Refer to last class #3 **Open program ReadPot\_GP26\_DimLED\_GP2.py**

from machine import Pin, ADC, PWM

import utime

PWM1A=PWM(Pin(2)) # GP2 pin 4

PWM1A.freq(1000)

PotWiperCounts=ADC(Pin(26)) # can also call machine.ADC(3)

#GP2OFFSET=288 # The PotWiperCounts when the pot is full CCW

#GP2GAIN=1.004 # GP2 read 65535 at full cw, however once 288 counts are

# subtracted from it, it needs multiplied by the ratio to make it back to 65535 full scale

while True:

print(PotWiperCounts.read\_u16())

# PotWiperCountsReal=PotWiperCounts\*1.0

# PotWiperCountsReal=(PotWiperCountsReal-(GP2OFFSET))

# yes, in code you can read a value, do math, and put the value back to the same name

# PotWiperCountsReal=1.004\*PotWiperCounts

duty=PotWiperCounts.read\_u16()

PWM1A.duty\_u16(duty)

utime.sleep(.05)

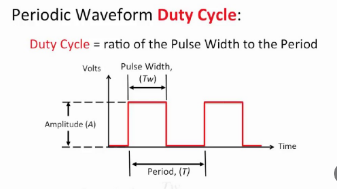
# print("Adjusted GP2 value is:", PotWiperCountsReal)

# MAKE a NOTE what the PotWiperCounts are at full CW - full up\_\_\_\_\_\_\_\_

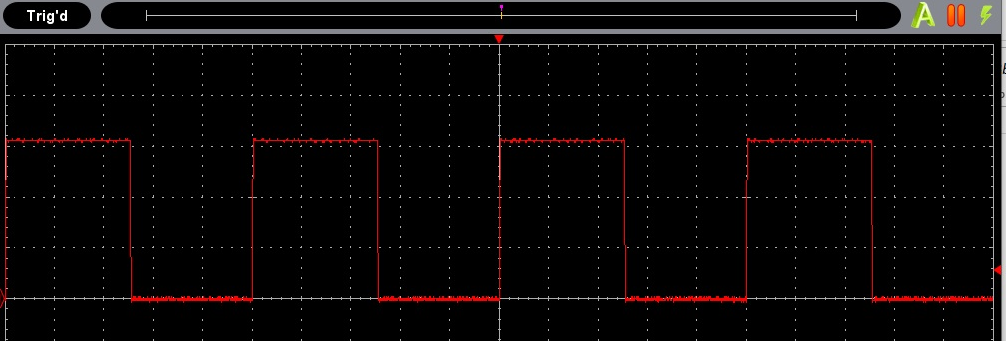
# MAKE a NOTE what the PotWiperCounts are at full CCW - full down\_\_\_\_\_\_\_\_

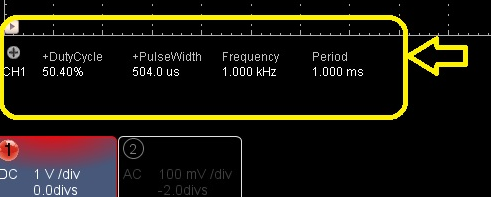
# The value at full CCW should be zero. The value it really shows is called OFFSET !!

# The next program you will correct the OFFSET and then also the gain error - The value at full CW !



MP720017 MultiComp USB oscilloscope

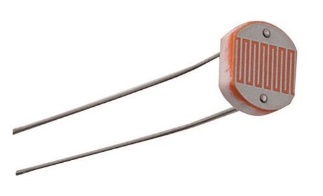




By adjusting the pot, observe change to duty cycle and pulse width.

Change the PWM in the code. Does Pico follow it well ?

-----------------------------------------------------------------------------------------------

****

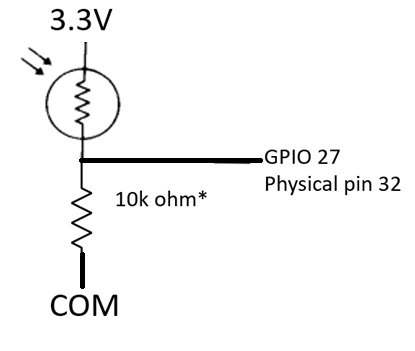
A photoresistor or photocell is a light-controlled variable resistor.

* The resistance of a photoresistor decreases with increasing light.
* These photoresistors are built to have the same spectral response as human eyes.
* The difference between component’s varies a lot, in bright light 100 ohms to 400 ohms.

In the dark, measures anywhere from 200k to 2 Mohm.

* Because of the low resistance in bright light, a series resistor is needed to limit the photo resistor current. That series resistor also creates a nice voltage divider to wire into GPIO for brightness detection.

The photoresistor datasheet details about light levels and range the photoresistor can be applied to.



Wired this way, the brightest light has a highest GPIO voltage & numeric value of analog input, as the resistor value is very low.

Darkness , the resistor value is high, much bigger than the 10K ohm resistor - the GPIO analog input value approaches zero.

Voltage divider formula:

Vgpio/ Vcc = 3.3V \*(10 k ohm / (Rphotoresistor + 10kohm))

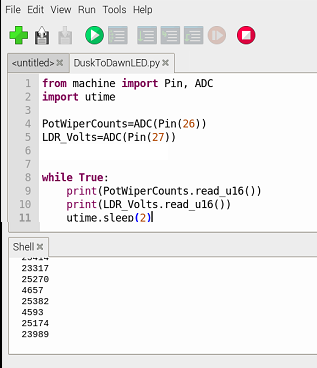
**Open program ReadPot\_GP26\_A0.py, and save as DuskToDawnLED.py**

**So lets first write a program to switch on the LED when the it gets dark outside**.

**Your house may have outside lights with this same photoresistor circuit.**

**First add in the code line to read the LDR\_Volts at GPIO 27 (A1), run the code to see the value.**

**Put your finger over the LDR and observe the value change, it should get smaller.**

****

**After proving the analogs are reading, then complete the coding**

**DuskToDawnLED.py (Building your own Dusk to Dawn light)**

from machine import Pin, ADC

import utime

PotWiperCounts=ADC(Pin(26))

LDR\_Counts=ADC(Pin(27))

u16\_2\_volts = 3.3/(65535) #create a conversion factor

DuskToDawnLED= Pin(25, machine.Pin.OUT)#GP25 is the Pico mounted LED

DuskToDawnLED.value(0)

LightOnAdjust=1.1 ## in volts at GPIO

while True:

#print(PotWiperCounts.read\_u16())

#print(LDR\_Counts.read\_u16())

LDR\_Volts=(LDR\_Counts.read\_u16())\*u16\_2\_volts

print(LDR\_Volts)

if ((LDR\_Volts) < (LightOnAdjust)):

DuskToDawnLED.value(1)

else:

DuskToDawnLED.value(0)

utime.sleep(2)

**more on hardware:**

If you’d like to turn on a bank of high power LEDS or a motor, you can use the GPIO output to drive a bipolar transistor base, or a MOSFET gate to switch on a lot bigger loads, or use a relay module.

**Building a LED lamp control with a TTP223 touch sensor and automatic power savings (a PWM dimmer when ambient is bright)**

* Touch to turn off & on, dimmer uses the photoresistor to dim in higher ambient light.
* The touch sensor wires to +3V, COM, and Pico pin 22(GPIO17)
* Leave the photoresistor divider circuit wired to Pin 32 (GPIO 27)
* Observe LED voltage behavior with LED and scope

**TouchLamp.py**

# use the touch switch to turn lamp off and on, PWM intensity set by photoresistor

from machine import ADC, PWM , Pin #pin means GPIO number

import time

OnSwitch=Pin(17,Pin.IN)

#PotWiperCounts=ADC(Pin(26))

LDR\_Counts\_U16=ADC(Pin(27)) # LDR\_Counts is the u16 volts value of the 10k resistor, 65535 if Light resistor = 0 , to 0 as it gets dark

LampBrightness= PWM(Pin(2))# physical pin 4

LampBrightness.freq(500)

DimmerDuty=65535

TurnOnLight=False

while True:

if ((OnSwitch.value()== 1) and (TurnOnLight==True)): # toggle on state off

TurnOnLight=False

LampBrightness.duty\_u16(0)

time.sleep(.5) #touch switch responds in 60ms after waking up

#

if ((OnSwitch.value()== 1) and (TurnOnLight==False)): # toggle off state on

TurnOnLight=True

LDR\_Counts=1.0\*(LDR\_Counts\_U16.read\_u16())

# print (LDR\_Counts) # 65355 in bright lite, 1000 counts in the dark

DimmerDutyABS=(abs(65535-LDR\_Counts))

DimmerDuty=int(DimmerDutyABS)

# print (DimmerDutyABS)

LampBrightness.duty\_u16(DimmerDuty)

time.sleep(.5)

# print(LDR\_Counts\_U16.read\_u16())

if ((OnSwitch.value()== 0) and (TurnOnLight==True)): # this dims while in normal operation

LDR\_Counts=1.0\*(LDR\_Counts\_U16.read\_u16())

DimmerDutyABS=(abs(65535-LDR\_Counts))

DimmerDuty=int(DimmerDutyABS)

# print (DimmerDutyABS)

LampBrightness.duty\_u16(DimmerDuty)

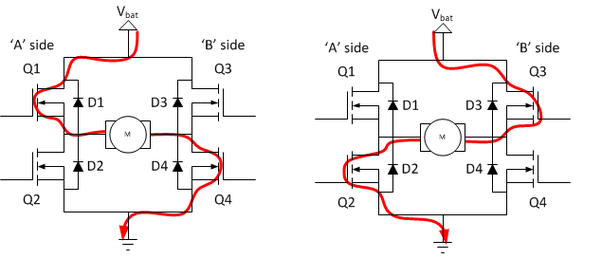
time.sleep(.3)

Drawing : H bridge forward and reverse voltage to a DC motor.

An H bridge module uses the same gating for Q1 and Q4 – forward bridge

An H bridge module uses the same gating for Q3 and Q2 – reverse bridge

Usually a FWD enable or reverse enable signal goes to the H bridge. An H bridge is interlocked so reverse and forward cannot be fired at the same time. The delay time between reverse and fwd is called deadband.



**RC\_PWM\_to\_H\_Bridge\_WORKS.py**

from machine import Pin, PWM, time\_pulse\_us

import utime

PWM1A=PWM(Pin(2)) # GP2 physical pin 4

PWM1B=PWM(Pin(3)) # GP3 physical pin 5

PWM1A.freq(1000)

PWM1B.freq(1000)

OnBoardLED=Pin(25,Pin.OUT)

Throttle=Pin(14,Pin.IN) #GPIO14 is pin#19 on Pico.It is wired to the FS-GR3E CH 2 Signal

FWD=False # initialize direction bit

REV=False # initialize direction bit

while True:

# machine.time\_pulse\_us(pin, pulse\_level, timeout\_us=1000000, /) GPIO14 is physical pin19

throttlePWM=machine.time\_pulse\_us(Throttle, 1, 21000) #

OnBoardLED.toggle() # just to let me know the program is running

#print(throttlePWM)

ThrottlePulseWidth = (throttlePWM-1500)

print(ThrottlePulseWidth)

if (ThrottlePulseWidth>15): # in milliseconds

FWD=True

REV=False

elif (ThrottlePulseWidth<-15):# in milliseconds

FWD=False

REV=True

elif (throttlePWM==-2):

FWD=False

REV=False

elif (throttlePWM==-1):

FWD=False

REV=False

else:

FWD=False

REV=False

# print (FWD)

# print (REV)

# using the L298N, to run motor A FWD IN1 = PWM GPIO2 pin4 , IN2 = LOW GPIO3 pin5

# using the L298N, to run motor A REVERSE IN1 = LOW, IN2 = PWM

PWMduty=abs(ThrottlePulseWidth)\*(65535/500) # scale the PWMduty for u16 vales

PWMduty\_Int=int(PWMduty)

# print(PWMduty)

print (PWMduty\_Int)

if FWD == True: # using a forward bit, allows to add actions based on direction

PWM1A.duty\_u16(PWMduty\_Int)

PWM1B.duty\_u16(0)

elif REV == True:

PWM1B.duty\_u16(PWMduty\_Int)

PWM1A.duty\_u16(0)

else:

PWM1B.duty\_u16(0)

PWM1A.duty\_u16(0)

# duty=PotWiperCounts.read\_u16()

# PWM1A.duty\_u16(duty)

utime.sleep(.01)

# print("Adjusted GP2 value is:", PotWiperCountsReal)