



# Serdar Arıtan

[serdar.aritan@hacettepe.edu.tr](mailto:serdar.aritan@hacettepe.edu.tr)

## Hacettepe Üniversitesi

[www.hacettepe.edu.tr](http://www.hacettepe.edu.tr)

## Biyomekanik Araştırma Grubu

[www.biomech.hacettepe.edu.tr](http://www.biomech.hacettepe.edu.tr)

# #9



# Basic GPIO Functions

We have already met the basic methods of the Pin object:

- `init(mode)` – Set mode to input or output.
- `value(x)` – the Sets the line to x, usually 0 or 1, but x can be anything that evaluates to true or false.
- `on()` – the set the line to high.
- `off()` – the set the line to low.
- `high()` – the set the line to high.
- `low()` – the set the line to low.
- `toggle()` – if the line is high set it low and vice versa.



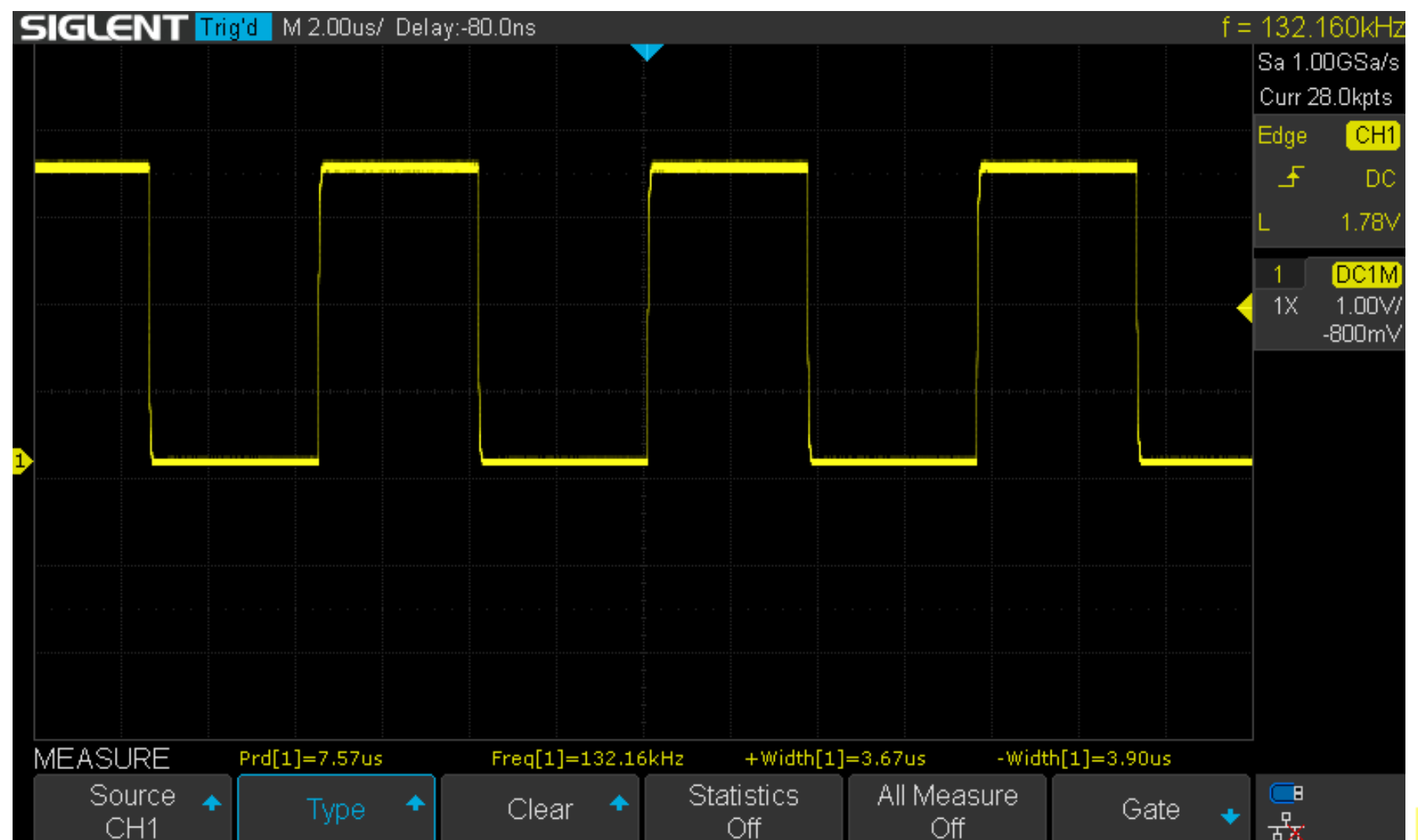
# How Fast

A fundamental question that you have to answer for any processor intended for use in embedded or IoT projects is, how fast can the GPIO lines work?

```
from machine import Pin
```

```
pin = Pin(22, Pin.OUT)
```

```
while True:  
    pin.value(1)  
    pin.value(0)
```





# How Fast

A fundamental question that you have to answer for any processor intended for use in embedded or IoT projects is, how fast can the GPIO lines work?

```
from machine import Pin
```

```
@micropython.native
```

```
def squareWave():
```

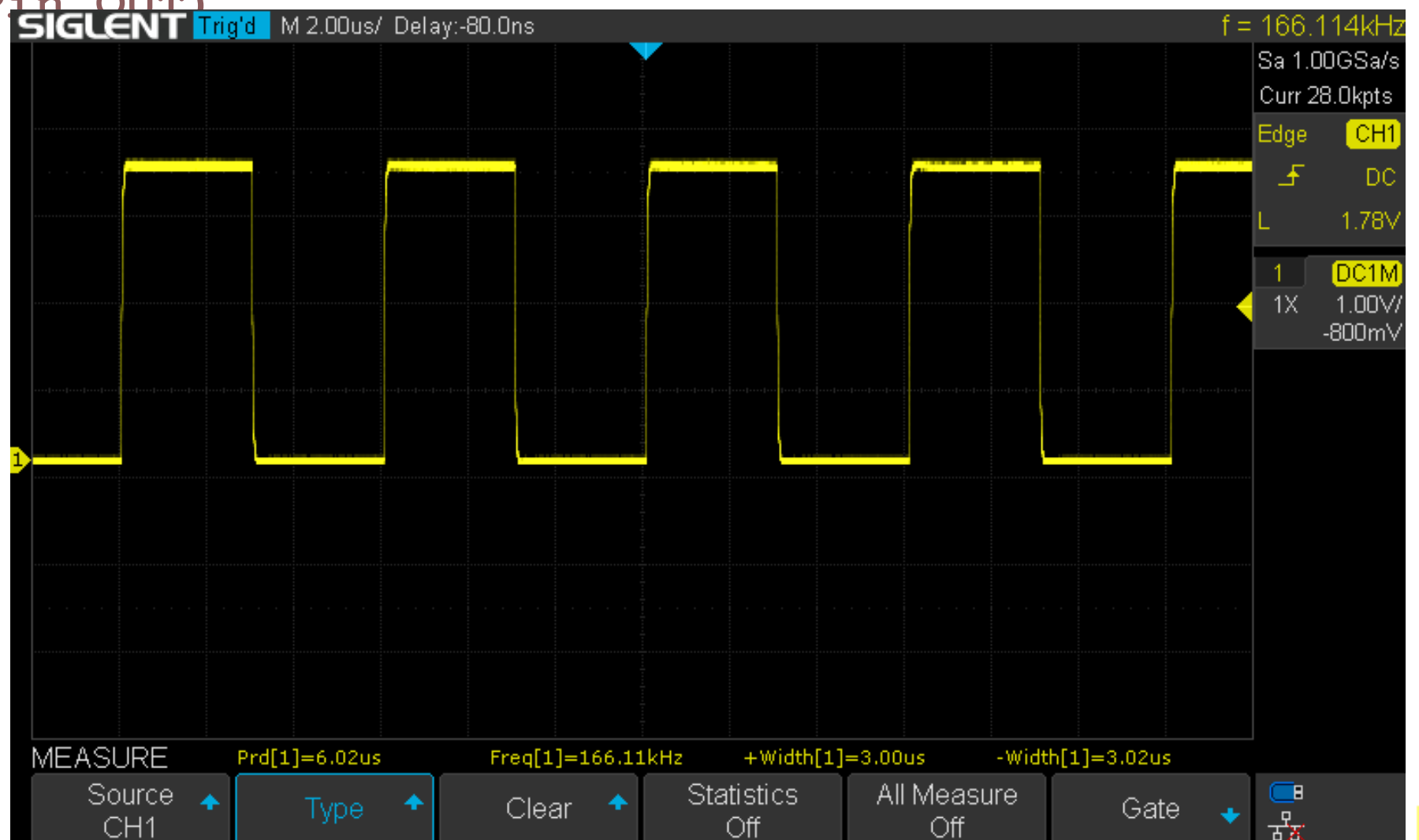
```
    pin = Pin(22, Pin.OUT)
```

```
    while True:
```

```
        pin.value(1)
```

```
        pin.value(0)
```

```
squareWave()
```



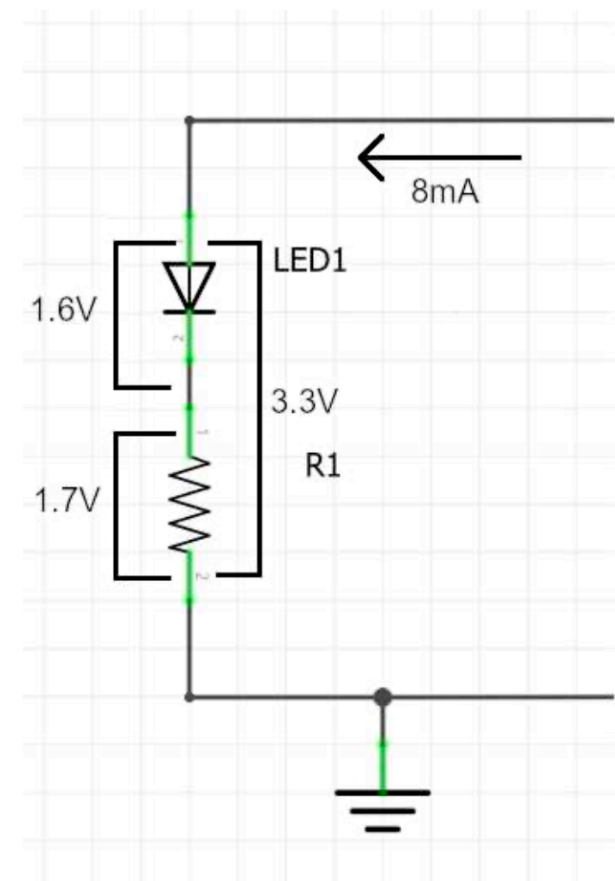


# Driving An LED

One of the first things we need to know how to do is compute the value of a current-limiting resistor. An LED is a non-linear electronic component – the voltage across it stays more or less the same irrespective of the current passing through the device. When you use an LED you need to look up its forward voltage drop, about 1.7V to 2V for a red LED and about 3V for a blue LED, and the maximum current, usually 20mA for small LEDs.

A GPIO line supplies 3.3V and if you assume 1.6V as the forward voltage, across the LED, that leaves 1.7V across the current-limiting resistor since voltage distributes itself across components connected in series. If we restrict the current to 8mA, which is very conservative, then the resistor we need is given by:

$$R = V/I = 1.7/8 = 0.212 \text{ k}\Omega$$

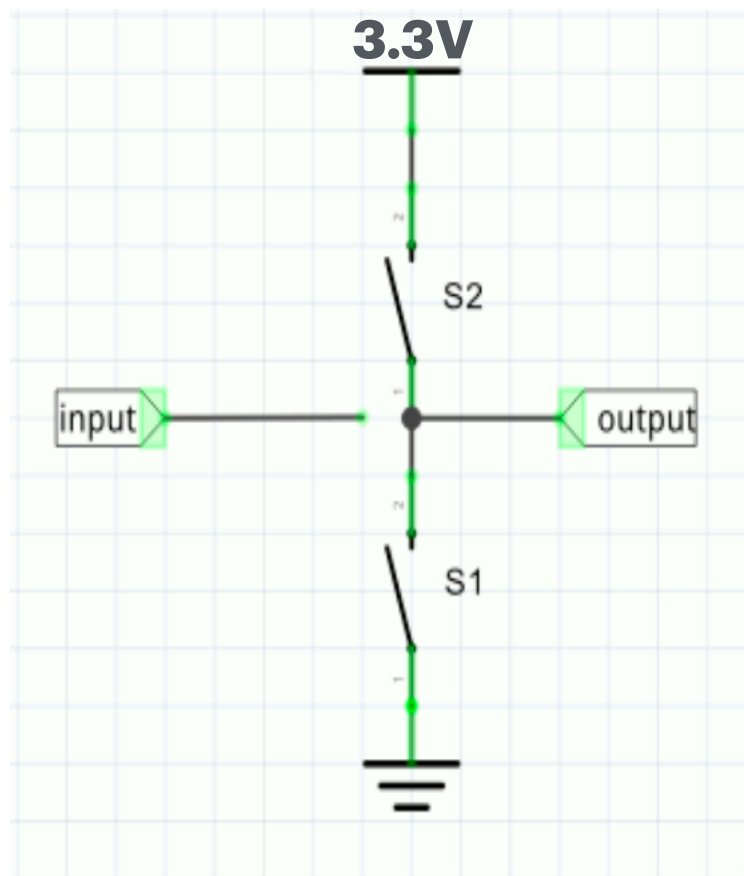




# Setting Drive Type

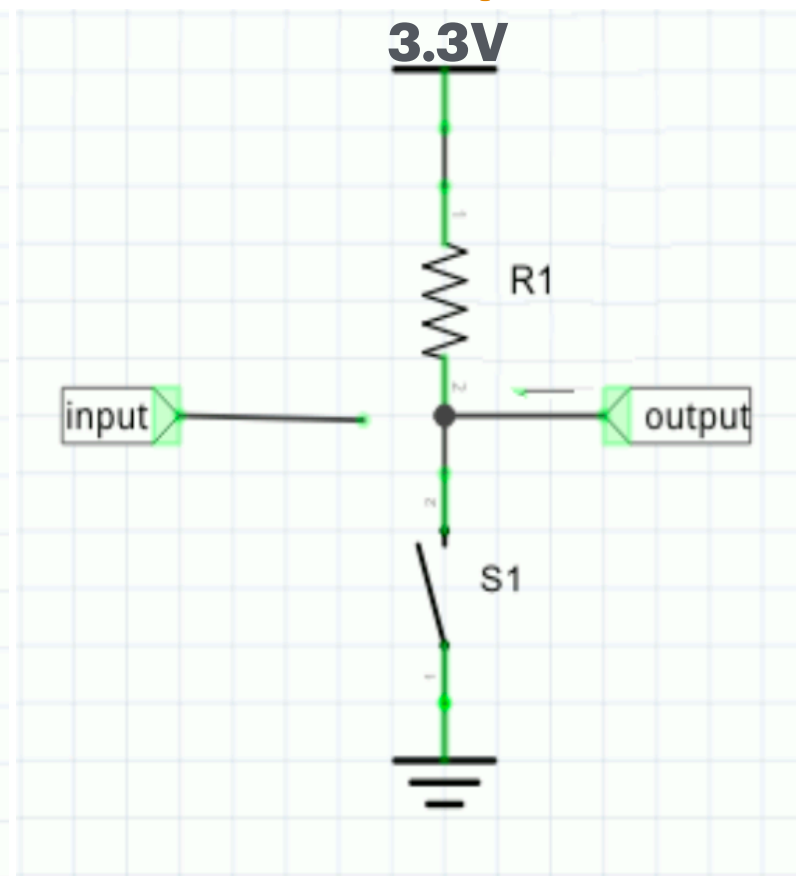
One of The GPIO output can be configured into one of a number of modes, but the most important is pull-up/down.

## Push-Pull



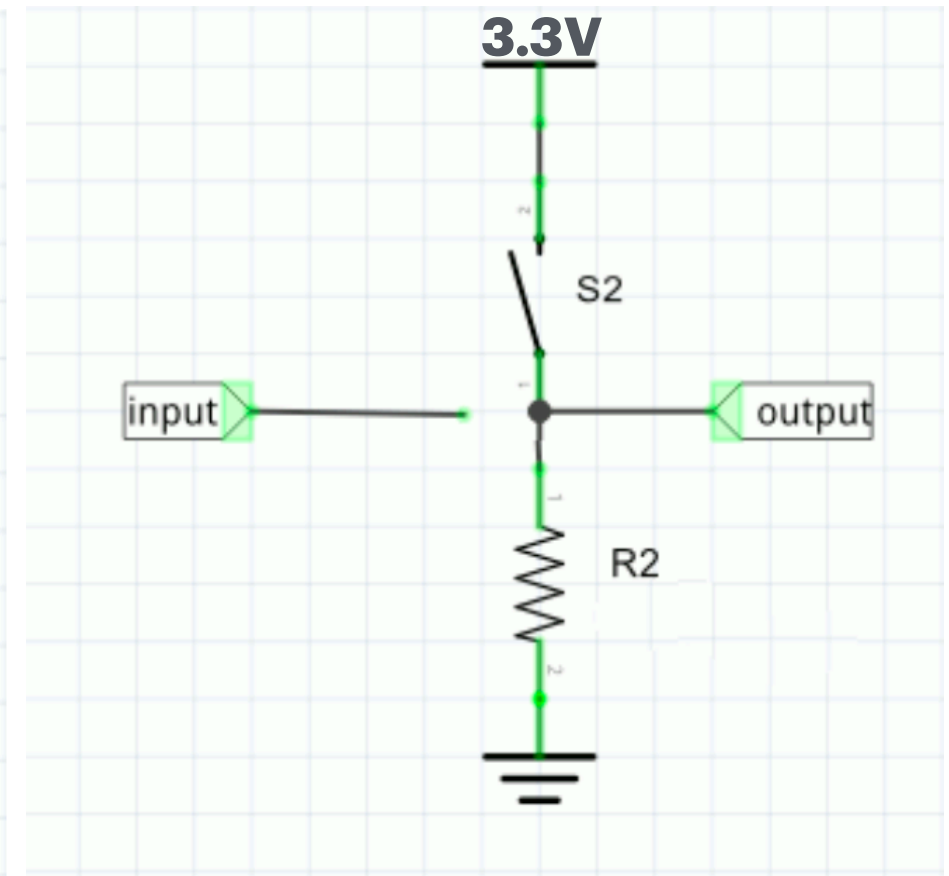
The circuit behaves like the two-switch. Only one switch is CLOSED ant time.

## Pull-Up



The circuit is equivalent to having a single switch. When the switch is closed, the output line is connected to ground and hence driven low. When the switch is open, the output line is pulled high by the resistor.

## Pull-Down



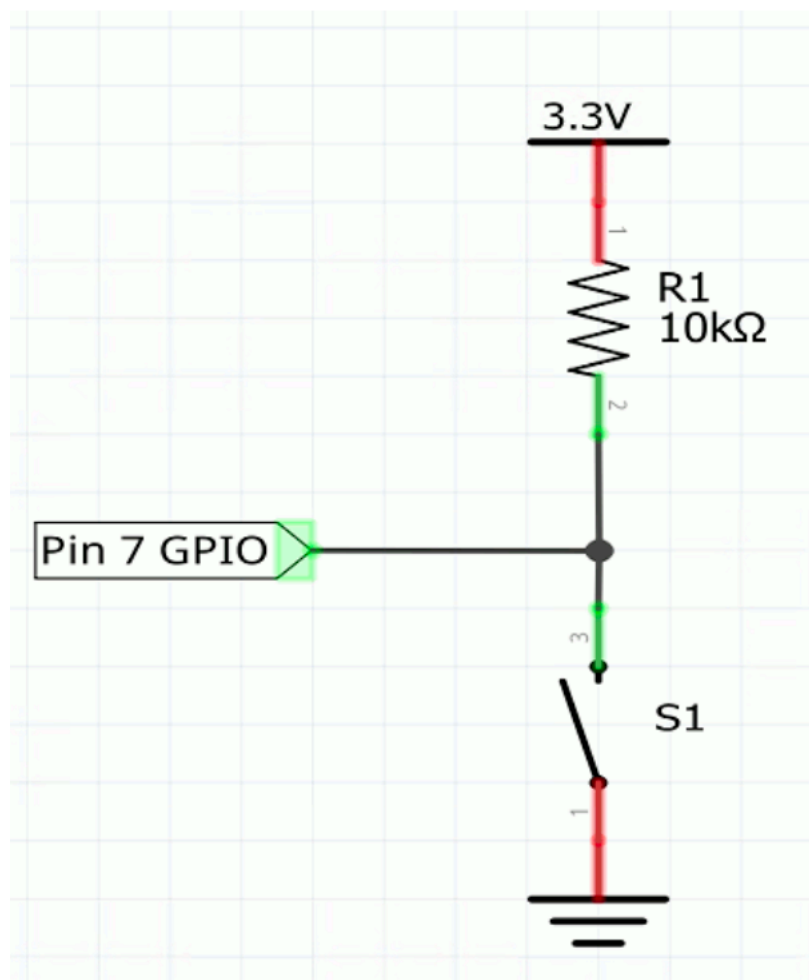
The the pull-down mode, which is the best mode for driving general loads, motors, LEDs, etc, is exactly the same as the pull-up only now the resistor is used to pull the output line low.



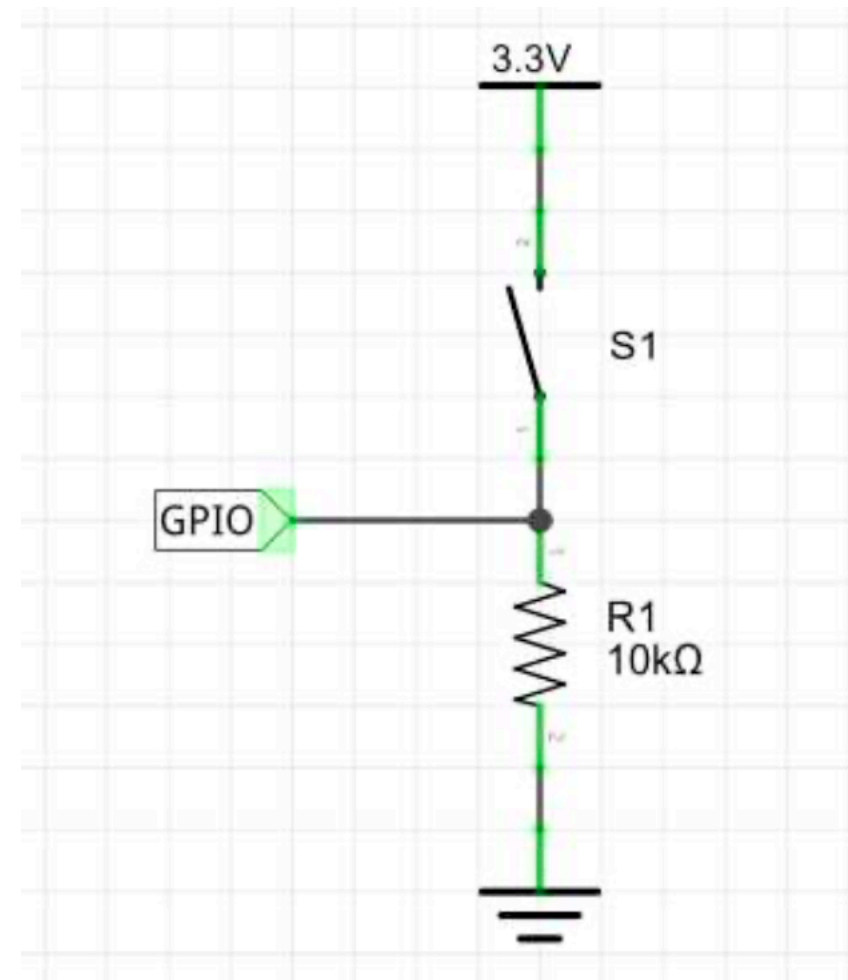
# Setting Drive Type

One of The GPIO output can be configured into one of a number of modes, but the most important is pull-up/down.

## Pull-Up



## Pull-Down



- The maximum current from any GPIO line should be less than 12mA and the total current should be less than 30mA.
- All of the GPIO lines work at 3.3V and you should avoid directly connecting any other voltage.





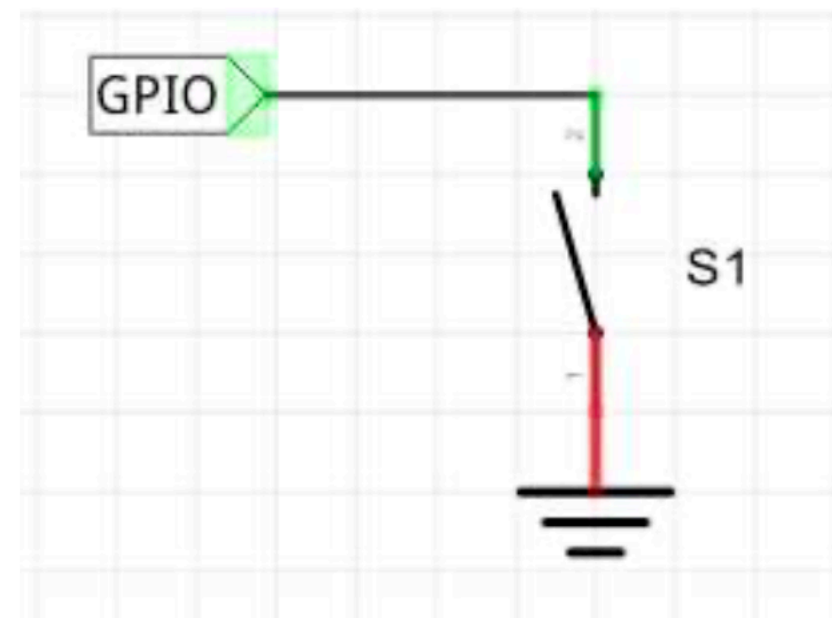
# Setting Drive Type

As the internal pull-up resistor is used, the switch can be connected to the line and ground without any external resistors:

```
from machine import Pin
import time
```

```
pinIN = Pin(22, Pin.IN, Pin.PULL_UP)
pinLED = Pin("LED", Pin.OUT)
```

```
while True:
    if pinIN.value():
        pinLED.on()
    else:
        pinLED.off()
    time.sleep(1)
```







## Programming the Pico

Learn Coding and Electronics with the Raspberry Pi Pico

Simon Monk

# Setting Drive Type

A convention, when connecting switches to a digital input, is to use what is called a pull-up resistor (Figure 7-1) connected to the digital input, that biases the input towards 3.3V (high). The switch is then connected between the digital input and *GND* (0V), so that when the switch is pressed the digital input is connected to GND. Think of the pull-up resistor as a spring that pulls the GPIO pin high unless pulled the other way more strongly by the switch.

This means that when the switch is not pressed the digital input is at 3.3V (high) and when the switch is pressed, the input becomes 0V (low). This can be confusing as, logically, high usually means *on* rather than *off* – but, that's fine, we can just compensate for this in the code.

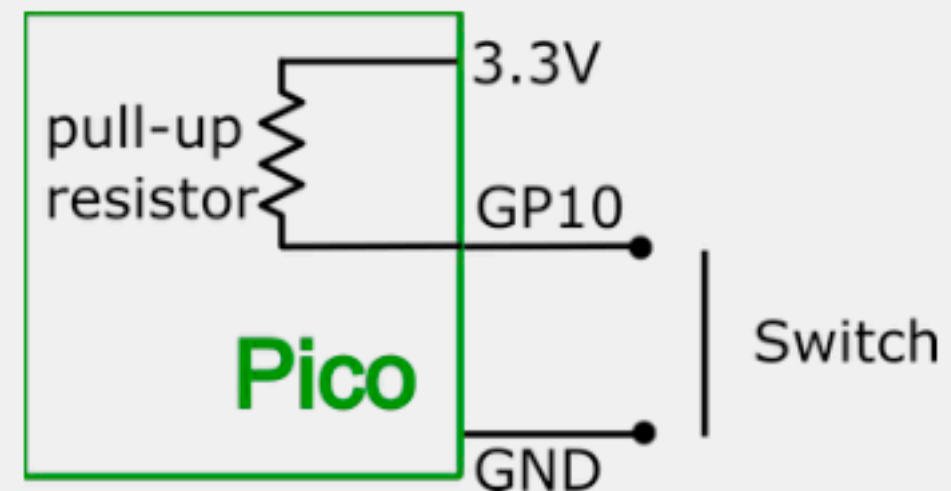


Figure 7.1. A pull-up resistor and digital input.



# Interrupts

An interrupt is a hardware mechanism that stops the computer doing whatever it is currently doing and makes it transfer its attention to running an interrupt handler. **Using interrupts means the outside world decides when the computer should pay attention to input and there is no need for a polling loop.** Most hardware people think that interrupts are the solution to everything and polling is inelegant and only to be used when you can't use an interrupt. This is far from the reality. There is a general feeling that real-time programming and interrupts go together and if you are not using an interrupt you are probably doing something wrong. In fact, the truth is that if you are using an interrupt you are probably doing something wrong. So much so that some organizations are convinced that interrupts are so dangerous that they are banned from being used at all.

Interrupts are only really useful when you have a low-frequency condition that needs to be dealt with on a high-priority basis. Interrupts can simplify the logic of your program, but rarely does using an interrupt speed things up because the overhead involved in interrupt handling is usually quite high.



# Interrupts

The Pico supports 32 distinct interrupts, but only 26 are actually used. All of the user GPIO lines act together to create a single IO interrupt. A subtle point is that each of the Pico's two processors can respond at the same time to an IO interrupt caused by a different GPIO line – that is, the IO interrupts are not shared between cores. In all other cases interrupts can only be enabled on one core at a time.

There is only **a single interrupt** for all of the **GPIO lines** means it is up to the **interrupt handler** to work out which line caused the interrupt and to reset it after dealing with it. Which GPIO line can generate an interrupt is specified by the same bits in the same register that we have been using as event indicators in the earlier sections. If an interrupt is enabled for a given line and a given event then the interrupt will occur if that bit is set to one.

The events that you can use are the same as before:

- GPIO\_IRQ\_LEVEL\_LOW
- GPIO\_IRQ\_LEVEL\_HIGH
- GPIO\_IRQ\_EDGE\_FALL
- GPIO\_IRQ\_EDGE\_RISE



# Interrupts

```
from machine import Pin
from utime import sleep

button = Pin(22, Pin.IN, Pin.PULL_UP)
pinLED = Pin("LED", Pin.OUT)

def handle_button(ignore):
    print('BUTTON PRESSED')
    if button.value():
        pinLED.off()
    else:
        pinLED.on()

button.irq(handle_button, Pin.IRQ_FALLING)
i = 0

try:
    while True:
        i += 1
        print(i)
        sleep(0.2)

except KeyboardInterrupt:
    print("Keyboard Interrupt")
```



# Interrupts

The Pico supports 32 distinct interrupts, but only 26 are actually used. All of the user GPIO lines act together to create a single IO interrupt. A subtle point is that each of the Pico's two processors can respond at the same time to an IO interrupt caused by a different GPIO line – that is, the IO interrupts are not shared between cores. In all other cases interrupts can only be enabled on one core at a time.

There is only **a single interrupt** for all of the **GPIO lines** means it is up to the **interrupt handler** to work out which line caused the interrupt and to reset it after dealing with it. Which GPIO line can generate an interrupt is specified by the same bits in the same register that we have been using as event indicators in the earlier sections. If an interrupt is enabled for a given line and a given event then the interrupt will occur if that bit is set to one.

The events that you can use are the same as before:

- GPIO\_IRQ\_LEVEL\_LOW
- GPIO\_IRQ\_LEVEL\_HIGH
- GPIO\_IRQ\_EDGE\_FALL
- GPIO\_IRQ\_EDGE\_RISE