DAQ 3: Introduction to Data Acquisition - Synchronous I/O

3.1 Introduction

In the DAQ laboratories so far, use have used MATLAB to set and read both digital voltages and analog voltages. In the first DAQ labs, exact timing has not been an issue - none of the values were affected by the rate at which data was taken. For this lab, you will be examining audio, and in this case it is critical that the samples be taken and replayed at a specific rate. The focus, therefore, will be on obtaining and setting data by establishing a sampling rate and then using the card to either measure or deliver voltages at that rate. By the end of the lab, you will have a program that reads a five-second sound clip at the sample rate of your choosing as well as a program that will both play the clip and blink lights in time with the music.

3.2 Resources

The additional resources required for this assignment include:

- Books: None
- Pratt Pundit Pages: `MATLAB:CB-68LP Pinout`, `http://pundit.pratt.duke.edu/wiki/EGR 53/DAQ Audio 1`
- Lab Manual Appendices: None

3.3 Getting Started

1. Each lab group should have one person sit at the PC that has the patch of tape (blue, green, or red) on it. This is **not** the same as the strip with the computer's name on it - it will literally be a small patch of tape all by itself.

2. Set the machine to "Log on to" `...(this computer)` and use the "User name" `mrglocal` and the "Password" `p1p2dell` (that is p-ONE-p-TWO-d-e-l-l). The other lab partner can log into the other computer either using the `mrglocal` account or any other valid NET ID (as log as the "Log on to" is set to `Kerberos`). You do *not* need to run X-Win or PuTTY at all on either computer.

3. Start MATLAB on the PC with the tape on it. The first time MATLAB is run, it may take some time to start up.

4. Once MATLAB starts, make sure the `Current Directory` listed at the top of the MATLAB window is

   ```
   C:\Documents and Settings\labuser\My Documents\MATLAB
   ```

5. In the `Current Directory` window on the left, select and delete any files in the `MATLAB` directory.

3.4 Equipment

The DAQ lab this week requires several different pieces of equipment, most of which have been used in prior DAQs. The latter includes:

- *DAQ (Data Acquisition) Board*
- *Breadboard*
- *LEDs* - you can choose any two of the LEDs in your parts box.
- *Resistors* - this week you will use one 470 Ω or 680 Ω resistor to limit the current to the LEDs from the output channel of the DAQ (we will use `DAC1OUT`).
- *Wires* - you will need three red wires, two black wires, and two green wires.

The pieces of equipment that are new this week are:

- *Signal Generator:* This device creates an electrical signal with frequencies in the range of human hearing. When the signal from the generator is applied to a speaker, you can hear it if the frequency is within the range of human hearing and the power of the signal is high enough to drive the speaker to create a strong enough pressure disturbance. For this lab, the signal generator will be an audio device (such as an iPod or CD player), so the signals generated should be audible if transmitted with sufficient power. If you have a CD or a memory stick with music on it, you can play that information through the other PC on the desk (aka "The one without the tape"). If this is the case, be sure to let a TA know so that the system can be wired accordingly.
- *Headphones:* Just everyday headphones. You will use these to convert the electrical impulses from the signal generator or from MATLAB into an signal that you can hear. For this lab, you will be using the two headphone jacks on the left side of the speaker bar attached under the monitor.

The audio path for this lab is as follows:

- The signal generator (iPod, CD player, PC, etc) will be connected to an extension cable that has a 1/8" stereo plug on either end.
- The other end of the stereo plug is connected to the "line in" jack of the computer running the DAQ. If your PC is asking you what to do about the audio plug in the back of the computer, pick "line in."
- At the front of this computer, there is an audio splitter that takes the voltage from the sound card and sends it to two outputs. These are split as follows:
  - One port of the splitter is connected to the speaker bar. This is so you can hear the sound generated by either your audio source or the PC with MATLAB through these speakers or through headphones connected to these speakers. If you have three people in your group, let a TA know so the signal can be further split.
  - The other port of the splitter is connected to a cable that has a 1/8" plug on one end (connected to the audio splitter) and red and black alligator clips on the other end. These clips will make it easier to measure voltages with the DAQ since they can clip onto the ends of regular wires, which can in turn be connected to the breadboard.

3.5 Scripts

For this week, the scripts you need have already been written. You will be making minor modifications to them in order to understand how the DAQ board functions. The files are at:

`http://www.duke.edu/~mrg/DAQS/DAQaudio1/`

and are called `MirrorInDone.m` and `MirrorOutDone.m`. Open a web browser (IE), type in the URL above, and then right click on the file you want. Choose `Save Target As...` go the the Desktop, then into the

```
C:\Documents and Settings\mrglocal\My Documents\MATLAB
```

directory and save the file there. Do this for all the files. Copies of these files are at the end of this handout so you can take notes regarding how they work.

3.6 Analog Input: Teaching The Computer To Listen

For this section, you will be using MATLAB's Data Acquisition software in concert (hah) with the National Instruments DAQ card to record electrical voltages created by some audio source. You will be capturing your own music, so be sure that it is something you would be willing to let your lab partner know that you own. Also, make sure you know who wrote the song, what album it is on, and what its total length is.

3.6.1 Connecting the Signal Generator

Connect the headphones to the two ports on the left side of the speaker bar, then turn the speaker bar on to its lowest setting. Plug the free end of the extension cable from the back of the PC running MATLAB into your audio source. If you are using a CD or a piece of music stored on a memory stick, and thus playing the music on the other computer, plug the cable into the headphone jack of that other computer.

Next, check the volume settings on the computer running MATLAB. Instructions for the different kinds of computers are available on the Pundit page for this lab.

Once that is set, assemble the connections given in Table 3.2 on page DAQ 3 – 11. Once done with that, if you play your music, you should hear it through your headphones. You can adjust the volume two places - the audio device itself and the knob on the right of the speaker bar. Set the audio so that you can hear the music, but it is not blaring. You should probably turn the audio device to just about full volume, then adjust the knob on the speaker bar.

3.6.2 Leveling the Input

In order to check your circuit and to make sure you are not producing voltages beyond what MATLAB is capable of handling, you are going to use the data acquisition toolbox's built in oscilloscope, called `softscope`, to view your signal. This way, you can set the volume level on your audio device to obtain a reasonable signal before actually running software to capture the voltages.

Double check your circuit connections, then type `softscope` in the MATLAB command window to bring up the built-in oscilloscope program. Note that if there are some initialization errors, you are most likely still running this correctly. Some computers have hardware that causes these false problems to show up. Others will start the oscilloscope without any problems. Just click right through the errors that MATLAB gives you (generally four of them, if any) and you will eventually get to the `Hardware Configuration` page.

Hardware Configuration

First, make sure that the scope is measuring signals from the `nidaq` adapter with ID 1. Set the sample rate to 1000 Hz meaning the card will measure 1000 voltages per second on each open channel. Next, set the card to `Differential mode`, which means the card will be measuring voltage differences between two channels rather than absolute voltages as measured from ground. This halves the number of voltages to be measured but is generally safer than trying to find common ground.

Channel Selection and Settings

`Unselect All` channels, then go back and turn on channels 1 and 2. Even though you are not using channel 2, there is a glitch with the oscilloscope settings that can only be overcome if there are two or more channels being measured.

We will want to measure voltages that are between -0.4 V and 0.4 V, so the most precise range for measurement is [-0.5 0.5]. Go ahead and change the range for channel 1. Note that setting the range too high would result in a less precise measurement, while setting the range too low would result in "clipping." When clipped, voltages outside the range are recorded as equal to the maximum value within the range. In this case, any voltages with magnitudes greater than 0.5 V will show up as voltages with magnitudes of 0.5 V. Once the settings are in, accept them by hitting the OK button.

Oscilloscope Settings

Now that the oscilloscope screen is open, maximize the window so the oscilloscope fills up the entire screen. Next, click the `Trigger` button and start playing a song. You should pick a section of a song that has both music and vocals - you will end up recording a 5 second clip a few times, so make sure the music and vocals last between 10 and 15 seconds. You should also make sure to note which part of the song you are using to calibrate the system so that you use the same segment or close to it each time.

Right-click in the middle of the scope screen and select `Autoscale`. You should now see a wave on the screen - if not, turn the volume up on your device or CD player software and reduce the volume on the speakers as necessary. Select `Autoscale` again. If the maximum output from your device does not produce a signal, ask a TA or instructor of help.

To help set the volume level of your audio device properly, you will want to use the horizontal and vertical scales of the oscilloscope. The vertical scale is given at the lower left of the screen. This measurement is reporting the voltage difference between the horizontal lines on the oscilloscope. The horizontal scale is given at the lower right of the screen. This measurement is giving the amount of time between the vertical lines on the screen. There are two ways to change these scales. The first is to use the virtual knobs to the right of the screen - you can "grab" one by pointing to the small divot on the knob and using the mouse to "spin" the dial. Be sure to grab the Scale dial and not the Offset dial. Go ahead and try to set the values such that there is 0.05 V/div vertically and 100 ms/div horizontally. For the specific monitors and resolutions in the lab, when the `softscope` window is maximized there are 30 vertical divisions and 35 horizontal divisions. This means that the maximum voltages that will show up on the screen are ± 0.75 V tall (15 vertical divisions from 0 V x 0.05 V/div) and the time across the screen is 3.5 sec (35 divisions total x 100 ms/div). Note that setting these values with the dials can be tedious.

Numerical Scope and Channel Settings

A more precise way to set these values is to use the `Edit` command at the top left of the screen. The horizontal, or time, scale is in the `Scope...` window under the `Scope Properties` page. Make sure `display1` is selected, then click on the value in the `HorizontalScale` property and change this to 0.100. Click `OK` to get out of this window. Next, go back to the `Edit` command and pick the `Channel...` window. Under `Channel Properties`, make sure `CH1` is selected, then click on the value in the `VerticalScale` property and change this to 0.05. Click `OK` to get out of this window.

Volume Adjustment

Using the volume control *on your device or music player software* adjust the voltage coming from your device so that the peaks are just under eight vertical divisions away from the center line. This means that your audio source is producing voltages between -0.4 V and 0.4 V. While your audio device is likely capable of producing somewhat larger values, this is a good range so as to not overload your device. Furthermore, as you saw when starting the oscilloscope, there are different input ranges for the DAQ card. We will be using the same -0.5 V to 0.5 V range for the entire experiment. Therefore, limiting the output of your device to ± 0.4 V will make sure that you do not overload the DAQ card, either.

Overloading the card will result in *clipping*, which will cause problems with the sound reproduction later. You can always adjust the *speaker* volume to make the sound louder or quieter. After the leveling is complete, **do not** change the volume level on your device, CD player software, or PC with MATLAB. Any of those will change the voltage seen by the DAQ card, while the speaker knob on the side of the speaker bar will *only* change the volume at the headphones.

Once you are sure your DAQ board is working and have leveled the output from the audio device or CD software, **close the window with the oscilloscope**. You have to get rid of the oscilloscope now because the DAQ system cannot allow two different programs to actively engage the card at a given time.

3.6.3 Acquiring the Signal

Now that **the oscilloscope is closed**, you can open the `MirrorInDone.m` program you retrieved earlier. A text copy is included in ¶3.11.2 on page DAQ 3 – 12. The following bullets describe in detail what each line of the program will do. Make sure to read the descriptions of what each line of code is doing - some are reminders from previous labs but several are now to this one given that we will either be taking an array of measurements or preloading the card with an array of voltages to be sent to the output at a particular rate.

- `delete(daqfind)`

  This line will locate any data acquisition objects currently active and remove them. This is a good line to have at the start of any data acquisition program so that the computer is truly starting from scratch.

- `AI=analoginput('nidaq',1)`

  This creates a handle for the National Instruments Data Acquisition board (or `nidaq`) called `AI`. A handle is merely an identifier that is associated with a certain object. The handle is used in order to access the properties of the object so that they can be modified or viewed. For example, in writing formatted text to a file, we use the `fid=fopen(FileName, 'w')` command to create a handle `fid` to whatever file we want to open. The same concept is at work here.

  Whenever we send information to the board, this `AI` handle is used. You can type `get(AI)` or `set(AI)` to see some of the properties.

- `addchannel(AI, 1)`

  This opens input channel 1 as the first channel from the board. To open multiple channels, simply replace the 1 above with a vector of the channels you wish to open (i.e. `addchannel(AI, [0 6 2])`, which would make channel 0 the first channel, channel 6 the second channel, and channel 2 the third channel).

- `samplerate=44500`

  This sets the variable `samplerate` equal to 44500. We will later use this as the number of samples per second. Note that the input sample rate can go as high as 200000, though the output rate cannot exceed 1000. The sound card in the system can achieve higher output rates than the DAQ card, and for this program, we will use the sound card to produce the output. In the next section, where we use the DAQ card to produce an output in time with the music, we will need to adjust the data and the sampling rate appropriately.

- `duration=5`

  This sets the variable `duration` equal to 5. We will later use this as the number of seconds for the board to collect data.

- `samples=duration*samplerate`

  This sets the variable `samples` equal to the duration multiplied by the sample rate (which, therefore, yields 222500 total samples in this case). This variable will later be used as the total number of samples that will be taken over the entire duration.

- `set(AI, 'SampleRate', samplerate)`

  This sets the property `SampleRate` of the object `AI` (which we have defined as the input from the DAQ board) to a value equal to the value of the variable `samplerate`. This is the line that actually tells the card how fast it should take measurements.

- `set(AI, 'TriggerType','Manual')`

  This sets the property `TriggerType` of the object `AI` to `Manual`, which means MATLAB expects us to tell it exactly when to start taking data rather than automatically starting to collect data when the connection to the board is made.

- `set(AI, 'SamplesPerTrigger', samples)`

  This sets the property `SamplesPerTrigger`, or the total number of data samples taken each time the board is triggered, of the object `AI` to the value of the variable `samples`.

- ```
  set(AI.Channel(1), 'InputRange',  [-0.5 0.5])
  set(AI.Channel(1), 'SensorRange', [-0.5 0.5])
  set(AI.Channel(1), 'UnitsRange',  [-0.5 0.5])
  ```

  These lines set the three range properties for the first channel of the `AI` object (in our case, channel 1) to go from -0.5 to 0.5 V. The default is to go from -5 to 5 V, which would not take precise measurements for the particular range of values you will be taking. Because we set the maximum amplitude to 0.4 V previously, the range of ± 0.5 V is as precise as we can get for our sounds.

- ```
  AI
  start(AI);
  fprintf('Press return to take data\n');
  pause
  trigger(AI);
  fprintf('Taking data...\n')
  ```

  The `AI` by itself will show the information about the `AI` object on the screen, while the `start(AI)` will activate the board. After the user hits return, the `trigger` command tells the board to begin taking data from the open channels using the previously specified sample rate and number of samples.

- ```
  [data, time]=getdata(AI)
  ```

  The `getdata` function reads the data as well as the time information from the board. The matrix `data` contains all the measured voltages from each open channel, with each column representing the readings for one channel. For this experiment, there is only one open channel so there will only be one column.

  The matrix `time` represents the corresponding times when each data point was taken. Note that whenever you take voltage data from the card, you must also take the time data. Even if you do not use it, you need to tell MATLAB to store it somewhere.

- ```
  plot(time, data)
  ```

  This line simply plots the data you obtained on the screen as a function of time. You can now see the voltage that the DAQ recorded for your sound. From this plot, you should be able to see where the sound was loudest and where it was quietest. You cannot, however, determine any sense of the pitch of the sound - the signal is simply oscillating too fast to get any useful frequency information out of this plot. For example - for this clip, there are 222500 samples to plot on the screen, and not nearly that many pixels on the screen to show them individually!

- ```
  fprintf('Press return to play sound\n');
  pause
  sound(data/max(abs(data)), samplerate)
  fprintf('Playing sound...\n');
  ```

  This code will wait until the user has had a chance to pause playback from the original source. After the user hits return, MATLAB will play the sound using its own `sound` command and the sampling rate specified in the second argument, `samplerate`. Note that MATLAB's `sound` command can handle values between -1 and 1, so the command above re-scales the data you took to expand it accordingly.

- ```
  FileName = input('Enter filename for sound data: ', 's');
  FileSave = sprintf('save %s data time samplerate', FileName);
  eval(FileSave)
  ```

  Finally we want to save the matrices `data` and `time` to a file, along with the sampling rate. The code above shows how to get a filename from the user and use the `sprintf` command to create a string containing the command you want to run. The `eval` command then "runs" the string.

Having gone through what the commands do, you can now run the script. After the script starts, hit play on your audio source and then hit return to start recording. When the program indicates that recording is over, you should pause playback on your device so that it is no longer sending a signal to the speakers. That way, you can listen to how the computer "heard" it - this will be played back using the same sampling rate, but the voltages at the headphones will be determined by MATLAB's sending electrical signals to the headphone jack rather than the computer passing through the voltages sent to it via the microphone jack.

Discuss the sound quality of the playback versus the original recording with your lab partner and write down the conclusion of your discussion for inclusion in your lab report. Note that the volume may be different due to the scaling. Finally, save the data to a file called `SoundData44500`. MATLAB will actually call the file `SoundData44500.mat` since it is saved to MATLAB-native format. MATLAB will save the `data`, `time`, and `samplerate` variables to this file.

3.7 Data Acquisition and Analysis

The acquisition and analysis portion of this lab is very simple. You will be acquiring your sound clip with six different sampling rates and giving a quantitative analysis of the sound storage and a qualitative analysis of the sound reproduction. Be sure that you and your lab partner discuss the difference between the original sound and the sound you hear sampled at the different rates before moving on to the next sampling rate. This information will need to appear in your lab report. You should save the files using the same format as above, `SoundData`SR where SR is the sampling rate.

Using the superior output capabilities of the sound card, you can investigate how well the DAQ board captures sound. Find a particularly interesting clip of music in your song (vocals, background music, etc.) and use the `MirrorInDone` script to view and hear the sound for the sample rates in Table 3.1. Also use the `whos` command to see how many samples were stored and how much space they took - these two pieces of information will go in the second and third columns of the table, respectively. The number of samples is given in the `Size` column for the `data` array and the amount of memory is given in the `Bytes` column for the `data` array. Do *not* report the grand total - that value includes all matrices currently in memory.

Since all clips should be 5 seconds long, you can figure out how much memory will be taken up by the entire song by first dividing the memory in your `data` matrix by 5 to get a memory-per-second value, then multiplying that number by the song length in seconds. For example, if a 5 second clip of a 3:22 song takes up 1200 bytes, the whole song will take up

$$\text{Whole Song} = \frac{1200 \text{ bytes}}{5 \text{ seconds}} * \frac{202 \text{ seconds}}{\text{song}}$$

which is 48480 bytes or 47.34 KB per song.

Note that the prefixes "kilo," "mega," and "giga," with respect to *memory* amounts, may refer not to integer powers of 1000 but instead to powers of 1024. A kilobyte can thus mean 1024 bytes. This is because while 1000 is not an even power of 2, 1024 is (2^{10}) and is fairly close to 1000. Since memory stores binary information and that information is generally grouped in integer powers of two, it made sense to use powers of 2 for the prefixes as well.

There is an ongoing discussion about reverting the SI prefixes to mean exact powers of 10 and replacing the terms for 2^{10}, 2^{20}, and 2^{30} with "kibi," "mebi," and "gibi" where the "bi" at the end refers to the binary nature of the prefixes.[1] This issue is not as potentially catastrophic as the difference between long and short scales for numbers[2] but, as memory sizes increase, the disparity between the decimal and binary versions increases. For abbreviations of these prefixes, the letter "i" is inserted after the traditional abbreviation to indicate the binary version, such that 1 KiB (a kibibyte) would be 1024 bytes and 1 GiB (a gibibyte) would be 1,073,741,824 bytes (or 2,147,483,648 nibbles - which makes two gibinibbles or 2 Gini[†]).

Once you know how much space your song will take up, you can determine how many copies of your song sampled at that particular rate would fit on a 700 MB CD-RW and a 20 GB iPod. For example, given the song above,

$$\text{Songs Per CD} = \frac{\text{Song}}{48480 \text{ bytes}} \frac{700 * 1024^2 \text{ bytes}}{\text{CD}} \approx 15140 \text{ songs per CD or } 14.79 \text{ KiSongs}^{\ddagger}$$

$$\text{Songs Per iPod} = \frac{\text{Song}}{48480 \text{ bytes}} \frac{20 * 1024^3 \text{ bytes}}{\text{iPod}} \approx 442963 \text{ songs per iPod or } 0.4224 \text{ MiSongs}^{\ddagger\ddagger}$$

[1] http://en.wikipedia.org/wiki/Binary_prefix
[2] http://en.wikipedia.org/wiki/Long_and_short_scales
[†] Yes. Really.
[‡] No. Not really.
[‡‡] IBID. (IBIBID?)

3.8 Analog Output: Delivering Signals

Now that you have seen how MATLAB can use the DAQ to measure signals, you will use it to create signals. The DAQ board we are using has two possible outputs that can change values up to 1000 times per second. This is fast enough to generate sounds, though the quality is not spectacular. Instead of connecting an output channel to a speaker, you will hook up `DAC1OUT` to a combination of two LEDs (facing opposite directions). This way, you will be able to "see" portions of higher voltage (and thus volume) due to the brightness of the light.

The script described below, `MirrorOutDone.m`, will send a scaled version of the data to DAC1OUT on the DAQ board while simultaneously playing the music using the `sound` command. The sampling rate and data set going to the output cards are adjusted such that the two should take the same amount of time; thus, the music and lights should be in sync. Note in this case that the values sent to the data card will have been scaled such that the voltages reach ±10 V - this is to make sure that the LEDs will be at their brightest when the song is at its loudest.

The following bullets go over in detail what each line of the program will do. Make sure to read the descriptions of what each line of code is doing. Again, some will be a reminder from previous work.

- `delete(daqfind)`

  Again, it is good practice to clear out any DAQ objects before progressing further.

- ```
  FileName = input('Enter filename for sound data: ', 's');
  FileLoad = sprintf('load %s', FileName);
  eval(FileLoad)
  ```

  These commands load the `data`, `time`, and `samplerate` matrices from the file created in the `MirrorInDone` program using the process described on the Pratt Pundit page `MATLAB:Flexible Programming`.

- `AO=analogoutput('nidaq', 1)`

  This defines a handle `AO` for the output of the board.

- `addchannel(AO, 1)`

  This adds output channel 1 `DAC1OUT` as the first (and only) accessible output on the board.

- ```
  Ratio = ceil(samplerate/500)
  set(AO, 'SampleRate', samplerate/Ratio);
  set(AO, 'TriggerType', 'Manual');
  ```

  These commands are the same as for taking input, though a `SamplesPerTrigger` command is not needed because that will simply be the size of the array given to the output. Note that the `samplerate` was taken from the data file. The `Ratio` value will determine how much faster than 500 samples per second the original data was obtained; the sample rate for the DAQ output is then set to some value at or slightly below 500. This is to make sure the card accurately produces the voltages - signals going to the output faster than 500 samples per second can cause problems with these cards.

- `VoltageOut = [10*data(1:Ratio:end)/max(abs(data)); 0];`

  This line produces a data set where only certain samples are used based upon the difference in sampling rates between the audio output and the DAQ output. For example, if data were obtained at 5000 samples per second, `Ratio` would be 10 and only every tenth point would be sent to the card. The 0 at the end makes sure that the lights turn off when the clip is over.

  The values sent to `DAC1OUT` are rescaled to go between -10 V and 10 V instead of between -0.4 V and 0.4 V set before. Again, this is so that the signal sent to `DAC1OUT` will have a high enough voltage to turn the LEDs on. Note that these voltage are far too high for most headphones.

- `putdata(AO, VoltageOut)`

  This puts the numbers in the second argument in a queue for the output channel of `AO` - these values will be sent to the output channel once triggered.

- `AO`
  `start(AO)`
  `trigger(AO)`

  This displays the information about `AO` on the screen, activates the board, and then triggers the board to start sending data from its output to the DAQ board.

- `sound(data/max(abs(data)), samplerate)`

  This will play the sound. note that once the `trigger` happens, MATLAB will continue to run - it does *not* wait for the card to finish before moving on.

- `disp('Done')`

  This displays the text `Done` in the MATLAB window to indicate that the program has completed.

Now that you have gone through each line of code, run the script and enter `SoundData5000` as the file name. You should see the light blink in time with the beat of the music. After confirming that the sound is being reproduced correctly, you can hear how mismatched input and output sampling rates can alter the sound. Simply add a line of code in the `MirrorOutDone` script just below where the file is loaded that says:

```
samplerate = 0.5*samplerate;
```

that changes that value of the `samplerate` so that instead of the `SampleRate` property of the output being the same as when the data was acquired, the voltages are sent to the output half as fast. Re-run the script with the `SoundData5000` file and note what you hear. Change the line so that now it is `2.0*samplerate`. Run the program on `SampleRate5000` and again note what you hear. The program will automatically compensate for the limitation on the sampling rate of the DAQ card.

3.9 Assignment

For this lab, you and your partner will fill out a lab report that has Table 3.1 as well as answers to the questions on the handout. If you want, you can copy the two programs to your OIT account but that is not required.

3.10 Clean-Up

Before leaving the lab, please perform the following tasks:

(1) Carefully disconnect all wires and electrical elements from the CB-68LP, the breadboard, and the audio device. Leave other connections in place.

(2) Put the electrical elements, breadboard, and screwdriver back in the bin.

(3) Have a TA verify the contents of your bin.

(4) Take the wires and bins and return them to the front table.

3.11 Sample Laboratory Assignment

Names and Net IDs:

-
-
-

Data Table

Sample Rate	Samples (5-sec Clip)	Memory (5-sec Clip)	Memory (Whole Song)	Copies per CD	Copies per iPod
200000					
44500					
15000					
5000					
2500					
500					

Table 3.1: Quantitative Sound Data

Questions

(1) What song did you use? Be sure to include the title, artist, album, and length of the song in your lab report. The length is especially important to get an idea of how much memory your song will take relative to the 5-second clip.

(2) What was the maximum voltage the DAQ measured (and `softscope` displayed) for your clip? Why was this value used?

(3) How do you and your lab partner think the quality of the sound changed as the sampling rate decreased? Which rates would you say are at or near "CD" quality? "FM radio" quality? "AM radio" quality? Write a sentence or two on the quality and overall sound of the output from each rate.

(4) What happened when you tried to send the sound data to the DAQ card at the wrong sampling rate?

(5) Given all the information in the table and the qualities you assigned to different sampling rates, do you think data files such as those created by MATLAB are how information is stored on a CD? On an iPod? Defend your answer in each case.

3.11.1 Overall Network Listing

Item	First	Second	Note
AudioRedClip	Splitter	RED1	Clip red lead to red wire
RED1	AudioRedClip	Pin B23	Put red wire from clip in B23
AudioBlackClip	Splitter	BLK1	Clip black lead to black wire
BLK1	AudioBlackClip	Pin X1	Put black wire from clip in X1
RED2	Pin A23	Line 33	ACH1
BLK2	Pin X11	Line 66	ACH9
GRN1	Pin X14	Line 32	AIGND
RED3	Pin A6	Line 21	DAC1OUT
GRN2	Pin Y23	Line 55	DGND
RES	Pin B6	Pin F6	Resistor
LED1	Pin I6	Pin Y5	Long end in I6
LED2	Pin J6	Pin Y7	Long end in Y7

Table 3.2: Network Listing for Synchronous I/O

3.11.2 MirrorInDone.m

```
%% Initialize variables and objects
clear; format short e; figure(1); clf;
delete(daqfind);

%% Prepare Input
% Create handle to input
AI=analoginput('nidaq',1);

% Add channel 1 to input
addchannel(AI, 1);

%% Prepare DAQ Card
% set variables for sample rate and duration
samplerate=44500;
duration=5;

% calculate number of samples
samples=duration*samplerate;

% set sample rate, trigger type, and samples per trigger
set(AI, 'SampleRate', samplerate);
set(AI, 'TriggerType', 'Manual');
set(AI, 'SamplesPerTrigger', samples);

% set input, sensor, and units range
set(AI.Channel(1), 'InputRange', [-0.5, 0.5]);
set(AI.Channel(1), 'SensorRange', [-0.5, 0.5]);
set(AI.Channel(1), 'UnitsRange', [-0.5, 0.5]);

%% Use DAQ Card
% examine AI, start AI, and trigger AI
AI
start(AI);
fprintf('Press return to take data\n');
pause
trigger(AI);
fprintf('Taking data...\n')

% take data from AI
[data, time] = getdata(AI);

%% Plot and Play Data
plot(time, data)
fprintf('Press return to play sound\n');
pause
sound(data/max(abs(data)), samplerate)
fprintf('Playing sound...\n');

%% Save Data
FileName = input('Enter filename for sound data: ', 's');
FileSave = sprintf('save %s data time samplerate', FileName);
eval(FileSave)
```

3.11.3 MirrorOutDone.m

```
%% Initialize variables and objects
clear; format short e
delete(daqfind);

%% Load Data
FileName = input('Enter filename for sound data: ', 's');
FileLoad = sprintf('load %s', FileName);
eval(FileLoad)

%% Change sample rate?
samplerate = 1.0 * samplerate;

%% Prepare Output
% Create handle to output
AO=analogoutput('nidaq',1);

% Add channel 1 to output
addchannel(AO, 1);

%% Prepare DAQ Card
% set sample rate and trigger type
Ratio = ceil(samplerate/500)
set(AO, 'SampleRate', samplerate/Ratio);
set(AO, 'TriggerType', 'Manual');

%% Use DAQ Card
% put data to AO; end with 0
VoltageOut = [10*data(1:Ratio:end)/max(abs(data)); 0];
putdata(AO, VoltageOut);

% examine AO, start AO, and trigger AO
AO
start(AO);
trigger(AO);

%% Play Music
sound(data/max(abs(data)), samplerate)

%% Display when done
disp('done')
```