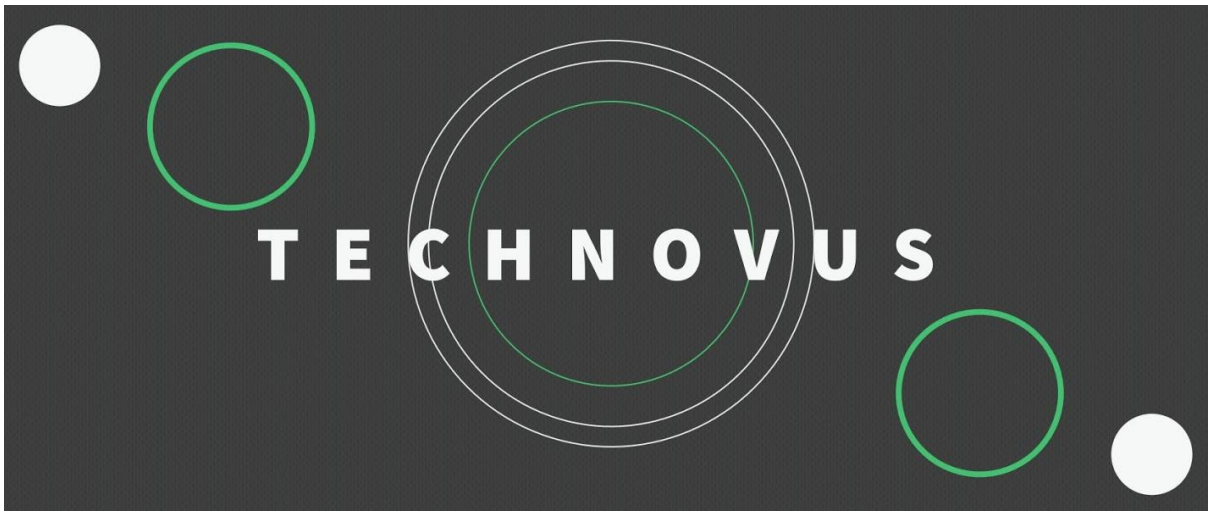


Build-It-Yourself Shops Document – Module 2

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 2: Digital Write/Read 1
Digital Vs Analog 1
Digital Read 2
Build! 2
Circuit 3
Schematic 3
Code 4
Version I 4
Good to know 6
Polling Vs Interrupts 6

Module 2: Digital Write/Read

Digital Vs Analog

A digital signal can only have discreet values (such as ON and OFF, also referred to as HIGH and LOW). As can be seen in the figure below, the digital signal jumps between LOW and HIGH values. A digital signal can never have an intermediate value between LOW and HIGH. In contrast, an analog signal can continuously vary between the smallest and the largest values of the signal.

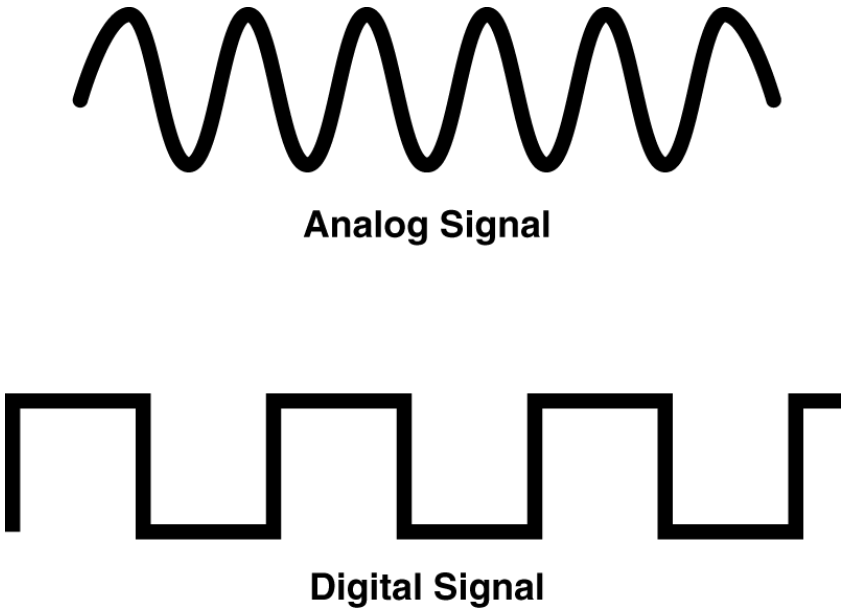


Figure 1: Digital Signal Vs Analog Signal (AllAboutCircuits.com)

In the context of programming Arduino microcontrollers, we will be dealing with digital signals where 0 Volts corresponds to LOW or OFF and 5V corresponds to HIGH or ON. In this workshop module, we will learn how to use sensors that output digital signals and control actuators (LEDs) using digital signals. Module 3 will cover analog signals.

Digital Read

```
digitalRead(pin number);
```

As the name implies, this function lets us read the signal from a specified pin. Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode() when using them as inputs. Pins configured this way are said to be in a high-impedance state. Input pins make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 megaohm in front of the pin. This also means however, that pins configured as pinMode(pin, INPUT) with nothing connected to them, or with wires connected to them that are not connected to other circuits, will report seemingly random changes in pin state, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin.

Similar to digitalWrite, we can set the mode for the pin number as input using the pinMode function within setup().

```
pinMode(pin number, INPUT);
```

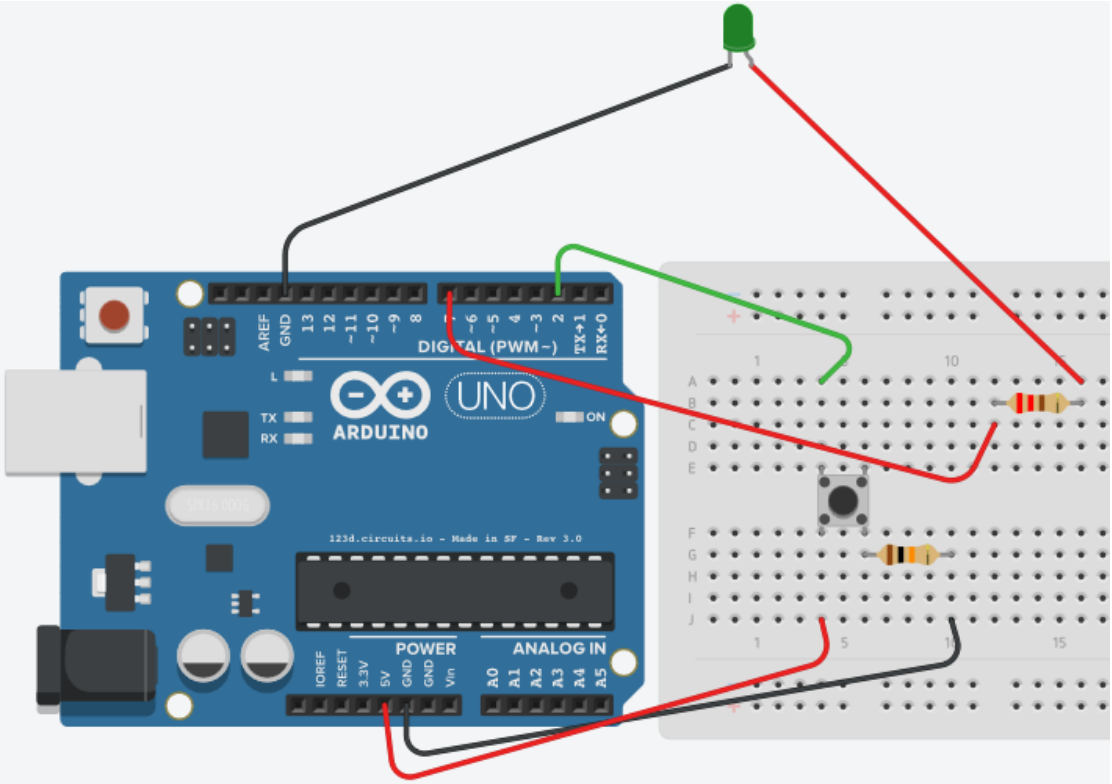
Build!

This building exercise involves controlling an LED with a pushbutton (switch).

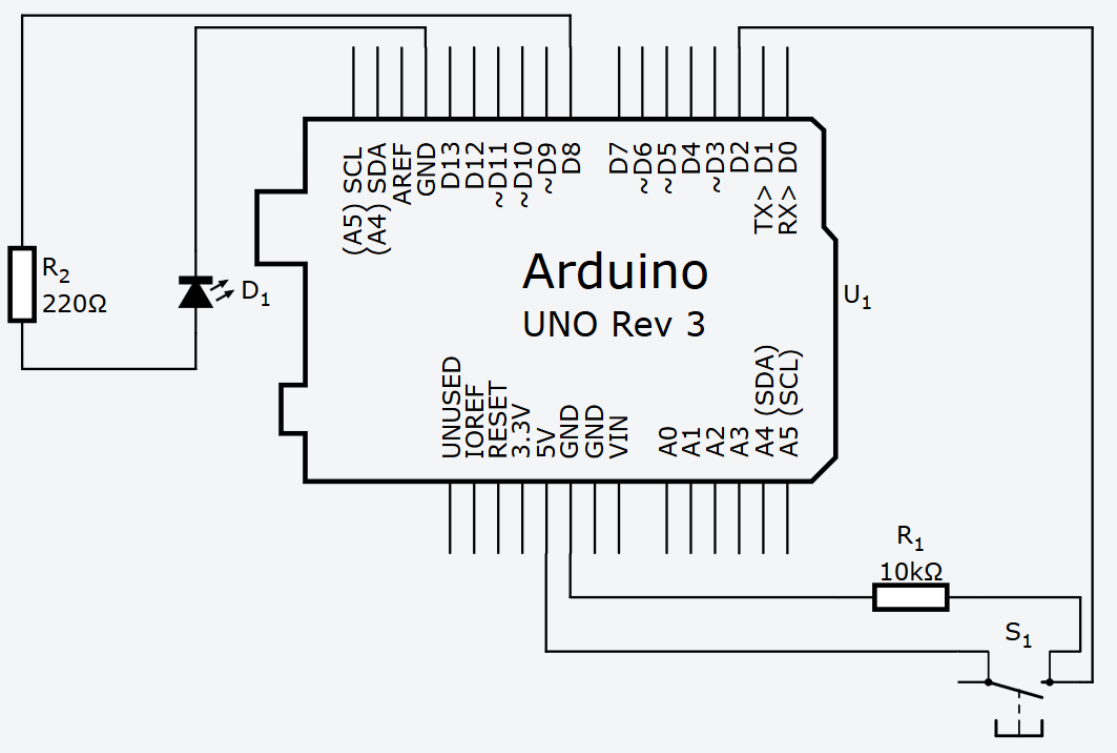
Items required:

1. Microcontroller
2. LED
3. Resistors – 220 ohm and 10K ohm
4. Pushbutton

Circuit



Schematic



Code

In this module, first we upload code to the microcontroller that does not include any delays. Next, we add delay of a few seconds to the code and observe how this affects the performance of the code.

Version I

```
const int LEDPin = 8;

const int buttonPin = 2;

bool button = LOW;

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

void setup() {

  pinMode(LEDPin, OUTPUT);

  pinMode(buttonPin, INPUT);

}


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

void loop() {

  button = digitalRead(buttonPin);

  if(button){

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

    Serial.println("Button is pressed");

  }

  else{

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

  }

}
```

Version II

```
const int LEDPin = 8;

const int buttonPin = 2;

bool button = LOW;

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

void setup() {

  pinMode(LEDPin, OUTPUT);

  pinMode(buttonPin, INPUT);

}


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

void loop() {

  button = digitalRead(buttonPin);

  if(button){

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

    Serial.println(“Button is pressed”);

  }

  else{

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

  }

delay(5000)      // wait for 5 seconds

}
```

Good to know

Polling Vs Interrupts

In this building exercise, we check if the button has been pressed or not by using the ‘if’ statement in the loop(). The ‘if’ statement runs during each iteration of the loop(). In other words, the microcontroller is polling for the value of button using the digitalRead function and the ‘if’ statement. Imagine if our code did more than just lighting up an LED, we can simulate this by adding a delay of a few seconds after the ‘else’ statement (as seen in version 2 of the code). Now, the button might not work perfectly. If you press and then un-press the button for a short duration while the microcontroller is running the delay function, or any line except reading the buttonPin, then the LED won’t turn on because the microcontroller would not detect the button to be pressed. This problem can be solved by using interrupts. More on interrupts will be described in Module 3.