

Input & Output and Communications

01266212

CYBER PHYSICAL SYSTEM DESIGN

SEMESTER 1-2022

Original contents from
Edward A. Lee and Prabal Dutta, UC Berkeley, EECS 149/249A

Contents

- Input and Output
- Interrupt
- Communications

Input & Output

Hardware CPS Design

- In the physical world, many things happen at once, but software is mostly sequential.
- Reconciling these two disparate properties is a major challenge and is often the biggest risk factor in the design of embedded systems.
- A system designer has to confront a number of issues, particularly the mechanical and electrical properties of the **interfaces**.
- Incorrect use of parts, such as drawing too much current from a pin, may cause a system to malfunction or may reduce its useful lifetime.

Connecting the Analog and Digital Worlds

Semantic mismatch:

Cyber:

- Digital
- Discrete in time
- Sequential

Physical:

- Continuum
- Continuous in time
- Concurrent

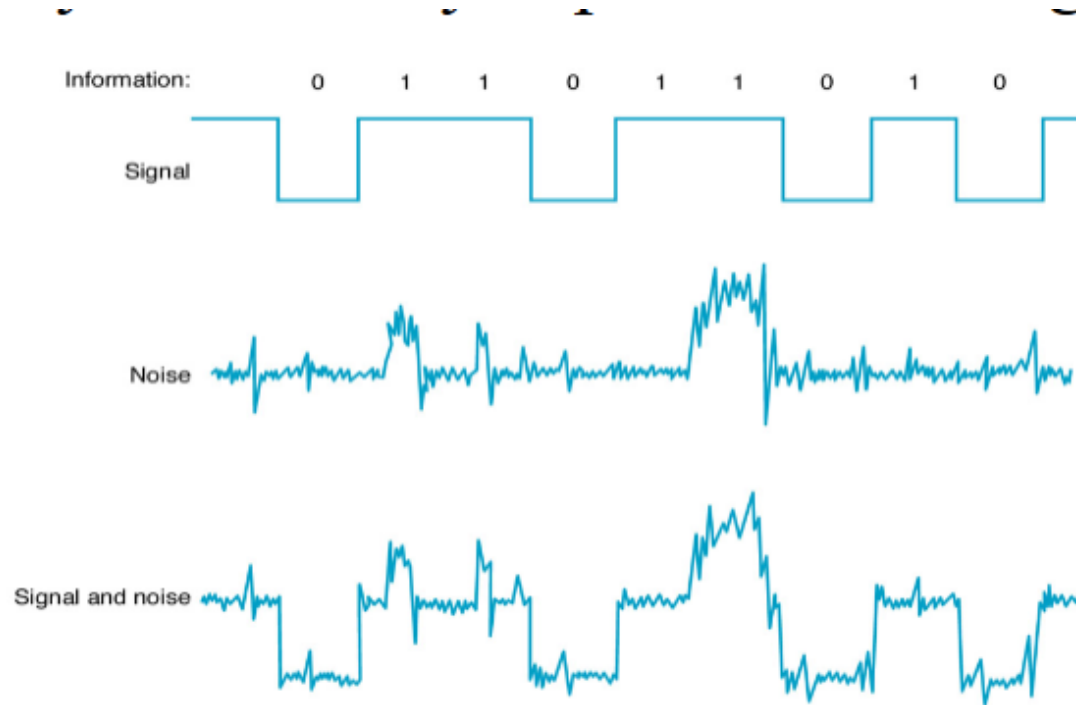
Practical Issues

- ✓ Analog vs. digital
- ✓ Wired vs. wireless
- ✓ Serial vs. parallel
- ✓ Sampled or event triggered
- ✓ Bit rates (Baud rate)
- ✓ Access control, security, authentication
- ✓ Physical connectors
- ✓ Electrical requirements (voltages and currents)

Noise

Noise is the part of a signal that we do not want.

If we want to measure $x(t)$ at time t , but we actually measure $x'(t) = x(t) + n(t)$ where $n(t)$ is the noise.



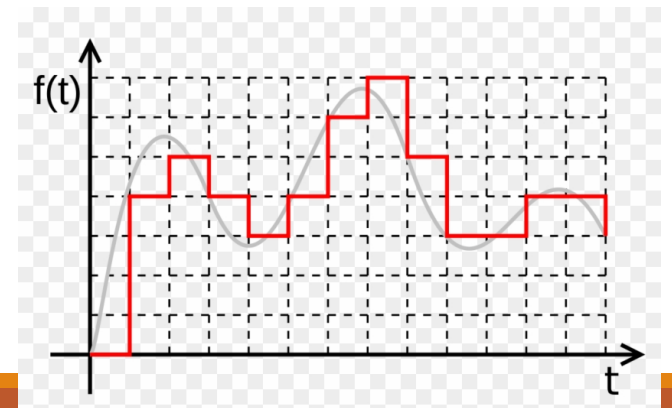
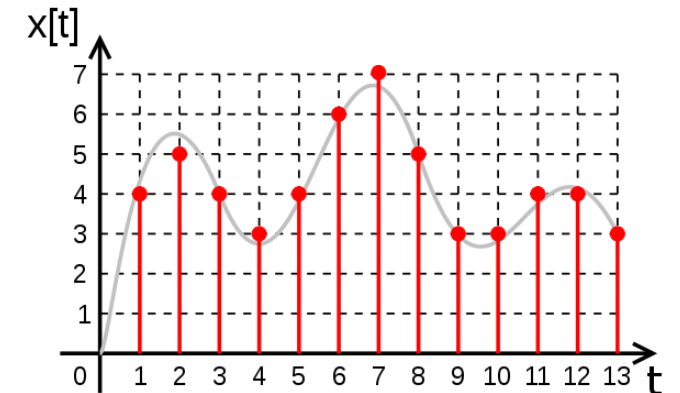
Sampling

A physical quantity $x(t)$ is a function of time t . A digital sensor will sample the physical quantity at particular points in time to create a discrete signal.

In uniform sampling, there is a fixed time interval T between samples; T is called the sampling interval.

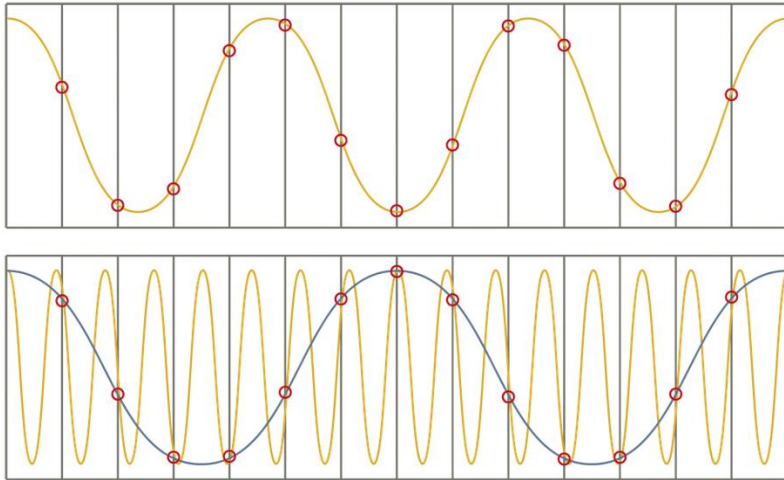
The resulting signal may be modeled as a function defined as follows,

The physical quantity $\forall n \in \mathbb{Z}, s(n) = f(x(nT))$, $t = nT$, and the measurement is subjected to the sensor distortion function. The sampling rate is $F = 1/T$, which has units of samples per second.

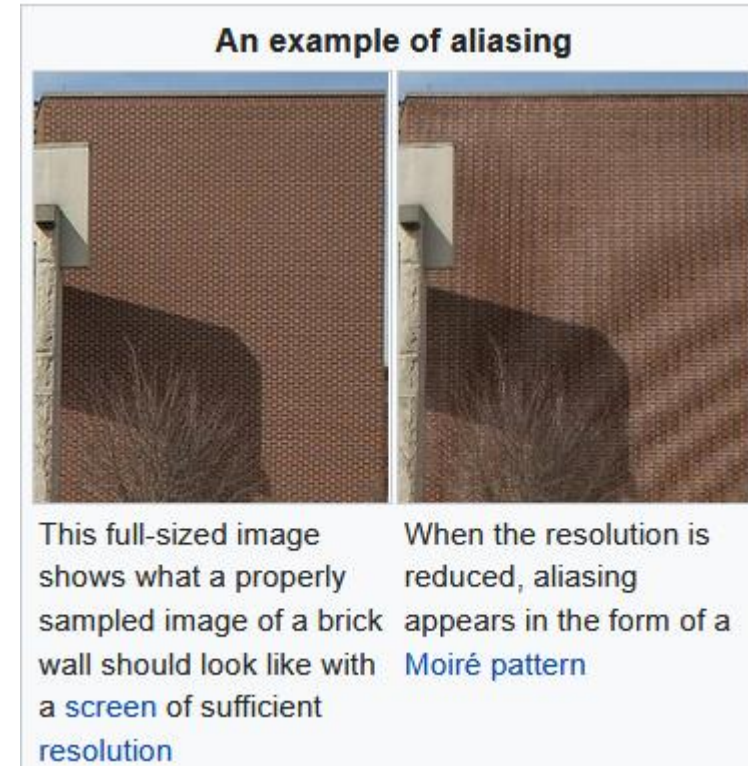


Aliasing

Aliasing occurs if there are signals present that are greater than half the frequency of the sampling frequency.



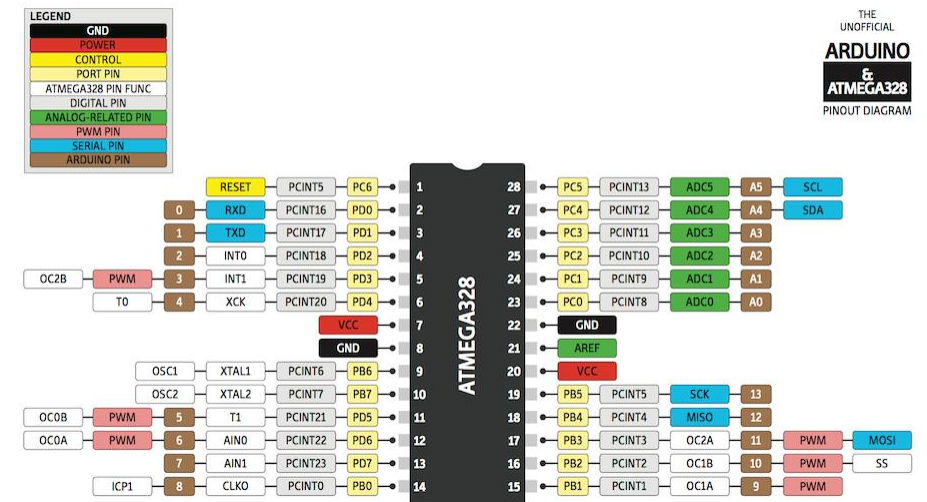
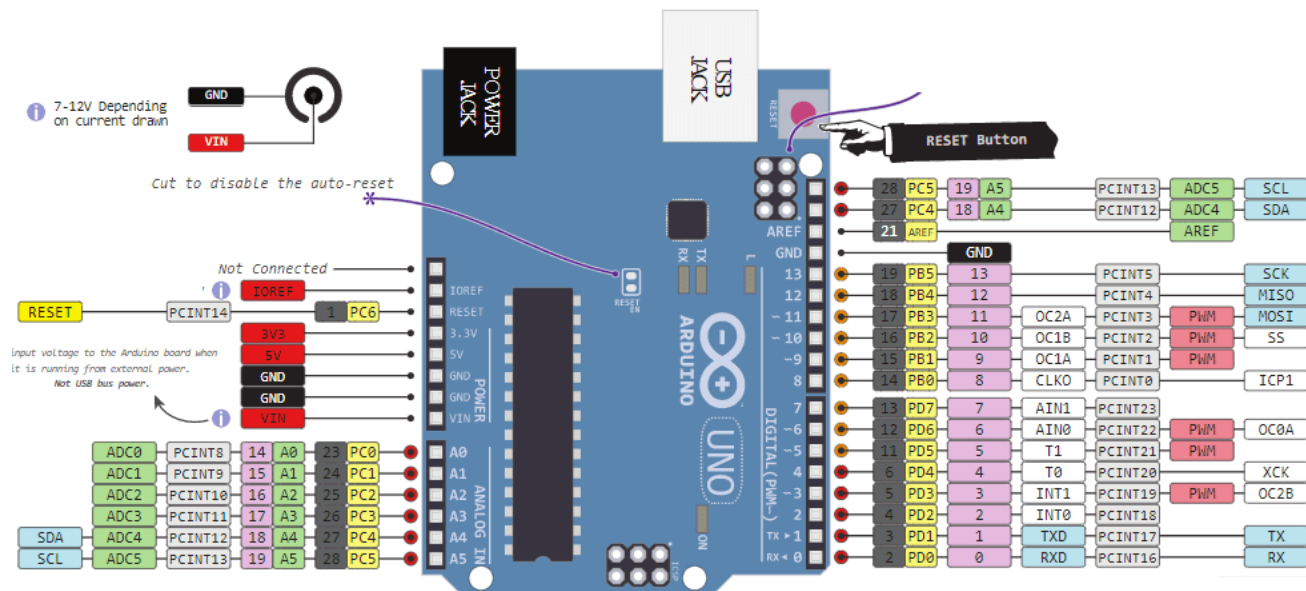
Source: <https://itectec.com/electrical/>



Source: wikipedia.org/

Input Output Hardware

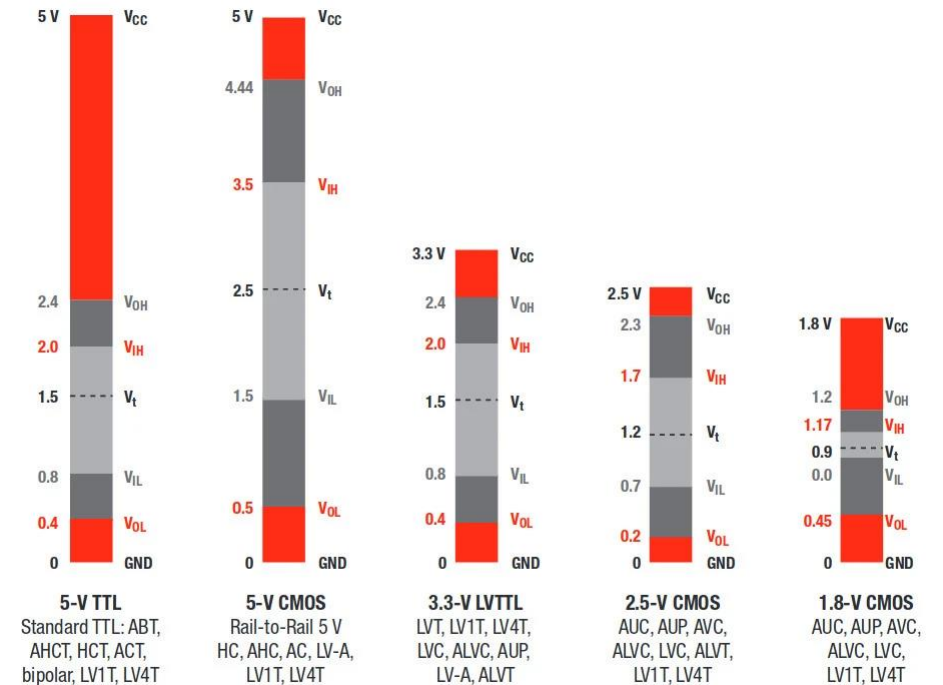
- Embedded processors typically include a number of input and output (I/O) mechanisms on chip, exposed to designers as pins of the chip.



General-Purpose Digital I/O

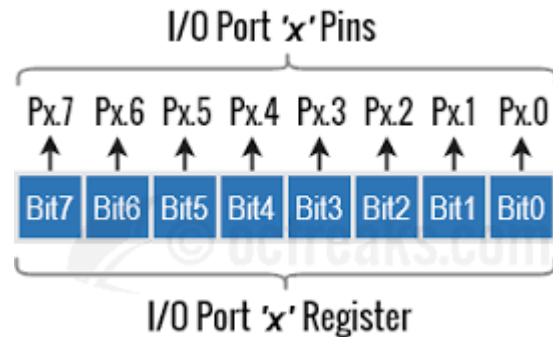
Many embedded processors have a number of **general-purpose I/O** pins (**GPIO**), which enable the software to either read or write voltage levels representing a logical zero or one.

If the processor **supply voltage** is V_{CC} or V_{DD} , in **active high logic** a voltage close to V_{CC} or V_{DD} represents a logical one, and a voltage near zero represents a logical zero. In **active low logic**, these interpretations are reversed.



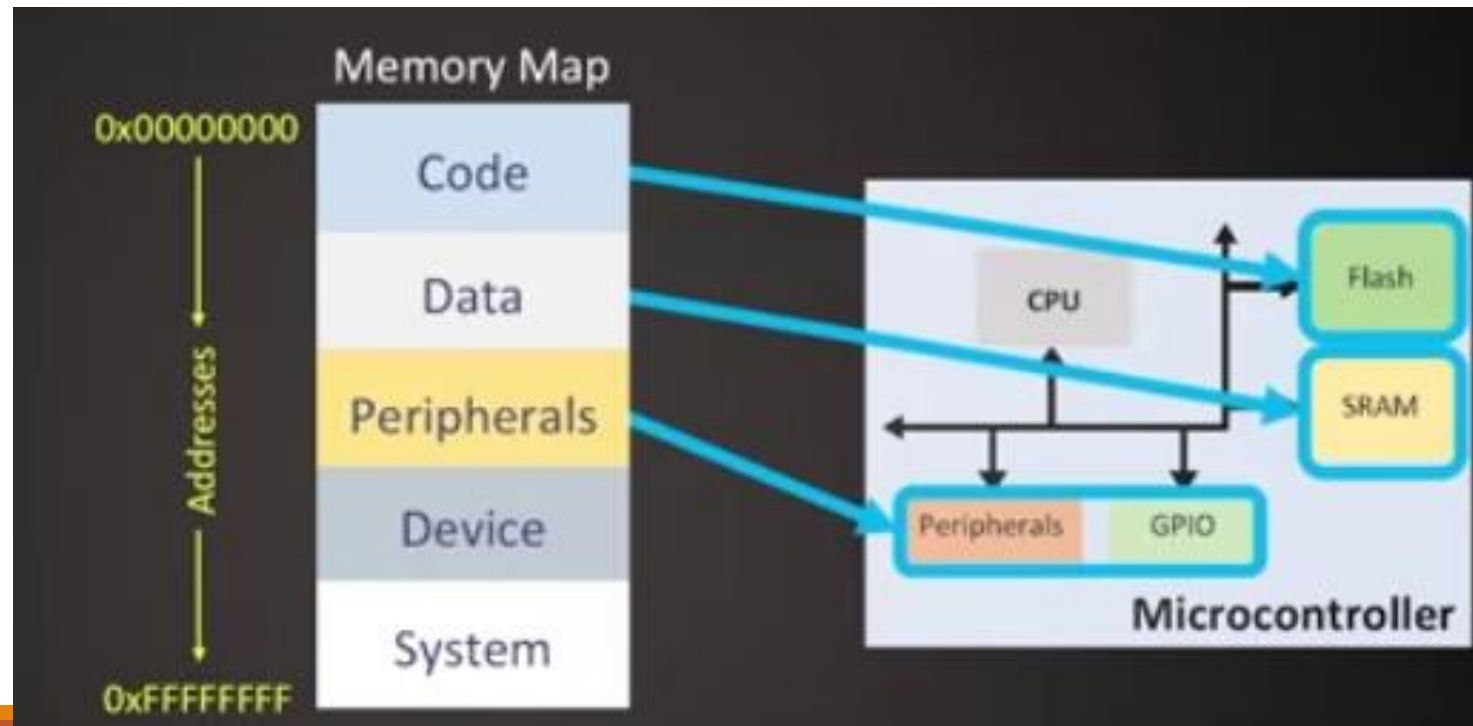
General-Purpose Digital I/O

In many designs, a GPIO pin may be configured to be an output. This enables software to then write to a [memory-mapped register](#) to set the output voltage to be either high or low. Software can directly control external physical devices.



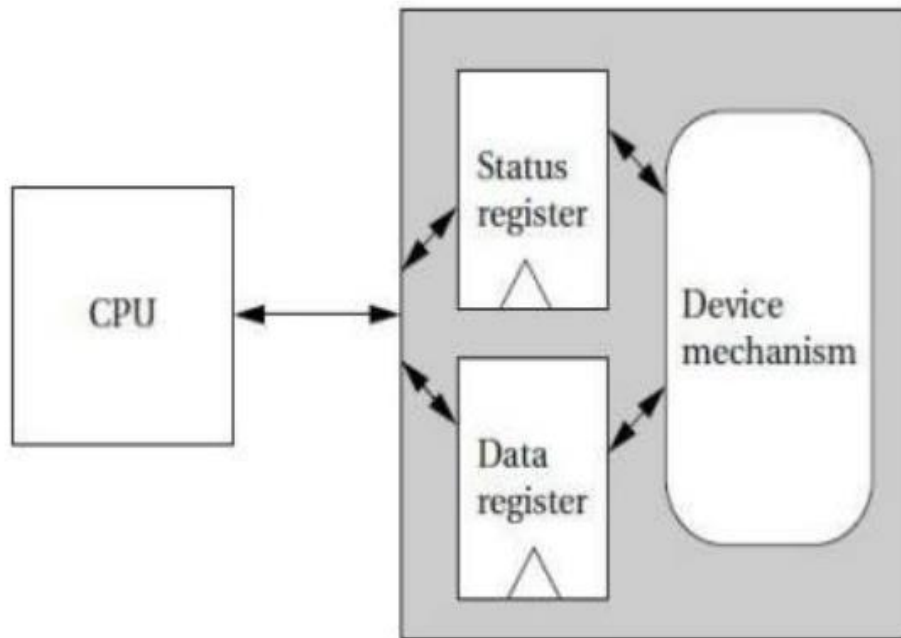
Memory Map I/O

- Provides a memory address to physical device mapping withing an address space for use in programming
- Mapped Components (Memory, Peripherals, Systems Config, etc.)



Input Output Hardware

- ❑ Embedded processors, CPU communicates to the respective register of I/O, as part of the memory, which is connected to I/O pin through I/O mechanism.



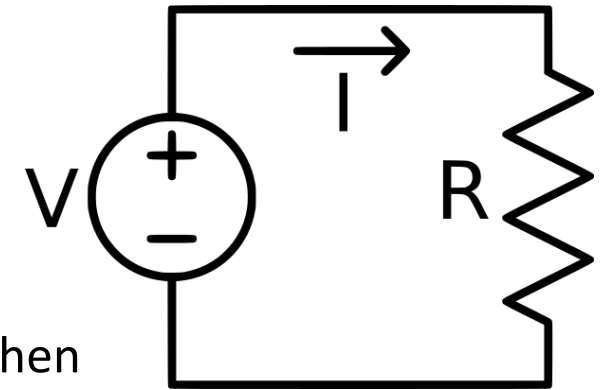
General-Purpose Digital I/O

When interfacing hardware to GPIO pins, a designer needs to understand the specifications of the device which are various voltage and current levels.

If a GPIO pin produces an output voltage of V_{DD}/V_{CC} when given a logical one, then the designer needs to know the current limitations before connecting a device to it.

If a device with a resistance of R ohms is connected to it, for example, then **Ohm's law** tells us that the output current will be

$$I = V_{DD}/R$$



It is essential to keep this current within specified tolerances. Going outside these tolerances could cause the device to overheat and fail. A power amplifier may be needed to deliver adequate current. An amplifier may also be needed to change voltage levels.

Atmel AVR Atmega328 I/O Pins

28.1 Absolute Maximum Ratings

Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Parameters	Min.	Typ.	Max.	Unit
Operating temperature	−55		+125	°C
Storage temperature	−65		+150	°C
Voltage on any pin except $\overline{\text{RESET}}$ with respect to ground	−0.5		$V_{CC} + 0.5$	V
Voltage on $\overline{\text{RESET}}$ with respect to ground	−0.5		+13.0	V
Maximum operating voltage		6.0		V
DC current per I/O pin		40.0		mA
DC current V_{CC} and GND pins		200.0		mA
Injection current at $V_{CC} = 0V$		$\pm 5.0^{(1)}$		mA
Injection current at $V_{CC} = 5V$		± 1.0		mA

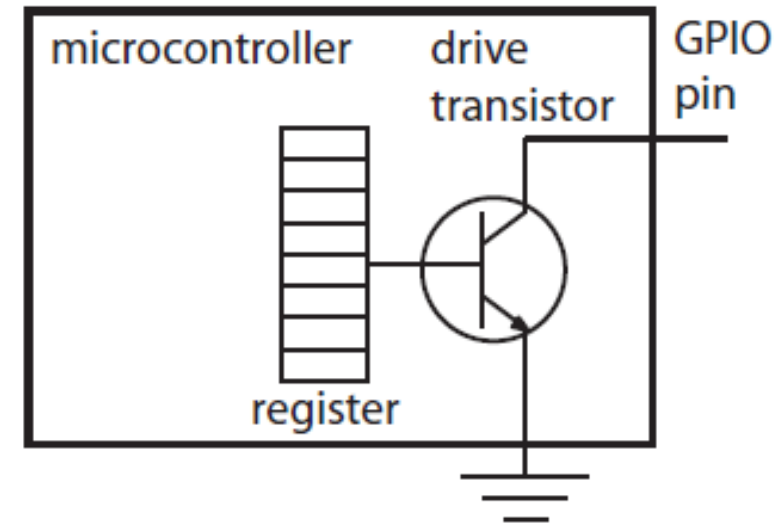
Note: 1. Maximum current per port = $\pm 30\text{mA}$

Simple Digital I/O: GPIO

Open collector circuits are often used on GPIO (general-purpose I/O) pins of a microcontroller.

Writing a logical one into the (memory mapped) register turns on the transistor, which pulls the voltage on the output pin down to (near) zero.

Writing a logical zero into the register turns off the transistor, which leaves the output pin unconnected, or “open.”



Share a Single Connection

- In many applications, several devices may share a single electrical connection.
- The designer must take care to ensure that these devices do not simultaneously drive the voltage of this single electrical connection to different values, resulting in a short circuit that can cause overheating and device failure.
- For example, all of these GPIO lines are wired together to a single control input of the piece of machinery.
 - If one microcontroller attempts to drive the shared line to a high voltage while another attempts to drive the same line to a low voltage.

Simple Digital I/O: GPIO

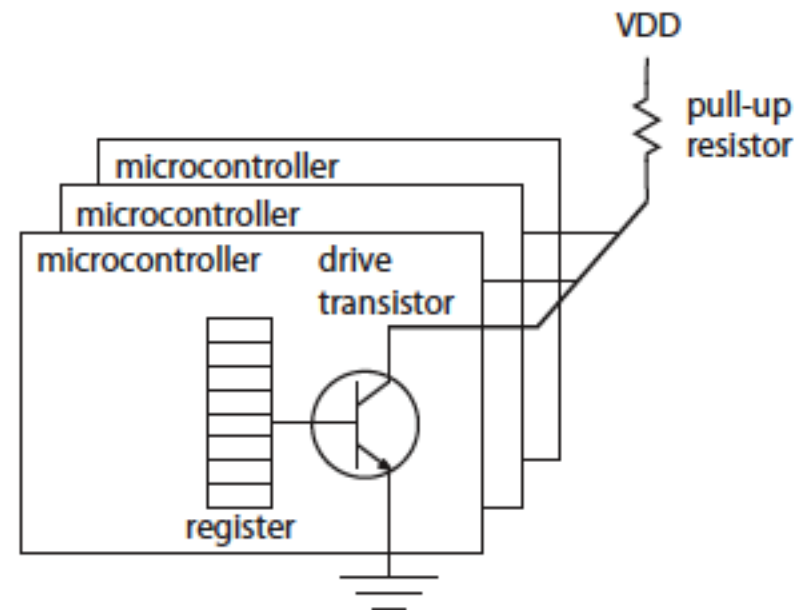
The shared line is connected to a **pull-up resistor**, which brings the voltage of the line up to V_{DD} when all the transistors are turned off.

If any one transistor is turned on, then it will bring the voltage of the entire line down to (near) zero without creating a short circuit with the other GPIO pins.

Logically, all registers must have zeros in them for the output to be high.

If any one of the registers has a one in it, then the output will be low.

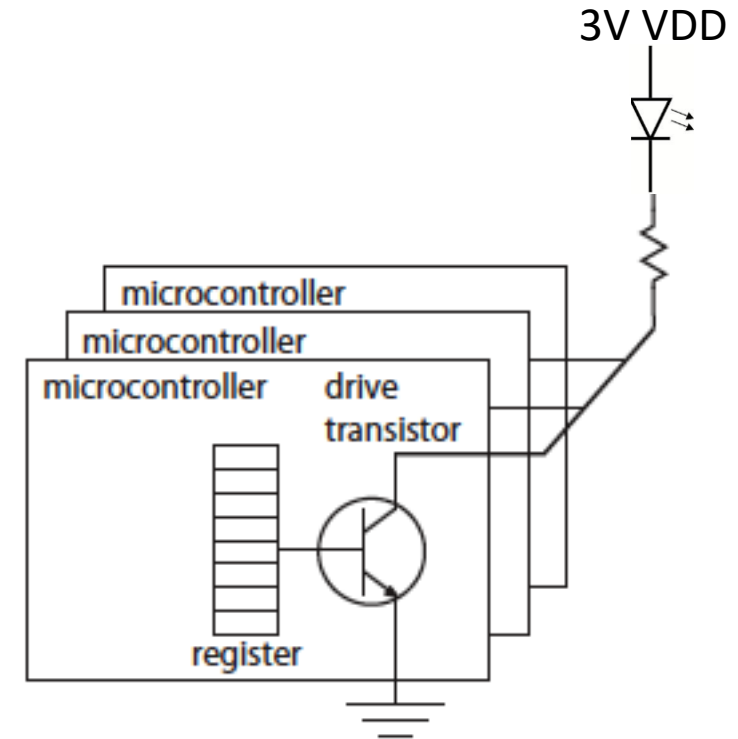
GPIO pins configured for bus output. Any one controller can pull the bus voltage down.



Example: Turn on an LED

Assume GPIO pins can sink up **to 18 mA**. Assume the LED, when forward biased (turned on), has a voltage drop of 2 volts.

What resistor should you use?



Example: Turn on an LED

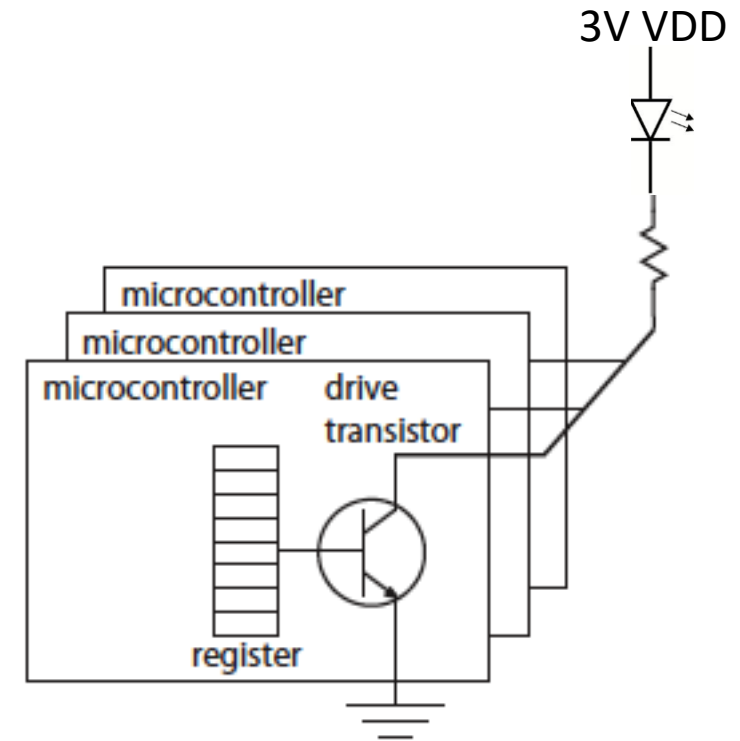
Ohm's law:

$$V = IR$$

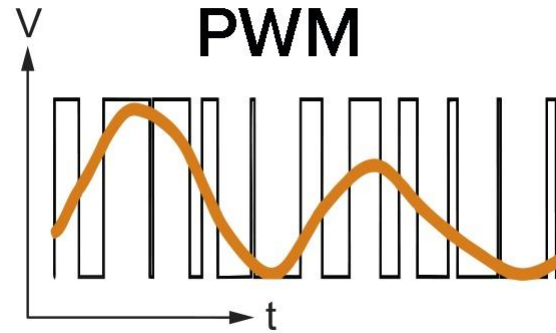
When LED is on, $V = 1$ volt.

To limit to 18mA,

$$R \geq 1/0.018 \approx 56 \text{ ohms}$$



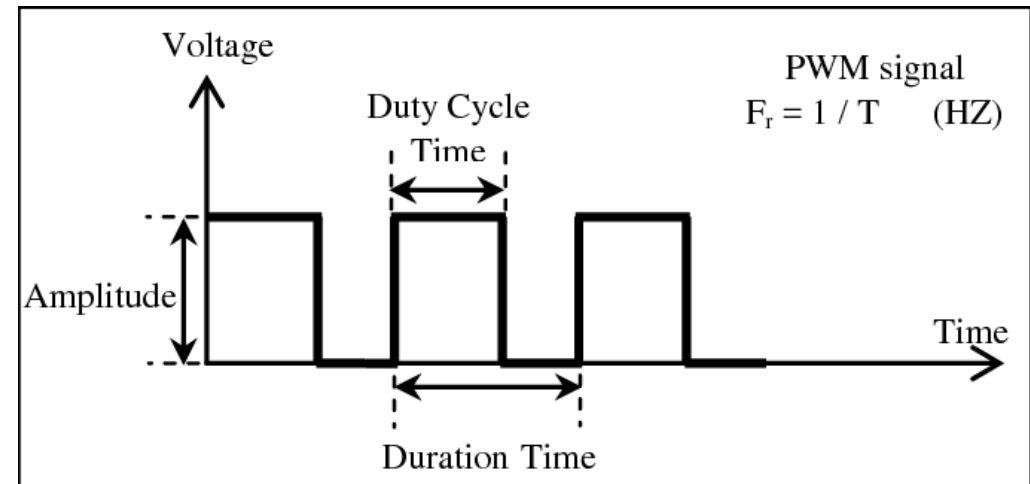
Pulse Width Modulation



- **Pulse width modulation** (PWM) is a technique for delivering a variable amount of power efficiently to external hardware devices. (Analog Output with Digital Signal)
- It can be used to control for example the speed of electric motors, the brightness of an LED light, and the temperature of a heating element.
- It can deliver varying amounts of power to devices that tolerate rapid and abrupt changes in voltage and current. Any device whose response to changes in current or voltage is slow compared to the frequency of the PWM signal is a candidate for being controlled via PWM.

Pulse Width Modulation

- A PWM signal rapidly switches between high and low at some fixed frequency, varying the amount of time that it holds the signal high
- The **duty cycle** is the proportion of time that the voltage is high.
- If the duty cycle is 100%, then the voltage is always high.
- If the duty cycle is 0%, then the voltage is always low.
- To use these, a programmer typically writes a value to a **memory-mapped register** to set the duty cycle (the frequency may also be settable).

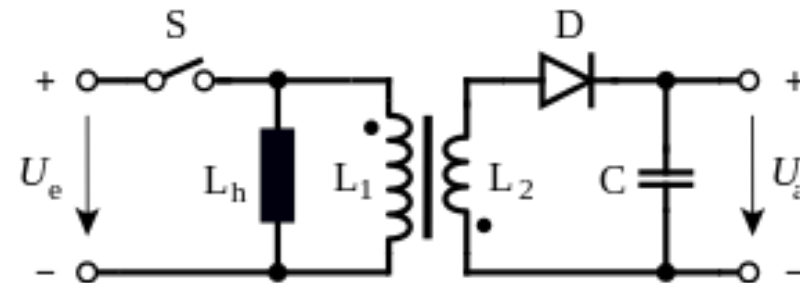
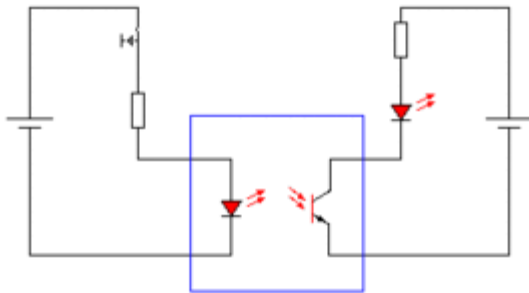


Pulse Width Modulation



Electrical Isolation

- The external devices may have messy (noisy) electrical characteristics that will make the processor unreliable if the noise spills over into the power or ground lines of the processor.
- Or the external device may operate in a very different voltage or power regime compared to the processor.
- A useful strategy is to divide a circuit into electrical domains, possibly with separate power supplies, that have relatively little influence on one another.
- Isolation devices such as opto-isolators and transformers may be used to enable communication across electrical domains.



Input/Output Mechanisms in Software

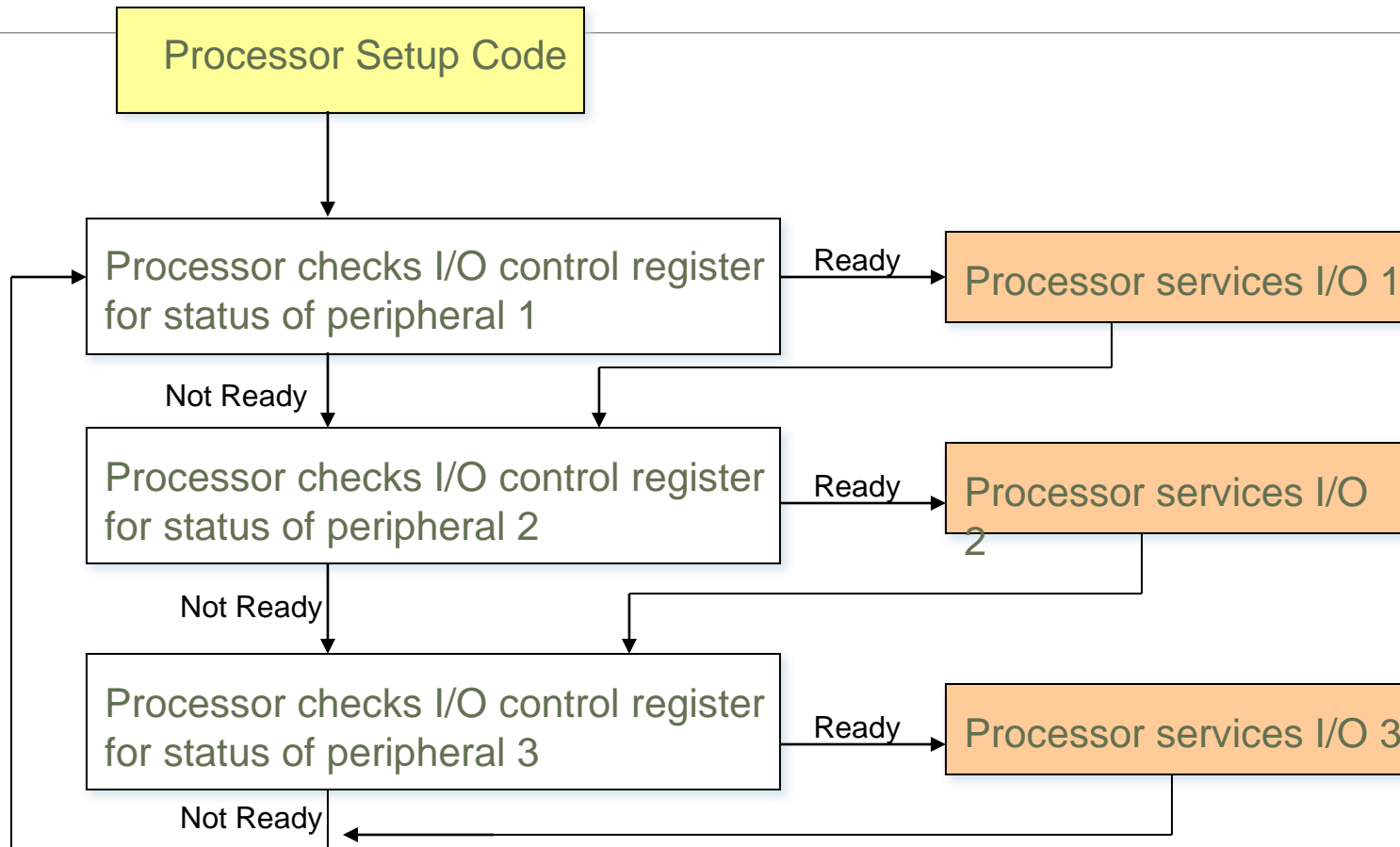
○ Polling

- Main loop uses each I/O device periodically.
- If output is to be produced, produce it.
- If input is ready, read it.

○ Interrupts

- External hardware alerts the processor that input is ready.
- Processor suspends what it is doing.
- Processor invokes an interrupt service routine (ISR).
- ISR interacts with the application concurrently.

Polling



Example Using a Serial Interface

In an Atmel AVR 8-bit microcontroller, to send a byte over a serial port, the following C code will do:

```
while(!(UCSR0A & 0x20));  
UDR0 = x;
```

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- x is a variable of type uint8.
- UCSR0A and UDR0 are variables defined in a header.
- They refer to memory-mapped registers in the UART (Universal Asynchronous Receiver-Transmitter)
- <https://maxembedded.com/2013/09/the-usart-of-the-avr/#UCSRA>

Send a Sequence of Bytes

```
for(i = 0; i < 8; i++) {  
    while(!(UCSR0A & 0x20));  
    UDR0 = x[i];  
}
```

How long will this take to execute? Assume:

- 57600 baud serial speed.
- $8/57600 = 139$ microseconds.
- Processor operates at 18 MHz.

Each for loop iteration will consume about 2502 cycles.

Receiving via UART

Again, on an Atmel AVR:

```
while (! (UCSR0A & 0x80)) ;  
return UDR0 ;
```

- Wait until the UART has received an incoming byte.
- *The programmer must ensure there will be one!*
- If reading a sequence of bytes, how long will this take?

Under the same assumptions as before, it will take about 2502 cycles to receive each byte.

Input Mechanisms in Software

○ Polling

- Main loop uses each I/O device periodically.
- If output is to be produced, produce it.
- If input is ready, read it.

○ Interrupts

- External hardware alerts the processor that input is ready.
- Processor suspends what it is doing.
- Processor invokes an interrupt service routine (ISR).
- ISR interacts with the application concurrently.

Interrupts and Exceptions

An **interrupt** is a mechanism for pausing execution of whatever a processor is currently doing and executing a pre-defined code sequence called an **interrupt service routine (ISR) or interrupt handler**.

Three kinds of events may trigger an interrupt.

1. A **hardware interrupt**: where some external hardware changes the voltage level on an interrupt request line.
2. A **software interrupt**: the program that is executing triggers the interrupt by executing a special instruction or by writing to a memory-mapped register.
3. **Exception**: where the interrupt is triggered by internal hardware that detects a fault, such as a segmentation fault.

Interrupts Arduino

Example Code

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

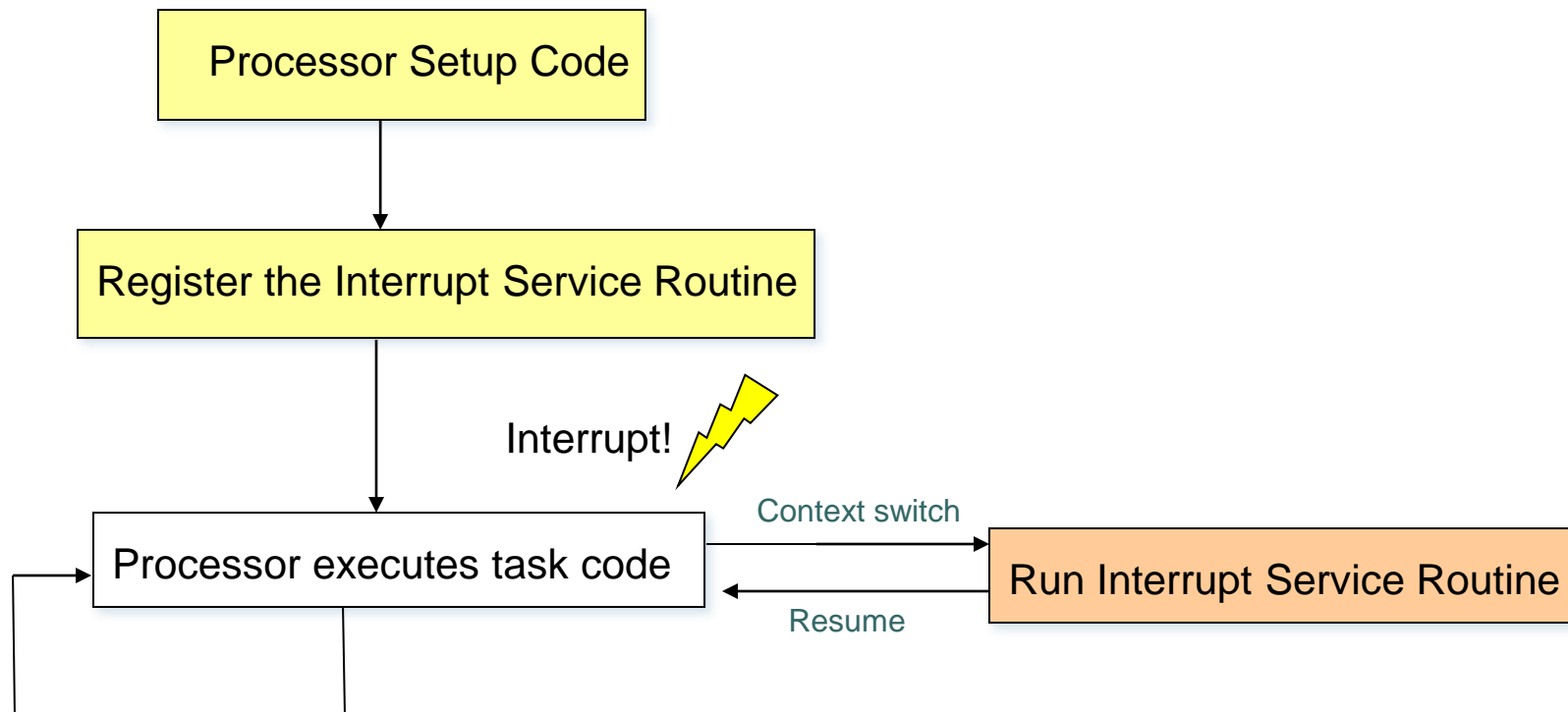
void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

Interrupts

○ Interrupt Service Routine

Short subroutine that handles the interrupt



Interrupts

- Upon occurrence of an interrupt trigger, the hardware must first decide whether to respond.
- If interrupts are disabled, it will not respond.
- The mechanism for enabling or disabling interrupts varies by processor. (some interrupts are enabled and others are not.)
- Interrupts and exceptions generally have priorities, and an interrupt will be serviced only if the processor is not already in the middle of servicing an interrupt with a higher priority.
- Exceptions have the highest priority and are always serviced.

Issues to Watch For

- Interrupt service routine execution time
- Context switch time
- Nesting of higher priority interrupts
- Interactions between ISR and the application
- Interactions between ISRs
- ...

Communications

Networking and Communications

Goal of lecture

- Want you to be aware of communication options and choices

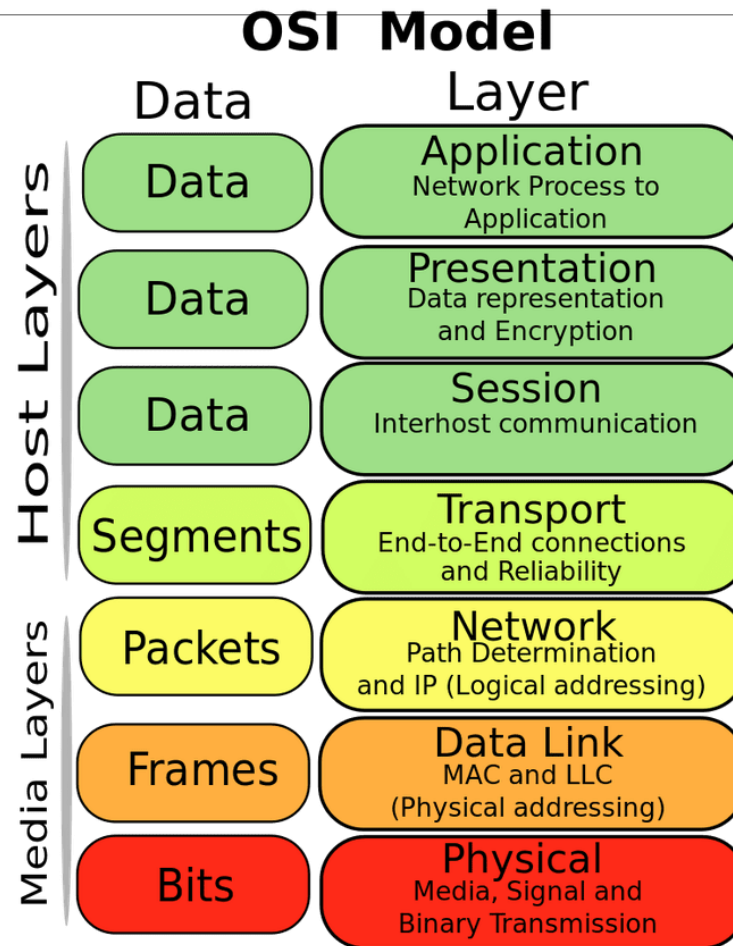
Outline

- Communications model
- Wired networks
- Wireless networks

The Alphabet Soup

- | | | |
|-------------|--------|--------------|
| • 1588 | • GSM | • REST |
| • 6LoWPAN | • HART | • TDMA |
| • 802.15.4 | • HTTP | • TSMP |
| • 802.1(AS) | • IoT | • TSN |
| • 802.11 | • IPv6 | • TTEthernet |
| • AVB | • LTE | • TTP |
| • BLE | • MAC | • WAN |
| • CAN | • PAN | • WLAN |
| • CoAP | • PTP | • WPAN |
| • CSMA/CA | • QoS | |

Communications Layers



Physical Layer Technologies

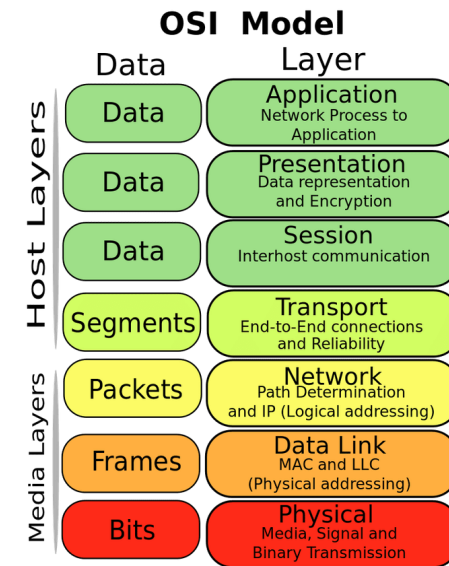
Specifies electrical characteristics

- Voltages
- Frequencies

Specifies how to map signals to data

- Example low voltage = 0 and high voltage = 1
- Example oscillation at a high frequency = 0, low = 1

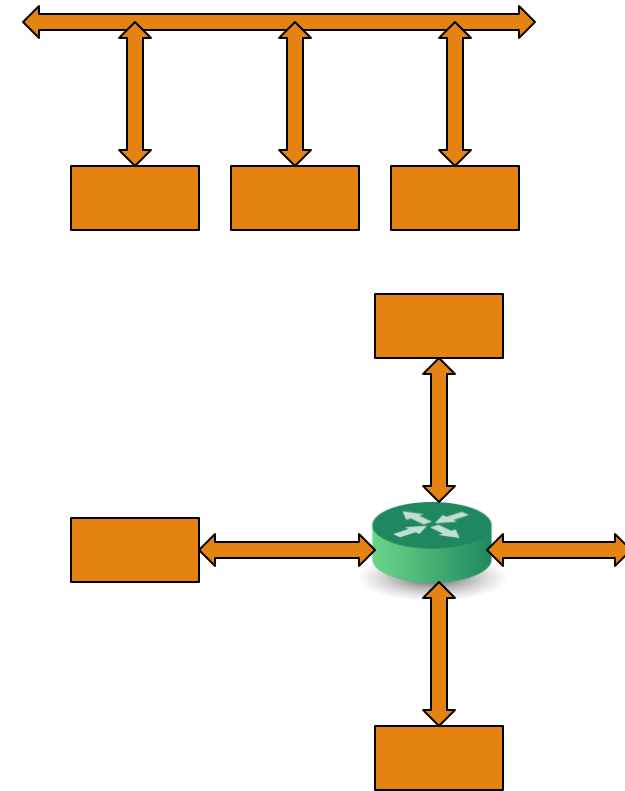
Specifies network topology as well



Network Topology

- Busses
 - Shared physical medium
 - MAC protocol dominates
- Star networks
 - Private medium
 - MAC protocol is less important
 - Routing protocols become important
 - Buffers in routers

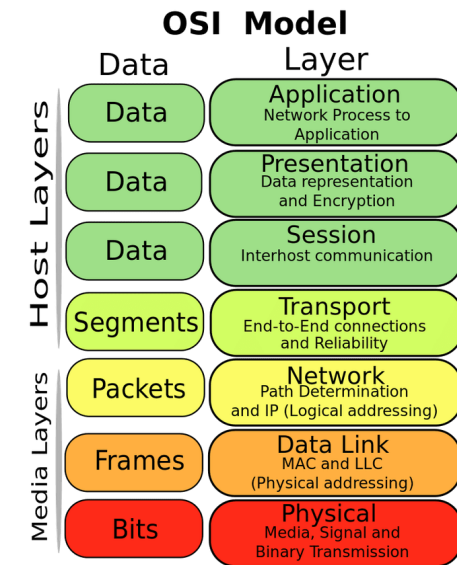
Is radio a bus?



Medium Access Controls

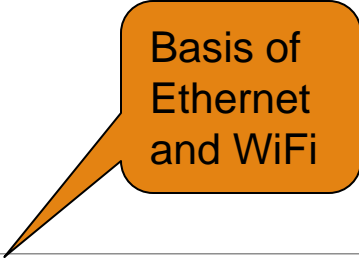
How do multiple devices share the same transmission technology?

- Don't worry about it, let collisions happen
- Listen for others and don't transmit if they are
- Coordinate with others and transmit at different times
- Transmit at different frequencies



MAC: Media Access Control

CSMA/CA vs. Time Slotted



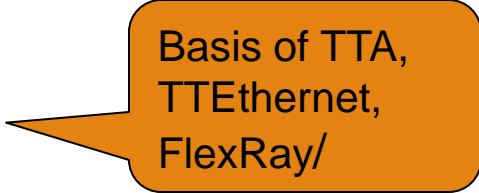
Basis of
Ethernet
and WiFi

Carrier Sense Multiple Access / Collision Avoidance

- Listen for idle channel
- Send
- Wait for ack, retransmit if no ack after some timeout

Time Division Multiple Access (TDMA)

- Wait your turn
- Send when it's your turn
- Add various schemes to recover unused slots
- Maybe add slots for CSMA/CA



Basis of TTA,
TTEthernet,
FlexRay/

MAC: Media Access Control

FDMA

Frequency Division Multiple Access

- Protocol supports multiple “channels” each at a different frequency
- Send on a specific channel to not conflict with others

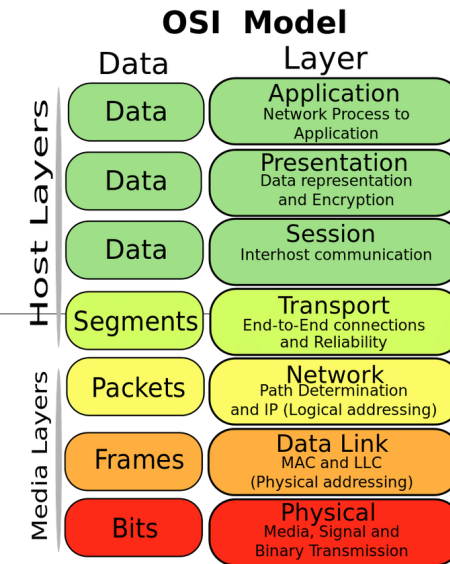
There are many other methods of sharing as well

Network Layer

How do I get a message to the correct device?

- How devices are named (addressed)?
- How are messages routed?

Example: IP

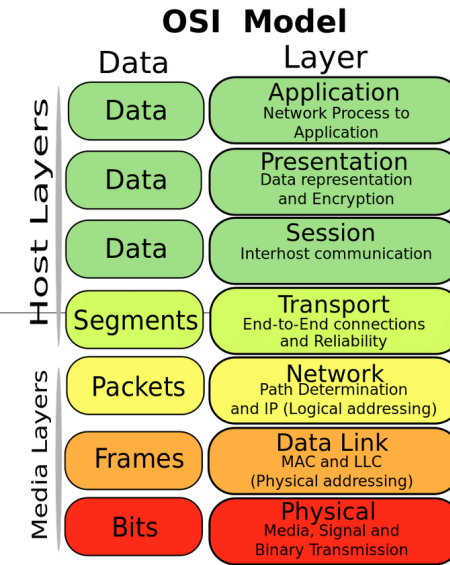


Transport Layer Reliability

How do I reliably get messages between two devices?

- What if the message is too big for one packet?
- How do I know if the recipient got the message?

Example: TCP and UDP

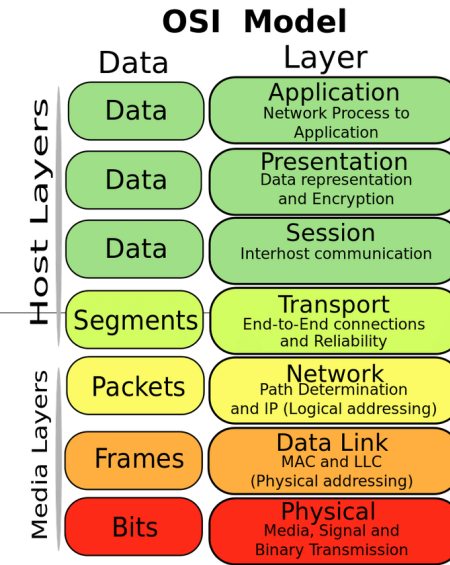


Higher Layers

How do we handle data?

Includes:

- maintaining connections across computers
- deciding what messages to send
- data compression and encryption



Wired Networks

I²C, UART, and SPI

Why aren't these good enough for everything?

UART

- Slow. No shared bus.

I²C

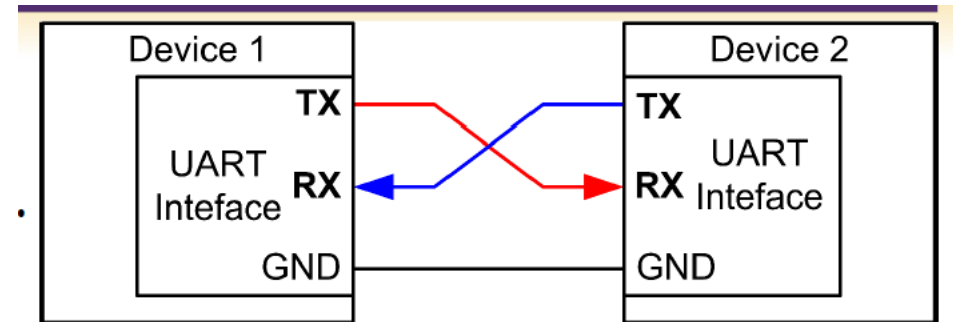
- Slow. Master-initiated communication. Short distance.

SPI

- Master-initiated communication. Lots of pins.

Universal Asynchronous Receiver and Transmitter (UART)

- Universal
 - Programmable format, speed, etc.
- Asynchronous
 - Sender provides no clock signal to receivers
- Half Duplex
- Any node can initiate communication
- Two lanes are independent of each other



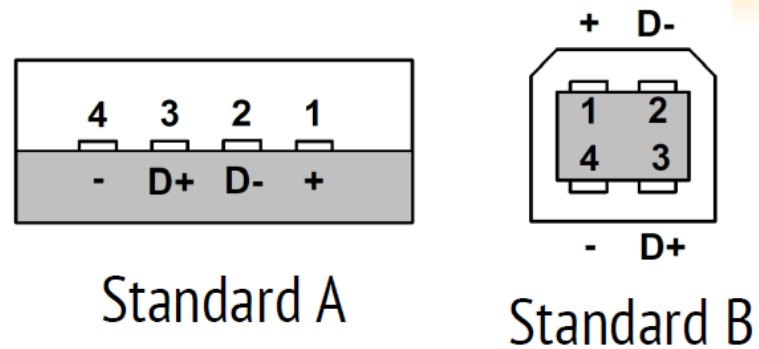
Baud Rate

- Historically used in telecommunication to represent the number of pulses physically transferred per second
- In digital communication, baud rate is the number of bits physically transferred per second

Example:

- Baud rate is 9600
- each frame: a start bit, 8 data bits, a stop bit, and no parity bit.
- Transmission rate of actual data
 - $9600/8 = 1200$ bytes/second
 - $9600/(1 + 8 + 1) = 960$ bytes/second
- The start and stop bits are the protocol overhead

USB Layers

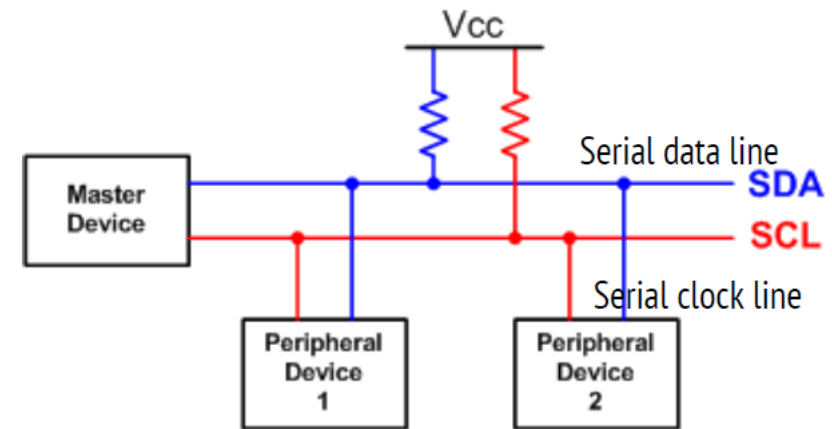


- Four shielded wires: two for power (+5V, ground), two for data (D+, D-)
- D+ and D- are twisted to cancel external electromagnetic interference



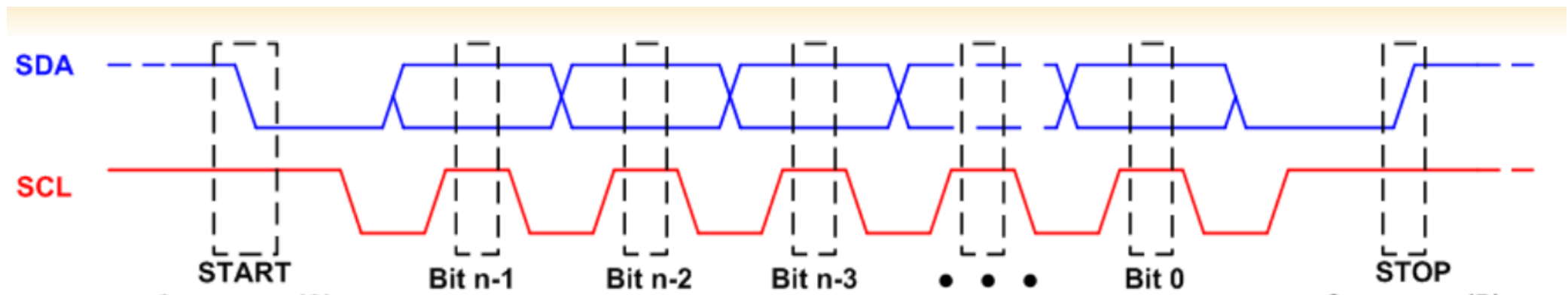
Inter-Integrated Circuit (I2C)

- A master device, such as the Rpi/Arduino, controls the bus, and many addressable slave devices can be attached to the same two wires.
- Up to 100 kbit/s in the standard mode, up to 400 kbit/s in the fast mode, and up to 3.4 Mbit/s in the high-speed mode.
- SDA and SCL have to be open-drain
 - Connected to positive if the output is 1
 - In high impedance state if the output is 0
- Each Device has a unique address (7, 10 or 16 bits). Address 0 used for broadcast



Inter-Integrated Circuit (I2C) Timming

- A **START** condition is a high-to-low transition on SDA when SCL is high.
- A **STOP** condition is a low to high transition on SDA when SCL is high.
- The address and the data bytes are sent most significant bit first.
- Master generates the clock signal and sends it to the slave during data transfer



The data line can not change when the clock line is high, it can change only when the clock line is low.

Example: Write 1 byte to device register

- **Master** sends a *start bit* (i.e., it pulls SDA low, while SCL is high).
- While the clock toggles, the 7-bit slave address is transmitted one bit at a time.
- A read bit (1) or write bit (0) is sent, depending on whether the master wants to read or write to/from a slave register.
- The slave responds with an *acknowledge* bit (ACK = 0).
- In write mode, the master sends a byte of data one bit at a time, after which the slave sends back an ACK bit. To write to a register, the register address is sent, followed by the data value to be written.
- Finally, to conclude communication, the master sends a stop bit (i.e., it allows SDA to float high, while SCL is high).

Working Modes

➤ Master-sender

- Master issues START and ADDRESS, and then transmits data to the addressed slave device

➤ Master-receiver

- Master issues START and ADDRESS, and receives data from the addressed slave device

➤ Slave-sender

- Master issues START and the ADDRESS of the slave, and then the slave sends data to the master

➤ Slave-receiver

- Master issues START and the ADDRESS of the slave, and then the slave receives data from the master.

Pros and Cons

Advantages :

- Can be configured in multi-master mode.
- Complexity is reduced because it uses only 2 bi-directional lines (unlike SPI Communication).
- Cost-efficient.
- It uses ACK/NACK feature due to which it has improved error handling capabilities.

Limitations :

- Slower speed.
- Half-duplex communication is used in the I2C communication protocol.

Serial Peripheral Interface (SPI)

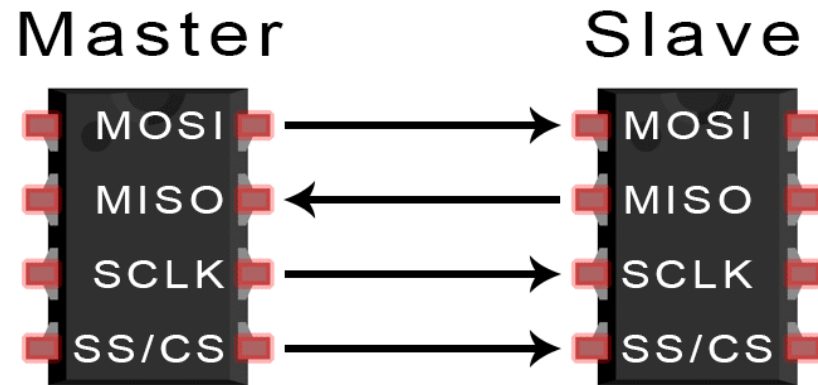
SPI requires 4 lines

- SPI allows only one Master
- SPI allows high data rate (clock rate up to 10MHz in some devices) full duplex connections
- In SPI, the slave devices are not addressable (CS line used)
- More Information:
 - <https://www.i2c-bus.org/specification/>

Serial Peripheral Interface SPI

- A synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems.
- For example, SD card modules, RFID card reader modules, and 2.4 GHz wireless transmitter/receivers all use SPI to communicate with microcontrollers.
- One unique benefit of SPI is the fact that data can be transferred without interruption. Any number of bits can be sent or received in a continuous stream.
- The master is the controlling device (usually a microcontroller), while the slave (usually a sensor, display, or memory chip) takes instruction from the master.
- The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave.

Serial Peripheral Interface SPI



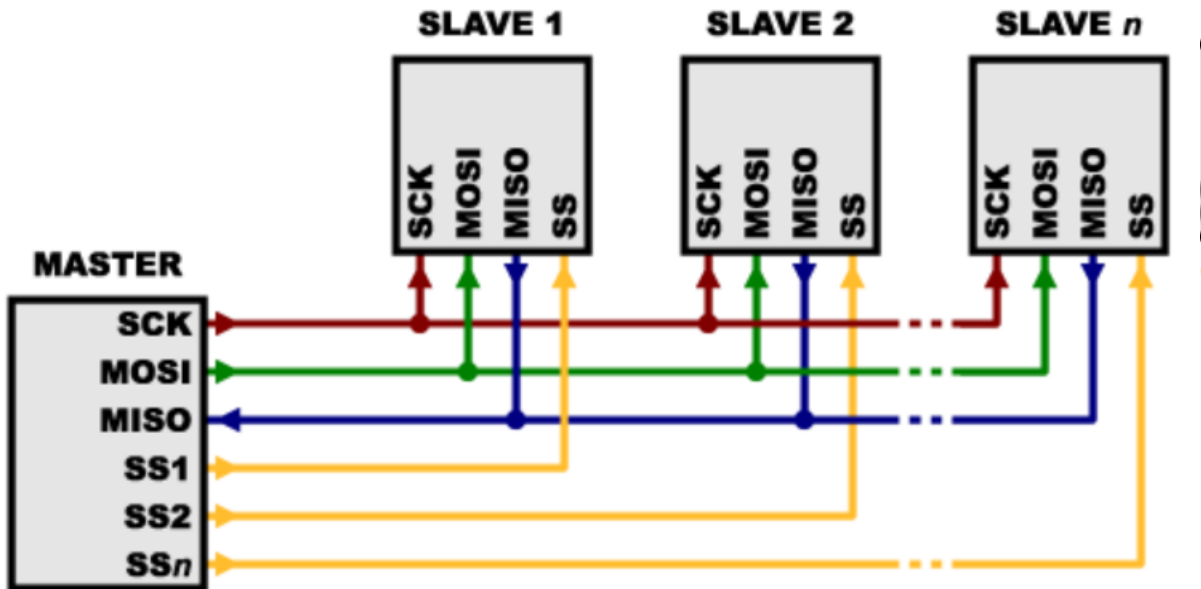
MOSI (Master Output/Slave Input) – Line for the master to send data to the slave.

MISO (Master Input/Slave Output) – Line for the slave to send data to the master.

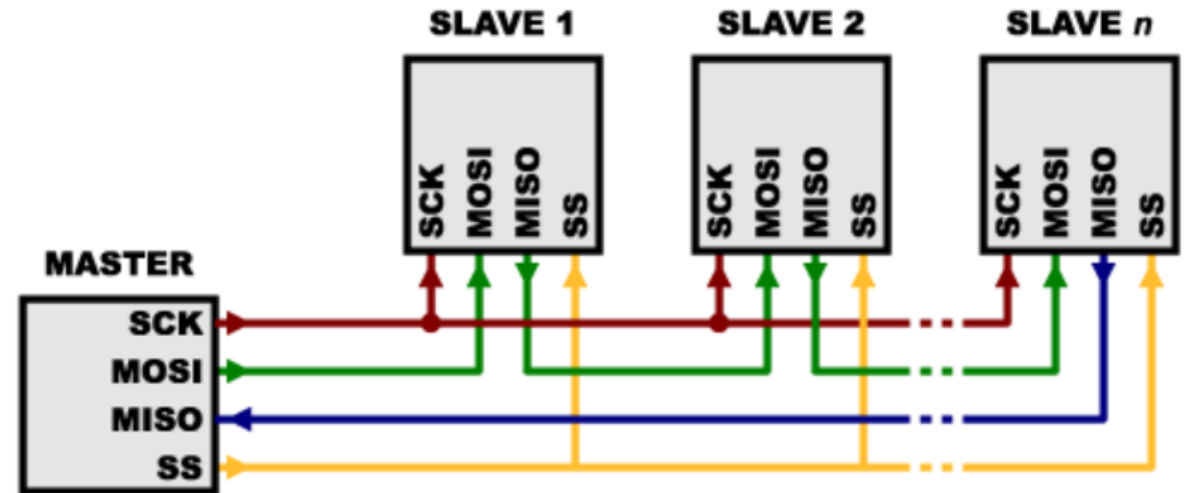
SCLK (Clock) – Line for the clock signal.

SS/CS (Slave Select/Chip Select) – Line for the master to select which slave to send data to.

Configurations



Independent slave configuration



Daisy Chain Configuration

Pros and Cons

Advantages :

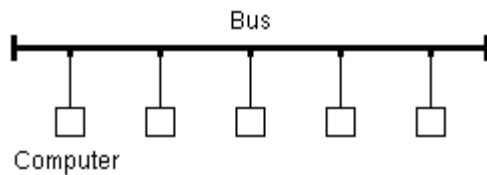
- No slave addressing system- unlike in I2C.
- Higher data transmission rate.
- Separate lines for MISO and MOSI which ensured the full duplex communication.

Limitations :

- Uses more buses than UART and I2C.
- No form of error checking like the parity bit in UART.
- Only allows for a single master.

Ethernet

- Shared network between peers
- Open contention for network CSMA/CD
 - Detect collisions as they occur



- Problem: collisions slow down the network

Note: this is all original Ethernet design

Arduino Wire library and I2C

This library allows you to communicate with I2C/TWI devices.

Board	I2C/TWI pins
UNO, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	20 (SDA), 21 (SCL), SDA1, SCL1

The Wire library uses 7 bit addresses throughout.

Functions in Wire Library

[begin\(\)](#)

[end\(\)](#)

[requestFrom\(\)](#)

[beginTransaction\(\)](#)

[endTransmission\(\)](#)

[write\(\)](#)

[available\(\)](#)

[read\(\)](#)

[setClock\(\)](#)

[onReceive\(\)](#)

[onRequest\(\)](#)

[setWireTimeout\(\)](#)

[clearWireTimeoutFlag\(\)](#)

[getWireTimeoutFlag\(\)](#)

I2C Lab

- Find a partner for this lab as need to work in pair.
- Open <https://docs.arduino.cc/learn/communication/wire#tutorials>
- Try Controller Reader Sketch and Controller Writer Sketch using two Arduino boards
- Try Controller Writer Sketch and Peripheral Receiver Sketch using two Arduino boards

Lab Assignment

- Create two Sketches for two Arduino boards to do the following tasks

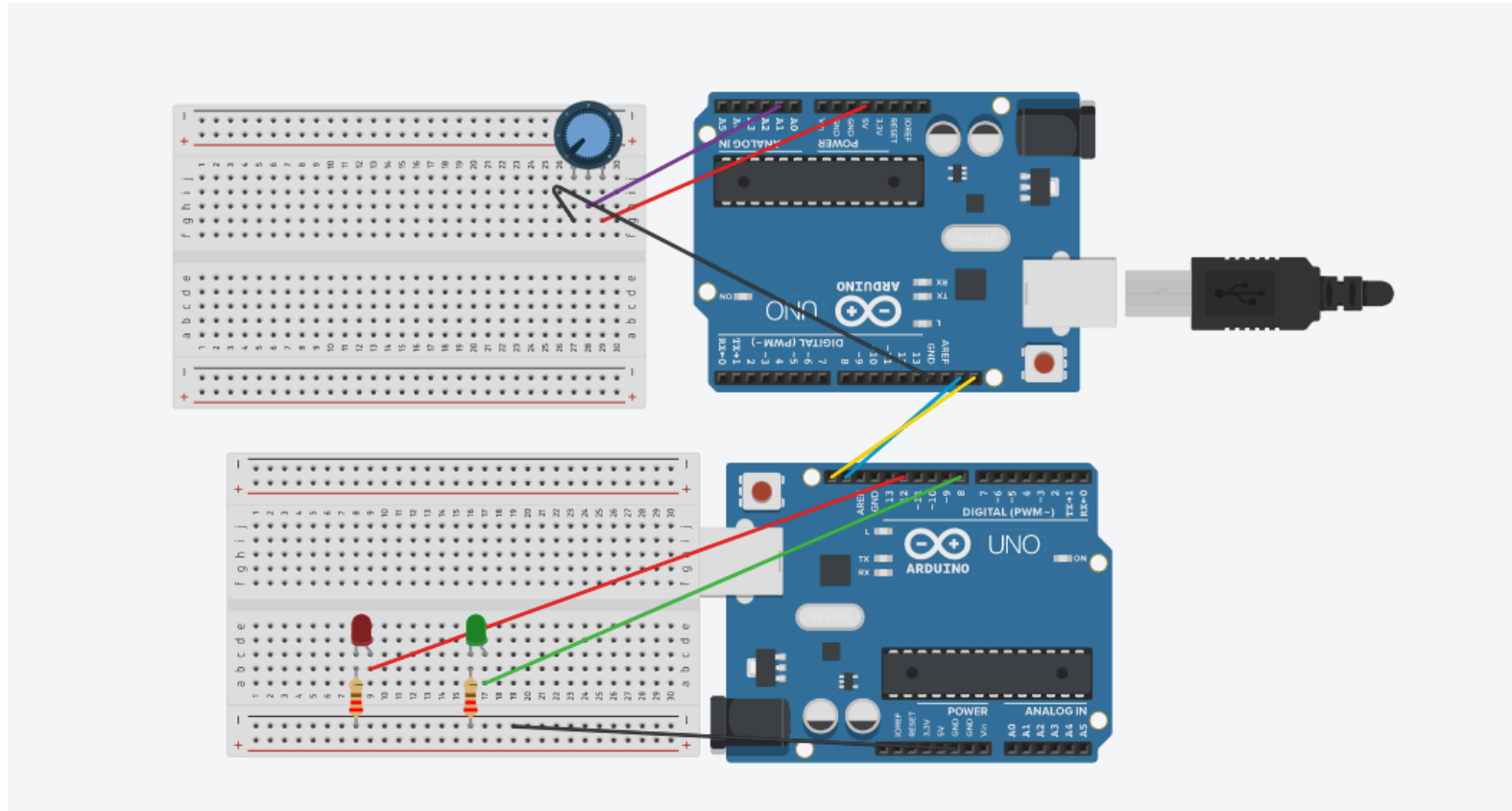
Slave Arduino

- Connect a potentiometer to one of the analog inputs of this slave Arduino board.
- Send the value of the potentiometer to the master whenever it is requested.

Master Arduino

- Connect two LEDs with different colors (red and green) to two digital outputs of this master Arduino board. (don't forget to use resistors)
 - Receive the value of the potentiometer from the slave Arduino every 5 seconds.
 - Check the received value and if above half or one specified value, turn on the red LED; otherwise turn on the green LED.
- Demonstrate your work to the Instructor or TAs.
 - Submit your code and the photo of your circuit.

Circuit



Reference

- <https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/>
- Lee, Edward & Seshia, Sanjit. (2011). Introduction to Embedded Systems - A Cyber-Physical Systems Approach.
- Lecture Note Slides from EECS 149/249A: Introduction to Embedded Systems (UC Berkeley) by Prof. Prabal Dutta and Sanjit A. Seshia
- Lecture Note Slides from ICEN 553/453– Fall 2018: Cyber-Physical Systems () by Prof. Dola Saha