The Design and Implementation of a Scanning Fabry-Perot Interferometer for Use in Measuring Multiple Modes in Lasers

David M Houston

***Abstract*—A scanning mirror Fabry-Perot Interferometer (sFPI) was designed, built, and implemented to measure the modes of a laser beam. It has a finesse of 155 and free spectral range of 3.75 GHz with a resolution bandwidth of 24 MHz. A piezoelectric device will change the spacing of the mirrors, allowing scanning across multiple free spectral ranges. A voltage-source driver provides a controllable signal to a piezoelectric device. The spectral frequency content will be captured by a photodetector and amplified by a transimpedance amplifier. This can be analyzed one of two ways. Either sent to a micro-controller-computer pair which processes, displays, and stores this data locally on a hard disk, or sampled by an oscilloscope which can take the place of the mircocontroller.**

***Index Terms*—FSR, sFPI, Python**

I. Introduction

The current sFPI (Spectra-Physics SP-470-06) is bulky and requires various appendages in order to operate. It requires a separate photodiode and data collection device (i.e. oscilloscope). This makes it cumbersome to use. The new model has been constructed to reduce the overall real estate and make the operation more user friendly. It has a pre-installed photodiode as well as an integrated application which communicates directly with an onboard microcontroller to process and display the spectral content of the cavity. The application increases the functionality of the device by calibrating the incoming signal based on user input of the free spectral range of the cavity which prompts the application to mathematically scale the output for analysis. While the newly development sFPI seems to be superior to the other it has its disadvantages. The frequency of the driver is only 2.35 Hz, making it much slower than real-time analysis.

II. Background

In order to understand how the sFPI works, a fundamental principle must be explained. In its static state, a cavity[1] can hold a specific number longitudinal modes, proportional to the length of the cavity (L). Verheyen's book on laser electronics gives an adequate definition for a cavity mode:

> A cavity mode is a field distribution that reproduces itself in relative shape and in relative phase after a round trip through the system [5]

Each mode has an order of the form *(m,n,q)*, where *m, n,* and *q* are integers. The transverse modes are represented by *m* and *n*, and the longitudinal modes are represented by q. Typically only the $TEM_{0,0,q}$ modes are examined because it makes the math easier, however higher order transverse modes are prevalent in the system. For our purposes we will only be examining the longitudinal modes. However, when the length of the cavity changes (ΔL), so does the orders of the modes which can occupy length $L + \Delta L$. In order to achieve a maximum transmission, ΔL must be on the order of a wavelength, corresponding to a q+1 increase in the mode order. Any other change in length (ΔL) not on that order will result in little to no transmission of the cavity beam. The separation of the maximum transmission peaks is defined as the free spectral range (FSR) and can be expressed by FSR $= c/2L$, where *c* is the speed of light in a vacuum. The FSR is described by a frequency, however it can also be related to spacial frequency as well. Figure **??** graphically describes the free spectral range.

Since this is not a perfect cavity, the photon-lifetime is described by an exponential decay rate, $e^{-t/\tau}$. The line shape of the cavity, which has the shape of a lorentzian, is the Fourier transform of the photon-lifetime. The line shape is the transmission

[1] Define what a cavity is

intensity of the deviations of frequencies from the fundamental. The full-width at half maximum of the line shape is the bandwidth of the cavity. Typically bandwidth is not expressed, but rather the finesse of the cavity. The finesse is defined as the ratio of the free spectral range to the bandwidth of the cavity (Eq. 1).

$$\mathcal{F} = \frac{\pi}{2sin^{-1}(1/\sqrt{F})} \tag{1}$$

The finesse is related by a factor defined as the coefficient of finesse (F) which is proportional to reflectivity of the mirrors used (Eq. 2).

$$F = \frac{4R}{(1-R)^2} \tag{2}$$

In order to increase the finesse and overall performance of the sFPI cavity, dielectrically coated mirrors with high reflectivity at the principle wavelength are used. The reflectivity of these mirrors is typically $> 90\%$. Since the reflectivity is greater than 50% the simplified Eq. 3 can be used to approximate the finesse. This is because at small values of x for $sin^{-1}(x)$ the relationship is 1 to 1.

$$\mathcal{F} = \frac{\pi R^{1/2}}{1-R} \tag{3}$$

We have discussed what happens when we change the length of the cavity, but we must further dive into how to move the cavity and by how much. We know that we must change the cavity length by a factor of a wavelength in order to have maximum transmission in the cavity. This only insinuates that $\Delta L = n\lambda$, where $\lambda = 632.8nm$. In order to generate a ΔL of that magnitude requires a piezoelectric device (PED). While other forms of precision motion control can be used (i.e. pressurized air), a PED's real estate is ideal for the application. A PED's deformation is directly proportional to the electric field applied across it and is related by the following equation where x is the strain of the PED, d_{33} is the piezoelectric strain constant, and E is the applied electric field [4].

$$x = d_{33}E \tag{4}$$

Since strain (x) is the ratio of the change in length by the length ($\Delta L/L$) of the material and the applied electric (E) is the ratio of the applied voltage (V) to the length (L) over which it is applied (V/L), Equation 4 can be expressed as $\Delta L = d_{33}V$, directly relating the change in length to the change in applied voltage. If you have any doubts, the units work out to meters, given that the units of d_{33} are pC/N [4]. For our applications a sawtooth voltage signal will be applied across the PZT. This will allow for scanning of the mirrors through the modes of the resonant cavity by systematically controlling ΔL for reproducible output.

The transmission intensity resulting from the scanning of the cavity by means of the PZT is incident on a photodiode. Photodiodes have a unique characteristic where light incident upon the semiconductor ($h\nu$) produces a current. This can be done in the photovoltaic or photoconductive regions of operation; however, the photoconductive region, which operates in reverse bias, has a much higher responsivity. The responsivity of a photodiode (R_λ) is the ratio of current (I_{P_n}) in amperes to the incident power (P_n) in watts, where *n* is an integer describing the different levels of incident power (Fig 1). The responsivity can be calculated by Equation 5, where η is the quantum efficiency, h is Plank's constant, ν is the frequency of the light, and e is the elementary charge **??**

$$R = \eta\frac{e}{h\nu} \tag{5}$$

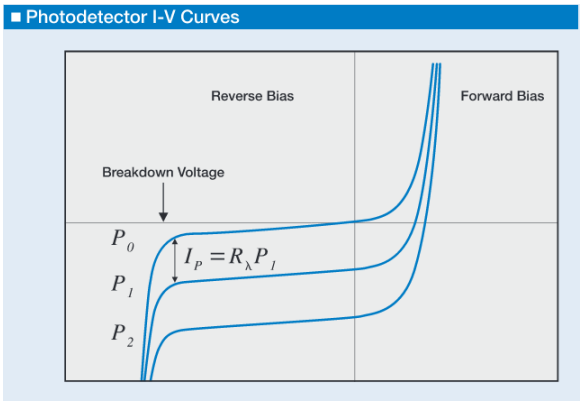


Fig. 1. Photoconductive Diode Curve [3]

A transimpedance amplifier can be implemented to convert the current signal into a voltage signal making it quite easy to collect the resulting data using an analog-to-digital converter. However, since the analog-to-digital converter has a limit on the conversion rate, the frequency of the PZT must match this rate to achieve maximum resolution.

III. sFPI Description

The scanning Fabry-Perot interferometer as discussed in Section II, the optical design, the mechanical design, and the electrical design are the three key design parameters. The cavity design was chosen based on the typical construction for a Fabry-Perot interferometer. The mechanical design was ... The electrical design was based on the operation of the Spectra Physics Scanning Interferometer Driver which consisted of a signal generator with the ability to change the .

IV. Optical Design

The optical design component, as illustrated in the block diagram, is a Fabry-Perot interferometer. This resonant cavity consists of two spherical mirrors with a radius of curvature equal to 40 mm $\pm$ 2 mm. In order to have a stable cavity the following equation applies.

$$0 \leq \left(1 - \frac{L}{R_1}\right)\left(1 - \frac{L}{R_2}\right) \leq 1 \tag{6}$$

Given that $R_1 = R_2 = 40$ mm $\pm$ 2 mm, the range of the length of the cavity (L), for a stable resonator, is:

$$0 \leq L \leq 40 \text{ mm} \pm 2 \text{ mm} \tag{7}$$

The surface of the mirrors are coated with a dielectric material which has a reflectivity ranging from $99.4 - 99.8\%$ for 632.8 nm light. The finesse of this cavity, as explained in Section II, is expected to be within the range of $522 - 1569$ for the respective reflectivity. The resonant cavity for the device is represented in Figure **??**.

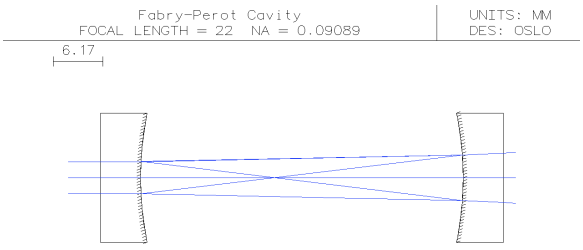


Fig. 2. The resonant cavity is comprised of two dielectrically coated spherical mirrors, which reflect 632.8 nm light with 98% reflectivity

An 75 mm PCX lens is used to couple the system and account for any misalignment. By adding in the auxiliary lens the spot size of the output is decreased as well as the divergence angle of the gaussian beam leaving the cavity. Figures 3 and 4 show the direct improvement of adding in an auxiliary lens to the system. This is done assuming a confocal resonant cavity with the focal point of the auxiliary lens matching the center of the resonant cavity.

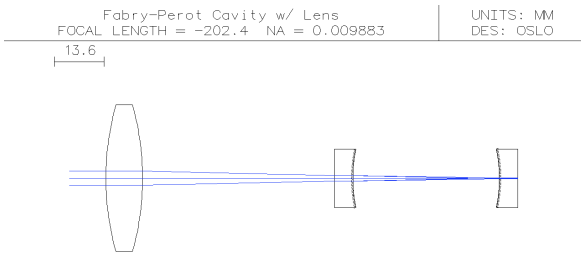


Fig. 5. The auxiliary lens couples the incoming beam to the resonant cavity by increasing the photon-lifetime of the cavity

V. Mechanical Design

The first surface mirror is mounted in a milled disk which is counter sunk, to keep the mirror from falling out (see Figure 6). This is held in place by a set screw.

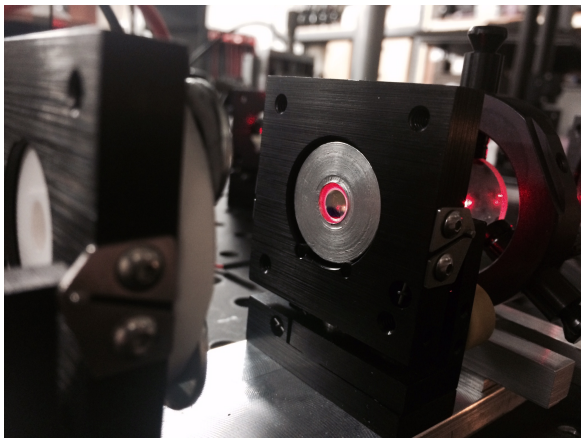


Fig. 6. Milled Disk for first surface mirror

The second surface mirror is attached to the piezo-electric device, which is described in more detail in the next subsection. This is done by the clear adhesive Gorilla Glue. The piezo-mirror combination is mounted to a plastic appendage (Figure 8). This plastic appendage is designed to keep the limiting aperture within the cavity, when properly aligned. It is designed such that 4 washers hold the Piezo-Mirror combination in place, without

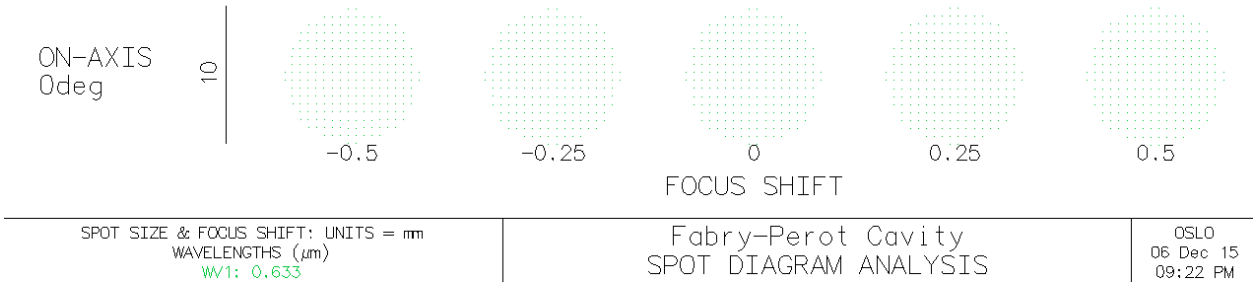


Fig. 3. The spot size of the light exiting the resonant cavity after 1 pass

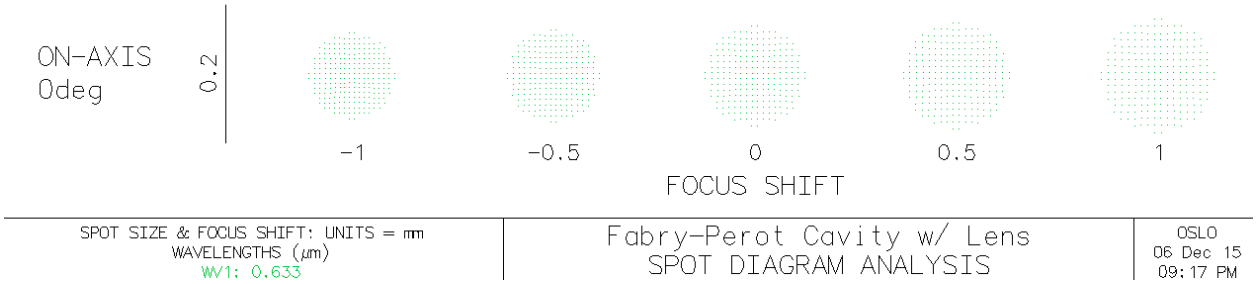


Fig. 4. The spot size of the light exiting the resonant cavity after 1 pass with an auxiliary lens of 75 mm focal length

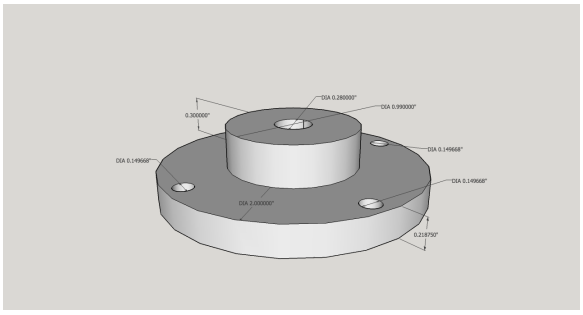


Fig. 7. Piezo-Mirror Holder

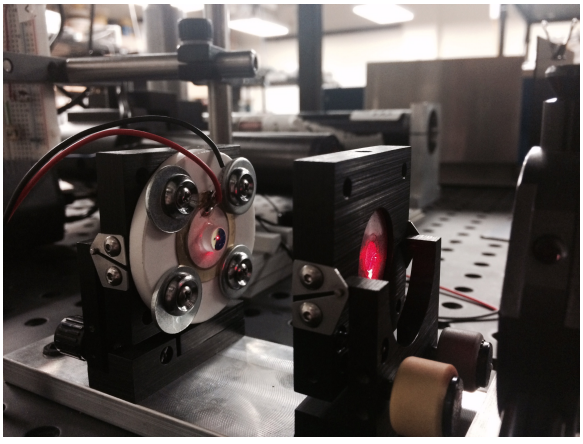


Fig. 8. Piezo-Mirror Holder Dimensions

grounding the plates, and so that it fits inside the adjust mounts (see Figure 7).

The entire cavity is held in place by dual-axis adjustable mounts for aligning purposes (See Figure 10). These are attached to an aluminum base with adjustable distances between the mirrors (Figure **??**).

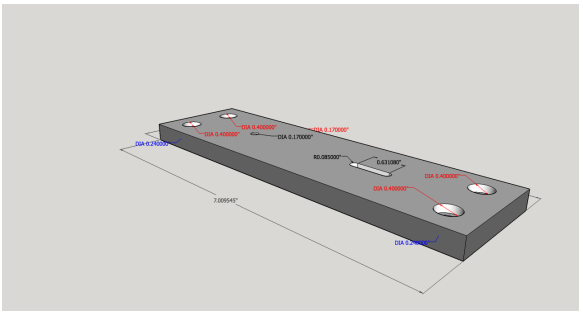


Fig. 9. Aluminum base for the sFPI cavity system. Full layout drawings with measurements can be found in on page 10

The photodiode is mounted on the back of the adjustable mounts (Figure **??**) by the specially made holder (Figure 11). A set screw is taped post-3D-print in order to secure the photodiode. This is done with a 4-40 drill and tap for a 4-40 set screw.

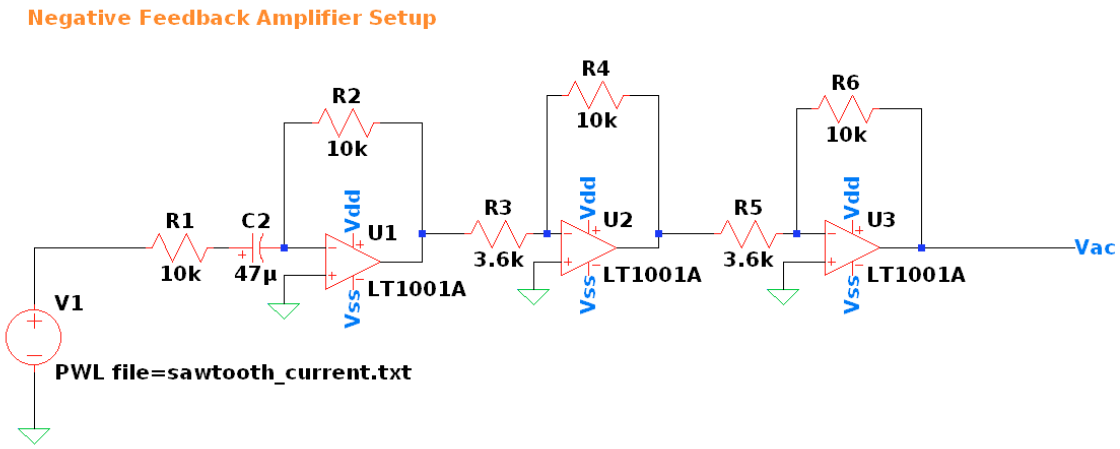


Fig. 12. The three-stage negative feedback amplifier configuration provides a range of 0.2 V/V up to 7.716 V/V of amplification to the input signal by changing resistors R3 and R5 from 3.6 $k\Omega$ to 10 $k\Omega$. V_{DD} and V_{SS} are the respective positive and negative voltage rails +17V and -17V. For more clarification the full LTSpice model is Figure **??** in Section F

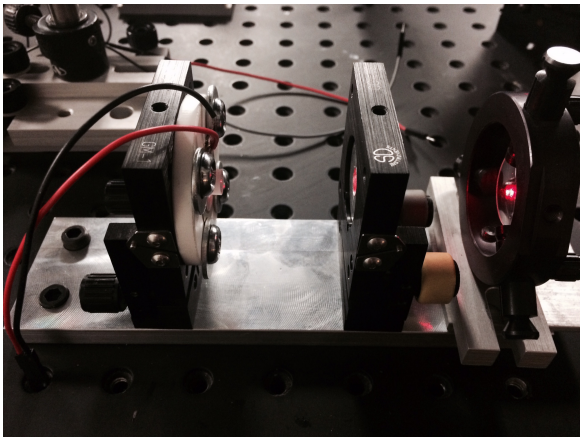


Fig. 10. Adjustable mounts for entire cavity setup

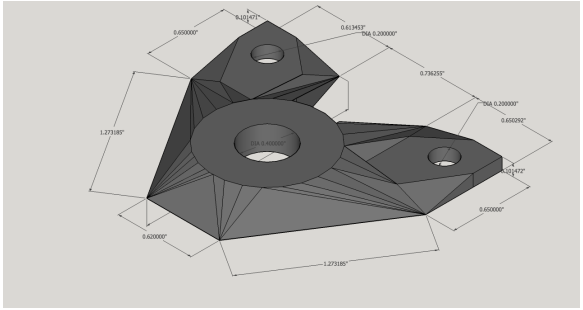


Fig. 11. The mounting device for the photodiode with all measurements and outlays specified on page 11

VI. Interferometer Driver Design

A. Function Generator

The micro-controller for the driver setup is constructed using an Arduino UNO (see Data Sheet on page 50). The Arduino UNO has firmware in which it communicates with a MCP4725 (see Data Sheet on page 46) 12-bit Digital-to-Analog Converter (DAC) via the I2C communication protocol (see Figure **??**).The DAC attenuates the +5V 10% voltage supply from the Arduino UNO with 2^12 individual output values. A peak-to-peak sawtooth voltage signal is the result, with a bit resolution of 1.2 mV/bit at a frequency of 2.5 Hz (see Figure 13). The MCP4725 signal-generator acts as a near-perfect voltage source with a series impedance of 1Ω when in normal operations. This will not affect input resistances for the analog design.

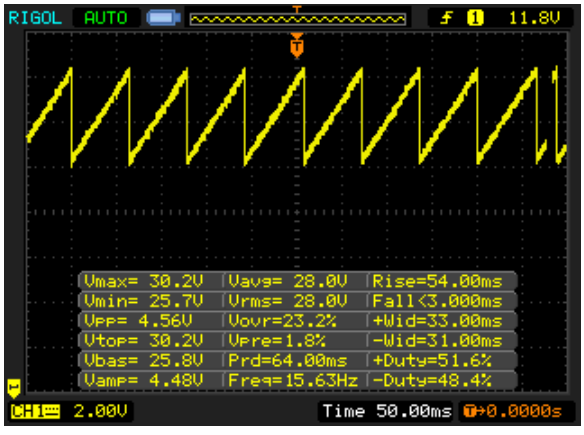


Fig. 13. The output of the MCP4725 DAC from the I2C communication with the Arduino which has 4096 individual steps from $0 - 5 \pm 10\% V$

B. Analog-Amplifier Circuit

The sawtooth waveform is sourced to a three-stage negative feedback amplifier configuration (Figure 16). The amplification (A_v) is adjustable from $0.2V/V$ up to $7.716V/V$. This is done through three stages: the dispersion stage and the two magnitude stages. The dispersion stage allows for small adjustments, 0.2x to 1x of the original signal based on $R_{in_{min}} = 10k\Omega$ and $R_{in_{max}} = 50k\Omega$. The magnitude stage allows for 1x to 3x of the original signal based on $R_{in_{min}} = 3.6k\Omega$ and $R_{in_{max}} = 10k\Omega$.

Resistors R1, R2, and R3 are representation of an array of resistors. Figure 14 details the configuration of R1, which is the adjustable input resistance of the dispersion stage of the amplifier.

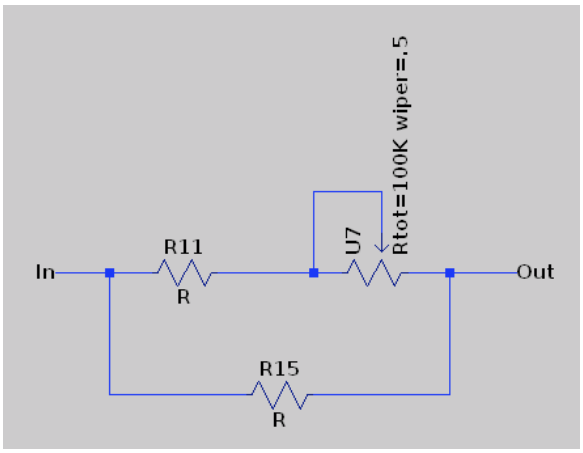


Fig. 14. This is the array of resistors which

has which are complimented by a 100k potentiometers having an effective range of $0-100k\Omega$This increases the AC range from 5 volts to 34 volts. Adjustment of this gain is done by a series of potentiometers governed by Eq. 8.

$$A_v = (R_f)^3\left(\frac{1}{R_1}\right)\left(\frac{1}{R_2}\right)\left(\frac{1}{R_3}\right) \tag{8}$$

A LTSpice Netlist program was developed (see code on page **??**). This provided the expected output of the signal given a specified input signal. This input signal is generated by a python-shell script which generates a PWL file for LTSpice based on a series of user inputs (see code on page **??**).

This is sourced to a buffer which uses a single NTE941M (see Datasheet on page 36) operation amplifier. This gives the AC signal an infinite output impedance, preventing impedance mismatching. Figure 15 denotes the circuit construction.

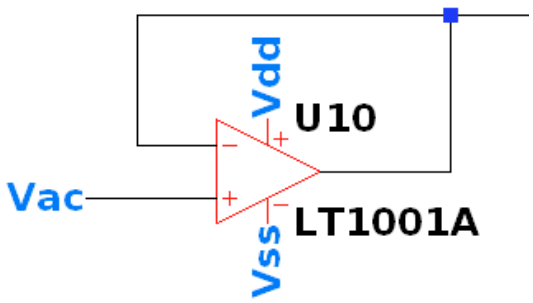


Fig. 15. The voltage follower (unity buffer amplifier) configuration for an OpAmp for circuit isolation and prevention of impedance mismatching.

To change the offset biasing of the sawtooth voltage signal an NTE957 (see Datasheet on page 38) adjustable DC voltage regulator is added using a summation circuit to the AC signal with the following configuration (Figure **??**).

C. Power Distribution

A VELLEMAN PSINO2512N 12-VOLT 25-WATT DC SWITCHING POWER SUPPLY (see Datasheet on page **??**) was complimented with a DROK Micro Electric DC/DC Boost Converter LM2577 Step-up Voltage Transformer (see Datasheet on page **??** for specifications of the DROK component and on page **??** for the LM2577 data sheet) to provide the power to the system. A NTE1234 5 VDC $\pm 1\%$ fixed voltage regulator (see Datasheet on page 36) and an MAX737 inverting buck-boost converter (see Datasheet on page **??**).

The 12VDC PSIN02512N voltage supply is sourced to LM2577, boosting the voltage to 17 VDC. The 17 VDC is sourced to the NTE1234 5-VDC producing +5 volts suppling power to the boot-loaded Arduino and the MCP4725 DAC.

Since the design calls for a signal with a max peak-to-peak voltage of approximately 34V, the rails must have a range +17 VDC to -17 VDC. Since the constraint that $V_s - V_out \leq 22V_DC$ holds for the buck converter, if the 5VDC rail is used, $V_{out_{max}} = -17V$ [**?**]. This is achieved by using values for $R_3 = 92\ k\Omega$ and $R_4 = 1.2\ M\Omega$ (Figure **??**).

The current design of the board, which will be discussed later in this section, was simulated in LTspice for power consumption calculations. The

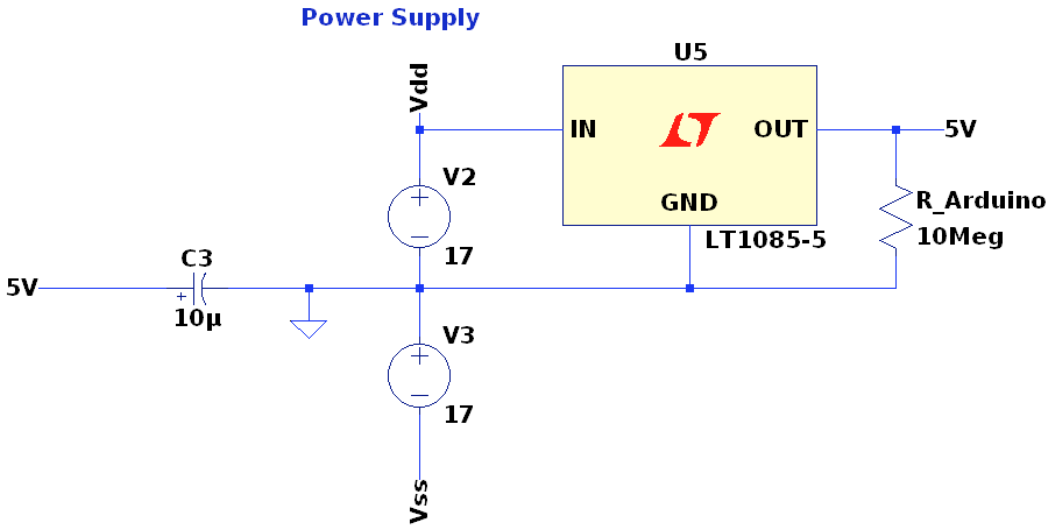


Fig. 16. The three-stage negative feedback amplifier configuration provides a range of 0.2 V/V up to 7.716 V/V of amplification to the input signal by changing resistors R3 and R5 from 3.6 $k\Omega$ to 10 $k\Omega$. V_{DD} and V_{SS} are the respective positive and negative voltage rails +17V and -17V. For more clarification the full LTSpice model is Figure **??** in Section F

negative power supply was sinking 22.69 mA of current. In order to calculated the range of inductance necessary for operations Eqs 9 and 10 were used.

$$I_{pk} = \frac{4(V_{out} + V_{DIODE})}{V_{in} - V_{sw}} * I_{out} \tag{9}$$

$$L = \frac{V_{in} - V_{sw}}{I_{pk}} * t_{on} \tag{10}$$

I_{pk} was found to be 426.8 mA. This is significantly less than the max current rating of 525 mA; therefore a current source is not necessary to complete this setup. Given that the switching time (t_{on}) for the converter is roughly 10 μs the range of inductors was found to be $73.8\ \mu H \leq L \leq 105\ \mu H$.

VII. Analog Data Conversion Processor

The photodiode is mounted to an adjustable plate at the rear of the sFPI for calibration, for maximum transmission of the spectral content. The photodiodes resulting current is reverse biased by an Arduino UNO (#2), which is supplying the 5V. Based on the IV curve of the photodiode, the voltage stays within the photoconductive range while remaining far enough away from the breakdown voltage.

A recent Ph.D. thesis details the quantum efficiency of doped GaAs being between 30% and 22% [1]. This means at 632.8 nm a good approximation for the responsivity is 154 mA/W. The incident power on the detector can be approximated based on the losses in the cavity. Section IV explained that the reflectivity of the mirrors was between $99.4\% - 99.8\%$. From this the transmitted power of the beam can be calculated for one pass through the cavity:

$$P_{out} = T_1 T_2 * P_{in} \tag{11}$$

Given the ranges for reflectivity given above, that $T = 1 - R$, and $P_{in} = 3mW$, the output power of the cavity was found to be $P_{max} = 0.108\mu W$ and $P_{min} = 120nW$. With an optical output of approximately 1 μW results in a current of roughly $0.154\ \mu A$.

A transimpedance negative-feedback amplifier with a voltage booster (TAV) converts the AC-current signal to a AC-voltage signal. The voltage gain of this amplifier setup is adjustable for tuning the sFPI, when the signal-to-noise ratio is small, to increase the intensity of the output of the cavity (Figure 30). This corresponding output is sampled

by an analog-to-digital converter (ADC). Since the Arduino, the micro-controller used for storing the data, uses a pre-scaler of 128, the ADC clock speed is set at 125 kHz. Given that it takes 13 ADC clocks for a conversion the ADC can sample data at a rate of 9615 Hz. Since our we have 4096 individual steps, to match the rate of the ADC a frequency of 2.35 Hz in used for the driver.

VIII. Digitally Analyzed Output

The content captured by the ADC (Figure 30) is transmitted to a computer to process, display (by a graphical representation), and stores the data by user command.

A. ADC-Computer Interface

The voltage signal from the TAV is sampled by the Arduino Uno, as discussed in Section VII. This is converted to a 512 byte array. The array is then encoded to send to the computer upon request (see code in Section B-B).

B. Data Analysis

This data will be combined in conjunction with the 5V sawtooth signal to plot the photodiode signal as a function of the geometrical change of the resonant cavity, which is proportional to the wavelength of the light. This will be done via a GUI model with a language that possesses the capability for universal interfacing. Python will be the base model for programming; however, other languages will be tested for user interface and compared based on speed, cross-platform configuration, and simplicity.

IX. Discussion

In order to determine the quality of the sFPI data was collected, analyzed and compared with that from the SP-470-04 Interferometer for comparison. (Show data and the conclusions drawn from it).

X. Conclusion

Throughout this document we described a low-cost version of a sFPI. (Talk more, once more data is collected).

References

[1] Ph.D. Gui, Gui. *Study of graphene bandgap engineering and betavoltaic and optoelectronic devices*. PhD thesis, THE UNIVERSITY OF WISCONSIN - MADISON, 2014.

[2] Igor A. Litvin. Stability of a laser cavity with non-parabolic phase transformation elements. *Optical Society of America*, 21(9), 2013.

[3] OSI Optoelectronics. *Photodiode Characteristics and Applications*.

[4] Kenji Uchino. Introduction to piezoelectric actuators and transducers. 25 1, Penn State University, 1986.

[5] Joseph T. Verdeyen. *Laser Electronics*. Prentice-Hall, Inc., 1981.

APPENDIX A
MECHANICAL ASSEMBLY DRAWINGS

A. PZT-Mirror Holder

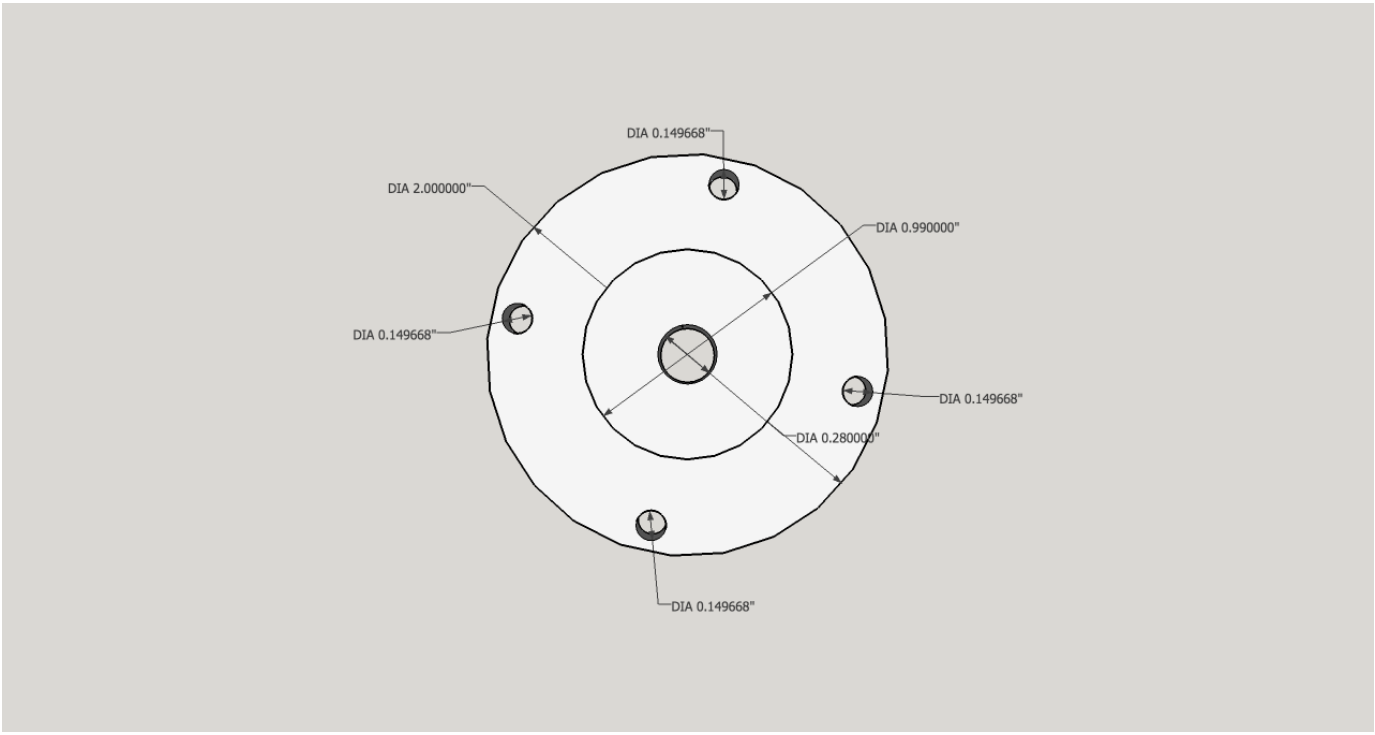


Fig. 17. The top down view and measurements of the PZT-Mirror holder

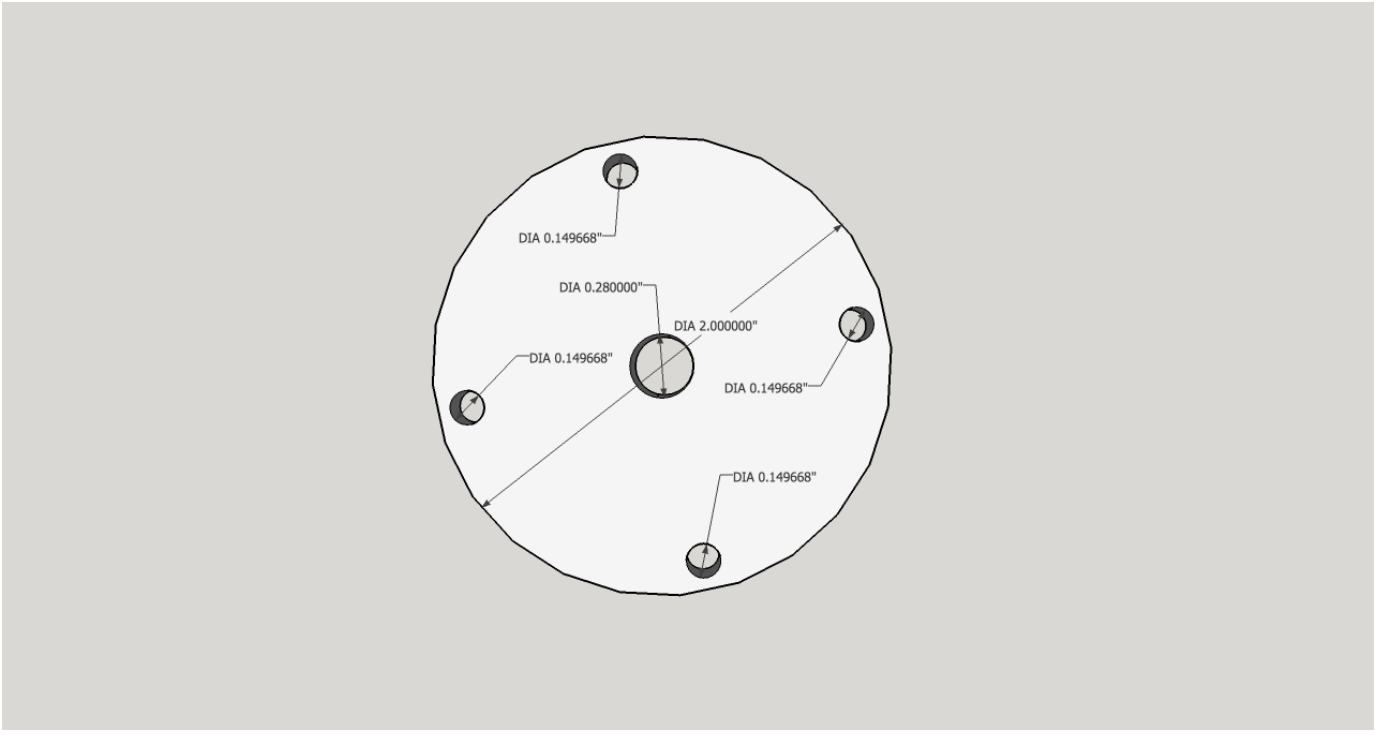


Fig. 18. The bottom up view and measurements of the PZT-Mirror holder

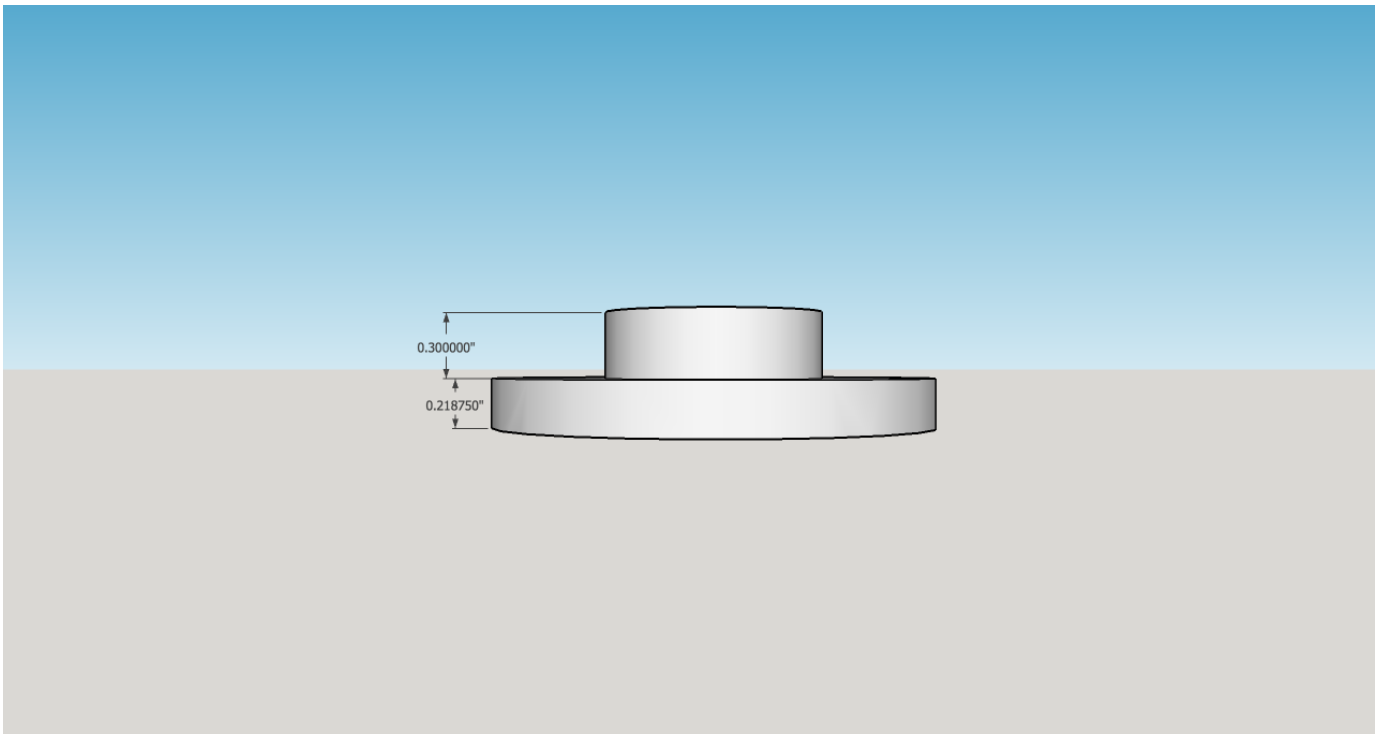


Fig. 19. The side view and measurements of the PZT-Mirror holder

B. sFPI Aluminum Base

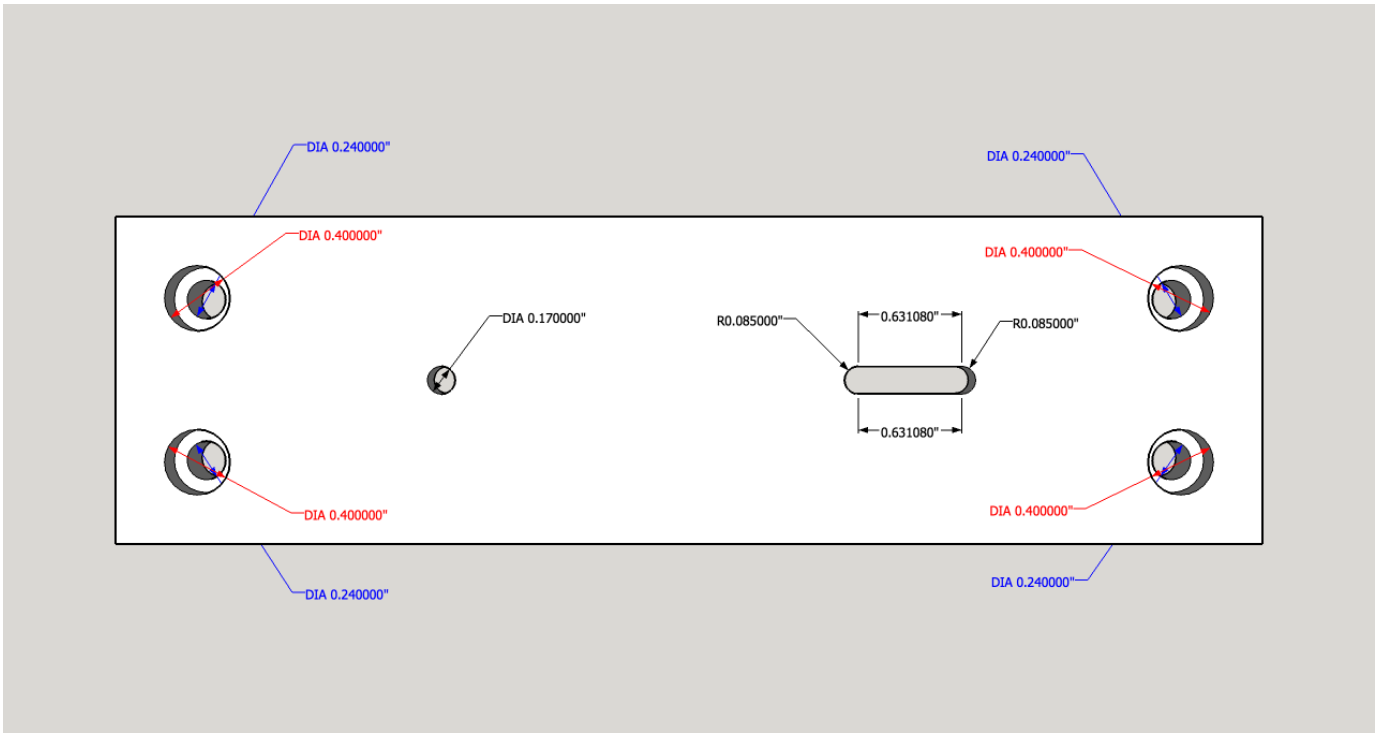


Fig. 20. The top down view and measurements of the sFPI aluminum base

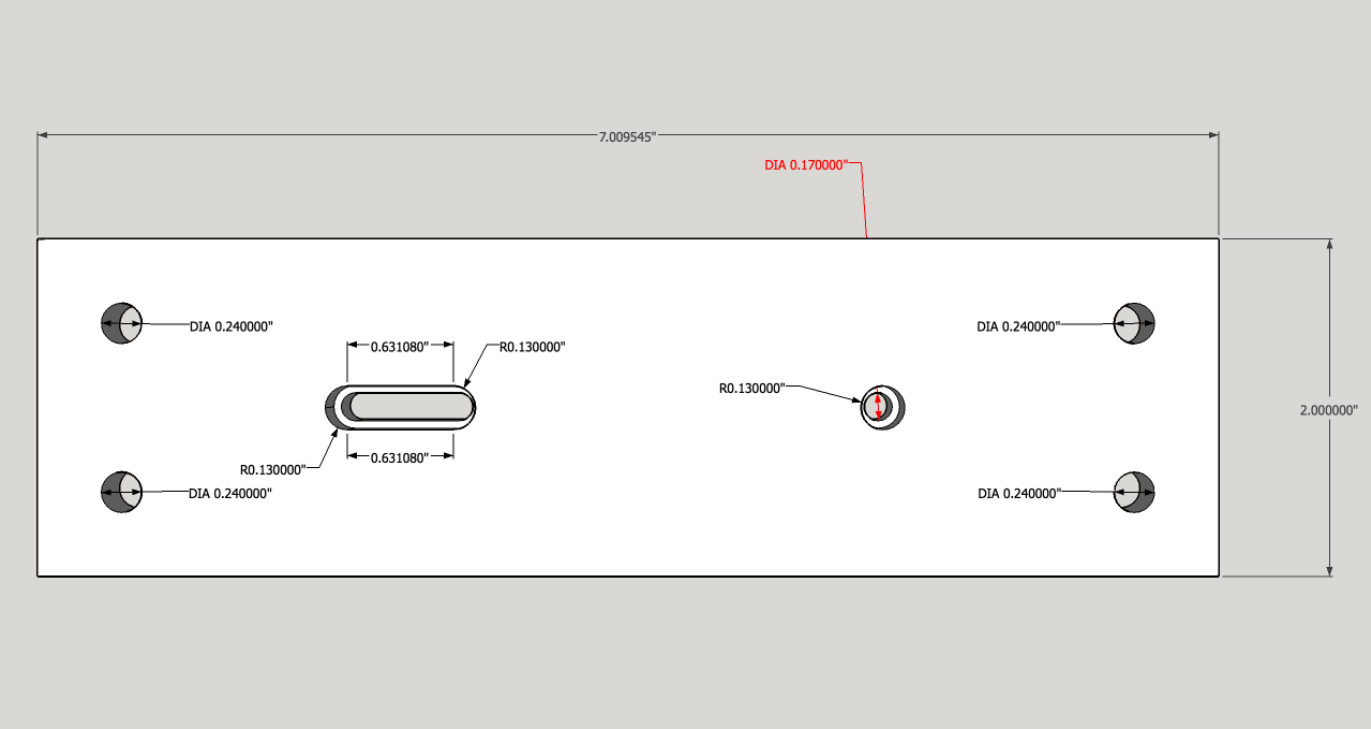


Fig. 21. The bottom up view and measurements of the sFPI aluminum base

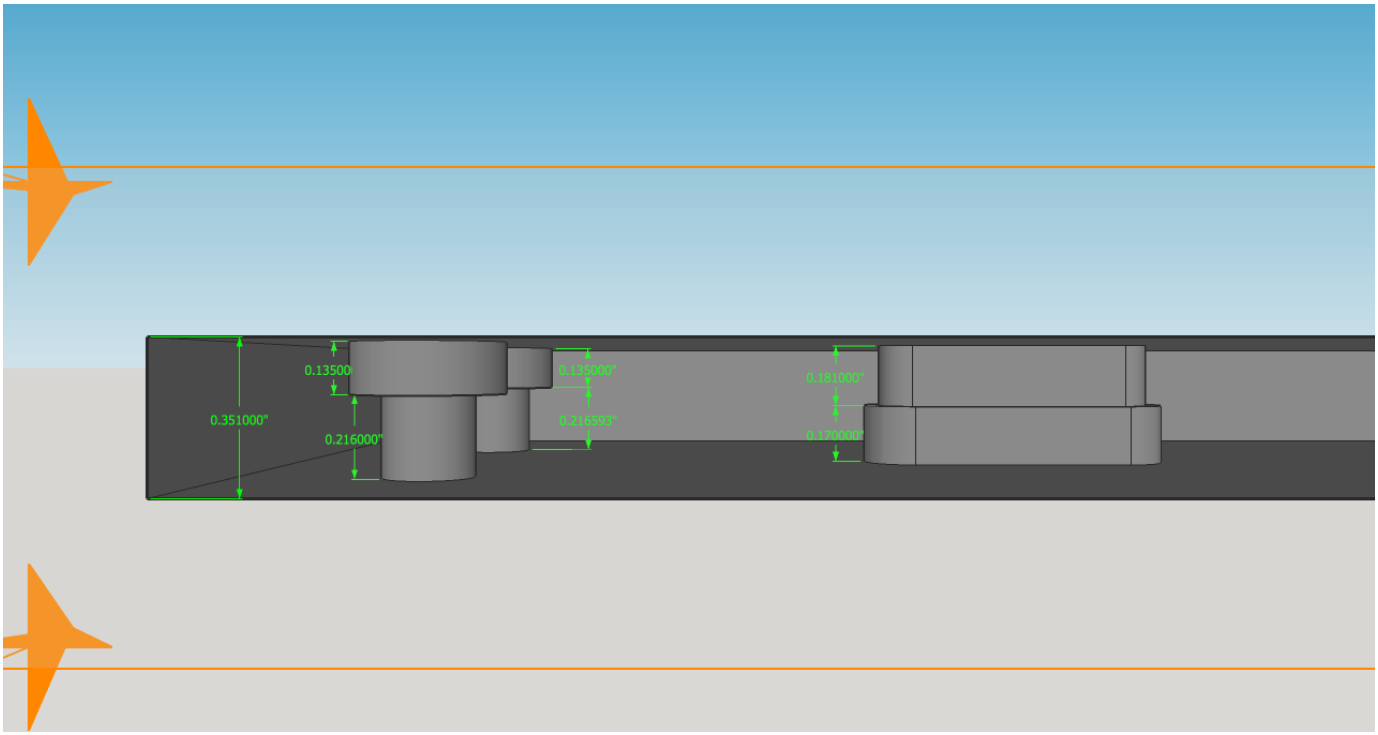


Fig. 22. The front cross section view and measurements of the sFPI aluminum base

C. Photodiode Holder

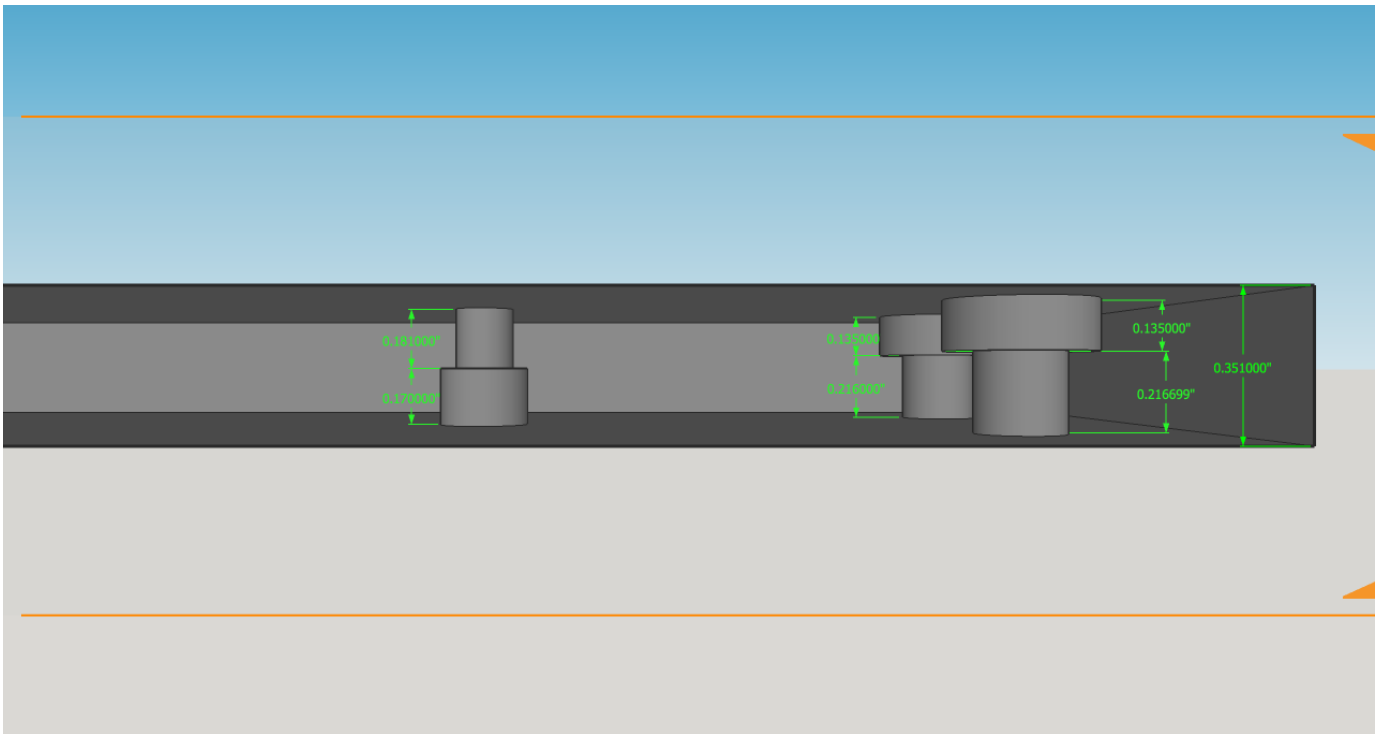


Fig. 23. The back cross section view and measurements of the sFPI aluminum base

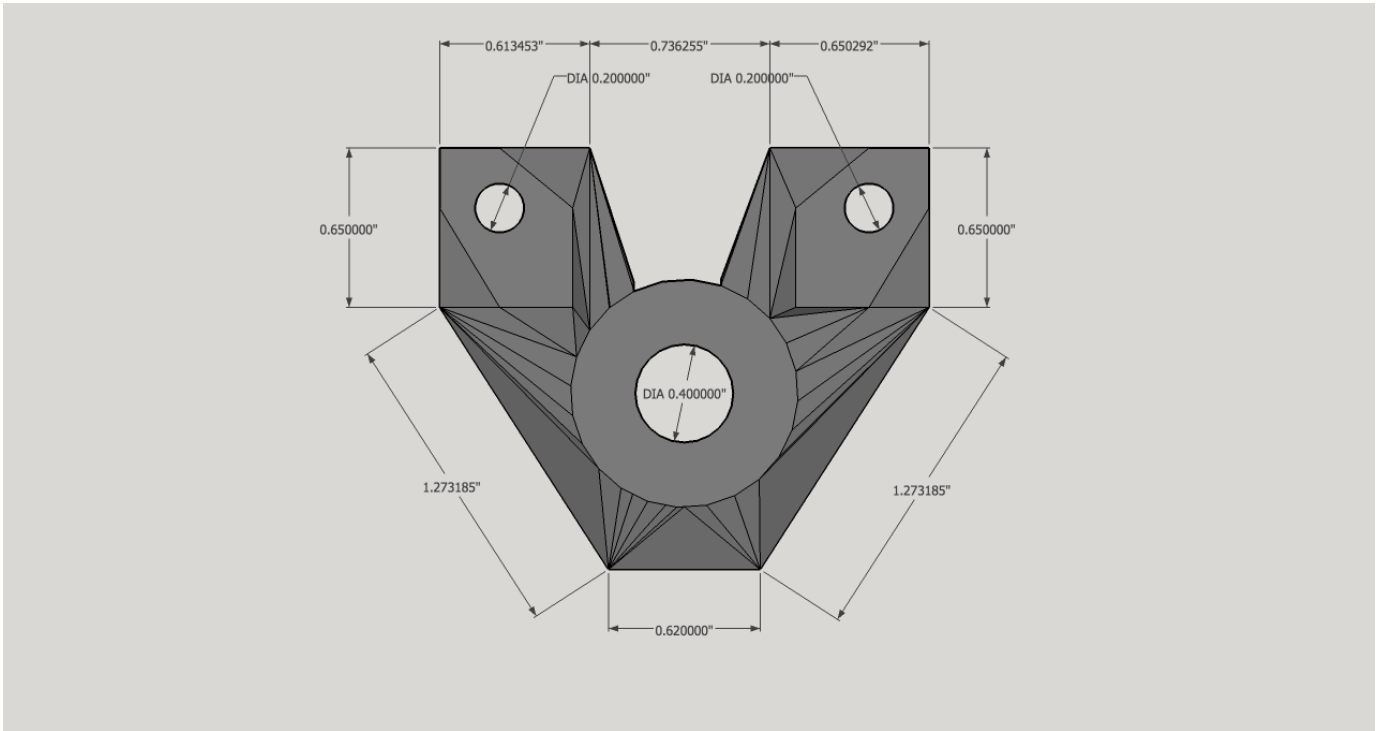


Fig. 24. The top down view and measurements of the sFPI aluminum base

APPENDIX B
ARDUINO PROGRAMS

A. Writing to the MCP4725 12-bit DAC via I2C

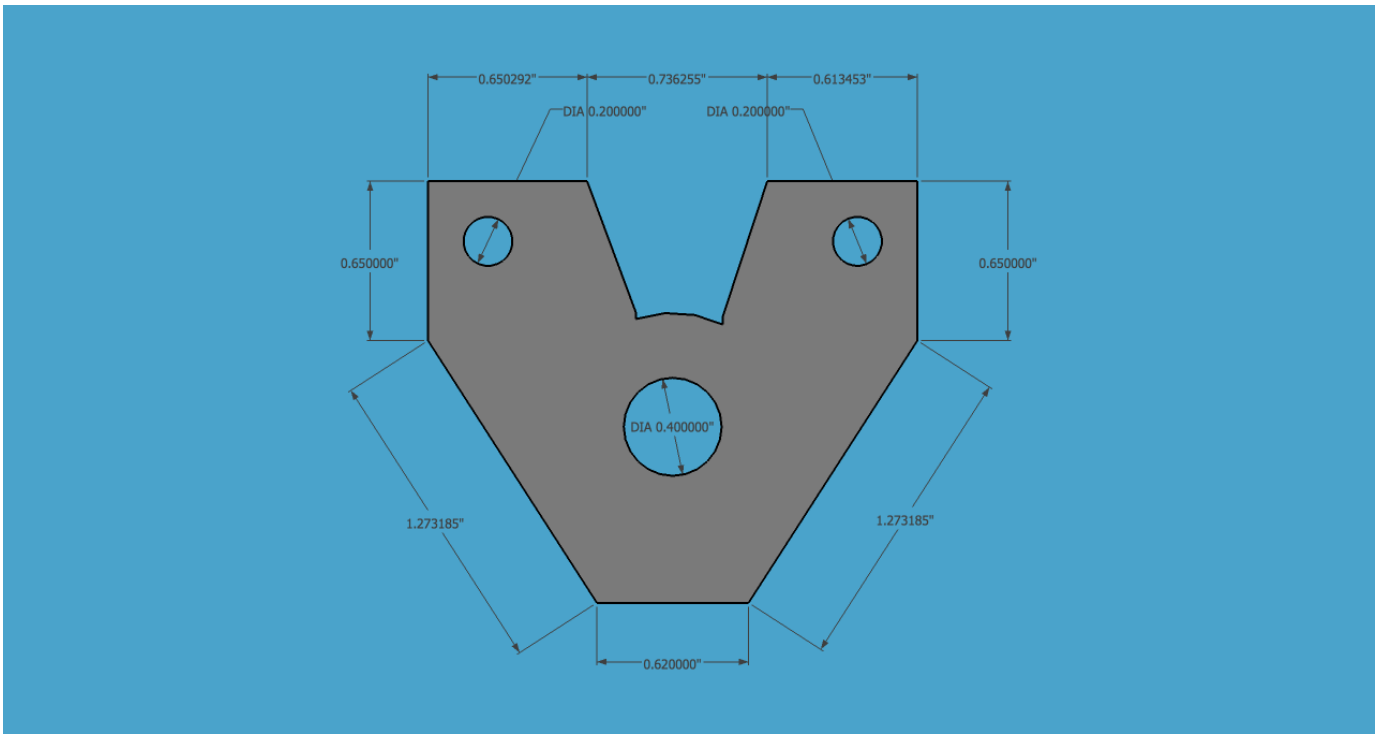


Fig. 25. The bottom up view and measurements of the sFPI aluminum base

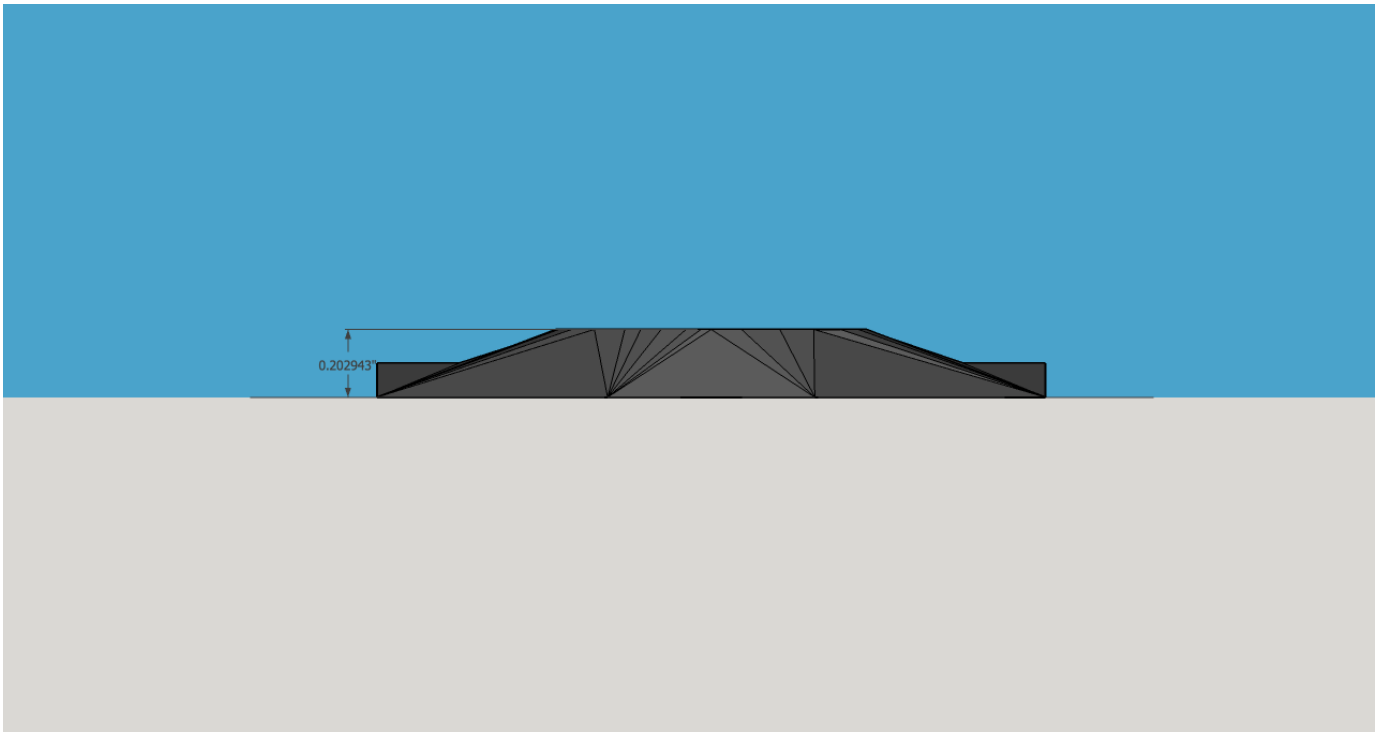


Fig. 26. The front cross section view and measurements of the sFPI aluminum base

```
/*  sFPI Driver version 1.5
    Copyright (c) 2015 David Miles Houston

    This sketch uses the following associated
```

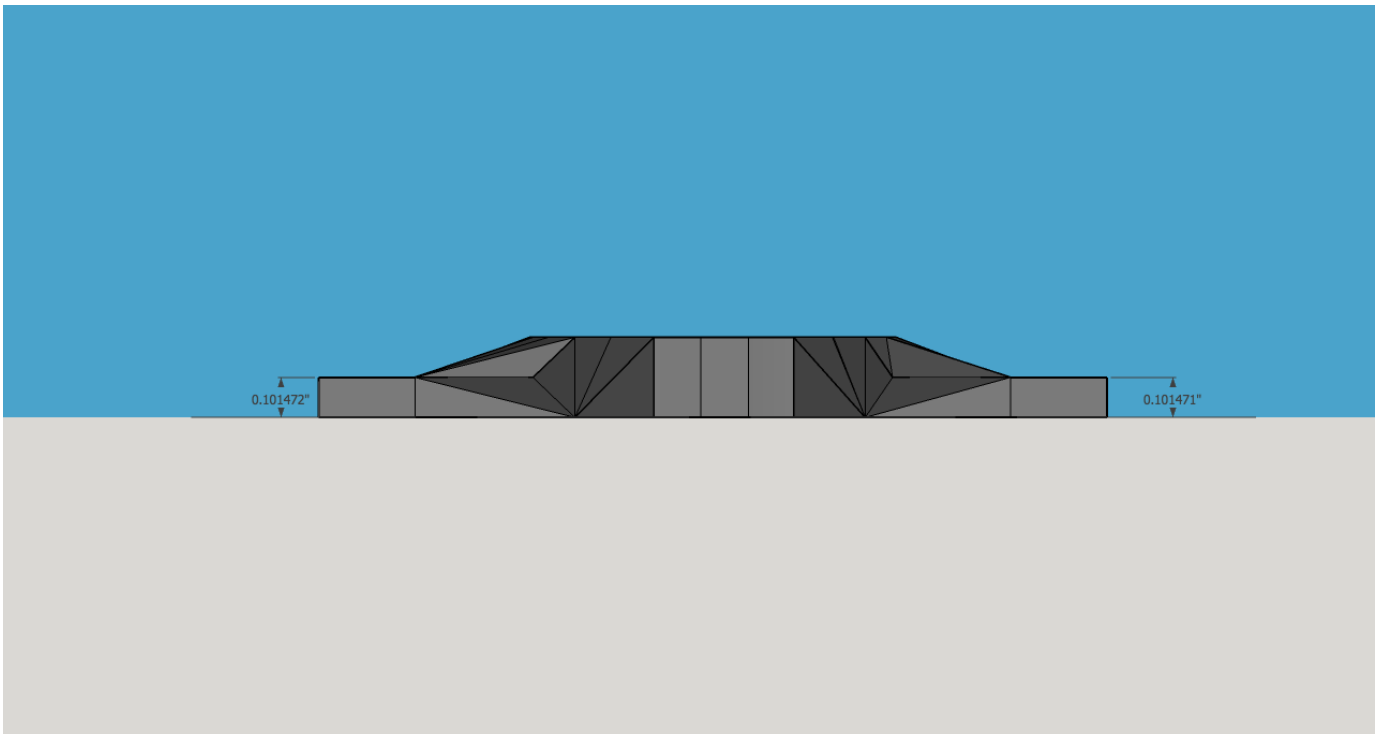


Fig. 27. The back cross section view and measurements of the sFPI aluminum base

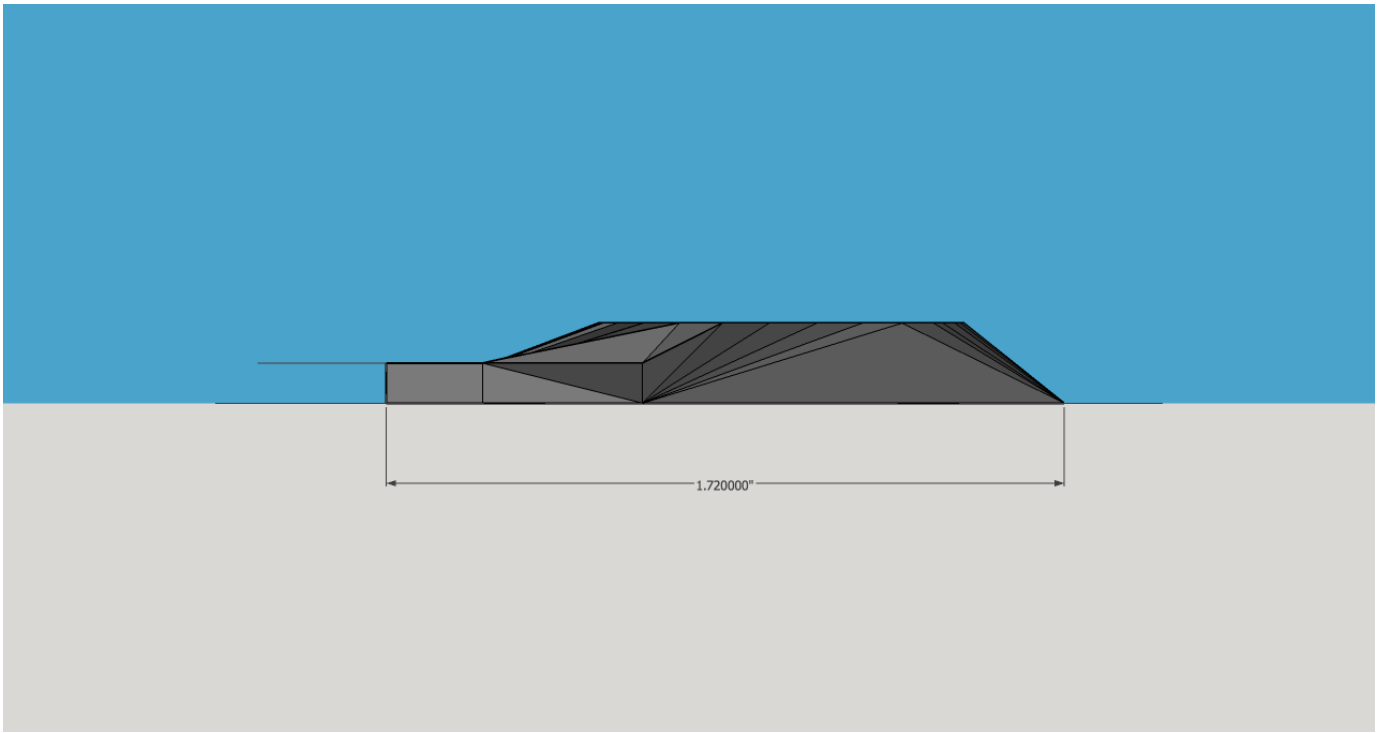


Fig. 28. The back cross section view and measurements of the sFPI aluminum base

```
pins in conjunction with the ,
for the use of a sawtooth signal generator.

Developers Note: Current Pin out is as follows
```

```
    Pin Name        Pin Location
    ---------     --------------
      SDA               A5
      SCL               A4
*/

#include <Wire.h>
#include "Adafruit_MCP4725.h"

// use layout from the Adafruit Header File
Adafruit_MCP4725 dac;

#define bitres 1024;
// Number of tunable bits at one time
const int numbits = 1024;
// Number of bits in the period of the signal
const int wave_bits = 4096;
const int DP1 = 9;
//Number of bits to skip each time to increase speed
uint32_t skip = 1;

// Setup Loop
void setup()
{
  // Setup for Serial Communication
  // Pin Assignments
  pinMode(DP1,OUTPUT);
  dac.begin(0x62);
}

// Infinite Loop
void loop()
{
        digitalWrite(DP1,HIGH);
    for (uint32_t i = 0; i < wave_bits-1; i = i + skip) {
        dac.setVoltage(i,false);
                if (i==0) {
                  digitalWrite(DP1,LOW);
                }
    }

}
```

B. *Program for capturing and sending serial data to the computer*

```
/*
 * Title: Data Collection for sFPI
 * Authors: Scott Prahl & David Houston
 * Version: 1.0
 * Date: November 2015
 *
 * Description:  This program accepts a byte of data
 * from the computer and then determines what ADC
 * pin to sample data from. It then sends the 512 bytes
 * to the computer via USB.
 */

// Arduino Pin A0 connected to photodiode output
const int PIN = A0;
// Arduino Pin A1 connected to ramp function output
const int SIN = A1;
// Pin connected to the other Arduino for communication
const int DP1 = 9;
```

```
const int PULSE = 10;
boolean run = false;
unsigned int data[257];

void setup()
{
    Serial.begin(57600);
    pinMode(PIN, INPUT);
    pinMode(SIN, INPUT);
    pinMode(DP1, INPUT);
}

void loop()
{
    // If data is requested by computer
    if (Serial.available() > 0) {
        char incomingByte = Serial.read();
        Serial.flush();

        while (run != true) {
          if (digitalRead(DP1) == HIGH){
            run = true;
          }
        }

        if (incomingByte == '0') {
          TakeData(SIN);
        }
        else
        {
          TakeData(PIN);
        }
    }
}

void TakeData(const int P)
{
    for (int i=0; i<256; i++) {
        data[i] = analogRead(P);
      }

    // send 512 bytes of data over serial interface
    // this is probably slowest of everything
    for (int i=0; i<256; i++) {
        Serial.write(byte(0x00FF & data[i]));
        Serial.write(byte(data[i] >> 8));
    }
}
```

APPENDIX C
PYTHON GUI

A. *Main program*

```
from Tkinter import *
import sys, os
from Application import GUI

# The backbone Application Class

def main():
```

```
    app = GUI(master=root)
    mods = app.master

    # Modify GUI features
    mods.title("GUI - Scanning Fabry-Perot Interferometer")
    app.mainloop()
    # send to GUI Application and display accordingly

if __name__ == "__main__":
    root = Tk()
    main()
    sys.exit()
```

B. *Application Class*

```
import csv, os
import numpy as np
import datetime
import FileDialog
import matplotlib
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg \
    import FigureCanvasTkAgg, NavigationToolbar2TkAgg
from matplotlib.figure import Figure
from USBInterface import SerialPort as SP
from Tkinter import *
# from GUIErrors import *

class GUI(Frame):
    def __init__(self,directory="/Applications/sFPI-GUI",
                 master=None,port=None,data=None,go=None):
        Frame.__init__(self,master,bd=1)
        # Initialize Class Variables and Functions
        self.cwd = directory
        self.master = master
        self.calibrated = False
        self.max_screenwidth = 1200
        self.max_screenheight = 690
        self.start_time = None

        # Set the geometry of the main windowing
        self.set_geometry()

        # Data Variables
        self.driver = [1]
        self.pdiode = [1]
        self.preferences_pause = False
        filename = self.cwd + "/docs/info.bin"
```

```
        if not os.path.exists(os.path.dirname(filename)):
            self.port_name = "New Port Address"
        else:
            self.port_name = open(filename,'r').read()
        # Set Port variable
        self.port = port
        # Execute Initialization Functions for Operation
        self.pack()
        self.clock()
        self.tell_time()
        self.create_menu_bar()
        self.preferences()
        self.display_port()
        self.update()

    def create_menu_bar(self):
        menu = Menu(self.master)
        self.master.config(menu=menu)

        # File Menu
        fileMenu = Menu(menu)
        menu.add_cascade(label="File",menu=fileMenu)
        fileMenu.add_command(label="New Project...",
                             command=self.save_project())
        fileMenu.add_command(label="Save Project",
                             command=self.save_project)
        fileMenu.add_separator()
        fileMenu.add_command(label="Exit",
                             command=self.master.destroy)

        # Edit Menu
        editMenu = Menu(menu)
        menu.add_cascade(label="Edit",menu=editMenu)
        editMenu.add_command(label="Copy Image",
                             command=self.save_project)

        # Tools Menu
        toolsMenu = Menu(menu)
        menu.add_cascade(label="Tools",menu=toolsMenu)
        toolsMenu.add_command(label="Change Serial Port",
                              command=self.serial_window)
        toolsMenu.add_command(label="Calibrate",
                              command=self.fsr_calibration)
        toolsMenu.add_command(label="Preferences",
                              command=self.preferences)
        toolsMenu.add_separator()
        toolsMenu.add_command(label="Pause", command=self.pause)
        toolsMenu.add_command(label="Start", command=self.resume)
        toolsMenu.add_command(label="Get Ramp",
                              command=self.get_ramp)
```

```
def graph_data(self):
    if bool(self.preferences_twoplot_var.get()) is False:
        f = Figure(figsize=(15,6), dpi=100,)
        self.graph = f.add_subplot(111)
        self.plot_data, = self.graph.plot(0,0)

        self.graph_canvas = FigureCanvasTkAgg(f, self)
        self.graph_canvas.show()
        self.graph_canvas.get_tk_widget().pack(side=BOTTOM,
                                               fill=BOTH, expand=True)

        toolbar = NavigationToolbar2TkAgg(self.graph_canvas, self)
        toolbar.update()
        self.graph_canvas._tkcanvas.pack(side=TOP,
                                         fill=BOTH, expand=True)
    else:
        f = Figure(figsize=(15,6), dpi=100,)
        self.graph1 = f.add_subplot(211)
        self.graph2 = f.add_subplot(212)
        self.plot_data_1, = self.graph1.plot(0,0)
        self.plot_data_2, = self.graph2.plot(0,0)

        self.graph_canvas = FigureCanvasTkAgg(f, self)
        self.graph_canvas.show()
        self.graph_canvas.get_tk_widget().pack(side=BOTTOM,
                                               fill=BOTH, expand=True)

        toolbar = NavigationToolbar2TkAgg(self.graph_canvas, self)
        toolbar.update()
        self.graph_canvas._tkcanvas.pack(side=TOP,
                                         fill=BOTH, expand=True)
    self.update_graph()

def update_graph(self):
    if bool(self.preferences_twoplot_var.get()) is False:
        self.graph.clear()
        self.graph.plot(np.array(self.driver),
                        np.array(self.pdiode),"r")
        self.graph.set_title('Spectral Plot of Cavity')
        self.graph.set_xlabel('Frequency (GHz)')
        self.graph.set_ylabel('Normalized Intensity')
        self.graph_canvas.draw()

    else:
        self.graph1.clear()
        self.graph1.plot(np.array(self.driver),"r")
        self.graph1.set_title('Ramp Function')

        self.graph2.clear()
```

```
            self.graph2.plot(np.array(self.pdiode),"r")
            self.graph2.set_title('Photodiode Output')
            self.graph_canvas.draw()

        if self.preferences_pause is False:
            self.after(500,self.update_graph)

    def save_project(self):
        filename = self.cwd + '/Logs/' + \
                   datetime.datetime.now().strftime("%m-%d-%Y-%H_%M_%S") \
                   + '.csv'
        if not os.path.exists(os.path.dirname(filename)):
            os.makedirs(os.path.dirname(filename))
        with open(filename, 'wb') as csvfile:
            spamwriter = csv.writer(csvfile, delimiter=',',
                            quotechar='|',
                                    quoting=csv.QUOTE_MINIMAL)
            spamwriter.writerow(self.driver)
            spamwriter.writerow(self.pdiode)
        csvfile.close()

    def pause(self):
        self.preferences_pause = True

    def resume(self):
        self.preferences_pause = False
        self.update_graph()
        self.serial = SP(port=self.port_name)
        self.serial_read()

    def get_ramp(self):
        self.serial = SP(port=self.port_name,get_ramp=True)
        self.serial_read(run_once=True)

    def set_geometry(self):
        if self.master.winfo_screenwidth() > self.max_screenwidth:
            self.screen_width = self.max_screenwidth
        else:
            self.screen_width = self.master.winfo_screenwidth()

        if self.master.winfo_screenheight() > self.max_screenheight:
            self.screen_height = self.max_screenheight
        else:
            self.screen_height = self.master.winfo_screenwidth()

        self.master.geometry('{}x{}'.format(self.screen_width,
                                            self.screen_height))
        self.master.minsize(self.screen_width,self.screen_height)
        self.master.maxsize(self.screen_width,self.screen_height)
```

```
def fsr_calibration(self):
    self.cal_window = Toplevel(self.master)
    self.cal_window.title("FSR (GHz)")
    iframe1 = Frame(self.cal_window,bd=2)

    # Get Port Button
    self.butt = Button(iframe1, text="Calibrate",fg="black",
                       command=self.begin_calibration)
    self.butt.pack(side=RIGHT, padx=5)

    self.fsr = Entry(iframe1)
    self.fsr.pack(side=RIGHT, padx=0)

    self.fsr.delete(0, END)
    self.fsr.insert(0, "FSR (GHz)")

    iframe1.pack(expand=1, fill=X, pady=10, padx=5)

def begin_calibration(self):
    self.calibrated = True
    self.cal_window.destroy()

def set_calibration(self):
    fsr = self.fsr.get()
    fsr_range = float(fsr)
    data_range = max(self.driver)-min(self.driver)
    scale = float(fsr_range/data_range)
    offset = self.driver[0]
    normalize = max(self.pdiode)
    for k in range(len(self.driver)):
        self.driver[k] = self.driver[k] - offset
        self.driver[k] = scale*self.driver[k]
        self.pdiode[k] = self.pdiode[k]/normalize
    self.graph.set_ylim((0,1),auto=False)
    self.graph.set_ymargin(0)

def preferences(self):
    self.preferences = Toplevel(self.master)
    iframe = Frame(self.preferences,bd=3)

    self.preferences_test_var = IntVar()
    self.Testing = Checkbutton(iframe, text="Testing",
                               variable=self.preferences_test_var,
                               onvalue=True, offvalue=False)
    self.Testing.pack()

    self.preferences_twoplot_var = IntVar()
    self.Plot2 = Checkbutton(iframe, text="Plot Seperate Graphs",
                             variable=self.preferences_twoplot_var,
                             onvalue=True, offvalue=False)
```

```
        self.Plot2.pack()

        self.Submit = self.GET = Button(iframe, text="Submit",fg="black",
                                        command=self.close_window,
                                        relief=SUNKEN)
        self.Submit.pack(fill=X)

        iframe.pack(expand=1, fill=X, pady=10, padx=5)
        self.preferences.attributes("-topmost", True)
        self.preferences.title("Preferences")
        self.preferences.geometry('{}x{}'.format(170, 90))
        self.preferences.minsize(170,90)

        self.preferences.update_idletasks()
        w = self.preferences.winfo_screenwidth()
        h = self.preferences.winfo_screenheight()
        size = tuple(int(_) for _ in
                     self.preferences.geometry().split('+')[0].split('x'))
        x = w/2 - size[0]/2
        y = h/2 - size[1]/2
        self.preferences.geometry("%dx%d+%d+%d" % (size + (x, y)))

    def serial_window(self):
        self.serial_window = Toplevel(self.master)
        iframe1 = Frame(self.serial_window,bd=2)
        self.serial_window.title("Enter Serial Port")

        # Get Port Button
        self.GET = Button(iframe1, text="Store Port",fg="black",
                                                command=self.get_port)
        self.GET.pack(side=RIGHT, padx=5)

        self.e = Entry(iframe1)
        self.e.pack(side=RIGHT, padx=0)

        self.e.delete(0, END)
        self.e.insert(0, self.port_name)

        iframe1.pack(expand=1, fill=X, pady=10, padx=5)

    def get_port(self):
        self.port = self.e.get()
        filename = self.cwd + "/docs/info.bin"
        if not os.path.exists(os.path.dirname(filename)):
            os.makedirs(os.path.dirname(filename))
        info = open(filename,'w')
        string = self.port
        info.write(string)
        info.close()
        self.port_name = string
```

```
        self.serial_window.destroy()
        self.update_port()

    def display_port(self):
        try:
            filename = self.cwd + "/docs/info.bin"
            if not os.path.exists(os.path.dirname(filename)):
                os.makedirs(os.path.dirname(filename))
            info = open(filename,'w')
            self.port_name = info.read()
            self.name_label = Label(self.master, text=self.port_name,
                                    bd=1, relief=SUNKEN, anchor=SE)
            self.name_label.pack(fill=X)

        except IndexError as ie:
            port = "No Port Selected"
            self.name_label = Label(self.master, text=port,
                                    bd=1, relief=SUNKEN, anchor=SE)
            self.name_label.pack(fill=X)

        except IOError as INOUT:
            port = "No Port Selected"
            self.name_label = Label(self.master, text=port,
                                    bd=1, relief=SUNKEN, anchor=SE)
            self.name_label.pack(fill=X)

    def update_port(self):
        self.name_label.configure(text=self.port_name)
        self.serial = SP(port=self.port_name)

    def close_window(self):
        self.preferences.destroy()
        if bool(self.preferences_test_var.get()) is True:
            data = {}
            filename = self.cwd + "/test/photodiode_output.csv"
            if not os.path.exists(os.path.dirname(filename)):
                os.makedirs(os.path.dirname(filename))
            with open(filename,'r') as csvdata:
                data_reader = csv.reader(csvdata)
                i = 0
                for row in data_reader:
                    if i == 0:
                        data['driver'] = row
                        i = i + 1
                    else:
                        data['pdiode'] = row
                for num in range(len(data['driver'])):
                    data['driver'][num] = float(data['driver'][num])
                    data['pdiode'][num] = float(data['pdiode'][num])
                self.driver = data['driver']
```

```
                self.pdiode = data['pdiode']
        self.graph_data()

    def clock(self):
        self.time = Label(self.master, bd=1, relief=SUNKEN, anchor=SE)
        self.time.pack(fill=X)

    def tell_time(self):
        time = datetime.datetime.now().strftime("Time: %H:%M:%S")
        self.time.config(text=time)
        #lab['text'] = time
        self.after(1000, self.tell_time) # run itself again after 1000 ms

    def serial_read(self,run_once=False):
        data = self.serial.Get_Data()
        if data is not False:
            if data['get_ramp'] is False:
                self.pdiode = data['values']
            else:
                self.driver = data['values']

            if self.calibrated == True:
                self.set_calibration()

            if run_once is False:
                self.after(1000,self.serial_read)
        else:
            self.name_label.configure(text="Communication Failed")
```

C. *USB Interface Class*

```
import serial, time, itertools
from GUIErrors import *

class SerialPort():
    def __init__(self,port=None,get_ramp=False):
        try:
            self.get_ramp = get_ramp
            self.error = False
            self.ser = serial.Serial(port, 57600,timeout=1)
        except OSError as Bad_Port:
            self.error = True

    def Get_Data(self):
        try:
            arduino_data = []
            data = []
            if self.error is not True:
```

```
                # VERY IMPORTANT!!!
                time.sleep(1/1000)
                self.ser.setDTR(level=0)
                time.sleep(1/1000)
                if self.get_ramp == True:
                    self.ser.write("0")
                else:
                    self.ser.write("1")
                self.bits = 256
                for i in range(0,(self.bits*2)):
                    arduino_data.append(self.ser.read())
                if arduino_data[0] == '':
                            raise DataError(arduino_data[0])
                arduino_data = self.deArduinoify(arduino_data)
                self.ser.flush()
                for i in range(0,self.bits-1):
                    data.append(self.add_hex(arduino_data[2*i],
                                             arduino_data[2*i+1]))
            else:
                data = []
                raise DataError('No Data')

            output = {'values':data,'get_ramp':self.get_ramp}
            return output
        except DataError as Bad_comm:
            return False

    def deArduinoify(self,data):
        values = []
        for k in range(len(data)):
            d = data[k].encode('hex')
            dnew = int(d,16)
            values.append(dnew)
        return values

    def add_hex(self,A,B):
        conv = B*256 + A
        return conv
```

D. GUI Errors Class

```
class Error(Exception):
    pass

class DataError(Error):
    def __init__(self,data):
        self.data = data
    def __str__(self):
```

```
        return repr(self.data)

class ImageError(Error):
    def __init__(self,data):
        self.data = data
    def __str__(self):
        return repr(self.data)
```

E. Setup Function

```
"""
This is a setup.py script generated by py2applet

Usage:
    python setup.py py2app
"""

from setuptools import setup

APP = ['GUI.py']
DATA_FILES = []
OPTIONS = {
    'iconfile':'GUI.icns',
    'plist': {'CFBundleShortVersionString':'1.0.0',}
}

setup(
    app=APP,
    name='GUI',
    data_files=DATA_FILES,
    options={'py2app': OPTIONS},
    setup_requires=['py2app'],
)
```

APPENDIX D
LTSPICE NETLISTS

A. Interferometer Driver LTSpice Netlist for Simulation

```
* Z:\Users\aviatorblue\Documents\sFPI\Electrical Design\Amplifier_Design\amplifier_setup_rev2.asc
* The LT1001 component is used in place of the NTE941M
XU1 0 N011 Vdd Vss N012 LT1001
XU2 0 N006 Vdd Vss N007 LT1001
XU3 0 N009 Vdd Vss N010 LT1001
XU4 0 N008 Vdd Vss Vac LT1001
R1 N011 N013 10k
R2 N012 N011 10k
R3 N006 N012 5k
R4 N007 N006 10k
R5 N009 N007 5k
R6 N010 N009 10k
R7 N008 N010 10k
R8 Vac N008 10k
* The sawtooth_current.txt file is generated using a python script located in the same directory
V1 N013 0 PWL file=sawtooth_current.txt
V2 Vdd 0 17
V3 0 Vss 17
* The LT1086 component is used in place of the NTEXXX (5 V regulator)
XU5 0 5V Vdd LT1085-5
R_Arduino 5V 0 10Meg
* The LT1086 component is used in place of the NTE956
XU6 N003 Vdc Vdd LT1086
R10 N003 Vss 2.6k
R11 Vdc N003 1.5k
XU9 0 N002 Vdd Vss Vout LT1001
XU10 Vac N004 Vdd Vss N004 LT1001
XU11 Vdc N001 Vdd Vss N001 LT1001
R19 N002 N001 100k
R20 N002 N004 100k
R12 Vout N002 100k
R13 Vout N005 300
C1 N005 0 15.51n
.tran 0 0.01 0
.lib LT1083.lib
.lib LTC.lib
.backanno
.end
```

APPENDIX E
SCHEMATICS

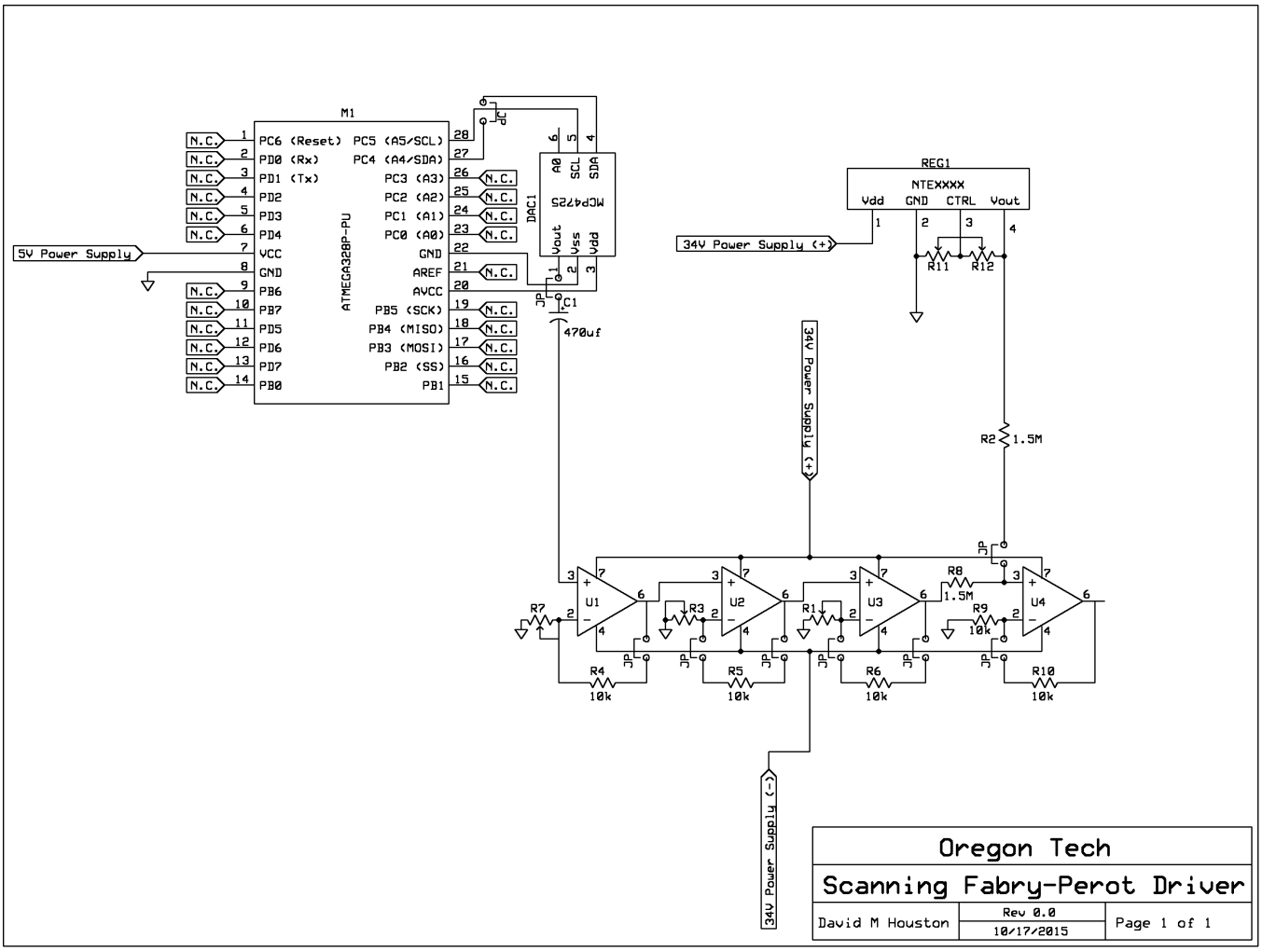


Fig. 29. Driver Schematic

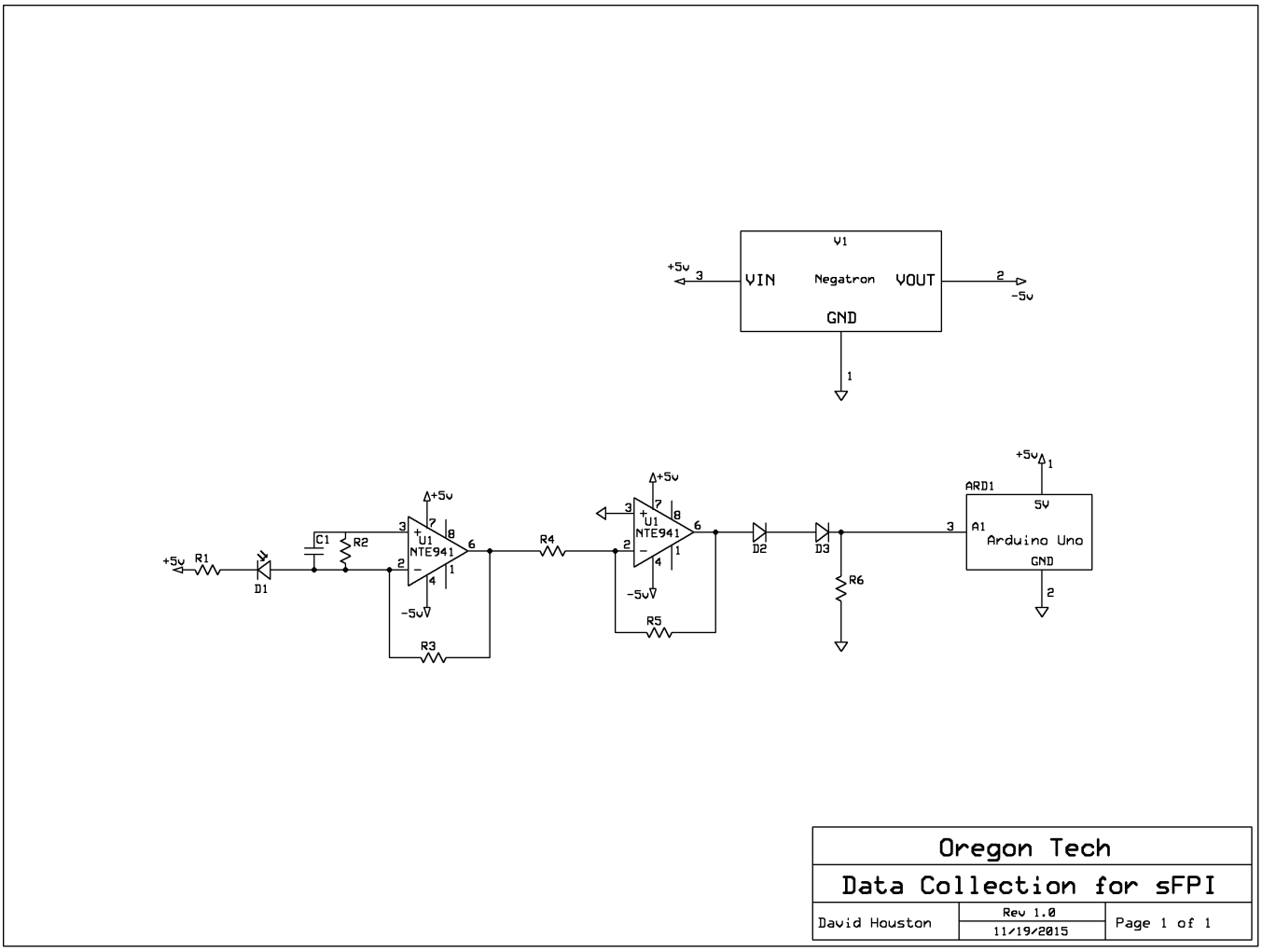


Fig. 30. Reverse Biased Transimpedance Amplifier of the Photodiode output

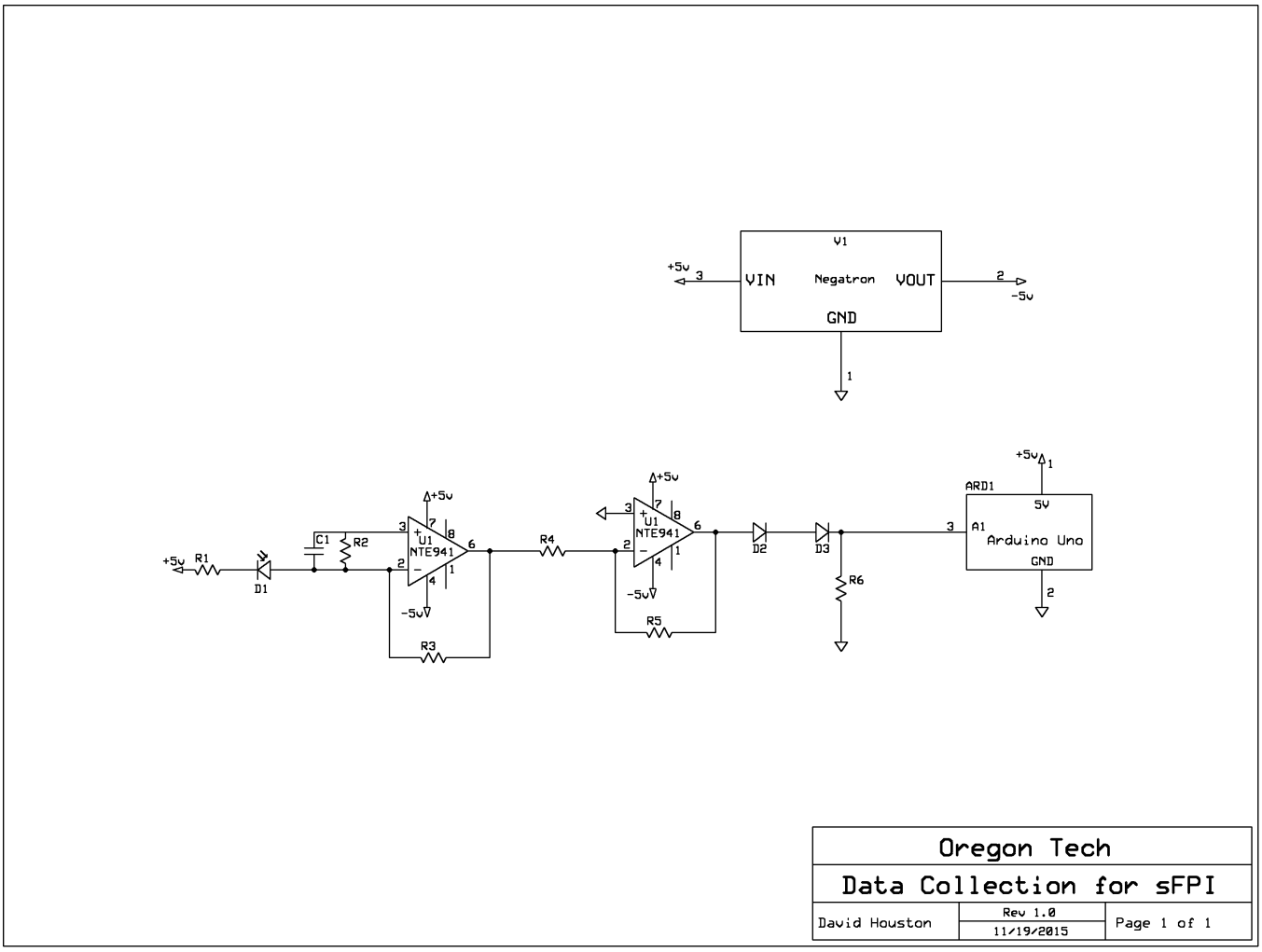


Fig. 31. Power Distribution Schematic

APPENDIX F
LTSPICE MODELS

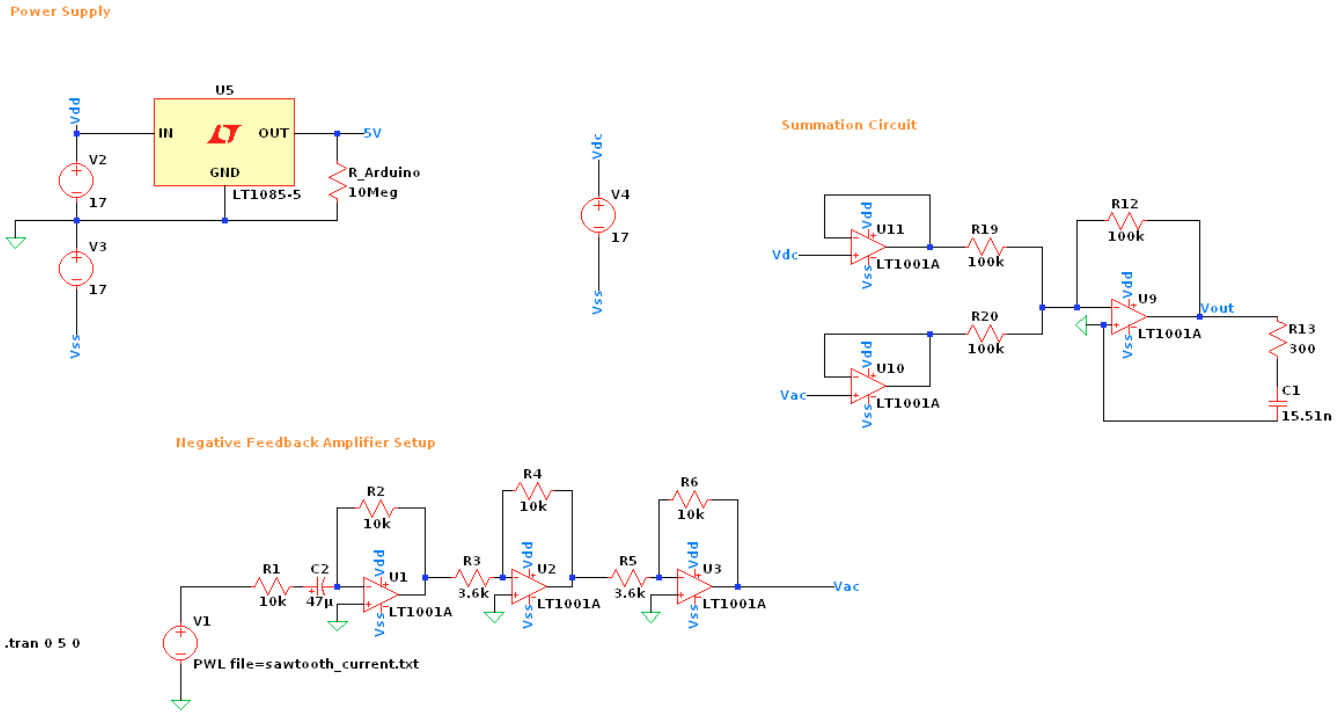


Fig. 32. LTSpice Model for sFPI electrical driver design

APPENDIX G
DATASHEETS

ELECTRONICS, INC.
44 FARRAND STREET
BLOOMFIELD, NJ 07003
(973) 748–5089
http://www.nteinc.com

NTE941M & NTE941SM
Integrated Circuit
Operational Amplifier

Description:

The NTE941M and NTE941SM (Surface Mount) are general purpose operational amplifiers in 8–Lead DIP type packages and offer many features which make their application nearly foolproof: overload protection on the input and output, no latch–up when the common mode range is exceeded, as well as freedom from oscillators.

Absolute Maximum Ratings:

Supply Voltage, V_S . . . . . ±18V
Differential Input Voltage, V_{ID} . . . . . ±30V
Common Mode Input Voltage (Note 2), V_{ICM} . . . . . ±15V
Power Dissipation (Note 1), P_D 500mW
Output Short–Circuit Duration, t_S . . . . . Continuous
Operating Temperature Range, T_{opr} . . . . . 0° to +70°C
Storage Temperature Range, T_{stg} . . . . . –65° to +150°C
Junction Temperature, T_J . . . . . +100°C
Lead Temperature (During Soldering, 10sec), T_L . . . . . +260°C
Thermal Resistance, Junction–to–Ambient, R_{thJA}
NTE941M . . . . . +100°C/W
NTE941SM . . . . . +195°C/W

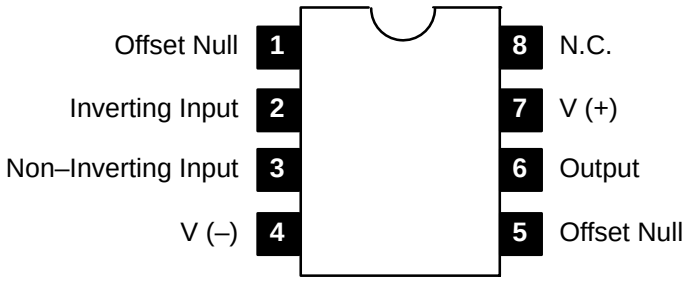
Note 1. For operation at elevated temperatures, these devices must be derated based on thermal resistance, and T_J Max ($T_J = T_A + (R_{thJA} P_D)$).

Note 2. For supply voltage less than ±15V, the absolute maximum input voltage is equal to the supply voltage.

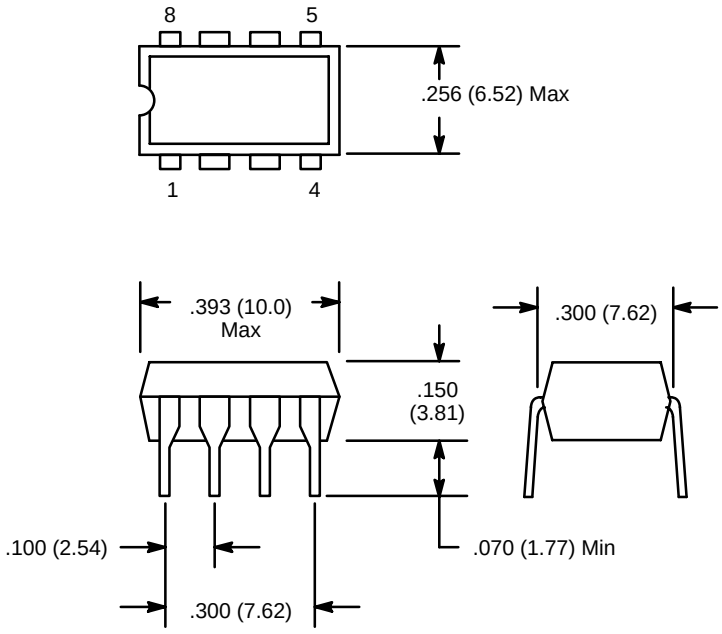
Electrical Characteristics: (V_S = ±15V, 0° ≤ T_A ≤ +70°C unless otherwise specified)

Parameter	Symbol	Test Conditions		Min	Typ	Max	Unit
Input Offset Voltage	V_{IO}	R_S ≤ 10kΩ	T_A = +25°C	–	2.0	6.0	mV
				–	–	7.5	mV
Input Offset Voltage Adjustment Range	V_{IOR}	V_S = ±20V, T_A = +25°C		–	±15	–	V
Input Offset Current	I_{IO}	T_A = +25°C		–	20	200	nA
				–	–	300	nA
Input Bias Current	I_{IB}	T_A = +25°C		–	80	500	nA
				–	–	0.8	µA
Input Resistance	r_i	V_S = ±20V, T_A = +25°C		0.3	2.0	–	MΩ
Common Mode Input Voltage Range	V_{ICR}	T_A = +25°C		–	±12	±13	V
Large Signal Voltage Gain	A_V	V_O = ±10V, R_L ≥ 2kΩ	T_A = +25°C	20	200	–	V/mV
				15	–	–	V/mV
Output Voltage Swing	V_O	R_L ≥ 10kΩ		±12	±14	–	V
		R_L ≥ 2kΩ		±10	±13	–	V
Output Short–Circuit Current	I_{OS}	T_A = +25°C		–	25	–	mA
Common–Mode Rejection Ratio	CMRR	R_S ≤ 10kΩ, V_{CM} = ±12V		70	90	–	dB
Supply Voltage Rejection Ratio	PSRR	V_S = ±20V to ±5V, R_S ≤ 10kΩ		77	96	–	dB
Transient Response Rise Time	t_{TLH}	T_A = +25°C, Unity Gain		–	0.3	–	µs
Transient Response Overshoot	os			–	5	–	%
Transient Response Slew Rate	SR			–	0.5	–	V/µs
Supplu Current	I_D	T_A = +25°C		–	1.7	2.8	mA
Power Consumption	P_C	T_A = +25°C		–	50	85	mW

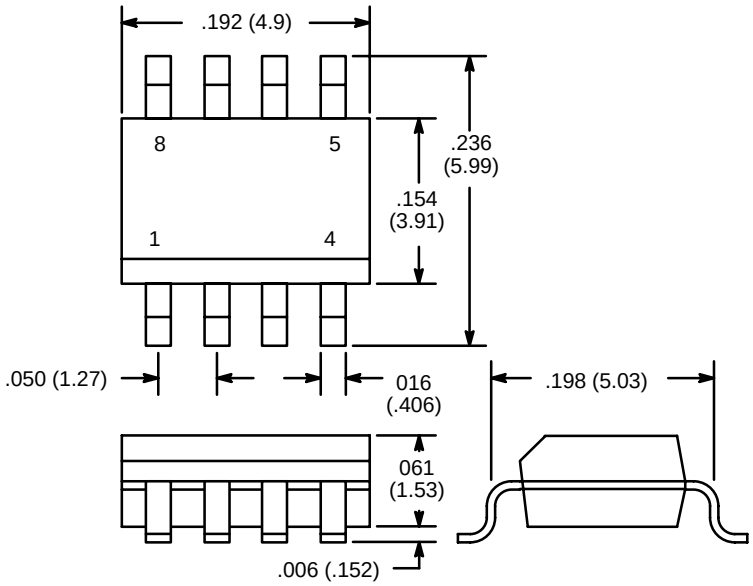
Pin Connection Diagram



NTE941M



NTE941SM



NOTE: Pin1 on Beveled Edge

ELECTRONICS, INC.
44 FARRAND STREET
BLOOMFIELD, NJ 07003
(973) 748–5089

NTE957
Integrated Circuit
3–Terminal Adjustable Negative Voltage Regulator

Description:

The NTE957 is an adjustable 3–terminal negative voltage regulator in a TO220 type package capable of supplying in excess of –1.5A over a –1.2V to –37V output range. The circuit design has been optimized for excellent regulation and low thermal transients. Further, the NTE957 features internal current limiting, thermal shutdown, and safe–area compensation, making this device virtually blowout–proof against overloads.

The NTE957 serves a wide variety of applications including local on–card regulation, programmable–output voltage regulation or precision current regulation. The NTE957 is the ideal complement to the NTE956 adjustable positive regulator.

Features:

- Output Voltage Adjustable from –1.2V to –37V
- Guaranteed 1.5A Output Current
- Line Regulation Typically 0.01%/V
- Load Regulation Typically 0.3%
- Excellent Thermal Regulation: 0.002%/W
- 77dB Ripple Rejection
- Temperature–Independent Current Limit
- Internal Thermal Overload Protection
- 100% Electrical Burn–In
- Eliminates the Need to Stock Many Voltages

Absolute Maximum Ratings:

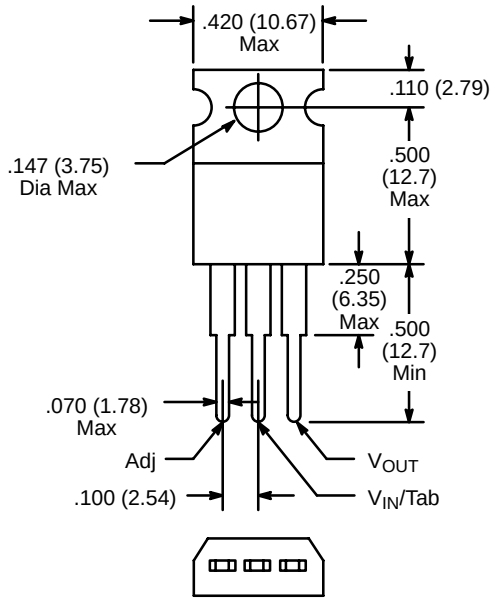
Power Dissipation, P_D . . . . . Internally Limited
Input–Output Voltage Differential, $V_I–V_O$ 40V
Operating Junction Temperature Range, T_J . . . . . 0° to +125°C
Storage Temperature Range, T_{stg} . . . . . –65° to +150°C
Typical Thermal Resistance, Junction–to–Case, R_{thJC} . . . . . 4°C/W
Lead Temperature (During Soldering, 10sec), T_L . . . . . +300°C

Electrical Characteristics: ($0° \le T_J \le +125°C$, $V_{IN}–V_{OUT} = 5V$, $I_O = 500mA$, $I_{MAX} = 1.5A$, Note 1 unless otherwise specified)

Parameter	Symbol	Test Conditions		Min	Typ	Max	Unit
Line Regulation	Reg_{line}	$T_A = +25°C$, $3V \le (V_{IN}–V_{OUT}) \le 40V$, Note 2		–	0.01	0.04	%/V
		$3V \le (V_{IN}–V_{OUT}) \le 40V$		–	0.02	0.07	%/V
Load Regulation	Reg_{load}	$T_A = +25°C$, $10mA \le I_O \le I_{MAX}$, Note 2	$V_{OUT} \le 5V$	–	15	50	mV
			$V_{OUT} \ge 5V$	–	0.3	1.0	%
		$10mA \le I_O \le 1_{MAX}$, Note 2	$V_{OUT} \le 5V$	–	20	70	mV
			$V_{OUT} \ge 5V$	–	0.3	1.5	%
Thermal Regulation		$T_A = +25°C$, 20ms Pulse		–	0.003	0.04	%/W
Adjustment Pin Current	I_{Adj}			–	65	100	µA
Adjustment Pin Current Change	ΔI_{Adj}	$10mA \le I_L \le I_{MAX}$, $2.5V \le (V_{IN}–V_{OUT}) \le 40V$, $T_A = +25°C$		–	2	5	µA
Reference Voltage	V_{ref}	$T_A = +25°C$		–1.213	–1.250	–1.287	V
		$3V \le (V_{IN}–V_{OUT}) \le 40V$, $10mA \le I_O \le 1_{MAX}$, $P \le P_{MAX}$		–1.200	–1.250	–1.300	V
Temperature Stability	T_S	$0° \le T_J \le +125°C$		–	0.6	–	%
Minimum Load Current	I_{Lmin}	$(V_{IN}–V_{OUT}) \le 40V$		–	2.5	10	mA
		$(V_{IN}–V_{OUT}) \le 10V$		–	1.5	6.0	mA
Maximum Output Current Limit	I_{max}	$V_{IN}–V_{OUT} \le 15V$		1.5	2.2	–	A
		$V_{IN}–V_{OUT} = 40V$		–	0.4	–	A
RMS Output Noise, % of V_{OUT}	N	$T_A = +25°C$, $10Hz \le f \le 10kHz$		–	0.003	–	%
Ripple Rejection Ratio	RR	$V_{OUT} = 10V$, $f = 120Hz$		–	60	–	dB
			$C_{ADJ} = 10µF$	66	77	–	dB
Long Term Stability	S	$T_A = +125°C$, 1000 Hours		–	0.3	1.0	%

Note 1. Although power dissipation is internally limited, these specifications are applicable for power dissipations of 20W.

Note 2. Regulation is measured at constant junction temperature, using pulse testing with a low duty cycle. Changes in output voltage due to heating effects are covered under the specification for thermal regulation.



MCP4725

12-Bit Digital-to-Analog Converter with EEPROM Memory in SOT-23-6

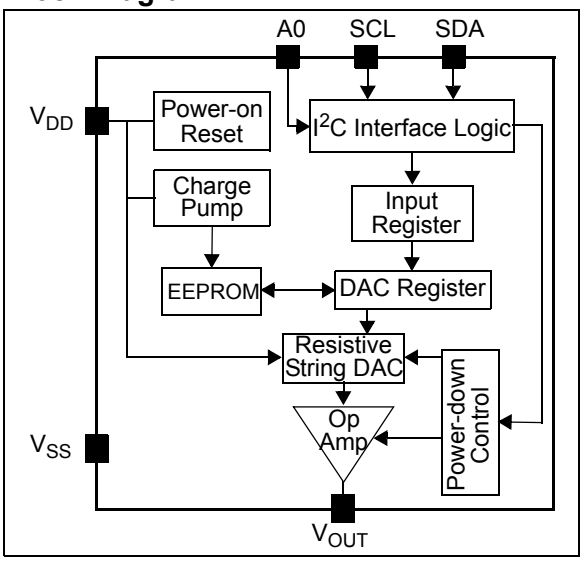
Features

- 12-Bit Resolution
- On-Board Non-Volatile Memory (EEPROM)
- ±0.2 LSB DNL (typical)
- External A0 Address Pin
- Normal or Power-Down Mode
- Fast Settling Time: 6 µs (typical)
- External Voltage Reference (V_{DD})
- Rail-to-Rail Output
- Low Power Consumption
- Single-Supply Operation: 2.7V to 5.5V
- I^2C^{TM} Interface:
  - Eight Available Addresses
  - Standard (100 kbps), Fast (400 kbps), and High-Speed (3.4 Mbps) Modes
- Small 6-lead SOT-23 Package
- Extended Temperature Range: -40°C to +125°C

Applications

- Set Point or Offset Trimming
- Sensor Calibration
- Closed-Loop Servo Control
- Low Power Portable Instrumentation
- PC Peripherals
- Data Acquisition Systems

Block Diagram



DESCRIPTION

The MCP4725 is a low-power, high accuracy, single channel, 12-bit buffered voltage output Digital-to-Analog Convertor (DAC) with non-volatile memory (EEPROM). Its on-board precision output amplifier allows it to achieve rail-to-rail analog output swing.

The DAC input and configuration data can be programmed to the non-volatile memory (EEPROM) by the user using I^2C interface command. The non-volatile memory feature enables the DAC device to hold the DAC input code during power-off time, and the DAC output is available immediately after power-up. This feature is very useful when the DAC device is used as a supporting device for other devices in the network.

The device includes a Power-On-Reset (POR) circuit to ensure reliable power-up and an on-board charge pump for the EEPROM programming voltage. The DAC reference is driven from V_{DD} directly. In power-down mode, the output amplifier can be configured to present a known low, medium, or high resistance output load.

The MCP4725 has an external A0 address bit selection pin. This A0 pin can be tied to V_{DD} or V_{SS} of the user's application board.

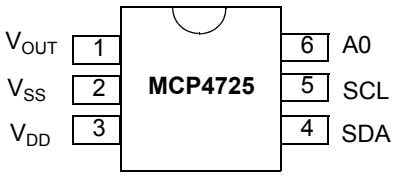
The MCP4725 has a two-wire I^2C™ compatible serial interface for standard (100 kHz), fast (400 kHz), or high speed (3.4 MHz) mode.

The MCP4725 is an ideal DAC device where design simplicity and small footprint is desired, and for applications requiring the DAC device settings to be saved during power-off time.

The device is available in a small 6-pin SOT-23 package.

Package Type

SOT-23-6



1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings†

V_{DD}.....6.5V
All inputs and outputs w.r.t V_{SS} .....–0.3V to V_{DD}+0.3V
Current at Input Pins .....±2 mA
Current at Supply Pins .....±50 mA
Current at Output Pins .....±25 mA
Storage Temperature .....-65°C to +150°C
Ambient Temp. with Power Applied .....-55°C to +125°C
ESD protection on all pins ..... ≥ 6 kV HBM, ≥ 400V MM
Maximum Junction Temperature (T_J) ..... +150°C

† Notice: Stresses above those listed under "Maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability

ELECTRICAL CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, all parameters apply at V_{DD} = + 2.7V to 5.5V, V_{SS} = 0V, R_L = 5 kΩ from V_{OUT} to V_{SS}, C_L = 100 pF, T_A = -40°C to +125°C. Typical values are at +25°C.

Parameter	Sym	Min	Typ	Max	Units	Conditions
Power Requirements						
Operating Voltage	V_{DD}	2.7		5.5	V	
Supply Current	I_{DD}	—	210	400	µA	Digital input pins are grounded, Output pin (V_{OUT}) is not connected (unloaded), Code = 000h
Power-Down Current	I_{DDP}	—	0.06	2.0	µA	V_{DD} = 5.5V
Power-On-Reset Threshold Voltage	V_{POR}	—	2	—	V	
DC Accuracy						
Resolution	n	12	—	—	Bits	Code Range = 000h to FFFh
INL Error	INL	—	±2	±14.5	LSB	**Note 1**
DNL	DNL	-0.75	±0.2	±0.75	LSB	**Note 1**
Offset Error	V_{OS}		0.02	0.75	% of FSR	Code = 000h
Offset Error Drift	ΔV_{OS}/°C	—	±1	—	ppm/°C	-45°C to +25°C
		—	±2	—	ppm/°C	+25°C to +85°C
Gain Error	G_E	-2	-0.1	2	% of FSR	Code = FFFh, Offset error is not included.
Gain Error Drift	ΔG_E/°C	—	-3	—	ppm/°C	
Output Amplifier						
Phase Margin	p_M	—	66	—	Degree(°)	C_L = 400 pF, R_L = ∞
Capacitive Load Stability	C_L	—	—	1000	pF	R_L = 5 kΩ, **Note 2**
Slew Rate	SR	—	0.55	—	V/µs	
Short Circuit Current	I_{SC}	—	15	24	mA	V_{DD} = 5V, V_{OUT} = Grounded
Output Voltage Settling Time	T_S	—	6	—	µs	**Note 3**

Note 1: Test Code Range: 100 to 4000.

2: This parameter is ensure by design and not 100% tested.

3: Within 1/2 LSB of the final value when code changes from 1/4 to 3/4 (400h to C00h) of full scale range.

4: Logic state of external address selection pin (A0 pin).

ELECTRICAL CHARACTERISTICS (CONTINUED)

Electrical Specifications: Unless otherwise indicated, all parameters apply at V_{DD} = + 2.7V to 5.5V, V_{SS} = 0V, R_L = 5 kΩ from V_{OUT} to V_{SS}, C_L = 100 pF, T_A = -40°C to +125°C. Typical values are at +25°C.

Parameter	Sym	Min	Typ	Max	Units	Conditions
Power Up Time	T_{PU}	—	2.5	—	µs	V_{DD} = 5V
		—	5	—	µs	V_{DD} = 3V Exit Power-down Mode, (Started from falling edge of ACK pulse)
DC Output Impedance	R_{OUT}	—	1	—	Ω	Normal mode (V_{OUT} to V_{SS})
		—	1	—	kΩ	Power-Down Mode 1 (V_{OUT} to V_{SS})
		—	100	—	kΩ	Power-Down Mode 2 (V_{OUT} to V_{SS})
		—	500	—	kΩ	Power-Down Mode 3 (V_{OUT} to V_{SS})
Supply Voltage Power-up Ramp Rate for EEPROM loading	V_{DD_RAMP}	1	—	—	V/ms	Validation only.
Dynamic Performance						
Major Code Transition Glitch		—	45	—	nV-s	1 LSB change around major carry (from 800h to 7FFh) (**Note 2)**
Digital Feedthrough		—	<10	—	nV-s	**Note 2**
Digital Interface						
Output Low Voltage	V_{OL}	—	—	0.4	V	I_{OL} = 3 mA
Input High Voltage (SDA and SCL Pins)	V_{IH}	$0.7V_{DD}$	—	—	V	
Input Low Voltage (SDA and SCL Pins)	V_{IL}	—	—	$0.3V_{DD}$	V	
Input High Voltage (A0 Pin)	$V_{A0\text{-}Hi}$	$0.8V_{DD}$	—	—		**Note 4**
Input Low Voltage (A0 Pin)	$V_{A0\text{-}IL}$	—	—	$0.2V_{DD}$		**Note 4**
Input Leakage	I_{LI}	—	—	±1	µA	SCL = SDA = A0 = V_{SS} or SCL = SDA = A0 = V_{DD}
Pin Capacitance	C_{PIN}	—	—	3	pF	**Note 2**
EEPROM						
EEPROM Write Time	TWRITE	—	25	50	ms	
Data Retention		—	200	—	Years	At +25°C, **(Note 2)**
Endurance		1	—	—	Million Cycles	At +25°C, **(Note 2)**

Note 1: Test Code Range: 100 to 4000.

2: This parameter is ensure by design and not 100% tested.

3: Within 1/2 LSB of the final value when code changes from 1/4 to 3/4 (400h to C00h) of full scale range.

4: Logic state of external address selection pin (A0 pin).

TEMPERATURE CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, V_{DD} = +2.7V to +5.5V, V_{SS} = GND.						
Parameters	**Sym**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
Temperature Ranges						
Specified Temperature Range	T_A	-40	—	+125	°C	
Operating Temperature Range	T_A	-40	—	+125	°C	
Storage Temperature Range	T_A	-65	—	+150	°C	
Thermal Package Resistances						
Thermal Resistance, 6L-SOT-23	θ_{JA}	—	190.5	—	°C/W	

3.0 PIN DESCRIPTIONS

The descriptions of the pins are listed in Table 3-1.

TABLE 3-1: PIN FUNCTION TABLE

MCP4725 SOT-23	Name	Description
1	V_{OUT}	Analog Output Voltage
2	V_{SS}	Ground Reference
3	V_{DD}	Supply Voltage
4	SDA	I^2C Serial Data
5	SCL	I^2C Serial Clock Input
6	A0	I^2C Address Bit Selection pin (A0 bit). This pin can be tied to V_{SS} or V_{DD}, or can be actively driven by the digital logic levels. The logic state of this pin determines what the A0 bit of the I^2C address bits should be.

3.1 Analog Output Voltage (V_{OUT})

V_{OUT} is an analog output voltage from the DAC device. DAC output amplifier drives this pin with a range of V_{SS} to V_{DD}.

3.2 Supply Voltage (V_{DD} or V_{SS})

V_{DD} is the power supply pin for the device. The voltage at the V_{DD} pin is used as the supply input as well as the DAC reference input. The power supply at the V_{DD} pin should be clean as possible for a good DAC performance.

This pin requires an appropriate bypass capacitor of about 0.1 µF (ceramic) to ground. An additional 10 µF capacitor (tantalum) in parallel is also recommended to further attenuate high frequency noise present in application boards. The supply voltage (V_{DD}) must be maintained in the 2.7V to 5.5V range for specified operation.

V_{SS} is the ground pin and the current return path of the device. The user must connect the V_{SS} pin to a ground plane through a low impedance connection. If an analog ground path is available in the application PCB (printed circuit board), it is highly recommended that the V_{SS} pin be tied to the analog ground path or isolated within an analog ground plane of the circuit board.

3.3 Serial Data Pin (SDA)

SDA is the serial data pin of the I^2C interface. The SDA pin is used to write or read the DAC register and EEPROM data. The SDA pin is an open-drain N-chan nel driver. Therefore, it needs a pull-up resistor from the V_{DD} line to the SDA pin. Except for START and STOP conditions, the data on the SDA pin must be stable during the high period of the clock. The high or low state of the SDA pin can only change when the clock signal on the SCL pin is low. Refer to **Section 7.0 "I^2C Serial Interface Communication"** for more details of I^2C Serial Interface communication.

3.4 Serial Clock Pin (SCL)

SCL is the serial clock pin of the I^2C interface. The MCP4725 acts only as a slave and the SCL pin accepts only external serial clocks. The input data from the Master device is shifted into the SDA pin on the rising edges of the SCL clock and output from the MCP4725 occurs at the falling edges of the SCL clock. The SCL pin is an open-drain N-channel driver. Therefore, it needs a pull-up resistor from the V_{DD} line to the SCL pin. Refer to **Section 7.0 "I^2C Serial Interface Communication"** for more details of I^2C Serial Interface communication.

3.5 Device Address Selection Pin (A0)

This pin is used to select the A0 address bit by the user. The user can tie this pin to V_{SS} (logic '0'), or V_{DD} (logic '1'), or can be actively driven by the digital logic levels, such as the I^2C Master Output. See **Section 7.2 "Device Addressing"** for more details of the address bits.

4.0 TERMINOLOGY

4.1 Resolution

The resolution is the number of DAC output states that divide the full scale range. For the 12-bit DAC, the resolution is 2^{12} or the DAC code ranges from 0 to 4095.

4.2 LSB

The least significant bit or the ideal voltage difference between two successive codes.

EQUATION 4-1:

$$LSB_{Ideal} = \frac{V_{REF}}{2^n} = \frac{(V_{Full\ Scale} - V_{Zero\ Scale})}{2^n - 1}$$

Where:

V_{REF} = The reference voltage = V_{DD} in the MCP4725. This V_{REF} is the ideal full scale voltage range

n = The number of digital input bits. (n = 12 for MCP4725)

4.3 Integral Nonlinearity (INL) or Relative Accuracy

INL error is the maximum deviation between an actual code transition point and its corresponding ideal transition point (straight line). Figure 2-5 shows the INL curve of the MCP4725. The end-point method is used for the calculation. The INL error at a given input DAC code is calculated as:

EQUATION 4-2:

$$INL = \frac{(V_{OUT} - V_{Ideal})}{LSB}$$

Where:

V_{Ideal} = Code*LSB

V_{OUT} = The output voltage measured at the given input code

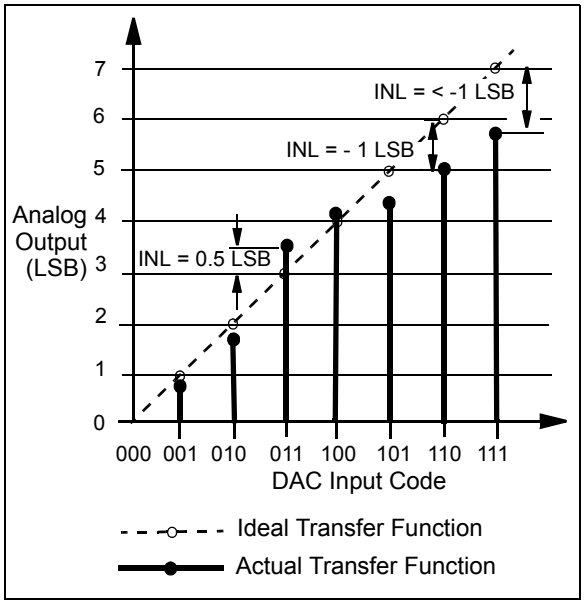


FIGURE 4-1: *INL Accuracy.*

4.4 Differential Nonlinearity (DNL)

Differential nonlinearity error (Figure 4-2) is the measure of step size between codes in actual transfer function. The ideal step size between codes is 1 LSB. A DNL error of zero would imply that every code is exactly 1 LSB wide. If the DNL error is less than 1 LSB, the DAC guarantees monotonic output and no missing codes. The DNL error between any two adjacent codes is calculated as follows:

EQUATION 4-3:

$$DNL = \frac{\Delta V_{OUT} - LSB}{LSB}$$

Where:

ΔV_{OUT} = The measured DAC output voltage difference between two adjacent input codes.

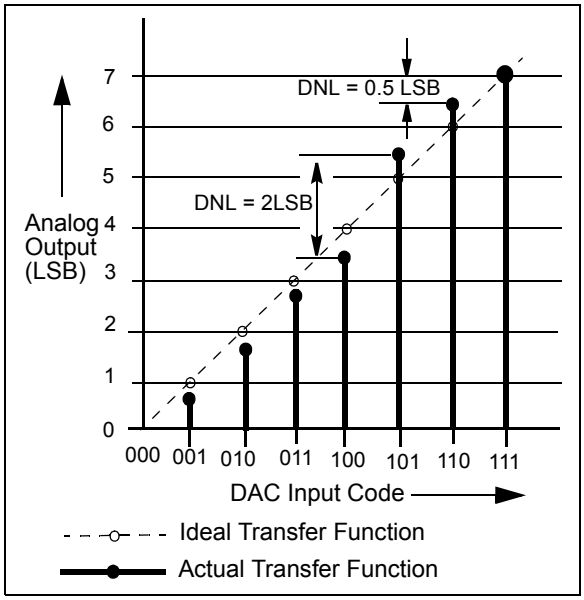


FIGURE 4-2: *DNL Accuracy.*

4.5 Offset Error

Offset error (Figure 4-3) is the deviation from zero voltage output when the digital input code is zero. This error affects all codes by the same amount. In the MCP4725, the offset error is not trimmed at the factory. However, it can be calibrated by software in application circuits.

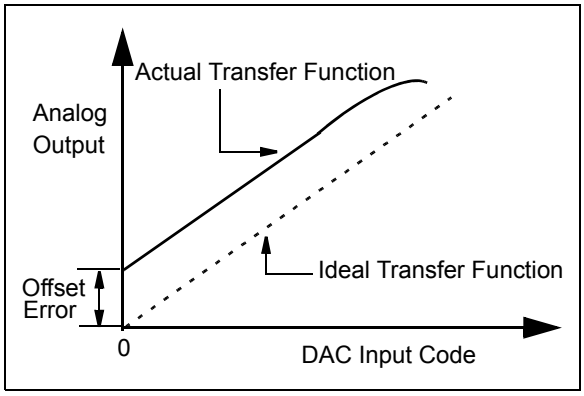


FIGURE 4-3: *Offset Error.*

4.6 Gain Error

Gain error (see Figure 4-4) is the difference between the actual full scale output voltage from the ideal output voltage on the transfer curve. The gain error is calculated after nullifying the offset error, or full scale error minus the offset error.

The gain error indicates how well the slope of the actual transfer function matches the slope of the ideal transfer function. The gain error is usually expressed as percent of full scale range (% of FSR) or in LSB.

In the MCP4725, the gain error is not calibrated at the factory and most of the gain error is contributed by the output op amp saturation near the code range beyond 4000. For the applications which need the gain error specification less than 1% maximum, the user may consider using the DAC code range between 100 and 4000 instead of using full code range (code 0 to 4095). The DAC output of the code range between 100 and 4000 is much linear than full scale range (0 to 4095). The gain error can be calibrated by software in applications.

4.7 Full Scale Error (FSE)

Full scale error (Figure 4-4) is the sum of offset error plus gain error. It is the difference between the ideal and measured DAC output voltage with all bits set to one (DAC input code = FFFh).

EQUATION 4-4:

$$FSE = \frac{(V_{OUT} - V_{Ideal})}{LSB}$$

Where:

V_{Ideal} = $(V_{REF})\ (1 - 2^{-n}) - V_{OFFSET}$

V_{REF} = The reference voltage.
$V_{REF} = V_{DD}$ in the MCP4725

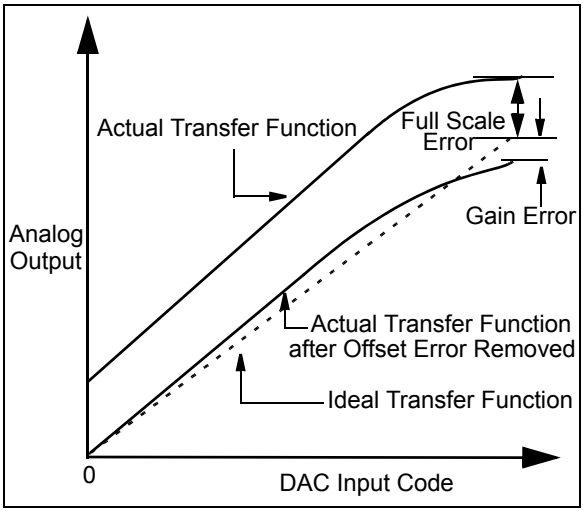


FIGURE 4-4: *Gain Error and Full Scale Error.*

4.8 Gain Error Drift

Gain error drift is the variation in gain error due to a change in ambient temperature. The gain error drift is typically expressed in ppm/^{o}C.

4.9 Offset Error Drift

Offset error drift is the variation in offset error due to a change in ambient temperature. The offset error drift is typically expressed in ppm/^{o}C.

4.10 Settling Time

The Settling time is the time delay required for the DAC output to settle to its new output value from the start of code transition, within specified accuracy. In the MCP4725, the settling time is a measure of the time delay until the DAC output reaches its final value (within 0.5 LSB) when the DAC code changes from 400h to C00h.

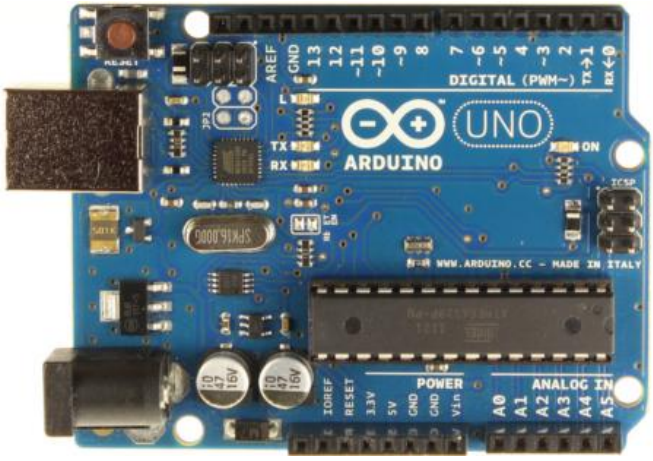
4.11 Major-Code Transition Glitch

Major-code transition glitch is the impulse energy injected into the DAC analog output when the code in the DAC register changes state. It is normally specified as the area of the glitch in nV-Sec. and is measured when the digital code is changed by 1 LSB at the major carry transition (Example: 011...111 to 100... 000, or 100... 000 to 011 ... 111).

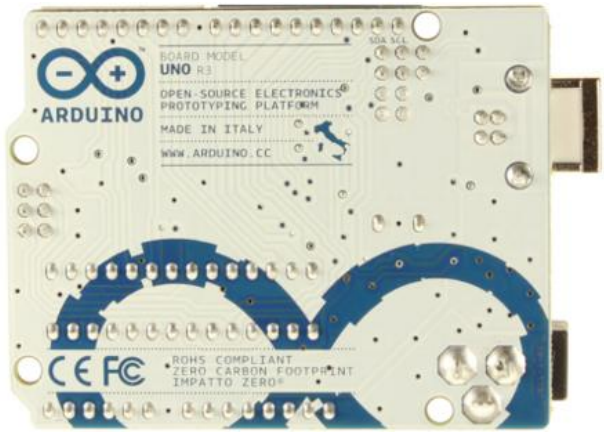
4.12 Digital Feedthrough

Digital feedthrough is the glitch that appears at the analog output caused by coupling from the digital input pins of the device. It is specified in nV-Sec. and is measured with a full scale change on the digital input pins (Example: 000... 000 to 111... 111, or 111... 111 to 000... 000). The digital feedthrough is measured when the DAC is not being written to the register.

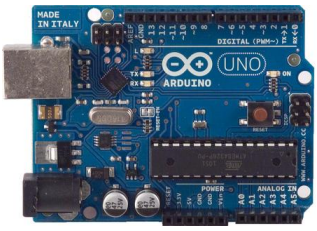
Arduino Uno



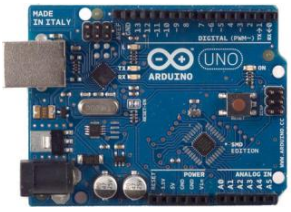
Arduino Uno R3 Front



Arduino Uno R3 Back

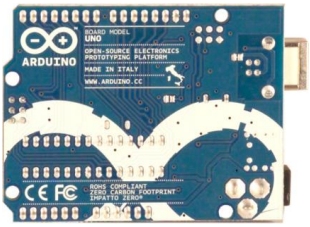


Arduino Uno R2 Front



Arduino Uno SMD

Arduino Uno Front



Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.
The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.
Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.
Revision 3 of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V

Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Schematic & Reference Design

EAGLE files: arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer)
Schematic: arduino-uno-Rev3-schematic.pdf
Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.
External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.**This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.

- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.
The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the reference and tutorials.
The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).
You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.
The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.
The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

SWITCHING POWER SUPPLY - 25 W - 12 VDC - CLOSED FRAME - FOR PROFESSIONAL USE ONLY

PSIN02512N

Support

Infosheet
Support

Features

- cooling: by free air convection

Specifications

- output: 12 VDC ± 1 %
- output current: 0.0-2.1 A
- ripple and noise: 100 mV
- power output: 25 W
- dimensions: 3.94" x 3.82" x 1.38"
- weight: 12.35 oz
- overload protection: 105-150 % fold back current limiting with auto-recovery
- input voltage: 100-120 Vac/200-240 Vac (switch selectable) 50/60 Hz
- input rush current: 15 A @ 110 V
- setup, rise and hold-up time: 200 ms, 100 ms, 300 ms at full load and 230 VAC
- working temperature:
  - 14 °F to 32 °F: 80 % load
  - 32 °F to 122 °F: 100 % load
  - 122 °F - 140 °F: 60 % load
- operating humidity conditions: 20-90 % RH
- overload protection: 105-150 % fold back current limiting with auto-recovery
- withstand voltage (max. 1 min):
  - I/P - O/P: 1.5 kVAC
  - I/P - FG: 0.5 kVAC
- DC adjustment range: ± 10 % rated output voltage (V adj)
- efficency: 74 %
- connection: 5, 7, or 9 pole terminal block, pitch 0.37"

REACH communicating information according to article 33
WEEE