Gorilla

Design Documentation

Table of Contents

Table of Figures

0.0 - Introduction

This document describes a variation of Microsoft's GORILLA, which was released with MS-DOS 5 in 1991. GORILLA is a two player turn-based game. Players begin at a random location atop opposing skyscrapers along a city skyline and take turns lobbing explosive bananas at each other until one of the players is destroyed. Random skylines and winds present additional challenges to players as they attempt to vanquish their opponents. This variation of GORILLA is different from the original in that it is designed to be played on a portable hand held device.

1.0 - Scope

This document describes the software and hardware design of the GORILLA hand-held video game. It includes the requirements, dependencies, and theory of operation. Design details and requirement validation testing are described at length. Complete software code and schematics are included in appendices at the end of this document.

2.0 - Design Overview

2.1 Requirements

The given requirements for the GORILLA video game are as follows:

1. The system shall run on an external 9v DC supply.
2. The system shall use a 64x128 pixel LCD to display the skyline, gorillas, wind speed indicator, launch speed indicator and launch angle indicator.
3. The system shall have two buttons. One reset button and one launch button.
4. The system shall have two potentiometers located near the LCD that can be used to adjust banana launch speed and angle.
5. The system shall have two DIP switches to adjust difficulty, one for each player. If a player's switch is off, the wind speed shall be set to zero when he launches his banana.
6. Upon reset, the system shall display the words "Ready" and "Press Launch Button to Start" in the center of the LCD display. The skyline and gorillas may be displayed in the background but it is not required.

7. When the launch button is pressed initially, the system will randomly select and display
    a. the skyline
    b. the wind speed
    c. the gorilla positions

8. Player 1 shall always play the leftmost gorilla and shall have the first turn. The rightmost gorilla is played by player 2. The system shall indicate whose turn it is.

9. A turn shall consist of a player
    a. adjusting launch speed and angle using the knobs
    b. pressing the launch button

   At this point, the system shall animate a banana launched from one gorilla that follows a trajectory consistent with requirement.

10. The turn ends when the banana
    a. strikes a building in the skyline and explodes
    b. strikes either gorilla and explodes
    c. traverses the left or right boundary of the LCD display

11. The velocity of the banana shall be Vx = Vxo + Wt, Vy = Vyo - gt where (Vxo,Vyo) is the initial velocity, w is proportional to the wind speed, and g is constant. If the banana goes off the top of the LCD display, the system shall continue to compute its trajectory but will not display it until its position is once again within the bounds of the LCD display.

12. The launch speed and angle must be accurately indicated on the LCD display while adjustment are being made.

13. The system shall generate a sound when a banana is launched and a different sound when the banana explodes.

14. When a banana hits either gorilla, the game is over, and "Game Over" and "Press Launch Button to Start" shall be displayed in the middle of the LCD display. The remaining gorilla and skyline may be displayed in the background, but it is not required. The system will remain in this state until the launch button is pressed, at which point a new game begins with a random skyline, wind speed and gorilla position.

15. When a banana hits the skyline and explodes, it may either erode the skyline or leave it unchanged.

2.3 Theory of Operation

Upon reset or power-up, the system will display a start-up screen on a 64x128 pixel LCD with instructions to begin the game. Players will have the option to set the difficulty of the game using a DIP switch to turn winds on or off during their turn. Once the "launch" button is pressed, the game begins. A city skyline is then drawn, made up of 8 adjacent skyscrapers, each at random heights of up to five building segments (Figure 1:E). Also, a random wind speed between 0 and 4 pixels per frame (Figure 1:C) and direction (Figure 1:D) is determined and displayed in the upper right hand corner of the LCD. Each player's gorilla will be placed on one of three buildings on their respective side of the screen. The two buildings in the center will act as a "no-man's-land", to alleviate an unfair advantage in the event that two players would be placed right next to each other. Player 1 will always be positioned on the left side of the screen and will go first. Player 2 will follow. A banana will be positioned above a player indicating their turn. Each player will continue taking turns until one of the gorillas is destroyed.

Two potentiometers are used to select a trajectory and initial velocity for the banana. An indicator in the top left corner of the LCD will display the angle from 0 to 90 degrees (Figure 1:B), with 0 degrees being the horizontal direction and 90 degrees being vertical; and an initial speed between 0 and 14 pixels per frame (Figure 1:A) for the banana's trajectory.

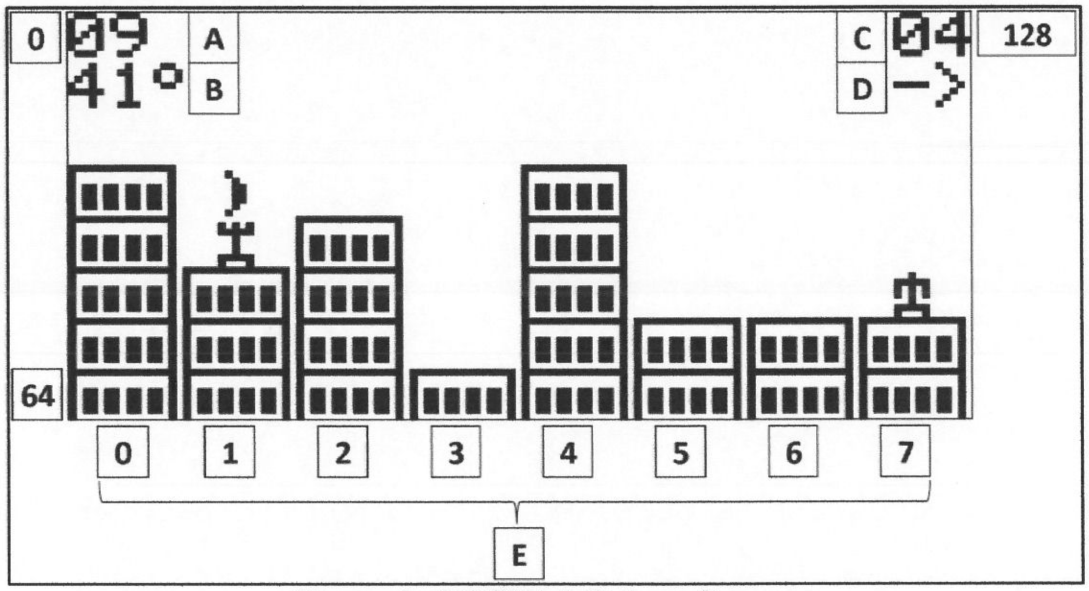


Figure 1: GORILLA Set-up Screen

Pressing the launch button will initiate a throw, causing the banana to move across the screen according to Newton's law of motion, taking into account gravity, initial trajectory, and the wind vector. An audible "beep" will also accompany the throw sequence. Should the banana move beyond the top of the LCD, it's trajectory will continue to be computed, but it will not be displayed until it returns back into the viewable portion of the screen. If the banana exits either side of the screen (Figure

2: A) the turn will end and the next player will begin their turn. If the banana makes contact with the skyline, it will destroy the building segment which it comes into contact with and shorten that skyscraper by one segment (Figure 2: C). An audible "crash" sound will accompany the destruction of a building segment. If the banana hits one of the gorilla's strike zone, the game will end and issue a short melody.

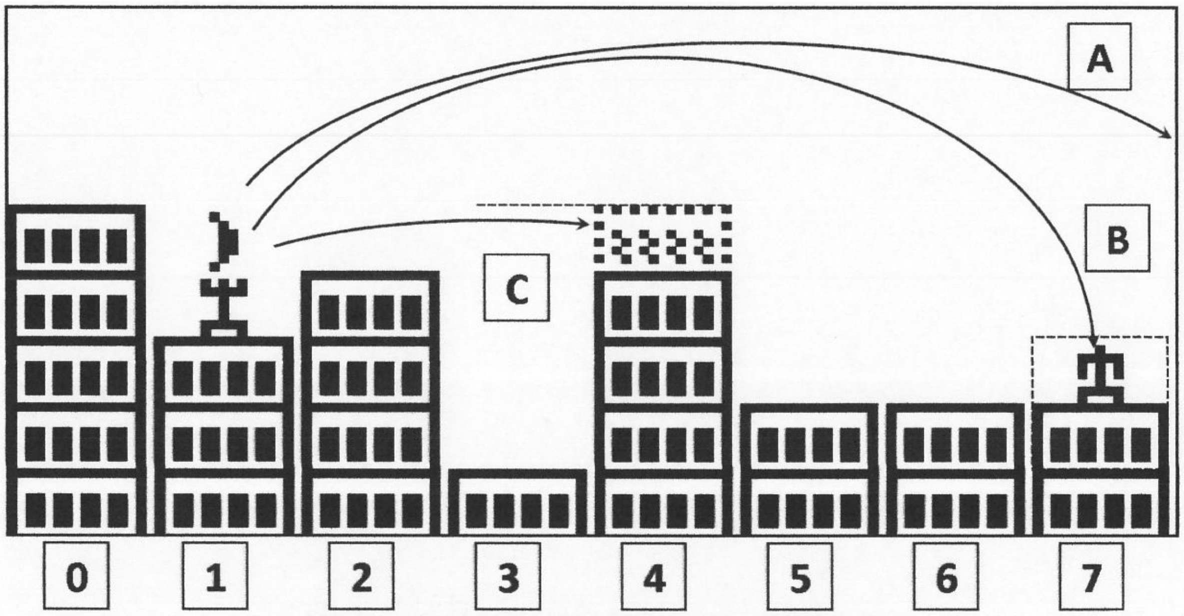


Figure 2: GORILLA Game Play Screen

Figure 3 contains a flow chart describing the overall game process.

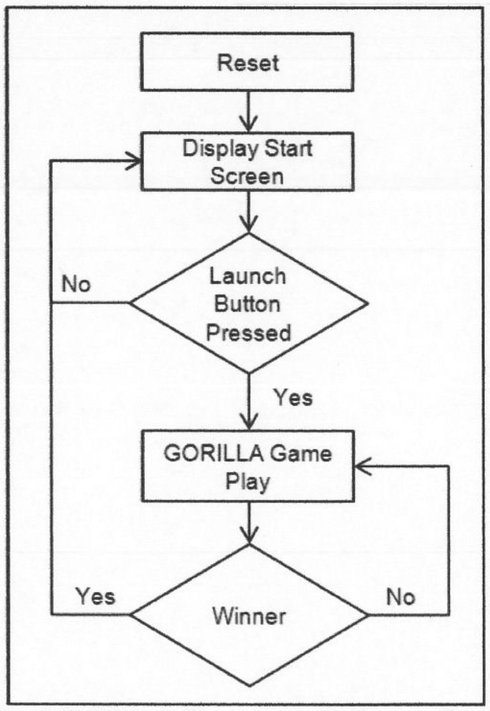


Figure 3: GORILLA Top Level Game Flow Chart

3.0 - Design Details

3.1 Hardware

GORILLA is implemented on an C8051F020 microcontroller development board and a customized daughter board with user interfaces; including an LCD, two (0 - 50kΩ) potentiometers, two push buttons, and eight 2-position DIP switches. The LCD is an ST7565R (65x132) dot matrix LCD Controller/Driver, which was manufactured by Sitronix. Only two of the eight DIP switches are used for GORILLA. Each player will use their DIP switch to select a difficulty level by turning on/off crosswinds. The potentiometers are read via a 12-bit ADC, which is built into the C8051F020, and are used to select the banana's trajectory and speed. Game sounds are produced by an AST-03208MR-R (8Ω) speaker and a TDA7052 amplifier. See Section 6 for a detailed schematic.

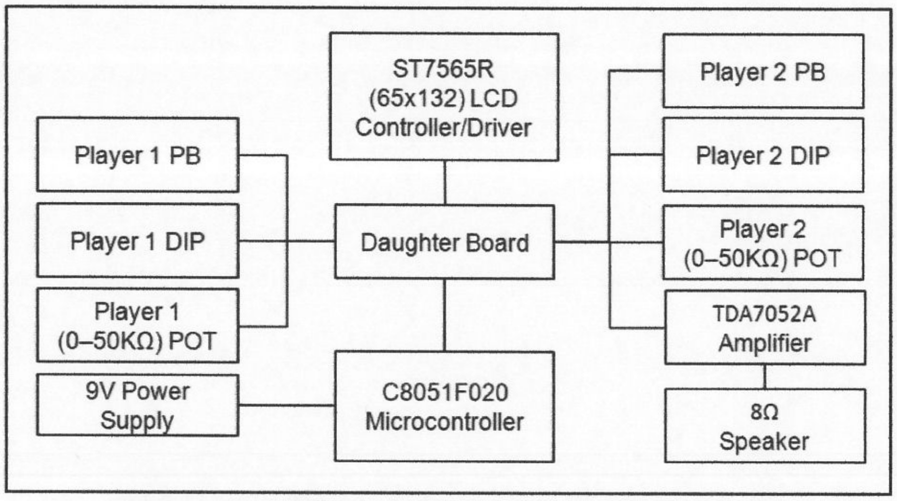


Figure 4: GORILLA Hardware Block Diagram

Figure 3-5: GORILLA Hardware Block Diagram

3.2 Software

The software for GORILLA is written in both C and Assembly. Functions used to control the LCD are written in Assembly and are ported into C. The C portion is divided into six functional segments; set-up, drawing, tracking, monitoring, outputting, and determining the winner.

The set-up segment includes clearing the screen and prompting the user to start, it also includes displaying "Game Over" when a game is complete and again prompting the user to start. The drawing segment consists of drawing the skyline, the gorillas, and the current position of the banana. Drawing also consists of displaying the text for the banana trajectory and wind vector. See section 3.2.1 for details on drawing the buildings, gorillas, and banana's.

The tracking segment covers computing and tracking the position and velocity of the banana and keeping track of the current location of the two gorillas. See section

3.2.2 for details on tracking the bananas. The monitoring segment involves monitoring all inputs; such as the potentiometers, the DIP switches, and the push-buttons. The outputting segment produces game sounds at the appropriate time and sends commands to the LCD to clear and refresh the screen at a rate of 60hz. The last functional segment is self-explanatory. When either of the gorillas is destroyed, this segment determines which gorilla won and prompts users to start a new game. Figure 6 contains the software flowchart illustrating how each of the segments is executed and how they are related to all of the other segments.

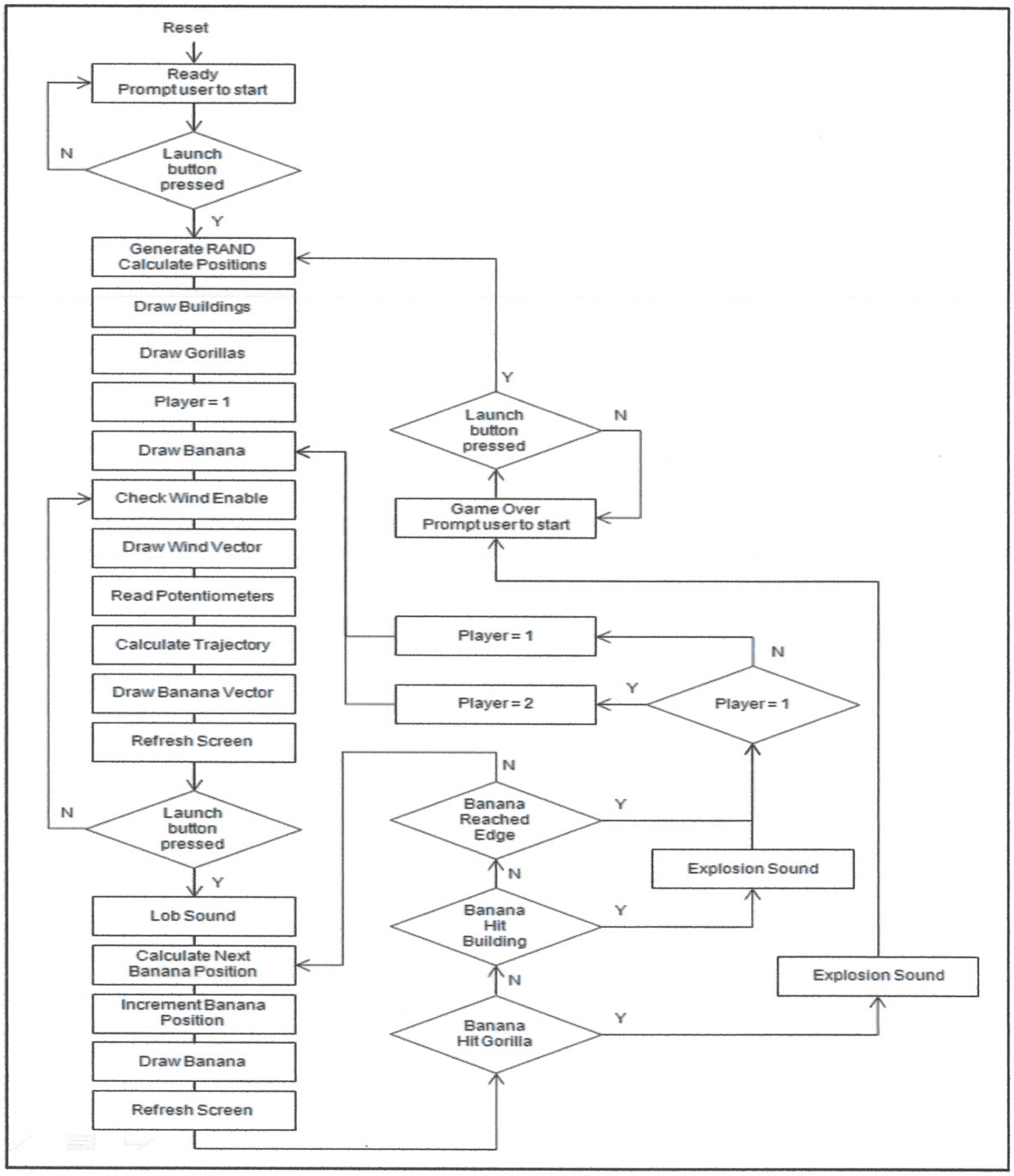
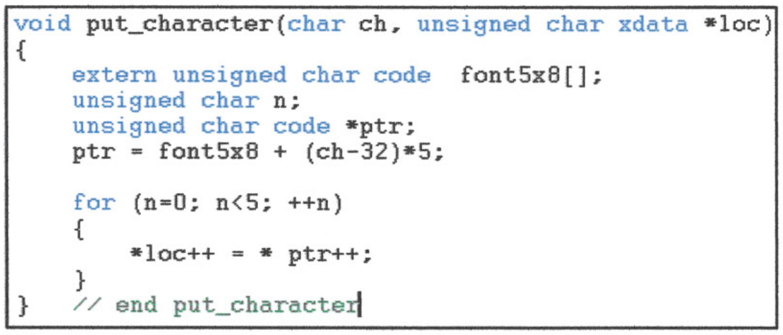


Figure 6: GORILLA Software Flow Chart

3.2.1 - Drawing the Characters

Text characters are handled with the put_character() function provided by the LCD.asm program. This program links the ASCII table in the LCD.asm program to the 1040 bytes of external memory allocated for the LCD display. A simple pointer conversion is used to transfer characters from the table to the display memory. A 5x8 font is used to display ASCII characters.



```
void put_character(char ch, unsigned char xdata *loc)
{
    extern unsigned char code  font5x8[];
    unsigned char n;
    unsigned char code *ptr;
    ptr = font5x8 + (ch-32)*5;

    for (n=0; n<5; ++n)
    {
        *loc++ = * ptr++;
    }
}   // end put_character
```

Figure 7: put_character Function

To handle drawing the gorillas and buildings, simple characters where created and ASCII characters were hijacked from an ASCII table in the LCD 5x8 ASCII table. Figure 3-2 shows the pixel design of the gorilla and building segments along with their equivalent ASCII characters. Each character uses the same 5x8 font characteristics as the ASCII characters. This way it is easy to use the same put_character() function for writing text to place the building segments and the gorillas.

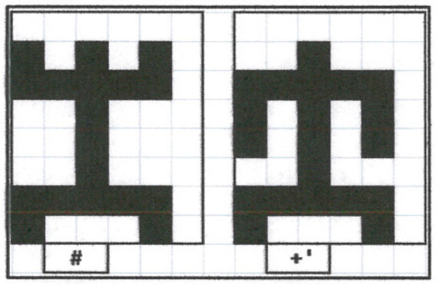


Figure 8: Gorilla Characters

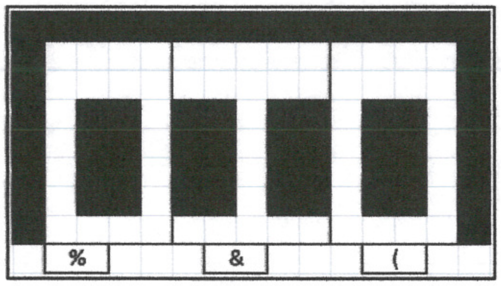


Figure 9: Building Characters

To draw the buildings at the appropriate height, an 8 element array is created to hold the random 1 to 5 segment heights for each of the 8 skyscrapers. Buildings begin at the last 16 pixel segment of the display (bottom right) and are written from right to left, bottom to top. As each segment is written, the corresponding array value is decremented. This process takes place until all of the array values have reached zero, meaning no more building segments. Gorillas are then position in one of their three possible positions. They are placed at a height equal to the height of the skyscraper in the given lateral position.

The banana characters employ a slightly different method for writing. They utilize the same 5x8 font employed for regular characters, but they are written with a different function. This is because the banana pixels are written to the display with a logic "OR" to allow the banana to appear in the foreground instead of erasing the background. See the put_banana() function in section 6 for details. Bananas are initially placed above the player who's turn it is.

3.2.2 - Making the Banana Move

To make the banana appear to move upward, it is necessary to draw 16 positions of the banana. This is because the banana needs to move upward one pixel at a time. Since each character column is 8 pixels tall, it is necessary to span two pages while making the banana move upward. This is done by first calculating the Y-position of the banana and dividing it by 8 to get the initial page value. The modulus is then taken to get the pixel height of the bottom of the banana relative to the current character position. If the modulus is zero, character 128 is chosen which is the banana's first position. For each modulus value, the character position is incremented up to eight times. If the page value is greater than one page, then a character eight positions greater than the lower page character is written onto the second page which corresponds to the character on the page below. For example, if character '130' is written onto page one, then character '138' is written onto page two. Figure 10 illustrates how the characters are drawn and stored in memory. They are positioned on top of each other to demonstrate how they are divided between the two pages.

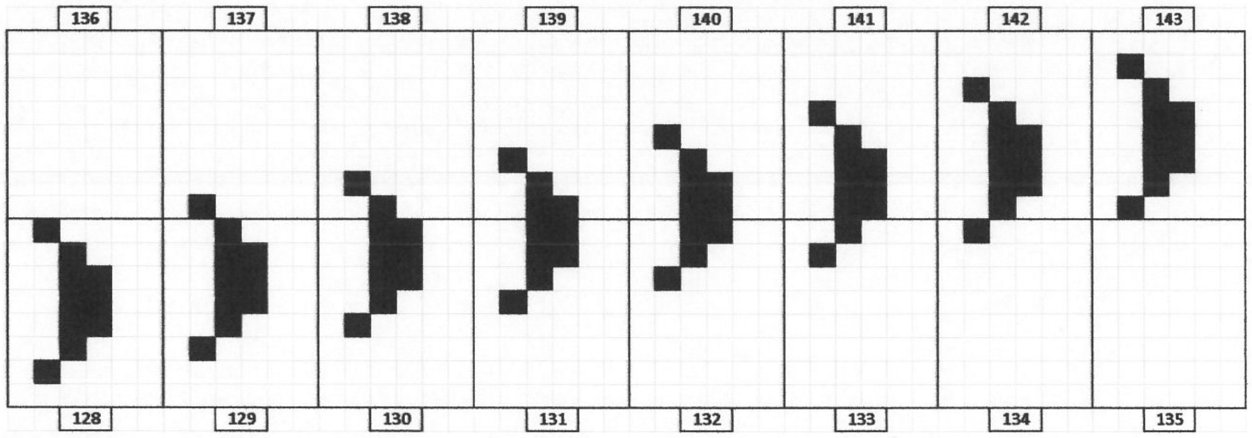


Figure 10: Banana Graphics for Motion

The code for moving the banana can be found in section 6, the appendix, under the function named display_banana(). This provides a simple way to make the banana appear to move up while incrementing the Y-position only one pixel at a time. Making the banana move in the X-direction is much simpler and only requires that the character location be moved one pixel at a time.

It is possible to calculate the trajectory by simply calculating the X and Y values separately and then combing the two into a single vector. The following figure contains code snippets used to calculate the trajectory of the banana.

```
banana_Vx = (x_factor*speed)/18;     // initial x velocity
banana_Vy = (y_factor*speed)/18;   // initial y velocity
```

```
void calculate_trajectory(void)
{
    if(player == 1)
        {
            banana_x = (banana_x + banana_Vx + wind_speed);
        }
    else
        {
            banana_x = (banana_x - banana_Vx + wind_speed);
        }

    banana_y = (banana_y + banana_Vy - gravity*time);
}
```

Figure 11: Banana Vector Code

A more simple method was chosen instead of creating a trigonometric function or even a trig table for calculating the relative magnitude of the X and Y directions. The X and Y magnitudes are derived directly from the angle potentiometer. The Y value ranges from 0 to 90 and the X value is calculated as (90 - Y value). The overall values are then divided by 90 to get inversely proportionate values. Thus, if the angle is set to 90 degrees, then the Y factor is '1' and the X factor is '0'. Likewise, when the angle is set to 0 degrees, then the X factor is '1' and the Y factor is '0'. When they are equal to each other, both are 1/2 which corresponds to 45 degrees. Thus, the banana velocity in the X and Y directions is calculated by multiplying the X and Y factors, respectively, by the speed. The wind speed is factored into the X velocity and the effects of gravity with respect to time are factored into the Y velocity. Once this is accomplished, the X and Y positions are tabulated by simply adding the current position to its respective velocity every frame.

3.2.3 - Detecting Boundary Collisions

It is necessary to monitor the position of the banana with respect to other objects in the game; such as buildings, screen edges, and gorillas. To check that the banana is within the bounds of the display, it is simply necessary to compare the current position to the dimensions of the display. However, to make sure that the banana doesn't appear on the wrong side of the display in the X direction, it is necessary to limit the X dimension to 0 to 125 pixels. This is because the banana is three pixels wide and will begin to appear on the left side of the screen if the banana position is allowed to be greater than 125. The Y position is limited between 0 and 64 pixels. If greater than 64, it is only necessary to stop drawing the banana. If less than 0, it is necessary to end the turn. Figure 12 contains a snippet demonstrating how to monitor the position of the banana.

```
if((banana_y > 0) && (banana_y < 63) && (banana_x > 0) && (banana_x < 125))
    {
    display_banana(banana_x, banana_y);
    }

if((banana_x < 0) || (banana_x >128) || (banana_y < 1))
    {
        collision_flag = 1;
    }
```

Figure 12: Screen Boundary Code

The method for checking if the banana has collided with a building consists of comparing the current 'lot' position of the banana, that is its position divided by 16, which is the same width of a building. Once the lot position is determined, the height of the building at that position is compared to the Y value of the banana's position. If the banana's position is higher it is above the building. If it is lower, then it has collided with the building and a collision flag is set. The banana's position is compared to every building to check if it has collided with one of them. This is illustrated in Figure 13.

```
for(n=0;n<8;n++)
    {
        if((banana_x >= (n*16)) && (banana_x <= (n*16 +16)) && (banana_y <= skyline[n]*8))
        {
            skyline[n]--;
            collision_flag = 1;
            hit = 1; // play hit sound for 200 cycles
            duration2 = 200;
        }
    }
```

Figure 13: Building Boundary Code

The same methods are used to detect when the banana has hit one of the gorillas. The only difference is that a strike zone is defined for the gorillas which moves with the gorillas as they are placed in random locations. The gorilla strike zone is defined as a 16 pixel by 16 pixel square which surrounds the gorillas and covers the building segment upon which it is standing. The banana position is defined as the lower left pixel of the banana. See Figure 14 for details. As the banana moves, its position is compared to the strike zones of each of the gorillas. If it moves inside one of those boundaries a collision flag is set, a winner flag is set, and the game ends. The 16x16 strike zone is important because it limits the speed at which the banana can move. If the banana is allowed to move faster than 16 pixels per calculation, it would be possible for the banana to pass through a gorilla or building without hitting it. The 16x16 boundary can also effect game play. It is possible for the gorilla to throw the banana at an angle of 0 degrees, but it is necessary to throw faster than 8 pixels per calculation, otherwise gravity will pull the banana into the strike zone and the gorilla will effectively commit suicide. However, players will learn to cope with this limitation in the same manner that they have to cope with the mistake of throwing the banana straight upward only to have it return to their feet.

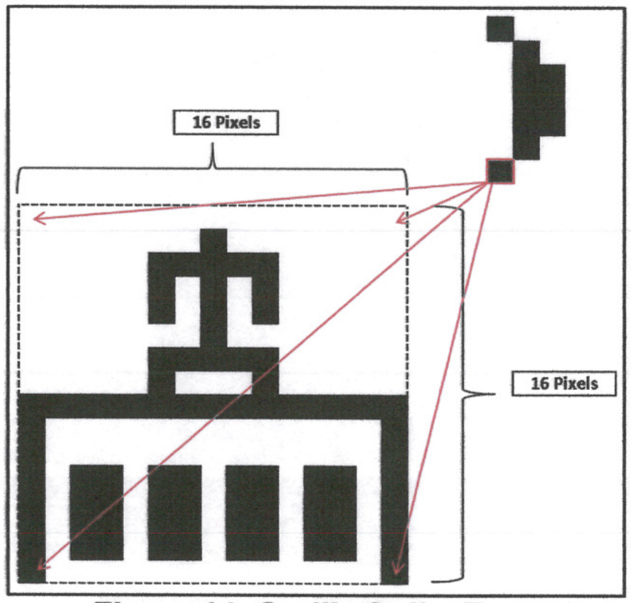


Figure 14: Gorilla Strike Zone

3.2.4 - Reading Potentiometer Values with ADC0

The 8051 microcontroller is equipped with a 12-bit analog to digital converter. This ADC is used to read both potentiometers for angle and speed selection. For GORILLA, ADCO inputs are configured for single-ended inputs with a reference voltage of 2.4V and an input gain of 1. The pertinent registers for controlling ADCO are REF0CN, AMX0CF, ADC0CF, ADC0CN, and EIE2. Detailed information regarding ADC0 registers can be found in the 8051 datasheet. ADC0 settings are described in the GORILLA.C code found in section 6. Both potentiometer readings are handled by a single Interrupt Service Routine (ISR) which is triggered by the ADC complete interrupt found in the EIE2 register. In the ADC_complete ISR, a toggle variable is used to switch between potentiometers, read the current value, initiate the read for the next potentiometer value, and reset the conversion flag. A second ISR triggered by the timer 0 interrupt is used to set AD0BUSY. Because this game is not dependent on specific input voltage values, arbitrary ranges for speed and angle were selected to make the mathematics of calculating vectors easier. The speed potentiometer range is set between 0 and 14, which is less than 16 (the max allowable speed of the banana as determined by strike zone size). The angle value is limited from 0 to 18 which provides for 5 degree (90 degrees / 18) increments per value, which is just slightly more fine than is possible given that the banana's velocity must be limited to 16 pixels per frame. For GORILLA, the maximum realizable angle resolution can be only (90 degrees/16) which is 5.6 degrees. For more detail regarding the use of ADC0 to collect speed and angle data, see the ADC_complete ISR in gorilla.c which can be found in section 6.

3.2.4 - Sound Effects Using DAC0

There are three sounds generated for the GORILLA game. A short 'beep' tone is issued when the banana is launched. When the banana strikes a building or another gorilla, a short 'crash' tone is generated. The last sound is actually a short three-tone melody to signify that the game is over. To do this, the on-board DAC0 of the 8051 is employed. To keep code size compact DAC0 was set to run at 360 Hz constantly. To change the pitch of the tone, different array values are used. For GORILLA, it was decided that the prompt arrays title[], prompt[], and restart[] would be adequate for generating tones, despite their non-sinusoidal characteristics. Title[] is set equal to " GORILLA ", prompt[] to "Press Launch", and restart[] is set to "Game Over!". Using just these arrays at 360 Hz it is possible to generate three distinct digital beeps that have a very retro tone. Each tone is generated for a duration of 200 cycles and simply repeat the array values over and over until the duration is up. To produce the crashing sound, the array pointer is set to the beginning of prompt[], but is then allowed to run past it into code space, reading whatever random data it may encounter. This produces a nice retro "thunk" sound which maintains the theme of the GORILLA game.

A simple ISR triggered by the timer2 interrupt is used to generate the tones necessary for the game. Tone flags and a duration interval are used to initiate specific tones. The tone flags are reset when the duration interval variable is decremented down to zero. For the non-crash tones, a phase variable is used to move the tone pointer back to the beginning of the array to re-read the artificial wave.

4.0 - Testing

4.0.1 Testing Power Supply

To verify that GORILLA meets the 9V power supply requirement, a multi-meter was used to measure the unregulated power from the power supply. A measurement of 9.0 volts was noted. This voltage remained constant during the operation of the GORILLA game, indicating that this power supply is adequate.

4.0.2 Testing the LCD Display

The LCD was tested by initiating the game sequence and drawing the skyline, gorillas, banana, wind vectors, and throwing vectors. All graphics were compared to design specs. The skyline array was compared to the actual position and height of each skyscraper. The location of each gorilla was compared to the position variable and gorilla height variable. The speed value was correctly located in the upper left hand corner along with the angle value. Each value was accurately displayed when the potentiometer were swept through their full range. Wind speed and direction were correctly displayed in the upper right hand corner. The LED backlight came on and remained on during testing, indicating proper function.

4.0.3 Testing the Launch and Reset Buttons

After power-up, the launch button was pressed. The set-up screen was then displayed, allowing for accurate manipulation of speed and angle variables. The launch button was pressed a second time, which initiated a banana launch. The reset button was then pressed, which returned GORILLA to the first prompt page stating that the game was ready to be played.

4.0.4 Testing the Angle and Speed Potentiometers

In the set-up screen, the angle and speed potentiometers were transitioned through their full ranges. The output on the LCD was compared to the variable values and were determined to be correct. The Speed POT successfully displayed the values 0 to 14. The Angle POT successfully displayed the values 0 to 90.

4.0.5 Testing the Wind Difficulty Switches

In the set-up screen, the Wind Difficulty Switches were turned off and the launch button was pressed. The wind value of 0 was accurately displayed for the wind speed. A banana was launched at a 90 degree angle. The banana traveled straight up and came back down on the gorilla, indicating that there was in fact no wind. The reset button was pressed to start a new game, this time the wind was turned on. In the set-up screen the new wind speed was compared to the wind speed variable and found to be accurate. Again, a banana was launched at an angle of 90 degrees. This time, the banana traveled in the indicated direction of the wind speed and at the correct speed.

4.0.6 Testing Boundary Conditions and Game Operation

In the set-up screen, the angle of the banana and speed were set to lob the banana off of the edge of the screen without striking any objects. As expected, the banana was launched, the launch sound was played, and banana followed the prescribed trajectory until it reached the edge, which ended the turn and the banana was placed on the next player. The angle and speed of the banana were then set to intercept a building segment. Upon launch, the launch sound was played, the banana followed the prescribed trajectory, which hit the building segment. When the banana hit the building segment, a crash sound was issued and the buildings height was reduced by one segment. The turn ended and the next player received the banana. The angle and speed of the banana were then set to intercept a gorilla. The launch button was pressed and the gorilla was hit. When the gorilla was hit, the game ended and the Game Over melody was played. The screen was erased and "Game Over" was displayed along with "Press Launch." This procedure demonstrated that GORILLA was operating correctly.

5.0 - Conclusion

GORILLA was successfully tested and demonstrated. Design, construction, and implementation were all completed within the three month deadline prescribed by the project. Through testing, all requirements were verified and found to be complete. A control group, consisting of 2 children and 2 adults, was employed to ensure that GORILLA not only met the given requirements, but was also fun to play. GORILLA received a positive reception by the control group. Though, there were some concerns about graphic resolution and the design of the gorillas. An updated version of GORILLA will likely have a more gorilla-like character. Respondents were unsure whether or not the gorilla had a head, or was even a gorilla. Respondents also suggested that the game keep score of which player was ahead. An updated version may be more successful if high scores are tracked and if levels became increasingly difficult, possibly with new types of objects to throw back and forth. Despite these concerns, it was agreed that GORILLA was a success.

6.0 - Appendices

6.1 - Hardware Schematics

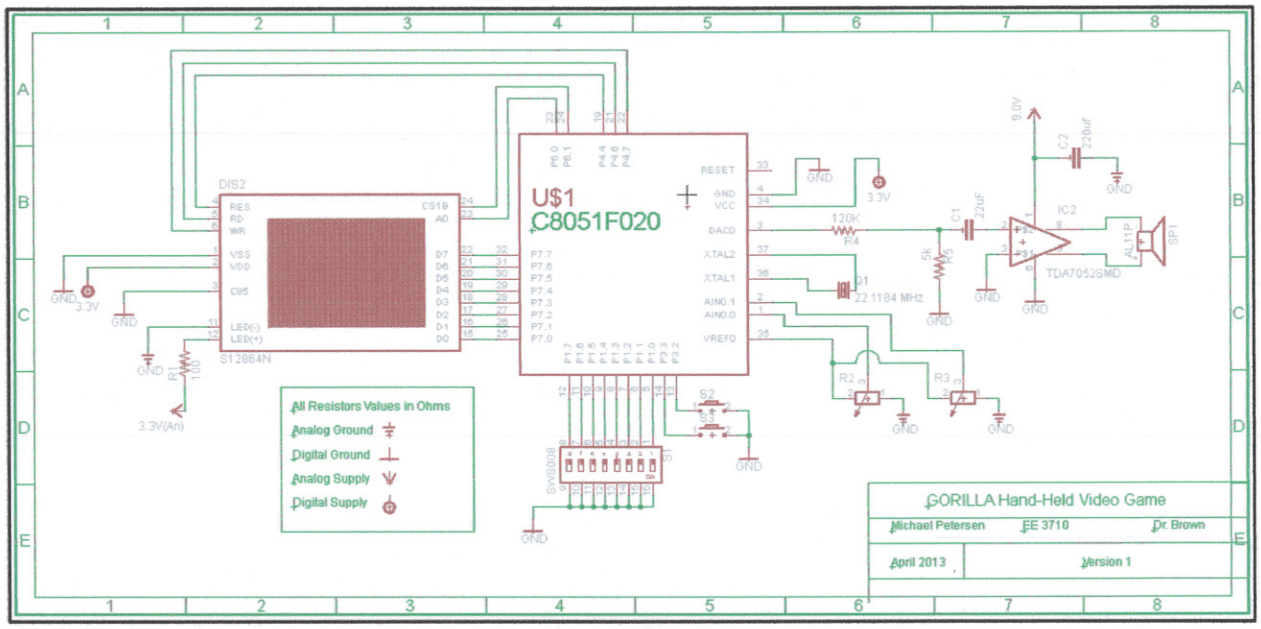


Figure 15 - GORILLA Top-Level Schematic

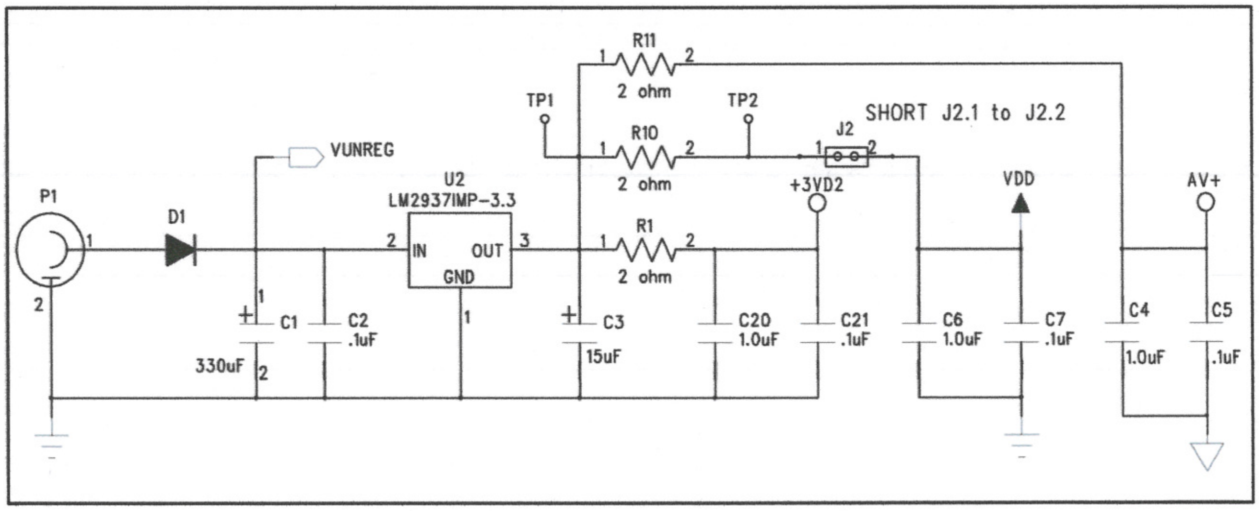


Figure 16 - GORILLA Power Supply Schematic

6.2 - Software Code

6.2.1 - ASM code for LCD control commands

```
$NOMOD51
$include (c8051f020.inc)

;*****************************************************************************
; program header
;*****************************************************************************

;*****************************************************************************
;lcd.asm
; Michael Petersen
; Original Code Date: 3/12/1013
; Modified 4/7/2013 (adapted for GORILLA)
; EE 3710
; Dr. Brown
; This program contains commands for a 64x128 LCD display
;*****************************************************************************

xseg at 0       ; reserve 1000 bytes for lcd map
lcd_map:        ds 1024

;*****************************************************************************
; public declarations
;*****************************************************************************

public lcd_map, blank_screen, lcd_init, refresh_screen, font5x8;
?pr?lcd segment code
rseg ?pr?lcd

;*****************************************************************************
;     coniguration bits
;*****************************************************************************

    mov EMI0CF, #00101001b    ; split mode with bank select
    mov EMI0TC, #0Ch          ; set pulse width to 180 ns
    LCD_CMD equ 1000h         ; high byte for memory map command external memory
    LCD_RESET equ 0F0h        ; deasserts wr, rd, and rst
    LCD_DAT equ 1100h         ; high byte for memory map data external memory

;*****************************************************************************
; blank screen functions
;*****************************************************************************

blank_screen: ; initializes the external memory for the LCD display controller
              ; destroys r1, r0, dptr
    mov r1, #8    ; set counter to 8 for pages
    mov r0, #128  ; set counter to 128 for columns
    mov dptr,#0h  ; start at external RAM location 0
    mov a, #0 ; initialize 1 bit
loop_1:         ; initializes the bits for external RAM
    movx @dptr, a ; store bit from acc to external memory
    inc dptr
    djnz r0,loop_1
    djnz r1, loop_1
    ret

;*****************************************************************************
; local wcom function
;*****************************************************************************

wcom_a:     mov     r0,a        ; save acc in R0 while we check BUSY
wcom:       mov     EMI0CN,#HIGH LCD_CMD ; command/status register
wcom1:      movx    a,@r0       ; r0 has no relevance here
    jb      acc.7,wcom1         ; wait for not BUSY
```

```
        mov     a,r0                    ; get the actual data to write
        movx    @r0,a                   ; write the command, r0 is irrelevant here
        ret

wdat_c:         movc    a,@a+dptr             ; lookup byte to write (handy for fonts)
wdat_a:         mov     r0,a                  ; save it in R0 while we check BUSY
wdat:           mov     EMI0CN,#HIGH LCD_CMD ; command/status register
wdat1:          movx    a,@r0                 ; r0 has no relevance here
        jb      acc.7,wdat1                   ; wait for not BUSY
        mov     EMI0CN,#HIGH LCD_DAT          ; data register
        mov     a,r0                          ; actual data to write
        movx    @r0,a                         ; write the data, r0 is irrelevant here
        ret

;**************************************************************************************
; initialize lcd function
;**************************************************************************************

lcd_init:
        mov p4, #not LCD_RESET
        mov     emi0cf,#28H             ; B5: P4-7, B4: non-muxed, B3-2 split bank
        mov     emi0tc,#0CH             ; pulse width 4 sysclock cycles
        mov     p74out,#0FFH            ; push-pull
        orl p4, #LCD_RESET

        mov     R0,#02FH                ; Boost on, voltage Reg and follower on
        call    wcom
        mov     R0,#0A2H;               ; 1/9bias selected
        call    wcom
        mov     R0,#0A1H                ; reverse segment driver output seg131-seg0
        call    wcom
        mov     R0,#0C0H                ; common output mode com0 to com63
        call    wcom
        mov     R0,#024H                ; Ra/Rb ratio
        call    wcom
        mov     R0,#081H                ; electronic vloume mode set
        call    wcom
        mov     R0,#026H                ; contrast
        call    wcom
        mov     R0,#040H                ; display line address = 0
        call    wcom
        mov     R0,#0A6H                ; normal video
        call    wcom
        mov     R0,#0AFH                ; display on
        call    wcom

;**************************************************************************************
; refresh screen function
;**************************************************************************************

refresh_screen:
  mov           dptr,#0                 ; start of 1k block of memory
        mov     r2,#0B0H ; command to set page number to 0
page_loop:
        mov     a,r2                    ; set page number n, n = 0, 1, 2...7
        call    wcom_a
        mov     a,#04H                  ; set column number to 4. If LCD is not
        call    wcom_a                  ; inverted, you will want to set column
        mov     a,#10H                  ; number to 0.
        call    wcom_a
        mov     r3,#128                 ; copy 128 bytes
byte_loop:
        movx    a,@dptr                 ; get byte from memory
        call    wdat_a                  ; and write it to the LCD
        inc     dptr
        djnz    r3,byte_loop
        inc     r2                      ; advance to next page, but bail if it is 8 (B8)
        cjne    r2,#0B8H,page_loop
        ret
```