Super Mario Bros. with Music

The Cooper Union

ECE 150 Digital Logic Design

Professor Risbud

Sharand Phadke & Ha Kyung Yoon

December 20, 2011

ABSTRACT

This project is an implementation of Nintendo's Super Mario Bros. Mario is represented by an LED and is stationary in the x-direction, and enemies, also represented by LEDs, approach him from the positive x-direction. The player receives points, displayed on two 7 segment displays, for jumping at the right time using a jump button and landing on the enemies. The player also has to jump over gaps in the ground, displayed on a second row of LEDs below Mario and the enemies. The game is synchronized with a short excerpt of the Super Mario Bros. song playing on a piezoelectric element or speaker, which repeats once it reaches its end. The song adds to the user experience of the game as well as the complexity of the project.

INVENTORY LIST

Function	Component	Part Number	Quantity
Master Clock	555 Timer	555	1
	DIP Switch		1
	D F/F	4013	1
Random Number Generator	4-bit Comparators	74HC85	2
	8-bit Parallel Shift Register	4034	2
	NOR	4001	1
	XOR	4070	1
	4 input AND	4082	1
Score Display	BCD Counter with 7-SEG driver	4426	2
	7-SEG displays		2
	7-SEG drivers	4511	2
Goomba and Gap	8-bit Parallel Shift Register	4034	1
	D F/F	4013	1
	4-bit PIPO SR	74LS194A	2
	Quad 2 input NAND	4011	1
	Quad 2 input NOR	4001	1
	Green LEDs		8
	Red LEDs		1
	Yellow LEDs		7
	Tri-Color LED		1
Jumping	Momentary Button		1
	555 Timer (monostable)	555	1
	Dual JK F/F	4027	1
	NOR	4001	1
Death Checking	D F/F	4013	1
	NOR	4001	1
	OR	4071	1
Music	4:16 MUX OR	4067	1
	3:8 MUX	4051	2
	8-bit Parallel Shift Register	4034	32
	555 Timer	555	15
	Potentiometers		15
	Speaker		1

Table 1. Inventory List

DESIGN: MUSIC & GAME

The Music and Game are the two separate designs, but they share the same master clock. The master clock is connected to the DIP switch which controls the music and the game speed. The clock also feeds a Dual D type Flip-flop (4013) to cut the frequency to quarters in order to feed the game component. The music component directly uses the master clock. This was implemented because the music was too slow to be aesthetically pleasing at the games slowest playable speed.

Music

The music component of the project consists of an array of 8-bit PIPO shift registers (4034), an array of astable clocks (555 timer), and two 8:1 multiplexers (4512) which combine the previous two elements. The clocks are tuned to audible frequencies corresponding to the notes of the Super Mario Bros. song using the correct combination of resistors, capacitors (from Cap 1 to Cap 15), and potentiometers (from Pot 1 to Pot 15).

15 clocks are used to produce 15 different notes used in the song. Each note is represented as a 4-bit number as shown in Table 1, and the two multiplexers are able to choose from among the 15 notes and ground (representing a rest in the music).

Notes From Low to High	DCBA Address	Notes From Low to High	DCBA Address
Rest	0000	D	1000
E_1	0001	D#	1001
G_1	0010	E_2	1010
G#	0011	F	1011
A_1	0100	F#	1100
A#	0101	G_2	1101
B	0110	A_2	1110
C_1	0111	C_2	1111

Table 1. Music Notes and Given DCBA Address

Table 2 represents the music sheet in terms of translated DCBA Address.

C_1	-	-	G_1	-	-	E_1	-
0111	0000	0000	0010	0000	0000	0001	0000
-	A_1	-	B	-	A#	A_1	-
0000	0100	0000	0110	0000	0101	0100	0000
G_1	E_2	-	G_2	A_2	-	F	G_2
0010	1010	0000	1101	1110	0000	1011	1101
-	E_2	-	C_1	D	B	-	-
0000	1010	0000	0111	1000	0110	0000	0000
-	-	G_2	F#	F	D#	-	E_2
0000	0000	1101	1100	1011	1001	0000	1010

-	G#	A_1	C_1	-	A_1	C_1	D
0000	0011	0100	0111	0000	0100	0111	1000
-	-	G_2	F#	F	D#	-	E_2
0000	0000	1101	1100	1011	1001	0000	1010
-	C_2	-	C_2	C_2	-	-	-
0000	1111	0000	1111	1111	0000	0000	0000

Table 2. Music Coded in DCBA Address

Each digit such as D, C, B, or A is fed to the multiplexers by the shift registers, which loop into themselves to repeat the music. There are 8 shift registers in each of D, C, B, and A. The master clock feeds all the shift registers, and they all have a common reset.

D							
00000000	00000000	01011011	01001000	00111101	00000001	00111101	01011000
C							
10000000	01010110	00011001	00010100	00110000	00110110	00110000	01011000
B							
10010000	00010000	11001010	01010100	00001001	01010010	00001001	01011000
A							
10000010	00000100	00010011	00010000	00101100	01010010	00101100	01011000

Table 3. Binary Code for the Shift Registers

The problems encountered in this component of the design included wiring shift registers and determining the correct control parameters based on the control pins on the 4034 registers. The first of these problems was simply a physical design problem, and was remedied in part by soldering rows of header pins for the parallel input pins on the registers. This saved two rows of wires on each breadboard, and in fact made it possible to reduce the number of boards. The second problem was fixed by testing the shift registers in each of the modes that were needed for full operation of the music.

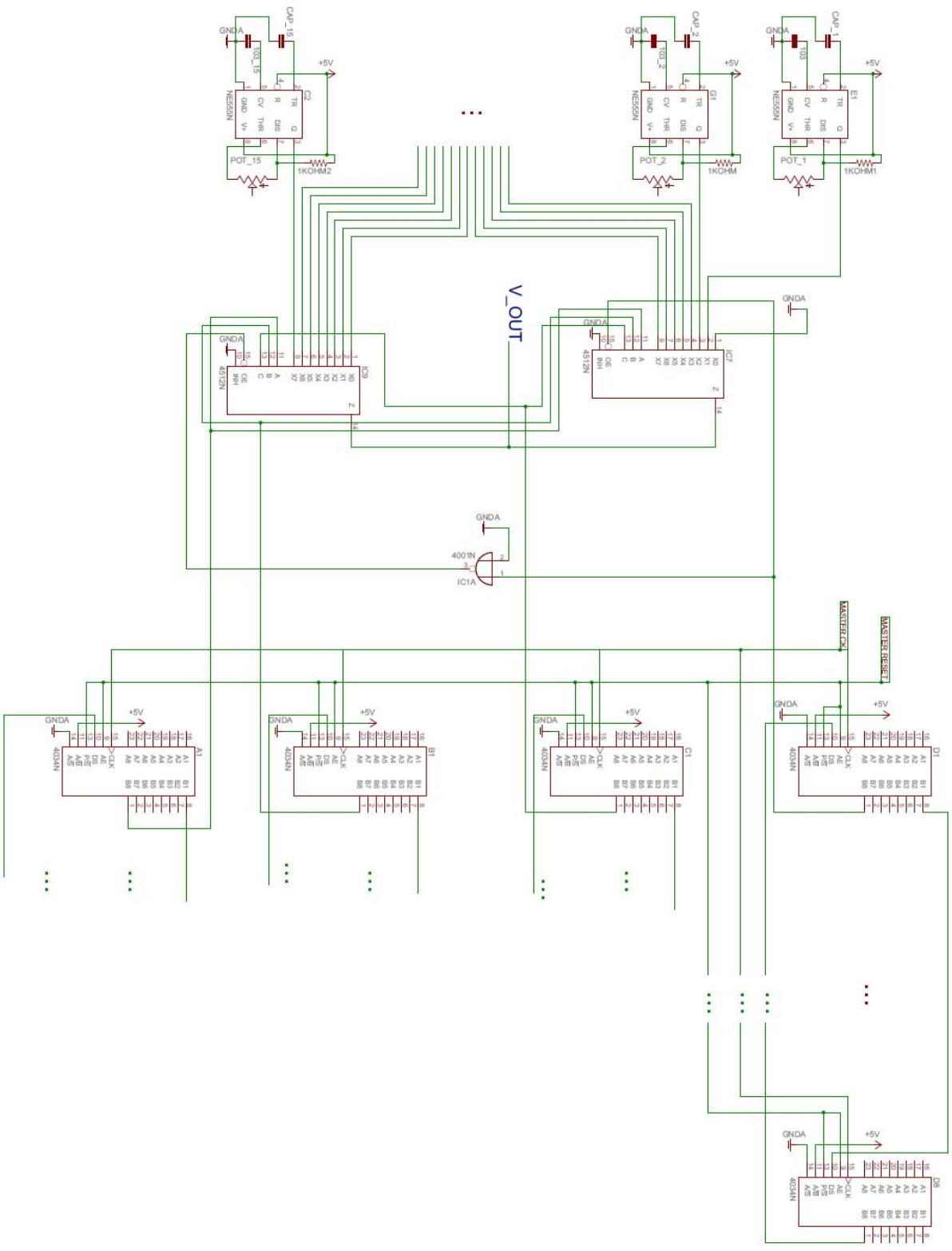


Figure 1. Circuit Diagram for Music

Game

This simplified Mario game has two kinds of enemies: Goombas (yellow) and Gaps (green). Mario is represented as a red LED and placed at a fixed position. Mario and the Goombas are on the ground that has holes. As a Goomba or a Gaps approaches, a user can press the jump button to cause Mario to jump. The scoring system is a set of logical gates that evaluate to 1 when Mario is jumping when a Goomba is one clock cycle away from colliding. If a point is scored, it is saved in a D F/F until the next clock cycle, at which point it is used in the death checking logic. Then, when Mario lands on the Goomba (or collides with the Goomba), the user either gets a point or dies, and the total score is displayed on a 7-seg display. Mario dies when he collides with a Goomba or a Gap without having scored a point on the previous clock cycle. This pushes the game and the music shift registers into a reset mode, where neither of them are functional. The game and music can then be started again by pressing the reset button.

Random Number Generator

In order for the appearance of Goombas and Gaps to be random, two pseudo random number generators were created using linear feedback shift registers. Each of these was composed of an 8-bit parallel shift register (4034), a comparator (7485), NOR (4001) gate, and two XOR (4070) gates. Four randomly chosen parallel outputs were put through a NOR gate and two XOR gates before being fed back into the serial input of the shift registers. This created a long, unpredictable sequence of 0’s and 1’s. Then, the other four parallel outputs from the shift registers were each put to a comparator, comparing the value of those four bits at any given time to four hard coded bits on the comparator chip. Changing the value of the hard coded 4-bit number allowed control over the average frequency of Goombas or Gaps. The shift registers were run on the game clock.

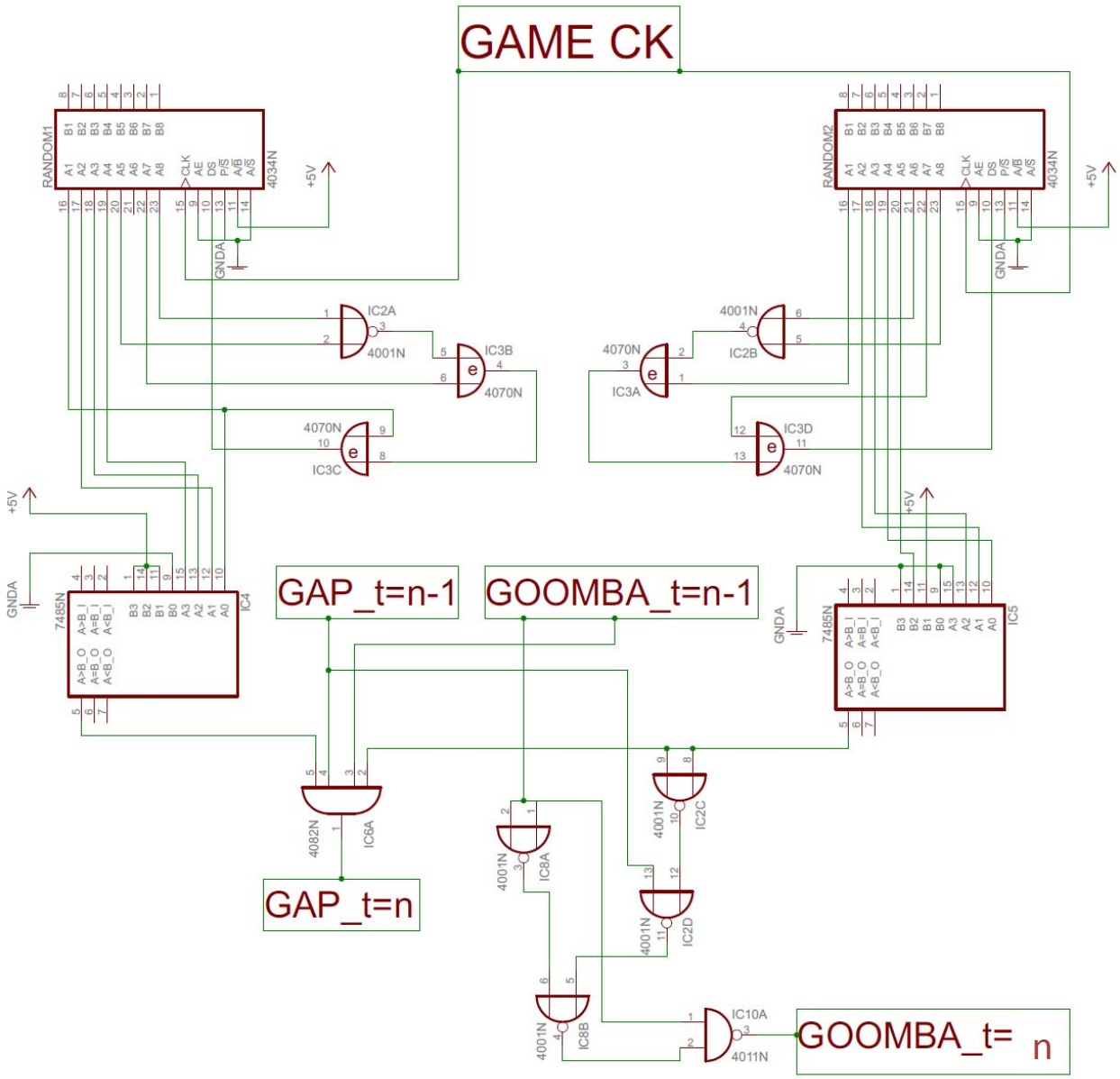


Figure 2. Schematic for the Pseudo Random Number Generators

Goombas & Gaps

Creating the Goombas and Gaps from the output of the pseudo random number generators involved implementing logic to prevent an overflow of enemies. For example, due to jumping gravity, it would be impossible for a user to jump over two enemies in a row, so a feedback control structure was implemented to prevent this. The following state table shows all possible states for the most recently produced Goomba or Gap (t=n-1) as well as what the pseudo random number generators have produced (t=n). The rightmost columns are the outputs of the logic control system, and will be the the t=n-1 Goombas and Gaps in the next clock cycle.

	t_(n-1): G1, Gap1		t_n : random# gen		light on 0	gap on 1
	R_Goomba	R_Gap	R_Goomba	R_Gap	Goomba	Gap
	0	0	0	0	1	0
	0	0	0	1	1	0
	0	0	1	0	1	0
Goomba	0	0	1	1	1	0
	0	1	0	0	X	X
	0	1	0	1	X	X
	0	1	1	0	X	X
impossible	0	1	1	1	X	X
	1	0	0	0	0	0
	1	0	0	1	0	0
	1	0	1	0	1	0
Nothing	1	0	1	1	1	1
	1	1	0	0	1	0
	1	1	0	1	1	0
	1	1	1	0	1	0
Gap	1	1	1	1	1	0

Table 4. Goomba and Gap State Table

As seen in this table, the impossible states (yellow) were ignored, and only the two green states needed modification (only in these states are the rightmost columns different from the center columns). Logic as depicted in the schematic diagram was put in place for each of these cases based on the appropriate Karnaugh Maps.

LED Display

Two sets of two 4-bit bidirectional shift registers were used for the LED display. The two registers in each set were wired in series to control a total of 8 LEDs. The leftmost Goomba LED had to also be able to display Mario, so a tri-color (RGB) LED was used. When a Goomba is on

the location of that LED, the red and green LEDs are lit up, creating yellow. When Mario is on that location, only the red LED is lit. The control pins on the registers were wired to achieve the correct reset conditions for both the Goombas and Gaps. Upon reset, all the outputs form the Goomba registers were set to 1 to remove all enemies from the field, while the outputs from the Gap registers were set to 0 to remove all gaps in the ground from the field.

Jumping and Gravity

A JK F/F chip, a monostable clock, a button, and a NOR chip were used to simulate gravity when using the jump button. The J and K inputs of the first JK F/F are tied together, so that it is always either in the "no change" mode, or the "toggle" mode. The Q_1 and Q_1' outputs of the first JK F/F are tied directly to the two LEDs where Mario can be at any time (they are always opposites). When a user presses the jump button, a monostable clock is triggered, which clocks the first JK F/F, toggling the position of Mario. If the user keeps pressing the jump button, however, Mario does not keep changing position, because the first JK switches to "no change" mode until "gravity "brings Mario back down to the ground. The second JK F/F is used as a D Type F/F, holds the value of the Q_1 for one clock cycle and then outputs it as Q_2. At this point, logic involving both of the Q's triggers a monostable clock which clocks the first JK F/F, bringing Mario back to the ground. Simultaneously, logic evaluates to produce a 1 as the J and k inputs to the first F/F, indicating that the user is free to use the jump button again.

Score & Score Display

The scoring system is simply a set of logic gates that take the position of Mario and the state of the 7^{th} Goomba (from the right) as inputs. The logic outputs a 1 if the user scored a point, triggering the "count up" function of a 7-Segment driver/counter (4426ABE). Two of these counters are linked in series to allow for score keeping from 0 to 99. They are each tied to a 7 SEG display.

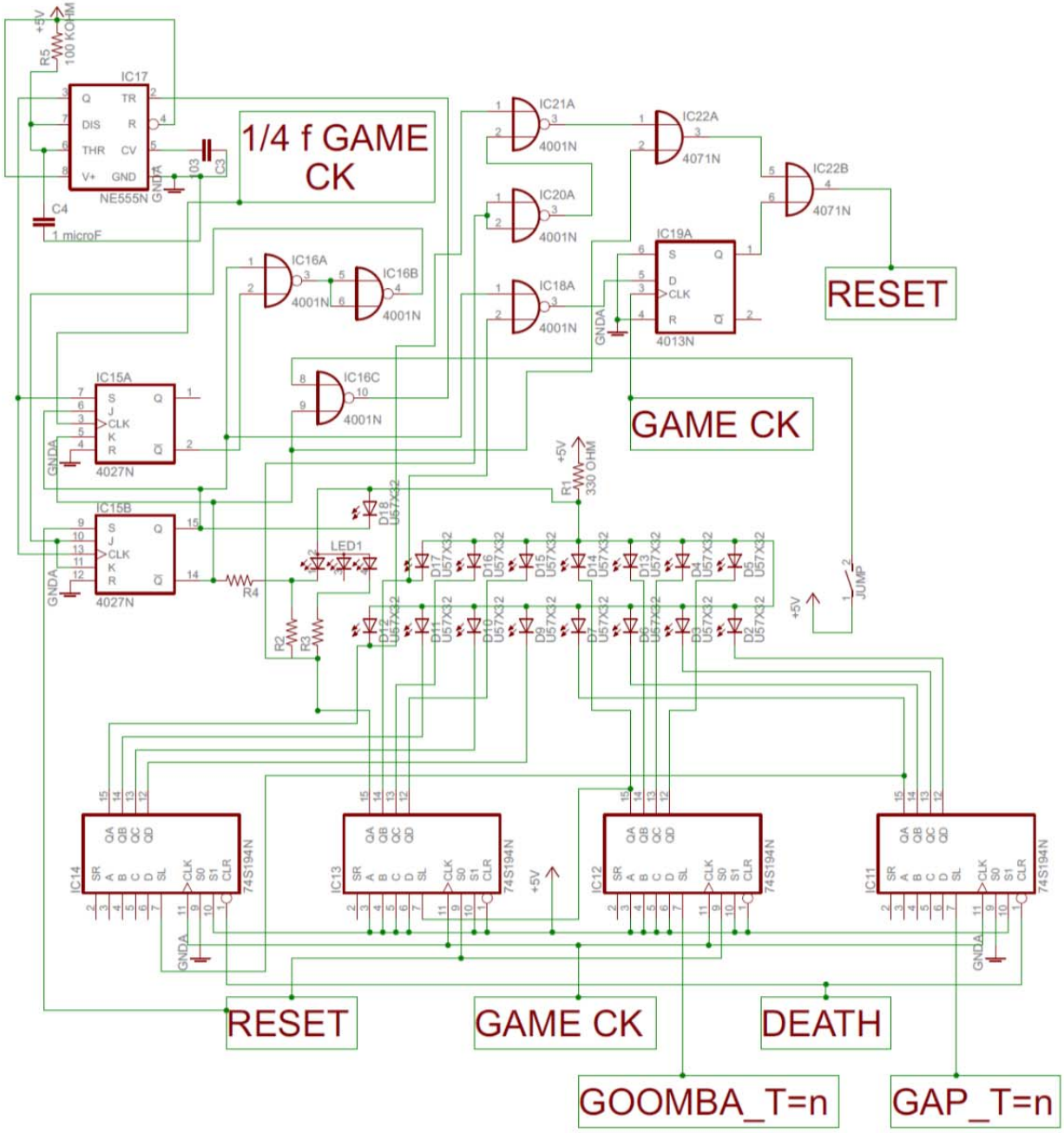
Death Checking

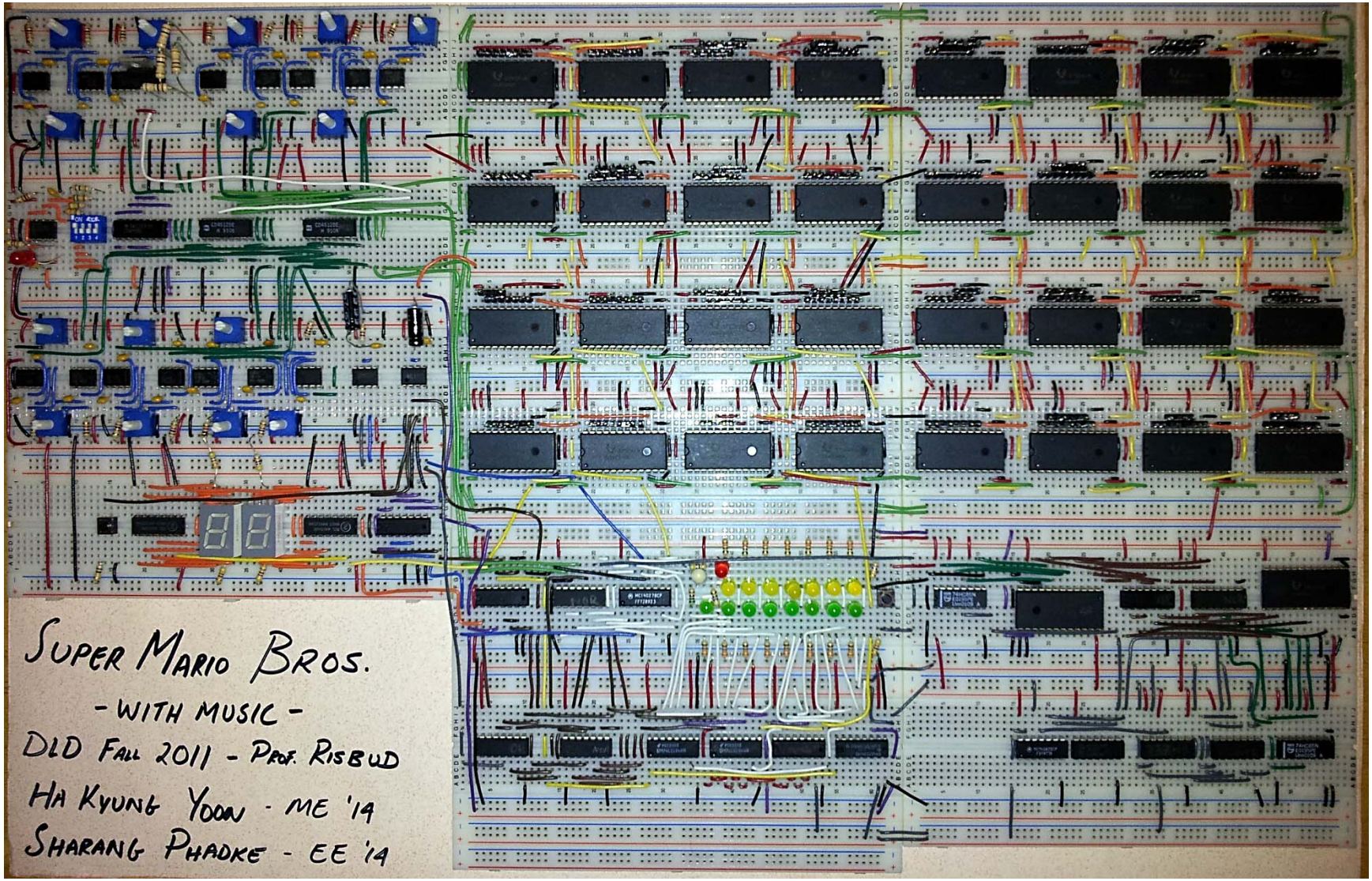
Death checking is also simply a series of logic gates that evaluate to 0 if Mario and either a Goomba or a Gap collide during a particular clock cycle. If Mario collided with a Goomba, there is then further logic to check whether the user had scored a point in the previous clock cycle. If so, then Mario would have landed on the Goomba, indicating that the user is not dead, and instead has scored a point.

Reset

The reset function is contained within a monostable clock, a D F/F which simply toggles between 0 and 1 (Q' is tied to D), and a reset button. Upon death, the monostable clock is triggered, and the D F/F toggles, resetting the music and game. Then, the system remains in reset mode until the user presses the reset button to toggle the D F/F back into normal function mode.

Figure 3. Schematic for Game Display, Death Checking, and Jumping/Gravity




SUPER MARIO BROS.
- WITH MUSIC -
DLD FALL 2011 - PROF. RISBUD
HA KYUNG YOON - ME '14
SHARANG PHADKE - EE '14