# Embedded Systems Programming (mostly in C)

# Outline

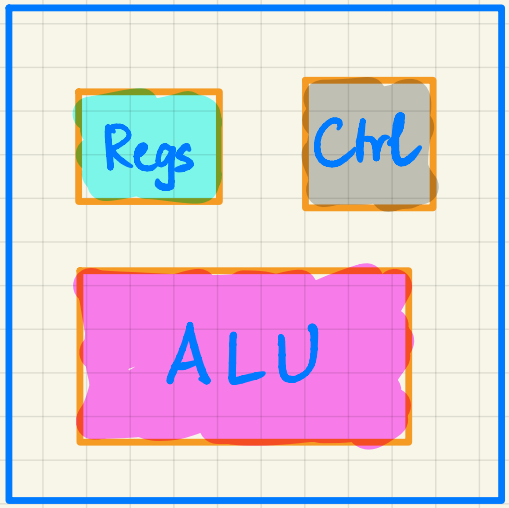
- Three ways to make an LED blink
- General Purpose Input / Output
- Communication with UARTs
- A word about Datasheets

Wokwi

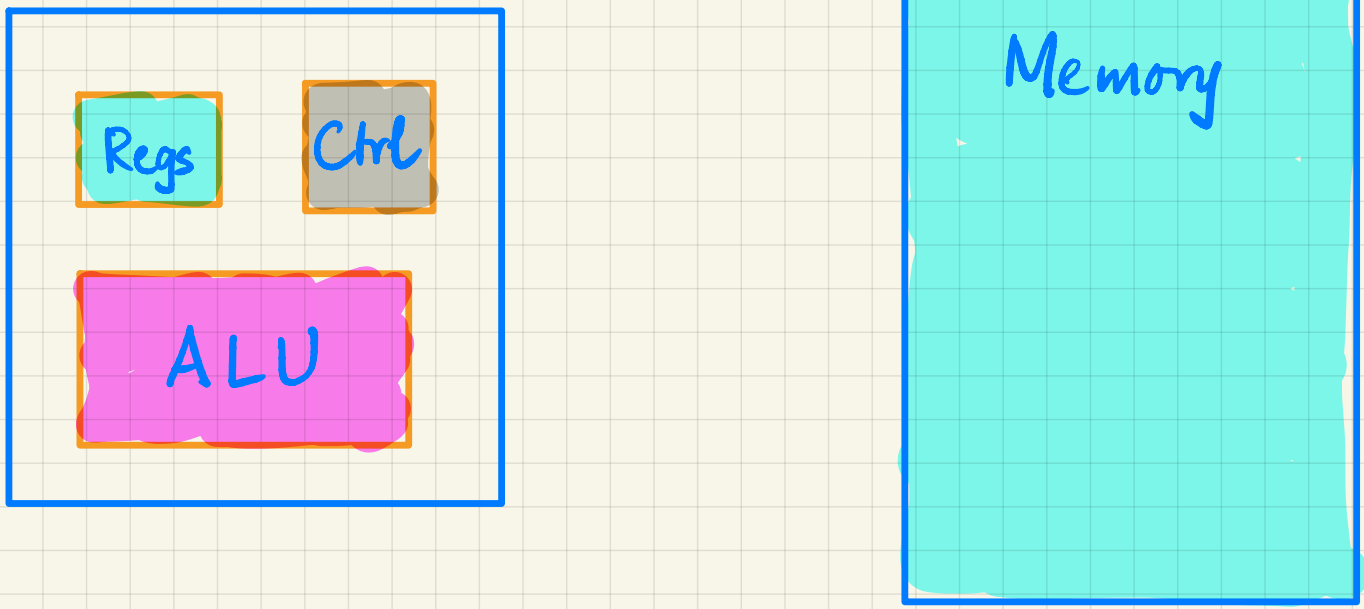
https://wokwi.com

# General Purpose Input/Output

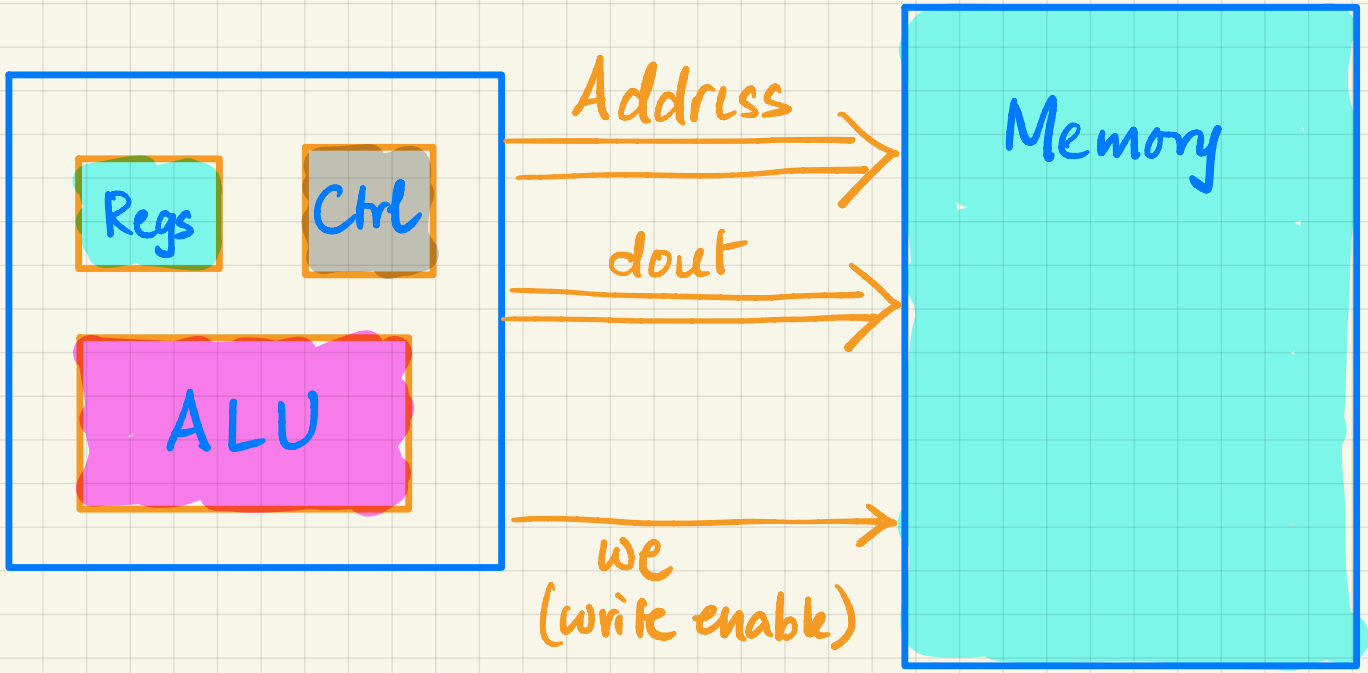
# CPU meets World



# CPU meets World



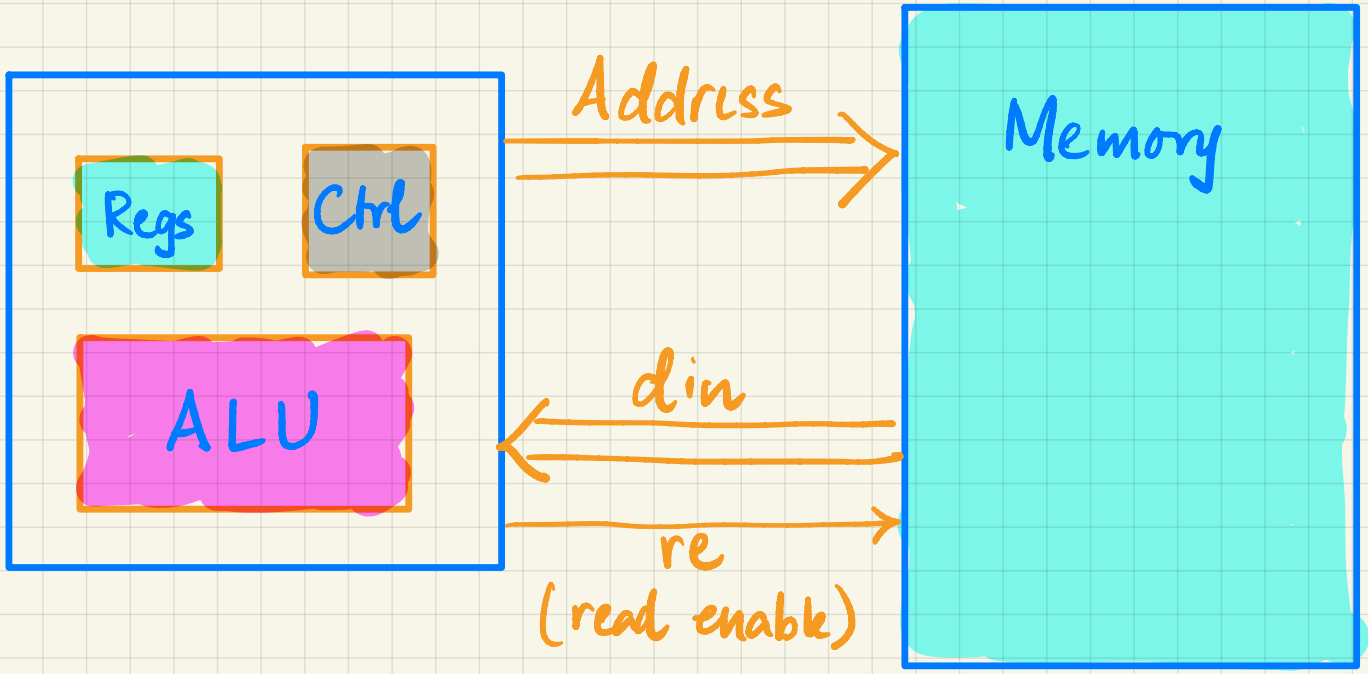
# CPU meets World




Store data in external memory

- Give an address
- Give data
- Set control signal to enable write

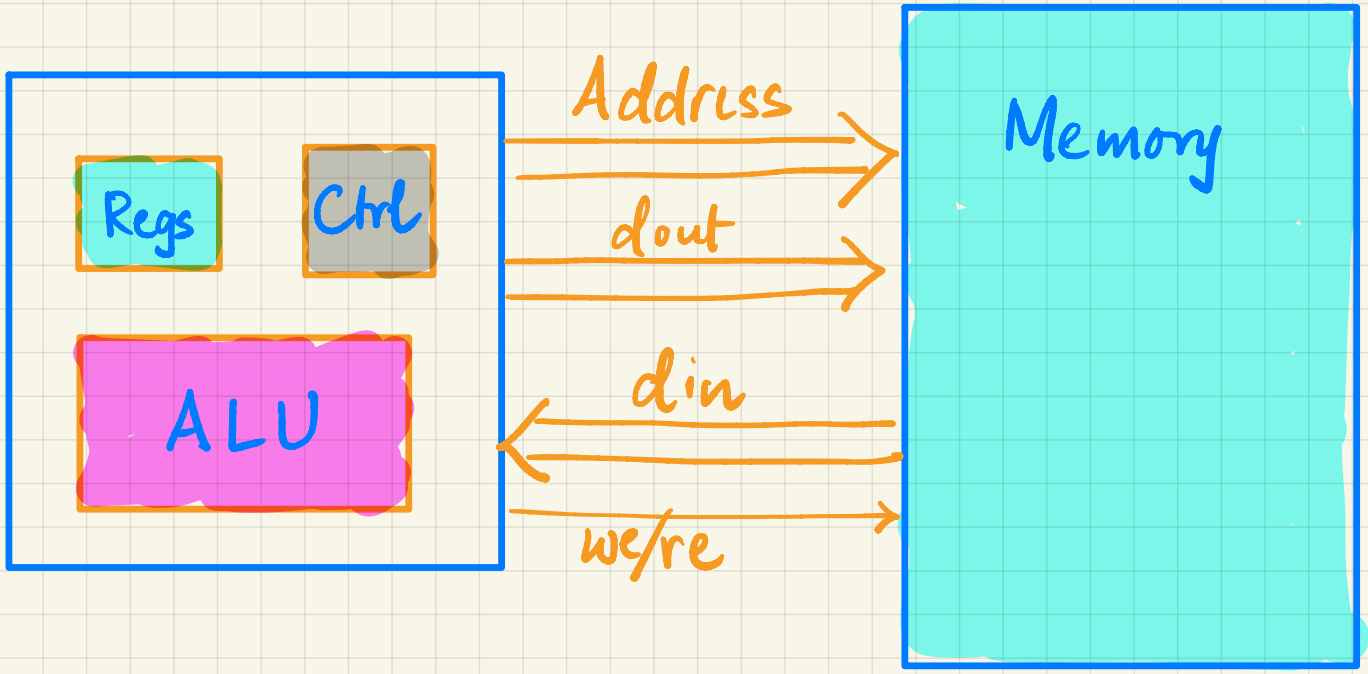
# CPU meets World




Read data from external memory

- Give an address
- Set control to enable read (optional)
- "Latch" data into CPU registers

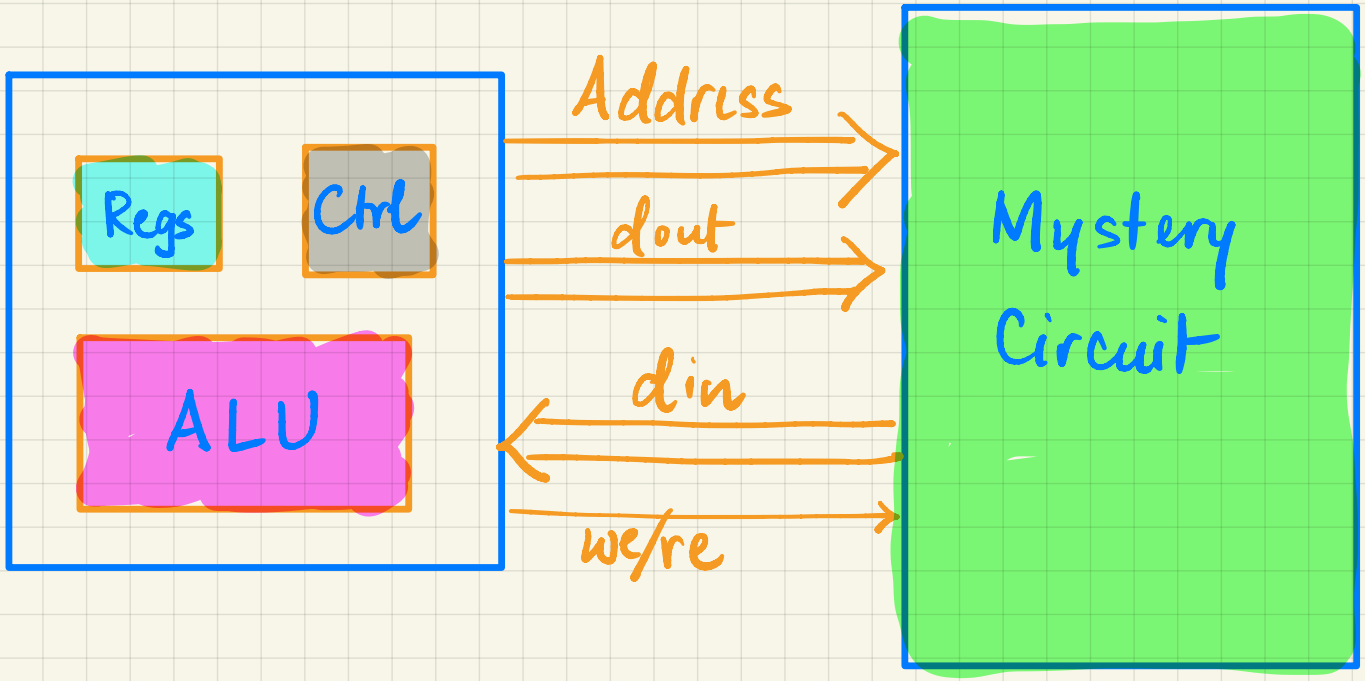
# CPU meets World




Bus : Wires connecting CPU to Memory

- Data
- Control

# CPU meets World

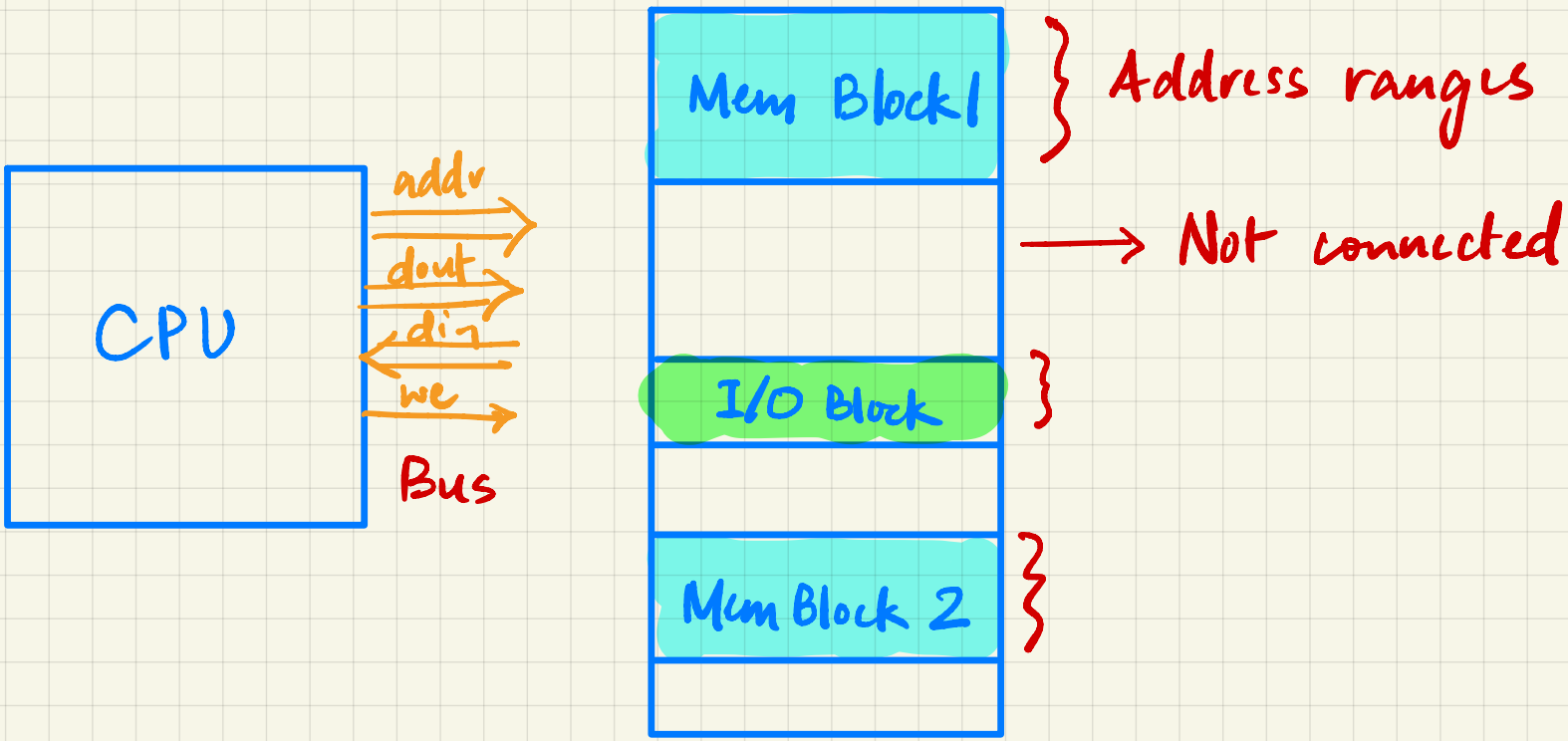



Bus: Wires connecting CPU to Memory

- Data
- Control

CPU cannot tell difference between memory and any other circuit!

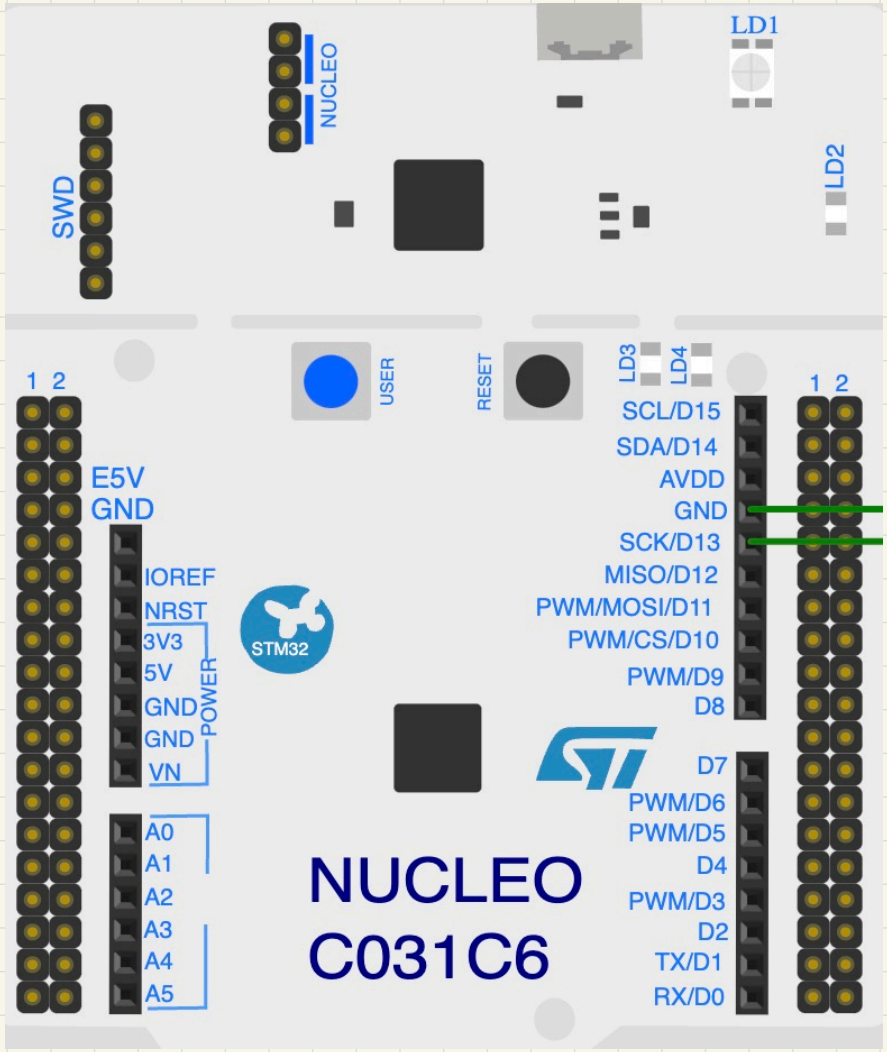
# Memory Map




- Decoding circuitry (digital decoders/multiplexers)
- Generate enable signals for memory / IO as per address
- Response from circuit that was enabled

  ⇒ CPU sees memory and IO same way

# Pins



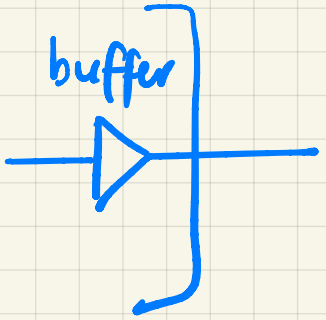

- Power / Ground
  - Digital
  - Analog
- Dedicated Comms
- Input / Output
  - GPIO

# GPIO

- Versatile (general purpose) IO pins
- Interface with devices, sensors, actuators
- Configurable behaviour
- Seen as bits in a register by CPU

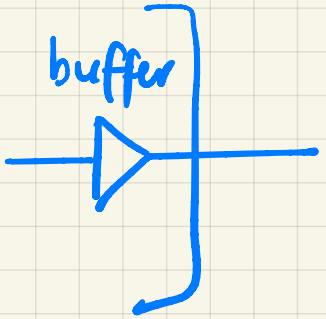
# Modes

- Output ⇒ CPU drives pin

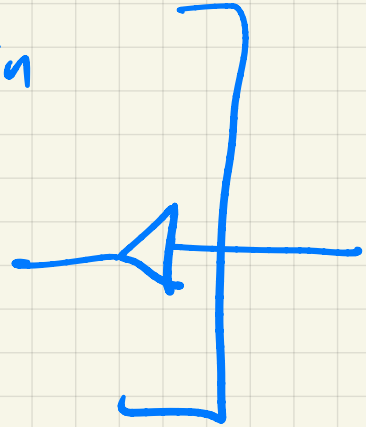


# Modes

- Output ⇒ CPU drives pin

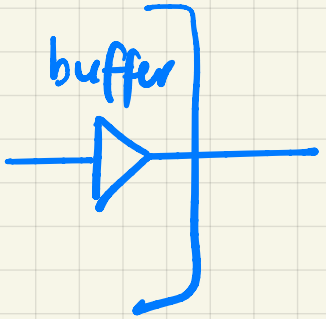



- Input ⇒ external circuit drives pin

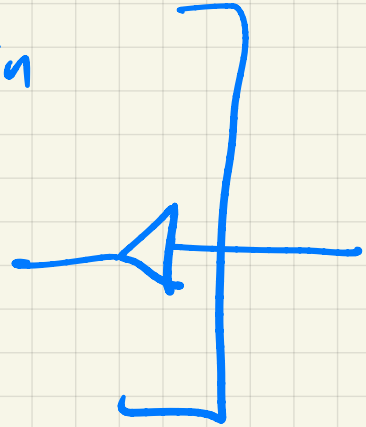


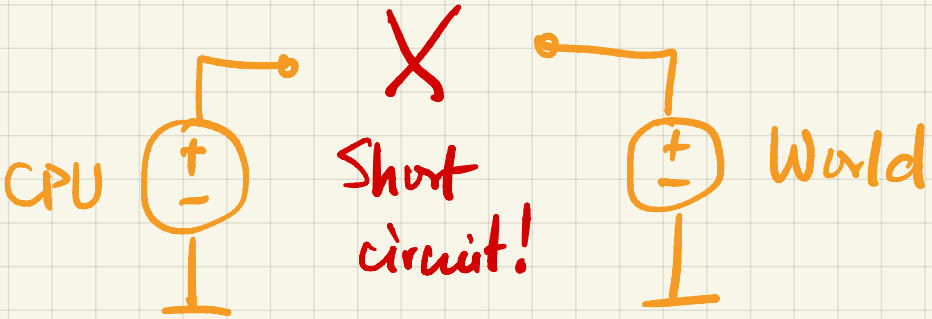
# Modes

- Output ⇒ CPU **drives** pin

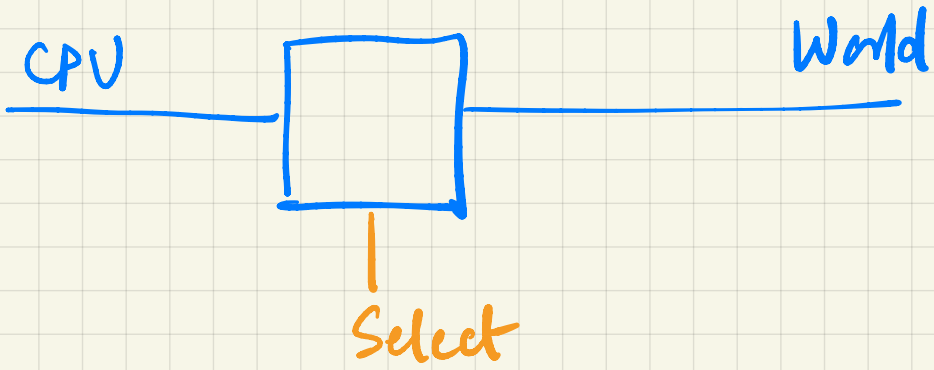



- Input ⇒ **external circuit** drives pin





# Mode select

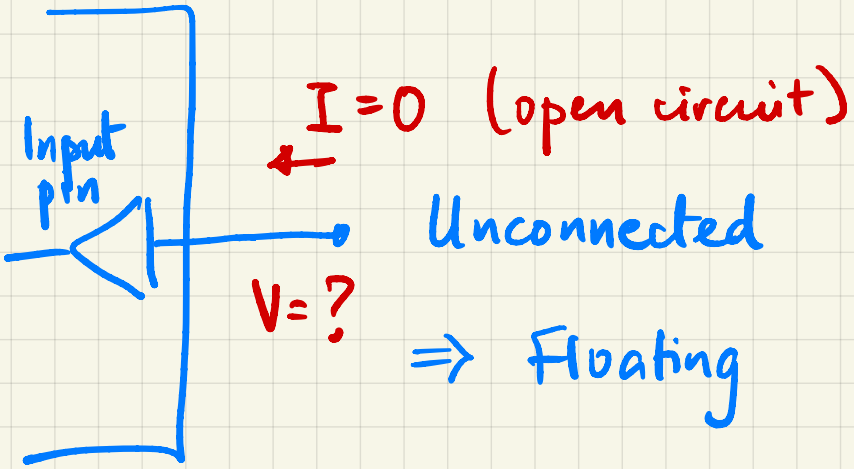



- Isolates driving circuit to prevent short circuit
- Default assume all pins inputs
  - CPU won't try to drive the pins
- Explicitly configure pins as input/output before use

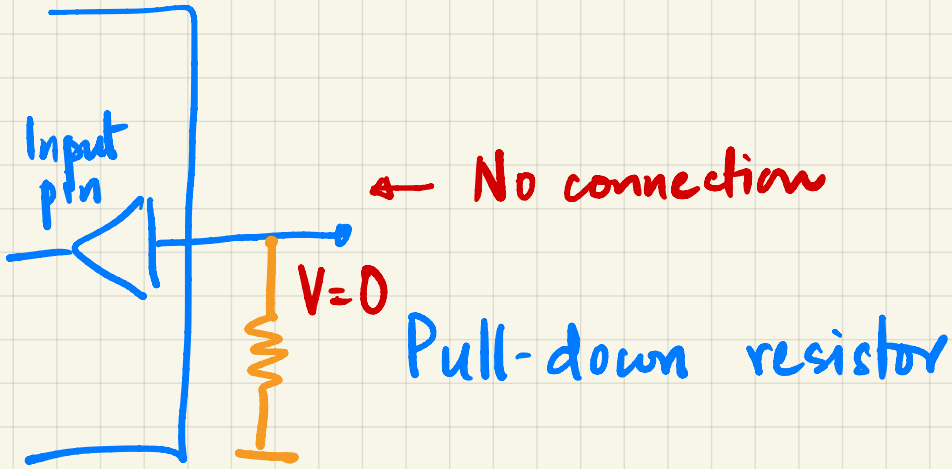
# Analog or Digital

- Real world signals : take arbitrary values
  - Real valued eg. 0 to 3.3V
- Digital signals
  - Represent 0 or 1
  - Ideally 0V or $V_{dd}$
  - Intermediate values thresholded
- Analog to Digital Conversion
  - Limited capabilities but Common

# Pull up / Pull down

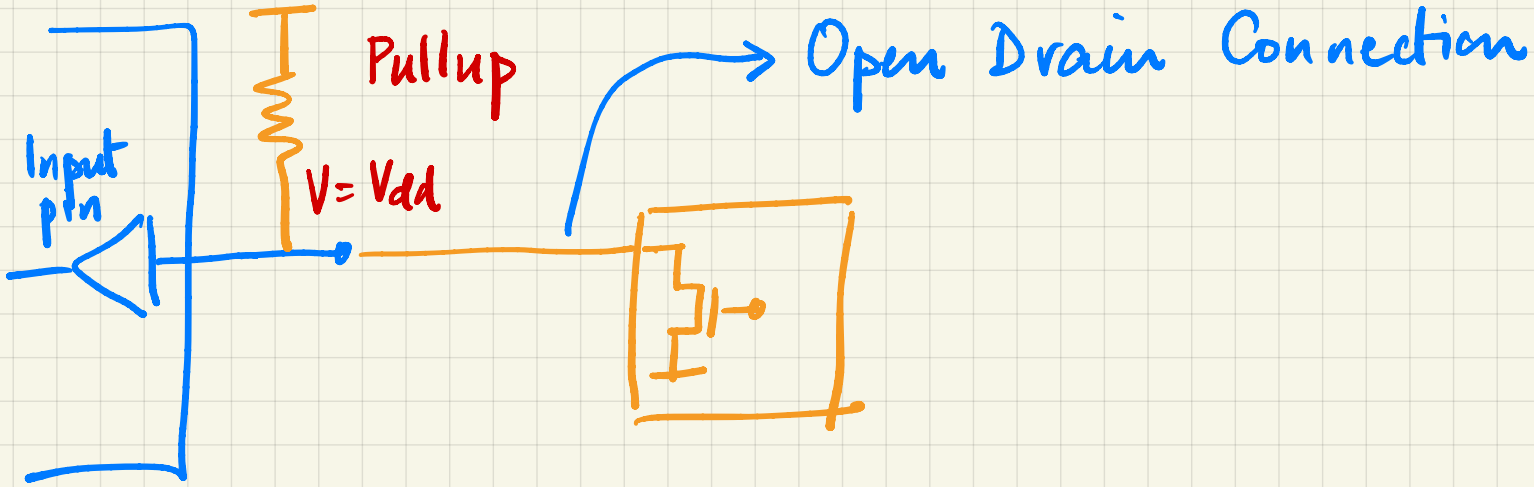


# Pull up / Pull down



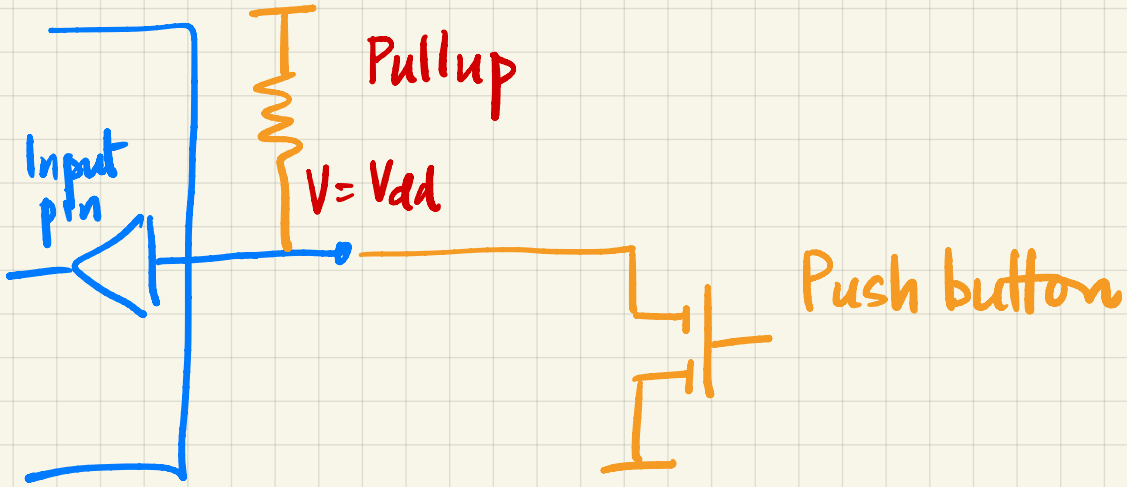

- Unconnected input
- Provide path to known voltage (ground)
- Prevent false triggering

# Pull up / Pull down




- Provide path to known voltage ( $V_{dd}$ )
- Open Drain Peripheral can pull down
  - but cannot pull up to $V_{dd}$
  - Pull up resistor works when peripheral not pulling down

# Pull up / Pull down




- Button either makes or breaks connection
- Broken connection
  ⇒ Pull up resistor pulls to Vdd
- Without resistor => Short circuit

# Hardware Abstraction Layer

- Programming MCU : set bits in registers

- MCU specific
  - hard to generalize
  - varies from one MCU to next

- Abstraction
  - Port definitions, numbers
  - Parameters, set/reset values

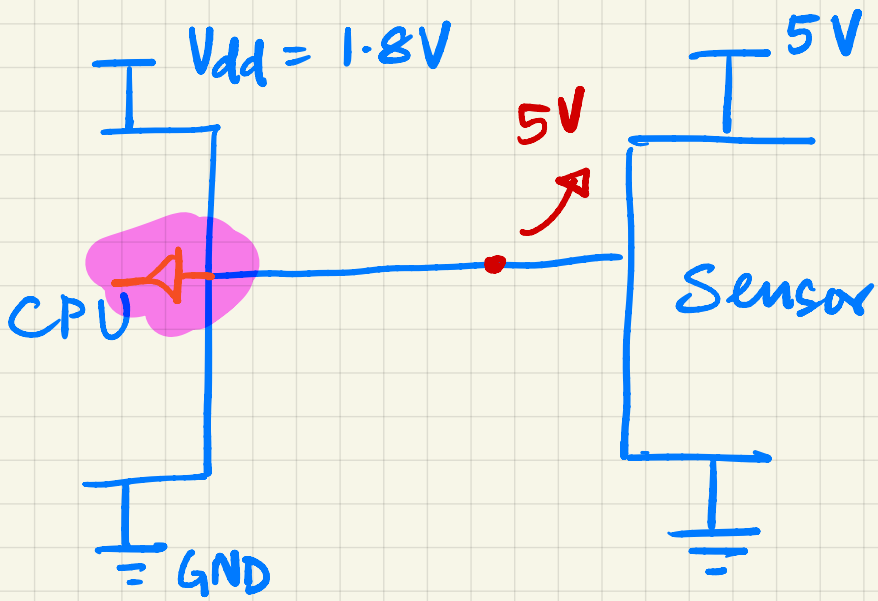
limited set of files customized

# HAL example

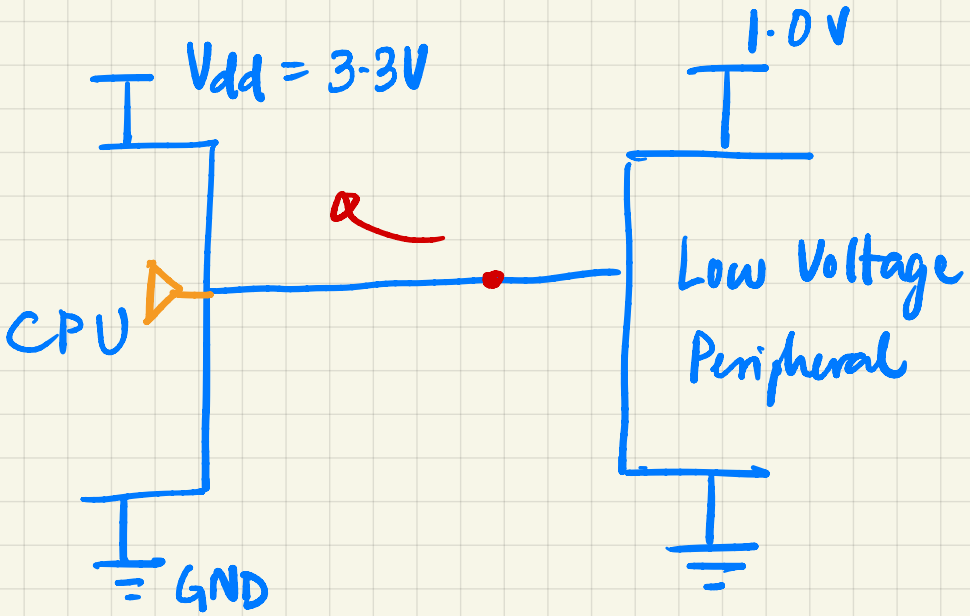
```
void initGPIO()
{
  GPIO_InitTypeDef GPIO_Config;

  GPIO_Config.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_Config.Pull = GPIO_NOPULL;
  GPIO_Config.Speed = GPIO_SPEED_FREQ_HIGH;

  GPIO_Config.Pin = LED_PIN;

  SET_BIT(RCC->IOPENR, RCC_IOPENR_GPIOAEN);
  HAL_GPIO_Init(LED_PORT, &GPIO_Config);

}
```

# GPIO Usage

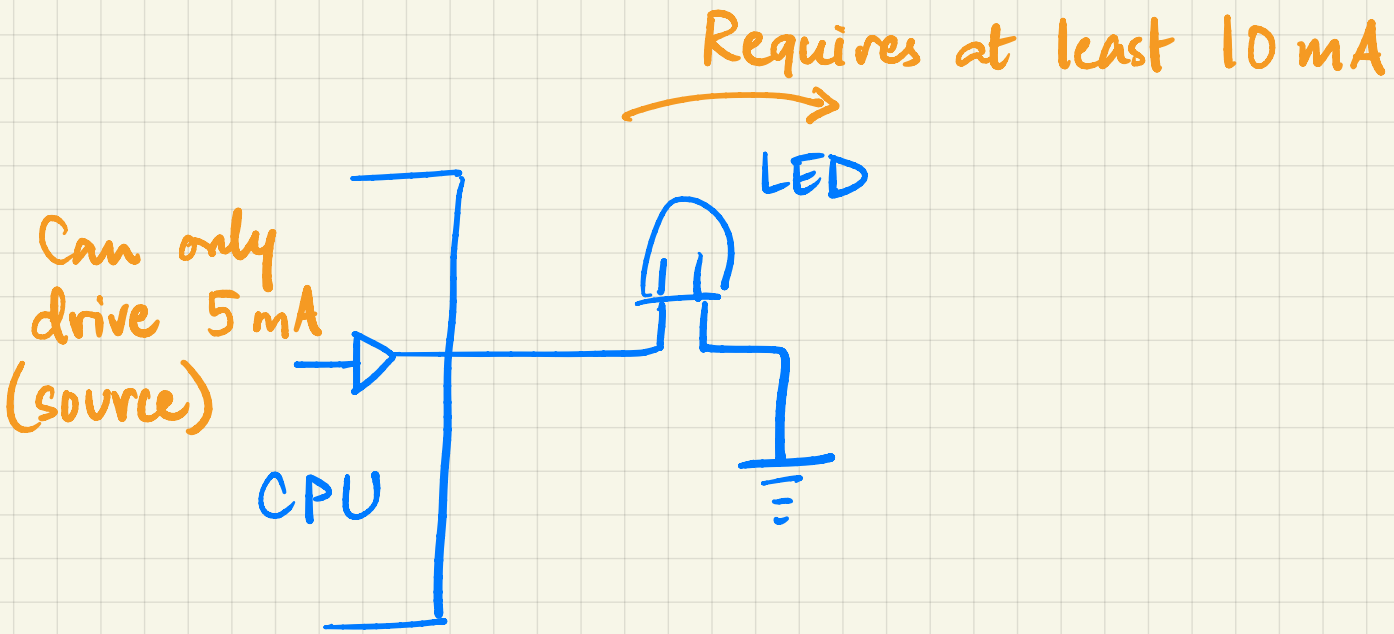
# Voltage Levels



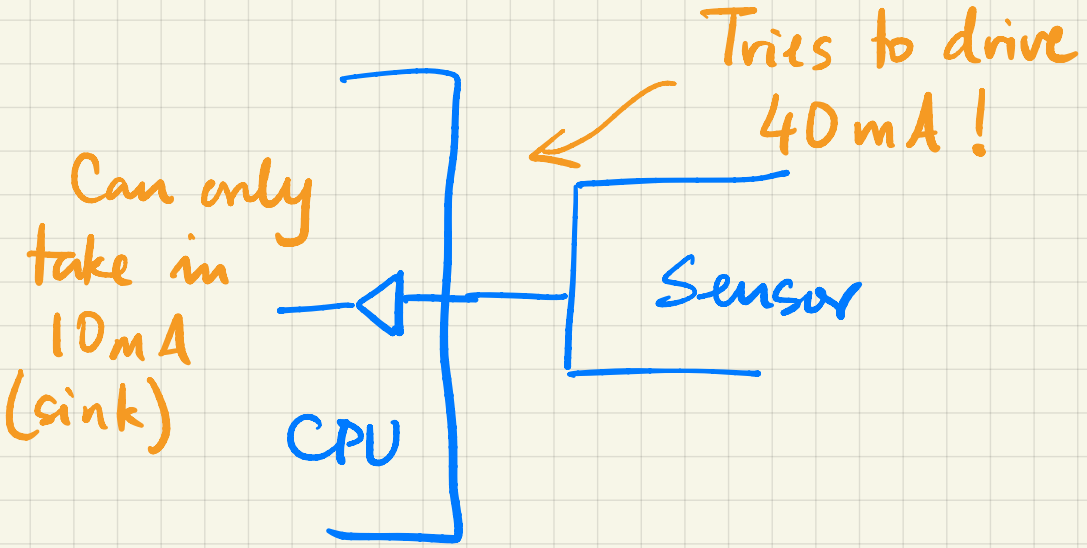

Potential for Damage!



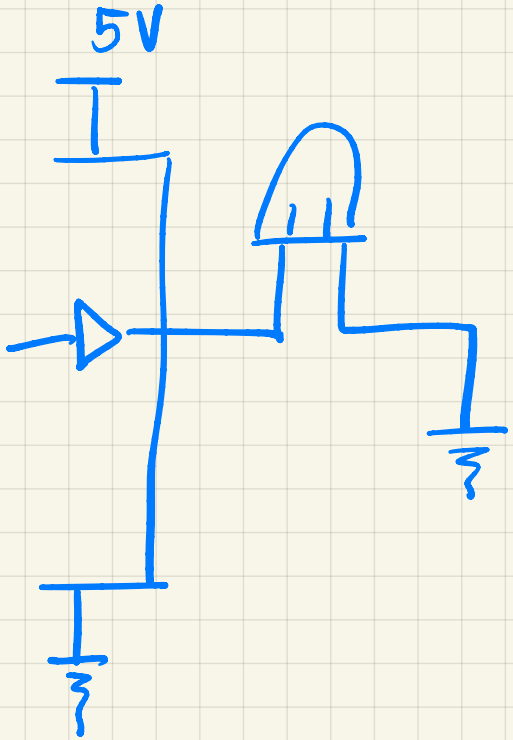
# Current Capacity

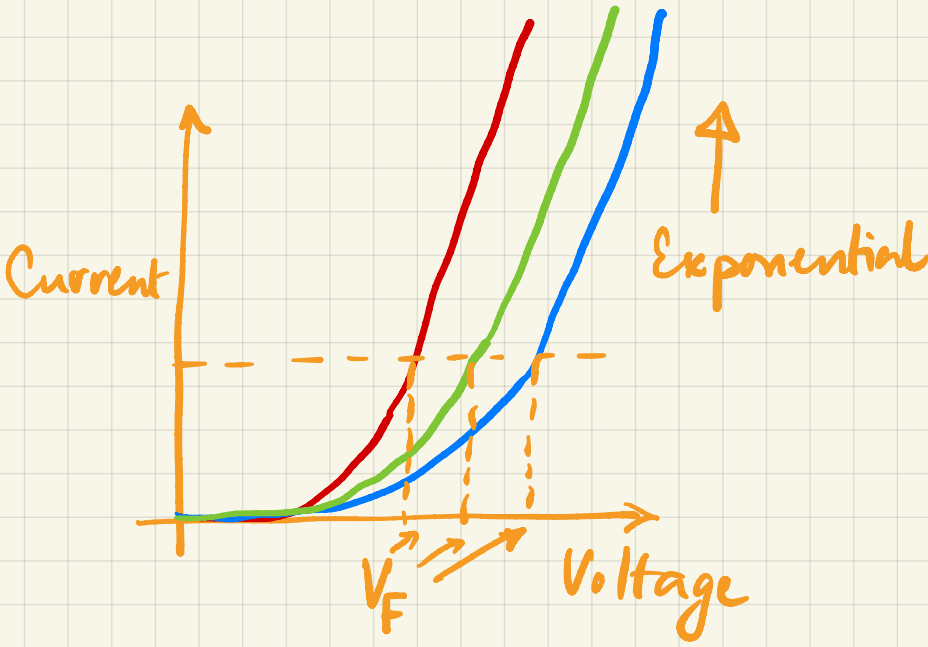



Potential for Damage!



# LED current limiting

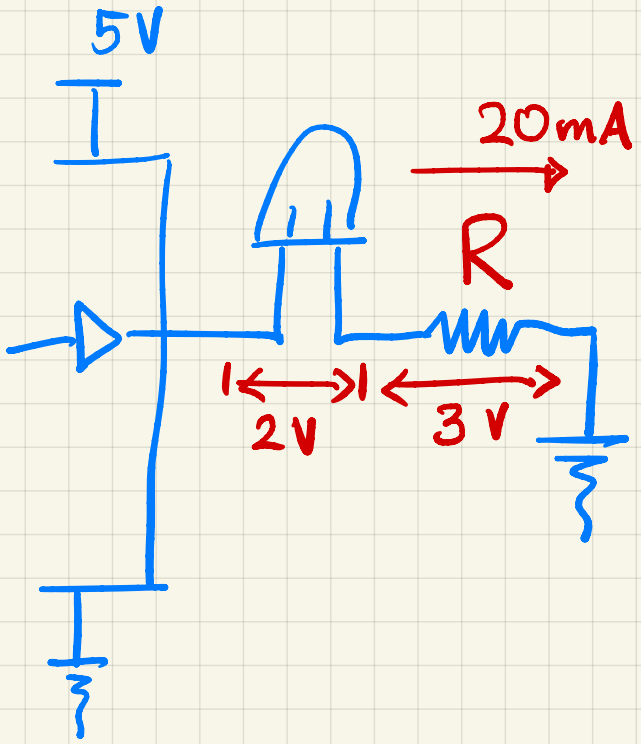


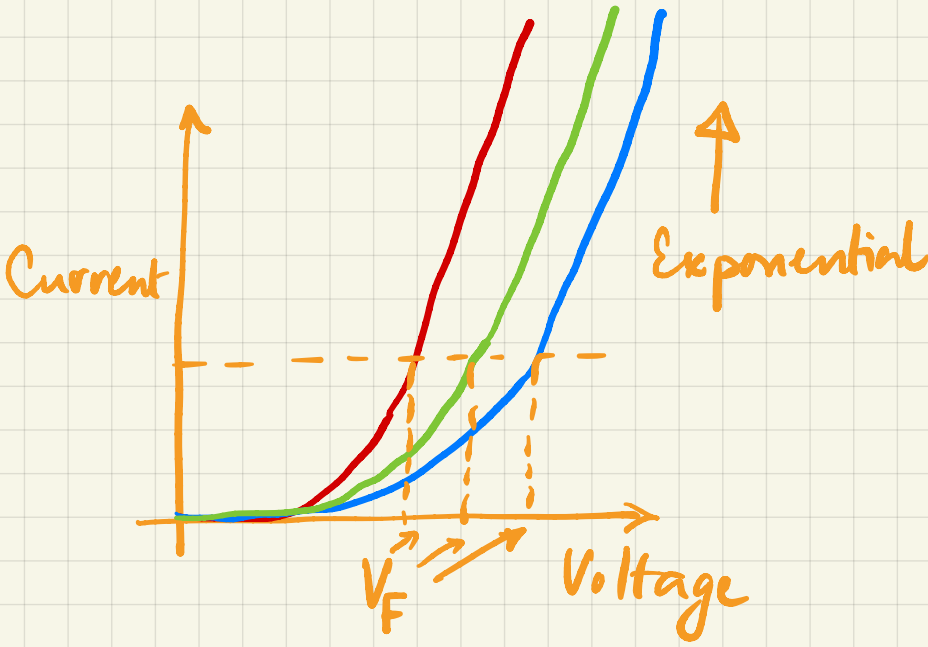

Example

- I at 2 V = 10 mA
- I at 2.5V = 50 mA
- 5 V ⇒ Breakdown

# LED current limiting




Current limiting resistor

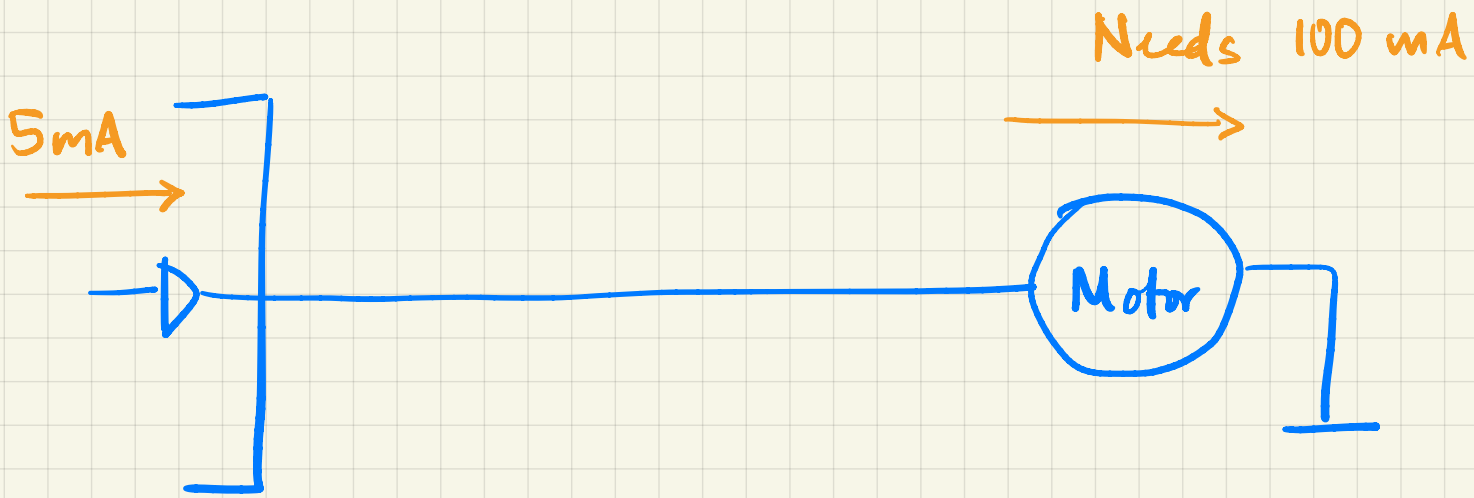



Example

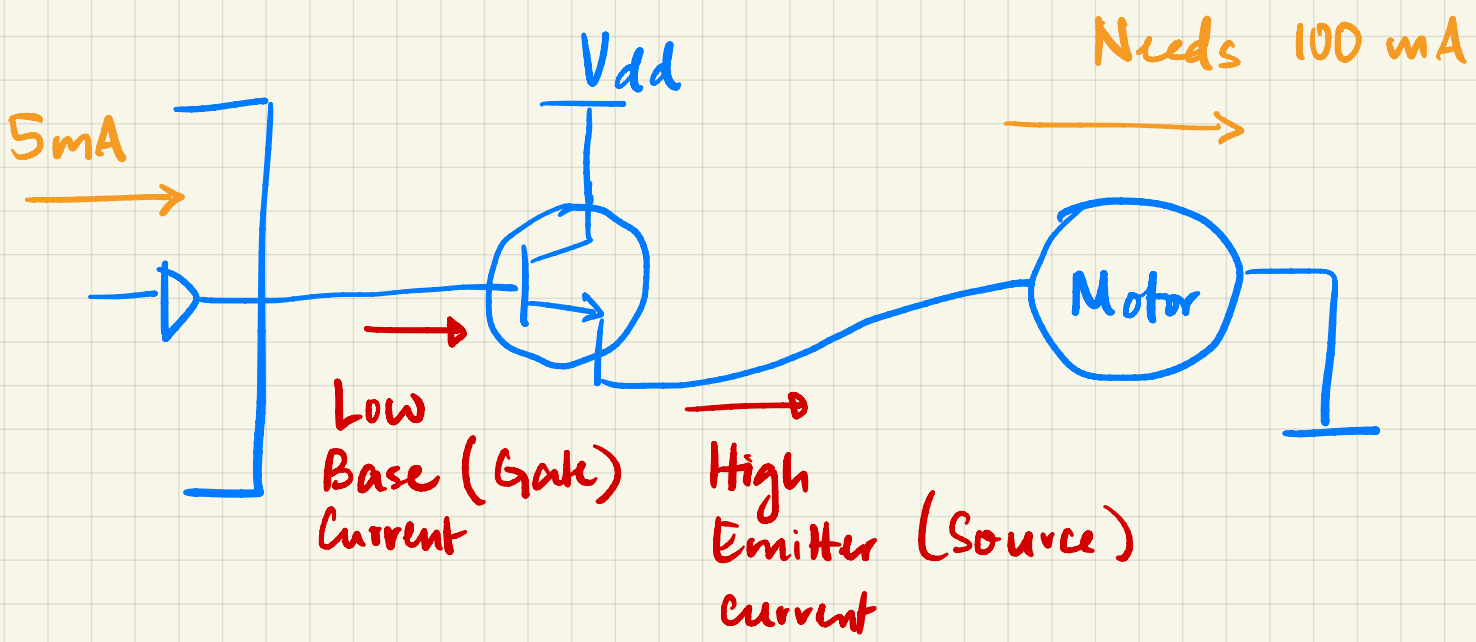
- 20mA rated current (for good brightness from datasheet)
- $V_f = 2V$

$$R = \frac{(5-2)}{20} = 150\ \Omega$$

# Current Drive



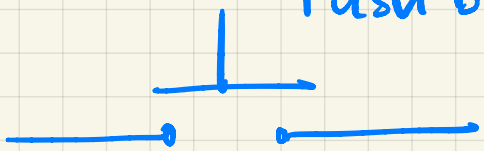
# Current Drive




- Transistor or Relay
- Low control current
- High load drive capacity

# Debouncing

Push button → mechanical switch



⇒ "Bounce" on physical contact

- Hardware – latches, custom debounce circuits
- Software – Wait till stable, check for no change over interval

# ESD protection

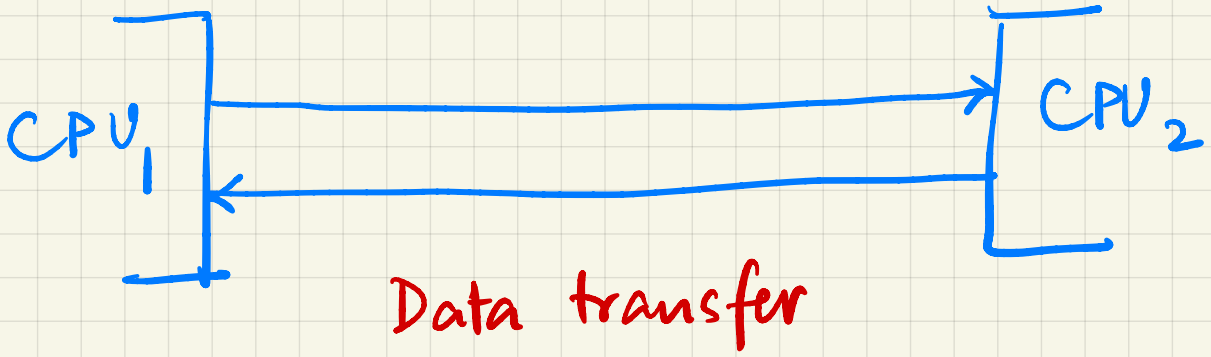
- Electro Static Discharge
- Buildup of static charge on surfaces / bodies
  - Never touch pins directly !
- Some circuit level protection possible
  - Not foolproof

# GPIO Examples

- LED control - signaling, status
- Button inputs
- Driving Relays / Transistors
- Sensor Interfacing
  - Analog, Digital
- Communication Protocols
  - Bit Banging (software implementation)

# Serial Communication (UART)

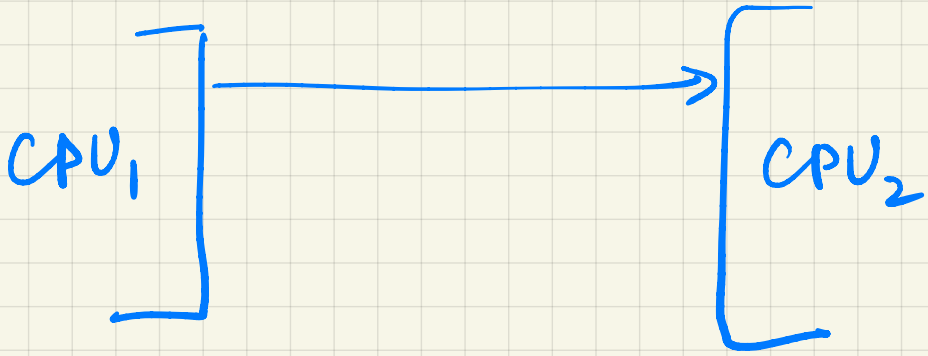
# Communication




Requirements

- Simplicity
- Asynchronous – no shared clock

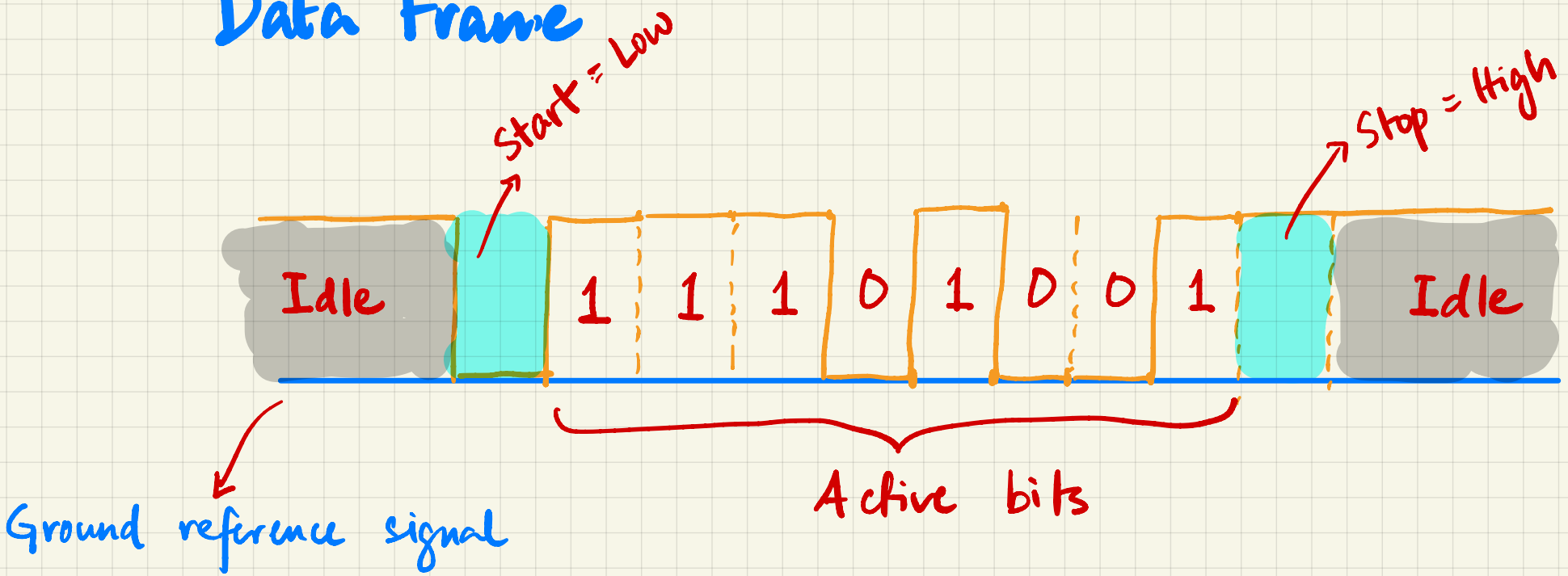
# Asynchronous




Assume shared Ground pin
(implicit 2nd pin here)

- How does $CPU_2$ know when communication starts
- How does $CPU_2$ know what bits were sent?

# Data Frame




- Default (Idle) = high voltage on line
- Pull low for 1 bit duration => start
- Next 8 (or whatever) bits => data
- Optimal Parity (for error detection)
- At least 1 bit interval high => stop

# Parameters

8 N 1 @ 9600

- 9600 bits/second (Baud - actually means symbol rate.)
- 8 ⇒ number of data bits
- N ⇒ no parity used
- 1 ⇒ One stop bit

— Receiver must know these parameters
  - else wrong decoding/errors

# Modes

- Simplex – Only in one direction

- Half-Duplex – Both directions
  – But only one at a time

- Full Duplex – Both directions simultaneously

– Configurable: # data bits
  Parity
  Data rate: up to 115200 bps commonly supported.

# Why UART

- Simplicity
  - Minimal hardware (2 wires if GND common)
  - No clock synchronization needed
- Versatility
  - Widely supported
  - Possible to emulate in software if needed
  - FPGA cores available
- Low cost

# Disadvantages

- Speed - Lower data rates
  - Sensitive to Baud rate mismatch
  - Higher speeds X longer distances
- No shared clock - hard to maintain timing
- Multi-master / Multi-slave not possible (normally)
- Framing Overhead significant
- Very basic error detection / no correction
- Cable length, Signal integrity
- Resources: CPU usage, dedicated pins

# Console Interface

- Text based interface
  - Debug, Monitor, Configuration, Control
- Usually a serial port
  - Easy to implement, standard protocol
  - Minimal hardware, no clock needed
  - Real-time, Interactive execution possible

# printf and friends

- UART can send bytes (8-bit) ⇒ ASCII codes
- Implement a `_write()` function in C
  - printf will call `_write()` to send characters
  - portable : implement one custom function
    ⇒ get full output
- Similar approach with `scanf()` and `_read()`

# Datasheets

# Datasheets

- Electronic components description
- Know what to look for and where
- Examples
  - STM32 μC
  - Standard Red LED (Farnell)