Andy Jeong

Chris Brancato

ECE 150B

Spring 2015

13 May 2015

Final Project: Flappy Bird

Table of Contents

Abstract

This project aims to emulate the mobile phone game Flappy Bird using IC logic chips. This game uses 64 individual LEDs oriented in 8-rows by 8-columns LED board, employing a simple set of user controls with a set of randomized obstacles continuously flowing in to the left from the right. Instead of using a row-column controlled LED matrix, this board setup allows for multiple columns and rows to be lit up at one given time. The user can only control to move a bird (represented by a single LED) one unit per clock pulse, while the bird constantly falls one unit at each pulse (gravity). Each randomized column block will contain at least one blank space through which the bird can go through without collision, and at such times, a score is incremented by one. As the score increments and reaches the following scores: 0, 3, 5, and 7, the time interval between each obstacle column is reduced, causing the speed of the game to be faster and thus making it more challenging. The player is expected to control the bird such that the bird is kept flying through the open space in each set of obstacles. The space between each randomized column is to be one space unit apart.

Introduction

As Flappy Bird game on common smartphone mobile game devices is easy to play, this game is mimicked on digital logic environment so that it is harder for the user to play. This game interface employs an 8x8 LED board with individually controllable pins. With this configuration, one could easily turn on the Led at a given spot whenever, multiple times. The game is to be played the same way; a bird is left in the air under the influence of gravity, while the user has the control to increase the position by one unit at a momentary signal (push button). The walls (obstacles) are generated though random number generator—a slight variation from typical linear feedback shift register (LFSR). While LFSR uses XOR gates to vary the outputs, this game utilizes randomized outputs using a fast counter and a shift register, which captures the outputs from the counter. As the game proceeds with the bird passing through the empty space in the walls, the speed is to be varied as well.

Description of the Game

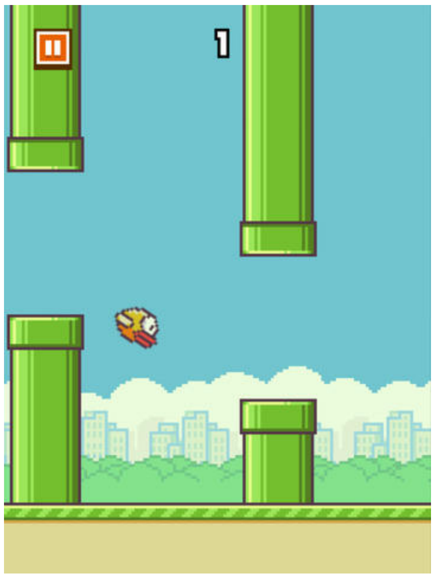


Image 1. "Flappy Bird" game currently in the application market

How to Play

1. Start the game by pressing the 'Start' button. This initiates the game by starting the clock.
2. Press the 'Up' button to keep the bird in motion in air
3. Navigate through the holes in the walls to avoid collision

In the game, a bird has to navigate through holes in the walls by making an up and down movement as the walls approach the bird. The hole in the walls is to be at pseudo-random location, while the width of each wall and the hole(s) is (are) fixed.

Inventory of Chips

- Momentary switch (2 units)
  - Signal to trigger the rising action of the bird
  - Initiation/Forced Termination of the game
- LM555 [Timer] (5 units)
  - Four were used in astable mode configuration while the other in monostable.
  - Astable Mode
    - The downward movement of the bird due to gravity
    - Random number generating circuit
    - Generation of different speeds (with a binary counter)
    - Shift Register separate shifting clock
  - Monostable Mode
    - The upward movement of the bird by the user control
- 4029 [Presettable Up/Down Bin/Dec Counter] (3 chips)
  - For the button to move upward and downward
  - For generating a set of random numbers
  - For score incrementation
- 4027 [JK Flip Flop] (3 gates)
  - Presetting the position of the bird upon pressing of the 'Start' button, with set and reset inverted against each other
  - In order to initiate the game, the toggle mode was used
- 4503 [Tristate Buffer] (3 chips, 16 gates)
  - For the storing of the random bits into the memory, some isolation of the data from the memory was necessary
  - In order to keep the score incremented when the bird is not colliding against the wall, a buffer was used, with enable being the output of OR gates of all shift register outputs at column 2 "bird column"
- 4051 [8-Channel Analog Multiplexer/Demultiplexer] (1 chip)
  - Used to select the speed by the scoring counter outputs
- 4081 [2-input AND] (5 chips, 18 gates)
  - Used in collision logic to compare the display with the bird control data
  - Used to select different speeds by the score
- 4071 [2-input OR] (5chips, 17 gates)
  - Displaying a gap between each wall (in an alternating manner)
  - Selecting different speeds
- 4072 [4-input OR] (3 chips, 5 gates)
  - Selecting different speeds
  - Collision logic
  - Used whenever multiple input OR gates were needed; when 2-input OR was not absolutely necessary
- 4035 [Parallel-In/Parallel-Out shift register] (1 chip)
  - Generating a set of random numbers
- 4015 [Serial-In/Parallel-Out shift register] (8 chips, 16 units)

        - Two 4-bit SIPO shift registers replaced an 8-bit shift register
        - Displaying the randomized columns and shifting them across from right to left
    - 4069 [Inverter] (3 chips, 10 gates)
        - Inverting Decoder outputs (bird position data)
        - Inverting Set-Reset for presetting the initial bird position
        - Inverting the collision logic output to be used for the score counter clock input
    - 2114 [4-bit RAM] (2 chips)
        - Store randomized columns and reading out to display
    - 4543 [7 segment display driver] (1 chip)
        - To help display the score on a 7-segment display unit
    - Resistors (510 Ω, 100 Ω, 470 Ω, 3.3kΩ, 100kΩ, 5.1kΩ, 1.2kΩ)
        - Clock circuit - 510 Ω, 510 Ω, 100 Ω, 470 Ω,
        - On the demultiplexer (for various speeds) fixing with resistors - 510 Ω
        - Bird circuit - (astable) 510 Ω, 3.3k Ω (monostable) 100k Ω
        - 7-segment common anode—3.3k Ω
        - PRNG - 5.1k Ω, 1.2k Ω
        - Shift Register clocks - 5.1k Ω, 5.1k Ω
        -
    a. 64 green LED's
        - One LED was used for each position of the 8x8 board. The color green was used mainly for aesthetical purpose since LED's of different colors showed not too much variance in the resistance when displaying with enough brightness level.

Design Process

- First approach: LED Matrix
    - Difficulties: hard to control and display multiple row s and columns at a time
- Second approach: LED board with individually controlled input to each LED
    - Difficulties: voltage drop/current sink due to too many LED's are paired

In this rendition of the game, the game will consist of an LED panel of 8 pixels in height and 8 pixels in width. The bird will be fixed to the second column and free to move up and down the column as controlled by the user. The walls will be generated by a set of random numbers generated and 1 pixel wide. In order to make the game challenging for the user to proceed, each wall is produced after a set of empty column followed by a wall column. This is produced by alternating (toggling) of the shifting clock. The location of the hole will then automatically determined by the random number generated. The bird will move at a rate of ¼ Hz and the walls will shift leftward from the right at a rate of 0.943 Hz. The rate of gravity was taken into consideration in determining the speed of falling.

Design breakdown

The design may be broken down into 6 major parts:

1. The Clock
2. Wall Generation
3. LED Display
4. Collision
5. Bird Control
6. Score Display

Clock

The clock with capability to vary speed according to the score counter is made by driving a 4029 binary counter with a fast clock of 13.9Hz. This allows four different rates to be produced from a single fast clock, and thus four different speeds of the clock can be implemented for shifting of the walls. Each speed is (13.9/2)Hz, (13.9/4)Hz, (13.9/8)Hz, and (13.9/16)Hz. The outputs of the counter are then AND-ed with the demux outputs. The demux is connected to a JK flip flop and XOR gate, which control whether column shifting should occur. The demux is controlled by the first three pinouts from the score counter, which will be discussed in detail below.

Wall Generation

For generation of each wall, a set of 8 numbers needed to be produced. This was made possible with an astable timer—4029 binary counter—4035 Parallel-In/Parallel-Out shift register circuit. With an astable timer producing a high frequency and the counter dividing that frequency, the shift register is designed to take inputs at random times, since it will be driven by the shifting clock attained from above explanation. This design exploits the time interval for which the data input should be kept high or low for the shift register to take in the data. After generating four random bits, three of the four that produced reasonably pseudo-random bits were selected. These were then decoded and put into memory.

LED Display

The display does not follow a typical common-row, common-column setup. 64 LED's were soldered in a way that would allow for grounding of each row together and control of each LED. Each of the random numbers goes through two buffers of inverted enables in order to produce a pixel gap between the walls. Then the outputs are inputted to Serial-in, Parallel-out shift registers to shift across each row. Each shift register was responsible for each row, and with the serial data input, the data was shifted across the row at the same time, making it appear as if the entire column is shifted together. The column #2 needed to be displayed differently, however. In column 2, the bird location had to be combined with the shift register output. The column will be shifted from right to left direction.

Bird Control

The bird position was controlled in a simple method using a 4029 counter. The up/down pin as connected to a debounced button so that when the momentary switch is not pressed, up/down is LOW

(=> down), and when pressed, up/down is HIGH(=>up). The preset of the counter was set to the middle of the column 2 upon pressing of the 'Start' button. The three outputs produced by the 4029 counter were decoded to produce 8 bits for the column 2.

Collision

Collision detection is managed by using the bird location and display bits. These two were AND-ed and combined with OR gates to check if there is any collision between the wall and the bird location. Then this was put through a tristate buffer controlled by the OR's of the random number column bits in order to trigger the score counter only when there's collision only at column 2. The inverted output of this buffer was then put into the XOR gate after the flip flop in the Clock section as explained above, so that even when the 'Start' button was pressed and JK is HIGH, if there is a collision there is no more shifting on the display.

Score Display

Score display was done using a 7-segment display. With the score counter outputting the correct 4-bit score, the 7-segment driver interprets and allows the 7-segment display to show the score.

Color Coding

- **Red**: Vdd
- **Green**: Clock astable timer 2-6 connection; clock button into JK flip flop clock; Q0 of counter on clock circuit; Demux output 7 into AND gates; connections in AND gates; Bird up button to monostable; connection to bird position decoder inputs; inverted outputs of the decoder on PRNG circuit; input into Data A pin of each shift register from OR gates
- **Blue**: Q2 of counter on clock circuit; Demux output 5,6 into AND gates; connection from clock button to JK flip flop set; inverting Set and Reset in an inverter; Outputs of the decoder for bird position; connection from the 3 bit pseudo random number to the inputs of the decoder; connection from OR gates to AND gates after the shift registers; 2-input, 4-input OR gate pin connections; input into 7 segment display from the driver
- **Orange**: Q1 of counter on clock circuit; Demux outputs 3,4 into AND gates; connection to OR gate; connection from score counter outputs into 7 segment driver
- White: connection from clock button to bird board; connection from decoded bird outputs to OR gates; the score counter clock input
- **Brown**: Memory outputs into OR gates; Q3 of counter on clock circuit; Demux outputs 0,1,2 into AND gates; Final clock signal output
- **Violet**: Column 2 data from each shift register parallel output
- **Black**: Vss

Justification

The color coding for this project did not group everything in one section with one color, as observable in Image 2. However, the color coding was done such that each function was connected with one solid color rather than multiple color, if not different outputs. The grouping by section/circuit would cause more confusion as it would be difficult to distinguish which function one color does, so the wires were chosen this way.

Problems encountered

- Not enough space for all the connecting wires to be placed in a neater appearance
- Some wires had to be tightly placed
- Interference between the wires (i.e. touching)
- Conflicting outputs / unstable outputs
- Comparators could have been used for collision detection
- Voltage drop due to the number of LED's
- Undesired output/random sequence
- Placement of the chips/ orientation of the LED board

Logic Explanation

As the logic diagrams show, each logic was carefully designed, built, and tested by parts, and then they were put together. The general logic is explained as such:

The Clock: an astable counter with a fast clock divides the clock into four different frequencies, and the lowest (Q3) is the speed for the scores 0, 1, and 2, and Q2 the speed for the scores 3 and 4, Q1 the speed for the scores 5 and 6, and Q1 the speed for the score of 7. The score counter outputs Q1-Q3 are signaled into the demux, whose outputs are then AND-ed with the corresponding output of the demultiplexer.

PRNG: an astable fast clock drives a counter dividing the frequency into 4 different pulses. Based on observation, the more random three-pair Q1-Q3 are chosen to be inputted into the shift register, which then delays the clock again. This generates a set of 3 random numbers. They are then decoded into the decoder, and since the decoder outputs only one HIGH state, all outputs are inverted to produce more than one high state.

Bird: an astable timer drives a counter which counts down in binary. Using a monostable, the up/down signal is triggered by a button press, making the bird to move upward. The preset enable can be performed using a JK flip flop with a button press of the start button. The preset is activated by the JK asynchronous set-reset function, and the preset is to be at 4 (or $100_{(2)}$).

The decoded outputs of the PRNG: they are put into tristate buffers, whose outputs are inputted into memory chips. On another set of buffers, ground is inputted, and using inverted enable switches, at a different clock output different buffer is enabled. As this occurs, the memory has its write enable the same as the enable for the buffer with the ground input, so according to Table 1, it writes when the buffer for the PRNG data is enabled.

Then the OR'ed output of the ground-input buffer and the memory outputs go into the data pin of each of 8 SIPO shift registers, which are clocked by a separate clock. The outputs of the shift register at our designed column 2 are OR'ed with bird position inputs so that on the display bird and shift register outputs are both shown. Other columns are simply the parallel outputs of the shift registers

Collision: The displayed output and the bird position inputs are AND-ed and OR-ed to produce the collision logic (HIGH when collide). This should enable/disable the first clock circuit. This is also inverted and put into a buffer, whose enable is the OR-ed output of all shift registers' column 2 outputs. That is, when there are shifting bits at column two, it will turn HIGH. This enables the tristate buffer, and the data input is the inverted collision signal. The output goes to the score counter clock, which becomes rising edge when the bird is able to dodge the wall and navigate through the holes.

Tables and Diagrams

Image 2. Circuit board without 8x8 LED board [Top view] (before final)

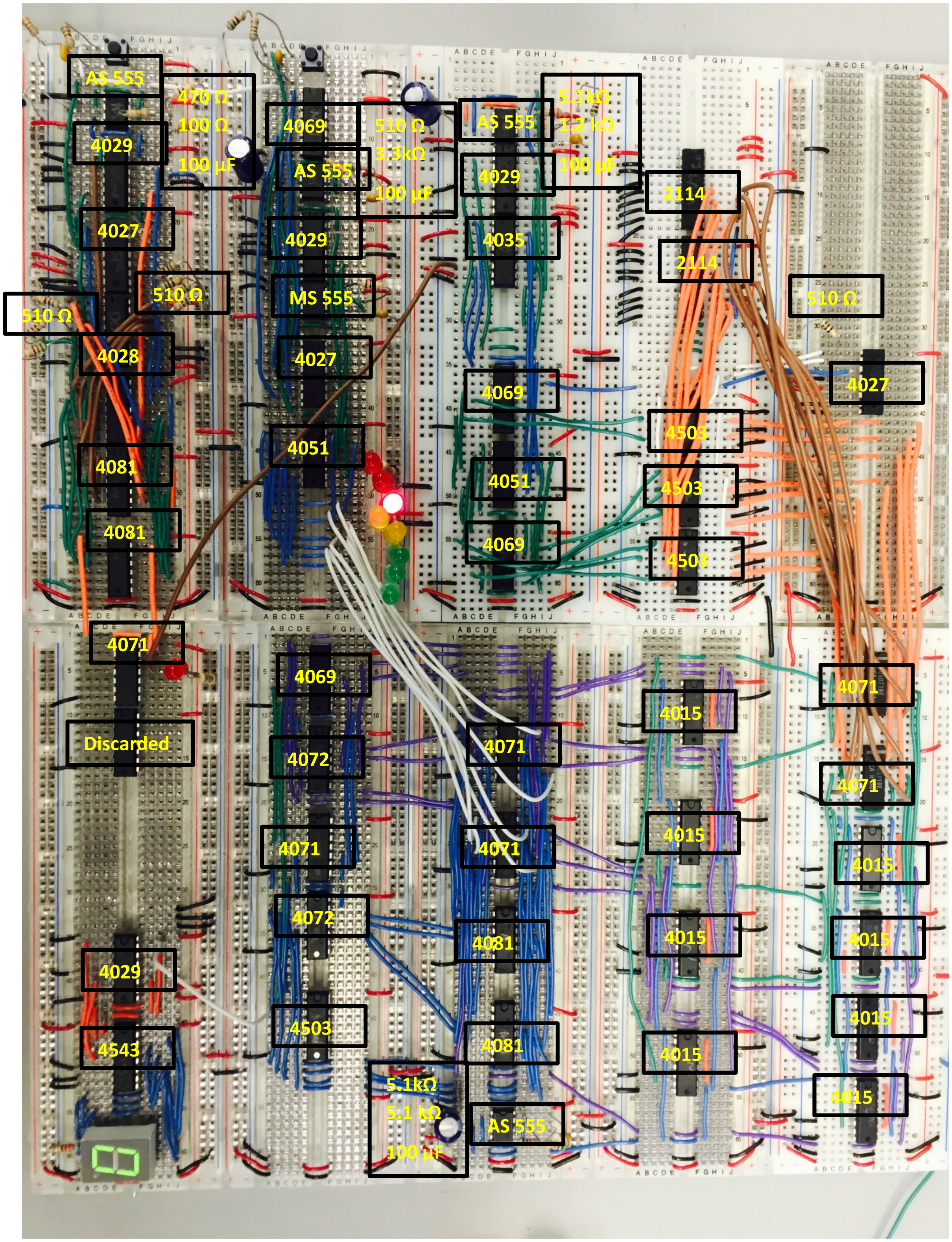
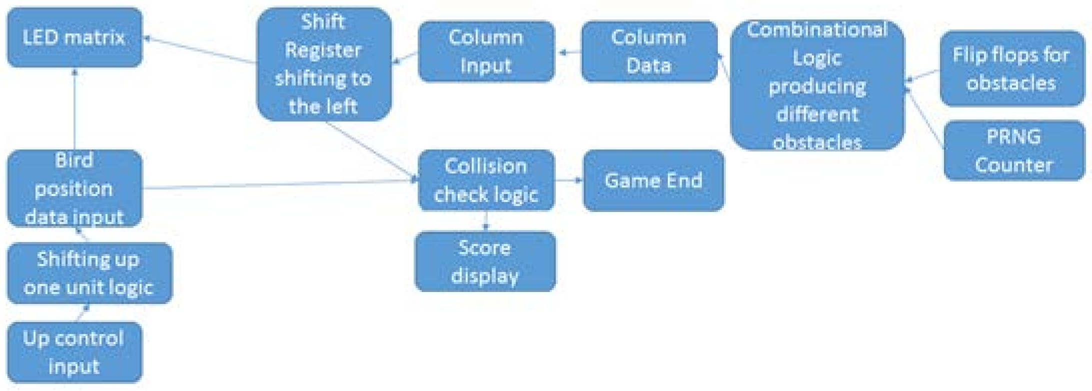


Table 1. State Table (by sections)

Clock[1]	**PRNG Clock**[2]	**JK flip flop for Buffer Enables (in toggle mode)** [3] **Q**	**Buffer for PRNG data into Memory n(EN)** [4]	**Buffer for Ground input (later OR-ed with** [4] **n(EN)** [5]	**Memory n(WE)** [6]	**Shift register Data in/empty column in** [7]
1	1	1	1	0	0 (write)	Empty column in
0	0	0	0	1	1 (read)	Data in

Diagram. 1 Functional Block diagram



Logic Diagram

Diagram 2. Bird

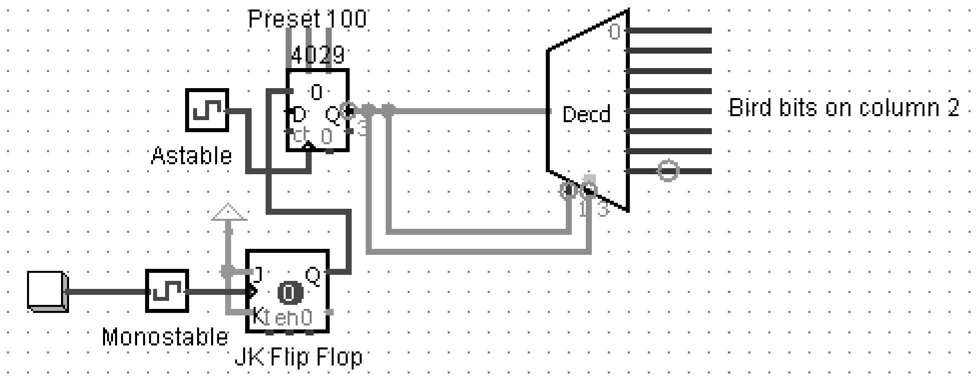


Diagram 3. Clock

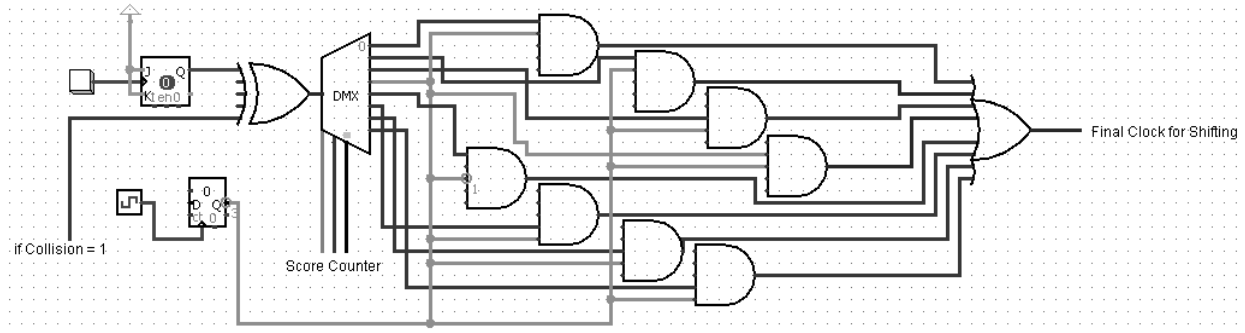


Diagram 4. PRNG

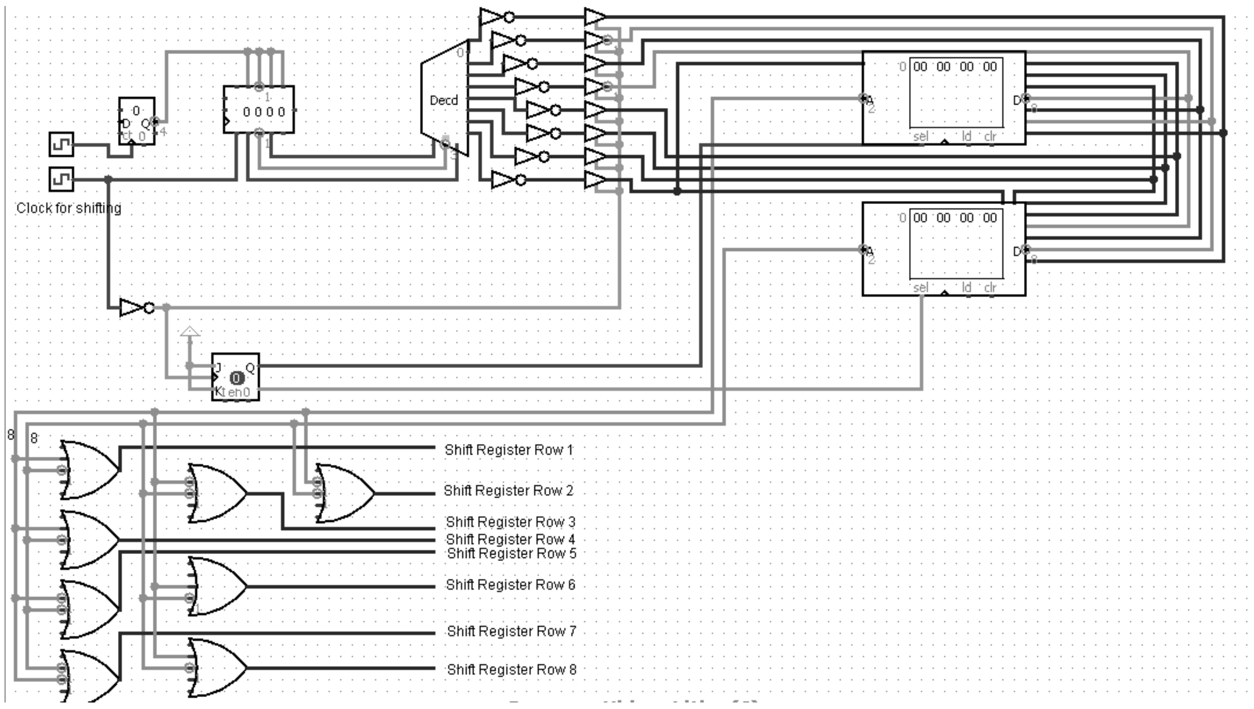
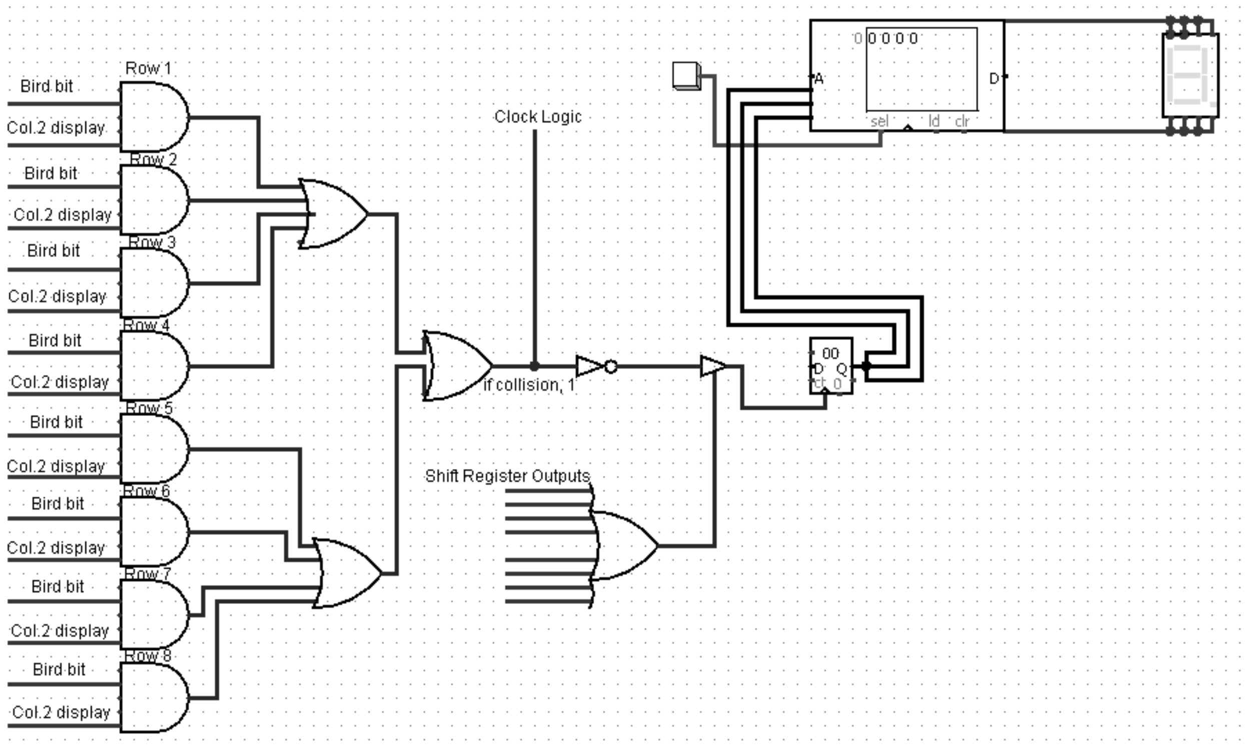


Diagram 5. Collision and Score



State Diagrams

Up/down Bidirectional binary counter was used throughout the design

8 states directional 3-bit binary counter up/down

*input not necessary for counter chip

*preset function often used

*no undesired states

Diagram 6. State Diagram for 3-bit binary bi-directional counter

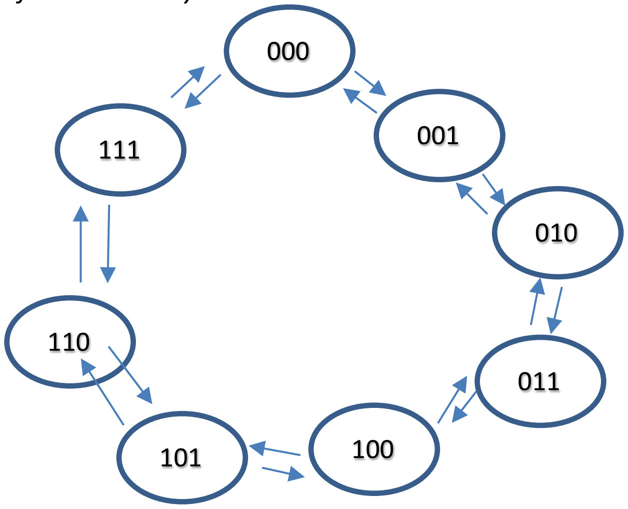


Diagram 7. 3-bit binary counter timing diagram (general)

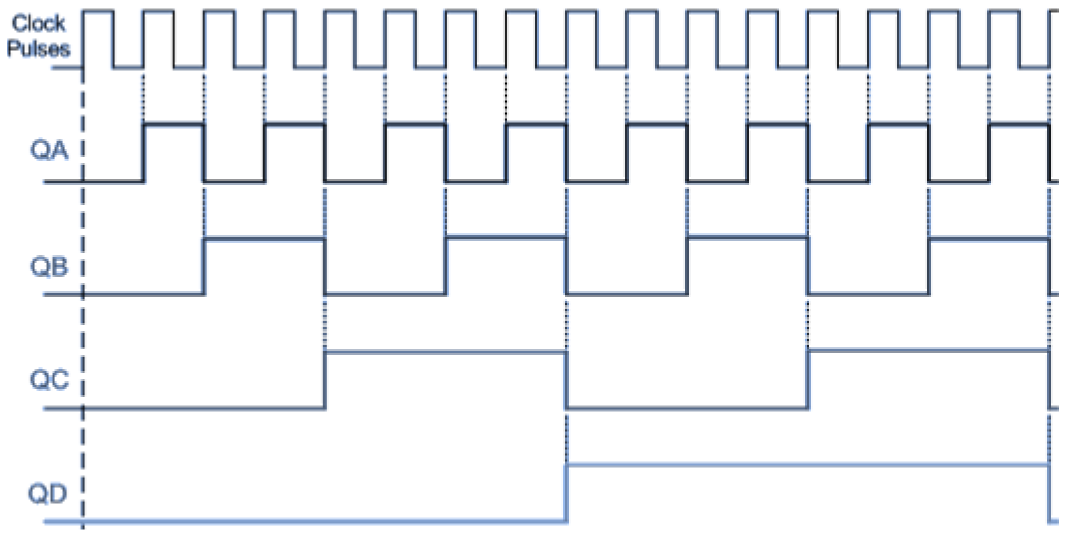
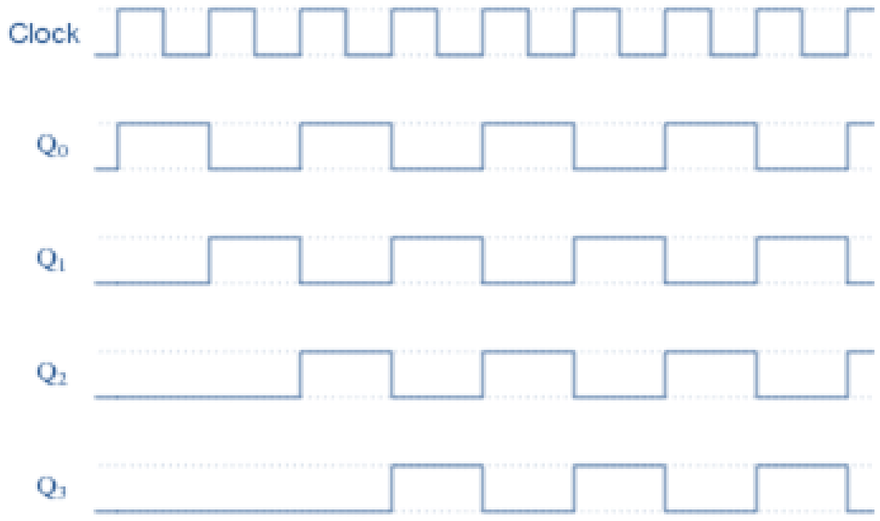
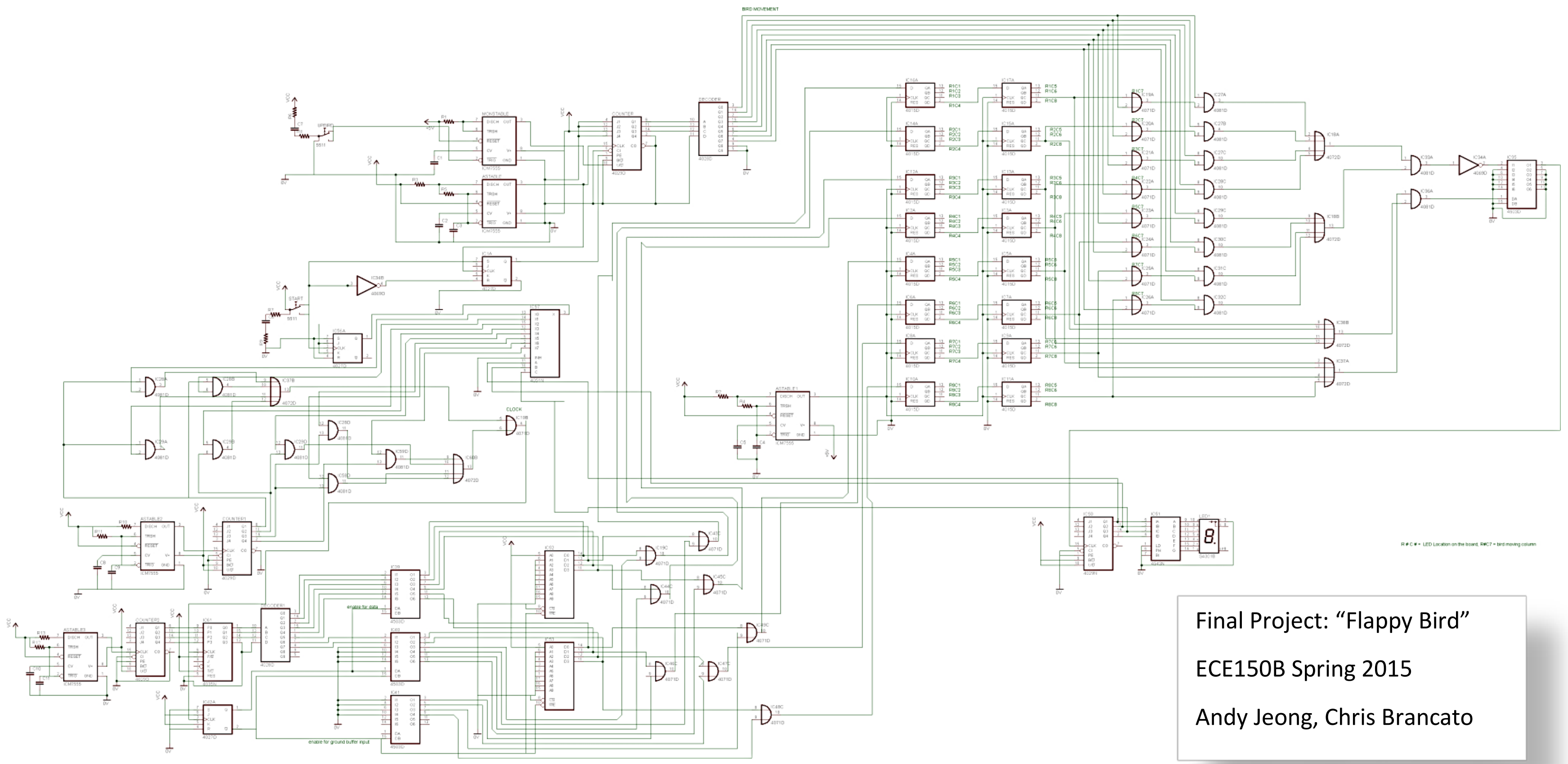


Diagram 8. Serial-in, parallel-out shift register timing diagram (general)



Schematic Diagram



Final Project: "Flappy Bird"

ECE150B Spring 2015

Andy Jeong, Chris Brancato

Appendix

Frequency Calculations $f=\frac{1.44}{(R1+2R2)C}$

1) Clock

   $f=\frac{1.44}{[(100+(2)(470)](100^{-6})} = 13.875\ Hz\ Hz$

   Divided into four phases by a counter: Q0 (fast)= mod-2, Q1 = mod-4, Q2 = mod-8, Q3 (slow) = mod-16

2) Bird

   $f=\frac{1.44}{[(510+(2)(3.3^{3})](100^{-6})} = 2.030\ Hz$

   Divided into four phases by a counter: Q0 = mod-2, Q1 = mod-4, Q2 = mod-8

3) PRNG

   $f=\frac{1.44}{[(5.1^{3})+(2)(1.2^{3})](100^{-6})} = 1.924\ Hz$

   Divided into four phases by a counter: Q0 = mod-2, Q1 = mod-4, Q2 = mod-8

   Also modulo 2 division by pipo shift register

4) Shift registers

   $f=\frac{1.44}{[(5.1^{3}+(2)(5.1^{3})](100^{-6})} = 0.943\ Hz$

Notes

- At the last minute due to some issues rising, some modifications were made to this original design:

  1. Display column to be only displaying columns 1,2,3,5 and 7, skipping 4,6 and 8. The random number shifting will be still shift from right to left, but would not be showing in order to allow for user-friendliness and a little bit more challenging game.
  2. Score counter: preset enable was connected to the collision detection logic signal, which would then cause the score counter to display a certain number when it collides/does not collide.
  3. Tristate buffer that was employed in score clocking was not functional in a stable manner, so it was no longer in use.
  4. Speed variance due to score counter outputs Q1-Q3 did not work because of some glitches on score incrementing and some interference in signals that would not allow simple combinational gates to function properly.
  5. The display alternating sequence (one data column in, then one empty column in) was no longer in use as it was decided that the columns 4,6 and 8 were not to be used for user-purposes.