Pomodoro Timer Developer Guide

Table of Contents:

System Overview:

This timer can detect an object such as a phone by use of a sonic ranger, and sound a 440Hz alarm via an attached speaker if the object was removed from sight. The timer has a 25-minute and 5-minute setting, as well as an automatic setting that switches between the two, and is accurate to less than one second of error per minute. The LED display has three brightness settings. The timer and its components run on an Arduino Uno. The timer can be powered via USB or 9V barrel jack, and power can be cut off from the device by unplugging it.

Electrical Specifications:

Recommended Supply Voltage: 7-12V
Maximum Supply Voltage: 20V
Operating Temperature: -55 to 125 °Celsius
Operating Current: 500mA
Operating Voltage: 6V

User Guide:

First, select the timer setting using the timer select switch: either 25 minutes, 5 minutes, or the automatic timer which will switch between 25 and 5 minutes once each timer has counted down to zero. Once this setting is selected, place an object in front of the sonic ranger that will stay there while the timer runs. Plug the timer into a power source via either USB cable or the 9V barrel jack. The LED display should show the time remaining based on the chosen timer mode. The brightness of this display can be changed at any time by using the brightness select switch.

If the object is removed from the view of the sonic ranger, an alarm will sound until the object is placed back into view. The timer will reset if this occurs. In automatic mode, the sonic ranger does not check for an object during the 5-minute timed sections. This allows the user to use the object during these 5 minutes, and place it back in the view of the sonic ranger once these 5 minutes have passed and the timer is ready to change to a 25-minute section.

Once the timer reaches zero, a sound will play informing the user that they have reached the end of the specified time. In either of the non-automatic modes, the user will have to disconnect power from the timer in order to choose a new timer setting or restart the timer, and reconnect it once this selection has been made. In automatic mode, the timer will switch from 25 minutes to 5 minutes (and vice versa) after this sound is played. The user can disconnect power from the timer whenever they are finished using it to deactivate the device.

Design Artifact Figures:

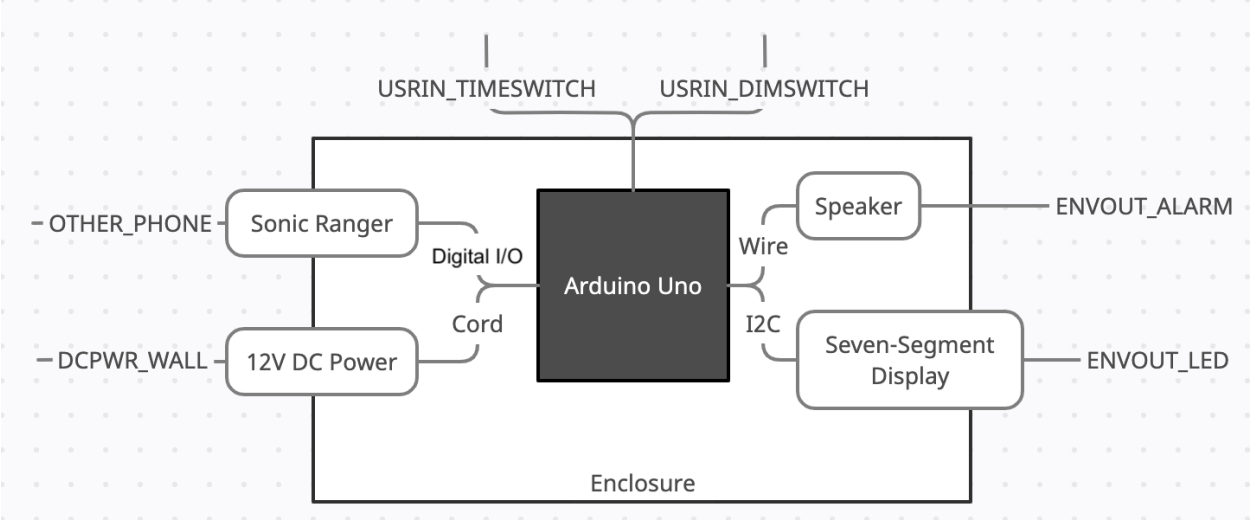

Figure 1: Block Diagram of Timer Design.

This block diagram shows the individual components of the design, and how they interact with the Arduino Uno and its processor. The LED and Alarm are used to output information like remaining time and lack of object detection to the user. The two switches on the top are used to control the timer's settings. The user also provides power through the DC barrel jack, and gives the sonic ranger an object to detect so that the timer can count down.

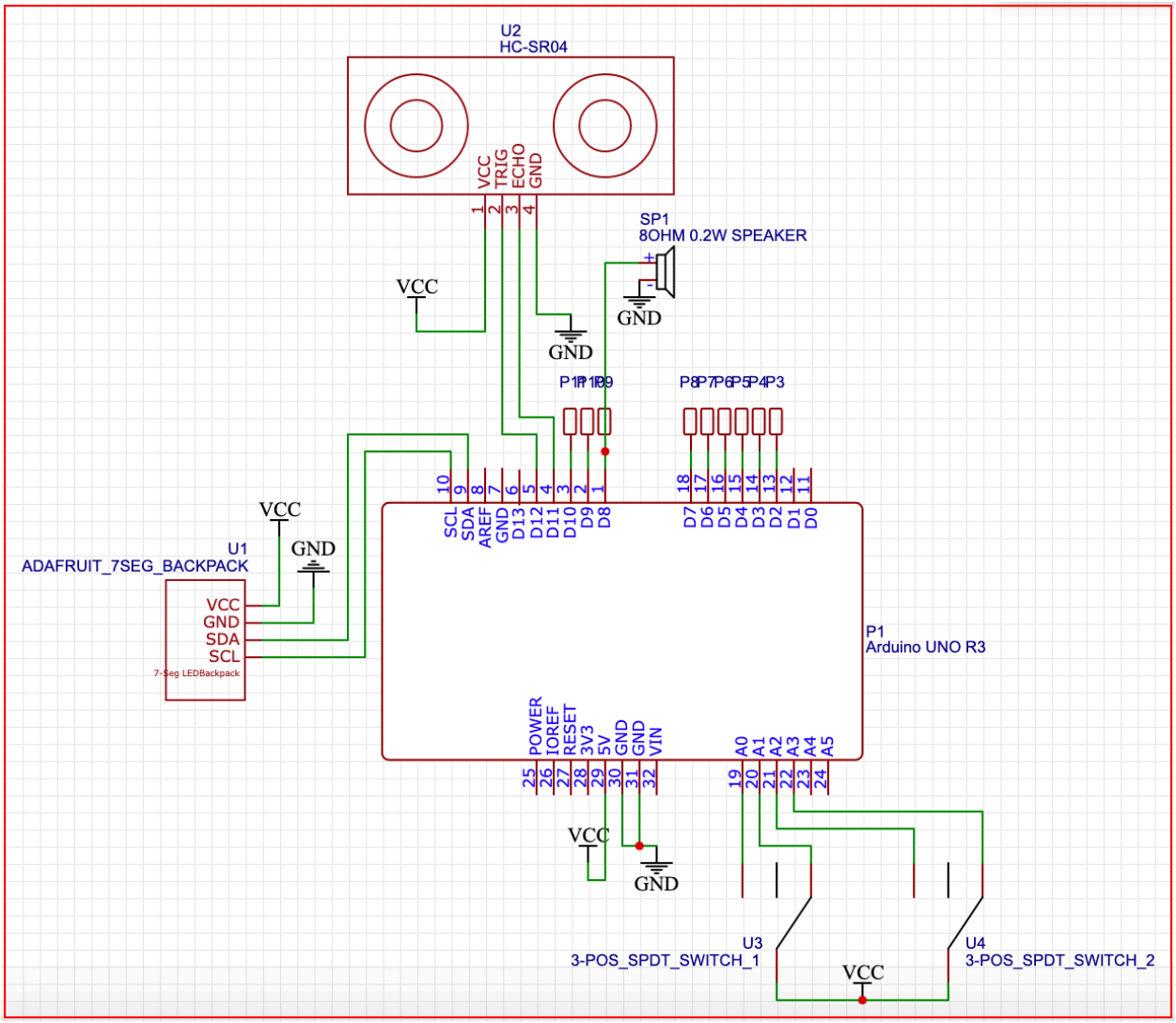

Figure 2: Wiring schematic for Timer.

This diagram shows how the different components are wired to the Arduino so that the Arduino's code will run the timer properly. The two switches are hooked up to Analog inputs on the Arduino, and the values they output when switched on or off are used to select the proper brightness and timer values. The Adafruit 7-segment I2C backpack allows us to run all of the many pins of a traditional 7-segment LED display on a mere four pins.

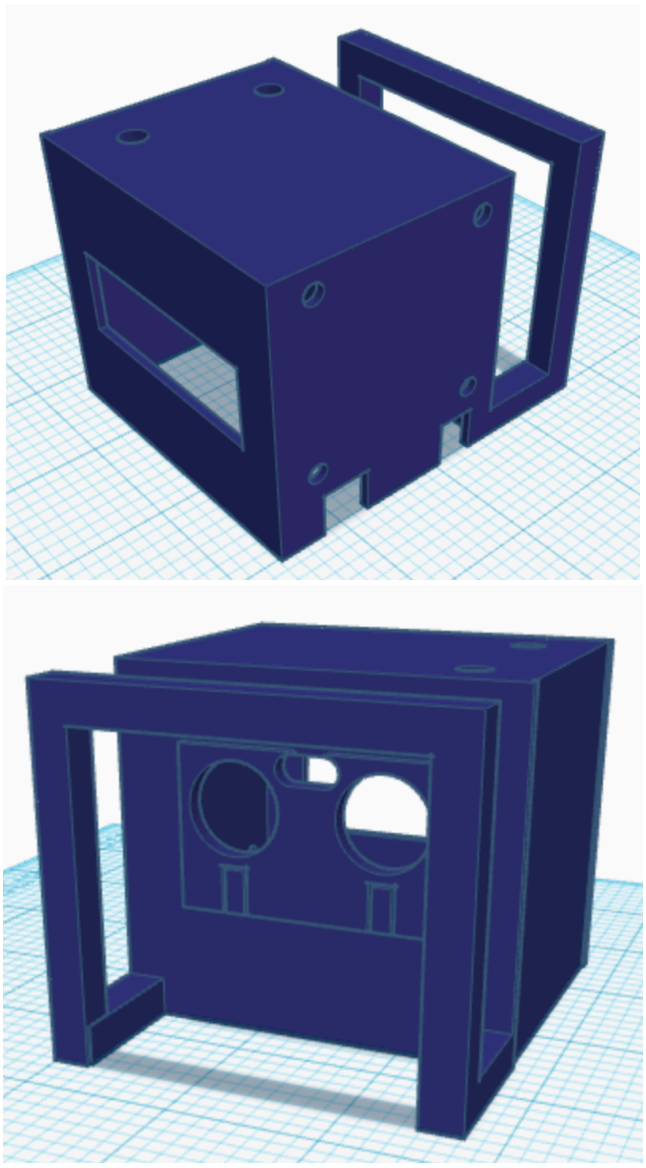

Figure 3: STL file of enclosure design.

The enclosure is designed so that all components can be secured via screws; the Arduino itself slides into the bottom of the enclosure and will be secured using a plastic base that was included with the Arduino Uno. This base will also screw into the enclosure. The arc in front of the sonic ranger is designed to avoid detection while also providing a space for an object to easily rest while the timer is in use. Two holes at the top of the enclosure are provided for the two switches, and the rectangular hole is designed for the LED display. The USB port and barrel jack have openings as well, allowing the device to be powered via USB or wall power.

PCB Information:

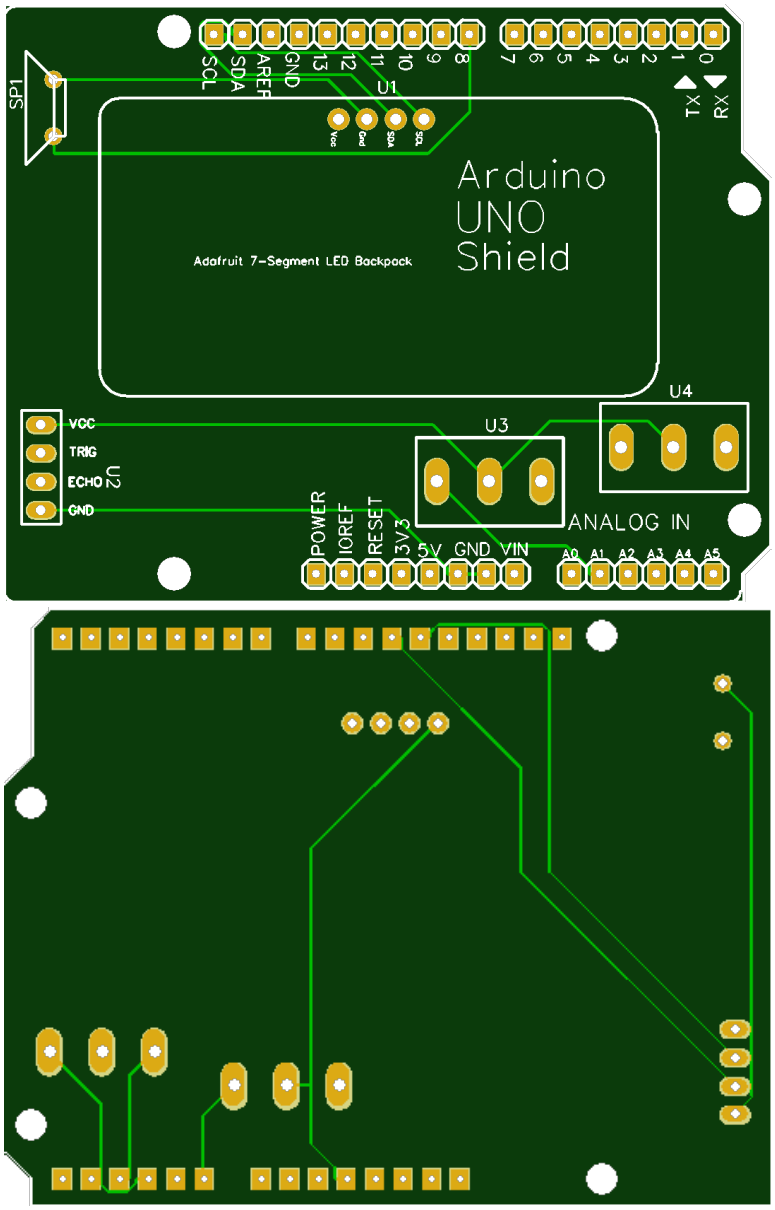

Figure 4: PCB schematic for Arduino shield.

This Arduino Uno shield PCB is 68.7mm wide and 53.5mm long. The traces in this PCB allow for all of the components to be easily hooked up to their respective locations on the shield, and the holes are placed so that screws can go through both the Arduino and the PCB to secure the entire circuit into the enclosure design.

Bill of Materials:

- 1 Arduino Uno Rev3
- 1 12V DC Power Supply
- 1 KW4-56NXUYA-P 4-digit 7-segment LED display
- 1 HT16K33 0.56” 7-segment LED I2C Backpack
- 1 Jameco 135694 8ohm 0.2W Speaker
- 1 HC-SR04 Sonic Ranger
- 2 MTS-103 3-position Toggle Switches

Arduino Code:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include "Adafruit_LEDBackpack.h"
#include "pitches.h"

const byte trigPin = 12;  // trigger pin for sonic ranger
const byte echoPin = 11;  // echo pin for sonic ranger
const byte buzzPin = 8;   // buzzer pin for alarm
bool readDistance = false; // activates sonic ranger

int highPin = A0;         // high brightness
int lowPin = A1;          // low brightness
int shortPin = A2; // 5 minute timer select
int longPin = A3;  // 25 minute timer select
bool autoTime = false;    // auto timer select

uint16_t counter; // timer variable for countdown
uint16_t COUNT; // initial value for timer
Adafruit_7segment matrix = Adafruit_7segment(); // seven-seg function package

int duration, distance; // sonic ranger variables

void victoryTune() {
  // melody code from Arduino IDE example
  // "toneMelody" by Tom Igoe

  // notes in the melody:
  int melody[] = {
    NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
  };

  // note durations: 4 = quarter note, 8 = eighth note, etc.:
  int noteDurations[] = {
    4, 8, 8, 4, 4, 4, 4, 4
  };

  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second divided by the note
type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(8, melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(8);
  }
}

void timerInit() {
```

```
  // set initial timer
  // when switch is active,
  // analogRead returns a value above 825

  if(analogRead(shortPin) >= 825) {
    COUNT = 2500; // 25-minute timer
  }
  else if(analogRead(longPin) >= 825) {
    COUNT = 500; // 5-minute timer
  }
  else {
    COUNT = 2500; // 25-minute timer
    autoTime = true; // auto timer activated
  }

  counter = COUNT; // initialize counter value
  readDistance = true; // initialize sonic ranger
}

void getDistance() {
  // Output pulse with 1ms width on trigPin
  digitalWrite(trigPin, HIGH);
  delay(1);
  digitalWrite(trigPin, LOW);
  // Measure the pulse input in echo pin
  duration = pulseIn(echoPin, HIGH);
  // Distance is half the duration divided by 29.1 (from HC-SR04 datasheet)
  distance = (duration/2) / 29.1;
}

void setup()
{
  pinMode(trigPin, OUTPUT); // outputs sonic pulse
  pinMode(echoPin, INPUT);  // measures pulse response
  pinMode(buzzPin, OUTPUT); // controls alarm
  pinMode(highPin, INPUT);  // high brightness input
  pinMode(lowPin, INPUT);   // low brightness input

  pinMode(shortPin, INPUT); // 5-minute timer select
  pinMode(longPin, INPUT); // 25-minute timer select

  matrix.begin(0x70); // initialize seven-seg
  timerInit();


}

void loop()
{
  matrix.println(counter);
  matrix.drawColon(true);
  matrix.writeDisplay();

  if(readDistance) {
    getDistance();
  }
  else {distance = 0;}
```

```
  while (distance >= 20) { // sound alarm if object is too far (>20 cm)
    tone(buzzPin, 440);
    delay(499); // beep for half a second
    counter = COUNT; // reset timer
    getDistance(); // check distance again
    noTone(buzzPin);  // stop beeping
    delay(500);       // for half a second
  }

  noTone(buzzPin);
  if (counter != 0){ // decrement counter
    counter--;
    if (counter % 100 == 99){ // skip from 99 to 59 when necessary
      counter -= 40;
    }
  }
  else {
    victoryTune(); // you did it!
    if(autoTime) { // swap timer from 25 minutes to 5 minutes
      if(COUNT == 2500) {
        readDistance = false; // stop yelling at the user
        COUNT = 500;         // for 5 minutes
        counter = 500;
      }
      else {
        readDistance = true;  // resume yelling at the user
        COUNT = 2500;        // for 25 minutes
        counter = 2500;
      }
    }
    else {while(true);}
  }

  // Detect brightness setting:
  // when switch is active,
  // analogRead returns a value above 850
  if(analogRead(highPin) >= 850) {
    matrix.setBrightness(15); // full brightness
  }
  else if(analogRead(lowPin) >= 850) {
    matrix.setBrightness(0); // low brightness
  }
  else {
    matrix.setBrightness(8); // mid brightness
  }
  delay(999); // wait for the rest of the second
}
```