Final Report: Control of Deflection Coils

Max Huggins – UCA Department of Physics Astronomy

February 9, 2021

Abstract

This experiment was done to determine a method of generating arbitrary analog functions from a digital signal and using them to control the deflection coils of a cathode ray tube (CRT) type accelerator. The MCP4921 Digital to Analog Converter Chip was used to convert digital functions to analog ones, the new analog function was amplified with a class AB amplifier circuit, and the amplified signal was put into the horizontal and vertical deflection coils of an old CRT TV.

1 Introduction

Electrostatic accelerators are somewhat of a forgotten technology. CRT TVs dominated the market for over half a century, but have since gone out of style. The first particle accelerators were that of the electrostatic type but have also found their limitations when compared to modern linear accelerators. This being said, the physics involved is not only an interesting topic to explore but is also an excellent way to develop new skills. The objective of this experiment was to determine a method of generating arbitrary analog functions from a digital source and use these functions to control the deflection coils of a CRT TV.
The MCP4921 is a digital to analog converter (DAC) chip that can be controlled with a serial peripheral interface communications protocol. With this, a function (such as $sinx$) is defined. The values at each iteration of the function are placed into digital slots and fed to the DAC. The DAC then forms a continuous signal from the data input. The more bits available to feed digital values, the smoother the analog function will output. This is shown in figure 1

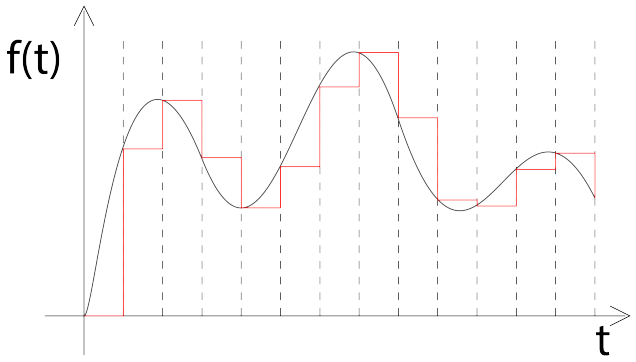

Figure 1: Here is an example of how the digital signal is converted to an analog one.

Since an X-Y input was desired, a second DAC was used for separate control. Both of these signals were fed into two class AB amplifiers which increase the total power output of the two functions. This amplified signal could then be fed into the deflection coils of a CRT tube.

2 Analysis and Setup

Lissajous patterns could be created by varying the phase difference of the X and Y functions and several interesting patterns were observed. Some are shown below.

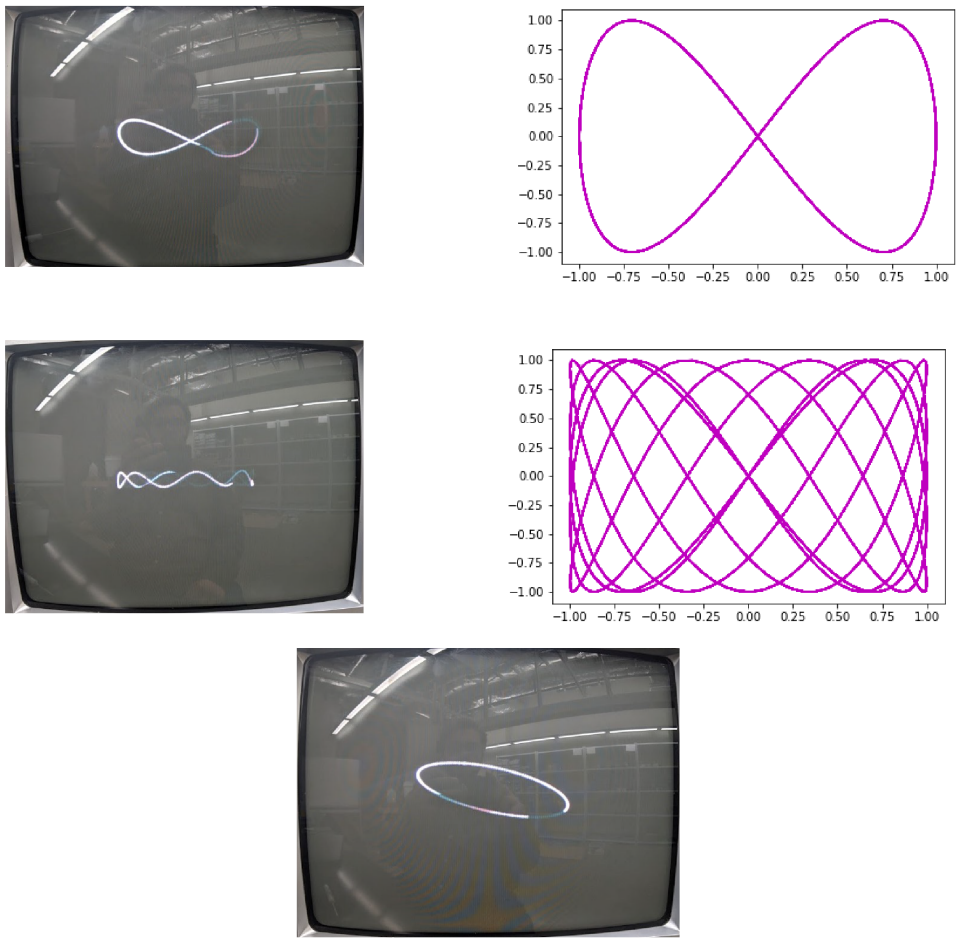

Figure 2: Here are several patterns from the CRT.

The computational models were made plotting a sine function on x and y coordinates at different phase angles. Unfortunately, these patterns are not totally seen on due to my camera's low refresh rate and the function's low frequency. It can be seen that the deflection coils do not have the same impedance for the horizontal and vertical deflection. This is because the typical CRT uses different refresh rates for the two axes and in order to transfer power more efficiently, the coils are designed to match impedance with these and because of it we do not get 1:1 gain on the separate axes. This is not hard to take into consideration and there are two methods that can be used to ensure the signals match. The first is simply modifying the arbitrary signal's amplitude in either direction. The next, a more robust method, is to match the input impedance for the drooping axis. This would ensure the DAC's resolution is not limited by changing the input function's range. Unfortunately, the second was not covered in this paper.

Some important considerations to note are discussed next. The first in regards to reference voltages. The Raspberry Pi's 3.3V and 5V rails are not reliable reference voltages. They have fluctuations that smoothing capacitors on the rails won't be able to take care of. You are already at a disadvantage when it comes to converting digital signals to analog ones and these fluctuations can cause major value shifts in the new functions. When it comes to using

the DAC chips it is most important to:
(1) Limit voltage fluctuations in reference voltages
(2) Limit the range of digital values
(3) It is going to be an easier task to create a small range of digital values given a resolution and amplify this signal with a well-designed amplifier than it is going to be creating a large range of voltage values for your DAC chip to make analog. Analog amplifiers can have excellent signal transfer with high gains where the DAC is limited by a certain resolution (4096 in this case.)
(4) Another thing to note that may be found in hindsight otherwise, is ensuring scripts are efficient. Print statements will bog down the digital to analog conversion quite noticeably!
The python script used for creating the arbitrary functions is outlined here:

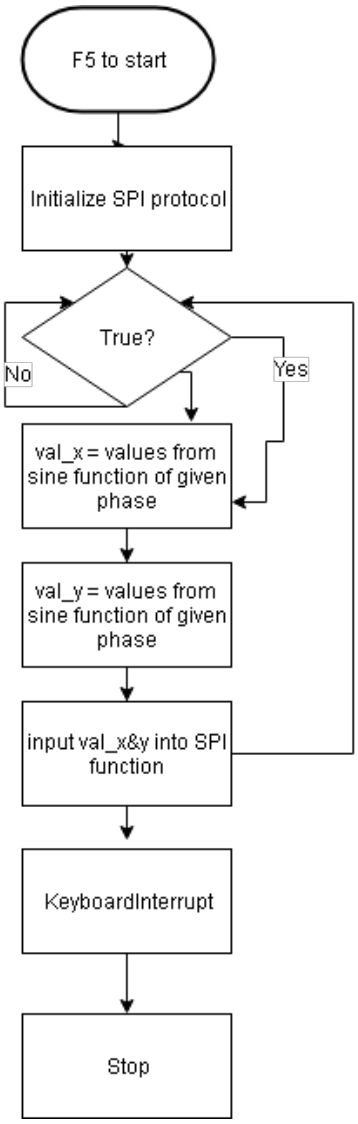

Figure 3: Here is the flow of the communication script.

The most important features of this script is the serial peripheral interface communica-

tions protocol. This is more complex than the simple ADC communication protocol. The DAC and amplifier circuits are shown here. The two coils represent the vertical and horizontal deflection coils.

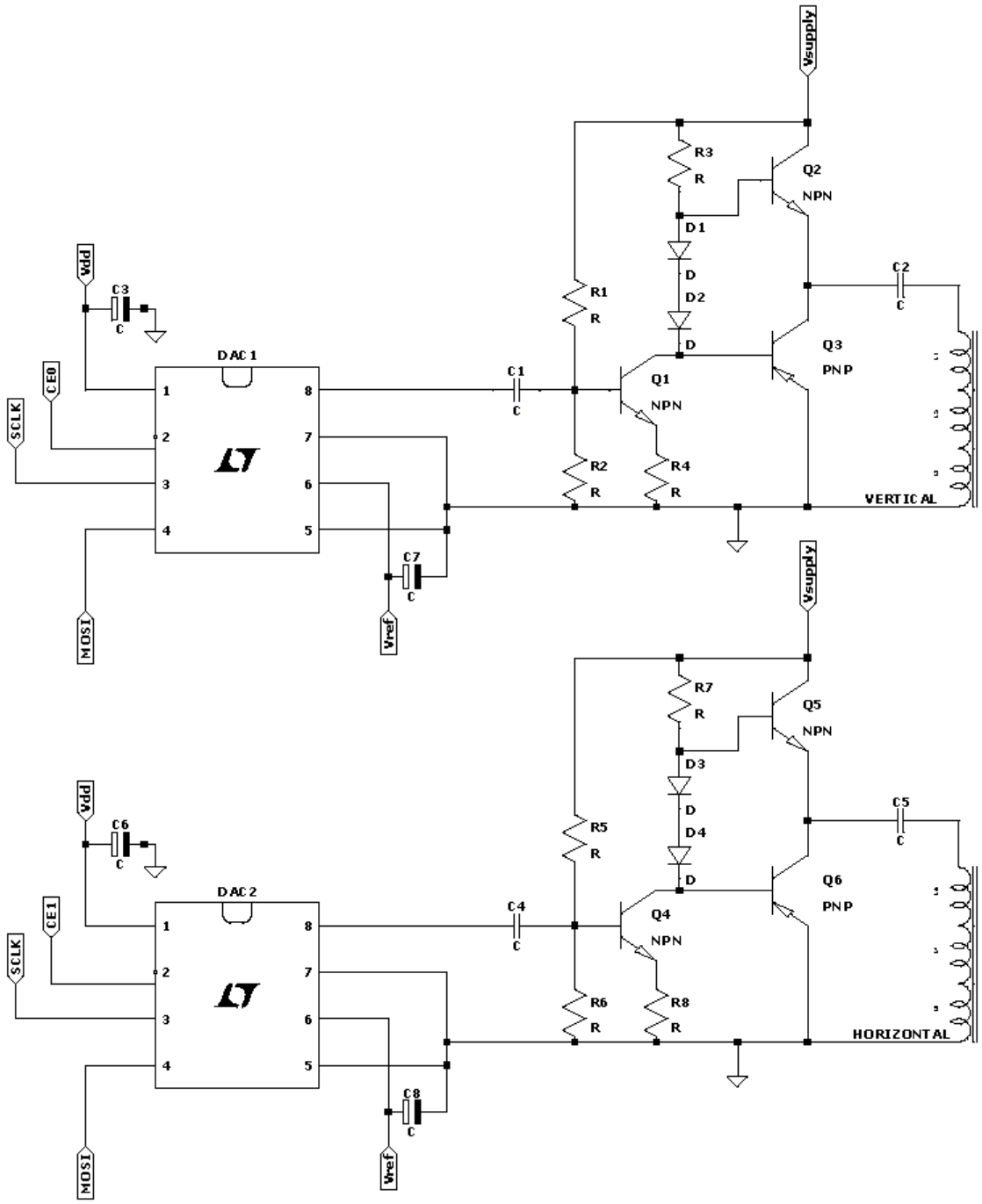

Figure 4: Here is the circuit schematic for the driver.

The CRT tube boils electrons from a tungsten filament. These electrons are accelerated

at large velocities and forced into a thin beam. The deflection coils use a magnetic field which produces a force given by equation 1.

$$F = qvB \tag{1}$$

Where q is the fundamental unit of charge, v is the velocity of the electron and B is the magnetic field produced by the coil. The magnetic field of the coil is proportional to the current (I) running through it and the geometry of the coil, and number of windings (length of wire, l).

$$F = BIl \tag{2}$$

Modeling this magnetic field is not a simple task so it is helpful to calibrate values based on observation. It was found that an input power of approximately 10W was enough to scan electrons from each side of the screen at any given frequency. Unfortunately, the accuracy of the DAC's limit the magnitude of the voltage values that can be used otherwise the output signal is compromised.

3 Conclusion

The methods used here can be improved and expanded on in several ways. The first is to sink heat away from the amplifier's transistors because of the amount of power dissipated through them. Next, higher resolution DAC's can be used to increase voltage magnitudes of the arbitrary functions and thus form a better pattern on the makeshift oscilloscope. Also, we have essentially made an arbitrary function generator that can be used for electronics projects. And finally, the ideas used here can now be used to create the deflection coils for an amateur electrostatic accelerator with scanning capabilities. Here we are limited by the beam characteristics of the accelerator, but when designing our own it would be of great use to have beam scanning in mind through design.

Python script with SPI communication protocols.

```
#=========================Modules=============================================#
import spidev
import numpy as np
import RPi.GPIO as GPIO
#=============================Constants=======================================#
N = 12
C = 0
CS = 32
CLK = 36
DOUT = 40
DIN = 38
#=========================SPI communication protocols=========================#
DEBUG = False
spi_max_speed = 4 * 1000000 # 4 MHz
V_Ref = 3300 # 3V3 in mV
Resolution = 2**N # 12 bits for the MCP 4921
CE_1 = 0 # CE0 or CE1, select SPI device on bus

# setup and open an SPI channel
spi_1 = spidev.SpiDev()
spi_1.open(0,CE_1)
spi_1.max_speed_hz = spi_max_speed

CE_2 = 1 # CE0 or CE1, select SPI device on bus

# setup and open an SPI channel
spi_2 = spidev.SpiDev()
spi_2.open(0,CE_2)
spi_2.max_speed_hz = spi_max_speed

def setOutput(val,xoy):
    # lowbyte has 8 data bits
    # B7, B6, B5, B4, B3, B2, B1, B0
    # D7, D6, D5, D4, D3, D2, D1, D0
    lowByte = val & 0b11111111
    # highbyte has control and 4 data bits
    # control bits are:
    # B7, B6,   B5, B4,     B3,  B2,  B1,  B0
    # W  ,BUF, !GA, !SHDN,  D11, D10, D9,  D8
    # B7=0:write to DAC, B6=0:unbuffered, B5=2:Gain=1X, B4=1:Output is active
    highByte = ((val >> 8) & 0xff) | 0b0 << 7 | 0b0 << 6 | 0b1 << 5 | 0b1 << 4
    #
    # by using spi.xfer2(), the CS is released after each block, transferring
    the
    # value to the output pin.
    if xoy == 'x':
        spi_1.xfer2([highByte, lowByte])
    elif xoy == 'y':
        spi_2.xfer2([highByte, lowByte])
#=========================Data Acquisition====================================#
try:
    phase = np.pi/10 #This helps make the Lissajous curves
```

```
    while True:
        for angle in np.linspace(0,360,10000000): #lots of data points and
            angle = angle * ((2 * np.pi) / 360) #high fequency is good for DAC

            val_x = .5*np.sin(100000*angle)
            val_x = int((val_x + 1 ) * 2**N / 8)
            setOutput(val_x,'x') #x values

            val_y = .5*np.sin(100000*angle*phase)**2
            val_y = int((val_y + 1 ) * 2**N / 8)
            setOutput(val_y,'y') #y values
#====================Get the hell outta dodge=============================#
except KeyboardInterrupt:
    print("Closing SPI channel")
    spi_1.close()
    spi_2.close()
    GPIO.cleanup()
```

Python script for useful functions.

```
import RPi.GPIO as GPIO
import time
import numpy as np

GPIO.setmode(GPIO.BOARD)

C = 0
CS = 32
CLK = 36
DOUT = 40
DIN = 38

GPIO.setup(CS, GPIO.OUT)
GPIO.setup(CLK, GPIO.OUT)
GPIO.setup(DOUT, GPIO.IN)
GPIO.setup(DIN, GPIO.OUT)

#=============================================================================#

def readADC():
    #set initial binary to as empty
    d = ''
    GPIO.output(CS,False)
    #oneclock pulse
    #set CLK low
    GPIO.output(CLK,False)
    #Set clk high
    GPIO.output(CLK,True)
    GPIO.output(CLK,False)
    #end CLK pulse
    #now to read data synced to more clokc pulses
    for n in range(0,8): #read in 8 bits, 0-7
        #one clock pulse
        #set CLK low then high
        GPIO.output(CLK,False)
```

```
        GPIO.output(CLK,True)
        GPIO.output(CLK,False)
        DO_state = GPIO.input(DO)
        if DO_state == True:
            d = d+'1'
        else:
            d = d + '0'
        #Do until all bits are read
    #End convo w/ ADC
    GPIO.output(CS,True)
    #Return binary
    return d

#=============================================================================#

def readMCP(C, CS, CLK, DOUT, DIN):
    d = ''
    #These next few lines start communication with our friends at channel 1
    GPIO.output(CS, False)
    GPIO.output(DIN, True)
    GPIO.output(CLK, False)
    GPIO.output(CLK, True)
    GPIO.output(CLK, False)
    ########################
    #Input bit selections
    #Sengle ended(not differential)
    #For channel 0 see pg 19 in data sheet
    if C == 0:
        din_control = '1000'
    if C == 1:
        din_control = '1001'
    if C == 2:
        din_control = '1010'
    if C == 3:
        din_control = '1011'
    if C == 4:
        din_control = '1100'
    if C == 5:
        din_control = '1101'
    if C == 6:
        din_control = '1110'
    if C == 7:
        din_control = '1111'
    for n in din_control:
        if n == '1':
            GPIO.output(DIN, True)
        else:
            GPIO.output(DIN, False)
        ############################
        GPIO.output(CLK, False)
        GPIO.output(CLK, True)
        GPIO.output(CLK, False)
        ############################
    GPIO.output(CLK, False)
```

```
    GPIO.output(CLK, True)
    GPIO.output(CLK, False)
    ############################
    #This reads in the data synced to the clock pulses
    for n in range(0,10):
        GPIO.output(CLK, False)
        GPIO.output(CLK, True)
        GPIO.output(CLK, False)
        #Listen to the DOUT pin
        DOUT_state = GPIO.input(DOUT)
        if DOUT_state == True:
            d = d + '1'
        else:
            d = d + '0'
    #Finish talking to MCP
    GPIO.output(CS, True)
    GPIO.output(DIN, False)
    return d

#=============================================================================#

def calc_volts(d, AoM, ref):
    d_int = int(d,2)
    if AoM == 'ADC':
        volts = ref*d_int / 256
        volts = round(volts, 2)
    if AoM == 'MCP':
        volts = ref*d_int / 1024
        volts = round(volts, 3)
    return volts

#=============================================================================#

V_0 = .5
T_c = .01
R1 = 9840

def calc_temp(ref,volts,sensor):
    if sensor == 'LM34':
        temp = volts / .01
        temp = (temp-32)*(5/9)
    elif sensor == 'BudgetLM34':
        temp = (np.abs(volts - V_0)) / T_c
    elif sensor == 'Thermistor':
        Resistor_value = (volts * R1)/(ref - volts)
        temp = (1 /(
                    1.506*10**(-3) +
                    1.615*10**(-4) * np.log(Resistor_value)+
                    4.242*10**(-7) * (np.log(Resistor_value))**3)
                ) - 273
    temp = round(temp,2)
    return temp

#=============================================================================#
```

```
144
145 def calc_resist_Photoresist(ref,volts):
146     Resistor_value = (volts * R1)/(ref - volts)
147     if Resistor_value < 15000:
148         return Resistor_value
149
150 #=============================================================================#
```