

# *Reactive Lights with Assembler*

*Kevin Morales-Nguyen*

*Devon Hood*

*Mico Guinto*

*Tyler Eastman*

Nuwave.flac

5/28/2020

## Team Members and Roles

Kevin Morales-Nguyen: Project Lead

- Main goal is to see that the project comes to fruition. This means working closely with all members to assure that development and deployment proceeds smoothly. Making sure that all members are on the same page and that milestones are met on time.

Devon Hood: Assembler Architect

- In charge of researching, developing and testing the software that embedded systems of the Arduino platform operate on, Atmel/AVR.

Mico Guinto: Assembler Architect

- In charge of researching, developing and testing the software that embedded systems of the Arduino platform operate on, Atmel/AVR.

Tyler Eastman: C Architect

- In charge of developing and maintaining the high level software that the Arduino platform operates on C.

Shared roles

- General development in C and assembly. Collaborations in methodologies, ideas, and implementations of code. Team members were responsible for being active and innovative in the development and problem solving of all complex problems faced in the development of the project.

## Project Summary

Our project comes in two forms, one implemented inside the Arduino development environment using C code, and the other implemented inside of ATMEL Studios utilizing assembly code. Unfortunately, we discovered some difficult conversions from C to assembly code and modified our project in such a way to reflect our progress in understanding and implementing assembly code in line with our project's central idea. Below I will outline our projects two forms:

### 1. [Arduino Development Environment C]

Our first and foremost plan was to implement use of the Arduino Uno board and create a simple machine that could react to external audio input. We set out to complete this by translating the changes in voltage from our given audio input and output similar reactive changes to an LED light strip. In this regard, we executed flawlessly. Our project does exactly as we planned. Inside of our program we have a few variables we can change in order to customize the colors, intensity, and threshold of the input sensitivity. Project demonstration will be attached in the following sections.

### 2. [ATMEL Studios Assembly Code]

The second phase of our project hit some roadblocks and we were forced to take a different approach on our deliverable. Developing inside of ATMEL using assembly, we decided to work with the core ideas that were in our original project proposal. Using the ideas of input and output, we attempted to implement the same fundamental ideas but were forced to nix the input, given a plethora of problems we faced (mainly the difficulty of maintaining contact with one another given the current situation of the world). Our project with assembly code simplifies our idea somewhat and instead of utilizing input, we force a delay to our output. The light essentially blinks in sequence with the ability to adjust the delay and color sequence. Although this is unfortunately a over simplified version of our original idea, we were begrudgingly proud to produce this inside of ATMEL Studios.

## Methodology

There were a few key points of this class we were interested in exploring and implementing. First and foremost, we just wanted to develop a simple machine from the most basic components available to us. Computer architecture played a large role in our small machine. We needed to understand how and why our hardware worked, or didn't work, when we placed pins in certain areas and uploaded a simple program into the board. This led to our understanding of the simplistic nature of the Arduino Uno board and its inherent and blatant limitations. Next on our priority list, was to implement our working project into low-level assembly code. Our project attempted to morph high-level code into basic machine language and utilize registers, vectors, and simple instruction sets to complete the same tasks as its sister program could in C. We researched and implemented in/out functions in C and successfully implemented out-functions in assembly. While the actual implementation of input in our final project inside of assembly was removed (for the time being), we spent extensive time researching and understanding the methodologies needed to eventually implement this function into our simple machine. As a general project goal when we successfully completed a task, we attempted to scale.

## Results

With heavy hearts we report that our final project removed some components within assembly code. The following is an overview of our project's completed functioning components:

1. Inside of the Arduino IDE our project works exactly as intended. We can take audio input translated into voltage readings, compare those readings with our given threshold, and when given an acceptable value we have our LED light strip flash in the specified color. Variables that are adjustable inside of our program are: LED color, threshold value (also adjustable through our hardware), and light intensity.
2. Inside of ATMEL Studios IDE we had to modify our project given the strenuous situation. We wanted to stay within our core ideas and implement input/output with audio and LED lights. Unfortunately, implementing the input proved to be a difficult task we did not foresee and had to force a delay into our LED output rather than take external input. Our project with assembly code is essentially a flashing sequence of lights with a small delay. Variables that are adjustable inside of the assembly program include: LED color sequence and delay time.

## Live Demonstrations

Revised/Modified C code project:

[https://www.youtube.com/watch?v=I96FUxHCw\\_c](https://www.youtube.com/watch?v=I96FUxHCw_c)

In this video you can see the modified functionality of the original project. Mid-way through the video if you notice the Arduino board is flashed with a new set of instructions to change the LED colors. The sensitivity and colors of this machine are completely customizable.

ASM Code Project:

<https://www.youtube.com/watch?v=Ao2966ekcNw&feature=youtu.be>

In this video we demonstrate how we upload asm code to the mcu and then display the simple led light show that operates on pure avr assembly.

## Reflections

Our project had many things go well and seemingly the same amount go poorly. Our project was mainly about developing a simple machine that “reacted” LED lights to music. In this regard, our project was perfect. Although, this course was also about implementing machine language so we were challenged to translate our program into low-level assembly. In this regard, we fell short of our intended goal. Our high-level language project implemented all the functions and designs we intended for it but our low-level project was simplified in order to achieve a recognizable modification to our original idea. While our project is complete in the sense of its function and development in high-level code, we dropped external audio input from our assembly implementation. In the end, we need to continue working on how to design external input through assembly. It proved to be a tougher challenge than we thought to bring input into our Arduino board through assembly language. We struggled to initialize and set the right registers that could be utilized by our output function. Given the simplicity of the Arduino Uno board we thought that once the LED output was developed, input would surely be the same. We were mistaken. Getting external audio input in assembly proved to be the most complicated portion of our code and, unfortunately for us, was the portion we saved for the end. Given more time in this project (and normal in-person working conditions) we believe that this problem would have been an easier hurdle to pass but were dealt a challenging set of circumstances to deal with. Although the class may be done and this final project report is coming to an end, we hope to add the audio input function through machine language in the future.

As a general reflection into this project, a notable problem arose when attempting to scale any function. Once completed in C, we attempted to add additional audio input to control multiple LED strips. This proved to be challenging and resolved to be a limitation of the Arduino Uno board, not our coding skills. In the future, we would like to utilize more complicated computer architecture in order to implement more complicated and easier scaling of developed functions.

## Conclusion

Overall, we rule this project as a success. Our code project was to develop a small machine which was executed flawlessly. The function developed work simply and correctly. Our code is simple, easy to follow, and able to be modified. In C we implemented input/output functions taking external audio as input, translating the incoming voltage readings, and exporting voltages to flash a LED light strip according to the breaking of a threshold value we set. Translating this project into assembly, on the other hand, proved to be a challenge we did not adequately prepare for. We struggled to implement the same functionality inside both low-level and high-level languages and were ultimately forced to simplify our project inside of machine language. While the functionality lacks an external audio input when developed in assembly language, we believe it still focuses on the core of what we wanted to learn from this course. We utilized registers and simple machine language instruction sets which seemed to be the main focus of our assembly language lessons. While higher level functions were a highlight of the end of our instruction period, we had difficulty implementing them into our project and decided it would be better to have a functioning machine through assembly language which had resemblance to its sister project in C, rather than a non-functioning machine that attempted to match each function exactly. We view this project as having two conclusions. One wildly successful, and one mildly successful. Nonetheless, we enjoyed our experience and learned a great deal about machines, their “thoughts”, and how to instruct them through their most basic languages.



# Codebase

```
1 ;Date: 5/25/2020
2 ;Authors: Kevin Morales, Devon Hood, Mico Guinto, Tyler Eastman.
3 ;
4 ;This program flashes a sequence of colors on a LED light strip.
5 ;The color sequence and delay time can be manipulated to preference
6
7 .include "m328pdef.inc" ; include definitions for atmega328p, this makes it easier to work with I/O registers
8
9 .dseg ; data segment directive
10
11 .def mask = r16 ; define a mask register
12 .def ledr = r17 ; define a led toggle register
13 .def loopct = r18 ; define a register to hold arg for delay subroutine
14 .def threshreg = r20 ; define register to hold threshold value for sensor
15 .def sensorval = r21 ; define register to hold sensor value
16
17 .def oLoopR = r19 ; outer loop register
18 .def iLoopRl = r24 ; inner loop register low
19 .def iLoopRh = r25 ; inner loop register high
20
21 .equ iVal = 39998 ; inner loop value
22 .equ delaytime = 100 ; delay time
23 .equ thresh = 100 ; threshold value for sensor input
24
25 .cseg ; code segment directive
26
27 .org 0x00 ; start at bottom of program memory
28
29 ldi r22, LOW(RAMEND) ; setup stack pointer so we can call subroutines
30 out SPL,r22
31 ldi r22, HIGH(RAMEND)
32 out SPH, r22
33
```

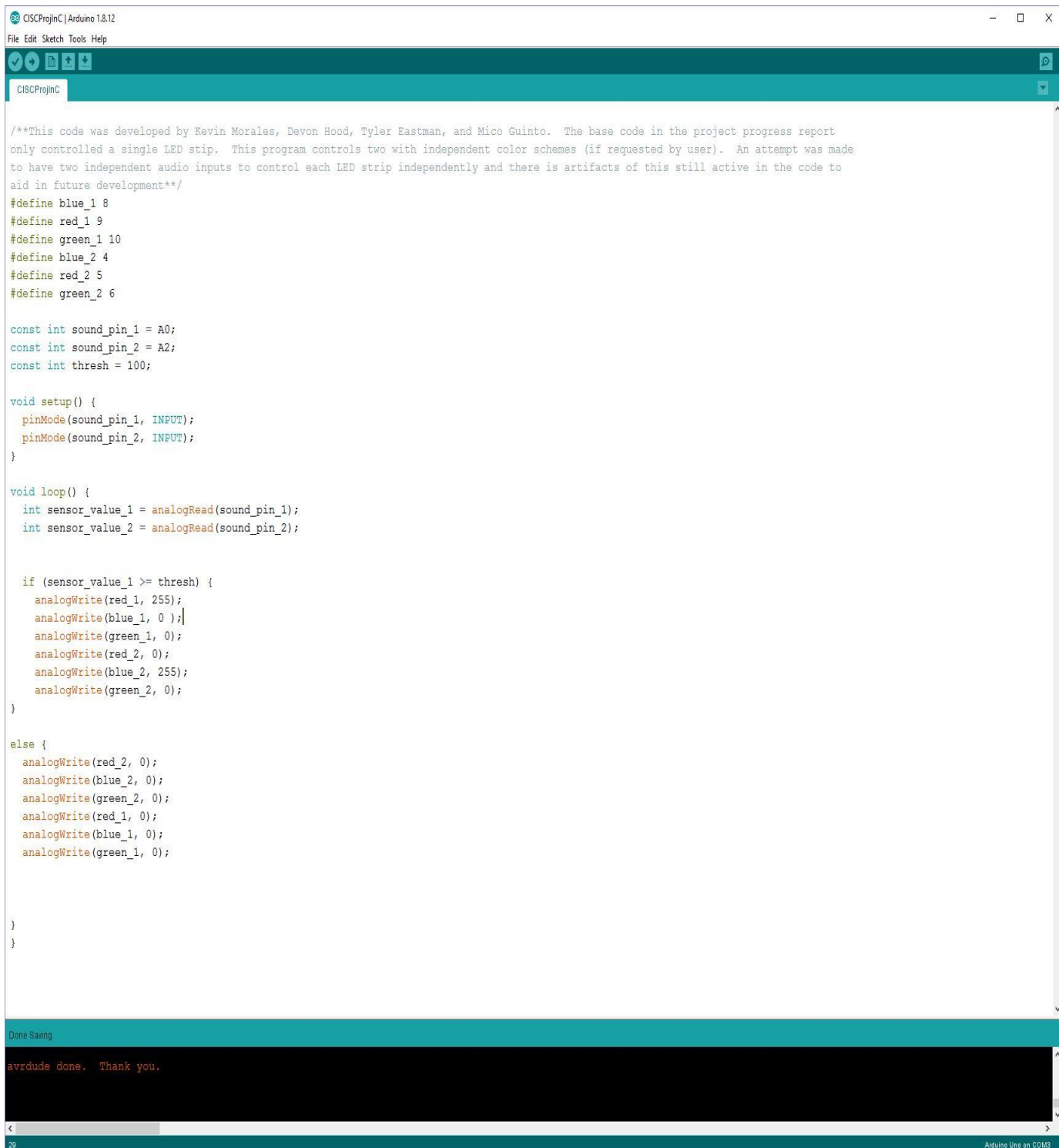
```
34 ;
35
36 ;in sensorval, ADCRA ; load sensor value into sensor register
37 ;cp threshreg, sensorval; compare treshhold value with sensor value
38 ;brge After ; if threshold is greater than sensor value, branch to after
39 ; otherwise fall through and blink led strip
40
41 ldi mask, (1<<PINB0) ;load port address into mask register
42 rcall Blink ;all Blink subroutine with rcall
43
44 ldi mask, (1<<PINB1) ;load port address into mask register
45 rcall Blink ;all Blink subroutine with rcall
46
47 ldi mask, (1<<PINB2) ;load port address into mask register
48 rcall Blink ;all Blink subroutine with rcall
49 After:
50
51 jmp start ; jump up to start label
52
53 Blink:
54 out DDRB, mask ;set pin addressed by mask register to output
55 eor ledr, mask ;toggle led register
56 out PORTB, ledr ;write led register to PORTB
57 ldi loopct, delaytime ;load delay subroutine argument into loopct
58
59 delay:
60 ldi iLoopRl,LOW(iVal) ; intialize inner loop count in inner
61 ldi iLoopRh,HIGH(iVal) ; loop high and low registers
62
```

```

63  iLoop:
64      sbiw    iLoopRl,1      ; decrement inner loop registers
65      brne    iLoop          ; branch to iLoop if iLoop registers != 0
66
67      dec     loopCt          ; decrement outer loop register
68      brne    delay          ; branch to oLoop if outer loop register != 0
69
70      nop                          ; no operation
71
72      ret                          ; return from subroutine

```

## C Code scaled/modified version:



The screenshot shows the Arduino IDE interface with a sketch titled "CISCProjnC". The code is written in C and is designed to control two LED strips based on audio input. The code includes a header comment, pin definitions, constant declarations, and two main functions: setup() and loop().

```
/**This code was developed by Kevin Morales, Devon Hood, Tyler Eastman, and Mico Guinto. The base code in the project progress report
only controlled a single LED strip. This program controls two with independent color schemes (if requested by user). An attempt was made
to have two independent audio inputs to control each LED strip independently and there is artifacts of this still active in the code to
aid in future development**/

#define blue_1 8
#define red_1 9
#define green_1 10
#define blue_2 4
#define red_2 5
#define green_2 6

const int sound_pin_1 = A0;
const int sound_pin_2 = A2;
const int thresh = 100;

void setup() {
  pinMode(sound_pin_1, INPUT);
  pinMode(sound_pin_2, INPUT);
}

void loop() {
  int sensor_value_1 = analogRead(sound_pin_1);
  int sensor_value_2 = analogRead(sound_pin_2);

  if (sensor_value_1 >= thresh) {
    analogWrite(red_1, 255);
    analogWrite(blue_1, 0);
    analogWrite(green_1, 0);
    analogWrite(red_2, 0);
    analogWrite(blue_2, 255);
    analogWrite(green_2, 0);
  }

  else {
    analogWrite(red_2, 0);
    analogWrite(blue_2, 0);
    analogWrite(green_2, 0);
    analogWrite(red_1, 0);
    analogWrite(blue_1, 0);
    analogWrite(green_1, 0);
  }
}
```

Below the code editor, the status bar shows "Done Saving." and the serial monitor displays the message "avrdude done. Thank you."

## References

We used this video to figure out what hardware we needed and how to set it all up. A quick glance at the code was all we needed to figure out how to use the arduino ide and its built in functions, the code was developed by us.

<https://www.youtube.com/watch?v=5M24QUVE0iU>

We used the avr instruction manual quite a bit.

<http://www.mmajunke.de/doc0856.pdf>

We used a website that had very good avr assembler tutorials and a couple of code examples. We would copy and paste the code examples, mess around with them to see how the instructions were used for I/O and then developed our final project around the code.

<http://www.rjhcoding.com/avr-asm-tutorials.php>

We used this image as reference for I/O.

ATmega328P pin mapping

Arduino function reset PC6 1 28 PC5 analog input 5

```
.include "m328pdef.inc"

.def    mask    = r16      ; mask register
.def    ledR    = r17      ; led register
.def    loopCt  = r18      ; delay loop count
.def    iLoopRl = r24      ; inner loop register low
.def    iLoopRh = r25      ; inner loop register high

.equ    iVal    = 39998    ; inner loop value

.cseg
.org    0x00
ldi     r16,LOW(RAMEND)    ; initialize
out     SPL,r16            ; stack pointer
ldi     r16,HIGH(RAMEND)   ; to RAMEND
out     SPH,r16            ; "

clr     ledR               ; clear led register
ldi     mask,(1<<PINB0)    ; load 00000001 into mask register
out     DDRB,mask          ; set PINB0 to output

start:  eor     ledR,mask    ; toggle PINB0 in led register
        out     PORTB,ledR  ; write led register to PORTB

        ldi     loopCt,50    ; initialize delay multiple for 0.5 sec
        rcall   delay10ms    ; call delay subroutine

        rjmp    start        ; jump back to start

delay10ms:
        ldi     iLoopRl,LOW(iVal) ; initialize inner loop count in inner
        ldi     iLoopRh,HIGH(iVal) ; loop high and low registers

iLoop:  sbiw    iLoopRl,1      ; decrement inner loop registers
        brne    iLoop         ; branch to iLoop if iLoop registers != 0

        dec     loopCt        ; decrement outer loop register
        brne    delay10ms     ; branch to oLoop if outer loop register != 0

        nop                     ; no operation

        ret                    ; return from subroutine
```

This code was a real huge help as it gave us hints as to how the output worked in avr. It is example 3, <http://www.rjhcoding.com/avr-asm-tutorials.php>