SUBJECT: MATLAB scripts for use with the 333D01

PURPOSE:

This document provides example MATLAB scripts to capture acceleration data from the 333D01 USB Smart Sensor.

EQUIPMENT & TOOLS REQUIRED:

One or more 333D01 USB Smart Sensors
MATLAB, optionally with the Digital Signal Processing package
ASIO4All (optional, but recommended)
(Download here)
ASIO Control (optional)
(Download here)
Links checked 11/26/2014.
Computer with one or more available USB 2.0 compatible ports

PROCEDURE:

Calibration Decoding Script Examples

The 333D01 has the serial number, calibrated sensitivity, and calibration date programmed into the sensor. This information is provided as part of the model name. The following scripts will decode the information and return stored information.

Note: Since ASIO4ALL hides the 333D01 device name from MATLAB, the decoder will not work when using the ASIO interface.

`DigiDecoder.m` – This is a re-usable function that returns the calibration information programmed into a 333D01. It can be used by your own scripts to simplify the process of obtaining calibrated data in engineering units.

`DigiDecoderDemo.m` – This is a stand-alone example that pops up a dialog box containing the calibration information programmed into the 333D01.

`spectralcalc.m` – This is a re-usable function that returns frequency spectrum information including FFT magnitudes, RMS values, and the window type.

MATLAB via Windows audio:

MATLAB captures sensor data via the audiorecorder object. Using this means that you don't have to use the ASIO control panel, however 24-bit resolution is limited to 44.1 kHz and 48 kHz sample rates. In addition, if you are not careful Windows can do some resampling of the incoming data, causing sampling related errors. Be sure to set the desired sample rate in the Windows Sound -> Recording Devices -> device properties menu before use.

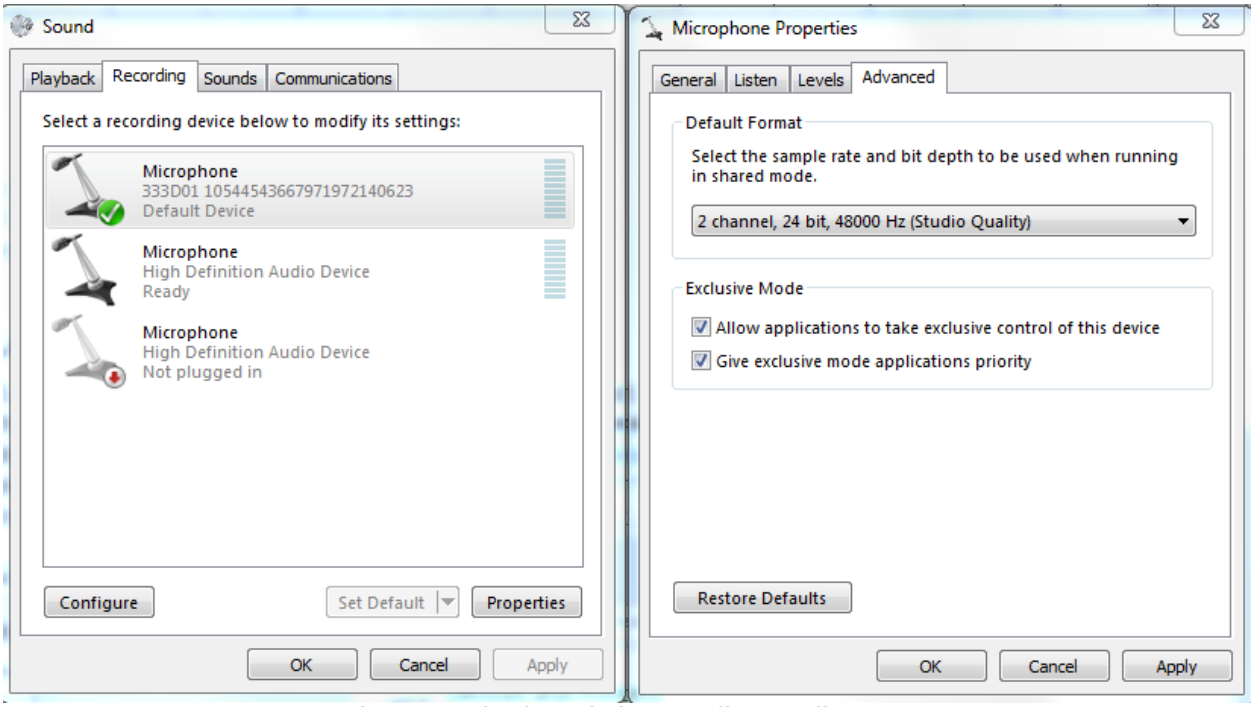

Figure 1- Selecting Windows Audio sampling rate

When using multiple devices, you can select the device that audiorecorder records from by its device ID, which you can check by creating an audiodevinfo structure, and checking each of the input fields in the list. Enumeration starts at 0, which is usually a built-in Windows audio driver. The 333D01 should show up by name; however on occasion there may be an error in loading the name from the device's memory and it may only display as a USB Audio Device. Disconnecting and reconnecting the device may help solve this problem, but it is not critical so long as you recognize it for what it is.

```
   >> a=audiodevinfo
a =
     input: [1x2 struct]
    output: [1x2 struct]
   >> a.input
ans =
1x2 struct array with fields:
    Name
    DriverVersion
    ID
   >> a.input(1)
ans =
              Name: 'Primary Sound Capture Driver (Windows DirectSound)'
    DriverVersion: 'Windows DirectSound'
                ID: 0
   >> a.input(2)
ans =
              Name: 'Microphone (67- TMS 333D) (Windows DirectSound)'
    DriverVersion: 'Windows DirectSound'
                ID: 1
```

Figure 2- Identifying which device ID to use

Once configuration is complete, you can start the script "saacqwfftwc_inmat_winaudio.m". A prompt will appear asking for the desired number of averages, the sample rate **that you selected in the Windows interface**, the block size to use in FFT calculations, and the sensitivity of your sensor. Sensitivity can be found in the calibration sheet provided with the sensor. If you have an old calibration sheet with units of mv/g, you can enter that value and the script will detect and convert it to the proper units. The prompt will appear populated with a set of default values, including default sensitivity. **Note: for accurate results, it is important to replace the default sensitivity values with those from the calibration sheet**.

Once you click OK, the audiorecorder will initialize, and time and frequency data will begin updating on the figure. The bottom subplot shows the built up average from all measurements. The time domain amplitude has units of m/s^2, while the frequency domain plots have units of dB relative to 1 m/s^2.If you are interested in keeping all of the data captured by the script, you can set the variable "`plotAllTime`" to true. This will store all time data captured in the variable `yHist`, as well as create plots after all data capture of the entire time history.

MATLAB via ASIO4ALL and the ASIO Control Active X:

Installing ASIO4ALL and ASIO Control

Click on both of the links at the start of this document to download ASIO4ALL and ASIO Control. Open the ASIO4All installer, and install it with all of the default options. If desired, you can change the installation directory. Next, unzip the ASIOControlSetup.zip file somewhere, and run the Setup.msi. You will need to select the install directory. Once ASIOControl is installed, go back to the unzipped file, and move ASIOControl.m into your MATLAB path. Finally, you will need to register the Active X. The process to do so is outlined in the "ASIO Control User Guide" included in the downloaded .zip file, but the process is simple. Open a command prompt **as administrator** (start -> run ->cmd), change directory to where you installed ASIO Control, for example with a command like: `cd "C:\Program Files (x86)\ASIOControl"` . Next, enter the command: `regsvr32 ASIOControl.ocx` . After completing this, ASIO Control can be used in MATLAB.

Using ASIO Control

ASIO Control provides one method of utilizing the ASIO driver in vanilla MATLAB. ASIO Control requires the "ASIOControl.m" file to be in MATLAB's path. To use this method, you must first configure your device(s) in the "ASIO Control" control panel, labelled "ASIO Recorder Control". Note: this is the MATLAB window, not the system tray icon that pops up. To open the ASIO Control window, start the script "saacqwfftwc_inmat_asiocontrol.m". The window will not move to the front if it is already open, so if the window does not appear, check the other open MATLAB windows. Key settings that require configuration in this panel, accessed by selecting ASIO -> Properties in the figure, are the sample rates and the inputs. The ASIO Recorder Control Panel also provides a button to access the ASIO4ALL control panel. The settings you select in the ASIO Recorder Control panel are highlighted in blue. Note that for sample rates, 64000, 88200, 96000, and 192000 are invalid sample rates for the 333D01.

The ASIOALL settings can be left at their default values. For help with ASIO4ALL, check the ASIO4ALL v2 Instruction Manual that was installed with ASIO4ALL.

Once configuration is complete, you can continue the script. A prompt will appear asking for the desired number of averages, the sample rate **that you selected in the ASIO Recording Control panel**, the block size to use in FFT calculations, and the sensitivity of your sensor. Sensitivity can be found in the calibration sheet provided with the sensor. Once you click OK, ASIO4ALL will initialize, and time and frequency data will begin updating on the figure. The bottom subplot shows the built up average from all measurements. The time domain amplitude has units of m/s^2, while the frequency domain plots have units of dB relative to 1 m/s^2.

MATLAB via ASIO4ALL and DSP toolbox:

Installing ASIO4ALL

Click on the link for ASIO4ALL at the start of this document to download ASIO4ALL. Open the ASIO4All installer, and install it with all of the default options. If desired, you can change the installation directory.

Using ASIO4ALL

MATLAB can directly utilize ASIO with the DSP toolbox, if it is available to your installation of MATLAB. To use ASIO with the DSP toolbox, you must go into MATLAB's preferences, select the DSP System Toolbox tab, and select ASIO as the audio hardware API. Once this is complete, all you need to do is run the script "saacqwfftwc_inmat_dspasio.m". A prompt will appear asking you for the number of averages you want, your desired sample rate, the FFT block size, and the sensitivity of both channel A and B for your sensor. Sensitivity can be found in the calibration sheet provided with the sensor. Note that for this method, the sample rate you enter in this prompt will be the actual sample rate used, compared to the previous two methods that required you to match the sample rate chosen elsewhere.

Keep in mind the 333D01 only supports sample rates of 8kHz, 11.025kHz, 16kHz, 22.05kHz, 32kHz, 44.1kHz, and 48kHz.

The next step is to ensure that ASIO4ALL is properly configured. Usually, for one device this is simple.

Click on the blue power icons  to toggle devices other than the 333D01 off (dark icon). You can adjust the ASIO Buffer Size slider to the left to reduce latency, though too small may result in data dropout. The default value of 512 should work fine.

After dismissing the messagebox, the script will begin capturing data. The default behavior of the script is to keep the data stream open for continuous capture, adding buffered data to the queue. If you experience buffer overrun messages, you can change the value of the variable `continuousMode` to false. This will release the device between captures while the spectral calculations are made, preventing the buffer from being overrun. However, this means that data from the sensor is not captured in the meantime. The bottom subplot shows the built up average from all measurements. The time domain amplitude has units of m/s^2, while the frequency domain plots have units of dB relative to 1 m/s^2.

ASIO4ALL notes:

These are the various ASIO4ALL icons. 

Clicking one of these will take you to the ASIO4ALL control panel. Hovering the mouse over the icons will display the currently used buffer size and sample rate.

If the 333D01 displays an unusual state in the ASIO4ALL control panel, try closing the panel, unplugging and replugging the device, and then restarting MATLAB. Refer to the ASIO4ALL v2 Instruction Manual included with the installation of ASIO4ALL for further ASIO4ALL troubleshooting.

MATLAB multi-device via ASIO4ALL and the DSP toolbox

It is possible to simultaneously capture data from more than one 333D01 at once with MATLAB. However, the data is not guaranteed to be synchronous due to USB framing becoming unsynchronized. This causes a ~1ms delay between sensor time waveforms. Capture from more than 2 devices will likely lead to a buffer overrun. To compensate, you can disable `continuousMode`, which will cause data capture to pause while the FFT and spectral calculations run. This has been tested to work with as many as 12 sensors at once, and likely more; though it will probably mean an increasing amount of time between plot updates. Alternatively, the spectral portion of the code can be commented out, and continuous time waveform capture and graphing can be used for more than 2 devices.

“saacqwfftwc_inmat_dspasio_multidevice.m” operates in the same manner as its single device version, with the exception that an extra field is added to the prompt displayed to enter the number of devices you want to use. Be sure to enable that many devices in the ASIO4ALL control panel, disabling other devices. You may need to restart MATLAB after enabling all the devices, if an error about incorrect number of channels appears.

MATLAB Digiducer Data Analyzer:

Digiducer_Data_Analyzer.m is a graphical user interface that extracts the calibration information from the 333D01 sensor and plots the acquired data. The GUI is able to analyze both WAV files and live data.

The WAV analysis portion can extract the embedded calibration information from the WAV file format and display the data scaled to engineering units.

The live data analysis can also extract the embedded calibration information from the 333D01 sensor and display the data scaled to engineering units as well.

The script uses ASIO4ALL to recognize the sensor as an audio device and the DSP AudioRecorder function to attain the live data. Based off user input settings, the GUI proceeds to display time data, the average and instantaneous frequency spectrums, and the calibration information embedded in the sensor.

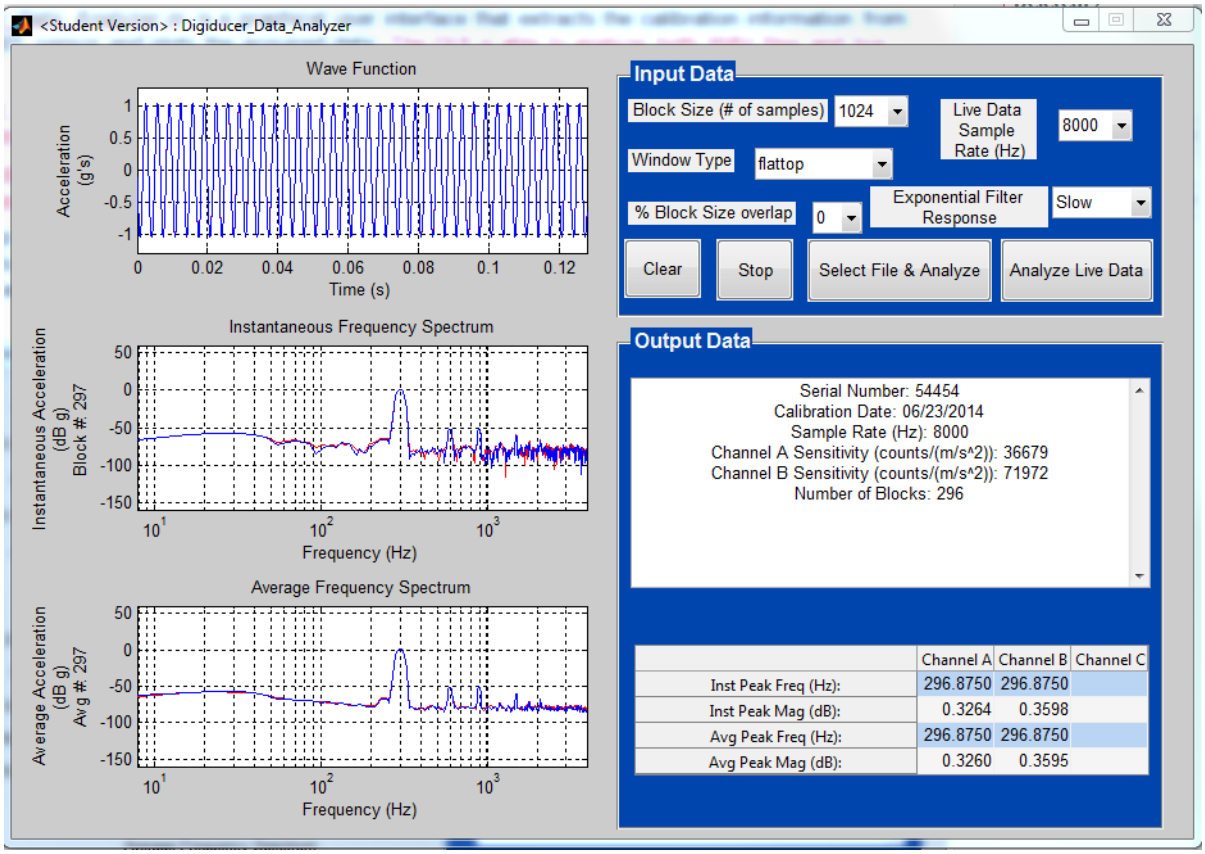

Figure 3- Digiducer_Data_Analyzer.m GUI

General Notes:

Each capture mode contains a setting, "`plotAllTime`", which when toggled accumulates all of the captured time history into a variable. This setting is disabled by default, but you can change the value in the script. The setting will cause a plot of Channel A and a plot of Channel B for all time to be generated. If using the multi-device script, the channel A's of all devices and channel B's of all devices will be grouped together in separate plots.