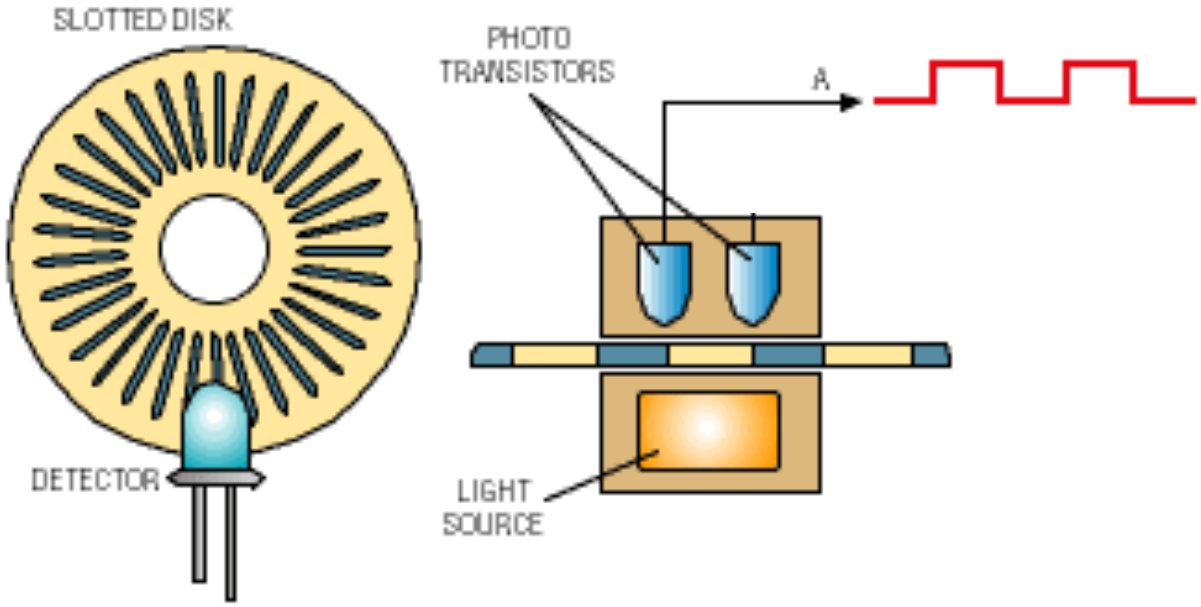
# Electronics for IoT

# Rotary Encoder

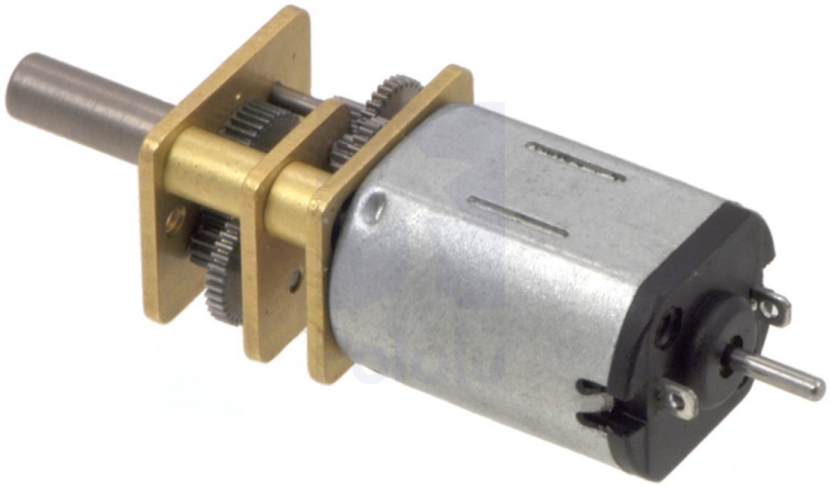
Bernhard E. Boser
University of California, Berkeley
boser@eecs.berkeley.edu

# Rotary Encoder

# Optical Encoder

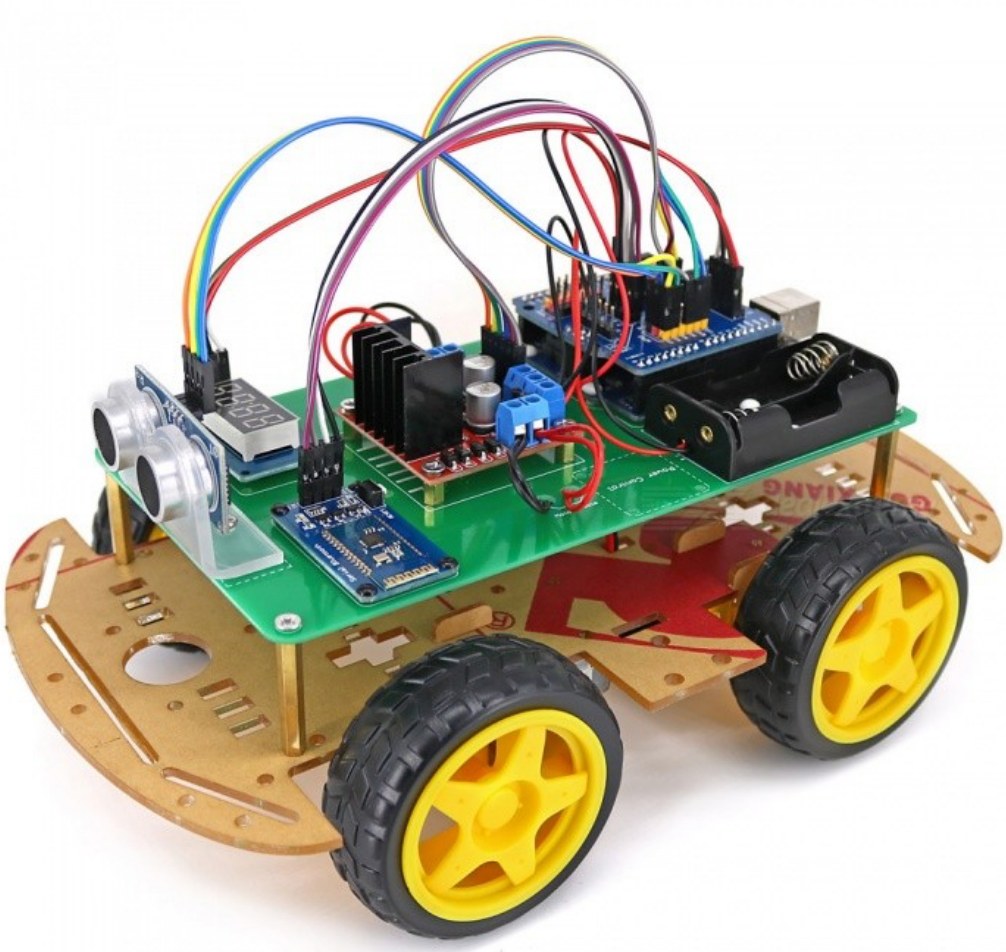


# Counts per Revolution (CPR)

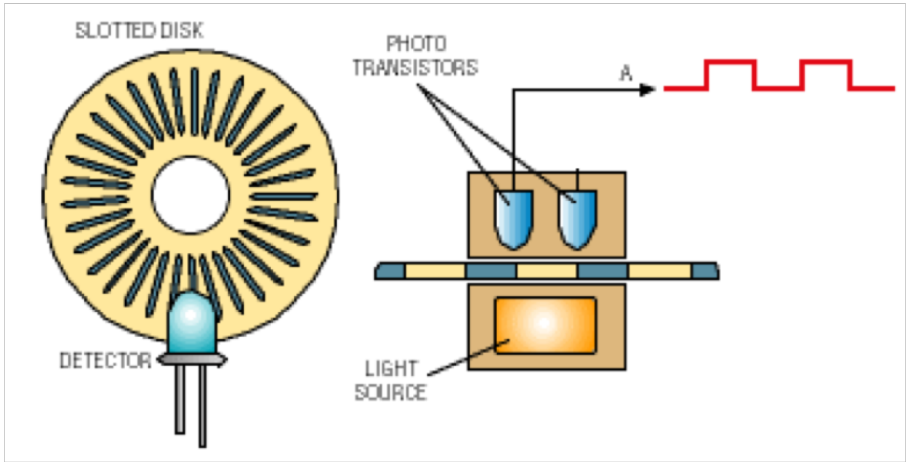


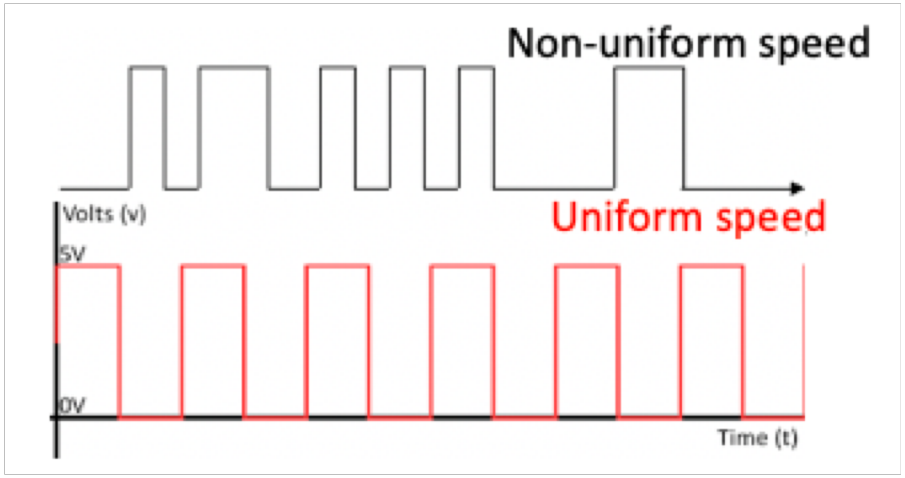
www.pololu.com

# Example: Distance Traveled



# Encoder Readout



# Readout: Polling versus Interrupts

- Mail example:
  - Polling: go to check your mailbox every hour just incase
  - Interrupt: get alert when mail arrives

- Interrupt method is often preferable
  - Can do other stuff instead of constantly checking mailbox

- Interrupt, e.g.
  - Get 0 → 1 (or 0 → 1 or either) transition at some pin (e.g. A7)
  - Call “interrupt handler”

# MicroPython Interrupt Syntax

Digital inputs can be configured to call a Python function whenever the value changes.

```
from machine import Pin
p = Pin(id, mode=Pin.IN, ...)
p.irq(handler, trigger=< Pin.IRQ_FALLING | Pin.IRQ_RISING >)
```

`trigger` may be either `Pin.IRQ_FALLING`, `Pin.IRQ_RISING` or `Pin.IRQ_FALLING | Pin.IRQ_RISING` causing the handler to be called when the input changes from `1 to 0`, `0 to 1`, or in either direction.

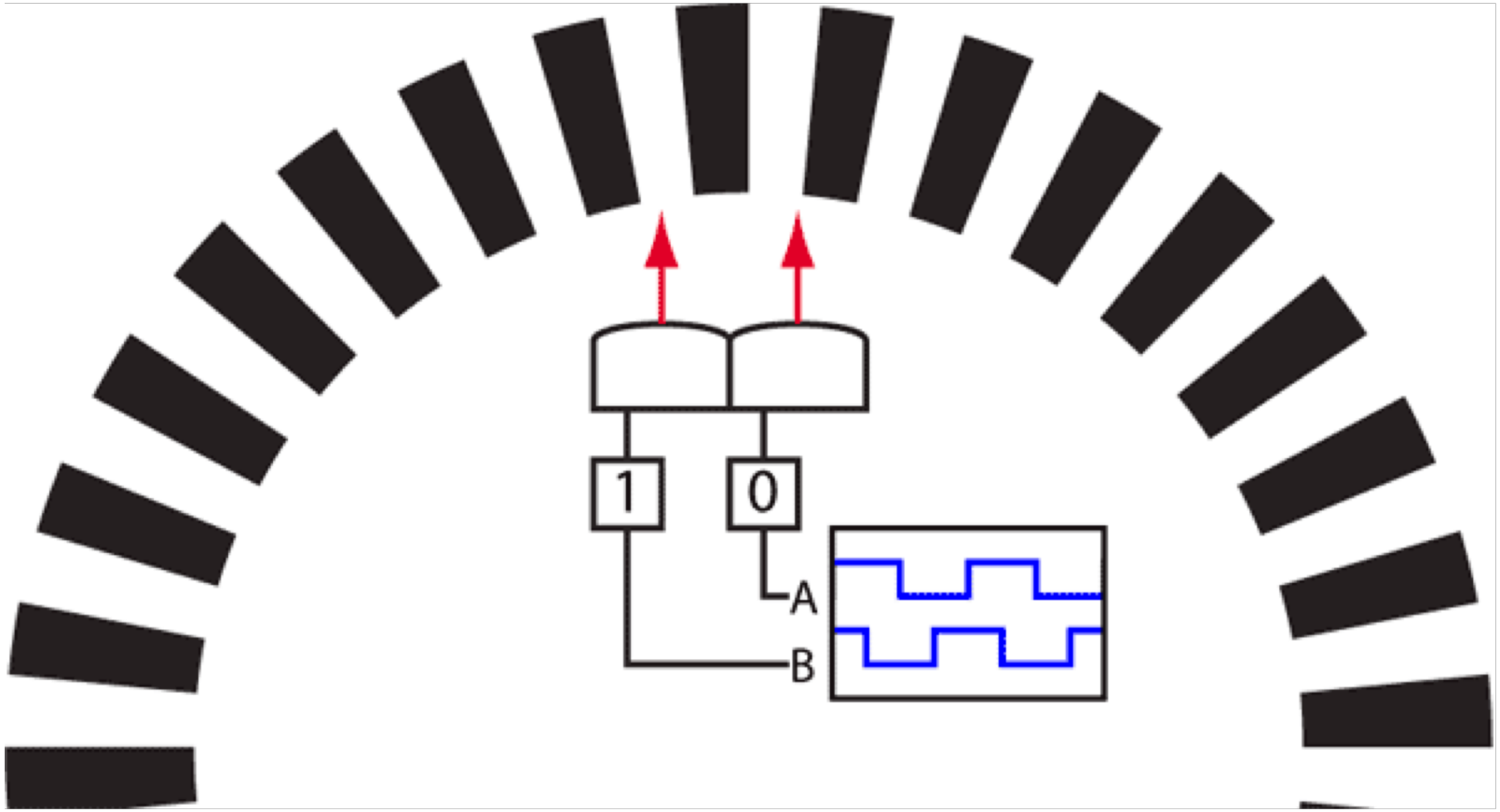
`handler` is a Python function with one argument (the `pin` that caused the interrrupt). E.g.

```
def irq_handler(pin):
    pass
```

Code in interrupt handlers must be short and not allocate memory (e.g. no floating point arithmetic, print statements, or manipulating lists). If any of these features are required or for longer computations, use the `schedule` function.

# Encoder Limitations

# Quadrature Encoder Operation

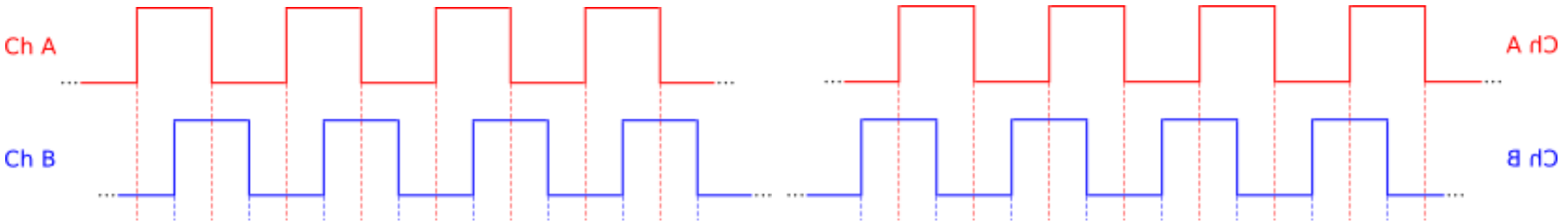



http://www.creative-robotics.com/sites/default/files/tutorials/QuadratureAnimation.gif

# Quadrature Encoder Output

## Forward

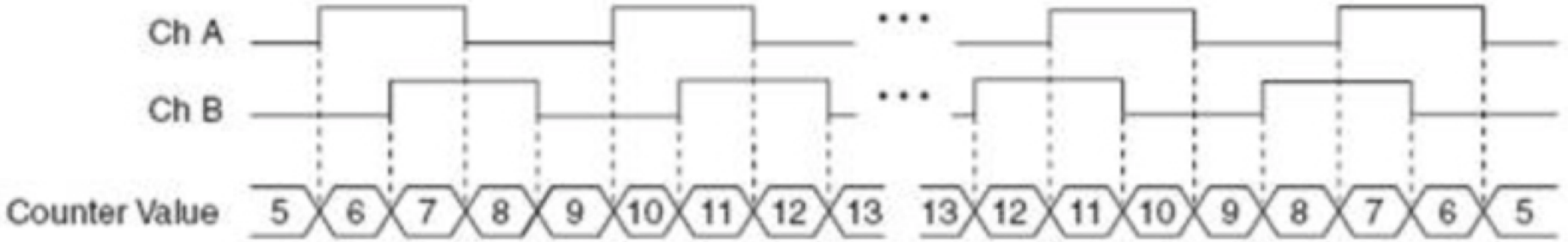
## Reverse



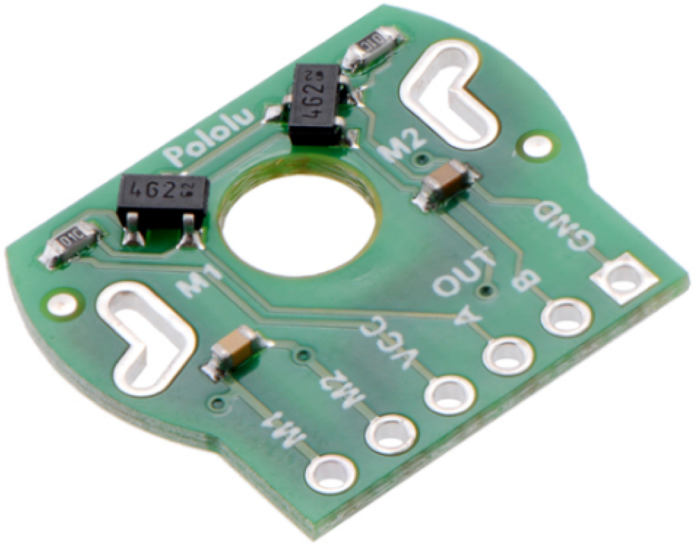
# Quadrature Decoder Output

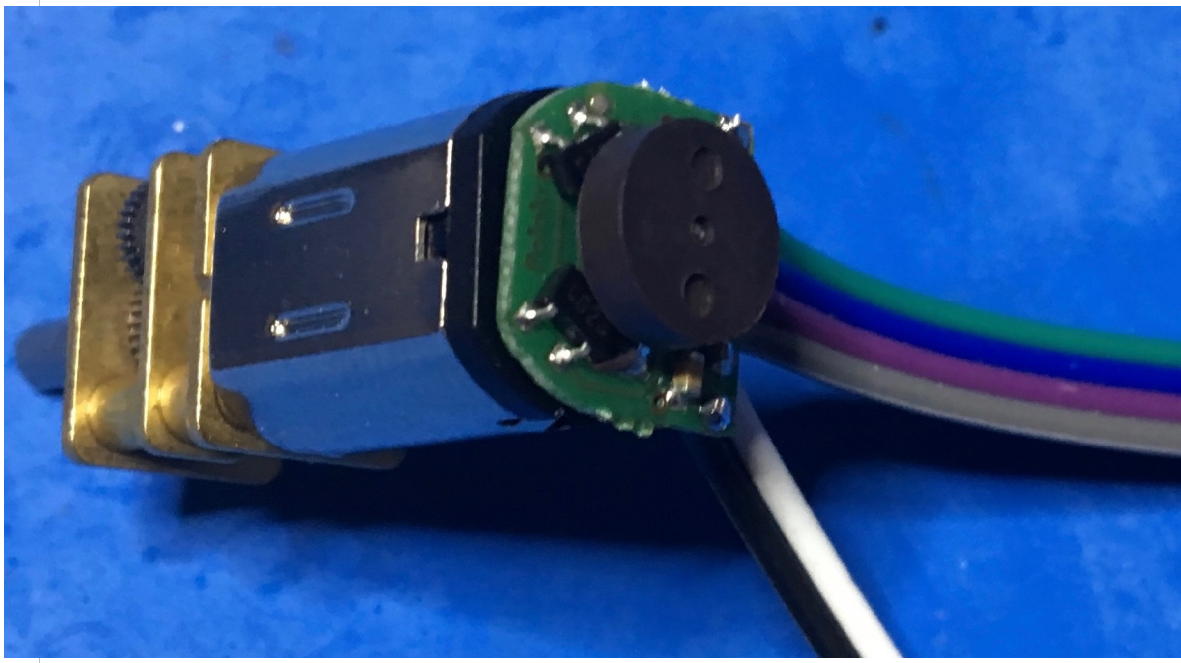
## Forward

## Reverse



# Magnetic Quadrature Encoder



# ESP32 Hardware “Decoder”

```
from machine import Pin
from machine import DEC
p1 = Pin(id1, mode=Pin.IN, ...)
p2 = Pin(id2, mode=Pin.IN, ...)
dec = DEC(<unit>, p1, p2)
```

```
dec.count()                 # returns the current count
dec.count_and_clear()       # returns the current count and resets the counter to 0
dec.clear()                 # sets the counter value to 0
dec.pause()                 # pauses counting
dec.resume()                # resumes counting
```