Technical Note **LIGO-T1800184–v4** 2018/12/14

Real-Time Universal Transfer Function Synthesizer

SURF Student: Izabella Pastrana — Mentors: Johannes Eichholz, Rana Adhikari, Christopher Wipf

1 Background

On September 14, 2015, at 09:50:45 UTC, the Laser Interferometer Gravitational-Wave Observatory (LIGO) directly detected gravitational waves for the first time since Albert Einstein predicted their existence in his 1916 theory of General Relativity. According to the theory, gravity is a consequence of the curvature of space-time, and gravitational radiation can ripple out from fluctuating mass-energy distributions at the speed of light. Far from these fluctuating sources, one can express the effect of gravitational waves as small perturbations to the otherwise flat space-time background [1]. Such small perturbations are at the heart of LIGO's process for detecting gravitational waves. The LIGO detectors are able to detect differential changes in interferometer arm length on the scale of 10^{-19}m, a fraction of the size of a proton.

In order to detect such minuscule signals, LIGO employs 4 km baseline Michelson interferometers, with Fabry-Perot cavity enhanced arms, at two facilities located in Hanford, Washington and Livingston, Louisiana. At low signal frequencies, where they actuate on the test masses, the gravitational wave signal is partially canceled by the control loops for test mass suspension. This partial cancellation does not mean the signal is lost, however, as it just now has to be combined from different data streams. Meanwhile, at higher frequencies, there will be an optical signal at the interferometer output, so the true gravitational wave signal has to be combined from the optical signal and the interferometer feedback. Either way, this detection scheme thus requires the utilization of feedback control systems. A *control system* is a machine whose output follows an input that experimenters control, while *feedback* describes the strategy by which the difference between the system's actual output and the desired output is used to adjust the system into the desired state [2].

To this end, Advanced LIGO's Control and Data System (CDS) design includes a number of computers in a distributed computing architecture to perform real-time (deterministic) interferometer control, monitoring, and data acquisition tasks [3]. One of the system's most important features is its ability to implement minimally intrusive changes in the control loop landscape via and in-situ parameter redefinitions such as filter order and coefficients. However, the existing global emulated real-time system utilizes analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) clocked slowly at maximum 64 kHz. These frequencies are suitable for slow processes, like temperature and suspension controls, but faster processes require higher bandwidths and are implemented using analog-only electronics. Processes that would benefit from higher clocking frequencies—and thus high frequency transfer function measurements—include measurements of down-conversion from radio frequency (RF) down to audio frequency (AF), as well as the monitoring of various analog control loops, such as the control loops governing the laser amplitude servo, laser frequency servo, and oscillator noise coupling. These control loops would not be made digital, but it could be advantageous for debugging purposes to monitor the loop signals at higher sampling rates offered by higher clocking frequencies.

2 Objectives

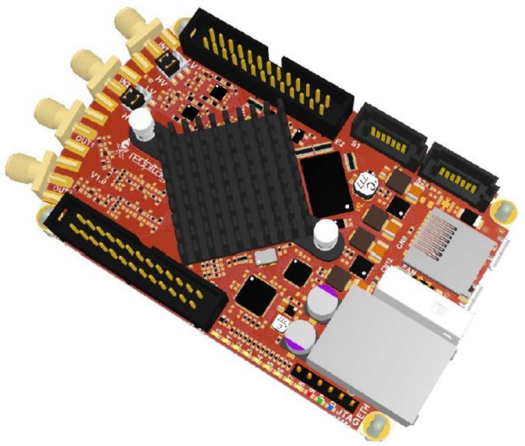
One way to increase clock rate in the control loop system is to use programmable fast-clocked real-time hardware that can interface with the original configuration tools, thus compromising between the flexibility of digital logic and the speed of analog circuits. The primary goal of this project is to assemble a near-arbitrary transfer function synthesizer with noise-shaping that can be used in control loops and physically instantiate simulated plants for loop analysis. The transfer functions synthesized would either imitate physical systems or be used to make servo controllers. For the former, precise transfer function measurements are required, with adaptive resolution if necessary, in order to accurately model control loops. Adaptive resolution refers to the idea in which initial transfer function measurements could be made quickly but with coarse resolution, and upon analysis of the resulting data, higher resolution transfer functions may be made at only the frequency bins where more resolution is needed. This adaptive structure would allow researchers to measure in high resolution just where the transfer function has features, rather than wasting time and energy making high resolution transfer function measurements even when they are not necessary. This will make more efficient the measurement of accurate transfer functions which maximize the amount of information learned about a system's control parameters [5], thus facilitating optimal system design [6]. However, for the transfer function synthesizer to be useful for LIGO, we must also develop a way to reference it to the same timing standard as the LIGO CDS.

Having achieved the principal objectives, the next task would be to design and implement high bandwidth real-time control loops for actual application—perhaps in the offset-phaselock between two lasers or their amplitude stabilization, as these would benefit from a high bandwidth which the existing real-time system is ill equipped to supply—and find the maximum bandwidth achievable with them. A tertiary goal is to explore machine learning avenues for the transfer function synthesizer. For example, we may increase feedback parameters while observing servo oscillations and minimize various noise in the control loop.

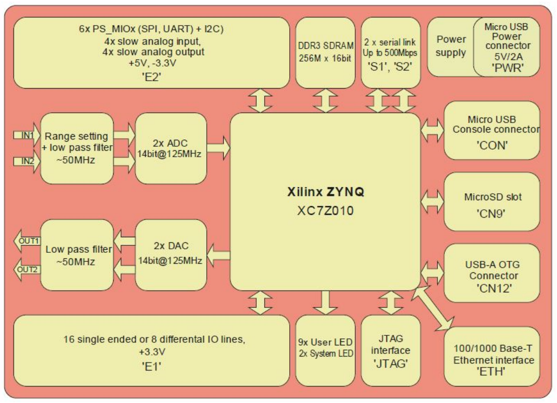
3 Methods

We have selected the Red Pitaya as the programmable fast-clocked real-time hardware for this project (see Figure 1). The Red Pitaya is an open source data acquisition and signal generation platform whose processor is one-half field-programmable gate array (FPGA) and one-half central processing unit (CPU) [7]. The FPGA is a Xilinx Zynq 7010 and is programmable using Verilog and Matlab Simulink while the CPU is a dual core cortex-ARM A9. Xilinx further offers free software packages to compile FPGA cores for the Red Pitaya. However, as there exists only a limited number of system resources, in the form of logical gates, we may reach a point where our design must compromise between accuracy and speed of the computations.

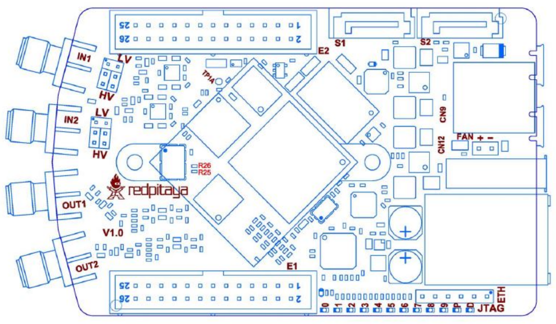
We will begin by measuring and modeling the transfer function of a known device, such as a Mini-Circuits low-pass filter, using first a conventional network analyzer and then the Red Pitaya. This will inform us of how the Red Pitaya performs as a transfer function measurement device, which is key in adapting it to eventually perform as a transfer function synthesizer. This step will further begin the process of creating an algorithm to make



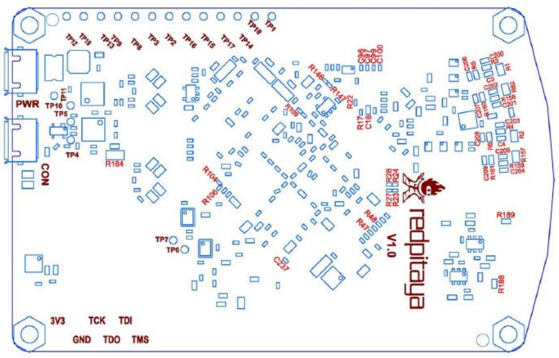
(a) The Red Pitaya


6x PS_MIOx (SPI, UART) + I2C)
4x slow analog input,
4x slow analog output
+5V, -3.3V
'E2'
DDR3 SDRAM 256M x 16bit
2 x serial link Up to 500Mbps 'S1', 'S2'
Power supply
Micro USB Power connector 5V/2A 'PWR'
IN1
IN2
Range setting + low pass filter ~50MHz
2x ADC 14bit@125MHz
Xilinx ZYNQ XC7Z010
Micro USB Console connector 'CON'
MicroSD slot 'CN9'
OUT1
OUT2
Low pass filter ~50MHz
2x DAC 14bit@125MHz
USB-A OTG Connector 'CN12'
16 single ended or 8 differental IO lines, +3.3V 'E1'
9x User LED 2x System LED
JTAG interface 'JTAG'
100/1000 Base-T Ethernet interface 'ETH'

(b) Block diagram of the Red Pitaya



(c) Top assembly of the Red Pitaya



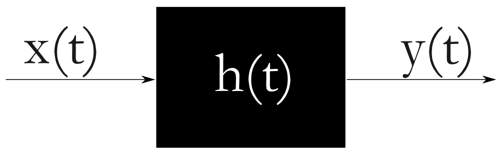
(d) Bottom assembly of the Red Pitaya

Figure 1: Red Pitaya's open source development schematics[8]

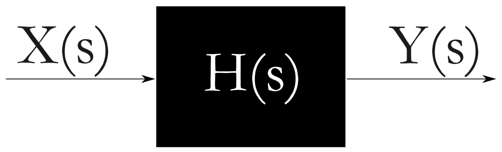
accurate models of measured transfer functions such that the Red Pitaya may synthesize and simulate them later.

Then, in order to actually design the Red Pitaya as a transfer function synthesizer, we will characterize the Red Pitaya (including the board's input noise, output noise, and transfer function characteristics). We will test what filters are possible to use on the board using conventional low-noise network and spectrum analyzers. The next step is to evaluate the possible filter complexity based on available system resources, paying particular attention to the trade-offs between infinite impulse response (IIR) and finite impulse response (FIR) filter architectures. FIR filters are far more resource-intensive than IIR filters, but IIR filters can have stability issues, particularly due to quantization errors.

Once we have characterized the board, we may then interface the Red Pitaya with other systems, such as lasers, photodetectors, etc. The ADC clock on the Red Pitaya FPGA can be provided by the default clock source on board, a 125 MHz oscillator, but as we ultimately aim to synchronize the Red Pitaya with the LIGO Data Acquisition, Network and Supervisory Control (DAQ), it is important to learn how to synchronize the FPGA with an external clock. There are instructions in the Red Pitaya documentation describing how to override the internal with an external clock [9], so the task reduces to designing a simple



(a) Black box diagram of a system in the time domain.



(b) Block diagram of a system in the Laplace domain.

Figure 2: Black box diagrams

driver circuit for the clock signal that is then fed into the Red Pitaya through one of its pin headers. One possible way to design the driver circuit is to have it convert an AC-coupled 125 MHz sine wave to an LVDS square wave to drive the ADC clock. Alternatively, we may look into locking an internal FPGA-generated clock to a 10MHz reference signal coming from the LIGO timing distribution system.

4 Transfer Functions

Transfer functions exist as compact descriptions of the relation between the input and output of a system in the Laplace/Fourier domain. Take, for example, some system in the time domain with an impulse response $h(t)$, an input signal $x(t)$, and an output signal $y(t)$, as shown in Figure 2a. In the time domain, the output $y(t)$ would be given by the convolution of the input $x(t)$ and the impulse response $h(t)$. Convolutions and differential equations, however, can quickly get unwieldy as the system increases in complexity.

But if we take the Laplace transform of the same system, we obtain $X(s)$, $H(s)$, and $Y(s)$, as shown in Figure 2b, where $s = \sigma \pm i\omega$ is the standard Laplace variable. Now, rather than having to grapple with convolutions, $Y(s)$ is given by the simple multiplication of $X(s)$ and $H(s)$. The output $y(t)$ in the time domain can then be calculated by taking the inverse Laplace transform of $Y(s)$. Thus, $H(s)$, known as the transfer function, is the Laplace transform of the impulse response $h(t)$ of a linear time-invariant (LTI) system when initial conditions are set to zero.

One common way to write the transfer function $H(s)$ is in what is known as pole-zero form. This form is written such that the zeros (the values of s that make $H(s) = 0$), the poles (the values of s that make $H(s)$ undefined), and the steady-state gain K are easily read, as in:

$$H(s) = K\frac{(s-z_1)(s-z_2)...(s-z_{n-1})(s-z_n)}{(s-p_1)(s-p_2)...(s-p_{n-1})(s-p_n)} \tag{1}$$

where z_i indicate the complex-valued zeros and p_i indicate the complex-valued poles of the system. Varying the value of the gain factor K as well as the number and placement of zeros and poles changes the behavior of the system represented by the transfer function.

If we let $s = i\omega$, then we move into the Fourier domain, and we may now generate a Bode plot of the transfer function. A Bode plot is a useful visualization tool as they depict a system's gain and phase response as a function of frequency. The gain response plots the magnitude of the transfer function (typically in decibels) against logarithmic frequency (in Hz), while the phase response plots the phase of the transfer function (typically in degrees) against logarithmic frequency. Bode plots are formatted such that the logarithmic frequency is on the horizontal axis, while gain and phase are on the vertical axis (as can be seen, for example, in Figure 5).

4.1 Measuring Transfer Functions

A system's transfer function is often measured using a vector network analyzer (VNA) and displayed as Bode plots. As the transfer function data researchers take can be noisy due to limited drive magnitudes, depending on the system under measurement, we would like to develop a way to add credibility to the transfer function estimates. One way to do this is by recording individual sweeps, estimating the error at each frequency, and using those errors in the fitting process.

A VNA measures the frequency response of a system by the mechanism illustrated in Figure 3. The VNA internally generates a drive or excitation signal $A\sin(\omega t)$ at some frequency ω that gets passed out through the source (SRC) channel of the VNA. This signal passes through a 50/50 power splitter, sending half of the signal directly back into the VNA through one input channel (CH1) and sending the other half through the device under test (DUT), allowing the sampling of the excitation signal before and after the DUT. The signal that passes through the DUT is then sent back into the VNA through a second input channel (CH2). Both signals are then "mixed" within the network analyzer—that is, both are multiplied by a sine and a cosine signal—and the resulting signals are manipulated in the following way to derive the gain and phase of the system at the given frequency.

Using the notation from Figure 3, and where i = 1, 2,

$$C_i(\omega) = \frac{B_i}{2}[\sin(\phi_i) + \sin(2\omega t + \phi_i)] \tag{2}$$

$$D_i(\omega) = \frac{B_i}{2}[\cos(\phi_i) - \cos(2\omega t + \phi_i)] \tag{3}$$

The $\omega t + \phi_i$ terms in the above equations are both filtered away, allowing for the rearrangement of equations to get

$$C_i(\omega)^2 + D_i(\omega)^2 = \frac{B_i(\omega)^2}{4}(\sin^2(\phi_i) + \cos^2(\phi_i)) \tag{4}$$

This can then be manipulated into

$$C_i(\omega)^2 + D_i(\omega)^2 = \frac{B_i(\omega)^2}{4} \tag{5}$$

and

$$\frac{C_i(\omega)}{D_i(\omega)} = \frac{\frac{B_i(\omega)2}{\sin(\phi_i)}}{\frac{B_i(\omega)2}{\cos(\phi_i)}} \tag{6}$$

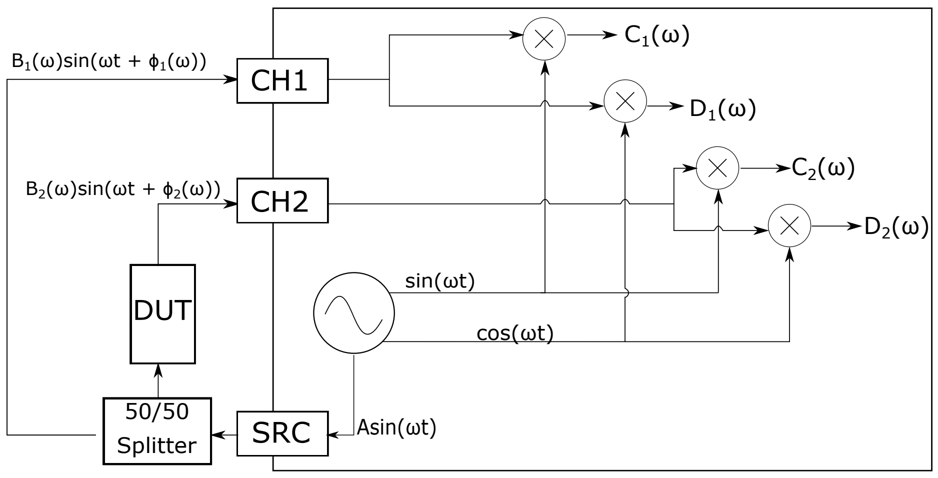


Figure 3: Block diagram of how transfer function measurements are collected.

Equation 5 rearranges to

$$B_i = \sqrt{4(C_i(\omega))^2 + D_i(\omega)^2)} \tag{7}$$

and the gain G of the system at that frequency ω will be given by

$$G = \frac{B_2(\omega)}{B_1(\omega)}. \tag{8}$$

Meanwhile, Equation 6 rearranges to

$$\phi_i = \arctan\left(\frac{C_i(\omega)}{D_i(\omega)}\right) \tag{9}$$

leading to the the system phase $\Delta\phi$ given by

$$\Delta\phi = \phi_2 - \phi_1 \tag{10}$$

The VNA then steps to the next frequency and repeats this process until all the data is collected. This data may then be displayed in a Bode plot.

4.2 Modeling Transfer Functions

Modeling measured transfer function data is an important task for the experimentalist as accurate models allow for better understanding of the system in question. To this end, I began by taking 25 transfer function measurements of a Mini-Circuits SLP-21.4 low-pass filter. Netgpib, a previously developed Python wrapper for the GPIB protocol, allows for a remote computer to interface with instruments using the GPIB-to-Ethernet devices attached to the instrument's GPIB port. I used this wrapper to communicate with the Hewlett Packard Agilent 4395A network analyzer (shown in Figure 4), and then created a bash script to collect 25 single-sweep frequency response measurements of the filter. The frequency range of the measurements was 1 kHz to 100 MHz with an IF bandwidth of 100 Hz and excitation signal amplitude of 0 dBm, sweeping 801 points. I next plotted the median Bode plot, shown

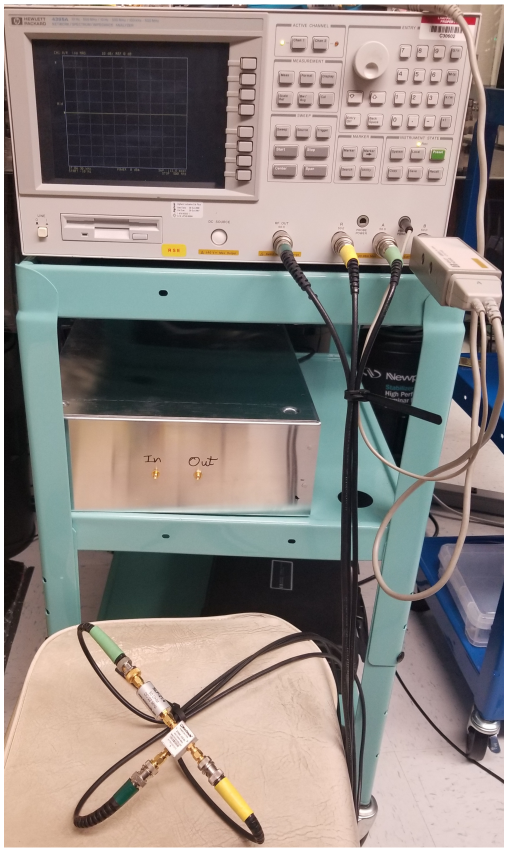


Figure 4: Experimental set-up: the Hewlett Packard Agilent 4395A vector network analyzer is connected to the Mini-Circuits SLP-21.4 low-pass filter for transfer function measurements.

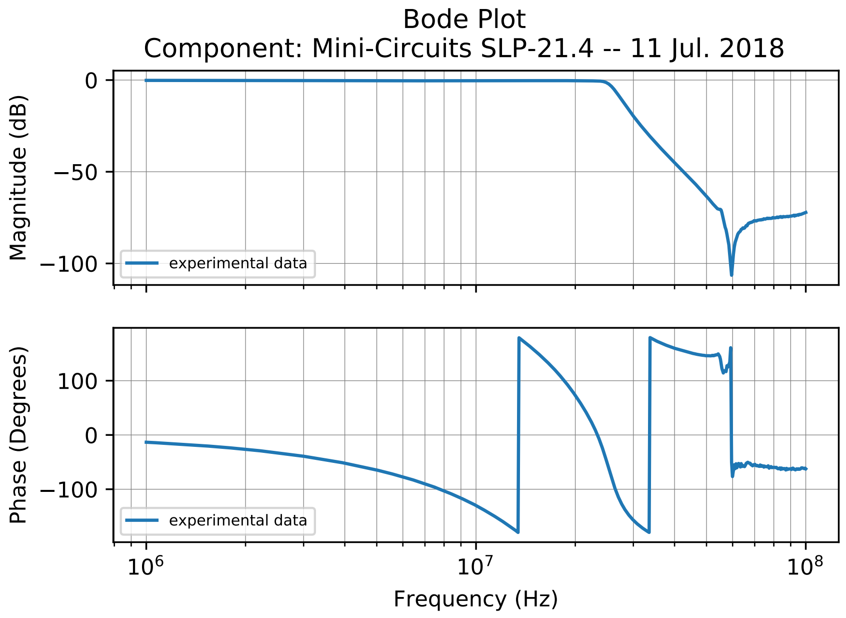
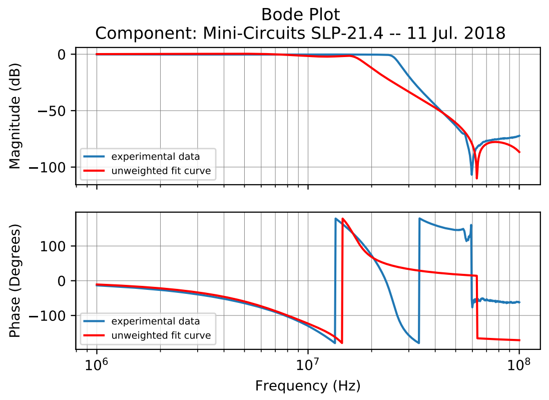
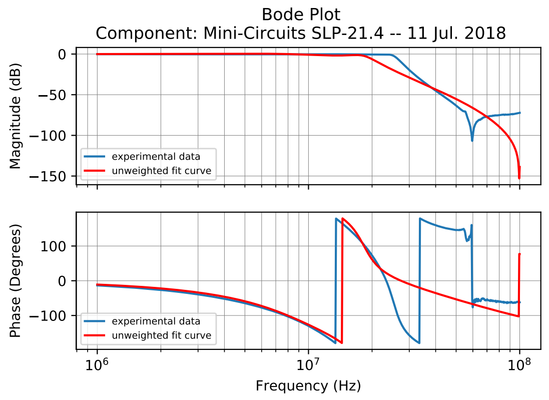


Figure 5: The median Bode plot of 25 transfer function measurements of the Mini-Circuits SLP-21.4 low-pass filter.



(a) 4th-order elliptic low-pass filter model, unweighted fit



(b) 5th-order elliptic low-pass filter model, unweighted fit

Figure 6: Comparison of unweighted fits to the data using 4th- and 5th- order elliptic filter models.

in Figure 5 of the 25 measurements. I plotted the median because it is more robust to data outliers compared to the mean.

Modeling the Mini-Circuits SLP-21.4 component I measured as a fourth-order elliptic filter, I modeled the transfer function as in Equation 11.

$$H(s) = K\frac{(s-z_1)(s-z_2)(s-z_3)(s-z_4)}{(s-p_1)(s-p_2)(s-p_3)(s-p_4)} \tag{11}$$

My code takes advantage of the Python signal processing package scipy.signal which allows me to pass in values for gain, zeros, and poles and use them to construct the resulting transfer function and Bode plots. I attempted to fit the model parameters (K, z_i, and p_i, taking care to allow z_i and p_i to be complex by treating their real and imaginary parts as independent fitting parameters) to the experimental data. Using the Python optimization package scipy.optimize, I defined the residual error as the absolute magnitude of the difference between the experimental and fit data, and the root mean squared error as the error function. This resulted in Figure 6a. However, the quality of the fit is very poor in the stop band and fails to resemble the measured transfer function closely in the transition region, as evidenced by the greater error between the fit and the experimental data. Possible reasons are insufficient optimization steps, poor initial parameter estimation, or the error estimation of complex valued functions.

It was also possible that the model function, though chosen based on the filter topology, does not allow for sufficient flexibility. Deeming that perhaps the fourth-order elliptic filter model I was trying to fit the transfer function to was incorrect, I modified my model to a fifth-order elliptic filter model by adding another pole parameter to the transfer function. This fitting process resulted in Figure 6b, which, though still imperfect, better matched the transition region of the experimental data than the fourth-order model.

As mentioned previously, to add credibility to the transfer function estimates, we could estimate the error at each frequency and use those errors in the fitting process. A transfer function and spectra plotting software called IRIS [10], developed by LIGO graduate student Craig Cahillane, proved helpful to this end. A particularly useful aspect of the IRIS software is its ability to find the complex median and uncertainty values for a series of transfer function sweeps. The uncertainty is assessed by first converting the data from magnitude gain and phase measurements to complex numbers with real and imaginary parts. The real and imaginary parts of the data are then used to define a basis matrix from which a covariance matrix is developed. The square root of the diagonal elements of the covariance matrix (converted back to magnitude and phase) are taken to be the magnitude and phase uncertainties.

I used IRIS to generate Figure 7, which shows the Bode plot with the median of the complex transfer function in the left column, and in the right column, the relative uncertainty of the magnitude and phase, with $\pm 1\sigma$ uncertainty.

I modified the original IRIS code so that I could extract the median points and uncertainty values from the IRIS software and write them to a file. To incorporate them into the fitting process, I multiplied the individual data point root mean square errors by their corresponding inverse uncertainties before summation. This gave more weight in the fitting process to the data points that had less uncertainty. Since I found that the fifth-order elliptic low-pass filter was a better model for the data than the fourth-order, I used the fifth-order filter model in my weighted fit procedure. The weighted fit of the fifth-order model, shown in Figure 8b, proved to be a much closer representation of the experimental data. Nevertheless, there remains room for improvement.

5 The Red Pitaya

There exists a trade-off in transfer function measurements: the more accurate you want your measurement over a particular bandwidth, the more time you will have to spend taking the data. The certainty of a transfer function made from noisy data, for example, may benefit from the averaging of repeated measurements or the longer integration of each data point. Both, however, require extra time spent on the measurement. One way to ease the problem of noisy data involves the weighted fitting of transfer function estimates explored in Section 4.2. If the measurement is considerably noisier in certain frequency bands, applying weights in the fitting routine can help improve the fit's accuracy, as points with higher uncertainties can be weighted less in the fitting process. For the accuracy of a single measurement, it is usually better to integrate at each data point for a longer time, but this does not produce a measure for the uncertainty. Taking the average of multiple, short, repeated measurements, however, does.

Feature-rich data poses a similar problem of accuracy vs. time. Rapidly changing—but not noisy—data requires increased frequency resolution to accurately resolve the transfer function, which translates to smaller frequency increments during the sweep, and thus more time performing the sweep. One way to optimize such data measurements would be to implement an adaptive algorithm which controls the step size depending on the rate of change in the transfer function. Such an algorithm would be able to execute a rough scan

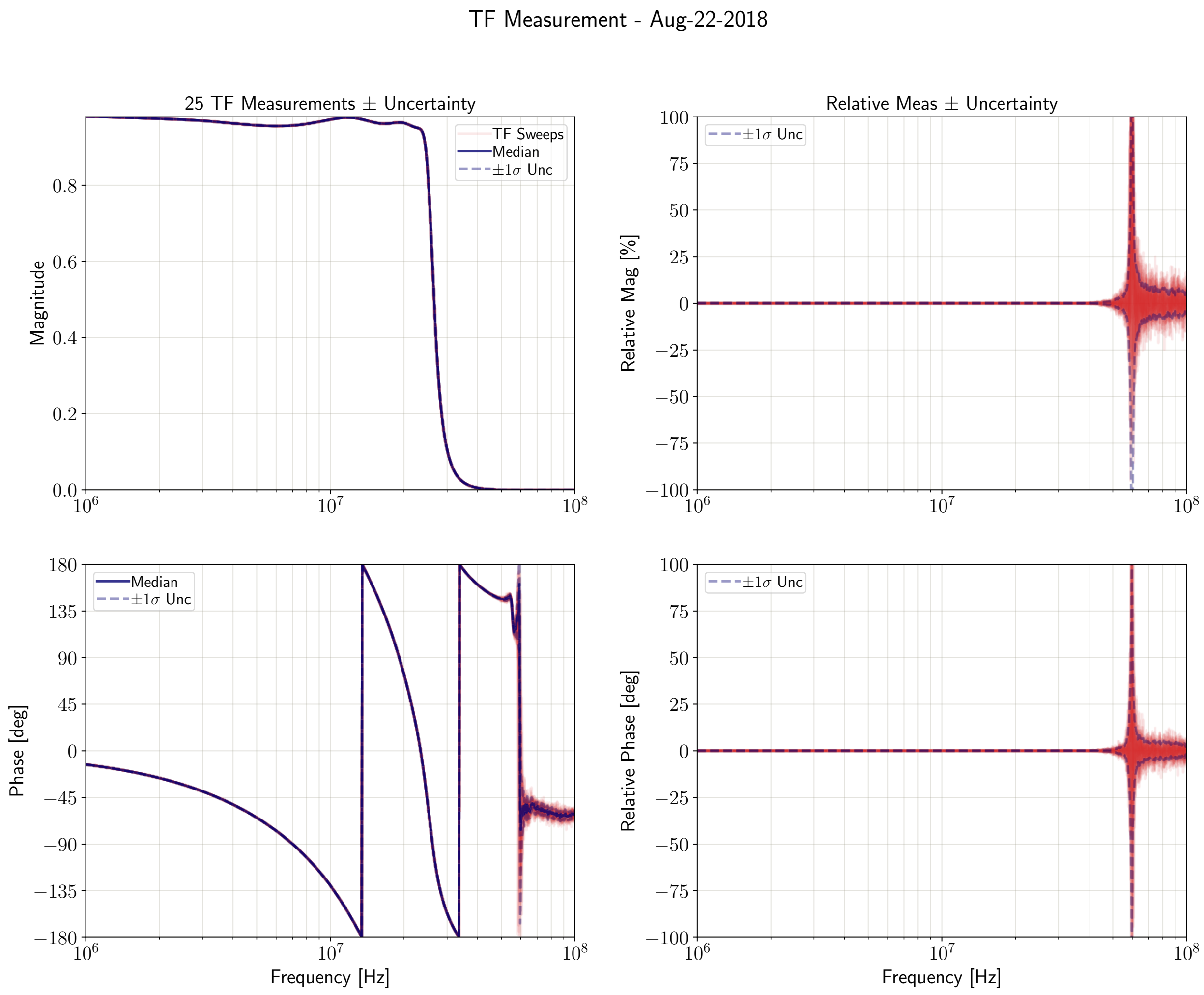
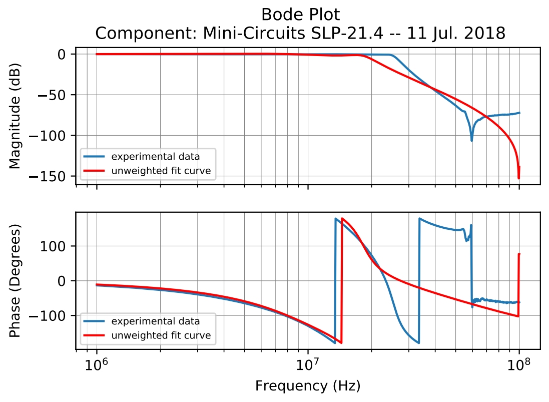
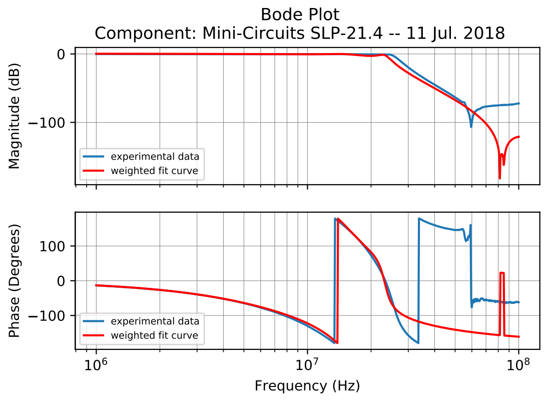


Figure 7: The IRIS generated "four square" plot, which shows the Bode plot with the median of the complex transfer function in the left column, and in the right column, the relative uncertainty of the magnitude and phase compared to the median, with $\pm 1\sigma$ uncertainty. Some portions of the curves are "blurry" due to the fact that IRIS plots all the of the 25 individual sweeps using a red transparency. The darker red the curve is, the more sweeps passed through that magnitude/phase. The solid blue curve indicates the median transfer function, and the dashed blue lines indicate the uncertainties. Note: the magnitude of this Bode plot is reported in relative amplitude, rather than in decibels.



(a) 5th-order elliptic low-pass filter model, unweighted fit



(b) 5th-order elliptic low-pass filter model, weighted fit

Figure 8: Unweighted vs. weighted fits to the data using a 5th-order elliptic filter model.

where there are not many features in the transfer function and increase the number of sample points only where there are rapidly changing regions in the data.

This adaptive sweeping capability means that the transfer function–measuring device will automatically identify regions of rapid change in a coarse initial transfer function measurement so that it can either go back to those specific frequency bins of interest and take higher resolution measurements there, or adapt its measurement resolution during a live sweep. Some existing network analyzers already have these auto-resolution features, such as the Hewlett Packard Agilent 3562A Dynamic Signal Analyzer, whose auto-resolution algorithm is illustrated in Figure 9. However, these features are hard-programmed into the device with little transparency or configurability to the actual programming of the device, i.e., users cannot directly access or manipulate the code governing the auto-resolution algorithm.

One could conceivably emulate auto-resolution functions in software and communicate it to the network analyzer, but the data transfer between older—but nevertheless expensive and difficult to replace—conventional network analyzers and host computers utilizes the General Purpose Interface Bus (GPIB) protocol. The GPIB consists of a ”bus on a cable,” providing a parallel data transfer path between an instrument and a host computer. The GPIB standard was first established in 1978 and had a nominal 1 MB/s maximum data transfer rate, which was perfectly adequate for the demands of the time and is even sufficient for many applications today [11]. Yet, by today's standards, the data transfer rate of the GPIB protocol is undesirably slow, creating a severe bottleneck as users wait for collected data to transfer completely.

Whatever measurement time one may save using an auto-resolution function (increasing the step size between consecutive data points based on small changes in value means less time wasted on unnecessary data measurements) will only be consumed by the slow rate of data transfer.

In certain situations, recording transfer functions with the Red Pitaya can address some of these difficulties. One benefit of using the Red Pitaya as a transfer function measurement

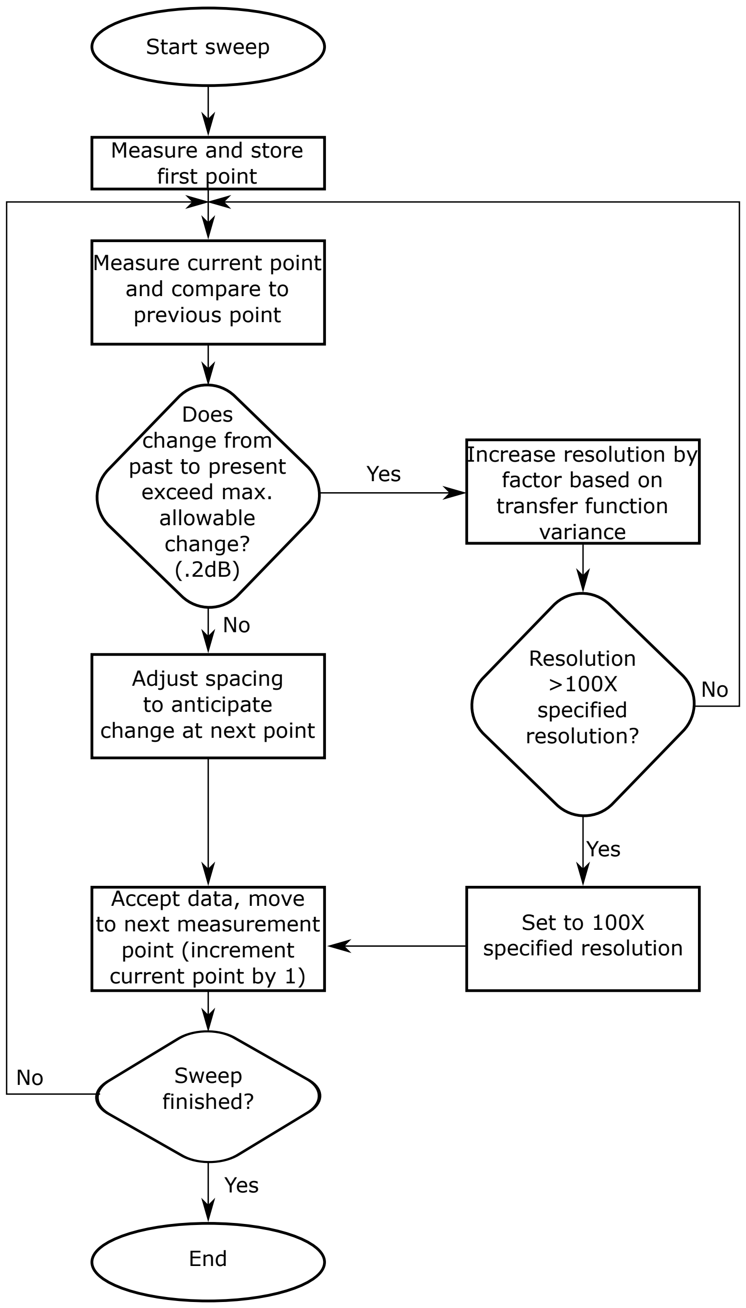

Start sweep
Measure and store first point
Measure current point and compare to previous point
Does change from past to present exceed max. allowable change? (.2dB)
Yes
Increase resolution by factor based on transfer function variance
No
Adjust spacing to anticipate change at next point
Resolution >100X specified resolution?
No
Yes
Set to 100X specified resolution
Accept data, move to next measurement point (increment current point by 1)
Sweep finished?
No
Yes
End

Figure 9: The flow diagram describing the auto-resolution function of the Hewlett Packard Agilent 3562A Dynamic Signal Analyzer (reproduced from the device manual) [16].

instrument is that the FPGA, which handles the signal demodulation at RF frequencies, shares access to the system RAM with the CPU portion of the chip, where an adaptive algorithm could adjust the sweep parameters in real-time. The Red Pitaya further supports data transfer via the Transmission Control Protocol / Internet Protocol (TCP/IP) as well as a gigabit Ethernet adapter, which is not limited to the timing standards of the GPIB protocol. In using the TCP/IP, the Red Pitaya effectively removes the data transfer bottleneck, and users may increase the repetition rate of their transfer function measurements. Though modern models of analyzers likely have innate Ethernet capabilities, the Red Pitaya is in principle more highly customizable by virtue of its readily programmable half-FPGA, half-CPU processor.

The problem now lies in programming the Red Pitaya's FPGA core to be able to operate as a transfer function–measuring device–the Red Pitaya must be able to generate a drive signal, coherently analyze two signals, and step through a range of frequencies to perform the demodulation described in Section 4.1. Further, additional control over the configuration of transfer function sweeps beyond those granted by conventional network analyzers would be particularly advantageous, such as an auto-resolution function. This is an enviable trait because, as evidenced in Figure 9, this existing auto-resolution functions only take into account changes in magnitude from one data point to the next. Such algorithms ultimately overlook the fact that transfer functions are complex by nature. In my planned auto-resolution algorithm for the Red Pitaya, I use changes in Euclidean distance between consecutive points to decide the need for measuring in higher resolution.

5.1 The Red Pitaya as a Transfer Function Measuring Device

Pavel Demin, an IT engineer at Université Catholique de Louvain, independently developed an open-source VNA application for the Red Pitaya that I used as a base off of which to work. Demin's VNA source folder [14] contains a TCL file for instantiating, configuring, and interconnecting all the necessary IP cores for the application on the Red Pitaya's FPGA, the specified block design for which is shown in Figure 10. In addition to the hardware, the software for the application is divided primarily into two parts: the source code for the client application, which consists of python code run on a remote host, and the source code for the TCP server, written in C and compiled for the Red Pitaya's ARM (Advanced RISC Machine) architecture. The server application maintains two TCP sockets, one over which it receives commands from the remote client, and a second on which it streams requested data. This data transfer chain is illustrated in Figure 11.

Demin's VNA application generates a drive signal from the output port OUT1 of the Red Pitaya, coherently analyzes two signals using the demodulation illustrated in 12, and steps through a range of frequencies. However, it steps only linearly through the frequency span, and it would be good to have the option of executing a log sweep over the frequency span as well. As Demin's VNA is set up such that the frequencies at which measurements will be taken are calculated Python-side before being passed into the TCP server as a single vector of frequencies, one could modify the instantiation of that frequency vector to step logarithmically—rather than linearly—throughout the specified span and pass to the TCP server this new logarithmic frequency vector instead of linearly-stepped one. Alternatively, it is possible to modify the Verilog modules that make up Demin's Red Pitaya FPGA design

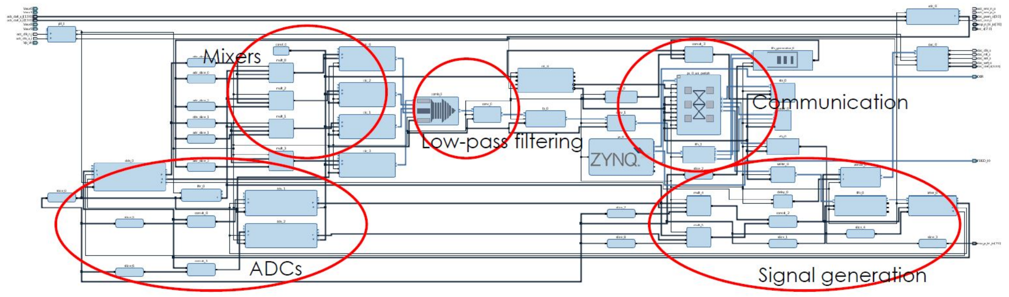


Figure 10: The block design of the Red Pitaya FPGA as visualized in the Xilinx Vivado Design Suite. The red circles and corresponding labels designate the regions of the block design that are associated with specific key functions of the Red Pitaya's transfer function–measurement capability.

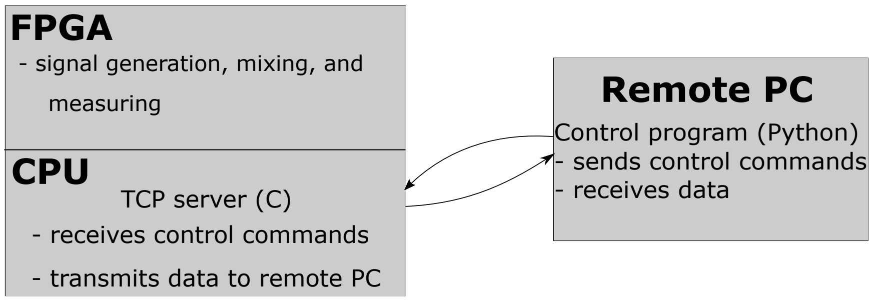

FPGA
- signal generation, mixing, and measuring
CPU
TCP server (C)
- receives control commands
- transmits data to remote PC
Remote PC
Control program (Python)
- sends control commands
- receives data

Figure 11: The data transfer chain between the remote PC and the Red Pitaya. The block on the left, divided into the FPGA and CPU halves, represents the Red Pitaya.

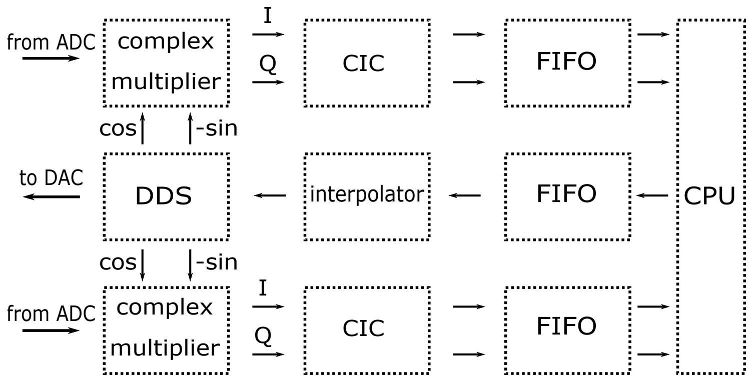


Figure 12: A basic block diagram of the signal demodulation process of the transfer function measurement system implemented in the Red Pitaya [13].

(making the necessary modifications to the TCP server as well to accommodate the module changes) such that there is more flexibility to the set of frequencies at which measurements will be taken. For example, instead of sending a largely inflexible vector of frequencies at the beginning of a measurement sweep, there could be live updates of these frequencies throughout the measurement.

Demin's VNA also writes out the raw data broken into its in-phase and quadrature components, which can be converted to real and imaginary data parts and thus used for the calculation of uncertainties. The default remote host Python code spawns a GUI application. Though the GUI is useful, it is easier to modify the program to run and take measurements if the application could execute and accept sweep parameter arguments from the command line compared to modifying and automating the GUI.

5.2 Developing Custom Functionalities for the Red Pitaya

I adapted Demin's code such that users can run the VNA application solely through the command line—bypassing the GUI—and trigger Red Pitaya measurements using syntax similar to that used to remotely trigger other analyzers (such as the Agilent 4395A). This involves passing sweep parameters as optional arguments for the modified Python script rpvna.py on the command line. The configurable sweep parameters are listed in Table 1.

Table 1: Configurable Sweep Parameters

Parameter	Minimum Value	Maximum Value	Default Value
Start Frequency (kHz)	1	60000	10
Stop Frequency (kHz)	1	60000	60000
IF Bandwidth (Hz)	1	10000	10000
Number of Points Measured	10	10000	6000
Signal Attenuation (dB)	0	90	0
Phase Offset (degrees)	0	360	0
Number of Sweeps to Measure	1	No limit	1
Linear or Log Sweep	–	–	Linear
Pass Data to IRIS	–	–	No

Late in the project, I discovered that the Red Pitaya transfer function measurements consistently observed odd wrapping in the phase measurements, as seen in Figure 13. This might perhaps be due to insufficient calibration of data, but until I could resolve this issue, I would run calculations on the raw data separate from the Red Pitaya.

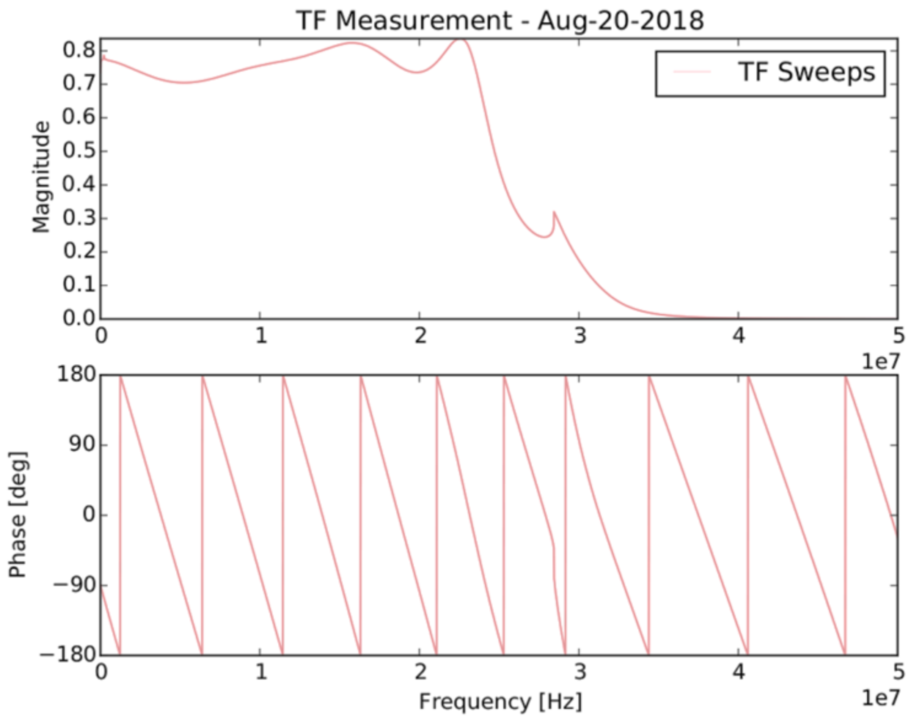


Figure 13: The Bode plot of the Red Pitaya transfer function measurement on the Mini-Circuits SLP-21.4 of Section 4.1. There is a peculiar wrapping in phase.

6 Conclusions and Next Steps

By summer's end, and in light of the odd phase wrapping in the Red Pitaya, I had only reached the point in my project where I could extract the raw data from the Red Pitaya and begin incorporating the data into my first attempts at an auto-resolution algorithm. The Red Pitaya as it currently exists cannot quite yet do adaptive sweeping as desired. Further work must be done to adapt the FPGA code to better fit the needs of auto-resolution.

Furthermore, a better transfer function fitting model needs to be developed so that we can take measurements, immediately estimate a transfer function for the system, and give back the transfer function zeros, poles, and gain. This particular functionality would be useful in the long-term goal of assembling a near-arbitrary transfer function synthesizer with noise-shaping that can be used in control loops and for physically instantiating simulated plants for loop analysis.

6.1 Potential Use at the LIGO 40 Meter Prototype Lab

Researchers at the LIGO 40 meter Prototype Lab at the California Institute of Technology have been utilizing a conventional VNA (the Agilent 4395A) to take spectroscopic scans of optical cavities in the interferometer arms. In these cavity scans, a phase-locked loop (PLL) maintains a controlled frequency difference between the pre-stabilized laser (PSL) and an auxiliary laser (AUX) such that the difference is equal to the frequency of the local oscillator

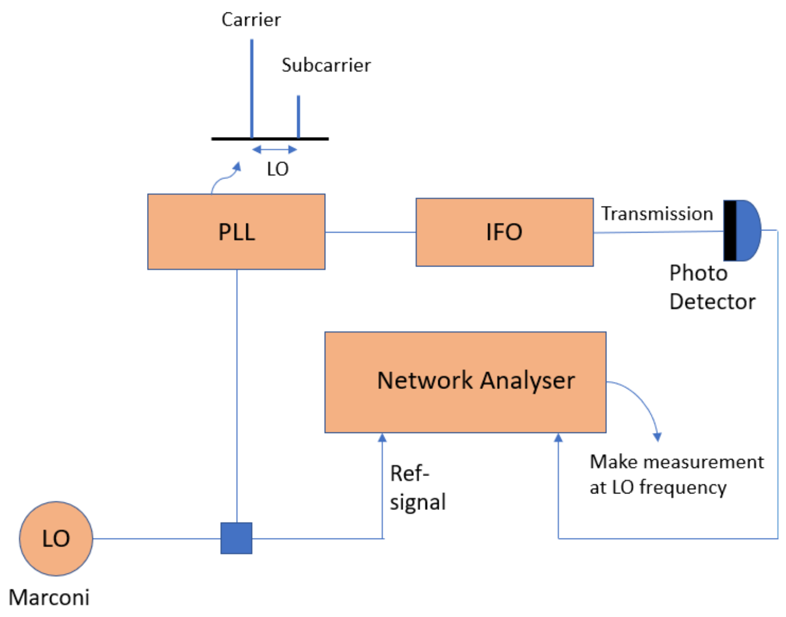


Figure 14: An example design of how arm cavity scans are taken at the 40 m Prototype Lab's inteferometer (labeled IFO). (reproduced from Keerthana S. Nair) [17]

(LO). The drive signal that a VNA generates during a usual transfer function measurement is used to sweep the RF offset of the AUX-PSL PLL and demodulate the cavity reflection or transmission signal at the offset frequency.

As it can be directly programmed to operate as desired, using the Red Pitaya as the VNA in the AUX laser scans—in lieu of using the Agilent, whose sweep configurations are limited by the manufacturer's design—would allow further control over these measurements, in addition to the benefit of the Red Pitaya's faster data transfer rate. The Red Pitaya VNA should be able to generate a drive signal at the specified start frequency of the sweep and hold there until a trigger that signals it to actually begin sweeping through frequencies. This hold at the start frequency will allow time for initial locking of the AUX and PSL lasers.

This modification would involve adapting the TCP server code such that the Red Pitaya may generate and hold a signal at the given start frequency, then wait for a trigger event before initiating actual sweeps. It would further be beneficial to code the Red Pitaya in such a way that it may swiftly sweep across frequencies between transmission intensity peaks and slow down for higher resolution measurements around the frequency neighborhoods of the peaks.

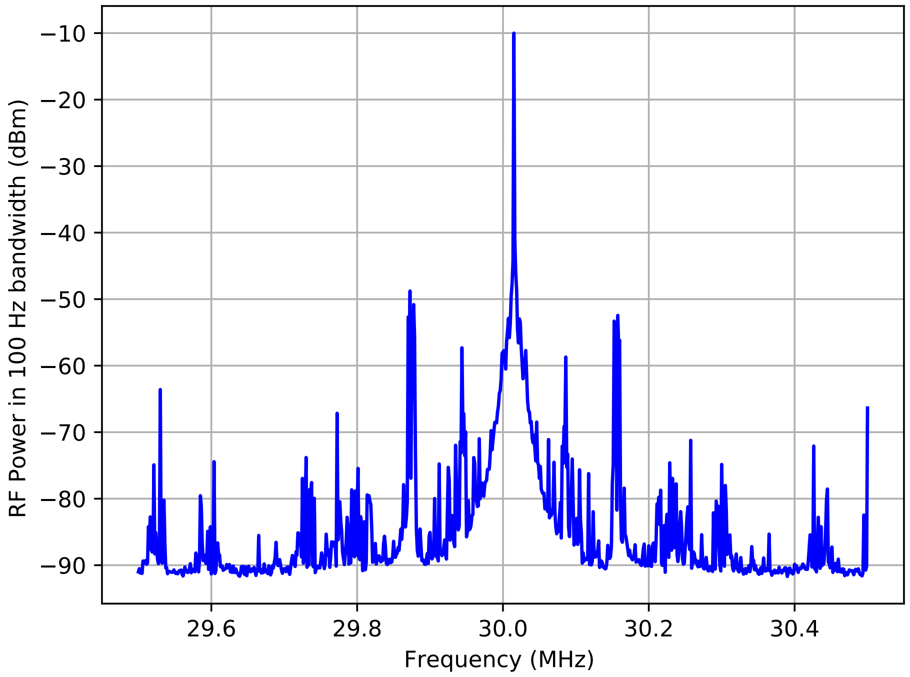


Figure 15: An example power spectrum for the 40 m interferometer's Y-arm reflection from the mirror. The peaks in the spectrum would benefit from higher resolution measurements. (reproduced from Keerthana S. Nair) [18]

References

[1] Rana Adhikari, *Sensitivity and Noise Analysis of 4 km Laser Interferometric Gravitational Wave Antennae*. Ph.D. thesis, MIT (2004).

[2] Peter Saulson, *Fundamentals of Interferometric Gravitational Wave Detectors*. World Scientific (2017).

[3] R. Bork and A. Ivanov. *aLIGO CDS Real-time Sequencer Software*. LIGO-T0900607-v3 (2010).

[4] Karl Johan Astrom and Richard M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers* (2008)

[5] E. D. Hall, *Fisher Matrix Methods for Transfer Function Measurement*. LIGO-T1500533-v1 (2015).

[6] Larry Price, *Optimal Experimental Design: Introduction and Application to System Identification*. LIGO-G1400084-v1 (2014).

[7] https://www.redpitaya.com/Catalog/p20/stemlab-125-14-starter-kit?cat=a99

[8] http://redpitaya.readthedocs.io/en/latest/developerGuide/125-14/shem.html

[9] http://redpitaya.readthedocs.io/en/latest/developerGuide/125-14/extADC.html

[10] https://git.ligo.org/40m/labutils/tree/master/iris

[11] https://www.omega.co.uk/temperature/z/overviewieee.html

[12] https://www.sqa.org.uk/e-learning/HardOSEss04CD/page_04.htm

[13] http://pavel-demin.github.io/red-pitaya-notes/vna/

[14] https://github.com/pavel-demin/red-pitaya-notes/tree/master/projects/vna

[15] https://github.com/CaltechExperimentalGravity/FastDigitalFiltering/tree/master/redPitaya/vna

[16] https://www.atecorp.com/ATECorp/media/pdfs/data-sheets/Agilent-3562A_Manual.pdf

[17] https://nodus.ligo.caltech.edu:8081/40m/13924

[18] https://nodus.ligo.caltech.edu:8081/40m/13995