INTERNET OF THINGS
DEPARTMENT OF COMPUTER ENGINEERING
SANTA CLARA UNIVERSITY, CALIFORNIA
BEHNAM DEZFOULI, PHD

Santa Clara University
Internet of Things
RESEARCH LAB
SIOTLAB

Internet of Things

Lab 2
GPIOs and Sensors
(Raspberry Pi)

Please Note!

- **Please do not copy and paste text from slides, this would result in non-standard characters that cannot be recognized by the command line interface (CLI), interpreters, and compilers**

- **Make sure you do not miss any single steps of the lab projects**

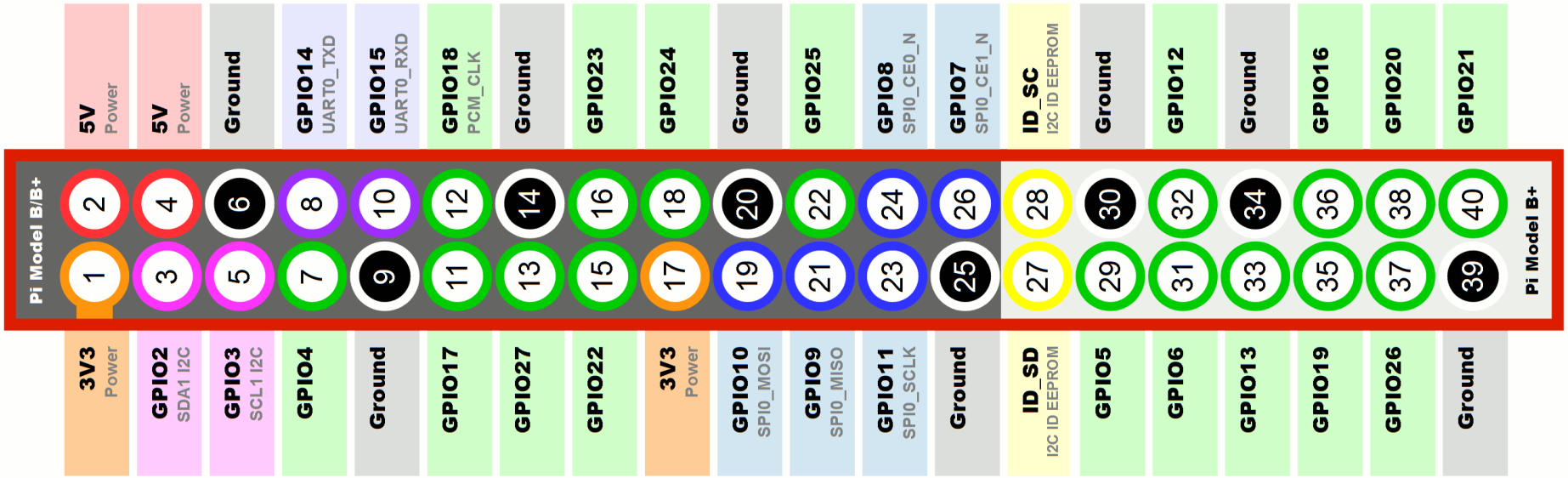
- **In most cases the errors and unwanted results are caused by typo errors and forgetting to complete some steps**

Introduction to GPIO

Introduction to GPIO

- GPIO stands for "**General Purpose Input/Output**"
- The RPi 3B has 26 pins marked as "GPIO" out of the 40 exposed header pins on the board
- A GPIO pin is a pin that has no specific usage defined by the processor, and therefore can be used by the programmer for any purpose
- GPIO pins can be used to power external devices, communicate data with sensors, or both, all by setting voltage outputs and measuring voltage inputs

Raspberry Pi Pinout

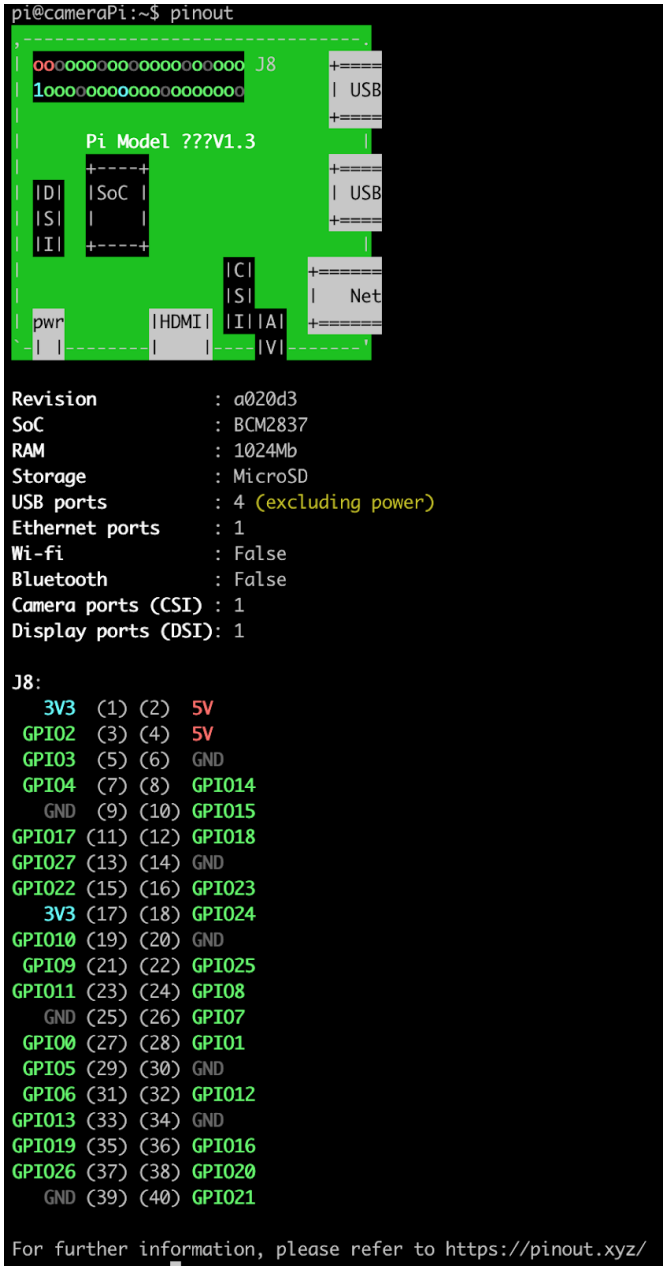


Please see: `https://pinout.xyz/pinout/pin28_gpio1`

Introduction to GPIO

- You can also see a quick reference pinout at any time on the RPi by running the **pinout** command:

➢ `pinout`



In this lab:

- **We will be attaching an ultrasonic sensor to the GPIO pins on the RPi**
- **We use a Python program to collect the readings from the sensor**

Introduction to Vim

Writing some Code

- The next step is to open and edit a file on the RPi
- However, you only have a terminal, no desktop environment, so **you'll need to learn how to use a terminal-based text editor**

- There are many terminal-based text editors available to you, such as:
  - **nano** (which you may have used before)
  - **vim**
  - **emacs**

- Since a **lightweight version of vim** (**vi**) is installed on every Linux system by default, **it is probably the most useful text editor to learn how to use**

Vim basics

- First, **connect the USB to Ethernet adapter** to a USB port on your computer and the Ethernet port of the Raspberry Pi

- **Install the full version of vim** (enter 'y' when prompted):
  `sudo apt-get install vim`

- Vim is different from *gedit* or notepad because it has several modes, such as
  - **insert mode**
  - **replace mode**
  - **visual mode**
  - **command mode**

- In a folder that you will remember, open a test file, call it "sensor.py":

➢ **`vim sample.py`**

Insert Mode

- You're now editing a blank file in command mode
- To change to **insert mode** and add text, **press "i" (for insert)**
- Now you should see a blinking cursor at the top, and the bar at the bottom (the status bar) should say `-- INSERT --`

- Let's insert some text:

```
import RPi.GPIO as GPIO
import time
```

- Later, you'll learn what these lines do, but for now let's save and close the file

Beginning Commands

- Since you're still in insert mode, you'll need to hit **`ESC`** in order to return to command mode
  - The status bar should change back to what it was before
- Now type "`:write`" including the colon to save your file
  - Press enter to write the file to disk (save it)
- Now type "`:quit`" to leave vim

- Use the `cat` utility to see the content of your file:
  - `cat sample.py`

- If you did everything correctly, you should see the content of your file on the screen

Shortcuts

- Open the file again (`vim sample.py`) and add the following line to the bottom of the file (don't forget you need to enter insert mode before typing):

➢ `GPIO.setmode(GPIO.BCM)`

- As a faster way to save the file and exit, return to command mode with ESC, and
  - "`:w`" to write the file.
  - "`:q`" is the shortcut to quit vim
- However, since saving and quitting together is a common use case, **you can also type ":wq" to both write and quit**

Taking vim Further

- If you make a big mistake in your file, you can close it without saving:

➢ `:q!`

- If you only want to undo the last few edits, you can push "`u`" **in command mode** to *undo*
- Likewise, `CTRL+R` in command mode stands for *redo*

- Vim is a very powerful tool and can dramatically improve your typing efficiency
- If you would like to learn more vim shortcuts and tricks, you can run the `vimtutor` command on the cli

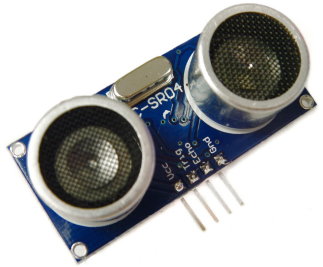
➢ `vimtutor`

Wiring the Ultrasonic Sensor

Wiring the Ultrasonic Sensor

Ultrasonic Basics

- The ultrasonic sensor in our lab has four pins, labeled **Vcc**, **Trig**, **Echo**, and **Gnd**
- The Vcc and Gnd pins are connected to the 5V Vcc and Gnd pins on the Rpi, respectively
- The Trig and Echo pins are connected to any available GPIO pins, with the Trig pin configured for output, and the Echo pin configured for input

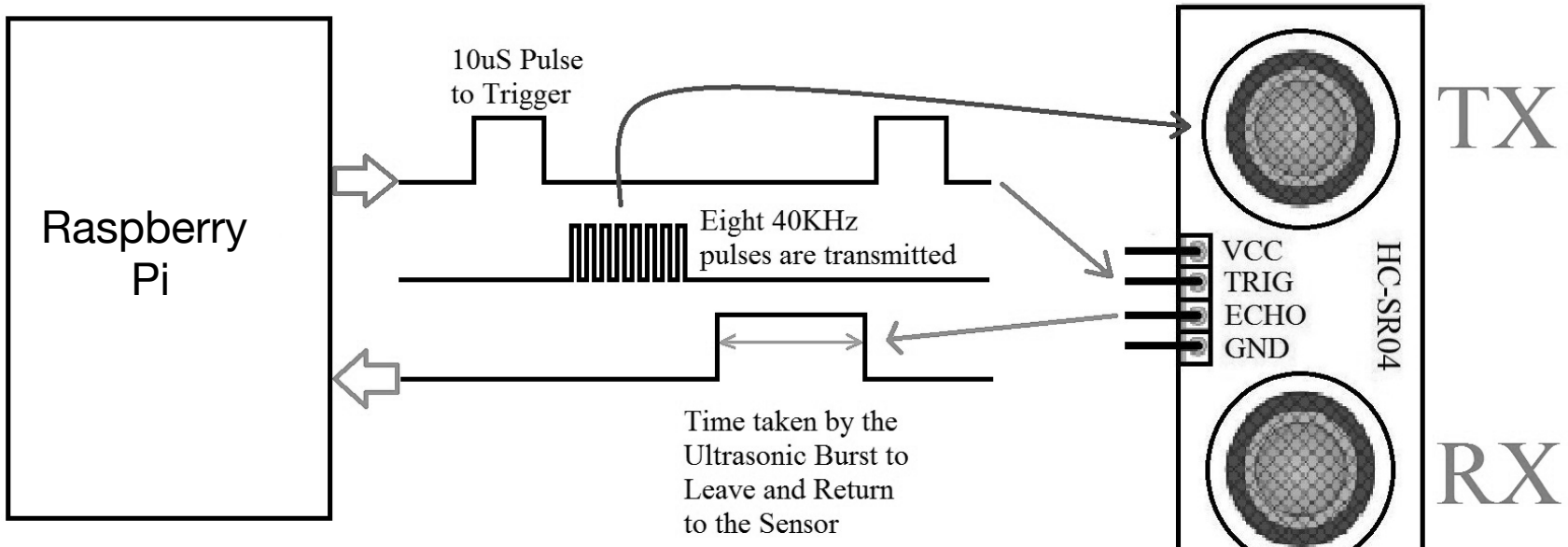


- **How it Works?**
  - **It transmits an ultrasonic pulse when the Trigger pin has a high voltage applied**
  - **When it picks up the ultrasound echo, it sends back a 5V pulse on the Echo pin**
  - **The duration of the ECHO pulse is proportional to the distance from the object**

Wiring the Ultrasonic Sensor

Ultrasonic Basics

- We need to **send a pulse to the TRIG pin for 10 microseconds** to take a reading
- After that, we need to listen to the input of the ECHO pin
- **When it changes from low voltage to high voltage, that signifies the reading is complete**

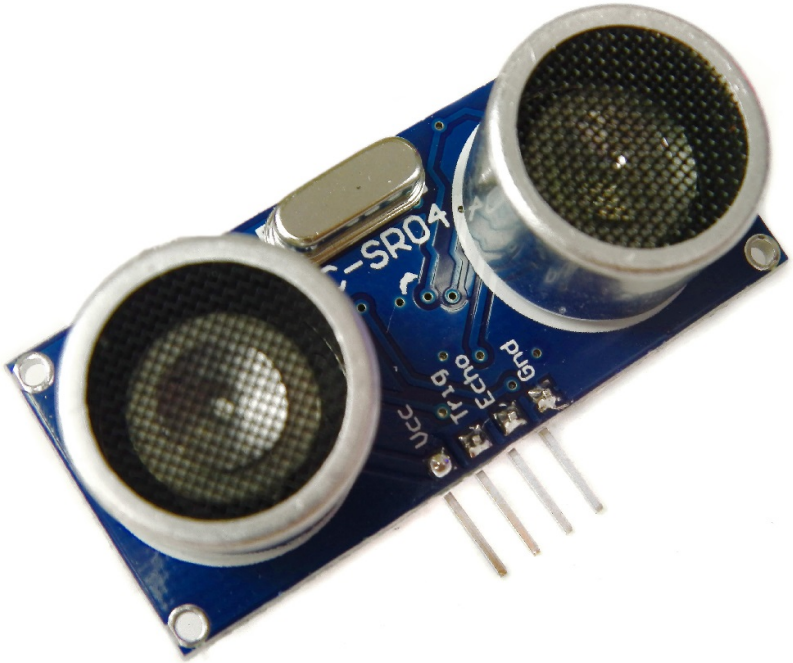


Length of the ECHO signal reflects the distance between the senor and object

Wiring the Ultrasonic Sensor

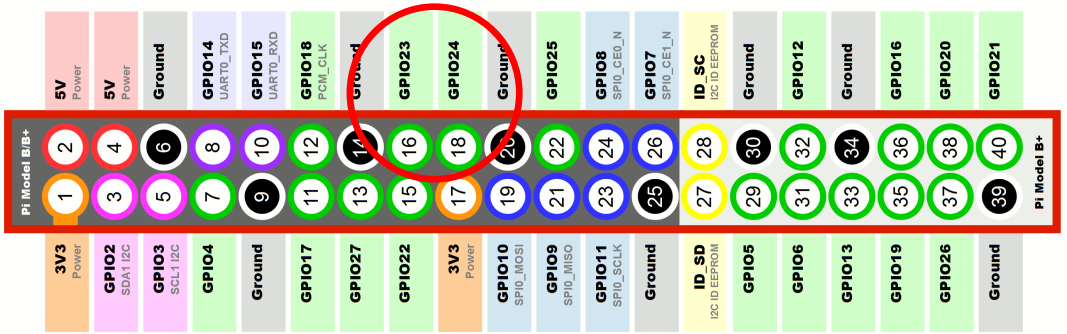
The Voltage Issue

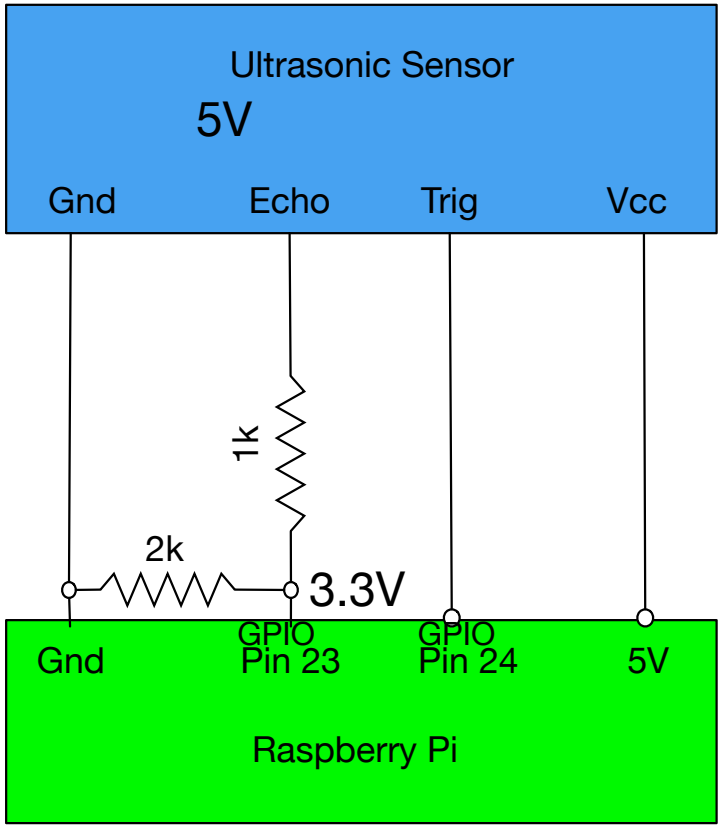
- We can't directly connect the Echo pin to the RPi as there's a small problem
- **The GPIO pins can only accept 3.3V of input power, and the sensor outputs 5V, which may damage the RPi**
- Therefore, we must use a **voltage divider**
- Wire the RPi and the ultrasonic sensor as shown in the diagram on the next slide



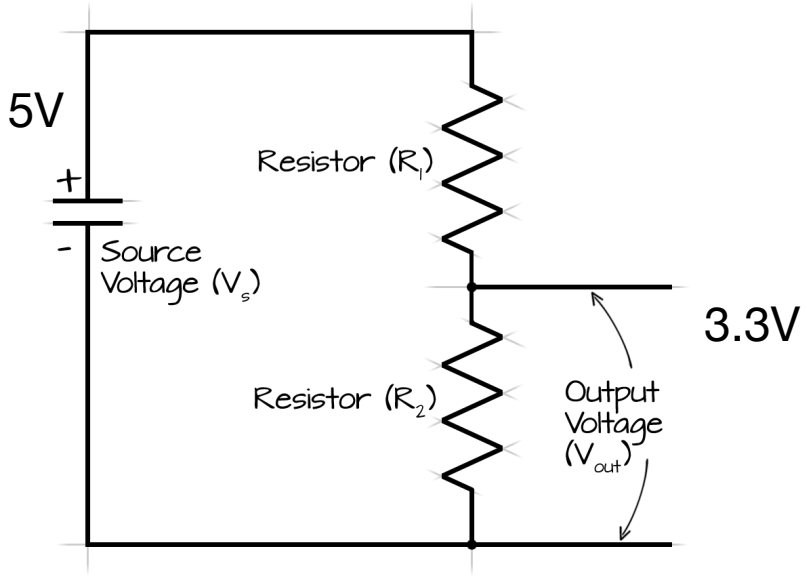
Wiring the Ultrasonic Sensor

Voltage Divider Circuit





$$V_{out} = \frac{V_s \times R_2}{(R_1 + R_2)}$$



Communicating with the Sensor using Python

Communicating with the Sensor using Python

GPIO Libraries

- There are five major Python modules commonly used to control the GPIO pins on the RPi, making Python an important player in GPIO
- The main packages are:
  - **`RPi.GPIO`**
  - `RPIO.GPIO`
  - `wiringPi`
  - `pigpio`
  - `gpiozero`

- The most popular package at time of writing is **RPi.GPIO**

import statements

- First, let's revisit the code you added to *sample.py* in the previous section:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
```

More on import Statements

How it works?

- This code always goes at the top of a Python script

- **`import`** statement tells the Python interpreter to find and load a package
  - The first line tells Python to import RPi.GPIO
  - Next, the **`as`** clause creates an alias while importing a module

- Next, Python will load the **`time`** package

- Finally, the third line configures the GPIO package to **use the Broadcom pin numbering scheme**
  - **This allows you to refer to GPIO pins in your code by the numbers on the pinout diagram shown earlier**

Quick vim Tip

- Since we're using vim, we should enable the **syntax highlighting features** to make the coding easier
- **In vim's command mode, type "`:syntax on`"**
- You should see some of the terms in your code are now **bold** or **underlined**
  - All the code is in black and white because you are connected over a serial connection
  - Transferring color-coded text is slower than transferring plain text, as there is more information to communicate

Enabling Color Support

- In order to make sure the editor is fast and easy to use, vim will default to black and white for slow connections (such as serial)
- Sometimes the loss in speed may be worth the **full-color syntax highlighting**
- In order to try it out, we first need to leave vim and change a shell environment variable:
  - Close vim (`:wq`) and run the following in bash:
  - `export TERM=xterm-256color`
  - Now, reopen vim and run the syntax command again
  - Notice everything is in full color, but very unresponsive
- If editing in full color is too slow for you, you can disable it at any time:
  - Close vim, and run:
  - `export TERM=vt102`
  - Reopen vim and the colors are gone

GPIO Initialization

- Continuing with the Python file, the next step is to decide which pins to use for our ultrasonic sensor
- Below, two global variables have been defined, TRIG and ECHO, and set to the pin numbers (more on this later)

```
TRIG = 24
ECHO = 23
```

- Next, the RPi needs to be told **how to use the pins**, i.e., if they are to be used for input or output

```
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)
```

Sensor Initialization

- Now, we need to initialize the sensor by sending a low voltage (0 value) to the TRIG pin and waiting 2 seconds for it to stabilize
  - The print line below is Python's version of "cout”

```
GPIO.output(TRIG, 0)
print("Waiting For Sensor To Settle")
time.sleep(2)
```

The measure() Fuction

- Next, we need to define our `measure()` function:

```
def measure():
    pulse_end = 0
    pulse_start = 0
```

- The above code defined a function called `measure()` which
  - has no arguments
  - has two local variables set to zero

- Note that there are no braces in the code above, as **Python uses indentation instead of braces to define scope**
  - **Indentation is very important in Python, so pay attention to your spacing!**

Reading Pulses

- We **send a pulse to the TRIG pin for 10** microseconds to take a reading
  - **Make sure the indentation on these lines of code matches the indentation of the variables**

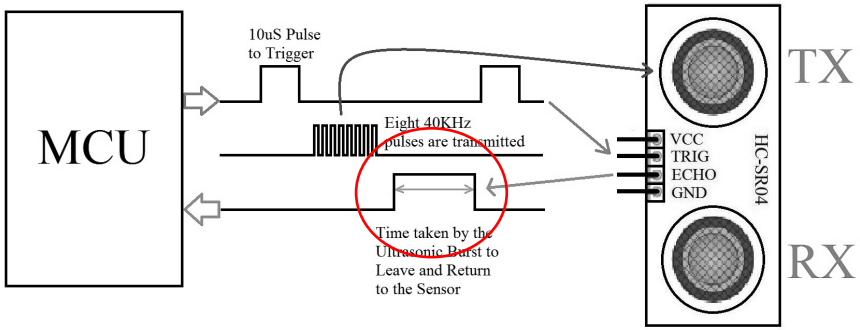
```
GPIO.output(TRIG, 1)
time.sleep(0.00001)
GPIO.output(TRIG, 0)
```

- Then, we listen to the input of the ECHO pin
- When it changes from low voltage to high voltage, that signifies the reading is complete and we need to store another timestamp

```
while GPIO.input(ECHO)==0:
   pulse_start = time.time()
while GPIO.input(ECHO)==1:
      pulse_end = time.time()
```

Code Explanation

- **How it works?**
  - The code in the previous slide is an example of two while loops in Python
    - While the ECHO pin's input is a low voltage (represented by 0) we set `pulse_start` to the current time
    - Then, until the ECHO pin's value changes from 1 back to 0, we update the `pulse_end` variable to the current time



- Using these two timestamps, we can take the difference to see how long the change in voltage took:

➢ `pulse_duration = pulse_end - pulse_start`

Returning the Distance

- With the time it takes for the signal to travel to an object and back again, we can calculate the distance using the following formula

$$Speed = \frac{Distance}{Time}$$

- The speed of sound is variable, depending on what medium it's travelling through, in addition to the temperature of that medium
- We will take our baseline as the **343m/s = 34300cm/s**

- We also need to **divide time by two** because what we've calculated above is actually the time it takes for the ultrasonic pulse to travel the distance to the object and back again
- We simply **want the distance to the object**

$$34300 = \frac{Distance}{Time/2}$$

$$17150 = \frac{Distance}{Time}$$

$$17150 \times Time = Distance$$

Returning the Distance

- So we must multiply the time duration by a constant to get our distance measurement, then round it to the hundredths' place:

```
distance = pulse_duration * 17150
distance = round(distance, 2)
return distance
```

`round(number[, ndigits])`

- The `round()` method returns the floating point number rounded off to the given ndigits digits after the decimal point
- If no ndigits is provided, it rounds off the number to the nearest integer

Final Results

- Now our function is defined, but we haven't called it
- On the next line and unindented, add the following to call the `measure()` function in a loop forever:

```
while True:
    reading = measure()
    print("Distance:",reading,"cm")
```

- Save and quit vim, as it's time to test our wiring and code:

➢ `sudo python3 sample.py`

- If you did everything correctly, **you should see a stream of measurements on the screen**
- If you're new to Python and having issues with your code, check your indentation, as Python beginners often have trouble with indentation rules

- Task 2-1.
  - Demo the project to the TA