CSU Northridge

Vandalism Bot

Final Project

**Author:**                                                                                             **Instructor:**

Ian O’Donnell                                                                                               Dr. Mirzaei

**Performed On:**                                                                                             **Date:**

04/22/21                                                                                                       05/17/2021

**Abstract**

This report examines the CAN bus interface to interface with a slave controller.

**TABLE OF CONTENTS**

**Abstract……………………………………………………………………………………………………………....1**

**Introduction………………………………………………………………………………………………………...3**

**Methodology……………………………………………………………………………...…………………………4-5**

**Testing Strategy…………………………………………………………………………..………………………6**

**Results…………………………………………………………………………..…………………………………7-10**

**Conclusion…………………………………………………………………………………………………………..11**

**Appendix A: Source Code ……………………………………………………………………………………... 12-18**

**Section 1: Introduction**

This project utilizes a Controller Area Network IP to utilize the Zybo’s CAN Peripheral. The data bus is one of the most critical aspects of a control and data handling system. This can be seen from applications from cars to spacecrafts. One of these data bus protocols is the Controller Area Network (CAN) bus. The Pros for using the CAN bus is that it is robust, low cost and has a low power consumption. This protocol is known for its applications in the automotive industry. A few other applications the CAN Bus is used for includes building automation, lighting control systems, and ships. This lab only has two CAN nodes, but the CAN bus can typically have up to 32 nodes on its system.

The objective this project is to understand how the CAN bus protocol works. Although the CAN bus may not be the best method for this application, the knowledge of how it works can be applied to other applications. This project also includes AXI GPIO (LEDs, Buttons, LEDS), and PMOD GPIO (Solid State Relay). The scope of this project is to be able to determine if the image that is captured by a camera controlled by a raspberry pi has been changed or not.

The function of the switches are the following: Switch 1(Silence Alarm), Switch 2 (Turn Alarm On / Off ), Switch 3 (Send Message to Slave Device). The functions of the buttons are the following: Button 1 (Set Instruction to Store new Default Image), Button 2 (Set Instruction to Compare Default Image and New Image), Button 3 (Set Instruction to Display Images on Monitor). The LEDs will display which button is currently active.

Finally, the image processing aspect of this project is not fully developed as it is out of the scope to fully develop this aspect. The equation, shown in figure 1.1 was researched and used to determine the amount of changed pixels between two pictures.

*err = np.sum((imageA.astype("float") - imageB.astype("float")) \*\* 2)*

*err /= float(imageA.shape[0] \* imageA.shape[1])*

**Figure 1.1:** Image processing Pixel Change equation

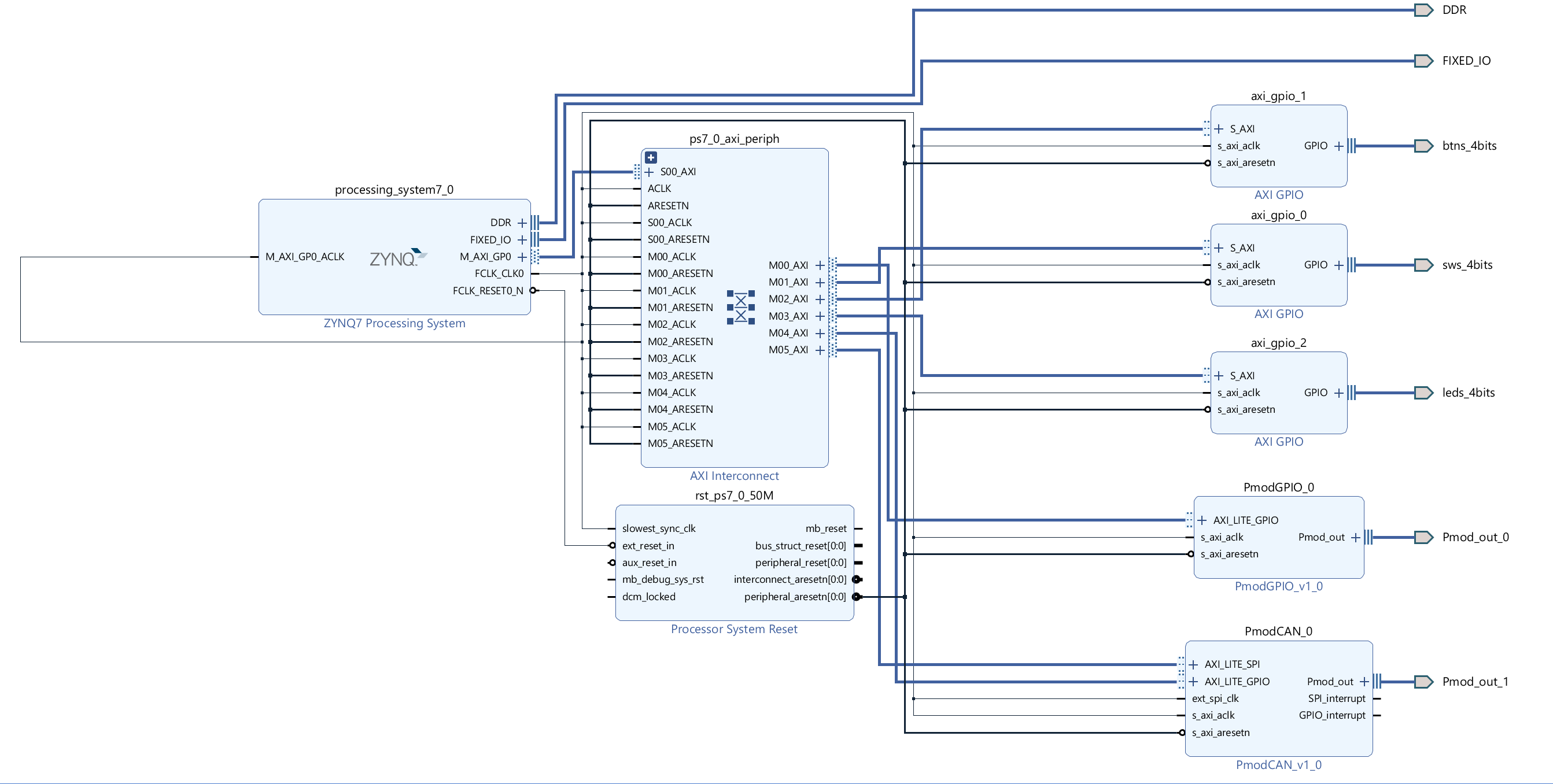
The purpose of this algorithm is to determine if an image has been vandalized, as seen in Figure 1.2.

**Figure 1.2:** A clean and vandalized stop sign image.

**Section 2: Methodology**

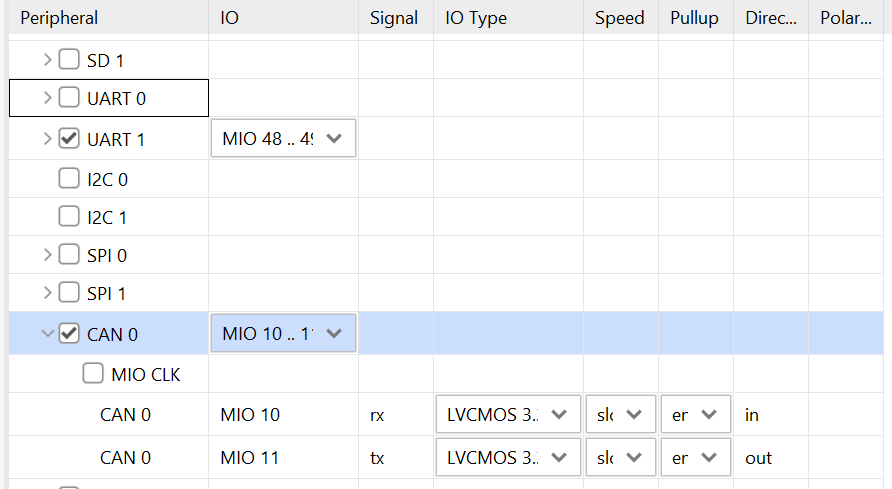
The first step in this design is to create a hardware block design. This block diagram can be seen in figure 2.1.



**Figure 2.1:** Hardware Block Diagram

After the ZYNQ7 IP is created, the AXI GPIO modules for the LED’s, Buttons, and Switches can be connected to it. Next, vivado-library-master is downloaded from Xilinx’s github page and added to the project sources. This allows for PmodCAN\_0 and PmodGPIO\_0 IPs to be easily added to the hardware block diagram.

Before exporting this design, the ZYNQ7 Processing System must be modified to enable the CAN peripherals, as seen in figure 2.2.



**Figure 2.2:** Zynq Peripherals

It can be seen that Can 0 is enabled and the IO is set to MIO 10..11.

The hardware is now ready to have a bitstream generated and have the hardware exported to SDK. With Vitis opened, the source file *run.c* located at *FinalProject\Source\_Files* can be imported. The rest of the configuration for the Zybo is configured as per the instructions found at *Zybo\_README.md.*

The next step is to configured the raspberry pi, as per the instructions found in *Python\_README.md*. As per the README file, *Project\_Schematic.pdf* should also be referenced on how the raspberry pi should be wired. The source files for this project can be found at *FinalProject\Source\_Files\PythonCAN.* The threshold for the python code is now calibrated. Using the stop sign images, shown in Figure 1.2, the difference was calculated at value 1000. And the difference between the same image is 0.0.

*pi@raspberrypi:~/Desktop/OpenCV $ python3 test.py*

*error: 1030.7716938271606*

*error: 0.0*

However, images taken from a camera are never that exact. Because of this, when a value of 1000 is used in the run.py program, every new image will trigger the vandalism alarm. Because of this, an error value of 4000 is used for testing to verify that the image has been modified. Ideally, an additional step of image processing would be done on the image before the error detection is computed on the two images so that the error would be easier to detect.

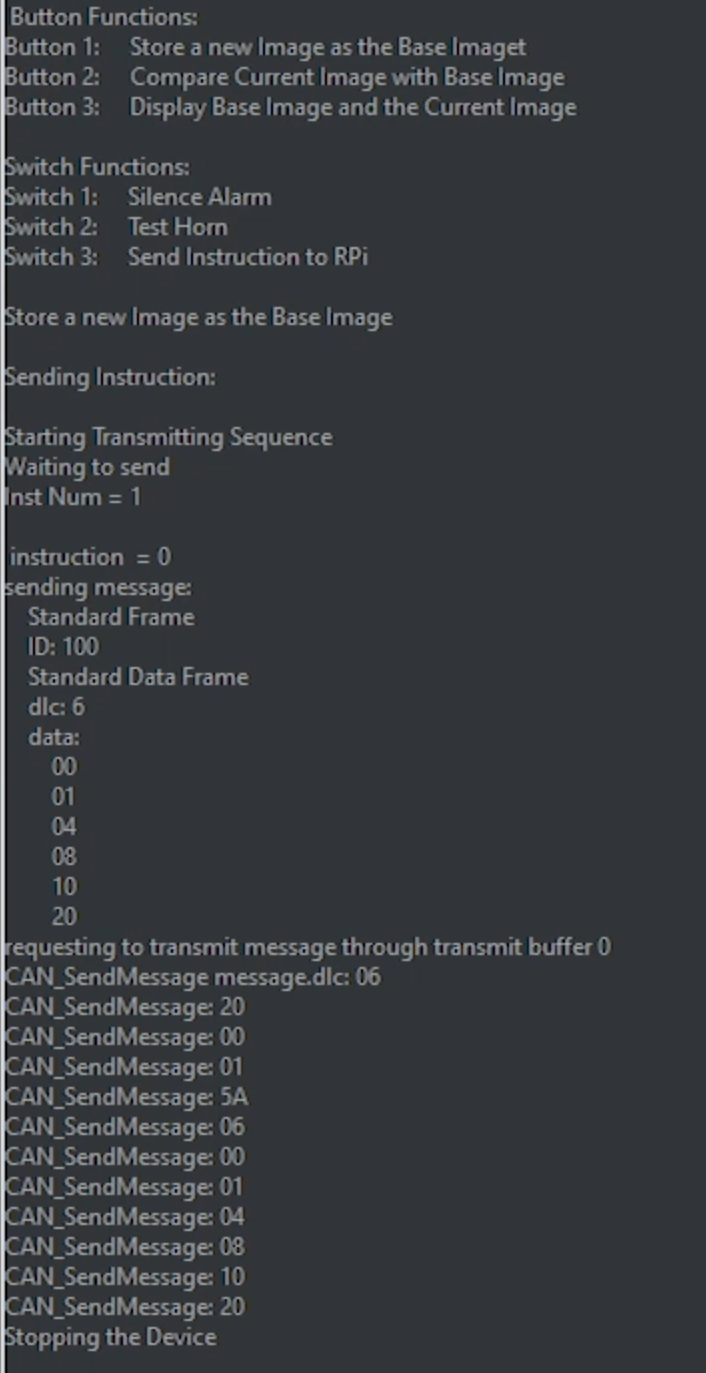
**Section 3: Testing Strategy**

This design can be tested by doing the following tests.

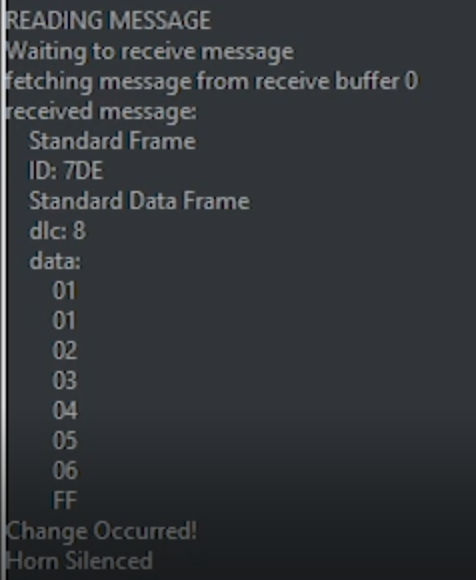
|  |  |
| --- | --- |
| **Perform on Zybo** | **Verify on Vitis Serial Terminal**  **(unless stated otherwise)** |
| Press Button 1 | Verify Message “Instruction Set: Store new Default Image” |
| Turn Switch 3 ON | Verify on **RPi** that the message was sent |
| Turn Switch 3 OFF and Press Button 2 | Verify Message “Instruction Set: Take Photo” |
| Turn Switch 3 ON | Verify on **RPi** that the message was sent |
| Turn Switch 3 OFF and then change the image in the camera somehow |  |
| Turn Switch 3 ON | Verify on **RPi** that the message was sent and the alarm turns ON |
| Turn Switch 1 ON | Silence the Alarm |
| Turn Switch 3 OFF and Press Button 3 | Verify Message “Instruction Set: Display Stored and Current Image” |
| Turn Switch 3 ON | Verify on **RPi** that the message was sent – Stored Images are displayed |

**Section 4: Results**

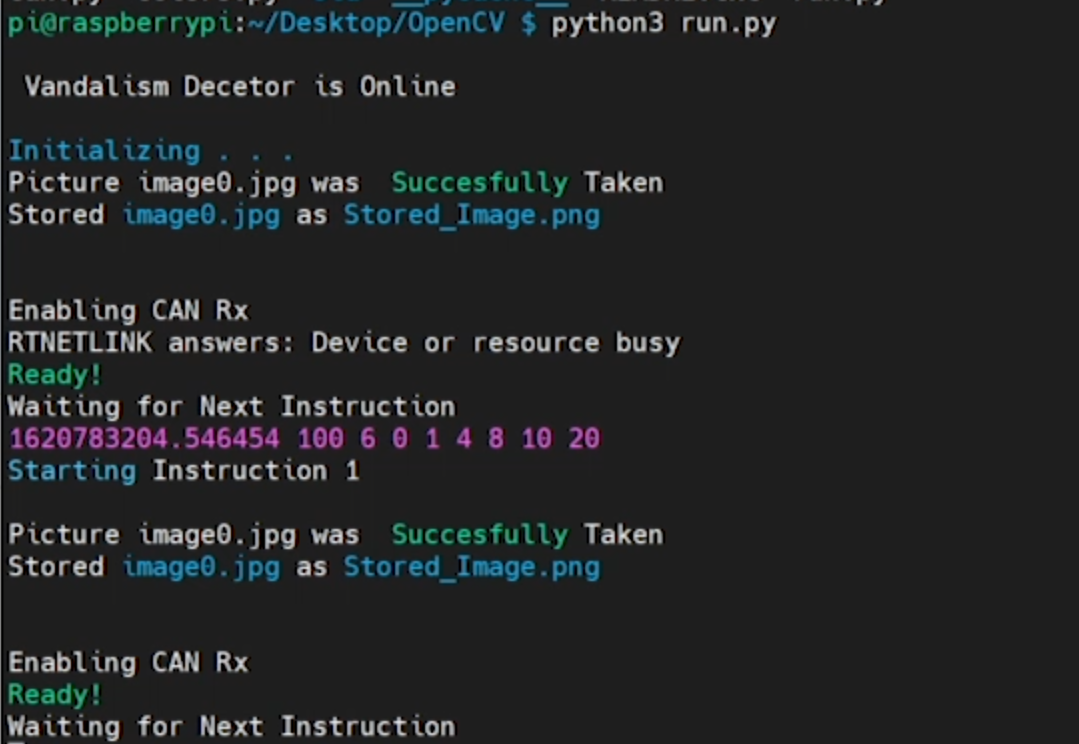
The results of a receive, transmit and images displayed can be seen in figure 4-1, 4-2, 4-3, 4-4. The results of what images are considered not changed and a set of images that are considered change are shown in figure 4-5 and 4-5.



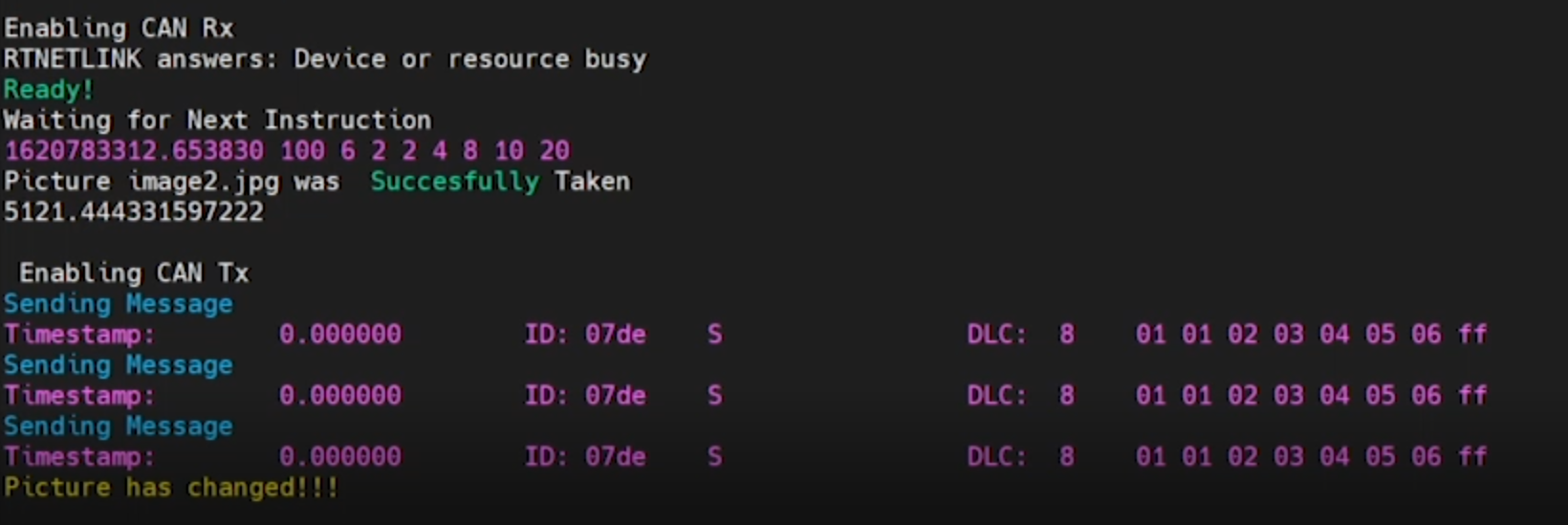
**Figure 4-1:** Zybo: Storing Instruction and Sending Message



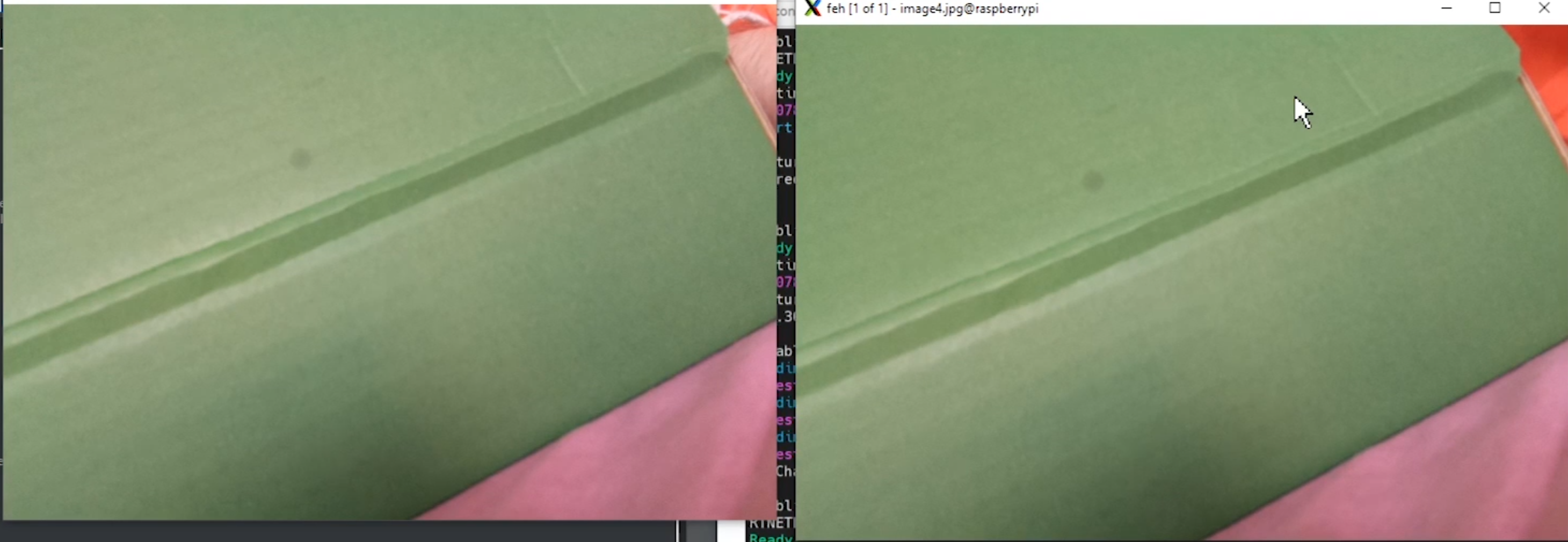
**Figure 4-2:** Zybo: Receiving a message



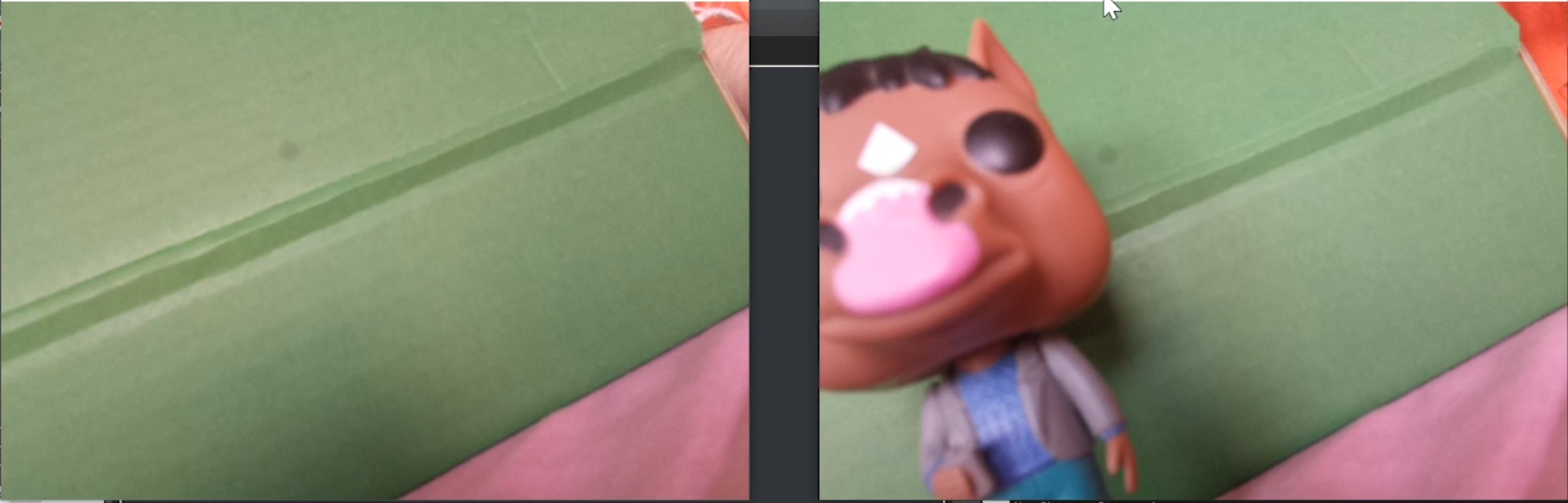
**Figure 4-3:** Raspberry Pi Receiving Message and decoding it



**Figure 4-4:** Raspberry Pi Sending Messages



**Figure 4-5:** Stored Image and Captured Image (No Vandalism)



**Figure 4-5:** Stored Image and Captured Image (Vandalism )

**Section 5: Conclusion**

This project was designed successfully with a Controller Area Network initialized with the Zybo controller to communicate with another device in both the software and hardware.

There are still a lot of future goals that can be done with this project. A few of these include, including an interrupt to switches. This will allow for the horn to be silenced immediately, regardless of what the current operation is. Additionally, an interrupt for the can messages (with a lower priority than the horn) will ensure that message packets are not missed. Another addition to this project includes adding a motion sensor to the input of the Zybo. This would allow for a new image to be taken each time motion is captured. This will eliminate the need to manually take pictures. Finally, adding the ability to store information on the vandalism would be useful. Even if the information is as simple as the time stamp on when the vandalism occurred.

**Appendix A: Source Code**

The source code for this project can be found below and at *Ian\_ODonnell\_MP1\Code*.

/\*--------------------------------------------------------------

 526 L                  Mini Project                Spring 2021

----------------------------------------------------------------

LAB Title: Final Exam Take Home Mini Project 1

File Name: Mini\_Project.c

Author: Ian O'Donnell

----------------------------------------------------------------

Version                 Date                    Description

1.0                     3/25/21              Initial Release

----------------------------------------------------------------

Purpose:

    generates a periodic waveform defined by five parameters:

    1. Final value, 2. Drop value, 3. Step Horizontal value,

    4. Step Vertical Value, 5. Period

---------------------------------------------------------------\*/

#include <stdio.h>

#include "xbasic\_types.h"

#include "platform.h"

#include "xil\_printf.h"

#include "xparameters.h"

#include "xstatus.h"

#include <time.h>

#include "sleep.h"

//write to BRAM: Xil\_Out16(address, value);

//read from BRAM: int num = Xil\_In16(address);

/\*###############################################################################################################

#   Purpose: Creates a delay for the program - Could also use sleep command

###############################################################################################################\*/

void DELAY(int delay)

{

    for (volatile int Delay=0; Delay<delay; Delay++);

}

/\*###############################################################################################################

#   Purpose:    Tells user if input values will generate an error

#   Input:      User Input Values

#   Output:     Print error message

###############################################################################################################\*/

void Error\_Handler(int Step\_Horizontal\_Value, int Step\_Vertical\_Value, int Period, int Drop\_Value, int Final\_Value)

{

//These Errors are not 'handled' as the design does not require it but the messages will display for the user.

    int Distance\_Traveled = Drop\_Value/Step\_Vertical\_Value \* Step\_Horizontal\_Value;

    if (Distance\_Traveled >= Period)

    {

        printf( "Period Value is too small, Final Value will not be reached. (Error: %d <= %d)\n", Period, Distance\_Traveled );

    }

    if ( Step\_Horizontal\_Value > Period )

    {

        printf("Horizontal Value must be less than the Period\n");

    }

    if ( Step\_Vertical\_Value > Drop\_Value )

    {

        printf( "Vertical Step Value must be less than the Drop Value\n");

    }

    if (Drop\_Value > Final\_Value)

    {

        printf( "Drop Value is greater than your Final Value, Can not use negative values");

    }

}

/\*###############################################################################################################

#   Purpose:    Generate Waveform based on user input values. First Value will be the Final Value.

                The wave form will ramp up, from the dropped value, until the final value is reached.

                The X axis is the period and y value is the vertical value.

#   Input:      User Input Values

#   Output:     Print error message

###############################################################################################################\*/

int main()

{

    short int Final\_Value, Drop\_Value, Period, Step\_Vertical\_Value, Step\_Horizontal\_Value;

    //initialize

    xil\_printf("Mini Project!\n");

    xil\_printf("Gathering User Inputs. . .\n");

/\*  //Hardcoded Values

    Final\_Value = 10000;

    Drop\_Value = 8500;

    Period = 851;

    Step\_Vertical\_Value = 20;

    Step\_Horizontal\_Value = 2;

\*/

    //Allow User to Enter Values in:

    printf("Enter The Final Value: ");

    scanf("%hu", &Final\_Value);

    printf("Enter Drop\_Value: ");

    scanf("%hu", &Drop\_Value);

    printf("Enter The Period: ");

    scanf("%hu", &Period);

    printf("Enter The Step\_Vertical\_Value: ");

    scanf("%hu", &Step\_Vertical\_Value);

    printf("Enter The Step\_Horizontal\_Value: ");

    scanf("%hu", &Step\_Horizontal\_Value);

    //Error Handler

    xil\_printf("Checking for Errors . . .\n");

    Error\_Handler(Step\_Horizontal\_Value, Step\_Vertical\_Value, Period, Drop\_Value, Final\_Value);

    //Write to BRAM (Calculate Values)

    xil\_printf("Generating X Values. . .\n");

    int Current\_Value = 0;

    int n = 2; //Step 2

    int H\_Max = n \*(Period / Step\_Horizontal\_Value);        //number of iterations needed for runs

    //Write the Horizontal Values (X) to the BRAM (H\_Max will always be greater or equal to V\_Max?)

    for (int j = 0; j <= H\_Max; j = j + n)

    {

        if (j == 0)

        {

            Current\_Value == 0;

            Xil\_Out16(j, Current\_Value);                            //First value will always start at 0

        }

        else

        {

            Current\_Value = Current\_Value + Step\_Horizontal\_Value;  //Additional Values will be incremented by the step value

            Xil\_Out16(j, Current\_Value);

        }

    }

    //Write the Vertical Values (Y) to the BRAM (in address starting after the last horizontal value.

    xil\_printf("Generating Y Values. . .\n");

    int Y\_Value = 0;

    int Y\_Base = H\_Max + n;

    for (int i = Y\_Base; i<= Y\_Base\*2; i = i + n)

    {

        if (i == Y\_Base)

        {

            Y\_Value = Final\_Value;                                      //First Value is Final Value

            Xil\_Out16(i, Y\_Value);

        }

        if (i == Y\_Base + n)

        {

            Y\_Value = Final\_Value - Drop\_Value;                         //Second Value is Final Value - Drop Value

            Xil\_Out16(i, Y\_Value);

        }

        else

            if ((Y\_Value + Step\_Vertical\_Value) < Final\_Value)          //be looking one step ahead to verify it does not go over final value

            {

                Y\_Value = Y\_Value + Step\_Vertical\_Value;                //Additional Values will be incremented by the step value

                Xil\_Out16(i, Y\_Value);

            }

            else                                                        //Don't increment Current\_Value once it has reached the Final Value, alsi set it to Final Vlaue

            {

                Xil\_Out16(i, Final\_Value);

            }

    }

    //Read Values (There is a chance that there are more X values than Y values, but never more Ys than Xs?

    xil\_printf("Final Value = %d, Drop Value = %d Period = %d, \nStep Vertical Value = %d Step Horizontal Value = %d\n",

                Final\_Value, Drop\_Value, Period, Step\_Vertical\_Value, Step\_Horizontal\_Value);

    xil\_printf(" X , Y \n");

    for (int k = 0; k <= H\_Max + 1; k = k + n)

    {

        int H\_Read = Xil\_In16(k);

        int Y\_Read = Xil\_In16((Y\_Base + k));

        xil\_printf(" %d , %d \n",H\_Read, Y\_Read);

    }

}

**Figure A-1:** Mini\_Proj\_1.c source code

/\*--------------------------------------------------------------

 526 L                  Mini Project                Spring 2021

----------------------------------------------------------------

LAB Title: Final Exam Take Home Mini Project 1

File Name: Graphing\_Script.c

Author: Ian O'Donnell

----------------------------------------------------------------

Version                 Date                    Description

1.0                     3/25/21              Initial Release

----------------------------------------------------------------

Purpose:

    Plot Data from an Excel File

---------------------------------------------------------------\*/

import csv

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.patches import StepPatch

from tkinter import \*

from tkinter.filedialog import askopenfilename

import os

import math

#Configure These Values

x\_data\_points = 50              #Enter # Of X Data Points

xlabel = 'Period'

ylabel = 'Value'

Title = 'Mini Project 1 - Test 3'

#School Colors:

Black = '#000000'

Patone = '#D22030'

#Secondary Colors:

Cool\_Gray = '#55565A'

#Set Colors

y\_color = Black

x\_color = Black

graph\_color = Patone

Title\_Color = Patone

filepath = askopenfilename()

print(filepath)

X\_Axis = []

Y\_Axis = []

with open(filepath, 'r') as csvfile:

    plots = csv.reader(csvfile,delimiter =',',)

    for i, row in enumerate(plots):

        if i == 0:                                                          #Initialize Graph

            Header = row                                                    #Grab the Header Descriptions

            Total\_Points = len(Header) - 1                                  #Calcualte how many Y values there are

        else:

            x\_obj = int(row[0])

            X\_Axis.append(x\_obj)

            y\_obj = int(row[1])

            Y\_Axis.append(y\_obj)

step = math.ceil(len(X\_Axis)/x\_data\_points)

fig, ax1 = plt.subplots()

lines, labels = ax1.get\_legend\_handles\_labels()

ax1.legend(lines, labels, loc = 0 )

ax1.set\_xticklabels(X\_Axis[::step], rotation=45)

ax1.tick\_params(axis = 'y', labelcolor = y\_color)

ax1.tick\_params(axis = 'x', labelcolor = x\_color)

ax1.set\_xlabel(xlabel)

ax1.set\_ylabel(ylabel)

ax1.set\_title(Title, color=Title\_Color)

#t = plt.title(Title)

plt.step(X\_Axis, Y\_Axis, color=graph\_color)

fig.tight\_layout()

plt.grid(True)

plt.show()

**Figure A-2:** Graphing\_Script.py