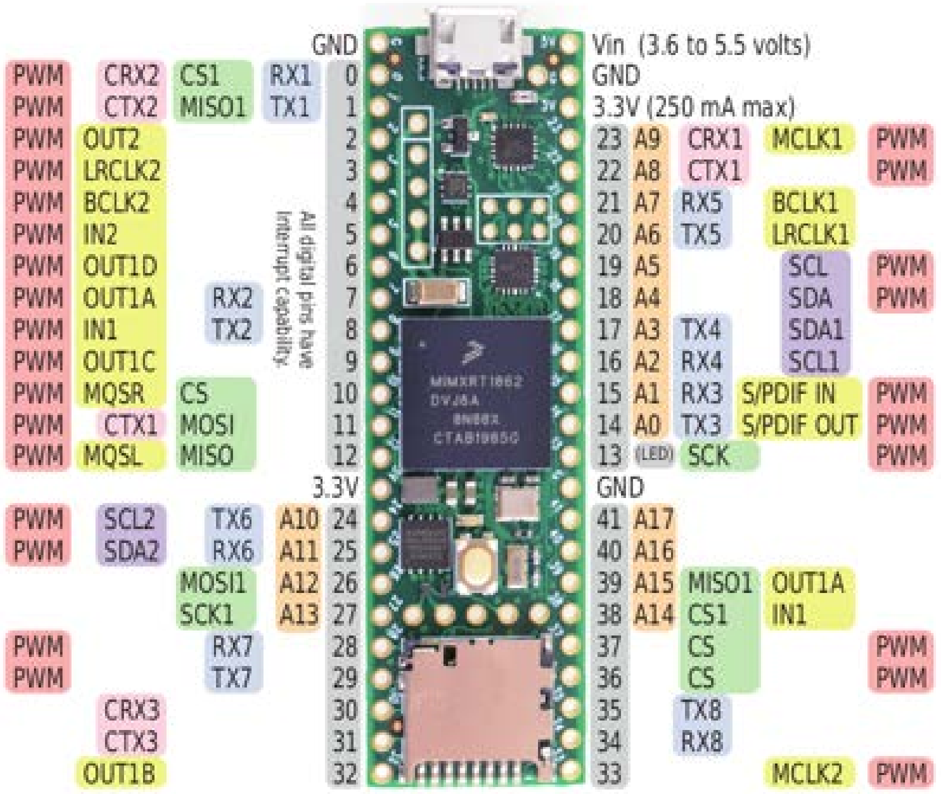
EECS 452 F21 Lab 3
Introduction to the Teensy 4.1 board



The purposes of this lab are to:

- **Provide an introduction to the Teensy 4.1, the board we will be using throughout this course.**
- **Provide an introduction to embedded systems and memory-mapped I/O devices.**
- **Provide an introduction to the FIR filtering on the Teensy Audio Shield.**

In addition, you will gain familiarity with the software development environment and the various forms of documentation and the sample code that comes with this platform.

Pre-Lab

Submit the answers to the Pre-Lab questions and other deliverables as part of the main lab report.

The remainder of the labs this semester will rely upon you having a fairly strong understanding of C programming. In EECS 280 and Engineering 101 you gained a fairly strong background in C++ programming. The differences between C and C++ are largely trivial. The biggest, for purposes of this class, are that:

- There are no classes, nor are there function members in structures.
- Rather than using streams ("cout" and "cin") to do the input and output we use "printf" and "scanf".

The following link offers a more comprehensive overview of the differences between the two languages:

https://www.geeksforgeeks.org/difference-between-c-and-c/

If it has been some time since you have coded in C++, it is recommended that you review C++ programming before the lab.

The second half of this lab will require you to perform basic FIR filtering on the Teensy. It is important that you recall what you learned about FIR filters. Lecture 4 reviews much of this FIR material but it is recommended that you also spend some time reviewing on your own. In particular, it will be helpful to review the following concepts before lab:

- Filter frequency response type (e.g. lowpass, bandpass, etc.)
- FIR filter structures (e.g. direct form, transposed form)
- Filter order and how the order can affect performance
- Group delay of a filter

Overview of Memory-Mapped input/output (MMIO)

When using an embedded system, you generally need some way to move data into, and out of, the processor. How does the computer do that? In EECS 280 the answer was simple: you call a function/class. For example, you most likely used *ofstream's open* function to open a file. But that begs the question: how does the function do it? How does the processor talk to the disk and other I/O devices?

The answer is fairly straightforward. Recall that your program and its data all live in memory. Each instruction, variable and array element are stored in a specific memory location. You can think of the computer's memory as a big array in which the program and its data reside. When you declare a variable, a memory location is reserved for that variable. Pointers serve as indices into the big array called *memory*. So when you assert *new* in C++, the pointer you get back from new is nothing other than the index into memory where the operating system has given you space to work.

It turns out certain locations in this big array called memory are dedicated for handling I/O devices. These memory locations are special whereby reads and writes to them are likely to have side effects. Suppose the memory location (i.e., address or pointer) OxDEADBEEF is mapped to an LED so that, for example, logical 1 (high) value at the location (or address) indicates that the LED should be "on" and a zero value indicates that the LED should be "off". One could also configure a different memory location where the value of some external switch can be determined. For instance, depending on the switch's setting, the read value at the memory location might be a 0 or a 1. Thus, your program could find the state of that switch simply by reading that memory location (using a pointer). Because such memory locations are already in use (by interfacing with external devices), they cannot be used by your program for other purposes. Writing a 1 into 0xDEADBEEF can certainly be read at a later point in your program, but it will also turn the external LED on, whether you intended to or not. Significantly more I/O devices can be addressed with Memory Mapped Input/Output (MMIO). In fact all I/O devices (including disk drives, monitors and keyboards) communicate with the processor via MMIO. Wikipedia has a *Memory Mapped I/O* overview which you might find helpful.

Details of Memory-mapped input/output on the Teensy

In most processors, Memory Mapped I/O (MMIO) is simply a part of the overall memory map. This is true with the PJRC Teensy boards as well. The primary way of interacting with the peripherals on the Teensy is through the use of the Hardware Abstraction Layer (HAL) APIs. The HAL APIs provide a way of interacting with the different devices and peripherals on the board without having to understand all the hardware level complexities (without knowing the actual address for I/O control because it is abstracted out by defining address mappings in header files). This allows for writing simpler code that is more portable across different devices. Additionally, you can also interact directly with the hardware by referencing the documentation to see where peripherals are mapped in memory, and then directly manipulating the values in memory. There are benefits and drawbacks to both approaches, and we will explore the HAL based approach throughout this semester.

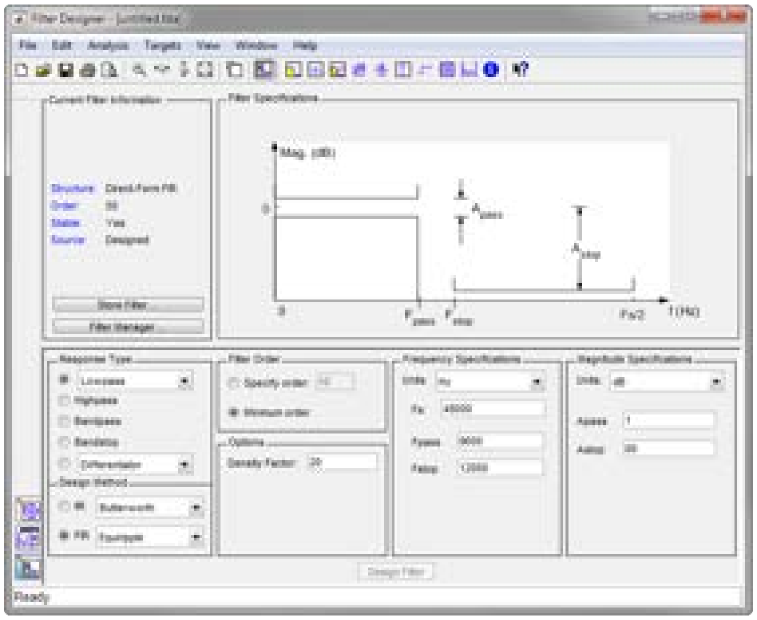
Teensy/Arduino IDE Setup

Complete the instructions defined in "`Teensy_Setup.pdf`", these instructions are meant to help you download and install the Arduino IDE as well as the Teensyduino add-on. These softwares are required to upload the code from the R-Pi onto the Teensy board.

FIR Filter Coefficient Generation

We are going to use MATLAB to generate an FIR filter because it has the tools to make a filter with hundreds of taps in seconds, where doing this by hand might take hours. One issue with our documenting this process is that MATLAB changes once or twice a year. This means that the "look and feel" of the tools you are using may be different than shown below. It also means that different sites may be running different versions.

With these caveats in mind, let's start the task of designing an FIR filter in MATLAB. Open up MATLAB on your computer. Type the command `filterDesigner` [R2017b and beyond] (`fdatool` in previous versions) into the command window. You should get the following window on your screen.



Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Units: Hz
Fs: 48000
Fpass: 9600
Fstop: 12000

Units: dB
Apass: 1
Astop: 80

Density Factor: 20

From here, you can design an FIR or IIR filter via this interactive MATLAB GUI. It's important to note, however, that you don't have to use the GUI to design a filter in MATLAB. Any filter designed interactively can also be designed using one of several MATLAB filter-design functions. This fact is useful when designing multiple filters or a filter whose frequency response may change dynamically over time. You can use the GUI to explore various options and narrow the scope of your design. Then you can write a script that automates the design steps.

Designing the lowpass filter

Using the GUI, design a low-pass filter as follows:

- Go to the left-hand side of the GUI and select "Lowpass" in the "Response Type" section. (Make sure the bubble is filled and the text field reads "Lowpass").
- Set the method in the "Design Method" section to "FIR" and make sure option is "Equiripple" (typically the default value).
- Next, you could let MATLAB decide how large to make your filter by selecting the "Minimum order" option in the Filter Order section of the window, just right of the Response Type window. However,

we will specify the order. To do this, select "Specify order" in the Filter Order section and enter 99 into the text field next to it.

- Now let's use the "Frequency Specifications" section of the GUI to set the lowpass frequency properties of this filter by specifying the transition band. We leave the "Fs" field alone because we will be sampling at 48 kHz, which is the default. Set Fpass to 1000, to determine the passband cutoff frequency, and Fstop to 3000, to determine the slope of the lowpass roll-off.
- Press the "Design Filter" button on the bottom of the window. MATLAB will calculate the filter coefficients and display the magnitude response of the design in the "Filter Specifications" figure.

Exporting the design

Now that MATLAB has generated our FIR filter, we need to export the coefficients so we can use them in a C program. Because it is so common to want to implement filters in C, MATLAB has a handy mechanism for generating the output as a C-header file. Select "Targets" in the GUI menu and then "Generate C header".

To ensure that your header matches the structures declared in the Lab 3 C code, modify the default values in the "Generate C Header" window as follows:

- Change the "Numerator:" field to "LP"
- Change the "Numerator length:" field to "LPL"
- Select the "Export as:" option and "Signed 16-bit integer" by scrolling through the pop-up menu.

Now click the "Generate" button to complete the process. You will be asked where you want to put your new file. You may want to make a new folder on your desktop to accomodate these header files. Rename the file to `low_pass.h` and click "Save".

From the above, you have created a header file where the FIR coefficients are represented as 16-bit signed integers. This file requires further modification to be compatible with the Teensy development environment. Edit your header file as follows:

- Delete the #include near the top of the file. Replace it with `#include <stdint.h>`
- Change the `const int LPL = 100;` to `#define LPL 100`
- Change `const int16_T LP[100]` to `int16_t LP[100]`

Save your changes. Do not add a semicolon after definitions. They will throw compiler errors.

Q1 Print the file you generated (and modified) as described above. **Q2** The default structure of the FIR filter is "Direct-Form FIR".

(a) How is this different from the Transposed-Form?

(b) Draw the Direct-Form and Transposed-Form FIR.

(c) What do you think the advantage of each structure is?

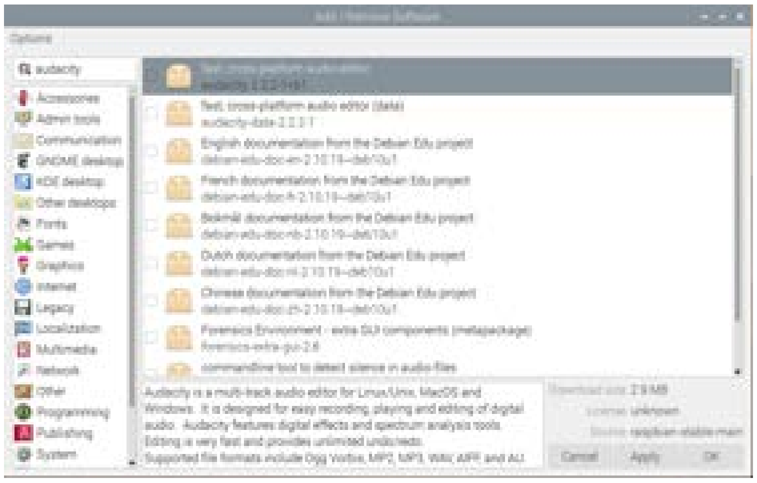
Q3 Once you have designed a filter in `filterDesigner`, you can examine properties of that filter using "Analysis" from the menu or by selecting one of the tool icons from the toolbar. Based on the `filterDesigner` display of your lowpass filter, answer the following questions:

(a) What is the expected group delay? [Hint: Use the "group delay" option in the Analysis menu.]

(b) What is the expected phase shift in a 5 kHz input? [Hint: Use the "Phase response" option in the Analysis menu and "Zoom in" feature from the toolbar (the magnifying glass icon).]

Go back to MATLAB to generate an equiripple band-pass filter that has the following properties:

- order=99



fast, cross-platform audio editor
audacity-2.2.2-1+b1

fast, cross-platform audio editor (data)
audacity-data-2.2.2-1

English documentation from the Debian Edu project
debian-edu-doc-en-2.10.19~deb10u1

French documentation from the Debian Edu project
debian-edu-doc-fr-2.10.19~deb10u1

Bokmål documentation from the Debian Edu project
debian-edu-doc-nb-2.10.19~deb10u1

Dutch documentation from the Debian Edu project
debian-edu-doc-nl-2.10.19~deb10u1

Chinese documentation from the Debian Edu project
debian-edu-doc-zh-2.10.19~deb10u1

Forensics Environment - extra GUI components (metapackage)
forensics-extra-gui-2.6

commandline tool to detect silence in audio files

Audacity is a multi-track audio editor for Linux/Unix, MacOS and Windows. It is designed for easy recording, playing and editing of digital audio. Audacity features digital effects and spectrum analysis tools. Editing is very fast and provides unlimited undo/redo. Supported file formats include Ogg Vorbis, MP2, MP3, WAV, AIFF and AU.

Download size: 2.9 MB
License: unknown
Source: raspbian-stable-main

Figure 1: Add/Remove Software

- Fstop1=1000
- Fpass1=3000
- Fpass2=4000
- Fstop2=6000
- Fs=48000

As before, generate a C header file and modify it as needed be used in your project (Same way as performed for the lowpass). Name the coefficient array "BP", and name its length "BPL". Name the coefficients file to band_pass.h. Make a copy of your pre-lab, you will need it for the in lab part.

In-Lab

1 Setup

1.1 Do This First

1. Check that your internet is active (log in to MGuest every time)
2. Check if the software below is already installed
3. Check your Arduino version – must be 1.8.X (if it's 2.XX run `$ sudo apt purge arduino -y`)
4. The default RPi password is `raspberry`

1.2 Audacity Setup

Audacity is a free, open source, cross-platform audio software. We will be using Audacity running on R-Pi to play and record audio signals to/from the Teensy board. This will allow us to visualize the effects of performing FIR filtering on our audio signal without using a lab equipment.

To install Audacity onto the Raspberry Pi, go to the applications menu in the top left corner → Preferences → Add/Remove Software and type "Audacity" in the search box in the top-left corner. This should show the different packages as shown in Figure 1.

Check the box next to `fast cross-platform audio editor` and click on `Apply` in the bottom right. A window should pop up requesting for the Raspberry Pi password as set by you during the RPi configuration.

Upon entering the password and clicking OK, the package will get installed by itself. Once this is done Click on OK in the Add/Remove Software window. Audacity is now installed on your RPi.

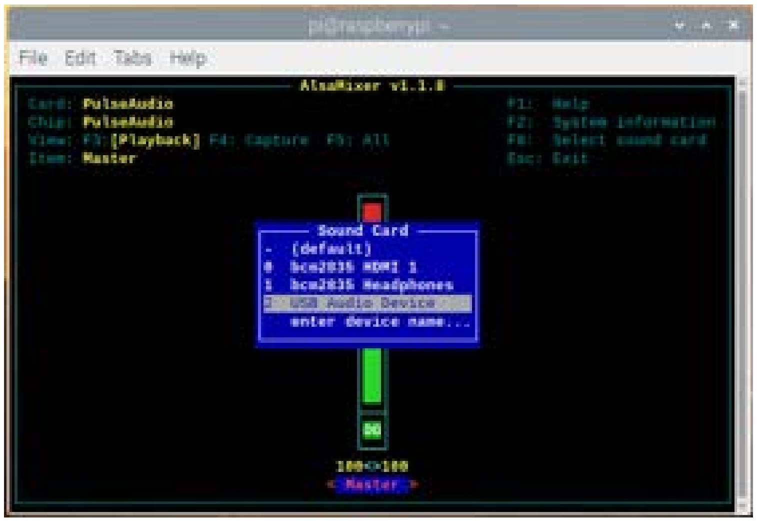
1.3 Volume Enabling

The following steps will help you debug any issues with volume if nothing is being played from the Raspberry Pi. Be sure to plug in the USB audio adapter to the Raspberry Pi for this section.

1.3.1 AlsaMixer

First thing to do is to check if alsamixer volume of your device is muted or not.

1. Go to terminal, which should be present in the upper left hand corner of the Raspberry Pi.
2. Type in the command **alsamixer**. This should open up alsamixer.
3. Press F6 to Select the Sound Card. Here, choose your output, and make sure the volume is full. As shown in Figure 2.



```
                         AlsaMixer v1.1.8
Card: PulseAudio                               F1:  Help
Chip: PulseAudio                               F2:  System information
View: F3:[Playback] F4: Capture  F5: All       F6:  Select sound card
Item: Master                                   Esc: Exit

                  Sound Card
            -  (default)
            0  bcm2835 HDMI 1
            1  bcm2835 Headphones
            2  USB Audio Device
               enter device name...

                      00
                   100<>100
                   < Master >
```

Figure 2: AlsaMixer

1.3.2 PulseAudio

If AlsaMixer did not work, then we may have to install PulseAudio.

1. Go to terminal, which should be present in the upper left hand corner of the Raspberry Pi.
2. To install pulseaudio, type **sudo apt-get install pulseaudio**.
3. To start PulseAudio, use the command **pulseaudio -D**.
4. Then, install pavucontrol by using the command **sudo apt-get install pavucontrol**.
5. Open up pavucontrol by typing **pavucontrol** into Terminal.
6. Here, make sure the output is set correctly, and ensure that it is not muted. Check all the tabs to ensure everything looks correct. Use Figure 3 as a reference.

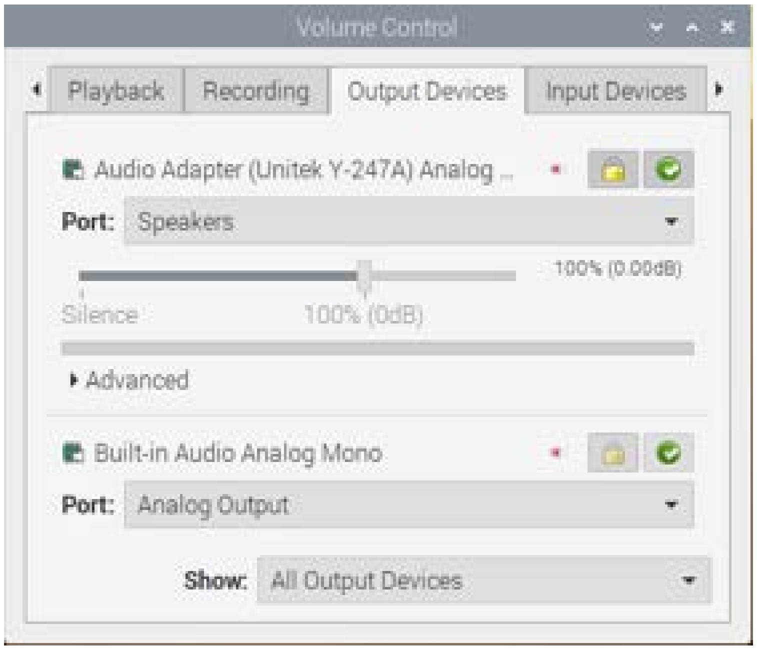


Figure 3: PulseAudio

2 Serial Interface and Interrupt-Based Timers

Now that we have Audacity installed, we can start getting familiar with the Teensy 4.1 board and the Arduino IDE. Teensy features an ARM Cortex-M7 processor capable at running at 600MHz. This makes it ideal for audio processing applications requiring high sampling rates.

To get started, download *blink_LED.zip* from Canvas. To extract the file on R-Pi, right click on it and click on '`Extract here`'. This will create the extracted folder of the file. Open Blink_LED.ino using Arduino IDE.

The **IntervalTimer** library allows us to use the in-built timers present on the Teensy. This allows us to attach interrupts to a timer. The line

```
myTimer.begin(blinkLED, 150000);
```

attaches the function *blinkLED* to the timer. That is, the function *blinkLED* is called whenever the timer elapses. The second parameter indicates the how often the timer elapses to call the *blinkLED* function. This value is entered in terms of microseconds. The **IntervalTimer** also gives us the option of updating this delay and we will be using this to control the blinking of the LED on our Teensy board.

Most microcontrollers have a Serial port. This mode of communication follows the UART protocol. You can read more about this protocol in the article found in the link below.

What is UART protocol?

This allows R-Pi to communicate with the Teensy using the USB cable. To do so, first we need to upload this code onto our Teensy board. After opening Blink_LED.ino using Arduino IDE, follow the instructions as listed below:

- Connect the Teensy to one of the Raspberry Pi's USB ports using the USB A to micro USB cable provided.
- Upon connecting, go to `Tools` → `Port` on Arduino IDE and select the port at which the Teensy is connected.
- Now go to `Tools` → `Board` → `TeensyDuino` → `Teensy 4.1` as shown in Figure 4.

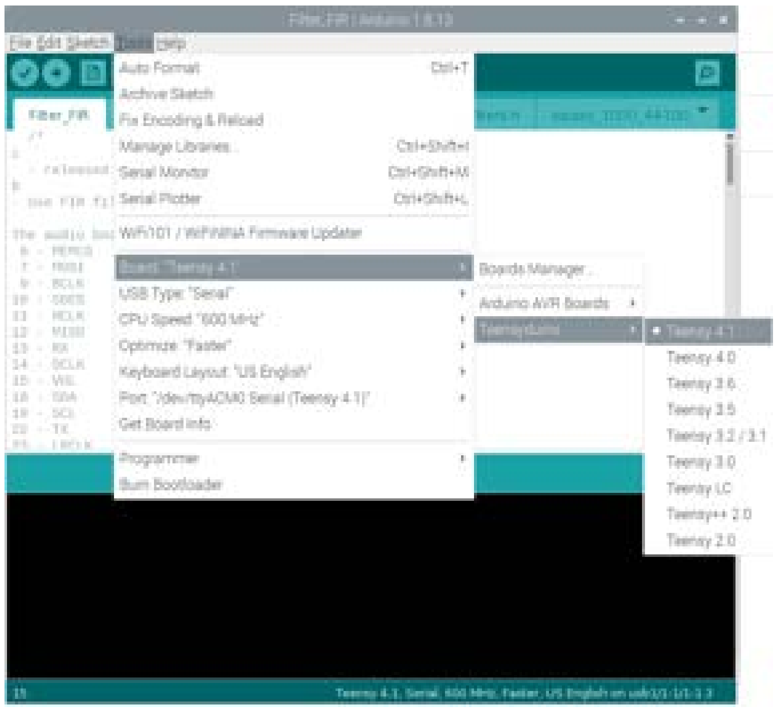


Figure 4: Selecting Teensy 4.1 board

- Make sure that `Tools` → `USB Type` is set to *'Serial'*
- Click on `Sketch` → `Upload` or on the right-arrow button on the left hand side of Arduino IDE. This should start compiling the code and then upload it onto the Teensy board.

Once you have the code uploaded, open the Serial monitor on the Arduino IDE. You can do this by either clicking the magnifying glass icon on the right top corner or by going to `Tools` → `Serial Monitor`. The code is defined to take a positive integer entered by the user and set that as the frequency with which the LED blinks. You can test the code by entering different numbers. Experiment with numbers between 1 and 1000. Upon entering larger numbers you should be able to observe that the LED blinks much slower.

Q4 Show that your LED program is properly working by taking pictures/video.

3 FIR Filter

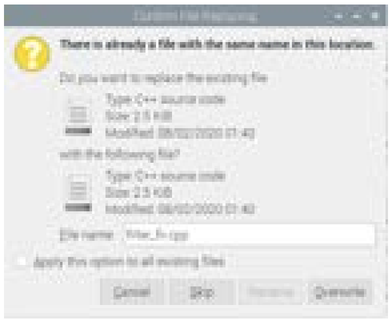
For the FIR Filter, first download from Canvas and extract the "FIR_Lab3" folder. Copy the "low_pass.h" and the "band_pass.h" files that you generated in the pre-lab into the same folder. Now open the FIR_lab3.ino file.

Since we don't want to use external oscilloscopes and function generators, we will be using Audacity to observe the audio input and output generated by Teensy. The Audio Shield attached to the Teensy is responsible for interfacing digital and analog audio signals from/to Teensy. It contains the *SGTL5000*, a low-power stereo audio codec with digital-to-analog (and vice versa) conversion.

We will first be replacing the *filter_fir.cpp* with our modified version of the file. To do so, open the filter_fir.cpp file which is present in the "FIR_filter" directory. Now copy this file into the following location: `arduino-1.8.XX`→`hardware`→`teensy`→`avr`→`libraries`→`Audio`. This will bring up a window requesting permission to overwrite this file as shown in Figure 5. Click on *Overwrite*.

Alternatively, if both your downloaded *filter_fir.cpp* file, as well as your Arduino folder, exist in your "Downloads" directory. You can run the following command in the terminal window (*terminal* is the fourth icon from the left in the top-left corner of the screen) :

```
$ sudo cp Downloads/filter_fir.cpp path-to/arduino-1.8.XX/hardware/teensy/avr/libraries/Audio/filter_fir.cpp
```


Confirm File Replacing

There is already a file with the same name in this location.

Do you want to replace the existing file

Type: C++ source code
Size: 2.5 KiB
Modified: 08/02/2020 01:40

with the following file?

Type: C++ source code
Size: 2.5 KiB
Modified: 08/02/2020 01:40

File name: filter_fir.cpp

Apply this option to all existing files

Figure 5: Overwrite filter_fir.cpp

For the hardware connections, ensure that the Teensy is disconnected from the Pi while making the connections. Mount the Teensy onto the Audio shield (ensure this part is done correctly, else you could end up short-circuiting your Teensy board). You can look up how the Teensy sits on the Audio shield online. Connect the black jumper wire of the audio jack adapter to one of the line-in ground pins (pin-out can be seen on the underside of the board) and the other two wires to the left and right line-in pins. Your setup should look as shown in Figures 6, 7, and 8. Connect the Audio shields line-in audio jack to the speaker output of the USB sound-card (green color) provided to you using an AUX cable. The speaker output is also mentioned in the manual that comes with the USB sound-card manual. Next, connect the line-out audio jack on the Audio shield to the microphone input (orange color) of the USB sound-card using an AUX cable. Connect the USB end of the Sound-card to one of the USB ports on the RPi. Plug the Teensy back to the R-Pi using the USB cable.

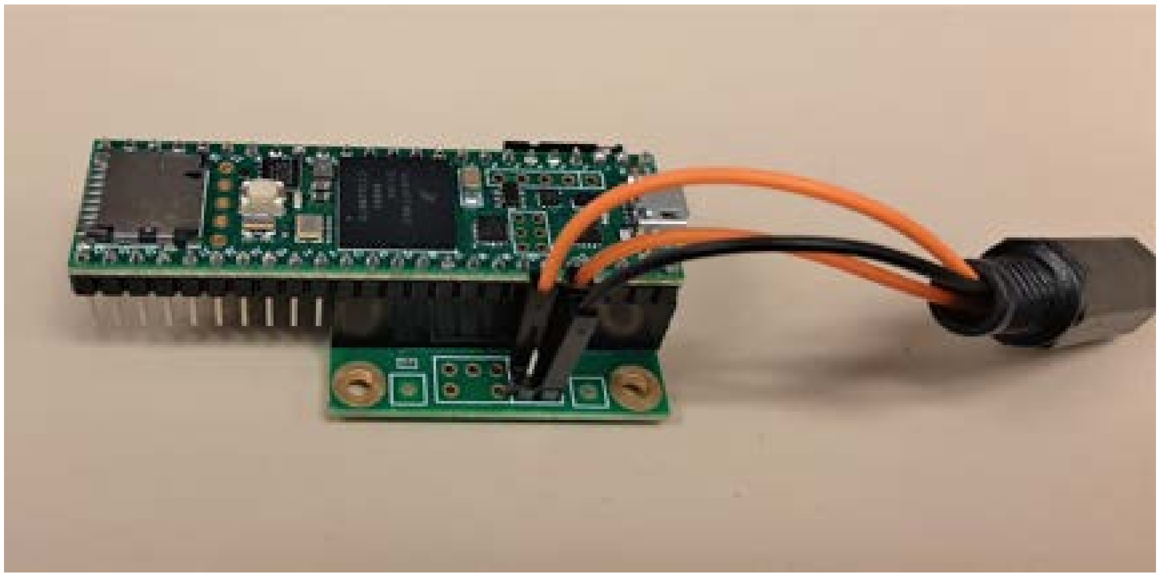


Figure 6: Connecting the audio jack adapter

Once you've got your hardware setup correctly configured, you can upload the *FIR_lab3.ino* code onto the Teensy board as was done earlier for the LED blinking section. When the Arduino IDE says "Done Uploading" we need to open Audacity to view the filter operation. To do so, go to the applications menu on the top-left corner, under the "Sound and Video" tab and click on Audacity. Once you have Audacity open, go to *Edit→Preferences* and click on the recording tab. Here ensure it it exactly matches that shown in Figure **??**.

Now go to *File* → *Open* and open the "FrequencySweep.wav" file from the Canvas lab folder. Once this is open, change the speaker output device and microphone input to device to *USB Audio Device*, by going

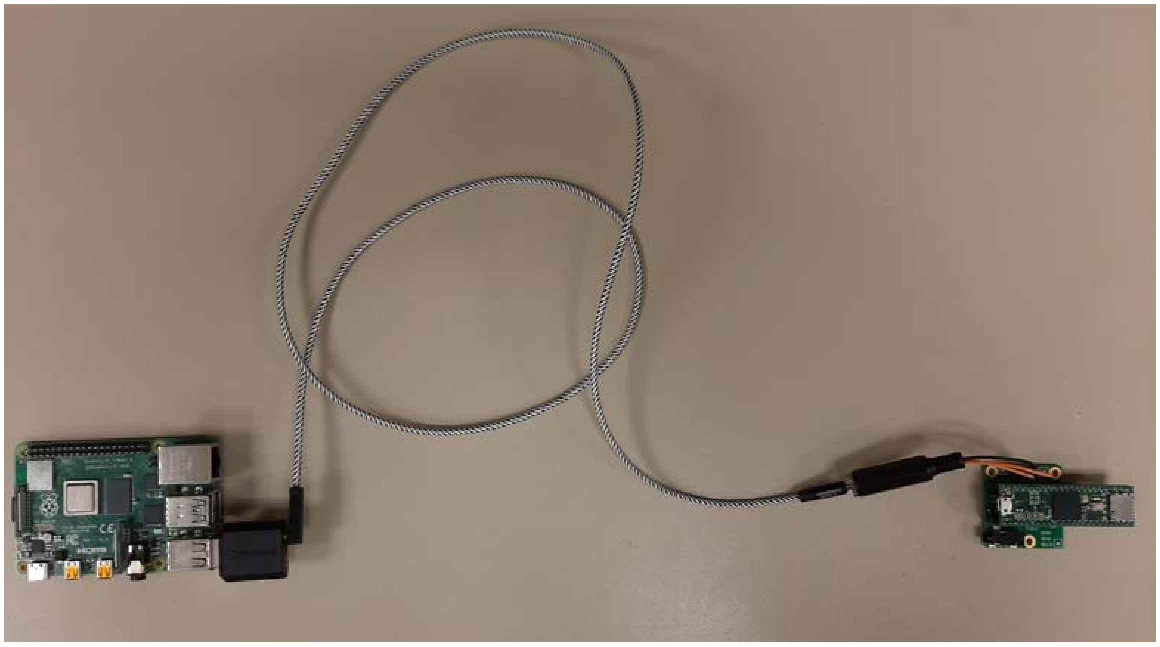


Figure 7: Connecting the audio output from the RPi to the Teensy

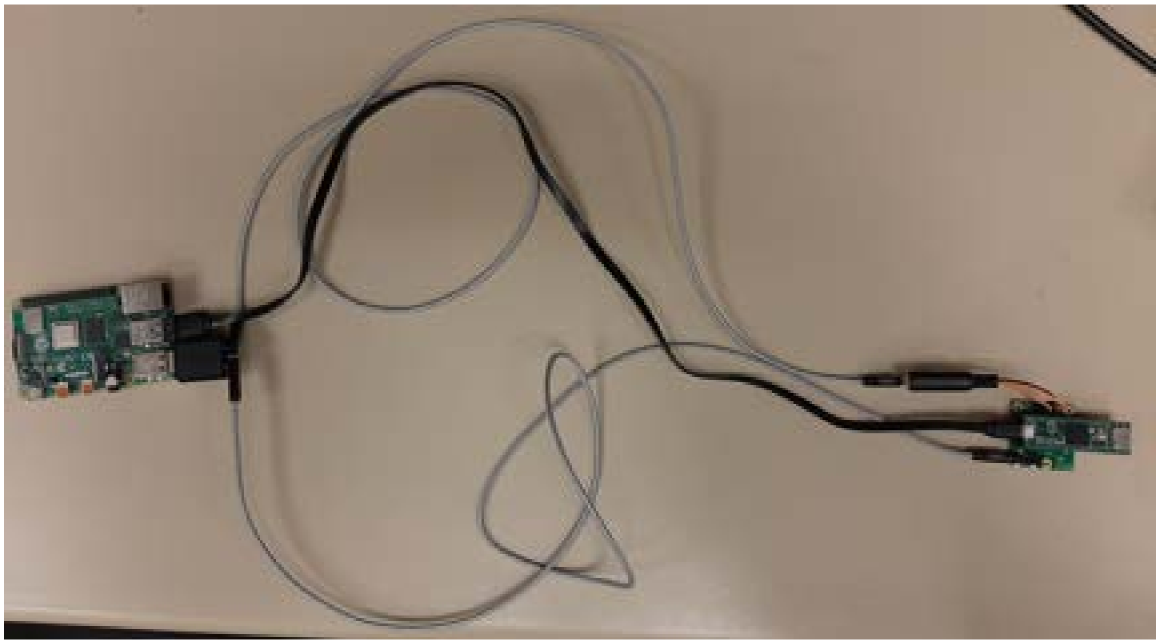
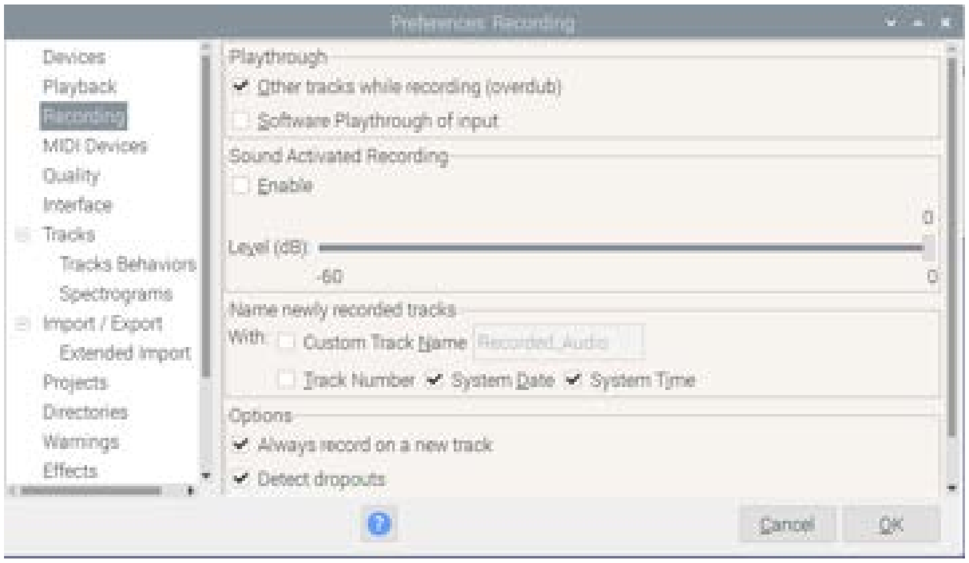


Figure 8: Complete Setup

to *Edit*→*Preferences* as shown in Figure 10. Reduce the speaker volume on the top-right to half.

Click on the drop-down menu next to "Frequency Sweep" and change it to the *Spectogram* option. This should show you the waveform in terms of time and frequency. The frequency sweep file is a sweep signal ranging from 20 Hz to 20 kHz. The coloration of the signal indicates the amplitude. Use the pointer and change the axis accordingly to view the entire signal. Click on the record button (Red) to start playing and recording the input simultaneously.

Once the new track appears with the input from Teensy after FIR filtering, make the same change to



Preferences: Recording

Devices
Playback
Recording
MIDI Devices
Quality
Interface
Tracks
Tracks Behaviors
Spectrograms
Import / Export
Extended Import
Projects
Directories
Warnings
Effects

Playthrough
[✓] Other tracks while recording (overdub)
[ ] Software Playthrough of input

Sound Activated Recording
[ ] Enable
Level (dB): -60 0

Name newly recorded tracks
With: [ ] Custom Track Name Recorded_Audio
[ ] Track Number [✓] System Date [✓] System Time

Options
[✓] Always record on a new track
[✓] Detect dropouts

Cancel OK

Figure 9: Audacity Preferences
fig:preferences‘

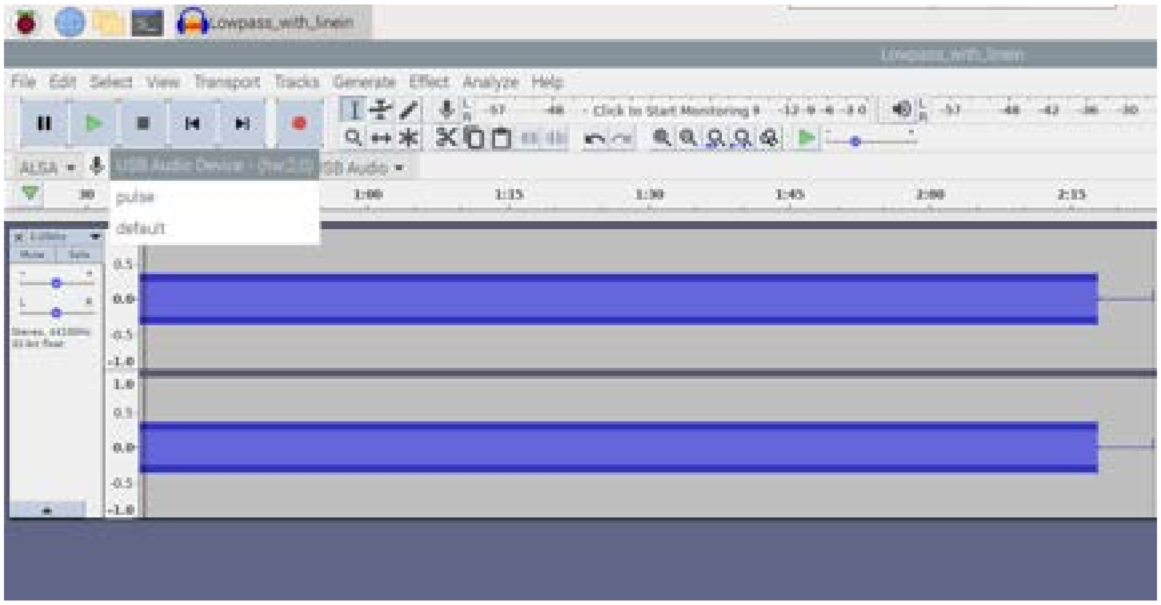

Lowpass_with_linein
File Edit Select View Transport Tracks Generate Effect Analyze Help
Click to Start Monitoring
ALSA
USB Audio Device - (hw:2,0)
USB Audio
pulse
default
1:00
1:15
1:30
1:45
2:00
2:15

Figure 10: Selecting USB audio device on Audacity

spectogram. Observe the recorded signal and record a screenshot of the same.

Now, let’s look into the code *filter_fir.cpp* running on Teensy with Audio Shield. The Audio library developed by PJRC for the Teensy Audio Shield board uses audio blocks to sample the input audio data. This is the “audio_block_t” data-type. This data type contains an array of 16-bit integer sample values. The length of this array is defined by AUDIO_BLOCK_SAMPLES. This constant is usually set at 128 samples. The sampling rate at which the Teensy audio shield works is set to a constant 44.1 k samples/sec. You can read more about the Audio library and its features to use it for your own projects in the following link:

https://www.pjrc.com/teensy/td_libs_AudioNewObjects.html

The “update” function is called as soon as 128 samples (i.e., a block) of audio data are received. It then allocates memory for a new block into which the filtered output is stored.

We will now take a look at the FIR filter supplied by ARM in the CMSIS Library. CMSIS is open source and available on GitHub (https://github.com/ARM-software/CMSIS) where you can read the source but more importantly you can find examples if you need to in the future.

For now, the documentation will have enough information for our application. You can find the CMSIS documentation at the link below.

http://www.keil.com/pack/doc/CMSIS/DSP/html/modules.html

Right now, we are interested in the documentation for FIR filters so navigate to the section on Finite Impulse Response (FIR) Filters. Once memory has been allocated to the new block we pass it to the *arm_fir_fast_q15()* function, which generates the time-domain filtered response. Note that this function implements a fixed-point FIR filter which takes 16-bit integer samples from Audio Shield. The *transmit()* function let us pass this block to our DAC output channels.

To measure amount of time used by the filter function, you need to copy the "Filter_FIR_micros.cpp" to the Audio library directory. To do so, download from Canvas the filter_fir_micros.cpp file present in the "Micros" directory. Now copy this file into the following location:
`arduino-1.8.XX`→`hardware`→`teensy`→`avr`→`libraries`→`Audio`. Once this is done you need to delete the "filter_FIR.cpp" in the library. This is done to remove duplicates of the underlying functions.

Alternatively, if both your downloaded *filter_fir.cpp* file, as well as your Arduino folder, exist in your "Downloads" directory. You can run the following commands in the terminal window (*terminal* is the fourth icon from the left in the top-left corner of the screen) :

```
$ sudo cp Downloads/filter_fir_micros.cpp Downloads/arduino-1.8.XX/hardware/teensy/avr/libraries/Audio/filter_fir_micros.c
$ sudo rm Downloads/arduino-1.8.XX/hardware/teensy/avr/libraries/Audio/filter_fir.cpp
```

The code prints out the amount of delay during each function call of *arm_fir_fast_q15*. This can be seen by opening the Serial monitor from Arduino IDE as we did for the LED blinking section.

Q5 Show in your report that the LPF is properly working (observed in Audacity).

Q6 How much time is spent in *arm_fir_fast_q15* during each call to the function?

Q7 How much time is the CPU idle between consecutive FIR input sample blocks (128 samples) of your filter? **Hint:** how would you measure time *outside* of the function in `filter_fir_micros.cpp`?

Q8 Answer the following questions on filter timing.

(a) If we have a 200MHz processor and we sample at about 50kHz (makes the math easier) what is the maximum number of cycles (of 200MHz clock) that our filter can use for each input block (128 samples) on average before we need to worry about not being able to meet our real-time constraints?

(b) About how many cycles does the current filter take per sample (note the function runs for a 128-sample block)? Calculate the number of cycles using the maximum 600 MHz clock frequency of the Teensy.

(c) Given the above, how large do you think we could make the filter before we started to run out of CPU time? (Hint: how does the number of multiplications per output sample change for filtering when you change the length of the filter? The processing time is linearly proportional to the number of multiplications in filtering.)

Repeat the experiment with the bandpass filter you designed in the pre-lab. To be able to run the band-pass filter instead of the low-pass change "*start_idx*" variable in the `FIR_Lab3.ino` file to 1 from 0 and re-upload. Record a screenshot of the Audacity window for the same.

Q9 Show in your report that the BPF is properly working (observed in Audacity).

4 Group Delay

To measure group delay, you need to copy the "Filter_FIR_GD.cpp" to the Audio library directory. To do so, download the filter_fir_GD.cpp file. Now copy this file into the following location:
`arduino-1.8.XX`→`hardware`→`teensy`→`avr`→`libraries`→`Audio`. Once this is done you need to delete the "filter_FIR_micros.cpp" in the library. This is done to remove duplicates of the underlying functions.

Alternatively, if both your downloaded *filter_fir.cpp* file, as well as your Arduino folder, exist in your "Downloads" directory. You can run the following commands in the terminal window (*terminal* is the fourth icon from the left in the top-left corner of the screen):

```
$ sudo cp Downloads/filter_fir_GD.cpp Downloads/arduino-1.8.XX/hardware/teensy/avr/libraries/Audio/filter_fir_GD.cpp
$ sudo rm Downloads/arduino-1.8.XX/hardware/teensy/avr/libraries/Audio/filter_fir_micros.cpp
```

Once this is done, open the file so we can try and understand how to view and measure the group delay cause by the FIR filter. We have printed out the audio samples before and after the ARM FIR function. This should show us the group delay of the filter. Now let's try viewing this data.

Be sure to change `start_idx` to 0 so we can measure the Group Delay of our low pass filter. Upload the "FIR_lab3.ino" to the Teensy board, this time with the "filter_FIR_GD.cpp" in the library. Once the code is uploaded, open Audacity and ensure the configurations are the same as earlier. Create a 1kHz Sine wave by clicking `Generate`→`Tone...` and updating the parameters accordingly. Make sure your signal is at least 1 minute long so you have time to measure the group delay. Click "OK" and the signal should now be in an audio track. Click on the play button, then return to the Arduino IDE and open the Serial monitor. Once it is open, reset the Teensy by pressing the button on the board. This is to ensure the code restarts as we print the audio sample values only once, this is done because printing takes significant amount of time, which would cause further delay in the output signal.

After pressing the reset button, you should see a bunch of values printed on the Serial monitor, copy these values into a CSV file. This can be done by open the "Applications menu" in the top-left corner and clicking on "Text Editor" under the "Accessories" tab. Once open, copy the values and save the file to the `GroupDelay` folder named as `"plot_gd.csv"`. Once you've saved the file, open the Terminal, navigate to the `GroupDelay` folder and type in the following commands to visualize the captured samples using Python:

1. `sudo apt install python-backports.functools-lru-cache`
2. `sudo apt-get install python-gi-cairo`
3. `python3 plot_from_csv.py`

Note that Step 1 - 5 are needed only once (not every time you plot data) to install necessary Python library functions. If followed correctly a plot should open up showing a line-graph of the input and output audio signal. The x-axis indicates the sample number and the y-axis has the amplitude of the signal. Use the difference in sample number between the peaks to calculate the group delay experienced by the signal. The sample rate is approximately 44.1k samples a second by default on the Teensy microcontroller.

Q10 In the pre-lab, you were asked to predict the group delay of the LPF. What group delay do you actually see? Is there any mismatch between the expected and actual group delay? Hint: think about the period of the signal and what happens when you delay the signal by the expected the group delay (which can be longer than the period of the signal).

Q11 What is the signal frequency at which the phase shift will be 180 degrees due to the group delay of the filter?

When the lab is finished, delete "Filter_FIR_GD.cpp" in the Audio library directory, and copy "filter_fir.cpp" from the 'Original' folder on Canvas to the Audio library directory (arduino-1.8.XX/hardware/teensy/avr/libraries/Audio/). This will allow you to use the original FIR filter code in the future without modification for this lab.

When things go wrong

- While uploading code, if you see an error message saying that the code was not able to be uploaded, try rebooting the Teensy by pressing the button on the board and upload again. If this still shows the error, then unplug and replug the Teensy board.
- If you find that your Serial Monitor is blinking and not showing the data properly as soon as you start playing audio through audacity, first try changing the USB port to which the Teensy is connected on the RPi. You will have to then re-select the port at which the Teensy is connected and then open up the Serial monitor. Try different ports if the problem persists.
- If you see an Error message on Audacity, reboot your RPi using the command `'sudo reboot'` on the terminal window.

5 Deliverables

Post your report with a typed set of answers to the questions. Post supplementary files (captured video, pictures, etc.) as necessary. If all the deliverables (screenshots, etc.) can be put into a single PDF file, then a single PDF file is an acceptable submission.