Service Learning Project Final Report
A Desktop LED Work Timer
ENGR 223 with Nick Macias

Linnea P. Castro

30 May 2023

Table of Contents

Service Learning Project Final Report

Introduction

For my Service Learning Project (SLP), I created a 25-minute battery powered work timer using an ELEGOO UNO R3 microcontroller and an LED grid. After the timer is initiated with a single press on the start button, the LED grid populates in appropriate time increments to build a lit heart image on the LED over the course of a 25-minute duration. The heart will flash on the grid until the user resets the timer, essentially clearing the grid and waiting for another push of the start button.

The main objective of my SLP was to build an LED based desktop work timer in order to fuse a physical product with programming elements. The motivation behind my project was twofold: I use a 25-minute timer several times on a daily basis to accomplish my academic work, and additionally, I felt a pull toward making and experimenting with physical hardware after taking ENGR 250.

Service Component Objectives

The service component of this project is that the desktop timer is designed to have a positive impact on a person's time management by giving them a doable way to dedicate time to practice something that they want to get good at, or accomplish work they need to get done. In my case it is something that either I would use to help me get my homework done or that I could imagine my oldest daughter using at her desk to create an engaging way for her to be more autonomous about her time management. I wanted to use a lighthearted graphical LED element like a heart to keep a spirit of fun and let the user see a visual reminder of what they have accomplished paired with motivation to see the image completed.

The concept of a 25-minute work timer is based on the "Pomodoro Technique," with a pomodoro being a 25-minute chunk of time which one dedicates to focused work[1]. The psychology of the method is brilliant and simple: 25-minutes of work is bite-sized and doable, gives me a win to build on, and encourages me to say "let's add one one more!" Using this simple timer method has given me a way to integrate one of my core values, practice, into my life in a tangible way. I value being able to show up and practice a skill, trusting that with time and repetition, my knowledge and creativity in that discipline will grow. Setting a short but substantial timer has helped me manage procrastination and given me a method to obtain a foothold on the projects and assignments I want to spend time practicing.

Dr. Gloria Mark, an informatics professor at U.C. Irvine has conducted studies on our changing attention spans, noting from conducting identical experiments over a period of 8 years, that the

[1] The Pomodoro Technique as created by Francesco Cirillo: https://francescocirillo.com/products/the-pomodoro-technique.

average time spent on a task before switching has declined over time[2]. I notice in myself that one of the most important factors to being able to have positive feelings associated with the work I accomplish is focus. When my focus is sharp, it builds momentum that helps me take further positive action. The service component of my project relates to creating a tool that can be used within a system to help build focus in individuals who use it. To be able to create an experience of focus is the motivation for my SLP.

When people feel like they are making progress toward their goals, doing the practice that they intend to do, it gives them a sense of self efficacy, which feels good. For me, accomplishing manageable wins makes me feel like I'm on the right path and reassures me that consistency in small doses is more important than a grand gesture - that 25 minutes of work is a step in the right direction, and something I can be proud of. This timer is designed to be a tool to give a person, whether adult or child, a sense of accomplishment and autonomy over their time management. The accomplishment of spending time doing things that are worthwhile to an individual can deflect procrastination and frustration caused by not spending time on things they know are aligned with their values. Thus, this timer is in service of time management, specifically, helping people spend time practicing the things they value and want to set aside time for.

Technical Objectives

My goal is to create a timer that is initiated by a start button that begins a 25-minute countdown. As the time counts down, rather than being represented by numbers counting down, as in a traditional timer, the timer will build the shape of a heart on an LED grid. The completion of the heart signifies the passing of the 25-minute time interval. When the time is complete, the heart will continue to flash on the LED grid until the user presses a reset button.

The technical objectives of my SLP include:

- Creating a functioning timer (prototype version) used for a set 25-minute time interval.
- Using a Start button to initiate the timer.
- Including a Reset button to reset mid-countdown or upon completion of a 25-minute interval.
- Integrating a battery power source for portability.
- Built with ELEGOO UNO R3 microcontroller.
- Integrate hardware components, like a push button, and MAX7219 driver to control the LED, and add nuance to the timer using C++ programming done in Arduino IDE and transferred over to ELEGOO UNO R3 hardware.
- Create code that compiles without errors.
- Understanding the basics of using the Serial Peripheral Interface to encode communications from MAX7219 driver to 8x8 LED grid.

[2] How to Focus Like it's 1990, Dana G. Smith, The New York Times, 09 January 2023. https://www.nytimes.com/2023/01/09/well/mind/concentration-focus-distraction.html

Learning Objectives

Through the creation of this timer, my objective was to become familiar with the ELEGOO UNO R3 microcontroller, wiring components, and using the Arduino IDE to develop, transfer, and test code on the UNO R3. I recognized in embarking on this project that I had very limited experience actually building something physical, and I chose this project to help me fill a perceived gap in my knowledge and build confidence as an engineer. This will help give me courage to work with hardware projects in the future.

To summarize the learning objectives of my SLP:

- Gain familiarity using C++ within Arduino IDE.
- Apply ENGR 250 concepts to wire circuits on the breadboard, implementing buttons, pull-up resistor, etc.
- Apply object oriented programming concepts to programming aspects of the timer.
- Transition the timer from USB powered to battery powered.
- Create an accurate schematic diagram which will allow me to take apart the project and rebuild it.
- Create an inventory of tools and source them.
- Test and integrate improvements along the way.
- Practice project management skills including meeting measurable milestones, documenting progress, and keeping a positive and curious attitude throughout. The process.

Initial Research and Alternate Solutions

The initial phases of my research revolved around creating an understanding of what tools I needed to use to build my timer. I wanted to be able to create a list of my materials and source them in as efficient a way as possible. I wanted to make sure I had enough backups to fall back on incase I had a malfunction or broke one of my components.

When deciding which materials to use for my project, I consulted YouTube and Google to get a general idea of which microcontrollers were most widely used and to understand what type of LED grids were available and compatible. I did not have any experience with breadboards or wiring components, so I originally thought that I would also need soldering materials. When I learned that there were solderless breadboards and accompanying components, I realized that this would be a great way to go for my first iteration of the project. Since I was building my project without soldering, it became important for me to find a MAX7219 component that was already soldered to the 8x8 LED grid.

I initially thought I would use the Arduino Uno Rev3 as a general microcontroller, but ended up going with the ELEGOO UNO R3, as it was lower in price and still compatible with the Arduino IDE for creating and uploading code to the microcontroller. I began sourcing components individually on Ebay, but then discovered that through a pre-assembled kit, I could obtain components like push buttons, LEDs, wires, 9 volt battery, a breadboard, and more all together, simplifying the procurement process. More detailed information on components and sourcing information can be found in the following sections of this report.

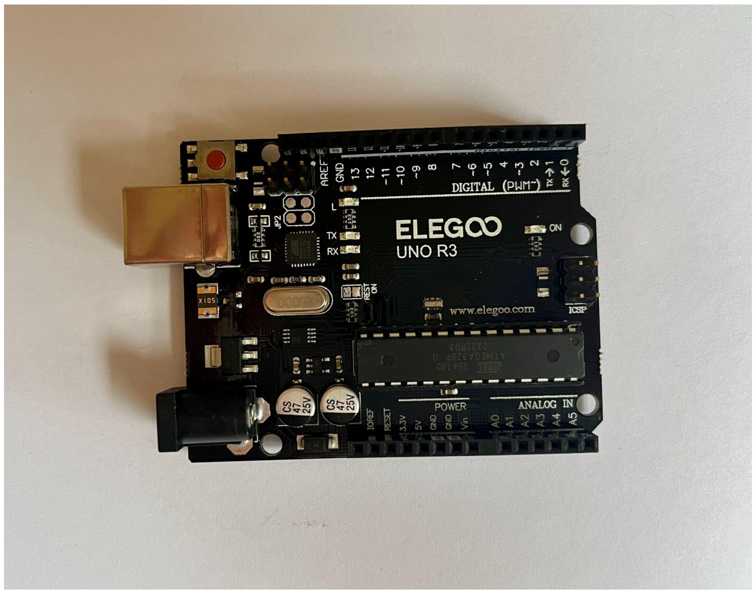
Parts List, Sourcing Information, and Pricing Details

- 1 830 Tie-Points breadboard (solderless)
- 1 ELEGOO UNO R3 microcontroller
- 1 9V battery and snap-on connector with plug-in
- 1 USB cable
- 1 1kΩ resistor
- 9 breadboard jumper wires
- 2 push buttons
- 5 female-to-male dupont wires
- 1 MAX7219 DotMatrix Module with 8x8 LED grid
- 2 individual LED lights (not necessary for final product, but useful for testing)
- Arduino IDE software

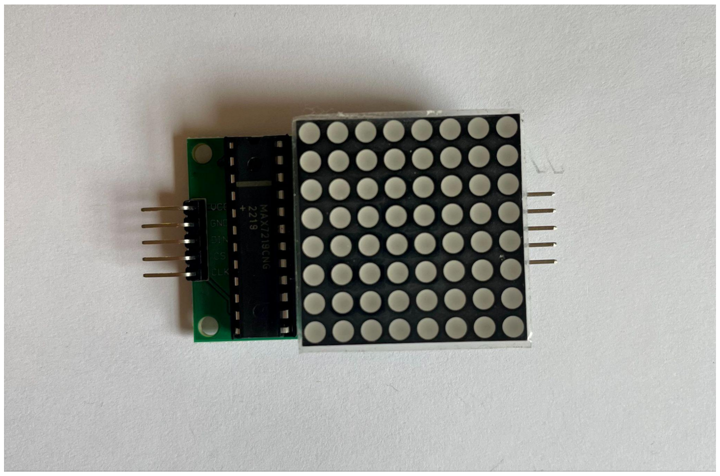
The Arduino IDE software was obtained via https://www.arduino.cc/en/software for no cost. All physical components were obtained in "The Most Complete Starter Kit UNO R3 Project" kit bought through ELEGOO's Amazon storefront for $52.07. For more detailed information on resources used, please see the Resources section at the very end of this document.

Photographs of Key Components

- ELEGOO UNO R3 microcontroller:

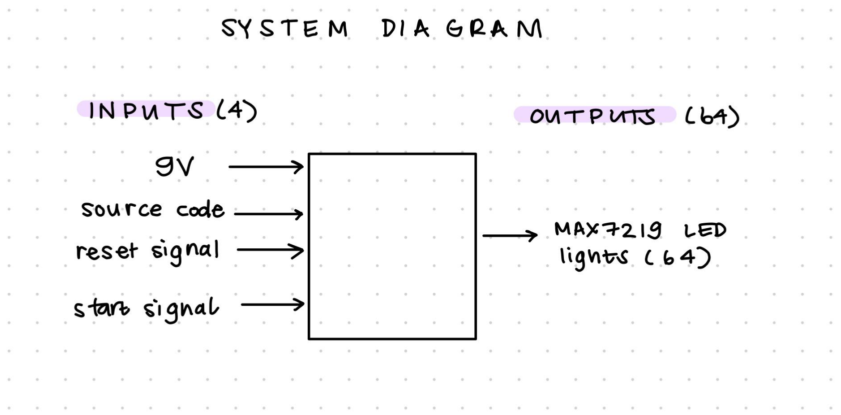


- MAX7219 DotMatrix Module with 8x8 LED grid:



- 9V battery and snap-on connector with plug-in:

System Diagram and High Level Design



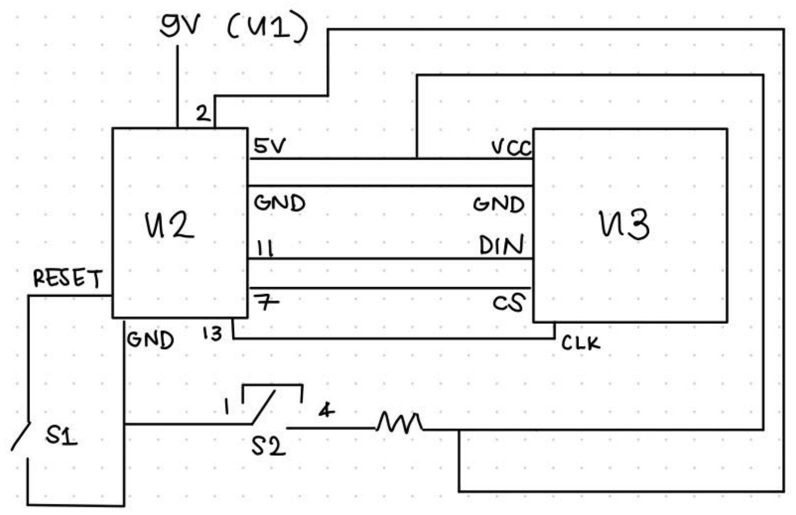
I simplified the inputs and outputs of my timer in the system diagram above. The main inputs being power from the 9 volt battery, source code sent in through USB via Arduino IDE, a reset signal, and a start signal. The ultimate output would be the signals sent through the MAX7219 ultimately turning on/off the lights on the 8x8 LED grid.

From a high level perspective, functional goals took precedence over aesthetic goals, however, I did want the timer to be somewhat portable and I did want it to have a start and reset button located on the breadboard. As soon as I had the timer functioning under power directly from my computer through a USB cable, I swapped the USB cord for a 9 volt battery and snap-on power adapter. Even though the arduino has a built-in reset button, I decided to install a reset push button to the right of the start button to mirror the implementation of the start button.

One challenge which I tackled early on in the project was how I would implement the start button. I knew that I didn't just want the timer to start as soon as it was connected to power, even though after setup, the microcontroller would enter into its main loop. This meant that I would need to have a way to signal to the arduino circuit that the user had pressed the start button.

I used a push button connected to a pull-up resistor to implement the start button. This is my best example in the project of using hardware and code together to solve a problem. The push button, creating a low-voltage state when pressed, signaled a flag value to change inside the main loop, which initiated the timer count down and the LED heart to start being built. See the following page for a full schematic, followed by all source code.

Schematic Diagram



COMPONENT ID BLOCK

NAME	TYPE	GND	VCC	N C
U1	9V battery	—	—	—
U2	Elegoo UNO R3	GND	battery plug	IOREF, 3.3V, Vin, AØ - A5, SCL, SDA, AREF, 12, ~1Ø, ~9, 8, ~6, ~5, 4, ~3, 1←TX, Ø→RX
U3	MAX7219	GND	VCC	—
S1	pushbutton (reset)	1, 4 to reset		2, 3
S2	push button (start)	1	4	2, 3

SERVICE LEARNING PROJECT
TIMER DIAGRAM
Linnea P. Castro
Ø3 JUNE 2Ø23

Source Code

```
// C++ code
//

#include <SPI.h> // Include library to use SPI (Serial Peripheral Interface) mode to
interact with MAX7219

#define Max7219_pinCLK 13 // Clock pin - synchronizes data transmission
#define Max7219_pinCS 7 // Chip Select pin.  Controller uses this pin to communicate
with LED grid, especially in Write_Max7219 method.
                       // Data is loaded while CS is LOW and latched on rising edge
(when CS goes high).
#define Max7219_pinDIN 11 // Data In pin.  Data is loaded into a 16-bit shift register
on the clock's rising edge.

int button; // Button variable, used in Start button

// WRITE TO 7219 METHOD  - This method was taken directly from the MAx 7219 Datasheet
provided with Elegoo parts.[3]
// Using digitalWrite to specify what pin will receive data and on what edge, together
with SPI keyword, transfer.
// This method uses the fact that the Max 7219's CS pin (7), loads data on LOW and
latches it in on the rising edge, when
// CS goes high.  This accommodates data transfer to the LED matrix and is the main
method involved in telling the matrix exactly which
// lights will be on, by specific address (row number), and value (0 means OFF and 1
means ON).
void Write_Max7219(uint8_t address, uint8_t value){ // uint8_t is data type unsigned
int 8 bit length
 digitalWrite(Max7219_pinCS, LOW); // Low to receive data
 SPI.transfer(address); // Send address 1-8, row of LED matrix
 SPI.transfer(value); // Send value for row to display B######## with # being either a
0/OFF 1/ON for respective row's position
 digitalWrite(Max7219_pinCS, HIGH); // Finish data transfer
}

// INITIALIZE 7219 METHOD - This method was taken directly from the Max 7219 Datasheet
provided with Elegoo parts.[4]
```

[3] Lesson 15 MAX 7219 8x8 Matrix Display Module, page 6. ELEGOO UNO R3.
[4] Lesson 15 MAX 7219 8x8 Matrix Display Module, page 6. ELEGOO UNO R3.

```
// The 7219 is in sleep mode upon start up, so this initialization method is needed to
reset registers prior to use.
void Init_Max7219(void){
 Write_Max7219(0x09, 0x00); // Decoding BCD, No-Op register
 Write_Max7219(0x0a, 0x03); // Brightness/Intensity register
 Write_Max7219(0x0b, 0x07); // Scan limit
 Write_Max7219(0x0c, 0x03); // Power down/shutdown mode
 Write_Max7219(0x0f, 0x00); // Display test
}

// SETUP METHOD - This method designates pins as either OUTPUT or INPUT.  The 7219's
Clock, CS, and Data In pins are all considered outputs
// from the Arduino and will become inputs to the LED matrix.  Pin 2, reserved for the
Start push button is a data input.  This method calls
// the SPI.begin() method and after a second's delay, runs the Init_Max7219 method to
set registers prior to using the timer.
void setup(){
 pinMode(Max7219_pinCLK, OUTPUT); // Clock, CS, and DIN pins on MAX7219 all designated
as outputs
 pinMode(Max7219_pinCS, OUTPUT);
 pinMode(Max7219_pinDIN, OUTPUT);
 pinMode(2, INPUT); // For Start button
 SPI.begin(); // Begin SPI mode
 delay(1000);
 Init_Max7219(); // Run this method to initialize 7219
}

// RESET LED GRID METHOD - This method is used right at the beginning of the main
loop, to ensure that all LEDs are starting from an OFF
// state.  It is also used when the timer is complete, to flash on and off.
void reset(){
 Write_Max7219(1, B00000000); // Reset LED grid
 Write_Max7219(2, B00000000);
 Write_Max7219(3, B00000000);
 Write_Max7219(4, B00000000);
 Write_Max7219(5, B00000000);
 Write_Max7219(6, B00000000);
 Write_Max7219(7, B00000000);
 Write_Max7219(8, B00000000);
}
```

```
// DISPLAY FULL HEART METHOD - This method is used at the completion of the timer
interval, when the heart flashes on and off.
// This method simplifies the displaying of the full heart image.
void fullHeart(){
 Write_Max7219(1, B01101110); // Flash full heart
 Write_Max7219(2, B11101111);
 Write_Max7219(3, B11111111);
 Write_Max7219(4, B11111111);
 Write_Max7219(5, B01111110);
 Write_Max7219(6, B01111110);
 Write_Max7219(7, B00111100);
 Write_Max7219(8, B00011000);
}

// ENTER TIMER LOOP WHEN START BUTTON IS PRESSED
void loop(){
  reset(); // Call reset method to start with empty grid

  button = digitalRead(2); // Button state (High/LOW) is determined by pin 2
  int flag; // Create a flag and initialize it to 0
  flag = 0;

  if (button == LOW){ // If button gets pressed, it goes to LOW
    flag = 1; // Set flag to 1 when button is pressed
  }

  while(flag == 1){ // Start button has been pressed, flag=1, when flag == 1, start
the timer and do the following:

  Write_Max7219(1, B01000000); // TOP ROW, ROW 1
  delay(33333);
  Write_Max7219(1, B01100000);
  delay(33333);
  Write_Max7219(1, B01101000);
  delay(33333);
  Write_Max7219(1, B01101100);
  delay(33333);
  Write_Max7219(1, B01101110);
  delay(33333);

  Write_Max7219(1, B01101110); // ROW 1
  Write_Max7219(2, B10000000); // ROW 2, beginning to build ROW 2
```

```
delay(33333);
Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11000000); // ROW 2
delay(33333);
Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11100000); // ROW 2
delay(33333);
Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101000); // ROW 2
delay(33333);
Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101100); // ROW 2
delay(33333);
Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101110); // ROW 2
delay(33333);
Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2 complete
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B10000000); // ROW 3, beginning to build ROW 3
delay(33333);
Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11000000); // ROW 3
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11100000); // ROW 3
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11110000); // ROW 3
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111000); // ROW 3
```

```
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111100); // ROW 3
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111110); // ROW 3
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3 complete
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B10000000); // ROW 4, beginning to build ROW 4
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11000000); // ROW 4
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11100000); // ROW 4
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11110000); // ROW 4
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
```

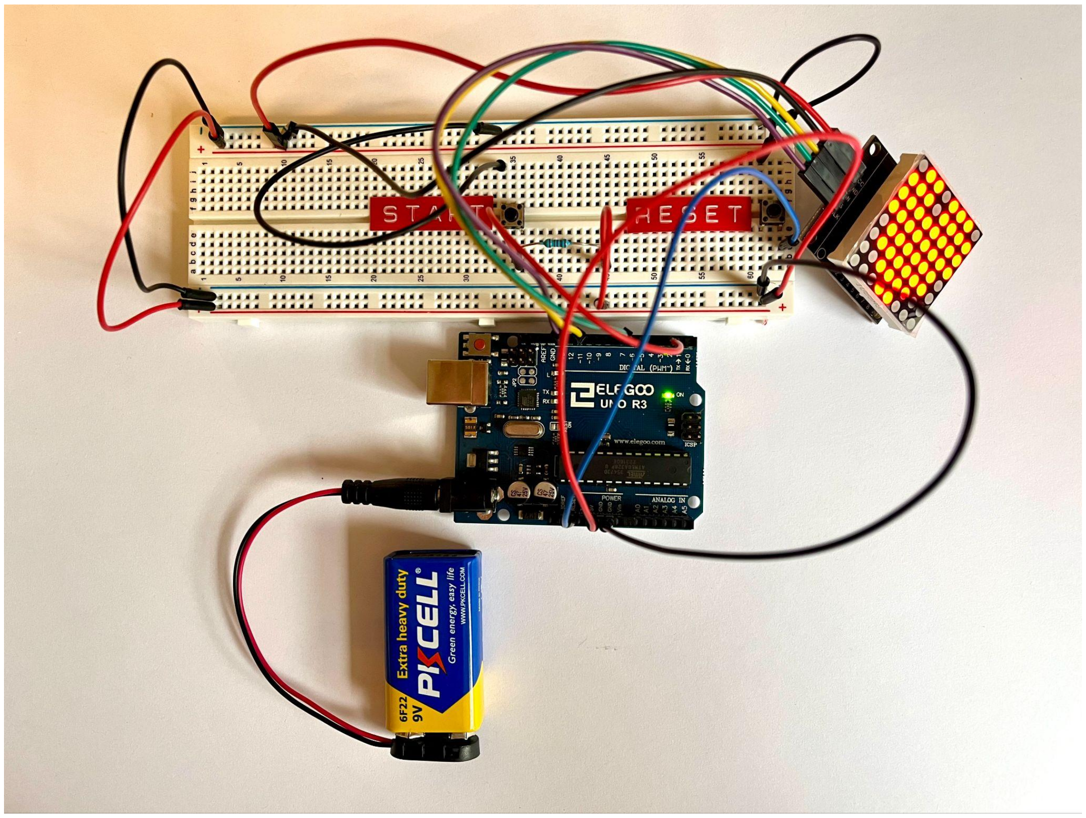
```
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111000); // ROW 4
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111100); // ROW 4
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111110); // ROW 4
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4 complete
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01000000); // ROW 5, beginning ROW 5
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01100000); // ROW 5
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01110000); // ROW 5
```

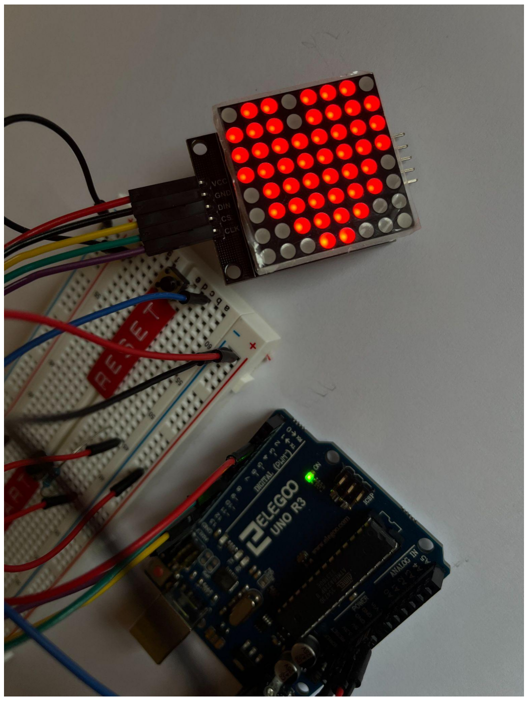
```
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111000); // ROW 5
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111100); // ROW 5
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111110); // ROW 5 complete
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111110); // ROW 5
Write_Max7219(6, B01000000); // ROW 6, beginning ROW 6
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111110); // ROW 5
Write_Max7219(6, B01100000); // ROW 6
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
```

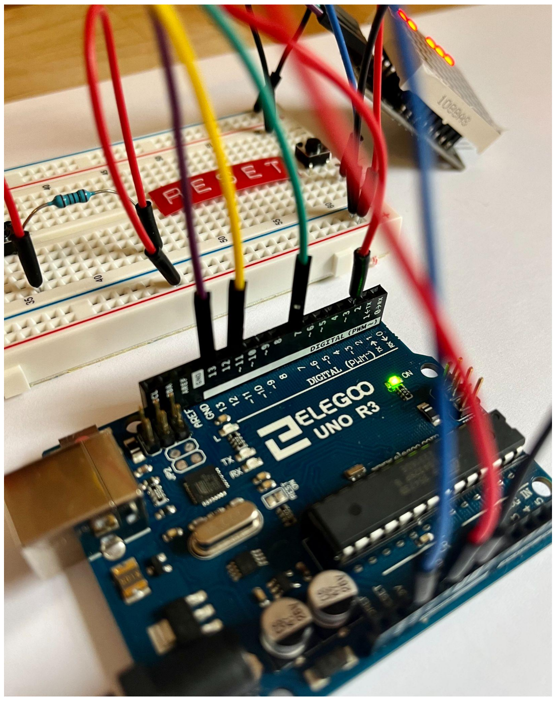
```
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111110); // ROW 5
Write_Max7219(6, B01110000); // ROW 6
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111110); // ROW 5
Write_Max7219(6, B01111000); // ROW 6
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111110); // ROW 5
Write_Max7219(6, B01111100); // ROW 6
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111110); // ROW 5
Write_Max7219(6, B01111110); // ROW 6 complete
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
Write_Max7219(5, B01111110); // ROW 5
Write_Max7219(6, B01111110); // ROW 6
Write_Max7219(7, B00100000); // ROW 7, beginning ROW 7
delay(33333);

Write_Max7219(1, B01101110); // ROW 1
Write_Max7219(2, B11101111); // ROW 2
Write_Max7219(3, B11111111); // ROW 3
Write_Max7219(4, B11111111); // ROW 4
```

```
  Write_Max7219(5, B01111110); // ROW 5
  Write_Max7219(6, B01111110); // ROW 6
  Write_Max7219(7, B00110000); // ROW 7
  delay(33333);

  Write_Max7219(1, B01101110); // ROW 1
  Write_Max7219(2, B11101111); // ROW 2
  Write_Max7219(3, B11111111); // ROW 3
  Write_Max7219(4, B11111111); // ROW 4
  Write_Max7219(5, B01111110); // ROW 5
  Write_Max7219(6, B01111110); // ROW 6
  Write_Max7219(7, B00111000); // ROW 7
  delay(33333);

  Write_Max7219(1, B01101110); // ROW 1
  Write_Max7219(2, B11101111); // ROW 2
  Write_Max7219(3, B11111111); // ROW 3
  Write_Max7219(4, B11111111); // ROW 4
  Write_Max7219(5, B01111110); // ROW 5
  Write_Max7219(6, B01111110); // ROW 6
  Write_Max7219(7, B00111100); // ROW 7 complete
  delay(33333);

  Write_Max7219(1, B01101110); // ROW 1
  Write_Max7219(2, B11101111); // ROW 2
  Write_Max7219(3, B11111111); // ROW 3
  Write_Max7219(4, B11111111); // ROW 4
  Write_Max7219(5, B01111110); // ROW 5
  Write_Max7219(6, B01111110); // ROW 6
  Write_Max7219(7, B00111100); // ROW 7
  Write_Max7219(8, B00010000); // ROW 8, beginning ROW 8
  delay(33348); // Last light is a few milliseconds longer than previous delay
intervals, to get as close as possible to 25 minutes

  Write_Max7219(1, B01101110); // ROW 1
  Write_Max7219(2, B11101111); // ROW 2
  Write_Max7219(3, B11111111); // ROW 3
  Write_Max7219(4, B11111111); // ROW 4
  Write_Max7219(5, B01111110); // ROW 5
  Write_Max7219(6, B01111110); // ROW 6
  Write_Max7219(7, B00111100); // ROW 7
  Write_Max7219(8, B00011000); // ROW 8 complete
  delay(1000); // Timer is already complete, this delay heads into flashing

  // TIMER COMPLETE, FLASH UNTIL RESET
  while(1==1){ // Infinite loop
    reset(); // Flash empty grid with reset method

    delay(1000); // 1 second delay

    fullHeart(); // Flash full heart with associated method

    delay(1000);
  }

}

} // End of source code
```

Photographs of the Finished Product







Test Plan

My first objective was to have the timer demo-ready for the SLP open house on the 6th of June. This meant having a portable, reliable, sped-up version of my timer that I could run several times over. To prepare for this, I adjusted my code, compiled, uploaded, and simulated my timer countless times. It was through testing and experiencing undesired or unexpected behaviors that I was able to make improvements to my timer. For example, I realized how important having and running a reset method before starting the timer was, otherwise the pattern displayed on the LED grid when I started it up was unpredictable, as I could tell that individual lights were holding on to their last programmed states.

I pared down the time delay between lights to half a second, making for a very speedy demo. After a successful SLP open house, I readjusted the delay interval to represent an interval of about 33333 milliseconds (with the last one being slightly longer), which over the course of all 46 heart-forming on lights, would count out a 25-minute period. My plan was to test the timer's function out while writing this report to time out my work intervals.

Test Results and Verification of Project Objectives

Referring back to my original Service Component, Technical, and Learning objectives, I would say that I did meet my benchmarks, with the caveat that there are quirks to my product and plenty of opportunities for future enhancements (to be discussed in the next section).

The biggest flaw in my timer was that occasionally I would experience unexplained inconsistent behavior in the timer. For example, sometimes it would work flawlessly all the way through a 25-minute interval, and other times, it would get stuck in the middle of row 2 and freeze and I would need to reset. I could not pin the problems down to a specific bug in my code, and I noticed that the timer worked more consistently with smaller time delays (half a second) than with the longer ones (as implemented in the full timer).

In terms of learning objectives, I feel capable of diving into other projects with the UNO R3, and definitely feel like I met my learning objectives. I still feel like there is much to learn about the microcontroller and its compatibility with the MAX7219 grid and the use of the SPI interface. I am certain that there are gaps in my knowledge that when filled could help me build a more reliable timer in all instances.

That being said, in terms of my service and learning oriented goals, the timer was very fun for me to use and did help me focus while writing this report - success! Now that I have this experience of building the timer, changing small pieces of it, and testing it over and over again, I feel like I am slightly less precious about the process of getting into it and building something. When I started the project, I was spending a lot of time in Tinkercad, almost hesitant to actually start working with the ELEGOO UNO R3. I would still recommend time spent experimenting in Tinkercad, however, I do feel like I have gained some confidence working with the Arduino IDE and microcontroller and am excited to try out future projects with it.

Future Enhancements

My timer could benefit from some consolidation and securing and containment of wires and components. This would involve using a smaller microcontroller and soldering components in place to make them less apt to move about or become disconnected. Despite the battery making my project more portable, the solderless breadboard construction was still finicky if wires happened to be jiggled during operation. I am fine with my timer looking like a prototype, but it would be great for it to be a more durable version of what it currently is.

Improvements could also be made to remove the unpredictable behavior mentioned in the previous section, through more learning and application of that learning leading to better coding implementation.

Recommended Series of Experiments

These are the “experiments” I conducted in the progression of building up to my final timer. My approach to creating these experiments was to fuse something I knew with something I didn’t yet know, thereby creating a way for me to move forward and make progress toward my project goals. For a student embarking on building this timer, I would recommend completing, in this order, the following series of experiments in Tinkercad:

- Connecting an Arduino to power and ground.

- Wiring a single LED and resistor to an Arduino, with the anode of the LED connected to one of the Arduino's numbered pins. Experiment with making the LED flash.
- Repeat the above experiment with 2 LEDs and make the LEDs flash in sync and then alternating.
- Build a circuit with a push button hooked up to a pull up resistor. When the push button is held down and receiving low voltage, make the LED flash. When the button is released, the LED stops blinking.
- Implement the same experiment above, however, use the button push to set a flag that keeps the light flashing (so the push button won't need to be held down). This concept is what will be used in the timer to start the timer, without a push button that stays in the down position.
- Implement the above experiment, using 3 LEDs in a row, change the code so that each light turns on one at a time, in a loop.
- Implement the above circuit, but this time adding in another push button to serve as the reset button, tied into the reset pin on the Arduino.
- Build the above circuit with your physical components, copying the original code from Tinkercad into the Arduino IDE and go from there!

Lessons Learned

In the course of this project, I have had the opportunity to practice new skills and ultimately build a desktop timer and bring my concept to life. This project was a great first step in building something using an ELEGOO UNO R3, with materials found inside a kit. This project was accessible and attainable, by the end of it, I felt confident in how to put something together that worked. At the same time, I also realize how much deeper I could go with this, and how much there is still to learn.

On a technical level, one of the most important concepts I grasped was the importance of giving each LED a state via the reset method if I didn't want unpredictable behavior on the grid when the power was on. My favorite problem that I solved was using the push button and a flag to workaround not having a push button that stayed in the down state. The biggest challenge I overcame was when I transitioned from using individual LEDs to using the MAX7219. Figuring out which pins to use in SPI mode and getting the syntax down correctly was a process that I had to continuously test and figure out.

My favorite learning concept that I used throughout the project was the premise of building upon something simple that worked, adding to it in small ways, and verifying that it was still working. This took me from a simple sketch in Tinkercad of the Arduino wired to vcc and ground to the timer that sits on my desk at this moment.

Licensing Terms

copy, modify, merge, publish, distribute, sublicense, build and/or sell. The Design is provided "as is" without warranty of any kind, express, or implied."[5]

Connections and Conclusion

I chose this project because of my interest in *simple* time management strategies and desire to have the experience of working with a microcontroller and building a physical project. I know that this is the first of more future microcontroller projects I will complete. The results of my testing let me know, however, that there are more steps I can take to create a version 2.0 prototype of my timer. I am grateful for the learning experience and fun I had working on this project.

Resources Used

- MAX7219/MAX7221 Serially Interfaced, 8-Digit LED Display Drivers Data Sheet. Maxim Integrated. https://www.analog.com/media/en/technical-documentation/data-sheets/max7219-max7221.pdf

- Lesson 15 MAX 7219 8x8 Matrix Display Module. ELEGOO UNO R3. https://www.elegoo.com/blogs/arduino-projects/elegoo-uno-r3-project-the-most-complete-starter-kit-tutorial

- Arduino & Serial Peripheral Interface (SPI) documentation. Arduino. https://docs.arduino.cc/learn/communication/spi

- Controlling 8x8 Dot Matrix with Max9219 and Arduino. Arduino Project Hub, by MariosIdeas. https://projecthub.arduino.cc/mdraber/controlling-8x8-dot-matrix-with-max7219-and-arduino-0c417a

- Tinkercad for creating initial sketches and simplified project versions. https://www.tinkercad.com/

- Arduino IDE for writing, compiling, uploading, and testing code with ELEGOO UNO R3 microcontroller. https://www.arduino.cc/en/software

- ELEGOO's Amazon storefront for "The Most Complete Starter Kit UNO R3 Project" and backup UNO R3. https://www.amazon.com/stores/ELEGOO/Homepage/page/E0F05684-D7AD-47CF-B08C-4084EBEE5BD3

[5] CSE 223 Project Report Documentation, page 1.

- ALAMSCN's Amazon storefront for backup MAX7219 Dot Matrix Module 8x8 LED Display Modules (LED grid already soldered to MAX7219, with Dupont wires)
https://www.amazon.com/s?k=ALAMSCN

- The Pomodoro Technique as created by Francesco Cirillo.
https://francescocirillo.com/products/the-pomodoro-technique

- How to Focus Like it's 1990, Dana G. Smith, The New York Times, 09 January 2023.
https://www.nytimes.com/2023/01/09/well/mind/concentration-focus-distraction.html