

ESCUELA POLITÉCNICA NACIONAL

INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROCESAMIENTO DIGITAL DE SEÑALES

ALUMNO: Marialejandra Contreras

Melanny Dávila

SEMESTRE: 2020-B

PARALELO: GR1

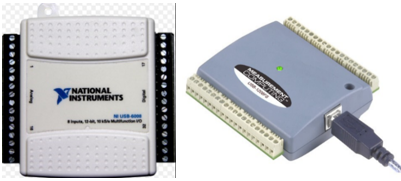
FECHA: 08/12/2020

PROFESOR:

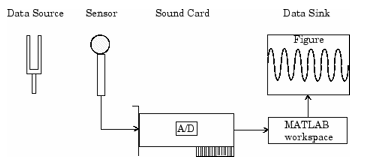
DR. ROBIN ÁLVAREZ

## 3.4 ADQUISICIÓN DE SEÑAL EN TIEMPO REAL MEDIANTE HARDWARE Y EL COMPUTADOR

En general, Matlab puede realizar la **adquisición de señal externa** empleando hardware de varias marcas (advantech, mcc, nidaq, etc) (Figura 3.18) o a través del puerto de audio por medio de un conversor analógico/digital (Figura 3.19).

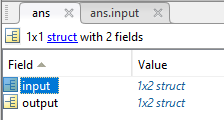


**Figura 3.18** Adquisición de señal mediante hardware de National Instruments (NI) y Measurement Computing (MCC)

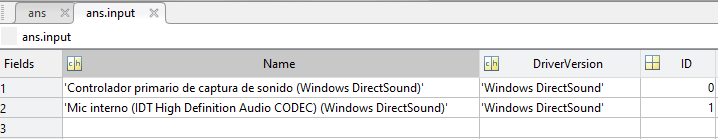


**Figura 3.19**. Adquisición mediante tarjeta de audio

Para saber qué dispositivos de adquisición son reconocidos por Matlab se tiene la función ***audiodevinfo*** que **devuelve un identificador o ID** para cada uno de los dispositivos reconocidos. Al ejecutarla, por ejemplo, si solo se tiene el puerto de audio, se obtendría la variable ANS y al hacer doble clic se obtiene lo siguiente:

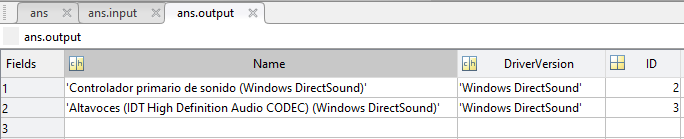


Al hacer doble clic sobre **input**, obtenemos lo siguiente:



Por ejemplo, si se desea usar el micrófono interno del computador, el ID sería el 1.

Al hacer doble clic sobre **output**, obtenemos lo siguiente:



Por ejemplo, si se desea usar los altavoces del computador, el ID sería el 3.

Una vez que se conocen los ID de los dispositivos disponibles, cualquiera de ellos puede ser empleado para adquirir una señal externa mediante el **objeto** **AUDIORECORDER**. La creación de una instancia de este objeto es la siguiente:

*recorderObj = audiorecorder (Fs, nBits, NumChannels, ID)*

Los parámetros de entrada son los siguientes:

***Frecuencia de muestreo (Fs):*** Frecuencia de muestreo en Hercios (Hz), especificada como un escalar numérico. Los valores válidos de la frecuencia de muestreo dependen tanto de las frecuencias de muestreo permitidas por MATLAB como del hardware de audio específico de su sistema. **MATLAB tiene una restricción de 1000 Hz <= Fs <= 384000 Hz**. En particular, los valores típicos admitidos por la mayoría de las tarjetas de sonido son 8000, 11025, 22050, 44100, 48000 y 96000 Hz.

***Bits por muestra (nBits):*** Los valores válidos de nBits dependen del hardware empleado, por ejemplo, 8, 16 o 24 bits/muestra**.**

***Número de canales (NumChannels):*** dependen del hardware empleado, en particular, el puerto de audio soporta la adquisición de 1 canal o 2 señales al mismo tiempo.

***Identificador del dispositivo (ID):*** especificado como un entero según el dispositivo de adquisición a ser empleado.

Una vez creado el objeto, una opción para adquirir la señal externa es definiendo el **tiempo de adquisición** **en segundos** por medio de la instrucción ***recordblocking***:

*recordblocking(recorderObj, tiempo\_adquisicion)*

Con lo cual, no devuelve el control hasta que se completa la grabación.

Transcurrido este tiempo se devuelve el control y ya se puede recuperar los datos muestreados por medio de ***getaudiodata****:* si se empleó un solo canal para adquirir la señal, se recuperará un vector y, si se emplearon dos canales, se recuperará una matriz de dos columnas. Su sintaxis es la siguiente:

*y = getaudiodata(recorder,dataType)*

dichos valores de las muestras pueden ser definidos por dataType que puede ser:

**double**(PREDETERMINADO): valores entre -1 y 1.

**single**: valores entre -1 y 1

**int16**: valores entre -32768 a 32767.

**int8**: valores entre -128 y 127

**uint8**: valores entre 0 a 255.

### PRÁCTICA:

Mediante audio tester generar una señal de 400 Hz y por medio del objeto audiorecorder monitorear el puerto de audio de forma indefinida y **actualizar** la gráfica del tiempo **cada segundo**.

**Resolución**:

clear all;close all;clc;

bloque\_adquisicion=1;

fs=8000;

x=0;

while x==0

% Crear un objeto con parámetros: fs, 16 bits, 1 canal (de los 2 posibles)

y=audiorecorder(fs,16,1);%Creación del objeto para obtención de señal

% Uso del objeto: adquiere senial en bloques de "duracion" en segundos:

recordblocking(y, bloque\_adquisicion);

% GETAUDIODATA: recupera la señal almacenada:

senial=getaudiodata(y, 'single');

L=length(senial);%Obtención de tamaño de señal

t=linspace(0,duracion,L);%creación del vector tiempo

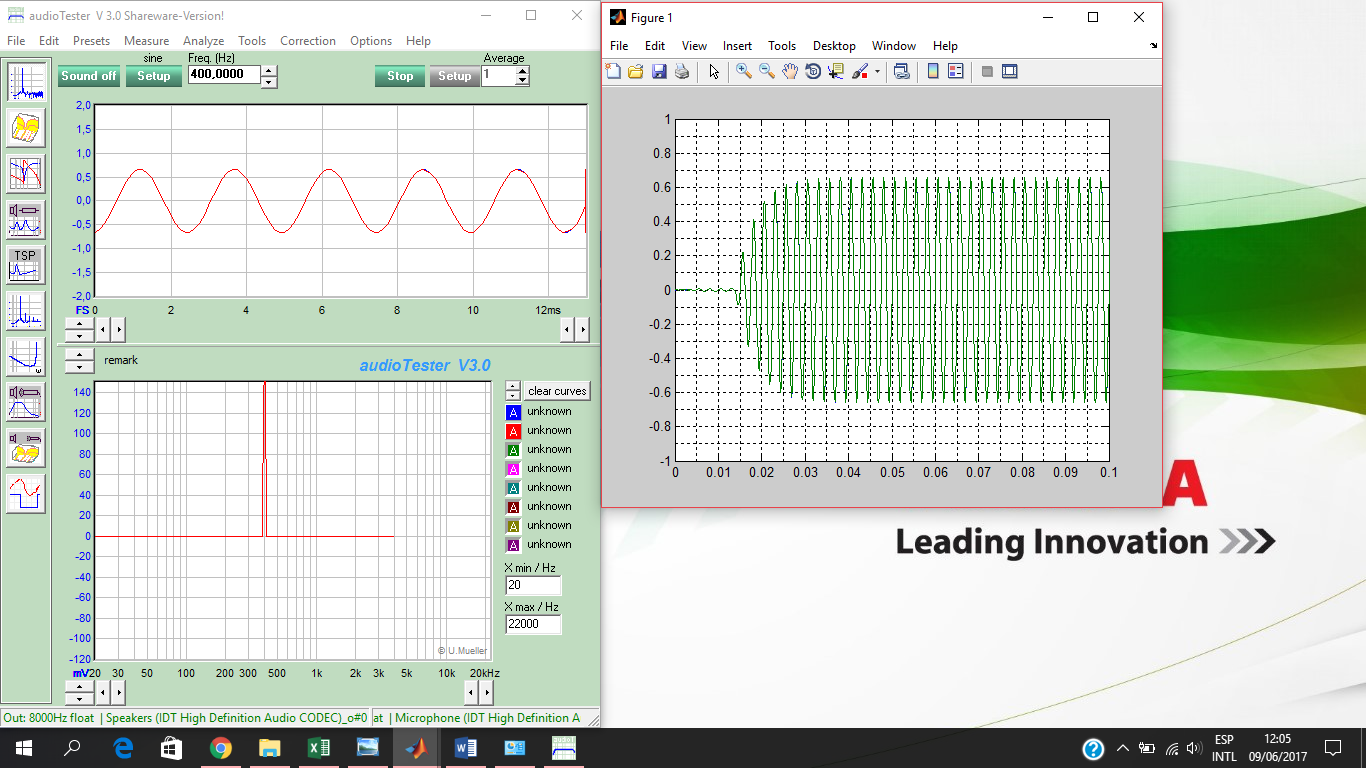
plot(t,senial);grid on; grid minor;

axis([0 1 -1 1])

pause(1)

end

**RESULTADO**: en la Figura 3.20 se ve cómo va cambiando tanto en Audiotester como en Matlab cada vez que el usuario va cambiando la frecuencia (400/500/600/…) en audio tester, sin hacer **sound off**.



**Figura 3.20.** Obtención de la señal de audio en tiempo real en Matlab. La señal es generada mediante el Audio Tester

**EJERCICIO:**

Adquirir una señal cuya frecuencia máxima se sabe que es de 8 KHz. Realizar la adquisición cada 1 segundo durante 10 segundos y mostrar una gráfica acumulativa de la señal. Finalmente grabe la señal completa en un archivo.

**SOLUCIÓN:**

clear all;close all;clc;

duracion=1; tiempo de adquisición de la señal empleado para el recordblocking

fs=24000; frecuencia de muestreo: Fs = 3 \* Fmax

senial=[ ];

while x<=10 % se adquiere durante 10 segundos

y=audiorecorder(fs,16,1);

recordblocking(y,duracion); % se adquiere en bloques de 1 segundo

senial=getaudiodata(y, 'single'); %ver si es fila o columna

L=length(senial);

t=0:1/Fs:L/Fs;

plot(t,senial);

grid minor;

senial=[senial;senial]; % cuidado: ver si es fila o columna

x=x+1;

end

% grabación de la señal total:

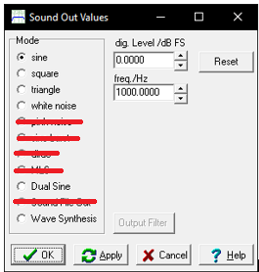
audiowrite('señal.wav', senial, fs);

Esta manera de realizar adquisición tiene la desventaja de que el usuario no tiene control sobre dicha adquisición. Otra manera de hacerlo controladamente, por ejemplo en **ambiente GUI**, es empleando funciones de este objeto como:

* “record”: comienza a grabar la señal.
* “pause”: pausa la grabación.
* “play”: reproduce la grabación.
* “resume”: reinicia la grabación desde un punto pausado.
* “stop”: detiene la grabación.

DEBER :

1. Generar con AUDIO TESTER cada una de la señales vistas en la gráfica siguiente (las que requieren, usted debe poner los parámetros que guste)



1. Realizar una **aplicación GUI** de modo que se tenga control de la adquisición (emplear las funciones anteriores: (record, pause, stop, etc) y mostrarlo en una gráfica dentro del GUI de forma continua (como un osciloscopio real), los datos nuevos siguen a continuación de los anteriores y cuando ya se llene la pantalla, debe seguir desapareciendo la primera parte y apareciendo la parte actual. Al finalizar la adquisición, deberá grabar la señal en un archivo: puede ser (.mat), (.txt), Excel o como usted prefiera.
2. Aplíquelo a la señal obtenida en un deber anterior en el que se tenía un GENERADOR SECUENCIAL donde se generaban tres señales de forma secuencial.
3. Grábelo en un **video** que deberá subir junto con el archivo en Word en el que se **explique** **detalladamente** todo el proceso realizado de modo que cualquier lector pueda seguirlo paso a paso sin problema.

classdef Osciloscopio\_App < matlab.apps.AppBase

% Properties that correspond to app components

properties (Access = public)

figure1 matlab.ui.Figure

graficaGen matlab.ui.control.UIAxes

btnRecord matlab.ui.control.Button

btnPause matlab.ui.control.Button

btnPlay matlab.ui.control.Button

btnResume matlab.ui.control.Button

btnStop matlab.ui.control.Button

btnSalir matlab.ui.control.Button

text2 matlab.ui.control.Label

end

properties (Access = public)

bloque\_adquisicion=1;

fs=8000;

end

% Callbacks that handle component events

methods (Access = private)

% Code that executes after component creation

function Osciloscopio\_OpeningFcn(app, varargin)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app); %#ok<ASGLU>

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to Osciloscopio (see VARARGIN)

% Choose default command line output for Osciloscopio

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

end

% Button pushed function: btnPause

function btnPause\_Callback(app, event)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

global aux;

aux=1;

end

% Button pushed function: btnPlay

function btnPlay\_Callback(app, event)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

global x;

x=2;

global senial;

[senial,app.fs] = audioread('señal.wav');

sound(senial,app.fs);

end

% Button pushed function: btnRecord

function btnRecord\_Callback(app, event)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

global y;

% Crear un objeto con parámetros: fs, 16 bits, 1 canal (de los 2 posibles)

y=audiorecorder(app.fs,16,1);%Creación del objeto para obtención de señal

global x;

x=0;

global senial;

senial=[ ];

global aux;

aux=0;

while x==0

% Uso del objeto: adquiere senial en bloques de "duracion" en segundos:

recordblocking(y, app.bloque\_adquisicion);

% GETAUDIODATA: recupera la señal almacenada:

senial=getaudiodata(y, 'single');

L=length(senial);%Obtención de tamaño de señal

t=linspace(0,app.bloque\_adquisicion,L);%creación del vector tiempo

plot(app.graficaGen, t,senial);

grid on;

axis([0 0.01 -0.2 0.2])

while aux==1

pause(1);

end

end

audiowrite('señal.wav', senial, app.fs);

end

% Button pushed function: btnResume

function btnResume\_Callback(app, event)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

global x;

global aux;

aux=0;

x=0;

end

% Button pushed function: btnSalir

function btnSalir\_Callback(app, event)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

% hObject handle to btnSalir (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

closereq();

end

% Button pushed function: btnStop

function btnStop\_Callback(app, event)

% Create GUIDE-style callback args - Added by Migration Tool

[hObject, eventdata, handles] = convertToGUIDECallbackArguments(app, event); %#ok<ASGLU>

global x;

x=1;

cla;

end

end

% Component initialization

methods (Access = private)

% Create UIFigure and components

function createComponents(app)

% Create figure1 and hide until all components are created

app.figure1 = uifigure('Visible', 'off');

app.figure1.Position = [680 567 601 479];

app.figure1.Name = 'Osciloscopio';

app.figure1.Resize = 'off';

app.figure1.HandleVisibility = 'callback';

app.figure1.Tag = 'figure1';

% Create graficaGen

app.graficaGen = uiaxes(app.figure1);

app.graficaGen.FontSize = 13;

app.graficaGen.NextPlot = 'replace';

app.graficaGen.Tag = 'graficaGen';

app.graficaGen.Position = [54 23 434 367];

% Create btnRecord

app.btnRecord = uibutton(app.figure1, 'push');

app.btnRecord.ButtonPushedFcn = createCallbackFcn(app, @btnRecord\_Callback, true);

app.btnRecord.Tag = 'btnRecord';

app.btnRecord.FontSize = 11;

app.btnRecord.Position = [508 274 59 20];

app.btnRecord.Text = 'Record';

% Create btnPause

app.btnPause = uibutton(app.figure1, 'push');

app.btnPause.ButtonPushedFcn = createCallbackFcn(app, @btnPause\_Callback, true);

app.btnPause.Tag = 'btnPause';

app.btnPause.FontSize = 11;

app.btnPause.Position = [508 227 59 20];

app.btnPause.Text = 'Pause';

% Create btnPlay

app.btnPlay = uibutton(app.figure1, 'push');

app.btnPlay.ButtonPushedFcn = createCallbackFcn(app, @btnPlay\_Callback, true);

app.btnPlay.Tag = 'btnPlay';

app.btnPlay.FontSize = 11;

app.btnPlay.Position = [508 180 59 20];

app.btnPlay.Text = 'Play';

% Create btnResume

app.btnResume = uibutton(app.figure1, 'push');

app.btnResume.ButtonPushedFcn = createCallbackFcn(app, @btnResume\_Callback, true);

app.btnResume.Tag = 'btnResume';

app.btnResume.FontSize = 11;

app.btnResume.Position = [508 132 59 20];

app.btnResume.Text = 'Resume';

% Create btnStop

app.btnStop = uibutton(app.figure1, 'push');

app.btnStop.ButtonPushedFcn = createCallbackFcn(app, @btnStop\_Callback, true);

app.btnStop.Tag = 'btnStop';

app.btnStop.FontSize = 11;

app.btnStop.Position = [509 94 59 20];

app.btnStop.Text = 'Stop';

% Create btnSalir

app.btnSalir = uibutton(app.figure1, 'push');

app.btnSalir.ButtonPushedFcn = createCallbackFcn(app, @btnSalir\_Callback, true);

app.btnSalir.Tag = 'btnSalir';

app.btnSalir.FontSize = 11;

app.btnSalir.Position = [509 51 59 20];

app.btnSalir.Text = 'Salir';

% Create text2

app.text2 = uilabel(app.figure1);

app.text2.Tag = 'text2';

app.text2.HorizontalAlignment = 'center';

app.text2.VerticalAlignment = 'top';

app.text2.FontSize = 27;

app.text2.Position = [159 408 223 37];

app.text2.Text = 'Osciloscopio';

% Show the figure after all components are created

app.figure1.Visible = 'on';

end

end

% App creation and deletion

methods (Access = public)

% Construct app

function app = Osciloscopio\_App(varargin)

% Create UIFigure and components

createComponents(app)

% Register the app with App Designer

registerApp(app, app.figure1)

% Execute the startup function

runStartupFcn(app, @(app)Osciloscopio\_OpeningFcn(app, varargin{:}))

if nargout == 0

clear app

end

end

% Code that executes before app deletion

function delete(app)

% Delete UIFigure when app is deleted

delete(app.figure1)

end

end

end

VIDEOS

https://epnecuador-my.sharepoint.com/:f:/g/personal/marialejandra\_contreras\_epn\_edu\_ec/EnrMh1xMxihIk9sAnvt4drkBDmroUvHgf5XTZduAIZVYlg?e=vhElk8