ES 4: I/O peripherals and protocols

Steven Bell

21 November 2023

Tufts UNIVERSITY

Logistics

- Ethics assignment due on Gradescope (tomorrow at 11:59pm)
- Lab 7 needs to be submitted on Gradescope (no report, just code)

Objectives

- Explain what "memory-mapped I/O" is, and why it's popular
- List the different signals needed for SPI, I2C, and UART, and explain what they do.

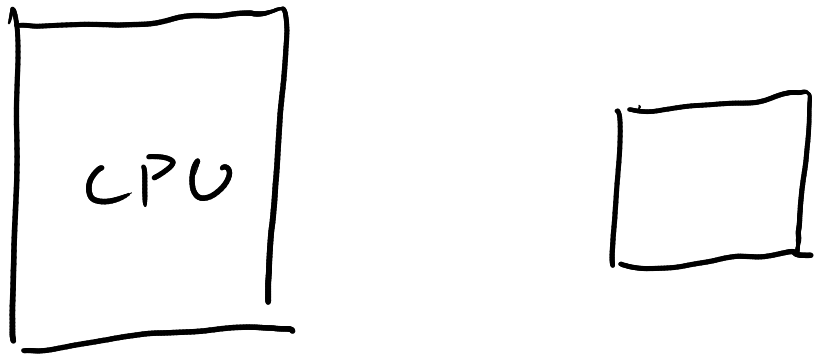
From the reading check

What's the point??

Will we use [I2C, SPI, UART] for our project?

Problem

You have a processor, and you want to connect a widget to it. How?



Problem

You have a processor, and you want to connect a widget to it. How?

Add special instructions to the processor

Problem

You have a processor, and you want to connect a widget to it. How?

- Add special instructions to the processor
- Make the widget act like memory

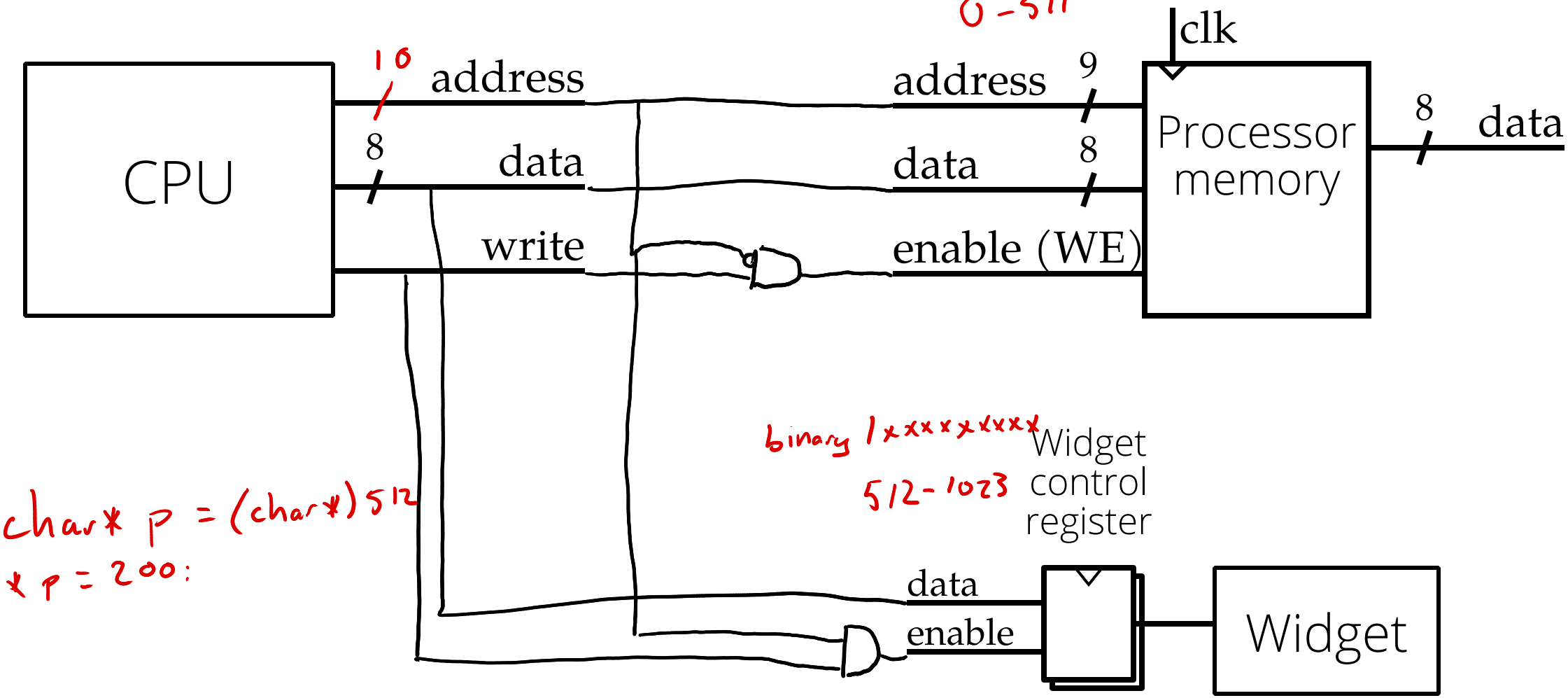
Memory-mapped I/O

(write-only for the moment)

$2^9 = 512$

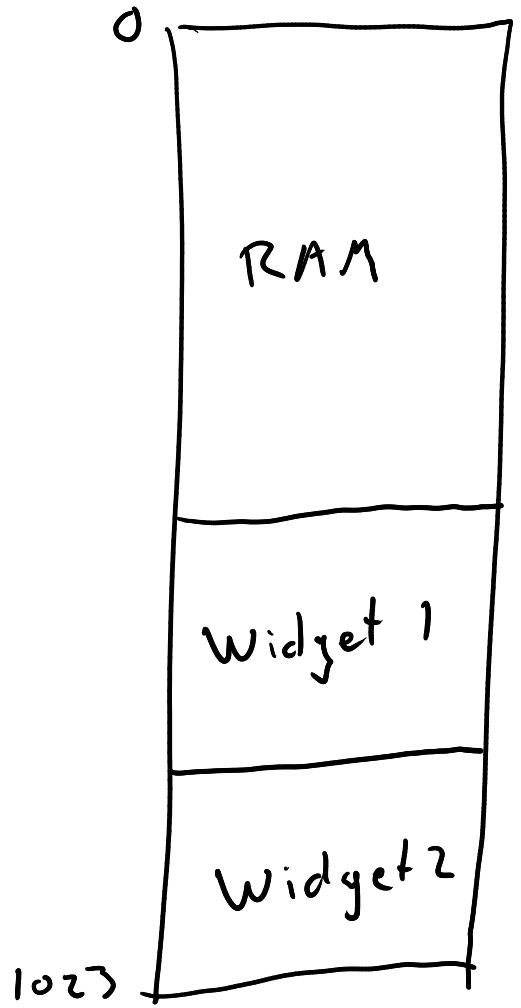
binary 0xxxx xxxxx

0-511

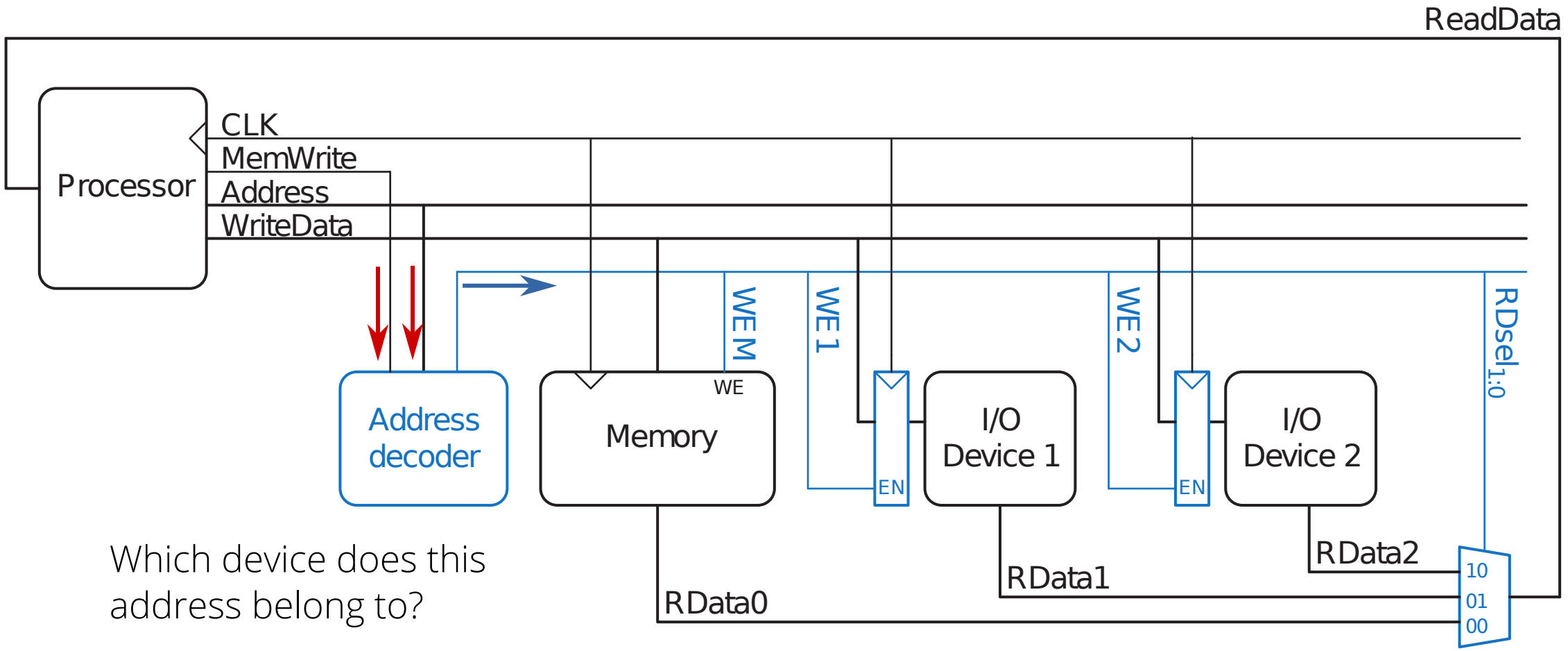


char* p = (char*)512

*p = 200;

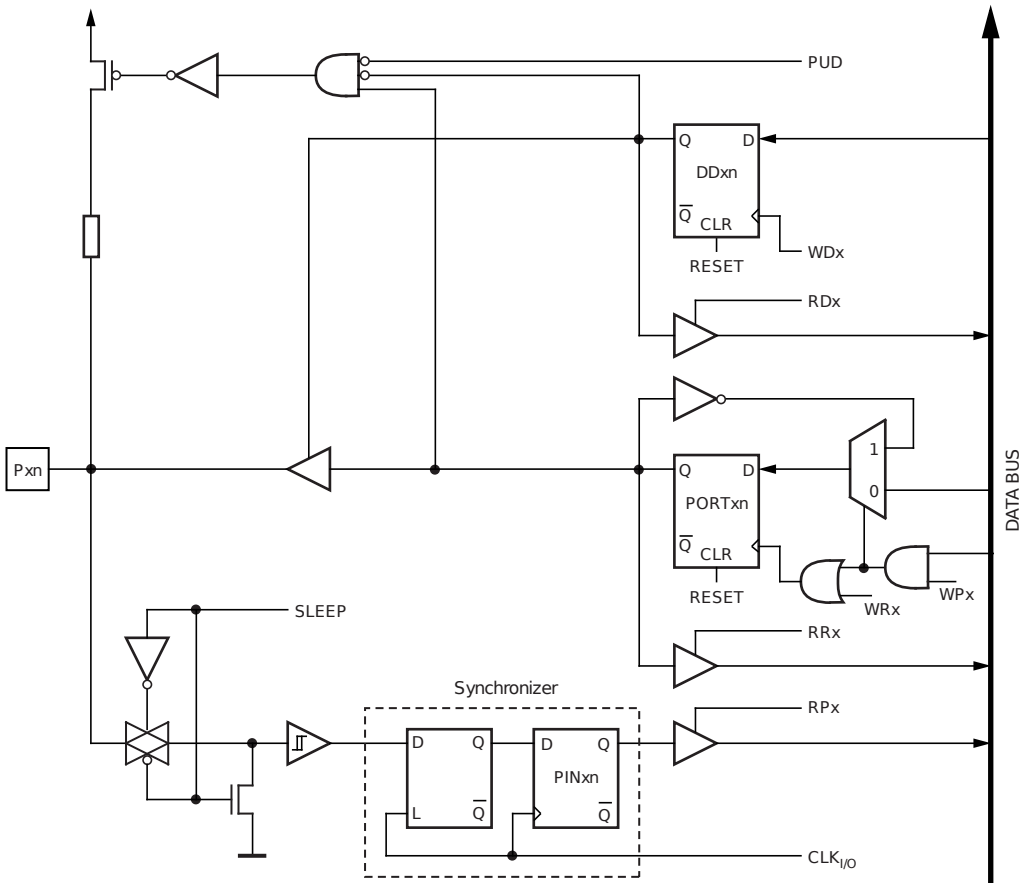

0
RAM
Widget 1
Widget 2
1023

Memory-mapped I/O



Which device does this address belong to?

Anatomy of an Arduino pin



PUD:	PULLUP DISABLE	WDx:	WRITE DDRx
SLEEP:	SLEEP CONTROL	RDx:	READ DDRx
$CLK_{I/O}$:	I/O CLOCK	WRx:	WRITE PORTx
		RRx:	READ PORTx REGISTER
		RPx:	READ PORTx PIN
		WPx:	WRITE PINx REGISTER

Memory-mapped I/O on Arduino

13.4.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS	BODSE	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See Section 13.2.1 “Configuring the Pin” on page 59for more details about this feature.

13.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	**PORTB7**	**PORTB6**	**PORTB5**	**PORTB4**	**PORTB3**	**PORTB2**	**PORTB1**	**PORTB0**	**PORTB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	**DDB7**	**DDB6**	**DDB5**	**DDB4**	**DDB3**	**DDB2**	**DDB1**	**DDB0**	**DDRB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

How could we use fewer wires?

Address/data is great for on-chip communication,
but what if I want to connect a [...] to it?

- Wires are bulky and cost money
- PCB traces must be routed
- I/O pins are limited (UPduino has ~38?)
- High-speed signals must have matching wire lengths

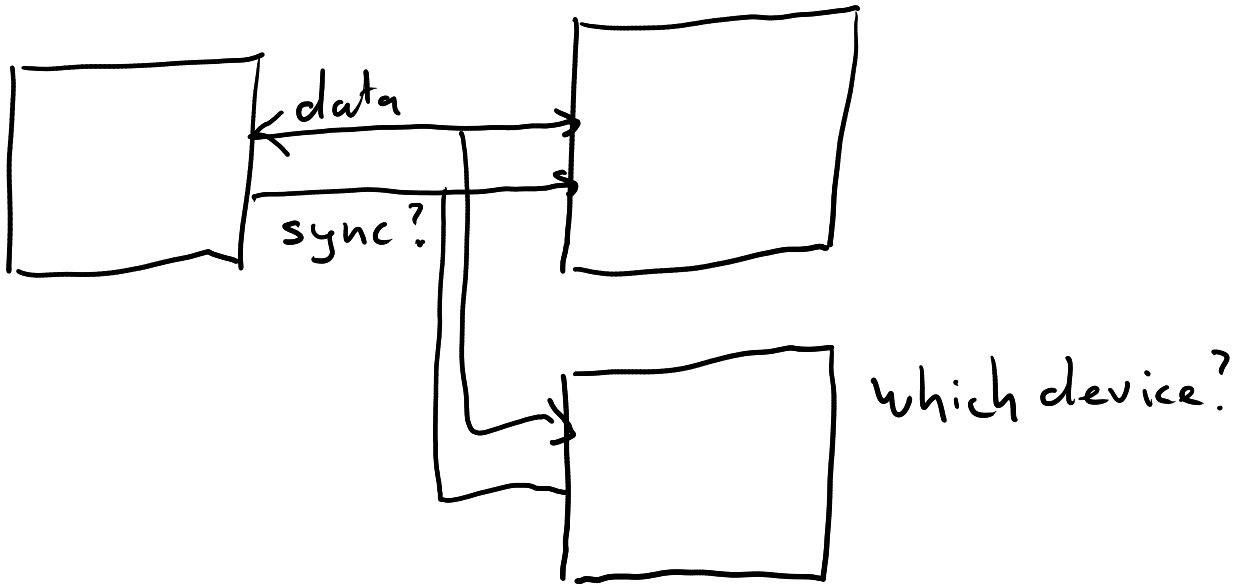
How could we use fewer wires?

Address/data is great for on-chip communication,
but what if I want to connect a [...] to it?

- Wires are bulky and cost money
- PCB traces must be routed
- I/O pins are limited (UPduino has ~38?)
- High-speed signals must have matching wire lengths

Send the bits one at a time!
This is called a **serial** bus.

Serial link



Serial link

With multiple devices

Terminology

- Most protocols have a "controlling" device

  One side has to control the clock, etc.

- Traditionally designed master / slave, but other terms are coming into widespread use:

  Controller / peripheral

  Host / device (used by USB)

  Data in / data out

A few reflections

- Words matter. Let's choose them carefully.
  Don't let "it's always been that way" stop us from making engineering a more inclusive field.

- But let's do so with gentleness and charity.

  "Do unto others as you would have them do unto you."

From the Hackaday comments:
"[I support this change, and it is long overdue.] Any other view has no legitimacy."

A bit of terminology

- **Bit-bang**: to implement a protocol "by hand",
  usually by writing software that controls GPIO pins.

  "Our microcontroller didn't have an SPI controller built in,
  so we had to bit-bang it."

  "The WS2812 LEDs use a funky one-wire protocol that you
  have to bit-bang."

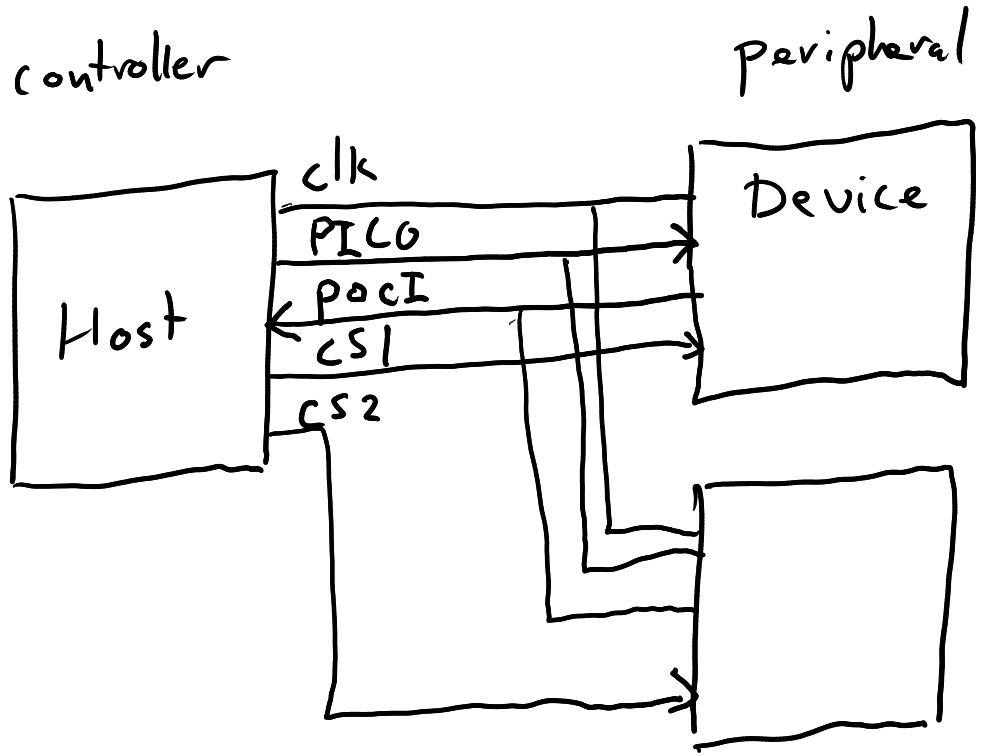
  (This will make a lot more sense in EE 14.)

Which one should I use?

Usually you don't get to pick!

SPI

- Clock and data
- Data lines are PICO (MOSI) and POCI (MISO)
  Or SDI / SDO, or SI / SO

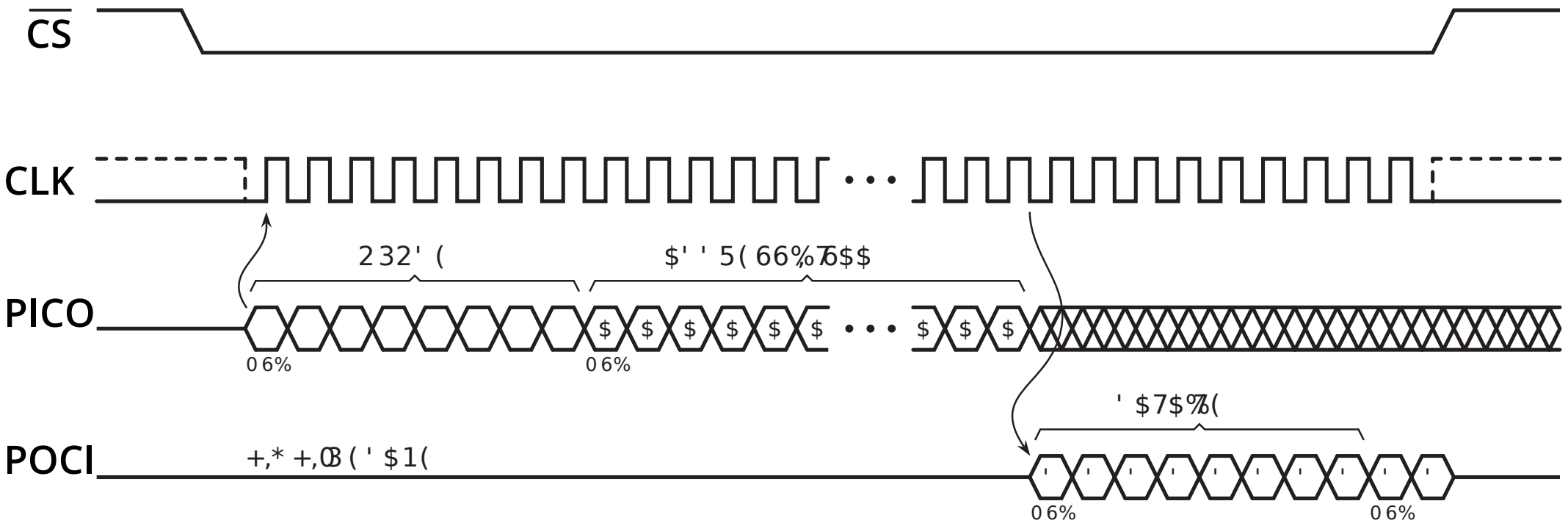


1. Pin Descriptions and Pinouts

Table 1-1. Pin Descriptions

Symbol	Name and Function	Asserted State	Type
$\overline{CS}$	**CHIP SELECT:** Asserting the $\overline{CS}$ pin selects the device. When the $\overline{CS}$ pin is deasserted, the device will be deselected and normally be placed in standby mode (not Deep Power-Down mode), and the SO pin will be in a high-impedance state. When the device is deselected, data will not be accepted on the SI pin. A high-to-low transition on the $\overline{CS}$ pin is required to start an operation, and a low-to-high transition is required to end an operation. When ending an internally self-timed operation such as a program or erase cycle, the device will not enter the standby mode until the completion of the operation.	Low	Input
SCK	**SERIAL CLOCK:** This pin is used to provide a clock to the device and is used to control the flow of data to and from the device. Command, address, and input data present on the SI pin is always latched in on the rising edge of SCK, while output data on the SO pin is always clocked out on the falling edge of SCK.	-	Input
SI (I/O_0)	**SERIAL INPUT:** The SI pin is used to shift data into the device. The SI pin is used for all data input including command and address sequences. Data on the SI pin is always latched in on the rising edge of SCK. With the Dual-Output and Quad-Output Read commands, the SI Pin becomes an output pin (I/O_0) in conjunction with other pins to allow two or four bits of data on (I/O_{3-0}) to be clocked in on every falling edge of SCK To maintain consistency with the SPI nomenclature, the SI (I/O_0) pin will be referenced as the SI pin unless specifically addressing the Dual-I/O and Quad-I/O modes in which case it will be referenced as I/O_0 Data present on the SI pin will be ignored whenever the device is deselected ($\overline{CS}$ is deasserted).	-	Input/Output

Figure 6-1. Read Array - 03h Opcode



SPI

PS/2 keyboard demo (not technically SPI, but similar)

I2C

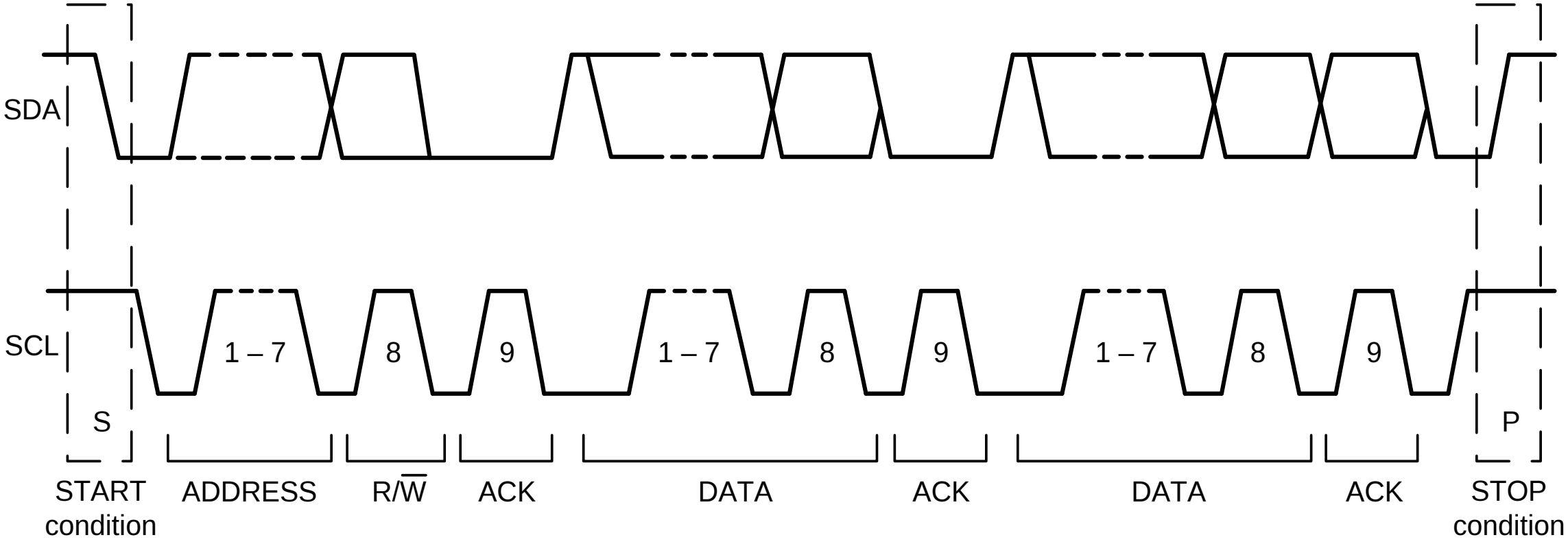
- Pronounced "eye-squared-see" or "eye-two-see"

  Sometimes also called "IIC" or two-wire-interface (TWI)

- No chip-select lines; use 7-bit address instead

- Clock (SCK) driven by host
  Data (SDA) shared by host and device(s)

I2C



Complete I²C Data Transfer

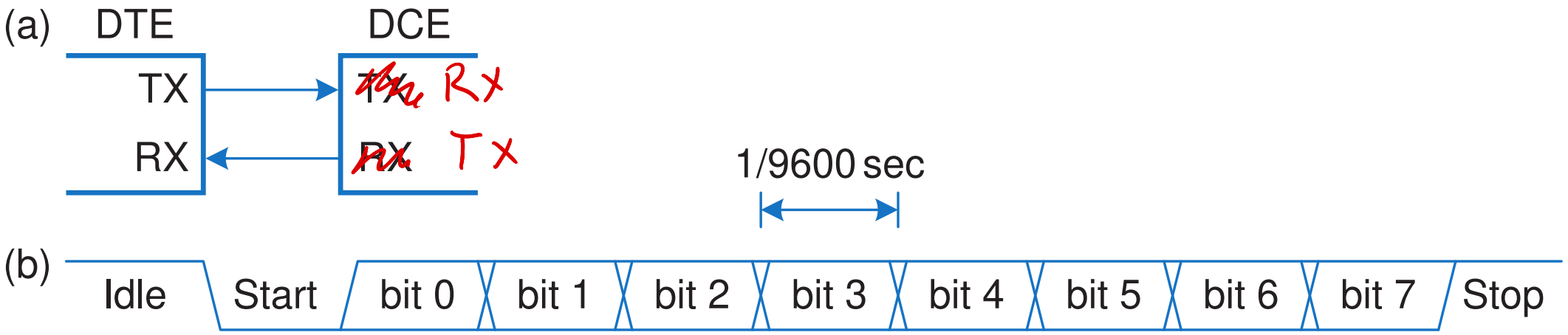
I2C

Wii nunchuck demo

UART

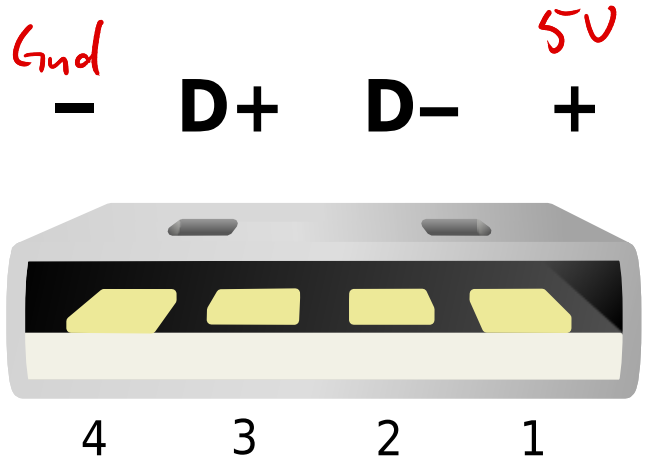
- As long as both sides agree on a clock speed, it's not necessary

  But you have to recognize when transmissions start!

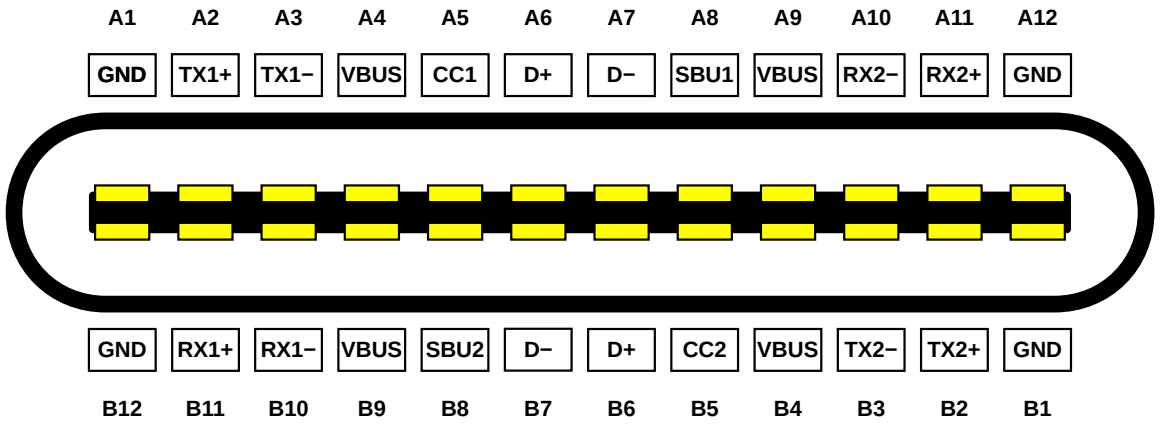


USB

USB-A plug



USB-C plug



(wikimedia)

	# wires	speed	multi-device?	complexity
SPI	3-4 1 / device	fast (10s MHz)	with CS	very simple
I^2C	2	slow (400 kHz)	yes	simple
UART	2	very slow (9600 baud) 115200 baud	no	simple
USB	2 kinda	wicked fast	star	absurd