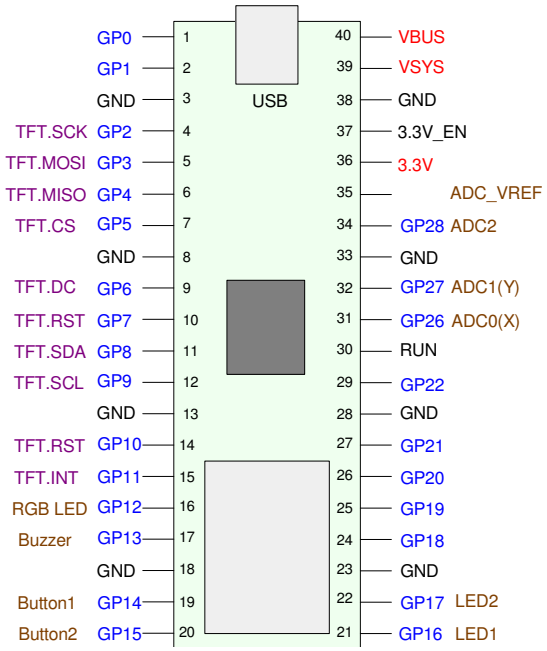
6. Binary Inputs

Parallel Inputs - Voting Machine - Hungry-Hungry Hippo



Introduction:

Each of the General Purpose pins on the Pi-Pico can be used for

- Binary Outputs (last lecture), or
- Binary Inputs (this lecture)

as well as other functions (coming later). Similar to our last lecture

- 0V is read as logic 0
- 3.3V is read as logic 1

When using an I/O pin as a binary input, you do need to be careful:

Do not apply 5V to the general-purpose pins on a Pi-Pico.
Doing so will desroy the Pico board.

This lecture looks at

- Converting push buttons to binary (0V & 3.3V) logic levels
- Converting voltages, resistance's, and temperatures to 0V / 3.3V logic levels,
- Building a random number generator using a push-button
- Counting edges & building a voting machine
- Counting multiple edges and writing a Hungry-Hungry Hippo games.

Reading Push Buttons:

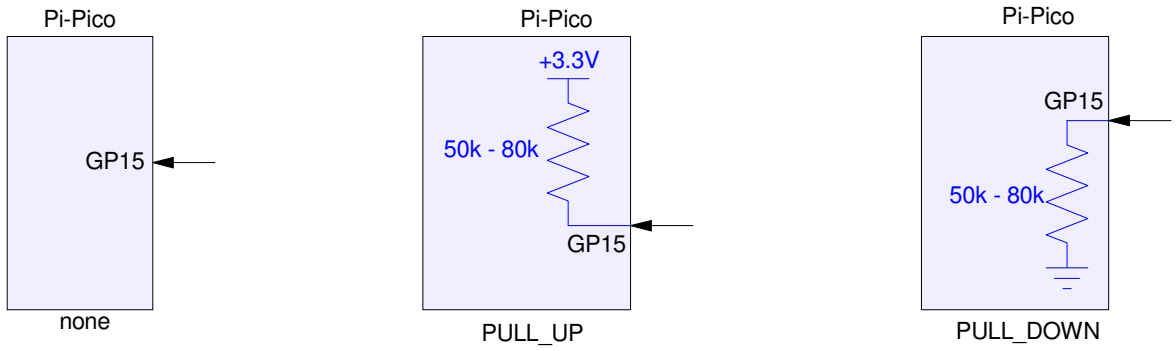
The way you set a pin, such as pin 15, to input is to use the Pin command from *machine:*

```
from machine import Pin

Button = Pin(15, Pin.IN)
Button = Pin(15, Pin.IN, Pin.PULL_UP)
Button = Pin(15, Pin.IN, Pin.PULL_DOWN)
```

The first option simply sets pin #15 to input and leaves it floating: it's up to the hardware to drive the pin high (3.3V) or low (0V). The problem with this approach is if a pin isn't tied high or low (termed floating), its voltage can be unpredictable

The second option connects a 50k-80k resistor from pin 15 to +3.3V internally to the Pi-Pico chip (termed *pull-up*). If a pin is left floating in this case, it is read as logic 1.

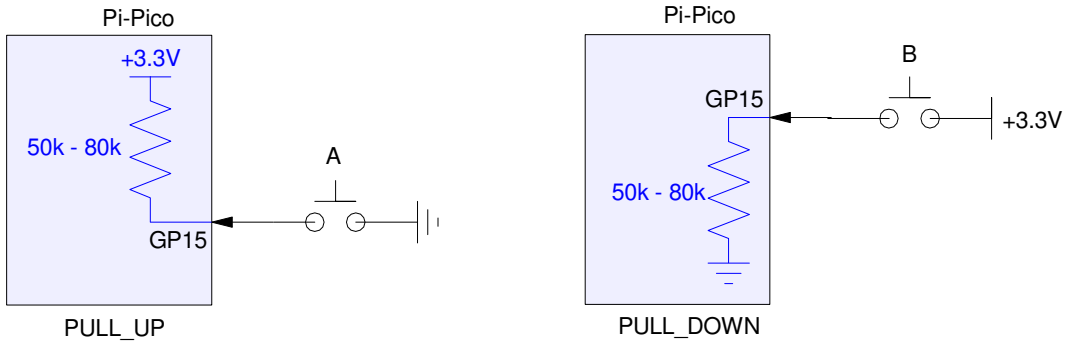
The third option connects a 50k-80k resistor from pin 15 to ground internally to the Pi-Pico chip (termed *pull-down*). If a pin is left floating in this case, it is read as logic 1.



Each input pin can be floating, pulled up to +3.3V, or pulled-down to ground

Both pull-up and pull-down can be used along with a momentary switch to read if the switch is pressed or not:

- Pull-Up: GP15 is logic 1 if A is not pressed and 0 if A is pressed
- Pull-Down: GP15 is logic 1 if A is pressed and 0 if A is not pressed



Push-button connections with a Pull-Up and Pull-Down setting

In general, the pull-up setting is safest

- Pushing the button will not damage the Pico chip - you're just connecting it to ground

The pull-down setting can damage your Pico board:

- If you accidentally use +5V rather than +3.3V, pressing the button will fry your Pico board

Stick with the pull-up option with the switch tied to ground.

Sample Code: The following program displays

- 1 when button 15 is not pressed
- 0 when button 15 is pressed

```
from machine import Pin
from time import sleep_ms

Button = Pin(15, Pin.IN, Pin.PULL_UP)

while(1):
    X = Button.value()
    print(X)
    sleep_ms(100)
```

Sidelight: Boolean Logic with Momentary Switches

One kind of neat feature of using switches and pull-up resistors is you can implement different logic functions by combining NO/NC switches in series and parallel. To see this, let's start with normally open switches.

If using a single momentary switch called A, the logic implements is

$$\overline{Y} = A$$

(If switch A is pressed (on), the output is 0V).

If you place two switches in series, you get an AND function:

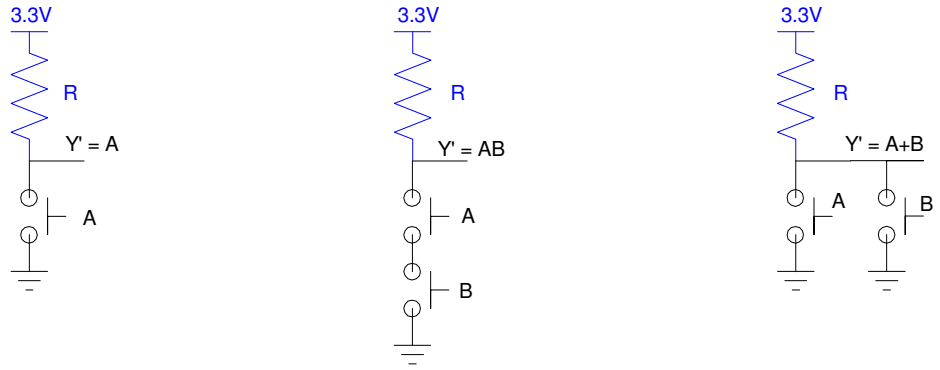
$$\overline{Y} = AB$$

(The output is grounded only if both A and B are pressed).

If you place two switches in parallel, you get an OR function

$$\overline{Y} = A + B$$

A NOT function can be implemented by using a normally-closed switch (NC) instead of normally open (NO).



Logic AND, OR, and NOT can be implemented by placing switches in series, parallel, or using NC switches

Recalling from Digital Systems, if you have AND, OR, and NOT functions, you can implement anything.

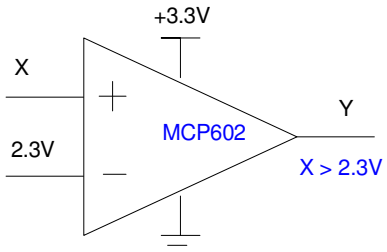
Reading Voltage, Resistance's, Temperature, & Light

If you want to read something else, such as a voltage, resistance, temperature, or light level, the trick is to convert these signals to 0V / 3.3V Boolean values.

Reading Voltage: To read a voltage, use an MCP602 op-amp as a comparitor. For example, if you want to generate the signal

$$Y > 2.3V$$

the circuit would be:



Circuit for the function (X > 2.3V)

Note that you need to use an op-amp similar to the MCP602

- The op-amp needs to be able to operate with 0V and 3.3V power supplies
- The op-amp needs to be rail-to-rail, meaning the output covers the entire range of 0..3.3V

The MCP602 does this - other op-amps such as the LM741 or LM833 do not.

Reading Resistance: If you want to read a resistance, such as

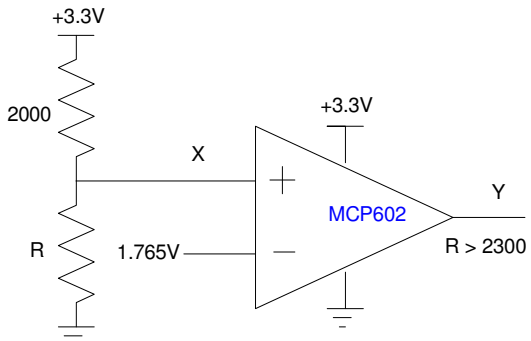
Y = R > 2000 Ohms

the trick is to

- Convert resistance to a voltage using a voltage divide, then
- Use the previous circuit

For example, if you use a 2k resistor in the voltage divider, you want to switch at

- R = 2300 Ohms
- $X = \left(\frac{R}{R+2000}\right) 3.3V = 1.765V$



Circuit to output the logic function R > 2300 Ohms

Reading Temperature: If you want to read a temperature, such as

Y = T > 15C

the trick is to

- Convert temperature to resistance using a thermistor (or similar device)
- Find the resistance where you want to switch, then
- Use the previous circuit

For example, come up with a circuit which outputs 3.3V (logic 1) when T > 15C.

Solution: Pick a thermistor, such as

$$R = 1000 \cdot \exp\left(\frac{3905}{T+273} - \frac{3905}{298}\right)\Omega$$

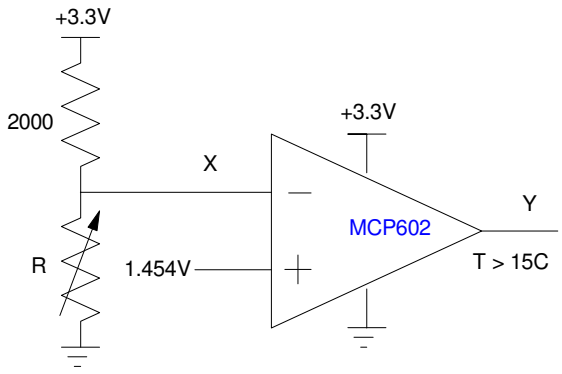
At 15C, R = 1576 Ohms

Assuming a 2k resistor for the voltage divider, at 1576 Ohms, X = 1.454V

If you connect to the plus input, you get 0V when it's really hot (R goes to zero as T goes to infinity). This implements the function

T < 15C

To negate this, flip the inputs to the plus and minus pins on the op-amp



Circuit to output the Boolean function T > 15C

Level-Sensitive Programs: Debate Moderator

Once you get the input in the form of a 0V / 3.3V Boolean signal, you can now use these inputs to control a program. Typically, the program will be

- Level-sensitive: do one thing while the input is high, another thing while low, or
- Edge-sensitive: only do something on the rising or falling edges.

An example of a level-sensitive program would be one which

- Has two timers, each initialized to 10.0 seconds.
- Every 100ms, it checks the buttons (15 and 14)
- If either button is pressed (person A or B is speaking), their time is decremented
- When a debater's time reaches zero, it stops at zero (and presumably something happens)

```
# Debate Moderator
from machine import Pin
from time import sleep_ms

ButtonA = Pin(15, Pin.IN, Pin.PULL_UP)
ButtonB = Pin(14, Pin.IN, Pin.PULL_UP)

ATime = 10.0
BTime = 10.0

while(1):
    if(ButtonA.value() == 0):
        if(ATime > 0):
            ATime -= 0.1
    if(ButtonB.value() == 0):
        if(BTime > 0):
            BTime -= 0.1
    print(ATime, BTime)
    sleep_ms(100)
```

As A or B holds down their button (indicating they are speaking), their time decrements down to zero and stops at zero.

Edge Sensitive Program: Voting Machine

A second type of program counts edges

- Action only takes place during the rising edge and/or falling edge of a signal

If you only have a single signal, you can count edges by

- Waiting while the signal is logic 1 (wait for it to go low), then
- Wait while the signal is logic 0 (wait for it to go high)

At that point, you just detected a rising edge.

For example, count and display the number of rising edges on button 15

- *Count the number of times you release button 15*

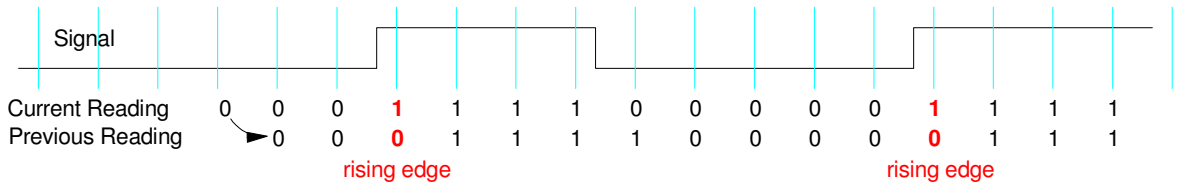
```
# Count rising edges


from machine import Pin
from time import sleep_ms

Button = Pin(15, Pin.IN, Pin.PULL_UP)

Count = 0
print('Press and release button to count')
while(1):
    while(Button.value() == 1):
        pass
    while(Button.value() == 0):
        pass
    Count += 1
    print(Count)
```

If you have multiple inputs, you need to use a different technique.

One technique that works i to look for a 0/1 transition:

- If the current signal is a 1, and
- The previous reading was a 0

you just detected a rising edge.



A rising edge is detected when the current signal is 1 and its previous value was 0

A voting machine with two inputs illustrates this

- As the start, each candidate has zero votes (PlayerA and B)
- When button 15 is pressed and released, Player A's score goes up by one (Na += 1)
  - A rising edge is when A = 1 and A's previous value (zA) is 0
- When button 14 is pressed and released, Player B's score goes up by one (Na += 1)
  - A rising edge is when B = 1 and B's previous value (zB) is 0
- The buttons are polled every 100ms, looking for a rising edge

```
# Voting Machine
# input 14 and 15

from machine import Pin
from time import sleep_ms

PlayerA = Pin(15, Pin.IN, Pin.PULL_UP)
PlayerB = Pin(14, Pin.IN, Pin.PULL_UP)

A = 1
B = 1
Na = 0
Nb = 0
time = 0
while(1):
    zA = A
    A = PlayerA.value()
    zB = B
    B = PlayerB.value()
    if( (A==1) & (zA==0) ):
        Na += 1
    if( (B==1) & (zB==0) ):
        Nb += 1
    print('Votes for A ',Na, '   Votes for B ',Nb)
    sleep_ms(100)
```

Hungry-Hungry Hippo

Finally, let's use the push buttons to play a game of *Hungry-Hungry Hippo*

- At the start of the game, a 10.0 second timer is set.
- Each player presses their button as fast as they can, with each button press (rising edge) tallied
- Once 10 seconds is over, the game is over.

This is similar to a voting machine, except

- The time is limited to 10 seconds. Once time is over, stop counting.
- The buttons are sampled every 10ms rather than 100ms, allowing faster button presses to be recorded

Also, rather than display the score after every 10ms, display the score after each button press.

To do the latter, a thing called a *flag* is used. Flags tell a program when something has happened (such as a button press). That flag then controls when the score is displayed (flag = 1) or skipped (flag = 0).

```
# Hungry Hungry Hippo
# input 14 and 15

from machine import Pin
from time import sleep_ms

PlayerA = Pin(15, Pin.IN, Pin.PULL_UP)
PlayerB = Pin(14, Pin.IN, Pin.PULL_UP)

A = 1
B = 1
Na = 0
Nb = 0
time = 0
flag = 0
print('Press and release buttons to count')
while(time < 10):
    zA = A
    A = PlayerA.value()
    zB = B
    B = PlayerB.value()
    if( (A==1) & (zA==0) ):
        Na += 1
        flag = 1
    if( (B==1) & (zB==0) ):
        Nb += 1
        flag = 1
    if(flag == 1):
        print(Na, Nb)
        flag = 0
    sleep_ms(10)
    time += 0.01
print('Game Over')
if(Na > Nb):
    print('Player A Wins')
elif(Nb > Na):
    print('Player B Wins')
else:
    print('Tie')
```

Summary

Each I/O pin can be set up as a binary input or binary output. For binary inputs

- 0V is read as logic 0,
- 3.3V is read as logic 1, and
- 5V destroys your Pico board (don't do it)

These inputs can control a program's flow

- Using the level of the signal (logic 1 or 0), or
- Using the edges of the signal (rising or falling)

References

Pi-Pico and MicroPython

- https://github.com/geeekpi/pico_breakboard_kit
- https://micropython.org/download/RPI_PICO/
- https://learn.pimoroni.com/article/getting-started-with-pico
- https://www.w3schools.com/python/default.asp
- https://docs.micropython.org/en/latest/pyboard/tutorial/index.html
- https://docs.micropython.org/en/latest/library/index.html
- https://www.fredscave.com/02-about.html

Pi-Pico Breadboard Kit

- https://wiki.52pi.com/index.php?title=EP-0172

Other

- https://docs.sunfounder.com/projects/sensorkit-v2-pi/en/latest/
- https://electrocredible.com/raspberry-pi-pico-external-interrupts-button-micropython/
- https://peppe8o.com/adding-external-modules-to-micropython-with-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-esp32-esp8266-micropython/