Binary Outputs

Machine & Time Library - Parallel Outputs

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #5

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

Introduction:

The Raspberry Pi Pico has 25 I/O pins

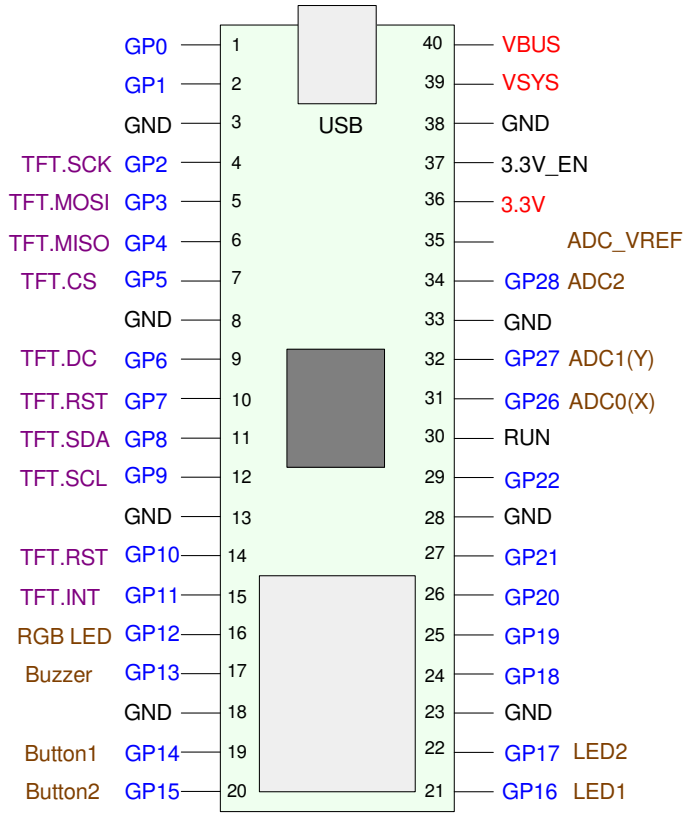
- General Purpose Input / Output pins
- Each can be binary in or out
- Many have other functions as well

These are not grouped

- No PORTA, PORTB, etc
- Each I/O is addressable separately

Logic Levels:

- 0V - 0.8V: logic level 0
- 2.0V - 3.3V: logic level 1
- 5V: smoke (don't apply 5V)



This lectures looks at

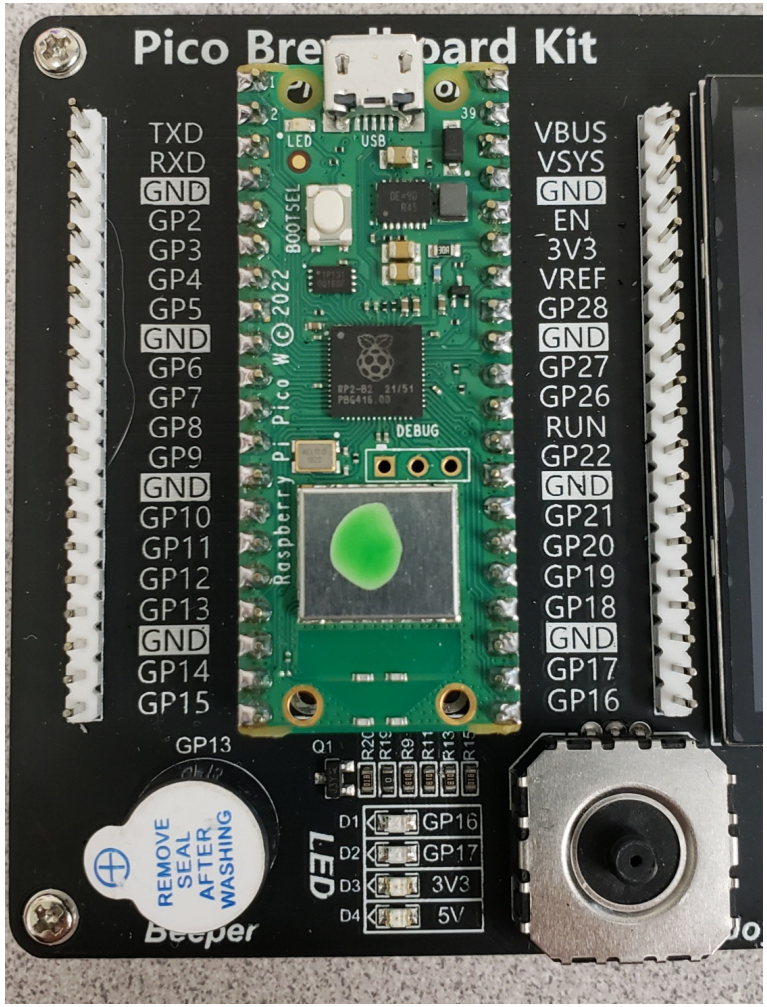
- Driving an LED
- Driving a Buzzer
  - Beep Five Times
  - Morse Code
- More Power
  - BJT (speaker, solenoid)
  - H-Bridge
- Parallel Outputs - LED Array
- Driving Multiple Outputs (PortA_Write)
- Display Routine (send to terminal pin values)
- Timing with Binary Outputs
  - Counter
  - Morse Code
- Frequency Out

Making a Light Blink

A simple program which makes the LED on pin 16 blink ten times is:

- GP13 = Beeper
- GP16 = LED D1
- GP17 = LED D2

```
1   from machine import Pin
2   from time import sleep
3
4   LED = Pin(16, Pin.OUT)
5
6   for i in range(0,10):
7       LED.toggle()
8       sleep(0.1)
9   LED.value(0)
```



How this program works

- It's a little cryptic for now
- It will make more sense shortly

- 1 & 2: These import routines used later on in the program
- 4: Sets up GPIO pin 16 to be output
- 7: Toggle the LED
- 8: Pause 0.1 second
- 9: Turn off the LED when done

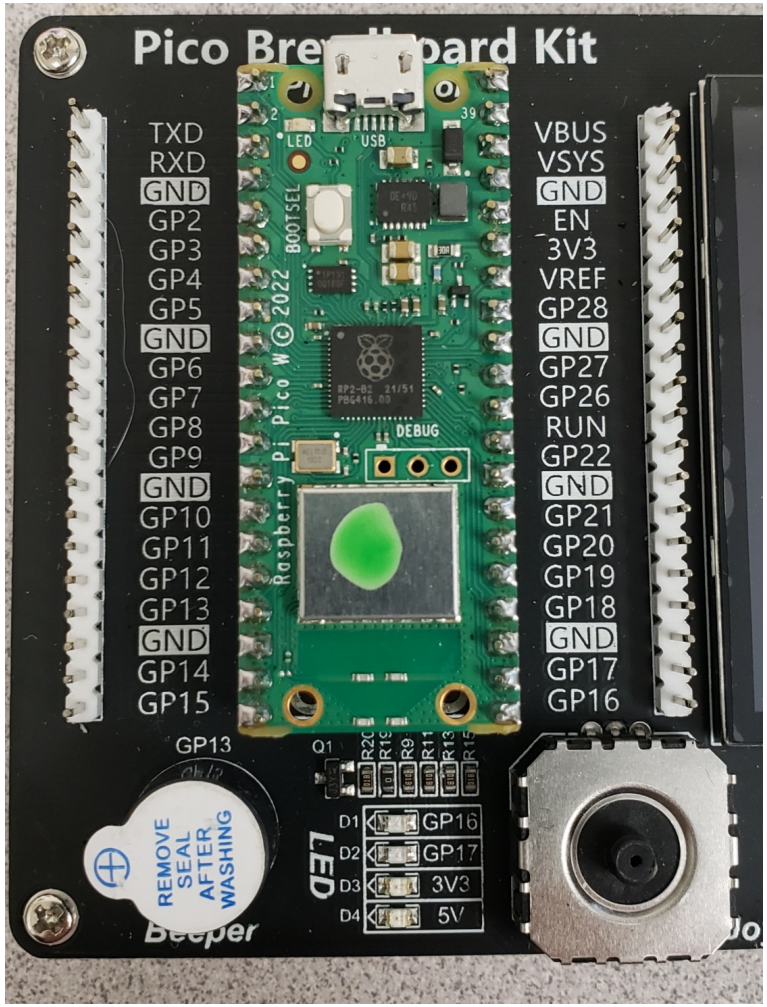
```
1  from machine import Pin
2  from time import sleep
3
4  LED = Pin(16, Pin.OUT)
5
6  for i in range(0,10):
7      LED.toggle()
8      sleep(0.1)
9  LED.value(0)
```

Beep Five Times

You can also write a program to beep five times with a simple change:

```
1   from machine import Pin
2   from time import sleep
3
4   Beeper = Pin(13, Pin.OUT)
5
6   for i in range(0,10):
7       Beeper.toggle()
8       sleep(0.1)
9   Beeper.value(0)
```

More details on how this work follows...



Background - Modules

Modules are standardized sets of subroutines

- You wrote or other people wrote
- Once you import a module, you have access to all of its subroutines

In C, you import subroutine libraries using *#incude* statements

```
// Start of a C program
#include <stdio.h>
#include <math.h>
```

In Python, you import subroutines usine *import* statements

```
#Start of a Python program
import machine
import time
import math
```

Addressing Subroutines

Slightly different from C

- The name of the module is part of the subroutine name
- This avoids conflicts
  - If two modules have two subroutines with the same name

Example: Use

- *cos()* from module *math*
- *sleep()* from module *time*

```
import math
import time

x = 2 * math.cos( 1.74 * math.pi )
time.sleep(0.1)
```

Shortcut for Subroutine Names

It can be clunky including the module name over and over

For commonly used routines, you can skip this

- Just make sure the names don't cause conflicts
- There can only be one function called *cos()*

```
from math import sin, cos, pi
from time import sleep

x = 2 * cos( 1.74 * pi )
sleep(0.1)
```

What's In a Module?

If you want to know what modules are available to use, in the shell window type:

```
>>> help('modules')
random    machine     math      time    ...
```

(a complete list modules is in the appendix).

If you want to see what's inside a given module, such as *machine* or *time*, type

```
>>> import machine
>>> dir(machine)
['PWM', 'Pin', time_pulse_us', ...]

>>> import time
>>> dir(time)
['sleep', 'sleep_ms', 'sleep_us', ...]
```

To get some help on a specific function within a module, use the help function:

```
>>> help(machine.PWM)

object <class 'PWM'> is of type type
  init -- <function>
  deinit -- <function>
  freq -- <function>
  duty_u16 -- <function>
  duty_ns -- <function>
```

A complete list of modules and functions in the appendix[1].

[1] note: The appendix is a place to put stuff which would kill the flow of your document. A useful tool in technical documents.

Binary Outputs (Software)

Starting out, let's turn on and off an LED.

GPIO pins are binary signals:

- Logic 0 = 0V
  - capable of sinking up to 12mA
- Logic 1 = 3.3V
  - capable of sourcing up to 12mA

Each GPIO pin can be set up for either

- Output (this lecture) or
- Input (next lecture).

To do this, routine *Pin* is used

- Part of the module *machine*.

```
from machine import Pin

LED1 = Pin(16,Pin.OUT)
LED2 = Pin(17,Pin.OUT)
```

Turning an LED On and Off

Commands to set and clear output pins:

`toggle()`	toggle the pin
`value(1)`	set the pin (LED on)
`value(0)`	clear the pin (LED off)
`high()`	set the pin (LED on)
`low()`	clear the pin (LED off)
`value()`	read whether the LED is on or off

```
from machine import Pin

LED1 = Pin(16,Pin.OUT)
LED2 = Pin(17,Pin.OUT)

LED1.toggle()

LED2.value(1)

LED2.value(0)

LED2.high()

LED2.low()

print(LED2.value())
```

Timing (sleep)

- To control the timing of a light turning on and off, routines from the module *time* are used
- These lock up the processor for a fixed amout of time

`time.sleep(1.234)`
- wait 1.234 seconds

`time.sleep_ms(123)`
- wait 123 milliseconds

`time.sleep_us(123)`
- wait 123 microseconds

```
from machine import Pin
from time import sleep

LED1 = Pin(16,Pin.OUT)
LED2 = Pin(17,Pin.OUT)

for i in range(0,10):
    LED1.toggle()
    sleep(1)
```

For example, the following program

- Sets up pin 16 for output,
- Turns on pin 16 for one second,
- Turns off pin 16 for one second, then
- Toggles pin 16 ten times every 100ms

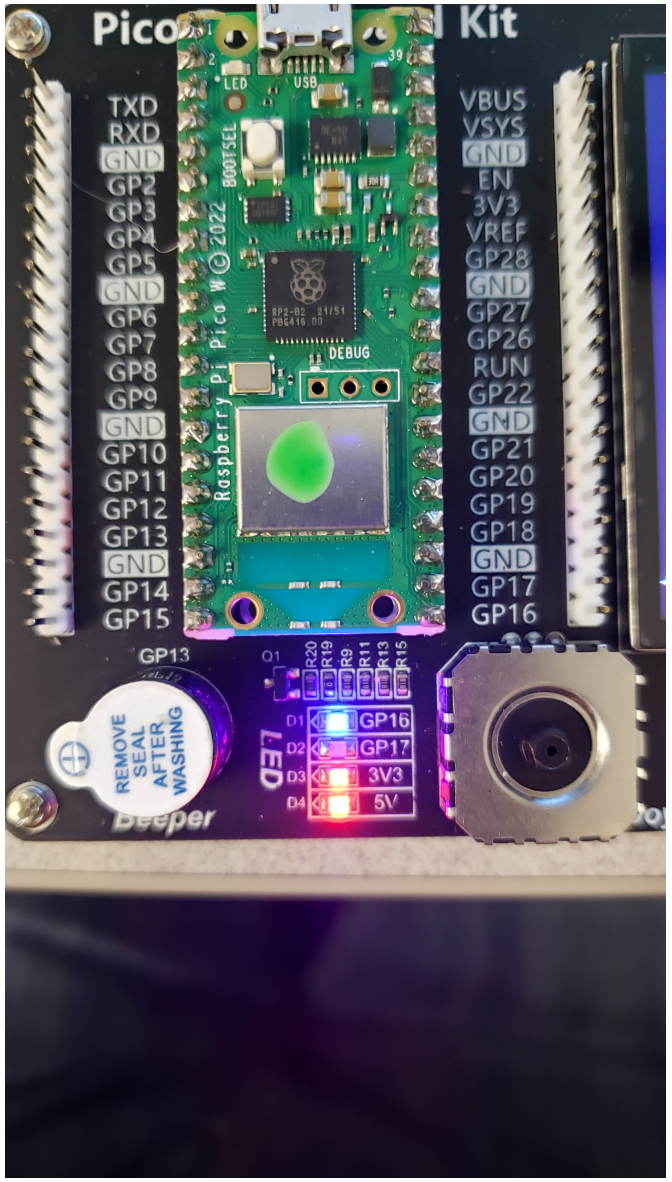
```
from machine import Pin
from time import sleep

LED = Pin(16, Pin.OUT)

print('Light On')
LED.value(1)
sleep(1)

LED.value(0)
sleep(1)

for i in range(0,10):
    LED.toggle()
    sleep(0.1)
```



Sidelight: Using Arduino Syntax

Auduino's use the syntax

`GPIO(pin, value)`

- *pin* is the pin number
- *value* is 1 or 0 for on/off.

You can mimic this function

- Create your own routine

```
from machine import Pin
from time import sleep_ms

LED = [16,17,18,19,20,21,22,26]
for i in range(0, len(LED)):
    LED[i] = Pin(LED[i],Pin.OUT)

def GPIO(X, Value):
    if((Value > 1) | (Value < 0)):
        LED[X].toggle()
    else:
        LED[X].value(Value)

# turn on output #0 (GP16)
GPIO(0,1)

# turn off output #2 (GP18)
GPIO(2,0)

# toggle output #4 (GP20)
GPIO(4,-1)
```

Binary Outputs (Hardware)

A Pi-Pico can drive more than just the LEDs on your development board.

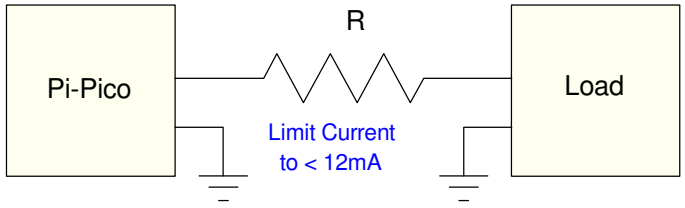
If you want to drive external devices, some simple electronic circuits work

Loads: < 3.3V and < 12mA:

The I/O pins can source/sink up to 12mA

If that's all your load needs, drive it directly

- Use a resistor to limit the current



Example: Drive an external red LED at 10mA

First, find the data sheets for the LED

Digikey	color	wavelength	Vf @ 20mA	mcd @ 20mA	price
732-5013-ND	red	628nm	2.0V	2600mcd	$0.18

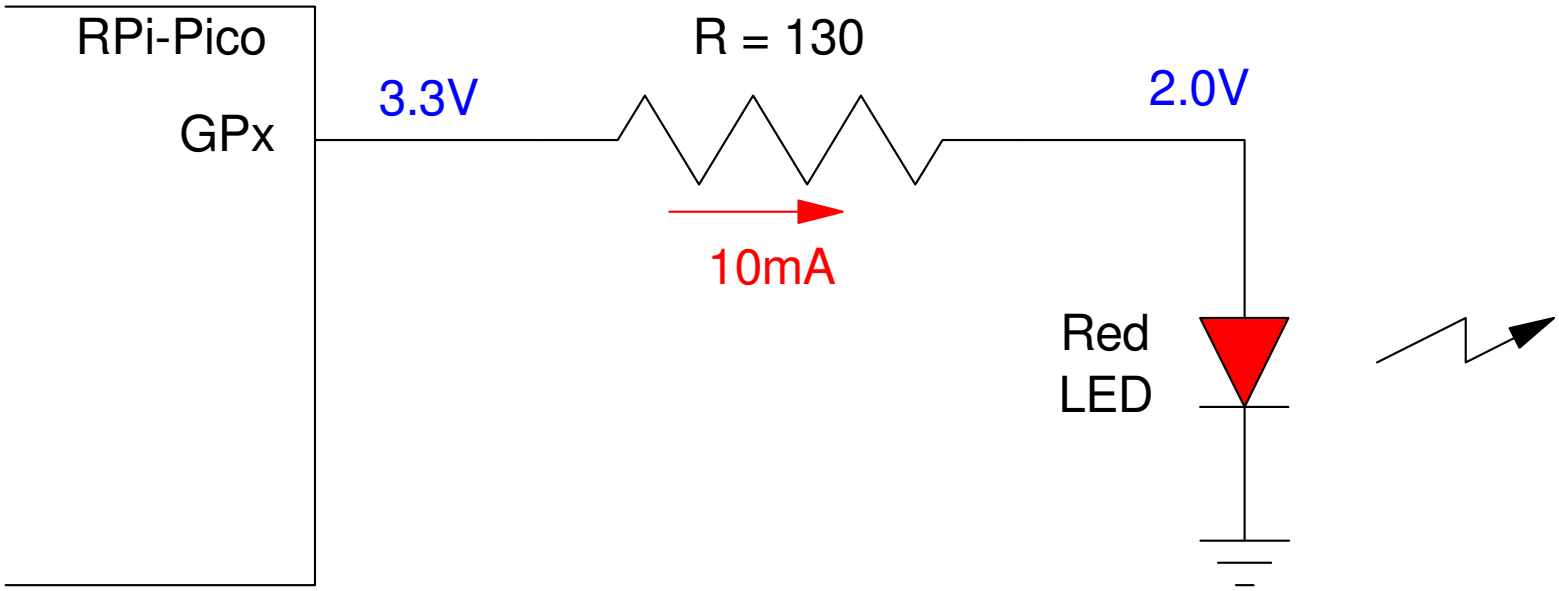
Vf tells you the voltage drop across the LED when turned on.

To limit the current to 10mA, add a resistor:

$$R = \left(\frac{3.3V - 2.0V}{10mA}\right) = 130\Omega$$

The brightness of the LED will then be proportional to the current:

$$\left(\frac{10mA}{20mA}\right) 2600mcd = 1300mcd$$



If driving a load that needs less than 3.3V and less than 12mA,
you can connect it directly to the RPi-Pico with just a resistor (to limit the current)

Loads: >3.3V or >12mA:

- Too much for a Pi-Pico
- Add a buffer (NPN transistor)

Example: Drive a 3W white LED at 750mA

Step 1: Find the data sheets:

ebay	color	Vf	Output	price
Lighthouse LEDs	warm white	3.6V @ 750mA	200lm @ 750mA	$2.06

- 3.6V is too much for a PiPico
- 750mA is too much for a PiPico

Step 2: Pick your favorite NPN transistor

- 2SC6411 NPN transistor
- hfe > (750mA / 12mA) = 62.5
- Ic(max) > 750mA
- Other NPN transistors will also work

Digikey	Vce (sat)	hfe (min)	Ic (max)	hfe
2SC6144SG	360mV	200	10A	$0.85

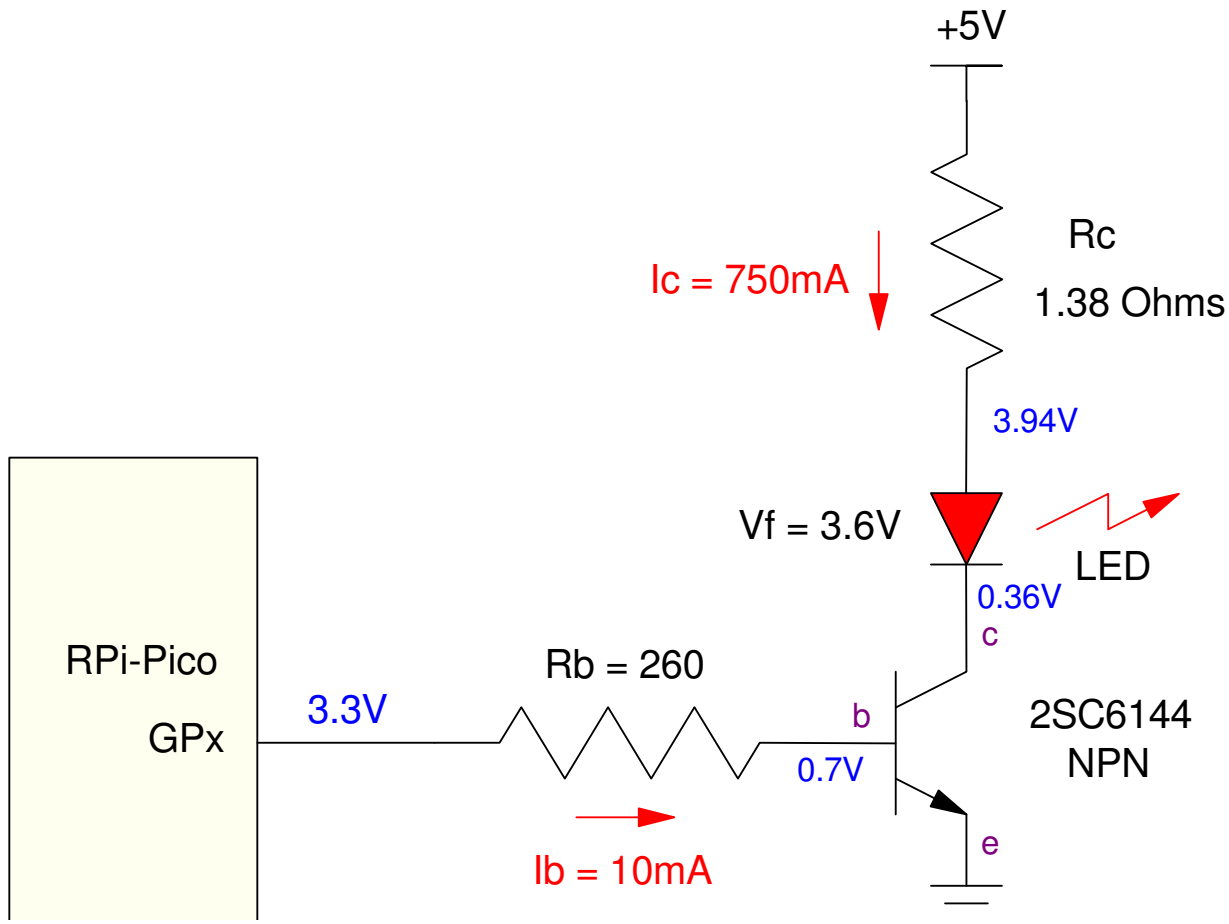
Step 3: Determine Rb and Rc: Assuming a 5V source, the calculations are:

$$R_c = \left(\frac{5V - 3.6V - 0.36V}{750mA}\right) = 1.38\Omega$$

$$I_b > \frac{I_c}{h_{fe}} = \frac{750mA}{200} = 3.75mA$$

Let Ib = 10mA

$$R_b = \left(\frac{3.3V - 0.7V}{10mA}\right) = 260\Omega$$



If your load needs more than 3.3V or more than 12mA, you can use a BJT transistor as a switch

Note that using a BJT transistor as a switch works for just about any load

- Any load with Ic < 2A
- $\max(I_c) = h_{fe} \cdot I_b$
- $= 200 \cdot 10mA = 2A$

This makes a BJT switch very versatile and very common.

With it, you can turn on and off

- LED lights
- DC motors
- Heaters
- Speakers,
- etc

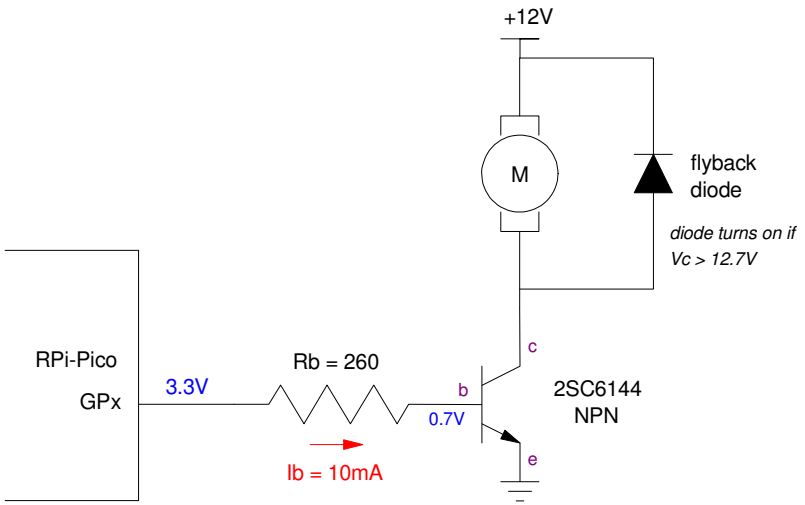
providing they need less than 2A when on.

Note on Inductive Loads:

If your load is inductive in nature:

- Solenoids
- DC motors

you need to include a flyback diode.



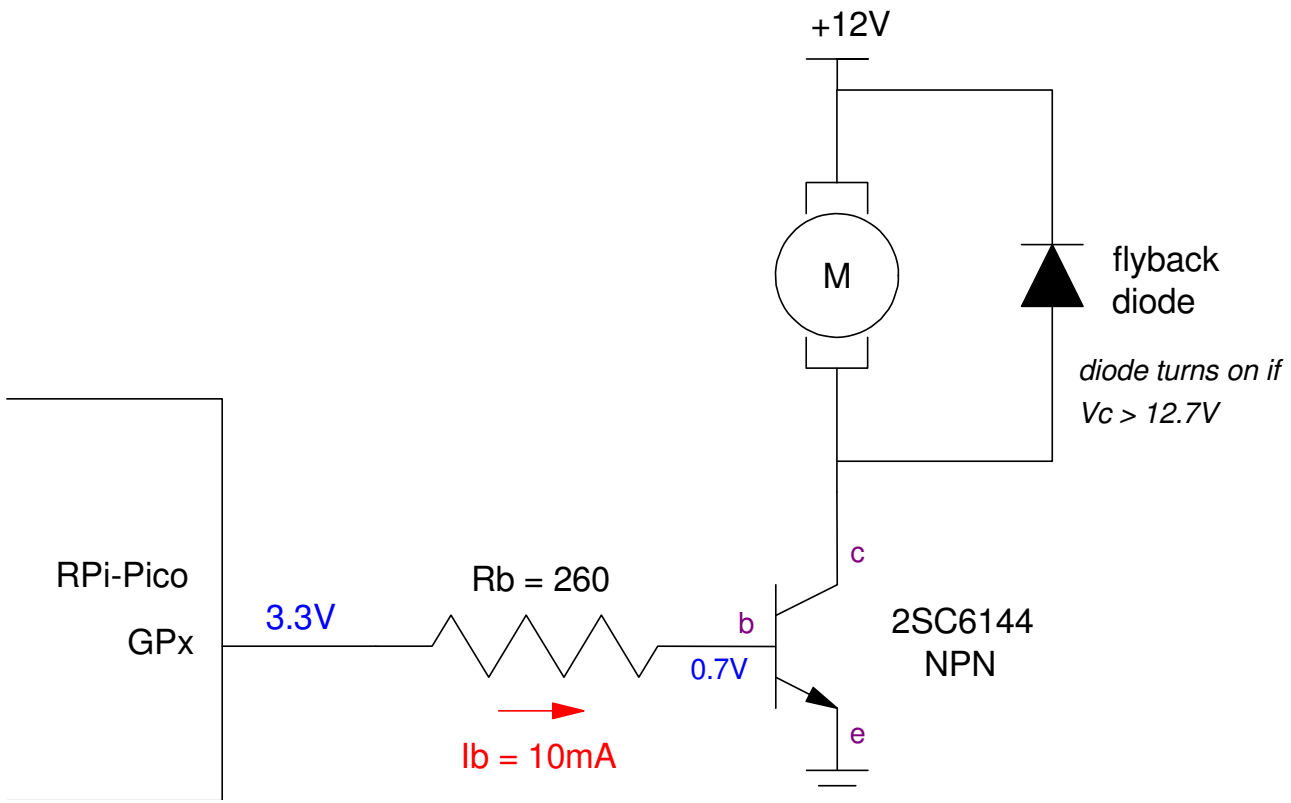
This limits the voltage at Vc to +12.7V

- For inductors, $V = L\frac{di}{dt}$
- When the transistor turns off, I suddenly goes to zero
- This causes the voltage to go to infinity, burning out your transistor.

What's happening is

- Energy is stored in the inductor as $E = \frac{1}{2}Li^2$
- When the transistor turns off, the stored energy *must* go somewhere.

To bleed off the stored energy, the inductor will raise its voltage until it finds a path to ground. With the flyback diode, this voltage is limited to 12.7V



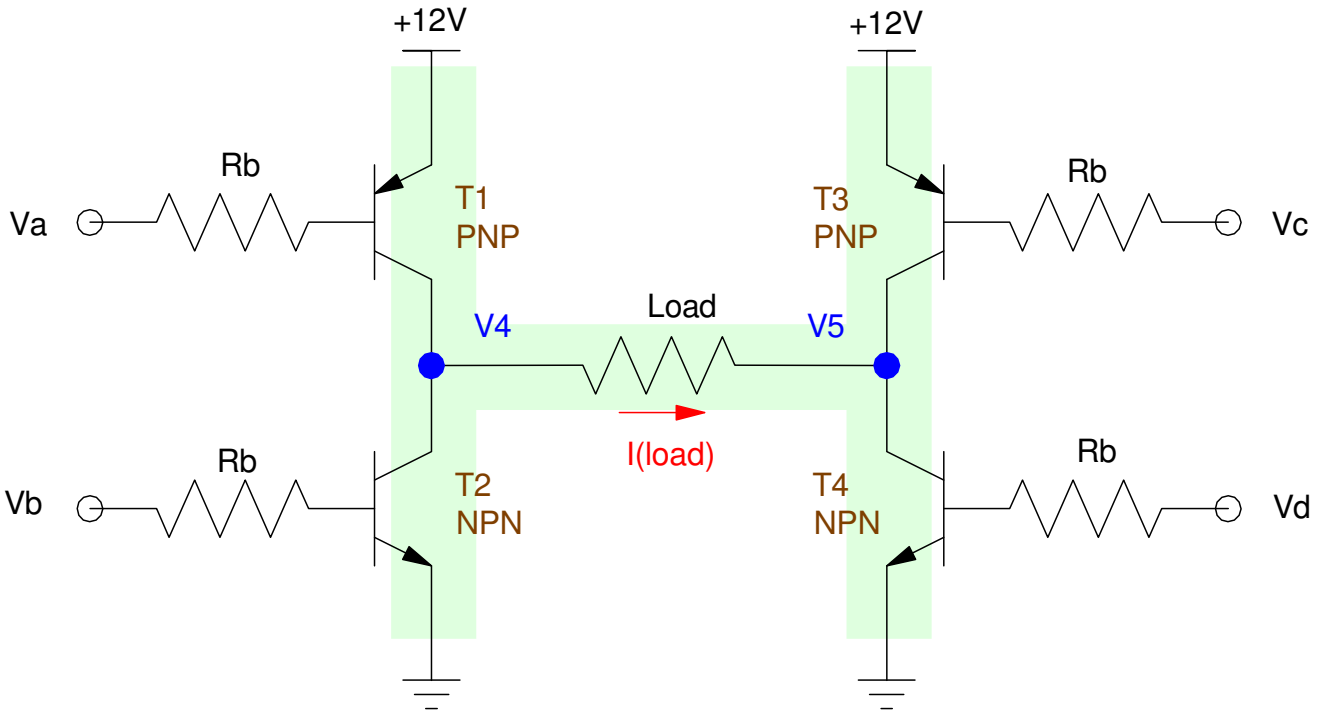
If you are turning on and off an inductive load (DC motor, solenoid), add a flyback diode to limit the voltage at Vc

Forward & Reverse: H-Bridge

If you want to

- Apply a positive and negative voltage to a load
- While using just a single power supply,

an H-bridge can be used.



H-Bridge Operating Modes:

+11.6V:

- T1 and T4 are on

-11.6V:

- T2 and T3 are on

Coast:

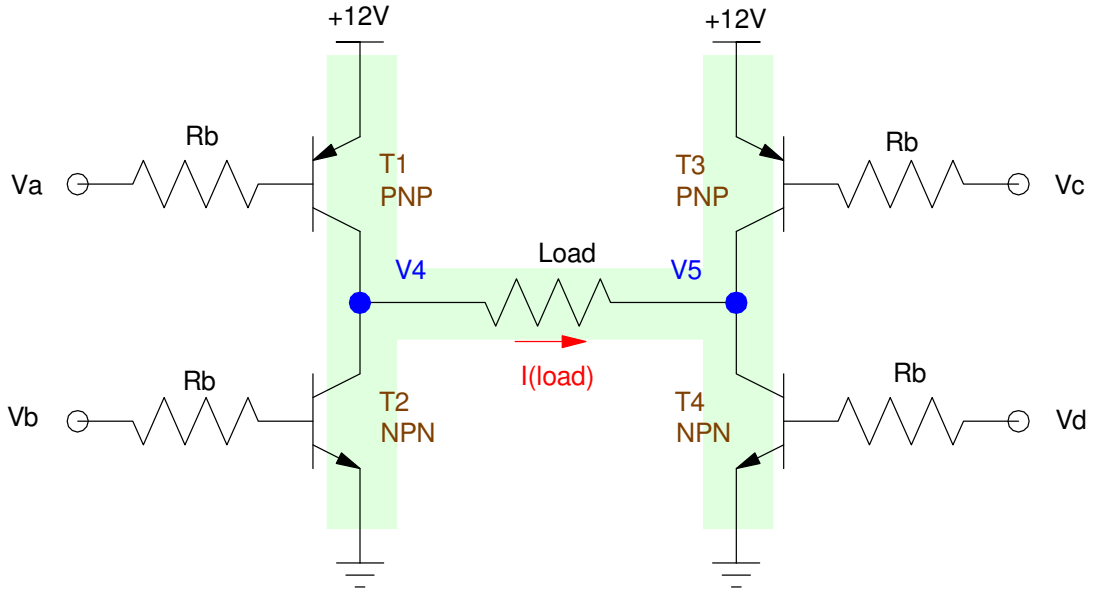
- All transistors off

Brake:

- T2 and T4 on

Smoke:

- All transisotors on
- Short power to ground (bad)



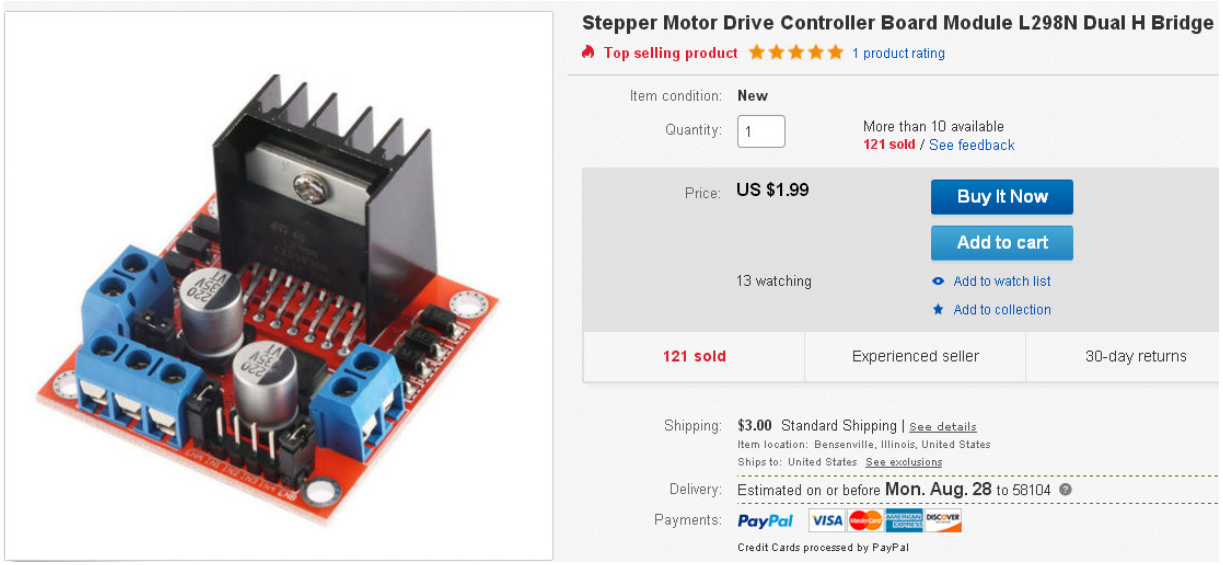
note: Each transistor has a slight voltage drop (Vce) when saturated - resulting in the load sseeing slightly less than +/- 12V when turned on

L298N H-Bridge

An inexpensive H-bridge is the L298N

- 5V to 35V operation
- Up to 3A

This actually has two H-bridges in each package



L298N Dual H-Bridge from ebay (search: Arduino H Bridge)

L298N & Pi-Pico Connections

+5V: powers the electronics on the H-bridge

0V: common ground

+5V..+35V: The power to the load

A/B: Connections to load #1

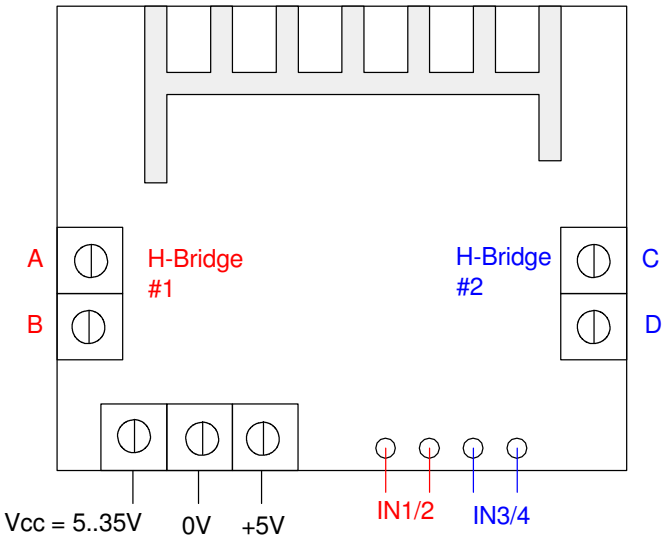
C/D: Connections to load #2

In 1/2: Pi-Pico output for Load #1

- 3.3V is OK here

In 3/4: Pi-Pico output for Load #2

- 3.3V is OK here as well



L29N & Software

In terms of software, you can control the voltage to the load using two GP output pins:

IN-1	IN-2	Vab
0V	0V	0 V
0V	3.3V	+ Vcc
3.3V	0V	- Vcc
3.3V	3.3V	0 V

IN-3	IN-4	Vcd
0V	0V	0 V
0V	3.3V	+ Vcc
3.3V	0V	- Vcc
3.3V	3.3V	0 V

Sidelight:

- The 298N is really designed to operate off of 5V
- Logic 1 is anything above 1.6V, however
- 3.3V from a Pi-Pico works as logic 1 for the inputs IN-x

Loads with a L298N H-Bridge

Once you have an H-bridge connected to your Pi-Pico, you can drive about any load that needs < 3A

- Speakers
- DC Motors
- Solenoids
- LEDs
- etc

The limitation is the output is binary:

- 100% forward
- 100% reverse
- Off

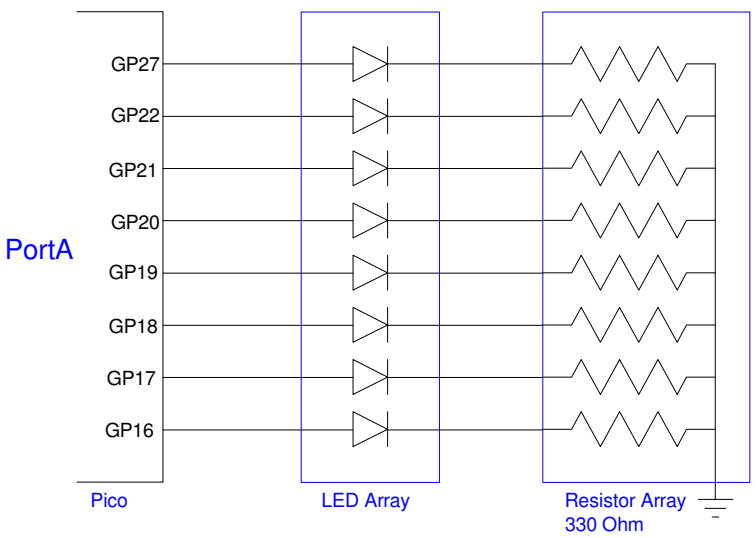
Outputs between 0% and 100% are also possible with software

- Coming later...

Binary Outputs: Ports

Many microcontrollers have ports

- Writing to a port writes to 8+ pins



Sometimes, this is useful:

- *Can you group IO pins together to create a port?*
- *Can I set up the Pi-Pico so that when I write to PortA, I'm writing to eight LEDs at once?*

The answer, of course is yes:

- You can do almost anything in software.
- Write a subroutine to mimic a port

Pi-Pico GPIO Pins

All GPIO pins are stand-alone pins

- good: any pin assignment works
- bad: you can't write to 8 bits at a time

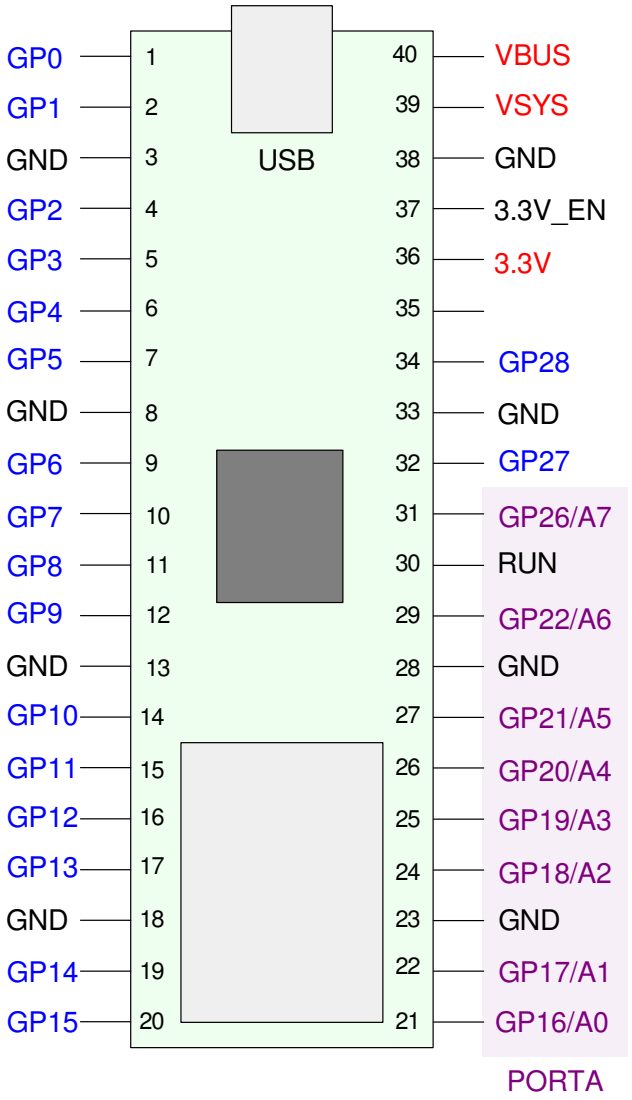
With software, you can mimic a port

In the following example

- Pins 16..26 are assigned to PORTA
- Writing to PORTA writes to all 8 bits

Good & Bad Features:

- good: you now have an 8-bit port
- bad: each bit has a slight time delay



Python Code

Assign pins to PORTA

Display each bit of PORTA

Writing to PORTA

- set/clear each bit
- one at a time

```
from machine import Pin
from time import sleep_ms

PORTA = [16,17,18,19,20,21,22,26]
for i in range(0, len(PORTA)):
    PORTA[i] = Pin(PORTA[i],Pin.OUT)

def display():
    X = ''
    n = len(PORTA)
    for i in range(0, n):
        X += str(PORTA[n-i-1].value)
    print(X)

def BinaryOut(X):
    for i in range(0, len(PORTA)):
        if(X & (1 << i)):
           PORTA[i].value(1)
        else:
           PORTA[i].value(0)

for i in range(0, 65535):
    BinaryOut(i)
    display()
    sleep_ms(50)

BinaryOut(0)
```

Fun With Binary Outputs: Blinking Light

With binary outputs, you can make a light blink

- Input a number from the keyboard
- Blink the light N times

Only engineers get excited about a light blinking

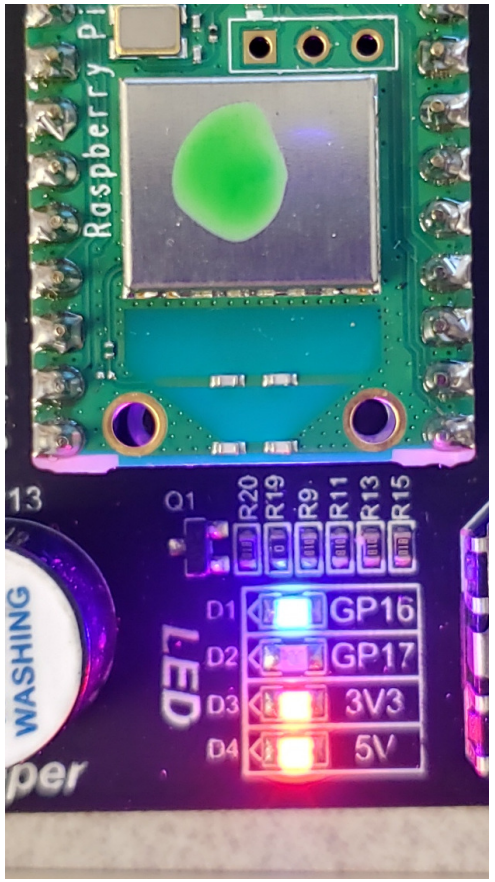
```
from machine import Pin
from time import sleep_ms

LED = Pin(16,Pin.OUT)

while(1):
    N = int(input('Number of Blinks: '))
    for i in range(0,2*N)
    LED.toggle()
    sleep_ms(100)
```

shell

```
Number of Blinks:  5
```



Fun with Binary Outputs: Night Rider

Create a 16-bit port

- GP0 to GP15

Turn on one LED

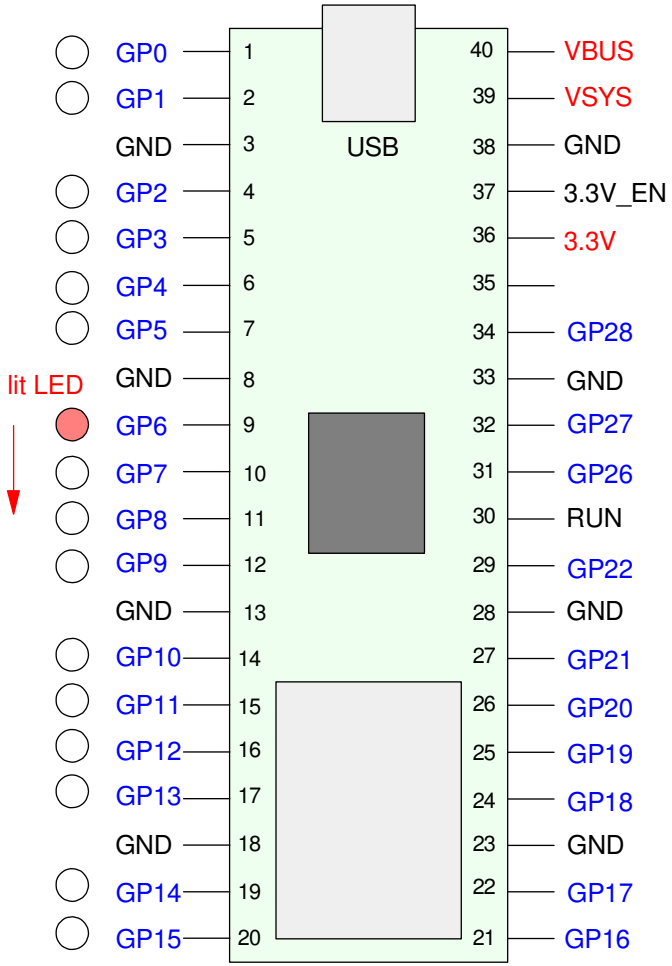
- Start with GP0

Shift the lit LED every 100ms

- light goes down

When you reach GP15, shift up

- LED bounces up and down



Python Code:

Create a 16-bit port

Make all pins output

Write to each bit
one at a time

start with PORTA = 1

shift left 16 times

then shift right 16 times

then repeat

```
from machine import Pin
from time import sleep_ms

PortA = [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]

def Init():
    for i in range(0, len(PortA)):
        PortA[i] = Pin(PortA[i], Pin.OUT)

def PortA_Write(X):
    for i in range(0, len(PortA)):
        if(X & (1<<i)):
            PortA[i].value(1)
        else:
            PortA[i].value(0)

dir = x = 1
Init()
while(1):
    PortA_Write(x)
    if(x & 0x8000):
        dir = -1
    if(x & 1):
        dir = 1
    if(dir == 1):
        x = x << 1
    else:
        x = x >> 1
    sleep_ms(100)
```

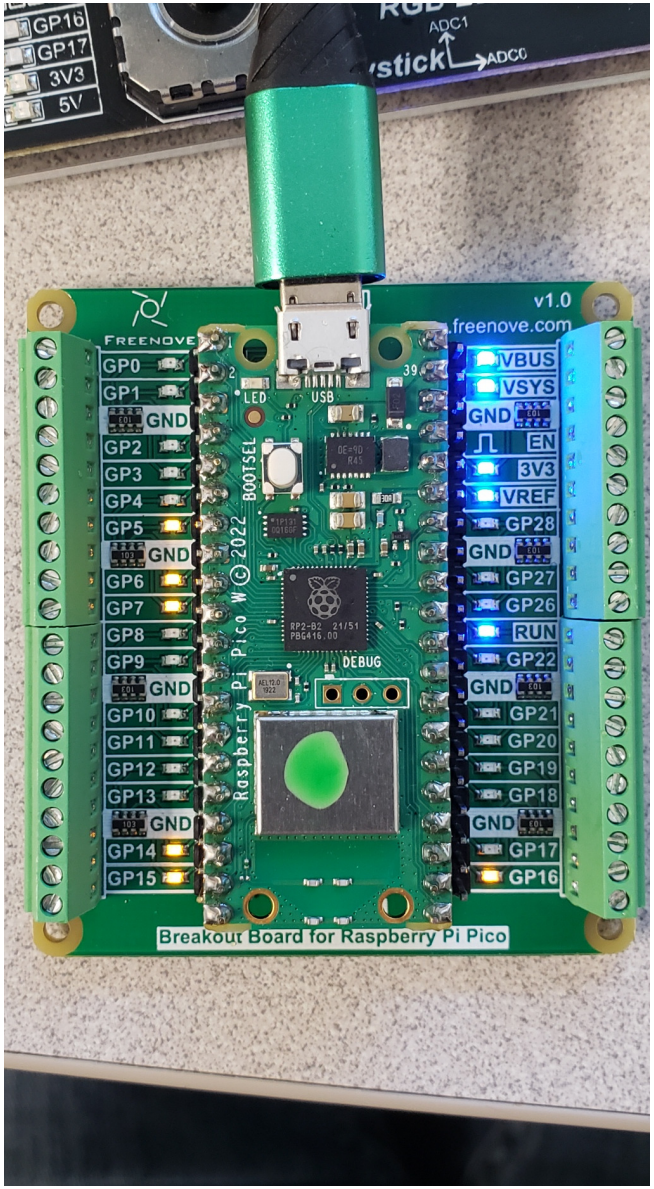
Night Rider Result:

Freenove board shown to right

- $12 from Amazon
- LEDs attached to each I/O pin

Adafruit also has breakout boards

- Maker Pi Pico Base ($9.95)
- Also includes buzzer, buttons, audio
- Doesn't include a graphics card



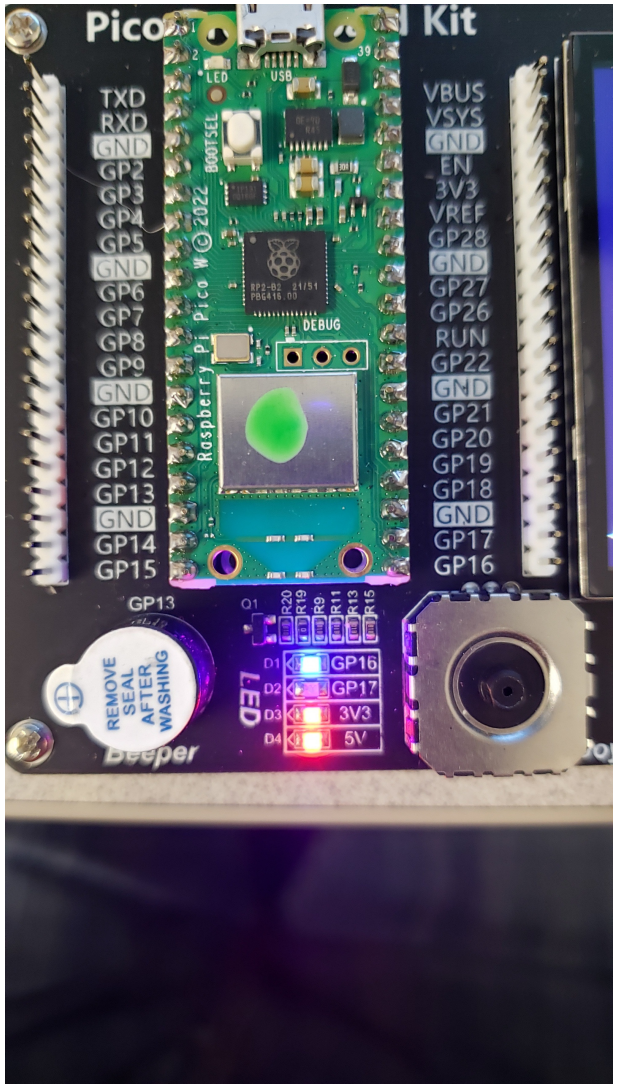
Fun with Binary Outputs: Morse Code

Play NDSU in Morse Code

- Use the beeper (GP13)
- Dit = 100ms on / 100ms off
- Dah = 300ms on / 100ms off

Program using bottom-up programming

- Define Dit & Dah
- Then define each letter
- Then combine to play NDSU



Code

Level 0:
   Dit & Dah

Level 1:
   N, D, S, U

Level 2:
   play message

```
from machine import Pin
from time import sleep_ms

Beeper = Pin(13, Pin.OUT)

def Dit():
    Beeper.value(1)
    sleep_ms(100)
    Beeper.value(0)
    sleep_ms(100)

def Dah():
    Beeper.value(1)
    sleep_ms(300)
    Beeper.value(0)
    sleep_ms(100)

def Pause():
    sleep_ms(300)
```

```
def _N():
    Dah()
    Dit()
    Pause()

def _D():
    Dah()
    Dit()
    Dit()
    Pause()

def _S():
    Dit()
    Dit()
    Dit()
    Pause()

def _U():
    Dit()
    Dit()
    Dah()
    Pause()

while(1):
    _N()
    _D()
    _S()
    _U()
    sleep_ms(1000)
```

Summary:

With the Pi-Pico, you can turn on and off devices using the general purpose pins.

- If the load needs less than 3.3V and less than 12mA, the Pi-Pico can drive that device directly, using only a resistor to limit the current,
- If the load needs more voltage or current, the Pi-Pico can drive the device using a BJT transistor as a switch or an H-bridge as a buffer.
- With software, you can also cluster GPIO pins together to create ports. These allow you to drive multiple devices with a single Pi-Pico board.

Appendix:

PWM Outputs

The following program sets up pin 16 for

- PWM output, 1000 Hz, Duty Cycle varies from 0 to 100%

note:

- duty_u16(x) sets the duty cycle (x = 0x0000) to 100% (x = 0xFFFF)
- duty_ns(x) sets the on-time as x nanosecondns02

```
from machine import Pin
from time import sleep

red = Pin(16, Pin.OUT)
red16 = PWM(Pin(16))
red16.freq(1000)
x = 0
while(1):
    red16.duty_u16(x)
    x = (x+1) & 0xFFFF
    sleep_us(10)
```

Pulse With (ns)

- Set the frequency to 50Hz (period = 20ms)
- Set the pulse width from 0.5ms (500,000ns) to 3.0ms (3,000,000ns)

Typical for servo-motor controls

```
from machine import Pin
from time import sleep

red = Pin(16, Pin.OUT)
red16 = PWM(Pin(16))
red16.freq(50)
x = 500_000
dx = 1000
while(1):
    red16.duty_ns(x)
    x += dx
    if(x > 3_000_000):
        dx = -dx
    if(x < 500_000):
        dx = abs(dx)
    sleep_us(10)
```

Standard Modules Available

```
>>> help('modules')
__main__           array              framebuf           random
_asyncio           asyncio/__init__   gc                 re
_boot              asyncio/core       hashlib
requests/__init__
_boot_fat          asyncio/event      heapq              rp2
_onewire           asyncio/funcs      io                 select
_rp2               asyncio/lock       json               socket
_thread            asyncio/stream     lwip               ssl
_webrepl           binascii           machine            struct
aioble/__init__    bluetooth          math               sys
aioble/central     builtins           micropython        time
aioble/client      cmath              mip/__init__       uasyncio
aioble/core        collections        neopixel           uctypes
aioble/device      cryptolib          network            urequests
aioble/l2cap       deflate            ntptime            webrepl
aioble/peripheral  dht                onewire            webrepl_setup
aioble/security    ds18x20            os                 websocket
aioble/server      errno              platform
Plus any modules on the filesystem
```

Functions within machine

```
>>> import machine
>>> dir(machine)
['__class__', '__name__', 'ADC', 'I2C', 'I2S', 'PWM', 'PWRON_RESET',
'Pin', 'RTC', 'SPI', 'Signal', 'SoftI2C', 'SoftSPI', 'Timer', 'UART',
'WDT', 'WDT_RESET', '__dict__', 'bitstream', 'bootloader',
'deepsleep', 'dht_readinto', 'disable_irq', 'enable_irq', 'freq',
'idle', 'lightsleep', 'mem16', 'mem32', 'mem8', 'reset',
'reset_cause', 'soft_reset', 'time_pulse_us', 'unique_id']
```

Functions within time

```
>>> 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']
```

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/