Engineering Technology (ENGR 101)

Arduino and servo motors

Mohammad Nekooei

Engineering and Computer Science

Victoria University of Wellington

Admin

- Lab 7
  - Due date is June 8, 19:00 (Xiamen Time)
  - This lab is in groups.
  - There is a correction
- Lab 8 has been released
  - Due date is June 15, 19:00 (Xiamen Time)
  - This lab is in groups.

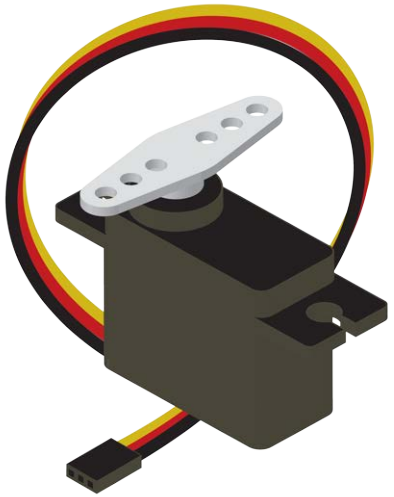
- Students who have not submitted their lab projects
  - Assignments 12% of final grade
  - Labs & project 38% of final grade
  - https://ecs.wgtn.ac.nz/Courses/XMUT101_2021T1/XMUT101CourseOutline

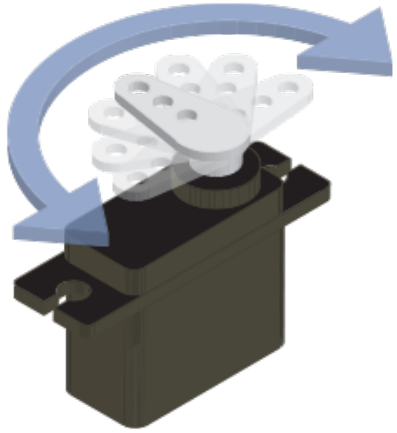
Road Map

Write Arduino programs							
Copy code from an example	Perform basic arithmetic operations	Control LEDs from Code	Read Analog and digital input	Read input from Sensors	Adjusting direction of servo motor	Interrupts	**Design and Create an Arduino-based Controller project**
Build circuits to support Arduino							
Use a breadboard to build a circuit	Apply Ohm's Law	Turn LEDs on and off	Connect buttons	Connect sensors	Build circuits to adjust direction of servo motor	Using Interrupts	

Servo Motor

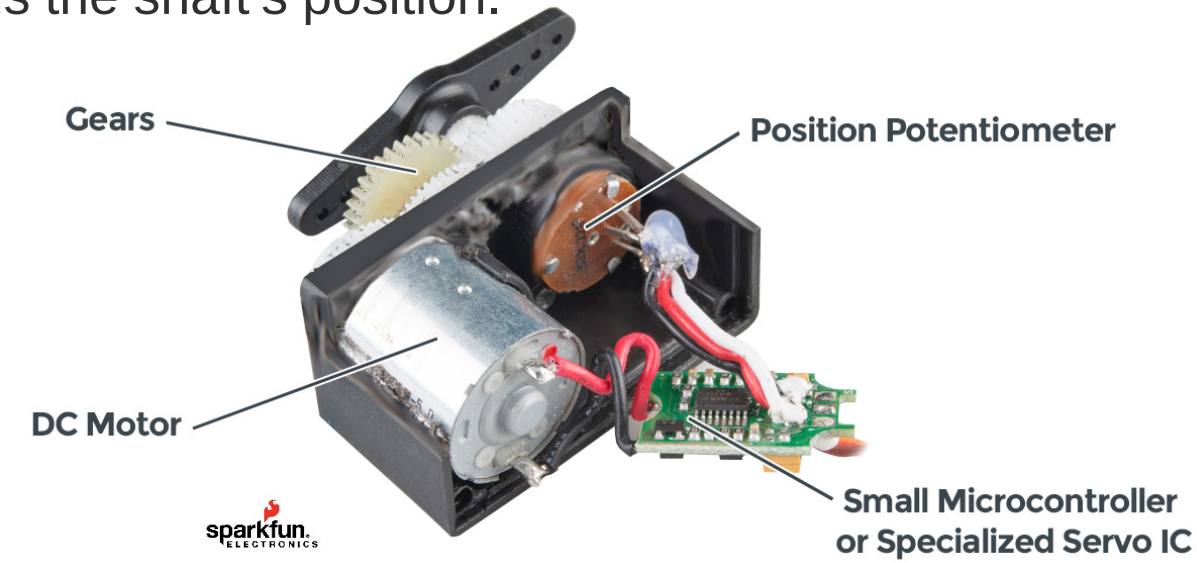
- We can use a motor to move a mechanical equipment to a particular location.
- With a Servo motor (Servo)
  - We can rotate the shaft to a particular location and hold it there.
  - We cannot move the arm more that 180 degrees.
- Servos are very useful, if you need to make precise movements but do not need to something to fully spinning around.
  - Robotics arms and hands





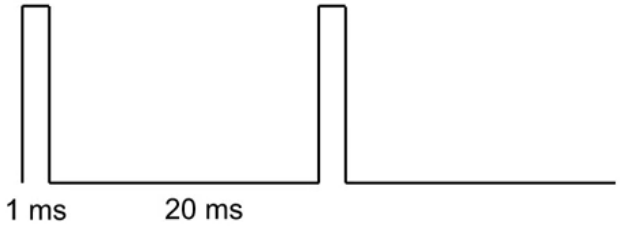
How does a servo work?

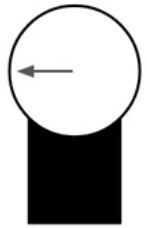
- Servo has a DC motor, a specialized circuitry (controller circuit), and a potentiometer
  - The DC motor is attached to a gearbox and output/drive shaft.
  - The circuitry reads signals sent by Arduino and determines where the shaft needs to turn to.
  - It powers the DC motor which turns the main shaft.
  - This shaft turns the potentiometer that the circuitry reads.
  - The potentiometer tells the circuitry how far the shaft has turned and acts as feedback mechanism.
  - Once the desire angle is reached, the servo holds the shaft's position.



Controlling a servo

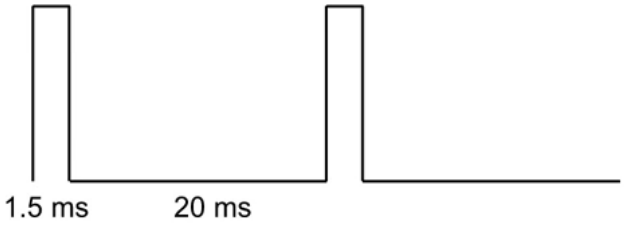
- To control a servo, we send precisely timed pulses.
  - To turn it as far as it goes counter clockwise, we send it 1ms digital HIGH pulse following 20ms of digital LOW

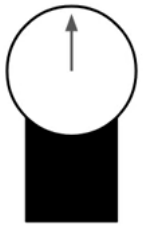




Full counterclockwise

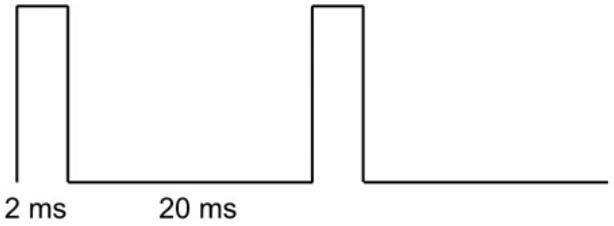
  - To centre the shaft, we send it 1.5ms digital HIGH pulse following 20ms of digital LOW

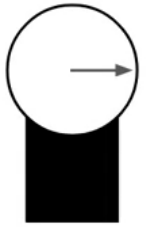




Centered

  - To turn it as far as it goes clockwise, we send it 2ms digital HIGH pulse following 20ms of digital LOW

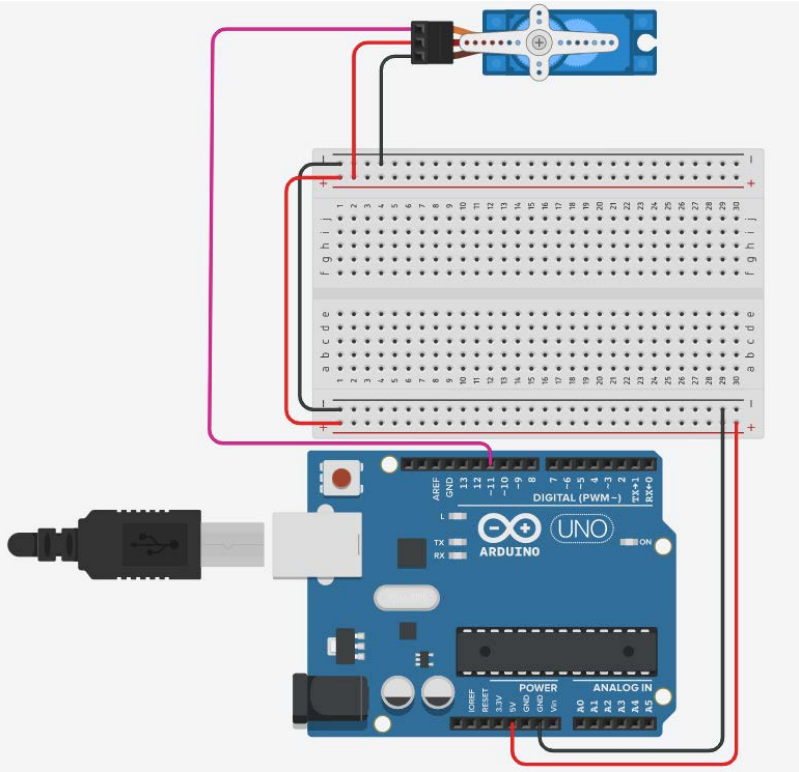




Fully clockwise

Controlling a servo with Arduino

- Arduino Uno can use PWM pins to control a servo.

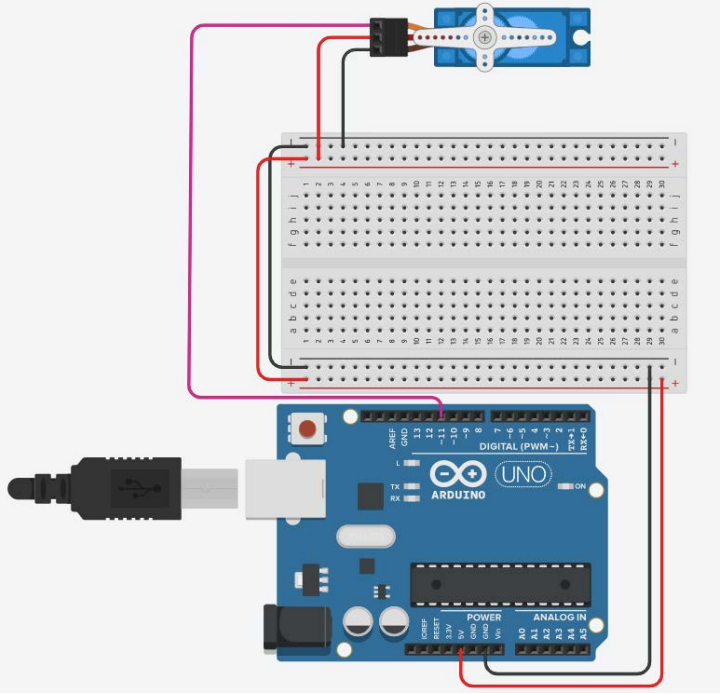


```
// Includes the Servo library
#include <Servo.h>
int servo_pin = 11;
Servo servo;
void setup() {
   // Attach pin to the servo motor
   servo.attach(servo_pin);
}

void loop() {
   servo.write(0);   // set servo to Fully clockwise
   delay(1000);
   servo.write(90);  // set servo to Centre
   delay(1000);
   servo.write(180); // set servo to Full counterclockwise
   delay(1000);
}
```

Controlling a servo with Arduino

- Arduino Uno can use PWM pins to control a servo.

```
// Includes the Servo library
#include <Servo.h>
int servo_pin = 11;
Servo servo;
void setup() {
   // Defines on which pin is the servo motor attached
   servo.attach(servo_pin);
}

void loop() {
   for(int pos = 0; pos <=180; pos++){
      servo.write(pos);
      delay(50);
   }
}
```



Road Map

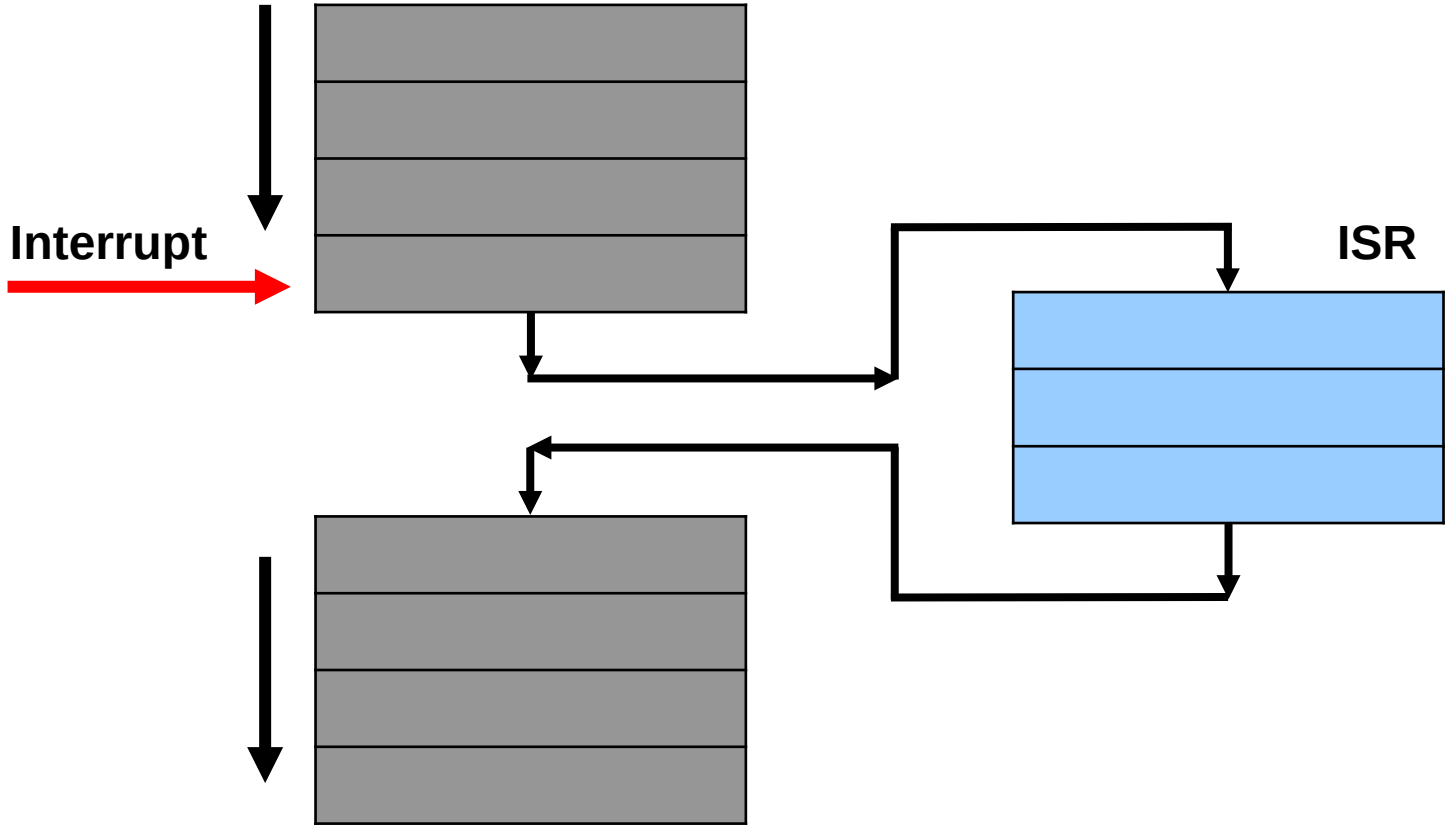
Write Arduino programs						
Copy code from an example	Perform basic arithmetic operations	Control LEDs from Code	Read Analog and digital input	Read input from Sensors	Adjusting direction of servo motor	Interrupts
Build circuits to support Arduino						
Use a breadboard to build a circuit	Apply Ohm's Law	Turn LEDs on and off	Connect buttons	Connect sensors	Build circuits to adjust direction of servo motor	Using Interrupts

Design and Create an Arduino-based Controller project

Interrupt

- Interrupts are a way for a microcontroller to temporarily stop what it is doing to handle another task.

- The currently executing program is paused, an ISR (interrupt service routine) is executed, and then your program continues.



Types of Interrupts

- **Hardware Interrupt:** It happens when an external event occurs like an external interrupt pin changes its state from LOW to HIGH or HIGH to LOW.

- **Software Interrupt:** It happens according to the instruction from the software. For example, *Timer interrupts* are software interrupt.

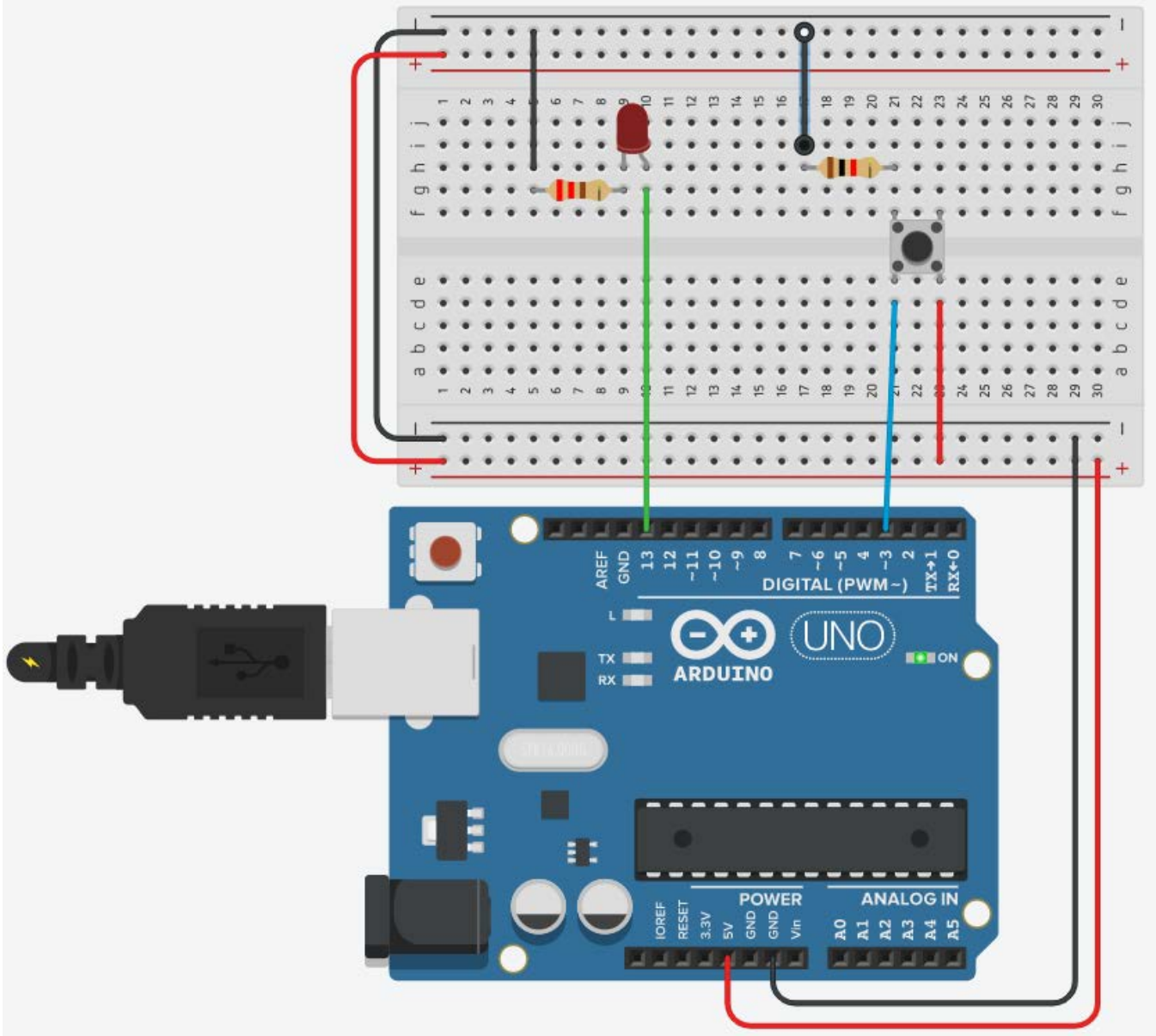
Interrupts in Arduino

- **External Interrupt:** These interrupt are interpreted by hardware and are very fast. These interrupts can be set to trigger on the event of RISING or FALLING or LOW levels.
  - External interrupts pins: 2 and 3

- **Pin Change Interrupt:** Arduinos can have more interrupt pins enabled by using pin change interrupts.

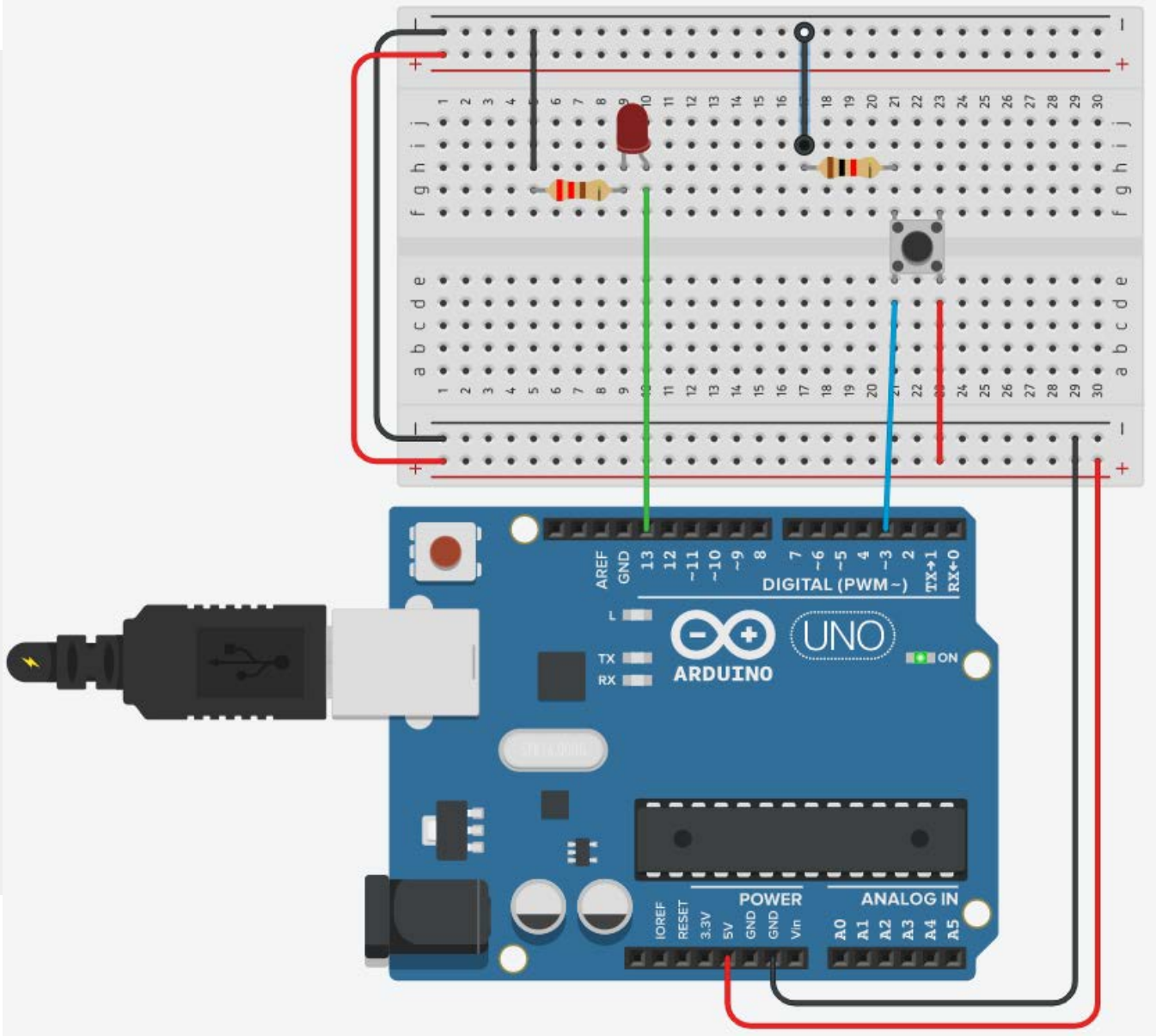
Example 1 (no interrupts)

```
const int LEDpin =  13;
const int switchPin = 3;
void setup() {
    pinMode(LEDpin,OUTPUT);
    pinMode(switchPin,INPUT);
}
void loop() {

    handleSwitch();
}
void handleSwitch (){
    int reading = digitalRead(switchPin);
    digitalWrite(LEDpin, reading);
}
```



Example 2 (no interrupts)

```
const int LEDpin =  13;
const int switchPin = 3;
void setup() {
    pinMode(LEDpin,OUTPUT);
    pinMode(switchPin,INPUT_PULLUP);
}
void loop() {

    handleSwitch();
    handleOtherStuff();
}
void handleSwitch (){
    int reading = digitalRead(switchPin);
    digitalWrite(LEDpin, reading);
}

void handleOtherStuff (){
    delay(1000);
}
```



Using Interrupts

- **Interrupt Service Routine (ISR)**

- Interrupt Service Routine or an Interrupt handler is an event that has small set of instructions in it.

- When an external interrupt occurs, the processor first executes these code that is present in ISR and returns to state where it left the normal execution.

ISR syntax in Arduino

- **attachInterrupt(** *digitalPinToInterrupt(*pin*)*, ISR, mode**)**

- *pin*: the Arduino pin number. In Arduino Uno, the pins used for interrupt are **2,3**.
- *ISR*: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.
- *mode*: defines when the interrupt should be triggered. Four constants are predefined as valid values:
  - LOW to trigger the interrupt whenever the pin is low,
  - CHANGE to trigger the interrupt whenever the pin changes value
  - RISING to trigger when the pin goes from low to high,
  - FALLING for when the pin goes from high to low.

ISR syntax in Arduino

- **digitalPinToInterrupt(pin)**, translate the actual digital pin to the specific interrupt number.
- For example, if you connect to pin 3, use **digitalPinToInterrupt(3)** as the first parameter to **attachInterrupt()**.
- Interrupt Service Routine function (ISR) must be as short as possible.
- **Delay**() function doesn’t work inside ISR and should be avoided.
- **millis**() relies on interrupts to count, so it will never increment inside an ISR.

Example 3 (interrupt)

```
const int LEDpin =  13;
const int switchPin = 3;
void setup() {
    pinMode(LEDpin,OUTPUT);
    pinMode(switchPin,INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(switch_pin), handleSwitch, CHANGE);
}
void loop() {

    // handleSwitch(); ← commented out
    handleOtherStuff();
}
void handleSwitch (){  //ISR
    int reading = digitalRead(switchPin);
    digitalWrite(LEDpin, reading);
}

void handleOtherStuff (){
    delay(1000);
}
```