Practica 2: ADC-DAC con microcontrolador

Univ. Juan Pablo Crespo Vargas

***Resumen*—Se estudia los periféricos de salida de un microcontrolador Atmega328p, bajo en entorno de desarrollo Arduino. Se estudia e implementa un circuito para mostrar el uso de los puertos de salida usando un arreglo de leds. Se muestra el uso del conversor análogo digital ADC y el conversor digital análogo DAC, desarrollando un generador de onda senoidal, cuadrática y diente de sierra, así como también, se desarrolla un osciloscopio.**

I. Resumen Teórico

Arduino [1] es una compañía de hardware libre y una comunidad que realiza el diseño y manufactura de placas de desarrollo de hardware y software, compuesto por circuitos impresos que integran un microcontrolador de la familia Atmel (variando el microcontrolador dependiendo el modelo usado) y un entorno de desarrollo (IDE), en donde se compila y programa cada placa.

Los microcontroladores son circuitos integrados que pueden ser programados para ejecutar órdenes grabadas en su memoria. Esta compuesto de varios bloques, que se pueden resumir en tres categorías: Procesadores, Memorias, Periféricos de entrada/salida.

Se estudia el uso dos periféricos de entrada/salida en específico; El conversor análogo digital 'ADC' (Analog to Digital Conversor), y el conversor digital análogo 'DAC' (Digital to Analog Conversor).

I-A. Conversor Análogo digital ADC

Los microcontroladores tienen la capacidad de detectar y trabajar bajo señales binarias de voltaje, es decir, que solo reconocen el uso de dos valores de voltaje; se considera 1 binario al estado HIGH, o alto, cuando se alcanza el valor mayor de voltaje, y se considera 0 binario al estado LOW, o bajo, cuando se alcanza el valor de menor voltaje. En los microcontroladores de la familia Atmel, este voltaje corresponde a 5 voltios como el valor alto, y 0 voltios al valor mas bajo. Estos valores se deben a que se usan transistores bipolares (TTL) para diseñar estos microcontroladores.

Este sistema binario, no permite usar valores intermedios de voltaje, como por ejemplo 0.1 voltios o 4.98 voltios. Muchos sistemas con los que el microcontrolador interacciona en el mundo real son analógicas, lo que significa que estas señales no se restringen a solo dos valores, sino a infinitos valores en un rango, como por ejemplo una señal senoidal, que varia infinitamente desde -1 hasta 1, dependiendo del angulo evaluado.

Para poder trabajar con este tipo de señales en un microcontrolador, se hace una transformación de valores analógicos a binarios. De manera que puedan representarse valores que puedan ser tratador por el ordenador.

En los microcontroladores de la familia Atmel, se incluye un componente denominado ADC como una entrada en un pin determinado. Este realiza la conversión de una señal analógica a un valor binario que puede ser representado por un número de bits. Este número depende del microcontrolador. En el Arduino UNO y NANO, que se usa en esta experiencia, el microcontrolador Atmega328p tiene una asignación de 10 bits para convertir esta señal, es decir, tiene un total de $2^10 = 1024$ valores distintos. Otros Arduinos como el DUE y ZERO tienen asignados 12 bits para el ADC, es decir, $2^12 = 4096$ niveles.

Los rangos de trabajo para los microcontroladores Atmel, debida su fabricación con transistores bipolares, van desde los 0 a 5 voltios, no pudiendo utilizar valores mayores para convertirlos en el ADC. Este valor de 5 voltios es el valor de referencia para el conversor. El ADC en el Arduino UNO y NANO tiene 1024 valores, donde asigna el valor de 0 a el voltaje de 0 voltios, y asigna el valor de 1023 a el voltaje de 5 voltios. Todos los valores intermedios entre 0 y 5 voltios corresponden a valores escalonados.

Para calcular el valor de cada escalón, se realiza una simple división $5V/1024 = 4{,}88mV$, entonces cada $4{,}88mV$ se asigna un nuevo valor. Los valores intermedios entre un 0 y 4.88 mv no son tomados en cuenta para la conversión.

I-A1. Conversor ADC en el IDE ARDUINO: El procedimiento para usar el ADC en el Arduino es simplificado al uso y conversión en el microcontrolador al usar un lenguaje de bajo nivel.

En la placa del Arduino UNO, se tienen referenciadas 6 salidas digitales, como entradas analógicas, desde A0 hasta A6. Estas salidas están pre programadas para ser usadas como entradas analógicas, por lo cual no es necesario iniciarlas como entradas.

Para esto solo es necesario tomar la instrucción 'analogRead()', cuyo argumento tiene a las entradas A0 hasta A6. Esta instrucción hace una conversión de voltaje a binario usando 10 bits que no pueden reducirse ni aumentarse en el entorno IDE de Arduino. El tiempo de conversión, que es proveído por el fabricante, toma 100 microsegundos desde la lectura de la entrada análoga. Por lo cual solo puede hacer un máximo de 10000 conversiones por segundo. Este valor tampoco puede ser modificado desde el IDE de Arduino.

I-B. Conversor Digital Análogo DAC

Los microcontroladores de la familia Atmel usados en los Arduinos UNO y NANO, no tienen asignado ningún componente que realice la conversión de un número binario a un valor de voltaje de manera directa. Sin embargo existen dos técnicas que nos permiten simular una salida digital; utilizando

una salida especial denominada PWM que se encuentra en los microcontroladores Atmel, y a través de un componente externo a nuestra plataforma, generalmente un chip o módulo para el Arduino, que tome los valores del binarios del Arduino, y los transforme en valores de voltaje correspondientes. Estos chips DAC se comunican con el Arduino, obteniendo los valores de este en binario por los puertos paralelos del Arduino, y convirtiendo el dato en un valor de voltaje.

Para realizar la conversión DAC, se trabaja con el PWM que esta disponible en los Arduinos, de manera mas conveniente que optar por un módulo externo al Arduino.

I-B1. Modulación por Ancho de Pulso PWM: La modulación por ancho o de pulso (PWM por sus siglas en inglés Pulse Width Modulation) es un tipo de señal de voltaje utilizado para enviar información o para modificar la cantidad de voltaje que se envía a una carga. Esta salida digital, emula una salida analógica, es decir, puede generar distintos tipos de voltaje entre los valores que corresponden al HIGH y LOW de la salida binaria, de 5 a 0 voltios.

Para poder realizar esta conversión el microcontrolador utiliza un onda cuadrada que se genera cuando el valor LOW de una salida digital pasa a HIGH, por un determinado tiempo, y luego vuelve a LOW por otro determinado tiempo. El conjunto de estos tiempos se llama ciclo de trabajo y sus unidades se representan en términos del porcentaje. El tiempo de duración de un ciclo, se denomina periodo, y este se repite constantemente como una señal periódica común. Matemáticamente el ciclo de trabajo se representa por la expresión 1:

$$D = \frac{t}{T} * 100\,\% \tag{1}$$

I-B2. Principio de funcionamiento PWM: Supongamos una señal que varia en el tiempo como en la figura 1, representa una onda rectangular, se evalúa la variación de esta señal a través del periodo y se calcula el promedio:

$$y = \frac{1}{T}\int T_0 f(t)dt$$
$$y = \frac{1}{T}\int DT_0 5dt$$
$$y = \frac{1}{T},5DT$$
$$y = D * 5$$

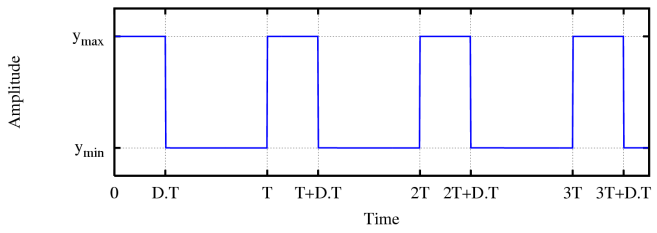


Figura 1. Señal de un PWM

El valor de salida promedio, depende del porcentaje que representa el ciclo de trabajo.

En el Arduino, las frecuencias que corresponden al periodo de una PWM no pueden modificarse, pues el entorno de desarrollo usa los valores máximos de estos, sin brindar la posibilidad de cambiarlos. Sin embargo estos valores si pueden modificarse al usar el microcontrolador en un nivel mas bajo, al realizar estos cambios de valor HIGH y LOW, utilizando para ello un timer interno del microcontrolador. Este timer en el Arduino es de 490Hz, y de 980Hz en los pines 5 y 6 del Arduino UNO. La instrucción analogWrite(pin, value), permite hacer el uso de la salida PWM del Arduino, sin embargo el valor a ser convertido depende de un número de 0 entre 255, asignando el valor de 0 para el voltaje menor a convertir, 0 volts, y asignando el valor de 255 para el valor mayor, que corresponde al valor máximo de salida, 5 volts.

Esto se debe a la asignación de bits que el microcontrolador tiene para el PWM. En el caso de los controladores Atmel, este valor es de 8 bits, teniendo $2^8 = 256$ valores.

II. Desarrollo de la práctica

Se realizan los ejercicios para manejar los periféricos de salida, para el Arduino UNO, estos están dados por el microcontrolador Atmega328p.

II-A. Parte 1 Puerto Paralelo

Conectar 8 leds al puerto paralelo B, y un pulsador a un bit del puerto D. Construir un programa para que los leds se enciendan y se apaguen cada 0.3segundos.

1. Uno a la vez iniciando en PB0 a PB7
2. Uno a la vez iniciando en PB7 a PB0
3. Cuatro y cuatro no

El cambio de secuencia se da cada vez que se presiona el pulsador.

Para realizar esta parte se armo un circuito como indica la figura 2, el cual consta de 8 leds, un pulsador y dos resistencias de 340Ω para protección de corriente para los leds y el pulsador.

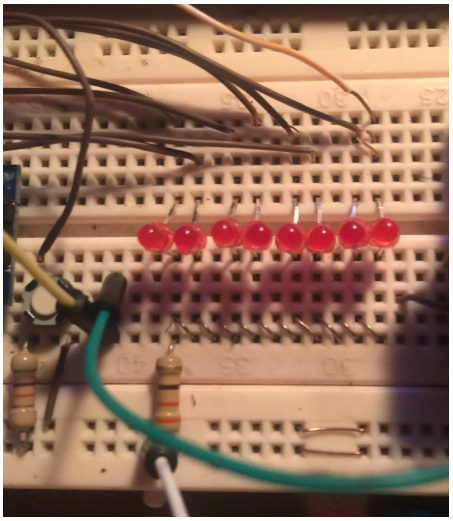


Figura 2. Circuito de la parte 1

Para resolver la secuencia, y el control de esta a través del pulsador, se utilizo un contador que se incrementa al ser

pulsado, por medio de una interrupción, y dependiendo del valor de este contador que va desde el 1 al 3, realiza la rutina de encendido de leds que corresponde.

El programa utilizado esta en el anexo I.

II-B. Parte 2, sensor(generador)+ADC+DAC

Construir un programa para introducir la señal análoga mediante el ADC y extraer la misma onda: en un puerto paralelo, en un PWM.

El circuito usado es el mismo, que en la anterior sección, pues se usara los leds para mostrar como se representan los datos obtenidos en la señal de entrada análoga, utilizando para ello 8 bits. Esto se realiza mediante software, cambiado un número de límites, es decir, cambiando el rango del número. El ADC obtiene números entre 0 y 1024. Para representarlos en binario se necesitan 10 bits, para usar solo 8 bits, los limites del número cambiar de 0 a 1023 a el nuevo rango dado por 8 bits, de 0 a 255. En el anexo 2 se puede ver el programa utilizado.

II-C. Parte 3 Aplicación, Generador de Ondas

Introducir en el puerto ADC0 un potenciometro, el valor de voltaje leído representara la frecuencia de nuestro generador de ondas.

1. Generar en el DAC una onda senoidal
2. Generar en el DAC una onda cuadrada
3. Generar en el DAC una onda diente de sierra

El pulsador seleccionara los casos.

El circuito utilizado para esta parte se muestra en la figura 3, utilizando los siguientes componentes: Un potenciometro de $10K\Omega$, un pulsador, dos resistencia de 330Ω, para la protección de corriente del pulsador y el potenciometro. Debido

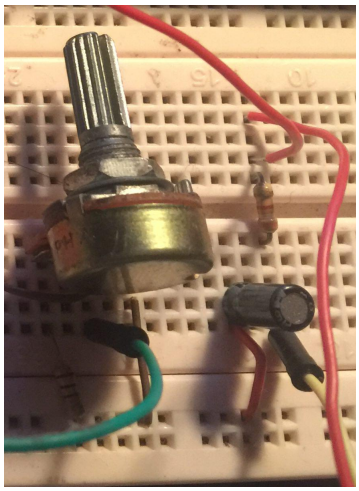


Figura 3. Circuito de la parte 3

a que los voltajes que entrega la salida digital que emula una salida analógica se componen del aporte individual de distintos valores a distintas frecuencias, es decir, tiene muchas componentes alternas que deben de ser filtradas. Para ello, se propone acoplar una fase de filtrado a la salida PWM, compuesta por un circuito RC, que es un filtro pasa bajos con una frecuencia de corte de 4Hz, para los valores de $3{,}9k\Omega$ y 10microfaradios de la resistencia y el condensador respectivamente.

Para que la salida PWM pueda representar una señal, se debe calcular los valores que debe tomar la onda, en un periodo de tiempo completo. La velocidad con que estos datos se vayan a ejecutar determinara la frecuencia de la onda. El PWM del Arduino UNO y NANO, dispone de 8 bits que se traduce en 256 niveles. Siendo este valor el máximo para muestrear una señal senoidal. Se decidió por conveniencia, calcular los valores de la onda senoidal cada 10 grados. Se obtiene un total de 36 muestras de una onda senoidal. Debido a que estos cálculos matemáticos no son apropiados para el controlador, por la resolución de bits del registro el redondeo puede inducir errores al calcular los valores a ser generados en el PWM, realizarlos constantemente no es una manera óptima, por lo cual se calculo en una computadora los 36 valores que corresponden a la onda senoidal. Lo que hace el Arduino es leer estos valores de una tabla cada determinado tiempo. De la misma forma se hace para las onda cuadrada y sierra, pero debido a que estos valores no necesitan un calculo matemático complejo, se pueden generar en el Arduino.

II-D. Optativo. Conexión con Matlab

Modificar y ampliar los programas anteriores en Arduino para que transmita datos mediante el port serial de los valores del ADC.

El circuito para esta etapa es la misma placa arduino, con tierra común al voltaje a ser medido. Este voltaje debe tener un valor máximo de 5 voltios, y ser positivo. Voltajes negativos pueden malograr la electrónica del Arduino. Para gráficar los valores en Matlab es necesario ejecutar un script en la pc, y a partir de esta ejecución, hace la recolección de datos y su gráfica en el tiempo que se estipule en el programa.

En el anexo 4 se muestran los programas tanto para Matlab como para el Arduino.

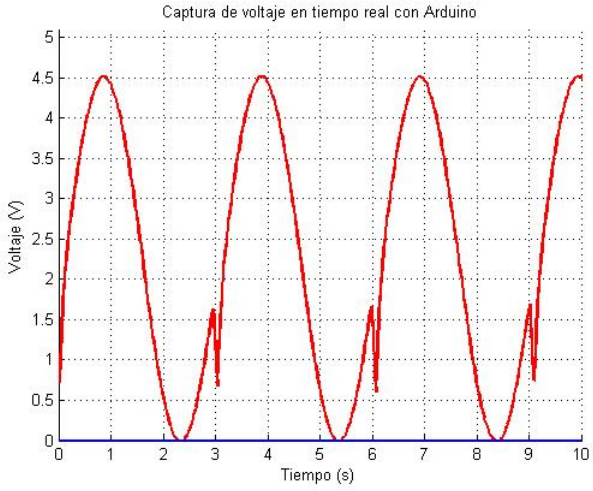
III. EXPERIMENTOS

Los experimentos que corresponden a la parte 1 y 2, en que se maneja el puerto paralelo del Arduino, muestran el funcionamiento esperado, luego de cargar el programa. Estos experimentos no precisan ningún tipo especial de instrumentos para poder apreciarlos.

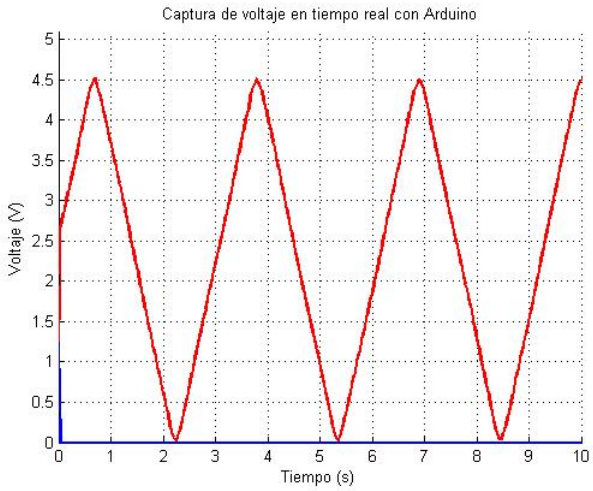
Para los experimentos de la parte 3, en el cual se usa una salida PWM para generar ondas; cuadrada, senoidal y diente sierra, se necesita un osciloscopio que nos permita visualizar la señal en función del tiempo. Esta herramienta llamada osciloscopio, es un instrumento electrónico que nos permite ver el voltaje en función del tiempo. Como parte de la aplicación, se fabrica un osciloscopio casero, utilizando para ello el ADC del Arduino, y el MatLab que gráfica los datos proveídos por el Arduino a través del puerto serial USB.

Para poder realizar las gráficas de los experimentos de la parte 3, usaremos un Arduino MEGA que haga la parte de la recolección de datos, cuyo programa se encuentra en el anexo 4, y el Arduino NANO que sera el generador de ondas. Ambos

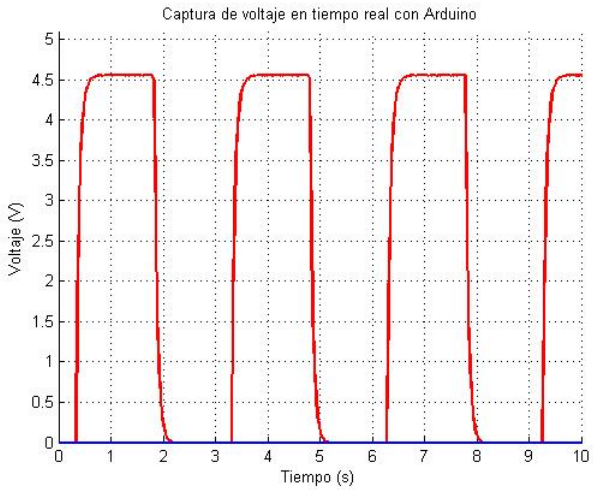
funcionan al mismo tiempo, y se puede apreciar en tiempo real a las figuras del generador de ondas.



(a) Onda senoidal

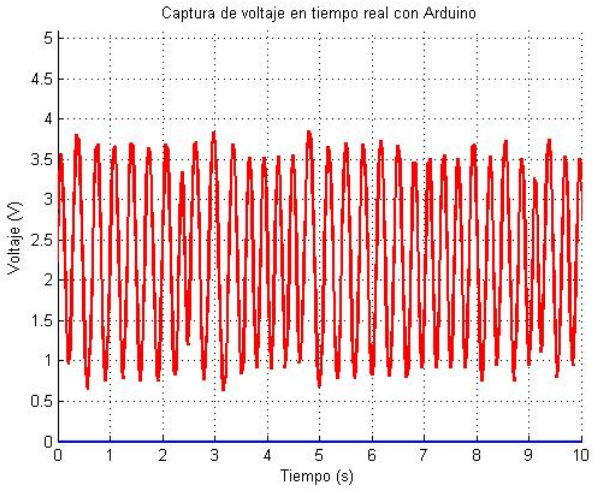


(b) Onda diente de sierra

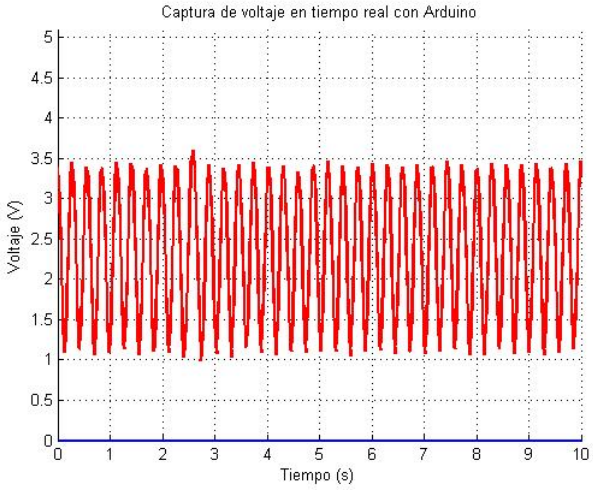


(c) Onda cuadrada

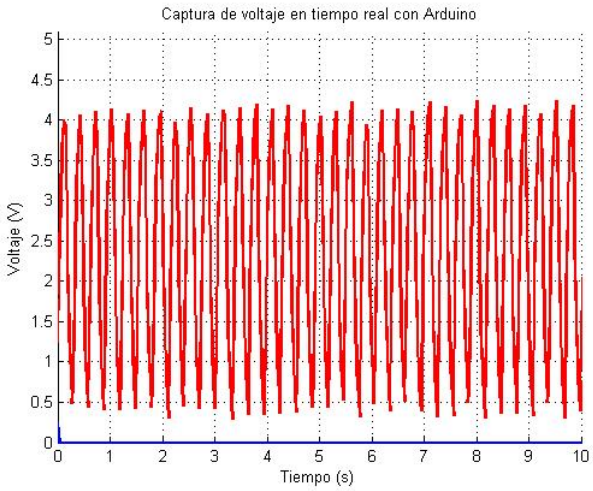
Figura 4. Generador de ondas con Arduino



(a) Onda senoidal



(b) Onda diente de sierra



(c) Onda cuadrada

Figura 5. Generador de ondas en frecuencias limite.

III-A. Generador de ondas

En la figura 4, se puede ver el conjunto de ondas, que se pueden seleccionar por un pulsador. Estas se encuentran a la

misma frecuencia para apreciar la salida y forma de cada una.

En la figura 5, se puede ver el conjunto de ondas, en una frecuencia donde empiezan a distorsionarse y variar el valor pico a pico. Para hallar esta frecuencia, se vario el valor de esta hasta que la señal alcanza a disminuir en 70 % su valor pico a pico.

IV. Reconstrucción de la señal

Para la figura 4, se calcula la frecuencia aplicada para generar la onda senoidal, cuadrada y diente de sierra utilizando el siguiente razonamiento.

La frecuencia es indirectamente proporcional al periodo. El periodo, que puede manipularse a través de un potenciometro, nos arroja datos desde los 5 a 2000 milisegundos, que es el tiempo que dura la señal en completar un periodo completo. Utilizando la relación $f = 1/T$. Para las ondas de la figura 4, el periodo calculado es de 2000ms, lo que representa una frecuencia de 0.5Hz.

Para las ondas de la figura 5, el periodo calculado es de 200ms, lo que representa una frecuencia de 5KHz.

V. Conclusiones

Se han montado circuitos y programado microcontroladores, bajo la plataforma Arduino, para demostrar el uso de los periféricos de salida, poniendo énfasis en el conversor análogo digital ADC, y el conversor digital análogo DAC; mediante el uso de un puerto paralelo y una salida PWM. Se describió la técnica que se utiliza para poder programar al Arduino como un generador de señales, de la misma manera que se mostró una aplicación al construir un osciloscopio. Se pudo verificar, que los microcontroladores no son perfectos en la generación de ondas, y son limitados en la adquisición veloz de datos para reconstruir la señal. Esto se debe a muchos factores que escapar al diseño teórico, por ejemplo, la suma de los tiempos de procesamiento del Arduino, el tiempo de transmisión se limita a la velocidad de la comunicación por puerto serial entre la PC y el Arduino mediante el USB. Al utilizar un filtro pasa bajo, que mejora la representación de la señal, también existen tiempos de cargado del condensador, que no fueron considerados y generan una señal en la práctica, ligeramente distinta a la señal teórica.

Referencias

[1] Documentación y hoja de datos de la fabricación del Arduino UNO y NANO. https://www.arduino.cc

ANEXO 1

```
int cnt=1;
int interruptor=1;
int led1=13;
int led2=12;
int led3=11;
int led4=10;
int led5=9;
int led6=8;
int led7=7;
int led8=6;



void setup() {

  attachInterrupt(interruptor, contador, RISING);
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
  pinMode(led4,OUTPUT);
  pinMode(led5,OUTPUT);
  pinMode(led6,OUTPUT);
  pinMode(led7,OUTPUT);
  pinMode(led8,OUTPUT);

}

void loop() {

switch (cnt) {
   case 1:
digitalWrite(led1, HIGH);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  digitalWrite(led5, LOW);
  digitalWrite(led6, LOW);
  digitalWrite(led7, LOW);
  digitalWrite(led8, LOW);
  delay(300);
  digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
  digitalWrite(led2, HIGH);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  digitalWrite(led5, LOW);
  digitalWrite(led6, LOW);
  digitalWrite(led7, LOW);
  digitalWrite(led8, LOW);
  delay(300);
  digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
```

```
digitalWrite(led2, LOW);
digitalWrite(led3, HIGH);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, HIGH);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, HIGH);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, HIGH);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, HIGH);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, HIGH);
```

```
delay(300);
  break;
  case 2:
  digitalWrite(led1, LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, HIGH);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, HIGH);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, HIGH);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, HIGH);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, HIGH);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, HIGH);
digitalWrite(led4, LOW);
```

```
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, HIGH);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
digitalWrite(led1,HIGH);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
  break;
  case 3:
    digitalWrite(led1, HIGH);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, HIGH);
digitalWrite(led3, HIGH);
digitalWrite(led4, HIGH);
digitalWrite(led5, LOW);
digitalWrite(led6, LOW);
digitalWrite(led7, LOW);
digitalWrite(led8, LOW);
delay(300);
// wait for a second
digitalWrite(led1, LOW);   // turn the LED on (HIGH is the voltage level)
digitalWrite(led2, LOW);
digitalWrite(led3, LOW);
digitalWrite(led4, LOW);
digitalWrite(led5, HIGH);
digitalWrite(led6, HIGH);
digitalWrite(led7, HIGH);
digitalWrite(led8, HIGH);
delay(300);

   break;
  default:
  cnt=1;  // default is optional
  break;
}
```

```
}

void contador() {
  cnt++;
  Serial.print("contador  ");
  Serial.println(cnt);

}
```

ANEXO 2

```
int cnt=1;
int led1=13;
int led2=12;
int led3=11;
int led4=10;
int led5=9;
int led6=8;
int led7=7;
int led8=6;

void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
    pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
  pinMode(led4,OUTPUT);
  pinMode(led5,OUTPUT);
  pinMode(led6,OUTPUT);
  pinMode(led7,OUTPUT);
  pinMode(led8,OUTPUT);

}

void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  int sensorValue1=constrain(sensorValue, 0, 255);
  delay(300);
  Serial.println(sensorValue1,BIN);
  for (int m=0; m<=7; m++) {
     digitalWrite(6+m, bitRead(sensorValue1, m));
      }
      delay(100);
}
```

ANEXO 3

```
int cnt=1;
int valor=0;
int seno[] = {127, 149, 170, 190, 208, 224, 236, 246, 252, 254, 252, 246, 236, 224, 208, 190,
170, 149, 127, 104, 83, 63, 45, 29, 17, 7, 1, 0, 1, 7, 17, 29, 45, 63, 83, 104};
int interruptor=1;
int freq=0;
void setup()
{
attachInterrupt(interruptor, contador, RISING);
 Serial.begin(9600);
}

void loop() {
  Serial.print("contador  ");
  Serial.println(cnt);
  freq=10*analogRead(A0);
  freq=constrain(freq,5,2000);
  Serial.print("freq  ");
  Serial.println(freq);
  // put your main code here, to run repeatedly:
switch (cnt) {
    case 1:  //senoidal
      for(valor=0;valor<=36;valor++)
        {
          analogWrite(9,seno[valor]);
          delay(freq/36); //1 a 500
        }
    break;

    case 2:   // cuadrada;
    analogWrite(9,255);
    delay(freq/2);
    analogWrite(9,0);
    delay(freq/2);
    break;

    case 3: //diente sierra
    for(valor=0;valor<=255;valor=valor+14)
    {
    analogWrite(9,valor);
    delay(freq/36);
    }
    for(valor=255;valor>=0;valor=valor-14)
    {
    analogWrite(9,valor);
    delay(freq/36);
    }
    break;

    default:
```

```
    cnt=1;  // default is optional
    break;
  }

}


void contador() {
  cnt++;
  Serial.print("contador  ");
  Serial.println(cnt);

}
```

ANEXO 4
PROGRAMA ARDUINO

```
int out1 = 0;
int out2 = 0;

void setup() {
  // inicializar puerto serie
  Serial.begin(9600);
}

void loop() {
  // leer pines
  out1 = analogRead(A0);
  out2 = analogRead(A1);
  // enviar
  Serial.print(out1);
  Serial.print(",");
  Serial.println(out2);
  // esperar
  delay(20);
}
```

PROGRAMA MATLAB

```
%borrar previos
delete(instrfind({'Port'},{'COM5'}));
%crear objeto serie
s = serial('COM5','BaudRate',9600,'Terminator','CR/LF');
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%abrir puerto
fopen(s);
% parámetros de medidas
tmax = 10; % tiempo de captura en s
rate = 33; % resultado experimental (comprobar)
% preparar la figura
f = figure('Name','Captura');
a = axes('XLim',[0 tmax],'YLim',[0 5.1]);
l1 = line(nan,nan,'Color','r','LineWidth',2);
l2 = line(nan,nan,'Color','b','LineWidth',2);

xlabel('Tiempo (s)')
ylabel('Voltaje (V)')
title('Captura de voltaje en tiempo real con Arduino')
grid on
hold on
% inicializar
v1 = zeros(1,tmax*rate);
v2 = zeros(1,tmax*rate);
i = 1;
t = 0;

% ejecutar bucle cronometrado
tic
while t<tmax
   t = toc;
   % leer del puerto serie
   a = fscanf(s,'%d,%d')';
   v1(i)=a(1)*5/1024;
   v2(i)=a(2)*5/1024;
   % dibujar en la figura
   x = linspace(0,i/rate,i);
   set(l1,'YData',v1(1:i),'XData',x);
   set(l2,'YData',v2(1:i),'XData',x);
   drawnow
   % seguir
   i = i+1;
end
% resultado del cronometro
clc;
fprintf('%g s de captura a %g cap/s \n',t,i/t);
%% Limpiar la escena del crimen
fclose(s);
delete(s);
clear s;
```