In my design, I believe that my custom AXI FIFO peripheral is operating as intended. The custom AXI FIFO peripheral consists of a default AXI 4 Lite interface for PS7 register space, and then an AXI 4 Stream Data FIFO to allow for data capture from the Custom Radio peripheral from lab 6.   
  
The AXI 4 Stream Data FIFO is configured with Independent clocks, and enabled the read and write counts. The AXI 4 Stream Data FIFO accepts the direct AXI 4 Stream input from the Radio peripheral, on the same PL Clock domain (125 MHz from the board). The of the Data FIFO is sent to the AXI Lite register space via signals internal to the Custom FIFO peripheral. The data is clocked out of the Data FIFO at the AXI Lite clock rate (AXI clock from the Zynq 7 at 125 MHz).   
  
The important signals to the AXI Lite interface are the read count and the AXI Lite 'tready' signal. The tready signal follows that of the default slv\_rden signal of the AXI Lite interface. The data is then clocked into register 0 every single time that the slv\_rden signal is high, which is whenever the PS7 performs a valid read, and the read count is also written to register 1. Thus, whenever the Processor reads a register, new data is accepted and the read count is available, so software can see how many samples are available.

My C code follows the rough format for a UDP Client from the code provided at

[https://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/#](https://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/)

The modifications I made are to form my own packet from the data in the custom FIFO peripheral registers. We know from the provided MATLAB script that each packet is expected to have 256 total complex samples, which consists of 2 2byte samples, resulting in a payload of 1024 bytes. The MATLAB Script also expects each packet to have a 2 byte string of counter information to see if there are any missed packets. Thus, software creates a simple counter at the beginning of a packet, and then reads the custom FIFO peripheral register 1 to see if there is data available, then checks if the data is new, and adds the new data to a byte array until a full 256 complex samples have been collected, and then sends a packet. This process will continue until a key is struck.

When using Wireshark, it is clear that the rate at which packets are received by the server closely matched that of the expected sample rate of the DAC. This is calculated by recording how many packets are sent/received in a tenth of a second, multiplying by 10, and then 256 for the samples in the packet. In a tenth of a second, there are between 18-20 samples sent, which puts the packet rate in the ballpark of the expected DAC sample rate of ~48 KHz. Software controls the sample rate / packet rate by waiting for a new piece of data from the register before adding it to the packet.   
  
However, when looking at the FFT plot in MATLAB, there appears to be a lot of noise, and the frequency peak does not match the expected frequency that is being output.

To troubleshoot, I did the following:

* Created an ILA that will watch the output of the radio system, and the latched data signal from low level DAC interface
* Since latched data signal is only 1 clock width, I set a capture control when latched data is 1 (or rising edge),  to observe each unique value of the data stream.
* Set the frequency to 6 KHz, tuning frequency of 0 Hz, bypassed the FIR Filter for raw data output
* Exported the ILA data and made some C functions to read the ILA data from file and create ethernet packets

From there, I modified software to get somewhat of a response on the MATLAB plot. It looks like the 6 Khz is showing up, but there was an additional spike at 42 KHz.

I double checked the endianness of my ethernet packets, and everything seemed to be lined up as expected.

Towards the end, I was able to figure *some* of it out - I had everything as unsigned values, and changing to signed helped me see my FFT frequency spike change accordingly; however, it seems like the offset is in the wrong direction. For instance, if I expect to see a frequency of 15 KHz, I actually see a frequency of 35kHz. Essentially, however far the signal is from the 25Khz middle point of the graph seems to be correct, just in the wrong direction.