DESIGN OF M iKLUNG

ANIS HAZWANA BT MOHAMAD WAZIR

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 FOR AUTO ANGKLUNG**

SESI PENGAJIAN: 2012/2013

Saya **ANIS HAZWANA BT MOHAMAD WAZIR (871227-08-5832)**
(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:

24, JALAN KERAPU
TAMAN SERI TENGGARA
34200 PARIT BUNTAR
PERAK

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).

EXAMINER'S APPROVAL DOCUMENT

I certify that thesis entitled 'Design of MIDI Decoder for Auto Angklung Orchestra' is written by Anis Hazwana Bt Mohamad Wazir with matric number FB09063. I have examined the final copy of this thesis and in my opinion, it is fully adequate in terms of language standard, and report formatting requirement for the award of the degree of Bachelor in Mechatronic Engineering. I herewith recommend that it be accepted in fulfilment of the requirements for the degree of Bachelor Engineering in Mechatronic Engineering.

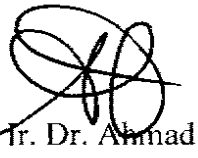
Signature : .....

Name of External Examiner : Mr Ismail bin Mohd Khairuddin

Institution : UNIVERSITI MALAYSIA PAHANG

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 Mohd Zain

Position : Professor

Date : 11 July 2013

STUDENT'S DECLARATION

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

Signature : .....

Name : Anis Hazwana Bt Mohamad Wazir

ID Number : FB09063

Date :

Dedicated to my beloved father, Mohamad Wazir Bin Abu Bakar,
mother, NoorHayati bt Zaini, brothers and sisters, friends
and my supervisor, Prof. Ir. Dr. Ahmad Faizal Mohd. Zain.

ACKNOWLEDGEMENT

Alhamdulillah, praise to Allah, the Most Gracious and the Most Merciful. Alhamdulillah, all praises to Allah for the strengths and His blessing in completing this thesis. Special appreciation goes to my supervisor, Prof Ir. Dr Ahmad Faizal bin Mohd Zain, for his germinal ideas, invaluable guidance, continuous encouragement and constant support in making this project possible. He has always impressed me with his outstanding professional conduct. I appreciate his consistent support from the start. I am truly grateful for his tolerance of my naïve mistakes, and his commitment to my future career.

I would like to express my appreciation to Dean, Faculty of Manufacturing Engineering, Prof. Madya. Dr. Wan Azhar bin Wan Yusuff and all lecturers for their support and help towards my Degree affairs. My acknowledgement also goes to all technicians and office staff of Faculty of Manufacturing Engineering for their co-operations.

My sincere thanks go to my Innovate teams for Innovate Malaysia Competition and friends, for their excellent co-operation, inspirations, supports and made my stay at UMP pleasant and unforgettable. Thanks for the friendship and memories.

Last but not least, my deepest gratitude goes to my beloved parents, Mr Mohamad Wazir bin Abu Bakar and Mrs. NoorHayati bt Zaini and also to my brothers and sister for their endless love, prayers and encouragement. Thank you very much.

ABSTRACT

This project describes a controller device, called **MIDI** (Music Instrument Digital interface) to control an Angklung (Traditional Music Instrument) with 2 Octaves diatonic automatically. MIDI device or MIDI files will generate MIDI data which is decoded into music synthesis commands to the electric motor (DC Motor). Angklung will be control from MIDI Keyboard or Computer as interface. This project will show how far the accuracy of the way to play Angklungs between human and autonomous technology.

ABSTRAK

Projek ini menerangkan alat kawalan, yang dipanggil MIDI (Musical Instrument Digital Interface) untuk mengawal Angklung (Alat Muzik Tradisional) dengan 2 Oktaf diatonic secara automatik. Peranti MIDI atau fail MIDI akan menjana data MIDI yang dinyahkod ke dalam arahan sintesis muzik kepada motor elektrik (DC Motor). Angklung akan dikawal dari keyboard MIDI atau komputer sebagai antara muka. Projek ini akan menunjukkan sejauh mana ketepatan cara untuk bermain angklung antara manusia dan teknologi autonomi.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

LIST OF ABBREVIATIONS

MIDI	Musical Instrument Digital Interface
SPI	Serial Peripheral Interface
SSI	Synchronous Serial Interface
PC	Personal Computer
USB	Universal Serial Bus
SCLK	Serial Clock
MOSI	Master Output Slave Input
MISO	Master Input Slave Output
SS	Slave Selects
TX	Transmitter
RX	Receiver
GUI	Graphical User Interface
DAC	Digital Analog Converter
PWM	Pulse Width Modulation

CHAPTER 1

INTRODUCTION AND GENERAL INFORMATION

1.1 INTRODUCTION

This chapter will briefly explain about the introduction of this project task. The introduction is general information regarding to the topic that will be discuss in this project. This chapter consist of project background, problem statement, objectives, scope and significant of the project.

1.2 BACKGROUND

Many traditional musical instruments have been modified according to the latest trends from an instrument which automatically controlled. For this project, Angklung will be implemented to produce an automatic control via MIDI (Musical Instrument Digital Interface) Sequencer.

Angklung is a musical instrument from Indonesia. This musical instrument is made from bamboo tube in which the sound is generated from air resonance due to the vibration at the internode of bamboo tube. It consists of two bamboo tube in a bamboo frame like shown in figure 1.1. Angklung sound is produced with a frequency that represent a particular tone by the size of internode of bamboo tube. Therefore, an Angklung musical instrument usually is a series of several section of bamboo internode to produce a variety of tones that covers the notes in a song.

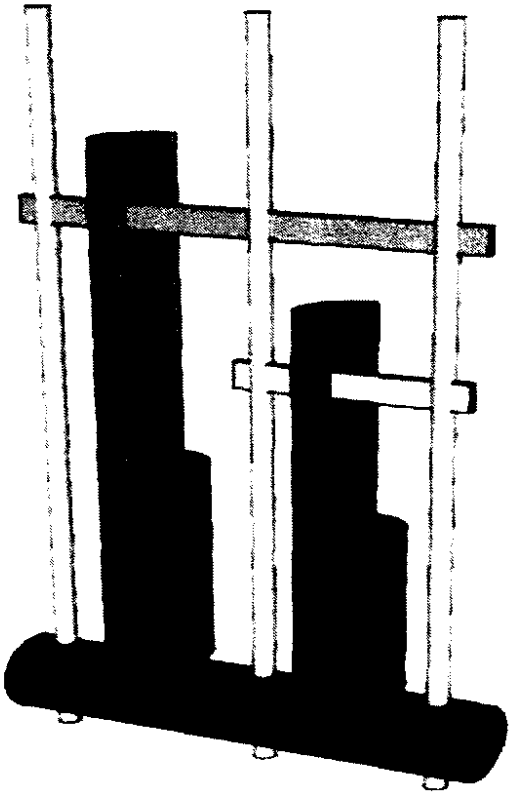


Figure 1.1: Angklung

An Angklung that represent a note usually consist of two section of bamboo internode of different size in length but have the same kind of tone. The length is designed to determine the high and low tones. Thus, in general, an Angklung can produce sound of a note with a combination of high and low tone frequencies.

Angklung tones usually correspond to either pentatonic or diatonic musical scales. The pentatonic scale consists of five notes per octaves, whereas the diatonic scale consists of seven notes in one octave. Therefore, the number of Angklung corresponds to the number of notes, both in diatonic and pentatonic scales, which are required to play a song. Generally the notes required are more than one octave. The set of angklung use in this project is shown in figure 1.2.

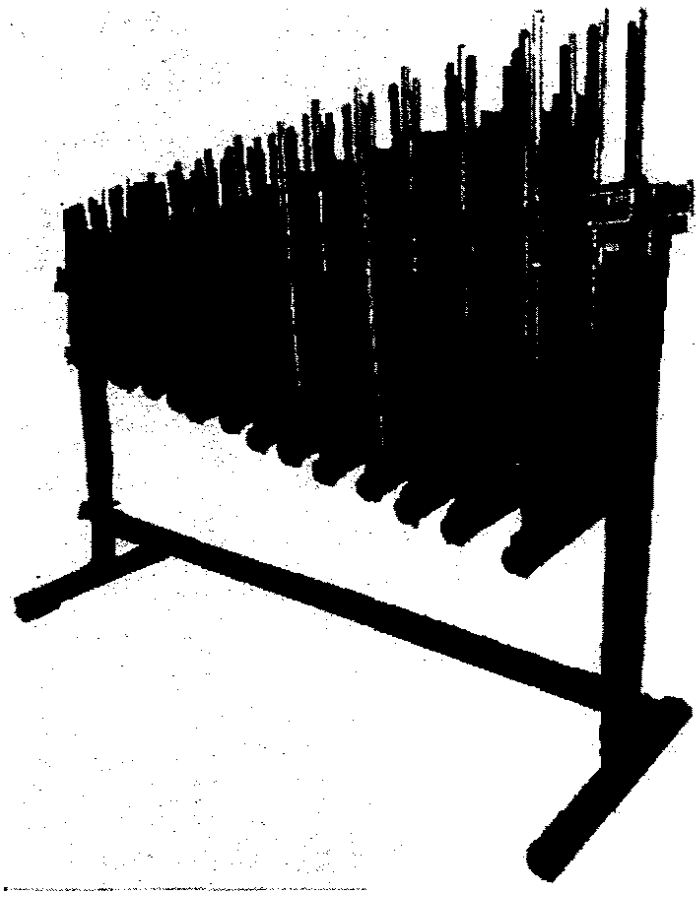


Figure 1.2: Set of Angklung [2]

The notes sound will be produced when shaken or vibrated. When shake the Angklung, the vibration of the bamboo against the base produces a pitch. Each instrument makes only one pitch, to make complete melody it takes many single Angklungs to be use. In this project, Angklung 2 and half octaves will be use, means needs 18 bamboos. Unfortunately, playing Angklung by a single player cannot produce musical sound perfectly, especially to play three notes at the same time (chord). In order to play a song perfectly, more than a person or a group of Angklung players are required. Therefore, it is required to invent a device for playing an Angklung musical instrument automatically.

The Angklung will be automatically controlled via MIDI Sequencer. A MIDI sequencer is in essence a MIDI music player program.

The sequencer created for this project is a program that runs on a Windows operating system and piano keyboard. Its function is to read Standard MIDI Files and parse the relevant information for sending to the MIDI decoder.

MIDI is a protocol which sends a series of message like "note on", "note off", "note/pitch" and many more.

The key is pushed down as shown in figure 1.3 and it produces sound in musical note (which continues to sound while the musician continues holding down the key). This single gesture is known as a Note-On in MIDI terminology. In figure 1.4, the key is release. This stops the musical note from sounding. This single gesture is known as a Note Off to MIDI.

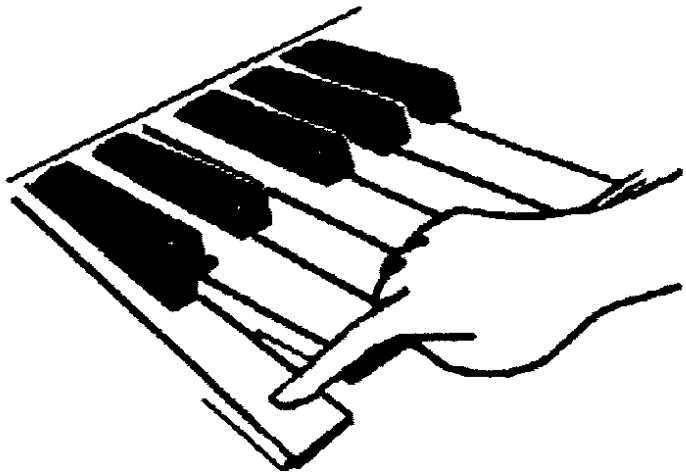


Figure 1.3: The key is pushes down (and hold down) on a keyboard. [3]

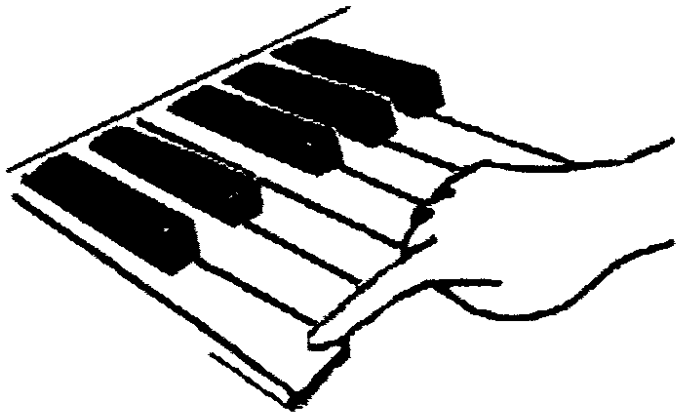


Figure 1.4: The key is releases (holding down) on a keyboard. [3]

In MIDI message contains a start bit, 8 data bits (1 Byte). This message comprised of two components that is "commands" and "data" bytes. The command byte tells what types of message are being send to the MIDI instrument and data bytes functions to store the actual data. Note on message consist of two piece information which is called "note" and "velocity".

This project, Design a MIDI decoder for auto Angklung is to develop 2 and half octave Angklungs that can play automatically without controlled by humans. MIDI sequencer will be decoded to send series data consist of channel message and velocity message. After decoding this message, it will be send to motor driver to control the DC motor to shake the Angklung follows from the MIDI messages. This project also will show how far the accuracy of the way to play Angklungs between human and autonomous technology.

1.3 PROBLEM STATEMENT

The idea of using automatic control for musical instruments it not new. Nowadays, many musical instruments have implemented to control automatically, for example, automatic playing violin, anonymous playing guitar and etc.

For this project, Angklung will be used to be control automatically. Actually, Angklung also have already implemented in automatically control, for example in Indonesia, their student have develop angklung-playing system under the names "KlungBot" and "Klungto Mobi". Innovation of these two instruments is just being preinstalled to the system using parallel communication and the songs that Angklung want to play automatically needs to be programmed in the memory of the microcontroller.

In this project, the difference between the previous inventions is using MIDI protocol to control the automatic Angklung. There are some problems need to be addressed to ensure that the objective is achieved successfully. The problems that need to be taken into account are as, extracting the MIDI message from a MIDI file, controlling an Angklung automatically using MIDI message and control it in real time.

1.4 OBJECTIVES

The objective of this project is to design and develop a MIDI decoder to automatically control an Angklung. Some of the specific aims of this project are:

i. To decode MIDI message
   Decode raw MIDI message from MIDI Keyboard or PC Based interface to microcontroller.

ii. To analyze transmission efficiency MIDI signal

iii. To analyze latency of human ear with produced sound
   MIDI Signal out from microcontroller will be measured, and analysis the delay whether this delay affect human ear latency.

1.5 SCOPE OF STUDY

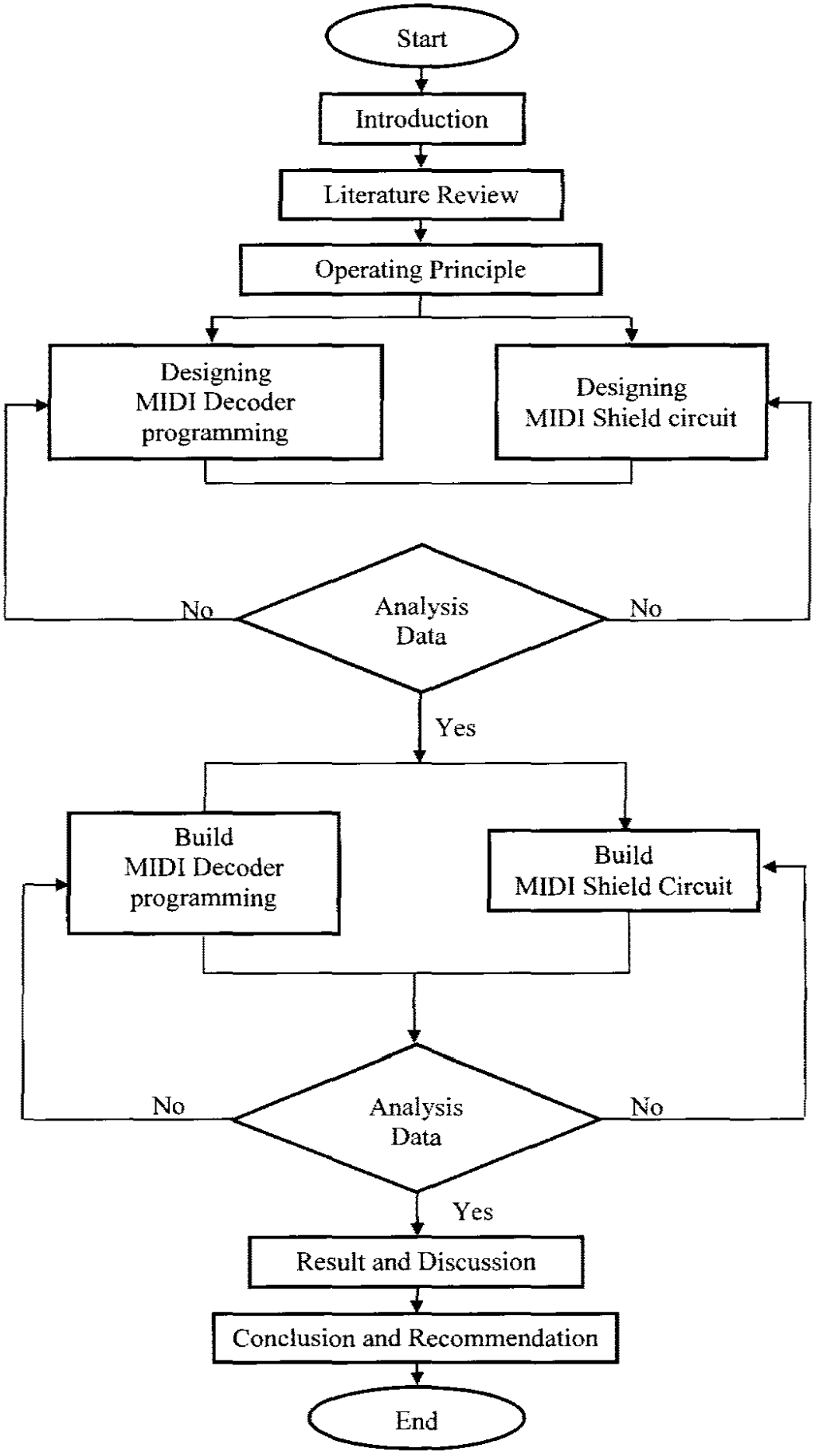
This project will comprise of several stages such as:

i. Learning and interfacing MIDI message Input and Output
ii. Program to decode a MIDI message
iii. Design MIDI sequence in graphical method
iv. Design extra interface circuitry
v. Performance evaluation

1.6 EXPECTED PERFORMANCE

Expected performance is to overcome the human limitation in playing Angklung musical instruments. MIDI message is decoded and signal data will be sending to motor driver to generate DC motor to shake the Angklung. Angklung is controlled from MIDI keyboard, Computer or Android as interface.

1.7 PROJECT FLOW CHART



1.8 THESIS OUTLINE

Chapter 1 outline briefly explains about the introduction of the project task. This chapter consists of Design a MIDI Decoder for auto angklung. Chapter 2 outline previous method of automatic Angklung controller, focusing on the use of MIDI sequencer, as well as the fundamental of MIDI, decoding MIDI, programming method and the Angklung. Chapter 3 discusses the method that use in this project and the entire working algorithm to the MIDI controller. Chapter 4 talks about the experimental result and performance analysis from decoding the MIDI sequencer. A summary of this project presented herein and along with of possible future work contained in Chapter 5. Finally, Appendix A contains the sketch of designing this project.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter will briefly explain about related to MIDI message, serial communication and musical instrument Angklung. The sources are taking from the journals, articles and books. Literature reviews helping in order to provide important information regarding previous researches which related to this project.

2.2 HISTORICAL DEVELOPMENT OF MIDI

MIDI is short for Musical Instrument Digital Interface. The creation of MIDI in 1983 is closely tied to development of music synthesizers, but it has spawned the whole industries of interactivity far beyond the dream of 1983.(MIDI Manufacturer, 1995)

2.2.1 Musical Synthesizers: In the Beginning

Electronic musical instruments had been around in some form since the late nineteenth century. The *Telharmonium* and the *Singing Telegraph* date back to the beginnings of electricity itself while throughout the first half of the twentieth century, electronic musical contraptions were quite the rage in Europe, from the French *Ondes-Martenot* to the German *Pianorad,* to the Russian *Theremin*. In figure

2.1, show that the telharmonium, the first electronic musical instrument develop in 1897. (MIDI Manufacturer, 1995)

The word 'Synthesizer' didn't arrive on the scene until the 1950s with the RCA Synthesizer I and II, but it wasn't long before these room-sized pieces of engineering had been, themselves, 'synthesized' down into more acceptable components and indeed 'modules' thanks to the pioneering work of visionaries like Dr Robert Moog, Don Buchla, Haorld Bode, Pete Zinovieff, and Dave Cockerell. (MIDI Manufacturer, 1995)

Moog is generally, and appropriately, credited for taking the synthesizer out of the university laboratory and putting it in the hands of musicians. Certainly from the time of Walter Carlos' ground-breaking *Switched on Bach* recording (1968) to the release of the *Mini Moog* (1970) both musicians and the music-buying public became enamored – if not frankly dazzled – by the sonic possibilities now seemingly on the musical horizon.

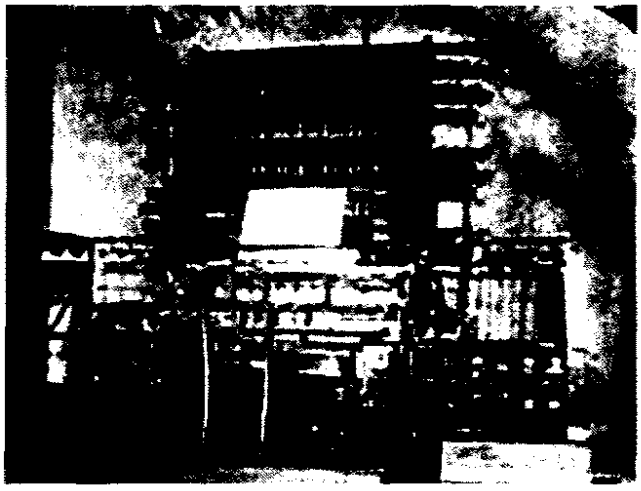


Figure 2.1: *Telharmonium [3]*

As it turned out it was a false dawn. The synthesizers of the 1970s might have been unrestricted sonically but in terms of playability, stability, polyphony, and compatibility they were still very limited indeed.

Early integrated circuits-based synthesizers from Moog, ARP, and EMS opened the door but it was the arrival of Japanese companies like Korg, Roland, and Yamaha in the middle 1970s that converted potential into popularity. (Nagle, Paul, 1995)

2.2.2 Digitally Controlled Synthesizers

The popularity of synthesizers got a major boost in 1978 when microprocessor-based instruments began to appear, spearheaded by a new California company Sequential Circuits. The Prophet-5, though still hugely limited by today's standards, offered reasonable levels of playability, stability, and polyphony, albeit at a hefty price at the time (around $4000). Soon Korg, Roland, and Yamaha's microprocessor-based offerings would slash prices in half, and by the turn of the decade the polyphonic synthesizer was firmly on the map for every self-respecting keyboard player from hobbyists to touring professionals. The days of the *Hammond* organ, the *Fender Rhodes* piano, and latterly the *Hohner Clavinet* were coming to an end.

Stability, playability, and polyphony continued to evolve in the early 1980s but *compatibility* remained a thorn in the side of manufacturers. The multifarious nature of synthesizer design meant that each manufacturer had been defining pitch and timing (Control Voltage and Gate) data in their own way. Once polyphonic, *digital* technology became available manufacturers they began to design unique digital interfaces that would, at the very least, allow to connect several Korg, or Roland, or Yamaha synths together. Roland developed its *DCB* (Digital Communication Bus), Yamaha its *Key Code Interface* etc.

Visionaries like Dave Smith from Sequential Circuits, and Ikutaru Kakehashi from Roland began to worry that this lack of compatibility between manufacturers

would restrict people's use of synthesizers, which would ultimately inhibit sales growth. Talk of a 'universal' digital communication system thus began circulating in 1981. Dave Smith and Chet Wood presented a paper that year at AES proposing a concept for a Universal Synthesizer Interface running at 19.2 kBaud, using regular 1/4" phone jacks. At the following NAMM show in January 1982 a meeting took place between the leading American and Japanese synthesizer manufacturers where certain improvements were made to the specification: increasing the Baud rate to 31.25 and adding the opt-isolation circuit.

2.2.3 MIDI is born

MIDI (an acronym for "musical instrument digital interface") as its name was ultimately chosen, was first announced to the public in 1982, and by as early as January 1983 actually appeared on an instrument; the Sequential Prophet-600. Roland's JX3P followed hot on its heels, was 'connected' successfully, and a new chapter in the history of electronic musical instruments was born. (MIDI Manufacturer, 1995)

In 1983, the MIDI Specification was only about 8 pages long and defined only the most basic instructions one might want to send between two synthesizers' things like how to play notes and how to control the output volume, etc. Very quickly, the arrival of this 'common (digital) language' created demand for new MIDI messages that enabled greater control of synthesizers but also for control of other recording gear and even stage lighting. MIDI also enabled computers to be applied to the music-making process. Although the way that MIDI works has not changed since 1983 (also almost preposterously inconceivable), the MIDI protocol has grown to encompass such additional concepts as: standardized MIDI song files (*General MIDI, 1991*); new connection mechanisms such as USB, FireWire, and Wi-Fi; new markets such as mobile phones and video games; and a whole world of 'alternative' and 'performance' based MIDI products.

The agreement to adopt a standard (and royalty-free) technology was an incredible achievement in itself – and substantially unmatched to this day – but it

was also remarkable for what it then enabled. Sequencers, sampling, digital drum machines, dedicated computer control, ultimately a complete revolution within the recording industry. It is hard to imagine that any of these technologies or developments would have occurred, or certainly have been as wide-reaching, without the glue of MIDI. (*General MIDI, 1991*)

2.4 MIDI Protocol

MIDI is a serial protocol that enables electronics musical instruments, computers and other equipment to communicate, control and synchronize with each other. (William Llord and Paul Terry,1996). The MIDI protocol is made up of messages. A message consists of string (ie, series) of 8-bit bytes. MIDI has many such defined messages. Some messages consist only 1 byte, two or three bytes in length, although some may be longer.

"Each MIDI message, regardless of its length contains a single status byte and zero or more data bytes. The numerical value of a status byte is always between 128 and 255 (0x80 to 0Xff). All data bytes fall between 0 and 127 (0x00 to 0x7F). This provides easy identification of status data, but limits the range of a single data value to seven bits." (Paul Messick, 1998)

MIDI message consist two basic types: Channel message and system message. Channel message are directed at a particular destination and are subdivided into channel note and channel mode message. (Paul Messick, 1998) System message come in three flavors: system common message, system Real-Time message and System Exclusive messages.

MIDI data bytes are organized into two major classification, Status bytes and Data bytes. Any Byte that has the MSB (Most Significant Bit) equal to 1 is a status Byte. Any Byte with the MSB equal to 0 is a data byte. Any given MIDI message consists of a Status Byte followed by any number of Data bytes, normally zero, one or two, but this is virtually unlimited.

What are status and data bytes? MIDI bytes fall into one of 2 categories. (Paul White, 2000)

i. Status bytes always start with a 1 and define the type of message being sent. This is an example status byte, **10010011,** which means Note On / Channel 2

ii. Data bytes start with a 0 and simply give a value between 0 and 127. Here is an example data byte, **01000001,** which means 65

MIDI Keyboard produced a MIDI message signal in each notes. (Paul White, 1996) It shows in table 2.1. In this table, each MIDI notes produce a message signal in HEX or binary numbers.

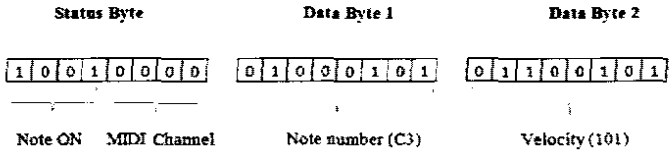
Table 2.1: MIDI numbers produce by MIDI Keyboard [4]

Midi Number	Note name	Midi Keyboard	Computer Keyboard	
24 25	C1		1 2	Select Video.
26 27	D1		3 4	
28	E1		5	
29 30	F1		6 7	
31 32	G1		8 9	
33 34	A1		0	
35	B1			
36 37	C2		Q W	Start video at selected point.
38 39	D2		E R	
40	E2		T	
41 42	F2		Y U	
43 44	G2		I O	
45 46	A2		P A	
47	B2		S	
48 49	C3		D F	
50 51	D3		G H	
52	E3			
53 54	F3			
55 56	G3			
57 58	A3			
59	B3			
60 61	**C4**		K L	Set Volume of video.
62 63	D4		Z X	
64	E4		C	
65 66	F4		V	
67 68	G4		B	play/stop/pause.
69 70	**A4**		M	
71	B4		N	

MIDI is an asynchronous serial interface. The baud rate is 31.25 Kbaud (+/-1%). There is 1 start bit, 8 data bits, and 1 stop bit (ie, 10 bits total), for a period of 320 microseconds per serial byte. (Paul White, 2000)

In table 2.2 show the example of simple MIDI message contains status byte and 2 data bytes. This message is telling a sound module set to respond on MIDI channel 1 to start playing a note (C3) at a velocity of 101

Table 2.2: Example of MIDI Message [5]



Many electronic instruments not only respond to MIDI messages that they receive (at their MIDI IN jack), they also automatically generate MIDI messages while plays the instrument (and send those message out their MIDI OUT jacks). (Paul Messick, 1998)

In figure 2.2, show that MIDI keyboard is pushed down. Not only does this sounding musical note, it also causes a MIDI Note-on message to be sent out of the keyboard's MIDI OUT jack. This message consists of three numeric values as shown below;

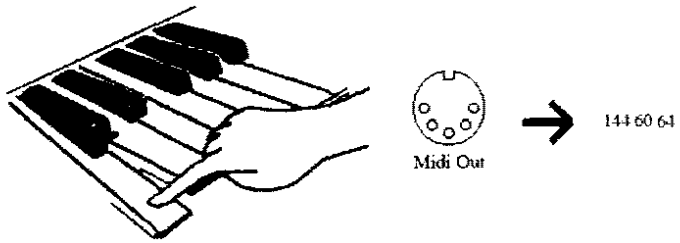


Figure 2.2: A middle C key on a keyboard is push down. [6]

Figure 2.3 show a keyboard is now release. Not only does this stop sounding the musical note, it also causes another message. A MIDI Note-Off message to be sent out of the keyboard's using MIDI OUT jack. This message consists of three numeric values as shown below. Note that one of the values is different than the note message.

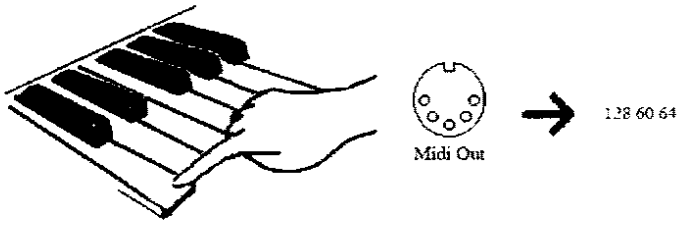
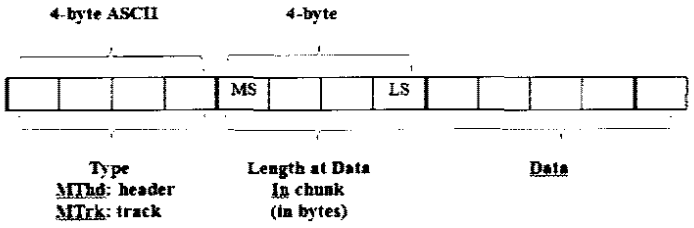


Figure 2.3: A middle C key on a keyboard is now releases. [7]

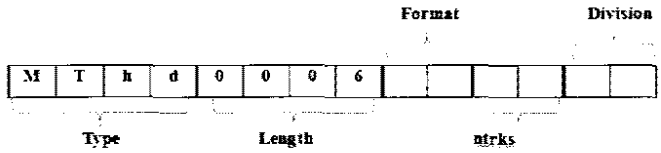
2.5 STANDARD MIDI FILES

Standard MIDI Files designed to allow exchange of sequence data between devices. These files represent data as events belonging to individual sequencer tracks, plus info such as track or instrument names and time signatures. There are 3 types of MIDI files for representing: single-track data (type 0), synchronous multi-track data (type 1) and asynchronous multi-track data (type 2). Data is organized as bytes grouped into header and track chunks. (William Llord and Paul Terry,1996).

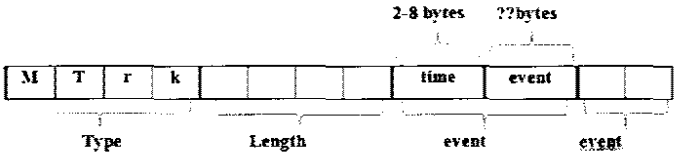


Header chunk is a format that defines the MIDI file type (0,1 or 2), ntrks defines the number of track chunks and division defines the timing format. The timing format is defined by MSB of the 2-byte division word. 0 indicates a division

of tricks per quarter note, while 1 indicates a division of ticks per time code frame. (William Llord and Paul Terry,1996)



Truck chunks contains string of MIDI events, MIDI events can be channel messages, SysEx and meta-events (containing labels and internal data). (William Llord and Paul Terry,1996).



2.6 DATA ACQUISITIONS FROM MIDI FILES

MIDI File is a data from original binary, making it possible to achieve and manipulate in the data frames on computer. (Howard Massey, 1988) There are two solutions for acquiring the data. The first method is "software-based', extracting MIDI data from MIDI Files. The "hardware-based" method utilized part of an actual MIDI system, interfaced with a PC, to transfer music data to the computer.

2.6 DATA TRANSMISSION PRINCIPLE

Data transmission use in this project is Serial Peripheral Interface (SPI) and Universal Asynchronous Receiver/Transmitter (UART)

2.6.1 Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. (John Catsoulies, 2002)

Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines. (John Catsoulies, 2002) In figure 2.4, there are 4 wires serial from Master to Slave. SPI is called a *four-wire* serial bus, contrasting with three, two and one wire serial buses. SPI is often referred to as SSI (Synchronous Serial Interface).

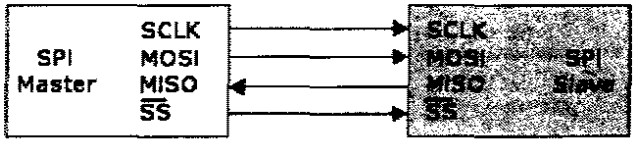


Figure 2.4: SPI bus: single master and single slave [8]

The SPI bus specifies four logic signals:

i. SCLK: serial clock (output from master)
ii. MOSI: master output, slave input (output from master)
iii. MISO: master input, slave output (output from slave)
iv. SS: slave selects (active low, output from master)

2.6.2 Universal Asynchronous Receiver/Transmitter (UART)

The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications subsystem of a computer. UART is also a common integrated feature in most microcontrollers. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Serial transmission of digital information (bits) through a single wire or other medium is much more cost effective than parallel transmission through multiple wires. Communication can be "full

duplex" (both send and receive at the same time) or "half duplex" (devices take turns transmitting and receiving). (RH2T Magazine *Vol.3, Dec 2010*)

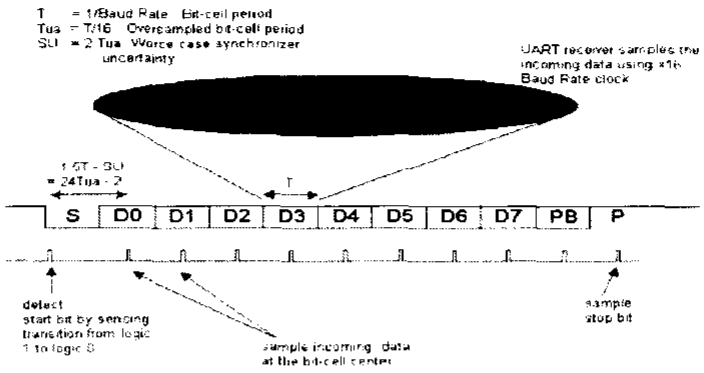


Figure 2.5: Data sampling point by the UART receiver [9]

Every operation of the UART hardware is controlled by a clock signal which runs at much faster rate than the baud rate. (RH2T Magazine *Vol.3, Dec 2010*) For example, in figure 2.5, UART with 16450 has an internal clock that runs 16 times faster than the baud rate. This allows the UART receiver to sample the incoming data with granularity of 1/16 the baud-rate period and has greater immunity towards baud rate error.

The receiver detects the Start bit by detecting the voltage transition from logic 1 to logic 0 on the transmission line. In the case of 16450 UART, once the Start bit is detected, the next data bit's "center" can be assured to be 24 ticks minus 2 (worse case synchronizer uncertainty) later. From then on, every next data bit center is 16 clock ticks later. (RH2T Magazine *Vol.3, Dec 2010*). UART communication between two device are shown in figure 2.6, which are from TX (Transmitter) to RX (Receiver)

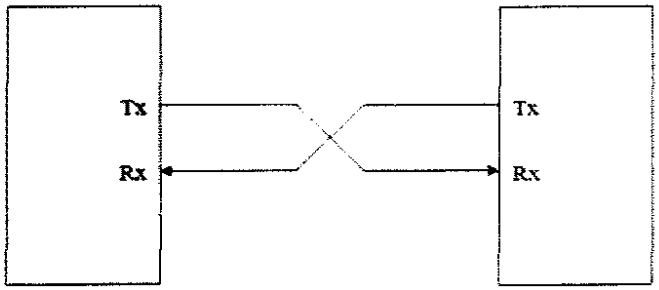


Figure 2.6: UART communication between two devices [10]

2.7 MIDI CIRCUIT INTERFACE

The MIDI circuit is current loop, 5mA. Logic 0 is current ON. One output drivers (and only one) input. According to *Madame Butterface (1990)*, "To avoid grounding loops and subsequent data errors, the input is opt-isolated. It requires less than 5mA to turn on. Rise and fall time for the opt-isolator should be less than 2 microseconds".

The standard connector used for MIDI is a 5 pin DIN. Separate jacks (and cable runs) are used for input and output, clearly marked on a given device. Cables are shielded twisted pair, with the shield connecting pin 2 at both ends. The pair is pin 4 and 5. Pin 1 and 3 are not used, and should be left unconnected. (*David Miles Huber, 2000*)

A 5-pin "DIN" connector is used as shown in figure 2.7. It used to be that connecting a MIDI device to a computer meant installing a "sound card" or "MIDI interface" in order to have a MIDI DIN connector on the computer.

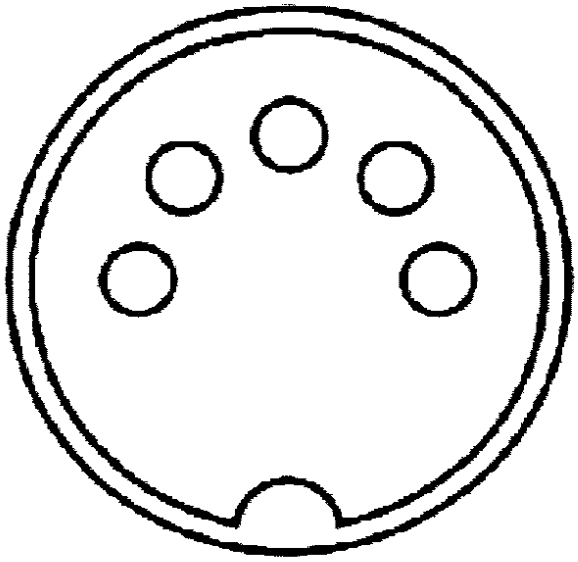


Figure 2.7: MIDI devices use DIN 5-pin at 180^0 connector for cables and connectors. The connectors on instruments are female, cables are male. [11]

MIDI messages are transferred between devices using UART. This means that the MIDI sequencer is connected to the MIDI decoder with a DIN 5-pin cable and the MIDI is transmitted using UART.

2.7.1 MIDI Shield

The MIDI Shield provides an opt-isolated MIDI-IN port as well as a MIDI-OUT port. A schematic of a MIDI (IN and OUT) interface shown in figure 2.8. The MIDI Shield can be mounted directly on top of an Arduino, connecting the MIDI-IN to the Arduino's hardware RX pin and the MIDI-OUT to TX. This MIDI Shield was specially design that can choose two jobs and have a button reset. The jobs are:

i. Play MIDI from MIDI Keyboard
ii. Play MIDI file from PC based

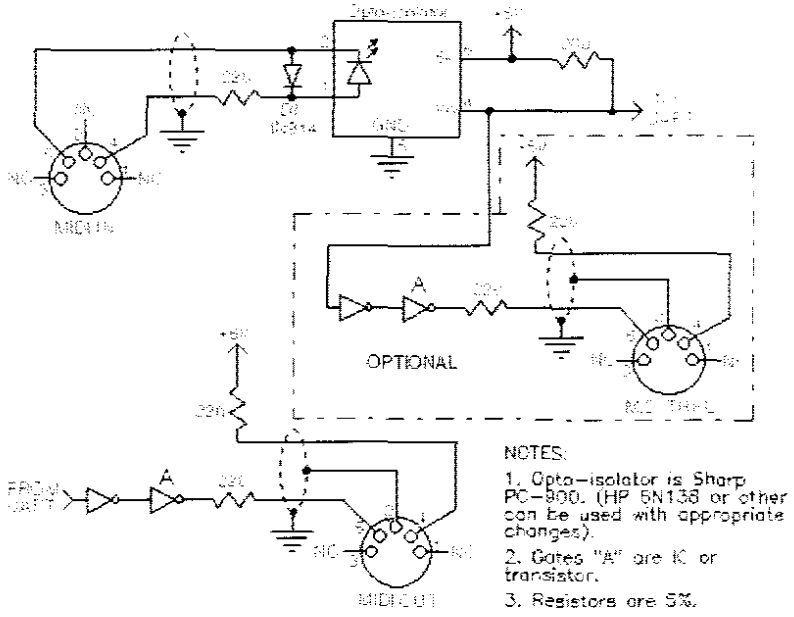


Figure 2.8: A schematic of a MIDI (IN and OUT) interface [12]

2.7.2 Opt-Isolator

In electronics, an opt-isolator, also called an opt-coupler, photo-coupler, or optical isolator, is a component that transfers electrical signals between two isolated circuits by using light. Opt-isolators are usually used for transmission of digital (on/off) signals, but some techniques allow use with analog (proportional) signals.

2.8 ANGKLUNG

Angklung is a music instrument made from joint pieces of bamboo. It consists of two to four bamboo tubes mounted together within a bamboo frame, as shown in figure 2.9, a bound with rattan cords. The angklung produce certain notes when the bamboo frame is shaken or tapped. Each angklung produces a single note or chord, so several players must collaborate in order to play melodies. The instrument has been known since ancient times in some parts of Indonesia, especially in West Java, Central Java, East Java, and Bali in Indonesia. (*Professor Kuo-Huang Han*)

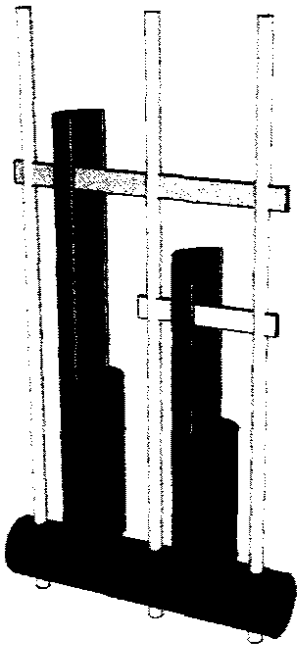


Figure 2.9: Example single note of the Angklung [13]

The interval between the differently-sized bamboo tubes on each Angklung is one octave. Most Angklung sets today are tuned to the western chromatic and diatonic scales. (Professor Kuo-Huang Han)

2.8.1 The Octave

In music, an octave or perfect octave is the interval between one musical pitch and another with half or double its frequency. The octave relationship is a natural phenomenon that has been referred to as the "basic miracle of music", the use of which is "common in most musical systems". It may be derived from the harmonic series as the interval between the first and second harmonics.

2.8.2 The Chromatic Scale

In music theory, a diatonic scale is commonly defined as a seven-note, octave-repeating musical scale comprising five whole steps and two half steps for each octave, in which the two half steps are separated from each other by either two

or three whole steps. This pattern ensures that, in a diatonic scale spanning more than one octave, all the half steps are maximally separated from each other. (*Adam Koss*)

Any sequence of seven successive natural notes, such as C-D-E-F-G-A-B, and any transposition thereof, is a diatonic scale. Piano keyboards are designed to play natural notes, and hence diatonic scales, with their white keys. It is made up of seven distinct notes, plus an eighth which duplicates the first an octave higherInsolfege, the syllables used to name each degree of the scale are "Do–Re–Mi–Fa–Sol–La–Ti–Do". Table 2.3 below is an example of Diatonic scale. (*Adam Koss*)

Table 2.3: Example Diatonic Scale

Notes in C major	C	D	E	F	G	A	B	C
Degrees in solfege	Do	Re	Mi	Fa	Sol	La	Ti	Do

2.9 PREVIOUS AUTOMATIC ANGKLUNG INVENTIONS

Actually, Angklung also have already implemented in automatically control, for example in Indonesia, their student have develop angklung-playing system under the names "KlungBot" and "Klungto Mobi". Innovation of these two instruments is just being preinstalled to the system using parallel communication and the songs that Angklung want to play automatically needs to be programmed in the memory of the microcontroller. In figure 2.10, show the interface for program a song before send to the microcontroller.

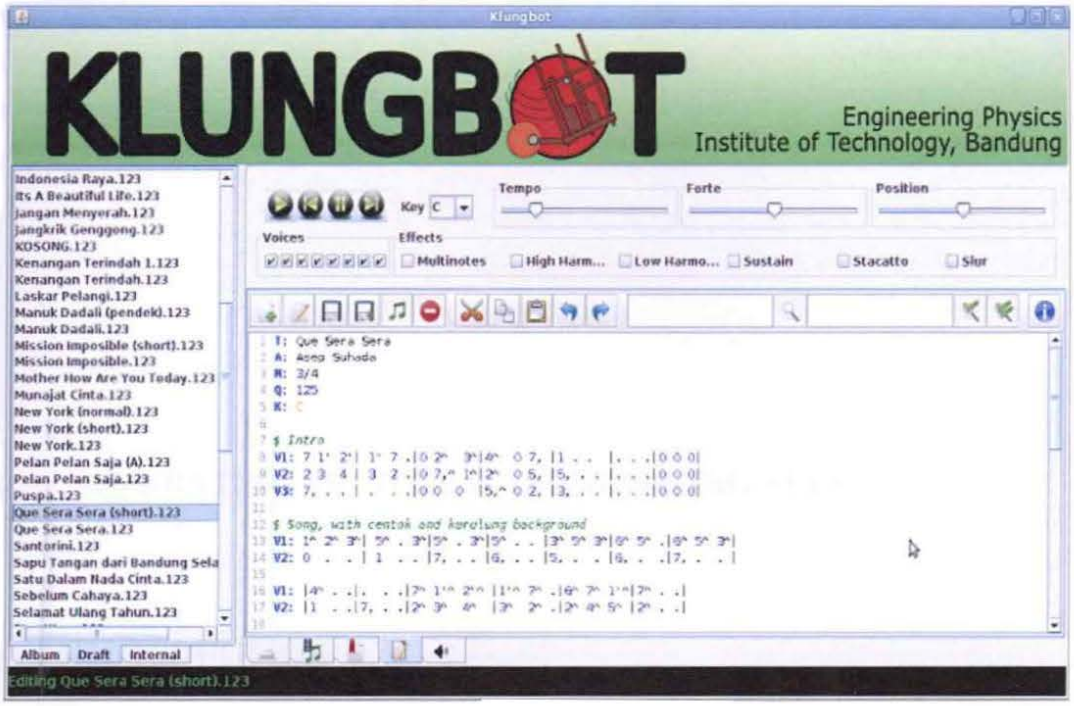


Figure 2.10: User Interface for program a songs to Automatic Angklung [14]

One of the Automatic Angklung make from automatic control system show below in figure 2.11 named KlungBot. KlungBot, is invented by Indonesia Student.

Figure 2.11: KlungBot [15]

CHAPTER 3

OPERATING PRINCIPLE AND WORKING ALGORITHM

3.1 INTRODUCTION

This chapter will be discussed the method that use to decode MIDI message in accordance to the objectives of the study. Methodology is like a strategy or plan for achieving some goal; methods that can be used to service the goals at the methodology. In essence, methodology provides the blueprints that prescribe how the process or tools should be used.

3.2 CONCEPT OF PROJECT

This project consists of two interfaces to control an Angklung which is MIDI keyboard or PC. This interface will send MIDI message to microcontroller and decode it to the output. The output consist 18 DC Motor to shake the Angklungs (2 and half octave) to produce the sound.

MIDI keyboard generates MIDI message when the keys is press. This message then sends to the computer. MIDI message send two piece of information that is which key was pressed and how fast it pressed. If the MIDI send message with higher velocity, the DC motor will generate speed according to the message.

Second method to generate MIDI message is directly from Computer using USB-MIDI or Android with Bluetooth protocol. This interface, stored various MIDI

files. User can choose what type of MIDI file they want to play. The block diagram for this project is shown in figure 3.1.

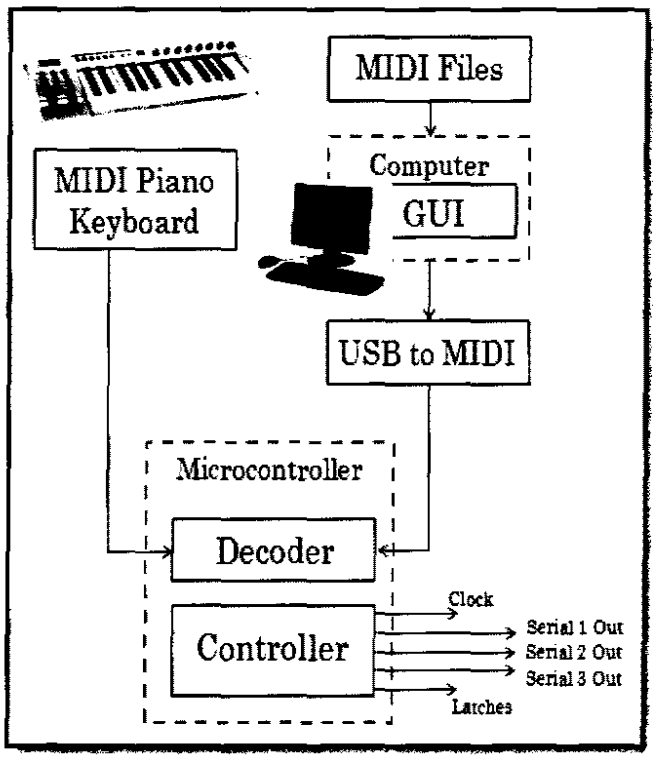


Figure 3.1: Block Diagram for MIDI decoder

However, this project is not using MIDI exactly as it is supposed to be used. Instead of connecting the MIDI sequencer to an Electronic piano's MIDI-In port, this project do the playing in a roundabout way by connecting the MIDI sequencer to a device, which translates the MIDI messages to movement of the motor driver which in term play the Angklung from the actual keys.

To produce the octaves sound, DC Motor is attaching to the Angklung. The DC Motor will shake the Angklung with a variant speed. For Example, the MIDI message sent very fast, and then DC motor will extract the message and produce higher speed (Velocity) to the Angklung.

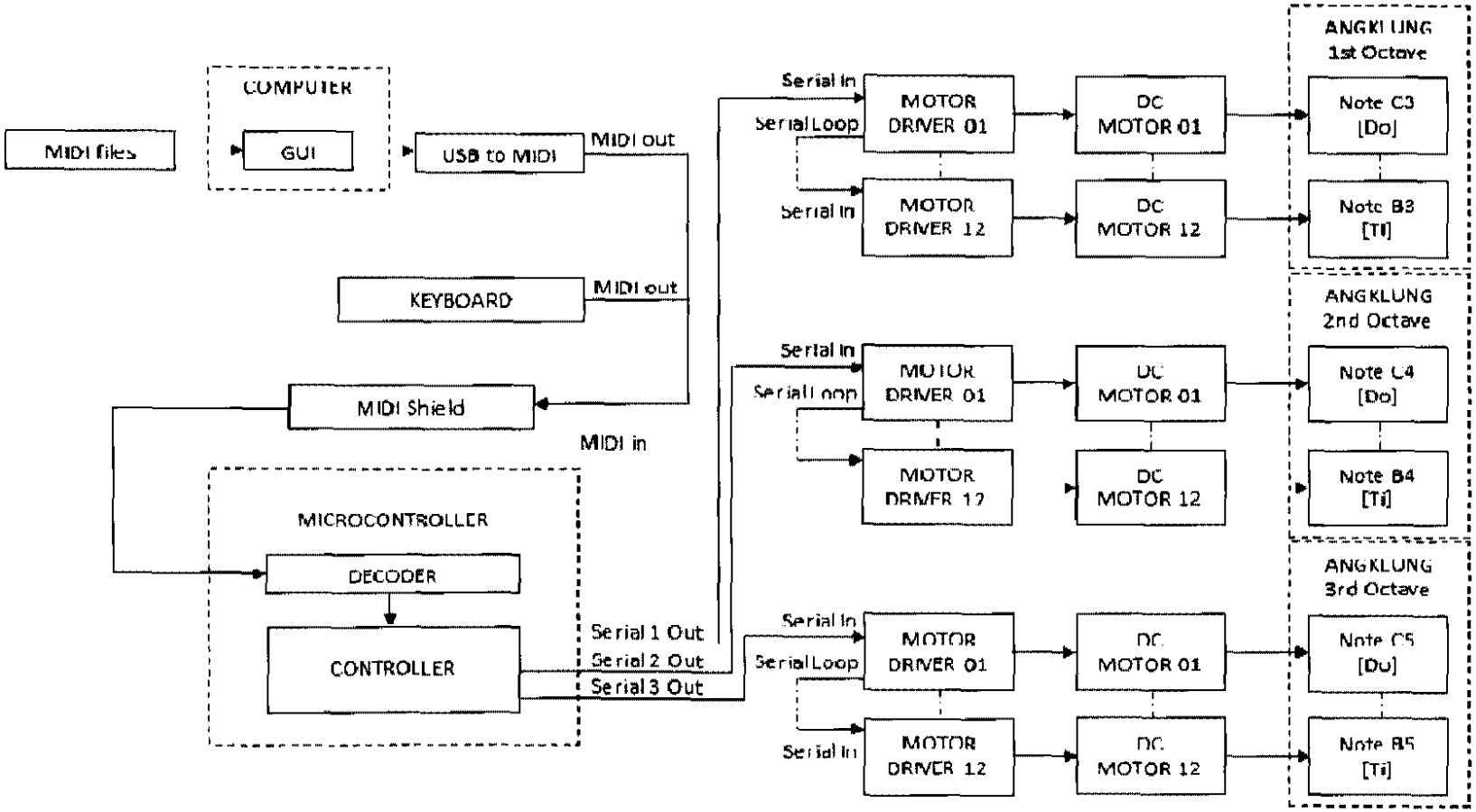


Figure 3.2: Whole Block Diagram

3.3 MIDI KEYBOARD

MIDI Keyboard as shown in figure 3.3 are used in this project is to send the MIDI signal to the microcontroller, decode it and sent it to the motor driver according to the address. A MIDI keyboard is typically a piano-style user interface keyboard device used to sending MIDI signals or commands over a USB or MIDI cable to other device connected and operating on the same MIDI protocol interface. MIDI to MIDI cable is used to send MIDI message to microcontroller. This MIDI to MIDI cable use in this project is shown in figure 3.4.

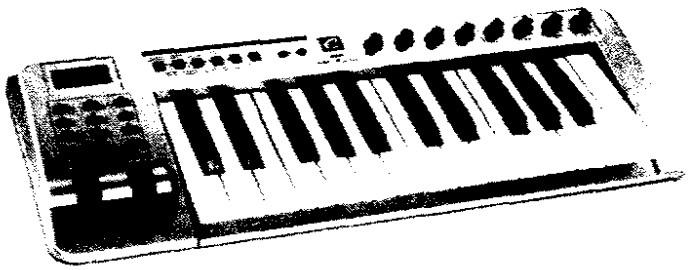


Figure 3.3: MIDI Keyboard [13]

Figure 3.4: MIDI to MIDI cable [14]

3.4 PERSONAL COMPUTER (PC)

Computer also use as user interface to control an Angklung via MIDI. A Graphical User Interface that can plays, read and send MIDI message can be used to interface it with microcontroller. Computer has a sound card that can read MIDI. In figure 3.5, show how to interfacing from a PC using USB to MIDI cable.

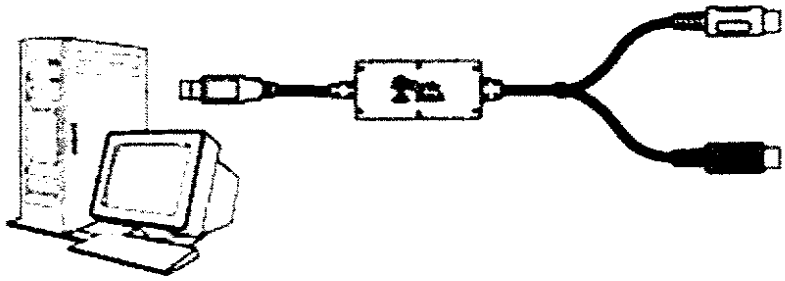


Figure 3.5: Interfacing PC Based using USB to MIDI [11]

3.4.1 Graphical User Interface (GUI)

GUI (Graphical user interface) used in this project is "Vanbasco's and MIDI Piano" as shown in figure 3.6 and 3.7. This user interface will generate midi message or midi files and send it to microcontroller to decode it. Furthermore, any GUI that can play, send and read MIDI message can be used with this microcontroller thru MIDI Shield.

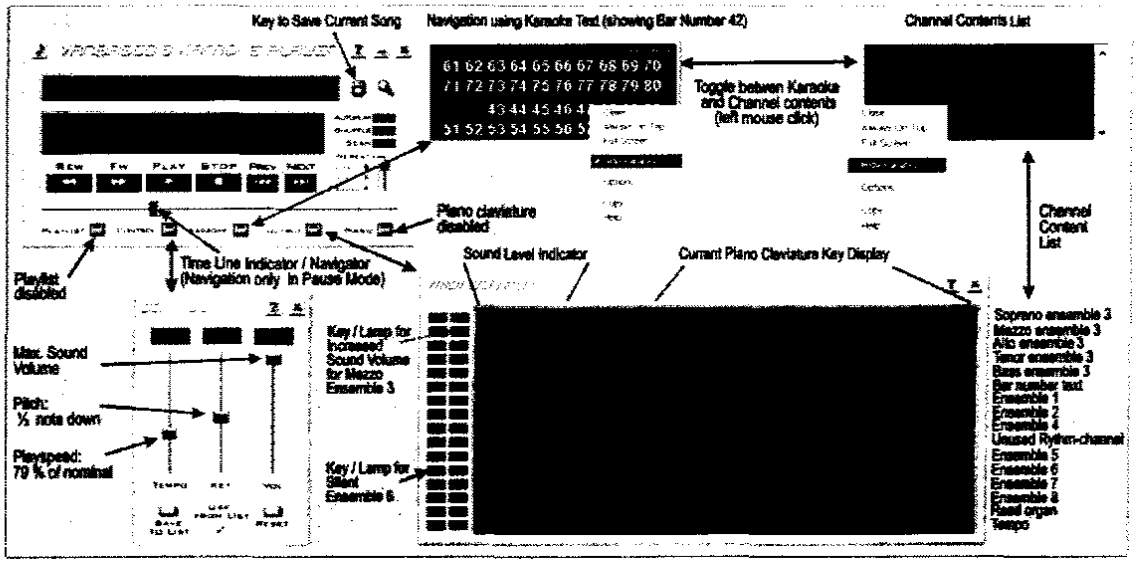


Figure 3.6: VanBasco User Interface [15]

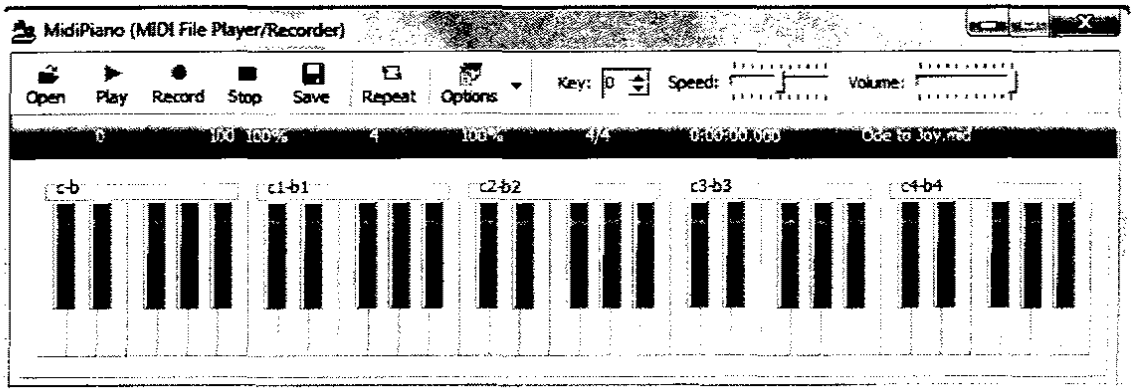


Figure 3.7: MIDIPiano User Interface [16]

3.4.2 USB MIDI Cable

USB to MIDI interface will be used to connect a computer with MIDI shield to microcontroller. This MIDI interface cable is self-powered and can be conveniently attached to computer's USB port, without the need for tools or computer disassembly. This MIDI interface cable use in this project shown in figure 3.8 is the simplest and most convenient way to connect a keyboard or controller to a computer via a USB port. This cable is plug and play's and will send the MIDI message in real time to microcontroller thru MIDI Shield.

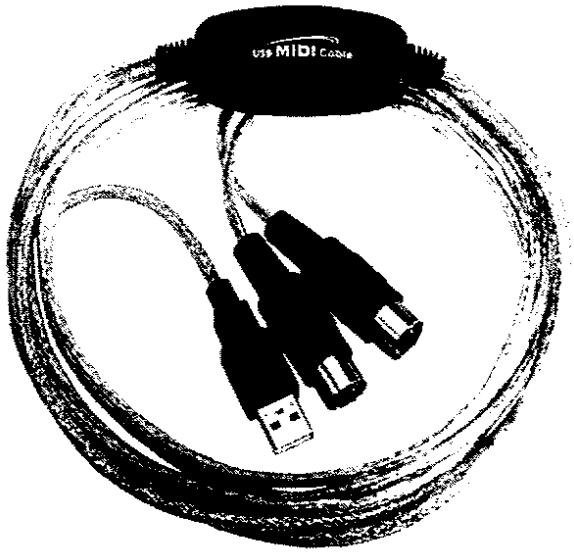


Figure 3.8: USB to MIDI cable [17]

3.5 MICROCONTROLLER

A microcontroller is used to decode the MIDI message. In this project, the Arduino DUE is used because it has a high speed clock that can send MIDI message with small delays.

3.5.1 Arduino DUE

The Arduino Due is a microcontroller board as shown in figure 3.9.It is based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is the first Arduino board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs, 4 UARTs (hardware serial ports), a 84 MHz clock, an USB OTG capable connection, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button.

The Atmel SAM3X8E ARM Cortex-M3 microcontroller is set to operate at 8 MHz which was more than enough to handle the MIDI decoder. The reason for using an 8 MHz clock was to get the right baud rate to receive the MIDI signal, as the UART receiver is dependent on the clock rate of the device it is part of. Therefore a clock rate has to be used which is compatible with a baud rate of 31250 baud.

When powered on the MIDI message continuously sends SPI messages to all the motor driver board. These messages contain information on which DC motor should be turn on. Table 3.1 below shows the specification for Arduino Due.

Table 3.1: Arduino DUE Specification

Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 12 provide PWM output)

Analog Input Pins	12
Analog Outputs Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz

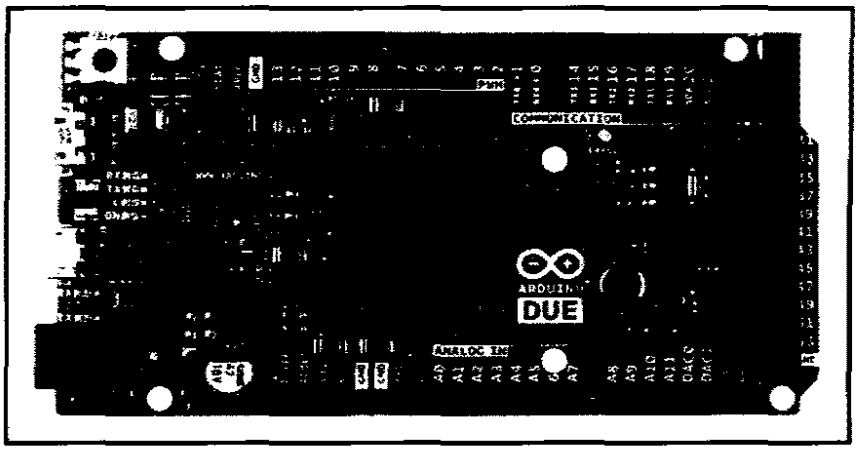


Figure 3.9: Arduino DUE microcontroller Board [18]

MIDI Decoder programming was developed to run on an Arduino DUE, Atmel SAM3X8E ARM Cortex-M3 microcontroller. The microcontroller unit was chosen because:

i. It is sufficiently powerful for handling MIDI Decoder
ii. It has UART capabilities
(Used to receive MIDI Message)
iii. It has SPI Header
iv. This microcontroller was already have MIDI library

3.5.2 Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. SPI is used to control the Motor Driver Boards.

With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices:

i. **MISO** (Master In Slave Out) - The Slave line for sending data to the master,
ii. **MOSI** (Master Out Slave In) - The Master line for sending data to the peripherals,
iii. **SCK** (Serial Clock) - The clock pulses which synchronize data transmission generated by the master

The Arduino Due's SPI interface works differently than any other Arduino boards. The library can be used on the Due with the same methods available to other Arduino boards or using the extended methods. The extended method exploits the SAM3X hardware and allows some interesting features like:

i. Automatic handling of the device slave selection.
ii. Automatic handling of different device configurations (clock speed, data mode, etc.) so every device can have its own configuration automatically selected.

Describing SPI is easier when it is cut down into separate parts. These separate parts are physical design, hardware protocol and software protocol.

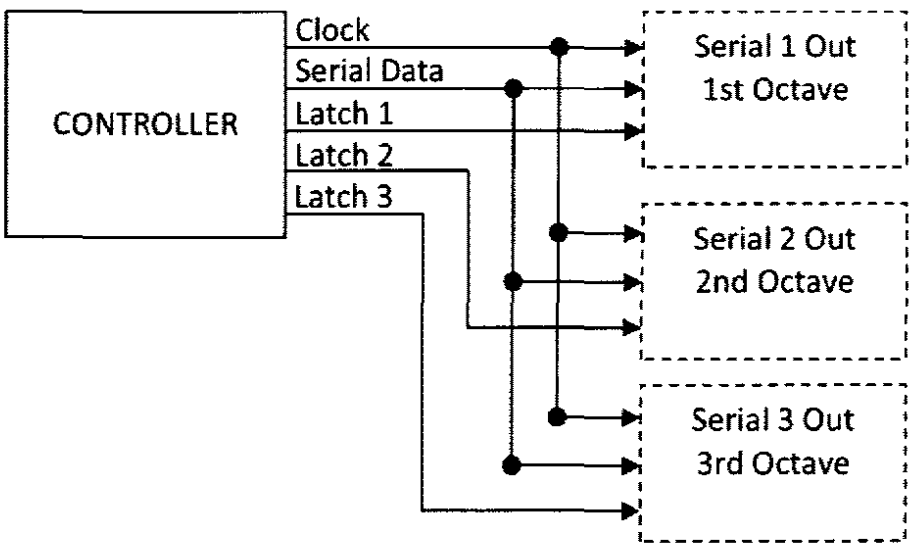


Figure 3.10: SPI OUT from microcontroller

3.6 C LANGGUAGE

The Arduino runs a simplified version of the C programming language, with some extensions for accessing the hardware. Program written using Arduino is called sketches as shown in figure 3.11. These sketches are written in the text editor. Sketches are saved with the file extension .ino. It has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino environment including complete error messages and other information. In this project, MIDI Decoder has been written in this language. (*Programming code; see Appendix*)

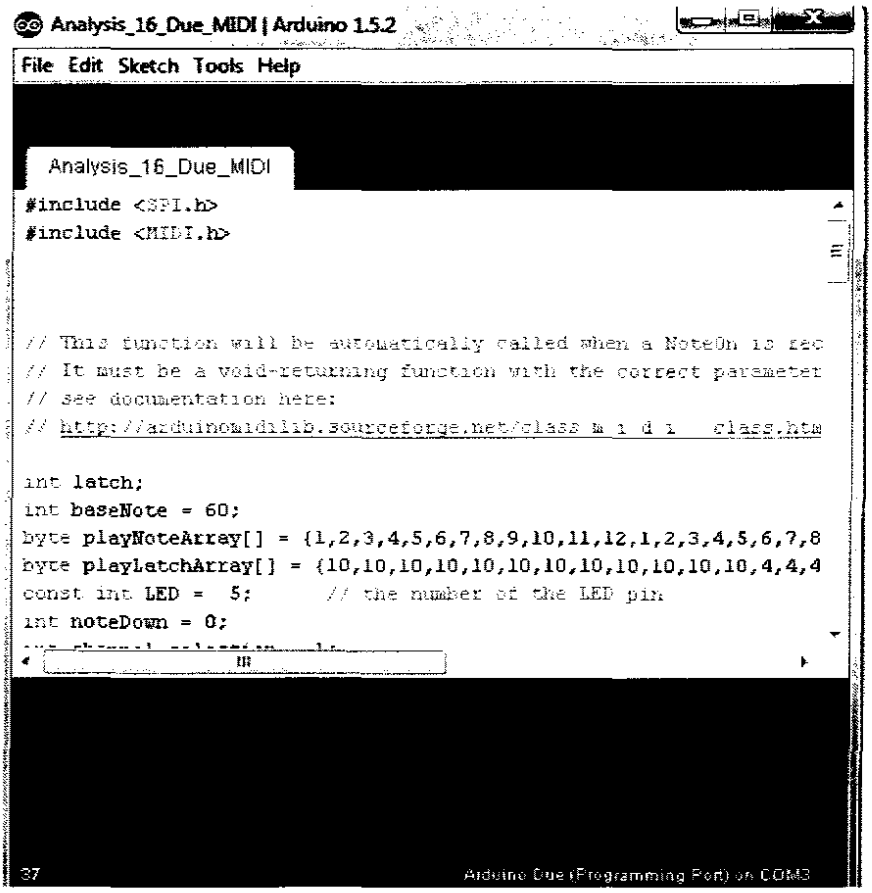


Figure 3.11: Arduino Programming Sketch interface

3.7 The Universal Asynchronous Receiver/Transmitter (UART)

The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications subsystem of a computer. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes.

The MIDI device use UART in a configuration that has one start bit, 8 data bits and one stop bit with the baud rate of 31250 baud. Thus MIDI sequencers sending the MIDI message and the MIDI decoder receiving the MIDI message have both been configured with these parameters.

3.8 MIDI SHIELD

The MIDI Shield provides an opt-isolated MIDI-IN port as well as a MIDI-OUT port. The MIDI Shield can be mounted directly on top of an Arduino, connecting the MIDI-IN to the Arduino's hardware RX pin and the MIDI-OUT to TX. This MIDI Shield was specially design that can choose two jobs and have a button reset. The jobs are:

iii. Play MIDI from MIDI Keyboard

iv. Play MIDI file from PC based

3.8.1 Operations

Opt-isolator use in this MIDI Shield is 6n138. The schematic is quite straight forward as shown in figure 3.12, on the transmit side the serial output is fed through a resistor to the base of a PNP transistor. These are sometimes called "upside down" transistors because a high voltage into it makes it stop conducting and a low voltage makes the collector / emitter conduct.

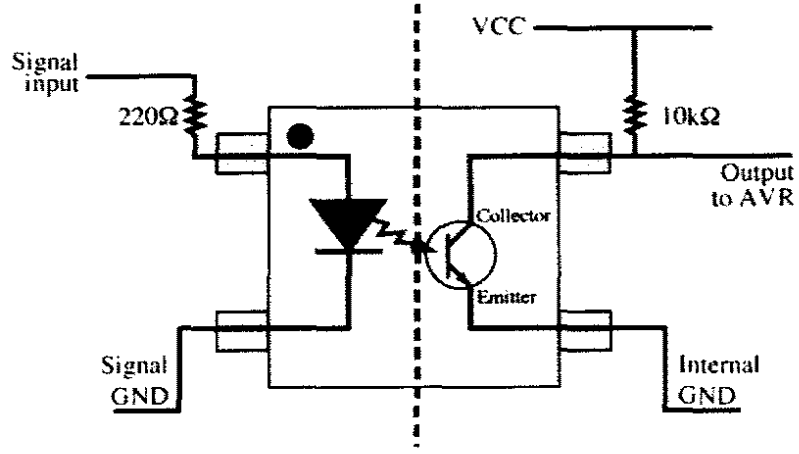


Figure 3.12: Opt-Isolator Schematic [19]

The output is then channeled through the MIDI receiving device through two 220R resistors. In effect this is a current output as opposed to the more normal voltage output. On the receive side the signal from the MIDI transmitting device is passed through a 6N138 opt isolator. This is basically a LED and a photo transistor in the one package the only connection between the sending device and the arduino is a light path. There is a resistor to protect the input from too much current and a diode to protect against reverse voltage if wired the MIDI leads up incorrectly. The photo transistor output is simply pulled up and fed into the receive pin of the arduino. The MIDI shield schematic for this project is shown in figure 3.13.

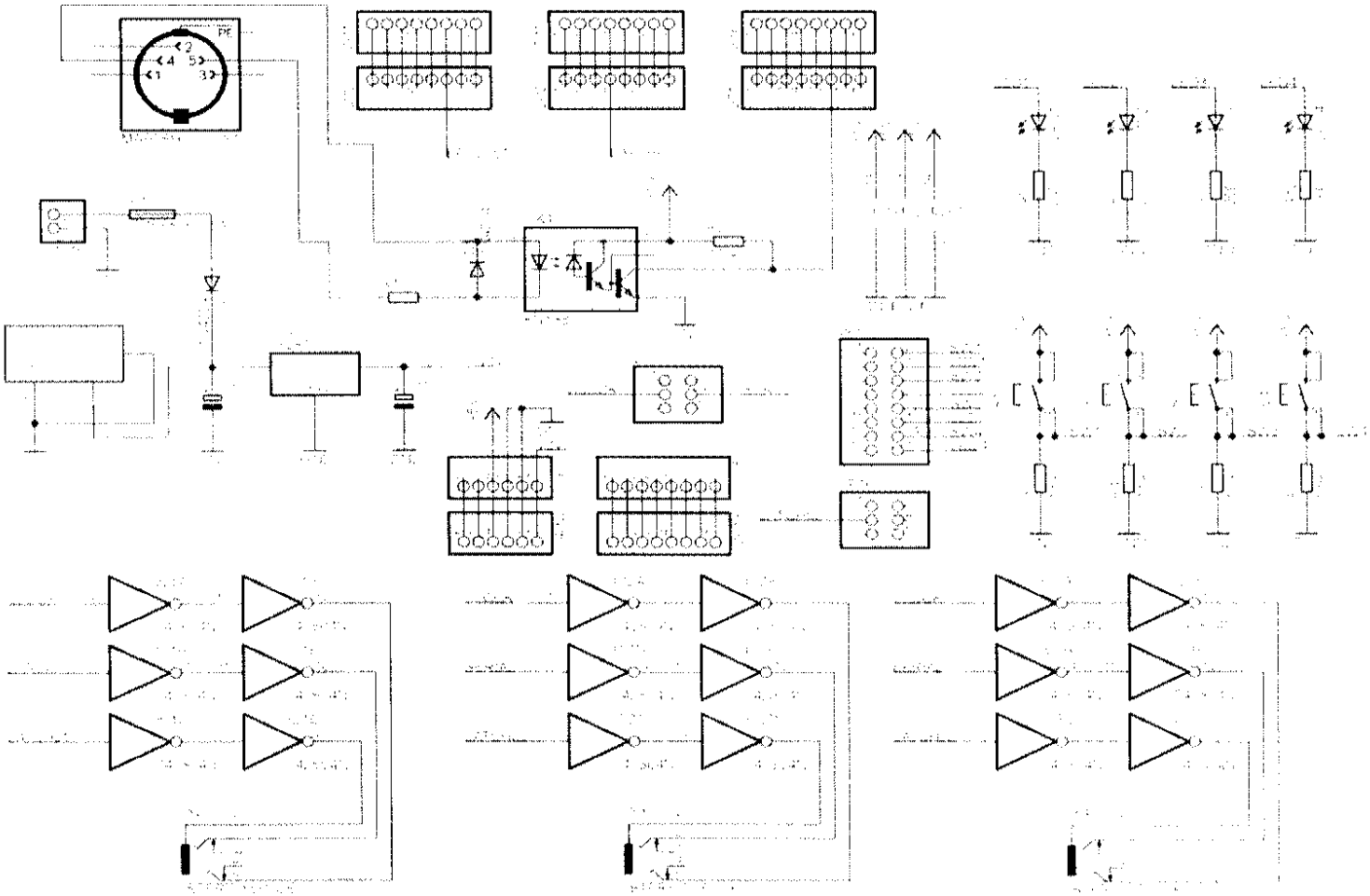


Figure 3.13: MIDI Shield Circuit

3.9 FLOW CHART OF CIRCUIT TESTING PROCESS

Flow chart circuit testing is important to ensure the system is perfectly done before combining with User interface and motor driver to avoid difficulties if the system does not work. It helps make the process easier to check connection of the component like opt-isolator, Arduino, and MIDI Shield.

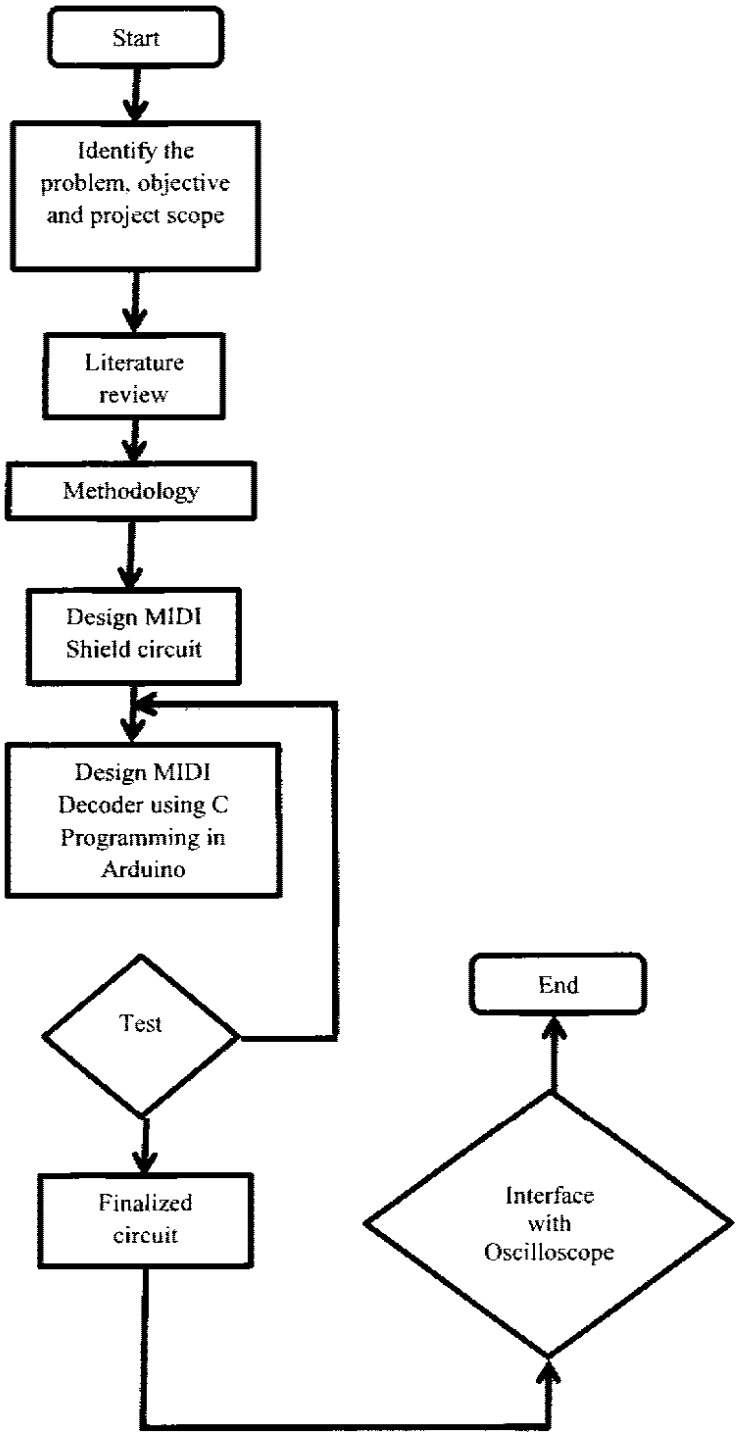


Figure 3.14: Process of testing MIDI Shield Circuit

CHAPTER 4

EXPERIMENTAL RESULT AND PERFORMANCE ANALYSIS

4.1 Introduction

The chapter will brief every experiment on the methodology or the flow of work is come out with the result and analysis. The result of this project will include the MIDI decoder program using microcontroller and output signals that have being handled by MIDI Shield to microcontroller. This chapter will discuss mainly about the problems encountered during the whole project was been carried out.

4.2 Development of MIDI Angklung Controller (MAC)

This project, MIDI decoder for Auto Angklung is using programming and circuit interfacing method. Programming part is the most important, which it use to decode the MIDI Sequencer and make sure the decoded MIDI can be read at the motor driver address.

4.2.1 MAC System Program

When powered on the board goes through the initialization phase. During the initialization, all the variables are initialized as well as UART.

The UART receiver is set for a baud rate 312500 baud with 8 word data bits without a parity bit. As covered before, this is the right configuration for receiving MIDI message bytes from the MIDI sequencer.

A controller will initialize MIDI library and SPI library. In MIDI setting, controller will define whether MIDI message is receive. If it receive correct MIDI message, the velocity in MIDI message will be set follow by velocity setting. Microcontroller now is ready.

There are 4 job selections in this project. First job is job 1 the task is to decode MIDI Files. Second job is to decode MIDI message and job reset is to reset all the data received.

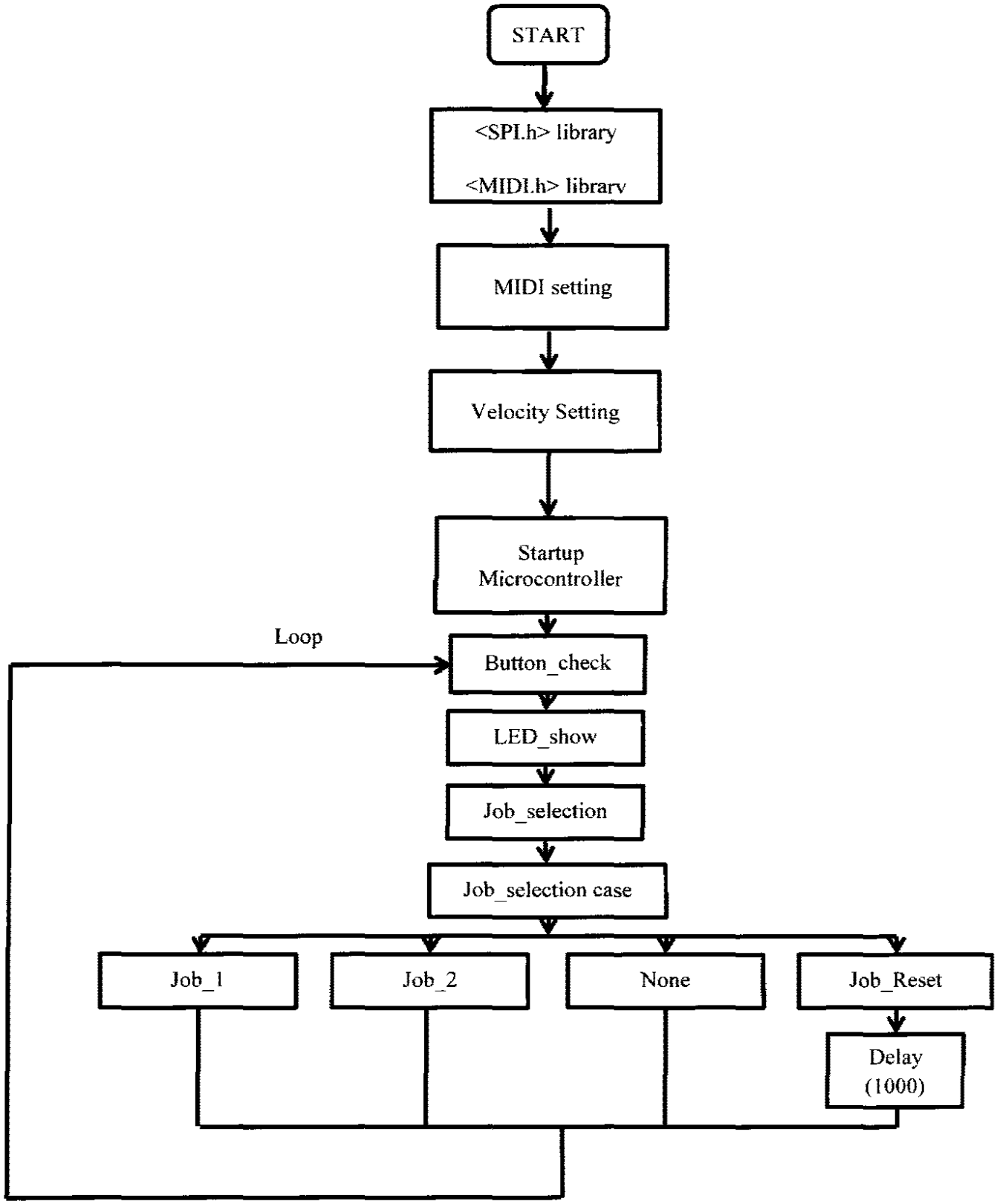


Figure 4.1: MAC System

4.2.2 Button Check Program

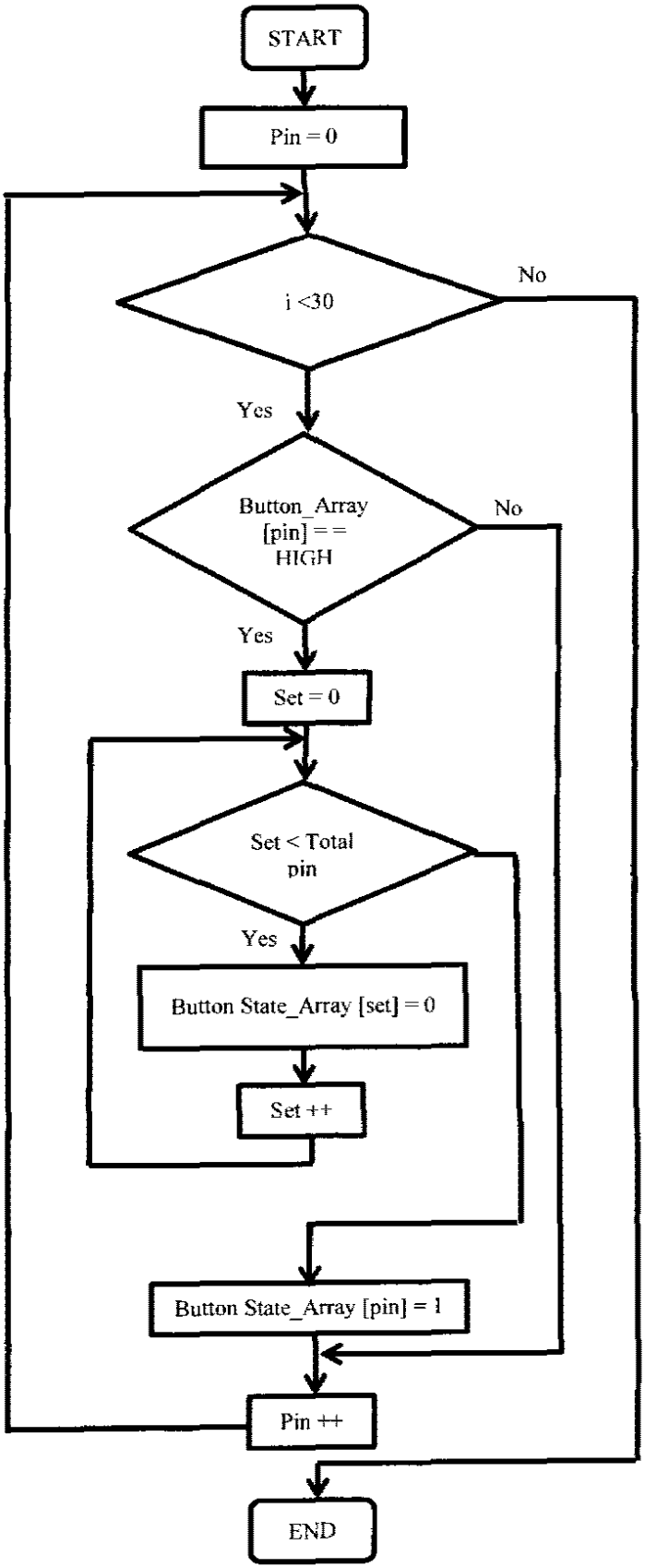


Figure 4.2: Process flow for Button Check Program

4.2.2.1 Program in Arduino for Button Check

```
Button_Check();
// show the result of button check using led.
void Button_Check()
{
 for (int Pin = 0; Pin < TotalPin; Pin++)
{
   if (digitalRead(button_Pin_Array [Pin]) == HIGH)
{
    for (int set = 0; set < TotalPin; set++)
{
      buttonState_Array [set] = 0;
}
    buttonState_Array [Pin] = 1;
   }
  }
}
```

4.2.3 Led_Show Program

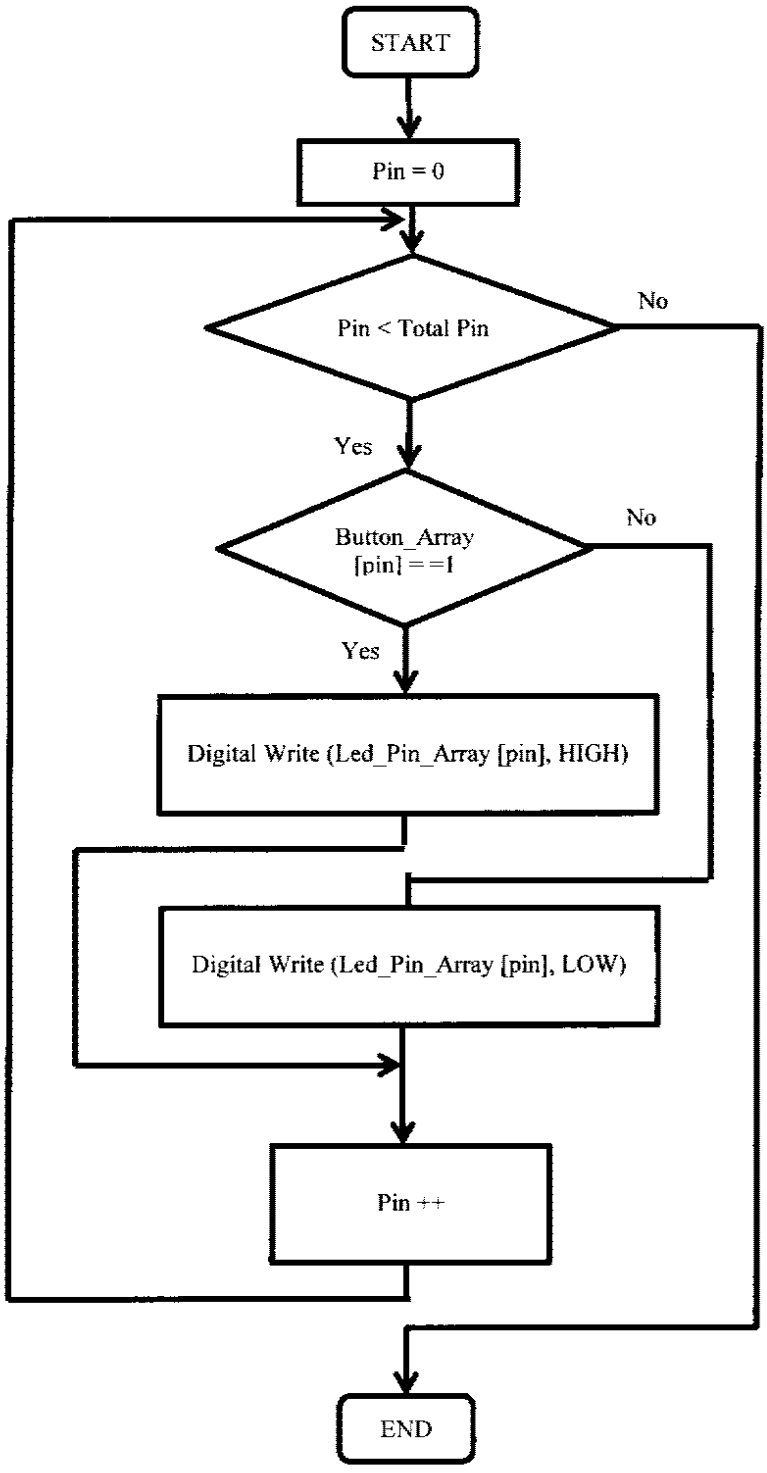


Figure 4.3: Process flow for Led Show Program

4.2.3.1 Program in Arduino for LED_show

```
LED_Show();
void LED_Show()
{
 for (int Pin = 0; Pin < TotalPin; Pin++)
{
  if (buttonState_Array [Pin] == 1)
{
// turn LED on:
    digitalWrite(led_Pin_Array [Pin], HIGH);
}
  else
{
// turn LED off:
    digitalWrite(led_Pin_Array [Pin], LOW);
}
```

4.2.4 Job Selector Flow Chart

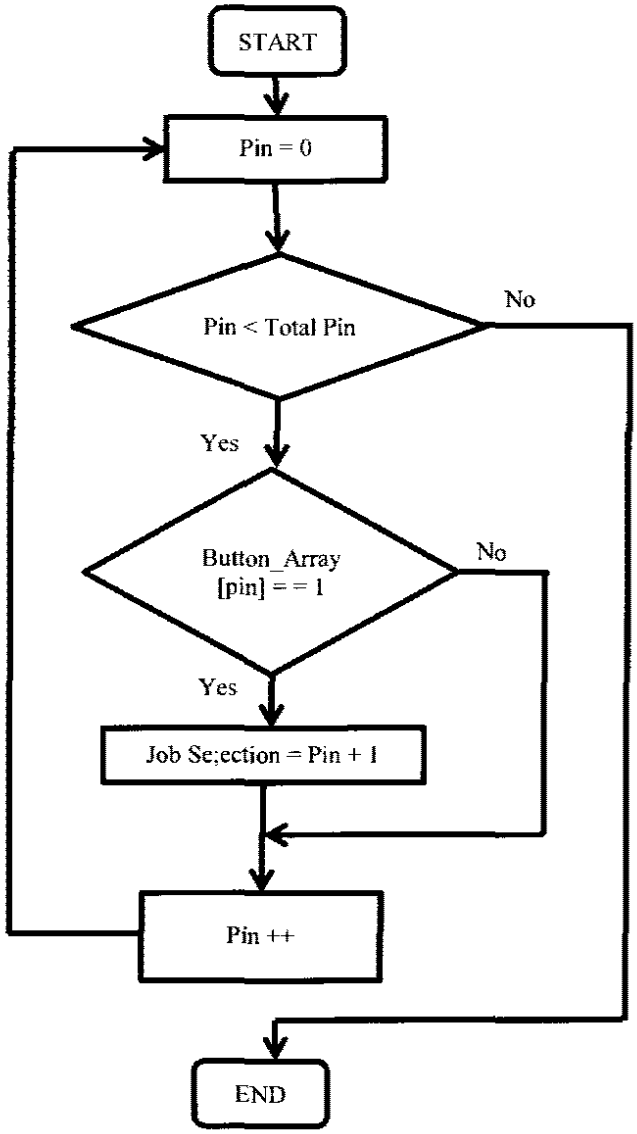


Figure 4.4: Process flow for Job Selector Program

4.2.4.1 Program in Arduino for Job Selector

```
Job_Selector();
  switch (Job_Selection)
{
  case 1:
       Job_01();
       break;
  case 2:
       Job_02();
       break;
  case 3:
   //do something when variable equals 2
  break;
  case 4:
       Job_Reset();
       delay(1000);
  break;
}
```

4.2.5 Job 01 (MIDI Files)

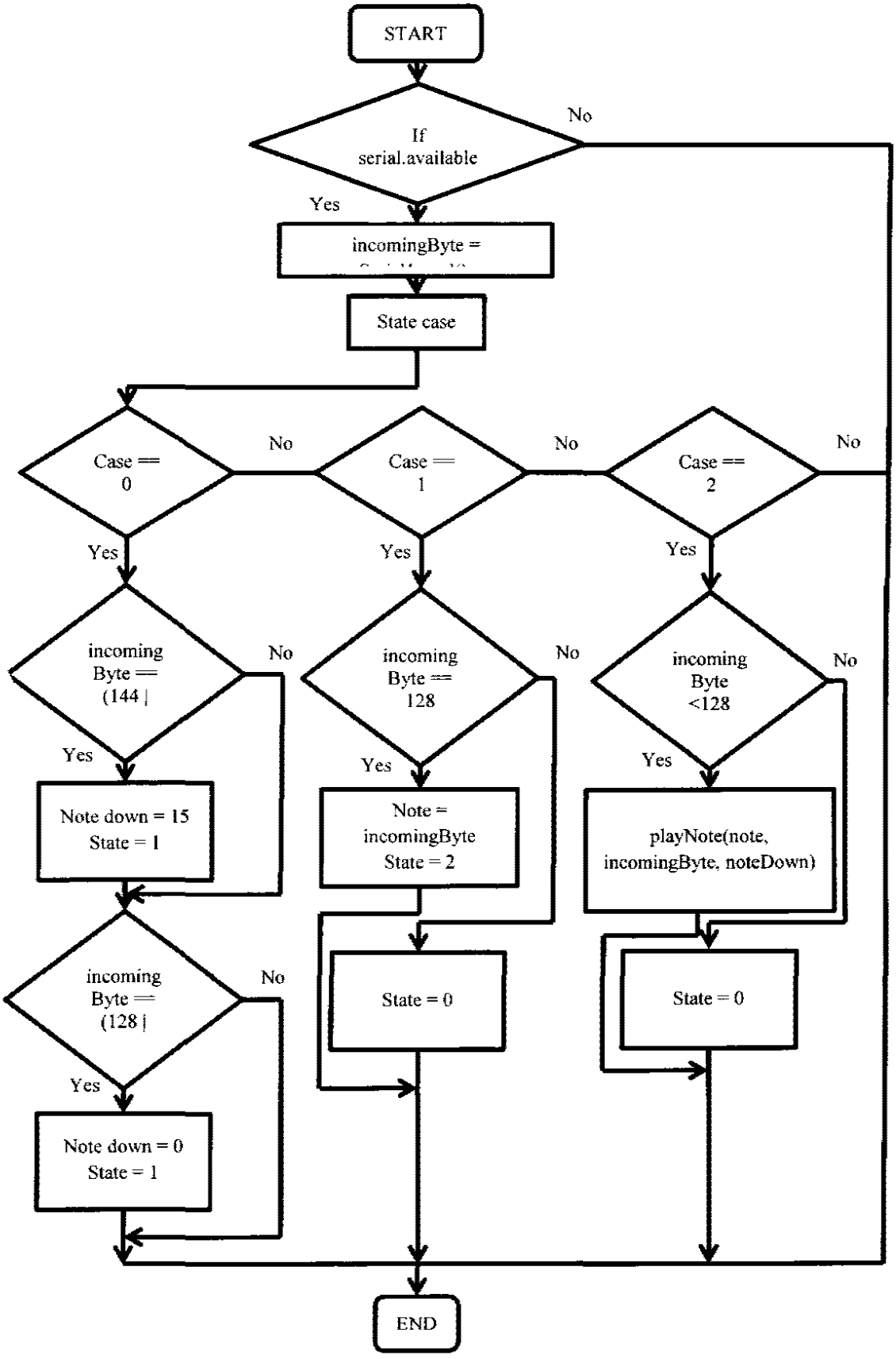


Figure 4.5: Job 01 – Read and decode MIDI Files

4.2.5.1 Program inArduino for Job 01(MIDI Files)

```
void Job_01()
{
 if (Serial1.available() > 0)
{
  incomingByte = Serial1.read();
  switch (state){
   case 0:
   if (incomingByte== (144 | channel))
{
    noteDown = 15;
    state=1;
}
   if (incomingByte== (128 | channel))
{
    noteDown = 0;
    state=1;
}
   break;
   case 1:
      if(incomingByte < 128)
{
    note=incomingByte;
    state=2;
}
   Else
{
    state = 0;
}
   break;
   case 2:
      if(incomingByte < 128)
{
    playNote(note, incomingByte, noteDown);
}
```

4.2.6 Job 02 (MIDI Message)

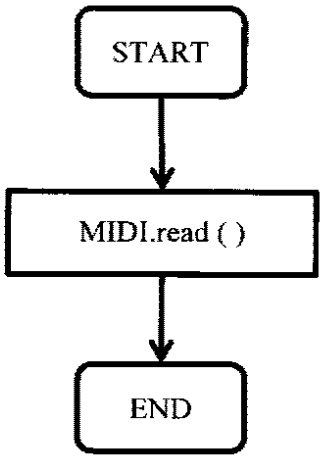


Figure 4.6: Job 02 – Read and decode MIDI Message

4.2.6.1 Program in Arduino for Job 02 (MIDI Message)

```
void Job_02()
{
 MIDI.read();
}
void playNote(byte note, byte velocity, int down)
{
 if ((down == 15) && (velocity == 0))
{
    down = 0;
}
 if(note>=baseNote && note<(baseNote + 36))
{
   byte now_note=playNoteArray[note-baseNote];
   byte now_latch=playLatchArray[note-baseNote];
   write_serial(now_note, down, now_latch);
   digitalWrite(LED, HIGH);
}
}
```

4.3 DEVELOPMENT OF CIRCUIT DESIGN

CadSoft EAGLE PCB Design Software is used to design the MIDI Shield circuit for this project. The MIDI Shield circuit provides an opt-isolated MIDI-IN port as well as a MIDI-OUT port. The MIDI Shield circuit can be mounted directly on top of an Arduino, connecting the MIDI-IN to the Arduino's hardware RX (Receiver) pin and the MIDI-OUT to TX (Transmitter). This MIDI Shield circuit was specially design that can choose two jobs and have a button reset.

4.3.1 COMPONENT AND EQUIPMENT

MIDI Shield circuit in figure 4.1 is used to communicate between devices with Arduino microcontroller. The MIDI Shield provides an opt-isolated MIDI-IN port as well as a MIDI-OUT port. The MIDI Shield can be mounted directly on top of an Arduino, connecting the MIDI-IN to the Arduino's hardware RX pin and the MIDI-OUT to TX. In table 4.1 shows the listing of the component used in this circuit

Component	Quantity
Opt-isolator 6n138	1
Diode 1n4001	1
5 PIN DIN (Female)	1
Resistor 200 Ω	4
7404	3
Push Button	4
Fius	1
Light Emitting Diode (LED)	3
Voltage regulator LN7805	1

Table 4.1: MIDI Shield circuit component

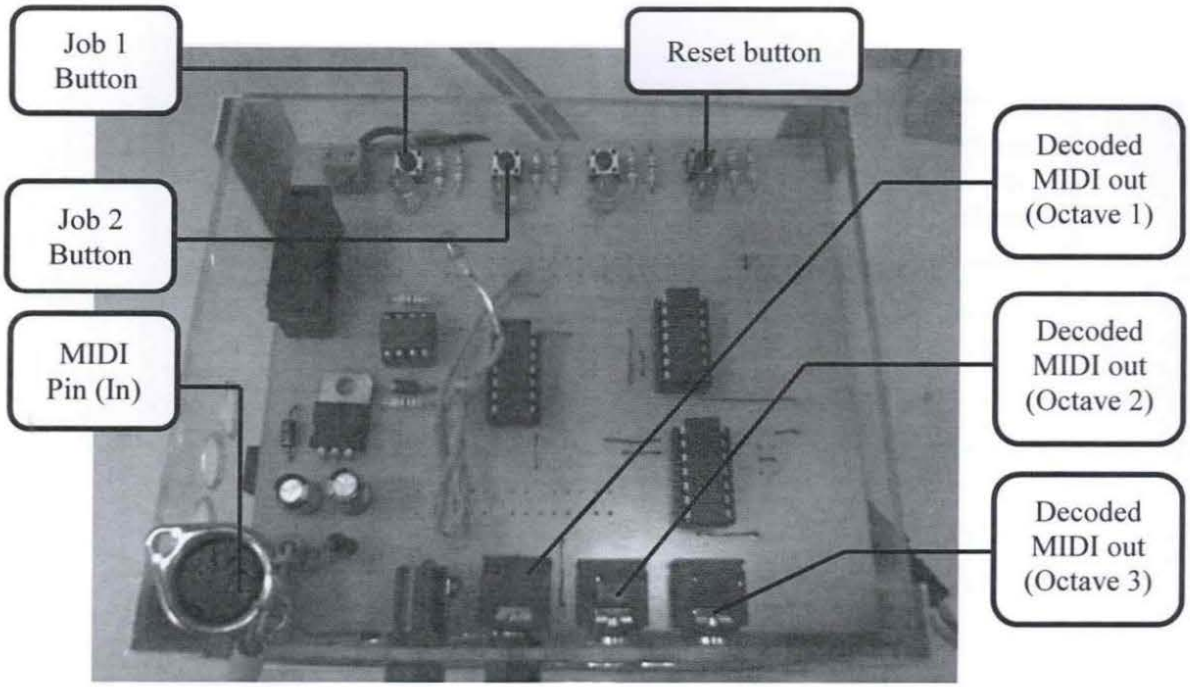


Figure 4.1: MIDI Shield circuit

4.4 RESULT AND PERFORMANCE ANALYSIS

4.4.1 Transmission Efficiency MIDI Signal

The output signal of the MIDI Decoder is read by the motor driver circuit. The signal is sent out to the correct motor driver through proper address decoding. This proved when sending a MIDI note for example note C3 to the microcontroller thru MIDI Shield. Then microcontroller will decode the MIDI note and send to the motor driver follow by the MIDI note send. For note C3, the motor driver address is 0001, so motor driver with 0001 will be running. Figure 4.2 below show the Motor Driver circuit use in this project.

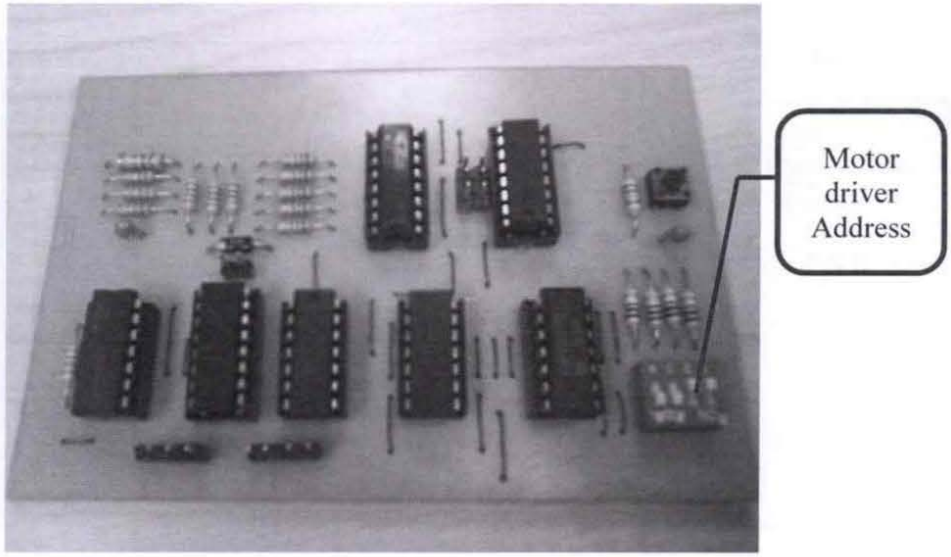


Figure 4.2: Motor Driver

4.4.2 Latency in Opt-Isolator

Figure 4.3: Latency after flow thru Opt-isolator is about 8.735µs.

Figure 4.3 above show the latency for MIDI message after flow in MIDI shield thru Opt-isolator from MIDI Keyboard or PC Based. The latency is about 8.735 µs.

4.4.3 Latency in microcontroller

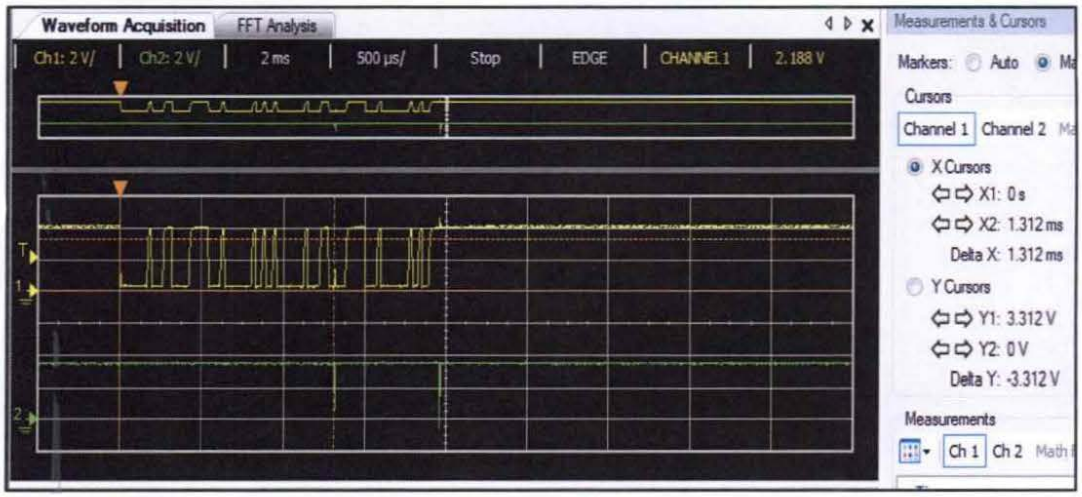


Figure 4.4: Latency in microcontroller after decodes the MIDI signal is 1.312ms

After decode MIDI in microcontroller, the latency MIDI signal is 1.312ms as shown in figure 4.4. This latency tells that to decode the MIDI signal in microcontroller is take less than 2ms.

4.4.4 Total Latency for MAC

$$\sum \text{latency} = \text{Latency Opto} + \text{Latency microcontroller}$$

$$\sum \text{latency} = 8.735\mu\text{s} + 1.312 \text{ ms}$$

$$\sum \text{latency} = 1.320735 \text{ ms}$$

In the audio world, "latency" is another word for "delay". It's the time it takes for midi into the device, for the decoded the midi, for the motor drive to shake the angklung, and for the sound to reach your ear.

Our brain is wired so that it doesn't notice if sounds are delayed 3 to 10 milliseconds (ms). This device produces latency at 1.320735 milliseconds (ms). This make the system fast enough to decode the midi signal from music instrument and play the angklung music instrument in real time processing.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This final chapter represents about conclusion and recommendation for the project. In this chapter will discuss mainly about the conclusion of the project, concluding all the process that involved. Besides that this chapter also contains recommendation about the project. So for this recommendation it can make improvement about the project in the next semester.

5.2 Conclusion

One could come to the conclusion that the MIDI Decoder in its entirety was a success. The problem of controlling the Angklung with MIDI messages was solved by creating a simple system to receive MIDI messages, decode them, and control the Motor driver according to them in real time.

5.3 Future Development

For future improvement, it is suggested to use Android to be a user interface to send a MIDI message to MIDI Decoder to decode. Besides controlling using Computer and MIDI Piano as user interface, develop a controller using android via Bluetooth transmission. With this additional, it can control an Angklung wirelessly.

REFERENCES

Professor Kuo-Huang Han. *Can You Shake It? The Angklung of Southeast Asia* Location: Northern Illinois University.

Floyd. *Digital Fundamental* 10th edition. Published by Person International Edition

Adam Koss. *A Comparison Of The Graphs Of The Chromatic And Diatonic Scale*

Cadance. (2000). *PSpice User's Guide*. Published by Cadence Design Systems, Inc.

Music in Sequence: Complete guide to MIDI **by William Lloyd and Paul Terry** (Jan 1, 1996)

MIDI Technical Brainwashing Center 2009c. Note-On. Read 30.8.2009. http://home.roadrunner.com/~jgglatt/tech/midispec/noteon.htm

MIDI Technical Brainwashing Center 2009d. Note-Off. Read 30.8.2009. http://home.roadrunner.com/~jgglatt/tech/midispec/noteoff.htm

Computer music in C / Phil Winsor & Gene DeLisa. Published by Blue Ridge Summit, PA : TAB Books (Windcrest label), c1991.

APPENDIX A

MIDI Decoder Source Code

```
#include <SPI.h>
#include <MIDI.h>
int latch;
byte incomingByte;
byte note;
byte velocity;
int noteDown = 0;
int state=0;

// setting for Angklung
int baseNote = 60;
byte playNoteArray[] =
{1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4,5,6,7,8,9,10,11,12};
byte playLatchArray[] =
{10,10,10,10,10,10,10,10,10,10,10,10,4,4,4,4,4,4,4,4,4,4,4,4,52,52,52,52,52,52,52,5
2,52,52,52,52};
int channel = 1;
int channel_selection = 1;

// constants won't change. They're used here to
// set pin numbers:
const int button_Pin_Array [] = {22,25,29,33};        // the number of the
pushbutton pin.
const int led_Pin_Array [] = {23,27,31,35};     // the number of the LED pin.
const int TotalPin = 4;
const int LED =  5;
// variables will change:
int buttonState_Array [] = {1,0,0,0};        // variable for reading the pushbutton
status.
int Job_Selection = 1;                  // variable for job to do.
```

```
void setup() {

  state = 0;
  Serial1.begin(31250);
  // Initiate MIDI communications
  MIDI.begin(channel_selection);

  SPI.begin(10);
  SPI.begin(4);
  SPI.begin(52);
  SPI.setDataMode(10,SPI_MODE0);
  SPI.setDataMode(4,SPI_MODE0);
  SPI.setDataMode(52,SPI_MODE0);
  SPI.setBitOrder(10,MSBFIRST);
  SPI.setBitOrder(4,MSBFIRST);
  SPI.setBitOrder(52,MSBFIRST);
  SPI.setClockDivider(10,21);
  SPI.setClockDivider(4,21);
  SPI.setClockDivider(52,21);
  pinMode(latch,OUTPUT);

  MIDI.setHandleNoteOn(HandleNoteOn);
  MIDI.setHandleNoteOff(HandleNoteOff);

  for (int Pin = 0; Pin < TotalPin; Pin++) {
    // initialize the LED pin as an output:
    pinMode(led_Pin_Array [Pin], OUTPUT);
    // initialize the pushbutton pin as an input:
    pinMode(button_Pin_Array [Pin], INPUT);
  }
  pinMode(LED, OUTPUT);
  digitalWrite(LED,LOW);

}

void loop(){
  // read the state of the pushbutton value:
  // check if the pushbutton is pressed.
  Button_Check();
// show the result of button check using led.
```

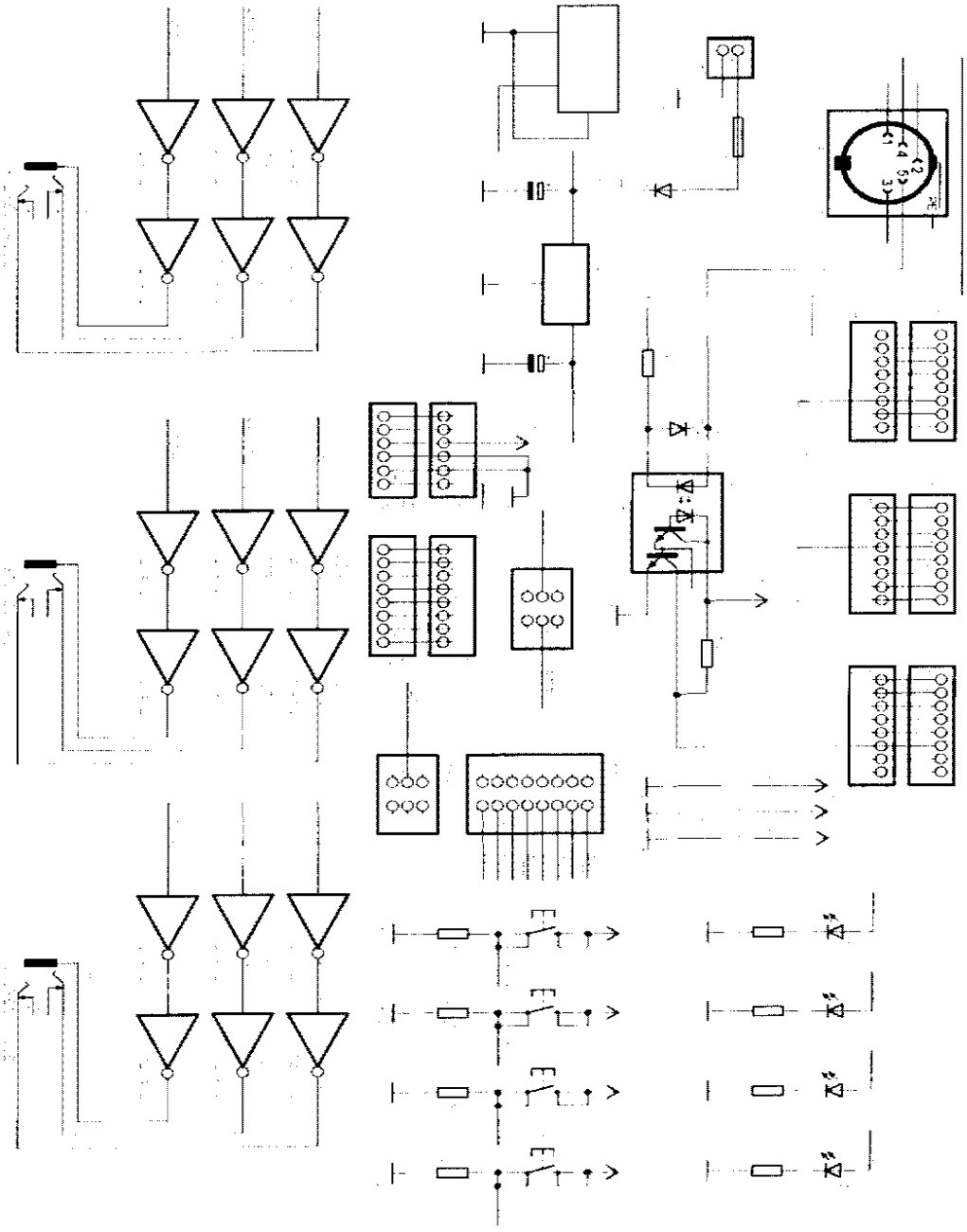
```
LED_Show();
 Job_Selector();

 switch (Job_Selection) {
  case 1:
   Job_01();
   break;
  case 2:
   Job_02();
   break;
  case 3:
   //do something when var equals 2
  break;
  case 4:
   Job_Reset();
   delay(1000);
  break;
 }


}
void Button_Check(){
 for (int Pin = 0; Pin < TotalPin; Pin++) {
  if (digitalRead(button_Pin_Array [Pin]) == HIGH){
   for (int set = 0; set < TotalPin; set++) {
    buttonState_Array [set] = 0;
   }
   buttonState_Array [Pin] = 1;
  }
 }
}
void LED_Show(){
 for (int Pin = 0; Pin < TotalPin; Pin++) {
  if (buttonState_Array [Pin] == 1) {
   // turn LED on:
   digitalWrite(led_Pin_Array [Pin], HIGH);
  }
  else {
 // turn LED off:
   digitalWrite(led_Pin_Array [Pin], LOW);
}
```

```
  }
 }
 void Job_Selector(){
   for (int Pin = 0; Pin < TotalPin; Pin++) {
    if (buttonState_Array [Pin] == 1) {
     Job_Selection = Pin + 1;
    }
   }
 }
 void Job_01(){
  if (Serial1.available() > 0)
  {
   incomingByte = Serial1.read();
   switch (state){
    case 0:
    if (incomingByte== (144 | channel)){
     noteDown = 15;
     state=1;
    }
    if (incomingByte== (128 | channel)){
     noteDown = 0;
     state=1;
    }
    break;
    case 1:
    if(incomingByte < 128){
     note=incomingByte;
     state=2;
    }
    else{
     state = 0;
    }
    break;
    case 2:
    if(incomingByte < 128){
     playNote(note, incomingByte, noteDown);
    }
else{
     state = 0;
  }
    break;
```

```
    }
  }
}
void Job_02(){
 MIDI.read();
}
void playNote(byte note, byte velocity, int down){
 if ((down == 15) && (velocity == 0)){
    down = 0;
 }
 if(note>=baseNote && note<(baseNote + 36)){
  byte now_note=playNoteArray[note-baseNote];
  byte now_latch=playLatchArray[note-baseNote];
  write_serial(now_note, down, now_latch);
  digitalWrite(LED, HIGH);
 }
}
void write_serial(int value, int speeed, int latch){
 byte serial = value*16+speeed;
 SPI.transfer(latch,0);
 SPI.transfer(latch,serial);
}
void HandleNoteOn(byte channel, byte note, byte velocity) {
 noteDown = 15;
 playNote(note, velocity, noteDown);
 digitalWrite(LED, LOW);
}
void HandleNoteOff(byte channel, byte note, byte velocity) {
 noteDown = 0;
 playNote(note, velocity, noteDown);
 digitalWrite(LED, LOW);
}
void Job_Reset(){
 for (byte note_reset = baseNote; note_reset < (baseNote + 36); note_reset++) {
  playNote(note_reset, 0, 0);
}
 for (int set = 0; set < TotalPin; set++) {
  buttonState_Array [set] = 0;
}
 buttonState_Array [0] = 1;
}
```

APPENDIX B

MIDI Shield Circuit



APPENDIX C

Expended MIDI message list

Expanded Messages List (Status Bytes)

The following table lists Status Bytes in binary numerical order (adapted from "MIDI by the Numbers" by D. Valenti, Electronic Musician 2/88, and updated 1995 By the MIDI Manufacturers Association.)

Table 2: Expanded Status Bytes List

STATUS BYTE		DATA BYTES	
1st Byte Value Binary |Hex| Dec	**Function**	**2nd Byte**	**3rd Byte**
10000000= 80= 128	Chan 1 Note off	Note Number (0-127)	Note Velocity (0-127)
10000001= 81= 129	Chan 2 Note off	Note Number (0-127)	Note Velocity (0-127)
10000010= 82= 130	Chan 3 Note off	Note Number (0-127)	Note Velocity (0-127)
10000011= 83= 131	Chan 4 Note off	Note Number (0-127)	Note Velocity (0-127)
10000100= 84= 132	Chan 5 Note off	Note Number (0-127)	Note Velocity (0-127)
10000101= 85= 133	Chan 6 Note off	Note Number (0-127)	Note Velocity (0-127)
10000110= 86= 134	Chan 7 Note off	Note Number (0-127)	Note Velocity (0-127)
10000111= 87= 135	Chan 8 Note off	Note Number (0-127)	Note Velocity (0-127)
10001000= 88= 136	Chan 9 Note off	Note Number (0-127)	Note Velocity (0-127)
10001001= 89= 137	Chan 10 Note off	Note Number (0-127)	Note Velocity (0-127)
10001010= 8A= 138	Chan 11 Note off	Note Number (0-127)	Note Velocity (0-127)
10001011= 8B= 139	Chan 12 Note off	Note Number (0-127)	Note Velocity (0-127)
10001100= 8C= 140	Chan 13 Note off	Note Number (0-127)	Note Velocity (0-127)
10001101= 8D= 141	Chan 14 Note off	Note Number (0-127)	Note Velocity (0-127)
10001110= 8E= 142	Chan 15 Note off	Note Number (0-127)	Note Velocity (0-127)
10001111= 8F= 143	Chan 16 Note off	Note Number (0-127)	Note Velocity (0-127)
10010000= 90= 144	Chan 1 Note on	Note Number (0-127)	Note Velocity (0-127)
10010001= 91= 145	Chan 2 Note on	Note Number (0-127)	Note Velocity (0-127)
10010010= 92= 146	Chan 3 Note on	Note Number (0-127)	Note Velocity (0-127)
10010011= 93= 147	Chan 4 Note on	Note Number (0-127)	Note Velocity (0-127)

10010100= 94= 148	Chan 5 Note on	Note Number (0-127)	Note Velocity (0-127)
10010101= 95= 149	Chan 6 Note on	Note Number (0-127)	Note Velocity (0-127)
10010110= 96= 150	Chan 7 Note on	Note Number (0-127)	Note Velocity (0-127)
10010111= 97= 151	Chan 8 Note on	Note Number (0-127)	Note Velocity (0-127)
10011000= 98= 152	Chan 9 Note on	Note Number (0-127)	Note Velocity (0-127)
10011001= 99= 153	Chan 10 Note on	Note Number (0-127)	Note Velocity (0-127)
10011010= 9A= 154	Chan 11 Note on	Note Number (0-127)	Note Velocity (0-127)
10011011= 9B= 155	Chan 12 Note on	Note Number (0-127)	Note Velocity (0-127)
10011100= 9C= 156	Chan 13 Note on	Note Number (0-127)	Note Velocity (0-127)
10011101= 9D= 157	Chan 14 Note on	Note Number (0-127)	Note Velocity (0-127)
10011110= 9E= 158	Chan 15 Note on	Note Number (0-127)	Note Velocity (0-127)
10011111= 9F= 159	Chan 16 Note on	Note Number (0-127)	Note Velocity (0-127)
10100000= A0= 160	Chan 1 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10100001= A1= 161	Chan 2 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10100010= A2= 162	Chan 3 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10100011= A3= 163	Chan 4 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10100100= A4= 164	Chan 5 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10100101= A5= 165	Chan 6 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10100110= A6= 166	Chan 7 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10100111= A7= 167	Chan 8 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10101000= A8= 168	Chan 9 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10101001= A9= 169	Chan 10 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10101010= AA= 170	Chan 11 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10101011= AB= 171	Chan 12 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10101100= AC= 172	Chan 13 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10101101= AD= 173	Chan 14 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10101110= AE= 174	Chan 15 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10101111= AF= 175	Chan 16 Polyphonic Aftertouch	Note Number (0-127	Pressure (0-127)
10110000= B0= 176	Chan 1 Control/Mode Change		
10110001= B1= 177	Chan 2 Control/Mode Change		
10110010= B2= 178	Chan 3 Control/Mode Change		
10110011= B3= 179	Chan 4 Control/Mode Change		
10110100= B4= 180	Chan 5 Control/Mode Change		

10110101= B5= 181	Chan 6 Control/Mode Change		
10110110= B6= 182	Chan 7 Control/Mode Change		
10110111= B7= 183	Chan 8 Control/Mode Change		
10111000= B8= 184	Chan 9 Control/Mode Change		
10111001= B9= 185	Chan 10 Control/Mode Change		
10111010= BA= 186	Chan 11 Control/Mode Change		
10111011= BB= 187	Chan 12 Control/Mode Change		
10111100= BC= 188	Chan 13 Control/Mode Change		
10111101= BD= 189	Chan 14 Control/Mode Change		
10111110= BE= 190	Chan 15 Control/Mode Change		
10111111= BF= 191	Chan 16 Control/Mode Change		
11000000= C0= 192	Chan 1 Program Change	Program # (0-127)	none
11000001= C1= 193	Chan 2 Program Change	Program # (0-127)	none
11000010= C2= 194	Chan 3 Program Change	Program # (0-127)	none
11000011= C3= 195	Chan 4 Program Change	Program # (0-127)	none
11000100= C4= 196	Chan 5 Program Change	Program # (0-127)	none
11000101= C5= 197	Chan 6 Program Change	Program # (0-127)	none
11000110= C6= 198	Chan 7 Program Change	Program # (0-127)	none
11000111= C7= 199	Chan 8 Program Change	Program # (0-127)	none
11001000= C8= 200	Chan 9 Program Change	Program # (0-127)	none
11001001= C9= 201	Chan 10 Program Change	Program # (0-127)	none
11001010= CA= 202	Chan 11 Program Change	Program # (0-127)	none
11001011= CB= 203	Chan 12 Program Change	Program # (0-127)	none
11001100= CC= 204	Chan 13 Program Change	Program # (0-127)	none
11001101= CD= 205	Chan 14 Program Change	Program # (0-127)	none
11001110= CE= 206	Chan 15 Program Change	Program # (0-127)	none
11001111= CF= 207	Chan 16 Program Change	Program # (0-127)	none
11010000= D0= 208	Chan 1 Channel Aftertouch	Pressure (0-127)	none
11010001= D1= 209	Chan 2 Channel Aftertouch	Pressure (0-127)	none
11010010= D2= 210	Chan 3 Channel Aftertouch	Pressure (0-127)	none
11010011= D3= 211	Chan 4 Channel Aftertouch	Pressure (0-127)	none
11010100= D4= 212	Chan 5 Channel Aftertouch	Pressure (0-127)	none
11010101= D5= 213	Chan 6 Channel Aftertouch	Pressure (0-127)	none

11010110= D6= 214	Chan 7 Channel Aftertouch	Pressure (0-127)	none
11010111= D7= 215	Chan 8 Channel Aftertouch	Pressure (0-127)	none
11011000= D8= 216	Chan 9 Channel Aftertouch	Pressure (0-127)	none
11011001= D9= 217	Chan 10 Channel Aftertouch	Pressure (0-127)	none
11011010= DA= 218	Chan 11 Channel Aftertouch	Pressure (0-127)	none
11011011= DB= 219	Chan 12 Channel Aftertouch	Pressure (0-127)	none
11011100= DC= 220	Chan 13 Channel Aftertouch	Pressure (0-127)	none
11011101= DD= 221	Chan 14 Channel Aftertouch	Pressure (0-127)	none
11011110= DE= 222	Chan 15 Channel Aftertouch	Pressure (0-127)	none
11011111= DF= 223	Chan 16 Channel Aftertouch	Pressure (0-127)	none
11100000= E0= 224	Chan 1 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100001= E1= 225	Chan 2 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100010= E2= 226	Chan 3 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100011= E3= 227	Chan 4 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100100= E4= 228	Chan 5 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100101= E5= 229	Chan 6 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100110= E6= 230	Chan 7 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100111= E7= 231	Chan 8 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101000= E8= 232	Chan 9 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101001= E9= 233	Chan 10 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101010= EA= 234	Chan 11 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101011= EB= 235	Chan 12 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101100= EC= 236	Chan 13 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101101= ED= 237	Chan 14 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101110= EE= 238	Chan 15 Pitch Wheel Control	Pitch Wheel LSB (0-	Pitch Wheel MSB (0-127)

		127)	
11101111= EF= 239	Chan 16 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11110000= F0= 240	System Exclusive	**	**
11110001= F1= 241	MIDI Time Code Qtr. Frame		
11110010= F2= 242	Song Position Pointer	LSB	MSB
11110011= F3= 243	Song Select (Song #)	(0-127)	none
11110100= F4= 244	Undefined (Reserved)	---	---
11110101= F5= 245	Undefined (Reserved)	---	---
11110110= F6= 246	Tune request	none	none
11110111= F7= 247	End of SysEx (EOX)	none	none
11111000= F8= 248	Timing clock	none	none
11111001= F9= 249	Undefined (Reserved)	---	---
11111010= FA= 250	Start	none	none
11111011= FB= 251	Continue	none	none
11111100= FC= 252	Stop	none	none
11111101= FD= 253	Undefined (Reserved)	---	---
11111110= FE= 254	Active Sensing	none	none
11111111= FF= 255	System Reset	none	none

** Note: System Exclusive (data dump) 2nd byte= Vendor ID (or Universal Exclusive) followed by more data bytes and ending with EOX.