Project 2: Breakout

Garrett Nadauld

&

Ryan Enslow

Date: May 13, 2024

Contents

1 Introduction

The BREAKOUT game project introduces a classic arcade experience brought to the 8051. Inspired by ATARI's iconic 1976 release, our project aims to recreate the engaging gameplay of BREAKOUT. At its core, the BREAKOUT game system consists of a portable electronic device designed to entertain players with its gameplay. Players control a paddle to bounce a ball against a wall of bricks, aiming to destroy them all while avoiding letting the ball fall off the bottom of the screen. With each successful hit, players earn points and strive to achieve the highest score possible. The game can be played with one player or two to allow for friendly competition.

Our project encompasses both hardware and software designs tailored to meet the specific requirements of the game. The following sections, will give insight into the detailed design specifications, outlining the hardware components, software algorithms, and integration strategies employed to bring the BREAKOUT game to life.

2 Scope

In this document we will be taking a look into the design of our project. This will include the requirements, dependencies, theory of operation, and design alternatives. We will also be covering the details of our design such as why we chose a specific approach and how we achieved that design. The next topic will be the testing we performed on our design such as how we verified working portions of code. Next we will look at the performance and functionality as well as how our design could be improved upon in future iterations. At the end of the document is an appendices containing relevant software and data listings.

3 Design Overview

In this section we will be looking into the design of our project.

3.1 Requirements

1. The system shall run on an external 9v DC supply.
2. The system shall use a 64x128 pixel LCD to display the playing surface.
3. The system shall have two buttons. One reset button and one start button.
4. The system shall have one potentiometer located near the LCD.
5. The system shall have a DIP switch that sets the number of players (1 or 2), the paddle size for each player, and the "slow" speed of the ball.
6. The bricks, ball and paddle shall be confined to a play area. The current scores for players 1 and 2, the high score and ball count shall be confined to a score area to the left of the play area. The score area shall be as narrow as reasonably possible. The font shall have a minimum height of 7 pixels.
7. The left, right and top borders of the play area shall always be displayed. During a game, the paddle shall also be displayed.
8. When the ball is in play, the display shall show the ball, which is a minimum of 5x5 pixels and a maximum of 8x8 pixels.
9. Upon reset and after all the bricks have been destroyed, the top third of the play area shall be populated with 5 or more rows of bricks. These bricks shall be 3 pixels in height and 5 or 6 pixels in

width. Bricks shall have 1 pixel of space between them.

10. The ball shall ricochet off bricks and the play area borders with an angle equal to the incident angle. If the ball hits the paddle, it shall bounce with an angle based on the point of impact.
11. If the ball hits a brick, that brick shall be erased and the player shall score 1 point. If the brick is deeper than the third brick row, the ball's speed shall be set to "fast".
12. If the player misses a ball and it falls through the bottom of the play area, the ball is "lost". When the player loses three balls, the game shall be over.
13. Upon reset, and until the start button is pressed, the display shall indicate that the game is ready.
14. Upon the loss of the last player's third ball and until the start button is pressed, the display shall indicate that the game is over.
15. When the start button is pressed, the players' scores shall be set to zero. At any time if a player's score exceeds the high score, the high score is updated.
16. The high score shall be stored in non-volatile memory. Some means shall be provided to clear the high score.
17. One second after the start button is pressed, and one second after the first or second ball is lost, a new ball shall be "served" at "slow" speed from a point near the center of the first brick row. (Exception: See Requirement 18.)
18. If a two-player game is selected, serves shall alternate between player 1 and player 2, and the text: "Player 1 Ready" or "Player 2 Ready" shall be displayed for at least 2 seconds before each serve.
19. The ball shall have a minimum speed of 100 pixels per second and a maximum speed of 250 pixels per second. The trajectory shall have one of at least 8 different angles (2 per quadrant), none of which are vertical or horizontal.
20. The paddle size for each player shall be configured (using 2 DIP switches each) to be 8 pixels wide, 12 pixels wide, 16 pixels wide, or 24 pixels wide.
21. The "slow" speed of the ball shall be configured (using 2 DIP switches) to be 100, 120, 150, or 200 pixels per second. The "fast" speed shall be faster than the slow speed.
22. A sound shall be generated each time the ball hits the paddle, a brick or the border. A sound shall also be generated when a ball is lost. The sound for each of these events shall be distinct and shall not exceed 250 milliseconds.

3.2 Dependencies

We made our system run on a 9v DC battery supply. Due to this it has no dependencies.

3.3 Theory of Operation

Below are major blocks of our code and a description of their function. The corresponding code can be fount in 7 APPENDIX CODE

1. The pot function takes the input from the potentiometer and averages it over the sample time. This calculation is used to move the paddle with the potentiometer.
2. The disp_char function takes the desired row, column, and character as an input. It then displays the desired character to the LCD at the desired location.

3. The disp_score function divides the score into integers for the ones, tens, hundreds, and thousands places. Once it has calculated these values it calls the display character function so that the score is updated on the LCD display.
4. The display function is used to draw the borders, the scores, as well as the active player. Also displays how many lives are left.
5. Wait screen provides a screen in between player's turns. It displays "Player (current player) Ready" and then waits for the start button to be pressed. If it's the start of the game it also serves as the configure game screen.
6. Game over screen function clears the screen except boarders and scores and prints "GAME OVER in the play area.
7. The turn over screen function is activated when a players ball touches the bottom of the screen. It checks if it's a two player game and configures the current player's settings accordingly.
8. Draw_bricks() function cycles through the current players brick array and prints each remaining brick on the screen and deletes any bricks that have been knocked out.
9. The draw paddle function takes the current player's paddle size and draws the paddle accordingly to the current potentiometer reading.
10. Draw_ball function does quite a lot it handles all the updating the ball on the screen as well as detecting hits. This includes bounces off walls, off the paddle, and off the bricks. Detects when the ball has hit a level 3 brick first and speeds up the ball accordingly. It also detects if the ball went off screen and sends the program to the end turn function.
11. Move the ball updates the current position of the ball according to the current xangle and yangle parameters and then calls draw_ball and updates the display.
12. The main loop waits for a certain number of interrupts from timer2 and then draws the paddle, draws the bricks, moves the ball, and then sets up timer4 to play a certain sound depending on if the ball bounced off something.

3.4 Design Alternatives

The breakout game, or any game for that matter, involves a series of critical design decisions that can significantly impact the gameplay and software development experience. These decisions span across various aspects of the game, including the physics, graphics, user interface, and the progression of difficulty. Each of these elements plays a crucial role in shaping the overall feel and appeal of the game.In the following sections, we will delve deeper into each of these aspects, exploring potential design approaches and the reasons for choosing one over another.

Drawing the bricks and the physics regarding the bricks is a area for numerous software implementations. For us we went with drawing the bricks byte by byte, this is due to multiple bricks being drawn on the same page. We could have printed the bricks as separate characters with the print char function. The issue with this is that you would have to have multiple characters depending on which brick is broken. This was a little challenging and we honestly just found it easier to and or or half the byte depending on which brick we were turning on or off.

We also kept track of the bricks with an array but you could also do this with one large character, or store it in some other way. We did an array because it matched the layout of our bricks on the screen and was easy to conceptualize. We then normalized our ball position to line up with the array when it hit a brick.

This hit detection could be done in any number of algorithms but we found that normalizing the balls current position seemed to be the easiest for us.

The graphics and user interface form the visual core of the game, dictating how players interact with the game and perceive its aesthetics. The score and play area could be organized in many ways on the screen. We chose to put all the scores and player balls on the right side of the screen this made ball bouncing physics easier as we could zero index them in relation to the left side of the screen. This just helped us conceptually with the movement algorithms.

Lastly, the game difficulty and progression determine how the game evolves as the player advances, thereby affecting the game's replay ability and the player's sense of achievement.

4 Design Details

4.1 Hardware

A list of hardware components can be found in Appendix A. This section will describe the use and wiring for each key component.

4.1.1 Buttons and Switches

This project we used the 8-position dip switch and one button. The dip switches are used to configure ball speed, paddle sizes, and one or two player mode. Button one was used to start the game. Button two was unused.

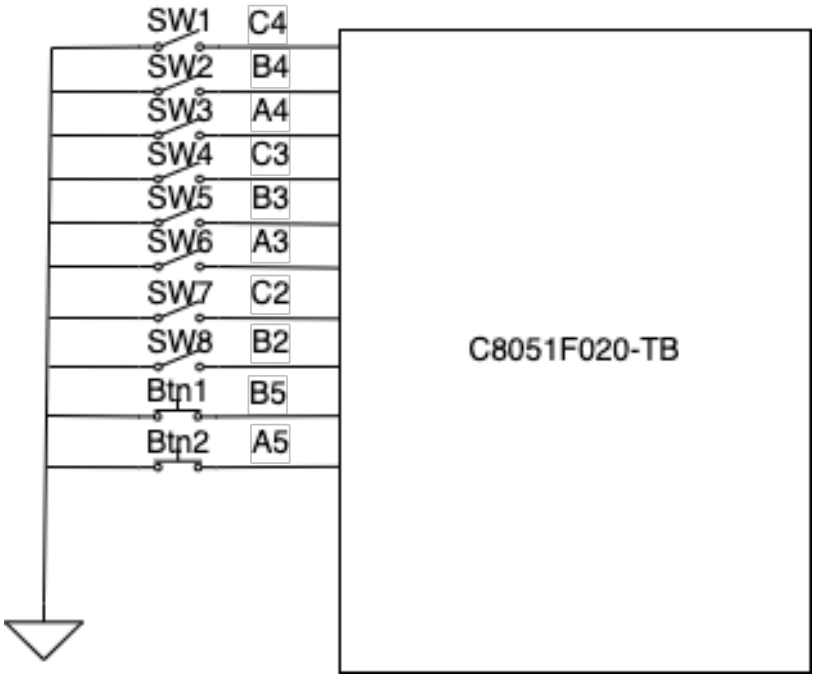

Figure 1: Wiring for switches and buttons

4.1.2 LCD Interface

The LCD interface is probably the most crucial component to this lab. It is what makes this game playable.

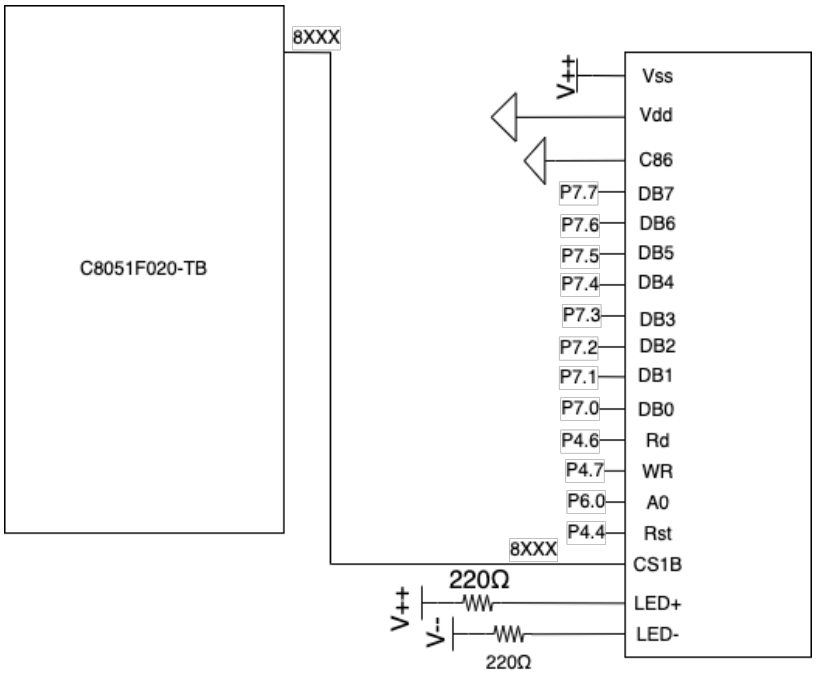

Figure 2: Wiring for LCD interface

4.1.3 Potentiometer

The potentiometer was used to control the paddle movement. Potentiometer two was unused.

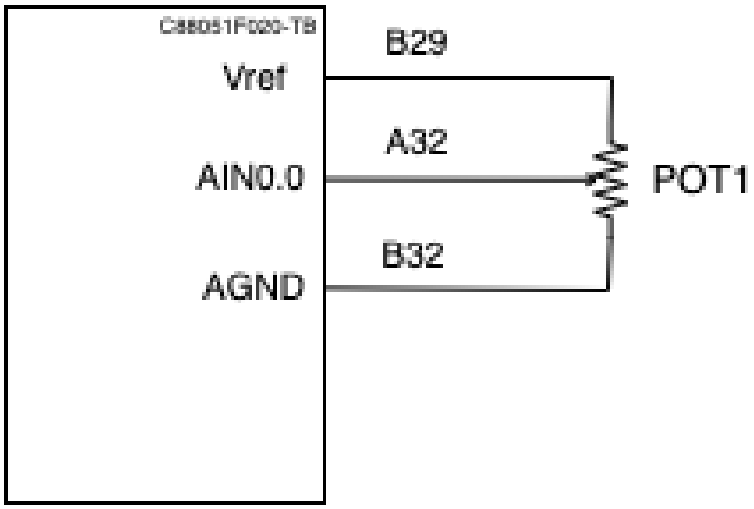

Figure 3: Wiring for potentiometer 1

4.1.4 Speaker

Below is a diagram for wiring the speaker which plays sounds for our breakout game.

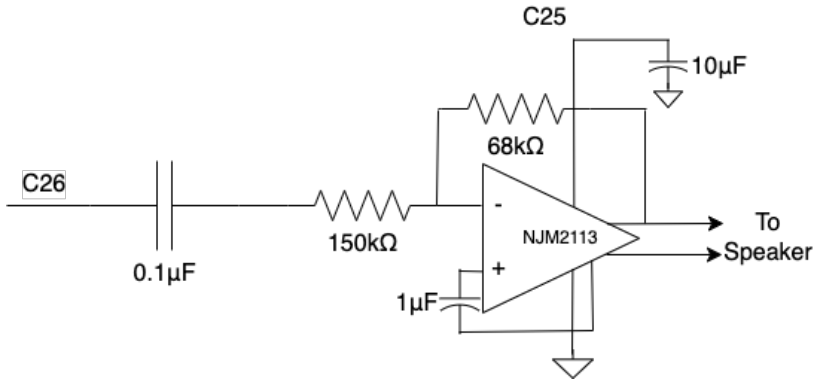

Figure 4: Wiring the speaker

4.2 Software

In this section, we will delve into the software architecture of the BREAKOUT game, focusing on the flow of the program and the functions that drive the gameplay.

The software flow of the game is visually represented in Appendix C through a software diagram. This diagram provides a high-level overview of the game's structure and the sequence of operations that occur during gameplay. It illustrates the basic gameplay loop, which is the cycle of events that repeats throughout the game, such as player input, game logic (like ball movement and collision detection), and screen updates.

The gameplay loop is the heart of the game, and it's where most of the checks occur. These checks include collision detection between the ball and the paddle or bricks, checking if the ball has fallen off the bottom of the screen, and checking if all bricks have been destroyed. The timing and order of these checks are crucial for the game to function correctly.

While the overall software flow provides a big-picture view of the game's operation, the individual functions within the program are where the real action happens. Each function has a specific role in the game, such as moving the paddle, updating the ball's position, or calculating the score. These functions are called at appropriate times within the gameplay loop to create the dynamic, interactive experience of playing BREAKOUT.

Although the software flow won't be discussed further in this section, the functions in the program will be examined in detail. This will include a discussion of what each function does, how it contributes to the overall game, and how it interacts with other functions. For the actual code implementation of these functions, please refer to the breakout.c file in Appendix B.

1. The pot function uses the ADC interrupt to update the pot value. Depending on the player it will put the pot value within range of the bottom of the screen. It then does this for 8 total samples and then averages the pot value for more accurate readings.

2. The disp_char function takes the desired row, column, and character as an input. It then converts the row to a position in the display register by multiplying it by 128. It then adds the x value to locate the exact byte of the display to start the print. It then calls font5x8 to print the character to the screen.

3. The disp_score function separates the score into 4 separate digits and and prints it in the specified area of the screen. It is similar in function to display char, except for the breaking up of the score into 4 separate characters using integer division and modulo.

4. The display function is used to draw the border by drawing two lines on the left of the screen and then two lines 80 bits to the right on on the screen. It also draws two lines on top connecting the side walls. The function also checks if the current score is greater than the high score and updates the high score accordingly. This function then prints the high score, current player's score and the player's lives on the right side of the screen.

5. Wait screen provides a screen in between player's turns. It turns off timer two which determines the speed of the game and essentially pauses it when in this screen. It then resets the ball to the center of the screen and sets a starting x and y angle. It then displays "Player (current player one or two) Ready". It then checks if it's the begging of the game and if so it allows you to configure the game settings. When the button is pressed the timer turns back on and the game resumes.

6. Game over screen just stops the clock and prints "GAME OVER" in the play area and prints the boarders and scores. It then waits until the game is reset.

7. The turn over screen function is activated when a players ball touches the bottom of the screen. It first sets up timer 4 for playing a distinct 784Hz sound byte for hitting the bottom. The speed is then reset to the initial speed. It then checks it's a one or two player game, if not player one's lives are checked and if there's no lives left the game is sent to Game Over otherwise the lives are decremented by one. If it's a two player game it does the same and then changes over all information for the next player's turn.

8. Draw_bricks() function checks who's turn it is and sets the blocks to be drawn. It then cycles through the blocks and either draws or deletes a block at that location based on whether a one or a zero is in the array. It draws the bricks in page two and each brick is 3 deep and 6 wide with a one pixel buffer in between each brick. While the bricks are being printed it counts how many zeros were detected. If the count is 55 that means all the bricks are broken. The function will then wait till the ball is below the bricks and repopulate the arrays so that a fresh set of bricks are drawn.

9. The draw paddle function takes one input and that is the current potentiometer reading that has been normalized for the screen size. It then checks the current player's selected paddle width and draws it from the bottom left corner o the play area or 898 the potentiometer offset is then added to that location and then the paddle is drawn from the starting location with the desired width represented by i. This prints a paddle height of two with the current players width.

10. Draw_ball function is a huge portion of this program and should probably be broken up into separate functions, but it worked enough to pass off. The x and y position of the center of the ball are passed into the function. From there the starting draw position is extrapolated from the center by subtracting 2 from x and y. Next the ball is drawn and stores a value in hit if it overwrites a turned on bit meaning a collision occured. The ball then checks if it hit a corner and inverts the x and y angle. Next it checks the walls and if it hits one of the walls the xangle is inverted. It then checks if the ball hit the top of the play area and if so it inverts yangle. If the ball hit the bottom of the screen it will send the game to the turn end function.

    When the ball is exactly at the 60 y height is the only time in which it could hit the paddle. So at that point we check if there's a collision and if so we check if it was on the outside of the paddle and return a shallow angle. If it's on the inside it returns a steep angle. It then checks if it's on the left side if it is it inverts the xangle so it moves left.

If none of the other conditions tripped and there was a hit it means the ball hit a brick. So we first normalize the location based on the directions the ball was moving. If a brick is there we delete it, if not it means we are hitting something side to side so we center y and then delete the brick to the left or right depending on if it's moving right or left. It then inverts the angle based on the direction it hit the ball from.

The last section checks if it's the first time a brick in the third row has been broken and if so the speed of the ball increases by modifying t_cnt.

11. Move the ball increments the current ball position by the current x and y angle it then draws the ball and then stores an value for the desired audio in bonk. Afterwards it updates the display.

12. The main loop uses the timer 2 interrupt to determine how often the screen updates. The amount of timer 2 interrupts before the screen and ball updates is saved in t_cnt. Each time the timer 2 interrupt is called it adds one to int_cnt. The main loop then checks when these are the same it then clears the screen, draws the paddle, draws the bricks, moves the ball and then clears the int_cnt. After this is checks for an audio code and plays the corresponding audio.

5 Testing

1. The system shall have two buttons. One reset button and one start button. To test this we pressed the restart button and saw if it reset the game and then pressed the start button and saw if it allowed the game to progress.

2. The system shall have a DIP switch that sets the number of players (1 or 2), the paddle size for each player, and the "slow" speed of the ball. We tested this by going through all configurations of the dip switches and seeing how it effected the game set up. Made sure it changed the players in a game and the starting speed of the ball as well as the seperate paddle sizes.

3. The bricks, ball and paddle shall be confined to a play area. The current scores for players 1 and 2, the high score and ball count shall be confined to a score area to the left of the play area. The score area shall be as narrow as reasonably possible. The font shall have a minimum height of 7 pixels. To test this we made sure the screen was displaying correctly and that all the values were in the right place.

4. The left, right and top borders of the play area shall always be displayed. During a game, the paddle shall also be displayed. To test this we played through many games and made sure the boarders and paddle and play area were always on the screen.

5. When the ball is in play, the display shall show the ball, which is a minimum of 5x5 pixels and a maximum of 8x8 pixels. We made sure the ball was 5x5 pixels by printing it on the screen and making sure it was displaying correctly.

6. Upon reset and after all the bricks have been destroyed, the top third of the play area shall be populated with 5 or more rows of bricks. These bricks shall be 3 pixels in height and 5 or 6 pixels in width. Bricks shall have 1 pixel of space between them. We tested this by pressing reset and making sure all bricks were reset and also by clearing the bricks in a game and made sure the bricks then repopulated.

7. The ball shall ricochet off bricks and the play area borders with an angle equal to the incident angle. If the ball hits the paddle, it shall bounce with an angle based on the point of impact. To test this we slowed the ball down and watched as it hit the sides and bricks and made sure that it bounced correctly. The only time the angle should change is when it bounces off the paddle.

8. If the ball hits a brick, that brick shall be erased and the player shall score 1 point. If the brick is deeper than the third brick row, the ball's speed shall be set to "fast". We tested this by test playing the game and making sure bricks were deleted and the score increased. We then broke bricks until we reached the third row and saw a noticeable speed up of the ball.
9. If the player misses a ball and it falls through the bottom of the play area, the ball is "lost". When the player loses three balls, the game shall be over. We tested this by playing the game until all lives were lost and a game over screen was displayed.
10. Upon reset, and until the start button is pressed, the display shall indicate that the game is ready. We tested this by seeing if the player ready screen was displayed while it waited for a game or turn to start.
11. When the start button is pressed, the players' scores shall be set to zero. At any time if a player's score exceeds the high score, the high score is updated. We tested this by setting a high score and then playing till we beat it and seeing if the score is updated. After we made sure the scores were set to zero at the start of a new game.
12. One second after the start button is pressed, and one second after the first or second ball is lost, a new ball shall be "served" at "slow" speed from a point near the center of the first brick row. We tested this by pressing start and counting until the ball starts moving. We then did the same between turns.
13. If a two-player game is selected, serves shall alternate between player 1 and player 2, and the text: "Player 1 Ready" or "Player 2 Ready" shall be displayed for at least 2 seconds before each serve. We tested this by playing the game and seeing the player one and player two ready screens in between turns.
14. The ball shall have a minimum speed of 100 pixels per second and a maximum speed of 250 pixels per second. The trajectory shall have one of at least 8 different angles (2 per quadrant), none of which are vertical or horizontal. We tested this by calculating the timer 2 count needed for all speeds. We then made sure there were 8 trajectory angles by having a step and shallow bounce of angle and then various configurations depending on which sides it bounced off of totaling in 8 angles.
15. The paddle size for each player shall be configured (using 2 DIP switches each) to be 8 pixels wide, 12 pixels wide, 16 pixels wide, or 24 pixels wide. We tested all dip switch configurations and made sure that they made all four paddle sizes.
16. The "slow" speed of the ball shall be configured (using 2 DIP switches) to be 100, 120, 150, or 200 pixels per second. The "fast" speed shall be faster than the slow speed. We tested this by setting all four combinations of the speed switches and made sure it created the four distinct speeds.
17. A sound shall be generated each time the ball hits the paddle, a brick or the border. A sound shall also be generated when a ball is lost. The sound for each of these events shall be distinct and shall not exceed 250 milliseconds. We tested this by bouncing the ball off walls, bricks, the paddle, and the bottom and listening to the audio subsequently played.

6 Conclusion

The endeavor to recreate the iconic BREAKOUT game on the 8051 microcontroller platform has been an intricate exercise in both hardware and software engineering. This project provided valuable insight into embedded systems and really tied all we learned together into one great whole.

The project's inception involved meticulous planning and strategic decision-making. Early-stage choices, encompassing game physics, user interface design, and game difficulty progression, had profound implications on the game play experience and the complexity of the implementation.

The software architecture of the game, visually represented in the software diagram, provided a macroscopic view of the game's structure. It delineated the fundamental game play loop and the sequence of operations integral to the game play. This served as a blueprint for the development process, guiding the implementation of the various functions within the program.

The hardware components were selected and integrated with the software to create a seamless and immersive gaming experience. The utilization of the 8051 microcontroller's I/O ports for display and user input proved to be a good exercise in tying the hardware and software together.

The project had numerous challenges, ranging from technical hurdles like optimizing the game for the resource-constrained 8051 microcontroller, to design challenges such as crafting an engaging and balanced game play experience. Each obstacle was met with thorough planning and innovative problem-solving, culminating in a successful outcome.

In conclusion, the BREAKOUT game project provided a solid base to showcase the skills we gained in embedded systems. It was a difficult class with a pretty rewarding final project, although I have to say this lab write up was very long. Overall it was a great experience in the class.

7 Appendices

Appendix A:

Part	Quantity
C8051FX20-TB	1
96-pin edge card connector	1
32-pin connector	3
6-pin bussed resistor networks	2
8-position DIP switch	1
Pushbutton switch	1
18-pin male header	1
18-pin female solder tail connector	2
S12864N LCD module	1
50k potentiometer	1
NJM2113	1
8Ω speaker	1

Table 1: Parts used for this project

Appendix B:

```
//=============================================================
// Program Name: breakout.c
//
// Authors: Ryan Enslow & Garrett Nadauld
//
// Description:
// This program is a two player breakout game devolped on the 8051. The game will begin by

// the first player starting the ball by pressing the start button. The switches will be u
sed to configure the
// game. Switches 1&2 will decide the paddle width for player 1. Switches 6&7 will decide
the
// paddle width for player 2. Switches 4&5 will be used to select the speed of the ball. S
witch
// 8 sets 1 or 2 player game. To restart the game the reset button will be pressed.
//
// Company: Weber State University
//
// Date            Version        Description
// ----            -------        -----------
// 4/8/2024         V0.1          Initial version
// 4/12/2024        V0.3          Display code working
// 4/15/2024        V0.5          Ball bouncing on walls
// 4/17/2024        V0.6          Paddle is working and bricks populating
// 4/18/2024        V0.7          Bricks are breaking but with low accuracy
// 4/19/2024        V0.8          All functionality works except audio
//                                Speed up is also almost unnoticeable
// 4/20/2024        V1.0          All functionality is working enough for
//                                Pass off
// 4/21/2024        V1.1          Added comments to code
// 4/22/2024        V4.2          Doctor Brown approved
//=========================================================================

#include <c8051f020.h>
#include <lcd.h>
#include<stdio.h>
#include<stdlib.h>

long score_2, score, score_1, high_score = 0;
sbit player_sw = P1^7;
char switches, bonk = 0;
bit player, num_player, su = 0;
int int_cnt, t_cnt, t_cnt_init, pad_w, pad_w2, pot_val, count = 0;
char ball_1, ball_2 = 3;
char xpos, ypos, xangle, yangle = 0;
code unsigned char ball[] = {0x0E, 0x1F, 0x1F, 0x1F, 0x0E};
code unsigned char sine[] = { 176, 217, 244, 254, 244, 217, 176, 128, 80, 39, 12, 2, 12, 3
9, 80, 128 };
xdata unsigned char blocks_1[11][5] = {
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1}
    };
xdata unsigned char blocks_2[11][5] = {
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
```

```
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1}
    };
unsigned long sum = 0;
sbit button = P2^6;
////////////////////////////////////////////////////////////////////////////////
//                        Timer 4 Interupt: Used for Game audio
//  Occurs when timer four flag is raised.
//  This plays the sound bite for the desired duration.
//  Audio is set up for certain frequencies and then played through this interupt.
//
////////////////////////////////////////////////////////////////////////////////

unsigned char phase = sizeof(sine)-1;   // current point in sine to outputcode

unsigned int duration = 0;      // number of cycles left to output

void timer4(void) interrupt 16
{
    T4CON = T4CON^0x80;
    DAC0H = sine[phase];
    if ( phase < sizeof(sine)-1 )   // if mid-cycle
    {
        phase++;                    // complete it
    }
    else if ( duration > 0 )    // if more cycles left to go
    {
        phase = 0;              // start a new cycle
        duration--;
    }
    if (duration == 0){         //Turn off timer when the sound is done
        T4CON = 0x00;
        }
}

////////////////////////////////////////////////////////////////////////////////
//                  Potentiometer Reading: Used for moving the paddle
//  Occurs when ADC flag is raised and it samples the potentiometer.
//  Takes several samples from the potentiometer and averages them.
//
////////////////////////////////////////////////////////////////////////////////
void pot() interrupt 15{

    unsigned long samp = 0;
    unsigned int D;
    AD0INT = 0;

    D = (ADC0L|(ADC0H << 8)); //Takes in the Data from the potentiometer
    if(player == 0){
    samp = (89L - pad_w)*D/4096;    //This normalizes the sample to fit within the play ar
ea for player 1
    }
    else{
    samp = (89L - pad_w2)*D/4096;   //This normalizes the sample to fit within the play ar
ea for player 2
    }
    sum += samp;                    //Takes the sum of 8 samples
```

```
    count++;

    if(count % 7 == 0)              //after 8 samples take the average and update the pote
ntiometers
    {
        pot_val = sum >> 3;         //bit shift for division
        sum = 0;
    }

}

////////////////////////////////////////////////////////////////////////////////
//                      Displays Characters to Screen
//  Input:
//      -Row: 0-7 On the screen
//      -Col: 0-127 Desired column to start the character
//      -single_char: One Keyboard character to be printed to the screen.
//  Output:
//      -Displays single character on the LCD display
////////////////////////////////////////////////////////////////////////////////
void disp_char(unsigned char row, unsigned char col, char single_char)
{
    int i, j;
    unsigned char k;
    i = 128*row+col;                //takes the row and column and translates it to the ex
act position on LCD
    j = (single_char - 0x20)*5;     //translates the single char to be comptible with the
font function.
    for(k = 0; k < 5; k ++) {
        screen[i+k] = font5x8[j + k]; //calls font and prints the char
        }
}

////////////////////////////////////////////////////////////////////////////////
//                      Displays Score to Screen
//  Input:
//      -X: 0-7 row on the screen
//      -Y: 0-127 Desired column to start the character
//      -Score: Four digit score to be printed on the screen
//  Output:
//      -Displays entire score on the LCD display
////////////////////////////////////////////////////////////////////////////////
void disp_score(int x, int y, unsigned long score){
    int thou = 0;
    int hund = 0;
    int tens = 0;
    int ones = 0;
    thou = score/1000;  //takes the thousands place
    score = score%1000; //remove the thousands place
    hund = score/100;   //takes the hundreds place
    score = score%100;  //remove the hundreds place
    tens =score/10;     //takes the tens place
    ones = score%10;    //takes the ones place
    disp_char(x, y, thou + '0');    //display thousands
    disp_char(x, y+6, hund + '0');  //display hundreds
    disp_char(x, y+12, tens + '0'); //display tens
    disp_char(x, y+18, ones + '0'); //display ones
}

////////////////////////////////////////////////////////////////////////////////
//                      Display configuration
//  This function sets up the display area for the game.
//  Prints the boarders for the play area 79x62 with two bit boarder on the outside
//  Displays high score, current player score, player 1's balls left,
```

```
//  and player two's balls left.
////////////////////////////////////////////////////////////////////////////////
void display()
{
    int i;

    if(score > high_score)  //updates high score if current score is greater
    {
        high_score = score;
    }


    for(i = 0; i < 82; i++)
    {
        screen[i] |= 3;     //Prints the top boarder
    }
    for(i = 0; i < 8; i++)  //print the side walls
    {
        screen[i*128] |= 255;       //255 is a full column on a page
        screen[i*128 + 1] |= 255;   //left two boarders
        screen[i*128 + 81] |= 255;  //right two boarders
        screen[i*128 + 80] |= 255;
    }

    //All scores and player's balls are printed on the right side of the screen.

    //Display High Score
    disp_char(0, 89, 'H');
    disp_char(0, 95, 'I');
    disp_char(0, 101, 'G');
    disp_char(0, 107, 'H');
    disp_char(0, 113, ':');

    disp_score(1, 93, high_score);

    //display current player's score
    disp_char(2, 87, 'S');
    disp_char(2, 93, 'C');
    disp_char(2, 99, 'O');
    disp_char(2, 105, 'R');
    disp_char(2, 111, 'E');
    disp_char(2, 117, ':');

    disp_score(3, 93, score);

    //Display player ones balls left
    disp_char(4, 99, 'P');
    disp_char(4, 105, '1');
    if(ball_1 == 3){
        disp_char(5, 87, '*');
        disp_char(5, 102, '*');
        disp_char(5, 117, '*');
        }
    else if (ball_1 == 2){
        disp_char(5, 87, '*');
        disp_char(5, 102, '*');
        }
    else if(ball_1 == 1) {
        disp_char(5, 87, '*');
        }

    //If more than one player display player two's info
    if(num_player == 1)
    {
```

```
        disp_char(6, 99, 'P');
        disp_char(6, 105, '2');
    }
    if(ball_2 == 3){
        disp_char(7, 87, '*');
        disp_char(7, 102, '*');
        disp_char(7, 117, '*');
        }
    else if (ball_2 == 2){
        disp_char(7, 87, '*');
        disp_char(7, 102, '*');
        }
    else if(ball_2 == 1) {
        disp_char(7, 87, '*');
        }

    //Displays "-P*-" according to who's turn it is
    if(player == 0)
    {
        disp_char(4, 93, '-');
        disp_char(4, 111, '-');
    }
    else
    {
        disp_char(6, 93, '-');
        disp_char(6, 111, '-');
    }

    //displays the screen
    refresh_screen();
}

////////////////////////////////////////////////////////////////////////////////
//                      Wait Screen
//  This function displays a screen in between players turns.
//  Prints "Player *current player* Ready"
//  Resets the speed and starts the ball in the middle of the screen.
//  Waits for the player to press the start button.
////////////////////////////////////////////////////////////////////////////////
void wait_screen(){

    TR2 = 0;    //stops the clock
    xpos = 40;  //middle of screen
    ypos = 40;  //one pixel below the bricks
    xangle = 1; //resets xangle
    yangle = 1; //resets yangle
    blank_screen();

    //Display "Player (current player) Ready"
    disp_char(2, 24, 'P');
    disp_char(2, 30, 'L');
    disp_char(2, 36, 'A');
    disp_char(2, 42, 'Y');
    disp_char(2, 48, 'E');
    disp_char(2, 54, 'R');

    if(player == 0){
        disp_char(3, 36, 'O');
        disp_char(3, 42, 'N');
        disp_char(3, 48, 'E');
    }
    else{
        disp_char(3, 36, 'T');
        disp_char(3, 42, 'W');
```

```
        disp_char(3, 48, 'O');
    }
    disp_char(4, 30, 'R');
    disp_char(4, 36, 'E');
    disp_char(4, 42, 'A');
    disp_char(4, 48, 'D');
    disp_char(4, 54, 'Y');

    display();  //Display boarders and scores

    //Checks if both players lives are at 3 if so it's the start of the game
    if(ball_1 == 3 && ball_2 == 3){
        while (button == 1){
        pad_w = P1&3;   //set player1 paddle size with switches (1 and 2)
        pad_w2 = P1&96; //set player2 paddle size with switches (6 and 7)
        pad_w2 = pad_w2 >>5; //shift player2 so that it is between 0 to 3

        //sets the initial speed of the ball
        t_cnt_init = P1&24; //switches 4 and 5
        t_cnt_init = t_cnt_init >>3; //shift so it's between 0 to 3

        num_player = player_sw; //One or two players based on switch 8;

        }

        //sets the paddle width for 24, 16, 12, and 8 bits
        if(pad_w == 3){
            pad_w = 24;
        }
        else{
            pad_w = pad_w2*4 +8; //(0 to 2)*4 +8 = 16-8
        }
        if(pad_w2 == 3){
            pad_w2 = 24;
        }
        else{
            pad_w2 = pad_w2*4 +8; //(0 to 2)*4 +8 = 16-8
        }

        //initialize speed of the ball
        t_cnt_init = t_cnt_init*10 + 30; //(0 t0 3)*10 + 30 = 30 - 60
        t_cnt = t_cnt_init;             //set current count to the inital
        if(num_player == 0){
        ball_2 = 0; //no two player remove their lives
        }
    }
    else{
        while (button == 1){}; //wait till button is pressed
    }
    TR2 = 1;    //turn timer back on
}

////////////////////////////////////////////////////////////////////////////////
//                      Game Over Screen
//  This function displays a screen at the end of the game
//  Prints "GAME OVER"
//  Waits for the reset button to be pressed
////////////////////////////////////////////////////////////////////////////////
void game_over()
{
    TR2 = 0; //turn timer off

    blank_screen(); //clear screen
    //print Game Over
```

```
    disp_char(2, 30, 'G');
    disp_char(2, 36, 'A');
    disp_char(2, 42, 'M');
    disp_char(2, 48, 'E');
    disp_char(3, 30, 'O');
    disp_char(3, 36, 'V');
    disp_char(3, 42, 'E');
    disp_char(3, 48, 'R');

    display(); //display boarders and scores
    while(1){} //Stays until board is reset

}

////////////////////////////////////////////////////////////////////////////////
//                        Turn Over Screen
//  This function displays a screen at the end of one players turn
//  Switches variables for the next player and checks the end game condition
//  Sends player to the wait screen or game over screen
////////////////////////////////////////////////////////////////////////////////
void turn_end()
{
    //Audio for ball touching the bottom
    RCAP4H = -1763>>8;          // set up for 784Hz
    RCAP4L = -1763;             // set up for 784 Hz
    duration = 196;             // quarter second
    T4CON = T4CON^0x04;         //enable timer 4

    //speed reset
    su = 0;                 //reset speed up for 3rd brick
    t_cnt = t_cnt_init;     //reset speed to original speed

    //Check if it's a two player game
    if(num_player == 1){
        //if it was the first players turn and they lost
        if(player == 0){
            ball_1 = ball_1 -1; //decrease lives by one
            player = 1;         //set player to second player
            score_1 = score;    //Save player One's current score
            score = score_2;    //Set the score to player two's
        }

        //if it was the second players turn and they lost
        else{
            //check game over condition no more lives left
            if(ball_2 == 1){
                ball_2 -= 1;
                game_over();}

            else{
                ball_2 = ball_2 -1; //decrease lives by one
                player = 0;         //set player to first player
                score_2 = score;    //Save player Two's current score
                score = score_1;    //Set the score to player One's
            }
        }
    }
    //If it's a single player game
    else{
        if(ball_1 == 1){ //Check for a game over
            game_over();}
        else{
            ball_1 = ball_1 -1; //Decrease lives
            }
```

```
    }
    wait_screen(); //Send to wait screen
}

////////////////////////////////////////////////////////////////////////////////
//                        Draw the Bricks
//  This function takes the arrays of bricks and draws them accordingly on the
//  screen. A one represents a brick to be drawn and a zero represents a brick to
//  be deleted. Bricks are 11x5
////////////////////////////////////////////////////////////////////////////////
void draw_bricks(){
    int i;
    int j;
    int k;
    int zero_cnt = 0;
    xdata unsigned char blocks[11][5] = {0};

    //Sets the bricks to be drawn dependent on the player
    if(player == 0)
    {
        for(i= 0; i < 11; i ++)
        {
            for(j = 0; j < 5; j ++)
            {
                blocks[i][j] = blocks_1[i][j];
            }
        }
    }
    else
    {
        for(i= 0; i < 11; i ++)
        {
            for(j = 0; j < 5; j ++)
            {
                blocks[i][j] = blocks_2[i][j];
            }
        }
    }

    //Loops through the bricks and draws them or deletes them from the screen
    for(i= 0; i < 11; i ++)
    {
        for(j = 0; j < 5; j ++)
        {
            if(blocks[i][j] == 1) //If there is a block at this location
            {
                if(j%2 == 1) //If y value is even draw on the first half of the page
                {
                    for(k = 0; k < 6; k++) //draws the brick 6 wide
                    {
                        screen[((j-1)/2)*128 + i*7 +k +131] |= 0x70; //Three bits at the t
op of the page with one space on top
                        //((j-1)/2)*128 = row to print, i*7 +131 = column, +k does it 6 ti
mes for the brick
                    }
                }
                else //If y value is odd draw on the second half of the page
                {
                    for(k = 0; k < 6; k++) //draws the brick 6 wide
                    {
                        screen[(j/2)*128 + i*7 + k +131] |= 0x07; //Three bits on the bott
om of the page with one top space
                        //((j/2)*128 = row to print, i*7 +131 = column, +k does it 6 times
 for the brick
```

```
                    }
                }
            }
            else
            {
                zero_cnt ++; //keeps track of the cleared bricks allowing the screen to re
populate at 55
                if(j%2 == 1)
                {
                    for(k = 0; k < 6; k++)
                    {
                        screen[(j-1)/2*128 + i*7 + k + 131] &= 0x07; //ands the top with z
ero to clear desired brick
                        //see above for math of drawing blocks

                    }
                }
                else
                {
                    for(k = 0; k < 6; k++)
                    {
                        screen[j/2*128 + i*7 + k + 131] &= 0x70;    //ands the bottom with
 zero to clear desired brick
                        //see above for math of drawing blocks
                    }
                }
            }
        }
    }
    if(zero_cnt == 55 && ypos > 40){    //Check if all the blocks have been cleared
        if(player == 0) //reset bricks of player 1
        {
            for(i= 0; i < 11; i ++)
            {
                for(j = 0; j < 5; j ++)
                {
                blocks_1[i][j] = 1; //loops and sets each value to one
                }
            }
        }
        else //reset bricks of player 2
        {
            for(i= 0; i < 11; i ++)
            {
                for(j = 0; j < 5; j ++)
                {
                blocks_2[i][j] = 1; //loops and sets each value to one
                }
            }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////
//                      Draw the Paddle
//  This function takes the current pot value passed in and draws the paddle
//  according to the paddle width selected.
//
////////////////////////////////////////////////////////////////////////////////
void draw_paddle(char x)
{
    int i;
    int padd_w;
```

```
    //Sets padd_w depending on who's turn it is
    if(player == 0){
        padd_w = (int)pad_w;

    }
    else{
        padd_w = (int)pad_w2;
    }

    //Draw the paddle
    for(i = 0; i < padd_w; i ++)
        {
            screen[898+x+i] |= 0xc0; //898 is the bottom left corner of the play area 0xc0
 is a paddle 2 bits thick
            //x offsets the paddle to the knob position, i then draws the paddle padd_w wi
de.
        }
}

////////////////////////////////////////////////////////////////////////////////
//                        Draw the Ball
//  This function does a lot:
//      -Draws the ball
//      -Bounces the ball off walls
//      -Bounces the ball off the paddle
//          -changes angle based on bounce off paddle
//      -Detects hits on bricks and breaks the corressponding brick
//      -Returns a hit code that determines which audio is played
//
////////////////////////////////////////////////////////////////////////////////
unsigned char draw_ball(int x, int y)
{   unsigned char row, col, shift, j, hit;
    int i;

    col = x-2; //correct for center of ball
    row = y - 2; //correct for center of ball
    shift = row%8;
    row = row/8;
    hit = 0; //initialize hit to zero

    //draw the ball and return a value greater than zero if a hit occurs
    for(j = 0, i = row*128+col; j < 5; i++, j++)
    {
        int mask = (int)ball[j] << shift;
        hit |= screen[i]&(unsigned char)mask;
        screen[i] |= mask;

        if(mask & 0xFF00)
        {
            hit |= screen[i+128]&(unsigned char)(mask >> 8);
            screen[i + 128] |= (unsigned char)(mask >> 8);
        }
    }

    //Checks if the ball is gonna hit the corner of the screen and inverts both angles
    if((x<5 || x > 78) && y < 3)
    {
        yangle = -1*yangle;
        xangle = -1*xangle;
        return -2;//audio code
    }
    //checks the left and right edges of the screen and inverts xangle if hit
    else if(x<5 || x > 78)
    {
```

```
        xangle = -1*xangle;
        return -2;//audio code
    }
    //checks the top of the play area and inverts y if hit
    else if (y < 3)
    {
        yangle = -1*yangle;
        return -2;//audio code
    }
    //checks the bottom of the screen and ends the game if hit
    else if(y > 61)
    {
        turn_end();
        return -3;
    }

    //Paddle hit detection and subsequent angle configs for different parts of the paddle.
    //This code devides the paddle into 4 sections and returns the angle as needed
    if( y == 60 && hit > 0){    //The only place the ball can hit the paddle is at y = 60
and if the ball detected a hit that means it hit the paddle

        char col = xpos - pot_val -2; //takes xposition of the ball subtracts the pot val
and -2 to tell where the ball is in relation to the paddle
        int div = 0;
        //Calculates the paddle division based on player and the size of their paddle.
        if(player == 0){
            div = pad_w/4;  //paddle separated into four sections
            }
        else{
            div = pad_w2/4; //paddle separated into four sections
            }
        if( col < div || col > 3*div) //did the ball hit the outside quarters of the paddl
e?
        {
            xangle = 2; //If so return a steeper angle
            yangle = -1;
        }
        else
        {
            xangle = 1; //If it hit the center return a shallower angle
            yangle = -2;
        }
        if(col < div *2) //did the ball hit the left side of the paddle?
        {
            xangle = -1*xangle; //If so send the ball to the left
        }
        return -1;//audio code
    }

    //All brick breaking logic, if it didn't hit the paddle or the sides, you hit a brick
    else if( hit > 0)
    {

        int x_b, y_b;
        score += 1; //hit a brick increase the score!

        //Normalize the y_position to return a value inside of brick matrix = (ypos -8)/4
        //Normalize the x_position to return a value inside of brick matrix = (xpos -3)/7
        if(yangle < 0)
        {
            y_b = (ypos -10)/4; //ball is moving up so normalize to the top edge of the ba
ll with extra -2
        }
        else{
```

```
            y_b = (ypos -6)/4;  //ball is moving down so normalize to the bottom edge of t
he ball with extra +2
        }
        if(xangle < 0)
        {
            x_b = (xpos -5)/7; //ball is moving left so normalize to the left edge of the
ball with extra -2
        }
        else
        {
            x_b = (xpos -1)/7;  //ball is moving right so normalize to the right edge of t
he ball with extra +2
        }

        if(y_b > 4 || x_b > 10){ //If the x_b and y_b positions aren't in the bounds of th
e matrix ignore it
        }

        //Update players bricks according to the brick that was hit
        else if(player == 0)
        {
            if(blocks_1[x_b][y_b] == 0) //Check if the current index of the ball has a bri
ck in it, if not it's probably on one of the sides
                {
                    if(xangle < 0) //ball is moving left so delete the ball to the left
                        {
                            y_b = (ypos -8)/4;  //Normalize position to the center ball
                            blocks_1[x_b-1][y_b] = 0; //delete the ball to the left
                        }
                    else
                        {
                            y_b = (ypos -8)/4;  //Normalize position to the center ball
                            blocks_1[x_b+1][y_b] = 0;   //delete the ball to the left
                        }
                    xangle = -1*xangle; //bounce the ball to the side
                }
            else
            {
                    blocks_1[x_b][y_b] = 0;//brick hit the top, delete that brick
                    yangle = -1*yangle; //bounce the ball up or down
            }
        }
        //Same logic described above is used for player 2's bricks
        else if(player == 1)
        {
            if(blocks_2[x_b][y_b] == 0)
            {
                if(xangle < 0)
                {
                    y_b = (ypos -8)/4;
                    blocks_2[x_b-1][y_b] = 0;
                }
                else
                {
                    y_b = (ypos -8)/4;
                    blocks_2[x_b+1][y_b] = 0;
                }
                    xangle = -1*xangle;
            }
            else
            {
                if(x_b >= 0 || y_b >= 0){
                    blocks_2[x_b][y_b] = 0;
                    yangle = -1*yangle;
```

```
                }
            }
        }

        //checks if the ball is hitting the third row of bricks for the first time
        if(hit > 0 && ypos <= 21 && su ==0) //21 is the first row in which the ball could
hit the 3rd row of bricks
        {
            t_cnt = t_cnt*0.6; //speed up the ball movement by a factor of .4
            su = 1;             //set the flag so it doesn't continuously speed up
        }
    }
    return hit; //return hit value mainly used for audio que
}

////////////////////////////////////////////////////////////////////////////////
//                        Move the Ball
//  This function updates the balls postition based on the current x and y angle
//  and then calls draw ball and stores the audio que in a variable called bonk
//
////////////////////////////////////////////////////////////////////////////////
void mov_ball() {
    xpos += xangle; //update xpos by xangle
    ypos += yangle; //update ypos by yangle
    bonk = draw_ball(xpos, ypos);   //saves audio code in bonk and draws the ball
    display();  //update the display
}

////////////////////////////////////////////////////////////////////////////////
//                        Timer2 Interrupt
//  This interrupt updates the int_cnt which will determine how often the ball
//  is updated and drawn which in turn effects the speed of the ball.
//
////////////////////////////////////////////////////////////////////////////////
void timer2(void) interrupt 5
{

    TF2 = 0;    //reset the timer flag
    int_cnt ++; //increase the timer count

}




void main()
{
    WDTCN = 0xde; //disable watchdog
    WDTCN = 0xad;
    XBR2 = 0x40;  //port output
    OSCXCN = 0x67; //crystal enabled
    TMOD = 0x20;  //wait 1ms: T1 mode 2
    TH1 = 167;    // 1ms/(1/(2Mhz/12)) = 166.666
    TR1 = 1;      //enable timer 1
    while( TF1 == 0) {} //1 ms wait for flag
    while( !(OSCXCN & 0x80)) {} //stabilize crystal
    OSCICN = 8;   // switch to 22.1184Mhz

    IE = 0xA0;   //Timer 2 interrupt enable
    EIE2 = 0x06; //Timer 4 and ADC

    T4CON = 0x00; //timer 4, auto reload
    RCAP4H = -1;    //timer 4
```

```
    RCAP4L = -144;  //timer 4
    DAC0CN = 0x9C;  //Speaker setup

    T2CON = 0x00;   //set up timer 2
    RCAP2H = -2211 >> 8;
    RCAP2L = -2211;

    ADC0CN = 0x8c;  //configure the ADC for the potentiometer
    REF0CN = 0x07;
    ADC0CF = 0x40;  //Configure the ADC for the audio
    AMX0SL = 0x0;
    CKCON = 0x40;   //Turn off divide by 12 for timer 4

    TR2 = 1;            //Turn on Timer 2
    T4CON = T4CON^0x04; //Turn on Timer 4

    init_lcd(); //initalize the LCD
    xpos = 40;//middle of screen
    ypos = 40;//one pixel below the bricks


    display(); //update display
    wait_screen(); //send game to wait for start



    while(1)
    {
        if(int_cnt==t_cnt) //check the clock's count with the desired update count
        {
            //Main gameplay loop
            blank_screen();         //blank screen
            draw_paddle(pot_val);   //update the paddle
            draw_bricks();          //update bricks
            mov_ball();             //move the ball
            int_cnt = 0;            //reset the count

            //Audio setup for brick breaking
            if(bonk > 0){
                RCAP4H = -2097 >> 8;        // set up for 659Hz
                RCAP4L = -2097;             // set up for 659Hz
                duration = 165;             //quarter second
                T4CON = T4CON^0x04;         //enable timer 4
            }
            //Audio for bouncing off paddle
            else if(bonk ==-1)
            {
                RCAP4H = -2642 >> 8;        // set up for 523Hz
                RCAP4L = -2642;             // set up for 523Hz
                duration = 131;             //quarter second
                T4CON = T4CON^0x04;         //enable timer 4
            }
            //Audio for bouncing off wall
            else if(bonk == -2)
            {
                RCAP4H = -2354>>8;          // set up for 587Hz
                RCAP4L = -2354;             // set up for 587Hz
                duration = 147;             //quarter second
                T4CON = T4CON^0x04;         //enable timer 4
            }
        }
    }
}
```

Appendix C:

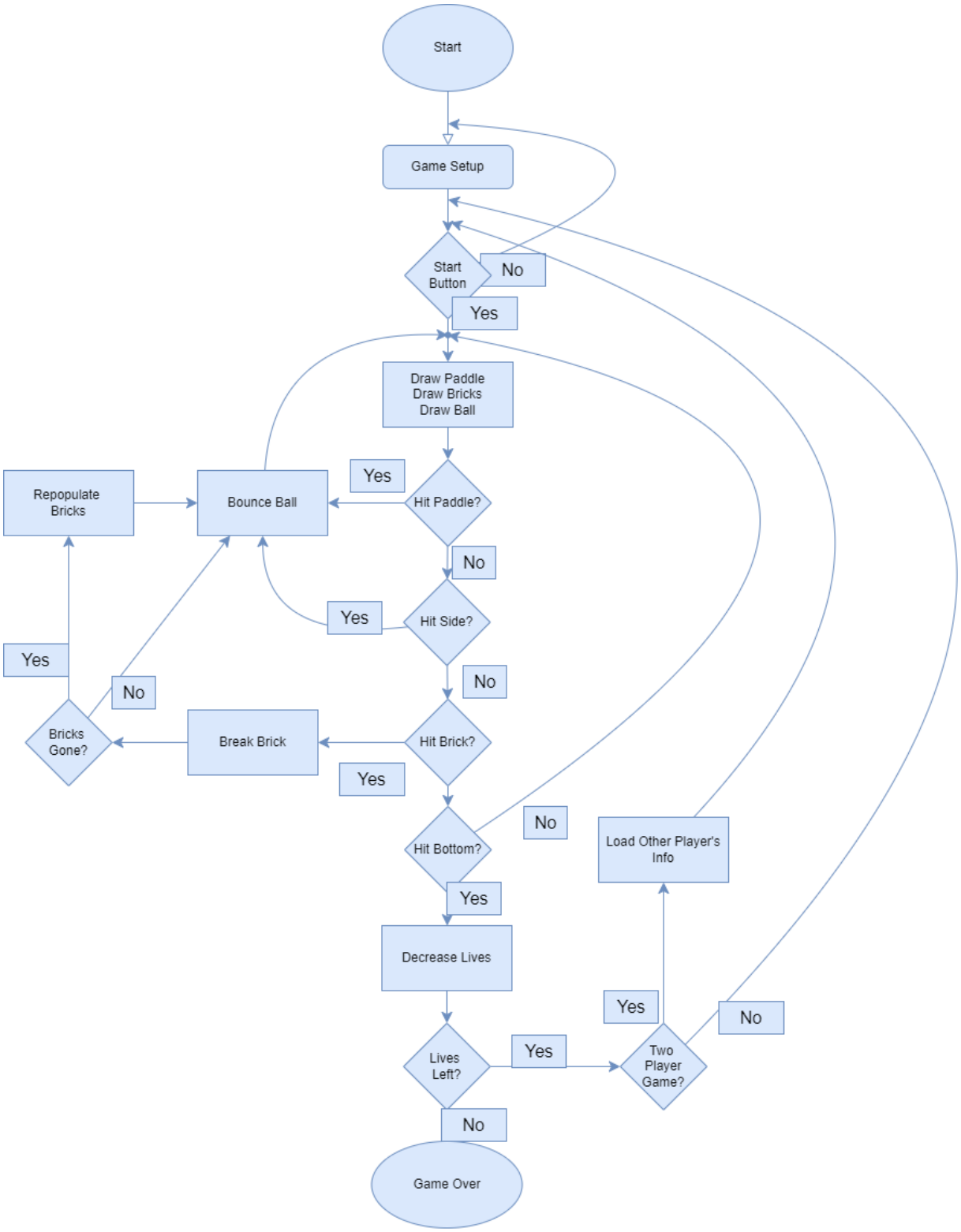

Figure 5: Software Flow Diagram