Basic Simulations of Real-Life Appliances with Arduinos

Thomas Li

Department of Physics, Miami University, Oxford, OH

11/01/2018

Introduction

Microcontrollers are tiny chips made of integrated circuits that act as small computers. Arduino is a company whose microcontroller products are among the most popular brands across the globe, while those microcontrollers themselves are often referred to as Arduinos as well. Arduinos are widely used in electronic engineering for its variety of functions that are programmable in their own programming language called Arduino Integrated Development Environment (IDE). One can code and run IDE programs (sketches) on an Arduino connected to a circuit to make orders on certain electronics, while the Arduino itself is exceedingly fast in processing and memorizing data and also low powered enough that one can power it by a battery for many days.

This work commits to a few basic Arduino-based programs along with the actual instrumentations to display some simulations related to real world applications, for example, playing of music and traffic light regulation with pedestrian buttons, with codes attached.

Blinks of Light

One of the most basic functions an Arduino is used to perform is blinking a LED light.

In the coding page of the IDE, the very first function line is often written as *void setup () {*, following which is a *pinMode(#, OUTPUT)*, meaning the pin number on the Arduino to be initialized as an output, and a second half brace symbol *}* on a third line to complete this initialization part. The following part is often a loop function written as *void loop() { }*, within the braces of is the area where one designs any function to be performed on the Arduino, while as the name itself suggests, an loop functions will always run repeatedly unless the device is turned off. To perform any instrumentation one must have a control of both start and stop. In IDE, it is often written as digitalWrite(#, HIGH) and digitalWrite(#, LOW), the number in the parenthesis meaning the pin number, and the following arguments *HIGH* and LOW meaning the voltage level that serve to turn on or turn off the pin whatever the pin is connected to. A delay

function is also frequently used in any function designs; the default unit of a delay function is in millisecond, for example, delay(1000) literally means a delay of one second during which the Arduino does nothing.

For the first blink we set up a two-second LED-on followed by a half-second off, by taking advantage of the delay function as described above. The sentences on the right hand side of the double slash // are annotations on what each line is doing and affect nothing on the function itself. The sketch with detailed annotations is shown in Appendix 1 - Sketch 1 - Blink 1.

For the second blink we had two durations of LED blinking, a two-second on followed by a half-second off, next a five-second on followed by a three-second off, then back to the beginning - the two-second on and so on. The sketch with detailed annotations is shown in Appendix 1 - Sketch 1 - Blink 2.

For the third blink, we put more variations of durations. A one-second on with a two-second off, followed by a three-second on with a one-second off, then a one-second on with a two-second off, after which it goes back to the beginning of the loop. The sketch with detailed annotations is shown in Appendix 1 - Sketch 1 - Blink 3.

Music

Sound stems from vibrations, while a tone is nothing but a set of frequencies of defined amplitudes of vibrations. By adjusting the magnitude of the voltage input and the frequency of an alternating current to a proper vibrator, one should be able to display a sound with certain pitch and strength. When a set of current pattern is applied to the vibrator, the combination of the sounds forms a music.

Adjusting the frequency and spacing of each sound is considerably easy in Arduino IDE, while the amplitude of the sound may be adjusted on a breadboard connected to a speaker. By using a tone function that follows the form: *tone(#1, #2, #3)*, where the first number is the output pin number on the Arduino, and the second is the frequency of the desired sound in Hz, with the third being the duration of time in millisecond, the Arduino will send a square wave voltage carrying the information of a sound to a speaker, a combination of which could therefore play a music. The music, or the tone functions will last until it meets a notone function, written as *notone(#)* with the number in the parenthesis being the pin number.

We chose part of Wii Music as our instrumentation pattern. To convert the music into frequencies we followed the musical notes frequency conversion, and each note stays the time interval differently and we gave an approximation in both the duration and the delay. For example the first part of the music follows $B_4\ E^b_5\ G^b_5\ E^b_5\ B_4$, which is corresponding to 493Hz, 622Hz, 739Hz, 622Hz, and 493Hz, and we set the duration of playing to be 0.3s for all of them, spaced with a delay of 0.6s, 0.3s, 0.6s, 0.6s, and 0.3s in between respectively. [1] The sketch of the music is shown in Appendix 1 - Sketch 2.

Buttons

There are many cases in real world applications we cannot have a machine run fully by itself, and we want to give orders without the need to reconstruct any internal structures or circuits each time; therefore, a button to press which calls a default previously designed function will satisfy both conveniency and productivity.

To build such a button function we connected a LED to the circuit board with the Arduino, where we also connected a physical button with connection to the power source and to the Arduino so that when one presses it, the Arduino starts to take the order and lighten the LED. In the IDE, for the first loop we set pin numbers using *const int buttonpin = 2*, *const int ledPin = 13*, to set the pins 2 and 13 being button and LED outlets respectively, with the variable for reading the pushbutton status set as *int buttonState = 0*. Next, by using the *pinMode* function written as: *pinMode(buttonPin, INPUT)*, and *pinMode(ledPin, OUTPUT)*, we assign the button as the input and the LED as the output to finish the first setup loop.

To execute the function demanded by the button, the second loop must check the previous setup loop repeatedly. By taking advantage of the *buttonState* function that follows *buttonState = digitalRead(buttonPin)*, the Arduino read the pushbutton value to check if the button is pressed, if so, the state reads HIGH, and the program then goes to a if function, in this case, written as: *if (buttonState == HIGH)*; knowing that the button is pushed the Arduino should send the message to the output pin controlling the LED, and this follows as: *digitalWrite(ledPin, HIGH)*; when the button is pushed, still in the if function, we use a *else* function following which is the *digitalWrite(ledPin, LOW)*, to not to turn on the LED pin. [2]

The sketch with detailed annotations is shown in Appendix 1 - Sketch 3.

Traffic Light

A traffic light button is a good example with the use of a button function. On side of the pedestrian of a road intersection, there is usually a button on the post of the traffic light to press which the system will receive the message and re-regulate the waiting time of the passengers as well as that of the cars. In Arduino, to perform such a function, one must incorporate button functions, loop functions, if functions, and two setups of traffic light patterns that differ depending on whether the button is pushed.

Firstly a setup of LED and button with connection to the output and input pins is necessary. We connected the three colors of LEDs to the different output pins on the Arduino, and attached the button to the input pin by using the *pinMode* function. In this case the button is not used to initialize the lights but to change the pattern. For non-passenger pattern, we chose 40s for the green light (LED), 1~2 seconds for the yellow, followed by a 20 seconds for the red; for the passenger pattern, it waits for 15 seconds of the green light or the remaining time of the green light whichever is shorter, then the yellow one remains the same time, followed by the red light lighting for 30 seconds.

To achieve so, we used a if function that it checks the button status every 5 milliseconds in green, and if the button is not pushed, it repeats the checking until 40 seconds elapsed; however if the button is pushed, it goes to another *if* function that relies on a time counter denoted as *timeElapsed* which add 5 for each checking, and if the *timeElapsed* reads less than 40000-15000, it finishes the 40 second followed by the passenger pattern mentioned above, elsewise, it finishes a 15 seconds and then the passenger pattern of yellow and red lights. If previously in green the button was not pushed and it is in yellow at the moment, the Arduino will check for each 5ms as well, and give the corresponding order of the red pattern. Similarly, when it is in red, the *if* function will decides whether finish the 20s or 30s depending on the button status. The detailed steps with annotations is attached in Sketch 4.

Pulse Width Modulation (PWM)

PWM is widely used in circuitry and optics; generally it is used to adjust the width of a waveform in part of or the whole circuit. For a rectangle waveform, PWM is often related to the

term duty cycle. The duty cycle of this waveform signal is the active fraction of one period of the signal, with the rest being silent; An example sketch is shown in Appendix 1 - Sketch 5.

Fading

Fading often refers to a gradual decrease phenomenon in an observable quantity, for example, of the luminosity of an LED. The sketch with detailed annotations is shown in Appendix 1 - Sketch 6.

3-Intensity-LED

This program will light an LED to full brightness when a button is pressed once. When the button is pressed again, the LED will decrease in brightness to 2/3rds its original brightness. When the button is pressed again, the LED will decrease in brightness to 1/3rd its original brightness. When it's pressed again the LED will turn off. And then it will loop back around. The sketch with detailed annotations is shown in Appendix 1 - Sketch 7.

Summary

Arduinos are versatile and efficient microcontrollers; by coding its own programming language Arduino IDE one is able to upload various types of functions that have a significance in real life applications. For instance, a piece of music could be composed on an IDE by organizing the frequency of musical notes and the spacing of duration, and by connecting the chip with a speaker it is able to play it out. In addition, Arduinos are also convenient in many advanced circuitries; for instance, it could modulate a square wave cycle duty which is widely used in voltage regulation and digital information.

References

1. Frequencies for equal-tempered scale, URL, https://pages.mtu.edu/~suits/notefreqs.html.Accessed November 1, 2018.

2. Arduino Tutorial – Button, URL http://www.arduino.cc/en/Tutorial/Button. Accessed November 1, 2018.

Appendix - Sketches

Sketch 1: Blink Variations

Blink 1:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(2000);              // wait for two seconds
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(500);               // wait for half a second
}
```

Blink 2:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(2000);              // wait for two seconds
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(500);               // wait for half a second
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
```

```
  delay(5000);              // wait for five seconds
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(3000);              // wait for three seconds
}
```

Blink 3:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for one seconds
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(2000);              // wait for two seconds
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(3000);              // wait for three seconds
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for one seconds
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for one seconds
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(2000);              //wait for two seconds
}
```

Sketch 2: Tone

```
void setup() {
  pinMode(5, OUTPUT);
}
```

```
void loop() {
  tone(5, 493, 300);
  delay(600);
  tone(5, 622, 300);
  delay(300);
  tone(5, 739, 300);
  delay(600);
  tone(5, 622, 300);
  delay(600);
  tone(5, 493, 300);
  delay(300);

  tone(5, 329, 250);
  delay(300);
  tone(5, 329, 250);
  delay(300);
  tone(5, 329, 300);

  delay(1800);
  tone(5, 493, 300);
  delay(300);
  tone(5, 622, 300);
  delay(300);
  tone(5, 739, 300);
  delay(600);

  tone(5, 622, 300);
  delay(550);
  tone(5, 493, 300);
  delay(300);
```

```
 tone(5, 932, 900);
 delay(900);
 tone(5, 880, 300);
 delay(300);
 tone(5, 783, 300);
 delay(300);
 noTone(5);
}
```

Sketch 3: Button

```
/*
 Button

 Turns on and off a light emitting diode(LED) connected to digital
 pin 13, when pressing a pushbutton attached to pin 2.

 The circuit:
 * LED attached from pin 13 to ground
 * pushbutton attached to pin 2 from +5V
 * 10K resistor attached to pin 2 from ground

 * Note: on most Arduinos there is already an LED on the board
 attached to pin 13.


 created 2005
 by DojoDave <http://www.0j0.org>
 modified 30 Aug 2011
 by Tom Igoe
```

```
 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/Button
 */

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;          // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;     // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
          // turn LED on:
          digitalWrite(ledPin, HIGH);
  }
  else {
```

```
        // turn LED off:
        digitalWrite(ledPin, LOW);
  }
}
```

Sketch 4: Traffic Lights

```
/*

  This program simulates a traffic light. The regular cycle for the traffic light
  starts at green for 40s, goes to yellow for 1-2s, and then red for 20s. If a button
  is pushed then this cycle breaks. It waits for 15s or the remaining time whichever is
  shorter, goes to yellow for regular amount of time, then goes to red for 30s.

 */

// Declares pin numbers for the button and LEDs
// it also establishes the initial button state
const int buttonPin = 2;
const int redLED = 13;
const int yellowLED = 12;
const int greenLED = 11;
int buttonState = 0;
int timeElapsed = 0;
int count = 0;

void setup() {
  //the button's pin is declared as an input
  pinMode(buttonPin, INPUT);

  //the LEDs' pins are declared as outputs
  pinMode(redLED, OUTPUT);
```

```
  pinMode(yellowLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
}

void loop() {
  // The initial LED on is green
  digitalWrite(greenLED, HIGH);
  // it checks every 5 ms for 40s to see if the button
  // is pressed if so it'll wait 15s or the remaining
  //time whichever is shorter. And then breaks out of
  //the for loop
  for(int i =0; i < 8000; i++) {
    buttonState = digitalRead(buttonPin);
          if(buttonState == 0) {
          delay(5);
          timeElapsed +=5;
          } else {
          if(40000 - timeElapsed > 15000) {
          delay(15000);
          } else {
          delay(40000 - timeElapsed);
          }
          i = 8000;
          }
  }
  // the green light is turned off and timeElapsed is reset
  timeElapsed = 0;
  digitalWrite(greenLED, LOW);

  // the yellow light is turned on
  digitalWrite(yellowLED, HIGH);
```

```
// it checks every 5 ms for 2s to see if the button
// is pressed if so it'll change the time that the red
// light is on to 30s
if(buttonState == 0) {
	for(int i = 0; i < 400; i++) {
	buttonState = digitalRead(buttonPin);
	if(buttonState == 0) {
	delay(5);
	timeElapsed += 5;
	} else {
	delay(2000 - timeElapsed);
	i = 400;
	}
	}
} else {
	delay(2000);
}
//the yellow light is turned off and the timeElapsed is reset
timeElapsed = 0;
digitalWrite(yellowLED, LOW);

//The red light is turned on
digitalWrite(redLED, HIGH);
// the red light is on for either 20s or 30s depending upon
// whether or not the button has been pushed
if(buttonState == 0) {
	delay(20000);
} else {
	delay(30000);
}
//the red light is turned off and timeElapsed is reset
```

```
  timeElapsed = 0;
  digitalWrite(redLED, LOW);

 buttonState = 0;
}
```

Sketch 5: PWM

```
int dc, hi_time; // declare the 2 variables as global so that
                 // they will be recognized in all functions
void setup() {
  pinMode(13, OUTPUT);      // built-in LED used as output

  dc = 70;                  // duty cycle is 70%
  hi_time = dc * 10;     // duration of pulse in micro seconds
                 // based on 1 kHz clock frequency
}

void loop() {
 digitalWrite(13, HIGH);
 delayMicroseconds(hi_time);

 digitalWrite(13, LOW);
 delayMicroseconds(1000 - hi_time);
}
```

Sketch 6: Fade

```
/*
 Fade
```

```
 This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 This example code is in the public domain.
 */

int led = 9;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
        fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Sketch 7: 3-Intensity LED

```
// declaring the pins used by the LED and button
// as well as the starting brightness (0) and
// button state.
int ledPin = 9;
int buttonPin = 3;
int buttonState = 0;
int brightness = 0;
int i = 0;

void setup() {
  pinMode(buttonPin, INPUT); // the buttonPin is an input
  pinMode(ledPin, OUTPUT);   // the ledPin is an an output
}

void loop() {
  // while the button press count is zero, the LED is off.
  while(i == 0) {
          analogWrite(ledPin, brightness);
          buttonState = digitalRead(buttonPin);
          // if the button is pressed the brightness is set to full
          // and the button pressed count, i, increases by one
          if(buttonState == 1) {
          i++;
          brightness = 255;
          delay(1000);
          }
  }
```

```
// checks to see if the button is held down, if so then the
// LED will stay at the brightness. It will escape this while
// loop after the button is released.
while(buttonState == 1) {
        analogWrite(ledPin, brightness);
        buttonState = digitalRead(buttonPin);
}

// while the button press count is 1, the LED is at full brightness.
while(i == 1) {
        analogWrite(ledPin, brightness);
        buttonState = digitalRead(buttonPin);
        // if the button is pressed the brightness is set to 170
        // and the button pressed count, i, increases by one
        if(buttonState == 1) {
        i++;
        brightness = 170;
        delay(1000);
        }
}

// checks to see if the button is held down, if so then the
// LED will stay at the brightness. It will escape this while
// loop after the button is released.
while(buttonState == 1) {
        analogWrite(ledPin, brightness);
        buttonState = digitalRead(buttonPin);
}

// while the button press count is 2, the LED is at 2/3rds full brightness
while(i == 2) {
```

```
        analogWrite(ledPin, brightness);
        buttonState = digitalRead(buttonPin);
        // if the button is pressed the brightness is set to 85
        // and the button pressed count, i, increases by one
        if(buttonState == 1) {
        i++;
        brightness = 85;
        delay(1000);
        }
}

// checks to see if the button is held down, if so then the
// LED will stay at the brightness. It will escape this while
// loop after the button is released.
while(buttonState == 1) {
        analogWrite(ledPin, brightness);
        buttonState = digitalRead(buttonPin);
}

// while the button press count is 2, the LED is at 1/3rd full brightness
while(i == 3) {
        analogWrite(ledPin, brightness);
        buttonState = digitalRead(buttonPin);
        // if the button is pressed the brightness is set to 0
        // and the button pressed count, i, is reset to 0
        if(buttonState == 1) {
        i = 0;
        brightness = 0;
        delay(1000);
        }
}
```

```
  // checks to see if the button is held down, if so then the
  // LED will stay at the brightness. It will escape this while
  // loop after the button is released.
  while(buttonState == 1) {
        analogWrite(ledPin, brightness);
        buttonState = digitalRead(buttonPin);
  }
}
```