自動化長笛與準則化音樂控制之研究

研究生：許擎亞　　指導教授：黃志方

國立交通大學音樂研究所

摘要

木管樂器的自動化演奏研究，包含以一支聲響長笛和相關的自動控制裝置與機構，以及電腦自動化產生之音樂創作系統之整合設計為基礎。自動控制裝置又分為兩個主要的部份 - 吹嘴裝置和按鍵裝置。 吹嘴裝置以運用步進馬達為主；而按鍵裝置以運用伺服機為主。 本研究亦包含兩種準則作曲方式，細胞自動機和機率演算法。 自動演奏長笛可以演奏即時運算的演算法音樂或是一般樂譜的曲子。 演奏設定依據不同的控制模式可分為三種：分斷 (detache)， 斷奏 (staccato) ，和圓滑奏 (legato)。最後該整合式自動演奏長笛系統雛型已成功地完成開發，其相關性能與電腦產生之音樂數據亦於本論文中討論。

關鍵字：準則作曲、細胞自動機、整合式自動演奏長笛系統雛型

A Research of Automated Flute with Algorithmic Music Control

Student: Ching-Ya Hsu Advisor: Dr. Chih-Fang Huang

Institute of Music
National Chiao Tung University

ABSTRACT

The research of automated wind instrument is based on the acoustic flute with automatic control mechanism, and the computer generative music composition. The control mechanism is divided into two major components: mouthpiece and keystroke mechanisms. Stepper motor is the choice for the mouthpiece mechanism and RC Servos for the keystroke mechanism.

Two algorithmic composition modules, "Probability module" and "Cellular Automata module" are implemented to the automatic flute system. Besides, algorithmic composition, music score can also be entered in the "Score-Play" module. The automatic flute could perform music from either one of the algorithmic composition module or from the music score. Three performance settings are also available. They are detache, staccato, and legato, according to the various control modes. Eventually the integrated automated flute system prototype has been successfully developed. The performance and generated music data will be discussed in the thesis.

Keywords: Algorithmic Composition, Cellular Automata, Integrated Automated Flute System Prototype

致謝

Halelujah! 感謝交大音樂所大大小小的一切。剛回國時知道進音樂所， 拿到了想了很久的台灣學號。其實我一直懷念小時候在台灣上課的情形，我想我渴望的應該不是平常一般的研究所生活，而是能跟台灣當地的藝術音樂人有相連，在生活、文化、以及藝術上。 簡單說這三年我收穫最大的是朋友，老師同學們的感情，像是一個大家庭 。 我們一起經歷很多音樂會、展覽、和研討會。 WOCMAT、 德國 Schatten、竹韻、紅樓 ， to name a few。 每當忙完大活動，老師，學姊，都會常請吃飯（很不好意思但心裡很高興）和找時間一起出遊。這些都是我想像不到的研究所生活。謝謝你們！

特別感謝黃志方老師指導，給我很多學業以及未來工作上的建議。感謝老師忍受我這個對於理工慢半拍又興趣缺缺的笨蛋。特別感謝董昭民老師的照顧，創作上的想法、態度、以及生活、感情上的一切。還有謝謝在所裡的每位老師，Phil Winsor 老師、金立群老師、辛幸純老師、李子聲老師、曾毓忠老師、以及瑞紋姊。

謝謝所上丹綾、如韻、佳蔚、以榮、舜評、亦湄、俊德、書蘋、芳瑜、幼齡、琦惠、郁馨、怡瑾、欣歆、立偉、世揚，科技組伙伴們，淑瑾、珍妮、楷婷、士涵、奕佐、 Jose、幸輯、聖賢，還有好朋友們婉玉、倩雯、信男、聖軒、的陪辦。And cheers to 亞書丹德！

ACKNOWLEDGEMENTS

I would like to give special thanks to Prof. Phil Winsor and Prof. Huang Chih-Fang for inspiration and support throughout the studies. Thanks to all my classmates at the Institute of Music for all the memorable moments.

CONTENTS

PART TWO
CONTENTS

ILLUSTRATIONS

Figure Page

PART TWO
ILLUSTRATIONS

Figure Page

PART THREE
ILLUSTRATIONS

Figure Page

TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

I am always in search of different means of music expressions. One approach lies in instrument making. Synthesizers, making instrumental patches from software, algorithmic compositions, and interactive music machines are all different forms of instrument making. Basically any objects that could assist in the production of sound in a creative way could be called instruments. All of the steps I am taken are nothing new. The histories is made as I walk in the foot steps of old school composers, such as Harry Partch, Varese, and many other individuals of today. The main motivation lies in the building of an actual physical instrument, which could realize the full potential of the flute sonorities and further expand on the already discovered extended-instrumental techniques of the flute. The research of automatic flute will bring about a new interactive platform between composers, performers, and machines.

1.2 Background

Most of the automated music instrument system is based on the electronic instruments, especially the synthesizers. Some cases of the acoustic music instruments have already developed into the commercial products, and the most famous one is Yamaha DISKLAVIER, which uses rapid hammer control mechanism with the digital MIDI signal command issued via either a personal computer or an electronic device.

Some of the acoustic music instruments with automatic control have been brought into the academic fields, including the automated guitar, automated drum, etc.[1] These researches are still under verification, and need more implementations with music technology integration to improve their feasibility. The automatic woodwind instruments, such as flute or recorder are seldom discussed in the music technology fields. It can be designed with some actuators and electrical controllers, and then the algorithmic composition can be used to integrate with the automated flute or recorder, via the MIDI programming.

[1] Yu-Wei Huang, *Improvements of Automatic Guitar Mechanism Design and a Practical Music Input System,* Master Thesis of the Electrical Engineering Department, Southern Taiwan University, 1996.

CHAPTER 2

METHODOLOGY

2.1 System Architecture

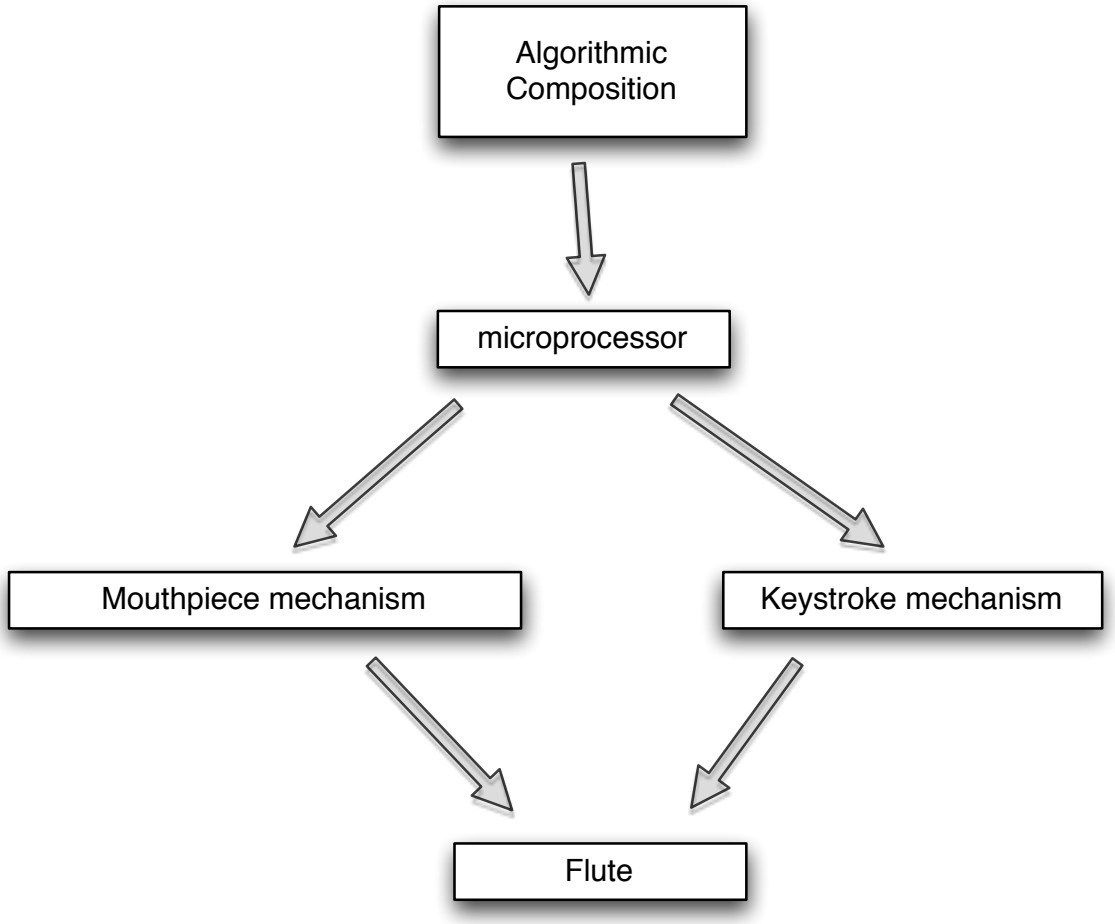


Fig. 2.1.1. Architecture Summary

The project is to build an automatic musical instrument, where when musical data is entered, whether as a music score or live generated algorithms, the instrument will be set to motion and creating sound. Figure 2.1.1 shows the major

processes from top to bottom. It is a non-feedback loop system. The algorithmic composition module, realized in Max/Msp, can run in three different modes: automatic, manual, and music score. When data enters the microprocessor, it is translated and redirected to two different output terminals: keystroke mechanism and mouthpiece mechanism. Both mechanical units will convert the input data into PWM signals and therefore creating motion, one moving the flute keys, the other controlling the airflow for the mouthpiece.

2.2 Overtone Series and Flute Acoustics

Overtone Series refers to a set of frequency components that are above a musical tone (fundamental)(Fig. 2.2.1). The fundamental is the strongest in dynamic, while other upper partials are relatively weak. Different strength combinations of the upper partials constitute the color of the sound object. The overtone series exist in nature and play a major factor in instrumental timbre.

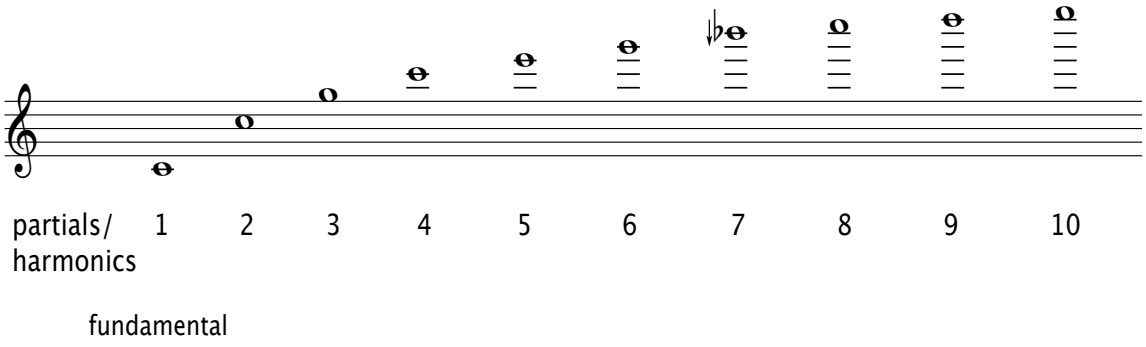


Fig. 2.2.1. Overtone Series

Imagine a string or cylindrical tube of length L, when set to motion (excites and vibrate the surrounding air), will generate a fundamental tone F. To get the upper harmonics of F, we reduce the string or tube length to L/(# of partials).

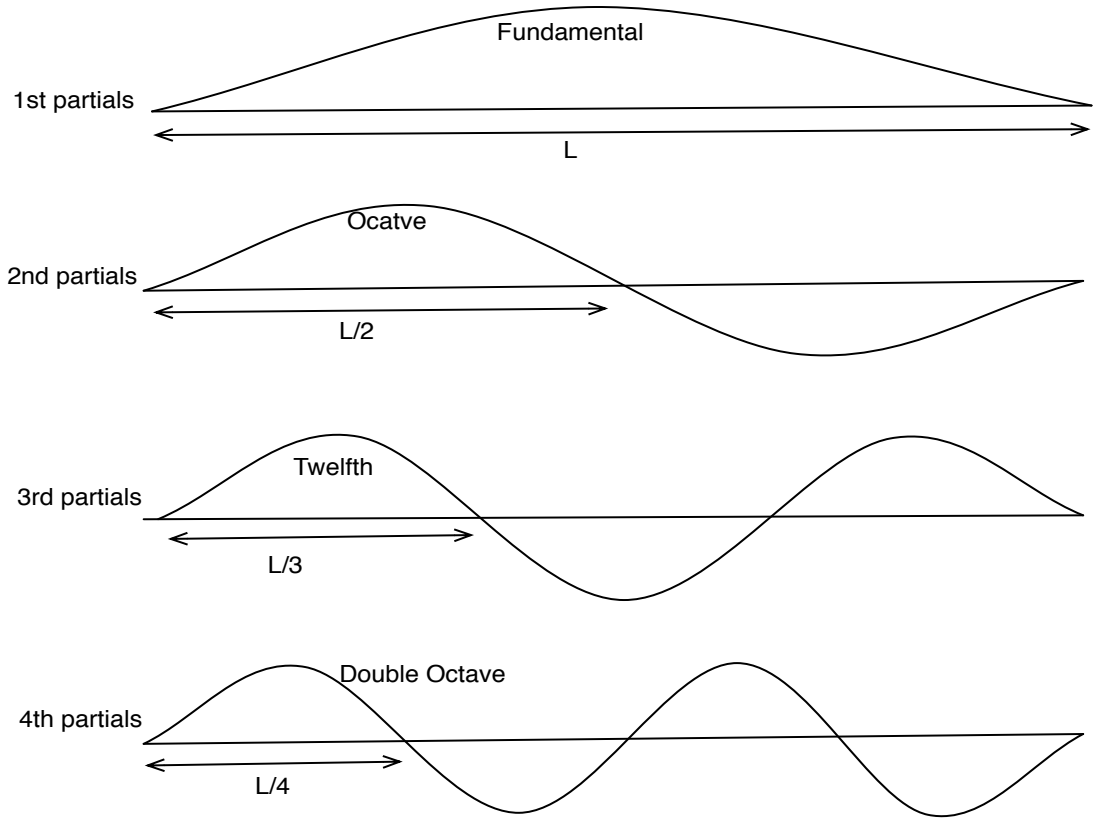


Fig. 2.2.2. Overtone Waveform

According to figure 2.2.2, if the string length is reduced in half, we get the second harmonic. If the string length is one third of the original, we get the third harmonic and so on.

Figure 2.2.1 shows the overtone series on the musical staff. If the fundamental note is C4, the 2^{nd}, 4^{th} , and 8^{th} partials will be C4's next higher octave respectively. From this, we denote that if the partial numbers are related by

multiple of 2, the pitch-note value will be octave equivalent. The 3rd partial, G5, is a perfect 5th above the 2nd partial. The 6th partial, which is 2 times the 3rd, will be G6. The 5th partial, E6, is a major 3rd above the 2nd octave and E7 would be found on the 10th. The 7th partial, Bb5 (very flat), is a major 2nd below the 3rd octave. Last, the 9th partial, D6, is a major 2nd above the 3rd octave.

Due to the nature of overtone series and acoustics of the flute, there are several ways of producing pitches on the flute. The simplest way would be to assume that one fingering combination produce one pitch-note. In reality, with an experienced player, it is possible to blow in a way that will vibrate the air column fractionally and bring out the harmonics above the fundamental. One of the easiest techniques would be overblown. In flute, when overblown the 2nd harmonic is produced. The resulting pitch would be an octave higher for the same fingering. It is also possible to get the first seven or eight harmonics by successively blowing with more force while all the tone holes are closed.[2] One example of different ways of producing the same pitch, for example G5, would be first, fingering G5 directly and the other finger C4 and play its third harmonic (refer Fig. 2.2.1).

For simplicity of the research, the current automatic flute design will take more precaution in neglecting the nature of flute acoustics and overtone series and will assume that one fingering combination produce one pitch-note. The model will not concentrate on the higher partials or the different vibrational mode[3] in detail.

---

[2] School of Physics, University of New South Wales, *Flute Acoustics: an introduction,* (Sydney: accessed 5 Octover 2008) available from http://www.phys.unsw.edu.au/jw/fluteacoustics.html

[3] Arthur H. Benade, *Horns, Strings, and Harmony,* (New York: Dover Publications, Inc., 1992), 48.

2.3 Automatic Flute

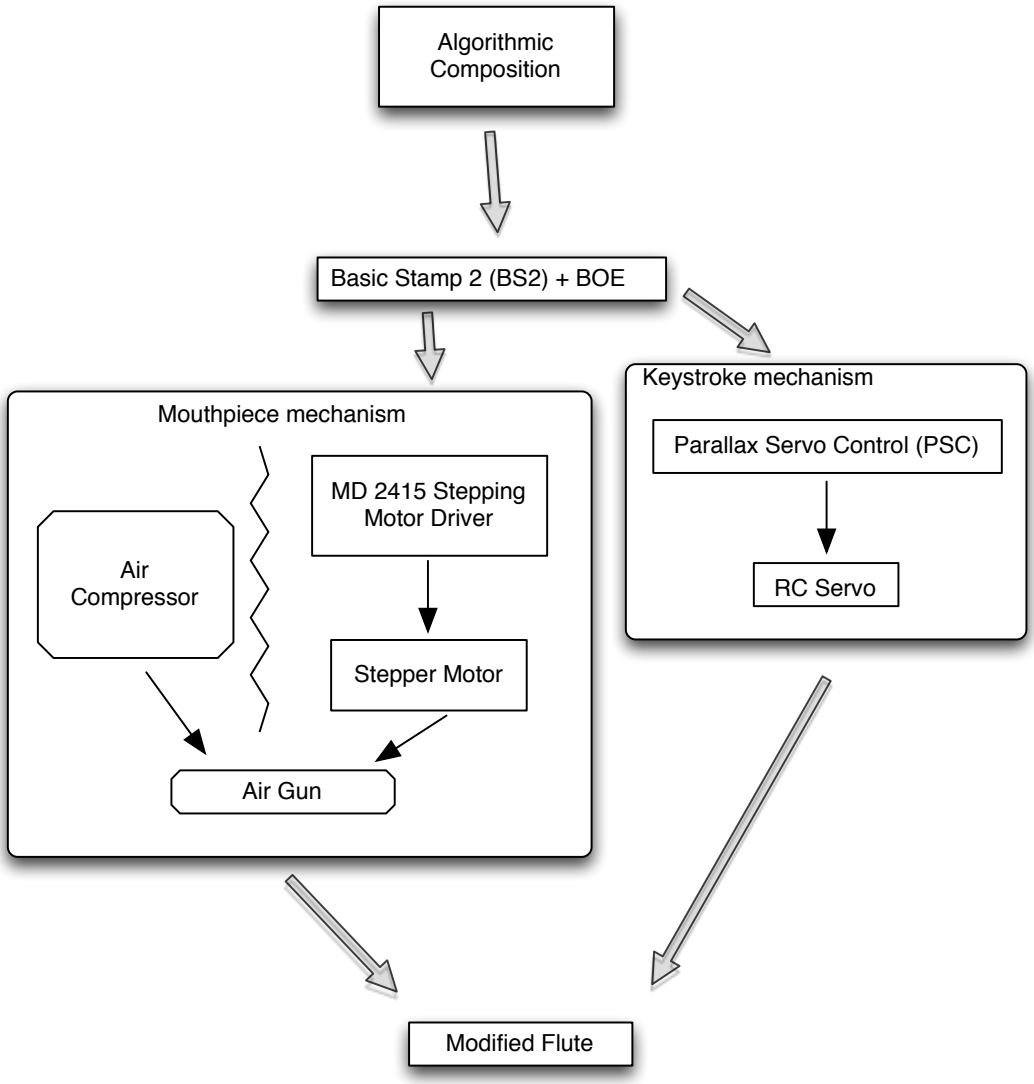


Fig. 2.3.1. System Architecture

The automatic flute system include a modified flute, a Parallax Basic Stamp 2 chip with Board of Education development board (BOE), a Parallax Servo Control board (PSC), a 3-Men micro stepping motor driver (MD2415), a 2-phase stepper motor, 12 RC servos, and an air compressor. It is driven by an algorithmic composition module, realized in Max/Msp and Basic Stamp.

The algorithmic composition module will sent out pitch, duration, and dynamics data. The BS2 then converts them into pwm and pulse signals. The information is further passed down through different pins, to the mouthpiece and keystroke mechanism. The MD2415 will translate the pulses to micro-steps for the stepping motor. The PSC will redirect the pwm signals to different servo address, where each servo addresses have a servo attached to a flute key. When all of the necessary flute keys are in place, the stepping motor will open the valve of the air gun and allow airflow to pass through the air compressor into the mouthpiece of the flute.

Under detache mode, for every note event, the servos and stepping motor will first open, pause for duration of the note, and then return to its original position. Ideally, the servos will always return to the original position, and that is when they are ready for the next note event. Under legato mode, the steeping motor will continue to remain open and will varying its degree of valve cavity per note-event. Which will result in a change of dynamics during a musical phrase. Only till the end of phrase or when silence is required will the stepping motor return to its original position.

The automatic flute is a non-feedback loop system for the most part. After the data is sent out of the BS2, the information is processed in PSC and MD2415 automatically. The exact positions of the servos and the stepping motor after a note-on event cannot be sent back. The position of the stepping motor can be calculated beforehand, but not for the servos. Given the number of servos that are involved and the nature of BS2 and PSC board, it is difficult to trace the position the servos.

Within the mouthpiece mechanism, the air compressor unit is independently controlled. When the tank pressure level is below 300Kpa, the compressor will automatically be turned on and when pressure is above 800Kpa, it will be shut off.

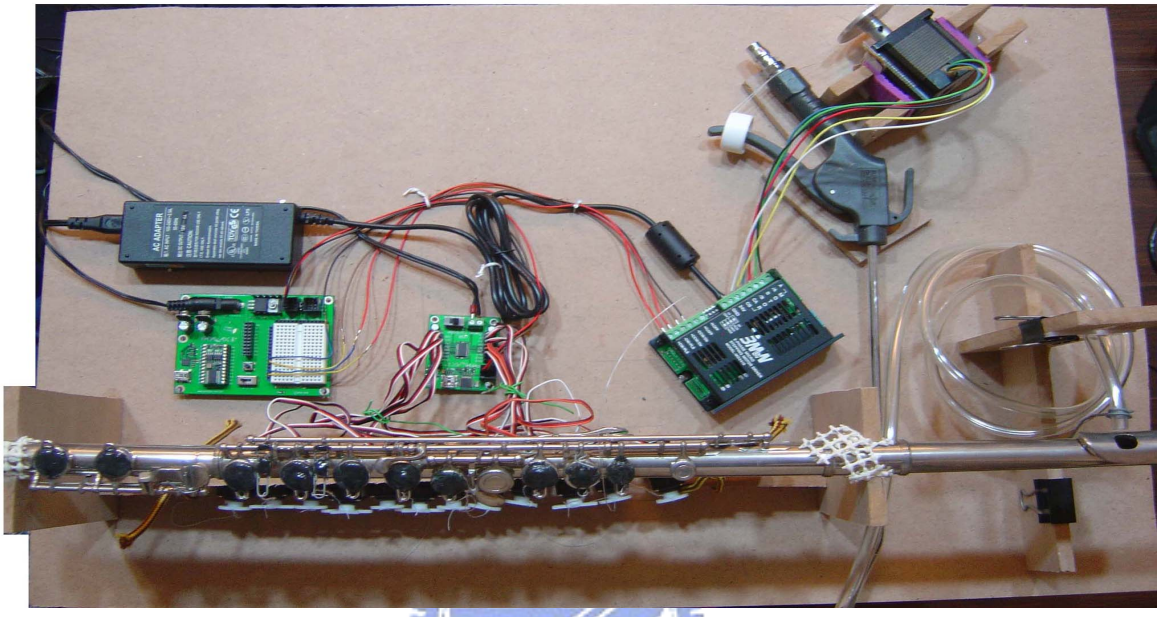


Fig. 2.3.2. Automatic Flute: Bird's-Eye View

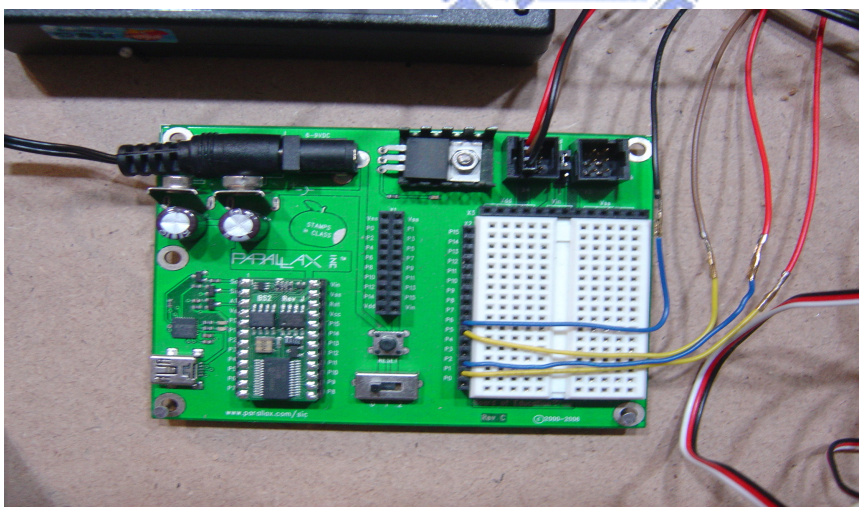


Fig. 2.3.3. Parallax BS2 and BOE Board

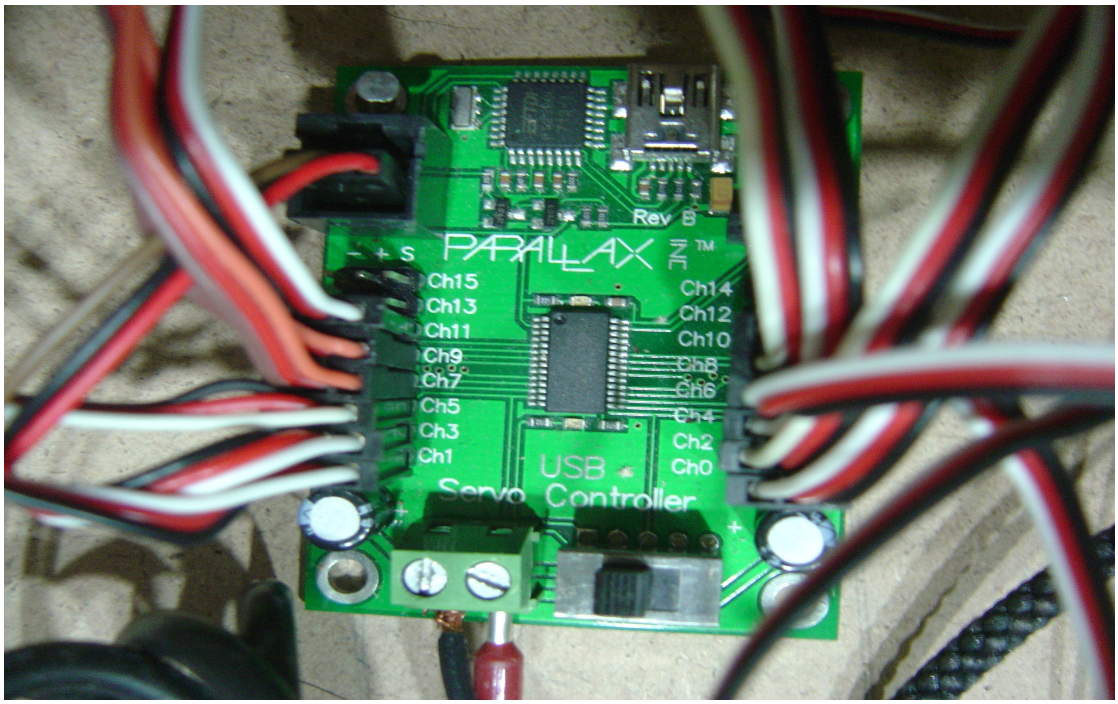


Fig. 2.3.4. Parallax PSC Board

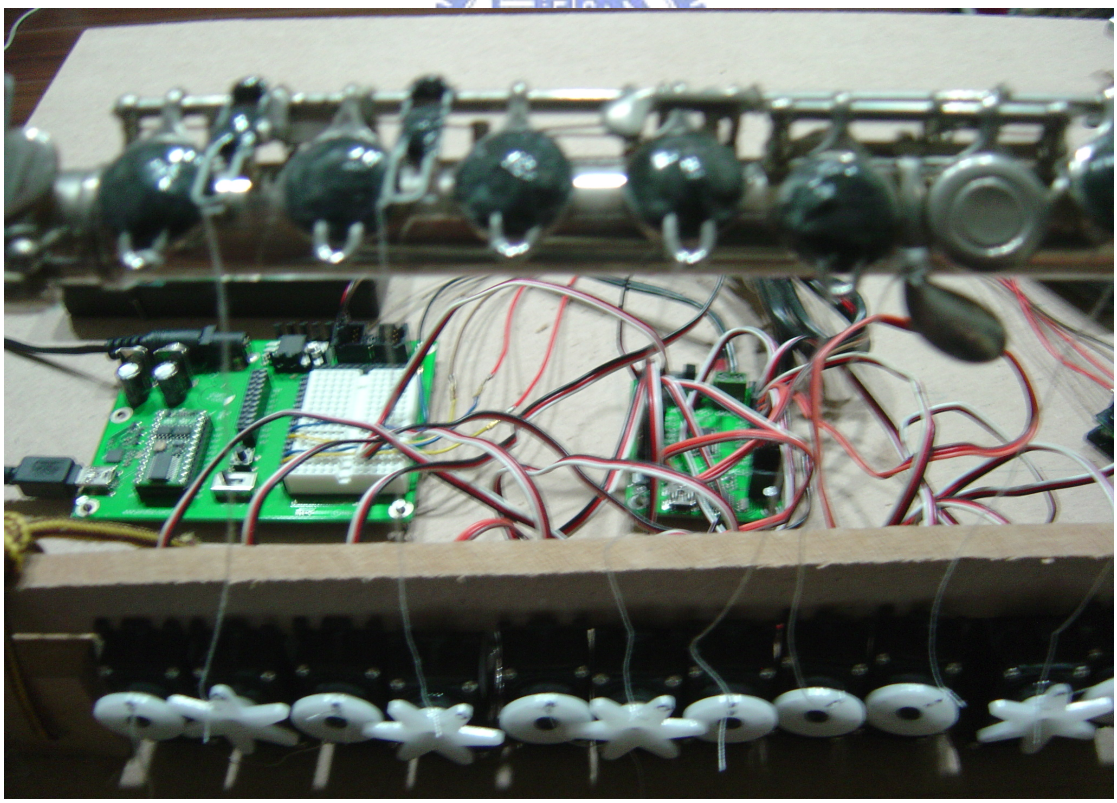


Fig. 2.3.5. Keystroke Mechanism

Fig. 2.3.6. Stepping Motor Driver MD2415

Fig. 2.3.7. Stepper Motor and Air Gun Unit

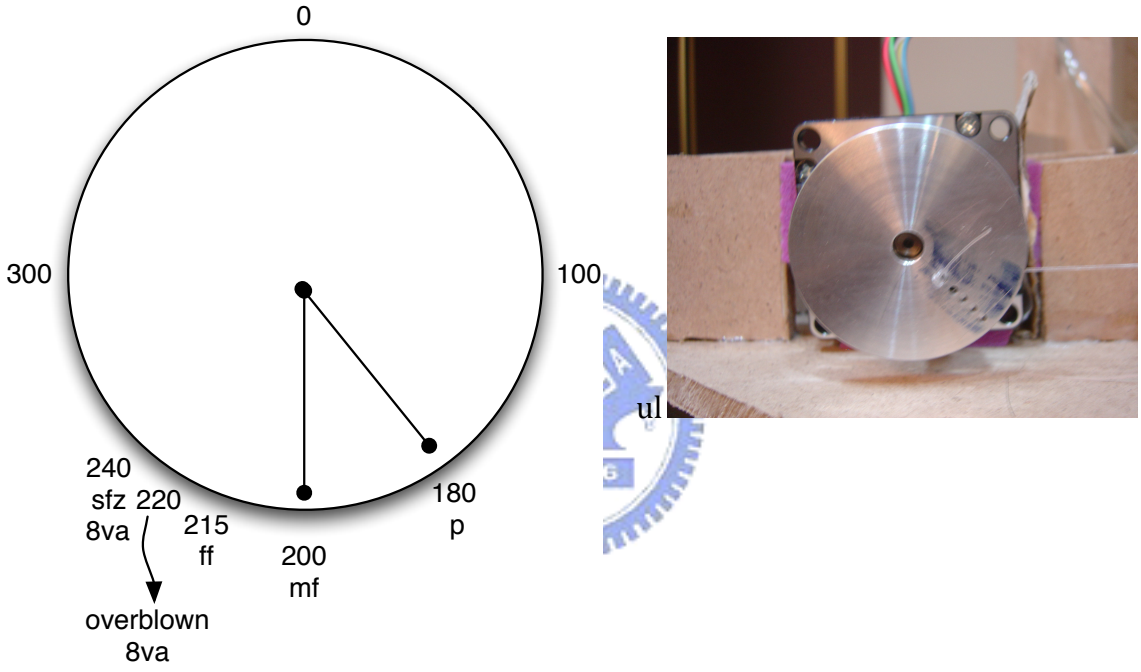
Fig. 2.3.8. Mouthpiece Unit

Fig. 2.3.9. Air Compressor

2.4 Airflow Setup and Fine-Tuning Dynamics

The resolution of MD2415 Stepping Motor Driver is adjusted to 400 steps (Fig. 2.3.7 and 2.4.1). It is calibrated so that at approximately 200 steps the dynamic would be relatively mezzoforte. All the major dynamic changes with its relative number of steps are shown in the later part of figure 2.4.1. Between 200 and 215 steps, the tones are full, solid and more focused. Above 220 steps, we are at the overblown and multiphonic section. At 220 steps, the overblown octave surfaces

and the original fundamental pitch will disappear. As the number of steps increases, the octave will loose focus and other higher partials will join in, creating a multiphonics effect (Fig. 2.2.1). Until reaching 240 steps, all multiphonics will disappear. A sharp and piercing shrill will be created. It is the result of a higher partial above the octave (the exact partial number is inconclusive).



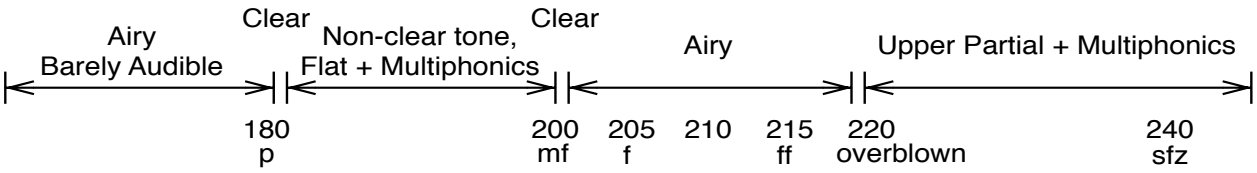


Fig. 2.4.1. Fine-Tuning Dynamics

2.5 Flute Fingering Charts

All of the flute keys are labeled with a number (Fig. 2.5.1). On the music staff, it shows all of the possible good-quality pitches from the current modified flute. Their relative pitch-class representations are placed above the music staff. Each note's representation keys on the flute are shown in the "Pitch Key-List" under the music staff.

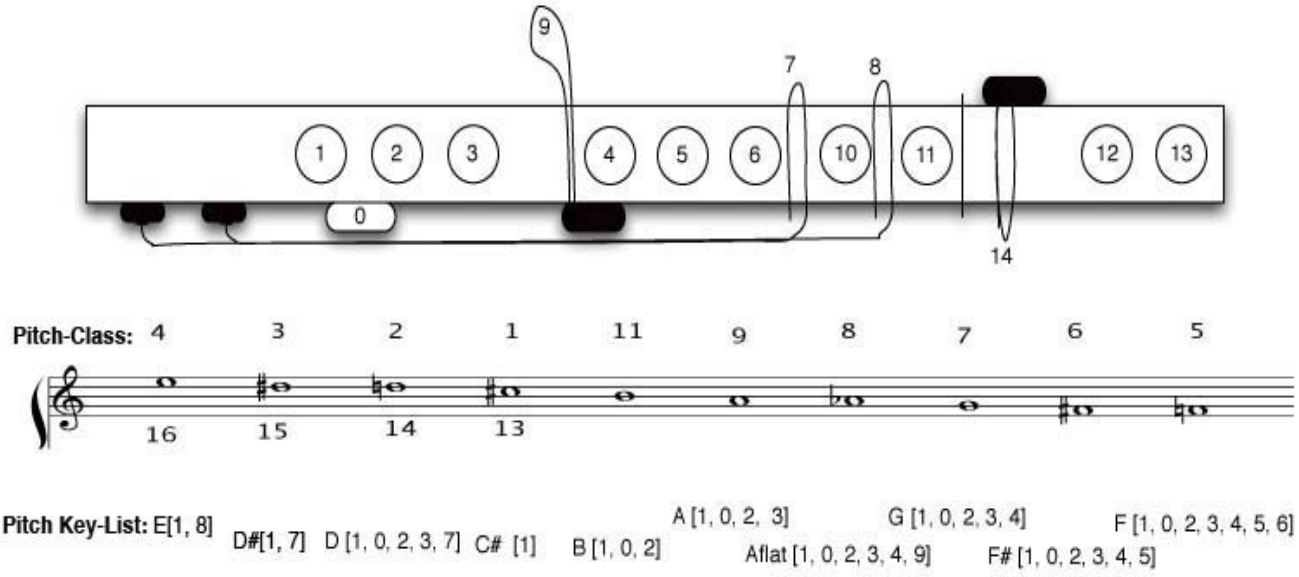


Fig. 2.5.1. Flute Fingering Charts

2.6 Max/Msp and BS2 Communications

The programming language for the Parallax Basic Stamp 2 board is Basic Stamp. The command SERIN, within the Basic Stamp language, allows BS2 to communicate to a foreign programming module. SERIN receives asynchronous serial data through RS-232. A simplified definition of SERIN, extracted from the Basic Stamp manual is shown in figure 2.6.1.

SERIN *Rpin, Baudmode, [InputData]*

Fig. 2.6.1. SERIN syntax

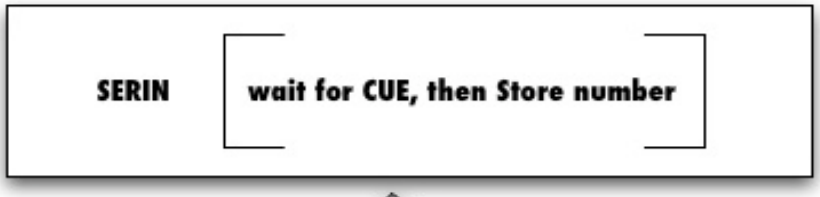
Rpin specifies the I/O pin through which the serial data will be received. Baudmode specifies the important characteristics of the incoming serial: the bit period, number of data and parity bits, and polarity. Input data contain a list of variables and formatters that tells SERIN what to do with the incoming data[4].

A sample code is shown in figure 2.6.2. The input data, in square brackets, will wait for the specific characters that are in between the quotation marks to be received and then assign the numerical value that are attached afterwards to the corresponding variable.

For example, in figure 2.6.2, the code will wait for the characters “t”, “i”, ”m”, and ”e” to be received, in the same order, then it looks for the number that follows, and last assign them to the variable “time”. If an outside programming module sent out “time123” the SERIN will first search for “t”, “i”, ”m”, and ”e” and then assign “123” to the variable “time”.

When Basic Stamp is connected to a PC, the data that are transferred will be in ASCII code. Therefore, command DEC, a decimal formatter specifics, will be needed in the Input Data of SERIN. It will convert the ASCII code into the actual numerical value and store them in the appropriate variable.

[4] SERIN of Basic Stamp manual




```
DO    'big serial loop

SERIN 16, 16780, [WAIT ("time"), DEC time, WAIT ("aa"),DEC aa, WAIT ("bb"),
DEC bb, WAIT ("cc"),DEC cc, WAIT ("dd"), DEC dd, WAIT ("ee"),DEC ee,
WAIT ("ff"), DEC ff, WAIT ("gg"),DEC gg, WAIT ("hh"), DEC hh, WAIT ("ii"),
DEC ii, WAIT ("jj"), DEC jj, WAIT ("kk"),DEC kk, WAIT ("ll"), DEC ll,
WAIT ("mm"), DEC mm, WAIT ("stp"), DEC change]
```

Fig. 2.6.2. SERIN sample

The majority of the algorithmic composition module is realized in Max/Msp. Max/Msp is a graphical development environment for music and multimedia[5]. In order for Max/Msp to communicate with BS2, the output data need to be assembled into the format that is understandable for the SERIN command. In Max/Msp, the object “serial”, send and receive characters from the serial ports (Fig. 2.6.3). The “c” follows the “serial” indicates the usb port name and the “2400” indicates the baud rate. The 2400 Baud Rate’s no parity, 8-bit, and inverted equivalence would be 16780. It is shown in the command SERIN of Basic Stamp (Fig. 2.6.2). Now both of the baud rate, from command SERIN and object “serial”, are properly tuned.

[5] It is developed and maintained by a San Francisco software company Cyling 74.

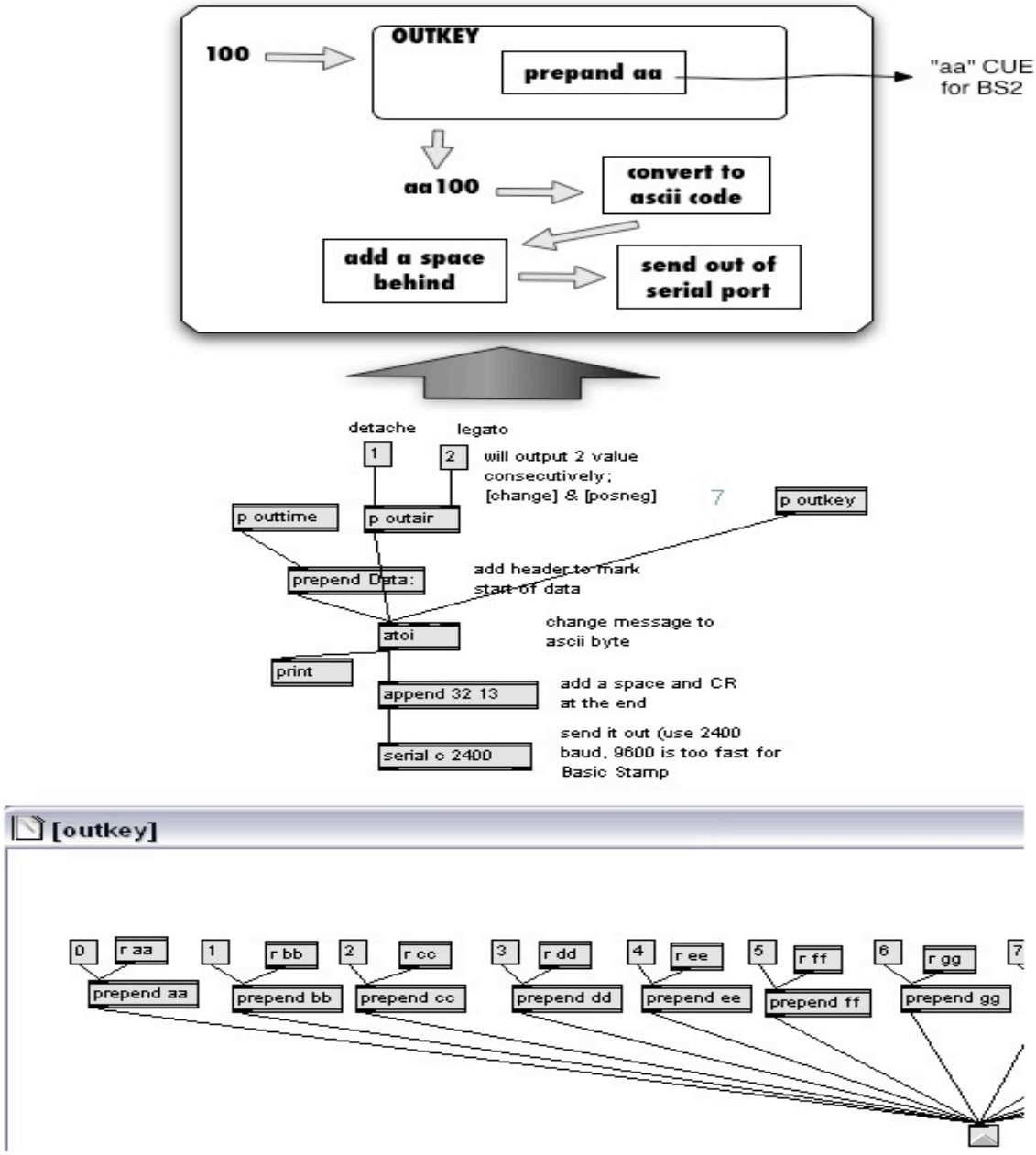


Fig. 2.6.3. Max/Msp Data Preparation for SERIN

Let us continue to examine the data preparation procedures within Max/Msp. The object “prepend”, will add a header before any variables that are passing through. For example, in the subpatch “outkey”, lower portion of figure 2.6.3, if a variable, “123”, passes through the object “prepend aa” the resulting data would

become “aa 123”. The data, “aa 123”, will later be converted into ascii code, have a space attaching to its tail, and sent out of the object “serial”. In the time being, the command SERIN of Basic Stamp, will wait for the data coming in. Only until the character “aa” is received, will command SERIN begin colleting the variables afterwards. It will ignore all data received if the condition “aa” is not met.

2.7 MD2415 and Stepper Motor Control

MD2415 Stepping Motor Driver, can operate in either 1-phase or 2-phase mode (Fig. 2.3.6). Its step resolution ranges from 200, 400, 800, and 1600. The optimal step resolution for the current system is 400 steps. The specification of the pulse that is required for the MD2415 to run in 2-phase mode is shown in figure 2.7.1. Each pulse’s duration and the distance between successive pulses need to be above 5 microseconds. The time between clockwise pulse and counter-clockwise pulse needs to be at least 10 microseconds apart.

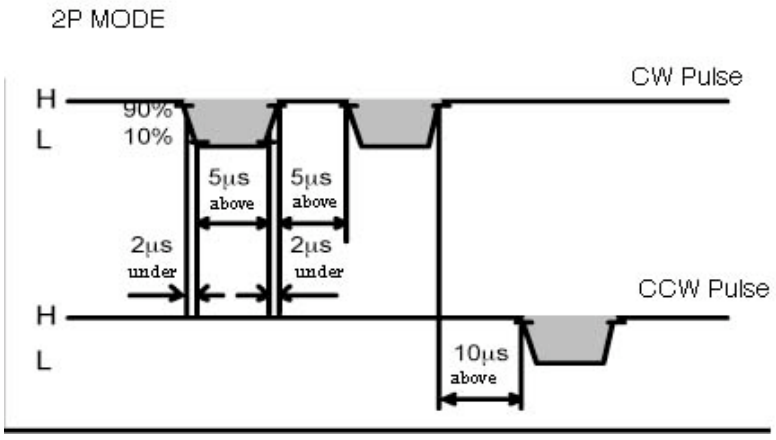


Fig. 2.7.1. MD2415 Pulse Specifics

The command PULSOUT within Basic Stamp, will generate a pulse on a pin with a specified width of duration (Fig. 2.7.2). The unit of duration is 2

microseconds and the maximum pulse width is 131.07 milliseconds. To meet the specification requirement of MD2415, the duration of PULSOUT is set to 10, which equals 20 microseconds (fig.21).

PULSOUT *Pin, Duration*

Fig. 2.7.2. PULSOUT syntax

The wiring between the BOE board and the MD2415 are shown in table 2.7.1. The first yellow and blue wire pair connects pin 1 and 2 of the BOE board to "+CW/PLS" and "-CW/PLS" of the MD2415 respectively (Fig. 2.3.3 and 2.3.6). The second yellow and blue wire pair connects pin 5 and 6 of the BOE board to "+CCW/Dir" and "-CCW/Dir" of the MD2415 respectively.

BOE	**MD2415**
Pin 1	+CW/PLS
Pin 2	-CW/PLS
Pin 5	+CCW/Dir
Pin 6	-CCW/Dir

Table 2.7.1. BOE and MD2415 Connections

In figure 2.7.3 the command LOW, before every PULSOUT loops, basically sets the specified pin to zero (a 0 volt level) and then sets its mode to output. When pulse is sent to pin1 and a low current (close to 0 volt) is passed through pin2 the MD2415 will rotate the stepping motor in the clockwise direction. If the same pulse and current is sent through pin5 and pin6, the stepping motor will rotate in counter clockwise direction. Every counter-loop represents one pulse and one step. If

“change” equals 400, which the loop will run 400 times, the stepping motor will move 400 steps, which equals one full rotation.

```
stepopen:
LOW 2
  FOR counter = 1 TO change
    PULSOUT 1, 10
    NEXT

RETURN

stepclose:

LOW 6
  FOR counter = 1 TO change
    PULSOUT 5, 10
    NEXT
```

Rotate CW
change = # of steps
400 steps = 1 complete rotation

Rotate CCW
change = # of steps
400 steps = 1 complete rotation

Fig. 2.7.3. Step Rotation Commands

2.8 PSC and RC Servo Control

Parallax Servo Control board, can control up to 16 servos simultaneously (Fig. 2.3.4). It is connected to the BOE board with a 3-pin lead connector. The setup of PSC with BOE is simpler comparing to the MD2415 with BOE. However, controlling the position of the servos with PSC is more complex than that of the stepping motor with MD2415. The position of the servos could not be precisely controlled. The syntax of position control in PSC, with the command SEROUT is shown in figure 2.8.1.

SEROUT PSC, N2400, [“!SC” C R P.LOWBYTE, P.HIGHBYTE, $0D]

```
SEROUT PSC,N2400,["!SC",servoaddr, 1,position.LOWBYTE, position.HIGHBYTE, CR]
```

Fig. 2.8.1. SEROUT syntax

“!SC” is the header for the PSC. “C” parameter indicates the servo address (0 to 15). “R” indicates the ramp speed (0 to 63). Ramp values of 1-63 correspond to speeds from ¾ of a second up to 60 seconds for a full 500uSec to 2.50 mSec excursion[6]. “P” indicates the position of servo (250-1250). The 250-1250 corresponds to 0 to 180 degrees of servo rotation with each step equaling 2 uSec.

The ramp values are insignificant to the overall speed among the servos. It is given a value of 1, which is the fastest throughout. The word “position” is misleading in that there are actually two constituents that determines the final position. One being the “P” value and the other is time. The “P” in the syntax of figure 2.8.1 could be imagined as the velocity of rotation. The centre value 750, half way between 250 and 1250, represents zero velocity (Fig 2.8.2). “P” values above 750 and towards 1250 will cause the servo to rotate in counter clockwise direction. The higher the “P” values, above 750, the faster the rotation (in CCW). The same principles apply to “P” values below 750 and towards 250. The lower the “P” values, below 750, the faster the rotation (in CW).

[6] Parallax, *Parallax Servo Controller (#28023) – Rev B manual,* (California: Parallax, Inc., 2004), 5.

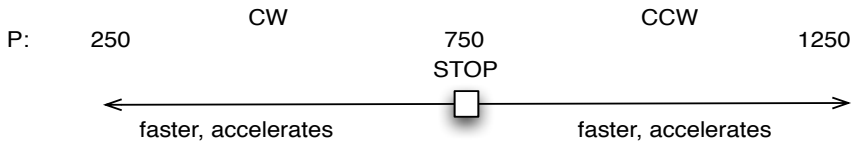


Fig. 2.8.2. Velocity equivalent of the “P” Value

Given the fact, if running the command line in figure 2.8.1 by itself, it will cause the servo to rotate in the specified direction indefinitely. The servo will continue to rotate until the same command line is applied again but with a “P” value of 750. Therefore to control the position of the servo, one need specified how fast, which includes the direction, for how long, and when to stop. Figure 2.8.3 is a simple example that will make the servo turn clockwise, at its fastest speed for half second. It will then stop for a second and then turn in the reversed direction, at its fastest speed, for another half second. In the end it will stop again.

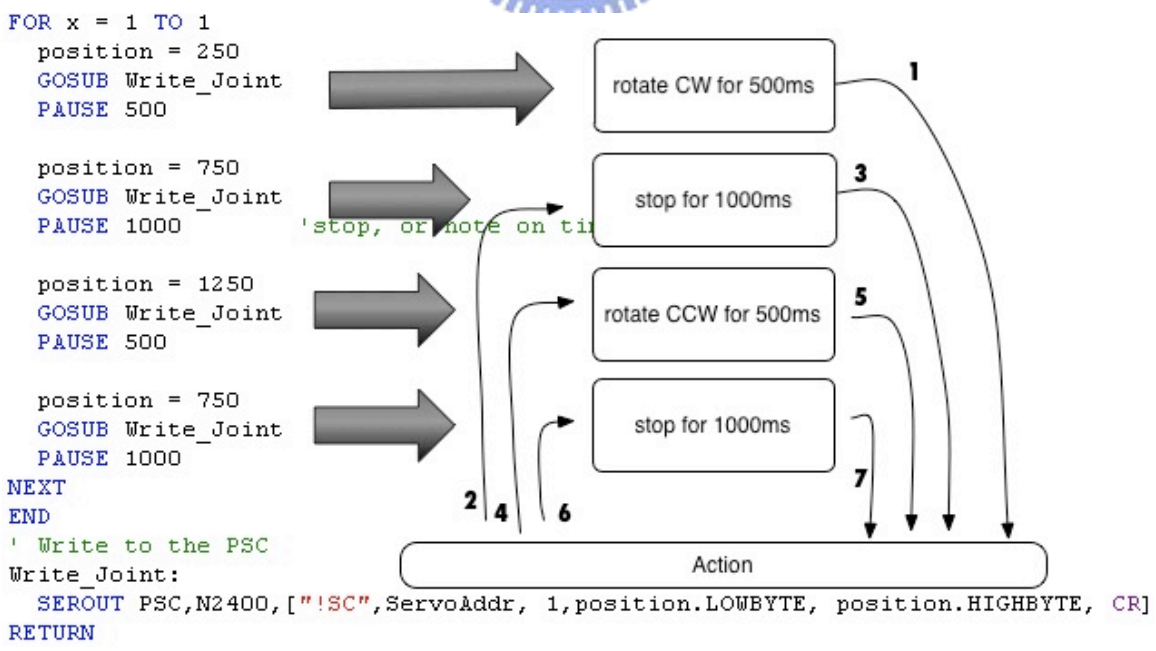


Fig. 2.8.3. NoteOn/NoteOff: sample Servo Rotate Command

At this point it is important to note that not all servos are exactly alike, which means a “P” value of 1250 will have different effects on two different servos. It is important to calibrate each servo accordingly, so that all of the servo could rotate in identical velocity. This issue will be further addressed in the implementation section.

2.9 Algorithmic Composition with Probability Table

The core of the algorithmic composition module is realized in Max/Msp. Probability theory was used as the main principle of design. It was the favored method for the algorithmic composition in that the current automatic flute system posses enough problems in itself that it is not capable of carrying out more complex algorithms. In plain words, the current automatic flute is like a novice flute player. If requesting him/her to play a big piece, even if carried out, the interpretation of the music will not even be close to a true representation of the score (<60% resemblance to score).

Probability by definition, for the current module, means that the number of occurrence of certain pitch, durations, or dynamics is represented as percentages. For example, a pitch that has a 90 percent weight within the probability table would occur close to nine out of ten times. Figure 2.9.1 shows a sample module of the probability table for the dynamics. The main Max/Msp objects that are used are “histo” and “table”. “Histo” will remember and keep count of the numbers that are inputted from its top left inlet. The left outlet of “histo” will output the number, for example X, and the right outlet will output the number of occurrences of X.

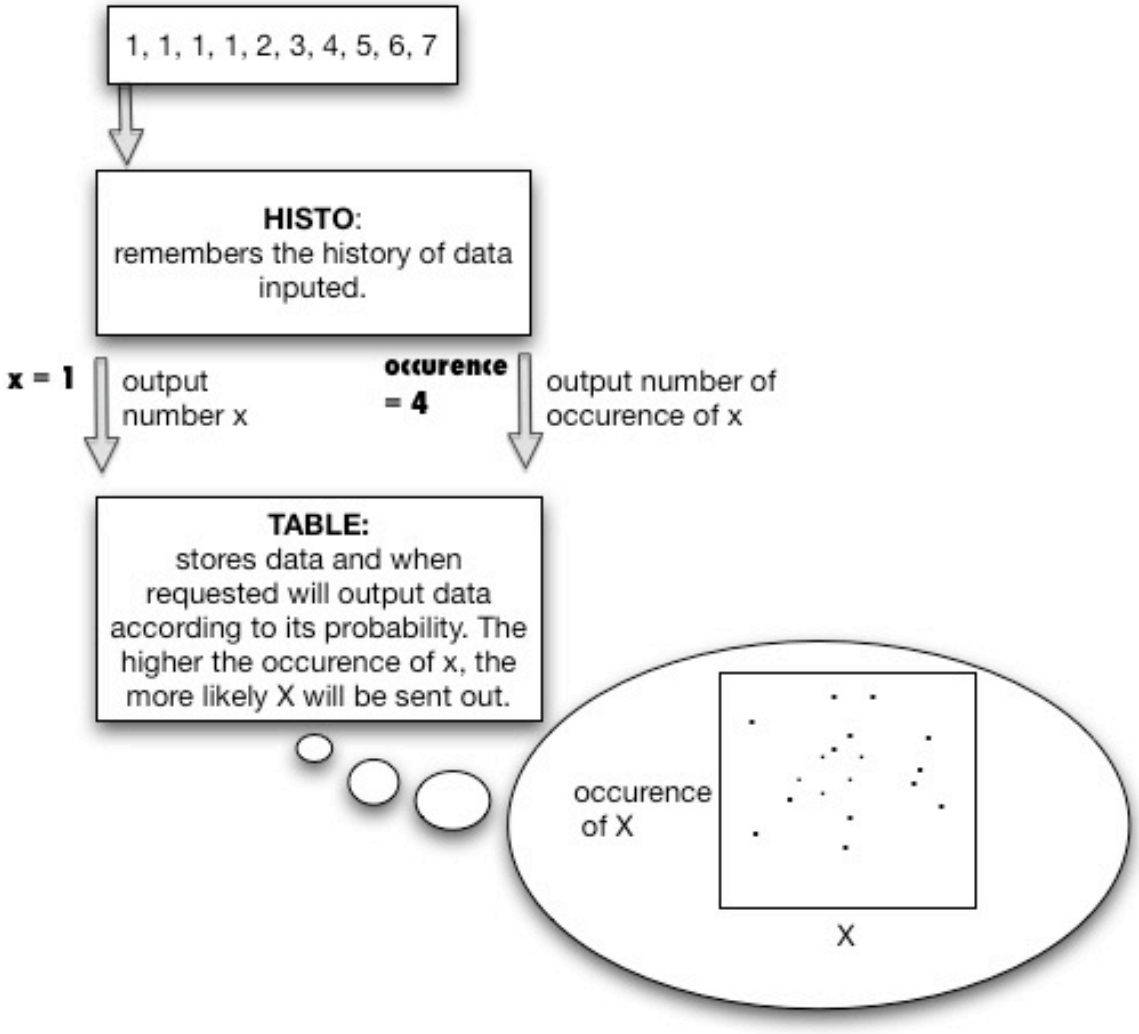


Fig. 2.9.1. Probability Module Summary

All of the information will be store in the Max/Msp object “table”. Within “table”, the x-axis represents the number X, and the y-axis represents the number of occurrences of X. The chance of the particular X to be sent out, when a “bang” message is applied to the object “table”, to request a number from table, will be in direct proportion to its stored value (number of occurrences).

All of the probability tables that are used in the algorithmic composition module are divided into three different categories: duration, dynamics, and pitch. Each category contains two probability tables, each representing a specific characteristic (mode). Figure 2.9.2 and 2.9.3 are the two duration probability tables. In Dennis mode, the probability distribution of the note durations is custom made to my own liking. In Short-Long mode, the options of note durations are limited to two. The Short-Long mode is necessary in that it will prevent the music into becoming overly complex. It will shift the attention to other aspects of the music and sometimes provides the proper emphasis on certain pitch over the other.

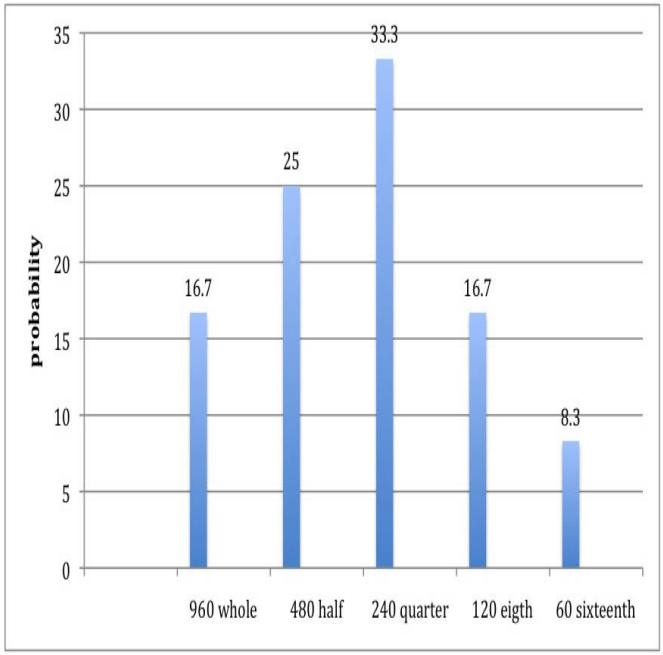


Fig. 2.9.2. Dennis Mode

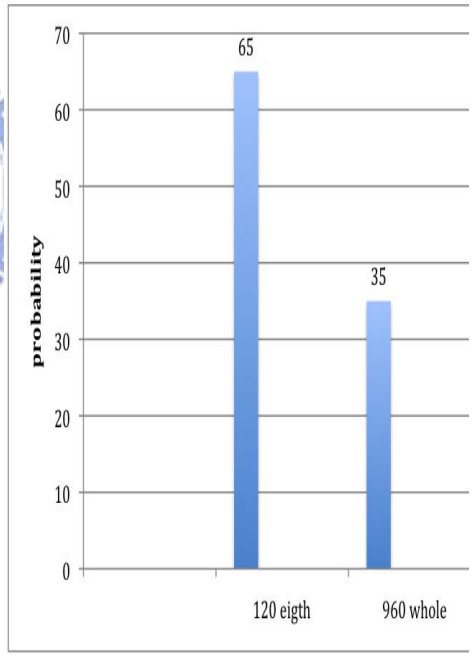


Fig. 2.9.3. Short-Long Mode

Figure 2.9.4 and 2.9.5 are the two dynamic probability tables. The not-too-loud mode, speaks for itself, represents the comfortable dynamic ranges for the ear.

The multiphonics mode, on the surface appears to be tamed, due to the equal distributions. In reality it is chaotic. The increment of the steps, which increase the air valve cavity, not only increase the dynamics linearly, but also change the combinations and the number of the higher harmonics that are involved. In simpler terms, not only do the dynamics change, the timbre also changes.

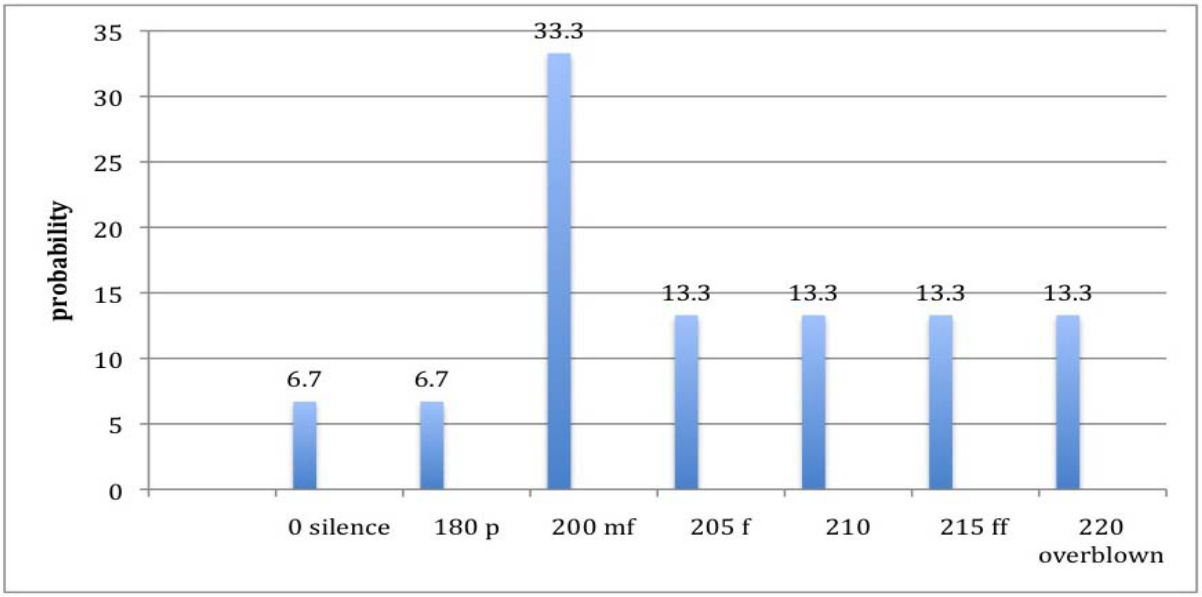


Fig. 2.9.4. Not too Loud Mode

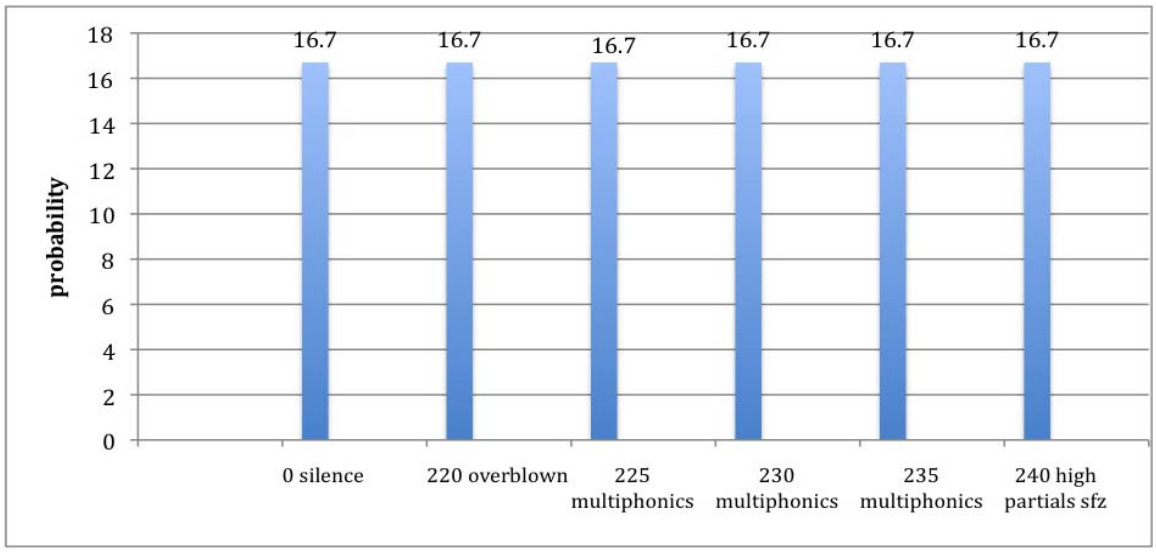


Fig. 2.9.5. Multiphonics Mode

The above two probability categories are in higher hierarchy than the pitch probabilities. When the dynamic equals to zero, the automatic flute will still play out the silence for a specific duration. However, a pitch note will not be required from the pitch probability tables.

Figure 2.9.6 and 2.9.7 are the two pitch probability tables. The less-error mode, contain pitches that require to hold down four keys or less. They will be less susceptible to error. The A-major like mode, contain pitches from the A-major scale and place more emphasis on the tonic and the dominant.

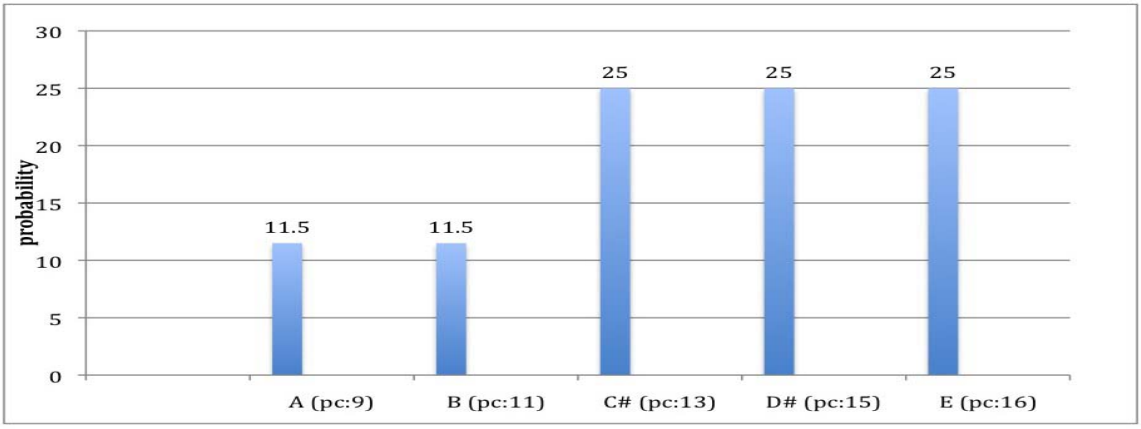


Fig. 2.9.6. Less-Error Mode

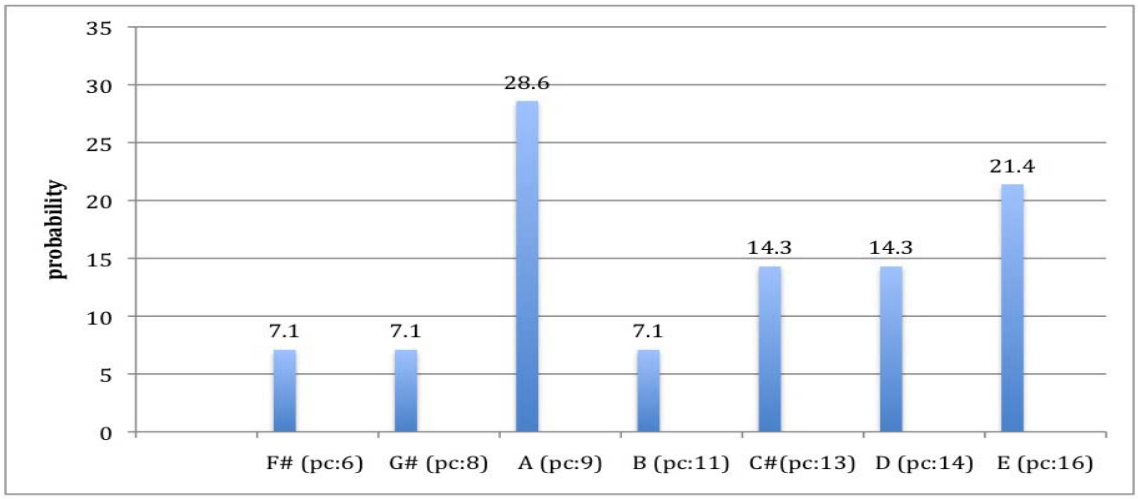


Fig. 2.9.7. A-Major Like

2.10 Interactive Platform: Cellular Automata with Automatic Flute

This is an ambitious attempt to incorporate the automatic flute within the evolution process of three other melodies from the Cellular Automata module. Cellular Automata (CA), a category of Genetic Algorithms, uses the processes of evolution from natural selections to determine whether an individual come to exist, continue to exist, or cease to exist.[7] In music, metric modulation, can be said to have the similar qualities to that of the Cellular Automata. Metric modulation by definition is a change (modulation) from one tempo (meter) to another wherein a note value from the first is made equivalent to a note value in the second, like a pivot.[8] A more concise description and how it is achieved by Elliott Carter is:

> There is constant change of pulse. This is caused by an overlapping of speeds. Say, one part in triplets will enter against another part in quintuplets and the quintuplets fade into the background and the triplets establish a new speed … The structure of such speeds is correlated throughout the work and gives the impression of varying rates of flux and change of material and character.[9]

In plain terms, there are two degrees of metric modulations. One is horizontal, which concerns the tempo change of a voice. The other is vertical, which concerns the tempo change of a voice relative to the others. The vertical degree of metric modulation will allow greater independence of voices, and therefore capable of creating greater contrasts. In our Cellular Automata model we will only concentrate on the vertical type of metric modulation. The tempo and velocity change of a voice can either be influenced by others or be influential to the others.

---

[7] David Cope, *Computer Models of Musical Creativity*, (Massachusetts: The MIT Press, 2005), 66.

[8] Wikipedia contributors, "Metric modulation," *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/w/index.php?title=Metric_modulation&oldid=107710843 (accessed January 10, 2007).

[9] Elliott Carter, "Shop Talk by an American Composer, " in *Collected Essays and Lectures, 1937-1995, ed.* Jonathan W. Bernard (Rochester: University of Rochester Press, 1997), 218.

Four characteristic modes, “A”, “B”, “C”, and “D” are created (Fig. 2.10.1). The four musical lines in SATB (Soprano, Alto, Tenor, and Bass), will begin with an initial probability-generated pulse and dynamics value. After a period of time, each musical line will then be categorize, according to its pulse and dynamics value, into zone “A”, “B”, “C”, and “D”. If at the specific time, the mode setting was set to “A”, the musical lines that are in zone “A” will be consider as the strongest. Musical lines that are in zone “B” will be considered as the next strongest. Musical lines that are in zone “C” will be the next in rank and the ones that are in zone “D” will be the weakest. The order of the rank is measure by first its pulse value than the dynamic value. The strongest will survive. Those that are in zone “B” and “C” will gradually adapt to become more “A” like. The weakest will disappear for a while and will rejoin at a later period. This cycle continues. Depending on the outcome of the adaptation of the 2nd and 3rd strongest musical lines and the specific mode setting at the time, the order of the ranking will continue to change and therefore the competitions among the four SATB lines will change. Figure 2.10.2 shows the Cellular Automata module.

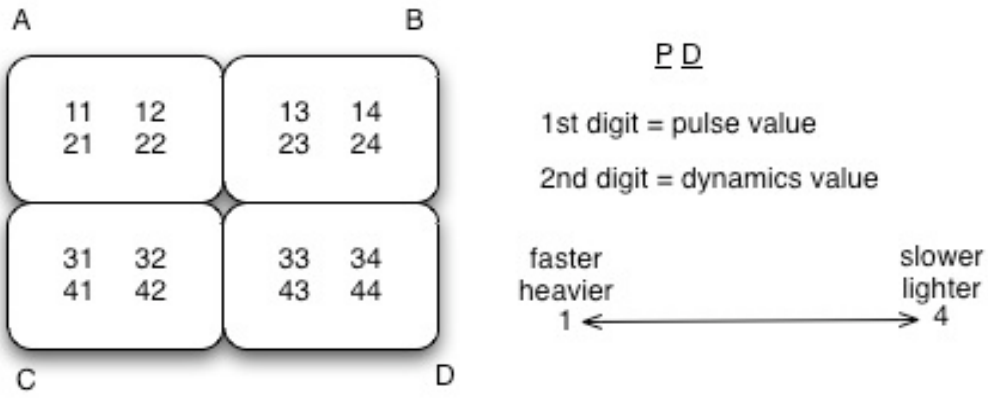


Fig. 2.10.1. Four Characteristic Modes: A, B, C, and D for Cellular Automata

To include the automatic flute into the CA process, the Alto voice's data will be transferred to the algorithmic composition module of the flute. Instead the automatic flute will carry out the Alto line and then participate in the cycle of Cellular Automata.

CHAPTER 3

IMPLEMENTATION

3.1 Data Preparation in Max/Msp

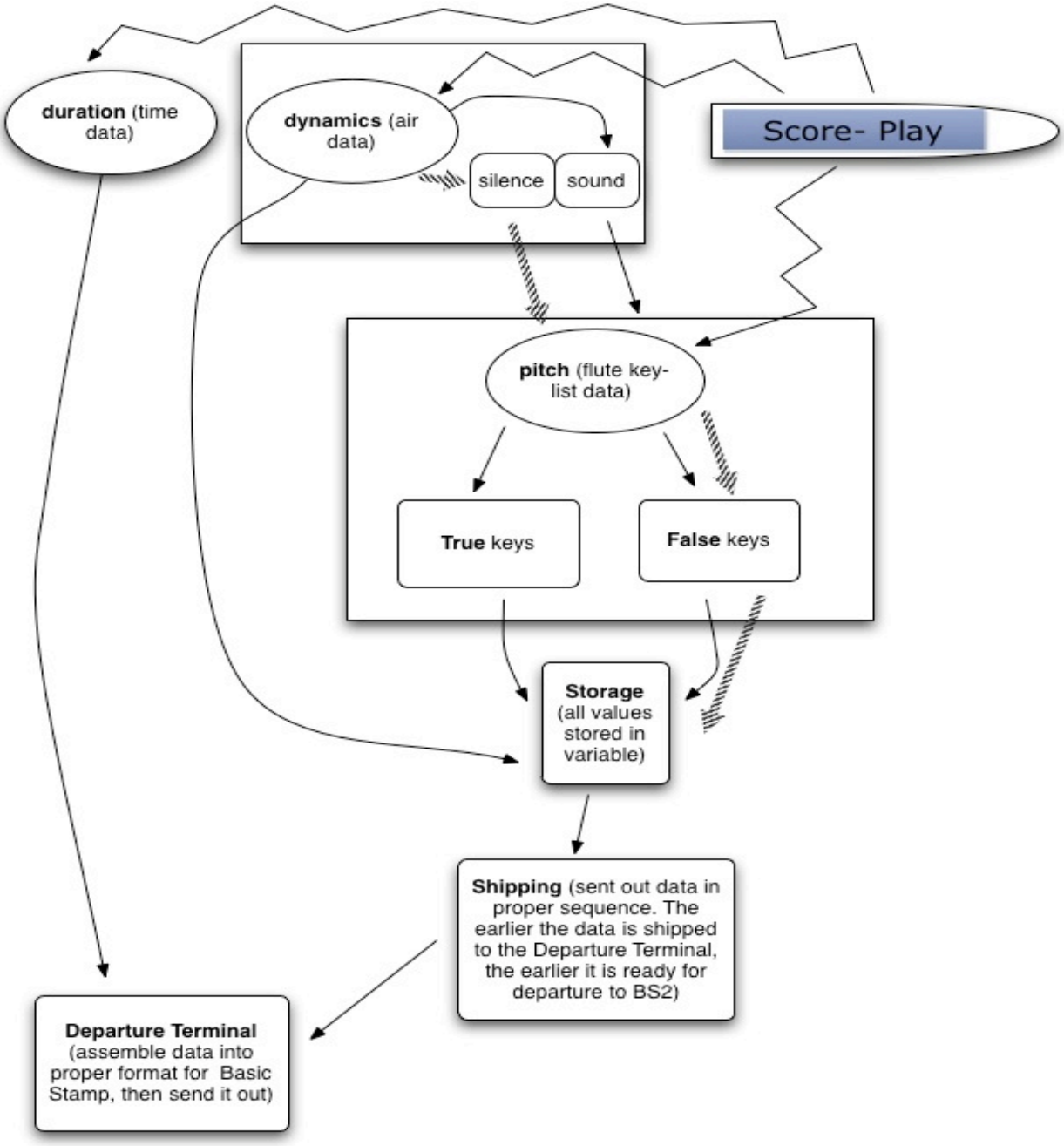


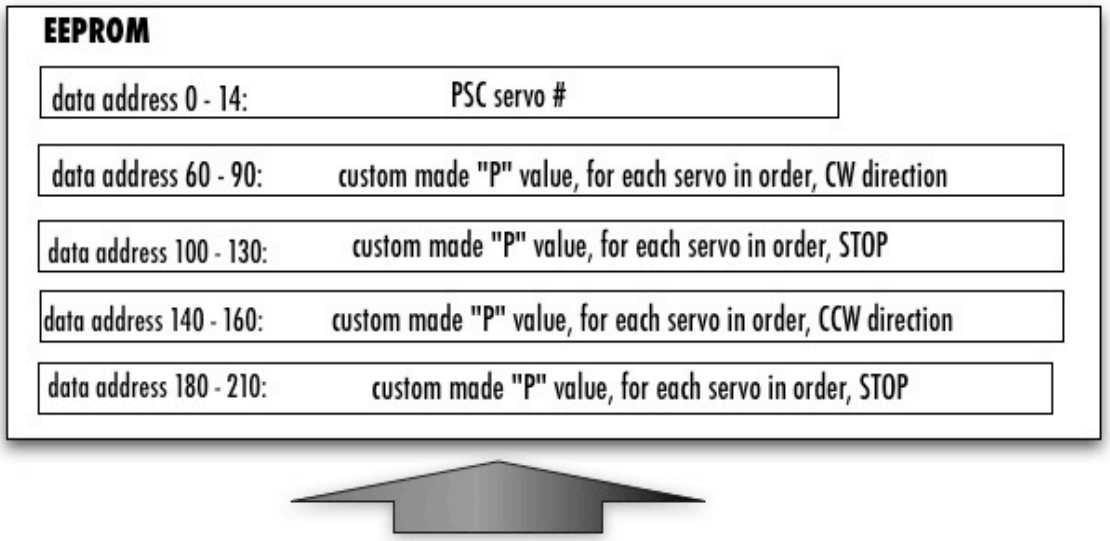
Fig. 3.1.1. Data Preparation in Max/Msp

Duration and dynamics module both operate independently of each other (Fig. 3.1.1). The duration-module will send data direct to the departure-terminal module, because the duration data are the first data required by BS2. The dynamics module, will either output air or no air. If the air condition is met, it will then trigger the pitch module to generate a pitch from the probability table. Every pitch will have a set of true-key-list and false-key-list. All of the proper key information deduced from the key-lists will be entered in the storage module. Later passed on to the shipping module and then the departure-terminal module. The specific functions of the modules are shown in figure 3.1.1. If the dynamics module outputs silence, it will then trigger the pitch module to automatically assign all keys to false. Eventually, the automatic flute doesn't have to move the keys while silence lasts for the intended durations.

If the system is under Score-Play mode, the duration, dynamics, and pitch module will not generate data according to the probability tables. Instead, the three-parameter-modules will receive the data from the Score-Play module.

3.2 Processing Data in Basic Stamp: Problems and Solutions

In section 2.8, we mention that each servo needs to be calibrated differently. If applying the command SEROUT, with a single "P" value, to 12 identical servos, each will actually return a different result (Fig. 2.8.1). Depending on the duration of the command, when given enough time, greater than 20 seconds, all servos will be pointing to a different clock position. It is not possible to have 12 WORD data type



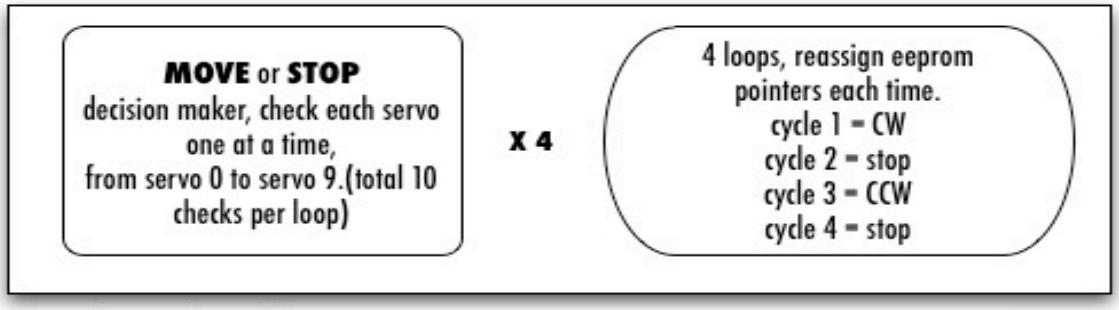

```
DATA  0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14

' put bunch of 250 ... byte size not word size.... pointer doesn't need to be
'every CW OR CCW pointer2 is reassigned
DATA @60, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,250, 25C
'
DATA @100, Word 750, Word 750, Word 750, Word 750, Word 750, Word 750,Word 75
'           100         102         104         106      108       110        112
ccw DATA @140, Word 925, Word 950, Word 987, Word 945, Word 908, Word 946, Wc
      'servo#    0           1           2        3          4          5

DATA @180, Word 750, Word 750, Word 750, Word 750, Word 750, Word 750, Word 7
```

Fig. 3.2.1. EEPROM Storage

variables for the servos. There is not enough of RAM space for BS2. Instead the solution was to store all of the calibrated "P" values in the eeprom and using pointers to locate the specific "P" values with the correct servos and at the specific

time. This will only temporarily resolved the problem of calibrating the servos. The issue will be discussed later.

The SERIN command line in figure 2.6.2 shows a list of variables that will be received from Max/Msp. After receiving all of the variables, BS2 will send them out to the decision maker, “MOVE or STOP” (Fig. 3.2.2). If a particular servo is being assign as the true-key from Max/Msp, it will pass its conditions in figure 3.2.2, such as “ee = 4” and continue to the subrountine “turnbaby”. Under the subrountine “turnbaby”, the servo will be executed and the pointer address of the corresponding PSC servo address will move up by one. The pointer address of “P” value will move up by two (Fig. 3.2.3). If a servo is being assign as the false-key from Max/Msp, it will fail the condition in figure 3.2.2 and will continue to the subrountine “freeze”. Under the subrountine “freeze” nothing would happen except that both of the pointer address, PSC servo address and “P” value address, will move up in a similar fashion.



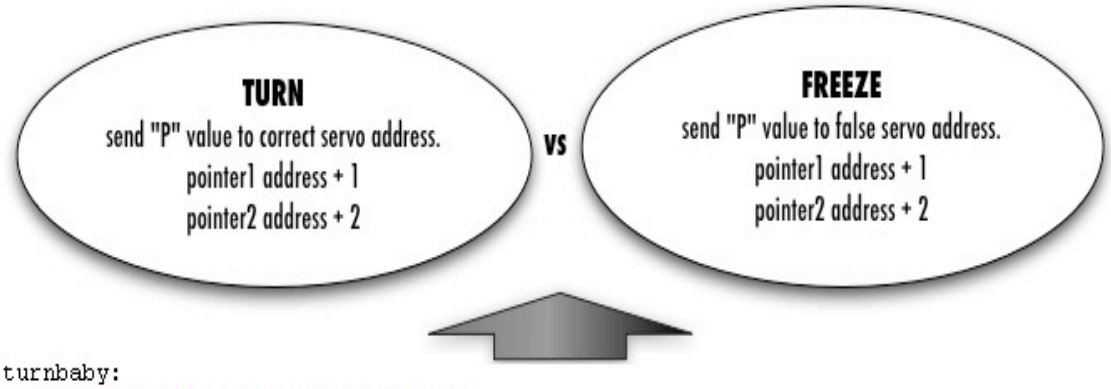

```
 pointer2 = 60

FOR x = 1 TO 4

 pointer = 0
IF aa = 0 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF bb = 1 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF cc = 2 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF dd = 3 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ee = 4 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ff = 5 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF gg = 6 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF hh = 7 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ii = 8 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF jj = 9 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF kk = 10 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF ll = 11 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF mm = 12 THEN GOSUB turnbaby : ELSE GOSUB freeze


  IF x = 1 THEN PAUSE 100
  IF x = 1 THEN pointer2 = 100
  IF x = 2 THEN GOSUB stepopen   'when all servo oper
  IF x = 2 THEN PAUSE time
  IF x = 2 THEN GOSUB stepclose   'since when all ser
  IF x = 2 THEN pointer2 = 140
  IF x = 3 THEN PAUSE 100
  IF x = 3 THEN pointer2 = 180
'  IF x = 4 THEN GOSUB stepclose
NEXT
LOOP
```

Fig. 3.2.2. Move/Stop Decision Maker for Servo

After calibrating the servos with differentiated “P” values in the eeprom, all of the servos are expected to be in proper position most of the times. However, it only happens eighty percent of the time while one or two servo still went out of position on different tests. In reality to make matters worse, those eighty percent

only happen when all of the servos are moving, drawing identical currents from the single PSC. Every time a different pitch is requested, the combination of the servos and the total number of the servos will be different. This will draw a varying degree of electric currents to different PSC's servo address each time. As a result some of the servos will be pointing to odd clock positions after playing more than five pitches. The solution to even out the electric current differences between pitches is to send a SEROUT command to an empty PSC address when the particular servo is not required to move (Fig. 3.2.3 freeze subroutine). This way the total current that is applied to every pitch should be identical and the executing time between "turnbaby" and "freeze" subrountines will be made equal.

Another problem exists when running the Basic Stamp codes in figure 3.2.2, servos that are higher in the code, "aa = 0", which also passed the condition will be executed first. The servo who are lower in the code, "mm = 12", will be executed last. The first servo in the question actually has an earlier start than the last. Even though each servo's total duration is identical given that the first servo also stop earlier than the last servo, the time offsets between the servos will reduced the total duration of the actual pitch. Part of the solution is to shorten the time differences by eliminating those servos that are not involved in the possible pitch list (Fig. 2.5.1). As a result 4 lines of the code in figure 3.2.2 was eliminated (in green).




```
turnbaby:
   '5 pointers servo need to be ran
READ pointer, servoaddr
IF (pointer2 < 90 ) THEN READ pointer2, position : ELSE READ pointer2, Word position
' 250 is a byte and therefore need to DATA and READ it differently

SEROUT PSC,N2400,["!SC",servoaddr, 1,position.LOWBYTE, position.HIGHBYTE, CR]
    pointer = pointer + 1
    pointer2 = pointer2 + 2   'increment of 2 since each is a word
RETURN


freeze:
'//the following won't do anything but will slow the BStamp down,
'//which is good in that it balance out the executing time between
'//turn AND freeze (so that all servo would move in the same steps..
'[
IF (pointer2 < 90 ) THEN READ pointer2, position : ELSE READ pointer2, Word position
     ' 250 is a byte and therefore need to DATA and READ it differently
SEROUT PSC,N2400,["!SC",14, 1,position.LOWBYTE, position.HIGHBYTE, CR]
                                                    ']
   pointer = pointer + 1
   pointer2 = pointer2 + 2
RETURN
```

Fig. 3.2.3. Turn/Freeze Action Module for Servo

3.3 Detache Mode

In music, detache means that there is a break between every note within a phrase. In Detache mode, every note is detached. It can easily be accomplished by cutting off the air in the end of every note. In other words, for every note event, after the air valve opens for the length of durations, the air valve will then return to the original position to provide the break for the next note. The bottom portion of figure 3.2.2 shows the procedures of actions, for the servos and the stepper motors, when running in detache mode. The procedures can be simplified as follows (Fig. 3.3.1).

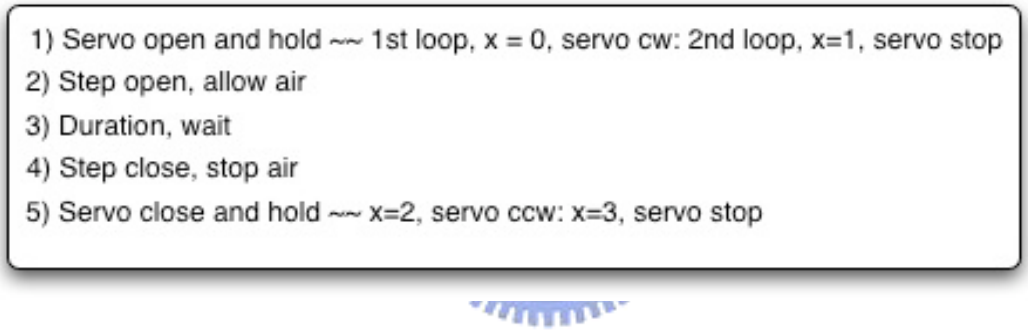


Fig. 3.3.1. Detache Mode Action Summary

The procedures in figure 3.3.1 represent a non-feedback loop system. It would be more efficient, for the servos, if it is a open loop system. The servos, which are already open and required for the next pitch, will remain in open position without having to close and open again. However, given the nature of the eeprom, it is difficult to accomplished in BS2. It would be difficult to locate the “P” value, position data, in the eeprom address. The probability of different pitches, which have overlapping keys must be studied and incorporated in the proper eeprom address.

The extent of this study, the feedback loop systems for servos, is too great for the current project.

3.4 Staccato Mode

In staccato, a form of detache, the duration of note is reduced into a very short value, while the duration-distance between notes remains intact. To make detache sound more staccato, in the program, instead of given a "time or duration" to the air valve, just provide a standard of 160 milliseconds to the stepper motor (or shorter). The distance between the current note and the next note stays the same while each note will sounds shorter.

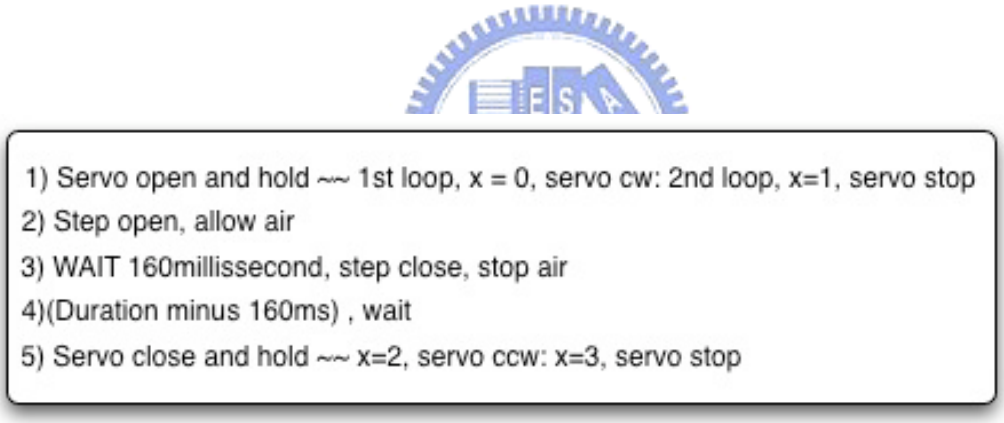


Fig. 3.4.1. Staccato Mode Action Summary

3.5 Legato Mode

Legato in music means the distance from one note to the next is well connected to the point that the break in between notes becomes non-perceivable. In legato mode, the servos will remain as a non-feedback loop system as that of the detache mode. The main difference lies in the control of the air valves. The principle

is simple in that the air valve in between notes will remain open until the phrase come to an end. The degree of openness of the air valve will change according to the dynamics of each note.

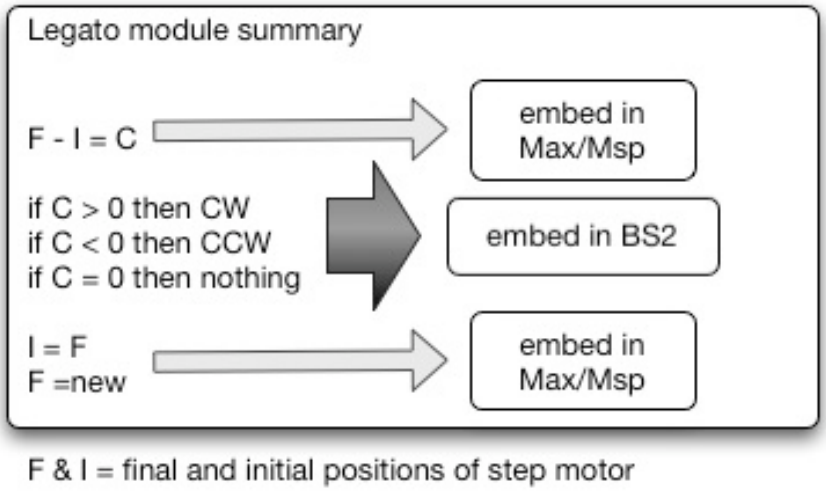


Fig. 3.5.1. Legato Module Summary

The main concept of legato mode is shown in figure 3.5.1. Final position minus initial position equals change. If change is greater than zero, the stepper motor will rotate clockwise, which will increase the air valve cavity and the dynamics. If change is less than zero, the stepper motor will rotate counter clockwise, which will decrease the air valve cavity and the dynamics. If change equals zero, nothing would happen. After the "if condition" is completed, the initial value will become the final value. Later, a new "P" value will be generated from the dynamics module and assign to the final (F). The process, loop, in figure 3.5.1 will occur for every note event.

For some reason, Basic Stamp wasn't able to receive negative values. Therefore, another variable, "posneg", is needed to indicate the positive or negative

status of C. Figure 3.5.2 shows all of the major subroutines for the legato mode. The subroutine “mouth”, will use the direction variable, rather than “C”, to decide the direction of the rotation.

```
mouth:
 '"change" will me made as an absolute value,
 'so use "posneg" as cue measure
IF posneg = 6 THEN stepopen
IF posneg = 5 THEN stepclose
'if posneg= 4 nothing ....
RETURN

stepopen:
LOW 2
  FOR counter = 1 TO change
    PULSOUT 1, 10
    NEXT

RETURN

stepclose:

LOW 6
  FOR counter = 1 TO change
    PULSOUT 5, 10
    NEXT
```

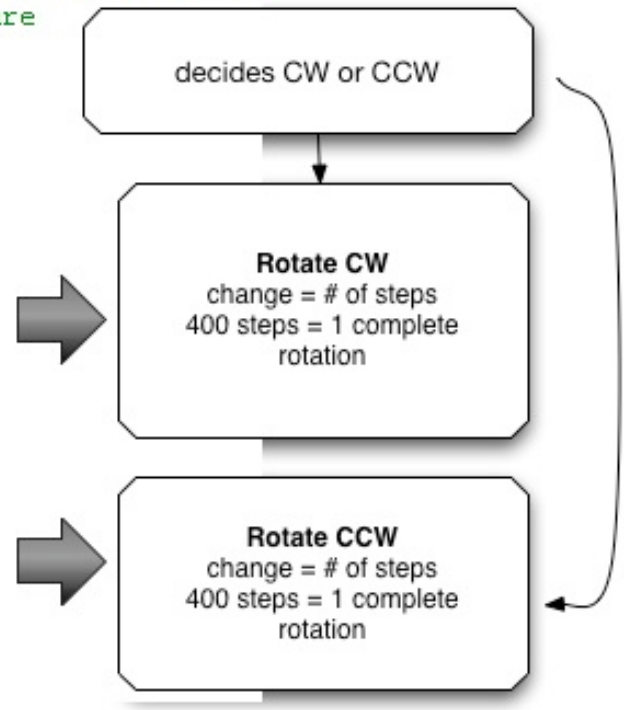


Fig. 3.5.2. CW versus CCW Decision Maker

3.6 Score Play

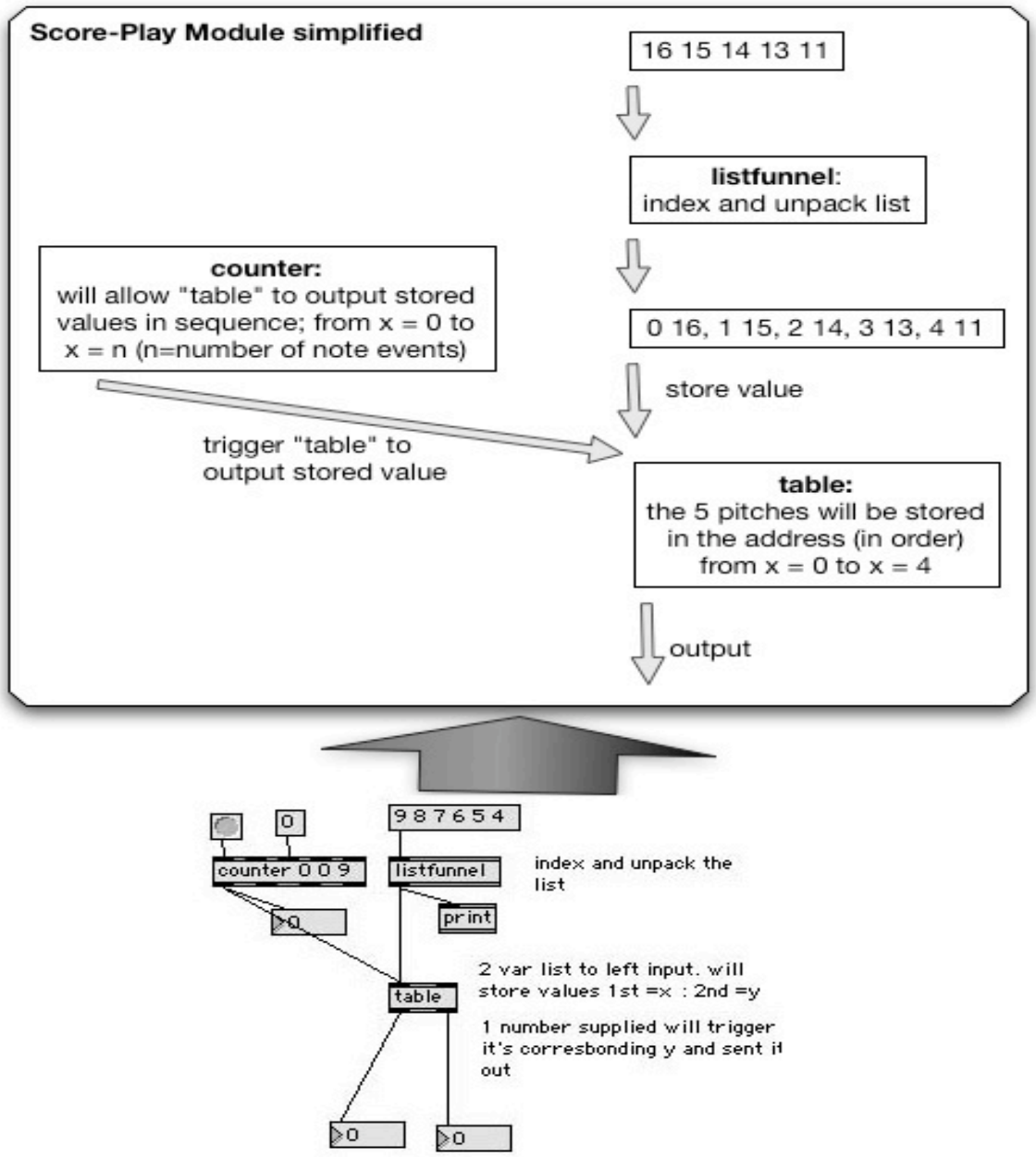


Fig. 3.6.1. Score-Play Module Principles

Music score can be keyed in as a number-list in the “message box” of Max/Msp. Music parameters such as, pitch, duration and dynamics can each be represented by a number-list and stored in the corresponding Max/Msp “table”. A

simplified version of the music score module is shown in figure 3.6.1. The object "listfunnel" will both index and unpack the list and then store them in the "table". When the "counter" objects, sends out an increment of numbers from 0 to 9, the first ten values store in the table will be sent out consecutively. Figure 3.6.2 shows an

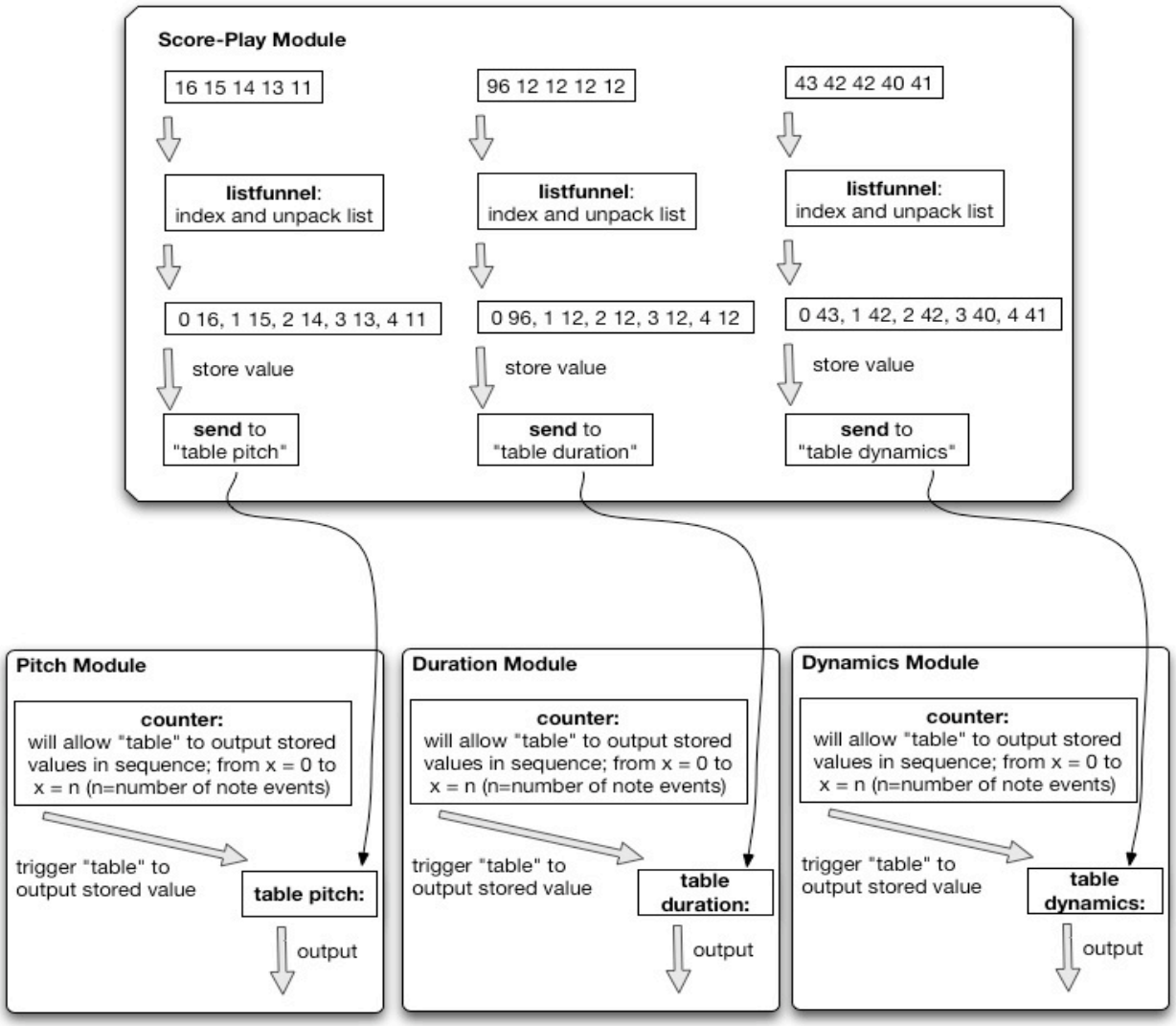


Fig. 3.6.2. Score-Play Module

excerpt of the actual sample of the Score-Play module. The Max/Msp object “send” will redirect the indexed and unpacked number-list to the appropriate pitch, duration, and dynamics module that existed outside of the current patch (Fig. 3.1.1).[10] Later the music scored tables can be played out like that of the probability tables.

[10] a patch in max/msp, is like a window or module that helps in packing all of the components of certain programs together. It organizes and categorizes different program components into a unit by itself.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Automatic Flute Performance: Case 1 Probabilities

	1	2	3	4	5	6	7	8	9
Duration:	96	12	12	96	12	12	96	12	12
Dynamics:	0	43	41	40	42	41	43	40	43
Pitch:	x	9	16	11	9	9	6	16	8

10	11	12	13	14	15	16	17	18	19
12	12	96	96	12	96	96	12	12	96
44	41	43	40	41	40	40	40	40	41
9	11	16	11	9	16	9	9	13	14

20	21	22	23	24	25	26	27		
12	96	96	96	96	96	12	12	96	
44	43	36	40	43	44	0	40	0	
6	6	14	9	14	9	x	13	x	

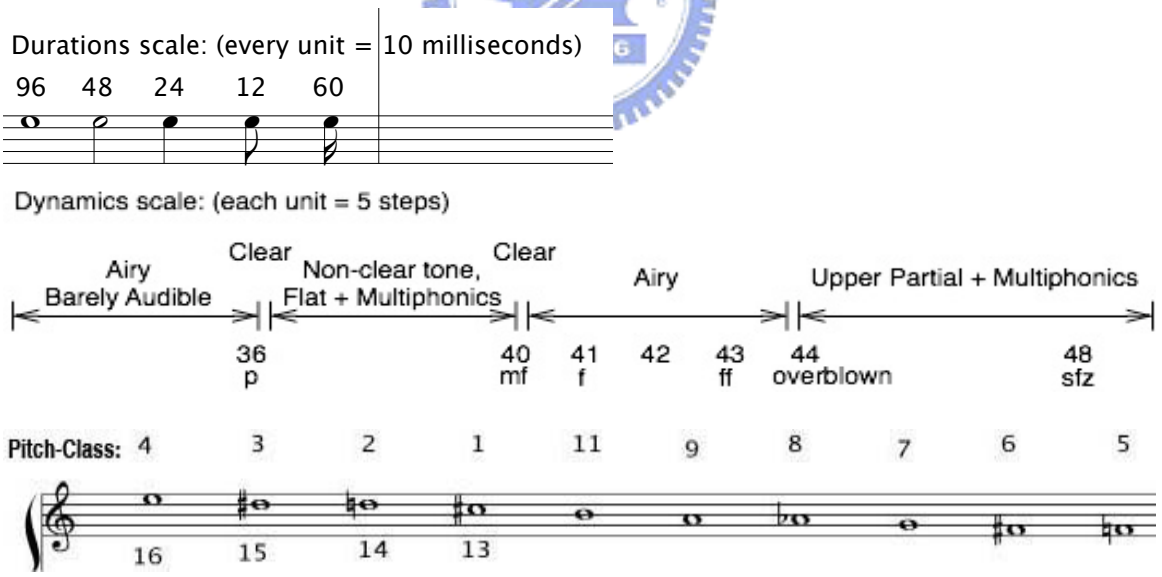


Table 4.1.1. Case 1 Probabilities: Data collected from Probability Module

The above data was collected from a performance under the short-long, not too loud, and A-major mode of the probability tables (Table 4.1.1). It is first

performed under the detache setting and the second in legato setting. The results will be studied and compared to the original data and midi soundfile. The music score representation of the performance data is as follows. The detache score, figure 4.1.2, is a dictation from the actual wav recording of the performance under detache mode.

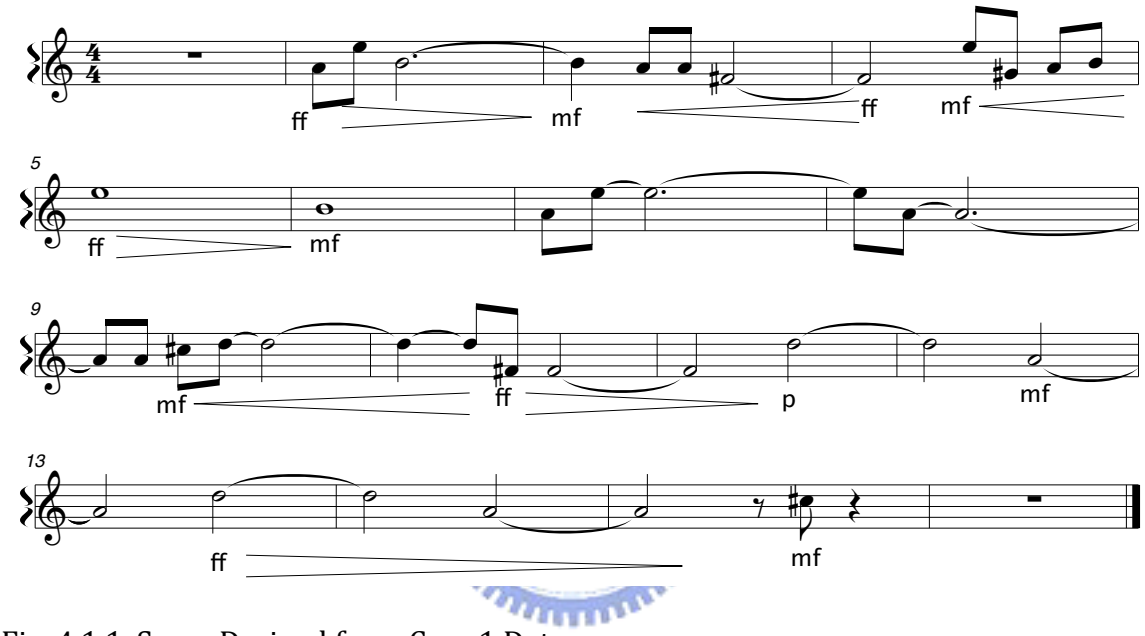


Fig. 4.1.1. Score Derived from Case 1 Data

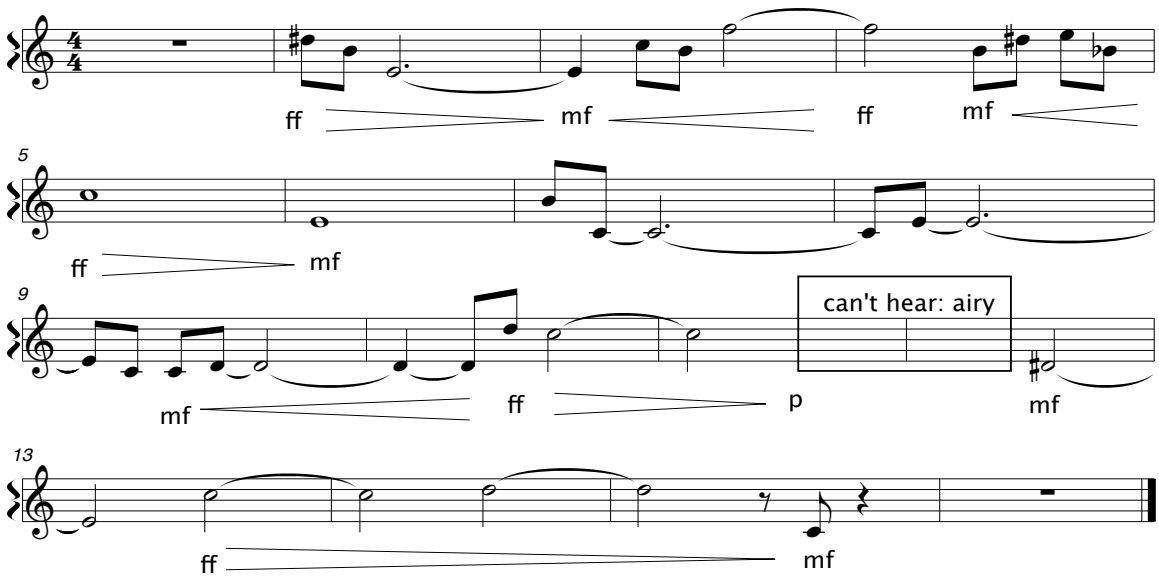


Fig. 4.1.2. Score Derived from Performance of Case 1: Detache

The duration of every note in figure 4.1.2 is accurate. The only major differences for time, is that there is a long silence separating every note (not shown in score). The top of figure 4.1.3 shows the waveform view of the midi sound file. Marker 1 and 2 separates the first two, 3-note motives, which finishes on the second beat of measure 4 (score Fig. 4.1.1). Visually, it is difficult to separate each note from the 3-note motive under waveform view. The middle of figure 4.1.3 represents the waveform view of the detache performance. In it, every note can clearly be isolated.

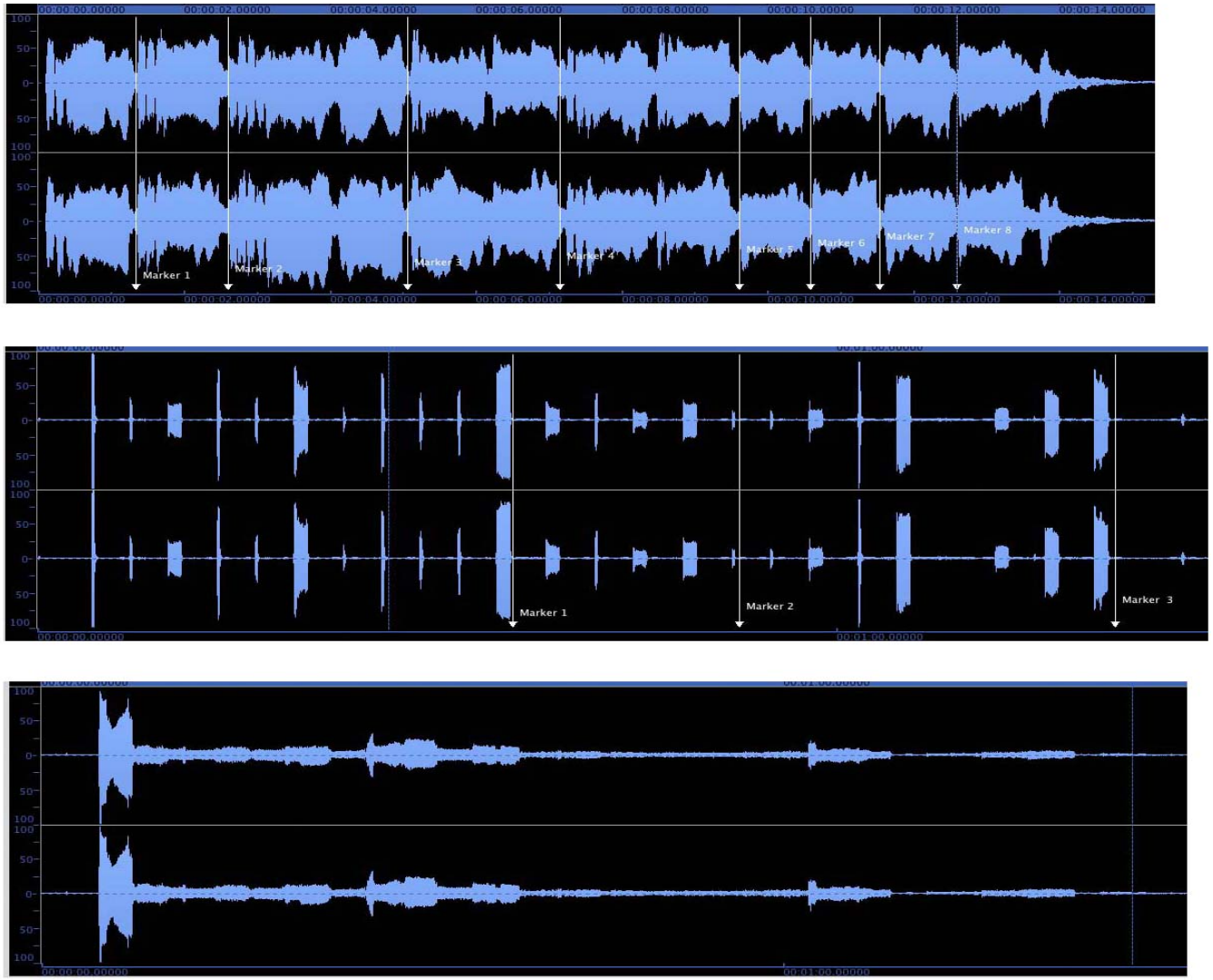


Fig. 4.1.3. Case 1 Performance Waveform View: Midi Flute, Detache, and Legato

Despite the fact that every note's duration is accurate, within acceptable uncertainties, the long silence that lies in between is too great that it throws the music off. The long silence can be explained in figure 4.1.4. Every servo action takes time, "Tn". Both directions of rotation take equal amounts of time. "Tm" is the pause time that is required for the BS2, Max/Msp, and all serial ports to be ready. Many attempts have being taken to shorten the amount of response time. However, the minimum response time is still between 2.2 to 2.5 seconds.

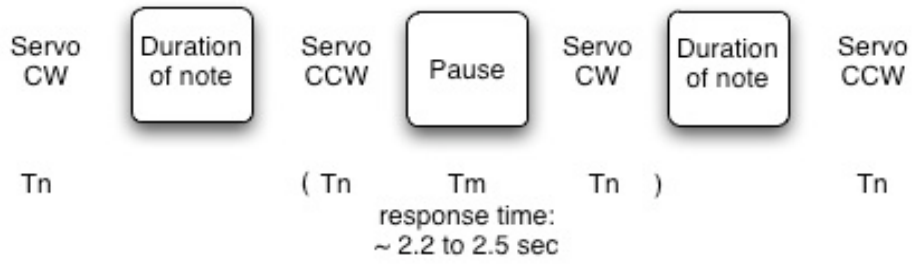


Fig. 4.1.4. System Processing Time Inspection

Upon first study, the detache performances and the original midi score, it is difficult to not notice the off pitches. However, figure 4.1.2 provides enough information in pointing out the major flaws of the flute system. First, pitches that are off within a whole tone can be overlooked, because of the intonation or the tuning of the flute during the performance.

There are two major issues that contribute to the off pitches. One is related to the keystroke mechanism and the other mouthpiece mechanism. Many times there will be one or two flute keys not pressed down completely. There will be a slight gap in between. Occasionally some of the pressed down keys do not exert

enough of pressure and air escapes. The wrong high pitches, measure 3 and 10, double forte, can be explained as overblown. When the air compressor tank is full, air gun is more tightly adjusted, and double forte is requested at the same time it is easy to overblown the flute. Sometimes it is overblown on the wrong notes.

The detache performance provides acceptable results. Despite wrong pitches and long silences in between notes, the duration of each note was close to accurate and that each output tone is clear and full. However it is not the case for the legato performance. Therefore a score representation of the legato performance will not be necessary. Imagine air being constantly supplied to a flute, while the keys are consecutively pressed and released. With occasional gaps between key-holes, and given a high response time, the messy sound that immerges and transforms without any breaks will become chaotic. When all keys are open, if the air continues, like that of the legato mode, a pitch, D4, will be produced instead of silence. In legato mode the air tank will ran out very quickly and as the air pressure of the tank decreases the air output per steps of stepping motor decreases. The air available will not be enough to support the proper pitch at any given time. The bottom of figure 4.1.3 shows the first few measure of the legato performance. The disaster is shown right after the first few notes. The open-hole tone that resulted during response time, overshadows and blurs the actual note. There is not a clear ADSR curve for every note and the air pressure is becoming too weak when required.

4.2 Automatic Flute Performance: Case 2 Score Play

The following melody is composed based on the possible good notes of the automatic flutes (Fig. 4.2.1). The main design purpose is for easy playabilities. The flute keys gradually increase from one to two and generally in the ascending order of the servo address numbers. Its table representation is shown in table 4.2.1.

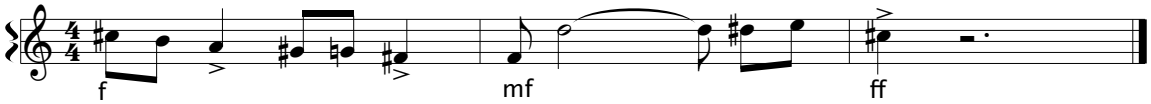


Fig. 4.2.1. Composed Melody

	1	2	3	4	5	6	7	8	9	10	11
Duration:	12	12	24	12	12	24	12	60	12	12	24
Dynamics:	41	41	42	41	41	42	40	40	41	41	42
Pitch:	13	11	9	8	7	6	5	14	15	16	13

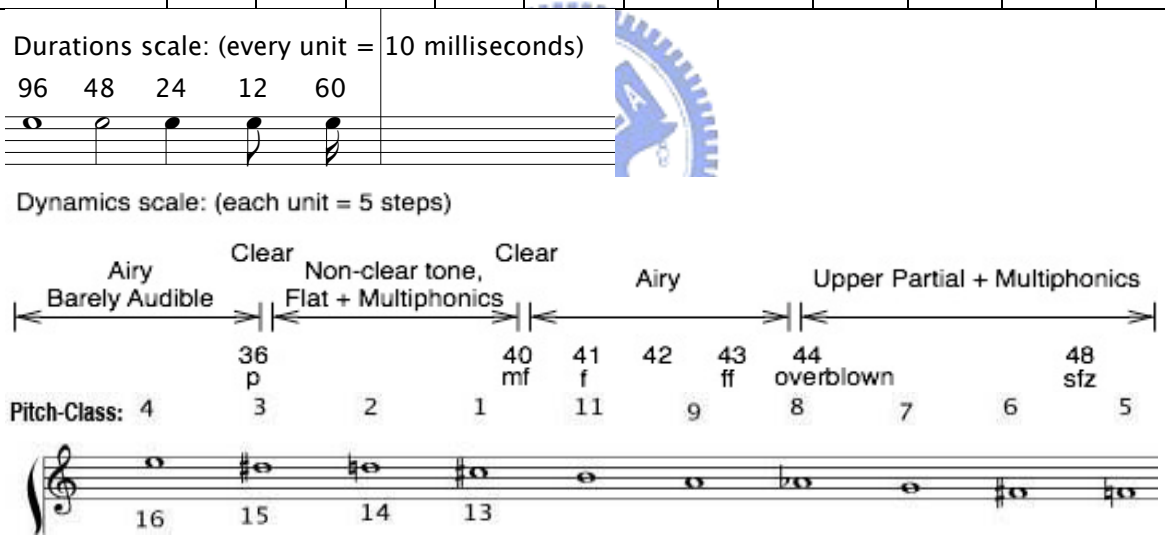


Table 4.2.1. Case 2 Score-Play: Data collected from Composed Melod

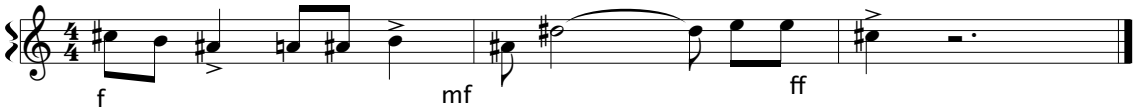


Fig. 4.2.2. Score Derived from Performance of Composed Melody: Detache

The data was entered and performed under the detache mode. The detache score, derived from the dictation of the performance is shown in figure 4.2.2. It will be compared and studied against the midi score.

The composed melody produce better results than the probability table modules. From figure 4.2.2, the major contours of the melody are intact. It gradually descends by step and jumps up in the second measure. The phrase finishes on the exact notes as that of the data. The slight chromaticism deviation around the third beat of the first measure and the second beat of second measure are the results of faulty servo-mechanism. Figure 4.2.3 shows the waveform view of the midi and detache performances. The three markers that are placed in the top of figure two, midi file, are placed before the accents. The pitches are “A”, “F#”, and “C#” (Fig. 4.2.1). This accent is also visible on the detache performance. The clear breaks which isolate the notes in the detache performance is the reflection of the “response time” of the automatic flute system.

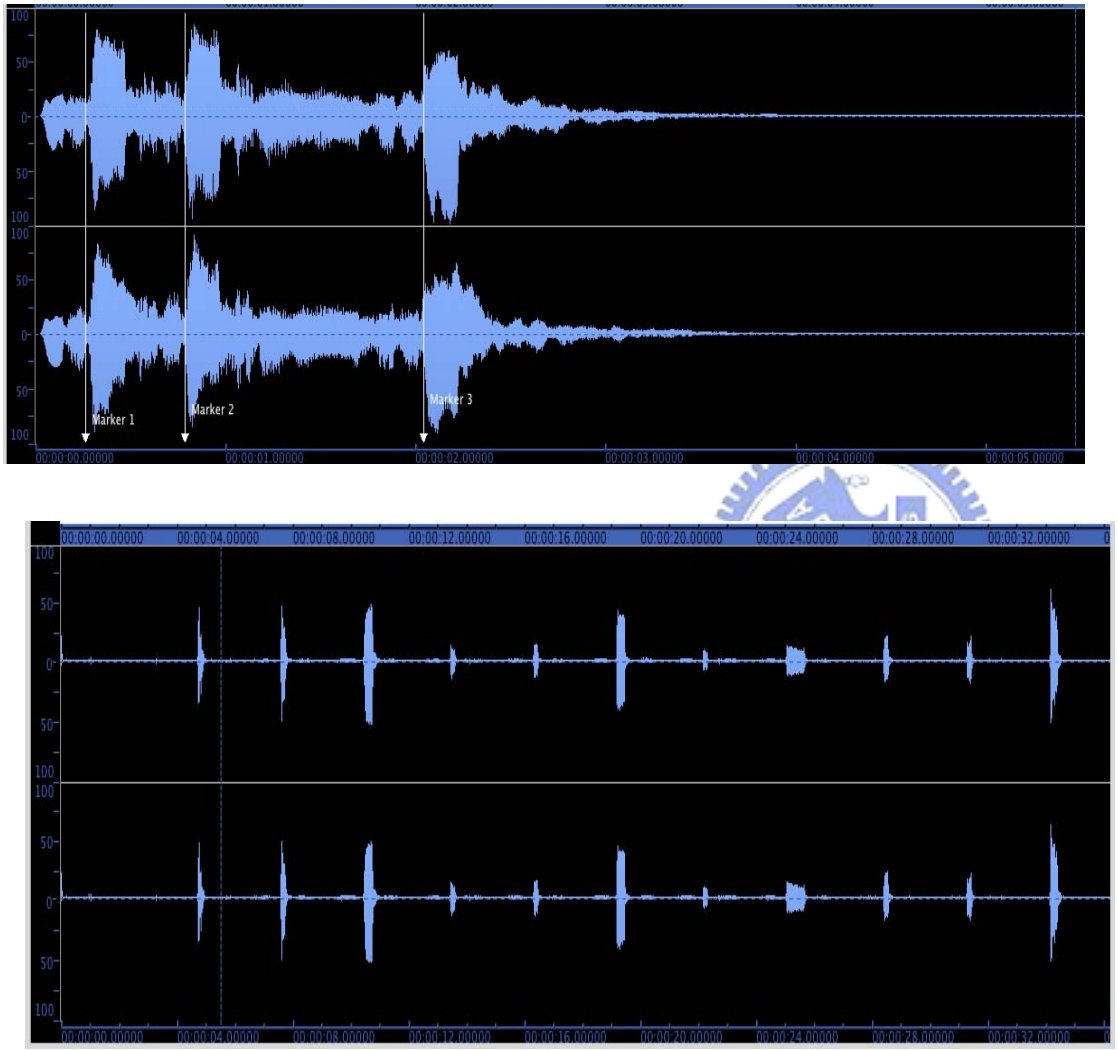


Fig. 4.2.3. Case 2 Score-Play Waveform View: Midi Flute, Detache

CHAPTER 5

CONCLUSIONS

Given the resource available, the Automatic Flute is considered as a successful prototype. The major issues are centered on the keystroke mechanism. The choice of servos and the PSC board is a mistake. Unlike the mouthpiece mechanism, the keystroke mechanism requires four lines of command, per servo, for open and close. Given the number of the servos that are required, minimum 9 for the current system, there are 36 position commands issued to the PSC board. The servos move sluggishly and the final positions are inaccurate. Whereas the mouthpiece mechanism, stepper motor and MD2415, it takes only two lines of command for open and close. The degree of steps is fast and accurate. The choice of stepper motor, brass plunger and other controller board would be more ideal for the keystroke mechanism. If the keystroke mechanism can be improved, the overall response time per note event would be drastically reduced. Figure 5.1 provides a list of the "more-urgent" improvements that is needed for the automatic flute prototype.

Keystroke Mechanism

1) brass plunger or stepping motor for flute keys

2) improved control with a feedback loop systems for the keystroke mechanism

Mouthpiece Mechanism

1) a device that could modify the shape of air that passes through

Fig. 5.1. Future Improvements

Besides the brass plunger that is already mentioned, having a feedback loop system for the keystroke mechanism will be ideal. The probability of different pitches, which have overlapping keys must be studied and incorporated in the feedback system. The call for improvement for the control system might mean choosing other controller boards besides BS2 and the use of C language programming over the less efficient Basic Stamp language. Unlike the keystroke mechanism, the current mouthpiece mechanism is more successful (Fig. 2.3.7 and 2.3.8). The air output is steady and the control of dynamics and articulations provides acceptable results. If one must enforce an improvement on the mouthpiece mechanism, it would be inventing a device that could modify the shape of air passing through. The current mechanism is capable of change the air output strength. However, it is not capable of changing the shape of the air output. Changing the shape of air output in real-time will allow different articulations, such as flutter tongue, slap tongue, and growl tone.

Despite all of the issues, it is fun to control the prototype manually. Real-time control of the mouthpiece airflow within Max/Msp and manual control of the flute-keys by hand will create interesting results. If a sensor is adapted to BS2, such as any pressure sensor or a popular USB dance pad, the airflow could then be controlled by a foot and leaving hands free for the keys, vibratos and other musical gestures. For aesthetic purposes, the flaws of the prototype could be overlooked, adapted and incorporated in itself as a form of expression for art performances. The robotic sound produced by the servo gear wheels, the high pitch frequencies of the charged currents within the stepper motor and the deep pulsic drone of the air compressor while filling the tank are all too romantic when combined with the flute.

REFERENCES

Adler, Samuel. *The Study of Orchestration.* New York: W. W. Norton & Company, Inc., 2002.

Allen, Thomas Tracy. "BASIC Stamp Support Index." 1 August 2008. <http://www.emesystems.com/BS2index.htm> (12 January 2009).

Blatter, Alfred. *Instrumentation and Orchestration.* Belmont: Schirmer, Thomson Learning, Inc., 1997.

Benade, Arthur H. *Horns, Strings, and Harmony.* New York: Dover Publications, Inc., 1992.

Carter, Elliott. *Flawed Words and Stubborn Sounds: a Conversation with Elliott Carter*. interview by Allen Edwards. New York: W. W. Norton and Company Inc., 1971.

_____. *Collected Essays and Lectures, 1937- 1995. ed.* Jonathan W. Bernard. Rochester: University of Rochester Press, 1997.

Cassidy, Christopher. *BASIC STAMP II – HOW II.* 5 December 2005. <http://www.maelabs.ucsd.edu/mae_ds/stamp/how/index.html> (20 February 2009).

Cope, David. *Computer Models of Musical Creativity.* Massachusetts: The MIT Press, 2005.

Dirks, Alex. *CrustCrawler Robotics.* 1 May 2007. <http://www.crustcrawler.com/> (10 January 2009).

Huang, Yu-Wei. "Improvements of Automatic Guitar Mechanism Design and a Practical Music Input System." Master Thesis of the Electrical Engineering Department, Southern Taiwan University, 1996.

Jones, Douglas W. *Control of Stepping Motors.* 14 February 2008. <http:// www.cs.uiowa.edu/~jones/step/> (10 December 2008).

Kuhnel, Claus, and Klaus Zahnert. *BASIC Stamp 2p: Commands, Features, and Projects.* Rocklin: Parallax, Inc. Press, 2003.

Norman, Katharine. *Sounding Art: Eight Literary Excursions through Electronic Music.* Burlington: Ashgate Publishing Company, 2004.

Parallax Team. *BASIC Stamp Syntax and Reference Manual version 2.2.* Rocklin: Parallax Inc., 2005.

Rowe, Robert. *Interactive Music Systems: Machine Listening and Composing.* Massachusetts: The MIT Press, 1993.

_____. *Machine Musicianship.* Massachusetts: The MIT Press, 2001.

Tingley, George Peter. "Metric Modulation and Elliott Carter's First String Quartet." Indiana Theory Review, vol. 4, issue 3 (spring 1981): 3-11.

Williams, Al. *Microcontroller Projects Using the Basic Stamp.* Berkeley: Publishers Group West, 2002

Winkler, Todd. *Composing Interactive Music: Techniques and Ideas Using Max.* Massachusetts: The MIT Press, 1998.

Winsor, Phil. *Automated Music Composition.* Hsin Chu: University of North Texas Press, 1989.

Wolfe, Joe. *Robot Clarinet.* 1 January 2009. <http://www.phys.unsw.edu.au/jw/clarinetrobot.html> (19 May 2009).

APPENDIX

I. B.Stamp code: Detache

```
' {$STAMP BS2}
' {$PBASIC 2.5}
'//////////////////
'DETACHE
'//////////////////
'-------------------- [Programming Notes] ----------------------------------------------
'-------------------------[ I/O Definitions ]-----------------------------------------------
        ' PSC module
PSC          PIN    15
#SELECT $STAMP
 #CASE BS2SX, BS2P
  N2400      CON    1021+$8000
 #CASE BS2PX
  N2400      CON    1646+$8000
 #CASE #ELSE
  N2400      CON    396+$8000
#ENDSELECT


'-------------------------------------------------------------------------------------------
'variable for keys
aa VAR Nib ' value btw 0-15
bb VAR Nib
cc VAR Nib
dd VAR Nib
ee VAR Nib
ff VAR Nib
gg VAR Nib
hh VAR Nib
ii VAR Nib
jj VAR Nib
kk VAR Nib
ll VAR Nib
mm VAR Nib


servoaddr VAR Nib ' value btw 0-15
x VAR Nib
position      VAR    Word
time VAR Word
```

```
pointer VAR Byte
pointer2 VAR Byte

'variable for mouthpiece
counter VAR Word
change VAR Word

'@0, @50, @74, @98, @122
'april/10 these newly calibrated data taken without (servo 6, 10, 11, 12 in loop),
under the condition that
'not all the remaining servo will move simlutaneously (testing with different pitches
everytime), and
'adding redundant PWM SEROUT in subroutin "freeze" to balance the executing
time difference between turn and freeze
'subroutine...

DATA  0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14

' put bunch of 250 ... byte size not word size.... pointer doesn't need to be specific....
since
'every CW OR CCW pointer2 is reassigned
DATA @60, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,250, 250, 250,
250, 250, 250, 250, 250, 250, 250,  250, 250, 250, 250, 250, 250
'
DATA @100, Word 750, Word 750, Word 750, Word 750, Word 750, Word
750,Word 750, Word 750, Word 750, Word 750, Word 750, Word 750
'        100       102       104       106    108     110      112       114     116     118
120     122
ccw DATA @140, Word 925, Word 953, Word 987, Word 945, Word 908, Word 946,
Word 975, Word 963, Word 1000, Word 969, Word 933
     'servo#   0        1        2      3        4        5        7          8         9

DATA @180, Word 750, Word 750, Word 750, Word 750, Word 750, Word 750,
Word 750, Word 750, Word 750, Word 750, Word 750, Word 750


DO    'big serial loop

SERIN 16, 16780, [WAIT ("time"), DEC time, WAIT ("aa"),DEC aa, WAIT ("bb"),
DEC bb, WAIT ("cc"),DEC cc, WAIT ("dd"), DEC dd, WAIT ("ee"),DEC ee,
WAIT ("ff"), DEC ff, WAIT ("gg"),DEC gg, WAIT ("hh"), DEC hh, WAIT ("ii"),
DEC ii, WAIT ("jj"), DEC jj, WAIT ("kk"),DEC kk, WAIT ("ll"), DEC ll,
WAIT ("mm"), DEC mm, WAIT ("stp"), DEC change]
```

```
'//////////////////////////////
 pointer2 = 60

FOR x = 1 TO 4

 pointer = 0                          ' MOVE or NOT, pointer will be in place ( or
follow)...
IF aa = 0 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF bb = 1 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF cc = 2 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF dd = 3 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ee = 4 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ff = 5 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF gg = 6 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF hh = 7 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ii = 8 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF jj = 9 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF kk = 10 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF ll = 11 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF mm = 12 THEN GOSUB turnbaby : ELSE GOSUB freeze


  IF x = 1 THEN PAUSE 100
  IF x = 1 THEN pointer2 = 100
  IF x = 2 THEN GOSUB stepopen
  IF x = 2 THEN PAUSE time
  IF x = 2 THEN GOSUB stepclose
  IF x = 2 THEN pointer2 = 140
  IF x = 3 THEN PAUSE 100
  IF x = 3 THEN pointer2 = 180
'  IF x = 4 THEN GOSUB stepclose
NEXT
LOOP
'//////////////////////
'mouthpiece valve open, hold(time), close
stepopen:
LOW 2
  FOR counter = 1 TO change
    PULSOUT 1, 10
    NEXT

RETURN
```

```
stepclose:

LOW 6
  FOR counter = 1 TO change
    PULSOUT 5, 10
    NEXT
'//////////////////////////////////////////////

turnbaby:
   '5 pointers servo need to be ran
READ pointer, servoaddr
IF (pointer2 < 90 ) THEN READ pointer2, position : ELSE READ pointer2, Word
position
' 250 is a byte and therefore need to DATA and READ it differently

SEROUT PSC,N2400,["!SC",servoaddr, 1,position.LOWBYTE, position.HIGHBYTE, CR]
    pointer = pointer + 1
    pointer2 = pointer2 + 2   'increment of 2 since each is a word
RETURN


freeze:
'//the following won't do anything but will slow the BStamp down,
'//which is good in that it balance out the executing time between
'//turn AND freeze (so that all servo would move in the same steps..
'[
IF (pointer2 < 90 ) THEN READ pointer2, position : ELSE READ pointer2, Word
position
    ' 250 is a byte and therefore need to DATA and READ it differently
SEROUT PSC,N2400,["!SC",14, 1,position.LOWBYTE, position.HIGHBYTE, CR]
                                    ']
  pointer = pointer + 1
  pointer2 = pointer2 + 2
RETURN
```

II. B.Stamp code: Staccato

```
' {$STAMP BS2}
' {$PBASIC 2.5}
'//////////////////
'STACCATO
'//////////////////
'-------------------- [Programming Notes] ----------------------------------------------
'-------------------------[ I/O Definitions ]-----------------------------------------------
        ' PSC module
PSC          PIN    15
#SELECT $STAMP
 #CASE BS2SX, BS2P
  N2400      CON    1021+$8000
 #CASE BS2PX
  N2400      CON    1646+$8000
 #CASE #ELSE
  N2400      CON    396+$8000
#ENDSELECT

'-------------------------------------------------------------------------------------------
'variable for keys
aa VAR Nib ' value btw 0-15
bb VAR Nib
cc VAR Nib
dd VAR Nib
ee VAR Nib
ff VAR Nib
gg VAR Nib
hh VAR Nib
ii VAR Nib
jj VAR Nib
kk VAR Nib
ll VAR Nib
mm VAR Nib


servoaddr VAR Nib ' value btw 0-15
x VAR Nib
position      VAR    Word
time VAR Word

pointer VAR Byte
pointer2 VAR Byte

'variable for mouthpiece
```

```
counter VAR Word
change VAR Word

'@0, @50, @74, @98, @122
'april/10 these newly calibrated data taken without (servo 6, 10, 11, 12 in loop),
under the condition that
'not all the remaining servo will move simlutaneously (testing with different pitches
everytime), and
'adding redundant PWM SEROUT in subroutin "freeze" to balance the executing
time difference between turn and freeze
'subroutine...

DATA  0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14

' put bunch of 250 ... byte size not word size.... pointer doesn't need to be specific....
since
'every CW OR CCW pointer2 is reassigned
DATA @60, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,250, 250, 250,
250, 250, 250, 250, 250, 250, 250,  250, 250, 250, 250, 250, 250
'
DATA @100, Word 750, Word 750, Word 750, Word 750, Word 750, Word
750,Word 750, Word 750, Word 750, Word 750, Word 750, Word 750
'       100     102     104     106   108    110     112     114    116    118
120    122
ccw DATA @140, Word 925, Word 953, Word 987, Word 945, Word 908, Word 946,
Word 975, Word 963, Word 1000, Word 969, Word 933
    'servo#   0       1       2     3       4       5       7       8       9

DATA @180, Word 750, Word 750, Word 750, Word 750, Word 750, Word 750,
Word 750, Word 750, Word 750, Word 750, Word 750, Word 750


DO    'big serial loop

SERIN 16, 16780, [WAIT ("time"), DEC time, WAIT ("aa"),DEC aa, WAIT ("bb"),
DEC bb, WAIT ("cc"),DEC cc, WAIT ("dd"), DEC dd, WAIT ("ee"),DEC ee,
WAIT ("ff"), DEC ff, WAIT ("gg"),DEC gg, WAIT ("hh"), DEC hh, WAIT ("ii"),
DEC ii, WAIT ("jj"), DEC jj, WAIT ("kk"),DEC kk, WAIT ("ll"), DEC ll,
WAIT ("mm"), DEC mm, WAIT ("stp"), DEC change]


'////////////////////////////////
 pointer2 = 60

FOR x = 1 TO 4
```

```
 pointer = 0                              ' MOVE or NOT, pointer will be in place ( or
follow)...
IF aa = 0 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF bb = 1 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF cc = 2 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF dd = 3 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ee = 4 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ff = 5 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF gg = 6 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF hh = 7 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ii = 8 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF jj = 9 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF kk = 10 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF ll = 11 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF mm = 12 THEN GOSUB turnbaby : ELSE GOSUB freeze


  IF x = 1 THEN PAUSE 100
  IF x = 1 THEN pointer2 = 100
  IF x = 2 THEN GOSUB stepopen
  IF x = 2 THEN PAUSE 160      'Step will only open for 160 milliseconds
  IF x = 2 THEN GOSUB stepclose
  IF x = 2 THEN pointer2 = 140
  IF x = 3 THEN PAUSE 100
  IF x = 3 THEN pointer2 = 180
'  IF x = 4 THEN GOSUB stepclose
NEXT
LOOP
'//////////////////
'/////////////////////////
'mouthpiece valve open, hold(time), close
stepopen:
LOW 2
  FOR counter = 1 TO change
    PULSOUT 1, 10
    NEXT

RETURN

stepclose:

LOW 6
  FOR counter = 1 TO change
    PULSOUT 5, 10
    NEXT
'/////////////////////////////////////////////
```

```
turnbaby:
   '5 pointers servo need to be ran
READ pointer, servoaddr
IF (pointer2 < 90 ) THEN READ pointer2, position : ELSE READ pointer2, Word
position
' 250 is a byte and therefore need to DATA and READ it differently

SEROUT PSC,N2400,["!SC",servoaddr, 1,position.LOWBYTE, position.HIGHBYTE, CR]
    pointer = pointer + 1
    pointer2 = pointer2 + 2   'increment of 2 since each is a word
RETURN


freeze:
'//the following won't do anything but will slow the BStamp down,
'//which is good in that it balance out the executing time between
'//turn AND freeze (so that all servo would move in the same steps..
'[
IF (pointer2 < 90 ) THEN READ pointer2, position : ELSE READ pointer2, Word
position
    ' 250 is a byte and therefore need to DATA and READ it differently
SEROUT PSC,N2400,["!SC",14, 1,position.LOWBYTE, position.HIGHBYTE, CR]
                                  ']
  pointer = pointer + 1
  pointer2 = pointer2 + 2
RETURN
```

III. B.Stamp code: Legato

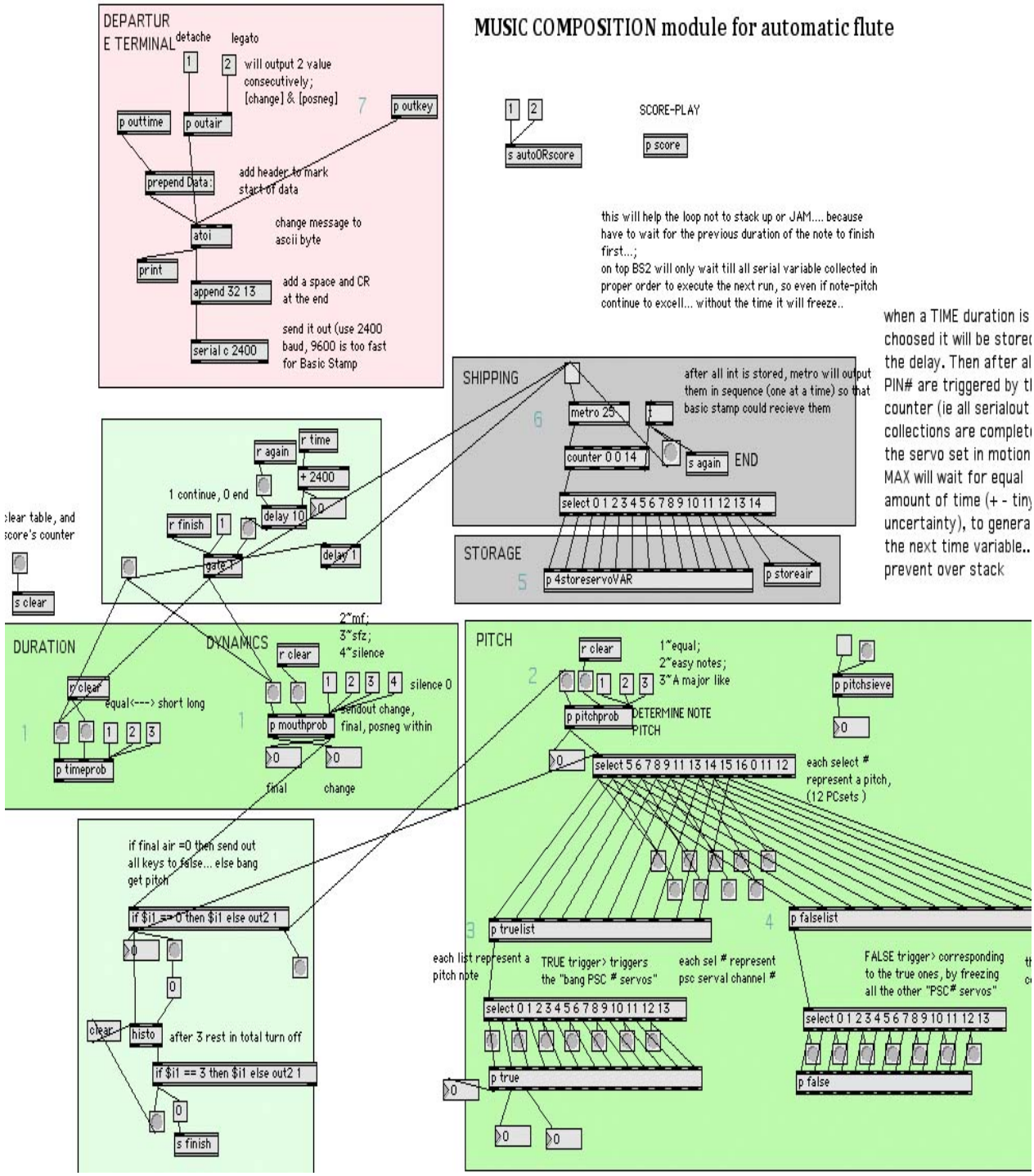
```
' {$STAMP BS2}
' {$PBASIC 2.5}
'//////////////////////
'LEGATO
'//////////////////////
'-------------------- [Programming Notes] ----------------------------------------------
'-------------------------[ I/O Definitions ]-----------------------------------------------
' PSC module
PSC          PIN     15
#SELECT $STAMP
  #CASE BS2SX, BS2P
    N2400       CON     1021+$8000
  #CASE BS2PX
    N2400       CON     1646+$8000
  #CASE #ELSE
    N2400       CON     396+$8000
#ENDSELECT
'-------------------------------------------------------------------------------------------
'variable for keys
aa VAR Nib ' value btw 0-15
bb VAR Nib
cc VAR Nib
dd VAR Nib
ee VAR Nib
ff VAR Nib
gg VAR Nib
hh VAR Nib
ii VAR Nib
jj VAR Nib
kk VAR Nib
ll VAR Nib
mm VAR Nib

servoaddr VAR Nib ' value btw 0-15
x VAR Nib
position       VAR     Word
time VAR Word

pointer VAR Byte
pointer2 VAR Byte

'variable for mouthpiece
counter VAR Word
change VAR Word
```

```
posneg VAR Nib ' BStamp serin won't recognize negative variables taken from
max/msp, therefore need one more variable.
          'seems that BStamp will takein the negative number and makes it absolute...

'@0, @50, @74, @98, @122

DATA  0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14

' put bunch of 250 ... byte size not word size.... pointer doesn't need to be specific....
since
'every CW OR CCW pointer2 is reassigned
DATA @60, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,
250, 250, 250, 250, 250, 250, 250, 250,  250, 250, 250, 250, 250, 250
'         50       52       54      56       58       60       62      64       66        68      70       72
DATA @100, Word 750, Word 750, Word 750, Word 750, Word 750, Word 750,
Word 750, Word 750, Word 750, Word 750, Word 750, Word 750
'         74       76       78      80       82       84       86      88       90        92      94       96
ccw DATA @140, Word 970, Word 975, Word 992, Word 976, Word 955, Word 967,
Word 1010, Word 1024, Word 1007, Word 1139, Word 1010
         '  0       1       2     3       4       5       7         8        9
         '  98      100      102     104      106     108      110     112      114       116
118      120
DATA @180, Word 750, Word 750, Word 750, Word 750, Word 750, Word 750,
Word 750, Word 750, Word 750, Word 750, Word 750, Word 750


DO    'big serial loop

SERIN 16, 16780, [WAIT ("time"), DEC time, WAIT ("aa"),DEC aa, WAIT ("bb"), DEC
bb, WAIT ("cc"),DEC cc,
WAIT ("dd"), DEC dd, WAIT ("ee"),DEC ee, WAIT ("ff"), DEC ff, WAIT ("gg"),DEC gg,
WAIT ("hh"), DEC hh,
WAIT ("ii"),DEC ii, WAIT ("jj"), DEC jj, WAIT ("kk"),DEC kk, WAIT ("ll"), DEC ll, WAIT
("mm"), DEC mm,
WAIT ("stp"), DEC change, WAIT ("pn"), DEC posneg]

'////////////////////////////////
 pointer2 = 60

FOR x = 1 TO 4

 pointer = 0                                ' MOVE or NOT, pointer will be in place ( or
follow)...
IF aa = 0 THEN GOSUB turnbaby : ELSE GOSUB freeze      'if freeze, then pointer
move + 1, pointer2 move + 2
IF bb = 1 THEN GOSUB turnbaby : ELSE GOSUB freeze      ' but no serial-out will be
```

```
sent
IF cc = 2 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF dd = 3 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ee = 4 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ff = 5 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF gg = 6 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF hh = 7 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF ii = 8 THEN GOSUB turnbaby : ELSE GOSUB freeze
IF jj = 9 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF kk = 10 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF ll = 11 THEN GOSUB turnbaby : ELSE GOSUB freeze
'IF mm = 12 THEN GOSUB turnbaby : ELSE GOSUB freeze

' original pause 75
  IF x = 1 THEN PAUSE 100
  IF x = 1 THEN pointer2 = 100

  IF x = 2 THEN GOSUB mouth   'when all servo opens and hold then stepper will
move

  IF x = 2 THEN PAUSE time
  IF x = 2 THEN pointer2 = 140
  IF x = 3 THEN PAUSE 100
  IF x = 3 THEN pointer2 = 180

NEXT
LOOP
'//////////////////////////
'mouthpiece valve open, hold(time), close

mouth:
 '"change" will me made as an absolute value,
 'so use "posneg" as cue measure
IF posneg = 6 THEN stepopen
IF posneg = 5 THEN stepclose
'if posneg= 4 nothing ....
RETURN

stepopen:
LOW 2
  FOR counter = 1 TO change
    PULSOUT 1, 10
    NEXT

RETURN
```

```
stepclose:

LOW 6
 FOR counter = 1 TO change
  PULSOUT 5, 10
  NEXT

'//////////////////////////////

turnbaby:
  '5 pointers servo need to be ran
  READ pointer, servoaddr
  IF (pointer2 < 90 ) THEN READ pointer2, position : ELSE READ pointer2, Word
position
   ' 250 is a byte and therefore need to DATA and READ it differently

 SEROUT PSC,N2400,["!SC",servoaddr, 1,position.LOWBYTE, position.HIGHBYTE,
CR]
  pointer = pointer + 1
  pointer2 = pointer2 + 2   'increment of 2 since each is a word
RETURN

freeze:
'//             the following won't do anything but will slow the BStamp down, which is
good in
'//that it balance out the executing time between turn and freeze (so that all servo
would move in the same steps..

IF (pointer2 < 90 ) THEN READ pointer2, position : ELSE READ pointer2, Word
position
   ' 250 is a byte and therefore need to DATA and READ it differently

 SEROUT PSC,N2400,["!SC",14, 1,position.LOWBYTE, position.HIGHBYTE, CR]
'//
  pointer = pointer + 1
  pointer2 = pointer2 + 2
RETURN
```

IV. Max/Msp patch1: include Detache, Staccato, Legato, and Score-Play



V. Max/Msp patch 2: Cellular Automata