DigiDuel: ECE 4510 Project

Fuad Ahmed ([redacted]), Ty D. Freda ([redacted]), Anna E. Goldsworthy ([redacted]), and Lucas M. White [redacted]

I. Introduction

DIGIDUEL the ECE 4510 Microprocessors and Digital Logic project, brings the fun of the Wild West into the lab! Two digital desperados, equipped with infrared (IR) remotes, face off to find the fastest draw in all of CSF. A duel starts when two players agree to a showdown by pressing the start buttons on the table between them. The table holds the microcontroller, buttons, IR sensor, and two servo motors with flags. Once both players agree to duel, a buzzer sounds, and the players turn and walk away from each other until they hear a second buzzer. At some point between 1 and 5 seconds, a third buzzer sounds, and the players must draw their IR remote and aim for the IR sensor. The winner of each round is proclaimed when a flag is raised on their side of the table, and +1 point is added to the streak counter. The game ends when a player reaches a streak of 3, and the sheriff is called. This report will cover the design and testing, and troubleshooting of Digiduel.

II. Game Design

This section will provide information about the game's design and implementation. It will detail the start game, main game, score keeping, and end game components. The fully implemented circuit can be seen in Fig. 1.

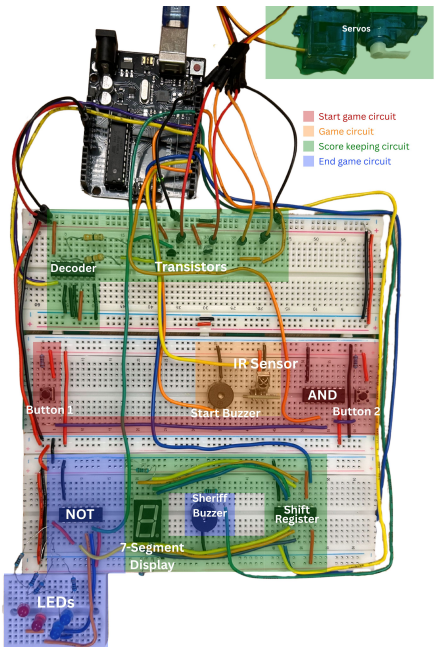

Fig. 1. Fully implemented circuit with labeled game sections.

A. Start Game Design

The game starts by initializing all pins and variables to their start-of-game values. This can be seen in detail in Appendix B. Following initialization, the start button status on the breadboard is checked. The outputs of both buttons are ANDed together using a 74HC08 AND gate. The output of this AND gate is read by digital pin 2 of the Arduino. When both buttons are pressed simultaneously, the `if` condition is met, and the code continues.

B. Main Game Design

Once both start buttons have been pressed, various buzzer-related functions are used. First, `song()` plays a cowboy tune by varying the frequency and duration of notes with the `notes(frequency, duration)` function. Next, `beeps()` notes to indicate when the walk, stop, and shoot. The players should walk no more than 150 cm to maintain accurate IR readings. Three seconds elapse between the first and second notes. Between the second and third notes, the `random(1000,5000)` function is used to select a random time between 1 and 5 seconds. After the final note, the buzzer is disabled and the IR sensor is enabled. The provided IR remotes use standard encoded signals. By using different buttons for different players, a single IR sensor can be used. The sensor receives one signal at a time; therefore, the first signal will determine the duel's winner. Player 1 presses 8 to shoot, while Player 2 presses 5. Once the IR sensor is enabled, the code enters a `do-while()` loop to check for signals with the `checkIR()` function. This function uses the `IRremote.h` library to check which button was pressed and update the score accordingly. If a signal is received, `IR.decode` will return true. Each signal contains a struct `IR.decodedIRData`. This struct contains several attributes to describe the IR signal. The command attribute indicates which button was pressed. Button 5 corresponds to command 64, and button 8 corresponds to command 21. The global variable `button_pressed` is updated to equal command, then `checkwinner()` updates the streak. This will be further discussed in Section II-C Scorekeeping component.

C. Score Keeping Design

There are two different score keeping mechanisms: one for the round score and one for the game score. The round score displays which player shot first after every buzzer-shot cycle. The game score displays a player's number of successful shots in a row. The circuits for both of these mechanisms are shown in Appendix A. When a signal is received, the `checkwinner(button)` function is used to determine the

round winner. First, the function checks to see if the winner is the same as the last round. It does so by checking if `button` equals `last_winner`, a global variable initialized at the start of the code with the value 0. If the winner is the same, `streak` is incremented by 1; if not `streak` is set to 1. If the button pressed was 8, a signal is sent to raise player one's flag, and vice versa for button 5 and player 2. The details of this signal are provided in Section II-D Round Score. If the button pressed was not 5 or 8, the shot is invalid, and the IR sensor continues to check for an incoming signal.

D. Round Score Design

The round score mechanism uses one 74HC138 decoder, two 2N222 NPN transistors, and two servos. This circuit schematic can be seen in Appendix A. When the servos are not in use the decoder is sent a 11_2 such that $\overline{Y1}$ and $\overline{Y2}$ are high, supplying both servos with power through the NPN transistors. The servos are both supplied the PWM signal for angle 0. This is done through the use of the `Servo.h` library. The NPN transistors are used as switches for enabling the servos. When a player wins a round the decoder is sent a 01_2 or 10_2 signal inversely corresponding to the player number. This causes $\overline{Y1}$ or $\overline{Y2}$ to go low, disabling the servo. While the loser's servo is disabled, the winning servo is sent the PWM signal for angle 90. After the servo is lowered, the decoder is sent 11_2 and both servos are supplied power once again.

The round score mechanism uses one 74HC138 decoder, two 2N222 NPN transistors, and two servos. The NPN transistors in the round score mechanism act as switches for enabling the servos. Either 01_2 or 10_2 is sent to the decoder from the Arduino, indicating which servo should be on. The base of one transistor is connected to $\overline{Y1}$, while the other is connected to $\overline{Y2}$. All other pins are disconnected. $\overline{Y1}$ and $\overline{Y2}$ represent the decoder receiving 01_2 or 10_2, the only two values that will be sent to the decoder by the Arduino. For 01_2 and 10_2, $\overline{Y1}$ and $\overline{Y2}$ are always in opposite states, resulting in only one servo having power. When resetting the servo, 11_2 is sent to the decoder. Given that the decoder is active low, this will set both NPN transistors to conduct, as $\overline{Y1}$ and $\overline{Y2}$ would be inactive, thus high [2]. This resets both servos to move to the zero position at the end of each round. A PWM signal is sent to the servos to control their movement. Given that the servos only move between two angles, and one is off, only one signal is required to control both servos. This was achieved using the Arduino Servo library, which has a single function for writing angles to the servos.

E. Game Score Design

The game score uses one 74HC164 8-bit serial in parallel out shift register and one 7-segment display. For score keeping, the group implemented a streak counter. The streak is tracked on the Arduino and displayed on a seven-segment display. Initially, the design used a counter and individual logic circuits. However, the SOP expressions for each number could not be simplified through Karnaugh maps. Instead, the group elected to use an 8-bit serial in parallel out shift register, as shown in Appendix A. Using the shift register allowed a 7-bit number to be sent from the Arduino. This was achieved using the `sendSerialNum` function. However, the number displayed is not what is passed into the function. Through a truth table, the group was able to determine which binary numbers corresponded to the displayed numbers. All of these integers are stored in the `active_high` array, as shown in Appendix B. Since the display is off for zero, the array only corresponds to digits one through nine. On every loop, the streak minus one is passed into the function, to ensure proper indexing. The register is first cleared, to ensure the correct number is loaded, and remove the need for tracking the current value. The function then extracts every bit from the decimal number starting from the most significant bit. The Arduino then pulses them to the serial A pin on the register. A corresponding clock pulse is also sent from the Arduino, as the shift register requires a clock pulse to time the inputs. The delay between the serial data being passed in and the clock turning low is 2 microseconds, as the setup time for the shift register is only 10 ns [1].

F. End Game Design

The end of game mechanism uses one buzzer, one 74HC04 inverter, four LEDs, and four $330\,\Omega$ resistors. The circuit schematic for this mechanism can be seen in Appendix A. In the main game loop the `streak` variable is compared to the value 3 after every buzzer-shot cycle. When `streak` equals 3 `arrest_lights()` is used to send an alternating high-low signal from the Arduino to the LEDs and play a siren sound effect from the buzzer. One pair of LEDs receives the digital signal, while the other pair receives the inverse. This allows the LEDs to alternate while only using one input signal. Due to the configuration of the 7-segment display, Segment C is only illuminated when three is displayed. To ensure that the LEDs are off for all numbers except three, the signal for Segment C is inverted, then connected to the cathodes of the LED array. Thus, when Segment C receives a high signal, the cathode of the LEDs gets a low voltage, which allows them to start conducting and emitting light.

G. Additional Features

Additional features could include a tilt sensor and a photoresistor to increase accuracy and ensure fairness. The tilt sensor would ensure accuracy when targeting the opponent. This would force players to aim the remote, instead of only pressing the button. The photoresistor holster could be implemented to ensure that the IR remote is held at waist level prior to the buzzer sounding. This will ensure that a player cannot fire before the final buzzer. These features were not implemented because they would involve long wires from the player to the Arduino which would impede the movement aspect of the game.

III. Testing Protocols

Each individual subsystem and feature was first tested independently. Upon attaining the desired results from each subsystem, they were then combined and implemented together. The independent testing allowed the group to develop

and troubleshoot the individual subsystems in parallel. This ensured that the development of each system could be completed within the allotted time for the project. This testing protocol allowed for each subsystem to be thoroughly tested, adjusted, and understood by the group.

A. Start Game Testing

The starting buttons and AND gate were the first to be wired, coded, and tested. They were routinely tested thereafter as new additions were implemented. Testing involved digitally printing the Arduino input readings using `serialPrintln()` in the Arduino sketch code, printing "HIGH" when both buttons are pressed and "LOW" otherwise. In this case, the tested results were compared to the AND gate's truth table. Additionally, a multimeter was used to check voltage levels at different points among the circuit, confirming the placement and functionality of the buttons and wiring.

B. Main Game Testing

The tune of the buzzer was tested for ease of recognition with classmates. No further testing was required. The `note()` and `delay()` functions were used to test walk, stop, shoot buzzer. The length of delay between notes was tested by simulating a round and measuring the length of time it took to walk from the centre to 150 cm from the IR sensor. Next, to test the true randomness of the third note, a timer was used to track the time between the second and third notes to ensure values were random each round. Test results confirmed that implementation was correct. The next individual subsystem of the circuit to be tested was the game circuit, containing the IR sensor and the buzzer used for the cowboy tune and walk, stop, shoot notes. The IR sensor was tested rigorously as it was integral to the game. To test if interference from simultaneous controller inputs would be an issue, the technique used in Section III-A Start Game Testing was also applied. The received input was printed while the buttons 8 and 5 were pressed rapidly. The received values were monitored to check for interference. The tests confirmed the IR sensor functioned as required. The range of the IR sensor was tested by running a while loop to constantly take readings from the sensor while a team member slowly moved away from the sensor until readings became less consistent. This distance, approximately 150 cm, was deemed the outer limit of its range. This test was repeated when the full circuit was implemented and results remained consistent.

C. Scorekeeping Testing

The 7-segment display was tested independently upon initial implementation. After connecting the segments to their corresponding pins, a script was written to manually send numbers to the display. Upon all numbers between one and nine displaying correctly, the display was implemented with the rest of the game. Once implemented, the game score system was tested with various scenarios. These included resetting to zero (blank display) at the start of the game, and responding appropriately to players' performance. The servo control system was also tested independently prior to full implementation. Initial tests failed as the power draw of the servos was beyond the capacity of the decoder. This issue is further discussed in Section IV-A Servo Troubleshooting. The implementation of the 2N222 NPN transistors were tested by connecting the base to the 5V rail to ensure they functioned properly. Next the system was tested by connecting the decoder, transistors, and servos. The system worked as expected when sent 00_2, 01_2, 10_2, and 11_2 signals.

D. End Game Testing

The sheriff buzzer and LED array were also tested independently upon first implementation. The buzzer was connected, and the `tone()` function was used to send the siren sound. Initial testing of the LEDs was performed by wiring the red and blue LEDs to separate digital pins of the Arduino in order to alternate flashing. To reduce the number of digital pins used, a NOT gate was implemented as discussed in II-F End Game Design. Testing was repeated once integrated into the full system and results were as expected.

E. Full Game Testing

Once independent tests of each subsystem were successfully completed the full game was tested through various scenarios. The first test case involved one player shooting at a time to confirm the corresponding servo activated and the score incremented. This was repeated for the second player. The next series of tests involved incrementing score, resetting streaks and ending the game. Finally, the authentic game experience was simulated. When the winner changed, the streak would reset to one. When a player won consecutively, the streak would increment. When the streak reached three, the sirens would activate. All of these results are as intended, and the testing was sufficient in determining the behavior of the circuits during the game.

IV. Troubleshooting and Results

Various issues were encountered throughout the design and testing of various components. The three main issues involved the servos, buzzer and IR sensor, and 7-segment display. The solutions to these issues are detailed in the following subsections. The troubleshooting process enabled the group to correct issues and use a broader range of learning outcomes from ECE 4510.

A. Servo Troubleshooting

The initial round score system design did not include transistors. The VCC of each servo was connected to $\overline{Y1}$ and $\overline{Y2}$. This causes issues as the current limit of the decoder is 5.2 mA, while the servo draws up to 50mA when moving [2] [3]. Thus, sufficient voltage would not be supplied to the servos, and they would not move. The decoder would output 2.2V, as it was in an overcurrent condition. To mitigate this, the group implemented the NPN transistor circuit. NPN transistors require a small current to turn on. The base of the transistor is connected to the output from the decoder through

a 330 Ω resistor. The collector is wired to the 5V rail, and the emitter is connected to the VCC pin of the servo. This solution successfully avoided the overcurrent condition of the decoder.

B. Buzzer and IR Receiver Troubleshooting

Another issue was encountered with the buzzer and the IR sensor. When tested independently, the IR sensor returned values consistent with button presses. However, when combined with the buzzer, the sensor returned zero, regardless of the input. This was discovered to be an issue with the libraries and Arduino timers. Both the `IRremote.h` library and buzzer control use the same timer. At the start of the game the buzzer would sound and continue to use the timer even when the IR sensor was enabled. This would interfere with the IR timing and return a constant 0 to the Arduino. This was remedied by disabling the IR sensor before the buzzer sounded and enabling it after it had stopped. This also prevented pre-firing. The IR is not monitored until shooting is expected, so any presses before the start are not registered.

C. 7-Segment Display Troubleshooting

Initially, the group planned to implement the 7-segment display streak counter with a counter and a logic circuit. However, after developing and analyzing the K-maps for each digit, as seen in Appendix C, no SOP expressions could be simplified. Instead, a serial in parallel out 8-bit shift register was used. This solution required only one integrated circuit (IC) instead of several and it reduced the number of digital pins from eight to three. The group assigned seven output pins from the shift register to the 7-segment display. Then, developed the truth table for each segment, seen in Appendix C, and converted them from binary to decimal. The Arduino takes the binary number, then extracts seven bits, essentially converting the number back to binary. These bits are shifted into the register from the Arduino, from MSB to LSB. This simplifies the circuit and code, as a single array stores the necessary values for displaying numbers between 1 and 9. The register clock and clear are also controlled by the Arduino, allowing the display to be reset and changed easily. This led to minimal delay when changing numbers and a simplified, effective circuit.

V. Conclusion

Several concepts covered in ECE 4510 were applied in the design and implementation of DigiDuel. Each subsystem presented unique challenges and design considerations. By designing and testing each circuit individually, the group was able to develop and debug each subsystem thoroughly and in parallel. Once all subsystems were implemented together, the full game was tested and optimized for best player experience. Through debugging and concept generation, the number of digital pins required for the game were reduced. This allowed the group to use one Arduino instead of two, which avoided complex communications errors and kept the game responsive and fast. This also allowed the game to use a broader range of learning outcomes from ECE 4510 by implementing systems in hardware and software. By using ICs for the 7-segment display, sheriff LEDs, and flag servos, there were many simplifications to the code and several digital pins left unused. Through using built-in libraries for the servo, IR sensor, and buzzers, the code remained concise and reliable. The game was designed, tested, and implemented successfully.

References

[1] Texas Instruments, *SN74HC164 8-bit parallel-out serial shift registers*, Datasheet, 2023. [Online]. Available: https://www.ti.com/lit/gpn/SN74HC164

[2] Onsemi, *MM74HC138 3-to-8 line decoder/demultiplexer*, Datasheet, Rev. 2, 2024. [Online]. Available: https://www.onsemi.com/pdf/datasheet/mm74hc138-d.pdf

[3] "Servo Motor Micro SG90," *ProtoSupplies*. [Online]. Available: https://protosupplies.com/product/servo-motor-micro-sg90/

APPENDIX A
CIRCUIT DIAGRAMS

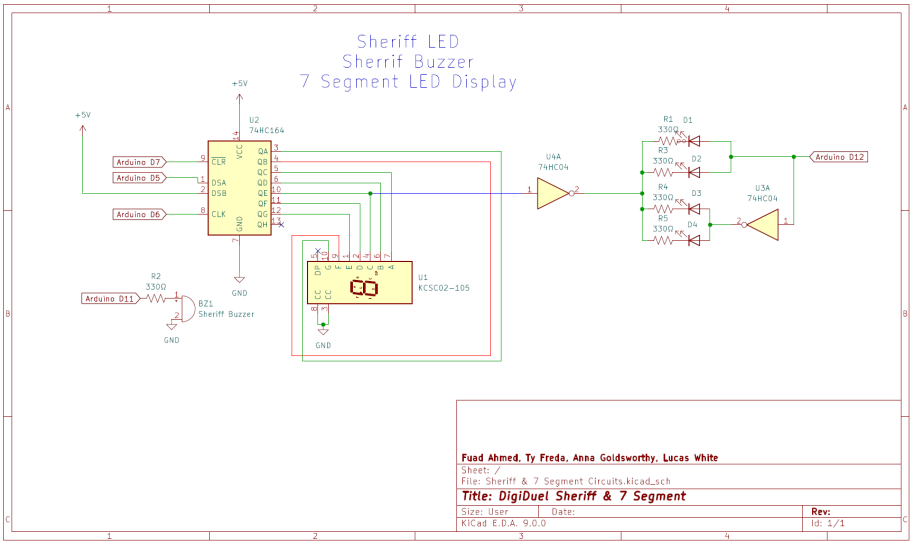

Fig. A.1. Circuit Diagram for End Game Circuit

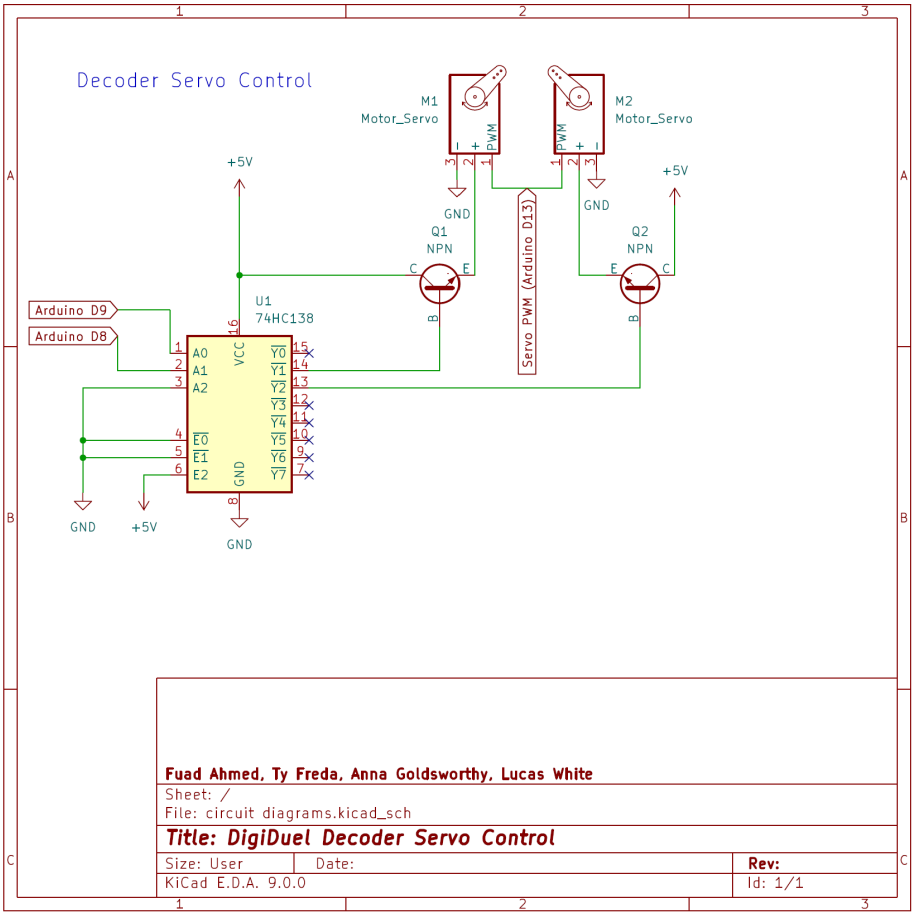

Fig. A.2. Circuit Diagram for Decoder Servo Control Circuit

APPENDIX B
COMPLETE CODE

```
#include <IRremote.h> //Library for IR
#include <Servo.h> //Library for controlling servo

//Globals for pin naming, since pin numbers are used often
#define IRPIN 4

#define BuzzPin 10
#define NoisePin 0
#define ReadyPin 2

#define sr_a 5
#define sr_clr 6
#define sr_clk 7

#define Arrest_Buzzer 11
#define LED_Pin 12

#define decode_a 9 //MSB sent to register
#define decode_b 8 //LSB sent to register
#define servo1_pwm 13

Servo servo_1;

int angle = 0; //Idle angle of servo
int servo_up = 90; //Angle of servo to indicate winner

int last_winner = 0; //Last winner set to 0, so no streak when game is fully restarted
int streak = 0; //Streak set to 0
bool shot = false; //No shot

//Array for storing the serial values that must be passed into the Arduino for the 7-Segment
int active_high[9] = {66, 109, 61, 27, 55, 119, 28, 255, 63};


IRrecv IR(IRPIN); //Attach IR Receiver library to Pin 4

void reset_servo(){
  //Sends 11 to the shift register, meaning both Y1 and Y2 are high. servos have power
  digitalWrite(decode_a, HIGH);
  digitalWrite(decode_b, HIGH);
  servo_1.write(0);
  Serial.println("Reset");
  delay(500);
  //Sends 00 to the shift register, meaning both Y1 and Y2 are high. servos have power
  digitalWrite(decode_b, LOW);
  digitalWrite(decode_a, LOW);
}

void clearRegister(){
  //Clock pulses and clear pulses to reset register
  digitalWrite(sr_clk, HIGH);
  digitalWrite(sr_clr, LOW);
  delayMicroseconds(2);
  digitalWrite(sr_clk, LOW);
  digitalWrite(sr_clr, HIGH);
```

```
}

void arrest_lights(int pin = Arrest_Buzzer //default pin is the arrest buzzer){
  //Both buzzers are off
  noTone(BuzzPin);
  noTone(pin);
  //LEDs are set LOW, only one pair turns on
  digitalWrite(LED_Pin, LOW);
  //Siren sound LOW sent to buzzer
  tone(pin, 400);
  delay(275);
  noTone(pin);
  //LEDs are set HIGH, only opposite pair turns on
  digitalWrite(LED_Pin, HIGH);
  //Siren sound HIGH sent to buzzer
  tone(pin, 1200);
  delay(275);
}

void sendSerialNum(int num) {
  clearRegister();

  // Shift out 7 bits (Bits 6-0), MSB first
  for (int i = 6; i >= 0; i--) {
    digitalWrite(sr_a, (num >> i) & 0x01);  // Extract bit
    digitalWrite(sr_clk, HIGH);
    //Register clock pulse with 2ms delay (10 ns setup time for IC)
    delayMicroseconds(2);
    digitalWrite(sr_clk, LOW);
  }
  Serial.print("Sent: ");
  Serial.println(num, BIN);
}

void note(int freq, int length){
  //Function to reduce code repetition with delays
  tone(BuzzPin, freq);
  delay(length);
  noTone(BuzzPin);
  delay(50);
}

void song(){

  //Start song
  note(392, 80);
  note(523, 80);
  note(392, 80);
  note(523, 80);
  note(392, 800);

  note(311, 450);
  note(349, 450);
  note(262, 1200);
}

void beeps(){
    delay(1000);
```

```
    note(500, 500);  //begin walk away buzzer
    delay(3000);
    note(500,500); //end walk away buzzer
    delay(random(1000, 5000));
    // tone(10, 1000);//turn and shoot buzzer
    note(1000, 400);
}

void checkwinner(int button){
  if (button == last_winner){
    //Compares current input against last input, increments if same, sets to 1 if different
    streak += 1;
  } else {
    streak = 1;
  }

  if (button == 21){
    // 10 is sent to the shift register, setting Y2 LOW, and cutting power to servo 2
    digitalWrite(decode_a, HIGH);
    digitalWrite(decode_b, LOW);
    delay(400);

    Serial.println("Player 1 shot first");
    //Winner is stored as last winner, for comparison in next round
    last_winner = 21;
    //Sets shot to true as input is detected
    shot = true;

  }

  else if (button == 64) {
    // 01 is sent to the shift register, setting Y1 LOW, and cutting power to servo 1

    digitalWrite(decode_a, LOW);
    digitalWrite(decode_b, HIGH);
    delay(400);
    //Winner is stored as last winner, for comparison in next round
    Serial.println("Player 2 shot first");
    shot = true;
    last_winner = 64;

  } else {shot = false;}

  if (shot == true){
    //Servo is set to the high position to raise the flag
    servo_1.write(servo_up);
    delay(250);
    }
}

void checkIR(){
  if (IR.decode()){
    //Command attribute is taken from the struct and
    //passed into the check winner function to determine the winner
    //.command only shows which button was pressed
    Serial.println(IR.decodedIRData.command);
    int button_pressed = IR.decodedIRData.command;
    checkwinner(button_pressed);
```

```
    IR.resume();
  }
}

void setup() {
  //Random seeding
  randomSeed(analogRead(0));
  //PinModes are set
  pinMode(BuzzPin, OUTPUT);
  pinMode(ReadyPin, INPUT);

  pinMode(sr_a, OUTPUT);
  pinMode(sr_clk, OUTPUT);
  pinMode(sr_clr, OUTPUT);

  pinMode(LED_Pin, OUTPUT);
  pinMode(Arrest_Buzzer, OUTPUT);

  pinMode(decode_a, OUTPUT);
  pinMode(decode_b, OUTPUT);

  //Servo is attached to output digital pin
  servo_1.attach(servo1_pwm);

  //Shift register clear is set high,
  //register will not clear as CLR is active low
  digitalWrite(sr_clr, HIGH);

  Serial.begin(9600);

  //Servos are reset
  reset_servo();

}

void loop() {
  //Shot set to false to restart round
  shot = false;
  //Sets a value to avoid floating pin
  digitalWrite(LED_Pin, HIGH);


  if (digitalRead(ReadyPin) == HIGH){ //Checks if both buttons are pressed (AND gate output)
  IR.disableIRIn(); //IR Receiver is disabled
  delay(250);
  song(); //Starting song is played
  beeps(); //Walking/timing beeps are played
  delay(50);
  IR.enableIRIn(); //IR is enabled now that tone is not being used

  do {
    //Check for IR inputs
    checkIR();
  } while (shot == false) //Will exit loop if there is a shot;

  IR.disableIRIn(); //IR disabled after shot is detected

  if (streak == 3) {
```

```
    //If the streak is 3, the arrest LEDs and buzzer turn on
    sendSerialNum(active_high[streak - 1]);
    for (int i = 0; i <15; i++){
      arrest_lights(Arrest_Buzzer);}

    //Streak is reset to restart game
    streak = 0;
    sendSerialNum(streak);

  } else {sendSerialNum(active_high[streak - 1]);}
//If streak < 3, streak is written to the display

  noTone(Arrest_Buzzer); //Both buzzers are turned off
  noTone(BuzzPin);
  delay(800); //Delay to allow servos to move to position
  reset_servo();} //Servos are reset to 0
  }
```

APPENDIX C
KARNAUGH MAPS

Segment E (Pin 1)				
Qd, Qc/ Qb, QA	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	0	0	0
10	1	0	0	0

Segment D (Pin 2)				
Qd, Qc/ Qb, QA	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	0	0	0	0
10	1	1	0	0

Segment C (Pin 4)				
Qd, Qc/ Qb, QA	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

Segment B (Pin 6)				
Qd, Qc/ Qb, QA	00	01	11	10
00	0	1	1	1
01	1	0	1	0
11	0	0	0	0
10	1	1	0	0

Segment A (Pin 7)				
Qd, Qc/ Qb, QA	00	01	11	10
00	0	0	1	1
01	0	1	1	1
11	0	0	0	0
10	1	1	0	0

Segment F (Pin 9)				
Qd, Qc/ Qb, QA	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	0	0	0	0
10	1	1	0	0

Segment G (Pin 10)				
Qd, Qc/ Qb, QA	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	0	0	0
10	1	1	0	0

Fig. C.1. Karnaugh Maps of Logic Circuit Implementation for 7-Segment Display

	Necessary Output from Register		What number gets loaded into Shift Register A	
Number	**Binary Value**	**Complement**	**Serial Data Sent (Active HIGH)**	**Serial Data Sent (Active LOW)**
1	1000010	0111101	66	61
2	1101101	0010010	109	18
3	0111101	1000010	61	66
4	0011011	1100100	27	100
5	0110111	1001000	55	72
6	1110111	0001000	119	8
7	0011100	1100011	28	99
8	11111111	00000000	255	0
9	00111111	11000000	63	192

Pin #	Segment	Register Output
1	E	Qg
2	D	Qf
4	C	Qe
6	B	Qd
7	A	Qc
9	F	Qb
10	G	Qa

Fig. C.2. Serial Implementation of 7 Segment Using 74HC164 Shift Register