

<https://www.halvorsen.blog>



Part 2: PWM and Push Buttons

LabVIEW LINX and Raspberry Pi

LabVIEW + LabVIEW LINX Toolkit + Raspberry Pi

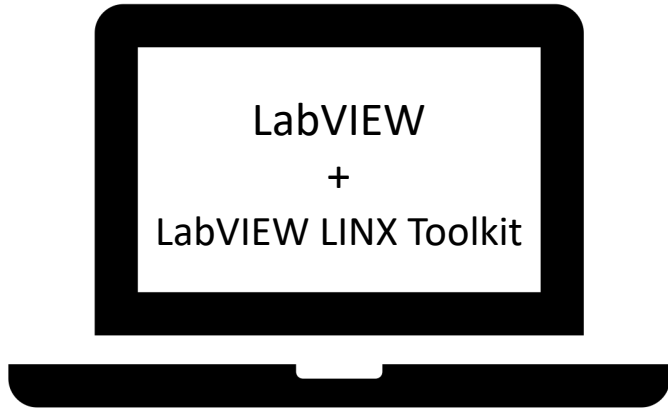
Hans-Petter Halvorsen

Table of Contents

- Raspberry Pi and LabVIEW LINX
- LabVIEW Raspberry Pi Project
- Digital Out and Digital In
- Push Buttons
- PWM
- Deployment

LabVIEW + LabVIEW LINX Toolkit

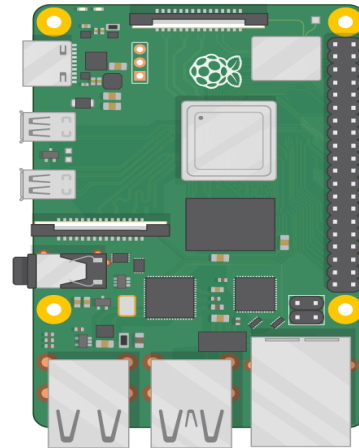
PC



Ethernet
or Wi-Fi



Raspberry Pi

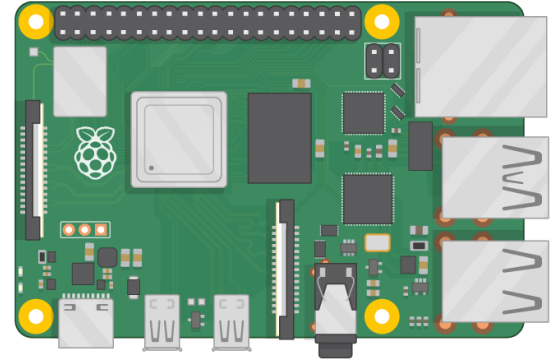
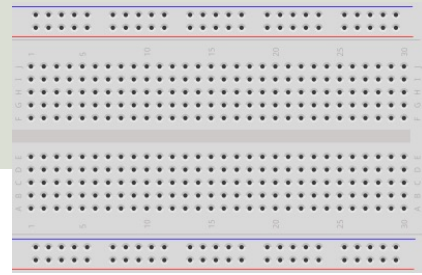
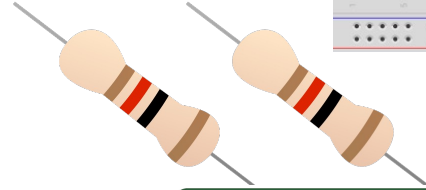
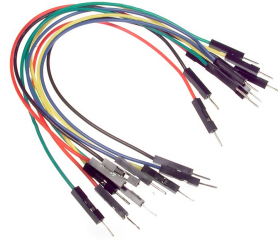


GPIO



Hardware

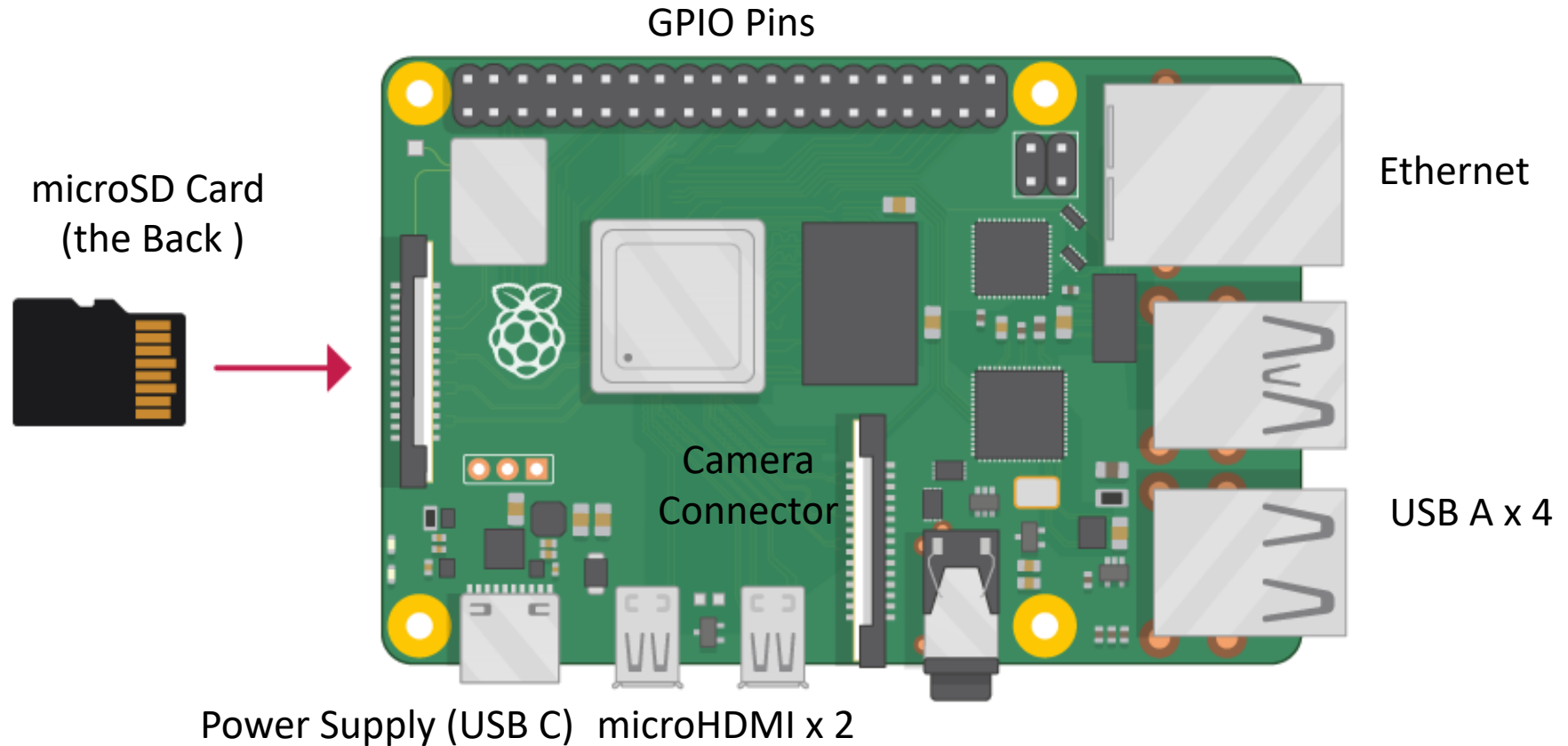
- Raspberry Pi
- Breadboard
- Wires (Jumper Wires)
- Resistors ($R = 270\Omega$)
- LED, Push Button



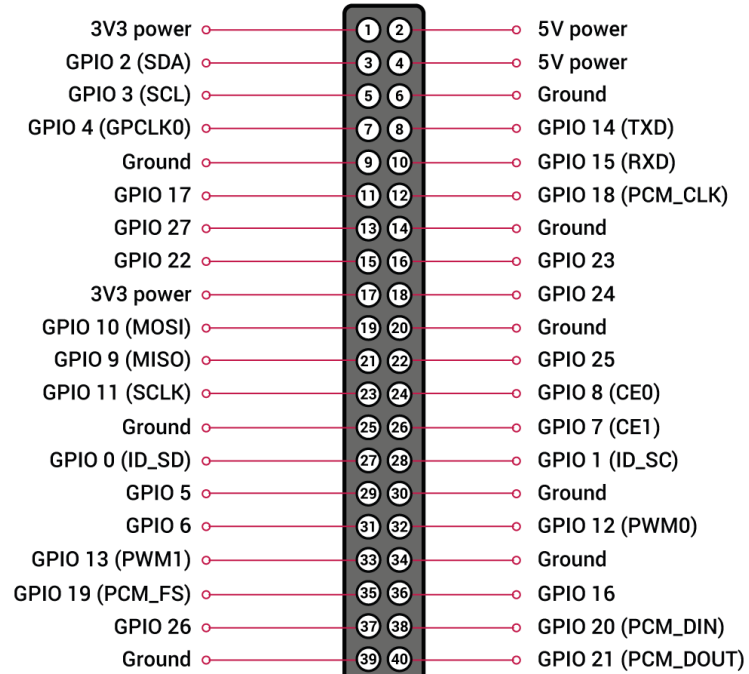
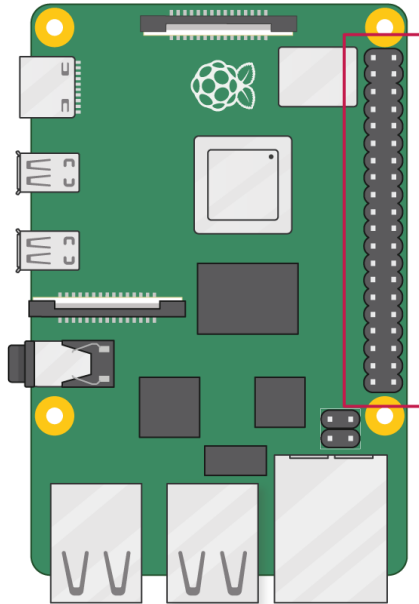


Raspberry Pi and LabVIEW LINUX

Raspberry Pi



GPIO



A powerful feature of the Raspberry Pi is the GPIO (general-purpose input/output) pins. The Raspberry Pi has a 40-pin GPIO header as seen in the image

Raspberry Pi OS

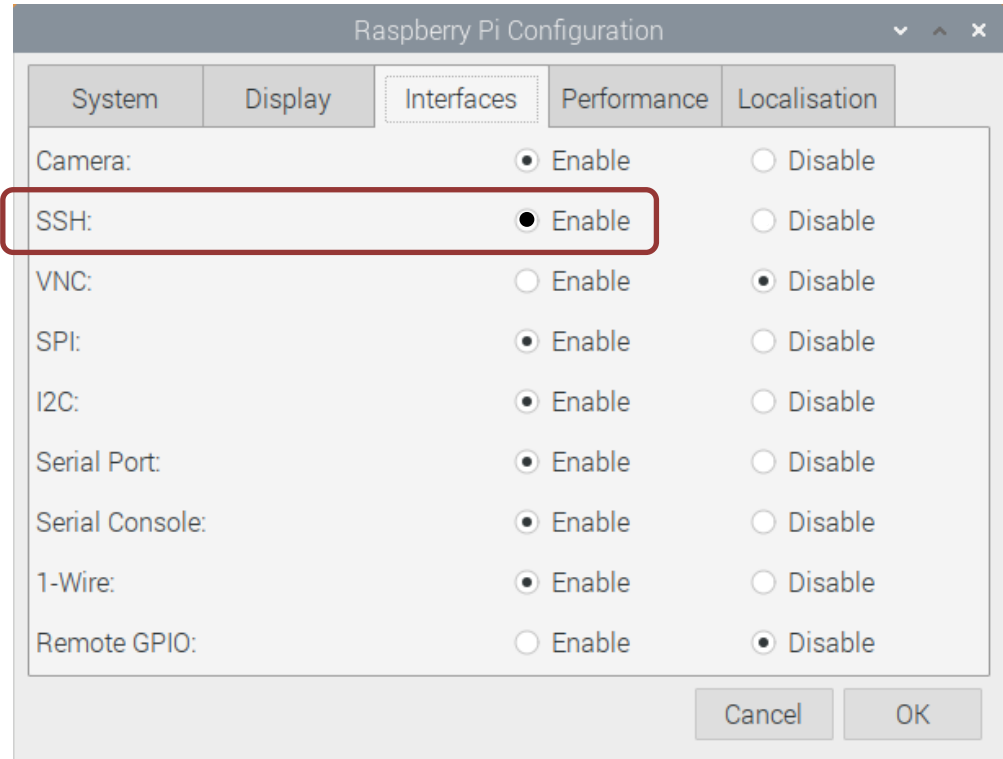
- In order to make your Raspberry Pi up and running you need to install an Operating System (OS)
- The OS for Raspberry Pi is called “**Raspberry Pi OS**” (previously known as Raspbian)
- Raspberry Pi runs a version of an operating system called **Linux** (Windows and macOS are other operating systems).
- To install the necessary OS, you need a **microSD** card
- Then you use the “**Raspberry Pi Imager**” in order to download the OS to the microSD card.

<https://www.raspberrypi.org/software/>

Raspberry Pi Configuration

You need to Enable **SSH** so you can remotely get access to the Raspberry Pi from your Computer

SSH, also known as Secure Shell or Secure Socket Shell, is a Network Protocol that gives users, particularly system administrators, a secure way to access a computer over an unsecured network.



Mobile Wi-Fi hotspot on Windows10

The screenshot shows the Windows 10 Settings application. The left sidebar is open to 'Network & Internet' > 'Mobile hotspot'. The main pane shows the following settings:

- Share my Internet connection with other devices:** On (toggle switch)
- Share my Internet connection from:** Wi-Fi (dropdown menu)
- Share my Internet connection over:** Wi-Fi (radio button selected), Bluetooth (radio button unselected)
- Network name:** Windows10HPH
- Network password:** [Empty text box]
- Network band:** Any available
- Edit:** Button
- Devices connected:** 1 of 8
- | Device name | IP address | Physical address (MAC) |
|-------------|------------------|------------------------|
| raspberrypi | [Empty text box] | [Empty text box] |
- Power saving:** Off (toggle switch)

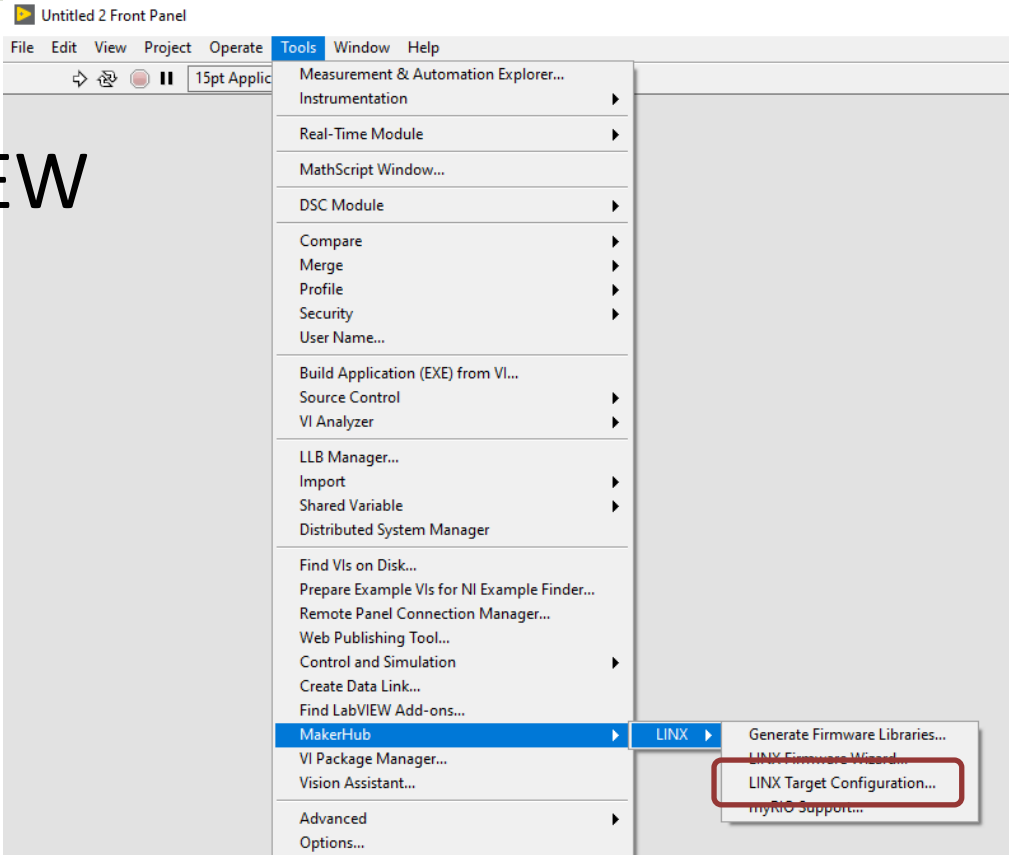
A red warning message is displayed: "You're sharing your connection over the 5 GHz network band. The network might not appear on devices that can only connect over the 2.4 GHz band."

You can connect your PC and the Raspberry Pi together using an Ethernet cable or using Wi-Fi.

I configured Mobile Wi-Fi hotspot on my Windows10 PC. Then I connected my Raspberry Pi to this Wi-Fi network

Raspberry Pi LINX Configuration

LabVIEW



Raspberry Pi LINUX Configuration

LINX Target Configuration

Connection

Installation

Network Settings

Target Info

Raspberry Pi

Hostname or IP: raspberrypi

Username: pi

Password: *****

Connect

Additional installation information: LabVIEWMakerHub.com

Not Connected

Successfully connected to the target.

OK

or Wi-Fi

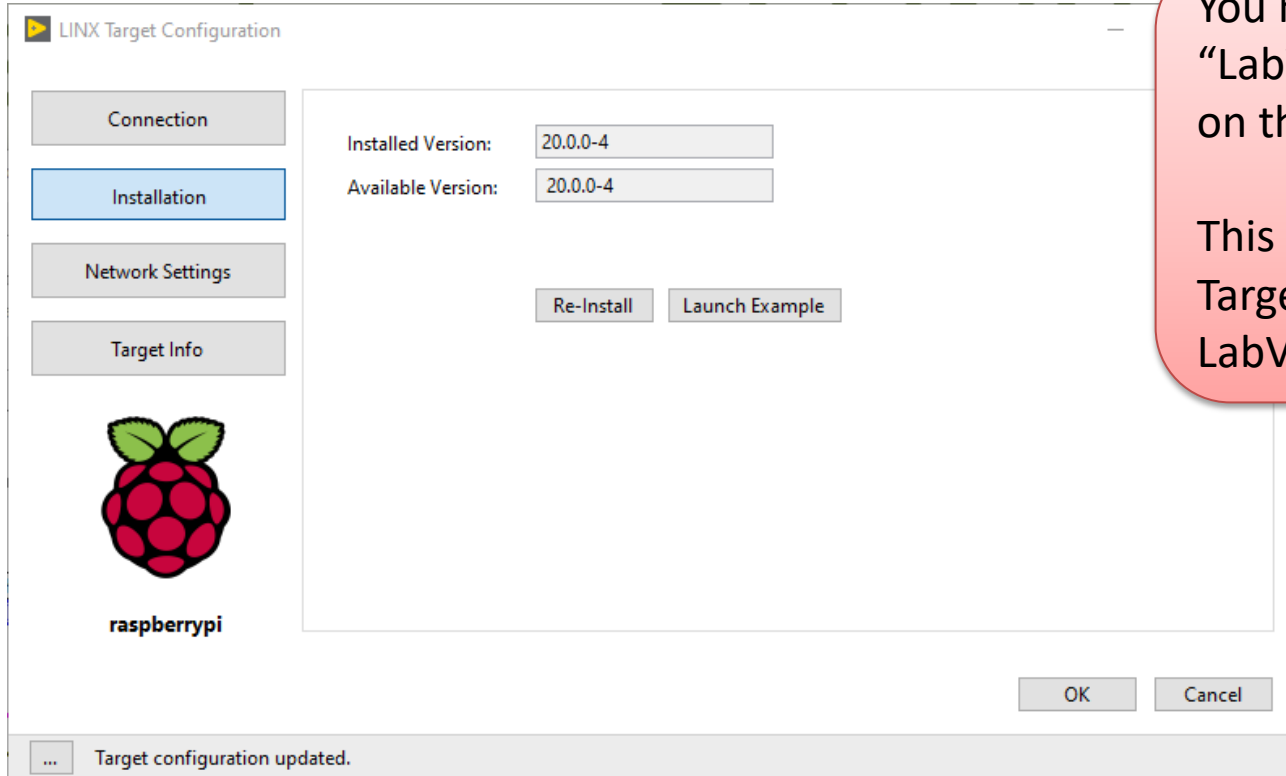
Connect your device via ethernet.
Use a monitor and mouse to enable SSH.
Username and **password** need to have sudo privileges on the target - 'pi' and 'raspberry' are the default.

Make sure you can connect to the Raspberry Pi from your PC where you have LabVIEW installed.

You can use Wi-Fi or an Ethernet cable

My Configuration: On my Windows PC I configured a Wi-Fi Mobile hotspot. On the Raspberry Pi I connected to this Wi-Fi hotspot

Raspberry Pi LINUX Configuration



You need to install
“LabVIEW Runtime Engine”
on the Raspberry Pi device.

This is done from the LINX
Target Configuration in
LabVIEW on your PC

<https://www.halvorsen.blog>



LabVIEW Raspberry Pi Project

Hans-Petter Halvorsen

[Table of Contents](#)

Create your Raspberry Pi Project

LabVIEW

File Operate Tools Help

LabVIEW™ 2020

Create Project

Open Existing

Recent Project Templates

- Blank Project

All Recent Files

- C:\Users\hansha\OneDrive\Development\RPIProject.lvproj
- C:\Temp\LabVIEW Raspberry Pi\LabVIEW
- LabVIEW Raspberry Pi Application.lvproj
- Database Script Generator.lvproj
- LabVIEW State Machine.lvproj
- Weather Station.lvproj
- Vision System for Pool Table Games.lvproj

Find Drivers and Add-ons

Connect to devices and expand the functionality of LabVIEW.

Community and Support

Participate in the discussion forums or request technical support.

Get Support

Find support documentation. Learn about service requests. [READ MORE](#)

Create Project

Choose a starting point for the project:

- All
- Templates
- Sample Projects
- Desktop
- Real-Time

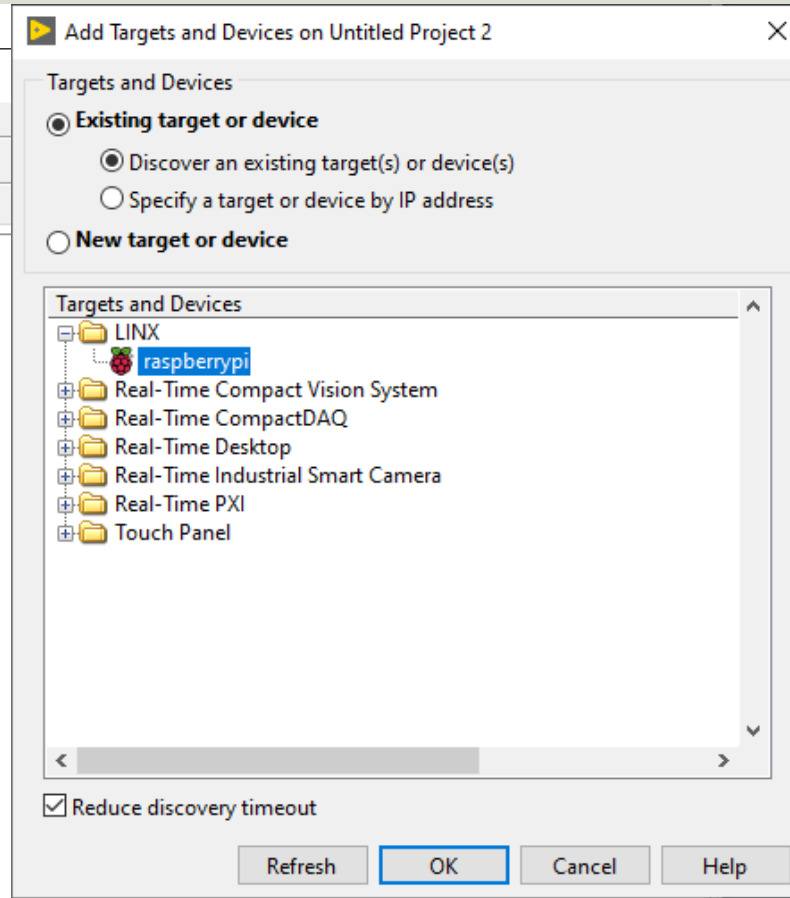
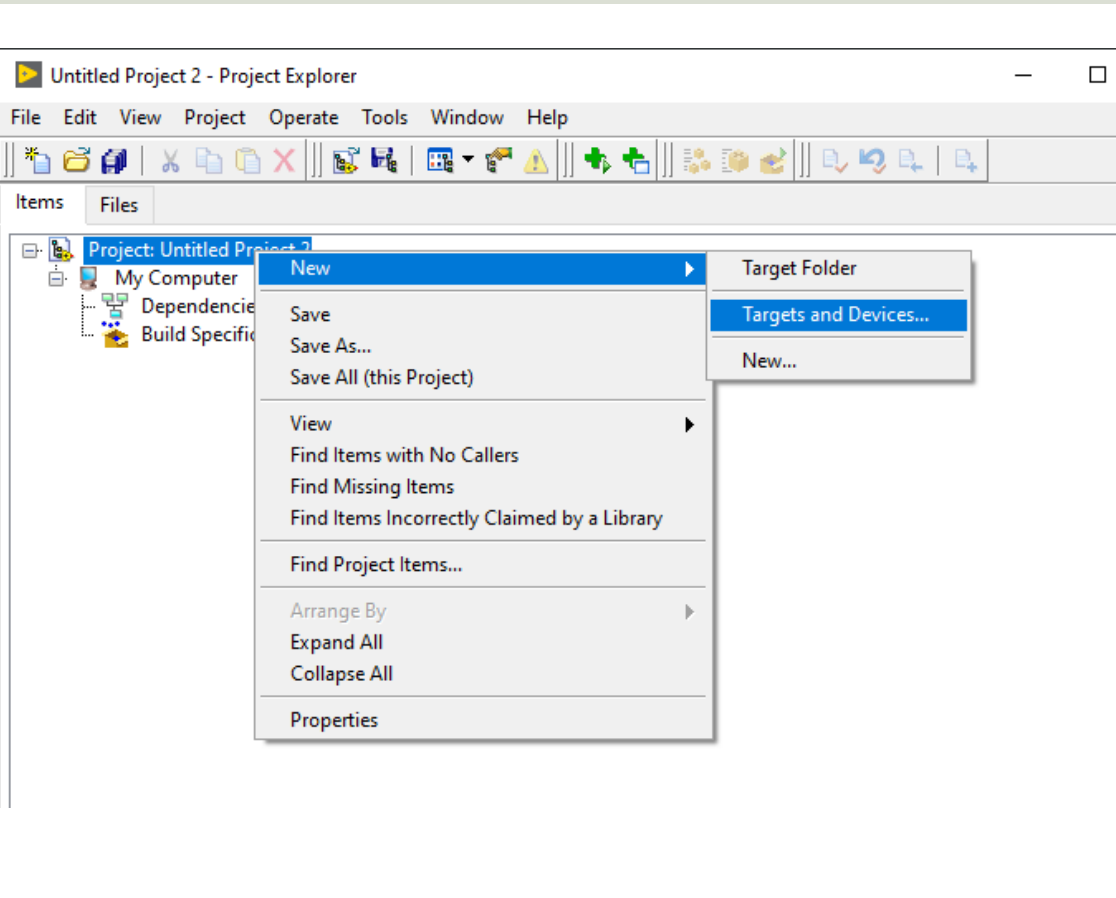
- Blank Project** *Templates*
Creates a blank project.
- Blank VI** *Templates*
Creates a blank VI.
- Simple State Machine** *Templates*
Facilitates defining the execution sequence for sections of code. [More Information](#)
- Channeled Message Handler** *Templates*
Uses channels to facilitate multiple sections of code running in parallel and sending data between them. [More Information](#)
- Queued Message Handler** *Templates*
Uses queue refs to facilitate multiple sections of code running in parallel and sending data between them. [More Information](#)
- Actor Framework** *Templates*
Creates an application that consists of multiple, independent tasks that communicate with each other. This template makes extensive use of LabVIEW classes. [More Information](#)
- Finite Measurement** *Sample Projects*
Acquires a finite measurement and provides options for exporting the measurement to disk. This sample project is based on the Simple State Machine template. [More Information](#)
- Continuous Measurement and Logging** *Sample Projects*
Acquires measurements continuously and logs them to disk. This sample project is based on the Queued Message Handler template. [More Information](#)
- Feedback Evaporative Cooler** *Sample Projects*
Implements an evaporative cooler with hot-swappable hardware, controllers, and user interfaces. This sample project is based on the Actor Framework template. [More Information](#)
- Instrument Driver Project** *Templates*

Additional Search

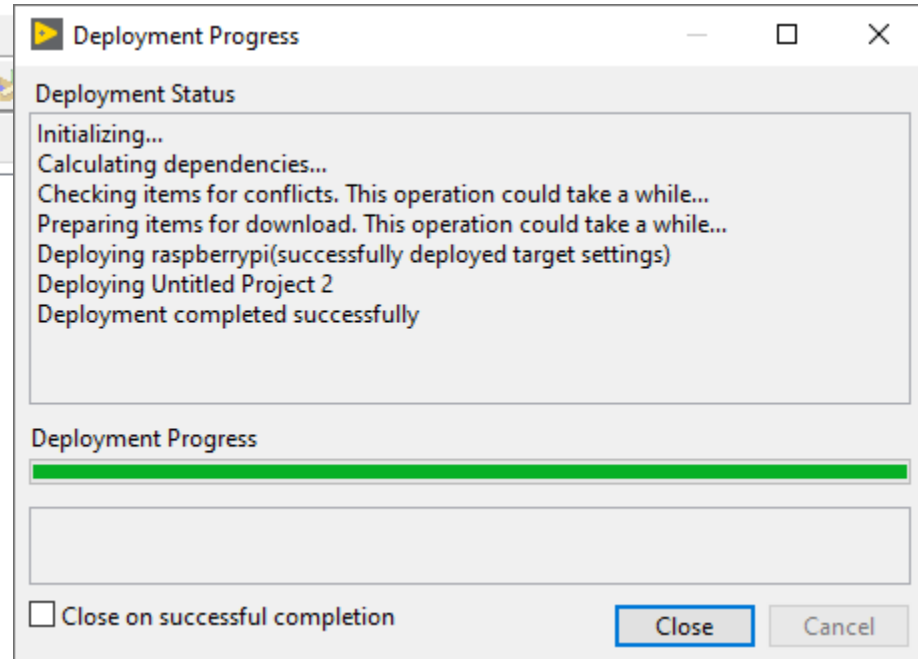
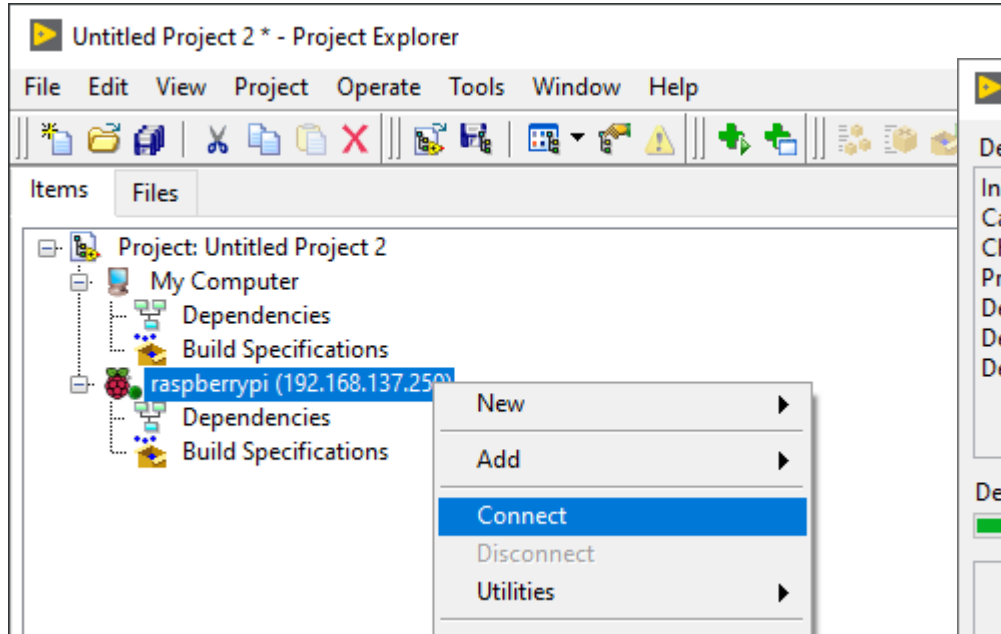
Keyword

Finish Cancel Help

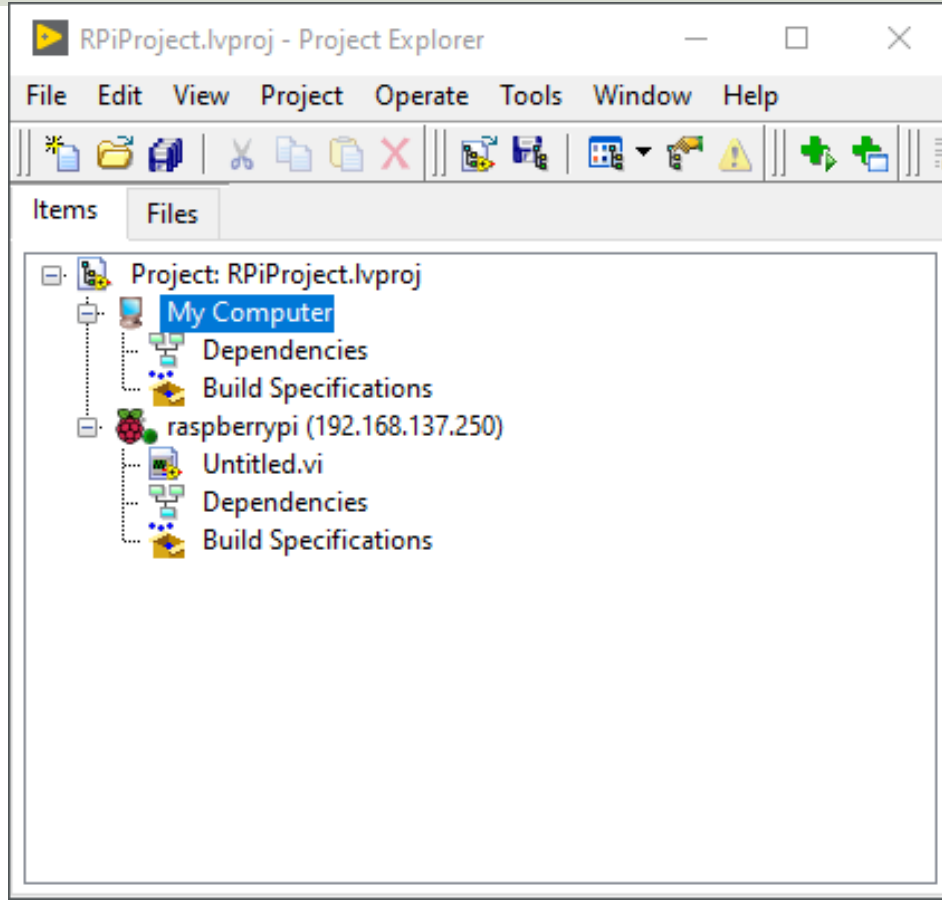
Create your Raspberry Pi Project



Create your Raspberry Pi Project



LabVIEW Project Explorer

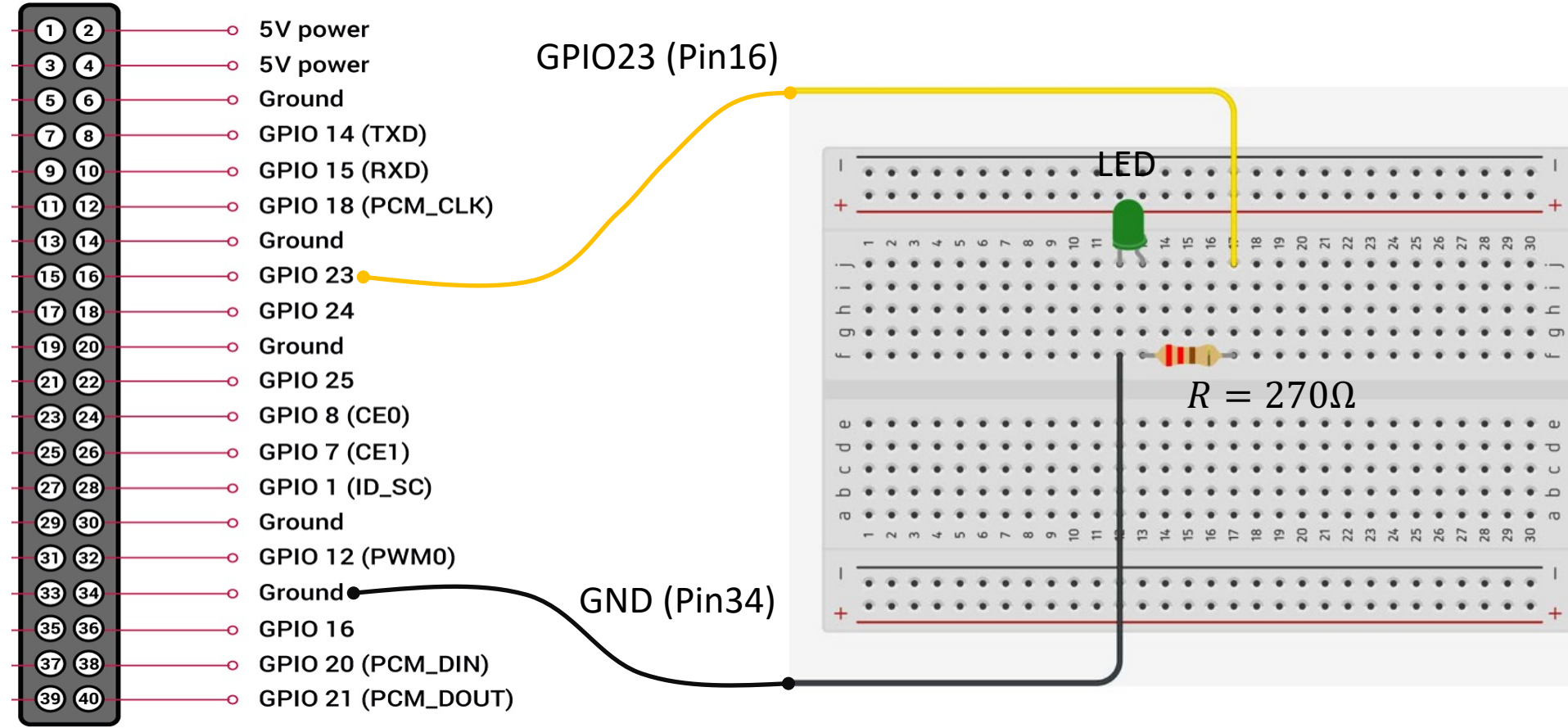


You are now ready to start creating LabVIEW Code that control the GPIO pins on the Raspberry Pi device



Digital Out (DO)

LED Wiring



LED ON/OFF - LabVIEW Example

The image displays the LabVIEW development environment for a project titled "LED.vi". The main window shows the "Block Diagram" for "LED.vi Block Diagram on LabVIEWPi.lvproj/raspberrypi".

Block Diagram Details:

- The diagram is enclosed in a **While Loop**.
- On the left, an **Open.vi** subVI is connected to a **Local I/O** control.
- The **While Loop** contains:
 - A **DO Channel** control (value 16) connected to a **Digital Write.vi** subVI.
 - An **LED Value** control (value 1) connected to a **Red LED** indicator.
 - A **Digital Write 1 Chan** subVI.
 - A **Stop Button** control connected to a **Stop** indicator.
- On the right, a **Close.vi** subVI is connected to a **Simple Error** indicator.
- The **While Loop** is controlled by a **Stop Button** (TF) and a **Stop** indicator.

Front Panel Details:

- The front panel is titled "LED.vi Front Panel on LabVIEWPi.lvproj/raspberrypi".
- It features a **DO Channel** control with a value of 16.
- There is a **Red LED** indicator.
- A **Stop** button is located at the bottom right.

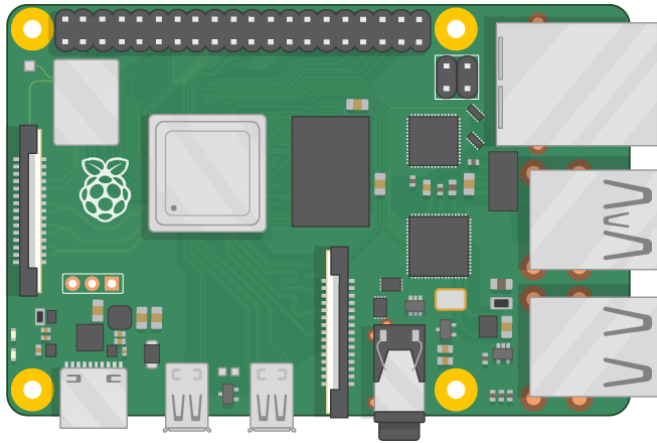


Digital In (DI)

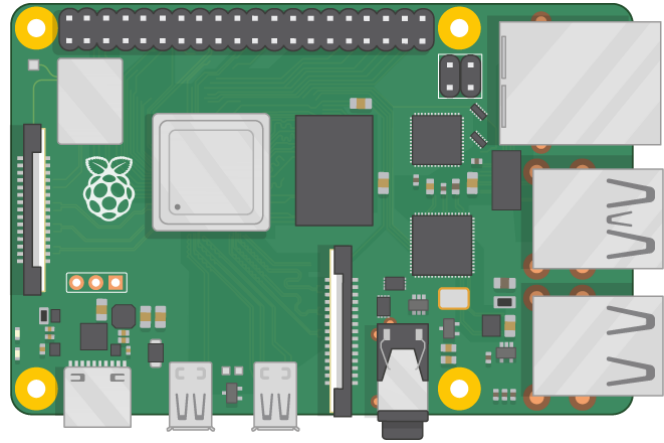
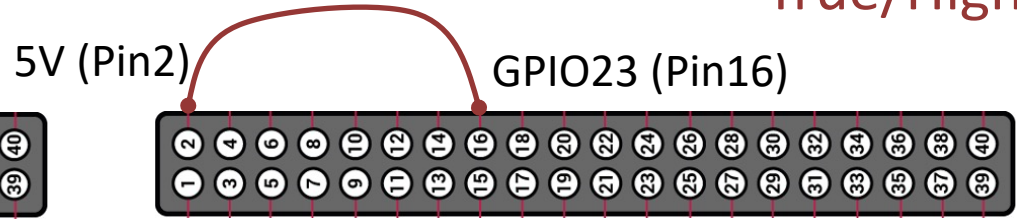
Test of Digital Read

We can test the Digital In (Read) by wiring to GND (False/Low) or 5V (True/High)
GPIO23 (Pin16) is used in this example, but you can of course use another GPIO pin

False/Low



True/High



LabVIEW - Digital Read

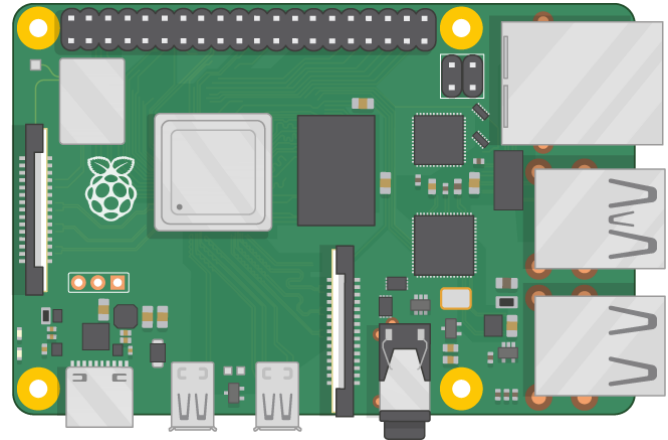
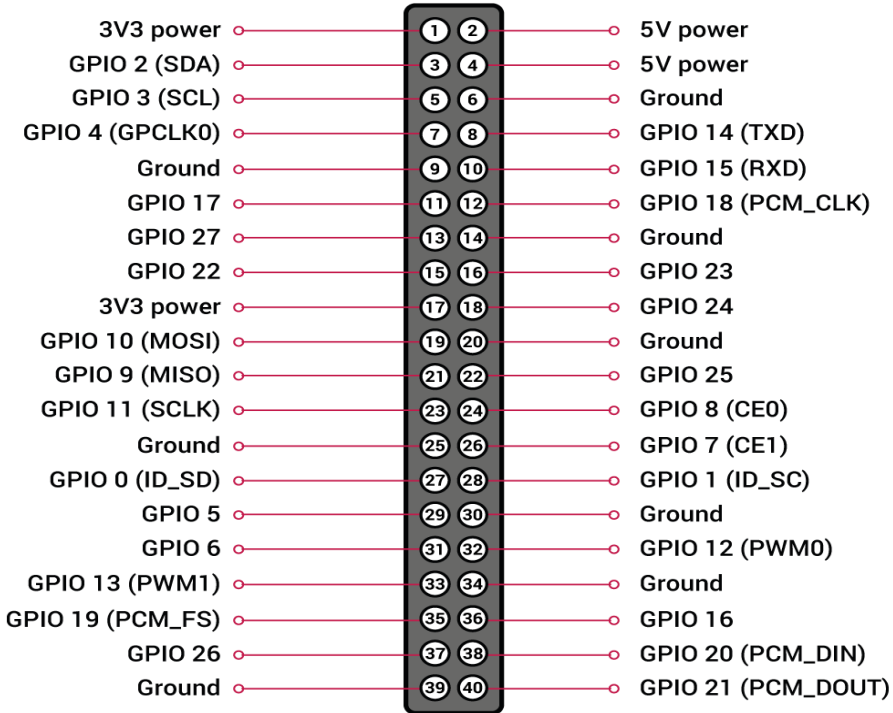
The image displays the LabVIEW development environment for a digital read application. The main window shows the **Block Diagram** for **Digital Read.vi** on a Raspberry Pi. The diagram features a **While Loop** containing the following elements:

- Open.vi**: A subVI that opens the digital read device, connected to a **Local I/O** control.
- DI Channel**: A numeric control set to **16**, which provides the channel number to the **Digital Read.vi** subVI.
- Digital Read.vi**: The core subVI that performs the digital read operation.
- Digital Read 1 Chan**: A numeric indicator that displays the result of the digital read.
- Stop Button**: A push button that, when pressed, sends a signal to a **Stop** indicator (a red circle) and a **Simple Error Handler** (a speech bubble icon).
- Close.vi**: A subVI that closes the digital read device.

The **Front Panel** of the VI is visible on the right, showing a **DI Channel** control with the value **16**, a **Digital Read** indicator (a green circle), and a **Stop** button with a red square.

LabVIEW Digital Write - Read

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



LabVIEW Digital Write - Read

The image displays the LabVIEW interface for a digital write-read application. It is divided into two main sections: the Block Diagram (left) and the Front Panel (right).

Block Diagram (Left): The diagram is titled "Digital Write - Read.vi Block Diagram on LabVIEWPi.lvproj/raspberrypi". It features a "While Loop" containing the following components:

- Open.vi:** A subVI icon for opening the device.
- Local I/O:** A dropdown menu for selecting the device.
- DO Channel:** A numeric control for selecting the Digital Output channel.
- Digital Write.vi:** A subVI for writing a digital signal to the DO channel.
- Boolean:** A Boolean control (checkbox) that determines the output state.
- DI Channel:** A numeric control for selecting the Digital Input channel.
- Digital Read.vi:** A subVI for reading the digital signal from the DI channel.
- Digital Read:** A subVI icon for reading the digital signal.
- Stop Button:** A button that triggers an error cluster.

Front Panel (Right): The front panel is titled "Digital Write - Read.vi Front Panel on LabVIEWPi.lvproj/raspberrypi". It contains the following controls:

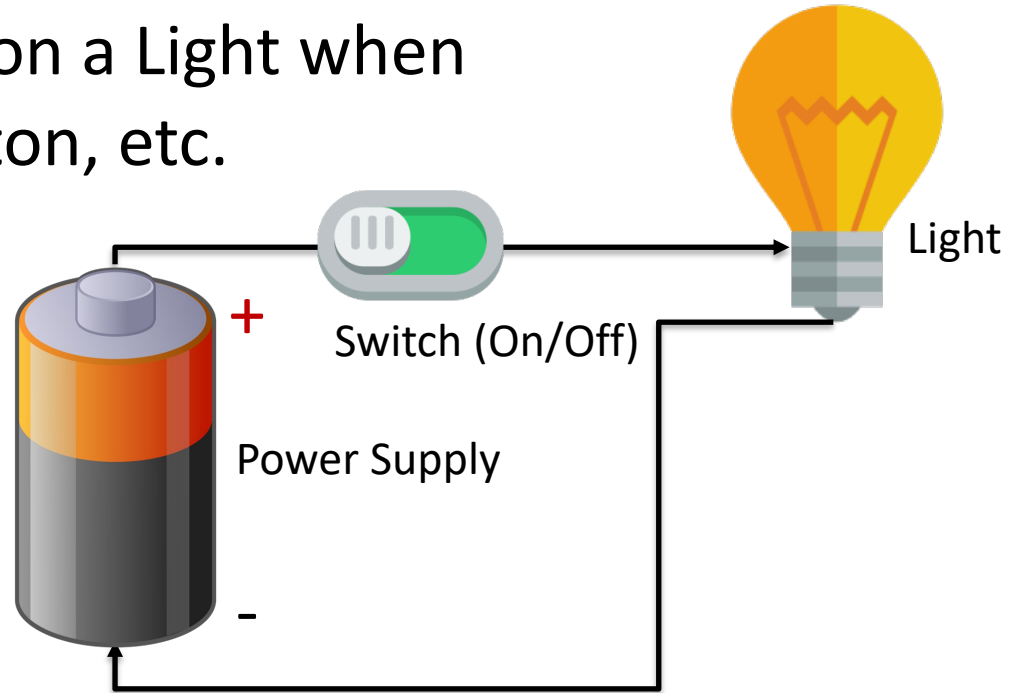
- DO Channel:** A numeric control set to 16.
- DI Channel:** A numeric control set to 18.
- Boolean:** A Boolean control (checkbox) currently set to false.
- Digital Read:** A circular indicator light, currently green.
- Stop:** A red button with a white square, used to stop the operation.



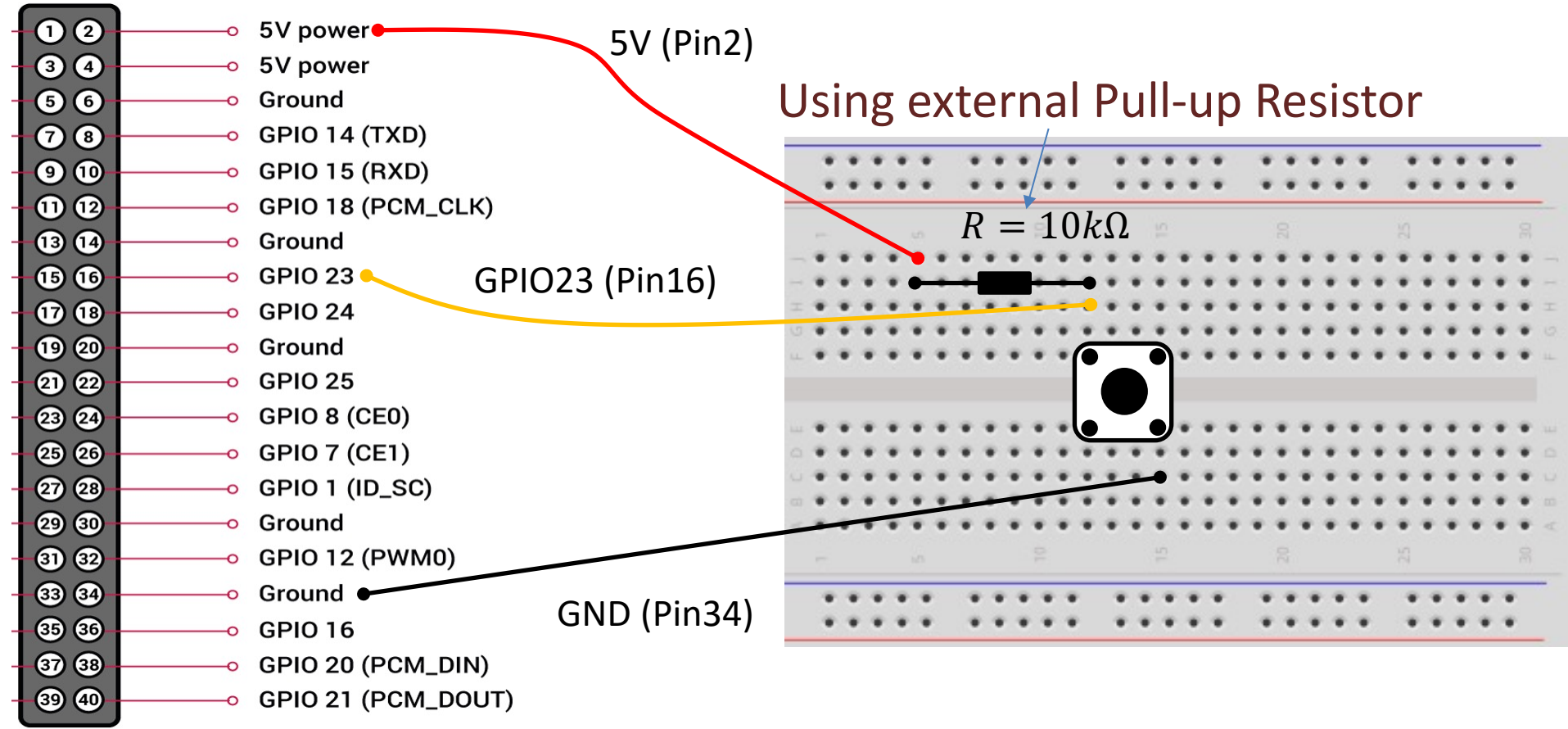
Push Buttons

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)

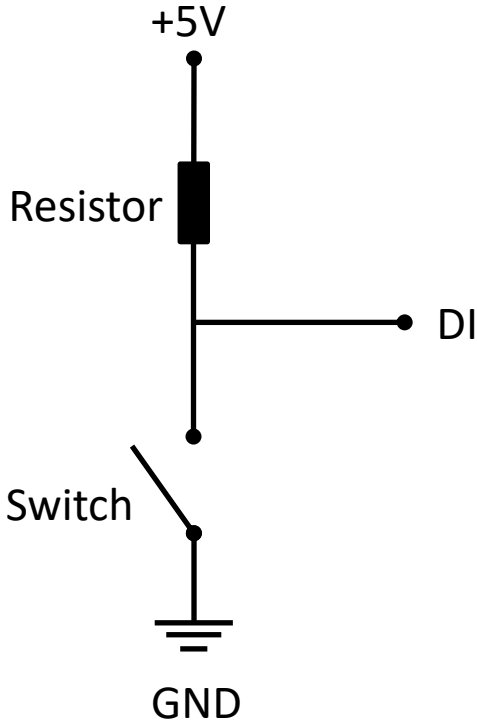


Pull-down/Pull-up Resistor

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.

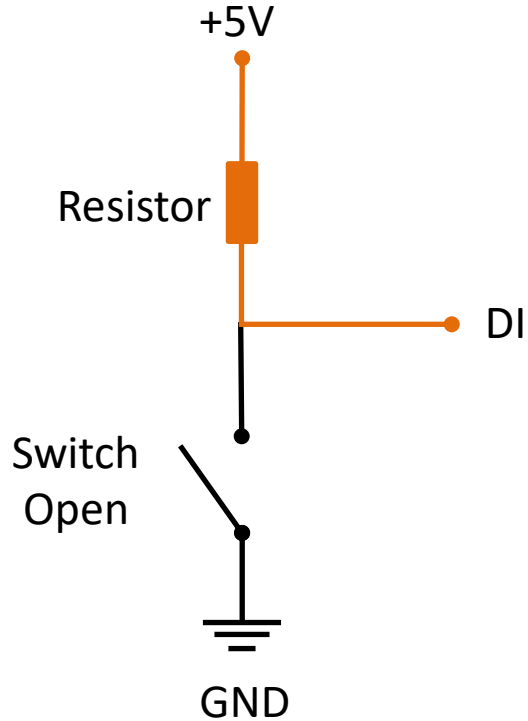
Pull-up Resistor



- 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).

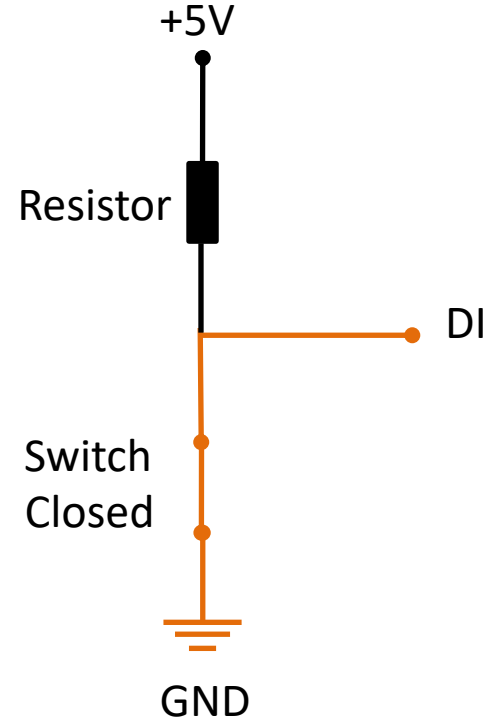
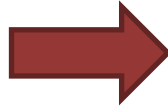
Pull-up Resistor

True/High



False/Low

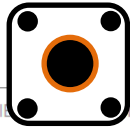
We Push the Button



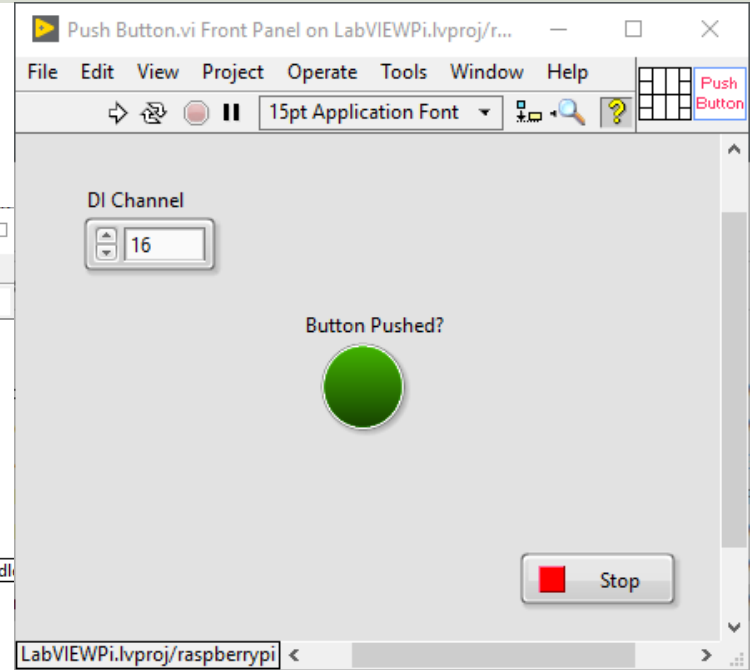
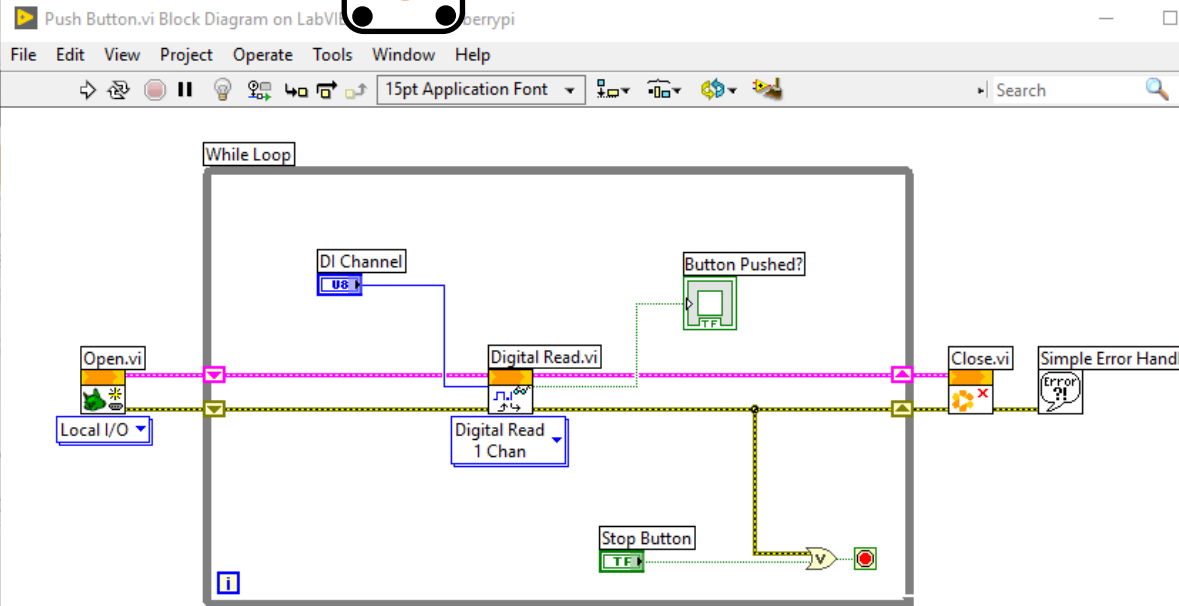
Push Button (Pull-up Resistor)



Button is NOT Pushed => True/High



Button is Pushed => False/Low



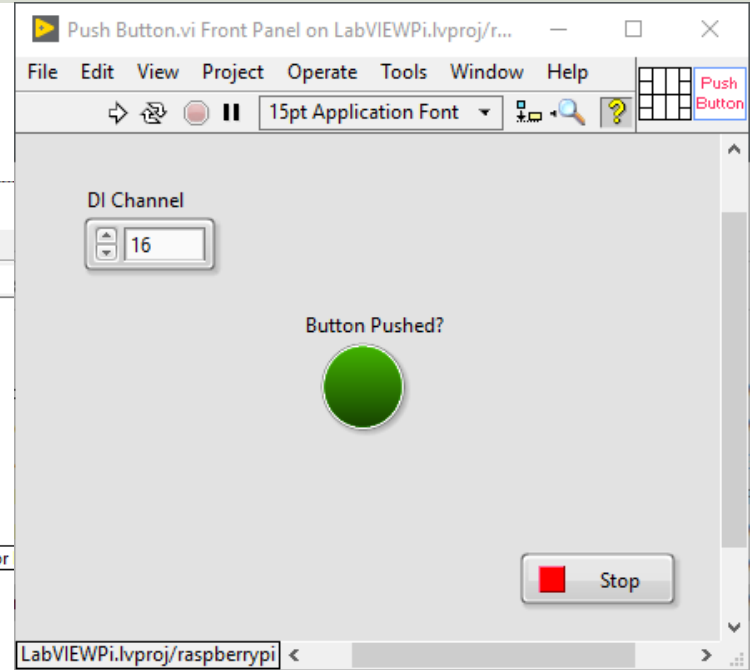
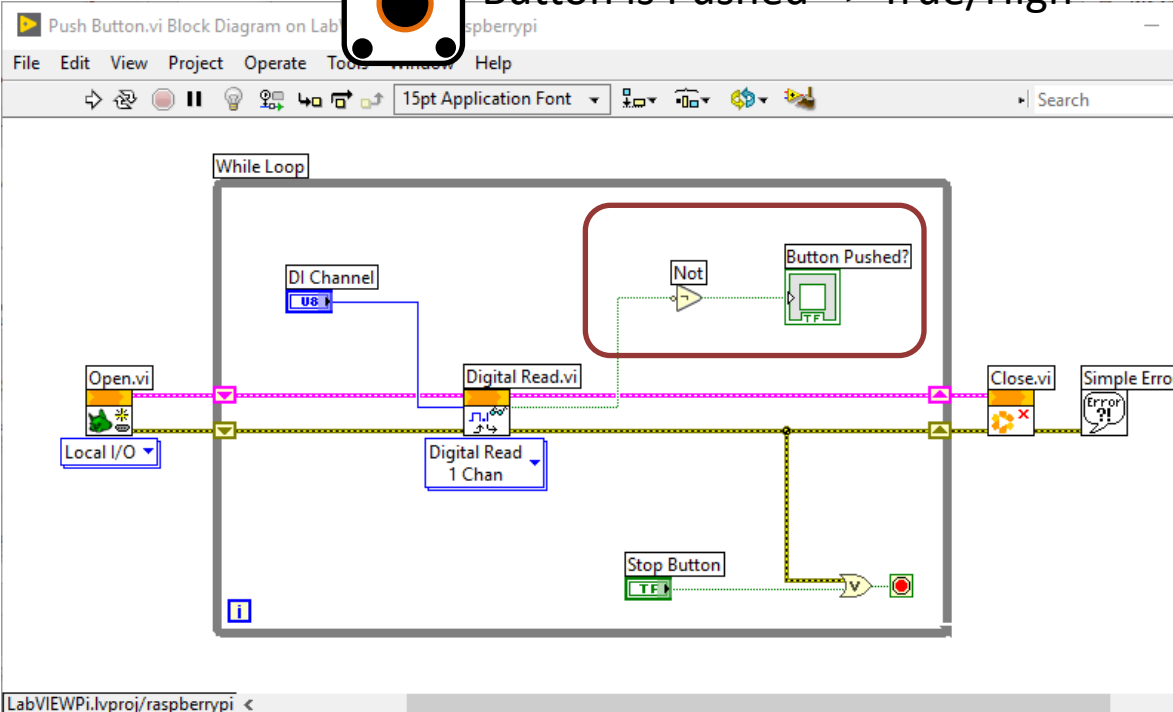
Adding a "NOT" block



Button is NOT Pushed => False/Low



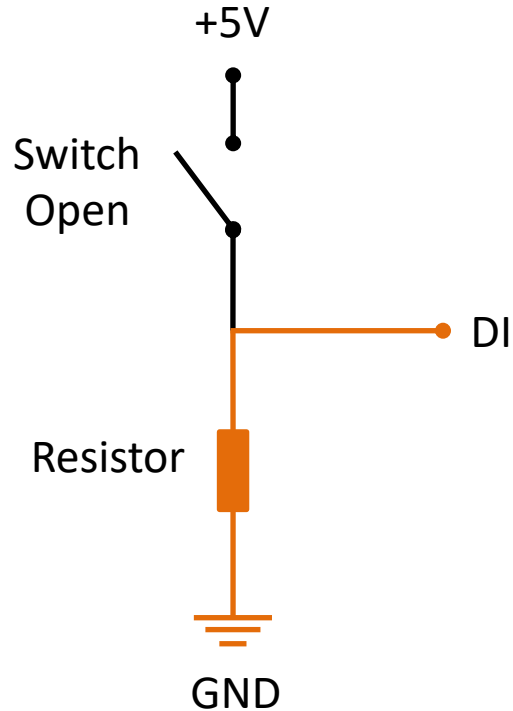
Button is Pushed => True/High



Pull-down Resistor

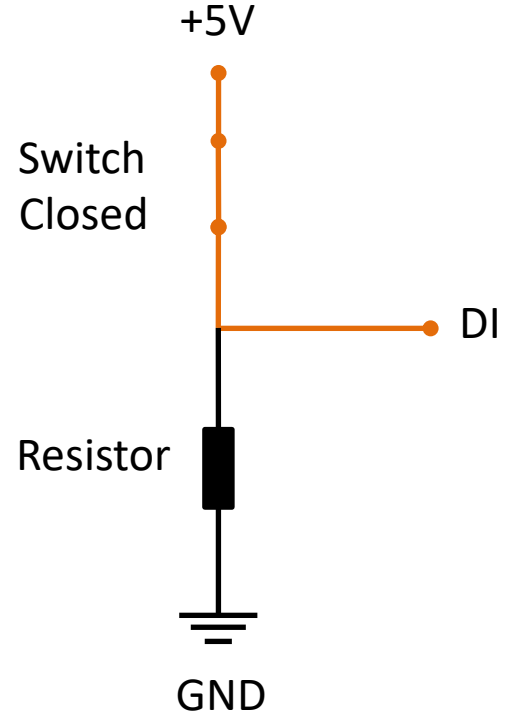
We could also have wired according to a “Pull-down” Resistor

False/Low

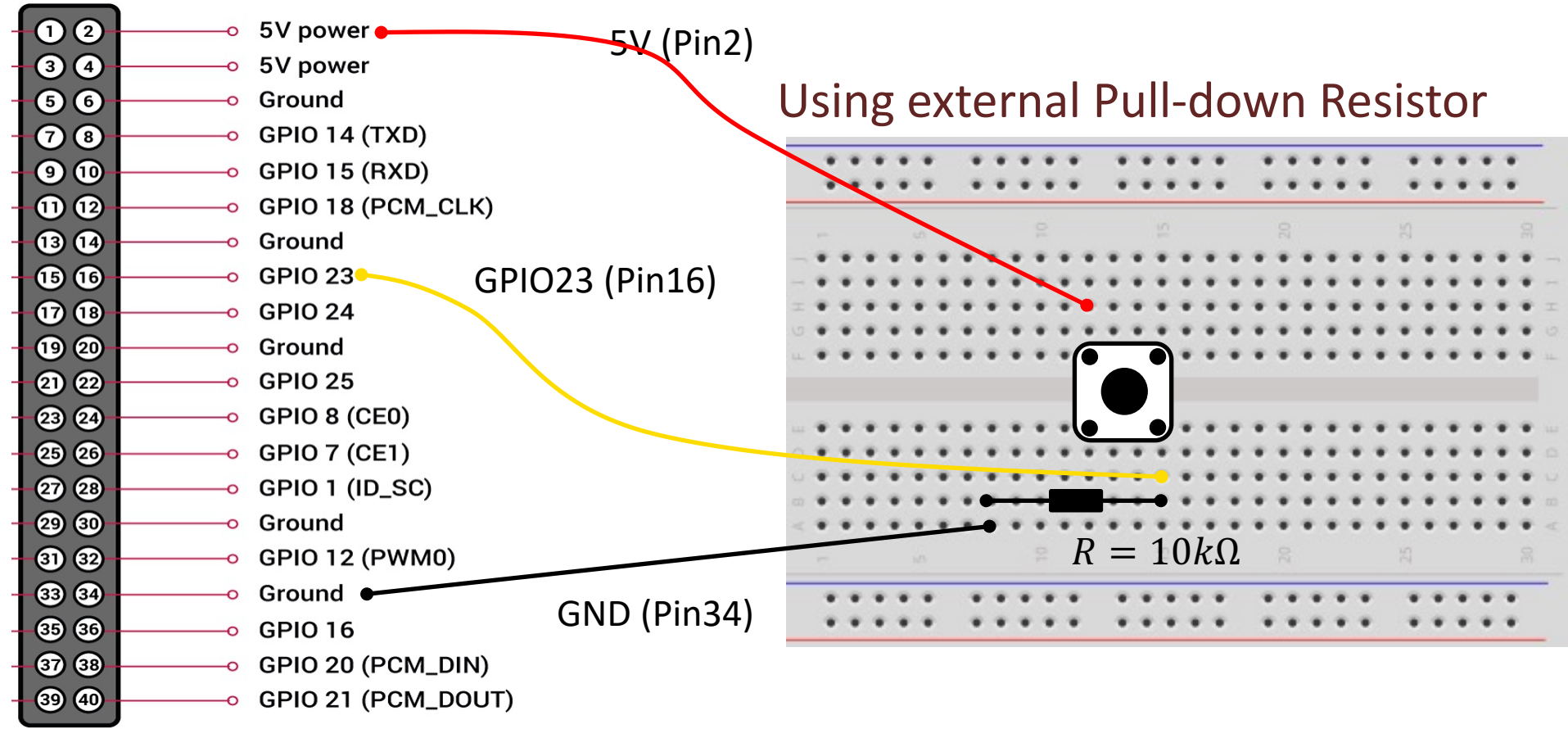


True/High

→
We Push the Button



Wiring (Pull-down Resistor)



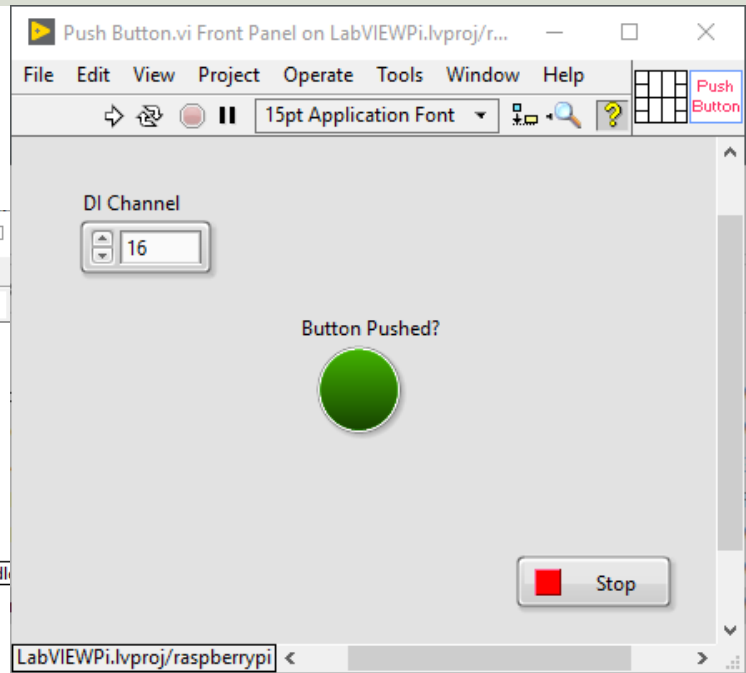
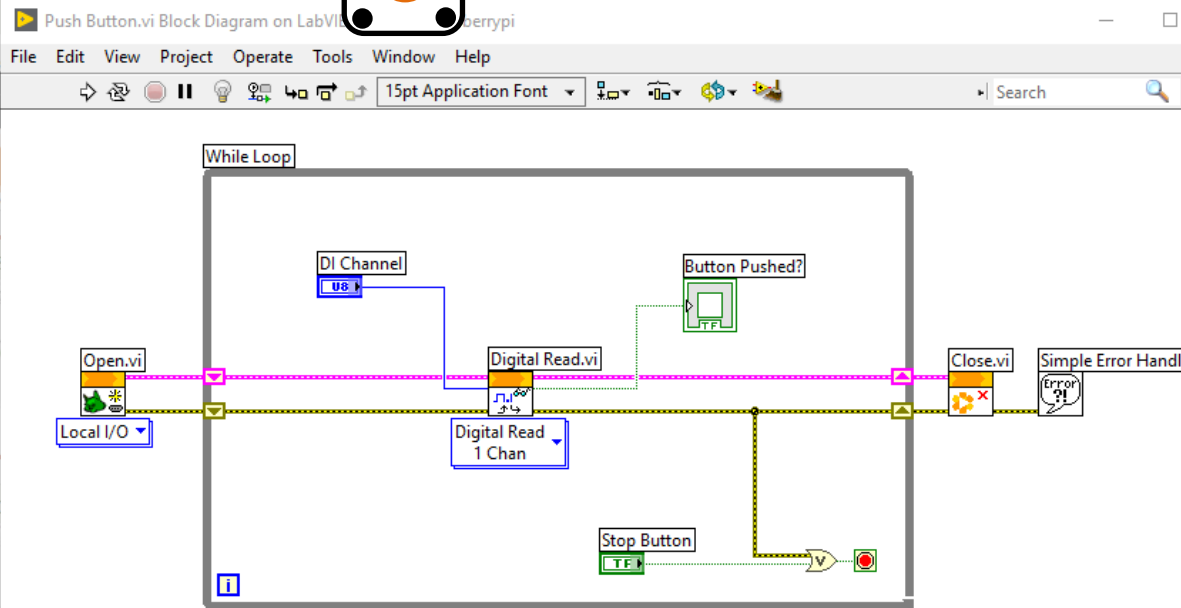
Push Button (Pull-down Resistor)



Button is NOT Pushed => False/Low



Button is Pushed => True/High

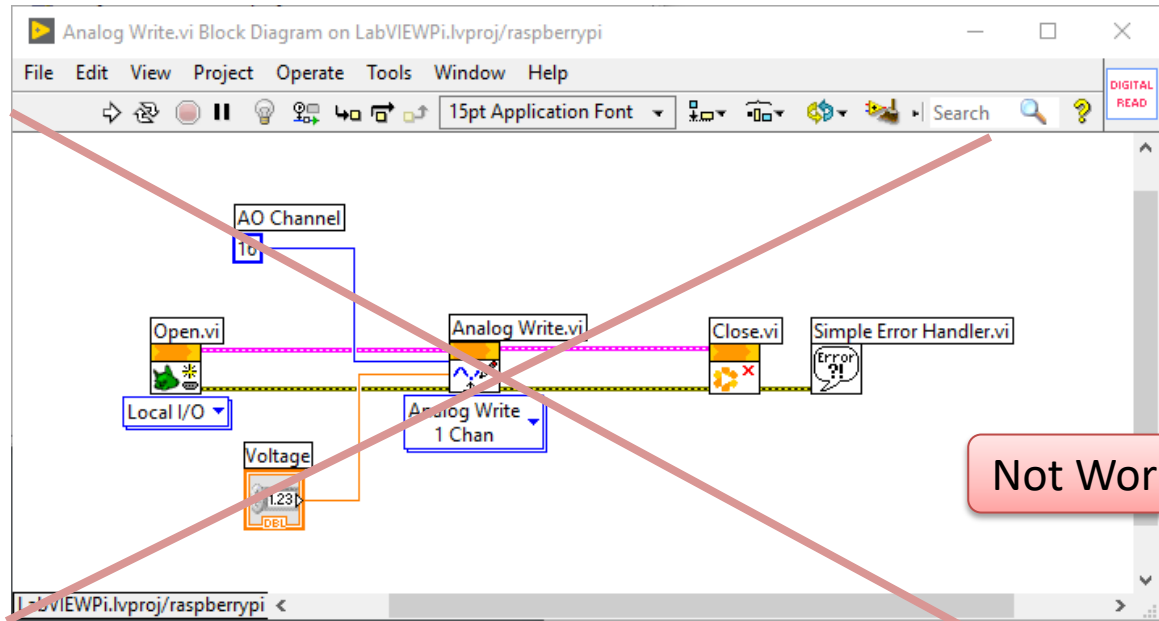




PWM

Analog Out

- Raspberry Pi has no Analog Out
- We need to use Pulse Width Modulation (PWM)

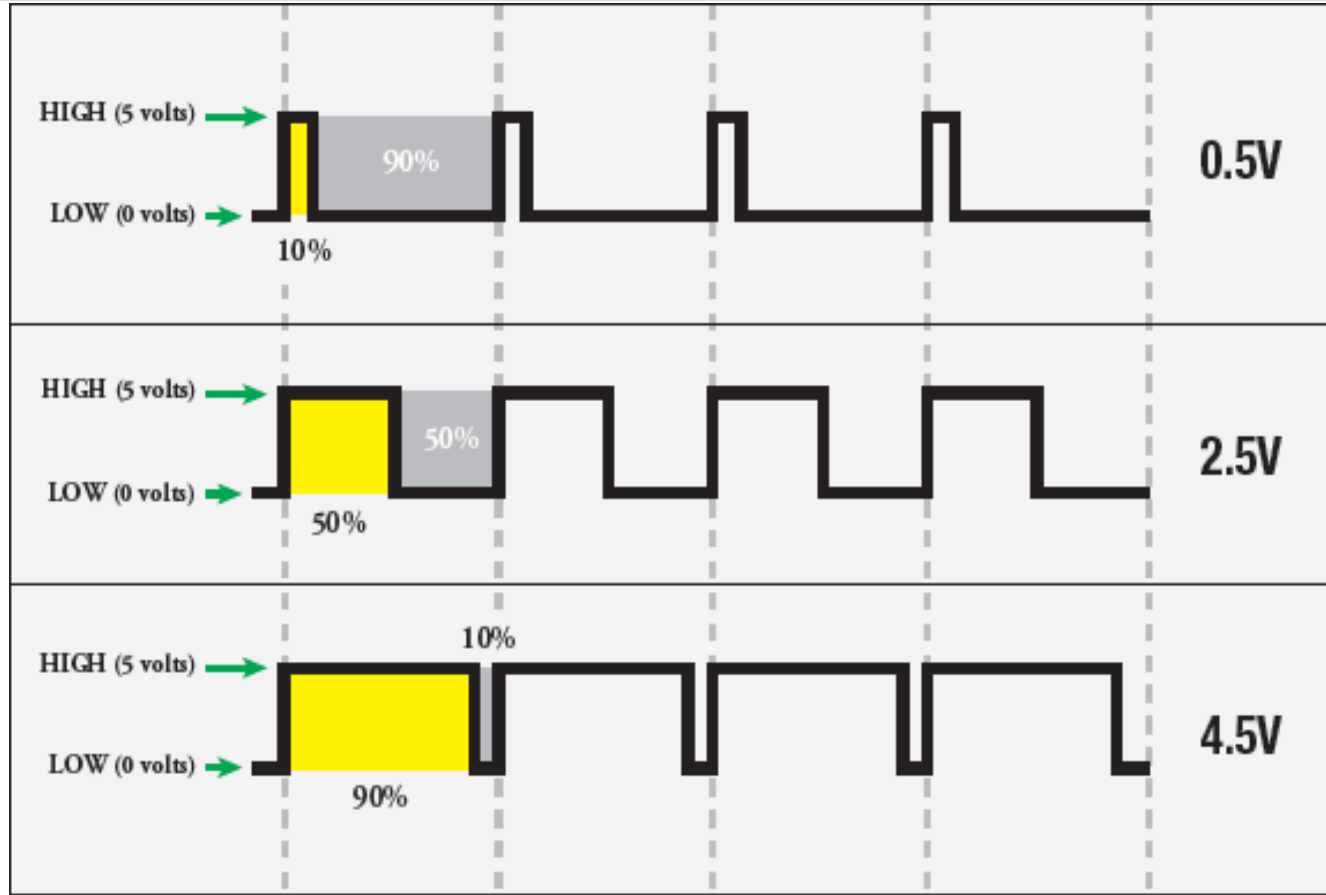


Not Working!

PWM as “Analog Out”

The Raspberry Pi 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

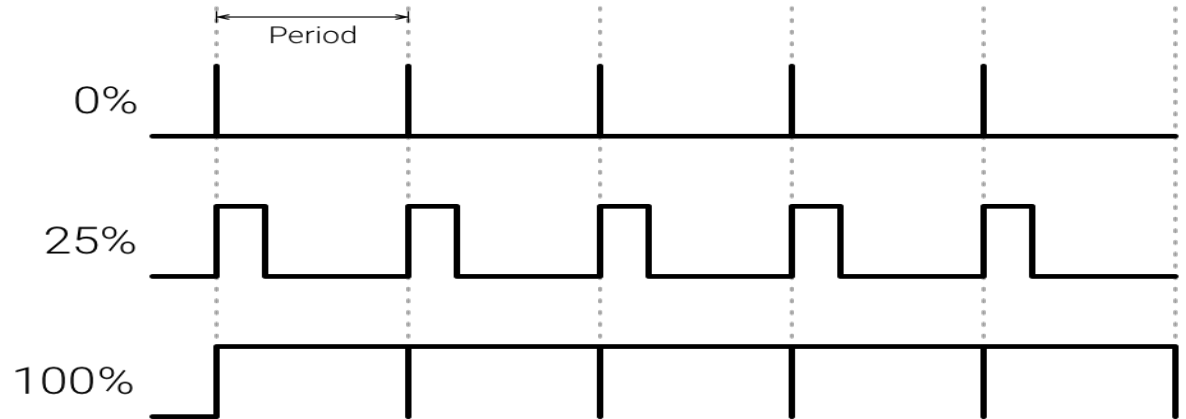
PWM is a digital (i.e., square wave) signal that oscillates according to a given *frequency* and *duty cycle*.

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

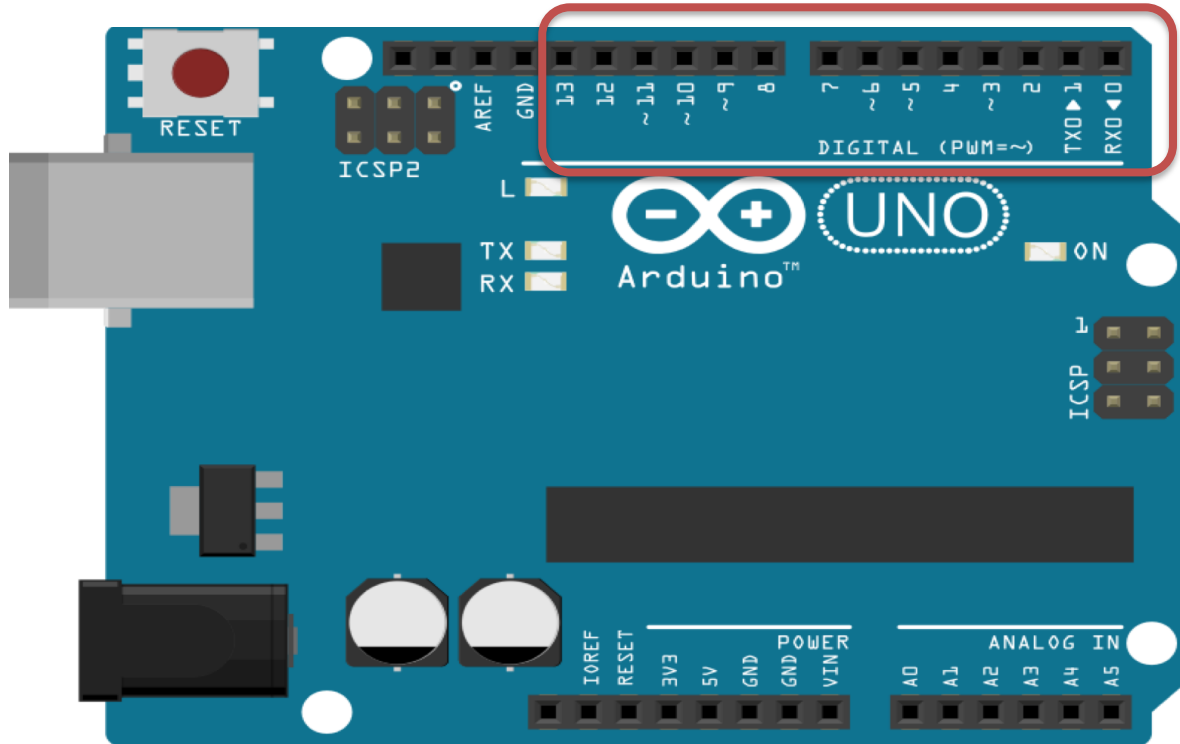
The period is the time each cycle takes and is the inverse of frequency.

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



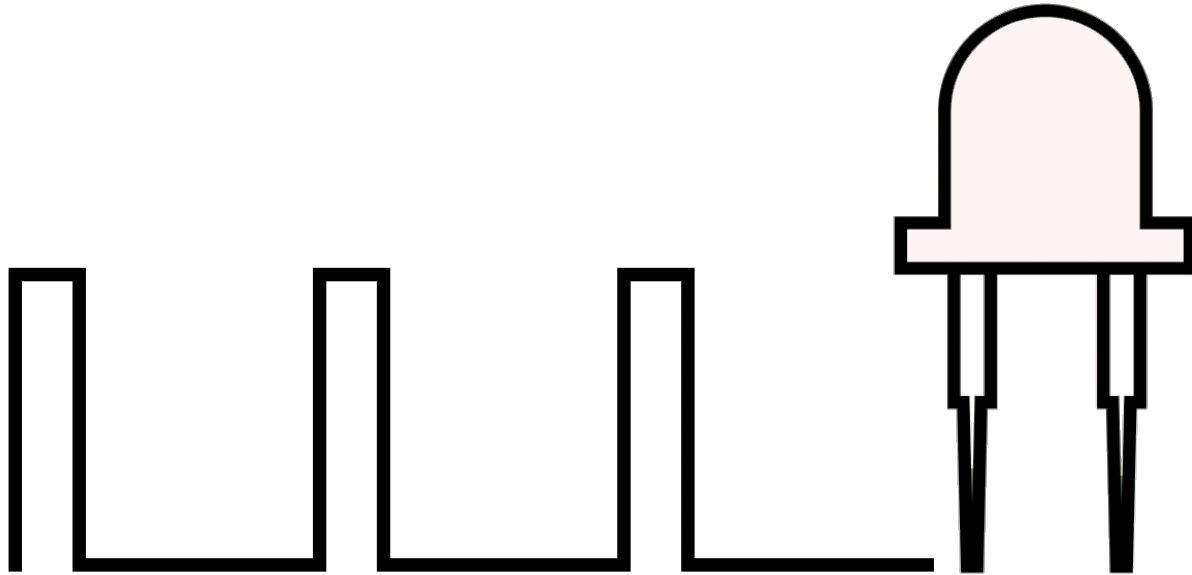
The Digital Pins marked with ~ can be used as "Analog Outputs", so-called PWM outputs

Control Brightness of a LED

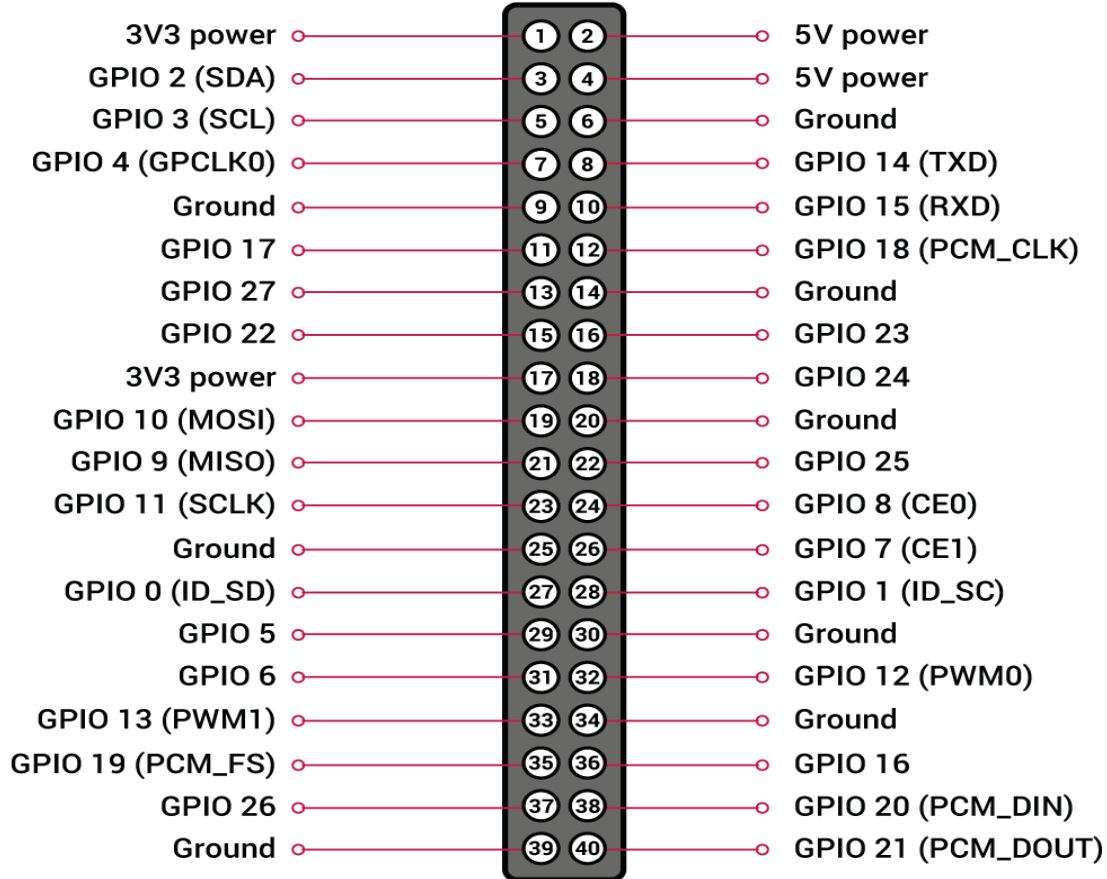
- We've seen how to turn an LED on and off, but how do we control its brightness levels?
- An LED's brightness is determined by controlling the amount of current flowing through it, but that requires a lot more hardware components.
- A simple trick we can do is to flash the LED faster than the eye can 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

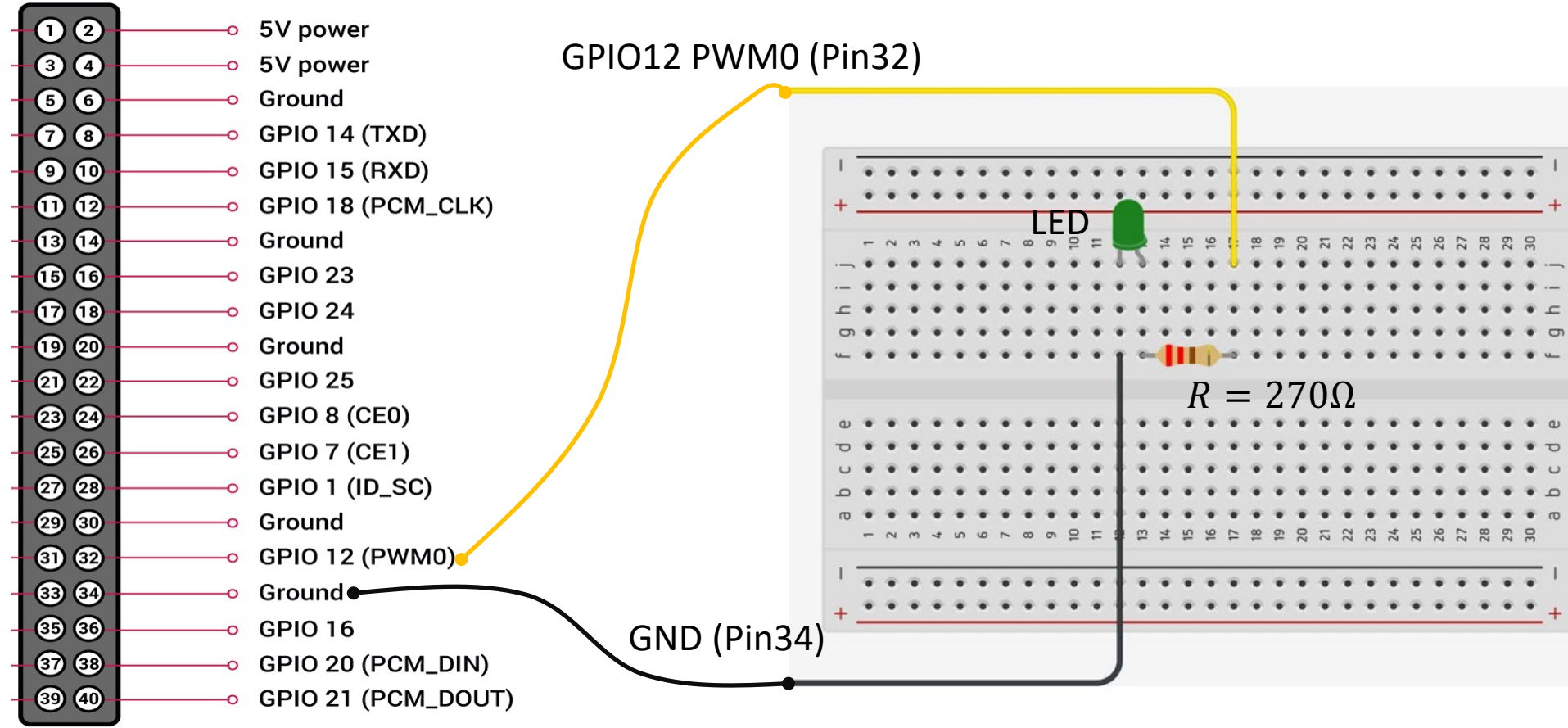
Below we see how we can use PWM to control the brightness of a LED



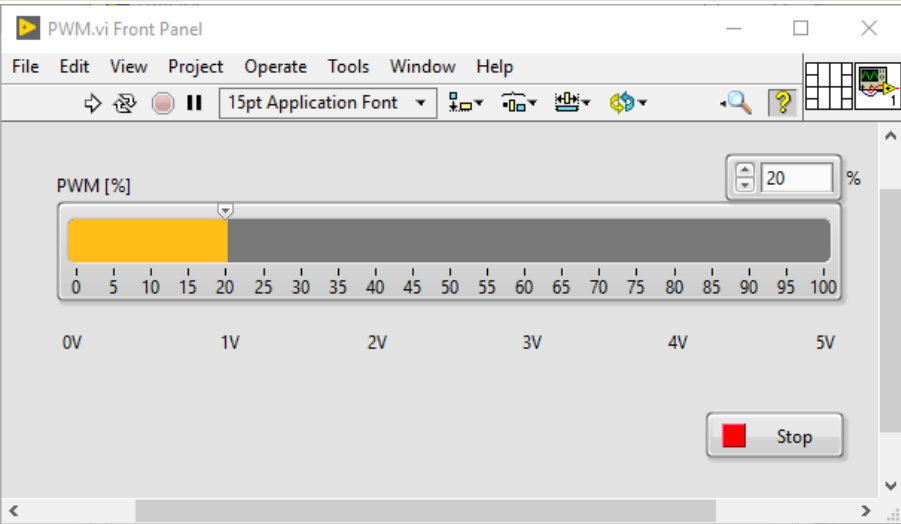
PWM pins on Raspberry Pi



PWM LED Wiring

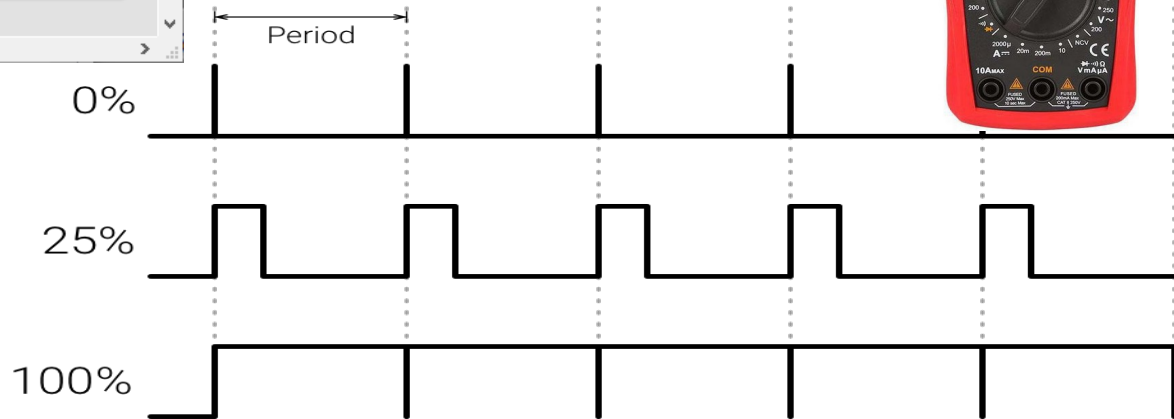
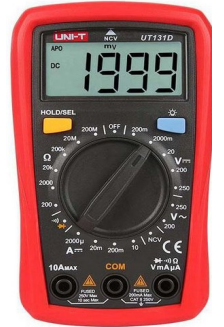


PWM Example

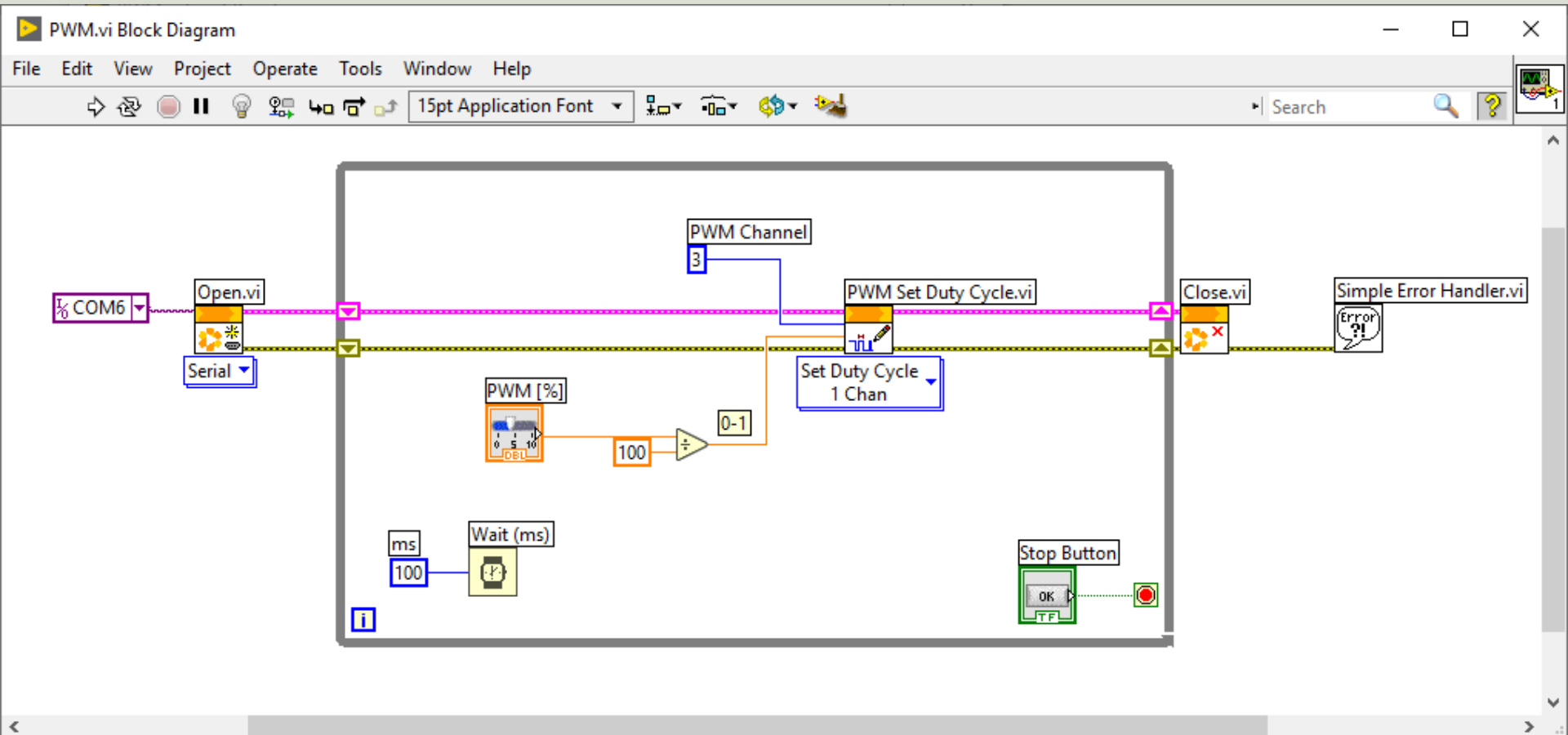


We will see the brightness of the LED will increase.

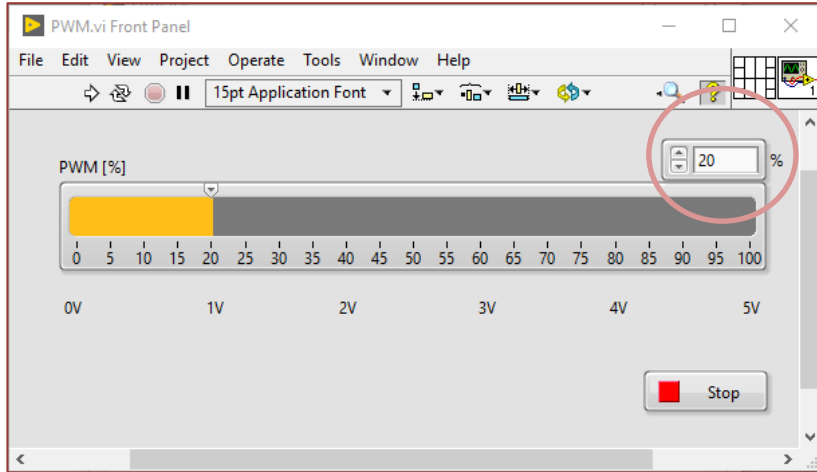
Or you can use a Multimeter and see the (average) voltage will increase



PWM Example



PWM Example



20% \rightarrow 1V



0-100% \rightarrow 0-5V





Build and Deploy Executable LabVIEW Application

Blinky Application

The image displays the LabVIEW development environment for a Blinky application. The main window is titled "Blinky.vi Block Diagram on LabVIEWPi.lvproj/raspberypi". The block diagram is enclosed in a "While Loop" and contains the following components:

- Open.vi**: A subVI icon for opening the device.
- Local I/O**: A dropdown menu for selecting the device.
- DO Channel**: A numeric control set to 16.
- Digital Write.vi**: A subVI icon for writing to the digital output.
- Digital Write 1 Chan**: A subVI icon for writing to a single channel.
- LED Status**: A subVI icon for reading the LED status.
- Wait (ms)**: A delay block set to 1000 milliseconds.
- Not**: A logical NOT operator.
- Stop Button**: A subVI icon for a stop button.
- Close.vi**: A subVI icon for closing the device.
- Simple Error Handler**: A subVI icon for handling errors.

The front panel, titled "Blinky.vi Front Panel on LabVIEWPi.lvproj/raspber...", features:

- LED Status**: A green circular indicator.
- Stop**: A red square button.

The LabVIEW interface includes a menu bar (File, Edit, View, Project, Operate, Tools, Window, Help) and a toolbar with various icons for navigation and execution. The status bar at the bottom shows "LabVIEWPi.lvproj/raspberypi".

Build Application

LabVIEWPi.lvproj - Project Explorer

File Edit View Project Operate Tools Window Help

Items Files

- Project: LabVIEWPi.lvproj
 - My Computer
 - Dependencies
 - Build Specifications
 - raspberrypi (192.168.137.250)
 - Analog Write.vi
 - Blinky.vi
 - Digital Read.vi
 - Digital Write - Read.vi
 - LED.vi
 - Push Button.vi
 - PWM.vi
 - PWM2.vi
 - Dependencies
 - Build Specifications

Context menu for Build Specifications:

- New
 - Real-Time Application
 - Packed Library
 - Source Distribution
 - Zip File
- Arrange By
- Help...

Blinky Properties

Category: Information

Source Files

Destinations

Source File Settings

Advanced

Additional Exclusions

Version Information

Web Services

Pre/Post Build Actions

Component Definition

Preview

Build specification name: Blinky

Target filename: startup.rtexe

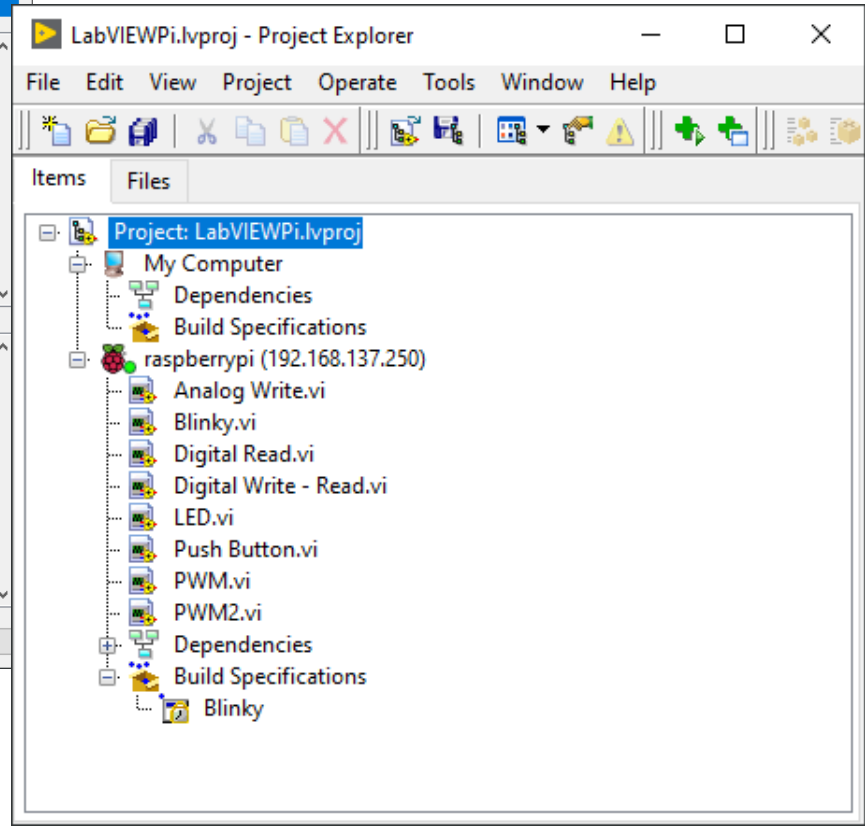
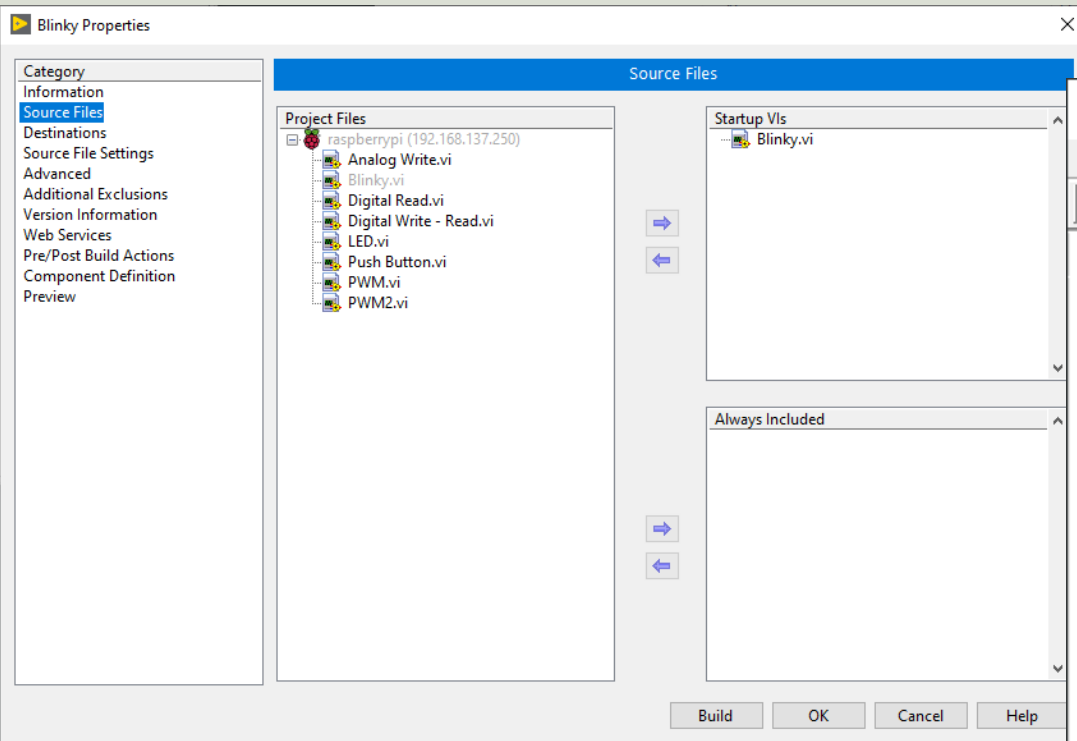
Local destination directory: C:\Users\hansha\OneDrive\Development\LabVIEW\LabVIEW LINUX\Raspberry Pi\builds\LabVIEWPi\raspberrypi\Blinky

Target destination directory: /home/lvuser/natinst/bin

Build specification description

Build OK Cancel Help

Build Application



Build Application

The image illustrates the steps to build and run a LabVIEW application on a Raspberry Pi. It consists of four sequential screenshots:

- Project Explorer:** The 'LabVIEWPi.lvproj - Project Explorer' window shows the project structure. The 'Blinky' file is selected, and a context menu is open with 'Build' highlighted.
- Build status dialog:** A dialog box titled 'Build status' displays the message: "The build is complete. You can locate the build at C:\Users\hansha\OneDrive\Development\LabVIEW builds\LabVIEWPi\raspberrypi\Blinky." It includes a progress bar and an 'Explore' button.
- Project Explorer (Run):** The 'LabVIEWPi.lvproj - Project Explorer' window shows the 'Blinky' file selected, and the context menu is open with 'Run as startup' highlighted.
- Reboot dialog:** A dialog box asks: "The target (raspberrypi) must be rebooted for the application to launch. Proceed with reboot?" with 'Yes' and 'No' buttons.

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

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

