University of British Columbia
Electrical and Computer Engineering
ELEC 291 / ELEC 292 Winter 2021
Instructor, Dr. Jesus Calvino-Fraga
Section 202

Project 1 - Capacitive Sensor Reaction Game

Student #	Student Name	% Points	Signature
18693002	Peter Kim	100	
97939987	Jonathan Li	100	
55747315	Manjot Mangat	100	

TABLE OF CONTENTS

INTRODUCTION

The objective of this project was to create a game by utilizing knowledge in assembly code, circuit analysis and capacitors. In doing so, the team was informed to ensure that all components of the project were designed from scratch and that the games created include some form of capacitor detection functionality.

The team approached the design challenge with two main goals:

1. Create a game where two players react to randomly generated sounds. The first to react at a high pitch and hit their side of the capacitor earns a point.
2. Create a piano-like game with the capacitors.

An overview of the results is as follows:

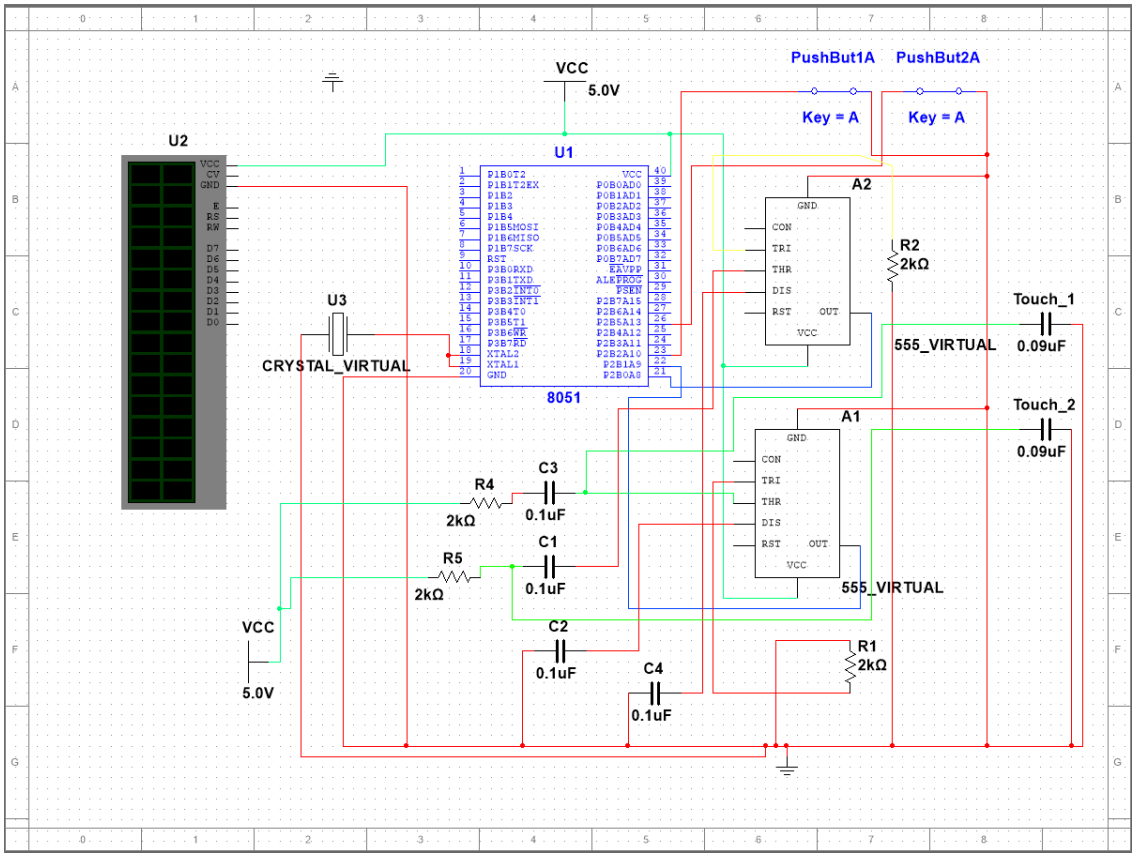
As the game begins, a greeting is displayed to welcome the players. Then, a menu of games follows that allows the user to choose from either ‘React!’ by pressing “Key1” (P0.0 on the microcontroller) or ‘Launch Pad’ by pressing “Key 2” (P0.3). Subsequently, the player is faced with a beginning menu that starts the game as soon as the player presses Key1.

In “React”, each player is responsible for their side of the capacitor. After either a high or low pitch is played, a player may press on their capacitor. If the player presses their capacitor after a high pitch sound is played, the player wins a point. Conversely,

when the player presses the capacitor after a low pitch sound is played, the player loses a point. The player that reaches five points first is deemed the winner by the program.

In “LaunchPad", a player is allowed to press any of the five keys to generate a sound. This sound has been fine-tuned so that each capacitor represents a note in a piano, and upon pressing Key2 or Key3 (P0.6), the octave is set to change. To maximize realism, the keys are set to respond in real-time with the user. In other words, as long as the player continues to press on the capacitor, the note will continue. Similarly, as the player lets go, the note stops, just like with an actual piano.

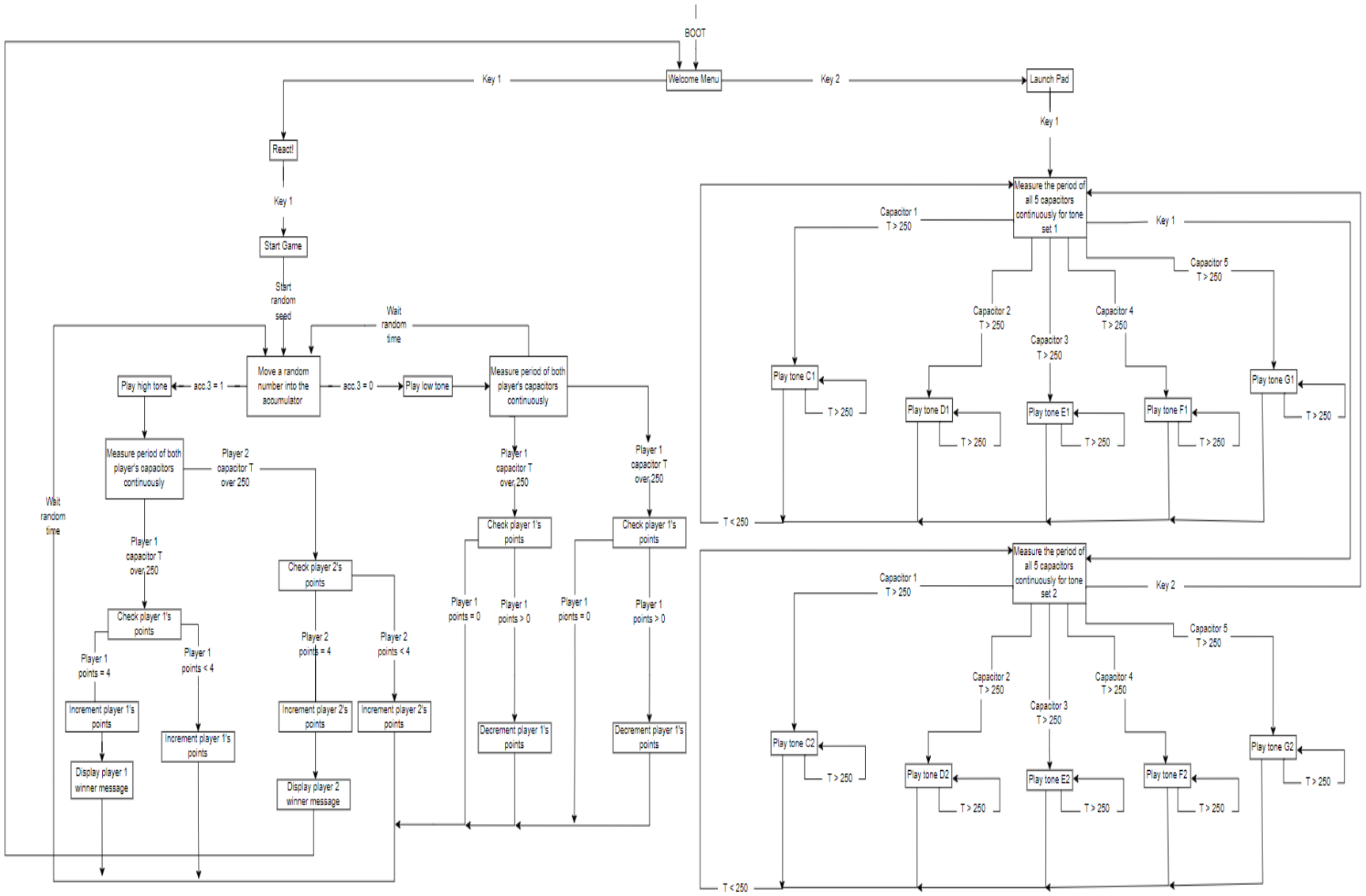
1. Simplified Circuit Diagram for the Game Interface (Multisim)



*Special Note:

The circuit diagram is not fully complete. Due to spacing issues, only two of the five 555 timers used in the project have been shown. Two of the six push-buttons used are modeled in the diagram. This is because the components and connections between the timers and buttons are identical. Improved readability and reduced wiring clutter are possible by showing this abridged version of the circuit. The 'Launchpad' keyboard game uses the same circuit.

2. Software Block Diagram



INVESTIGATION

Idea Generation

Going into the project, the team focused on generating as many ideas as possible, regardless of the quality. The team understood the merit of having a multifaceted approach to an issue, with the belief that every idea had the potential to create a working implementation. With this in mind, the individual team members were assigned to come up with at least 3 game ideas that met the criteria. Once this was done, the team drew a brainstorming diagram connecting different factors to each other to determine the most feasible solutions. Among them, some of the major ideas included the “React!” and “Launchpad", but also a “Simon Says” type of game where the player has to press the correct capacitor based on a specific sound that accelerates as the game proceeds.

Investigation Design

In gathering the information with regards to the project, the lecture notes and videos provided prior to the start of the project were mostly used, in addition to some external sources describing various assembly language implementations. For example, in the lecture, it was shown that braided wires increased the overall performance of the capacitors by reducing the noise of the measurement. After the team had conducted a few trials, it was confirmed that braided wires indeed lowered the noise significantly. Therefore, the design was immediately put to use. Furthermore, the team has also gone through various experimental analyses to determine optimal design options. After determining the best fitting capacitor for both our games, the team had a number of differing sizes of capacitors.

In designing the code, the team referenced the sample provided in the prior labs of the course ELEC 291 Winter 2022, including “math.inc” and “LCD4bit.inc”.[1][2]

Data Collection

Due to the nature of this project, the team has not found any significant need for numerical data collection. However, to ensure the functionality and the quality of our capacitor, we had undergone a set of attempts to do this. The team prepared a simple excel spreadsheet that allowed a transfer of data from one to another, and a comparative analysis was conducted upon data was collected.

Data Synthesis

Comparative analysis was done with variously built capacitors. We understood that the material, size and connectivity of the individual capacitors would yield different results, making the comparison recognizable of its quality.

Upon completing the analysis, the team has recognized a pattern in which the output of the capacitors with certain characteristics saw much fewer fluctuations and stability upon human touch. The team has collected such findings and constructed a capacitor that fulfilled our needs to proceed in our initial game design.

Analysis of Results

The motivation for such an analysis was because of our second game, “LaunchPad.” The team has realized without a greatly performing capacitor, that this game would not be in any way possible. While even with mediocre capacitors, the first game might function.

The validity of the result was apparent as the former capacitors did not come close to the later built capacitor with a difference in their error ranging around 40%, with our best one having a 10% error margin in its measurement.

DESIGN

Use of Process

Development was carried out in a systematic and logical manner. Before beginning development for both games, the team created an extensive mind map that contained brainstormed ideas for how to approach the specifications of the project. Identifying errors was a crucial step, which we approached via systematic testing. When developing the ‘React!’ game, we encountered countless bugs while testing, which we approached by testing individual branching. Furthermore, commenting on outlines we were unsure of before testing was a good way to test the code’s influence on the function of the program.[3] When we determined the areas in which bugs occurred, we tried two approaches. The first was attempting to spot syntax or logical errors and then tweak them. If that approach did not work, we would completely rewrite a new section with a different method. This debugging strategy was particularly useful for the ‘React!’ game, as there were many branches we had to pay close attention to.

Need and Constraint Identification

As for most other platforms of games, the team aimed the design of our game to those who desire to be entertained and spend time in a fun and enjoyable manner. In doing this, the team followed a list of expectations that we deemed essential:

- The game must be fully functional. Without any bugs or delays.
- The game must offer some degree of freedom to its users. In our case, allows the users to choose among different games.
- The game must be entertaining.
- The user interface must be straightforward. So that even an 8-year-old could follow its instructions.
- Every capacitor must work to a good degree. (Not randomly trigger the software)

After identifying the needs of potential consumers, the next logical step was to determine programming constraints to construct a comprehensive and user-friendly interface. These include but are not limited to:

- Navigate the menu using a push button.
- Ensure an efficient subroutine to measure capacitance.
- Ensure that available push buttons share a clear purpose and instructions.

Problem Specification

The first component of the project entailed creating a game in which multiple players could interact with touch capacitors linked to 555 timers, in order to play a reaction speed game. The specifications of the project design goals were as follows:

1. The buzzer must randomly generate two different noises
2. Each player must have a scoring system that decrements or increments based on their interaction with the touch capacitors
3. The game must stop once a player reaches the winning number of points
4. The score for each player must increment when they react to the higher-pitched frequency, and decrement when they react to the lower-pitched frequency
5. Only the first player to tap the touch capacitor during the buzzer period will change their score
6. If both players succeed in ignoring a low-frequency pitch, the buzzer will play another random sound after five seconds
7. If the winning sound is played by the buzzer, the round will last until a player taps their capacitor
8. The system must continuously track frequencies in real-time, in order to detect the change in capacitance
9. Each player must have their own 555 timer and touch capacitor

With all of these design goals determined, the team focused first on the harder tasks, such as measuring capacitance in real-time by using a network of flags and logic branches, or by creating “rounds” for the game. Random number generation was a key component, as it created

an element of unpredictability within the program. Wait times, buzzer frequency, as well as buzzer sound duration, were all tied to random seeds.

The second component of the project was the implementation of a working keyboard. This project entailed the usage of multiple touch capacitors, with each linked to a specific 555 timer. If the touch capacitor was touched, its corresponding frequency would change, and the frequency linked to it would be played by the buzzer. The design specifications were:

1. Each capacitor had to have its frequency tracked continuously, and the buzzer would only play if the change in frequency for each capacitor exceeded 20% of its original measured value
2. The 'Launch Pad' was to have two different sets of tones - one for a higher pitch, and one for a lower pitch. They could be toggled using two buttons, and would change the sound played when each capacitor was hit.
3. Make use of multiple 555 timers to add different capacitors or 'notes'
4. The buzzer must stop when a capacitor is not being played

With these specifications, a project outline was then designed and followed to create the piano game, eventually named 'Launchpad'.

Solution Generation

The team followed a strategy of working in phases using a step-by-step approach. For the 'React!' game, the team focused on objectives that were to be completed in a predetermined schedule. For instance, figuring out how to measure capacitance synchronously was the first goal

that was set. This was accomplished by getting the LCD to display the capacitances of the touch capacitor. Once it was obvious that the capacitors were being measured in real-time, the team moved forward to creating the gameplay sequence. When the team created code that did not work as intended, new solutions were tested by using different copies of the program. Synchronous collaboration via visual studio was also utilized, in order to share ideas and test them quickly. This method helped to discover bugs more quickly and was a great way to test many ideas.

Solution Evaluation

The team evaluated the solution by comparing the end result to the self-set criterion. For 'React!', the team evaluated the design by checking if it was able to perform the requirements set and evaluating how accurate and consistent the solution was. Blemishes such as slight instruction delays or glitches were ironed out by streamlining the subroutines and removing extraneous operations. When the program was able to measure capacitance in real-time, continuously track the values of two capacitors, as well as cycle through multiple game states with no delay, the team decided that the implemented solution for the 'React!' game was complete and successful.

For 'Launchpad', the objective was to ensure that the buzzer was able to play each note perfectly, cycle between each octave with pushbuttons, and operate with no delay when the touch capacitors were interacted with. We used a frequency conversion website to achieve this.[4] As the team's implemented solution was able to switch between pitch modes with two input pushbuttons, the first requirement was fulfilled. The program's ability to track synchronous capacitance changes, and play its mapped tone simultaneously was a great accomplishment,

satisfying the most important requirement of the game. In addition, the lack of delays of any kind led the team to evaluate the program as a successful implementation.

Detailed Design

The building blocks of the design are separated into three, interface, capacitor and game development. The interface covers the overall flow of the game, ensuring bidirectional communication; and the game functionalities cover the overall purpose of the game, motivating the user's attempt to communicate with our design.

The interface is composed of an opening sequence and mode changes. The opening sequence is aimed to maximize the user experience and attempt to leave a good first impression on the user. It is a collection of string calls with adjusted timings that results in a short animation displaying the game menu followed by its device's title (Appendix B.1). Once entered in the menu, the code initializes a flag (Appendix B.2) that is used as the determinant of the mode changes in the latter part of the code. The alleged 'mode' (Appendix B.4, B.5, B.6), is an assembly of functions and jumps that represent different stages which the code is to be in response to different input from the user. This not only allows a contrasting interpretation of the same input but also increases the efficiency of the code concurrently. For example, if the user is at the main menu, the LCD will display the available games and the two push buttons, 1 and 2, will act as a device that determines whether the code would jump to the 'React!'(mode1) (Appendix B.5) or 'Launch Pad'(mode2) (Appendix B.6). However, when the user is at mode2, the LCD changes its display to appropriate texts and the two buttons serve as a means of note changing. Implementing this, one must begin by declaring a top 'loop' which determines the

modes it wants to jump via values changes in ‘a’ (Appendix B.3). The changes in ‘a’ are carried out in the following lines of code known as mode0 (Appendix B.4). Here, the code interpresets the user input from the push button and modifies the value of ‘a’ such that once looped back to the ‘loop’, the code jumps to the correct mode. The gaming interface format runs parallel to the pre-game interface. For instance of mode1, it ensures that the game does not begin unless the button1 is inputed. Once fulfilled, the code proceeds into initializing the random seeds and timer that is needed for sound generation and capacitance measurement, while displaying appropriate texts to to prepare the user. Upon entering the game function (Appendix B.7), the code will call the seeds and detect user touches in capacitors, thus, allowing various jumps to be made, scores to be modified.

In designing capacitors, the team paid great attention to utilizing the best materials available that were locatable in the allotted time. The reason being that the team members understood the physical nature in which having a good capacitor built will not only increase the performance of the game itself but also help the due course of the team's work in coding immensely by letting fewer fluctuations for the team to consider. Therefore, as appendix A.2 shows, we have utilized the least conducting material we could locate, a rubber board, and invested time in making the wires in the most conductive form, by twisting them and allowing a large surface area taped to the board. As per software, the measurement of capacitors utilizes the LM555 chip, measuring the frequency, then converting the numeric values using mathematical operations. Such a process can be seen in the ‘measure_key’ function, in addition to ‘next’ loops that ensure a delay to the measurement, to allow the capacitor to receive inputs for a reasonable time.

The game development involves the use of both the interface and capacitor. However, the reason why such division of parts is made is because the team feels it is not only undemanding for the readers to understand our overall design, but also, it is how the team approached both planning and creating the game. There are in total two games present in the design, 'React!' and 'Launch Pad'. The first game, 'React!' utilizes the already established functions. In general, the game flow is as follows: first, the code enters according to mode once the user presses the matching button (Appendix B.3), then the game prepares the user by displaying another screen waiting for the user's input. Once the button is pressed, it immediately initializes all the necessary items such as timer, counter and seeds (Appendix B.5). Next, the actual game function calls all the required functions within the code such as 'random', 'measure capacitor' and at the same time carries out the modification of the player scores (Appendix B.7). The 'tone2' function uses the 0 timer to control the sound according to the random seed it accepts, thus, making sure that the generated sound cannot be anticipated by any of the two players.

The second game, 'Launch Pad' involves a more rigorous repetition of the code, as it tries to establish a note for each capacitor, and going beyond that, the team has successfully implemented an octave changing function within the design. Similar to the first game, the code enters the according mode once the user presses the matching button (Appendix B.3), then the game prepares the user by displaying another screen waiting for the user's input. Once the button is pressed, it goes into another looping mechanism that sets the code in such a way that a button pressed will always be considered while the game is being played (Appendix B.8). While in such a loop, the code measures each capacitor individually and once a measurement is detected, it

jumps to the according-to sound function that generates a specific sound. Each known sound, or in this case, a note, inhabits its individual function (Appendix B.9) and establishing an infinite loop by jumping back to the base function of the game, allows a live transition of notes. Extra functions such as C2_11 and F_2111 (Appendix B.8), are used to compensate for the 'exceeds -128/+128' error.

The hardware of our design does not include an extreme compilation. Referring to Appendix A.2, we can observe it consists of the processor and 5 LM555 chips that represent the number of capacitors that are in use. Different capacitors are connected to various ports of the processor, which the team could fetch to our needs. Also, for sound generation, speakers can be observed at the top left side of the board.

Solution Assessment

To ensure stability and longevity of the design, the team has assessed the solution by breaking down the individual components, in an attempt to minimize the margin of errors in each of their performances and confirm that each met the requirements and needs of our stakeholders.

The teams have undergone a beta-testing strategy for the interface of the game. Where one goes through various ways a user might interact with the game, and assess the performance at each stage, making sure that there are no loopholes that break, stop, or exit the game. Doing this test, our team has established the game's ability to return all the user's calls.

With regards to capacitors, the team went through a repertoire of measurements to ensure the stability of the capacitor values. To do this, the team constructed a spreadsheet of recorded data and compared various recordings in comparison to other capacitors that one might consider better built. In doing so, the team established the level of stability of our capacitors and acquired a reasonable threshold value that made the integration between the capacitor and the software possible.

For the assessment of the two games, the team focused on going through a detailed analysis of the overall responsiveness. This combines and goes beyond the two prior assessments by making sure that the end product of the two components is exceptional, and at the same time, delivers the best user experience possible. With respect to the ‘React!’, the team conducted a simple test to discover which interval of sounds makes the game feel utmost entertained. We discovered that the longer and the more frequent the intervals between the sounds are, the better the user experience. With respect to the ‘Launch Pad’, the team conducted a reactivity

LIFE-LONG LEARNING

During this project, our team has benefited from our prior knowledge in assembly code that was introduced in CPEN 211. However, in various instances, we had to make use of instructions that were beyond the course’s scope, thus, requiring us to go through a rigorous trial and error process to successfully implement the desired outcome. Furthermore, we also expanded our overall skill in assembly coding by having to utilize different sets of loops that made our game more interactive with the user. Especially, in making the user interface to allow the user to choose between the two games, also, making two alternating sets of notes in our so-called ‘piano

game'. Looking back, a more fundamental understanding and proficiency in assembly code would have helped immensely.

The countless methods on how we implemented various functionalities have helped us gain a broader understanding of how certain features of 8051 assembly function, such as the 'JB' and 'JNB' operations, and usage of flags to check program status. Trial and error methodology has helped the team practice using an array of different creative approaches to solving the problem, an invaluable experience for learning.

CONCLUSION

The team's design processes culminated in the creation of a complete and user-friendly game interface. The ability to choose between two game modes, with the ability to play with many others, was a key objective that was satisfied in this project. The 'React!' game mode successfully presents two or more players a reaction time and judgment-based game, where games would go on until one player wins. The 'Launchpad' game mode offers the user a more relaxing and creative experience, where the user is free to play any note they would like, with the possibility of composing music. Overall, the team was very satisfied with how the project turned out, and every key milestone and objective was fulfilled.

Throughout the extensive design process, the team encountered many difficulties. Countless bugs and syntax errors forced a constant re-evaluation of coding rules, as well as long periods of brainstorming. For instance, when developing the program for 'Launchpad', a consistent error was the inability to track synchronous capacitance changes, and the ability to

change pitch modes using the pushbuttons. Through extensive testing of ideas the team generated, both problems were eventually solved. The challenging nature of the project encouraged the team to work together to come up with creative solutions.

Retrospectively, the team agreed that if another project like this were to be taken, something that would be done differently would be to identify potential areas of failure well in advance, so as to avoid hours of debugging.

In total, the project took the team a commitment of 28 hours, and it was a thoroughly enjoyable and enlightening experience. Every member learned a lot more about assembly language and circuit design.

REFERENCES

[1] Dr. J. Calvino-Fraga, “math.inc” Vancouver .

[2] Dr. J. Calvino-Fraga, “LCD4Bit.inc” Vancouver .

[3] 8051 Instruction Set. 8051 instruction set. (n.d.). Retrieved February 15, 2022, from https://www.win.tue.nl/~aeb/comp/8051/set8051.html

[4] “Tuning,” Frequencies of Musical Notes, A4 = 440 Hz. [Online]. Available: https://pages.mtu.edu/~suits/notefreqs.html. [Accessed: 04-Mar-2022].

[5] “MyMicrochip.” [Online]. Available: https://www.microchip.com/en-us/product/AT89LP51RC2. [Accessed: 16-Feb-2022].

[6] D. Yuhas, “Speedy science: How fast can you react?,” *Scientific American*, 24-May-2012. [Online]. Available: https://www.scientificamerican.com/article/bring-science-home-reaction-time/. [Accessed: 10-Feb-2022].

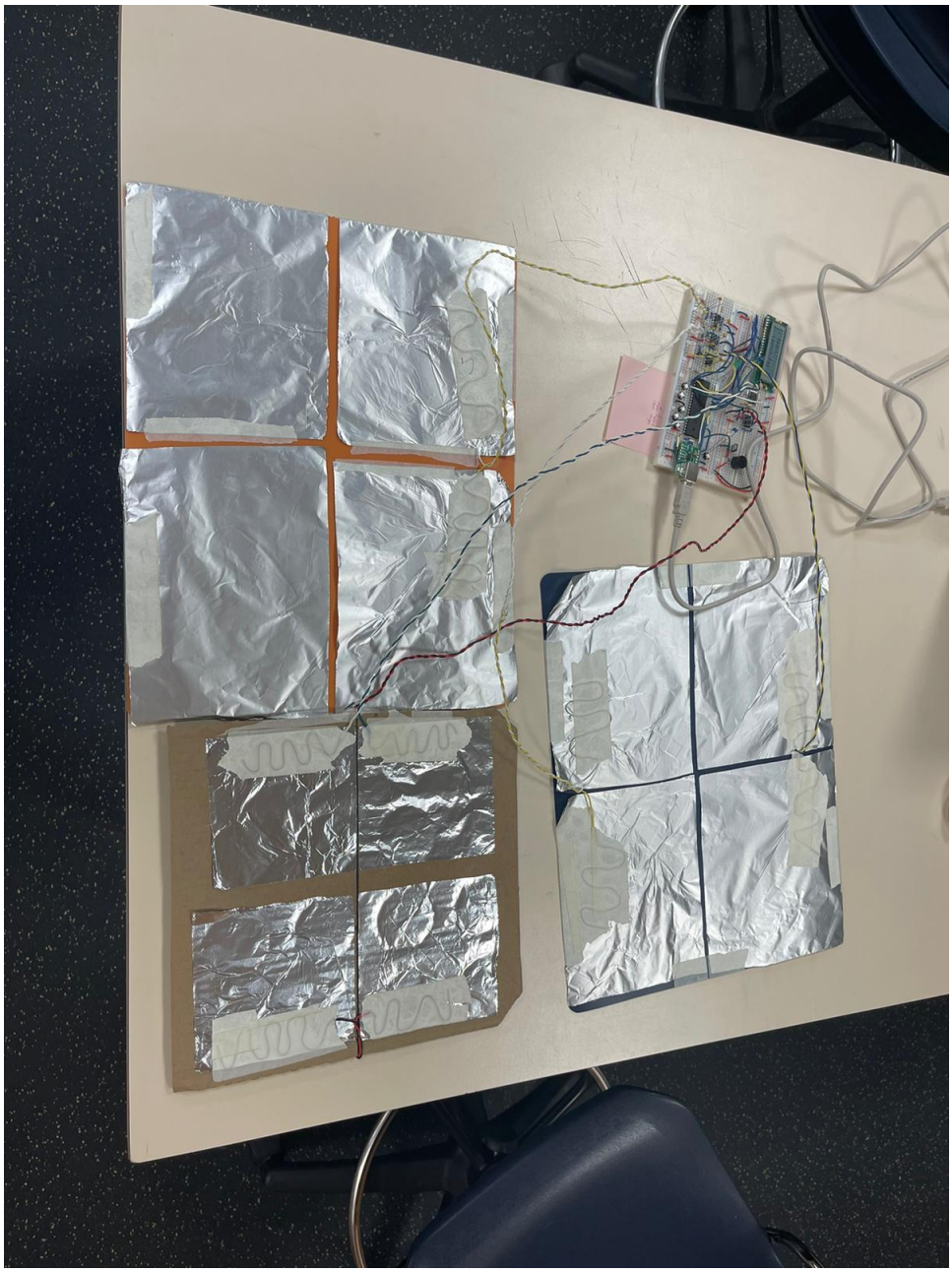
BIBLIOGRAPHY

- **"Embedded system interrupts in 8051 microcontroller - javatpoint,"** ***www.javatpoint.com*****. [Online]. Available: https://www.javatpoint.com/embedded-system-interrupts-in-8051-microcontroller. [Accessed: 14-Feb-2022].**
- **"MIT - Massachusetts Institute of Technology." [Online]. Available: http://web.mit.edu/6.115/www/document/8051.pdf. [Accessed: 15-Feb-2022].**
- **"555 timer tutorial - the monostable multivibrator,"** ***Basic Electronics Tutorials*****, 23-Apr-2020. [Online]. Available: https://www.electronics-tutorials.ws/waveforms/555_timer.html. [Accessed: 15-Feb-2022].**

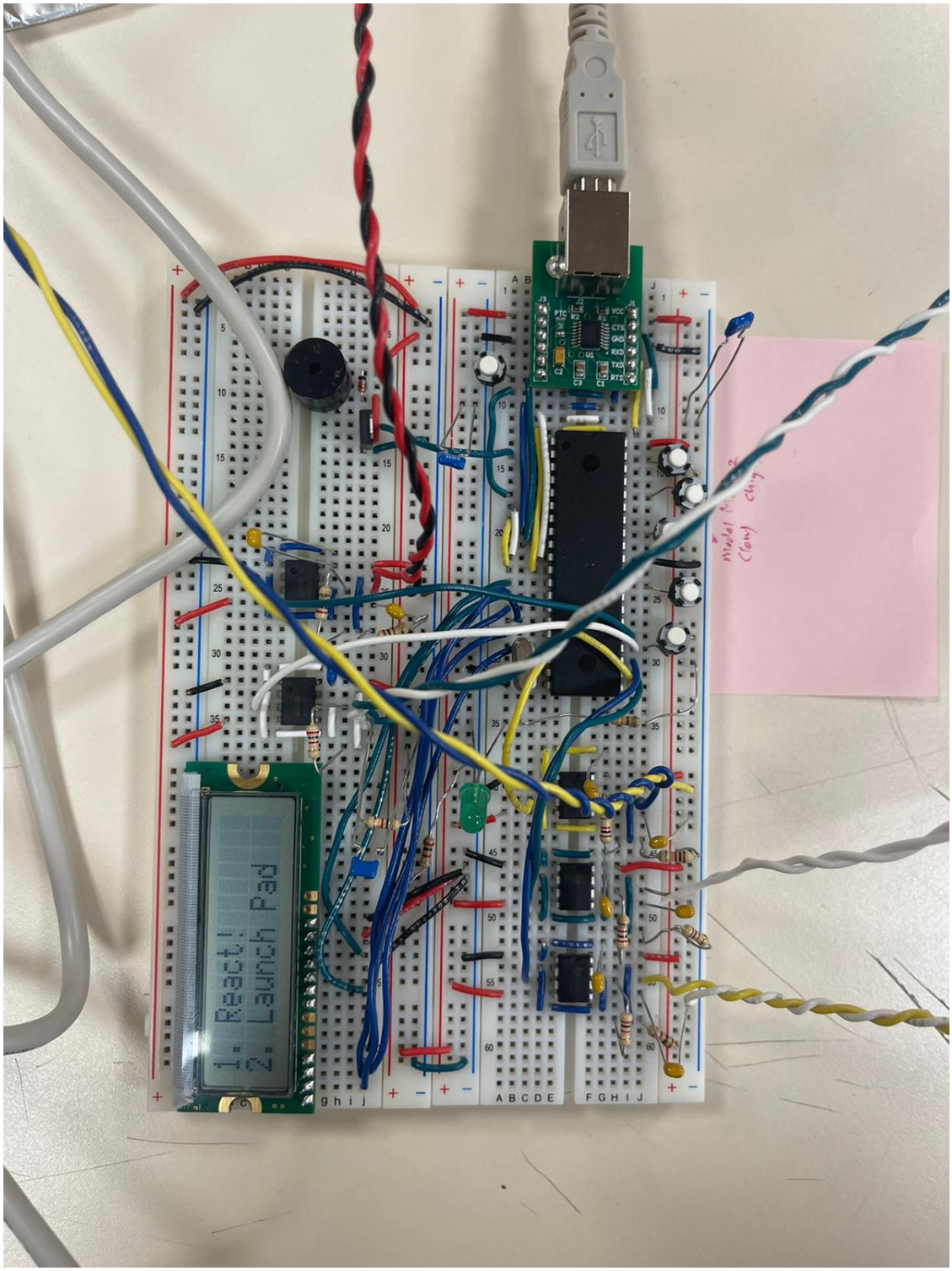
APPENDICES

APPENDIX A (Hardware)

1. Picture of Capacitor Setup



2. Breadboard Circuit with User Interface on LCD



APPENDIX B (Software)

1. Opening Sequence

```
       lcall intro

    mov R2, #50       ;Wait
       lcall WaitSec

       lcall blanks

       mov R2, #50    ;Wait
       lcall WaitSec

       Set_Cursor(1, 1)
    Send_Constant_String(#menu0)
       Set_Cursor(2, 1)
       Send_Constant_String(#menu1)

       mov R2, #250   ;Wait
       lcall WaitSec
```

2. Flag

```
    setb    tick_flag
    ; set mode
    mov        mode,                #0x00
    ; initialize time
    mov     a,  #0x00
```

3.Loop

```
loop:
    clr        c
    mov        a,  mode
    jz         mode0                ; if mode == 0
    subb       a,      #0x01
    jnz        loop_notMode1  ; if mode == 1
    ljmp       mode1
loop_notMode1:
    clr     c
    mov        a,  mode
    subb       a,      #0x02
    jnz        loop_notMode2   ; if mode == 2
    ljmp    mode2

loop_notMode2:
       ; reset mode back to 0
       mov            a,             #0x00
       mov            mode,   a
    ljmp    mode0_d
```

4. Mode0

```
mode0:
    jb      BUTTON_1, mode0_a
    Wait_Milli_Seconds(#DEBOUNCE_DELAY)
    jb         BUTTON_1, mode0_a
    jnb        BUTTON_1, $

    ; button 1 is pressed here (goto mode 1)
```

```
    ; setup screen
    Set_Cursor(1, 1)
    Send_Constant_String(#game0)
    Set_Cursor(2, 1)
    Send_Constant_String(#game1)

    ; change mode
    mov     a,      #0x01
    mov     mode,   a
    ljmp       mode0_d


mode0_a:
    jb      BUTTON_2, mode0_d
    Wait_Milli_Seconds(#DEBOUNCE_DELAY)
    jb         BUTTON_2, mode0_d
    jnb        BUTTON_2, $

    ; button 2 is pressed here (goto mode 2)
    ; setup screen
    Set_Cursor(1, 1)
    Send_Constant_String(#game0)
    Set_Cursor(2, 1)
    Send_Constant_String(#game1)
    ; change mode
    mov     a,      #0x02
    mov     mode,   a
    ljmp       mode0_d

mode0_d:
    clr        tick_flag ; We clear this flag in the main ; display every second
    ljmp    loop
```

5. Mode1

```
mode1:
        jb      BUTTON_BOOT,    mode1_a
    Wait_Milli_Seconds(#DEBOUNCE_DELAY)
    jb      BUTTON_BOOT,    mode1_a
    jnb     BUTTON_BOOT,    $

    Set_Cursor(1, 1)
    Send_Constant_String(#game0)
    Set_Cursor(2, 1)
    Send_Constant_String(#game1)

    mov     a,      #0x00
    mov     mode,   a
    ljmp    mode1_d


mode1_a:
        jb      BUTTON_1,    mode1_b
    Wait_Milli_Seconds(#DEBOUNCE_DELAY)
    jb      BUTTON_1,    mode1_b
    jnb     BUTTON_1,    $

        ljmp mode1_d

loop_1:
        ljmp loop
```

```
mode1_b:
       Set_Cursor(1,1)
    Send_Constant_String(#react)
       Set_Cursor(2,1)
    Send_Constant_String(#start)

mode1_d:
       jb P0.0, loop_1

       ;---------As button 1 is pushed, it runs the code below
       WriteCommand(#0x01)

    mov R2, #50       ;Wait
       lcall WaitSec

       mov p1_ctr, #0x00
       mov p2_ctr, #0x00
       Set_Cursor(1,1)
    Send_Constant_String(#player1)
       Set_Cursor(2,1)
    Send_Constant_String(#player2)
       Set_Cursor(1,11)
    Display_BCD(p1_ctr)
    Set_Cursor(2,11)
    Display_BCD(p2_ctr)
    lcall InitTimer2

    mov R2, #180      ;Wait
       lcall WaitSec

       clr TR0
       mov Seed+0, TH0
       mov Seed+1, #0x01
       mov Seed+2, #0x87
       mov Seed+3, TL0
       sjmp react1
```

6. Mode2

```
mode2:
       jb      BUTTON_BOOT,    mode2_a
    Wait_Milli_Seconds(#DEBOUNCE_DELAY)
    jb      BUTTON_BOOT,    mode2_a
    jnb     BUTTON_BOOT,    $

    Set_Cursor(1, 1)
    Send_Constant_String(#game0)
    Set_Cursor(2, 1)
    Send_Constant_String(#game1)

    mov     a,      #0x00
    mov     mode,   a
    ljmp    mode1_d


mode2_a:
       jb      BUTTON_1,    mode2_b
    Wait_Milli_Seconds(#DEBOUNCE_DELAY)
    jb      BUTTON_1,    mode2_b
    jnb     BUTTON_1,    $
```

```
       ljmp mode2_d

mode2_b:
       Set_Cursor(1,1)
    Send_Constant_String(#piano)
       Set_Cursor(2,1)
    Send_Constant_String(#start)

mode2_d:
       jb P0.0, loop_2

       ;---------As button 1 is pushed, it runs the code below
       WriteCommand(#0x01)

    mov R2, #50       ;Wait
       lcall WaitSec

       Set_Cursor(1,1)
    Send_Constant_String(#changekey)


    mov R2, #180      ;Wait
       lcall WaitSec
       ljmp piano1
```

7. React!

```
tone22:
       ljmp tone2

react1:
       lcall Random
       mov a, Seed+1
       mov c, acc.3
    jc tone22 ; jump to tone2 if c = acc.3 is 1, else run the code below. Thus making 2
different sounds.

    clr TR0
       mov RH0, #high(TIMER0_RELOAD1)
       mov RL0, #low(TIMER0_RELOAD1)
       setb TR0
    lcall wait_milli_random
    clr TR0


    ;--------------------------detect and increment/decrement here
       mov dec_ctr, #0
       ;lcall hex2bcd2
    lcall measure_cap2
       ljmp react1

tone2:
       lcall Random
       mov a, Seed+2

       clr TR0
       mov RH0, #high(TIMER0_RELOAD0)
       mov RL0, #low(TIMER0_RELOAD0)
       setb TR0
```

```
    lcall wait_milli_random
    clr TR0


       ;-------------------------detect and increment/decrement here
       lcall measure_cap
    ;ljmp react1 ; Repeat!
```

8. Launch Pad

```
loop_2:
       ljmp loop

piano1:
       Set_Cursor(2,1)
    Send_Constant_String(#Key1)
    ;-------------------------Capacitor sound here
       lcall measure_key

piano2:
       Set_Cursor(2,1)
    Send_Constant_String(#Key2)

       lcall measure_key_2

measure_key_2:
    jnb P0.6, piano1
       ; Measure the period applied to pin P2.0
       clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.0, $
    jnb P2.0, $
    setb TR2 ; Start counter 0
    jb P2.0, $
    jnb P2.0, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, C2_11
       sjmp D2_11
C2_11:
       ljmp C_2
       ;Detect Key2
D2_11:
       clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.1, $
    jnb P2.1, $
```

```
    setb TR2 ; Start counter 0
    jb P2.1, $
    jnb P2.1, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
    jb mf, D_2111
       sjmp E_211
D_2111:
       ljmp D_2
E_211:
       clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.2, $
    jnb P2.2, $
    setb TR2 ; Start counter 0
    jb P2.2, $
    jnb P2.2, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, E_2111
       sjmp F_211
E_2111:
       ljmp E_2
F_211:
    clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.3, $
    jnb P2.3, $
    setb TR2 ; Start counter 0
    jb P2.3, $
    jnb P2.3, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
```

```
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, F_2111
       sjmp G_211
F_2111:
       ljmp F_2
G_211:
       clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.4, $
    jnb P2.4, $
    setb TR2 ; Start counter 0
    jb P2.4, $
    jnb P2.4, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
    jb mf, G_2111
       sjmp C_31
G_2111:
       ljmp G_2
C_31:
       ljmp measure_key_2



piano22:
       ljmp piano2
measure_key:
    jnb P0.3, piano22
       ; Measure the period applied to pin P2.0
    clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.0, $
    jnb P2.0, $
    setb TR2 ; Start counter 0
    jb P2.0, $
    jnb P2.0, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
```

```
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, C_11
       sjmp D_11
C_11:
       ljmp C_1
       ;Detect Key2
D_11:
       clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.1, $
    jnb P2.1, $
    setb TR2 ; Start counter 0
    jb P2.1, $
    jnb P2.1, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
    jb mf, D_111
       sjmp E_11
D_111:
       ljmp D_1
E_11:
       clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.2, $
    jnb P2.2, $
    setb TR2 ; Start counter 0
    jb P2.2, $
    jnb P2.2, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
```

```
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, E_111
       sjmp F_11
E_111:
       ljmp E_1
F_11:
    clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.3, $
    jnb P2.3, $
    setb TR2 ; Start counter 0
    jb P2.3, $
    jnb P2.3, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, F_111
       sjmp G_11
F_111:
       ljmp F_1
G_11:
clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    jb P2.4, $
    jnb P2.4, $
    setb TR2 ; Start counter 0
    jb P2.4, $
    jnb P2.4, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, G_111
       sjmp C_21
G_111:
       ljmp G_1
```

```
C_21:
	ljmp measure_key
```

9. Notes

```
C_1:
	lcall WaitSec_1
	clr TR0
	mov RH0, #high(C__1)
	mov RL0, #low(C__1)
	setb TR0

    mov TL2, #0
    mov TH2, #0
    jb P2.0, $
    jnb P2.0, $
    setb TR2 ; Start counter 0
    jb P2.0, $
    jnb P2.0, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
	lcall hex2bcd2
	Set_Cursor(1,13)
    jb mf, C_1
	clr TR0

	ljmp piano1

D_1:
	lcall WaitSec_1
	clr TR0
	mov RH0, #high(D__1)
	mov RL0, #low(D__1)
	setb TR0

    mov TL2, #0
    mov TH2, #0
    jb P2.1, $
    jnb P2.1, $
    setb TR2 ; Start counter 0
    jb P2.1, $
    jnb P2.1, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
```

```
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, D_1
       clr TR0

       ljmp piano1

E_1:
       lcall WaitSec_1
       clr TR0
       mov RH0, #high(E__1)
       mov RL0, #low(E__1)
       setb TR0

    mov TL2, #0
    mov TH2, #0
    jb P2.2, $
    jnb P2.2, $
    setb TR2 ; Start counter 0
    jb P2.2, $
    jnb P2.2, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, E_1
       clr TR0

       ljmp piano1

F_1:
       lcall WaitSec_1
       clr TR0
       mov RH0, #high(F__1)
       mov RL0, #low(F__1)
       setb TR0

    mov TL2, #0
    mov TH2, #0
    jb P2.3, $
    jnb P2.3, $
    setb TR2 ; Start counter 0
    jb P2.3, $
    jnb P2.3, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
```

```
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, F_1
       clr TR0

       ljmp piano1

G_1:
       lcall WaitSec_1
       clr TR0
       mov RH0, #high(G__1)
       mov RL0, #low(G__1)
       setb TR0

    mov TL2, #0
    mov TH2, #0
    jb P2.4, $
    jnb P2.4, $
    setb TR2 ; Start counter 0
    jb P2.4, $
    jnb P2.4, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, G_1
       clr TR0

       ljmp piano1

C_2:
       lcall WaitSec_1
       clr TR0
       mov RH0, #high(C__2)
       mov RL0, #low(C__2)
       setb TR0

    mov TL2, #0
    mov TH2, #0
    jb P2.0, $
    jnb P2.0, $
    setb TR2 ; Start counter 0
    jb P2.0, $
```

```
    jnb P2.0, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, C_2
       clr TR0

       ljmp piano2

D_2:
       lcall WaitSec_1
       clr TR0
       mov RH0, #high(D__2)
       mov RL0, #low(D__2)
       setb TR0

    mov TL2, #0
    mov TH2, #0
    jb P2.1, $
    jnb P2.1, $
    setb TR2 ; Start counter 0
    jb P2.1, $
    jnb P2.1, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
    jb mf, D_2
       clr TR0

       ljmp piano2

E_2:
       lcall WaitSec_1
       clr TR0
       mov RH0, #high(E__2)
       mov RL0, #low(E__2)
       setb TR0

    mov TL2, #0
    mov TH2, #0
```

```
    jb P2.2, $
    jnb P2.2, $
    setb TR2 ; Start counter 0
    jb P2.2, $
    jnb P2.2, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, E_2
       clr TR0

       ljmp piano2

F_2:
       lcall WaitSec_1
       clr TR0
       mov RH0, #high(F__2)
       mov RL0, #low(F__2)
       setb TR0

    mov TL2, #0
    mov TH2, #0
    jb P2.3, $
    jnb P2.3, $
    setb TR2 ; Start counter 0
    jb P2.3, $
    jnb P2.3, $
    clr TR2 ; Stop counter 2, TH2-TL2 has the period
    ; save the period of P2.0 for later use
    mov period1+0, TL2
    mov period1+1, TH2
    Set_Cursor(1,1)
    lcall hex2bcd2
    mov x+0, period1+0
    mov x+1, period1+1
    mov x+2, #0
    mov x+3, #0
    load_y(175)
    lcall x_gt_y
       lcall hex2bcd2
       Set_Cursor(1,13)
    jb mf, F_2
       clr TR0

       ljmp piano2

G_2:
       lcall WaitSec_1
       clr TR0
       mov RH0, #high(G__2)
```

```
    mov RL0, #low(G__2)
    setb TR0

mov TL2, #0
mov TH2, #0
jb P2.4, $
jnb P2.4, $
setb TR2 ; Start counter 0
jb P2.4, $
jnb P2.4, $
clr TR2 ; Stop counter 2, TH2-TL2 has the period
; save the period of P2.0 for later use
mov period1+0, TL2
mov period1+1, TH2
Set_Cursor(1,1)
lcall hex2bcd2
mov x+0, period1+0
mov x+1, period1+1
mov x+2, #0
mov x+3, #0
load_y(175)
lcall x_gt_y
    lcall hex2bcd2
    Set_Cursor(1,13)
jb mf, G_2
    clr TR0

    ljmp piano2
```