AUTOMATIC NOTATION SYSTEM WITH CAPACITIVE TOUCH PIANO

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Wendian Jiang (wj225)

MEng Field Advisor: Bruce Land

Degree Date: May, 2016

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Automatic notation system with capacitive touch piano

Author: Wendian Jiang

Abstract: This project aims to develop an automatic notation system with a touch piano based on PIC32 microcontroller. The main function of this system is to record the notes people play on the piano keyboard and displaying them on a staff on TFT screen under the rules of musical notation. After the notation process is done, people can replay the music they record or start a new one by selecting the options provided on the user interface. This system is not only an efficient tool for people who work on music composition, but also provides a convenient way for beginners to learn about music notation.

Executive summary

Musical notation is always tedious in composition process. For music fans who lack of the related knowledge, they cannot record their music inspiration in a scientific way. Therefore, an automatic notation system is not only an efficient tool for people who works on music composition, but also an accessible way for beginners to learn about musical notation.

Based on this idea, an automatic notation system with a touch piano is built in this projection based on PIC32 microcontroller. The piano keyboard contains 37 keys which are made of capacitive sensing buttons and the sound outputs through a speaker using Karplus-Strong algorithm. To display the result of automatic notation, a 3.5 inch TFT touchscreen is connect with PIC32 through SPI. The touch function is also enabled to support the operations from users.

The automatic notation system support up to eighth notes. 8 bars are available on the screen which is suitable for recording simple songs. A complete musical symbol library is the foundation of display. All symbols are drawn with functions given by the TFT with different methods. The drawing of each symbol is packed in to new functions which are available in the musical symbol library.

The notation process is separated into two steps. In the first step, “raw notes” are shown on the staff in real time only with the information of pitches. The second step starts when each bar’s done. All notes within the finished bar are rearranged according to their values and replace the “raw notes”. In this step, the first note and all eighth notes should be specially deal with according the notation rules. A replay function is also designed for playing the whole song back.

A user interface is build based on the touch function of the TFT screen. Mode selection and other basic functions are implemented using a Mealy FSM control unit. Users can choose the mode, restart a new notation, play the whole song back or quit the current mode they are in by touching the related place on the screen or the capacitive sensing buttons on the keyboard.

Various tests are designed to verify the functionality of the automatic notation system. To test the notation process, multiple input patterns are applied to make sure whether the notation is correct under different situation. Various combination of the operations are also used for testing the transform logic and output logic of the FSM control unit. The whole system has already passed all test which indicate its robustness and correctness of the functionality.

Table of Contents

1. Introduction

Musical Notation is an indispensable process in composition. Plenty of time and work should be spent on it when people create their own music pieces. To free composers from this tedious work, many musical notation software are developed on PC recently. Most of them provide a completed music symbol library and staffs to work on. People can insert or modify their pieces on computer instead of writing it on paper. However, these software are not suitable for people who are not familiar with music notation. For those who are interested in composition but know little about notation, it is extremely difficult to write their inspiration down in a scientific way, which means beginner can hardly enjoy the music creating process unless they spend time on learning notation before.

Based on this consideration, this project is focused on creating an automatic notation system with a keyboard based on PIC32 microcontroller. There are mainly two parts in this project. The first part is a self-made keyboard with 37 keys which act as the music input. The keys are built with capacitive sensing buttons. People do not need to "press" the key but just touch the key to generate the sound.

The second part is a TFT screen with the touch function that used for notation display. People can choose the mode or complete the configuration by touching the screen. Two modes are designed in this notation system. The first one is the free-play mode. In this mode, the notation people is disabled. People can play the piano board freely with all keys enabled. In the second mode, the notation mode, an empty staff is shown on the screen and waiting for the start signal. After it's done, people can select whether restarting a new one or replaying what they record. Also, a "quit" button is available on the screen to quit the mode.

2. Previous work

The Piano keyboard was already finished before the building of the notation system. It is a course project finished on ECE4760 which also used as the foundation of the musical notation system [1]. The touch piano is shown in Figure 2.1. It contains totally 37 keys (3 octaves) which is capable to play many music pieces with both hands. Copper tape and aluminum foil are used for building the white and black keys on a wooden board. Behind the board, a solder board with electronic components on is fasten by black tape. Since 37 keys are supposed to be maintained at the same time. 5 multiplexers are used as the pin extension. PIC32 microcontroller takes in charge of scanning all keys repeatedly and output the sound according to which key is touched.

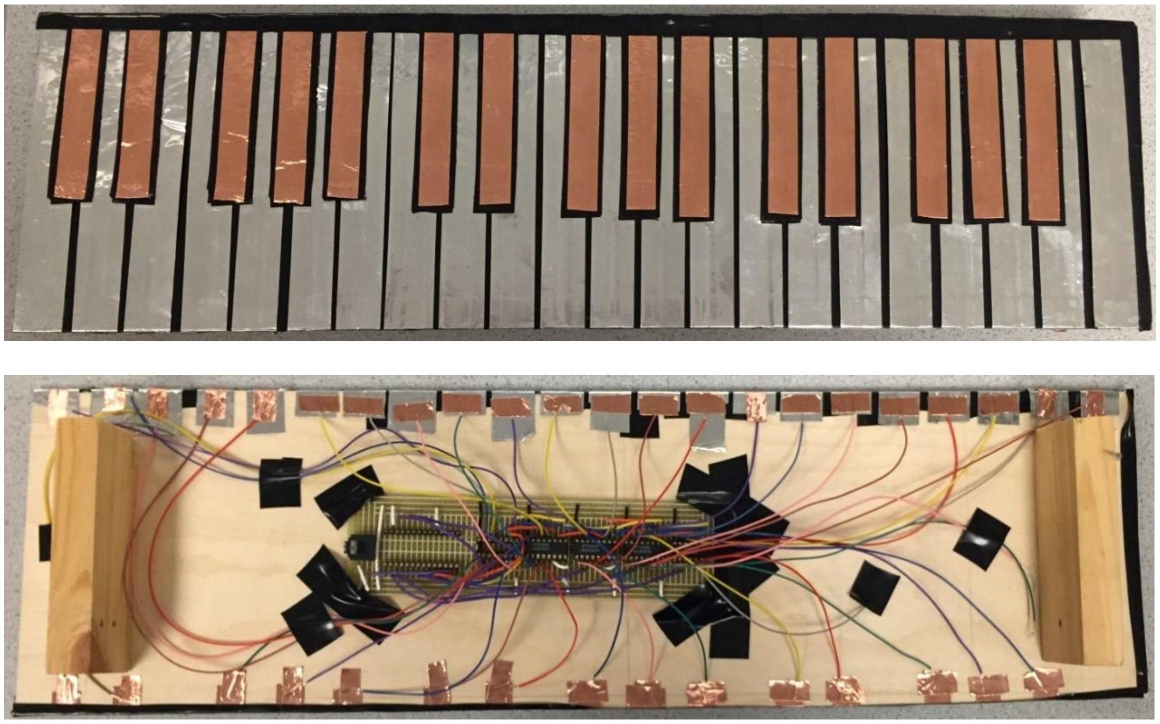


Figure 2.1 Capacitive Sensing Touch Piano

The capacitive sensing keys are built with CTMU and ADC inside the microcontroller. CTMU is also to provide an accurate current to charge the keys and ADC obtains the voltage and convert it into digital signal. In each scan loop, each key is discharged and charged to see whether human touch it. Once a touch is detected, the corresponding sound is generated and output through PWM to a speaker. The generation of sound is implemented by Karplus-Strong Algorithm (KSA) which is a waveguide string synthesis method. In addition, debounce function and chord function are also available on the touch piano. Up to 4 keys can be played as a chord at the same time.

The piano keyboard is built successfully but with no “control” system. After giving the voltage supply, the keyboard will work and people can play it. To design a musical notation based on this. A complete system should be built to control the keyboard. For instance, all notes that generated by people’s touching should be recorded for replay and displayed on the screen under the notation rules. Mode select may disable part of the keyboard or enable it at the proper time. To sum up, a large and completed system are supposed to be designed for automatic notation.

3. Initial approach

The buildup of the automatic notation system can be divided into three steps. The first step is creating a symbol library for all music symbols such as staffs, notes, and rests. The second. The second step is to create a notation algorithm to judge what should be displayed for each notes. The second step is building a user interface to merge everything together. People can easily see which mode they are in and what function can be used on the screen. The details for each step are as follows.

3.1. Symbol library

The display of LCD is based on drawing pixels. It is relatively simple to draw basic geometric shapes such as triangles, rectangles and circles, while the drawing of music symbols will be complicated. For instance, the drawing of treble clef is a combination of numerous arcs and lines. It is impossible to draw such a figure pixels by pixels or through base functions provided by TFT library. Therefore, the function draw_bitmap() is chosen for displaying the musical symbols. It is suitable for drawing images with irregular figures. The bitmap of different symbol can be found on internet or easily converted from other image format such as .png and .jpg.

3.2. Automatic notation algorithm

The main idea of notation algorithm is shows in Figure 3.1: after the initial tempo is set, there will be a cursor shows on the screen to indicate where the note will be drawn. As the notation process start, the vertical line moves rightward at the configured speed. When input signals come, corresponding notes will be drawn at the recent location of the cursor. After each bar complete, the notes in a bar will be rearranged to what they should be. This process will follow the rule of music notation. For example, if there are 8 notes located evenly in the bar, then each of them will be rearranged to eighth note. If 7 notes located in the bar and there is a slot at the end, then they will be rearranged to 6 eighth note and one quarter note.

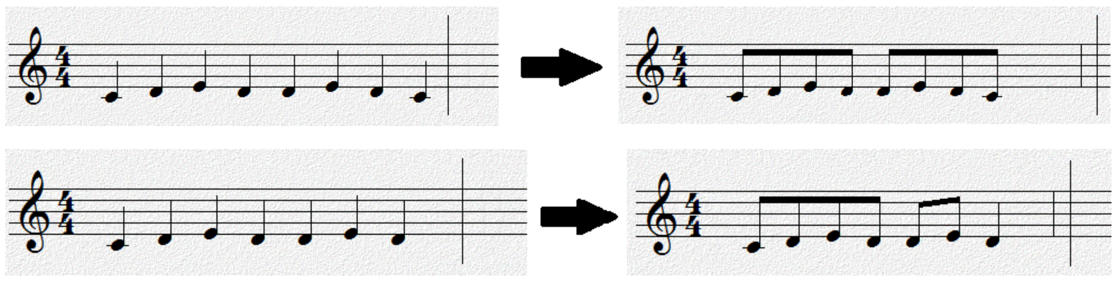


Figure 3.1 Examples of automatic notation system

3.3. User interface

A user interface is necessary for the notation system. Since users are given various options to choose, a well-organized user interface can tell users which mode they are in and which function is available now. The user interface can be implemented by finite state machine (FSM) in software. For each state, the FSM waits for the input, which is the choice of users, to change into the next stage. As soon as the input is given, Output should occur the same time as FSM changes its stage to display a new interface for users. Therefore, a Mealy FSM is more suitable for the building of user interface in this notation system.

4. Hardware implementation

The hardware implementation for this projection is shown in Figure 4.1. As mentioned before, the automatic notation system in this project is based on a self-made keyboard, which contains a PIC32 microcontroller, 5 multiplexers and a speaker. CD4051BCN, which is an 8-to-1 multiplexer, is used as pin extension. Up to 40 pins are supported in this project with 5 multiplexers. Since each octave has 12 keys including both black keys and white keys, 37 of 40 pins are chosen to be the piano keys which is 3 octaves plus 1. The rest 3 pins are idle but can be design as the control buttons. A speaker is connected to PIC32 as the sound output. PWM is used as the generation method of the sound because it is feasible to tune which tones accurately.

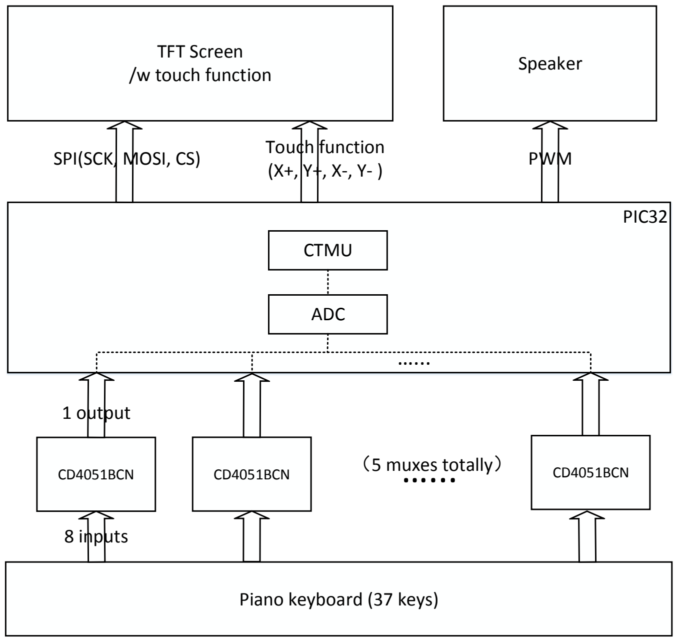


Figure 4.1 Hardware design block diagram

Beside the keyboard implementation, a TFT screen is essential in this system to show the user interface and the results of musical notation. After careful consideration, a 3.5'' TFT touchscreen shown in Figure 4.2 is chosen to take in charge of displaying. This TFT screen has a resolution of 480x320, which is big enough to show a staff on it. The touch function facilitates the mode select by touching the screen. Given the features above, this screen is ideal for the design in this project.

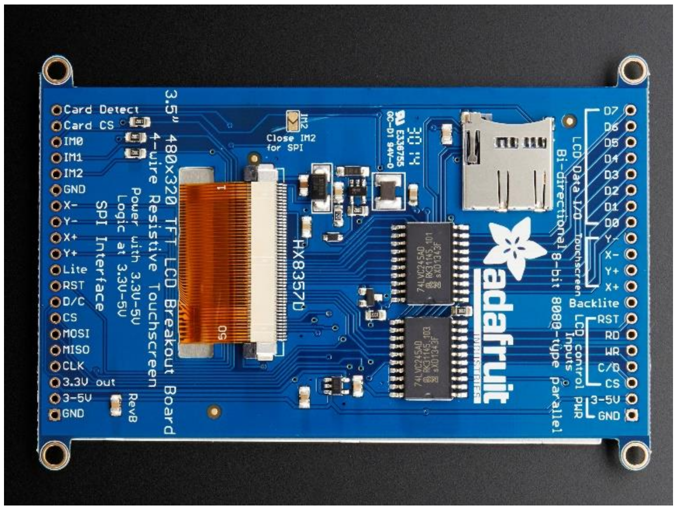


Figure 4.2 3.5'' TFT touchscreen (https://learn.adafruit.com/assets/18885)

This TFT screen can works in two mode. The first one is the 8-bit mode in which 8 bits can be sent at a time. TFT can work in a high speed but with all D0 to D8 connecting to the microcontroller. Due to the limitation of the pin number on PIC32, this working mode has to be abandoned. Instead, the second mode, SPI mode is chosen in this project. Generally, SPI includes a connection bundle of 4 pins – SCK, CS, MOSI, and MISO. MISO is not necessary in this project since no data for TFT to MCU is required. A D/C pin is supposed to be used to switch the sending of data and command. Therefore, only 4 pins together are needed to maintain the display of TFT. Touch function requires a connection of 4 additional pins – X+, X-, Y+, and Y- on the TFT screen. Correspondingly, two digital ports and two analog ports with ADC function on MCU should be used to realize the touch function. See Section 5.2.1 for details.

5. Software implementation

The main part of this projection is on software coding. It includes the building of a musical symbol library, an automatic notation algorithm and also a user interface. The musical symbol library requires multiple functions to draw various symbol on the screen. For complexed symbols, a prerequisite step is needed to convert them into bitmaps. Notation algorithm also consist of multiple functions. Notes with different duration should be recognized and then treated with different displaying methods. The details are discussed below.

5.1. Musical symbol display

5.1.1. Simple symbols

"Simple symbol" stands for the musical symbol with only basic geometric combination. These symbols can be drawn by the fundamental functions provided by TFT. For example, as shown in Figure 5.1, quarter notes include note heads and node tails. These two part can be easily drawn with function tft_fillRoundRect() and tft_drawLine(). Extra works should be spend on adjusting the size and ratio.

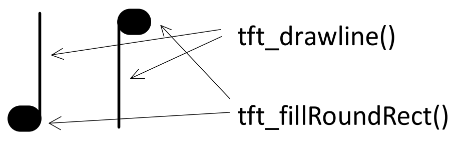
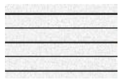
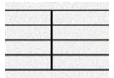
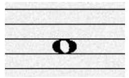
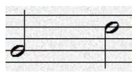
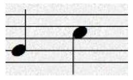
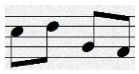
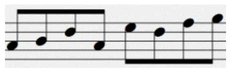
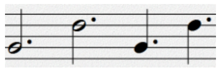
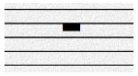
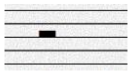


Figure 5.1 Example of drawing quarter note

All simple symbols are included in Table 5.1. The notation system supports up to eighth notes, so all kinds of notes that have a longer duration are required to be shown on the staff. Dotted quarter notes and dotted half notes are also available in this notation system. Since eighth note have various notations. Their symbols should be drawn separately. All symbol drawings are packed into different function with arguments of location. For the symbols that share a great similarity, they are packed in to a general function. This method makes the notation recognition process easier to manipulate. After giving x and y coordinates and call the related function, a symbol would be displayed on the screen in the given location. After the complete of this symbol library, it not only can be used in this musical notation project but also available for other projects related to musical symbol display.

Symbol name	Function name	Drawing symbol	Involved functions
Staff	drawstaff()		tft_drawline()
Bar	drawbar()		tft_drawline()
Whole Note	drawnote()[1]		tft_drawRoundRect()
Half Note	drawnote()		tft_drawRoundRect(), tft_drawline()
Quarter Note	drawnote()		tft_fillRoundRect(), tft_drawline()
Eighth Note[2]	draw2eighth()		tft_fillRoundRect(), tft_drawline()
Eighth Note	draw4eighth()		tft_fillRoundRect(), tft_drawline()
Dotted Note	Add_dot()		tft_fillCircle()
Whole Rest	drawrest()[3]		tft_fillRect()
Half Rest	drawrest()		tft_fillRect()

1. All types of single note drawings are packed into one function drawnote(), specific type can be chosen by giving different values to the "type" argument.
2. Eighth Notes have three types on displaying
3. All types of rests are packed into one function drawrest(), specific type can be chosen by giving different values to the "type" argument.

Table 5.1. Library for simple symbols

5.1.2. Complex symbols

Besides the symbol shows above, there are various kinds of symbol that cannot be easily drawn by basic shapes. For example, treble clef is a very common symbol appears on staffs. It contains arcs and lines that cannot be covered by basic functions. Therefore, the function draw_bitmap() is chosen to draw all symbols like this. The first step for displaying a treble clef, as shown in Figure 5.2, is to convert the image of this symbol into bitmap. There are many software available online can finish this step. The data obtained after the conversion is an array with 8-bit data. "0" stands for a white in this pixel and "1" stands for a black. Although this method is relatively slow for drawing symbols since it has to scan all pixels with the rectangular boundary, it is a convenient way to display any images that want to shows on the screen. Similar to the simple symbols, all symbols are packed into functions in Table 5.2.

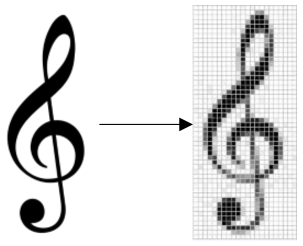


Figure 5.2. Example of drawing treble clef

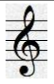
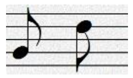
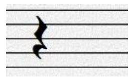
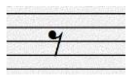
Symbol name	Function name	Drawing symbol	Involved functions
Treble Clef	drawstaff()		tft_drawBitmap()
Eighth Note	drawnote()		tft_drawBitmap()
Quarter Rest	drawrest()		tft_drawBitmap()
Eighth Rest	drawrest()		tft_drawBitmap()

Table 5.2. Library for complex symbols

5.1.3. Symbol variations

Among all symbols in the library, some of them has only a single shape such as the treble clef and all kinds of rest symbols. Once decide the coordinates, symbols can be shown on TFT screen at the certain location. However, notes have different shapes which depend on its pitch. For instances, as the situation shows in Figure 5.3, all notes below A4 has a tail downward while other notes at the upper place have upward tails. For the notes cannot be located on the 5-line staff, like C4, an additional line should be drawn according to the notation rules. Things become more complicated when comes to the connections of eighth notes. If two eighths notes sit next to each other, a connector should be drawn to make them a group. The angle of the connection line and the length of the tail for each note should depend on which two notes are being drawn. Therefore, plenty of conditional logic should be designed to properly draw each note and fit them in the staff. The variation is one of the most important part in automatic notation and is solved in the library functions.

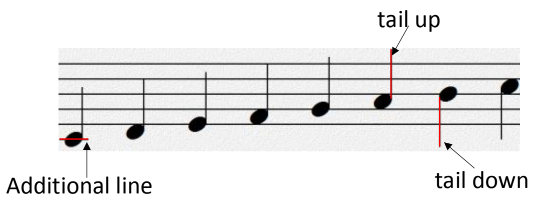


Figure 5.3 Example of symbol variations

5.2. Automatic notation process

In principle, the automatic notation process is to detect the pitch and duration for each note and display them properly on the screen. Pitches are easier to detect since it is all depend on which key people play. If a middle C is pressed, then a note should be display on its place to represent a middle C. Duration is far more difficult to detect since it is involved with time and the relationship with adjacent notes. Rest symbols are also necessary to be inserted according to what people play.

5.2.1. Pitch detection

Pitches determine the vertical location of notes on the staff. The process is shown in Figure 5.4. The information can be retrieved from the key scanning process. If a key is detect to be touch, the key number will be obtained immediately. Each key number maps to a unique vertical location. The larger the key number is, the higher the pitch is. As people continue playing the keyboard, notes would shows on the screen with different height. Notes variations should also be detected along with the pitches, which makes all notes displays under the notation rules with a proper format. The notes generated after pitch detection process are just “raw note”, which means the duration of each note has not been detected yet. For instance, after pressing one note, we cannot immediately judge whether it is a quarter note or not. Therefore, a whole note which stands for the “raw note” will be drawn for each note until the duration is detected.

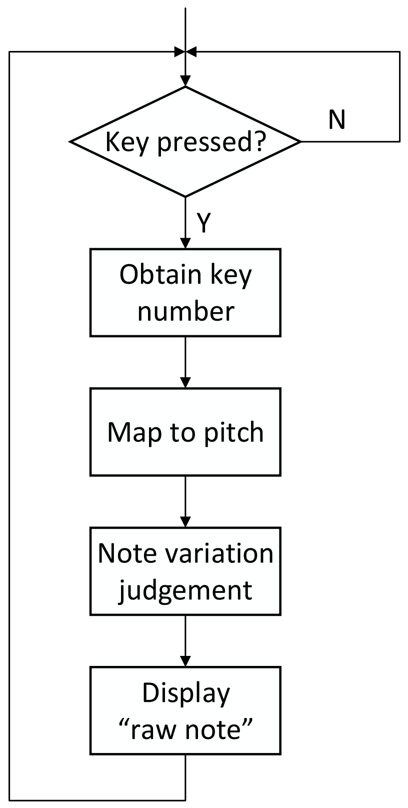


Figure 5.4. Pitch detection flow chart

5.2.2. Duration detection

The main task for duration detection is to replace each "raw note" with a proper notation according to how long the notes last. Since the duration is the interval time between the current note and the next one, the information cannot be obtained until the next input note is certain. Therefore, instead of doing a real-time judging for each note's duration, we start this process after each bar is done. As the process flow chart shown in Figure 5.5, duration detection mainly include three steps. The first step is to do the judgment for the first note, this note is in special situation that should be deal with. The second step is to judge all the interval notes from end to the beginning. The time it consumes varies depend on the number of notes in each bar. The last step is to rearrange all eighth notes. Eighth notes are special since its notation depends on how many eighth notes are placed next to each other. Each step replaces some of the "raw notes" with correct notations. After all steps done, all notes in this bar will be rearranged into proper notations by following the rules.

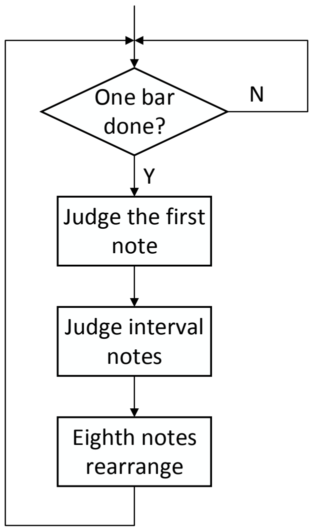


Figure 5.5 Duration detection flow chart

1) first note Judging

The first note means the note at the beginning of each bar. This note is in a special situation since no more notes are inserted between it and the bars. In real notation process, the beginning of a bar may not start with a note, which means an empty place may remain at the beginning which is a typo in notation. Therefore, not only the first note itself should be arranged, rest symbols may need to be inserted before it. The flow chart is shown in Figure 5.6. First, whether the first note is existed will be judge. If no notes are in this bar, then a whole rest will be drawn in the middle of this bar. If the first note does exists, we would judge whether a rest symbol should be drawn before the first note according to its distance to the beginning bar line. Then, the first note finishes and we jump to the second step.

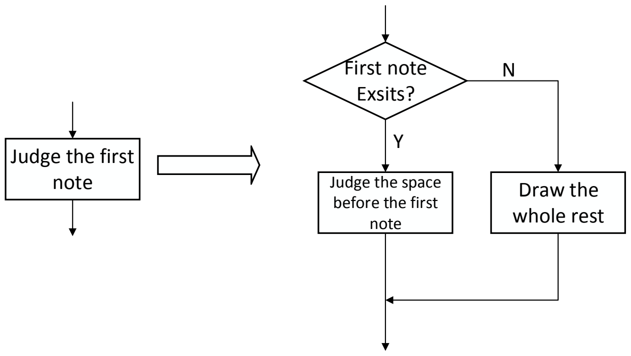


Figure 5.6 Flow chart of first note judging

An example is shown in Figure 5.7. There are three notes place in the bar when it finishes. If the duration detection process is done correctly but without first note judging, all three notes will be arranged to quarter notes but left an empty space at the beginning of this bar, which makes a notation error. First note judge is aimed to judge the interval space between the first note and the beginning bar line. In this example, the distance to the beginning bar line is large enough to draw a quarter note here. So a quarter rest symbol will be drawn here to keep the notation correct.

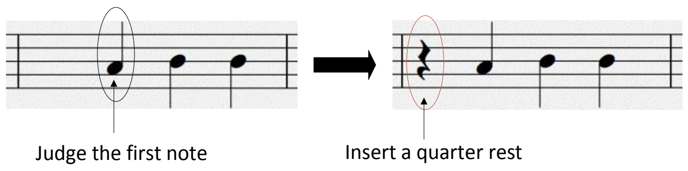


Figure 5.7 Example of first note judging

2) Interval note Judging

After the complete of the first note, all rest notes should be judged one by one as shown in Figure 5.8. The values of the notes depend on the distance from it to the next one, so the process should start from the end bar and execute from right to the left. The first node judged in this process is the right-most one. After obtain its value by judging the distance from it to the end bar line, this “raw note” can be replaced by a correct one following the notation rules. Then the note which on the left side can be judged based on this note. This process will iterative multiple times until all notes within the bar are done. However, eighth notes won’t be directly drawn in this step. This is because eighth notes have various notation format depend on how many eighth notes are next to each other. Therefore, we give each eighth note a marker which would be used in the next step and jump to the next note without drawing it.

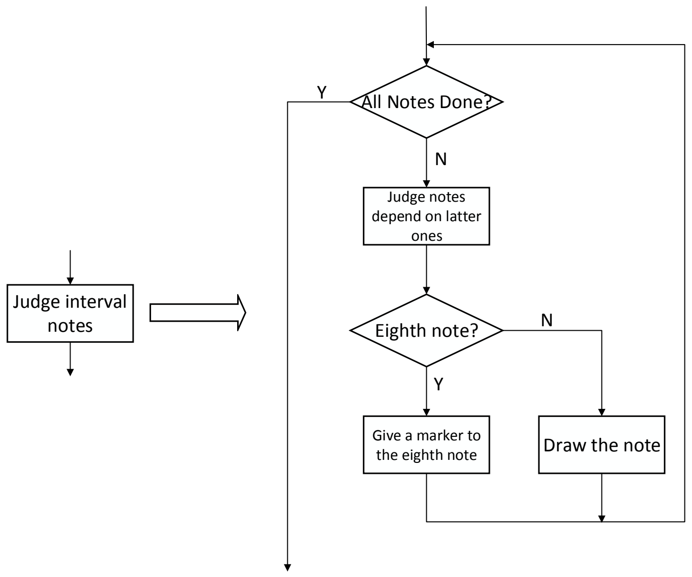

All Notes Done?
Y
N
Judge notes depend on latter ones
Judge interval notes
Eighth note?
N
Y
Give a marker to the eighth note
Draw the note

Figure 5.8 Flow chart of interval note judging

A specific example is shown in Figure 5.9. After the recoding and the pitch detection process. Three raw notes are placed within the bar. At the beginning, the note on the right side next to the end bar is taken into the judgement. The distance between this note and the end bar indicates a quarter note should be here to replace the raw note. After the first one done, the note on its left side is being judge. Likewise, it's a quarter note by judging the distance. The third note on the left side is the last one that to be executed. Due to the longer distance comparing to first two notes, this note is rearranged to be a half note. By then, all raw notes within the bar are replaced by the ones with correct values.

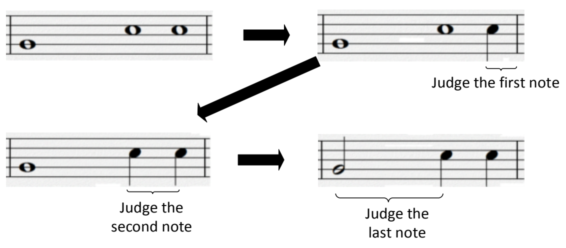


Figure 5.9. Example of Interval note judging

3) Eighth note rearrangement

If the notation system only include whole notes, half notes, and quarter notes, the first two step is completely enough for making the notation correct. However, when it comes to the judgement of eighth notes, things becomes quite different. If an eighth note is detected during the second step, it cannot be drawn as a single eighth note immediately to replace the row note. If there is another eighth note places next to it, a connection line should be drawn instead of their own tails. Similarly, if four eighth notes are lying closely with any other note in the middle. A longer horizontal line is supposed to be drawn to connect all these four notes together. Therefore, eighth note should be judged by another way differs from other notes as shown in Figure 5.10.

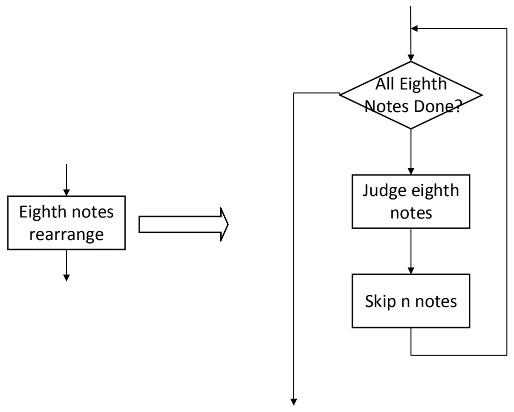


Figure 5.10 Flow chart of eighth note rearrangement

The method used in this project is shown in Figure 5.11. After the second step, six eighth notes and one quarter node are detected in this bar. Instead of drawing them all, only quarters are drawn at the end of the second step. For eighth notes, an additional judgement will be executed in the third step to make them correct. First, the note on the left side is under judgement. If there are three more eighth nodes lie next to this note, all the four eighth notes will be connected by a connection line. Then, the judgement will skip all these three notes since they have already rearranged to the correct notation based on the first one. When judging the fifth note, there are no three extra eighth notes on the right side. Instead, one extra eighth note is adjacent to the current note. These two notes will be drawn together with a connection line and then skip the judgement of the next one. By now, all eighth notes are rearranged to their correct notation.

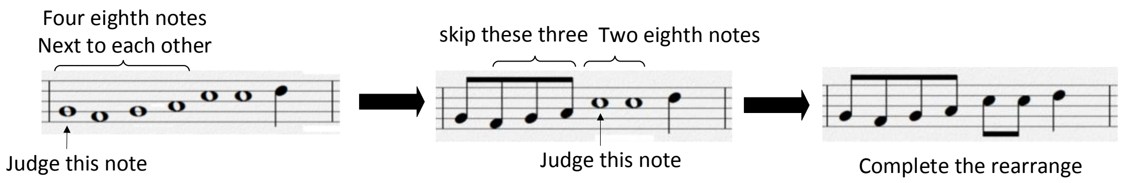


Figure 5.11 Example of eighth note rearrangement

5.2.3. Replay function

Replay function is built based on the automatic notation process. In duration detection, two array are used to record pitch and duration of each notes respectively. These arrays are used for rearrange the notes and draw them to the correct place to replace the “”raw notes”. As a result, only notes within a bar will be recorded. After the finish of each bar, both two arrays will be reset to empty for next time. To implement the replay function into the system. Two global arrays are used to record the pitches and durations for all the notes that are ever played since the start of the notation system. When calling the replay function, the cursor is initialized and start scanning the global duration array. Corresponding sound will be generated by converting the data in the global pitch array when the cursor meets the note stored in the duration array. As the scanning goes on, each note will be replayed according to its pitch until all data in the arrays are done. This function can help people to get what they played on the keyboard before. They can replay their song as many times as possible with the replay function implemented in this system. Meanwhile, the replay function is also an efficient way for testing the correctness of notation. We can use replay function and notation function to check each other in order to verify the functionality of the whole system.

5.3. User interface buildup

To integrate the keyboard and the automatic notation system together, a user interface is built base on FSM and the touch function. This user interface offers users two modes – free mode and notation mode. In free mode, the notation is disabled and all keys on the board is being scanning to generate the sound. People can use it purely as a keyboard to play songs. Notation mode only enable 15 keys for notation purpose, which is two octaves on the keyboard. An empty staff will be displayed on the TFT and wait for the start signal of users. In addition several operations are available such as starting a new one, quitting the current mode and replay the recorded song. Details are described below.

5.2.1. Touch function

The TFT used in this project has a resistive touchscreen. The touch function is implemented by two sheets overlapping on the top of the screen. When people press somewhere on the screen, a voltage divider is formed to indicate the x and y coordinates that people touched. Four pins – X+, X-, Y+, Y- are used for touch function, they should be able to configure with digital port to offer voltage supply and change to analog input to get the value through ADC To read the coordinate value. The first step is to provide voltage supply to X+ and X-, then one of Y+ or Y- is used as analog input to read the value after it becomes stable. This value is the x coordinate of the pressed location. The second step is swap the voltage to Y+ and Y-. X is used in this step to read the value using ADC. Then the first step is executed again and reiterates forever until the touch function is disabled. When touch function is enabled, ADC should be used iteratively for obtaining the coordinates. This means the scanning of the piano keyboard should be disabled since ADC cannot be shared by multiple channel at the same time. Therefore, the keyboard is disabled when users are selecting mode or push other buttons on the user interface and touchscreen cannot be used when users are playing the keyboard or doing the notation.

5.2.2. Finite State Machine

The FSM control unit designed for user interface is shown in Figure 5.12. There are 5 states in the control unit. The initial state is the START state. A welcome image is shown in this state when the system powers on. Touching anywhere will change the state to MODE state. In this state, choices are given to users to select the mode they want. If the free mode is chosen, then the FSM jumps to the FREE state. All keys on the keyboard are enabled for generating the sound and people can people the keyboard whatever they want. Since the keyboard and the touchscreen cannot be enabled at the same time. The quit button is built on the keyboard instead of on the touchscreen. The keyboard can supporting the scanning of 40 keys, which just 37 keys are using for piano keys. One extra key is attach at the back side of the keyboard to quit this mode and restart the touchscreen. If user choose the notation mode. An empty staff will show on the screen and start the notation process. To make users get prepared for the notation process, there are 8 beats delay before the cursor begins to move. As the notation process start, user can play songs using the keyboard and all input notes will be rearrange to correct notations after each bar's done. After the process finishes, there are three buttons enabled for users to choose. The first one is a "new" button. By pressing this, the system will restart and begin with a new staff for automatic notation. The second button is the 'replay" button, songs recorded before can be played back as many times as people want. The third button is the "quit" button. System will jump back to MODE state after this button being pressed.

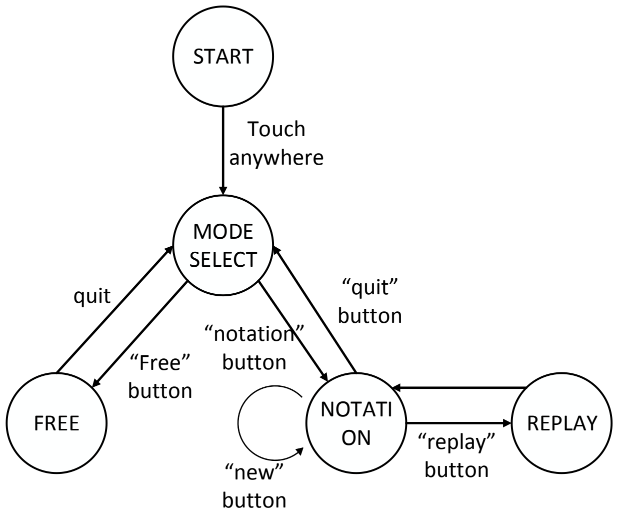


Figure 5.12 FSM control unit for user interface

6. Tests and results

The testing for the system is done separately to the three parts – symbol library, automatic notation process, and user interface. The testing of symbol library is relatively simple since we can see directly from the screen whether the symbol are drawn correctly or not. If the ratio or the sizing is not suitable on the screen, they are modified by changing the parameters and being redrawn on the screen again. The testing process iterative until all symbols are in proper sizes and shapes.

The testing of automatic notation process can be divided into two parts. The first part is the testing of the pitches of each notes. Once an input is detected by the keyboard scanning, a note with proper height and format should be displayed on the staff. Slight modification are applied for each notes until all pitches are shown correctly. Then the duration detecting process is tested with several steps. Firstly, the first note judgment should be tested to see whether rest symbols are inserted correctly. Then we should move to the interval note judgment. To make sure all are correct, different input combinations are supposed to be tested until all patterns pass the test. Finally we should do verification for eighth notes. This step also includes multiple combinations that are required to be tested carefully. We should see whether a raw note is replace with correct notation format, whether rest symbols are inserted with no mistakes and whether the connect line for eighth notes are drawing in a proper place and angle. A result of the automatic notation process is shown in Figure 6.1.

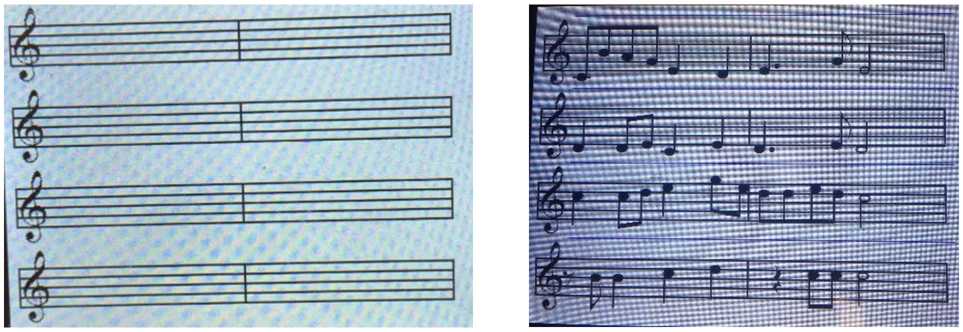


Figure 6.1. Automatic notation result (arbitrary note testing)

After the verification of the notation process, the build of user interface is taken under test. What should be verified is the transform logic, the output logic, and whether the buttons are operated correctly. The testing does not comes after all designs are done. Instead, an incremental test strategy is applied through the whole design process. Each single part will be under carefully test before adding more functions to the system.

7. Conclusion

A musical symbol library is built using basic functions applied by TFT screen. Some simple symbols are directly drawn by basic geometric shapes such as lines and rectangle while other complicated symbols are drawn using bitmaps. By building this library, various symbols can be easily drawn by the functions packed in the library. Each function has an input of xy-coordinates and types, which facilitates the drawing of musical symbols on any location on the screen with any types. This library is not only a foundation used for the displaying of automatic notation system, but also can be applied for other music-related project requires the display of staffs.

The automatic notation process can be separate into pitch detection and duration detection. After the end of each bar, all the notes in this bar will be rearranged and replaced with correct formats. This process take little time which is not detectable by human. The first note and eighth notes are separated from other notes due to the special situation they are in. After the notation system completes, multiple input patterns are used for testing it robustness and functionality. The system can support up to eighth notes with eight bars which is enough for recording basic songs. A replay function is also design for playing the whole song back. Users can replay their recorded songs as many time as they want by using this replay function.

A user interface is also design for mode selections and other operations. The touch function of the TFT screen is used for supporting these operations. Users can select the modes, start a new notation, replay the song and quit the current mode by pressing the buttons available on the screen or keyboard. All the features together make the automatic notation system capable enough to support basic composition process. Users can create their own songs and show them scientifically on the staff provided by the TFT screen.

8. Future improvement

Although the automatic notation system is already capable enough to deal with simple notation, it is not completed yet comparing to the huge amount of notation rules and symbol exist in real music composition. Future works may include but not limit to:

- Enrich the musical symbol library to support the display of more symbols
- Support the input from black key, implement sharp and flat symbols to the system.
- Support the input with multiple notes at the same time.
- Design new functions in FSM to support more pages of staff.
- Offer users the options to change the tempo.

9. Acknowledge

I would like to thank Professor Bruce Land for his guidance and instruction during this process. His insights for selecting electronic components help me to avoid potential hazards. Dr. Land always responds my questions with great enthusiasm and precise directions. During our meeting time every week, I receive lots of helpful advices and patient guidance from him. I truly appreciate his dedication in helping me to move forward.

10. Reference

- https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2015/wj225_hj424_lw569/wj225_hj424_lw569/wj225_hj424_lw569/index.html
- https://github.com/adafruit/Adafruit_HX8357_Library
- https://github.com/adafruit/Touch-Screen-Library
- https://people.ece.cornell.edu/land/courses/ece4760/PIC32/Microchip_stuff/2xx_datasheet.pdf
- http://ww1.microchip.com/downloads/en/DeviceDoc/60001168J.pdf

Appendix

User’s manual

1) Press anywhere on the touchscreen to start the system and enter the mode selection user interface
2) Choose the mode also by touching the screen.
3) If free mode is selected, all keys on the keyboard will be ready for play. To quit this mode press the touch button on the back side of the keyboard.
4) If notation mode is selected, wait eight beats until the process starts. Recording notes by pressing the keys on the keyboard.
5) After the notation process is done. Press the options shows on the screen to start a new one, replay the songs or quit this mode.

Code

1. Main.c

```
#include <math.h>
#include <inttypes.h>
#include "config.h"
#include "tft_master_1.h"
#include "tft_gfx.h"
#include <stdlib.h>    // need for rand function
#include <stdio.h>
#include <plib.h>
#include "touch_master.h"
#define        SYS_FREQ 40000000
#include "pt_cornell_1_2.h"

#include "notation_function.h"

// to use for cap charging time
#define NOP asm("nop");
// 1/2 microsec
#define wait20 NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;NOP;
// one microsec
#define wait40 wait20;wait20;

#define        WHOLE 0
#define        HALF 1
#define        QUARTER 2
```

```
#define       EIGHTH 3

static UINT8 system_state = 0;

#define STATE_START                 0
#define STATE_MODE      1
#define STATE_FREEPLAY  2
#define STATE_NOTATION_PRE 3
#define STATE_NOTATION_RUN 4

volatile int piano_start = 0 , piano_end = 0, freemode = 0, blink = 0;;



// thread control structs
static struct pt pt_ctmu, pt_music, pt_pattern, pt_drawkey, pt_time, pt_UI, pt_touch, pt_replay;

volatile UINT16 XMIN = 40, YMIN = 70, XMAX = 340, YMAX = 710;

// used in CTMU thread for pin scanning
volatile int pattern=0,key=0,key_num=0,s=0;
int key_save[40],press[40]; //store key number for which is pressed
int flag=0;    //trigger sound generation
int raw_adc0[40]; //store ADC value
int i=0,x=0,y=0; // variables for scanning

//Convert float to fix. a is a float
typedef signed int fix16 ;
#define float2fix16(a) ((fix16)((a)*65536.0)) // 2^16
#define fix2float16(a) ((float)(a)/65536.0)
// Fast fixed point multiply assembler macro
#define multfix16(a,b) ((fix16)(((( signed long long)(a))*(( signed long long)(b)))>>16)) //multiply two fixed 16:16

int generate_period = 2000 ; // PWM sample time
volatile int pwm_on_time = 1000 ;

// noise table
volatile unsigned int p;
volatile signed int noise_table[256] ;

//variables used in KSA calculation
//string 0
```

```
volatile signed int string[256], last_tune_out, last_tune_in, lowpass_out ;
signed int tune, damping ;
volatile char pluck, pushed ;
volatile int ptrin, ptrout;
//string 1
volatile signed int string1[256], last_tune_out1, last_tune_in1, lowpass_out1 ;
signed int tune1, damping1 ;
volatile char pluck1, pushed1 ;
volatile int ptrin1, ptrout1 ;
//string 2
volatile signed int string2[256], last_tune_out2, last_tune_in2, lowpass_out2 ;
signed int tune2, damping2 ;
volatile char pluck2, pushed2 ;
volatile int ptrin2, ptrout2;
//string 3
volatile signed int string3[256], last_tune_out3, last_tune_in3, lowpass_out3 ;
signed int tune3, damping3 ;
volatile char pluck3, pushed3 ;
volatile int ptrin3, ptrout3 ;

//used in music thread
char buffer[60];
volatile int drawkey=0,dtemp=0;
// string length
int note[45]={153,144,136,129,121,115,108,102,96, 91, 86, 81,
              76, 72, 68, 64, 61, 57, 54, 51, 48, 45, 43, 40,
              38, 36, 34, 32, 30, 28, 27, 25, 24, 22, 21, 20,
              19, 32, 19, 32, 204};
volatile char key_name[28][4]={"C4", "C4#","D4","D4#","E4","F4", "F4#","G4","G4#","A4","A4#", "B4",
                               "C5", "C5#","D5","D5#","E5","F5", "F5#","G5","G5#","A5","A5#", "B5",
                               "C6",    "N",    "N", "D5#"};
int note_last=0;
int string_length[4]={122,122,122,122};// tune the string by it's length
int str_en[4]={0,0,0,0};    //string enable

volatile int get_chord = 3, last_chord =50,cont=1;
#define C_chord 0
#define G_chord 2
#define NONE_chord 13
volatile char chord_name[3][10]={"C_chord", "G_chord","None"};
volatile int time_tick=0, notation_start=0, notation_run=0, notation_end=0;

volatile int dis[20], pit[20],a=0,b=0, dis_c[100], pit_c[100];

void ADC_init();




//interrupt service routine for KSA sound generation, sample time is 20kHz
void __ISR(_TIMER_2_VECTOR, ipl2) Timer2Handler(void)
{
        mT2ClearIntFlag(); //clear flag
        SetDCOC2PWM(pwm_on_time); // PWM output
        //string 0
          // low pass filter
          lowpass_out = multfix16(damping, (string[ptrin] + string[ptrout])>>1) ;
          // tuning all-pass filter
          string[ptrin] = multfix16(tune,(lowpass_out - last_tune_out)) + last_tune_in ;
        // all-pass state vars
        last_tune_out = string[ptrin];
        last_tune_in = lowpass_out;
          // update and wrap pointers
        if (ptrin==string_length[0]) ptrin=1;
        else ptrin=ptrin+1;
        if (ptrout==string_length[0]) ptrout=1;
        else ptrout=ptrout+1;

        //string 1
          // low pass filter
        lowpass_out1 = multfix16(damping1, (string1[ptrin1] + string1[ptrout1])>>1) ;
          // tuning all-pass filter
          string1[ptrin1] = multfix16(tune1,(lowpass_out1 - last_tune_out1)) + last_tune_in1 ;
        // all-pass state vars
        last_tune_out1 = string1[ptrin1];
        last_tune_in1 = lowpass_out1;
        // update and wrap pointers
        if (ptrin1==string_length[1]) ptrin1=1;
        else ptrin1=ptrin1+1;
        if (ptrout1==string_length[1]) ptrout1=1;
        else ptrout1=ptrout1+1;
```

```
//string 2
  // low pass filter
lowpass_out2 = multfix16(damping2, (string2[ptrin2] + string2[ptrout2])>>1) ;
  // tuning all-pass filter
  string2[ptrin2] = multfix16(tune2,(lowpass_out2 - last_tune_out2)) + last_tune_in2 ;
// all-pass state vars
last_tune_out2 = string2[ptrin2];
last_tune_in2 = lowpass_out2;
// update and wrap pointers
if (ptrin2==string_length[2]) ptrin2=1;
else ptrin2=ptrin2+1;
if (ptrout2==string_length[2]) ptrout2=1;
else ptrout2=ptrout2+1;

//string 3
  // low pass filter
lowpass_out3 = multfix16(damping3, (string3[ptrin3] + string3[ptrout3])>>1) ;
  // tuning all-pass filter
  string3[ptrin3] = multfix16(tune3,(lowpass_out3 - last_tune_out3)) + last_tune_in3 ;
// all-pass state vars
last_tune_out3 = string3[ptrin3];
last_tune_in3 = lowpass_out3;
// update and wrap pointers
if (ptrin3==string_length[3]) ptrin3=1;
else ptrin3=ptrin3+1;
if (ptrout3==string_length[3]) ptrout3=1;
else ptrout3=ptrout3+1;

// calculation trigger
//init string 0
if(str_en[0]==1)
{
        //string assignment
        for (p=0; p<string_length[0]; p++)
        {
                string[p] = noise_table[p] <<16 ;
        }
        //init pointer and string enable
        ptrout = 2;
        ptrin = 1;
        str_en[0]=0;
}

//init string 1
if(str_en[1]==1)
{
        //string assignment
        for (p=0; p<string_length[1]; p++)
        {
             string1[p] = noise_table[p] <<16 ;
        }
        //init pointer and string enable
        ptrout1 = 2;
        ptrin1 = 1;
        str_en[1]=0;
}

//init string 2
if(str_en[2]==1)
{
        //string assignment
        for (p=0; p<string_length[2]; p++)
        {
             string2[p] = noise_table[p] <<16 ;
        }
        //init pointer and string enable
        ptrout2 = 2;
        ptrin2 = 1;
        str_en[2]=0;
}

//init string 3
if(str_en[3]==1)
{
        //string assignment
        for (p=0; p<string_length[3]; p++)
        {
                string3[p] = noise_table[p] <<16 ;
        }
        //init pointer and string enable
        ptrout3 = 2;
```

```
            ptrin3 = 1;
            str_en[3]=0;
        }


        //assign value for    output
        pwm_on_time =    1000 + ((fix2float16(string[ptrin]))+
                                        (fix2float16(string1[ptrin1]))+
                                        (fix2float16(string2[ptrin2]))+
                                        (fix2float16(string3[ptrin3])))*4 ;

}

static PT_THREAD (protothread_touch(struct pt *pt))
{
        PT_BEGIN(pt);

                do{
                        // yield time 5 milliseconds
                        PT_YIELD_TIME_msec(5);
                        touch_getPoint();

                        if (touch_point.DONE && touch_point.Z) {
//                              debug();
                                if (touch_point.X >= XMIN && touch_point.X <= XMAX)
                                        touch_point.X = touch_point.X - XMIN;
                                else
                                        touch_point.Z = 0;  // make it an invalid touch

                                if (touch_point.Y >= YMIN && touch_point.Y <= YMAX)
                                        touch_point.Y = touch_point.Y - YMIN;
                                else
                                        touch_point.Z = 0;  // make it an invalid touch
                        }


                } while (touch_point.DONE==0 || touch_point.Z==0);
            //debug
            /*tft_fillRoundRect(0,10, 150, 40, 1, HX8357_BLACK);
                        tft_setCursor(0, 10);
                        tft_setTextColor(HX8357_YELLOW); tft_setTextSize(2);
                        sprintf(buffer,"x:%d, y:%d",touch_point.X , touch_point.Y );
                        tft_writeString(buffer);*/

        PT_EXIT(pt);
          PT_END(pt);
} // touch thread

int num_re = 1;
static PT_THREAD (protothread_replay(struct pt *pt))
{

        PT_BEGIN(pt);
          do{
                    // yield time 5 milliseconds
                    PT_YIELD_TIME_msec(14);

                    if (time_tick == dis_c[num_re])
                {
                            string_length[0] = note[pit_c[num_re]]; //assign length
                       str_en[0]=1;
                       num_re++;
                    }
                /*tft_fillRoundRect(0,100, 150, 40, 1, HX8357_BLACK);
                tft_setCursor(0, 100);
                tft_setTextColor(HX8357_YELLOW); tft_setTextSize(2);
                sprintf(buffer,"%d,%d",dis_c[num_re],time_tick );
                tft_writeString(buffer); */
                //PT_YIELD_TIME_msec(100);
                time_tick++;
          } while (num_re != b+1);
                //debug

        PT_EXIT(pt);
          PT_END(pt);
} // touch thread

static PT_THREAD (protothread_UI(struct pt *pt))
{
        PT_BEGIN(pt);
        system_state = STATE_START;
        while(1)
```

```
{
        PT_YIELD_TIME_msec(50);
        if(system_state == STATE_START)
        {
                //tft_fillScreen(HX8357_BLACK);
                tft_setTextColor(HX8357_BLACK);
                tft_setTextSize(3);
                sprintf(buffer, "PRESS TO START\n");

                tft_setCursor(115, HX8357_TFTHEIGHT/2);
                tft_writeString(buffer);

                touch_reset();
                do{
                        touch_getPoint();
                        delay_ms(10);
                } while (touch_point.DONE==0 || touch_point.Z==0);

                system_state = STATE_MODE;
                if(system_state = STATE_MODE)
                {
                        tft_fillScreen(HX8357_WHITE);
                        tft_setTextColor(HX8357_BLACK);
                        tft_setTextSize(4);
                        sprintf(buffer, "SELECT THE MODE:\n");
                        UINT16 left = (HX8357_TFTWIDTH-strlen(buffer)*6*2)/2;
                        tft_setCursor(left, 100);
                        tft_writeString(buffer);
                        sprintf(buffer, "Free        Notation\n");
                        tft_setCursor(left, 180);
                        tft_writeString(buffer);
                        sprintf(buffer, "Mode            Mode\n");
                        tft_setCursor(left, 210);
                        tft_writeString(buffer);
                }
        }
        else if(system_state == STATE_MODE)
        {
                piano_end = 0;
                PT_SPAWN(pt, &pt_touch, protothread_touch(&pt_touch));
                if( touch_point.Y < 300 ){
```

```
            system_state = STATE_NOTATION_RUN ;
            drawwholestaff();
            time_tick = 0;
            notation_start = 1;
            /*system_state = STATE_NOTATION_PRE;
            tft_fillScreen(HX8357_BLACK);
            tft_setTextColor(HX8357_YELLOW);
            tft_setTextSize(4);
            sprintf(buffer, "SET THE TEMPO:\n");
            UINT16 left = (HX8357_TFTWIDTH-strlen(buffer)*6*2)/2;
            tft_setCursor(left, HX8357_TFTHEIGHT/3);
            tft_writeString(buffer); */
        }
        else if( touch_point.Y > 300 ){
            system_state = STATE_FREEPLAY;
            tft_fillScreen(HX8357_WHITE);
            tft_setTextColor(HX8357_BLACK);
            tft_setTextSize(4);
            sprintf(buffer, "HAVE FUN!\n");
            UINT16 left = (HX8357_TFTWIDTH-strlen(buffer)*6*2)/2;
            tft_setCursor(left+30, HX8357_TFTHEIGHT/4);
            tft_writeString(buffer);
        }
    }
    else if(system_state == STATE_FREEPLAY){
        piano_start = 1;

        if(piano_end)
        {
            piano_start = 0 ;
            tft_fillScreen(HX8357_WHITE);
            tft_setTextColor(HX8357_BLACK);
            tft_setTextSize(4);
            sprintf(buffer, "SELECT THE MODE:\n");
            UINT16 left = (HX8357_TFTWIDTH-strlen(buffer)*6*2)/2;
            tft_setCursor(left, 100);
            tft_writeString(buffer);
            sprintf(buffer, "Free        Notation\n");
            tft_setCursor(left, 180);
            tft_writeString(buffer);
            sprintf(buffer, "Mode           Mode\n");
            tft_setCursor(left, 210);
            tft_writeString(buffer);
            system_state = STATE_MODE;
        }
    }

    else if(system_state == STATE_NOTATION_PRE)
    {
        PT_SPAWN(pt, &pt_touch, protothread_touch(&pt_touch));
        if(touch_point.X >200 && touch_point.X < 250 && touch_point.Y > 500 && touch_point.Y < 600 ){
            system_state = STATE_NOTATION_RUN;
            drawwholestaff();
            time_tick = 0;
            notation_start = 1;
        }
        if(touch_point.X >0 && touch_point.X < 50 && touch_point.Y > 500 && touch_point.Y < 600 ){
            system_state = STATE_START;
        }

    }
    else if(system_state == STATE_NOTATION_RUN)
    {
        if(notation_end)
        {
            //replay?
            PT_SPAWN(pt, &pt_touch, protothread_touch(&pt_touch));
            if(touch_point.X >0 && touch_point.X < 25 && touch_point.Y > 20 && touch_point.Y < 100 ){
                b = 0;
                notation_end = 0;
                time_tick = 0;
                drawwholestaff();
                notation_start = 1;
                num_re = 1 ;
            }
            else if(touch_point.X >50 && touch_point.X < 100 && touch_point.Y > 20 && touch_point.Y < 100 ){
                //notation_end = 0;
                /*system_state = STATE_START;*/
                time_tick = 0;
                num_re = 1 ;
                PT_SPAWN(pt, &pt_replay, protothread_replay(&pt_replay));
```

```
                    }
                    else if(touch_point.X >130 && touch_point.X < 150 && touch_point.Y > 20 && touch_point.Y < 100 ){
                        notation_end = 0;
                        tft_fillScreen(HX8357_WHITE);
                        tft_setTextColor(HX8357_BLACK);
                        tft_setTextSize(4);
                        sprintf(buffer, "SELECT THE MODE:\n");
                        UINT16 left = (HX8357_TFTWIDTH-strlen(buffer)*6*2)/2;
                        tft_setCursor(left, 100);
                        tft_writeString(buffer);
                        sprintf(buffer, "Free      Notation\n");
                        tft_setCursor(left, 180);
                        tft_writeString(buffer);
                        sprintf(buffer, "Mode          Mode\n");
                        tft_setCursor(left, 210);
                        tft_writeString(buffer);
                        system_state = STATE_MODE;
                    }

                }//piano enable



            }
        }
        PT_END(pt);
}

// Thread for string length assignment & debounce & chord playing
static PT_THREAD (protothread_music(struct pt *pt))
{
        PT_BEGIN(pt);
        while(1)
        {
            PT_WAIT_UNTIL(pt, flag == 1 ); // wait for start
            flag = 0; // clear flag
            //string assignment
            for(x=0;x<40;x++)
            {
                if(system_state == STATE_NOTATION_RUN)
                {
                if(x>11 && x != 13 && x != 15 && x != 18 && x != 20 && x != 22 && x != 25 && x != 27 && x != 30 && x != 32 && x != 34
&& x != 37){

                    //assign string0 and jump to string1
                    if(key_save[x] == 1 && press[x]==1)
                    {
                        string_length[y] = note[x]; //assign length
                        str_en[y]=1;    //string enable
                        dtemp=x;
                        press[x]=0;      // debounce
                        y++; //jump pointer
                        drawkey=1;
                        if(y == 3) y = 0;

                    }
                }
                else{
                    if(key_save[x] == 1 && press[x]==1)
                    {
                        get_chord = x;
                        if(get_chord == last_chord) get_chord = NONE_chord;
                        last_chord = get_chord;

                    /*tft_fillRoundRect(0,10, 150, 40, 1, HX8357_BLACK);
                    tft_setCursor(0, 10);
                    tft_setTextColor(HX8357_YELLOW); tft_setTextSize(2);
                    sprintf(buffer,"%s",chord_name[last_chord] );
                    tft_writeString(buffer); */
                    }
                }
                }
                if(system_state == STATE_FREEPLAY)
                {
                    //assign string0 and jump to string
                    if(key_save[39] == 1 && press[39]==1)
                    {
                        piano_end = 1;

                    }
                    else if(key_save[x] == 1 && press[x]==1)
                    {
                        string_length[y] = note[x]; //assign length
                        str_en[y]=1;    //string enable
                        dtemp=x;
                        press[x]=0;     // debounce
                        y++; //jump pointer
                        if(y == 3) y = 0;
                    }
                }
            }
        }
        PT_END(pt);
}


short nx=30;
short cbase=0;
short line=0;
static PT_THREAD (protothread_drawkey(struct pt *pt))
{
        PT_BEGIN(pt);
        cbase = 70;
        nx = 40;
        while(1)
        {
            PT_WAIT_UNTIL(pt, drawkey == 1 ); // wait for start
            drawkey = 0; // clear flag
            nx = 40 + time_tick;//23;
            smartnote(dtemp,nx,cbase,WHOLE);
            a++;
            dis[a] = time_tick;
            pit[a] = dtemp;
            b++;
            dis_c[b] = time_tick+381*line;
            pit_c[b] = dtemp;
            /*tft_fillRoundRect(0,100, 150, 40, 1, HX8357_BLACK);
            tft_setCursor(0, 100);
            tft_setTextColor(HX8357_YELLOW); tft_setTextSize(2);
            sprintf(buffer,"%d,%d",dis[a],a );
            tft_writeString(buffer);
            PT_YIELD_TIME_msec(100);*/
        }
        PT_END(pt);
```

```
}

volatile int flip=0/*,first=0,d=0*/;
static PT_THREAD (protothread_time(struct pt *pt))
{
    PT_BEGIN(pt);
    while(1)
    {
        PT_YIELD_TIME_msec(14);
        if(notation_start)
        {
            if(time_tick%24     == 0 || time_tick == 0)
            {

                if(flip == 0) {
                    tft_fillRoundRect(1,1, 8 ,8, 4, HX8357_RED);
                    flip = 1;
                    string_length[3] = 6; //assign length
                    str_en[3]=1;    //string enable
                }
                else if(flip ==1){
                    tft_fillRoundRect(1,1, 8 ,8, 4, HX8357_YELLOW);
                    flip = 0;
                    string_length[3] = 5; //assign length
                    str_en[3]=1;    //string enable
                }
            }
            time_tick++;
            if(time_tick == 191)
            {
                notation_start = 0;
                time_tick = 0;
                notation_run = 1;
                piano_start = 1;
            }
        }
        if(notation_run)
        {
```

```
if(line == 0 && time_tick == 381) {
        line++;
        cbase = 145;
        nx = 40;
        time_tick = 0;
}
else if(line == 1 && time_tick == 381) {
        line++;
        cbase = 220;
        nx = 40;
        time_tick = 0;
}
else if(line == 2 && time_tick == 381    ) {
        line++;
        cbase = 295;
        nx = 40;
        time_tick = 0;
}
else if(line == 3 && time_tick == 381    ) {
        line=0;
        notation_run = 0;
        notation_end = 1;
        piano_start = 0;
        cbase = 70;
        nx = 40;
        flip = 0;


}

if(time_tick%24        == 0 || time_tick == 0)
{
        //tick-tock
        if(flip == 0) {
                tft_fillRoundRect(1,1, 8 ,8, 4, HX8357_RED);
                flip = 1;
                string_length[3] = 6; //assign length
                str_en[3]=1;    //string enable
        }
        else if(flip ==1){
                tft_fillRoundRect(1,1, 8 ,8, 4, HX8357_YELLOW);
```

```
                    flip = 0;
                    string_length[3] = 5; //assign length
                    str_en[3]=1;    //string enable
                }
            }



            time_tick++;
            tft_drawPixel(time_tick+40,cbase-60 , HX8357_BLUE);

            if(time_tick==180) {smartjudge2(dis, pit, a, cbase, 180);a=0;}
            if(time_tick==372) {smartjudge2(dis, pit, a, cbase, 372);a=0;}
            //if(time_tick==265) {smartjudge2(dis, pit, a, cbase, 265);a=0;}
        }
    }
    PT_END(pt);
}

static PT_THREAD (protothread_pattern(struct pt *pt))
{
    PT_BEGIN(pt);
    while(1) {
        if(get_chord == C_chord)
        {
            string_length[3] = note[0]; //assign length
            str_en[3]=1;    //string enable
            PT_YIELD_TIME_msec(650);
            string_length[3] = note[7]; //assign length
            str_en[3]=1;    //string enable
            PT_YIELD_TIME_msec(650);
            string_length[3] = note[4]; //assign length
            str_en[3]=1;    //string enable
            PT_YIELD_TIME_msec(650);
            string_length[3] = note[7]; //assign length
            str_en[3]=1;    //string enable
            PT_YIELD_TIME_msec(650);
        }
        if(get_chord == G_chord)
        {
            string_length[3] = note[40]; //assign length
```

```
                str_en[3]=1;    //string enable
                PT_YIELD_TIME_msec(650);
                string_length[3] = note[7]; //assign length
                str_en[3]=1;    //string enable
                PT_YIELD_TIME_msec(650);
                string_length[3] = note[2]; //assign length
                str_en[3]=1;    //string enable
                PT_YIELD_TIME_msec(650);
                string_length[3] = note[7]; //assign length
                str_en[3]=1;    //string enable
                PT_YIELD_TIME_msec(650);
            }
                PT_YIELD_TIME_msec(5);

        }
        PT_END(pt);
}

//    CTMU thread for capacitance measurement & pin scanning
static PT_THREAD (protothread_ctmu(struct pt *pt))
{
        PT_BEGIN(pt);
        CTMUCONbits.ON = 1; // Turn on CTMU
        // choose a current level
        CTMUCONbits.IRNG = 3;
        // scanning and measurement
        while(1) {
            //Outer loop
            PT_WAIT_UNTIL(pt, piano_start == 1 ); // wait for start
            for(i=0; i<5; i++)
            {
                // change ADC channel after inner loop done
                if        (i==0)    SetChanADC10(    ADC_CH0_POS_SAMPLEA_AN9    );
                else if(i==1)    SetChanADC10(    ADC_CH0_POS_SAMPLEA_AN11 );
                else if(i==2)    SetChanADC10(    ADC_CH0_POS_SAMPLEA_AN0    );
                else if(i==3)    SetChanADC10(    ADC_CH0_POS_SAMPLEA_AN1    );
                else if(i==4)    SetChanADC10(    ADC_CH0_POS_SAMPLEA_AN5    );
                //inner loop
                for(pattern=0; pattern<8; pattern++)
                {
                    AcquireADC10(); // start ADC sampling
                    key = pattern;    // record key number
                    pattern <<= 2;    // shift for use
                    mPORTASetBits(pattern); // set address pattern

                    // dischrge the cap
                    CTMUCONbits.IDISSEN = 1; // start drain of circuit
                    // wait for discharge
                    WriteTimer3(0);
                    PT_YIELD_UNTIL(pt, ReadTimer3()>5000); // opt:2000    steady:4000
                    //stop discharging
                    CTMUCONbits.IDISSEN = 0; // End drain of circuit

                    // start charging and wait 2 microsecs
                    ConfigIntTimer2(T2_INT_OFF | T2_INT_PRIOR_2);    //disable ISR when charging
                    CTMUCONbits.EDG1STAT = 1;
                    wait40;wait40;
                    // end charging
                    CTMUCONbits.EDG1STAT = 0;
                    ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_2); // enable ISR after done

                    // stop samping and start conversion
                    ConvertADC10(); // end sampling & start conversion
                    // wait for complete
                    while (!AD1CON1bits.DONE){}; // Wait for ADC conversion
                    // read the result of channel from the idle buffer
                    raw_adc0[8*i+key] =    ReadADC10(0) ;

                    //judgment using the results
                    if(raw_adc0[8*i+key]<250) // touche220
                    {
                        key_save[8*i+key]=1;      //record key number in the array
                    }
                    else    if(raw_adc0[8*i+key]>300) // non-touched300
                    {
                        key_save[8*i+key]=0;
                        press[8*i+key]=1;            // for debounce
                    }
                    mPORTAClearBits(pattern);    // clear address pattern
                    pattern >>= 2;                          // shift for plus1

                }// inner loop end
```

```
        }// outer loop end
        flag=1;    // music thread begins to work
    } // NEVER exit while
   // END WHILE(1)
    PT_END(pt);
} // ctmu thread

/*static PT_THREAD (protothread_free(struct pt *pt))
{
    PT_BEGIN(pt);
    // scanning and measurement
    while(1) {
        //Outer loop
        PT_WAIT_UNTIL(pt, free_mode == 1 ); // wait for start
        if(blink)
        {
            for ()
            tft_fillCircle(short x0, short y0, short r, unsigned short color);
        }

    }
    PT_END(pt);
}*/
// === Main    ======================================================
void main(void) {

    SYSTEMConfigPerformance(PBCLK);
    ANSELA = 0; ANSELB = 0; CM1CON = 0; CM2CON = 0;

    //configuration for CTMU thread
    ADC_init();
    // open timer2 for discharge
    OpenTimer3(T3_ON    |    T3_PS_1_1 , 50000);



    //Configuration for KSA
    //Create table
     for (i=0; i<256; i++)
    {
        noise_table[i] =    (i<17)? i*7 : (i<241)? 128-i : i*7-1792    ;
    }
```

```
//string0 configuration
ptrout = 2; // circular pointer
  ptrin = 1; // circular pointer
  last_tune_out = 0 ;
  last_tune_in = 0 ;
  tune = float2fix16(0.12) ;
  damping = float2fix16(0.993) ; // must be 0.5<damping<=1.0
  pluck = 0;

//string0 configuration
ptrout1 = 2; // circular pointer
  ptrin1 = 1; // circular pointer
  last_tune_out1 = 0 ;
  last_tune_in1 = 0 ;
  tune1 = float2fix16(0.12) ;
  damping1 = float2fix16(0.993) ; // must be 0.5<damping<=1.0

//string2 configuration
ptrout2 = 2; // circular pointer
  ptrin2 = 1; // circular pointer
  last_tune_out2 = 0 ;
  last_tune_in2 = 0 ;
  tune2 = float2fix16(0.12) ;
  damping2 = float2fix16(0.993) ; // must be 0.5<damping<=1.0

//string3 configuration
ptrout3 = 2; // circular pointer
  ptrin3 = 1; // circular pointer
  last_tune_out3 = 0 ;
  last_tune_in3 = 0 ;
  //string_length[1] = 122 ;
  tune3 = float2fix16(0.12) ; //0.8
  damping3 = float2fix16(0.993) ; // must be 0.5<damping<=1.0

// KSA basic configuration
OpenOC2(OC_ON | OC_TIMER2_SRC | OC_PWM_FAULT_PIN_DISABLE, pwm_on_time , pwm_on_time);
PPSOutput(2, RPB5, OC2);
  OpenTimer2(T2_ON | T2_SOURCE_INT | T2_PS_1_1, generate_period);
ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_2);
mT2ClearIntFlag(); //    clear the interrupt flag
```

```
    //config threads
    PT_setup();
    INTEnableSystemMultiVectoredInt();
    PT_INIT(&pt_ctmu);
    PT_INIT(&pt_music);
    PT_INIT(&pt_drawkey);
    //PT_INIT(&pt_pattern);
    PT_INIT(&pt_time);
    PT_INIT(&pt_UI);

    //    config port
    PORTSetPinsDigitalOut(IOPORT_A, BIT_2);
    PORTSetPinsDigitalOut(IOPORT_A, BIT_3);
    PORTSetPinsDigitalOut(IOPORT_A, BIT_4);

    tft_init_hw();
    tft_begin();
    tft_fillScreen(HX8357_WHITE);
    tft_setRotation(3);
    touch_init();
    tft_drawBitmap(0, 0, image_data_piano, 480, 320,HX8357_BLACK) ;


   // round-robin scheduler for threads
   while (1){
        PT_SCHEDULE(protothread_ctmu(&pt_ctmu));
        PT_SCHEDULE(protothread_music(&pt_music));
        PT_SCHEDULE(protothread_drawkey(&pt_drawkey));
       // PT_SCHEDULE(protothread_pattern(&pt_pattern));
        PT_SCHEDULE(protothread_time(&pt_time));
        PT_SCHEDULE(protothread_UI(&pt_UI));
    }
} // main
// === end    ======================================================

void ADC_init()
{
    // set up the ADC ////////////////////////////////////////
      CloseADC10();        // ensure the ADC is off before setting the configuration
    #define PARAM1    ADC_FORMAT_INTG16 | ADC_CLK_MANUAL | ADC_AUTO_SAMPLING_OFF //
```

```
        #define  PARAM2    ADC_VREF_AVDD_AVSS  |  ADC_OFFSET_CAL_DISABLE  |  ADC_SCAN_OFF  |  ADC_SAMPLES_PER_INT_1  |
ADC_ALT_BUF_OFF | ADC_ALT_INPUT_OFF
      #define    PARAM3    ADC_CONV_CLK_PB    |    ADC_SAMPLE_TIME_15    |    ADC_CONV_CLK_Tcy    //ADC_SAMPLE_TIME_15|
ADC_CONV_CLK_Tcy
        #define PARAM4    ENABLE_AN9_ANA | ENABLE_AN11_ANA | ENABLE_AN0_ANA    | ENABLE_AN1_ANA | ENABLE_AN5_ANA
| ENABLE_AN2_ANA | ENABLE_AN3_ANA
        #define PARAM5    SKIP_SCAN_ALL
        SetChanADC10( ADC_CH0_NEG_SAMPLEA_NVREF | ADC_CH0_POS_SAMPLEA_AN9 ); // configure to sample AN11
        OpenADC10( PARAM1, PARAM2, PARAM3, PARAM4, PARAM5 ); // configure ADC using the parameters defined above
        EnableADC10(); // Enable the ADC
}
```

2. Notation function.c

```
#include "tft_master_1.h"
#include "tft_gfx.h"

#include "symbol_library.h"

#define        WHOLE 0
#define        HALF 1
#define        QUARTER 2
#define        EIGHTH 3

char buffer[60];

void drawstaff(short loc)
{
      tft_drawLine    (10, loc,        413, loc,        HX8357_BLACK);//31
      tft_drawLine    (10, loc+1,    413, loc+1,    HX8357_BLACK);//32
      tft_drawLine    (10, loc+9,    413, loc+9,    HX8357_BLACK);//40
      tft_drawLine    (10, loc+10, 413, loc+10, HX8357_BLACK);//41
      tft_drawLine    (10, loc+18, 413, loc+18, HX8357_BLACK);//49
      tft_drawLine    (10, loc+19, 413, loc+19, HX8357_BLACK);//50
      tft_drawLine    (10, loc+27, 413, loc+27, HX8357_BLACK);//58
      tft_drawLine    (10, loc+28, 413, loc+28, HX8357_BLACK);//59
      tft_drawLine    (10, loc+36, 413, loc+36, HX8357_BLACK);//67
      tft_drawLine    (10, loc+37, 413, loc+37, HX8357_BLACK);//68
      tft_drawBitmap(10, loc-18, clef, 32, 72,HX8357_BLACK);
}

void drawbar(short loc)
```

```
{
        tft_drawLine(9,     loc, 9,     loc+37, HX8357_BLACK);
        tft_drawLine(10,    loc, 10,    loc+37, HX8357_BLACK);
        tft_drawLine(220, loc, 220, loc+37, HX8357_BLACK);
        tft_drawLine(221, loc, 221, loc+37, HX8357_BLACK);
        tft_drawLine(413, loc, 412, loc+37, HX8357_BLACK);
        tft_drawLine(412, loc, 413, loc+37, HX8357_BLACK);
}

void drawwholestaff()
{
        tft_fillScreen(HX8357_WHITE);
        drawstaff(30);
        drawstaff(105);
        drawstaff(180);
        drawstaff(255);

        drawbar(30);
        drawbar(105);
        drawbar(180);
        drawbar(255);
        tft_setTextColor(HX8357_GREEN);
        tft_setTextSize(2);
        sprintf(buffer, "NEW\n");
        tft_setCursor(425, 40);
        tft_writeString(buffer);
        tft_setTextColor(HX8357_BLUE);
        sprintf(buffer, "REPLAY\n");
        tft_setCursor(410, 150);
        tft_writeString(buffer);
        tft_setTextColor(HX8357_RED);
        sprintf(buffer, "QUIT\n");

        tft_setCursor(420, 260);
        tft_writeString(buffer);
}

void drawnote(short x, short y, short base, short type)
{
        //HEAD
        if(type == WHOLE || type == HALF)
```

```
{
        tft_drawRoundRect(x,      y,      12, 8, 4, HX8357_BLACK);
        tft_drawRoundRect(x+1, y+1, 10, 6, 3, HX8357_BLACK);
}
else if(type == QUARTER || type == EIGHTH)
{
        tft_fillRoundRect(x, y, 12 ,8, 4, HX8357_BLACK);
}

//TAIL
if(type == HALF || type == QUARTER || type == EIGHTH)
{
        if(y <= base-22){
                tft_drawLine(x+2, y+4, x+2, y+32, HX8357_BLACK);
                tft_drawLine(x+1, y+4, x+1, y+32, HX8357_BLACK);
                if(type == EIGHTH)tft_drawBitmap(x-1, y+8, eighttail2, 13, 29,HX8357_BLACK) ;
        }
        else{
                tft_drawLine(x+10, y+4, x+10, y-24, HX8357_BLACK);
                tft_drawLine(x+11, y+4, x+11, y-24, HX8357_BLACK);
                if(type == EIGHTH) tft_drawBitmap(x+8, y-30, eighttail1, 13, 28,HX8357_BLACK) ;
        }
}
//LINES
if(y == base )
{
        tft_drawLine(x-3, y+3, x+14, y+3, HX8357_BLACK);
        tft_drawLine(x-3, y+4, x+14, y+4, HX8357_BLACK);
}
if(y == base-51 || y == base-55 )
{
        tft_drawLine(x-3, base-48, x+14, base-48, HX8357_BLACK);
        tft_drawLine(x-3, base-47, x+14, base-47, HX8357_BLACK);
}
if(y == base-60 )
{
        tft_drawLine(x-3, base-48, x+14, base-48, HX8357_BLACK);
        tft_drawLine(x-3, base-47, x+14, base-47, HX8357_BLACK);
        tft_drawLine(x-3, base-56, x+14, base-56, HX8357_BLACK);
        tft_drawLine(x-3, base-57, x+14, base-57, HX8357_BLACK);
}
```

```
}

void smartnote(int dtemp,short nx,short cbase, short type)
{
        switch(dtemp){
                case 12:drawnote(nx,cbase      ,cbase,type);break;
                case 14:drawnote(nx,cbase-3 ,cbase,type);break;
                case 16:drawnote(nx,cbase-7 ,cbase,type);break;
                case 17:drawnote(nx,cbase-12,cbase,type);break;
                case 19:drawnote(nx,cbase-16,cbase,type);break;
                case 21:drawnote(nx,cbase-21,cbase,type);break;
                case 23:drawnote(nx,cbase-25,cbase,type);break;
                case 24:drawnote(nx,cbase-30,cbase,type);break;
                case 26:drawnote(nx,cbase-34,cbase,type);break;
                case 28:drawnote(nx,cbase-39,cbase,type);break;
                case 29:drawnote(nx,cbase-43,cbase,type);break;
                case 31:drawnote(nx,cbase-47,cbase,type);break;
                case 33:drawnote(nx,cbase-51,cbase,type);break;
                case 35:drawnote(nx,cbase-55,cbase,type);break;
                case 36:drawnote(nx,cbase-60,cbase,type);break;
                default:drawnote(nx,cbase,cbase,type );
        }
}

void adddot(int dtemp,short nx,short cbase)
{
        switch(dtemp){
                case 12:tft_fillCircle(nx+18,cbase+1 ,2,HX8357_BLACK);break;
                case 14:tft_fillCircle(nx+18,cbase+1 ,2,HX8357_BLACK);break;
                case 16:tft_fillCircle(nx+18,cbase-8 ,2,HX8357_BLACK);break;
                case 17:tft_fillCircle(nx+18,cbase-8 ,2,HX8357_BLACK);break;
                case 19:tft_fillCircle(nx+18,cbase-17,2,HX8357_BLACK);break;
                case 21:tft_fillCircle(nx+18,cbase-17,2,HX8357_BLACK);break;
                case 23:tft_fillCircle(nx+18,cbase-26,2,HX8357_BLACK);break;
                case 24:tft_fillCircle(nx+18,cbase-26,2,HX8357_BLACK);break;
                case 26:tft_fillCircle(nx+18,cbase-35,2,HX8357_BLACK);break;
                case 28:tft_fillCircle(nx+18,cbase-35,2,HX8357_BLACK);break;
                case 29:tft_fillCircle(nx+18,cbase-44,2,HX8357_BLACK);break;
                case 31:tft_fillCircle(nx+18,cbase-44,2,HX8357_BLACK);break;
                case 33:tft_fillCircle(nx+18,cbase-52,2,HX8357_BLACK);break;
                case 35:tft_fillCircle(nx+18,cbase-52,2,HX8357_BLACK);break;
```

```
            case 36:tft_fillCircle(nx+18,cbase-61,2,HX8357_BLACK);break;
            default:tft_fillCircle(nx+18,cbase,2,HX8357_BLACK );
        }
}

void drawrest(short x, short base, short type)
{
        if(type == WHOLE)    tft_fillRect(x, base - 30, 10,4,HX8357_BLACK);
        if(type == HALF)      tft_fillRect(x, base - 25, 10,4,HX8357_BLACK);
        if(type == QUARTER)tft_drawBitmap(x, base - 35, quarterrestbm, 8, 27,HX8357_BLACK) ;
        if(type == EIGHTH) tft_drawBitmap(x, base - 27, eighthrestbm, 12, 16,HX8357_BLACK) ;
}

void smartjudge(int dis[50], int pit[50], int a, short cbase, int time)
{
        int first=0,d=0;
        first++;
        if(a == 0) drawrest(time-12,cbase, WHOLE);
        while(a != 0)
        {
            if(first)
            {
                first = 0;
                //if(dis[a]>=0 && dis[a]<12) smartnote(pit[a],dis[a]+35,cbase,WHOLE);
                if(dis[a]>=time-70 && dis[a]<time-47) {smartnote(pit[a],dis[a]+40,cbase,QUARTER); drawrest(time+8,cbase, HALF);}
                else if(dis[a]>=time-47 && dis[a]<time-24) smartnote(pit[a],dis[a]+40,cbase,HALF);
                else if(dis[a]>=time-24 && dis[a]<=time) smartnote(pit[a],dis[a]+40,cbase,QUARTER);
            }
            else
            {
                d = dis[a+1]-dis[a];
                if(d< 33 )                smartnote(pit[a],dis[a]+40,cbase,QUARTER);
                if(d>=33 && d<56 )smartnote(pit[a],dis[a]+40,cbase,HALF);
                if(d>=56 )                {smartnote(pit[a],dis[a]+40,cbase,HALF);adddot(pit[a],dis[a]+40,cbase);}//////////////////
            }
            //smartnote(pit[a],dis[a]+35,cbase,QUARTER);
            a--;
        }
        if(first == 0)
        {
            a++;
            //if(dis[a]>=0 && dis[a]<12) smartnote(pit[a],dis[a]+35,cbase,WHOLE);
            if(dis[a]>=time-70 && dis[a]<time-47) drawrest(time-37,cbase, QUARTER);
            else if(dis[a]>=time-47 && dis[a]<time-24) drawrest(time-26,cbase, HALF);
            else if(dis[a]>=time-24 && dis[a]<=time) {drawrest(time-26,cbase, HALF); drawrest(time+8,cbase, QUARTER);}
            a--;
        }
}

void smartjudge2(int dis[50], int pit[50], int m, short cbase, int time)
{
        int first=0,d=0,id[50]={0},i,a;
        first++;
        a = m;
        if(a == 0) drawrest(time-65 ,cbase, WHOLE);
        while(a != 0)
        {
            if(first)
            {
                first = 0;
                if      (dis[a]>=time-168  &&  dis[a]<time-144)  {id[a]  =  1;  drawrest(time-120+40,cbase,  QUARTER);drawrest(time-
48+40,cbase, HALF);}
                else if(dis[a]>=time-144 && dis[a]<time-120) {smartnote(pit[a],dis[a]+40,cbase,QUARTER); drawrest(time-48+40,cbase,
HALF);}
                else if(dis[a]>=time-120 && dis[a]<time-96 ) {id[a] = 1; drawrest(time-48+40,cbase, HALF);}
                else if(dis[a]>=time-96    && dis[a]<time-72 ) smartnote(pit[a],dis[a]+40,cbase,HALF);
                else if(dis[a]>=time-72    && dis[a]<time-48 ) {id[a] = 1; drawrest(time-24+40,cbase, QUARTER);}
                else if(dis[a]>=time-48    && dis[a]<time-24 ) smartnote(pit[a],dis[a]+40,cbase,QUARTER);
                else if(dis[a]>=time-24    && dis[a]<=time      ) id[a] = 1;
            }
            else
            {
                d = dis[a+1]-dis[a];
                if       (d < 36 )                 id[a] = 1;
                else if(d>=36    && d<60    ) smartnote(pit[a],dis[a]+40,cbase,QUARTER);
                else if(d>=60    && d<84    ) {smartnote(pit[a],dis[a]+40,cbase,QUARTER);adddot(pit[a],dis[a]+40,cbase);}
                else if(d>=84    && d<108 ) smartnote(pit[a],dis[a]+40,cbase,HALF);
                else if(d>=108 && d<132 ) {smartnote(pit[a],dis[a]+40,cbase,HALF);adddot(pit[a],dis[a]+40,cbase);}
                else if(d>=132 && d<156 ) {smartnote(pit[a],dis[a]+40,cbase,HALF);adddot(pit[a],dis[a]+40,cbase);}
                else  if(d>=156                       )  {smartnote(pit[a],dis[a]+40,cbase,HALF);adddot(pit[a],dis[a]+40,cbase);drawrest(time-
36+40,cbase, EIGHTH);}
```

```
        }
        //smartnote(pit[a],dis[a]+35,cbase,QUARTER);
        a--;
    }

    for(i = 1;i<=m;i++)
    {
        if(id[i])
        {
            if(i+3<=m&&id[i+1]&&id[i+2]&&id[i+3])
{draw4eighth(dis[i]+40,pit[i],dis[i+1]+40,pit[i+1],dis[i+2]+40,pit[i+2],dis[i+3]+40,pit[i+3],cbase);i=i+3;}
            else
if(i+2<=m&&id[i+1]&&id[i+2]){draw2eighth(dis[i]+40,pit[i],dis[i+1]+40,pit[i+1]  ,cbase);smartnote(pit[i+2],dis[i+2]+40,cbase,EIGHTH);i
=i+2;}
            else if(i+1<=m&&id[i+1]) {draw2eighth(dis[i]+40,pit[i],dis[i+1]+40,pit[i+1],cbase);i=i+1;}
            else smartnote(pit[i],dis[i]+40,cbase,EIGHTH);
        }
    }

    if(first == 0)
    {
        if      (dis[1]>=time-168 && dis[1]<time-144)    drawrest(time-142,cbase, EIGHTH);
        else if(dis[1]>=time-144 && dis[1]<time-120)    drawrest(time-126,cbase, QUARTER);
        else if(dis[1]>=time-120 && dis[1]<time-96)    {drawrest(time-126,cbase, QUARTER);drawrest(time-96,cbase, EIGHTH);}
        else if(dis[1]>=time-96    && dis[1]<time-72)    {drawrest(time-90, cbase, HALF);}
        else if(dis[1]>=time-72    && dis[1]<time-48)    {drawrest(time-90, cbase, HALF);drawrest(time-36,cbase, EIGHTH) ;}
        else if(dis[1]>=time-48    && dis[1]<time-24)    {drawrest(time-90, cbase, HALF);drawrest(time-30,cbase, QUARTER);}
        else  if(dis[1]>=time-24     &&  dis[1]<=time)          {drawrest(time-90,  cbase,  HALF);drawrest(time-30,cbase,  QUARTER);
drawrest(time,cbase, EIGHTH);}
    }
}

void drawnotehead(int x, int y ,short base)
{
    tft_fillRoundRect(x, y, 12 ,8, 4, HX8357_BLACK);
    if(y == base )
    {
        tft_drawLine(x-3, y+3, x+14, y+3, HX8357_BLACK);
        tft_drawLine(x-3, y+4, x+14, y+4, HX8357_BLACK);
    }
```

```
        else if(y == base-51 || y == base-55 )
        {
                tft_drawLine(x-3, base-48, x+14, base-48, HX8357_BLACK);
                tft_drawLine(x-3, base-47, x+14, base-47, HX8357_BLACK);
        }
        else if(y == base-60 )
        {
                tft_drawLine(x-3, base-48, x+14, base-48, HX8357_BLACK);
                tft_drawLine(x-3, base-47, x+14, base-47, HX8357_BLACK);
                tft_drawLine(x-3, base-56, x+14, base-56, HX8357_BLACK);
                tft_drawLine(x-3, base-57, x+14, base-57, HX8357_BLACK);
        }
}

int dtemp2y(int dtemp, short cbase)
{
        switch(dtemp){
                case 12:return cbase;       break;
                case 14:return cbase-3; break;
                case 16:return cbase-7; break;
                case 17:return cbase-12;break;
                case 19:return cbase-16;break;
                case 21:return cbase-21;break;
                case 23:return cbase-25;break;
                case 24:return cbase-30;break;
                case 26:return cbase-34;break;
                case 28:return cbase-39;break;
                case 29:return cbase-43;break;
                case 31:return cbase-47;break;
                case 33:return cbase-51;break;
                case 35:return cbase-55;break;
                case 36:return cbase-60;break;
                default:return cbase;       break;
        }
}

void draw2eighth(int x0, int dtemp0, int x1, int dtemp1, short base)
{
        int y0,y1;
        y0 = dtemp2y(dtemp0,base);
        y1 = dtemp2y(dtemp1,base);
```

```
drawnotehead(x0, y0 ,base);
drawnotehead(x1, y1 ,base);

if(y1>y0+6)
{
        if(y0 <= base-22){
                tft_drawLine(x0+2, y0+4, x0+2, y1+32-5, HX8357_BLACK);
                tft_drawLine(x0+1, y0+4, x0+1, y1+32-5, HX8357_BLACK);
                tft_drawLine(x1+2, y1+4, x1+2, y1+32, HX8357_BLACK);
                tft_drawLine(x1+1, y1+4, x1+1, y1+32, HX8357_BLACK);

                tft_drawLine(x0+1, y1+32-5, x1+2, y1+32, HX8357_BLACK);
                tft_drawLine(x0+1, y1+31-5, x1+2, y1+31, HX8357_BLACK);
                tft_drawLine(x0+1, y1+30-5, x1+2, y1+30, HX8357_BLACK);
                tft_drawLine(x0+1, y1+29-5, x1+2, y1+29, HX8357_BLACK);
        }
        else{
                tft_drawLine(x0+10, y0+4, x0+10, y0-24, HX8357_BLACK);
                tft_drawLine(x0+11, y0+4, x0+11, y0-24, HX8357_BLACK);
                tft_drawLine(x1+10, y1+4, x1+10, y0-24+5, HX8357_BLACK);
                tft_drawLine(x1+11, y1+4, x1+11, y0-24+5, HX8357_BLACK);

                tft_drawLine(x0+10, y0-24, x1+11, y0-24+5, HX8357_BLACK);
                tft_drawLine(x0+10, y0-23, x1+11, y0-23+5, HX8357_BLACK);
                tft_drawLine(x0+10, y0-22, x1+11, y0-22+5, HX8357_BLACK);
                tft_drawLine(x0+10, y0-21, x1+11, y0-21+5, HX8357_BLACK);
        }
}
else if(y1<y0-6)
{
        if(y0 <= base-22){
                tft_drawLine(x0+2, y0+4, x0+2, y0+32, HX8357_BLACK);
                tft_drawLine(x0+1, y0+4, x0+1, y0+32, HX8357_BLACK);
                tft_drawLine(x1+2, y1+4, x1+2, y0+32-5, HX8357_BLACK);
                tft_drawLine(x1+1, y1+4, x1+1, y0+32-5, HX8357_BLACK);

                tft_drawLine(x0+1, y0+32, x1+2, y0+32-5, HX8357_BLACK);
                tft_drawLine(x0+1, y0+31, x1+2, y0+31-5, HX8357_BLACK);
                tft_drawLine(x0+1, y0+30, x1+2, y0+30-5, HX8357_BLACK);
                tft_drawLine(x0+1, y0+29, x1+2, y0+29-5, HX8357_BLACK);
        }
```

```
            else{
                tft_drawLine(x0+10, y0+4, x0+10, y1-24+5, HX8357_BLACK);
                tft_drawLine(x0+11, y0+4, x0+11, y1-24+5, HX8357_BLACK);
                tft_drawLine(x1+10, y1+4, x1+10, y1-24, HX8357_BLACK);
                tft_drawLine(x1+11, y1+4, x1+11, y1-24, HX8357_BLACK);

                tft_drawLine(x0+10, y1-24+5, x1+11, y1-24, HX8357_BLACK);
                tft_drawLine(x0+10, y1-23+5, x1+11, y1-23, HX8357_BLACK);
                tft_drawLine(x0+10, y1-22+5, x1+11, y1-22, HX8357_BLACK);
                tft_drawLine(x0+10, y1-21+5, x1+11, y1-21, HX8357_BLACK);
            }
        }
        else
        {
            if(y0 <= base-22){
                tft_drawLine(x0+2, y0+4, x0+2, y0+32, HX8357_BLACK);
                tft_drawLine(x0+1, y0+4, x0+1, y0+32, HX8357_BLACK);
                tft_drawLine(x1+2, y1+4, x1+2, y1+32, HX8357_BLACK);
                tft_drawLine(x1+1, y1+4, x1+1, y1+32, HX8357_BLACK);

                tft_drawLine(x0+1, y0+32, x1+2, y1+32, HX8357_BLACK);
                tft_drawLine(x0+1, y0+33, x1+2, y1+33, HX8357_BLACK);
                tft_drawLine(x0+1, y0+34, x1+2, y1+34, HX8357_BLACK);
                tft_drawLine(x0+1, y0+35, x1+2, y1+35, HX8357_BLACK);
            }
            else{
                tft_drawLine(x0+10, y0+4, x0+10, y0-24, HX8357_BLACK);
                tft_drawLine(x0+11, y0+4, x0+11, y0-24, HX8357_BLACK);
                tft_drawLine(x1+10, y1+4, x1+10, y1-24, HX8357_BLACK);
                tft_drawLine(x1+11, y1+4, x1+11, y1-24, HX8357_BLACK);

                tft_drawLine(x0+10, y0-24, x1+11, y1-24, HX8357_BLACK);
                tft_drawLine(x0+10, y0-25, x1+11, y1-25, HX8357_BLACK);
                tft_drawLine(x0+10, y0-26, x1+11, y1-26, HX8357_BLACK);
                tft_drawLine(x0+10, y0-27, x1+11, y1-27, HX8357_BLACK);
            }
        }

    }
```

```
int findmax(int x0,int x1,int x2, int x3)
{
        int max;
        if(x0>x1) max=x0;else max=x1;
        if(x2>max)max=x2;
        if(x3>max)max=x3;
        return max;
}

int findmin(int x0,int x1,int x2, int x3)
{
        int min;
        if(x0>x1) min=x1;else min=x0;
        if(min>x2)min=x2;
        if(min>x3)min=x3;
        return min;
}

void draw4eighth(int x0, int dtemp0, int x1, int dtemp1,int x2, int dtemp2, int x3, int dtemp3,    short base)
{
        int y,y0,y1,y2,y3;
        y0 = dtemp2y(dtemp0,base);
        y1 = dtemp2y(dtemp1,base);
        y2 = dtemp2y(dtemp2,base);
        y3 = dtemp2y(dtemp3,base);

        drawnotehead(x0, y0 ,base);
        drawnotehead(x1, y1 ,base);
        drawnotehead(x2, y2, base);
        drawnotehead(x3, y3 ,base);
        if(y0 <= base-22){
                y = findmax(y0,y1,y2,y3);

                tft_drawLine(x0+2, y0+4, x0+2, y+32, HX8357_BLACK);
                tft_drawLine(x0+1, y0+4, x0+1, y+32, HX8357_BLACK);
                tft_drawLine(x1+2, y1+4, x1+2, y+32, HX8357_BLACK);
                tft_drawLine(x1+1, y1+4, x1+1, y+32, HX8357_BLACK);
                tft_drawLine(x2+2, y2+4, x2+2, y+32, HX8357_BLACK);
                tft_drawLine(x2+1, y2+4, x2+1, y+32, HX8357_BLACK);
                tft_drawLine(x3+2, y3+4, x3+2, y+32, HX8357_BLACK);
                tft_drawLine(x3+1, y3+4, x3+1, y+32, HX8357_BLACK);
```

```
            tft_drawLine(x0+1, y+32, x3+2, y+32, HX8357_BLACK);
            tft_drawLine(x0+1, y+31, x3+2, y+31, HX8357_BLACK);
            tft_drawLine(x0+1, y+30, x3+2, y+30, HX8357_BLACK);
            tft_drawLine(x0+1, y+29, x3+2, y+29, HX8357_BLACK);
        }
        else{
            y = findmin(y0,y1,y2,y3);

            tft_drawLine(x0+10, y0+4, x0+10, y-24, HX8357_BLACK);
            tft_drawLine(x0+11, y0+4, x0+11, y-24, HX8357_BLACK);
            tft_drawLine(x1+10, y1+4, x1+10, y-24, HX8357_BLACK);
            tft_drawLine(x1+11, y1+4, x1+11, y-24, HX8357_BLACK);
            tft_drawLine(x2+10, y2+4, x2+10, y-24, HX8357_BLACK);
            tft_drawLine(x2+11, y2+4, x2+11, y-24, HX8357_BLACK);
            tft_drawLine(x3+10, y3+4, x3+10, y-24, HX8357_BLACK);
            tft_drawLine(x3+11, y3+4, x3+11, y-24, HX8357_BLACK);

            tft_drawLine(x0+10, y-24, x3+11, y-24, HX8357_BLACK);
            tft_drawLine(x0+10, y-23, x3+11, y-23, HX8357_BLACK);
            tft_drawLine(x0+10, y-22, x3+11, y-22, HX8357_BLACK);
            tft_drawLine(x0+10, y-21, x3+11, y-21, HX8357_BLACK);
        }

    }
```