

ESCUELA POLITÉCNICA NACIONAL

INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROCESAMIENTO DIGITAL DE SEÑALES

ALUMNOS: Pablo Pulache Sarmiento

Alex Remache Chimbolema

SEMESTRE: 2020B

PARALELO: GR1

FECHA: 9 de diciembre de 2020

PROFESOR: DR. ROBIN ÁLVAREZ

TEMA: Señales Analógicas Básicas

SUBTEMA: Generación y adquisición de onda mediante hardware Parte 1

# METODOLOGÍA PARA GENERAR CUALQUIER TIPO DE ONDA

# 

# Por un lado, el Arduino-Due permite configurar la resolución (número de bits / muestra) deseada mediante la instrucción:

# *analogWriteResolution(n)* donde *n* puede ser 8, 10 o 12 bits

El Arduino DUE presenta 12 entradas análogas y 2 salidas analógicas, la resolución de las lecturas y de las salidas es de 12 bits, es decir 4096 valores posibles en la conversión tanto de digital a análogo como de análogo a digital. Por defecto la resolución de las lecturas esta determinada en 10 bits para tener compatibilidad con otras tarjetas. La resolución es posible cambiar con la instrucción analogReadResolution() y analogWriteResolution. .

# Por otro lado, para generar un valor de voltaje analógico por un pin DAC (DAC0 o DAC1), la instrucción es *analogWrite(DAC, Valor\_decimal)* donde el *Valor\_decimal* estaría en el rango (0 - 2n-1). El máximo valor decimal correspondería al máximo valor de voltaje que puede obtener el Arduino-Due (3.3V)

Las entradas y salidas tanto análogas como digitales trabajan con un voltaje de 3.3.[V], si se aplica un valor mayor a estos voltajes se pueden dañar los pines de microcontrolador. Cabe destacar que la instrucción analogReference es ignorada en el Arduino Due.

**DEBER:**

**Tratando de vencer las limitaciones del puerto de audio, especialmente en las frecuencias bajas (menores a 100 Hz), y ya que muchísimos experimentos emplean frecuencias en este rango, se emplea el Arduino tanto para generación como para adquisición y análisis.**

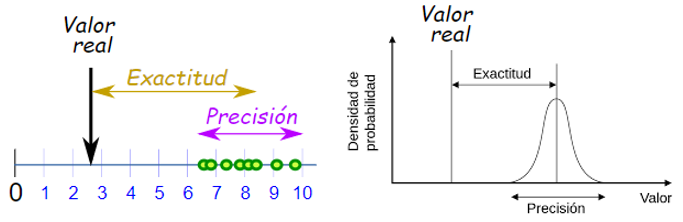
**PARTE 1:** **GENERADOR** **BÁSICO** CON ARDUINO Y **MEDICIÓN** CON AUDIO TESTER

Comparar los siguientes cuatro tipos de formas de generación para las ondas básicas (sinusoidal, triangular, cuadrada y diente de sierra):

1. Con funciones directas de Arduino, por ejemplo la instrucción ***sin***, si es que las hay.
2. Con librería externa del arduino.
3. Con metodología aprendida en este documento.
4. Proponga un nuevo método.

Compare **exactitud** (realice una tabla para cuatro frecuencias diferentes: 500Hz, 750 Hz, 1000 Hz y la máxima que usted consiga). No se olvide que, en la metodología aprendida, la Fmax conseguida dependía del (número de muestras / período) que se considere. Grabe un video corto con los resultados.

**OBSERVACIÓN**: La **exactitud**es la cercanía de una medida al **valor real**, mientras que la **precisión** es el grado de cercanía de los valores de **varias medidas** en un punto.



1. GENERACIÓN DE SEÑALES CON FUNCIONES PREDETERMINADAS:

En la ayuda de Arduino que se puede encontrar en la página <https://www.arduino.cc/reference/en/> se observa el lenguaje de referencia con el cual se manejan las distintas tarjetas. El lenguaje se divide en funciones, valores (variables y constantes) y estructuras.

Las funciones son un grupo de instrucciones para controlar la tarjeta y desarrollar cálculos a nivel computacional. En este grupo se encuentran funciones para las entradas y salidas tanto digitales como análogas. En este grupo se encuentran funciones para determinar la resolución de lecturas análogas, funciones para crear retardos de tiempo, funciones para configurar comunicación serial, funciones para configurar interrupciones, funciones matemáticas, funciones para operar números binarios y funciones trigonométricas. En este último grupo se encuentran las funciones sin(), cos() y tan() las cuales realizan estas operaciones a los números en su interior.

Cabe mencionar que Aduino no dispone de una manera directa de generar una señal analógica y que esta aparezca en los pines DAC0 y DAC1, de esta manera no se puede generar con una señal con una función predeterminada ya que para lograrlo se necesitan de parámetros como los bits de resolución y el tiempo entre muestras. Lo que realiza la función sin() es calcular el seno del ángulo en radianes, con lo cual se obtiene un resultado entre –1 y +1. No hay funciones como la cuadrada, triangular o diente de sierra, las cuales necesitarían parámetros como tiempos de subida y bajada para poder ser configuradas.

De esta manera para lograr que salga una función sinusoidal por el pin analógico DAC0 o DAC1 se necesita crear una base de tiempo, la cual se puede crear en Matlab de la siguiente manera:

|  |
| --- |
| clc;  clear all;  close all;  Fo=1; %frecuencia fundamental de la señal  To = 1 / Fo; % período de la señal  duracion = To; %duración: 1 PERIODO  N = 60; %número de muestras por periodo deseado  Ts= To / N; % tiempo entre muestras  tn=0:Ts:duracion; %vector de tiempo discreto |

En este código se genera un vector de tiempo llamado “tn” el cual va desde 0 hasta el valor de duración que para defecto de obtener cualquier frecuencia es de 1 [s]. El número de muestras que se propone es de 60 y con el vector generado en Matlab se procede a realizar el vector en Arduino para realizar el barrido de las muestras:

|  |
| --- |
| //tabla del vector de tiempo  float tn[]={0,0.0167,0.0333,0.0500,0.0667,0.0833,0.1000,0.1167,0.1333,0.1500,0.1667,0.1833,  0.2000,0.2167,0.2333,0.2500,0.2667,0.2833,0.3000,0.3167,0.3333,0.3500,0.3667,0.3833,  0.4000,0.4167,0.4333,0.4500,0.4667,0.4833,0.5000,0.5167,0.5333,0.5500,0.5667,0.5833,  0.6000,0.6167,0.6333,0.6500,0.6667,0.6833,0.7000,0.7167,0.7333,0.7500,0.7667,0.7833,  0.8000,0.8167,0.8333,0.8500,0.8667,0.8833,0.9000,0.9167,0.9333,0.9500,0.9667,0.9833,1}; |

Posteriormente se procede a crear las constantes necesarias para correr el código, como el índice del barrido, el tiempo entre muestras y la señal de salida:

|  |
| --- |
| nt i=0;//indice  int f\_senal=500;//frecuencia de la señal a generar  int Ts\_us=1000000/(f\_senal\*61);//tiempo entre muestras para generar una frecuencia determinada  double y;  int ys[61];//valor entero de la señal de salida  double c=4095;//constante |

Luego se realiza la declaración del PIN DAC0 como salida análoga, se configuran los bits de resolución del conversor digital análogo del microcontrolador y la operación para obtener los valores de salida mediante un lazo for:

|  |
| --- |
| void setup() {   pinMode(DAC0,OUTPUT);//DAC0 como pin de salida   analogWriteResolution(12);//bits de resolución   for(i=0;i<61;i++){//barrido de los valores de salida para cada valor de tiempo     y=0.5+(0.5\*sin(2\*PI\*tn[i]));     ys[i]=c\*y;   }  } |

Finalmente se realiza el barrido de cada valor de salida del vector ys[i] mediante un lazo for, de esta manera se pone cada valor en el pin análogo y se espera un tiempo hasta el próximo:

|  |
| --- |
| void loop() {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,ys[i]);     delayMicroseconds(29);   }  } |

El código completo queda de la siguiente manera:

|  |
| --- |
| //tabla del vector de tiempo  float tn[]={0,0.0167,0.0333,0.0500,0.0667,0.0833,0.1000,0.1167,0.1333,0.1500,0.1667,0.1833,  0.2000,0.2167,0.2333,0.2500,0.2667,0.2833,0.3000,0.3167,0.3333,0.3500,0.3667,0.3833,  0.4000,0.4167,0.4333,0.4500,0.4667,0.4833,0.5000,0.5167,0.5333,0.5500,0.5667,0.5833,  0.6000,0.6167,0.6333,0.6500,0.6667,0.6833,0.7000,0.7167,0.7333,0.7500,0.7667,0.7833,  0.8000,0.8167,0.8333,0.8500,0.8667,0.8833,0.9000,0.9167,0.9333,0.9500,0.9667,0.9833,1};  int i=0;//indice  int f\_senal=500;//frecuencia de la señal a generar  int Ts\_us=1000000/(f\_senal\*61);//tiempo entre muestras para generar una frecuencia determinada  double y;  int ys[61];//valor entero de la señal de salida  double c=4095;//constante  void setup() {   pinMode(DAC0,OUTPUT);//DAC0 como pin de salida   analogWriteResolution(12);//bits de resolución   for(i=0;i<61;i++){//barrido de los valores de salida para cada valor de tiempo     y=0.5+(0.5\*sin(2\*PI\*tn[i]));     ys[i]=c\*y;   }  }  void loop() {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,ys[i]);     delayMicroseconds(29);   }  } |

En este código se genera un vector de tiempo tn[] donde están todos los valores de tiempo, luego se genera el vector de tiempo de salida ys[61]. La fórmula de tiempo de retardo es dividir 1000000 para el producto entre la frecuencia que se requiere generar por el número de muestras, posterior a ello se genera el vector de salida ys[i], el cual es producto de la operación sin() de cada uno de los valores de tiempo y generando valores positivos al multiplicar y sumar 0.5 ya que la instrucción sin() genera valores entre -1 y 1. Finalmente en el lazo principal se realiza el barrido de la señal ys[i] , cada retardo de tiempo Ts\_us el cual se configura para una frecuencia de 500[Hz]. El resultado de generar este código se muestra en Audio Tester, para poder observar estos valores en el programa se necesita conectar el pin de salida análogo DAC0 al siguiente circuito:

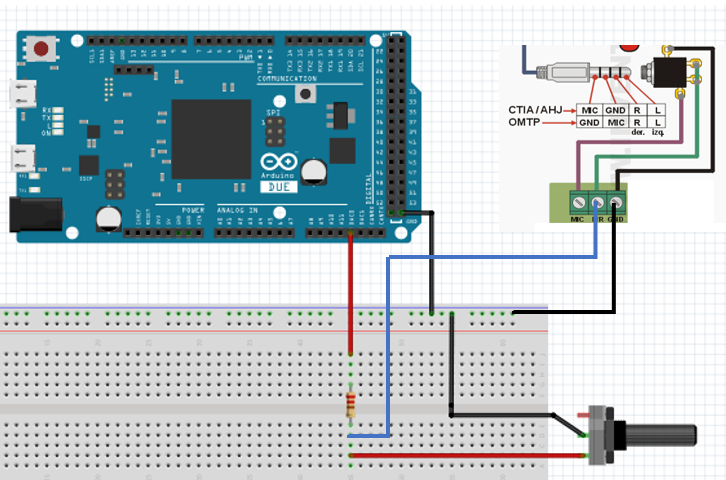


Figura 1. Conexión del circuito para observar la señal generada en Audio Tester.

El circuito presenta un divisor de voltaje donde se pone una resistencia de 10 [kΩ] y un potenciómetro para ajustar el nivel de voltaje de salida a un voltaje menor a 1[V], se elige un valor de 0.5[V], con esto el divisor de voltaje queda de la siguiente manera:

El valor de la resistencia se ajusta con el potenciómetro en 1.8[kΩ], se tiene que tener en cuenta que este potenciómetro no debe superar y valor mayor a 4.5[kΩ], ya que ese valor haría que el sonido salga mayor a 1[V]. Se puede observar el circuito implementado en la siguiente gráfica:

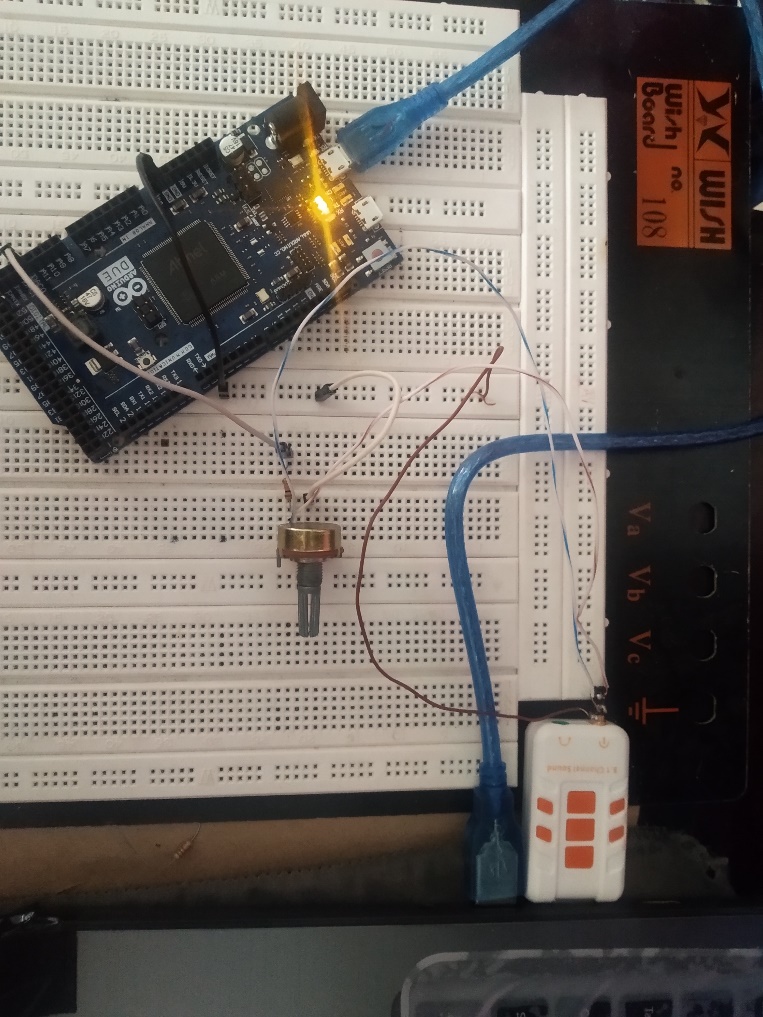


Figura 2. Implementación del circuito mediante la tarjeta de sonido para poder observar la señal de salida en Audio Tester.

Subiendo el código a la tarjeta Arduino se obtiene el siguiente resultado en Audio Tester:

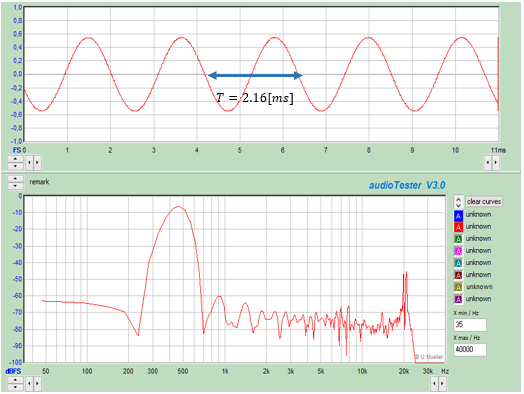


Figura 3. Función sinusoidal generada de 462.96 [Hz].

Se puede observar que el resultado del análisis armónico presenta a la onda en una frecuencia un poco menos a 500[Hz], esto se corrige mediante el cambio en el parámetro Ts\_us del código a un valor menor ya que las instrucciones del microcontrolador consumen un tiempo en la ejecución, si se cambia el valor de retardo a 29 [us] se consigue el siguiente resultado:

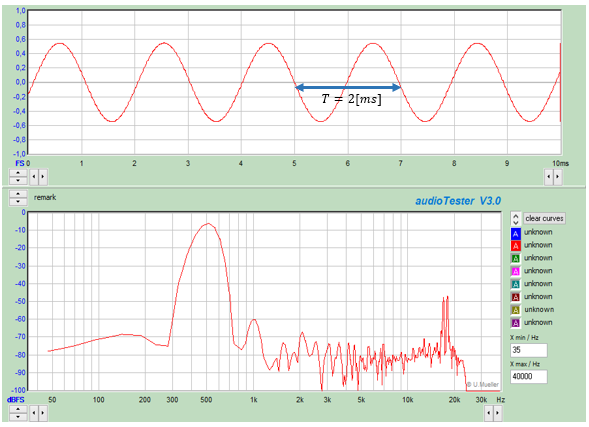


Figura 4. Resultado bajando el tiempo de retado a 29 [us], onda sinusoidal a 500 [Hz].

Como se puede observar se debe tener un factor de corrección ya que el valor calculado no hace que la señal llegue a su frecuencia determinada, por tanto, primero se observa el valor calculado que quedo de la siguiente manera:

Como el valor de Ts se lo declara como entero, el programa lo redondea en 32, por tanto el factor de corrección es:

Agregando este facto al código se tiene lo siguiente:

|  |
| --- |
| float factor=0.90625;  unsigned int Ts\_us=factor\*(1000000/(f\_senal\*61)); |

Se realizan las ondas para frecuencias de 750 [Hz], 1000[Hz] y la máxima que se puede conseguir, para esto se cambia el valor de f\_senal en el código:

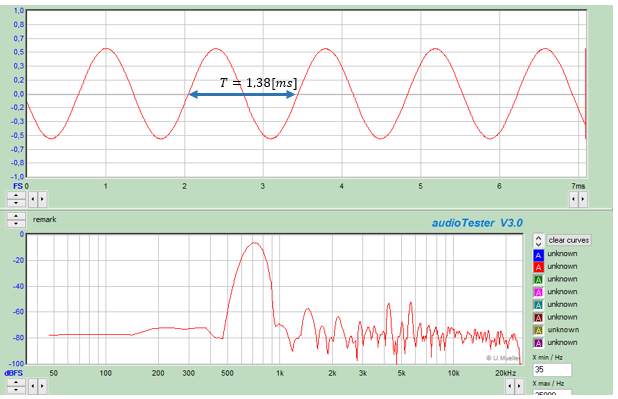


Figura 5. Señal sinusoidal a 724.63[Hz].

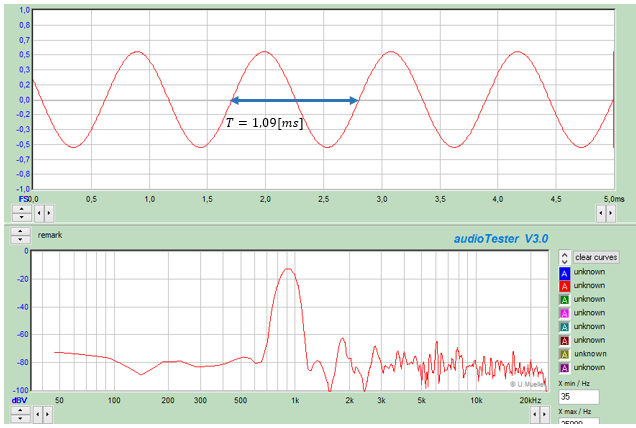


Figura 6. Señal sinusoidal a 917.43[Hz].

La máxima frecuencia se consigue al eliminar el tiempo de retardo:



Figura 7. La máxima frecuencia que se consigue es de 4166.66[Hz]

Las otras funciones como la diente de sierra, cuadrada o triangular NO se pueden obtener con alguna función predeterminada en ARDUINO de esta manera solo la señal sinusoidal se puede crear de esta manera. Para ver los resultados se obtiene una tabla de errores.

Tabla I. Tabla de errores para el primer método.

|  |  |  |
| --- | --- | --- |
| Tabla de error de la frecuencia | | |
| Frecuencia real [Hz] | **Frecuencia medida [Hz]** | **Error [%]** |
| 500 | 500 | 0 |
| 750 | 724.63 | 3.38 |
| 1000 | 917.43 | 8.257 |
| Frecuencia máxima obtenida | | **4166.66[Hz]** |

1. Con librería externa del ARDUINO:

Utilizando librerías externas de Arduino se debe buscar la librería por medio de internet o también se puede utilizar el Administrador de Bibliotecas disponible en el menú del IDE de Arduino:

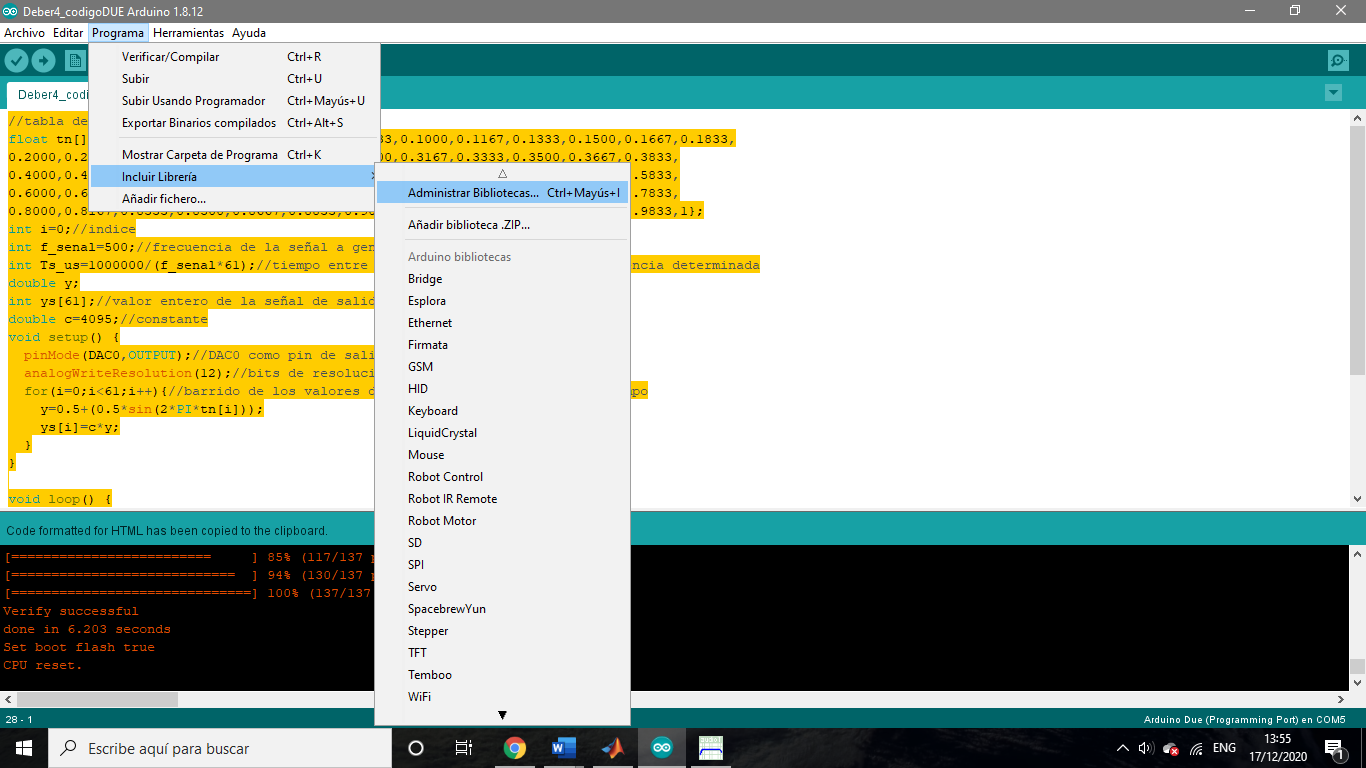


Figura 8. Seleccionar el administrador de bibliotecas para buscar librerías de Arduino.

Al abrir el administrador simplemente de pone en la busque algún título relacionado con lo que se desea, de esta manera para un generador de funciones se puede buscar de la siguiente manera:

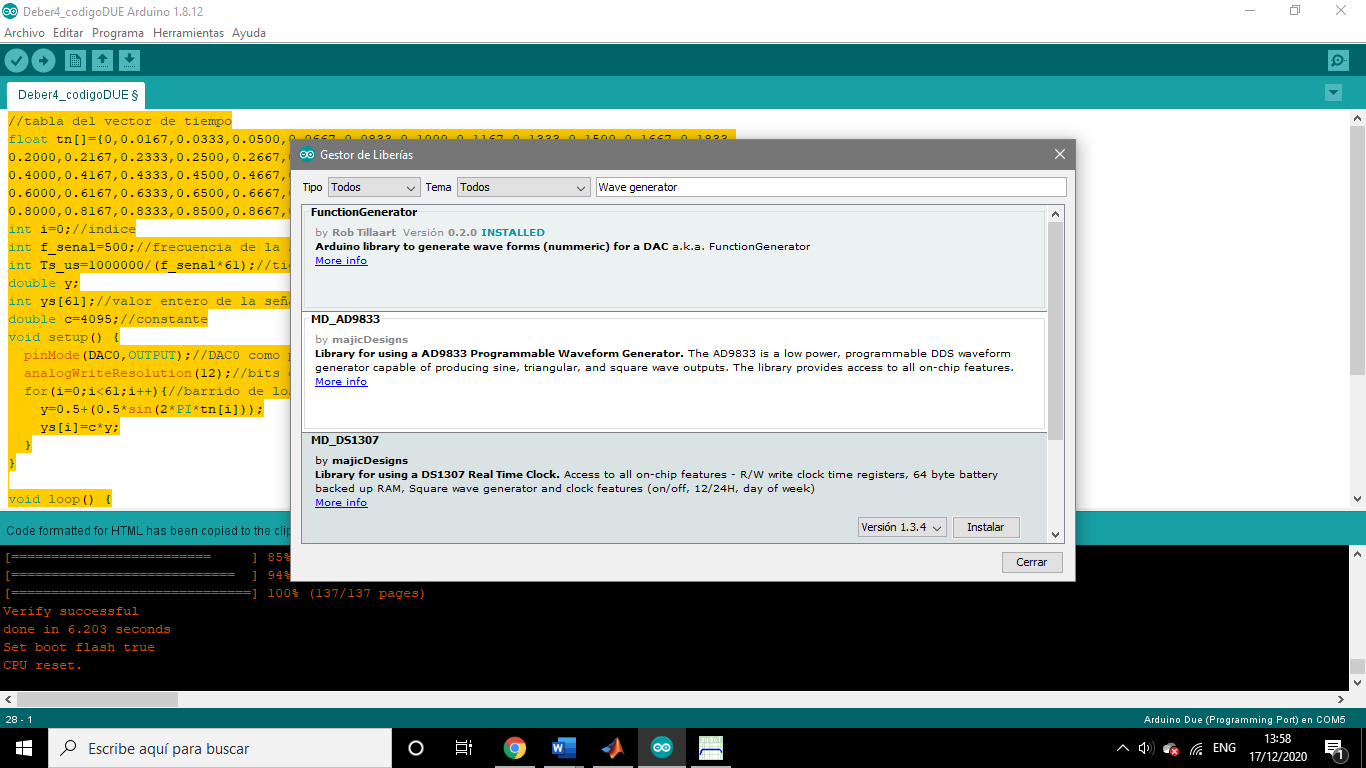


Figura 9. Gesto de librerías.

Al ingresar “Wave Generator” se encontró la librería “FunctionGenerator” del autor Rob Tilaart, si la librería no se encuentra instalada aparece un botón que automáticamente instala la librería para poder usarla. En la parte de “More Info” se direcciona al buscador para obtener las instrucciones que ayudan a poder usar la librería.

En la ayuda obtenida en <https://github.com/RobTillaart/FunctionGenerator> se tiene que la librería sirve para generar formas de onda numéricas para la una salida DAC.

Las formas de onda que se pueden obtener son las siguientes:

* float sawtooth(float t); Señal diente de sierra
* float triangle(float t); Señal triangular
* float square(float t); Señal cuadrada
* float sinus(float t); Señal sinusoidal
* float stair(float t, uint16\_t steps = 8); Señal escolada
* float random(); Señal aleatoria
* float line(); Señal lineal
* float zero(); Señal zero

Para la configuración de las ondas se tienen las siguientes funciones:

* void setPeriod(float period = 1.0); Establece el período de la señal
* void setFrequency(float freq = 1.0); Establece la frecuencia de la señal
* void setAmplitude(float ampl = 1.0); Establece la amplitud de la señal
* void setPhase(float phase = 0.0); Establece el Angulo de fase de la señal
* void setYShift(float yShift = 0.0); Establece la componente DC de la señal

Para poder usar la librería se debe ir a los ejemplos para ver como se usan las instrucciones y las diferentes y uno de los problemas con esta librería es que la base de tiempo con la que se generan las ondas es en base a la instrucción millis(), la cual retorna el número en milisegundos pasados desde que la tarjeta Arduino comienza a funcionar. El problema con utilizar esta base de tiempo es que se pretende que la señal sea en forma continua, de esta manera existirá problemas tanto en la frecuencia como en la amplitud para la generación de valores aptos para el conversor Digital-Analogo del Arduino DUE, pero la gran ventaja de esta librería es que se tienen las funciones para poder usarlas, de esta manera se procede de la misma manera que en el anterior literal.

El primer paso es crear un vector de tiempos en Matlab, con el siguiente código:

|  |
| --- |
| clc;  clear all;  close all;  Fo=1; %frecuencia fundamental de la señal  To = 1 / Fo; % período de la señal  duracion = To; %duración: 1 PERIODO  N = 60; %número de muestras por periodo deseado  Ts= To / N; % tiempo entre muestras  tn=0:Ts:duracion; %vector de tiempo discreto |

Entonces se determina un número de muestras de 60 y los valores de tiempo se generan cada Ts[s] que se determina mediante la división entre la frecuencia de 1[Hz] y el número de muestras.

El siguiente paso es pasar este vector generado al IDE de Arduino, con esto se obtiene lo siguiente:

|  |
| --- |
| float tn[]={0,0.0167,0.0333,0.0500,0.0667,0.0833,0.1000,0.1167,0.1333,0.1500,0.1667,0.1833,  0.2000,0.2167,0.2333,0.2500,0.2667,0.2833,0.3000,0.3167,0.3333,0.3500,0.3667,0.3833,  0.4000,0.4167,0.4333,0.4500,0.4667,0.4833,0.5000,0.5167,0.5333,0.5500,0.5667,0.5833,  0.6000,0.6167,0.6333,0.6500,0.6667,0.6833,0.7000,0.7167,0.7333,0.7500,0.7667,0.7833,  0.8000,0.8167,0.8333,0.8500,0.8667,0.8833,0.9000,0.9167,0.9333,0.9500,0.9667,0.9833,1}; |

Para usar la librería se usan la siguiente línea en la parte inicial del código:

|  |
| --- |
| #include "functionGenerator.h" |

Luego se crea una variable para control de los barridos de los vectores, también una variable objeto para la señal a generar y el vector que contendrá los valores de salida de la señal:

|  |
| --- |
| int i;//indice de los lazos for  **funcgen** gen1;//declaracion del objeto señal  float y[61];//señal de salida |

Luego se procede a establecer los valores de frecuencia, amplitud, ángulo de fase y componente DC de la señal a generar en el lazo de void setup(). En este caso se procede a generar una frecuencia de 1 y amplitud de 1 para poder usar la función sinusoidal de manera adecuada:

|  |
| --- |
| gen1.setFrequency(1);   gen1.setAmplitude(1);   gen1.setPhase(0);   gen1.setYShift(0); |

Luego, como en el anterior caso, se procede a declarar el pin de salida análogo, la resolución en bits del conversor Digital-Análogo, la creación de la señal de salida con la función sinus() de la librería y el se calcula el tiempo de retardo entre muestras para obtener la frecuencia de la señal que considera el anterior factor de corrección.

|  |
| --- |
| pinMode(DAC0,OUTPUT);//DAC0 como pin de salida  analogWriteResolution(12);//bits de resolución    for(i=0;i<61;i++){     y[i] = 4095\*(0.5+(0.5\*(gen1.sinus(tn[i])))); //señal de salida  }  int f\_senal=500;//frecuencia de la señal a generar  float factor=0.90625;  unsigned int Ts\_us=factor\*(1000000/(f\_senal\*61)); |

Una vez generado el vector de salida es cuestión de sacar cada valor mediante un lazo for para posteriormente esperar un tiempo entre muestras que determina la frecuencia de salida de la señal, este valor de retardo es Ts\_us:

|  |
| --- |
| void loop()  {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,y[i]);     delayMicroseconds(Ts\_us);   }  } |

El código completo queda de la siguiente manera:

|  |
| --- |
| #include "functionGenerator.h"  float tn[]={0,0.0167,0.0333,0.0500,0.0667,0.0833,0.1000,0.1167,0.1333,0.1500,0.1667,0.1833,  0.2000,0.2167,0.2333,0.2500,0.2667,0.2833,0.3000,0.3167,0.3333,0.3500,0.3667,0.3833,  0.4000,0.4167,0.4333,0.4500,0.4667,0.4833,0.5000,0.5167,0.5333,0.5500,0.5667,0.5833,  0.6000,0.6167,0.6333,0.6500,0.6667,0.6833,0.7000,0.7167,0.7333,0.7500,0.7667,0.7833,  0.8000,0.8167,0.8333,0.8500,0.8667,0.8833,0.9000,0.9167,0.9333,0.9500,0.9667,0.9833,1};  int i;//indice de los lazos for  **funcgen** gen1;//declaracion del objeto señal  float y[61];//señal de salida  int f\_senal=500;//frecuencia de la señal a generar  float factor=0.90625;  unsigned int Ts\_us=factor\*(1000000/(f\_senal\*61));  void setup()  {   gen1.setFrequency(1);   gen1.setAmplitude(1);   gen1.setPhase(0);   gen1.setYShift(0);   pinMode(DAC0,OUTPUT);//DAC0 como pin de salida   analogWriteResolution(12);//bits de resolución     for(i=0;i<61;i++){     y[i] = 4095\*(0.5+(0.5\*(gen1.sinus(tn[i]))));//señal de salida   }  }  void loop()  {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,y[i]);     delayMicroseconds(Ts\_us);   }  } |

El resultado en Audio Tester para frecuencias de 500 [Hz], 750 [Hz], 1000[Hz]:

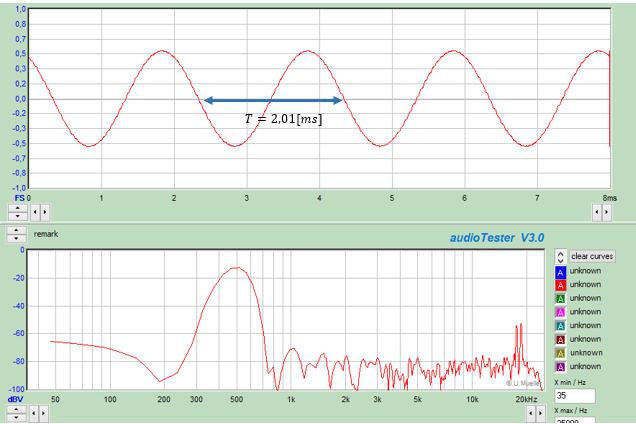


Figura 10. Señal generada a 497.51[Hz].

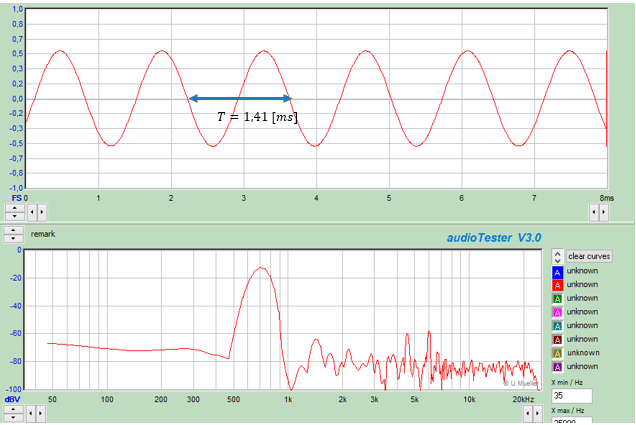


Figura 11. Señal generada a 709.21[Hz].

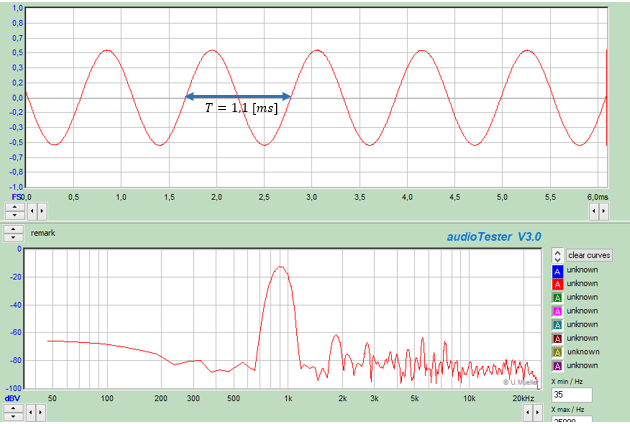


Figura 12. Señal generada a 909.09[Hz].

La máxima frecuencia alcanza con esta metodología es quitando el tiempo de espera del cógido para obtener la frecuencia más alta:

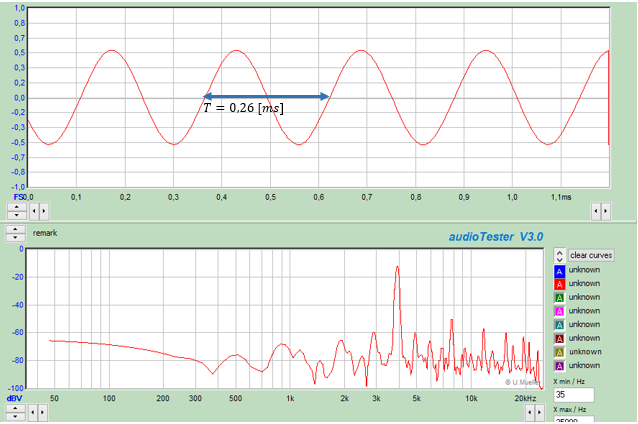


Figura 13. Frecuencia máxima obtenida de 3846.15[Hz].

Ahora, para generar una señal triangular, se cambia la función sinus() por la función triangle() y se tendría el código de la siguiente manera:

|  |
| --- |
| #include "functionGenerator.h"  float tn[]={0,0.0167,0.0333,0.0500,0.0667,0.0833,0.1000,0.1167,0.1333,0.1500,0.1667,0.1833,  0.2000,0.2167,0.2333,0.2500,0.2667,0.2833,0.3000,0.3167,0.3333,0.3500,0.3667,0.3833,  0.4000,0.4167,0.4333,0.4500,0.4667,0.4833,0.5000,0.5167,0.5333,0.5500,0.5667,0.5833,  0.6000,0.6167,0.6333,0.6500,0.6667,0.6833,0.7000,0.7167,0.7333,0.7500,0.7667,0.7833,  0.8000,0.8167,0.8333,0.8500,0.8667,0.8833,0.9000,0.9167,0.9333,0.9500,0.9667,0.9833,1};  int i;//indice de los lazos for  **funcgen** gen1;//declaracion del objeto señal  float y[61];//señal de salida  int f\_senal=1000;//frecuencia de la señal a generar  float factor=0.90625;  unsigned int Ts\_us=factor\*(1000000/(f\_senal\*61));    void setup()  {   gen1.setFrequency(1);   gen1.setAmplitude(1);   gen1.setPhase(0);   gen1.setYShift(0);   pinMode(DAC0,OUTPUT);//DAC0 como pin de salida   analogWriteResolution(12);//bits de resolución     for(i=0;i<61;i++){     y[i] = 4095\*(0.5+(0.5\*(gen1.triangle(tn[i]))));//señal de salida   }  }  void loop()  {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,y[i]);     delayMicroseconds(Ts\_us);   }  } |

El resultado en Audio Tester para frecuencias de 500 [Hz], 750 [Hz], 1000[Hz]:

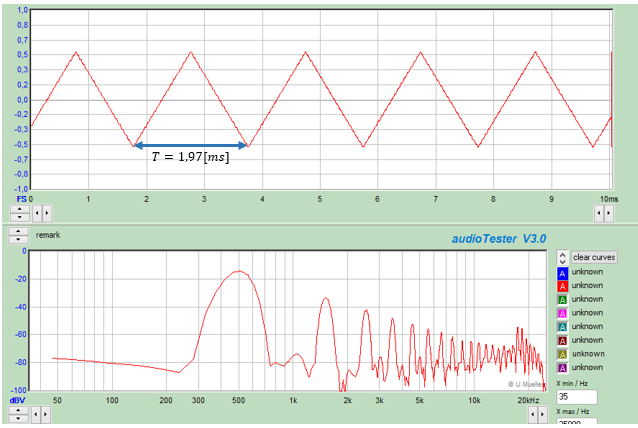


Figura 14. Señal triangular vista en Audio Tester a 507.61[Hz].

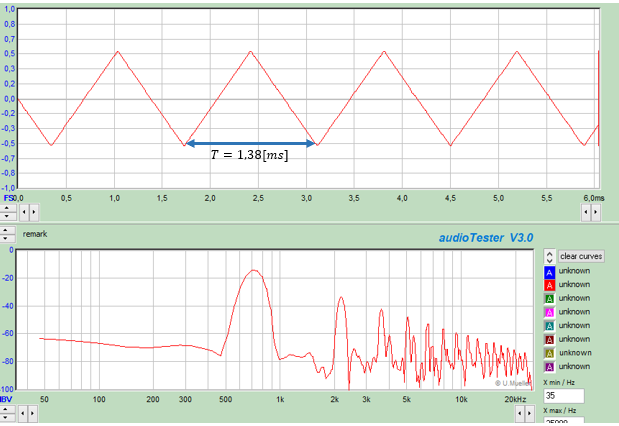


Figura 15. Señal obtenida de 724.63[Hz].

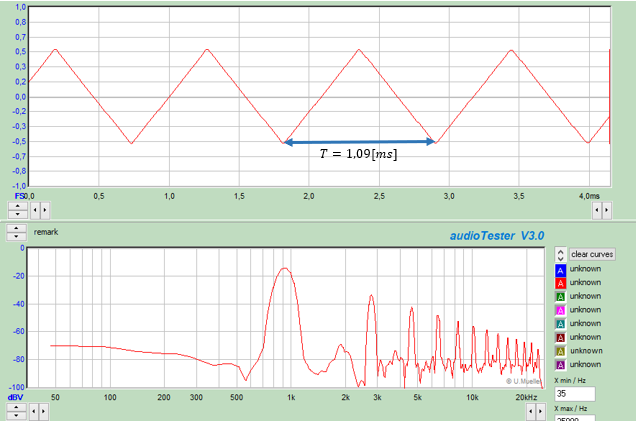


Figura 16. Señal obtenida a 917.43[Hz].

La mayor frecuencia que se puede obtener es quitando la instrucción de retardo:

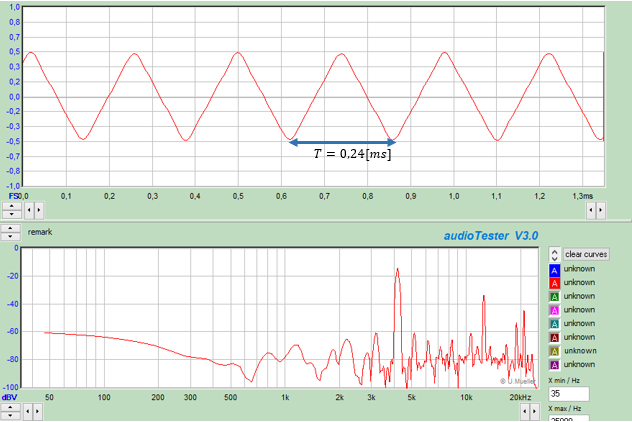


Figura 17. Frecuencia máxima obtenida de 4166.16[Hz].

Ahora, para generar una señal cuadrada, se cambia la función triangle() por la función square() y se tendría el código de la siguiente manera:

|  |
| --- |
| #include "functionGenerator.h"  float tn[]={0,0.0167,0.0333,0.0500,0.0667,0.0833,0.1000,0.1167,0.1333,0.1500,0.1667,0.1833,  0.2000,0.2167,0.2333,0.2500,0.2667,0.2833,0.3000,0.3167,0.3333,0.3500,0.3667,0.3833,  0.4000,0.4167,0.4333,0.4500,0.4667,0.4833,0.5000,0.5167,0.5333,0.5500,0.5667,0.5833,  0.6000,0.6167,0.6333,0.6500,0.6667,0.6833,0.7000,0.7167,0.7333,0.7500,0.7667,0.7833,  0.8000,0.8167,0.8333,0.8500,0.8667,0.8833,0.9000,0.9167,0.9333,0.9500,0.9667,0.9833,1};  int i;//indice de los lazos for  **funcgen** gen1;//declaracion del objeto señal  float y[61];//señal de salida  int f\_senal=500;//frecuencia de la señal a generar  float factor=0.90625;  unsigned int Ts\_us=factor\*(1000000/(f\_senal\*61));    void setup()  {   gen1.setFrequency(1);   gen1.setAmplitude(1);   gen1.setPhase(0);   gen1.setYShift(0);   pinMode(DAC0,OUTPUT);//DAC0 como pin de salida   analogWriteResolution(12);//bits de resolución     for(i=0;i<61;i++){     y[i] = 4095\*(0.5+(0.5\*(gen1.square(tn[i]))));//señal de salida   }  }  void loop()  {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,y[i]);     delayMicroseconds(Ts\_us);   }  } |

El resultado en Audio Tester para frecuencias de 500 [Hz], 750 [Hz] y 1000 [Hz]:

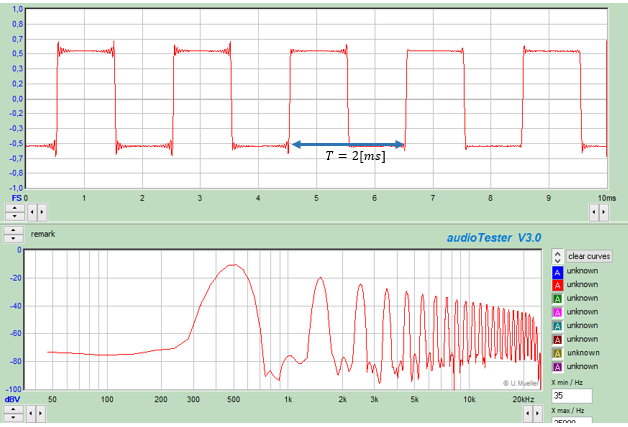


Figura 18. Señal cuadrada generada por medio de la librería a 500[Hz].



Figura 19. Señal obtenida a 704.22[Hz].

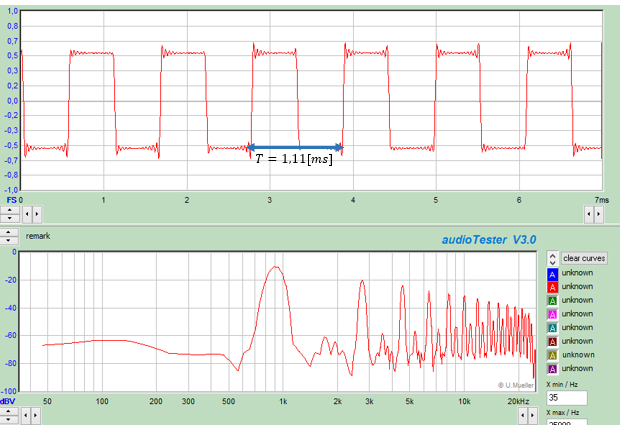


Figura 20. Señal obtenida de 900.09[Hz].

La máxima frecuencia obtenida se obtiene agregando un microsegundo de retardo, ya que sin ese tiempo la onda sale muy distorsionada:



Figura 21. Mayor frecuencia obtenida son la señal cuadrada a 3333.33[Hz].

Ahora, para generar una señal diente de sierra, se cambia la función square() por la función sawtooth() y se tendría el código de la siguiente manera:

|  |
| --- |
| #include "functionGenerator.h"  float tn[]={0,0.0167,0.0333,0.0500,0.0667,0.0833,0.1000,0.1167,0.1333,0.1500,0.1667,0.1833,  0.2000,0.2167,0.2333,0.2500,0.2667,0.2833,0.3000,0.3167,0.3333,0.3500,0.3667,0.3833,  0.4000,0.4167,0.4333,0.4500,0.4667,0.4833,0.5000,0.5167,0.5333,0.5500,0.5667,0.5833,  0.6000,0.6167,0.6333,0.6500,0.6667,0.6833,0.7000,0.7167,0.7333,0.7500,0.7667,0.7833,  0.8000,0.8167,0.8333,0.8500,0.8667,0.8833,0.9000,0.9167,0.9333,0.9500,0.9667,0.9833,1};  int i;//indice de los lazos for  **funcgen** gen1;//declaracion del objeto señal  float y[61];//señal de salida  int f\_senal=1000;//frecuencia de la señal a generar  float factor=0.90625;  unsigned int Ts\_us=factor\*(1000000/(f\_senal\*61));    void setup()  {   gen1.setFrequency(1);   gen1.setAmplitude(1);   gen1.setPhase(0);   gen1.setYShift(0);   pinMode(DAC0,OUTPUT);//DAC0 como pin de salida   analogWriteResolution(12);//bits de resolución     for(i=0;i<61;i++){     y[i] = 4095\*(0.5+(0.5\*(gen1.sawtooth(tn[i]))));//señal de salida   }  }  void loop()  {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,y[i]);     delayMicroseconds(Ts\_us);   }  } |

El resultado en Audio Tester para frecuencias de 500 [Hz], 750 [Hz] y 1000 [Hz]:

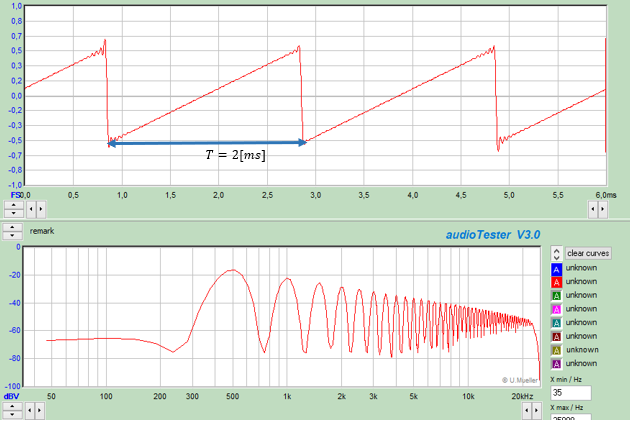


Figura 22. Señal diente de sierra a 500[Hz].

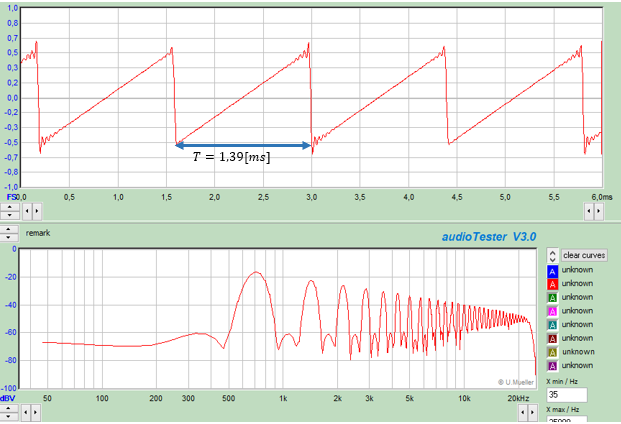


Figura 23. Señal diente de sierra a 719.42[Hz].

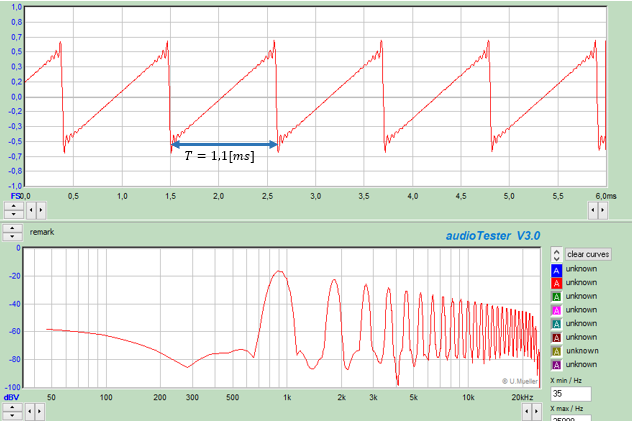


Figura 24. Señal diente de sierra a 909.09[Hz].

Para obtener la mayor frecuencia se retira el tiempo de retardo dando el siguiente resultado:

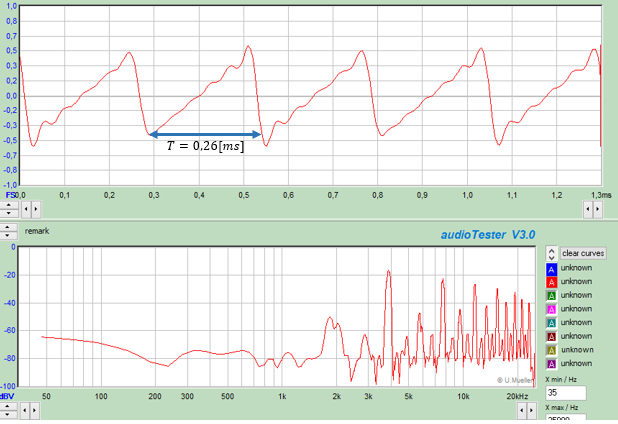


Figura 25. Señal diente de sierra generada a 3846.15[Hz].

A continuación, se realiza la tabla de errores en la medida de las frecuencias:

Tabla II. Tabla de errores para el segundo método.

|  |  |  |
| --- | --- | --- |
| Tabla de error de la frecuencia | | |
| Señal sinusoidal | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 497.51 | 0.498 |
| 750 | 729.21 | 2.772 |
| 1000 | 909.09 | 9.091 |
| Frecuencia máxima obtenida [Hz] | | **3846.15** |
| Señal triangular | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 507.61 | 1.52 |
| 750 | 724.63 | 3.38 |
| 1000 | 917.43 | 8.257 |
| Frecuencia máxima obtenida [Hz] | | **4166.66** |
| Señal Cuadrada | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 500 | 0 |
| 750 | 704.22 | 6.104 |
| 1000 | 909.09 | 9.091 |
| Frecuencia máxima obtenida [Hz] | | **3333.33** |
| Señal diente de sierra | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 500 | 0 |
| 750 | 719.42 | 4.077 |
| 1000 | 909.09 | 9.091 |
| Frecuencia máxima obtenida [Hz] | | **3846.15** |

1. Con metodología aprendida en este documento.

Con la metodología aprendida en el documento se necesita de la ayuda de Matlab oara generar los valores discretos con los cuales se trabajará para enviar al conversor análogo digital del Arduino Due.

El primer paso es generar esos valores formando un vector de tiempos y luego aplicando las funciones ya conocidas de las ondas anteriormente, como estas funciones generan funciones entre -1 y 1 se debe corregir a valores positivos para poder manejar el conversor del microcontrolador.

Para generar la señal sinusoidal se tiene que generar la base de tiempos en base a una frecuencia de 1[Hz], para luego construir las muestras en un periodo de 1[s], a estos tiempo se aplica la función sinusoidal y se le añade una componente en DC para tener valores positivos, luego se multiplica por la resolución de 12 bits, es decir 4095 y se tienen los valores discretos de la función a generar. Cabe mencionar que el tiempo entre muestras depende del número de muestras que para este caso es 60 de la siguiente manera:

|  |
| --- |
| clc;  clear all;  close all;  Fo=1; %frecuencia fundamental de la señal  To = 1 / Fo; % período de la señal  duracion = To; %duración: 1 PERIODO  N = 60; %número de muestras por periodo deseado  Ts= To / N; % tiempo entre muestras  tn=0:Ts:duracion; %vector de tiempo discreto  yn=0.5+0.5\*sin(2\*pi\*Fo\*tn); %señal deseada: SOLO POSITIVA Y AMPLITUD = 1  n = 12; % Resolución  yn1=yn\*(2^n-1);% valores decimales entre 0 y (2^n -1).  yn2=round(yn1); %redondear a valores enteros  plot(tn,yn2,'o') %gráfico de comprobación  title('Muestras positivas en un período de la señal') |

La señal en base a sus muestras queda de la siguiente manera:



Figura 26. Muestras de la señal sinusoidal.

Como Matlab hace la mayoría del trabajo, el código en Matlab es más sencillo con lo cual solo se copian los valores obtenidos en el Workspace de Matlab y se ingresan en un vector en el IDE de Arduino.

|  |
| --- |
| int seno[]={2048,2262,2473,2680,2880,3071,3251,3418,3569,3704,  3821,3918,3995,4050,4084,4095,4084,4050,3995,3918,  3821,3704,3569,3418,3251,3071,2880,2680,2473,2262,  2048,1833,1622,1415,1215,1024,844,677,526,391,274,  177,100,45,11,0,11,45,100,177,274,391,526,677,844,  1024,1215,1415,1622,1833,2048}; |

Luego se crean las variables para el control del barrido y para la frecuencia en base a la fórmula y códigos anteriores:

|  |
| --- |
| int i;//variabel de control de barrido  int f\_Senal=500;//frecuencia de la señal  int Ts\_us=1000000/(f\_Senal\*61);//tiempo entre muestras |

Luego se establece al pin analógico como salida y los bits de resolución:

|  |
| --- |
| void setup() {   pinMode(DAC0,OUTPUT);//pin de salida   analogWriteResolution(12);//bits de resolución  } |

Finalmente, de barren los valores del vector mediante un lazo for que se repite indefinidamente en el lazo principal del programa:

|  |
| --- |
| void loop() {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,seno[i]);//salida del valor i     delayMicroseconds(Ts\_us);//retardo entre muestras   }  } |

De esta manera el código completo queda de la siguiente manera:

|  |
| --- |
| int seno[]={2048,2262,2473,2680,2880,3071,3251,3418,3569,3704,  3821,3918,3995,4050,4084,4095,4084,4050,3995,3918,  3821,3704,3569,3418,3251,3071,2880,2680,2473,2262,  2048,1833,1622,1415,1215,1024,844,677,526,391,274,  177,100,45,11,0,11,45,100,177,274,391,526,677,844,  1024,1215,1415,1622,1833,2048};  int i;//variabel de control de barrido  int f\_Senal=500;//frecuencia de la señal  int Ts\_us=1000000/(f\_Senal\*61);//tiempo entre muestras  void setup() {   pinMode(DAC0,OUTPUT);//pin de salida   analogWriteResolution(12);//bits de resolucion  }  void loop() {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,seno[i]);//salida del valor i     delayMicroseconds(Ts\_us);//retardo entre muestras   }  } |

El resultado observado en Audio Tester es el siguiente para una frecuencia determinada en el código de 500[Hz]:

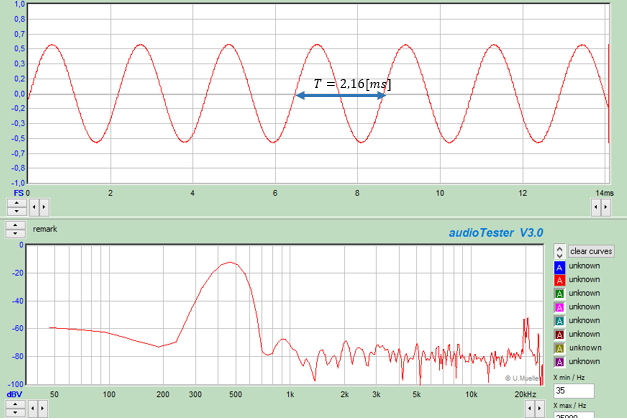


Figura 27. Señal sinusoidal generada a 462.96[Hz].

Como no se lograron los 500[Hz] el tiempo de retardo en el lazo es muy grande por lo que se debe disminuir multiplicando al valor calculado por un factor de corrección que se obtiene de la siguiente manera:

Este valor se ingresa en el código de la siguiente manera:

|  |
| --- |
| int f\_Senal=500;//frecuencia de la señal  float factor=0.9259262;  int Ts\_us=factor\*(1000000/(f\_Senal\*61));//tiempo entre muestras |

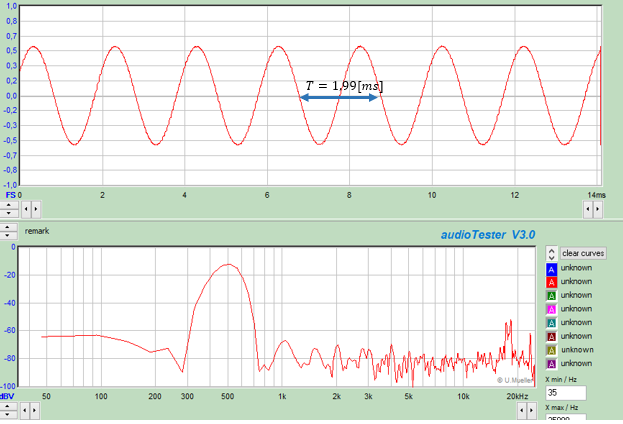


Figura 28. Señal obtenida a 502.51[Hz].

Manteniendo este factor se generan señales a 750 y 1000 [Hz].

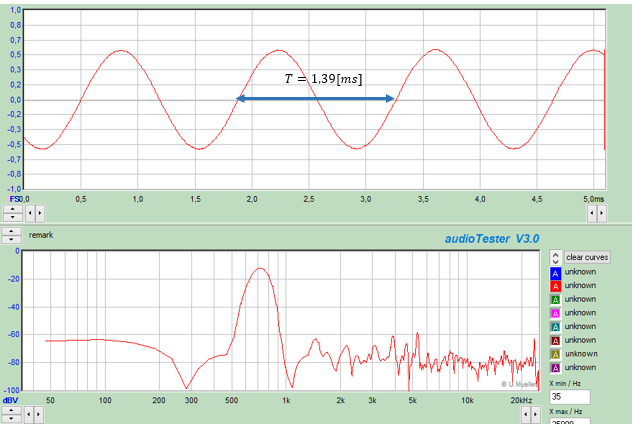


Figura 29. Señal obtenida a 719.42 [Hz].

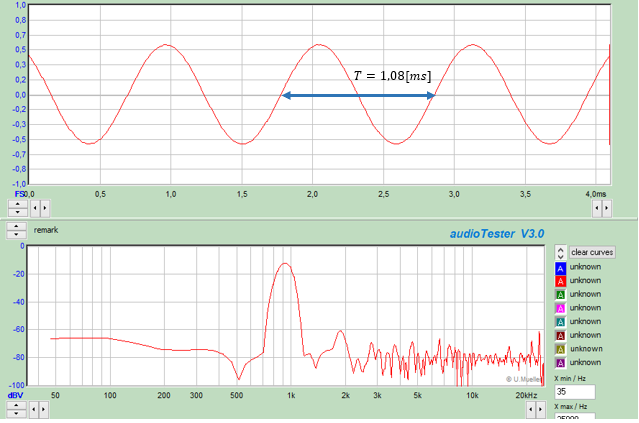


Figura 30. Señal obtenida a 925.92 [Hz].

Para obtener la mayor frecuencia se quita el retardo de tiempo y el resultado es el siguiente:

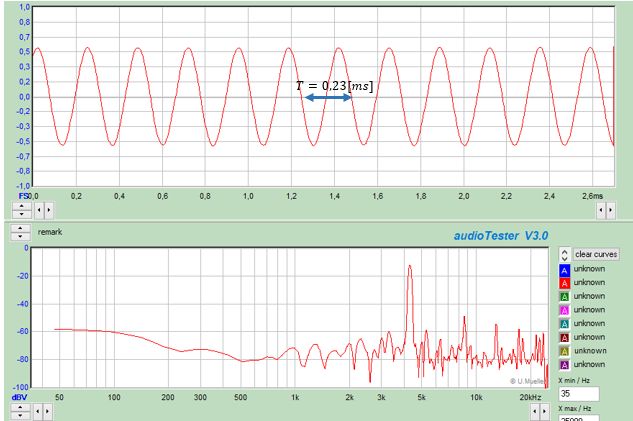


Figura 31. Frecuencia de mayor valor obtenida de 4347.83 [Hz].

Para la señal triangular se realiza lo mismo, pero cambiando la función sin por sawtooth al 50% en Matlab de la siguiente manera:

|  |
| --- |
| clc;  clear all;  close all;  Fo=1; %frecuencia fundamental de la señal  To = 1 / Fo; % período de la señal  duracion = To; %duración: 1 PERIODO  N = 60; %número de muestras por periodo deseado  Ts= To / N; % tiempo entre muestras  tn=0:Ts:duracion; %vector de tiempo discreto  yn=0.5+0.5\*sawtooth(2\*pi\*Fo\*tn,50/100); %señal deseada: SOLO POSITIVA Y AMPLITUD = 1  n = 12; % Resolución  yn1=yn\*(2^n-1);% valores decimales entre 0 y (2^n -1).  yn2=round(yn1); %redondear a valores enteros  plot(tn,yn2,'o') %gráfico de comprobación  title('Muestras positivas en un período de la señal') |

Las muestras que se obtiene son las siguientes:



Figura 32. Muestras de la señal triangular.

El código en IDE de Arduino con las muestras queda de la siguiente manera:

|  |
| --- |
| int tri[]={0,136,273,409,546,682,819,956,1092,1229,  1365,1502,1638,1775,1911,2048,2184,2321,2457,2594,  2730,2867,3003,3139,3276,3413,3549,3686,3822,3959,  4095,3959,3822,3685,3549,3413,3276,3139,3003,2867,  2730,2594,2457,2321,2184,2048,1911,1774,1638,1502,  1365,1229,1092,956,819,683,546,409,273,137,0};  int i;//variabel de control de barrido  int f\_Senal=500;//frecuencia de la señal  float factor=0.9259262;  int Ts\_us=factor\*(1000000/(f\_Senal\*61));//tiempo entre muestras  void setup() {   pinMode(DAC0,OUTPUT);//pin de salida   analogWriteResolution(12);//bits de resolucion  }  void loop() {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,tri[i]);//salida del valor i     delayMicroseconds(Ts\_us);//retardo entre muestras   }  } |

Los resultados para 500 [Hz], 750 [Hz] y 1000 [Hz] son los siguientes:

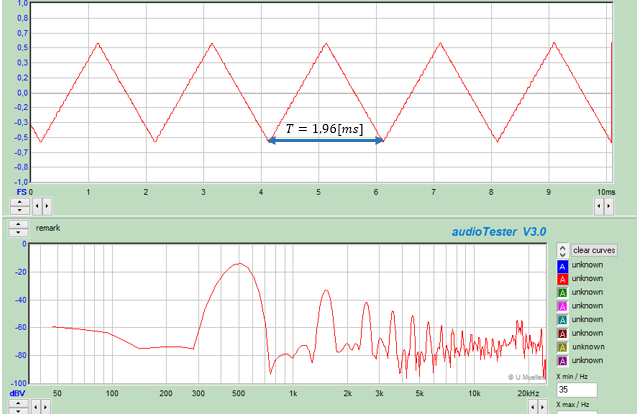


Figura 33. Onda generada a 510.2 [Hz].

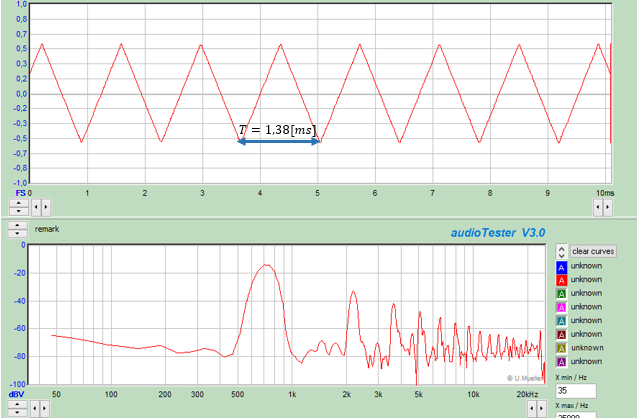


Figura 34. Onda generada a 724.64 [Hz].

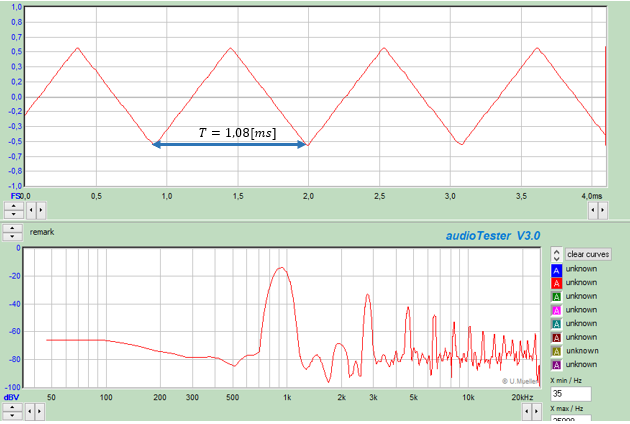


Figura 35. Señal obtenida a 925.92 [Hz].

Al quitar el tiempo de retardo se obtiene la máxima frecuencia de:

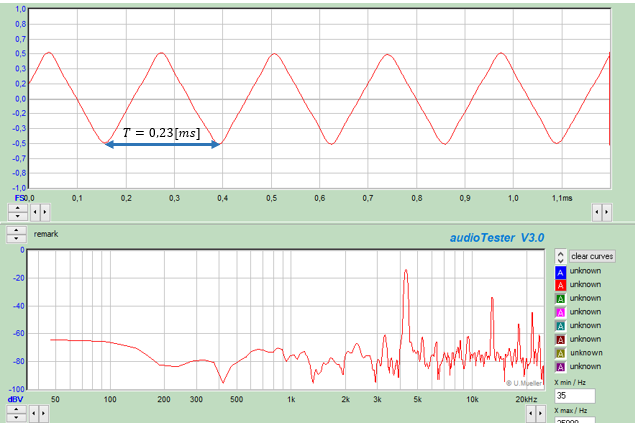


Figura 36. Máxima frecuencia obtenida de la onda a 4347.83 [Hz].

Para la señal cuadrada se cambia la función sawtooth por la función square al 50%:

|  |
| --- |
| clc;  clear all;  close all;  Fo=1; %frecuencia fundamental de la señal  To = 1 / Fo; % período de la señal  duracion = To; %duración: 1 PERIODO  N = 60; %número de muestras por periodo deseado  Ts= To / N; % tiempo entre muestras  tn=0:Ts:duracion; %vector de tiempo discreto  yn=0.5+0.5\*square(2\*pi\*Fo\*tn,50); %señal deseada: SOLO POSITIVA Y AMPLITUD = 1  n = 12; % Resolución  yn1=yn\*(2^n-1);% valores decimales entre 0 y (2^n -1).  yn2=round(yn1); %redondear a valores enteros  plot(tn,yn2,'o') %gráfico de comprobación  title('Muestras positivas en un período de la señal') |

La señal muestreada es la siguiente:



Figura 37. Muestras obtenidas de la señal cuadrada.

El código en IDE implementando las muestras es el siguiente:

|  |
| --- |
| int cua[]={4095,4095,4095,4095,4095,4095,4095,4095,4095,4095,  4095,4095,4095,4095,4095,4095,4095,4095,4095,4095,  4095,4095,4095,4095,4095,4095,4095,4095,4095,4095,  0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,4095};  int i;//variabel de control de barrido  int f\_Senal=1000;//frecuencia de la señal  float factor=0.9259262;  int Ts\_us=factor\*(1000000/(f\_Senal\*61));//tiempo entre muestras  void setup() {   pinMode(DAC0,OUTPUT);//pin de salida   analogWriteResolution(12);//bits de resolucion  }  void loop() {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,cua[i]);//salida del valor i     //delayMicroseconds(10);//retardo entre muestras   }  } |

Los resultados obtenidos para 500 [Hz], 750 [Hz] y 1000 [Hz]:

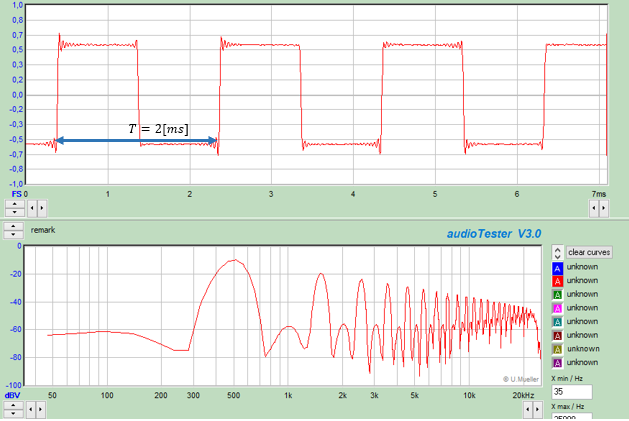


Figura 38. Señal obtenida a 500 [Hz].

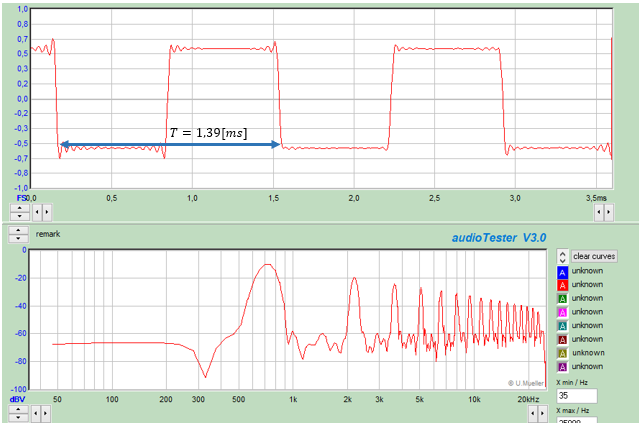


Figura 39. Señal obtenida a 719.42 [Hz].

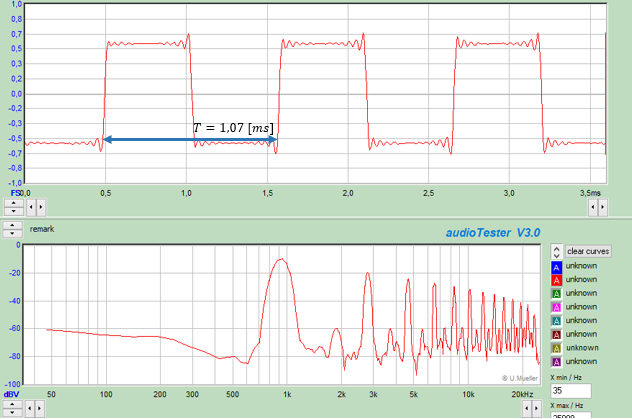


Figura 40. Señal obtenida a 934.58 [Hz].

La máxima frecuencia obtenida sin la instrucción de retardo es:

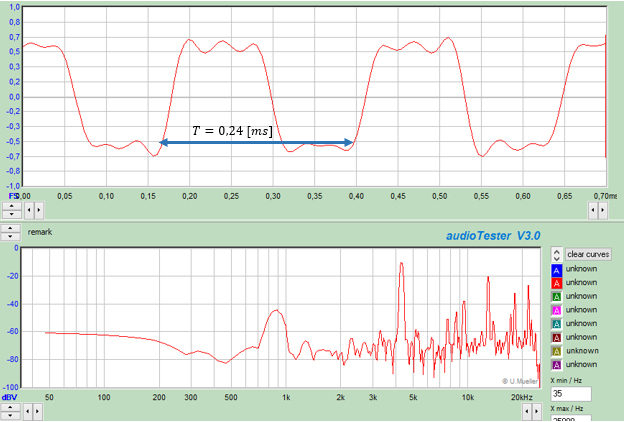


Figura 41. Señal obtenida a 4166.667 [Hz].

Para la generación de la señal diente de sierra se utiliza la función sawtooth al 100%, de esta manera en Matlab se obtiene lo siguiente:

|  |
| --- |
| clc;  clear all;  close all;  Fo=1; %frecuencia fundamental de la señal  To = 1 / Fo; % período de la señal  duracion = To; %duración: 1 PERIODO  N = 60; %número de muestras por periodo deseado  Ts= To / N; % tiempo entre muestras  tn=0:Ts:duracion; %vector de tiempo discreto  yn=0.5+0.5\*sawtooth(2\*pi\*Fo\*tn,1); %señal deseada: SOLO POSITIVA Y AMPLITUD = 1  n = 12; % Resolución  yn1=yn\*(2^n-1);% valores decimales entre 0 y (2^n -1).  yn2=round(yn1); %redondear a valores enteros  plot(tn,yn2,'o') %gráfico de comprobación  title('Muestras positivas en un período de la señal') |

De esta manera las muestras quedan de la siguiente manera:



Figura 42. Muestras de la señal diente de sierra obtenidas en Matlab.

El código en el IDE de Arduino queda de la siguiente manera con las muestras implementadas:

|  |
| --- |
| int ds[]={0,68,136,205,273,341,409,478,546,614,  682,751,819,887,956,1024,1092,1160,1229,1297,  1365,1433,1502,1570,1638,1706,1775,1843,1911,1979,  2048,2116,2184,2252,2321,2389,2457,2525,2594,2662,  2730,2798,2867,2935,3003,3071,3139,3208,3276,3344,  3413,3481,3549,3617,3686,3754,3822,3890,3959,4027,0};  int i;//variabel de control de barrido  int f\_Senal=1000;//frecuencia de la señal  float factor=0.9259262;  int Ts\_us=factor\*(1000000/(f\_Senal\*61));//tiempo entre muestras  void setup() {   pinMode(DAC0,OUTPUT);//pin de salida   analogWriteResolution(12);//bits de resolucion  }  void loop() {   i=0;   for(i=0;i<60;i++){     analogWrite(DAC0,ds[i]);//salida del valor i     delayMicroseconds(Ts\_us);//retardo entre muestras   }  } |

Para las frecuencias de 500 [Hz], 750 [Hz] y 1000 [Hz] el resultado es el siguiente:

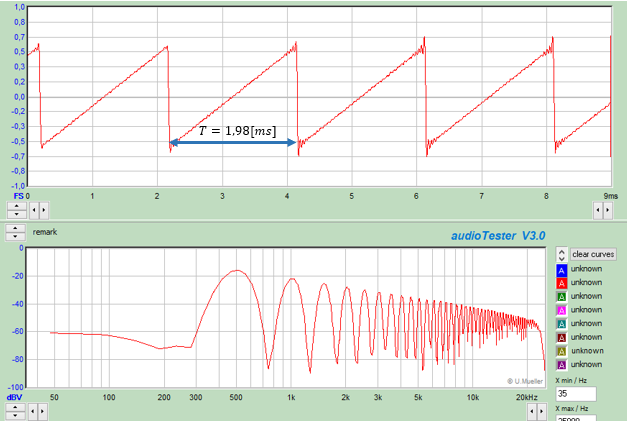


Figura 43. Señal obtenida a 505.05 [Hz].

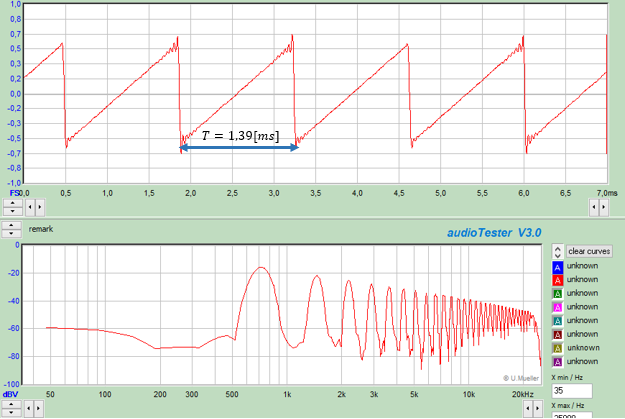


Figura 44. Señal obtenida a 719.42 [Hz].

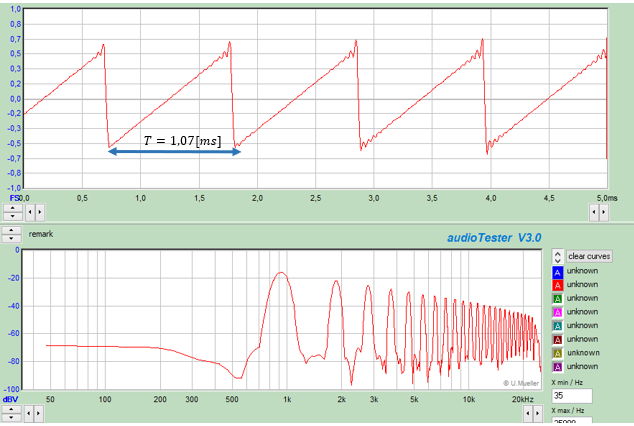


Figura 45. Señal obtenida a 934.58 [Hz].

La máxima frecuencia obtenida sin la instrucción de retardo es:

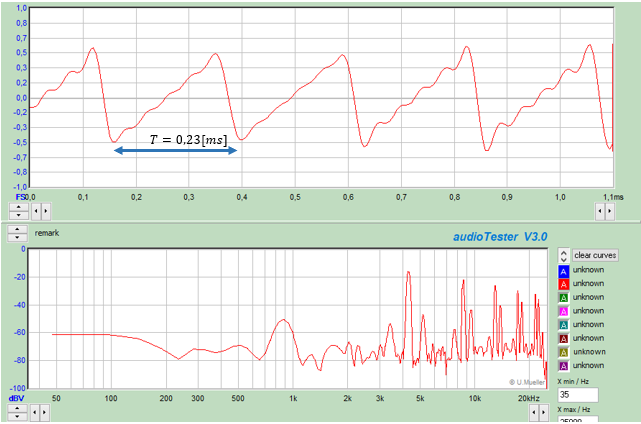


Figura 46. Señal obtenida a 4347.83 [Hz].

A continuación, se realiza la tabla de errores en la medida de las frecuencias:

Tabla III Tabla de errores para el tercer método.

|  |  |  |
| --- | --- | --- |
| Tabla de error de la frecuencia | | |
| Señal sinusoidal | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 502.51 | 0.502 |
| 750 | 719.42 | 4.07 |
| 1000 | 925.92 | 7.41 |
| Frecuencia máxima obtenida [Hz] | | **4347.83** |
| Señal triangular | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 510.2 | 2.04 |
| 750 | 724.64 | 3.38 |
| 1000 | 925.92 | 7.41 |
| Frecuencia máxima obtenida [Hz] | | **4347.83** |
| Señal Cuadrada | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 500 | 0 |
| 750 | 719.42 | 4.07 |
| 1000 | 934.58 | 6.54 |
| Frecuencia máxima obtenida [Hz] | | **4166.67** |
| Señal diente de sierra | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 505.05 | 1.01 |
| 750 | 719.42 | 4.07 |
| 1000 | 934.58 | 6.54 |
| Frecuencia máxima obtenida [Hz] | | **4347.83** |

1. Proponga un nuevo método.

Una forma sencilla de realizar esto mismo es utilizar la librería Arduino que ayuda a crear estas ondas, esta librería se llama waveforms.h. En este caso es necesario copiar el código directamente al IDE. La forma de selección de onda es en base a un índice que va de 0 a 3. El índice 0 representa a una señal seno, el índice 1 a una señal triangular, el índice 2 a una señal cuadrada y el índice 3 a una señal diente de sierra. Aquí ya no se genera ninguna base de tiempos y solo se barre la señal como se indica a continuación:

|  |
| --- |
| #include <Waveforms.h>//libreria  #define oneHzSample 1000000/maxSamplesNum  // muestras  int i = 0;//indice  volatile int wave0 = 0;//indice que indica la onda a generar  //0: sinusoidal;1:triangular;2:cuadrada;3:diente de sierra  void setup() {   analogWriteResolution(12);  // resolucion de 12 bits   pinMode(DAC0,OUTPUT);//pin de salida  }  void loop() {   for(i=0;i<maxSamplesNum;i++){     analogWrite(DAC0, waveformsTable[wave0][i]);  // se seleccina la onda de salida     delayMicroseconds(12);  // se setea la frecuencia   }  } |

El problema del uso de esta técnica es que el uso de la librería hace que se desconozca cuanto de demora la función “waveformTable()” por lo que el tiempo que se pone en el retado es cuestión de probar hasta que la frecuencia deseada salga, de esta manera los resultados para 500 [Hz], 750 [Hz] y 1000 [Hz] son los siguientes:

Para la señal sinusoidal se tienen los siguientes resultados:

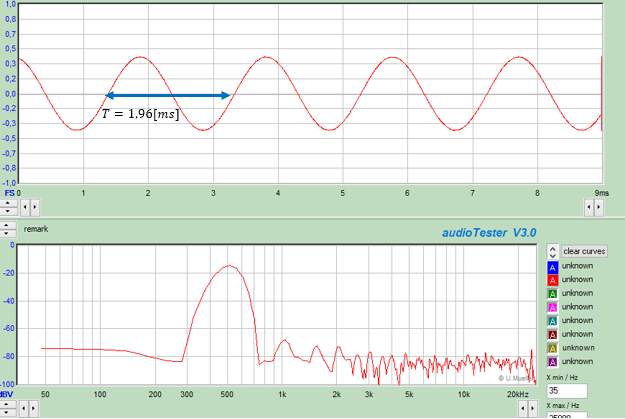


Figura 47. Señal obtenida a 510.20 [Hz].

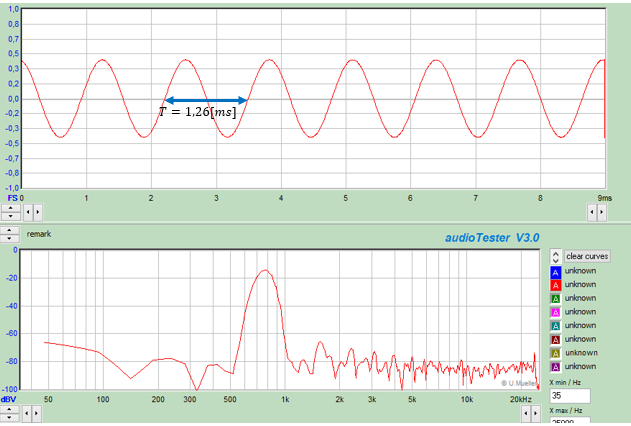


Figura 48. Señal obtenida a 793.65 [Hz].

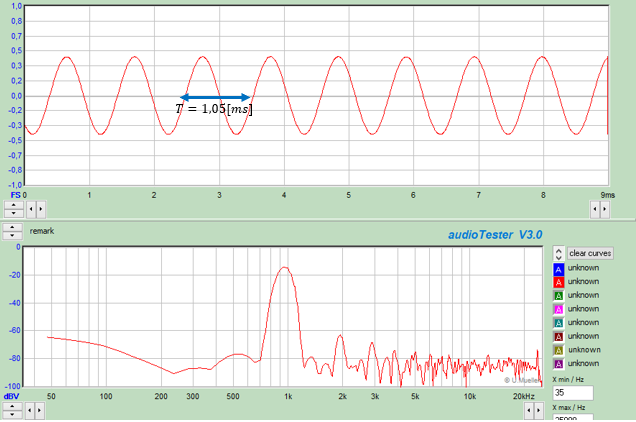


Figura 49. Señal obtenida a 952.38 [Hz].

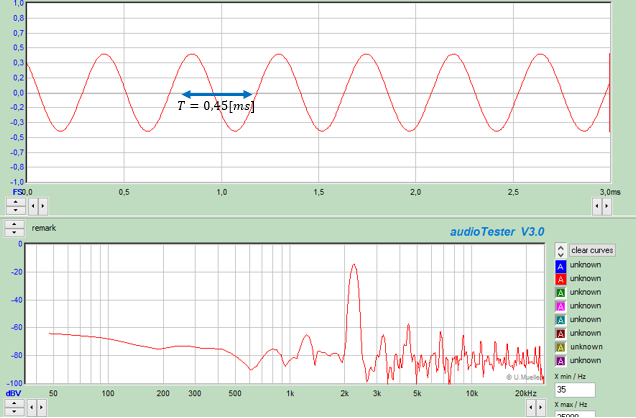


Figura 50. Señal obtenida a máxima frecuencia de 2222.22 [Hz].

Para la Señal triangular se tienen los siguientes resultados:

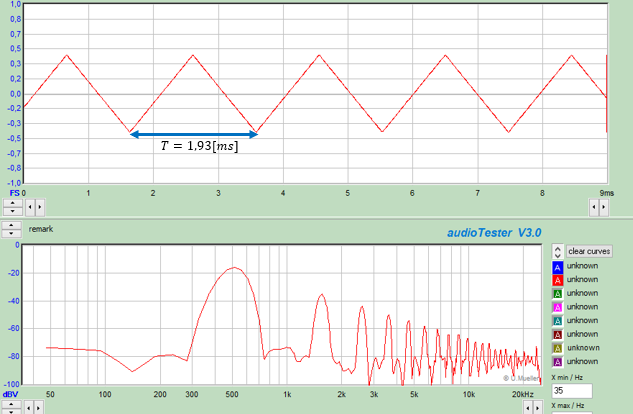


Figura 51. Señal obtenida a 518.13 [Hz].

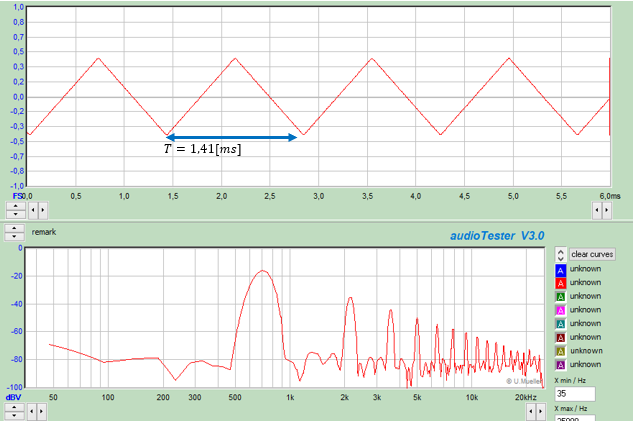


Figura 52. Señal obtenida a 709.22 [Hz].

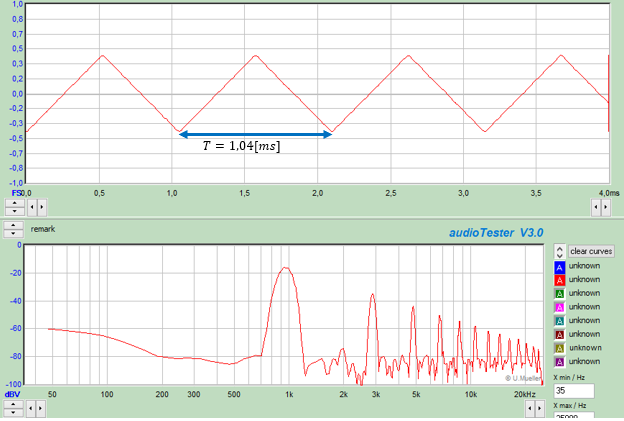


Figura 53. Señal obtenida a 961.54 [Hz].

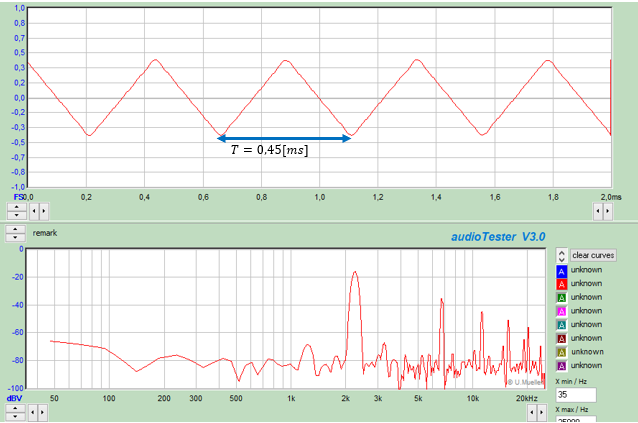


Figura 54. Señal obtenida a máxima frecuencia de 2222.22 [Hz].

Para la Señal diente de sierra se tienen los siguientes resultados:

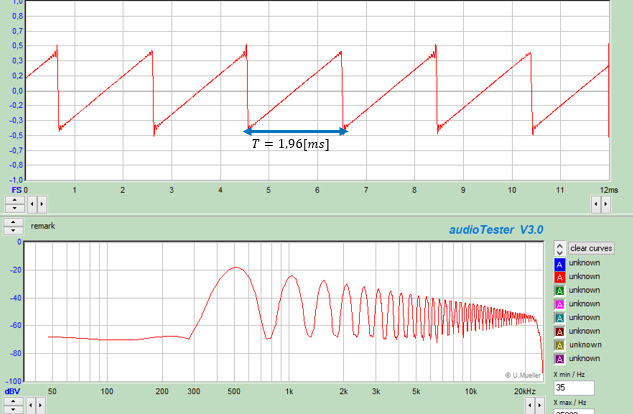


Figura 55. Señal obtenida a 510.20 [Hz].



Figura 56. Señal obtenida a 714.28 [Hz].

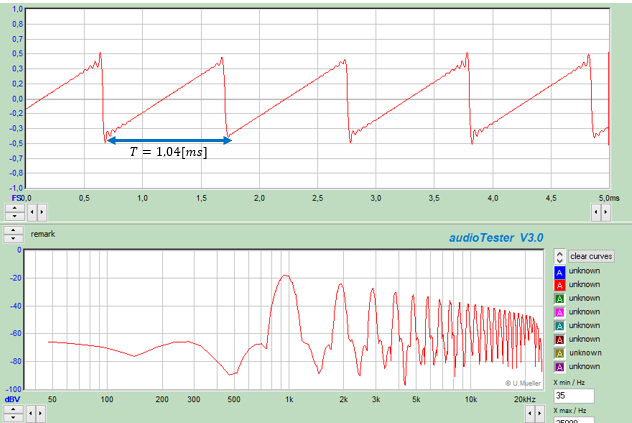


Figura 57. Señal obtenida a 961.54 [Hz].

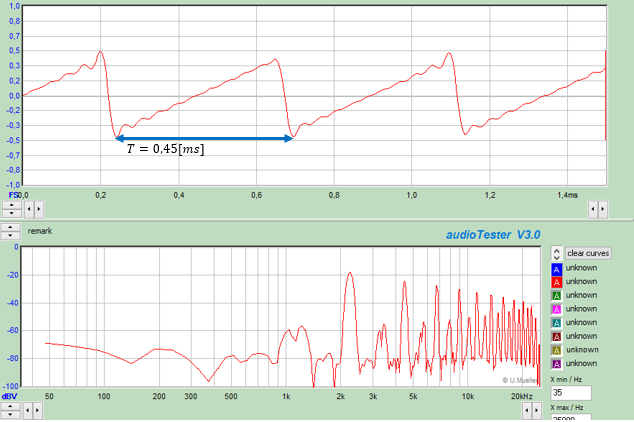


Figura 58. Señal obtenida a máxima frecuencia de 2222.22 [Hz].

Para la Señal diente de sierra se tienen los siguientes resultados:

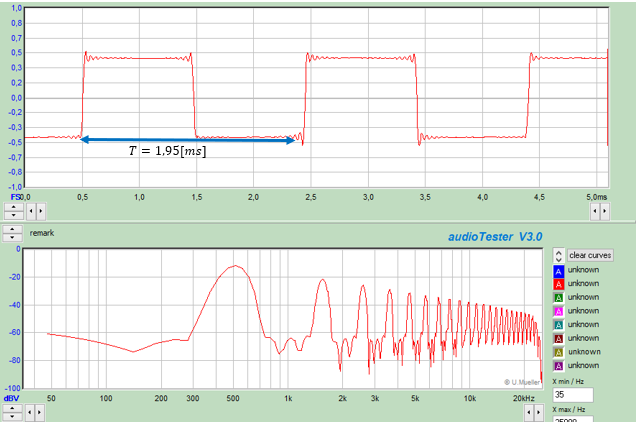


Figura 59. Señal obtenida a 512.82 [Hz].

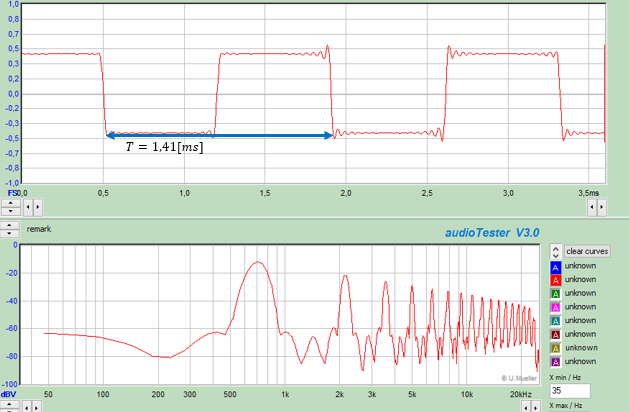


Figura 60. Señal obtenida a 709.22 [Hz].

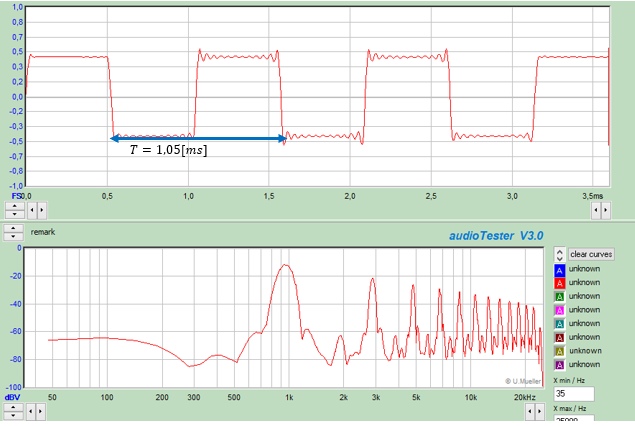


Figura 61. Señal obtenida a 952.38 [Hz].

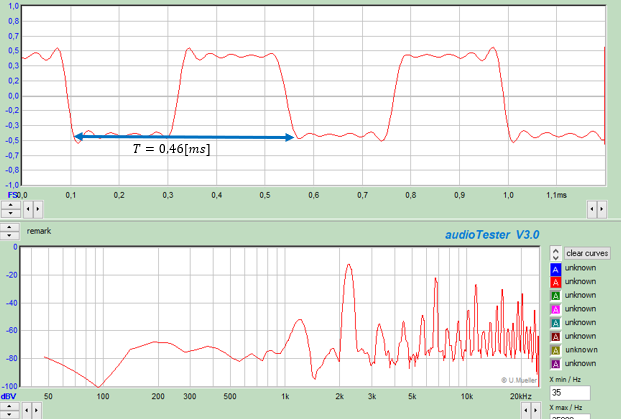


Figura 62. Señal obtenida a máxima frecuencia de 2173.91 [Hz].

A continuación, se realiza la tabla de errores en la medida de las frecuencias:

Tabla IV. Tabla de errores del cuarto método.

|  |  |  |
| --- | --- | --- |
| Tabla de error de la frecuencia | | |
| Señal sinusoidal | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 510.20 | 2.04 |
| 750 | 793.65 | 5.82 |
| 1000 | 952.38 | 4.76 |
| Frecuencia máxima obtenida [Hz] | | **2222.22** |
| Señal triangular | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 518.13 | 3.63 |
| 750 | 709.22 | 5.44 |
| 1000 | 961.54 | 3.85g |
| Frecuencia máxima obtenida [Hz] | | **2222.22** |
| Señal Cuadrada | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 510.2 | 2.04 |
| 750 | 714.28 | 4.76 |
| 1000 | 961.54 | 3.85 |
| Frecuencia máxima obtenida [Hz] | | **2222.22** |
| Señal diente de sierra | | |
| Frecuencia Real [Hz] | Frecuencia Medida [Hz] | Error [%] |
| 500 | 512.82 | 2.564 |
| 750 | 709.22 | 5.44 |
| 1000 | 952.38 | 4.76 |
| Frecuencia máxima obtenida [Hz] | | **2173.91** |

ANÁLISIS:

Observando los resultados de las tablas se puede llegar a la conclusión de que los tres primeros métodos son bastante aceptables en cuanto a versatilidad en la fijación de la frecuencia de la onda, en especial el método aprendido en clase ya que en lazo de control principal del Arduino solo se barren los valores de la tabla ya determinada en Matlab, esto conlleva a que se puede formar cualquier tipo de onda por lo cual es más fácil que lo otros métodos. En cuanto a último método se puede observar que no se tiene tanto control sobre la frecuencia de la señal y en los resultados se puede observar que depende del tiempo de retardo que esta en valores pequeños para alcanzar altas frecuencias, esto porque se usa una librería que es llamada y se demora mucho tiempo, por lo cual no se tiene tanta exactitud en la frecuencia que se requiera generar. También se pudo observar que el factor de corrección se tendería que hacer para cada frecuencia, es decir que se tiene que calcular de nuevo para lograr mayor exactitud en las frecuencias deseadas.

# Referencias

|  |  |
| --- | --- |
| [1] | ARDUINO, «ARDUINO STORE,» [En línea]. Available: https://store.arduino.cc/usa/due. [Último acceso: 17 Diciembre 2020]. |
| [2] | M. G. S. TOVAR, Enero 2016. [En línea]. Available: http://galia.fc.uaslp.mx/~ducd/cursos/DSP/Arduino\_DUE\_Enero2016.pdf. [Último acceso: 18 Diciembre 2020]. |