INTERRUPT DRIVEN MICROCONTROLLER PROJECT

CYLON EYES

Shane R. Tinklenberg
Charles J. Young
Matthew W. Van Eps
Lucas J. Nelson

April 22, 2020

This page number heading should not appear on this page.

INTERRUPT DRIVEN MICROCONTROLLER PROJECT CYLON EYES

Shane R. Tinklenberg, Charles J. Young, Matthew W. Van Eps, Lucas J. Nelson
April 22, 2020

I. Introduction

Cylon Eyes are a fun toy that emulate the behavior of the Cylon Centurion in the Battlestar Galactica Series. Comprised of series of LEDs that that flash back in forth in a sequential fashion, building a set of Cylon Eyes is a fun and educational project for individuals interested in learning about embedded microcontroller systems. Although more simple methods are capable of sequentially flashing these LEDs in order, many users may find it advantageous and educational to use internal interrupts to schedule the routines responsible for flashing the LEDs.

Interrupt Driven Task Scheduling ensures reliable and precise execution along with the ability to free up tradition looping routines to handle other necessary computations without involving more elaborate delay() routines. This also prevents wasting CPU clock cycles to run these traditional delay routines. For example, if your loop routine requires functions such as delay(), and a delay is active during the scheduled time of the internal interrupt, the CPU will recognize the interrupt flag, store the status of the all of the CPU registers, and then proceed to execute the Interrupt Service Routine. Following the execution of the interrupt service routine, the CPU will reload the previous states of the CPU registers and continue to execute the previously active routine.

Because it is a simple task, there are a large variety of microcontrollers capable of implementing Cylon Eyes on the market. Any microcontroller with an internal timer

and I/O capable of driving the number of LEDs desired within the system with a proper amount of current should suffice. In the lab, it has been demonstrated that an ATMEGA-328P based Arduino Uno Development board is more than capable of properly driving eight LEDs each at a rated continuous current of 10 mA. The Arduino platform provides a simple and easy to use Integrated Development Environment with a well-established set of libraries that allow for quick and efficient development.

II. Hardware

The lab implementation of Cylon eyes utilized a single Arduino Uno Development Board, eight 330-ohm resistors, and eight 10 mA blue LEDs. Each resistor was connected to one of the eight PORTD parallel port pins (Pins zero through seven on the Arduino) in a sequential fashion such that bit zero was connected to the right most LED and bit 7 was connected to the left most LED. The opposite end of each current limiting resistor was connected to its own individual rail on a breadboard. Then, the eight LEDs were wired such that each current limiting resistor shared a common rail with the cathode of a single LED. The anode of each LED was then connected to a common grounded bus on the breadboard which was then connected back to the ground connection on the Arduino board. A picture of the assembled hardware can be found in Figure 1.1.

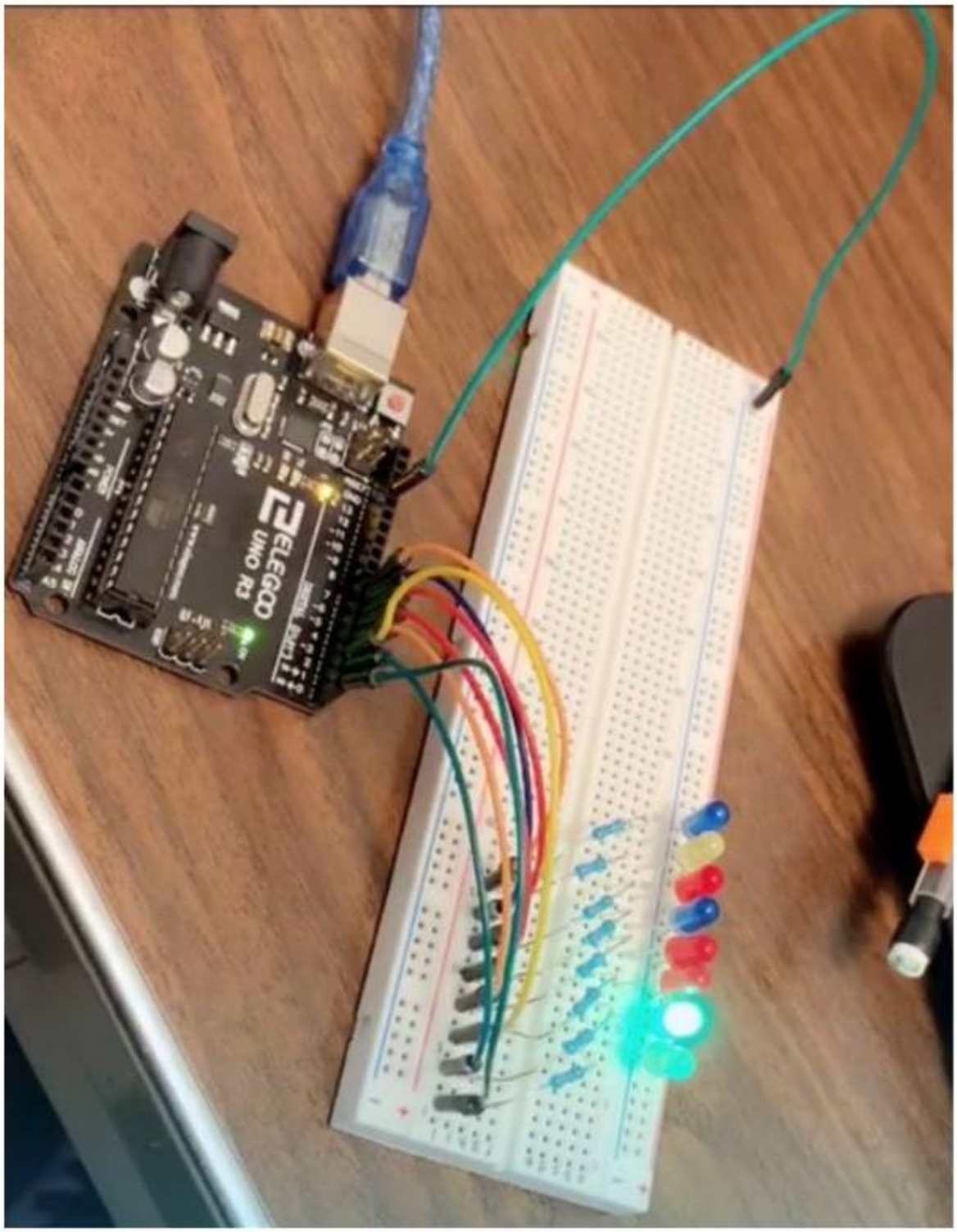

Figure 1.1. Assembled Cylon Eyes.

In this case the combination of the text on page 2 and this photo is something I judge sufficient to allow duplication of the project, but just barely so. A schematic would be a more definitive way to accomplish that goal.

III. Software

One of the greatest struggles when designing software, particularly software that will run on resource-limited systems like a microcontroller, is ensuring that CPU cycles will not be wasted on unnecessary operations like the delay () method in the Arduino Development Environment. Instead, the use of specific resources such as system timer internal interrupts allows users to schedule the execution of specific procedures on a time-based basis. This allows for the design of a loop procedure that does not rely upon specific timing to execute the desired methods and ensures the timely execution of specific interrupt service routines.

The project code utilizes a library titled “TimerInterrupt” written by GitHub user Khoi Hoang (see References for link to the repository). In the setup routine, there are three important procedures. First, the data direction register for Parallel Port D is setup up such that all pins are configured in as an output by setting the data direction register to hexadecimal value 0xFF which represents eight binary one digits in the line DDRD = 0xFF (As seen in Figure 2.1). The program then initializes system timer 1 using the 1Timer1.init(); method. Lastly, the program attaches a void function named TimerHandler1 to the timer compare register using the attachInterruptInterval() method as seen in Figure 2.1. The TIMER1_INTERVAL_MS complier defines dictates how long of an interval will take place between the execution of the interrupt service routine.

```
#define TIMER1_INTERVAL_MS    250

void setup()
{
  DDRD = 0xFF;

  ITimer1.init();
  ITimer1.attachInterruptInterval(TIMER1_INTERVAL_MS, TimerHandler1);
}
```

Figure 2.1. Timer Interrupt Code.

To control the state of our parallel port outputs the program first defines an enumerated type called EYE_DIRECTION_ENUM with three enumerated states: ED_FORWARD, ED_BACKWARD, and ED_NO_DIRECTION. In the TimerHandler1() function the program creates two static variables that remain on the stack even after the completion of the TimerHandler1 function. The two variables represent the direction the program is incrementing the LEDs in an EYE_DIRECTION_ENUM type. The second variable is a single eight-bit unsigned integer that stores the state of the Port D parallel port register. This value is statically initialized to the binary value 0b00000001. The program then uses a series of switch statements that determine which direction to bit shift the singly active bit on the Port D register. The routine is set to execute every 250 milliseconds. The entirety of the program code can be found in Appendix I.

The code would better be placed in the report itself. It is essential to the operation of the project. An appendix is something "added on" for extra interest.

IV. Results

Although it is not the simplest implantation of a Cylon Eyes project possible, designing an interrupt driven Cylon Eyes system serves as a great demonstration of the advantage of using interrupt scheduled tasks within an embedded system to prevent wasted CPU cycles, maintain easy to read code, and ensure the timely execution of specific procedures. However, despite its robust capabilities, it is important to analyze the reliability of systems designed using Interrupt Driven Task Scheduling. System designers must take careful measures to ensure that interrupt scheduling will not cause failure due to prolonged routine durations or routine calls that occur too frequently.

V. Conclusion

After successfully wiring the breadboard and uploading the code into the Arduino, it was demonstrated that it is possible to design Cylon Eyes around Interrupt Driven Task Scheduling. The proof of concept successfully demonstrated the sequential illumination of LEDs in a forward and backward motion at a consistent and predictable rate. Even after adding an infinite for loop with a 10 second delay function call into the Arduino's loop routine, the LEDs continued to illuminate in the same pattern as previously expected while completely unaffected by the change in loop procedure.

Grading

___A-___ Style

___A____ Completeness

___A____ Accuracy

Overall report grade: A

VI. References

Link to "TimerInterrupt" Repository: https://github.com/khoih-prog/TimerInterrupt

The black background wastes ink or toner. Typically one can "negate" the colors to change the black to white if necessary.

VII. Appendix I

```
//These define's must be placed at the beginning before #include "TimerInterrupt.
h"
#define TIMER_INTERRUPT_DEBUG      0

#define USE_TIMER_1     true
#define USE_TIMER_2     false
#define USE_TIMER_3     false
#define USE_TIMER_4     false
#define USE_TIMER_5     false

#include "TimerInterrupt.h"
#include <ISR_Timer.h>

enum EYE_DIRECTION_ENUM
{
  ED_FORWARD = 0,
  ED_BACKWARD,
  ED_NO_DIRECTION
};

void TimerHandler1(void)
{
  static EYE_DIRECTION_ENUM eyeDirection = ED_FORWARD;
  static uint8_t bitState = 0b00000001;

  switch(eyeDirection)
  {
    case ED_FORWARD:
    {
      switch(bitState)
      {
        case B00000001:
        {
          eyeDirection = ED_BACKWARD;
          bitState = (bitState << 1);
          PORTD = bitState;
        }
```

```
        break;

        default:
        {
          bitState = (bitState >> 1);
          PORTD = bitState;
        }
      }
    }
    break;

    case ED_BACKWARD:
    {
      switch(bitState)
      {
        case B10000000:
        {
          eyeDirection = ED_FORWARD;
          bitState = (bitState >> 1);
          PORTD = bitState;
        }
        break;

        default:
        {
          bitState = (bitState << 1);
          PORTD = bitState;
        }
      }
    }
    break;
  }
}

#define TIMER1_INTERVAL_MS    250

void setup()
{
  DDRD = 0xFF;

  ITimer1.init();
  ITimer1.attachInterruptInterval(TIMER1_INTERVAL_MS, TimerHandler1);
}

void loop()
```

```
{

}
```

Interrupt-Driven Platforming Program

Dan Kelly, Zachary Sanford, Ty White, and Ryan Zevenbergen

April 22, 2020

I. INTRODUCTION

IEEE style requires double spacing.

Platform gaming has been a hobby in Western culture for nearly 40 years. Many individuals around the world enjoy the activity of digital gaming. Games such as Super Mario Bros, Space Invaders, Donkey Kong, and Pac Man all use a form of interrupt-driven Input/Output (IO) programming. The goal of this lab was to design and build a microcontroller circuit that would operate with interrupt-driven IO. The Arduino provided an adequate platform for such a device, as digital pins 2 or 3 can act as hardware interrupt pins [1]. A Liquid Crystal Display (LCD) screen was acquired and wired to a breadboard [2]. Then, a program animation of an individual running was drafted. Finally, obstacles for the individual to jump over were generated with a random distribution to give the game some level of difficulty. Measures to ensure no impossible situation arose that was unwinnable for the user. A separate, external button was used as the hardware interrupt. The interrupt would cause the individual to "jump," and miss the obstacle in its way. If the user fails to "jump" the obstacle, the program ends. The current circuit utilizes a reasonably inefficient wiring design, which is one of the areas that it could improve upon. In addition, there is a time delay from when the button is pressed and when the "jump" animation begins. This could be due to code inefficiency or the wiring. Overall, the system runs well enough to be functional and enjoyable.

II. METHODS

a. Idea and Proposal

Many options for Project 2 were discussed amongst the team. An audio amplifier that passively plays music and implements a Public Address interrupt feature was one idea. This idea was disregarded because audio processing with an Arduino is a difficult task and unreliable at best. The second option proposed was the game design that was unanimously decided upon. Certain changes to the program were proposed, such as changing the individual running into a dinosaur and allowing the individual to run on top of the obstacles. The dinosaur idea was disregarded, but the obstacle idea was incorporated into the final design. After the brainstorming session, a proposal was written and approved so that the team could move on to the conceptual design phase.

b. Conceptual Design

First, a program tutorial of how to activate an LCD screen in Arduino was referenced [3]. From this, information on how to write a program incorporating such a device allowed the team to progress with their own design.

c. Design Implementation

The circuit was constructed after the conceptual design was completed. The LCD screen was wired to a breadboard. After consulting documentation for connecting a 1602A LCD Screen to an Arduino Uno [2], the following connections were made. Pins *VSS*, *RW*, and *K* of the LCD is connected to ground (GND). Pin *VDD* of the LCD is connected to +5 V to provide the screen with power. Pin *V0* of the LCD is connected to the wiper pin of a 10 $k\Omega$ potentiometer, with the other ends of the potentiometer are connected to +5 V and GND. The potentiometer exists to change the contrast on the screen to allow the characters to show up lighter or darker. Pin *RS* of the LCD is connected to digital pin 12 of the Arduino Uno. Pin *E* of the LCD is connected to digital pin 11 of the Arduino Uno. Pins *D0-D3* of the LCD are unused and remain disconnected from the circuit. Pin *D4* of the LCD is connected to digital pin 6 of the Arduino Uno. Pin *D5* of the LCD is connected to digital pin 5 of the Arduino Uno. Pin *D4* of the LCD is connected to digital pin 6 of the Arduino Uno. Pin *D7* of the LCD is connected to digital pin 3 of the Arduino Uno. Pin *A* of the LCD is connected to a 220 Ω resistor leading to GND. Lastly, a pushbutton is used as our interrupt connected from GND to digital pin 2 of the Arduino Uno. The interrupt had to be connected to digital pin 2 because that is the only pin available on the Arduino Uno to handle an interrupt input that was not already in use.

III. RESULTS

a. Experimental Data

The first test run of the game had some issues. Originally, the man walked and when the button was pushed, and a stream of blocks came along in a steady stream. The man would then phase through the blocks rather than jumping over them. This was because the code read through an array of ones and zeroes which acted as signals to display a block or a blank space. The problem was found to be in the verification step of the code. The code needed to utilize a a double equals for equality instead of just a single equal for assignment. The single equals set the checksum to always be assigned a value of one, thus signaling that a block should always be generated. After solving this problem, the code went through the array successfully resulting in a stream of random ones and zeroes. The next issue that occurred was that the game would crash if the pushbutton interrupt was held down. Due to the interrupt, the game was constantly flagging the interrupt, causing the game to overload and shut down. Analyzing the code found that the code was always sending an interrupt on a low signal. Changing this to sending an interrupt only on a falling edge allowed only a single interrupt to be sent. The next

issue found was that the array itself did not seem to work when attempting to send a random distribution of blocks. So instead, an array of predefined values simulating a course was generated and set to repeat. The random number generator was eventually debugged using the Arduino forums [4]. Lastly, the final feature added was that the man could jump and land on top of the blocks. If he fails to jump, the game ends. The final product is a working game where the man goes through a course of blocks randomly generated and when the button is hit the man jumps over the blocks. The player's score is shown on the top of the display. If the player loses, the game ends. The button may be pressed again to restart the game.

b. Schematics and Figures

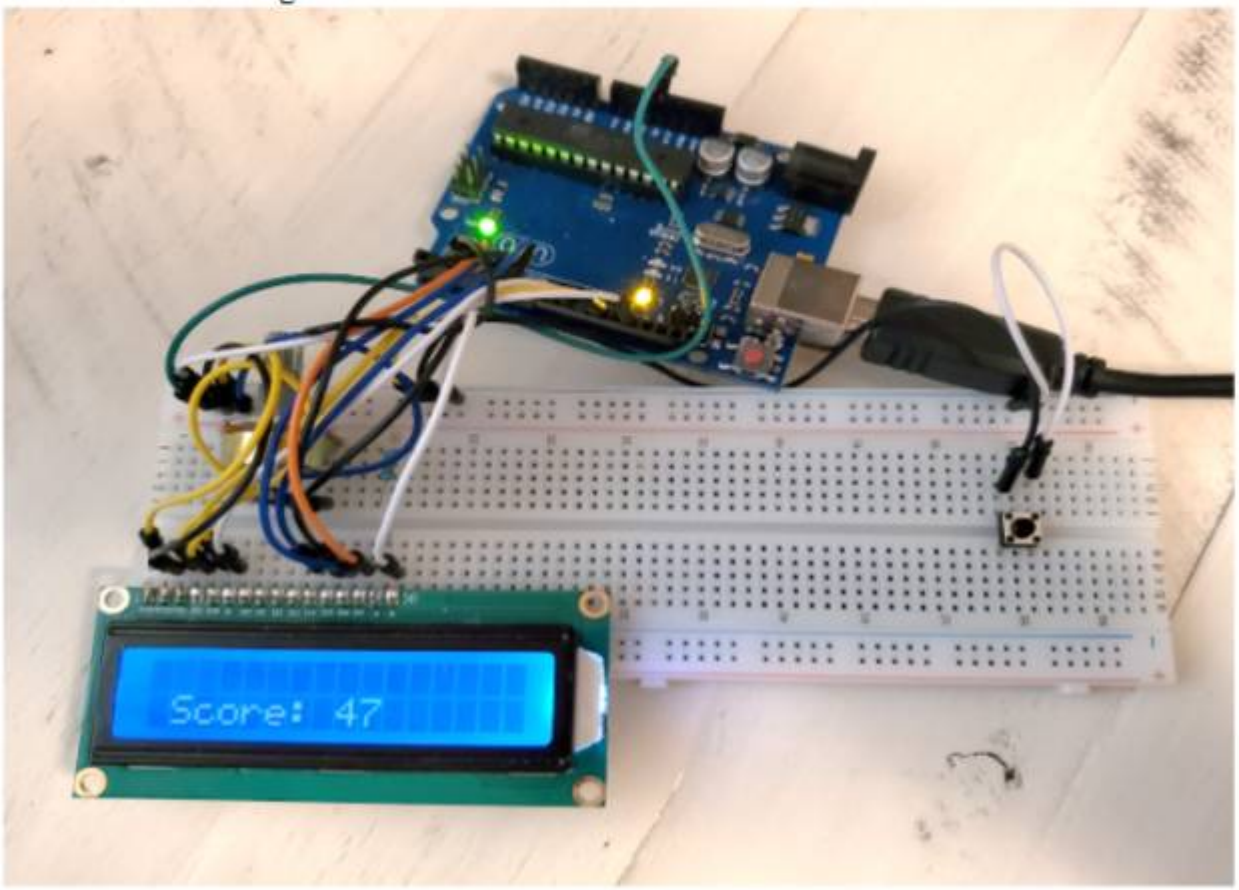

Figure 1--Configuration of Schematic to Arduino

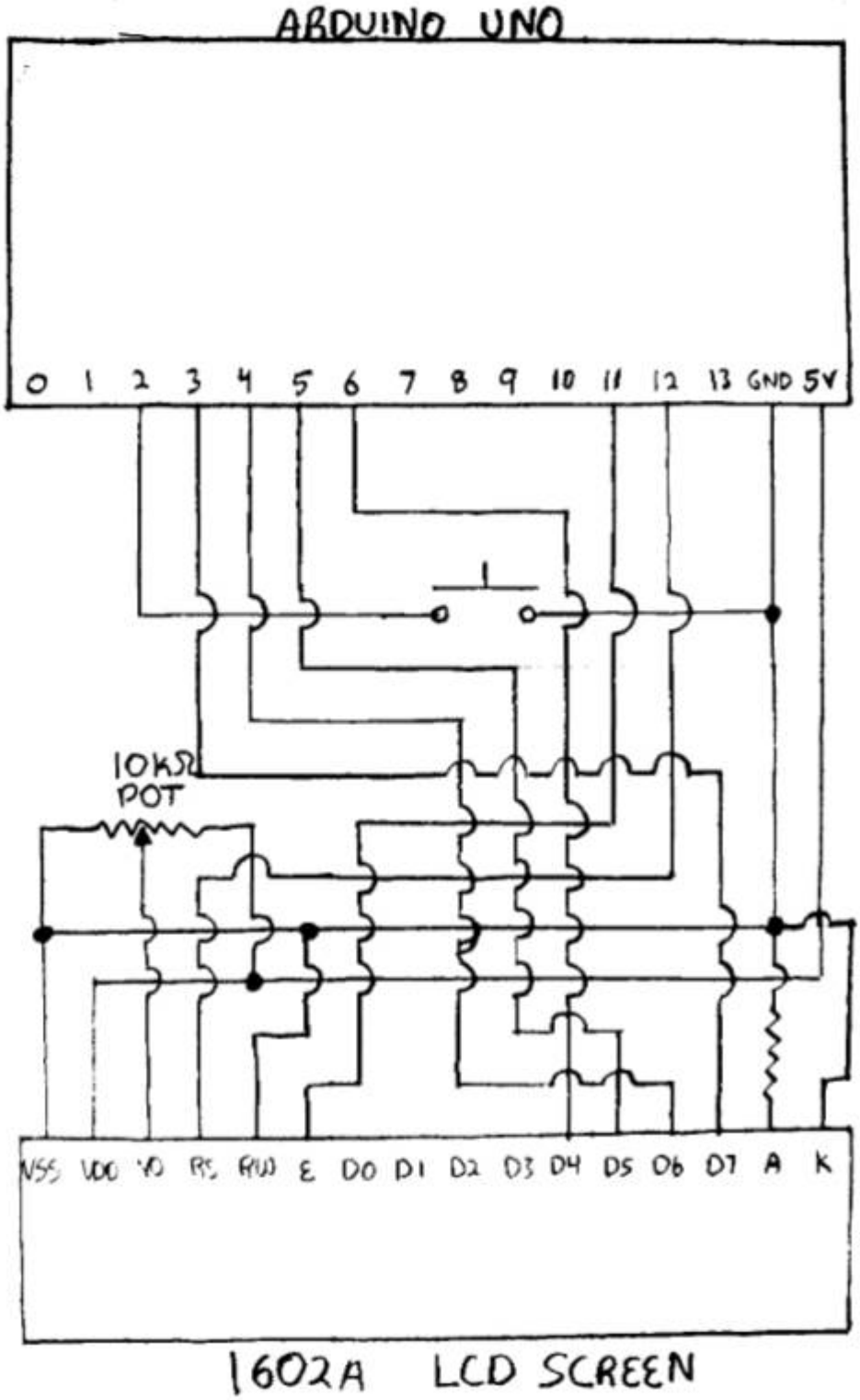

Figure 2--Schematic of Arduino Connection to LCD

IV. CONCLUSION

In conclusion, interrupt-driven IO is a very important concept for engineers to understand. It is especially used in the field of videogames, which are used for entertainment all over the world. For the platforming game designed in this project, the transferring of void loop code to interrupt driven code in the Arduino sketch proved to be successful. Debugging the code with help from the Arduino Forum gave insight into making the program run more effectively [4]. After several issues were encountered and overcome, the circuit's final iteration functions adequately for its purposes to provide entertainment.

V. APPENDICES

a. Appendix I: Materials List

Part	Quantity
Arduino Uno	1
Adafruit 1602A LCD Screen	1
Breadboard (generic)	1
Resistor 220Ω	1
Potentiometer $10k\Omega$	1
SparkFun Pushbutton switch 12mm	1
Jumper wires (generic)	16
USB-A to B Cable	1

b. Appendix II: Arduino Code

```
#include <LiquidCrystal.h> // include the library code:
LiquidCrystal lcd(12, 11, 6, 5, 4, 3);

int level = 1; //level=0 means top of the LCD, level= 1 is the bottom
int pin_button = 2; //button attached to pin 2
boolean jumping = false;
int jumpService = 0;
int speed_of_game = 200;
int score = 0;
int x = 0; //Because arduino is dumb

int landscape[]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int landscapeTemp[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int iteration = 0;

//Below are custom bytes that are used to make custom characters on
//the lCD. A "1" is a white square on the LCD, and a "0" is a blank
//dark spot. Each space on the LCD is 5 columns by 8 rows.
//run animation 0
byte run0[8]={
  B01100,
  B01100,
  B00000,
  B01110,
  B11100,
  B01100,
  B11010,
  B10011,
};

//run animation 1
byte run1[8]={
  B01100,
  B01100,
  B00000,
  B11100,
  B01110,
  B01100,
  B01100,
  B01110,
};

//jump animation 0
byte jump0[8]={
  B01100,
  B01100,
  B01110,
  B11100,
  B01100,
  B11010,
  B10011,
  B00000,
};

//jump animation 1 (head on top level)
byte jump10[8]={
  B00000,
  B00000,
  B00000,
  B00000,
```

VI. REFERENCES

[1] Microchip. *ATmega328P megaAVR Data Sheet.* Microchip Technology. Published 2018. Accessed 8 April, 2020. http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf

[2] Xiamen Amotec Display CO., LTD. *Specifications of LCD Module: ASM1602K-NSW-FBS/3.3V.* Published 29 October, 2008. Accessed 8 April, 2020. https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf

[3] Magdy, Mohammed. *Arduino Game by LCD.* Create Arduino. Published 18 October, 2016. Accessed 25 March, 2020. https://create.arduino.cc/projecthub/muhamd-magdy/arduino-game-by-lcd-9a3bc2

[4] Arduino Forum. *Arduino Forum and Help Instructions.* Accessed 1 April, 2020. https://forum.arduino.cc/ Published 29 October, 2008. Accessed 8 April, 2020. https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf

[4] Arduino Forum. *Arduino Forum and Help Instructions.* Accessed 1 April, 2020. https://forum.arduino.cc/

Grading:

___A-___ Style (Not double-spaced)

___A____ Completeness

___A____ Accuracy

Overall Grade: A

Interrupt Driven Cylon Eyes

Patrick Munsey, Kyle Waas, Nolan Vande Griend, and Stefan Walicord

Professor De Boer

EGR 304 Lab, Dordt University

April 15th, 2020

Headers are not IEEE style.

I. Introduction

This lab's purpose is to implement a microcontroller hosted, interrupt-driven program, which the team fulfilled by using an Arduino that powers five LEDS in a row to create the Cylon eyes effect, where the light moves up and down continuously. The project was named *TimeInterrupts* and was developed in the Arduino IDE in C code. The LEDs, breadboard, Arduino system, and 1000 ohm resistors were all provided by Dordt University's Engineering Department.

II. Method

The team were limited to 1000 ohm resistors because the hardware was assembled at home, meaning lower resistance values were unavailable. The LEDs were installed on the board and faced the same direction without overlapping so that the wiring would not accidentally create a short-circuit. The LEDs were connected through pins 8 through 12 on the Arduino with the GND pin connecting to a common column node. The wiring from the Arduino nodes went to a resistor and then to the rounded side of the LED, and then into the common ground.

There is just enough information here, along with Figure 2, to recreate the circuit. A schematic would have been a more professional and concise way to convey this information.

On the programming side, the team found an Arduino library called MsTimer2 that creates a periodic interrupt which calls the function. In the *TimeInterrupts* code, this MsTimer2 interrupt triggers every 300 ms and calls the *flash* function. This *flash* function works with two variables: *current* and *currentDirection*. The first keeps track of which LED activates next and the following tracks which direction the light path moves. MsTimer2 calls *flash* which turns the

Calls which function? Your reader stumbles here hand has to figure something out that could have been directly stated. (Poor grammar--missing antecedent.)

Your code shows "500 ms." It is sort of an insignificant factual error, but it leaves the reader with a taste of nonchalance that reduces the veracity of the entire report.

correct light on, then off, and increments the variables up or down as needed so the correct light would be called in the next interrupt (Figure 1).

III. Results

Other than some struggles getting LEDs to light up due to some incorrect resistor values (see Methods), the program executed without error, and timing values were trimmed to make the Cylon eyes effect look as intended.

IV. Conclusion

The team successfully implemented an interrupt-driven version of Cylon eyes without major problems or troubleshooting. The biggest difficulty was finding an interrupt library for Arduino to use, but MsTimer2 ended up being versatile, and the use of *flash* as a separate function that did most of the work enabled the team to avoid more difficult topics like interrupt scheduling. A secondary difficulty lay in sharing files through Discord, which were not received well by the Arduino program; this is remedied by sharing the file by either copying the entirety of the file and pasting it as text or sharing the file through Outlook or another file sharing service.

V. Appendix

The code is not an "add-on" for curiosity. It is at the heart of this report, thus it should not be an appendix. It should be a figure.

```
// Toggle LED on pin 13 each second
#include <MsTimer2.h>

 int current = 8;
 int currentDirection = 1;

void flash() {
     static boolean output = HIGH;
     int wait = 5000;

     Serial.print("Now Running through Flash");

    digitalWrite(current, HIGH);
    delay(wait);
    digitalWrite(current, LOW);

   //switch our direction
   if(current == 12){

     Serial.print("the direction has changed and is going up");
     currentDirection = 0;
   }
   else if(current == 8){

     Serial.print("The direction has changed and is going down");
     currentDirection = 1;
   }

   //add or subtract from our current LED
   // Direction == 1 means that directions goes down
   // Direction == 0 means that direction is going up
   if(currentDirection == 1){
     current ++;
   }


      else{
        current --;
      }
}

void setup() {

  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);

  Serial.begin(9600);

  MsTimer2::set(1000, flash); // 500ms period
  MsTimer2::start();
}

void loop() {

}
```

Figure 1: Full Code Snips

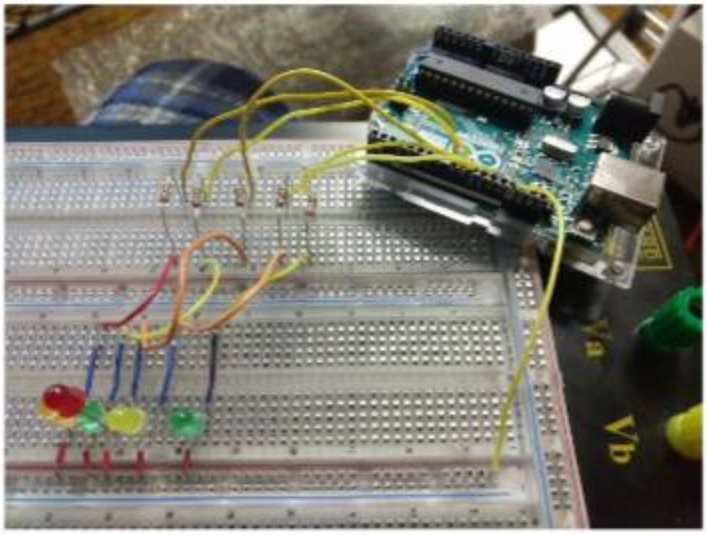

Figure 2: an Example of the Physical Set-up

You have a whole page to use here.
Re-size that photo to fill the page just as wide as a paragraph of text would be. (As wide as this comment.) Keep the aspect ratio unchanged and let the photo be as high as it needs to be. Make illustrations as communicative as possible. Since this photo meets minimum criteria it is OK, but it could have been better.

Grading

___A-___ Style (see annotations)

___A____ Completeness

___A-___ Accuracy (300 ms or 500 ms?)

Overall report grade: A-