1. **GENERACIÓN DE ONDAS MEDIANTE ARDUINO**

**I.1 Introducción**

La familia Arduino tiene una gran variedad de placas cuya comparación se muestra en la Tabla 1. De aquí podemos ver que el Arduino-Due es el único que dispone de 2 convertidores D/A que facilita la generación de ondas pues para el resto de familias Arduino es necesario el uso de chips convertidores D/A externos.

**Tabla 1.** Comparación de características principales de familias Arduino.

|  |  |
| --- | --- |
| **FAMILIA ARDUINO** | |
| **TIPO** | **DESCRIPCIÓN** |
| Arduino Mega | * Pines digitales de entrada/salida: 54 (solo 15 proveen salidas PWM) * Pines de entrada analógica: 16 |
| Arduino Uno | * Pines de entrada/salida digital: 14 (solo 6 son salidas PWM) * Pines de entrada analógica: 6 |
| Arduino Uno Wifi | * Pines digitales de entrada/salida:14 * Pines de entrada analógica:6 |
| Arduino Uno R3 SMD | * Pines digitales de entrada/salida:14 * Pines de entrada analógica: 6 |
| **Arduino DUE** | * Pines digitales de entrada/salida:54 * Pines de entrada analógica: 12 * Pines de salida analógica: 2 |
| Arduino Leonardo | * Pines digitales de entrada/salida:20 * Pines de entrada analógica:12 |
| Arduino PRO Micro | * Pines digitales de entrada/salida:12 * Pines de entrada analógica:4 |
| Arduino NANO | * Pines digitales de entrada/salida:14 |
| Arduino YUN | * Pines digitales de entrada/salida:20 * Pines de entrada analógica:12 |
| Arduino ZERO | * Pines digitales de entrada/salida:20 * Pines de entrada analógica:6 * Pines de salida analógica: 1 |
| Arduino MKR WAN 1300 | * Pines digitales de entrada/salida:8 * Pines de entrada analógica:7 * Pines de salida analógica: 1 |
| Arduino MKR GSM 1400 | * Pines digitales de entrada/salida:8 * Pines de entrada analógica:7 * Pines de salida analógica: 1 |
| Arduino 101 | * Pines digitales de entrada/salida:14 * Pines de entrada analógica:6 |
| Arduino Ethernet | * Pines digitales de entrada/salida:14 * Pines de entrada analógica:6 |
| Arduino TRE | * Pines digitales de entrada/salida:14 * Pines de entrada analógica:6 |

Dichos pines D/A del Arduino-Due se pueden ver en la parte inferior izquierda de la Figura 1 (DAC0 y DAC1).



**Figura 1.** Dalidas D/A del Arduino-Due (DAC0 y DAC1)

# Detalles de la instalación del Arduino-Due lo tenemos en el siguiente enlace:

<https://www.youtube.com/watch?v=vftesYlN6oM>

**I.2 GENERACIÓN DE ONDA CON EL ARDUINO DUE**

# Si bien se pueden tener funciones directas, por ejemplo *sin* para el caso de función sinusoidal, o librerías externas que permitan generar ondas de forma simple, las limitaciones son evidentes pues no se podrá generar ondas que estén fuera de las opciones predeterminadas. Por esta razón, a continuación se tiene una metodología que permita generar cualquier tipo de onda.

# 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

# (confirmar esto y detallarlo buscando en la ayuda del Arduino Due).

# analogWriteResolution () es una extensión de la API analógica para Arduino Due. Establece la resolución de la función analogWrite (). Tiene un valor predeterminado de 8 bits (valores entre 0-255) para compatibilidad con las placas basadas en AVR.

# Arduino Due tiene las siguientes capacidades de hardware: 12 pines que por defecto son PWM de 8 bits, como las placas basadas en AVR. Estos se pueden cambiar a una resolución de 12 bits. 2 pines con DAC de 12 bits (convertidor digital a analógico)

# Si se configura el valor analogWriteResolution () en un valor superior a las capacidades de la placa, Arduino descartará los bits adicionales. Por ejemplo: al usar Due con analogWriteResolution (16) en un pin DAC de 12 bits, solo se usarán los primeros 12 bits de los valores pasados ​​a analogWrite () y se descartarán los últimos 4 bits. Si establece el valor analogWriteResolution () en un valor menor que las capacidades de la placa, los bits faltantes se rellenarán con ceros para completar el tamaño requerido por el hardware. Por ejemplo: usando Due con analogWriteResolution (8) en un pin DAC de 12 bits, Arduino agregará 4 bits cero al valor de 8 bits usado en analogWrite () para obtener los 12 bits requeridos.

# 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) (confirmar esto y detallarlo buscando en la ayuda del Arduino Due).

# Al establecer la resolución de escritura en 12, puede usar analogWrite () con valores entre 0 y 4095 para aprovechar la resolución DAC completa o para establecer la señal PWM sin pasar.

# En base a lo anterior, el procedimiento para generar cualquier onda sería el siguiente:

# PASO 1: Generar de tabla de valores de cualquier onda mediante Matlab

En un capítulo anterior ya se aprendió a generar con Matlab cualquier tipo de onda. Una consideración a tener en cuenta es que un microcontrolador trabaja **solo con voltajes positivos** por lo queen Matlab se debe obtener un conjunto de **N muestras solo positivas entre 0 y 1** para **un período de duración**. Luego, dependiendo de la **resolución** deseada (n = 8, 10 o 12 bits) y considerando que la **cantidad de niveles de cuantificación** queestá dado por 2n y los **valores enteros en decimal de las muestras de la señal** estarían entre 0 y 2n-1 estas muestras deberían ser multiplicada por el factor 2n-1. En la Tabla 2 podemos ver estos valores para 8, 10 y 12 bits de resolución:

**Tabla 2.** Niveles de cuantificación y rango de valores para distintas resoluciones.

|  |  |  |
| --- | --- | --- |
| **Resolución** | **Cantidad de niveles de cuantificación = 2n** | **Rango de valores decimales (0-2n-1)** |
| 8 bits | **2n** = 256 | 0 - 255 |
| 10 bits | **2n** = 1024 | 0 - 1023 |
| 12 bits | **2n** = 4096 | 0 - 4095 |

**EJEMPLO:** Generar con el Arduino-Due una señal sinusoidal de 100 Hz con una resolución de 8 bits / muestra.

**Resolución**: empezamos encontrando las **N muestras positivas en un ciclo de onda sinusoidal**, independientemente de la Frecuencia por lo que se considera Fo = 1. Después, cuando en el Arduino Due se realice el barrido mediante un lazo for, la **frecuencia real** estará determinada por el **tiempo entre muestras Ts**. Además, en las pruebas se podrá observar en qué influye este número de muestras.

%Obtención de N muestras positivas de la señal sinusoidal entre 0 y 1:

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 = 40; %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 = 8; % Resolución

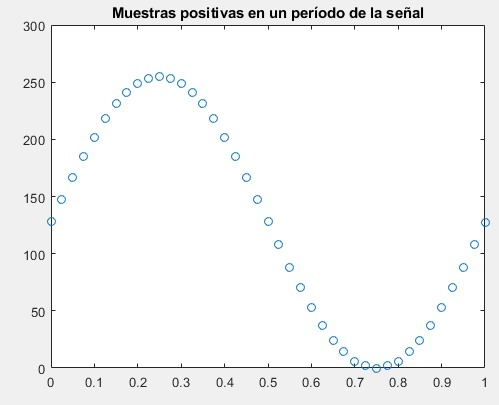
yn1=yn\*(2^n-1);% valores decimales entre 0 y (2^n -1).

yn2=round(yn1); %redondear a valores enteros

plot(tn,yn2,'o') %gráfico de comprobación

title('Muestras positivas en un período de la señal')

El resultado lo podemos ver en la Figura 2.



**Figura 2.** N = 40 muestras positivas en un ciclo de onda sinusoidal.

Del workspace podemos obtener las N=40 muestras encontradas (yn2):

128 147 167 185 202 218 231 241 249 253 255 253 249 241 231 218 202 185 167 147 128 108 88 70 53 37 24 14 6 2 0 2 6 14 24 37 53 70 88 108 127

**PASO 2. A continuación, realizamos el programa para el Arduino-Due:**

Las muestras anteriores generadas en Matlab pueden ser copiadas y pegadas en una tabla dentro del programa en el Arduino Due, que en este caso se llamará **seno**:

int **seno**[ ] = {

128,147,167,185,202,218,231,241,249,253,255,253,249,241,231,218,202,185,167,147,128,108,88,70,53,37,24,14 6,2,0,2,6 ,14,24,37,53,70,88,108,127};

Luego definimos el pin DAC0 como salida y configuramos la resolución deseada, que en este ejemplo es de 8 bits:

void setup()

{

pinMode(DAC0, OUTPUT);

analogWriteResolution(8);

}

**Determinación del retardo entre muestras:**

Después, en base a la **frecuencia fundamental** (**Fo)** de la onda deseada, se calcula el **período To** de la onda (To = 1/Fo). Con esto hay que calcular el tiempo entre muestras (Ts = To / N) que vendrá a ser el retardo entre muestras y que será empleado dentro de un **lazo for** con el que se envían estas muestras hacia un pin DAC del Arduino. Ya que estetiene una **instrucción de retardo en microsegundos** (delayMicroseconds(us)), el **Ts** en micro segundos estaría dado por:

# Ts\_us = (To / #MUESTRA)\* 1000000 (us)= 1000000 / (Fo \* #muestras)

# En el ejemplo planteado, si tenemos una onda de Fo = 100 Hz y tomamos 40 muestras por período:

# Ts\_us = 1000000/(Fo\* 40) que debería ser un valor entero lo cual se lo define mediante *int*:

# Entonces, la instrucción quedaría de la siguiente manera:

# // para Fo = 100 Hz y 40 muestras por período:

# int Ts\_us = 1000000/(100\*40);

# Barrido de las N muestras incluido el retardo:

# Las N muestras de la tabla son enviadas al DAC mediante un lazo for que incluye el retardo anteriormente encontrado. Para el caso de una resolución de 8 bits (valor máximo = 255) tenemos la programación siguiente:

void loop()

{

count=0;

# for (count=0; count<255; count++)

# {

# analogWrite(DAC0, seno[count]);

# delayMicroseconds(Ts\_us );

# }

}

Al compilar y ejecutar el programa anterior en el Arduino Due, el bucle realizará un barrido continuo de las muestras de la tabla, que son las muestras de un periodo de la señal, generando así la onda deseada.

**Mediciones por medio de instrumento externo**

Algunas mediciones que se podrían realizar serían las siguientes:

1. Encontrar el error cometido en la frecuencia obtenida respecto de la frecuencia deseada.

¿Por qué se da?

1. Medir la mínima frecuencia (Fmin):

¿Se puede generar la onda con una Frecuencia tan baja como se desee o se tiene alguna limitación?

Ya que el audio tester, por la limitaciones en frecuencia del puerto de audio, no puede realizar mediciones en la parte baja de frecuencia, hace falta necesariamente un instrumento externo u otra metodología de medición como por ejemplo por medio de (arduino + Matlab) que lo veremos más adelante.

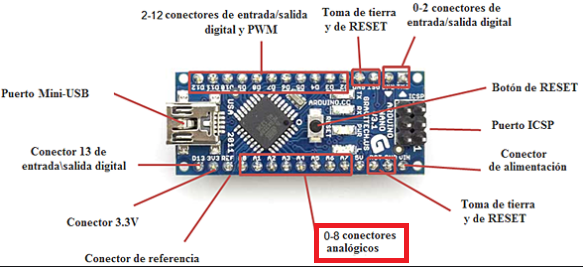
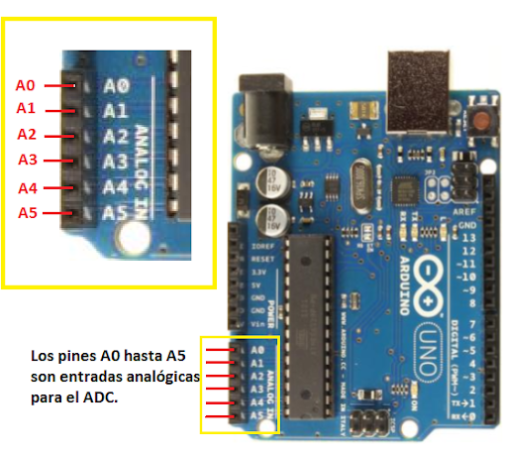
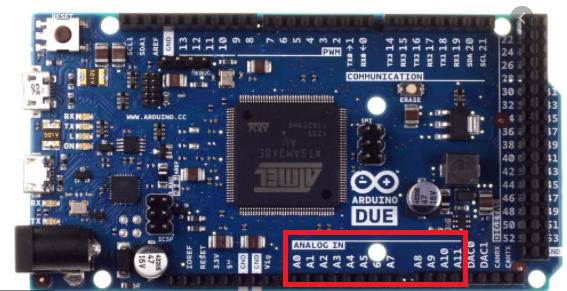
1. Medir la máxima frecuencia (Fmax) controlada para un T\_us = 1, pues ya no se puede poner un valor inferior a este. Se deberá observar el error cometido respecto de la frecuencia deseada.

Mientras no se sobrepase las limitaciones del puerto de audio (hasta unos 20 KHz), se lo podrá realizar con audio Tester, de lo contrario se requerirá necesariamente un instrumento externo u otra metodología de medición como por ejemplo por medio de (arduino + Matlab) que lo veremos más adelante.

1. Analizar la influencia del Número de muestras (N =20, 40, 60, 80, 100) sobre Fmax y Fmin. Se podría realizar una tabla de resultados.
2. Analizar la influencia de la resolución (n = 8, 10, 12 o lo que permita el Arduino-Due) sobre Fmax y Fmin. Se podría realizar una tabla de resultados.
3. Medir de voltajes obtenidos. Aquí, si se lo hace con audio Tester, se debería tener cuidado recordando que el puerto de audio está acoplado en AC por lo que las componentes DC que tuviera la onda quedarían eliminados y no se observaría la onda original de forma correcta.
4. **ADQUISICIÓN DE SEÑALES ANALÓGICAS MEDIANTE ARDUINO**

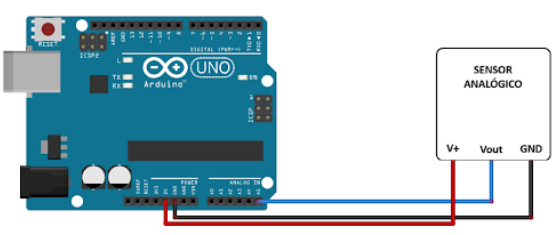
**II.1 INTRODUCCIÓN**

Prácticamente todas las familias de Arduino traen entradas ADC por medio de las cuales se puede adquirir muestras de señales analógicas externas (Figura 3).

****

**Figura 3.** Entradas analógicas para Arduino Due, Uno y Nano.

Por lo anterior, un **sensor que obtenga una señal analógica** deberá ser alimentado desde el Arduino y dicha señal analógica deberá ser conectada a una de sus entradas analógicas. Para el caso de un Arduino UNO, se lo puede ver en la Figura 4.



**Figura 4.** Sensor de señal analógica siendo alimentado por el Arduino (GND y V+) y dicha señala analógica ingresando a una de sus entradas analógicas.

**II.2 Programación en el Arduino**

La instrucción que permite leer un pin analógico es ***analogRead*** y su sintaxis es la siguiente:

Variable = **analogRead** (pin\_analógico);

Un ejemplo de programa que lea la entrada de un sensor conectado a la **entrada analógica A2** y que muestre los valores adquiridos por medio de la **herramienta comunicación serial** disponible en el IDE, sería el siguiente:

int SENSOR; // variable para almacenar valor leído de A2

void setup(){

Serial.begin(9600); // inicializa comunicación serie a 9600 bps

// entradas analógicas no requieren inicialización

}

void loop(){

SENSOR = analogRead(A2); // lee y almacena valor leído de entrada analógica A2

Serial.println(SENSOR); // imprime en monitor serial el valor leído

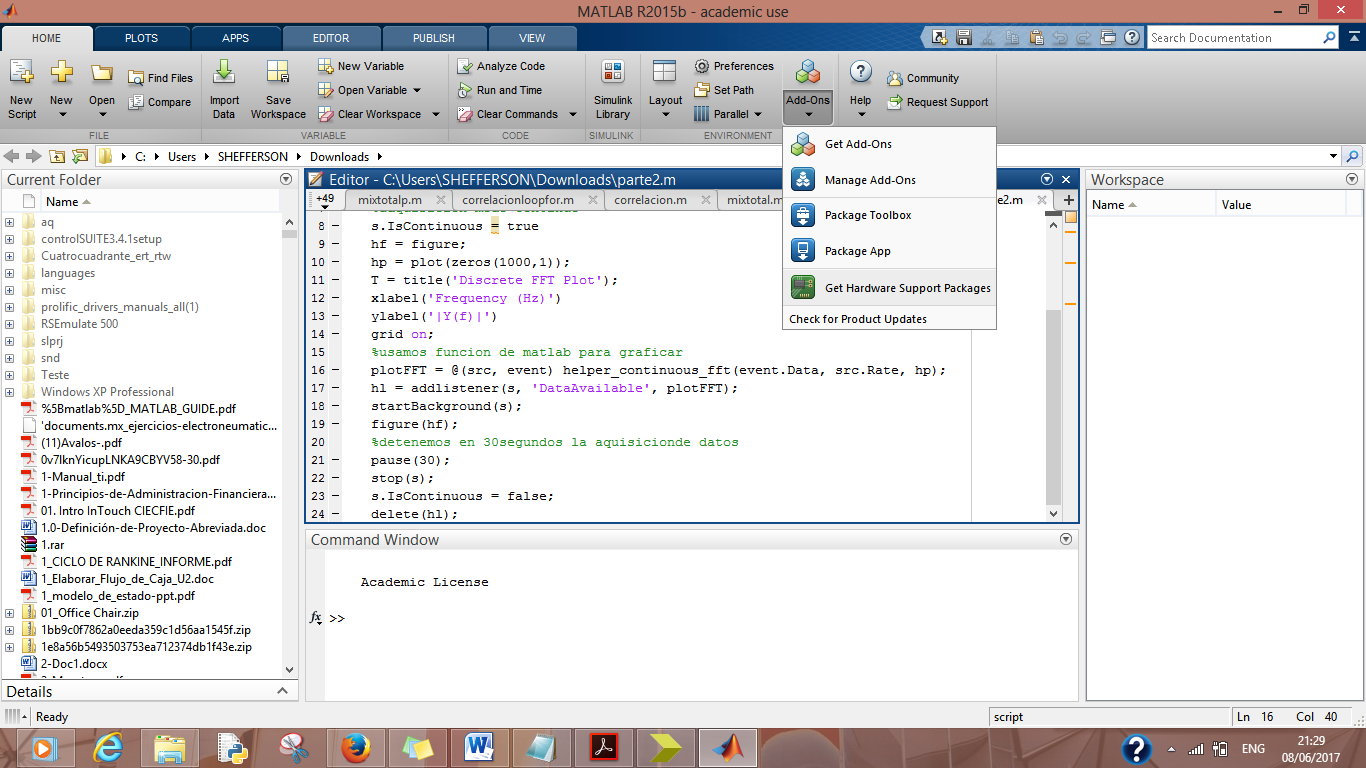
delay(100); // demora de 100 mili segundos

}

Si bien se tiene la posibilidad de observar numéricamente dichos valores adquiridos por el sensor, son demasiado limitadas las opciones que se tiene para observarlas y guardarlas en un archivo. Por esta razón, se propone realizarlo mediante interacción entre Arduino y Matlab el cual tiene todas las potencialidades para mostrar dicha señal en gráficas, analizar las señales tanto en el dominio del tiempo como de la frecuencia y almacenarlas en archivos de distinto tipo (txt, Excel, mat, etc).

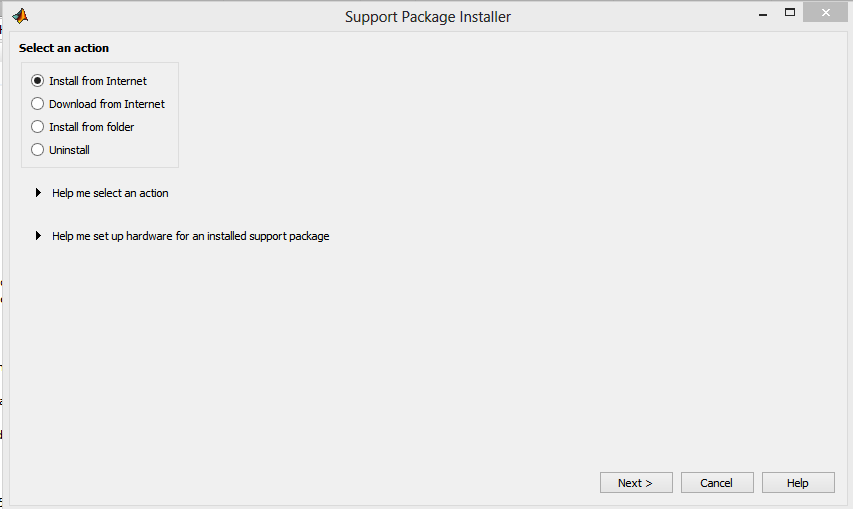
**II.3 PROCESO DE INSTALACIÓN DE LA TARJETA ARDUINO EN MATLAB**

Para instalar la librería de Arduino en Matlab nos dirigimos a la opción Add-Ons / Get Hardware Suport Packages (Figura 5).



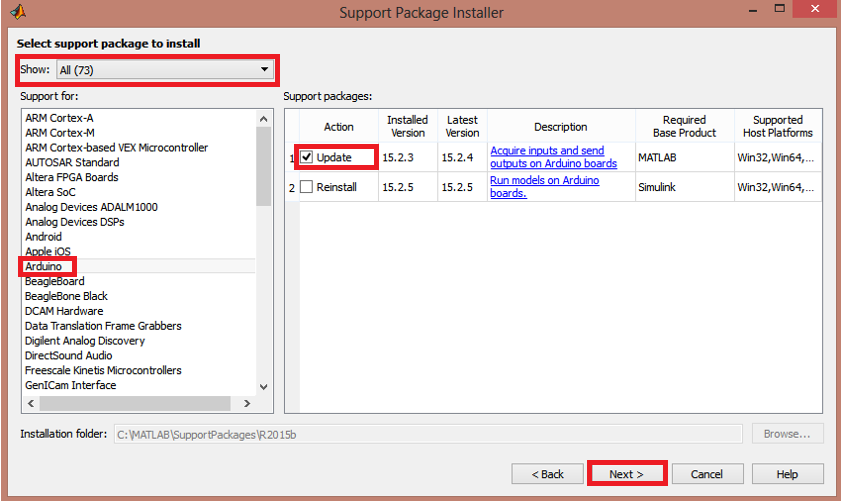
**Figura 5.** Opción Add-Ons / Get Hardware Suport Packages de Matlab.

Aparecerá un cuadro donde se deberá escoger **Install from Intenet** y hacemos clic en Next (Figura 6).



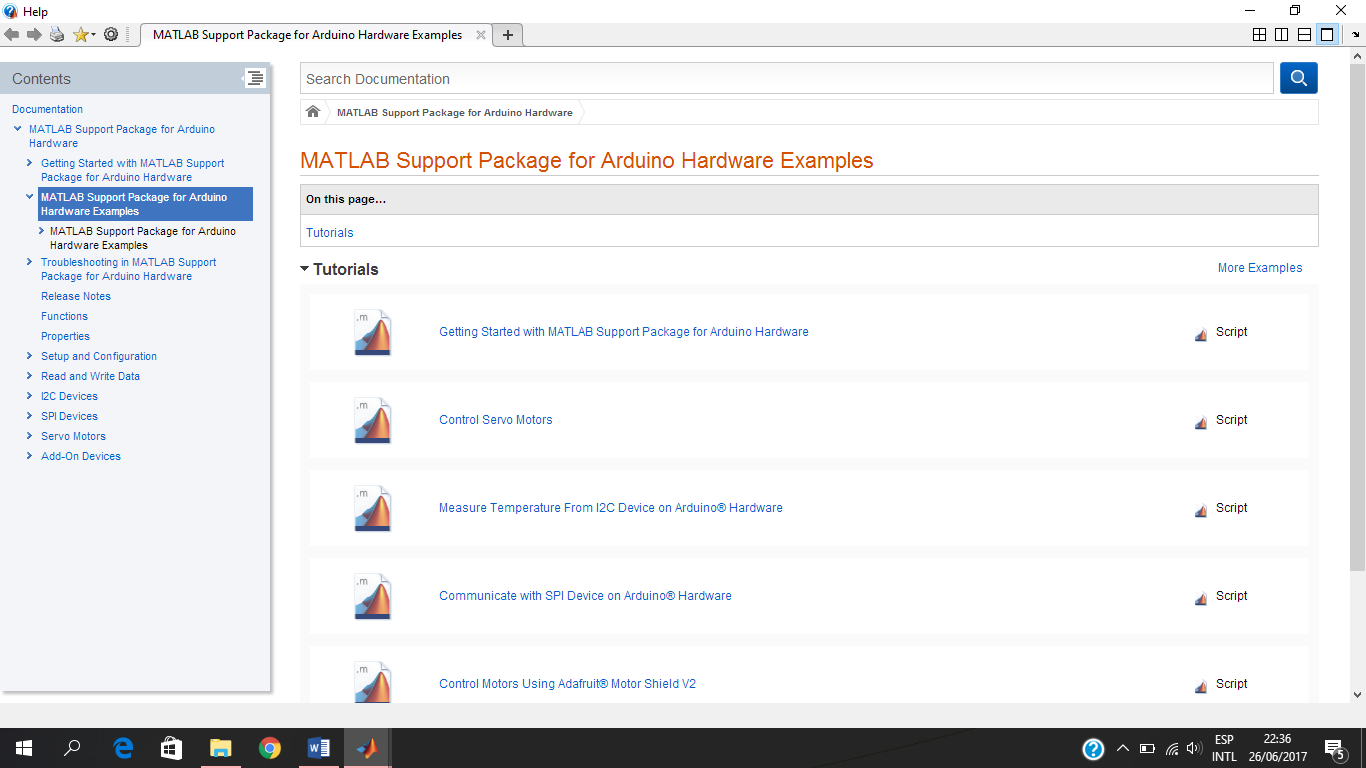
**Figura 6.** Seleccionar la Opción **Install from Intenet**.

En la opción **Show** escogemos **All,** seleccionamos **ARDUINO,** clic en la opción **Update** y **Next** (Figura 7).



**Figura 7.** En la opción **Show** escogemos **All,** seleccionamos **ARDUINO,** clic en la opción **Update** y **Next**.

Se inicia la instalación de las distintas librerías necesarias y al final aparecerá la ayuda mostrada en la Figura 8 que puede ser encontrada también en la **ayuda de Matlab** buscando la palabra clave Arduino.



**Figura 8.** Pantalla de ayuda que aparece al finalizar la instalación de las librerías de Arduino dentro de Matlab.

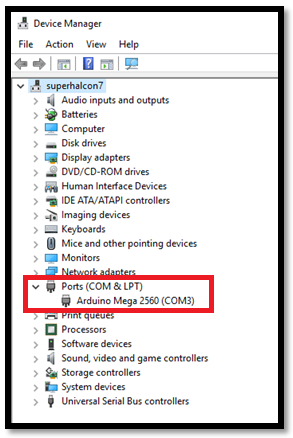
A continuación mostramos un par de ejemplos que constan en la ayuda anterior:

**EJEMPLO 1**: Si se tiene conectado un Arduino Mega, realizar el encendido y apagado de un led (conectado al pin digital D13) con tiempo de encendido 0.5 segundos y apagado 1 segundo.

**Resolución**:

Primero conectamos el led al pin 13 de salida.

Mediante la instrucción ***arduino***, se crea un objeto que requiere sabe en qué puerto COM está conectada la tarjeta Arduino y qué tipo de tarjeta es. En nuestro caso, vemos que el ARDUINO MEGA está conectado al puerto COM3 (Figura 9):



**Figura 9.** Pantalla que indica que el Arduino Mega está conectado en el puerto Com3.

Con este detalle, el programa de Matlab que realizará lo solicitado, es el siguiente:

clc, close all, clear all

%% Crear un objecto arduino:

a = arduino('com3', 'Mega2560');

% LED: encendido 0.5 segundos y apagado 1 segundo:

for i = 1:5

writeDigitalPin (a, 'D13', 1);

pause(0.5);

writeDigitalPin (a, 'D13', 0);

pause(1);

end

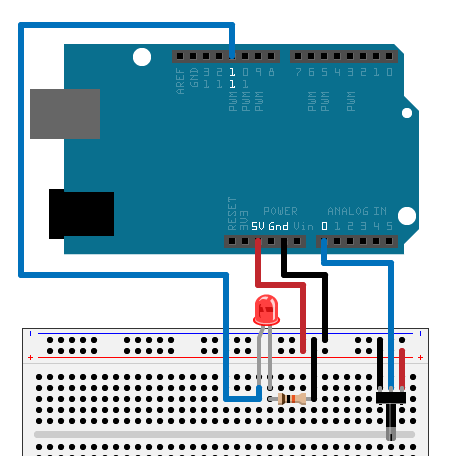
**EJEMPLO 2: Control del brillo de un LED usando un potenciómetro externo**

Conecte un potenciómetro al hardware Arduino con la pata del medio conectada al pin analógico 0 y los otros dos conectados a 5V y GND (Figura 10). El led está conectado al pin digital 11.

El brillo del LED es controlado por medio de un potenciómetro, cuyo voltaje variable es conectado al pin analógico 0, que es utilizado para establecer el nivel de voltaje en el pin PWM.

Las instrucciones empleadas son:

* ***readVoltage***: lee el pin A0 donde está conectado el potenciómetro
* ***writePWMVoltage***: es diferente al anterior writePWMDutyCycle



**Figura 10.** Conexiones del potenciómetro al pin analógico de entrada 0.

El programa quedaría de la siguiente manera:

time = 200;

while time > 0

voltage = readVoltage (a, 'A0');

writePWMVoltage (a, 'D11', voltage);

time = time - 1;

pause(0.1);

end

**OBSERVACIÓN:**

La instrucción ***pause***(*segundos)* estaría haciendo que las muestras de voltaje sean leídas cada 0.1 segundos, es decir, la tasa de muestreo sería de 10 muestras / segundo o 10 Hz.

**¿Cuál será el mayor valor de tasa de muestreo que se puede tener?**

**NOTA:** En los ejemplo de la ayuda anteriormente indicada existen muchos más ejemplo con los que usted podría entrenarse.

**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.**

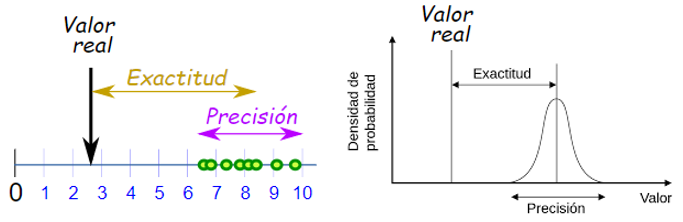
**PARTE 1:** **GENERADOR** **BÁSICO** CON ARDUINO Y **MEDICIÓN** CON AUDIO TESTER

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.

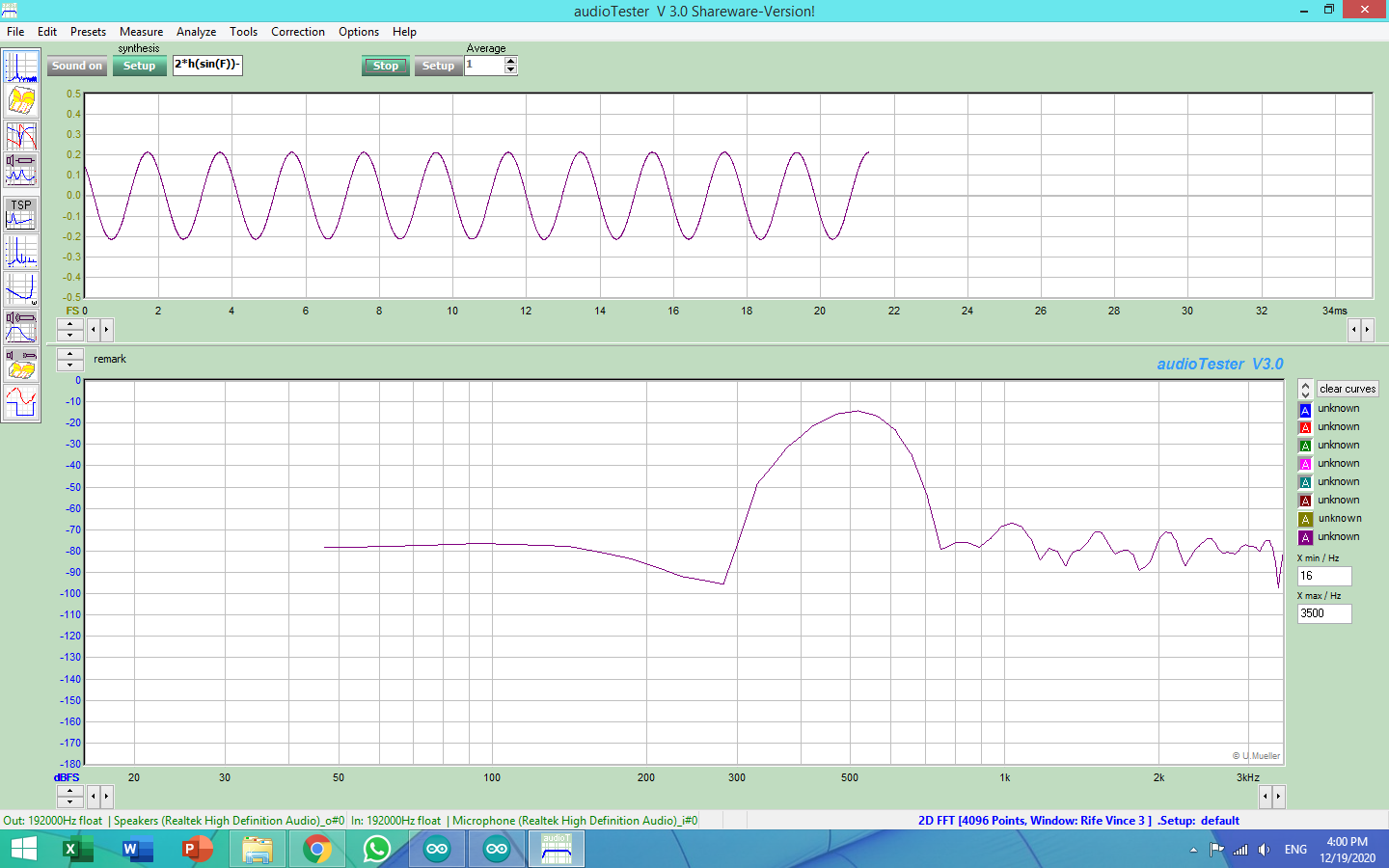
**OBSERVACIÓN**: La **exactitud**es la cercanía de una medida al **valor real**, mientras que la **precisión** es el grado de cercanía de los valores de **varias medidas** en un punto.



**RESULTADOS OBTENIDOS: SEÑAL SINUSOIDAL**

1. **Uso de la función Sin**

**f=500Hz**



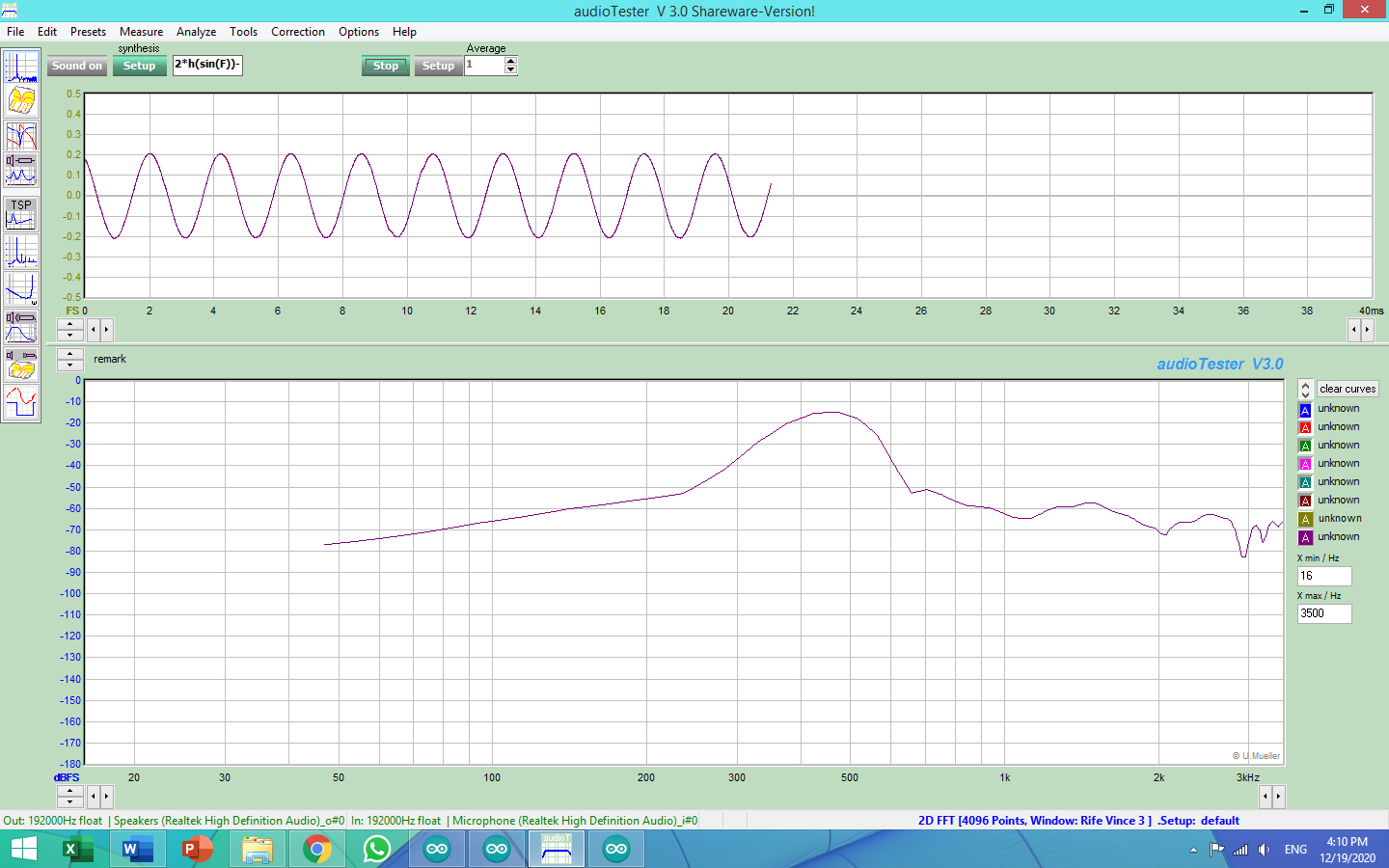
9.01 ms

**2 ms**

11.01 ms

**Fig.1 Deber Resultado obtenido con el uso de la función Sin con f=500Hz**

1. **Uso de una librería externa de Arduino**



10.34 ms

8.25 ms

**2.09 ms**

**Fig.2 Deber Resultado obtenido con el uso de una librería externa con f=500Hz**

1. **Metodología aprendida en clase:**



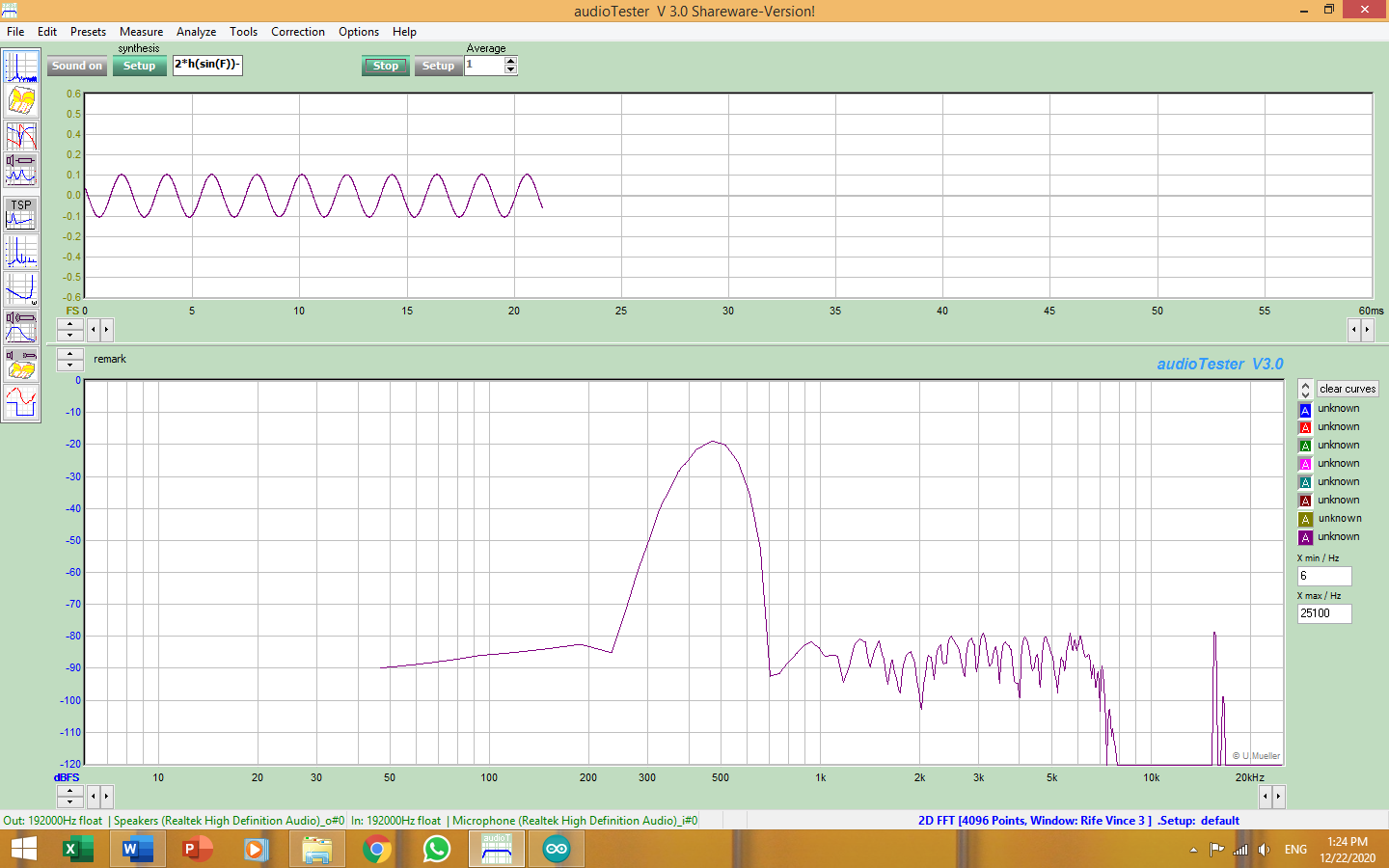
8.85 ms

10.98 ms

**2.13 ms**

**Fig.3 Deber Resultado obtenido con método aprendido en clase con f=500Hz**

1. **Método nuevo**



12.73 ms

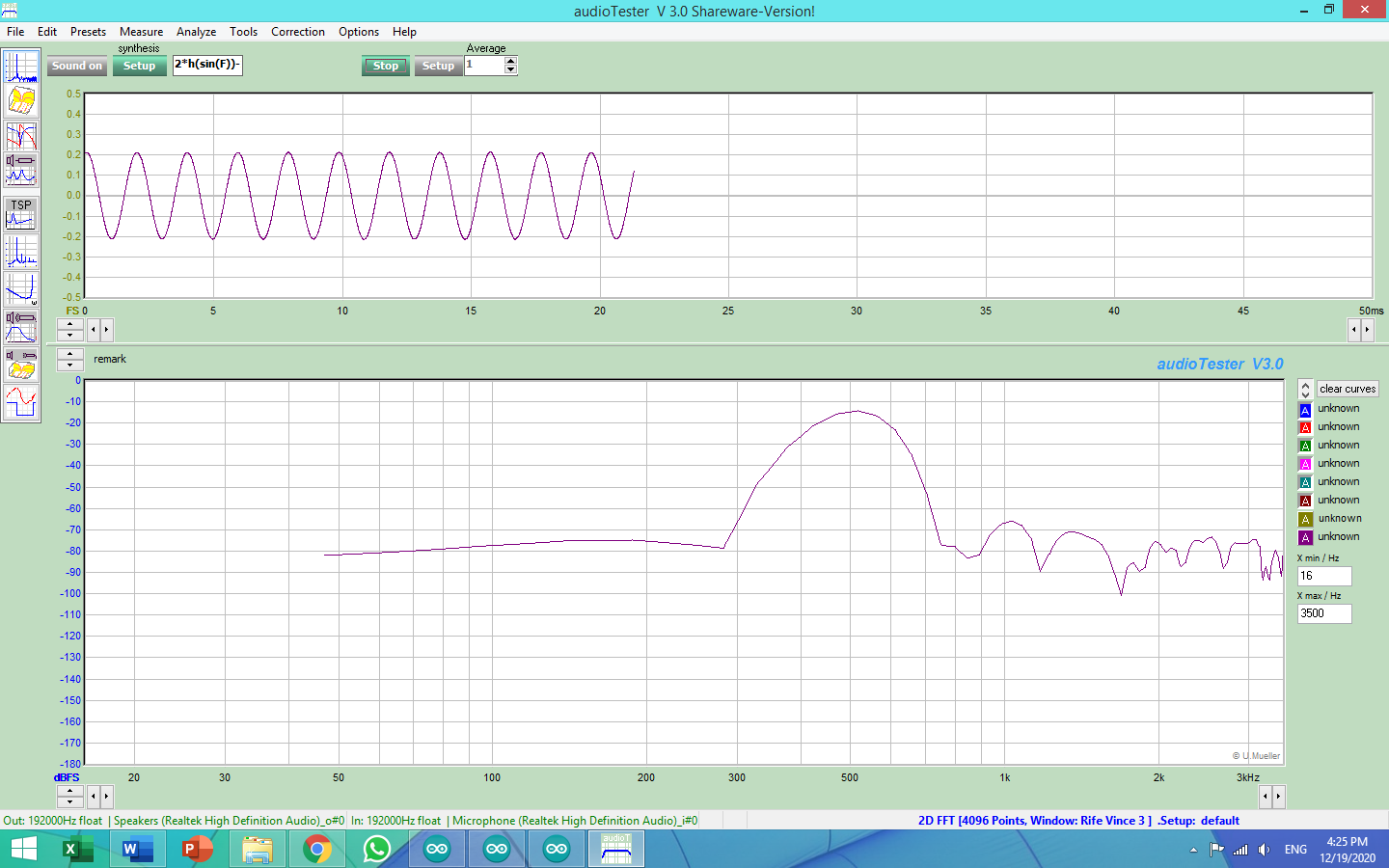
14.71 ms

**1.98 ms**

**Fig.4 Deber Resultado obtenido con el uso de un nuevo método con f=500Hz**

**F=750 Hz**

1. **Uso de la función Sin**



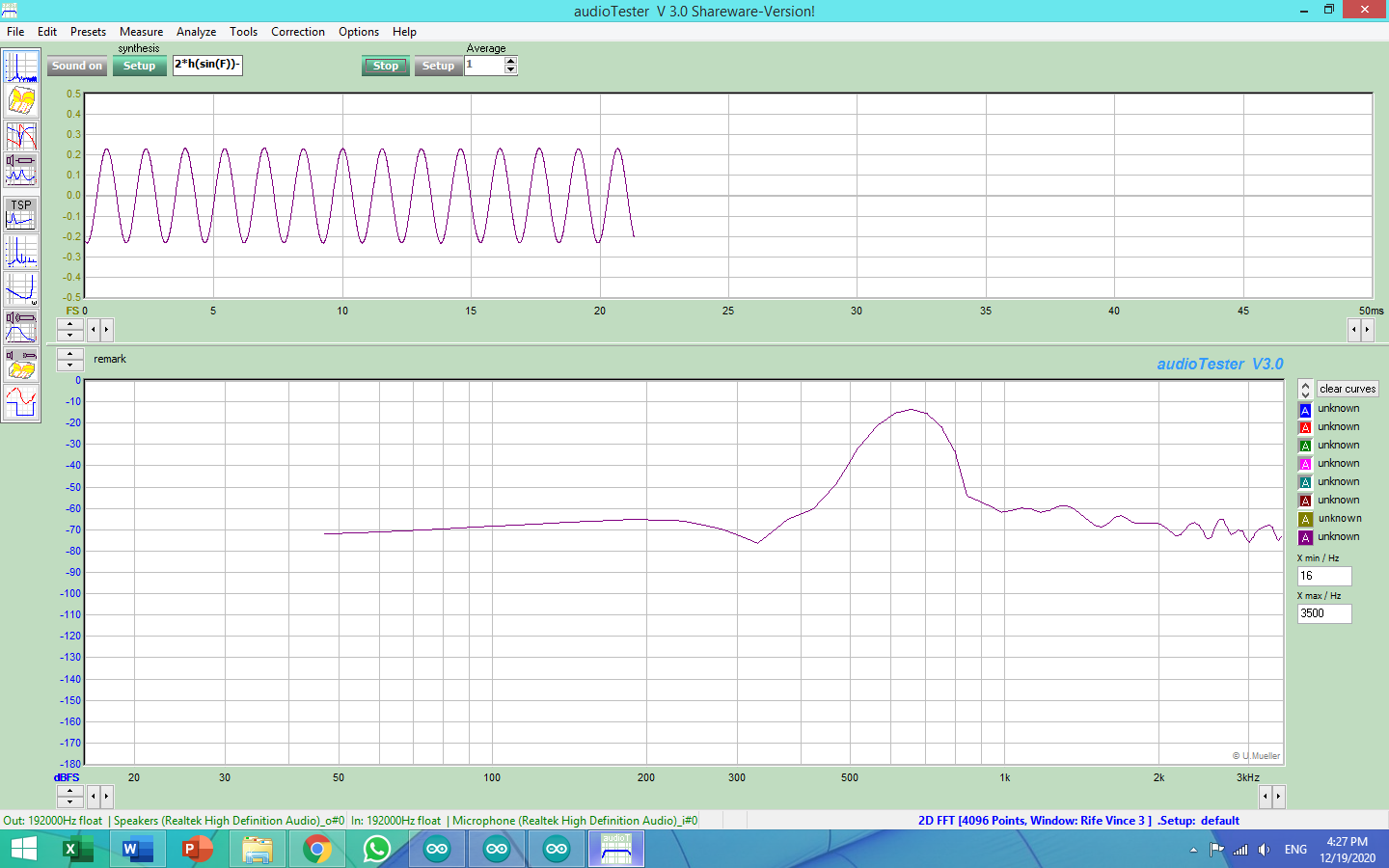
11.21 ms

**1.25 ms**

9.96 ms

**Fig.5 Deber Resultado obtenido con el uso de la función Sin con f=750Hz**

1. **Uso de una librería externa de Arduino**



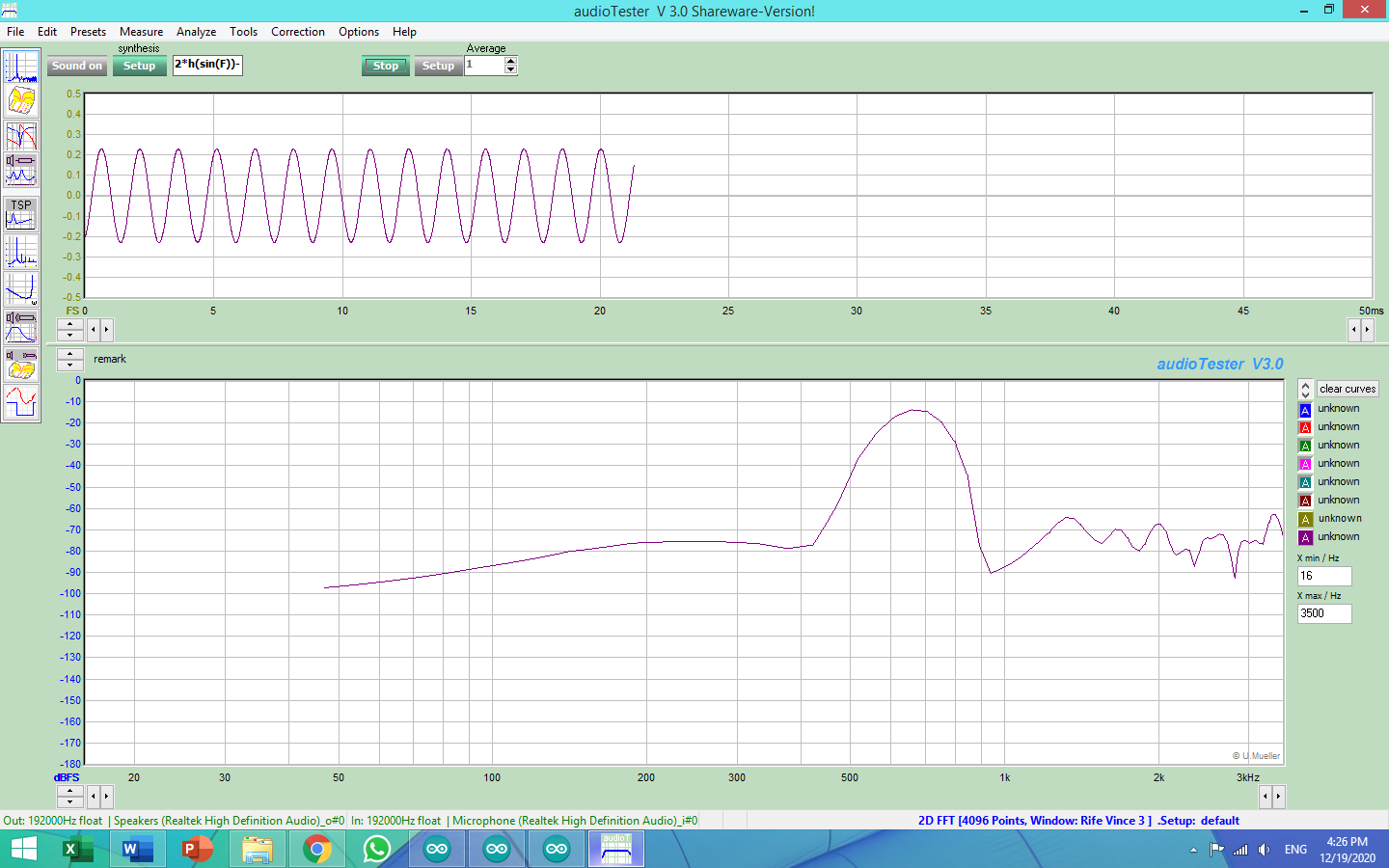
8.22 ms

9.78 ms

**1.56 ms**

**Fig.6 Deber Resultado obtenido con el uso de una librería externa con f=750Hz**

1. **Metodología aprendida en clase:**



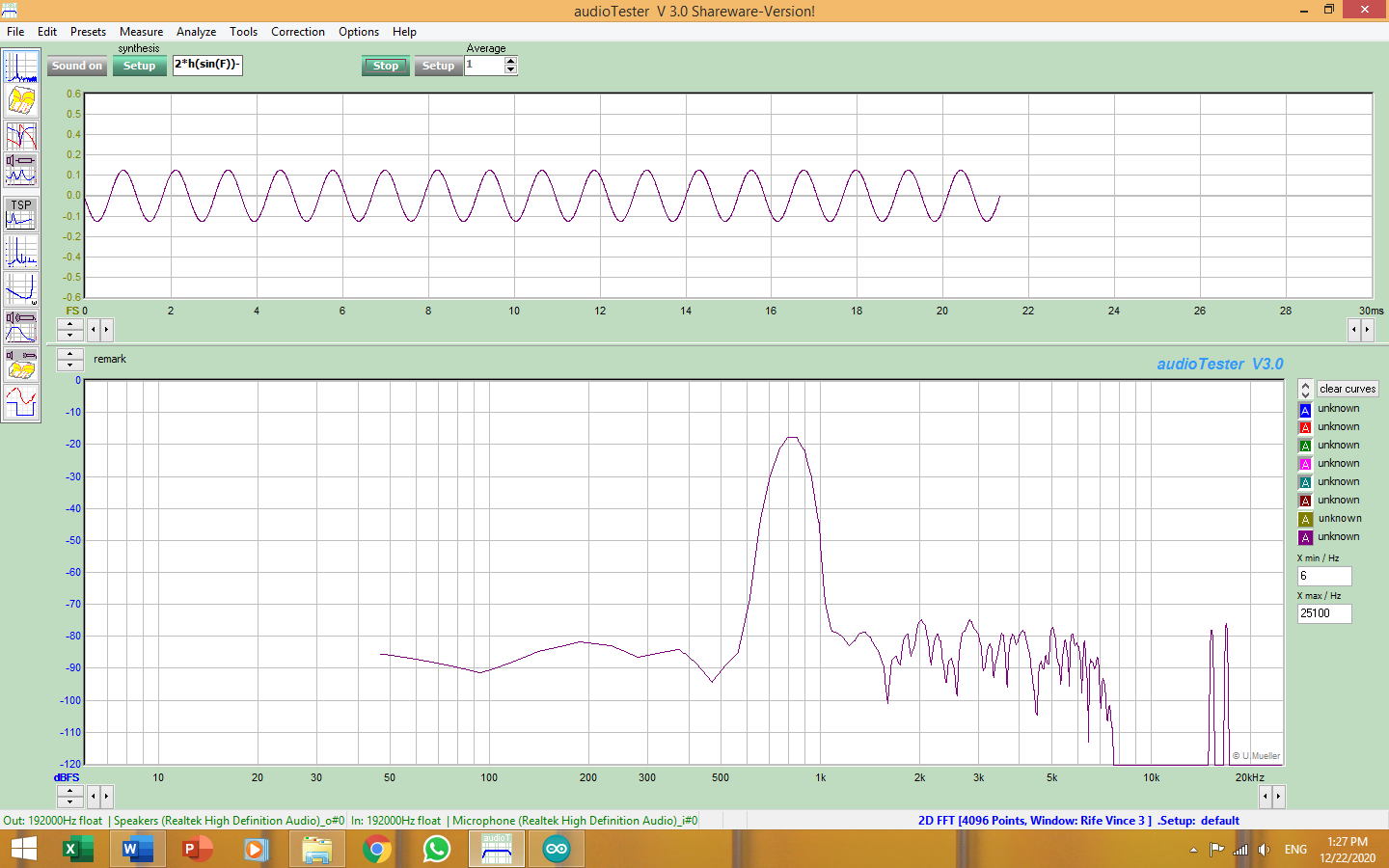
10.16 ms

11.66 ms

**1.5 ms**

**Fig. 7 Deber Resultado obtenido con método aprendido en clase con f=750Hz**

1. **Método nuevo**



**1.24 ms**

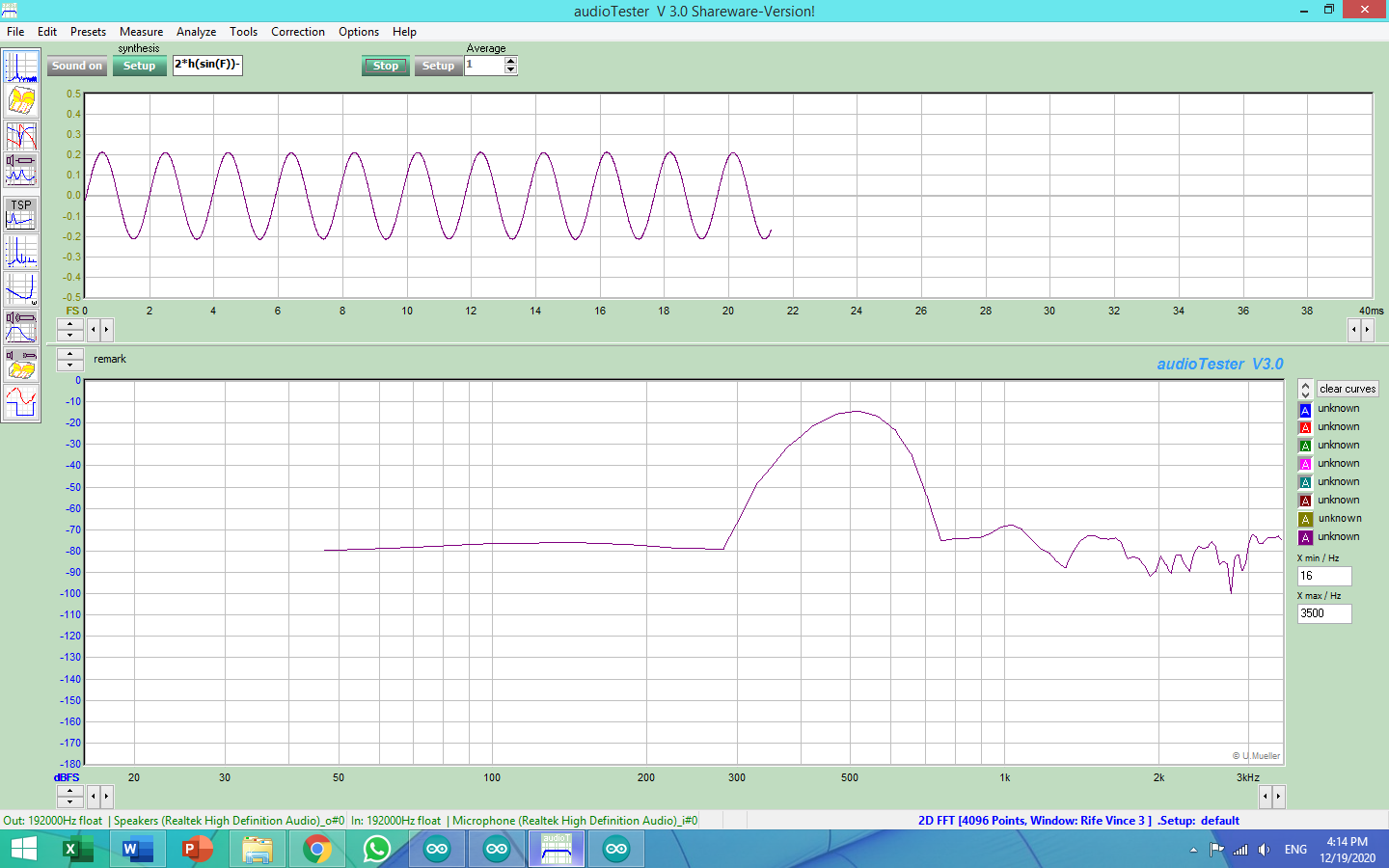
9.58 ms

8.34 ms

**Fig. 8 Deber Resultado obtenido con un nuevo método con f=750Hz**

**F=1000 Hz**

1. **Uso de la función Sin**



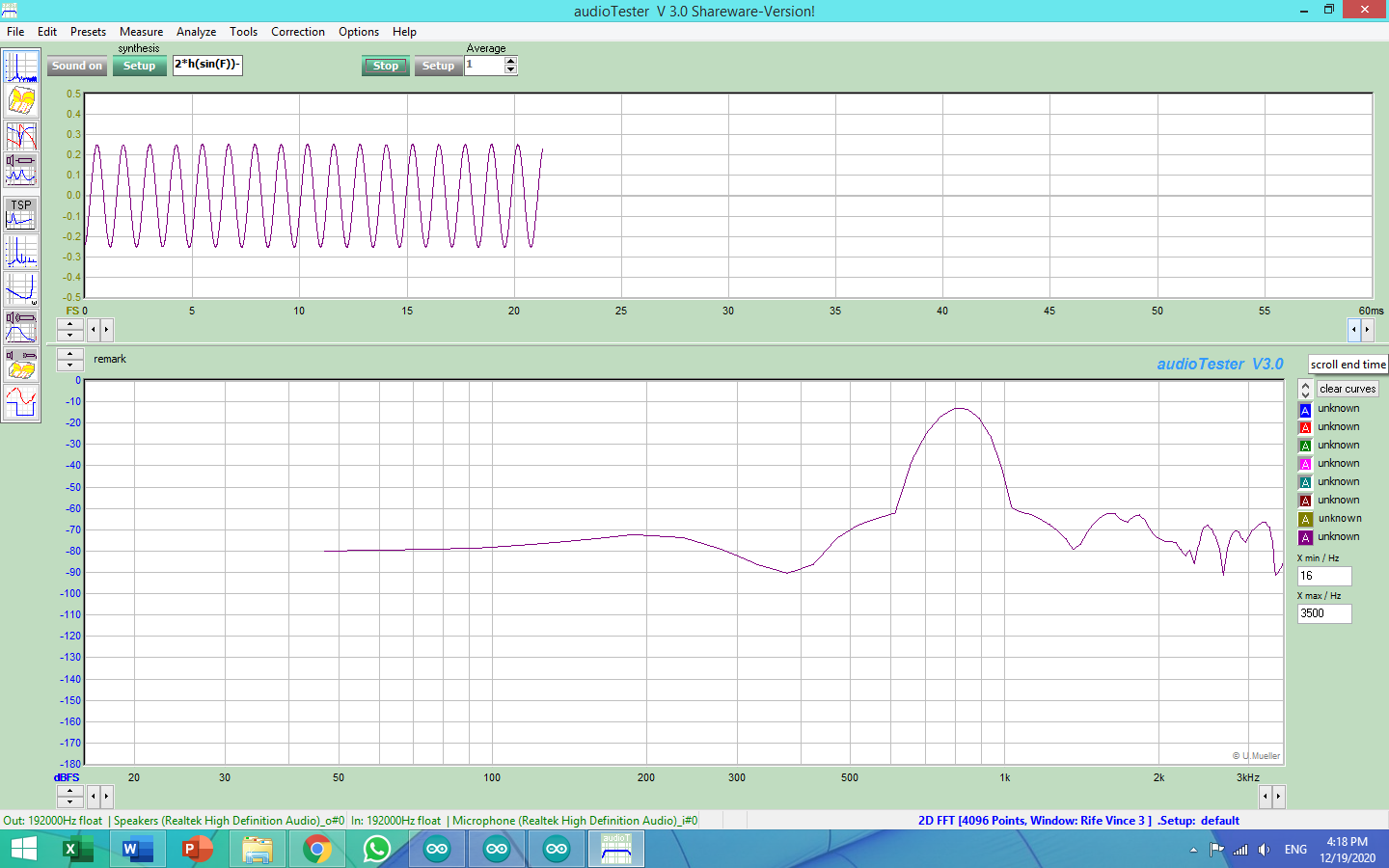
9.42 ms

11.42 ms

**2 ms**

**Fig. 9 Deber Resultado obtenido con el uso de la función Sin con f=1000Hz**

1. **Uso de una librería externa de Arduino**



**1.07 ms**

9.75 ms

8.68 ms

**Fig.10 Deber Resultado obtenido con el uso de una librería externa con f=1000Hz**

1. **Metodología aprendida en clase:**



9.66 ms

8.63 ms

**1.03 ms**

**Fig. 11 Deber Resultado obtenido con método aprendido en clase con f=1000Hz**

1. **Método nuevo**



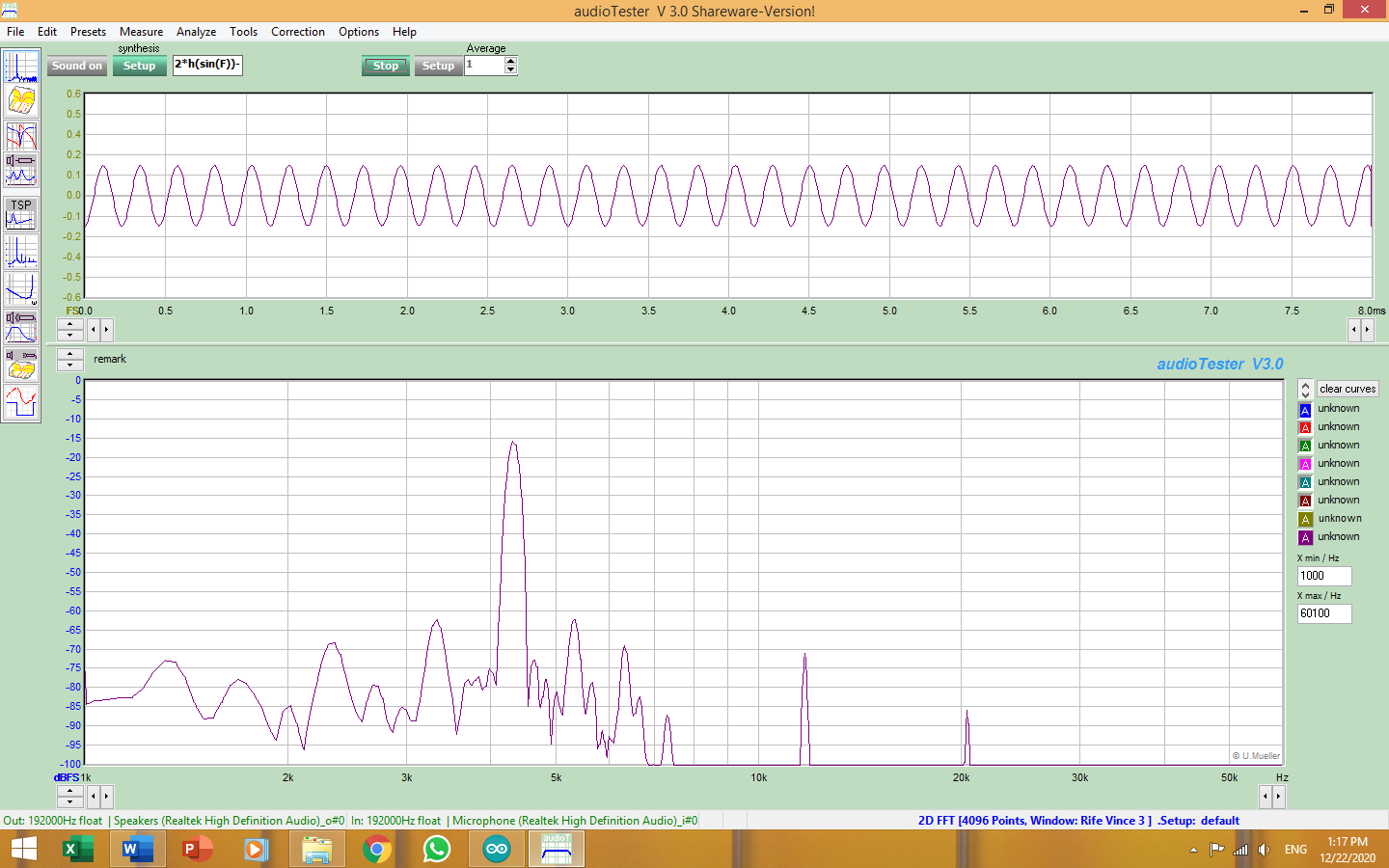
**0.1 ms**

11.39 ms

10.22 ms

**Frecuencia Máxima**

1. **Uso de la función Sin**

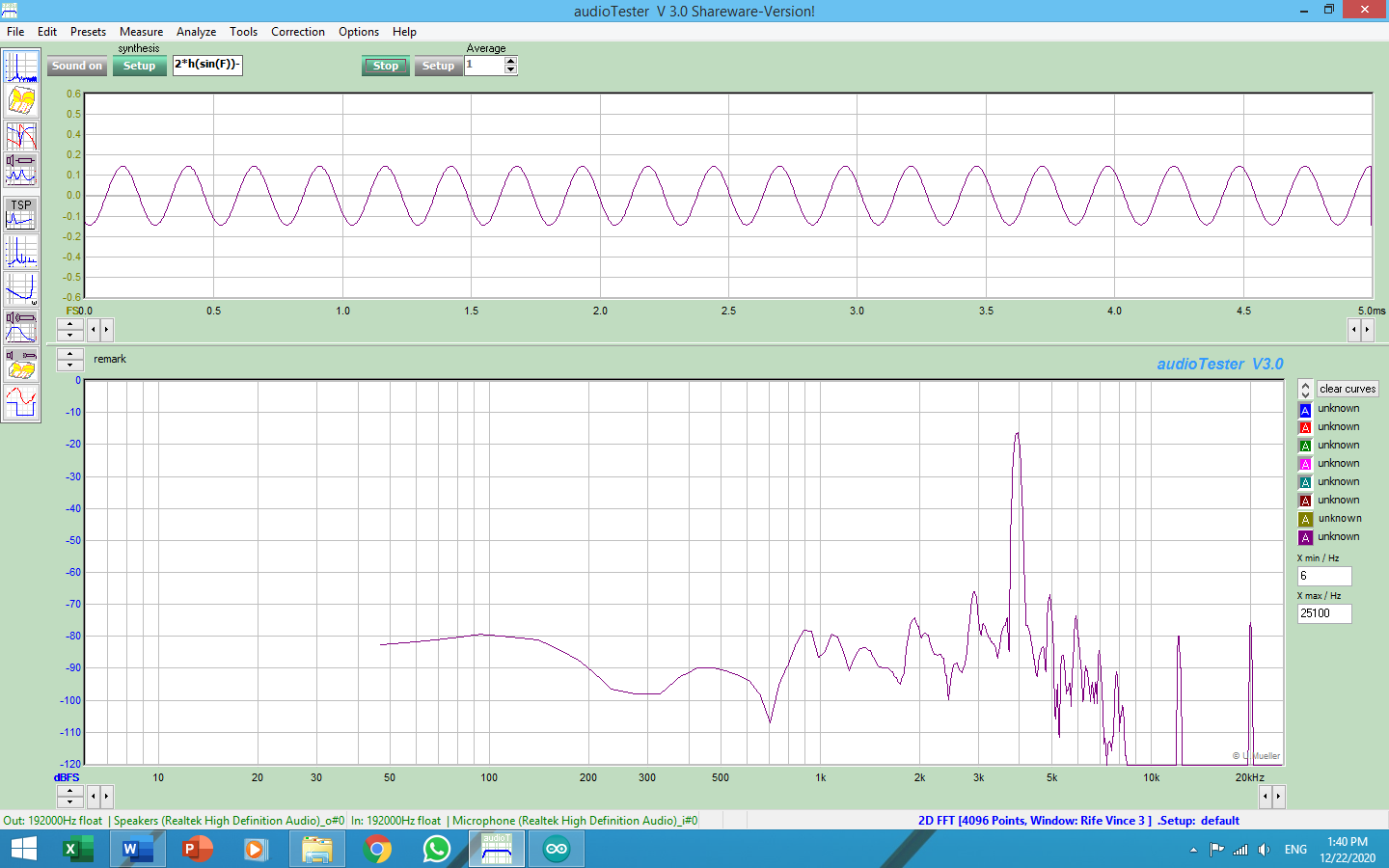


3.05 ms

3.26 ms

**0.21 ms**

1. **Librería externa**

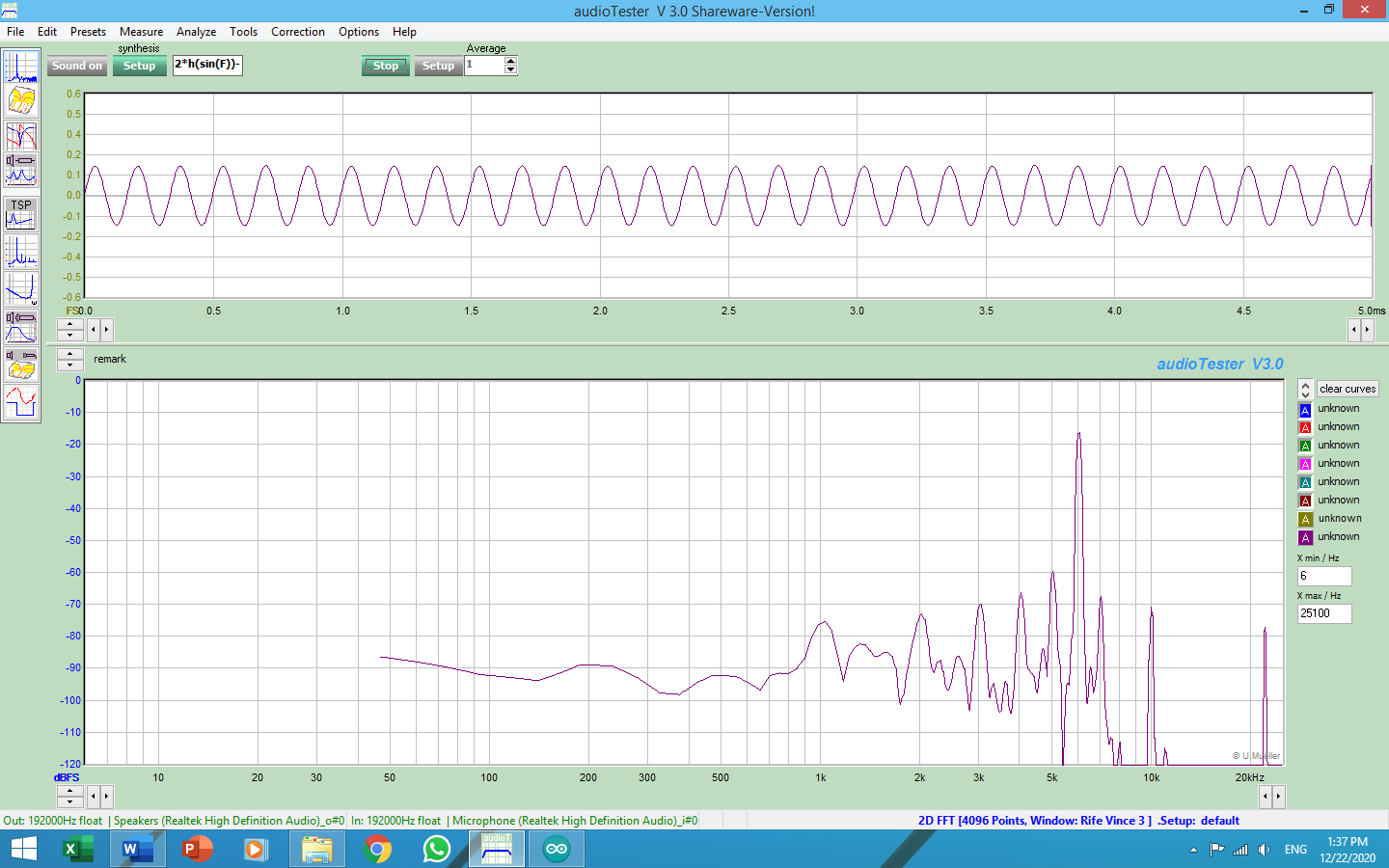


2.41 ms

2.22 ms

**0.19 ms**

1. **Metodología aprendida**

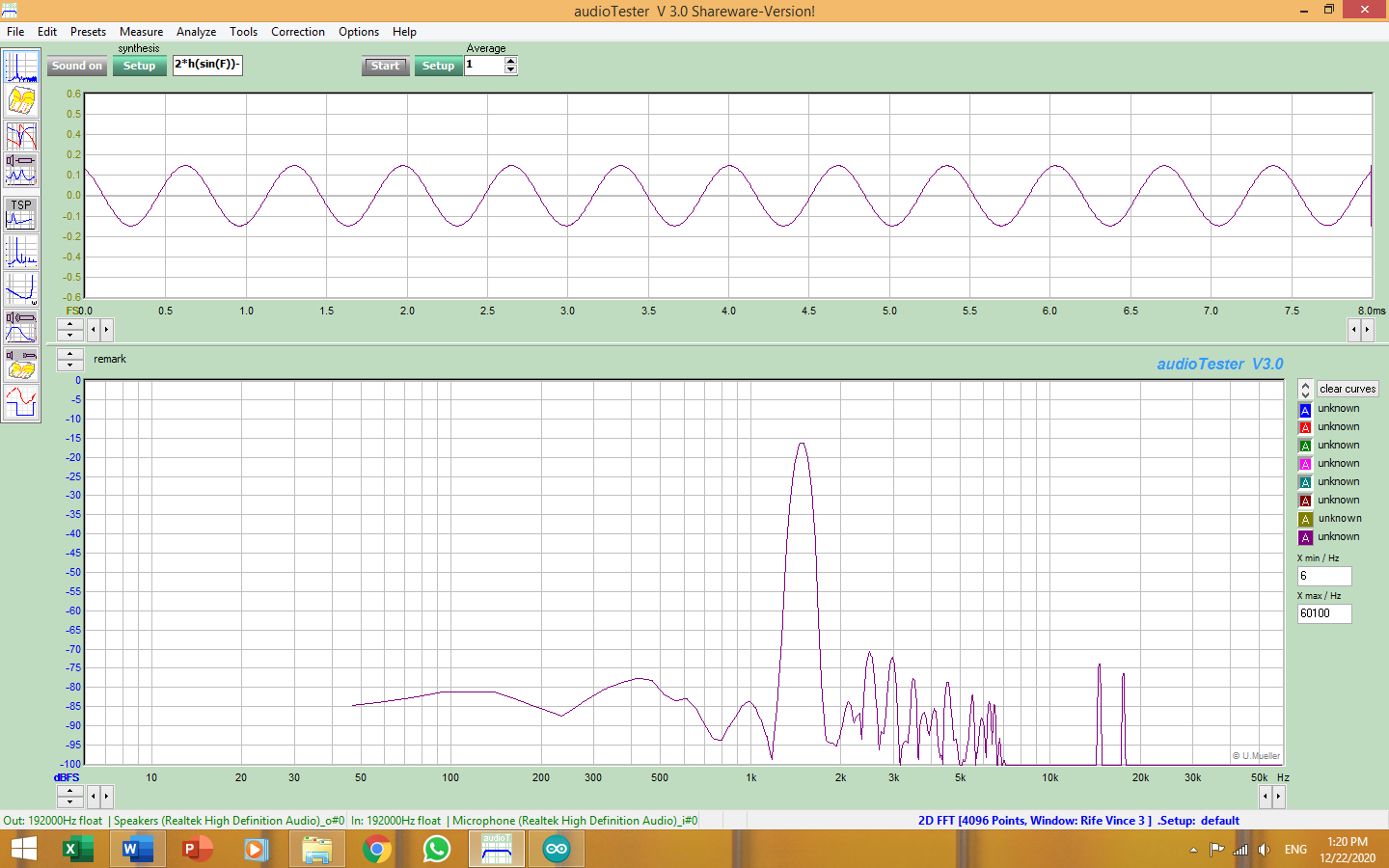


2.29 ms

2.13 ms

**0.16 ms**

1. **Nuevo método**



3.24 ms

**0.5 ms**

2.74 ms

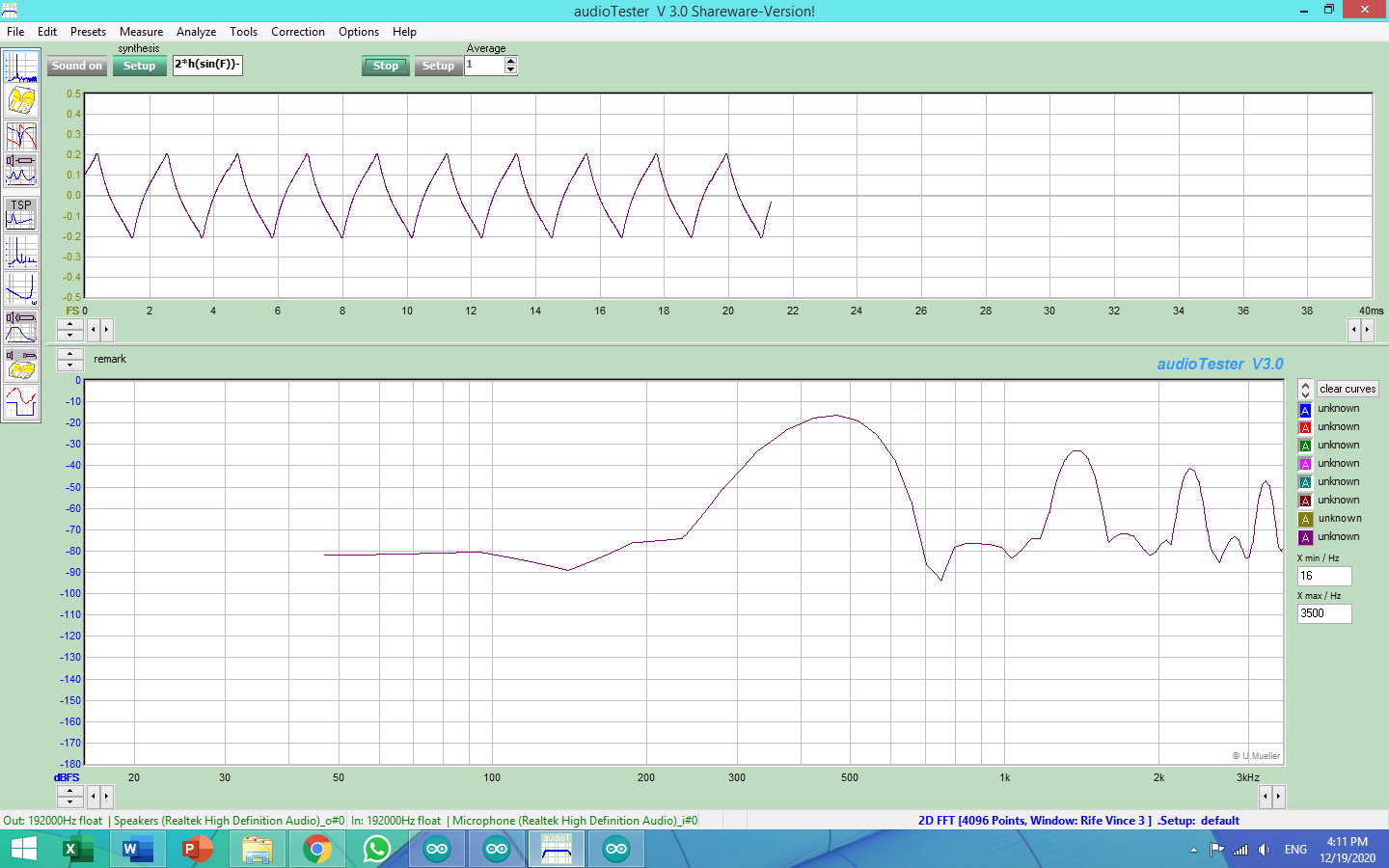
**RESULTADOS OBTENIDOS: SEÑAL TRIANGULAR**

**f=500Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**



11.12 ms

9 ms

**2.12 ms**

1. **Metodología aprendida**
2. **Nuevo método**

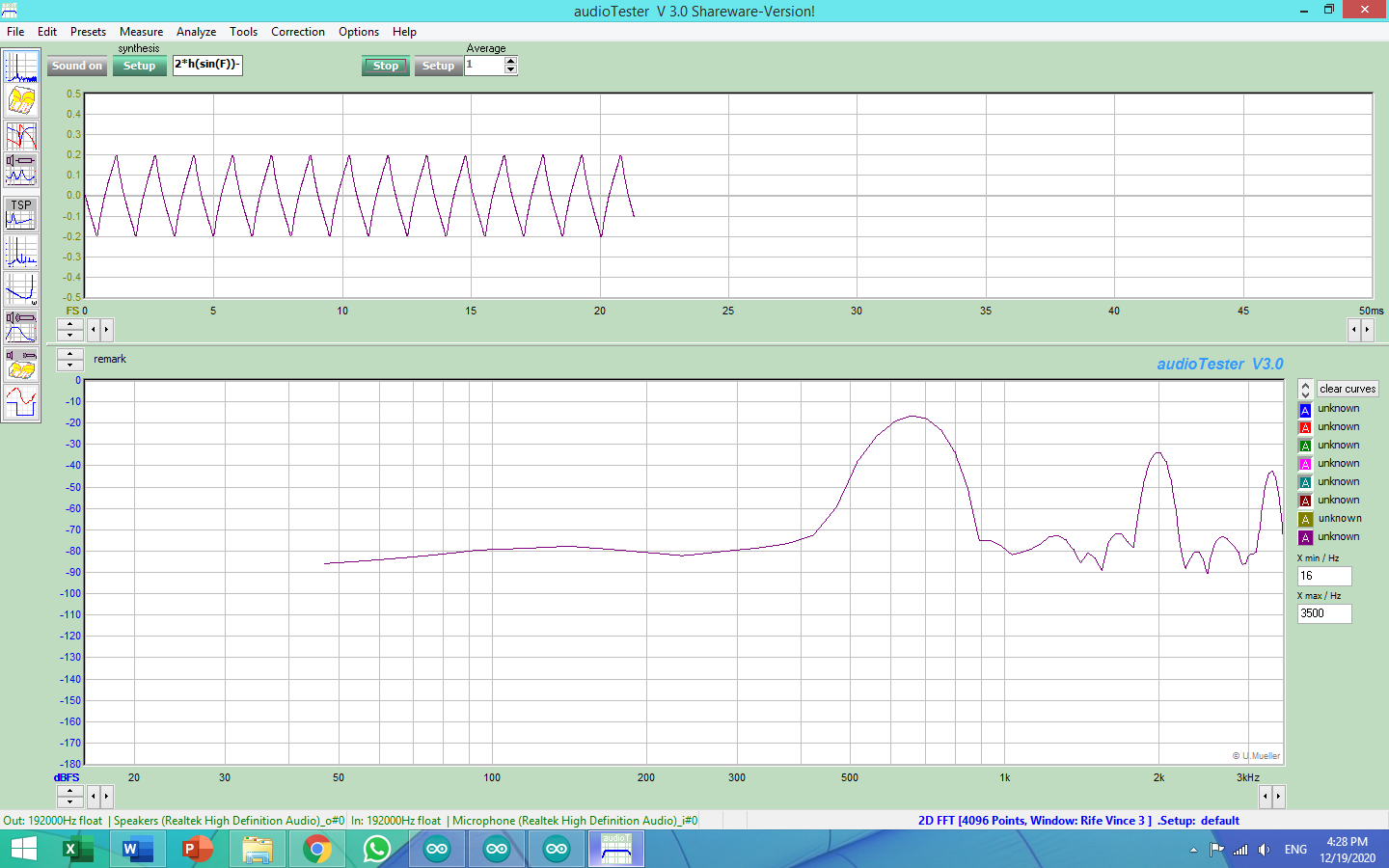
**Fig. 13 Deber Resultado obtenido con función con f=500Hz**

**f=750Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**



8.67 ms

10.13 ms

**1.46 ms**

1. **Metodología aprendida**
2. **Nuevo método**

**F=1000Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**



**1.1 ms**

10.46 ms

9.36 ms

1. **Metodología aprendida**
2. **Nuevo método**

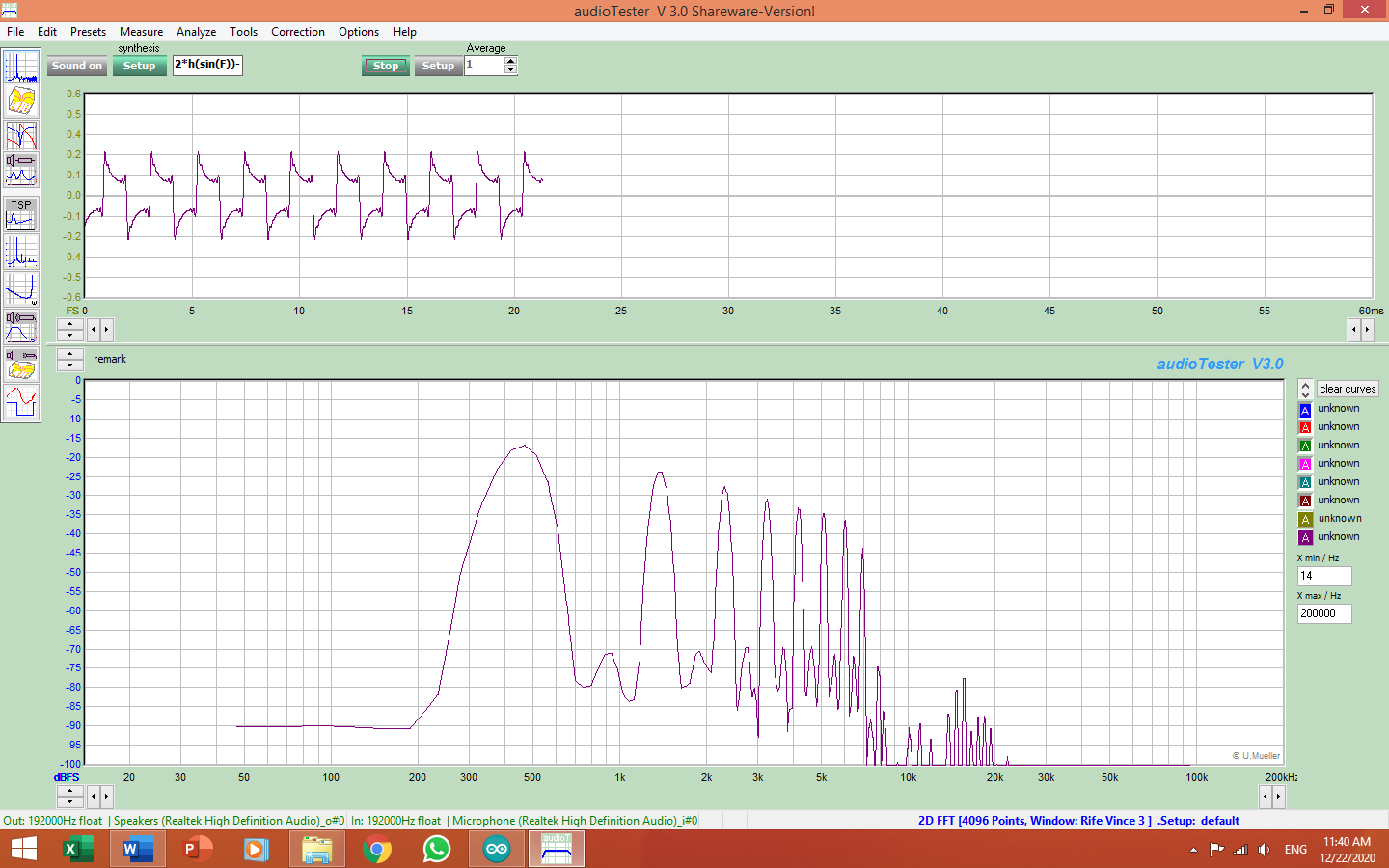
**RESULTADOS OBTENIDOS: SEÑAL CUADRADA**

**f=500Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**



**2.19 ms**

9.15 ms

6.96 ms

1. **Metodología aprendida**



11.99 ms

9.81 ms

**2.18 ms**

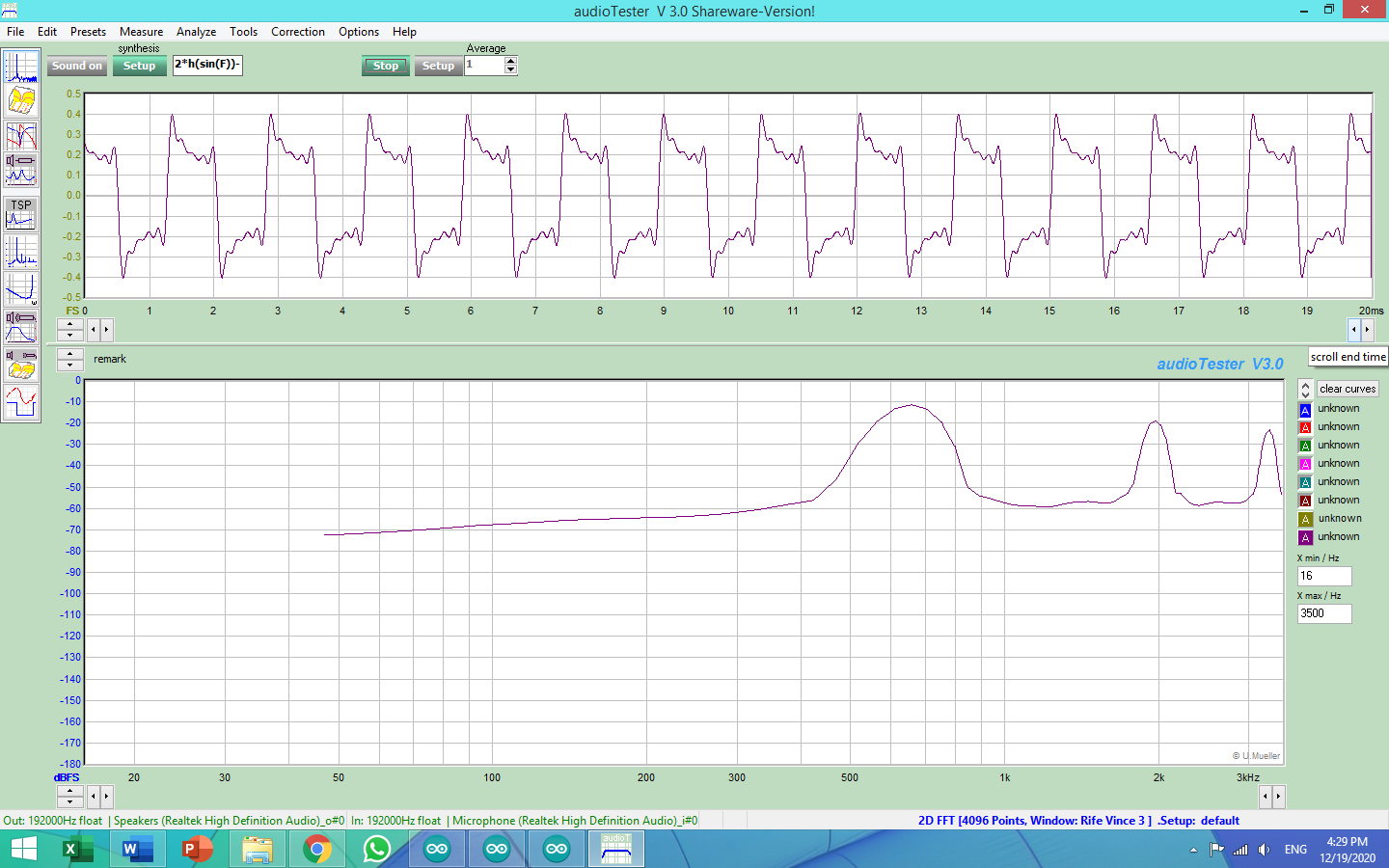
1. **Nuevo método**

**f=750 Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**

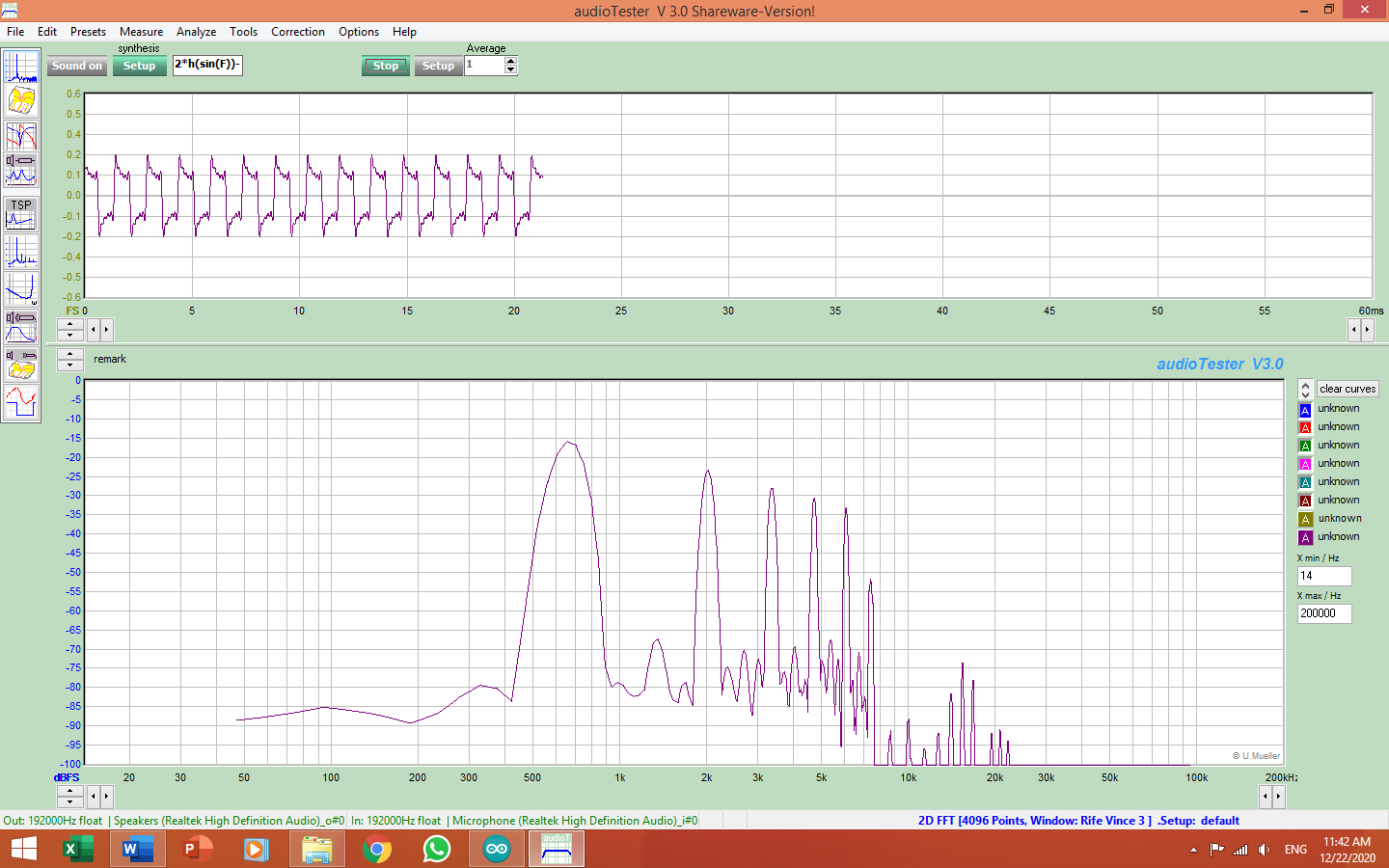


9.93 ms

8.4 ms

**1.53 ms**

1. **Metodología aprendida**



7.5 ms

9.01 ms

**1.51 ms**

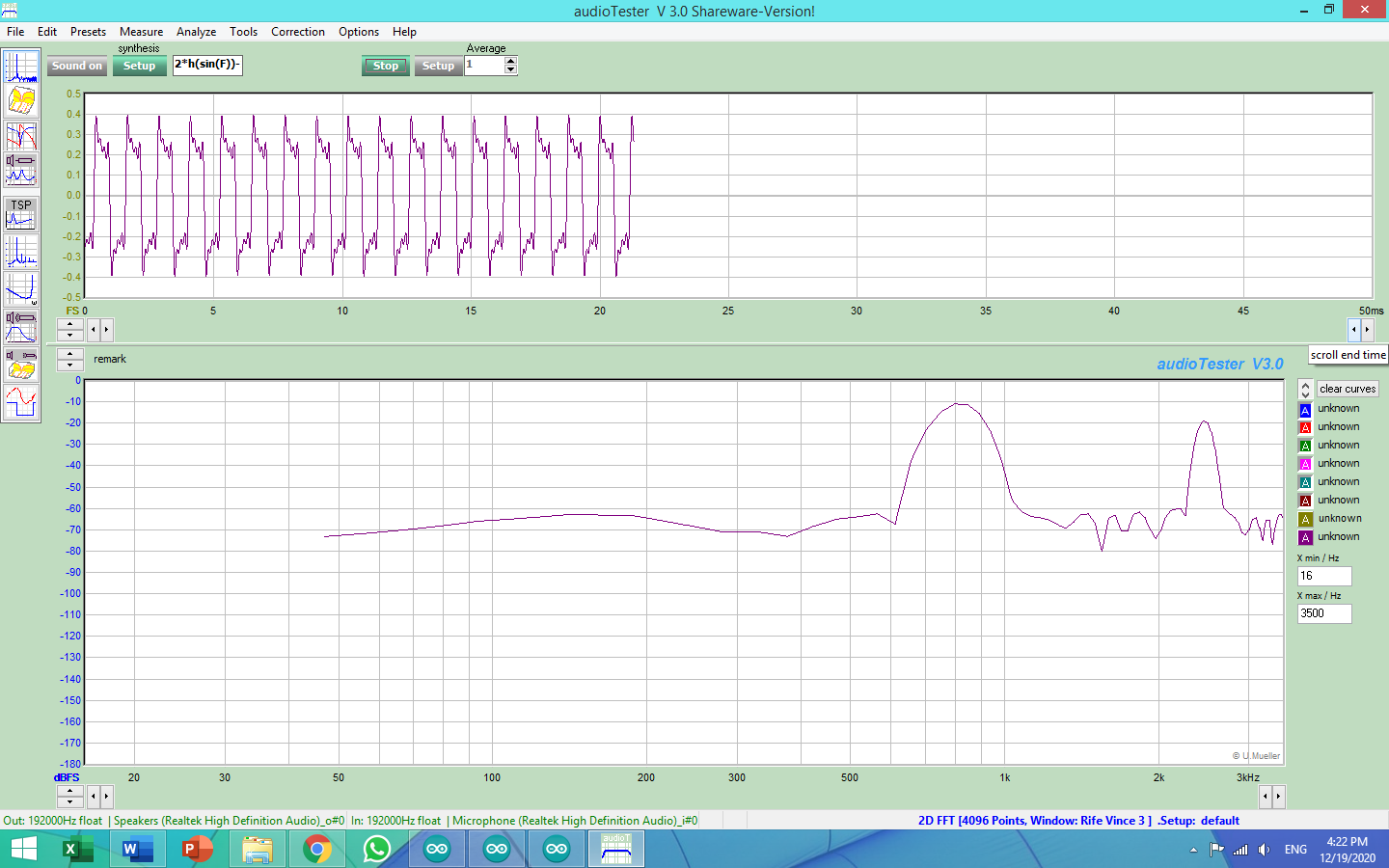
1. **Nuevo método**

**F=1000Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**

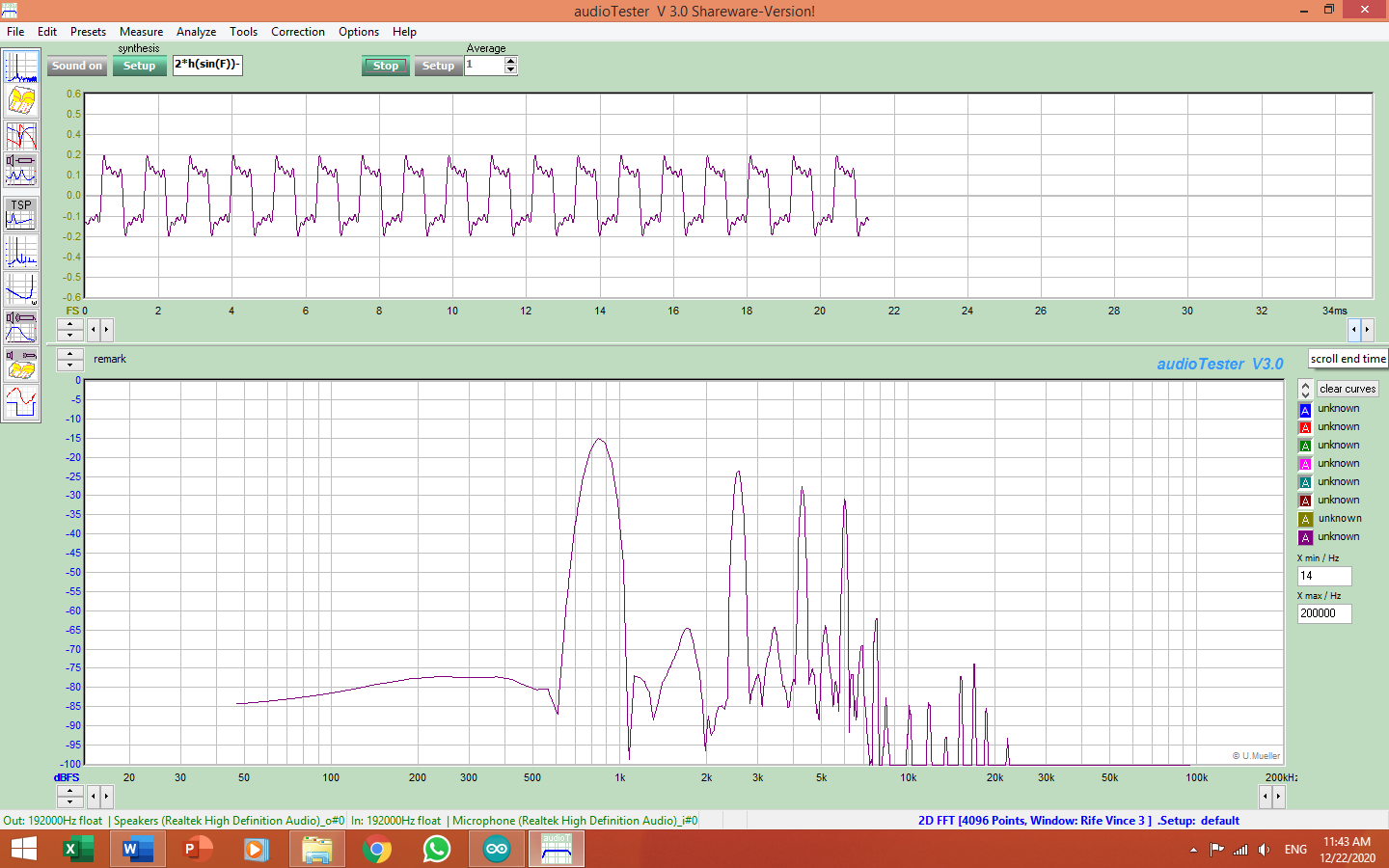


**1.05 ms**

8.71 ms

7.66 ms

1. **Metodología aprendida**



10.73 ms

9.64 ms

**1.09 ms**

1. **Nuevo método**

**Frecuencia Máxima**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**
2. **Metodología aprendida**
3. **Nuevo método**

**RESULTADOS OBTENIDOS: SEÑAL DIENTE DE SIERRA**

**f=500Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**

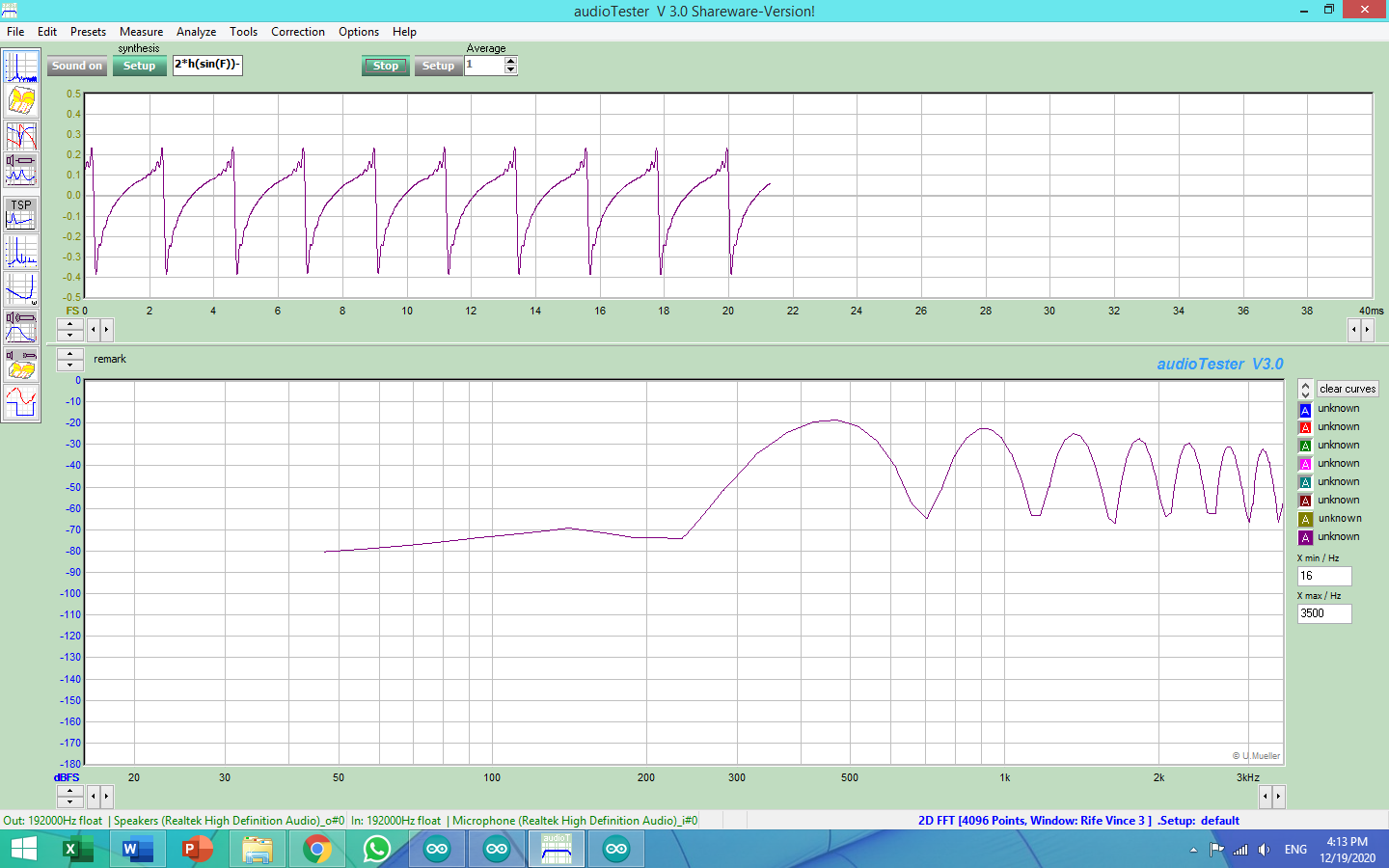


**2.16 ms**

10.71 ms

8.55 ms

1. **Metodología aprendida**



8.01 ms

10.19 ms

**2.18 ms**

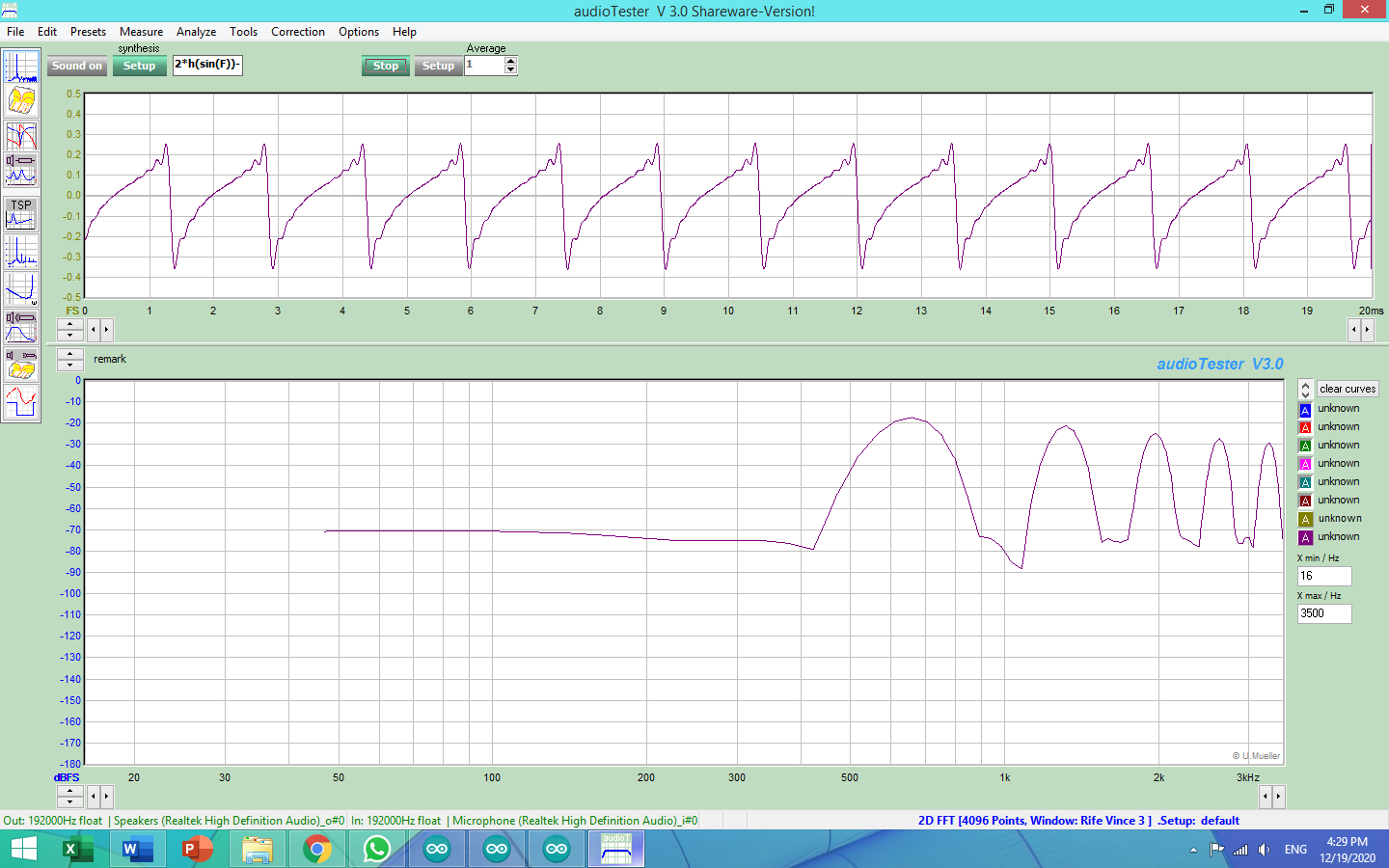
1. **Nuevo método**

**F=750Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**



12.05 ms

10.58 ms

**1.47 ms**

1. **Metodología aprendida**



**1.46 ms**

11.04 ms

9.58 ms

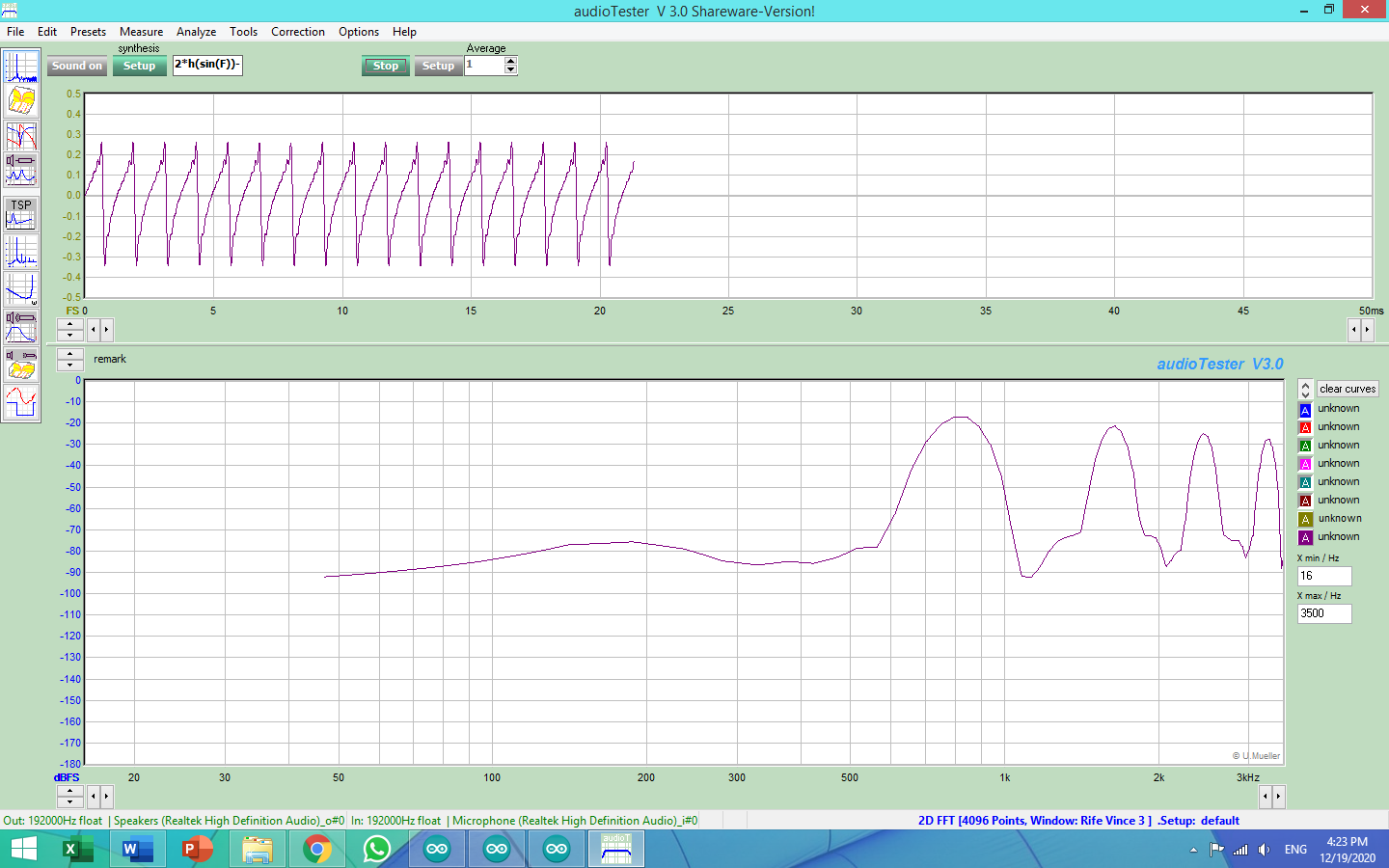
1. **Nuevo método**

**f=1000Hz**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**

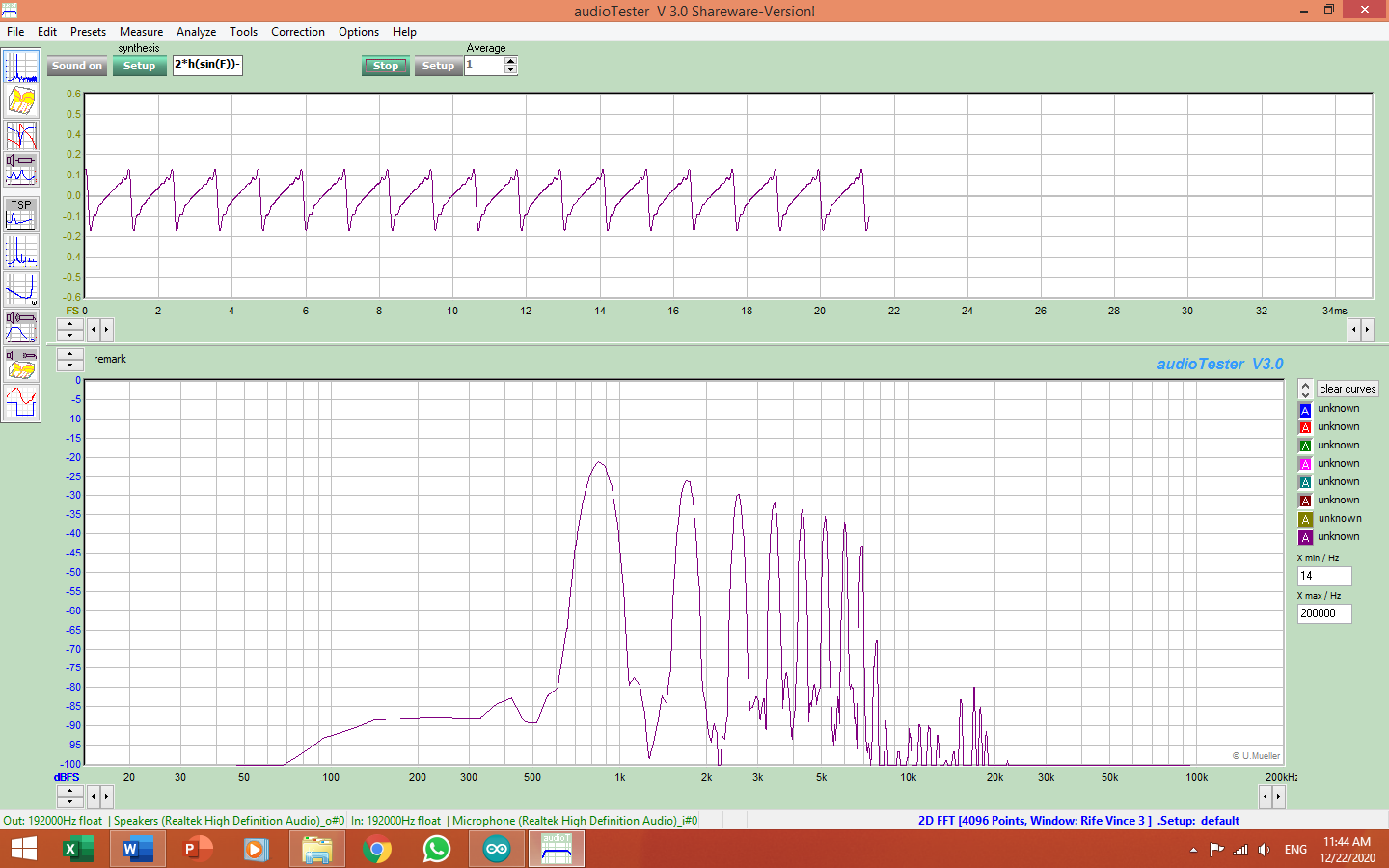


9.06 ms

10.13 ms

**1.07 ms**

1. **Metodología aprendida**



8.73 ms

7.59 ms

**1.14 ms**

1. **Nuevo método**

**Frecuencia Máxima**

1. **Uso de la función**

**No existe una función propia de Arduino que describa a esta señal.**

1. **Librería externa**
2. **Metodología aprendida**
3. **Nuevo método**

**PARTE 2:** **GENERADOR** **SECUENCIAL** CON ARDUINO Y **MEDICIÓN** CON AUDIO TESTER.

El generador secuencial implementado en **un deber anterior en Matlab**, ahora debe ser implementado en Arduino. Los tiempos de duración de cada una de los tipos de onda deben ser datos que se los pone dentro del programa en Arduino. Por ejemplo: tiempo\_onda\_1 = 30 segundos; tiempo\_onda\_2 = 70 segundos; tiempo\_onda\_3 = 100 segundos. De igual manera para las frecuencias: F1 = 0.5Hz, F2= 5 Hz y F3=10 Hz. Tanto la forma de las ondas, sus frecuencias fundamentales y los tiempos deben ser medidos por medio de audio Tester y grabados en un video. Para que el video sea corto pruebe con tiempos bajos de duraciones de las ondas, por ejemplo: tiempo\_onda\_1 = 7 segundos; tiempo\_onda\_2 = 10 segundos; tiempo\_onda\_3 = 5 segundos.

Compare **exactitud** (realice una tabla para cuatro frecuencias diferentes: 0.5Hz, 5 Hz, 10 Hz y la máxima que usted consiga). No se olvide que en la metodología aprendida, la Fmax conseguida dependerá del número de (muestras / período) que se considere. Grabe un video corto con los resultados.

**PARTE 3:** **GENERADOR** **SECUENCIAL** CON ARDUINO, **MEDICIÓN** CON ARDUINO+MATLAB Y **ALMACENAMIENTO** EN ARCHIVO MEDIANTE MATLAB.

Repita la parte 2 pero ahora la verificación de las ondas debe hacerlo desde Matlab (en un GUI) a través del Arduino y no mediante el puerto de audio.

¿Cuál es la frecuencia máxima que puede ser adquirida correctamente desde Matlab?

Realice una discusión de las razones por las que se tiene esta limitación y cómo podría ser mejorado este resultado.

**NOTA**: Los informes deben estar en documentos de Word separados y desarrollados en formato paso a paso con lujo de detalles. Los códigos deben estar comentados y realizados también en partes. Al final debe adjuntar el código completo.