Lab 3 Part 2: FIR Filter Hardware Implementation

Haohan Zhu 22308057

1. FIR design and expect effect

I designed a high pass filter to filter the noise in the “audio.wav”. The original “audio.wav” has a very heavy click in the sound file. The filter expects to clear the click. But based on the original sound has many white noises in the spectrum. The filter cannot clear all the noise.

Because the noise is at the beginning of the signal and decreasing with the frequency. So I used the High Pass filter to filter the signal.

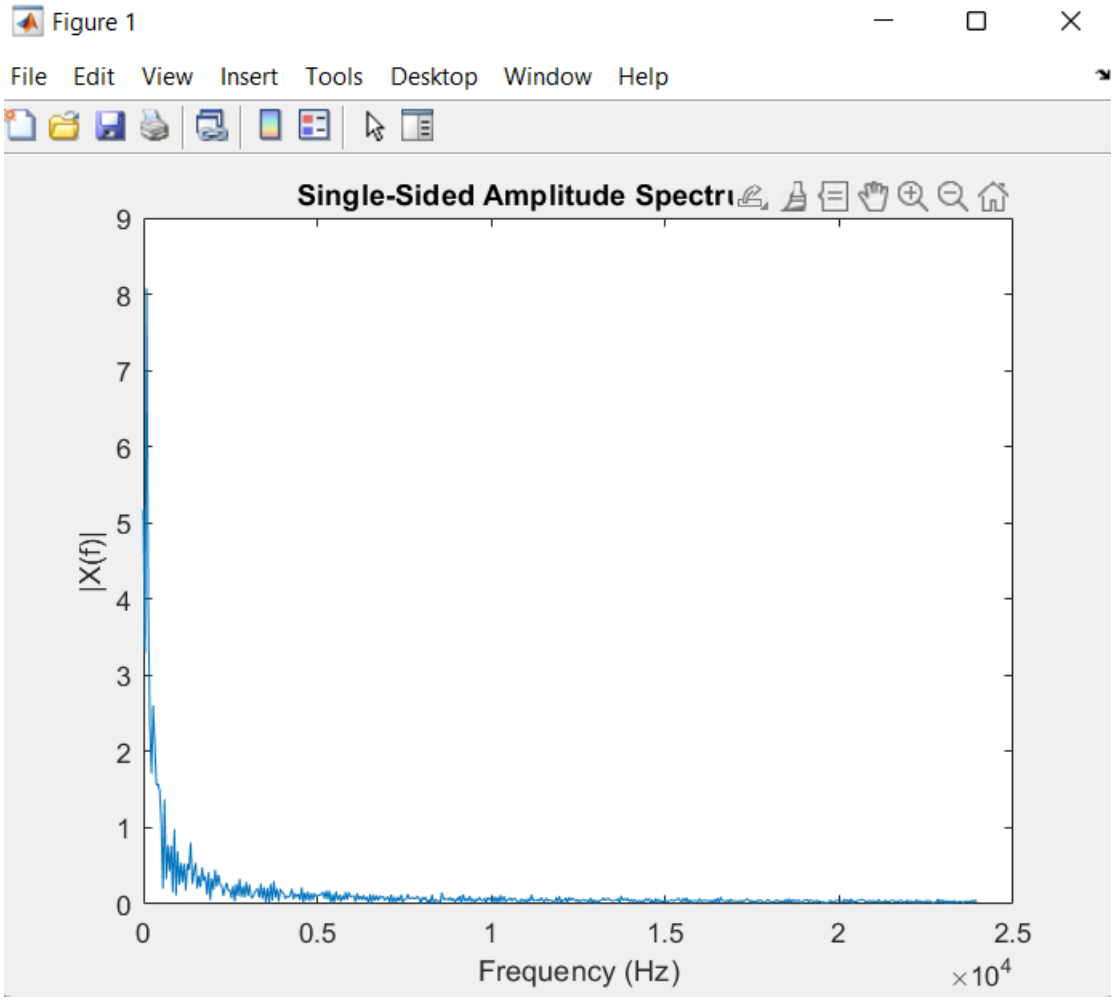


Figure 1: Original Audio.wav

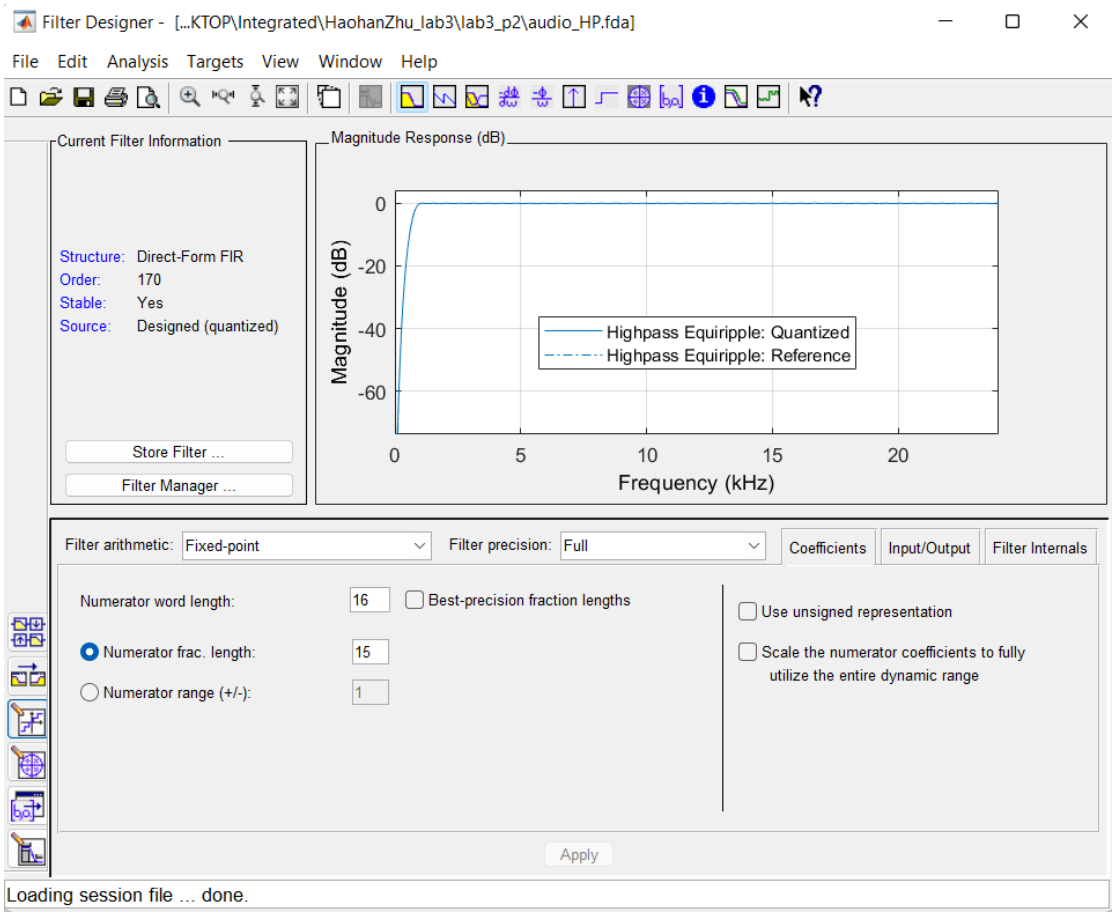


Figure 2: Designed High Pass filter.

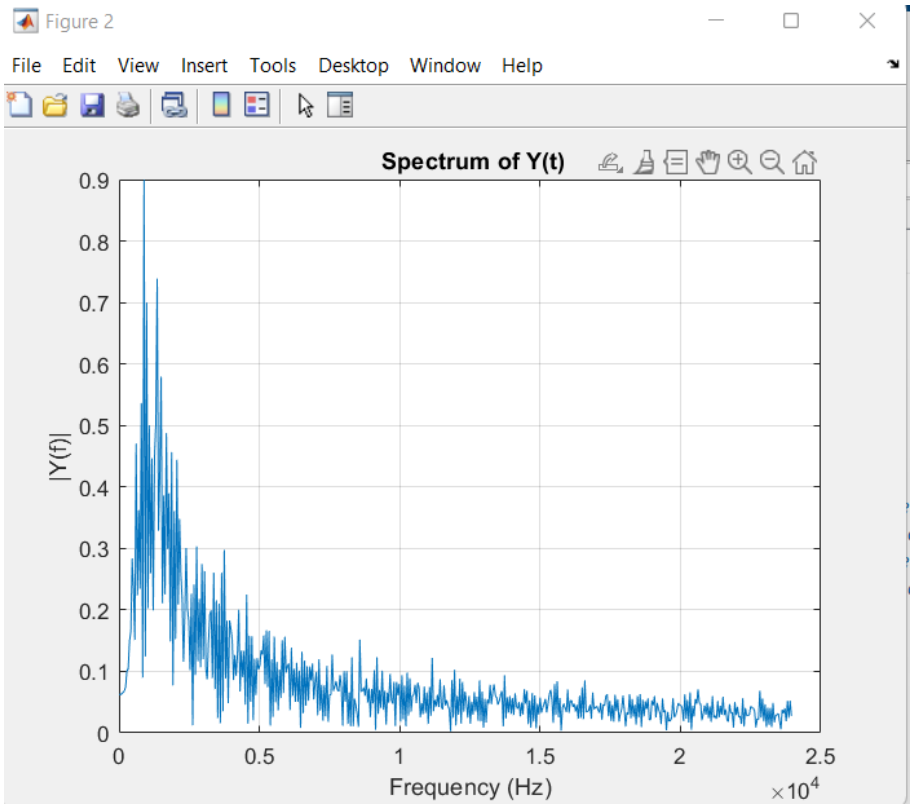


Figure 3: Clean Audio

2. The role of each IP in the overlay

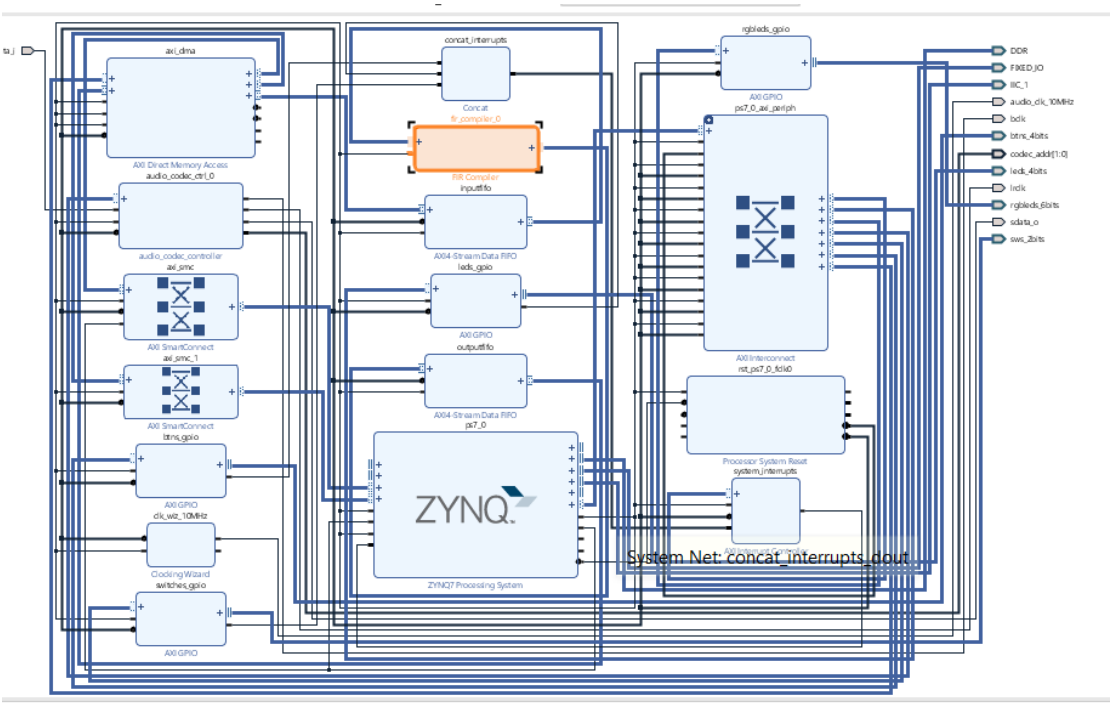


Figure 4: Overlay

AXI Direct Memory Access: AXI DMA provides high bandwidth direct memory access between memory and the AXI4-Stream target peripheral. Its optional scatter/gather function also offloads data movement tasks from the central processing unit (CPU).

Audio_codec_controller: An audio codec device can digitize an analog audio signal and convert the digitized signal back to analog format.

AXI SmartConnect: AXI SmartConnect core connects one or more AXI memory-mapped master devices to one or more memory-mapped slave devices. AXI SmartConnect is a drop-in replacement for the AXI Interconnect v2 core. AXI SmartConnect is more tightly integrated into the Vivado design environment to automatically configure and adapt to connected AXI master and slave IP with minimal user intervention.

AXI GPIO: The AXI GPIO provides a general purpose input/output interface to the AXI (Advanced eXtensible Interface) interface. This 32-bit soft IP core is designed to interface with the AXI4-Lite interface

Clocking Wizard: Clocking Wizard generates HDL source code to configure a clock circuit to user requirements. The wizard can either automatically select an appropriate clocking primitive and configure buffering, feedback, and timing parameters for a clocking network, or help the user configure the attributes for a manually selected primitive. If desired, the user may also override any wizard-calculated parameter. Besides generating source HDL for the clocking circuit, the wizard also invokes the Xilinx timing analysis tools to generate a timing parameter report.

Concat: The Concat IP core is used for concatenating bus signals of varying widths.

FIR complier: The Finite Impulse Response (FIR) Filter is one of the most ubiquitous and fundamental building blocks in DSP systems. Although its algorithm is extremely simple, the

variants on the implementation specifics can be immense and a large time sink for hardware engineers today, especially in filter-dominated systems like Digital Radios.The FIR Compiler reduces filter implementation time to the push of a button, while also providing users with the ability to make trade-offs between differing hardware architectures of their FIR Filter specification.

AXI4-Stream Data FIFO: The AXI Streaming Data FIFO allows memory mapped access to a AXI Streaming interface. The core can be used to interface to the AXI Ethernet without the need to use DMA. The principal operation of this core allows the write or read of data packets to or from a device without any concern over the AXI Streaming interface. The AXI Streaming interface is transparent to the user.

ZYNQ7 Processing System: Xilinx provides the Processing System IP Wrapper for the Zynq®-7000 to accelerate design and its configuration for embedded products

AXI Interconnect: The AXI Interconnect IP connects one or more AXI memory-mapped Master devices to one or more memory-mapped Slave devices. The AXI interfaces conform to the AMBA® AXI version 4 specifications from ARM®, including the AXI4-Lite control register interface subset. The Interconnect IP is intended for memory-mapped transfers only; AXI4-Stream transfers are not applicable. The AXI Interconnect IP can be used from the Vivado® IP catalog as a pcore from the Embedded Development ToolKit (EDK) or as a standalone core from the CORE Generator™ IP catalog.

Processor System Reset: The Xilinx Processor System Reset Module design allows the customer to tailor the design to suit their application by setting certain parameters to enable/disable features. The parameterizable features of the design are discussed in Processor System Reset Module Design Parameters.

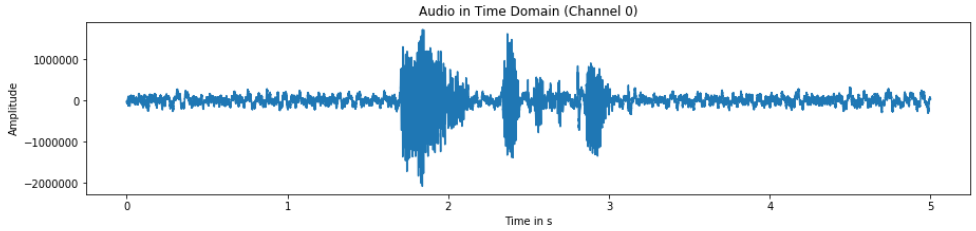
AXI Interrupt Controller: The LogiCORE™ IP AXI Interrupt Controller (AXI INTC) core concentrates multiple interrupt inputs from peripheral devices to a single interrupt output to the system processor. The registers used for checking, enabling, and acknowledging interrupts are accessed through a slave interface for the AMBA® protocol's AXI (Advanced Micro controller Bus Architecture Advanced eXtensible Interface) specification. The number of interrupts and other aspects can be tailored to the target system. This AXI INTC core is designed to interface with the AXI4-Lite protocol.

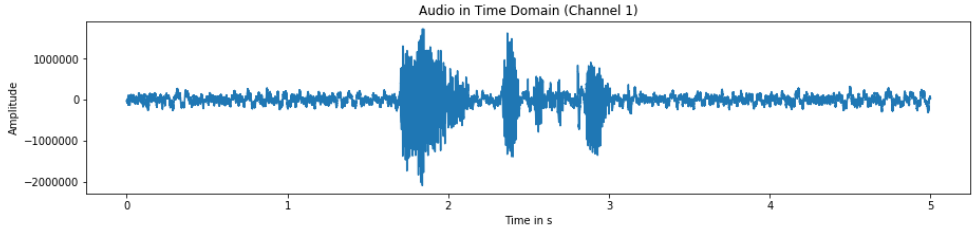
3. Observation

This is original audio waveform.

Plot the original audio from the 2 channels

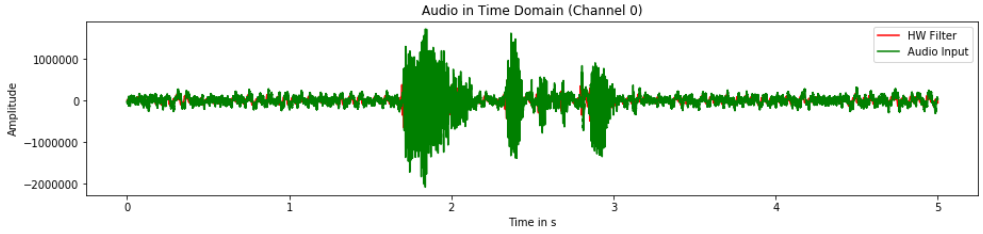
In [22]:
```
for channel_index in range(num_channels):
    plt.figure(num=None, figsize=(15, 3))
    plt.title('Audio in Time Domain (Channel {})'.format(channel_index))
    plt.xlabel('Time in s')
    plt.ylabel('Amplitude')
    time_axis = np.arange(0, num_frames/sample_rate, 1/sample_rate)
    plt.plot(time_axis, frames[:, channel_index])
    plt.show()
```

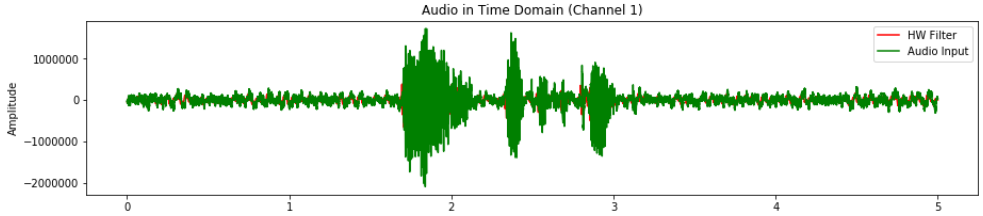




Compare Filtered Signal with Original Audio

In [37]:
```
for channel_index in range(num_channels):
    plt.figure(num=None, figsize=(15, 3))
    plt.title('Audio in Time Domain (Channel {})'.format(channel_index))
    plt.xlabel('Time in s')
    plt.ylabel('Amplitude')
    time_axis = np.arange(0, num_frames/sample_rate, 1/sample_rate)
    plt.plot(time_axis, frames_out[:, channel_index], 'r-', label="HW Filter")
    plt.plot(time_axis, frames[:,channel_index], 'g-', label="Audio Input")
    plt.legend()
    plt.show()
```





Hardware execution time:

Determine the hardware execution time

In [17]:
```
hw_exec_time = et - st
print("Hardware Execution time is: ",hw_exec_time)
```

```
Hardware Execution time is:  0.4232809543609619
```

Software execution time:

Calculate software execution time

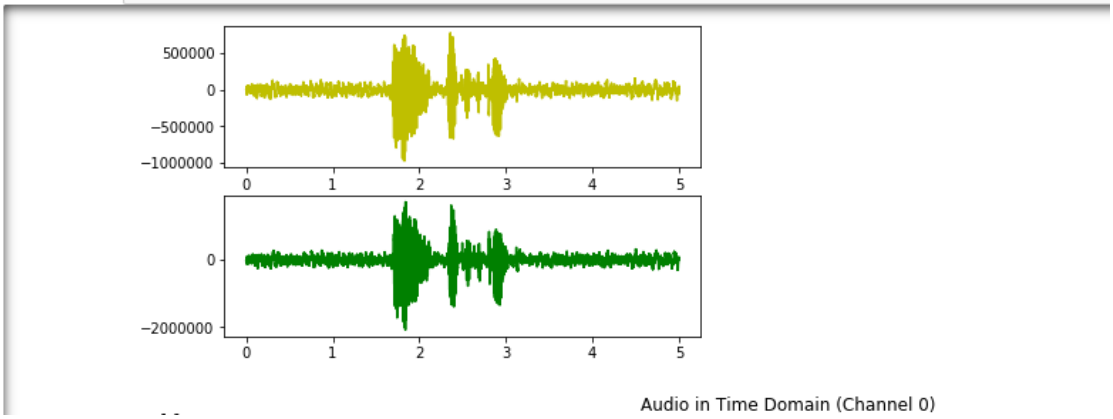
n [26]:
```
st = time.time()
swfilter = lfilter(coeffs, 128e3, inbuffer)
et = time.time()
sw_exec_time = et - st
print ("Software execution time: ", sw_exec_time)
```

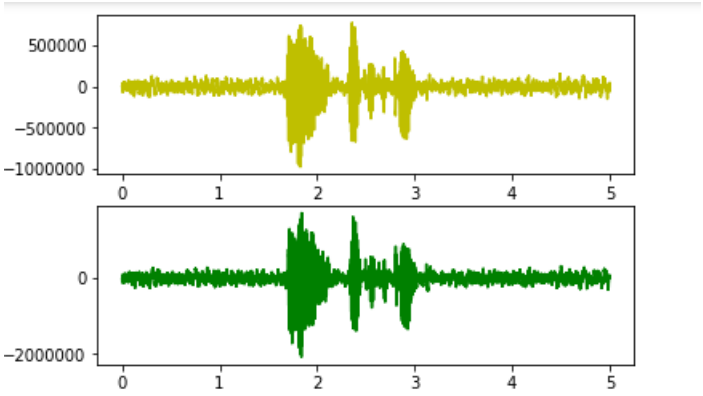
```
Software execution time:  0.6583516597747803
```

Software implementation channel 0 and 1:

Plot the software implementation result

In [36]:
```
for channel_index in range(num_channels):
    fig, axs = plt.subplots(2)
    plt.figure(num=None, figsize=(15, 3))
    plt.title('Audio in Time Domain (Channel {})'.format(channel_index))
    plt.xlabel('Time in s')
    plt.ylabel('Amplitude')
    time_axis = np.arange(0, num_frames/sample_rate, 1/sample_rate)
    axs[0].plot(time_axis, frames_out[:,0], 'y-', label="SW Filter")
    axs[1].plot(time_axis, frames[:,0], 'g-', label = "Input Audio")
    plt.show()
```





4. Compare the trade-offs between implementing the filter in hardware verses software

The results of the filter implementation running on software and hardware are not very different. The software execution time depends on the device being run, while the hardware execution time depends on a number of factors such as architecture design, power consumption, device performance and so on. However, if a simpler design is run, the execution time in hardware will be shorter than the execution time in software. In addition, the software run may result in a more desirable situation. The hardware may absorb some external noise and add it to the signal, so we can see a slight difference between the two in the hardware and software comparison graph.

The key factor is that the high-capacity hardware filter will have a lower cost and power requirement. The software version will be flexible and cheaper at smaller volumes. Designers will need to ensure that the software version meets latency and performance requirements and that the CPU is fast enough to run it and perform all other operations.