Lecture 13
Wired Communication: SPI and I2C

CE346 – Microprocessor System Design

Branden Ghena – Spring 2021

Some slides borrowed from:
Josiah Hester (Northwestern), Prabal Dutta (UC Berkeley), Sparkfun

Northwestern

Today's Goals

- Discuss additional wired communication protocols: SPI and I2C

- Understand tradeoffs in design
  - UART, SPI, and I2C are each useful for different scenarios

Outline

- **SPI**

- I2C

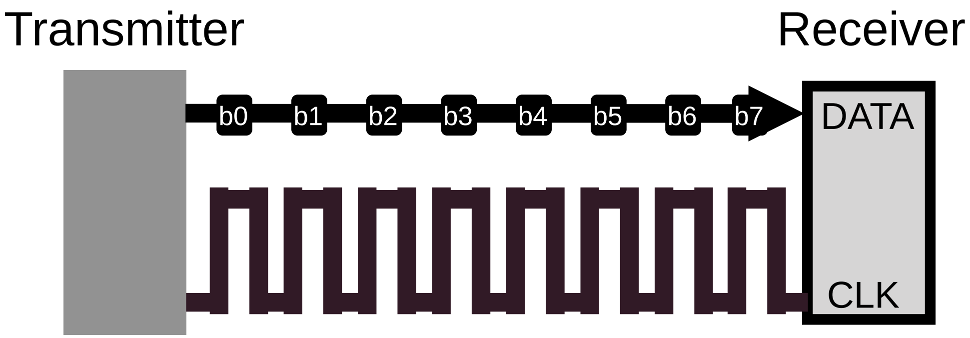
UART Pros and Cons

- Pros
  - Only uses two wires
  - No clock signal is necessary
  - Can do error detection with parity bit

- Cons
  - Data frame is limited to 8 bits (20% signaling overhead)
  - Doesn’t support multiple device interactions (point-to-point only)
  - Relatively slow to ensure proper reception

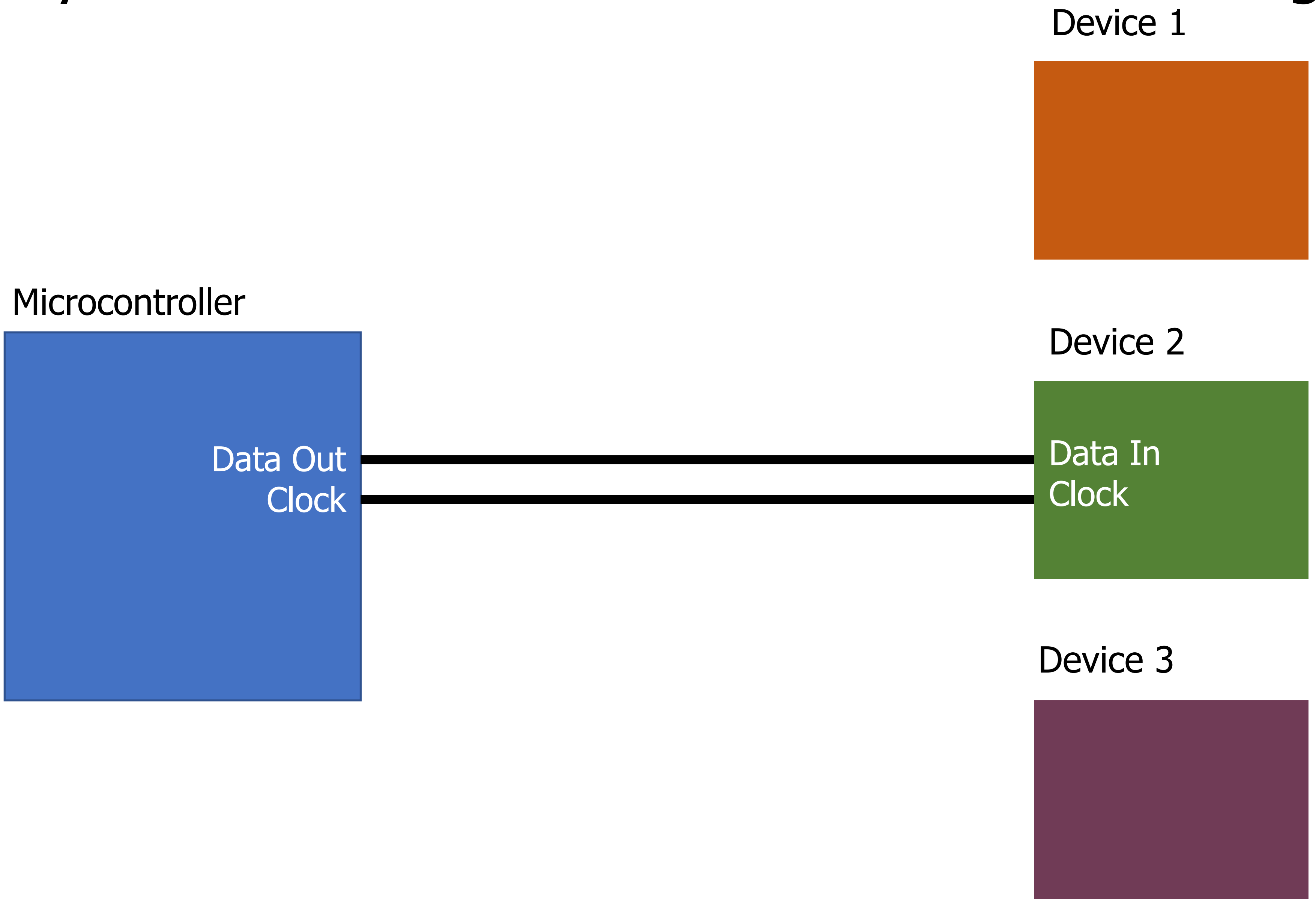
- Let’s get rid of all the cons (by sacrificing on all the pros)

Synchronous UART

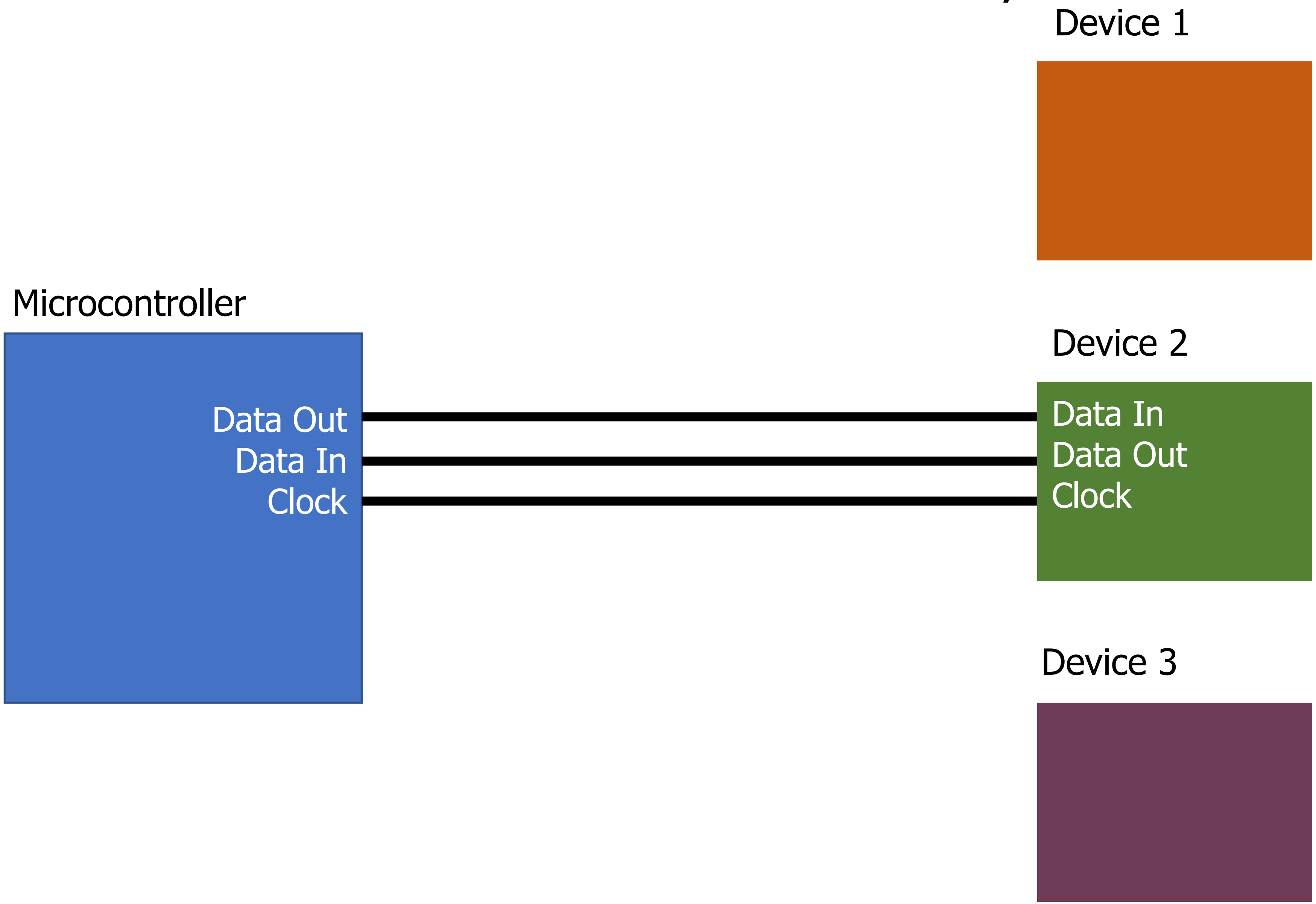


- USART
  - Synchronous/Asynchronous
  - Just add a clock line
- Common peripheral in many microcontrollers to allow adaptable communication
  - Could build various protocols (like SPI) on top of it
- Still point-to-point limited in this form

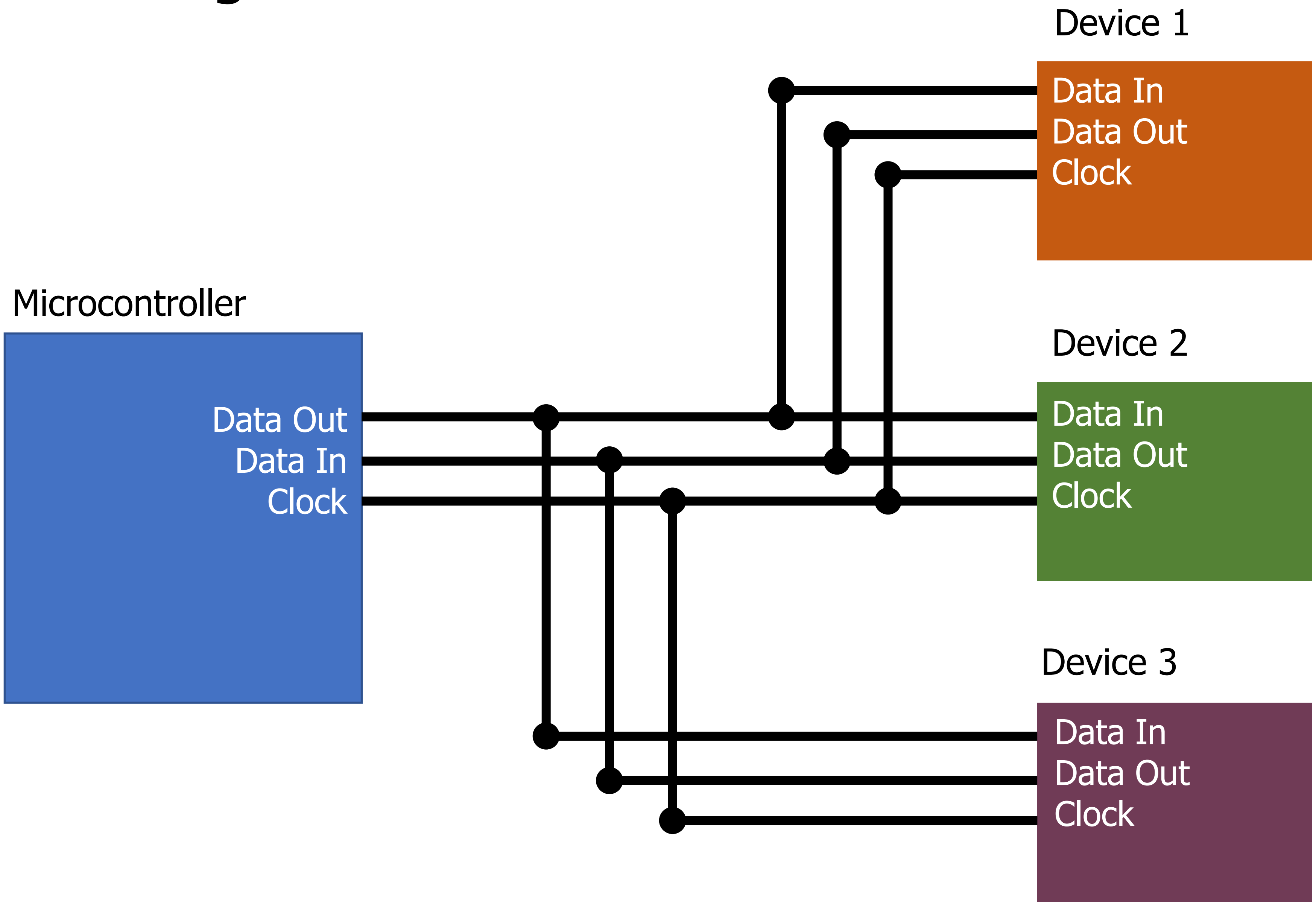
Synchronous serial communication with a single device



Want bi-directional communication, so three wires



Wire signals to all devices to form a bus



Communicating on a bus

How do you distinguish which device you are talking to?

- Address for each device

- GPIO pin for each device

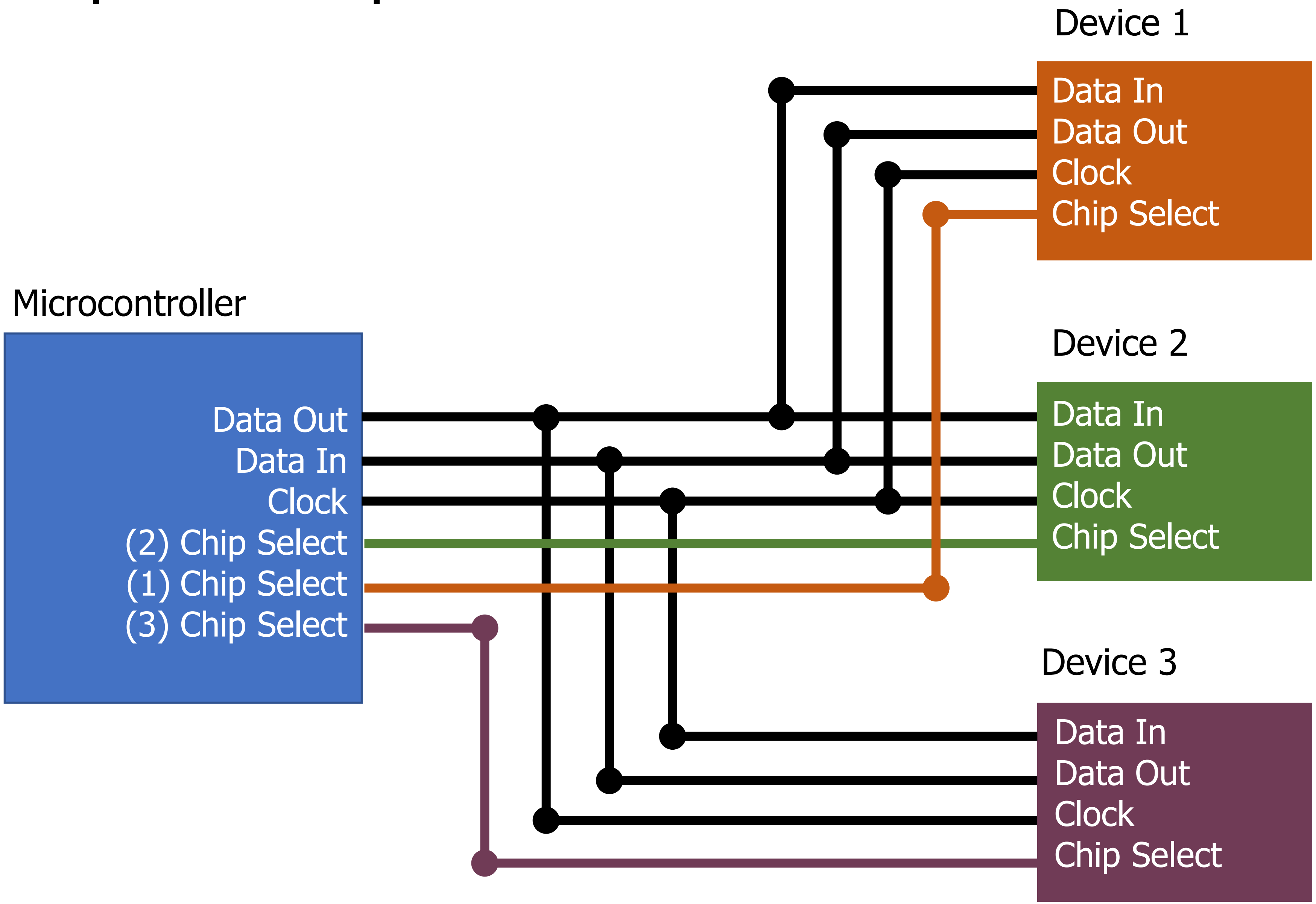
Communicating on a bus

How do you distinguish which device you are talking to?

- Address for each device
    - Devices must always listen and then discard messages that aren’t for them
    - Need to define packet format so it’s clear where the address is
    - Need a method for addressing devices

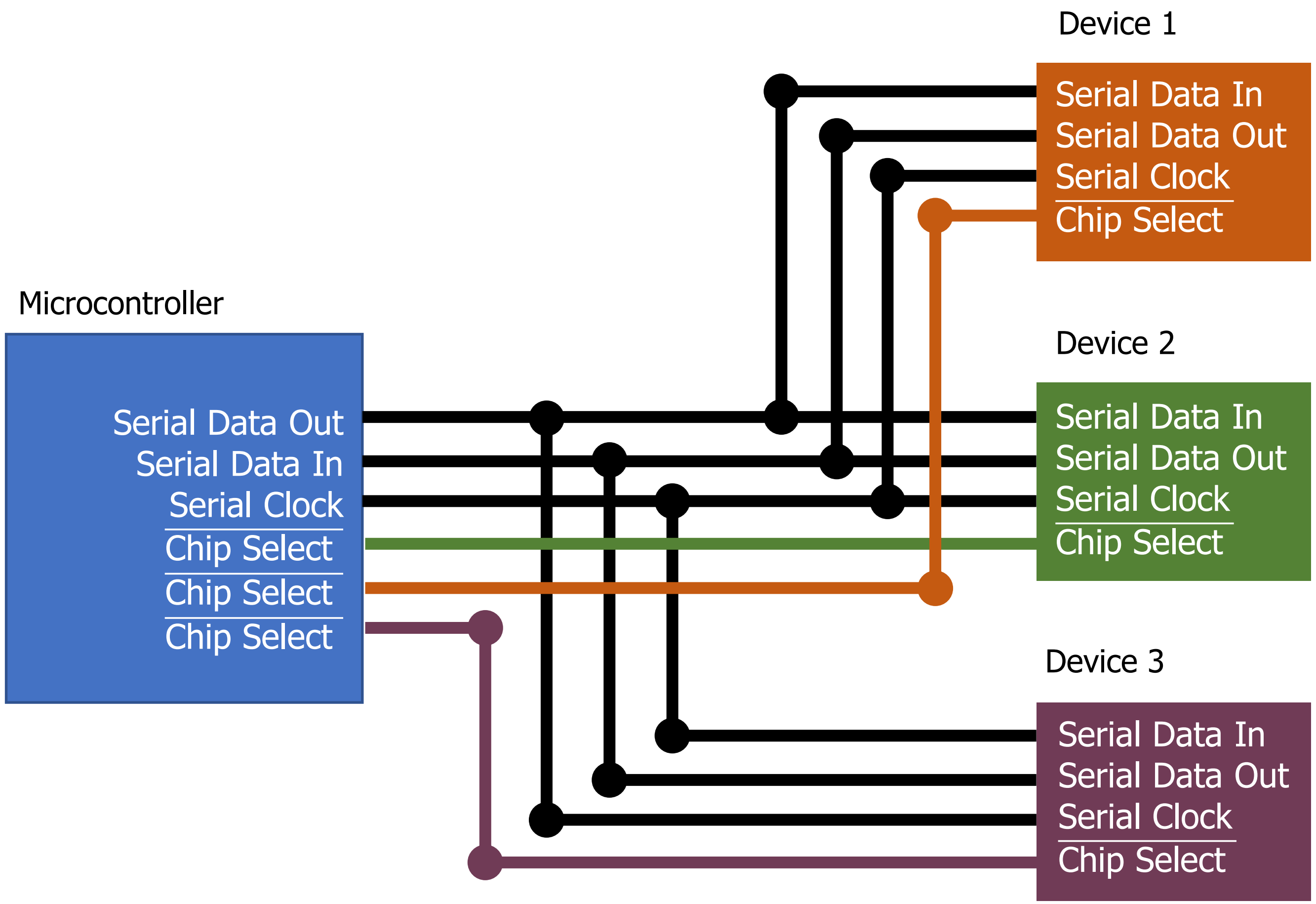
- GPIO pin for each device
    - Signal which device is being communicated with
    - Only activates communication on transition of “select” line

Separate chip select line for each device



Serial Peripheral Interface (SPI)

- Serial, synchronous, bus communication protocol
- Single controller with multiple peripherals
  - Within a circuit board
- High-speed communication
  - Multiple Mbps



A note on outdated notation

- Master/Slave paradigm
  - Master is the “Computer” and is in charge of interaction
  - Slave is the “Device” and has little control over interaction parameters
  - Really common notation in EE side of the world.
    - Not intended to be harmful, but also literally inconsiderate.

- Field is changing for the better. It’s going to take some time.
  - **Controller/Peripheral**
  - Central/Peripheral
  - Device/Peripheral
  - Master/Minion
  - Primary/Secondary

SPI naming schemes

- Historical SPI Naming
  - MISO – Master In Slave Out
  - MOSI – Master Out Slave In
  - SS – Slave Select

- Revised SPI Naming
  - CIPO – Controller In Peripheral Out
    - SDI – Serial Data In (for devices which could act as either role)

  - COPI – Controller Out Peripheral In
    - SDO – Serial Data Out (for devices which could act as either role)

  - CS – Chip Select

https://www.oshwa.org/a-resolution-to-redefine-spi-signal-names

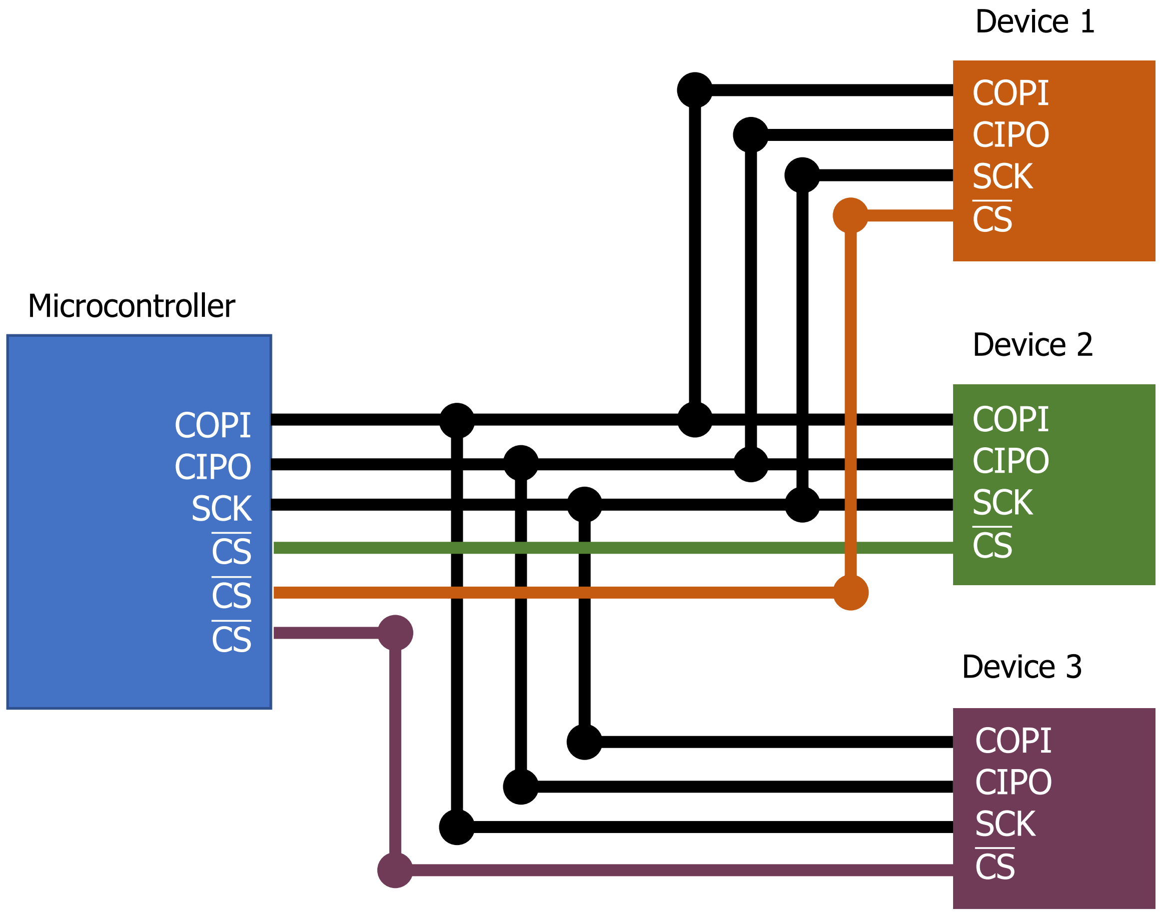
https://www.sparkfun.com/spi_signal_names

SPI wiring

- 3+*N* wires for *N* peripherals

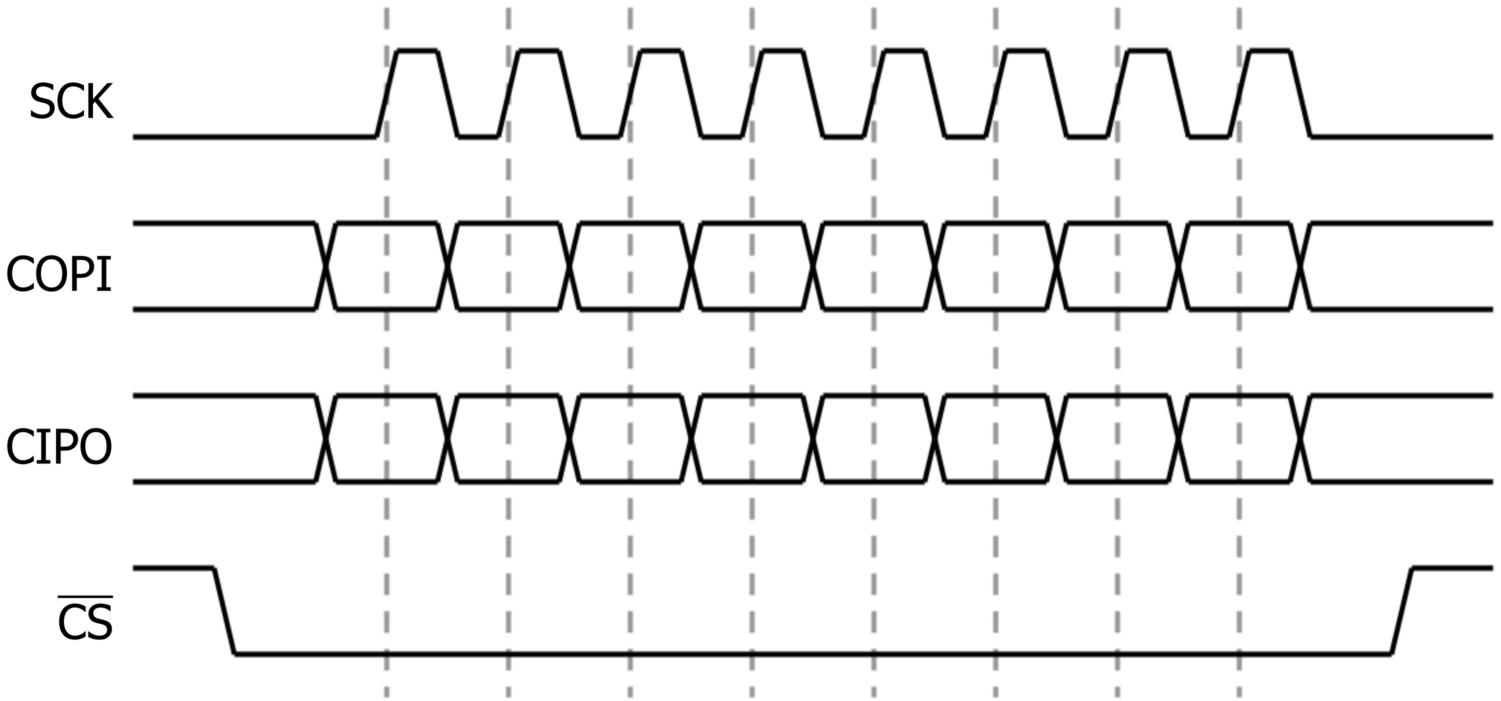
- COPI – Controller Out Peripheral In
- CIPO – Controller In Peripheral Out
- SCK – Serial ClocK
- CS – $\overline{\text{Chip Select}}$
  - Active low signal

- Longer names remove ambiguity about communication
  - COPI to COPI
  - CIPO to CIPO



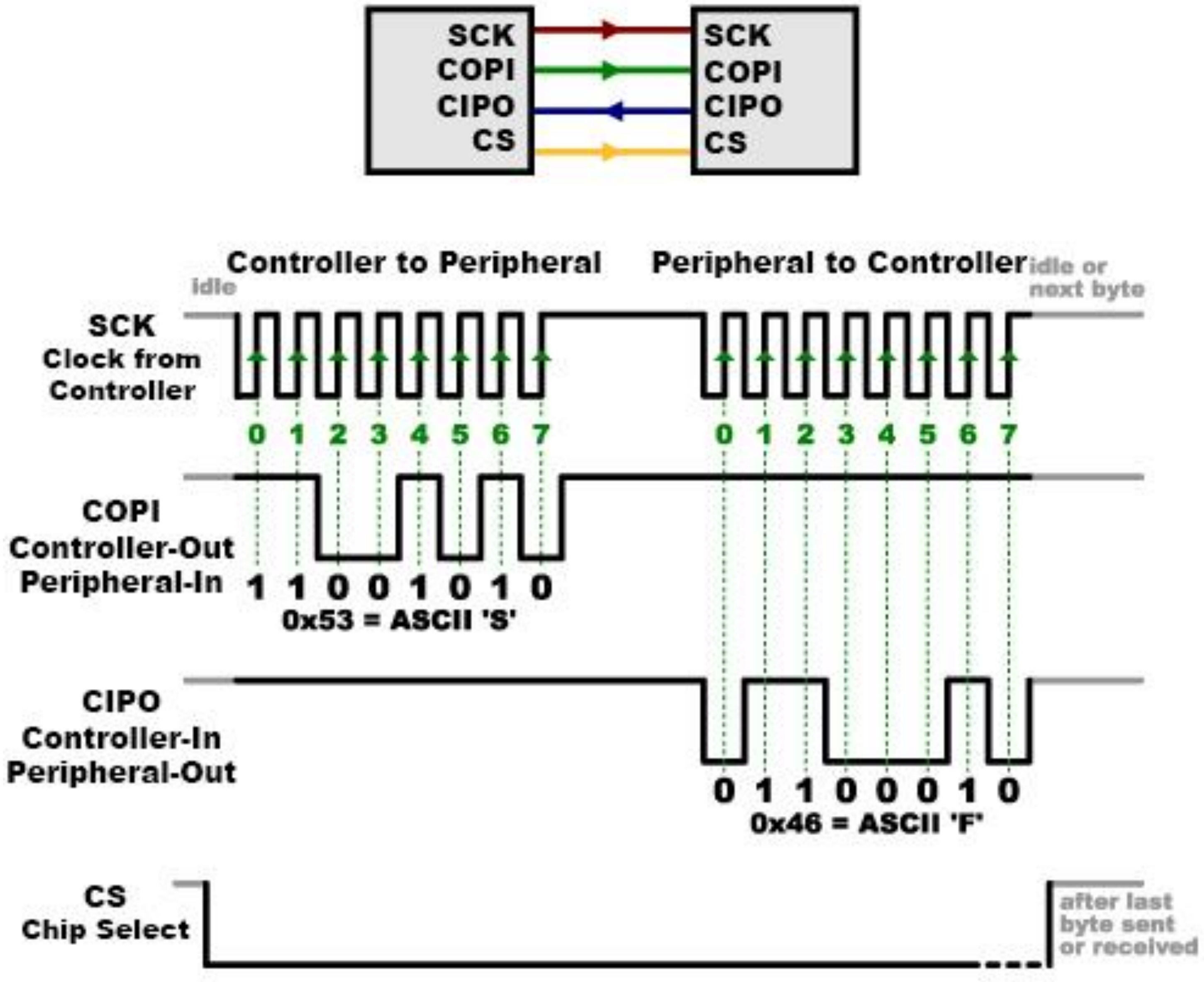
SPI timing diagram

- CS goes low to start transaction and high to end
- Data is sent synchronously with clock signals
- Capable of full-duplex transfers
  - Both directions at the same time

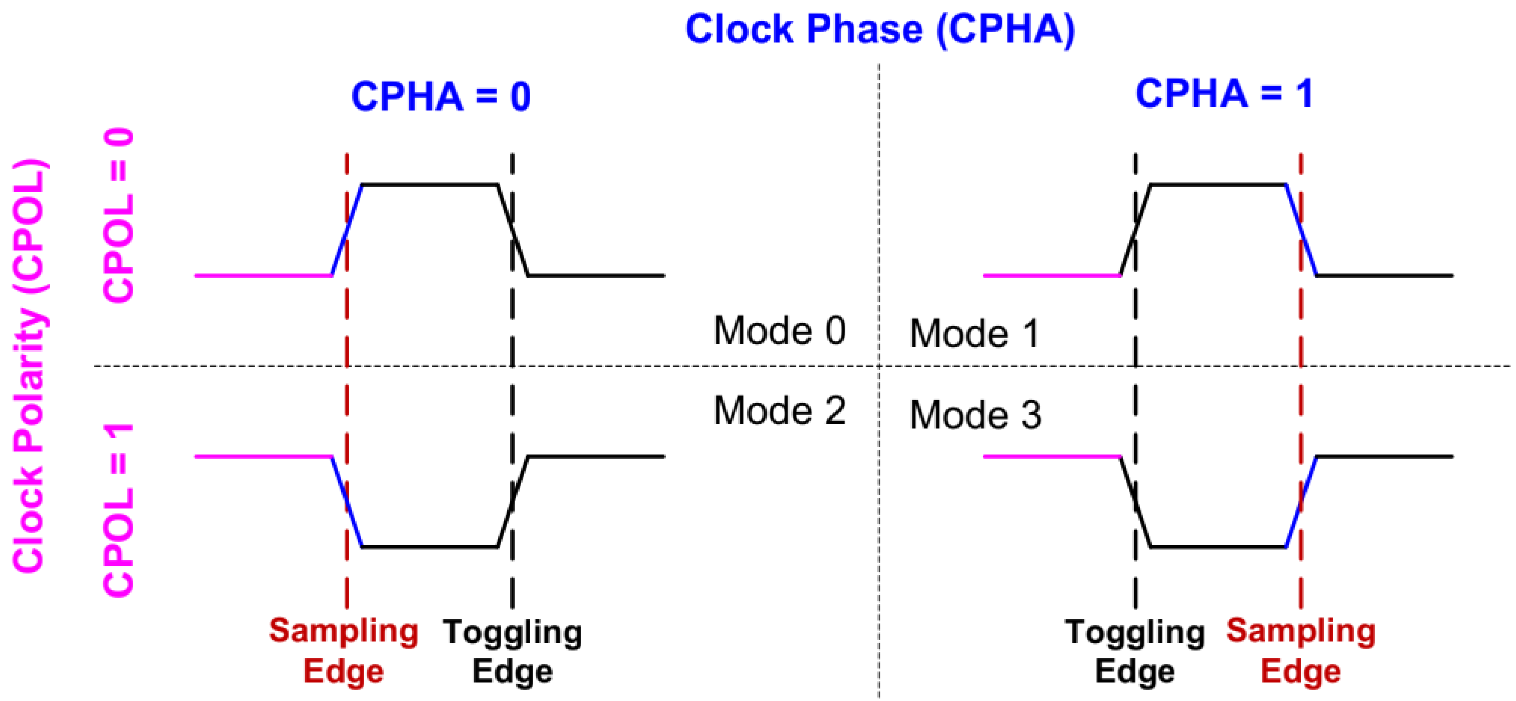


SPI communication

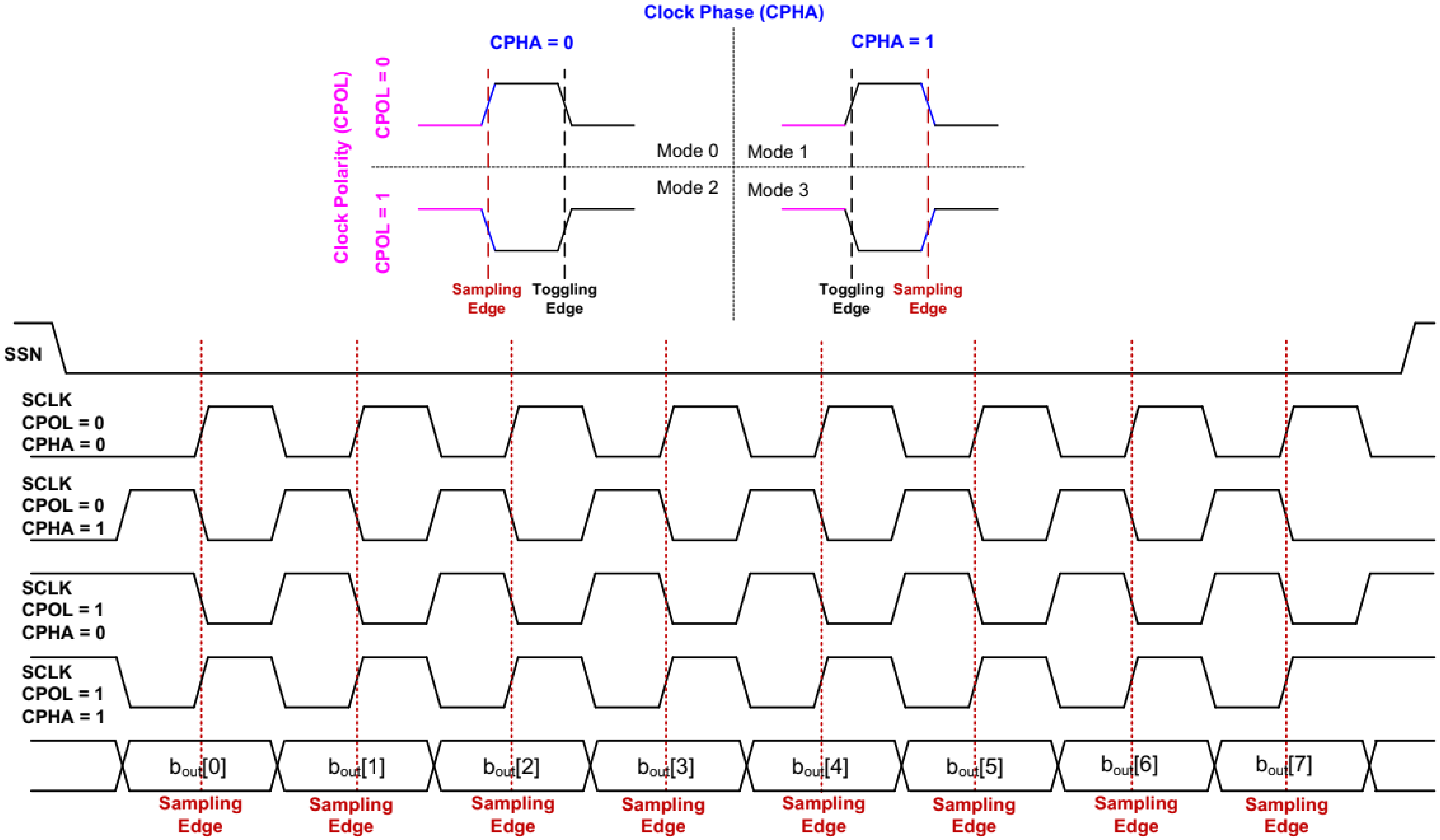
- Transactions usually in multiples of bytes (as many as needed)
- Bytes are sent LSb first
- No need for framing bits (start/stop)
    - CS handles that



SPI configurations



- CPOL – is the clock default low or default high
- CPHA – is data read on first edge or second edge
- Peripherals tell you what their configuration is


Clock Phase (CPHA)
CPHA = 0
CPHA = 1
Clock Polarity (CPOL)
CPOL = 0
CPOL = 1
Mode 0
Mode 1
Mode 2
Mode 3
Sampling Edge
Toggling Edge
Toggling Edge
Sampling Edge
SSN
SCLK CPOL = 0 CPHA = 0
SCLK CPOL = 0 CPHA = 1
SCLK CPOL = 1 CPHA = 0
SCLK CPOL = 1 CPHA = 1
$b_{out}[0]$
$b_{out}[1]$
$b_{out}[2]$
$b_{out}[3]$
$b_{out}[4]$
$b_{out}[5]$
$b_{out}[6]$
$b_{out}[7]$
Sampling Edge

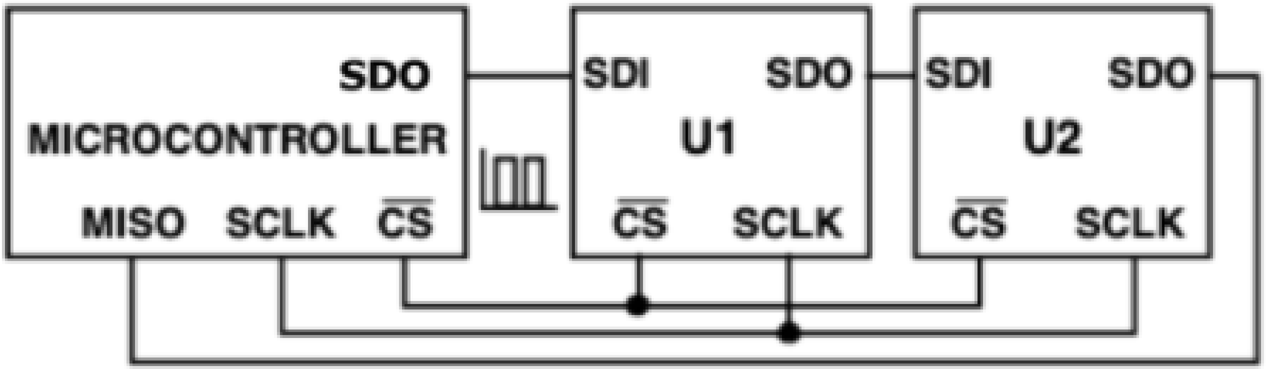
SPI data rate

- No particular requirements
  - Speed can go as fast as your clock and line capacitance can handle

- Datasheet for devices will specify their speeds
  - Sort of standards (less so than UART, for example)
    - 700 kbps
    - 3.4 Mbps
    - 10 Mbps

Daisy-chaining SPI

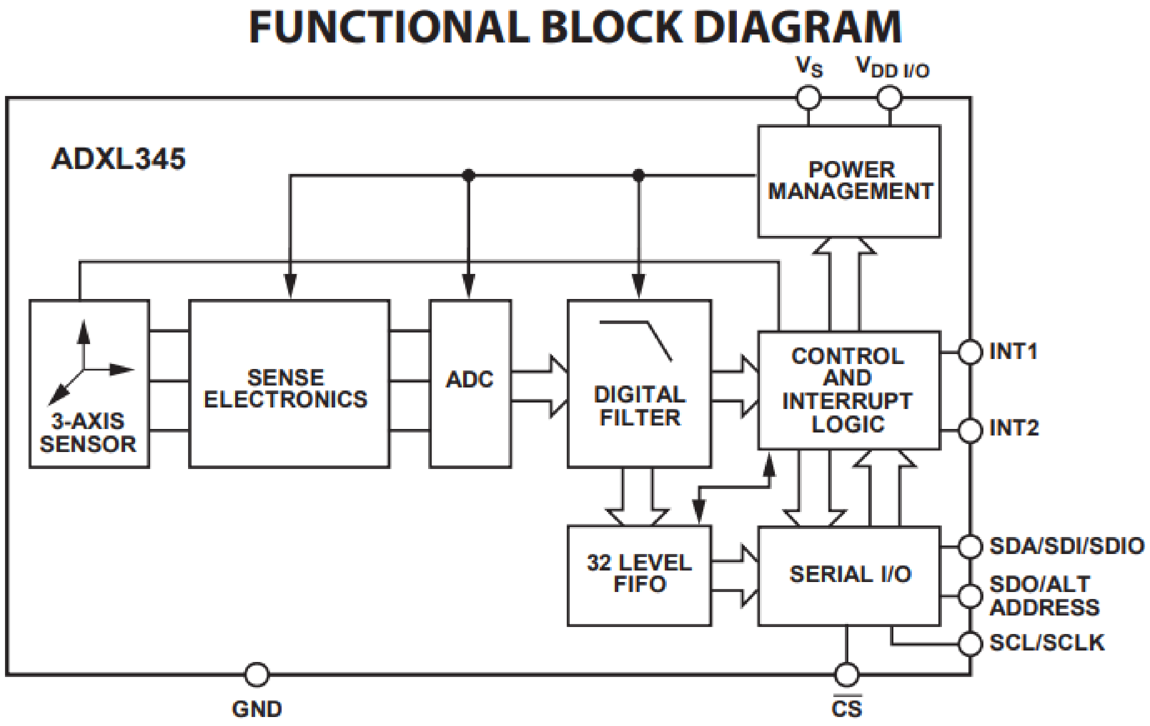
- SPI can also be formed into a ring bus
- Doesn't save on pins, but does reduce wires…
  - At the cost of reliability and speed
- Fairly rare in practice



Bi-directional communication

- Controller starts/stops SPI transfers

- Peripherals often add interrupt outputs to signal controller that an event has occurred
  - More pins, yay!



Use Cases

- High-speed peripherals
  - Microphone, Accelerometer, External ADC

- External memory
  - Memory chips
  - SD cards
    - All SD cards support a SPI communication mode

  - QSPI – Quad SPI (four COPI lines for more throughput)
    - Often used for communication with external memory

SPI Pros and Cons

- Pros
  - Faster throughput (and no overhead)
  - No restrictions on data frame
    - No addressing requirements or word size assumptions
  - Full duplex transfers

- Cons
  - Many pins: $3+N$ (for N peripherals)
    - CS line scales linearly (other signals are a bus)
  - Controller must initiate all transfers
    - Not designed for multi-controller scenarios

Outline

- SPI
- **I2C**

Choosing different tradeoffs from other wired communication

- Things we like from SPI
  - Communication over a bus
  - Synchronous communication

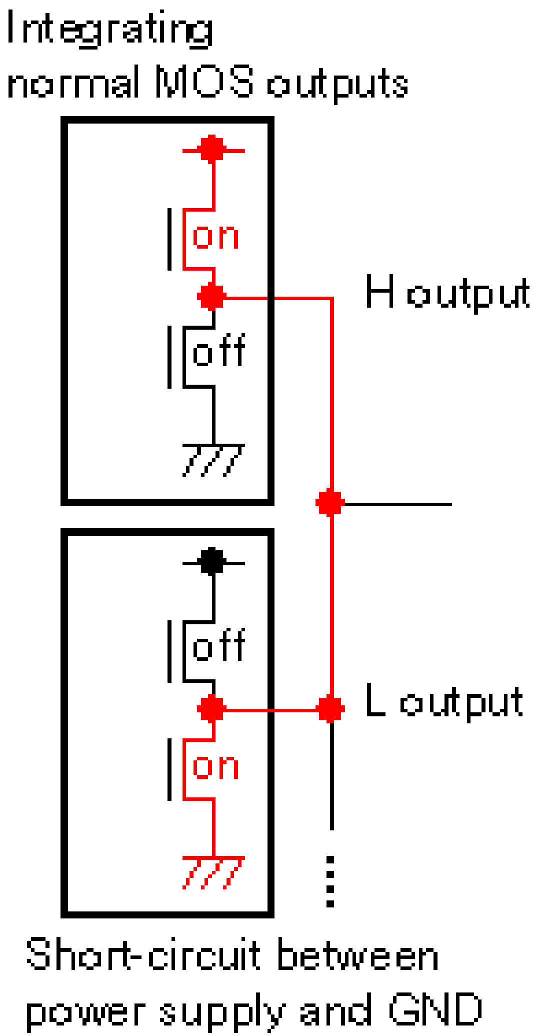
- Things we want from new protocol
  - Fewer I/O pins
    - Use a single data line for bi-directional communication
    - Needs addressing and more specified data frame

  - Multiple controllers sharing the bus
    - Needs a bus contention solution

Bus contention could short a shared bus

- Want to enable multiple controllers

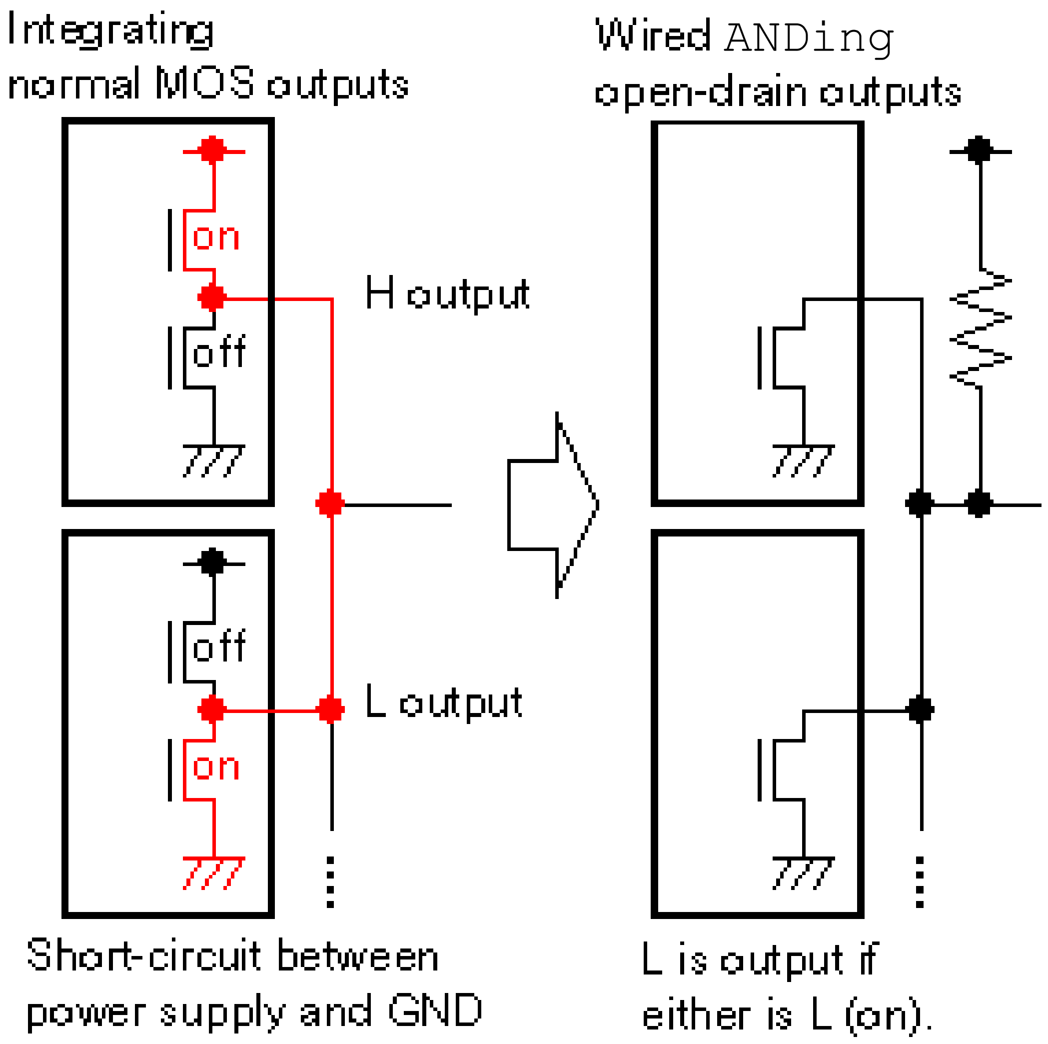
- Problem
  - What if they each try to transmit different data?
  - At some point, there will be a short-circuit



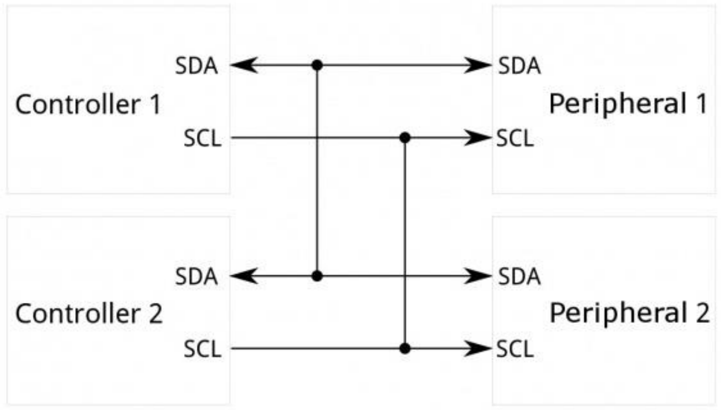
Disconnected I/O pins enable shared communication

- I/O pins often have three states
  - High
  - Low
  - Disconnected
    (also known as High-Impedance/High-Z)

- We can use this third state to enable communication over a shared line
  - Low or Disconnected
  - Wired-AND
    - 1 if they are all disconnected
    - 0 if any are low



Inter-Integrated Circuit (I^2C)



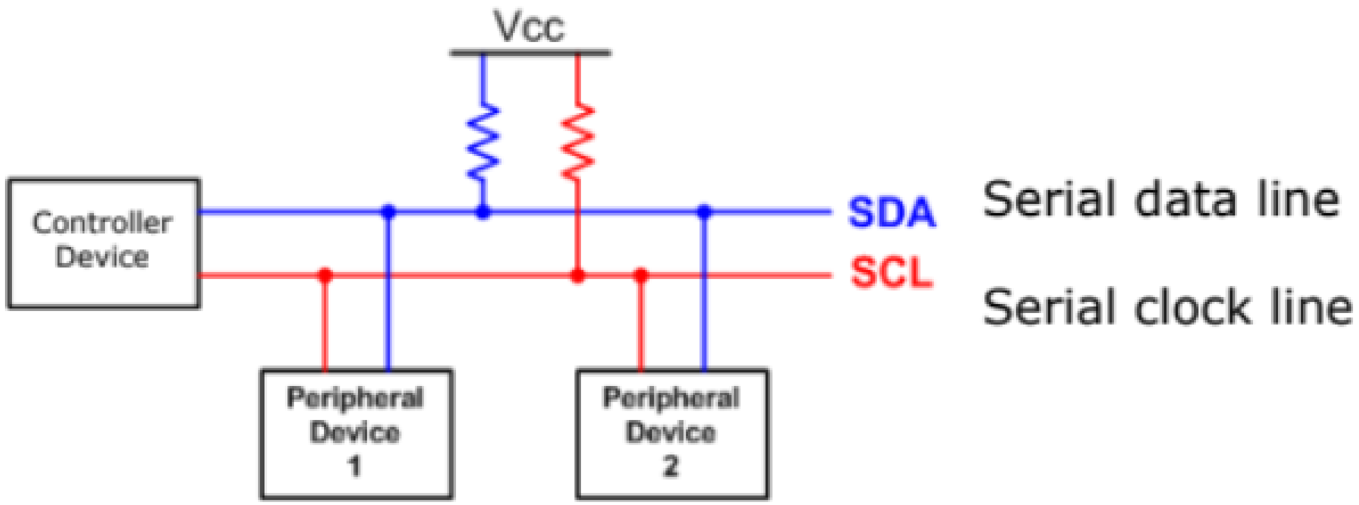
- Two-wire, synchronous, bus communication
  - Ubiquitous in the embedded world
  - De-facto standard for sensors

- Invented and patented by Phillips (now NXP)
  - Patent expired in 2004

- Also known as Two-Wire Interface (TWI)
  - Occasionally as System Management Bus (SMBus or SMB) but that's actually a related but separate thing

I2C overview

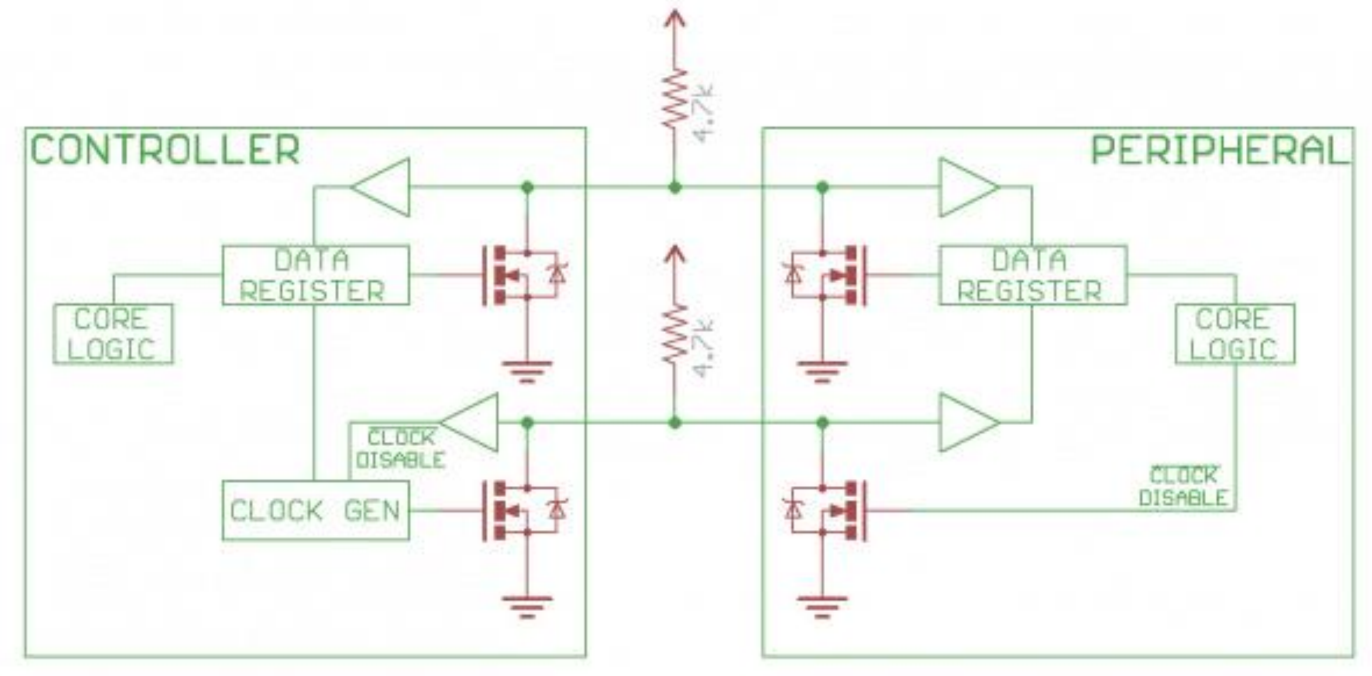
- SDA – Serial Data
- SCL – Serial Clock
  - Usually 100 kHz or 400 kHz



- Communication is a shared bus between all controller(s) and peripheral(s)

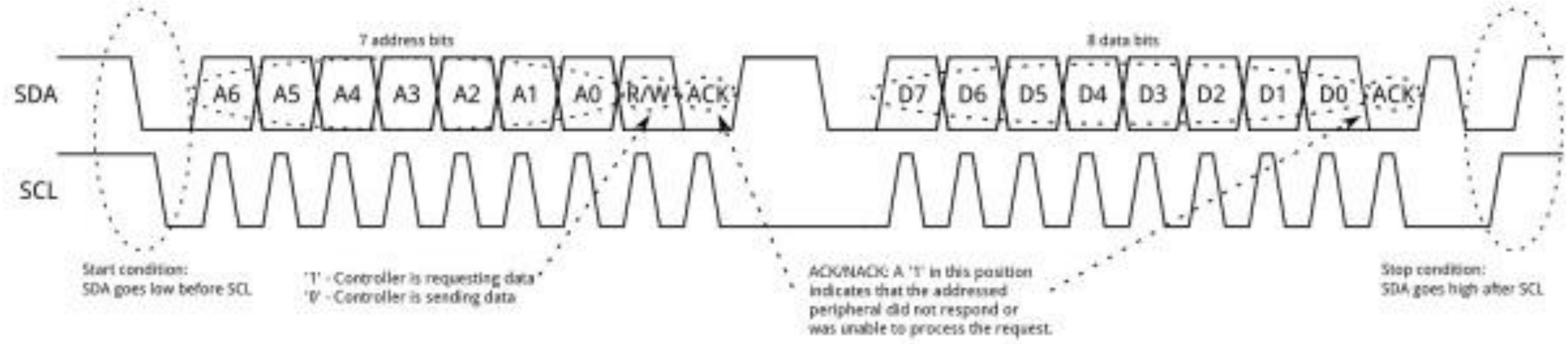
- Pull-up resistors for open-drain communication

Open drain bus communication



- SDA and SCL are open-drain
  - 1 – high-impedance, let line float high
  - 0 – active drive, pull line low
- Pull-up resistor to provide high signal
  - Low enough resistance that current can flow in a reasonable amount of time
  - Common value: 4.7 kΩ

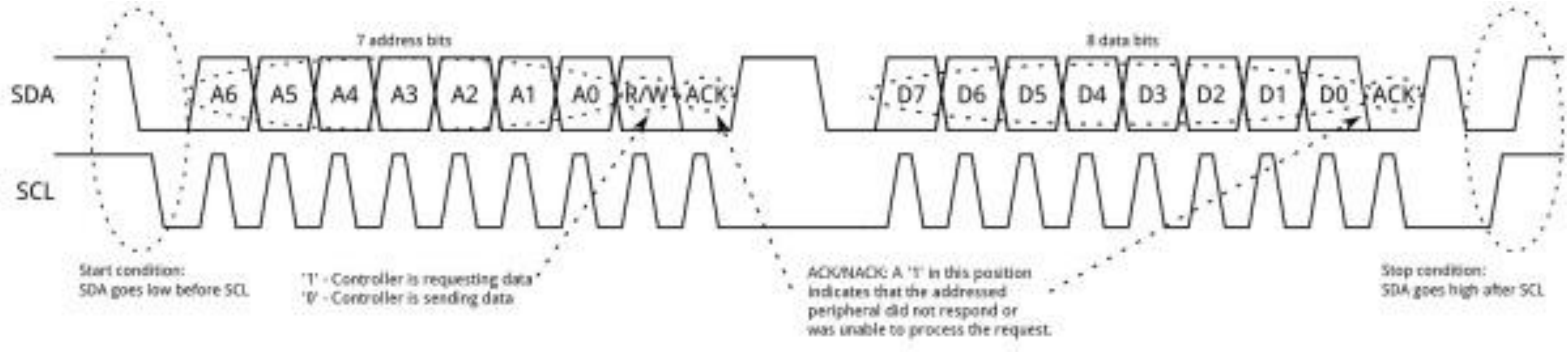
I2C transactions



- Default
  - Both lines float high (pull-up resistor)

- Start condition
  - Drive SDA low while SCL is still high

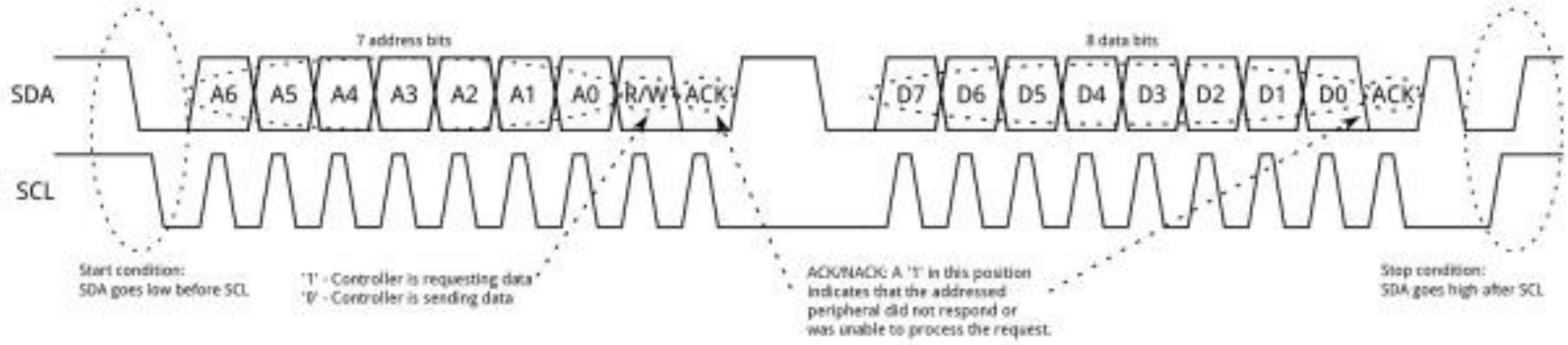
I2C transactions



- First byte is chip address + R/W indication
  - Address: 7-bit value that needs to be different for each participant
  - R/W: 1 for read, 0 for write

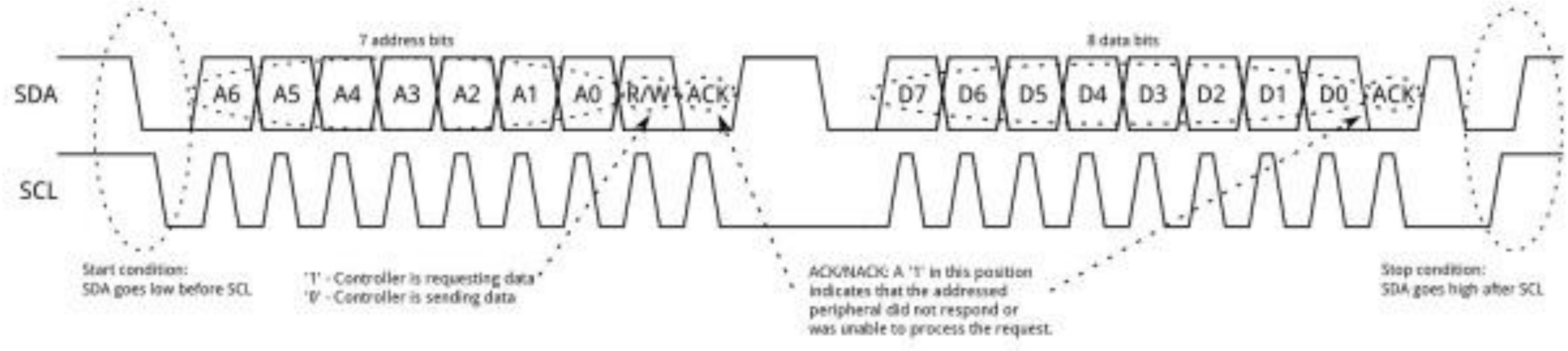
- Values are sent MSb first (reverse of other protocols)

I2C transactions



- Acknowledgement from peripheral follows each byte
  - Controller lets line float high
  - Peripheral drives line low to signal receipt of message

I2C transactions



- Data frame(s) follow
  - Sent as entire bytes, plus and ACK
  - As many as needed before Stop condition

- Stop condition
  - SDA goes high while SCL is high (normally data only changes when clock is low)

Bus arbitration

- Arbitration decides which controller gets to proceed if multiple try to communicate simultaneously

- **What happens in I2C if one controller wants a low bit and the other wants a high bit?**

Bus arbitration

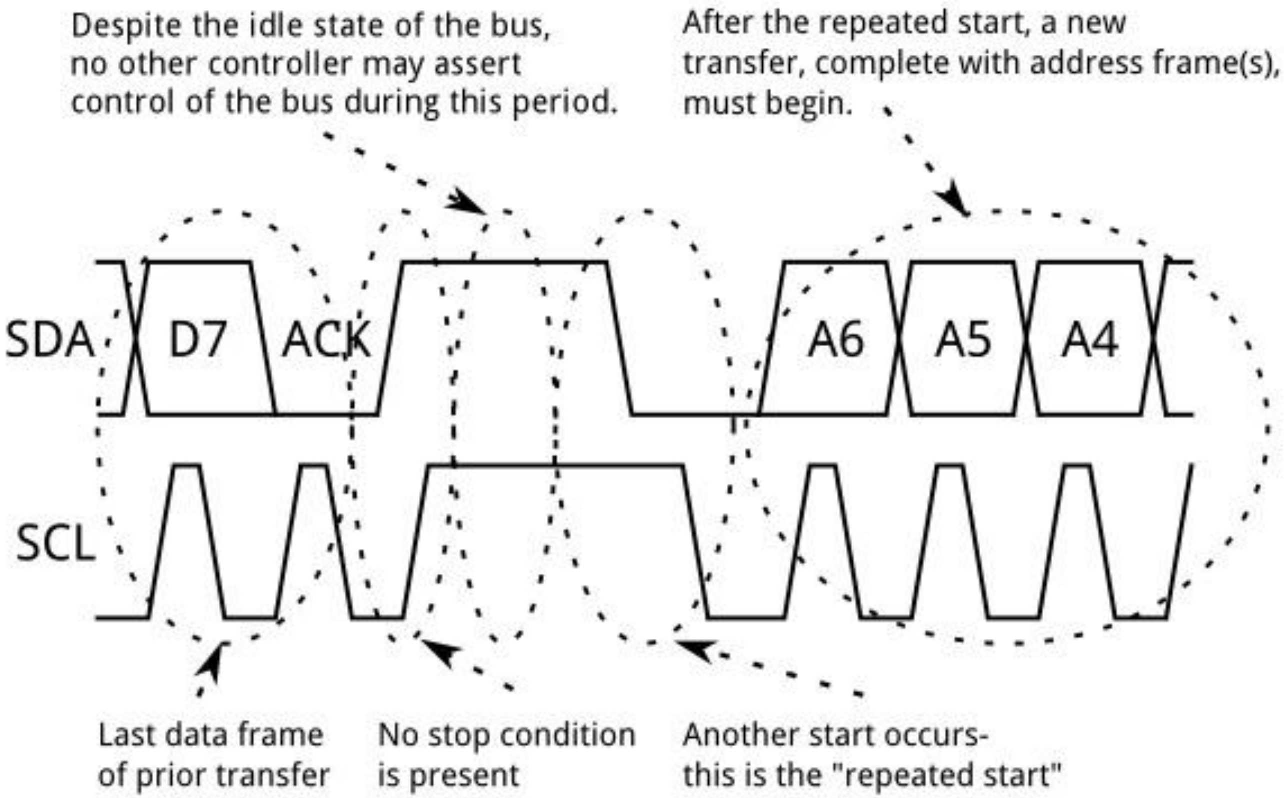
- Arbitration decides which controller gets to proceed if multiple try to communicate simultaneously

- **What happens in I2C if one controller wants a low bit and the other wants a high bit?**
  - Low bit wins! (so smaller address or data)

- Each controller constantly checks whether SDA matches the voltage level it expects
  - Stops attempting to transmit if it ever does not
  - (Only actually needs to check high signals)

Repeated start conditions

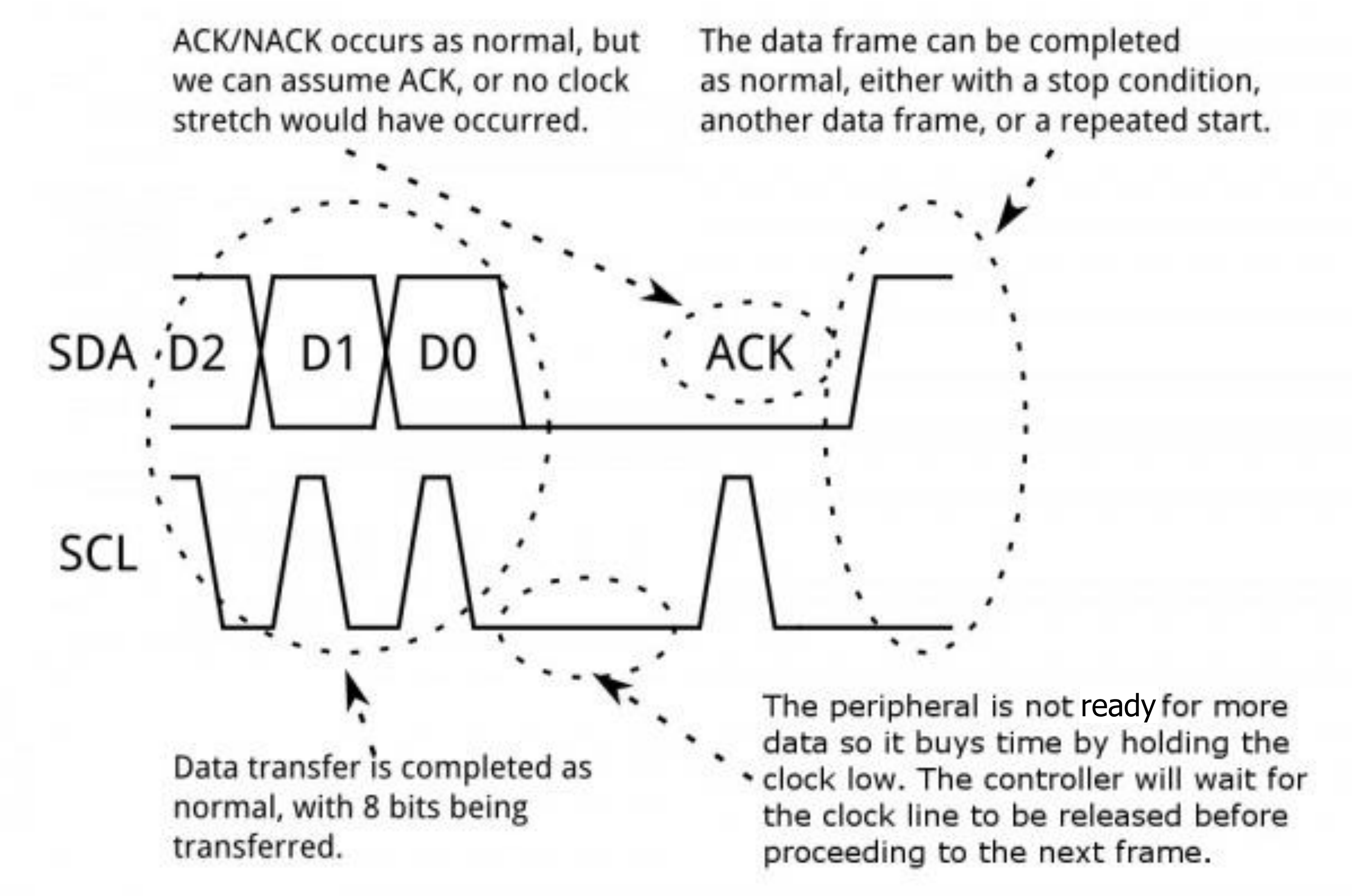
- Repeated start conditions allow the bus to be used again while arbitration was won
- Trigger another Start condition without triggering Stop condition
  - Send address again
- Frequently used for write then read pattern
  - Write which value you want
  - Then repeated start and read



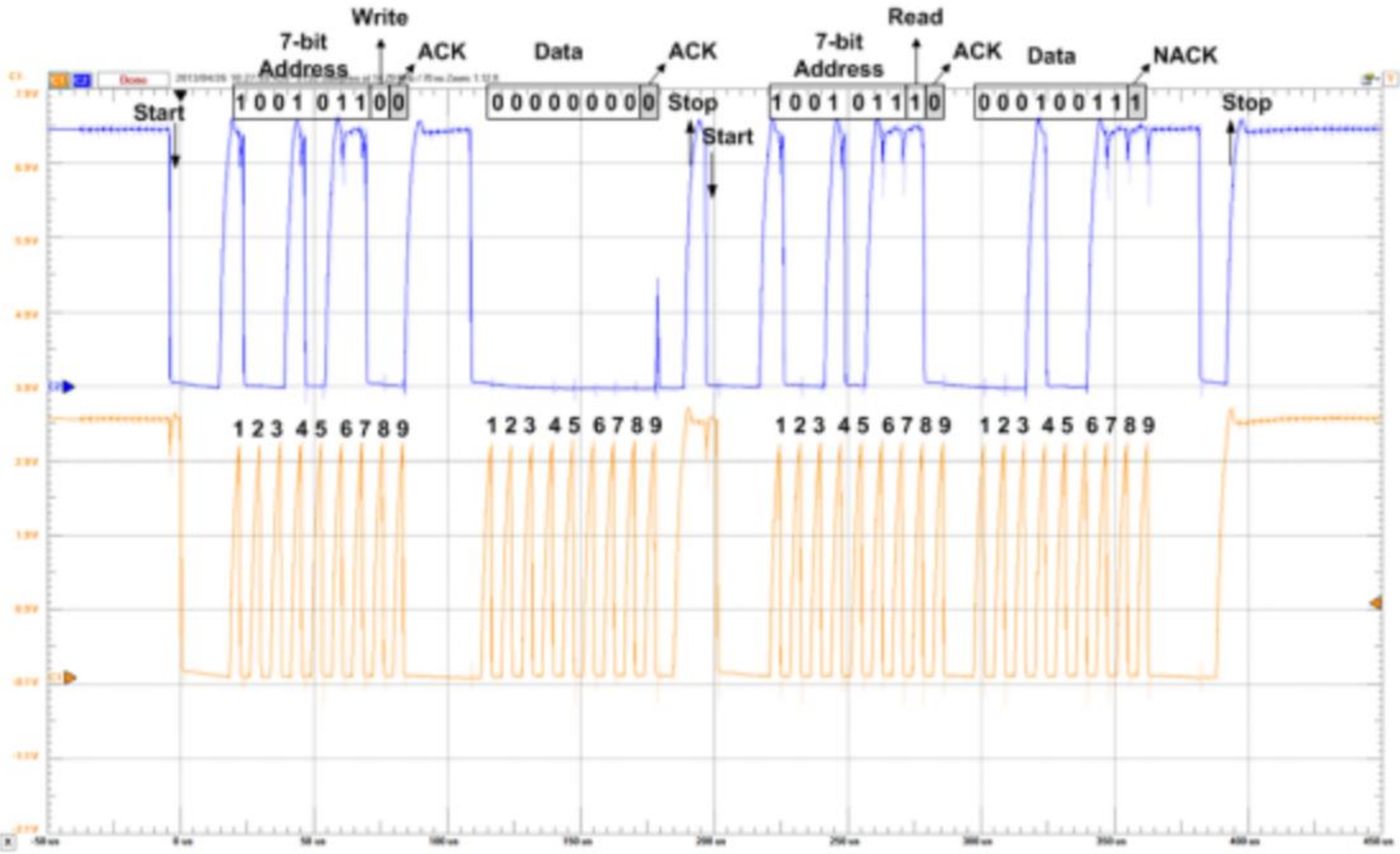
Clock stretching

- Clock is an open-drain line too
  - Either device could keep it low

- Transaction can be briefly paused by holding SCL low

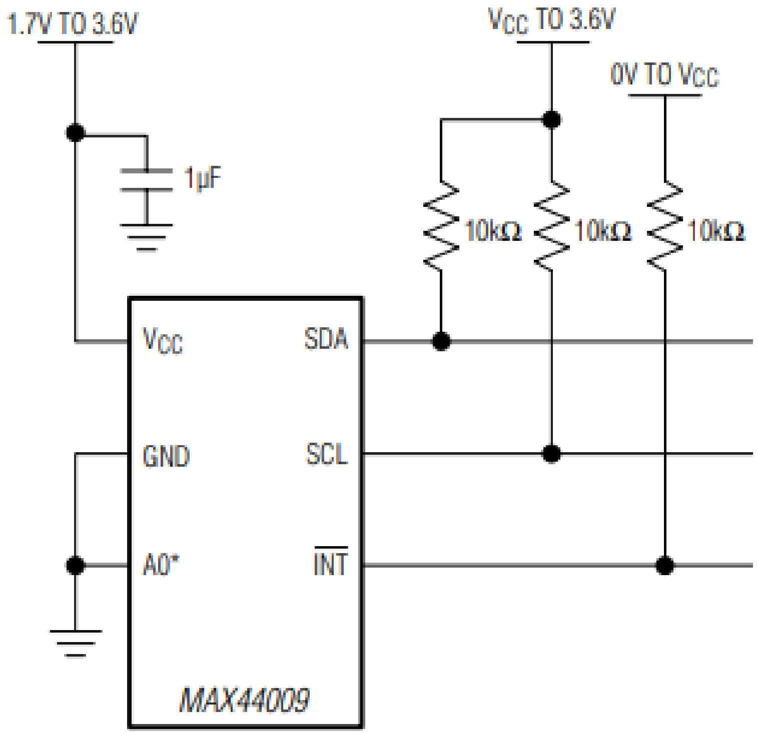


Real-world I2C transactions



Each I2C device on a bus mush have a different address

- Shared addresses would cause both to respond
- ICs often have one or more address pin(s) used to select bit(s) of address
  - 0 pins: only one may be on bus
  - 1 pin: two may be on bus
  - 2 pins: four may be on bus
- If no address pins (or not enough), need an I2C address translator chip
  - Translates addresses for one or more peripheral chips

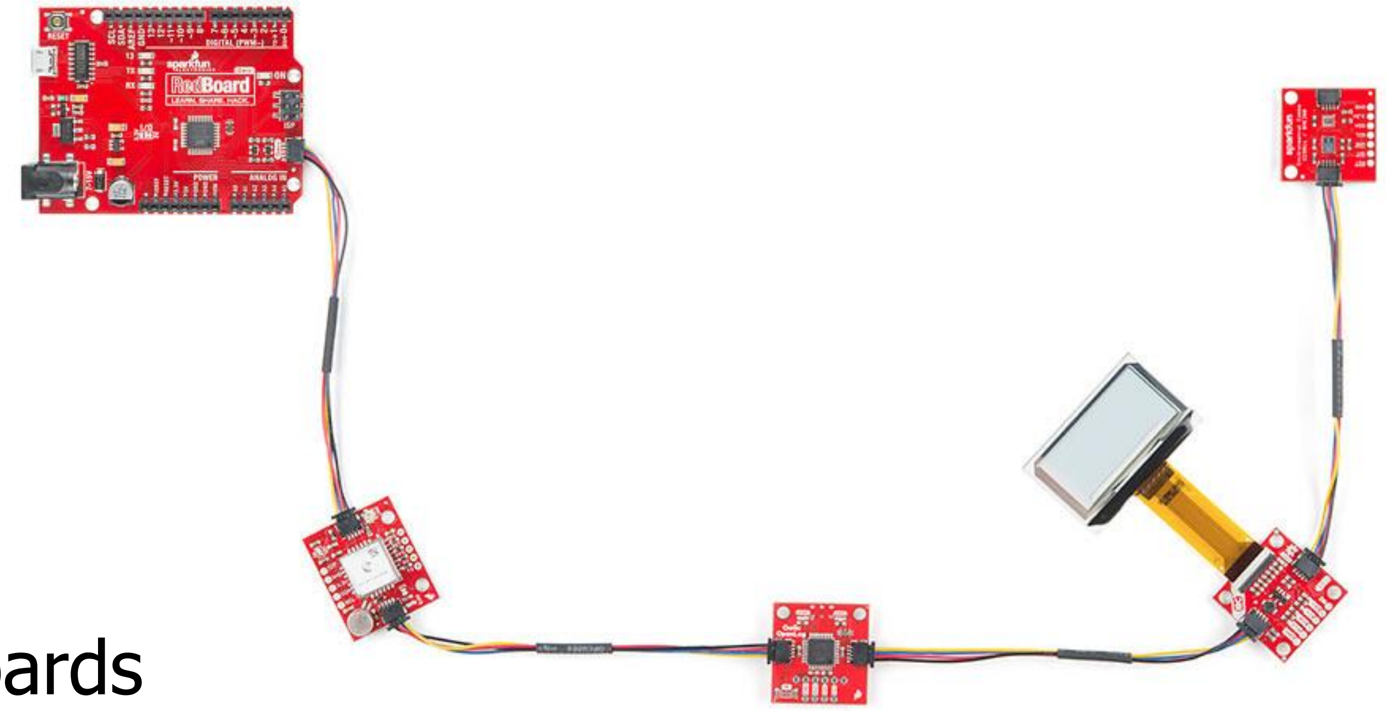


A0 is low: address 1001010x
A0 is high: address 1001011x

Sparkfun Qwiic connect system

- System for wiring multiple prototyping boards together

- Four-pin connector
  - VCC (3.3 volts)
  - Ground
  - SDA
  - SCL



- Daisy-chains through boards
  - Actually connects to chips in parallel as a bus

https://www.sparkfun.com/qwiic

System Management Bus (SMBus)

- Related communication specification
  - A little more strict in places, but generally interoperable

- Adds ability to broadcast or unicast messages
  - Generic addresses for Contoller and various peripherals (Battery)

- Adds an open-drain shared interrupt signal
  - High-impedance or pull low, just like SDA and SCL
  - Allows any device to alert a controller
    - Controller has to probe bus to determine which device wants attention

I2C use cases

- Various sensors
  - Usually low to medium speed
  - Even relatively high speed stuff often has I2C for convenience
    - Accelerometers and microphones
    - Often with intelligent filtering built in
- Communication between microcontrollers
  - Either can act as the Controller when necessary
- Commonly exists internally within smartphones and laptops too
  - Light sensors, Temperature sensors, etc.

I2C Pros and Cons

- Pros
  - Wiring is simple
  - Only uses two pins
  - Very widely supported

- Cons
  - Relatively slow communication rate
  - Speed versus power use tradeoff (due to pull-down resistor)
  - Open collector makes debugging difficult

Outline

- SPI
- I2C