NANOSECOND-LEVEL DATA ACQUISITION FOR EXPERIMENTAL ASTROPHYSICS

VINCENT WEILER
Bachelor of Science, University of Lethbridge, 2016

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Physics and Astronomy
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

NANOSECOND-LEVEL DATA ACQUISITION FOR EXPERIMENTAL ASTROPHYSICS

VINCENT WEILER

Date of Defence: July 12, 2019

Dr. Locke D. Spencer Thesis Supervisor	Associate Professor	Ph.D.
Dr. David Naylor Thesis Examination Committee Member	Professor	Ph.D.
Dr. Hadi Kharaghani Thesis Examination Committee Member	Professor	Ph.D.
Dr. Chad Povey Chair, Thesis Examination Committee	Instructor	Ph.D.

Dedication

The events of 2011 lead me to take a path less travelled.

To the new friends that were made on that path.

And the few that supported the journey.

Abstract

Linear translation stages are an integral part of Fourier spectroscopic and interferometric instruments; corresponding metrology data must be properly characterized to mitigate any deleterious effects in subsequent data processing and analysis. In order to profile the linear stages, micro-controllers, coarse counting timing circuits, and field programmable gate array solutions are constructed and compared to determine accuracy, ease of use with respect to purpose, cost, and development time. The final linear stage calibration products will be used to determine velocity jitter for Aerotech linear stages, which in turn may be applied to increase accuracy of spatial/spectral interferometric observations and simulate expected errors in the respective system.

Acknowledgements

My first semester back, after a long hiatus, I didn't know what to expect. Being twice the age of the usual first-year student, and not having a group of friends from high school, I was unsure if returning to university was going to be successful or end up another failed dream. That all changed when I made my first friend, or rather an old acquaintance.

While I was answering fellow students questions, on an online forum for linear algebra, Dr. Hadi Kharaghani had recognized me from almost 20 years earlier. He asked me, "Didn't you have a brother who worked for Radio Shack?" I replied, "No, that was me." Now every time Hadi introduces me to anyone, he starts with "He has a brother you know."

Working with Dr. Kharaghani during my first summer as an undergraduate set the tone of my academic experience. Under the supervision of his graduate student, Darcy Best, I was introduced to contest programming and Dr. Howard Cheng. Programming led to working with Dr. David Euston in the Neuroscience department. This chain of events has kept me employed throughout my degree, working for Dr. David Naylor, Dr. Adriana Predoi-Cross, and finally my supervisor, Dr. Locke Spencer.

I would like to thank Dr. Hadi Kharaghani, Dr. David Naylor and Dr. Locke Spencer, whom without, my undergraduate and graduate experience would not have been as enjoyable or successful. The amount of support and engagement from all the faculty I've had the opportunity to work with is appreciated; thank you to all.

I would like to thank Aaron Janz for his help with with the Aerotech PID controller, and Adam Christiansen and Jacob Groeneveld for introducing me to FPGA programming. Also, thank you to Roddy MacCrimmon and Adam Sundburg for enhancing my control programs

with graphical user interfaces and live plotting. A special thanks to Jeremy Scott, whom I've bounced ideas off of and has double checked a lot of my work.

I also need to acknowledge Rebecca Sirota and Greg Tompkins for their help with electronics, double checking my printed circuit boards and offering suggestions to improve the circuits I developed.

My acknowledgements would not be complete without recognizing the patience and grace my wife has displayed over the years required to accomplish my bachelors and masters degrees. I already see my example of hard work and disapline has transfered to my children, as they pursue their goals.

There are many more friends, peers, and colleagues, such that I cannot mention them all here. I want to thank all of them for helping me with my homework and research, and including me in gatherings and playing cards during lunch. I know I will see their names on some amazing papers, as they are the best and the brightest.

Thank you all.

Contents

List of Tables

List of Figures

List of Abbreviations

ADC analog-to-digital converter

ASIC Application Specific Integrated Circuit

ARTY Artix-7 FPGA Development Board

ASCII American Standard Code for Information Interchange

CDA central difference approximation

CLK clock

CPU counts per unit

DAC digital to analog converter

DAQ data acquisition system

DRT data request trigger

ENOB effective number of bits

EQD emulated quadrature divisor

FIFO first in first out

FIR far-infrared

FPGA field programmable gate array

FTDI Future Technologies Devices International

FTS Fourier Transform Spectroscopy

GPIO general purpose input/output

GUI graphical user interface

HeNe helium neon laser

I^2C inter-integrated circuit

IC integrated circuit

IP internet protocol

IPv4 Internet Protocol version 4

kSPS kilosamples per second

LCH latch

LED light emitting diode

LSB least significant bit

MAC media access control

MOSFET metal-oxide-semiconductor field-effect transistor

MSB most significant bit

MSPS mega-samples per second

MUX multiplexer

NOP null operation instruction

PCB printed circuit board

PID proportional, integral, and derivative controller

PSO position synchronized output

QAD quarter amplitude dampening

OPD optical path difference

OPL optical path length

OSC oscillator

RGB red, green, and blue

RLE10 Renishaw Laser Encoder

RMS root mean square

RPI20 Renishaw Parallel Interface

SAR successive approximation register

SNR signal to noise ratio

TEENSY Teensyduino 3.2 micro-controller

TEENSY2 Teensyduino 2.0++ micro-controller

TI time interval

UDP User Datagram Protocol

USB universal serial bus

ZPD zero path difference

Notes on Format

The following style conventions were selected:

Reference Parenthetic citation was utilized for references to external works by other authors (e.g., [1-3,7] refers to citations 1,2,3 and 7). Internal references to different chapters/appendix and sections are denoted by *chapter/appendix.section.subsection*. Figure, table and equation numbers are denoted by *chapter/appendix.value*, where value is the incremental numerical value of the figure, table or equation.

Equations Units are provided for every numbered equation, presented in square brackets (e.g., $[\mu m \cdot s^{-1}]$ indicates micrometres per second). If the units are not provided, the equation is unitless. Units are provided in the standard International System of Units (SI), unless otherwise noted.

Text The `typewriter` font was used to denote specific Arduino and Python commands. Verilog and Python code is presented in a mono spaced font, where line numbers are grey, keywords are presented in blue, commenting is a dark green and strings are shown in mauve.

Chapter 1

Introduction

"The fact that we live at the bottom of a deep gravity well, on the surface of a gas covered planet going around a nuclear fireball 90 million miles away and think this to be normal is obviously some indication of how skewed our perspective tends to be."

–The Salmon of Doubt

– Douglas Adams

Our view of the universe has advanced over the centuries, from being at the centre of creation to existing at nowhere special. As little as thirty years ago we believed we understood what 96% of the universe consisted of. As our instrumentation and technology has improved, we have since discovered that only 4% of the universe consists of regular matter, and the rest being dark matter and dark energy, which are indirectly detectable by their gravitational effects [1]. The light-gathering and resolving power of a telescope is dependent upon the wavelength of light being observed and the diameter of the primary mirror or lens. The angular resolution of the instrument, with respect to diffraction through a circular aperture, can be described by the Rayleigh criterion [2]

$$\theta = 1.22\frac{\lambda}{D}\ [rad], \tag{1.1}$$

where D is the diameter of the primary optic, λ is the wavelength of light of interest and θ is the resultant minimum angle of separation in radians between two objects that can be discerned.

Galileo's telescope was at the peak of optics at the beginning of the 17^{th} century. Even though, details on Mars, Jupiter and Saturn could only barely be discerned. Considering that Galileo's telescope had a 37 mm objective lens [3], and observed in the visible spectrum of light (between 400-650 nm wavelengths), he was able to discern objects separated by approximately 4 arcseconds. The largest single-aperture far-infrared (FIR) telescope to date, Herschel, had a primary mirror of 3.5 m diameter. This is a considerable increase in aperture but, due to the much larger wavelengths, between 55 μm and 672 μm [4], the angular resolution was still similar to that of Galileo's original telescope, centuries prior.

Making the diameter of the primary mirror larger for an FIR instrument is not a practical solution for a few reasons. Primarily, the Earth's atmosphere is mostly opaque to FIR radiation [5], which requires FIR instruments to be placed above the atmosphere. Vehicles such as the Ariane 5G rocket have a limited payload diameter of 5 m [6] and, depending on the final destination, the weight of the payload is a limiting factor. Complex segmented

folding mechanisms, like the design of the James Webb Space Telescope [7], allow for a larger diameter primary mirror, but the weight and engineering constraints limit how far this design may be expanded. Using equation 1.1, with $\lambda = 50\,\mu$m and $\theta = 0.5$ arcseconds, astronomers wishing for sub-arcsecond resolutions, at FIR wavelengths, would require a primary mirror with a diameter greater than 250 m.

In 1887, Michelson and Morley used an instrument to verify the medium through which electromagnetic radiation propagated [8]. The null result illuminated the nature of light, demonstrating that electromagnetic radiation does not need a medium in which to travel. The device, named the Michelson interferometer, had the ability to measure very small displacements of a single mirror and consequently the wavelength of light. The utility of interference methods for measuring the wavelength and intensity of light, called spectral interferometry, had been understood by Michelson [9], but required the advancement of optics, electronics, computing, and the development of a Fast Fourier Transform algorithm (1965) to make the realization of Fourier Transform Spectroscopy (FTS) [10] possible.

Michelson had realized the application of his instrument for spatial interferometry for astronomy, demonstrating the ability to measure the angular size of Jupiter's moons [11]. As early as 1890, Michelson demonstrated that interferometry could increase the resolution of telescope observations, allowing one the ability to discern double-stars too close to be resolved by the "most powerful telescopes" of the time [12]. Michelson and Pease demonstrated the ability to measure the diameter of α Orionis (also known as Betelgeuse) in 1921 [13], using a 20-foot interferometer on the Palomar telescope. Pease continued to advance stellar interferometry, after Michelson's passing in 1931, but had many difficulties due to the technical limitations of the time. Ryle and Vonberg demonstrated the application of spatial interferometry in 1946 [14], after constructing the first multi-element radio telescope. The use for optical wavelength astronomy wasn't fully realized until Antoine Labeyrie [15] demonstrated the ability to combine the light from multiple telescopes.

During the 1970's Labeyrie used interferometry to increase the angular resolution of

optical wavelength telescopes [15]. The interference patterns of light from two instruments separated by a baseline(B), can be used to synthesize the resolution of an equivalent diameter objective [16], such that

$$\theta = \frac{\lambda}{B} \tag{1.2}$$

where θ is the angular resolution for a monochromatic source of wavelength (λ).

As a pre-cursor to a space-based mission, proof of concept instrument must be established and achieve the required technology readiness level (TRL) [17]. Generally, satellites of this nature are located in an orbit beyond the moon (commonly know as "L2" or Lagrangian point 2 [18]). This stationary orbit provides more stable thermal equilibrium, as the satellite would not pass in and out of the Earth's shadow causing warming and cooling effects. Due to the distance and expense to place objects in this orbit, it is almost impossible to retrieve or repair them. Thus, test bed instruments must ensure that technology is proven and dependable, before these concepts can proceed towards inclusion in a space mission.

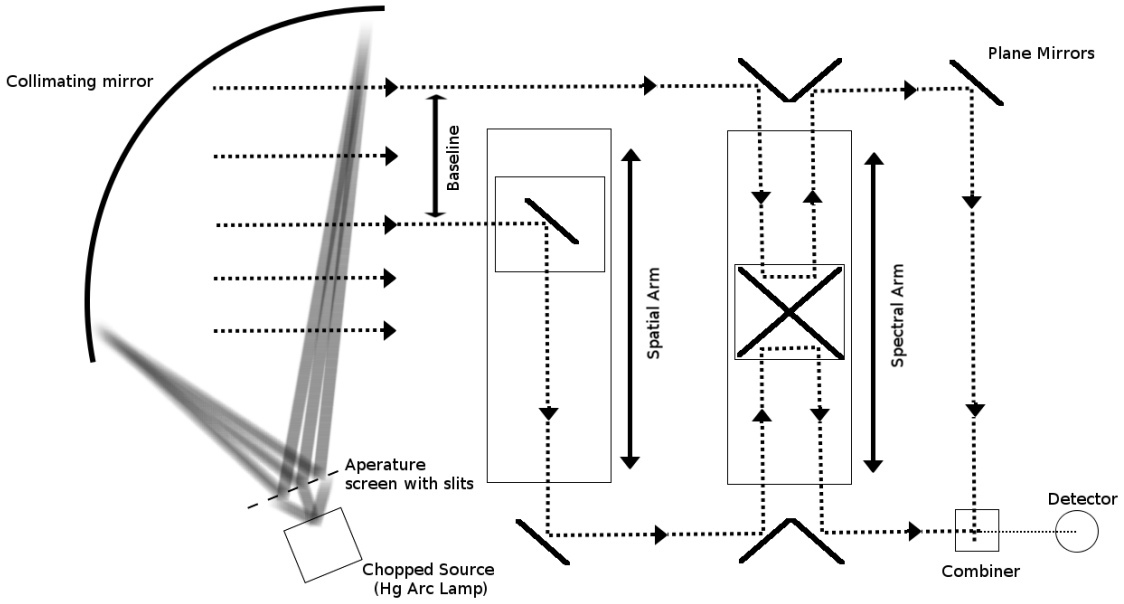


Figure 1.1: The proposed test bed instrument will scan spectra for a variety of baseline locations on the spatial arm. The double roof-top mirror configuration, located on the spectral arm, creates a four-times linear displacement optical path length change. Each linear translation stage will have an absolute displacement of 0.45 m.

Figure 1.1 depicts the proposed instrument which is already under construction, similar to the apparatus used by William F. Grainger, et al., (2012) [19, 20]. The spectral arm of the device employs the efficient design of a Mach-Zehnder interferometer [21, 22], where the spatial displacement of the stage results in four times the optical path length change due to the double roof-top mirror configuration. The baseline is determined by the perpendicular displacement with respect to the input aperture. The spectral and spatial linear translation stages are an integral part of the test bed instrument and their motion must be characterized to more fully understand the data observed.

1.1 Summary

This thesis objective was to develop some of the tools to profile and test the spectral linear translation stage being utilized in the test bed instrument. Before a metrology device can be implemented, the function and precision must be understood. Chapter 2 delves into the theory behind the operation of an interferometer and methods used to analyze observed data. This includes methods of numerical derivation, the operation of coarse counting temporal metrology devices, the basic understanding of proportional, integral, and derivative controllers (PIDs), and an understanding of potential negative effects of using a successive approximation register (SAR) analog-to-digital converter (ADC).

Chapter 3 studies one of the simplest devices available for data acquisition to the laboratory, a micro-controller. Introducing the Teensyduino 3.2 micro-controller (TEENSY), the strengths and weaknesses are examined for measuring time interval (TI), and serial communication. The onboard ADC was tested and profiled by methods described in Chapter 2. The end result decided that increased bandwidth (i.e. data sampling rate) and digital precision was required to achieve the end goals.

Chapter 4 develops the first attempt to increase performance of the TEENSY by adding discrete electronics in unison with multiple micro-controllers. Both asynchronous and synchronous circuit designs were explored, finally constructing a 66 MHz synchronous coarse

counting clock with two discrete 16-bit ADCs for use with photo-detectors, referred to as TEENSY data acquisition system (DAQ).

The ability to measure the position of the linear translation stage was performed by three metrics, the internal encoder of the stage itself, a Michelson interferometer and a commercial laser interferometer. Before the commercial laser interferometer, designed by Renishaw, could be used, an interface between Renishaw Parallel Interface (RPI20) and the recording computer was updated for increase sampling bandwidth. Chapter 5 outlines the construction of the interface, which took advantage of the efficient Artix-7 FPGA Development Board (ARTY). The resultant printed circuit board (PCB), which married the RPI20 card and ARTY, allowed for a $100\times$ improvement in the sampling rate. Serial communication constraints were eliminated by adapting the User Datagram Protocol (UDP) network transfer for data transmission.

The lessons learned while developing the Renishaw interface were applied to creating a version of the previous TEENSY DAQ which replaced most of the electronics and all of the micro-controllers with an ARTY. Chapter 6 discuss the process of designing the field programmable gate array (FPGA) version of the DAQ, which would see the upgrade of the coarse counting clock to 200 MHz and the 16-bit ADC capable of 2 mega-samples per second (MSPS). While the TEENSY DAQ was sufficient for the initial task at hand, these upgrades enhanced the resolution of the instrument by an order of magnitude, and increased sampling bandwidth by a factor of twenty.

Chapter 7 applied the developed metrology instruments to a real-world physical system. A linear translation stage, model ALS20045 made by Aerotech, was examined for accuracy of commanded displacement and velocity, by using the three previous metrology systems. The position synchronized output (PSO), generated by the Aerotech linear translation stage, emits a pulse dependent on the distance travelled that was studied for precision. A final examination of the noise spectra was performed for later use in simulating spatial/spectral results.

Chapter 8 provides a summary of the performance of the designed metrology devices and the linear translation stage, and a description of continuing work. An overview of the electronic design, Python control interfaces and controller programming for each of the designed instruments is provided in appendices which follow.

Chapter 2

Considerations and Methods for Design and Analysis

“ Where two portions of the same light arrive in the eye by different routes, either exactly or very nearly in the same direction, the appearance or disappearance of various colours is determined by the greater or less difference in the lengths of the paths.”

–Lecture XIV. ‘Of Physical Optics’.

In A Syllabus of a Course of Lectures on Natural and Experimental Philosophy (1802)

– Thomas Young

Thomas Young's double slit experiment (1801) demonstrated that light travelled as a wave [2]. This set the stage for the development of instrumentation which used the constructive and destructive interference of light for scientific research.

Keeping in mind that an end goal of this thesis project was to design temporal and spatial metrology systems, specifications were required to ensure functional compliance. By understanding the operation of various interferometer designs, presented in Section 2.1, an upper bound on the bandwidth required for data collection was obtained.

As stated in the last chapter, the linear translation stage is a key component for the proposed astronomical test bed instrument. Section 2.2 takes a general approach to describe the proportional, integral, and derivative controller (PID), aspects of the position encoding system, and position synchronized output (PSO). These systems control the internal velocity and position metrology and need to be shown to agree within error with any of the final data acquisition system (DAQ) devices.

Since velocity is the derivative of position and time, Sections 2.3 and 2.4 provide the design considerations for the temporal and spatial metrology systems. The methods of analyzing the motion of the linear stage are introduced in Sections A.1, including algorithms for numerical differentiation.

Finally, an overview of how a successive approximation register (SAR) analog-to-digital converter (ADC) functions is provided in Section 2.5. The efficiency of the various tested ADCs was measured using the signal to noise ratio (SNR) described in Section 2.5.2 and the corresponding effective number of bits (ENOB) in Section 2.5.3.

2.1 Interferometer Designs

The optical path length (OPL) is defined as the perceived geometric distance light in a vacuum has travelled, accounting for changes in the index of refraction. When the source light is split, the difference between the two OPLs is referred to as optical path difference (OPD). Figure 2.1 depicts the common arrangement for a Michelson interferometer.

Although, the figure shows a linear stage being used to alter the path length the light must travel, in Michelson and Morley's original experiment both mirrors were fixed and it was assumed that any OPD was introduced by the motion of the Earth through the aether. It was also possible to evoke a change in OPL by introducing a temperature or pressure change in a medium that the source light passed through, thus an interferometer has the ability to measure many physical properties besides wavelength and position. Most recently, the detection of gravitational waves has demonstrated the sensitivity that can be achieved by interferometric methods [23]. If air turbulence, vibration and other sources of OPD are accounted for, and the linear stage is considered the only source of motion, then the displacement of the plane mirror will have produced twice the change in OPL.

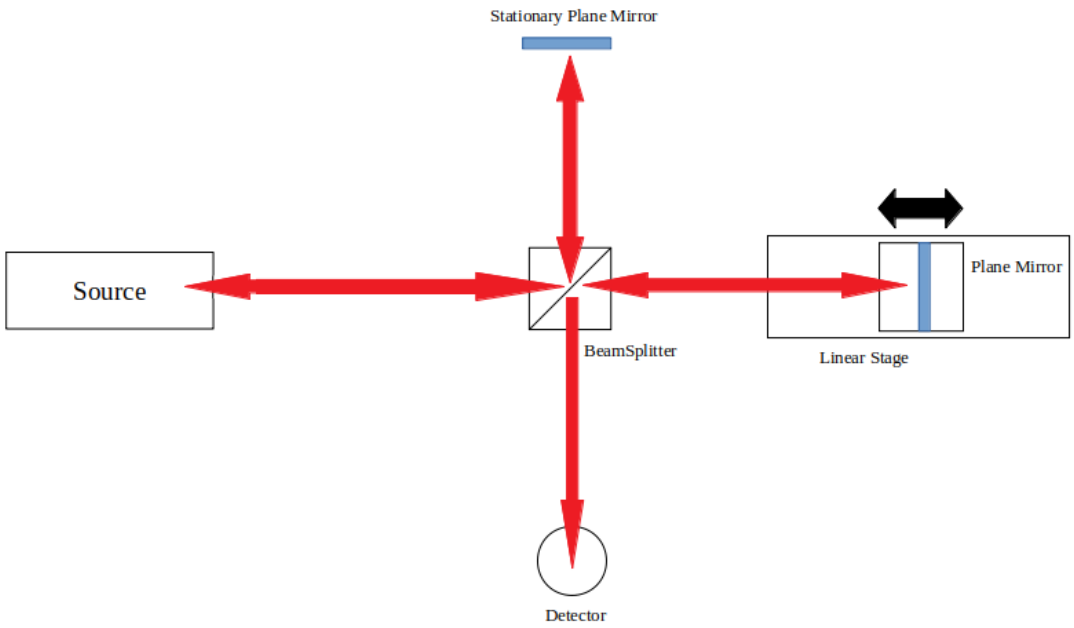


Figure 2.1: The configuration of a Michelson interferometer. Coherent light emitted from the source and passed through a beam splitter (ideally 50%-50%), displays constructive and destructive interference upon recombination. The interference displayed is reliant on the position of the plane mirrors which directly determines the path length the light must travel through each arm.

All interferometers have two input and output ports [24]. One of the drawbacks of the Michelson interferometer, depicted in Figure 2.1, is that the design causes one of the output ports to fall directly back on one of the input ports. The loss of 50 % of the input

flux, back upon the input port, limits the strength of the signal falling upon the detector. Signal strength is a factor for consideration when designing the ADC circuit, determining the amount of amplification and offset that may be required. If amplification is required, the distortion of the signal should also be considered, especially in spectroscopy where the line shape may be affected.

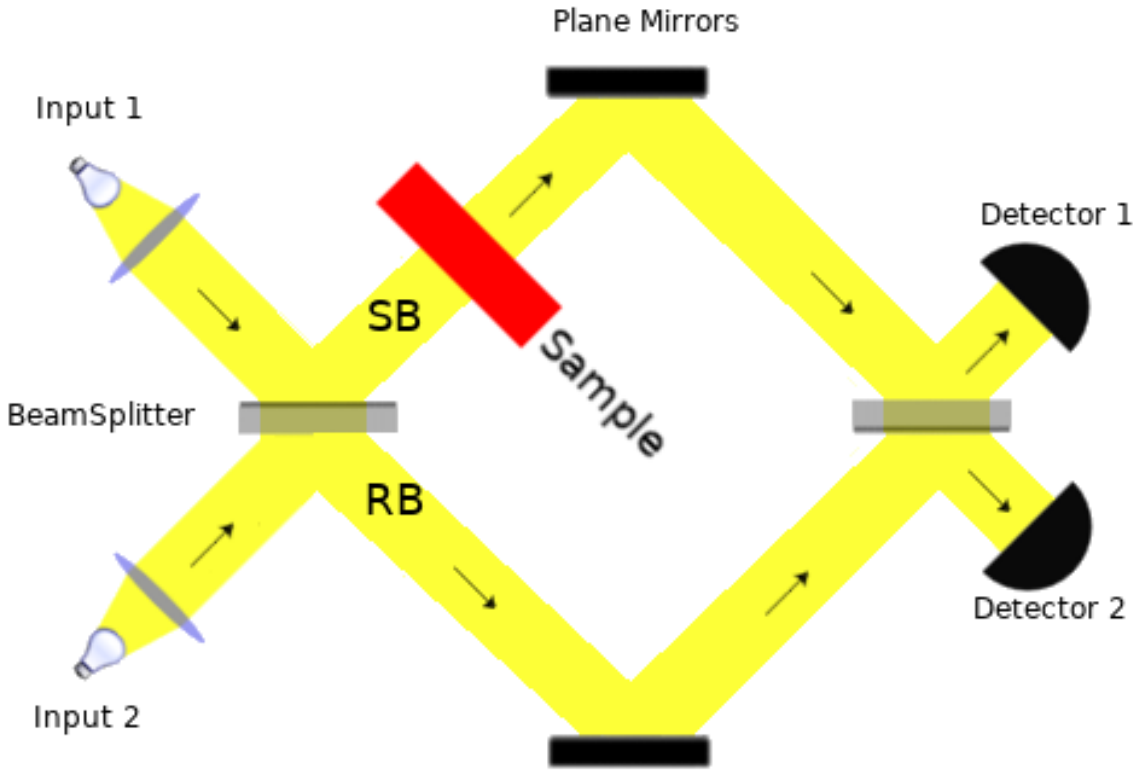


Figure 2.2: The path of light through a Mach Zehnder interferometer. The sample beam (SB) experiences an optical path length (OPL) change due to the differences between material index of refraction compared to beam path. The reference beam (RB) and SB have identical path lengths (assuming the thickness of the two beam splitters is the same) when no sample is present. It can be shown that detector 1 and 2 are π radians out of phase. Image modified from original concept from Daniel Mader. [25]

The Mach Zehnder interferometer, pictured in Figure 2.2, allowed for greater precision by orienting the optical components to assure that path length was kept equal in both arms of the device. Since pressure and heat changes have a noticeable change in the index of

refraction, the Mach Zehnder is commonly used visualizing flow in wind tunnels and heat transfer [26, 27].

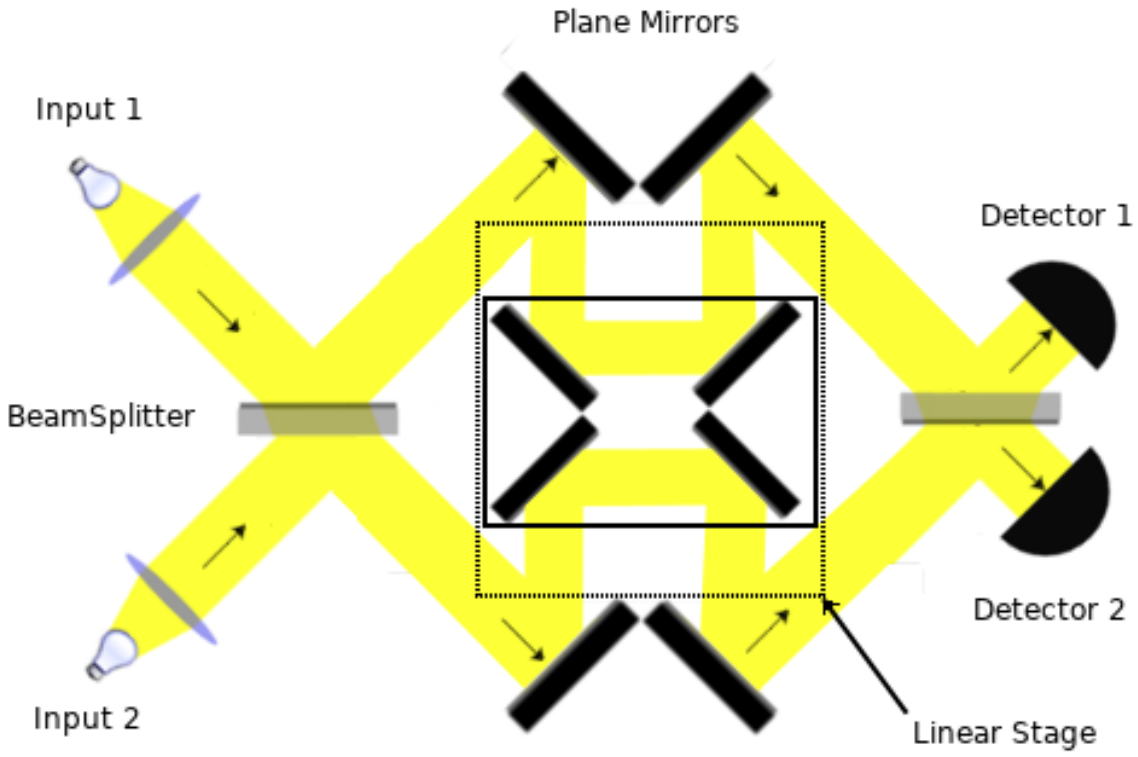


Figure 2.3: Mach Zehnder configuration for the Spectral Stage. Instead of a sample introducing a change of path length due to index of refraction, the linear stage introduces a physical displacement which in turn changes to the optical path length by a factor of four times. Original image concept by Daniel Mader [25].

The intended test bed instrument utilizes two linear stages in a Mach Zehnder configuration, described in Figure 1.1 and Figure 2.3. The spectral arm used the rapid-scan method of recording, where samples are taken at even spatial or temporal intervals while the stage transverses a distance at a constant velocity. The spatial arm steps to different predetermined baseline distances and holds for the duration of the spectral arm scan. The end result is a two-dimensional intensity map of the interference, providing a spatial modulation in one dimension. It should be noted that the roof-top mirror arrangement on the spectral arm produces an OPD of four times the physical displacement of the linear stage.

Due to the ease of alignment, the Michelson configuration was used to verify the mo-

tion of an Aerotech linear stage, in Chapter 7. To produce a dark fringe on the detector, destructive interference requires the optical path length difference to be an integer plus one half the wavelength of the source light. The time-domain frequency of the fringes falling on the detector can be calculated by

$$f_{signal} = \frac{2\mathrm{v}}{\lambda}[Hz] \tag{2.1}$$

where 2v is the optical path difference velocity due to the linear stage is motion and λ is the wavelength of light. The frequency of the interference pattern being received by the detector is important as it determines the minimum sampling frequency. Thus, the upper bound on the speed which the stage travels is dependent upon the rise/fall time of the photo-diode and the maximum bandwidth of the ADC.

2.2 Linear Translation Stage Considerations

The linear translation stages require testing to determine the noise introduced by the PID controller, the variation of the velocity from the commanded speed, and accuracy of the PSO. Effects from the PID, and deviations from the commanded velocity lead to uncertainty in knowing the position.

2.2.1 PID Controller

The standard equation for describing a PID controller is:

$$U(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t) + U(0) \tag{2.2}$$

where the output $U(t)$ is the time-dependent setting of some independent control based on the difference signal, $e(t)$, from the set-point and current output. Respectively, K_p, K_i, K_d are the constants for proportional, integral and derivative control parameters. A classic example of a PID controller is the cruise control in a car. The independent variable being

controlled is the amount of fuel being provided to the engine of the car. The error being measured is the difference between the set and the actual speed the vehicle is travelling (the dependent variable). Equation 2.2 transforms the current error in speed into a signal which controls the fuel being delivered. The response is not precise, but rather a process of overshooting and falling short until the error diminishes.

The proportional response, controlled by K_p, adjusts the process variable by a commensurate value to the perturbation. "Sensitivity" is defined as the amount of output change depending on the input to the process. Since the input to the process is directly affected by the proportional response, the proportional response is commonly referred to as sensitivity.

The setting of the three constants is a search in three-dimensional vector space, where some solutions provide better response to different frequencies of perturbations and others may cause wildly oscillating signal and loss of stability. In most cases, a full mathematical description of the system is complicated, and in some cases impossible, to account for all the variables that may affect it.

The Ziegler-Nichols [28] and Tyreus-Luyben [29] tuning rules were established using quarter amplitude dampening (QAD) as an objective. QAD attempts to reduce the time to correct errors between the set-point and output signal, but it requires that it overshoots the set-point creating oscillations. The amplitude of these oscillations decreases each cycle by a ratio of 4:1, hence the name, as in Figure 2.4.

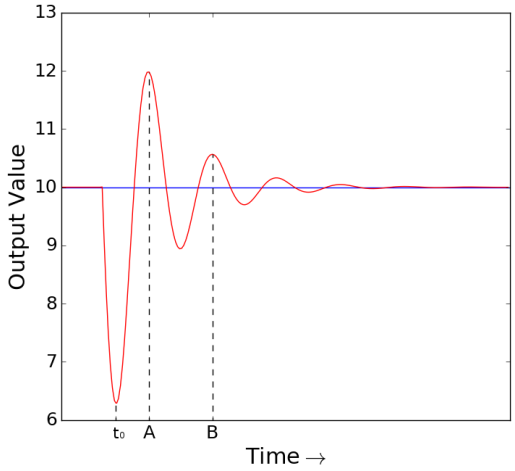


Figure 2.4: Quarter Amplitude Dampening is employed by some PID controllers to mitigate the effects of a perturbation. The first lowering peak, at t_o, caused by a disturbance from the set point (blue line), results in some overshoot, at times A and B. The amplitude of B is one-quarter the amplitude of A.

There are no restrictions from using a different tuning objective besides QAD, illustrated in Figure 2.4. Depending on the system in question, tuning objectives may require less overshoot or desire to decrease mechanical movement to lower maintenance expenses.

The actual PID controller for the Aerotech linear translation stage, that was utilized in this research, is investigated in Chapter 7, along with a discussion about attempts to tune the controller constants.

2.2.2 Linear Translation Stage Motor Encoder

This section applies to the Aerotech linear translation stage and the accompanying Ensemble controller [30] that was used to develop and test the instrumentation developed in this thesis. Different linear translation stages may incorporate alternate sub-systems used to monitor and track position, which may or may not be similar.

The motor encoder measures displacement in steps of the counts per unit (CPU), which is set to 16 million counts per millimetre. The feedback from the encoder has an upper limit of 25.0 MHz, limiting the velocity which the stage can reliably measure position. The maximum velocity can be determined by dividing the encoder limit by the CPU value resulting

in $1.56\,mm \cdot s^{-1}$; higher velocities result in inaccurate PSO pulses and irregular stage behaviour. The solution, to allow greater velocities, is the emulated quadrature divisor (EQD) parameter, which effectively is a frequency divider, consequently decreasing resolution of the position measurement. The appropriate value for the EQD can be determined by,

$$EQD = \left\lceil \frac{velocity \times CPU}{25.0 \times 10^6\, MHzCounts \cdot s^{-1}} \right\rceil [unitless] \tag{2.3}$$

where EQD is an integer value determined by the ceiling function, and is required for determining the proper settings of other linear stage functions [30].

2.2.3 Position Synchronized Output

There is a maximum output frequency, for the model of Aerotech linear translation stages used for this research, limiting the PSO to 12.5 MHz maximum [30]. Due to the encoder output having twice the maximum frequency output, the linear translation stage will accurately continue to move at higher velocities even though the PSO will cease to function correctly. If the PSO output frequency, for the desired distance and velocity, is greater than 12.5 MHz, the EQD can be recalculated to an appropriate value by altering Eq. 2.3 to

$$EQD_{PSO} = \left\lceil \frac{velocity \times CPU}{12.5 \times 10^6\, Counts \cdot s^{-1}} \right\rceil [unitless] \tag{2.4}$$

which cannot violate the constraint of Equation 2.3 as it will always result in a larger value. The PSO provides a pulse of the specified period when the linear stage has travelled a desired distance since the last PSO pulse, provided in counts and monitored by the Ensemble controller, such that

$$PSO_{counts} = \left\lfloor \frac{PSO_{mm} \times CPU}{EQD} \right\rfloor [counts] \tag{2.5}$$

Since the PSO_{counts} is an integer number, the count value must be rounded. The effect of integer counts was that only PSO distances that are multiples of CPU (63 pm) times EQD are achievable. The actual distance travelled between PSO pulses had to be verified and is

investigated further in Chapter 7. The accurate timing of the interval between PSO pulses was required to determine the velocity profile of the linear stage.

2.3 Temporal Metrology

Experiments often require data acquisition with an accurate time stamp to describe the moment that an event occurs. Considering the fringes of light from an interferometer fall on the detector at a rate that is dependent on the wavelength of the light source and the rate OPD changes, then the frequency of the incoming signal can be calculated by

$$f_{sig} = \frac{n \times \mathrm{v}}{\lambda}[Hz], \tag{2.6}$$

where n is the mechanical displacement to OPD conversion factor, the velocity (v) of the linear stage is assumed to be the only source of OPD, with the wavelength (λ) and flux of the light source(s) constant. Any mechanical displacement, for a Michelson interferometer, alters the OPD by a factor two times the displacement, therefore $n = 2$. The conversion factor is dependent on the configuration of the interferometer and should be adjusted accordingly.

The Nyquist sampling criterion [31] states that a minimum of twice the highest frequency of interest is required for DC band-limited signal reconstruction to be possible. If the PSO is the source for requesting the moment when a sample is observed, then an accurate time relative to the start of the experiment must be correlated to each pulse. It is desirable to have more than two points per period for fitting routines to estimate the data, therefore

$$f_{samp} = \frac{kn \times \mathrm{v}}{\lambda}[Hz], \tag{2.7}$$

where f_{samp} is the sampling rate and k is the oversampling factor per period such that $k > 1$. Then, the desired PSO distance can be calculated by

$$PSO_{dist} = \frac{\mathrm{v}}{f_{samp}} = \frac{\lambda}{kn}[m], \tag{2.8}$$

The result of Eq. 2.8 must be inserted into Eq. 2.5 to determine the effect of rounding and then can be converted back to the actual PSO distance that can be commanded. Finally, the time between PSO pulses can be estimated by dividing the PSO distance by the velocity of the linear stage. A small relative error was desired for measuring the TI between PSO pulses, putting constraints on the oscillator used for comparison.

To put this in perspective, consider the linear stage moving at $500\,\mu ms^{-1}$ and it is desired to sample a HeNe ten times per fringe using a Michelson interferometer (n=2). Assume the EQD is 1, as the velocity is below $781\,\mu ms^{-1}$. The PSO distance, calculated using Eq. 2.8, is 31.64 nm. Rounding the result from Eq. 2.5, then converting back to a distance, determined an actual PSO distance of 31.626 nm. From this, the expected frequency of PSO pulses would be 15810 Hz with a period of $63.2\,\mu s^{-1}$. For 0.1% error, the clock measuring the TI would require a resolution of 63 ns, assuming the error in timing is ± 1 oscillator pulse.

2.3.1 Coarse Counting Clock

A common method of measuring the TI between events is to count the uniform periods of a reference oscillator. This method results in the time resolution being less precise than the length of one oscillation period, and therefore is called a *coarse* counting method [32]. For greater accuracy, it is necessary to decrease the size of the coarse time periods, demanding a faster reference oscillator or some quadrature signal. It is also required that the TI being recorded is not coincident with the oscillator pulse to the counter (asynchronous), to prevent the counter incrementing during requested data acquisition.

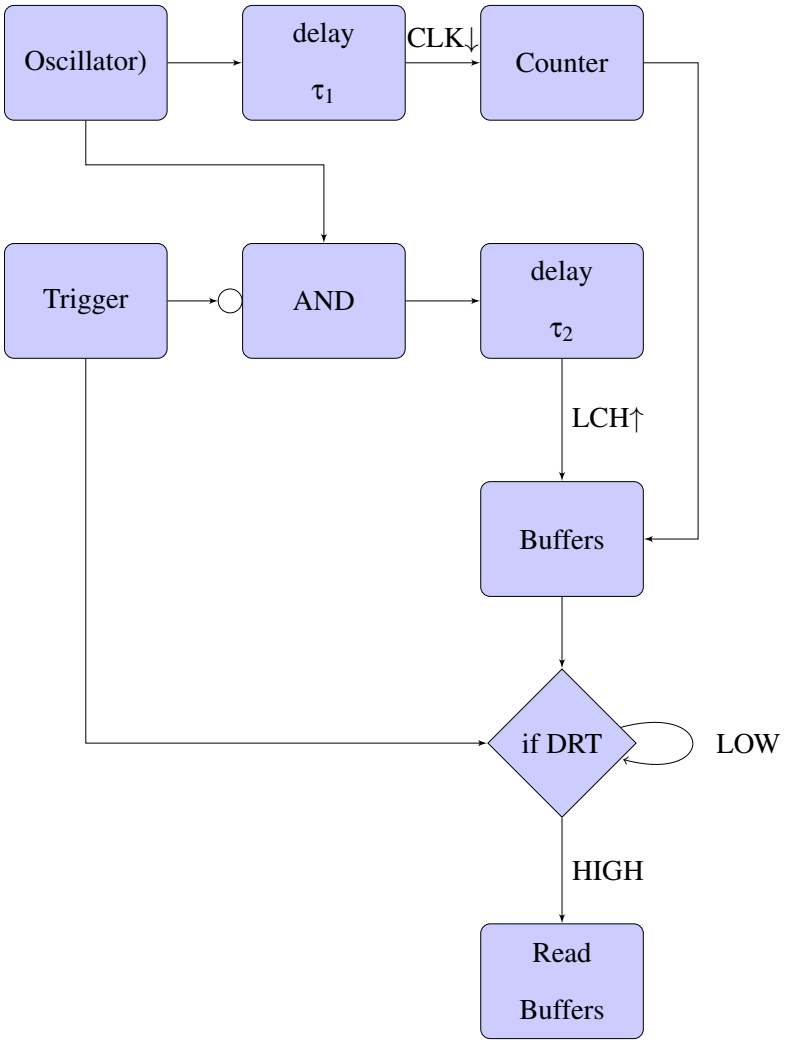


Figure 2.5: The design used to build the coarse counting clocks for collecting temporal metrology. Count values are incremented on the falling edge of the oscillator and stored in the buffers on the rising edge. When the rising edge of the trigger occurs, the buffers are held from collecting new values and can be read by a micro-controller.

The circuit, illustrated in Figure 2.5, consists of an oscillator, a binary counter, a set of latches to hold the count value to be read by a micro-controller and some logic to trigger the latches with an appropriate delay. A micro-controller by itself, an asynchronous ripple counter, and an asynchronous binary counter were all tested for accuracy and efficiency. The results for the micro-controller on its own are discussed in Chapter 3, and the asynchronous ripple and synchronous binary counters in Chapter 4. The design was also implemented on the ARTY FPGA development board and these results are reported in

Chapter 6.

2.4 Spatial Metrology

Since the PSO has not been verified for accuracy, it could not be used as the only measure of the position of the linear stage. Thus, a commercial interferometer, which has been well tested and trusted, was borrowed from Dr. David Naylor's group in our department. The Renishaw interferometer has a resolution of 38.6 pm in a vacuum, but when subjected to air currents, and nominal temperature, humidity, and pressure changes in our laboratory, it has been shown to provide reliable measurements in the tens of nanometres. Chapter 5 discussed the design and construction of the computer interface required to capture data from the Renishaw Parallel Interface (RPI20). To allow for a meaningful comparison of the Renishaw and Michelson interferometer, the plane mirror must be perpendicular to the incoming light source, making the light path parallel to the motion of the linear stage. Cosine error introduced due to the plane mirror being miss-aligned is discussed in Section A.2.1.

As a secondary verification of the linear translation stage positioning encoder, a Michelson interferometer was also used to inspect the motion. Primarily, used to count fringes during the constant velocity motion of the stage, different methods to retrieve displacement measurements are discussed in Chapter 7. In order to properly inspect the spectral line of the HeNe laser, the position of the stage that provides an absolute zero path difference (ZPD) would be required, but a suitable white light source was not available at the time.

2.5 Theoretical ADC Operation

The perfect analog-to-digital converter (ADC) would have a linear response across the entire input range where the output would be an integer value such that,

$$\chi_{ADC} = \left\lfloor \frac{v_{in}}{v_{ref}} \times \left(2^N - 1\right) \right\rceil \tag{2.9}$$

and N is the number of bits, or bit-depth of the ADC. To model the parameters of the system, we will assume that the v_{in} will be positive values only, but the theory applies to ADCs which can convert voltages which span the origin.

2.5.1 Successive Approximation-Register Analog to Digital Converter Operation

There are many different methods for the internal operation of an ADC. All of the ADCs utilized were of the variety known as SAR. The SAR ADC required few components, a comparator, digital to analog converter (DAC), and a register. The operation consists of latching a sample, as changes to the input during conversion would result in an erroneous result, and comparing the sample to the output of the DAC. The voltage output from the DAC is set by the results of the previous comparisons, starting with the highest bit first. The output is set with the value used to set the DAC after all bits have been evaluated. The benefit of this type of converter is generally low power consumption. The drawback is the time to convert a sample is dependent on the number of bits in the ADC, as the comparator must iterate through each bit for every sample.

The SAR method was simple to model in Python, using a loop that counts backward from the highest bit to zero. It should be noted that with a 16-bit ADC, the bit values are zero indexed, thus are numbered from 15 to 0. The operation (1«n) set the n^{th} bit to 1, as shown on lines 4 and 7 of Figure 2.6.

```
def SAR_ADC_OUT(x, bitdepth, Vref): #where x is the voltage being evaluated
   result = 0 #output register
   for n in range (bitdepth-1,-1): #will stop 1 before last number in range
      DAC = result + (1<<n) #DAC is equal existing value plus a value of
          current bit
      Vout = DAC / 2.0**bitdepth * Vref
      if Vout < x :
         result = result + (1<<n)
   return result
```

Figure 2.6: Emulating the output of an SAR ADC in Python. The function follows the loop for each bit, comparing the digital to analog output to the input being evaluated. The ideal SAR ADC output is returned.

The code snippet shown in Figure 2.6 was used to convert the function for measured waveform into the ideal ADC response.

2.5.2 Signal to Noise Ratio for Ideal Analog to Digital Converter

Due to the effect of rounding in Eq. 2.9, the error in any value output from the ideal ADC can be $\pm$ half the step size(δ), typically expressed as half least significant bit (LSB). Step size is defined as the reference voltage divided by $2^N - 1$, where N is the bit-depth of the ADC. Since the ideal ADC is assumed to be linear, it makes sense to model the error in any value as a ramp from $\frac{-\delta}{2} \leq e(v_{in}) \leq \frac{\delta}{2}$ for $\frac{-\delta}{2} \leq v_{in} \leq \frac{\delta}{2}$ as a starting point.

The SNR provides a measurement of the quality of the signal being received. To measure SNR we need the root mean square (RMS) equivalent value of the noise and the signal.

$$e_{rms} = \sqrt{\frac{1}{\delta}\int_{\frac{-\delta}{2}}^{\frac{\delta}{2}} v_{in}^2 dv} = \sqrt{\frac{1}{\delta}\left(\frac{v_{in}^3}{3}\Big|_{\frac{-\delta}{2}}^{\frac{\delta}{2}}\right)} = \sqrt{\frac{\delta^2}{12}} = \frac{\delta}{2\sqrt{3}} \tag{2.10}$$

Assuming a sinusoidal signal being input to the ADC which provides a full-scale range of voltages, then $V_{pp} = (2^N - 1) \approx 2^N\delta$.

$$sig_{rms} = \sqrt{\frac{1}{T}\int_0^T \left(\frac{2^N\delta}{2} sin(\omega t)\right)^2 dt} = \sqrt{\frac{2^{2N}\delta^2}{4T}\cdot\frac{T}{2}} = \frac{2^N\delta}{2\sqrt{2}} \tag{2.11}$$

The SNR is usually converted to the power decibel (dB) scale, where the number of decibels is ten times the square of the ratio of the RMS signal and RMS noise.

$$\begin{aligned} SNR &= 20log_{10}\left(\frac{sig_{rms}}{e_{rms}}\right) \\ &= 20log_{10}\left(2^N\sqrt{\frac{3}{2}}\right) \\ &= 20log_{10}(2)N + 10log_{10}\left(\frac{3}{2}\right) \\ &= 6.02N + 1.76[dB] \end{aligned} \tag{2.12}$$

The result of Eq. 2.12 is the ideal ADC SNR, for a sinusoidal input. This implies that a perfect ADC with 16-bit resolution would have an $SNR \approx 98\,dB$, assuming the only noise is from quantization.

A saw tooth wave is used in future chapters to test linearity of the ADC, ensuring that the converter responds the same across it's dynamic range of input values. The change in wave form alters the ideal SNR equation 2.12 such that,

$$sig_{sawrms} = \sqrt{\frac{1}{T}\int_0^T \left(\frac{2^N\delta}{T}\right)^2 t^2 dt} = \sqrt{\left(\frac{2^{2N}\delta^2}{T^3}\right)\frac{T^3}{3}} = \frac{2^N\delta}{\sqrt{3}}. \tag{2.13}$$

$$\begin{aligned} SNR_{sawtooth} &= 20log_{10}\left(\frac{sig_{sawrms}}{e_{rms}}\right) \\ &= 20log_{10}\left(2^{N+1}\right) \\ &= 20\,(N+1)\,log_{10}(2) \\ &= 6.02N + 6.02[dB] \end{aligned} \tag{2.14}$$

Equation 2.14 demonstrates that a saw tooth waveform should provide a greater ideal SNR $\approx 102\,\text{dB}$, for a 16-bit ADC.

2.5.3 Effective Number of Bits

If the SNR was experimentally measured for a sawtooth input signal, and the result was substituted into Eq. 2.14, and solved for N, the outcome would be the effective number of bits (ENOB) for the physical ADC (use Equation 2.12 for a sinusoidal input). The ENOB provides a measure of the error in any measurement taken by an ADC. Thus, the effective number of bits for a known SNR is

$$ENOB = \frac{SNR - 6.02}{6.02} = \frac{SNR}{6.02} - 1[bits]. \tag{2.15}$$

For example, if a 16-bit ADC has an ENOB of 13, then the bottom 3-bits cannot be assumed to be accurate, and the measurement would be ± 7 arbitrary bins of the conversion values.

2.5.4 Increasing Effective Number of Bits through Averaging

With the desire to increase the resolution of the converter, oversampling and averaging provides an increase to SNR. This is possible only if the source of the noise is random and evenly distributed on either side of the mean, for example, white noise. The over sampled frequency, required to provide one additional effective bit, must be four times the Nyquist sampling frequency [33]. From this the equation to select an over sampled frequency (f_{os}) based on the number of desired additional bits (W) is

$$f_{os} = 4^W \cdot f_s \tag{2.16}$$

Using the signal of a Michelson interferometer as an example, the expected frequency of the fringes passing a detector was 1.58 kHz, when the linear stage was moving at $500\,\mu s^{-1}$ and the wavelength of the laser is $0.6328\,\mu m$ (calculated using equation 2.6). The Nyquist

frequency was then determined to be 3.16 kHz. Since least squared fitting routines, available in Python, benefit from having more points than the minimum two per period required by Nyquist rules, a sampling frequency of 15.8 kHz was finally chosen. A 16-bit ADC was utilized to record data, with only 14 ENOB. To effectively gain some of the ineffective bits back through oversampling and averaging, the sampling frequency would have had to be sixteen times faster, at a rate of 253 kHz. Averaging groups of 16 samples causes the noise to be decreased while not affecting the signal, thus increasing the SNR and ENOB. If the throughput of the ADC was only 100 kHz, it would not be possible to sample at an adequate rate to achieve this gain.

2.6 Conclusions

Understanding the operation and motion of the linear stage is critical to understanding the data collected by an interferometer. Developing the methods necessary to reduce the relative error in measurements allows the accurate description of the motion, providing the means to correct spectral/spatial data, that will be obtained in a future configuration of the system.

As the granularity of the coarse counting clock becomes finer, the precision of timing intervals between events increases, assuming that systematic error remains constant for each sample. In Chapter 3, the microsecond timer of the Teensyduino 3.2 micro-controller (TEENSY) is examined for the ability to measuring time intervals. Precision is increased in Chapter 4 by increasing the oscillator to 66 MHz, thus reducing the timer resolution to 15 ns. Chapter 6 extends the capability further, by using field programmable gate array (FPGA) efficiency to create timers with 2.5 and 5 ns resolution clocks.

The ability to record interferometric metrology requires the use of an ADC and a photodetector. Section 3.5 provides detail on the internal ADC that is part of the TEENSY. Trying to improve upon bandwidth and resolution, the ADS8371 16-bit 750 kSPS ADC is profiled in Section 4.5.1, and the ADS8411 16-bit 2 MSPS ADC is used with the Artix-7

FPGA Development Board (ARTY) in Section 6.2.2.

Finally, knowing the position of the linear translation stage with precision has been demonstrated to be necessary when examining the velocity profile, PID controller and position synchronized output (PSO) systems. An upgrade to the Renishaw interferometer, required to increase sampling bandwidth, provides a source of spatial metrology in Chapter 5. The results, comparing the linear stage's positioning system, the Renishaw interferometer, and the position based off of counting fringes from a Michelson interferometer are discussed in Chapter 7.

Chapter 3

Micro-controllers

We are stuck with technology when what

we really want is just stuff that works.

–The Salmon of Doubt

– Douglas Adams

This chapter looks at one of the simplest components available for automated data acquisition, the micro-controller. As outlined in Chapter 2, the usefulness of a micro-controller to measure small time intervals (TIs) with microsecond accuracy, and collect data from an external source using an analog-to-digital converter (ADC). Any inherent limitations or errors must be identified such that they may be taken into account.

A description of a micro-controller is introduced in Section 3.1; Section 3.2 looks at the average time necessary for a single specific command to execute on a Teensyduino 3.2 micro-controller. The ability of a micro-controller to record the TI between pulses from a function generator is explored in Section 3.4 and Section 3.5 discusses the ability of the internal ADC to record an accurate representation of a source signal.

3.1 What is a Micro-controller?

A single board computer which integrates a processor unit, memory and peripheral interfaces such as universal serial bus (USB), inter-integrated circuit (I^2C), DAC, and ADC, is referred to as a micro-controller [34]. The advantage of using a micro-controller is the ability to automate tasks, providing an improvement of accuracy and repeatability provided by a faster, and more consistent reaction to trigger events than human ability.

Before 2003, programming micro-controllers was difficult for non-engineers or professionally trained individuals. Programming involved special interfaces to communicate with the controller, direct control of individual registers, and often machine level coding or assembly language. When the Arduino brand micro-controller was developed, it introduced an open-source development platform with simple libraries of functions, making programming easier as the coding was modelled after C++ [35]. The simplification of programming, without intimate knowledge of the architecture of the processor, has allowed access to this technology to the general public. This raises the question of the feasibility of using what has been deemed “hobbyist electronics” in science and research.

Since 2003, many companies have produced Arduino compatible micro-controllers.

The choice to use Teensyduino 3.2 micro-controller (TEENSY) was made because it had the ability to overclock the processor unit, provided an extended command set allowing easier programming, and utilized the full bandwidth of the USB allowing faster data transfer than standard serial communications. Experience using the TEENSY on other projects, such as a weather station and other various tasks around the laboratory, made it a familiar choice.

Before the TEENSY could be used for collecting data, or as a measurement device for the purpose of determining how other lab equipment performs, it was required to profile the controller itself. Determining the time necessary to process common commands, the accuracy of the internal oscillator and the maximum bandwidth of which data could be collected, provided a necessary baseline that is to assess suitability for laboratory data acquisition.

3.2 Determining Average Operation Period

The Agilent DSO-X 2002A oscilloscope has a maximum bandwidth of 70 MHz [36], which limits the period that can be measured between operations to only 15 nanoseconds. 72 MHz is the stock frequency of the TEENSY's internal clock (that can be overclocked to 96, 120 or 144 MHz) that means it is possible to execute a function faster than can be registered on the oscilloscope. Knowing that environmental variables can affect the operation of electronics, thus changing the time to execute a single command, the average period for a command was measured by toggling a single digital output pin high or low after the desired command was given one 1024 times in succession. Since the time to toggle a single pin was negligible compared to thousands of other operations it was ignored in the calculation of the period to execute the command. The resulting square wave period was measured on the oscilloscope (also averaging 2048 samples) and divided by 2048 to determine the average time to execute the command once. The results of the above procedure are depicted in Table 3.1 and Figures 3.1 & 3.2. It should be noted that the internal interrupts, used for monitoring the regular operation of the TEENSY were left on during these tests. It was desired to

depict the average operating periods of specified commands, under regular conditions.

The commands tested relate to reading or writing to the general purpose input/output (GPIO) pins, and are `digitalRead()`, `digitalWrite()`, `digitalReadFast()`, `digitalWriteFast()`, `GPIOx_PDIR()` and `GPIOx_DOR()`. The last four commands are unique to the Teensyduino family of micro-controllers. The `GPIOx_PDIR()` reads a register to return the state of a set of GPIO pins that form a port. The `GPIOx_PDOR()` command writes to the port setting the pin values to the equivalent binary value of an integer. The `digitalReadFast()` and `digitalWriteFast()` are internally based off of the GPIO port commands, but only return the value of a single pin.

The `millis()` and `micros()` commands return the number of milliseconds or microseconds since the TEENSY powered on. Both return a 16-bit unsigned integer which may overflow periodically requiring compensation if long time periods are being monitored.

The internal ADC, onboard the TEENSY, is read through the `analogRead()` command whose parameter determines which designated analog pin is read. The performance of the ADC dictates the performance of the TEENSY as a DAQ.

The final command tested, null operation instruction (NOP), is an assembly language call which adds the shortest delay possible, supposedly executing consistently in only one or a few machine cycles. NOP is constructed such that the state of registers, flags or memory are kept static, while providing a timing delay normally for synchronization purposes.

Table 3.1: Average Period to Execute a Command Measured with an Oscilloscope. The operational time for 2048 (100000 for NOP and 1000 for `analogRead()`) iterations of a specific command was measured directly on an oscilloscope. The oscilloscope was set to average 2048 additional periods. The average time to execute a single command was calculated by dividing the period measurement by the number of iterations of the specific command. The error was indeterminate as statistics couldn't be obtained from automatic averaging on the oscilloscope.

Clock Frequency (Mhz)	72	96	120	144
Clock Period (ns)	13.9	10.4	8.3	6.9
Command	Execution Time per function call [ns]			
`digitalRead()`	282.23	211.91	170.41	141.60
`digitalWrite()`	435.06	326.66	261.72	217.77
`digitalReadFast()`	100.59	75.68	61.23	51.07
`digitalWriteFast()`	59.03	44.78	36.38	30.32
`GPIOx_PDIR()`	100.63	75.93	61.18	51.07
`GPIOx_PDOR()`	59.08	44.78	36.38	30.33
`micros()`	463.38	348.10	279.05	232.18
`millis()`	114.75	86.43	69.34	58.11
`analogRead()`	8257	10376	7969	8608
`NOP`	58.89	44.68	36.08	30.03

The oscilloscope method of measuring the execution time of micro-controller functions, shown in figure 3.1, was unable to determine the uncertainty in the measured values as only the average period could be displayed on the display. To determine the deviation the micro-controller code was modified to record the period for 2048 iterations of a command, in microseconds, for ten thousand periods. The internal memory constraint of the TEENSY made the limit of ten thousand periods pertinent, as any serial communications would add

unwanted delays. Due to the long time required to perform the `analogRead()` command only one thousand iterations were measured and the short period for the NOP command required one million iterations. The average period to execute a single given command were then calculated by dividing the average period by ten thousand and the error was taken as the standard deviation of the periods also divided by ten thousand. The results of repeating the measurement of execution time, outlined above, is in Table 3.2 and Figures 3.1 & 3.2.

Table 3.2: Average Period to Execute a Command Measured using Micro-controller Oscillator. The operational time for 2048 (100000 for NOP and 1000 for `analogRead()`) iterations of a specific command were measured 10000 times and then mean execution time for a single command with error was calculated.

Clock Frequency (MHz)	72	96	120	144
Clock Period (ns)	13.9	10.4	8.3	6.9
Command	Execution Time per function call [ns]			
`digitalRead()`	321.4 ±0.3	241.0 ±0.2	192.9 ±0.2	160.7 ±0.2
`digitalWrite()`	503.0 ±0.2	376.8 ±0.2	301.5 ±0.2	251.2 ±0.2
`digitalReadFast()`	84.5 ±0.2	63.4 ±0.2	50.8 ±0.2	42.3 ±0.2
`digitalWriteFast()`	56.7 ±0.2	42.5 ±0.2	34.1 ±0.2	28.4 ±0.2
`GPIOx_PDIR()`	84.5 ±0.2	63.4 ±0.2	50.8 ±0.2	42.3 ±0.2
`GPIOx_PDOR()`	56.6 ±0.2	42.5 ±0.2	34.1 ±0.2	28.4 ±0.2
`micros()`	517.0 ±0.1	387.4 ±0.2	309.9 ±0.2	258.2 ±0.2
`millis()`	112.4 ±0.2	84.4 ±0.2	67.5 ±0.2	56.2 ±0.2
`analogRead()`	8014 ±7	10388 ±6	7939 ±5	9283 ±2
`NOP`	56.7 ±0.3	42.5 ±0.2	34.1 ±0.2	28.4 ±0.2

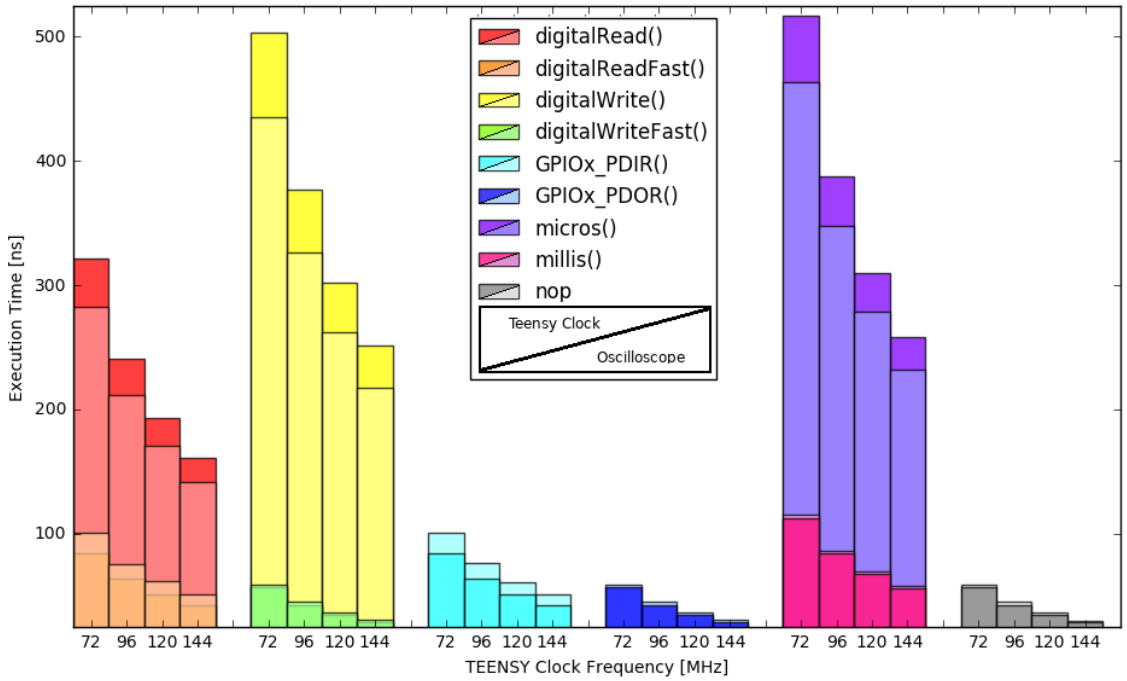


Figure 3.1: The execution times from Tables 3.1 & 3.2 have been plotted to make comparison of results graphical. The bold version of each colour represents the measurements taken using the TEENSYs internal clock (Table 3.2), where the lighter version of the same colour represents the oscilloscope method of measurements (Table 3.1). Error was not represented as it was too small for the y-axis scale. The `analogRead()` command was omitted as measurement was too large for the y-axis scale. Similar commands (i.e. read or write) were stacked to illustrate the advantage of the "fast" version of the command. The `GPIOx` were kept separate so they wouldn't block the "fast" version bars.

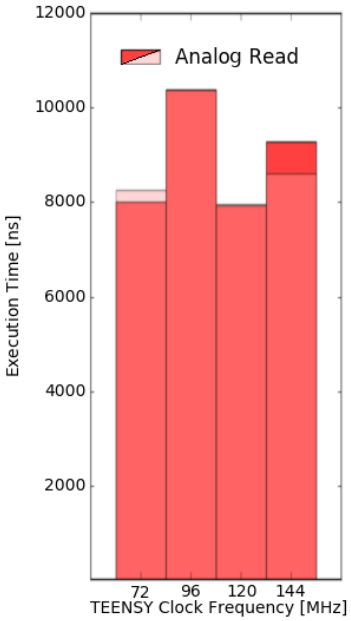


Figure 3.2: Analog Read execute time for Teensy 3.2. Due to the ADC operating on a separate internal clock, overclocking the micro-controller had adverse affects causing a longer average period to execute. Requiring 8 μs to execute constrains the upper limit to the sampling rate to 125 kHz.

The largest difference between the two different trials, attempting to find the average operational period for specified commands, is what occurs after running a command multiple times. In Table 3.1, a GPIO pin was toggled states, where for Table 3.2 the period was calculated using the `micros()` command and stored in internal memory. Since the oscilloscope was able to average samples continuously, the sample size was larger and may have attributed to the slightly smaller execution times, seen in Table 3.1.

Except for the case of the `analogRead()`, the scale of the error depicted in table 3.2 and Figure 3.1 is similar to the expected amount of jitter produced by the micro-controllers own clock. Shown in Figure 3.2, the fairly constant `analogRead()` time can be attributed to the ADC using its own sampling clock and averaging circuitry [37]. The overclocking of the processor offered no advantage to analog operations, and in some cases made it require longer execution time.

3.3 Serial Communication time

An outgoing serial buffer waits until it is either full, or a time-out condition is met, before transmitting data. If the amount of data being sent is greater than the buffer size, it is held until the initial buffer has been transmitted. This creates a situation where a time delay is introduced while the data is shifted onto the transmit buffer, and in extreme cases may cause data to be lost. The serial buffer on a TEENSY is set to only 64 bytes. This can be modified by editing the *serial1.c* and *serial6.c* files (located in the Arduino hardware directory for the TEENSY3). A larger buffer alone will not solve the problem of sending too much data, rather it will delay the overflow as transmit time is a function of the buffer size. The amount of data being sent will determine the overall data rate required and the sampling rate can be adjusted to compensate for the serial transmission.

The window is defined as the total length, in bits per transmission, of a serial packet. This includes start bits, data, parity, and stop bits. The settings chosen were one start bit, eight data bits, no parity, and one stop bit. The window is ten bits long for every byte of data sent. It should be noted that these extra bits are added during transmission and are not stored in the transmit buffer.

Serial communications require a sending clock that is well defined with a known frequency. The receiving end synchronizes its clock with the falling edge of the start bit, thus clocking errors only accumulate during a single window. The goal of the receiving end is to sample the level of each bit near the centre of the bit-period and avoid the edges where the data is unreliable. The error tolerance in the period of the clock is 2.0 % maximum error [38]. The TEENSY can synthesize standard baud rates up to 4.608 Mbps within ± 0.79 %. The exception being 300 bps, where the error in period has been observed to be 144.14 %, when the processor is clocked to at least 96 MHz [39]. The error in the baud rate is a product of the internal clock not being an exact multiple of the required serial clock, and some combinations do not work as well as others. It should be noted that the TEENSY communicates at 12 Mbps over the USB, regardless of serial parameters selected. The USB

port is still subject to the constraints of the serial buffer size. Serial protocols and the clocking considerations apply to serial communications using GPIO pins between devices. Other Arduino based micro-controllers may not utilize the full bandwidth of the USB port, and may be subject to the limitations of clocking and serial baud rates.

3.4 Timing Periods between Events

To determine average velocity of an object in motion, the change in position over a change in time is required. The position was measured by the Renishaw interferometer which will be discussed in Chapter 5. The TEENSY is required to measure the period of time between PSO pulses emitted every time the linear stage has moved a desired distance (the accuracy of these pulses will be discussed in chapter 7).

Two scenarios were tested for acquiring timing data. First, using the memory to store values from the `micros()` command for each rising edge, then transmitting the entire data set, restricted to ten thousand samples as a result of the TEENSY internal memory constraints. Second, transmitting the `micros()` value for every rising edge immediately. In both cases the data was transferred in a binary format, requiring only four bytes per sample, where sending strings of American Standard Code for Information Interchange (ASCII) characters would require much more overhead. It should be noted that the `micros()` command for Arduino has a stated error of $\pm 4\mu s$ for standard 16 MHz micro-controllers. System interrupts were disabled during data collection, so any possible error in the timing in excess of thos from the input source is attributed to delays in the code executed in the loop at the time of recording.

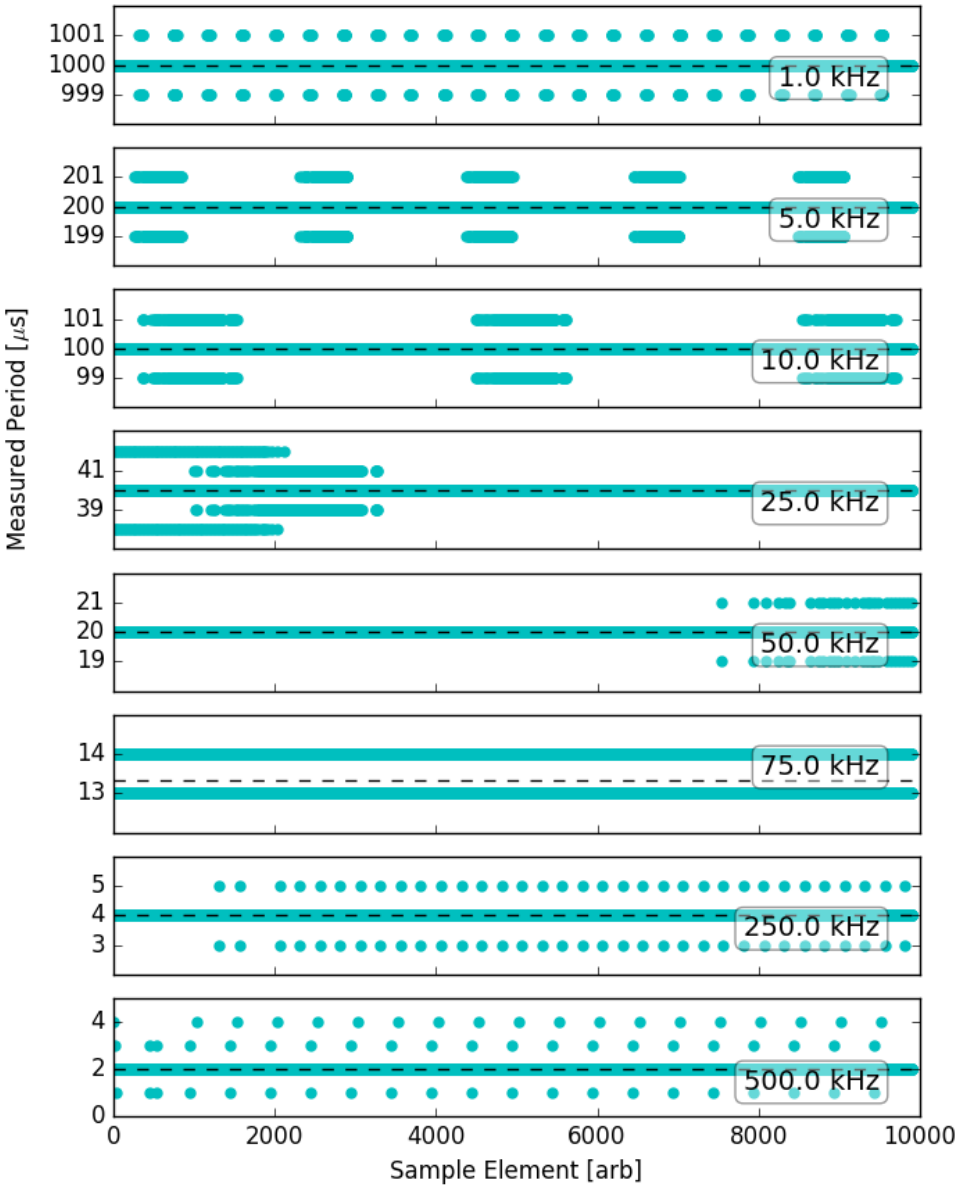


Figure 3.3: Measuring the period of a square-wave, generated by an Agilent function generator, using a TEENSY. The error of the period generated by the Agilent function generator is $< 0.1\%$. The ideal period is marked with the dashed black line. Ten-thousand samples were taken, then transmitted upon completion through serial communications. The periodic delays were attributed to the microsecond resolution of the coarse clock and the slight phase offsets introduced by jitter on the function generator and timing clock.

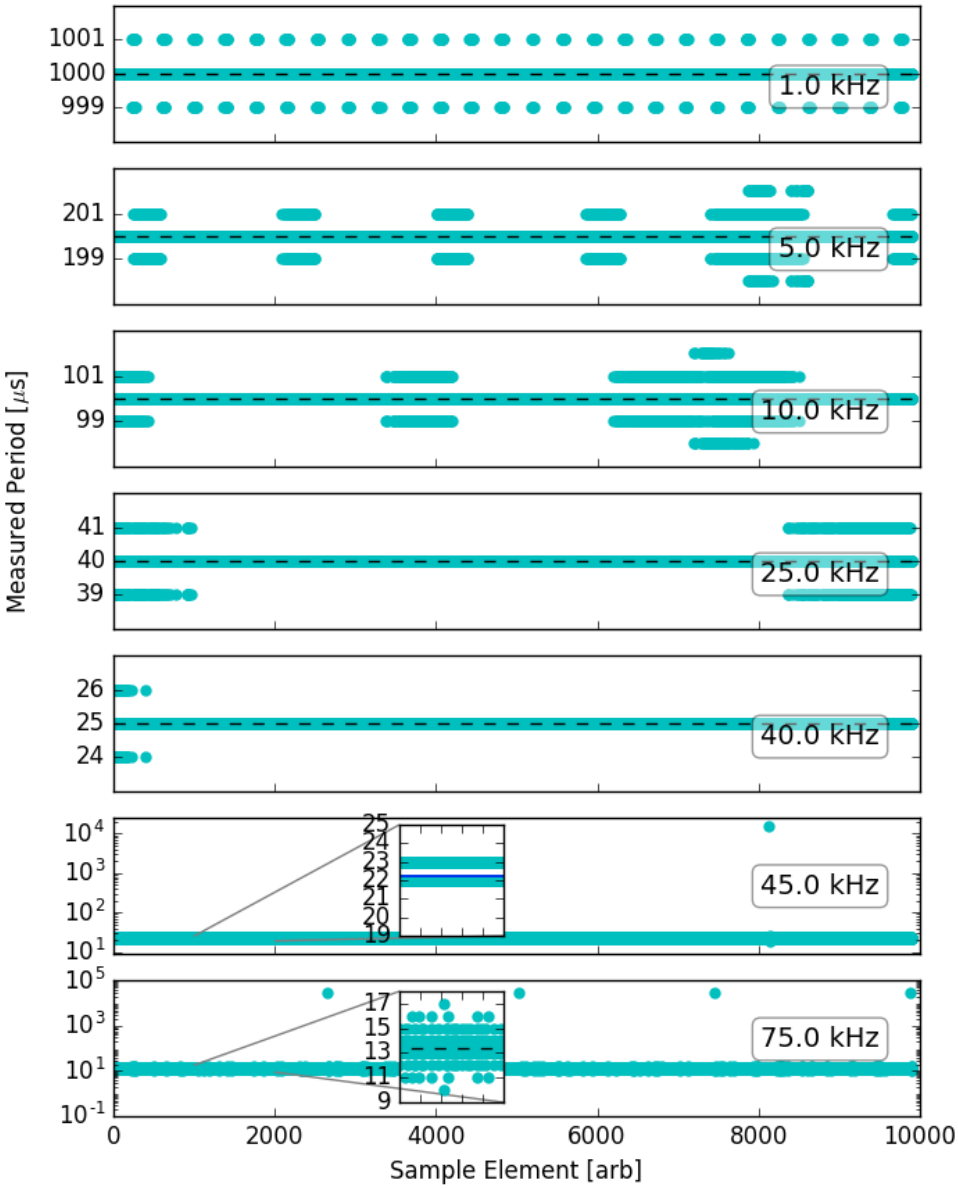


Figure 3.4: Repeating the Figure 3.3 test, to measure the period of the rising edge of a square-wave, with the results immediately transmitted via serial connection instead of being stored in local memory. Delays in the recorded time values can be seen as early as 45kHz, and were attributed to the delay introduced by serial communication.

In Figure 3.3, the 25 kHz data is the only set which demonstrates greater than one microsecond from the expected value, until a frequency of 500 kHz is reached. It is assumed

that the longer delays are periodic and maybe created by system watchdog interrupts that monitor the micro-controllers internal processes. The longer periods do appear in Figure 3.4, for 5 kHz and 10 kHz, which are not attributed to the serial communications. Considering that the number of erroneous points was a relatively small proportion of the total data, and the scale is the same as the microsecond resolution, the results are as expected. In Figure 3.3, a periodic error occurs in all but the data collected at a rate of 75.0 kHz. It should be noted that the relative error increases as the period of the square-wave approaches the resolution of the microsecond clock. All of the error when measuring the period of the function generator pulses was attributed to the TEENSY, as the error introduced by the function generator was $< 0.1\%$ which was smaller than the resolution of the microsecond clock. Although measuring a 250 kHz period within a single clock pulse appears acceptable, the actual measurement was only within 25 % of the period, for many of the samples.

Expecting a count of ± 1 clock tick as the smallest error in the measurement of a TI, errors larger than this were observed in the 45 kHz and 75 kHz data sets of Figure 3.4, compared to the acceptable results up to 250 kHz found in Figure 3.3. With the only difference between trials being the immediate transfer of data upon recording the sample compared to storing samples until memory was full, the increase in error at smaller frequencies was attributed to delays related to serial communication.

3.5 Profile of the On-board 16-bit Analog to Digital Converter

Using an Agilent arbitrary waveform generator to produce a ramp wave with a period of sixty seconds, the TEENSY was triggered to take an `analogRead()` at a rate of 25 kHz, with the expectation that every bin would receive an equal number of samples over a sufficient amount of time, as the sampling rate was much greater than the frequency of the ramp wave. The resolution of the internal ADC can be set to a bit-depth up to 16-bits, providing 65536 individual bins. The bins represent an integer value between zero and the 3.3 V reference voltage. The ramp wave was set to provide a range of voltages 0.1 - 3.1 V,

to demonstrate that the full range of the ADC responded linearly. Figure 3.5 was created by sampling in this configuration for a twelve hour period.

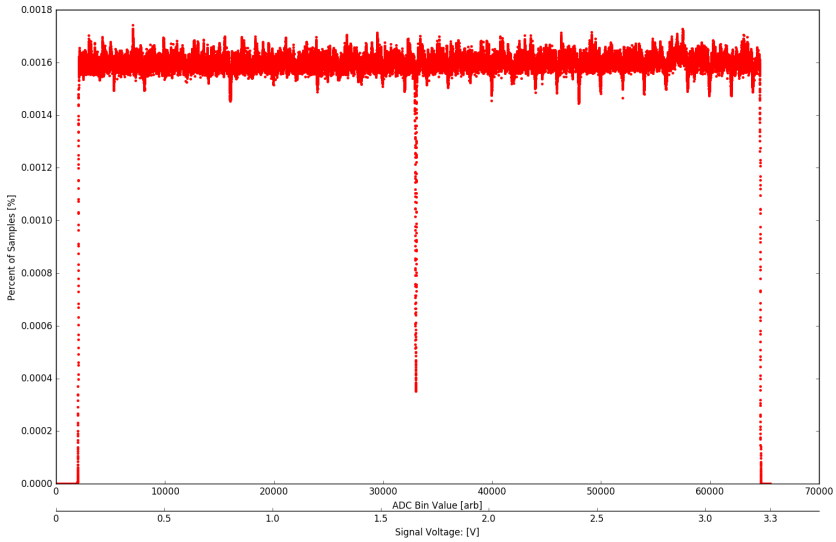


Figure 3.5: A histogram of the population of samples occurring in each 16-bit ADC bin, demonstrating the linearity of the onboard converter for the Teensyduino 3.2. A voltage range, 0.1-3.1 V, was used to provide 90% full-scale response to prevent clipping.

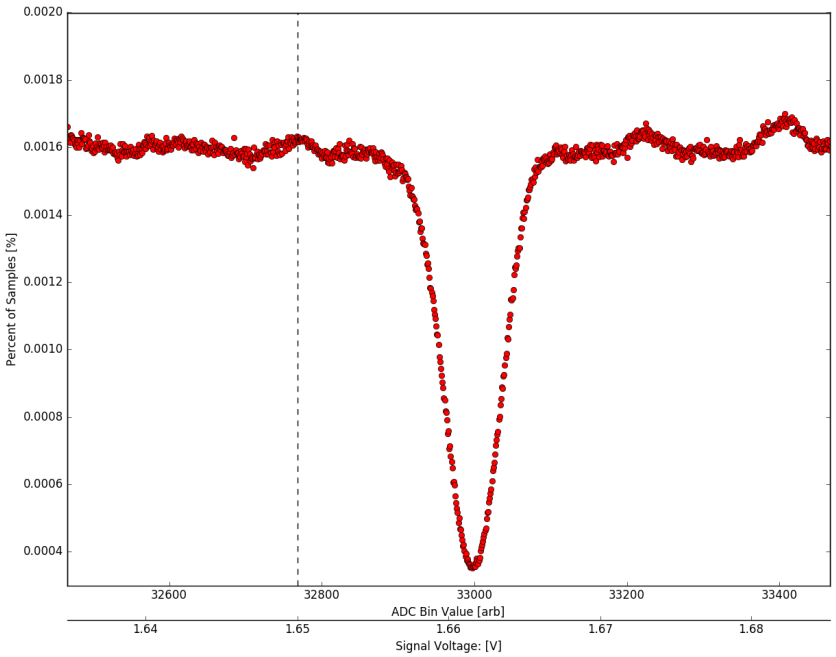


Figure 3.6: Magnification of the spike that appears in the linearity histogram of the response of the internal ADC of the Teensyduino 3.2. Possibly caused by the interleaving of the two internal ADCs, requiring further investigation. For reference, the 32768^{th} arbitrary ADC bin marking a signal voltage half of the reference is marked with the black dashed line.

Figure 3.5 produced expected results with the exception of the spur occurring precisely at the half-way point of the ADC range. The TEENSY utilizes two ADCs to increase bandwidth of the conversions by interleaving [37]; samples are acquired by alternating between the ADCs. This is problematic when there are timing, gain, bandwidth and offset mismatches which causes the two ADCs to produce slightly different results [40]. The spur, magnified in Figure 3.6, is assumed to be caused by the two ADCs having slightly different buffers and reference voltages, but further investigation was left as a future exercise. The bins at the extrema, having little or no samples, was produced by giving the ramp wave an offset and amplitude such that it would stay between 0.1 V and never pass above the reference voltage of 3.3 V.

To measure the SNR, as outlined in Chapter 2.5.2, a sine-wave with a period of ten seconds and a range of voltages, 0.1 - 3.1V, was sampled at 25 kHz. This was repeated with the

internal ADC set to 12-bit and 16-bit bit-depths, and the internal averaging set to off, four, eight, or sixteen samples. The internal averaging of samples occurs at the rate determined by the bus clock frequency and clock divider [37], which were left at their default settings. The Arduino programming language lacks access to the clocking registers, thus changing the clock from defaults goes beyond the scope of this investigation. Averaging beyond 16 samples was not tested as the conversion time was greater than the period of the 25 kHz sampling rate.

The observed data were fit with an ideal ADC response, as outlined in Chapter 2.5.1. An example of the fitting is provided in Figure 3.7, where the ADC was configured to collect data with a bit depth of 12-bits and averaging eight samples.

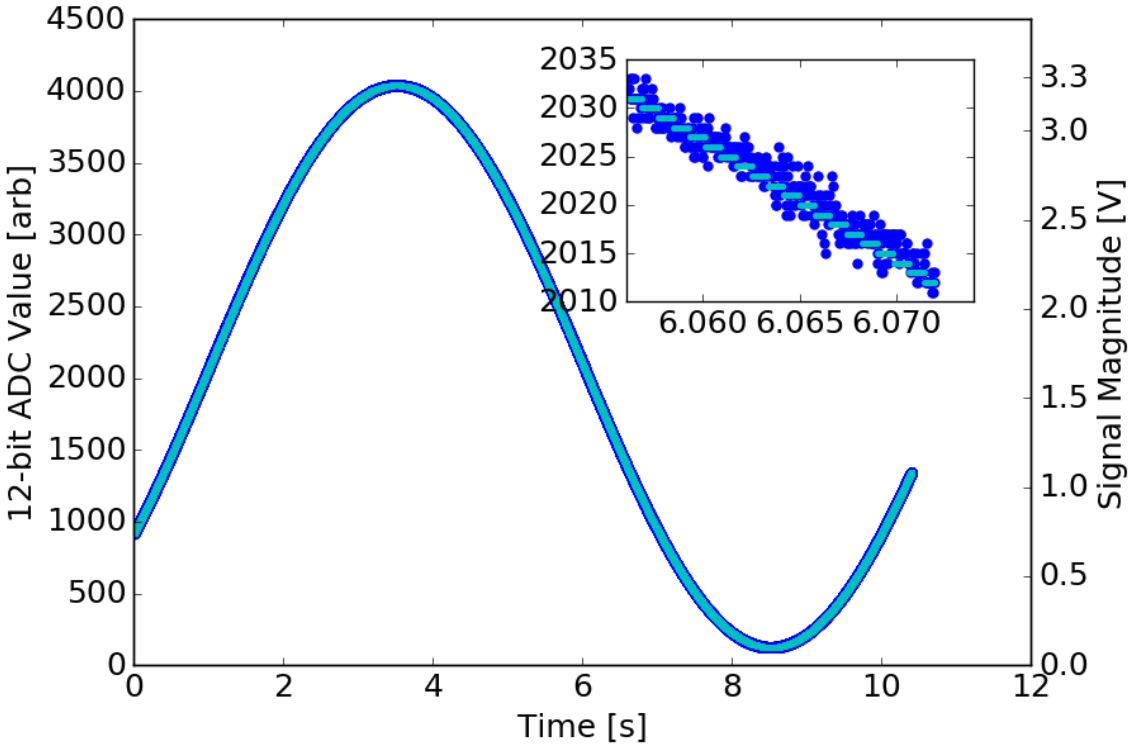


Figure 3.7: The least squares fit of the sinusoidal signal (blue), allowed for the calculation of the ideal ADC response (cyan). The discrete levels of the ideal response can be identified in the magnified inset. The signal had a frequency of 0.1 Hz and was sampled at 25 kHz. The inset demonstrates the step-like digitization of the ideal ADC response in contrast to the actual signal.

As the frequency of the sine-wave was considerably smaller than the sampling rate, many samples should fall to a single ADC value as depicted by the ideal ADC fit in the inset of Figure 3.7. A least-squares fit was used to estimate the amplitude, frequency, phase and offset of the data and to establish the parameters utilized in calculating the ideal ADC values.

To estimate the noise, attributed to 60 Hz line noise and inefficiency in the ADC itself, the difference between the ideal response and observed data was calculated. Two histograms, Figures 3.8 & 3.9, illustrate the level of noise, measured in mV, and the smoothing effects of averaging samples, for 12-bit and 16-bit settings. Since internal averaging is performed at the fastest rate that the ADC can operate, a decrease in noise and corresponding increase in ENOB was expected, as described in Section 2.5.4.

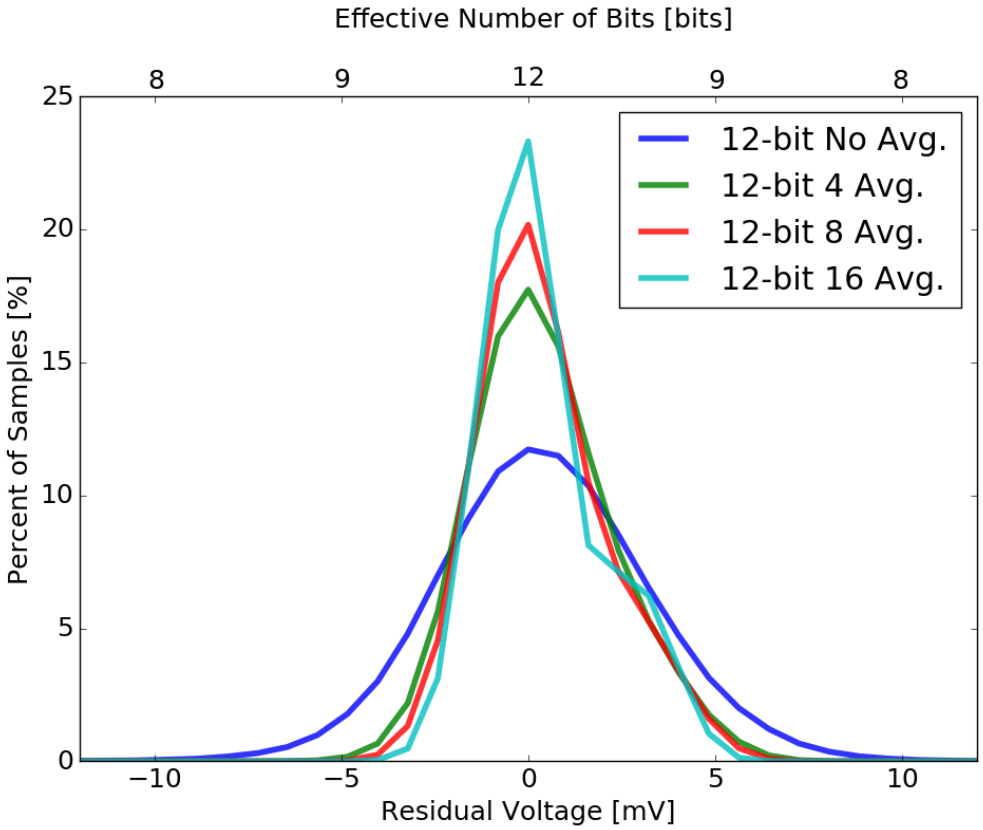


Figure 3.8: The difference between ideal and observed internal ADC was used to demonstrate the noise in the signal, when the ADC was set to 12-bit resolution. The smoothing effect of averaging is demonstrated by the decreasing width of the distribution, as averaging increased.

When no averaging is performed, 12-bit and 16-bit residual results appear Gaussian, symmetrical and comparable in width, in Figures 3.8 & 3.9. This suggests no advantage to a 16-bit value, as the finer bin size did not reduce the level of noise in the signal, or provide any increase in the level of detail. The asymmetry depicted in the noise of the higher averaged samples is believed to be due interleaving of the two internal ADCs, possibly slightly different ground and reference voltages may also be contributing and remains under study. Averaging more than eight samples didn't decrease the width of the distribution further, implying that the increased period to convert a sample is not advantageous.

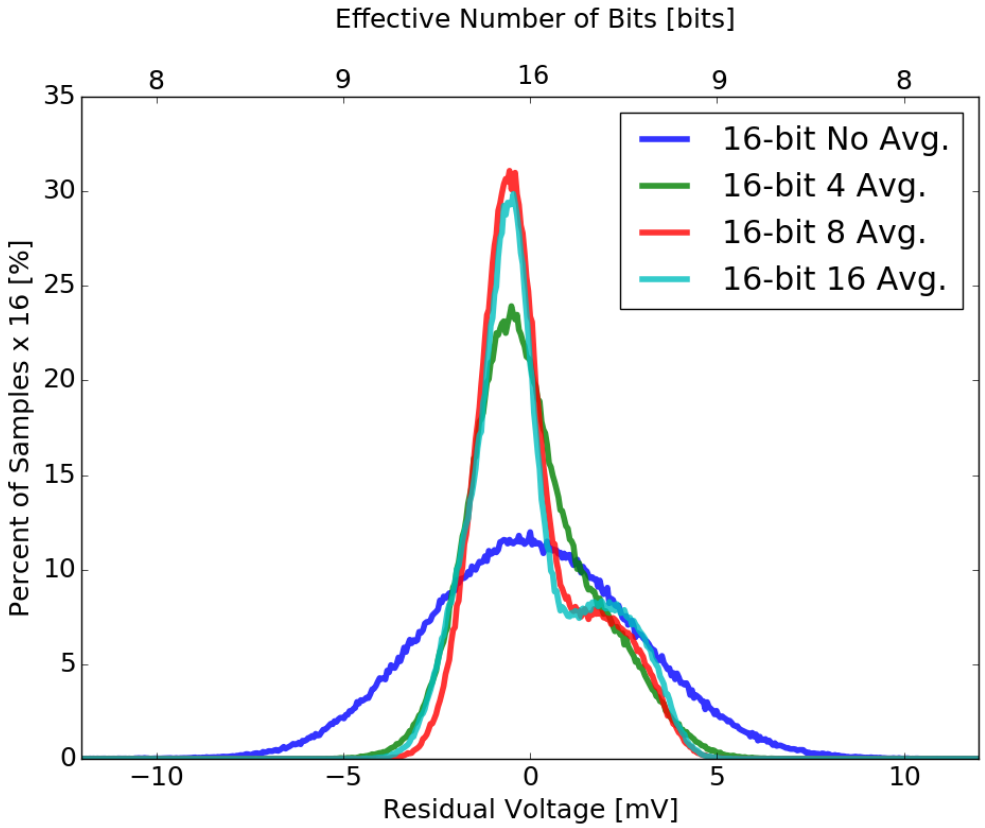


Figure 3.9: 16-bit observations were used to find the estimated the level of noise acquired on the internal ADC. The y-axis was scaled by a factor of 16 to account for the increased number of bins between 12-bit and 16-bit observations, making it easier to compare to Figure 3.8.

The SNR was calculated as described in 2.5.2 for each bit depth in Figures 3.9 and 3.8. The result, shown in Figure 3.10, allowed for comparison of bit-depth and averaging for the TEENSY micro-controller.

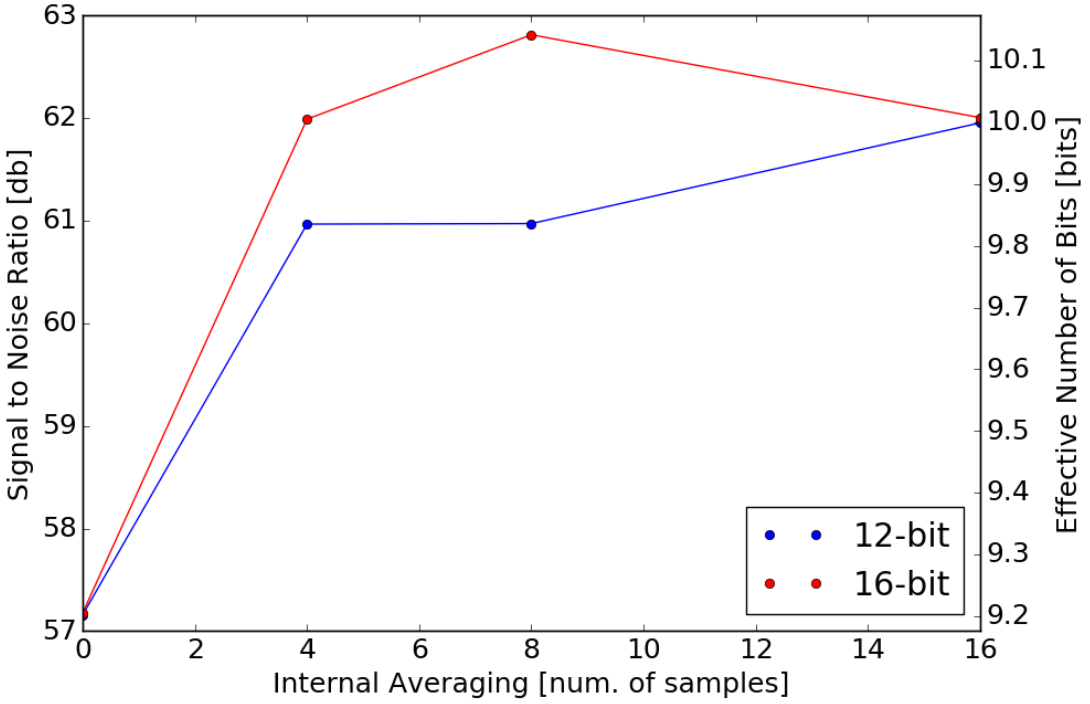


Figure 3.10: The SNR and ENOB were determined for 12-bit and 16-bit resolution settings. The effect of averaging can be seen by the increased the level of SNR, where the higher the value is considered a better signal.

The ENOB demonstrated in Figure 3.10 is below the reported 13-bits usable [39] for the onboard ADC. The SNR was not considerably increased by the higher bit depth of 16-bit, compared to the 12-bit. Without exploring in depth the TEENSY's register settings at the machine level, the Arduino language did not provide a simple way to explore the deficiencies observed in the TEENSY's onboard ADC.

3.6 Conclusions

The TEENSY had an increasing relative error in the measuring of TIs as the resolution of the clock was approached, demonstrated in figure 3.3. The expected sampling rate of 25-30 kHz (determined by the number of samples desired, expected velocity of the stage, and wavelength of light being investigated), resulted in $< 5.0\%$ relative error in the measuring of TIs, which occurred at a periodic interval. The error was attributed to deficiencies in the

`micros()` command and possible delays due to clock jitter and serial communication.

Continuous recording of data is also required by the final test bed instrument, which means that sampling data without serial interruptions is impossible due to the limited memory space on the micro-controller. Demonstrated in figure 3.4, as the sampling rate increased, a point where the serial buffer becomes full occurs causing a long delay in the operation and missed sampling occurs starting at 45.0 kHz. Even if the few points of extreme difference could be accounted for, the act of sending data introduced error at the microsecond level as seen by the increase deviation, from $< \pm 1\mu$s to $> \pm 3\mu$s, comparing the 75 kHz data sets in figures 3.3 and 3.4.

The ability to read data, such as light intensity from at least one photo-diode, would also be an asset for use in interferometry, making a complete DAQ. The average period to request a single value from the onboard ADC limited the sampling rate under to 100 kHz. When delays from serial transfer are accounted for, the maximum sampling rate dropped below 30 kHz with no data loss.

It is obvious, for the intended purpose, the TEENSY is inadequate on its own. The main advantage of the TEENSY was the ability to read and write to its GPIO pins within tens of nanoseconds, making it an ideal interface between a discrete coarse counting timing circuit or an external high-speed ADC with parallel digital output. Chapter 4 explores the use of an external digital coarse counting clock to rectify the deficiencies of the `micros()` command, and a discrete 16-bit high-speed ADC that can maintain a higher sampling speed of 750 kHz.

Chapter 4

Advancing Micro-Controller Capabilities with Discrete Electronics

---

This is obvious. But consider:

our technologies are compatible. Cybermen plus Daleks

- together, we could upgrade the universe.

– Cybermen, Doctor Who (BBC)

It has been substantiated by the profiling of the TEENSY, as discussed in Chapter 3, that the micro-controller had the ability to read digital pins at high speed, reading eight general purpose input/output (GPIO) pins in less than 60 ns, but could only measure periods between events within a few microseconds. It was lagging when reading values from the analog-to-digital converter (ADC), requiring up to 10 μs to digitize a sample. By adding an external clock source and coarse counting circuitry, as discussed in Section 2.3.1, to increase the timing resolution, the deficiency of the `micros()` command can be circumvented. A discrete ADC, the ADS-8371 made by Texas Instruments [41], which can operate at 750 kHz with a 16-bit parallel digital output was incorporated to replace the onboard ADC.

Section 4.1 discusses the prototype circuit used to develop the original data acquisition system (DAQ) design. Sections 4.2 and 4.3 looks at some of the pitfalls experienced when using a binary ripple counter and a possible correction algorithm to allow more bandwidth while collecting data. The move to asynchronous counting integrated circuits (ICs) is presented in Section 4.5, followed by testing of various components in Section 4.7.

4.1 Proof of Concept

The prototype circuit was tested using a square wave generator, which provided evenly spaced intervals that varied by $< 0.1\%$ of the oscillator period. This provided predictable data which allowed for straightforward diagnostics and the development of an algorithm to correct issues found in the data.

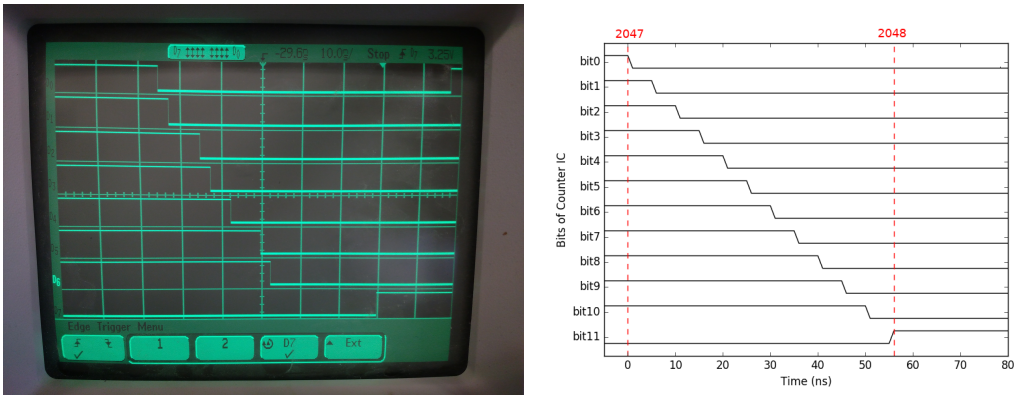
The first prototype circuit, used only for placing a time stamp on the rising edge of the provided square waves, contained a 12-bit ripple counter, two D-type memory [42] ICs and a 27 MHz oscillator [43]. The oscillator required a frequency divider, dividing it in half to 13.5 MHz, to reduce it to a frequency within the ripple counters specifications. The ripple counter had to increment every falling edge of the oscillator and the buffers had to immediately latch values once the data request trigger (DRT) signal went to a high state.

The chosen D-type memory [44] only latched a value from the counter on the rising edge of the latch (LCH) signal and would hold the value independent of the LCH state until the next rising edge. Table 4.1 illustrates that the required ripple counter clock (CLK) signal matches the oscillator (OSC) to fulfill the first requirement. Values were placed into the memory on the falling edge of the OSC only when the DRT was low, holding the last recorded value upon the rising edge of the DRT. The micro-controller then records the value after a delay to ensure the buffers have settled. The duration of the DRT pulse can be quite long, in our case 5 μs, compared to the 74 ns period of the oscillator, allowing ample time for the micro-controller to read the value, as long as it completes before the next DRT pulse. Figure 4.2, in Section 4.3, illustrates this process.

Table 4.1: The logic determined to control the counter and memory circuit. As desired, the required CLK signal matches the OSC, and the required LCH is determined by OSC $\overline{AND}$ DRT logic.

Required CLK	**Required LCH**	**OSC**	**DRT**	**OSC $\overline{AND}$ DRT**
0	0	0	0	0
0	0	0	1	0
1	1	1	0	1
1	0	1	1	0

Table 4.1 does not account for intentional delays introduced in the circuit (τ_1 and τ_2 in Figure 2.5) used to manipulate the timing of when the counter and buffers received the OSC pulse. From this table, the only situation where required CLK and LCH values exist coincide with the OSC in a high state and the DRT in a low state, resulting in the desired action of holding a value on the buffers while the DRT is active. The CLK pulse is merely the OSC signal delayed by τ_1 and the LCH pulse was the inverse logical AND ($\overline{AND}$) of the OSC and DRT signal delayed by τ_2, where τ_1 and τ_2 are the delays described in Figure 2.5.



(a) Oscilloscope results.

(b) Simulated results

Figure 4.1: Monitoring bits 0 through 6, and triggering on the twelfth bit (labelled D7), the actual time to ripple through all twelve bits can be measured(a). To reduce jitter the oscilloscope was set to average 4096 samples. The horizontal divisions mark 10 ns intervals. From the falling edge of D0 to the rising edge of D7, the estimated time is 55 ns. To visualize the process occurring, the results were modelled in Python (b), where the time required from the falling edge of the 0^{th} bit to the rising edge of the 11^{th} bit was also 55 ns.

To allow the ripple counter to toggle all the necessary bits before the rising edge of the oscillator, the 12-bit counter requires approximately 5 ns to toggle a single bit [45], and 60 ns to toggle all twelve bits (worst case scenario). The time to ripple all twelve bits was verified on a digital oscilloscope, depicted in Figure 4.1, and was estimated to within 55 ns. The maximum frequency would have a period twice the settling time to assure the settling of the counter doesn't collide with the latching of counted values, limiting the clock frequency in this case to 8.3 MHz. For use with a faster oscillator (13.5 MHz in the prototype circuit) it is necessary to find a way to correct reading values before the binary counter has settled, called a "bit-wise collision" due to the combining of two integer numbers at a certain bit level, that occurs when the number of bit transitions on the ripple counter requires more time than half of the oscillator period. This would not be an issue if the clock stopped when the value was read, even at the maximum ripple counter frequency of 20 MHz, but a systematic error caused by the time required to read the value on the counter would affect the period being measured.

4.2 The Argument for Using a Ripple Counter

In a laboratory setting, the use of a ripple counter for high-frequency operations is discouraged, when synchronous counters are easily obtainable and prevent any of the issues observed. The use of asynchronous technology can still have benefits, and sometimes even be desirable. The most important difference between asynchronous and synchronous circuitry is with design, the parts of an asynchronous circuit can be disabled or switched off during times when not needed [46]. For instance, the twelfth bit of the prototype circuit turns on the least, therefore half the time the flip flop controlling the state of the bit can be powered down. Usually, asynchronous circuitry requires fewer gates, which provides a simpler design and less power draw. The imaging detectors on some satellites require low power consumption to limit heating and reduce the draw on a limited power source, requiring only $100\,\mu$W peak power to prevent the detector from being brighter than the source [47]. With this in mind, power-sensitive situations like the constraints dictated by satellite manufacturers, asynchronous designs may be desirable.

4.3 Asynchronous Ripple Counter Correction Algorithm

Consider the case of a data acquisition system that timestamps each DRT with the value that is currently on the ripple counter IC. The falling edge of an oscillator pulse is used as the clock signal for an asynchronous counter IC and upon the rising edge of the oscillator pulse, the value is stored in local memory. The DRT causes memory to latch, preserving the last count value in memory without interrupting the counting, and provides ample time for a micro-controller to record the latched binary value. The maximum settling time for the count-up IC must be less than half the period of the oscillator, assuming a 50% duty cycle, to ensure that the count settling period and rising oscillator pulse are complementary.

To achieve greater time resolution, it was necessary to investigate the effect caused by driving the frequency of the oscillator such that the period is shorter than the maximum settling time of the binary counter. It should be noted that the counter IC is not driven faster

than its maximum specified count rate, but the maximum time required to toggle all the necessary bits is longer than the half period of the driving oscillator, occasionally causing the memory to request a value while bits are still changing. Thus, the counter is not missing any input pulses, but may LCH a value that is different than requested due to the trigger being asynchronous to the settling of the ripple counter. The circuit is over-clocked in the sense that a request to LCH a value may occur before the bit-ripple is complete. In this situation the value latched will be an amalgamation of the binary digits from two integers $\chi \texttt{and} (\chi + 1)$.

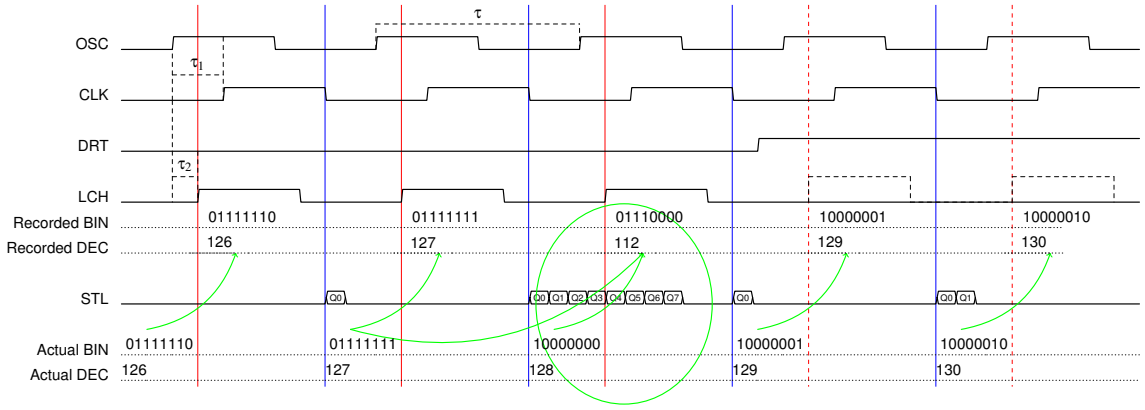


Figure 4.2: Timing diagram illustrating the sequence occurring in the prototype circuit while counting from 126 to 130. The counter increments on the falling edges (blue) and the latch records the time on the rising edges (red). The settling time (STL) of the binary ripple counter is depicted such that the current changing bit progresses in time. Note the third rising edge of the LCH occurs while the STL is currently at bit Q4 and would be a point where a collision would occur if the DRT were to request a value.

The adjusting of the timing delays (τ_1 and τ_2 from Figure 2.5) affected the duty cycle of the timing signal (where the rising edge triggers the counter and the falling edge latches the buffers), simplifying the identification of the bit-wise collisions by tuning the event location to the 10^{th} bit. The resulting 12-bit value (χ') was a composite of two numbers, $\{\chi \texttt{and} (\chi + 1) \,|0 \leq \chi \leq 4095\}$. This raises the question of what outcomes are possible, in order to understand how the resulting value was formed.

Let χ be an arbitrary integer such that $\{0 \leq \chi \leq 4095\}$. Consider the case when χ is even. In binary, the LSB must be a zero. For $\chi \rightarrow (\chi + 1)$, the 11^{th} most significant bit

(MSB) does not change from χ and only the LSB toggles to a 1. Let η, where $(0 \leq \eta \leq 12)$, be the bit-wise position that these two integers will be combined such that $(12-\eta)$ MSB of χ and η LSB of $(\chi+1)$ are combined to make a single 12-bit integer. It is simple to see that when η is zero, the resulting number must be χ. For $(1 \leq \eta \leq 12)$, the result must be $(\chi+1)$ as the LSB is the only bit to change. Therefore, for all even values of χ, only two possible results can occur, as in the example depicted in table 4.2.

Table 4.2: Example of different bit-level collisions, using an 8 bit even value where $\chi = 126 = 01111110$ and $\chi+1 = 127 = 01111111$. The result (χ') can only have two outcomes, χ and $\chi+1$.
† *The* | *denotes the position of* η *where bits to the left are from* χ *and to the right are from* $\chi+1$.

η	Result (binary)†	Result (decimal)	χ'
0	0 1 1 1 1 1 1 0|	126	χ
1	0 1 1 1 1 1 1|1	127	$\chi+1$
2	0 1 1 1 1 1|1 1	127	$\chi+1$
3	0 1 1 1 1|1 1 1	127	$\chi+1$
4	0 1 1 1|1 1 1 1	127	$\chi+1$
5	0 1 1|1 1 1 1 1	127	$\chi+1$
6	0 1|1 1 1 1 1 1	127	$\chi+1$
7	0|1 1 1 1 1 1 1	127	$\chi+1$
8	|0 1 1 1 1 1 1 1	127	$\chi+1$

When χ is odd, the LSB must be a one. Adding one to χ will result in all the bits toggling until the least significant zero is reached. Let the position of the least significant zero be called ϕ, then χ and $(\chi+1)$ have $(12-\phi)$ MSB in common. As before, $\eta = 0$ results in χ, but $(1 \leq \eta < \phi)$ will result in ϕ different potential values. The remaining bits $(\phi \leq \eta \leq 12)$ all have the same values as $(\chi+1)$. For even values of χ there are $\phi+1$ possible different

results, as illustrated by table 4.3.

Table 4.3: Example of different bit-level collisions, using an 8 bit odd value where $\chi = 127 = 01111111$ and $\chi + 1 = 128 = 10000000$. The result can have up to nine outcomes depending on where the bit-wise collision occurs.
† *The* | *denotes the position of* η *where bits to the left are from* χ *and to the right are from* $\chi + 1$.

η	Result (binary)†	Result (decimal)	χ'
0	0 1 1 1 1 1 1 1|	127	χ
1	0 1 1 1 1 1 1|0	126	$\chi - 2^0$
2	0 1 1 1 1 1|0 0	124	$\chi - 2^1 - 2^0$
3	0 1 1 1 1|0 0 0	120	$\chi - 2^2 - 2^1 - 2^0$
4	0 1 1 1|0 0 0 0	112	$\chi - 2^3 - \cdots - 2^0$
5	0 1 1|0 0 0 0 0	96	$\chi - 2^4 - \cdots - 2^0$
6	0 1|0 0 0 0 0 0	64	$\chi - 2^5 - \cdots - 2^0$
7	0|0 0 0 0 0 0 0	0	$\chi - 2^6 - \cdots - 2^0$
8	|1 0 0 0 0 0 0 0	128	$\chi + 1$

The frequency of each bit toggling from $1 \rightarrow 0$ halves as each power of 2 is incremented. For a 12-bit integer, there are 2048 even numbers with a zero in the first LSB, since the first bit toggles high and low for each incremental clock pulse. Likewise, there are 1024 odd numbers with a zero in the second LSB as the second bit toggles high for two clock pulses and then low for two clock pulses. Similarly, the third bit toggles high for four clock pulses and then low for four clock pulses, providing 512 odd numbers with a zero in the fourth LSB location, and so on. For a 12-bit system, we can calculate an upper limit of 12260 hybrid values, which is relatively small and easily stored in a look up table. Generally, let β be the number of bits available on the binary counter being used, then the number of possible hybrid values can be stated as

$$\begin{aligned}&(2^{\beta-1}\times 2)+(2^{\beta-2}\times 3)+\cdots+(2^{0}\times(\beta+1))\\&=\sum_{i=1}^{\beta}2^{\beta-i}\times(i+1)\end{aligned}\tag{4.1}$$

Eq. 4.1 provides the maximum search space dependant on the number of bits available on the counter. A 16-bit counter would have a possible 50 million conceivable hybrid values, but a 32-bit counter would jump over 12 billion. Thus, even though the algorithm is scalable, without extra constraints placed on the possible values the practicality of using the algorithm beyond a 24-bit counter diminishes.

To investigate this effect, using the method described in Figure 4.3, a lookup table is constructed given all the possible values of χ', for a given χ,$\chi+1$ pair. Each integer value of the counter is checked to estimate the time required to toggle necessary bits from $\chi \rightarrow (\chi+1)$ (settling time). If the settling time is greater than half of the period of the oscillator, including the effect of the delays, then the location of the bit-wise collision is estimated and a hybrid value is calculated. Modelling of data is then performed by stepping through the expected values determined by the number of oscillator pulses per TI and looking up the corresponding hybrid values from the table.

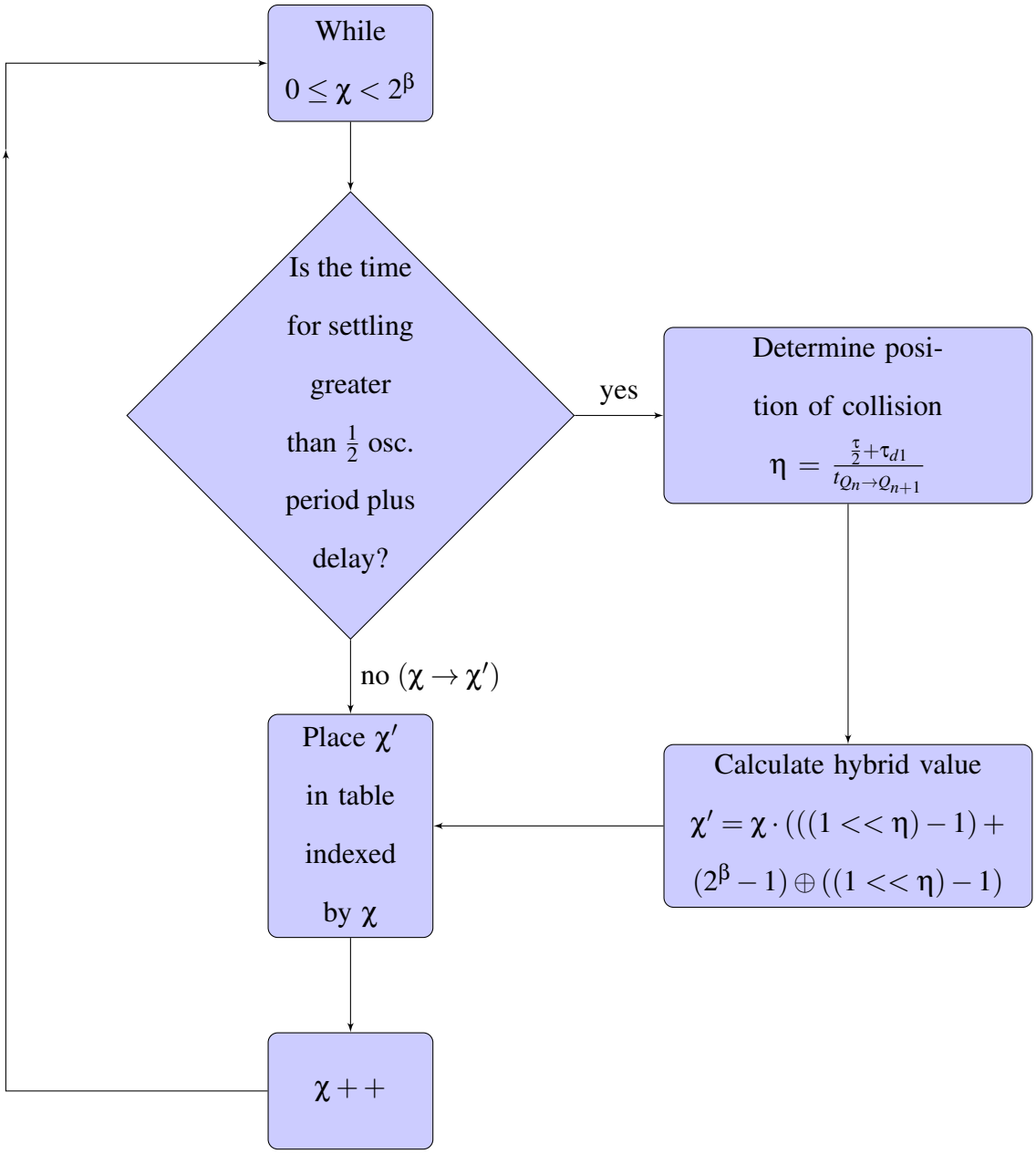


Figure 4.3: The method used to simulate data with bit-wise collisions.The average time for the binary counter to settle is calculated and compared to the half period of the clock and the imposed delay. If it is greater, the position of the collision is calculated and the value of χ' is assigned the post-collision value.

To reverse the process, another lookup table was generated, using an algorithm similar to the one demonstrated in Figure 4.3. The fix table was constructed using the hybrid value as the index to a list of the $\chi+1$ values that result in a bit-wise collision returning the hybrid value.

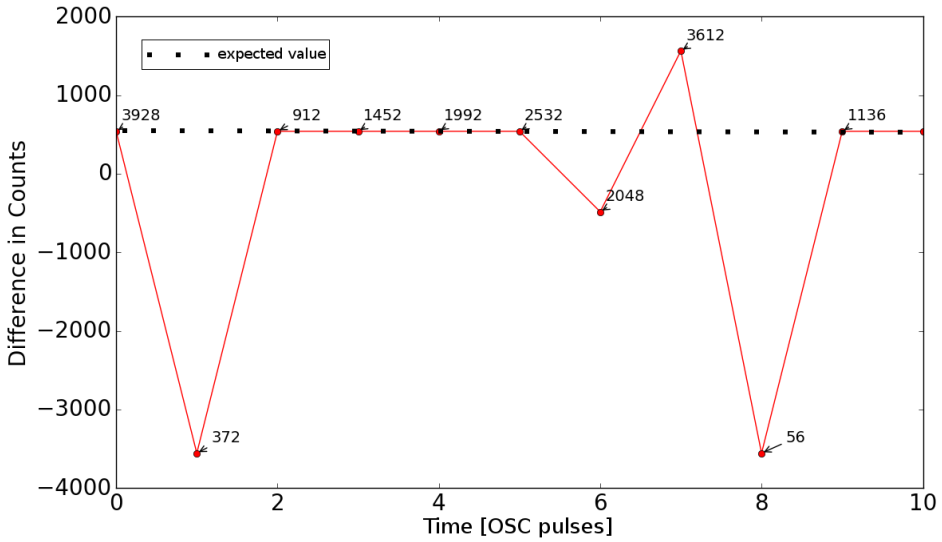


Figure 4.4: Differential count values for a data sample from the prototype circuit with a DRT occurring at 25 kHz. The 13.5 MHz oscillator divided by 25 kHz results in the expected number of pulses between samples to be 540 counts, for this example. The value of the 12-bit ripple counter labels each data point and the difference with the preceding point determines the vertical axis value.

In Figure 4.4 we see the relationship of the recorded counter value and the difference between each point with the preceding value. The obvious 12-bit overflow occurrences happen between the zeroth and first point ($3928 \rightarrow 372$) and the eight and ninth ($3612 \rightarrow 56$). If we assumed that any point lower in value than the point before it indicated an overflow, the seventh point (2048) would not be correctable as $(-484+4096) \mod 4096 = 3612$. Since the approximate expected change in counts is known, in this case, to be 540 counts, and the possible values, which, due to bit collision (derived similarly as in table 4.3, but for twelve bits) lead to 2048 are : 2047, 2048, 2049, 2051, 2055, 2063, 2079, 2111, 2175, 2303, 2559, 3071, and 4095. The value of 3071 provides the closest to the expected value, implying that the value was recorded during the period the counting IC was changing from $3070 \rightarrow 3071$. In order for a hybrid value to occur, the oscillator pulse to increment the counter from $\chi \rightarrow (\chi+1)$ must have occurred, therefore the best choice for correction is $\chi+1=3071$.

For a regularly sampled large data set, it is assumed that the median difference between

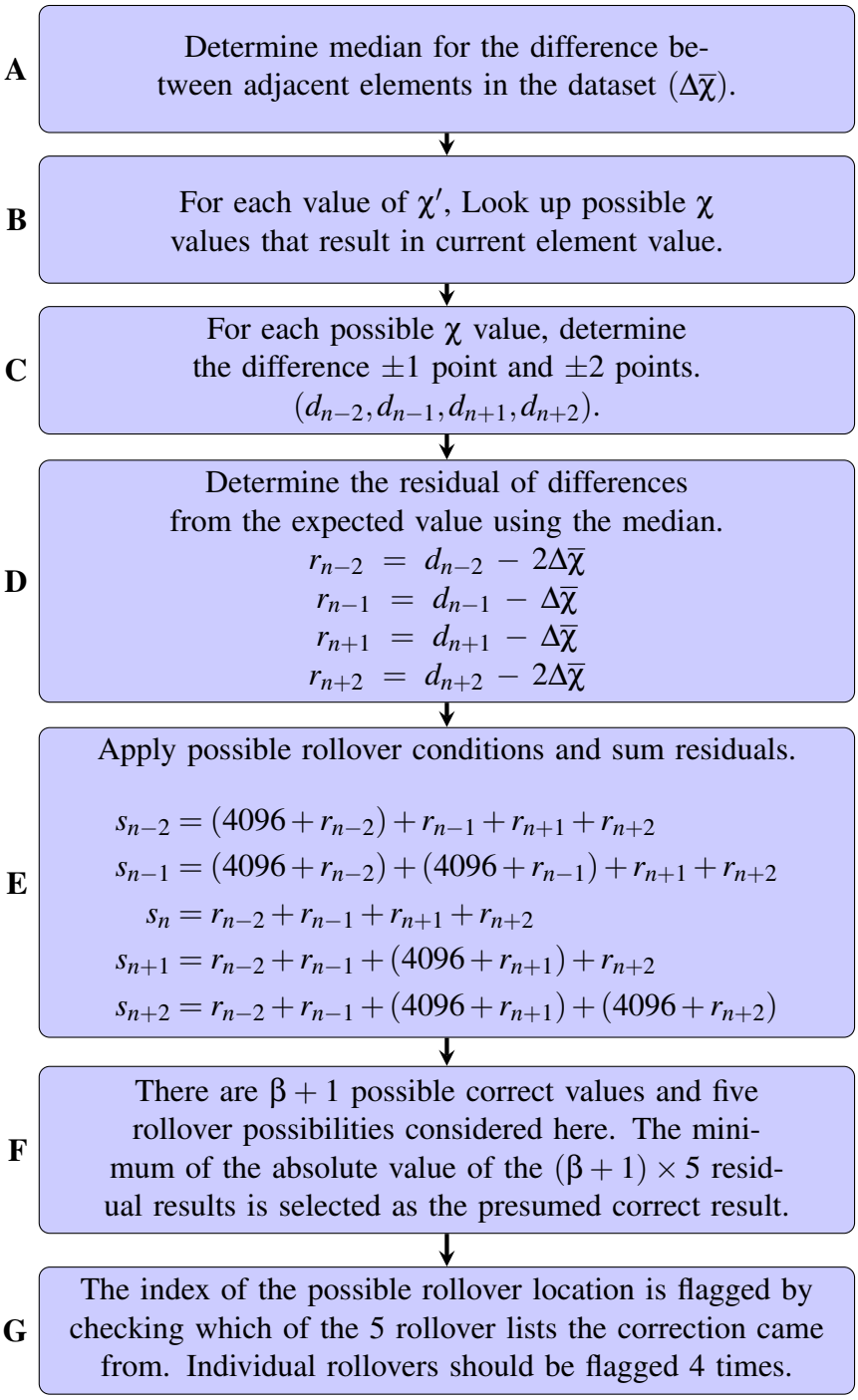


Figure 4.5: The method used to correct data examines the difference between the preceding and following two elements, allowing the identification of hybrid values which restores evenly spaced time intervals. The method also ranks the position of counter rollovers between 0 and 4, applying a correction for the event of passing the highest possible value and starting counting over at zero.

counts between each point should be the same for data sets that may or may not be affected by collisions, since the occurrence of collisions is a relatively small percentage of the total elements. For each point in the data set, the differences between the two preceding and the two subsequent data points are calculated for each candidate on the list of possible fixes, as depicted in Figure 4.5 box C. The residual from the expected value is found by subtracting one or two times the median difference as described in box D of Figure 4.5.

Also in need of consideration in this application is the rollover of the counter from $(2^n - 1) \rightarrow$ zero. There are four positions that a rollover could occur when using the two points before and after the sample in question, and there is the possibility that there is no rollover present (described in block E of Figure 4.5). Assuming the rollover occurred between the second and first preceding points, 2^{12} must be added to the S_{n-2} sum. Similarly, for a rollover occurring between the preceding and current point, 2^{13} (2^{12} to both preceding elements) is added to the S_{n-1} sum. Rollovers that take place between the point of interest and the succeeding point, require a 2^{13} increment to the S_{n+1} sum. Finally, a 2^{12} increase in sum S_{n+2} compensates for a rollover taking place between first and second succeeding points. Nothing is added to the S_n sum, to account for no overflow occurring during the set of elements under search. In the case of applying a rollover where none is present, the absolute residual sum of the list is increased. Not applying a rollover when one did occur, results in an increased residual absolute value. Out of all possible results, the absolute residual that is closest to zero is presumed to be the correct fix table value.

After constructing the circuit described in Figure 2.5, the arbitrary wave generator provided uniform square pulses at a variety of trial frequencies. Depending on the frequency of the oscillator, the required time delays τ_{d1} and τ_{d2} were configured using discrete components. The rising edge of the pulse acted as the DRT and a TEENSY recorded the counter values. The results with no correction were compared to the modelled results. Negative time differences were not possible, and were an indication of a rollover.

Since bit-wise collisions normally result in a smaller value for χ'(except for when $\eta = 0$

or $\eta = 12$), a smaller difference in time was expected. When comparing adjacent samples, where one has a large value of η, it is possible to get a negative difference. To keep this from being confused with overflow conditions, each result is checked for collision first.

4.4 Results For Asynchronous Ripple Counter

Multiple sampling frequencies, between 10 kHz and 60 kHz, were tested at 1kHz intervals. The results provided represent a sample from different segments of the test data. There is little difference between 10kHz and 15kHz, but features can be seen developing as the sample frequency increases. The correction algorithm was not shown applied to the modelled data, in Figure 4.6, as it is the inverse of the generating function and returns the domain as expected.

In Figure 4.6, the two additional prominent spikes in both the simulated and experimental data are the result of bit-wise collisions. The 10 kHz sample data displays the expected bar on the lower side and the complementary bar on the upper side of the median count value. The bars appear on the same side of the average TI for the higher experimental frequencies, as the lower differential time value becomes a negative value, and after correction to a positive integer value it was moved to the far right side.

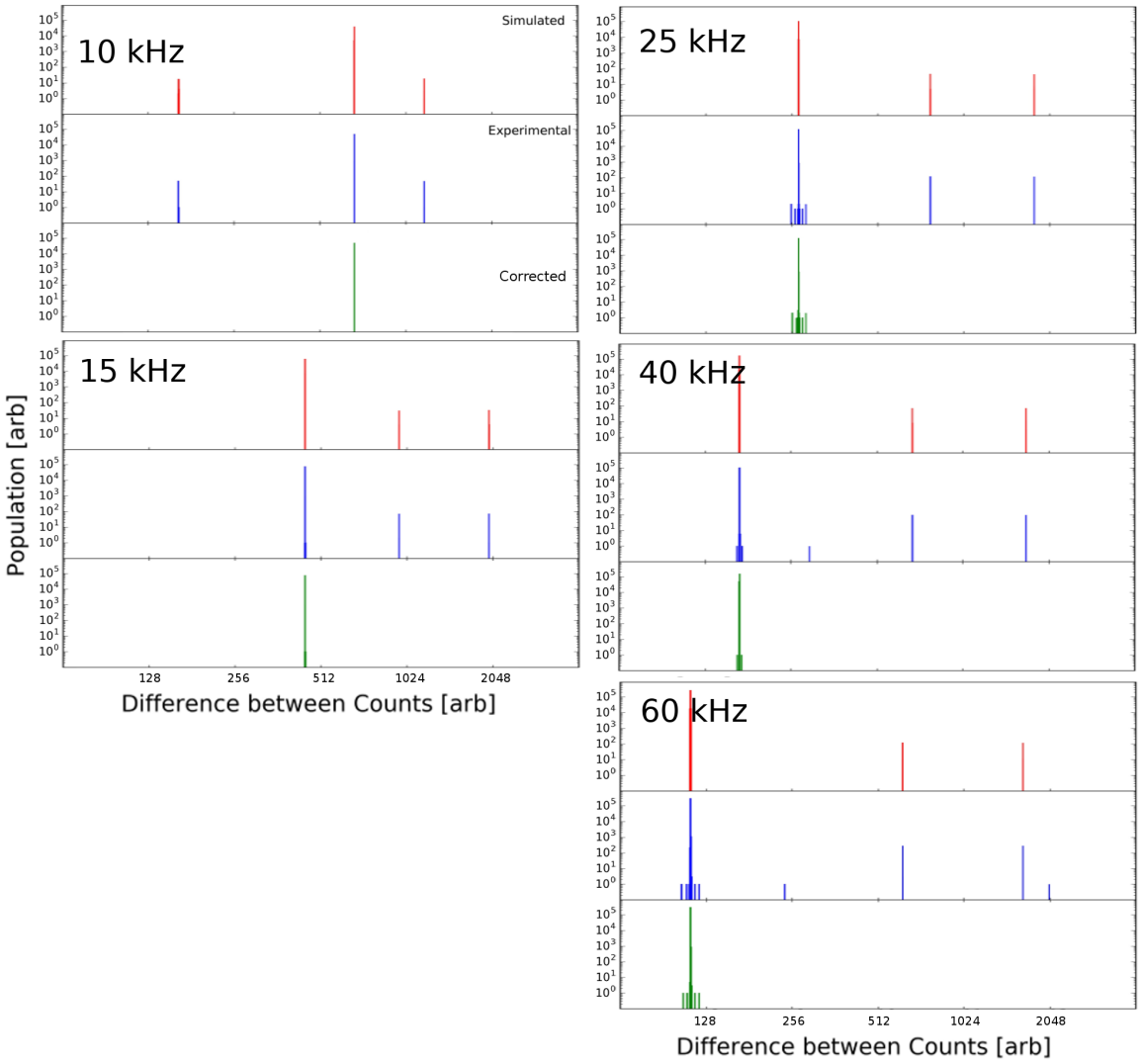


Figure 4.6: Measurement of time intervals within each sub-figure presenting data for a given data rate, the simulated data (top) accurately estimates the number of collisions seen in the experimental data (middle). The corrected experimental data (bottom) still displays errors determined to be caused by a different source. In the 40 kHz data set, the time interval that exists at approximately 256 counts should have a mate close to the 4095 mark (lost due to scaling), which represents a single collision at a different bit level. Similarly, the 60 kHz has a single collision just below the 256 counts mark and its match just above 2048. Note the $\log_2$ x-abscissa reflects the highest bit level of the sampled data, which causes the last sixth of the plot to represent the equivalent area of the first five-sixths.

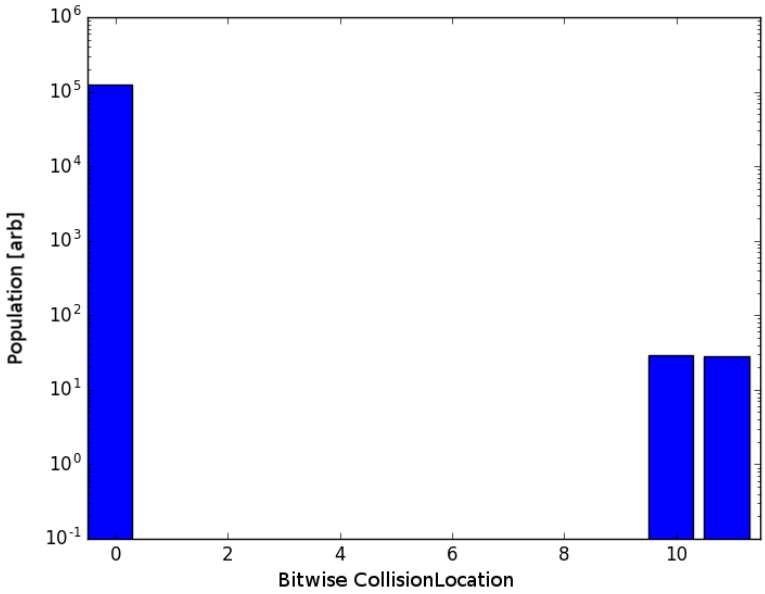


Figure 4.7: 25 kHz experimental data, created while using a function generator as DRT, demonstrates that the position of the bit-wise collisions have been tuned to the tenth and eleventh bits.

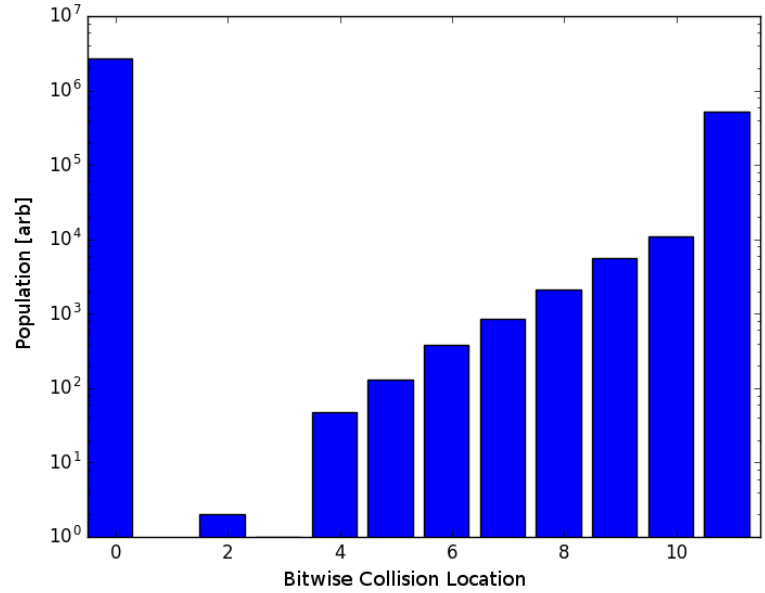


Figure 4.8: Aerotech linear stage experimental data, created while using the position synchronized output as DRT. These results demonstrate that the position of the bit-wise collisions vary greatly but the tuning moved the majority to the eleventh bit. Collisions occurring on the first bit were ignored due to the level of noise on the system.

Figures 4.7 and 4.8 were created by tracking the location of the bit-wise correction (η) applied to each of the experimental systems. The precise periodicity of the function generator is illustrated in Figure 4.7, as the tuned prototype circuit consistently places bit-wise collisions on the tenth and eleventh bits at the expected relative proportions. The variance in the motion of the Aerotech stage introduced a variance in the period of the DRT, which manifests in the bit-wise collisions being more dispersed in Figure 4.8.

Data collected in Figure 4.9, profiling the velocity of the Aerotech linear stage [48], were collected by using the prototype circuit to timestamp PSO pulses and a Renishaw laser interferometer, a commercial position metrology system, provided a second external metrology system to verify the data. The PSO pulse was generated whenever the linear stage moved 15.63 nanometres and the rising edge of this pulse acted as the DRT. The velocity of the stage was then calculated using the five-point derivative of the recorded timestamps. The timestamp data was then processed with the correction algorithm and the velocity was recalculated.

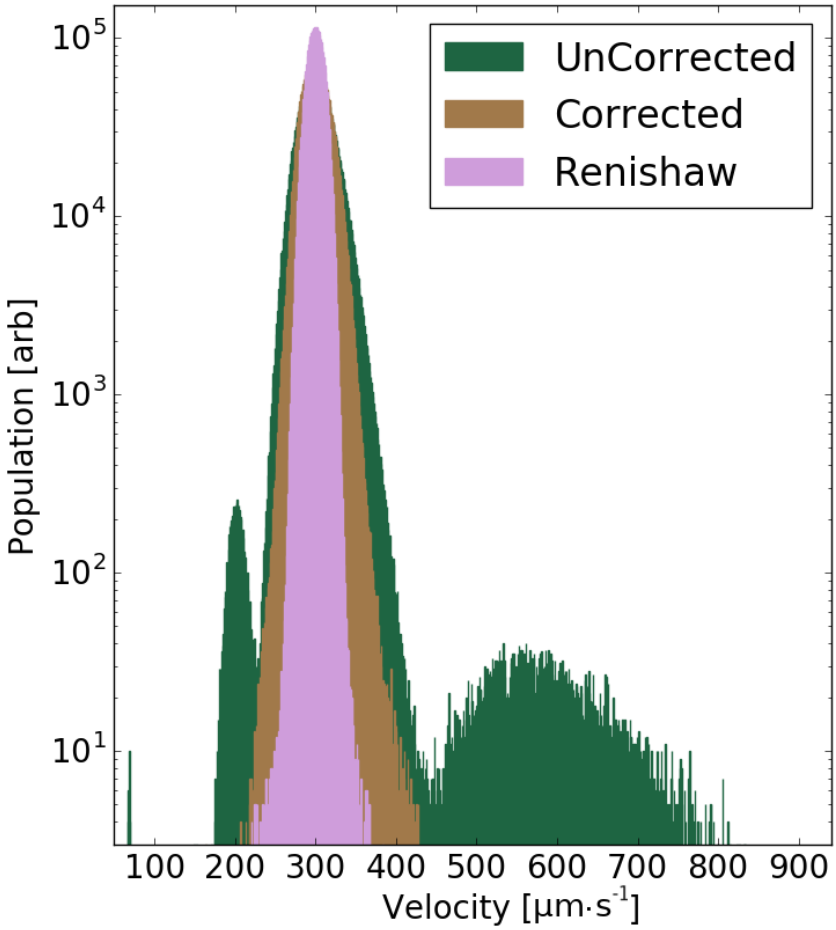


Figure 4.9: The corrected data set displays less variance in the measured velocity of the linear stage. The Aerotech linear stage was commanded to travel at 300 μms^{-1} and the position was measured by position synchronized output pulses and the Renishaw interferometer over a 50mm travel distance.

The Renishaw interferometer, with an accuracy of tens of nanometres through the air, was used as a second verification of the position of the Aerotech stage. Since the original Renishaw sampling rate of 8192 Hz (before the redesign introduced in chapter 5) was about 2.5× less than the sampling frequency of the PSO, the Renishaw data were interpolated onto the same sampling grid for comparison. However, this results in an artificially smoother data set. This was demonstrated by interpolating the 25 kHz data onto the slower timescale, which also removed the features related to the bit-wise collisions and the correc-

tion algorithm. Therefore the data in Figure 4.9 are presented at the 25 kHz resolution.

The obvious solution for bit collisions is to either use a synchronous clock, where all the bits settle at the same time, or put precautions in place to ensure that the latch is asynchronous to the settling of the binary counter. Another possible solution was to design a Gray code circuit, such that only one bit ever changes as the counter increments. Given that this feature was observed, this section explores correction in post-processing, allowing this hardware to be driven at a faster rate. It has been shown that it is possible to correct TI measurements within ± 1 counts by using this post-processing technique, allowing the use of common binary ripple counters to be driven at higher data rates. Data were modelled and collected at frequencies between 5 and 60 kHz, where the modelled data matched the observed for more than 99% of the samples. Experimental data shows a small percentage of points where the TI deviates from true periodicity causing deviations from simulations.

The anomalous data points, that appear in all samples except the lower frequencies, are attributed to a different problem directly related to the circuit design. The lower frequency data appear immune to this issue as fewer samples mean the statistical odds of seeing the problem diminish. The problem appears to be a result of a counter pin transitioning from 1 to 0, and whether due to extra capacitance or poor grounding of the breadboard and wiring, the pin remains high during the latching process. This appears to be an issue with the lower pins exclusively, since the process of binary counting causes least significant bits to toggle at a higher frequency than the most significant bits, reducing the amount time available to discharge the pin. Although not addressed by the correction algorithm, it would be possible to find data points where a single bit transitioning $1 \rightarrow 0$ would correct the data point, but this may be difficult to justify in a physical system. In Figure 4.6, the number of stuck bits that could not be corrected for the 25 kHz sample was three points, with an additional three complementary points appearing incorrect.

Using a post-processing technique allowed for the correction of an over-driven binary counting IC. The method is scalable allowing for larger bit numbers and faster oscillator

frequencies. For convenience, a bit size should be selected such that the number of counts in a single TI can be achieved without rolling over the counter. In the case of multiple rollovers and large TI, the number of rollovers could be counted on a separate counter, in effect increasing the bit size of the counting IC.

This method also could be used to correct a naive approach, where the DRT itself is used to latch the value instead of the rising clock edge. Even at low frequencies, there would be a chance to latch during the settling time of the counter IC, creating a bit-wise collision. The bit level location of such a collision would be arbitrary in this case, as the tuning, by adding delays in order to force collisions towards the MSB, would not exist.

The algorithm described relies on a stable and uniform TI and will not function as well (or possibly at all) on data sets that are uneven or randomly spaced. A simple solution to this problem would be to use a clock with a more coarse resolution alongside the counter circuit. If TIs have a measurably large discrepancy from the coarse clock estimate then the values could be flagged and corrected to the closest collision value that matches the expected TI.

Using a physical system to verify the correction algorithm required some fine tuning of the procedure. The Aerotech linear stage did not move at a constant velocity, but oscillated about the commanded speed due to the PID controller, which meant that the spacing of TIs was no longer even. To correct for the changes in velocity, the algorithm had to be modified to look at smaller segments of data, where it could be assumed that the median TI would be approximately the same. This rolling method of analyzing the data was applied back onto the function generator data with no change in results, assuring that the algorithm function hadn't changed fundamentally.

Applying the correction algorithm to a physical system, measured with the prototype circuit, demonstrated the utility of the post-processing correction technique. Figure 4.9 illustrates that the mean number of counts per TI are similar between position synchronized pulses and the secondary measurement made by the Renishaw interferometer. The lobes

on either side of the dominant peak, in the experimental data, are a direct result of bit-wise collision as they do not appear in the Renishaw data and the error in counts is too large to be physically real. After the correction algorithm had been applied, the lobes were completely removed and the variance was marginally reduced. A Gaussian distribution around the expected number of counts was expected as the velocity does have some variation from the commanded value, for this configuration.

Although other hardware methods exist to avoid or correct the issue of reading a binary ripple counter before it has time to settle, an algorithm was demonstrated which allows for post-processing and recovery of data from hybridized values caused by bit-wise collisions. The benefit of a finer time resolution provides more accurate measurement of TIs and thus allows any asynchronous system to be overclocked for some applications.

4.5 Maximizing Bandwidth with Synchronized Counter

Since post-processing data is time-consuming, and asynchronous timers are inefficient at high sampling rates it was necessary to change the circuit to a synchronous counter where all of the bits change at the same clock pulse. Due to chaining four 4-bit synchronous counters together to provide a 16-bit unsigned value, the propagation delay was reduced to 15 ns from the 60 ns required by the 12-bit ripple counter. The overall flow of the timing circuit is identical to 2.5, and the only block that was changed was the counter portion of the circuit.

The second prototype circuit, constructed on breadboards as seen in Figure 4.12, was designed with four 4-bit synchronous counters (16-bit integer output), a TEENSY [39], and a 16 MHz oscillator. To verify the increased precision of the synchronous circuit various samples at different frequencies (between 5 and 100 kHz) were analyzed.

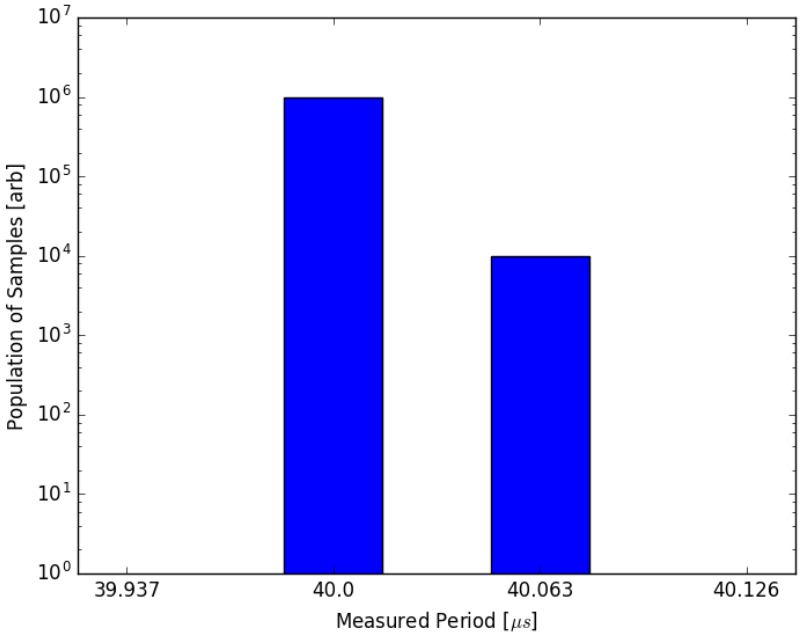


Figure 4.10: A 25 kHz square wave was recorded with the synchronous prototype circuit. The expected period of 40 μs was measured for the majority of the recorded samples, and all other samples were +1 clock pulse (0.063 ns) longer. The error of +1 clock pulse is shared between a slight phase and jitter of the coarse clock oscillator at 16 MHz and the square wave signal from the Agilent function generator.

As the error in the measured period was reduced to a single oscillator pulse, depicted in Figure 4.10, by using synchronous counting components, construction of a fabricated model was fashioned. The complete design incorporated two 16-bit ADCs, besides the control TEENSY and clock circuit. The interaction between components is illustrated in Figure 4.11.

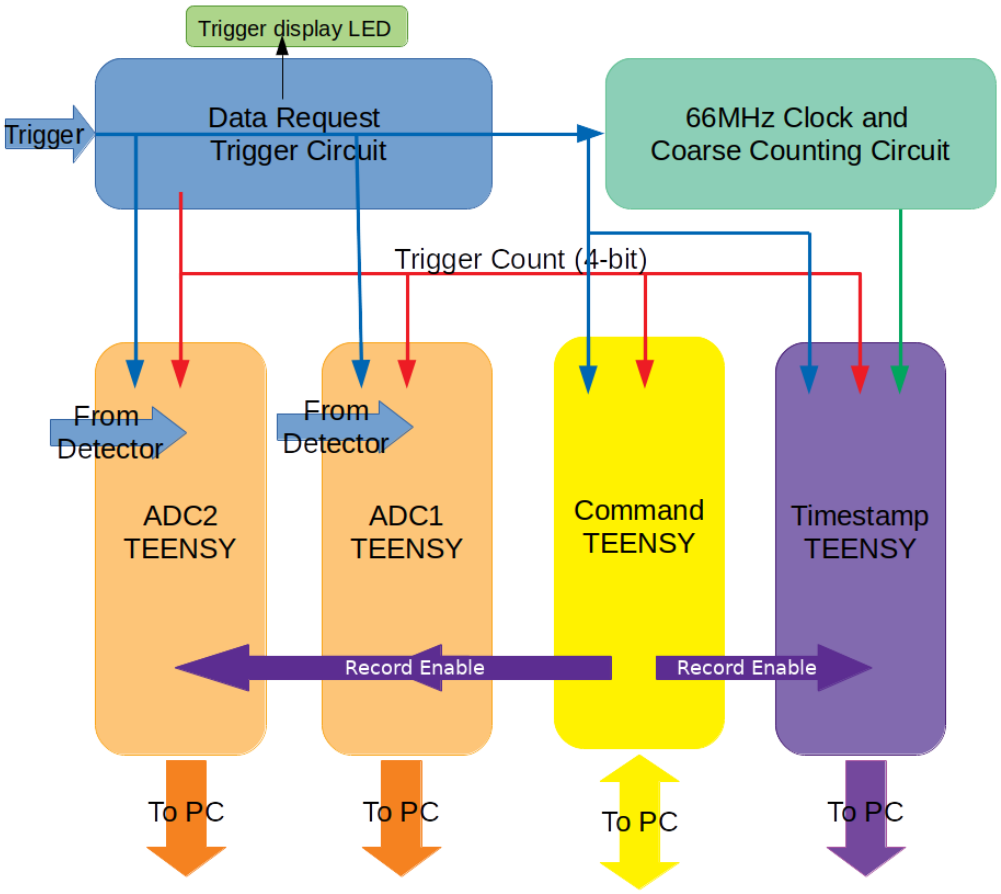


Figure 4.11: The desired components of the completed DAQ included two ADCs, a coarse counting circuit, a data request trigger circuit, an event timer, and a micro-controller to oversee the operation. The "command" TEENSY maintains two-way communication to allow control of the circuit, while the other components only transmit data to the computer. All data is encoded with a 4-bit count to allow synchronization between data streams and account for missing data.

The finished prototype, in Figure 4.12, was tested for operation. The selection of electronics for the new design was made with optimizing bandwidth in mind, with most components capable of operating over 150 MHz.

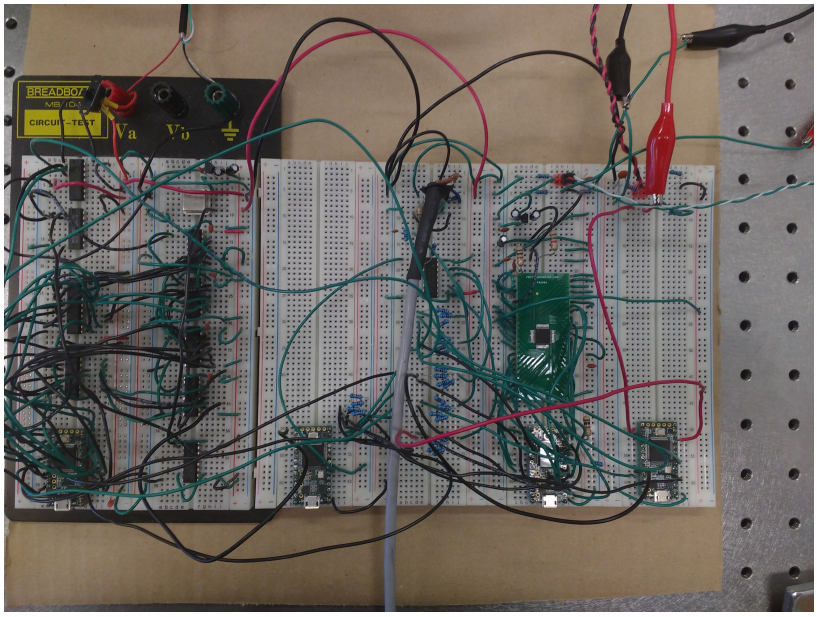


Figure 4.12: The complete discrete DAQ prototype circuit included a 16-bit synchronous counters circuit (far-left), a command TEENSY to control the other circuits (mid-left), a single discrete ADS8371 ADC (mid-right), and a TEENSY utilizing an onboard ADC (far-right).

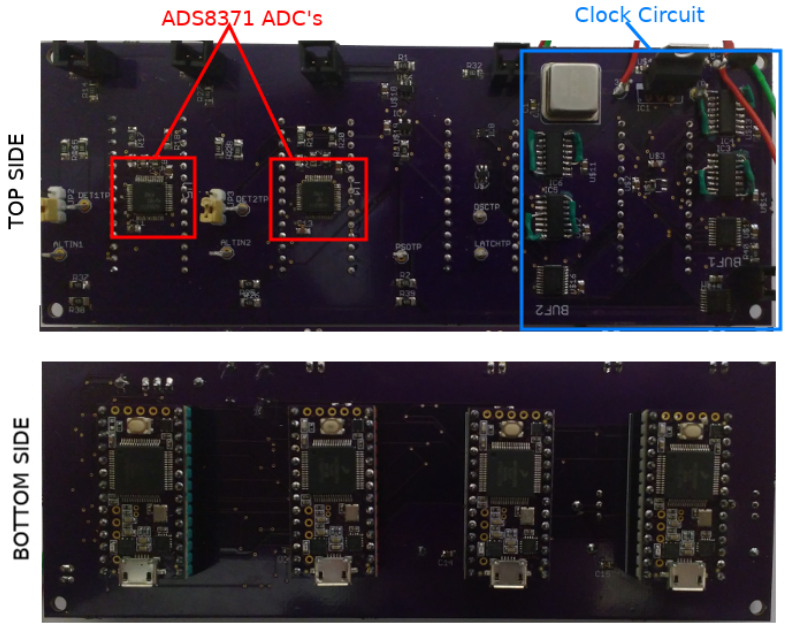


Figure 4.13: The first version of the discrete TEENSYPCB contains two discrete 16-bit ADC chips (red), and a 16-bit clock circuit (blue) with a 66 MHz oscillator. The synchronous counting ICs allow for the faster clock and ± 15.2 ns accuracy in timing intervals. The green wires seen on parts of the clock circuit are corrections necessary due to errors in PCB design. The bottom side of the board contains the four TEENSY micro-controllers.

A printed circuit board (PCB) was designed and fabricated, shown in Figure 4.13, which reduced the complications due to breadboards and multiple wire connections. In order to reduce the footprint, and facilitate routing of traces, the micro-controllers were placed on one side of the board and the corresponding circuitry for the ADCs, PSO input, clocking, and buffers were placed opposite on the other side of the board.

4.5.1 Discrete ADC Module

An external ADC IC was used to increase the performance of the TEENSY. The ADS8371 from Texas Instruments has a parallel output that can be read as one 16-bit value or two 8-bit values in succession at a rate up to 750 kilosamples per second (kSPS).

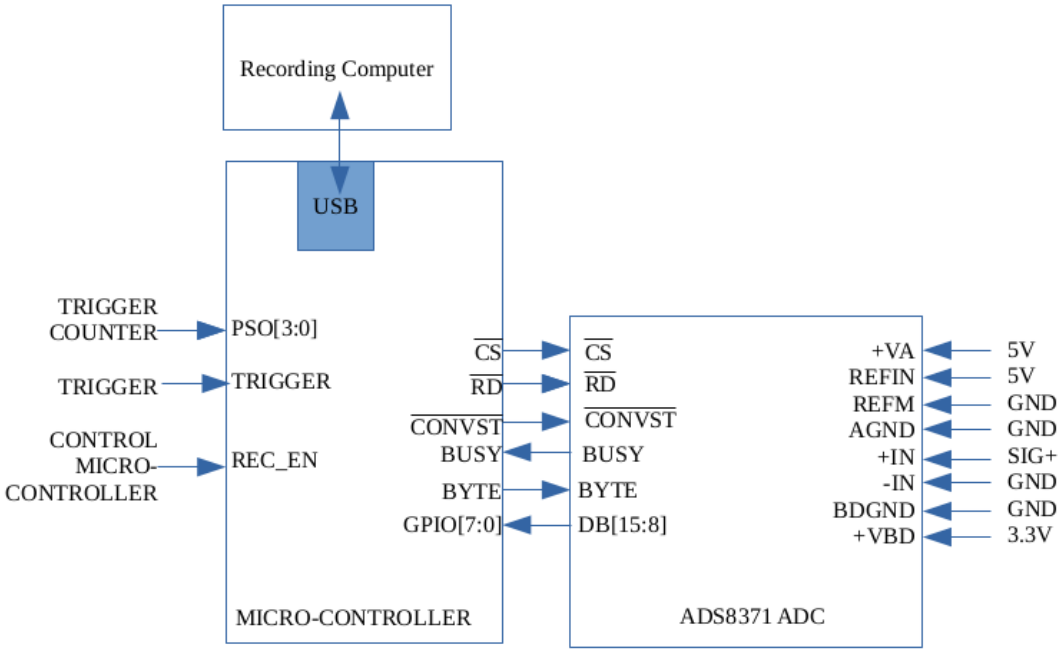


Figure 4.14: The flow diagram for micro-controller to ADS8371 ADC. A single micro-controller performs two read cycles to capture the 16-bit word values representing the signal being converted by the ADC. A four bit counter counts the number of trigger pulses to allow synchronization between this detector, the clock, and the second detector.

The micro-controller sends data consisting of a 4-bit trigger count value and a 16-bit ADC value, for each trigger pulse when the record enable (REC_EN) pin is held high. The GPIO pin selection and data request algorithm are discussed in Section B.6.

4.5.2 The Command TEENSY to Control Data Collection

To minimize the delays caused by system interrupts, the interrupts are disabled during recording on all data collection micro-controllers. This has the side effect of blocking any incoming serial communications. The device still requires a way to start, stop and reset operation, requiring one TEENSY to control the others. Turning off the interrupts does not impede the reading of GPIO pins, so by utilizing a set of output pins on the control TEENSY, which maintains serial communication, overall control is achieved. The Teensyduino code for the command TEENSY is provided in Section B.4.

4.6 Python Library for Teensy-DAQ

A separate library was written in Python, containing a coherent collection of modules required to use the discrete DAQ device. The command list and example routines are available in appendix B.2. Either detector can be utilized, with or without the clock providing time stamps for data. A method to identify components connected to the controlling computer was provided, employing the unique identity of each micro-controller. To increase the recording bandwidth, all data is transferred using an efficient binary file format, universal between all components. The library has been utilized on Windows, and multiple versions of Linux including Linux Mint, Lubuntu, and Raspbian operating systems. Flexibility between operating systems allows transferability between different computers existing in the laboratory.

Identify Routine

The final circuit board contained four micro-controllers, which must all be identified such that the correct data can be labelled and saved. Sending an ASCII character “I” will receive a response of either “COM” or “D #”, where the pound symbol will be 0,1, or 2. The identities are labelled in Figure 4.11. A Python routine polls all the serial ports and determines a list of serial devices that respond with recognized identities. The command

TEENSY and at least one detector (including the event clock) must be present to function.

Binary File Format

Although serial communication is capable of sending straight ASCII text, this method is not an efficient use of bandwidth. For example, a number between zero and 65535 can be represented with just two bytes, but as text would require five bytes up to transfer. For each additional byte, there are also handshaking bits added, as described in Section 3.3. Therefore, a binary encoded value is transmitted for each sample, that has consistent packet structure and requires a single method of decoding. Whether the detector is providing the time or sampled data from the discrete ADCs, the three data streams all output data in the same format. Sent in packets of three bytes, the first byte containing the trigger counting value, and next two bytes provide the 16-bit integer value for the data being recorded.

4.7 Testing and Implementation

An Agilent function generator was set to provide various frequencies of pulses with a 3.3V amplitude and a 5 μs width. The value on the 16-bit counter was recorded and the difference between values furnished the number of 15.15 ± 0.15 ns (66 MHz) [43] periods that elapsed between rising pulse edges. The results are provided in Figure 4.15.

Comparing Figures 4.15 and 3.4, the first difference noticed is the discrete circuit was not recorded at 1 kHz. The electronics introduced a lower bound, where if the period being measured is longer than $2^{16} \times 0.15152\mu s = 993\mu s$ the counter will overflow before the next rising edge. Although it is possible to systematically track overflows, and account for multiple occurrences in a single period, data collection is simplified by having at least one sample occur before the clock rolls over.

The error in the measured period was decreased by 132 times, from the $\pm 2\mu s$ accuracy for the TEENSY alone to the ± 15 ns for the discrete board, where the difference in a measured period was only ± 1 coarse clock period. Reading a 16-bit value from the GPIO

occurs about 6× faster than using the `micros()` command, demonstrated in Section 3.2, which improved efficiency in serial transmission allowing sampling rates over 120 kHz (compared to the $<$ 45 kHz observed in Figure 3.4).

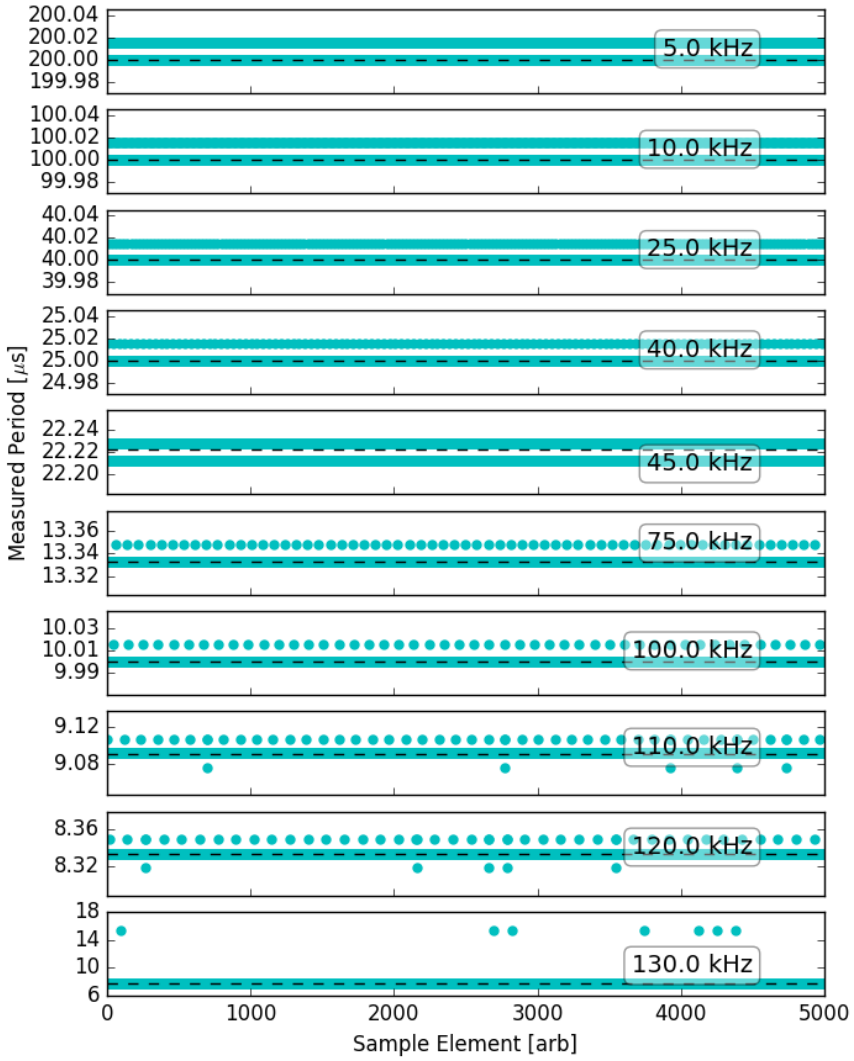


Figure 4.15: A 16-bit counting circuit, incrementing at 66 MHz, was read at even intervals provided by a function generator at various frequencies. The ideal period is marked with the dashed black line. The difference of a single oscillator pulse was 0.015 μs, which is the spacing between digitized bars seen in each sample, except for the 130.0 kHz observation, where the circuit is seen to be on the threshold of missing triggers.

4.7.1 Profile of ADS8371 ADC

Scrutiny of the ADS8371 ADC was performed, similar to the test performed in Section 3.5, to determine the linearity of the ADC response. A 1 Hz saw-tooth wave was sampled, at a rate of 100 kHz, for a period 30 minutes. A subset of the data was selected, as the data set was large, of 8.3 million samples. The saw-tooth wave was set between 0.1 and 3.2 V, covering $> 95\%$ of the full-scale range of the ADC, show in Figure 4.16. A histogram of the data should return a nearly flat line, as each ADC bin should receive a similar number of samples (within quantization noise on the ADC).

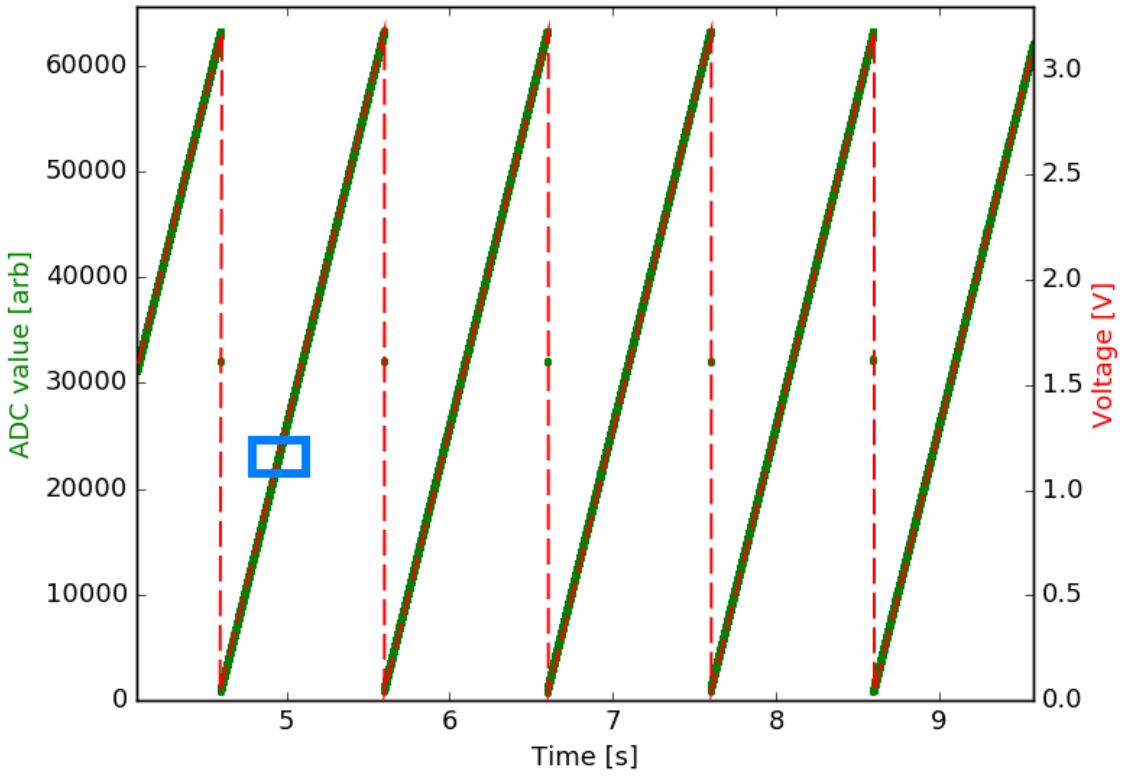


Figure 4.16: An ADS 8371 ADC sampled a 1Hz ramp wave at 95% dynamic range at a rate of 100 kHz. The ideal signal is depicted by the dashed red line. The number of samples between peaks of the saw-tooth wave was determined to be 99704±2467, which places the accuracy of the frequency to 1.00±0.02 Hz. The slope of each individual saw-tooth ramp provides a measure of the change of voltage over the change of time and was measured to be 3.133±0.005 Vs^{-1}. Both these values correlate with the settings and specifications of the function generator.

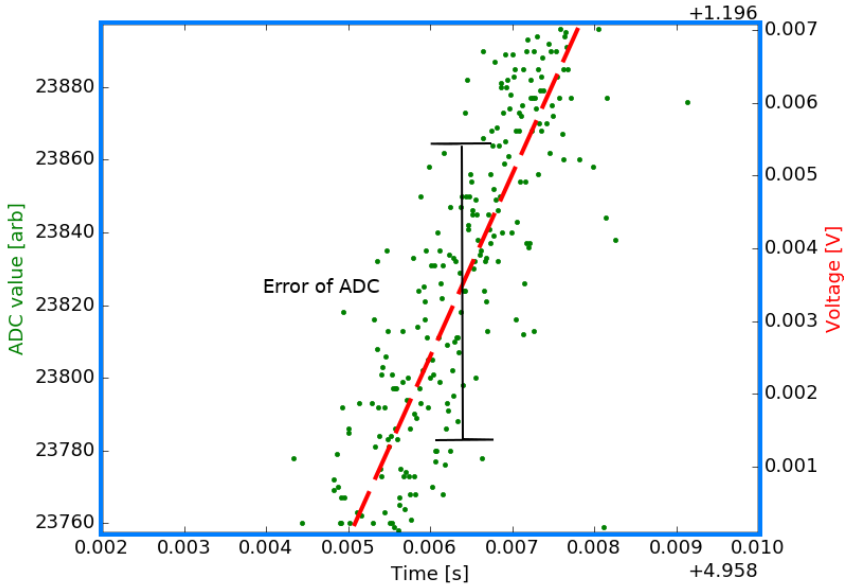


Figure 4.17: Zooming into the blue square of Figure 4.16, the level of noise is depicted by the vertical displacement of samples from the red dashed ideal line. The range of error transverses close to 80 arbitrary ADC integer bins, suggesting the lower 6.3 bits were ineffective. This was confirmed by calculating the SNR of 61.1 dB and ENOB of 9.15 bits.

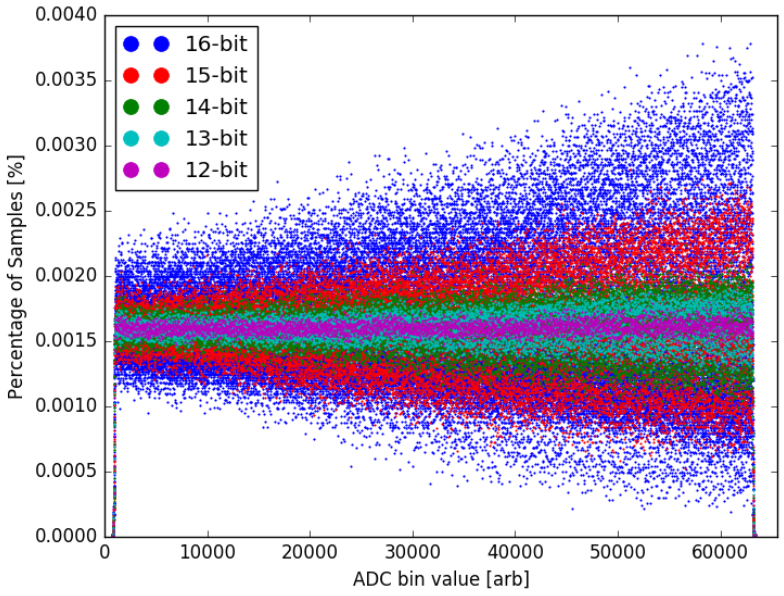


Figure 4.18: The distribution of ADS8371 bin values, when sampling a 95% dynamic range saw tooth wave. Recorded data was reduced in bit-depth and scaled to demonstrate that the linearity of the ADC returns as the bit-depth approaches the effective number of bits. The ENOB, was calculated to be 9.15-bits.

Figure 4.18 demonstrates that the ADS8371 was susceptible to more noise than the previous test established. Reducing the number of bits available in the raw data and scaling the plot to cover the same voltage range, validates the ENOB by returning to a linear response as the resolution of the ADC decreases to the ENOB level. The reduction of bits was performed by bitwise right shifting the integer by four bits (an operation that essentially divides by 16 and truncates any remainder), then multiplying by 16 to scale the data back to the same bin range.

Reducing the bit-depth of the ADC by removing the least significant bits, shown in Figure 4.18, truncates the noisy bits that are unreliable. The response became fairly linear once reduced to 12-bit integers, even though the calculated ENOB was 9.15 bits. A significant reduction in bit-depth is disappointing, but a 10-bit ADC with a reference voltage of 3.3 V is still capable of probing signal to a 3 mV range, which is still useful for the intended purpose.

While attempting to determine the source of the noise, digital low pass filters (part of the Scipy.signal Python library) were used. This was problematic due to the averaging or smoothing that the filter performed, making it impossible to identify a cutoff frequency. The Fourier transform of the filtered data demonstrated an abrupt cutoff of frequencies which caused distortion of the signal. This was realized while looking at the Fourier series for a 1 Hz saw-tooth wave, which is composed of exponentially decreasing harmonics extending to Nyquist frequency. The source of the noise was later determined to be due to a ground fault loop, allowing high-frequency noise (either from the sample request trigger or coarse clock oscillator), producing noise on the analog ADC inputs. This problem is addressed in more detail in Section 6.2.3.

4.8 Conclusions

The prototype circuit provided an understanding of how to latch counting values, without interrupting the process of counting oscillator pulses. While attempting to use the small-

est period possible, to increase the resolution of the coarse counting clock, a situation where data values were captured while the binary ripple counter was still in the process of incrementing a value. The post-processing method of correction was a feasible solution, but in a laboratory setting, where power consumption and equipment are easily obtained, it made more sense to use synchronous counting ICs.

The 66 MHz oscillator improved the resolution of the clock, down to 15 ns, which was two orders of magnitude finer than the TEENSY on its own and five times more precise than the prototype circuit. The resolution of the ADC was not improved upon, but the time to convert a sample was reduced to $< 2\mu$s [41], allowing the discrete TEENSY DAQ to achieve a sampling rate of 120 kHz, compared to the TEENSY onboard ADC which could only maintain 30-40 kHz. The noise level on the ADC circuit became a handicap, reducing the expected resolution, nonetheless the data collected still has an adequate SNR to be useful when measuring in the millivolt range.

Chapter 5 discusses the development of the Renishaw interface using an FPGA, which will lay the groundwork to develop a version of the discrete TEENSY DAQ circuit into an FPGA version in Chapter 6, where the source of ADC noise is also discussed and corrections are proposed to reduce its impact in Section 6.2.3.

Chapter 5

The Renishaw Interferometer Interface

Dreams about the future are always filled with gadgets.

– Neil deGrasse Tyson

During my bachelor's degree, I was part of the development, with Greg Tompkins, of an interface to record data from a commercial interferometer, built by Renishaw [49]. The initial design utilized a single Teensyduino 2.0++ micro-controller (TEENSY2), a slower version of the Teensyduino 3.2 micro-controller (TEENSY), with forty-eight general purpose input/output (GPIO) pins. The serial communication issues lead to the creation of a two-TEENSY2 system, where one micro-controller kept contact with the recording computer and the other sent data. The slower clock rate of the TEENSY2 (16 MHz) limited recording to 8 kHz, which was not an issue since the required data bandwidth was only 10 Hz. The schematic for the original circuit has been included in Appendix C.2 for completeness.

Using the Renishaw interferometer with the Aerotech linear stages required a considerable increase in bandwidth, in order to record position data at the same bandwidth as the position synchronized output (PSO) from the Aerotech Ensemble controller, which pulsed every few nanometres of travel (described in more detail in Chapter 7). An interface to take advantage of the Renishaw interferometers' ability to sample into the millions of samples per second was designed and implemented with the ARTY FPGA.

Section 5.1 presents more detail on the advantages an FPGA provides. Section 5.2 discusses the necessary operations required by the interface between the Renishaw Parallel Interface (RPI20) and controlling computer. Details on the considerations required for the printed circuit board (PCB) are presented in Section 5.3. Finally, Section 5.4 details the procedures taken to verify the operation of the new interface board.

5.1 What is a Field Programmable Gate Array?

A field programmable gate array (FPGA) is a semiconductor device that contains a matrix of configurable logic blocks, whose configuration can be programmed, erased, and re-used. The previous chapters utilized Application Specific Integrated Circuits (ASICs), which are manufactured to perform specific tasks. Components had to be chosen carefully,

with placement being critical, to complete custom circuits. If a mistake was made in the design, repair after fabrication can be time-consuming, if not impossible. The benefits of an FPGA include the ability to reprogram when flaws in the design are found. Requiring fewer components, as most can be emulated by the programmable hardware, reduces the design overhead of any required PCBs. The ability to update features when the design requires new functionality, without the need to change or re-fabricate the circuit board, allows the development of a robust system. The Artix-7 FPGA Development Board (ARTY), illustrated in Figure 5.1, makes the need for a complicated circuit board unnecessary as it already provided slide switches, buttons, display LEDs, and multiple methods of external communication.

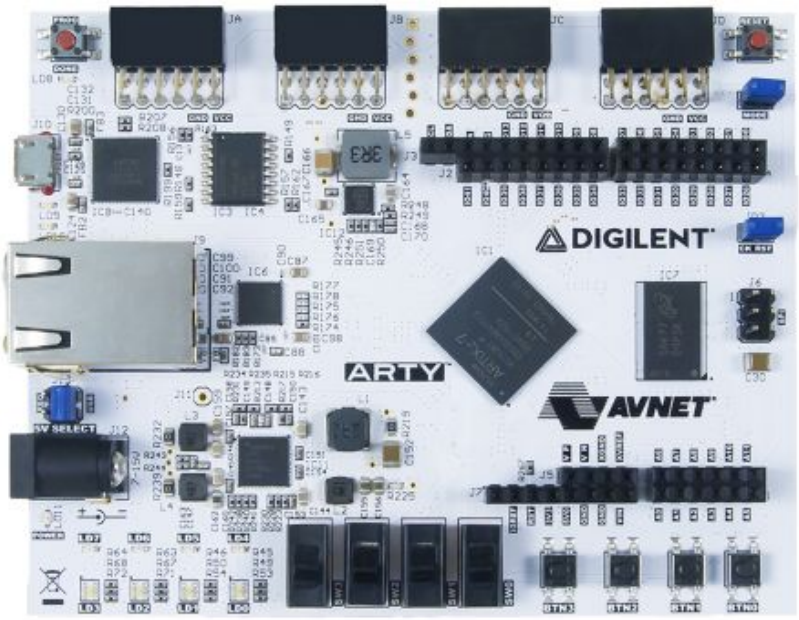


Figure 5.1: The ARTY development board by Digilent is equipped with 29200 flip-flop gates, a clock capable of 450 MHz, 10/100 Mbps Ethernet, 4 switches, 4 buttons, 4 LEDs and 4 RGB LEDs.[50]

Figure 5.2: The RPI20 parallel interface card is connected to the RLE10 laser encoder through the 15 pin din connector located on the front of the card. The rear of the card has a Japanese Aviation Electronics (JAE) 60 pin connector which needs to be connected through the designed shield to the ARTY FPGA.

The advantage of an FPGA over a micro-controller is the ability to parallel process data. This is similar to multi-threading on modern computer processors, a capability that most micro-controllers lack. This allowed the serial communications portion of the circuit to exist independent of the data collection, removing the constraint of serial interrupts affecting the bandwidth of sampling. Originally, two micro-controllers were required to perform the task of interfacing the RPI20 to a computer, in concert with a handful of integrated circuit (IC)s. For the ARTY version, this was accomplished with only five resistors and the appropriate connector. Finally, with no fixed hardware structure, the limitations of 16-bit or 32-bit architectures are removed, and the ability to process odd sized registers coupled with higher frequency clocks made the new design superior.

5.2 Theory of Renishaw Interferometer Operation

The RPI20 returns a 36 bit two's complement value, demonstrated in A.2, which is an integer number representing the distance travelled from the home position in multiples

of the set resolution. As a single axis measurement, the RPI20 can operate up to 4 million samples per second. The designed circuit is required to transmit the data from the RPI20 to a computer via USB or network connection. Further specifications are provided in Appendix C and documentation[51, 49, 50].

5.3 Circuit Design Considerations

The design implemented on the ARTY FPGA used a single main program to control many sub-modules underneath. This was desirable to keep timing synchronized between serial communications, sample clock, UDP transmission, and communication with the RPI20 card.

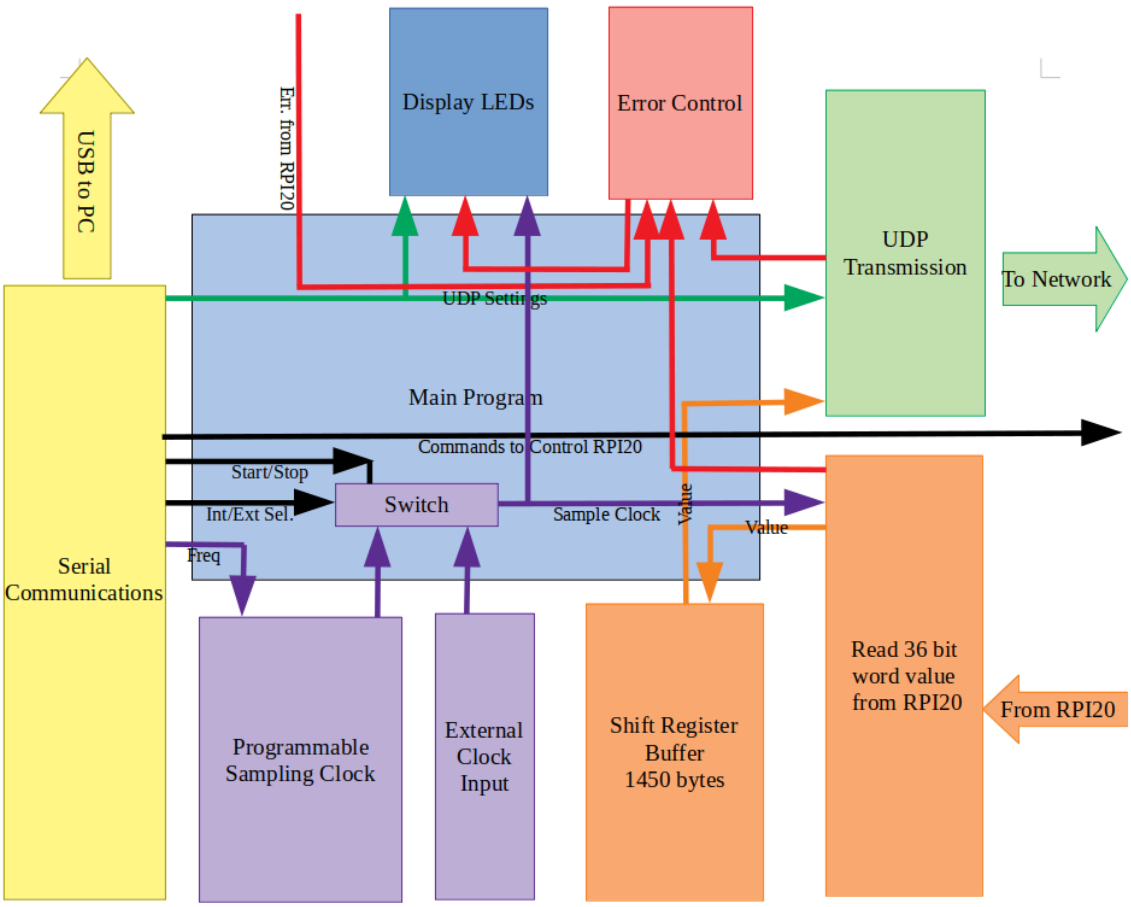


Figure 5.3: A flow diagram of the ARTY programming to interface with the RPI20. All components pass through the main program, which controls, synchronizes and redirects as needed to other components. There is no case where two sub-components speak directly with each other.

The operational design, seen in Figure 5.3, implemented a "Main Program" which controlled all of the sub-components. The serial communications (yellow) provides control and feedback through a USB connection. The sampling clock (purple) is used as a trigger to request data samples. UDP transmission (green) is a set of modules which create the correct Internet Protocol version 4 (IPv4) and UDP headers, check-sum values, and data into a single transmittable packet. The shift-register component (orange) is actually a part of the main program (light blue), but is shown separately as a container for the data read from the RPI20. Error control (red) monitors the error conditions and triggers an appropriate response in the main program and display LEDs (dark blue). The following sections

will discuss each component in further detail.

5.3.1 Sampling Clock

The original circuit used multiple settable synchronous up/down counters to divide a 32 kHz oscillator to control the sampling rate. The master TEENSY2 used sixteen GPIO pins to divide the oscillator frequency. The FPGA allowed for the entire portion of this circuit to be emulated by software.

With a sufficiently high enough oscillator frequency, such that the desired output clock is only 1-2% of the master clock frequency, an accurate sampling clock can be emulated. Although up to 50% of the master clock frequency can be emulated, an increasing relative error occurs for values that are not mathematical factors of the original oscillator rate. The Renishaw internal oscillator was set to 200 MHz, which is reasonable for creating a sampling clock between zero and two MSPS, with minimal jitter.

The method for dividing the internal master clock to a slower clock rate was originally designed for providing a baud rate clock for serial communications, which must have less than 2.5% error in period length[52]. Start with an integer counter, which has a negative equivalent value of the frequency of the master clock. For each rising edge of the master clock, the counter is incremented by a value equivalent to the frequency of the desired output clock. At some point, dependent on the master and output clock frequencies, the counter will pass zero and become positive. At this point the output clock is pulsed, and the counter is decremented by the frequency value of the master clock minus the desired clock. This process repeats indefinitely, creating an output clock with the desired frequency, except for a small error when the desired clock frequency does not divide evenly into the master clock frequency.

```
1  module sample_clock( input wire clk, input reg [31:0]clk_freq, input
       reg[23:0] samp_freq,
2  output wire o_samp_clk);
3  reg [31:0] counter;
4  wire [31:0] counter_inc = counter[29] ? (samp_freq) : (samp_freq -
       clk_freq);
5  always @(posedge clk)
6     begin
7     counter = counter + counter_inc;
8     end
9  assign o_samp_clk = ~counter[29];
10 endmodule
```

Figure 5.4: The Verilog code to generate a sampling clock using the internal oscillator of the ARTY. The rising edge of `clk` increments the `counter` by a set value dependant upon the MSBs greater than the integer value of the oscillator frequency.

Figure 5.4 displays the Verilog code used to generate a sample clock from the FPGA's internal oscillator. The 32-bit register `clk_freq` passes the integer frequency value of the master clock `clk`. Since two hundred million can be represented with only 28-bits, any MSBs greater than the 28^{th} will indicate when the variable `counter` is a negative number, by two's complement constraints described in Section A.2. Thus, the variable, `counter_inc`, which determines the amount to increment the counter, is dependent on the 29^{th} bit state. When the 29^{th} bit is high, `counter_inc` is a value equal to the integer value of the desired sampling clock frequency, (`samp_freq`). For the 29^{th} bit in a low state, when the `counter` is greater than zero, an integer value of (`samp_freq - clk_freq`) is given to `counter_inc`, on line 4. Line 7 increments `counter` by `counter_inc` on the rising edge of `clk`. Finally, line 9 assigns the inverse state of the 29^{th} bit to the output, where a pulse with the same period of `clk` is emitted each time `counter` passes zero. This process is mathematical division, where the remainder is allowed to compound, causing a slight

jitter in the period. For this reason, sample clock rates that evenly divide into the master clock frequency provide the best results, especially when the sample clock frequency is comparable in magnitude to the master clock frequency.

If jitter on the input oscillator is considered negligible, this is a purely binary process that can be simulated and the error for each frequency can be predetermined. A Python version of Figure 5.4 was used to determine the relative error in the output clock period and generate Figure 5.5.

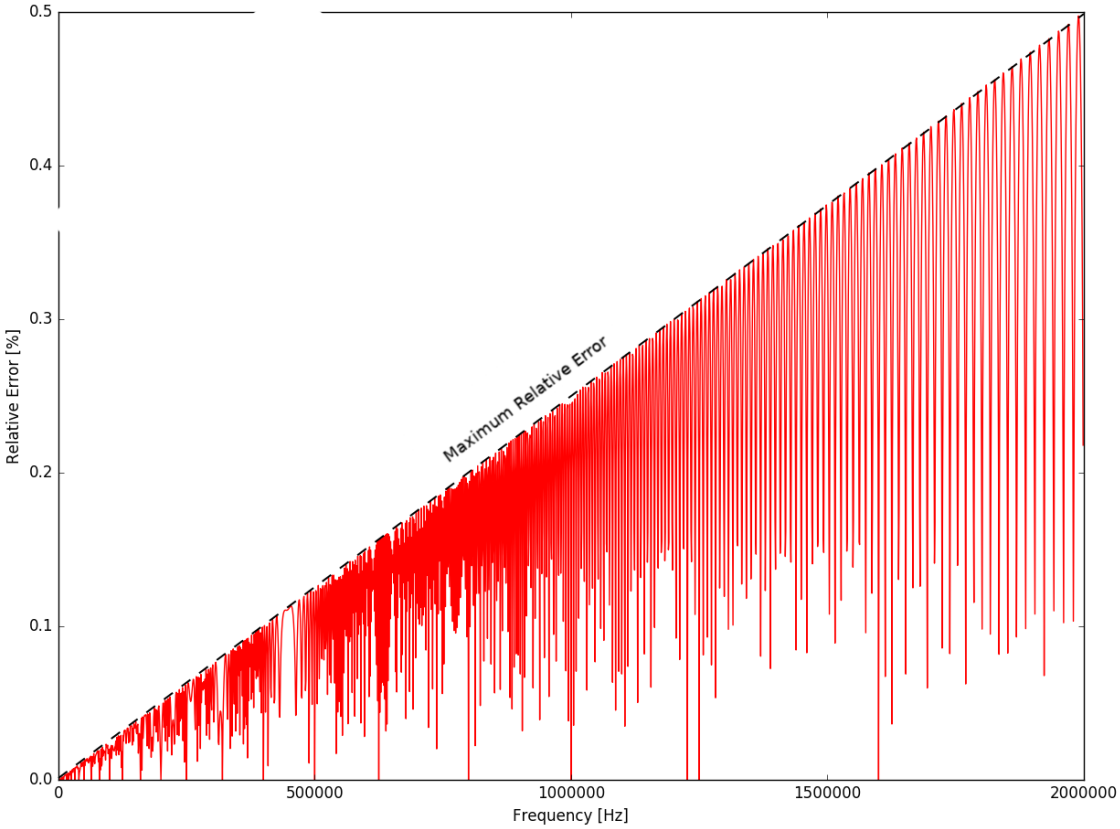


Figure 5.5: Simulated sampling clock collected 1000 samples measuring the number of 200 MHz clock pulses that occur during each output clock pulse. The relative error as a percentage of the desired output clock period demonstrates that for relatively low output frequencies ($< 2\%$ of the master) the output clock can be considered accurate ($< 0.5\%$ rel. err.). Note that output clock frequencies whose periods are multiples of the master clock period have zero relative error.

For Figure 5.5, a pattern can be seen where multiples of the 2.5 ns (200 MHz) master clock period have zero relative error in period. The maximum relative error, which is shown

by the straight line running diagonally through Figure 5.5, is calculated by

$$RelClkErr_{max} = \frac{100}{MasterClockFrequency} \times OutputClockFrequency. \tag{5.1}$$

The actual relative error, that accounts for the fractional ratios of output to master clock ratio, can be calculated by

$$RelClkErr_{act} = \left(1 - \left\lfloor \frac{MasterClockFrequency}{OutputClockFrequency} \right\rfloor \times \frac{OutputClockFrequency}{MasterClockFrequency}\right) \times 100 \tag{5.2}$$

Equation 5.1 is adequate to estimate the relative error when the output clock frequency is only a few percent of the master clock frequency. For output clock rates that approach half of the master clock rate, Equation 5.2 will provide a more accurate estimate of the error expected. Equation 5.2 demonstrates the utility of selecting multiples of the master clock period, as this negates the need for the floor function, resulting in zero relative error. Neither of these equations account for jitter or frequency error existing on the master clock.

5.3.2 Error Control

A re-settable mono-stable multi-vibrator (or one-shot) provided error control. The original two TEENSY2 PCB set the one-shot on the data request trigger (DRT) delivered by the sample clock, and was reset by the micro-controller when data was collected via a GPIO pin. Should another sample be required before the one-shot had been reset the system would flag an error. Errors that occurred due to poor signal strength or a break in the laser beam were delivered directly by the laser encoder and received by monitoring a GPIO pin.

The new system was required to ensure no sample requests had been missed, and was broken down into three different error states in the FPGA. First was to flag an error if a data request occurred while the state machine was in the process of collecting the data, preventing missed sampling requests. The second was if transmitting a new packet of data occurred while a previous packet was already being delivered, the new packet would be

lost, thus raising an error. Finally, the encoder was monitored for interruptions in the laser beam and poor signal errors, in the same fashion as the original circuit, flagging an error state should one occur.

5.3.3 Serial Communications

A serial connection was still required to control the device, but data was no longer sent via this method. Unlike a micro-controller, there were no native serial communications devices built into this FPGA. A pre-written third-party library was used to send or receive a single byte at a time[53], and a module was written to provide an ASCII menu structure with handshaking.

Serial communications with the ASCII menu is performed at a baud rate of 115200 bps, 8 data bits, no parity, no flow control, and 1 stop bit. Digilent Future Technologies Devices International (FTDI) drivers are required by Windows operating systems, but Linux and Raspberry Pi recognize the ARTY serial device automatically.

All serial commands will respond with at least 1-byte confirmation that the command has been received and executed. Generally, an asterisk '*' will be received upon a successful command, a percent sign '%' if recording data was attempted when the device was in an error state, and the capital letter 'E' when commands are not understood. More than 1 byte will be received for commands that also return a form of data. Commands and their appropriate responses can be found in Section C.1.

5.3.4 Shift Register and Reading Values

Obtaining the values from the RPI20 requires a single pin to be held in a low state for a sufficient period of time, while five additional GPIO pins are held with the appropriate address to read the position. Additionally, the address also provided the functionality to get the status of the laser encoder, reset the device, and acquire test information, discussed in Appendix C. The data is then transferred to a shift register. The shift register contained 11600 bits of data, for a total of 1450 bytes. For each byte of data added to the register,

the existing data was pushed eight bits to the left, towards the MSB, and the LSB would be filled with the new data. When 11600 bytes of data was filled, the register was copied to the UDP transmit buffer, and then all bits were filled with zeros. This method was simpler than tracking the number of recorded bytes and filling specific register positions, as it required a single shift (easily programmed) and writing data to the same locations repeatedly.

5.3.5 UDP Packet Transmission

The ARTY includes many peripheral devices, like switches, LED indicators and a network port. The ability to utilize the 100 Mb transfer speed of the network port provided ample bandwidth to deliver data in the regime of mega samples per second (depending on the structure of the data packet). Since the data packet has a predefined structure it is possible to extract a sample of the data at the time of receipt, allowing for semi-live (delays between the taking of data, transmitting and receiving data are not accounted for) monitoring of what is being sampled.

An initial version of UDP packet transmission was written by Mike Field[54], designed for the ARTY FPGA board. The original version had a static payload, one kilobyte of zero values, and hard-coded internet protocol (IP) and media access control (MAC) addresses, and port value. From this example, the code was re-written to include programmable IP and MAC addresses, and a changeable payload.

The incoming data packet, received on a network computer to hard-coded port 4096, contained up to 290 data samples. The first two bytes contain a sequence number which can be checked to maintain packet order and verify that all packets are sequential. Two bytes of zero value are then transmitted, followed by 290, 5-byte sequences. If the data sequence ends before 290 samples have been recorded, the remaining packet is filled with zero-valued bytes to maintain the exact size of the packet (1454 bytes).

5.3.6 Display LEDs

The original board design had a single LED to indicate power was turned on and a single LED to indicate an error condition. The desire was expressed after considerable use, to have a way to visually verify operation. Appendix C describes the use of the ARTY development boards' four green LEDs and four tri-coloured LEDs for the Renishaw interface. Full utilization of all the LEDs provides an indication that MAC and IP addresses, and the sampling frequency are set. The tri-colour LEDs indicate the three error conditions as outlined in the above **Error Control** section, and a sequential touring LED display when the sampling clock is active. Examples and pictures of the display functionality are presented in Appendix C.1.1.

5.3.7 Printed Circuit Board

The ARTY development board included an Arduino/ChipKit "shield" connector, composed of 49 GPIO pins, seen around the edges of Figure 5.6. These provided a method to attach the PCB to the ARTY, as well as functioned in the electronics circuit.

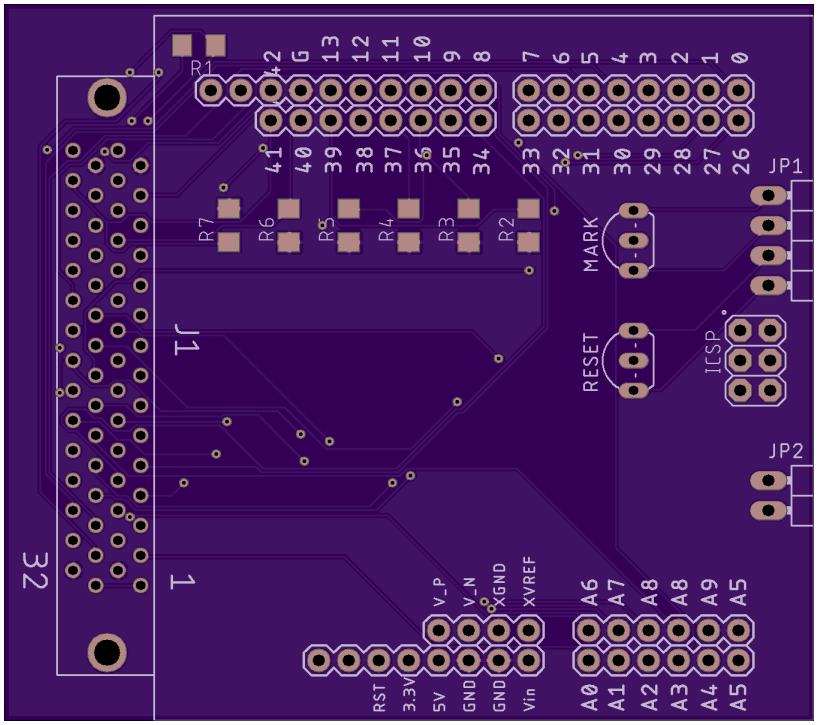


Figure 5.6: The Front side of Renishaw PCB shield for the ARTY FPGA. Components populate only top side of the shield, but the board has traces on the front and back. The large 60 pin connector (J1) provided the attachment to the RPI20 interface card. The ARTY development board had an Arduino Shield compatible set of GPIO pins, which also provided the mount to the FPGA board.

In Figure 5.6, the reduction of the number of components to only seven resistors and two P-channel metal-oxide-semiconductor field-effect transistors (MOSFETs), reduces the complexity of the original circuit depicted in Section C.2. The MOSFETs act as switches for resetting the RPI20 card and Renishaw Laser Encoder (RLE10) encoder. It should be noted the reset function also zeros or homes the position. The resistors are required to pull-down the address GPIO pins, providing more reliable communications between the shield and the RPI20 card.

5.4 Implementation and Testing

As the Renishaw interferometer is the most precise position measurement tool in the laboratory, the only available method to verify its precision was by comparing fringe data from a Michelson interferometer, discussed further in Chapter 7. Keeping this in mind,

a direct measurement of the Renishaw interferometer error cannot be obtained without an instrument which has finer resolution, but sources of error have been identified and can be quantified. Verifying that all the data has been received, by checking the sequence of packet numbers, allowed for the timing to be adjusted for any missing packets. Trying to ascertain the positional error of a moving object is difficult, especially when the deviation from linear motion is the object of study, but a stationary object removes the need to consider this error. A series of samples taken from when the linear stage was still was used to estimate the error introduced by humidity and air currents in the laboratory. An estimate of cosine error, defined in Section A.2.1, can be achieved by scaling the data such that the error is within the absolute error of the linear stage. Finally, the fastest sampling rate that did not trigger an error was obtained.

5.4.1 Verification of Packet Delivery

UDP packets are not guaranteed to arrive in order, and as a connection-less communications protocol, packets may be lost. The first two bytes of each packet are numbered by the FPGA in sequential order. This makes looking for errors easy by examining the packet numbers for a difference greater than one. As all testing has been done on a closed network, there has been no loss of packets recorded, but this check should always be performed.

It should be noted that the packet number does not have to start at zero, and may be any 16-bit number. At some point, the packet value would overflow during recording, causing the packet value to start at zero again. This wouldn't indicate a lost packet but would no longer have a difference of one, and should be considered when examining for lost or miss-ordered packets.

5.4.2 Error in Position of a Stationary Object

For this test, the linear stage was disabled and mounted on a vibration dampened optical test bench. These two precautions prevented motion due to the stepper motors holding the position, and vibration from disturbances in the lab. Jarring motions, such as hitting or

bumping the table would have resulted in large displacements of the linear stage, greater than the few nanometers observed. Taking samples from different trials, when there was no motion of the linear stage, the error in the absolute position was determined by taking the standard deviation of the samples. A series of data were taken while testing the maximum frequency that sampling could occur. This provided millions of measurements of the same position at various sampling rates. Sampling was performed for various durations for 15, 25, 200, 210, 220, 230, 240 and 250 kHz sampling rates. For each data set, the average position was adjusted to zero and the standard deviation was calculated, as depicted in Figure 5.7.

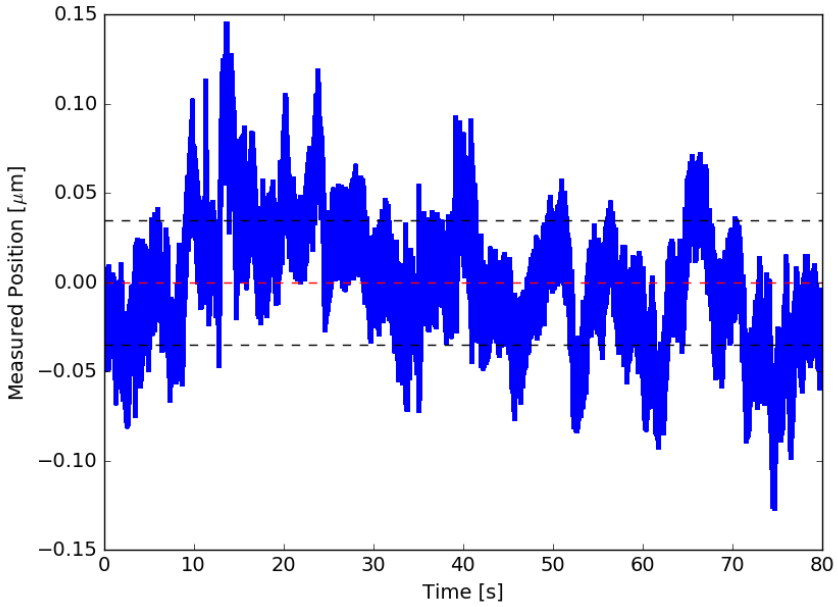


Figure 5.7: An illustration of the noise observed by the Renishaw interferometer, while measuring the position of a stationary linear stage. The above data set was recorded at a 250 kHz sampling rate.

Using an integration-time weighted average, so those data sets with shorter sampling durations have less influence, the average standard error in the Renishaw position measurements of a stationary stage was 35 ± 4 nm. The un-weighted average was 35 ± 8 nm. Figure 5.7 was observed for the longest duration of over 80 seconds. It should be noted that temperature, air pressure, humidity, traffic in the lab, and other environmental considera-

tions are expected to have an effect on this measurement, so these uncertainties should be regarded as upper limits. All the measurements were taken sequentially, during the same period, thus any environmental concerns were subjected to all samples.

5.4.3 Cosine Error Estimation

The only method available, with current test data, to estimate the cosine error, discussed in Appendix A.2.1, was to compare multiple trials of the same commanded displacement of the linear stage to the measured distance travelled. This must be restricted to trials taken consecutively, as alignment may have changed between periods of use.

More detail on the positioning accuracy of the Aerotech linear stages is provided in Chapter 7, which for now is accepted at the given value of 1 μm relative positional error for any commanded displacement [48]. Therefore, it is reasonable to assume that 100 000 μm travel would result in 100 000$\pm$1 μm actual displacement, and any values much less than this is a direct result of cosine or a systematic error. A series of motions were observed, consisting of five sinusoidal oscillations (1 mm amplitude peak to peak and 0.25 Hz frequency), 100 mm travel, another set of oscillations and then 100 mm travel in the opposite direction. The positions where the stage was stationary were identified by finding when the numerical derivative equalled zero. Then the displacement for the 100 mm travel section was ascertained, by taking the difference between stationary positions on either end, depicted in Figure 5.8. Fluctuations, while the linear stage was stationary, were ignored, as the stationary error was five orders of magnitude smaller than the linear travel displacement, and was considered negligible.

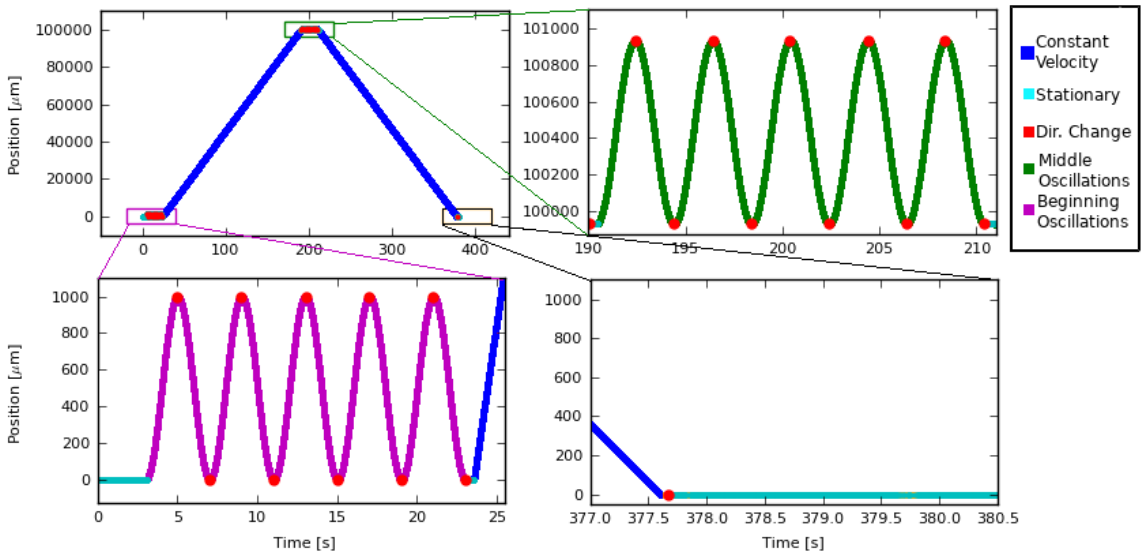


Figure 5.8: The motion of the Aerotech linear stage observed by the Renishaw interferometer. The full range of motion is depicted in the top left, where the 100 mm displacement constant velocity sections of motion are illustrated in dark blue. Highlighted in magenta (bottom left), the first set of 1 mm peak to peak oscillations at 0.25 Hz, used for calibration and timing. The green zoomed-in portion (top right) depicts the second set of oscillations that occurred after 100 mm of displacement. The bottom right zoomed-in figure shows the linear translation stage finally coming to rest.

An example of the observed stage motion for one trial with the linear stage travelling at $500\,\mu m \cdot s^{-1}$, is shown in Figure 5.8. Measurements were recorded at 5 different velocities (300-700 $\mu m \cdot s^{-1}$), each repeating the same commanded motion. From each trial, measurements of the 100 mm displacements (one moving forward and one in reverse) were made using the Renishaw and Michelson interferometers, sampled at a rate of 50 kHz (results presented in Table 5.1). Points of directional change, red markers in Figure 5.8, were determined by finding when the magnitude of velocity was virtually zero. The difference in position between points of directional change was accepted as the measurement of the displacement observed.

Table 5.1: Comparison of linear translation stage displacement measured by the Renishaw Interferometer and the HeNe laser coupled Michelson Interferometer fringe count. Results for trials of 100 mm stage travel conducted at 5 independent velocities for both forward and reverse motion are provided.

Velocity $[\mu m \cdot s^{-1}]$	Commanded linear stage travel $\pm 1\,[\mu m]$	Renishaw Forward Displacement $\pm 3\,[\mu m]$	Renishaw Reverse Displacement $\pm 3\,[\mu m]$	Fringe Count Forward Displacement $\pm 7\,[\mu m]$	Fringe Count Reverse Displacement $\pm 7\,[\mu m]$
300	100000	99930	99929	99997	99997
400	100000	99930	99930	99996	99997
500	100000	99933	99931	99998	99999
600	100000	99929	99924	99993	99993
700	100000	99925	99924	99984	99979

As shown in Table 5.1, the fringe count displacements are consistent with the commanded motion (with the exception of the 700 $\mu m \cdot s^{-1}$). The greater displacement discrepancies at the $700\,\mu m \cdot s^{-1}$ velocity trial can be credited to the linear stage encoder approaching the internal frequency limit, and noise causing fringe counting to be less reliable. The errors for the Renishaw and the fringe counted observations were determined using the distribution of differences between the commanded and observed travel. The Renishaw measurements are $72 \pm 3\,\mu m$ shorter than the commanded 100 mm of linear travel. If the alignment between the Renishaw and translation stage axes where responsible for this discrepancy, the corresponding angle was calculated to be $2.16 \pm 0.05^{\circ}$. The alignment procedure for the Renishaw interferometer involved sliding the stage the full 450 mm travel distance, and minimizing the drift observed by the spot created by the HeNe laser beam. Based on this alignment procedure, there is an upper limit on the relative alignment of these two axes of 0.25°. Thus, optical alignment can only be responsible $< 11\,\mu m$ of the $\sim 72\,\mu m$ discrepancy between the Renishaw measured distance and the commanded linear stage travel.

Further investigation into these data demonstrated a consistent 1.00065 ± 0.00004 cor-

rection. The Renishaw's resolution is 38.6 pm in vacuum, not accounting for atmospheric effects. Using the accepted standard value of the index of refraction of air of 1.0002714 (defined at an air pressure of 1013.25 mbar, temperature of 20°C and relative humidity of 50 %) explains in part the correction of the observed discrepancy. Future use of the Renishaw interferometer will be accompanied by measurement of the laboratory environment to allow for full compensation of the effective laser wavelength in air.

5.4.4 Sampling Frequency Test

The stability to record at high sampling frequencies was verified by recording at various rates, and then determining that all packets were received, and in order. Any error flags were taken as an immediate failure of the test. An ad-hoc network, where the ARTY was directly connected to the network port of the recording computer was established, such that no routers were between them. The calculated maximum bandwidth, based on the number of samples in the packet and 100 Mbps network speed, was 2.4 MSPS. The highest observed sample rate which triggered no errors was 2.1 MSPS.

5.5 Software Versions

The first cross platform library was written in Python, supplying the ability to set addresses, frequencies, and start and stop recording. The listening thread was included, but later was separated, making it possible to record the data on a network computer which was different from the serial attached control computer. A graphical user interface (GUI) was added by A. Sundberg during a 2018 summer internship, which also contained the option of a live plot display and a stand-alone listener with GUI. Both versions make it possible to operate the Renishaw either alone or scripted along with other devices.

5.6 Conclusions

The use of System Verilog, Verilog, and Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) programming languages at first seem non-intuitive, compared to computer and micro-controller languages (such as C++ and Python), which are usually sequential in execution. In FPGA programming all modules are executing at the same time, synchronized at the passing of the rising or falling edge of the clock, possibly asynchronously or using different clocks. This leaves the designer needing to employ careful consideration of the timing, and making sure that synchronization with different clock regimes stays consistent.

Besides the difficulty of programming, the FPGA provided a faster solution to acquiring data, producing over one hundred times improvement of sampling bandwidth from the second version of the two TEENSY2 interface board (20 kHz to 2 MHz). UDP transmission made most of the sample rate increase possible, with the ability to send over 8000 packets per second and 290 samples per packet resulting in a theoretical transfer bandwidth of 2.3 MSPS. There was a significant simplification of the PCB due to the FPGAs ability to emulate much of the original circuit, which justified the expense of the ARTY, saving time and money.

It is recommended to record a considerable large sample of test data from the Renishaw interferometer before any experimental session is started. Since environmental parameters may change the effective error in position measurements, a long sample of stationary motion is helpful in validating tolerances. Before any trials, it is recommended to take many samples of varied displacements to ascertain any scalar error. Current laboratory environmental conditions should be recorded to ascertain an accurate value for the refractive index of air.

The FPGA re-build of the Renishaw interface resulted in many of the same methods and tools required to redesign the discrete data acquisition circuit previously developed on an FPGA platform. The next chapter will explore the use of FPGAs to replace the circuit

developed in Chapter 4, with the goal of increased sampling bandwidth and precision of the coarse counting circuit.

Chapter 6

Creating the ARTY - DAQ

I do not pretend to start with precise questions.

I do not think you can start with anything precise.

You have to achieve such precision as you can, as you go along.

– Bertrand Russell

The discrete data acquisition system (DAQ), developed in Chapter 4, suffered from bandwidth issues due to the limitations of serial communications. Clearly, the advantages observed in the development of the Renishaw interface, namely the 100 Mb network communications, would be beneficial to the discrete DAQ. The Artix-7 FPGA Development Board (ARTY) also allows higher clock frequencies, up to 400 MHz, exceeding previously designed circuits by a factor of six times. This chapter describes the development of an FPGA-based DAQ system as a next-generation system to the TEENSY-DAQ described in Chapter 4.

A brief overview of the intended design of the data acquisition system is provided in Section 6.1. Section 6.1.2 discusses the software changes to the process of collecting and sending UDP packets, making it more streamlined and efficient. Sections 6.1.3 & 6.1.4 provide information on options that were designed, but have become future objectives, in order to get a functioning device in the allotted time frame. Section 6.1.5 examines the upgrades to the coarse counting clock from previous versions. Finally, Section 6.2 provides the results of testing the coarse counting clock precision and the ADC for accuracy and linearity.

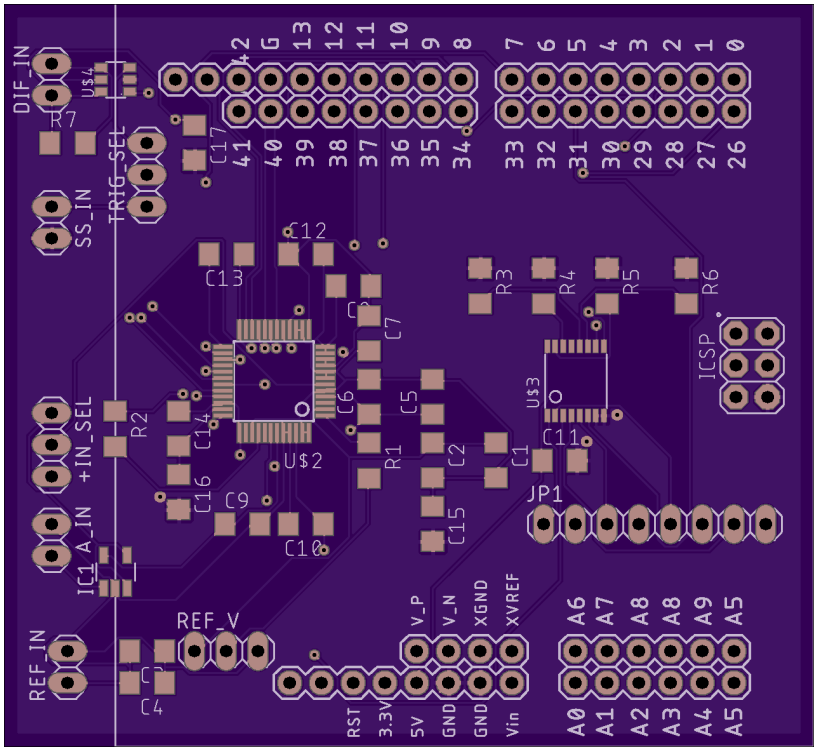


Figure 6.1: The first version of the ARTY DAQ shield, equipping the FPGA with a differential input and upgrading the ADC to 2 MSPS and 16-bit resolution. Components were placed only on the top side of the board, shown, but connections occur on the underside of the PCB as well. The design includes a multiplexer allowing up to four inputs to be read on the ADC.

A shield, shown in Figure 6.1 was designed to incorporate a 2 MSPS discrete ADC, exceeding the onboard 1 MSPS and 1 V dynamic range that is included with ARTY. The circuitry required to input the differential PSO signal, for use as a DRT, and a multiplexer (MUX) to allow four analog inputs, was implemented on the PCB. To be compatible with the already developed FPGA software described in Section 5.3, utilized on the Renishaw interface board, the main clock was set to 200 MHz to simplify synchronizing timing between modules.

6.1 Overview of Functional Design

The combination of the discrete DAQ, described in Chapter 4 and many of the core design features of the FPGA software developed for the Renishaw interface, provided in

Chapter 5, were required for the creation of the ARTY DAQ.

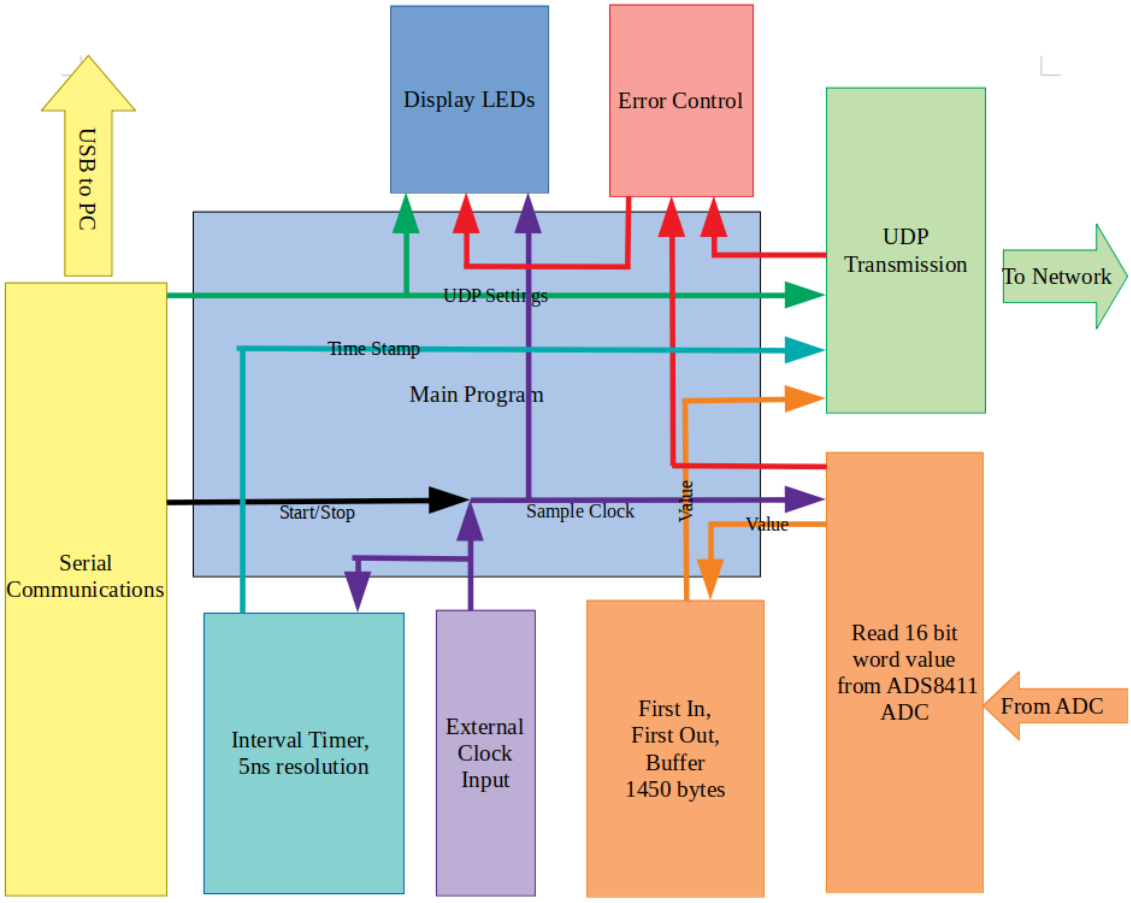


Figure 6.2: Similar to the Renishaw design, the ARTY DAQ removes the sample clock and adds an interval timer. The function of three Teensyduino micro-controllers (command, stopwatch and one ADC) was converted to a Verilog program run by the ARTY FPGA board.

The command micro-controller utilized on the discrete Teensy DAQ, was entirely replaced with the serial communication module, depicted in yellow on Figure 6.2. Similarly, the external 66 MHz oscillator, synchronous counting ICs, and buffers to latch the requested count value, were encoded into the interval timer module (in cyan), see Section 6.1.5 for details on enhancements to the coarse counting clock. A discrete ADC was incorporated to allow a higher sampling rate of 2 MSPS with 16-bit resolution, over the internal 12-bit and 1 MSPS ADC [50]. Like the Renishaw FPGA program, the design uses a single top module which all other modules must pass through to ensure timing remains synchronized.

6.1.1 Serial Communications Module

Using the same external library as in Section 5.3.3, serial communications was encoded to allow control from an external computer through a python library. The outgoing transmission of serial data was updated with a 32-byte (256 bit) shift register, instead of a 5-byte (40 bit) static outgoing buffer, used by the Renishaw interface. This provided the advantage of continuous communication, where information was not lost while waiting for the transmission to finish, and efficiency as only the data required to be transmitted was sent instead of 5-byte chunks of empty data. The serial module utilized the same handshaking as the Renishaw ARTY interface, where an asterisk ('*') was returned upon successful reception of a command and the character 'E' was returned for commands that were not understood.

6.1.2 UDP Transmission

The Renishaw interface board contained a shift register of 11600 bits (1450 bytes), the size of a transmission packet, which was used to pass data to the UDP transmitter. Registers of this size can cause issues compiling in the Vivado software suite, which tries to simulate all possible configurations with the goal of streamlining the end design. To correct the naive approach, a first in first out (FIFO) buffer replaced the shift register. Bytes of data were split into nibbles (four-bit blocks) and then placed into the FIFO least significant nibble first. The buffer was also configured for the first-word fall-through[55], where the top of the stack appears on the output buffer automatically, making it possible to read the output first and then request the next value.

In order to facilitate the Renishaw and the ARTY DAQ both sending data to the same computer, the port value was made programmable, versus the static value of 4096 which the Renishaw interface uses.

6.1.3 ADS8411–16-bit, 2 MSPS Analog to Digital Converter

The ARTY comes with a stock 1 MSPS 12-bit ADC. Consider the goal of matching the performance of the TEENSY DAQ circuit, where the maximum sampling rate achieved

was 120 kSPS. The increase in bit-depth that would be achievable can be found by using Equation 2.16.

$$\begin{aligned} f_o s &= 4^W \cdot f_s \\ 10^6\,[Hz] &= 4^W \cdot 1.20 \times 10^5\,[Hz] \\ 12 &= 4^W \\ W &= \frac{log(12)}{log(4)} \approx 1.8\,[bits] \end{aligned} \tag{6.1}$$

From Equation 6.1, it is clear that the 12-bit ADC would at best be able to perform similar to the ADS8371 ADC, as shown in Section 6.2.2. Through averaging, only 1.8 bits can be recovered, and since the effective number of bits (ENOB) of a 12-bit ADC will be less than 12, it is not likely to achieve better results.

Needing a faster ADC with better resolution, to utilize the advantages provided by the ARTY, the choice to use a discrete ADC was made. The ADS8411 provided a 2 MSPS sampling bandwidth and a 16-bit resolution. With a dynamic range of up to 4 V, it is possible to reverse-bias the photo-detectors for greater sensitivity[56], without saturating the ADC.

The shield was designed with the option to include a four-channel multiplexer, allowing four analog signals to be sent to the single ADC. There is a single jumper to select whether the multiplexer is in use or dormant. Current firmware on the ARTY board does not utilize this feature, and, due to time constraints this option is left as a future objective.

6.1.4 Sampling Clock

The current version only permits the ARTY DAQ to be triggered by an external source. A sampling clock, similar to the Renishaw interface board in Section 5.3.1 was planned, but was removed to simplify the code. During the original encoding of the firmware, many clock domains were required for the sub-systems, which was creating timing issues and preventing the code from operating. All unnecessary clocks were removed, leaving the required 25 MHz UDP transmission clock and the 200 MHz main system clock. A future

objective will be to re-implement the internal sampling clock, allowing the ADC to function without an external trigger. Since temporal metrology, timing intervals between external events was a project initiative, the internal sampling clock was not a priority.

6.1.5 Coarse Counting Clock Bit Size

In the previous versions of the circuit, the bit size of the counter has been limited by the ICs being used. The ripple counter had 12-bits, and the synchronous counter was composed of four 4-bit chips. The FPGA, consisting of programmable hardware, has the advantage of being able to create a synchronous counter of any size desirable. The counter was extended to 24-bits, making each sample for timestamp and ADC a total of five bytes. The advantages of having a larger bit size include less processing of overflow conditions and the ability to record at lower frequencies, without modification. The TEENSY DAQ had a $> 1\,\text{kHz}$ minimal sampling frequency to ensure that one sample was taken before the counter would overflow; Even with greater than three times faster clock frequency, the 24-bit counter can measure a maximum period of $83886\,\mu\text{s}$ or a minimum sampling frequency of $12\,\text{Hz}$. It is possible to modify the "command" TEENSY to record low order data rates, but doing so added an extra layer of post-processing.

It may be desirable to record data at even lower sampling frequencies than $12\,\text{Hz}$. For instance, measuring the temperature change during warm-up or cool-down of a cryostat is a slow process requiring hours and sometimes days to complete. Although, accurate measurement of large time periods to the precision of nanoseconds would be excessive, it could be achieved through a few modifications. A 32-bit register would allow $21.4\,\text{s}$ intervals or $46.5\,\text{mHz}$, but would sacrifice the amount of data that could be delivered in a single packet. A method that was utilized with the discrete DAQ, was to use a micro-controller with a coarser clock to provide a rough time measurement of events and finer clock overflows, allowing reconstruction of the timeline with the finer clock. If a programmable counting clock was utilized, or possibly two clocks (one with larger granularity), it would be possible

to adjust the coarseness to suit the situation.

6.2 Implementation and Testing Results

6.2.1 Interval Timing

Initially, a 400 MHz coarse counting clock was tested for the timing of event intervals; the results are illustrated in figure 6.3. Although abandoned due to timing and compatibility issues with existing code, this clearly shows the accuracy possible, ± 1 clock tick, when using FPGA devices. The lowest data sampling frequency that could be measured without timing errors due to the counter roll-over was 23.84 Hz, due to the 24-bit counter. This is an advantage over the discrete Teensy board, measuring periods forty-one times larger. The upper limit of the sampling rate is determined by the amount of data that can be placed in a packet, and the associated UDP bandwidth. Theoretically, packet size can be upwards of 65536 bytes, but large packet size can lead to lost fragments during network transversal. UDP does not have a mechanism to request re-transmission of packets, thus the entire packet would be lost. The packet size of 1452 data bytes was selected to match the maximum transmission unit (MTU) of most network switches and routers, minimizing data loss. The bandwidth of the UDP network was 100 Mb, which would allow approximately 3.8 MSPS when IPv4 and UDP header size was taken into consideration. It should be noted, the maximum bandwidth calculated is for a packet consisting of 3-byte temporal metrology data only, and will decrease if additional data, such as from the ADC, reduces the number of samples per packet.

Timing complications arose while implementing the ADC, thus the main clock which operated the coarse counter was reduced to 200 MHz. To ensure the 200 MHz clock responded equivalently to the 400 MHz event timer in Figure 6.3, the process of verifying the accuracy of time interval measurement was repeated, shown in Figure 6.4. Since the ADC inclusive version included a discrete ADC with a 2 MSPS maximum, the fastest sampling was restricted to 2 MSPS.

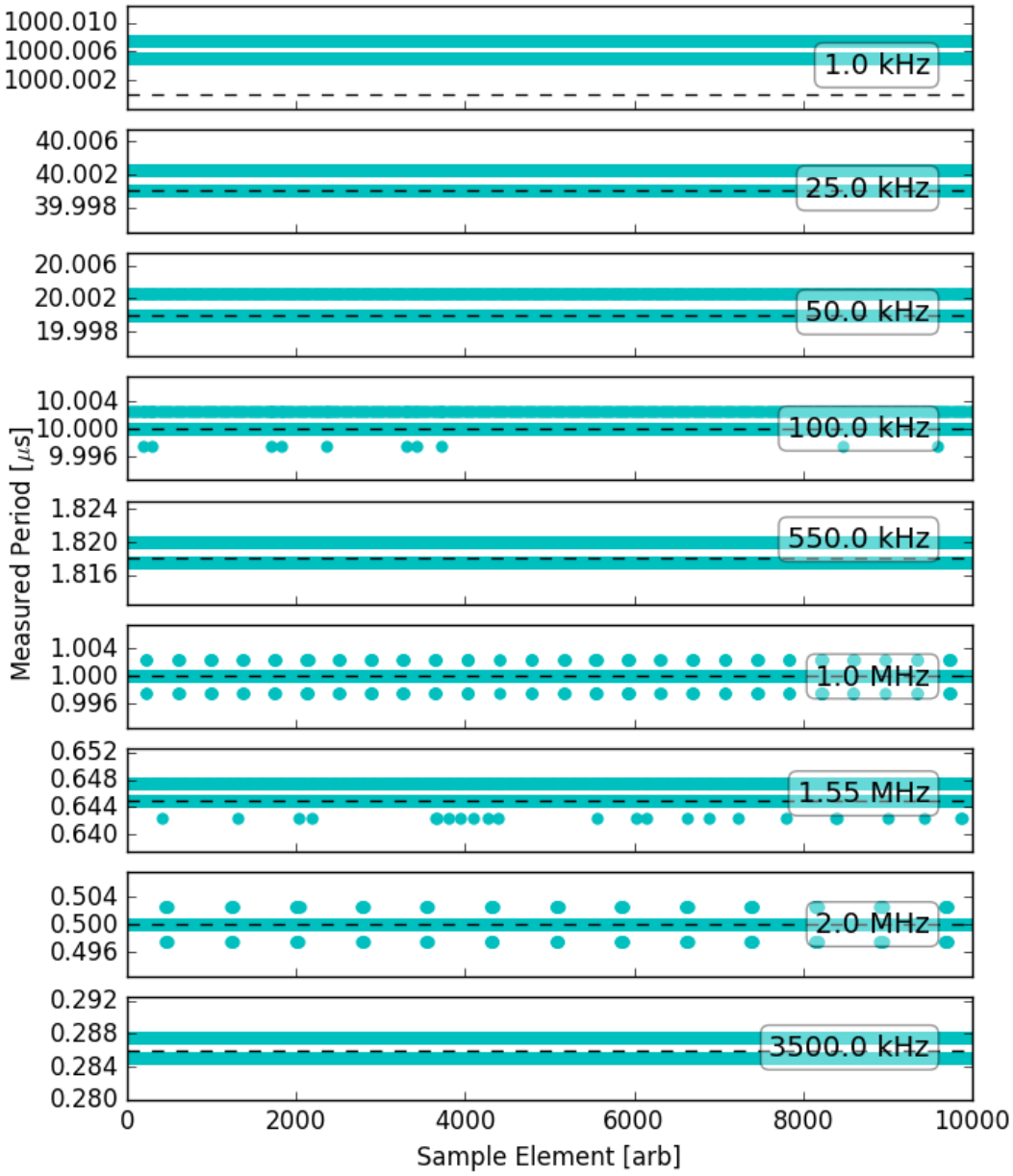


Figure 6.3: A 24-bit counting circuit, incrementing at 400 MHz, was read at even intervals provided by a function generator at various frequencies. The chosen frequencies were picked to depict the range of operation. The ideal period is marked with the dashed black line. The spacing of the digitization is exactly ± 1 clock tick of the 2.5 ns.

With half the clock resolution, Figure 6.4 depicts intervals being measured within $\pm$ 1 clock pulse, a resolution of 5 ns. The 1 kHz samples in both Figures 6.3 & 6.4 demonstrate a slight imprecision in the function generator of 0.001% error. The evenly spaced fluctuations in the period, seen in both Figures, could be the systematic compiling of the error, a real physical jitter in the function generator, or a combination of both.

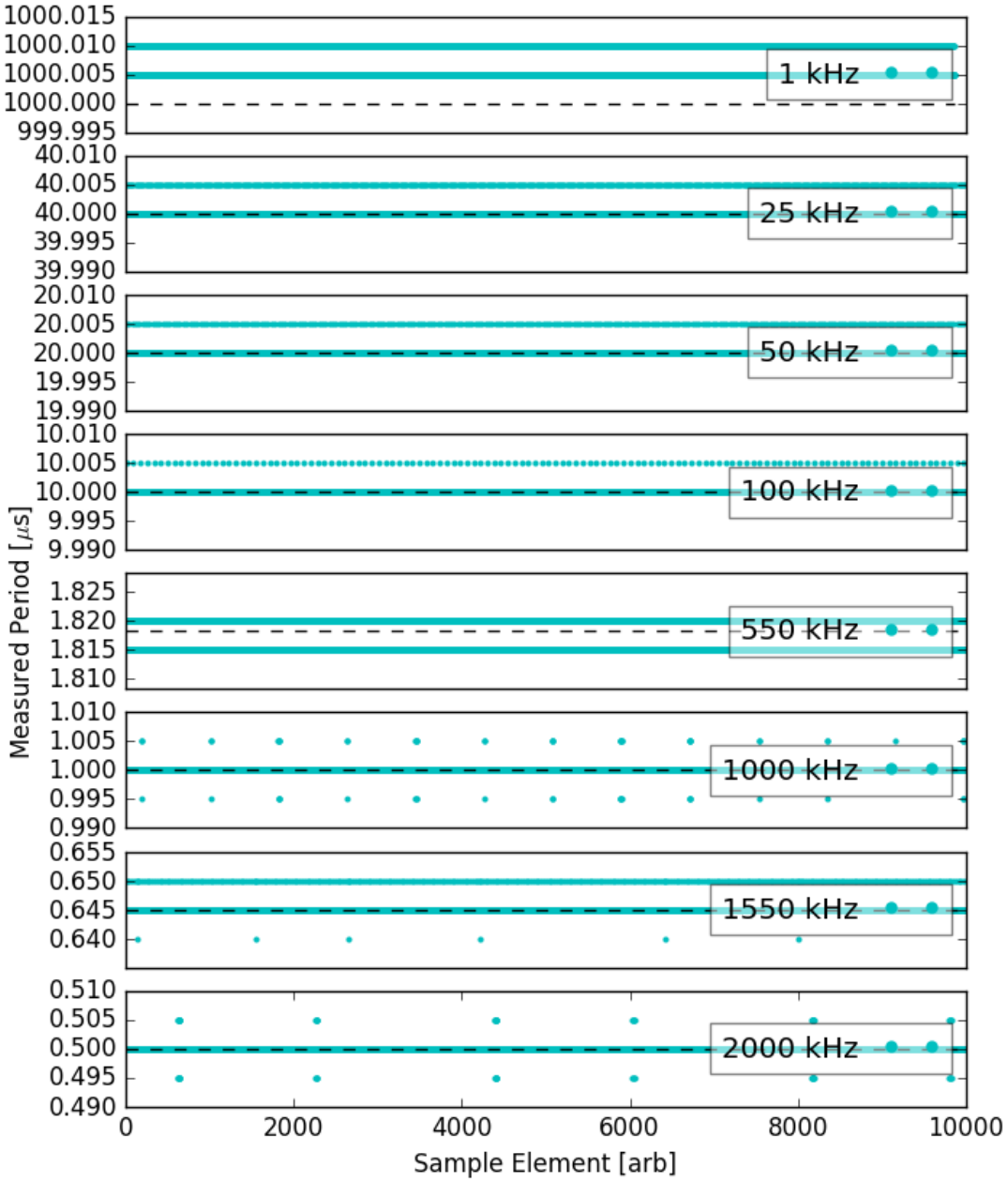


Figure 6.4: A 24-bit counting circuit with ADC, incrementing at 200 MHz, was read at even intervals provided by a function generator at various frequencies. The ideal period is marked with the dashed black line. The highest period measured was 500 ns, as 2 MSPS is the greatest sampling rate for the ADC.

6.2.2 ADS8411 Analog to Digital Converter

The ADS8411 ADC had the exact same footprint and similar circuit requirements as the ADS8371 discussed in Chapter 4.7.1. There were two major differences between the ADCs. The 4.09 V internal reference voltage and the increased sampling bandwidth of 2 MSPS were provided on the ADS8411. Where, the ADS8371 had an external reference voltage only and 750 kSPS sampling bandwidth. Unlike the discrete circuit that was designed previously, without the serial communications bottleneck of the micro-controllers full sampling speed was achieved with the ARTY FPGA.

Results of ADC Testing

While sampling at 100 kSPS, a slow ramp wave, with a period of 60 seconds, and approximately 75% full-scale range, was used to estimate SNR and ENOB. A histogram was generated to depict the linearity of the ADCs response.

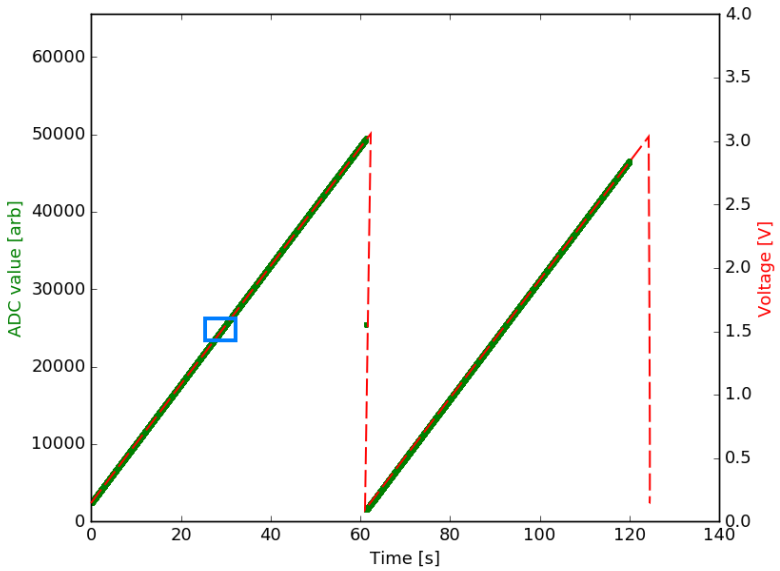


Figure 6.5: Example of ADS8411 ADC sampling a 16 mHz ramp wave compared to ideal signal (red). One full period of a 16 mHz ramp wave covered approximately 75% full scale of the ADC. The blue square is presented in Figure 6.6.

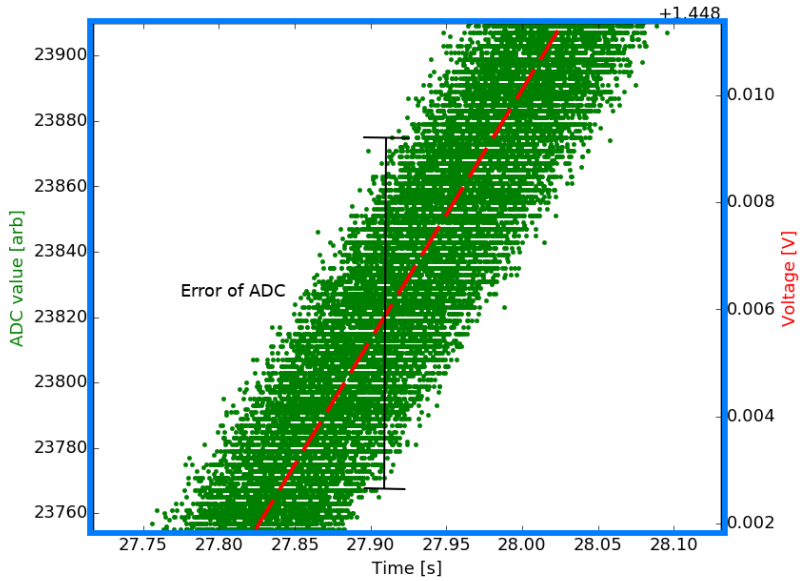


Figure 6.6: Inset to Figure 6.5 displays the level of error present in the converted samples by the ADS8411 ADC. The range of error transverses close to 100 arbitrary ADC integer bins, suggesting the lower 7 bits were ineffective. This was confirmed by calculating the SNR of 58.5 dB and ENOB of 8.7 bits.

Overall, the result in Figure 6.5 look as expected, but the 58.5 dB SNR and 8.7 ENOB calculated from fitting the ramp wave with a linear regression are much lower than the suggested 82 dB and 13.4 ENOB by the data sheet[57]. Assuming that every ADC value should have been sampled a similar number of times, over two recorded periods of the ramp-wave, except for the regions purposely not engaged as the wave only covered 75% full-scale, there should be approximately 250 samples, about 0.002%, for each ADC bin.

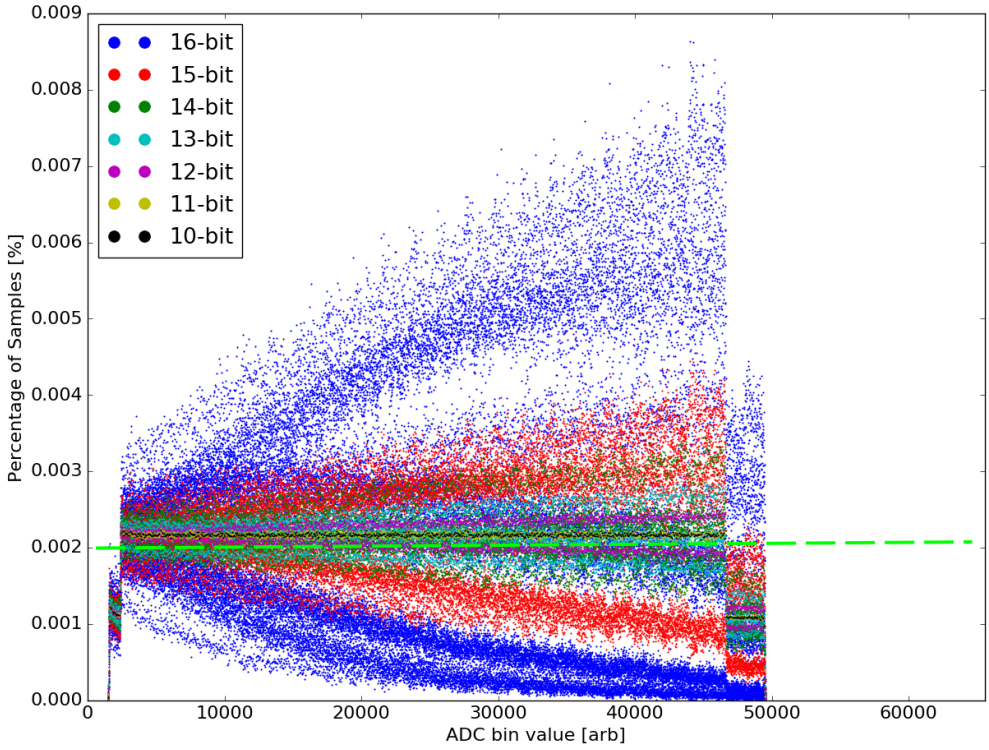


Figure 6.7: Distribution of 16-bit ADC values demonstrating linearity of response for 75% dynamic range of the ADS8411 converter. A 75% full-scale ramp wave, with a period of 60 seconds was sampled at 100 kHz. An expected 0.002% of samples should have been placed in each of the bins between 1500-50000, indicated by the green dashed line.

Clearly, Figure 6.7 displays that the ADC is not responding linearly, as select bins appear to be favoured above others. This was attempted at different sampling frequencies from 10 kHz to 1 MHz, with almost identical results every time. The spacing of the favoured bins was consistent and periodic, suggesting an oscillating signal distorting the ADC results.

The circuit was dismantled, first removing the operational amplifier that was used as a buffer to protect the ADC from voltages greater than maximum allowed input. Care was taken that signals stayed within specification. This yielded the same results as before. As the noise was distorting the LSB, the use of the BYTE pin to fold the LSB onto the MSB pins, requiring two separate read stages, was implemented. The result was the same, implying that noise was not on the output side of the ADC, and coming from a different

source.

It was noticed that when no 1 Hz ramp-wave signal was applied to the ADC inputs, but the function generator, which provided the request for a sample was operational, a signal leaked into the ADC inputs. The common ground on the ARTY 5V and 3.3V power supplies, which also powered the ADC, was introducing a ground loop fault. Ground loop faults, generally heard as a hum in audio or ghosting in video applications, are the result of oscillation of current resultant from a difference in impedance for paths back to the ground.

6.2.3 Isolation and the New Prototype

The first version of the ARTY circuit was based on the discrete Teensy DAQ design. The 750 kSPS ADS8371 was less susceptible to high-frequency noise due to the fact the serial communications bottleneck reduced the bandwidth to 120 kSPS. The discrete board also employed its own power supply and regulators for 3.3 V and 5.0 V, and didn't rely on the USB power connection as the power source. Both designs utilized decoupling capacitors on all power pins of the ADC, which was insufficient for eliminating noise on either the ADS8371 or the ADS8411 ICs.

The proposed second version of the ARTY DAQ will be operated with an external power supply, separating the USB power from the circuit and removing noise generated by the controlling computer power supply. Ground loops must be eliminated by using a 5 V DC to DC voltage converter to power the analog input side of the ADC. The digital output side of the ADC should still be powered with the 3.3 V supply provided from the ARTY FPGA, but the ground traces will be enlarged to reduce impedance and additional low-pass filtering is also included. Isolating the 3.3 V digital side was considered, but many factors made the decision to take other precautions more desirable. The main deciding factor was that the propagation time for an opto-isolator was up to 10 MHz, which would reduce the maximum sampling rate to 1.6 MHz, due to the increased delay in addition to the required sampling time. The additional expense of four 4-channel opto-isolators was another factor,

which could be mitigated by only using the option for 8-bit reading (reducing the number of opto-isolators by two, by reading bits 0→7 and then 8→15), but the additional read time, due to propagation, would double further reducing the maximum sampling rate to 1.4 MHz.

Control pins, used to read, reset, fold the output to 8-bit, and start the conversion of samples, also must be isolated. An opto-isolator, a device that utilizes an LED and detector combination, will allow signals provided by the ARTY GPIO to be transferred from the FPGA domain to the isolated 5 V analog domain of the ADC.

Finally, when routing the PCB care must be taken that signal traces are not placed adjacent to one another to prevent any induction between the lines. The use of terminating resistors and separate ground traces of similar length should be provided for each signal, minimizing and isolating any difference in impedance on routes back to the ground.

6.3 Conclusions

In the case of measuring event-driven timing intervals, the FPGA is superior to other methods previously used. When used as an event timer only, there was at least a thousand times improvement of timing accuracy compared to a micro-controller alone, with only a minor decrease in performance when used with the external ADC.

The SNR of 58.5 dB being up to 30 dB less than specification, resulting in the loss of 7 effective bits, was poor compared to the other ADCs utilized and specifications provided on the ADS8411 data sheet. The signal was adequate for the purposes of fringe counting, discussed in Chapter 7, but considerably lower than suggested by the data sheet. An isolated design is currently being fabricated during the time of this writing and testing will be performed again on the next version.

Updating the methods used to buffer and send data to the UDP transmission module has made the FPGA code more robust. A programmable port value allows for the option of multiple devices being able to send to a single computer. Implementing other features such as the programmable sample clock and multiplexed input for the ADC are required future

work, which will provide a flexible and accurate laboratory instrument.

In Chapter 7 the performance of the instrumentation are evaluated in a real physical setting, and the function of the Aerotech linear stage will be investigated.

Chapter 7

Aerotech Linear Stage

"When a man in a forest thinks he is going forward in a straight line, in reality he is going in a circle, I did my best to go in a circle, hoping to go in a straight line."

– Samuel Becket, Molloy

The two linear translation stages utilized in the test bed instrument, introduced in Figure 1.1, are a key component. The image of one Aerotech ALS20045 linear translation stage, seen in Figure 7.1, is provided as an example. The spectral arm used a rapid-scan mode, developed by Mertz[58] and employed, for example, by the Herschel SPIRE imaging spectrometer instrument[24], where the optical signal was sampled at uniform increments of time while the stage travelled at a constant velocity. For this reason, it is critical to understand the velocity profile of the stage, so corrections can be performed to improve the spectral response and resolution of the test bed instrument.

Figure 7.1: Photo of Aerotech ALS20045 linear translation stage. Two ALS20045 Aerotech linear stages, one shown here as an example, are the building blocks for the proposed spatial/spectral FIR test bed interferometer.

Section 7.1 explores the Aerotech feedback mechanism which balances error in position, velocity, and acceleration of the linear stage. The Python library developed to allow communications through the network interface with the stage controller is presented in Section 7.2. Determining the accuracy of the velocity and position, exploiting the devices designed in the previous chapters, is presented in Sections 7.3 and 7.4. Finally, a noise profile is ascertained in Section 7.5, for use in future simulation work.

7.1 Motor Controller and PID Settings

The servo loop mechanism was intended to minimize the error in the velocity and position, providing accuracy and control of the linear stage. The standard equation for a PID controller, Eq. 2.2, was intended for a single feedback source. The block diagram in Figure 7.2, provided by Aerotech [30], depicts a parallel system. Changes to the mass and distribution of weight on the linear translation stage can have profound effects on the efficiency of the feedback loop. The servo loop update rate occurs at a default 4 kHz, but is adjustable in the linear stage parameters for values of 1, 2, 4, 5, 10 and 20 kHz. The 20 kHz update rate is not recommended as the Ensemble controller may have performance issues[49].

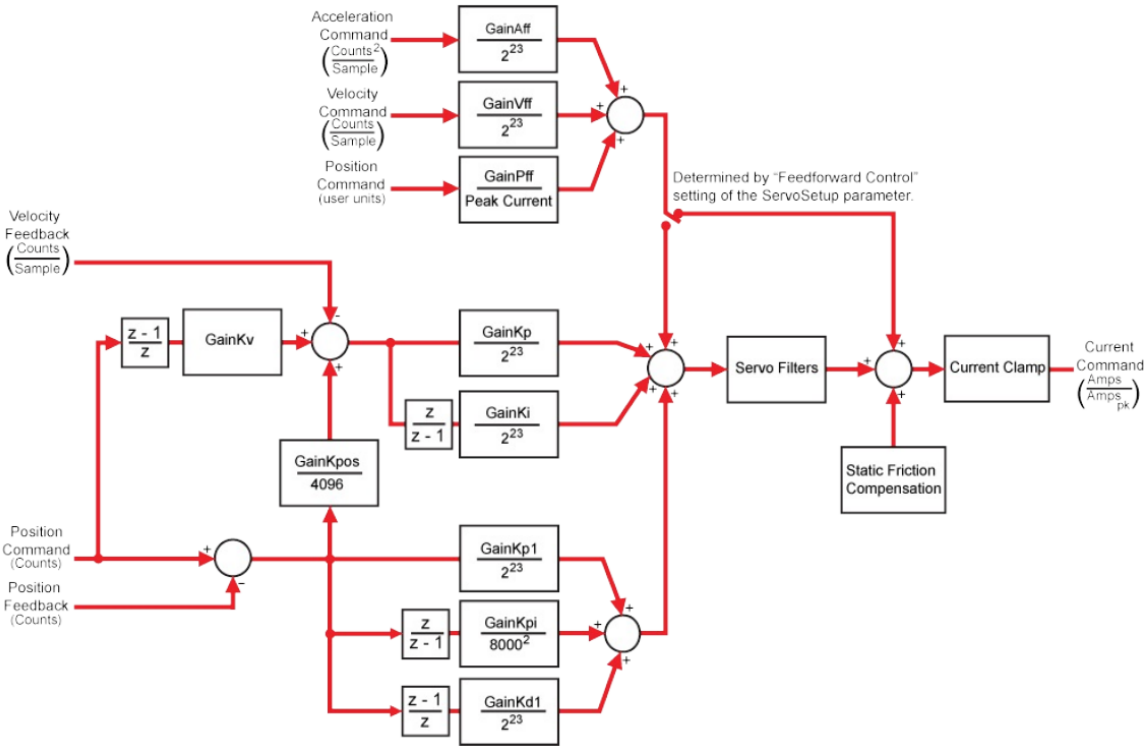


Figure 7.2: Schematic of the Aerotech Servoloop, (i.e., the feedback loop used on the Aerotech linear stage), obtained from the Aerotech help manuals[48]. The stages use multiple feedback sources to balance position, velocity and acceleration errors. The feedback from velocity and position sensors are used to set the current for the servo motor.

The block diagram, Figure 7.2, can be reduced to a transfer function such that

$$C(z) = \left[K_p + K_i \frac{1}{1-z^{-1}} + K_d\left(1-z^{-1}\right)\right] \times \left[\left(1 + \frac{GainKsi1 \times 2\pi}{ServoRate(Hz)} \times \frac{1}{1-z^{-1}}\right)\left(1 + \frac{GainKsi2 \times 2\pi}{ServoRate(Hz)} \times \frac{1}{1-z^{-1}}\right)\right] \tag{7.1}$$

where

$$K_p = \left(\frac{GainKi}{2^{23}} + \frac{GainKp}{2^{23}} \times \frac{GainKpos}{4096} + \frac{GainKp1}{2^{23}},\right) \tag{7.2}$$

$$K_i = \left(\frac{GainKpi}{8000^2} + \frac{GainKi}{2^{23}} \times \frac{GainKpos}{4096}\right), \tag{7.3}$$

$$K_d = \left(\frac{GainKp}{2^{23}} + \frac{GainKd1}{2^{23}}\right), \tag{7.4}$$

The proportional factor, Eq. 7.2, provides an immediate response to any disturbances to the velocity and position. Integrating error over time, K_i attempts to correct for any systematic error that is constant. The derivative of the error in position and velocity, Eq. 7.4, provides an adjustment dependant on the rate the error is changing. Thus, for small fluctuations or noise, the derivative term may retard the proportional response, but large fast changes may cause spikes.

Methods for adjusting the tuning parameters were attempted, normally resulting in the linear stage emitting a whistle, while it tried to hold a stationary position. The whistle was caused by constant corrections to the position of the stage, creating a vibration instead of holding a still position. Using an iterative search method for three variables is difficult, as discussed in Section 2.2.1, and due to the parallel nature of the Aerotech servo loop, the search space is extended for nine variables.

Aerotech provides software (requiring Windows OS) for auto-tuning the controller for the linear stage, which minimizes error while monitoring the motion of the stage. The auto-tuning is specific to the load configuration and should be run when the stage load configu-

ration changes. The auto-tune software uses a spike in velocity to simulate a perturbation to the stage. Performance after using the auto-tune software was preferential, compared to any manual adjustments. In many cases, the auto-tune software was the only way to restore functionality to the linear stage, after manual methods created undesirable effects.

7.2 Python Library for Communication with Linear Stages

Many of the laboratory computers function with a Linux operating system, which was not supported by the Aerotech Ensemble software. To work around this issue and provide continuity between the developed devices, a Python library was written for communication and control of the linear stages. The required changes to the stage configuration and an outline of the commands available are listed in Appendix E. While tuning of the stage PID parameters still requires Windows OS and the Ensemble configuration software, operation is platform independent. Although the library contains the ability to set some parameters, through the `setparm()` method, most require a reset of the Ensemble controller. For this reason, setting parameters outside of the Ensemble software is cumbersome and is not recommended.

7.3 Verifying Motion and Position

In order to test the discrete TEENSY DAQ, ARTY DAQ and ARTY version of the Renishaw interface with a physical system, a preliminary profile for a portion of travel on one of the Aerotech linear stages was performed, utilizing a Michelson interferometer. The Teensy DAQ was configured to record the time of the PSO pulses and the detector signal monitoring the Michelson interferometer detector photo-diode. Unfortunately, the detector signal was connected to the wrong input port of the Teensy DAQ and was not recorded, but the timing of the PSO pulses was successfully recorded. The Arty FPGA and Renishaw were triggered externally by an Agilent 33220A arbitrary waveform generator, which was set to provide a 5.0V pulse with a width of 5.0 μs at a frequency of 50 kHz. The ARTY

DAQ recorded the time of the pulses and the intensity of the Michelson interferometer, while the Renishaw provided the displacement of the stage relative to the starting position of a predetermined set of motions.

For all trials, the linear motion was preceded by five periods of a quarter Hertz oscillation which had an amplitude of 1 mm. The linear motion was performed at five different velocities, ranging from 300 - 700 $\mu m \cdot s^{-1}$, in both forward and reverse directions.

7.3.1 Fringe Counting Routine

The fringe counting performed in Section 5.4.2 only considered the total number of peaks to estimate the distance. In order to provide a more detailed time and position, based on the fringe data from the HeNe, a different method was employed. In a similar fashion as Section 5.4.3, the indices of the peak data elements were identified by the SciPy signal method `find_peaks`. The indices of the troughs were then identified, by using the same method on an inverted data set. A verification that peak and trough indices alternated accordingly, was performed, and any adjacent peaks or troughs were ignored. An array of data, consisting of ten elements on either side of the peak or trough, was then fit to a second-degree polynomial. When the derivative of the polynomial was equated to zero, the time of the peak or trough was recorded. The distance between a peak to a trough, or vice versa, represents the wavelength of the laser divided by four in physical displacement. The wavelength of the laser was provided as 632.80 ± 0.05 nm, therefore each peak-trough / trough-peak transition was the equivalent of 158.200 ± 0.006 nm of linear stage displacement.

Since the effective number of bits (ENOB) of the ARTY DAQ was less than ten, the HeNe signal values were bitwise shifted seven places to the right, resulting in a 9-bit ADC signal. Truncating the uncertain bits is analogous to not reporting digits beyond significant figures. An averaging of samples would have also smoothed the noise present on the photo-diode data, restoring confidence in more bits, at the cost of sampling bandwidth. A

reduction of sampling rate would have hidden fine details and masked frequencies present in the signal, due to a decrease in Nyquist frequency. After providing a correction factor to the Renishaw displacement metrology, determined in Section 5.4.3, the data were then interpolated to the same time scale as the fringe counted displacement measurements. A time shift and offset were applied to the Renishaw data, in order to align the starting of motion of the stage with the displacement measurements constructed from fringe counting.

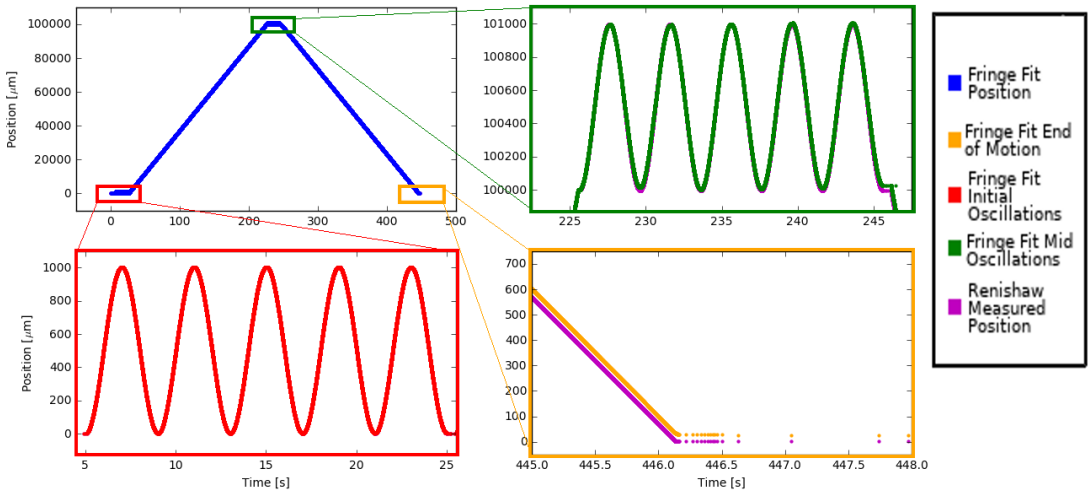


Figure 7.3: Fringe counting of the interference pattern of an HeNe laser was compared to the commercial Renishaw system. A correction factor of 1.0006155 was required to align the Renishaw measured displacement (magenta) with the calculated displacement (blue, green, red and orange) due to fringe counting. The HeNe laser experienced poorer signal after the first 100 mm travel, causing the second set of oscillations (upper right) to miscount fringes, noticeable by the poorer overlap. The linear stage was commanded to travel at $500\,\mu m \cdot s^{-1}$ for the linear portions and oscillate at 0.25 Hz with an amplitude of 1 mm for the sinusoidal sections. The error in the Renishaw and HeNe measurements are too small to see on this scale.

The scale of the agreement between counting fringes and the Renishaw interferometer is best demonstrated by the residual between the two measurements. The points where constant velocity travel started and stopped were identified by the derivative of the position/time plot. Discrepancies in the starting position, attributed to poor fringe counting were compensated for in the reverse travel direction only. Finally, the reverse travel residual array was inverted, such that the position of the stage compared to the homed position was the same for both forward and reverse motion.

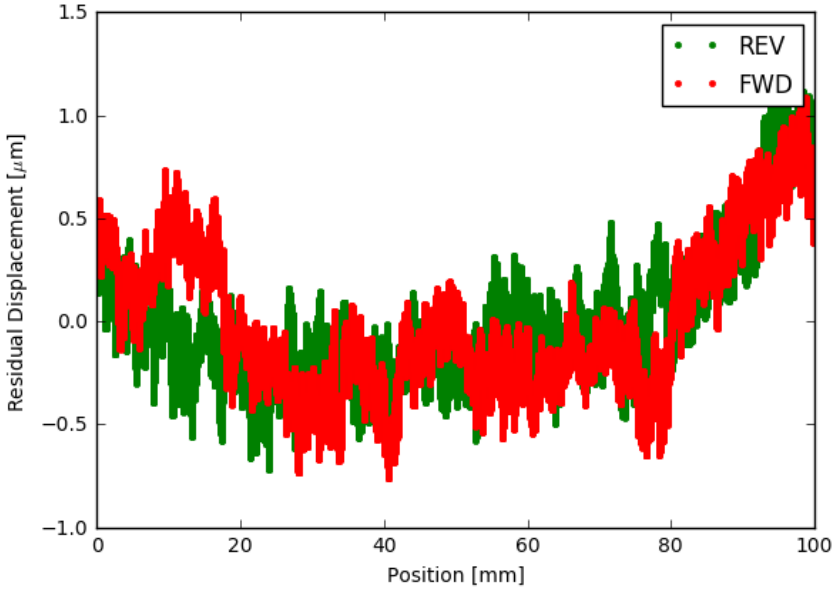


Figure 7.4: The residual difference in the measured position of the Aerotech linear stage over 100 mm of travel. Both the Renishaw and Michelson interferometers were in agreement within 1 μm. The example data was recorded while the linear stage was commanded to travel 100 mm at a velocity of 500 $\mu m \cdot s^{-1}$.

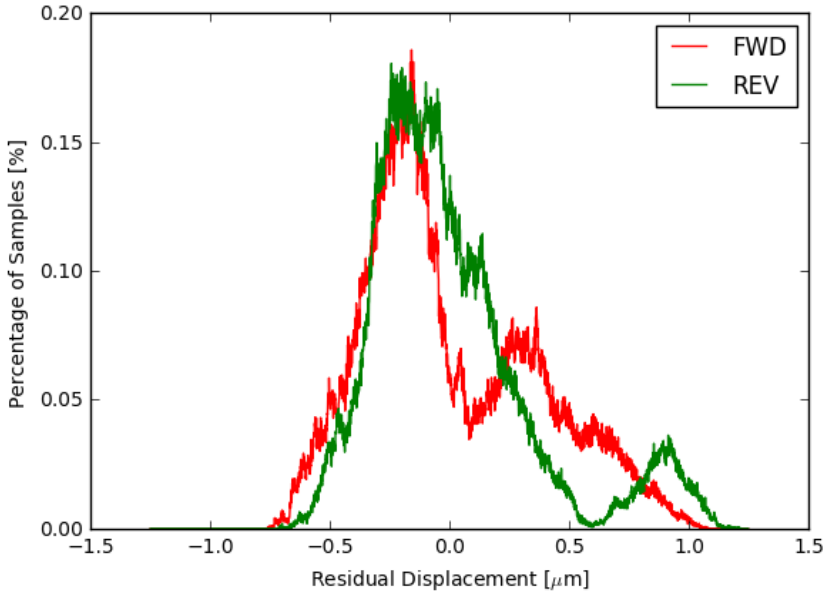


Figure 7.5: The distribution of the residual displacement from the best fit line, depicted in Figure 7.4, for 100 mm forward and reverse travel of the Aerotech linear translation stage.

Figure 7.4 confirms that the linear stage, Renishaw, and Michelson interferometer all

agree within error when the stage was commanded to travel forward and reverse for 100 mm at a speed of $500\,\mu m \cdot s^{-1}$. Although not shown here, the other trial speeds of 300, 400, 600 and $700\,\mu m \cdot s^{-1}$ all had similar results, where the residual displacement between the corrected Renishaw, using the same static factor, and the HeNe fringe counting was on the scale of ± 1 micron, and also within the accuracy of the linear stage.

The distribution of the error in position, illustrated in Figure 7.5, shows a main central peak which appears Gaussian almost about zero (a slight offset still exists, due to imprecise fit parameters). This feature was apparent for all trials, forward and reverse. The secondary peaks appear at different locations for all trials, suggesting possible missed counted fringes or bumps that jarred the actual position of the stage. Further investigation is required, into the non-linear residual shown in Figure 7.4, and the uneven associated distribution shown if Figure 7.5.

7.3.2 Linear Fit to determine Velocity

Using SciPy's `linregress()` function to ascertain the slope of the forward and reverse best-fit line for the linear portions of Figure 7.3, and by definition the velocity, yielded the results shown in Table 7.1.

Table 7.1: The forward and reverse constant velocity travel sections were fit with a linear fit. The slope of the best fit line provided a measure of the mean stage velocity. The error in the slope was $> 10^{-5}$, demonstrating the quality of fit, but the slope was only reported with three decimal places as position measurement accuracy was not better than this.

Commanded Velocity $[\mu m \cdot s^{-1}]$	Direction	Linear Fit Slope $[\mu m \cdot s^{-1}]$
300	FWD	299.983
300	REV	299.980
400	FWD	399.975
400	REV	399.973
500	FWD	499.969
500	REV	499.966
600	FWD	599.962
600	REV	599.958
700	FWD	699.956
700	REV	699.955

The derivative of the time array was then calculated using a 5-point stencil, and given that the change in position was the constant fringe interval, the instantaneous velocity was determined. Subtracting the slope of the best fit line, the residual velocity was then plotted, and a histogram was constructed (see Figure 7.7).

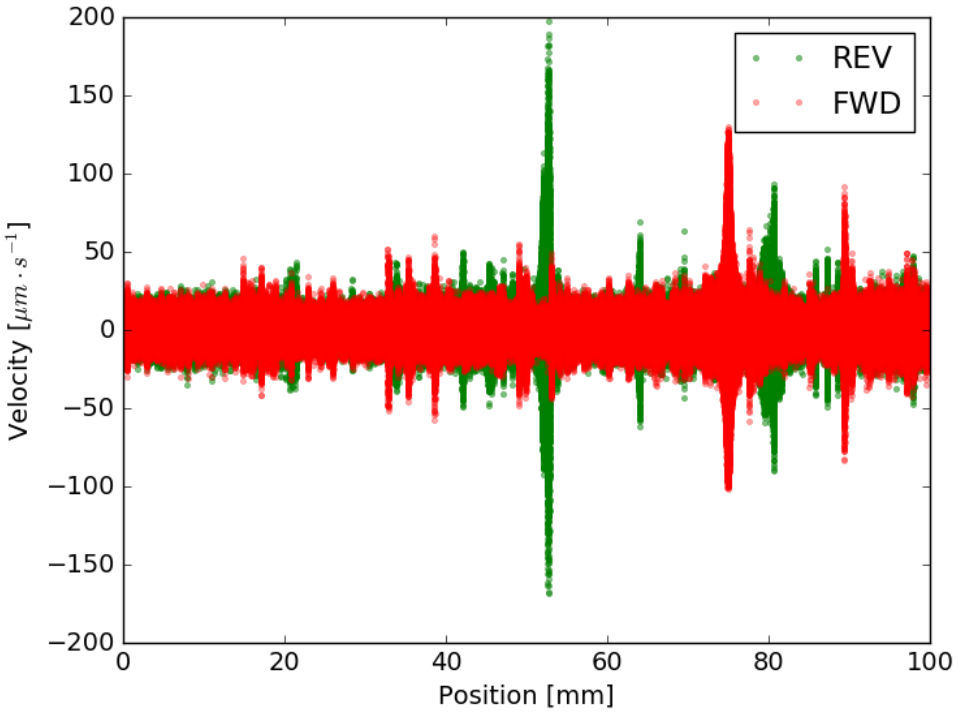


Figure 7.6: The residual velocity profile of the Aerotech linear translation stage from the slope of the best fit line. Subtracting the slope of the best fit line, from the derivative of the 100 mm forward and reverse travel data sets, resulted in the residual velocity profile. Spikes in the residual velocity were attributed to small vibrations, noise due to numerical derivative, and disturbances like bumps to the table, occurring during sampling.

Figure 7.6 shows the velocity deviation of the linear translation stage from the best fit line. The large spikes are attributed to vibration, possibly air turbulence or foot-steps in the building which caused deviation from a constant velocity. These spikes occur at different positions and times, and were inconsistent for each trial. This is evidence that it is not a physical issue with the stage, but likely an external source affecting the end results.

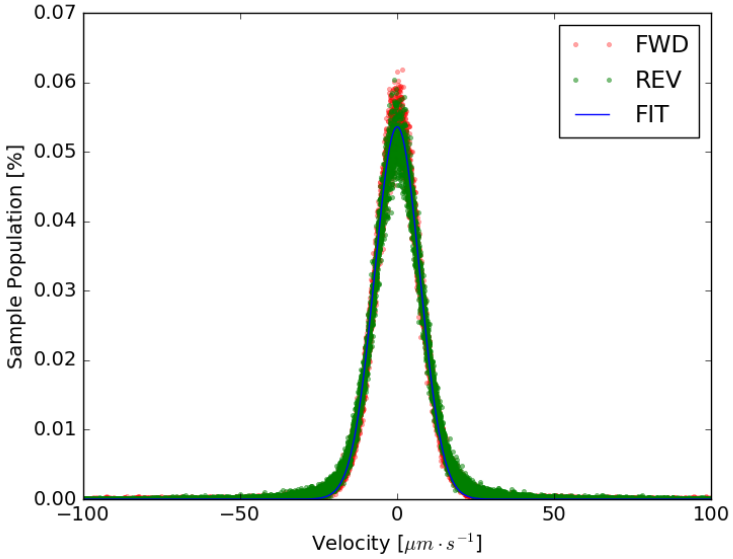


Figure 7.7: The Gaussian fit of the distribution of residual velocity of the Aerotech linear stage. The Gaussian fit parameters returned a mean value of $0.025\,\mu m \cdot s^{-1}$, essentially zero as expected, and a standard deviation of $7.2\,\mu m \cdot s^{-1}$. The stage was travelling with a $500\,\mu m \cdot s^{-1}$ velocity over 100 mm displacement.

White noise, the effect of random error, is expected to appear with a Gaussian distribution. For each trial, the histogram of the forward velocity residual was fit with a Gaussian curve. The standard deviation was taken to be the error in commanded velocity that exists due to the standard operation of the linear stage.

Table 7.2: The error in the velocity of the Aerotech linear stage compared to the commanded speed. The mean velocity was rounded to the nearest $\mu m \cdot s^{-1}$.

Commanded Velocity $[\mu m \cdot s^{-1}]$	Gaussian Fit Velocity $[\mu m \cdot s^{-1}]$
300	300±6
400	400±7
500	500±7
600	600±8
700	700±8

The results from Figure 7.7 suggest that the error in the magnitude of velocity is in-

dependent on the actual speed, but rather a measure of the ability of the PID controller to adjust for small fluctuations due to mechanical resistance, and friction. Keep in mind that, the discrepancy between the errors reported in Tables 7.1 & 7.2 is the difference between the error present in an average value and the error in the commanded velocity at any given time.

7.4 Position Synchronized Output Accuracy

The desire to trigger an event, or in our case take a sample with a detector, when the linear translation stage has travelled a specified distance is the function of the position synchronized output (PSO), described initially in Section 2.2.3. To provide a measure of the accuracy of the PSO, the Renishaw interferometer was externally triggered by the Aerotech linear stage. The difference of the measured positions was then fit with a Gaussian curve to provide a measure of error.

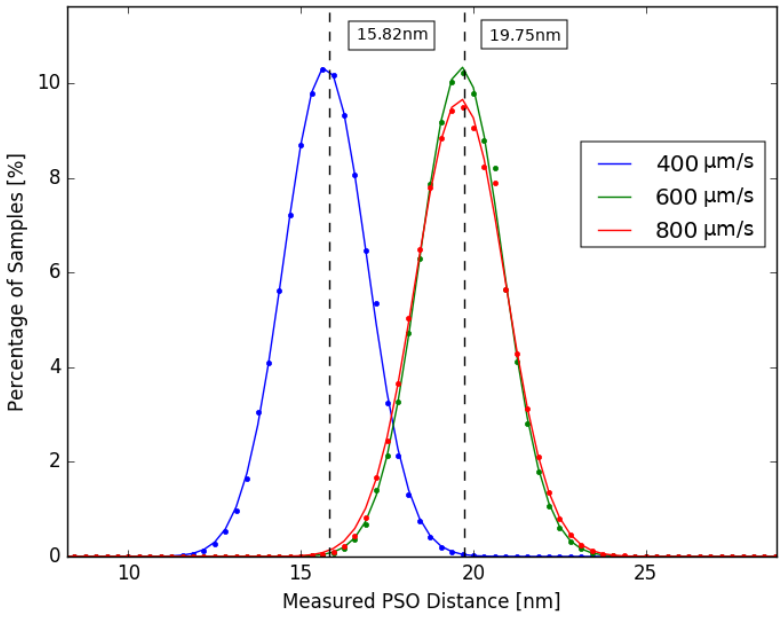


Figure 7.8: Verification of the accuracy of the PSO distance. The commanded distance between position synchronized output (PSO) pulses was set for 15.82 nm and 19.75 nm respectively (marked with the dashed black vertical lines). The Renishaw interferometer was then triggered, recording the displacement between pulses.

The standard deviation of the Gaussian fit, for the 400 and 600 $\mu m \cdot s^{-1}$ data was 1.2 nm.

The $800\,\mu m \cdot s^{-1}$ data set had a standard deviation of 1.3 nm. Illustrated in Figure 7.8, the mean PSO distance for the 400, 600 and $800\,\mu m \cdot s^{-1}$ was 15.71, 19.64 and 19.62 nm, respectively. Given that the Renishaw displayed an accuracy of 30 nm while measuring the displacement of a stationary object, described in Section 5.4.2, the accuracy demonstrated in Figure 7.8 exhibits a magnitude better resolution while measuring the displacement of a moving target. This apparent increase in resolution may be due to the lower sampling rate of 25 kHz (opposed to the 250 kHz of the stationary trials) or implies that at some level of consistent motion, the SNR of the quadrature signal that the Renishaw interferometer uses to measure position changes. The proportionate level of error contribution in the measured PSO distance between the linear translation stage and Renishaw interferometer requires more study. The choice for nanometre scale PSO distances was based on the desire to sample the HeNe laser, used as a test case to sub-Nyquist resolution. Since the actual instrument will operate with a larger wavelength, tens to hundreds of micrometers, PSO distances will most likely be on the scale of microns. Further testing of the PSO accuracy must be performed, especially at the distances most likely to be used by the actual test bed instrument.

7.5 Determining Noise produced from Controller

The SHFTS program [59] was designed to make preliminary estimations of the spectra which would be returned from the SPIRE instrument aboard the Herschel Space Observatory. It was important to model using known parameters of the instruments linear stage, including the spectrum of noise created by the PID controller. The linear stage aboard Herschel operated at a single velocity of $452\,\mu m \cdot s^{-1}$, with a travel distance of 3.5 cm.

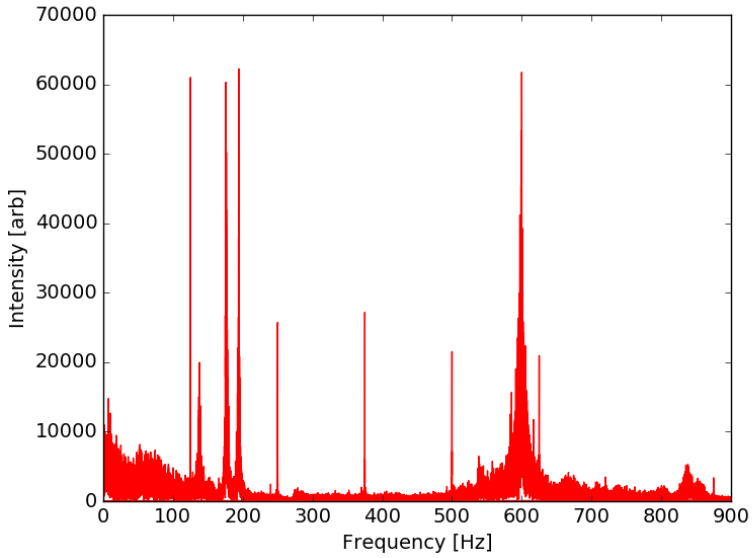


Figure 7.9: An Example of the Residual Velocity Spectra of the Aerotech linear stage. Performing the Fourier transform on the residual velocity data displays the frequency components of the noise in the signal. The linear stage was commanded to translate 100 mm at a speed of 500 $\mu m \cdot s^{-1}$.

In Figure 7.9, the key frequencies which were present for all velocities, forward and reverse, occurred at 137, 176, 194, 538 and 600 Hz. In contrast to the SPIRE instruments linear stage, the laboratory stage has a 45 cm travel and the ability to move at programmed velocities that range from sub-micron per second to one metre per second. The velocity is constrained by the DAQ systems being used, and likely will be kept under 1 $mm \cdot s^{-1}$, to maintain precision. The tests that have been performed were to verify that the DAQ devices that have been developed operate as desired, and more testing must be performed to provide a complete profile of the linear stage.

7.6 Conclusions

More effort needs to be spent if a Python/Linux version of the PID tuning program is deemed required. Currently, the optimal way to tune the linear stage, best done whenever the load changes, is by using the Ensemble software provided by Aerotech. After the initial changes to configuration and tuning, the Python library provided the necessary control to

command the linear stage for any desired positioning or motion.

Counting the HeNe laser fringes confirmed the motion of the linear stage within error. A comparison to the results from the Renishaw interferometer, including measurements taken with the Teensyduino 2.0++ micro-controller (TEENSY2) system, resulted in a correction to the Renishaw data by scaling with a factor of 1.0006155 corresponding to a double pass in the lab atmosphere ($n \approx 1.0003$). This was the equivalent of adjusting the resolution from 38.60 ± 0.05 pm to 38.6238 pm. With three measurements of position in agreement, the variation in the velocity was determined to be a consistent $\pm 7 \mu m \cdot s^{-1}$ for speeds between 300 to 700 $\mu m \cdot s^{-1}$.

Triggering the Renishaw interferometer with the PSO pulses, an estimate of the PSO accuracy was determined to be ± 1.2 nm, in Section 7.4. The source of error was not determined, most likely a combination of the Renishaw interferometer and the Aerotech stage, and further experimentation is required.

The spectrum of the residual velocity highlighted a series of frequencies that were consistent between all data sets. The three most intense features appear at 176, 194 and 600 Hz. Although further study is required to confirm these results, a profile of the PID controller noise can be used to simulate and predict the spectral/spatial results of the finished instrument.

Chapter 8

Concluding Remarks and Future Goals

"Wisdom comes from experience. Experience is often a result of lack of wisdom."

– Terry Pratchett

The clear progression of building upon previous versions of data acquisition devices has been illustrated throughout this thesis. Starting with understanding the intended operation of the instrument, in Chapter 2, and the precision required to verify that its components operated sufficiently, to two final devices, Chapters 5 and 6, which provided a greater level of accuracy and bandwidth than necessary.

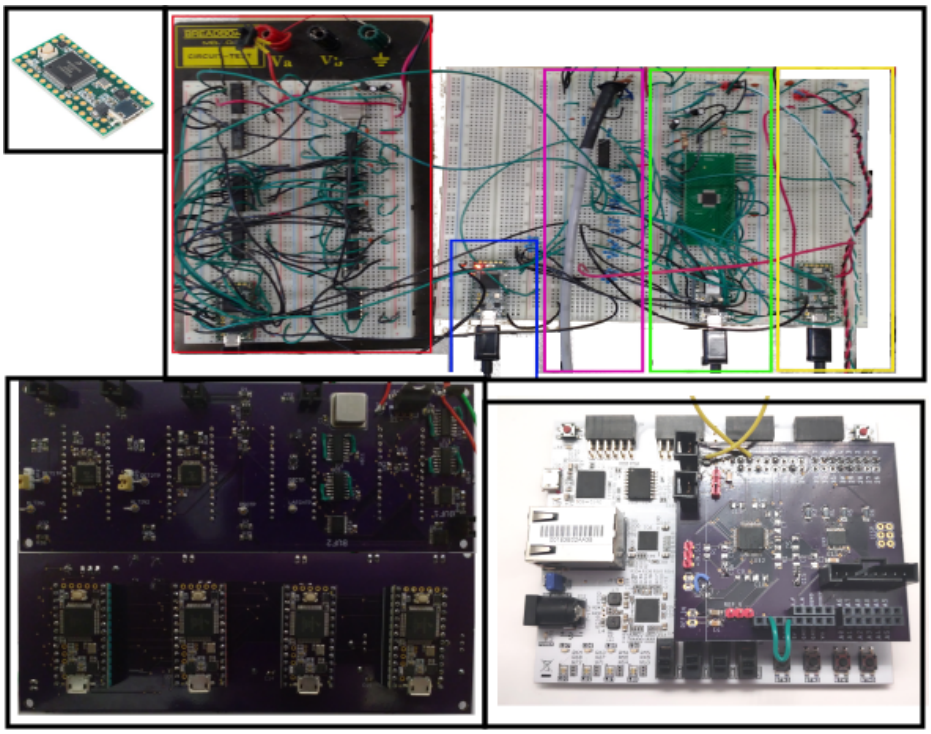


Figure 8.1: The upper-left shows the simple starting component, the TEENSY. The upper-right is the prototype circuit that used four TEENSY micro-controllers and a collection of discrete electronics. Lower-left shows both sides of the eventual fabricated printed circuit board (PCB) for the TEENSY DAQ. Finally, the lower-right depicts the PCB shield sitting atop of the Artix-7 FPGA Development Board (ARTY).

Figure 8.1 steps through the instrument development phases, producing three orders of magnitude increase in precision of timing and twenty times the bandwidth for data collection.

In Chapter 3, the Teensyduino 3.2 micro-controller (TEENSY) was found to be an adequate device for monitoring signals at sample rates less than 10 kHz with less than one percent error. Beyond this, the relative error started to increase due to the resolution and

inaccuracy of the microsecond clock. The on-board ADC is slow due to built-in averaging, which produced very precise results but limited the sampling rate. Serial communications caused further delays in the sampling frequency, and memory restrictions prevented storing data locally when the sample size was greater than ten thousand samples. For low-frequency control systems, that only require millisecond accuracy, the inexpensive cost is justified.

By taking the load off the micro-controller, utilizing the ability to read digital input/output pins, the performance of the TEENSY could be improved, as discussed in Chapter 4. Adding an external oscillator, up to 66 MHz, and a faster 16-bit ADC with parallel output, provided 120 kHz sampling capability. The construction of this instrument required four micro-controllers to host two ADCs, but could be built with three if only one ADC is required. The expense of approximately $200 is still lower than commercial systems (costing into the thousands of dollars), providing ± 15 nanosecond time resolution and fourteen effective bits of digital precision. The restriction of serial communications still placed an upper limit on performance.

The opportunity to rebuild the interface for the Renishaw interferometer, described in Chapter 5, provided a challenge to program in Verilog, System Verilog and VHDL. The Artix-7 FPGA Development Board (ARTY)'s network interface contributed to solving the serial communication constraint and allowed for 2 mega-samples per second (MSPS) bandwidth. The Renishaw interface saw over a one-hundred times improvement in the sampling rate and demonstrated the advantages of the higher clock frequency and parallel processing capabilities of a field programmable gate array (FPGA). The ARTY provided other advantages besides speed. The ability to code most of the "hardware" meant less expensive printed circuit board (PCB) design, requiring fewer components and smaller surface area.

Using the ARTY's extended abilities, it was shown that an FPGA could replace the four TEENSYs and most of the external electronics that was necessary in the discrete TEENSY DAQ. The decision to include a 2 MSPS 16-bit ADC was made, as the stock ADC on the ARTY was only 1 MSPS 12-bit ADC.

Preliminary data provided in Chapter 7, measuring the performance of the Aerotech linear stage, suggests that it is operating within specifications. The observed velocity varied by less than $\pm 2\%$ and the travelled distance agreed within error to that of the commanded displacement. Through testing of the linear stages, a slight systematic error was identified on the Renishaw interferometer, where measured displacements were consistently smaller than expected. It was found that a single scalar correction of 1.00062 to the Renishaw data made all observations agree. The correction factor was the equivalent of changing the resolution value, from 38.6 pm to 38.62 pm, which was within error if assumed that the error in the value provided was in the second decimal place.

8.1 Continuing and Future Work

During the development of the ARTY DAQ, more efficient code was implemented, utilizing a first in first out (FIFO) buffer. This provided improved memory management and data transfer to the UDP transmission module. The Renishaw FPGA code should be updated to this version. The Renishaw would also benefit from other improvements, such as time-stamps on each data sample and an identify routine to simplify serial connection.

Realizing that the position synchronized output (PSO) distances that were measured were below the distance that the Renishaw could accurately measure in air, a series of tests should be run to test larger PSO distances. Considering the wavelength for far-infrared (FIR) observations will be tens or possibly hundreds of times longer than the HeNe laser used during testing, PSO distances of tens of microns may be closer to actual operating requirements and reduce the relative error.

Continuing the development of the ARTY DAQ will require a new PCB to be fabricated and populated. The original multiplexer (MUX), which was never utilized, will be replaced with a dual analog 4-way switch, and better isolation precautions have been included, to decrease the noise observed on the ADC. The 4-way switch will allow four detectors to be observed, at the expense of bandwidth. The addition of an internal sample clock should be

considered as a complementary feature, reducing the number of required connections.

It was found during the testing of the linear stage that synchronizing the triggering of separate devices made the alignment of data sets easier, but the software contains delays turning off and on devices at different times. These delays required arbitrary offsets necessary to align data sets. A system to minimize this offset or a way to add a marker simultaneously to each data set would be a useful feature.

8.2 Summary

The TEENSY micro-controller provided a simple programming language, fast setup and reasonable results (within limits). The TEENSY was an inexpensive tool that fills basic needs for automation in the laboratory, and light data acquisition. When the task requires more than one micro-controller, or complicated circuit design, the ARTY FPGA was the superior choice. The designed devices made it possible to start investigating the operation of a key component in an astronomical testbed instrument.

References

[1] Mark H. Jones and Robert J. A. Lambourne, editors. *An Introduction to Galaxies and Cosmology*. Cambridge University Press, 2004.

[2] Eugene Hecht. *Optics (4th Edition)*. Addison-Wesley, 4th edition, 2001.

[3] Jim Quinn. Stargazing with Galileo: when Galileo Galilei first turned a telescope to the heavens four centuries ago, he discovered amazing things - and you can follow in his footsteps. *Nightsky(Cambridge, Mass.)*, 3:44, 2006.

[4] European Space Agency. *Herschel Observers' Manual*, herschel-hsc-doc-0876 edition, March 2014. `http://herschel.esac.esa.int/Docs/Herschel/html/Observatory.html`.

[5] ESO/Hubble (F.Granato). Transparency of the atmosphere. `https://www.eso.org/public/images/atm_opacity/`, May 2010.

[6] European Space Agency (ESA). Launch Vehicles. `http://www.esa.int/Our_Activities/Space_Transportation/Launch_vehichles/Ariane_5_Generic2`, October 2012.

[7] NASA. About Webb Innovations. `https://jwst.nasa.gov/mirrors.html#1a`.

[8] Albert A. Michelson and Edward W. Morley. LVIII. On the relative motion of the earth and the luminiferous aether. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 24(151):449–463, dec 1887.

[9] Albert A. Michelson. XXXVIII. On the application of interference-methods to spectroscopic measurements.—I. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 31(191):338–346, apr 1891.

[10] Pierre Connes. Early history of fourier transform spectroscopy. *Infrared Physics*, 24(2-3):69–93, may 1984.

[11] A. A. Michelson. Measurement of Jupiter's Satellites by Interference. *Publications of the Astronomical Society of the Pacific*, 3:274, sep 1891.

[12] Albert A. Michelson. I. On the application of interference methods to astronomical measurements. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 30(182):1–21, jul 1890.

[13] A. A. Michelson and F. G. Pease. Measurement of the diameter of alpha Orionis with the interferometer. *The Astrophysical Journal*, 53:249, may 1921.

[14] M. Ryle and D. D. Vonberg. Solar Radiation on 175 Mc./s. *Nature*, 158(4010):339–340, sep 1946.

[15] Antoine Labeyrie. Stellar interferometry methods. *Annual Review of Astronomy and Astrophysics*, 16(1):77–102, 1978.

[16] Peter Lawson. *Principles of Long-Baseline Stellar Interferometry*. Wiley-Interscience, 2016.

[17] European Space Agency (ESA). Technology Readiness Level (TRL), November 2017. http://sci.esa.int/sci-ft/50124-technology-readiness-level/.

[18] European Space Agency (ESA). L2, The Second Lagrangian Point. website, December 2010. https://www.esa.int/Our_Activities/Space_Science/Herschel/L2_the_second_Lagrangian_Point.

[19] William F. Grainger and Roser Juanola-Parramon and Peter A. R. Ade and Matt Griffin and Flo Liggins and Enzo Pascale and Giorgio Savini and Bruce Swinyard. A Demonstration of Spectral and Spatial Interferometry at THz Frequencies, 2012.

[20] David T. Leisawitz and Brad J. Frey and Douglas B. Leviton and Anthony J. Martino and William L. Maynard and Lee G. Mundy and Stephen A. Rinehart and Stacy H. Teng and Xiaolei Zhang. Wide-field imaging interferometry testbed I: purpose, testbed design, data, and synthesis algorithms. In *Interferometry in Space*. SPIE, feb 2003.

[21] Ludwig Mach. Ueber einen Interferenzrefraktor. *Zeitschrift fÃijr Instrumentenkunde*, 12:89–93, 1892. https://archive.org/details/zeitschriftfrin14gergoog/page/n3.

[22] Ludwig Zehnder. Ein neuer Interferenze Refraktor. *Zeitschrift fÃijr Instrumentenkunde*, 11:275–285, 1891. https://archive.org/details/zeitschriftfrin14gergoog/page/n14.

[23] Barry C. Barish and Rainer Weiss. LIGO and the Detection of Gravitational Waves. *Physics Today*, 52(10):44–50, oct 1999.

[24] Locke D. Spencer. Spectral Characterization of the Herschel SPIRE Photometer. Master's thesis, University of Lethbridge, 2005.

[25] Daniel Mader. Effect of a sample on the phase of the output beams in a Mach-Zehnder interferometer, October 2005. https://en.wikipedia.org/wiki/Mach%E2%80%93Zehnder_interferometer.

[26] R. Chevalerias, Y. Latron, and C. Veret. Methods of Interferometry Applied to the Visualization of Flows in Wind Tunnels. *Journal of the Optical Society of America*, 47(8):703, aug 1957.

[27] P. Hariharan. *Basics of Interferometry*. ACADEMIC PR INC, 2006.

[28] J. G. Ziegler and N. B. Nichols. Optimum Settings for Automatic Controllers. *Transactions of the American Society of Mechanical Engineers*, 1942.

[29] Bjorn D. Tyreus and William L. Luyben. Tuning PI controllers for integrator/dead time processes. *Industrial & Engineering Chemistry Research*, 31(11):2625–2628, nov 1992.

[30] Aerotech, Inc. *Ensemble Software Help Files*, 2019. http://aerotechmotioncontrol.com/manuals/index.aspx.

[31] H. Nyquist. Certain Topics in Telegraph Transmission Theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, apr 1928.

[32] Józef Kalisz. Review of methods for time interval measurements with picosecond resolution. *Metrologia*, 41(1):17–32, dec 2003.

[33] Silicon Labs Inc., 400 West Cesar Chavez Austin, TX 78701. *Improving ADC Resolution by Oversampling and Averaging*, rev. 1.3 edition, July 2013.

[34] I. Susnea and M. Mitescu. *Microcontrollers In Practice*. Springer, 2005.

[35] David Kushner. The Making of Arduino. website, October 2011. https://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino.

[36] Agilent Technologies, 1900 Garden of the Gods Road Colorado Springs CO 80907. *Agilent InfiniiVision 2000 X-Series Oscilloscopes User's Guide*, third edition, June 2011.

[37] Freescale Semiconductor, Inc. *K20 SubFamily Reference Manual*, rev. 1.1 edition, December 2012. https://www.pjrc.com/teensy/K20P64M72SF1RM.pdf.

[38] Maxim Integrated. Determining Clock Accuracy Requirements for UART Communications. Technical report, Maxim Integrated 160 Rio Robles San Jose, CA 95134 USA, 2003. https://pdfserv.maximintegrated.com/en/an/AN2141.pdf.

[39] www.pjrc.com. Using the Hardware Serial Ports. Website. https://www.pjrc.com/teensy/td_uart.html.

[40] Peggy Liska. Using interleaving with SAR ADCs for lower power, smaller size and lower cost. *Analog Design Journal*, 4Q 2017, 2017. http://www.ti.com/lit/an/slyt731/slyt731.pdf.

[41] Burr-Brown Products from Texas Instruments. *16-BIT, 750-kHz, Unipolar Input, Micro Power Sampling Analog-To-Digital Converter with Parallel Interface*, rev. b edition, February 2005.

[42] Texas Insturments. *SNx4HC273 Octal D-Type Flip-Flops with Clear Data Sheet*, 2016.

[43] Abracon LLC. Half Size DIP Low voltage 3.3 VHCMOS/TTL Compatible Crystal Clock Oscillator. Website, January 2016. https://abracon.com/Oscillators/ACHL.pdf.

[44] Nexperia. *74HC273; 74HCT273 Octal D-type filp-flop with reset; positive-edge trigger*, rev. 5 edition, February 2016. http://www.nexperia.com.

[45] Texas Instruments. CD4020B, CD4024B, CD4040B Types CMOS Ripple-Carry Binary Counter/Dividers. http://www.ti.com/product/CD4040B, December 2003.

[46] Justin Roark and Scott C. Smith. Demonstration of the Benefits of Asynchronous vs. Synchronous Circuits. In *Midwest Section Meeting Achrives*, 1818 N Street N.W. Suite 600, Washington DC 20036, September 2012. American Society for Engineering Education (ASEE). https://www.asee.org/papers-and-publications/papers/section-proceedings/midwest/2012.

[47] A. Maestrini and C. Tripon-Canseliet J. Gill G. Chattopadhyay E. Schlecht I. Mehdi J.S. Ward, H. Javadi. Local oscillator chain for 1.55 to 1.75THz with 100-/spl mu/W peak power. *IEEE Microwave and Wireless Components Letters*, 15(12):871–873, dec 2005.

[48] Aerotech, Inc., 101 Zeta Drive, Pittsburgh, PA, 15238. *ALS20000 / ALS25000 Series Stage User's Manual*, revision 1.03.00 edition, April 2011.

[49] Renishaw Public Limited Company (PLC). *RPI20 parallel interface*, m-9904-2254-06-a edition.

[50] Digilent, 1300 Henley Court, Pullman, WA 99163. *Arty FPGA Board Reference Manual*, arty rev. c. edition, June 2017. http:\www.digilentinc.com.

[51] Renishaw Public Limited Company (PLC). *RLE Fibre Optic Laser Encoder Installation Guide*, m-5225-0568-05-b edition.

[52] James Bowman. Making an arbitrary frequency clock in VHDL and Verilog. website. https://excamera.com/sphinx/vhdl-clock.html.

[53] Sebastien Bourdeauducq. RS232 Uart. Website, August 2010. https://opencores.org/projects/mmuart.

[54] Mike Field. ArtyEtherentTX. GitHub, Inc. (US), June 2016. https://github.com/hamsternz/ArtyEtherentTX.

[55] Xilinx, Inc., 2100 Logic Drive San Jose, CA 95124 U.S.A. *FIFO Generator v13.1*, April 2017. https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v13_1/pg057-fifo-generator.pdf.

[56] Thorlabs Inc. SM05PD1B Mounted Si Photodiode Spec Sheet. https://www.thorlabs.com/thorproduct.cfm?partnumber=SM05PD1B.

[57] Texas Instruments Inc. *16-Bit, 2 MSPS, Unipolar Input, Micro Power Sampling, Analog-To-Digital Converter with Parallel Interface and Reference*, December 2004. http://www.ti.com/lit/ds/slas369b/slas369b.pdf.

[58] L. Mertz. *Transformations in Optics*. John Wiley & Sons Inc, 1965.

[59] John V. Lindner. SHIFTS: Simulator for the Herschel Imaging Fourier Transform Spectrometer. Master's thesis, University of Lethbridge, 2006.

[60] Timothy Sauer. *Numerical Analysis, 2nd Edition*. Pearson, 2011.

[61] 1001 Murphy Ranch Road Renesas Electronics America Inc. and C. A. 95035 Milpitas. ±16.5kV ESD Protected, +125°C, 3.0V to 5.5V, SOT-23/TDFN Packaged, 20Mbps, Full Fail-safe, Low Power, RS-485/RS-422 Receivers, July 2015.

Appendix A

Applied Mathematical Methods

A.1 Methods of Numerical Differentiation

The metrology systems that have been developed provide accurate measurements of position and time of the linear stage being used in the interferometer. It is assumed that if the stage is working to specification, then a commanded velocity is constant. To verify this, and measure the effect of the PID controller on the velocity, it is necessary to take the derivative of the position with respect to time, which can only be done numerically as no exact function of position exists and most likely would be complicated to account for all parameters affecting stage motion.

Let us consider the approximation for the velocity occurring at the position $P(t_i)$ by evaluating the Taylor series at the time t_i :

$$P(t_{i+1}) = P(t_i) + (t_{i+1} - ti)P'(t_i) + \frac{(t_{i+1} - t_i)^2}{2}P''(t_i) + \ldots, \tag{A.1}$$

$$P'(t_i) = \frac{P(t_{i+1}) - P(t_i))}{t_{i+1} - t_i} - \frac{t_{i+1} - t_i}{2}P''(t_i) + \ldots. \tag{A.2}$$

Changing variable P' to v for velocity, and taking the first term approximation, the equation above becomes

$$v_i = \frac{P_{i+1} - P_i}{t_{i+1} - t_i} + O(\Delta t), \tag{A.3}$$

where $\Delta t = t_{i+1} - t_i$. The second term represents the error in the calculated velocity [60] in big O notation, meaning that the error is on the same scale as the time period.

A.1.1 3 Point Differentiation

The function representing the position of the linear stage must be continuous and smooth, as the position cannot instantaneously appear at a new location without passing through all possible locations in between. The same can be said about the signal received by the photo-detector, as fringes vary dependent on the phase difference of the two light beams. For a continuous and smooth function, the central difference approximation (CDA) can provide a superior approximation of the derivative, where statistically the derivative can be approximated as a normally distributed probability between symmetrically spaced values on either side of the desired point. Therefore, the first derivative of a position and time P_i,

t_i can be estimated by using points P_{i+1}, t_{i+1} and P_{i-1}, t_{i-1}, respectively such that,

$$v_i = \frac{P_{i+1} - P_{i-1}}{2\Delta t} + O(\Delta t^2), \tag{A.4}$$

where Δt is the evenly spaced period between samples of position. The error in the measurement has been reduced to the order of the time period squared. This formula can be derived by subtracting the two Taylor polynomials for P_{i+1} and P_{i-1}. Three-point numerical derivative provides an advantage, decreasing error, only while Δt is larger than the cube root of the error present in the difference of the positions P_{i+1} and P_{i-1} [60].

A.1.2 5 Point Differentiation

Continuing in the same fashion as equation A.4, five points can be utilized to reduce error to the order of Δt^4. Also called a five-point central difference theorem[60], the five point approximation of the derivative at point P_i can be calculated by

$$v_i = \frac{P_{i-2} - 12P_{i-1} + 12P_{i+1} - P_{i+2}}{12\Delta t} + O(\Delta t^4). \tag{A.5}$$

This process can be repeated for higher and higher orders, with greater accuracy, but sacrificing the end values as points that were not measured are required to calculate them. Assuming a wider time region in the points used, the effect is to smooth out the data. Lower order formulas could be used to estimate the endpoints. Higher order differentiation formulas tend to be more accurate for larger Δt, as the rounding error is inversely proportional to Δt^4.

A.2 Two's Complement Binary Numbers

An unsigned 8-bit number can hold the integer values zero through to two hundred and fifty-five. This does not allow for representing negative values, thus ones and two's complement representations were developed. In ones complement zero can be represented as -0 (11111111) and +0 (00000000). The advantage of two's complement representation is that it only has one value for zero and does not require any carry values in order to perform addition or subtraction operations. An 8-bit two's complement value can hold the integer values -128 to +127.

Let N be the total number of bits in a two's complement integer. Indexing of the binary bits generally starts at zero; Let i be the arbitrary index value, where $0 \leq i \leq (N-1)$, and $(N-1)$ being the sign bit. This convention allows the weighting of each bit to carry the value of 2^i, hence the least significant bit (LSB) holds the value of 0 or 1. The most significant bit (MSB), in a two's complement represented value, is the sign bit which carries the weight of -2^{N-1}.

Conversion to integer values is simply the addition of the weighted values of the bit positions which contain a 1, ignoring any 0 values. The values for an 8-bit number between 0 and 127 retain their unsigned binary representation. When the sign bit is 1, an 8-bit number would have the equivalent of the 7-bit unsigned value less 128.

Example A.1. Given the unsigned 8-bit value b10110101, which is equivalent to the unsigned integer 181. To find the two's complement value :

Let N = 8, then find the unsigned integer value of the lower seven bits. Thus, b0110101 is 53.

As the eighth bit is 1, subtract $2^7 = 128$ from the value of the lower seven bits. The end result is -75.

A second method of converting two's complement values involves inverting all the bits, swapping zeros for ones and vice versa, and then adding one. With this method, a negative sign must be added if the MSB is a one.

Example A.2. Given the unsigned 8-bit value b10110101, which is the equivalent to the unsigned integer 181. To find the two's complement value :

First, invert all the bits, then add one. The inverted binary version of 181 is b01001010. Adding 1 inverts the LSB. The result is b01001011 which has the integer value of 75. Finally, add a negative sign as the original binary value had a one in the MSB. The end result is -75.

The value agrees with the previous method, in Ex. A.1.

The Renishaw interferometer delivers a 36-bit two's complement word value, representing the integer multiple of the resolution (preset on the dip switches located on the RPI20 parallel interface card). Values can be negative or positive depending on the displacement from the homed position upto one metre [49]. The FPGA delivers the position information, through the serial interface for a single reading, and through the network interface for continuous data collection, in 5-byte portions. The four MSBs should all be zeros and can be ignored as the sign bit is the 36th bit.

A.2.1 Cosine Error

If the alignment is slightly off, a cosine error factor will be introduced, and the measured length of stage travel will be smaller than the actual displacement.

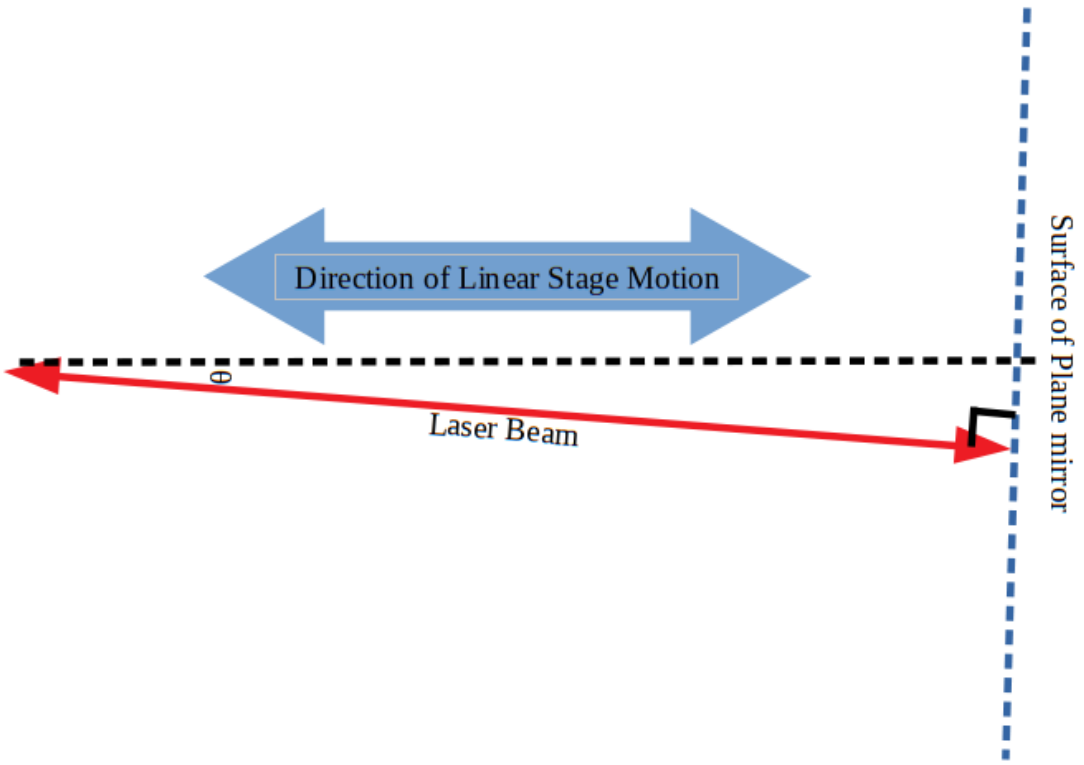


Figure A.1: Cosine error is created by a small angle between the vector of motion and the vector of the measuring instrument that will result in a small reduction in the actual distance measured.

In Figure A.1 the angle θ is exaggerated to demonstrate the effect of cosine error. In reality the error introduced by a 1° misalignment would result in less than 0.016% change of measurement. This seems negligible, but over 100 mm of travel would result in 15.3 μm of lost travel. If counting fringes on a Michelson interferometer, this difference would be comparable to forty-eight fewer fringes.

Appendix B

Teensy 3.2 with Discrete Electronics

The Teensyduino 3.2 micro-controller (TEENSY) DAQ (profiling results in Chapter 4) requires some basic set up before it can be used. Proper wiring connections are outlined in Section B.1.1 and the functionality of the Python library is described in Section B.1.2. Schematic diagrams of the electronics are provided in Section B.3. Following, the Arduino code for the "Command", "Stopwatch", and discrete ADCs is given in Sections B.4, B.5 & B.6.

B.1 Setup and Configuration

B.1.1 Wiring

A power supply, set to provide 6 V and 0.8 A, is required to power the TEENSY DAQ, first described in Chapter 4. The red/green wires soldered directly to the board, depicted in Figure B.1, provide the input to the power supply.

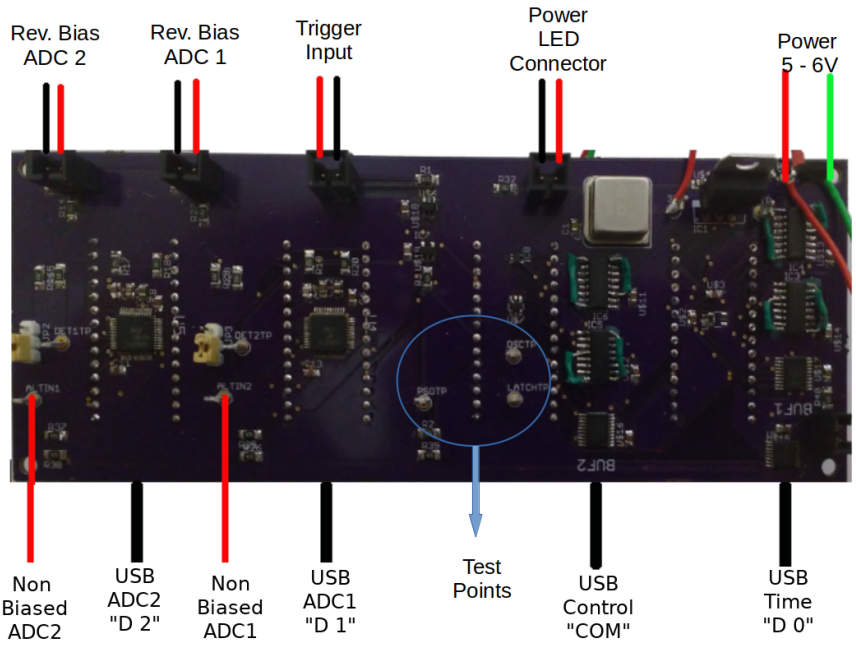


Figure B.1: The top side of the discrete TEENSY DAQ PCB, labelled for proper wiring connection.

The reverse bias ADC inputs, top-left of Figure B.1, were designed for a Thorlabs photo diode[56] and included a voltage divider to reduce the maximum signal to 3.3V tolerance for the ADS8371 ADC. The jumper, found directly above the non-biased input point, allows selection of which input is connected to the ADC. If the non-biased input is used, the signal should be between 3.3 V reference voltage and ground.

The trigger input, top-middle of Figure B.1, passes the trigger signal through an RS-485/RS-422 receiver[61]. This converts the differential Aerotech PSO output into a single sided (TTL) trigger signal. A function generator can also be used to send a trigger signal without any modification.

Test points, blue circle in middle of Figure B.1, provide the ability to monitor the PSO input, the latching signal, and the 66 MHz oscillator output. There are also test points adjacent to the bias/non-bias jumper to monitor the incoming ADC signal.

B.1.2 Calling the TEENSY DAQ in Python

Before a TEENSY object can be called, the serial ports must be identified and the identity of each micro-controller must be logged to track function. Although the discrete board contains four micro-controllers, additional units could be added externally. There can exist only one "command" TEENSY, identified with the "COM" label. All other TEENSYs are "detector" TEENSYs and are labelled with a captial 'D', followed by a space and then a unique number. At least one detector must be additionally connected along with the command TEENSY

```
1 import portsId
2 import TeensyRecorder
3
4 ports, Id = portsId.identify_ports()
5 tdaq = TeensyRecorder.teensy(ports)
6 tdaq.setSaveDir("\home\user\data\")
7 tdaq.setFilename("A_meaningful_description", ports, Id)
```

Figure B.2: Code snippet to identify the connected TEENSYs. textttports is an array of serial ports, and `Id` is the corresponding unique identifiers which are utilized to name the function of the detectors.

After calling the identify routine, line 3 in Figure B.2, the location to save the data files is set, and unique file names are created using a list of detector names contained in the *TeensyRecorder.py* library. If different names are required for the detectors, edit the following line in the *TeensyRecorder.py* library.

```
self.detectorname = {0: 'CLOCK', 1: 'ADC1', 2: 'ADC2', 3: '', 4: '',
    5: '', 6: '',7: '', 8:'', 9: ''}
```

Make sure that the number corresponds to the correct detector and that the name is unique.

B.2 Teensy Recorder Commands

Command	Function
`setSaveDir(<directory>)`	Creates/sets the directory for save data files
`setFilename(<description>,<ports>,<Id>)`	Sets filename for each identified detector. The <ports> and <Id> are arrays returned from the identify ports routine, which is executed before making a teensy object.
`start()`	Creates an individual recording thread for all identified detectors and starts listening for incoming data.
`stop()`	Stops recording threads and closes data files.

B.3 Teensy DAQ with Discrete Electronics Schematics

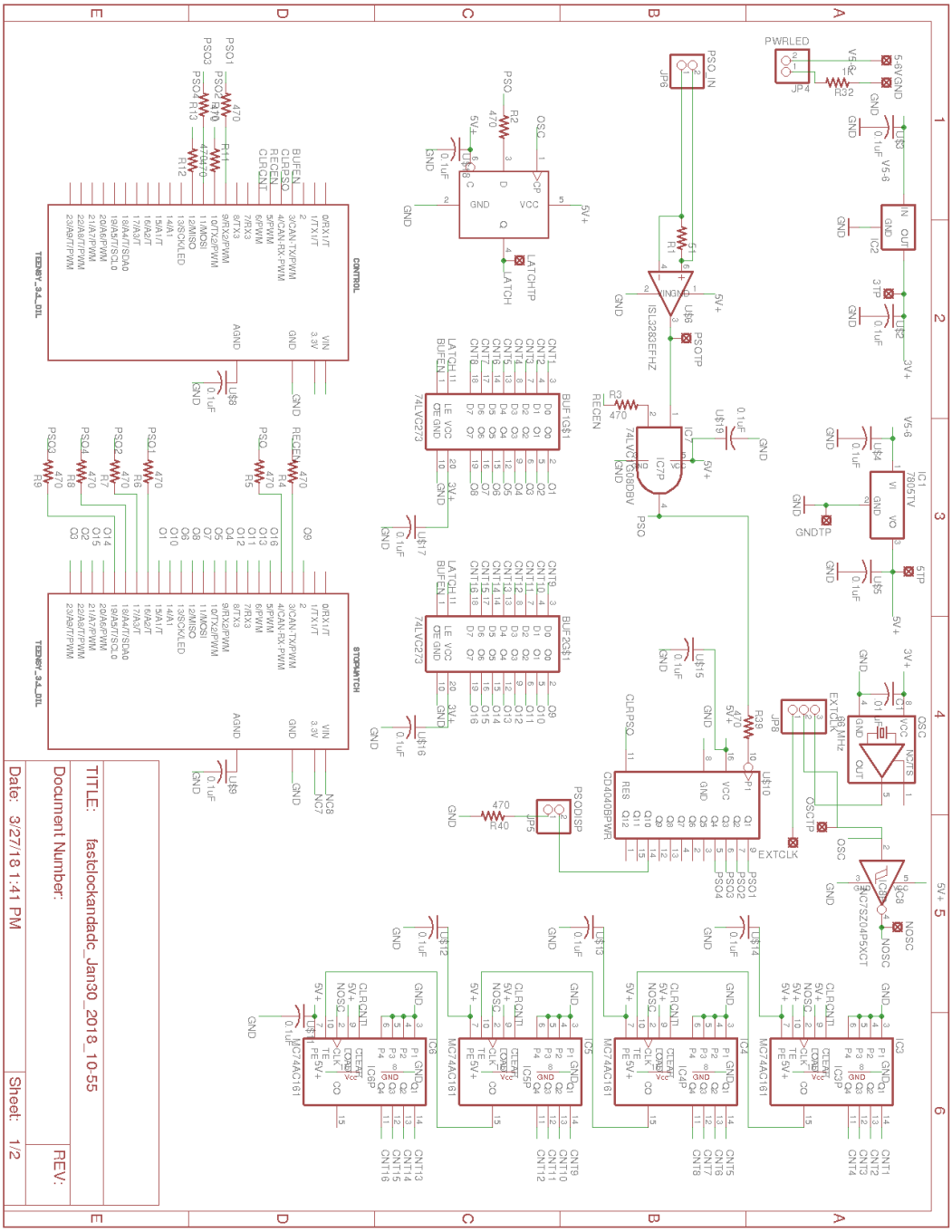


Figure B.3: Schematic of TEENSY DAQ circuit. Sections A1-A3 contain the 6V power input and 5.0 V and 3.3 V regulators for powering the other devices. A4-A5 contain the 66 MHz oscillator and inverter. B1-B5 depicts the differential input and PSO counting circuit. A6-D6 is the synchronous counting circuit, which increments on the falling edge of the oscillator. C1-C4 has the latching circuit which aligns the PSO pulse with the rising edge of the oscillator and the buffer circuit which holds the current value on the synchronous counter. At the bottom are the control and clock teensyduino micro-controllers.

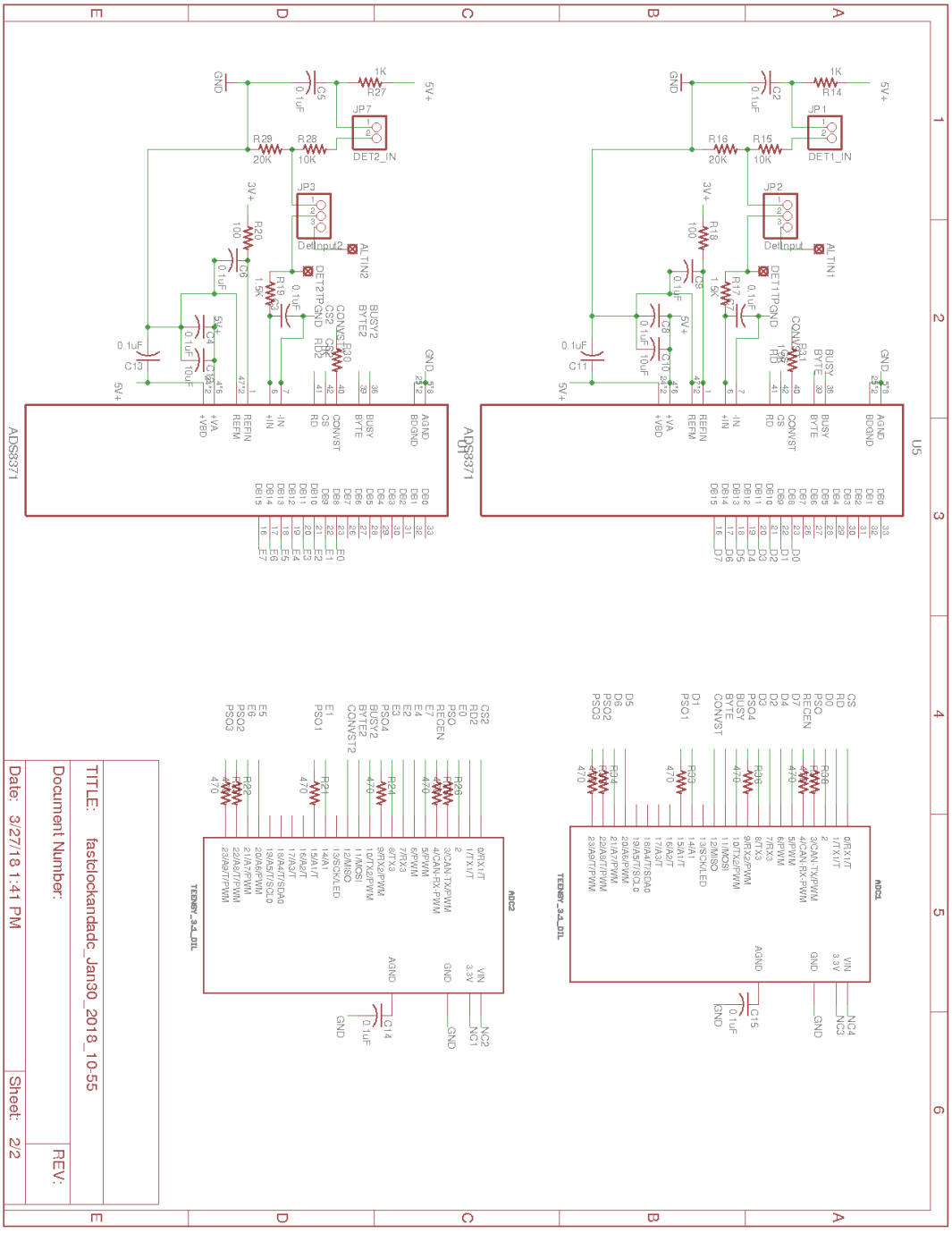


Figure B.4: Schematic for ADS8371 ADC to TEENSY. The top and bottom depict the same circuit which connects the Texas Instruments ADS 8371 parallel ADC to its respective Teensyduino for recording value an external voltage source.

B.4 Command Micro-Controller Arduino Code

The Arduino code used to program the controlling TEENSY for the discrete DAQ circuit.

```
//define control logic
#define rec_en 5 //Enable/Disable recording output
#define clkrst 6  //reset clock counter pin
#define psorst 4  //reset pso counter pin
#define bufclr 3  //reset clear buffer status
#define bit1  9 //pso counter bit 1
#define bit2  10 //pso counter bit 2
#define bit3  11 //pso counter bit 3
#define bit4  12 //pso counter bit 4

//count overflows of trigger counter
volatile long overflow_counter;

void overflow_ISR(){ //Interrupt Service Routine - Count Trigger Overflows
overflow_counter++;}

int read_pso(){
int result =0;
for( int i=0; i<4; i++){
if (digitalRead(i+9)) result = result | (1<<i);
}
return result;}

void M_reset(){ //toggle reset pins
digitalWrite(clkrst, HIGH);
digitalWrite(bufclr, LOW);
digitalWrite(psorst, LOW);
delayMicroseconds(1);
digitalWrite(clkrst, LOW);
digitalWrite(bufclr, HIGH);
digitalWrite(psorst, HIGH);}

void setup() {
//define control logic
pinMode(rec_en, OUTPUT);
pinMode(clkrst, OUTPUT);
pinMode(psorst, OUTPUT);
pinMode(bufclr, OUTPUT);
pinMode(bit1, INPUT_PULLDOWN);
pinMode(bit2, INPUT_PULLDOWN);
pinMode(bit3, INPUT_PULLDOWN);
pinMode(bit4, INPUT_PULLDOWN);
digitalWrite(rec_en, LOW); //start with rec_en off
M_reset();
Serial.begin(115200);}
```

```
void loop() {
if (Serial.available() > 0) {
char c = Serial.read();
if (c == 'P') {
//M_reset();
noInterrupts();
overflow_counter=0;
Serial.println(read_pso());
interrupts();
attachInterrupt(digitalPinToInterrupt(bit4), overflow_ISR, FALLING);
digitalWrite(rec_en, HIGH);
while(true){//wait for recording to end
if (Serial.available()>0){ //any serial breaks recording
char c = Serial.read();
digitalWrite(rec_en, LOW);
detachInterrupt(digitalPinToInterrupt(bit4));
noInterrupts();
Serial.println(overflow_counter);
interrupts();
Serial.println(read_pso());
break;
}
}
} else if(c=='I') Serial.write("COM"); //Send Identify String
}
}
```

B.5 Time Micro-Controller Arduino Code

The TEENSY which records the time-stamp of trigger pulses was programmed with the following code.

```
//define control logic
#define PSO  4  //PSO trigger pin
#define rec_en 3  //turn recording on
#define nop  __asm__("nop\n\t") //null operation - shortest delay
#define bufsz  3 //number of bytes to transmit
//Setup Port Inputs on Teensy 3.2
//D-Byte (2,14,7,8,6,20,21,5) A-Byte (15,22,23,9,10,13,11,12)
uint8_t buf[bufsz]; // transmit buffer for data
byte pinTable[]={15,22,23,9,10,13,11,12,2,14,7,8,6,20,21,5,16,17,19,18};
uint8_t state;
uint8_t p;

void setup() {
pinMode(PSO, INPUT_PULLDOWN);
state = 0;
for (int i = 0; i < 20; i++) pinMode(pinTable[i] , INPUT_PULLUP);
pinMode(rec_en, INPUT_PULLDOWN);
Serial.begin(115000);}

void loop() {
if (Serial.available() > 0) {
char c = Serial.read();
if (c == 'I') Serial.write("D 0"); //Send Identify String
}
if(digitalReadFast(rec_en)){
uint16_t saveSREG=SREG; //save state of registers
cli(); //turn off interrupts
while (digitalReadFast(rec_en)) {
//wait for PSO to go HIGH
if ((digitalReadFast(PSO)) & (state==0)){
delayMicroseconds(1);
buf[0]=GPIOB_PDIR & 0x0F; //4bit trigger counter
buf[2]=GPIOD_PDIR & 0xFF; //MSB of counter
buf[1]=GPIOC_PDIR & 0xFF;  //LSB of counter
Serial.write(buf,3);
state = 1; //track rising edge of PSO
}
if ((!digitalReadFast(PSO)) & (state==1)) state = 0;
}
SREG = saveSREG; //turn interrupts back on
}
}
```

B.6 ADC Micro-Controller Arduino Code

The discretet ADS8371 ADC was controlled by the following code, including the collection and transmission of data.

```
//define control logic
#define BUSY  10 //used for interrupt on falling edge
#define BYT   11  //HIGH - read bits 0:7, LOW- read bits 8:15
#define CONVST 12  //Falling edge ends aquisition
#define PSO    3 //Trigger for acquiring data (Position Sync Output)
#define CS     0 //Chip select on ADS8371 ADC
#define RD     1 //Read on ADS8371 - active low
#define REC_EN   4  //Recording enabled on control micro-controller
#define nop  __asm__("nop\n\t") //null operation - shortest delay
//Setup Port Inputs on Teensy 3.2
//D-Byte (2,14,7,8,6,20,21,5) A-Byte (15,22,23,9)
byte pins[] = {2,14,7,8,6,20,21,5, 15,22,23,9};
uint8_t buf[3];    //storage for transmitting adc bytes
int prev, curr;    //track rising edge of trigger

void setup() {  //run once after power on
//set pin operation
pinMode(BUSY, INPUT);
pinMode(BYT, OUTPUT);
pinMode(CONVST, OUTPUT);
pinMode(PSO, INPUT);
pinMode(RD, OUTPUT);
pinMode(REC_EN, INPUT);
for(int i=0; i<12; i++){pinMode(pins[i],INPUT_PULLDOWN);}
//initial setup
digitalWrite(CONVST, HIGH); //active low
digitalWrite(RD, HIGH);  //active low
//LOW reads most significant byte[15:8]
//HIGH reads least significant byte[7:0]
digitalWrite(BYT, LOW);
Serial.begin(115200);}

void loop(){
if (Serial.available() > 0) {
char c = Serial.read();
if (c == 'I') Serial.write("D 2"); //Send Identify String
}
if(digitalReadFast(REC_EN)){
noInterrupts();
while(digitalReadFast(REC_EN)){
//IF PSO HIGH take sample
prev=curr;
curr = digitalReadFast(PSO);
if ((prev== LOW) && (curr == HIGH)){
```

```
digitalWrite(CS, LOW);
nop;
digitalWrite(CONVST, LOW);
nop;
nop;
digitalWrite(CONVST,HIGH);
digitalWrite(CS, HIGH);
//wait for busy to be done
while(digitalReadFast(BUSY)==HIGH);
//read data bytes
digitalWriteFast(CS, LOW);
digitalWriteFast(RD, LOW);
//PSO counter
buf[0]= GPIOC_PDIR & 0x0F;
//second byte
buf[2]= GPIOD_PDIR & 0xFF;
//first byte
nop;
digitalWriteFast(BYT, HIGH);
for(int i=0; i<17; i++) nop;
buf[1]=GPIOD_PDIR & 0xFF;
//reset to recieve next pulse
nop;
digitalWrite(RD, HIGH);
digitalWrite(BYT, LOW);
digitalWrite(CS, HIGH);
Serial.write(buf,3); //transmit data
}
}//while(REC_EN)
interrupts();}
}//END LOOP
```

Appendix C

Renishaw Interferometer

The updated Renishaw Interferometer interface, utilizing the Artix-7 FPGA Development Board (ARTY), has many updated features and abilities compared to the first version created with two Teensyduino 2.0++ microcontrollers (TEENSY2s). A description of the development and testing is provided in Chapter 5. The ASCII command menu is described in Section C.1. The LEDs provided on the ARTY were used as indicators for operation and error warnings. The LED locations and functions are described in Section C.1.1. Table C.1.2 describes the slide switch positions, required to divide the sampling frequency such that the LED display works correctly. A schematic version of the two TEENSY2 interface board is supplied in Section C.2 and for comparison, the ARTY Renishaw interface board schematic in Section C.3. Finally, the conversion routine, to convert received packets to useful position data, is given in Section C.11.

C.1 ASCII Commands

CHR	Function	Response
I	Set destination IP address for the UDP packet receiving computer. Must be followed by 4 bytes representing the IP address.	*
i	Return destination IP address for the UDP packet receiving computer.	4 bytes representing IP followed by an *
J	Set source IP address for the FPGA (must be an nonexistant address on the network) Must be followed by 4 bytes representing the IP address.	*
j	Return source IP address for the FPGA.	4 bytes representing IP followed by an *
M	Set MAC address of UDP packet receiving computer. This is followed by 6 bytes representing the physical address of the data collecting computer.	*
m	Return MAC address of UDP packet receiving computer.	6 bytes representing the MAC address
N	Use external sample clock	*
n	Use internal sample clock (default)	*

F	Set sampling frequency for data collection. Followed by 3 bytes that composes the integer frequency of the sampling clock.	*
f	Return sampling frequency.	3 bytes representing sampling rate followed by an *
P	Start continuous sampling of data	* - successful % - not started due to error state
Q	Stop continuous sampling of data	*
R	Reset RPI20 parallel interface card	*
S	Return STATUS information from RPI20 parallel interface card.	5 bytes of STATUS information followed by an *
T	Return TEST† pattern from RPI20 parallel interface card.	5 bytes of TEST pattern followed by an *
1	Return a single position reading.	5 byte position value followed by an *

† A description of the TEST and STATUS results can be found in the Renishaw RPI20 parallel interface users manual.

After a command to set an IP address (I or J), 4 bytes must be sent representing the the 4 octets. Example : Setting the destination IP of 192.168.1.7.

The 4 octets of the IP are always values less than 255 and therefore are single byte numbers. Since values for 1 and 7 cannot be representing by an ASCII character the string will be represented in hexadecimal (HEX) notation. 'I' in represented with a hexadecimal value is '0x49', 192 is '0xC0', 168 is '0xA8', and so on. Thus the entire string to set the destination IP would appear as : '0x49 0xC0 0xA8 0x01 0x07'

The process is similar for setting the MAC address, except 6 bytes will follow in the same order as read from the physical address of the network adapter.

The frequency setting is followed by 3 bytes which represent the integer frequency for the sampling clock. Example: Setting the sampling clock frequency to 750 kHz.

The number 750000 converted to HEX is 0x0B71B0. Since 'F' is 0x46, the entire frequency setting string would appear as : '0x46 0x0B 0x71 0xB0'

C.1.1 LED Display and Error Warnings

The ARTY development board provides four tri color indicator lights and a separate four monochrome green LEDs.

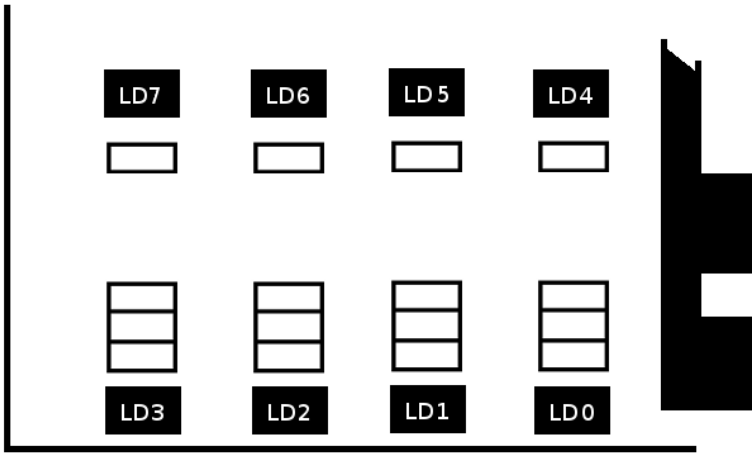


Figure C.1: The position of the provided LED display on the ARTY development board.

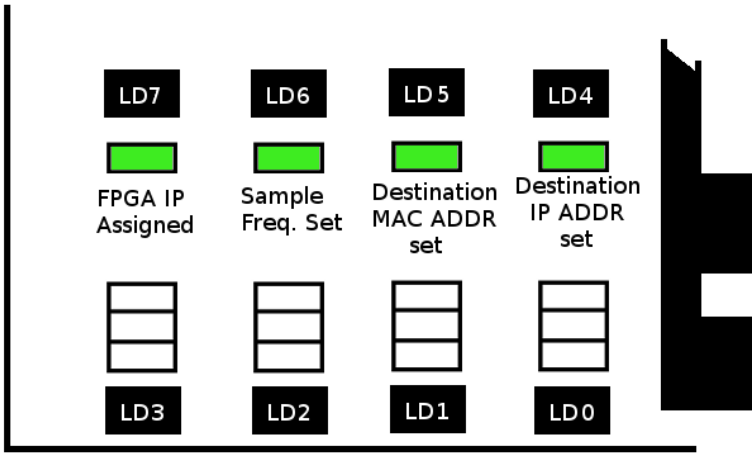


Figure C.2: LED Indicators for parameter settings on the Renishaw ARTY interface. The four monochrome LED indicators provide verification that the required configuration settings have been recieved and set. The light being on doesnt not signify that the setting is correct and should be checked by requesting the setting back through the serial connection.

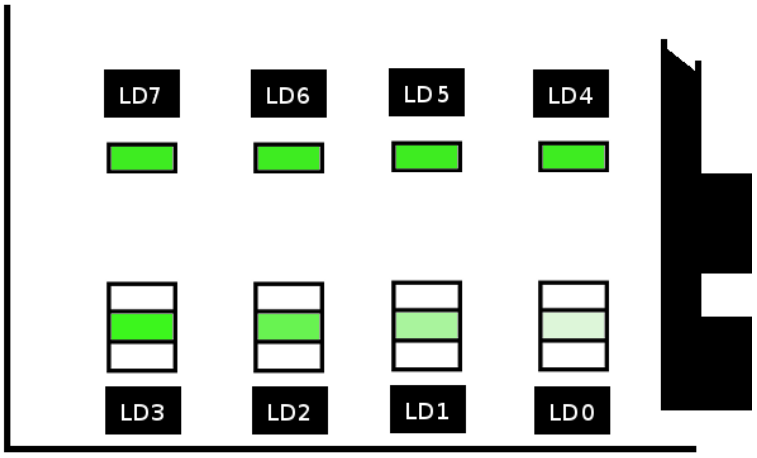


Figure C.3: LED Display to Indicate when Renishaw ARTY Interface is Sampling. When sampling has started the green LED in the tri color indicators will cycle. If all four leds appear to be on or slightly dimmed, adjust the divider settings with four slide switches.

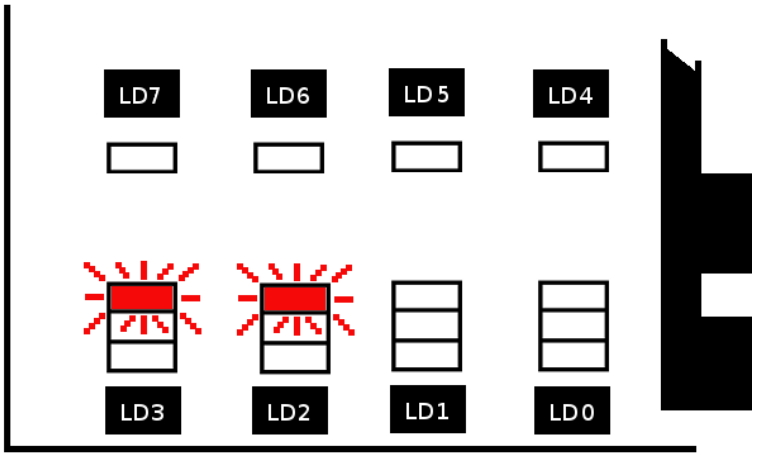


Figure C.4: LED Indicator for Break Beam Error on the Renishaw ARTY Interface. If the Renishaw experiences low signal or the beam is broken, the left two red indicators will flash untill the ARTY is reset and the problem has been corrected.

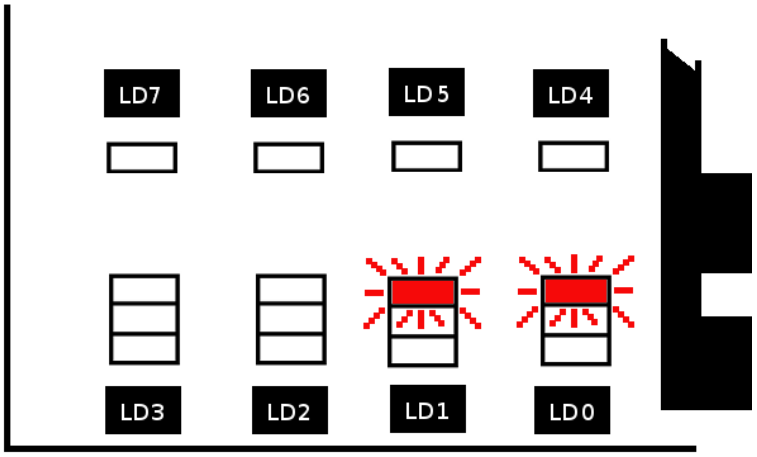


Figure C.5: LED Error State for missed Sample on the Renishaw ARTY Interface. After each position is read, a verification pulse is required to ensure that no samples are missed. Upon missing a sample the right two red LEDs will flash. This often occurs upon ending a session at high sampling rates, but should only be a concern if occurring in the middle of a session.

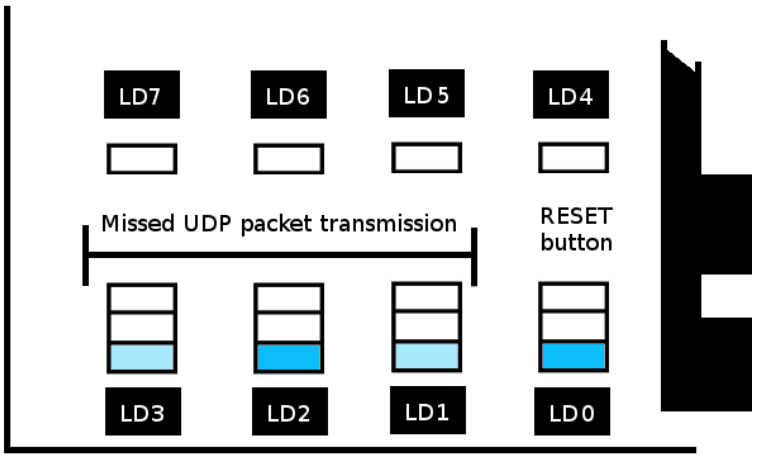


Figure C.6: LED Indicator for missed UDP Packet transmission on the Renishaw ARTY Interface.If the request to send a complete UDP packet is made while another packet is currently being sent, the packet and data will be lost. If this occurs the blue LEDs will count up in binary as an indicator.

C.1.2 Indicator Sample Divisor

Table C.2: The position of the DIP switches to determine the frequency divisor for the scrolling LED's demonstrating that sampling is active.

SW3	SW2	SW1	SW0	Divider
HIGH	HIGH	HIGH	HIGH	2000000
HIGH	HIGH	HIGH	LOW	1500000
HIGH	HIGH	LOW	HIGH	1000000
HIGH	HIGH	LOW	LOW	750000
HIGH	LOW	HIGH	HIGH	500000
HIGH	LOW	HIGH	LOW	200000
HIGH	LOW	LOW	HIGH	100000
HIGH	LOW	LOW	LOW	50000
LOW	HIGH	HIGH	HIGH	25000
LOW	HIGH	HIGH	LOW	10000
LOW	HIGH	LOW	HIGH	5000
LOW	HIGH	LOW	LOW	2000
LOW	LOW	HIGH	HIGH	1000
LOW	LOW	HIGH	LOW	100
LOW	LOW	LOW	HIGH	10
LOW	LOW	LOW	LOW	1

C.2 Renishaw with Dual Teensy 2.0++

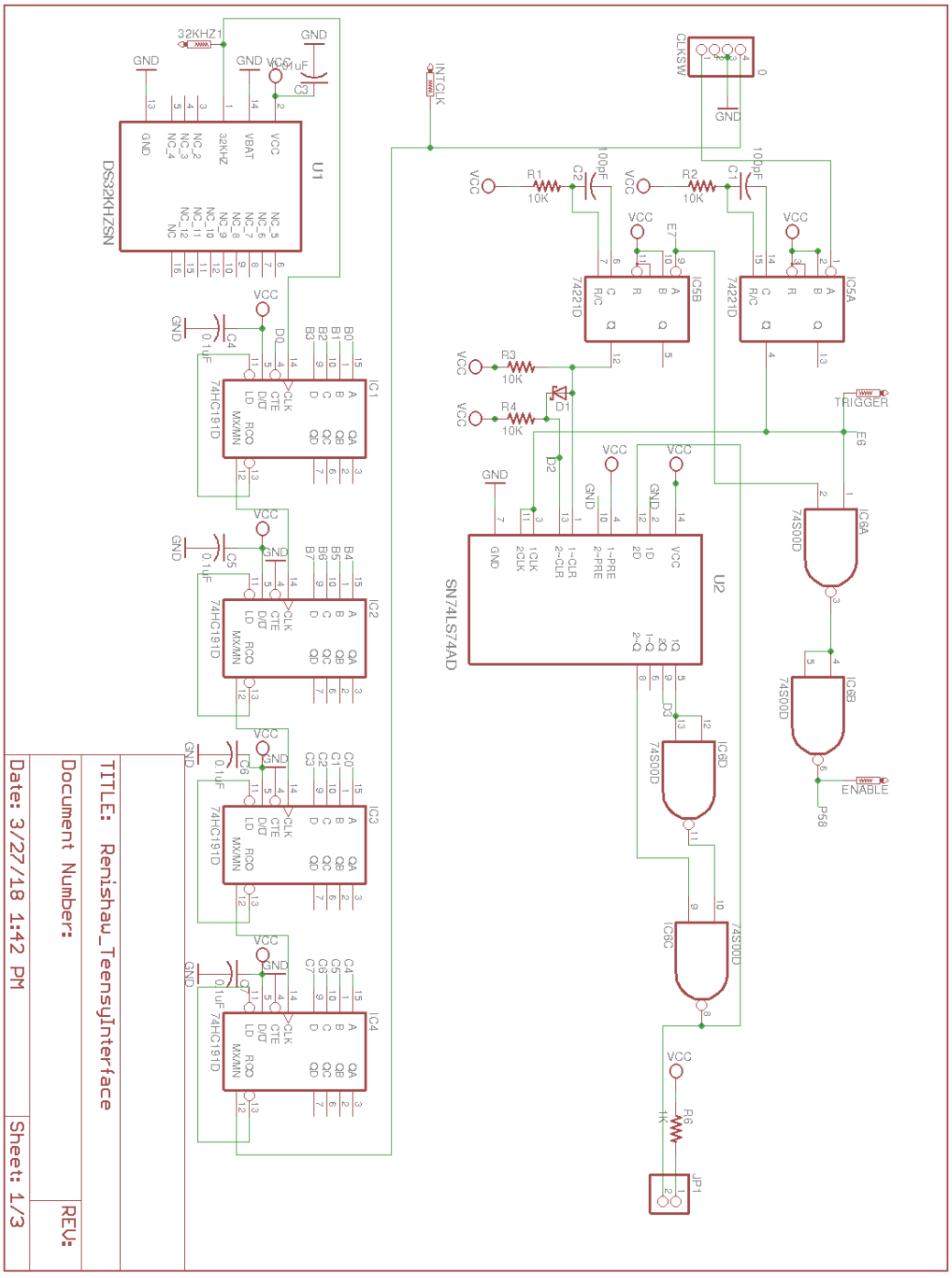


Figure C.7: Schematic for Dual Teensy 2.0++ Depicting Sampling Clock and Error Control. The top portion of the circuit verifies that for every sampling clock pulse an acknowledgement is received ensuring that no samples are missed. The bottom circuit contains a 32 kHz clock and four 4-bit synchronous counting circuits which allowed for a programmable sampling clock.

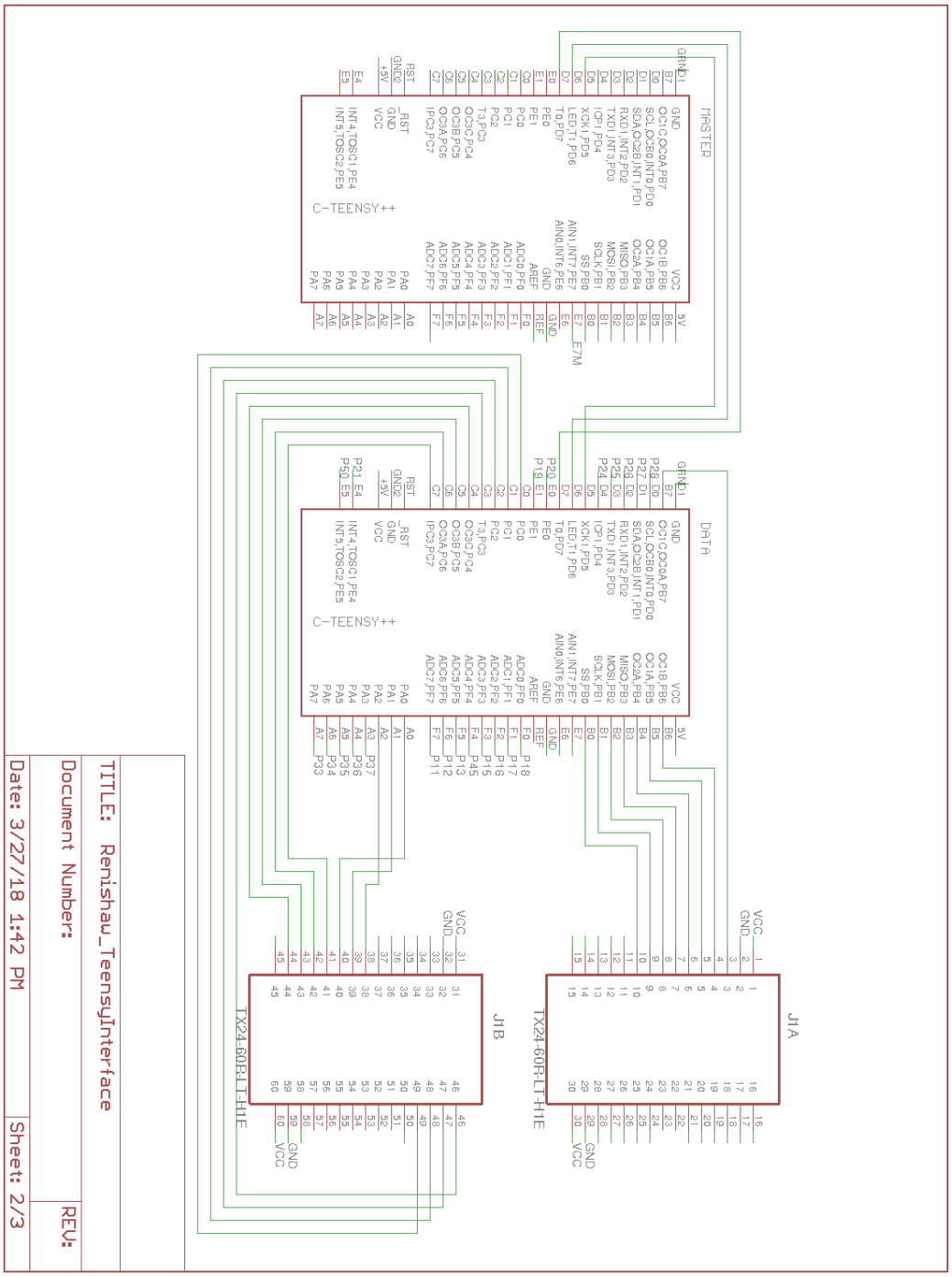


Figure C.8: Schematic for Dual Teensy 2.0++ Renishaw Interface Depicting Slave Teensy and Data Connections. The data pins of the RPI-20 interface card are connected one to one with the data teensy 2.0++ micro-controller for fast parallel reading of position. The master teensy relays commands through a 3-bit binary value on the digital input/output pins which allows the data teensy to send data with no interruptions.

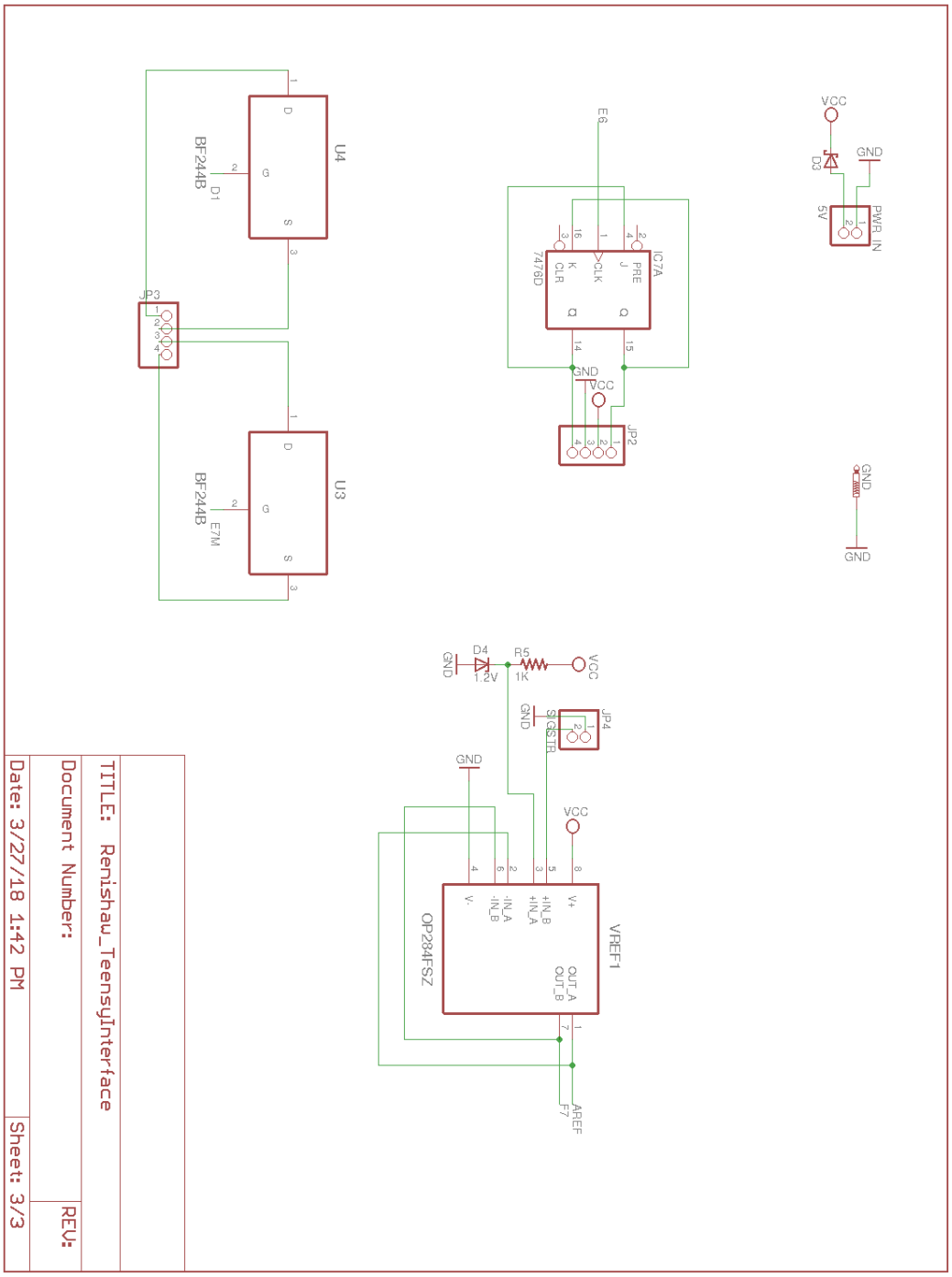


Figure C.9: Supplemental Circuits for Dual Teensy 2.0++ Renishaw Interface. Supplemental portions of the circuit to provide extra features such as being able to read the signal strength, reset the RLE10 laser encoder, place markers in the data and provide a display to give a visual indicator when the device is recording.

C.3 Renishaw with Arty FPGA

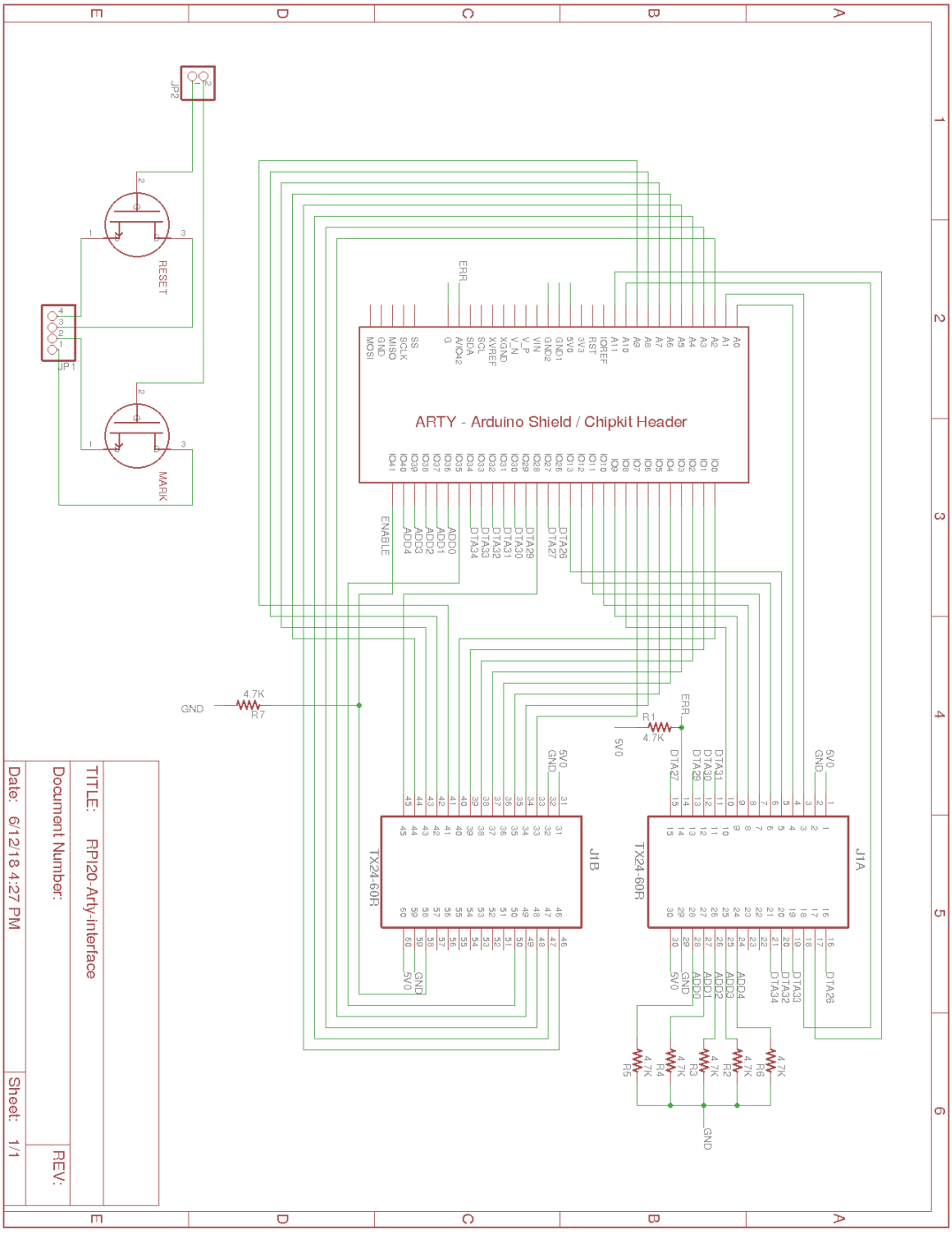


Figure C.10: Schematic for Renishaw ARTY Interface Shield. The shield designed to connect the RPI20 interface card to the ARTY FPGA development board. Almost the entirety of the teensyduino version is emulated by the FPGA, decreasing the number of components while providing 100× increase in performance.

C.4 Renishaw Packet Conversion Algorithm

Data is read from each individual packet and stored in a MySQL data file for easy recall and selection of data. Lines 3-8 of Figure C.11 creates two tables, one to hold the packet numbers and the other to hold the actual position data.

```
1  def RENconvert(fdir,datafn, dbfilename) :
2     print "Renishaw Conversion Started"
3     dbfile = sqlite3.connect(fdir+dbfilename)
4     cur = dbfile.cursor()
5     cur.execute('CREATE TABLE reni(id INTEGER PRIMARY KEY, position INTEGER)')
6     dbfile.commit()
7     cur.execute('CREATE TABLE renipv(id INTEGER PRIMARY KEY, packet_num
          INTEGER)')
8     dbfile.commit()
9     datafile = open(fdir+datafn, mode='rb')
10    sz = getSize(datafile)
11    packet_num=[]
12    position=[]
13    for x in range(0,int(sz/1056.0)-1):
14       data = datafile.read(2)
15       if len(data)<2 : break
16       pn=struct.unpack(">BB",data)
17       pv = ((0|pn[1]) <<8) + pn[0]
18       packet_num.append(pv)
19       pos=2
20       while(pos<1452):
21          data = datafile.read(5)
22          if len(data)<5 : break
23          result = struct.unpack(">BBBBB",data)
24          val =38.6E-6 * twos_comp( (0|result[4]) | (result[3]<<8) |
                (result[2]<<16) | (result[1]<<24)| ((result[0] & 0x0F)<<32), 36)
25          position.append(val)
26          pos = pos + 5
27    for pn in packet_num:
28       cur.execute('INSERT INTO renipv(packet_num) VALUES ({p})'.format(p=pn))
29    dbfile.commit()
30    for pos in position:
31       cur.execute("INSERT INTO reni (position) VALUES ({p})".format(p=pos))
32    dbfile.commit()
33    datafile.close()
34    dbfile.close()
```

Figure C.11: Binary Conversion Algorithm for Renishaw Spatial Metrology. The algorithm to convert data collected by the Renishaw interferometer into a SQL database file. Tables are created for the unique packet numbers and observed positions.

For each packet, the sequential packet number is extracted on line 17 of Figure C.11, and lines 20-26

repeatedly process 5 byte sequences into positional data.

Each 5 byte sequence is reconstructed by the following process :

1. Read 5 bytes (indexed from 0 to 4), where zero is the most significant byte and 4 is the least.
2. Byte 4 is OR'd with a 32-bit zero value.
3. Byte 3 is shifted 8 bits left and ORed with the result from step 2.
4. Byte 2 is shifted 16 bits left and ORed with the result from step 3.
5. Byte 1 is shifted 24 bits left and ORed with the result of step 4.
6. Byte 0 is ANDed with the value 15, then shifted 32 bits left and OR'd with the result from step 5.
7. The two's complement value is then computed on the 36 bit integer result.
8. Finally, multiply the value from step 7 by the resolution of the Renishaw RPI20 parallel interface card, normally 38.6 picometres.
9. A multiplier is applied to adjust the final value to a reasonable unit of millimetres, micrometers or nanometres.

Finally, lines 27-32 place the data into the SQL tables.

Appendix D

Artix-7 FPGA Development Board Data Acquisition System

The ASCII command menu, for the Artix-7 FPGA Development Board (ARTY) DAQ, is provided in Section D.1. The UDP packet conversion routine, necessary to make binary data into coarse clock counts and observed ADC values into their integer values, is given in Section D.2. A description of how the ARTY LEDs are utilized for indicating the status of the DAQ is illustrated in Section D.3. Finally, the schematic diagrams for the ARTY shield and proposed updated version of the shield are shown in Sections D.4 & D.5.

D.1 ASCII Commands

CHR	Function	Response
T	Set destination IPv4 address for the UDP packet receiving computer. Must be followed by 4 bytes representing the IP address.	*
t	Return destination IP address for the UDP packet receiving computer.	4-bytes representing IP followed by an *
R	Set source IP address for the FPGA (must be a nonexistent address on the network) Must be followed by 4 bytes representing the IP address.	*
r	Return source IP address for the FPGA.	4-bytes representing IP followed by an *
M	Set MAC address of UDP packet receiving computer. This is followed by 6 bytes representing the physical address of the data collecting computer.	*
m	Return MAC address of UDP packet receiving computer.	6 bytes representing the MAC address
N	Use external sample clock	*
n	Use internal sample clock (default)	*
P	Set PORT for UDP packet reception	*
p	Return PORT for UDP packet reception	*

S	Start recording data	*
s	Stop recording data	*
I	Identify ARTY DAQ on serial interface	ARTYDAQ
V	Empty buffer	*

D.2 Conversion Algorithm for ARTY DAQ data packets

Data is read from each individual packet and stored in a MySQL data file for easy recall and selection of data. Lines 3-8 of the code snippet creates two tables, one to hold the packet numbers and the other to hold the temporal metrology and ADC data.

```
1  def ARTconvert(fdir, datafn, dbfilename) :
2     print "Arty DAQ Conversion Started"
3     dbfile = sqlite3.connect(fdir+dbfilename)
4     cur = dbfile.cursor()
5     cur.execute('CREATE TABLE arty(id INTEGER PRIMARY KEY, time FLOAT, adc
          INTEGER)')
6     dbfile.commit()
7     cur.execute('CREATE TABLE artypv(id INTEGER PRIMARY KEY, packet_num
          INTEGER)')
8     dbfile.commit()
9     #get number of data bytes in file
10    data=open(fdir+datafn,"rb")
11    fs = getSize(data)
12    packet_num=[]
13    time_val=[]
14    adc_val =[]
15    roll_over=0
16    prev_time=0
17    while(True) :
18       if fs <= data.tell() : break
19       pack_num = data.read(2)
20       pack_val = struct.unpack('>BB', pack_num)
21       pv = int(0|pack_val[0]) + int(pack_val[1]<<8)
22       packet_num.append(pv)
23       pos = 2
24       while pos<1452 :
25          d = data.read(5)
26          val = struct.unpack(">BBBBB",d)
27          val_done= (int(val[0])<<16) + (int(val[1])<<8) + int(val[2])
28          daq_val =  (int(val[3])<<8) + int(val[4])
29          if val_done < prev_time :
30             roll_over=roll_over+1
31          time_val.append( float((val_done)+(roll_over*2**24))/settings.artCLK
```

```
            )
32          adc_val.append(daq_val)
33          prev_time = val_done
34          pos = pos + 5
35      for pn in packet_num:
36          cur.execute("INSERT INTO artypv(packet_num) VALUES ({p})".format(p=pn))
37      dbfile.commit()
38      for pos in range(0, len(adc_val)):
39          cur.execute("INSERT INTO arty(time, adc) VALUES
              ({t},{a})".format(t=time_val[pos], a=adc_val[pos]))
40      dbfile.commit()
41      dbfile.close()
42      data.close()
```

For each packet, the sequential packet number is extracted on line 21 of the code snippet, and lines 24-34 repeatedly process 5 byte sequences into temporal metrology and ADC data.

Each 5 byte sequence is reconstructed by the following process :

1. Read 5 bytes (indexed from 0 to 4), where zero is the most significant byte and 4 is the least.
2. Byte 4 is converted to a 32-bit integer.
3. Byte 3 is converted to a 32-bit integer and then shifted 8 bits left
4. Step 2 and 3 are added together and are assigned to the ADC value (line 28).
5. Byte 2 is converted to a 32-bit integer and then shifted 16 bits left.
6. Byte 1 is converted to a 32-bit integer and then shifted 24 bits left.
7. Byte 0 is converted to a 32-bit integer.
8. The results from steps 5-7 are then added together and assigned to the number of oscillator counts to have lapsed (line 27).
9. Lines 29 and 30 track the occurrence of the counter overflowing.
10. Lines 31 and 32 store the converted values in an array.

Finally, lines 35-40 place the data into the SQL tables. It should be noted that temporal values recorded in the database file are in units of oscillator counts and not time. Currently the coarse counting clock is counting 5 ns periods, but should a variable counting clock be implemented this could be dependent on configuration parameters set at the time of recording. The desire for a coarser resolution may be driven by a slow sample period; currently the minimum period is 83.8 ms, or a sample frequency of 11.9 Hz.

D.3 ARTY DAQ Display LEDs

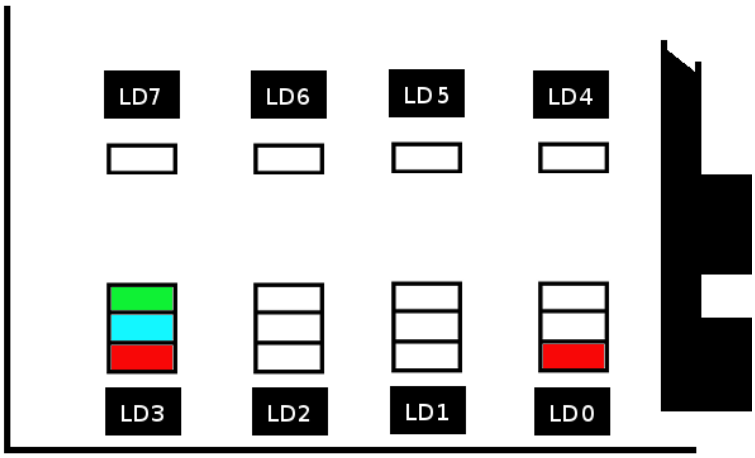


Figure D.1: LED Indicators used on the ARTY DAQ. The ARTY DAQ LED display was simplified during the process of trouble shooting, and reduced to using two RGB LED. The main focus being the status of the FIFO buffer.

In Figure D.1, only two RGB LEDs were used, LD3 and LD0. LD3 is green when the FIFO is empty, blue during a reset, and red if FIFO becomes full. A full buffer is indication that samples are being missed. The red LD0 lamp was an indicator that an attempt to read the ADC occurred while a previous conversion was in progress, most likely due to sampling faster than the ARTY can read values.

D.4 ARTY DAQ Schematic V0.1

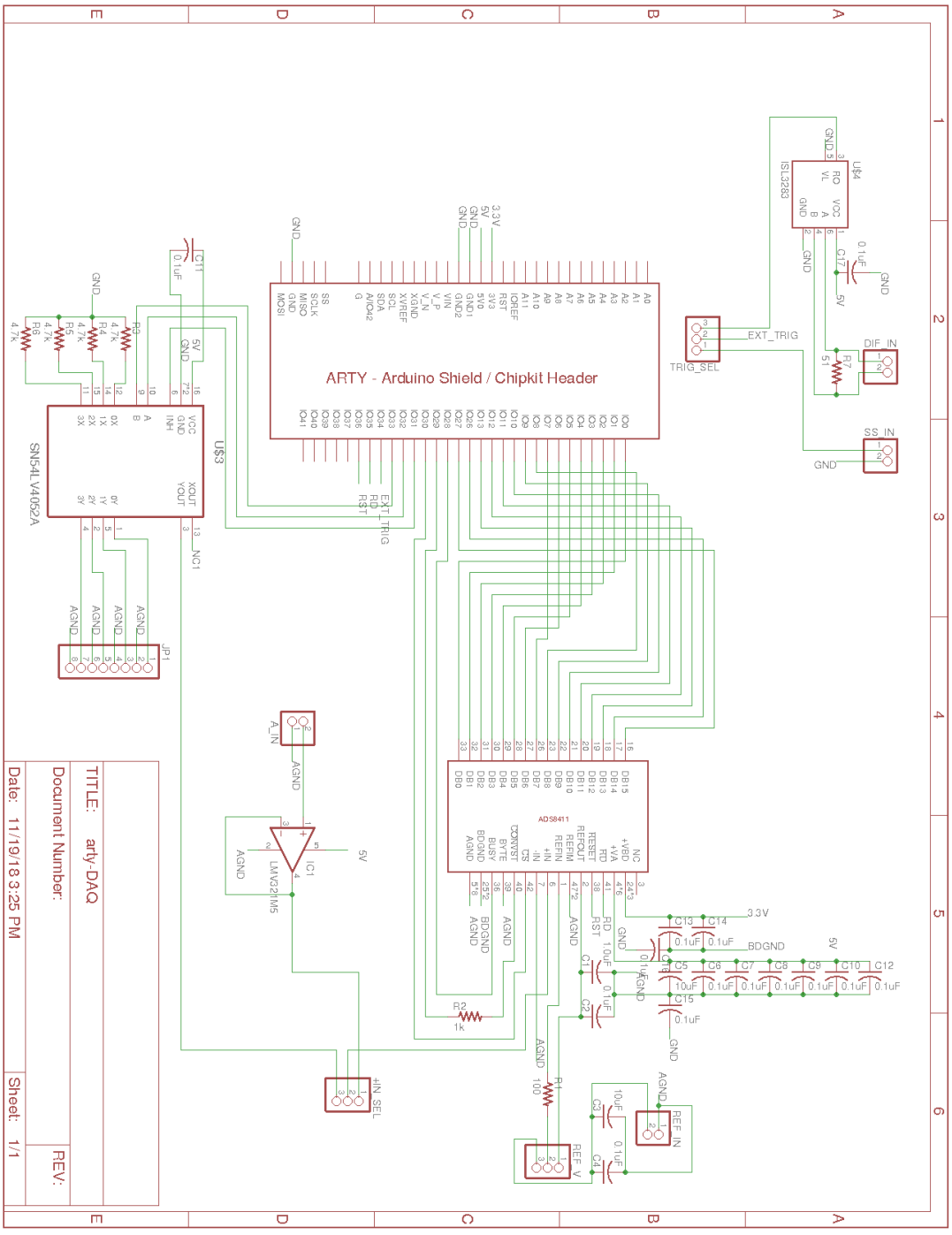


Figure D.2: Schematic for ARTY DAQ Shield. A single 16-bit parallel ADC was connected to the ARTY development board, with a differential PSO input (A1 - A3). E3 depicts a multiplexer to allow up to four analog signals to be input to the ADC, but has not been implemented in the FPGA code.

D.5 ARTY DAQ Schematic V0.2

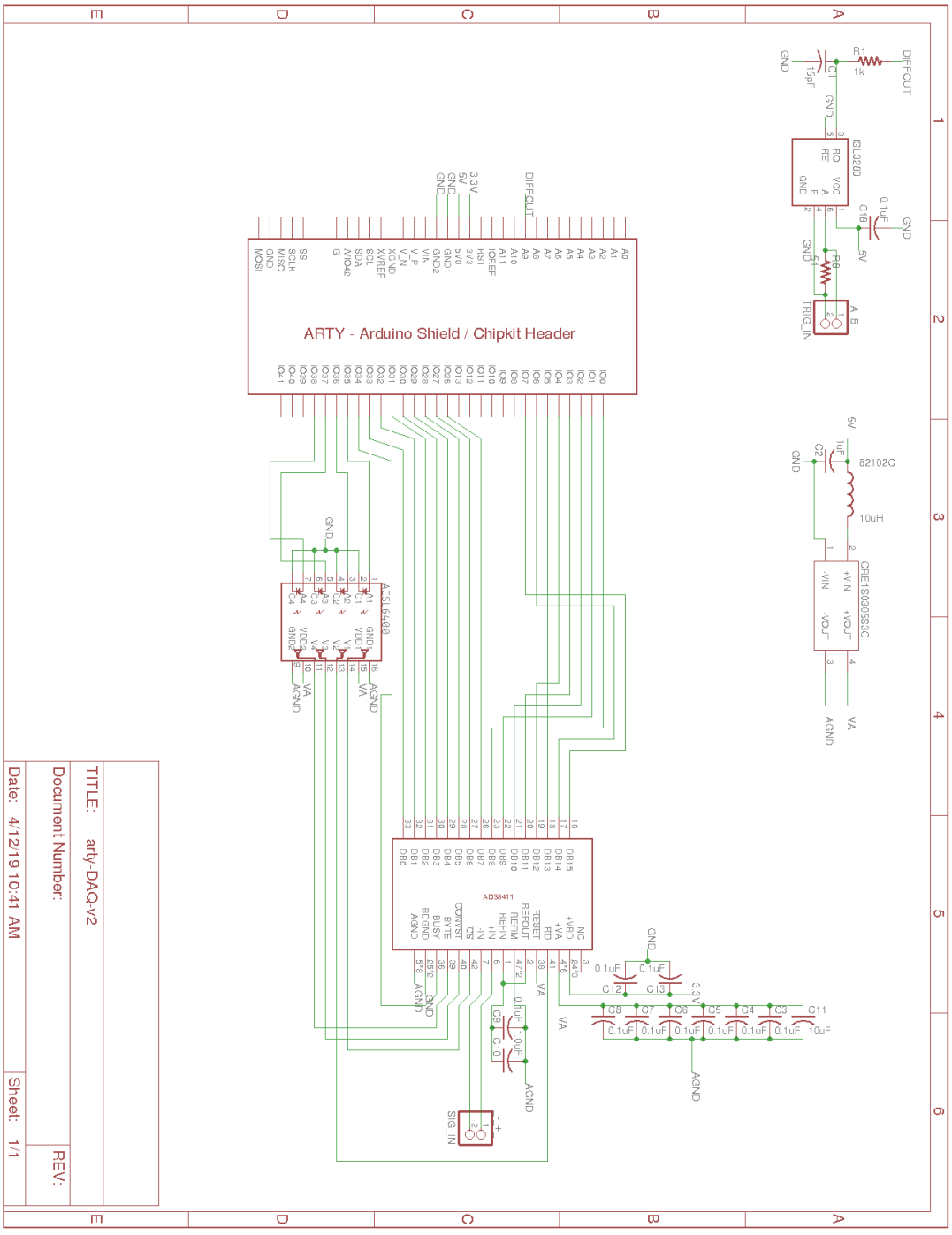


Figure D.3: Schematic for Proposed Updated ARTY DAQ Shield. Updates to help with noise control on the ADC included a 4-channel opto-coupler and a DC to DC converter. The PCB has separate ground pours such that the analog ground isolates the ADC from the 3 V, 5 V, and common ground of the ARTY board. Further changes may include a MUX to allow four inputs to the ADC.

Appendix E

Aerotech Linear Stage

E.1 Aerotech Linear Stage

Before the Aerotech linear stages can be used with the python interface, the initial parameters to allow ASCII text strings sent via network socket must be sent to the stage through the Ensemble Configuration software which requires a computer running the Windows operating system. In the *Configuration Manager*, find the textttCommandSetup Parameter. The ASCII command active port section allows setting which interface to use. The IP address and `PORT` value are required for Python to connect. Once setup the python library written for the stage can be run from any operating system that can run the python scripted language.

First the library must be imported and connection to the stages must be initiated by the following code snippet.

All functions can be called by entering using dot notation to call methods and attributes for the created `Aerotech` object.

`stage.<method>(<parameters>)`

Further description of the commands, and the parameters required, can be found in the Ensemble help files[30] under ASCII command interface.

```
1 import aerotech
2
3 stage = Aerotech()
4 stage.setIP('192.168.1.16')
5 stage.setPORT(16384)
6 if stage.open() :
7    #Do motion commands with delays to keep commands from executing all at once
8 stage.close()
```

Figure E.1: Code Snipped for Initiating Connection with Aerotech Stage. The correct IP address and *PORT* value is required before a connection can be made to the linear translation stage. These values can be found and set in the *Configuration Manager*.

E.2 Python Library Commands

Command	Function
abort($< axis >$)	Aborts all motion on specified axis.
acknowledge()	Clears all error flags on the ensemble controller
close()	Closes the socket connection to the stage.
disable($< axis >$)	Disables the motor for an axis. Axis can be 'X' or 'Y'.
enable($< axis >$)	Enables the motor for an axis. Axis can be 'X' or 'Y'.
home($< axis >$)	Homes an axis and set the relative zero point. Axis can be 'X' or 'Y'.
linear($< axis >, < dist >, < speed >$)	Executes a linear move on the specified axis at the desired speed.
moveabs($< axis >, < dist >, < speed >$)	Causes the stage, specified by axis, to move to the absolute position, specified by dist, relative to the home position.
moveinc($< axis >, < dist >, < speed >$)	Incrementally moves the specified axis, the specified distance.
open()	Connects to the linear stage via network socket.
oscillate($< axis >, < dist >, < freq >, < cycles >, < NumFreq > *$)	Generates sinusoidal osciallation with amplitude equal to specified distance, at given frequency for specified number of cycles. If NumFreq is greater than 1 (default value), the frequency is doubled and distance is halved, up to 4 times.
pcmd($< axis >$)	Get position command of an axis.
pfbkcal($< axis >$)	For the specified axis, gets the position command that includes the output of axis calibration.
pfbkprog($< axis >$)	Gets the program position command of an axis.
program($< command >, < task >, < filename >$)	Commands include "RESUME","RUN", and "STOP". Runs a program, stored on the controller, as a separate task.

psocontrol(< *axis* >,< *mode* >)	Enables/disables PSO hardware.
psodistance(< *axis* >,< *dist* >)	Sets the distance to travel between PSO firing events.
psoFixedDistance(< *axis* >,< *dist* >,< *tottime* >,< *ontime* >)	A function that calls all the required commands to set the PSO to fire at a specified distance.
psoFixedDistanceWin(< *axis* >,< *dist* >,< *tottime* >,< *ontime* >,< *start* >,< *end* >)	A function that calls all the required commands to set the PSO to fire at a specified distance, but includes a windows so acceleration regions of motion can be excluded from the PSO operation..
psooutput(< *axis* >,< *mode* >)	Sets the PSO output mode.
psopulse(< *axis* >,< *tottime* >,< *ontime* >)	Configures the PSO pulse.
psoToggle(< *axis* >,< *dist* >)	Function to set the PSO to alternate high and low states, as the requested distance is travelled.
psotrack(< *axis* >,< *mode* >,< *parameter* >)	Configures the PSO to track distance counters.
psowindow_range(< *axis* >,< *low* >,< *high* >)	Sets the PSO low and high comparison values for specified window.
psowindow_input(< *axis* >)	Sets the window input for specified axis.
psowindow_off(< *axis* >)	Turns window off.
psowindow_on(< *axis* >)	Turns window on.
reset	Resets the controller. Connection will be lost.
setEQD(< *axis* >,< *value* >)	Sets the EQD, requires reset of device and reconnection to socket.
setIP(< *value* >)	Sets the IP used by the linear stage connection socket.
setparm(< *axis* >,< *parameter* >,< *value* >)	Sets a parameter that configures the functionality of the stage. Consult the Ensemble help files [30] for a list of numbered values that relate to the parameter desired (e.g. value = 213 is required for setting EQD).

setPORT(< *value* >)	Sets the port used by the linear stage connection socket.
verbose	TRUE or FALSE attribute which displays detailed output in the console for each command.
waitmode(< *value* >)	"NOWAIT", commands execute immediately interrupting previous command. "MOVEDONE" (default), waits for previous move to finish before executing next command. "INPOS", waits for motion to complete and axis to be in position.