

<https://www.halvorsen.blog>



# LabVIEW LINX and Arduino

LabVIEW + LabVIEW LINX Toolkit + Arduino

Hans-Petter Halvorsen

# Contents

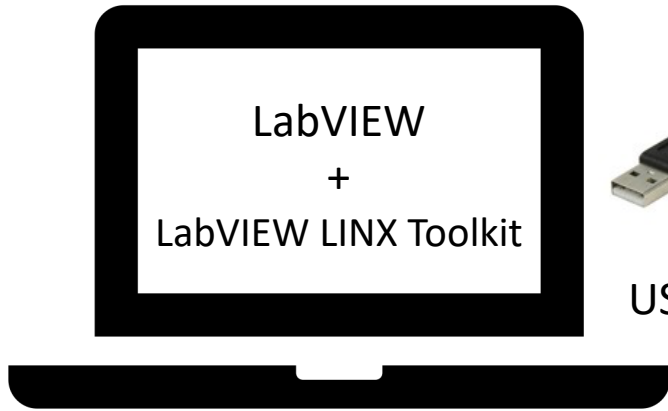
- This Tutorial shows how we can use Arduino in combination with the LabVIEW Programming environment
- “LabVIEW LINX Toolkit” is an add-on for LabVIEW which makes it possible to program the Arduino device using LabVIEW
- In that way we can create Data Logging Applications, etc. without the need of an expensive DAQ device
- If you don't have “LabVIEW Professional” Software, you may use the “LabVIEW Community Edition” (free for non-commercial use). You then get a very low-cost DAQ/Datalogging System!

# Table of Contents

- Introduction to LabVIEW LINX
- DAQ System
- Input/Output Channels
  - Digital I/O
    - Digital Out/Write
    - Digital In/Read
  - Analog I/O
    - Analog Out/Write -> PWM (Pulse Width Modulation)
    - Analog In/Read
- Sensors
  - TMP36 Temperature Sensor
  - Thermistor Temperature Sensor

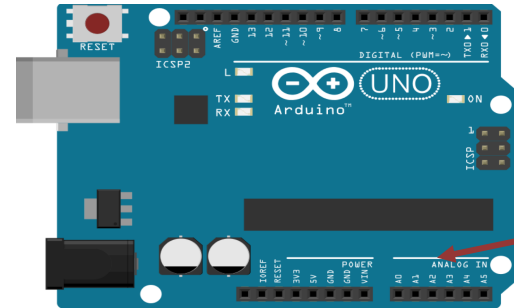
# LabVIEW + LabVIEW LINX Toolkit

PC

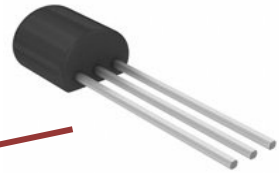


USB cable Type A-B

Arduino UNO



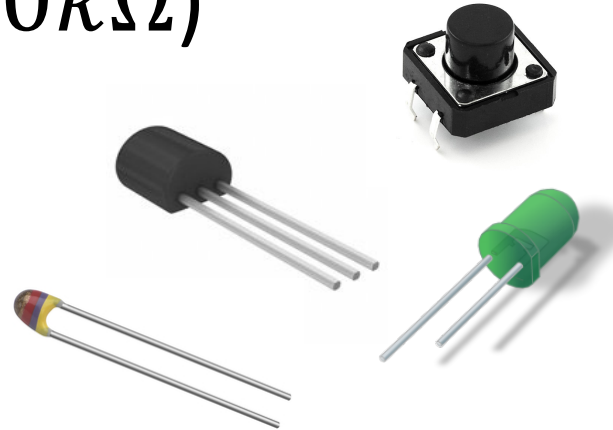
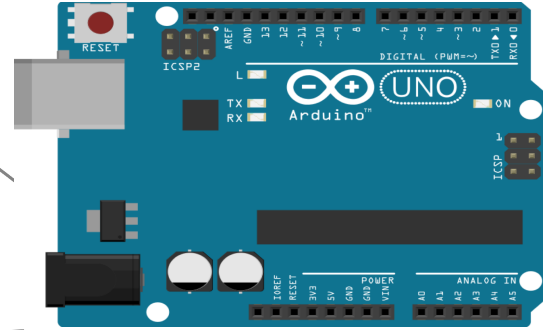
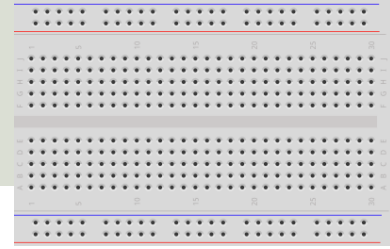
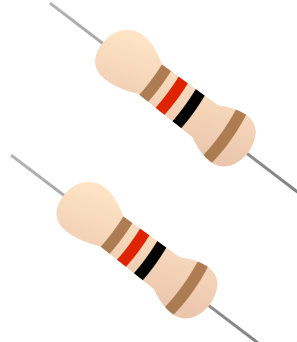
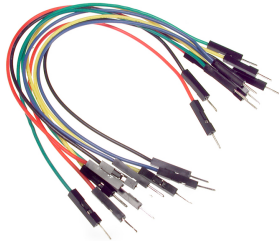
Sensors



TMP36  
Temperature  
Sensor

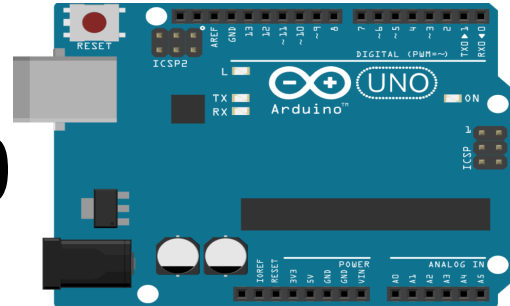
# Hardware

- Arduino
- Breadboard
- Wires (Jumper Wires)
- Resistors ( $R = 270\Omega$ ,  $R = 10k\Omega$ )
- LED, Push Button
- TMP36 Temperature Sensor
- Thermistor



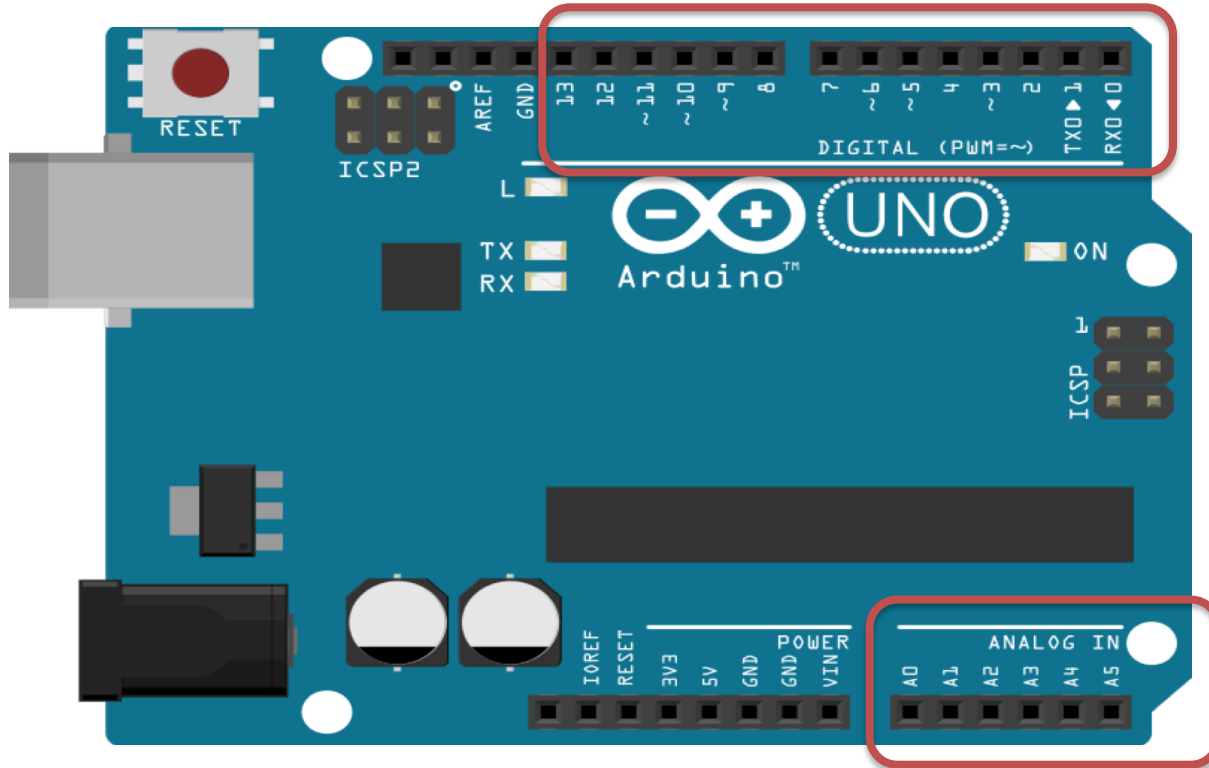
# Arduino UNO

- Arduino is a Microcontroller
- Arduino is an open-source platform with Input/Output Pins (Digital In/Out, Analog In and PWM)
- Price about \$20
- Arduino Starter Kit ~\$40-80  
with Cables, Wires, Resistors, Sensors, etc.



# Arduino I/O Channels

## Digital Inputs and Digital Outputs



You can choose from the code if they are to be inputs or outputs

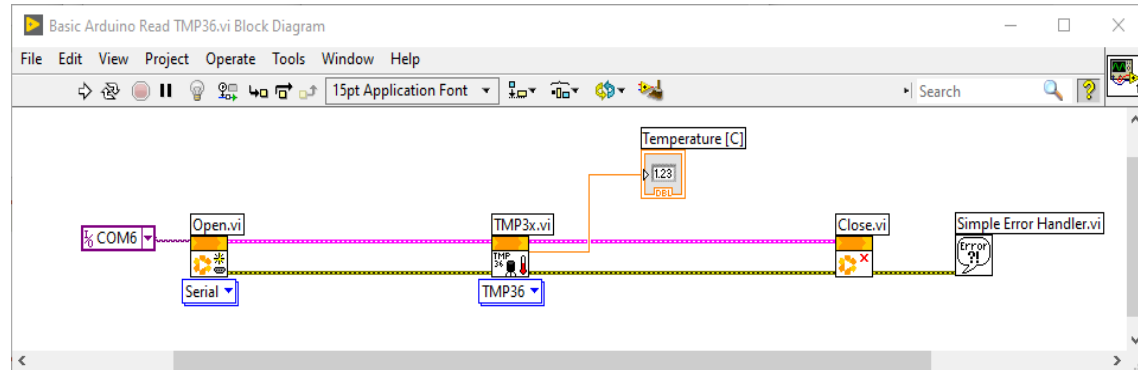
Those marked with ~ can also be used as "Analog Outputs", so-called PWM outputs

## Analog Inputs

# LabVIEW

- LabVIEW is Graphical Software
- LabVIEW has powerful features for simulation, control and DAQ applications

Basic LabVIEW Example:

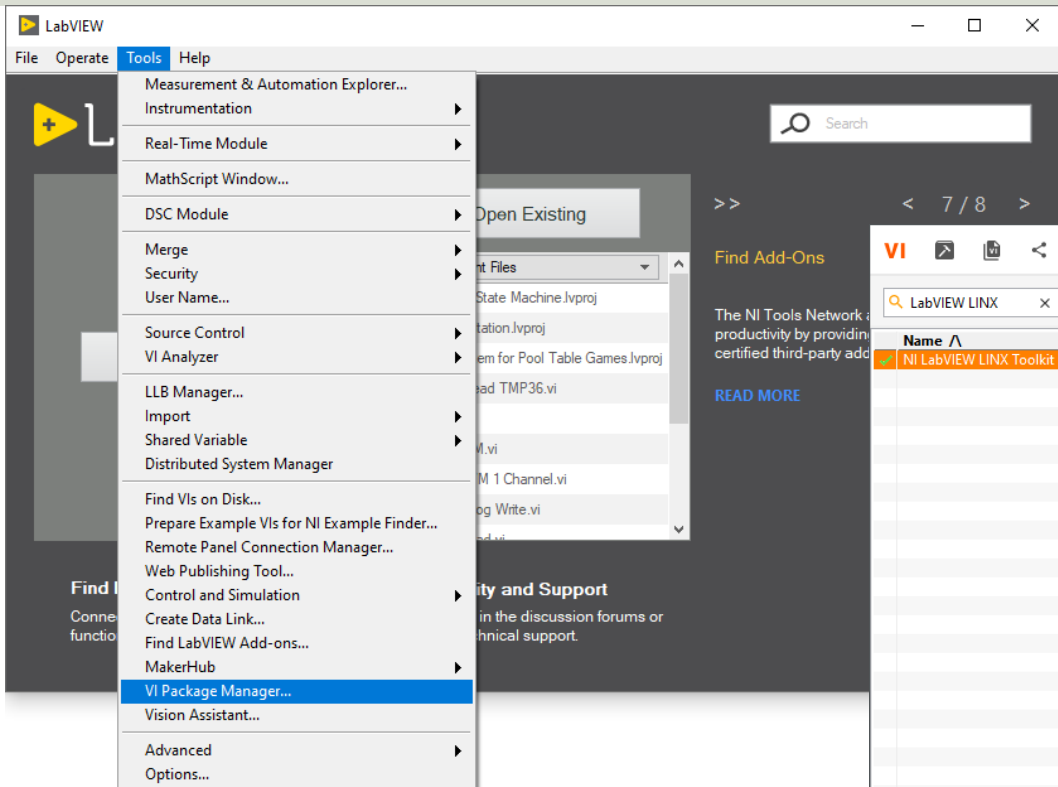




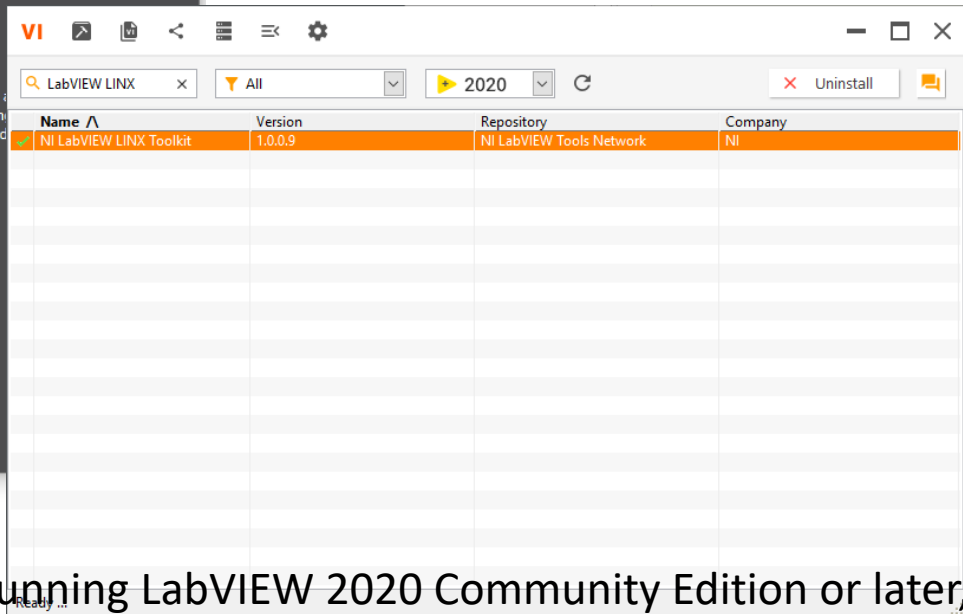
# LabVIEW LINX Toolkit

- The LabVIEW LINX Toolkit adds support for Arduino, Raspberry Pi, and BeagleBone embedded platforms
- We will use Arduino Uno in this Tutorial

# Installing LabVIEW LINX Toolkit

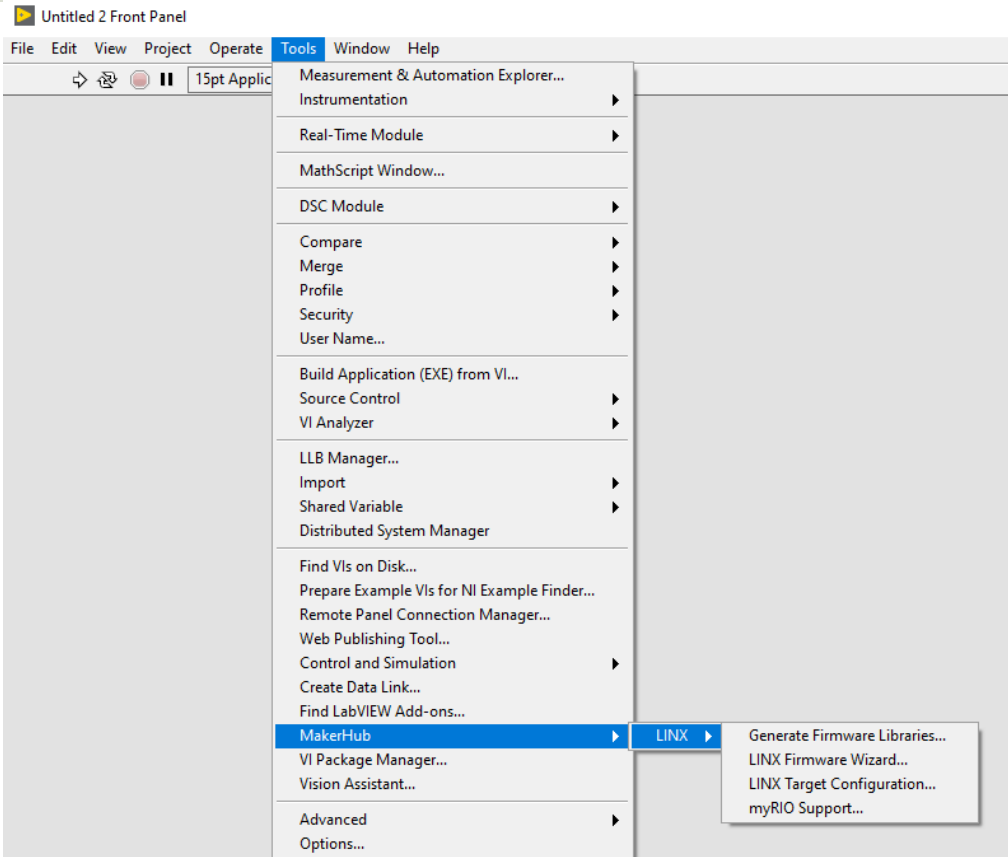


Use VI Package Manger



Note: Do not install this package if you are running LabVIEW 2020 Community Edition or later, as the Community Edition already includes the LabVIEW LINX Toolkit

# LabVIEW LINX



# LINX Firmware Wizard

LINX Firmware Wizard

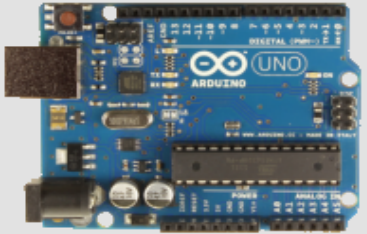
LabVIEW  
MakerHub

**LINX  
Firmware Wizard**

**Device Family**  
Arduino

**Device Type**  
Arduino Uno

**Firmware Upload Method**  
Serial / USB



Help Settings Next Cancel

# LabVIEW Palette

Sensors

↑ Search Customize

Accelerometer Beta Community

Display Distance Digilent

Lights Mindstorms Misc

Motion Pmods Temp

Sig Gen

Detailed description: This is the Sensors palette in LabVIEW. It features a search bar and a 'Customize' button. The palette contains 13 icons arranged in a grid. The first row includes Accelerometer, Beta, and Community. The second row includes Display, Distance, and Digilent. The third row includes Lights, Mindstorms, and Misc. The fourth row includes Motion, Pmods, and Temp. The fifth row includes Sig Gen.

LINX

↑ Search Customize

Open Close Peripherals

Sensors Utilities

Detailed description: This is the LINX palette in LabVIEW. It features a search bar and a 'Customize' button. The palette contains 5 icons arranged in a grid. The first row includes Open, Close, and Peripherals. The second row includes Sensors and Utilities.

Peripherals

↑ Search Customize

Analog Digital PWM

I2C SPI UART

Detailed description: This is the Peripherals palette in LabVIEW. It features a search bar and a 'Customize' button. The palette contains 6 icons arranged in a grid. The first row includes Analog, Digital, and PWM. The second row includes I2C, SPI, and UART.

Utilities

↑ Search Customize

Custom CMD Loop Freq

Check Channel Get User ID Set User ID

Config Enet Config Wifi

Detailed description: This is the Utilities palette in LabVIEW. It features a search bar and a 'Customize' button. The palette contains 7 icons arranged in a grid. The first row includes Custom CMD and Loop Freq. The second row includes Check Channel, Get User ID, and Set User ID. The third row includes Config Enet and Config Wifi.



# DAQ System

# DAQ System

DAQ – Data Acquisition

## Input/Output Signals

Analog Signals



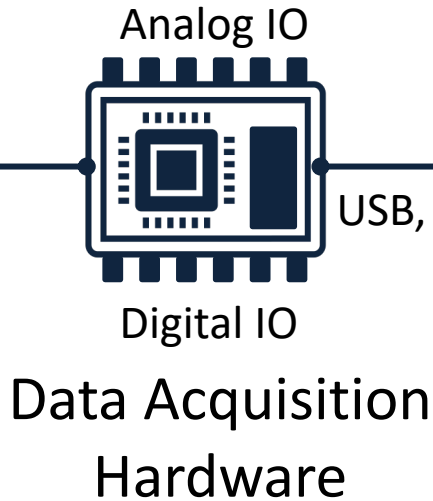
Digital Signals



Sensors



(Analog/Digital Interface)



USB, etc.



Software



We will use an Arduino Uno as the DAQ Hardware

# I/O Module



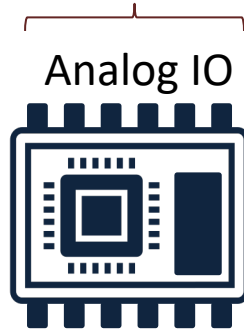
Analog Sensors

Analog Signals



0 – 5V

I/O Module



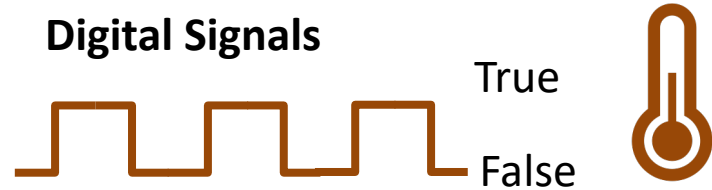
Analog Input (AI)

Analog Output (AO)

Digital Input (DI)

Digital Output (DO)

Digital Signals



Sensors with Digital Interface (e.g., SPI, I2C)





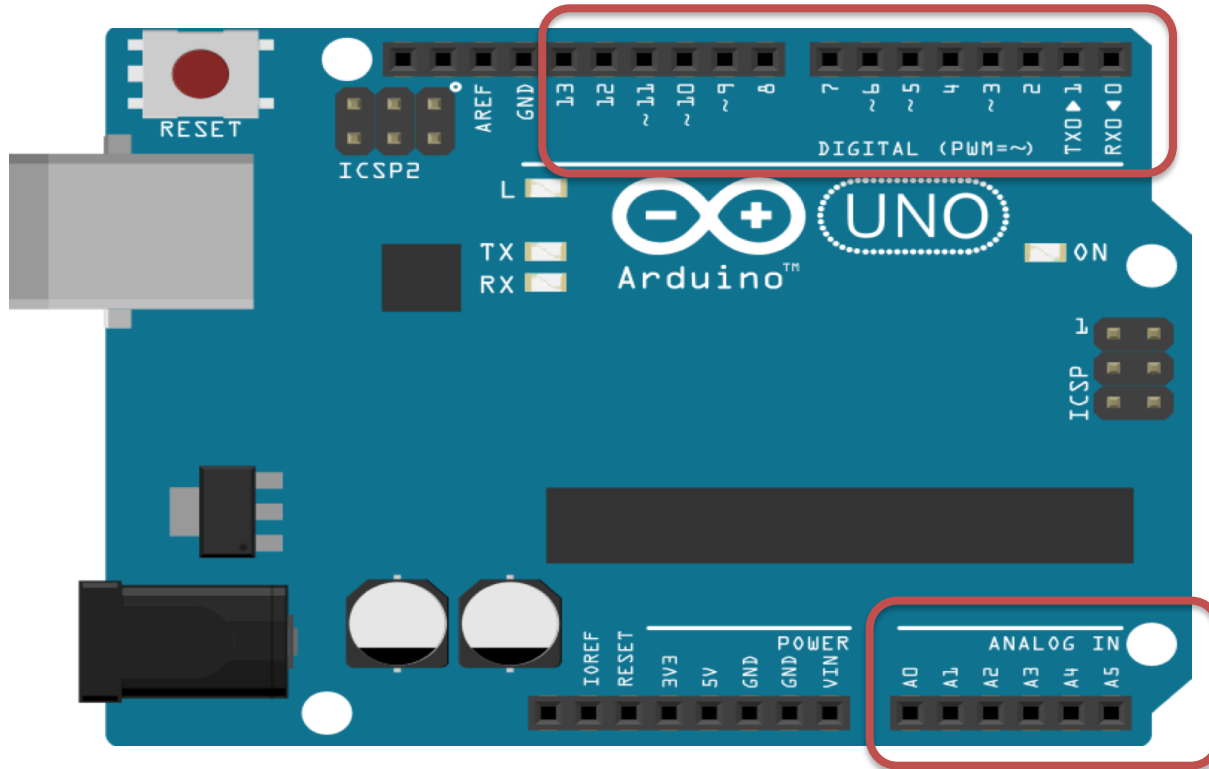
# LabVIEW LINX Arduino DAQ System



# Input/Output Channels

# Arduino I/O Channels

## Digital Inputs and Digital Outputs



You can choose from the code if they are to be inputs or outputs

Those marked with ~ can also be used as "Analog Outputs", so-called PWM outputs

## Analog Inputs

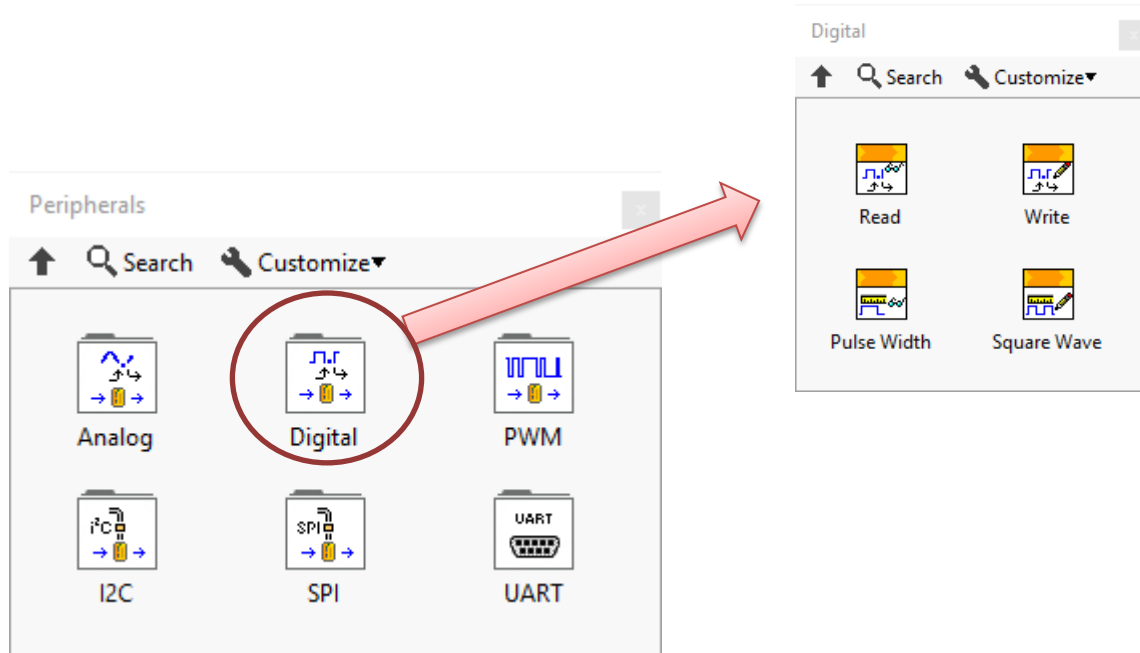
# Input/Output Channels

- Digital
  - Digital Out
  - Digital In
- Analog
  - Analog Out
  - Analog In



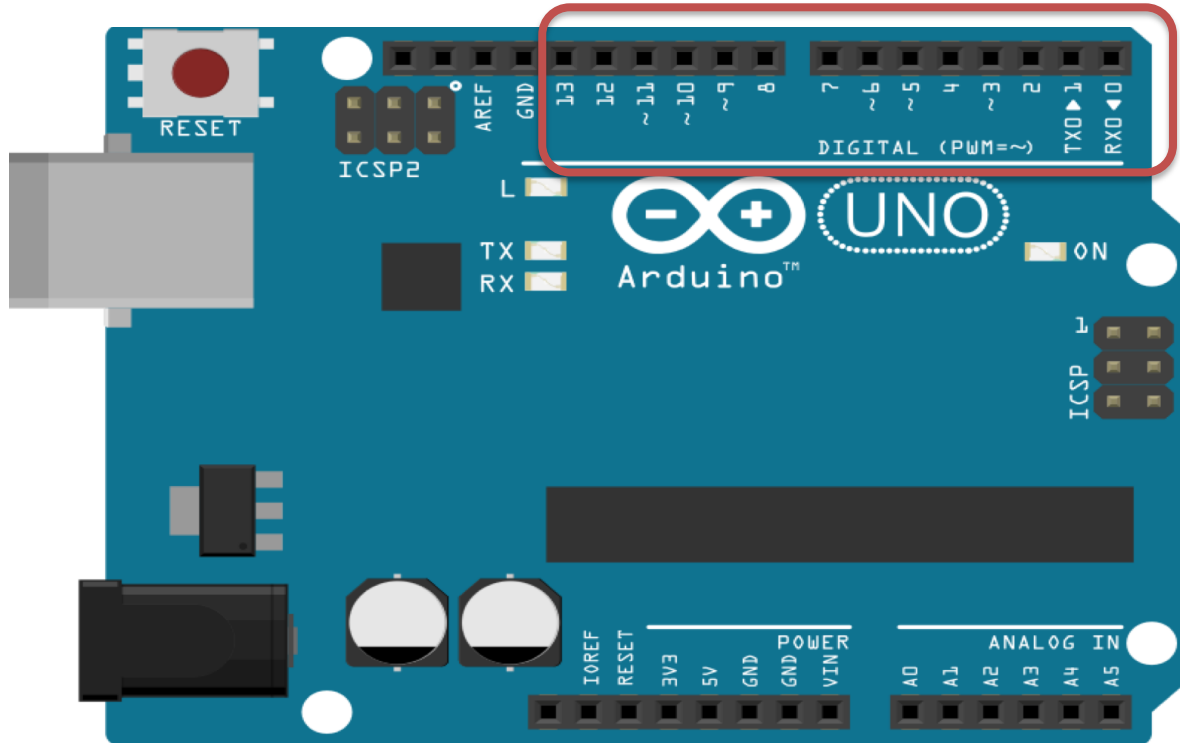
# Digital I/O

# LabVIEW Palette – Digital I/O



# Digital I/O

## Digital Inputs and Digital Outputs



You can choose from the code if they are to be inputs or outputs

Those marked with ~ can also be used as "Analog Outputs", so-called PWM outputs

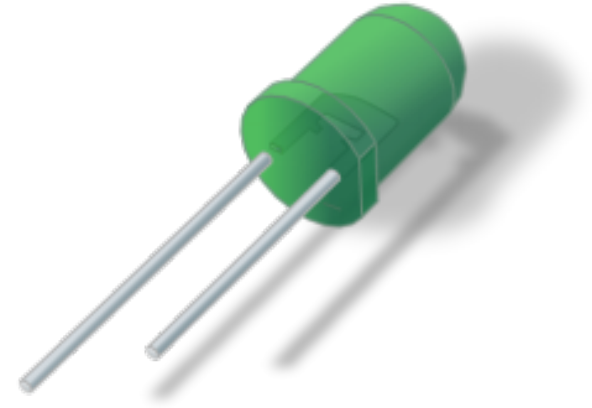


# Digital Out (DO)



# Digital Out (DO)

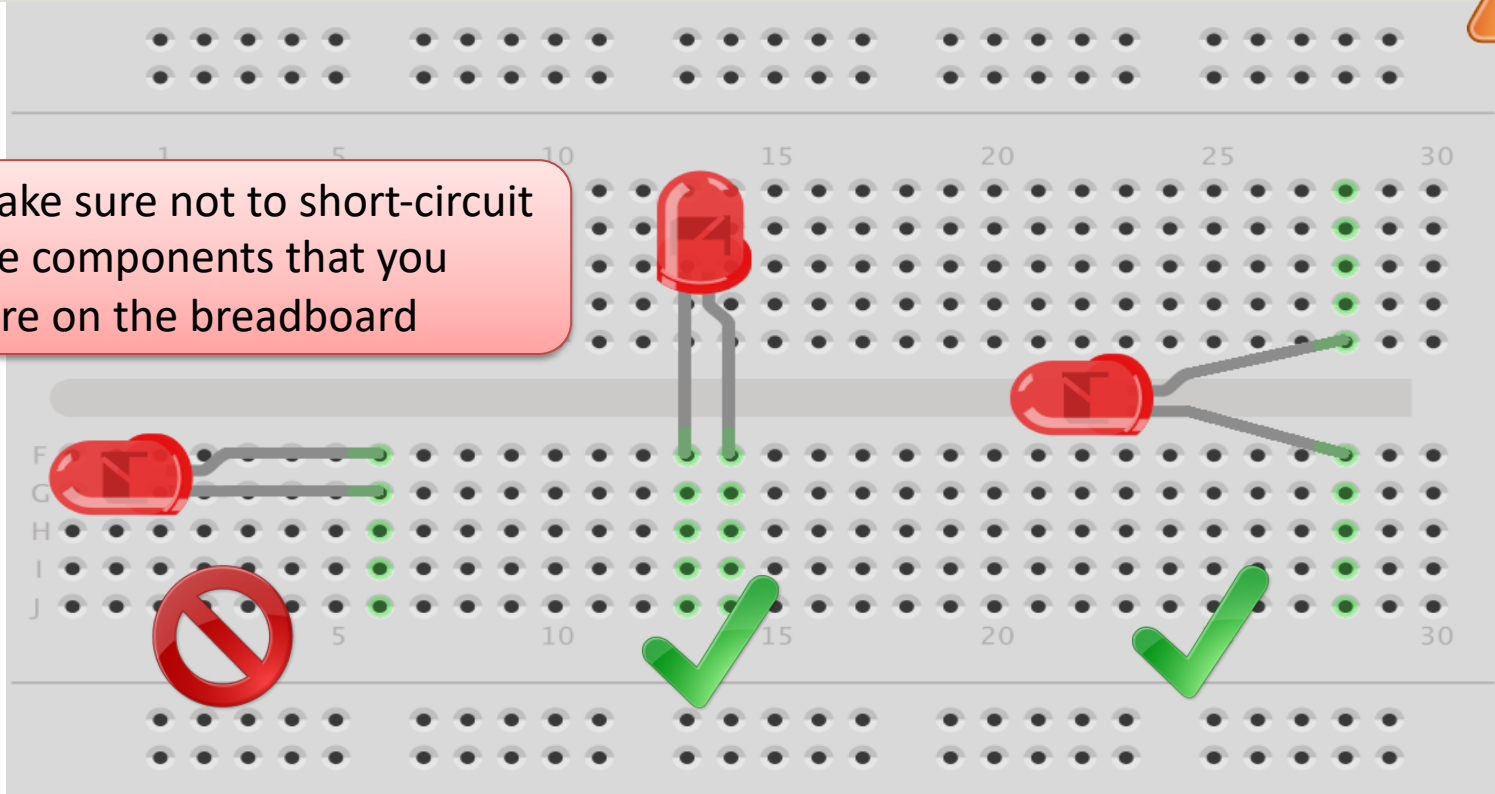
- We will use the Digital Out pins to turn on/off a LED



# Breadboard Wiring



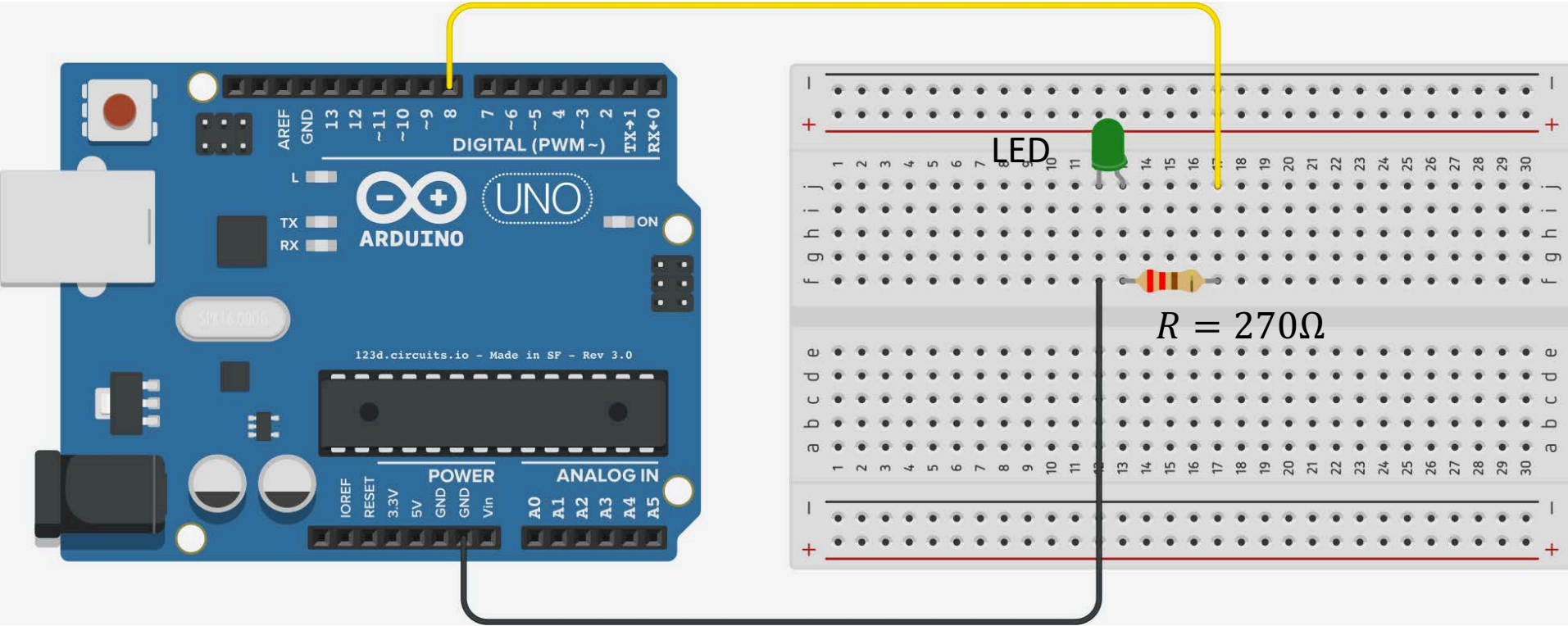
Make sure not to short-circuit the components that you wire on the breadboard



The Breadboard is used to connect components and electrical circuits

fritzing

# LED Wiring



# LabVIEW Example

The image displays a LabVIEW project window for a program named "Digital Out.vi". The window is split into two main views: the Block Diagram on the left and the Front Panel on the right.

**Block Diagram (Left):** This view shows the internal logic of the program. It starts with a "COM6" port selection, leading to an "Open.vi" block. The connection then goes to a "Digital Write.vi" block, which is configured with a "DO Channel" of 8 and a "Digital Write 1 Chan" mode. This is followed by a "Close.vi" block. A "Simple Error Handler.vi" block is connected to the error output of the "Close.vi" block. A "Wait (ms)" block is set to 100 milliseconds. A "Stop Button" is also present, which is connected to a "Digital Write.vi" block. The "LED ON/OFF" indicator is connected to the output of the "Digital Write.vi" block.

**Front Panel (Right):** This view shows the user interface. It features a "LED ON/OFF" indicator, which is a circular slider currently positioned to the right, indicating the LED is on. Below the indicator is a "Stop" button with a red square icon.

# Multiple Digital Out Channels

The image displays the LabVIEW development environment for a VI titled "Digital Out - Multiple Channels.vi".

**Block Diagram (Left):**

- The VI is connected to a COM6 port via a "Serial" interface.
- The process starts with an "Open.vi" block.
- A "DO Channels" control is used to specify the number of channels (set to 2).
- The "Digital Write N Chan" block receives the channel count and the "Values" input.
- A "Wait (ms)" block is set to 100 milliseconds.
- The process concludes with a "Close.vi" block and an "Error!" indicator.

**Front Panel (Right):**

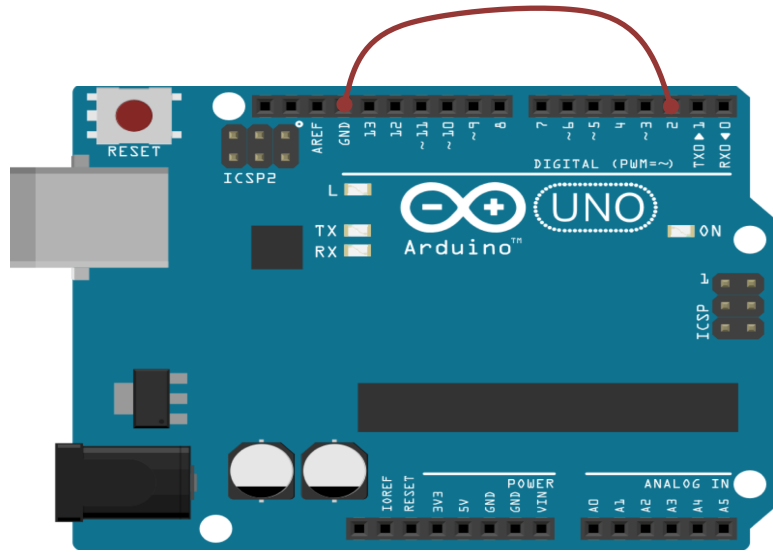
- The "DO Channels" section contains five numeric input fields with values 0, 2, 3, 4, and 5.
- The "Values" section contains a numeric input field with the value 0 and four indicator lights (represented by yellow triangles).
- A "Stop" button is located at the bottom right.



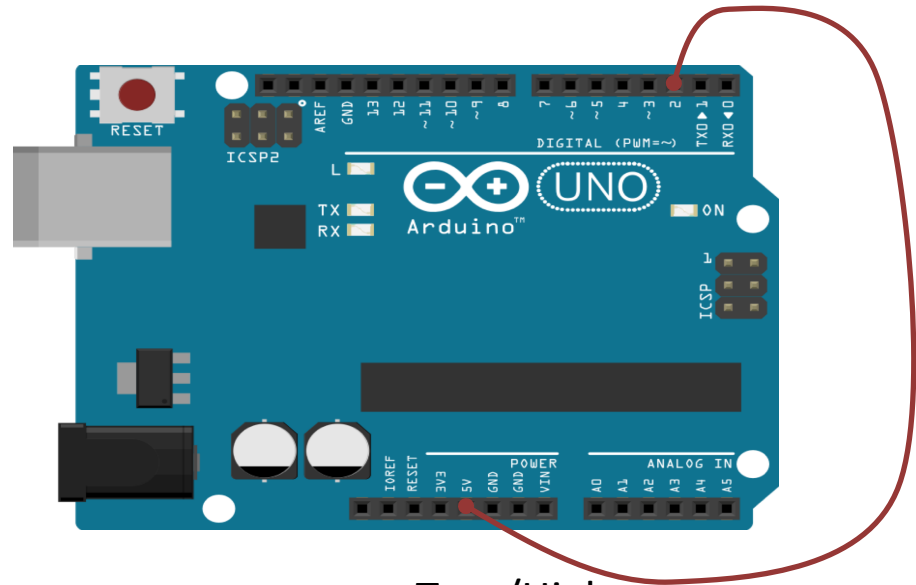
# Digital In (DI)

# LabVIEW - Digital In

We can test the Digital In (Read) by wiring to 5V (True/High) or GND (False/Low)



False/Low



True/High

# LabVIEW - Digital In

The image displays the LabVIEW development environment for a 'Digital In' virtual instrument (VI). It is divided into two main sections: the Block Diagram and the Front Panel.

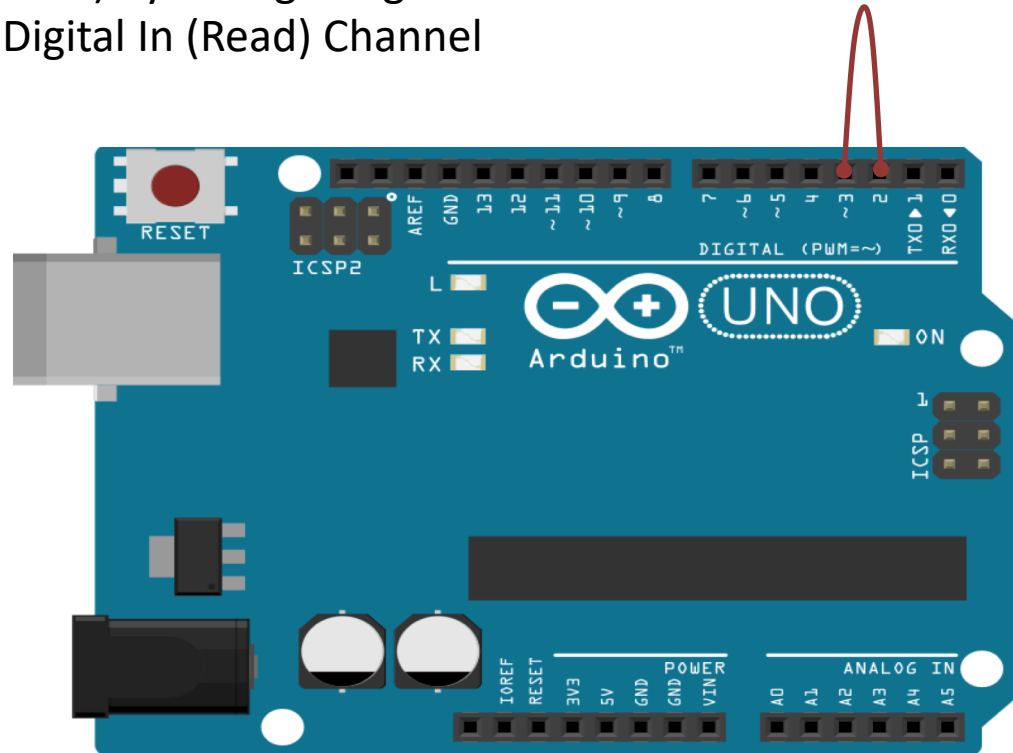
**Block Diagram (Left):** This section shows the internal logic of the VI. It starts with a 'COM6' port selected in a dropdown menu, leading to an 'Open.vi' block. The 'Serial' dropdown is set to 'Serial'. The data then flows to a 'Digital Read.vi' block, which is configured with 'Digital Read 1 Chan'. A 'DI Channel' control is set to '2'. Below the main loop, there is a 'Wait (ms)' block set to '100' and a 'Stop Button' control. The output of the 'Digital Read.vi' block is connected to a 'DI Value' indicator. The loop ends with a 'Close.vi' block and an error handling block labeled 'Simple Er...'. A search bar is visible at the top of the block diagram window.

**Front Panel (Right):** This section shows the user interface of the VI. It features a 'DI Value' indicator, which is currently displaying a green circle. Below it is a 'Stop' button with a red square icon. The window title is 'Digital In.vi Front Panel' and the menu bar includes 'File', 'Edit', 'View', 'Project', 'Operate', 'Tools', and 'Wi...'. The font size is set to '15pt Application'.

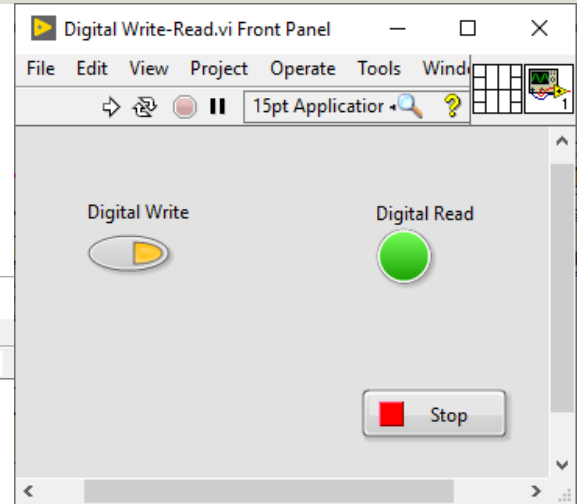
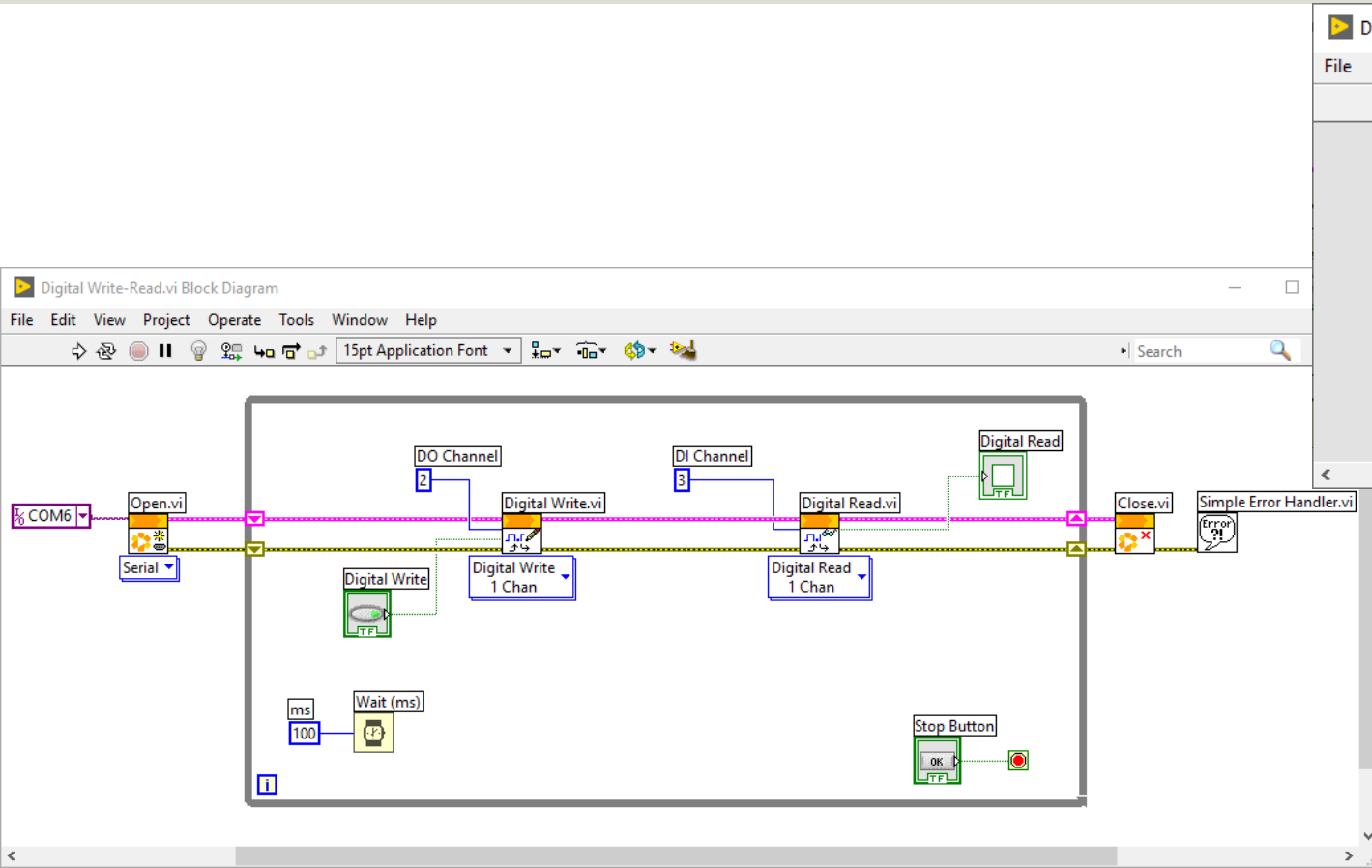


# LabVIEW Digital Write - Read

We can test the Digital In (Read) by wiring a Digital Out (Write) Channel to the Digital In (Read) Channel

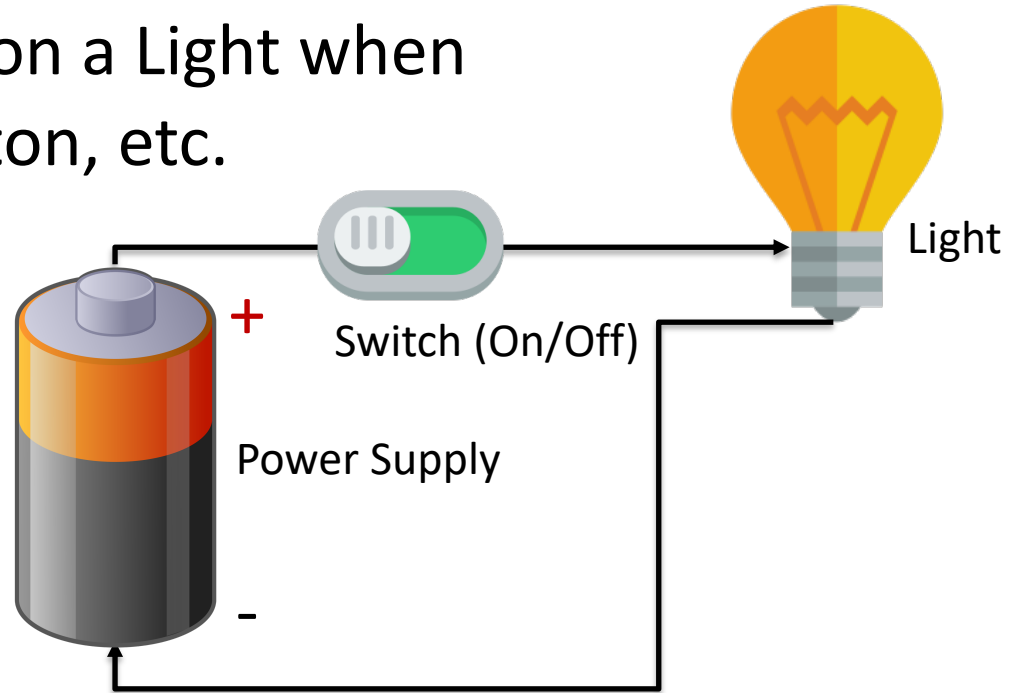


# LabVIEW Digital Write - Read



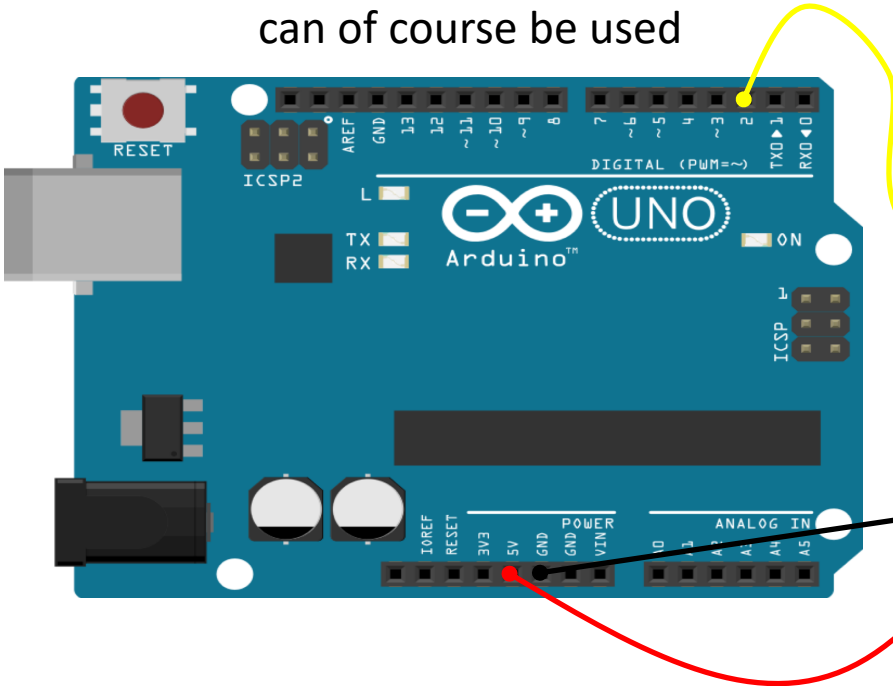
# Push Button/Switch

- Pushbuttons or switches connect two points in a circuit when you press them.
- You can use it to turn on a Light when holding down the button, etc.

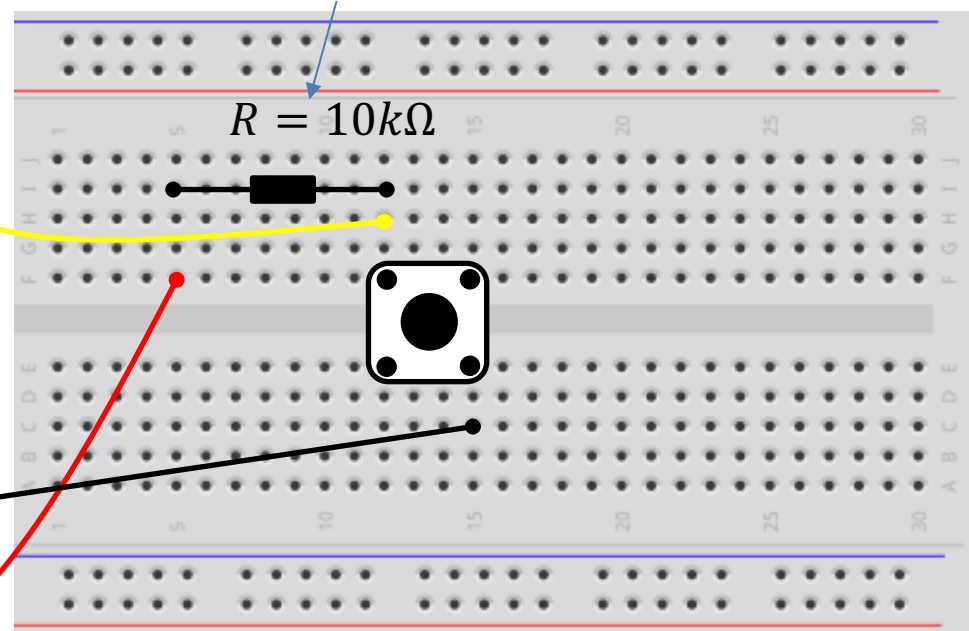


# Wiring (Pull-up Resistor)

Another Digital Channel  
can of course be used



Using external Pull-up Resistor

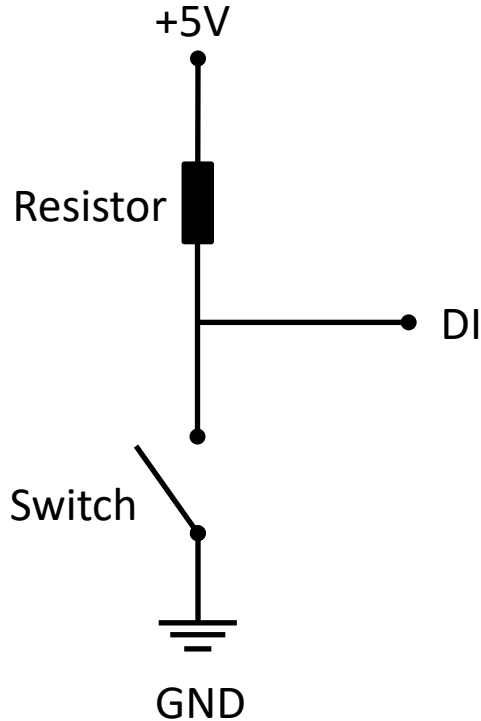


# Pull-down/Pull-up Resistor

Why do we need a pull-up or pull-down resistor in the circuit?

- If you disconnect the digital I/O pin from everything, it will behave in an irregular way.
- This is because the input is "floating" - that is, it will randomly return either HIGH or LOW.
- That's why you need a pull-up or pull-down resistor in the circuit.

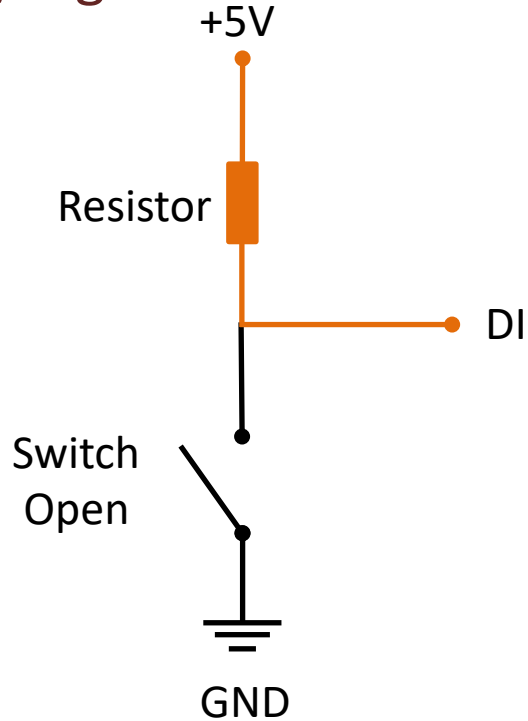
# Pull-up Resistor



- When the pushbutton is open (unpressed) there is a connection between 5V and the DI pin.
- This means the default state is **True** (High).
- When the button is closed (pressed), the state goes to **False** (Low).

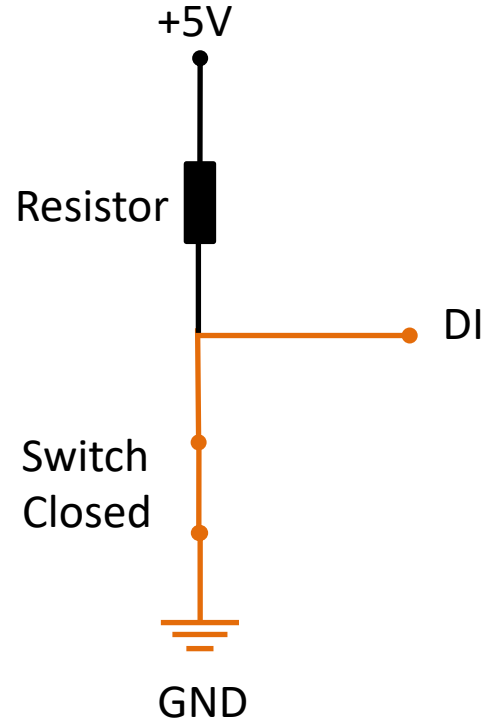
# Pull-up Resistor

True/High



False/Low

We Push the Button



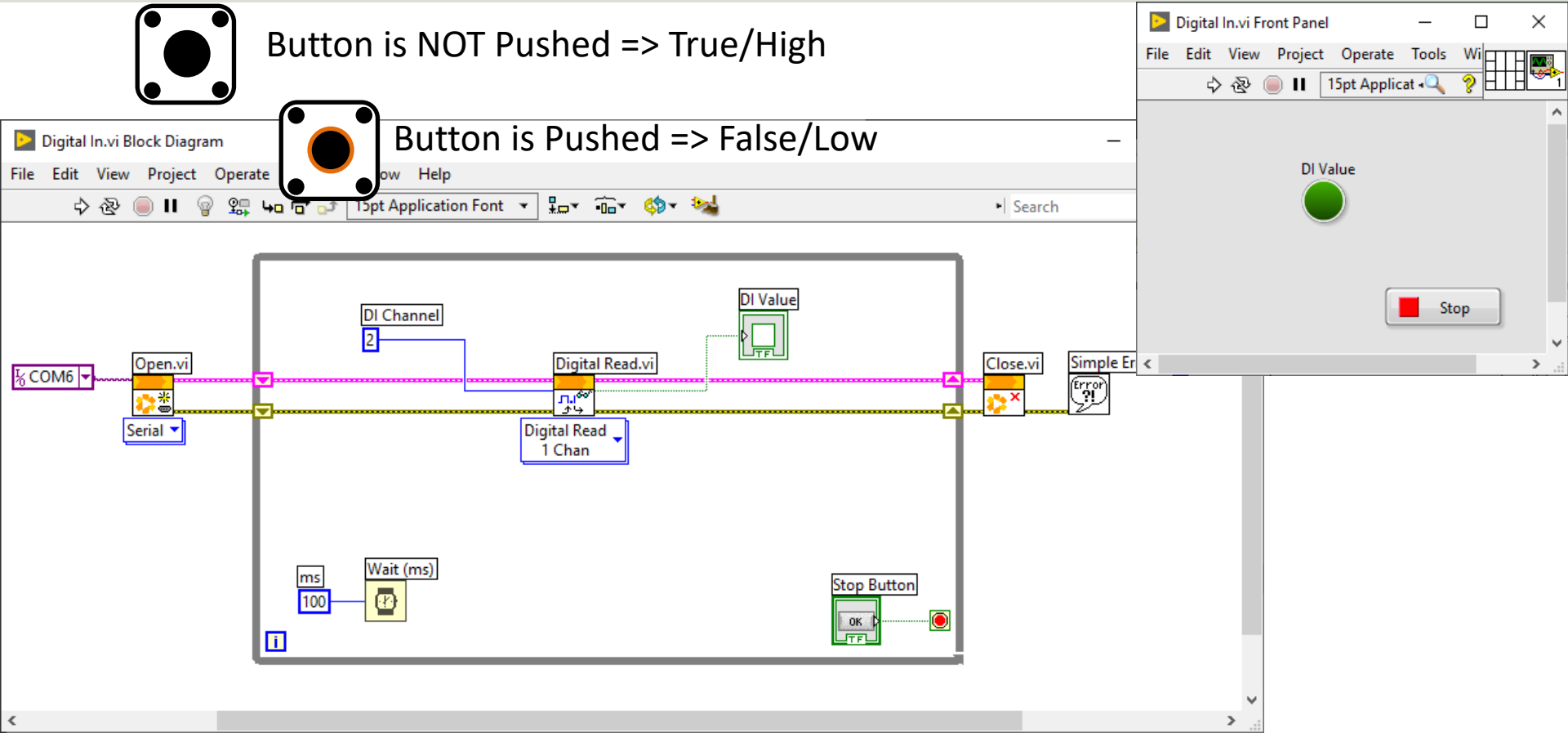
# Push Button (Pull-up Resistor)



Button is NOT Pushed => True/High



Button is Pushed => False/Low





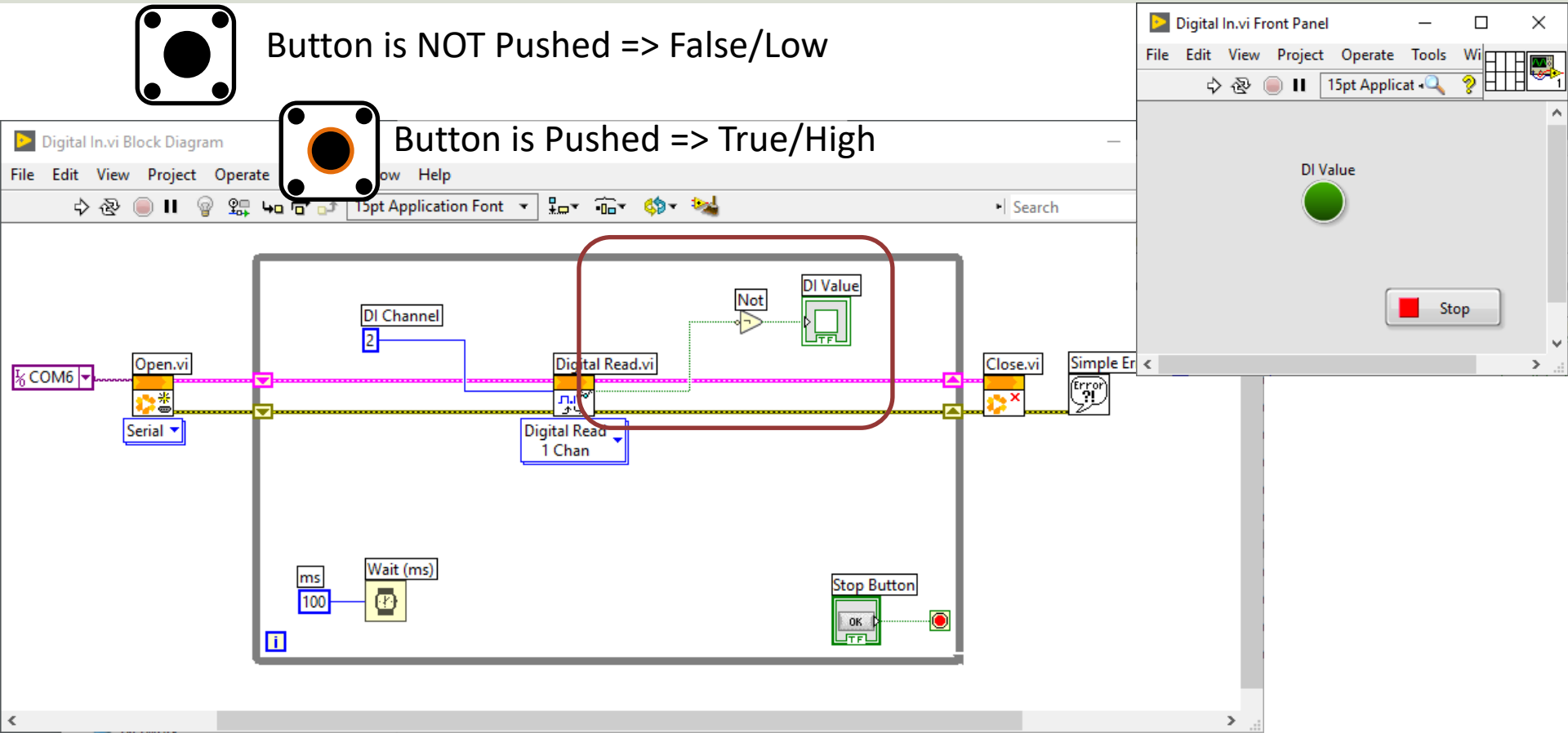
# Adding a "NOT" block



Button is NOT Pushed => False/Low



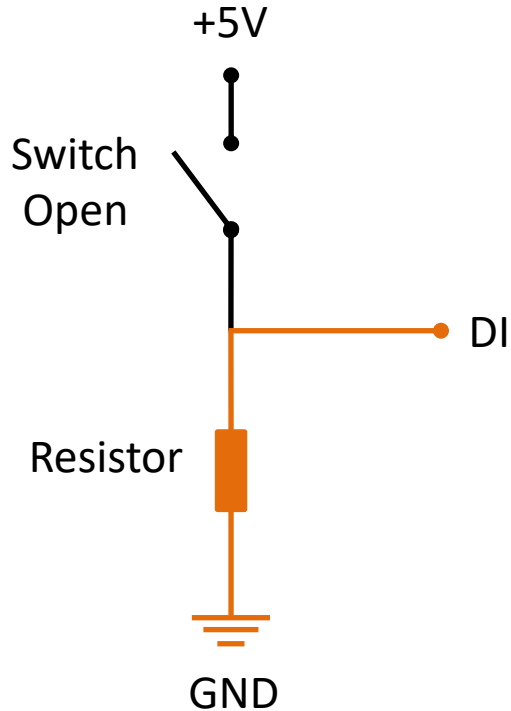
Button is Pushed => True/High



# Pull-down Resistor

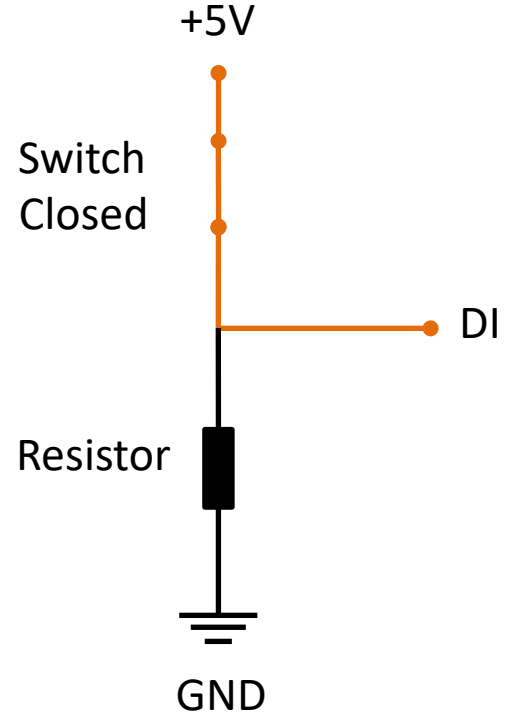
We could also have wired according to a “Pull-down” Resistor

False/Low



True/High

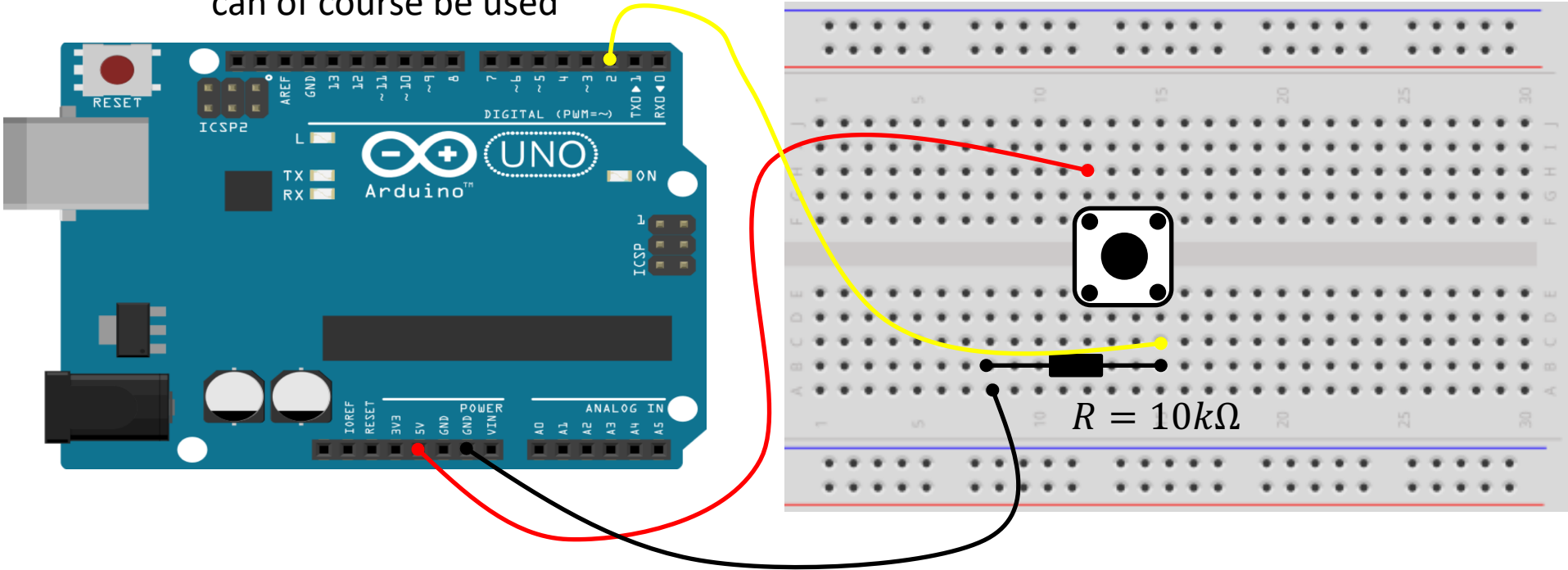
→  
We Push the Button



# Wiring (Pull-down Resistor)

Another Digital Channel  
can of course be used

Using external Pull-down Resistor



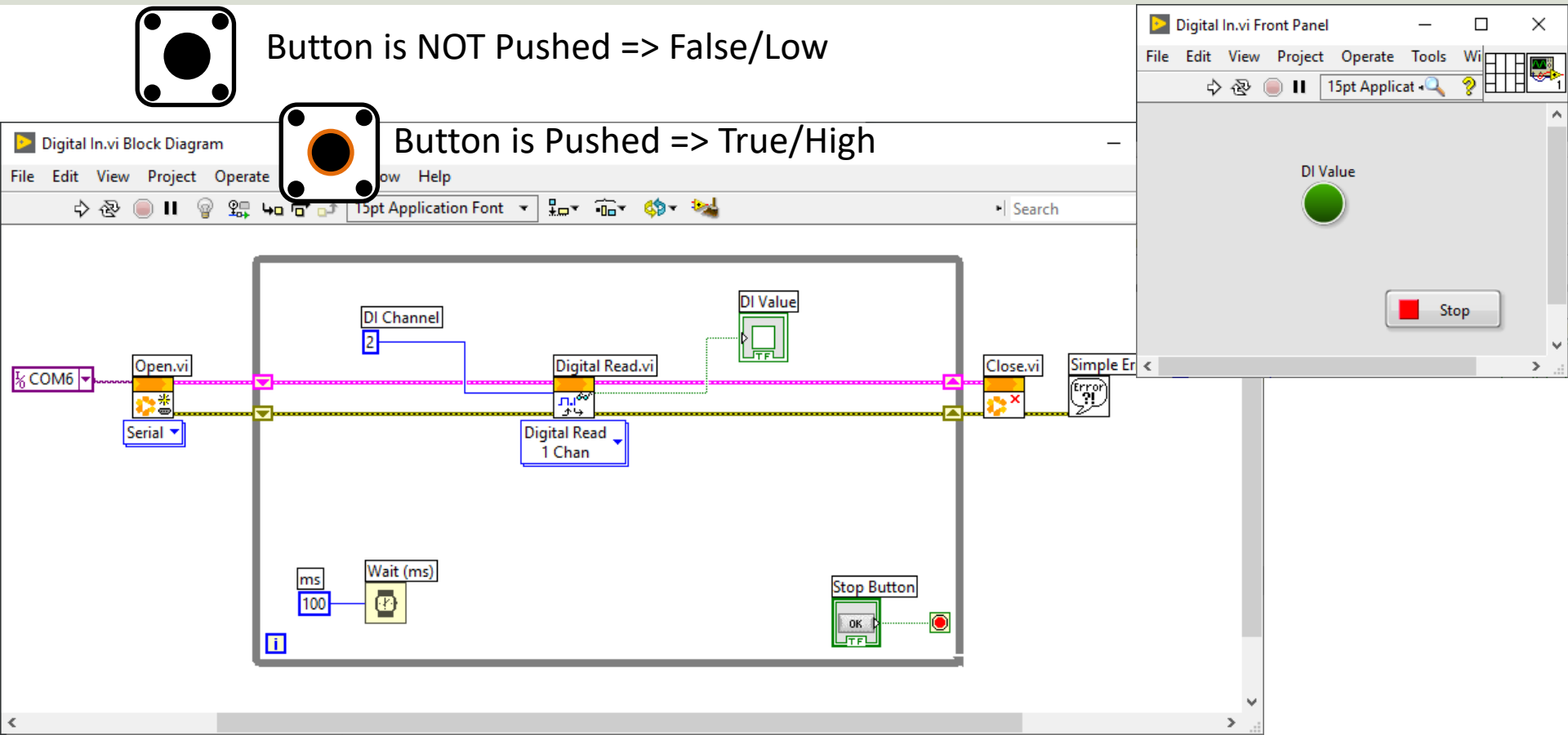
# Push Button (Pull-down Resistor)



Button is NOT Pushed => False/Low



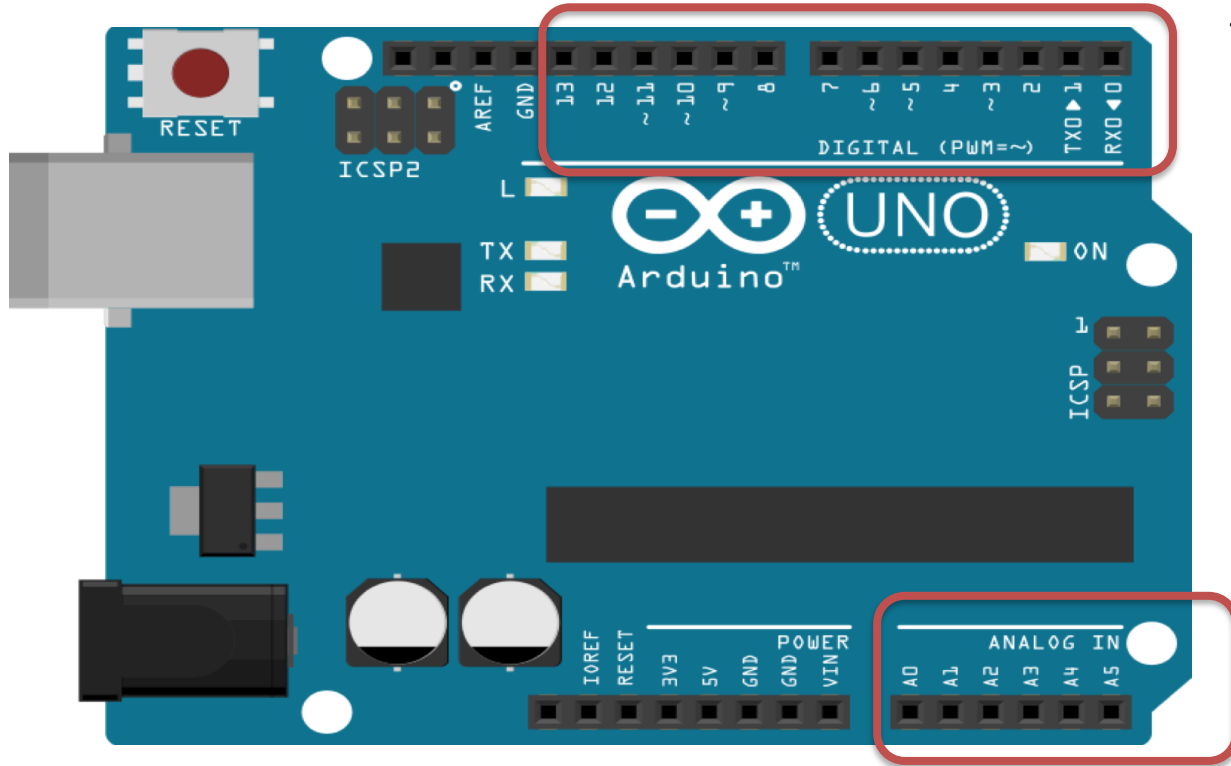
Button is Pushed => True/High





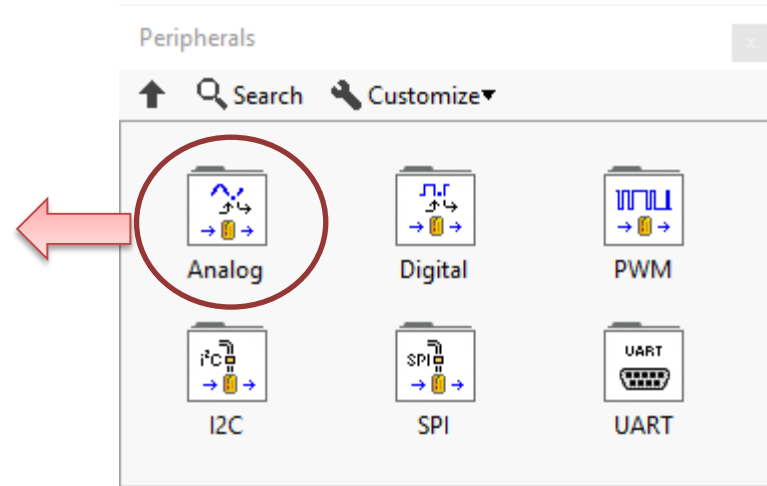
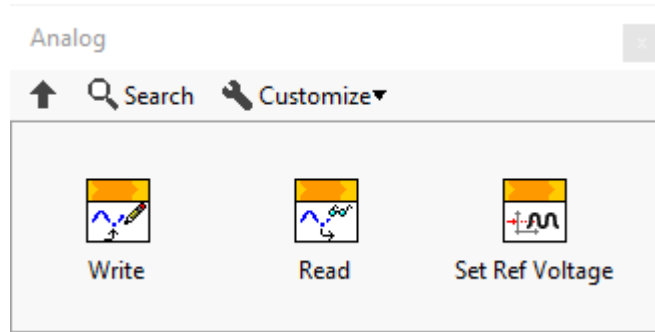
# Analog I/O

# Analog I/O



Those marked with ~ can also be used as "Analog Outputs", so-called PWM outputs

# LabVIEW Palette – Analog I/O



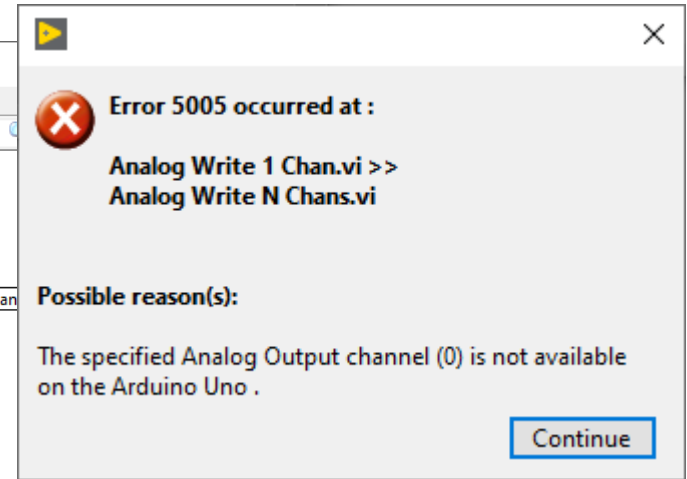
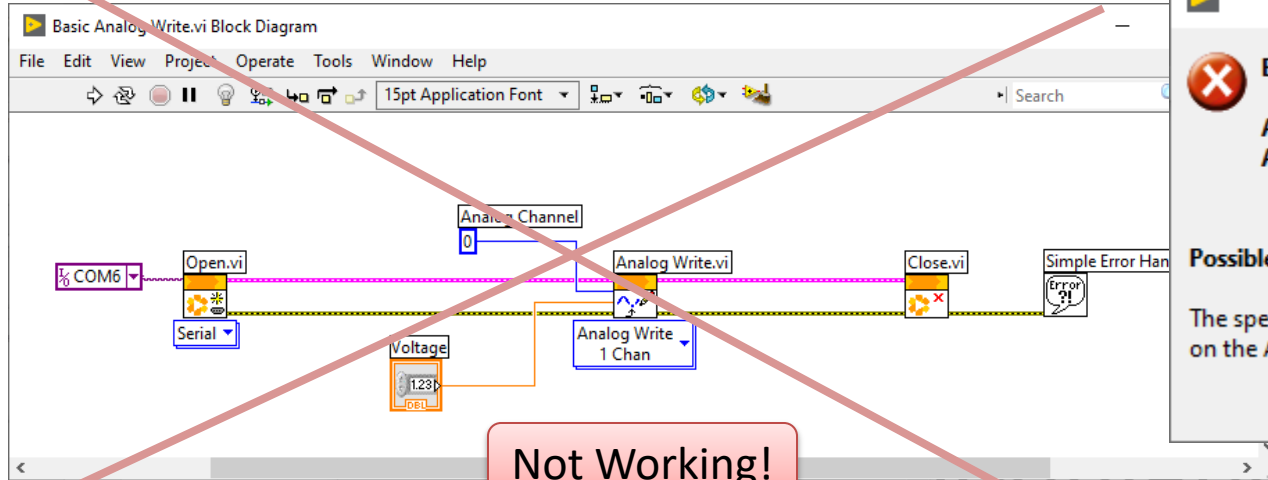


# Analog Out (AO)



# Analog Out

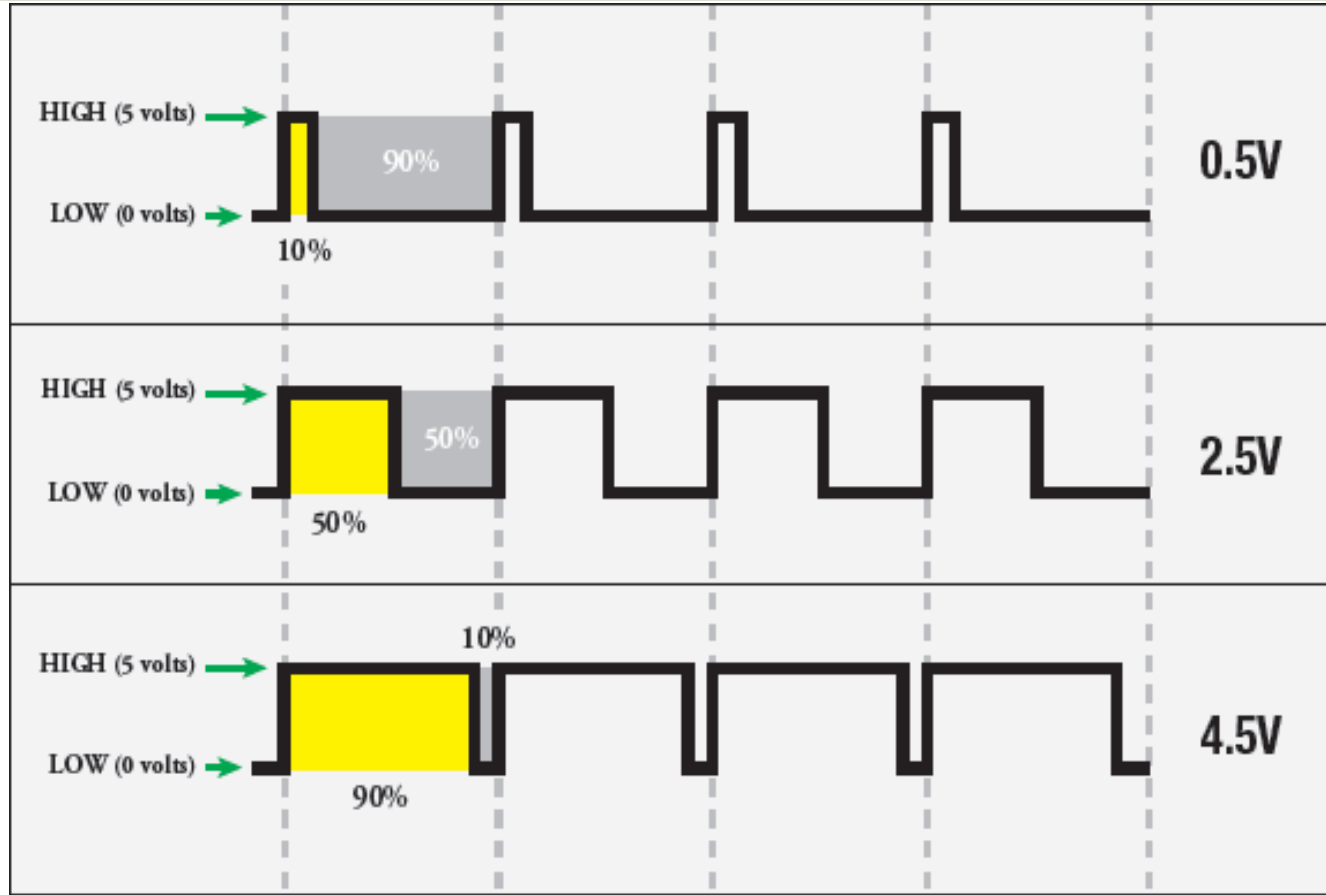
- Arduino UNO has no real Analog Out
- We need to use Pulse Width Modulation (PWM)



# PWM as “Analog Out”

The Arduino UNO has no real Analog Out pins, but we can use a PWM pin.

PWM can be used to control brightness of a LED, control the speed of a Fan, control a DC Motor, etc.



# PWM

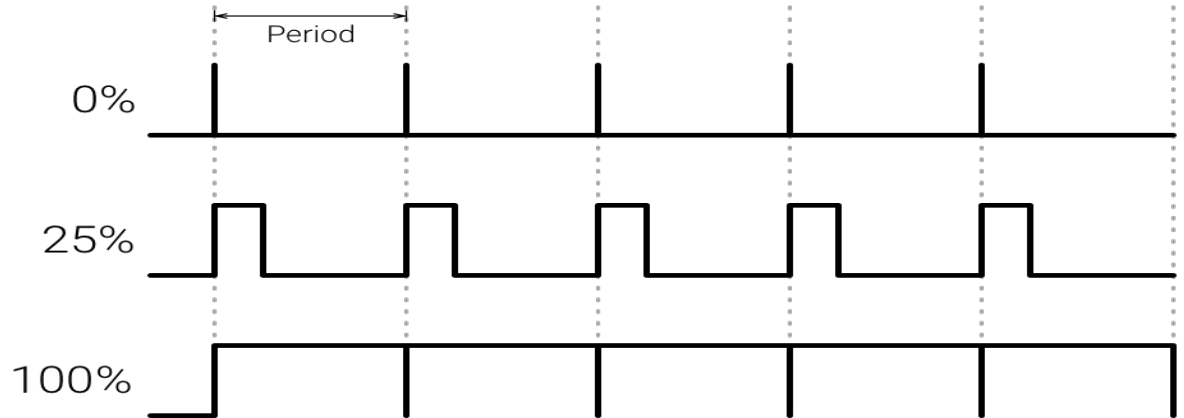
PWM is a digital (i.e., square wave) signal that oscillates according to a given *frequency* and *duty cycle*.

The frequency (expressed in Hz) describes how often the output pulse repeats.

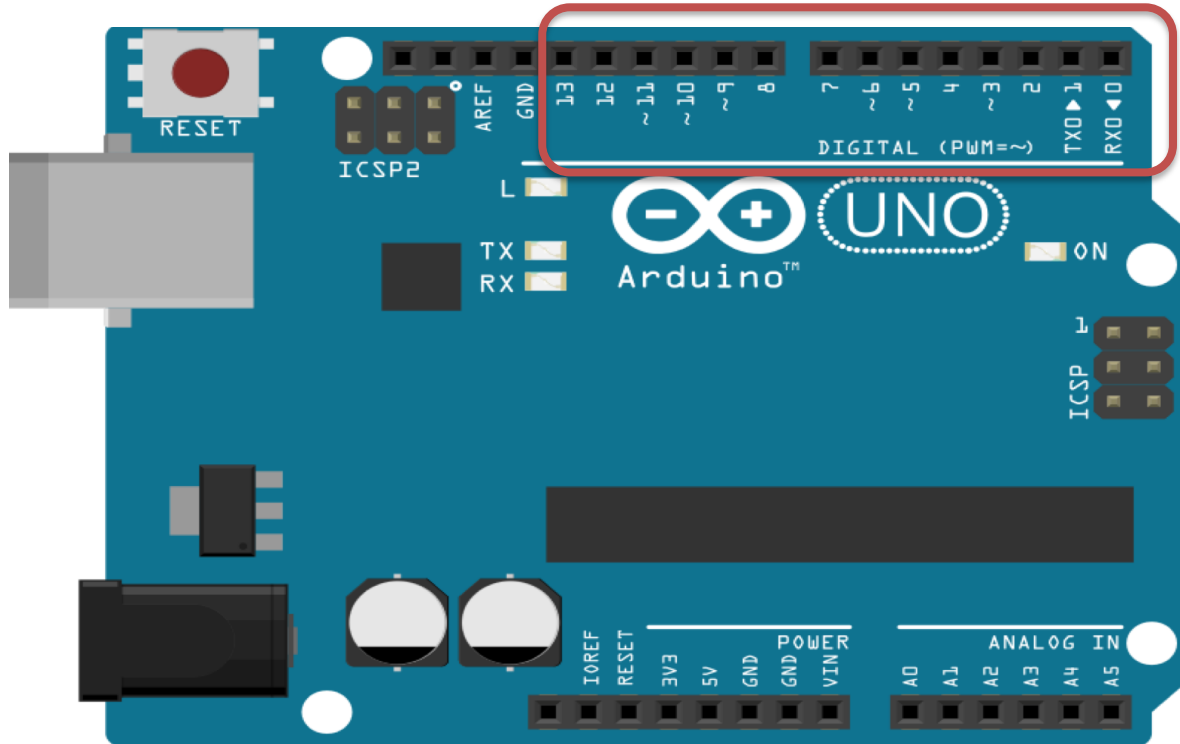
The period is the time each cycle takes and is the inverse of frequency.

The duty cycle (expressed as a percentage) describes the width of the pulse within that frequency window.

You can adjust the duty cycle to increase or decrease the average "on" time of the signal. The following diagram shows pulse trains at 0%, 25%, and 100% duty:



# PWM



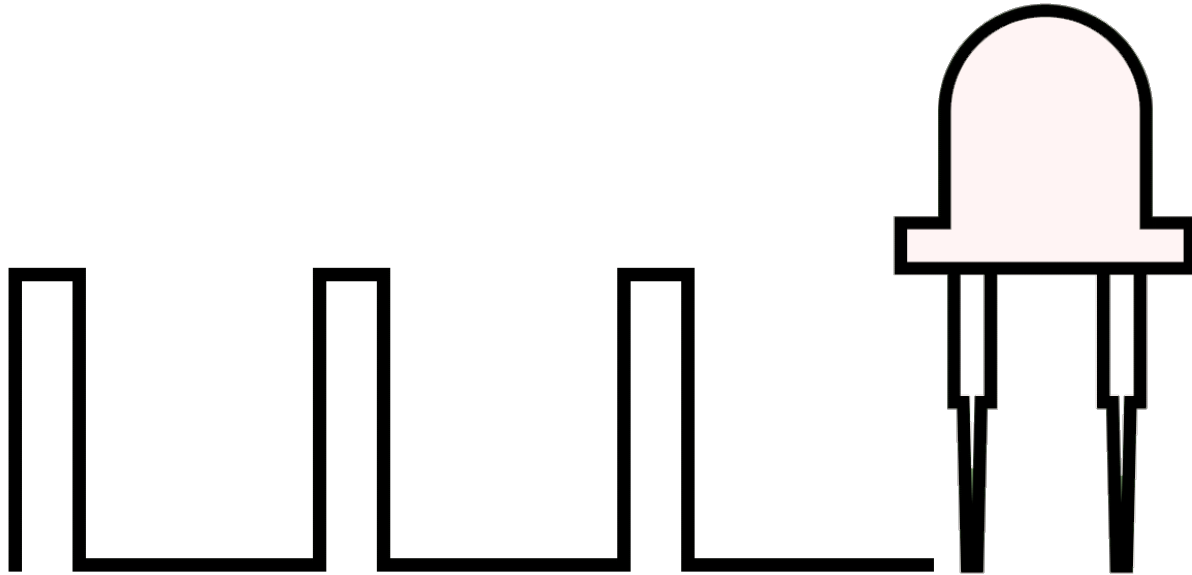
The Digital Pins marked with ~ can be used as "Analog Outputs", so-called PWM outputs

# Control Brightness of a LED

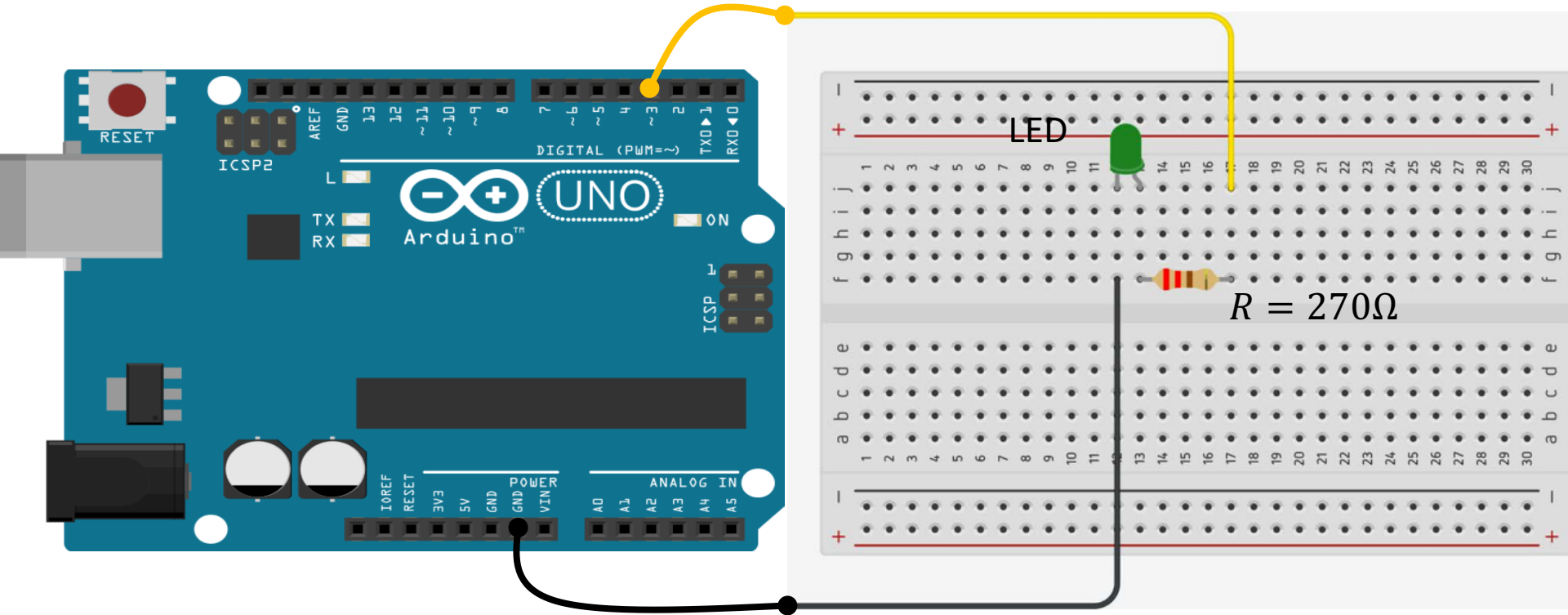
- We've seen how to turn an LED on and off, but how do we control its brightness levels?
- An LED's brightness is determined by controlling the amount of current flowing through it, but that requires a lot more hardware components.
- A simple trick we can do is to flash the LED faster than the eye can see!
- By controlling the amount of time the LED is on versus off, we can change its perceived brightness.
- This is known as *Pulse Width Modulation* (PWM).

# Control Brightness of a LED

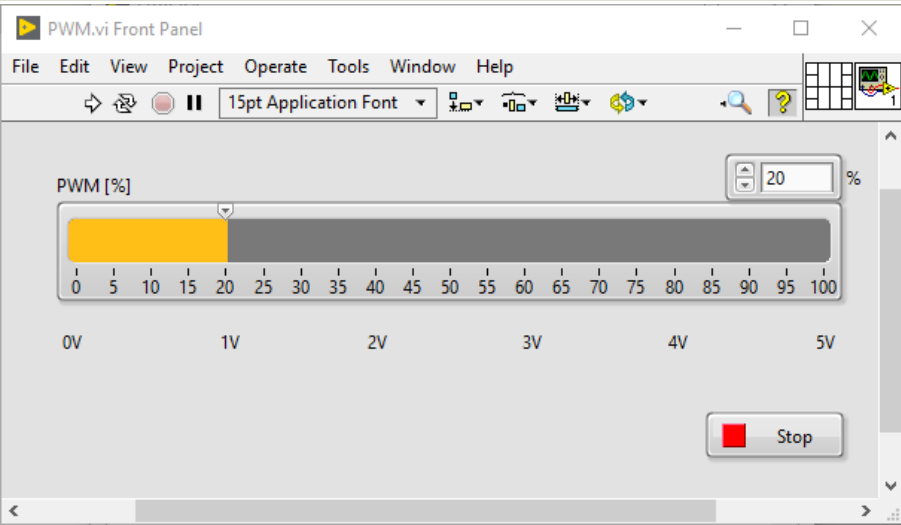
Below we see how we can use PWM to control the brightness of a LED



# Wiring

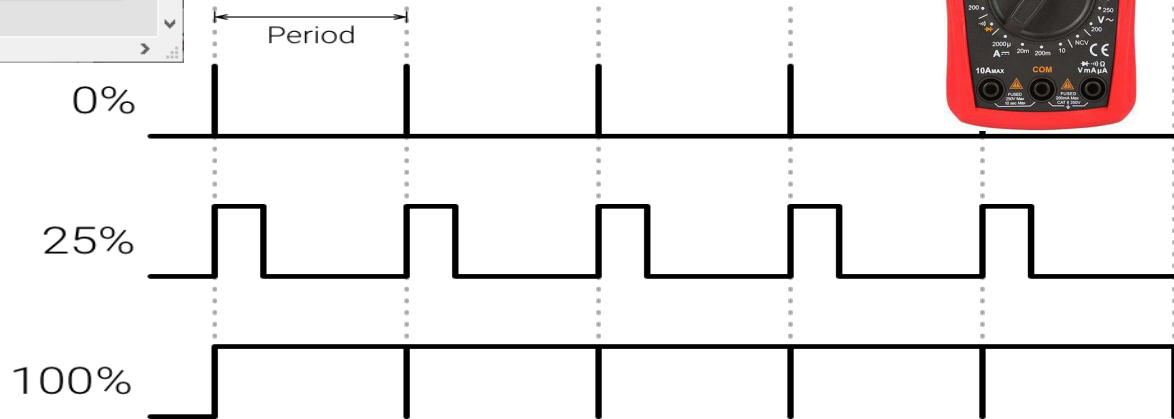
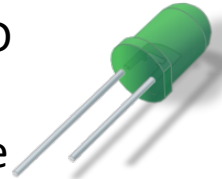


# PWM Example



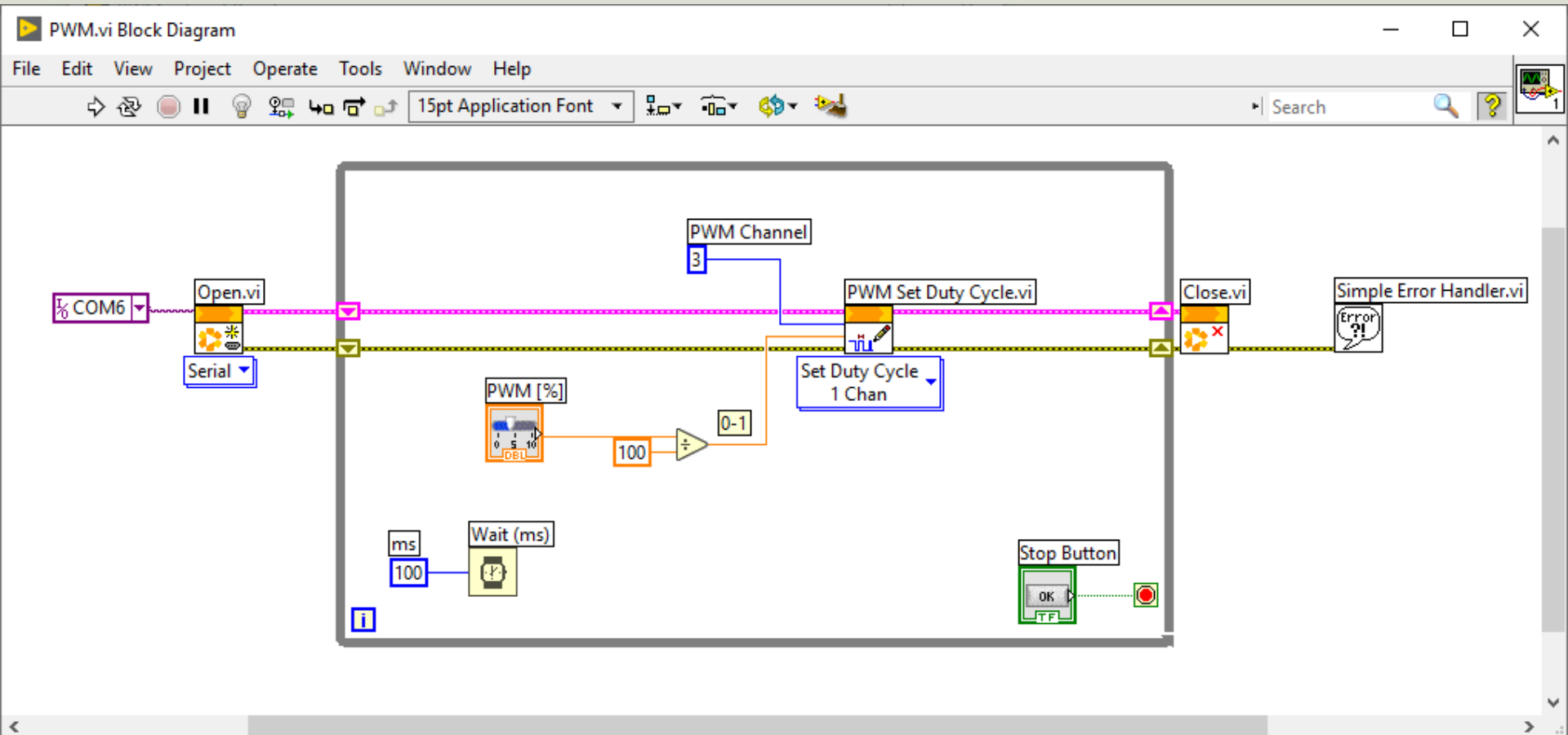
We will see the brightness of the LED will increase.

Or you can use a Multimeter and see the (average) voltage will increase

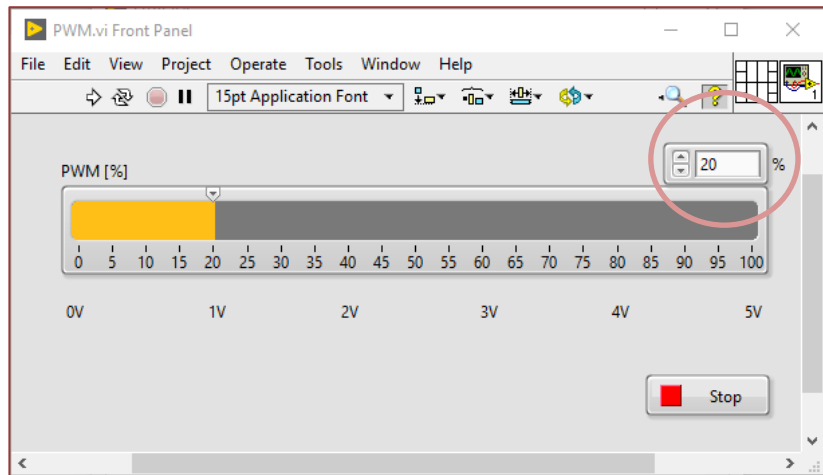




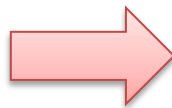
# PWM Example



# PWM Example



20%  $\rightarrow$  1V



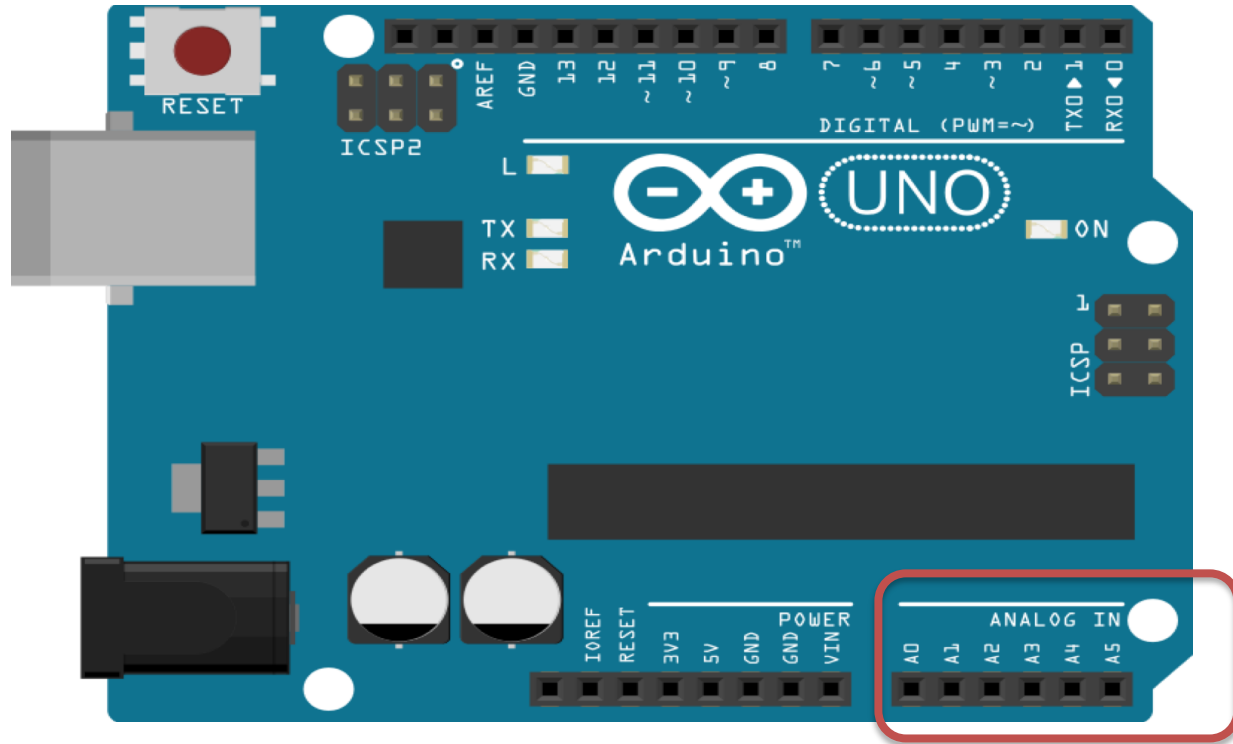
0-100%  $\rightarrow$  0-5V



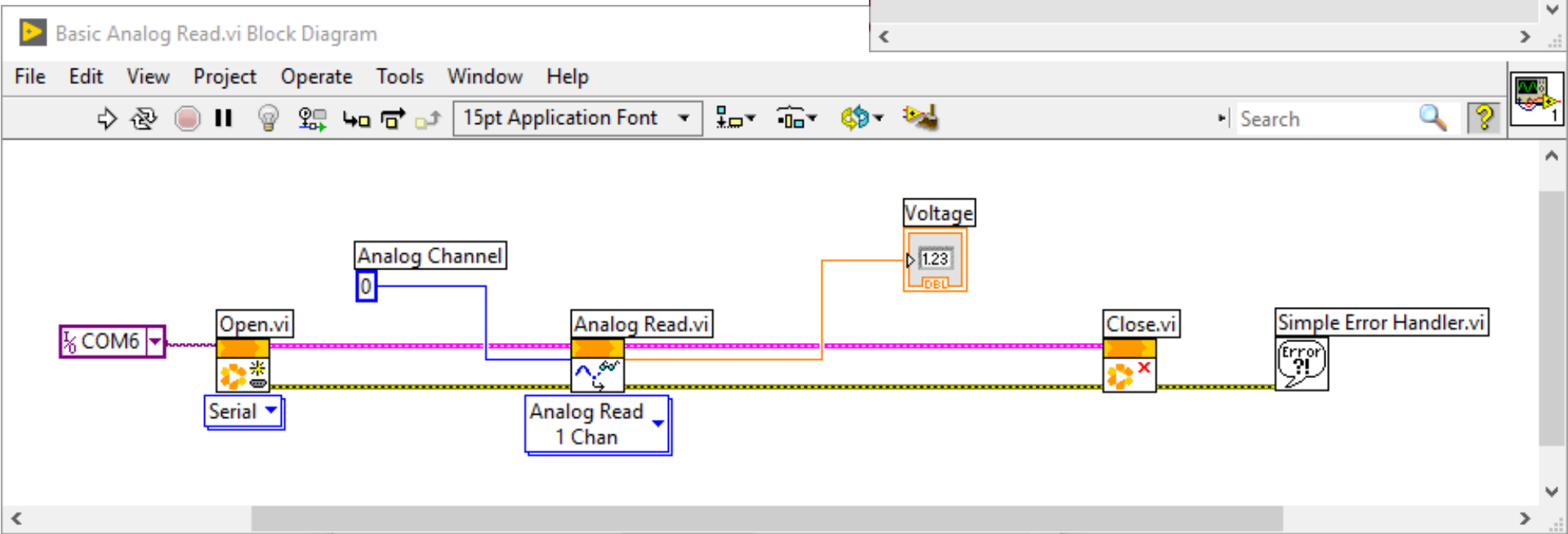
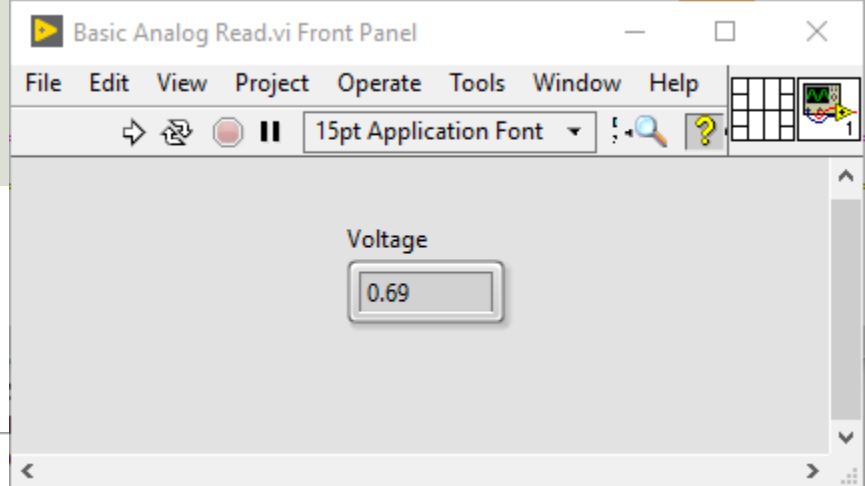


# Analog In (AI)

# Analog Input



# Analog In



# Analog In

Analog Read.vi Front Panel

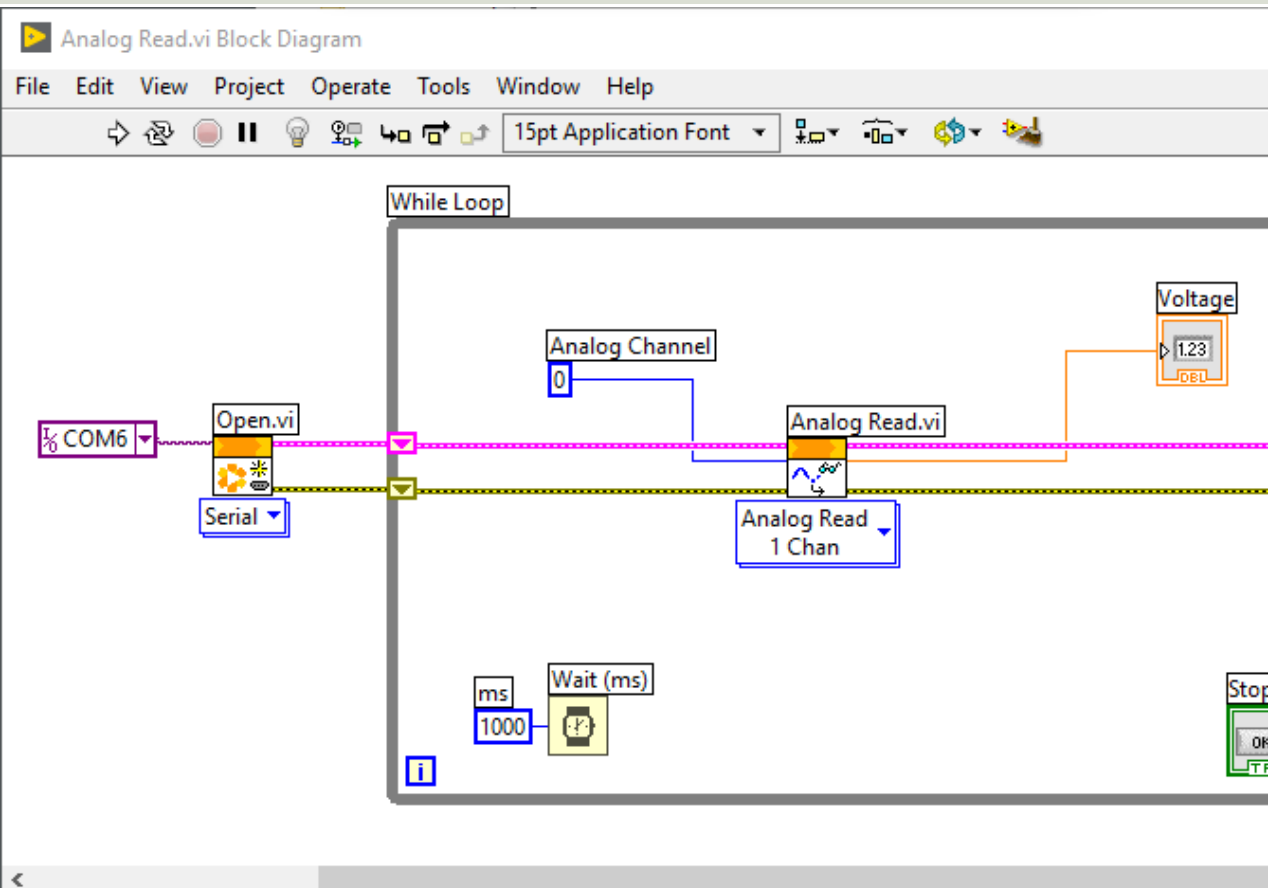
File Edit View Project Operate Tools Window

15pt Application Font

Voltage

0.74

Stop





# LabVIEW LINX

## Arduino

# Temperature Sensors



# Sensors

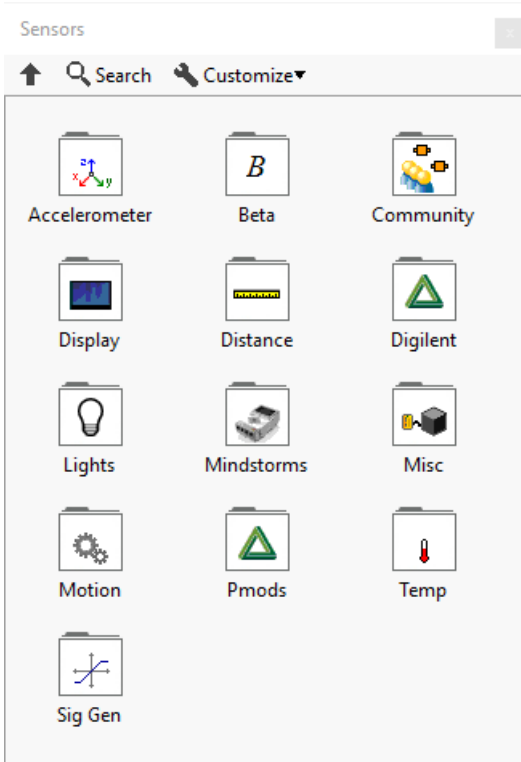


# Sensors

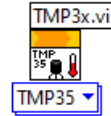
We will use 2 different types of Temperature Sensors:

- TMP36 Temperature Sensor
- Thermistor Temperature Sensor

# Sensors



Some Examples of  
premade Sensor VIs



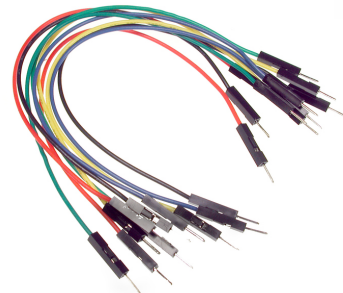
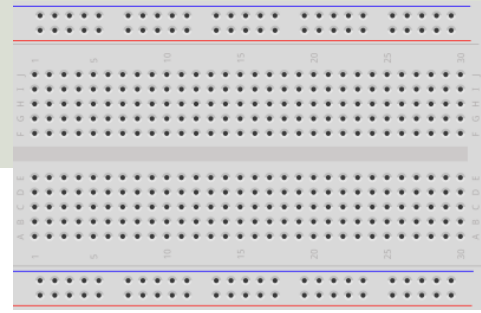
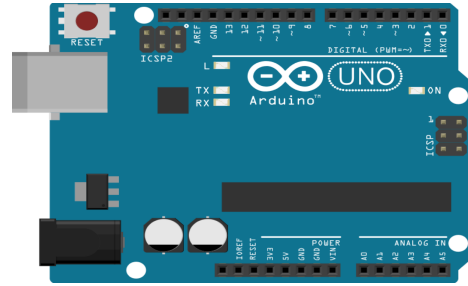
You can use these, or you  
can also easily make your  
own VIs for interfacing  
Sensors from scratch



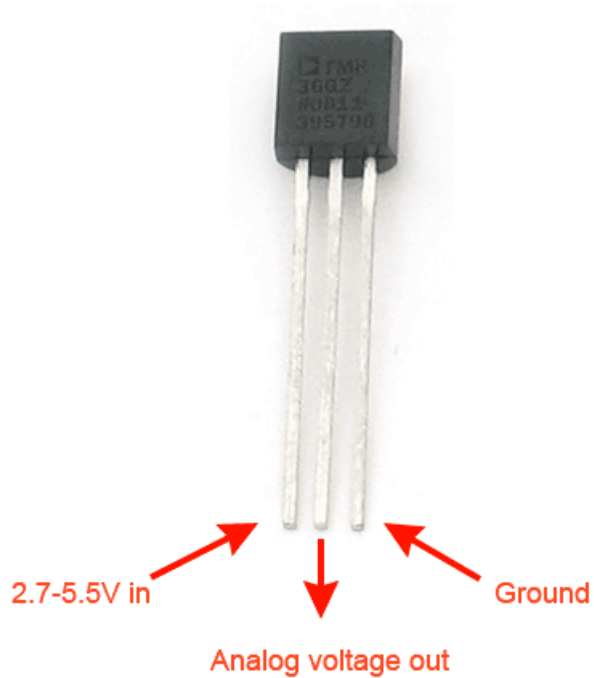
# TMP36 Temperature Sensor

# Hardware

- Arduino
- Breadboard
- TMP36 Temperature Sensor
- Wires (Jumper Wires)



# TMP36 Temperature Sensor



- A Temperature sensor like TM36 use a solid-state technique to determine the temperature.
- They use the fact as temperature increases, the voltage across a diode increases at a known rate.
- It costs only about \$1



<https://learn.adafruit.com/tmp36-temperature-sensor>

# TMP36



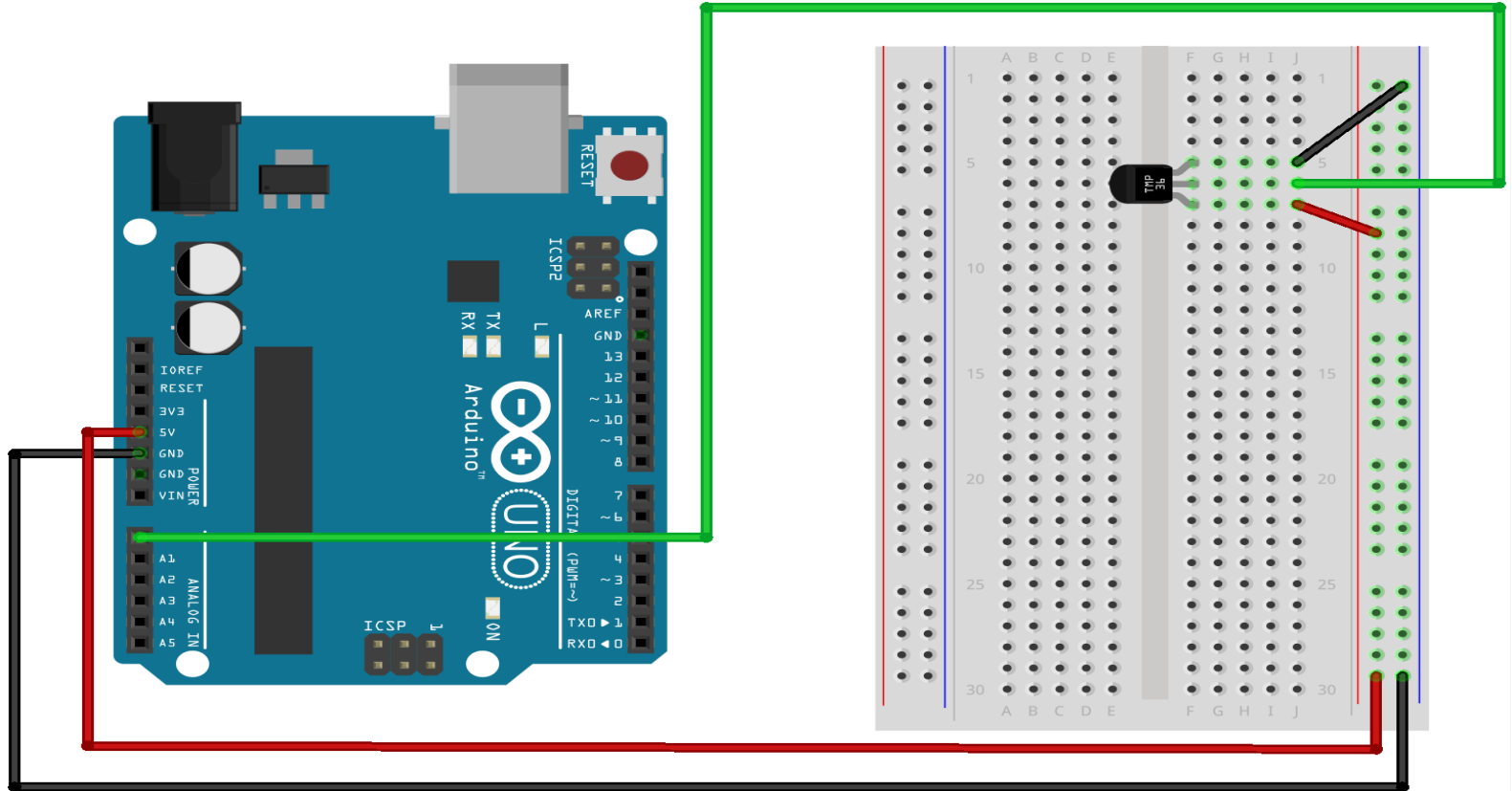
FRONT



BACK

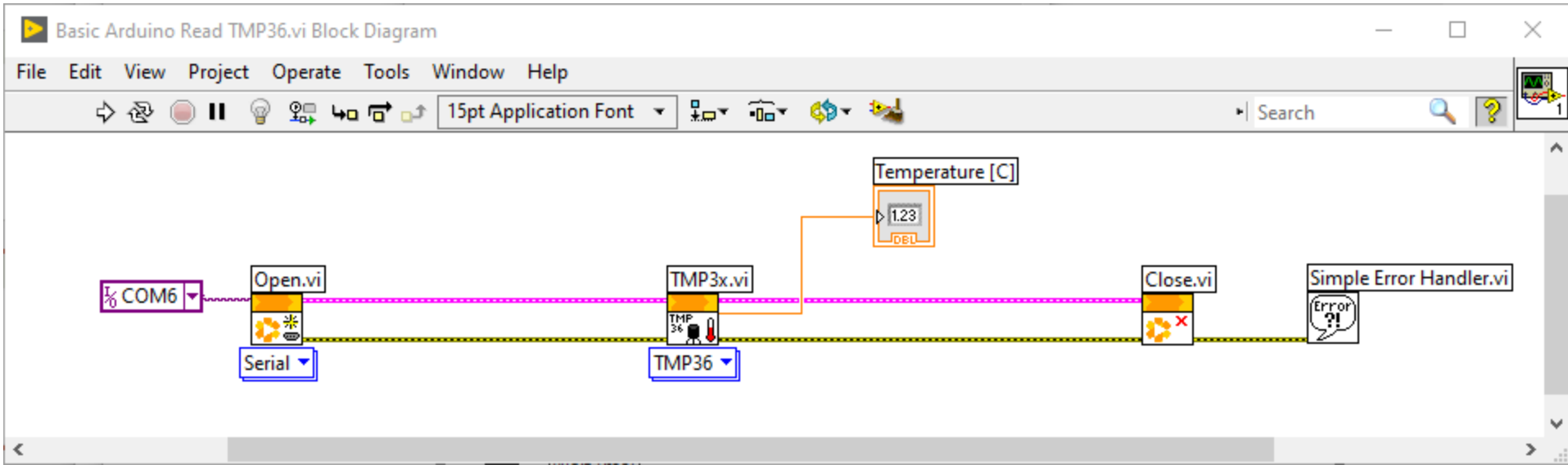
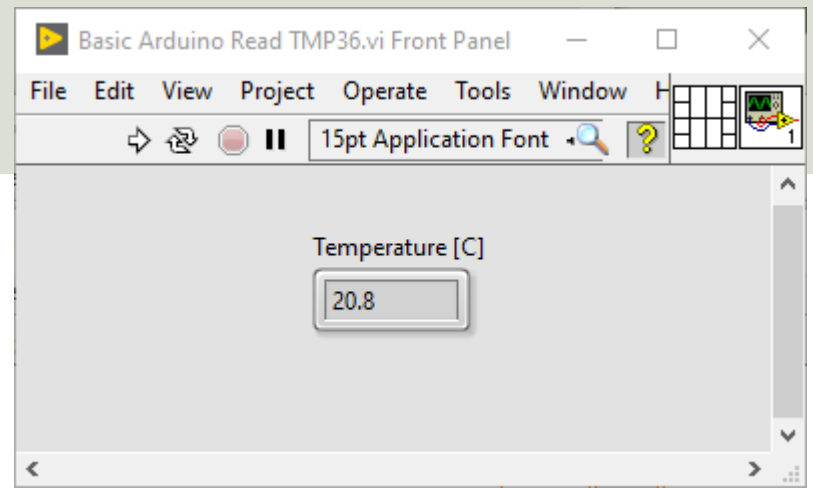
TMP is a small, low-cost temperature sensor and cost about \$1 (you can buy it “everywhere”)

# Wiring



# LabVIEW

LabVIEW LINX has a built-in SubVI for TMP3x Sensors





# LabVIEW

Arduino Read TMP36.vi Front Panel

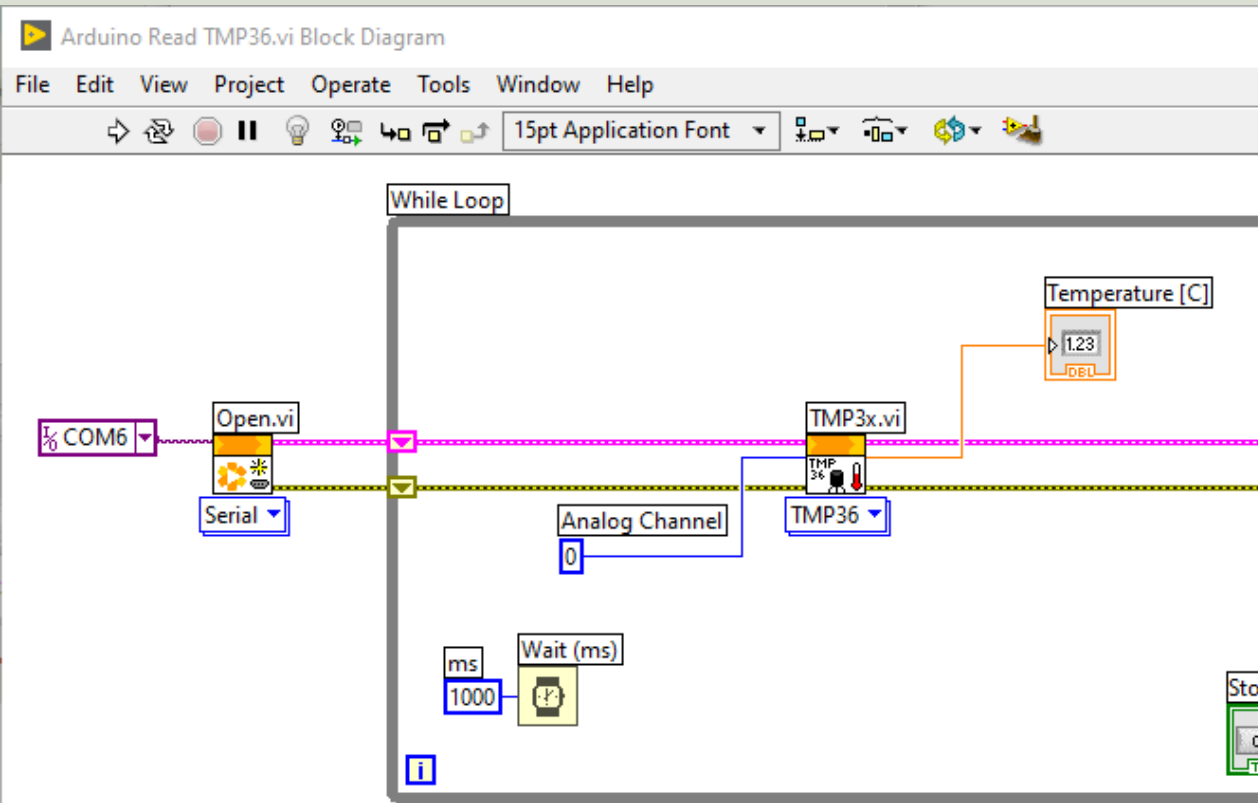
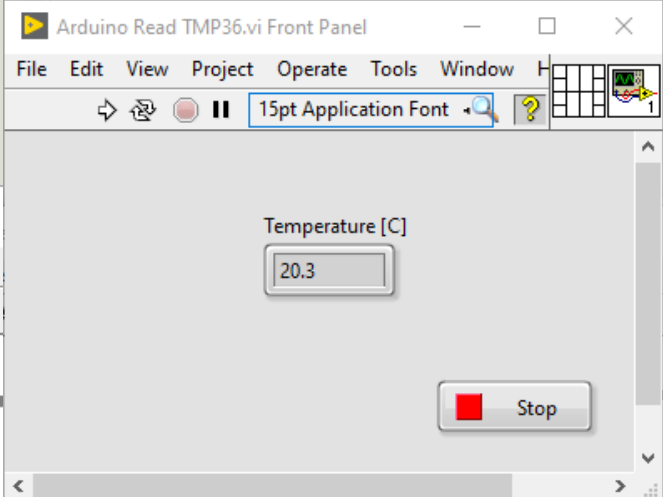
File Edit View Project Operate Tools Window

15pt Application Font

Temperature [C]

20.3

Stop

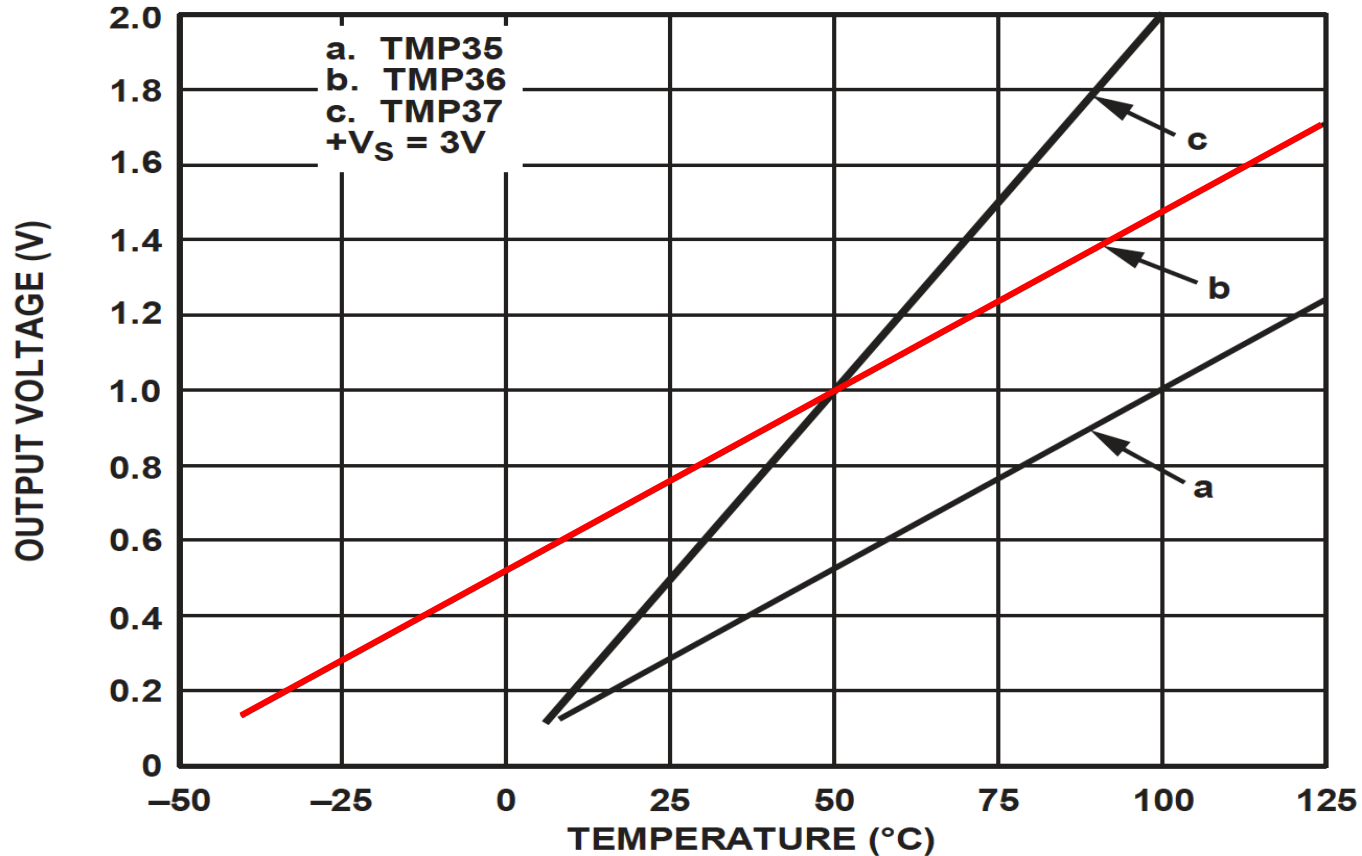


# Do it from Scratch

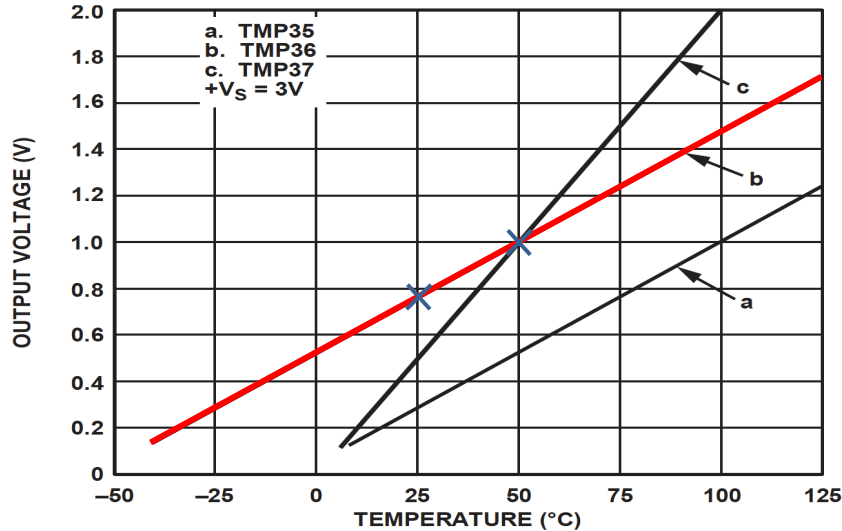
- LabVIEW LINX has a built-in SubVI for TMP3x Sensors (TMP3x.vi)
- Let's see how we can do it from “Scratch”
- We use the Datasheet and the ordinary “Analog Read.vi”

# Datasheet

## Output Voltage vs. Temperature



# Linear Scaling



This gives:

$$y - 25 = \frac{50 - 25}{1 - 0.75} (x - 0.75)$$

Then we get the following formula:

$$y = 100x - 50$$

Convert from Voltage (V) to degrees Celsius  
From the Datasheet we have:

$$(x_1, y_1) = (0.75V, 25^\circ C)$$
$$(x_2, y_2) = (1V, 50^\circ C)$$

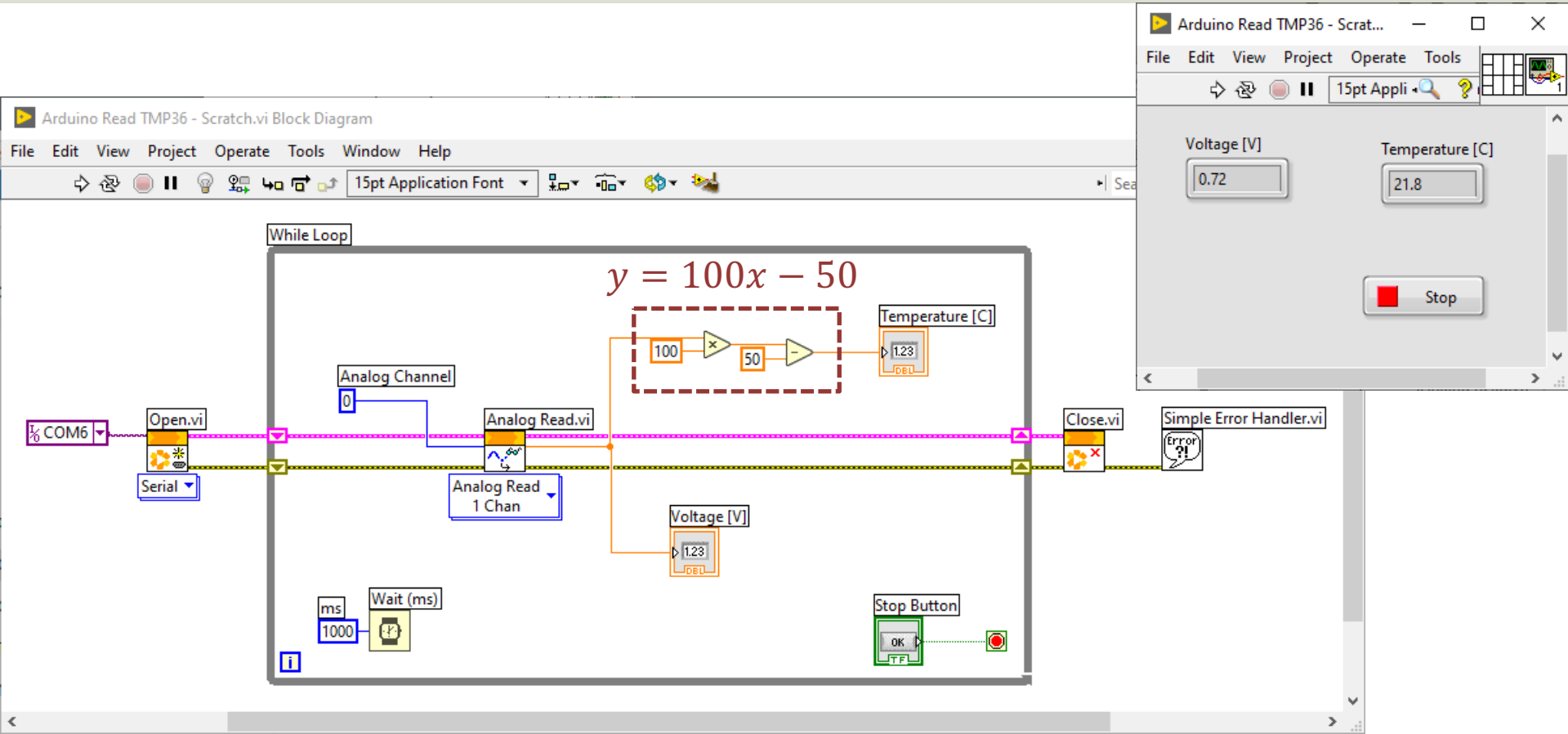
There is a linear relationship between  
Voltage and degrees Celsius:

$$y = ax + b$$

We can find a and b using the following  
known formula:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1} (x - x_1)$$

# LabVIEW Example



The image displays a LabVIEW interface for an Arduino-based temperature measurement system. The main window is titled "Arduino Read TMP36 - Scratch.vi Block Diagram".

**Block Diagram:**

- The process begins with an **Open.vi** block, which is connected to a **Serial** port dropdown menu set to **COM6**.
- The data flow enters a **While Loop**. Inside the loop, an **Analog Channel** block is set to **0**.
- The **Analog Read.vi** block is configured to read from **Analog Read 1 Chan**.
- The raw data is split into two paths:
  - One path goes to a **Voltage [V]** indicator, which displays **1.23**.
  - The other path goes through a dashed red box containing a mathematical conversion:  $y = 100x - 50$ . This is implemented using a **multiply** block (multiplier **100**) followed by a **subtract** block (subtractor **50**).
- The result of the conversion is sent to a **Temperature [C]** indicator, which also displays **1.23**.
- At the bottom of the loop, there is a **Wait (ms)** block set to **1000** and a **Stop Button** (a green button with "OK" and "LTF" labels).
- The loop terminates with a **Close.vi** block.
- An **Simple Error Handler.vi** block is connected to the error chain.

**Front Panel:**

The front panel, titled "Arduino Read TMP36 - Scratch...", shows the results of the measurements in a grey-themed interface:

- Voltage [V]:** A numeric indicator showing **0.72**.
- Temperature [C]:** A numeric indicator showing **21.8**.
- A **Stop** button with a red square icon.



# Thermistor Temperature Sensor

# Thermistor



A thermistor is an electronic component that changes resistance to temperature - so-called Resistance Temperature Detectors (RTD). It is often used as a temperature sensor.



Our Thermistor is a so-called NTC (Negative Temperature Coefficient). In a NTC Thermistor, resistance decreases as the temperature rises.

There is a **non-linear relationship** between resistance and excitement. To find the temperature we can use the following equation (**Steinhart-Hart equation**):

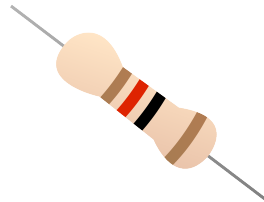
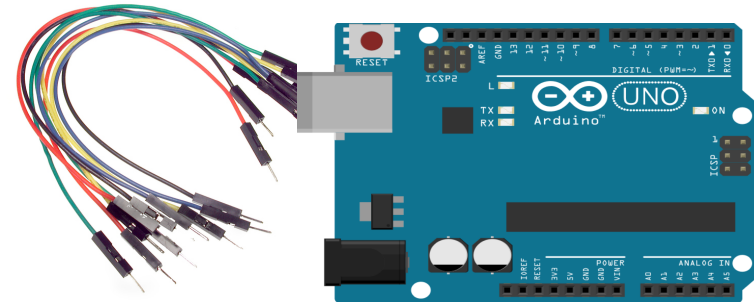
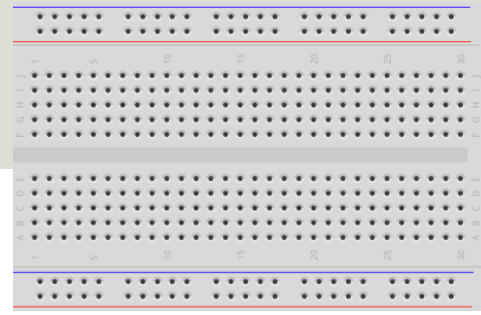
$$\frac{1}{T} = A + B \ln(R) + C (\ln(R))^3$$

where  $A, B, C$  are constants given below [Wikipedia]

$A = 0.001129148, B = 0.000234125$  and  $C = 8.76741E - 08$

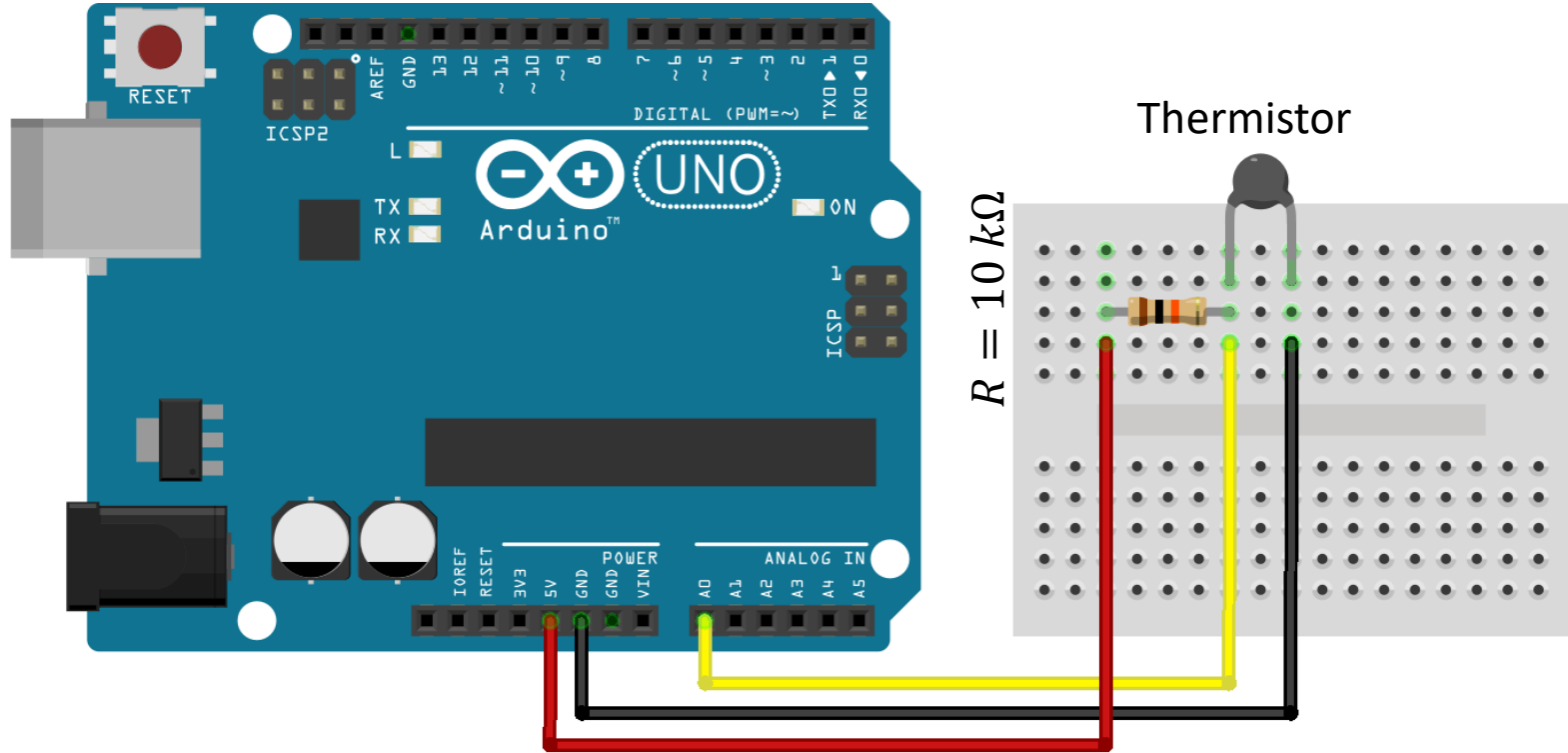
# Hardware

- Arduino
- Breadboard
- Thermistor 10K (Temperature Sensor)
- Wires (Jumper Wires)
- Resistor 10 k $\Omega$



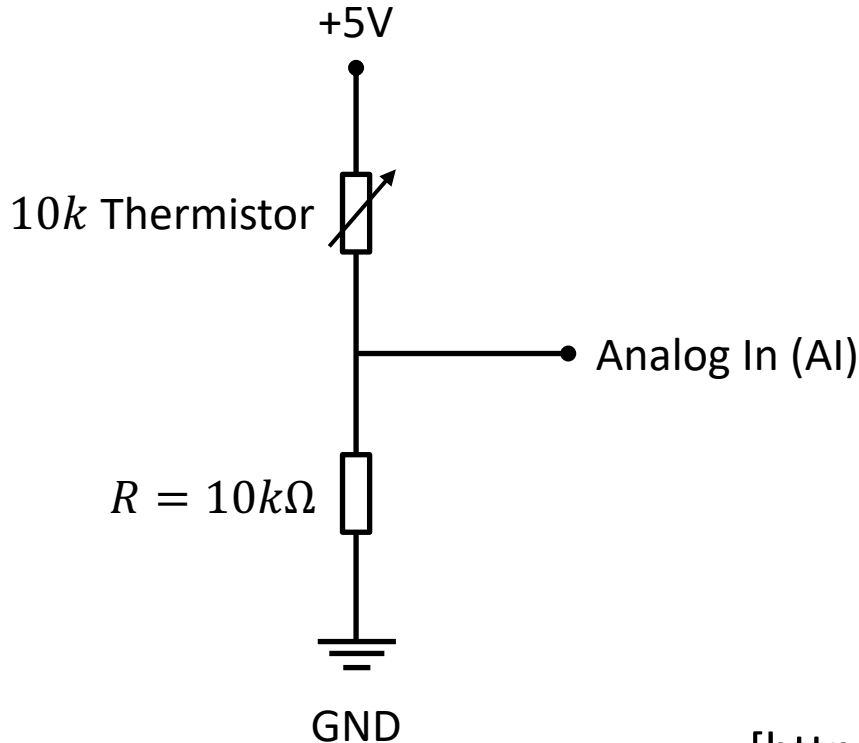


# Wiring



# Voltage Divider

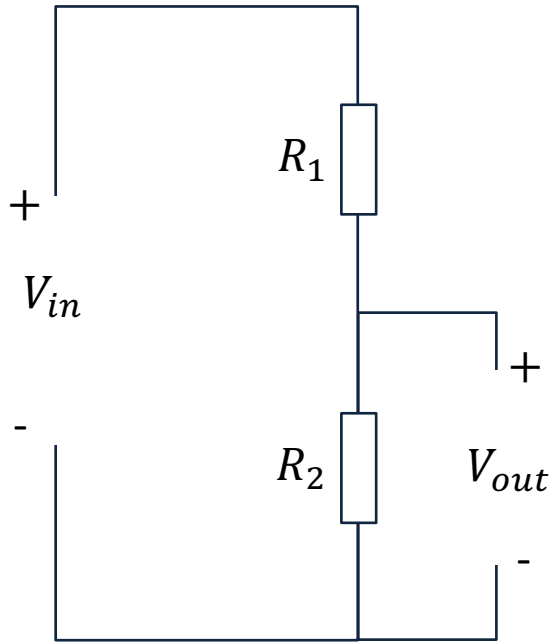
The wiring is called a “Voltage divider”:



[[https://en.wikipedia.org/wiki/Voltage\\_divider](https://en.wikipedia.org/wiki/Voltage_divider)]

# General Voltage Divider

We want to find  $V_{out}$



Formula:

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2}$$

# Voltage Divider for our System

Voltage Divider Equation:

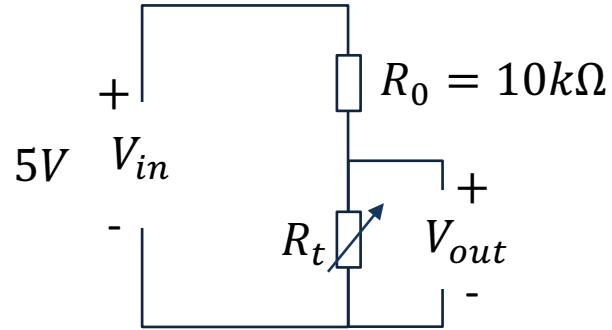
$$V_{out} = V_{in} \frac{R_t}{R_0 + R_t}$$

We want to find  $R_t$ :

$$R_t = \frac{V_{out}R_0}{V_{in} - V_{out}}$$

Steps:

1. We wire the circuit on the Breadboard and connect it to the DAQ device
2. We measure  $V_{out}$  using the DAQ device
3. We calculate  $R_t$  using the Voltage Divider equation
4. Finally, we use Steinhart-Hart equation for finding the Temperature



$R_t$  - 10k Thermistor. This varies with temperature. From Datasheet we know that  $R_t = 10k\Omega @25^\circ\text{C}$

# Steinhart-Hart Equation

To find the Temperature we can use Steinhart-Hart Equation:

$$\frac{1}{T_K} = A + B \ln(R) + C (\ln(R))^3$$

This gives:

$$T_K = \frac{1}{A + B \ln(R) + C (\ln(R))^3}$$

Where the Temperature  $T_K$  is in Kelvin

$A, B$  and  $C$  are constants

$$A = 0.001129148$$

$$B = 0.000234125$$

$$C = 0.0000000876741$$

The Temperature in degrees Celsius will then be:

$$T_C = T_K - 273.15$$

# Pseudo Code

1. Get  $V_{out}$  from the DAQ device (Arduino UNO)
2. Calculate  $R_t = \frac{V_{out}R_0}{V_{in}-V_{out}}$
3. Calculate  $T_K = \frac{1}{A+B \ln(R_t)+C(\ln(R_t))^3}$
4. Calculate  $T_C = T_K - 273.15$
5. Present  $T_C$  in the User Interface

# Pseudo Code

```
float Vin = 5;  
float Ro=10000;  
float Rt = (Vout*Ro)/(Vin-Vout);  
  
//Steinhart constants  
float A = 0.001129148;  
float B = 0.000234125;  
float C = 0.0000000876741;  
  
//Steinhart-Hart Equation  
float TempK = 1 / (A + (B * ln(Rt)) + (C * ln(Rt)**3));  
  
//Convert from Kelvin to Celsius  
float TempC = TempK - 273.15;
```

# LabVIEW

The image displays the LabVIEW development environment for a project titled "Thermistor Example2.vi".

**Block Diagram (Left Panel):**

- The diagram starts with a "COM6" port selection.
- An "Open.vi" block is connected to a "Serial" dropdown menu.
- The data flow continues to an "Analog Read.vi" block, which is configured with an "AI Channel" of 0 and a sampling rate of 16.
- The output of "Analog Read.vi" is connected to a "Steinhart-Hart Equation.vi" block, which also receives a "Steinhart-Hart" coefficient input.
- The result of the equation is displayed in a "Temperature [C]" indicator, showing a value of 1.23.
- The process concludes with a "Close.vi" block and an "Error" indicator.
- A "Wait (ms)" block is set to 100 ms, and a "Stop Button" (OK) is present at the bottom.

**Front Panel (Right Panel):**

- The front panel features a "Temperature [C]" indicator displaying the value 25.4.
- A "Stop" button with a red square icon is located below the indicator.



# LabVIEW - Steinhart-Hart Equation

Steinhart-Hart Equation.vi Block Diagram

File Edit View Project Operate Tools Window Help

15pt Application Font

Search

Alternative 1: Formula Node

Something may be easier to do in a text-based programming language

```
float Vin = 5;
float Ro=10000;
float Rt = (Vout*Ro)/(Vin-Vout);

//Steinhart constants
float A = 0.001129148;
float B = 0.000234125;
float C = 0.0000000876741;

//Steinhart-Hart Equation
float TempK = 1 / (A + (B * Ln(Rt)) + (C * Ln(Rt)**3));

//Convert from Kelvin to Celsius
float TempC = TempK - 273.15;
```

$1/T = A + B*(\ln R) + C*(\ln R)^3 \rightarrow T = 1 / (A + B*(\ln R) + C*(\ln R)^3)$

# LabVIEW - Steinhart-Hart Equation

## Alternative 2: Pure LabVIEW Code

Steinhart-Hart Equation - LabVIEW Code.vi Block Diagram

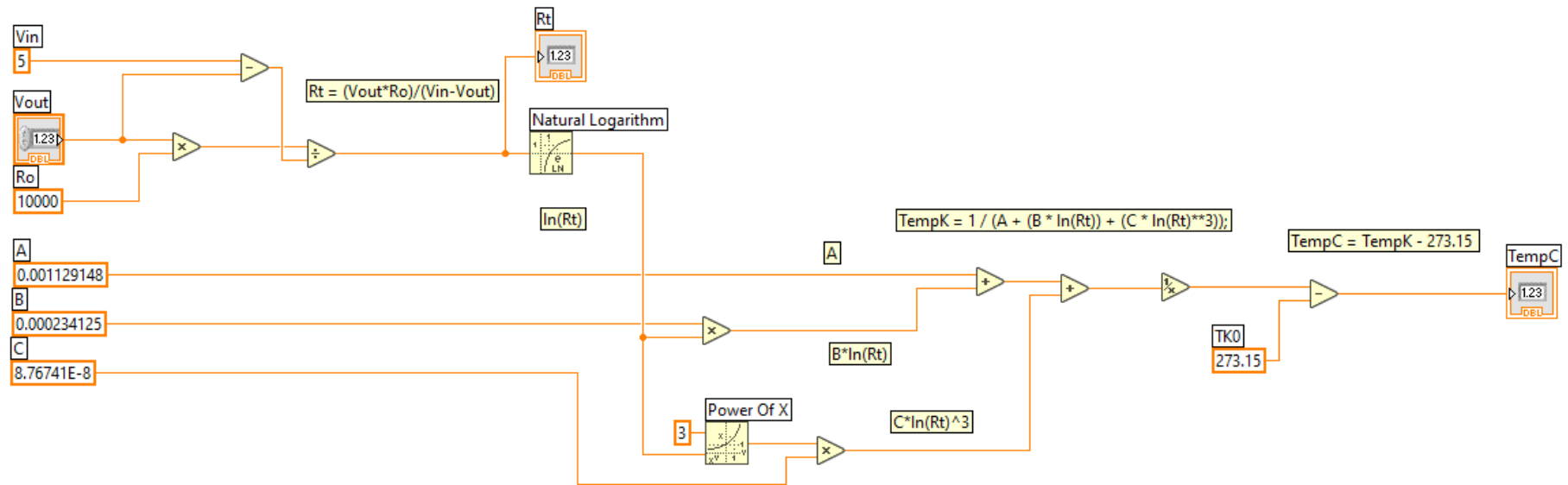
File Edit View Project Operate Tools Window Help

15pt Application Font

Search

Steinhart

$$1/T = A + B*(\ln R) + C*(\ln R)^3 \rightarrow T = 1 / (A + B*(\ln R) + C*(\ln R)^3)$$



# Summary

- This Tutorial has shown how we can use Arduino in combination with the LabVIEW Programming environment
- “LabVIEW LINX Toolkit” is an add-on for LabVIEW which makes it possible to program the Arduino device using LabVIEW
- In that way we can create Data Logging Applications, etc. without the need of an expensive DAQ device
- If you in addition use the “LabVIEW Community Edition” (free for non-commercial use) you get a very low-cost DAQ/Datalogging System!
- You can also easily add features for logging data to Files or a Database System like SQL Server, or an OPC Server

# Hans-Petter Halvorsen

University of South-Eastern Norway

[www.usn.no](http://www.usn.no)

E-mail: [hans.p.halvorsen@usn.no](mailto:hans.p.halvorsen@usn.no)

Web: <https://www.halvorsen.blog>

