

HIGH SPEED DATA ACQUISITION SYSTEM USING
RASPBERRY PI AND ADC UNIT

NAZRUL AZLI BIN RAZALI

BACHELOR OF ENGINEERING TECHNOLOGY
(ELECTRICAL)

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : NAZRUL AZLI BIN RAZALI
Date of Birth :
Title : HIGH SPEED DATA ACQUISITION SYSTEM USING
RASPERRY PI AND ADC UNIT
Academic Session : 2020/2021

I declare that this thesis is classified as:

- CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)*
- RESTRICTED (Contains restricted information as specified by the organization where research was done)*
- OPEN ACCESS I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

(Student's Signature)

(Supervisor's Signature)

Date: 8/2/2021

PM. DR. Mohd Mawardi Saari
Date:

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER (OPTIONAL)

Librarian,
Perpustakaan Universiti Malaysia Pahang,
Universiti Malaysia Pahang,
Lebuhraya Tun Razak,
26300, Gambang, Kuantan.

Dear Sir,

CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.

Author's Name NAZRUL AZLI BIN RAZALI
Thesis Title HIGH SPEED DATA ACQUISITION SYSTEM USING
RASPERRY PI AND ADC UNIT

Reasons (i)

(ii)

(iii)

Thank you.

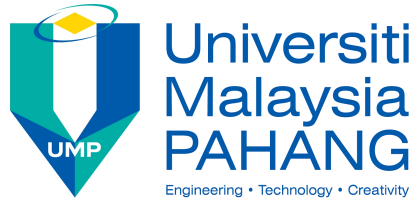
Yours faithfully,

(Supervisor's Signature)

Date:

Stamp:

Note: This letter should be written by the supervisor, addressed to the Librarian, *Perpustakaan Universiti Malaysia Pahang* with its copy attached to the thesis.



SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis and in our opinion, this thesis is adequate in terms of scope and quality for the award of degree of Bachelor of Engineering Technology in Electrical.

(Supervisor's Signature)

Full Name : PM. DR. MOHD MAWARDI SAARI

Position : LECTURER, FACULTY OF ENGINEERING TECHNOLOGY
ELECTRIC AND ELECTRONIC, UNIVERSITI MALAYSIA PAHANG

Date :



STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

(Student's Signature)

Name: NAZRUL AZLI BIN RAZALI

ID Number: TB17007

Date: 31 / 1 / 2021

HIGH SPEED DATA ACQUISITION SYSTEM USING RASPBERRY
PI WITH ADC UNIT

NAZRUL AZLI BIN RAZALI

Thesis submitted in fulfillment of the requirements
for the award of the degree of
Bachelor of Engineering Technology in Electrical

Faculty of Engineering Technology Electric and Electronic

UNIVERSITI MALAYSIA PAHANG

JANUARY 2021

ACKNOWLEDGEMENTS

I am truly grateful to ALLAH “S.W.T” for giving me wisdom, strength, patience and, assistance to complete our analysis project despite being tested in a pandemic environment. Had it not been due to His will and favor, the completion of this research would not have been achievable.

Without the instructions and the directions, this thesis would not have been necessary. The assistance of many persons in the planning and implementation of this analysis who contributed and extended their useful assistance. First of all, I would like to show my gratitude to my supervisor, PM. DR. MOHD MAWARDI SAARI for his guidance, suggestions and supports. I also like to give my gratitude to my project partner, AHMAD AIMAN SAFWAN BIN ZAKARIA who was working together to finish this project.

ABSTRAK

Sistem pemerolehan data (DAQ) adalah sejenis sistem yang mengukur informasi masa sebenar seperti suhu, tekanan, jarak dan lebih lagi setiap masa. Proses yang terlibat adalah membaca nilai analog daripada sensor, pembetulan dan pembaikan isyarat jika perlu, menukar isyarat analog kepada isyarat digital dan menganalisis data digital untuk proses selanjutnya. Nilai yang diambil daripada sistem DAQ dipanggil “sample” dan kelajuan untuk sistem ini melaksanakan seluruh operasi untuk satu “sample” boleh didefinisikan sebagai kadar kelajuan “sample” dan ini mempengaruhi resolusi bentuk gelombang yang dihasilkan. Kadang kala, data ini terlalu sukar untuk diproses menggunakan kaedah tradisional yang mungkin disebabkan oleh data terlalu besar, terlalu cepat atau terlalu kompleks untuk diproses. Kelajuan sistem DAQ sangat penting untuk aplikasi seperti kawalan kualiti bagi menentukan getaran yang dihasilkan, aplikasi sistem audio yang melibatkan pembatalan bunyi, radar dan aplikasi navigasi, dan banyak lagi. Kebelakangan ini, penggunaan produk Raspberry Pi untuk projek elektronik semakin meningkat. Ini disebabkan Raspberry Pi itu sendiri adalah komputer atau lebih tepat lagi komputer ARM yang dapat disambungkan ke rangkaian Internet. Raspberry Pi beroperasi dalam Sistem Operasi Linux yang boleh dikemas kini hampir setiap masa yang dikehendaki. Tetapi Raspberry Pi tidak boleh menerima isyarat analog sebagai isyarat data, sebab itulah ia memerlukan unit penukaran analog kepada digital (ADC) untuk proses penukaran. Dalam projek ini, kami akan mencipta osiloskop berasaskan Raspberry Pi dengan menggunakan Arduino Nano sebagai unit ADC dan hasil analisis projek ini ditentukan oleh frekuensi maksimum yang boleh dicapai oleh osiloskop ini.

ABSTRACT

A data acquisition system (DAQ) is a system that takes real-world measurements such as temperature, pressure, distance, and more in a function of time. The processes involved are reading the analog value from a sensor, signal conditioning if necessary, converting the analog signal into a digital signal, and analyzing the digitized signal for further processing. A value that is registered by the DAQ system is considered a sample, and the speed at which the DAQ system executes the entire operation for one sample can be defined by the sampling rate, and this sampling rate can influence the smoothness of the waveform created. Sometimes, these data are too difficult to be processed using traditional methods which are due to the data is too big, too fast, or too complex. The applications that highlight the speed of the DAQ system are critical are quality control, audio system applications involving noise cancellation, radar, navigation applications, and more. Lately, the usage of Raspberry Pi products for electronic projects is increasing. This is because Raspberry Pi itself is a computer or accurately an ARM computer that can be connected to the internet network. However Raspberry Pi cannot receive an analog signal as a data signal, that is why it needs an ADC unit for the conversion. In this project, we will be developing a Raspberry Pi-based oscilloscope using Arduino Nano as an ADC unit and the result of the analysis will be determined by how far the frequency which it can measure.

TABLE OF CONTENTS

	PAGE
DECLARATION TITLE	
ACKNOWLEDGEMENTS	I
ABSTRAK	II
ABSTRACT	II
TABLE OF CONTENTS	V
LIST OF TABLES	VI
LIST OF FIGURES	VII
LIST OF ABBREVIATIONS	VIII
LIST OF SYMBOLS	IX
LIST OF APPENDICES	X
CHAPTER 1 INTRODUCTION	1
1.1 PROJECT BACKGROUND	1
1.2 PROBLEM STATEMENT	2
1.3 OBJECTIVES	2
1.4 PROJECT SCOPE	3
CHAPTER 2 LITERATURE REVIEW	4
2.1 INTRODUCTION	4
2.2 RASPBERRY PI	4
2.2.1 RASPBERRY PI 3B	6
2.3 OSCILLOSCOPE	6
2.4 ARDUINO	7
2.4.1 ARDUINO NANO	8
2.5 PYGAME FRAMEWORK	9
2.6 TIME-STRETCHED ANALOGUE-TO-DIGITAL CONVERSION	10
2.7 CHOPPER SAMPLING	11
CHAPTER 3 METHODOLOGY	12
3.1 FLOWCHART	12
3.2 THEORY	13

3.3 PROJECT DESIGN	15
3.3.1 ARDUINO NANO ADC	15
3.3.2 RASPBERRY PI 3B DESIGN	15
CHAPTER 4 RESULTS & DISCUSSION	21
4.1 RESULTS & ANALYSIS	21
4.2 DAQ SYSTEM'S TECHNICAL SPECIFICATION	25
4.3 COST OF MATERIALS	26
CHAPTER 5 CONCLUSION & RECOMMENDATION	27
5.1 CONCLUSION	27
5.2 RECOMMENDATION	27
REFERENCES	29
APPENDICES	31

LIST OF TABLES

Table 1	: Prescale Register	9
Table 2	: Result of Peak Voltage Measurement	22
Table 3	: Result of Peak Voltage Error Test	23
Table 4	: Result of Frequencies Error	24
Table 5	: Waveform at Given Frequencies	25
Table 6	: DAQ System's Technical Specification	25
Table 7	: Cost Of Materials	26

LIST OF FIGURES

Figure 1	: Overview of Raspberry Pi Hardware	5
Figure 2	: Raspberry Pi LCD Display Unit	5
Figure 3	: Raspberry Pi 3B	6
Figure 4	: InfiniiVision 1000 X-Series	7
Figure 5	: Arduino Nano	8
Figure 6	: Block diagram for ADC system (Bhushan et al., 1998)	10
Figure 7	: Block diagram of the parallel ADC with offset compensation. 3 of 16 cells are drawn. The critical parts are implemented in the test chip and the rest is implemented in Matlab.(Eklund & Gustafsson, 2000)	11
Figure 8	: Methodology Flowchart	12
Figure 9	: System Flowchart	14
Figure 10	: ADC Arduino Nano Circuit	15
Figure 11	: Product Design (Front)	15
Figure 12	: Product Design (Back)	16
Figure 13	: Arduino Nano Top View	16
Figure 14	: Raspberry Pi Pinout	18
Figure 15	: Internal Product Design	18
Figure 16	: Graphical User Interface (GUI)	19
Figure 17	: Graphical User Interface (GUI) More Visible Trigger Line	20
Figure 18	: Connection Via BNC to Crocodile Cable with Oscilloscope Probes	21
Figure 19	: Connection Via BNC Coaxial Male to Male Cable	22

LIST OF SYMBOLS

SYMBOLS	MEANING
V	Voltage
V _{max}	Maximum Voltage
V _p	Peak Voltage
V _{pp}	Peak To Peak Voltage
kSa/s	Kilo-sample per second
KB	Kilo-Byte
kHz	Kilo-Hertz
mA	Milli-ampere
dB	Decibel
%	Percent
mm	Milli-meter
g	Gram
Ω	Ohm
μ F	Micro-Farad
°	Degree

LIST OF ABBREVIATIONS

ABBREVIATIONS	MEANING
DAQ	Data Acquisition
ADC	Analog To Digital Converter
IoT	Internet of Things
JPG	Joint Photographic Group
GUI	Graphical user interface
ARM	Advanced RISC Machines
CPU	Computer Processor Unit
LCD	Liquid Crystal Display
AC	Alternate Current
DC	Direct Current
USB	Universal Serial Bus
IDE	Integrated Development Environment
Max	Maximum
HDMI	High-Definition Multimedia Interface
BNC	Bayonet Neill-Concelman
TFT	Thin Film Transistor
RM	Ringgit Malaysia

LIST OF APPENDICES

Appendix A : ARDUINO NANO ARCHITECTURE & PROGRAMMING.....	31
Appendix B : RASPBERRY PI 3B ARCHITECTURE & PROGRAMMING.....	41
Appendix C : GANTT CHART SDP 1.....	47
Appendix D : GANTT CHART SDP 2.....	48

CHAPTER 1

INTRODUCTION

1.1 PROJECT BACKGROUND

The data acquisition (DAQ) system contains these key processes which acquire the analog signal, signal conditioning, transform the analog signal into digital so that it can be computerized. A research device typically used to display and interpret the waveform of electronic signals is called an oscilloscope whose purpose to draws a graph of the instantaneous voltage signal as a function of time. Unfortunately, this device can be very costly. In order DAQ system to work properly as an oscilloscope, it needs a fast sampling speed to display the high resolution output as the oscilloscope does. The waveform will become rough without it and the hidden glitches will never be detected.

During this period, the Internet of Things (IoT) became the main focus of the industry when reaching Industrial Revolution 4.0. So, the term of Big Data is also included since it is related to this project. Sometimes the data is too big or too slow or too complex to process. Hence, we use Raspberry Pi because this device is a computer in a small package at a small price. It even can be connected to an internet network to upload and download data.

Raspberry Pi nowadays at a lower price becomes more capable of processing information and much quicker than before. But unlike Arduino, Raspberry Pi cannot read analog signals except the most recent release Raspberry Pi Pico. That is why it needs an Analog to Digital Converter (ADC) to be able to accomplish analog reading and convert it into a digital signal in which Raspberry Pi can read and process the data. Raspberry Pi has an OS that can be installed in it, and it can install many useful programs that process the input data and display it in a continuous graph per time. Additional Python programming is required to add a few additional functions of an oscilloscope.

1.2 PROBLEM STATEMENT

Commercial oscilloscopes are rather costly and most commercial oscilloscope features, such as high sampling rate or the number of channels that differ by their costs, are not needed by certain beginner electronic hobbyists. Some of them just want an oscilloscope for minimal usage like to look for how the condition of the output signal look for repairing and analyzing noise and other problem of an audio system.

To turn a data acquisition system (DAQ) into an oscilloscope, the DAQ system's process itself needs to be fast enough to sample data at high speed. The whole process of taking the input sample, converting the analog signal into a digital signal, and processing the information at high speed to get high resolution output. Some of the available DAQ systems are too slow and have visible noise to be used as an oscilloscope. The idea is to ensure that the entire DAQ system is running at a high speed to achieve high-resolution output. You can determine the speed of the DAQ system by the number of samples read per second or the sampling rate.

Another concern is that some of the commercial oscilloscopes require big working space and room for storage. Plus, they also required multiple usages of power plugs for operating an electronics experiment. So the idea for using Raspberry Pi and ADC unit as a DAQ system comes to mind since Raspberry Pi nowadays is used in many projects due to their capabilities are almost rival modern days computer. Even if a cheaper oscilloscope is available, some electronic enthusiasts might already have Raspberry Pi for their previous project and they can salvage those to make an oscilloscope. This shows that the availability of Raspberry Pi is very high among people.

1.3 OBJECTIVES

This project has objectives as below:

1. To design a DAQ system circuit with high speed and high resolution output.
2. To program Raspberry Pi to be able sample data at high speed.
3. To evaluate the performance of Raspberry Pi as a high speed DAQ system with ADC unit.

1.4 PROJECT SCOPE

This project is all about how fast the Raspberry Pi DAQ system is and how well it can be implemented as an oscilloscope. The Raspberry Pi or in this case Raspberry Pi 3 cannot sample an analog signal. That is why we use Arduino Nano as an ADC unit. Therefore, the scope of this project will involve designing a DAQ circuit using Raspberry Pi and ADC unit, programming Raspberry Pi to process the data, and analyzing the performance of the DAQ system.

1. Designing A DAQ Circuit Using Raspberry Pi and ADC Unit

The circuit of DAQ system will consist of analog input which is a probe that is connected to BNC adapter, an ADC unit which is Arduino Nano and Raspberry Pi 3 to process the digitized signal to be displayed in signal waveform. The Arduino Nano is connected to Raspberry Pi 3 USB port.

2. Programming Raspberry Pi to Process the Data

Python is the programming language used to program codes in Raspberry Pi which run on Linux. This ARM computer will be mainly tasked to read the digital signal, determine the parameters value such as V_{max} , V_{pp} , frequency etc., plotting a waveform from the samples collected and save them in a JPG file.

3. Analyzing the performance of DAQ system

The parameter for performance analyzing is the sampling rate which is how much samples can the DAQ system process in a period of time. The main goal for the sampling rate is to surpass 50kSa/s and 25kHz maximum frequency for the oscilloscope.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter will explain the fundamental concept and theory of the Raspberry Pi and ADC unit to achieve high speed data acquisition. The overall parameters will be discussed as well as each of its contributions to an oscilloscope performance adding with the technique used in the preparation and designation of the oscilloscope.

2.2 RASPBERRY PI

The Raspberry Pi is the creation of the Raspberry Pi Foundation, a charity conceived at Cambridge University Computer Laboratory to create an inexpensive, practical, self-contained machine that school students can use to learn programming in an interesting environment, separated from school IT systems where such activities are not enabled. Based on a Broadcom-made ARM microcontroller, it was originally designed to be used with a very powerful GPU in set-top boxes, while incorporating a very weak CPU. Besides the cost, means that it accidentally popular among hobbyists who use it in similar ways has become wildly common.

With this project, the Raspberry Pi's price is especially attractive. The Raspberry Pi, like the Arduino, provides low-level access to the device interfaces. Combined with standard PC interfaces, it offers a valid platform for the development of a custom interface that can be conveniently, cheaply, and efficiently incorporated into the GPIO header of the Raspberry Pi. Figure 1 displays the Raspberry Pi with the various interfaces marked in diagram form.

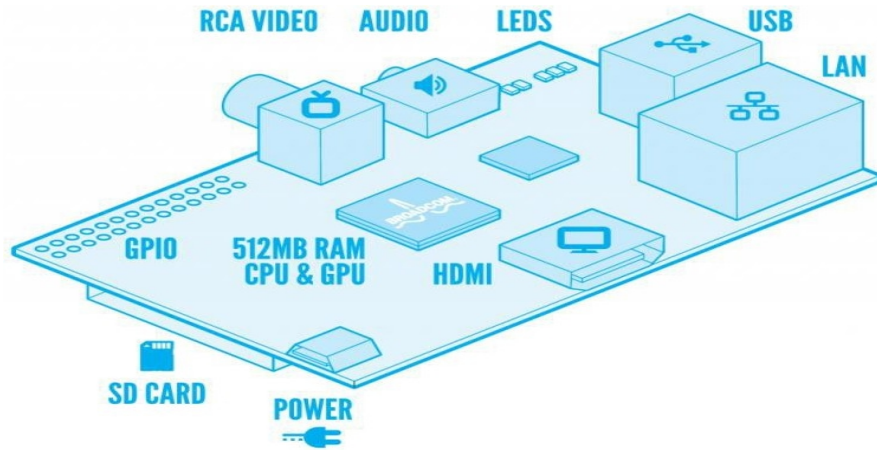


Figure 1: Overview of Raspberry Pi Hardware

Raspberry Pi, which is very suitable for a computer because it has both analog video out and HDMI display ports. This is possible even without the need for external computer units such as Arduino to display its function in real-time. (Kakade & Lokhane, 2016) Raspberry Pi oscilloscope project uses the desktop to display their results while (Prof.V.P.Patil Sir, Shivani Pagare, Kajol Pagare & Utkarsha Patil, 2017) use Android device for display connected via Bluetooth. Having more than one device as an oscilloscope system is a very inconvenience in our opinion. Plus, it uses two different power sources for each device. A small LCD unit like Figure 2 below would be appropriate since it receives power from Raspberry Pi itself.



Figure 2: Raspberry Pi LCD Display Unit

Source: <https://my-live-01.slatic.net/p/3499be76cabe4191d7ba5de0da90c554.jpg>

2.2.1 RASPBERRY PI 3B

The principal component of the system is quite similar to the microcontroller. The Raspberry Pi USB port will be the input of the digital data after conversion of the analog input signal into digital. The architecture is for USB connectivity built-in Python. There is a graphical user interface (GUI) and the data will appear on the LCD screen which will be powered by Raspberry Pi GPIO pins and the signal is sent through an HDMI connection.



Figure 3: Raspberry Pi 3B

2.3 OSCILLOSCOPE

An oscilloscope is a device that reads a variety of voltage signals such as AC voltage, DC voltage, pulse, and more and it will display them in a two-dimensional plot as a function of time. The oscilloscope can read a signal or two in microseconds even in nanoseconds and display them in different frequencies. (Asadi et al., 2016) The whole oscilloscope device is a high speed data acquisition system that involves ADC. A virtual oscilloscope system which was designed by Shulin Shi (1974) is made up of signal gathering, signal processing and demonstrated finally three major parts, the signal gathering part is realized by the hardware, the signal processing and demonstrated finally two parts both are realized by the software. These parts fulfill the DAQ system requirements.

The speed of an oscilloscope can be determined by its max sample rates and bandwidth which varies according to the prices which can be very expensive. So, in order to make an oscilloscope, a DAQ system sample rate must be high enough to plot signal as a function of time at an acceptable resolution. A commercial oscilloscope

nowadays has a sample rate range from 1GSa/s to 128GSa/s and beyond. A study done by Seema Kakade & Dr. Mrs. S S Lokhane(2016) shows that the system they created gives ac and dc voltage and frequency measurement with good resolution and the range of voltage is $\pm 20V$ AC and range of frequency is up to 1MHz using Bit scope micro, Raspberry Pi 2 processor, desktop and the programming of the algorithm and GUI is done in python.

The sampling rate is twice the maximum frequency which means the sampling rate of their system is 2MHz. However, to create a smooth waveform, it requires at least 10 samples which makes the maximum frequency is 200kHz. A commercial 1GSa/s oscilloscope like InfiniiVision 1000 X-Series costs RM 2,363 (price may vary according to the third-party supplier). The oscilloscope we will try to make cannot look into the price point of view alone. Other factors tie with the cost such as the functions provided, number of channels, and more.



Figure 4: InfiniiVision 1000 X-Series

2.4 ARDUINO

Focused on easy-to-use hardware and software, Arduino is an open-source electronics platform. It's meant for someone who creates creative projects. By sending a series of instructions to the microcontroller on the board, you can tell the board what to do. To do so you will need to use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing. Arduino has been used in thousands of different projects and applications, thanks to its simplicity and open user interface. For beginners, the Arduino app is simple-to-use but versatile enough for experienced users. The other factors are, it is an inexpensive, cross-platform, simple, clear programming environment, open-source and

extensible software and hardware.

2.4.1 ARDUINO NANO

A lightweight, full, and breadboard-friendly board based on the ATmega3288 is the Arduino Nano (Arduino Nano 3.x). With its specs of ATmega328 microcontroller, 16MHz clock speed, 2KB SRAM, flash memory of 32 KB of which 2 KB used by bootloader and power consumption of 19 mA. According to an article done by (Willem Maes, May 1, 2018), the Arduino board only as fast as its clock (disregarding multi-core processors), which the Arduino Uno defaults to using a 16MHz crystal. Arduino Nano also has a clock speed of 16MHz but a different microcontroller which is ATmega328 series while Arduino Uno uses ATmega328p. However, there is no major difference between those two microcontrollers except power consumption.

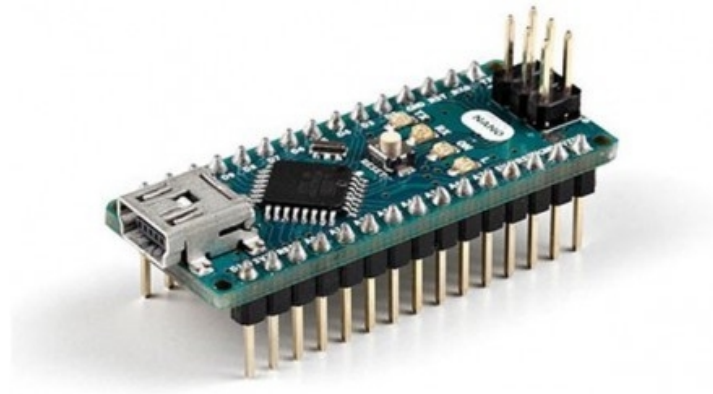


Figure 5: Arduino Nano

Also from research done by (Willem Maes, May 1, 2018), the Arduino ADC or `analogRead()` function, the default sampling rate is 9600Hz in theory but when he runs a simple code, the sampling rate is 8928Hz. To make it go faster, his idea is to change the ADC Prescaler factor.

Table 1: Prescale Register

Prescale factor	ADPS2	ADPS1	ADPS0	Clock Frequency (MHz)	Sampling Rate (kHz)
2	0	0	1	8	615
4	0	1	0	4	307
8	0	1	1	2	153
16	1	0	0	1	76.8
32	1	0	1	0.5	38.4
64	1	1	0	0.25	19.2
128	1	1	1	0.125	9.6

2.5 PYGAME FRAMEWORK

Pygame is a series of Python modules designed for video game writing. On top of the excellent SDL library, other features can also be added. This enables you to create completely featured python-language multimedia programs besides games. Pygame can be considered as a game engine and a game engine is a framework that handles graphic rendering for display, audio output, motion, and animation, generate a graphical user interface (GUI), and more.

Since, an oscilloscope requires animation to create a live continuous waveform, (Mike Baker, 2018) had implemented Pygame for his Raspberry Pi oscilloscope project to generate GUI and animation. The GUI buttons for each function he created are simple and because of them, the hardware uses fewer mechanical parts and costs can be reduced like mechanical buttons.

2.6 TIME-STRETCHED ANALOGUE-TO-DIGITAL CONVERSION

A new concept for analog-to-digital (ADC) conversion is proposed and demonstrated. The analog signal is stretched in time before sampling and quantization. (Bhushan, Coppinger, & Jalali, 1998). Time stretching improves the ADC's input bandwidth and sampling rate and is best used with optoelectronic techniques. Digital signal processing (DSP) constantly proliferates in High-performance systems stresses the importance of Advances in the application of analog-to-digital converters (ADC). The digital receiver is one example of systems whose needs far outweigh the efficiency of current Help converters (ADCs). The AID conversion is carried out at IF or RF frequencies in such systems, placing stringent requirements on the AID sampling frequency and input bandwidth. Although the performance of electronic A/D continues to increase, the pace of progress is too slow to satisfy the demands of advanced systems shortly.

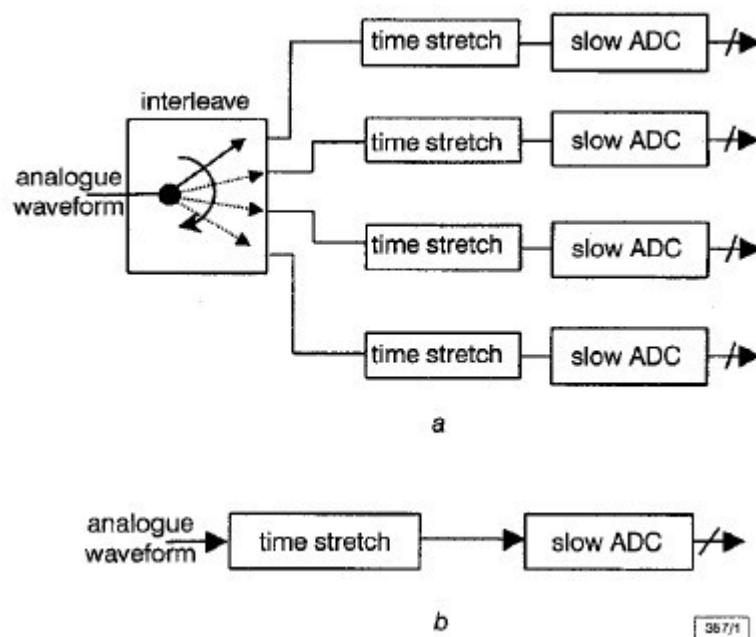


Figure 6: Block diagram for ADC system (Bhushan et al., 1998)

Figure 6 shows a block diagram of the proposed system. The analog signal is segmented into parallel channels and interleaved into M . Before entering a slow ADC each segment is time-stretch with an M magnification factor.

2.7 CHOPPER SAMPLING

An ADC that uses many parallel cells suffers from offset variations between the cells, which creating non-harmonic distortion. A method is proposed to eliminate the Digital Domain Offset. The method is based on a PRBS-controlled at the ADC input chopper which converts any input signal into noise. The randomization controls the batch size needed by a mean value calculation function to eliminate the offset. The measured results are SFDR = 72dB and SNDR = 59.0dB @ 22MS/s, an improvement of 19dB and 10dB respectively (Eklund & Gustafsson, 2000).

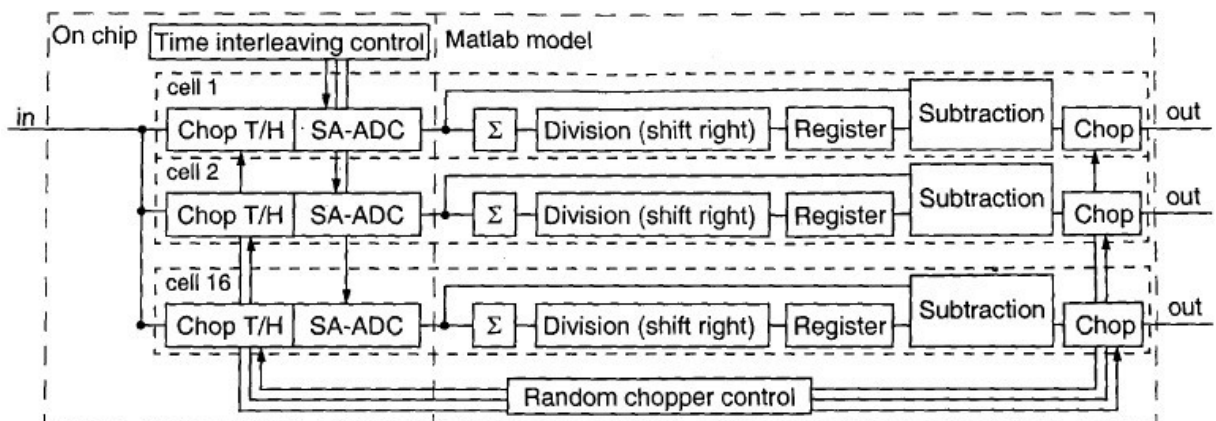


Figure 7: Block diagram of the parallel ADC with offset compensation. 3 of 16 cells are drawn. The critical parts are implemented in the test chip and the rest is implemented in Matlab. (Eklund & Gustafsson, 2000)

The implementation of the proposed method is shown in figure 7. The ADC output values are corrected by the offset value in the register. The offsets, calculated from one lot of values, are used to adjust the next stack. When the offset is eliminated, the output signal is chopped with the same series as the input and the value of the signal is restored. That batch measures a new calculation and hence the offset values are changed during service.

CHAPTER 3

METHODOLOGY

3.1 FLOWCHART

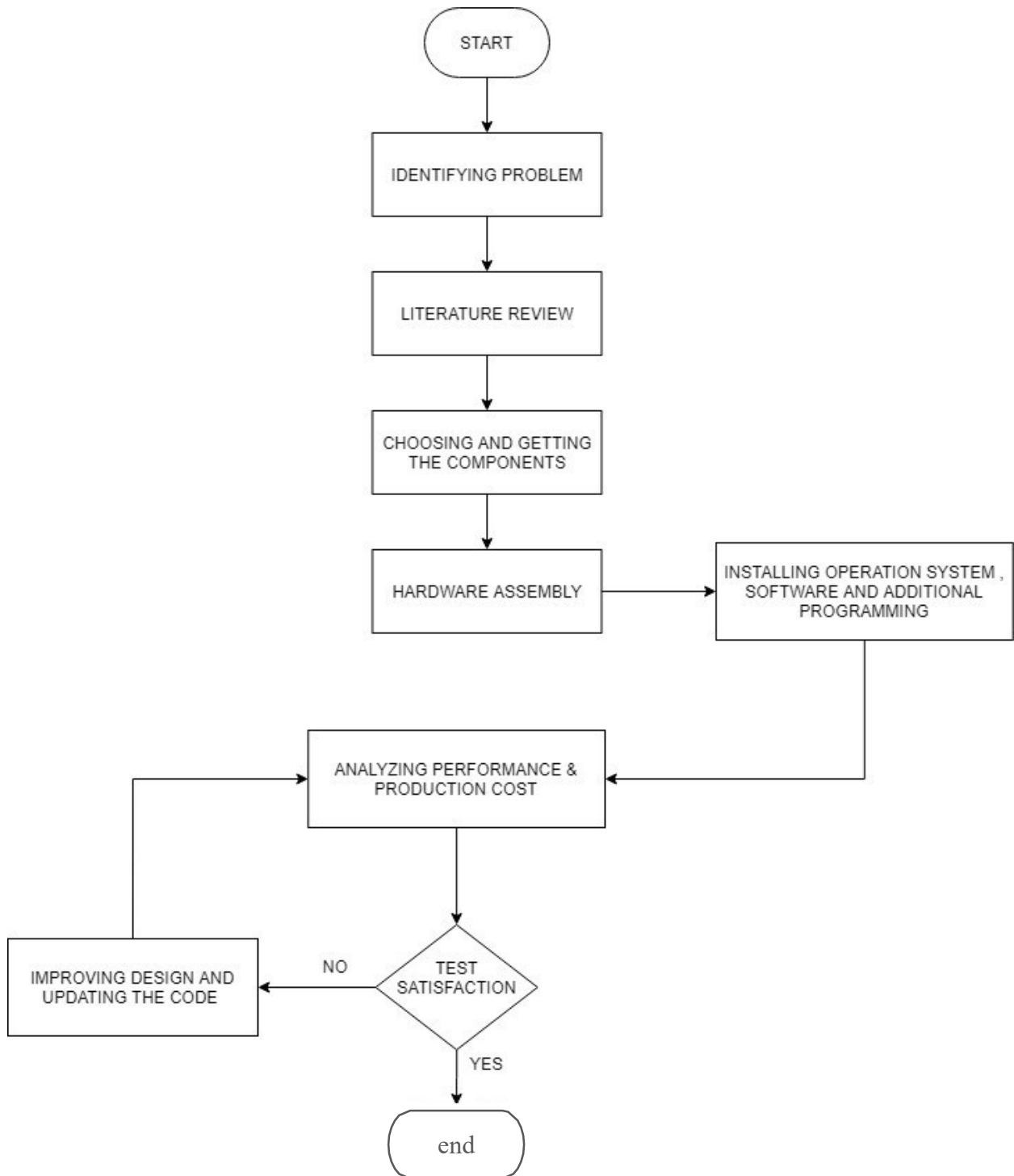


Figure 8: Methodology Flowchart

3.2 THEORY

Firstly, we need to decide how Raspberry Pi and ADC unit which is Arduino Nano communicate. Since most of the Raspberry Pi's GPIO pins have been used to power up and to receive inputs from the touch screen of the LCD, the other option is to use Universal Serial Bus (USB) to exchange information between them.

Next, the serial communication speed or baud rate of Arduino Nano must be determined to make sure that the speed for sending and receiving data is suitable for our project. We set it to 115200 through coding in Arduino IDE which is the limit to how fast data can be transferred for the most microcontroller. The ADC clock speed or ADC prescale factor of the Arduino Nano also must be decided to the ones that Raspberry Pi can handle for our project.

After that, the Raspberry Pi will execute the oscilloscope program which was written in Python 3, and let the Python framework handle all the graphical operations for display. Soon, it will need to use the information received from the Arduino Nano in digital form and process them to provide output. We call this DAQ system ArPi Scope which stands for Arduino Raspberry Pi oscilloscope. The flowchart below is how the whole system works.

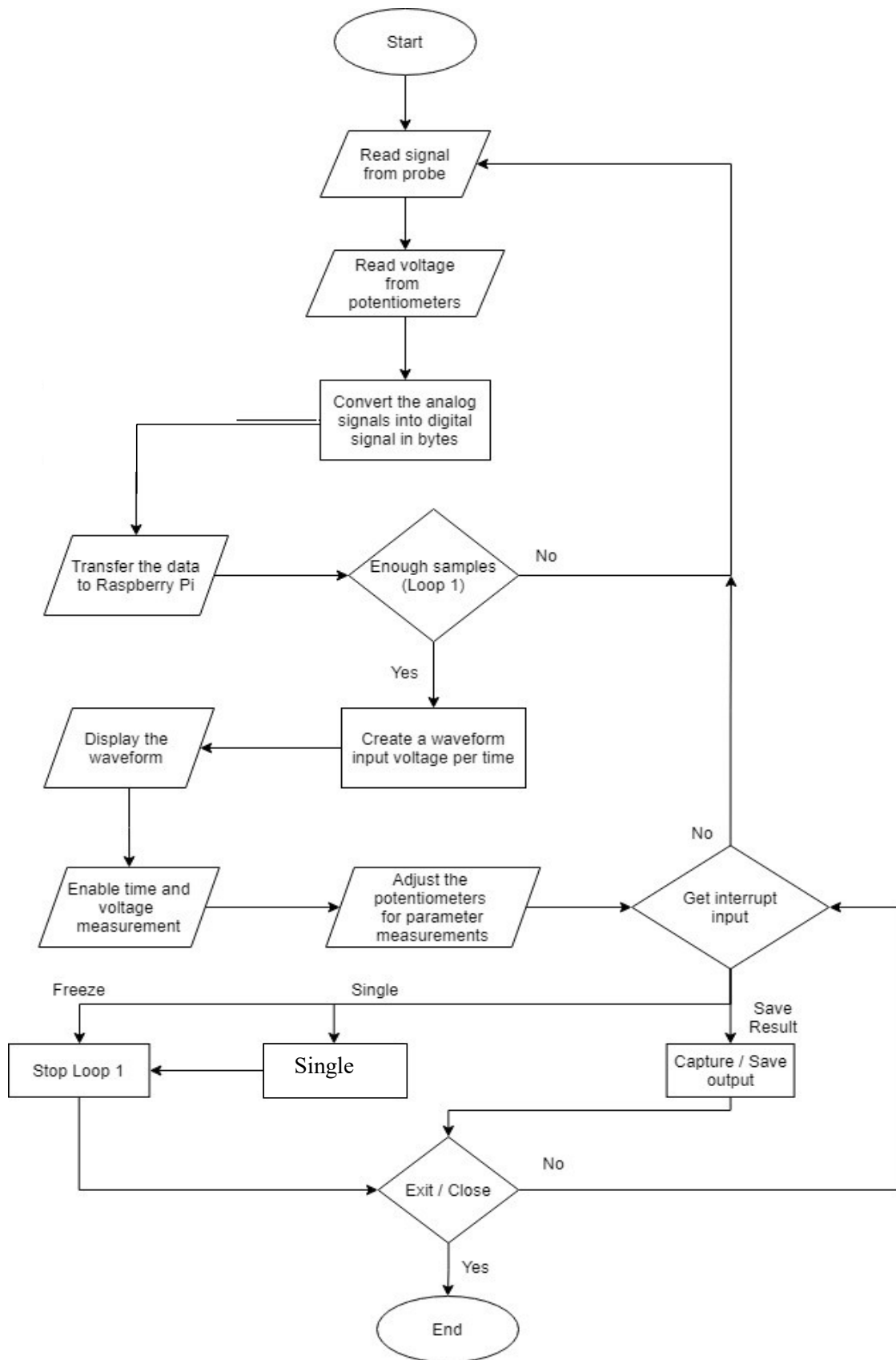


Figure 9: System Flowchart

3.3 PROJECT DESIGN

The circuit below contains capacitors ensures that no DC components from the input get through and provide a little bit of protection for overvoltage. As for the resistors, they will limit the current flow. To control the oscilloscope and its AC coupled biased voltage inputs, potentiometers were used. The input from the BNC probe is connected to pin A0 while pin A3, A4, and A5 were connected to the potentiometers.

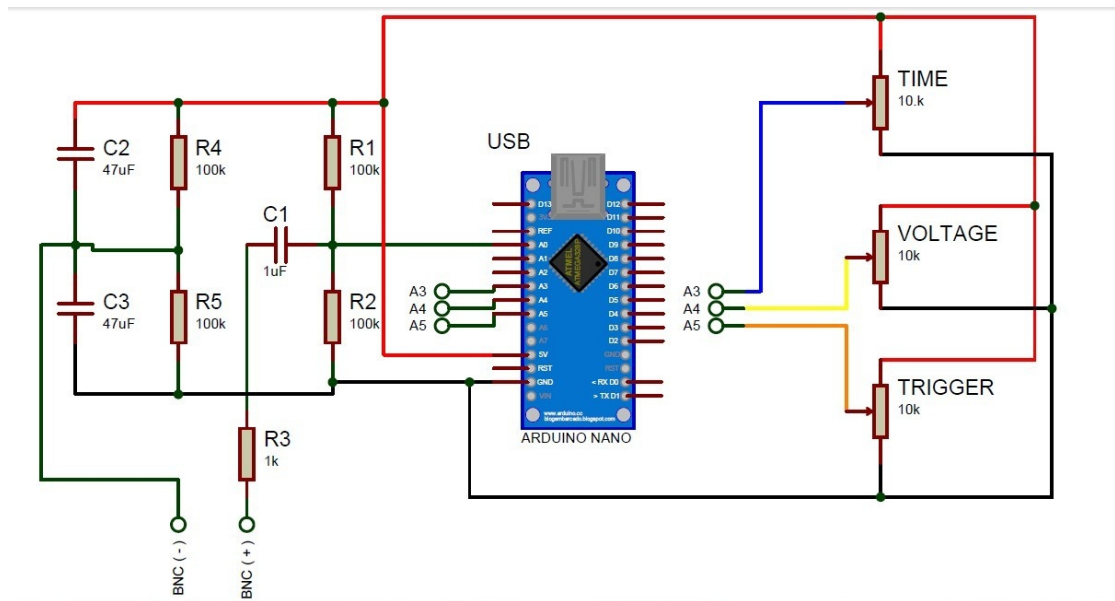


Figure 10: ADC Arduino Nano Circuit



Figure 11: Product Design (Front)

The blue, yellow, and red potentiometers control time, voltage, and trigger respectively. GPIO pins from 1 until 26 were used for powering up the LCD screen and its touchscreen input. The video signal from Raspberry Pi 3 to the LCD screen is transferred via HDMI cable. The mini USB cable is used for communication between Raspberry Pi 3 with Arduino Nano. Because of these, we only need to power up Raspberry Pi 3 with a 5V, 2.5A power supply to power up the whole system.



Figure 12: Product Design (Back)

3.3.1 ARDUINO NANO



Figure 13: Arduino Nano Top View

The programming for the Arduino is done in Arduino IDE in C++ language. To set up the Arduino Nano to be able to sample at a rate 307kHz, first, we need to set the ADC Prescaler factor to 4 just like in Table 5. The clock speed of the clock crystal inside Arduino Nano which is 16MHz will be divided by a prescale factor, (p.f) to get ADC clock speed like the

equation below.

$$\text{ADC clock} = \frac{16\text{MHz}}{p.f}$$
$$\text{ADC clock} = \frac{16\text{MHz}}{4} = 4\text{MHz}$$

Then, this ADC clock will be divided by 13 since a conversion takes 13 ADC clocks. So the default maximum sampling rate is as shown in the equation below.

$$\text{Maximum sampling rate} = \frac{\text{ADC clock}}{13}$$
$$\text{Maximum sampling rate} = \frac{4\text{MHz}}{13} = 307\text{kHz}$$

However, a smooth waveform requires at least 10 points so the Max sample rate will be divided by 10 for maximum frequency to get a smooth waveform.

$$\text{Maximum frequency} = \frac{\text{Maximum sampling rate}}{10}$$
$$\text{Maximum frequency} = \frac{307\text{kHz}}{10} = 30.7\text{kHz}$$

To program it, we have to define these two functions first.

```
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))  
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
```

To be simplified, they will allow us to use commands below to register bit to 0 or 1. "cbi" command will clear bit or register it as 0 while the "sbi" command will register bit to 1. Next, we can register the desired bits to get our desired Prescaler which is 4.

```
cbi (ADCSRA, ADPS2); // ADPS2 = 0  
sbi (ADCSRA, ADPS1); // ADPS1 = 1  
cbi (ADCSRA, ADPS0); // ADPS0 = 0
```

It is not recommended to set Prescaler below 16 however, we have to because we want to achieve the maximum sampling rate possible for our project. The method to collect sample, conversion, and oscilloscope's control input is like the flowchart in the appendix.

3.3.2 RASPBERRY PI 3

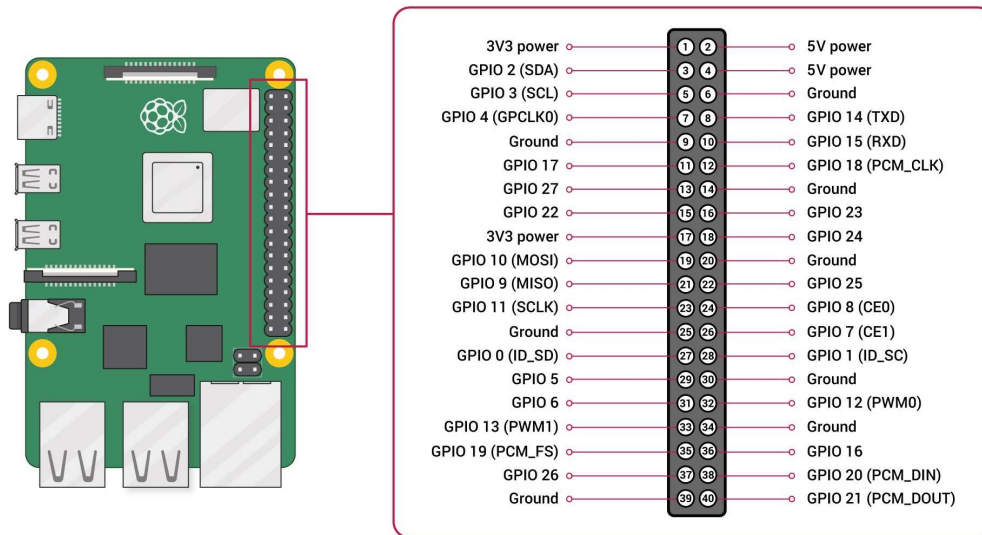


Figure 14 : Raspberry Pi Pinout

Since most of the GPIO pins had being used for powering up LCD and its touchscreen inputs, we have to improvise the buttons for the oscilloscope's function. The physical buttons were swapped into virtual buttons in the GUI.



Figure 15 : Internal Product Design

The programming is done in Python 3 and the framework and animations will be handled by Pygame Framework. It also helps to generate GUI with the buttons.

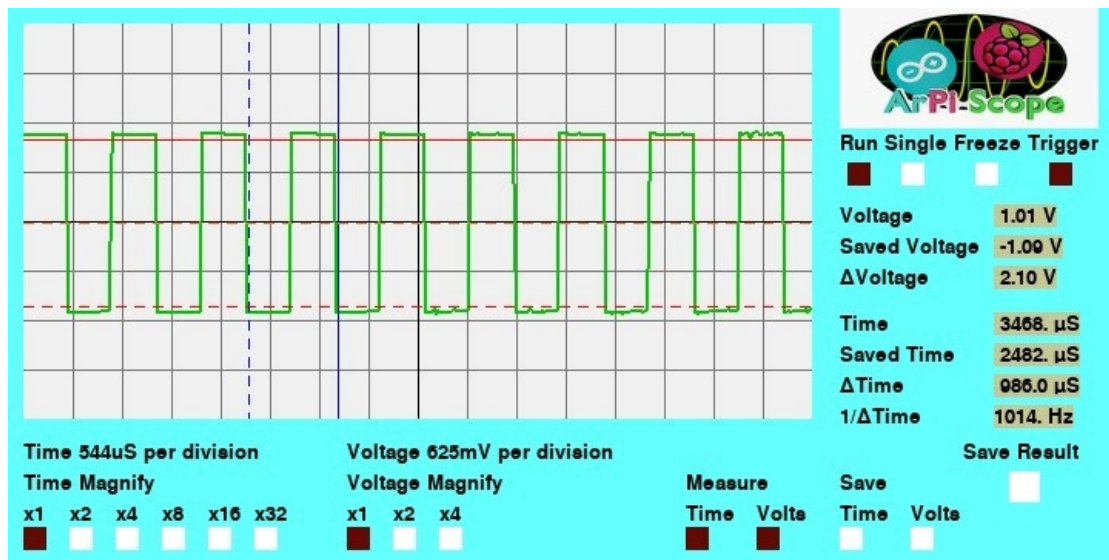


Figure 16 : Graphical User Interface (GUI)

Let's start with the Time Magnify buttons. Each of these buttons will zoom in horizontally depending the scale factor. If we push the button, the magnify variable which in binary will shift 1 to the left in which scale factor you press.

```

250     for n in range(0,6) :
251         if LedRect[n].collidepoint(pos):
252             expandT = 1<<n

```

The same case with the Voltage Magnify buttons but it will zoom in vertically. Moving on to Measure and Save buttons which both have Time and Volts buttons. If the Measure's Time button is press, the solid vertical blue line will show up like in Figure and it can be controlled via a blue potentiometer. If the Save's Time button is press, it will mark the current position of a solid vertical blue line with the dotted vertical blue line. The same case with Volt buttons but it is a horizontal red line and can be controlled via a yellow potentiometer. The orange dotted line is the trigger and it can be controlled via a red potentiometer.

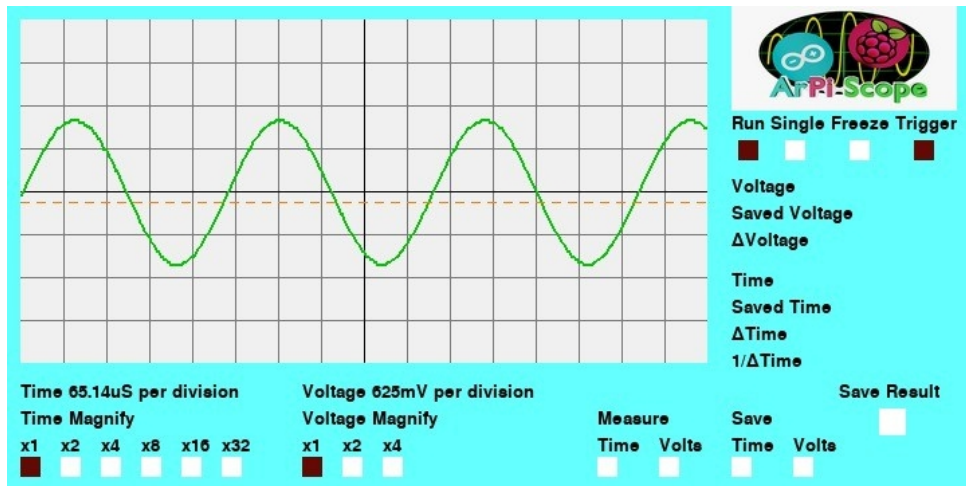


Figure 17 : Graphical User Interface (GUI) More Visible Trigger Line

By pressing those Measure buttons, the parameters will show up on the right. The Run, Single, Freeze buttons, it work similarly like other commercial oscilloscopes. The final button is the Save Result button which function as a screenshot button.

```

301
302     if LedRect[17].collidepoint(pos): # screenshot
303         pygame.image.save(screen, "image.png")
304

```

CHAPTER 4

RESULTS & DISCUSSION

4.1 RESULTS & ANALYSIS

There are a few experiments done for analyzing this system's performance. To begin with, the Raspberry Pi 3 of Python 3 program can't handle the Arduino Nano's prescale of 2 with 8MHz ADC clock speed. The first experiment is to measure the peak voltage at different frequencies. Note that, the experiment is done with a probe that switched to an X1 attenuator which allows us to measure up to +/- 2.5V. Depending on the probe, this DAQ system capable to measure up to +/-25V with an X10 probe's attenuator.

In these experiments we had done, we connect the signal generator with ArPi Scope and a PC oscilloscope, PicoScope through BNC to crocodile cable, and oscilloscope probes with X1 attenuator. Sometimes, we connect the ArPi Scope with PicoScope directly through BNC coaxial cable since PicoScope can be a signal generator too. The trigger is set to 0°.

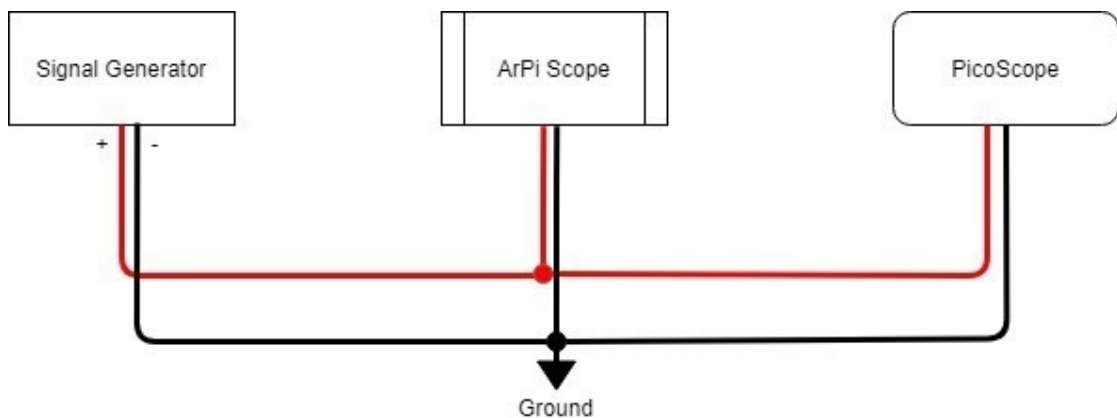


Figure 18 : Connection Via BNC to Crocodile Cable with Oscilloscope Probes

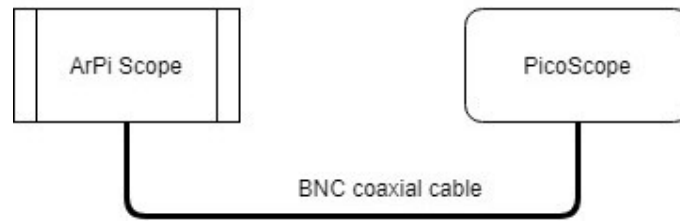


Figure 19 : Connection Via BNC Coaxial Male to Male Cable

1. A sine wave signal from the function generator with various voltages in different frequencies being measured for input references.

Table 2 : Result of Peak Voltage Measurement

No.	Voltage (V)	Input	Peak Voltage (V _p) at 100Hz	Peak Voltage (V _p) at 1kHz	Peak Voltage (V _p) at 10kHz
1	0.01		0	0	0
2	0.03		0	0.05	0.03
3	0.05		0.07	0.07	0.07
4	0.07		0.11	0.1	0.09
5	0.10		0.13	0.14	0.11
6	0.30		0.33	0.32	0.33
7	0.50		0.54	0.53	0.50
8	0.70		0.76	0.73	0.70
9	1.00		1.07	1.04	1.04

According to the result above, the errors are less than ± 0.10 V and its maximum error is ± 0.07 V. There are a few factors that might cause an error during measurements which is the human error and the display error which is the number of pixels that the display unit supported and the number of pixels used for generating GUI.

2. This part of experiment is for capturing the voltage peak with the different frequency input.

Amplitude = 1 V

Table 3 : Result of Peak Voltage Error Test

No.	Frequency Input	Peak Voltage (Vp)	Error (%) [$\frac{V_p - 1V}{1V} \times 100$]
1	1 Hz	0	0
2	3 Hz	0	0
3	5 Hz	0	0
4	7 Hz	0	0
5	10 Hz	0.97	-3
6	30 Hz	1.07	7
7	50 Hz	1.07	7
8	70 Hz	1.05	5
9	100 Hz	1.05	5
10	300 Hz	1.05	5
11	500 Hz	1.03	3
12	700 Hz	1.03	3
13	1000 Hz	1.02	2
14	10000 Hz	1.02	2
15	30000 Hz	1.03	3
16	50000 Hz	1.05	5
17	100000 Hz	1.05	5

According to the results above, we couldn't measure the Vp from 1Hz to 7Hz because the oscilloscope can't generate full cycle or at least 1/4 cycle so that we can get the peak of a waveform. The waveform is too wide to be measured. As for 10Hz, we started to be able to see the peak only, so we can measure the peak voltage however, we couldn't measure the frequency for next experiment. The average error is +3.77%.

3. This part of experiment is for measuring the difference between the input frequency with the measured frequency at same amplitude.

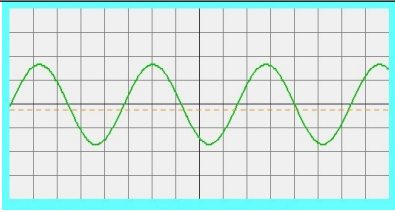
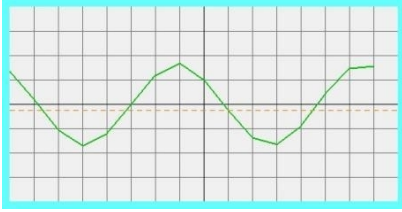
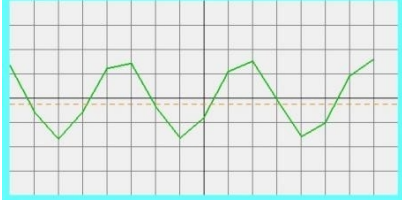
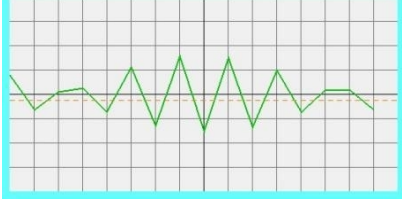
Amplitude = 1 V

Table 4 : Result of Frequencies Error

No.	Frequency Input	Frequency Output	Error (%) [$\frac{\text{Output} - \text{Input}}{\text{Input}} \times 100$]
1	80 Hz	79.45 Hz	-0.69
2	100 Hz	99.9 Hz	-0.1
3	300 Hz	302.1 Hz	0.7
4	500 Hz	504.9 Hz	0.98
5	700 Hz	691.5 Hz	-1.21
6	1000 Hz	1016 Hz	1.6
7	3000 Hz	3010 Hz	0.33
8	5000 Hz	4992 Hz	-0.16
9	7000 Hz	7058 Hz	0.83
10	10000 Hz	9904 Hz	-0.96
11	30000 Hz	29954 Hz	-0.15
12	50000 Hz	52260 Hz	4.52
13	100000 Hz	52260 Hz	-47.74

Since the oscilloscope can't generate a full cycle from 1Hz, we must start measuring at 80Hz since the waveform is narrow enough to generate more than 1/4 cycle. To generate at least one full cycle of the waveform, the frequency must be close to 300Hz. According to the result above, the error surpasses 2% starting at frequency 50kHz which is above the maximum frequency which is 30kHz theoretically. So the average error we calculate until 30kHz is +0.11%. In addition to the accuracy, the waveform also started to get rough once it reaches 30kHz.

Table 5 : Waveform at Given Frequencies

Oscilloscope's Waveform	Frequency
	1 kHz
	26 kHz
	30 kHz
	70 kHz

So, according to the table above, the actual optimal maximum frequency is somewhere around 26kHz to 28kHz and the sampling rate that Raspberry Pi process is around 260kHz.

4.2 DAQ SYSTEM'S TECHNICAL SPECIFICATION

Table 6 : DAQ System's Technical Specification

Sampling rate	260kHz @ 260kSa/s	
Maximum frequency	26kHz	
Maximum voltage	X1 Probe's Attenuator	X10 Probe's Attenuator
	2.5V	25V
Average Voltage error	+3.77%	
Average Frequency error	+0.11%	

4.3 COST OF MATERIALS

Table 7 : Cost Of Materials

Material	Price (RM)
Raspberry Pi 3 Model B	155.00
5V 2.5A AC/DC Adapter Micro B USB cable (UK plug)	20.00
32 GB Micro SD Card	40.00
Arduino Nano (CH340)	15.00
Resistor 100k Ω (4 pieces)	0.44
Resistor 1k Ω	0.10
Capacitor 47 μ F (2 pieces)	1.10
Capacitor 1 μ F	0.45
16 Pin Female Header	1.54
Carbon Type Potentiometer (3 pieces)	1.95
Potentiometer Plastic Knob (3 pieces)	0.90
Donut Board Single Side 5*7cm	0.98
BNC Female Coax to Screw Adapter	4.25
40 Ways Male to Female Jumper Wire (20cm)	2.50
10 mm PCB Stand Screw and Nut (12 pieces)	6.00
25 mm PCB Stand Screw and Nut (4 pieces)	4.80
HDMI 5 inch TFT LCD Touch Screen	186.80
Male to Female 0.3m HDMI cable	9.90
8*6*2 inch Clear Plastic Box	8.00
TOTAL	459.71

CHAPTER 5

CONCLUSION &

RECOMMENDATION

5.1 CONCLUSION

The major point we want to see from this project is how fast Raspberry Pi DAQ system process the whole tasks. The ADC unit sampling rate is different from Raspberry Pi because the method of transferring the data through serial USB limits the speed. Also, the speed of sampling will decrease for each line of code available in the program. So far, the wired transferring medium is still faster than the wireless one so the wired IoT Big Data is still superior to wireless in terms of speed.

The oscilloscope maximum frequency sacrifices the minimum frequency it can measure if it uses a lower prescale factor. The lower prescale factor also slows down the Raspberry Pi performance for this particular program and the recommended prescale factor is 16 and above. The reason behind why we could not measure lower frequency is because the minimum magnifies factor is 1 in binary. To magnify, the bit of the binary will be shifted to the left to 2, 4, 8, 16, 32, and so on. However, it cannot be shifted to the right beyond less than 1.

For improvements, we need more time, healthier environment conditions, hard work, and more knowledge about Python language to explore it even deeper than what is capable of. Nevertheless, the main objectives were achieved. It becomes clearer once we swap the ADC unit from ADS1256 chip into Arduino Nano, we were able to implement our knowledge we learned in the Engineering Technology Electrical program. It becomes clearer for us because Arduino uses an ATmega microcontroller which we have basic knowledge of in C++ language and the tough part for this project is understanding the Python language.

5.2 RECOMMENDATION

Our recommendation for future improvement is to use the latest version of Raspberry Pi 3B like Raspberry Pi Model B+ which has faster CPU clock speed, uses the latest processor, and better heat reduction since it already has a built-in heatsink on the processor. We also noticed that when running this oscilloscope program, the Raspberry Pi generates a lot of heat, so any approach to reduce the heat also helps to improve its performance. As for GUI, we recommend using the bigger button for ease of usage since we were using a touch pen which more accurate than a finger.

We also recommend using superior Raspberry Pi than Raspberry Pi 3 like Raspberry Pi 4. Raspberry Pi 4 is built like an actual personal computer complete with high RAM and more. As for the ADC unit, we recommend the brand new Raspberry Pi Pico microcontroller which has 3 ADC capable pins and released on 21st January 2021. Please note that this is the first Raspberry Pi microcontroller unit which is similar to Arduino Nano but better in terms of architecture. Maybe in the future, this Raspberry Pi product will be sold in form of a kit that contains both Raspberry Pi computers with Raspberry Pi microcontroller which may reduce the cost of materials.

REFERENCES

- Asadi, A. A., Bagheri, S., Imam, A., Jalayeri, E., Kinsner, W., & Sepehri, N. (2016, 13-15 Oct. 2016). *A data acquisition system based on Raspberry Pi: Design, construction and evaluation*. Paper presented at the 2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON).
- Bhushan, A. S., Coppinger, F., & Jalali, B. (1998). Time-stretched analogue-to-digital conversion. *Electronics Letters*, *34*(9), 839-841. doi:10.1049/el:19980629
- Candes, E. J., & Wakin, M. B. (2008). An Introduction To Compressive Sampling. *IEEE Signal Processing Magazine*, *25*(2), 21-30. doi:10.1109/MSP.2007.914731
- Eklund, J., & Gustafsson, F. (2000, 28-31 May 2000). *Digital offset compensation of time-interleaved ADC using random chopper sampling*. Paper presented at the 2000 IEEE International Symposium on Circuits and Systems (ISCAS).
- Kakade, S., & Lokhane, D. M. S. S. (2016). Oscilloscope Using Raspberry Pi Processor.
- Laska, J., Kirolos, S., Massoud, Y., Baraniuk, R., Gilbert, A., Iwen, M., & Strauss, M. (2006, 29-30 Oct. 2006). *Random Sampling for Analog-to-Information Conversion of Wideband Signals*. Paper presented at the 2006 IEEE Dallas/CAS Workshop on Design, Applications, Integration and Software.
- Patil, P., & Bhole, K. (2018, 15-17 Feb. 2018). *Real time ECG on internet using Raspberry Pi*. Paper presented at the 2018 International Conference on Communication, Computing and Internet of Things (IC3IoT).
- Breidenbach, M., Frank, E., Hall, J., & Nelson, D. (1978). Semi-Autonomous Controller for Data Acquisition the Brilliant ADC. *IEEE Transactions on Nuclear Science*, *25*(1), 706-710. doi:10.1109/TNS.1978.4329397
- Shi, S. L., & Pi, G. R. (2011). Design of Virtual Oscilloscope Based on S3C2410. *Advanced Materials Research*, *189-193*, 227-230. doi:10.4028/www.scientific.net/AMR.189-193.227
- Carley, L. (1987). An oversampling analog-to-digital converter topology for high-resolution signal acquisition systems. *IEEE Transactions on Circuits and Systems*, *34*(1), 83-90. doi:10.1109/TCS.1987.1086039
- Schoukens, J. (1995). A critical note on histogram testing of data acquisition channels. *IEEE Transactions on Instrumentation and Measurement*, *44*(4), 860-863. doi:10.1109/19.392871
- Soon, Y. L., Gan, K. B., & Abdullah, M. (2015, 10-12 Aug. 2015). *Development of very low frequency (VLF) data acquisition system using Raspberry Pi*. Paper

presented at the 2015 International Conference on Space Science and Communication (IconSpace).

Odunlade, E. (2018). Raspberry Pi Based Oscilloscope. Retrieved from <https://circuitdigest.com/microcontroller-projects/raspberry-pi-based-oscilloscope>

Sharma, D., Samuel, K., Ramoutar, K., Lowe, T., & David, I. (2017). Raspberry Pi Based Real Time Data Acquisition Node for Environmental Data Collection. *Journal of Basic and Applied Engineering Research*, 4, 307-312.

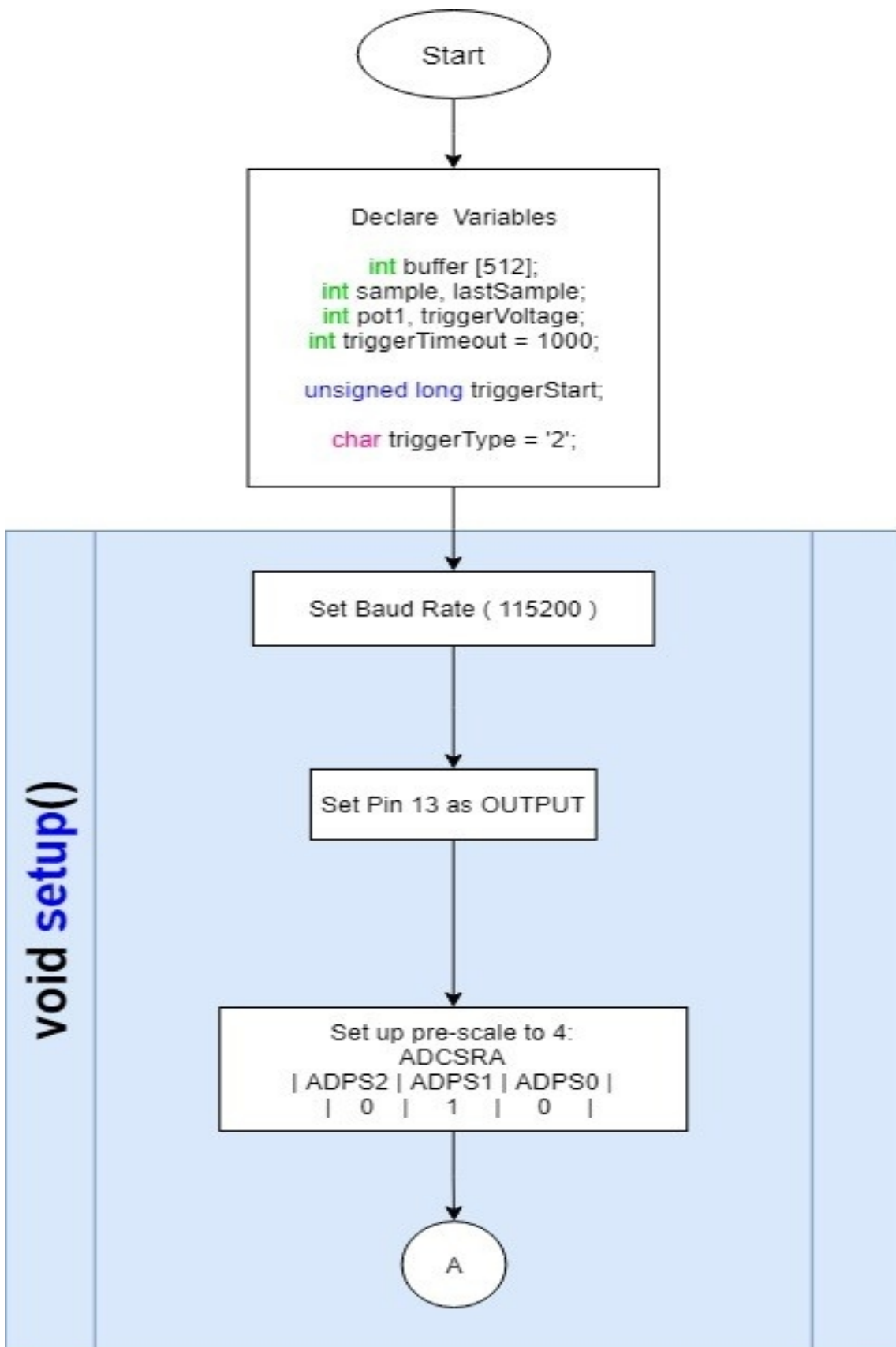
Mike Baker (July 2018). BUILD AN OSCILLOSCOPE. The MagPi, Issue 71, Page 44-51. <https://magpi.raspberrypi.org/issues/71>

Willem Maes (May 1, 2018). How to Make an Arduino Fast Enough to...

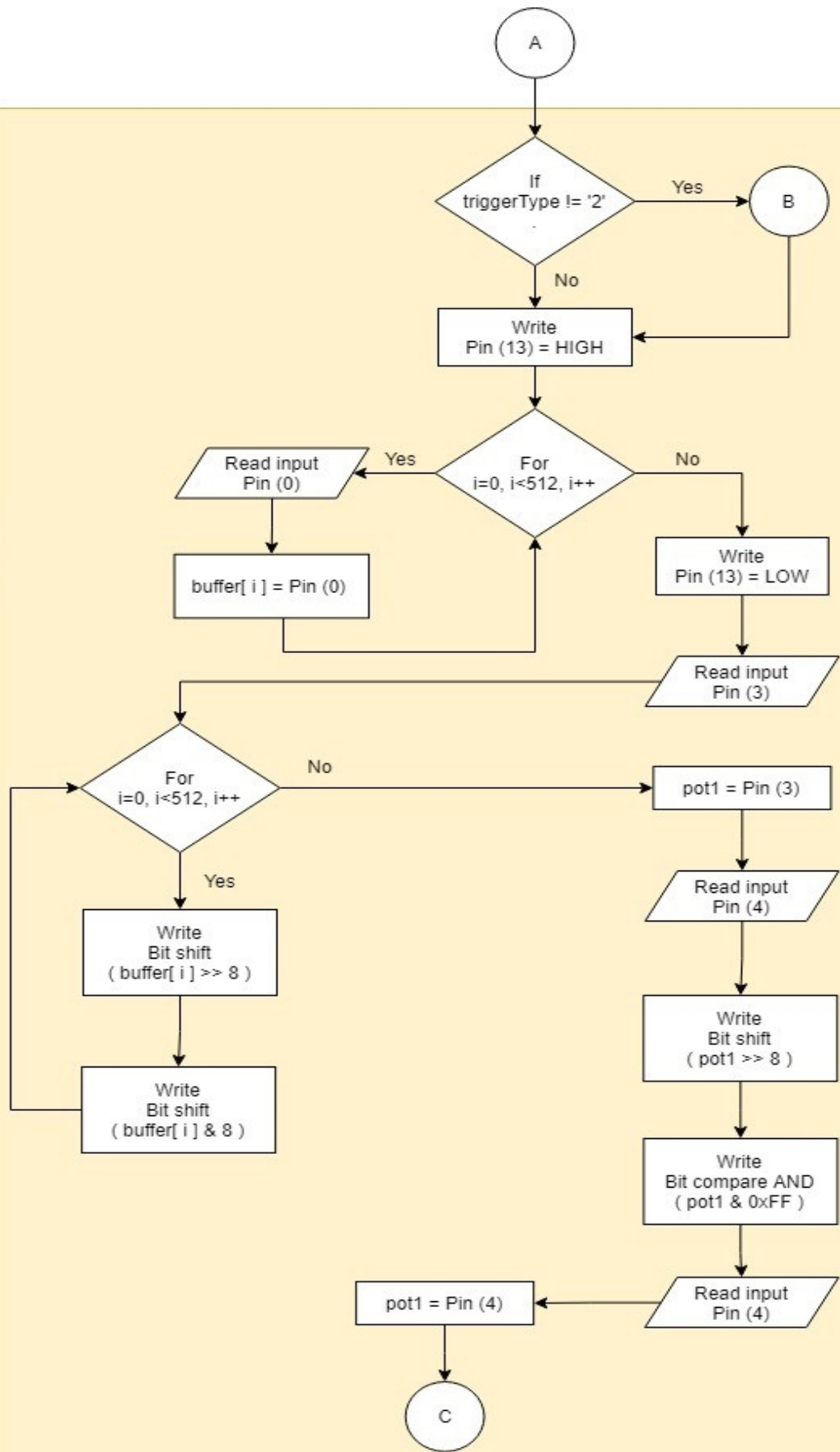
APPENDIX A

ARDUINO NANO ARCHITECTURE & PROGRAMMING

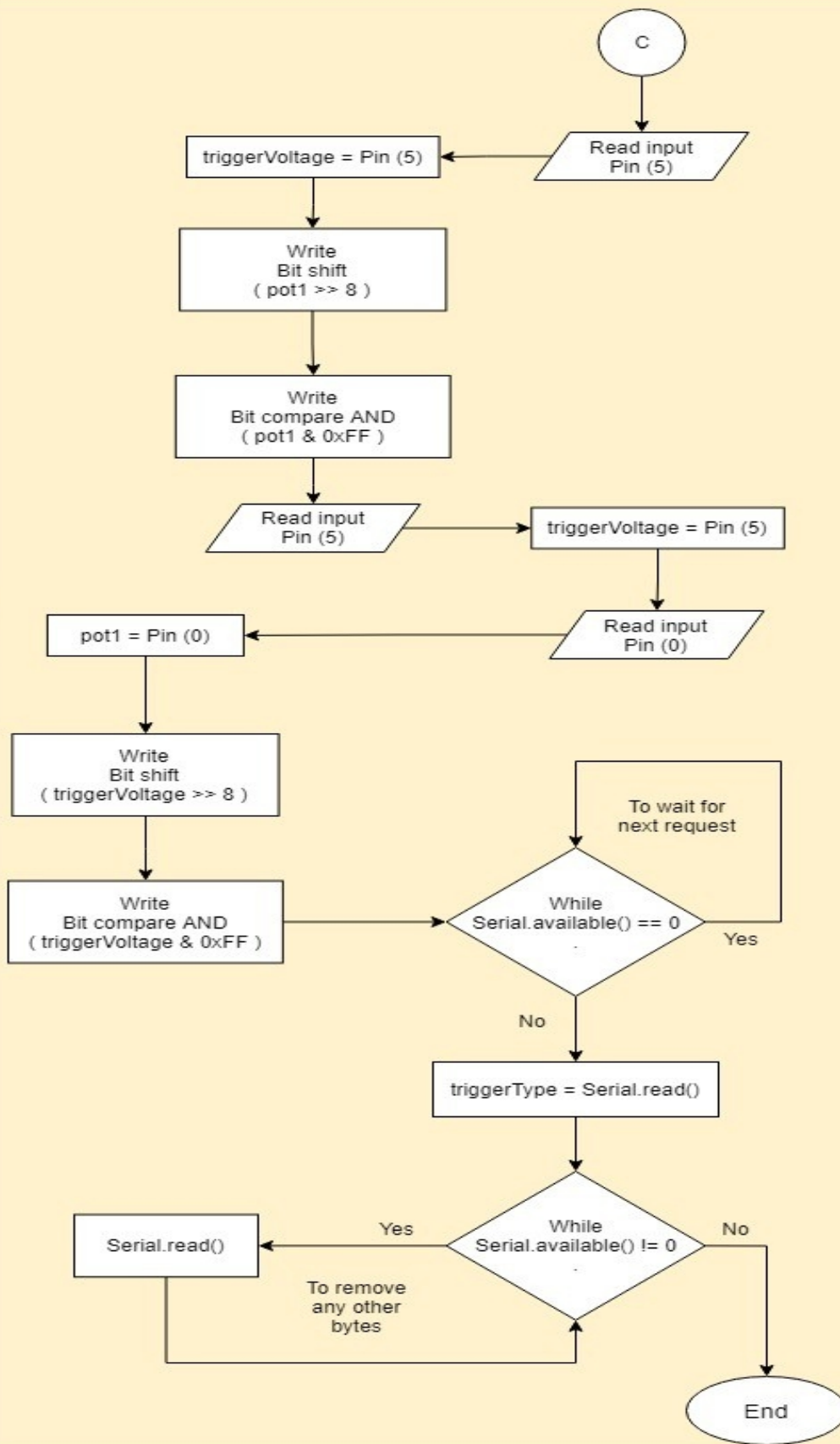
Microcontroller	ATmega328
Architecture	AVR
Operating Voltage	5 V
Flash Memory	32 KB of which 2 KB used by bootloader
SRAM	2 KB
Clock Speed	16 MHz
Analog IN Pins	8
EEPROM	1 KB
DC Current per I/O Pins	40 mA (I/O Pins)
Input Voltage	7-12 V
Digital I/O Pins	22 (6 of which are PWM)
PWM Output	6
Power Consumption	19 mA
PCB Size	18 x 45 mm
Weight	7 g
Product Code	A000005



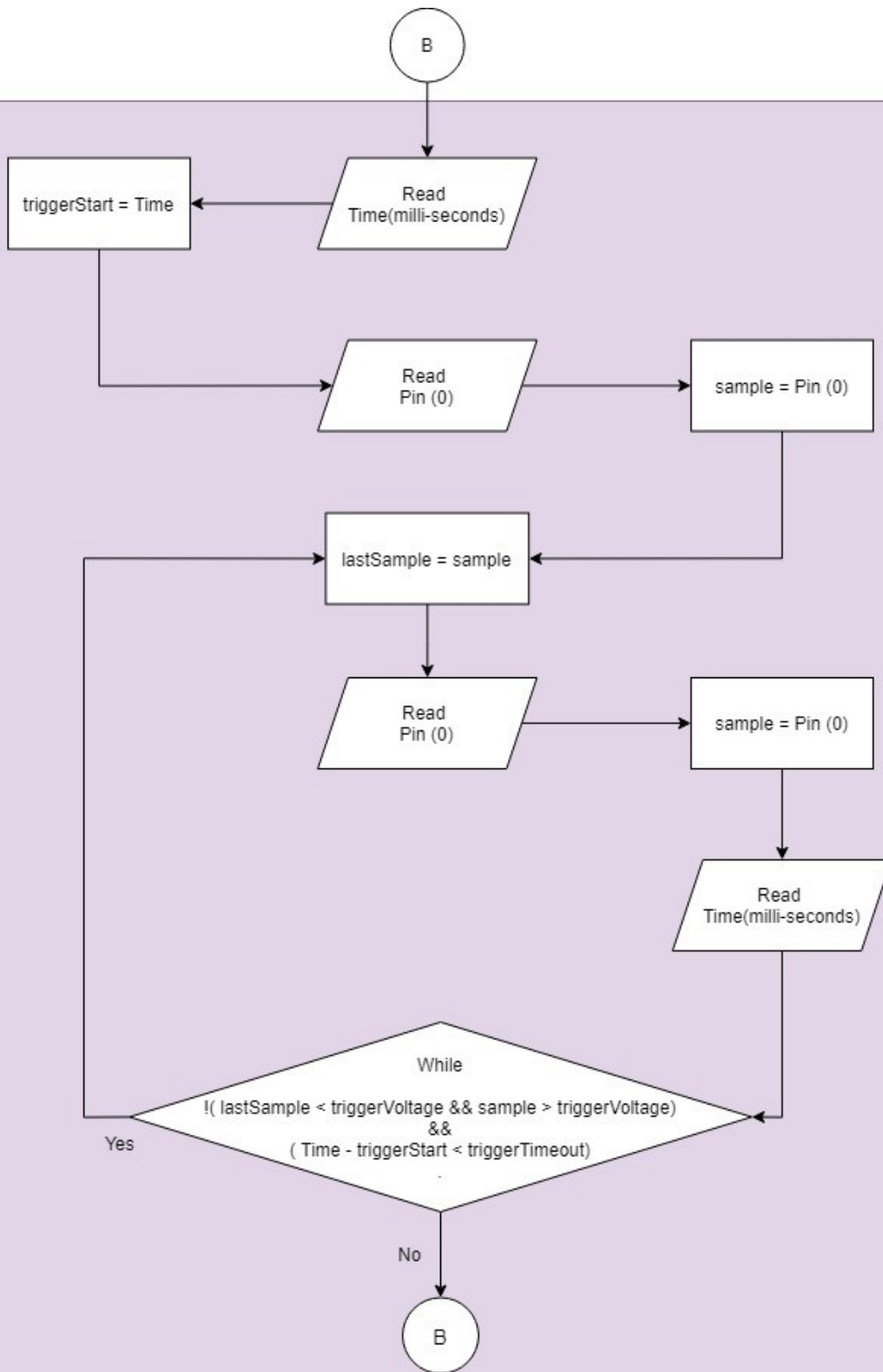
void loop()



void loop()



void trigger()





```

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
/* macro to clear bit in special function register
 *
 * The cbi() is a macro to set the bit(the second argument) of the address(the first argument) to 0.
 *
 * From the command line above it sets the bit-th bit of the content of sfr to 0.
 */
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
/* macro to set bit special function register
 *
 * The sbi() is a macro to set the bit(the second argument) of the address(the first argument) to 1.
 *
 * From the command line above it sets the bit-th bit of the content of sfr to 1.
 */
int buffer [512]; // 1K input buffer
int sample, lastSample;
int Analog_In, triggerVoltage;
int triggerTimeout = 1000; // time until auto trigger
unsigned long triggerStart;
char triggerType = '2';

void loop(){
  if( triggerType != '2') trigger(); // get a trigger
  digitalWrite(13,HIGH); // timing marker
  for(int i=0; i<512 ; i++){
    buffer[i] = analogRead(0);
  }
  digitalWrite(13,LOW); // timing marker
  Analog_In = analogRead(3); // switch channel to cursor pot
  for(int i=0; i<512 ; i++){
    Serial.write(buffer[i]>>8);
    Serial.write(buffer[i] & 0xff);
  }
  // send back pot values for cursors
  Analog_In = analogRead(3);
  analogRead(4); // next cursor pot
  Serial.write(Analog_In>>8);
  Serial.write(Analog_In & 0xff);
  Analog_In = analogRead(4);
  triggerVoltage = analogRead(5);
  Serial.write(Analog_In>>8);
  Serial.write(Analog_In & 0xff);
  triggerVoltage = analogRead(5);
  Analog_In = analogRead(0); // prepair for next sample run

```

```
Serial.write(triggerVoltage>>8);
Serial.write(triggerVoltage & 0xff);

while(Serial.available() == 0) { } // wait for next request
triggerType = Serial.read(); // see what trigger to use
while (Serial.available() != 0) { // remove any other bytes in buffer
  Serial.read();
}
}

void trigger(){
  // trigger at rising zero crossing
  triggerStart = millis();
  sample = analogRead(0);
  do {
    lastSample = sample;
    sample = analogRead(0);
  }
  while(!(lastSample < triggerVoltage && sample > triggerVoltage) && (millis() - triggerStart < triggerTimeout));
}
```

Done printing.

Sketch uses 2216 bytes (7%) of program storage space. Maximum is 30720 bytes.

39

Features

- High performance, low power AVR[®] 8-bit microcontroller
- Advanced RISC architecture
 - 131 powerful instructions – most single clock cycle execution
 - 32 × 8 general purpose working registers
 - Fully static operation
 - Up to 16MIPS throughput at 16MHz
 - On-chip 2-cycle multiplier
- High endurance non-volatile memory segments
 - 32K bytes of in-system self-programmable flash program memory
 - 1Kbytes EEPROM
 - 2Kbytes internal SRAM
 - Write/erase cycles: 10,000 flash/100,000 EEPROM
 - Optional boot code section with independent lock bits
 - In-system programming by on-chip boot program
 - True read-while-write operation
 - Programming lock for software security
- Peripheral features
 - Two 8-bit Timer/Counters with separate prescaler and compare mode
 - One 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
 - Real time counter with separate oscillator
 - Six PWM channels
 - 8-channel 10-bit ADC in TQFP and QFN/MLF package
 - Temperature measurement
 - Programmable serial USART
 - Master/slave SPI serial interface
 - Byte-oriented 2-wire serial interface (Philips I²C compatible)
 - Programmable watchdog timer with separate on-chip oscillator
 - On-chip analog comparator
 - Interrupt and wake-up on pin change
- Special microcontroller features
 - Power-on reset and programmable brown-out detection
 - Internal calibrated oscillator
 - External and internal interrupt sources
 - Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby, and extended standby

Table 23-4. Input Channel Selections

MUX3_0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8 ⁽¹⁾
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V _{AG})
1111	0V (GND)

Note: 1. For temperature sensor.

23.9.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In single conversion mode, write this bit to one to start each conversion. In free running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, auto triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC trigger select bits, ADTS in ADCSRB.

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the data registers are updated. The ADC conversion complete interrupt is executed if the ADIE bit and the i-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a read-modify-write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC conversion complete interrupt is activated.

- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

Table 23-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

23.9.3 ADCL and ADCH – The ADC Data Register

23.9.3.1 ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

23.9.3.2 ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When an ADC conversion is complete, the result is found in these two registers.

When ADCL is read, the ADC data register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

- **ADC8:0: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in Section 23.7 "ADC Conversion Result" on page 215.

APPENDIX B

RASPBERRY PI 3B ARCHITECTURE & PROGRAMMING

Processor	Broadcom BCM2837 64bit ARM Cortex-A53 Quad Core Processor SoC running
CPU Clock Speed	1.2GHz
RAM	1 GB DDR2
USB Ports	4 x USB2.0 Ports with up to 1.2A output
Storage	MicroSD
Power Source	5V, 2A
Ethernet	10/100 Ethernet
Wireless Connectivity	BCM43143 (802.11 b/g/n Wireless LAN and Bluetooth 4.1)
Operating Temperature	-40°C to +85°C
Operating System	LINUX

```

1 import serial, pygame, os, time
2
3 pygame.init()
4 os.environ['SDL_VIDEO_WINDOW_POS'] = 'center'
5 pygame.display.set_caption("Raspberry Pi Oscilloscope")
6 pygame.event.set_allowed(None)
7 pygame.event.set_allowed([pygame.KEYDOWN, pygame.MOUSEBUTTONDOWN, pygame.QUIT, pygame.MOUSEBUTTONUP])
8
9 textHeight=20 ; font = pygame.font.Font(None, textHeight)
10 screenWidth = 720 ; screenHeight = 360
11 screen = pygame.display.set_mode([screenWidth,screenHeight],0,32)
12 display = pygame.Surface((512,256))
13 backCol = (100,255,255) ; black = (0,0,0) # background colours
14 pramCol = (200,200,150) # parameter colour
15 logo = pygame.image.load("images/SDPimage.png").convert_alpha()
16
17 sampleInput = serial.Serial("/dev/ttyUSB0",115200, timeout = 5)
18 # /dev/ttyUSB0 may differ if the Arduino USB is plugged in different RPi USB port
19
20 displayWidth = 512 ; displayHeight = 256
21 ledRect = [ pygame.Rect((0,0),(0,0))] * 18
22 inBuf = [0]*512 # quick way of getting a 512 long buffer
23 chOff = displayHeight//2 # Channel Offset
24 run = [True,False,False,True,False] # run controls
25 expandT = 1 ; expandV = 1 # voltage & time expansion
26
27 sampleTime = 6.514 # uS for 307kHz sample = 3.257uS
28 samples_cm = 10 * sampleTime
29 volts_sample = 5/1024 # volts per sample
30 measureTime = False ; measureVolts = False;savedTime = 0;savedVoltage = 0
31 cursorT = 0 ; cursorV = 0 ; vMag = 1 ; svLed = False ; stLed = False
32 triggerC = 512 ; savedVoltsC = -1 ; savedTimeC = -1 ; button = 0
33
34 |
35
36 def main():
37     pygame.draw.rect(screen,backCol,(0,0,screenWidth,screenHeight+2),0)
38     defineControls()
39     drawControls()
40     time.sleep(0.1)
41     sampleInput.flushInput() # empty any buffer contents
42     sampleInput.write(b'2') # tell Arduino to get a new buffer
43     while(1):
44         time.sleep(0.001) # let other code have a look in
45         readArduino() # get buffer data
46         plotWave() # draw waveform
47         if measureTime or measureVolts :
48             updateControls(True)
49             drawScope() # display new screen
50             checkForEvent()
51             while run[4]: # if in hold mode wait here
52                 checkForEvent()
53             if run[3]:
54                 sampleInput.write(b'1') # tell Arduino to get an other buffers
55             else:
56                 sampleInput.write(b'2') # buffer but no trigger
57
58 def screenShot():
59     loop = 1
60     while loop:
61         # checks every user interaction in this list
62         for event in pygame.event.get():
63             if event.type == pygame.QUIT:
64                 loop = 0
65             if event.type == pygame.KEYDOWN:
66                 if event.key == pygame.K_s:
67                     pygame.image.save(screen, "image.png")
68

```

```

69         #draw(event)
70         #pygame.display.flip()
71         pygame.quit()
72
73
74
75
76 def drawGrid():
77     pygame.draw.rect(display, (240,240,240), (0,0,displayWidth,displayHight),0)
78     for h in range(32,256,32): # draw horizontal
79         pygame.draw.line(display, (120,120,120), (0,h), (512,h),1)
80     for v in range(32,512,32): # draw vertical
81         pygame.draw.line(display, (120,120,120), (v,0), (v,256),1)
82     pygame.draw.line(display, (0,0,0), (256,0), (256,256),1)
83     pygame.draw.line(display, (0,0,0), (0,128), (512,128),1)
84
85 def drawControls():
86     drawWords("Time Magnify",10,300,black,backCol)
87     drawWords("Voltage Magnify",220,300,black,backCol)
88     drawWords("Measure",440,300,black,backCol)
89     drawWords("Time",440,320,black,backCol)
90     drawWords("Volts",486,320,black,backCol)
91     drawWords("Save",540,300,black,backCol)
92     drawWords("Time",540,320,black,backCol)
93     drawWords("Volts",586,320,black,backCol)
94     drawWords("1/"+chr(0x394)+"Time",540,257,black,backCol)
95     drawWords(chr(0x394)+"Time",540,237,black,backCol)
96     drawWords("Saved Time",540,217,black,backCol)
97     drawWords("Time",540,197,black,backCol)
98     drawWords(chr(0x394)+"Voltage",540,167,black,backCol)
99     drawWords("Saved Voltage",540,147,black,backCol)
100    drawWords("Voltage",540,127,black,backCol)
101    drawWords("Run Single Freeze Trigger",540,80,black,backCol)
102    drawWords("Save Result",620,280,black,backCol)]
103
104    screen.blit(logo,(540,0))
105    updateControls(True)
106
107 def updateControls(blank):
108     global vDisp
109     if blank:
110         pygame.draw.rect(screen,backCol,resultsRect,0)
111         if expandT*smples_cm >= 1000:
112             drawWords("Time "+str((expandT*smples_cm)//1000)+"mS per division ",10,280,black,backCol)
113         else:
114             drawWords("Time "+str(expandT*smples_cm)+"uS per division ",10,280,black,backCol)
115         volts_cm = int(volts_sample*128*1000/expandV)
116         drawWords("Voltage "+str(volts_cm)+"mV per division",220,280,black,backCol)
117         for n in range(0,6): # time option LED
118             drawWords("x"+str(1<<n),10+n*30,320,black,backCol)
119             drawLED(n,expandT == 1<<n)
120         for n in range(6,9): # voltage options
121             drawWords("x"+str(1<<(n-6)),220+(n-6)*30,320,black,backCol)
122             drawLED(n,expandV == 1<<(n-6))
123         drawLED(9,measureTime)
124         drawLED(10,measureVolts)
125         drawLED(11,stLed)
126         drawLED(12,svLed)
127         drawLED(17,button)
128         for n in range(13,17):
129             drawLED(n,run[n-13])
130         if measureTime :
131             t = (cursorT>>1)*sampleTime / expandT
132             drawWords(" "+trunk(t,5)+" "+chr(0x3bc)+"S",640,197,black,pramCol) # current time
133             drawWords(" "+trunk(savedTime,5)+" "+chr(0x3bc)+"S",640,217,black,pramCol)
134             drawWords(" "+trunk(t-savedTime,5)+" "+chr(0x3bc)+"S",640,237,black,pramCol) # delta time
135             if t-savedTime != 0 :
136                 drawWords((trunk(1000000 / abs(t-savedTime),5))+" Hz",640,257,black,pramCol)
137         if measureVolts :

```

```

137     vDisp = (((1024-cursorV)>>2)-128)*volts_sample * vMag
138     delta = vDisp - savedVoltage
139     drawWords(" "+trunk(delta,4)+" V",640,167,black,pramCol)
140     drawWords(" "+trunk(savedVoltage,4)+" V",640,147,black,pramCol)
141     drawWords(" "+trunk(vDisp,4)+" V",640,127,black,pramCol)
142
143 def trunk(value, place): # truncate a value string
144     v=str(value)+"000000"
145     if value>0:
146         v = v[0:place]
147     else:
148         v = v[0:place+1] # extra place for the minus sign
149     return v
150
151 def drawLED(n,state): # draw LED
152     if state :
153         pygame.draw.rect(screen,(100,10,5),LedRect[n],0)
154     else :
155         pygame.draw.rect(screen,(255,255,255),LedRect[n],0)
156
157 def defineControls():
158     global LedRect, resultsRect
159
160     for n in range(0,6):
161         LedRect[n] = pygame.Rect((10+n*30,336),(15,15))
162     for n in range(6,9):
163         LedRect[n] = pygame.Rect((220+(n-6)*30,336),(15,15))
164     LedRect[9] = pygame.Rect((440,336),(15,15)) # time
165     LedRect[10] = pygame.Rect((486,336),(15,15)) # volts
166     LedRect[11] = pygame.Rect((540,336),(15,15)) # save time
167     LedRect[12] = pygame.Rect((586,336),(15,15)) # save volts
168     LedRect[13] = pygame.Rect((545,100),(15,15)) # run
169     LedRect[14] = pygame.Rect((580,100),(15,15)) # single
170     LedRect[15] = pygame.Rect((628,100),(15,15)) # freeze
171
172     LedRect[16] = pygame.Rect((676,100),(15,15)) # trigger
173     LedRect[17] = pygame.Rect((650,300),(20,20)) # button
174     resultsRect = pygame.Rect((639,125),(90,153))
175
176 def plotWave():
177     global vMag
178     lastX=0 ; lastY=0
179     vMag = 2 # adjust voltage scale
180     if expandV == 1:
181         vMag = 4
182
183     if expandV == 4:
184         vMag = 1
185
186     drawGrid()
187     s = 0 # sample pointer
188     for n in range(0, displayWidth, expandT):
189         y = (512-inBuf[s])/vMag + chOfff
190         if n != 0:
191             pygame.draw.line(display,(0,200,0),(lastX ,lastY), (n ,y ),2)
192             lastX = n
193             lastY = y
194             s += 1
195
196     if measureTime :
197         pygame.draw.line(display,(0,0,255),(cursorT>>1,0), (cursorT>>1,256),1)
198         if savedTimeC != -1:
199             for n in range(0,256,12):
200                 pygame.draw.line(display,(0,0,255),(savedTimeC,n),(savedTimeC,n+6),1)
201
202     if measureVolts :
203         pygame.draw.line(display,(255,0,0),(0,cursorV>>2), (512,cursorV>>2),1)
204     if savedVoltsC != -1:

```

```

205         for n in range(0,512,12):
206             pygame.draw.line(display,(255,0,0),(n,savedVoltsC),(n+6,savedVoltsC),1)
207
208     if run[3] : # use trigger
209         y = (triggerC-512)//vMag + chOff
210         for n in range(0,512,12):
211             pygame.draw.line(display,(255,128,0),(n,y),(n+6,y),1)
212
213 def drawScope(): # put display onto scope controls
214     screen.blit(display,(10,10))
215     pygame.display.update()
216
217 def drawWords(words,x,y,col,backCol) :
218     textSurface = font.render(words, True, col, backCol)
219     textRect = textSurface.get_rect()
220     textRect.left = x
221     textRect.top = y
222     screen.blit(textSurface, textRect)
223
224 def readArduino(): # get buffer and controls
225     global cursorT, cursorV, triggerC, run
226     global t0, t1, t, voltmax, voltmin
227     voltmax = 0
228     voltmin = 0
229     if run[2] : #if in freeze mode funnel data into junk
230         for i in range(0,1024):
231             junk = sampleInput.read()
232
233     else: # otherwise read into the buffer
234         for i in range(0,512):
235             inBuf[i] = (((ord(sampleInput.read())) << 8) | ord(sampleInput.read()))
236
237     cursorT = (((ord(sampleInput.read())) << 8) | ord(sampleInput.read()))
238     cursorV = 1024 - (((ord(sampleInput.read())) << 8) | ord(sampleInput.read()))
239     triggerC = 1024 - (((ord(sampleInput.read())) << 8) | ord(sampleInput.read()))
240
241
242     if run[1]: #single sweep requested
243         run[1] = False
244         run[2] = True # put in freeze mode
245         updateControls(True)
246
247 def handleMouse(pos): # look at mouse down
248     global expandT,expandV,measureTime,measureVolts,svLed,stLed
249     global savedVoltsC, savedTimeC, run, zoomT
250     for n in range(0,6) :
251         if LedRect[n].collidepoint(pos):
252             expandT = 1<<n
253
254     for n in range(6,9) :
255         if LedRect[n].collidepoint(pos):
256             expandV = 1<<(n-6)
257
258     if LedRect[9].collidepoint(pos): #toggle time measurement
259         measureTime = not(measureTime)
260         if not measureTime :
261             savedTimeC = -1
262
263     if LedRect[10].collidepoint(pos):
264         measureVolts = not(measureVolts) # toggle volts measurement
265         if not measureVolts :
266             savedVoltsC = -1
267
268     if LedRect[11].collidepoint(pos) and measureTime: # save time
269         stLed = True
270         savedTimeC = cursorT>>1
271         |

```

```

272     if LedRect[12].collidepoint(pos) and measureVolts: # save volts
273         svLed = True
274         savedVoltsC = cursorV>>2
275
276     # run controls logic
277     if LedRect[13].collidepoint(pos) and not run[1]: # run
278         run[0] = not(run[0])
279         if not run[0]:
280             run[2] = True
281         else:
282             run[2] = False
283
284     if LedRect[14].collidepoint(pos): # single
285         run[1] = True
286         run[0] = False
287         run[2] = False
288         run[4] = True
289         updateControls(False)
290         drawScope()
291
292     if LedRect[15].collidepoint(pos) and not run[1]: # freeze
293         run[2] = not(run[2])
294         if not run[2]:
295             run[0] = True
296         else:
297             run[0] = False
298
299     if LedRect[16].collidepoint(pos): # trigger
300         run[3] = not(run[3])
301
302     if LedRect[17].collidepoint(pos): # screenshot
303         pygame.image.save(screen, "image.png")
304
305     | updateControls(False)
306
307 def handleMouseUp(pos): # look at mouse up
308     global savedVoltage,savedTime, svLed, stLed, run
309     if LedRect[12].collidepoint(pos) and measureVolts:
310         savedVoltage = vDisp
311         svLed = False
312         updateControls(False)
313     if LedRect[11].collidepoint(pos) and measureTime:
314         savedTime = (cursorT>>1)*sampleTime / expandT
315         stLed = False
316         updateControls(False)
317     if LedRect[14].collidepoint(pos): # single
318         run[4] = False
319         updateControls(False)
320
321 def terminate(): # close down the program
322     pygame.quit() # close pygame
323     os._exit(1)
324
325 def checkForEvent(): # see if we need to quit
326     event = pygame.event.poll()
327     if event.type == pygame.QUIT :
328         terminate()
329     if event.type == pygame.KEYDOWN :
330         if event.key == pygame.K_ESCAPE :
331             terminate()
332         if event.key == pygame.K_s : # screen dump
333             os.system("scrot -u")
334     if event.type == pygame.MOUSEBUTTONDOWN :
335         handleMouse(pygame.mouse.get_pos())
336     if event.type == pygame.MOUSEBUTTONUP :
337         handleMouseUp(pygame.mouse.get_pos())
338
339     |
340 # Main program logic:
341 if __name__ == '__main__':
342     main()
343

```

APPENDIX C

GANTT CHART SDP 1

MONTH ACTIVITIES	FEB	MAR	JUNE	JULY	AUG
TOPIC UNDERSTANDING					
PROJECT DISCUSSION					
BASIC DESIGN					
MATERIAL SELECTION					
DRAFT PROPOSAL					
PRESENTATION					
REDESIGN PROJECT FOR IMPROVEMENT					

APPENDIX D

GANTT CHART SDP 2

WEEK															
ACTIVITIES	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BUYING THE REQUIRED MATERIALS															
ASSEMBLE THE IMPORTANT HARDWARES															
PROGRAMMING FOR ADC CONVERSION															
PROGRAMMING FOR OSCILLOSCOPE APPLICATION															
RESEARCH FOR OTHER ALTERNATIVE ADC UNIT															
REDESIGN THE PROJECT															
TESTING															
PRESENTATION															
FULL HARDWARE ASSEMBLY															