JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor in Technology

in

ELECTRONICSS AND COMMUNICATION ENGINEERING

on

DIGITAL SYNTHESIZER

Under guidance of

MR. DHEERAJ KUMAR SHARMA

Asst. Professor

(Department of Electronics and Communication Engineering)

By:-

Aayush garg 101037

CERTIFICATE

This is to certify that the work titled "**DIGITAL SYNTHESIZER** " submitted by Aayush Garg in partial fulfillment for the award of degree of B.Tech ELECTRONICS AND COMMUNIC Engineering of Jaypee University of Information technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other university or institute for the award of this or any other degree or diploma.

Mr. Dheeraj kumar Sharma

Assistant Professor (Grade –II)

Department of Electronics and Communication Engineering

ACKNOWLEDGMENT

We would like to express our deepest appreciation to our project guide Asst. Prof. **Mr. Dheeraj Kumar Sharma** (Department. Of Electronics and communication Engineering) for his guidance for the duration of this project. His efforts and his guidelines were always an opportunity for us to learn and played an important role in the completion of this project.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

A digital music synthesizer has been build and an external device is used to control it.

The synthesizer is programmed in a powerful programming language platform Chuck and can be used to play any kind of sound. Chuck is a programming language designed specifically for real-time sound synthesis and music creation. “Real-time” means that Chuck synthesizes the sound as you are hearing it (rather than just playing back a sound file), often in response to signals and gestures from the “outside world”.

The external device is a “CAPACITIVE TOUCH MIDI SENSOR”. Capacitive touch sensor is build through Arduino uno. Arduino uno is a microcontroller board based on ATmega328. To send midi messages through serial outputs of arduino uno , a software “HAIRLESS MIDI” is used.

Finally a virtual port is created in chuck to receive midi messages and this is done through a software “LOOP MIDI”.

CHAPTER-1

INTRODUCTION

1.2 SYNTHESIZERS

A sound synthesizer (often abbreviated as “synthesizer” or “synth”) is an electronic musical instrument capable of producing a wide range of sounds. Synthesizers may either imitate other instruments (“imitative synthesis”) or generate new timbres. They can be played (controlled) via a variety of different input devices (including keyboards, music sequencers and instrument controllers). Synthesizers generate electric signals (waveforms), and can finally be converted to sound through loudspeakers or headphones.

Synthesizers are often controlled with a piano-style keyboard. Other forms of controllers resemble fingerboards, guitars (guitar synthesizer), violins, wind instruments (wind controller), drums and percussions (electronic drum), etc. Synthesizers without built-in controllers are often called sound modules, and are controlled via MIDI and other methods.

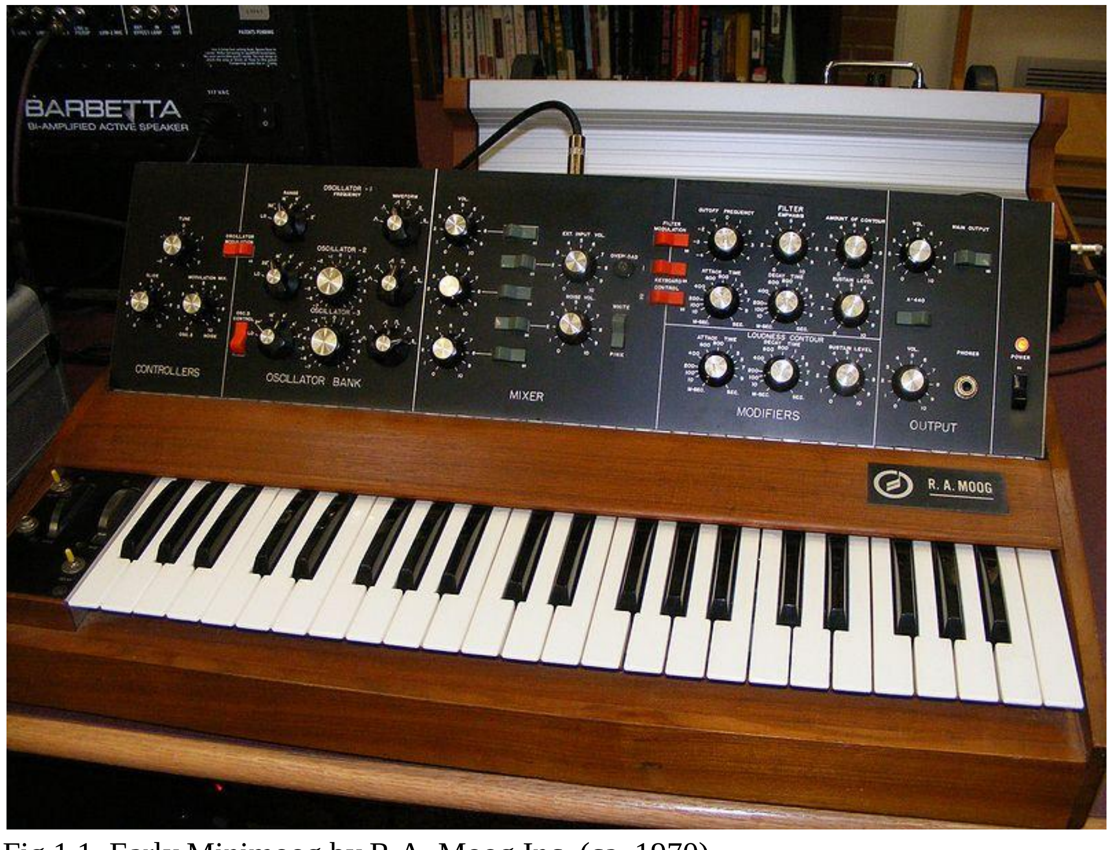


Fig 1.1 Early Minimoog by R.A. Moog Inc. (ca. 1970)

1.3 TYPES OF SYNTHESIZERS

1.4 ADDITIVE SYNTHESIS

The technique assumes that any periodic waveform can be modelled as a sum sinusoids at various amplitude envelopes and time-varying frequencies. It works by summing up individually generated sinusoids in order to form a specific sound.

Additive synthesis is a sound synthesis technique that creates timbre by adding sine waves together. The timbre of musical instruments can be considered in the light of Fourier theory to consist of multiple harmonic or inharmonic partials or overtones. Each partial is a sine wave of different frequency and amplitude that swells and decays over time. Additive synthesis most directly generates sound by adding the output of multiple sine wave generators. Alternative implementations may use pre-computed wavetables or the inverse Fast Fourier transform. A very powerful and flexible technique but it is difficult to control manually and is computationally expensive. Musical timbres: composed of dozens of time-varying partials. It requires dozens of oscillators, noise generators and envelopes to obtain convincing simulations of acoustic sounds. The specification and control of the parameter values for these components are difficult and time consuming.

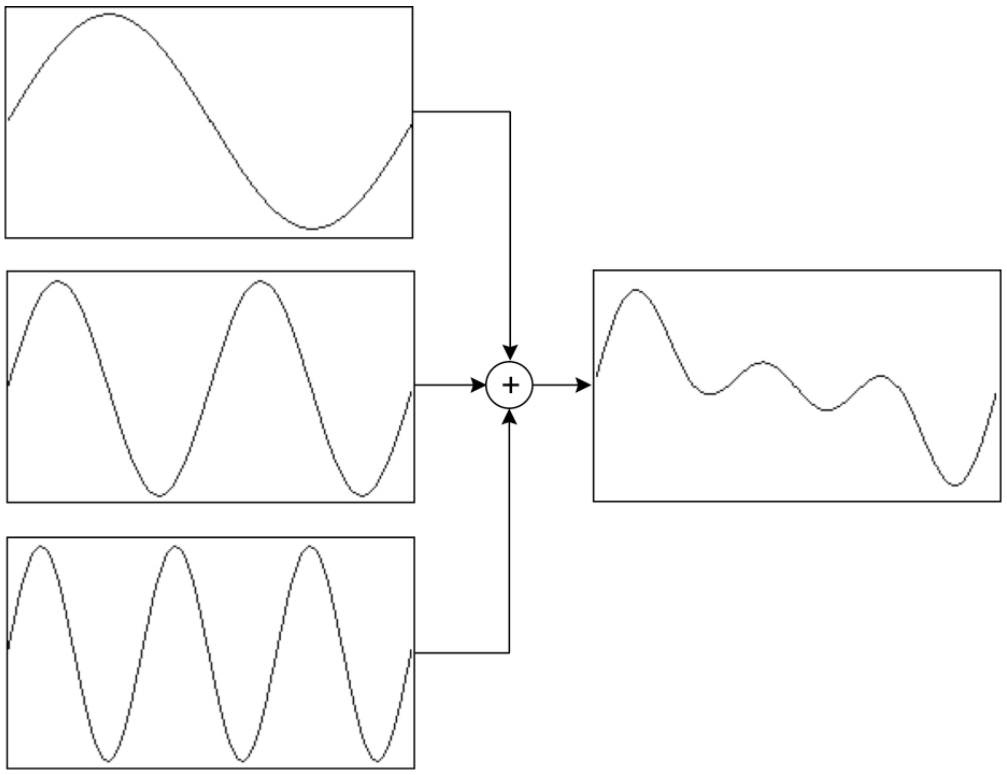


FIG 1.2

1.2.2 SUBTRACTIVE SYNTHESIS

Most musical instruments can be modeled as a resonating chamber stimulated by acoustic waveforms with certain spectral and temporal properties Subtractive synthesis is based upon the principle that the 8odeling of an instrument is determined by two main components: excitation source and resonator. The role of excitation is to produce a raw signal that will be shaped by the resonator. A variety of spectra can be obtained by varying the acoustic properties of the resonator. The technique is called "subtractive" because the resonator alters the spectrum of the excitation source by subtracting unwanted partials of its spectrum while favoring the resonation of others. From a signal processing point of view, the resonator acts as a filter (or bank of filters) applied to an excitation signal. Subtractive synthesis has been successfully used to model percussion-like instruments and the human voice.

The implementation of a subtractive synthesizer generally uses a signal generator with a rich spectrum to act as an excitation source and a bank of filters to simulate a resonating chamber. Excitation of two types of signals are commonly used: noise and pulse generators. Both are rich signals; i.e., in terms of variety of partials. Noise generators produce a large number of random partials within a broad bandwidth. Pulse generators produce periodic waveforms, or train of pulses at specific frequencies, with high energy partials. The spectrum of a pulse is determined by the ratio of the pulse width to the period of the signal, the smaller the ratio, the narrower the pulse and therefore the higher the energy of the high-frequency partials.

1.2.3 FM SYNTHESIS

FM synthesis (frequency modulation synthesis) is a process that usually involves the use of at least two signal generators (sine-wave oscillators, commonly referred to as "operators" in FM-only synthesizers) to create and modify a voice. Often, this is done through the analog or digital generation of a signal that modulates the tonal and amplitude characteristics of a base carrier signal. FM synthesis was pioneered by John Chowning , who patented the idea and sold it to Yamaha. Unlike the exponential relationship between voltage-in-to-frequency-out and multiple waveforms in classical 1-volt-per-octave synthesizer oscillators, Chowning-style FM synthesis uses a linear voltage-in-to-frequency-out relationship and sine-wave oscillators. The resulting complex waveform may have many component frequencies, and there is no requirement that they all bear a harmonic relationship. Sophisticated FM synths such as the Yamaha DX-7 series can have 6 operators per voice; some synths with FM can also often use filters and variable amplifier types to alter the signal's characteristics into a sonic voice that either roughly imitates acoustic instruments or creates sounds that are unique.

FM synthesis is especially valuable for metallic or clangorous noises such as bells, cymbals, or other percussion. Modulation occurs when some aspect of an audio signal (carrier) varies according to the 8odeling of another signal (modulator). FM = when a modulator drives the frequency of a carrier. Vibrato effect is good example to illustrate the principle of FM, with the difference that vibrato uses sub-audio as the modulator (below 20 Hz).Simple FM: uses only 2 sinewave oscillators. The output of the modulator is offset by a constant, represented as fc. If the amplitude of the modulator is equal to zero, then there is no modulation. In this case the output of the carrier will be a simple sinewave at frequency fc. In the amplitude of the modulator is greater than zero, then modulation occurs. The output from the carrier will be a signal whose frequency deviates proportionally to the amplitude of the modulator.

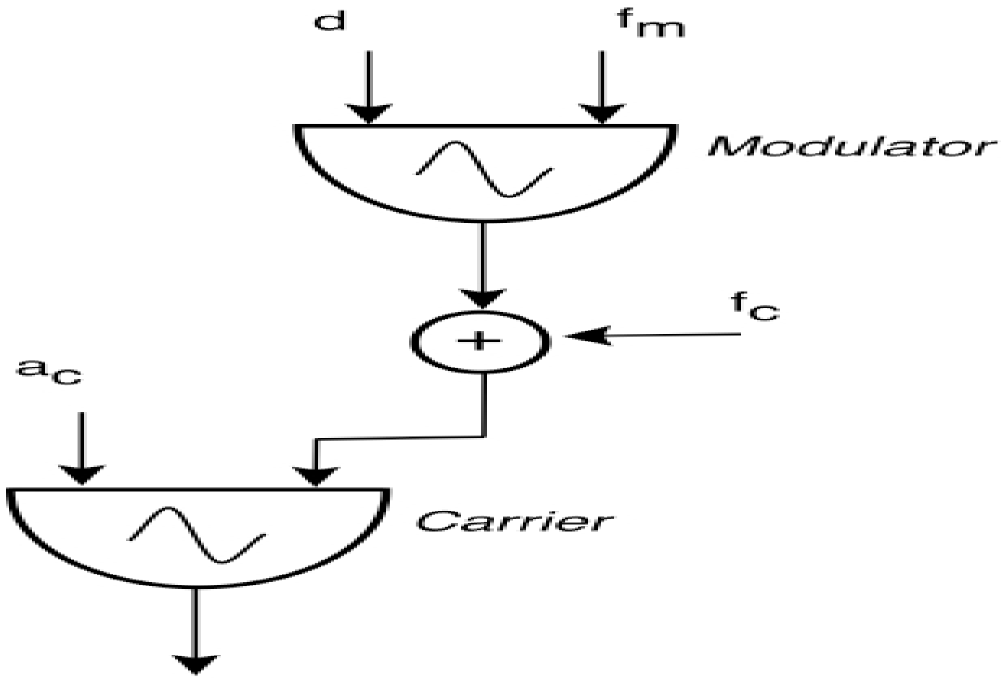


FIG 1.3

1.2.4 PHYSICAL MODELLING SYNTHESIS

Physical 9odeling synthesis is the synthesis of sound by using a set of equations and algorithms to simulate a real instrument, or some other physical source of sound. This involves taking up models of components of musical objects and creating systems that define action, filters, envelopes and other parameters over time. The definition of such instruments is virtually limitless, as one can combine any given models available with any amount of sources of modulation in terms of pitch, frequency and contour. For example, the model of a violin with characteristics of a pedal steel guitar and perhaps the action of piano hammer. When an initial set of parameters is run through the physical simulation, the simulated sound is generated. Although physical modeling was not a new concept in acoustics and synthesis, it was not until the development of the Karplus-Strong algorithm and the increase in DSP power in the late 1980s that commercial implementations became feasible. Physical modeling on computers gets better and faster with higher processing.

1.2.5 SAMPLE-BASED SYNTHESIS

Sample-based synthesis is one of the easiest synthesis systems is to record a real instrument as a digitized waveform, and then play back its recordings at different speeds to produce different tones. This is the technique used in “sampling”. Most samplers designate a part of the sample for each component of the ADSR envelope, and then repeat that section while changing the volume for that segment of the envelope. This lets the sampler have a persuasively different envelope using the same note.

1.5 DIGITAL VS ANALOG SYNTHESIZERS

Electronic music synthesis systems generally can be classified as “analog” or as “digital” systems, and there is often an associated synthesis technique that goes with them. Generally, analog synthesizers work in a voltage-controlled mode and employ “subtractive synthesis,” which is the filtering down of a harmonically rich waveform. Digital systems on the other hand generally work on what would be considered “additive synthesis,” the building up of a waveform from component frequencies. Both types of system may also use “modulation synthesis,” the enriching of a waveform by modulating it and listening to the result (i.e., not demodulating it). Further, most analog systems are capable of a limited degree of additive synthesis and many digtal systems are capable of some subtractive synthesis by making use of digital filters programmed in. There are no real rules however –most things that can be done in analog manner can be done in acorresponding digital manner. Most things that can be done in a digital manner can be done in an analog manner, although perhaps with limitations. No system is limited by theory to any one synthesis technique, and there are no rules that say that you can not employ two or more techniques at the same time.For example we can modulate a voltage-controlled oscillator, producing a complex set of component frequencies, and then filter this subtractively.

To a degree analog systems are considered to be a voltage controlled or “Moog” synthesizer. By the same degree of grouping, digital systems are considered to be “computer music” systems. Analog systems thus have a starting commercial price in the range of $1000, and perhaps as low as $200 for certain simple kits or for other home-built alternatives. Computer systems on the other hand have an implied set of instructions that begins with “first build a computer system” which is quite a bit more expensive when the various needed peripherals (terminals, CRT’s, disk memories, D/A converters) are counted. Probably at least a factor of 10 difference is reasonable to consider. Yet this is not absolute, as the moat costly analog system (over $50,000) is quite a bit more than the least expensive digital system (under $30,000). Then when you look at the details of the systems, you will probably find a lot of digital elements in the analog system (particularly with regard to control), and the digital system may well have given up a lot of its original generality so that its features resemble quite closely those of a corresponding analog system. In short, you have to look closely at the individual system for its features, its method of implementation, and its costs.

Another way of classifying systems has to do with the way it is used. There are “studio machines”. Again the distinction is not clear with regard to the actual pieces of machinery gathered together, but rather the distinction is with regard to portability (concert) or lack thereof (studio) and association with quality recording equipment (studio). Some concert setups may involve just one $1000 voltage-controlled synthesizer. Other more affluent performers may take on the road more equipment than some university studios possess. Many academic studios are computer based, although a few computer based systems are capable of going on the road. Some small concert or “performance oriented” machines will be all that is needed in some studios that are mainly aimed at recording bands with many other instruments, or will be all that the studio has invested in so far.

A digital sound is the result of counting all these values many times per second for a certain defined length/time. Each measurement value is called a sample, and this kind of process is called sampling. In order to process sounds on the computer, the analog sound must be converted into a digital format understandable by computer; that is, binary numbers.

Computers are normally configured to function based upon strings of bits (binary numbers) of fixed size, called words. For example, a computer configured for processing 4-bit words would represent the decimal numbers 0,1,2 and 3 as the binary numbers

0000 = 0

0001 = 1

0010 = 2

0011 = 3

0100 = 5

0101 = 6

etc.

The analog-to-digital conversion process is called sampling. The frequency of a sound is equal to the number of cycles which occur every second (“cycles per second”, abbreviated “cps” or “Hz”). In order to convert an analog sound signal into digital representation one needs to sample the signal many times per second. The frequency of this sampling process is called sampling frequency or sampling rate, and it is measured in Hertz (Hz).

1.6 MUSICAL INSTRUMENT DIGITAL INTERFACE(MIDI)

MIDI (Musical Instrument Digital Interface) is a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another. A single MIDI link can carry up to sixteen channels of information, each of which can be routed to a separate device.

MIDI carries event messages that specify notation, pitch and velocity, control signals for parameters such as volume, vibrato, audio panning, cues, and clock signals that set and synchronize tempo between multiple devices. These messages are sent to other devices where they control sound generation and other features. This data can also be recorded into a hardware or software device called a sequencer, which can be used to edit the data and to play it back at a later time.

MIDI technology was standardized in 1983 by a panel of music industry representatives, and is maintained by the MIDI Manufacturers Association (MMA). All official MIDI standards are jointly developed and published by the MMA in Los Angeles, California, US, and for Japan, the MIDI Committee of the Association of Musical Electronics Industry (AMEI) in Tokyo. Advantages of MIDI include compactness (an entire song can be coded in a few hundred lines, i.e. in a few kilobytes), ease of modification and manipulation and choice of instruments.

The Standard MIDI File (SMF) is a file format that provides a standardized way for sequences to be saved, transported, and opened in other systems. The compact size of these files has led to their widespread use in computers, mobile phone ringtones, webpage authoring and greeting cards. They are intended for universal use, and include such information as note values, timing and track names. Lyrics may be included as metadata, and can be displayed by karaoke machines. The SMF specification was developed and is maintained by the MMA. SMFs are created as an export format of software sequencers or hardware workstations. They organize MIDI messages into one or more parallel tracks, and timestamp the events so that they can be played back in sequence. A header contains the arrangement's setup data, which may include such things as tempo and instrumentation, and information such as the song's composer. The header also specifies which of three SMF formats the file is in. A type 0 file contains the entire performance, merged onto a single track, while type 1 files may contain any number of tracks. Type 2 files are rarely used. Microsoft Windows bundles SMFs together with Downloadable Sounds (DLS) in a Resource Interchange File Format (RIFF) wrapper, as RMID files with a .rmi extension. RIFF-RMID has been deprecated in favor of Extensible Music Files (XMF).

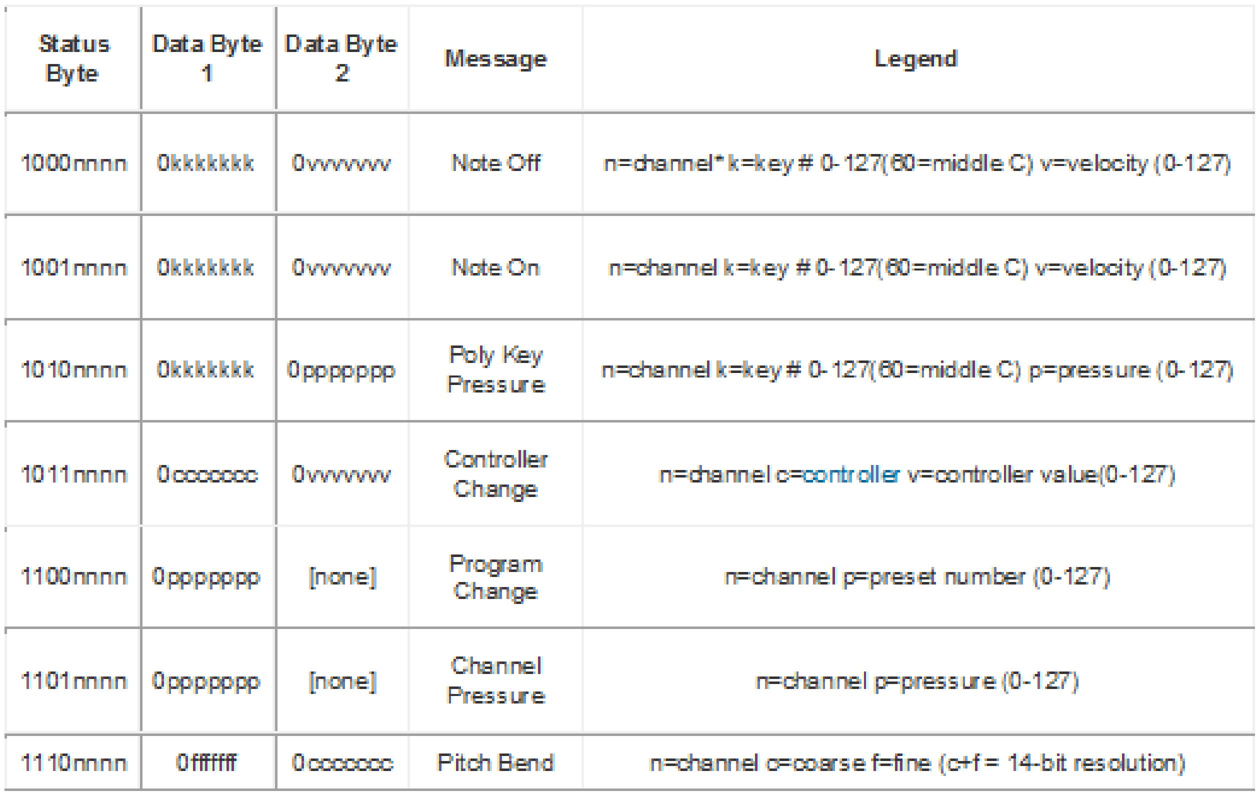
A MIDI file is not a recording of actual music. Rather, it is a spreadsheet-like set of instructions, and can use a thousand times less disk space than the equivalent recorded audio. This made MIDI file arrangements an attractive way to share music, before the advent of broadband internet access and multi-gigabyte hard drives. Licensed MIDI files on floppy disks were commonly available in stores in Europe and Japan during the 1990s. The major drawback to this is the wide variation in quality of users' audio cards, and in the actual audio contained as samples or synthesized sound in the card that the MIDI data only refers to symbolically. Even a sound card that contains high-quality sampled sounds can have inconsistent quality from one instrument to another, while different model cards have no guarantee of consistent sound of the same instrument. Early budget cards, such as the AdLib and the Sound Blaster and its compatibles, used a stripped-down version of Yamaha's frequency modulation synthesis (FM synthesis) technology played back through low-quality digital-to-analog converters. The low-fidelity reproduction of these ubiquitous cards was often assumed to somehow be a property of MIDI itself. This created a perception of MIDI as low-quality audio, while in reality MIDI itself contains no sound, and the quality of its playback depends entirely on the quality of the sound-producing device (and sound recording of samples in the device).

MIDI messages are made up of 8-bit words (commonly called bytes) that are transmitted serially at 31.25 kbaud. This rate was chosen because it is an exact division of 1 MHz, the speed at which many early microprocessors operated.[5]:286 The first bit of each word identifies whether the word is a status byte or a data byte, and is followed by seven bits of information.[2]:13–14 A start bit and a stop bit are added to each byte for framing purposes, so a MIDI byte requires ten bits for transmission.[5]:286

A MIDI link can carry sixteen independent channels of information. The channels are numbered 1-16, but their actual corresponding binary encoding is 0-15. A device can be configured to only listen to specific channels and to ignore the messages sent on other channels ("Omni Off" mode), or it can listen to all channels, effectively ignoring the channel address ("Omni On"). An individual device may be monophonic (the start of a new "note-on" MIDI command implies the termination of the previous note), or polyphonic (multiple notes may be sounding at once, until the polyphony limit of the instrument is reached, or the notes reach the end of their decay envelope, or explicit "note-off" MIDI commands are received). Receiving devices can typically be set to all four combinations of "omni off/on" versus "mono/poly" modes.

A MIDI message is an instruction that controls some aspect of the receiving device. A MIDI message consists of a status byte, which indicates the type of the message, followed by up to two data bytes that contain the parameters. MIDI messages can be "channel messages", which are sent on only one of the 16 channels and can be heard only by devices receiving on that channel, or "system messages", which are heard by all devices. Any data not relevant to a receiving device is ignored. There are five types of message: Channel Voice, Channel Mode, System Common, System Real-Time, and System Exclusive.

Channel Voice messages transmit real-time performance data over a single channel. Examples include "note-on" messages which contain a MIDI note number that specifies the note's pitch, a velocity value that indicates how forcefully the note was played, and the channel number; "note-off" messages that end a note; program change messages that change a device's patch; and control changes that allow adjustment of an instrument's parameters. Channel Mode messages include the Omni/mono/poly mode on and off messages, as well as messages to reset all controllers to their default state or to send "note-off" messages for all notes. System messages do not include channel numbers, and are received by every device in the MIDI chain. MIDI time code is an example of a System Common message. System Real-Time messages provide for synchronization, and include MIDI clock and Active Sensing.



Status Byte	Data Byte 1	Data Byte 2	Message	Legend
1000nnnn	0kkkkkkk	0vvvvvvv	Note Off	n=channel* k=key # 0-127(60=middle C) v=velocity (0-127)
1001nnnn	0kkkkkkk	0vvvvvvv	Note On	n=channel k=key # 0-127(60=middle C) v=velocity (0-127)
1010nnnn	0kkkkkkk	0ppppppp	Poly Key Pressure	n=channel k=key # 0-127(60=middle C) p=pressure (0-127)
1011nnnn	0ccccccc	0vvvvvvv	Controller Change	n=channel c=controller v=controller value(0-127)
1100nnnn	0ppppppp	[none]	Program Change	n=channel p=preset number (0-127)
1101nnnn	0ppppppp	[none]	Channel Pressure	n=channel p=pressure (0-127)
1110nnnn	0fffffff	0ccccccc	Pitch Bend	n=channel c=coarse f=fine (c+f = 14-bit resolution)

Table 1.1

CHAPTER-2

CHUCK

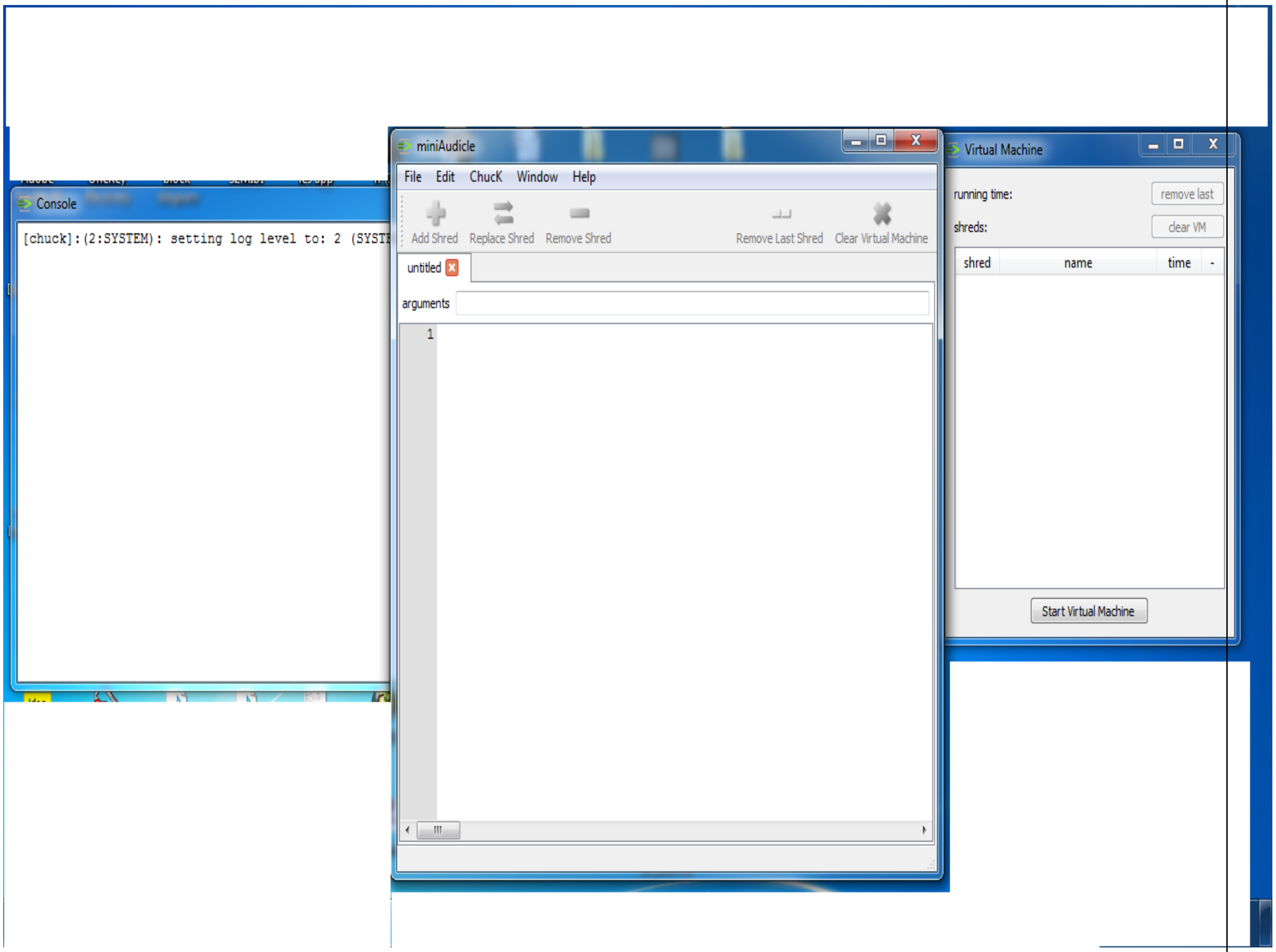
2.1 WHAT IS CHUCK

ChucK is a programming language designed specifically for real-time sound synthesis and music creation. “Real-time” means that ChucK synthesizes the sound as you are hearing it (rather than just playing back a sound file), often in response to signals and gestures from the “outside world.” Sound controlling gestures might include you typing on the keyboard, moving the computer mouse, manipulating a joystick or other game controller, or playing the keys on a musical keyboard connected to your computer.. Not just for sound and music, ChucK is also good for controlling and/or interacting with almost any type of real-time computer media and art, such as graphics, robots, or whatever can communicate with your computer.

ChucK was designed specifically to allow and encourage “on-the-fly” programming, which means you can add, remove, modify and edit, and layer segments of code at any and all times, hearing the results instantly, without interrupting other sounds being synthesized and heard. This is one of the primary differences of ChucK from all other languages, which makes it extremely fun to learn and use, because you can try things out and immediately hear the results. Most other languages require you to compile, run, and debug code in a way that doesn’t let you hear what you’re doing immediately. In fact, most computer languages (such as C, C++, or Java) were not designed specifically from the ground-up for sound, music, and other real-time tasks. But ChucK makes immediate, real-time sound a priority.

There are other computer languages such as Java or C++, or even other music/sound languages such as Csound, SuperCollider, Jsyn, MaxMSP, or PD (Pure Data), but ChucK is really different. It’s more expressive and powerful at manipulating time and sound than the graphical interfaces of MaxMSP and PD, giving much more “under the hood” access than these other languages and systems. Compared to other text-based music/sound languages such as SuperCollider or Csound, ChucK is generally more succinct, requiring much less code (lines of typed text) than these other languages in order to accomplish any particular task. ChucK is different from other languages for sure, but it shares many things that will be similar and recognizable to programmers of nearly any language. Another great feature of ChucK is that it is "open-source" (not secret or protected by licenses, passwords, keys, etc.), and it is freely available on all major computer platforms, including Mac OSX, Windows, and Linux. Open-source means that the community of ChucK users can have direct input into the process of making ChucK better in the future. It also means that ChucK doesn't cost anything to get and use.

There are three main windows in the miniAudicle. The main window that you see (top of Figure 1.6) is initially called "Untitled," and is the window in which you write your programs. The lower right window in Figure is the Virtual Machine Monitor, which shows the status of the Virtual Machine (VM) server while your ChucK code runs. Once started, the VM is always there, waiting for things to do.



```
[chuck]:(2:SYSTEM): setting log level to: 2 (SYSTE
```

Fig 2.1

2.2 ADVANTAGES OF CHUCK

• It’s all about time. Time is at the core of how ChucK works, and how one works with ChucK to make sound. As a programmer, you specify how to move through time and assert control at specific points in time – and sound essentially just happens (and, conveniently , for precisely the amount of time you have moved through). Why such emphasis on time? Sound is a time-based phenomenon; without the passage of time, there would be no sound. By controlling how and when we do things through time, it’s a different and powerful way to work with sound at every level – every molecule of it.

• It’s text, plain and simple. While programming with text may initially seem more abstract/complex than say, with graphical representations – it is arguably much easier once you start adding a lot of expressive nuance and logic into your code (which you’ll invariably need to do). There is little hidden or inferred – the important parts are in plain sight (for example, how time flows in a program). At the same time, much of the mundane aspects are taken care of under the hood (scheduling, real-time sound input/output, book keeping for all the sound generators, etc.). Readability is a central design goal of the language, and that makes it a good learning tool as well.

• It’s fun and immediate. ChucK was designed to be a fun language and environment to work in, to experiment in. You can synthesize sounds, make fantastical automations, map physical gestures (e.g., with controllers) to sound, network different computers together, and even analysis to computational make sense of sound.

CHAPTER-3

ARDUINO UNO

The Arduino Uno is a microcontroller board based on the Atmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board’s power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

VIN. The input voltage to the Arduino board when it’s using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

5V.This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 – 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don’t advise it.

3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

GND. Ground pins.

IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

The Atmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the Atmega8U2 USB-to-TTL Serial chip. External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details. PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function. SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library. LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

AREF. Reference voltage for the analog inputs. Used with analogReference().

Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The Atmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An Atmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.

The Atmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). The Atmega328 on the Arduino Uno comes preburned with a bootloader that allows to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header.

The Atmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The Atmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.

On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel’s FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader)

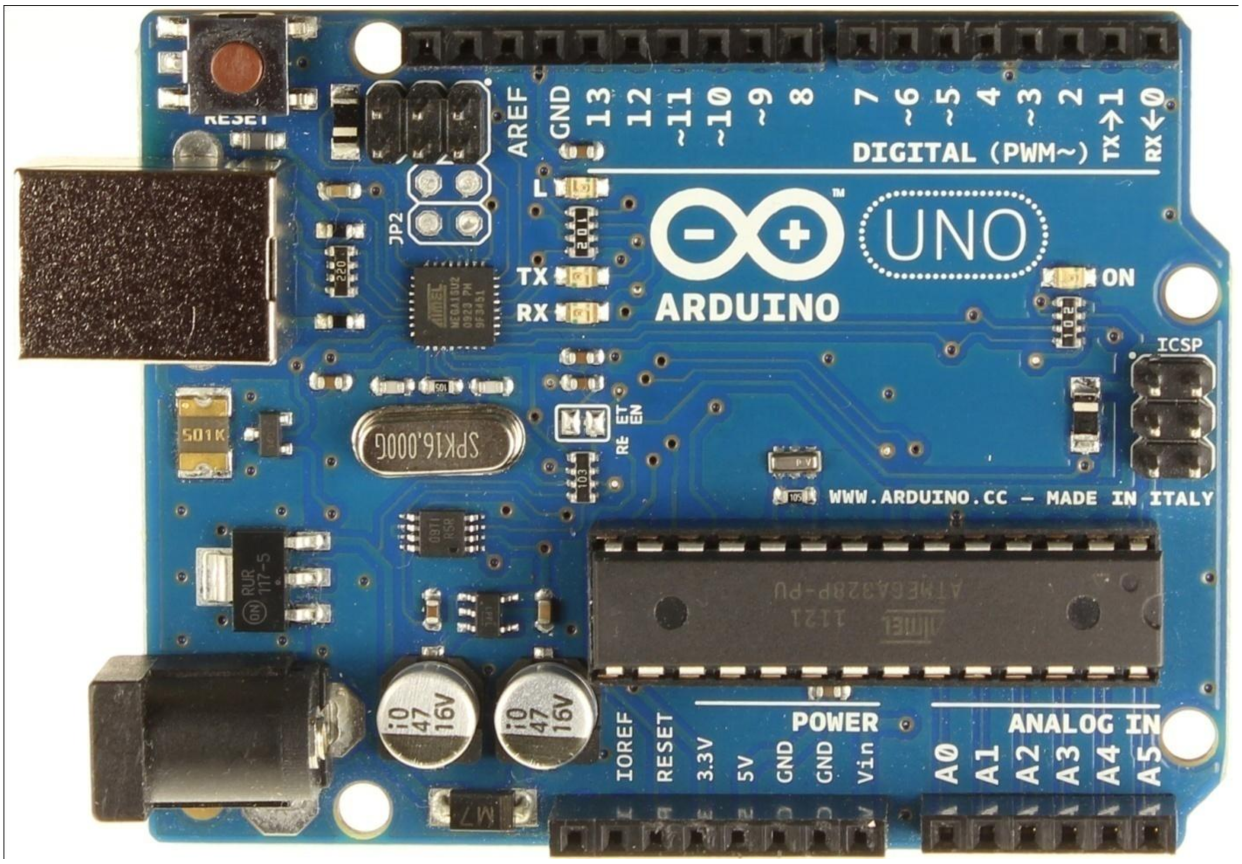


Fig 3.1

Arduino uno front

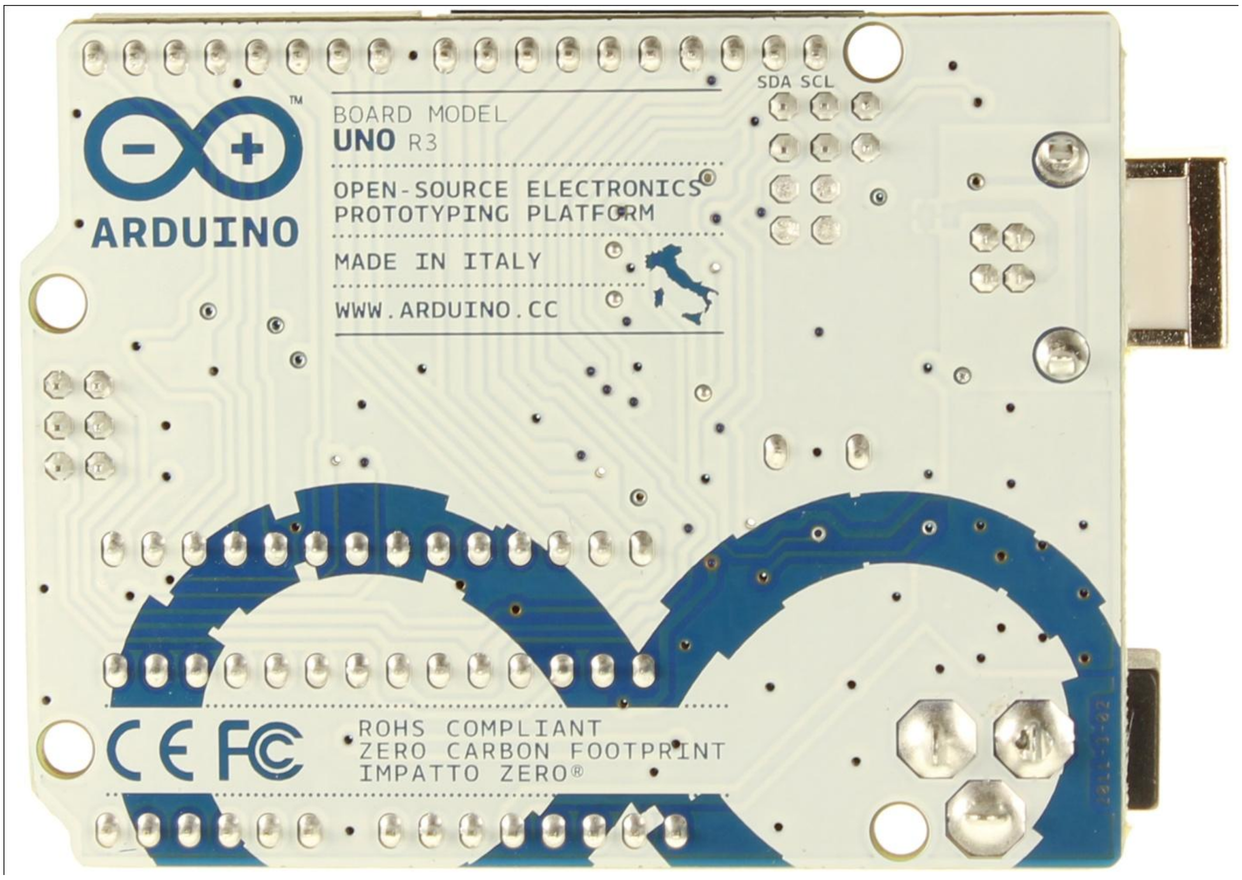


Fig 3.2

Arduino uno back

CHAPTER-4

CAPACITIVE SENSOR

4.1-INTRODUCTION

Capacitive sensors can directly sense a variety of things—motion, chemical composition, electric field—and, indirectly, sense many other variables which can be converted into motion or dielectric constant, such as pressure, acceleration, fluid level, and fluid composition. They are built with conductive sensing electrodes in a dielectric, with excitation voltages on the order of five volts and detection circuits which turn a capacitance variation into a voltage, frequency, or pulse width variation. The range of application of capacitive sensors is extraordinary.

• Motion detectors can detect 10-14 m displacements with good stability, high speed, and wide extremes of environment, and capacitive sensors with large electrodes can detect an automobile and measure its speed

• Capacitive technology is displacing piezoresistance in silicon implementations of accelerometers and pressure sensors, and innovative applications like fingerprint detectors and infrared detectors are appearing on silicon with sensor dimensions in the microns and electrode capacitance of 10 fF, with resolution to 5 aF (10-18 F).

• Capacitive sensors in oil refineries measure the percent of water in oil, and sensors in grain storage facilities measure the moisture content of wheat

• In the home, cost-effective capacitive sensors operate soft-touch dimmer switches and help the home craftsman with wall stud sensors and digital construction levels

• Laptop computers use capacitive sensors for two-dimensional cursor control, and transparent capacitive sensors on computer monitors are found in retail kiosks.

4.2-APPLICATIONS

Capacitive sensors have a wide variety of uses. Some are

• Flow–Many types of flow meters convert flow to pressure or displacement, using an orifice for volume flow or Coriolis effect force for mass flow. Capacitive sensors can then measure the displacement.

• Pressure–A diaphragm with stable deflection properties can measure pressure with a spacing-sensitive detector.

• Liquid level –Capacitive liquid level detectors sense the liquid level in a reservoir by measuring changes in capacitance between conducting plates which are immersed in the liquid, or applied to the outside of a non-conducting tank.

• Spacing–If a metal object is near a capacitor electrode, the mutual capacitance is a very sensitive measure of spacing.

• Scanned multiplate sensor–The single-plate spacing measurement can be extended to contour measurement by using many plates, each separately addressed. Both conductive and dielectric surfaces can be measured.

• Thickness measurement–Two plates in contact with an insulator will measure the insulator thickness if its dielectric constant is known, or the dielectric constant if the thickness is known.

• Ice detector–Airplane wing icing can be detected using insulated metal strips in wing leading edges.

• Shaft angle or linear position–Capacitive sensors can measure angle or position

with a multiplate scheme giving high accuracy and digital output, or with an analog output with less absolute accuracy but faster response and simpler circuitry.

• Lamp dimmer switch–The common metal-plate soft-touch lamp dimmer uses 60 Hz excitation and senses the capacitance to a human body.

• Keyswitch–Capacitive keyswitches use the shielding effect of a nearby finger or a moving conductive plunger to interrupt the coupling between two small plates.

• X-Y tablet–Capacitive graphic input tablets of different sizes can replace the computer mouse as an x-y coordinate input device. Finger-touch-sensitive, z-axis-sensitive and stylus-activated devices are available.

4.3-THEORY

Capacitive sensors directly sense electrode motion, conductive or dielectric object motion, or the dielectric properties of a local material. All other variables must be first converted into one of these.

As the technology is capable of excellent, stable, low-noise sensing, this indirect method works well. As an example, a cube of brass one cm square expands sufficiently with temperature to make a good thermometer, with better than one degree C accuracy. An infrared imager has been built which converts IR energy into the displacement of thousands of small (14 µm) bimetallic strips which are read capacitively. The equivalent circuit of a capacitor can be approximated by this circuit, with small series resistance and inductance neglected for our high-impedance uses. Good capacitor dielectrics have a very large shunt resistance; polypropylene capacitors have an RC product of over 300 hours. Other materials have a much smaller shunt resistance, sometimes reaching 5-10% of the impedance of the capacitor. Although the dielectric constant K of most materials is stable, the shunt resistance or its equivalent, loss tangent, may show considerable variation with material properties or with frequency. As an example, dry leather has a loss tangent of 0.045, but with a relative humidity of 15% the loss tangent increases to

1.4–possibly a good hygrometer. Aviation gas at 100 octane exhibits a loss tangent at 1 kHz of 0.0001, but at 91 octane loss tangent increases to 0.0004. Water has a high K (80) and a loss tangent which peaks at low frequencies and again at 1010 Hz. With this high dielectric activity, the loss tangent or the dielectric constant of water can be used to detect the moisture content of materials. Another characteristic of capacitor dielectrics which may have some use in detecting material properties is dielectric absorption. It is measured by charging a capacitor, discharging for 10 s, and measuring the charge which reappears after 15 min. A relatively low-quality dielectric like 24etalized paper has a dielectric absorption of 10%.

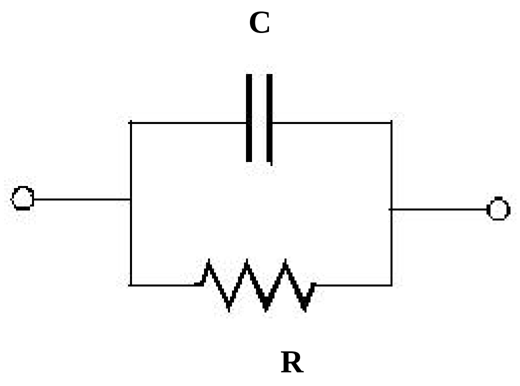


Fig 4.1

Signal conditioning

Signal conditioning circuits convert capacitance variations into a voltage, frequency, or pulse width modulation. Very simple circuits can be used, but simple circuits may be affected by leak-age or stray capacitance, and may not be suitable for applications with very small capacitance sense electrodes.

Excitation frequency

The excitation frequency should be reasonably high so that electrode impedance is as low as possible. Typical electrode impedance is 1-100M ohms. Ideally, the excitation frequency will be high enough to reject coupling to power waveforms and also high enough so that the overall sen-sor frequency response is adequate; about 50 kHz is usually acceptably high. The frequency should also be low enough for easy circuit design, CMOS switches work well at 100 kHz and below.

Excitation waveshape is usually square or trapezoidal, but a triangle waveform can be used to allow a simpler amplifier with resistive feedback and a sine wave offers better accuracy at high frequency. Square wave excitation produces an output bandwidth which can be higher than the excitation frequency by 10x or more, other waveshapes usually result in an output bandwidth 2x or 3x lower than the excitation frequency.

Sensors excited with a continuous wave signal usually use synchronous demodulators. This demodulator type offers high precision and good rejection of out-of-band interference.

Pulse operation

A single pulse can be used to sample a variable capacitor, like a microcomputer read pulse, or a train of pulses can be used. This method can result in simpler electronics but will have higher noise.

Oscillator

An R-C relaxation oscillator such as the venerable 555 or its CMOS update, the 7555, converts capacitance change into a change of frequency or pulse width

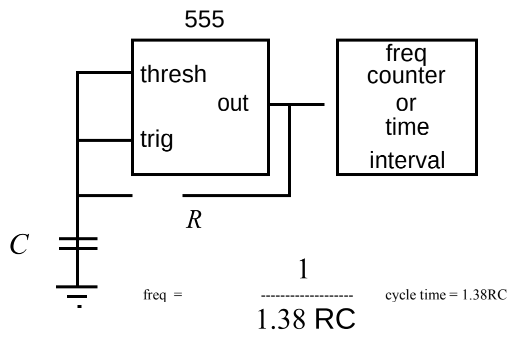


Fig 4.2 RC OSCILLATOR

The RC oscillator used with a spacing-variation capacitor will produce a frequency output which is linear with spacing, while an area-variation capacitor is linearized by measuring pulse width.

The circuit as shown has no way of accommodating stray capacitance. If, for example, the capacitor is connected by a coaxial cable, the cable capacitance adds to the measured capacitance and spoils stability and sensitivity. Often a computer can be used to calibrate these errors, but the synchronous demodulator circuits shown below are a more accurate choice.

Another drawback with the RC oscillator is that capacitance is measured relative to the fixed resistor, and the resistor stability and temperature coefficient may not track well.

Synchronous demodulators allow the use of capacitance bridges so the reference capacitance tempco will track accurately, yielding higher accuracy sensors.

CHAPTER-5

METHODOLOGY

5.1-ARDUINO CODE

```
#include <CapacitiveSensor.h>

#include <MIDI.h>


CapacitiveSensor   cs_4_2 = CapacitiveSensor(4,2);  // 10M resistor between pins 4 & 2, pin 2
is sensor pin, add a wire and or foil if desired




void setup()

{

  Serial.begin(9600);

}


void loop()

{

   long start = millis();

   long total1 =  cs_4_2.capacitiveSensor(30);

     Serial.print(total1);                   // print sensor output 1

     delay(100);// arbitrary delay to limit data to serial port

    static int lastInput1 = 0;

   int newInput1 = total1;

   if((lastInput1 < 500) && (newInput1 > 500))

   {

       MIDI.sendNoteOn(44,127,1);  // Send a Note (pitch 42, velo 127 on channel 1)
```

```
        delay(100);
    }
    if((lastInput1 > 500) && (newInput1 < 500))
    {
      MIDI.sendNoteOff(44,0,1);  // Send a Note (pitch 42, velo 127 on channel 1)


    delay(50);


    }
    else
    MIDI.sendNoteOff(0,0,1);
    lastInput1 = newInput1;


}
```

5.2- CHUCK CODE

```
MidiIn min;
// MIDI Port (ChucK Menu: Window > Device Browser > MIDI > Input)
0 => int port;
// open the port, fail gracefully
if( !min.open(port) ) //
{
<<< "Error: MIDI port did not open on port: ", port >>>;
me.exit();
}
// holder for received messages
MidiMsg msg; //
// make an instrument to play
Rhodey piano => dac; //
// loop
while( true ) //
{
min => now; // advance when receive MIDI msg //
while( min.recv(msg) )
{
<<< msg.data1, msg.data2, msg.data3 >>>;
if (msg.data1 == 144) {
Std.mtof(msg.data2) => piano.freq;
msg.data3/127.0 => piano.gain;
1 => piano.noteOn;
}
else {
1 => piano.noteOff;
}
}
}
```

5.3- BLOCK DIAGRAM

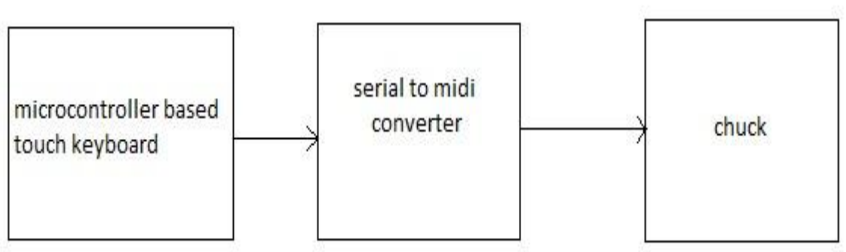


Fig-5.1

5.4- CIRCUIT DIAGRAM

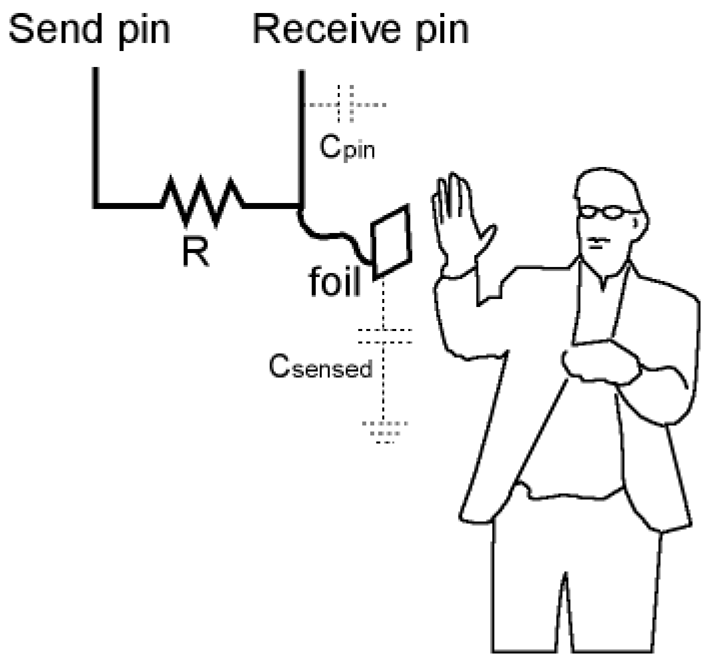


Fig 5.2

5.5- CAPACITIVE SENSOR WORKING

The capacitiveSensor method toggles a microcontroller send pin to a new state and then waits for the receive pin to change to the same state as the send pin. A variable is incremented inside a while loop to time the receive pin's state change. The method then reports the variable's value, which is in arbitrary units.

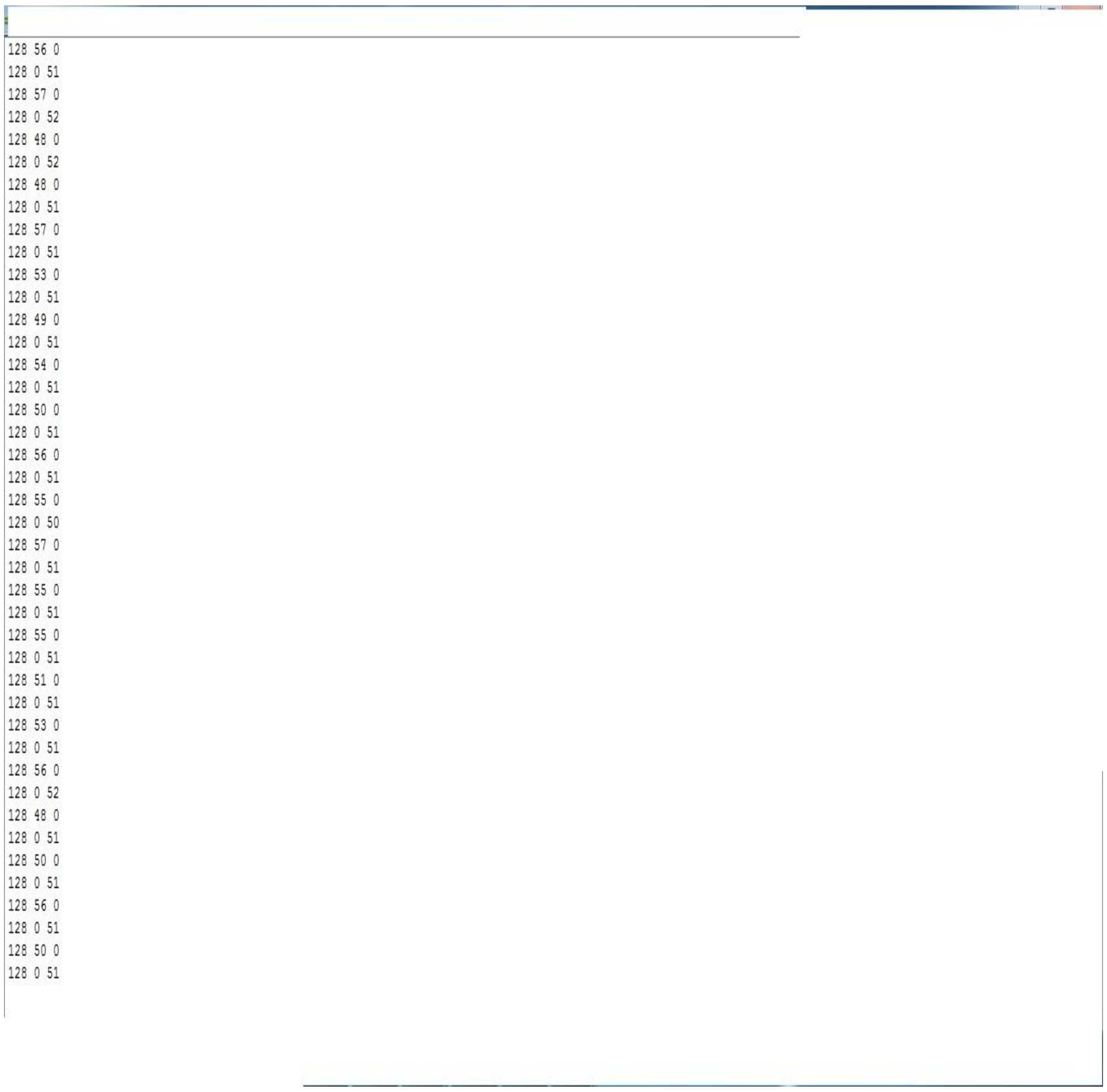
The physical setup includes a medium to high value (100 kilohm – 50 megohm) resistor between the send pin and the receive (sensor) pin. The receive pin is the sensor terminal. A wire connected to this pin with a piece of foil at the end makes a good sensor. For many applications, a more useful range of values is obtained if the sensor is covered with paper, plastic, or another insulating material, so that users do not actually touch the metal foil. Research has shown that a small capacitor (100 pF) or so from sensor pin to ground improves stability and repeatability.

When the send pin changes state, it will eventually change the state of the receive pin. The delay between the send pin changing and the receive pin changing is determined by an RC time constant, defined by R * C, where R is the value of the resistor and C is the capacitance at the receive pin, plus any other capacitance (e.g. human body interaction) present at the sensor (receive) pin. Adding small capacitor (20 – 400 pF) in parallel with the body capacitance, is highly desirable too, as it stabilizes the sensed readings.

RESULTS

6.1 SIMULATION

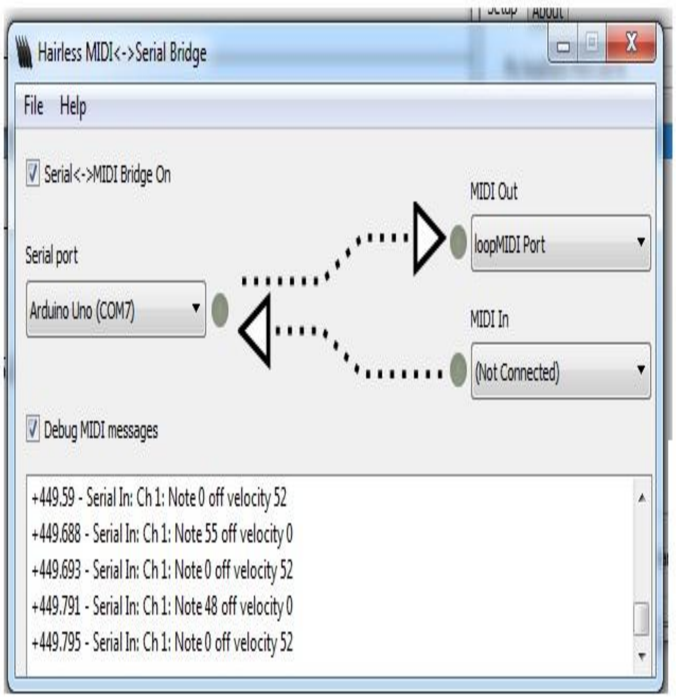
6.1.1 CHUCK CONSOLE:



```
128 56 0
128 0 51
128 57 0
128 0 52
128 48 0
128 0 52
128 48 0
128 0 51
128 57 0
128 0 51
128 53 0
128 0 51
128 49 0
128 0 51
128 54 0
128 0 51
128 50 0
128 0 51
128 56 0
128 0 51
128 55 0
128 0 50
128 57 0
128 0 51
128 55 0
128 0 51
128 55 0
128 0 51
128 51 0
128 0 51
128 53 0
128 0 51
128 56 0
128 0 52
128 48 0
128 0 51
128 50 0
128 0 51
128 56 0
128 0 51
128 50 0
128 0 51
```

Fig 6.1

6.1.2 HAIRLESS MIDI:



```
+449.59 - Serial In: Ch 1: Note 0 off velocity 52
+449.688 - Serial In: Ch 1: Note 55 off velocity 0
+449.693 - Serial In: Ch 1: Note 0 off velocity 52
+449.791 - Serial In: Ch 1: Note 48 off velocity 0
+449.795 - Serial In: Ch 1: Note 0 off velocity 52
```

6.2

CONCLUSION

Arduino is sending serial data to hairless midi which is then sending the corresponding midi messages to chuck.

The sound is being played by touching the wire or any conductor which is in contact with it. This sound is synthesized in chuck and any desirable sound can be produced by coding in chuck.

REFRENCES

1. Programming for musicians and digital artists by Ajay Kapur.

2. COURSEERA.

3.Aurduino tutorial by Jeremy blum.

4. www.instructables .com

5. P. Cook. “Principles for Designing Computer Music Controllers,” Proceedings of the Conference on New Interfaces for Musical Expression (NIME), Seattle, 2001.

6. Wearable sensors for real-time musical signal processing by A.Kapur