UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

SE 423: Introduction to Mechatronics

Lecture 3: GPIO

Marius Juston

Wednesday, January 28th, 2026

Reminders

Office Hours:

- **Monday:** 4-6 PM, Neel
- **Tuesday:** 11-1 PM, Lakshmi
- **Tuesday:** 1-3 PM, Samuel
- **Tuesday:** 2-5 PM, Dan & Marius

Lab: Starting Lab 1 this week, will be using C so be sure to be prepared! (don't forget to turn in commented code)!

Homeworks (Next week):

- All check-offs for homework 1 are due **February 3, 5PM**
- Answers and code submissions for homework 1 are due **February 4, 9AM**

LabVIEW (Next week):

- All check-offs for LabVIEW 1 are due **February 5, 5PM**

Small easy Quiz at the end of this lecture, like Monday, due at 11AM.

Questions?

Homework, Lab, LabVIEW, quiz questions?
(I sent an update for the reason for the 2 counters question)

Digital Inputs / Outputs

Digital Inputs / Outputs

What does it mean for something to be “digital”?

It is the control of your system / sensor using binary (0 or 1) signals.

Digital I/O is the lowest level electrical interface between software and hardware.

Almost every mechatronic system constrains at least one digital input / output.

Sensors often reduce complex physical phenomena to binary decisions (limit reached / not reached, turned on or off)

Digital signals are inherently discrete

Digital I/O provides deterministic, low-latency interaction with the environment

Digital Outputs

A digital output drives a pin to a logic level:

- Logic 0 (LOW) → ≈0 V
- Logic 1 (HIGH) → ≈3.3 V / 5 V

Logic levels are represented by voltage ranges, not exact values

Internally controlled by a latch / register
Can source or sink limited current (GPIO should not be used to drive power)

Digital outputs are not power supplies and must not directly drive loads. The GPIO have a total maximum of current it can supply, shared throughout all the pins.

If you power too many devices using GPIO, you might run out of power!

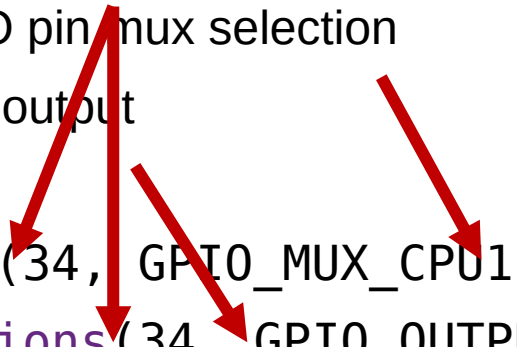
Examples:

- Controlling LEDs

Digital Outputs

Digital output control consists of :

1. Choosing the GPIO number
2. Selecting the GPIO pin mux selection
3. Setting direction = output



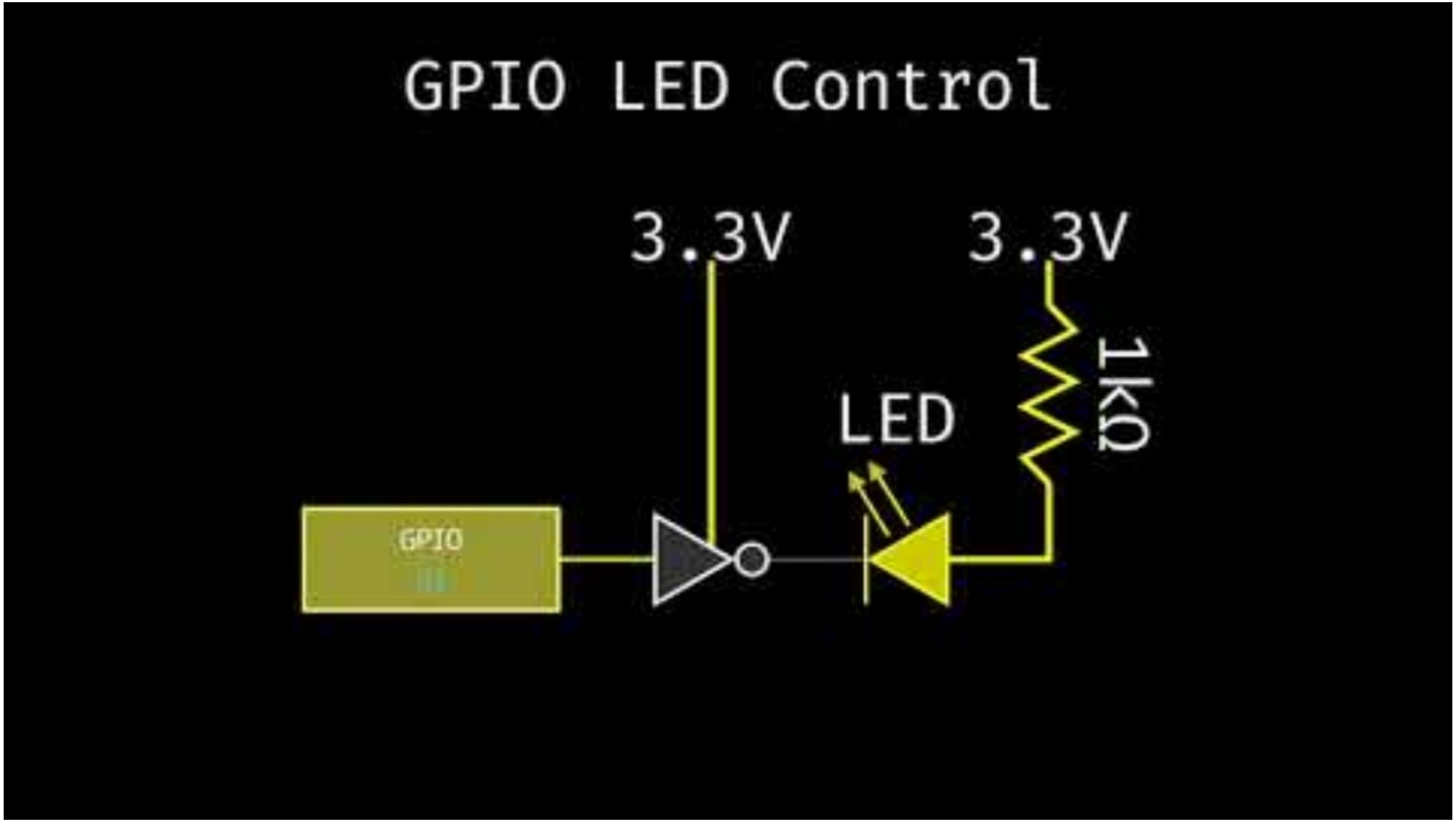
```
GPIO_SetupPinMux(34, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(34, GPIO_OUTPUT, GPIO_PUSHPULL);
```

You never write voltage, you write **bits in registers**

Digital Outputs

		GPIO Mux Selection								
		GPIO Index	0,4,8,12	1	2	3	5	6	7	15
ME461	LaunchPad	GPyGMUXn	00,01,10,11	00b			01b			11b
Default	Pins	GPyMUXn	00b	01b	10b	11b	01b	10b	11b	11b
PWM1A	40/J4.10		GPIO0	EPWM1A				SDAA		
PWM1B	39/J4.9		GPIO1	EPWM1B		MFSRB		SCLA		
PWM2A	38/J4.8		GPIO2	EPWM2A			OUTXBAR1	SDAB		
PWM2B	37/J4.7		GPIO3	EPWM2B	OUTXBAR2	MCLKRB	OUTXBAR2	SCLB		
PWM3A	36/J4.6		GPIO4	EPWM3A			OUTXBAR3	CANTXA		
PWM3B	35/J4.5		GPIO5	EPWM3B	MFSRA	OUTXBAR3		CANRXA		
EQEP3A	80/J8.10		GPIO6	EPWM4A	OUTXBAR4	EXTSYNCOUT	EQEP3A	CANTXB		
EQEP3B	79/J8.9		GPIO7	EPWM4B	MCLKRA	OUTXBAR5	EQEP3B	CANRXD		
PWM5A	78/J8.8		GPIO8	EPWM5A	CANTXB	ADCSOCAO	EQEP3S	SCITXDA		
PWM5B	77/J8.7		GPIO9	EPWM5B	SCITXDB	OUTXBAR6	EQEP3I	SCIrXDA		
PWM6A	76/J8.6		GPIO10	EPWM6A	CANRXB	ADCSOCBO	EQEP1A	SCITXDB		UPP-WAIT
GPIO11	75/J8.5		GPIO11	EPWM6B	SCIRXDB	OUTXBAR7	EQEP1B	SCIRXDB		UPP-START
CANTXB	J12		GPIO12	EPWM7A	CANTXB	MDXB	EQEP1S	SCITXDC		UPP-ENA
GPIO14/SCITXB	74/J8.4		GPIO14	EPWM8A	SCITXDB	MCLKXB		OUTXBAR3		UPP-D6
GPIO15/SCIRXDB	73/J8.3		GPIO15	EPWM8B	SCIRXDB	MFSXB		OUTXBAR4		UPP-D5
BUZZER	33/J4.3		GPIO16	SPISIMOA	CANTXB	OUTXBAR7	EPWM9A		SD1_D1	UPP-D4
CANRXB	J12		GPIO17	SPISOMIA	CANRXB	OUTXBAR8	EPWM9B		SD1_C1	UPP-D3
SPICLKA/SPIRAM	4/J1.4		GPIO18	SPICLKA	SCITXDB	CANRXA	EPWM10A		SD1_D2	UPP-D2
SPIRAM CS	3/J1.3		GPIO19	SPISTEA	SCIRXDB	CANTXA	EPWM10B		SD1_C2	UPP-D1
EQEP1A	J14		GPIO20	EQEP1A	MDXA	CANTXb	EPWM11A		SD1_D3	UPP-D0
EQEP1B	J14		GPIO21	EQEP1B	MDRA	CANRXB	EPWM11B		SD1_C3	UPP-CLK
LED1/PWM	8/J1.8		GPIO22	EQEP1S	MCLKXA	SCITXDB	EPWM12A	SPICLKB	SD1_D4	
WIZNET INT	34/J4.4		GPIO24	OUTXBAR1	EQEP2A	MDXB		SPISIMOB	SD2_D1	
LS7366#3 CS	51/J6.1		GPIO25	OUTXBAR2	EQEP2B	MDRB		SPISOMIB	SD2_C1	
Ls7366#4 CS	53/J6.3		GPIO26	OUTXBAR3	EQEP2I	MCLKXB	OUTXBAR3	SPICLKB	SD2_D2	
WIZNET RST	52/J6.2		GPIO27	OUTXBAR4	EQEP2S	MFSXB	OUTXBAR4	SPISTEB	SD2_C2	
F28027 CS	11/J2.1		GPIO29	SCITXDA	EMSDCKE		OUTXBAR6	EQEP3B	SD2_C3	
Board Blue LED			GPIO31	CANTXA	EM1WE		OUTXBAR8	EQEP3I	SD2_C4	
GPIO32	2/J1.2		GPIO32	SDAA	EM1CS0					
Board Red LED			GPIO34	OUTXBAR1	EM1CS2			SDAB		

Controlling an LED



Digital Inputs

Digital input samples external voltage

Compared against logic thresholds:
- LOW
- HIGH

Examples:
- Buttons
- Switches
- Some types of encoders

Problem with floating inputs, if input is not driven:
- Voltage is undefined
- Susceptible to noise
- Input may randomly read 0 or 1

More on this with ADC

Pull-Up / Pull-Down Resistors

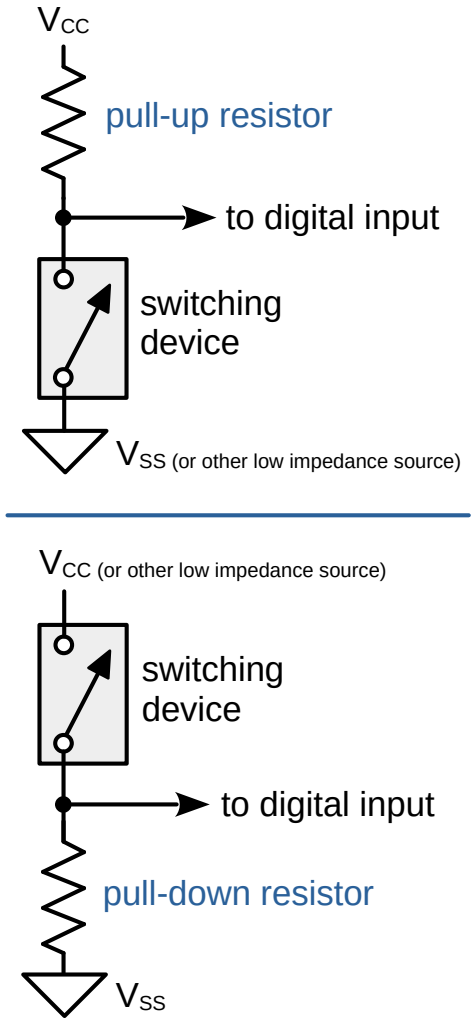
A pull-up resistor:
- Connects input to Vcc
- Defines default logic HIGH

When switch is:
- Open: input → HIGH
- Closed: input short circuits to GND → LOW

A pull-down resistor (not used in our application):
- Connects input to GND
- Defines default logic LOW

When switch is:
- Open: input → LOW
- Closed: input short circuits to GND → HIGH

This is all internal on the Red Board, makes it cleaner!
You don’t need to understand this, just put it in code!



Digital Inputs

Digital output control consists of :

1. Choosing the GPIO number
2. Selecting the GPIO pin mux selection
3. Setting direction = input
4. Enable pull-up resistor

```
GPIO_SetupPinMux(7, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(7, GPIO_INPUT, GPIO_PULLUP);
```

You read the binary input using:

```
if(GpioDataRegs.GPADAT.bit.GPIO7){
    // Do stuff
}
```

Push-Pull - Output

A **push-pull output** uses **two transistors**:

- A PMOS transistor pulls the pin up to 3.3 V
- An NMOS transistor pulls the pin down to GND

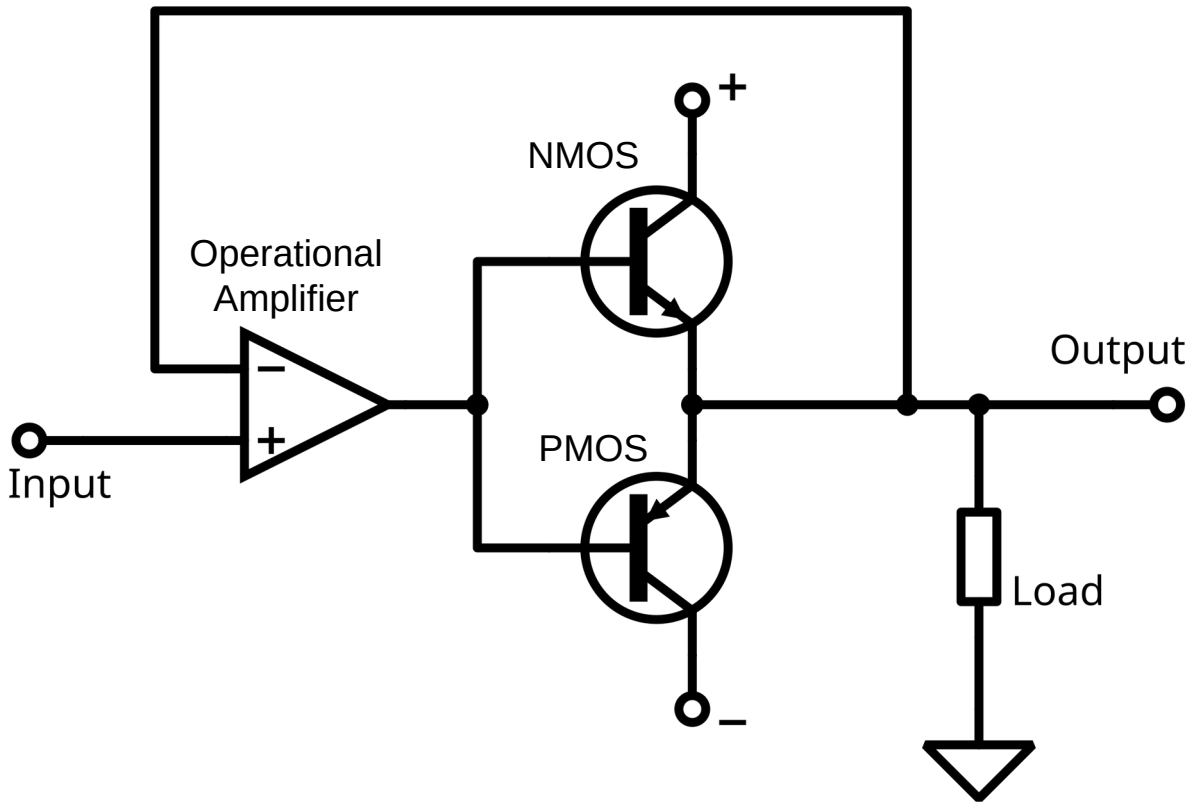
Only **one is ON at a time**.

When a pin is an output:

- You want deterministic voltage
- You want noise immunity
- You want defined transitions

A more detailed video here.

You don't need to understand this, just put it in code!



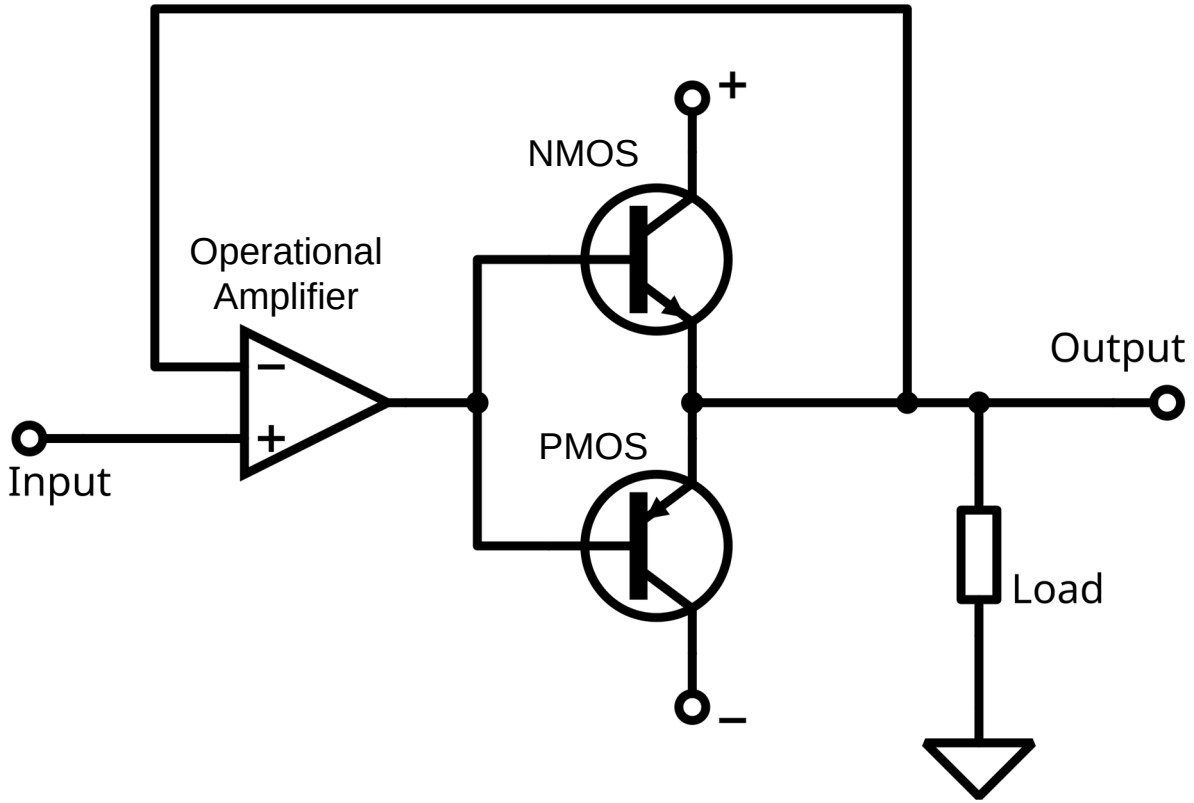
(I wanted to create the animation for this, but if this text is still here, then it has not been done yet.)

Push-Pull - Output

A pull-up resistor is:

- Slow
- Incapable of pulling LOW

So, for outputs, pull-ups are electrically irrelevant and often harmful.



(I wanted to create the animation for this, but if this text is still here, then it has not been done yet.)

Digital Outputs – too many GPIO output

Why would we run out of power?
Each GPIO output has its own transistors and can safely source or sink only a limited current.

However, all GPIOs draw that current from shared internal power and ground networks inside the chip.

Even if each individual pin is within its limit, the sum of all GPIO currents can exceed what the silicon, bond wires, or package can safely carry.

This is why microcontrollers specify both per-pin and total GPIO current limits, and why **GPIOs must never be used as power supplies**.

Think of each GPIO as a faucet drawing water from the same pipe.

- Each faucet has a **local limit** (per-pin current)
- The pipe has a **global flow limit** (total GPIO current)

Opening too many faucets:

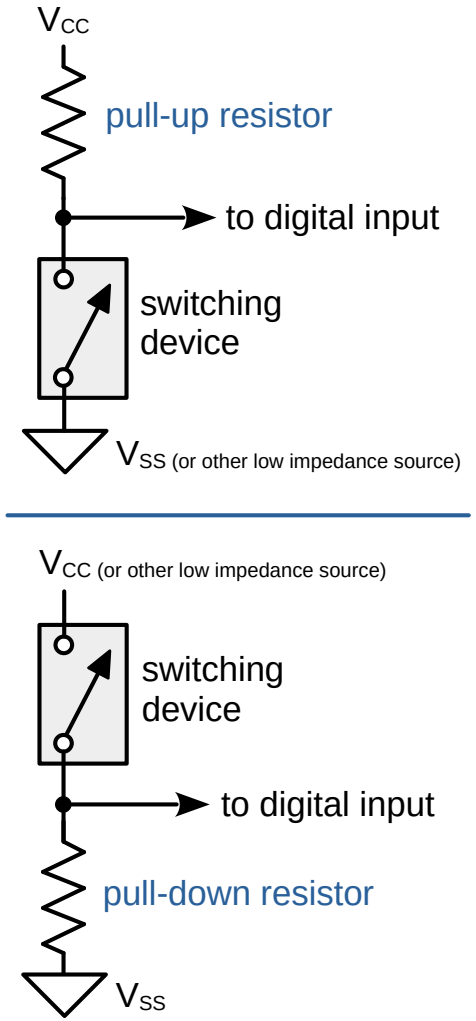
- Reduces pressure
- Stresses the pipe
- Causes failure elsewhere

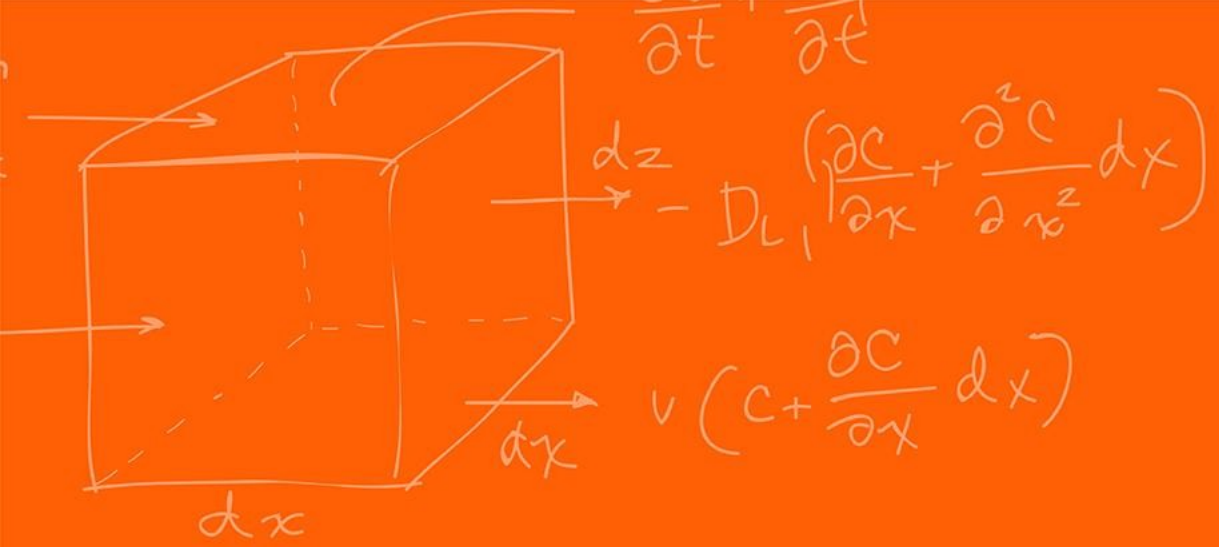
Pull-Up / Pull-Down Resistors - Input

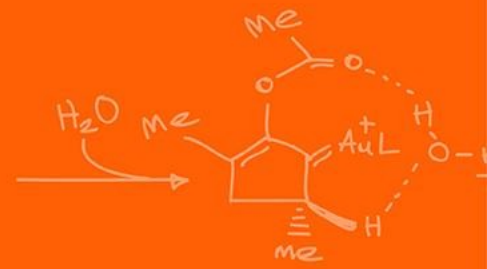
What an input pin is:
- Pin becomes high-impedance, sensitive to low current, allowing for high voltages
- Sensitive to noise

The internal pull-up:
- Weak resistor (~20-50 kΩ)
- Slightly biases pin to logic HIGH
- Can be overridden by external hardware (button to GND)

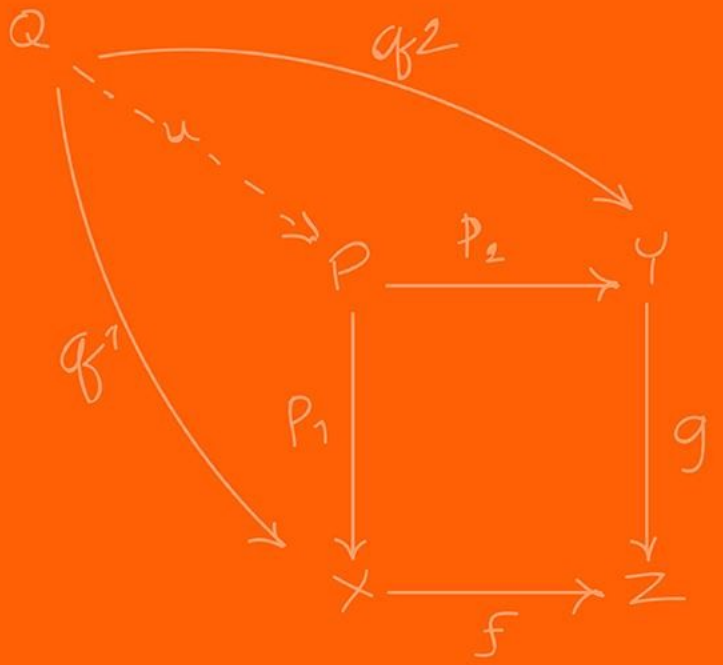
Without pull-up, just using floating-inputs, you would be sensitive to random noise, returning:
- Random bits
- Thus, unpredictable control behavior







Pin Mux



Pin Mux

The Red Board has only **80** pins but **160** GPIO!

We also need the pins to do multiple specialized things,

- GPIO (General Purpose Input-Output)
- PWM output (Pulse Width Modulation)
- SPI (Serial Peripheral Interface)
- I2C (Inter-Integrated Circuit)
- CAN (Controller Area Network)
- UART (Universal Asynchronous Receiver-Transmitter)
- ADC trigger, etc. (Analog-to-Digital Converter)

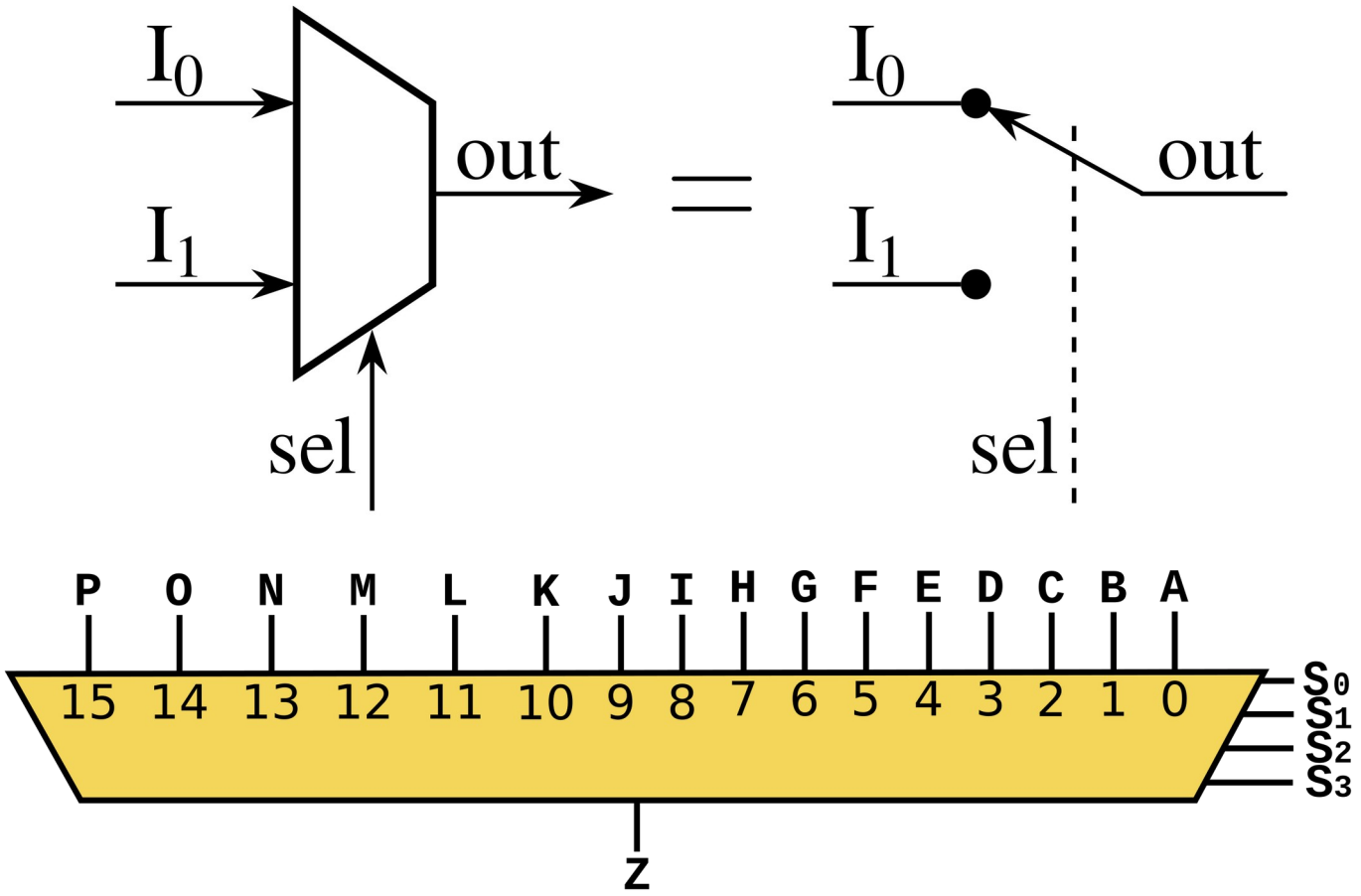
Whole list is in the pin mux list, you will be using this in Lab 1!

This sharing is managed by the **pin multiplexer**!

Pins and GPIOs are not for hardware wiring / optimality reasons.

What is the Pin Multiplexer (PinMux)

This is a hardware switch that selects which peripheral controls a pin:



Pin Mux

Up to twelve independent peripheral signals are multiplexed on a single GPIO-enabled pin in addition to the CPU-controlled I/O capability.

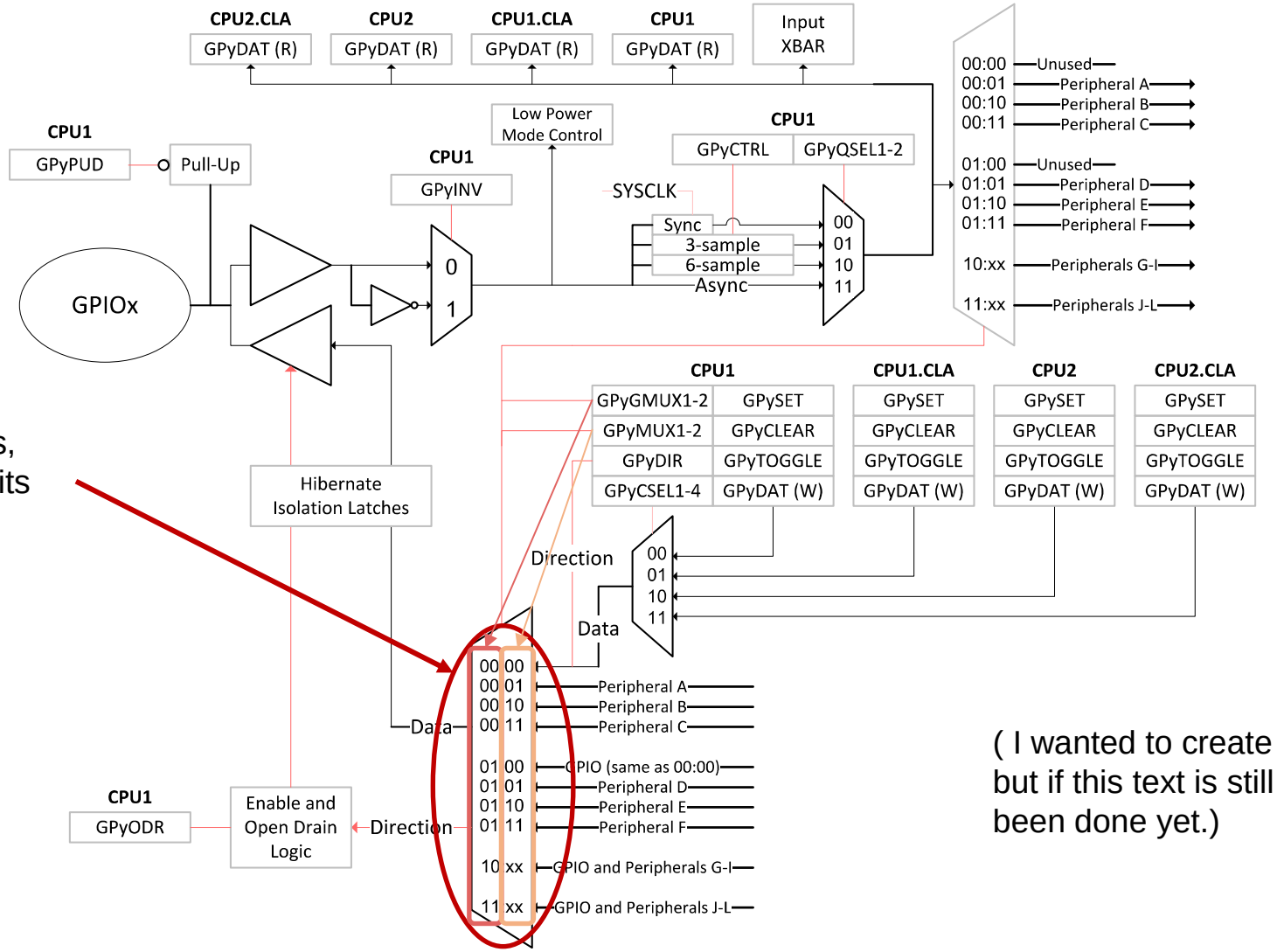
Each pin output can be controlled by either a peripheral or one of the four CPU masters (CPU1, CPU1.CLA, CPU2, or CPU2.CLA). There are six I/O ports:

- Port **A** consists of GPIO0-GPIO31
- Port **B** consists of GPIO32-GPIO63
- Port **C** consists of GPIO64-GPIO95
- Port **D** consists of GPIO96-GPIO127
- Port **E** consists of GPIO128-GPIO159
- Port **F** consists of GPIO160-GPIO168

```
GpioDataRegs.GPATOGGLE.bit.GPIO34 = 1;
```

Pin Mux

Figure 8-1. GPIO Logic for a Single Pin



PinMux has 12 inputs, requires how many bits in the selector?

(I wanted to create the animation for this, but if this text is still here, then it has not been done yet.)

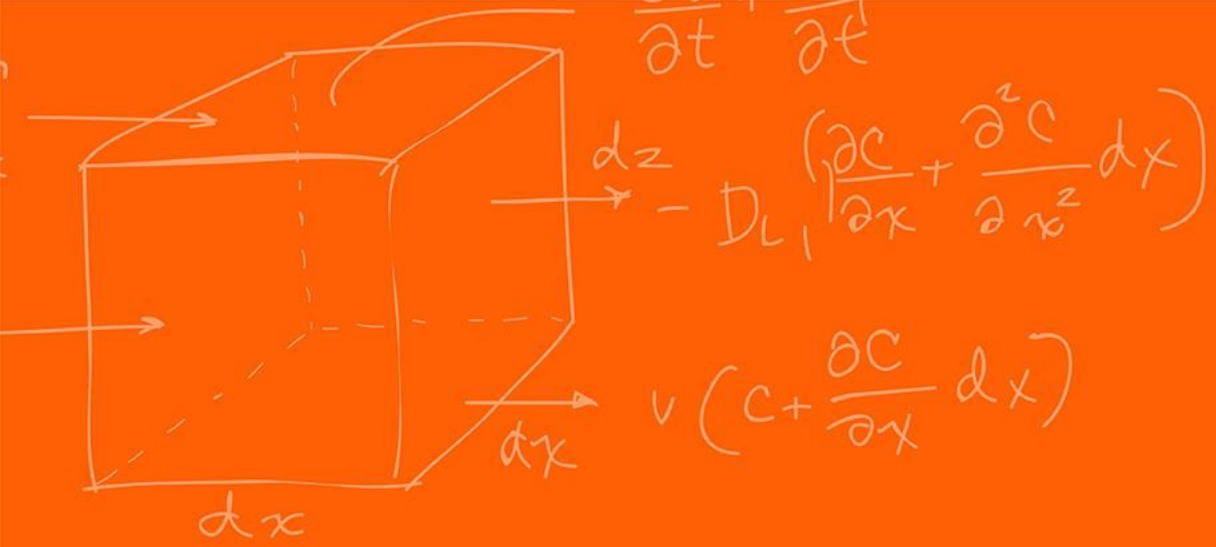
Pin Mux – Code (Not the exact code for reduced spacing)

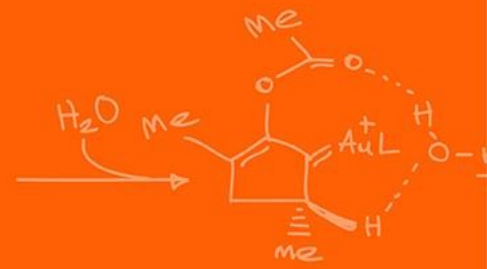
```
void GPIO_SetupPinMux(Uint16 gpioNumber, Uint16 cpu, Uint16 muxPosition)
{
    volatile Uint32 *gpioBaseAddr; // volatile means that the variable may change from external sources (i.e.
interrupts)
    volatile Uint32 *mux, *gmux, *csel;
    Uint16 pin32, pin16, pin8;

    pin32 = gpioNumber % 32;
    pin16 = gpioNumber % 16;
    pin8 = gpioNumber % 8;
    gpioBaseAddr = (Uint32 *)&GpioCtrlRegs + (gpioNumber/32)*GPY_CTRL_OFFSET;

    mux = gpioBaseAddr + GPYMUX + pin32/16;
    gmux = gpioBaseAddr + GPYGMUX + pin32/16;
    csel = gpioBaseAddr + GPYCSEL + pin32/8;

    EALLOW; // block that controls the physical multiplexer

    // 0x3UL means 0b11 but in int32_t (i.e. 0b11 with 30 leading 0s)
    *mux &= ~(0x3UL << (2*pin16));
    *gmux &= ~(0x3UL << (2*pin16));
    *gmux |= (Uint32)((muxPosition >> 2) & 0x3UL) << (2*pin16);
    *mux |= (Uint32)(muxPosition & 0x3UL) << (2*pin16);

    *csel &= ~(0x3L << (4*pin8));
    *csel |= (Uint32)(cpu & 0x3L) << (4*pin8);
    EDIS;
}
```

Common PinMux mistake

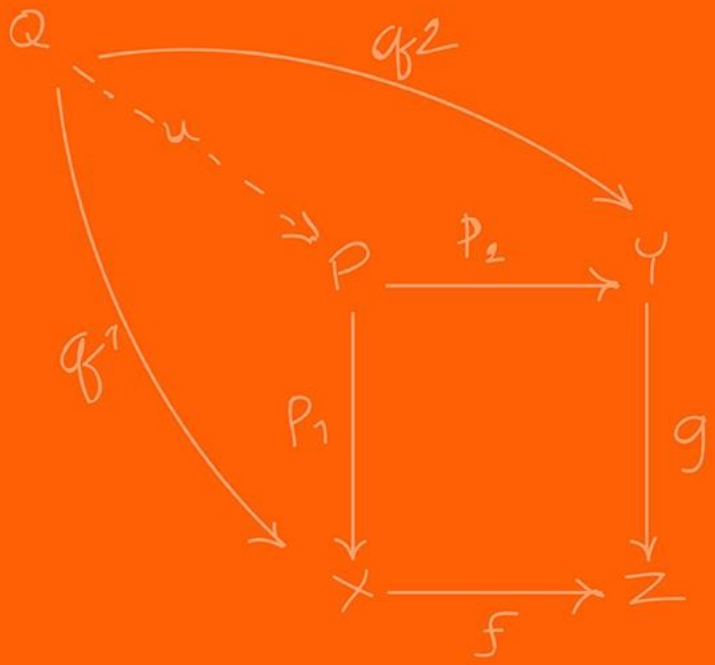
```
GPIO_SetupPinMux(19, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(19, GPIO_OUTPUT, GPIO_PUSHPULL);
```

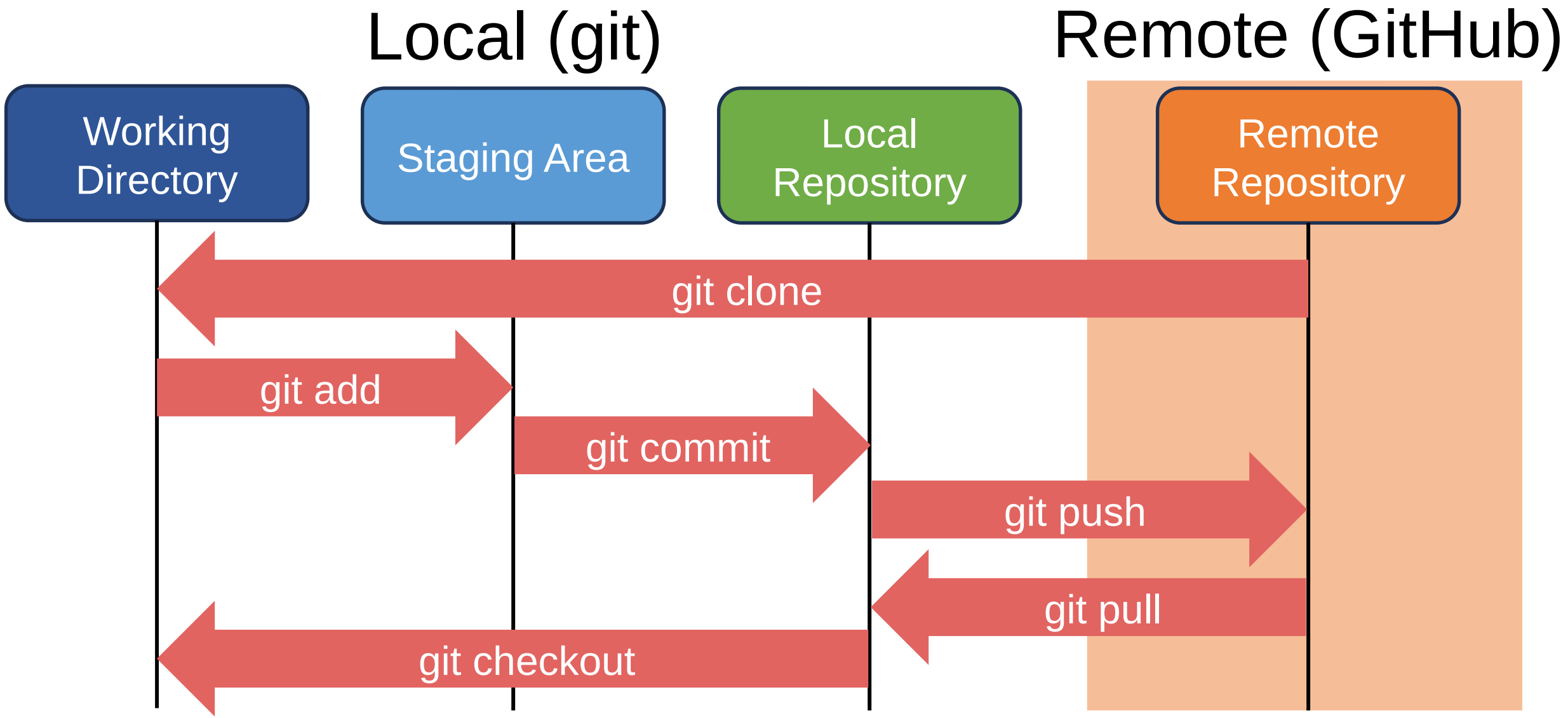
- Using the pin number instead of the GPIO number
- Using the wrong GPIO for the pin that you want
- Direction is not set correctly
- Pull-up disabled unintentionally
- Make sure to read the documentation



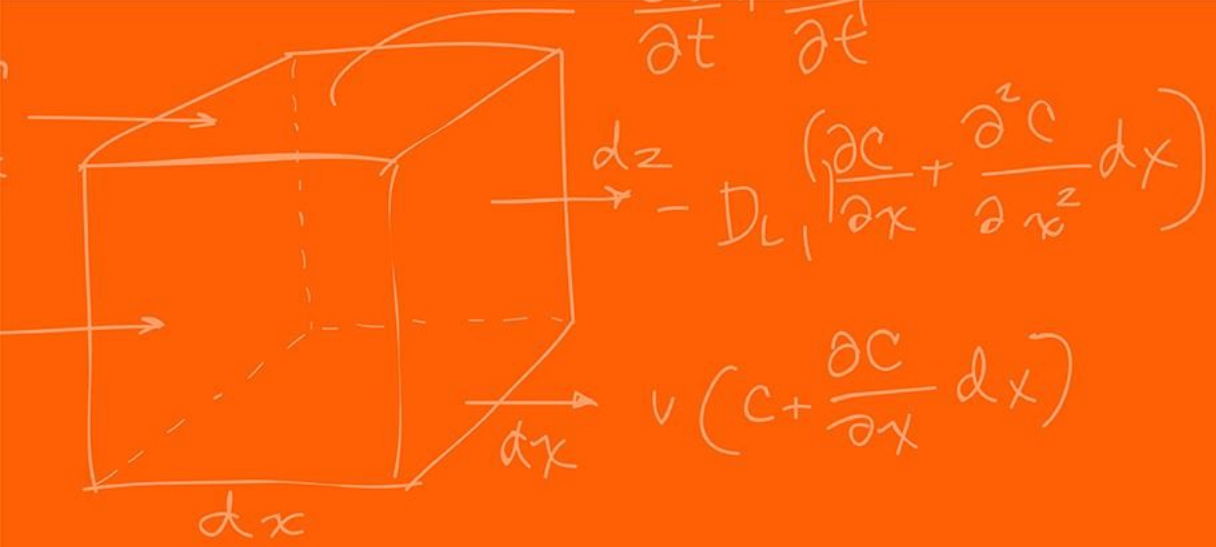


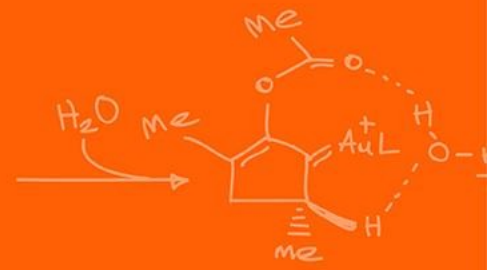
Git / GitHub




Local (git)
Remote (GitHub)
Working Directory
Staging Area
Local Repository
Remote Repository
git clone
git add
git commit
git push
git pull
git checkout

Questions?





The Grainger College of Engineering

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN