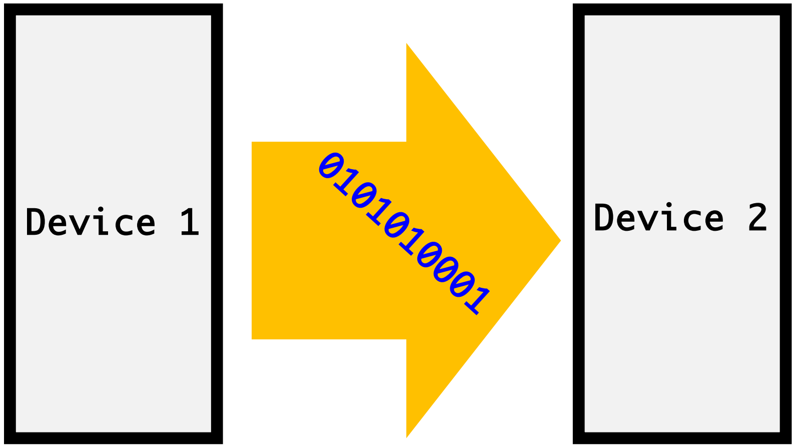
Communications and Networking

6.9000

Spring 2025

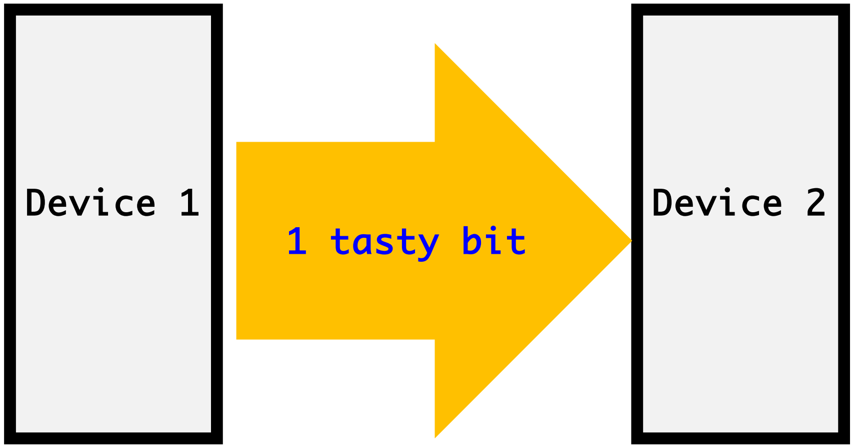
Send Information Between Devices

- All information is encoded using ones and zeroes (bits)
- How to do we convey ones and zeroes between entities?
  - What things do we vary in existence to achieve this?



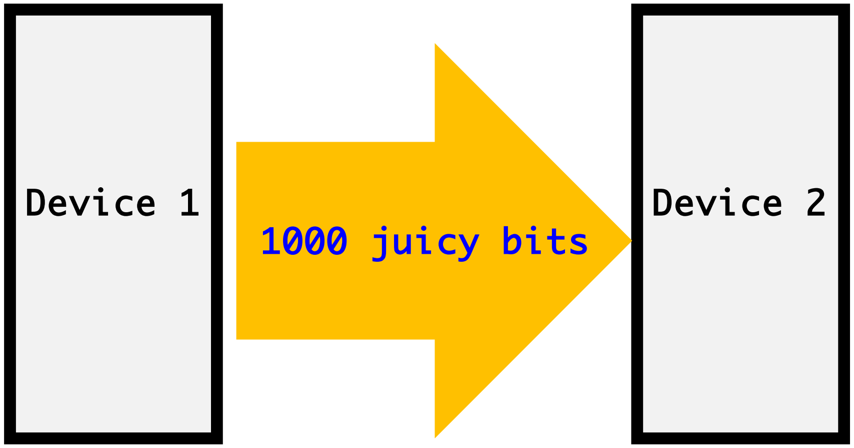
Sending Information

- If you have 1 bit of information to send how do you do it?



Sending Information

- If you have 1000 bits of information to send how do you do it?



In almost all systems

- Bits are conveyed using some mixture of:

- **Energy Patterns** that exist *in the three dimensions of space*
- **Time**…*the fourth dimension*

- Using these four dimensions let's us convey information

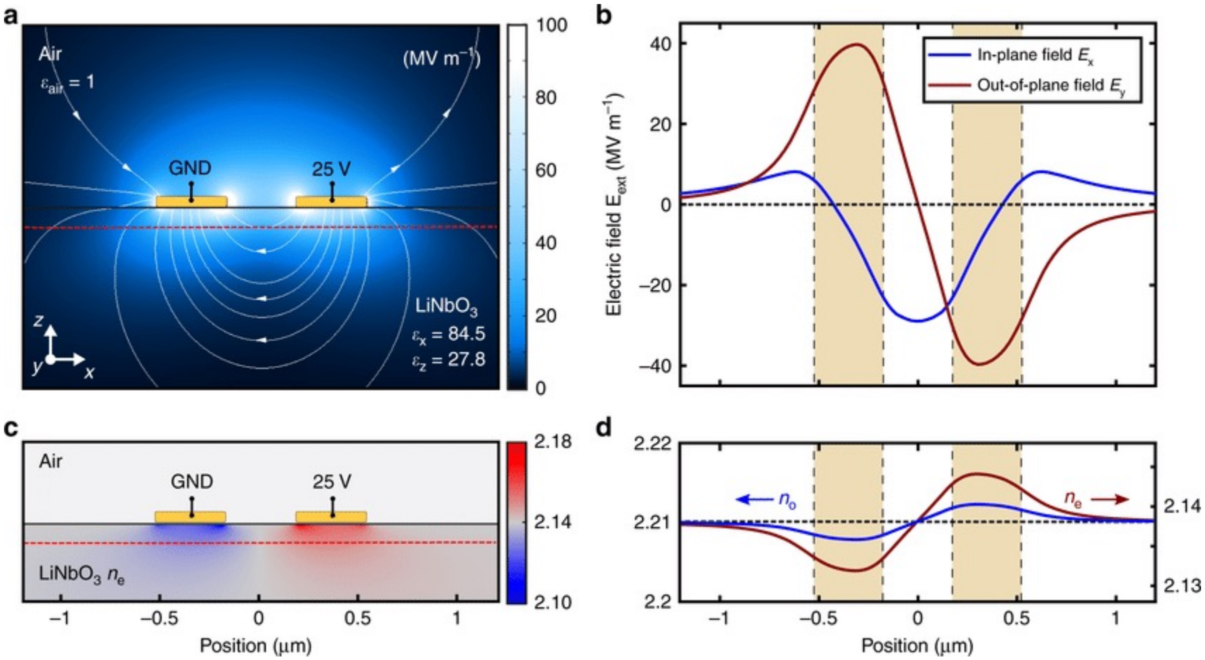
Electromagnetic Energy Patterns in Space

- Applying energy to Electromagnetic phenomena proved to be the most scalable and workable and is why Course 6 won out and Course 2 lost

- In order to make electromagnetic energy patterns in space we need to be able to:
  - Put energy in one area
  - Not put energy in another area

Conductors and Insulators

- In the case of "circuits" we use variations in the complex electric permittivity (ε) of different materials ***to focus*** electromagnetic field (with particular emphasis on the electric field).



https://www.researchgate.net/figure/Electrostatic-field-simulation-of-the-two-wire-configuration-a-Color-coded-electric-field_fig3_339076022

Convert one ability into another

- We're really good at making patterns of metal and not-metal…
- Which means we can be really good at making spatial electric energy patterns!
- Can do it manually using wires like a bunch of peasants (necessary in prototyping so no shame)

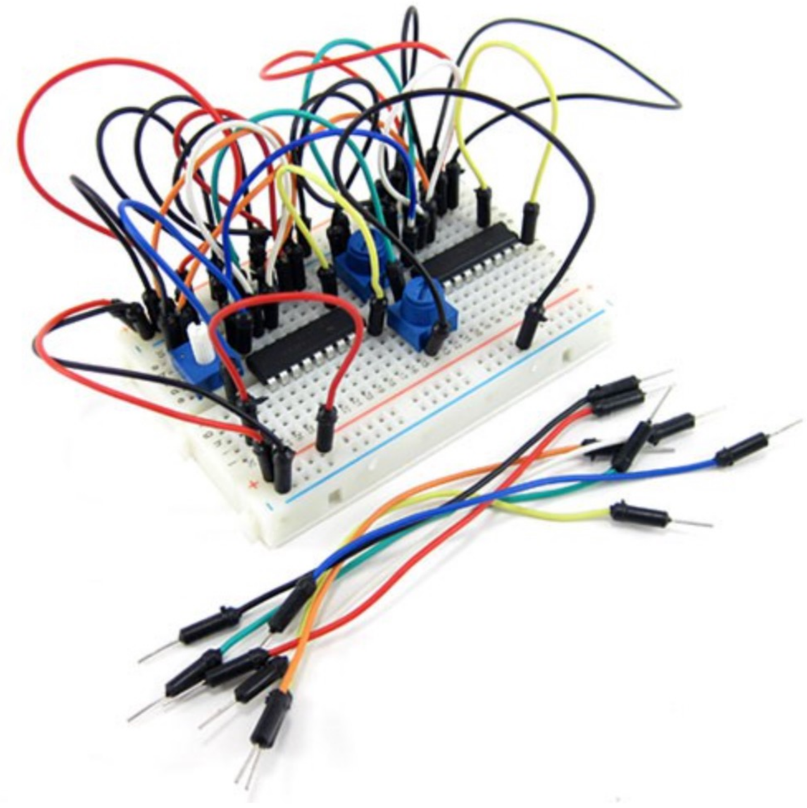
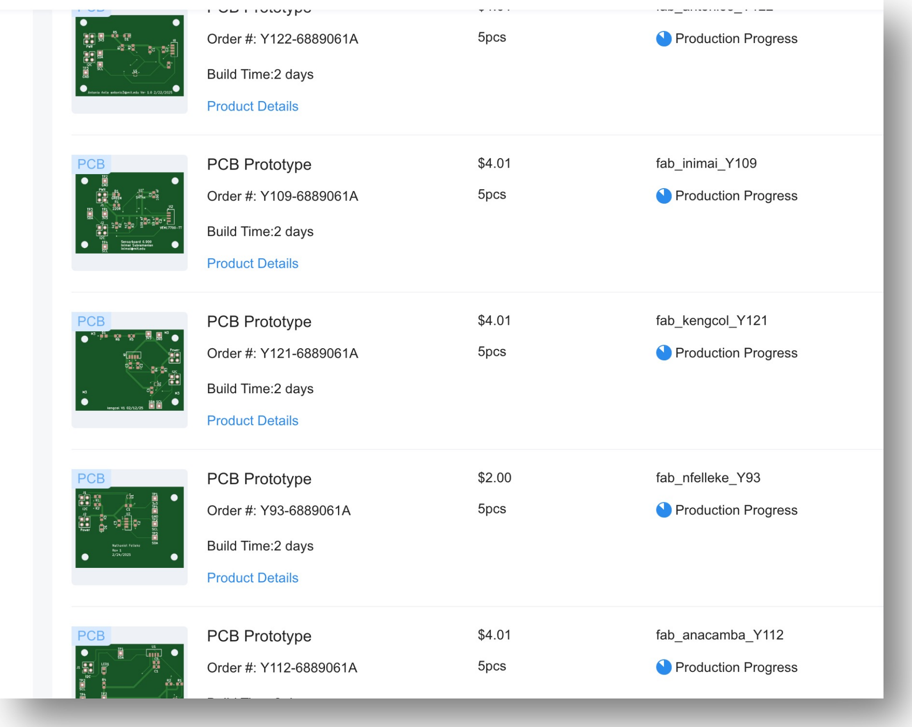


Image Credit: Ben Finio, Science Buddies / Science Buddies

https://www.sciencebuddies.org/science-fair-projects/references/how-to-use-a-breadboard

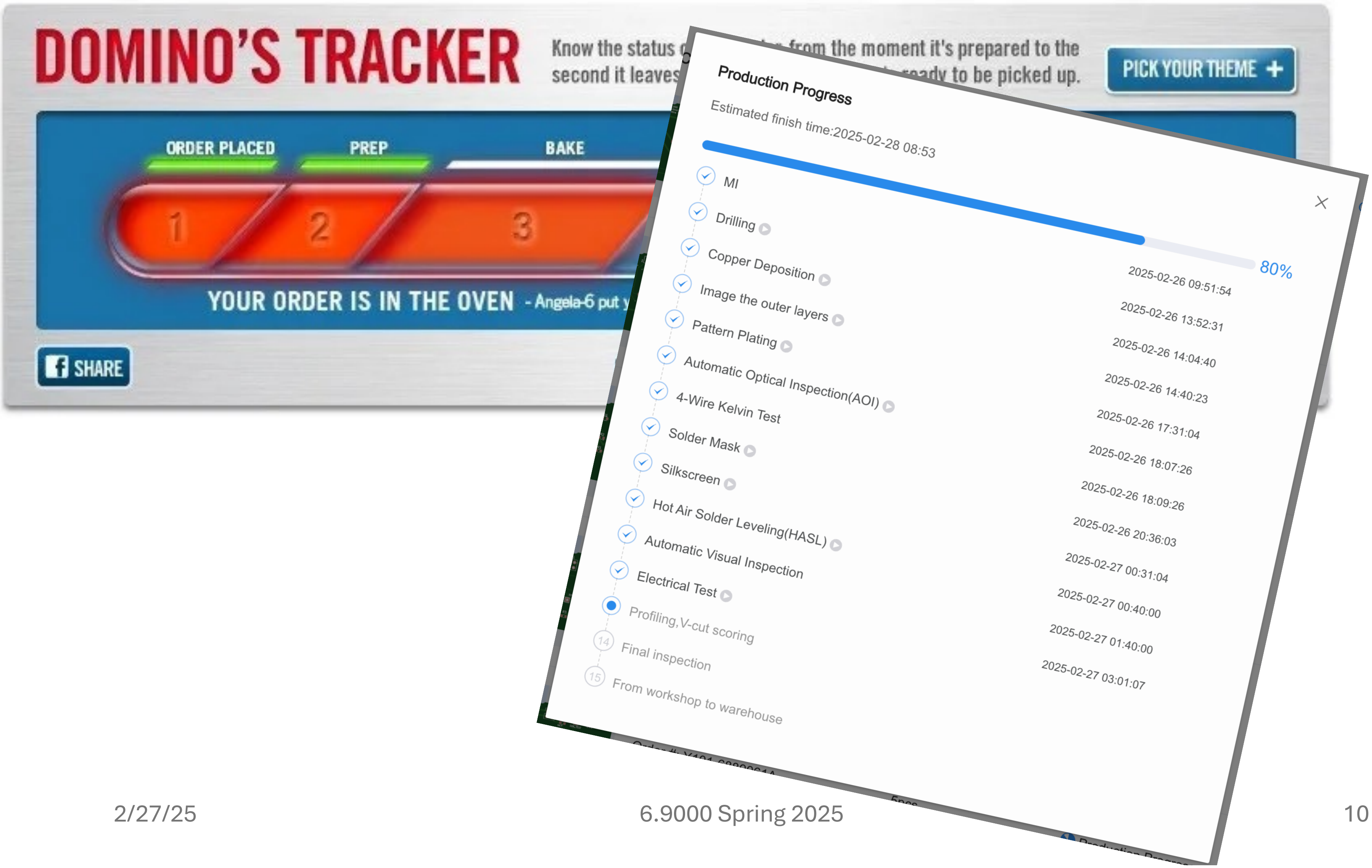
Convert one ability into another

- We’re really good at making patterns of metal and not-metal…
- Which means we can be really good at making spatial electric energy patterns!
- Can do it using CAD and fancy automated material fabrication techniques to make printed circuit boards



	Order #: Y122-6889061A Build Time:2 days Product Details	5pcs	Production Progress
PCB	PCB Prototype Order #: Y109-6889061A Build Time:2 days Product Details	$4.01 5pcs	fab_inimai_Y109 Production Progress
PCB	PCB Prototype Order #: Y121-6889061A Build Time:2 days Product Details	$4.01 5pcs	fab_kengcol_Y121 Production Progress
PCB	PCB Prototype Order #: Y93-6889061A Build Time:2 days Product Details	$2.00 5pcs	fab_nfelleke_Y93 Production Progress
PCB	PCB Prototype Order #: Y112-6889061A	$4.01 5pcs	fab_anacamba_Y112 Production Progress

Aside: PCBs are coming along!


DOMINO'S TRACKER
Know the status o… from the moment it's prepared to the
second it leaves … ready to be picked up.
PICK YOUR THEME +
ORDER PLACED
PREP
BAKE
1
2
3
YOUR ORDER IS IN THE OVEN - Angela-6 put y
SHARE

Production Progress

Estimated finish time:2025-02-28 08:53

80%

✓ MI	2025-02-26 09:51:54
✓ Drilling	2025-02-26 13:52:31
✓ Copper Deposition	2025-02-26 14:04:40
✓ Image the outer layers	2025-02-26 14:40:23
✓ Pattern Plating	2025-02-26 17:31:04
✓ Automatic Optical Inspection(AOI)	2025-02-26 18:07:26
✓ 4-Wire Kelvin Test	2025-02-26 18:09:26
✓ Solder Mask	2025-02-26 20:36:03
✓ Silkscreen	2025-02-27 00:31:04
✓ Hot Air Solder Leveling(HASL)	2025-02-27 00:40:00
✓ Automatic Visual Inspection	2025-02-27 01:40:00
✓ Electrical Test	2025-02-27 03:01:07
● Profiling,V-cut scoring	
14 Final inspection	
15 From workshop to warehouse	

In almost all systems

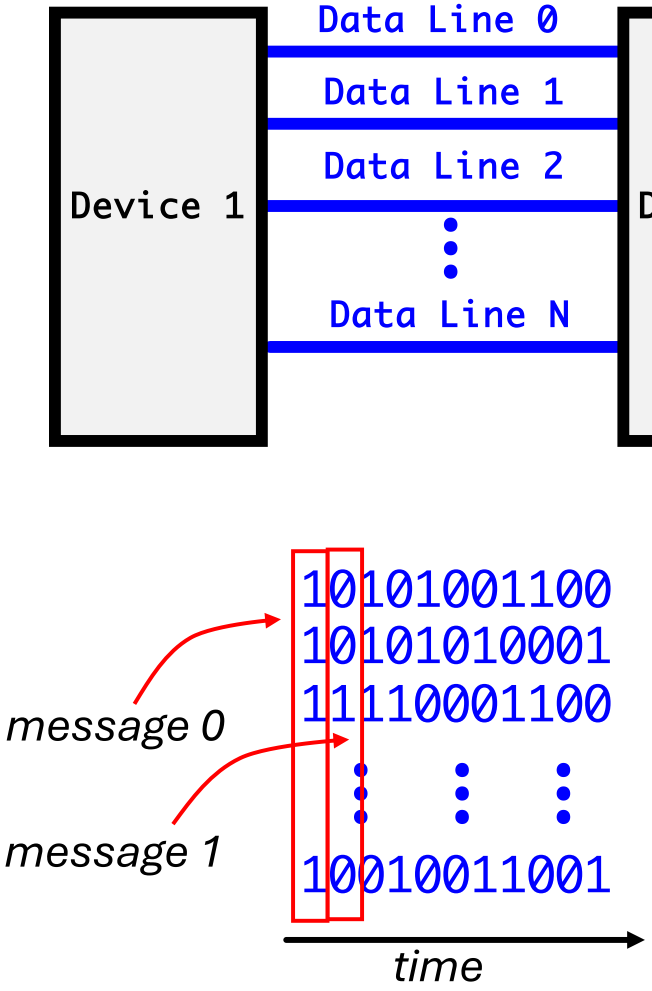
- Bits are conveyed using some mixture of:

- **Energy Patterns** that exist *in the three dimensions of space*
- **Time**…*the fourth dimension*

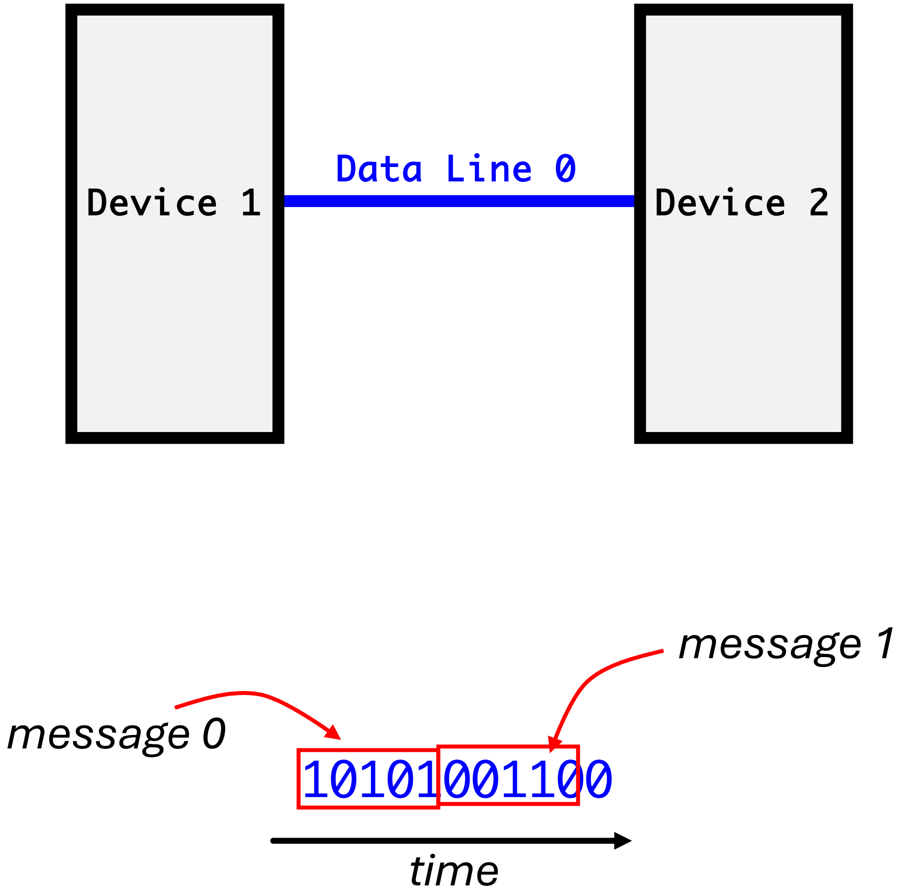
- Using these four dimensions let's us convey information

Sending Data…Using the dimensions

Parallel Link:



Serial Link:

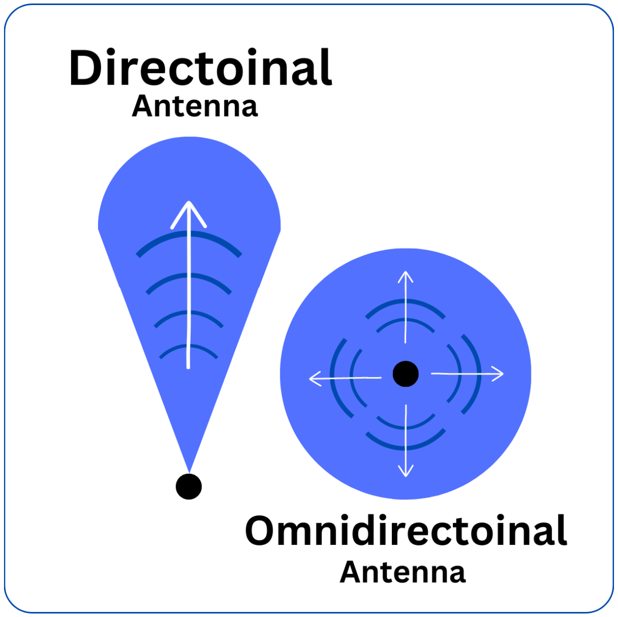


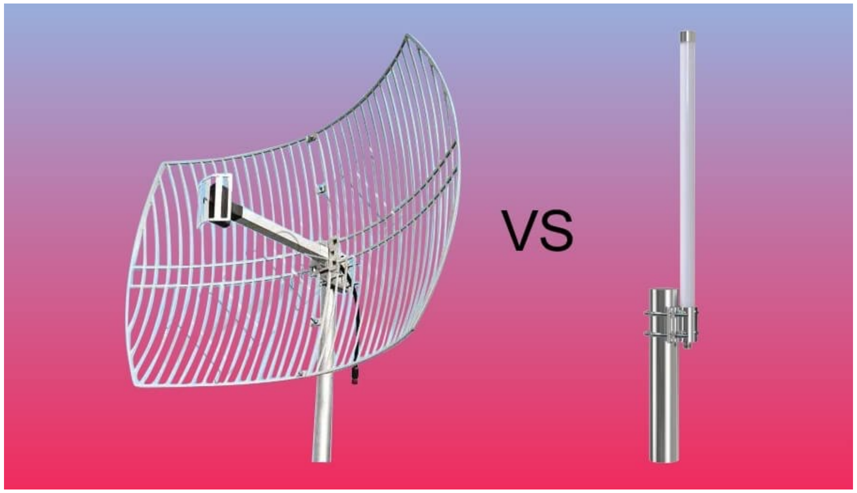
What about in Wireless/Radio?

- Any particular challenges applying the ideas so far to RF waves?

Spatial constraint in Wireless

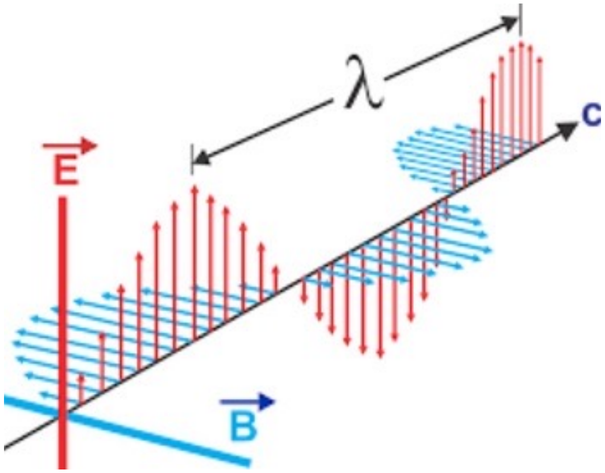
- You can directionally focus beams, but it is very, very hard to get the spatial resolution that we get with wires when working in the RF space





Wavelength of EM phenomena plays huge part here

- Wavelength is the spatial length of a electromagnetic wave



Speed of light (300,000 km/s

$$\lambda = \frac{c}{f}$$

Wavelength
(meters)

frequency
(Hz or cycles per second)

Wavelength vs. spatial features

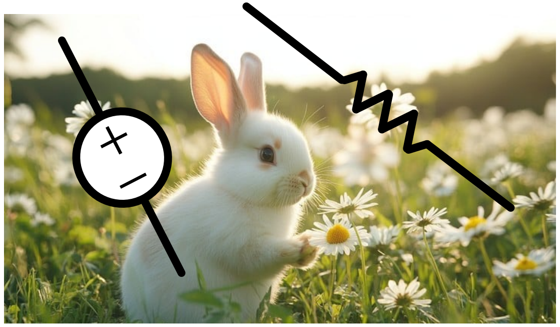
- When spatial features are *much, much smaller* than the wavelength of EM phenomena involved, it is very easy to get good spatial resolution of our electromagnetic fields
- Meaning…
  - Wires act like wires
  - Insulators act like insulators
  - Resistors act like resistors
  - Circuit

Some Wavelengths

$$\lambda = \frac{c}{f}$$

- 1 Hz signal: 300,000,000m
- 1 kHz signal: 300,000m
- 1 MHz signal: 300 m
- 10 MHz signal: 30 m
- 100 MHz signal: 3 m
- 1 GHz signal: 30 cm
- 10 GHz signal: 3 cm
- 100 GHz signal: 3 mm

“DC”



“RF”

In Very High Frequencies…

- It becomes harder and harder to spatially isolate EM signals using wires and things because stuff just doesn’t “behave” anymore.
- The size of physical features ends up being similar to the size of the EM wavelengths

In almost all systems

- Bits are conveyed using some mixture of:

- **Energy Patterns** that exist *in the three dimensions of space*
- **Time**…*the fourth dimension*

- Using these four dimensions let's us convey information

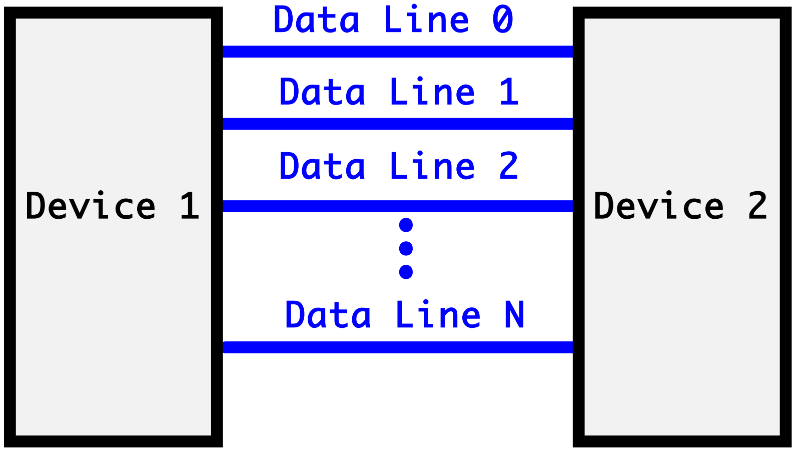
IN RF and High Frequency…

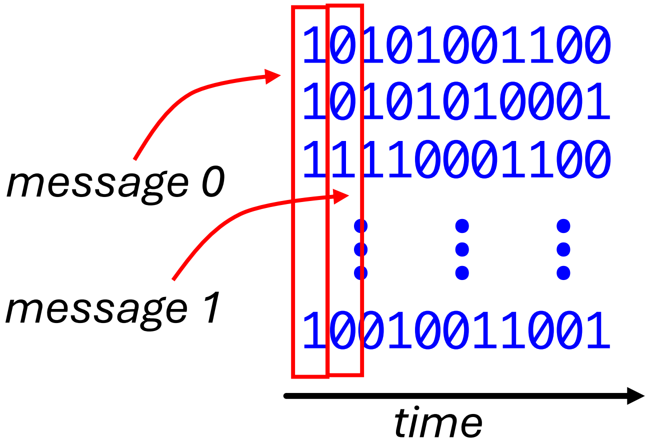
- **Energy Patterns** that exist *in the three dimensions of space*
- **Time**…*the fourth dimension*

- You kinda end up mixing time and space dimensions a bit more to rely on frequencies of the signals themselves to isolate/separate channels of information.

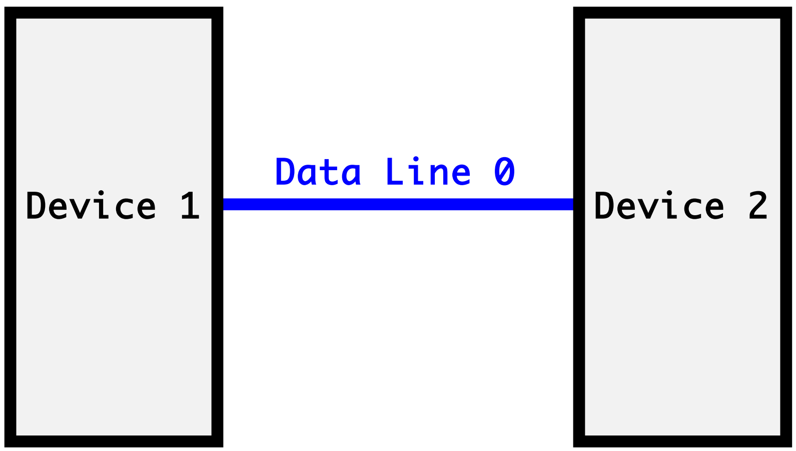
Data Transmission with Wires...

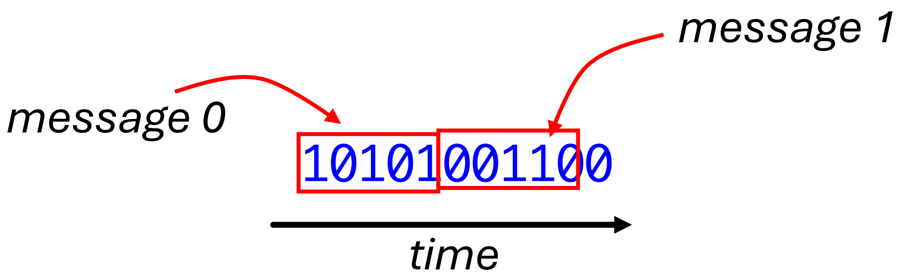
Parallel Link using Wires:





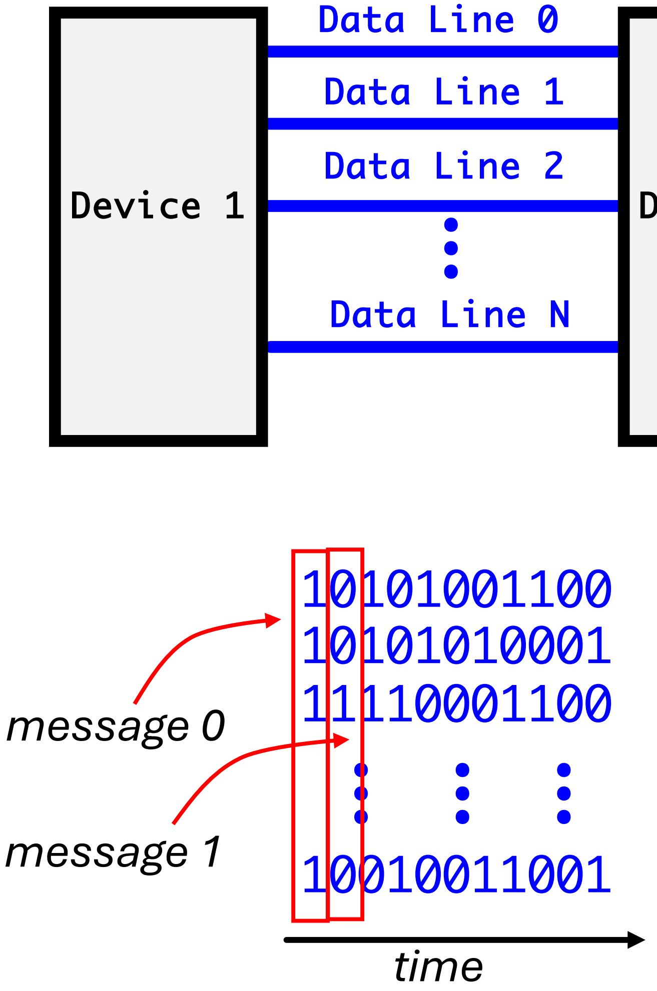
Serial Link using Wires:



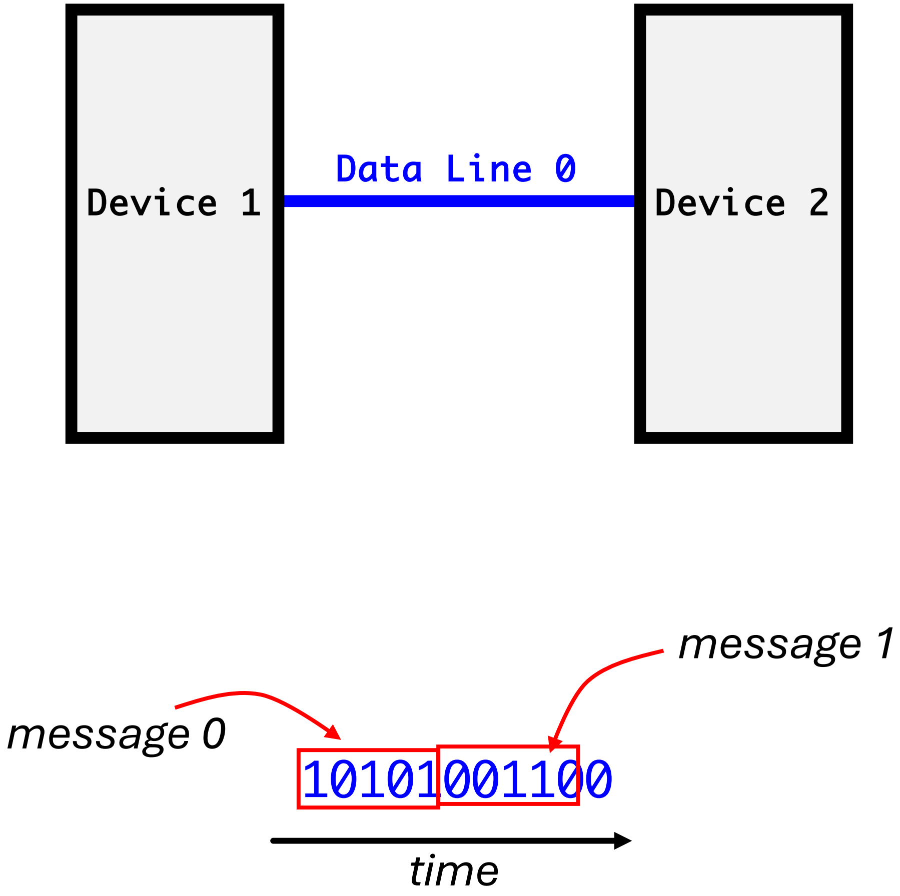


Data Transmission without Wires...

Parallel Link using frequency channels:



Serial Link using frequency channel:



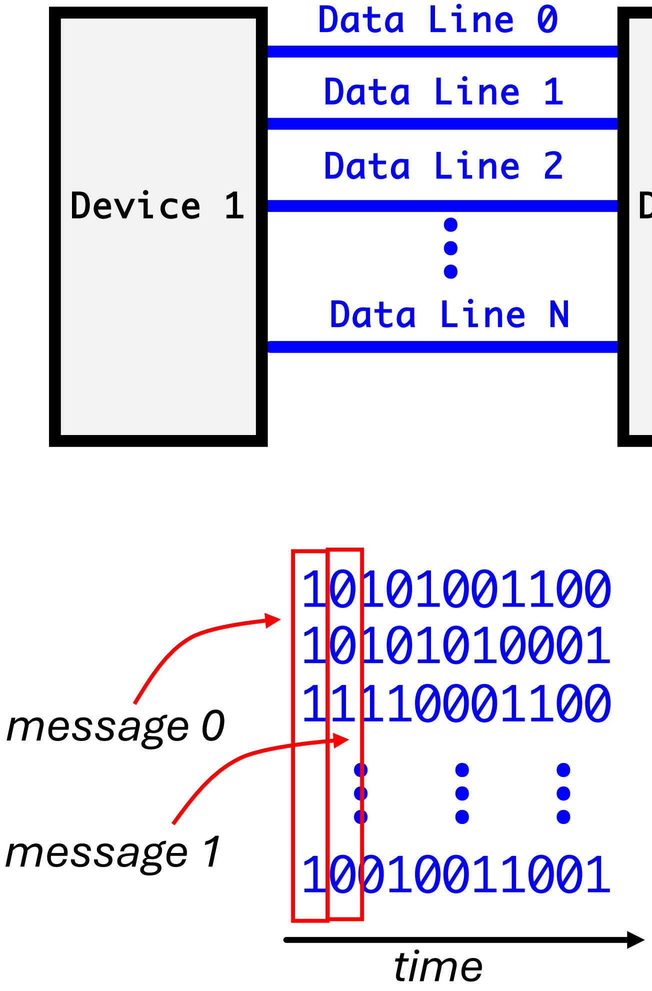
Anyways…

- Let’s talk about Wired Communication Protocols

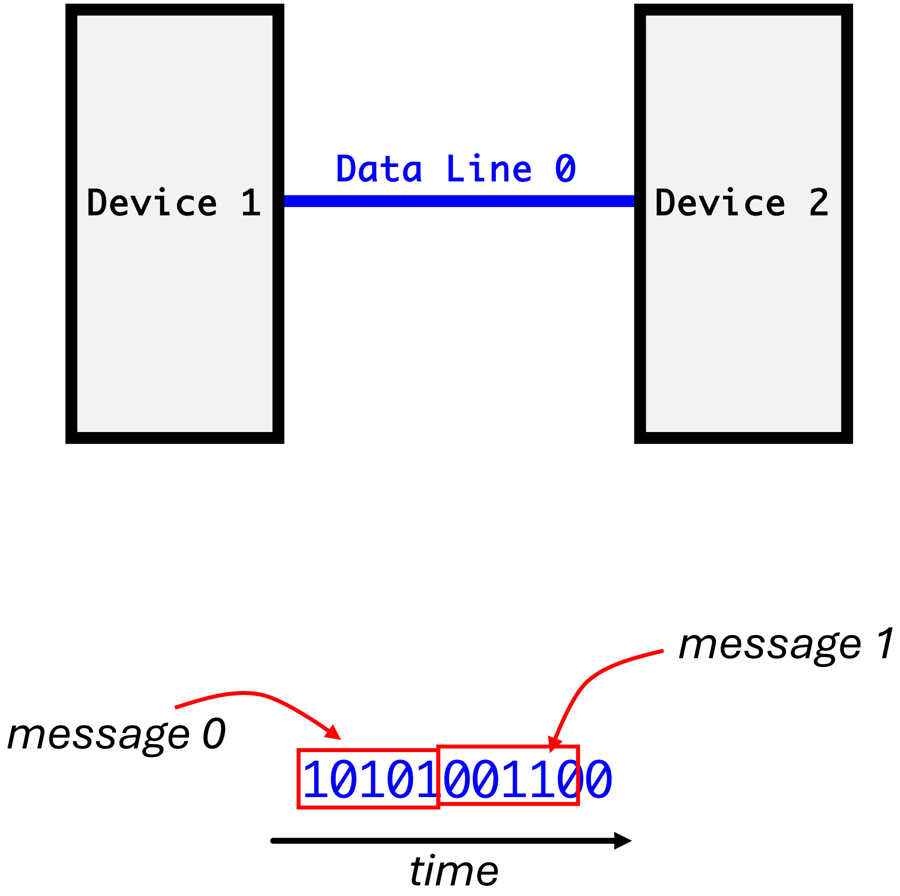
- But while doing so, keep wireless ideas in the back of your mind.

Data Transmission with Wires…

Parallel Link using Wires:



Serial Link using Wires:



Parallel vs. Serial in Wires

PARALLEL PROTOCOLS

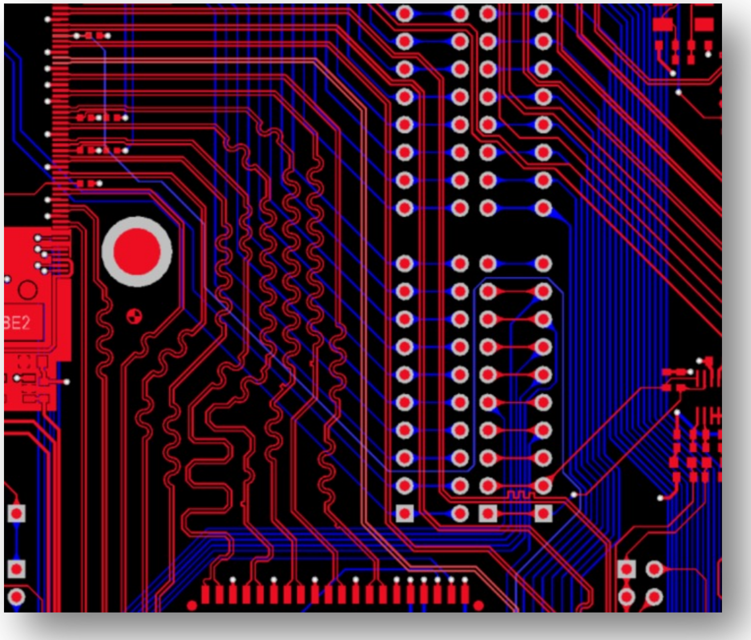
- **Parallel** (not so much on individual small devices)…mostly memory and things that need to send data at **very** high rates such as a camera, high-speed ADCs, etc…

SERIAL PROTOCOLS

- **UART "Serial"** very common
- **SPI (Serial Peripheral Interface)** very common
- **I2C (Inter-Integrated Circuit Communication)** very common

When Choose Parallel?

- When you need to transfer **large** amounts of data over short distances, parallel is a better choice.
- Data Transfer Rate will scale ~linearly with number of wires
- But Have to be careful of wiring length:
  - Ensure bits arrive same time
- Uses lots of space!!!

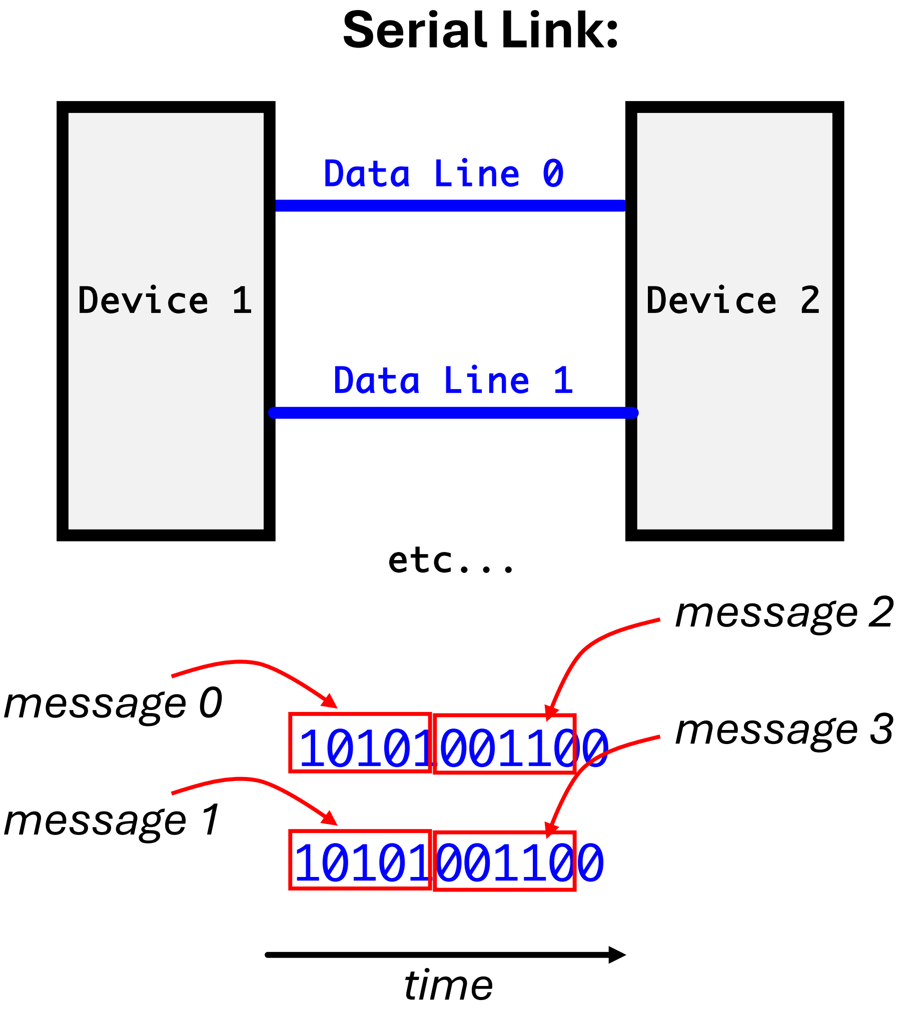


https://docs.toradex.com/102492-layout-design-guide.pdf

Communications Trends

- Serial: good for long distance (save on cable, pin and connector cost, easy synchronization). Requires “serializer” at sender, “deserializer” at receiver
- Parallel: issues with clock skew, crosstalk, interconnect density, pin count. Used to dominate for short-distances (eg, between chips).
- **BUT for *high data* movement, modern preference is for parallel, but independent serial links (eg, PCI-Express x1,x2,x4,x8,x16) as a hedge against link failures. Ethernet, USB, etc… these all follow that same pattern**

Multiple Serial Links in Parallel



- Multiple separate serial channels coexist.
- Generally data sent on each channel isn't intricately tied together (maybe separate packets/message)...n splitting bits across multiple wires

Serial Standards

- A zillion Serial standards
  - Asynchronous (no explicit clock) vs. Synchronous (CLK line in addition to DATA line).
  - Recent trend to reduce signaling voltages: save power, reduce transition times
  - Control/low-bandwidth Interfaces: SPI, I^{2}C, 1-Wire, PS/2, AC97, CAN, I2S,
  - Networking: RS232, Ethernet, T1, Sonet
  - Computer Peripherals: USB, FireWire, Fiber Channel, Infiniband, SATA, Serial Attached SCSI
  - Graphics: DVI, HDMI, DisplayPort

Common Chip-Chip Communication Protocols

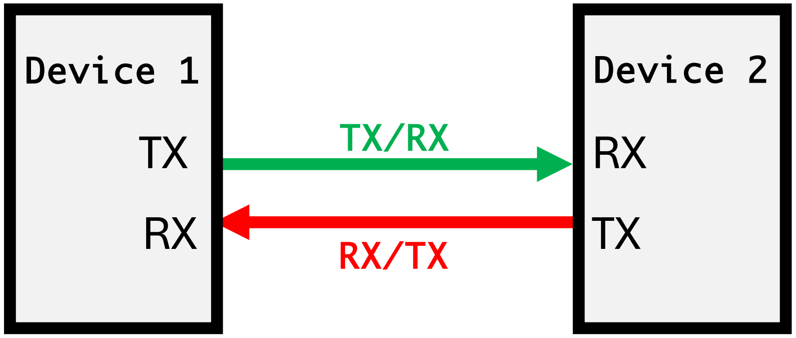
- Parallel (not super common, but exists in high speed situations).

- **UART (Universal Asynchronous Receive Transmit) "serial" very common**

- SPI (Serial Peripheral Interface) very common

- I2C (Inter-Integrated Circuit Communication) very common

UART aka "Serial"



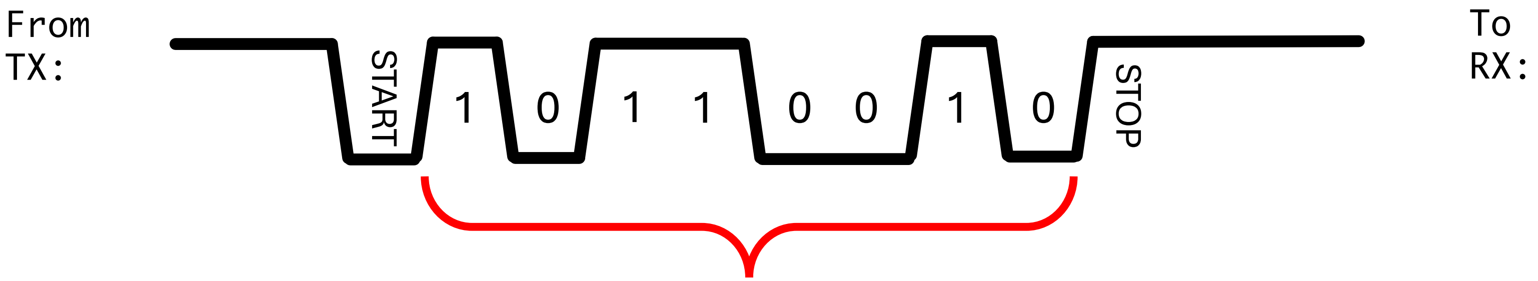
- Stands for Universal Asynchronous Receiver Transmitter
- Requires ***agreement ahead-of-time*** between devices regarding things like clock rate (BAUD), etc…
- Two wire communication for bi-directional (or one if you only want to talk and not listen like a bad relationship partner)
- Cannot really share
  - (every pair of devices needs own pair of lines so wires scales as $2n$ where n is the number of devices)
- Data rate generally < 1Mbps (though can maybe push a little bit)
- Data sent least significant bit (lsb) first

The Naming on UART is Perpetually a Mess with the TX/RX confusion

- When working with UART take care to pay attention to the TX and RX pins.
- They are complementary…one device’s TX talks to another devices RX.
- But boards and datasheets will sometimes label things backwards

UART

- Line High at rest ("high" an "low" depend on system specs…5V/0V…3.3V/0V, -12V/+12V…)
- Drops Low to indicate start
- 8 (or 9 bits follows) sent least significant bit first
- Goes high (stop bit)
- Can have optional parity bit for simple error correction



0x8d of 'M' in ASCII
sent lsb first!

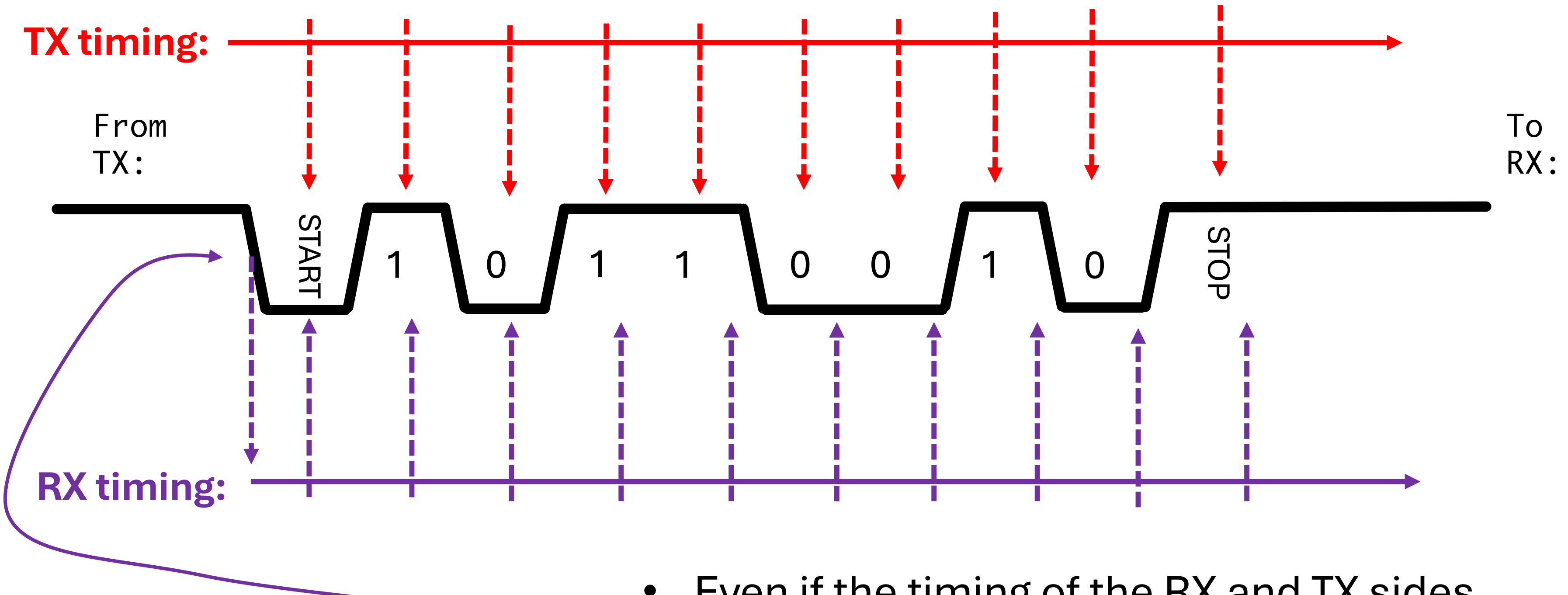
In UART, messages *must* be short (one byte)

- Both parties must agree ahead of time to a bit rate.
- A bit rate is bits per second
- Does everyone know what a second is?
- Does everyone *actually* know what a second is?
- What is a second?
- What are we even doing here?
- What are the implications of imperfect synchronization?

Timing Differences

- Atomic Clocks can range from $1500 to $200,000 depending on how good you want them.
- If we want commodity electronics to be cheap, $200,000 makes that hard to do.
- They must use “good-enough” local clocks and we build up communication protocols to accommodate for that.
- You must Synchronize your data transmission and reception

Synchronization

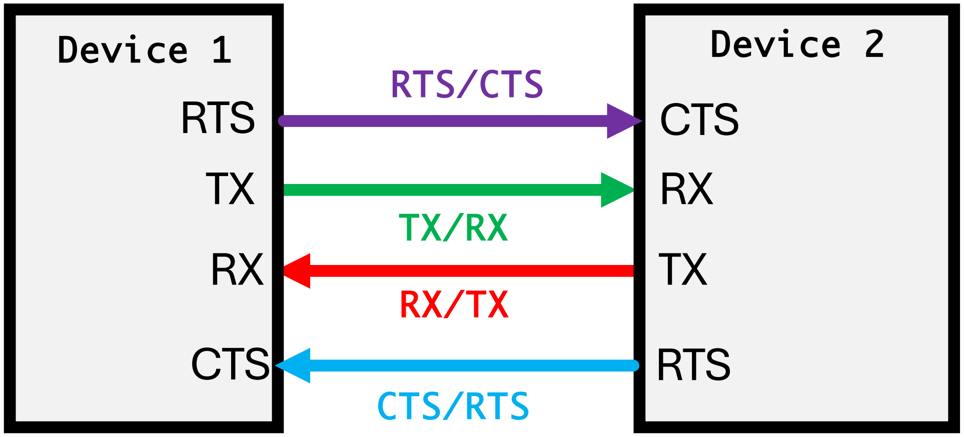


Receiver sees the high signal and waits for it to fall. From that edge it starts its timing

- Even if the timing of the RX and TX sides differ slightly, by keeping the messages short, the chance of getting too far out of sync is very, very low.
- Every new byte forces a resynchronization so errors never get a chance to accumulate too far!

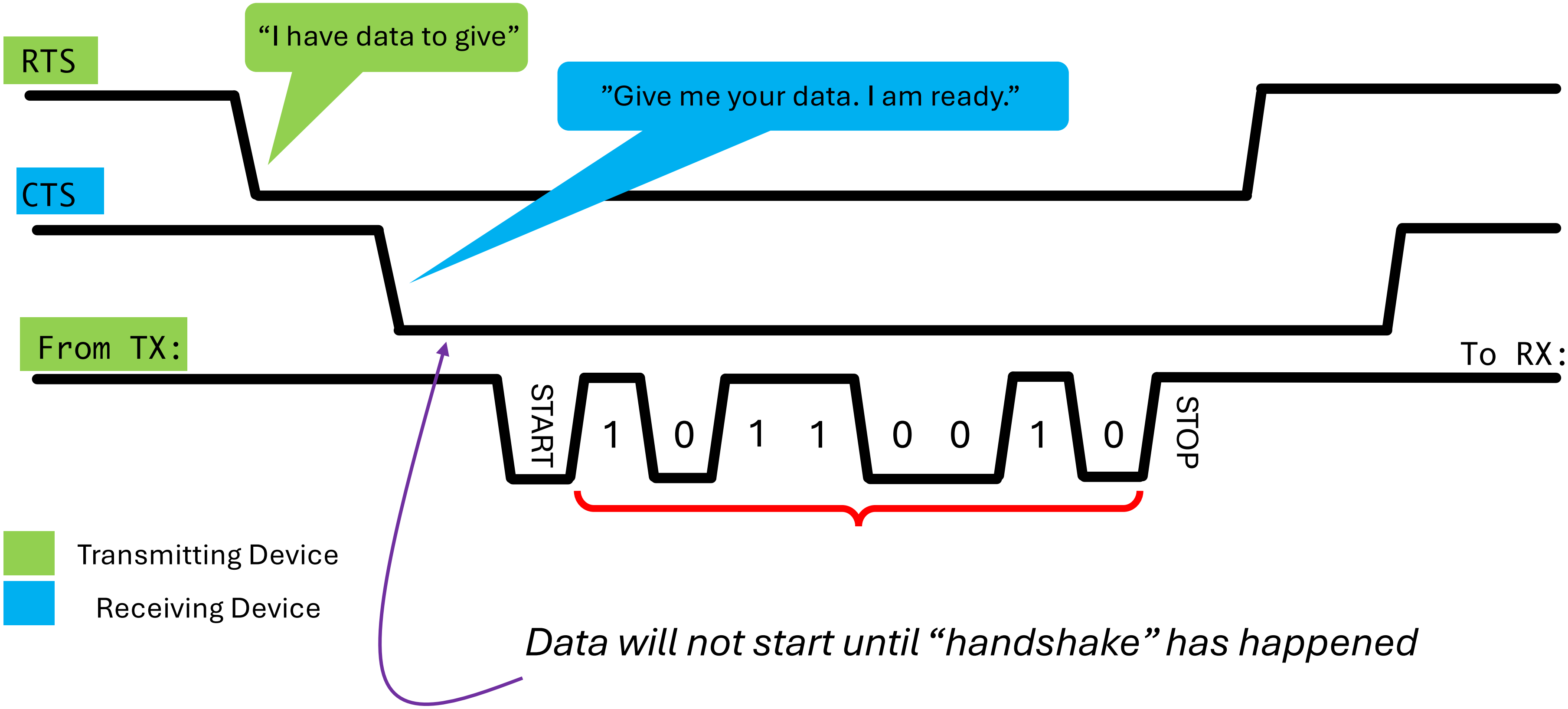
UART and RX/TX and RTS/CTS

- UART will also sometimes come with
  - “Ready to Send” signals (RTS)
  - “Clear to Send” signals (CTS)
- These are Flow-Control Signals that allow the two parties to tell each other if they have data to send if they are ready to receive data



UART Transmission

- RTS and CTS sit high. Each device in charge of setting the RTS and listening to the CTS
- Device pulls RTS low. Other device sees that and then pulls its CTS low in response



Data will not start until “handshake” has happened

UART Thoughts? Goods? Bads?

- Everything contained within one wire for the most part?

- Not super fast

Data Synchronization

- In UART, small data bursts with periodic resynchronizations are needed to make sure both parties produce and read data at the same time.

- How else to do this?

Common Chip-Chip Communication Protocols

- Parallel (not super common, but exists in high speed situations).

- UART (Universal Asynchronous Receive Transmit) “serial” very common

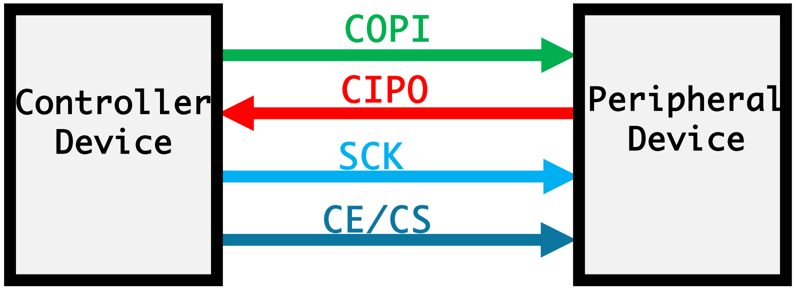
- **SPI (Serial Peripheral Interface) very common**

- I2C (Inter-Integrated Circuit Communication) very common

Note on Terminology

- Master/Slave terminology is heavily used in SPI and I2C…Master controls a bus, Slave listens.
- Acknowledge the issues with it, but also because many datasheets/vendors still use it, it is hard for us to separate from it completely.
- Changing slowly
- Maybe use ”Main”/”Secondary” to keep the letters the same or ”Controller” and “Peripheral”
- Also seeing SDO/SDI for “Serial Data Out/In” with respect to controlling device more recently

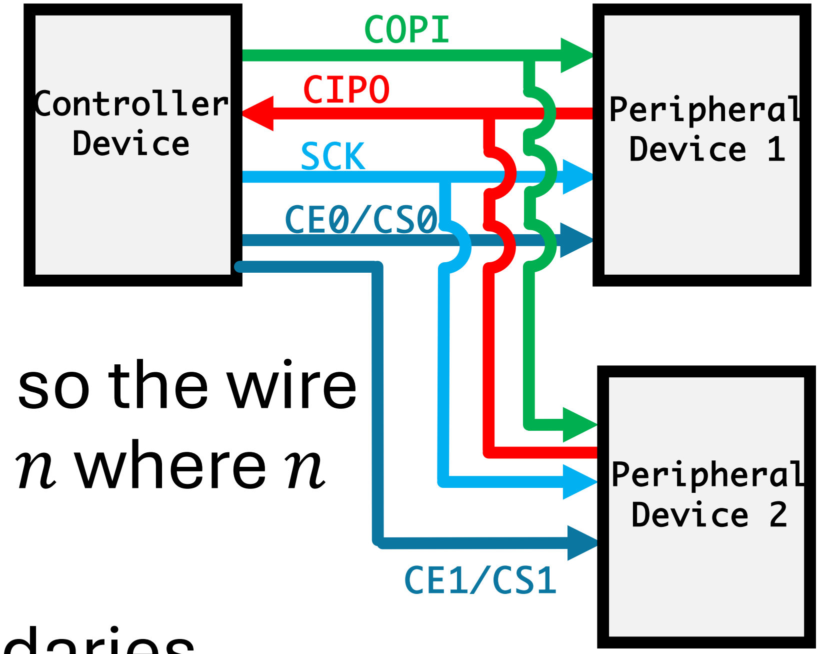
SPI



MOSI also = SDO "serial data out"
MISO also SDI "serial data in"
Also seeing now:
COPI = Controller Out Peripheral In
CIPO = Controller In Peripheral Out

- Stands for Serial-Peripheral Interface
- Four Wires:
  - COPI: Controller-Out-Peripheral-In...
  - CIPO: Controller-In-Peripheral-Out...
  - SCK: Serial Clock
  - CE/CS (Chip Enable or Chip Select)
- SCK removes need to agree ahead of time on data rate (from UART)...makes data interpretation much easier!
- High Data Rates: (1MHz up to ~70 MHz clock (bits))
- Data msb or lsb first...up to devices

SPI Expansion

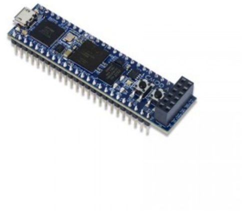


- Can share COPI/CIPO Bus so the wire requirement scales as $3 + n$ where n is the number of devices
- Addition of multiple secondaries requires additional select wires
- Hardware/firmware for SPI is pretty easy to implement:
  - Wires are uni-directional
  - Classic “duh” sort of approach to digital communication, but very robust.

SPI Example

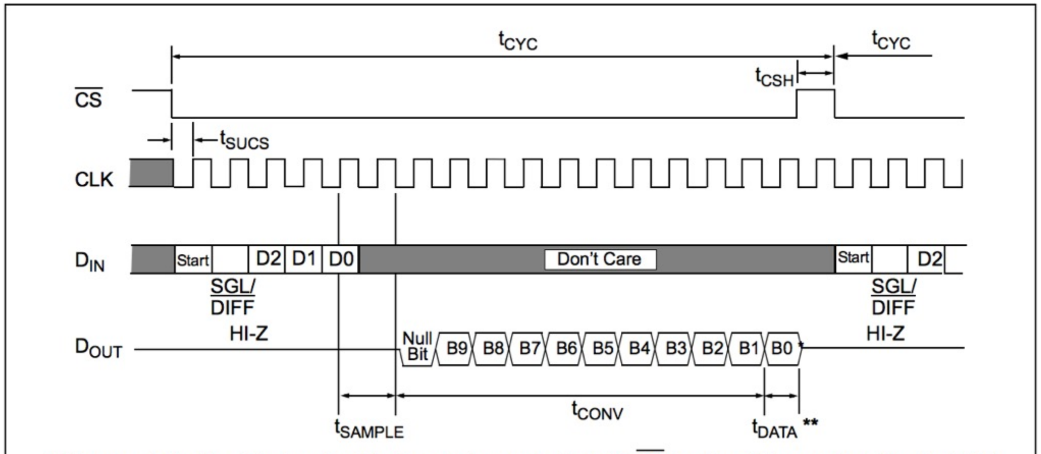
MCP3008 is a 8-channel 10 bit Analog to Digital Converter from Microchip Semi that communicates over SPI

CMOD-A7-35T



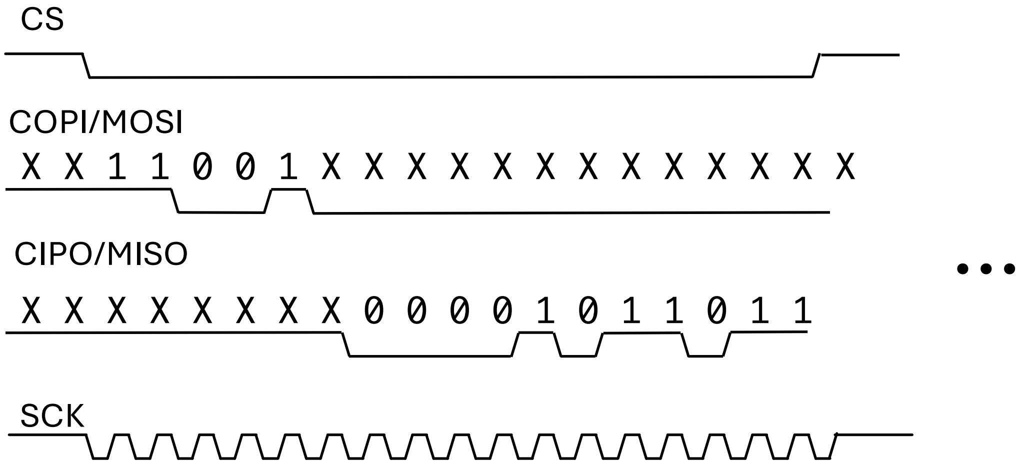
MCP3008

From MCP3008 Datasheet



* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the A/D converter will output LSB first data, then followed with zeros indefinitely. See Figure 5-2 below.

** t_{DATA}: during this time, the bias current and the comparator powers down while the reference input becomes a high-impedance node.



CS

COPI/MOSI

X X 1 1 0 0 1 X X X X X X X X X X X X

CIPO/MISO

X X X X X X X X 0 0 0 0 1 0 1 1 0 1 1

SCK

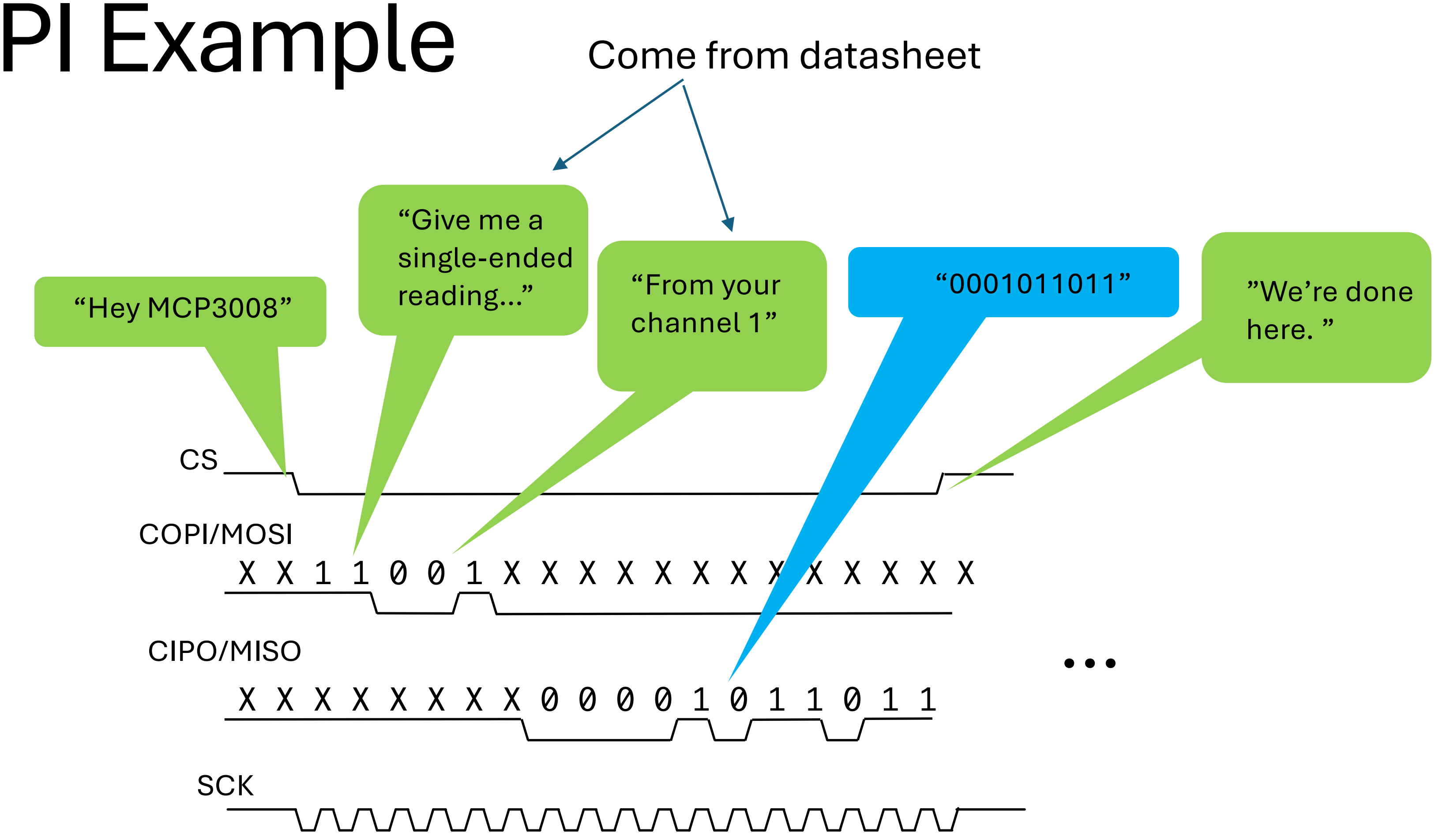
...

Here I am talking to a MCP3008 10 bit ADC

Sends its data **msb** first

Not all devices do this (must check datasheet)

SPI Example



🟩 (Controller/Main Device) Dialog

🟦 MCP3008 (Peripheral/Secondary Device) Dialog

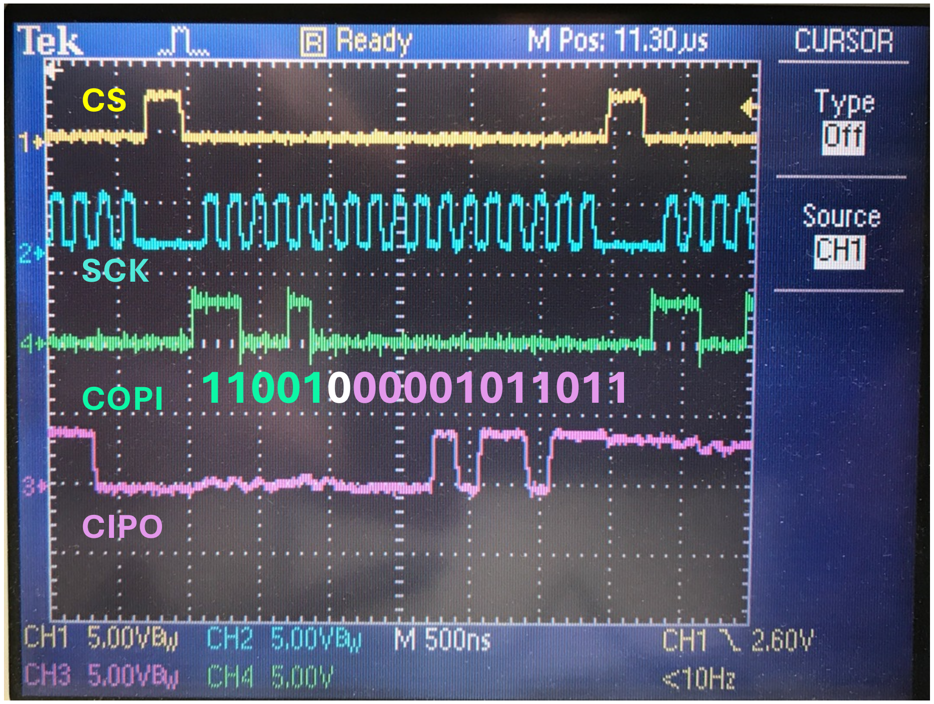
X means don’t care as in could be 1 or 0

SPI In Real Life

- Here I am talking to the same chip I was daydreaming about talking to on the previous slide.

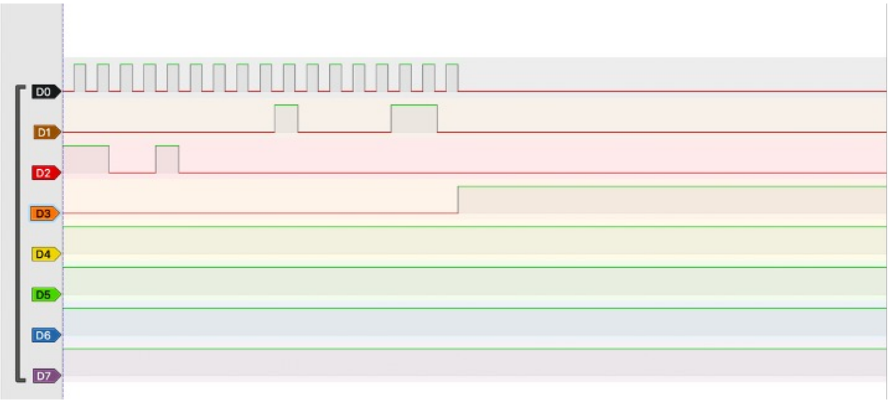
- Dreams do come true

- I’m saying, “give me your measurement on Channel 1,” and it is responding with “10’b0001011011” mapped to 3.3V or 0.293 V

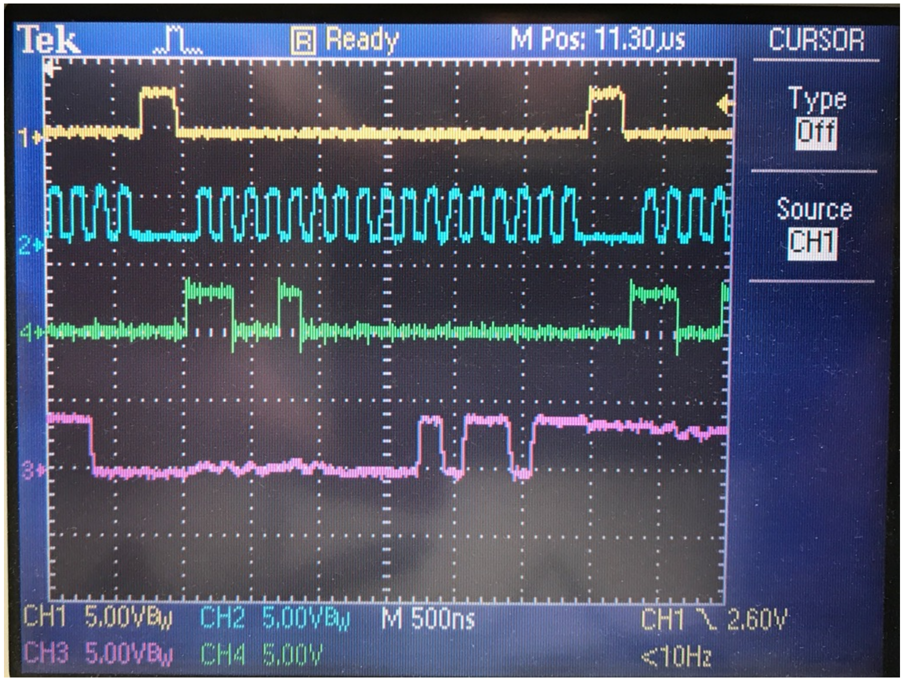


REMINDER: Digital In Analog Life vs Digital in Digital Life

- *What noise? I don’t care about noise (within reason)*

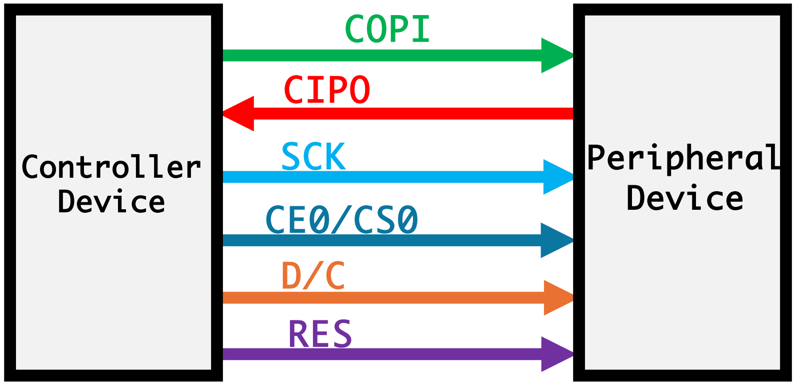


Logic Analyzer Capture of SPI transaction



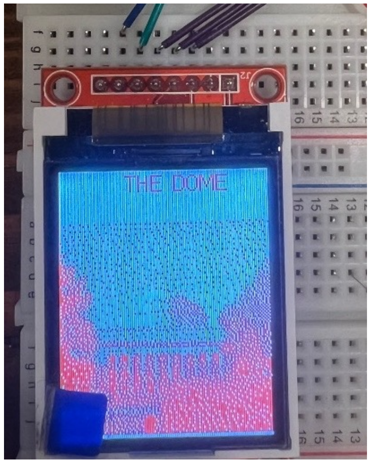
Oscilloscope Analog Capture of different SPI transaction

SPI Variations



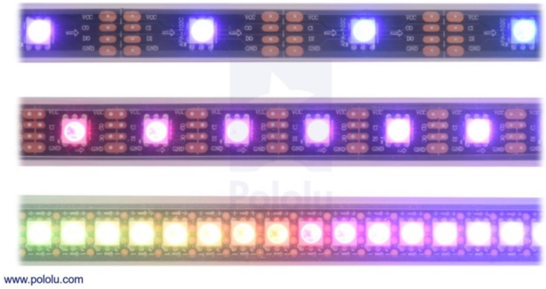
- Six Wires:
  - COPI: Controller-Out-Peripheral-In
  - CIPO: Controller-In-Peripheral-Out
  - SCK: Clock
  - CE/CS (Chip Enable or Chip Select)
  - RES: Reset Device
  - D/C: Data/Command (often seen in devices where you need to write tons of data (i.e. a display)

LCD Display:

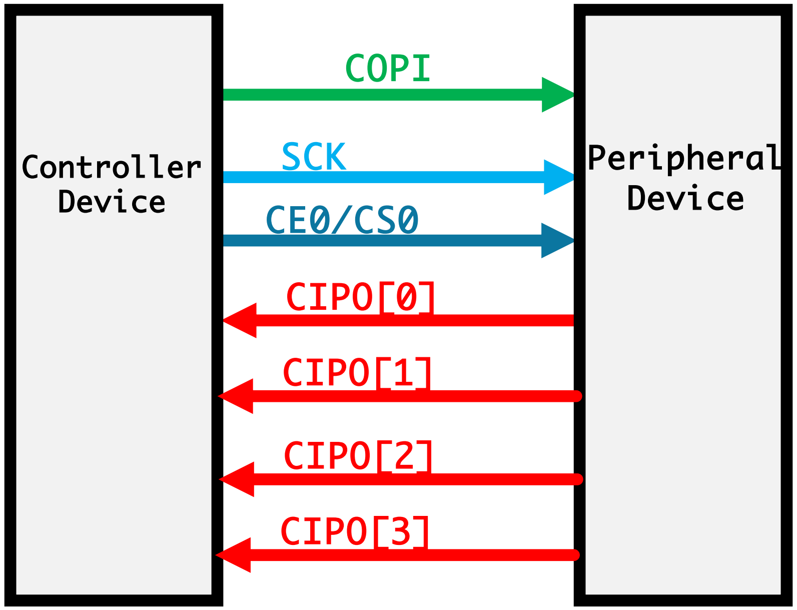


- Three/Two Wires:
  - If a device has nothing to say, drop CIPO:
  - If you assume only one device on bus drop CE/CS, so only have SCK and COPI, sometimes just called “DO” (for data out) in this situation

Twitch Streamer LEDs:

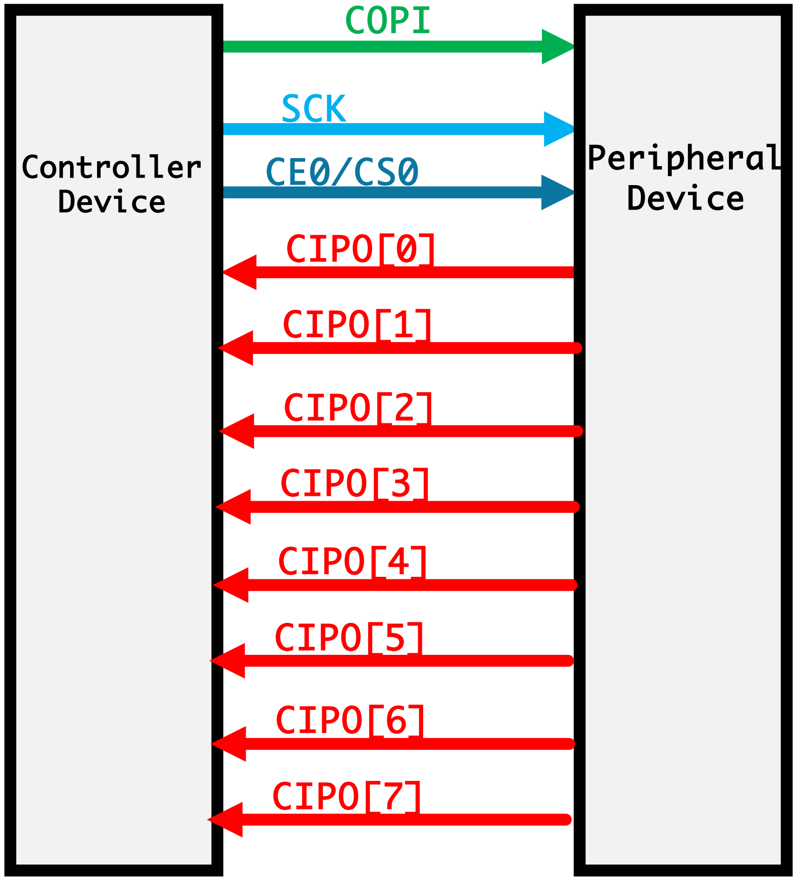


Other SPI Variations



- QSPI: “Quad SPI”
- This is basically SPI…
- But there will be four data transfer pins instead of one
- See in a lot of flash memory chips
- This really isn’t “Serial” in the way God meant it though…the bits are usually sampled together so it is a parallel data transfer just poorly named

Other SPI Variations



- 8SPI: “Octal SPI”
- This is basically SPI…
- …
- But there will be eight data transfer pins instead of one
- See in a lot of weird hybrid RAM chips
- Again if this isn’t parallel data transmission, I don’t know what is…but it is called SPI

SPI Conclusions

- It is a very simple and very robust ”idea” of a protocol.
- The simplicity comes at the expense of any wires
- And often…some of the complexity is deferred…There are many variations and dialects of it, so you should always always always read the datasheet for these things.

Be careful! Read Datasheets

- A bit screw-up point is mixing up clock polarity...sometimes data is sampled on rising edge...others on falling edge

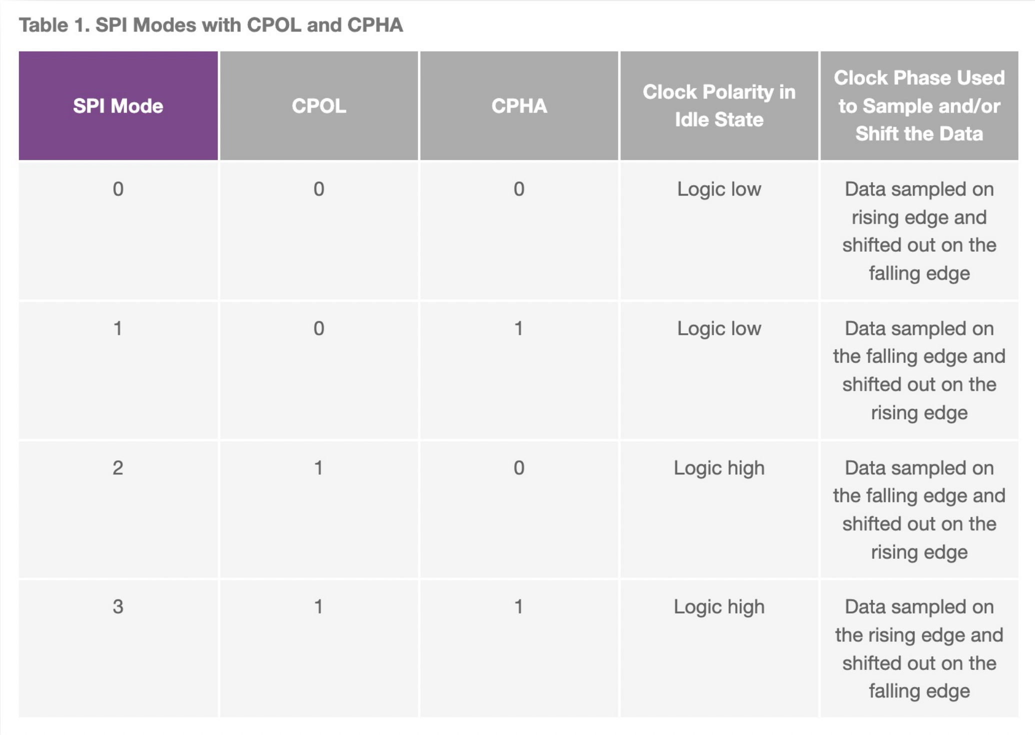


Table 1. SPI Modes with CPOL and CPHA

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic high	Data sampled on the rising edge and shifted out on the falling edge

- Clock level at idle can also matter sometimes

SPI Upsides?

- Simple to Implement
- Capable of Very High Speeds (50 MHz is not rare for some displays)

SPI Downsides?

- A lot of wires…which might seem like nbd, but in reality pincount is a huge cost factor in chip manufacture…tons of economic pressure to minimize this.

- At very high speeds it actually gets really noisy especially QSPI or OSPI

Common Chip-Chip Communication Protocols

- Parallel (not super common, but exists in high speed situations).

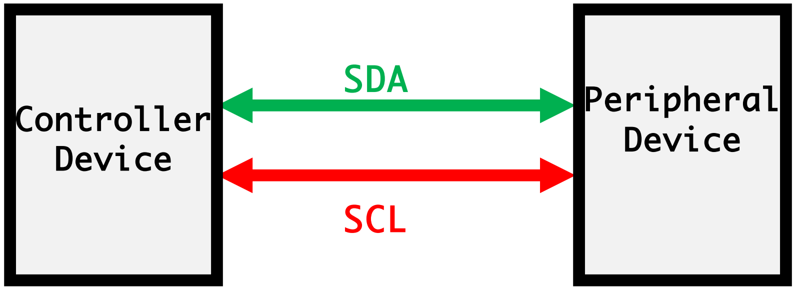
- UART (Universal Asynchronous Receive Transmit) very common

- SPI (Serial Peripheral Interface) very common

- **I2C (Inter-Integrated Circuit Communication) very common**

I2C

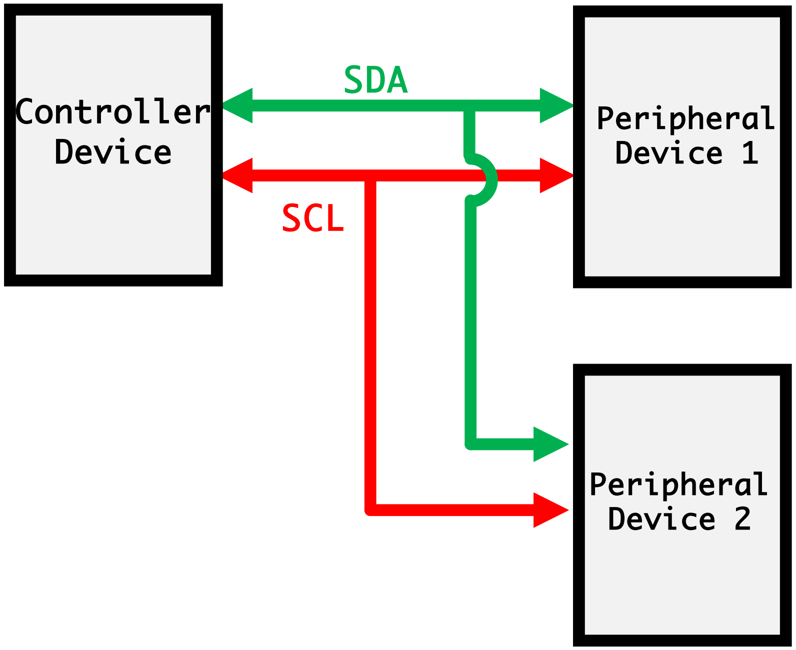
- Stands for Inter-Integrated Circuit communication
- Invented in 1980s
- Two Wire, One for Clock, one for data…Both wires are technically bidirectional, meaning each side can use them
- Usually 100kHz or 400 kHz clock (newer versions go to 3.4 MHz)



On i2C Multiple Devices Require Same # of Wires

- Devices come with their own ID numbers (originally a 7 bit value but more modern ones have 10 bits)…allows potentially up to 2^7 devices or 2^{10} on a bus (theoretically anyways)

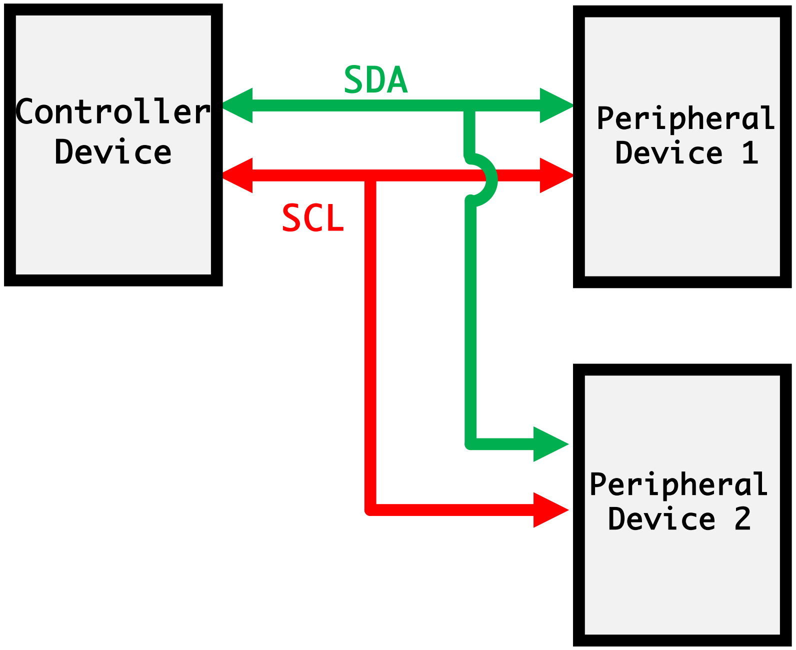
- ID's are specified at the factory*, usually several to choose from when you implement and you select them by pulling external pins HI or LOW



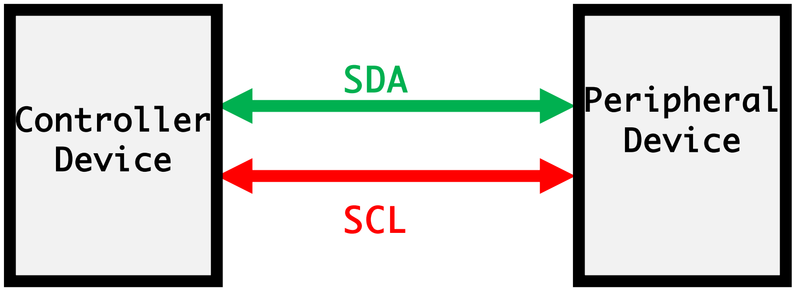
*sometimes programmable

I2C

- Only two wires…one used for synchronizing data and one used for conveying data in both directions:
  - Controller → Peripheral
  - Peripheral → Controller
- And also you need to let multiple devices possibly speak and listen…
- There's a lot here…
- It needs more complicated:
  - Hardware
  - Communication Protocols



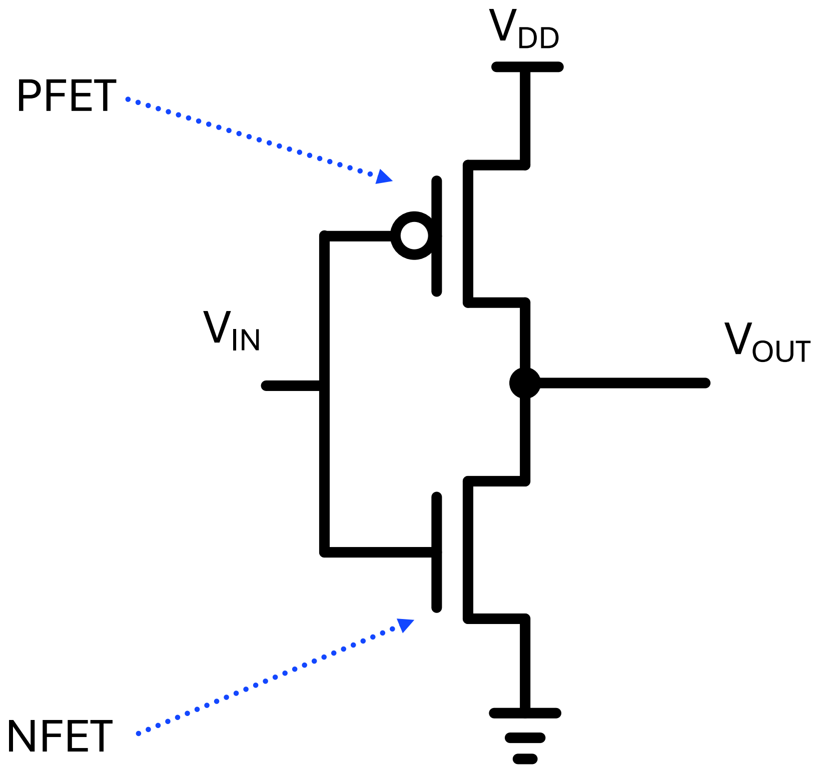
Bi-Directional Communication



- Hey those arrows are going both ways…how does that work?

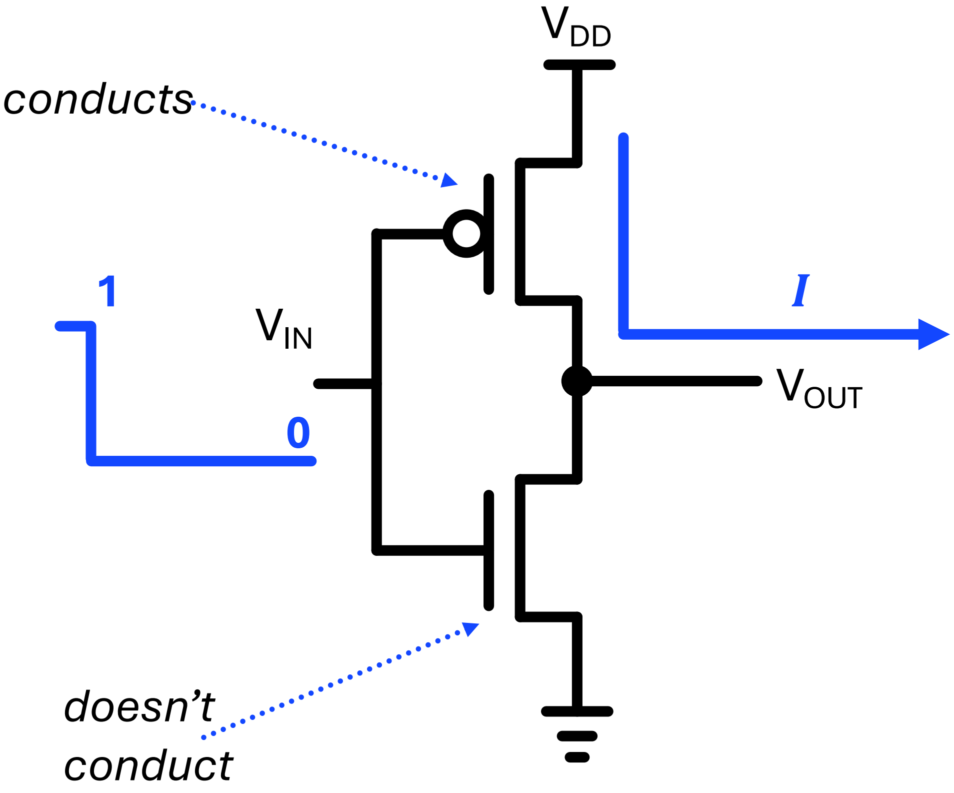
How Do Digital Electronics Set Voltages on a Line?

- We use CMOS Logic
- A pair of complementary transistors that can alternately connect and isolate from VDD and Ground



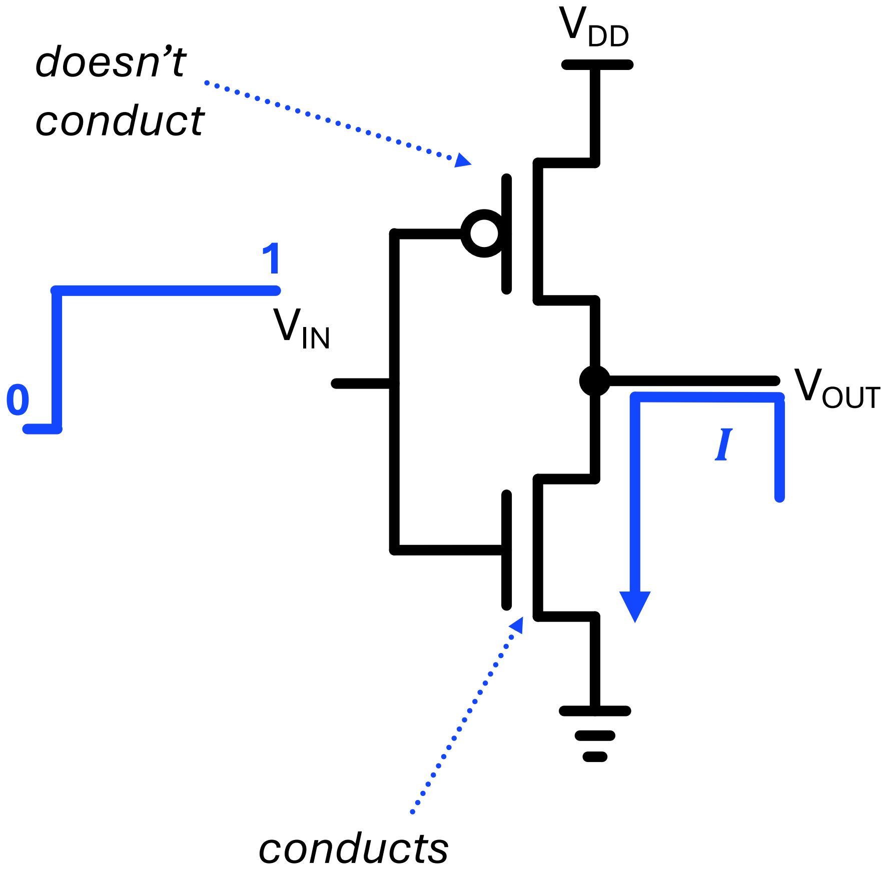
Put 1 on output?

- Low input
- PMOS conducts
- NMOS no conduct



Put 0 on output?

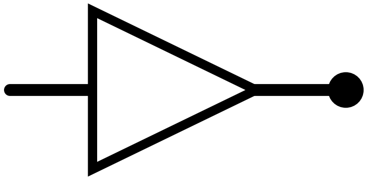
- High input
- PMOS no conducts
- NMOS conduct



How Do Digital Electronics Listen to Voltages on a Line?

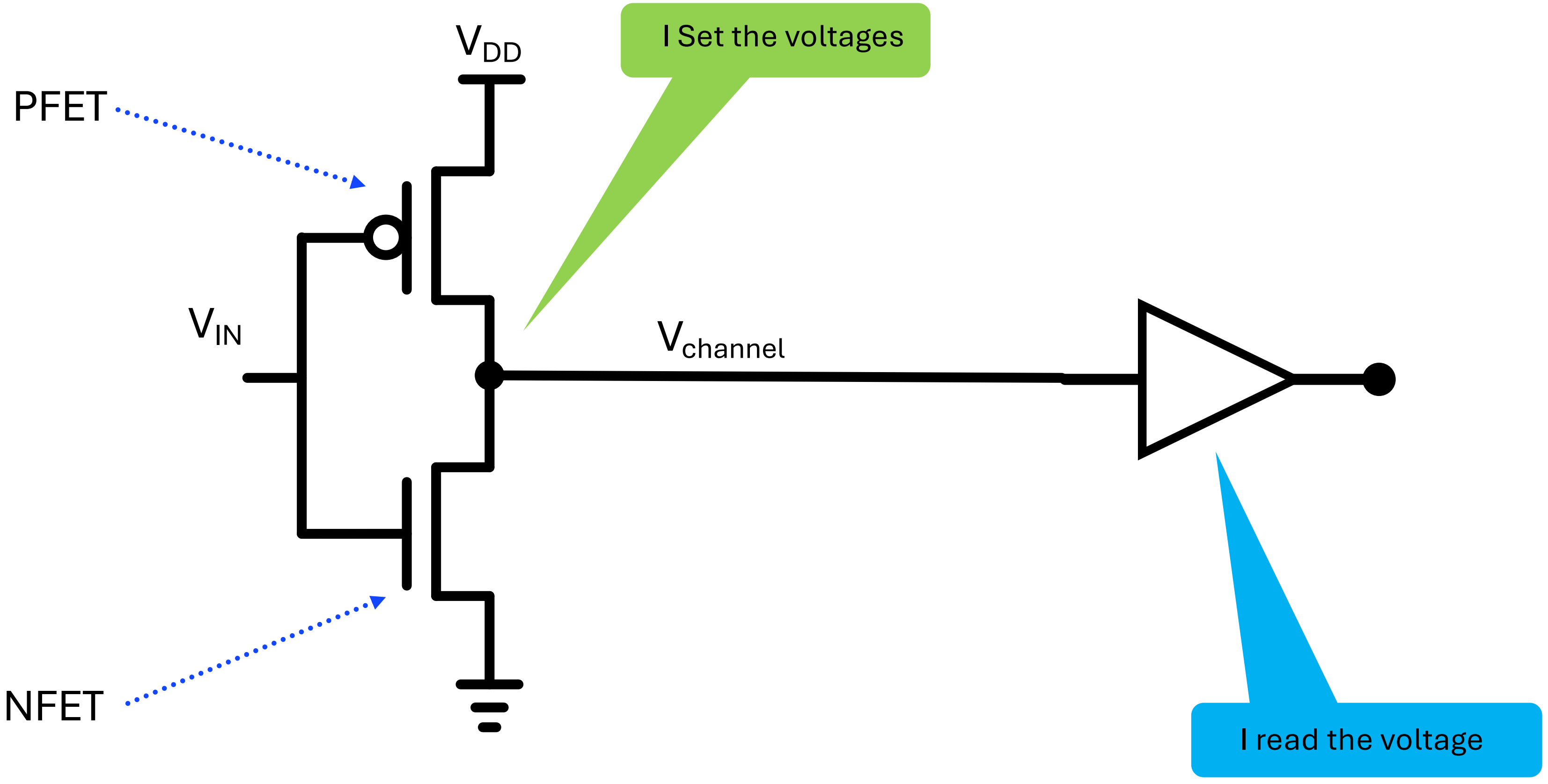
- Some sort of buffer

- Its input takes very little power/current from the line



- A quiet, ideal observer

So in Unidirectional Communication Schemes…



I Set the voltages

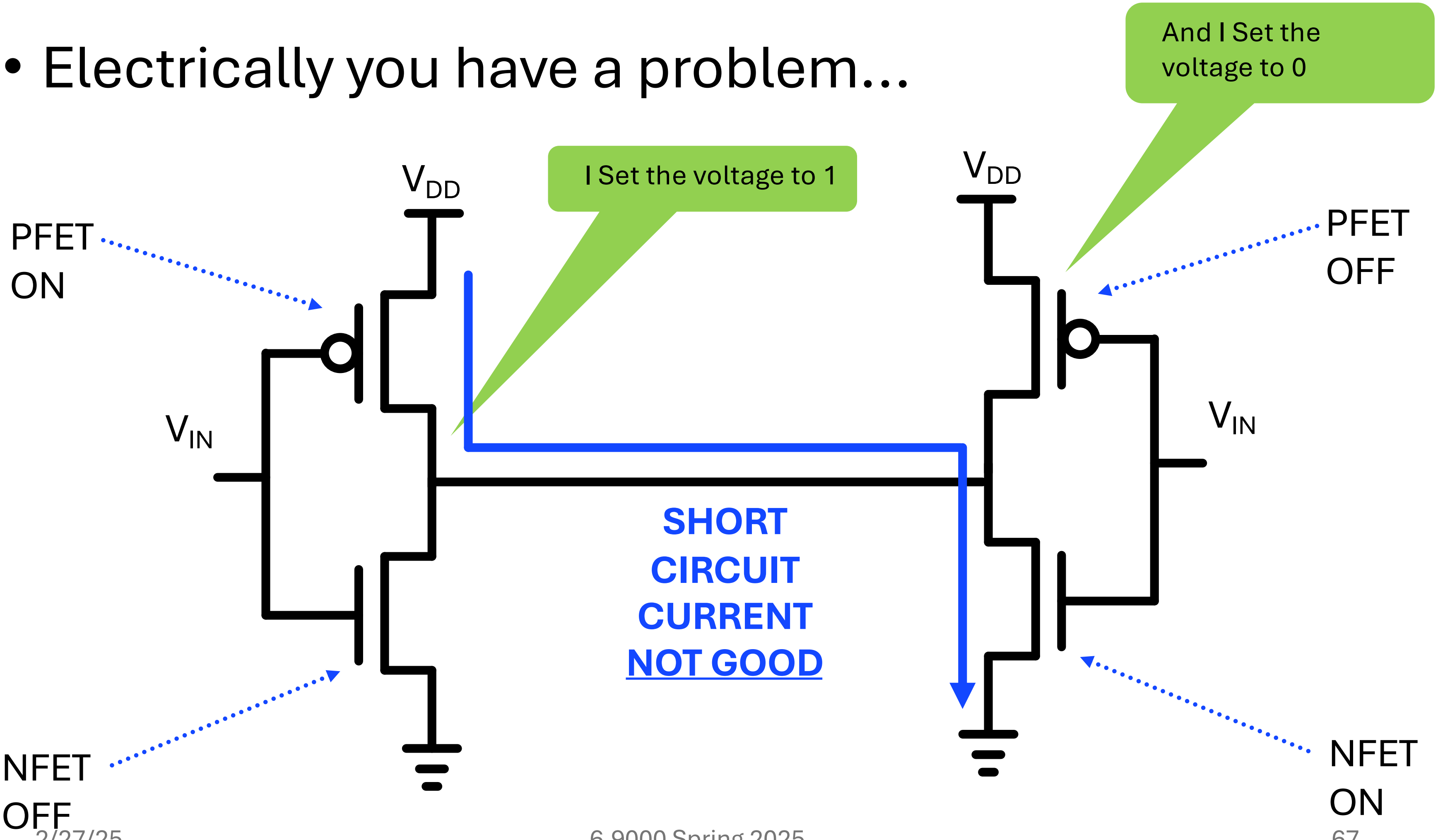
I read the voltage

What about if two or more devices want to use one common wire to communicate?

- Sounds like socialism…can't have that…it'll be a mess and none of the transistors will ever want to work…
- jkjk
- But seriously electrically you have a problem…

What about if two or more devices want to use one common wire to communicate?

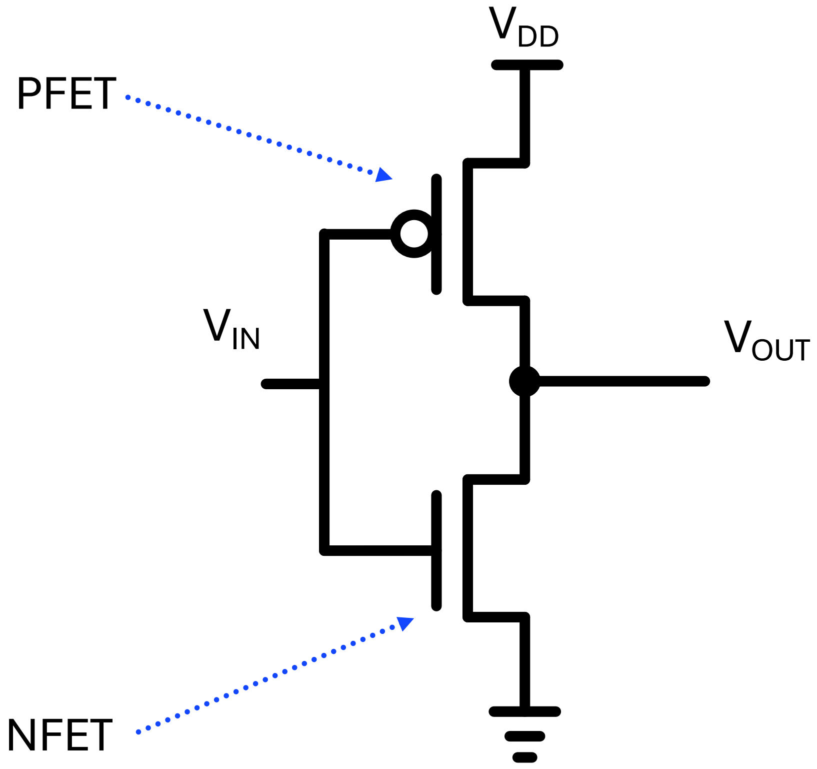
- Electrically you have a problem…



What Do You Do in Times of Conflict?

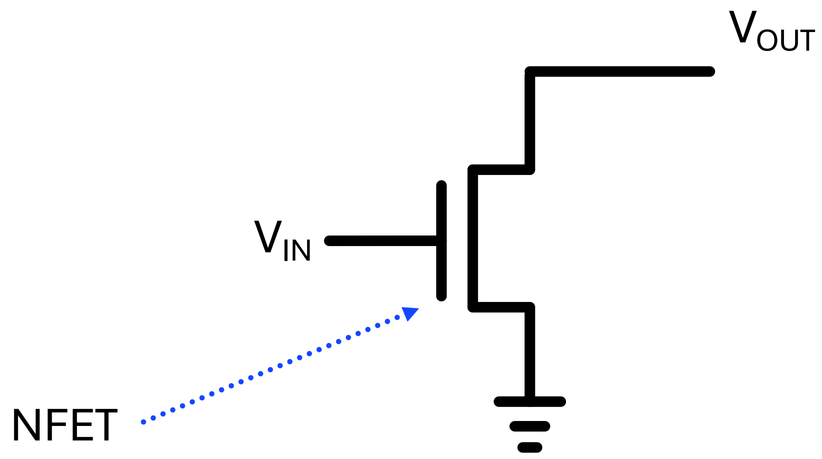
- You dig your heels in and waste tons of energy. Screw the other transistor. The correct answer is 1, not 0…everyone on r/1 agrees with me 1News Comment section backs me up.
- jkjk
- You come up with some compromises…everyone gives a little bit…everyone gets a little bit
- Each side gives up a transistor

Before We Had This…



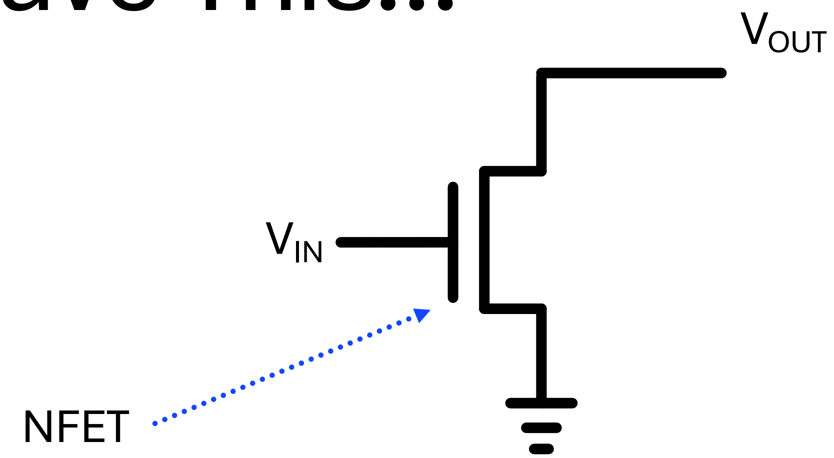
Now We Have This…

Call this "Open Drain" since the drain terminal of the transistor connects nowhere



ut is either Low

Now We Have This…



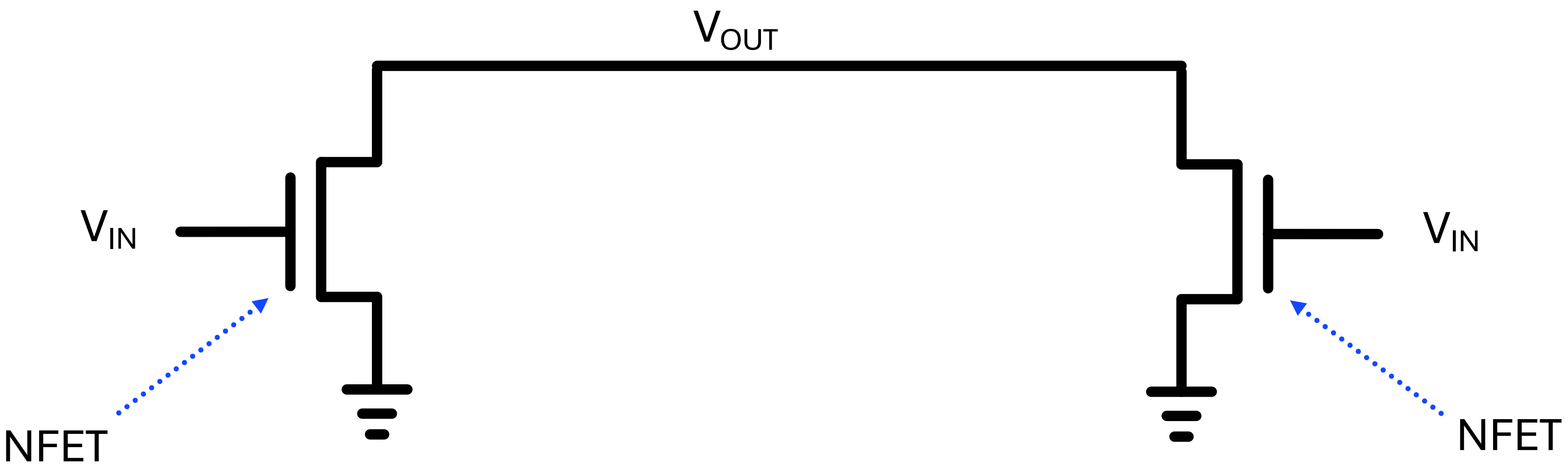
- Vout is either?????

- 0 (when transistor conducts)

- “HiZ”…basically electrically undefined (when transistor does not conduct)

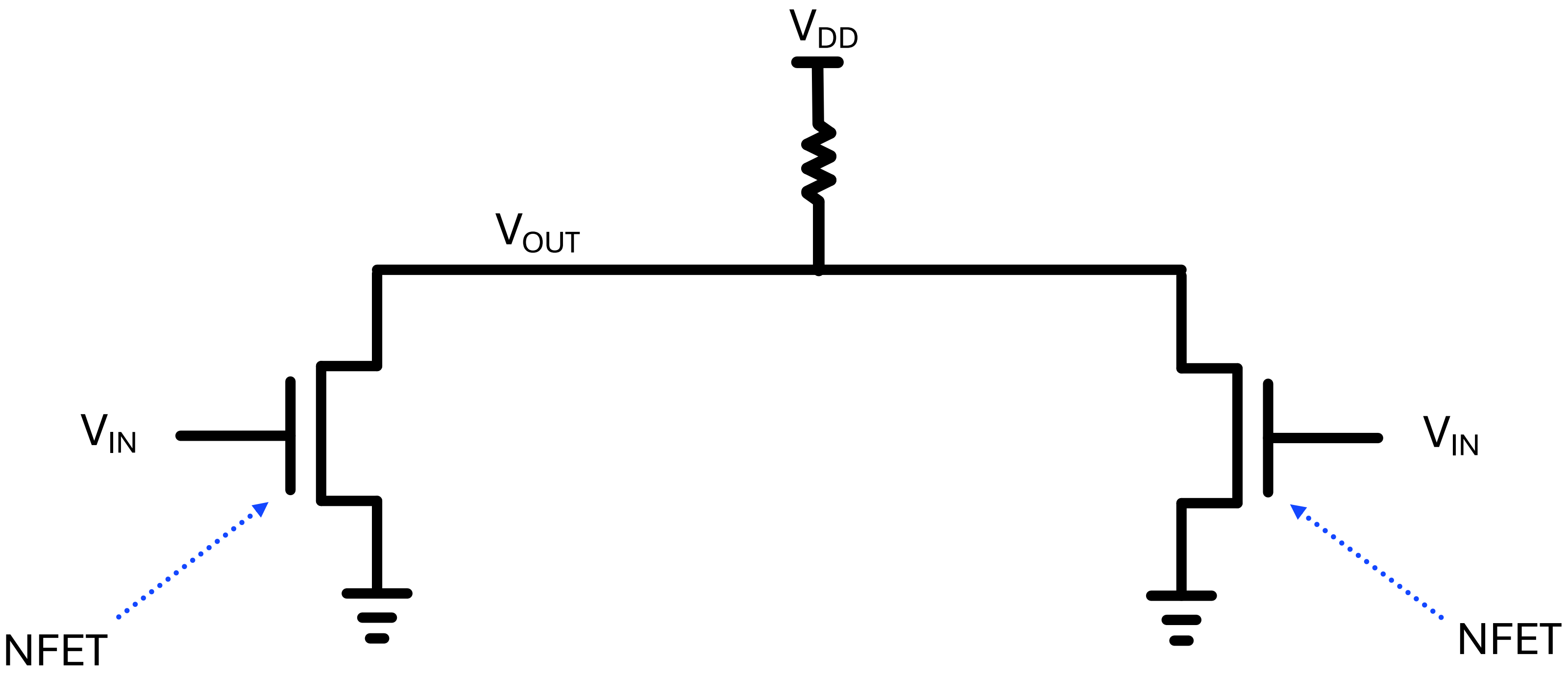
Now Connect up two of these circuits…

- Yeah…what does this give us?
- Each side can make a 0 by activating its transistor
- Can each side make a 1? **NO**



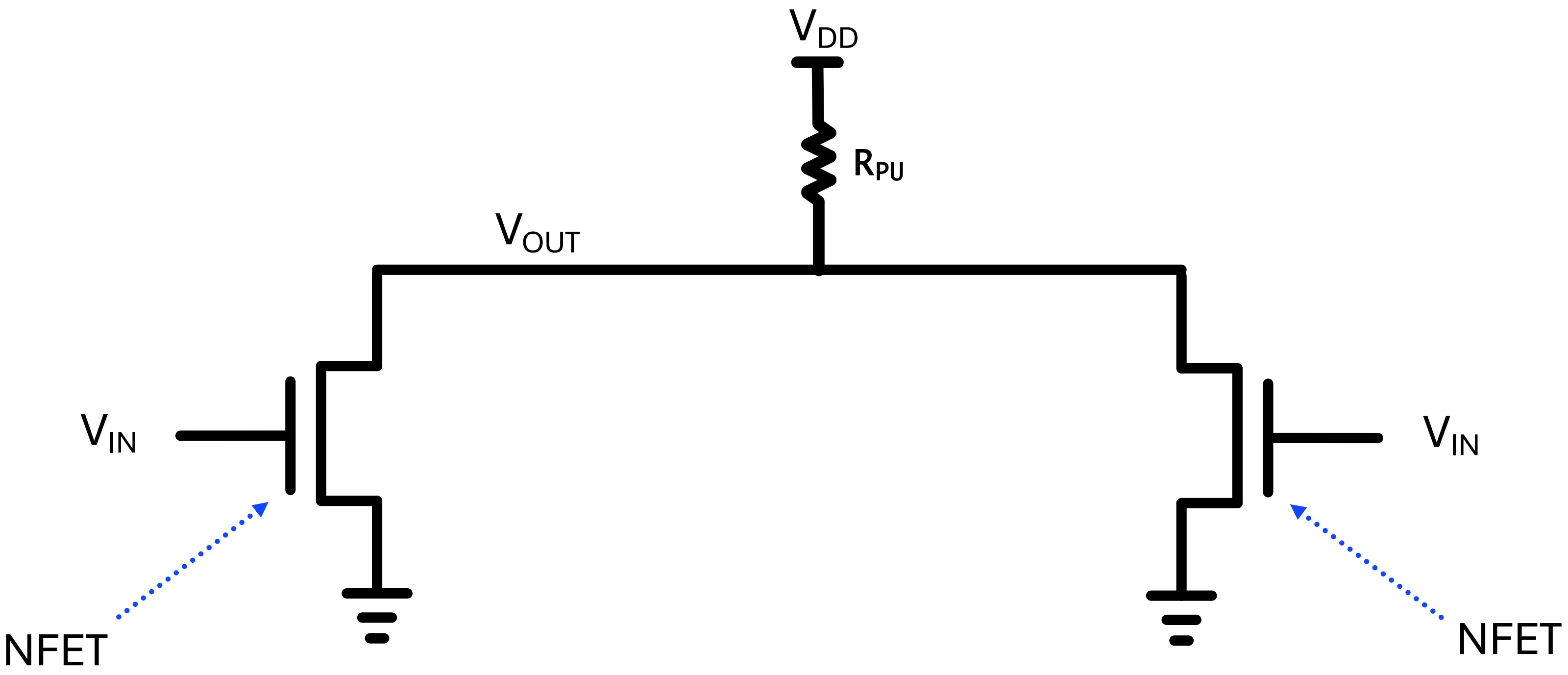
Bring in an Ombuds Component

- Each side can make a 0 by activating its transistor
- We use a neutral third-party component, trained in conflict mediation to give us 1.

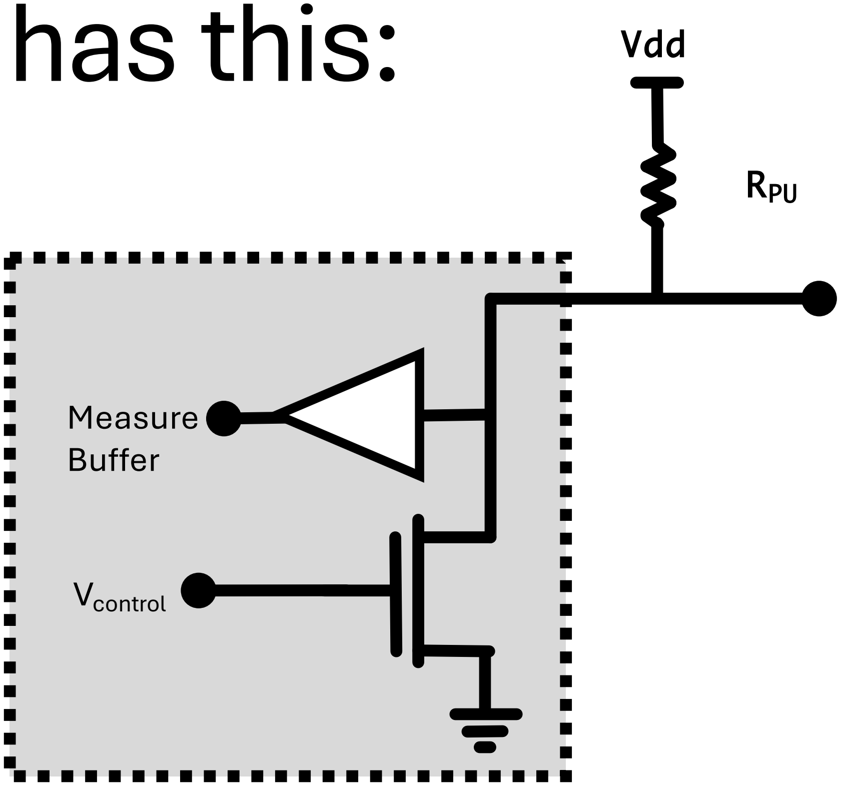


Common Pull-Up Resistor

- Prevents the Possibility of Short Circuits Always must go through this resistor (choose size to limit current)
- End up choosing several Kohms usually to keep current below 1mA

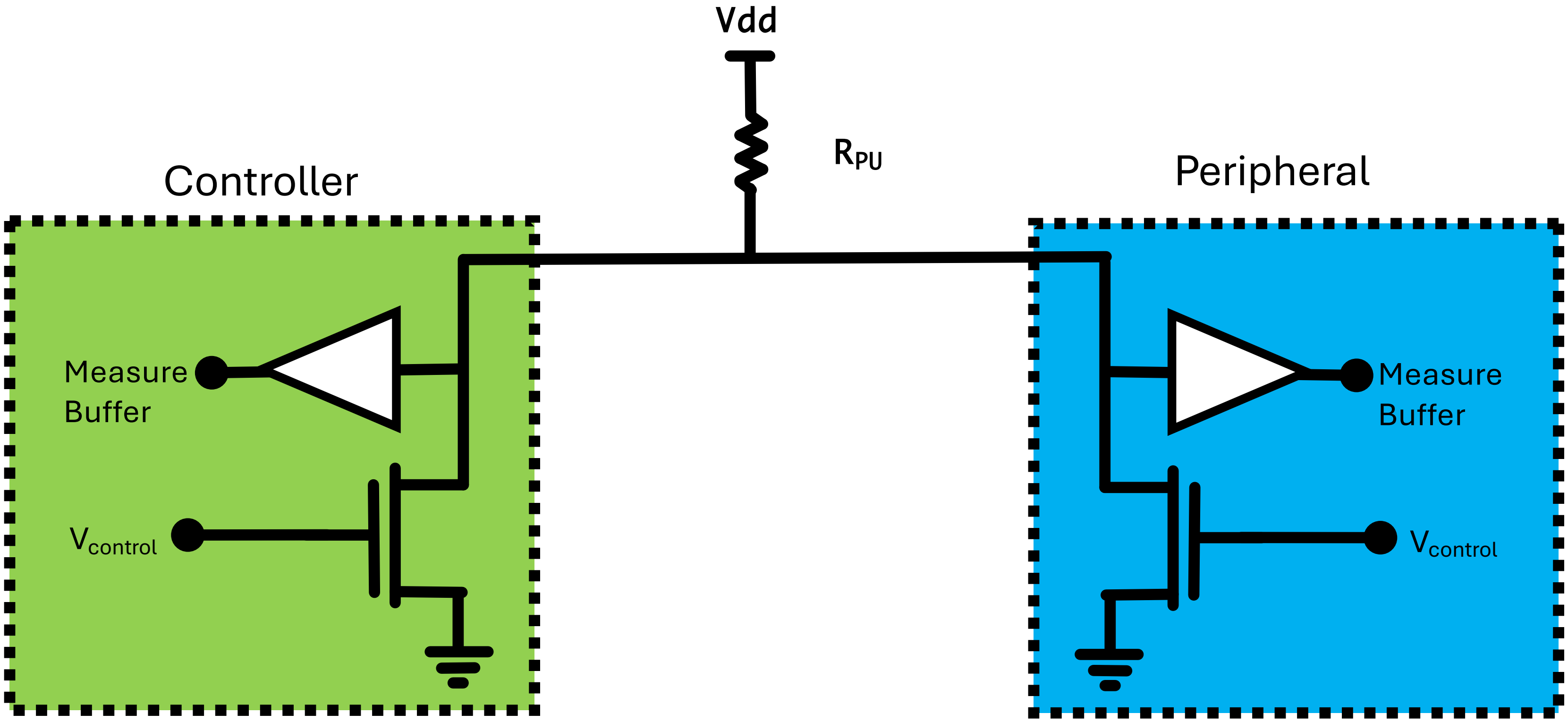


Result each side has this:



- If you want to say ”0”, you activate your transistor
- If you want to say ”1”, you inactivate your transistor and let resistor pull you up
- If you also want to listen you inactivate your transistor and monitor the line voltage

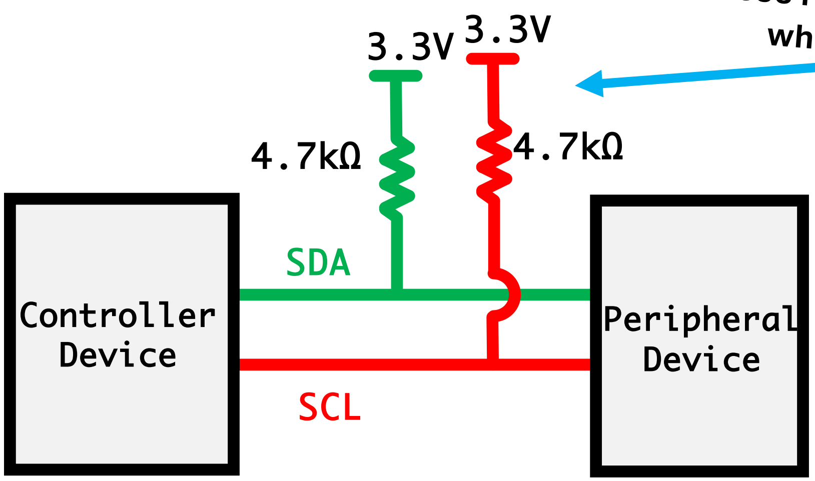
As a result:



Mode	Controller	Peripheral
Controller Transmit	HiZ (HI) or LOW	HiZ (listening)
Peripheral Transmit	HiZ (listening)	HiZ (HI) or LOW

So in Deployment…

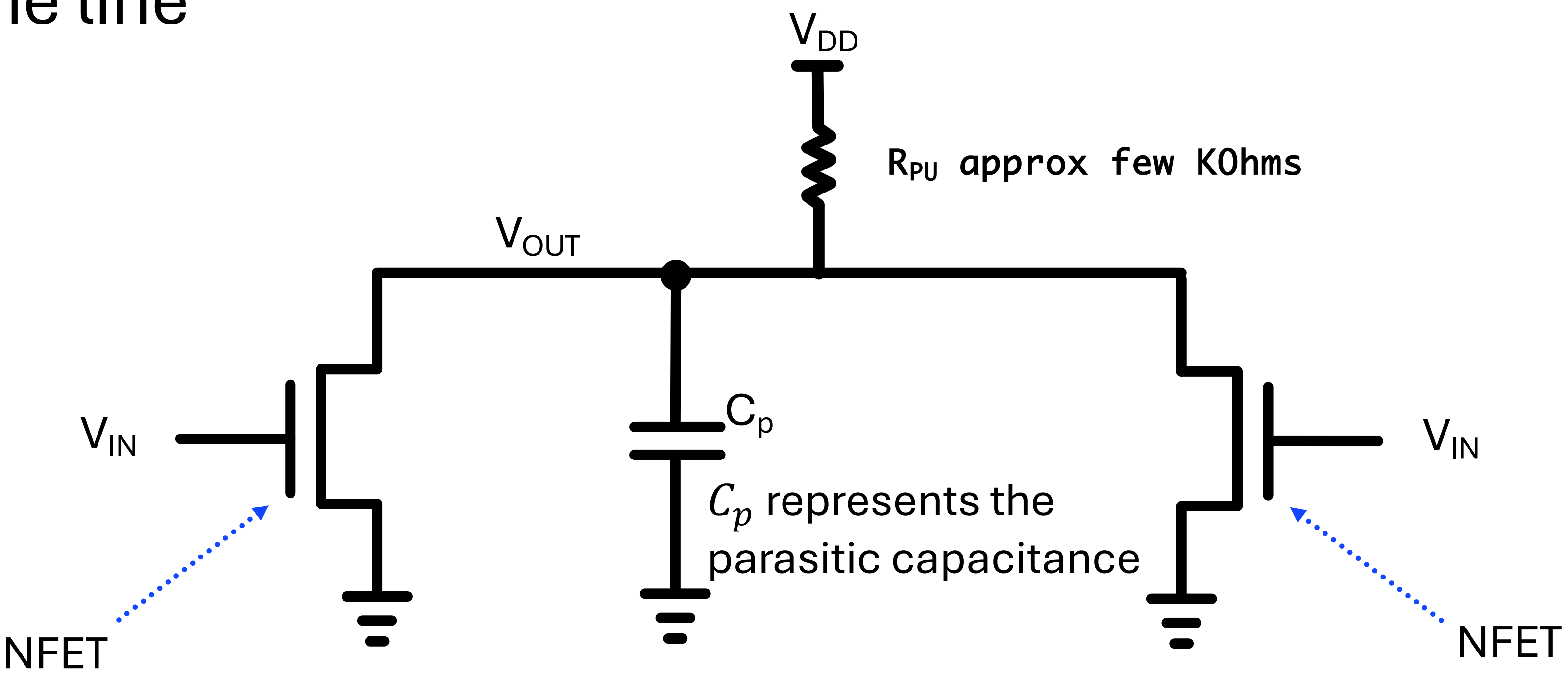
- i2C uses an open drain
- Meaning both Controller and Peripheral Device are either:
  - LOW
  - “High-Impedance”
- Need external pull-up resistors on both parts of I2C to make it work



These resistors are large reason why data rate is so low!

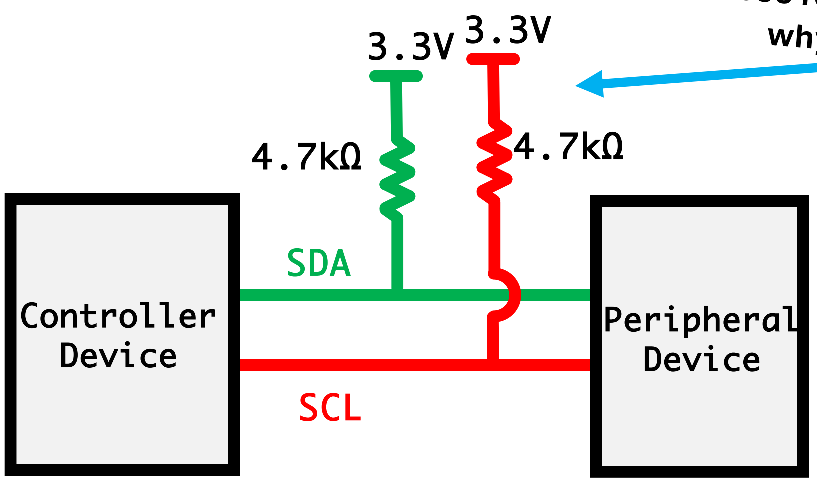
Common Pull-Up Resistor

- We choose the pull up resistors to be in the K range usually to keep current/power down.
- This has the downside that parasitic capacitances lead to relatively large time constants in charging/discharging the line


V_{DD}
R_{PU} approx few KOhms
V_{OUT}
V_{IN}
C_p
V_{IN}
C_p represents the parasitic capacitance
NFET
NFET

So in Deployment…

- So with all this together…we can see that there needs to be a lot more order in how to use the I2C wires…things pull double-duty depending on context.



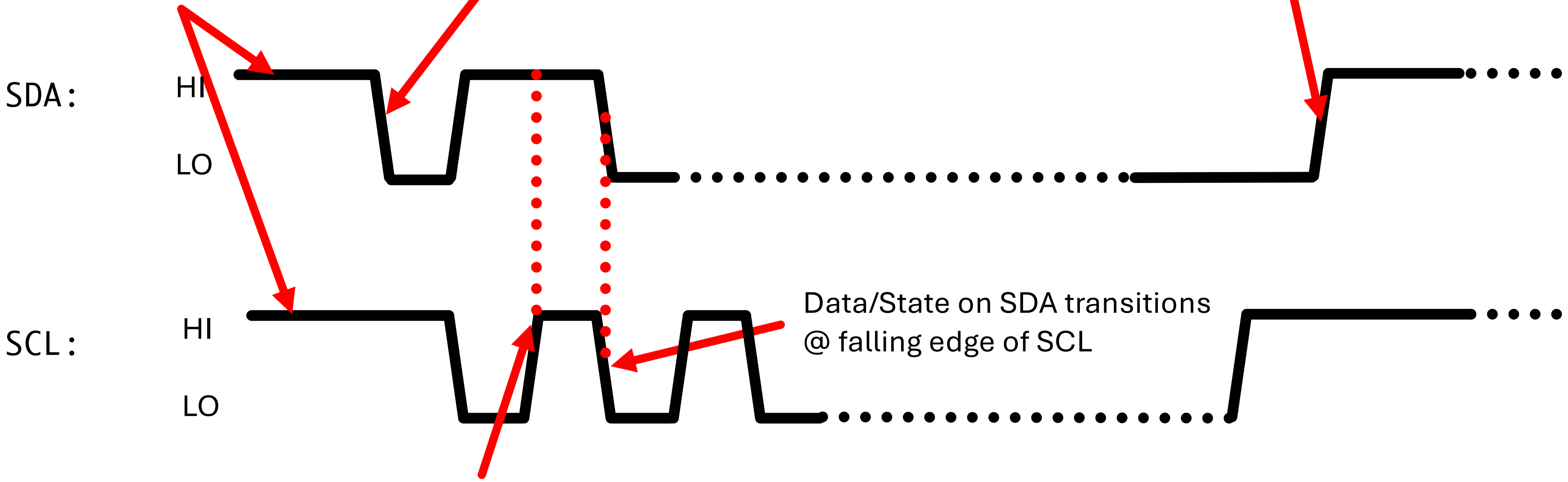
These resistors are large reason why data rate is so low!

i2C Operation

- Data is conveyed on SDA (Either from Main or Secondary depending on point during communication)
- SCL is a 50% duty cycle clock
- SDA generally changes on falling edge of SCL (isn’t required, but is a convenient marker for targeting transitions)
- SDA sampled at rising edge of SCL
- Main is in charge of setting SCL frequency and driving it
- Data is sent **msb** first

Notice how much more rigid this is compared to SPI

Meanings I: (Start, Stop, Sampling)



Idle State
SDA and SCL sit HI

Controller Claims Bus (START)
By pulling SDA LOW while SCL is HI

Controller Releases Bus (STOP)
By pulling SDA HI while SCL is HI

Data/State on SDA transitions
@ falling edge of SCL

Data from SDA sampled @ posedge of SCL

Meanings II Address

- First thing sent by Controller is 7 bit address (10 bit in more modern i2C…don’t worry about that)

- If a device on the bus possesses that address, it acknowledges (ACK=0/NACK=1) and it becomes the secondary for the time being.

- All other devices (other than Controller/Peripheral Devices) will ignore until STOP signal appears later on.

Meanings III (Read/Write Bit)

- After sending address, a Read/Write Bit is specified by Controller on SDA:
  - If Write (0) is specified, the next byte will be a register to write to, and following bytes will be information to write into that register
  - If Read (1) is specified, the Peripheral Device will start sending data out, with the Controller Device acknowledging after every byte (until it wants data to not be sent anymore)

Meanings IV (ACK/NACK)

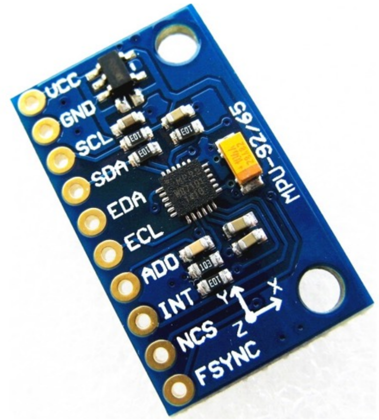
- After every 8 bits, it is the *listener's* job to acknowledge or not acknowledge the data just sent (called an ACK/NACK)
- Transmitter pulls SDA HI and listens for next reading the next time SCL transitions high:
  - If LOW, then receiver acknowledges data
  - If remains HI, no acknowledgement
- Transmitter/Receiver act accordingly

Meanings V

- For Controller Device to **write** to Peripheral Device:
    - START
    - Send Device Address (with Write bit)
    - Send register you want to write to
    - Send data…until you're satisfied, doing ACK/NACKs along the way
    - STOP
- For Controller Device to **read** from Peripheral Device a common (though not universal procedure) is:
    - START
    - Send Device Address (with Write bit)
    - Send register you want to read from (*think of this like setting a cursor in the register map)*
    - ***ReSTART*** communication
    - Send Device Address (With Read bit)
    - Read the bits (*it'll start from where the cursor was left pointing at)*
    - After every 8 bits, it is Controller's job to ACK/NACK Peripheral…continued acknowledgement leads to continued data out by Peripheral.
    - Not-Acknowledge says "no more data from Peripheral"
    - STOP leads to Controller ceasing all communication

MPU-9250

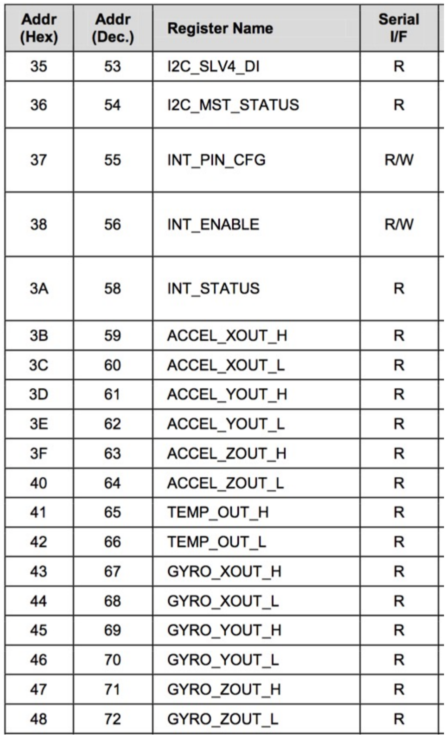
Board: $5.00 from Ebay
Chip: $1.00 in bulk



- **3-axis Accelerometer (16-bit readings)**
- **3-axis Gyroscope (16-bit readings)**
- **3-axis Magnetic Hall Effect Sensor (Compass) (16 bit readings)**
- SPI or I2C communication (!)…no analog out
- On-chip Filters (programmable)
- On-chip programmable offsets
- On-chip programmable scale!
- On-chip sensor fusion possible (with quaternion output)!
- Interrupt-out (for low-power applications!)
- On-chip sensor fusion and other calculations (can do orientation math on-chip or pedometry even)
- So cheap they usually aren’t even counterfeited! ☺
- **Communicates using either I2C or SPI**

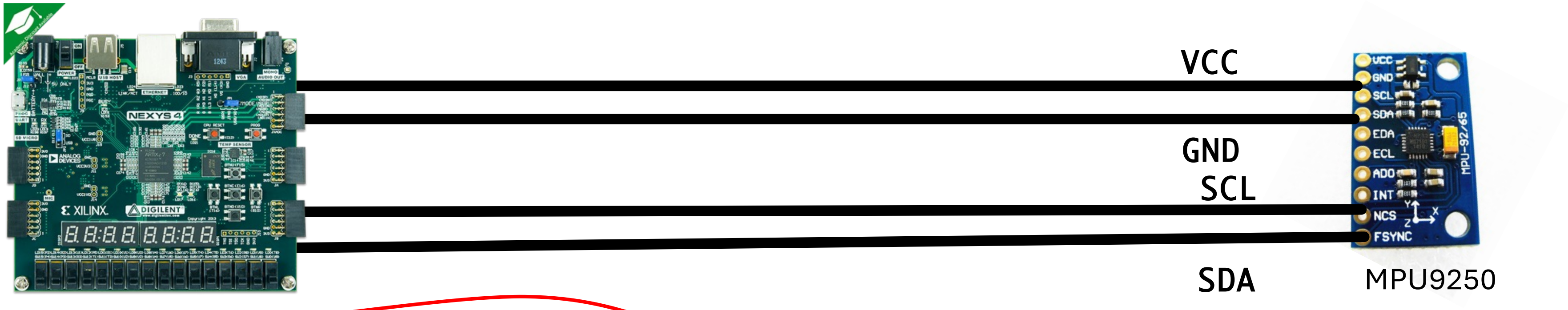
Implementing i2C on FPGA with MPU9250:

- Made Controller i2C module in Verilog
- Used MPU9250 Data sheet: 42 pages (basic functionality, timing requirements, etc…)
- MPU9250 Register Map: 55 pages



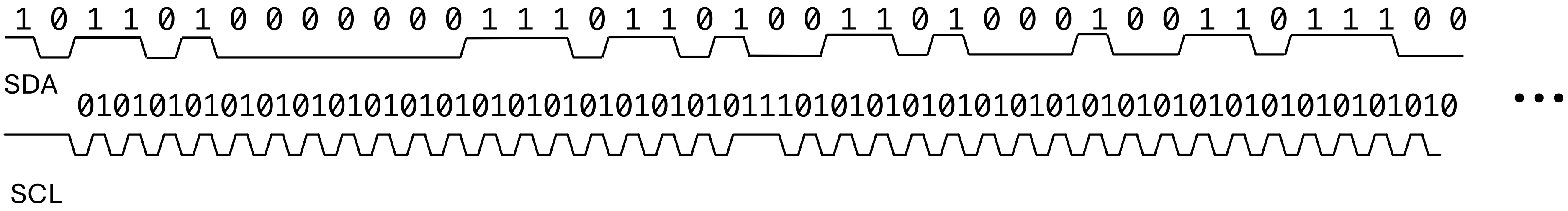
Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F
35	53	I2C_SLV4_DI	R
36	54	I2C_MST_STATUS	R
37	55	INT_PIN_CFG	R/W
38	56	INT_ENABLE	R/W
3A	58	INT_STATUS	R
3B	59	ACCEL_XOUT_H	R
3C	60	ACCEL_XOUT_L	R
3D	61	ACCEL_YOUT_H	R
3E	62	ACCEL_YOUT_L	R
3F	63	ACCEL_ZOUT_H	R
40	64	ACCEL_ZOUT_L	R
41	65	TEMP_OUT_H	R
42	66	TEMP_OUT_L	R
43	67	GYRO_XOUT_H	R
44	68	GYRO_XOUT_L	R
45	69	GYRO_YOUT_H	R
46	70	GYRO_YOUT_L	R
47	71	GYRO_ZOUT_H	R
48	72	GYRO_ZOUT_L	R

Communication Part

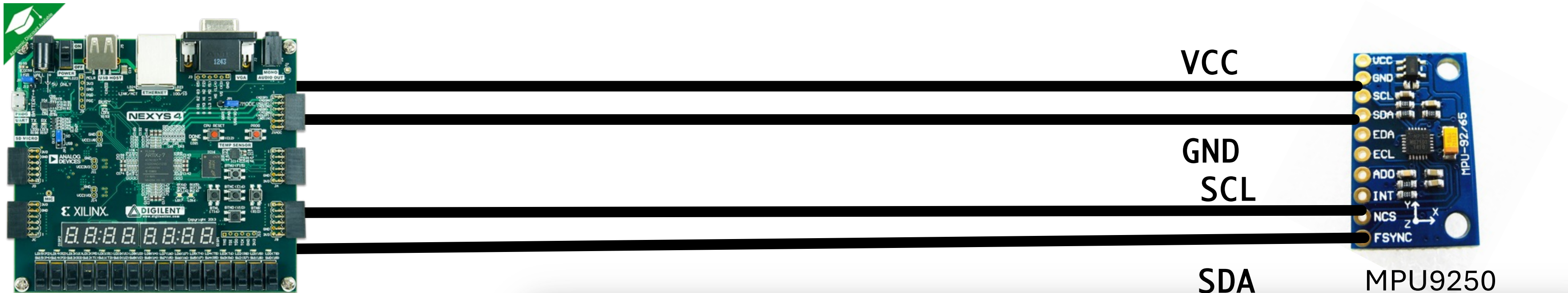


Nexys4

Can also do this on our current board



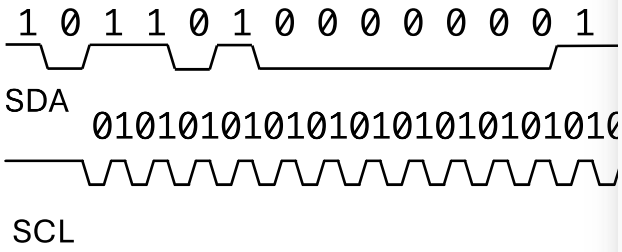
Communication Part

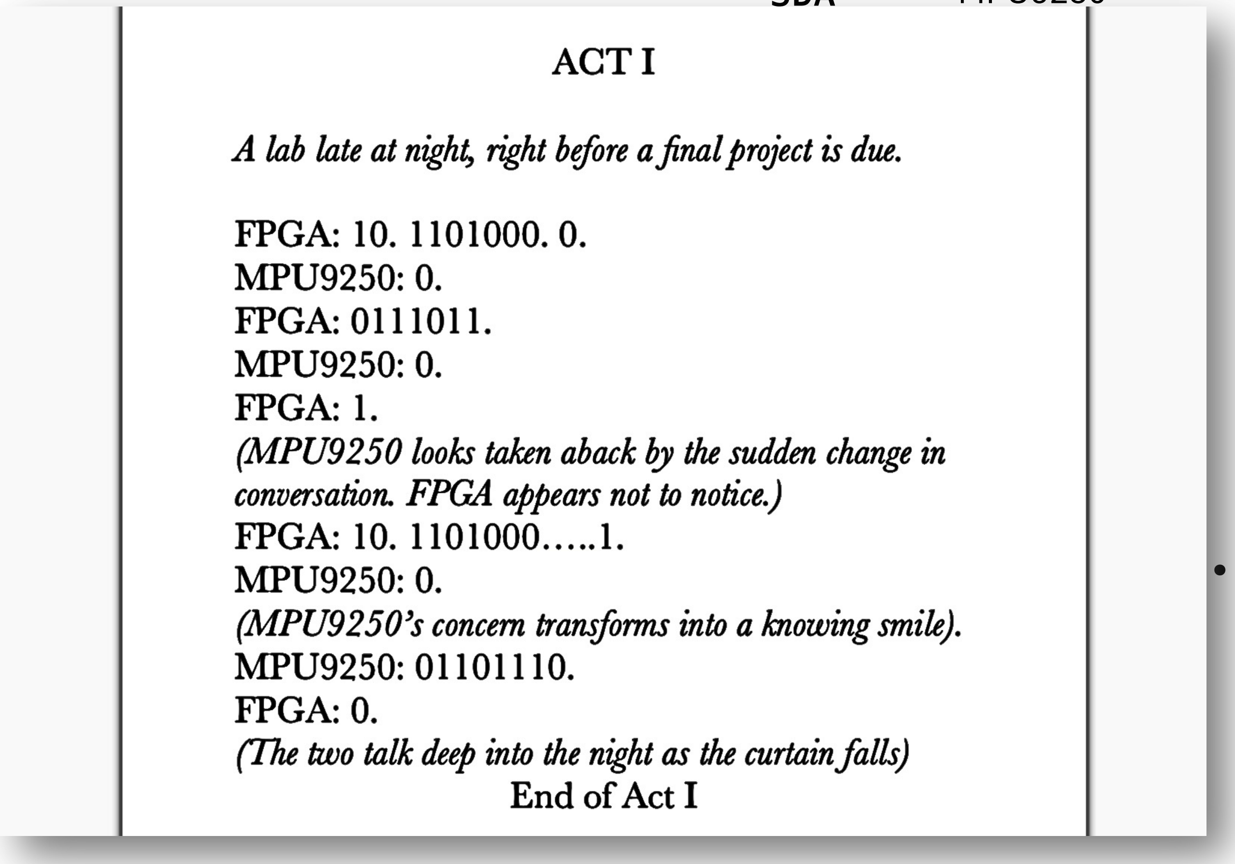


Nexys4

Can also do th

Needs a dialogue





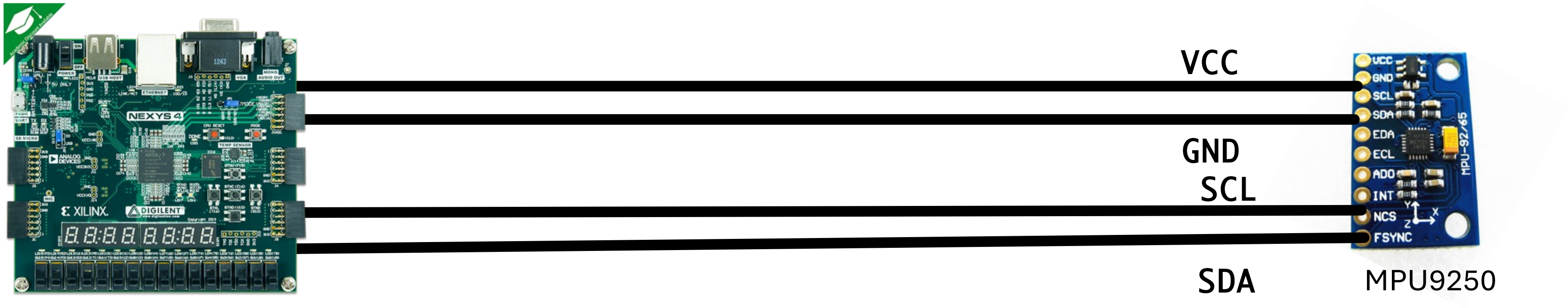
ACT I

A lab late at night, right before a final project is due.

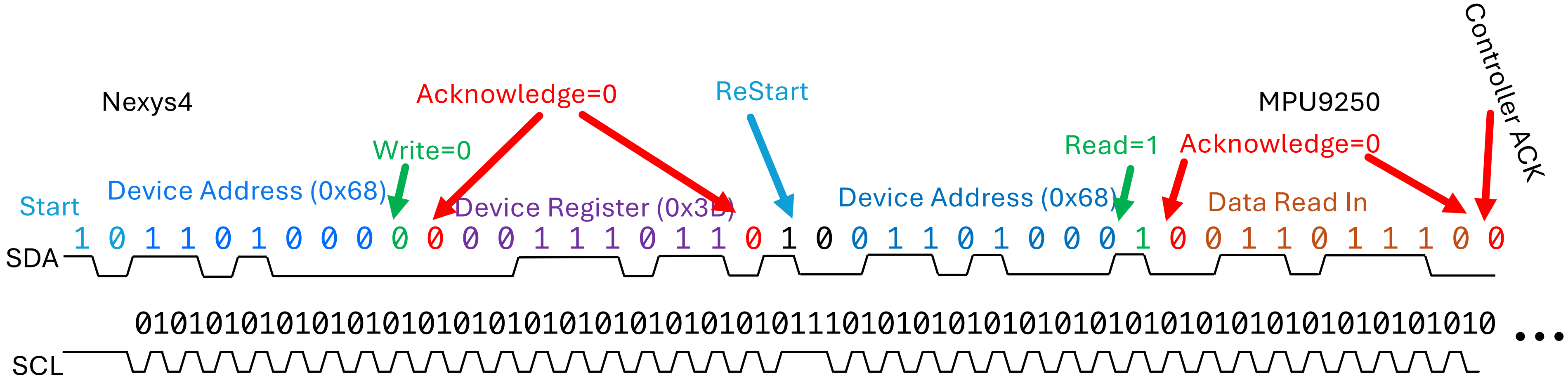
FPGA: 10. 1101000. 0.
MPU9250: 0.
FPGA: 0111011.
MPU9250: 0.
FPGA: 1.
(MPU9250 looks taken aback by the sudden change in conversation. FPGA appears not to notice.)
FPGA: 10. 1101000…..1.
MPU9250: 0.
(MPU9250's concern transforms into a knowing smile).
MPU9250: 01101110.
FPGA: 0.
(The two talk deep into the night as the curtain falls)

End of Act I

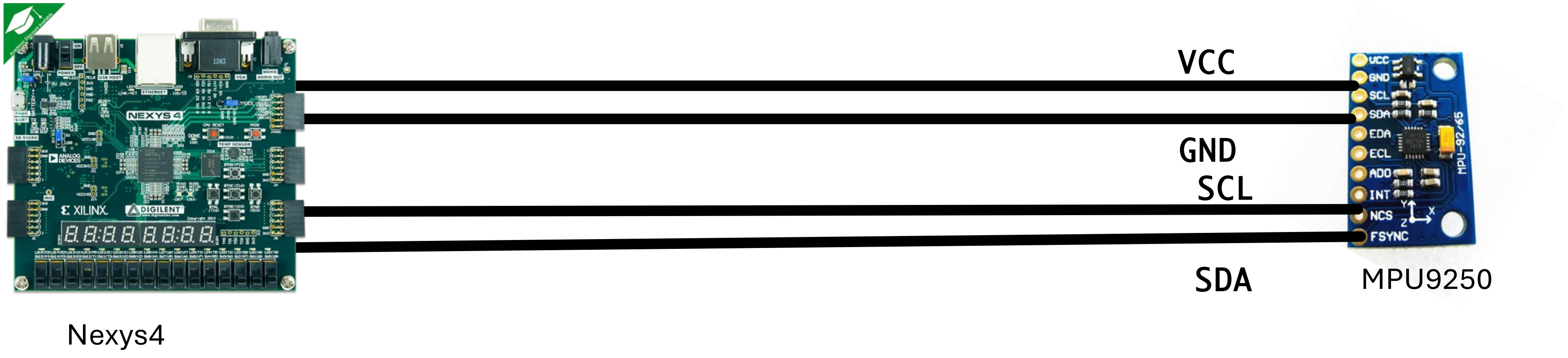
Communication Part

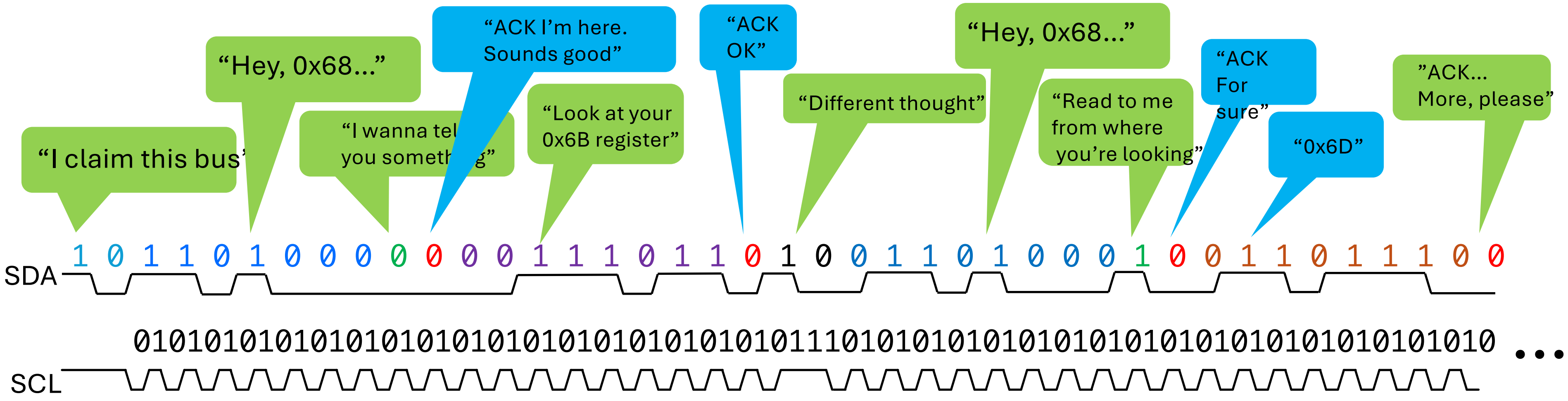


Nexys4



Communication Part





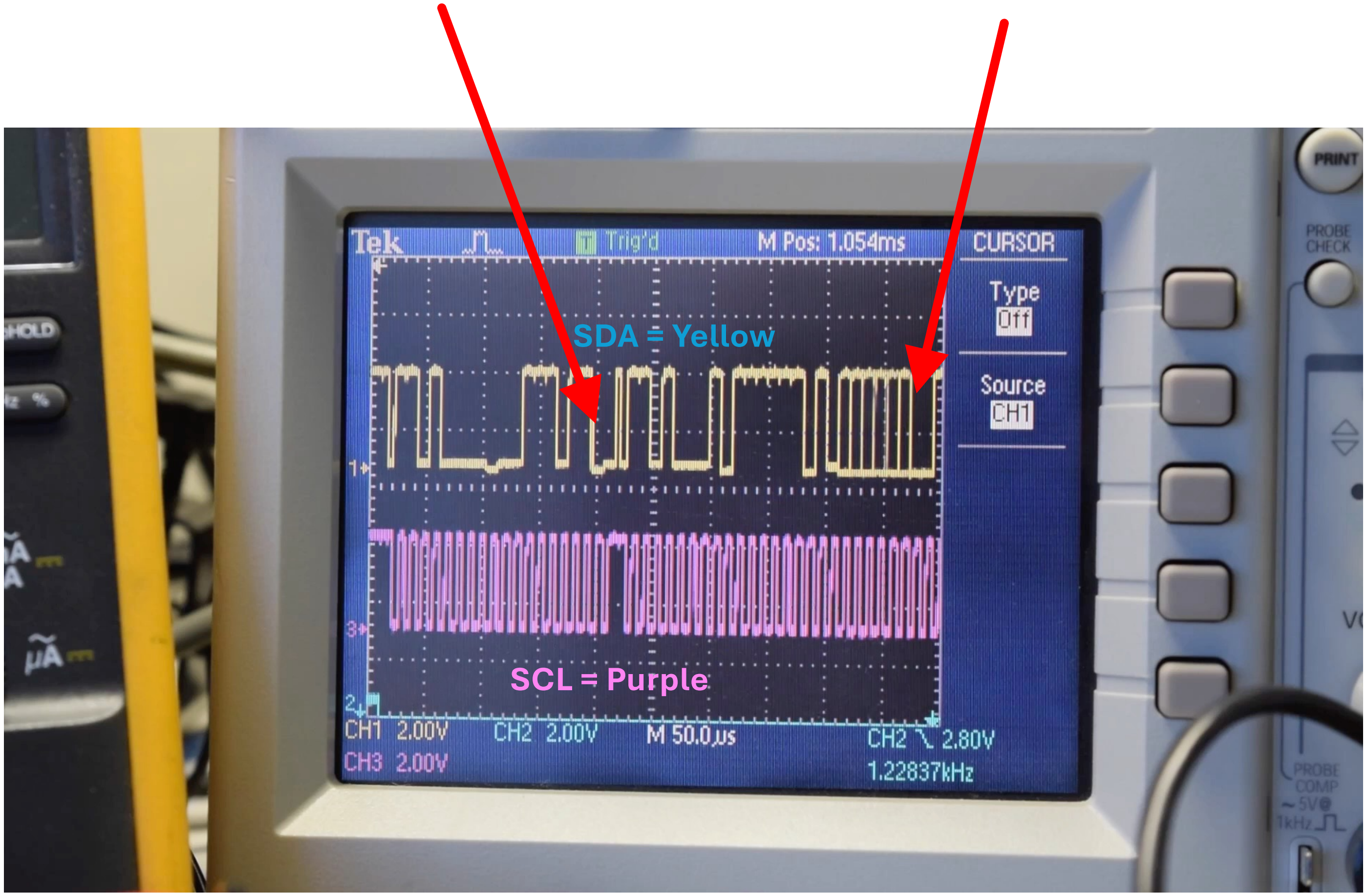
🟩 Nexys4 (Controller Device) Dialog

🟦 MPU9250 (Peripheral Device) Dialog

Communication in Real-Life:

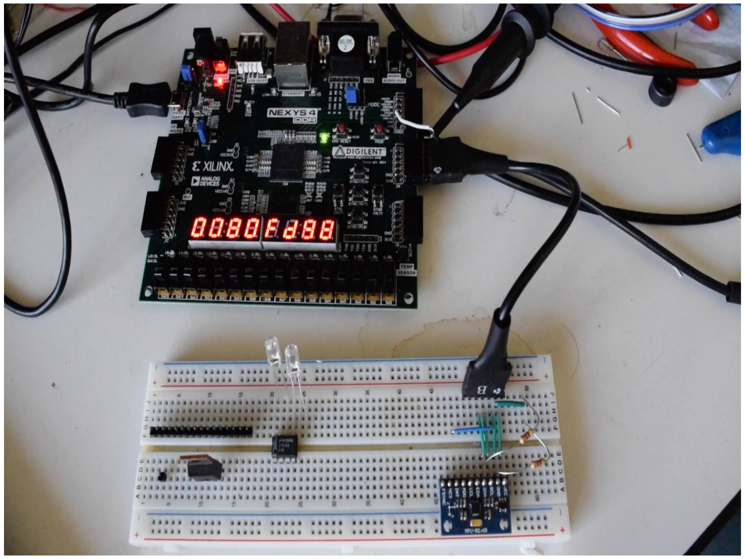
Data being sent **to** MPU9250

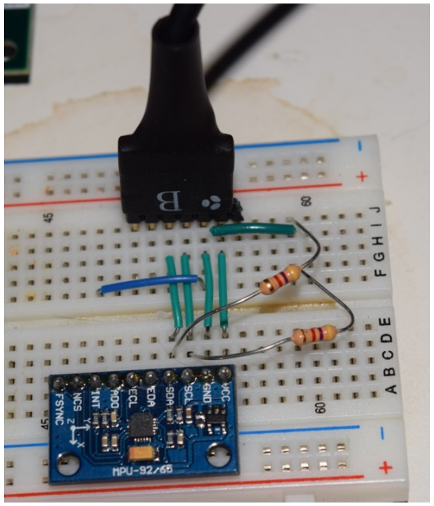
Data being sent **from** MPU9250



Triggered on leaving IDLE state

Running and reading X acceleration:





HOOKUP

Horizontal:

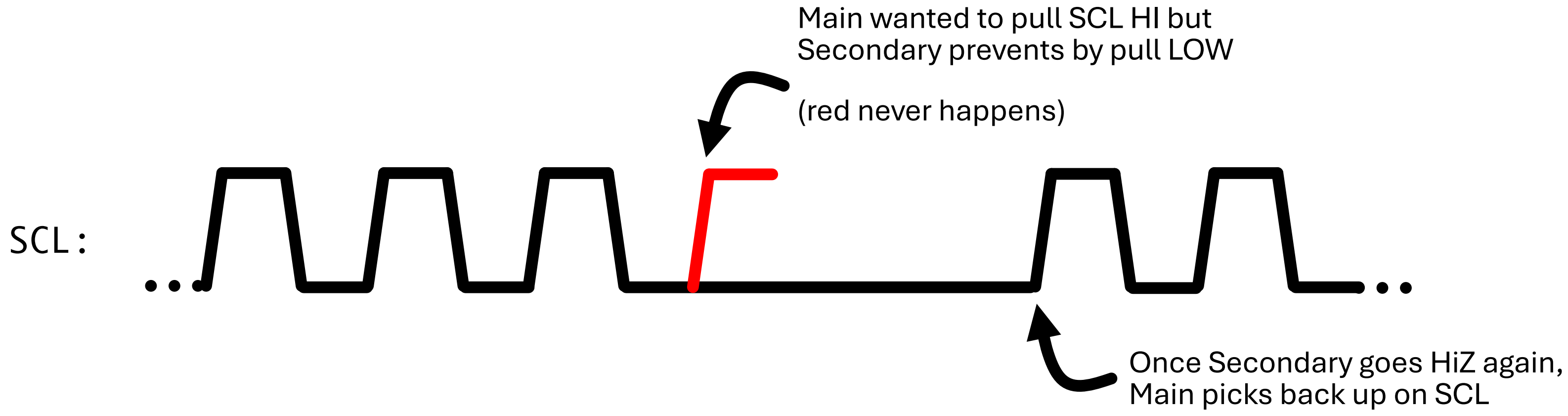
16’hFD88 = 16’b1111_1101_1000_1000 (2’s complement)
Flip bits to get magnitude: 16’b0000_0010_0111_0111
=-315
Full-scale (default +/- 2g)
-315/(2**15)*2g = -0.02g ☺ makes sense

Vertical:

16’h4088 = 16’b0100_0000_1000_1000 (2’s complement)
Leave bits to get magnitude: 16’b0100_0000_1000_1000
=+16520
Full-scale (default +/- 2g)
-16520/(2**15)*2 = +1.01g ☺ makes sense!

Clock-Stretching (Cool part of i2C!!!)

- Normally Controller drives SCL, but since Controller drives SCL high by going hiZ, it leaves the option open for Peripheral to step in and prevent SCL from going high by pulling SCL LOW



Main wanted to pull SCL HI but Secondary prevents by pull LOW

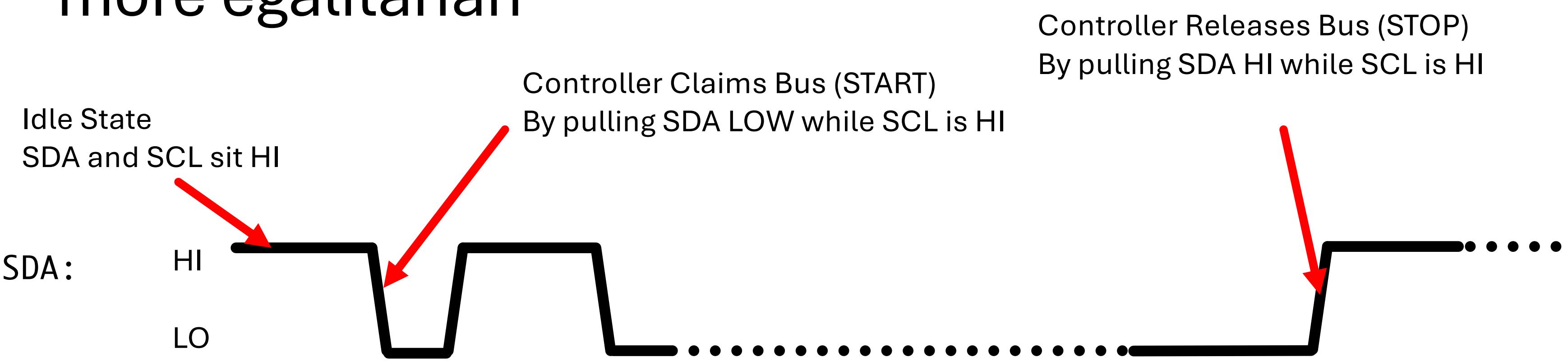
(red never happens)

Once Secondary goes HiZ again, Main picks back up on SCL

- Allows Peripheral a way to buy time/slow down things (if it requires multiple clock cycles to process incoming data and/or generate output)

I2C Can Also Be a “Multi-Controller” Bus

- In SPI, there is a pre-determined device in charge of the system. I2C is potentially much more egalitarian



Idle State
SDA and SCL sit HI

Controller Claims Bus (START)
By pulling SDA LOW while SCL is HI

Controller Releases Bus (STOP)
By pulling SDA HI while SCL is HI

- Devices can be design to yield based on who claims a bus first…but you have to be careful…what if two devices claim a bus at the same time…potential problems? Can get bus contention so need to be careful

Compare and Contrast?

- Generally the fewer the wires the more rigid the protocol
- SPI can be very flexible and high speed (have only 10 bits to send? No problem…send 10!…can't do that do that with i2C…need to zero-pad up to the next full byte (16 bits)
- In terms of implementation, generally with communication protocols, the more wires, the easier the protocol/less overhead

Which to Choose?

- SPI is generally easier and more flexible to implement, but only certain devices use it since it takes up a lot of pins (and pins are expensive/limited)
- ”Slow” and “Fast” data rates are relative too…i2C is not as much of a compromise now as it was fifteen years ago, particularly with high-speed i2C (or even now that 400 kHz rates are common)
- Remember, these are all meant for chip-to-chip communications!
- Check out the example i2C code from this lecture for the IMU…see if you can add clock-stretching! (not required)