

Lecture 02

Microcontrollers

CE346 – Microprocessor System Design
Branden Ghena – Fall 2022

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

Class Updates

- Reminder: no lab this Friday
 - I won't be there. Don't show up.
- My office hours are by request right now
 - Will be posting an office hours schedule as soon as possible
- Guide on personal lab setups posted on Campuswire
 - Totally optional
 - Also posted on the Canvas homepage

Today's Goals

- Explore microcontrollers
 - How do they compare to computers?
 - Purpose, capabilities, tradeoffs
- Describe history and state-of-the-art for microcontrollers
- Explore the microcontroller(s) on the Microbit

Outline

- **Microcontrollers and Computers**
- Microcontroller Design
- Microcontroller History

- Microbit microcontroller

What's inside a computer?

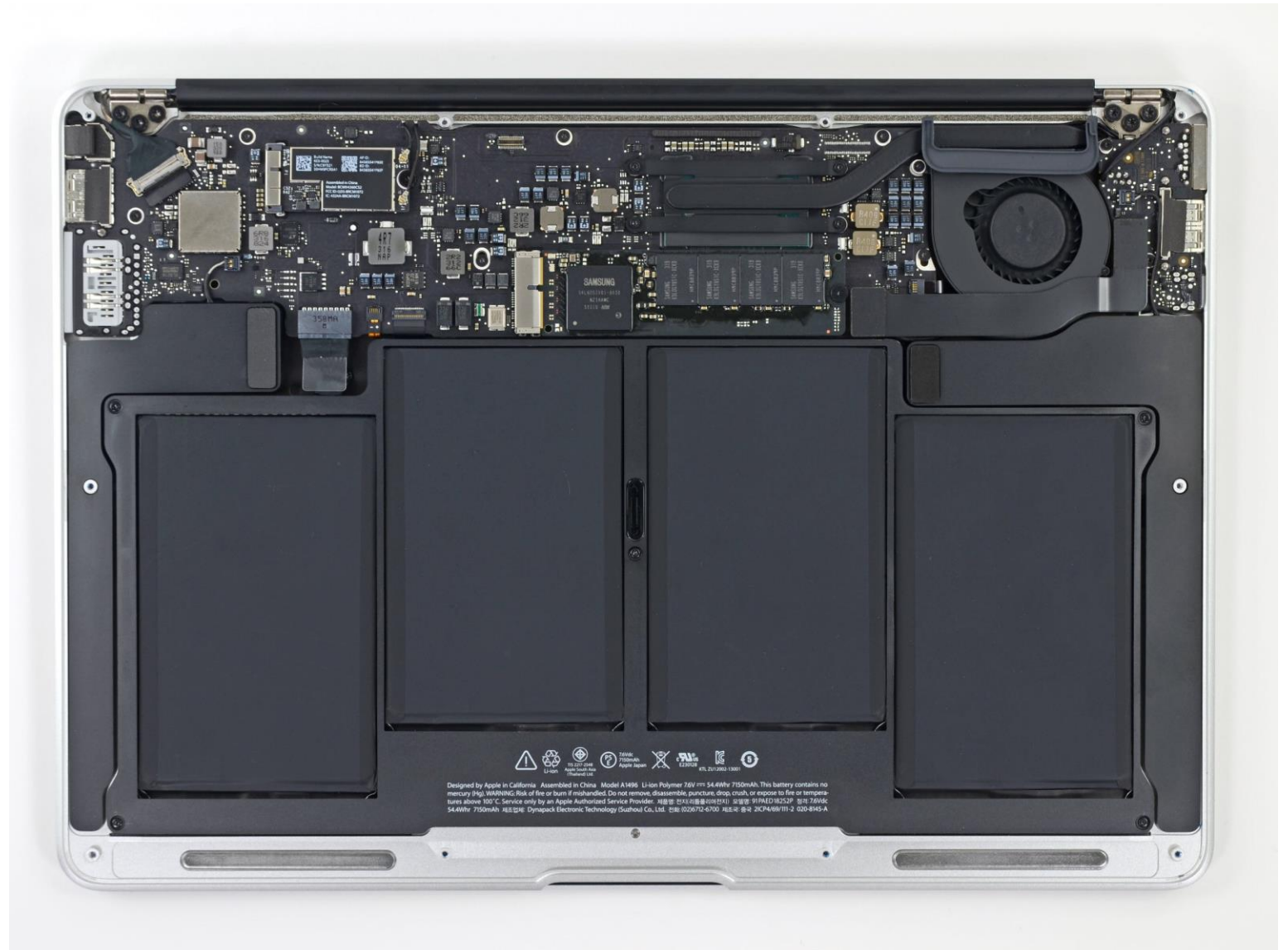
- Macbook Air (circa 2013, 2nd gen)
 - Modern computers are mostly similar
 - Intel Core i5 processor
 - 128 GB Flash storage
 - 4 GB DDR3 RAM
 - 1440x900 pixel display
- Picked because there is a teardown I like of an embedded device from the same year



<https://www.ifixit.com/Teardown/MacBook+Air+13-Inch+Mid+2013+Teardown/15042>

Unscrewing the bottom cover

- Top half is the motherboard
 - Holds and connects all the parts of the computer
- Bottom half are battery packs



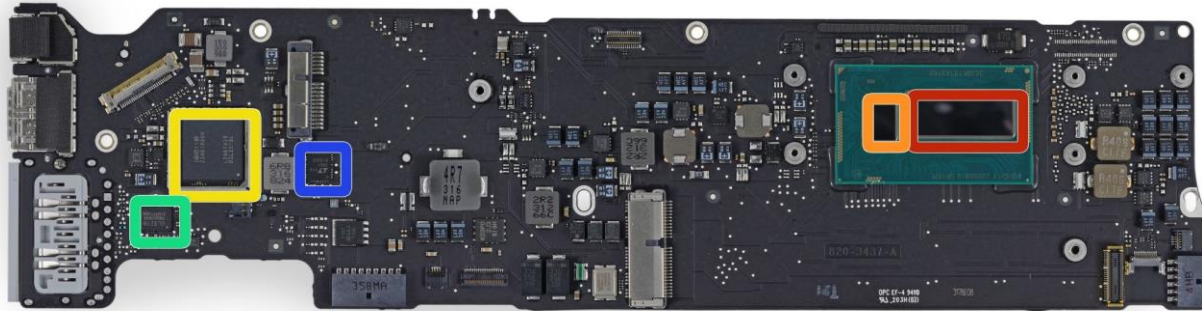
Non-volatile long-term storage: SSD

- The SSD is a module that connects through PCIe
- In modern laptops the SSD is often soldered directly onto the motherboard



Investigating the motherboard

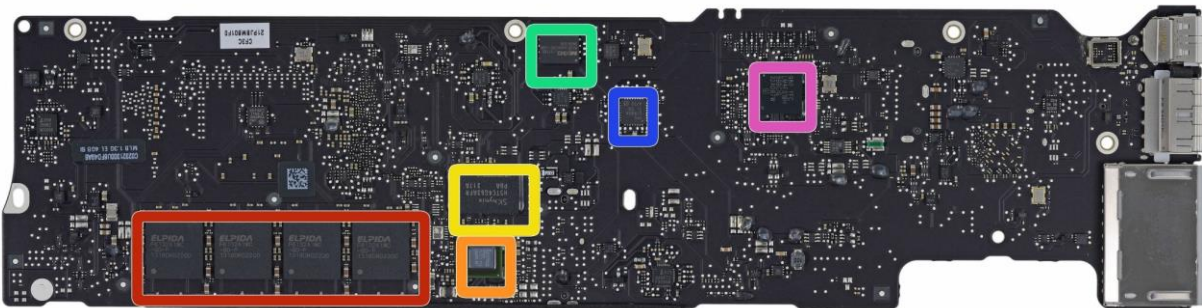
Front



- Red (front, right)
 - Intel core i5 processor

- Red (bottom, left)
 - Elpida LPDDR3 RAM, 4 GB

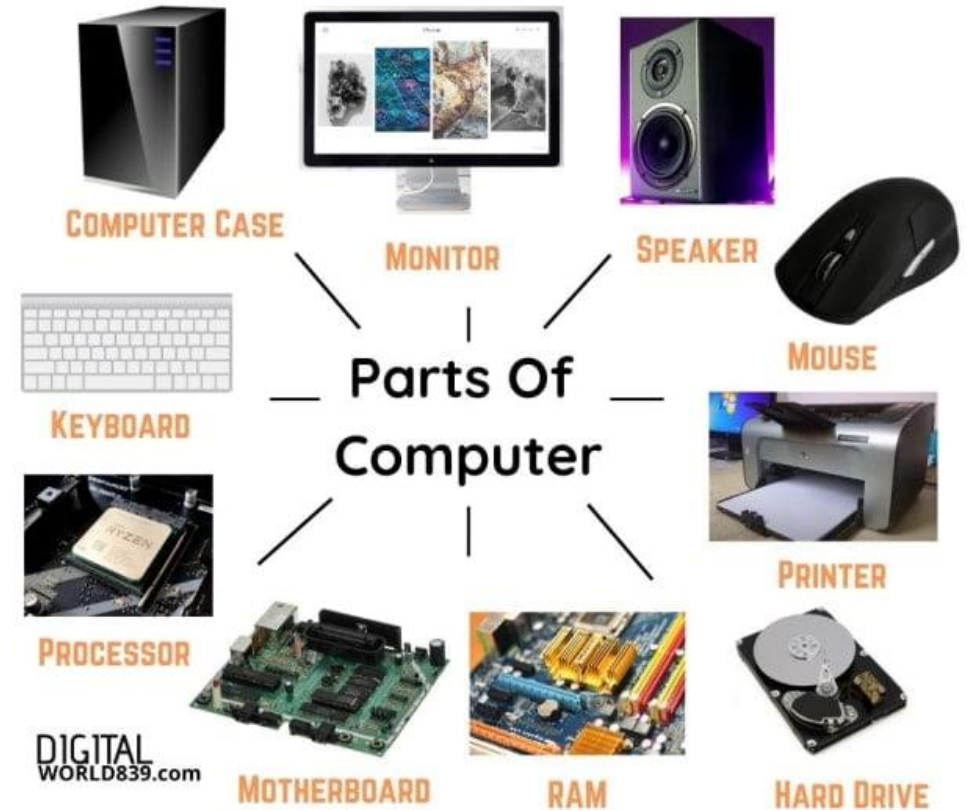
Back



- Other stuff are mostly radios, USB/Thunderbolt controllers, and power supply

Generalizing computer design

- Computers *usually* need
 - Processor
 - Memory (RAM)
 - Storage (Flash/SSD)
 - External communication
 - USB, Thunderbolt, SATA, HDMI, WiFi
 - Power management
 - Maybe batteries and charging
- Something to connect it all: motherboard



What's inside a Fitbit?



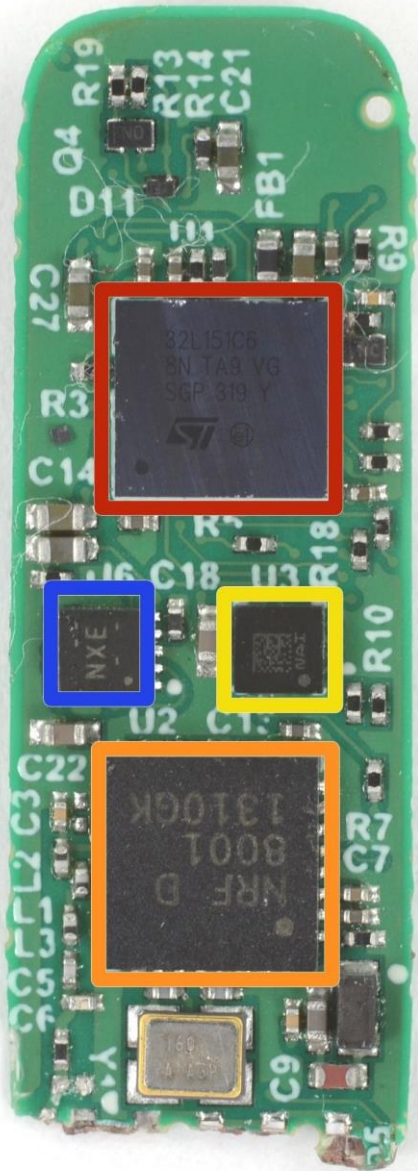
- Fitbit flex (circa 2013)
- Features
 - Counts your steps
 - Reports via Bluetooth Low Energy
 - Lights up some LEDs based on your goals
 - Vibrates when its battery is low

Fitbit teardown



Fitbit circuit board front

The back is uninteresting



- Red (top)
 - STMicro 32L151C6 Microcontroller
- Blue (left)
 - TI BQ24040 Battery Charger
- Yellow (right)
 - STMicro LIS2DH Accelerometer
- Orange (bottom)
 - Nordic nRF8001 Bluetooth Low Energy Radio



Fitbit as a computer

- Computers *usually* need

- Processor
 - Memory (RAM)
 - Storage (Flash/SSD)
 - External communication
 - USB, Thunderbolt, SATA, HDMI, WiFi
 - Power management
 - Maybe batteries and charging
 - Something to connect it all: motherboard
-
- Microcontroller
 - Bluetooth radio
 - Vibratory motor
 - Battery and power management
 - Circuit board

Sidebar: you could make a Fitbit yourself

- All those parts are commercially available, so you could make your own Fitbit if you wanted to
 1. Make a circuit board
 2. Buy those parts and solder them on
 3. Write some software
 4. Make a plastic case
 5. You've got a Fitbit!

Categories of computer chips

- Microprocessor (MPU or Processor)
 - CPU + Bus/Pins
 - No memory or peripherals
- Microcontroller (MCU)
 - CPU + Pins + Memory + Peripherals
 - Peripherals: separate hardware units within chip
 - Often I/O interfaces
- System on a Chip (SoC)
 - Microcontroller + Extensive Peripherals
 - Example: Radios or ML Accelerators
 - Essentially multiple chips combined
- Evolution of increased complexity over time



Break + Extend your thinking

- Microprocessor (MPU or Processor)
 - CPU + Bus/Pins
 - No memory or peripherals
- Microcontroller (MCU)
 - CPU + Pins + Memory + Peripherals
 - Peripherals: separate hardware units within chip
 - Often I/O interfaces



- **What are some downsides to using a microcontroller?**

Break + Extend your thinking

- Microprocessor (MPU or Processor)
 - CPU + Bus/Pins
 - No memory or peripherals
- Microcontroller (MCU)
 - CPU + Pins + Memory + Peripherals
 - Peripherals: separate hardware units within chip
 - Often I/O interfaces
- **What are some downsides to using a microcontroller?**
 - No choice about memory or peripherals
 - Less complicated designs, because they all have to fit in one chip

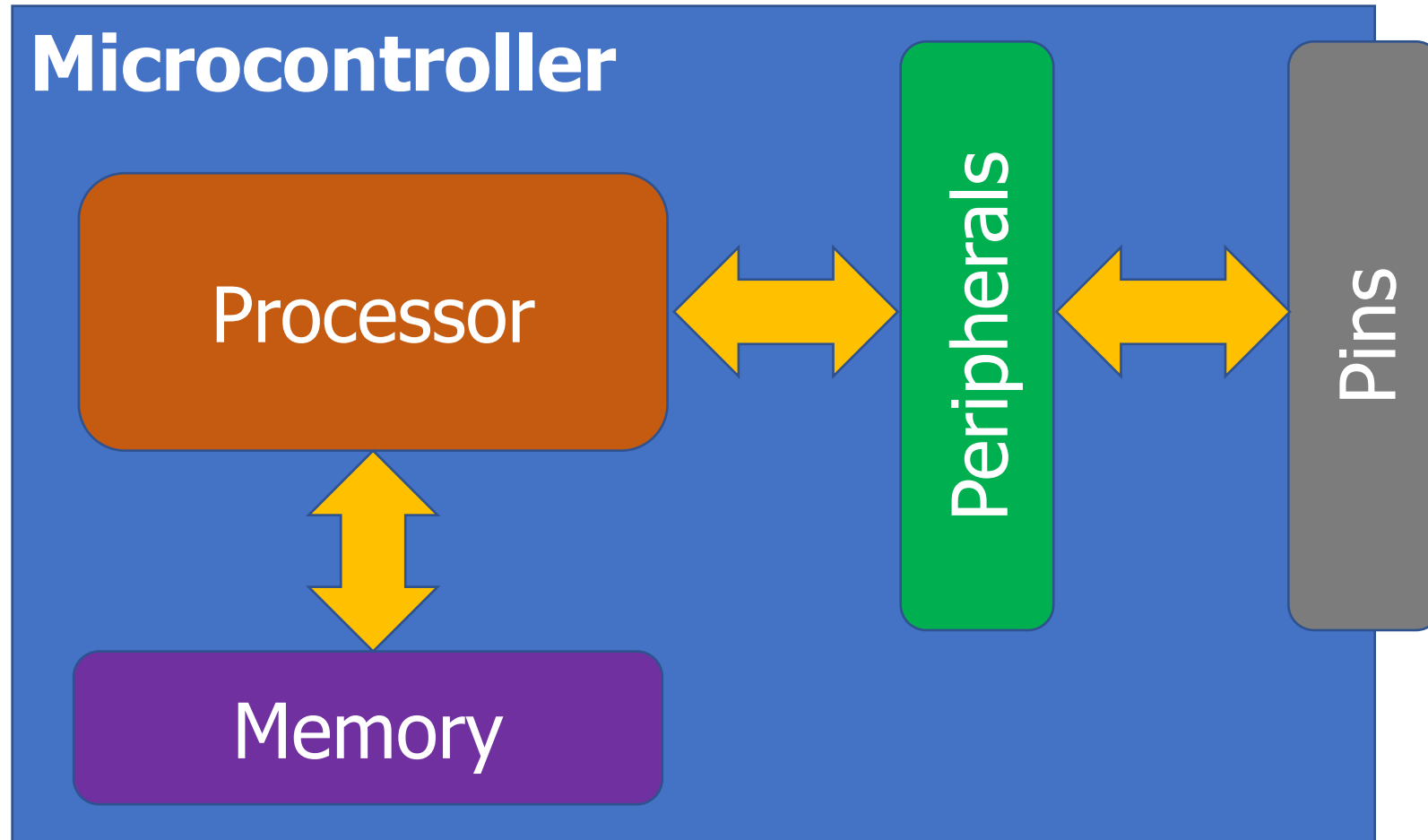


Outline

- Microcontrollers and Computers
- **Microcontroller Design**
- Microcontroller History

- Microbit microcontroller

Generic microcontroller block diagram



Processor

- Microcontroller needs to execute software instructions
- Emphasis: simple and reliable
 - i.e. **not** x86 CISC
- Often simple, in-order, short-pipeline RISC architectures
 - Literally a three-stage pipeline you may have learned in CS361
- Many different ISAs are possible
 - Custom ISAs and ARM (ARMv7E-M) are both common

Background: Instruction Set Architecture

- ISA includes the actual instructions as well as the model for how the computer interacts with memory
- Does the ISA for a microcontroller matter if you're not programming in assembly? (and I promise we won't)

Background: Instruction Set Architecture

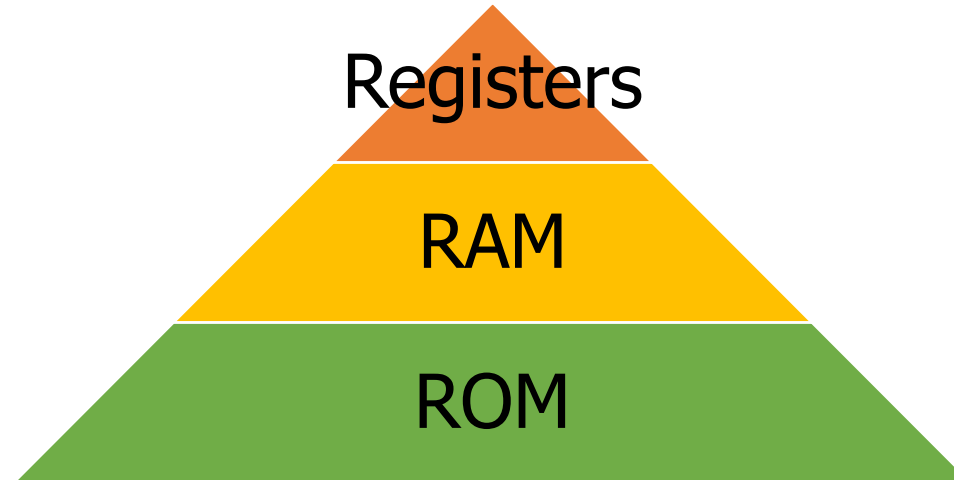
- ISA includes the actual instructions as well as the model for how the computer interacts with memory
- Does the ISA for a microcontroller matter if you're not programming in assembly? (and I promise we won't)
 - Yes
 - Differences in instruction efficiency and amount are important
 - Differences in compiler support are VERY important

Processor design choices for embedded

- How many bits is the architecture?
 - 8-bit and 16-bit not uncommon
 - 64-bit is very rare (who has that much memory?)
- What instructions are supported?
 - Normal stuff yes
 - What about single-cycle multiply?
 - What about floating-point operations?
 - Vector instructions?
- How fast does it run?
 - 1-100 MHz is common on modern systems (faster is more energy cost)
 - Occasionally MUCH slower (think 32 kHz)

Memory

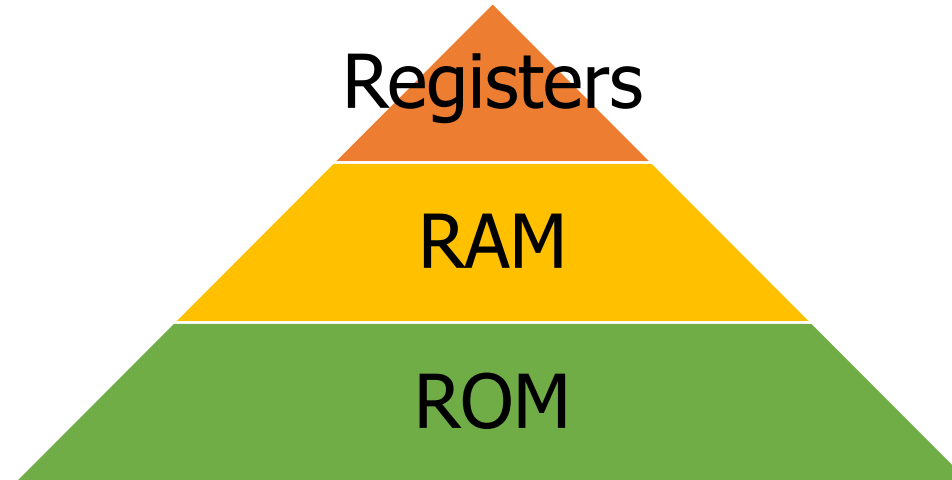
- Embedded memory hierarchy



- **What's missing? Why?**

Memory

- Embedded memory hierarchy



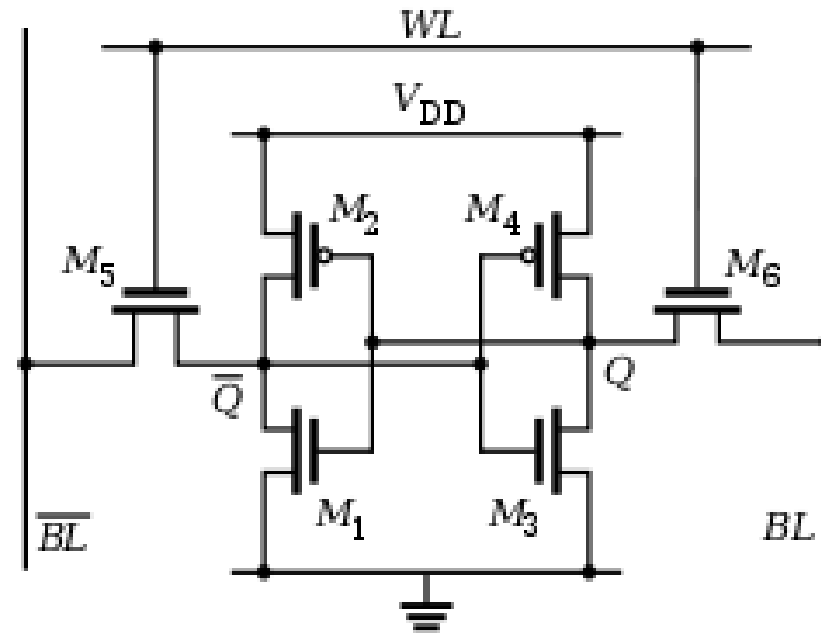
- **What's missing? Why?**
 - **Cache**
 - **Makes timing unreliable**

More modern microcontrollers often have optional instruction caches

Can be enabled by software

Volatile memory: SRAM

- Same design as registers in traditional computer systems
 - No need to refresh it -> lower energy costs
- Embedded systems use SRAM for all of their memory (no DRAM)
- Tradeoffs:
 - Way less energy cost
 - Faster to access
 - More expensive
 - Physically larger



Non-Volatile memory: Flash (usually)

- Same design as SSDs and Flash drives
- Memory is stored with no energy costs
 - Reading is lowish energy and quick
 - Writing is high energy and very slow
 - Writing also has to occur in blocks (e.g. 512 bytes)
- Concern: Flash has a limited lifetime $\sim 10,000$ writes
 - Only writing to Flash when you load code? Essentially forever
 - Recording data to Flash every second? 2.8 hours

SRAM (RAM) versus Flash (ROM) memory

- Size and time difference between non-volatile and volatile memory is *significantly* reduced from traditional computing
 - Non-volatile store 2-10x larger than volatile
 - Single-cycle RAM. Maybe two-cycles to Flash (to read)
- Major difference: energy and writability
 - SRAM is low-energy to read and write (no refresh needed)
 - Flash is lowish-energy to read, but very high energy to write
- Hierarchical structure is not enforced
 - Same address space for RAM and Flash (very different from traditional)
 - Run instructions right inside of Flash
 - Keep variables in RAM (or const variables in Flash)

Memory design choices

- How much memory can we fit without making it too high cost or too high power?
 - The answer has slowly increased over time
 - 15 years ago: 2 KB RAM – 32 KB Flash
 - Today: 256 KB RAM – 1 MB Flash
- How to provide memory safety?
 - Usually no virtual memory (all addresses are real addresses)
 - Nothing stops arbitrary memory overwriting
 - Modern systems: Memory Protection Unit
 - Specify a small number of ranges and permissions

Peripherals

- Hardware modules that perform some action
- Common examples
 - Control digital input and output pins
 - Read analog inputs
 - Send messages over various simple wire protocols (UART, SPI, I2C)
 - Set and check timers
- Less common examples
 - Cryptography accelerators
 - Complicated wire protocols (USB, CAN)
 - Wireless radios (BLE, 802.15.4, WiFi)
- We'll spend most of class learning various peripherals

Peripheral design choices

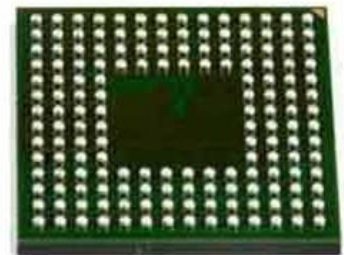
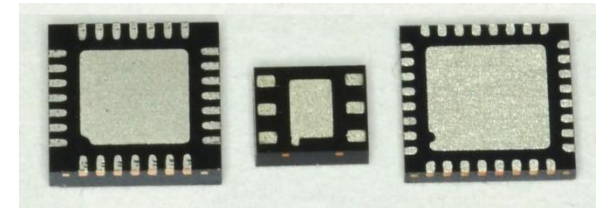
- More peripherals mean more use cases
 - But also means more cost for the chip
 - Peripherals occupy silicon
 - Most peripherals will go unused for a given application...
- Flexibility tradeoffs within a given peripheral
 - One capability is easier to use
 - Many capabilities are more useful



Apple A8 SoC

Pins

- Peripherals need to electrically connect to outside world
 - Attach to pins on the exterior of the chip
- More pins allow for more connections
 - At increased cost and size
- Pin layouts can add more pins for cheaper
 - But make soldering and debugging difficult



Internal connections to pins

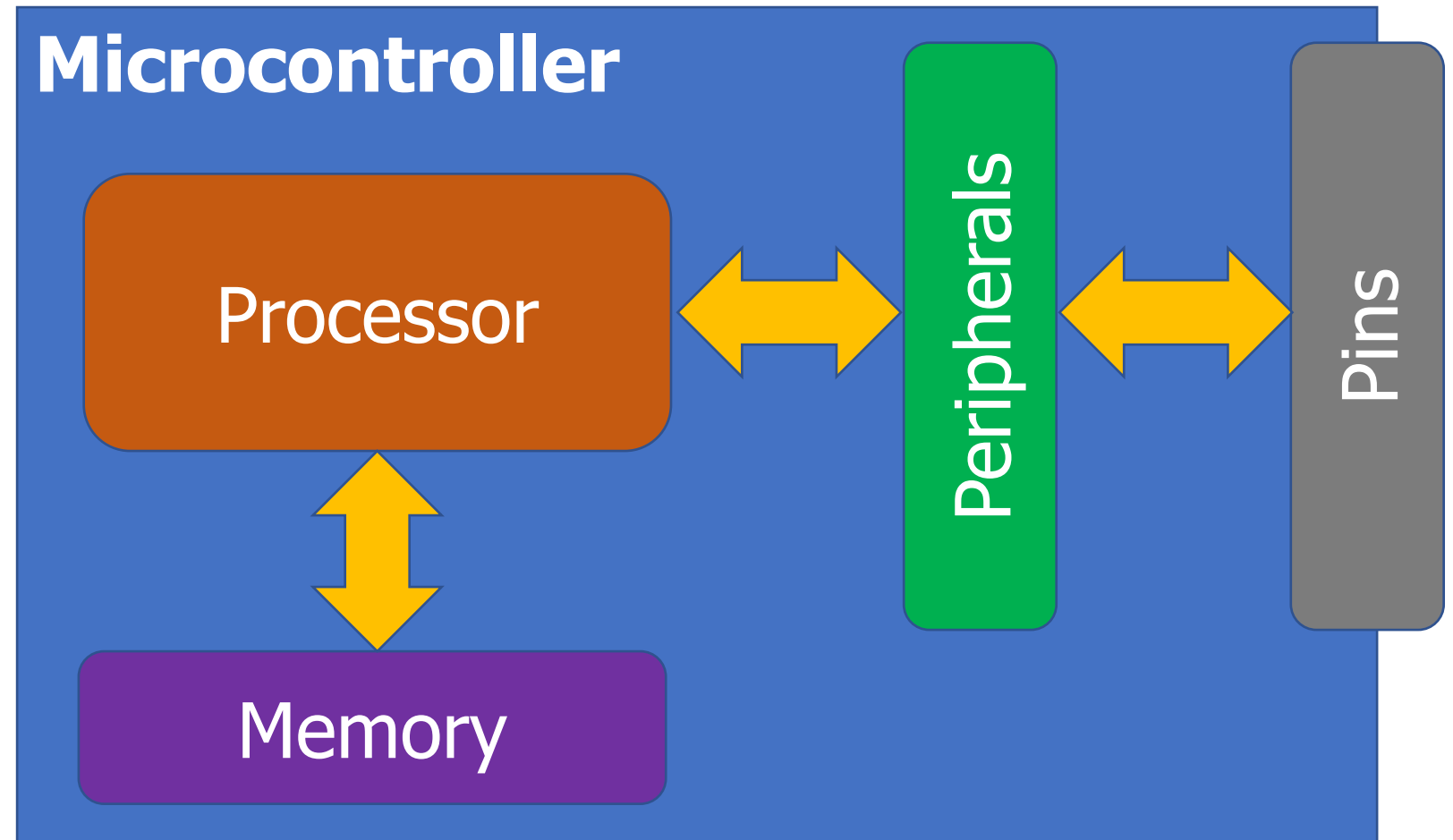
- Peripherals need to connect to external pins
 - Can any peripheral connect to any pin, or are there limited mappings?
 - Modern microcontrollers allow any-to-any connections

- Older MCUs had mapping tables and pin selection was more challenging

Table 3-1. 100-pin GPIO Controller Function Multiplexing (Sheet 1 of 4)

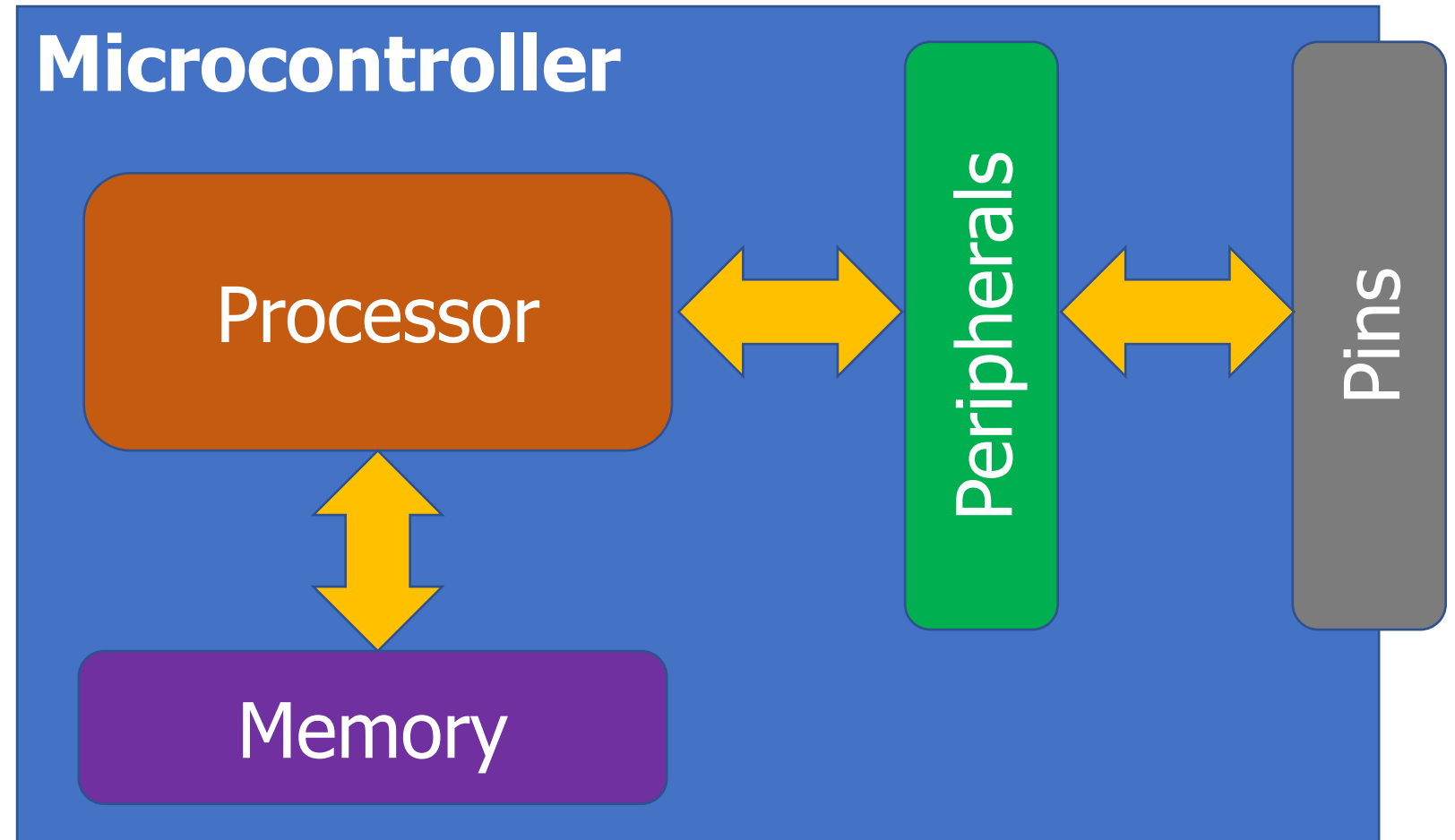
ATSAM4LC		ATSAM4LS		Pin	GPIO	Supply	GPIO Functions							
QFN	VFBGA	QFN	VFBGA				A	B	C	D	E	F	G	
5	B9	5	B9	PA00	0	VDDIO								
6	B8	6	B8	PA01	1	VDDIO								
12	A7	12	A7	PA02	2	VDDIN	SCIF GCLK0	SPI NPCS0						CATB DIS
19	B3	19	B3	PA03	3	VDDIN		SPI MISO						
24	A2	24	A2	PA04	4	VDDANA	ADCIFE AD0	USART0 CLK	EIC EXTINT2	GLOC IN1				CATB SENSE0
25	A1	25	A1	PA05	5	VDDANA	ADCIFE AD1	USART0 RXD	EIC EXTINT3	GLOC IN2	ADCIFE TRIGGER			CATB SENSE1
30	C3	30	C3	PA06	6	VDDANA	DACC VOUT	USART0 RTS	EIC EXTINT1	GLOC IN0	ACIFC ACAN0			CATB SENSE2

What haven't we talked about?



What haven't we talked about?

- **Power**
- **Programming**
- **Others?**
 - Clocks
 - Antennas



Powering microcontrollers

- Usually require a specific voltage
 - E.g. 5 volts, 3.3 volts, 1.8 volts
 - Must be stable and supply enough current (or MCU “browns out”)
 - Noisy power supply can be an issue
- Some microcontrollers have wider ranges of acceptable voltages
 - Need to pay attention to acceptable range on I/O though

Programming microcontrollers

- JTAG (Joint Test Action Group)
 - Hardware built into the microcontroller for testing purposes
 - Can arbitrarily read/write memory
 - Can single step process too, at runtime!
 - GDB can connect to it! (sort of)
- Bootloaders
 - Software runs on the microcontroller at boot that waits a short time for someone to contact it and upload code
 - Via I/O pins
 - Convenient, but sometimes unreliable

Break + Open Question

- How little memory (RAM and/or ROM) can a computer have and still be useful?

Break + Open Question

- How little memory (RAM and/or ROM) can a computer have and still be useful?
 - If we want to run C code, we need a stack and a global data section
 - Function calls may require local variables to be placed in the stack
 - To use less memory, we'd need to write applications in registers
 - Most variables could *probably* fit in registers?
 - So the limit could get *very* low
 - Enough ROM space for some useful code + a tiny bit of RAM
- Real-world answer coming on the next slide

Outline

- Microcontrollers and Computers
- Microcontroller Design
- **Microcontroller History**
- Microbit microcontroller

Older microcontrollers (90s-00s)

- Focus: cheap, small computer systems
- PIC
 - 8, 16, and 24-bit MIPS architecture
 - PICAXE available with Basic interpreter
- AVR
 - 8-bit custom architecture (AVR architecture)
 - Used in Arduinos
 - AT Tiny 4: \$0.30 per unit
 - 4 I/O pins, 32 Bytes RAM, 512 Bytes ROM



More capable microcontrollers (00s-10s)

- Focus: diversify into extreme low power and more capable systems
- MSP430
 - 16-bit custom architecture (MSP430 architecture)
 - Capable, but also extremely low power
 - $<1 \mu\text{A}$ sleep current
- STM32, Atmel SAM series (Cortex-M0, M3, M4, M4F)
 - 32-bit ARM architecture (ARMv7)
 - Leverage success of ARM on smartphones
 - Every peripheral under the sun. Plus a variety of memory choices

Cortex M Series

- A number of options for increasing capabilities
- In practice:
 - Cortex-M0+ for low-end systems
 - Cortex-M4 for high capability systems

ARM Cortex-M instruction variations

Arm Core	Cortex M0 ^[2]	Cortex M0+ ^[3]	Cortex M1 ^[4]	Cortex M3 ^[5]	Cortex M4 ^[6]	Cortex M7 ^[7]
ARM architecture	ARMv6-M ^[9]	ARMv6-M ^[9]	ARMv6-M ^[9]	ARMv7-M ^[10]	ARMv7E-M ^[10]	ARMv7E-M ^[10]
Computer architecture	Von Neumann	Von Neumann	Von Neumann	Harvard	Harvard	Harvard
Instruction pipeline	3 stages	2 stages	3 stages	3 stages	3 stages	6 stages
Thumb-1 instructions	Most	Most	Most	Entire	Entire	Entire
Thumb-2 instructions	Some	Some	Some	Entire	Entire	Entire
Multiply instructions 32x32 = 32-bit result	Yes	Yes	Yes	Yes	Yes	Yes
Multiply instructions 32x32 = 64-bit result	No	No	No	Yes	Yes	Yes
Divide instructions 32/32 = 32-bit quotient	No	No	No	Yes	Yes	Yes
Saturated instructions	No	No	No	Some	Yes	Yes
DSP instructions	No	No	No	No	Yes	Yes
Single-Precision (SP) Floating-point instructions	No	No	No	No	Optional	Optional
Double-Precision (DP) Floating-point instructions	No	No	No	No	No	Optional

Modern system-on-chips (10s-?)

- Focus: increase memory and capability
 - Include radios in the same chip
- nRF51 and nRF52 series
 - 32-bit ARM microcontrollers (Cortex M)
 - Include 2.4 GHz radio: Bluetooth Low Energy and 802.15.4/Thread
- Others have followed this same path
 - TI CC26XX
 - Apollo3
 - STM32WL

Future microcontrollers

- Multi-core systems
 - Not for performance, but for separation of concerns
 - Run radio code on one core
 - Application code goes on the other core
- Also allows BIG.little architecture
 - Higher performance core when interpreting data
 - Lower performance core for sampling sensors
- nRF53 series is already doing this
 - But support is still fairly nascent

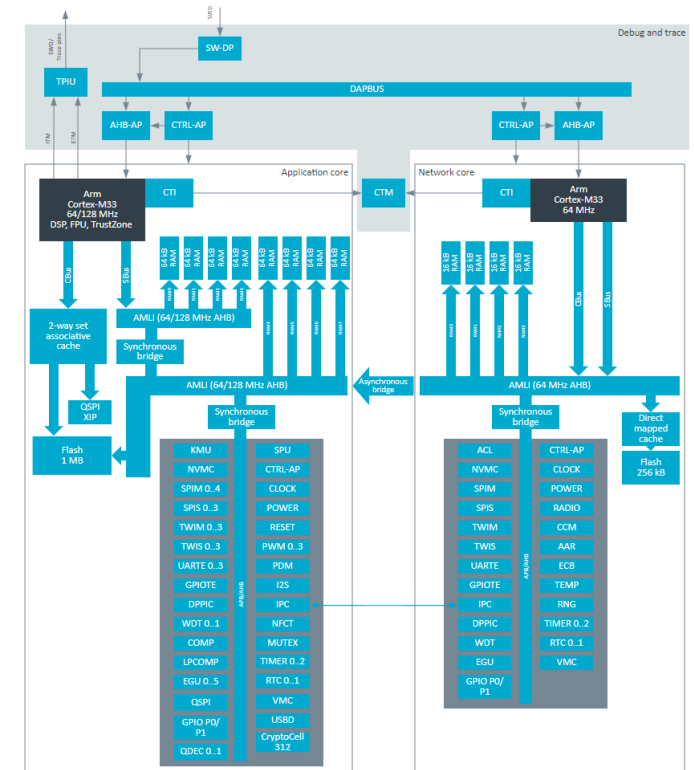


Figure 1. Simplified block diagram

Break + Open Question

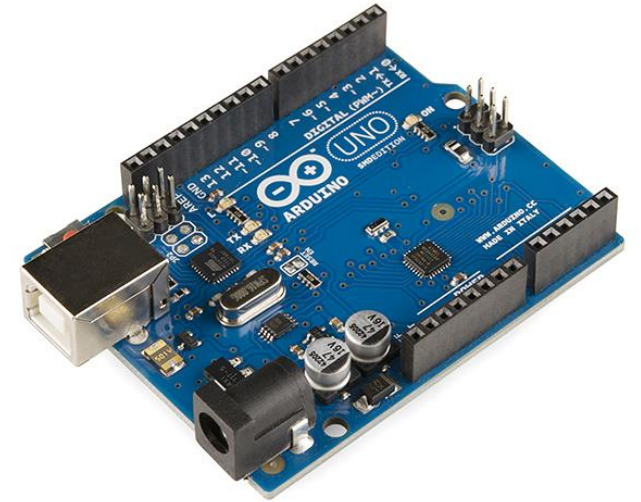
- Consider the lifecycles of computer systems
 - i.e., how long are people still actively using them for?
 - **How long does a processor generation need to be supported?**
 - **Think Intel Core i7-7500U or Apple M1**
 - **How long does a microcontroller chip need to be supported?**
 - **Think nRF52833 or MSP430**

Break + Open Question

- Consider the lifecycles of computer systems
 - i.e., how long are people still actively using them for?
 - **How long does a processor generation need to be supported?**
 - **Think Intel Core i7-7500U or Apple M1**
 - After some number of years, new products no longer use it
 - Old products eventually wear out or are unsupported
 - **How long does a microcontroller chip need to be supported?**
 - **Think nRF52833 or MSP430**
 - New products might still use them years later, because redesigning embedded systems is hard and new chips aren't that much better

Popular components rarely die

- Majority of popular older options still exist
 - Designers can trade off cost, capability, and efficiency alongside “modern features”
- Upgrading for an existing product is unlikely
 - Large cost, little benefit
- Example: Arduino still primarily uses AVR
 - Works just fine for its needs
 - Also releases new boards with nRF52s (Arduino Nano 33 BLE)



Outline

- Microcontrollers and Computers
- Microcontroller Design
- Microcontroller History

- **Microbit microcontroller**

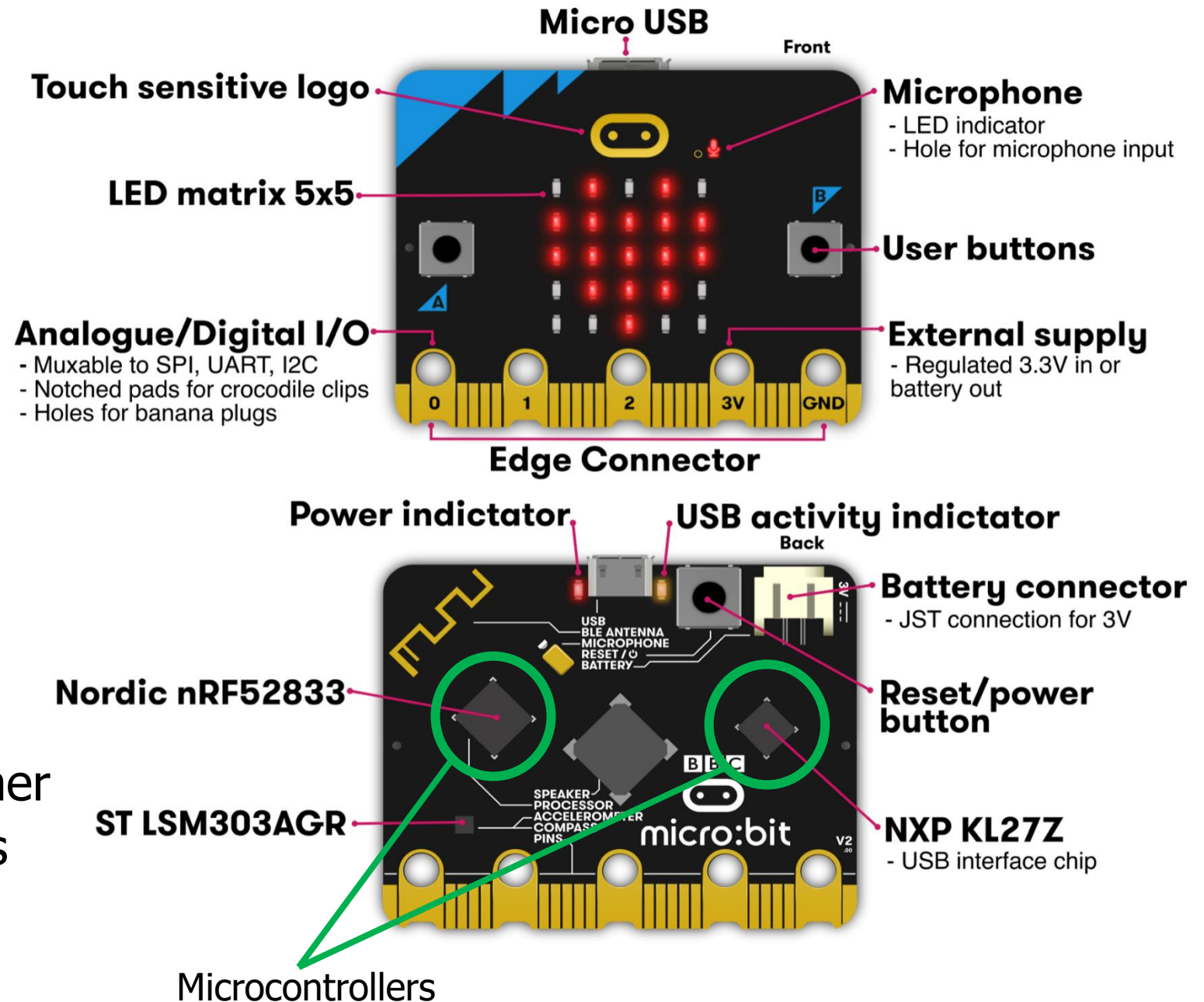
Micro:bit v2

- Circuit board

- Entire thing
- a.k.a "Dev Board"
- a.k.a PCB
(Printed Circuit Board)

- Microcontroller

- The computer on it
- Microbit has two
 - One as a programmer
 - One for applications



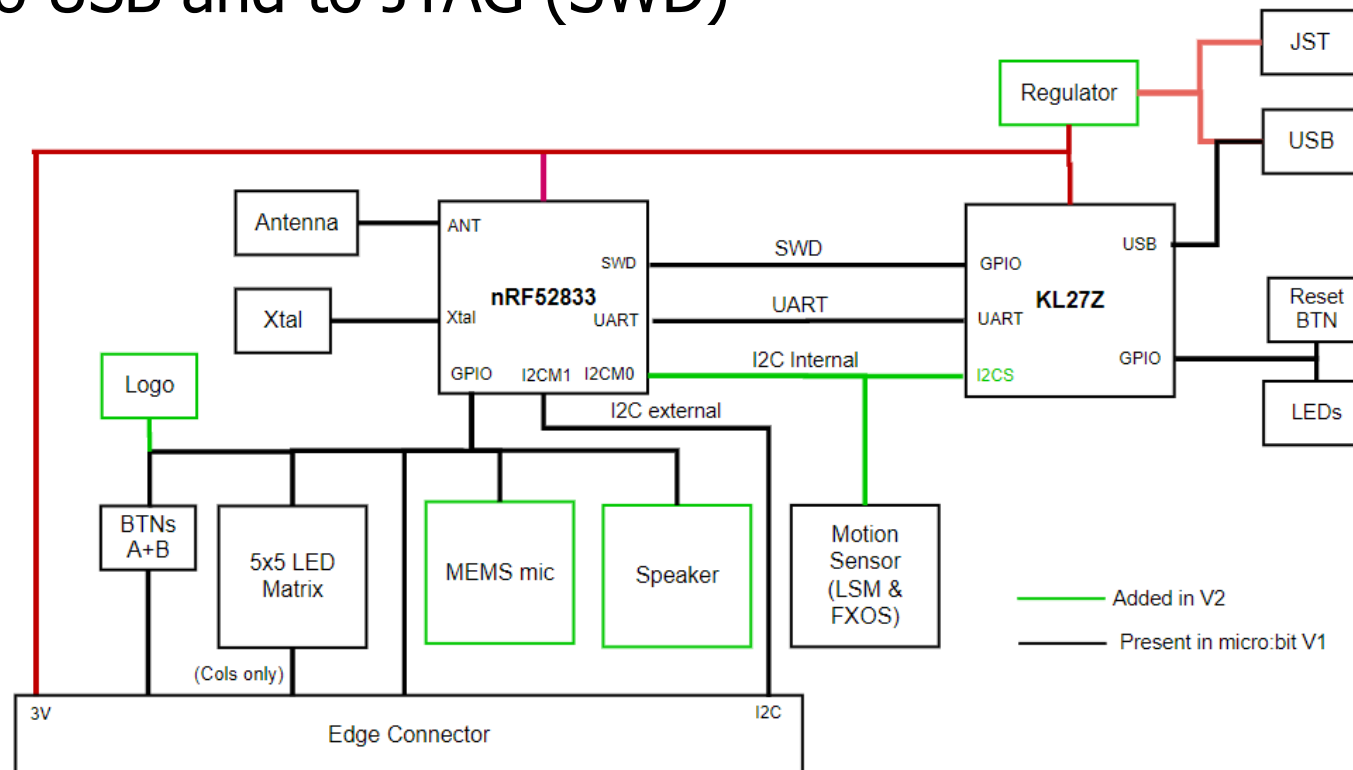
nRF52833 – Microcontroller on the Micro:bit v2

- 32-bit ARM Cortex-M4F microcontroller
 - 64 MHz core, 128 KB RAM, 512 KB Flash
 - Floating point support
 - 2.4 GHz Radio: Bluetooth Low energy / 802.15.4
 - Various peripherals
 - ADC, PWM
 - I2C, UART, SPI, USB
 - RNG, 32-bit Timers, Watchdog, Temperature
 - Up to 42 I/O pins



KL27Z – Programmer on the Micro:bit v2

- 32-bit ARM Cortex-M0+ microcontroller
 - 48 MHz core, 16 KB RAM, 256 KB Flash
- Acts as a programming interface to nRF52833
 - Connects to USB and to JTAG (SWD)



Microbit version 2.2

- Circa 2020-??? there has been a global chip shortage
 - Lots of demand for electronics
 - Reduced demand due partly to pandemic
- Microbit had to come out with a revision
 - Replaced KL27Z (the programmer chip) with a WAY more capable nRF52820
- This has caused some annoyances for lab that I need to fix
 - The programming interface changed a little

To the datasheet!

- nRF52833 Product Specification

- Online:

- https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52833%2Fkeyfeatures_html5.html

- PDF: https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf

Outline

- Microcontrollers and Computers
- Microcontroller Design
- Microcontroller History

- Microbit microcontroller(s)