Table of Contents

Summary

For the term project in Introduction to Mechatronics at San Jose State University, our team decided to build a variant of "Ski-Ball," the common arcade game. The main differences between our version of Ski-Ball and the well known version are that ours forces the user to aim for a certain target in order to earn points, rather than aiming for any target and getting a score dependant on which target is hit. Our game also differs in that it has a ninety second timer during which the user can hit as many targets as time allows, while the common version of the game offers an unlimited amount of time but a limited number of balls to toss.

To provide the timing, scorekeeping, and random target generation, we used the OOPic microcontroller from Savage Innovations. The code for the game was almost entirely composed of Virtual Circuits, which greatly increase the speed of the computations and allow for many operations to be performed simultaneously. The hardware for the target was constructed mainly out of quarter inch plywood and one-eighth inch particleboard. Single pole-single throw switches were used to tell if a ball had passed through a target, and basic green light emitting diodes were used to indicate which target would result in points if hit. Four seven-segment LEDs were used along with a display driver to display the current score and remaining time.

During the course of this project, we learned a great deal about the programming of the OOPic microcontroller and electrical and mechanical construction. Through the course of this project, we learned how to program, create, and showcase a game that a user was able to easily interact with using the OOPic microcontroller.

Introduction

In order to produce a game that would offer the user a certain amount of

interaction and entertainment, our group first generated several ideas each and presented them to each other. These ideas ranged from puzzle and trivia games to a racecar game, and even a game that would allow a housecat to play. We then presented a list of ten ideas to the course professor, and received feedback on which would offer us many educational opportunities, as well as the ability to become an entertaining game. The professor returned out ideas and suggested our idea of a target game, which we agreed to work on and the modified it to a variant of the common arcade game known as Ski-Ball.

Our version of Ski-Ball integrated several principles that were covered during the semester. Among these were microcontroller programming, basic mechanical design, and circuit design and building. The portions of the project that incorporate microcontroller programming are obvious, the timer, scoreboard, random target generation and score counting are all done on the OOPic. The mechanical design portion was mainly the design of the target and ball return mechanism, and had little to do with what was discussed during the semester. The electrical circuit design was the other section of the project that dealt with the class a great deal. The main circuits that needed to be built were the display drivers for the scoreboard and timer, and the transistor switches to turn the target LEDs on and off in response to signals from the OOPic.

Game Overview

Our game consists of three major systems: hardware, electronics, and software. The hardware consists mainly of the target and ball return mechanism. The electronics system has major subsystems: display drivers, and switches for lighting LEDs and detecting successful scores. The software has several subsystems, among these are: random number generator for the target determination, counters for the clock and timer, and clock dividers. A system block diagram of our game is shown in **Figure 1.**

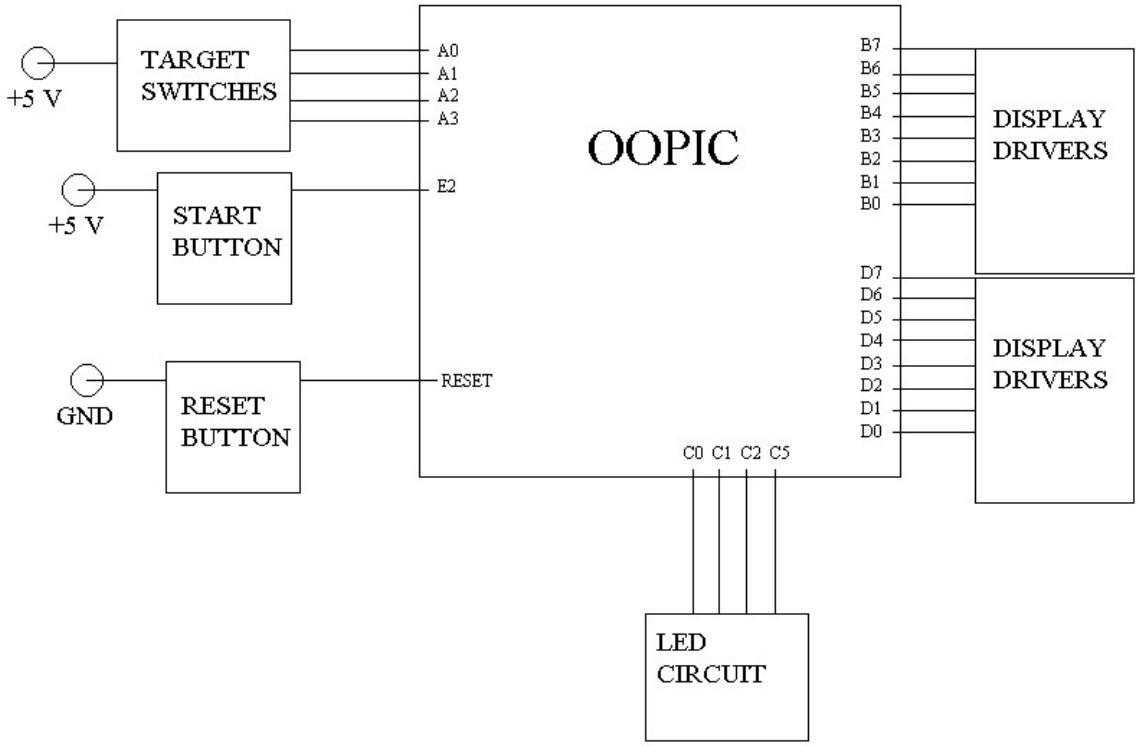

Figure 1. System Block Diagram.

Hardware

Our target board was constructed out of an 18 by 18-inch piece of one-quarter inch plywood. Holes were drilled for targets and slots were cut for pieces of one-eighth inch particleboard to serve as guides. Smaller holes were drilled above each target for LEDs to fit inside, to light up each target. Strips of felt were glued onto surfaces that the balls regularly hit to cut down on the noise and lower the amount of bouncing. Our target is shown below in **Figures 2** and **3**.

Figure 2. Target Board.

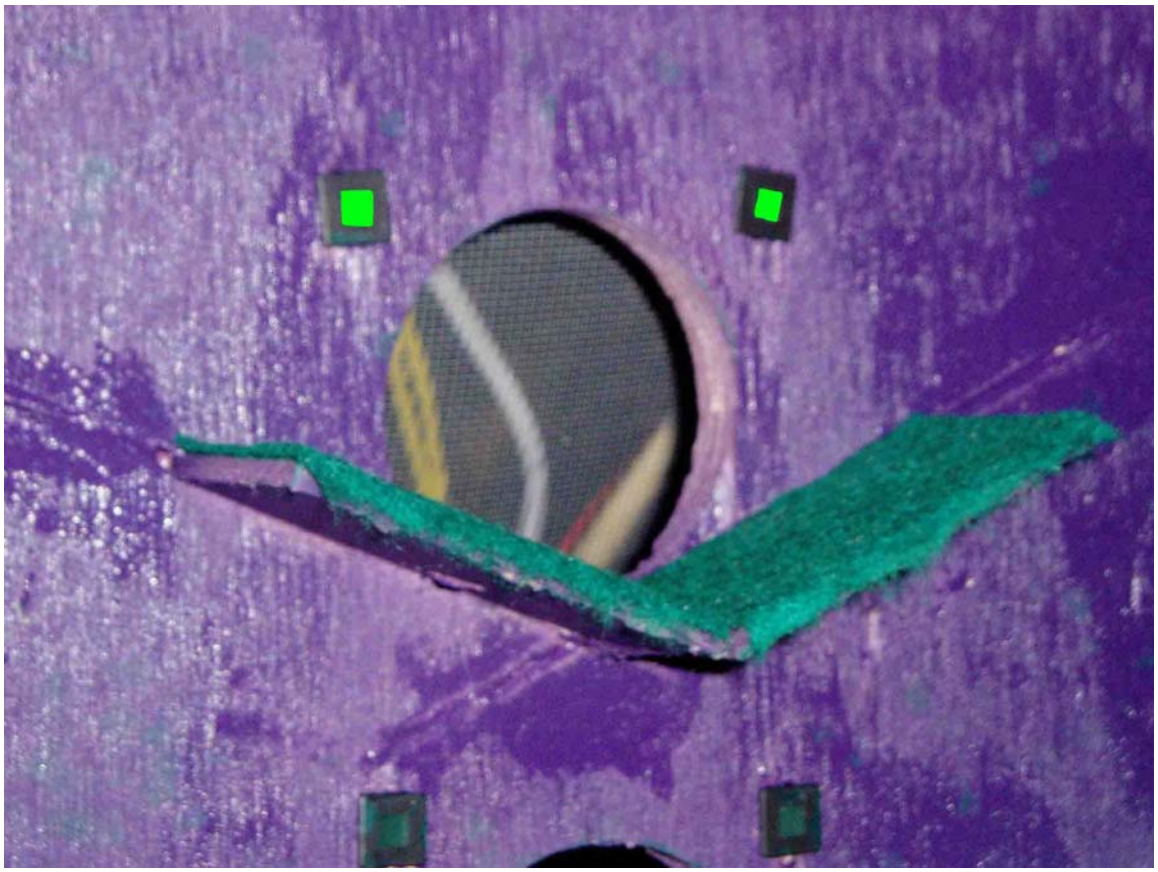

Figure 3. Close-Up of One Target Hole with LEDs.

Electronics

Electronic circuit design played a large part in our project. We needed to design display drivers for each of our four seven segment LEDs that indicate the score and remaining time. We also needed to design a circuit that lit standard LEDs and allowed the OOPic to detect button presses. A photograph of out entire electronic circuit assembly is shown in **Figure 4**.

Display Drivers

For our display drivers we used 7447 decoder chips to convert a four bit binary number into a value that could be displayed as a one-digit number on a seven segment LED. In order to prevent out seven segment LEDs from burning out, we first sent the signal from the 7447 decoder through a 220-Ohm resistor before sending it to the LED Display. Each of the four display drivers are identical, so all schematics and photographs

only show one driver.

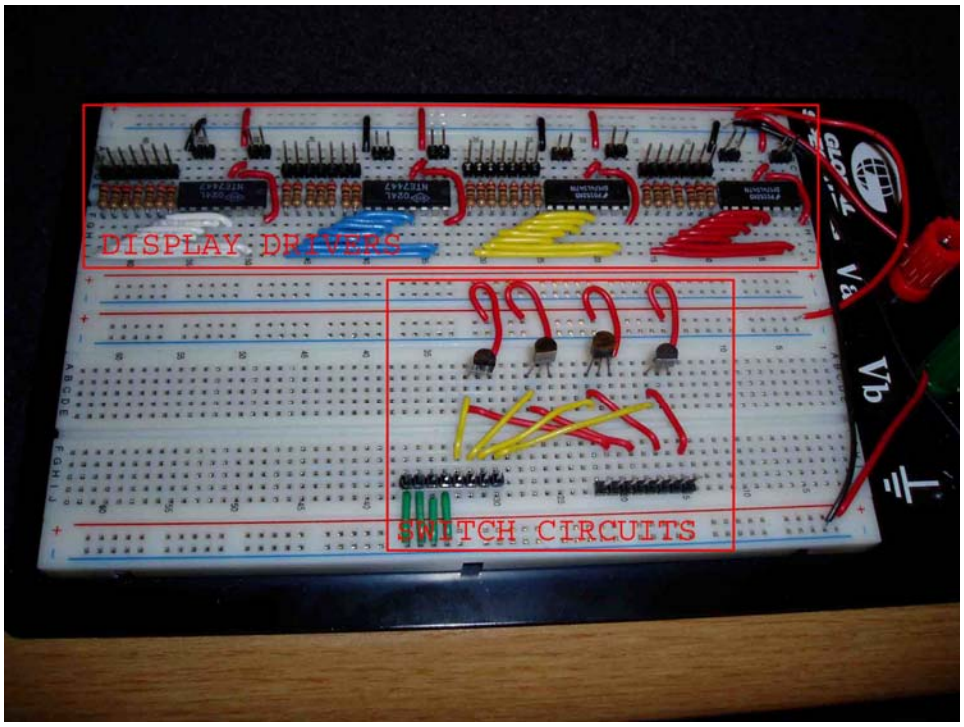

Figure 4. Circuit in Breadboard.

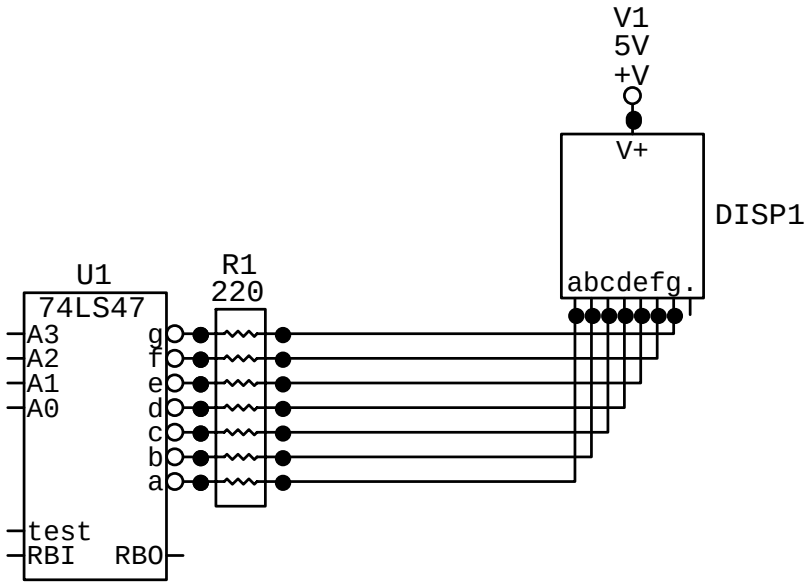

Figure 5. Schematic of Display Diver.

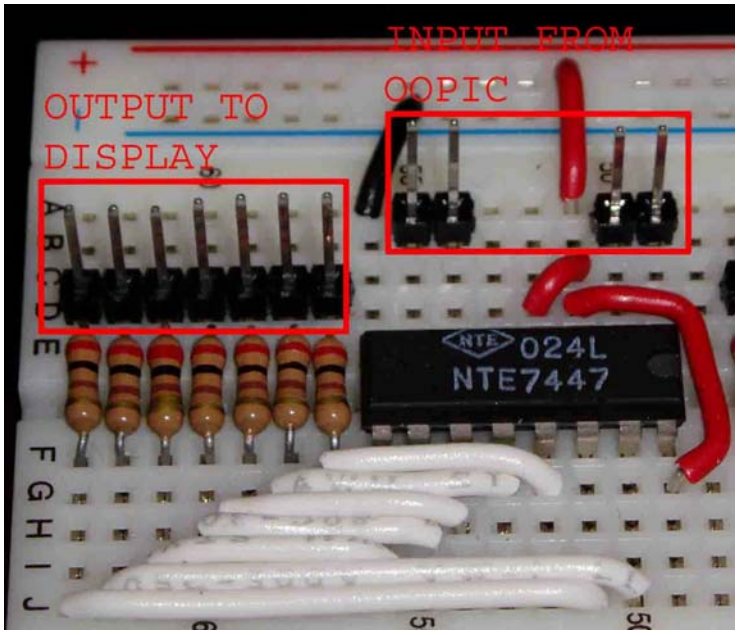

Figure 6. Photograph of Display Driver.

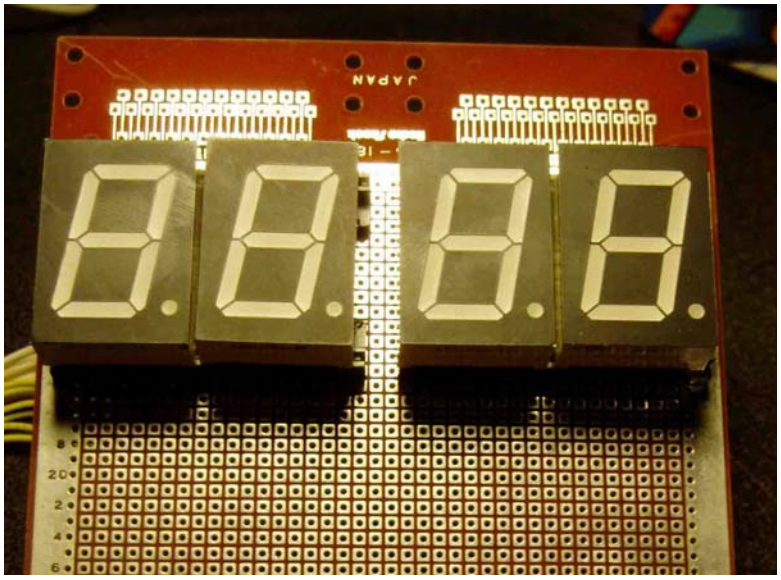

Figure 7. LED Array Used for Scoreboard and Timer.

Switches

Our game uses two different types of switches to accomplish two different tasks, mechanical switches to detect when a ball passes through a target, and electrical

transistor switches to turn on LEDs. The mechanical switches are basic double pole-single throw switches shown in **Figure 8**. When the lever is pressed, the two pins with the leads soldered on are shorted, while the center pin and the pin with no lead create an open circuit. As we only need a switch that shorts when pressed, the third pin was not needed in this case.

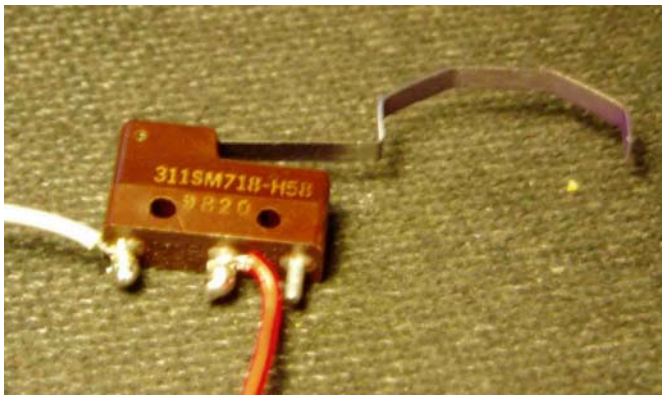

Figure 8. Single Pole-Double Throw Switch.

The transistor switches that were used were common NPN Bipolar Junction Transistors with an h_{fe} of approximately 100. In order to ensure that only the target with the lit LED could potentially send a hit signal to the OOPic, the mechanical switches were wired to the emitter of the transistor. The inputs on the OOPic were then fed into an Or gate to output whenever a switch was pressed. The output of the OR gate was then linked to an oBit object called score. So if a switch is pressed when its LEDs are on, score is momentarily taken high. This enables us to use the OOPic to keep the appropriate score. This circuit is shown in **Figure 9**.

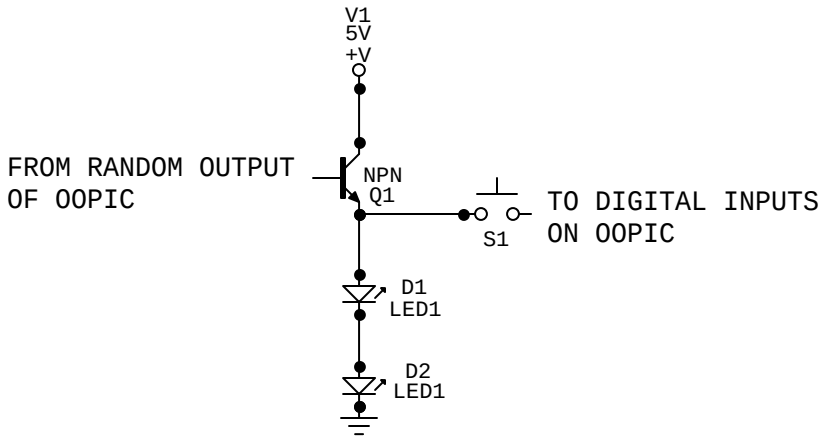

Figure 9. Schematic of Switch Circuit.

Software

The last and most time-consuming portion of the project was by far the programming. This took the longest largely due to the combination of the material being new and various bugs within the OOPic programming language. The main subsystems of the program are the counters to count the score upward and the time downwards, clock dividers to enable one digit of each display to serve as a tens digit, and a random number generator to randomly choose a target. For reference, the entire code along with a flow chart for our game is shown in **Appendix A.**

Counters

One of the most basic functions of our game is to keep track of several values and display them on a seven-segment LED. One set of LEDs needs to count the time down, and the other needs to count the score up. The OOPic object best suited to our needs here is the oCounter object. Its function is to take in a signal from a source that it is linked to, and increment or decrement the value of another object that it is also linked to. The oCounter object can also function as a counter for a quadrature motor encoder, but the oQencode object is better suited to this use. To ensure proper counting for a one-digit display, the Input property can be used to provide a limiting value for each digit. If the

object is counting up, when the value equals the Input value, it is reset to zero. Likewise, if the object is counting down, when the value gets less than zero, it is set to the Input value and then decremented. In both cases, an Input value of ten results in the proper wrap-around values for each digit. Thus, we used an oNibble object with the value set to ten and linked it to the Input value of the oCounter object for all four digits. Each digit of our display uses its own oCounter object to either increment or decrement its value, the two digits for the score counting up and the two for the time counting down. The ClockIn property of each counter is the rate at which it counts. For the ones digit of the timer, we simply link its counter to the OOPic.Hz1 frequency that is provided by the OOPic. Likewise, for the ones digit of the scoreboard, we can link to the output of the score object to the ClockIn of its counter. The tens digit is where problems arise. In order to ensure that the tens digit incremented when the ones digit wraps around, we need to employ clock dividers.

Clock Dividers

Clock dividers allow us to generate nearly any frequency we want by manipulating onboard frequencies. The OOPic object that performs a clock divider function is the oDivider object. Like the oCounter, the oDivider has a ClockIn property that links to a frequency. The Rate property is that value that the input frequency is divided by. By dividing by ten for both dividers, we get a frequency that is one tenth of the input. By inputting the OOPic.Hz1 frequency to the oDivider for the clock and the score variable to the oDivider for the score, we get counters that count properly on both the ones and tens digits.

Random Number Generation

In order for the OOPic to randomly choose a target for the user to aim for, we need to use the oRandomizer object and four logic gates. The oRandomizer object has two function called Result and Result2, which generate random Boolean values. By feeding these values into four logic gates, we can get one of four oDio1 objects to go high depending on one of the four combinations of the two Boolean values. A representation of this Virtual Circuit is shown in **Figure 10.**

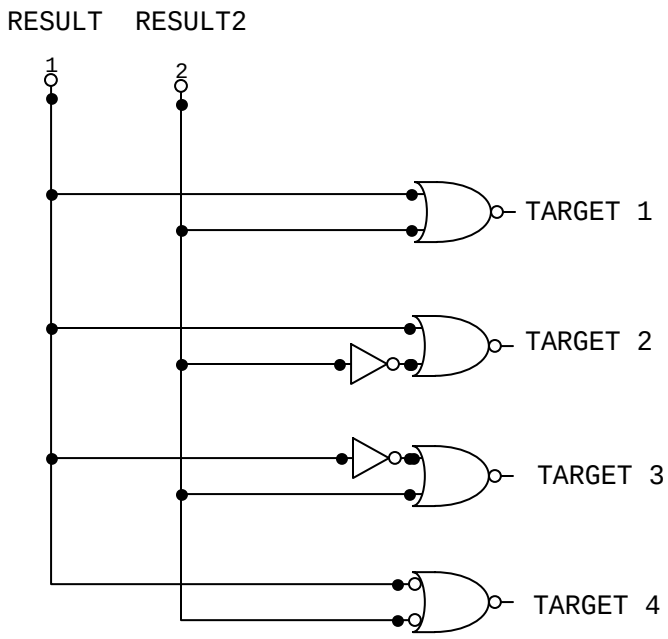

Figure 10. Virtual Circuit used in Randomization.

Programming Bugs

During the course of the project, we encountered a few bugs associated with the OOPic that need to be discussed briefly. The first bug that we ran into came about when we used the oEvent object to execute a portion of code and then return to the main program. It seemed that we were not able to call the same function twice in the same while loop. But this problem was solved when the Operate property of the oEvent was set back to cvFalse after the event was done running. This allowed us to call the same event several times in the same loop.

The other problem was with the OOPic.Reset function. The problem with it is that it simply does not work. When used, it either freezes the OOPic or does nothing at all. This problem was solved by soldering an extra pin onto the reset line of the OOPic. We could then place a jumper between that pin and any other pin, and when the other pin was taken low, the OOPic would reset. The location of the extra pin is shown in **Figure 11.**

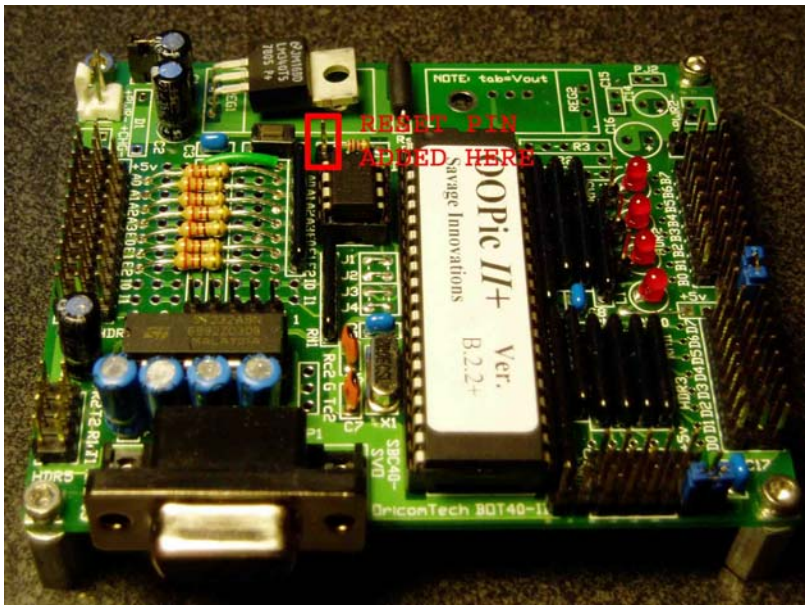

Figure 11. Location of Reset Pin.

Outcome

After presenting our game to the class, there were a few things that we might like to change if we had the time or were required to repeat the project. First, the game was much harder than we had anticipated. This was largely due to the excessive bouncing around of the balls. It seemed that every time someone was about to score in the appropriate target, the ball would bounce out into the wrong one. Another problem that we had was that there was no way to contain a ball that was mistakenly thrown off target. If someone tossed the ball and it completely missed the target, it simply went flying across the room. If we were to build some sort of wall to keep the balls from flying off, the game would likely be much more enjoyable. Looking back at the project, we also feel that it would make the game more fun if we added some sound effect to tell the user when the game starts and stops, and when points are scored. Although there were a few problems that we encountered, none of them had anything to do with our program or any of our circuits. As the bulk of the class dealt with these two subjects, I feel that this project was very beneficial to my education.

Reference

Alciatore, D.G. And Histand, M.B. Introduction to Mechatronics and Measurement Systems (2nd ed.) Tata McGraw - Hill.

Furman, BJ. ME 106 Lab Manual. San Jose State University : Mechanical and Aerospace Engineering Department, Fall 2003.

Furman, BJ. ME 106 - Term Project Information . San Jose State University : Mechanical and Aerospace Engineering Department, Fall 2003.

Appendix A

```
//Code for Ski-Ball Game
//San Jose State University
//ME 106
//Fall 2003


//Objects for seven segment LEDs
oDio4 digit1 = new oDio4;        //Object for First Digit
oDio4 digit2 = new oDio4;        //Object for Second Digit
oDio4 digit3 = new oDio4;        //Object for Third Digit
oDio4 digit4 = new oDio4;        //Object for Fourth Digit

//Objects to light up target LEDs
oDio1 target1 = new oDio1;
oDio1 target2 = new oDio1;
oDio1 target3 = new oDio1;
oDio1 target4 = new oDio1;

//Objects to detect when a target is hit
oDio1 Hit1 = new oDio1;
oDio1 Hit2 = new oDio1;
oDio1 Hit3 = new oDio1;
oDio1 Hit4 = new oDio1;

//Object to be taken high for score incrementation
oBit score = new oBit;

//Limiting Values for all oCounter Objects
oNibble Limit = new oNibble;

//Counter Objects - One for each digit
oCounter clock1 = new oCounter;  //Object to countdown the tens digit
                                        //in the countdown clock
oCounter clock2 = new oCounter;  //Object to countdown the ones digit
                                        //in the countdown clock
oCounter count1 = new oCounter;  //Object to count the tens digit of
                                        //the score
oCounter count2 = new oCounter;  //Object to count the ones digit of
                                        //the score

//Gates for random target generation
oGate LED1 = new oGate(2);
oGate LED2 = new oGate(2);
oGate LED3 = new oGate(2);
oGate LED4 = new oGate(2);

//Gate to tell when target is hit
oGate Target_Hit = new oGate(4);


//Clock Dividers
oDivider divide_clock = new oDivider;   //Divider for clock
oDivider divide_count = new oDivider;   //Divider for score


//Event Objects
oEvent Start = new oEvent;              //Starts game
oEvent Stop = new oEvent;               //Stops game

//Wire objects
oDio1 Start_Game = new oDio1;           //Start button
oWire Pass_Start = new oWire;           //Passes value of button to
                                        //Start.Operate

//Randomizer object
```

```
oRandomizerC rand = new oRandomizerC;



sub void main(void)
{

//*****SETUP DISPLAY*****//
digit4.IOGroup = 1;              //Make digit1 a 4 bit digital IO on pins D0 to
D3
digit4.Nibble = cvLow;
digit4.Direction = cvOutput;

digit3.IOGroup = 1;              //Make digit2 a 4 bit digital IO on pins D4 to
D7
digit3.Nibble = cvHigh;
digit3.Direction = cvOutput;

digit2.IOGroup = 3;              //Make digit3 a 4 bit digital IO on pins B0 to
B3
digit2.Nibble = cvLow;
digit2.Direction = cvOutput;

digit1.IOGroup = 3;              //Make digit4 a 4 bit digital IO on pins B4 to
B7
digit1.Nibble = cvHigh;
digit1.Direction = cvOutput;
//*****DISPLAY SETUP*****//

//*****SET INITIAL VALUES ON CLOCK AND SCORE
digit1.Value = 9;
digit2.Value = 0;
digit3.Value = 15;  //MUST BE 15 TO COUNT CORRECTLY
digit4.Value = 0;
//*****INITIAL VALUES SET*****//


//*****SETUP TARGET SENSORS*****//
Hit1.IOLine = 1;                 //Make target sensors digital
Hit1.Direction = cvInput;  //inputs on lines 1 through 4

Hit2.IOLine = 2;
Hit2.Direction = cvInput;

Hit3.IOLine = 3;
Hit3.Direction = cvInput;

Hit4.IOLine = 4;
Hit4.Direction = cvInput;
//*****TARGET SENSORS SETUP*****//



//*****SETUP TARGET LIGHTS*****//
target1.IOLine = 16;             //Make target lights digital
target1.Direction = cvOutput;    //outputs on lines 16 through
                                 //18 and 21
target2.IOLine = 17;
target2.Direction = cvOutput;

target3.IOLine = 18;
target3.Direction = cvOutput;
```

```
target4.IOLine = 21;
target4.Direction = cvOutput;
//*****TARGET LIGHTS SETUP*****//


//*****SETUP START BUTTON*****//
Start_Game.IOLine = 7;                  //Make start button a digital
Start_Game.Direction = cvInput;  //input on line 7
//*****START BUTTON SETUP*****//



//*****SETUP CLOCK DIVIDERS*****//
divide_clock.ClockIn.Link(OOPic.Hz1);
divide_clock.Rate = 10;

divide_count.ClockIn.Link(score);
divide_count.Rate = 10;
divide_count.InvertC = cvTrue;   //MUST BE TRUE TO COUNT CORRECTLY
//*****CLOCK DIVIDERS SETUP*****//



//*****SETUP COUNTERS*****//

//Clock Counters
clock2.ClockIn1.Link(OOPic.Hz1);  //Link input of clock2 to 1 Hz frequency
clock2.Output.Link(digit2.Value);//Link output of clock2 to value of ones digit
in timer
clock2.Direction = 1;                   //clock2 counts down
clock2.Mode = 0;
clock2.Tick = 0;
clock2.Input.Link(Limit.Value);  //Reset to nine

clock1.ClockIn1.Link(divide_clock.Result);     //Link input of clock1 to .1Hz
frequency
clock1.Output.Link(digit1.Value);//Link output of clock1 to tens digit of clock
clock1.Direction = 1;                   //clock1 counts down
clock1.Mode = 0;
clock1.Tick = 0;
clock1.Input.Link(Limit.Value);         //reset to nine

//Score Counters
count1.ClockIn1.Link(score);            //Link input of count1 to button
count1.Output.Link(digit4.Value);//Link output of count1 to ones digit of
scoreboard
count1.Direction = 0;                   //count1 counts up
count1.Mode = 0;
count1.Tick = 0;
count1.Input.Link(Limit.Value);

count2.ClockIn1.Link(divide_count.Result);
count2.Output.Link(digit3.Value);
count2.Direction = 0;
count2.Mode = 0;
count2.Tick = 0;
count2.Input.Link(Limit.Value);

//Limiting Value for all counters
Limit.Value = 10;                       //Value to reset to after digits get to
zero
//*****COUNTERS SETUP*****//
```

```
//*****SETUP WIRE*****//
Pass_Start.Input.Link(Start_Game);            //Input is start button
Pass_Start.Output.Link(Start.Operate);  //Output is event to start game
Pass_Start.Operate = cvTrue;
//*****WIRE SETUP*****//


//*****SETUP RANDOMIZER*****//
rand.ClockIn.Link(score);        //Generate new random number when
rand.Operate = cvTrue;                 //point is scored
//*****RANDOMIZER SETUP*****//


//*****SETUP GATE FOR TARGET*****//
Target_Hit.Input1.Link(Hit1);           //When an activated switch is
Target_Hit.Input2.Link(Hit2);           //pressed, make score high
Target_Hit.Input3.Link(Hit3);
Target_Hit.Input4.Link(Hit4);
Target_Hit.Output.Link(score);
Target_Hit.Operate = cvTrue;
//*****TARGET GATE SETUP*****//


//*****SETUP RANDOMIZER GATES*****//
LED1.Input1.Link(rand.Result);   //Make target1 high if
LED1.Input2.Link(rand.Result2);  //both result and result
LED1.Output.Link(target1);       //are low
LED1.InvertOut = cvTrue;
LED1.Operate = cvTrue;

LED2.Input1.Link(rand.Result);   //Make target2 high if
LED2.Input2.Link(rand.Result2);  //result = 0 and
LED2.Output.Link(target2);       //result2 = 1
LED2.InvertIn2 = cvTrue;
LED2.InvertOut = cvTrue;
LED2.Operate = cvTrue;

LED3.Input1.Link(rand.Result);   //Make target3 high if
LED3.Input2.Link(rand.Result2);  //result = 1 and
LED3.Output.Link(target3);       //result2 = 0
LED3.InvertIn1 = cvTrue;
LED3.InvertOut = cvTrue;
LED3.Operate = cvTrue;

LED4.Input1.Link(rand.Result);   //Make target4 high if
LED4.Input2.Link(rand.Result2);  //result and result2
LED4.Output.Link(target4);       //are high
LED4.InvertIn1 = cvTrue;
LED4.InvertIn2 = cvTrue;
LED4.InvertOut = cvTrue;
LED4.Operate = cvTrue;
//*****RANDOMIZER GATES SETUP*****//

while(1)
{

//When time runs out, execute Stop code

if( (digit1.Value == 0) & (digit2.Value == 0) )
Stop.Operate = cvTrue;
}

}
```

```
//This function stops all of the counters and dividers,
//ending the game
sub void Stop_Code(void)
{
divide_clock.Operate = cvFalse;
divide_count.Operate = cvFalse;
count1.Operate = cvFalse;
count2.Operate = cvFalse;
clock1.Operate = cvFalse;
clock2.Operate = cvFalse;
rand.Operate = cvFalse;

}


//This function starts all of the counters and dividers,
//starting the game
sub void Start_Code(void)
{

divide_count.Operate = cvTrue;
divide_clock.Operate = cvTrue;
count1.Operate = cvTrue;
count2.Operate = cvTrue;
clock1.Operate = cvTrue;
clock2.Operate = cvTrue;


   }
```