Project 1

Capacitive Sensor Reaction Game

Student #	Student Name	% Point	Signature
24042467	Avya Malhotra	100	AM
25550245	Kieran Ross	100	KR
15576333	Oliver Kis	100	OK

University of British Columbia
Electrical and Computer Engineering
ELEC 291 - Electrical Engineering Design Studio I
Section 201 & L2A/L2B winter 2022
Instructor: Dr. Jesus Calvino-Fraga P.Eng.

Date of Submission: March 4, 2022

Table of Contents

I. Introduction

For project 1, our given goal was to design, build, and program a competitive capacitive sensors-based reaction game. The key objectives were to use the 8051 microcontrollers, build at least two capacitive sensors, use A-stable oscillators, produce sound with the given speaker, keep a record of points with the LCD, build the program from the assembly, ensure the sensitivity is accurate, and to produce a 2100Hz and 2000 Hz tone. For hardware, the initial approach had two parts. First, the hardware includes the AT89LP51RC2 Microcontroller and the BO230XS USB adapter. The buttons, SD1602H LCD, four NE555 timers, and the four sensors were implemented. Second, the physical boxes design includes plywood and 2x2s. After spray painting and drilling holes for the wiring, the Microcontroller setup and the box are assembled.

For software, the process is a lot more intricate. We load the microcontroller, boot, and reset starting at the beginning. Our program is sequential, starting at the top. Our speaker plays the Mario tune, and the game begins. We use a random number generator to randomize the timing of the tones and the frequencies. The tone is played, and during the tone, each sensor's period is measured, converted to capacitance, and determined which sensor hit its threshold first. If nobody hits a sensor, the program loops back to random number generation. Multiple flags are set if somebody crossed their threshold, checking if a hit was detected. Depending on the frequency, the fastest person will either gain or lose a point if a hit is detected. When the first person reaches 5 points, the game speeds up, and subsequently, when somebody reaches 10 points, the game speeds up again. Lastly, the first person to reach 15 points wins the game.

II. Investigation

a. Idea Generation

Ideas to accomplish project requirements, challenges, and additional features were the foundation of a successful project. Thus, the idea generation stage was revisited constantly throughout the project's timeline. The project's requirements were divided into smaller goals that needed to be accomplished, which would then be compiled as a coherent product. Broadly these divisions included but were not limited to a) sound generation, b) capacitor plates, c) hit detection, d) randomization and e) UI/UX and aesthetics. Each division called for the constant need for idea generation. Ideas were evaluated based on creativity, feasibility, and necessity. When problems were faced, such as the interference of capacitance between four players or the issue of capacitance detection while the program was halted due to initial methods of sound generation, it was the abundance of practical ideas pitched by the team that allowed us to fix these issues.

Although implementing requirements was a priority, additional generation of ideas was required since additional functionality and creativity were a significant portion of the available grade. Several ideas were discussed, voted on, and implemented at different project stages. For example, while finalizing the sound generation stage, it was unanimously agreed upon to implement different tones, which led to the development of the "*Mario Theme*" music being played each time the game started. During implementing randomization of the tone played and wait times between each point, several ideas were brainstormed to utilize seed generation to increase the game's difficulty. We settled on multiple rounds, where the random wait time between tones would be reduced each round. Similarly, we decided to implement a four-player version of the game towards the end of the completion of the project. The choice to considerably improve aesthetics, moving beyond the initial crude build to a robust, commercially viable product, was a boon of the idea generation stage.

b. Investigation Design

For Data gathering, we utilized Lab #1, which taught us how to connect the LCD and alter the display. Next, we used code from Lab #2, which taught us to connect the speaker, use flags, and set the speaker's frequency. Lab #3 taught us how to do arithmetic in 8051 assembly, set up 555-oscillating timers, and read capacitance from the 555 timer square waves. In addition, we utilized Piazza as pieces of code were published occasionally. Pulling code from each previous lab and Piazza, we documented what each portion of code did and how it affected future code. We left comments and drew flow charts to increase clarity. Our final step was experimenting with our code. We made sure we did not have compilation errors, uploaded the code to our hardware which contained a speaker, LCD, NE555 timer to detect capacitance, and tested each feature. Piece by piece, we continued to add to the functionality of our board until we had capacitive sensors and were able to play the game.

c. Data Collection

With the project being qualitative, the amount of necessary data we accrued was minimal. However, information such as the project requirements, ideas to implement those requirements, and interesting shortcuts were stored collectively to ensure we effectively met the minimum requirements. Also, through trial and error, our observations were documented on what worked and what needed to be fixed. To store all this information, we used a program called Notion. As a result, we could store information regarding requirements, each rendition of code, hardware designs and collect additional feature designs in one place. Furthermore, in Notion, we carefully labeled each new section; in return, we could quickly revisit older ideas if need be. In addition, Notion allowed us to upload our updated code for each member to use in real time.

d. Data Synthesis

Our team synthesized data with VSCode and Notion as we are all able to cooperatively write on VSCode and store any critical sections of code on Notion. To reach conclusions, we typically presented our ideas and voted on them, especially with bonus features and the physical design of the project. Since our group worked in person, we utilized group meetings to finalize our ideas.

e. Analysis of Results

We came to conclusions regarding the validity of ideas by appraising the difficulty of the idea, how much time we could allot for implementing the idea, and how many people would be required to research and contribute towards the idea. Since this project was a combination of the three previous labs, most material for the base requirements were taught. However, we used some techniques that required additional research for bonus features. We used the same technique for each bonus feature.

III. Design

a. Use of Process

We utilized the seven-step engineering design process. During our first meeting, we defined the problem, listed objectives specifications, and broke the project down into smaller components like pieces of a puzzle. We then set out to identify resources and conduct research. We researched the constraints of assembly language and the microcontroller we were utilizing. Next, we listed parts that we needed as a team to implement all the planned functionality. We divided tasks between members based on strength but always allowed members less confident in certain areas to work with those stronger in said areas to allow for novel ideas and collective

growth. We then brainstormed possible solutions to our goals, including planning out different sections to the software and hardware requirements. Each section was created and tested for edge cases, bugs, and general improvements. The construction of a complete prototype with basic requirements was our next goal. After several cycles of evaluation and interaction, we arrived at our final product.

b. Need and Constraint Identification

A microcontroller-based capacitive sensor reaction game was to be built by our group. The project was restricted to an 8051 based microcontroller, preferably the AT89LP51RC2 microcontroller. The capacitive sensors were hand-built using household materials, preferably utilizing aluminum between two sheets of transparent plastic. The change in capacitance was measured using A-stable oscillators, preferably built using the 555 timer circuit. The game would be required to utilize both the CEM-1028 mini speaker and the LCM-201602DTR/M Liquid Crystal Display (LCD). All programming was to be completed in assembly. The capacitive sensors were to reliably and quickly detect a hand on top of them. As for sound generation, either a 2100hz or 2000hz tone was produced randomly. A high tone would result in a point increment, while a low tone would result in a point decrement; any other rules were up to the teams' discretion.

c. Problem Specification

Several additional design requirements were specified based on needs and constraints. Since the capacitance sensors were hand-built, we utilized robust materials to ensure the game would not break down under constant play. Since 555 timer circuits were to be utilized for each

player. We limited each breadboard to contain not more than two timer circuits as this would increase interference and cause the game to be unplayable due to ever-changing thresholds. Since we were restricted to the CEM-1028 speaker, additional functionality such as playing music at the start of each game would be made possible only by breaking down melody into individual notes played with time intervals instead of a .mp3 file, a prime example of design requirement based on given constraints. Furthermore, it was a tricky yet necessary design requirement to plan out how details of the score of all four players would fit in a 2x16 LCD.

d. Solution Generation

After carefully laying out every aspect of the problem at hand, we brainstormed ideas regardless of project requirements and specifications. This method allowed us to assemble a large assortment of ideas that we could choose from.

Concerning the capacitor plate design and material, several solutions were considered:

1. Aluminum cut from soda cans would result in increased durability for the product, while also being more complex and more hazardous due to the sharp edges.
2. Sourcing the materials for the plates from aluminum tape was another possibility discussed in class. However, not much information was known about them and their reliability. More testing was required.
3. The final method suggested while brainstorming was the use of aluminum foil. It is a very malleable material to work with but poses an issue of equipment longevity.

Our initial array of solutions included some soon-abandoned ideas, such as songs being played after every point and round-change animations.

e. Solution Evaluation

While mapping out the project timeline, we worked on prototyping and testing code-blocks individually to see if a given feature would be a good fit for our game. Eventually, we concluded on our final design due to fierce testing, programmer capabilities, and time constraints. We opted for a detailed and slick protective box for our inner circuitry with four highly responsive yet durable capacitance detection pads.

To ensure the high quality of our capacitive plates, we ended up deciding to use aluminum foil due to its straightforward and risk-free nature. We aimed to solve the issue of durability previously mentioned in three ways. First, we guaranteed their resilience by wrapping them in airtight plastic, giving them a solid cardboard backbone, and fastening them to the box with highly adhesive electrical tape.

The introduction of music upon starting the system proved to be much more of a challenge, so we elected to polish the rest of our project as opposed to integrating different songs and sounds on the accrual or deduction of points. We considered exchanging the wait time between each point for a waiting song of a predetermined length, but in doing so, we would lose the randomness between each round.

We knew that our capacitance pads would be incurring heavy damage, so a sturdy and long-lasting design was necessary. Therefore, the frame was made from woodshop-grade timber constructed with a hollowed circuitry cradle inside.

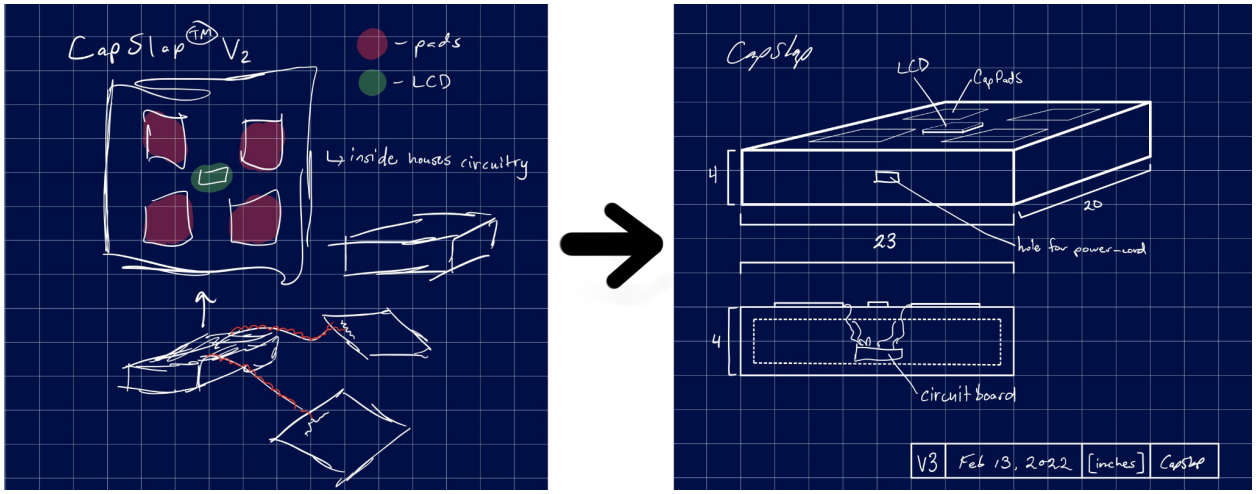

Figure 1. *Design Schematic Evolution*

f. Safety/Professionalism

Unlike previous years, we were not using any electrical design labs. Instead, we met in person at each others' homes. We did not use heavy-duty equipment; however, when we worked on the circuitry, we wore safety glasses in case of an emergency. In addition, when the box was spray painted, we used a well-ventilated area.

To maintain professionalism, each team member respected and supported each other. We met on time and made sure we set team boundaries. Finally, each code source used was cited and adequately managed.

g. Detailed Design

Hardware

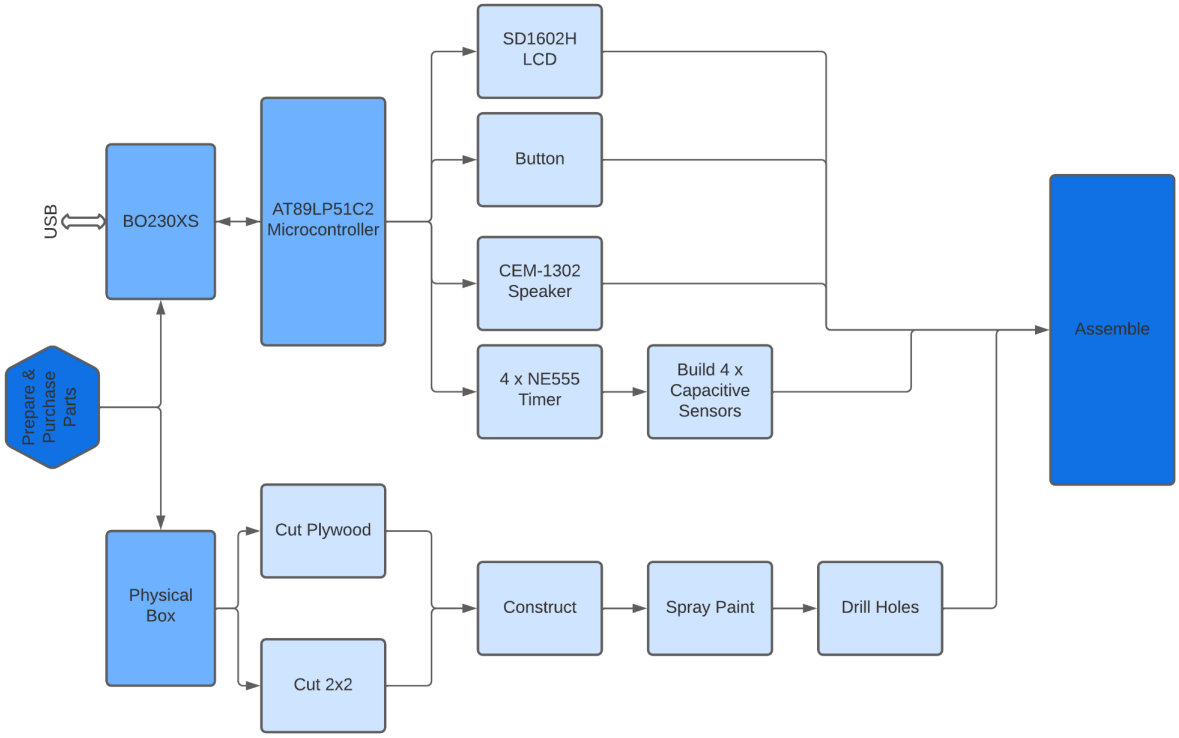

Figure 2. *Hardware Flowchart*

The hardware component was built on many skills we have learned over the past couple of years. For example, building circuits, working with microcontrollers, using speakers which can be configured in several ways, wiring an LCD, and creating large-scale capacitors. In addition, we have gained proficiency with hand tools, power tools, and constructing a working breadboard circuit.

AT89LP51RC2 Microcontroller and BO230XS USB Adapter

The AT89LP51C2 is the first microcontroller we have been introduced to this semester. It uses 8051 assembly language and is very compatible with breadboards. It uses 40 pins while there are 34 I/O pins. There is plenty of information regarding this hardware, making research

more accessible. In addition, there is the BO230XS USB adapter. As stated in the name, this allows us to connect our breadboard to our operating device.

The AT89LP51RC2 microcontroller allows us to process all commands for our project while the BO230XS connects to our devices. Together, we can program our breadboards as we see fit. The LCD, speaker, buttons, BO230XS, and the NE55 timers are connected to the microcontroller.

HD44780 LCD Display

Hitachi's HD44780 is the given LCD for this semester. It has two lines, each with 16 columns for character display. With minimal wiring required, this LCD works well for our needs. For our project, the LCD must update each player's scores continuously. When the first player reaches 15 points, the LCD will display "Win" for that player. We also disconnected the LCD from our main breadboard and placed it on top of our box, allowing for an integrated experience.

CEM-1203 Speaker

The CEM-1203 is the speaker we have been using since lab #2. Multiple setups for the speaker were given however our group opted for the MOSFET setup. With the MOSFET setup, the speaker was at its loudest. Since our hardware was inside the wooden box, we needed the speaker to be loud. We also configured our code to play several different frequencies, and the speaker could handle them all.

NE555 Timer and Sensors

The NE555 timer is an astable oscillator. The 555 timer generate square waves and, from here, can measure the period of the wave. Afterwards, using basic arithmetic to calculate the frequency. With our project, we had four NE555 timers set up. They were constantly measuring the capacitance from our sensors with this formula, the Sensors we built worked with the 555

Timers to read the capacitance. We made four functioning capacitors using tinfoil, cardboard, plastic sheets, and tape. Using two pieces of tinfoil with a plastic sheet on top, we imitated a parallel plate capacitor. We attached wires to both plates and plugged the wiring into the 555 Timers.

Box

The box was the physicality of our project. Plywood, 2x2's, and several screws made a 20x23-inch box. We spray-painted the box and drilled holes for all the wiring. One note to add, all wires were spun together to ensure problems with noise did not occur. In the end, we cushioned the hardware inside, moved the LCD onto the top, attached the four sensors, and labeled the box. With it wholly assembled, our final product was complete.

Software

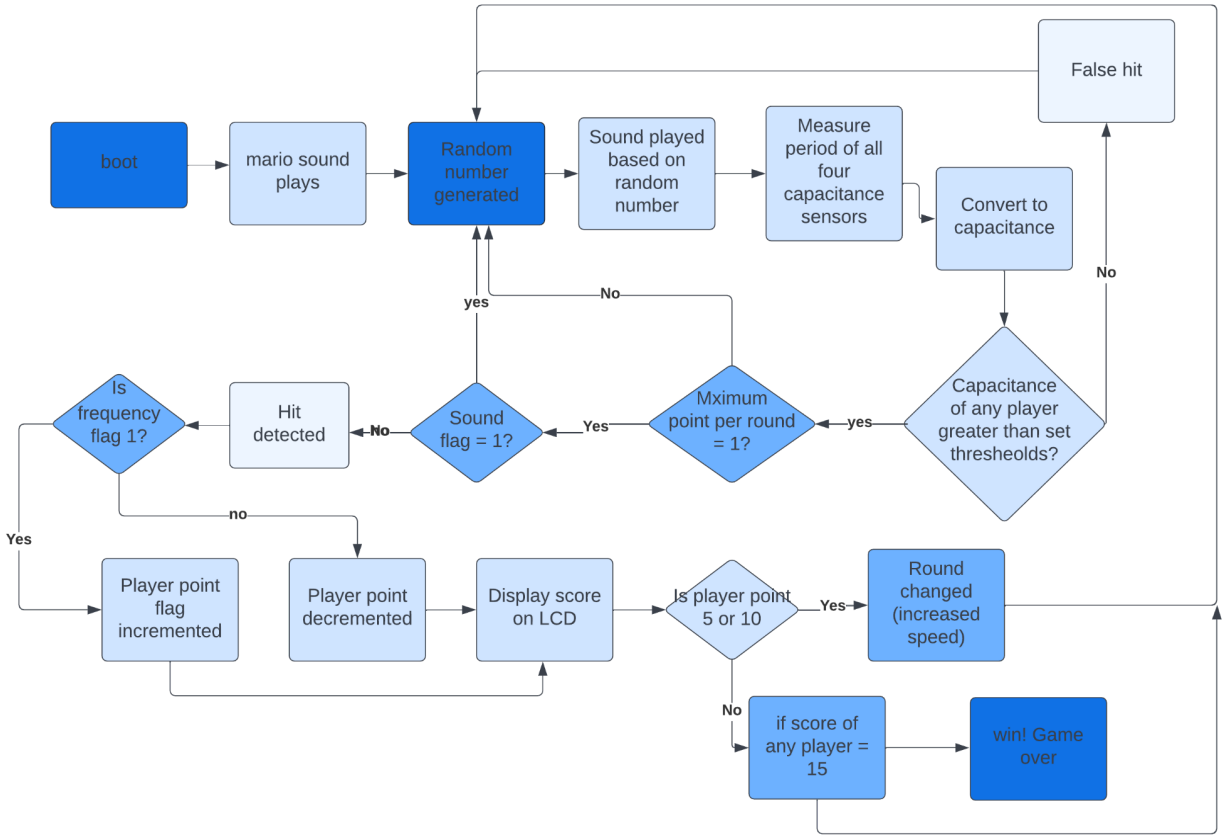

Figure 3. *Software Flowchart*

Speaker Control

Speaker Control

```
Timer0_ISR:
	jnb alarm_flag, Done
	jb C_flag, CNote
	jb G_flag, GNote
	jb Gl_flag, GlNote
	jb E_flag, ENote
	jnb fr_flag, LowFr

	mov TH0, #high(TIMER0_RELOAD)
	mov TL0, #low(TIMER0_RELOAD)

	mov RH0, #high(TIMER0_RELOAD)
	mov RL0, #low(TIMER0_RELOAD)
	ljmp HighFr

LowFr:
	mov TH0, #high(TIMER1_RELOAD)
	mov TL0, #low(TIMER1_RELOAD)

	mov RH0, #high(TIMER1_RELOAD)
	mov RL0, #low(TIMER1_RELOAD)

HighFr:
	cpl SOUND_OUT ; Connect speaker to P1.1!
Done: reti
```

Initially, we checked if the speaker should be turned on with the alarm flag; using several flags, we can control the output frequency of the speaker. The C, E, G and G1 flags are used to set the alarm eponymously. If none are set, the primary frequency flag is used to indicate either our high frequency or low frequency - in which players can gain or lose points, respectively.

Timer Interrupt Delays

A crucial problem we faced during our design process was keeping the speaker consistent without using delays (loops). With loops halting the program, we would be unable to play a sound and measure capacitance concurrently. The solution we reached was to use the Timer 2 interrupt to count to a given number - based on the current in-game round - and once reached, the speaker would register that it had finished waiting.

Timer Interrupt Delays

```
Timer2_ISR:
	clr TF2
	inc T2ov
	jnb wait_flag, Timer2_ISR_done
	mov a, counter
	add a, #0x01
	da a
	mov counter, a
	jb finalroundflag, HighSpeed
	jb speedflag, MedSpeed
LowSpeed:
	cjne a, #0x99, Timer2_ISR_done
	ljmp Inc_Major
MedSpeed:
	cjne a, #0x60, Timer2_ISR_done
	ljmp Inc_Major
HighSpeed:
	cjne a, #0x45, Timer2_ISR_done
Inc_Major:
	mov a, #0x00
	mov counter, a
	mov a, counterMajor
	add a, #0x01
	da a
	mov counterMajor, a
	cjne a, #0x03 , Timer2_ISR_done
Inc_Done2:
	clr wait_flag
	mov a, #0x00
	mov counterMajor, a
Timer2_ISR_done:
	reti
```

Random Number Generator and Randomized Delays

Random Number Generator and Ranomized Delays

```
SeedGen:
	setb TR2
	jb BOOT, $
	mov Seed+0, TH2
	mov Seed+1, #0x01
	mov Seed+2, #0x87
	mov Seed+3, TL2
	clr TR2
	ret
Random:
	mov x+0, Seed+0
	mov x+1, Seed+1
	mov x+2, Seed+2
	mov x+3, Seed+3
	Load_y(214013)
	lcall mul32
	Load_y(2531011)
	lcall add32
	mov Seed+0, x+0
	mov Seed+1, x+1
	mov Seed+2, x+2
	mov Seed+3, x+3
	ret
Wait_Random:
	Wait_Milli_Seconds(Seed+0)
	Wait_Milli_Seconds(Seed+1)
	Wait_Milli_Seconds(Seed+2)
	Wait_Milli_Seconds(Seed+3)
	ret
Wait_Random_Compounded:
	lcall Wait_Random
	lcall Wait_Random
	lcall Wait_Random
	lcall Wait_Random
	ret
```

To ensure that players would not predict the subsequent frequency or when it was played, we generated a seed upon starting that would be used for random number generation throughout the game. As given in class, the subroutine, "Random," multiplied the seed by 214013 then added 2531011. Consequently, this generated a new pseudorandom number for our program to use. Next, we sought to have our game wait a random number of seconds between each tone. The "Wait_Random" code block used the Wait_Milli_Seconds macro with our randomized seed to generate this delay. We found that this period was not long enough, so we created another subroutine that compounded Wait_Random; giving us the perfect range of wait times: enough to be substantial while not boring the players.

Initialization of Hardware and Software

This code section is a prime example of initializations made throughout the code. Such code does not vary in form compared to defining statements or pin initialization as necessary but runs once during the program duration. This section initializes hardware and sets the period measuring pins for the four 555 timer circuits used for the four players and clears scores of all players to avoid errors at boot.

Initialization of Hardware and Software

```
MyProgram:
    mov SP, #7FH
    lcall Initialize_All
    setb P2.0
    setb P2.1
    setb P0.0
    setb P0.1
    Set_Cursor(1, 1)
    Send_Constant_String(#C1)
    Set_Cursor(1,10)
    Send_Constant_String(#C2)
    Set_Cursor(2, 1)


    Send_Constant_String(#C3)
    Set_Cursor(2,10)
    Send_Constant_String(#C4)
    Set_Cursor(1,7)
    Send_Constant_String(#seven)
    Set_Cursor(2,7)
    Send_Constant_String(#seven)


mov a, #0x00
      mov player1count, a
            mov a, #0x00
      mov player2count, a
             mov a, #0x00
      mov player3count, a
            mov a, #0x00
      mov player4count, a
```

Gameplay Mechanics Example (Player 1)

Gameplay Mechanics example (Player 1)

```
forever:
    clr TR2 ; Stop counter 2
    mov TL2, #0
    mov TH2, #0
    mov T2ov, #0
    jb P2.0, $
    jnb P2.0, $
    mov R0, #0 ; 0 means repeat 256 times
    setb TR2 ; Start counter 0
    meas_loop1:
    jb P2.0, $
    jnb P2.0, $
    djnz R0, meas_loop1 ; Measure the time of 100
periods
    clr TR2 ; Stop counter 2, TH2-TL2 has the period

    mov x+0, TL2
    mov x+1, TH2
    mov x+2, #0
    mov x+3, #0

    jb win1_flag, contt
    Set_Cursor(1, 14)
    Display_BCD(player1count)

    jb cheatcodeflag, display
    jnb alarm_flag, display
    jb start_flag, display
    jnb fr_flag, decre
    load_y(1400)
    lcall x_gt_y
    jnb mf, wejump
    mov a, player1count
    add a, #0x01
    da a
    mov player1count, a

    setb cheatcodeflag
    sjmp display

decre:
    mov a, player1count
    cjne a, #0x00,decc
    sjmp display
    decc:
    load_y(1400)
    lcall x_gt_y
    jnb mf, display
    dec player1count
    setb cheatcodeflag
    sjmp display

Wejump3:
ljmp round2
Wejump:
ljmp cont

Display:
mov a, player1count
cjne a, #0x05, wejump3
setb speedflag
ljmp round2

Wejump2:
ljmp wincondition

round2:
    mov a, player1count
    cjne a, #0x010, wejump2r
    setb finalroundflag

wincondition:
   cjne a, #0x15, cont
   Set_Cursor(1, 14)
   Send_Constant_String(#win)
   setb win1_flag34
```

This section of code is the foundation of the gameplay mechanics. Although it only highlights code for player 1, it is sufficient to understand the code logic and flow as this section is quite repetitive for additional players. For example, “Forever” begins by measuring the period of the selected pin (100 times in this case) and then saves the period in TH2-TL2 in the variable x for future use. A flag is then used to immediately check win conditions and the branch to appropriate display statements. If the flag is not set, the code proceeds to check several other flags. For example, the start flag ensures that the appropriate start commands have been executed before gameplay (such as start sounds). The alarm flag makes sure that the sound has entirely played before points are counted. At the same time, the frequency flag ensures that points increment or decrement accordingly.

Speaker Functionality

Speaker Functionality

```
soundstuff:
      jb win1_flag, connect
      jnb start_flag, NoSoundBridge
      ljmp StartingSounds
NoSoundBridge:
      ljmp NoSound
StartingSounds:
      setb fr_flag
      setb E_flag
      setb alarm_flag
      lcall Wait_Half
      clr alarm_flag
      Wait_Milli_Seconds(#20)
      setb alarm_flag
      lcall Wait_Half
      clr alarm_flag
      lcall Wait_Half
      setb alarm_flag
      lcall Wait_Half
      clr alarm_flag
      lcall Wait_Half
      setb alarm_flag
      setb C_flag
      lcall Wait_Half
      clr alarm_flag
      clr C_flag
      Wait_Milli_Seconds(#20)
      setb alarm_flag
      lcall Wait_Full
      clr alarm_flag
      Wait_Milli_Seconds(#20)
      setb alarm_flag
      setb G_flag
      lcall Wait_Full
      clr alarm_flag
      clr G_flag
      lcall Wait_Full
      setb alarm_flag
      setb Gl_flag
      lcall Wait_Full
      lcall Wait_Full
      clr Gl_flag
      clr E_flag
      clr start_flag
      sjmp NoSound

connect:
ljmp theend

NoSound:
      jb wait_flag, Waiting
      clr alarm_flag
      lcall Random
      lcall Wait_Random_Compounded
      lcall Random
      mov a, Seed+1
      mov c, acc.3
      mov fr_flag, c
      setb alarm_flag
      setb wait_flag
      clr cheatcodeflag
```

The first code block, "sound stuff," checks if a player has won the game. If someone has, we will not need to play any more sounds from the speaker, so this block jumps to the end. The goal for "StartingSounds" was to use the frequency-control flags previously integrated above. On starting the game, the code for the intro-music would run once. Using 1-bit flags in conjunction with delays, we were able to time the frequency changes of the CEM-1203 mini speaker to mimic an actual song.

The final code block is the crux of how the game was played. First, if the program were not currently playing a sound and waiting, the alarm would turn off. We then generated a random number and waited this random amount of time. Next, we assigned the frequency flag a randomized bit so that the players would not predict the pitch of the following sound. Finally, we set the wait flag once the sound was playing again. The timer interrupt would begin counting; Thus, halting the sound from changing sporadically and allowing the program to continue.

h. Solution Assessment

Design Component	Testing Methods and results	Strengths/Weaknesses
Capacitance Threshold	Each Capacitance meter was calibrated separately and thresholds were adjusted constantly to maximize smooth gameplay. Each threshold was adjusted as new players were added as the noise generated would impact gameplay.	Throughout the project, the capacitance threshold posed the largest issues, constantly requiring recalibration. In the end, capacitive plates were very sensitive and would be able to reliably detect when they had been pressed. If we were to do it again, we would attempt to create a more consistent calibration system so that testing mid-design was easier
Random Number Generator	Several real life tests were conducted to ensure the numbers generated were adequate to make the game fun to play but not too challenging to a limit beyond average human capability. Testing was done to ensure the random wait times were not unreasonable.	Numbers generated were reliably random and the game was unpredictable; however, in the unlikely event that the seed was the exact same, the numbers would be predictable.
Sound Generation	Testing was imperative to allow for sound generation that added to the gameplay experience. Testing was done to ensure the two tones were distinguished enough to avoid confusion, while not being too annoying as the players would constantly be exposed to the sound. Testing was also done to make sure the sound was loud enough to be heard inside the aesthetic box.	Our team used the MOSFET circuit, allowing for the loudest possible speaker. However, it was tedious to code. We had planned several more bonus features, but the coding wasn't too simple.
Functionality of Sensors	Testing individual sensors included plugging them into a NE555 timer and observing whether there was a change in capacitance or not.	A strength of sensors was they were air tight, resulting in a smaller threshold, and they were reinforced to withstand impact. Their one weakness was the wire connection as it would easily come loose.
LCD Animation	Since we've tested the LCD plenty of times in previous labs, the only test was to make sure the "Win" statement came up when somebody reached 15 points.	The only weakness to the LCD was the amount of space we had to display 4 separate players and who the winner was.

IV. Live-Long Learning

This project was a learning experience for each member. Each member was able to work on their weaknesses and contribute. With each section of the project, we were all-inclusive and ready to support each other. A new experience for several members was Notion and the ability to store all renditions of code for idea generation collectively.

The most extensive knowledge gap seen by the team was timer interrupts and Random Seed generation. We ran into issues with timer interrupts as our program was being held up due to a wait function, so after some research, we realized the program was halted due to a wait function. We then researched timer interrupts and realized that utilizing a timer interrupt would fix our problem. Lastly, nobody was familiar with random seed generation, but overcoming this issue took less than five minutes. Information was provided on Piazza, and our professor, Jesus Calvino Fraga, released files to assist us. As a group, we also learnt not to over-engineer our work. By not making things overly complicated and by creating shortcuts in our code. As future engineers, we also found that the project's aesthetics were also important. We thought if this product were on the market, it would need to look presentable. So, an equal balance was put into the software and hardware. Another valuable skill learnt all around was time management. With other courses, a heavy midterm schedule, and webwork due every other day, we had to make sure we could still focus on the rest of our workload. Lastly, when it came to soft skills, collectively, we learnt better communication skills.

V. Conclusions

In this project, we set out with the goal of making a fast-paced, free-for-all, reaction-time based game. We have discussed our initial requirements, our design process, our challenges, our solutions to those problems, and what we have learned along the way.

To complete our game, we were required to incorporate our 8051 microcontroller system, homemade capacitive touchpads, astable oscillators using 555 timers, the CEM-1203 speaker, and our 2x16 LCD screen. Additionally, all programming for the project was constrained to assembly. The original design for the game was given as a two-person, one-versus-one contest with any additional rules up to the discretion of the design team.

Our team's design process followed the seven-step engineering design process very closely. First acquiring a tight grasp on the problem specifications, then going far beyond the bounds of the project for idea generation to foster a bountiful cornucopia of solutions. Using these ideas we were able to narrow our scope on a manageable yet ambitious project solution design.

Throughout our design process we assembled blocks of code and hardware structures individually, only fully integrating them once we had tested them enough to ensure they would work under our design vision. The greatest challenges we faced were achieving a consistent random-number generator, designing highly responsive and reliable capacitive plates, integrating timer interrupts to halt only specific parts of the program, and creating a high-quality ready-for-market frame to house our game's hardware.

Overall, the project took us about 65 hours to complete, with heavy periods of time being spent on bug-fixing for the speaker, random-number generation and 4-player capabilities. In the end, every member of the team played to their strengths, while also managing to learn something new, whether it be high-quality wood-working, random-seed generation, or timer interrupt integration.

VI. References

[1] Calvino-Fraga, Jesus, “Project_1_Capacitive_Sensor_Reaction_Game.pdf”, University of British Columbia, Vancouver, 2022.

[2] Calvino-Fraga, Jesus, “Lecture_Project_1.pdf”, University of British Columbia, Vancouver, 2020.

[3] Calvino-Fraga, Jesus, “Timers, Interrupts, and Pushbuttons.pdf”, University of British Columbia, Vancouver, 2022.

[4] Calvino-Fraga, Jesus, 2022, Period_RC2_math[asm], University of British Columbia, Vancouver.

[5] Calvino-Fraga, Jesus, 2022, Math32[inc], University of British Columbia, Vancouver.

[6] Calvino-Fraga, Jesus, 2022, LCD_4bit[inc], University of British Columbia, Vancouver.

[7] Calvino-Fraga, Jesus, 2022, ISR_example[asm], University of British Columbia, Vancouver.

VII. Bibliography

Texas Instruments, “LM555 Timer”, LM555 datasheet, 2015.

CUI Devices, “CEM-1203”, Magnetic Buzzer Transducer specification, 2019.

Microchip Technology, “AT89LP51RC2”. AT89LP51RC2 datasheet, 2011.

VIII. Appendix

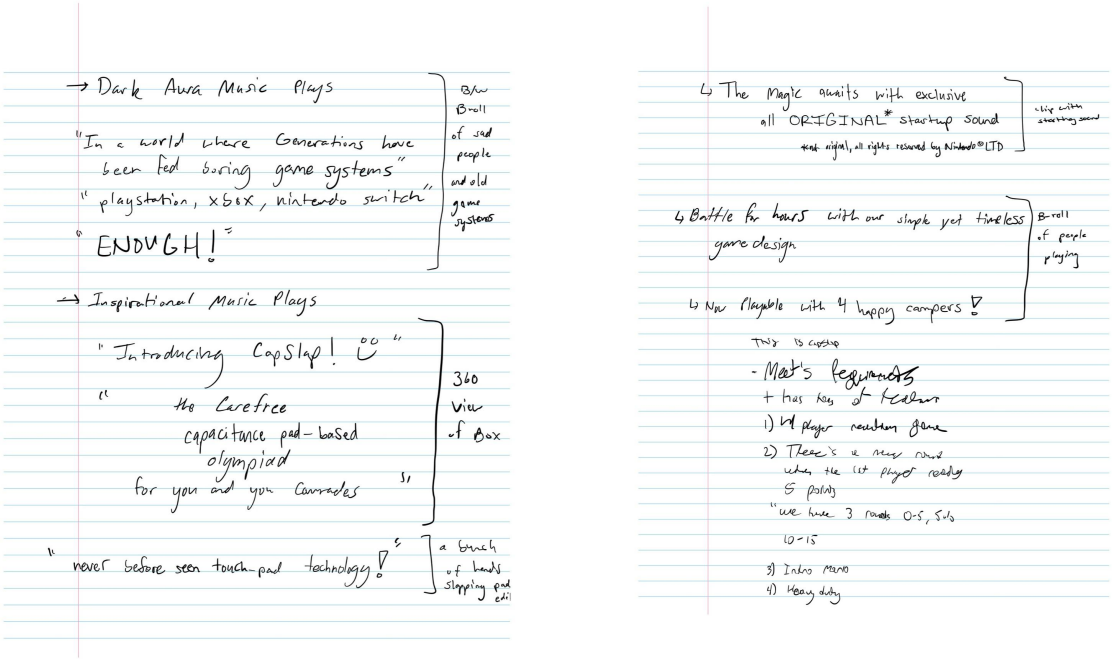

Figure 4. *Script for Skit Displaying Product*

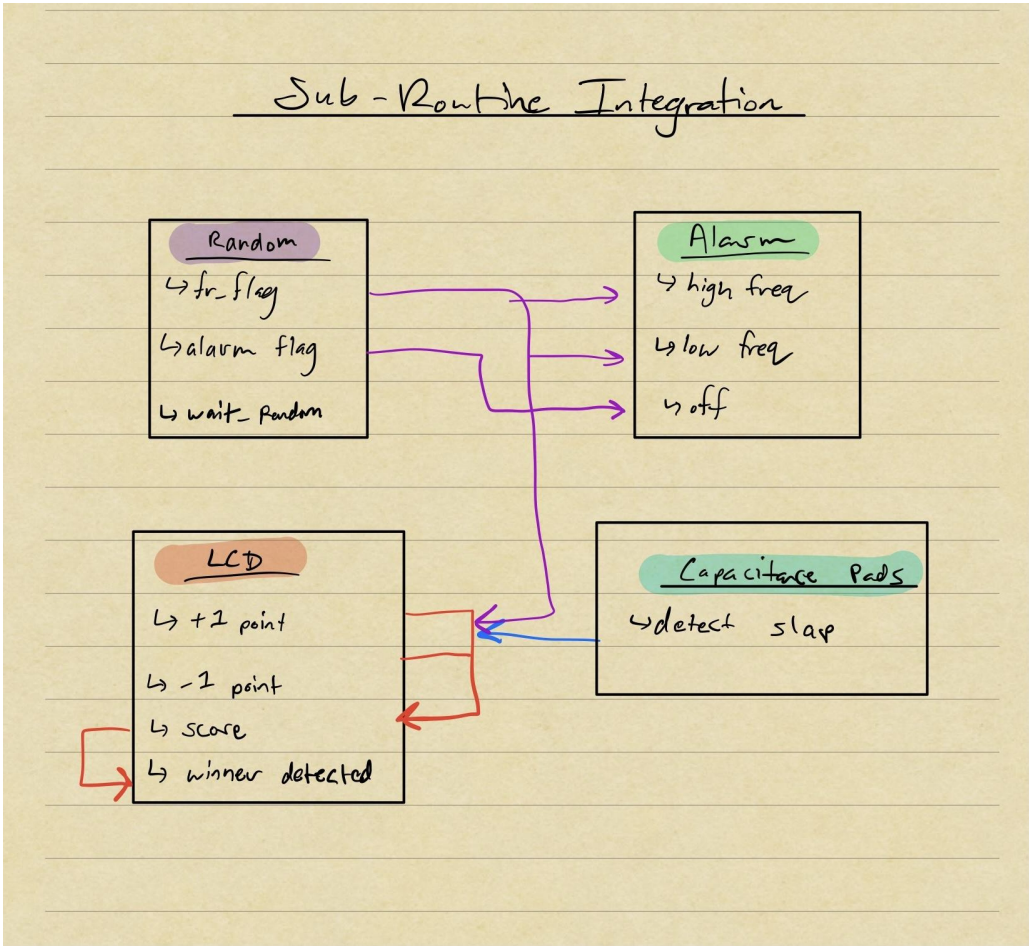

Figure 5. *Early-Design Software Flowchart*

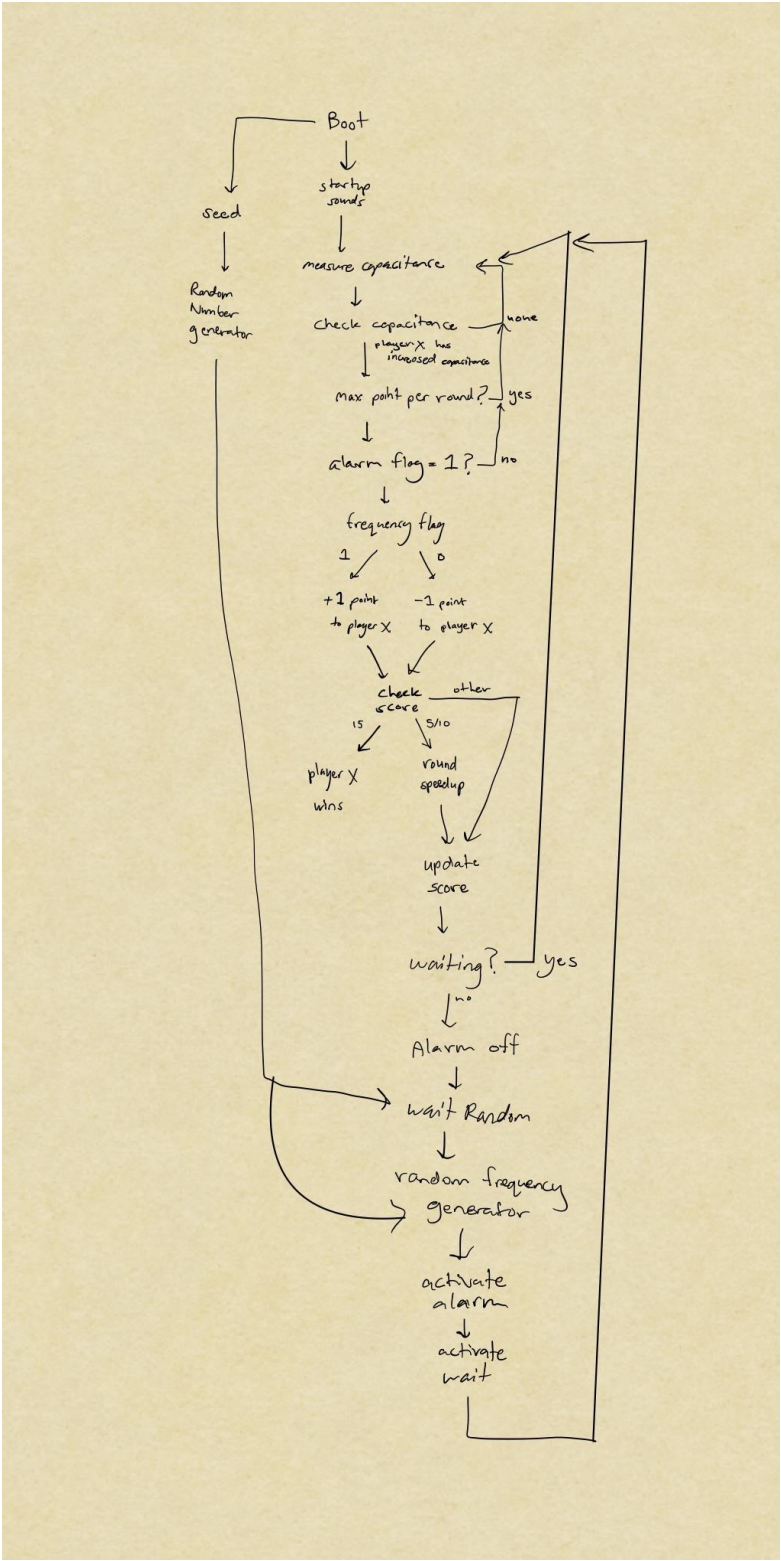

Figure 6. *Mid-Design Full Software Flowchart*