



DESIGN OF MIDI DECODEK AND CONTRKOLLER
PROGRAM FOR MIDI AUTO SAXOPHONE

MUHAMMAD HALIFI BIN MUHAMAD

Report submitted in partial fulfillment of the requirements
for the award of Bachelor of Mechatronics Engineering

Faculty of Manufacturing Engineering
UNIVERSITI MALAYSIA PAHANG

JUNE 2013

UNIVERSITI MALAYSIA PAHANG

BORANG PENGESAHAN STATUS TESIS*

JUDUL: **DESIGN OF MIDI DECODER AND CONTROLLER PROGRAM
FOR MIDI AUTO SAXOPHONE**

SESI PENGAJIAN: 2012/2013

Saya MUHAMMAD HALIFI BIN MUHAMAD (900827-11-5011)
(HURUF BESAR)

mengaku membenarkan tesis (Sarjana Muda/Sarjana/Doktor Falsafah)* ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Tesis adalah hakmilik Universiti Malaysia Pahang (UMP).
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. **Sila tandakan (√)

SULIT

(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)

TERHAD

(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan)

TIDAK TERHAD

Disahkan oleh:


(TANDATANGAN PENULIS)


(TANDATANGAN PENYELIA)

Alamat Tetap:

**46-1 KG TELAGA PAPAN
24300 KERTEH
TERENGGANU DARUL IMAN**

**PROF IR DR AHMAD FAIZAL
BIN MOHD ZAIN**
(Nama Penyelia)


Tarikh: **11 JULAI 2013**

Tarikh: **11 JULAI 2013**

- CATATAN:
- * Potong yang tidak berkenaan.
 - ** Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh tesis ini perlu dikelaskan sebagai atau TERHAD.
 - ♦ Tesis dimaksudkan sebagai tesis bagi Ijazah doktor Falsafah dan Sarjana secara Penyelidikan, atau disertasi bagi pengajian secara kerja kursus dan penyelidikan, atau Laporan Projek Sarjana Muda (PSM).


SUPERVISOR'S DECLARATION

I hereby declare that I have checked this project report and in my opinion this project is satisfactory in terms of scope and quality for the award of Bachelor of Mechatronics Engineering.

Signature : 
Name of Supervisor : Prof. Ir. Dr. Ahmad Faizal Bin MohdZain
Position : Professor
Date : 11 July 2013

STUDENT'S DECLARATION

I hereby declare that the work in this report is my own except for quotations and summaries which have been duly acknowledged. The thesis has not been accepted for any degree and is not concurrently submitted for award of another degree.

Signature :  .
Name : Muhammad Halifi Bin Muhamad
ID Number : FB09037
Date : 11 July 2013

*Dedicated to my beloved parents,
Mr Muhamad Bin Zakaria and Mrs Normah Binti Salleh.*

ACKNOWLEDGEMENTS

Completing my degree is probably the most challenging activity of my first 23 years of my life. The best and worst moments of my journey as student of Mechatronics Engineering have been sharing with many people. It has been a great privilege to spend several years in the Faculty of Manufacturing Engineering at University Malaysia Pahang, and its members will always remain dear to me.

My first debt of gratitude must go to my advisor, Prof Ir Dr Ahmad Faizal Bin Mohd Zain. He patiently provided the vision, encouragement and advice necessary for me to proceed through my journey to gain knowledge and complete my project.

I wish to thank my parents, Muhamad bin Zakaria and Normah binti Salleh. Their love provided my inspiration and was my driving force. I owe them everything and wish I could show them just how much I love and appreciate them.

Special thanks to my fellow friends, Diana, Anas, Shukran, Anis Hazwana, Khairunnisa and Afifah for their support, guidance and helpful suggestions. Not forget to all Mechatronic Engineering friends that willing to cooperate with me to complete my study for 4 years. Their guidance has served me well and I owe them my heartfelt appreciation.

Members of Faculty of Manufacturing Engineering also deserve my sincerest thanks, their friendship and assistance has meant more to me than I could ever express. I could not complete my work without invaluable friendly

ABSTRACT

The saxophone is a musical instrument that lends itself to be controlled electronically. When air is blown into the mouthpiece, tones are produced through a combination of saxophone key presses. A MIDI decoder program is used to extract the MIDI data that send by the computer and send the instructions to the controller program. the controller program sends the instruction signals to the 18 servo motors to depress the saxophone keys and solenoid valve to control the air flow into the saxophone mouthpiece. Each of the 30 musical notes is programmed to get the correct press of saxophone keys combination. An array of 18 switching signal is used for determining which tones to be produced by the saxophone. This cause the significant signal delay to produce a lag between the data input and key presses. At the end of the project, the MIDI decoder was able to deliver correct switching signals according to the saxophone fingering chart although there was a lag between the data input and key presses on the saxophone.

ABSTRAK

Saksofon adalah salah satu instrumen muzik yang membenarkan ia dikawal secara elektronik. Apabila udara ditiup ke dalam muncung saksofon, bunyi di keluarkan melalui kombinasi kunci-kunci saksofon yang ditekan. Program dekoder MIDI digunakan untuk mengeluarkan data MIDI yang dihantar oleh komputer dan menghantar arahan kepada program pengatur. Program pengatur akan menghantar isyarat arahan kepada 18 motor servo untuk menekan kunci kunci saksofon dan injap solenoid untuk mengawal aliran udara ke dalam muncung saksofon. Setiap 30 nota muzikal diprogram untuk mendapatkan kombinasi kunci saksofon yang ditekan dengan betul. 18 signal suis yang disusun di dalam rangkaian diguna untuk menentukan bunyi manakah yang perlu dikeluarkan oleh saksofon. Ini menyebabkan penangguhan signal untuk menghasilkan kelewatan di antara kemasukkan data dan kunci yang ditekan. Di akhir projek, program decoder MIDI berupaya menghantar signal suis yang tepat berdasarkan carta kekunci saksofon walaupun terdapat kelewatan di antara kemasukkan data dan kunci yang ditekan pada saksofon.

TABLE OF CONTENTS

| | Page |
|---|-------------|
| SUPERVISOR’S DECLARATION | ii |
| STUDENT’S DECLARATION | iii |
| DEDICATION | iv |
| ACKNOWLEDGEMENTS | v |
| ABSTRACT | vi |
| TABLE OF CONTENTS | viii |
| LIST OF TABLES | xi |
| LIST OF FIGURES | xii |
| LIST OF ABBREVIATIONS | xiii |
| | |
| CHAPTER 1 INTRODUCTION | 1 |
| | |
| 1.1 Introduction | 1 |
| 1.2 Project Background | 1 |
| 1.3 Problem Statement | 3 |
| 1.4 Project Objective | 4 |
| 1.5 Project Scope | 4 |
| 1.6 Expected Outcome | 4 |
| 1.7 Thesis Outline | 5 |
| | |
| CHAPTER 2 LITERATURE REVIEW | 6 |
| | |
| 2.1 Introduction | 6 |
| 2.2 Musical Instrument Digital Interface | 6 |
| 2.2.1 MIDI Message | 7 |
| 2.2.2 MIDI Decoder Program | 9 |
| 2.2.3 USB MIDI Converter | 12 |
| 2.3 Alto Saxophone | 13 |
| 2.3.1 Mechanic of Saxophone | 15 |
| 2.3.2 Alto Saxophone Fingering Chart | 15 |

| | | |
|-------------------|--|-----------|
| 2.4 | Arduino Microcontroller Board | 16 |
| 2.4.1 | Arduino Mega 2560 Microcontroller Board | 17 |
| 2.4.2 | Arduino Software | 18 |
| 2.5 | Universal Asynchronous Receiver Transmitter | 20 |
| CHAPTER 3 | METHODOLOGY | 22 |
| 3.1 | Introduction | 22 |
| 3.2 | Process Flow of Project | 22 |
| 3.3 | Block Diagram of Complete System | 23 |
| 3.4 | MIDI Decoder Program to Extract MIDI Data | 25 |
| 3.5 | Controller Program to Control Saxophone Keys | 27 |
| 3.5.1 | Controller Programming Flowchart | 28 |
| 3.6 | Opto Coupler Circuit for MIDI Shield | 30 |
| CHAPTER 4 | RESULT AND DISCUSSION | 31 |
| 4.1 | Introduction | 31 |
| 4.2 | Propagation Delay Time in Arduino Programming | 31 |
| 4.3 | Pulse Width Modulation Duty Cycle | 33 |
| 4.4 | Result of Controller Programming Output | 37 |
| 4.5 | Propagation Delay Time at Opto Coupler Circuit | 41 |
| CHAPTER 5 | CONCLUSION AND RECOMMENDATION | 43 |
| 5.1 | Introduction | 43 |
| 5.2 | Conclusion | 43 |
| 5.3 | Recommendation | 44 |
| REFERENCES | | 45 |

| | |
|--|----|
| APPENDICES | 46 |
| A Project Flowchart | 46 |
| B Saxophone Fingering Chart | 47 |
| C PCB Drawing for MIDI Shield Board | 49 |
| D ATmega2560 Microcontroller Pin Configuration | 50 |
| E Arduino Programming | 51 |

LIST OF TABLES

| Table No. | | Page |
|------------------|---|-------------|
| 2.1 | Musical Note Data | 6 |
| 4.1 | Result of Calculation of Servo Duty Cycle | 30 |

LIST OF FIGURES

| Figure No. | | Page |
|-------------------|--|-------------|
| 1.1 | MIDI Auto Sax Part | 2 |
| 1.2 | WASEDA Flutist Robot | 3 |
| 2.1 | MIDI Converter Using Case Diagram | 10 |
| 2.2 | MIDI Converter Activity Diagram | 11 |
| 2.3 | USB-MIDI Interface | 12 |
| 2.4 | Alto Saxophone Structure | 14 |
| 2.5 | Saxophone Fingering Chart | 16 |
| 2.6 | Example of ATMEL AVR Microcontroller | 17 |
| 2.7 | Arduino Mega 2560 board | 18 |
| 2.8 | Example of Arduino Sketch Program | 19 |
| 2.9 | UART Data Arrangement | 20 |
| 3.1 | Block Diagram of Complete System | 24 |
| 3.2 | MIDI Decoder Program Flowchart | 26 |
| 3.3 | Saxophone Fingering Chart | 27 |
| 3.4 | Controller Program Flowchart | 29 |
| 3.6 | Opto Coupler Circuit | 30 |
| 4.1 | Propagation Delay Time in Arduino Mega 2560 Programming | 32 |
| 4.2 | Angular Distance of Hand of Servo Motor Need to Travel | 34 |
| 4.3 | Programming of Controller Program | 38 |
| 4.4 | Example of Notes Switching | 39 |
| 4.5 | MIDI Shield Board | 40 |
| 4.6 | Opto Coupler Circuit Attach to Oscilloscope | 41 |
| 4.7 | Example of C ₄ Notes pressed | 42 |
| 4.8 | Propagation Delay between Input and Output of the Opto Couple | 42 |

LIST OF ABBREVIATIONS

| | |
|------|---|
| MIDI | Musical Instrument Digital Interface |
| UART | Universal Asynchronous Receiver Transmitter |
| PWM | Pulse Width Modulation |
| USB | Universal Serial Bus |
| LED | Light Emitting Diode |
| IDE | Integrated Development Environment |
| MSB | Most Significant Bit |

CHAPTER 1

INTRODUCTION

1.1 Introduction

This chapter will briefly explain about the introduction of this project. The general information about the project is including the discussion of topics related to this project. These chapters consist of the project background, problem statement, objectives, project scope and expected outcome. The information in this chapter is important to make further study of the problem.

1.2 Project Background

Music is an art that involves a combination of sounds that can consign human feelings and thoughts. Of all the music instruments, saxophone is one kind that is contributing a lot to the music industry nowadays. The saxophone is a musical instrument that belongs to woodwind instrument family. Originally it is popular with military bands; saxophone soon became a part of popular music and jazz. They are made of brass and played with a single reed mouthpiece similar to playing clarinet. But, do we offend see the saxophone can play automatically?

Alternatively, the saxophone is the suitable instrument to apply the self-playing concept. Briefly about the saxophone, they were built in many types such as Alto Saxophone, Soprano Saxophone, Tenor Saxophone and many more. As the other instrument, saxophones were assembling in many parts. The main part of the saxophone is the mouthpiece which consists of the mouthpiece and the reed, the hollow body part, and the saxophone keys. Each type of saxophones was different in note key they can

play, the numbers of key pressed and the octave note. Usually, the alto saxophone is the one that chosen by many people due to the note that can be play are standard. The MIDI Auto-Sax will use the Alto Saxophone as the instrument. The Alto saxophone has 23 keys on the body to control which note will produce when it sounds.

The invention of the MIDI Auto-Sax is actually an initiative to build up an autonomous Alto saxophone that played via MIDI by combining few elements including MIDI decoder program, Digital I/O and mechanical and pneumatic actuator. A saxophone that can tune automatically through the MIDI is the aim of this project. The MIDI Auto-Sax is one of Musical Robot that will conduct the saxophone automatically. The autonomous parts are controlled by the computer program or the microprocessor.

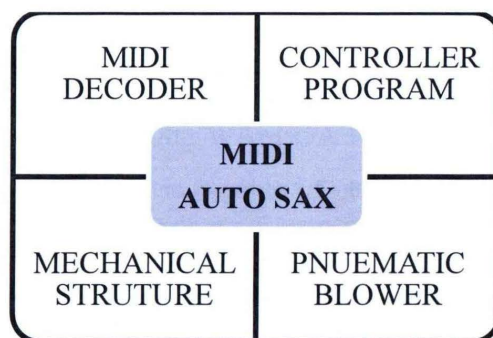


Figure 1.1: MIDI Auto Sax Part

The MIDI Auto Sax is divided into four parts as shown in Figure 1.1. The MIDI decoder and controller program is the software part of the device while the mechanical structure and pneumatic blower are the hardware part which consist of electronic, pneumatic and mechanical elements. This intention of this project is to complete the software part of the MIDI Auto Sax device.

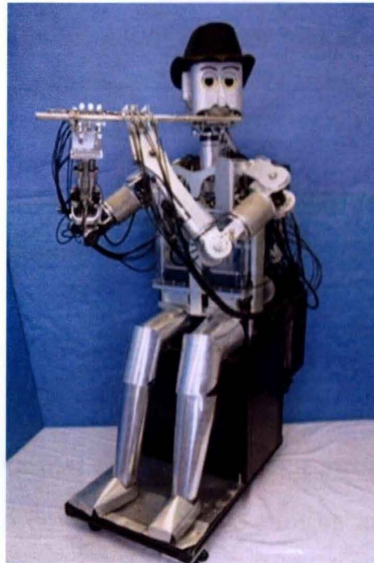


Figure 1.2: WASEDA Flutist Robot

The general idea of this device is the MIDI input converts by the decoder to send out output to play the instrument. Taking WASEDA Flutist Robot in Figure 1.2 as an example, it is a well-known robot that is capable of playing a wind instrument such as saxophone, clarinet and flute. These are the humanoid robots that are programmed based on human minds and body. The program and the body are the mechanism to make the robot act.

1.3 Problem Statement

In real life, the way to play saxophone is complicated and need a very well trained to play the instruments. Maybe some people like to hear and very fanatic on saxophone sound but cannot afford to pay the saxophone player to play the saxophone every day. Through the technology development become more sophisticated nowadays, the concept of self-playing musical instrument can be applied to the Saxophone and see how far the system can play the saxophone compare to human. In order to control the Auto Saxophone, the MIDI will file is the suitable format to send the musical instruction to the device.

When it comes to MIDI, there are a few questions on how making the Musical message can be sent to the device to play the saxophone.

- How to extract the MIDI message from a MIDI file?
- How to control the Servo Motors and Solenoid Valve from the input data?

1.4 Project Objective

- To design the MIDI decoder that extracts the MIDI message
- To construct a program to control the Servo motors and solenoid valve on the Alto saxophone

1.5 Project Scope

- The MIDI Auto Saxophone device is only can play Alto Saxophone
- Read only MIDI data
- A program to control the 18 servo motors to press the keys of saxophone, 1 servo motor to control the reed and a solenoid valve to allow the air enter the saxophone mouthpiece
- Constant air pressure flow to the mouthpiece

1.6 Expected Outcome

- The MIDI message can be extracted and send through the USB port
- The servo motors on the saxophone keys are correctly pressed according to the **Saxophone Fingering Chart**
- The Pneumatic valve can switch according to the signal sent by the **microcontroller**

1.7 Thesis Outline

This thesis is a documentary delivering the idea generated, concept applied, activities done, and finally the project product. It consists of five chapters. Following is the chapter by chapter description of information in this thesis.

Chapter 1 explains about the introduction of the project. The general information about the project is including the discussion of topics related to this project. These chapters consist of the project background, problem statement, objectives, project scope and expected outcome. The information in this chapter is important to make further study of the problem.

Chapter 2 is a literature review of theoretical concepts applied in the project. This chapter explains about the topic that is related to MIDI, alto saxophone, microcontroller and data communication. The sources are taken from the journals, articles, books and website. The literature review is helping in order to provide important information regarding the previous research which is related to this project.

Chapter 3 describe about the method used to decode the MIDI message and send the switching signal and PWM to the servo motors and pneumatic solenoid valve. The research methodology is a set of procedures or methods used to conduct the research. Methodology is needed for a guideline in order to ensure the result is accurate based on the objective.

Chapter 4 will discuss the result obtain from the project and discuss the related result of the project. The result of this project will include the result of the program, output signal that needs to come out and the calculation of the Pulse Width Modulation duty cycle.

Chapter 5 explain about the conclusion and recommendation of the whole project paper. This chapter will discuss mainly about the conclusion of the whole project. Other than that, this chapter also will briefly explain the recommendation for the future development of the project.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter will explain about the topic that is related to MIDI, data communication, microcontroller and alto saxophone. The sources are taken from the journals, articles, books and website. The literature review is helping in order to provide important information regarding the previous research which is related to this project. The information gathered in this chapter is important before proceeding to the analysis and further study of the project.

2.2 Musical Instrument Digital Interface (MIDI)

MIDI (Musical Instrument Digital Interface) is a protocol used to transmit musical instructions to a music device or instrument that is capable of converting the instructions into musical sounds ^[1]. The musical instructions were in digital waveform and transmitted through the MIDI cable to the device. The MIDI data can be stored in an architecture file format which is called MIDI file (.mid). In MIDI, there was also a standard MIDI text event. It contains the text data, such as trademark, song title, and copyright information. They were commonly stored in midi file along with the MIDI message data. But, the standard MIDI text events were only benefited to human to show the information of the MIDI file. They typically not transmitted along to the MIDI device.

2.2.1 MIDI Message

The MIDI messages have their own protocol that is arranged in sequence. There are the note message, tempo events, controller events, and duration event for example. All of these events were sent to the synthesizer in digital message which is in Byte. The MIDI serial data were flowing at the rate of 31250 bits per second and being organized in 10-bit words ^[2]. The first of the 10 bits is called as the start bit which is always 0 for midi data. The next 8 bits were indicated the information data. The last 1 bit is the stop bit which is always 1. MIDI data were sent in asynchronous data transmission. A MIDI message consists of status byte and followed by 0, 1 or 2 data byte. The MSB bit of the status byte is always 1 and the MSB of data byte is always 0. The MIDI messages are shown below:

| Status Byte | 1 st Data Byte | 2 nd Data Byte |
|-----------------|---------------------------|---------------------------|
| 1 T T T N N N N | 0 X X X X X X X | 0 X X X X X X X |

T – Type of Status message

N – Channel Number

X – Data in Binary

The status byte is used to identify and instruct the receiving devices that the particular MIDI function and channel is being addressed. The MIDI channels were up to 16 channels for user used. The data byte is used to encode the actual numeric values that are attached to the accompanying status byte. The velocity data byte was used to indicate the volume. The velocity can be defined from 0 to 127 in decimal. Table 2.1 shows the MIDI musical note data that are arranged in sequence and numbered in decimal value ^[3].

Table 2.1: Musical Note Data (in decimal)

| | | M | U | S | I | C | N O T E | | | | | | |
|----------------------------|----|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| | | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| O C T A V E | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| | 1 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| | 2 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| | 3 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| | 4 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| | 5 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |
| | 6 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |
| | 7 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| | 8 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 |
| | 9 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 |
| | 10 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | - | - | - | - |

There are 12 notes in an octave and MIDI can support up to $11\frac{2}{3}$ octaves. This means, the total notes can be read and write in MIDI protocol is 128 notes. The music notes start with the C note of the first octave which in the table the first octave is indicated as 0. The last note in the first octave is B and the note after B note is repeated again with the C note in the next octave. The note sequence remains the same until the octave of 11. In, 10 octave, there are only 8 notes. Each of the notes is identified in MIDI by their number of arrangements.

The Alto saxophone note range is between $A\#_3$ to $D\#_6$. But, in this project, note range is up one octave for easier song compiler in a computer. The range of the note should receive from the computer is $A\#_4$ to $D\#_7$ as highlighted in Table 2.1. The data value of the computer must be in between 58 to 87 to make the saxophone play. Else, the saxophone will not play the other musical note data.

2.2.2 MIDI Decoder Program

MIDI decoder is used to decode and extract the message in the MIDI files. After extracting the MIDI messages, the extracted data are sent to the synthesizer to produce the sound according to the message sent.

Based on Figure 2.1, the user will start by giving an input to the application regarding the location of the MIDI file ^[4]. The file directory can be done in two ways; using the open dialog box, where the user can browse the location of MIDI file and choose it and by entering the file path manually in the text box. Then the file validity will be checked whether the MIDI files chosen are MIDI type 2, a non MIDI file, non-existent file or correct format of MIDI data. If the file is detected as MIDI type 2, a non MIDI file, or non-existent file, the system will show the error dialog box to notify the user the file is not correct. If correct path is confirmed; the user can run the MIDI converter.

Figure 2.2 shows the activity diagram of MIDI converter. After the MIDI file is chosen, the application will read the header of the file and check whether it is in type 0 or 1 MIDI file. If the parameter is confirmed, the application will start to read the events contained in the MIDI file until the end. Then the application will show the result generated from the converter.

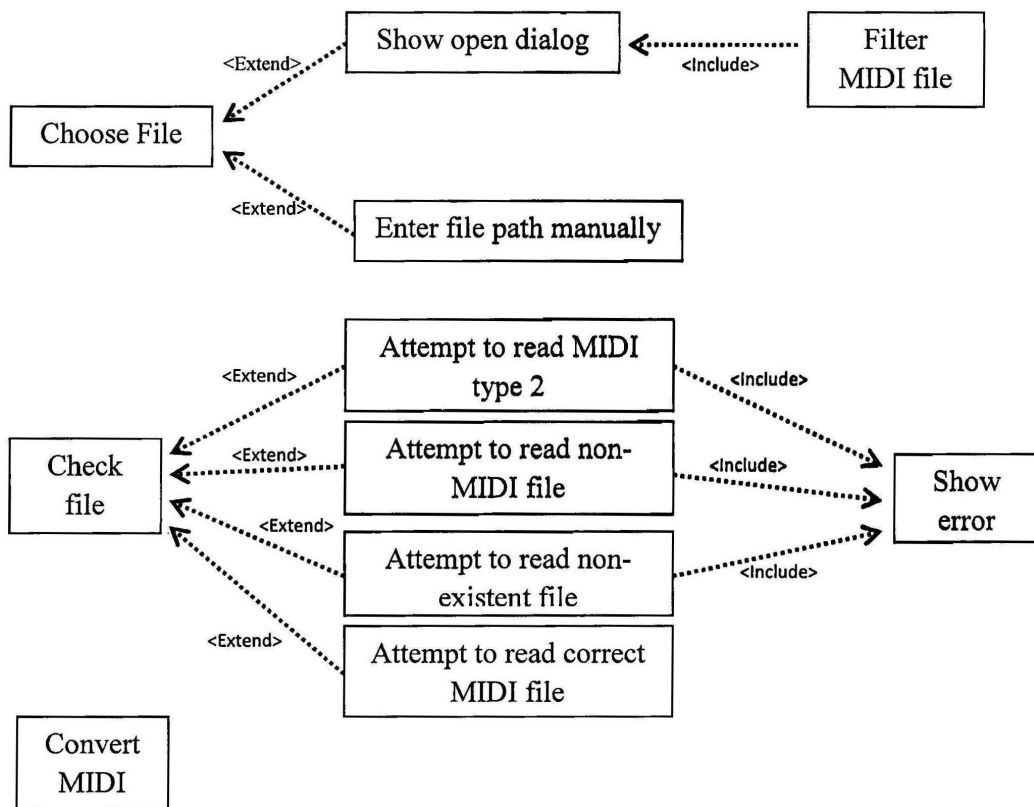


Figure 2.1: MIDI Converter Using Case Diagram

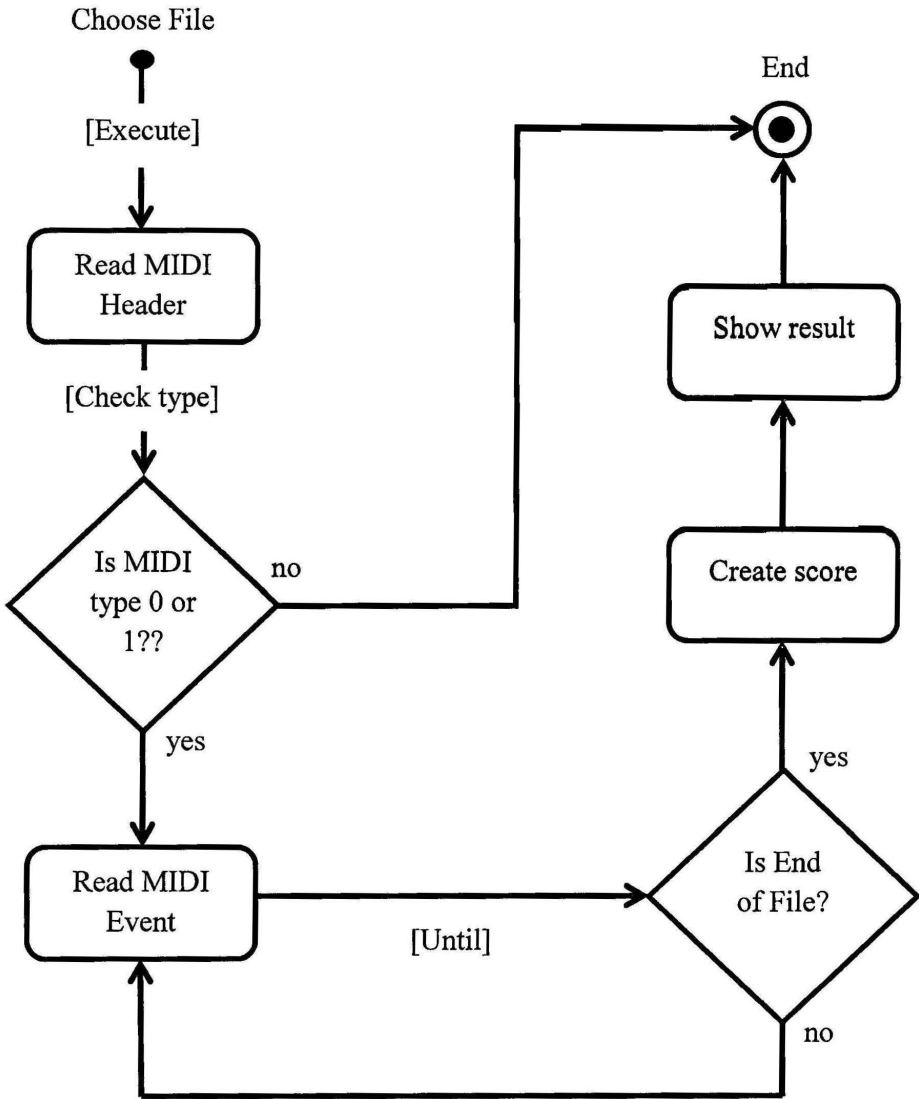


Figure 2.2: MIDI Converter Activity Diagram

2.2.3 USB-MIDI Converter

USB MIDI Converter is used in providing the link between the host and USB-MIDI interface. By interfacing with the USB hub, the MIDI data can be exchanged between the host and the USB-MIDI endpoints of the device ^[5]. The device main function is to match clock speeds between the MIDI device and the computer. The USB-MIDI Converter were typically contained more MIDI IN and/or MIDI out endpoints. These endpoints use bulk transfer to exchange data with the host. That means, large USB-MIDI data can be transferred simultaneously at the same time to many devices without missing data.

Figure 2.3 shows the block diagram of USB-MIDI interfaces. The USB is connected to the USB port from the computer. For receiving data from the MIDI devices such as the keyboard, the data from the keyboard were transferred from the MIDI OUT from the keyboard to the MIDI IN port of the cable. The data then sent to the USB-MIDI Converter which the circuit is functioning to match the clock speed between the MIDI devices and computer. Then, the data were transmitted to the Receiver pin on the USB.

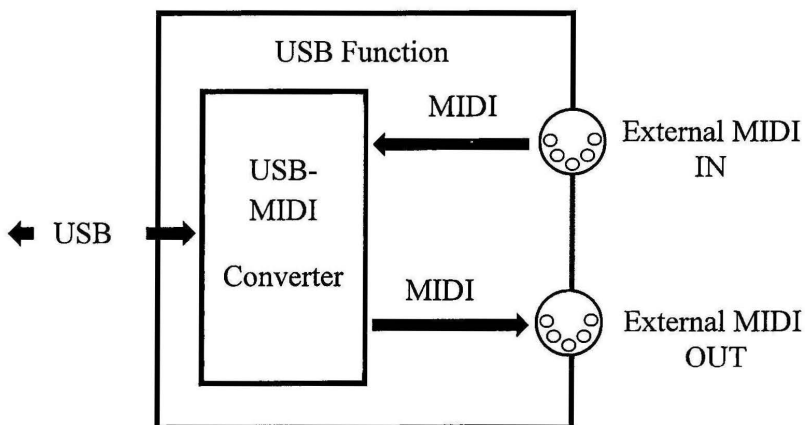


Figure 2.3: USB-MIDI Interfaces

For transmitting data from the computer, the process is backward processed of the receiving data from MIDI devices. Beginning the process with the data transmit from the computer to the transmitter pin on the USB then sent to the USB-MIDI Converter. Then the data were sent to the MIDI OUT port of the cable which the port is connected to the MIDI IN of the MIDI devices.

This device is used in the project to transfer the MIDI data from the USB port of the computer to the microcontroller serial communication pin. The software will send the MIDI data based on a musical note that played at that time to the corresponding USB port that connects to the USB MIDI cable. Then the data will transmit to the microcontroller.

2.3 Alto Saxophone

Alto saxophone is one of the members of the saxophone family and categorized as woodwind instrument. The Alto saxophone was invented by Adolphe Sax of Belgian in 1845 [6]. The invention of the alto saxophone by Adolphe Sax is to combine the tone of the clarinet with the brassy, brighter sound of trumpet. The size of the Alto saxophone is smaller than the Tenor saxophone but larger than Soprano saxophone [8]. Alto saxophone is the type most used in classical composition.

The first Alto saxophone was relatively simple compared to modern saxophone nowadays. The Alto saxophones nowadays have the improvement in the key mechanism. The original Alto saxophone had two to three octave keys compared to one octave key associated with saxophone today's [6]. Usually, the saxophone note range has been between B below the staff to the F fourth space above [8]. The modern Alto saxophone now has keys added to make the high F# note. The range of the Alto saxophone is from concert C#₃ to concert G#₅ [7].

The Alto Saxophone consists of several parts [9]. Referring to the Figure 2.4, the figure shows the structure of the Alto Saxophone. From the top of the saxophone is the mouthpiece. the mouthpiece is connected to the neck of the saxophone. This is where the musician place lips and blows air into the instrument to produce sound. The reed of

the saxophone is attached with the mouthpiece. The saxophone neck is a metal tube attached to the body of the saxophone and removable. The octave vent is a single hole that connects the neck and body of the saxophone. There were octaves keys attach to the octave vent. The saxophone body is a cone shaped brass tube that has plates attached to the body and holds the rods keys and other parts. The straight part of the body is called tube. The U-shape part is called bow. The end of the saxophone body is called bell.

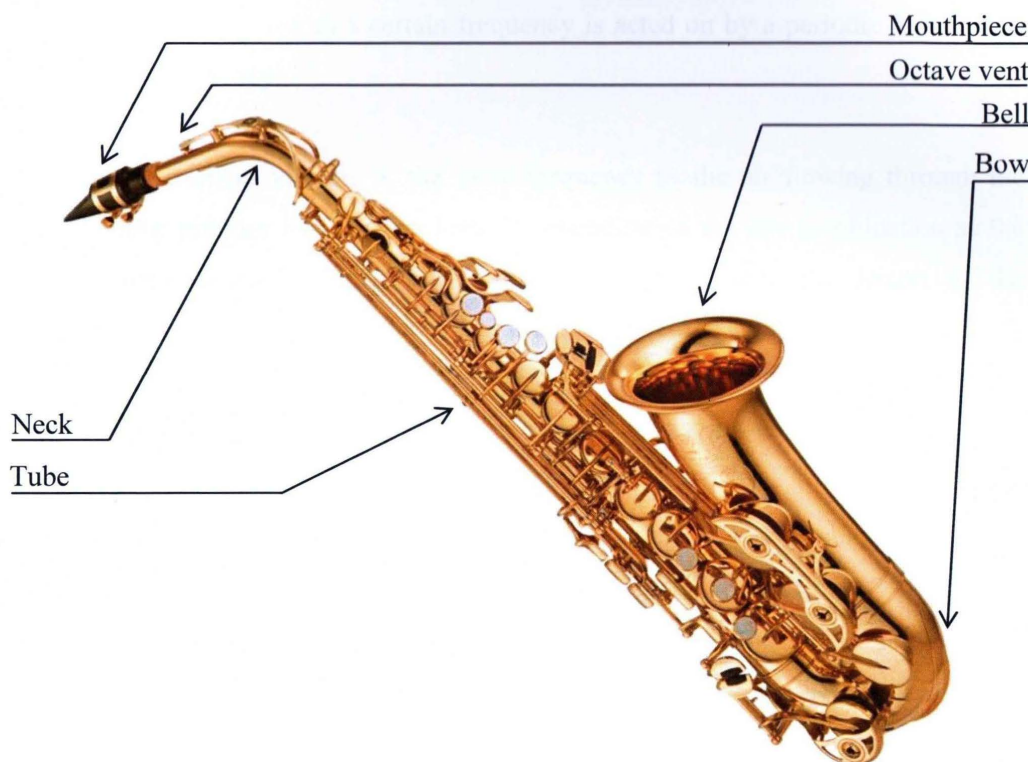


Figure 2.4: Alto Saxophone Structure. Reprinted from How to play the Alto Saxophone, in *Instructable*. Retrieved on December 15, 2012 from <http://www.instructables.com/id/How-to-play-the-Alto-Saxophone>. Copyright by max596789.

2.3.1 Mechanic of the Saxophone

The Alto saxophone is one of the reed instruments^[6]. The reed is made from a springy piece of cane and has vibrational tendency depending on the size and type of reed. The reed is secured to the mouthpiece and forced to vibrate when the musician started to blow the saxophone. The air vibrations within the instrument make the saxophone to resonate. Resonance refers to the creation of large amplitude vibrations as a system which vibrates with a certain frequency is acted on by a periodic disturbance with the same frequency.

The instrument vibrates at the same frequency as the air flowing through the body. The tone produce by the saxophone is depending on the key combination as the length of the instrument shortened or lengthened. The lower the length of the saxophone, the longer the time take by the wave to travel through the saxophone body and resulting low frequency.

2.3.2 Alto Saxophone Fingering Chart

In playing the Alto saxophone, to produce certain musical note it needs a combination of saxophone keys while the air enters the saxophone body. The combination of saxophone keys can refer to the saxophone fingering chart. The chart will tell which combination of the keys will produce the right note.

Figure 2.5 shows the example of saxophone fingering chart. Each of the musical notes has different of saxophone key combination. The pattern of the fingering chart refers to the arrangement of the keys on the saxophone. The black mark keys are the keys need to be pressed to play the corresponding musical note. The saxophone fingering chart is used in the project to get the correct result of switching signal for each key to the saxophone body. The switching signal is used to trigger which of the 18 servomotors will press the saxophone keys.

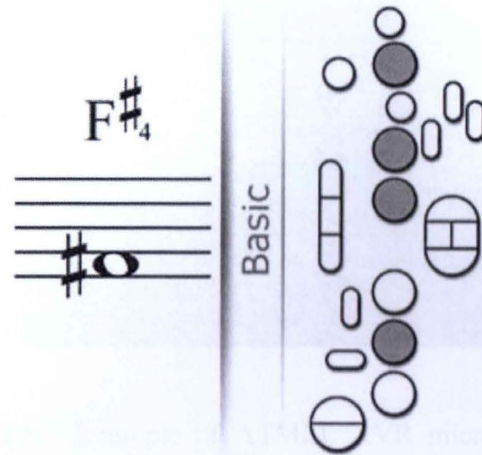


Figure 2.5: Saxophone Fingering Chart. Reprinted from Saxophone Fingering Chart Demo, in Appszoom. Retrieved on December 15, 2012, from http://www.appszoom.com/android_applications/reference/saxophone-fingering-chart-demo_bzazn.html. Copyright by Appszoom.

2.4 Arduino Microcontroller Board

Arduino is a single board microcontroller designed to make the process of using electronics in multidisciplinary project more accessible ^[10]. It is an open-source physical computing platform based on a simple input output board and a development environment that implements the open source programming language ^[11].

The hardware part of the Arduino microcontroller board consist of 8 bits Atmel AVR microcontroller that surrounded by a simple open source hardware. Figure 2.6 shows the example of Atmel AVR microcontroller which is the ATmega328 that used in the Arduino UNO board. The software part consists of a standard programming language compiler which is C Programming and a boot loader that used to execute the program to the processor of the Arduino board. It can be used to develop stand alone interactive project or communicate with the software in the computer.

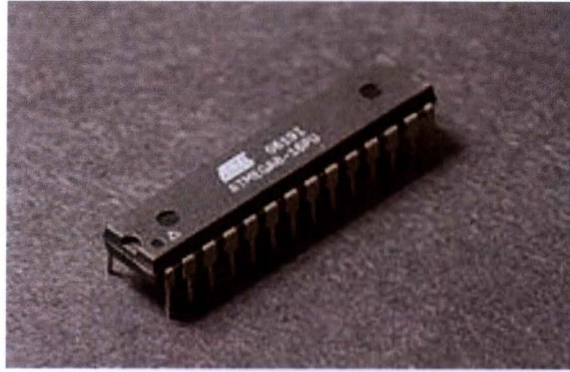


Figure 2.6: Example of ATMEL AVR microcontroller. Reprinted from Atmel AVR, in *Wikipedia*. Retrieved on December 19, 2012, from http://en.wikipedia.org/wiki/Atmel_AVR.

2.4.1 Arduino MEGA 2560 Microcontroller Board

The Arduino Mega 2560 is a microcontroller board that based on the ATmega2560 processor. It has 54 digital I/O pins on the board which 14 of them can be used as Pulse Width Modulation (PWM) signal output^[12]. It also has 16 analog input pins and 4 UART (Universal Asynchronous Receiver Transmitter) which consist of receiver and transmitter pins for 4 different serial communication. Referring to the Figure 2.7 that show the appearance of the Arduino Mega 2560, all the pins are numbered and shows on the board.

The microcontroller board is compatible with the most of the shields that designed for the other Arduino boards such as Arduino UNO and Arduino Due. The board is powered by the USB cable that connect to the computer or AC to DC adapter. It has 16 MHz crystal oscillator.

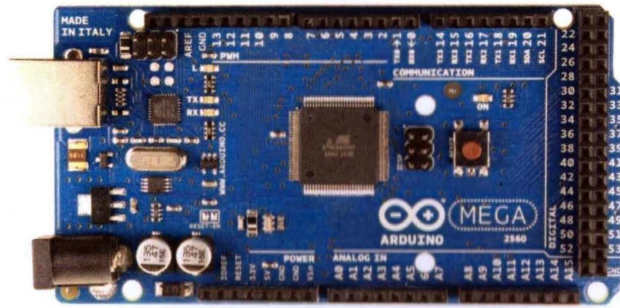


Figure 2.7: Arduino Mega 2560 board. Reprinted from Arduino Mega 2560, in *Arduino*. Retrieved on December 19, 2012, from <http://arduino.cc/en/Main/arduinoBoardMega2560>. Copyright by Arduino.

The Arduino Mega 2560 is suitable used in this project because the digital I/O pins of the board is enough to be used for 19 switching signal and give out the PWM signal compare to Arduino UNO that has only 13 digital I/O pins including the PWM output pin.

2.4.2 Arduino Software

The Arduino integrated development environment (IDE) is a multi platform application that is written in Java ^[11]. It includes a code editor with features such as syntax highlighting and capable compiling and uploading the program to the board in a single click. The program to write the code for the Arduino is called Sketch.

The program can write on the Arduino Sketch in C or C++ programming. It also comes out with software library which make the user more easier and simplify the program for the corresponding library used. To use the Sketch, user only needs to define two functions to make the program run which is setup and loop function.


```
int led = 13;
void setup()
{
    pinMode(led, OUTPUT);
}
void loop()
{
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
```

Figure 2.8: Example of Arduino Sketch Program

Figure 2.8 shows the example of an Arduino Sketch program in the computer. The sketch program is used in this project to write the MIDI decoder and controller programming, compile and execute to the Arduino Mega 260 microcontroller board.

2.5 Universal Asynchronous Receiver Transmitter (UART)

UART is one of the type of data communication that used in transmission of serial data either in transmitting or receiving. The UART is an integrated circuit consists of transmitters which convert the parallel data to serial data and the receiver which convert the serial data to parallel data ^[13]. This device can transmit up from 9600 up to 38400 bits per second. The UART system is based on the principle of data conversion in the shift register.

An asynchronous serial has advantage of less transmission line, high reliability, and long transmission distance. This form of serial data is widely used in data exchange between computer and peripheral. UART data transmissions have the protocol in transmitting data. UART usually include with start bit, data bit, parity bit, stop bit and idle state. The data arrangements are shown in Figure 2.2.

Figure 2.9 shows the data arrangement of UART. When a word is given to the UART, a bit called start bit is added to the beginning of each word that is to be transmitted. The start bit function is to alert the receiver that a word is about to be sent and to trigger the clock in the receiver. Then, followed by data bits of the word by the MSB come first into the synchronization with the clock in the transmitter.

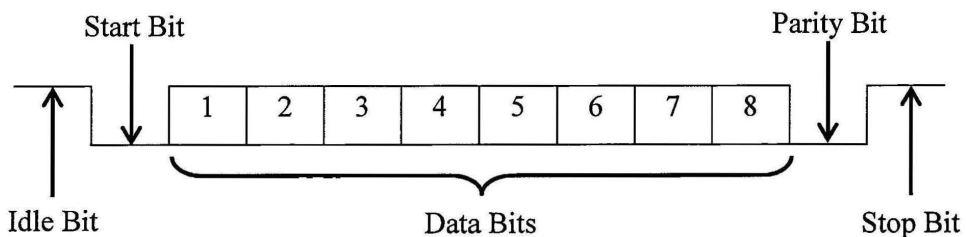


Figure 2.9: UART Data Arrangement

After entire data bits has been sent, the transmitter may add a parity bit that the transmitter generates. The parity bit can be used by the receiver to check the data error. Then followed by stop bit which means the whole word data has been sent. If the receiver detected an incorrect formatted data, the UART will signal a framing error. If another data byte is received before the previous data is read, the UART will signal an overrun error

CHAPTER 3

THEORETICAL ANALYSIS

3.1 Introduction

This chapter will describe about the method used to decode the MIDI message and send the signal to the actuators and pneumatic solenoid valve. The research methodology is a set of procedures or methods used to conduct research. Methodology is needed for a guideline in order to ensure the result is accurate based on the objective. There are several steps need to be followed to ensure the objective of the research can be achieved starting from finding literatures until submitting the final report.

3.2 Process Flow of Project

Flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. Flowchart is one of the step need project to make sure the flow of the project is in sequence and help the viewer to understand the flow of the process. Flow chart methodologies were constructed related to the scope of product as a guided principal to formulate this research successfully, in order to achieve the objectives of the project research (refer Appendix A).

3.3 Block Diagram of Complete System

In the MIDI Auto Sax project, all the elements involved in playing saxophone including saxophone keys control, blow and controlling the reed will be conducted at the same time. The system is divided into 3 major parts which is the opto coupler circuit, MIDI decoder and controller program.

The block diagram of the complete system is shown in Figure 3.1. The MIDI IN means the MIDI data are sent from the computer through the USB-MIDI cable to the opto coupler circuit. The opto circuit as the coupler between two different circuits in which this case is between the data flow from the computer and the data to be transmitted to the Arduino.

After the Arduino receives the data, a program call MIDI decoder is used to read the MIDI data receive. The data are read in sequence based on the MIDI protocol which is three data byte flows starting from status byte followed by note byte and velocity byte. Then the program will repeat again to read the data sequence. The MIDI decoder also takes the value of the data read and send them to the controller program to do the controlling job. The process flow of the MIDI decoder program will be explained more in the MIDI Decoder Program section.

The controller program is used to control all the mechanical and electronic on the alto saxophone including pressing the saxophone keys and blow the mouthpiece. The controller program takes the data from the MIDI decoder to indicate the status of the data which is on or off and to indicate which musical notes data are sent. Before doing the reading job, the controller will send the off state PWM which is to set the servomotors to the not pressed condition. The controller stores the array program call saxophone fingering program that use to indicate which trigger signal need to be turned on when the specific notes data are coming based on the saxophone fingering chart. The blower program is used to control the mouthpiece blower. The trigger signal of blower program also included in the array program. Each time the trigger signal is sent, the on state PWM also sent to turn the servomotors to the press condition. All the output of the controller program is sent to the controlling circuit.

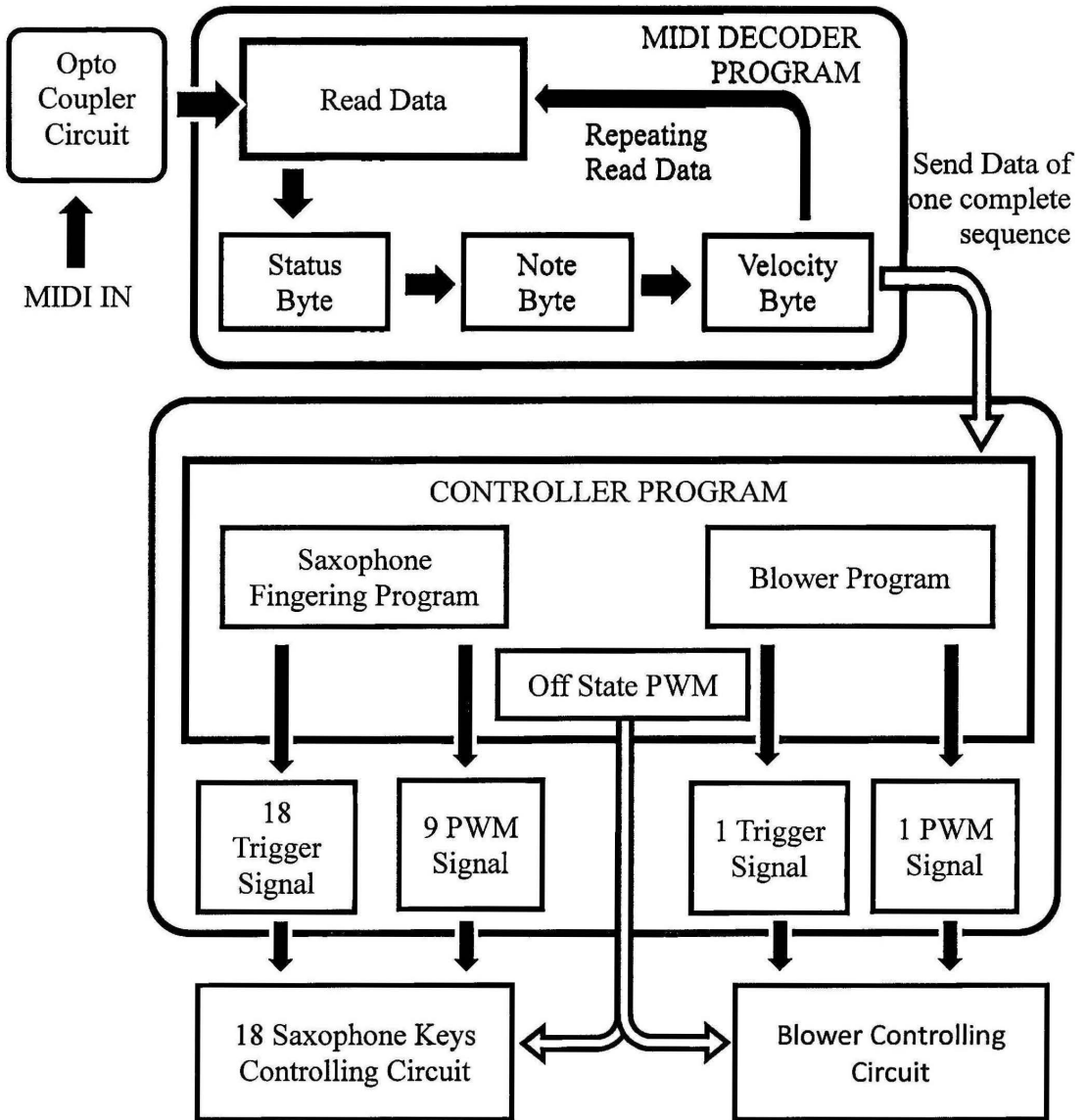


Figure 3.1: Block Diagram of Complete System

3.4 MIDI Decoder Program to Extract MIDI Data

The MIDI decoder program is purpose in read and extracting the MIDI data sent by the computer and sent the data to the controller program. The MIDI decoder are stored in the Arduino Mega 2560 Micro controller board. Figure 3.2 shows the flowchart of the MIDI decoder program. First, the program executes the initial PWM pulse. Then the program starts to receive and read a data byte coming. Then the data are categorized into 3 data type which is data type 0 for the status byte, data type 1 is for the notes byte and the data type 2 is for the velocity byte. If the first byte coming is higher than 128 in decimal value, it will indicate as status byte. Then the data receive is read in the data type 0 case and if the data byte is equal to 144 in decimal, the key press is indicated as high and change the data type into 1. If the value not equal to 144, the data will read as 128 and then the keypress are indicated as low and the data type still change into 1 because the next coming byte is the notes byte.

Then the data type case is closed and the program loop again to receive the next data byte. Now the coming byte is the notes byte. The data type was changed in the data type 0 case. So this time, the data byte is entering the data type 1 case. The byte must be lower than 128 values in decimal (refer to chapter 2.2.1) because the notes byte must be lower than 128. If yes, the key press is indicated as high and change the data type into data type 2. If no, the data type will change into 0 because not following the MIDI protocol. Then the data type case is closed and loop back to receive the coming data byte.

Then, the coming byte is the velocity byte. The data are entered into the case data type 2 because it was changed in the data type 1 case. Then, if the data read is lower than 128, it will proceed to the. If not, it will not proceed to the controller controller program and run the program program. Then the data type are changing back into 0 because the sequence of 3 data byte of MIDI is complete and ready to read again the incoming data byte.

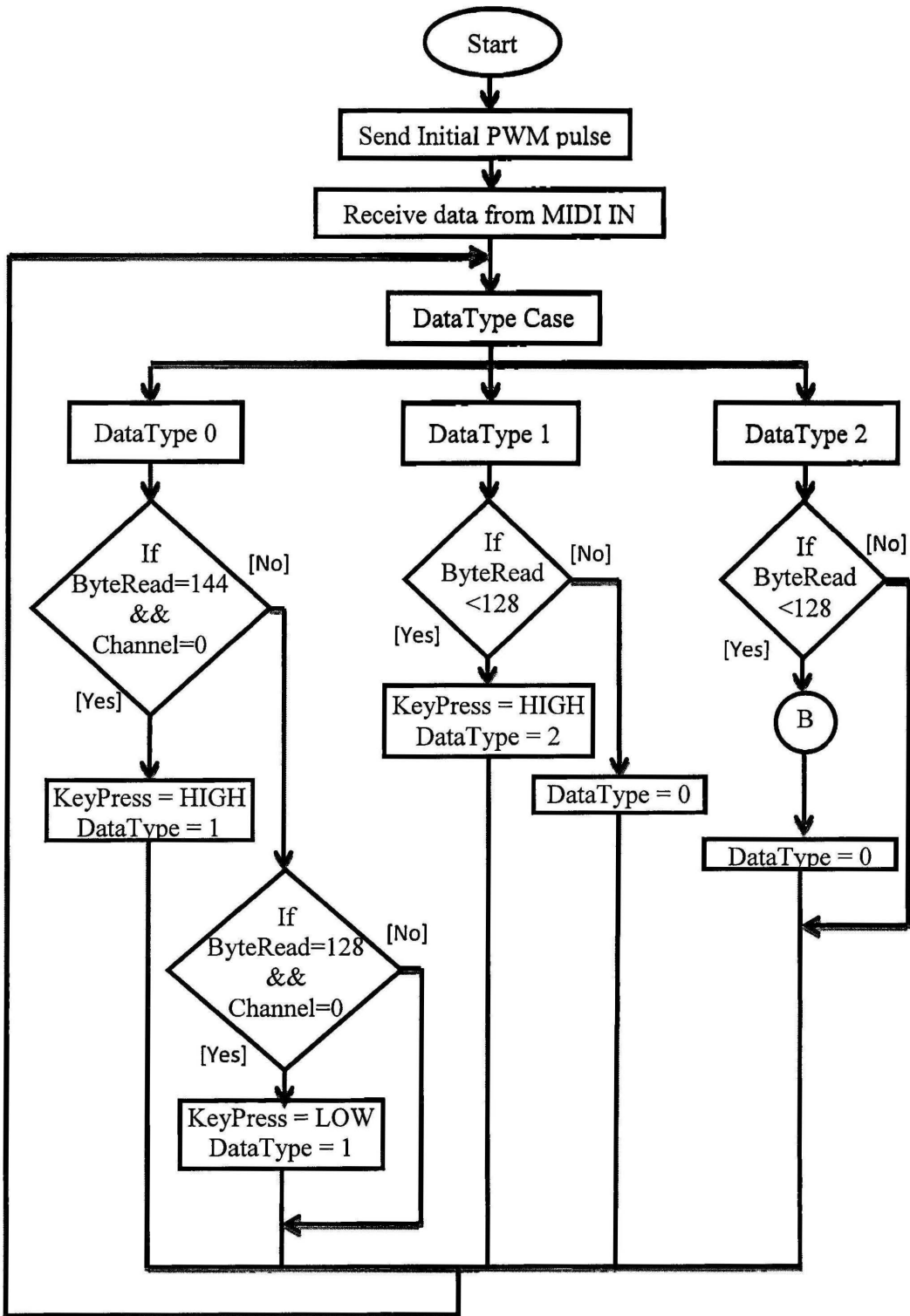


Figure 3.2: MIDI Decoder Program Flowchart

3.5 Controller Program to Control Saxophone Keys

The Alto saxophone has 23 keys. Each note played have different keys need to be pushed. In this project, the keys will be controlled by the servo motors that will turn and push the saxophone keys when the signal are sent. Each of the keys will attach to a servo motor. In the controller programming, each of the notes will be assigned to the key output signal according the saxophone fingering chart. For example, the C# note will be assigned to the servo motors that control the C# note keys on the saxophone based on the saxophone fingering chart. The saxophone has 2 and a half octave. It means the total keys can be played by saxophone is 30 notes. The 30 notes will have different saxophone keys pushed when the notes come in.

Figure 3.3 (a) show the saxophone fingering chart. The number to each of the keys is referred to the output pin number on the Arduino Mega 2560 board. The black mark keys refer to the keys that called as alternative keys. The alternative keys in saxophone are the shortcut keys that represent the combination of other keys. In this project, the alternative keys are not used. The Figure 3.3 (b) shows the example of key combinations for note C₄. The black mark shows the keys that need to be pressed and the blank refer to keys not to be pressed.

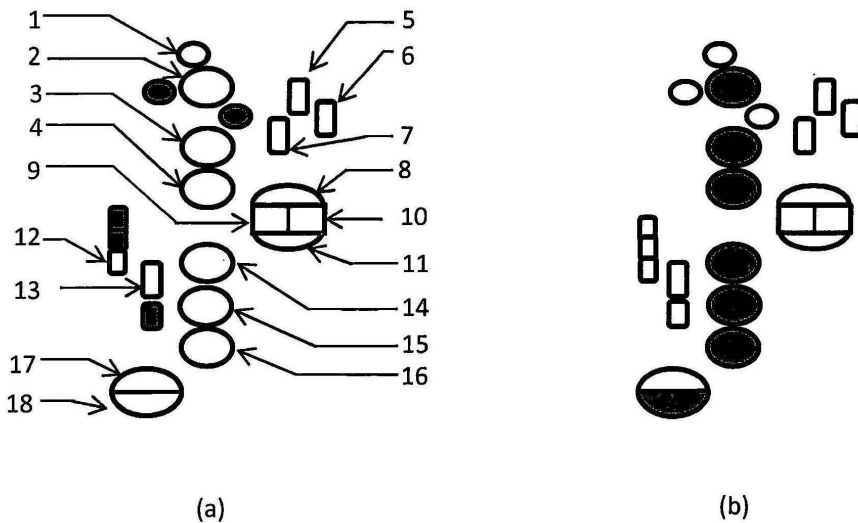


Figure 3.3: Saxophone Fingering Chart

3.5.1 Controller Programming Flowchart

The Arduino Mega 2560 board is used to store and run the controller program. There were two types of output need to be sent out from the Arduino which is the switching signal and the PWM pulse. The data read by the MIDI decoder program is the use of the controller program because the value of the data indicates there are key was pressed and to indicate what musical notes are read. Figure 3.4 shows the flow chart of the controller program.

First, the program starts with comparing the key press and the velocity value. If the key press read in the MIDI decoder program is high and the velocity is equal to 0, the keypress is assigned as low because zero velocity means there were no key press. If no, the program proceeds to the next step which is indicates the key press status again. If the key press is high, then the program will proceed to read the notes data value. If no, the program will proceed to call the array of note low which means the data read are not be able to call any notes array.

Then the process to read the notes data. The note data value must be in range of base note and 30 above. If value not in the range, the program straight forward go to end. If the data value in the range, the data will calculate which note case should be proceed to call the notes array. The calculation is calculated by minus the value of Note data read with the base note value. If the calculation gets the value of zero, the program will go to the end. If not, it will enter the note case. The note case is selected using the value get from the calculation. After selecting the right case, the program proceeds to call the switching array. Each note array where store different data because to indicate which saxophone keys need to be pressed. After that, the switching signal is sent out to the assigned pin on the Arduino.

Then the program will again indicate the key press for the data in pin on pin 13 on the Arduino board. If key press is high. Then the pin will signal as on. Then the program will send the on state PWM pulse. Then the end of the program and continue to the MIDI decoder program.

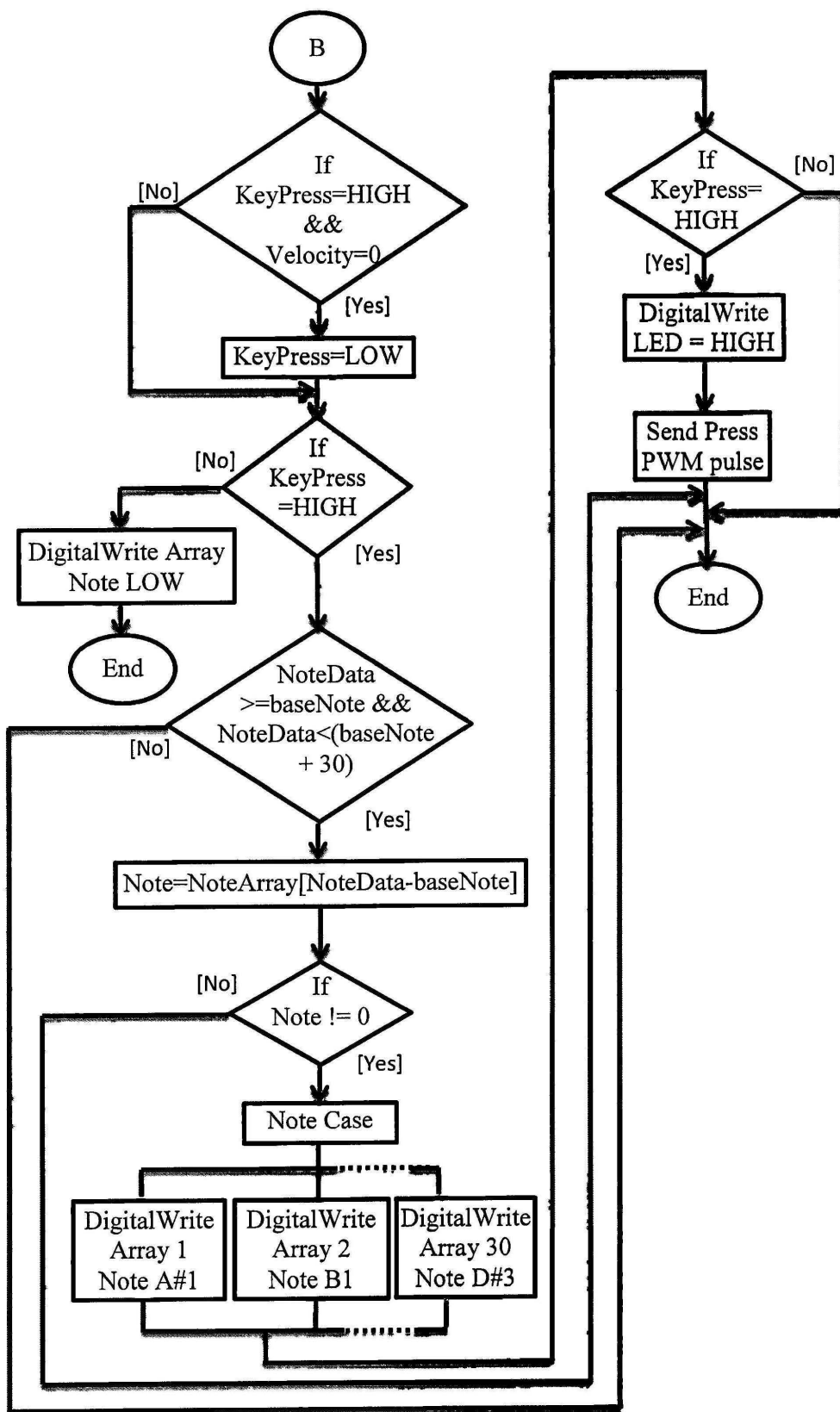


Figure 3.4: Controller Program Flowchart

3.6 Opto Coupler Circuit for MIDI Shield

The opto coupler is used in between the two isolator circuit. In this project, the opto coupler is used in between the MIDI Out Port cable and the communication pin on the Arduino which is the Serial data receiver pin. Figure 3.5 below show the propose circuit of the opto coupler.

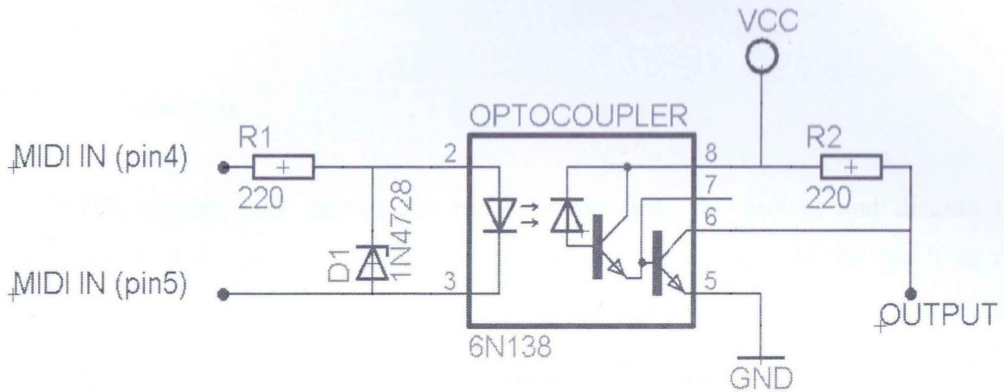


Figure 3.5: Opto Coupler Circuit

CHAPTER 4

RESULT AND DISCUSSION

4.1 Introduction

This chapter will discuss the result obtain from the project and discuss the related result of the project. The result of this project will include the result of the program, output signal that needs to come out and the calculation of the Pulse Width Modulation duty cycle.

4.2 Propagation Delay Time in Arduino Programming

The Arduino Mega 2560 is used as the microcontroller to read data, control the servo movement and switching the pneumatic valve. The programming include in the microcontroller is the MIDI decoder program and controller program. But, there were a problem occur which is the propagation delay time between the beginning of the programming and the end of the programming.

The delay time of the programming is affected by the clocking speed. The higher the clocking speed of the microcontroller the less the delay time of the programming. The clock speed of the Arduino Mega 2560 is 16 MHz. Figure 4.4 shows the delay time in the programming.

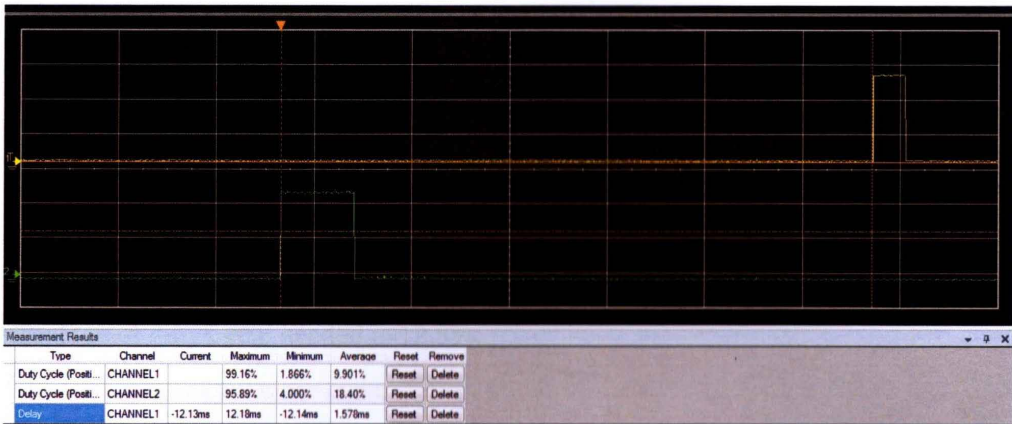


Figure 4.1: Propagation Delay Time in Arduino Mega 2560 Programming

In Figure 4.1, the top signal capture is the result of the off state PWM while the bottom is resulting from the on state PWM. Refer to the Figure 3.6 and Figure 3.7 in Chapter 3, at the beginning of the programming is to send the off state PWM and the end of the program loop sends the on state PWM. So, the delay capture is the propagation delay time of a loop of the programming. This means from the beginning of the program till the end of the program.

The propagation delay time in the programming is about 12.13ms. The value is too high for a microcontroller because the human can detect the delay of the result of the program which is the movement of the servo. For example, when a MIDI data send out from the computer, we can detect the delay time from when the notes come in (piano keys pressed in the computer application) and the servo starts to move to the on state (saxophone keys pressed). But, the propagation delay time of the program were not multiply and not resulting the increasing of the delay time over the real time pass.

4.3 Pulse Width Modulation Duty Cycle

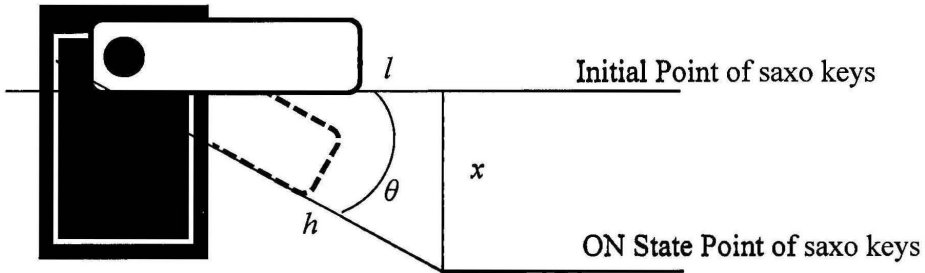
The Pulse Width Modulation (PWM) is used in this project to control the movement and position of the servo motors. The percentage of the duty cycle of the PWM defines how much the angular movement of the servo motors. For example, for a 0° position of the servo motors, the percentage of duty cycle needed is 2.5% and for 180° position, the percentage of duty cycle needed is 12.5%.

On the Alto saxophone, all the keys of the saxophone have a different distance to travel to the completely pressed state. So, the servo need to be defined how much it need to rotate from the initial position to the completely pressed position. All the servo is set to 0° for the initial state (off state), then the servo will move to the key press position (on state) according to the distance need to travel for the completely saxophone key press. The on state PWM needs to be calculated according to the distance of the servo hand need to travel. The calculation involve in this section is the Pythagoras Theorem and trigonometry of the sine function.

$$\text{Pythagoras Theorem,} \quad h = \sqrt{x^2 + l^2}$$

$$\text{Sine Function,} \quad \theta = \sin^{-1} \frac{x}{h}$$

Figure 4.2 shows the diagram of servo motor and the variable involve in calculating the angular distance of the servo motor's hand need to travel to press the saxophone keys. The length of the servo motors fixes to 20mm. The distance of key press is the height between the initial points of the key (not press) and the completely press point (press). The hypotenuse and theta value can be calculated using the Pythagorean Theorem and Sine Function Formula respectively.



l – length of servo hand
 x – distance of key press
 h – hypotenuse value

Figure 4.2: Angular Distance of Hand of Servo Motor Need to Travel

Take $x = 2\text{mm}$ for the first calculation. Find the hypotenuse value.

Using Pythagoras Theorem Formula,

$$h = \sqrt{x^2 + l^2} \quad ; x = 2\text{mm}, l = 20\text{mm}$$

$$h = \sqrt{2^2 + 20^2}$$

$$h = 20.09$$

Then find the theta, θ value using the Sine Function Formula,

$$\theta = \sin^{-1} \frac{x}{h}$$

$$\theta = \sin^{-1} \frac{2}{20.09}$$

$$\theta = 5.7134^\circ$$

Calculate the percentage of the duty cycle,

$$0^\circ = 2.5\%, 180^\circ = 12.5\%$$

$$\frac{\theta}{180} = \frac{\text{Duty Cycle}}{(12.5 - 2.5)}$$

$$\frac{5.7134}{180} = \frac{\text{Duty Cycle}}{10.5}$$

$$0.03174 = \frac{\text{Duty Cycle}}{10.5}$$

$$\text{Duty Cycle} = 0.33327\%$$

The duty cycle value gets above is the increase of duty cycle. To get the actual value, the value must be plus with 2.5 (the initial point).

$$\text{Duty Cycle} + 2.5\% = 2.83327\%$$

The percentage of duty cycle for $x = 2\text{mm}$ is 2.83%

From the result in Table 4.1, the value of the approximate duty cycle can be rounded off to the nearest digit because the value only has small difference. The duty cycle of the distance travelled between 2mm to 6mm, duty cycle value can round off to 3%. The duty cycle of the distance travelled between 7mm to 10mm, the duty cycle value can be rounded off to 4%.

Table 4.1: Result of Calculation of Servo Duty Cycle

| x (mm) | h (mm) | θ (°) | Approximate Duty cycle (%) |
|-------------|-------------|--------------|-------------------------------|
| 2 | 20.09 | 5.71 | 2.83 |
| 3 | 20.22 | 8.53 | 3.00 |
| 4 | 20.39 | 11.31 | 3.16 |
| 5 | 20.62 | 14.03 | 3.32 |
| 6 | 20.88 | 16.69 | 3.47 |
| 7 | 21.19 | 19.28 | 3.62 |
| 8 | 21.54 | 21.80 | 3.77 |
| 9 | 21.93 | 24.23 | 3.91 |
| 10 | 22.36 | 26.57 | 4.05 |

x – Distance between open and close keys

h – Hypotenuse value

θ – Angle of movement

4.4 Result of Controller Program Output

The controller program is used to give the instruction for the switching signals and create on state PWM. The controller will call the array that store in the programming to indicate which 19 switches (18 for saxophone keys and 1 for reed) will be turned on in on state (when there is data coming). Figure 4.3 shows the programming of the controller program. The programming of the controller program is referring to the flow chart in in Figure 3.6 in Chapter 3. The program starts to indicate the key press is on or off (line 3 to 7). If the key press is on, the program will proceed to define what the musical notes coming. First, the rule set to the value of note byte read must be between the value of the base note and 30 notes above the base note (line 9) because the alto saxophone only can play 30 musical notes. The base note is stated to a value of 58 because the first note on the alto saxophone can play is $A\#_3$ which the value of the note in decimal is 58 (refer Table 2.1).

If the note data value read is correct, the program start to calculate which case should be chosen (line 11). Take the example of note C_4 , which the value in decimal is 60.

```
NoteArray [NoteData-baseNote];
      NoteArray [60-58];
      NoteArray [2];
```

Then the program will find the note case number 2 which then recall the switching signal for note C_4 that store in the array. Then the program will send the switching signal to the 19 pin of the Arduino (including servo controlling the reed). There were 30 note case in the programming and each one will recall the notes that store the switching signal in array form. Then the program will send out the on state PWM to move the servo motor from the initial position to the key press position.

```

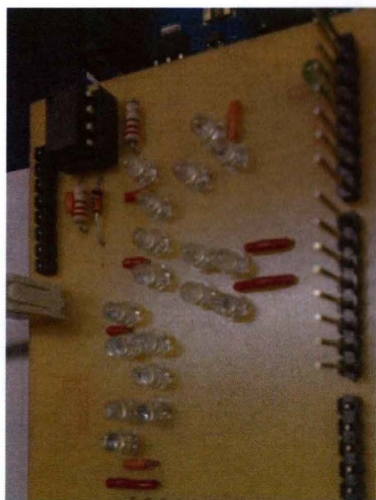
1 void Controller (byte NoteData, byte Velocity, int KeyPress)
2 {
3   if ((KeyPress == HIGH) && (Velocity == 0))
4     {
5       KeyPress = LOW;
6     }
7   if (KeyPress == HIGH)
8     {
9     if (NoteData >= baseNote && NoteData < (baseNote + 30))
10    {
11      byte Note = NoteArray [NoteData - baseNote];
12      if (Note != 0)
13        {
14          switch (Note)
15          {
16            case 1:
17              for (i=30; i<50; i++)
18                {
19                  digitalWrite (i, as1 [i]);
20                }
21              break;
22            }
23          if (KeyPress == HIGH) digitalWrite (LED, HIGH);
24        }
25        digitalWrite (PWMPin , 7);
26        digitalWrite (PWMPin2, 10);
27        digitalWrite (PWMReed, 100);
28      }
29    }
30    else
31      {
32        for (i=30; i<50; i++)
33          {
34            digitalWrite (i, low [i]);
35          }
36      }
37    }

```

Figure 4.3: Programming of Controller Program

If there is no data coming and read by the MIDI decoder program, the controller program were kept recalling the low array which stores the off switch for all saxophone keys and reed.

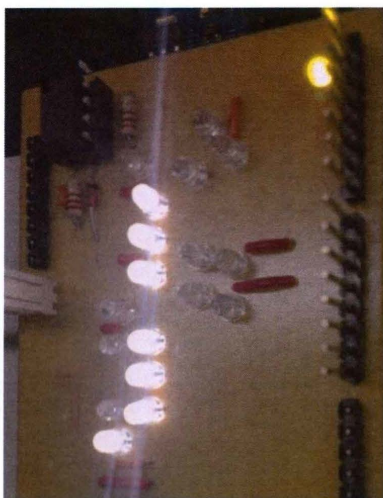
Figure 4.4 shows the example of the notes switching. The LED is used as the indicator. Figure 4.4 (A) shows the off state where there is no data read. Figure 4.4 (B) shows the on state for the note $A\#_3$ and Figure 4.4 (C) shows the on state for the note C_4 .



(A)



(B)



(C)

Figure 4.4: Example of Notes Switching

Figure 4.5 shows the MIDI shield board. The board can be attached to the Arduino Mega 2560 board. The board called MIDI Shield is attached to the Arduino Mega 2560 Micro controller board. The board consists of the opto coupler circuit and LED indicator. There were also the pin ports for jumpers for transferring the signal and PWM to the controlling circuit.

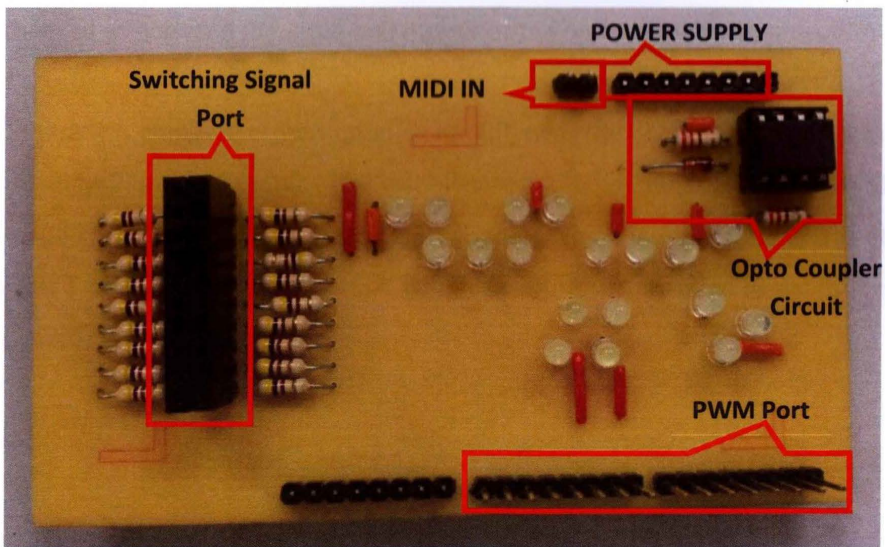


Figure 4.5: MIDI Shield Board

4.5 Propagation Delay Time at Opto-Coupler Circuit

The opto coupler is functioning to transfer the electrical signals between two isolated circuits by using light. It used the same concept with the relay but more efficient in high speed in transferring the signal. But, there were also a problem occur in the operation of the opto coupler which is the propagation delay time. The delay time can be checked in between the input and output of the opto coupler using oscilloscope.

Figure 4.6 shows the opto coupler circuit with the oscilloscope attaches to the input and output of the circuit. The oscilloscope use is the Agilent USB Modular Oscilloscope U2701a. The input of the opto coupler which is on the pin 2 was attached to the probe 1 and the output of the opto coupler circuit was attached to the probe which is on the pin 6.

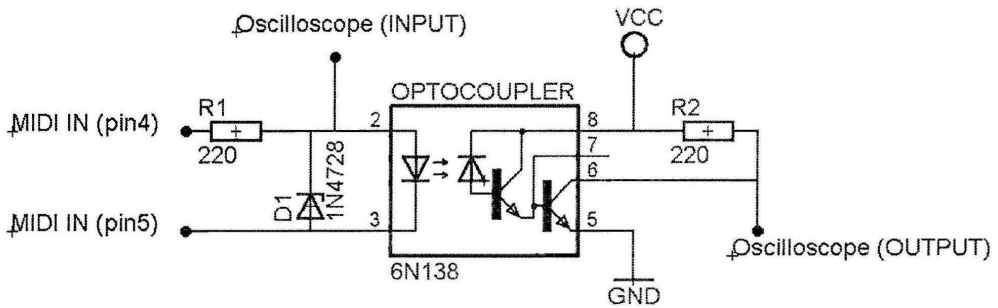


Figure 4.6: Opto Coupler Circuit Attach to Oscilloscope

Figure 4.7 shows the example data capture by the oscilloscope which is the C₄ notes pressed. The top data in the figure is the result from the probe 1 and bottom data is the result from probe 2.

Figure 4.8 shows the propagation delay time at the beginning of the data. The top data capture is from the probe 1 and the bottom data is from the probe 2. The propagation delay time in the opto coupler circuit is about 2.652μs. From the data sheet of the opto coupler (6N139), the typical propagation time is 2.0μs.

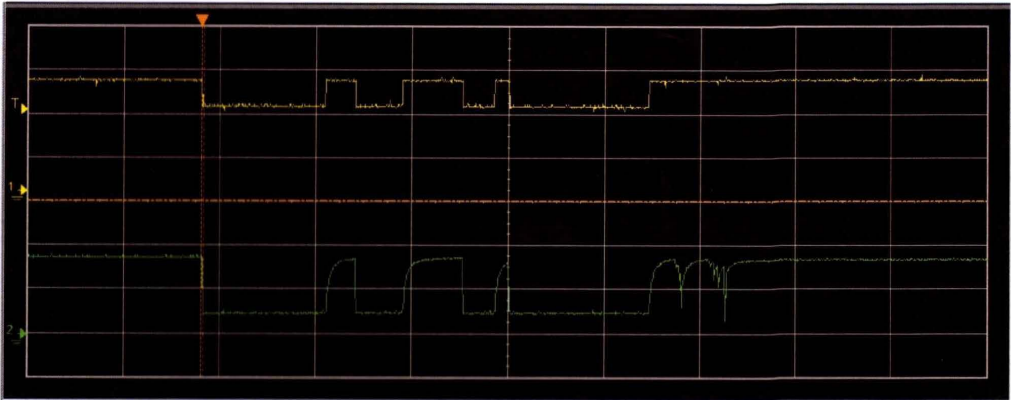


Figure 4.7: Example of C₄ Notes pressed

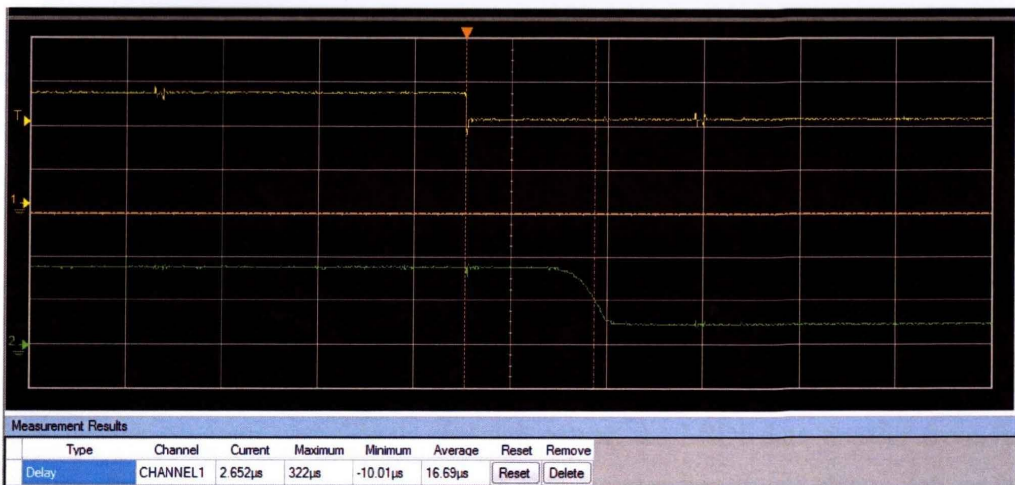


Figure 4.8: Propagation Delay between Input and Output of the Opto Coupler

The value of the propagation delay time at the opto coupler can be accepted because the propagation delay time is too small and not affected the data transfer between the MIDI output port and the Arduino receiver pin.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This chapter will explain about the conclusion and recommendation of the whole project paper. This chapter will discuss mainly about the conclusion of the whole project. Other than that, this chapter also will briefly explain the recommendation for the future development.

5.2 Conclusion

From the result and discussion obtain from the chapter 4, as the conclusion, the MIDI decoder program was able to read and extract the MIDI data transfer from the computer through the USB-MIDI cable. The opto coupler circuit was able to deliver the right message from the computer although there was a little propagation delay time. The controller program also succeeds to deliver the right output of switching signal and pulse width modulation. The LED that use as the indicator on the MIDI shield board indicates that the switching signal outputs was correct according to the data store in the array in the programming. Although the result of the delay time in the overall programming was quite large, but the delay can be ignored because not cause the multiple delay time.

5.3 Recommendation

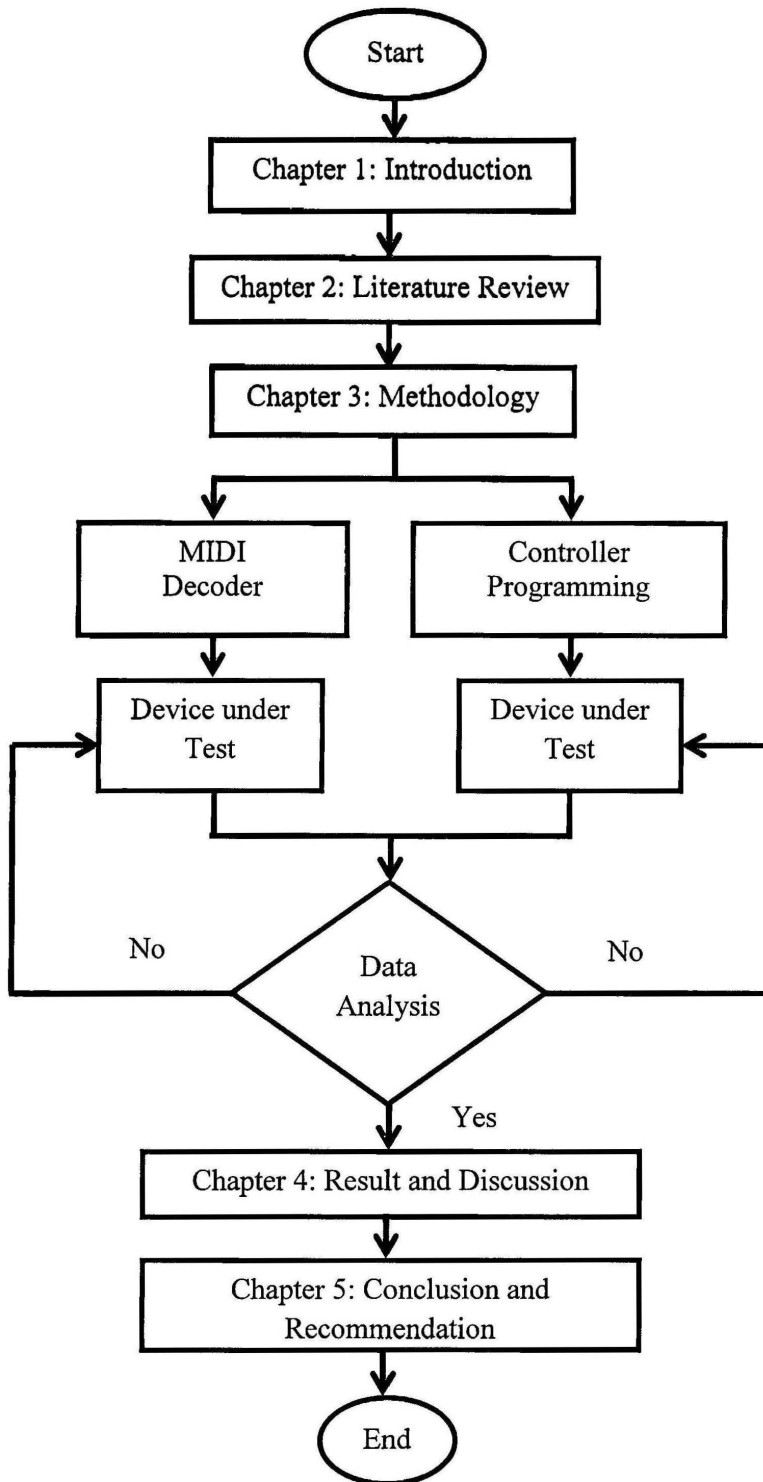
For the recommendation, the programming in the Arduino Mega 2560 can be minimized by using the parallel programming technique which is more efficient and reduce the delay time in the programming. The other option is more advanced which building the operating system program for the MIDI decoder and the controller program.

Another suggestion is using the electronic actuator such as plunger because the device only used a switching signal rather than servo motors that need the switching signal and the pulse width modulation signal. This way can reduce the overall project delay time.

REFERENCES

- [1] Jerry, W. (1999). U.S. Patent No. 6,121,536. Armonk, NY: U.S.
- [2] D. M. Huber, *The MIDI Manual*, Third Edition: A Guide To MIDI in the Project Studio, ISBN 0240807987, Focal Press, 2007.
- [3] Kai Y, Xi Z, (2012). *MIDI-Lab, a Powerful Visual Basic Program for Creating MIDI Music*, Vol. 3, No. 4.
- [4] Rikip G, Ivan I, (2011). *MIDI Conversion to Musical Notation*, pp 97-99.
- [5] Mike K, Geert K, (1999). *USB MIDI Converter*. Universal Serial Bus Device Class Definition for MIDI Devices (pp.12).
- [6] Ben M, Craig R, (2012). *Acoustical Properties of the Alto Saxophone*.
- [7] Alto Saxophone. Retrieved December 16, 2012 from Wikipedia site: http://en.wikipedia.org/wiki/Alto_saxophone
- [8] Larry T. The Instrument. In *The Art of Saxophone Playing* (page 13). Summy-Birchard Music in New Jersey, US.
- [9] Parts of the Saxophone. Retrieved December 16, 2012 from About.com site: <http://musiced.about.com/od/lessonsandtips/a/saxparts.htm>
- [10] Arduino. Retrieved December 19, 2012 from Wikipedia site: <http://en.wikipedia.org/wiki/Arduino>
- [11] Arduino Mega 2560. Retrieved December 19, 2012 from Seed Studio site: http://www.seeedstudio.com/wiki/Arduino_Mega_2560
- [12] Arduino Mega 2560. Retrieved December 19, 2012 from Arduino site: <http://arduino.cc/en/Main/ArduinoBoardMega2560>
- [13] Amanpreet K, Amandeep K, (2012). *An Approach for Designing A Universal Asynchronous Receiver Trasmmitter (UART)*. Vol. 2 (Issue 3), pp.2305-2311.

APPENDIX A
Project Flowchart



APPENDIX B

Saxophone Fingering Chart

A4/Bb B C C#/Db

D D#/Eb E F

F#/Gb G G#/Ab A

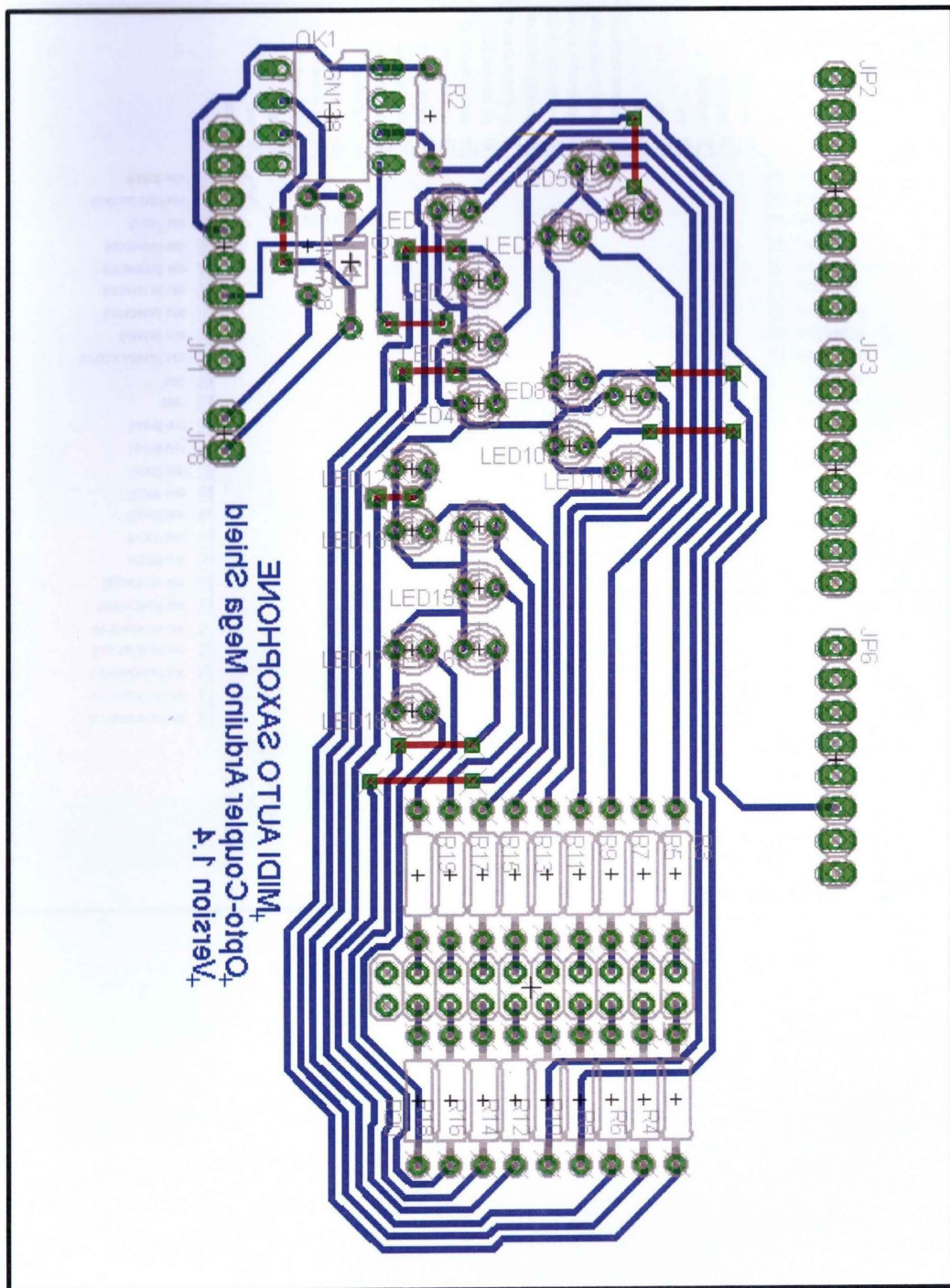
Staff 1: Chord diagrams for B \flat , B, C, and C \sharp m. Each diagram shows a guitar fretboard with black dots for fretted notes and open circles for open strings. The B \flat diagram has notes on strings 1, 2, 3, 4, 5, and 6. The B diagram has notes on strings 1, 2, 3, 4, 5, and 6. The C diagram has notes on strings 1, 2, 3, 4, 5, and 6. The C \sharp m diagram has notes on strings 1, 2, 3, 4, 5, and 6.

Staff 2: Chord diagrams for D, D \sharp /E \flat , E, and F. Each diagram shows a guitar fretboard with black dots for fretted notes and open circles for open strings. The D diagram has notes on strings 1, 2, 3, 4, 5, and 6. The D \sharp /E \flat diagram has notes on strings 1, 2, 3, 4, 5, and 6. The E diagram has notes on strings 1, 2, 3, 4, 5, and 6. The F diagram has notes on strings 1, 2, 3, 4, 5, and 6.

Staff 3: Chord diagrams for F \sharp /G \flat , G, G \sharp /A \flat , and A. Each diagram shows a guitar fretboard with black dots for fretted notes and open circles for open strings. The F \sharp /G \flat diagram has notes on strings 1, 2, 3, 4, 5, and 6. The G diagram has notes on strings 1, 2, 3, 4, 5, and 6. The G \sharp /A \flat diagram has notes on strings 1, 2, 3, 4, 5, and 6. The A diagram has notes on strings 1, 2, 3, 4, 5, and 6.

APPENDIX C

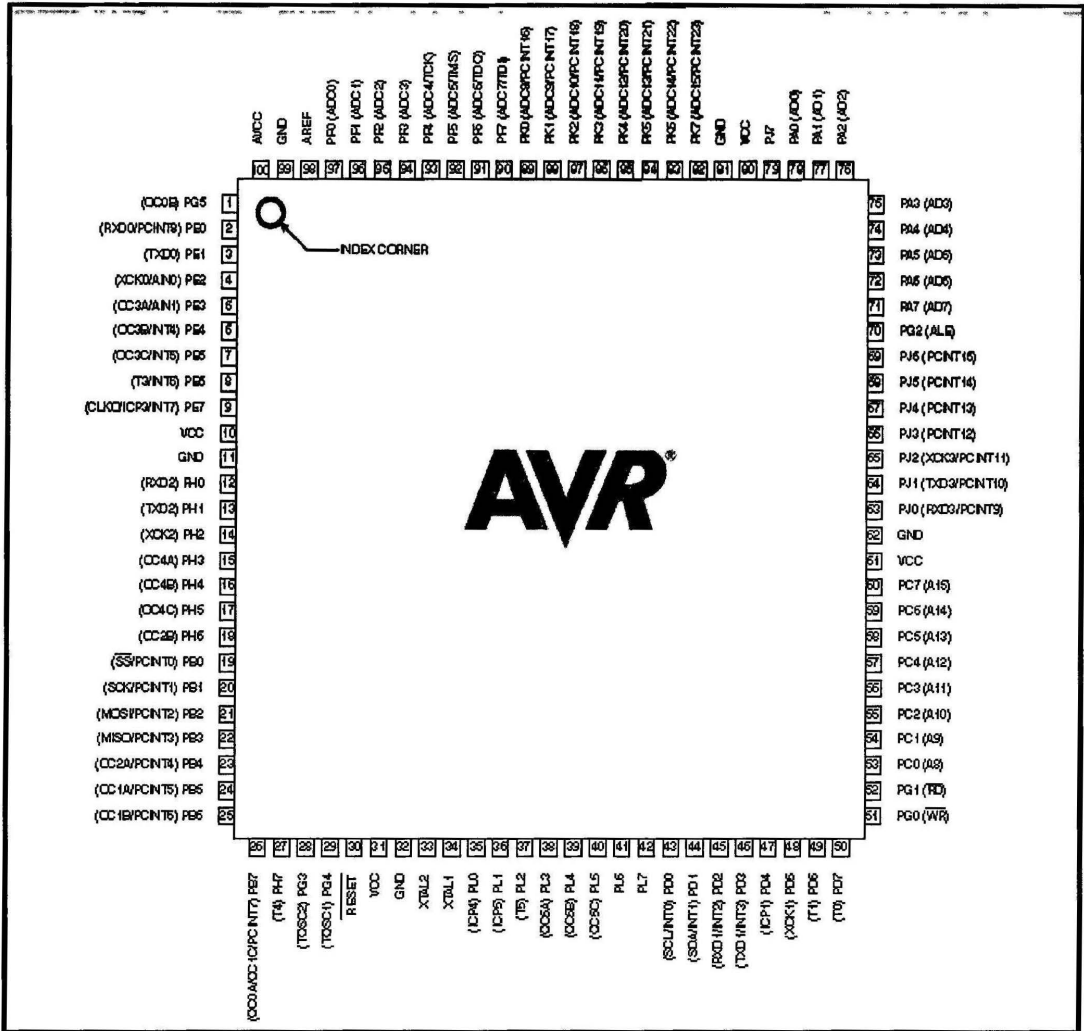
PCB Drawing of the MIDI Shield Board



blieir spgM onIubrtA relqu-C-otiq₊
A. l' noIzIeV₊
EIMIQOXA2 OTUA IDIM₊

APPENDIX D

ATmega2560 Microcontroller Pin Configuration




```

pinMode(36,OUTPUT);
pinMode(37,OUTPUT);
pinMode(38,OUTPUT);
pinMode(39,OUTPUT);
pinMode(40,OUTPUT);
pinMode(41,OUTPUT);
pinMode(42,OUTPUT);
pinMode(43,OUTPUT);
pinMode(44,OUTPUT);
pinMode(45,OUTPUT);
pinMode(46,OUTPUT);
pinMode(47,OUTPUT);
pinMode(48,OUTPUT);

myservo.attach(2);
myservo1.attach(3);
myservoReed.attach(4);

DataType = 0;

Serial1.begin(31250);
digitalWrite(LED,LOW);
}

void loop ()
{
myservo8.write(10);

if (Serial1.available() > 0)
{
    DataRead = Serial1.read();
    digitalWrite(LED,LOW);
    switch (DataType)
    {
        case 0:
            if (DataRead == (144 | channel))
            {
                KeyPress = HIGH;
                DataType = 1;
            }
            if (DataRead == (128 | channel))
            {
                KeyPress = LOW;
                DataType = 1;
            }
            break;

        case 1:
            if(DataRead < 128)

```

```

        {
            NoteData = DataRead;
            DataType = 2;
        }
        else
        {
            DataType = 0;
        }
        break;

        case 2:
        if(DataRead < 128)
        {
            Controller(NoteData, DataRead, KeyPress);
        }
        DataType = 0;
        break;
    }
}
}

```

```

void Controller(byte NoteData, byte Velocity, int KeyPress)
{
    if ((KeyPress == HIGH) && (Velocity == 0))
    {
        KeyPress = LOW;
    }
    if (KeyPress == HIGH)
    {
        if(NoteData>=baseNote && NoteData<(baseNote + 30))
        {
            byte Note = NoteArray[NoteData-baseNote];
            if(Note != 0)
            {
                switch(Note)
                {
                    case 1:
                    for(i=30; i<50; i++)
                    {
                        digitalWrite(i, as1 [i]);
                    }
                    break;

                    case 2:
                    for(i=30; i<50; i++)
                    {
                        digitalWrite(i, b1 [i]);
                    }
                }
            }
        }
    }
}

```

```
break;
```

```
case 3:  
for(i=30; i<50; i++)  
{  
    digitalWrite(i, c1 [i]);  
}  
break;
```

```
case 4:  
for(i=30; i<50; i++)  
{  
    digitalWrite(i, cs1 [i]);  
}  
break;
```

```
case 5:  
for(i=30; i<50; i++)  
{  
    digitalWrite(i, d1 [i]);  
}  
break;
```

```
case 6:  
for(i=30; i<50; i++)  
{  
    digitalWrite(i, ds1 [i]);  
}  
break;
```

```
case 7:  
for(i=30; i<50; i++)  
{  
    digitalWrite(i, e1 [i]);  
}  
break;
```

```
case 8:  
for(i=30; i<50; i++)  
{  
    digitalWrite(i, f1 [i]);  
}  
break;
```

```
case 9:  
for(i=30; i<50; i++)  
{
```

```
        digitalWrite(i, fs1 [i]);
    }
    break;

    case 10:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, g1 [i]);
    }
    break;

    case 11:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, gs1 [i]);
    }
    break;

    case 12:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, a1 [i]);
    }
    break;

    case 13:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, as2 [i]);
    }
    break;

    case 14:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, b2 [i]);
    }
    break;

    case 15:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, c2 [i]);
    }
    break;

    case 16:
    for(i=30; i<50; i++)
    {
```

```
        digitalWrite(i, cs2 [i]);
    }
    break;

    case 17:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, d2 [i]);
    }
    break;

    case 18:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, ds2 [i]);
    }
    break;

    case 19:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, e2 [i]);
    }
    break;

    case 20:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, f2 [i]);
    }
    break;

    case 21:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, fs2 [i]);
    }
    break;

    case 22:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, g2 [i]);
    }
    break;

    case 23:
    for(i=30; i<50; i++)
    {
```

```
        digitalWrite(i, gs2 [i]);
    }
    break;

    case 24:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, a2 [i]);
    }
    break;

    case 25:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, as3 [i]);
    }
    break;

    case 26:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, b3 [i]);
    }
    break;

    case 27:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, c3 [i]);
    }
    break;

    case 28:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, cs3 [i]);
    }
    break;

    case 29:
    for(i=30; i<50; i++)
    {
        digitalWrite(i, d3 [i]);
    }
    break;

    case 30:
    for(i=30; i<50; i++)
    {
```



```
                digitalWrite(i, ds3 [i]);
            }
            break;

        }
        if(KeyPress == HIGH)
            digitalWrite(LED, HIGH);
    }
    myservo.write(45);
    myservo1.write(45);
    myservoReed.write(80);
}
}
else
{
    for(i=30; i<50; i++)
    {
        digitalWrite(i, low [i]);
    }
}
}
```