Review for Test #1

ECE 476 Advanced Embedded Systems

Jake Glower

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

Format for Test #1

Five questions

- 1-2 Hardware
- Rest Python programming

Available in-person or on BlackBoard

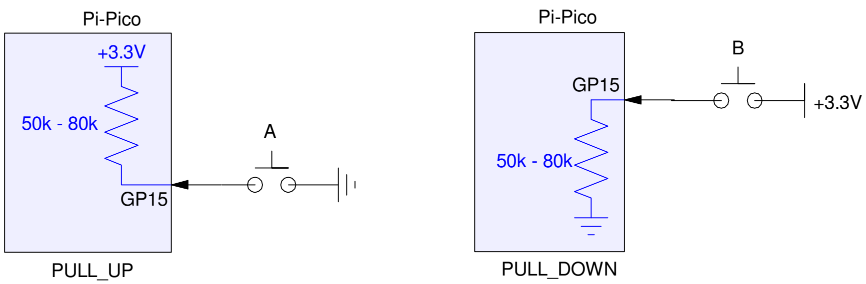
- In-Person
  - 50 minutes
  - Work problems in any order
  - Able to go back to probelms
- BlackBoard
  - 100 minutes
  - Random order with no backtracking
  - Must submit answers to first problem to move on to the next
  - Extra time due to no-backtracking, having to download, scan, upload problems

Hardware: Binary Inputs

- 0V = logic 0
- 3.3V = logic 1

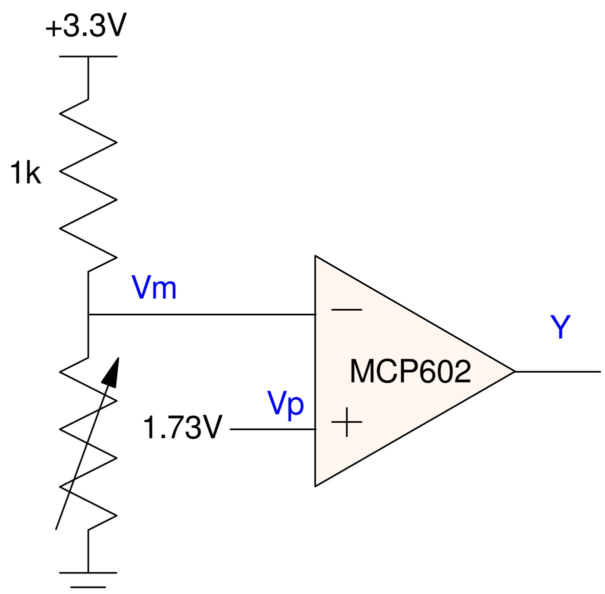
Push-Buttons:

- Pull-up or pull-down resistor



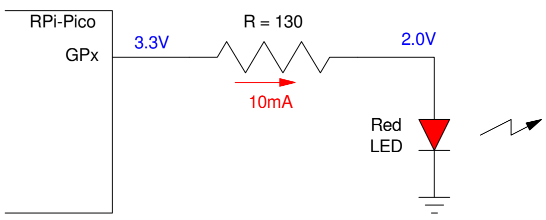
Voltages:

- Convert to 0V / 3.3V with a comparitor
  - Also works with resistors
  - Also works with thermistors
  - Also works with CdS light sensors

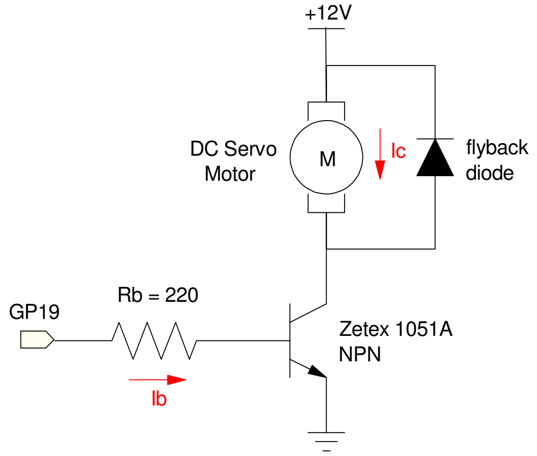


Hardware: Binary Outputs

- logic 0 = 0V
- logic 1 = 3.3V
- Capable of up to 12.5mA

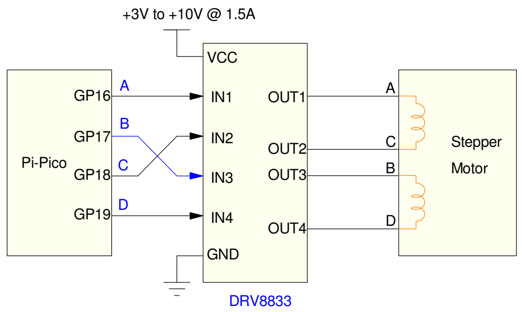


If load is less than 3.3V / 12.5mA, use a resistor



If load is more,

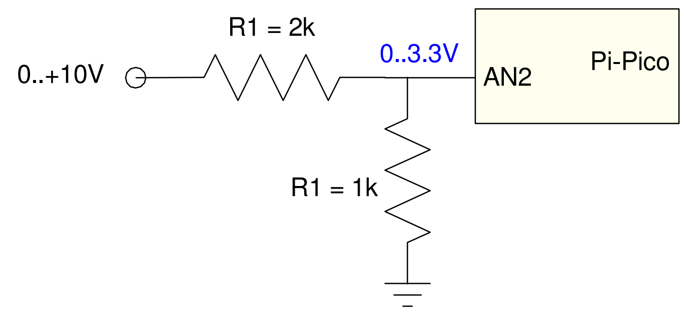
- Use a transistor (on / off)
- Use an H-bridge (- / off / +)



Hardware: Analog Inputs

On-board A/D reads 0V to 3.3V

- 12-bit A/D
- 0x0000 to 0xFFFF
- Part of *machine* library



Resistor circuits can convert wider ranges

- Voltage divider reduces max voltage to 3.3V
- Three resistors convert range to 0-3.3V
  - Weighted average

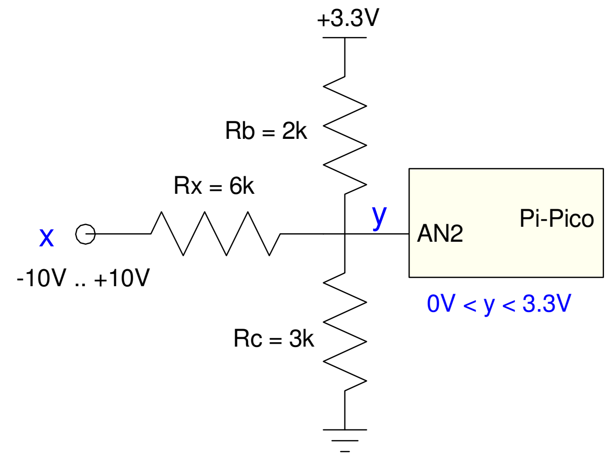
$$y = ax + b \cdot 3.3V + c \cdot 0V$$

$$a + b + c = 1$$

$$R_a = R_0/a$$

$$R_b = R_0/b$$

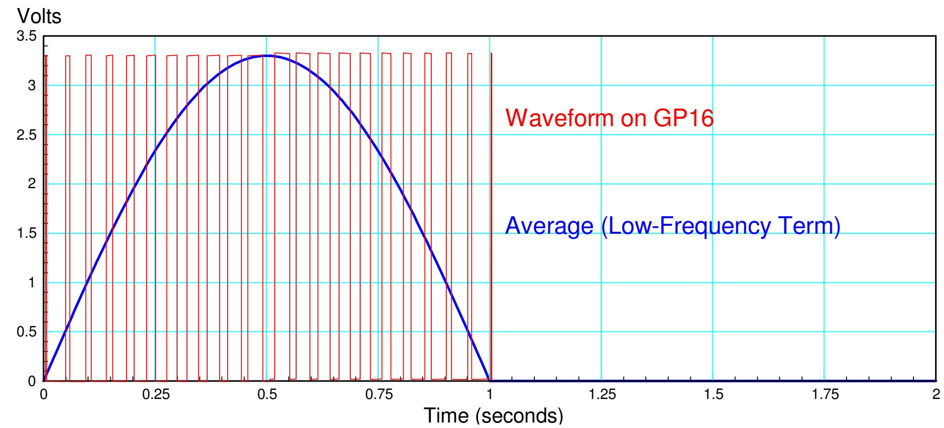
$$R_c = R_0/c$$



Hardware: Analog Outputs

Option 1: Use PWM

- 0% to 100% output
- BTJ:
  - positive output
- H-bridge
  - positive and negative output
- PWM part of *machine* library



Option 2: Use D/A and analog

- Topic for ECE 320 Electronics
- Push-Pull amplifier
- Instrumentation Amplifier
- Not on test #1 in 476/676

Software: Python Programming

Programs like Matlab

Works with complex numbers

- Add, subtract, multiply, divide
- and, or, xor, not

Shell Window

```
>>> j = (-1) ** 0.5
>>> Z3 = - j*70
>>> Z2 = 1 / ( 1/(j*60) + 1/(40 + Z3))
>>> Z1 = 1 / ( 1/50 + 1 / (j*30 + Z2))
>>> Z0 = 20 + Z1
>>> print('Zab = ',Z0)

Zab =  (58.96067+9.111071j)
```

Software: Python Programming

Inlcudes loops

- if - elif - else
- for loops
- while loops

Indentation is important

- Signifies contents of loops
- Loops end when indentation ends

```
for i in range(0,6):
    d1 = i
    for j in range(0,6):
        d2 = j
        y = d1 + d2


t = 0
dt = 0.01
while(t < 5):
    y = sin(t)
    t += dt


if(x < 3):
    y = 2*x + 4
elif(x < 5):
    y = 3 - 2*x
else:
    y = 0
```

Software: Subroutines

Subroutines declared with *def* statement

- Can pass zero, one, many terms
- Can return zero, one, many terms

Subroutines can be reused

- Call multiple times if needed


```
def Operate(A, B):
    C0 = A + B
    C1 = A - B
    C2 = A * B
    C3 = A / B
    return([C0, C1, C2, C3])


X = Operate(4,6)
print(X)
```

shell

```
>>>
[10, -2, 24, 0.666667]

>>> C = Operate(8,7)
>>> print(C)
[15, 1, 56, 1.4142857]
```

Software: Binary I/O

Pin function

- Part of *machine* library

Inputs can be

- floating
- pulled up
- pulled down

Outputs are just

- 0V logic 0
- 3.3V logic 1

```
# Binary Input
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)
```

```
#Binary Output
from machine import Pin
from time import sleep

LED = Pin(16, Pin.OUT)

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

Software: Binary Inputs

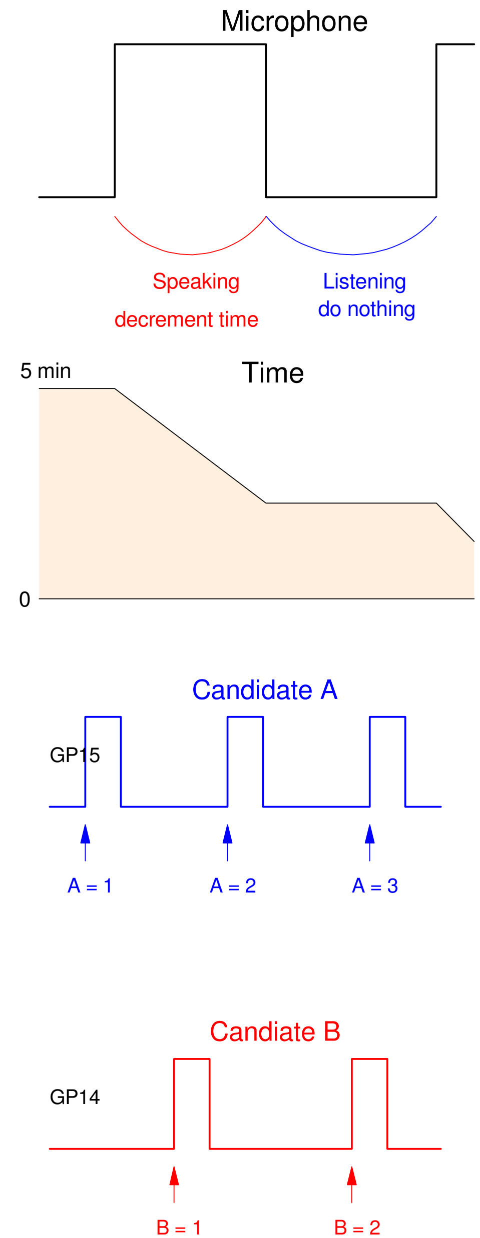
Programs can respond to levels

- 1 or 0
- Level sensitive prorams

Programs can respond to edges

- Rising edge
- Falling edge

Several techniques to detect edges



Software: Analog Inputs

12-Bit A/D on Pico

- Part of *machine* library

Range = 0V to 3.3V

- 0V reads 0x0000
- 3.3V reads 0xFFFF

```
from machine import ADC
from time import sleep_ms

a2d0 = ADC(0)
a2d1 = ADC(1)

while(1):
    x = a2d0.read_u16()
    y = a2d1.read_u16()

    print(x, y)
    sleep_ms(200)
```

Software: Analog Outputs

- Several methods possible
- PWM is probably the easiest

PWM

- Part of *machine* library
- Able to set the frequency
- Able to set the duty cycle
- Able to set the pulse width in ns

```
from machine import Pin, PWM

Aout = Pin(16, Pin.OUT)
Aout = PWM(Pin(16))
Aout.freq(1000)
Aout.duty_u16(6553)
while(1):
    pass
```

Software: Measuring Time

- Part of *time* library

```
ticks_ms      time since power up in ms
ticks_us      time since power up in us
sleep(1.23)   pause 1.23 seconds
sleep_ms(10)  pause 10ms
sleep_us(10)  pause 10us

ticks_us      time since power up in us
```

```
from time import ticks_us, sleep

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

shell

```
1000004
```

Software: Measuring Pulse Width

- Part of *machine* library

Measure the width of a high-pulse

- 1

Measure the width of a low-pulse

- 0

```
from machine import Pin, time_pulse_us

Button = Pin(17, Pin.IN, Pin.PULL_UP)
while(1):
    low  = time_pulse_us(17, 0, 500_000)
    high = time_pulse_us(17, 1, 500_000)
    print(low, high)
```

shell

```
51494  21223
48585  23313
57623  21313
55358  22313
60112  12831
39496  18231
```

Software: SPI Communications

SPI - 4 wires

- CS
- CLK
- MOSI (TX)
- MISO (RX)

Can control pins via software

- bit-banging

Can control pins via hardware

- SPI function in *machine*