LaACES Student Ballooning Course

Digital I/O

Connecting to the Outside World With Digital Signals

What is Digital I/O?

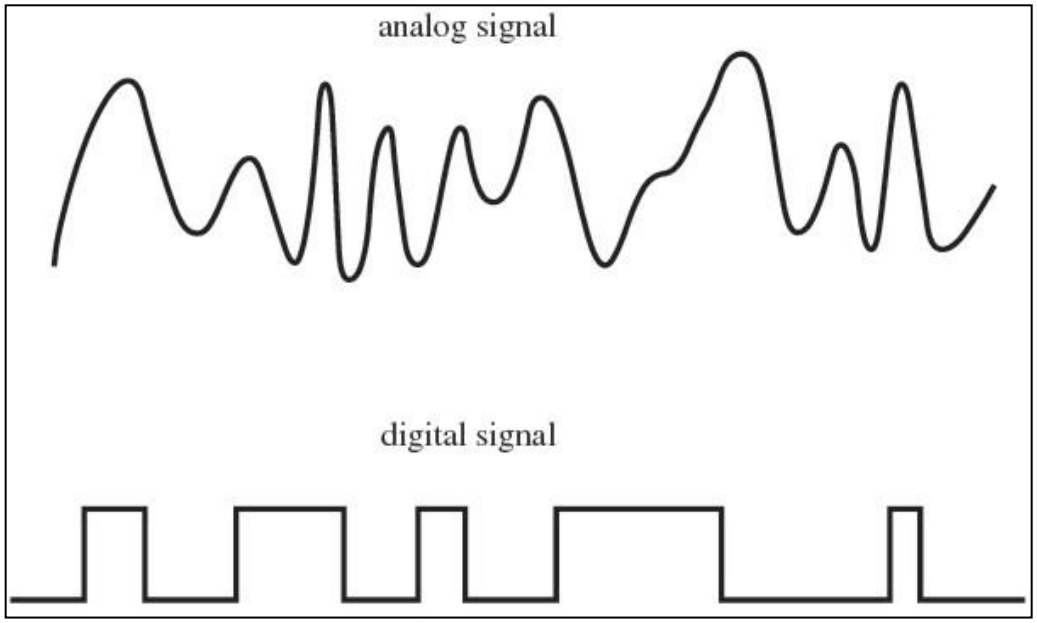
I/O stands for "input & output" and *digital* means the signal wave is square, discontinuous, or stepping. Digital is opposite of *analog* which has a smooth, continuous signal wave. Overall, digital I/O is an essential requirement for most micro-controller applications. Digital I/O is used by the CPU to:

- Activate indicators such as lamps, LEDs, or alarms.
- Control motors, solenoids, or valves.
- Communicate with other external IC's or devices.
- Read or monitor the state of external switches or sensors.
- User interface controls.
- Configuration option switches.

Digital vs Analog

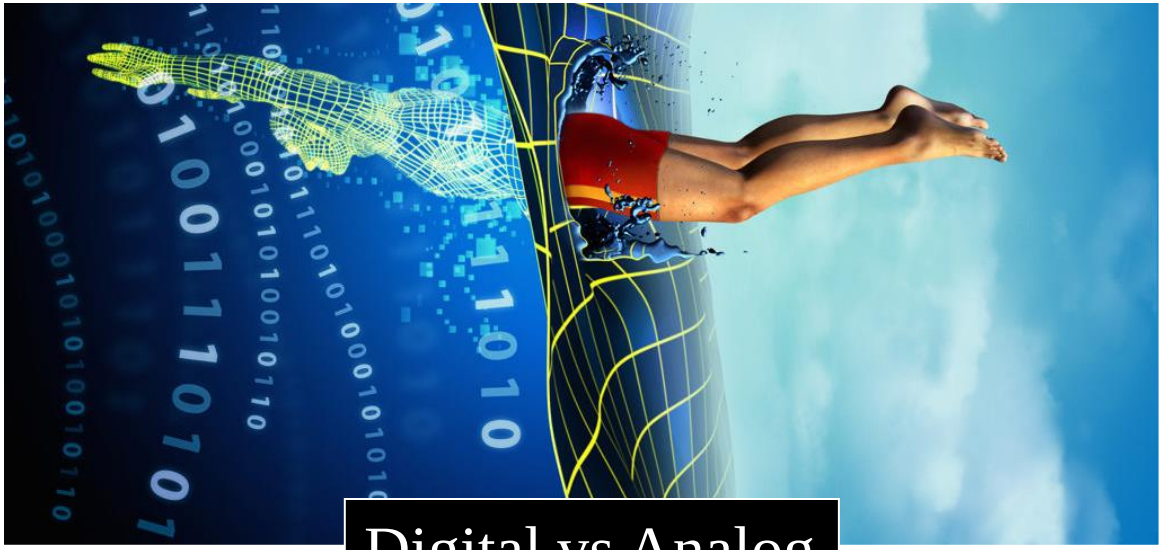
A more in-depth understanding of analog signals will be covered in another lecture, but a visual comparison is given now so one can see the difference.

- Analog is smooth & continuous
- Digital has sharp, sudden changes



We live in an analog world so how do we create digital signals?

- Digital signals are generated by a change in voltage. Digital waves are not smooth because they only have two values, low and high.
- 0 is assigned to the lower voltage and 1 is assigned to the higher voltage. In Boolean Logic they represent true or false.

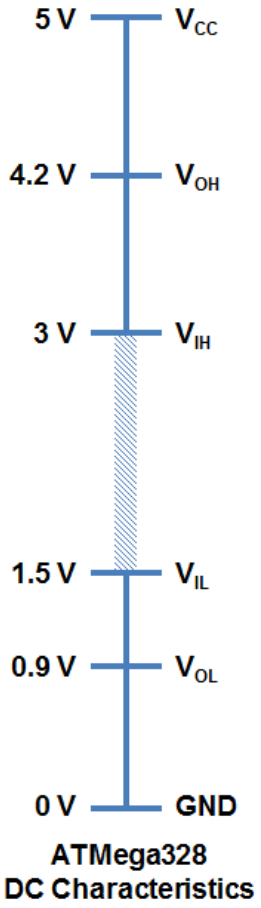


Digital vs Analog

So how do you assign a given voltage a value ?

The people in charge of assigning voltages a value are called *Logic Levels*. Logic Levels decide what voltage levels mean 0 and 1 . Therefore, they give voltage values logical meaning. Hence the name, Logic Level.

To the right is a picture of an Arduino Mega's logic level. Let's break it down a little



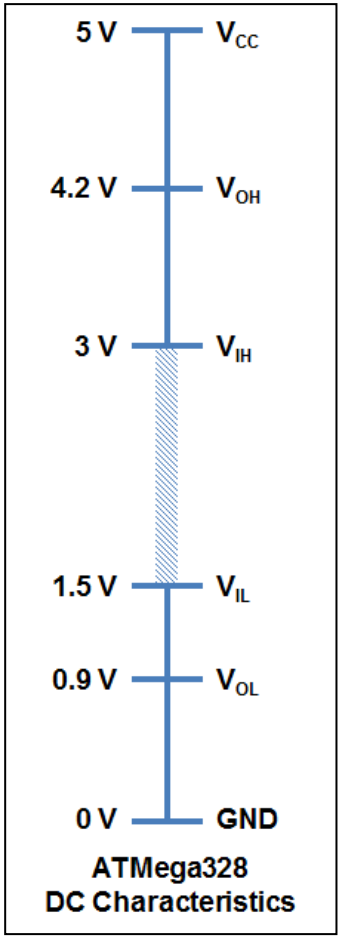
Logic Level

The symbols on the right side mean the fallowing:

- V_{OH} – Minimum OUTPUT Voltage level to be considered a HIGH.
- V_{IH} – Minimum INPUT Voltage level to be considered a HIGH.
- V_{OL} – Maximum OUTPUT Voltage level to be considered a LOW.
- V_{IL} – Maximum INPUT Voltage level to still be considered a LOW.

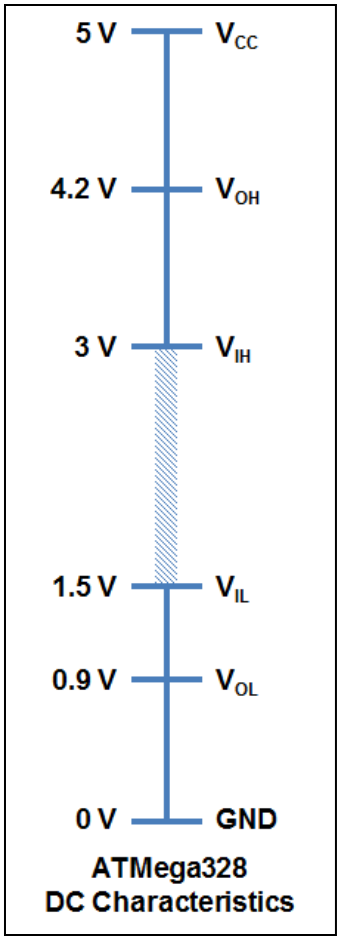
This means for <u>output</u> 5V = 1 and 0.9V = 0

This means for <u>input</u> 4.2V = 1 and 1.5V = 0



Logic Level

- Notice the gap between V_{IH} and V_{IL} ? Because of this gap there is a sharp distinction between 0 and 1. Therefor digital signals are less susceptible to errors caused by "*background noise"* which would normally affect an analog signal.
- Also, you may hear a person point to a component on a circuit board and call it "a logic level". This is colloquial speech for the part of the board that performs the logic level tasks



Types of Digital I/O

Now that you have an understanding of what a digital I/O is, let's look at a few types of digital I/O signals. The type of signal used can depend upon what type of current the component uses.

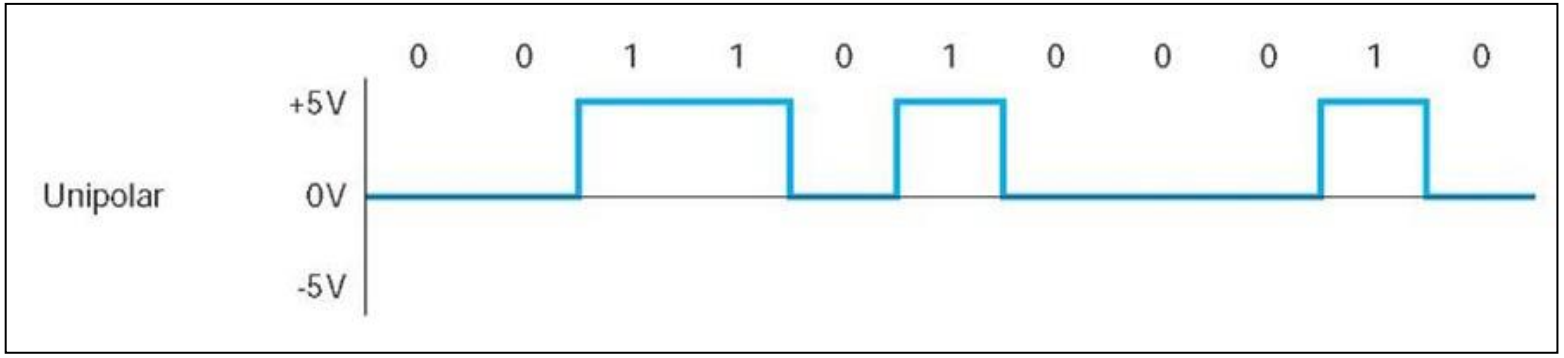
There are two types of current. First, there is direct current (DC). In DC, the flow of electricity flows in one direction. The other type of electrical current is alternating current (AC). In AC, the flow of electricity reverses direction periodically.

- DC components use a *unipolar* type signal.
- AC components use a *bipolar* signal.
- There is a third type of digital signal called *Pulse Width Modulation* (PWM) which will be covered last

Types of Digital I/O

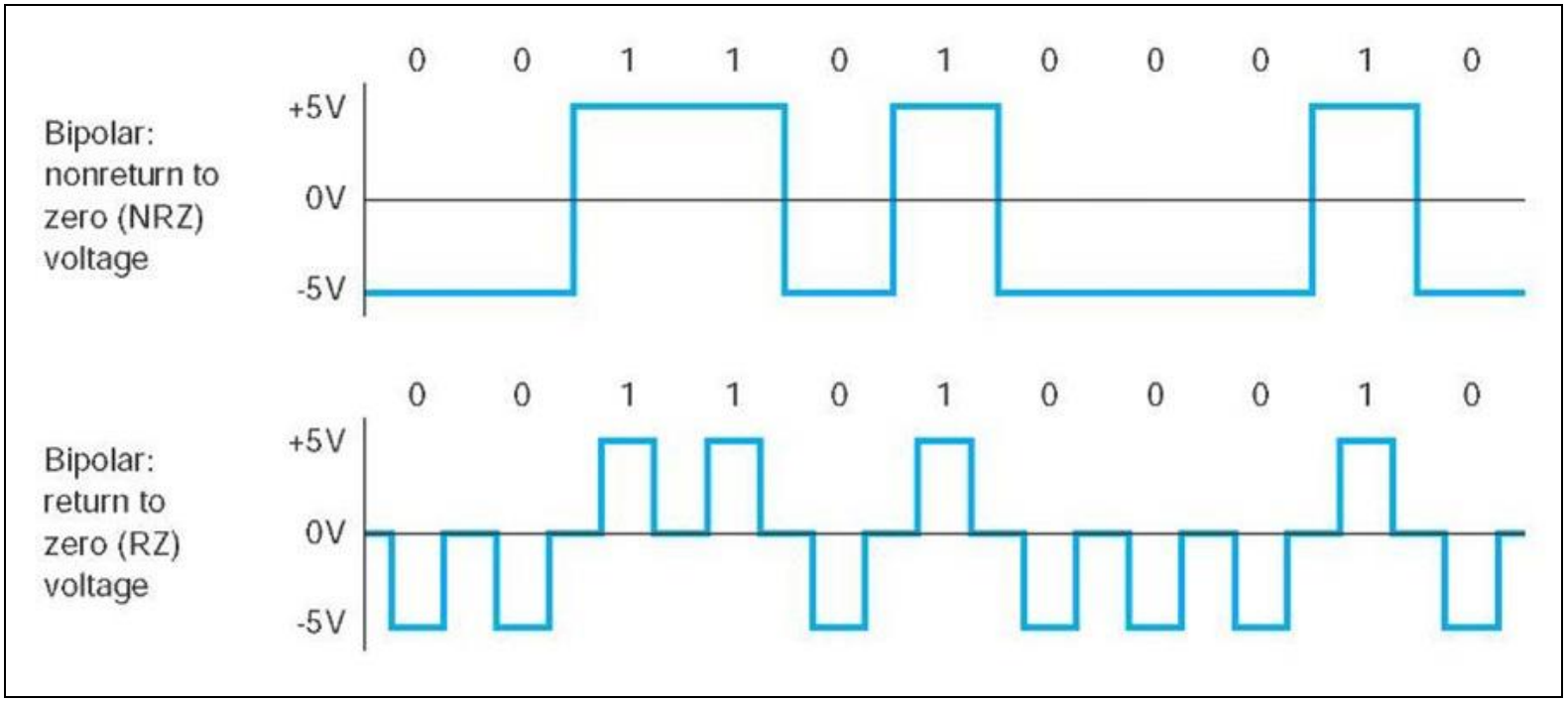
In unipolar, low is assigned to 0 voltage, a.k.a ground. The high is assigned to the output voltage.

It is a logical question to ask how one sends data when there is zero voltage. The answer is the lack of a voltage is a signal in itself, and therefor the lack of voltage is just as important as the emission of a voltage.



Types of Digital I/O

Digital AC components use 1 of 2 types of *bipolar* signals. One *skips over* 0 volts while the other doesn't. Note the negative voltage represents the reverse of the current.



Pulse Width Modulation (PWM)

PWM is a method of generating an analog-like signal using digital signals. This new voltage can be used in a variety of way. A few examples are provided below:

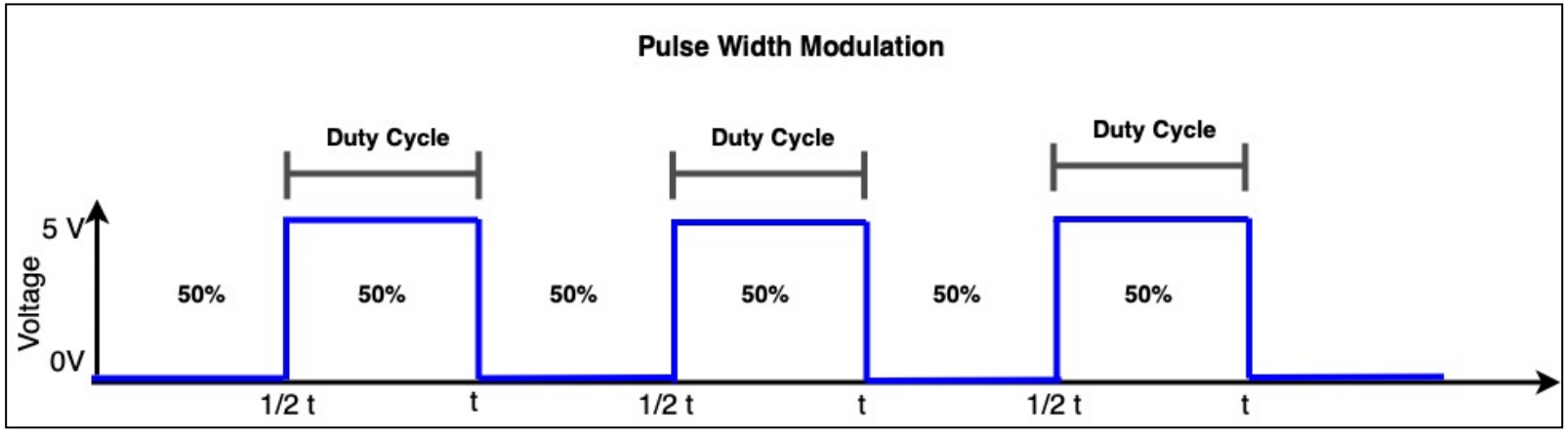
- Controlling an LED's luminescence
- Regulate a DC motor
- Communication in sophisticated circuitry
- Control the direction of a servo

How PWM Works

In PWM you have a signal that repeats at some predefined period.

Over the that period that signal can only take values of on or off just like any digital signal.

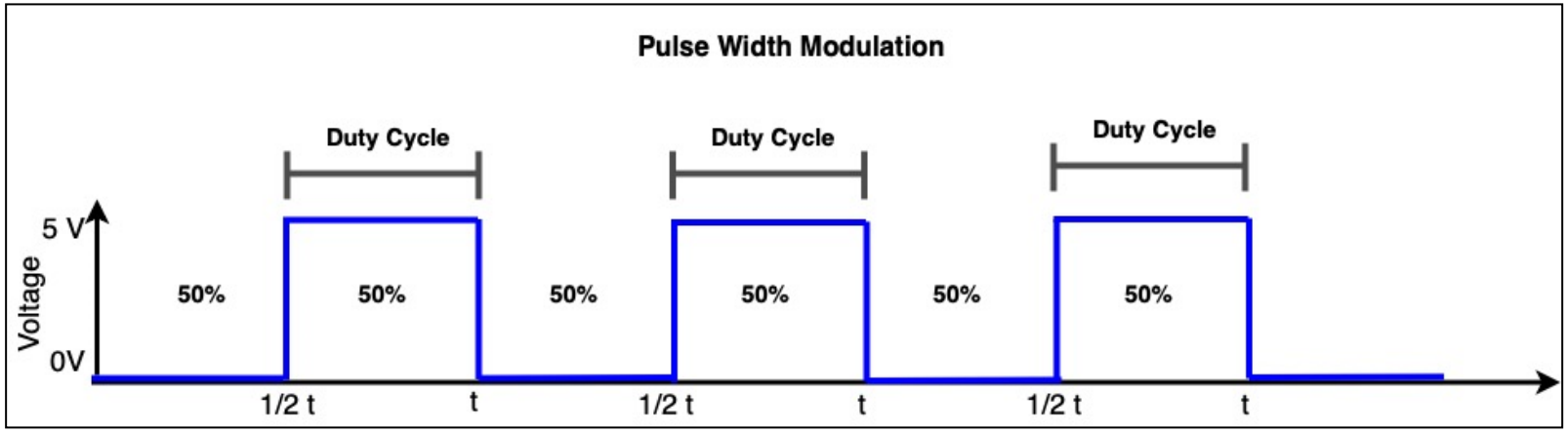
What is varied is the amount of time that the signal is on or off in an individual period.



How PWM Works

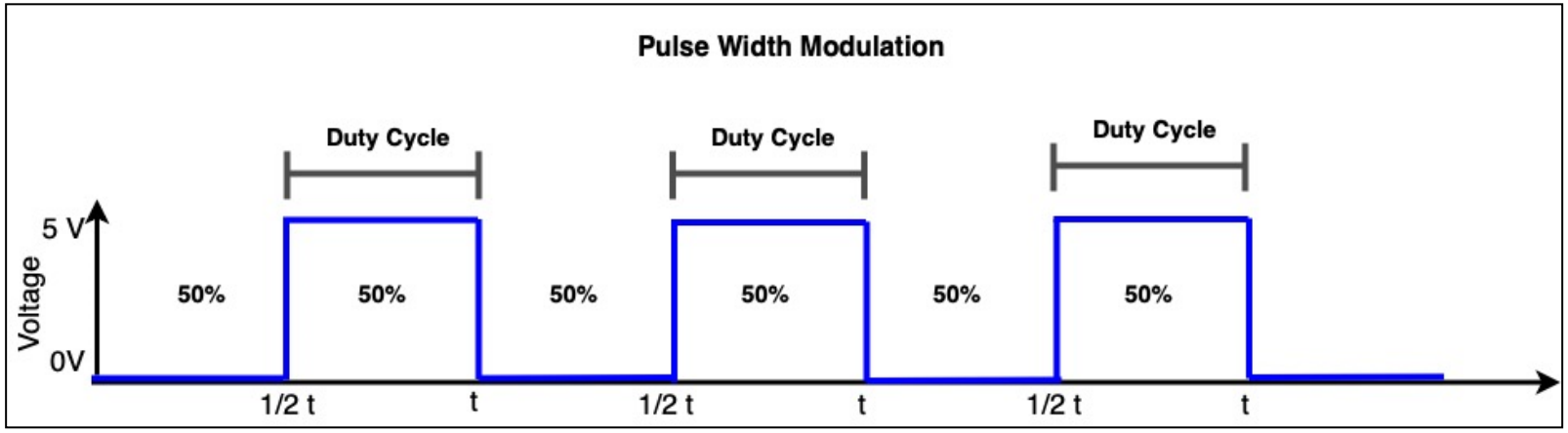
Say you have a potential of 5 volts(V), and this potential is available for a preset amount of time called " ***t*** ". Now say you only want 5V for 50% of ***t*** to control the brightness of a light. This is accomplished by sending a signal of 5V for 50% of ***t*** and a signal of 0V for the remainder of ***t*** .

The result of this repeated signal change looks like the graph below. What you get is a *square wave* and the high voltage signal time is called the *duty cycle.*



How PWM Works

- The average voltage received is 2.5V, because 5V x 0.5 = 2.5
- By changing the duty cycle we can get an average value (***f***) of the square wave from 0 to 5V.
- The light's brightness is controlled by ***f*** , and ***f*** is dependent on the duty cycle so to alter the brightness of the light you change the duty cycle.



Let's take a step back

Remember a whole 5 slides ago when I told you that the lack of a voltage is just as important as the presence of voltage?

Another question, are unused pins on the Arduino Mega sending information to the CPU?

The answer is yes, they are sending *rogue* zeros! How are we going to fix this?

Here is another thing to ponder, do you think the voltage being supplied is always a perfect 5V?

The answer is no, in the real-world things are never perfect.

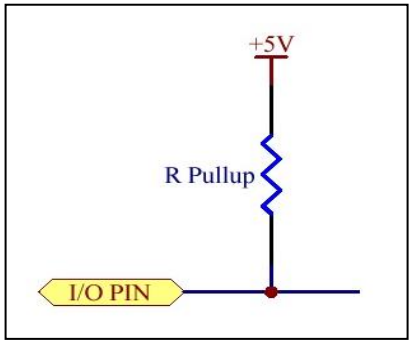
So, what is there to stop a *rogue* 1 or 0 being generated by a not so perfect voltage?

Resistors to the Rescue

We want to ensure there is always a definite know signal connected to digital inputs. This is accomplished by connected the pin to the high or low value via resistors. These resistors are call *Pull Up* and *Pull Down* resistors.

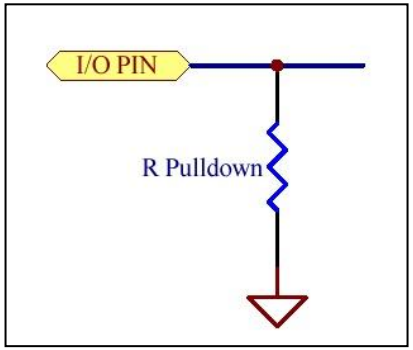
- Pull Up resistors:

They are placed along the pathways on the circuit board to "pull up" the voltage to the recognized 1 value.



- Pull Down resistors:

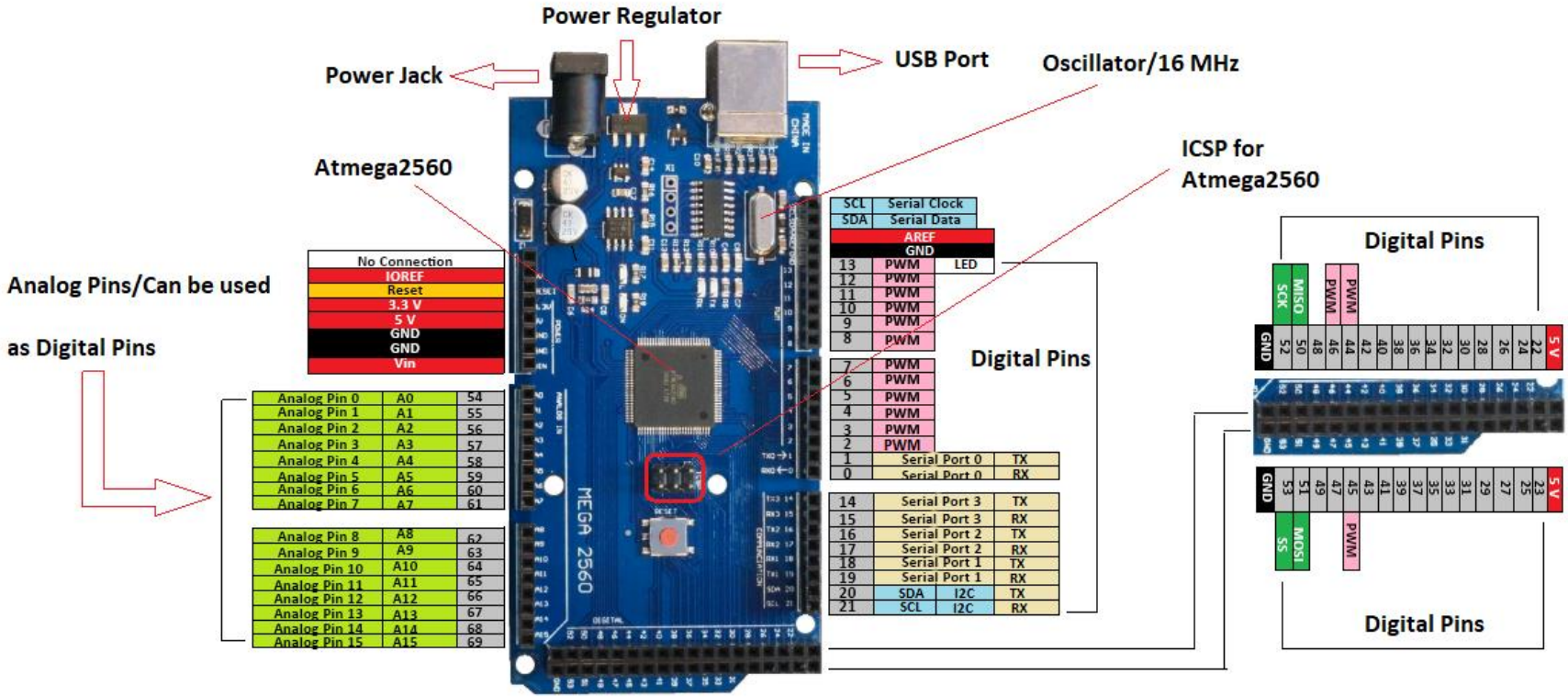
They are placed along the pathways on the circuit board to "pull down" the voltage to the recognized 0 value.



Pull Up and Pull Down Resistors

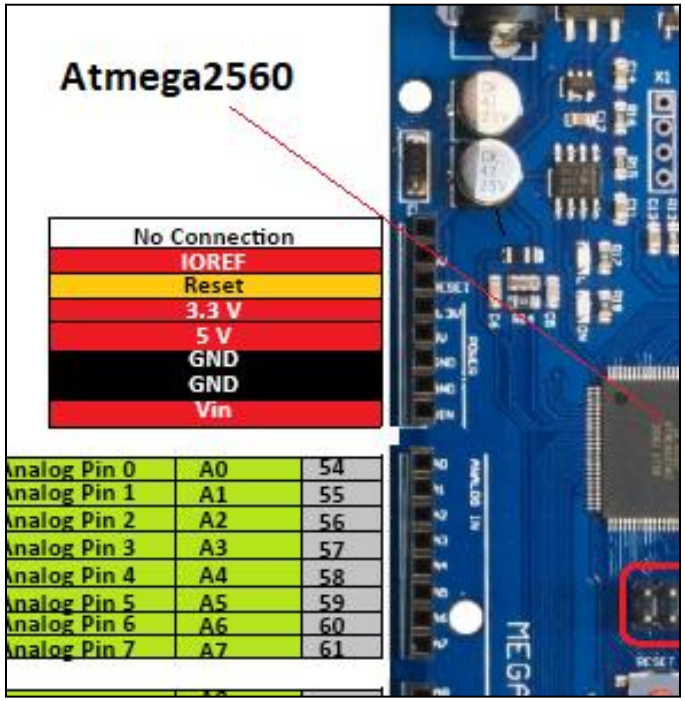
- Pins can be in an unpredictable HIGH or LOW state when charged and are discharged by leakage paths on the printed circuit board.
- Many misbehaving circuits or micro-controller programs can be traced to unconnected input pins.
- All inputs should be properly terminated with a high value resistor to either VCC (a pull up) or GND (pull down). Termination to VCC with a pull up resistor is usually the preferred method.
- The resistance value for these resistors is not usually critical. 10k ohms is common but can be from 1k to 100k ohms.

Arduino Mega Pin Layout



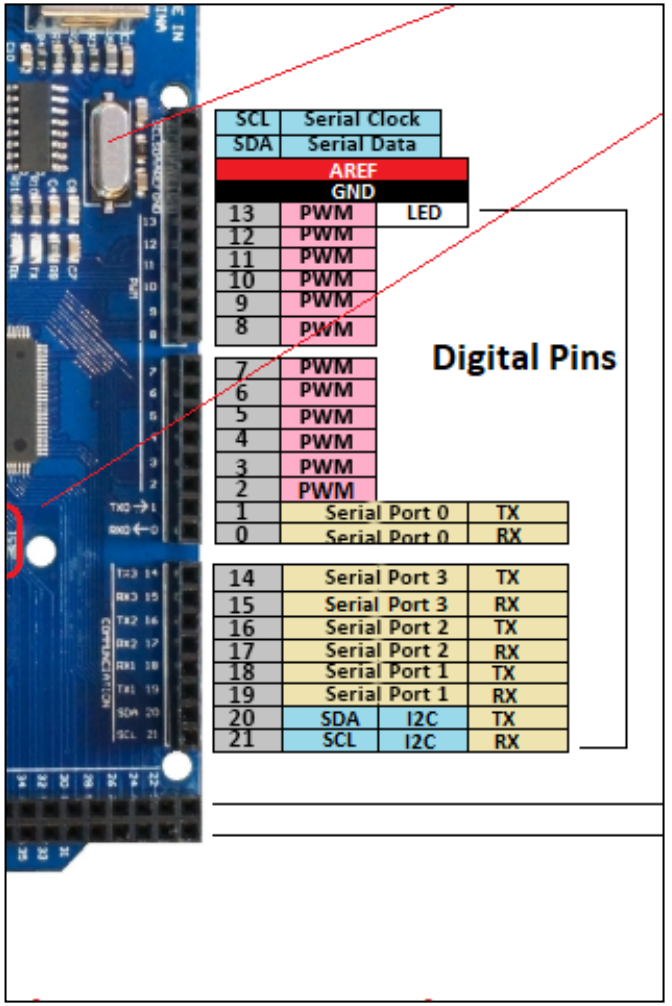
Lets talk Pins

- GND is ground, there are 5 GND pins on the board.
- Vin is the input voltage
- 3.3V & 5V regulate their respective output voltages.
- *Reset* resets the board when set to LOW
- IOREF is the voltage corresponding to the I/O of the board.



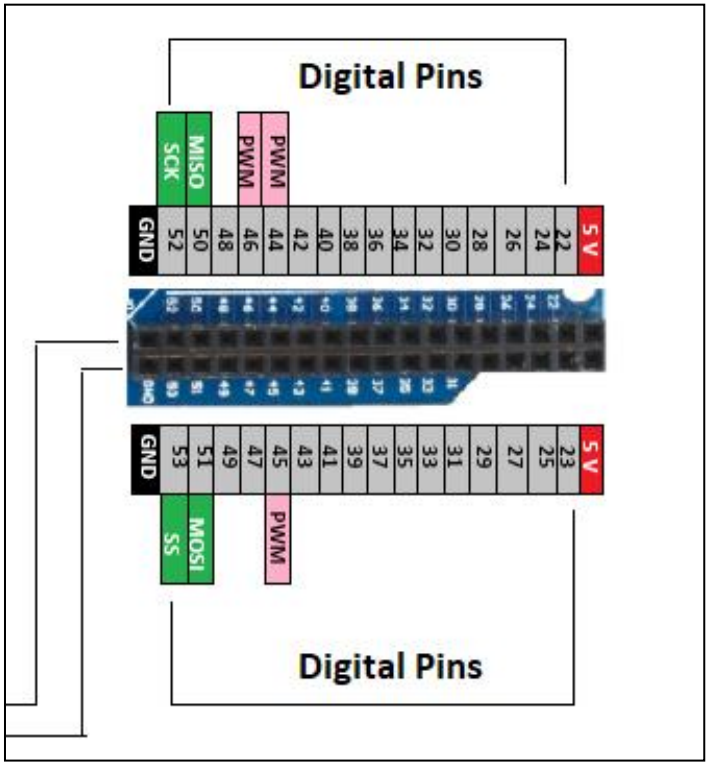
Let's talk Pins

- Pins 2-13 are PWM
- Pins 0-1, 14-15 are serial ports
  - TX and RX mean transmit and receive respectively
  - *Serial* means sending 1 bit at a time
- Pins 20-21 support I2C
  - I2C is a type of communication between two devices where one is the “master” and the other is the “slave” .
- SCL synchronize the master’s and slave’s clocks for data transfer
- SDA is the actual data sharing between the devices.

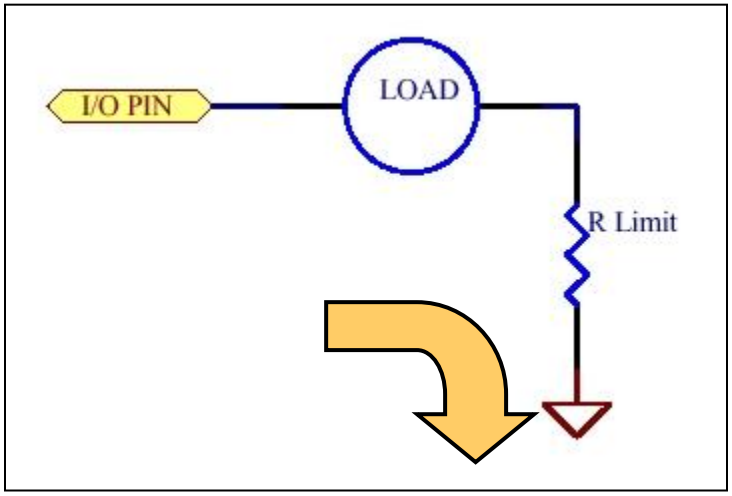


Lets talk Pins

- Pins 44-46 are PWM
- MISO and MOSI stand for *Master-In Slave-Out* and *Master-Out Slave-In* respectively
- SCK is the serial clock for data transmission by the master
- SS is *Slave Select* which allows the master to select a slave.



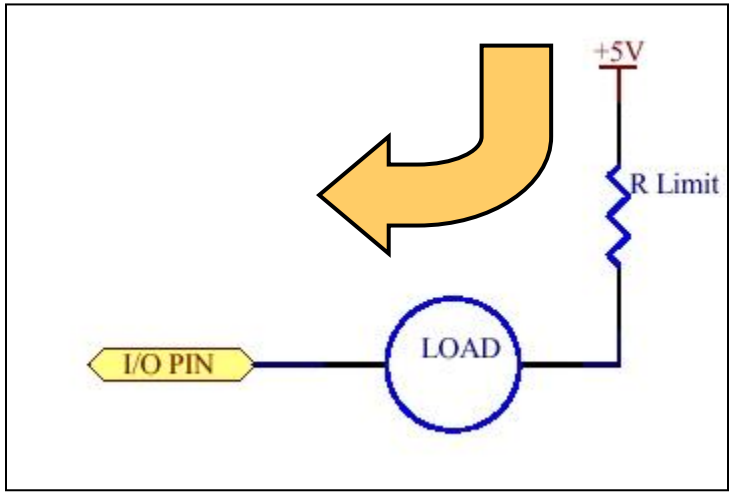
I/O pin as a Current Source



The current limiting resistor may be needed in some applications.

- *Source* in current source refers to the pin's ability to **supply** current
- When a pin is HIGH, current flows from the pin through the load to GND.
- When a pin is LOW, no potential difference between pin and GND, no current flows.
- The instruction **HIGH pin** would activate the load.
- The instruction **LOW pin** turns the load off.

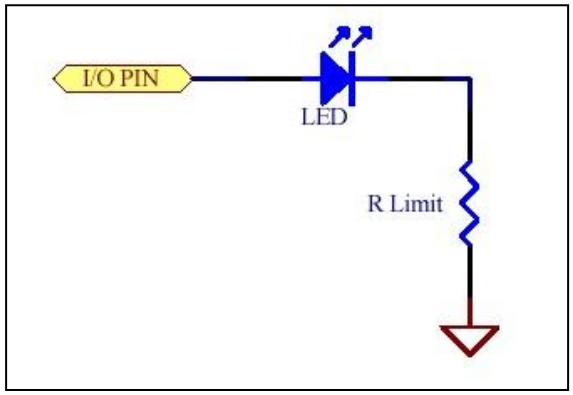
I/O Pin as a Current Sink



The current limiting resistor may be needed in some applications.

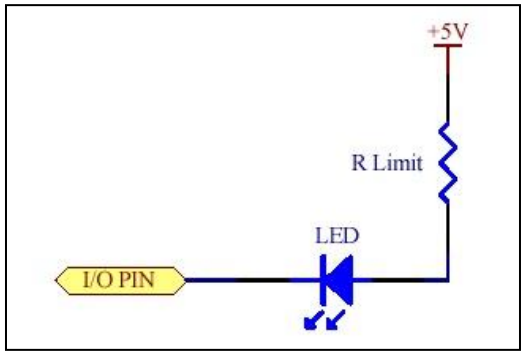
- *Sink* in current sink refers to the pin's ability to **receive** current
- When pin is LOW, current flows from the +5V power supply through the load to I/O pin.
- When pin is HIGH, no potential difference between pin and +5V, no current flows.
- The instruction **LOW pin** would activate the load.
- The instruction **HIGH pin** turns the load off.

Digital Output Applications



Driving an LED with I/O pin as a current source.

HIGH pin turns LED on.



Driving an LED with I/O pin as a current sink.

LOW pin turns LED on.

Configuring Pins

- The digital pins on the Arduino Mega are set to input as default. They are limited to 40 Milliamps (mA)
- From the programing lecture you should recall functions. To configure digital pins we have 3 different functions. All 3 functions return nothing to the system:
  - *pinMode()* sets the pin as an INPUT or OUTPUT
  - *digitalWrite()* is used designate OUTPUT pins as HIGH or LOW
  - *digitalRead()* is used designate INPUT pins as HIGH or LOW

Configuring Pins Examples

Here is an example of setting pin 13 as an OUTPUT, and then "turns on" the pin for 1 second and then turns it off.

```
void setup()
{
  pinMode(13, OUTPUT);           // sets the digital pin 13 as output
}

void loop()
{
  digitalWrite(13, HIGH);        // sets the digital pin 13 on
  delay(1000);                   // waits for a second
  digitalWrite(13, LOW);         // sets the digital pin 13 off
  delay(1000);                   // waits for a second
}
```

Setting up PWM Pins

To setup a PWM pin we use the function *analogWrite()*

Using this function has nothing to do with analog I/O, and you do not need to use *pinMode()* before using *analogWrite().*

The parameters used are *analogWrite(pin, value)*, where value equals the duty cycle. The duty cycle can range from 0 (always off) to 255 (always on)

Note:

- This is limited to pins 2-13
- Due to the board's construction, pins 5 & 6 will have higher than expected duty cycles