versátil como para poder medir rangos más grandes si se rediseñan sus resistencias.

2.6. Dimensionamiento de las resistencias de protección de los LED

Para todo ingeniero eléctrico, técnico en electrónica o entusiasta, es bien sabido que los LED son elementos eléctricos sumamente frágiles y tienden a quemarse por sobre corrientes muy fácilmente. Es por esto que siempre que se tiene un LED, el mismo debe ir acompañado de una resistencia en serie. El valor de esta resistencia va a estar dado por el resto de componentes del circuito. [Dorf and Svoboda(2011)]
Como en este caso se conoce que la corriente máxima que da el microcontrolador es de 40mA, se puede usar este dato para el dimensionamiento de la resistencia. Sin embargo por temas de protección del mismo microcontrolador, nunca se debe apuntar a utilizar el máximo, sino un poco menos, por lo que para este diseño se procurará estar por debajo de los 20mA por pin en todo momento.
Para ahorrar en resistencias, se quiere usar únicamente resistencias de 100Ω y como se quiere una corriente menor a los 20mA, se aplica ley de ohm y se verifica si la tensión obtenida es menor a la máxima que puede dar el microcontrolador.

$$V = IR \longrightarrow V = 20mA \cdot 100 = 2V \tag{7}$$

Como la tensión máxima del microcontrolador es 5V, el MCU es totalmente capaz de entregar 2v por pin.
Como eventualmente este diseño se quiere hacer de forma física, el valor de resistencia de 100Ω, se encuentra en la bodega de la escuela de ingeniería eléctrica.



1 Ω	4.7 Ω	8.2 Ω	10 Ω	12 Ω
15 Ω	18 Ω	22 Ω	27 Ω	33 Ω
39 Ω	47 Ω	56 Ω	68 Ω	82 Ω
100 Ω	120 Ω	150 Ω	180 Ω	220 Ω
270 Ω	330 Ω	390 Ω	470 Ω	510 Ω
560 Ω	680 Ω	820 Ω	1 KΩ	1.2 KΩ
1.5 KΩ	1.8 KΩ	2.2 KΩ	2.7 KΩ	3.3 KΩ
3.9 KΩ	4.7 KΩ	5.6 KΩ	6.8 KΩ	8.2 KΩ
10 KΩ	12 KΩ	15 KΩ	18 KΩ	22 KΩ
27 KΩ	33 KΩ	39 KΩ	47 KΩ	56 KΩ
68 KΩ	82 KΩ	100 KΩ	110 KΩ	120 KΩ
150 KΩ	180 KΩ	220 KΩ	270 KΩ	330 KΩ
390 KΩ	470 KΩ	560 KΩ	680 KΩ	820 KΩ
910 KΩ	1 MΩ	1.2 MΩ	1.5 MΩ	1.8 MΩ
2.2 MΩ	2.7 MΩ	3.3 MΩ	3.9 MΩ	4.7 MΩ
5.6 MΩ	6.8 MΩ	8.2 MΩ	10 MΩ	12 MΩ
15 MΩ	18 MΩ	22 MΩ	27 MΩ	

Figura 12: Resistencias Disponibles en bodega. (Creación Propia)

2.7. Tratando el rebote del botón de entrada

Cuando se enciende o apaga un interruptor, ya sea un switch, o un botón, la salida del mismo no es totalmente limpia, sino que presenta rebotes debido que los materiales físicos rebotan entre ellos. Esto es un problema ya que los rebotes pueden inevitablemente introducir falsas lecturas al sistema. [Boylestad and Nashelsky(2009)]

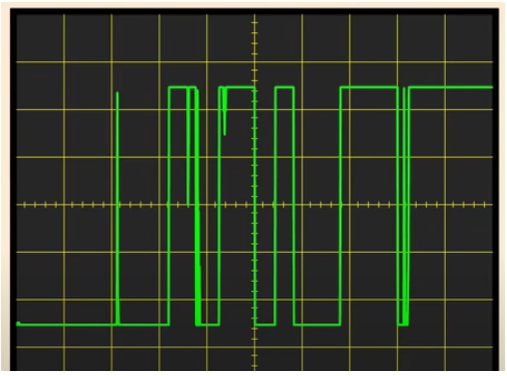


Figura 13: Efecto de rebote de un botón.[Christoffersen(2015)]

Existen diversas formas para contrarrestar este efecto, ya sea por hardware o por software. Para este diseño específico se decidió ir por la solución implementada en hardware, ya que es fácil, eficiente y barata de implementar.
Para esto se aplica un filtro con el fin de eliminar los rebotes del switch.

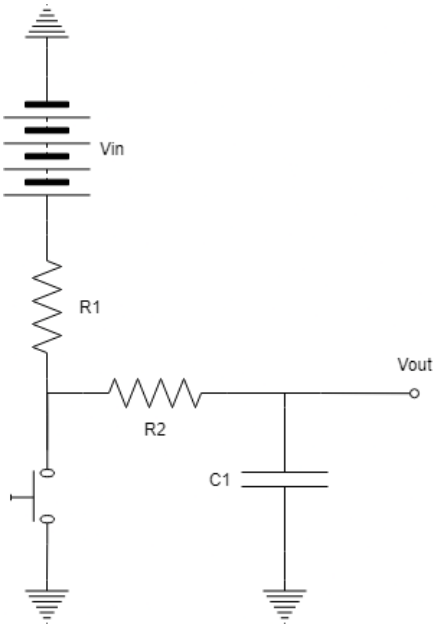


Figura 14: Filtro para eliminar el efecto de rebote del botón. (Creación Propia)

Como no se tiene una entrada sinusoidal, y además no se pretende filtrar ciertas frecuencias, no hay parámetros importantes para dimensionar el capacitor, con solo que se cargue y se descargue en un tiempo prudente es suficiente. Por esto mismo se escoge de los capacitores en bodega, uno de 100pF, lo cual se va a cargar sumamente rápido y lo mismo para la descarga, cumpliendo así el objetivo de filtrar la señal del switch.
Teniendo esto se pueden dimensionar las resistencias teniendo en cuenta el tema del tiempo. Cabe mencionar que ese circuito cirve tanto para cuando se trata de un botón o un switch o interruptor.

$$\tau = RC \longrightarrow 100\Omega \cdot 100pF \approx 10nS \tag{8}$$

Este valor es lo suficientemente pequeño como para que sea imperceptible por el usuario, además de que los tiempos de medición dentro del Arduino son mucho mayores por lo que esto es totalmente irrelevante.

3. Diseño Final del voltímetro digital

Tomando en consideración todas las secciones anteriores, se puede crear un diseño funcional para lo solicitado en el enunciado del laboratorio. Tomando en cuenta que se quiere simular un voltímetro digital, es necesario utilizar 4 LEDs, y 4 resistencias de protección para los mismos. Además se colocan otros cuatro LEDs en la entrada de cada canal con el objetivo de asignar un color único a cada canal y cuando se pase de los valores umbrales máximos se encienda un LED de emergencia del mismo color que el canal donde se está midiendo el exceso de tensión.

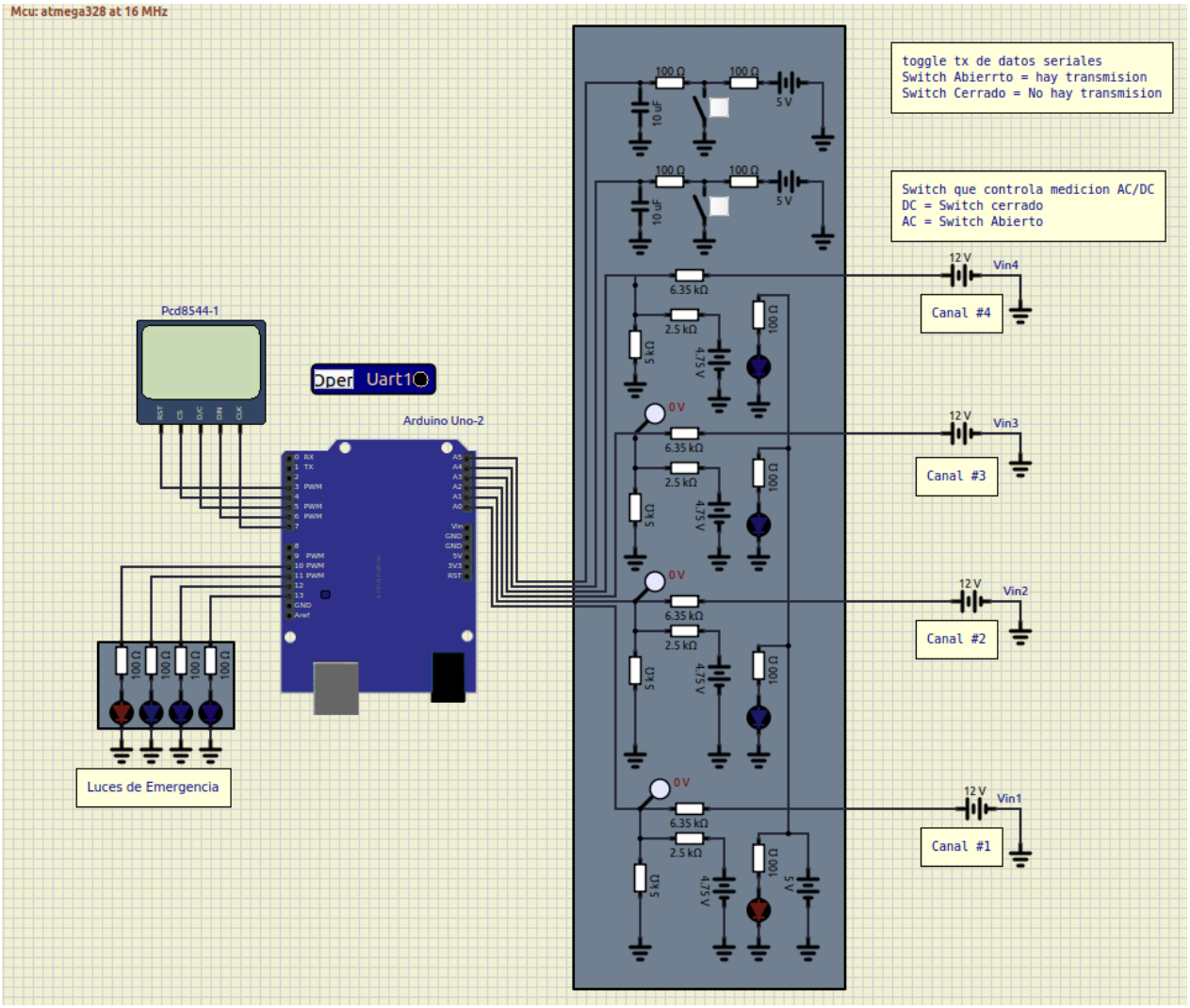


Figura 15: Esquemático del circuito montado. (Creación Propia)

Los componentes de los filtros de entrada para los switches son los mismos discutidos en la sección anterior, dos resistencias, un switch, la batería y el capacitor.
Buscando vendedores de los componentes anteriormente mencionados, se tiene el problema de que no venden la cantidad exacta de componentes; sin embargo esto no presenta un problema debido a que un paquete con 100 resistencias cuesta alrededor de 6 dólares, lo que sí sube considerablemente el precio del proyecto, pero sigue siendo un proyecto que sale por aproximadamente 30 mil colones. (Las capturas de los precios se presentan en la sección de anexos)

Componente	Cantidad	Precio
Arduino UNO	1	$29
Resistencias Paquete	1	$15
Switch	2	$8
LED Paquete	1	$6
Total	$58	

Cuadro 2: Tabla de costos y componentes. (Creación Propia)

4. Análisis de resultados:

Primeramente se muestran las diferentes características y requerimientos del voltímetro digital:

4.1. Medición de tensión DC

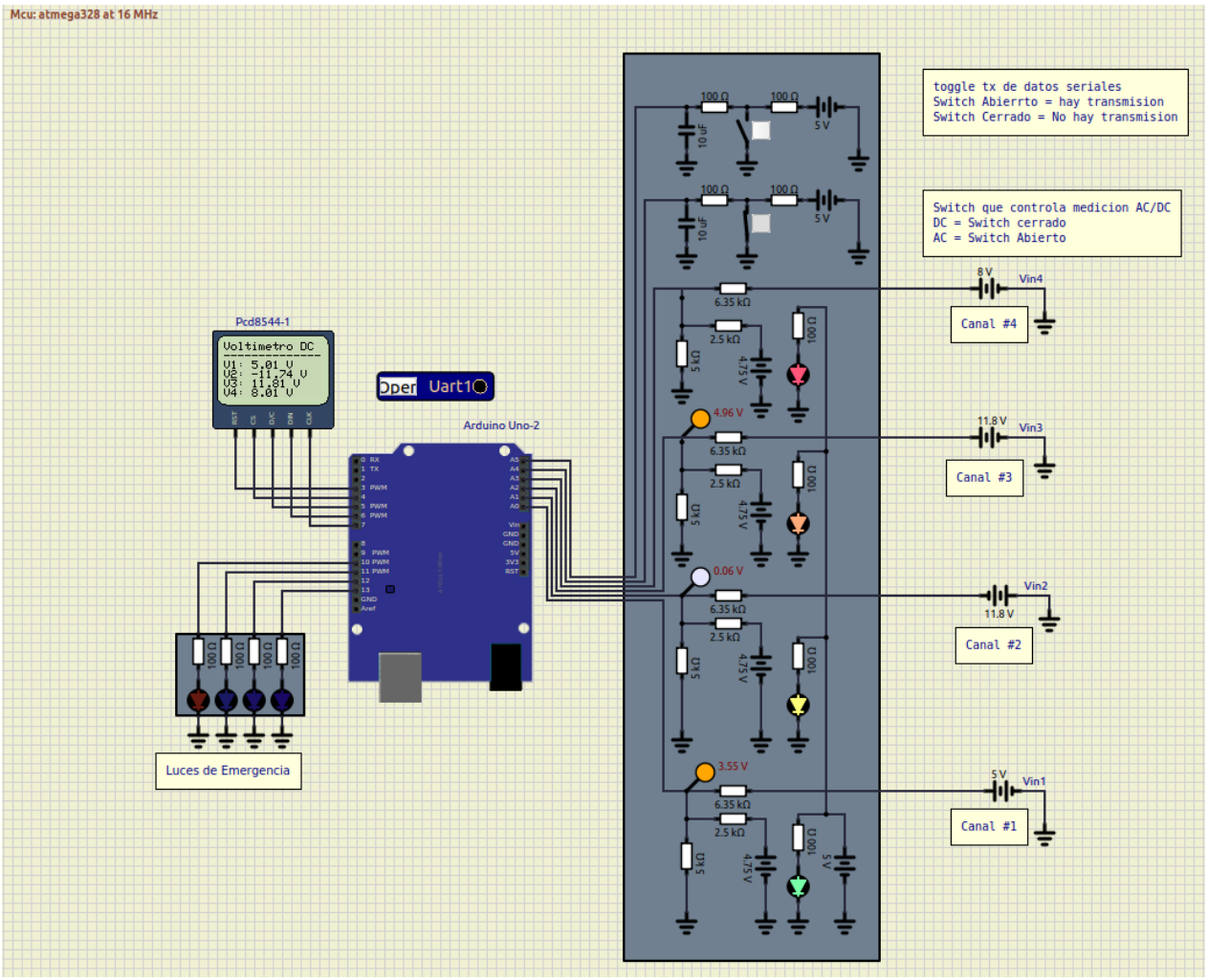


Figura 16: Ejemplo de medición DC. (Creación Propia)

De la sección de diseño eléctrico se sabe que el la entrada puede variar en un rango de [-12V a 12V], pero a la salida del divisor de tensión se tiene un rango de [0V a 5V] siempre y cuando la entrada se encuentre en el rango antes mencionado.
Sin embargo, la tensión que mide el Arduino en sus pines analógicos se encuentra en un rango de [0V a 5V], por lo que si se desea mostrar el valor medido real, se debe transformar por Software.
Para atacar este problema, nuevamente se va a utilizar la expresión matemática del ADC:

$$V_{IN} = \frac{ADC \cdot V_{REF}}{1023} = \frac{ADC \cdot 5}{1023}$$

Esta función retorna un valor dentro de un rango [0V a 5V], por lo que la idea que se tuvo fue pasar el rango de valores de [0 a 5] a [0 a 24], hecho esto, se procede a restarle 12 a ese valor, por lo que si se lee un cero, al restarle 12, a la salida se tiene un -12, y si se lee un 5, se estaría transformando en un 24 y al restarle 12, el valor real de lectura sería 12 y se cumple el objetivo con la medición DC. Por regla de 3 se puede transformar el rango de valores, el valor de la ganancia obtenida para esto es: 4.8

```
V1 = analogRead(A0);
v1k = (((V1*5)/1023)*k1)-12;
```

4.2. Medición de tensión AC

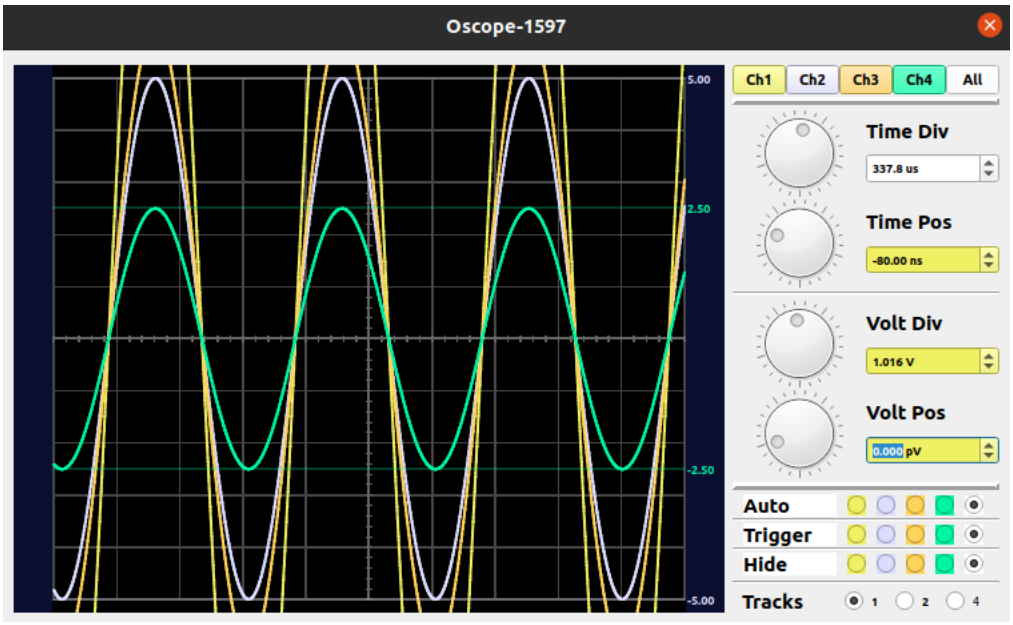


Figura 17: Input AC. (Creación Propia)

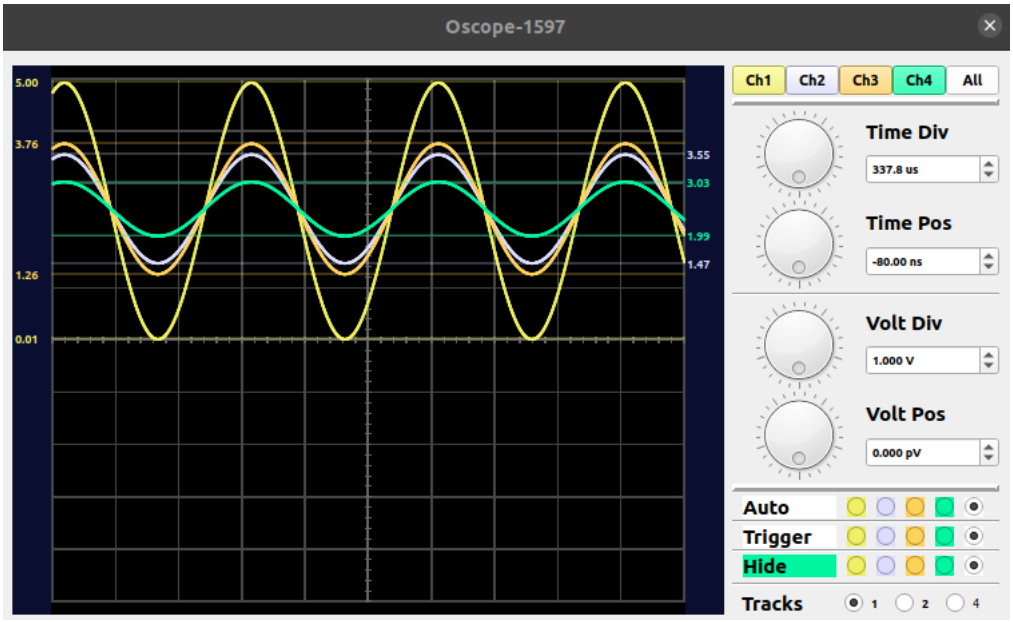


Figura 18: Output del divisor de tensión ante una señal AC. (Creación Propia)

Para comprobar el funcionamiento del divisor de tensión ante una señal de tensión alterna, se muestran ambas figuras generadas por el osciloscopio, y como se ve la señal con amplitud 12 se transforma en una con amplitud de 2.5V, por lo que se puede decir que el divisor va a limitar la amplitud y siempre y cuando la tensión de entrada no supere los 12V de amplitud el divisor la va a limitar a los valores de entendimiento del Arduino. Como se quiere medir una tensión alterna, las mismas se miden en valor RMS, para lo cual se divide la amplitud de la onda entre raíz de 2, por lo que entonces surge un problema, cómo obtener la tensión máxima y encontrar la amplitud de la onda.

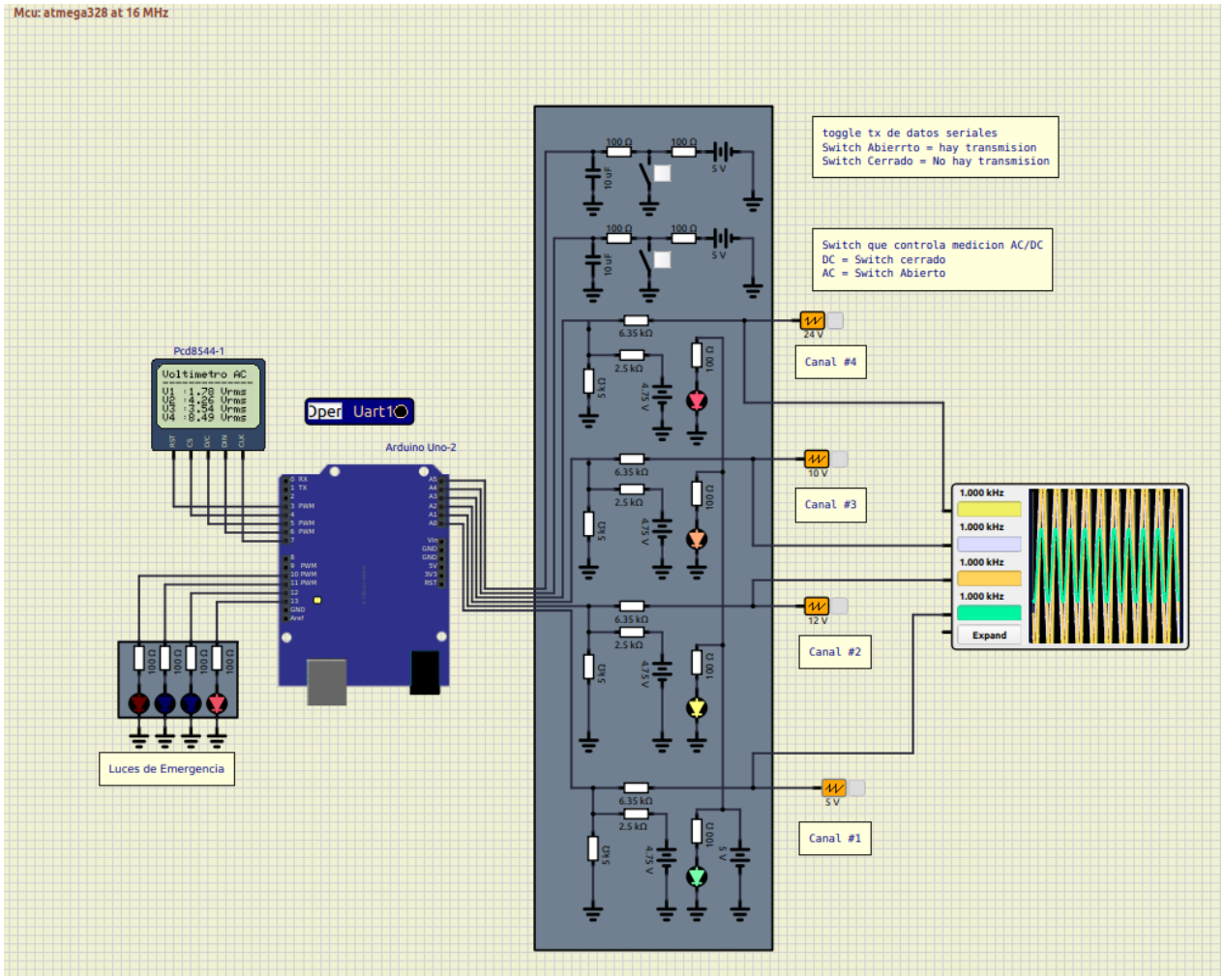


Figura 19: Medición AC. (Creación Propia)

Como se muestra en la figura anterior, los valores que se ven en pantalla están en RMS, por lo que también hay que ajustar el valor de tensión máxima para gatillar la alerta que va a encender el LED de emergencia. El código utilizado para obtener el valor de la amplitud de la señal se muestra en seguida:

```
float get_max_v1() {
  float max_v = 0.00;
  for(int i = 0; i < 100; i++) {
    float r = analogRead(A0);  // Lee el valor en el primer puerto (A0)
    if(max_v < r) max_v = r;
    // El valor del delay debe ser de 200 debido a la frecuencia de la señal
    delayMicroseconds(200);
  }
  return max_v;
}
```

Hecho esto, se puede usar el mismo método que para la medición DC; con un paso extra, ya que se debe dividir entre raíz de 2.

```
    float VAC1 = get_max_v1(); //Obtiene la amplitud
    VAC1 = ((((VAC1*5)/1023)*k1)-12);
    if ((VAC1+0.65) == 12) VAC1 += 0.65; //Hace el ajuste de la tensión
    VAC1 /= sqrt(2); // Obtiene el valor RMS
```

4.3. Luces de emergencia

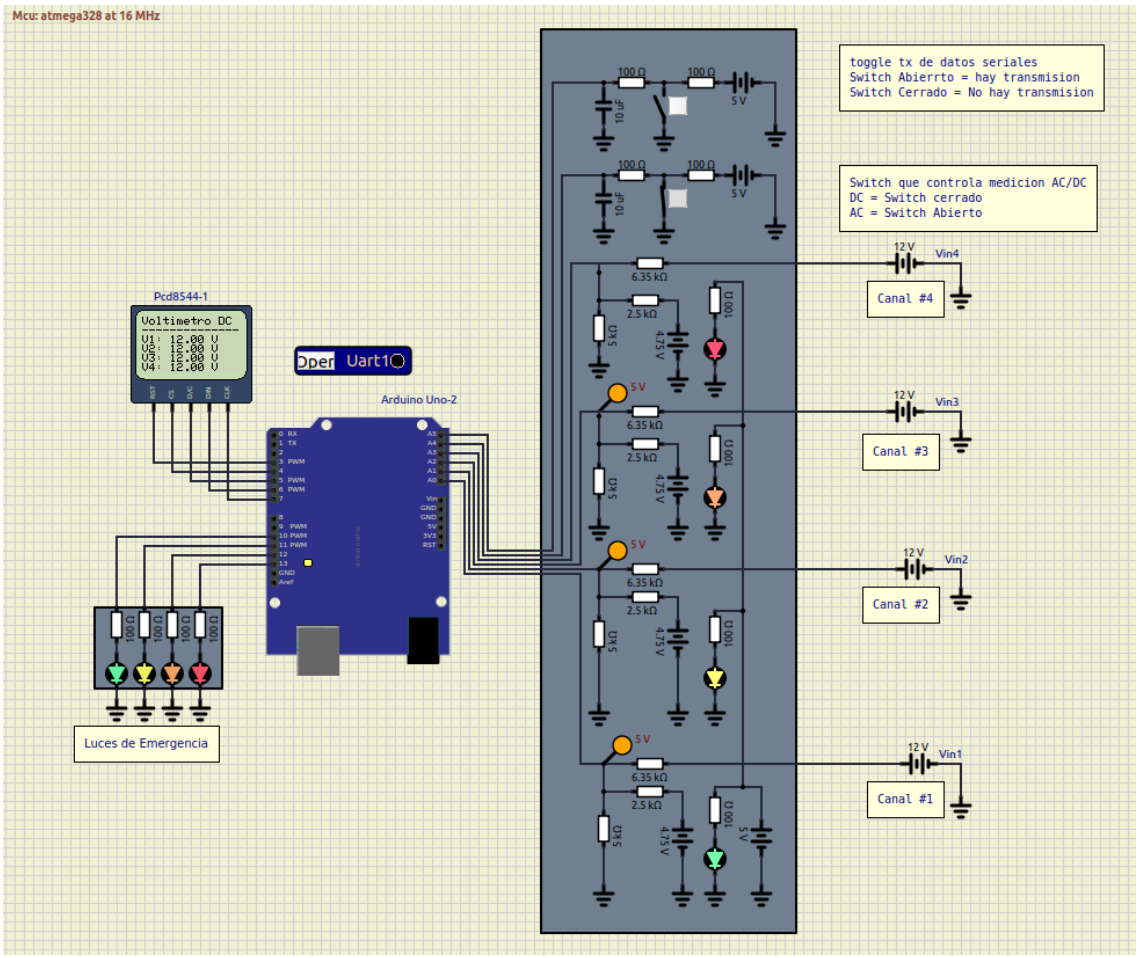


Figura 20: Ejemplo de las luces de emergencia. (Creación Propia)

Para construir la lógica para las luces de emergencia, se hace un simple **if** en el cual si la lectura del pin digital de ese canal es mayor a 11.90 o menor a -11.90.

```
void warning_DC(float v1, float v2, float v3, float v4){
  if((v1 >= 11.90||  (v1 <= -11.90))){
      digitalWrite(10, HIGH);
    }
  else{
      digitalWrite(10, LOW); //Mientras no pase del valor de umbral mantiene el led apagado
    }
  .
  .
  .
  if (v4 >= 11.90 || v4 <= -11.90){
      digitalWrite(13, HIGH);
    }
  else{
      digitalWrite(13, LOW); //Mientras no pase del valor de umbral mantiene el led apagado
    }
}
```

Para hacer este procedimiento con la tensión alterna, se hace lo mismo pero con el valor RMS.

4.4. Transmisión Serial de datos

El Arduino tiene la capacidad de mandar lecturas de forma serial utilizando un protocolo de comunicación llamado SPI del cual se habló en secciones anteriores.
Como se está trabajando con un simulador y no con el MCU real, se debe abrir el canal de comunicación de forma virtual o mas bien de forma emulada y el simulador permite comunicarse por este canal. Por el lado de la computadora se tiene un Script de python, con un ciclo infinito el cual va a estar leyendo los valores que transmite el microcontrolador.

4.4.1. Comunicación serial desde Arduino:

El API de Arduino presenta una serie de funciones pre hechas las cuales sirven para facilitar la vida, como por ejemplo lo sería el void setup() o el void loop(). En la función de setup, se inicializan todos pines de I/O y los puertos de comunicación. el código se muestra en seguida:

```
void setup(){
  Serial.begin(9600); //Inicio del puerto Serial
  //Configira los pines como salida
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
  //Inicia la pantalla y se selecciona el contraste
  display.begin();
  display.setContrast(75);

  //Muestra la flor de Adafruit
  display.display();
  delay(2000);
  display.clearDisplay();   //Limpia el display
}
```

Para escribir datos en el puerto serial, se utiliza también funciones incorporadas en el API de Arduino.

```
void loop(){
  boton = analogRead(A4); // Revisa si el boton de modo (AC/DC) esta encendido
  interruptor = analogRead(A5);

  if (interruptor > 3.00){
    while (etiqueta){
      Serial.println("Canal 1");
      Serial.println("Canal 2");
      Serial.println("Canal 3");
      Serial.println("Canal 4");
      Serial.println("AC/DC");
      etiqueta = false;
    }
  }
```

En la siguiente figura se muestra cómo el Arduino manda los datos al puerto indicado.

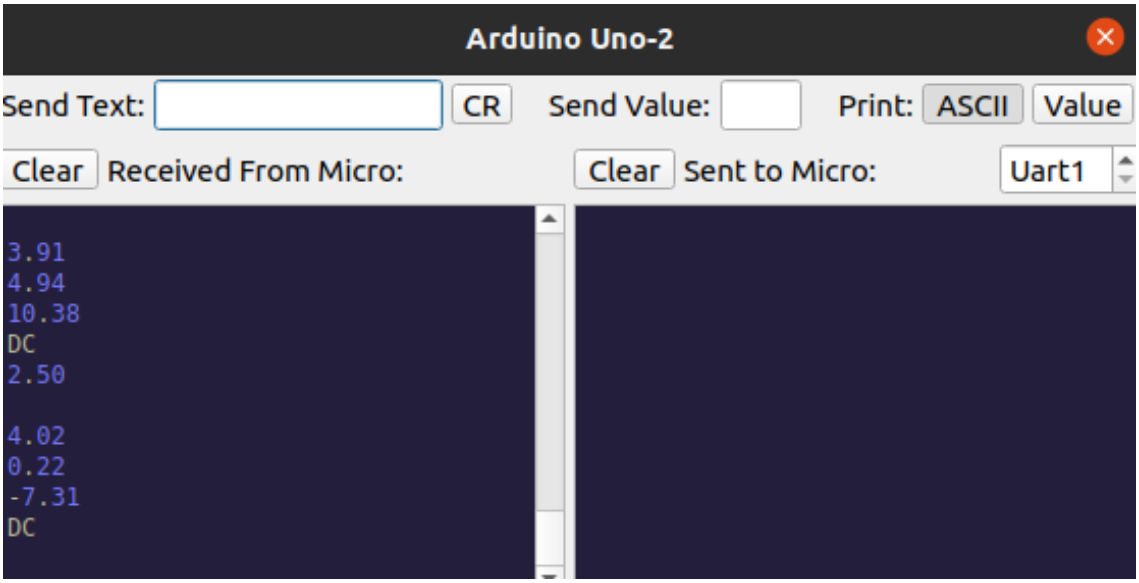


Figura 21: Info siento enviada en el puerto Serial del Arduino. (Creación Propia)

4.4.2. Comunicación serial desde Python:

Por el otro lado, en la computadora propiamente, se reciben los datos para posteriormente procesarlos y archivarlos en un documento de tipo CSV.

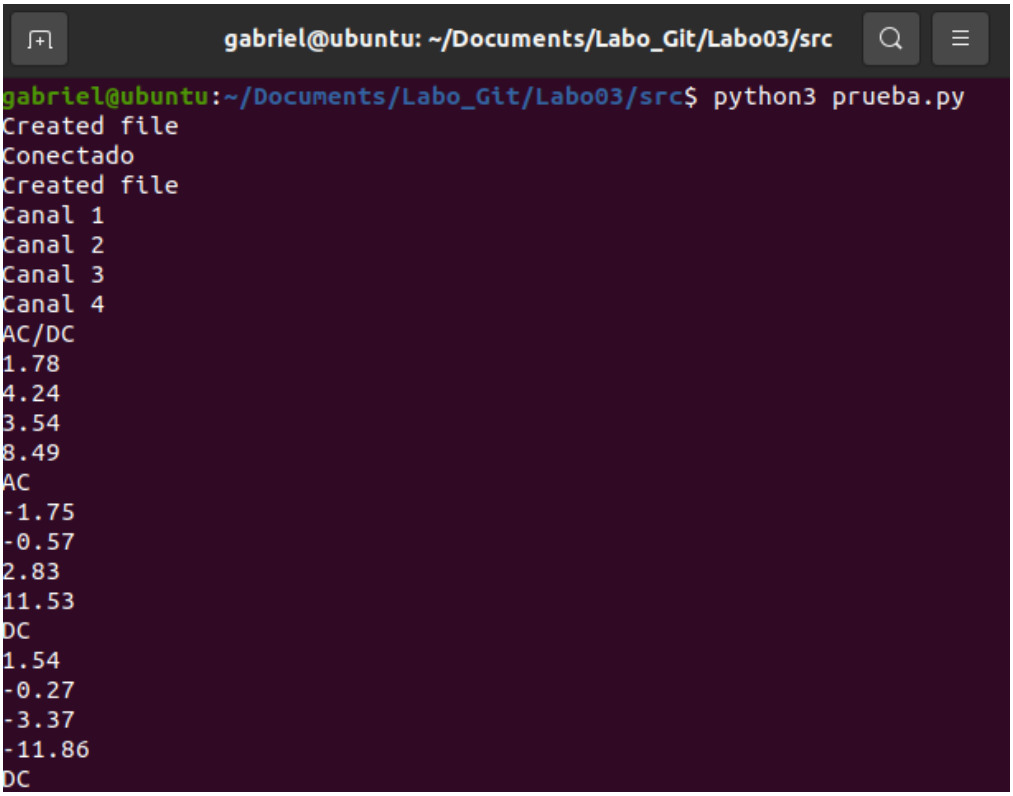


Figura 22: Datos seriales siendo recibidos con python. (Creación Propia)

	A	B	C	D	E
1	Canal 1	Canal 2	Canal 3	Canal 4	AC/DC
2	1.78	4.24	3.54	8.49	AC
3	-1.75	-0.57	2.83	11.53	DC
4	1.54	-0.27	-3.37	-11.86	DC
5	1.78	4.26	3.54	8.47	AC
6	-2.22	-5.78	-2.87	0.93	DC
7	2.01	1.33	-2.26	-11.01	DC
8	-2.22	-2.38	1.42	9.98	DC
9	1.78	4.26	3.54	8.49	AC
10	2.45	3.67	-0.2	-8.06	DC
11	-2.03	-1.54	2.12	10.85	DC
12	-2.26	-5.71	-2.76	1.3	DC
13	-0.13	3.6	4.87	10.76	DC
14	1.78	4.26	3.54	8.49	AC
15	-1.75	-0.57	2.83	11.55	DC
16	0.74	-2.36	-4.42	-11.62	DC
17	1.56	-0.2	-3.32	-11.84	DC
18	-0.57	-4.8	-4.96	-8.65	DC
19	-2.22	-5.78	-2.92	0.79	DC

Figura 23: Archivo CSV generado por python. (Creación Propia)

El código python se encarga de agarrar los datos que le envía el micro controlador, formatearlos y así pasarlos al formato CSV.

```
import serial
baud = 9600 #sample rate
filename = "datos.csv" #Nombre del archivo de texto que se va a crear
samples = 29 # va a tomar 5 mediciones de tension
file = open(filename, 'w') #Crea el archivo de texto
file.close()# cierra el archivo de texto
print("Created file")
ser = serial.Serial("/tmp/ttyS1", baud) #Abre la comunicacion con el arduino usando el puerto
print("Conectado")
file = open(filename, 'w')
print("Created file")
line = 0
contador  = 0
datos = []
while (1):
    getData = str(ser.readline())
    data = getData[2:][:-5]
    print(data)

    if contador == 4:
        file = open(filename, "a")
        file.write(data + "\n")
        contador = 0
    else:
        file.write(data + ",")
        contador+=1
```

4.5. Código

El Código para esta implementación es bastante extenso por lo que no se mostrará todo, además que se han venido mostrando fragmentos en son de ejemplos.
Sin embargo el código para este laboratorio, se divide en dos grandes secciones que se fueron trabajando en paralelo, pero realmente son independientes.

4.5.1. Escritura en la pantalla

Como se mostró, en la sección de la nota teórica en la cual se habla del comportamiento de la pantalla, primero se debe inicializar la misma incluyendo las librerías necesarias y crear la instancia de la pantalla. Hecho esto se tiene acceso a todos los métodos de la clase, los cuales son útiles para escribir mensajes en pantalla y facilitan mucho el trabajo.

```
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>
// Software SPI (slower updates, more flexible pin options):
    // pin 7 - Serial clock out (SCLK)
    // pin 6 - Serial data out (DIN)
    // pin 5 - Data/Command select (D/C)
    // pin 4 - LCD chip select (CS)
    // pin 3 - LCD reset (RST)
//Configuracion de pines para el display
Adafruit_PCD8544 display = Adafruit_PCD8544(7, 6, 5, 4, 3);

void setup(){
  //Inicia la pantalla y se selecciona el contraste
  display.begin();
  display.setContrast(75);

  //Muestra la flor de Adafruit
  display.display();
  delay(2000);
  display.clearDisplay();   //Limpia el display

}
```

De esta forma, se imprimen los valores en pantalla.

4.5.2. Escritura/Lectura en el Puerto Serial

Al igual que con la pantalla, existen funciones predefinidas capaces de facilitar mucho los cálculos y la implementación general.

```
void setup(){
  Serial.begin(9600); //Inicio del puerto Serial
  // Imprime en el puente serial lo que se le pasa como parámetro
  Serial.println("Canal 1");
  Serial.println("Canal 2");
  Serial.println("Canal 3");
  Serial.println("Canal 4");
  Serial.println("AC/DC");
}
```

5. Conclusiones y recomendaciones

Primeramente, se cumplió el objetivo de tal manera que el circuito cumple con todas las especificaciones de diseño y sirve bien.
Se tuvieron varios problemas al intentar implementar las cosas de primera mano o con lo primero que se vino a la cabeza. Debido a problemas con el simulador, los puentes de diodos no funcionaron y los circuitos activos con amplificadores operacionales tampoco. Por esto mismo es que se tuvo que usar únicamente el divisor de tensión.
Sin embargo programar en Arduino fue una experiencia mucho menos frustrante que con los otros microcontroladores, debido a que cuenta con mucha más memoria y el API de Arduino facilita mucho la programación, aún así debido a la cantidad de requisitos presentes en el enunciado de este laboratorio, ha sido el que más ha tomado tiempo, debido a que cada una de los requisitos hubo que probarlos y corregir errores hasta que funcionaran bien.
Además y a modo de recomendación es mejor buscar los componentes en tiendas dentro del país, para ahorrar en envío y demás.

Referencias

[Atmel(2016)] Atmel. Atmega328 data sheet, 2016.

[SPI(2021)] I2C, SPI y UART: Las interfaces de E/S a baja velocidad., 2021. URL `https://hardzone.es/reportajes/que-es/i2c-spi-uart/`.

[Dorf and Svoboda(2011)] R. Dorf and J. Svoboda. *Circuitos Eléctricos*. Alfaomega, 8 edition, 2011.

[Boylestad and Nashelsky(2009)] R. Boylestad and L. Nashelsky. *Electrónica: Teoría de Circuitos y Dispositivos Electrónicos*. Pearson Educación, 10 edition, 2009.

[Christoffersen(2015)] Jens Christoffersen. Switch bounce and how to deal with it. `https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/` , 2015.

[Podkalicki(2018)] Lukasz Podkalicki. Microcontrollers and basic bit-level operations. `https://blog.podkalicki.com/microcontrollers-and-basic-bit-level-operations` , 2018.

6. Anexos

Figura 24: Precio para el MCU

Figura 26: Precio para los LED

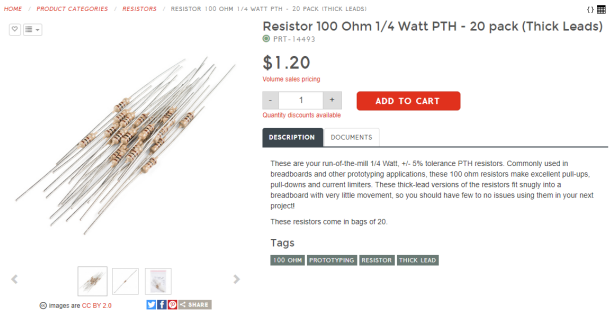


Figura 25: Precio para la resistencia de 100ohms

Figura 27: Precio para conjunto de resistencias.

Figura 28: Precio para el Switch.

Figura 29: Precio para el Capacitor.