Conditions

- To make decisions in Arduino code we use an ‘if’ statement
- ‘If’ statements are based on a TRUE or FALSE question

IF Condition

```
if(true)
{
    “perform some action”
}
```

Value comparisons

GREATER THAN

a > b

GREATER THAN OR EQUAL

a >= b

LESS

a < b

LESS THAN OR EQUAL

a <= b

EQUAL

a == b

NOT EQUAL

a != b

IF Example

```
int counter = 0;

void setup() {
   Serial.begin(9600);
}
void loop() {

   if(counter < 10)
   {
        Serial.println(counter);
   }
   counter = counter + 1;

}
```

IF - ELSE Condition

```
if( “answer is true”)
{
      “perform some action”
}
else
{
      “perform some other action”
}
```

IF - ELSE Example

```
int counter = 0;
void setup() {
     Serial.begin(9600);
}
void loop() {

     if(counter < 10)
     {
          Serial.println("less than 10");
     }
     else
     {
          Serial.println("greater than or equal to 10");
          Serial.end();
     }
     counter = counter + 1;
}
```

IF - ELSE IF Condition

```
if( “answer is true”)
{
      “perform some action”
}
else if( “answer is true”)
{
      “perform some other action”
}
```

IF – ELSE IF Example

```
int counter = 0;
void setup() {
      Serial.begin(9600);
}
void loop() {

      if(counter < 10)
      {
            Serial.println("less than 10");
      }
      else if (counter == 10)
      {
            Serial.println("equal to 10");
      }
      else
      {
            Serial.println("greater than 10");
            Serial.end();
      }
      counter = counter + 1;
}
```

Boolean operators - AND

- If we want all of the conditions to be true we need to use 'AND' logic (AND gate)
- We use the symbols **&&**

Example

```
if ( val > 10 && val < 20)
```

Boolean operators - OR

- If we want either of the conditions to be true we need to use ‘OR’ logic (OR gate)
- We use the symbols **||**

Example

```
if ( val < 10 || val > 20)
```

Brightness Example

```
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()  {
pinMode(led, OUTPUT); // declare pin 9 to be an output

}

// the loop routine runs over and over again forever:
void loop()  {

  analogWrite(led, brightness); // set the brightness of pin 9

  // 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 ;
  }

  delay(30);   // wait for 30 milliseconds to see the dimming effect
}
```

Boolean variables

```
Boolean done = true;
```

Example

```
boolean done = false;
void setup()
{
Serial.begin(9600);
}
void loop()
{
  if(!done)
  {    Serial.println(“HELLO WORLD”);
       done = true;
  }

}
```

Reading data from Arduino

```
void setup()
{
Serial.begin(9600);
}

void serialtest()
{
int i;
for(i=0; i<10; i++)
Serial.println(i);
}
```

Using switches and buttons

```
const int inputPin = 2; // choose the input pin

void setup()
{
pinMode(inputPin, INPUT); // declare pushbutton as input
}

void loop()
{
int val = digitalRead(inputPin); // read input value
}
```

The const keyword stands for constant. It is a variable qualifier that modifies the behavior of the variable, making a variable "read-only". This means that the variable can be used just as any other variable of its type, but its value cannot be changed. You will get a compiler error if you try to assign a value to a const variable. You can use either const or `#define` for creating numeric or string constants. For arrays, you will need to use const
In general const is preferred over `#define` for defining constants.

Reading analog inputs and scaling

```
const int potPin = 0; // select the input pin for potentiometer

void loop()
{
int val; // The value coming from the sensor

int percent; // It will be the mapped value

val = analogRead(potPin); // read the voltage on the pot (val
                          //ranges from 0 to 1023)

percent = map(val,0,1023,0,100); // percent will range from
                                 // 0 to 100.
}
```

Creating a bar graph using LEDs for sensor value

```
const int NumLEDs = 8;
const int ledPins[] = { 6, 7, 8, 9, 10, 11, 12, 13};
const int analogInPin = 0; // Analog input pin const int
const boolean LED_ON = HIGH;
const boolean LED_OFF = LOW;

int sensorValue = 0; // value read from the sensor
int ledLevel = 0; // sensor value converted into LED'bars'

void setup()
{
   for (int i = 0; i < NumLEDs; i++)
   {
        pinMode(ledPins[i], OUTPUT); // make all LED pins outputs
   }
}
```

Creating a bar graph using LEDs for sensor value

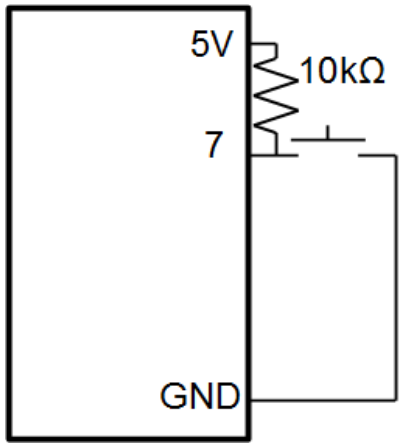
```
void loop()
{
sensorValue = analogRead(analogInPin); // read the analog in value
ledLevel = map(sensorValue, 0, 1023, 0, NumLEDs); // map to the
                                                  //number of LEDs
for (int i = 0; i < NumLEDs; i++)
   {
   if (i < ledLevel )
        {
        digitalWrite(ledPins[i], LED_ON); // turn on pins less than
                                          //the level
        }
   else
        {
        digitalWrite(ledPins[i], LED_OFF); // turn off pins higher
                                           //than the level
        }
   }
}
```

Arduino Digital Input

```
int ledPin = 13;        // LED connected to digital pin 13
int inPin = 7;         // pushbutton connected to digital pin 7
boolean val = 0;       // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);     // sets the digital pin 13 as
output
  pinMode(inPin, INPUT);      // sets the digital pin 7 as input
}

void loop()
{
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);  // sets the LED to the button's
                              //value
}
```

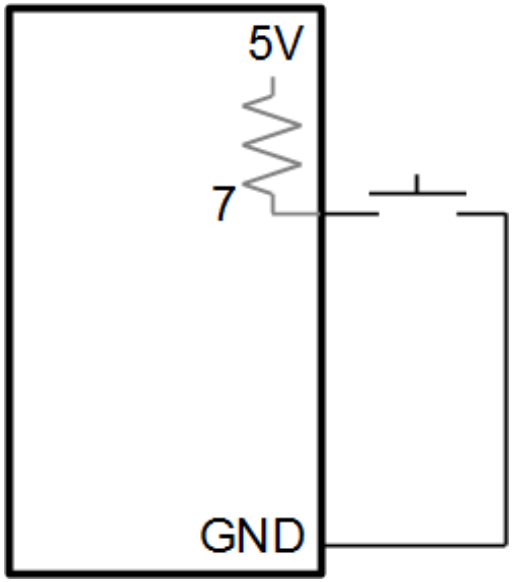


Arduino Digital Input

- Pins Configured as INPUT_PULLUP
- The Atmega chip on the Arduino has internal pull-up resistors (resistors that connect to power internally) that you can access. If you prefer to use these instead of external pull-down resistors, you can use the INPUT_PULLUP argument in pinMode().

Example

```
pinMode(7,INPUT_PULLUP);
```



Arduino Digital Input

```
int analogPin = 3; // potentiometer wiper (middle terminal)
                   //connected to analog pin 3 and the others
                  //lead to ground and +5V

int val = 0;           // variable to store the value read

void setup()
{
  Serial.begin(9600);          //  setup serial
}

void loop()
{
  val = analogRead(analogPin);    // read the input pin
  Serial.println(val);            // debug value
}
```

Arduino Digital Input

```
int ledPin = 9;      // LED connected to digital pin 9
int analogPin = 3;   // potentiometer connected to analog pin 3
int val = 0;         // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);   // sets the pin as output
}

void loop()
{
  val = analogRead(analogPin);   // read the input pin
  analogWrite(ledPin, val / 4);
}
```

Note : analogRead values go from 0 to 1023, analogWrite values from 0 to 255.

Measuring Temperature

```
const int inPin = 0; // analog pin 0

void loop()
{
int value = analogRead(inPin);

float millivolts = (value / 1024.0) * 3300; //3.3V analoginput
float celsius = millivolts / 10; // sensor output is 10mV per
                                 //degree Celsius

delay(1000); // wait for one second
}
```

ADC Example

```
// These constants won't change.  They're used to give names to the pins used:
const int analogInPin = A0;  // Analog input pin that the potentiometer is attached to
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
    Serial.begin(9600);  // initialize serial communications at 9600 bps:

}

void loop() {
    sensorValue = analogRead(analogInPin);      // read the analog in value:

    outputValue = map(sensorValue, 0, 1023, 0, 255); // map it to the range of analog out

    analogWrite(analogOutPin, outputValue);          // change the analog out value:

  Serial.print("sensor = " );              // print the results to the serial monitor
  Serial.print(sensorValue);
  Serial.print("\t output = ");           //it prints one tab and then prints output=
  Serial.println(outputValue);

  delay(2);   // wait 2 milliseconds before the next loop for the analog-to-digital
             //converter to settle after the last reading

}
```

Connecting LCDs

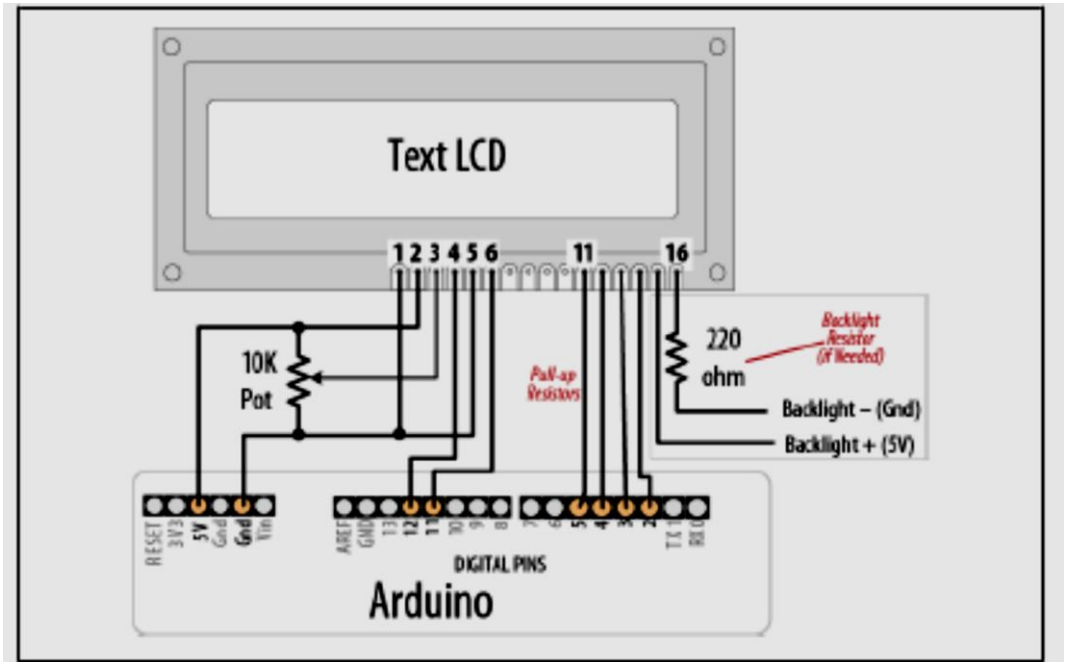
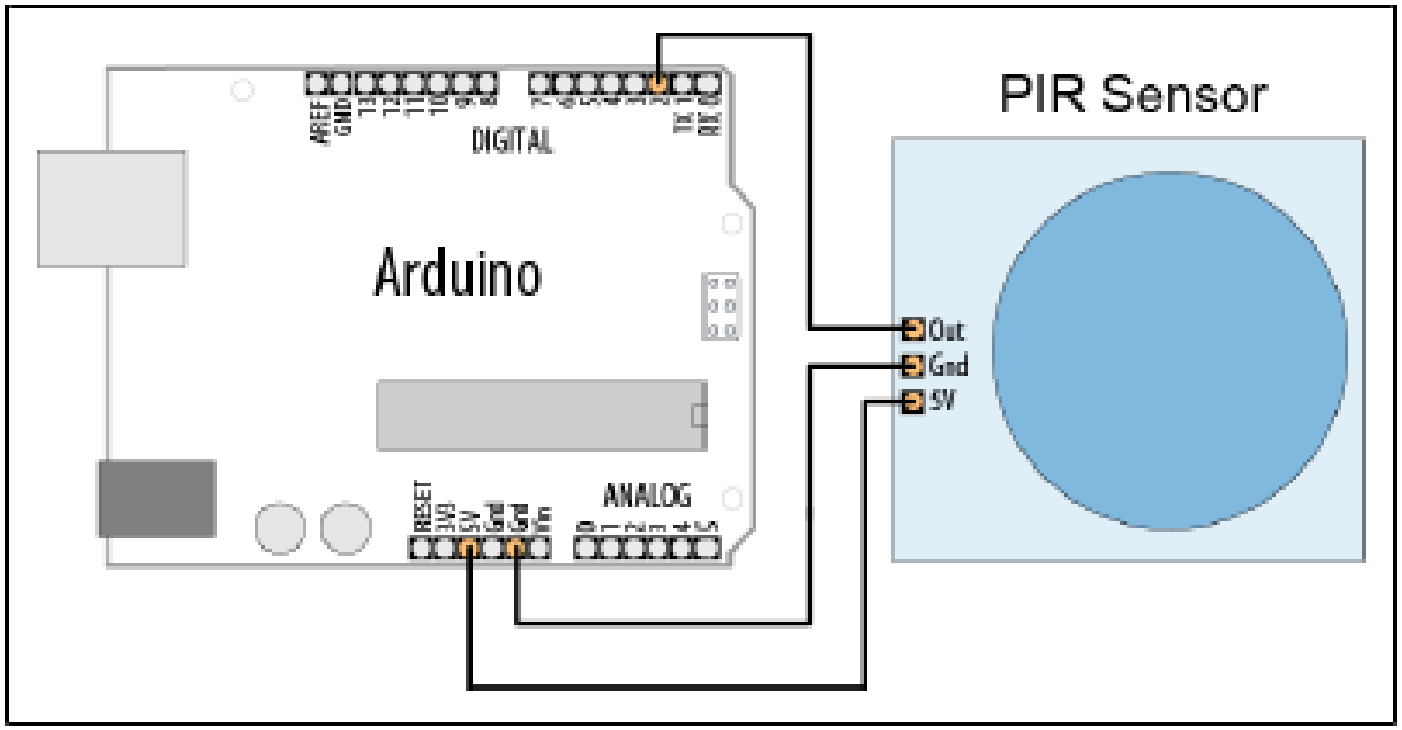


Table 11-1. LCD pin connections

LCD pin	Function	Arduino pin
1	Gnd or 0V or Vss	Gnd
2	+5V or Vdd	5V
3	Vo or contrast	
4	RS	12
5	R/W	
6	E	11
7	D0	
8	D1	
9	D2	
10	D3	
11	D4	5
12	D5	4
13	D6	3
14	D7	2
15	A or analog	
16	K or cathode	

```
#include <LiquidCrystal.h> // include the library code
//constants for the number of rows and columns in the LCD:
const int numRows = 2;
const int numCols = 16;
// initialize the library with the numbers of the interface pins:
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup()
{
lcd.begin(numCols, numRows);
lcd.print("hello, world!"); // Print a message to the LCD.
}
```

Using PIR motion sensors



PIR sensors allow you to sense motion, almost always used to detect whether a human has moved in or out of the sensors range. They are often referred to as PIR, "Passive Infrared", "Pyroelectric", or "IR motion" sensors.

Using PIR motion sensors

```
const int ledPin = 13; // pin for the LED
const int inputPin = 2; // input pin (for the PIR sensor)

void setup()
{
pinMode(ledPin, OUTPUT); // declare LED as output
pinMode(inputPin, INPUT); // declare pushbutton as input
}

void loop()
{
int val = digitalRead(inputPin); // read input value

   if (val == HIGH) // check if the input is HIGH
   {
   digitalWrite(ledPin, HIGH); // turn LED on if motion detected
   delay(500);
   digitalWrite(ledPin, LOW); // turn LED off
   }
}
```

Sensor Characteristics

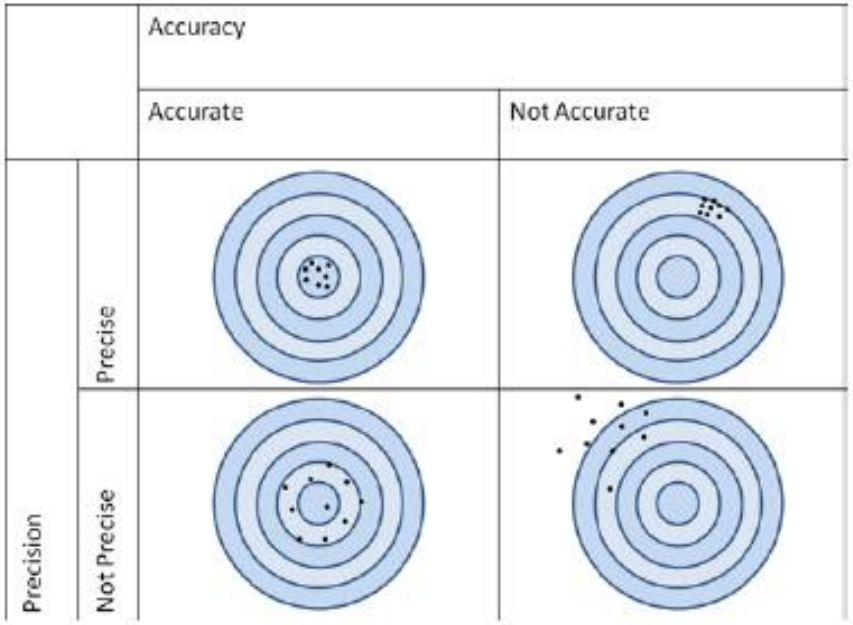
- Range
- Accuracy
- Precision
- Sensitivity the change in input required to generate a unit change in output
- Transfer Function S=F(x), where x is the measure and and S the electrical signal (commonly Voltage)

Sensor Characteristics

- Accuracy
- Precision

$$Percentage\ Relative\ Error = \frac{(Measured\ Value - True\ Value)}{(True\ Value)} \times 100$$

$$Percentage\ Standard\ Deviation = \frac{(Standard\ Deviation)}{(Mean)} \times 100$$

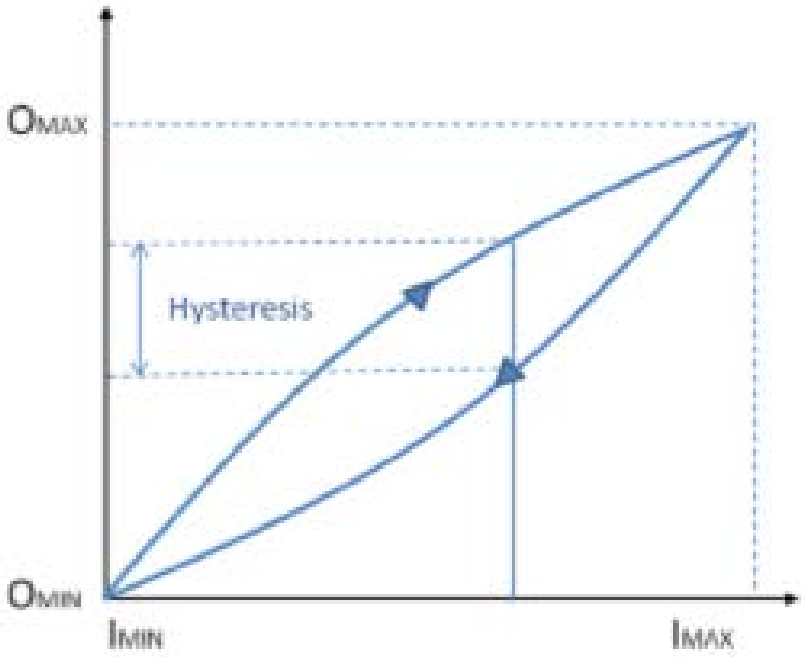


Sensor Characteristics

- Error: the difference between the measured value and true value
- Systematic errors are reproducible inaccuracies that can be corrected with compensation methods
  - Interference errors
  - Operator errors etc
- Random error
  - Noise

Sensor Characteristics

- Hysterysis: The difference in output between the rising and falling output values for a given input



Example: Smoke sensor

- An MQ-2 smoke sensor reports smoke by the voltage level it puts out
- The more smoke there is, the higher the voltage
- Built-in potentiometer for adjusting sensitivity
- Three pins:
  - Vdd input
  - Ground input
  - Analog output

Smoke sensor

```
const int smokePin = 5; //sensor input
void setup()
{
  pinMode(smokePin, INPUT);
  Serial.begin(9600);
}
void loop()
{
int smoke_level = analogRead(smokePin); //read sensor
Serial.println(smoke_level);
  if(smoke_level > 120) {     //calibrate accordingly
    Serial.println("Smoke detected");
  }
  delay(100); // ms
}
```

Using ultrasonic sensor Example

- The “ping” soundpulse is generated when the pingPin level goes HIGH for two microseconds.
- The sensor will then generate a pulse that terminates when the sound returns.
- The width of the pulse is proportional to the distance the sound traveled
- The speed of sound is 340 meters per second, which is 29 microseconds per centimeter.
- The formula for the distance of the round trip is:

  RoundTrip = microseconds / 29

Using ultrasonic sensors Example

```
const int pingPin = 5;
const int ledPin = 13; // pin connected to LED
void setup()
{
Serial.begin(9600);
pinMode(ledPin, OUTPUT);
}
void loop()
{
int cm = ping(pingPin) ;
Serial.println(cm);
digitalWrite(ledPin, HIGH);
delay(cm * 10 ); // each centimeter adds 10 milliseconds delay
digitalWrite(ledPin, LOW);
delay(cm * 10);
}
```

Using ultrasonic sensors Example

```
int ping(int pingPin)
{
long duration, cm; //establish variables for duration of the ping, and the
                   //distance result in centimeters
pinMode(pingPin, OUTPUT);
digitalWrite(pingPin, LOW);
delayMicroseconds(2);
digitalWrite(pingPin, HIGH);
delayMicroseconds(5);
digitalWrite(pingPin, LOW);
pinMode(pingPin, INPUT);
duration = pulseIn(pingPin, HIGH);
// convert the time into a distance
cm = microsecondsToCentimeters(duration);
return cm ;
}
long microsecondsToCentimeters(long microseconds)
{
// The speed of sound is 340 m/s or 29 microseconds per centimeter.
// The ping travels out and back, so to find the distance of the
// object we take half of the distance travelled.
return microseconds / 29 / 2;
}
```

Using Interrupts

- On a standard Arduino board, two pins can be used as interrupts: pins 2 and 3.

- The interrupt is enabled through the following line:

```
attachInterrupt(interrupt, function, mode)
attachInterrupt(0, doEncoder, FALLING);
```

Interrupt example

```
int led = 13;
   volatile int state = LOW;

   void setup()
   {
     pinMode(led, OUTPUT);
     attachInterrupt(1, blink, CHANGE);
   }

   void loop()
   {
     digitalWrite(led, state);
   }

   void blink()
   {
     state = !state;
   }
```

Timer functions (timer.h library)

- `int every(long period, callback)`

Run the 'callback' every 'period' milliseconds. Returns the ID of the timer event.

- `int every(long period, callback, int repeatCount)`

Run the 'callback' every 'period' milliseconds for a total of 'repeatCount' times. Returns the ID of the timer event.

- `int after(long duration, callback)`

Run the 'callback' once after 'period' milliseconds. Returns the ID of the timer event.

Timer functions (timer.h library)

- `int oscillate(int pin, long period, int startingValue)`
  - Toggle the state of the digital output 'pin' every 'period' milliseconds. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.
- `int oscillate(int pin, long period, int startingValue, int repeatCount)`
  - Toggle the state of the digital output 'pin' every 'period' milliseconds 'repeatCount' times. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.

Timer functions (Timer.h library)

- `int pulse(int pin, long period, int startingValue)`
  - Toggle the state of the digital output 'pin' just once after 'period' milliseconds. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.
- `int stop(int id)`
  - Stop the timer event running. Returns the ID of the timer event.
- `int update()`
  - Must be called from 'loop'. This will service all the events associated with the timer.

Timer Library Example

```
#include "Timer.h"
Timer t;
int ledEvent;
void setup()
{
  Serial.begin(9600);
  int tickEvent = t.every(2000, doSomething);
  Serial.print("2 second tick started id=");
  Serial.println(tickEvent);
  pinMode(13, OUTPUT);
  ledEvent = t.oscillate(13, 50, HIGH);
  Serial.print("LED event started id=");
  Serial.println(ledEvent);
  int afterEvent = t.after(10000, doAfter);
  Serial.print("After event started id=");
  Serial.println(afterEvent);
}
```

Timer Library Example

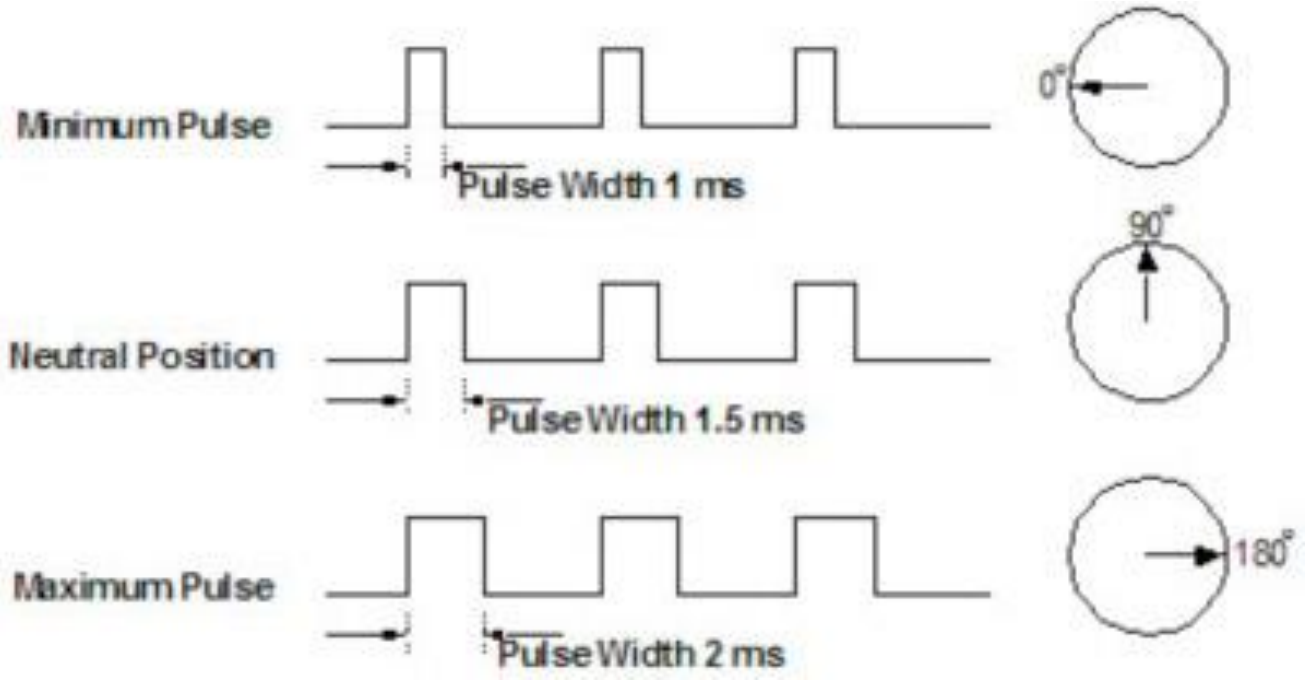
```
void loop()
{
  t.update();
}

void doSomething()
{
  Serial.print("2 second tick: millis()=");
  Serial.println(millis());
}

void doAfter()
{
  Serial.println("stop the led event");
  t.stop(ledEvent);
  t.oscillate(13, 500, HIGH, 5);
}
```

Servo motors

- A DC motor with a control circuit
- Servo motors are controlled by PWM through the control pin

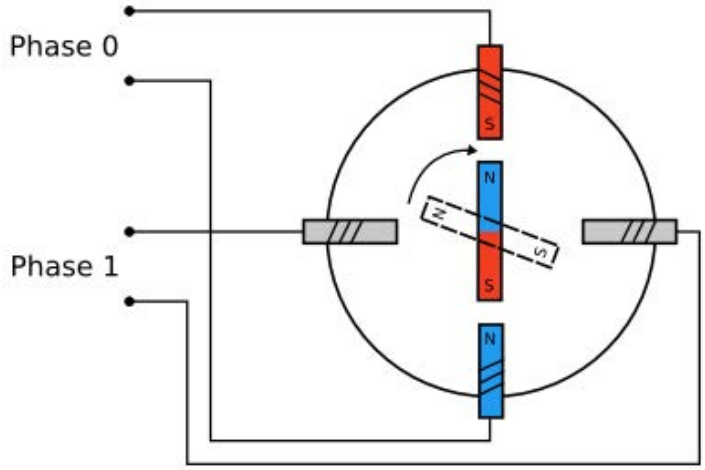


Servo motor control

```
#include <Servo.h>
   Servo myservo;  // create servo object to control a
   servo
   void setup() {
   int val = 180;    // variable to control servo
     myservo.attach(9);  // pin 9 is a PWM pin
   }
   void loop() {
     myservo.write(val);                  // constant servo
   speed
     delay(15);                       // waits for the servo
   to get there
   }
```

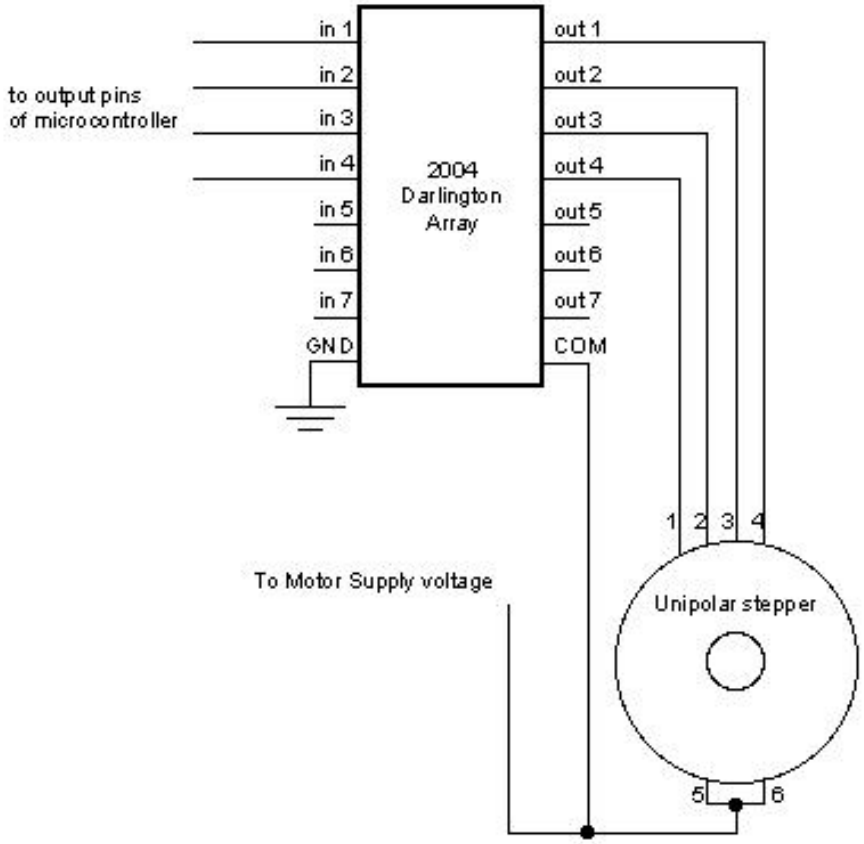
Stepper Motors

- The center shaft has a series of magnets mounted on it
- The coils are alternately given current or not, creating magnetic fields which repulse or attract the magnets on the shaft, causing the motor to rotate.
- Allows for very precise control of the motor. It can be turned in very accurate steps to set degree increments
- two basic types
  - unipolar
  - bipolar



Example: Unipolar stepper motor

- Four digital pins required to control the motor
- Darlington transistor array used for supplying the power required



Stepper motor control

```
#include <Stepper.h> //the control sequence is in this
   library

   const int stepsPerRevolution = 200;  // motor-dependent

   Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
   //pins used

   void setup() {
     // set the speed at 60 rpm:
     myStepper.setSpeed(60);  //actually sets the delay
   between steps
   }

   void loop() {
     // step one revolution  in one direction:
     myStepper.step(stepsPerRevolution);
     delay(500);
}
```