Timing & the Time Library

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #8

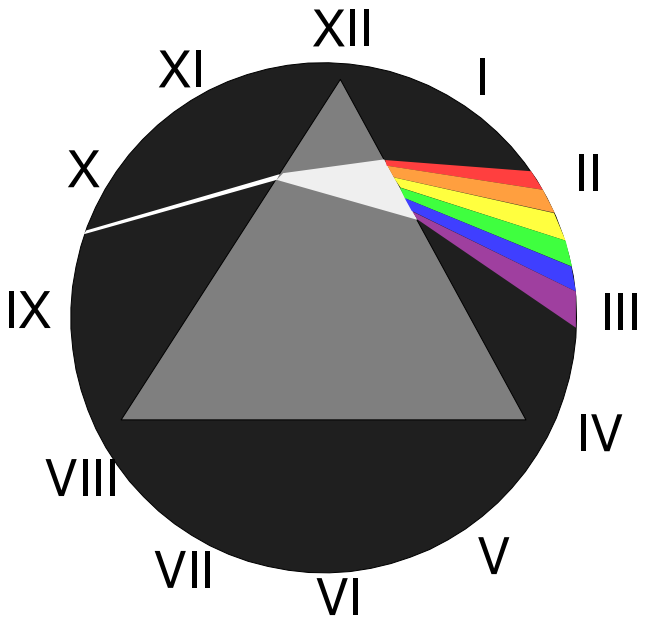
Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Introduction:

In this lecture, we look things related to time.

This includes measuring:

- The time between events
- The width of a pulse
- The period of a square wave
  - and hence its frequency



We'll also look at

- How to generate a square wave
  - Set the frequency and duty cycle

With this, we'll be able to

- Measure your reflex time,
- Measure distance using an ultrasonic range sensor,
- Measure resistance, capacitance, and temperature using a 555 timer, and
- Play a tune with your Pi-Pico

Time Library

One of the more useful libraries

You can see the functions included by typing:

```
>>> import time
>>> dir(time)
['__class__', '__name__', '__dict__', 'gmtime', 'localtime', 'mktime',
'sleep', 'sleep_ms', 'sleep_us', 'ticks_add', 'ticks_cpu',
'ticks_diff', 'ticks_ms', 'ticks_us', 'time', 'time_ns']
```

To measure time, the funcitons we're going to use are:

```
ticks_ms     time since power up in ms
ticks_us     time since power up in us
ticks_cpu    time since power up in cpu clocks (varies with uP)
             recommended you don't use ticks_cpu
```

How Long Does sleep(1) Take?

ticks_us() records time sinnce reset

- Record the time prior to *sleep(1)*
- Record the time after

The time difference is the execution time

- dt = 1,000,064us

note: This also includes the execution time of ticks_us()

```
from time import ticks_us, sleep

x0 = ticks_us()
sleep(1)
x1 = ticks_us()
print(x1 - x0)
```

shell

```
1000064
```

How Long Does sleep(1) Take (better)

The last measure is a little high

- Also includes ticks_us() execution time

You can subtract this out

- sleep(1) = 1,000,004us
- ticks_us() = 60us

```
from time import ticks_us, sleep

x0 = ticks_us()
sleep(1)
x1 = ticks_us()
x2 = ticks_us()
print(x1 - x0 - (x2-x1))
```

shell

```
1000004
```

How Long Does sleep(10) Take?

sleep(10) takes

- 10,000,006us

Note:

- The sleep() function is really accurate!
- We can measure time to 1us (!!)

```
from time import ticks_us, sleep

x0 = ticks_us()
sleep(10)
x1 = ticks_us()
x2 = ticks_us()
print(x1 - x0 - (x2-x1))
```

shell

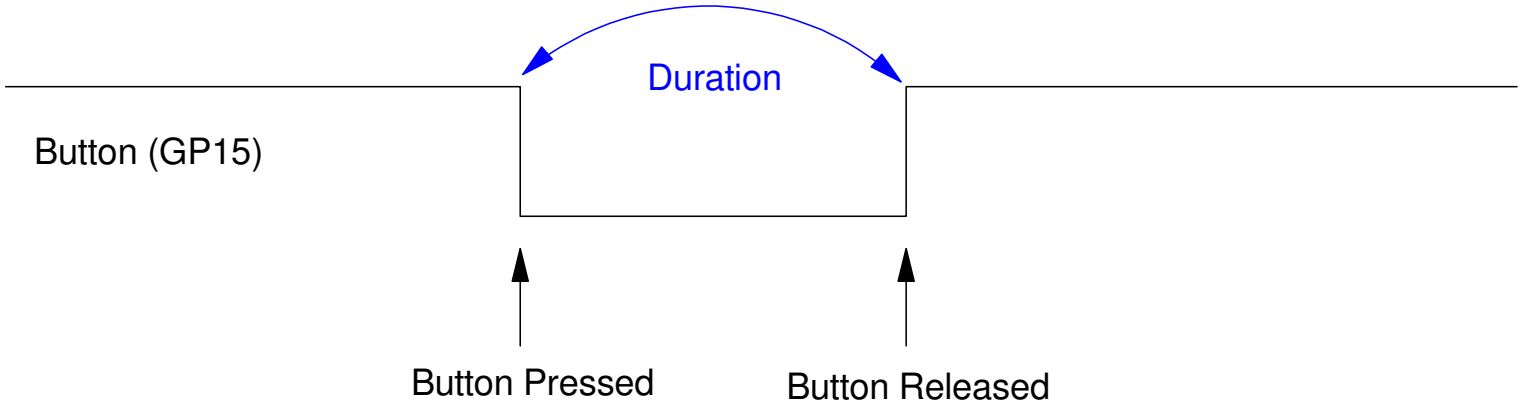
```
10000006
```

OK - so now that we can measure time, let's have some fun with it.

Button Press Game:

What's the shortest time I can press and release a button?

- Wait until you press the button (value goes to zero). Record that time.
- Then wait until you release the button (value goes to one). Record that time.
- The difference in time is how long you held the button down.



Button Press Game: Measure the time the button is held down.
Try to get the lowest score.

In Code:

In six attempts

- Shortest time was 39,496us
- 60us high due to ticks_us()

With a little more code,

- The time of ticks_us() can be removed
- The best score can be updated

```
from time import ticks_us, ticks_ms
from machine import Pin

Button = Pin(15, Pin.IN, Pin.PULL_UP)
while(1):
    while(Button.value() == 1):
        pass
    x0 = ticks_us()
    while(Button.value() == 0):
        pass
    x1 = ticks_us()
    print(x1-x0)
```

shell

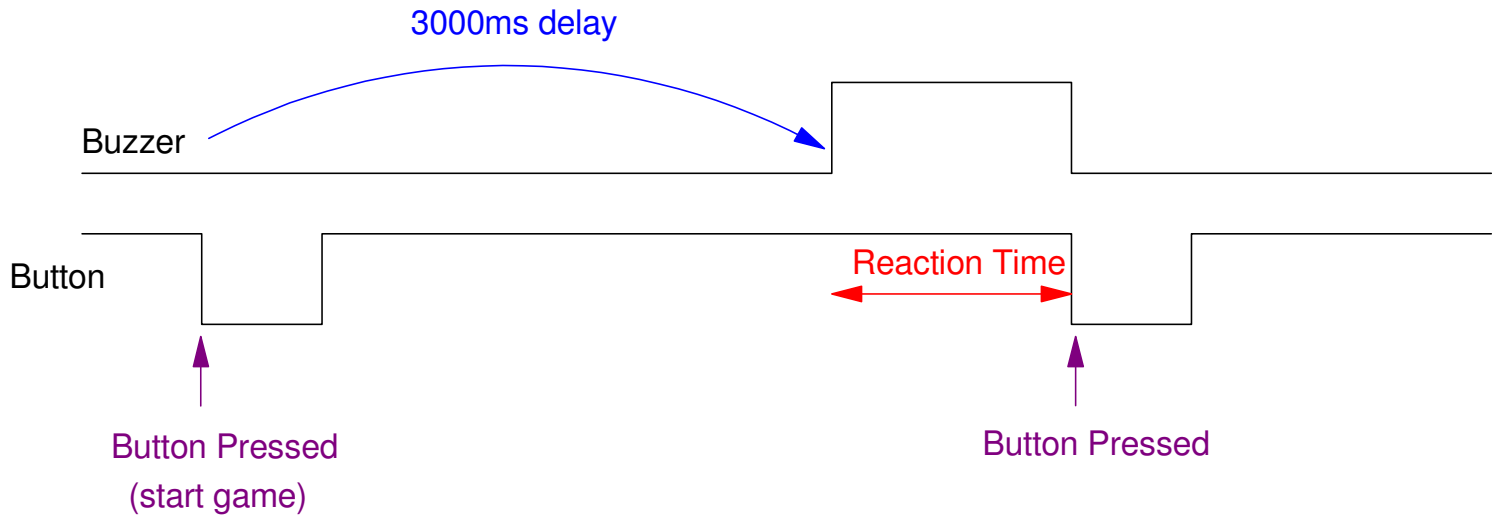
```
51494
48585
57623
55358
60112
39496
```

Reaction Time Game:

For some more fun, determine my reflex time.

- Start out by pressing a button.
- 3 seconds later, turn on the buzzer
- As soon as the buzzer turns on, press the button

The time delay from hearing the buzzer and pressing the button is your reflex time.



Reaction Time Game: Measure the time between when the buzzer turns on and you press a button

In code:

Wait for a button to be pressed and released

- starts the 3 second timer

The buzzer then turns on

- record that time

Wait for a button to be pressed

- record that time

Time difference = reaction time

- Best score = 125.309us

```
from time import ticks_us, sleep_ms
from machine import Pin

Buzzer = Pin(13, Pin.OUT)
Button = Pin(15, Pin.IN, Pin.PULL_UP)
while(1):
    while(Button.value() == 1):
        pass
    while(Button.value() == 0):
        pass
    sleep_ms(3000)
    Buzzer.value(1)
    x0 = ticks_us()
    while(Button.value() == 1):
        pass
    x1 = ticks_us()
    Buzzer.value(0)
    print(x1 - x0)
```

shell

```
134063
160489
125309
```

Code Improvements

- Make the delay random rathern than 3.000 seconds fixed
- Record your best time
- Keep track of mean and standard deviation...

Note: Once you can measure your reflex time, you can ask...

- What frequency works best?
- Is a solid tone or a series of beeps better?
- Do you respond to light faster than sound?
- What color of light are people most responsive to?
- Do your reflexes improve after exercise?
- What affect does caffine have on your reflexes?
- etc.

New Problem: Generate a Fixed Frequency

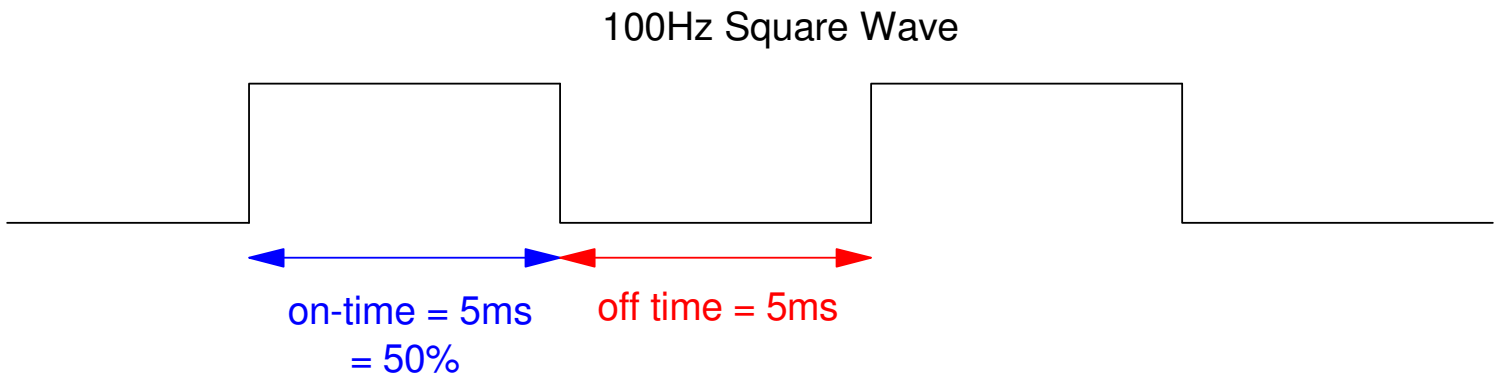
Use time.sleep()

- Ties up the processor while waiting

Use the *PWM* function from library *machine*

- Better way to do it
- Uses hardware to set the frequency rather than software
- Leaves the processor free to do other stuff

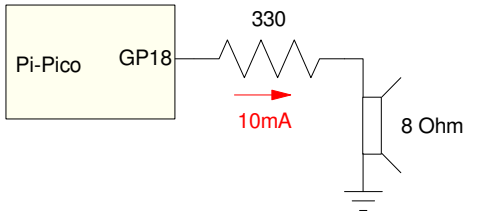
For example, to set up GP18 to output a 100Hz square wave



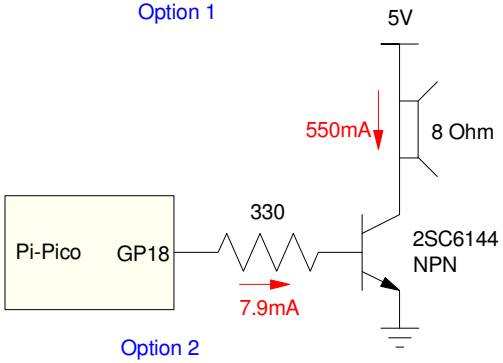
Hardware:

Several ways to connect a Pico to a speaker:

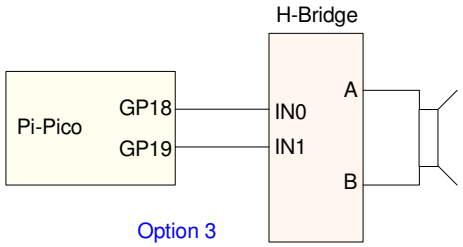
- With a 330 Ohm wire (to limit current)
  - Easy - not too loud
- With an NPN transistor
  - Louder, more annoying
  - Can increase voltage to 5V or more
- With an H-bridge
  - Loud, annoying
  - Can increase current to 3A
  - Can drive speaker forward & back



Option 1



Option 2



Option 3

Software:

- GP18 drives the speaker

The code would be:

- Pin18 is a PWM output
- Frequency = 100Hz
- Duty cycle = 50%

```
from machine import Pin, PWM

Spkr = Pin(18, Pin.OUT)
Spkr = PWM(Pin(18))
Spkr = freq(100)
Spkr.duty_u16(32768)

while(1):
    pass
```

note:

- 100% duty cycle = 65,535
- Hardware takes care of the rest

Software (take 2)

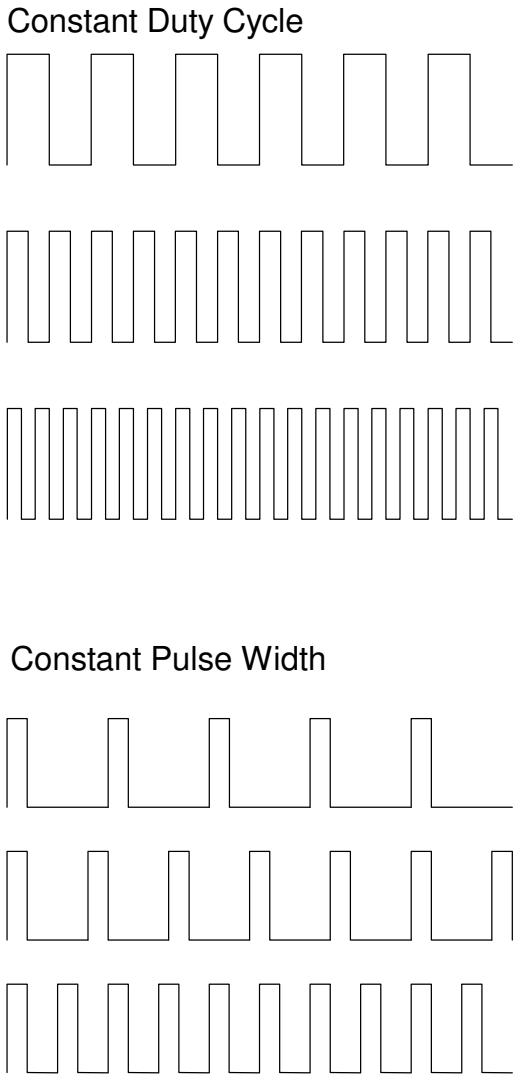
duty_u16() maintains a constant duty cycle

- Duty cycle doesn't change with frequency
  - 0 0% duty cycle square wave (off)
  - 32768 50% duty cycle square wave
  - 65535 100% duty cycle square wave (on)

duty_ns() maintains constant pulse width

- On-time doesn't change with frequency
- Example: Pulse width = 5ms (fixed)

```
Spkr = Pin(18, Pin.OUT)
Spkr = PWM(Pin(18))
Spkr = freq(100)
Spkr.duty_ns(5_000_000)
```



Turning a speaker on & off

On:

- Set the duty cycle to 50%
- duty_u16(32,768)

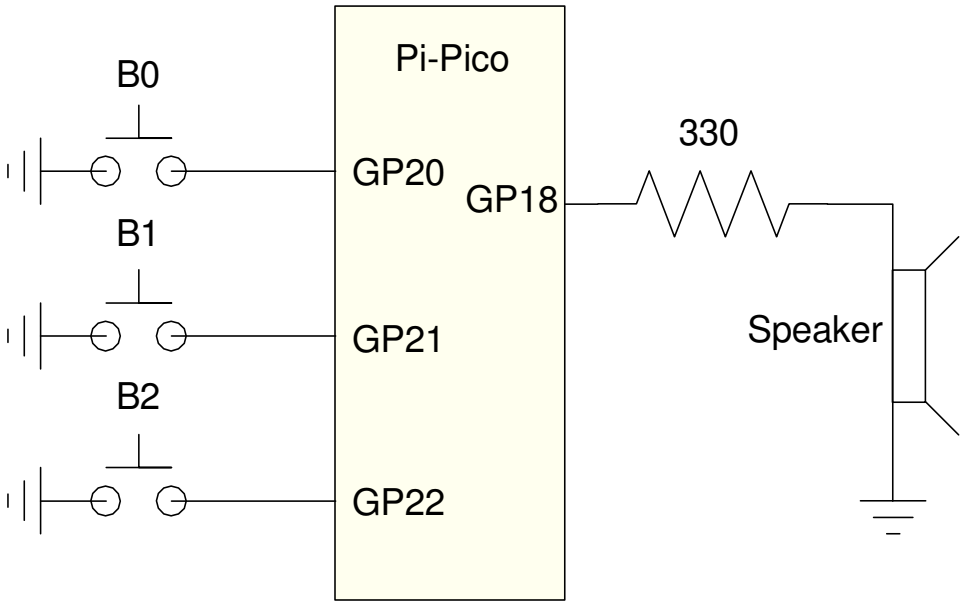
Off:

- Set the duty cycle to 0% (0)
- duty_u16(0)

```
from machine import Pin, PWM
from time import sleep_ms

Spkr = Pin(18, Pin.OUT)
Spkr = PWM(Pin(18))
Spkr = freq(100)
Spkr.duty_u16(32768)

while(1):
    Spkr.duty_16(32768)      # buzzer on
    sleep_ms(500)
    Spkr.duty_16(0)          # buzzer off
    sleep_ms(500)
```

3-Key Piano: Now that we can play a single note, play three different notes

- When GP20 is 0 (button pressed), play 220Hz
- When GP21 is 0, play 250Hz
- When GP22 is 0, play 280Hz
- Otherwise, remain silent



Code:

Pull-up resistors for buttons

- Press = logic 0

Check buttons

- if-statements
- Sets the frequency
- Sets the duty cycle
- Keep playing while button pressed

Turn off speaker if no button is pressed

```
import time
from machine import Pin, PWM

# Construct PWM object, with LED on Pin(25)
Spkr = PWM(Pin(18))
B0 = Pin(20, Pin.IN, Pin.PULL_UP)
B1 = Pin(21, Pin.IN, Pin.PULL_UP)
B2 = Pin(22, Pin.IN, Pin.PULL_UP)

while(1):
    if(B0.value() == 0):
        Spkr.freq(220)
        Spkr.duty_u16(32768)
        while(B0.value() == 0):
            pass
    if(B1.value() == 0):
        Spkr.freq(250)
        Spkr.duty_u16(32768)
        while(B1.value() == 0):
            pass
    if(B2.value() == 0):
        Spkr.freq(280)
        Spkr.duty_u16(32768)
        while(B2.value() == 0):
            pass
    pwm.duty_u16(0)
```

What happens if two buttons are pressed?

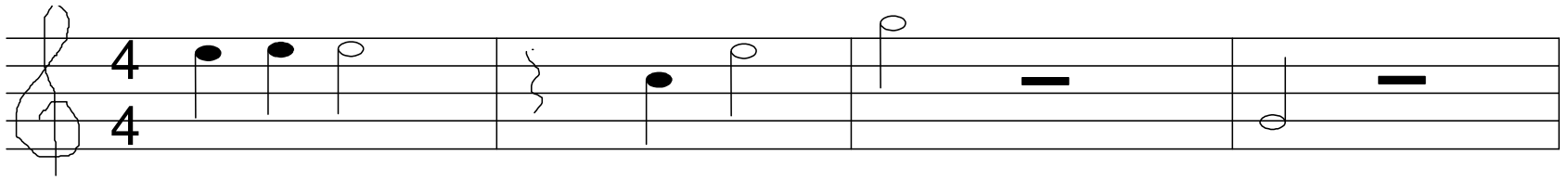
Piano does something

- Code dictates what happens

```
import time
from machine import Pin, PWM

# Construct PWM object, with LED on Pin(25)
Spkr = PWM(Pin(18))
B0 = Pin(20, Pin.IN, Pin.PULL_UP)
B1 = Pin(21, Pin.IN, Pin.PULL_UP)
B2 = Pin(22, Pin.IN, Pin.PULL_UP)

while(1):
    if(B0.value() == 0):
        Spkr.freq(220)
        Spkr.duty_u16(32768)
        while(B0.value() == 0):
            pass
    if(B1.value() == 0):
        Spkr.freq(250)
        Spkr.duty_u16(32768)
        while(B1.value() == 0):
            pass
    if(B2.value() == 0):
        Spkr.freq(280)
        Spkr.duty_u16(32768)
        while(B2.value() == 0):
            pass
    pwm.duty_u16(0)
```

Super Mario Brothers Theme:

Play the first four bars of SuperMario Brothers:



Translation:

- Play E4 for 1/4 beat
- Play E4 for 1/4 beat
- Play E4 for 1/2 beat
- pause for 1/4 beat
- Play C4 for 1/4 beat
- Play E4 for 1/2 beat
- Play G4 for 1/2 beat
- pause for 1/2 beat
- Play G3 for 1/2 beat
- pause for 1/2 beat

Option 1: Create a subroutine for each note

Create a subroutine which plays a given note for a fix duration:

- Hz is the frequency of the note in Hz
- Eighths sets the duration of the note in 1/8th notes
- The last 50ms of each note is silent, allowing you to hear the same note played twice:

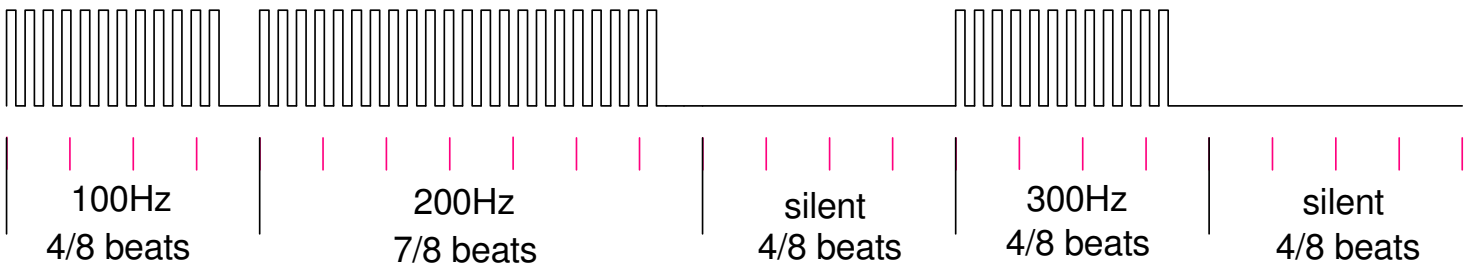
```
def Play(Hz, Eighths):
    if(Hz > 0):
        Spkr.freq(int(Hz))
        Spkr.duty_u16(32768)
    else:
        Spkr.duty_u16(0)
    time.sleep_ms(75 * Eights - 50)
    Spkr.duty_u16(0)
    time.sleep(0.05)
```

With this routine, you could play

- 100Hz for 4/8th beat, then
- 200Hz for 7/8th beat
- Go silent for 4/8th beat
- 300Hz for 4/8 beat

with the following program:

```
Play(100, 4)
Play(200, 7)
Play(0, 4)
Play(300,4)
Play(0,4)
```



Output of the Play() subroutine

Adding a bunch of *play()* routines plays the tune

- place in a while(1) loop
- pause one second between songs

```
from time import sleep_ms
from machine import Pin, PWM

Spkr = PWM(Pin(18))

def Init():
   Spkr.freq(100)
   Spkr.duty_u16(0)

def Play(Hz, Eighths):
    if(Hz > 0):
        Spkr.freq(int(Hz))
        Spkr.duty_u16(32768)
    else:
        Spkr.duty_u16(0)
    sleep_ms(75 * Eights - 50)
    Spkr.duty_u16(0)
    sleep_ms(50)

Init()
while(1):
    play(E4, 2)
    play(E4, 2)
    play(E4, 4)
    play(0,  2)
    play(C4, 2)
    play(E4, 4)
    play(G4, 4)
    play(0,  4)
    play(G3, 4)
    play(0,  4)
    sleep_ms(1000)
```

Placing the tune into an array is more stylish

- Set the notes and duration

Allows different tunes by changing two lines of code

```
from time import sleep_ms
from machine import Pin, PWM

Spkr = PWM(Pin(18))

def Init():
   Spkr.freq(100)
   Spkr.duty_u16(0)

def Play(Hz, Eighths):
    if(Hz > 0):
        Spkr.freq(int(Hz))
        Spkr.duty_u16(32768)
    else:
        Spkr.duty_u16(0)
    sleep_ms(75 * Eights - 50)
    Spkr.duty_u16(0)
    sleep_ms(50)

G3 = 195
C4 = 262
E4 = 330
G4 = 392

Notes   =  [E4, E4, E4, 0, C4, E4, G4, 0, G3, 0]
Dur     =  [2,  2,  4,  2,  2,  4,  4, 4,  4, 4]

def Play_Tune():
    for i in range(0, len(Notes)):
        Play(Notes[i], Dur[i])
Init()
while(1):
    Play_Tune()
    time.sleep(1)
```

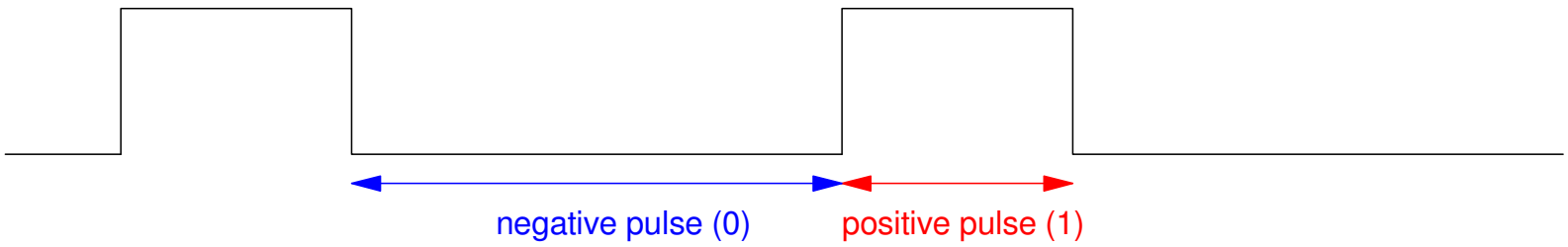
Measuring Pulse Width

A little more stylish way to measure a pulse width is to use the *time_pulse_us()* function in library *machine*.

The format for using this funciton is:

```
Tp = time_pulse_us(17, 1, 100_000)  # time of a positive pulse
Tm = time_pulse_us(17, 0, 100_000)  # time of a negative pulse
```

- The first number (17) is the pin you're trying to measure.
- The second number (1, 0) indicated whether you're measuring a positive pulse (1) or negative pulse(0)
- The third number is the max time in microseconds. If a pulse isn't detected withing this time it kicks out rather than being stuck in an infinite loop.



time_pulse_us() lets you measure the width of a negative or positive pulse

Example,

- Measure the pulse width of pin #17 (positive pulse default)
- Measuring the negative pulse (0)
  - Time the button was held down
- Time-out if longer than 5,000,000us

Shortest time was 39,496us

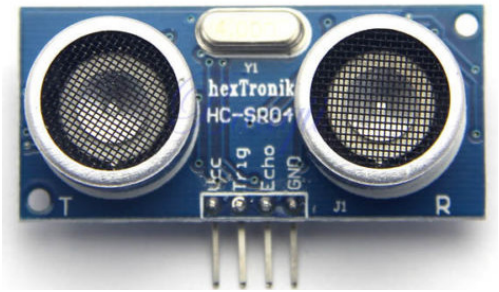
```
from machine import Pin, time_pulse_us

Button = Pin(17, Pin.IN, Pin.PULL_UP)
while(1):
    x = time_pulse_us(17, 0, 5_000_000)
    print(x)
```

shell

```
51494
48585
57623
55358
60112
39496
```

Ultrasonic Range Sensor:

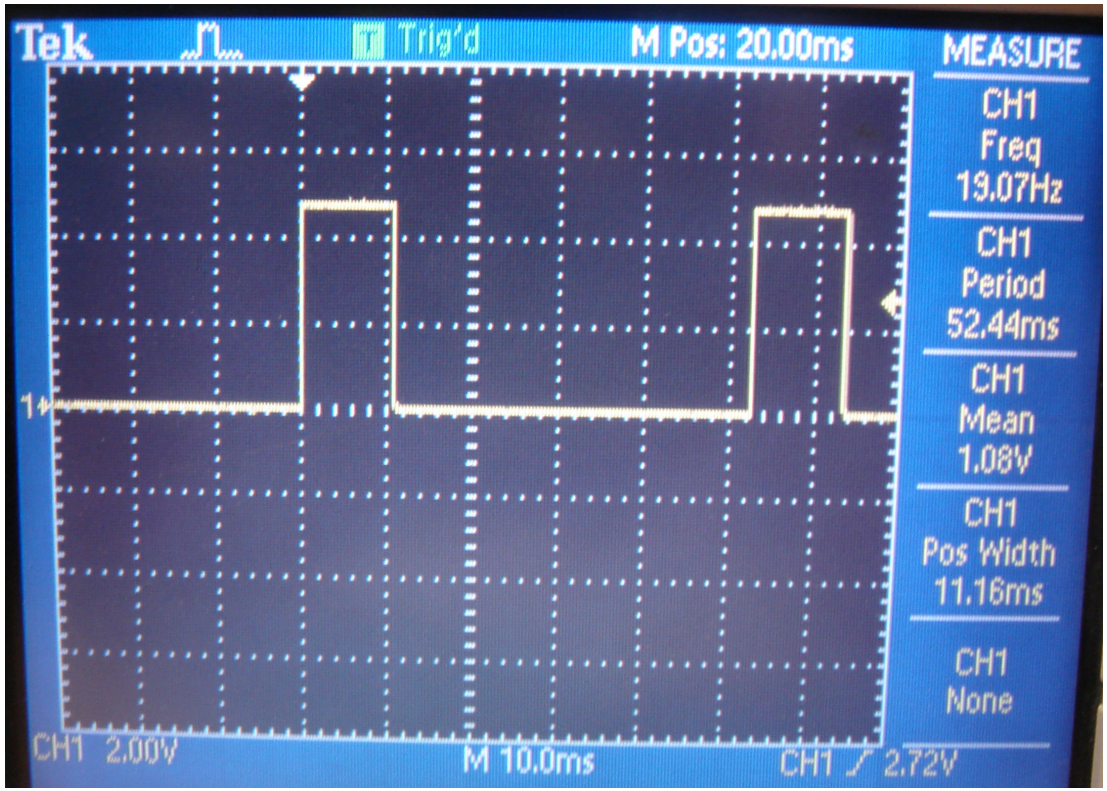
With this funciton you can measure distance using an ultrasonic range sensor.



This device has four pins:

- Vcc: input: +5V
- Trig: input: Square wave from the RPi-Pico
- Echo: output: Pulse to the RPi-Pico (note: you need to drop this down to 3.3V)
- Gnd: input: 0V

Each time you sent from the range sensor. The time it takes for the sound to return is the duration of the pulse on Echo. For example, if Trig is a 20Hz square wave, the signal on Echo might look like this:

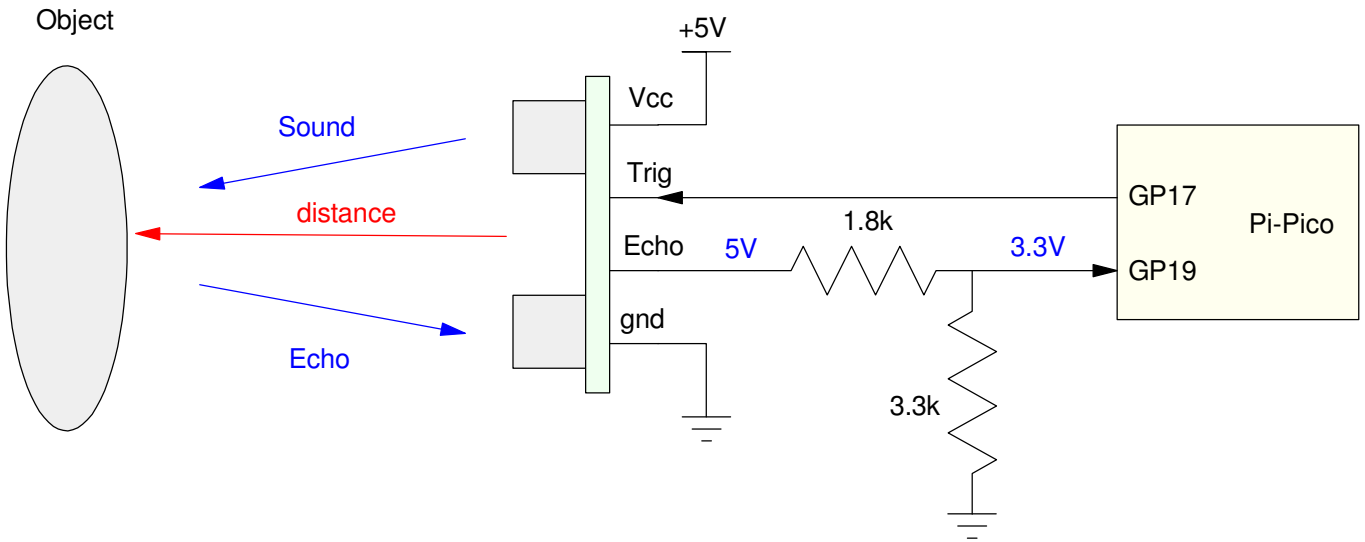


The pulse width is a measure of distance to an object. Assuming the speed of sound is 343 m/s, each microsecond of pulse width corresponds to a distance of

$$2d = (343\tfrac{m}{s}) \cdot (1\mu s)$$

$$d = 171.5\mu m$$

(the 2 is due to the sound having to travel to and back from the object - so the effective distance the sound travels is 2d)

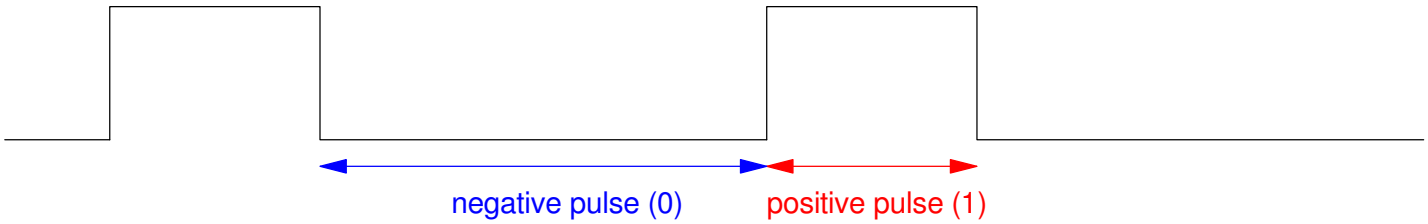


With the range sensor connected to pins 17 (trigger) and 19 (echo), the program would look like:

```
from machine import Pin, PWM, time_pulse_us
from time import sleep_ms

TRIG = 17
ECHO = 19

def setup():
    global p_Trig, p_Echo
    p_Trig = Pin(TRIG, Pin.OUT)
    p_ECHO = Pin(ECHO, Pin.IN)
    p_Trig = PWM(Pin(TRIG))
    p_Trig.freq(50)
    p_Trig.duty_ns(1000)
    p_Echo = Pin(ECHO, Pin.IN, Pin.PULL_UP)

def distance():
    mm = time_pulse_us(ECHO, 11) * 0.1715
    return mm

# Main Routine
setup()
while(1):
    dis = distance()
    print (dis, 'mm')
    sleep_ms(300)
```

Measure Period (or frequency)

With *time_pulse_us()* you can measure the positive or negative pulse of a square wave. Add the two together and you get the period.



Measure the period of a 100Hz square wave

- Set up GP18 to be
  - a 100Hz square wave with
  - a positive pulse of 10ms (10,000 us)
- Set up GP17 to be an input pin
- Short pin 18 to pin 17
- Measure the period of the signal on GP17

```
from machine import Pin, PWM, time_pulse_us
from time import sleep_ms

buzzer = Pin(18, Pin.OUT)
buzzer = PWM(Pin(18))
buzzer = freq(100)
buzzer.duty_ns(10000)

Button = Pin(17, Pin.IN, Pin.PULL_UP)
while(1):
    x = time_pulse_us(17, 1, 100_000)
    y = time_pulse_us(17, 0, 100_000)
    print('Period = ,x+y,' us')
    sleep_ms(100)
```

shell

```
Period = 9808 us
```

Measure Resistance (LM555 Timer)

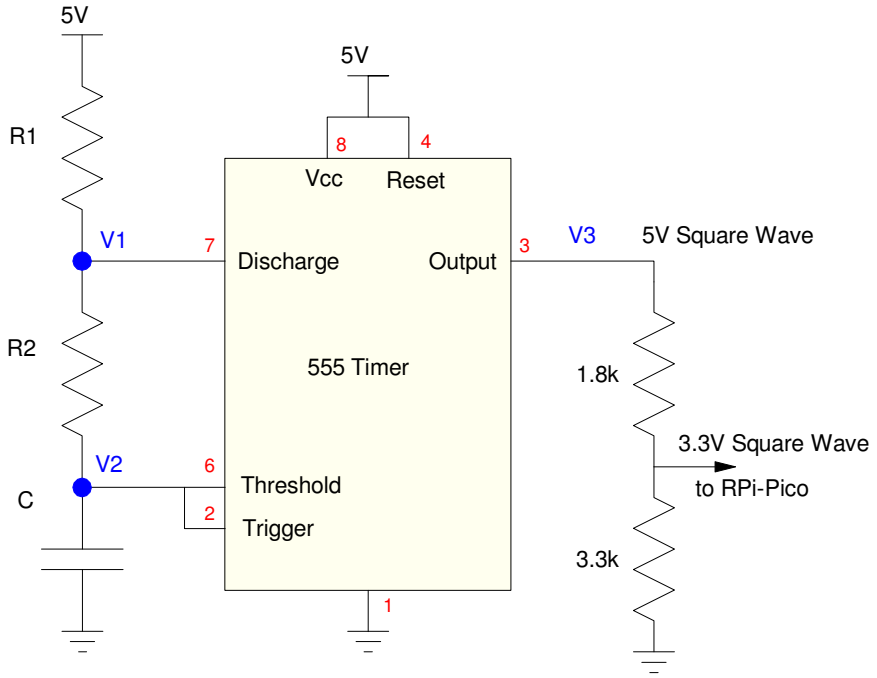
If you can measure frequency, you can measure resistance.

The following 555 timer outputs a square wave where

$$T_{on} = (R_1 + R_2) \cdot C \cdot \ln(2)$$

$$T_{off} = R_2 \cdot C \cdot \ln(2)$$

If R1 and C are known, you can determine R2 by measuring the period (or the off-time)



Assume R1 = 10k, R2 = 100k, and C = 0.1uF. Then

$$T_{off} = 6931.47\mu s$$

$$R_2 = 100k\Omega \cdot \left(\frac{T_{off}}{6931.47\mu s}\right) = 14.427 \cdot T_{off}(\mu s)$$

Code:

```
from machine import Pin, PWM, time_pulse_us
from time import ticks_cpu, ticks_ms, ticks_us

T555 = Pin(17, Pin.IN, Pin.PULL_UP)
while(1):
    Toff = time_pulse_us(17, 0, 100_000)
    R2 = 14.427 * Toff
    print(R2)
    sleep_ms(100)
```

Measure Temperature (555 Timer)

If you can measure resitance, you can measure tempeature. Replace R2 with a thermistor, such as

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

and you can compute temperature in degees C (T) as a funciton of pulse width.

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

or

$$T = \left(\frac{3905}{\ln\left(\frac{14.427 \cdot T_{off}}{1000}\right)+\left(\frac{3905}{298}\right)}\right) - 273$$

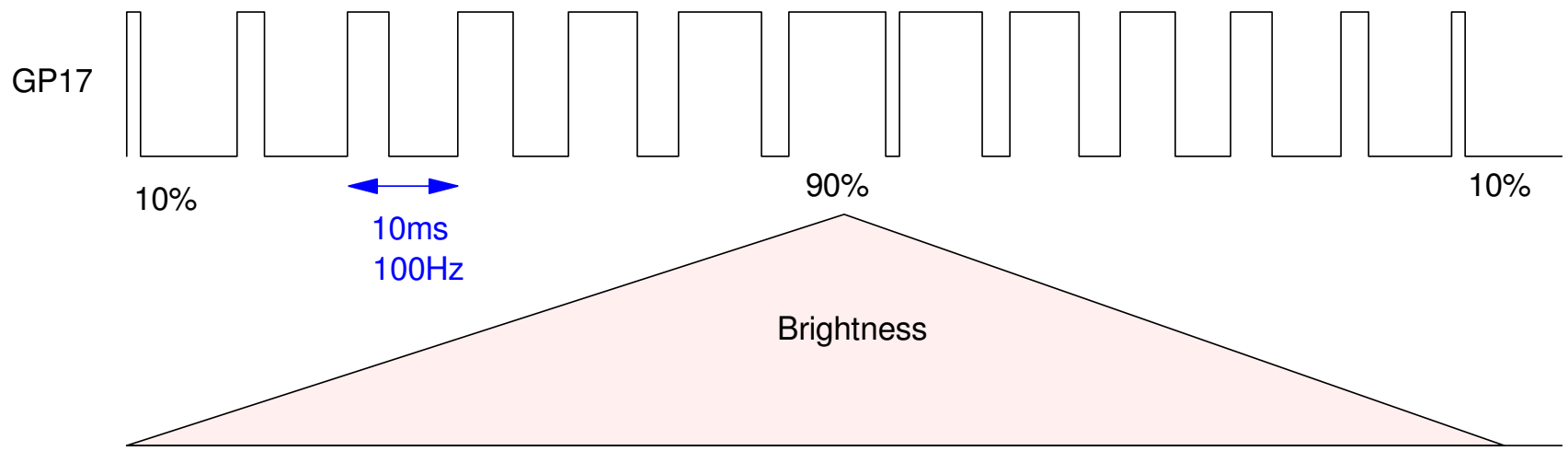
Code

```
from machine import Pin, PWM, time_pulse_us
from time import ticks_cpu, ticks_ms, ticks_us
from math include log

T555 = Pin(17, Pin.IN, Pin.PULL_UP)
while(1):
    Toff = time_pulse_us(17, 0, 100_000)
    R2 = 14.427 * Toff
    T = 3905 / ( log(R/1000) + (3905/298) ) - 273
    print(T)
    sleep_ms(100)
```

This is termed *theoretical calibration*: given the reading, go backwards through the calculations to get the temeprature.

Vary Brightness of LED

Finally, by varying the duty cycle, you can vary the brightness of an LED. The following code makes the LED on GP17 vary from 0% on to 100% on then back over and over again



Code:

- LED connected to GP17
- 0% to 100% duty cycle

```
from time import sleep_ms
from machine import Pin, PWM

LED = Pin(17, Pin.OUT)
LED = PWM(Pin(17))
LED.freq(100)

x = 0
dx = 100

while(1):
    x += dx
    LED.duty_u16(x)
    if(x > 65000):
        dx = -abs(dx)
    if(x <= 0)
        dx = abs(dx)
    sleep_ms(1)
```

Summary:

The Pi-Pico is really quite versitile. With it, you can

- Output square waves at a given frequency and duty cycle
- Measure time to one micro-second
- Mesure the width of a pulse (positive or negative),

amoung other things.

Add in a sensor, and you can measure distance, temperature, light, etc.