TECHNOVUS

Build-It-Yourself Shops Document – Module 3

Intended Use & Scope

This document is developed by Technovus as an aid for members attending the workshops. The scope and context of the topics covered here is limited to embedded programming (C/C++), especially programming Arduino microcontrollers. The Build-It-Yourself Shops consist of six modules. Each module ends with a building exercise which will provide a way to practically use the information provided in the module.

Module 1 is designed to quickly introduce programming to beginners. We will also take a look at the Serial command available in the Arduino IDE. Module 2 covers reading and writing digital data. Module 3 progresses to reading and writing analog data along with working with interrupts. Module 4 will introduce servo-motor control using Pulse Width Modulation (PWM). Module 5 and 6 will allow members to put their newly acquired knowledge to test by building a mini-project that encompasses topics covered in previous modules.

The ultimate objective of this document and the Build-It-Yourself Shops is to get members equipped for better and bigger projects.

Contents

Module 3: Analog Write/Read & Interrupts

Analog Read

```
analogRead(pin number);
```

This function allows us to read an analog signal from a specified pin. The Arduino board maps the voltage on the pin (0 to 5V) to an integer value between 0 and 1023. The function returns this integer value. The maximum frequency at which an analog input can be read is 10,000 times per second. As analog pins are already specified on the board (pin numbers 0 to 5 in Arduino Uno), there is no need to use pinMode function to set the mode of the pin.

Analog Write

```
analogWrite(pin number, value);
```

This function allows us to write analog values from a specified pin. This functions accepts numbers between 0-255 for the value parameter. To be specific, this function doesn't output an actual analog signal. The value parameter corresponds to the duty cycle of the outputted signal. Arduino uses pulse width modulation (PWM) to produce a signal between 0 and 5 Volts. We will also use PWM in module 4.

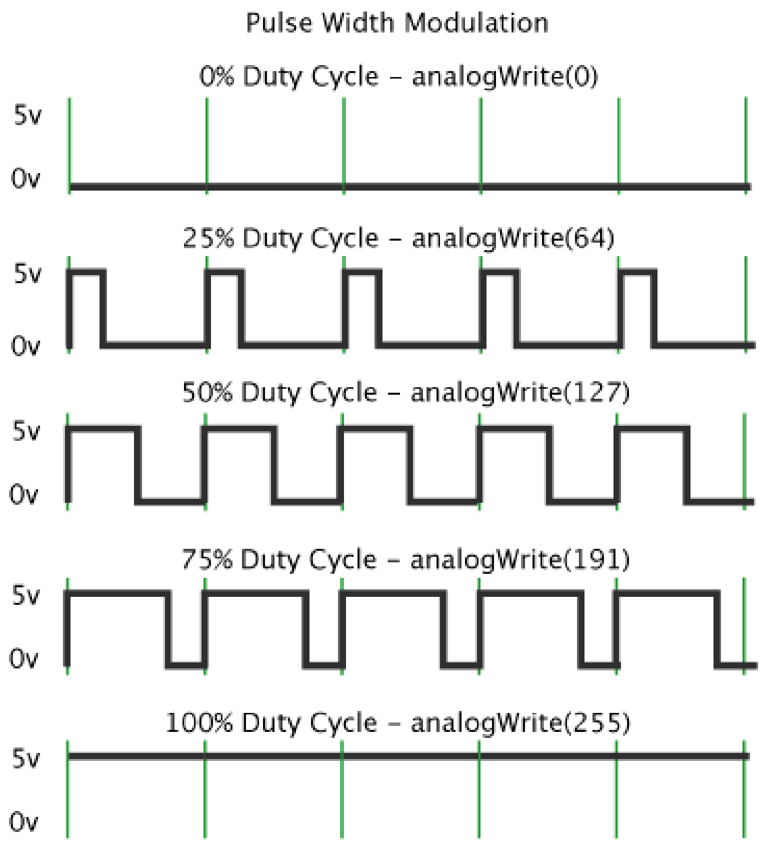

Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
5v
0v
25% Duty Cycle – analogWrite(64)
5v
0v
50% Duty Cycle – analogWrite(127)
5v
0v
75% Duty Cycle – analogWrite(191)
5v
0v
100% Duty Cycle – analogWrite(255)
5v
0v

Figure 1: Pulse Width Modulation (Source: Arduino.cc)

Interrupts

As we saw at the end of Module 2, using interrupts offers a robust way to ensure that important events are handled right away. An Interrupt Service Routine (ISR) is a special function that cannot accept any parameters and cannot return anything. In order to setup an interrupt in Arduino, we need to attach an ISR function to an interrupt. Anytime the interrupt is triggered by an external event, the ISR will run as soon as the interrupt is triggered. After the ISR function has run, the program resumes normal operation.

Attaching Interrupts

```
attachInterrupt(digitalPinToInterrupt(pin number), ISR, mode);
```

This function is used to attach interrupts to a pin number. Based on the mode, the ISR will be triggered. Mode can be either LOW, CHANGE, RISING, or FALLING. If mode is LOW, the ISR will be triggered whenever the signal received by the specified pin is low. If mode is CHANGE, the ISR will be triggered whenever the signal received by the specified pin changes from HIGH to LOW or LOW to HIGH. If mode is RISING, the ISR will be triggered whenever the signal received by the specified pin changes from LOW to HIGH. If mode is FALLING, the ISR will be triggered whenever the signal received by the specified pin changes from HIGH to LOW. This function should be called within the setup() Arduino function.

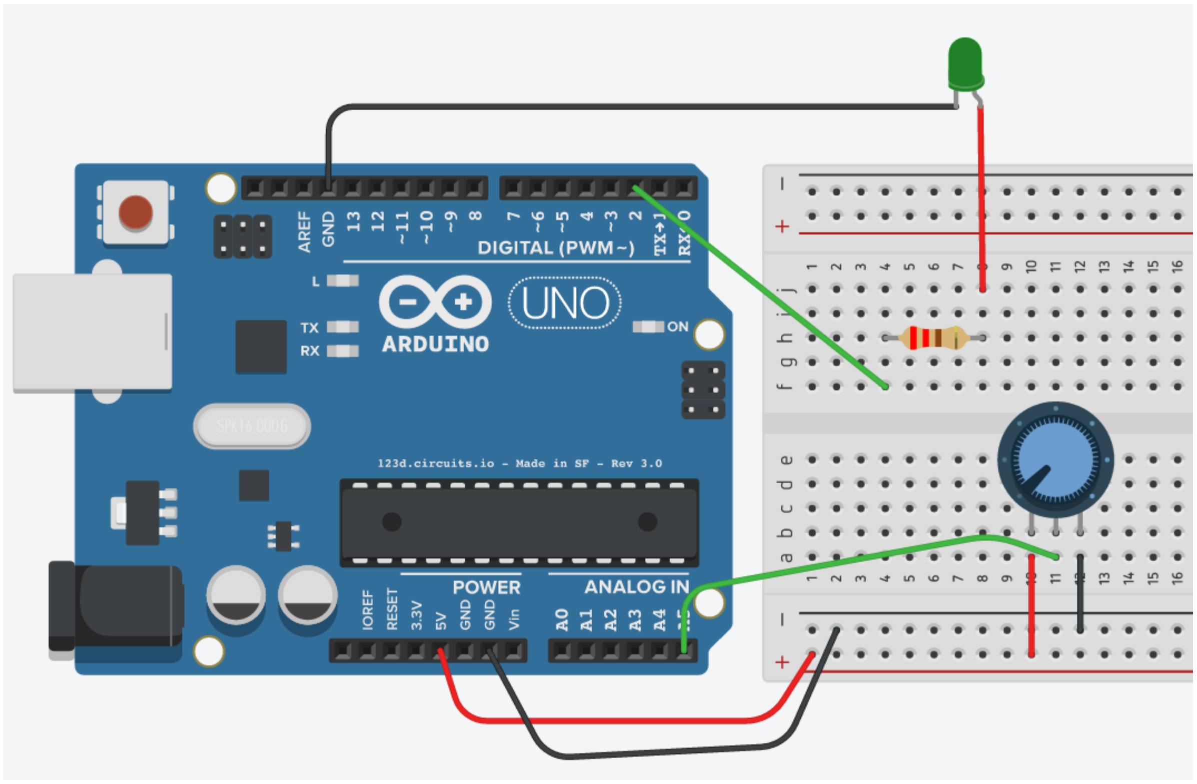
Build! (1)

This module consists of two exercises. The first one involves changing the blinking rate of an LED using the potentiometer.

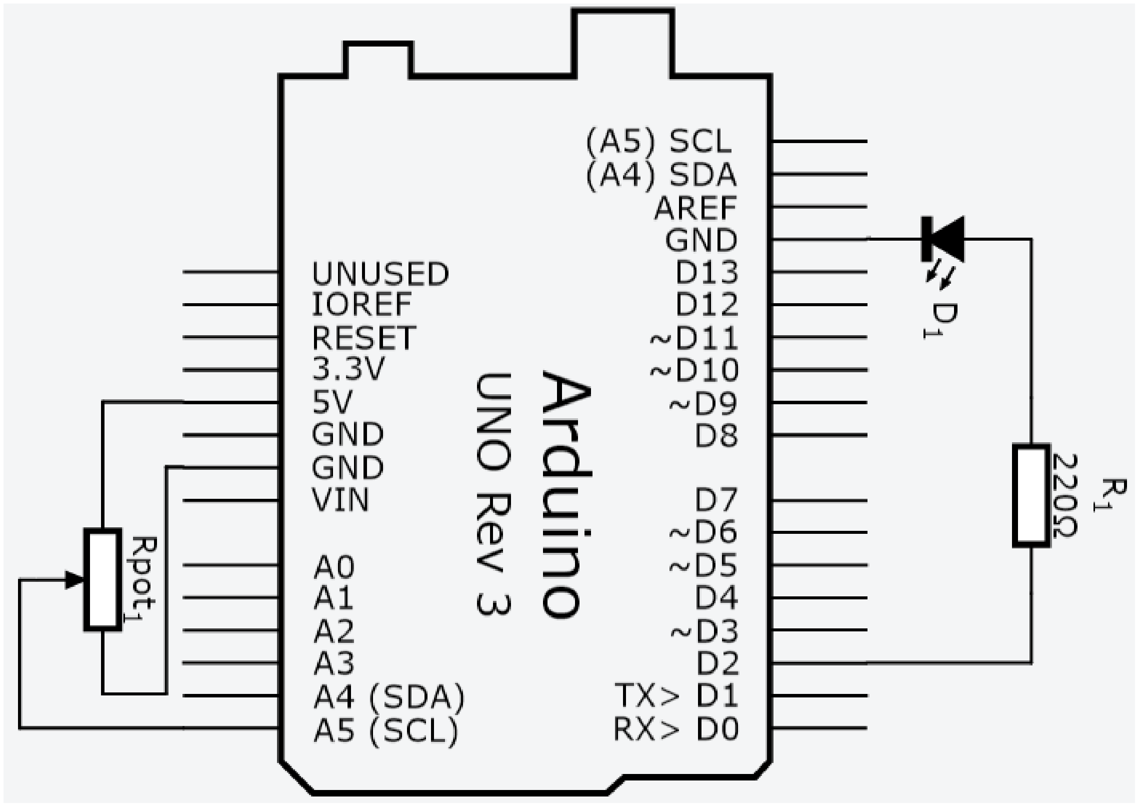
Items required:

1. Microcontroller
2. LED
3. Resistor – 220 ohm
4. Potentiometer

Circuit



Schematic



Code

```
const int ledPin = 2;

const int potPin = A5;

int delayValue = 0;


// the setup routine runs once when you press reset:

void setup() {

  // initialize the digital pin as an output.

  pinMode(ledPin, OUTPUT);

}


// the loop routine runs over and over again forever:

void loop() {

  delayValue = analogRead(potPin);    // read potentiometer value

  digitalWrite(ledPin, HIGH);    // turn the LED on

  delay(delayValue);                // wait

  digitalWrite(ledPin, LOW);     // turn the LED off

  delay(delayValue);                // wait

}
```

To see PWM in action, we can replace digitalWrite with analogWrite and use any pin capable of outputting PWM. A delay of 30ms within loop() may be required to be able to see the dimming effect. Now, varying the potentiometer value will vary the brightness.

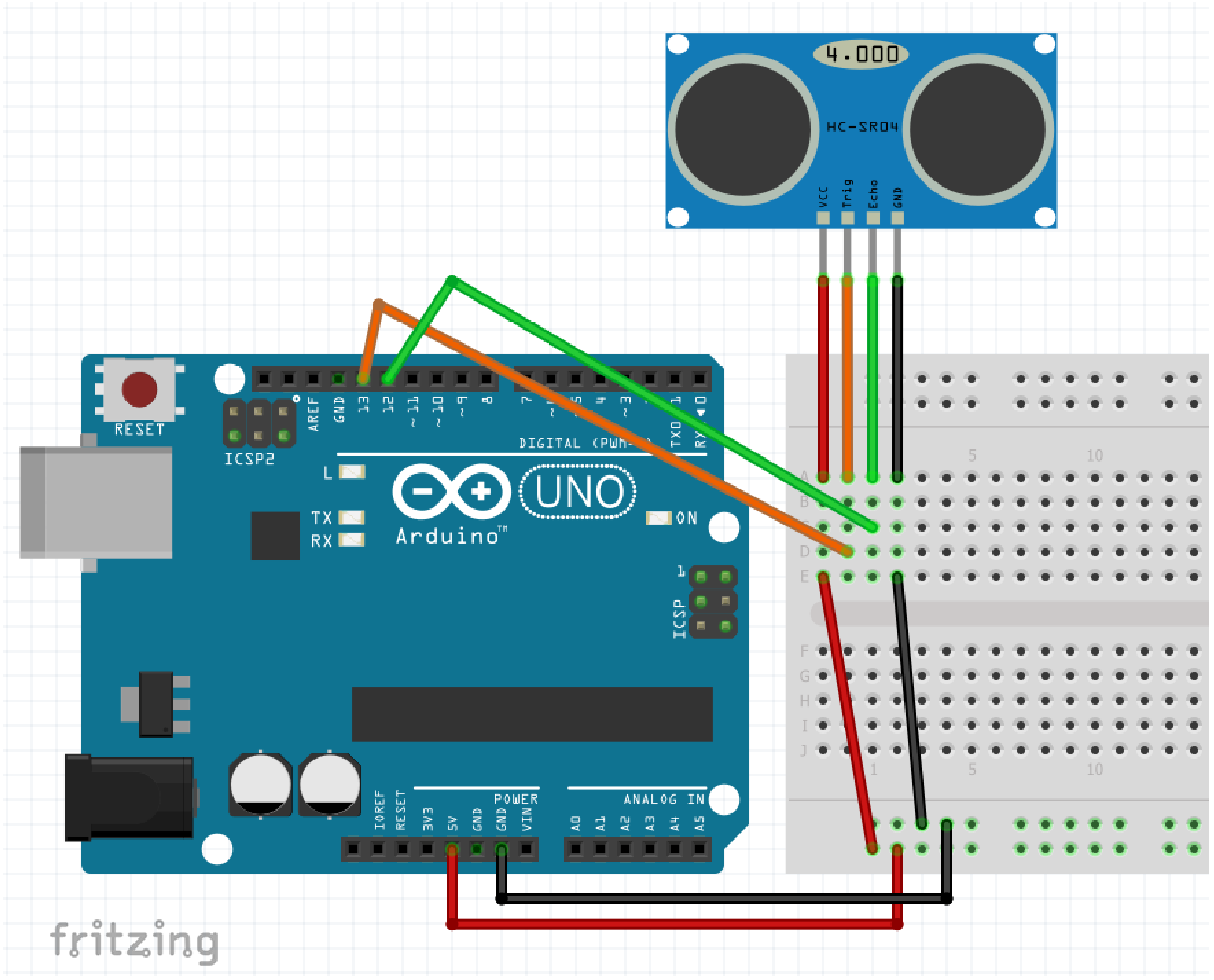
Build! (2)

The second exercise involves reading the distance from an ultrasonic sensor and displaying it using the serial monitor. An interrupt is used to read the echo signal from the ultrasonic sensor.

Items required:

1. Microcontroller
2. Ultrasonic Sensor

Circuit



Code

```
const int trigPin = 9;

const int echoPin = 2;

volatile long echoStart = 0;

volatile long echoEnd = 0;

volatile long echoDuration = 0;

long distance = 0;


void setup() {

  Serial.begin (9600);

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  attachInterrupt(digitalPinToInterrupt(echoPin), getDuration,CHANGE);

}


void loop() {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  distance = (echoDuration / 58);

  Serial.println(distance);

}
```

```
void getDuration()
{
  if (digitalRead(echoPin)) // Check if Echo Signal is High or Low
  {
    echoEnd = 0; // Reset the end time to 0
    echoStart = micros(); // Save the start time
  } else {
    echoEnd = micros(); // Save the end time
    echoDuration = echoEnd - echoStart; // Calculate pulse duration
  }
}
```