Summary of Project Work During Student Internship

May 2018 to December 2018

Keenan McConkey

Supervisor:

Dr. Sascha Epp

Contents

List of Figures

1 Abstract

The purpose of this report is to document work done on various projects during my undergraduate internship from May to October 2018. This will include an overview of each of these projects, and an explanation of the approach used to complete them. It may also include discussion on the results and future of these projects. It is hoped that this document will serve as a clear reference for the continuation and maintenance of these projects by future student interns.

2 Introduction

The majority of project work was done under the supervision of Dr. Sascha Epp of the MPSD, as part of completing the co-op portion of my BASc. degree at the University of British Columbia, Engineering Physics faculty. Much of the work done on the **VEED/REGAE Chopper System** and **VEED Experimental Setup** was a collaboration with fellow student intern Pawel Mirksi. Note that a large amount of progress on the **Chopper System** and the **BNC Server Arduino** was inherited from previous student interns Mohamed-Ali Hached and Saman Shariat Jaffari. For the purpose of comprehensiveness, there will be some overlap between our respective reports.

The various projects detailed in this report include the use of Arduino microcontrollers, servo motors, programmable logic controllers, control theory, laser optics, CAD, amplifier design, interrupts, serial communication, and TCP/IP networking. Programming languages used include Arduino-C, Python, HTML, and IEC 61131-3. All applicable reference documentation is found in the appendix of this report.

3 REGAE/VEED Chopper System

3.1 Overview

In certain beam line experiments, mechanical choppers are used to control the frequency and pulse length at which a beam hits a sample. In our case, this is done using a aluminum disk with a machined opening along its radius. While spinning at a constant RPM of 50 rev/s, the disk blocks an incoming beam at a controlled frequency. The dimensions of the disk's opening then control the time window during which the beam can pass through the chopper.

In our setup, a secondary chopper is also placed in the beam line after the primary one. This chopper acts to "pick and choose" beam pulses outputted from the first chopper at sub-multiples of the frequency at which they leave the first. Unlike the first, this secondary chopper is not a disk. Rather, it is a machined cylinder with multiple holes along its radius. By switching between a blocked position and a position where a beam can pass through one of these holes, this chopper can select pulses when triggered by an input trigger signal.

Since Chopper 2 takes some time to move from closed to open position, it always has to be triggered off a Trigger rising edge one edge before it actually needs to be open to allow the beam to pass. When triggered, it waits delays for a set time, determined to be 41.5 ms, before beginning its movement to the open position.

For the rest of all documentation this primary and secondary chopper are referred to as **Chopper 1** and **Chopper 2** respectively.

3.1.1 Feedback HeNe Laser

An important component of this chopper system is the feedback HeNe laser. Since it is critical to know the time at which Chopper 1 will let a pulse pass through, a second opening has been machined into the disk of Chopper 1 with a continuous-wave HeNe laser aligned at its radius (Note that it is not actually a HeNe laser but is called so by convention). While the chopper spins, the HeNe hits a detector on the chopper's opposite side which converts the periodic chopping of the HeNe into a 50 Hz pulse signal, which we call the **HeNe Reference** signal. By the geometry of Chopper 1's disk, we know a HeNe Reference pulse will be generated 2 ms before the chopper is open to the beam (when spinning at 50 rev/s). Thus we precisely know when the beam will be able to pass through the chopper.

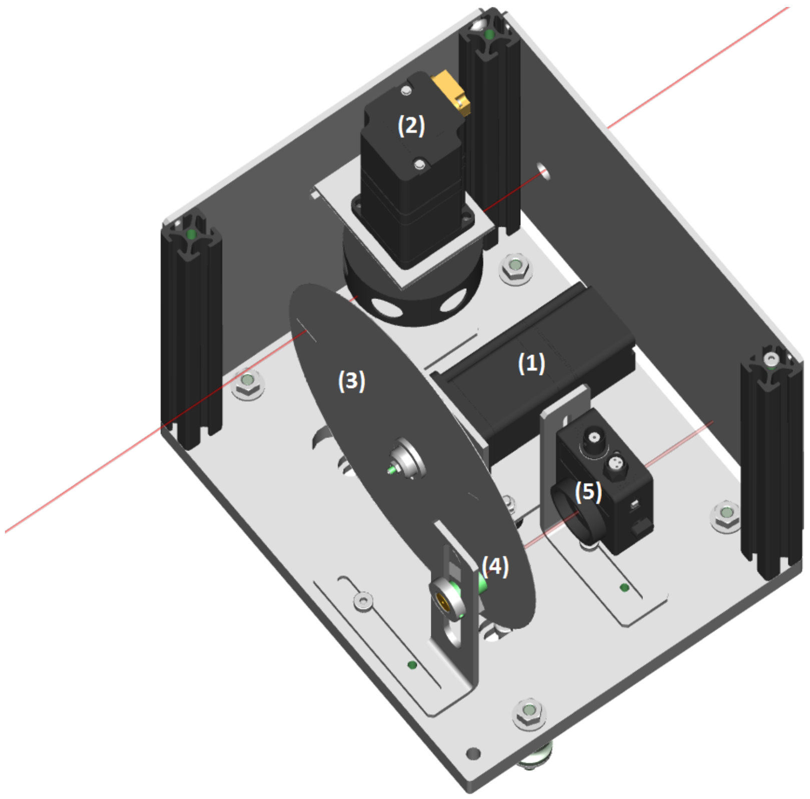


Figure 1: CAD model of Chopper System: (1) Chopper 1; (2) Chopper 2; (3) Chopper 1 disk with two openings; (4) HeNe Laser; (5) HeNe Detector

3.1.2 Yaskawa Motion Controllers

Each of Chopper 1 and Chopper 2 are controlled by two independent Yaskawa MP2600iec programmable logic controllers (1), which are connected to Yaskawa Sigma-5 servo motors (2). These controllers are intended for use in high-precision, high-speed servo motor applications, and include software that allows for programming of various movements. They also include connected digital and analog I/O ports. Similarly they can communicate via TCP/IP for configuration by a PC or for general client-server interactions.

3.2 Delay Error Correction

Knowing the time at which the beam can pass through the chopper is only useful if we can align it to another action. We use one square wave signal to align the timing of the whole system (simply called the **Trigger** or **12.5 Hz Trigger**). This trigger is generated independently of the choppers.

We use a Arduino Due to calculate timing error, which measures the time delay between the HeNe Reference pulse and the rising edge of the Trigger. The Arduino then converts this to a analog voltage which is read by the Yaskawa motor controller for Chopper 1 as a time delay. The motor controller continuously adjusts Chopper 1 so that this time delay is 2 ms, i.e. Chopper 1 open consistently at the rising edge of the Trigger (demonstrated in Figure 2).

If Chopper 1 becomes unaligned so that this delay is not 2 ms, Chopper 1 corrects by spinning at slightly higher or lower velocity until the its open position aligns again with the rising edge of the Trigger. The velocity profile is shown in Figure 3. By integrating the area under this Velocity-Time curve, one can calculate the position correction obtained by executing this velocity shift. Formulas were determined for calculating Δv and t_1.

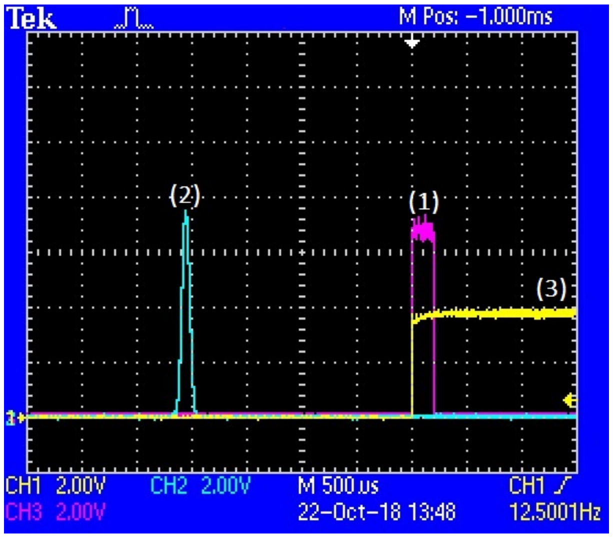


Figure 2: Oscilloscope view of Chopper System: (1) Pulse passing through the chopper; (2) HeNe Reference; (3) 12.5 Hz Trigger rising edge

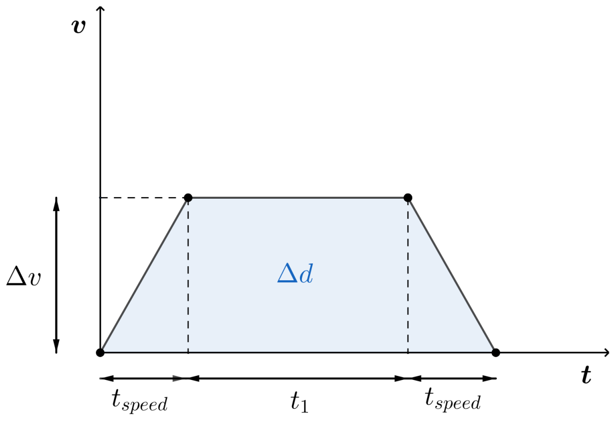


Figure 3: Velocity profile of Chopper 1 correction

$$t_1 = \frac{t_{speed}}{2} + \sqrt{\frac{t_{speed}^2}{4} + \frac{|\Delta t|}{T_{rev} \cdot a}} \tag{1}$$

$$\Delta v = a \cdot t_1 \tag{2}$$

Where...

- t_{speed} = Time it takes for Chopper 1 to accelerate to new velocity, constant (36 ms)
- Δt = Time error calculated by Arduino (deviance from 2 ms)
- T_{rev} = Period of one revolution of Chopper 1 at 50 rev/s (20 ms)
- a = Acceleration of Chopper 1, constant

Since the velocity of the disk is not precisely 50 rev/s at all times (a property of servo motors), some error in time delay between HeNe Reference, and therefore open hole position, and 12.5 Hz Trigger rising edge will always accumulate over time. Thus this correction routine must be performed continuously as Chopper 1 runs.

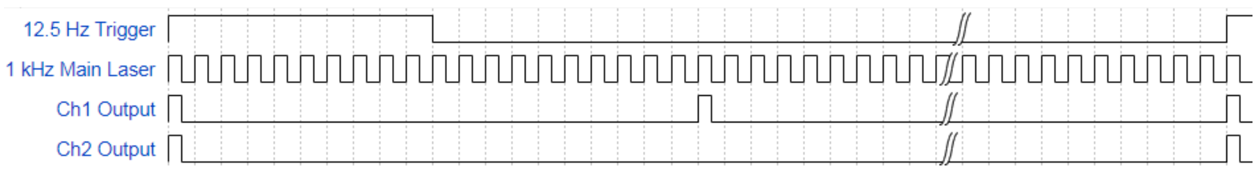


Figure 4: Timing diagram for REGAE

3.3 REGAE vs VEED

The Chopper System is intended to be used in two separate beam line setups at the institute:

- **REGAE** - Relativistic Electron Gun for Atomic Exploration
- **VEED** - Virtual Electron Daemon

In both of these setups, Chopper 1 is to spin at a constant speed of 50 rev/s, which fixes the frequency of beam pulses exiting it at 50 Hz (or once every 20 ms). Chopper 2 is free to be configured at any sub multiple of Chopper 1's frequency as set by the Trigger signal.

3.3.1 REGAE

In the **REGAE** application, a pulsed laser beam (sometimes called the **P-Laser** or **Main Laser**) with a pulse length of approximately 100 femtoseconds is to hit the dual choppers at a frequency of 1 kHz. Chopper 1 spinning at 50 rev/s allows 1 out of every 20 of these 1 kHz pulses to pass, and Chopper 2 then further chooses 1 out of 4 of these pulses, resulting in pulses output at a frequency of 12.5 Hz. In this is experiment, we want the Main Laser to hit a sample at the exact same time as a pulsed electron beam. The Trigger in this case is generated by that electron beam, with equivalent phase.

An important condition of this setup is the "slippage" of the Trigger. All AC wall power electrical lines maintain some frequency (in Germany this is 50 Hz), but due to fluctuations in customer usage this frequency is not constant. This causes an accumulation in phase error between the Main Laser generated by wall power and the Trigger which is generated independently. Therefore after the system runs for some time the Trigger will suddenly slip out of phase with the Main Laser, such that the Trigger is now aligned with a Main Laser pulse one pulse earlier than before. This is pictured in Figure 5.

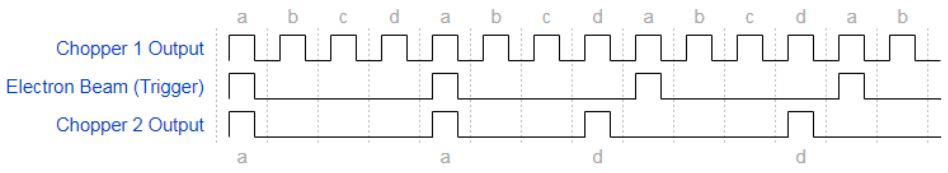


Figure 5: REGAE slippage condition

When this slippage condition occurs, the time delay between HeNe and Trigger rising edge suddenly becomes very high. Chopper 1 must be able to correct for this sudden change as quickly as possible to avoid misaligned pulses hitting the sample. Currently, this correction time takes approximately 160 ms.

3.3.2 VEED

In the **VEED** application, a continuous laser beam is to be chopped into pulses by the system. Again Chopper 1 is fixed at 50 rev/s, and Chopper 2 chooses pulses output from Chopper 1 to allow through. However, in the VEED case Chopper 2 can theoretically select any integer sub-multiple of 50 Hz to output, i.e. 12.5 Hz, 25 Hz or 50 Hz. A new disk for Chopper 1 was designed with a small opening slit (width 1 mm) to generate short pulses from the input continuous beam. In this experiment, the Trigger is only used for synchronization of the system, which we create with a function generator.

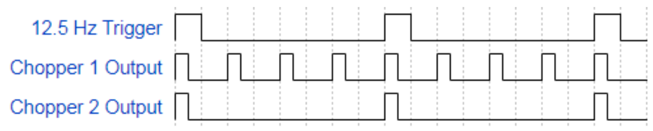


Figure 6: Timing diagram for VEED

As the beam input to the system is continuous, we don't need to worry about slippage from the trigger. Constant correction of Chopper 1 is still needed because of the previously mentioned condition of non-constant disk velocity.

3.4 Implementation

3.4.1 Motionworks

The Yaskawa motion controllers use a custom programming environment called Motionworks IEC. It already include libraries for motor control routines. It also allows for the creation of SFCs (Sequential Function Charts) which makes implementing the logic of motion control easier. Yaskawa has created a series of YouTube videos (3) as an introduction to their programming environment. Additional documentation (4) is always useful.

3.4.2 Ethernet Control

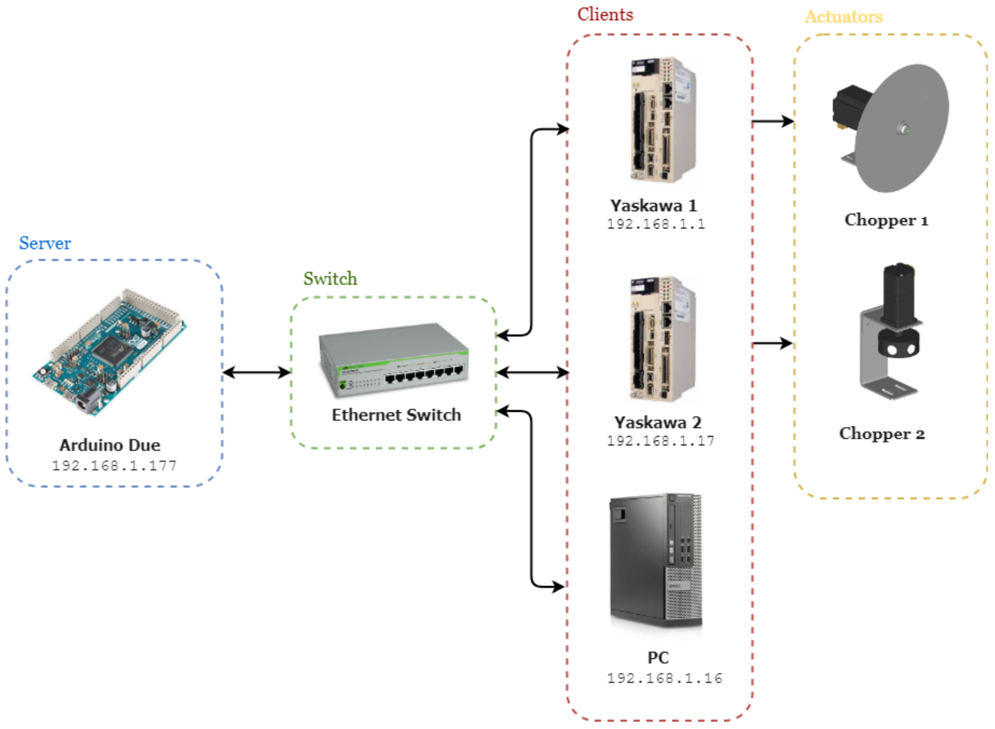
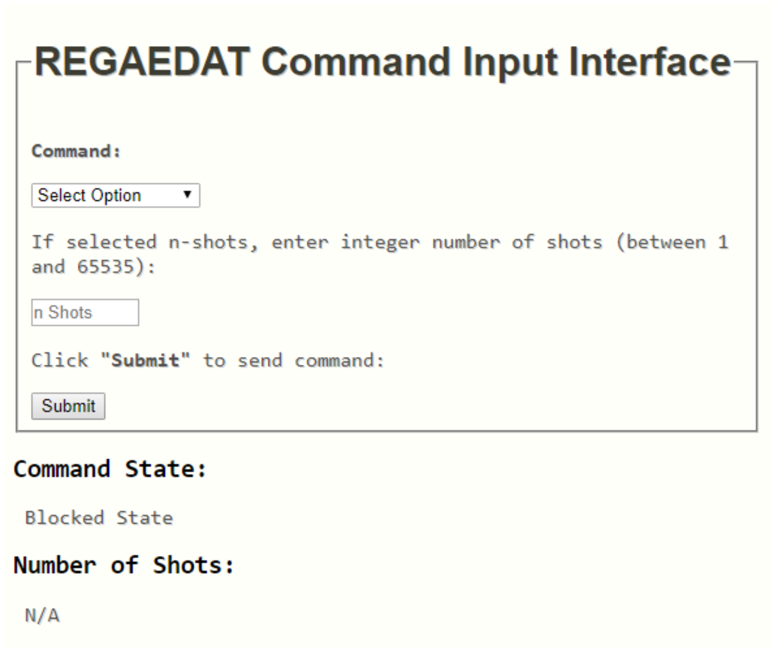


Figure 7: LAN for Ethernet control of Chopper System

One of my main contributions to the Chopper System during my stay was the creation of a network of devices to control the system. The central device of this network is the **BNC Server Arduino**, which acts as a server to accept client connections over Ethernet (on top of performing the delay error calculations above).

With this Arduino acting as a server, a PC on the local network can connect as a client and send commands to be executed by the chopper system. This is implemented in the form of a `HTML` website hosted by the Arduino, which allows a PC user to enter information to be transmitted as an `HTTP Query`. `Javascript` is then used to verify user data. When a new command is sent, the website updates to reflect the change.



REGAEDAT Command Input Interface

Command:

If selected n-shots, enter integer number of shots (between 1 and 65535):

Click **"Submit"** to send command:

Command State:

Blocked State

Number of Shots:

N/A

Figure 8: HTML Command interface hosted by the Server Arduino

The Yaskawa motion controllers are also able to connect to the Arduino server as a client. When a command is received by the Arduino from a PC, the Arduino stores this command locally. A Yaskawa controller is then able to connect to the Arduino at regular intervals and update itself on a change in command from the PC, which changes the functionality of the choppers. The control loop implementation of client behaviour is shown in Figure 23.

A custom transmit/receive buffer had to be created for the choppers and Arduino to send/receive data on the network. To minimize transmission delay, it was decided that the optimal buffer size would be `16-bytes`. Yaskawa 1 and 2 and the Arduino all read/write predefined `char` to the buffer to represent different actions. The configuration of bytes Yaskawa 1 and 2 send/receive with the Arduino over these buffers is as follows:

`[0|1|2|3|4|...|15]`

Yaskawa 1:

`0` : Must be '>' character (ASCII 62). Similar to '?' in HTTP

`2` : Value of Command State for Yaskawa 1 (0 to 255)

Yaskawa 2:

`0` : Must be '<' character (ASCII 60). Similar to '?' in HTTP

`2` : Value of Command State for Yaskawa 2 (0 to 255)

`4` : Least significant byte of n Shots data (0 to 65535)

`5` : Most significant byte of n Shots data

`7` : Protocol bit for transmission control (1 or 0)

`9` : Value of Command State for Y1 to send/receive (0 to 255)

3.4.3 Chopper 1

Most of functionality of Chopper 1 had already been implemented by previous interns. However, a loss of data meant parts of the code had to be rewritten. We also worked on adding Ethernet communication functionality to both choppers. Chopper 1 code was modified to be in the SFC form shown in Figure 24 (Appendix). The current implementation of Chopper 1 allows for four different commands to be executed:

- **Stop** - Stop all movement of the chopper
- **Spin** - Run regular chopping functionality
- **Open** - Align the chopper hole to allow the beam to pass
- **Close** - Align the chopper hole to block the beam

Chopper 1 receives an update on its command state from the Arduino, but for timing purposes, Chopper 2 controls when the Arduino changes its locally stored command state for Chopper 1, i.e. we cannot directly send commands to Chopper 1. Chopper 1 essentially acts as the slave in a Master-Slave relationship with Chopper 2.

3.4.4 Chopper 2

Chopper 2 functionality was greatly expanded. A number of commands needed to be implemented for use in the different setups, including:

- **Block** - Align the chopper hole to block the beam
- **Continuous Wave** - Align the chopper hole to allow the beam to pass
- **Single Shot** - Allow a single beam pulse to pass through the chopper
- **N Shots** - Allow n number of beam pulses to pass through the chopper
- **Continuous Shots** - Allow beam pulses to pass until stopped

The execution of these different commands is done using an SFC control loop, shown in Figure 25 (Appendix). Chopper 2 functionality is completely determined by what is input into the HTML interface and can function without Chopper 1.

3.4.5 System Housing

Work was done during my stay to reduce the size of the Chopper System's footprint. Chopper 2 was adapted to fit directly inside the housing originally only used for Chopper 1. Pieces were reused in conjunction with newly manufactured parts to minimize the size of the system while still providing a safe housing. Overall, a 46% reduction in system footprint area was achieved, without including the area reduction from moving Chopper 2 into the housing.

3.5 Discussion

In its current state, the Chopper System was used effectively for experiments in the VEED application. Possible areas of improvements in the system include: reducing overall delays in the system, including Chopper 1 velocity correction delays and Chopper 1/2 startup delays; optimizing and testing Chopper 2 to work with frequencies other than 12.5 Hz; creating verification tool that checks that an "N Shots" command sends the correct number of shots; and improving overall error-handling in the system.

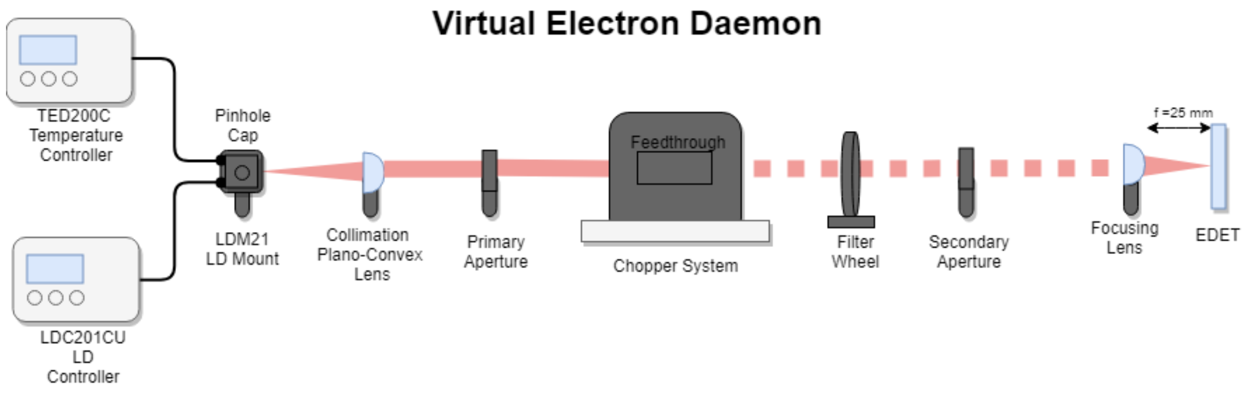


Figure 9: Schematic of VEED

4 VEED Experimental Setup

4.1 Overview

VEED is a beam line daemon created to simulate a pulsed electron beam using packets of photons with equivalent energy. This is to be used for calibration and debugging of an **EDet** (Electron Detector) sample, in this case a high-speed detector for use in imaging applications. Energy equality of n photons equal to one electron can be calculated at a given wavelength:

$$E_{e^-} = n \cdot E_\gamma = n \cdot \frac{hc}{\lambda} \tag{3}$$

Where...

- h = Planck's constant
- c = Speed of light
- λ = Wavelength of the photons

In order to reach an energy comparable to singular electrons an 880 nanometer laser has to be greatly reduced in optical power. Note that since optical power and number of photons per second are proportional we can think of them interchangeably.

4.1.1 Chopping

As previously described, the input to the VEED is a continuous wave laser beam, which is chopped into small pulses by the Chopper System. One can calculate the power of a single pulse output from the system using the geometry of Chopper 1 and both Chopper 1 and 2's chopping frequency.

$$P_{out} = \frac{P_{in}}{v_1 v_2} \cdot \frac{d_{slit}}{2\pi r_{slit}} \tag{4}$$

Where...

- P_{in} = Optical power of the input continuous-wave beam
- v_1, v_2 = Speed of Choppers 1 and 2, respectively, in revolutions per second
- d_{slit} = Width of machined slit in disk of Chopper 1
- r_{slit} = Radius of machined slit, assumed large compared to width

4.1.2 Filtering

Beam power is further reduced by filtering to get down to power equivalence of an electron. We use neutral density (ND) filters to greatly reduce beam power. An ND filter is characterized by its optical density, which reduces the power of the beam exponentially:

$$\frac{I}{I_0} = \frac{P}{P_0} = 10^{-d} \tag{5}$$

Where...

- I_0 = Incident beam intensity
- P_0 = Incident beam power
- d = Optical density of the filter

4.1.3 Beam Collimation

To be used in the relatively long distances of our setup, the input beam needs to be collimated. A perfectly collimated beam of light has parallel rays, such that it does not diverge as it propagates. Perfect collimation is impossible due to diffraction, but we still try to minimize divergence as much as possible using lenses (e.g. 40 mm plano-convex lens).

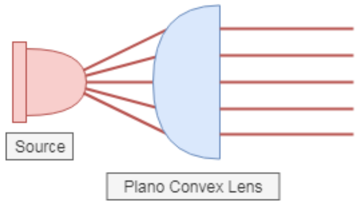


Figure 10: Collimation with a plano-convex lens

4.1.4 Diffraction

As shown in Figure 9, apertures are used in the setup to reduce beam diameter. However, at the low diameter apertures we use, diffraction becomes an issue as the wavelength becomes comparable to the aperture size. Thus various sized apertures are used to cut off diffracted beam patterns at multiple points along the setup.

4.1.5 Focusing

After finally reaching the desired beam power and thus the desired number of photons per second, the beam is focused in order to hit a single pixel of the detector. A beam diameter of approximately 100 micrometers was found to be effective for this goal.

4.2 Pinhole Power Cap

A laser cap needed to be designed to ensure the beam power does not exceed the Class 1 laser limit of 1 mW at normal operating current. This cap had to fit on a purchased laser diode mount (LDM21) (5). Optical power was measured at varying current inputs to the diode mount, and an optimal current for power close to 1 mW was determined (Figure 12).

Special cap-screws were used to make it difficult for inexperienced users to remove the cap, and the cap was anodized in matte-black to minimize all back-reflections of the beam. In order to ensure we could change the size of the pinhole fairly easily, the cap was split into a pinhole piece and a mount, so that new pinhole pieces could be manufactured without having to remake the entire part.

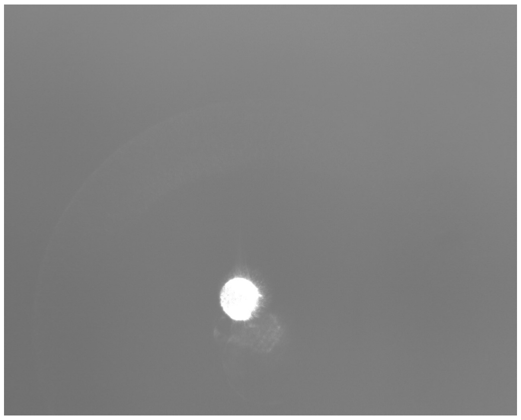


Figure 11: Unfocused VEED beam capture taken before Chopper System

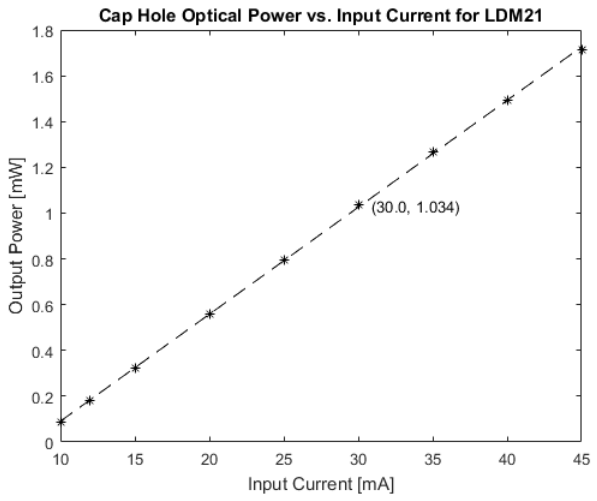


Figure 12: Optical power of capped laser

4.3 Feedthrough Wall

In order to minimize the effects of any stray photons on the sample, the VEED apparatus needs to be enclosed in a light-tight environment. This results in some difficulties in providing the necessary cables for the system. For general cables, a premade cable enclosure was purchased that allows for the insertion of variously sized cables while still keeping the VEED enclosure sealed.

One of the cables for the EDet sample could not be fit into the premade part, so a custom feedthrough panel was designed to enable the insertion of this cable without compromising light sealing. This panel allows for rotation of the cable by 90°, and has adjustable clamps to hold the cable tightly.

4.4 Detector Stage

A liear 3-axis stage needed to be designed to facilitate movement of the beam to different pixels of the grid of pixels making up the detector. The detector also has to move into the 25 mm focal length of the beam. A custom mounting plate was machined for this purpose. In the future this could be motorized but currently the stages can only be moved manually.

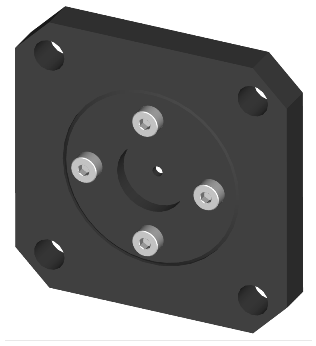


Figure 13: Rear view CAD model of Laser Cap

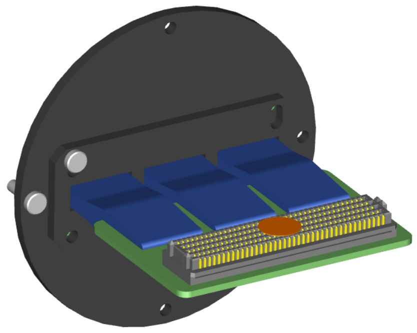


Figure 14: CAD model of feedthrough panel

4.5 Discussion

The VEED system has shown potential as a calibration device for electron detectors. The setup was taken to the Max Planck HLL detector lab in Munich for testing. There we produced illumination of a single detector pixel, which is crucial component detector calibration. As described, the detector can be manually moved using the stage for probing of different detector pixels. Different probing modes can be selected using the Chopper System HTML interface. Unfortunately, I do not have access to the electron detector data and thus cannot include relevant graphics showing pixel detection.

The obvious area of improvement of this project is the motorization of the detector stage. After this is completed, the detector's movement could be synchronized with the software of the detector for automated calibration of every pixel.

5 BNC Server Arduino

5.1 Overview

An Arduino Due has been used to implement a variety of different functions simultaneously, as needed for the Chopper System. These include

- Acting as an Ethernet server with connected SD card memory
- Communicating with other devices over USB and RS232 serial
- Measuring the time delay between two input pulse signals (**VTT**)
- Counting number of pulses over time on an chosen input signals

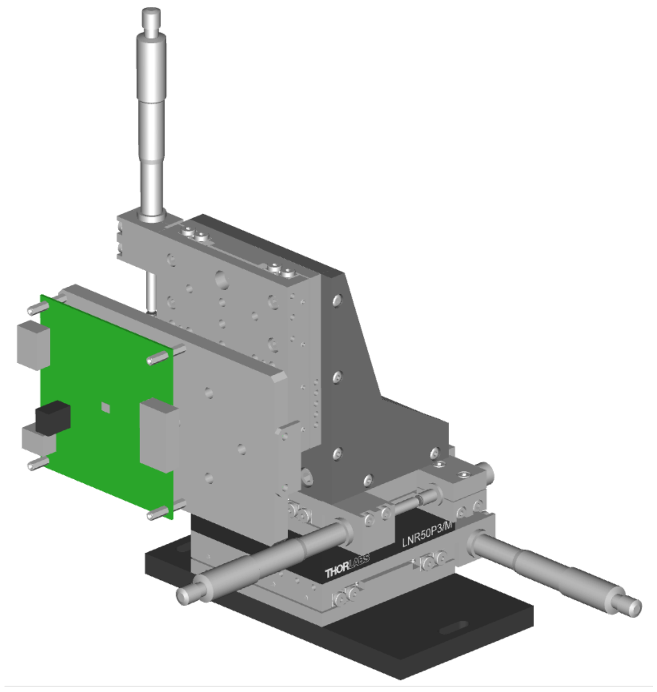


Figure 15: CAD of detector on the detector stage

Implementation of time delay and USB/RS232 serial communication functionality was completed by previous interns, however work was done to greatly expand the number of commands executable over serial. These commands can be used to debug all of the different functions of the Arduino over a PC connection via USB. At present, the commands available over serial are as follows:

```
Basic Commands:
* ? - Returns info on the system
* help - Returns a list and info on commands
* server:? - Returns info on server settings
SD Card Commands:
* sd:? - Get information about the SD card
* sd:print - Get the contents of the SD card root directory
* sd:printFile:[file] - Print the contents of a file
* sd:remove:[file] - Remove a file from the root directory
VTT Commands:
* vtt:? - Get the current config of VTT
* vtt:[d_in1]:[d_in2]:[a_out] - Set the BNC input/outputs
* vtt:[time(us)] - Set the maximum delay time
* vtt:start:2CHAN - Start 2 channel function
* vtt:stop - Stop VTT function
Yaskawa Commands:
* yaskawa:? - Get current command state and n shots
* yaskawa:state:[0-255] - Change Yaskawa state
* yaskawa:nShots:[0-65535] - Change Yaskawa n shots
CountPulses Commands
* countPulses:? - Get CountPulses config
* countPulses:command - Print all CountPulses commands
```

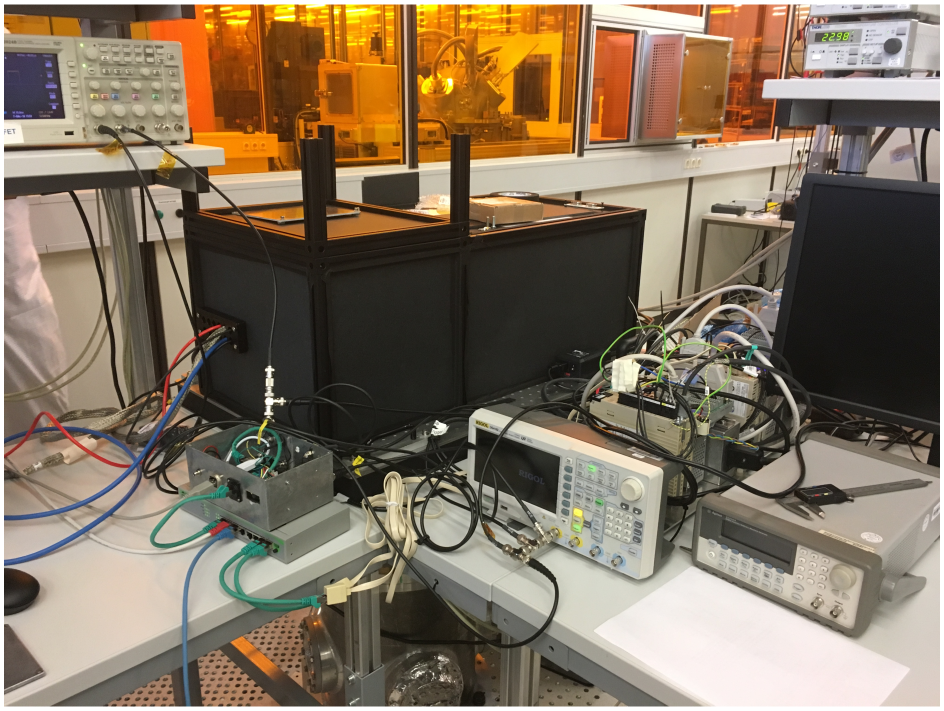


Figure 16: VEED setup in MPG HLL Munich

5.2 Ethernet Server

By attaching an Ethernet Shield, the Arduino can act as an Ethernet server that accepts client connections from both PCs and Yaskawa motion controllers. When connected to by a PC,the Arduino hosts a web interface which the PC can then use to send commands back to the Arduino. The Arduino stores these commands in local variables, which the Yaskawa controllers can then access by connecting to the Arduino themselves. Put together, this allows a user to control the Yaskawa controllers remotely from any PC using a straightforward web interface.

Website HTML code is stored on a SD card connected to the Arduino. This implementation uses the native Arduino libraries `Ethernet` (6) and `ESP8266 SD` (7), which are well-documented. Communication protocol is further detailed in Section 3.4.

5.3 Time-to-Voltage Converter

The Arduino measures the time delay between two digital input signals through the use of interrupts. When a rising edge is detected on Digital Input 1, ISR 1 starts a time counter. The Arduino then waits until a second rising edge is detected on Digital Input 2, at which point ISR 2 resets the counter and writes it to Analog Output (which has a built in DAC). If a rising edge detected on Digital Input 1 before Digital Input 2, the counter is reset and the value output on Analog Output is not changed.

Analog output on the Arduino Due has a resolution of `12-bits`, so the time value is converted to a value between 0 and $2^{12} = 4096$. Thus a maximum time delay is needs to be set at 4095. When delay goes over this maximum, its value resets to 0. Details on the use of interrupts in Arduino Due can be found in the documentation of its Atmel 32-bit microcontroller (8) on board the Arduino.

In the chopper implementation, `Digital Input 1` is the HeNe Reference, `Digital Input 2` is the 12.5 Hz Trigger, and `Analog Output` is input into Yaskawas 1 and 2. This voltage was calibrated on the side of the Yaskawas, by graphing varying delay and calculating the linear relationship between delay and voltage measured at the Yaskawa analog input, shown in Figure 17.

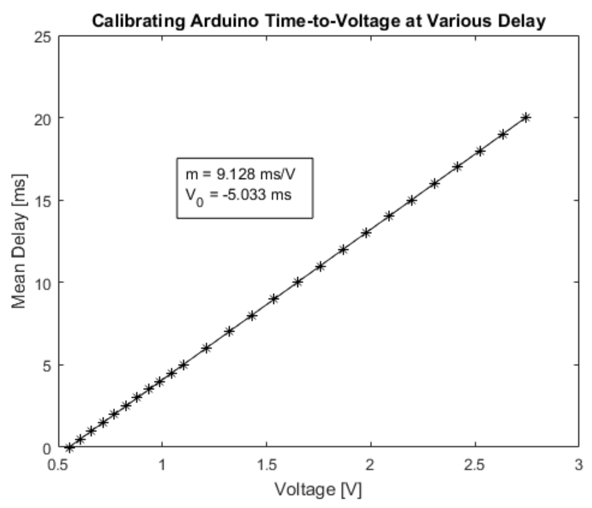


Figure 17: Yaskawa-side voltage for varying delay input into Arduino

5.4 Pulse Counter

The Arduino is able to count the number of pulses (i.e. the number of rising edges) detected on a chosen `Digital Input`. An `ISR` is run to increment a counter every time a new pulse is detected. We can use this to verify that the number of shots output by the chopper system is correct by using a photodiode as the input.

5.5 Discussion

Overall, the BNC Server Arduino has been proven effective in multiple its multiple pre-described applications, including the VEED experiment. At present, no new features seem to be required, but testing and error-checking of the device are always helpful.

6 Nanosecond Pulsing of an LED

6.1 Overview

To illuminate a sample for plume imaging during ablation experiments, a pulsed light source was needed with a pulse length on the order of nanoseconds and a repetition rate below 100 Hz. It was determined that an LED could be used for this application but a high-speed amplifier would be needed to boost input currents and voltage at high frequencies. The use of LEDs at nanosecond pulse-lengths is based on the work of Rose et al (9). High-speed amplifiers designs suitable for this purpose was found in Texas Instriuments' guide (10).

6.2 Implementation

This project is currently at the breadboard prototyping stage. A circuit (see Figure 22) was designed combining the circuits found in (9) and (10). Certain resistors have been modified/removed to tune for the best possible amplification. An ultrahigh speed operation amplifier (11), which performs at speeds up to 1 GHz, was purchased for use in the circuit.

Amplification has been demonstrated with both nanosecond scale pulses (with varying repetition rates), and nanosecond period sine and square waves. Pulse length and input amplitude were found to have a significant effect on both the pulse length and a amplifier gain. Above 50 MHz pulse repetition rate the gain starts to decrease.

An oscilloscope probe was used to measure the voltage drop across the diode, as shown in Figures 18 and 19. Amplifier gains of over 10dB were demonstrated. There was some difficulty with observing the LED light up at low pulse lengths, suggesting that the red LED(12) we chose may not be optimal for the nanosecond scale.

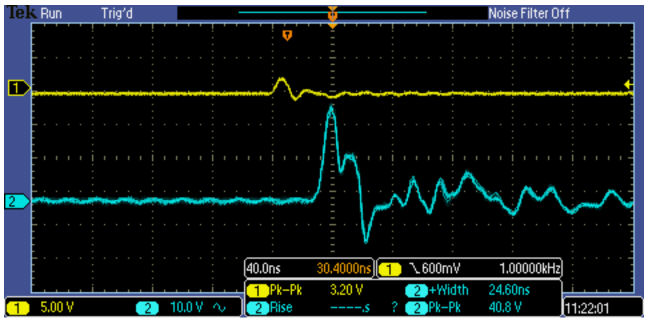


Figure 18: Nanosecond-scale amplification of a 8 ns pulse

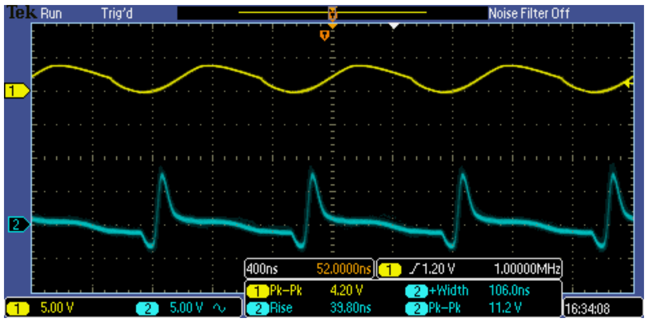


Figure 19: Nanosecond-scale amplification of a 1 MHz sine wave

7 M-Drive Serial Control Server

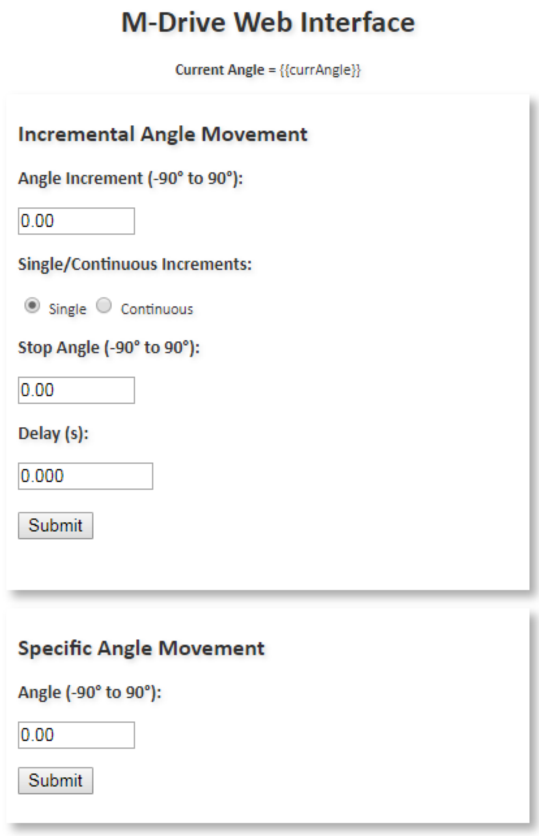
7.1 Overview

In optical setups at the institute, a rotating platform was needed that could be made to rotate to various angles with a precision of 0.1°. An M-Drive (13) servo motor connected to this platform could be controlled via a USB serial connection to a computer for this purpose. It was thought that the motor should be able to take commands from multiple computers, which means implementing client-server system. A local server program was created which runs on the computer connected to the M-Drive motor. The servers hosts a web interface where a user can enter commands which are translated to motor movements.

7.2 Implementation

The server has been implemented in Python using an open-source server microframework called `Flask` (14). The library enables the creation of locally hosted servers, and allows clients to transmit `HTTP` requests to the server, which uses Javascript to validate user data. To communicate with the M-Drive, the server uses the serial library `PySerial` (15), which lets the server send a string of text to the M-Drive that is interpreted as different pre-defined movements.

A client of the server is able to tell the motor to move to a certain angle, or they can have it execute a relative movement of an incremental angle, either a single time or continuously until some stop angle is reached. A delay time between angle movements can also be set. Further documentation and source code can be found on the project's GitHub page (16).


M-Drive Web Interface

Current Angle = {{currAngle}}

Incremental Angle Movement

Angle Increment (-90° to 90°):

0.00

Single/Continuous Increments:

[✓ Single] Continuous

Stop Angle (-90° to 90°):

0.00

Delay (s):

0.000

Specific Angle Movement

Angle (-90° to 90°):

0.00

Figure 20: HTML web interface for entering M-Drive motor commands

8 Conclusion

Many of these projects are not completed to their fullest potential, and further work can always be done to test and improve their functionality. Hopefully this document clearly outlines the implementation of these projects for future student interns. Of course if anything is unclear, I can be contacted through Dr. Epp with any questions.

9 Appendix

9.1 Relevant File Locations

- Public files from `Regaedat` computer - Including old Motionworks code, inherited files from previous interns

  ```
  N:\regaedat\cfel
  ```

- Public files from Keenan - Including Motionworks and Arduino code, Data, MATLAB scripts, Documentation, SolidEdge parts and drawings

  ```
  N:\mpsdgst5\cfel\Keenan_2018
  ```

- Public files from Pawel

  ```
  N:\mpsdgst9\cfel\Pavel_2018
  ```

Note 1: There may be some overlap between files found in these directories.
Note 2: The newest Arduino and MotionWorks code files should be named `BNCServer` and `CH1 07 12 2018 / CH2 07 12 2018` respectively
Note 3: A backup of relevant files has also been put onto a USB stick, see Dr. Epp.

9.2 Operational Setup of Chopper System

9.2.1 Function Generator Setup

1 Turn on the function generator

2 Set Channel 1 to Generate a square-wave from 0 V to 5 V at 12.5 Hz, set to the smallest duty cycle

3 Initially, make sure that the Channel Outputs of these functions are off

9.2.2 Arduino Due Box Setup

1 Turn on the Arduino Due by connecting a USB cable from a Windows computer (which has the Arduino interfacing tools installed on it) to the Arduino Due box

2 Make sure that the `BNCServer.ino` Arduino Sketch has been uploaded onto the Arduino

3 Ensure that an SD card with the correct file `regaedat.html` is loaded onto the Arduino

- It is possible to view all files on the SD card sending the `sd:print` command over serial connection
- The SD card must be formatted as `FAT16/FAT32`

4 Connect to the Arduino over serial USB with the correct settings: `Port = COM4` and `Baudrate = 9600`

- The Arduino IDE can be used or any other software for serial connections, for example PuTTy

5 By sending the serial command `vtt:?`, validate that the Arduino Due Box is set to the following configuration:

(a) `Max Time Delay = 20000(us)`
(b) `Digital Input 1 = D3 IN`
(c) `Digital Input 2 = D1 IN`

(d) `Analog Output = A2 OUT`

(e) `State:  vtt2`

(f) `LowerBound = 0`

(g) `UpperBound = 4095`

6 By sending the serial command `server:?`, verify that the Arduino's IP Address is set to `192.168.1.177` and port `80`

7 Connect the 12.5 Hz Trigger signal from the function generator to the BNC port `D IN 1` on the Arduino Box

8 Connect the HeNe Reference signal from the photodiode to the BNC port `D IN 3` on the Arduino Box

9 When all the inputs are connected, begin the signal processing on the Arduino by sending the command `vtt:start:2CHAN`

10 Connect the `A OUT 2` BNC port of the Arduino to the `VTT OUT` labelled BNC cable of Chopper 1.

9.2.3 Ethernet Connection Setup

1 Connect the Ethernet output of the Arduino BNC Box to the Ethernet Switch

2 Connect an Ethernet port of any PC to the Ethernet Switch

3 Make sure the IP Address of the PC's Ethernet port is on the same local subnet of the Arduino server, i.e. the first 3 bytes of the IP Address are the same as the Arduino's (e.g. `192.168.1.3`)

4 Connect Ethernet port CN11A of Chopper 1's MP2600iec to the Ethernet Switch

5 Make sure the IP Address of this MP2600iec is set to `192.168.1.17` using the Hardware Config menu in MotionWorks software

(a) To do this, the built-in E-INIT switch of the MP2600iec (found just above the large CN13 input port) must be set to OFF

(b) This allows a user to set a non-default IP Address

6 Connect Ethernet port CN11A of Chopper 2's MP2600iec to the Ethernet Switch

7 Make sure the IP Address of this MP2600iec is set to `192.168.1.1` using the Hardware Config menu in MotionWorks software.

(a) This should be the default IP Address of the Yaskawa

(b) You can force it to this default address by setting the built-in E-INIT switch of the controller to ON

8 Open any browser on the PC, and connect to the IP Address of the Arduino Server, i.e. `192.168.177`

9 You should now be able to send commands via browser to the Arduino by selecting a command and clicking the Submit button

9.2.4 Setup of Choppers 1 and 2

1 Ensure connections are as shown in Figure 9 schematic

(a) Connect the 12.5 Hz signal from the function generator to the solid-state relay of Chopper 1 labelled as "12.5 Hz"

2 After all cables have been connected, there are 2 power sources that may need to be toggled for the motor controller

   (a) One for the solid-state relays, and one for the motor controller itself
   (b) Pay careful attention when working with these power sources as they handle very high voltages

3 The choppers should automatically find their homing positions even if they were moved while offline

4 Upload the MotionWorks code onto the MP2600iec motor controllers as follows:

   (a) Turn on MotionWorks 2 software and open the latest project file
   (b) Select Build - Rebuild Project
   (c) Open the Project Control Dialog Box found in one of the top panels of the MotionWorks window
   (d) On the Project Control Dialog Box, click the Download button
   (e) On the Project Control Dialog Box, click the 'Warm' Button to enable the project
   (f) Repeat for other chopper

5 Go to the function generator hardware and turn on the output of the 12.5 Hz and 1 kHz signals

6 Connect to either Choppers by entering their IP address into any browser

   (a) For administrative access, the username is `Admin` and the password is `MP2600` (case sensitive)
   (b) If the Choppers are not operating correctly, try rebooting through this menu

7 You should now be able to send commands via browser to the Arduino by selecting a command and clicking the Submit button

   (a) You can observe live values of code variables in MotionWorks by clicking 'Debug On/Off'
   (b) Logic Analyzer is also a very useful tool
   (c) Both of these are explained in the Yaskawa online video training series

References

[1] Y. A. Inc., *MP2600iec Reference Page*. [Online]. Available: https://www.yaskawa.com/products/motion/machine-controllers/mpiec-series/mp2600iec

[2] ——, *Sigma-5 Servo Reference Page*. [Online]. Available: https://www.yaskawa.com/products/motion/sigma-5-servo-products

[3] ——, *Motionworks Introductory Video Series*. [Online]. Available: https://www.youtube.com/playlist?list=PLNAENlyEDCkzMwkIpWNwX0DeJdlVFwyB0

[4] ——, *Motionworks Reference Page*. [Online]. Available: https://www.yaskawa.com/products/motion/machine-controllers/software-tools/motionworks-iec

[5] Thorlabs, *LDM21 Manual*. [Online]. Available: https://www.thorlabs.com/thorproduct.cfm?partnumber=LDM21

[6] Arduino, *Ethernet Library Reference*. [Online]. Available: https://www.arduino.cc/en/Reference/Ethernet

[7] F. Sallemi, *ESP8266 SD Library Reference*. [Online]. Available: https://links2004.github.io/Arduino/d6/dd9/class_sd2_card.html

[8] A. Corporation, *SAM3X Series ARM Microcontrollers Datasheet*. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf

[9] J. Rose, S. Bradbury, I. Bond, P. Ogden, A. Price, R. Oliver, and Y. Farkhatuly Khassen, "Driving LED in a Nanosecond Regime by a Fast Operational Amplifier," *arXiv e-prints*, p. arXiv:1011.1954, Nov. 2010. [Online]. Available: https://ui.adsabs.harvard.edu/#abs/2010arXiv1011.1954R/

[10] T. Instruments, *Application Report - AN-272 Op Amp Booster Designs*, 1981. [Online]. Available: http://www.ti.com/lit/an/snoa600b/snoa600b.pdf

[11] A. D. Inc., *AD8009 Ultrahigh Speed Op Amp Data Sheet*. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad8009.pdf

[12] C. Inc., *XLamp XP-E LED Data Sheet*. [Online]. Available: https://www.cree.com/led-components/media/documents/XLampXPE-25A.pdf

[13] I. M. S. Inc., *M-Drive 14 Plus Motion Control*. [Online]. Available: https://imshome.com/downloads/manuals/mdi_14.pdf

[14] A. Ronacher, *Flask Reference Page*. [Online]. Available: http://flask.pocoo.org/

[15] C. Liechti, *PySerial Documentation*. [Online]. Available: https://pythonhosted.org/pyserial/

[16] K. McConkey, *Servo Serial Interface GitHub Page*. [Online]. Available: https://github.com/KeenanMcConkey/Servo-Serial-Interface

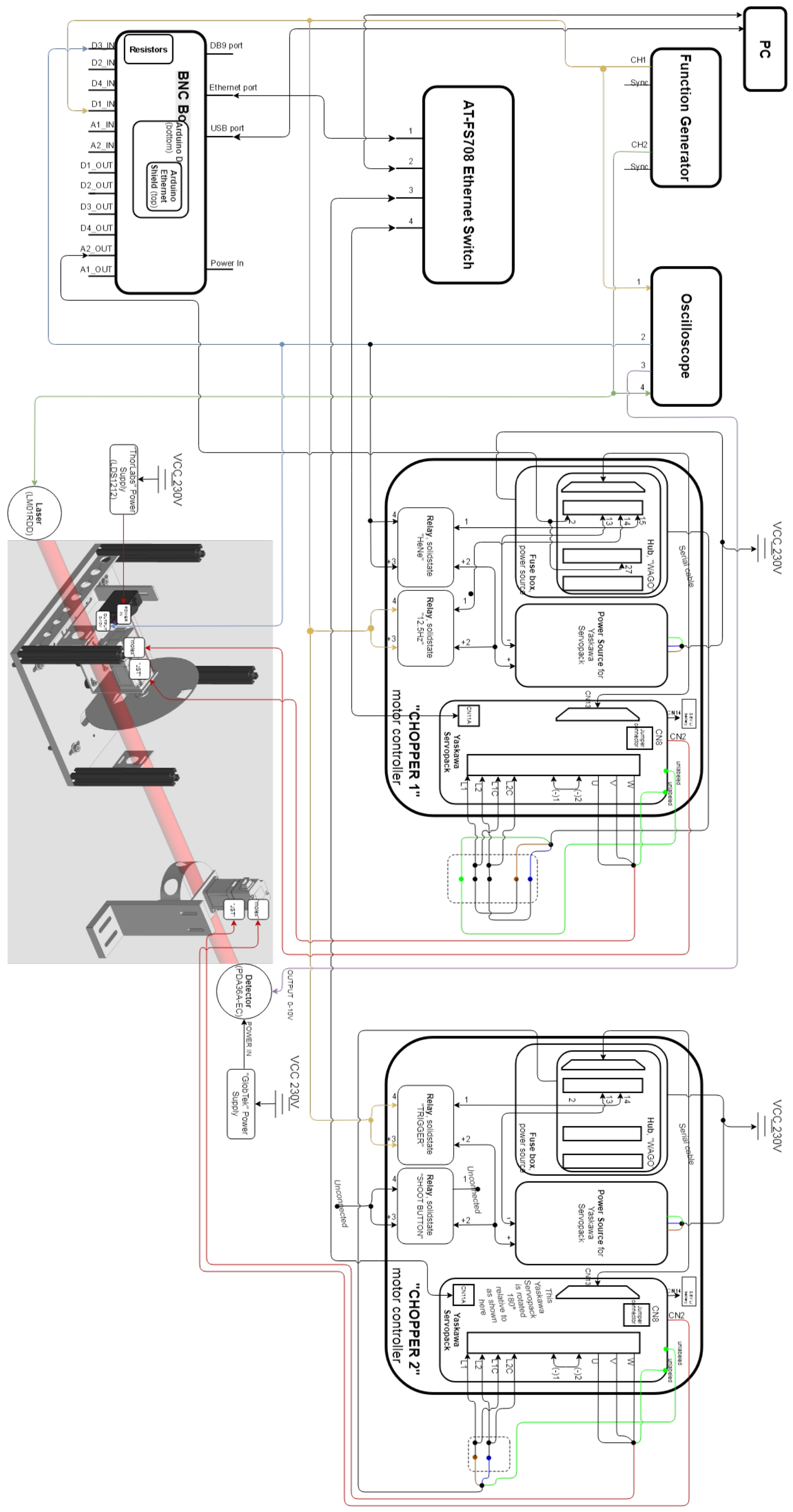


Figure 21: Full operational schematic of Chopper System

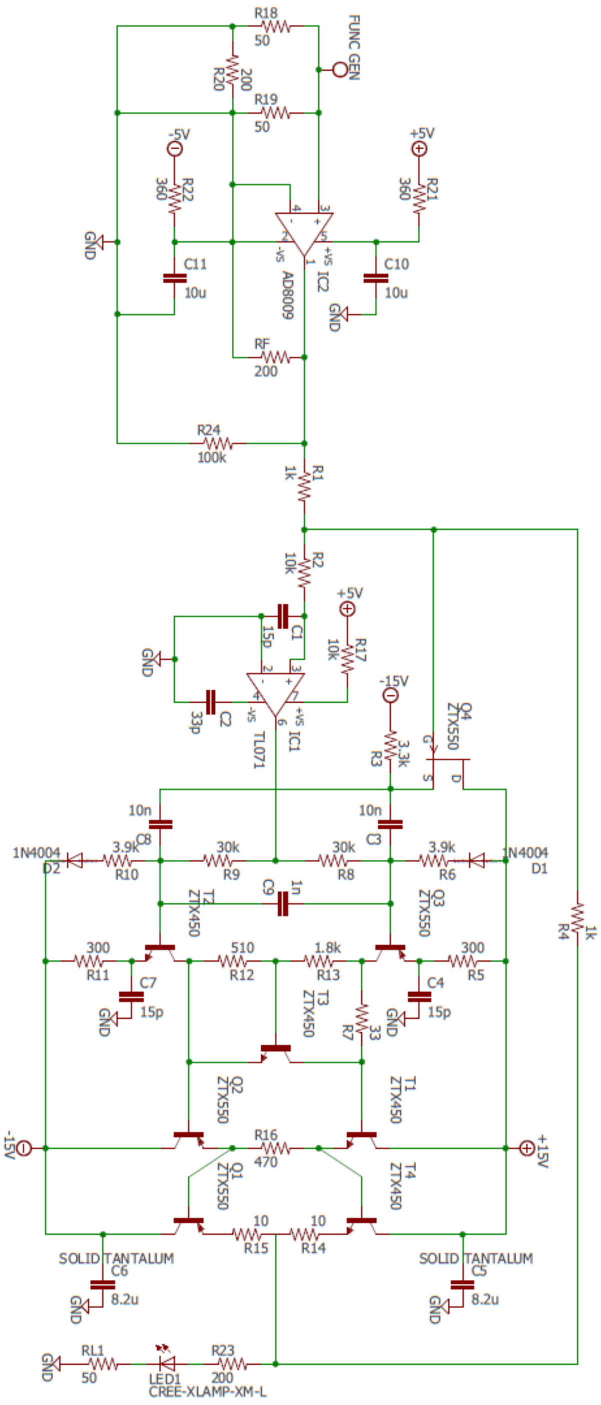


Figure 22: Circuit to generate nanosecond-scale LED pulses

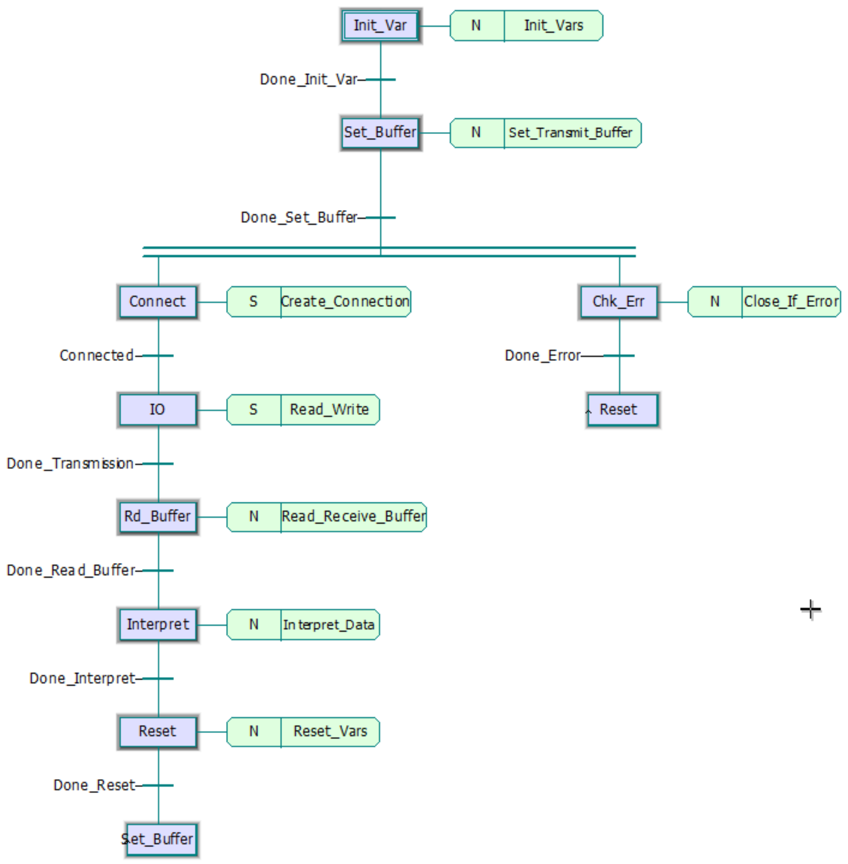


Figure 23: Ethernet IO SFC

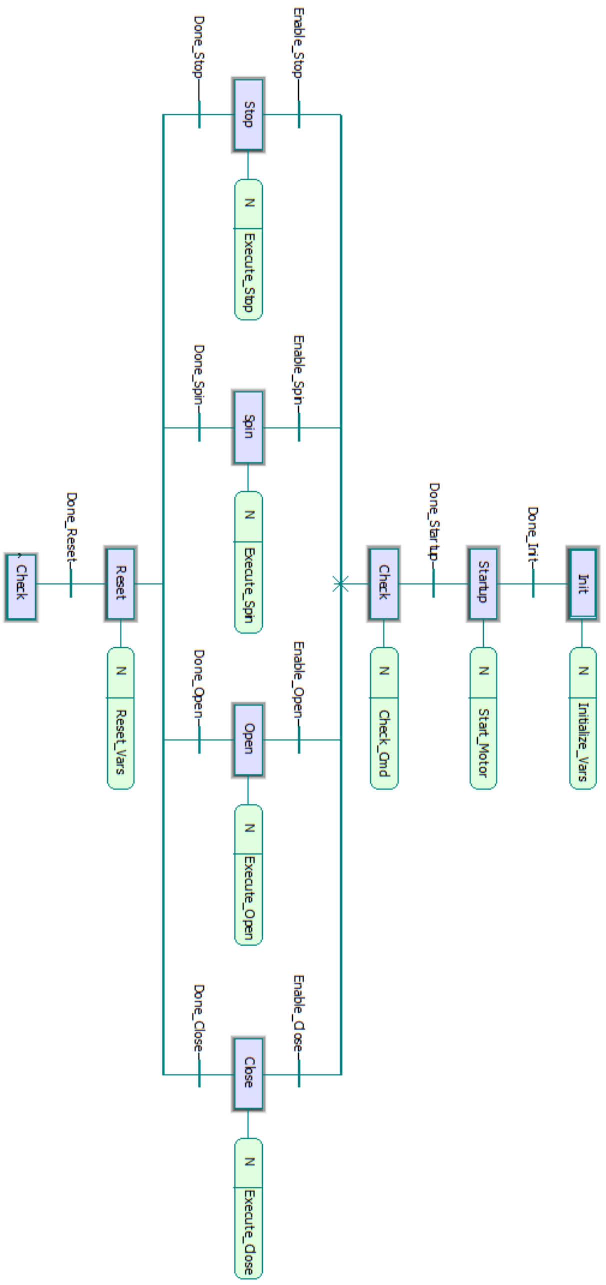


Figure 24: Chopper 1 SFC

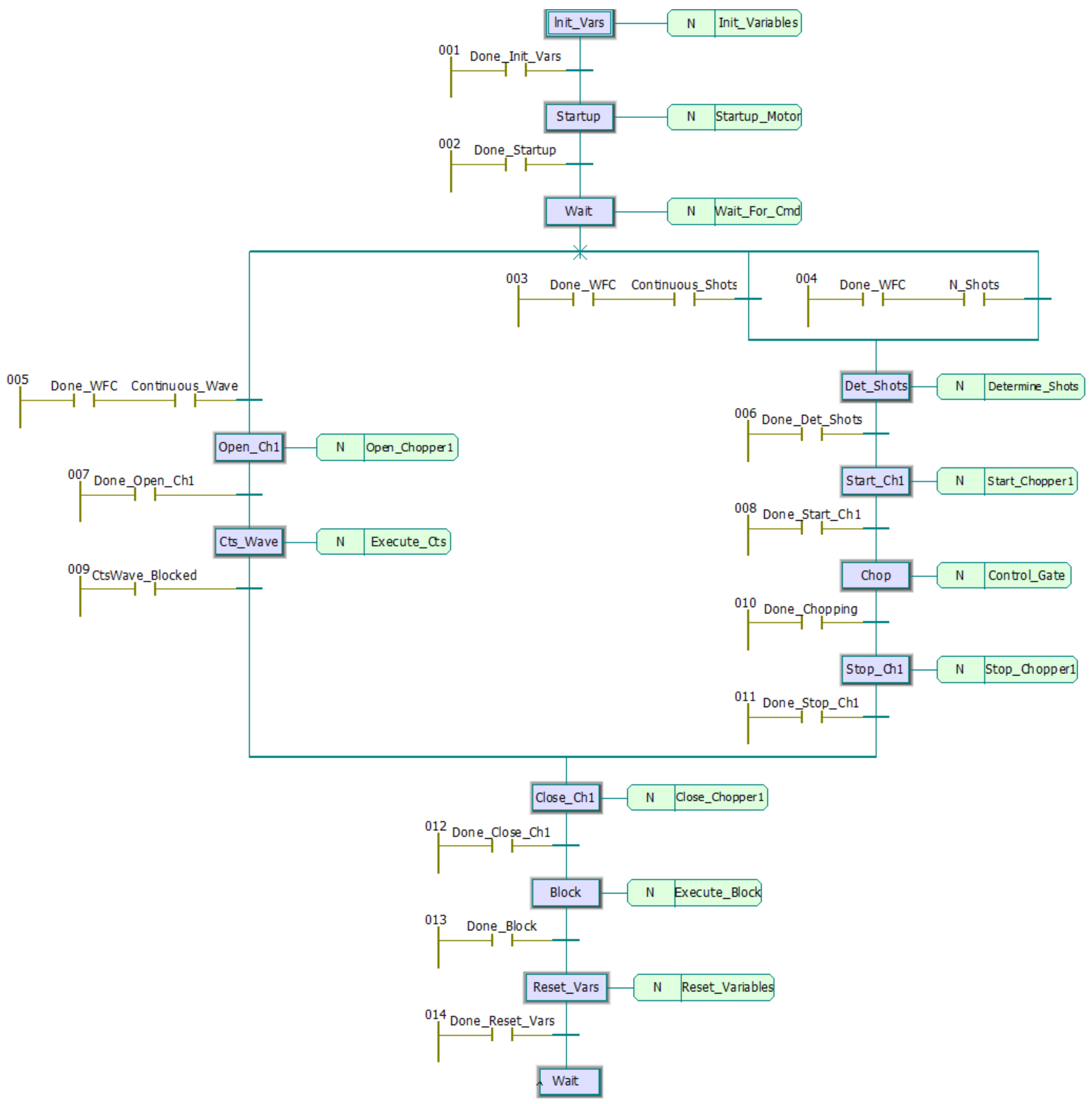


Figure 25: Chopper 2 SFC