EE 186: Introduction to Embedded Systems

Zerina Kapetanovic

Fall 2026

Have you ever wondered…

- How do thing ***really*** work inside that Fitbit fitness tracker?
- How do sensors interface with a microcontroller?
- How is data sent from one “smart” device to another? To the cloud?
- How do we program these devices to do what we want?

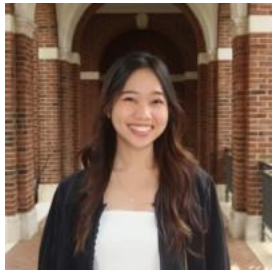
We are going to answer these questions and more, in this class!

Course Overview

Course Staff

Zerina Kapetanovic

Assistant Professor



Belle Angkanapiwat

Graduate Student (EE)

Lan Mei

Graduate Student (EE)

Pete Mahowald

Adjunct Lecturer

Learning goals for this class

Understand how wireless embedded systems work

Master the fundamentals and build your own wireless embedded system!

Prerequisites

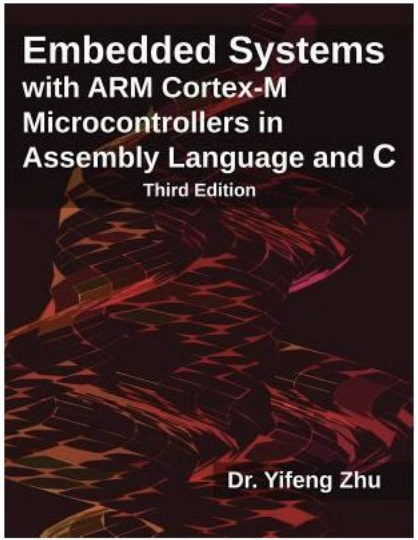
- EE 40M: An Intro to Making: What is EE
- CS 106B: Programming Abstractions

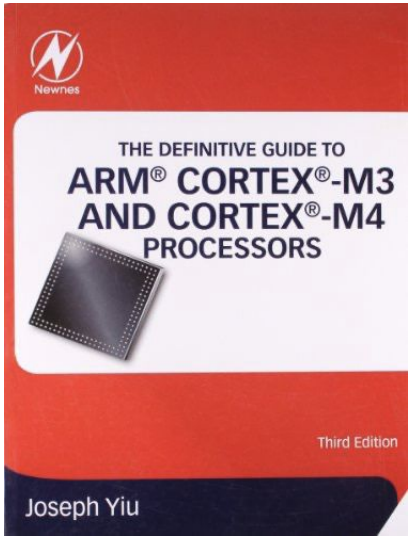
Not required, but helpful:

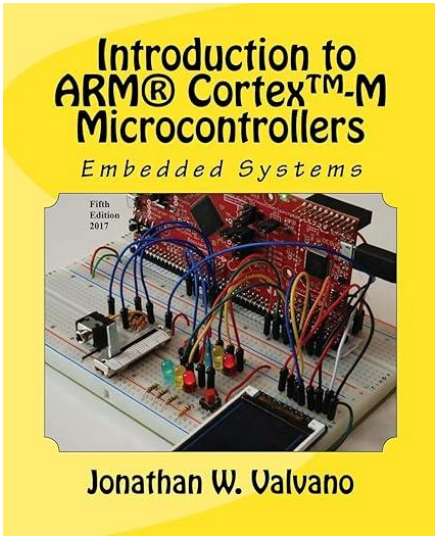
- CS 107E: Computer Systems from the Ground Up
- EE 108: Digital System Design

Books

The following books are good resources:







Other Resources

- ARM
  - ARMv7 Architecture Reference Manual
  - AMBA Advanced High-performance Bus (AHB) Protocol Specification
  - AMBA Advanced Peripheral Bus (APB) Protocol Specification
- STM32
  - STM32 Reference Manual
  - STM32 Datasheet
  - STM32CubeIDE
  - STM32CubeMX
  - STM32 Videos
  - Nucleo-144 User Manual
  - Nucleo Development Board Website

AI Policy

- AI can be a useful tool to **help you learn the materials** in this class and you are welcome to use it

- Using generative AI tools to substantially complete an assignment or exam (e.g., by entering exam or assignment questions) **is not permitted**.

- You are required to acknowledge the use of generative AI (other than incidental use) and default to disclosing such assistance when in doubt.

Office Hours

- The course will start off with reduced office hours
- **Please fill out the survey** that was sent out so we can pick times that work well for everyone
- If you can't make office hours, please email us! We are happy to find another time to meet with you

We are counting on you to let us know if office hours need to be increased. Please let us know!

- ARM Architecture
- Assembly and Machine Code
- Going from Assembly to C
- Debugging
- Clocks
- Timing
- Interrupts

- Interfacing with Peripherals
- Buses (UART, SPI, I2C)
- Memory
- Analog
- ADC/DAC
- Wireless
- Special Topics: Energy harvesting, PCB design

Class Schedule

Week	Topics
1	• Introduction, overview, architecture, GPIO Primer • ISA, Assembly, ABI
2	• Memory Mapped I/O, APB, MMIO + GPIO • Interrupts
3	• Interrupts • Timers, PWM
4	• SPI, I2C, UART • Project Selection and Team Forming
5	• Analog Signals • Midterm Review
6	• Midterm • ADC/DACs

Week	Topics
7	• No Class. Democracy Day (November 4th) • Wireless Communication
8	• Energy Harvesting • Printed Circuit Board Design
9	• Special Topics • No Lecture. Work on Final Project
10	• No Class. Thanksgiving Break.
11	• No Class. Work on Final Project.
12	• Final Project Demos and Presentations

NOTE: Lecture schedule is subject to change

Course Grading

Labs	30%
Homework	5%
Midterm	20%
Final Project	40%
Final Project Presentation	5%
Total	100%

Lectures

VS.

Labs

- Overview and perspective on the field of embedded systems
- Foundational learning of embedded systems concepts
- Provide background so students can understand the datasheet

- Take a deep-dive, explore, and apply the concepts learned in class
- Work with real embedded hardware and debugging tools

Lab Assignments

- An opportunity to apply the concepts you learned in class!
- Labs must be completed individually
- Estimate 5-6 hours per lab…some will be shorter and some will be longer

Lab Assignments

There are four labs in this class:

Lab 1: Hello World!
Assigned: Week 2

Lab 2: Sorry to Interrupt!
Assigned: Week 3

Lab 3: I^2C You!
Assigned: Week 4

Lab 4: Bit by Bit, Hertz by Hertz!
Assigned: Week 5

Lab Assignments: Getting Started Videos

- We have “getting started” videos for each lab assignment
- Watching the videos is not required **but recommended**
- If you are having trouble getting started with the labs, the videos can help!

Quiz Sections

- 5-minute individual section time to go over your lab submission
- Online over Zoom
- Quiz sections are recorded
  - Allows us to review the video for grading if necessary
  - If you have questions about your grade, it allows us to go back and review your answers together
- Sign-up sheet for each quiz section will be made available

Lab Due Dates

- HW and labs are released on Thursdays after lecture
- HW is due the following Tuesday
- Labs are due the Tuesday after HW

	SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
Week 1					Lab 1 Released		
Week 2			HW 1 Due				
Week 3			Lab 1 Due		Lab 2 Released		
Week 4			HW 2 Due				
Week 5			Lab 2 Due		Lab 3 Released		

Exam

There is **one midterm exam** that will be on **October 27th** (Week 6)

No final exam.

Closed book. You are allowed one page of notes (A4 paper).

There will be a midterm review the week prior to the exam.

Final Project

- **Goal:** learn how to build a wireless embedded system
- The final project is the primary focus towards the end of the course
- Some lectures are canceled so that you have time to work on your final project
- You will work in groups of 3-4
- **Extremely important to start early!**
- **REQUIRED:** It is required that you attend the lecture on **October 15th** for project selection and team forming
- Project idea slides due **October 13th**
    - 2-3 slides describing your project idea
    - More details coming soon!

Final Project

- Your group is required to demo the project to the course staff prior to the poster session
  - Sign-up sheet will be made available for demos

- We will have a public poster session and demo at the end of the quarter
  - December 11th
  - 12:15 – 3:15 PM

Late Policy

< 24 hours: -10%

< 48 hours: -30%

< 72 hours: -60%

< 72 hours: not accepted

If you have any concerns about the deadlines due to extenuating circumstances, please reach out to course staff **before the deadline**.

Joint Work Policy

- Homework can be completed in groups
- Labs will be completed individually
  - All submitted lab work must be from the students own efforts, and not any other sources
- 👍 OK
  - Studying together for exams
  - Discussing lectures or readings
  - Talking about general approaches
  - Help in debugging, tools peculiarities, etc.
- 👎 Not OK
  - Developing a lab together
  - Submitting work that you did not generate

Course Website, Assignment Submissions, etc.

- Assignments will be submitted through Gradescope
- Ed for discussion
- **Course Website:** ee186.stanford.edu

Markers for Success

Be curious and motivated to learn!

How to thrive:

Leverage our resources, support, and feedback
Learn by doing. Ask for help and help others!

This is a new (ish) course! We'd love your feedback so we can keep improving!

What is an embedded system?

Computing system embedded with things/gadgets – nearly everything has an embedded systems platform

Washing Machine

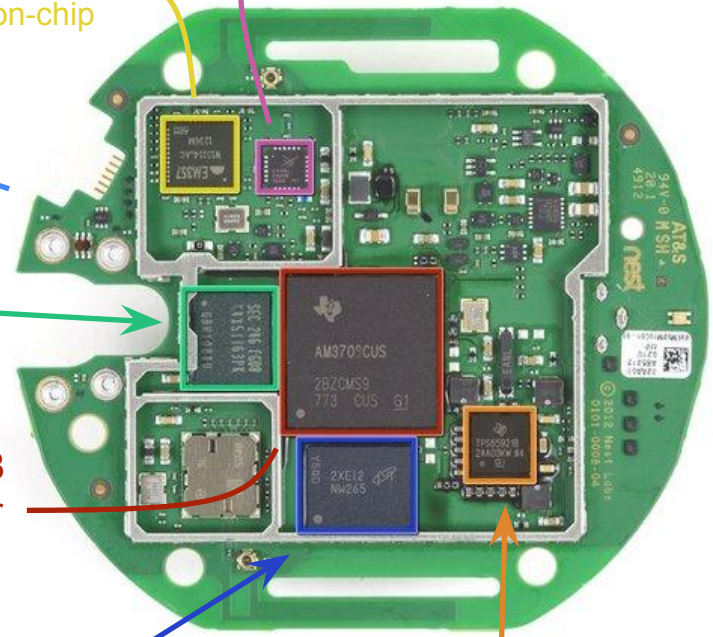
Calculator

Apple Watch

Amazon Echo Dot

Wireless Embedded Systems

Example of Wireless Embedded System



Integrated ZigBee/902.15.4 system-on-chip

High-power 2.4GHz 802.15.4 front-end module

DRAM

ARM Cortex A8 microprocessor

NAND flash memory

Power management and USB single chip

Photo Credit: IFIXIT

What about autonomous vehicles?

>> 30 second brainstorm <<

Photo Credit: Uber ATG

What about autonomous vehicles?

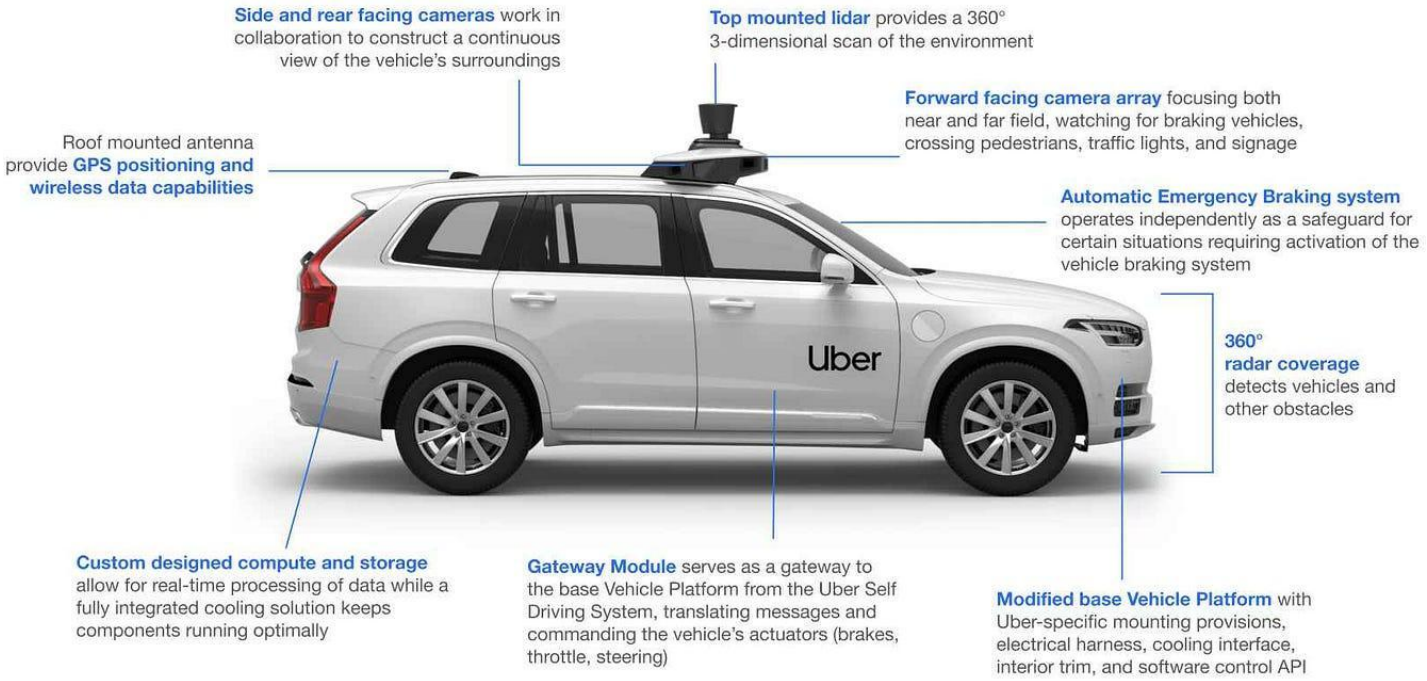


Photo Credit: Uber ATG

What about SD Cards?

>> 30 second brainstorm <<

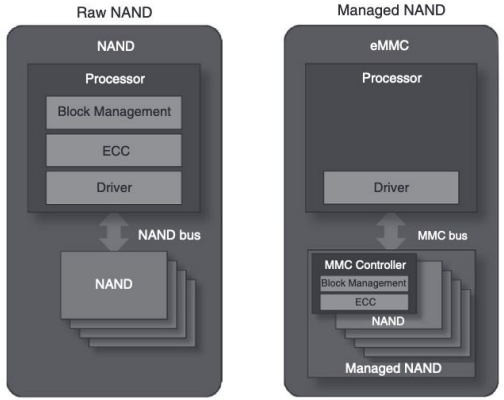
Photo Credit: SanDisk

What about SD Cards?

Flash memory is “unreliable”

- You are not storing data. You are storing a probabilistic approximation of your data
- **Workaround:** error-correcting code (ECC)

Small embedded controller in every “managed flash” device



Define Embedded System

>> 30 second brainstorm <<

Come up with your own definition of embedded systems

Think of computing devices that are …

- **definitely** an embedded system
- on the **broadline**
- **clearly not** an embedded system

Textbook definitions all have their limits…

“a digital system that provides service as part of a larger system”

– G. De Micheli

“any device that includes a programmable computer but is not itself a general purpose computer”

– M. Wolf

”a single-functioned, tightly constrained, reactive computing system”

– F. Vahid

“ a less visible computer”

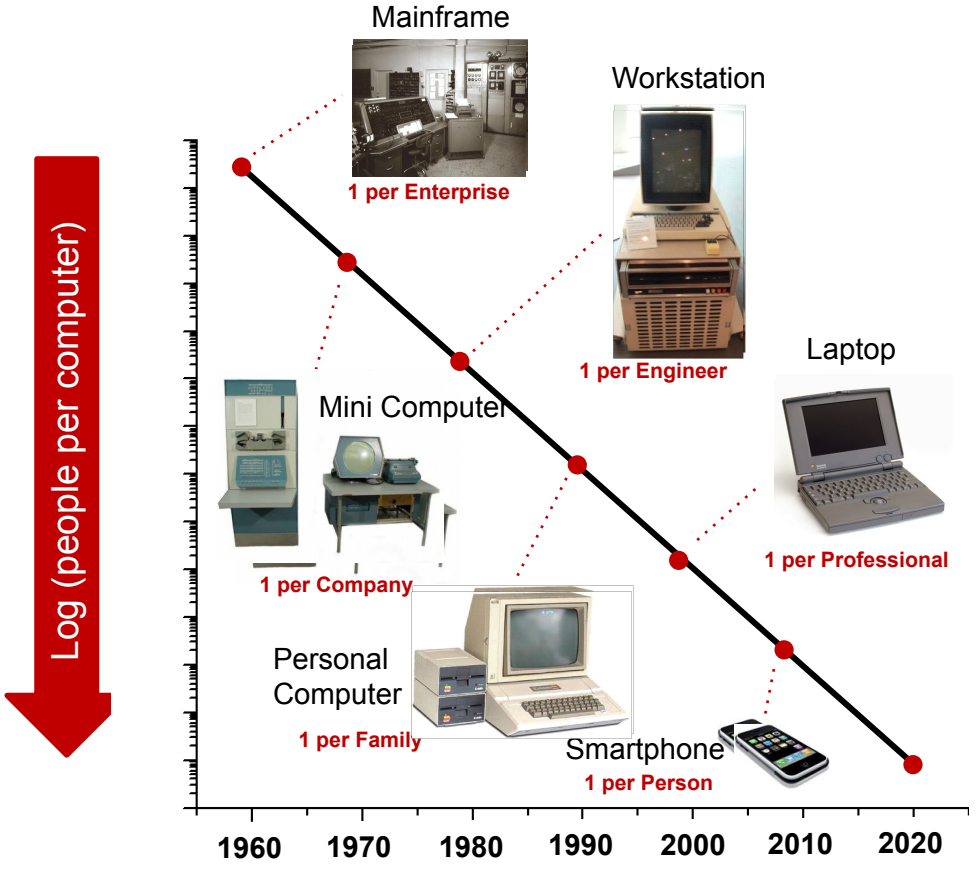
– E. Lee

“a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints”

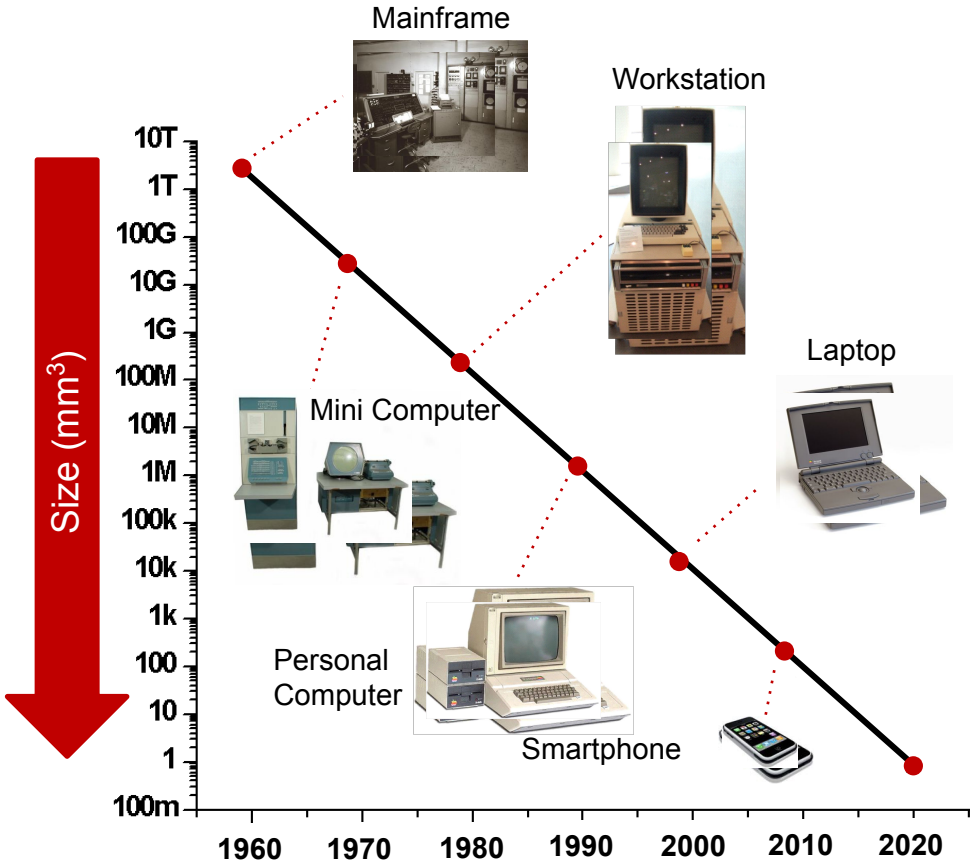
– Wikipedia

So how did we get here?

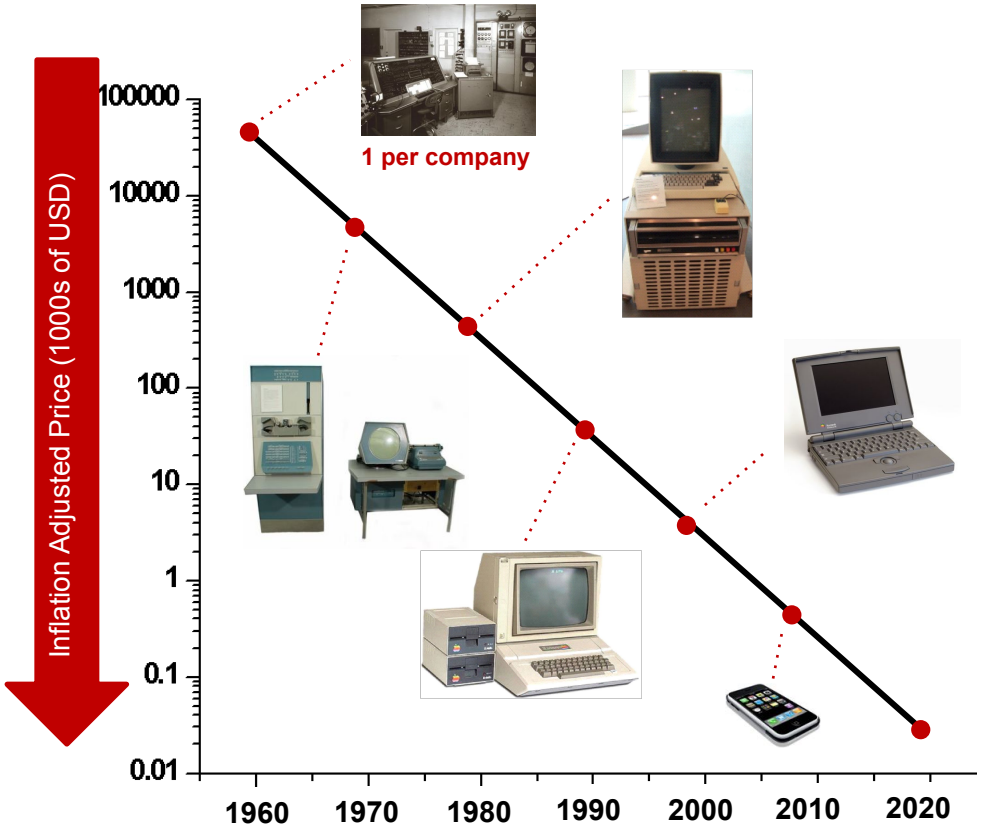
Number of computers per person grows



Computer volume shrinks every decade



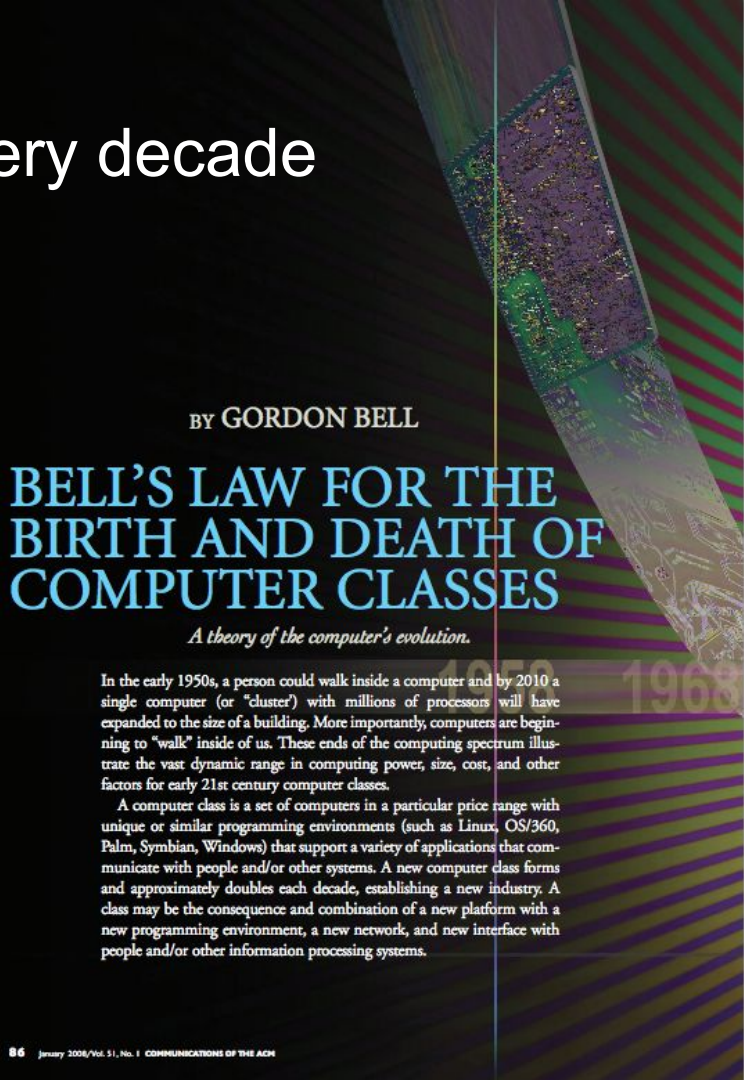
Price falls dramatically



Bell's Law: A new computer class every decade

"Roughly every decade a new, lower priced computer class forms based on a new programming platform, network, and interface resulting in new usage and the establishment of a new industry."

- Gordon Bell [1972,2008]



BY GORDON BELL

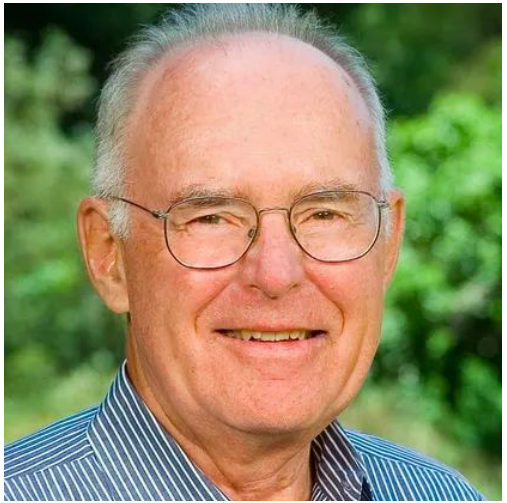
BELL'S LAW FOR THE BIRTH AND DEATH OF COMPUTER CLASSES

A theory of the computer's evolution.

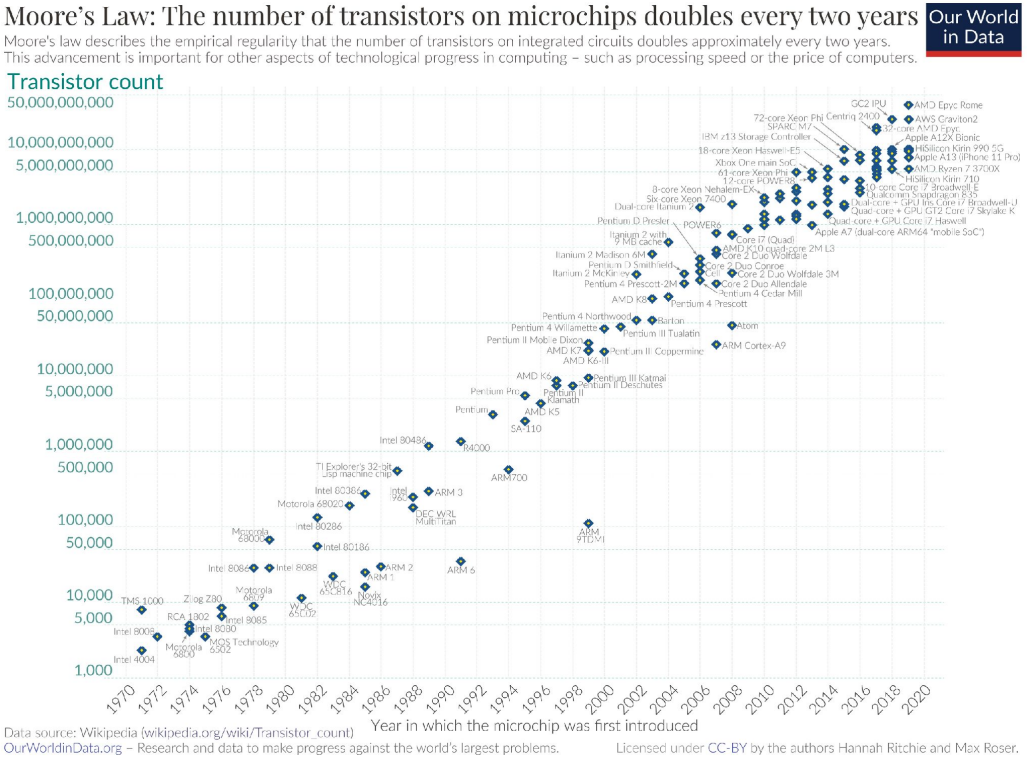
In the early 1950s, a person could walk inside a computer and by 2010 a single computer (or "cluster") with millions of processors will have expanded to the size of a building. More importantly, computers are beginning to "walk" inside of us. These ends of the computing spectrum illustrate the vast dynamic range in computing power, size, cost, and other factors for early 21st century computer classes.

A computer class is a set of computers in a particular price range with unique or similar programming environments (such as Linux, OS/360, Palm, Symbian, Windows) that support a variety of applications that communicate with people and/or other systems. A new computer class forms and approximately doubles each decade, establishing a new industry. A class may be the consequence and combination of a new platform with a new programming environment, a new network, and new interface with people and/or other information processing systems.

Moore’s Law



Gordon Moore

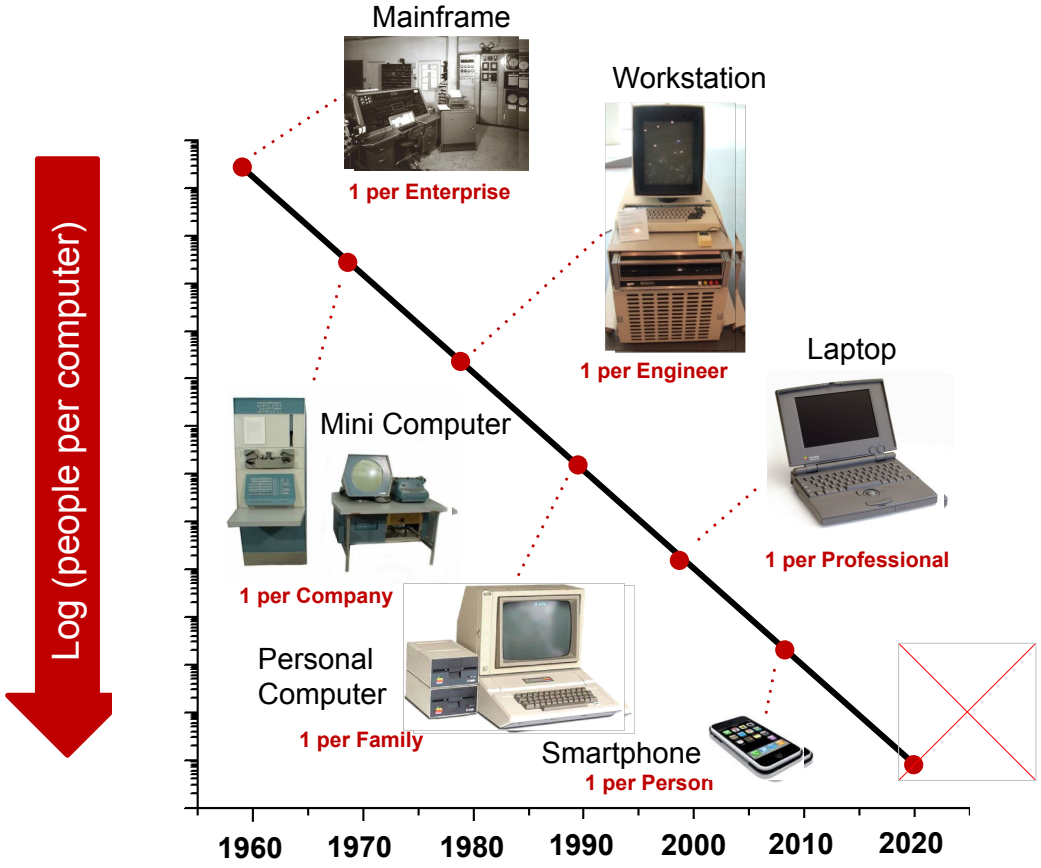


What’s driving Bell’s law?

What is driving Bell's Law?

- Moore's Law
  - Made transistors *cheap*
- Dennard's Scaling
  - Made chips *fast*
  - And *lower voltage*
- Result of Moore's and Dennard?
  - Holding number of transistors constant
    - Exponentially lower cost
    - Exponentially lower power
  - Small, cheap, and low power
    - Microcontrollers
    - Memory
    - Radios

What is the next class of computer?



What is the next class of computer?

- What size is the device?
- How many of these devices are there?
- How much will the device cost?
- What will the device do?
- How will users interact with it?
- How are they controlled?

>> 30 second brainstorm <<

One possible answer….

Mobile, sensing and computing devices
without traditional user interfaces

Key Takeaways

- Technological and market forces are driving the evolution of computing
- By specializing, embedded systems enable new computing devices not previously possible
- There is no precise definition of embedded systems, but…you should be able to identify one when you see it

Computer Architecture Basics

Some background/terminology on computer architecture

A computer combines a processor, random access memory (RAM), read only memory (ROM) and input/output ports.

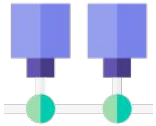
Software is a sequence of specific instructions (stored in memory) that defines exactly when certain tasks are to be performed

The **processor** executes software by retrieving and interpreting instructions one at a time

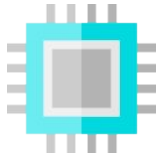
A **microprocessor** is a *small* processor (i.e., fits in your hand) such as the Intel i9 or the AMD Ryzen Threadripper

A **microcomputer** is a *small* computer (i.e., you can carry it) such as a desktop PC or laptop

We are working with microcontrollers in this class

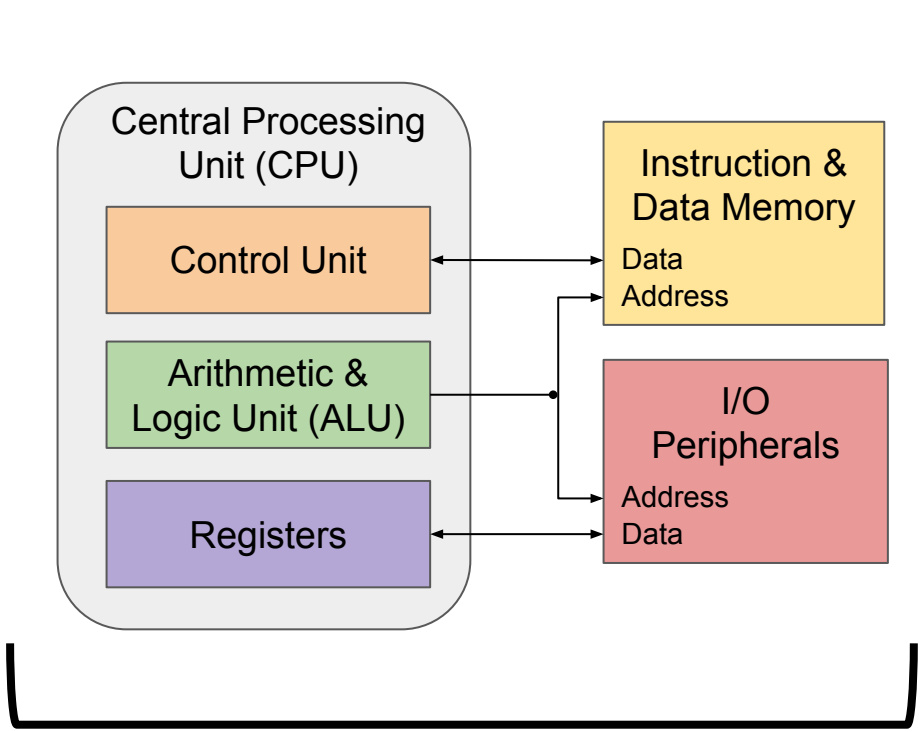


A **port** is a physical connection between the computer and the outside world

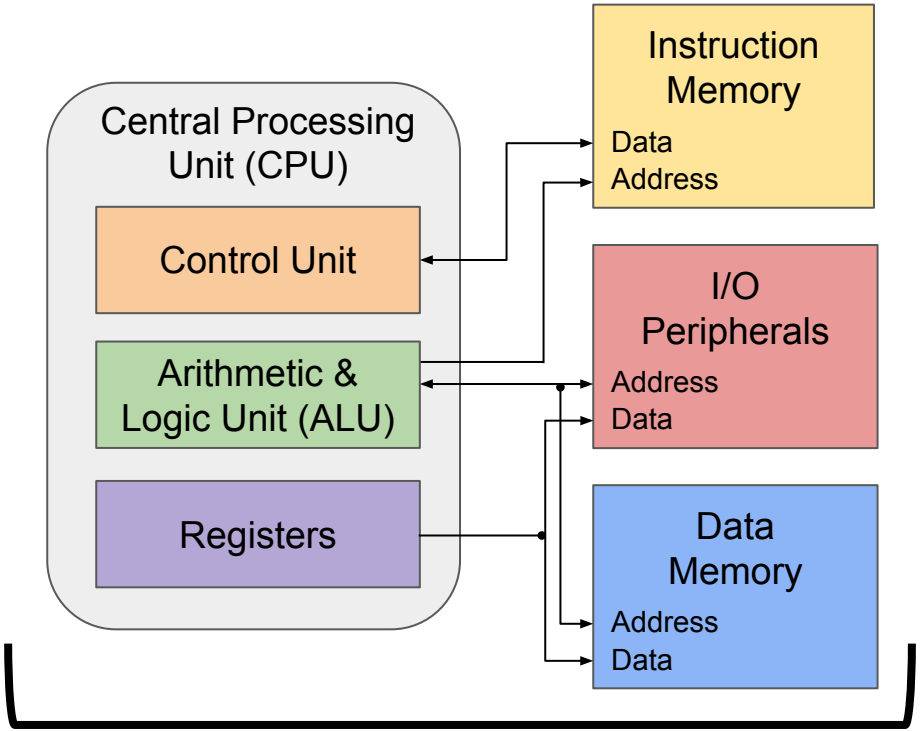


A **microcontroller** is a single chip that contains all of the components of a computer

Some background on computer architecture



Von Neumann Architecture



Harvard Architecture

Von Neumann

- Relatively simple and inexpensive
- Allows only one memory access at any time instant

VS.

Harvard

- Access data memory and instruction memory concurrently
- Tends to be more energy efficient and often offers faster processing speed

Memory

Random Access Memory (RAM) - A computer can store or retrieve information in RAM by either writing to it or reading from it.

RAM is ***volatile*** … what does this mean?

If power is interrupted, all data is lost

Read-Only Memory (ROM) - A type of *non-volatile* memory, where the data stored in the ROM cannot be erased.

Programmable **ROM (PROM)** can be erased and reprogrammed, but erase/programming is significantly slower than writing to RAM.

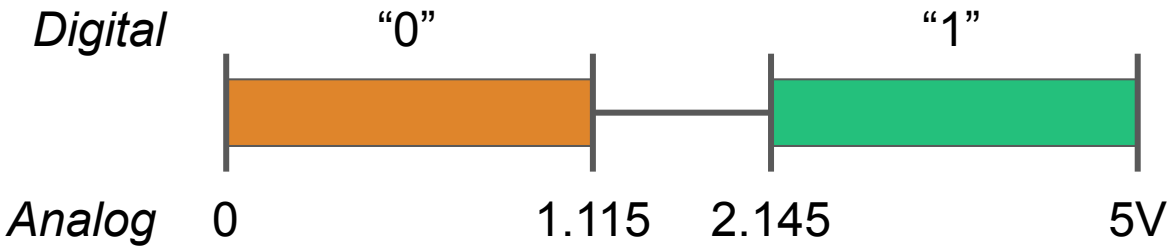
Electrically Erasable **PROM (EEPROM)** are erased and programmed using voltages. Other types of PROMs can be erased with ultraviolet light and programmed with voltages.

How is information stored on a computer?

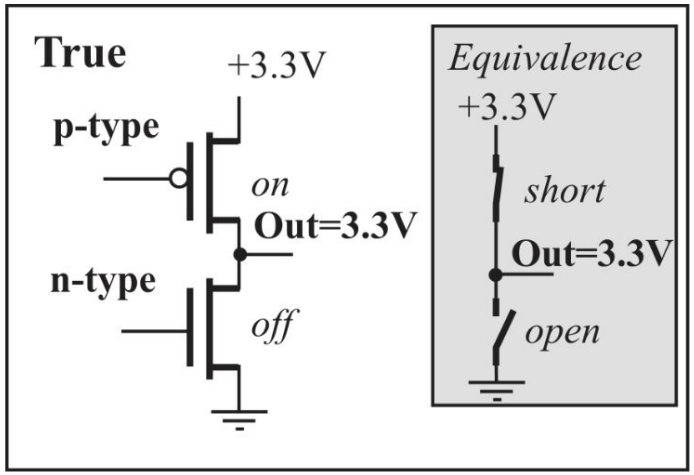
We store information in binary form and a binary bit has two possible states:

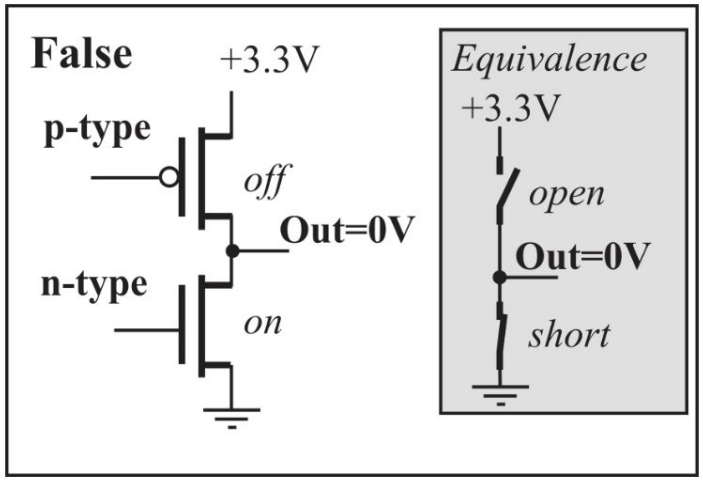
“1” or **“0”** **TRUE** or **FALSE** **LOW** or **HIGH**

These states are represented by **voltages**, where an **analog voltage is mapped to the corresponding digital value**

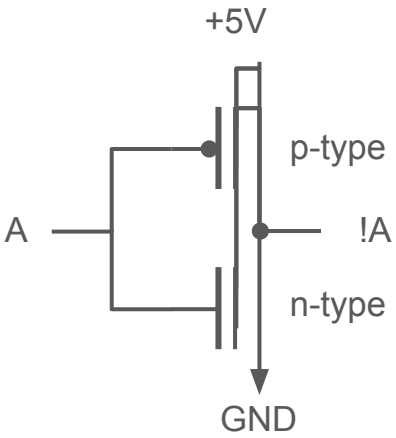


Binary Information Implemented with MOS transistors

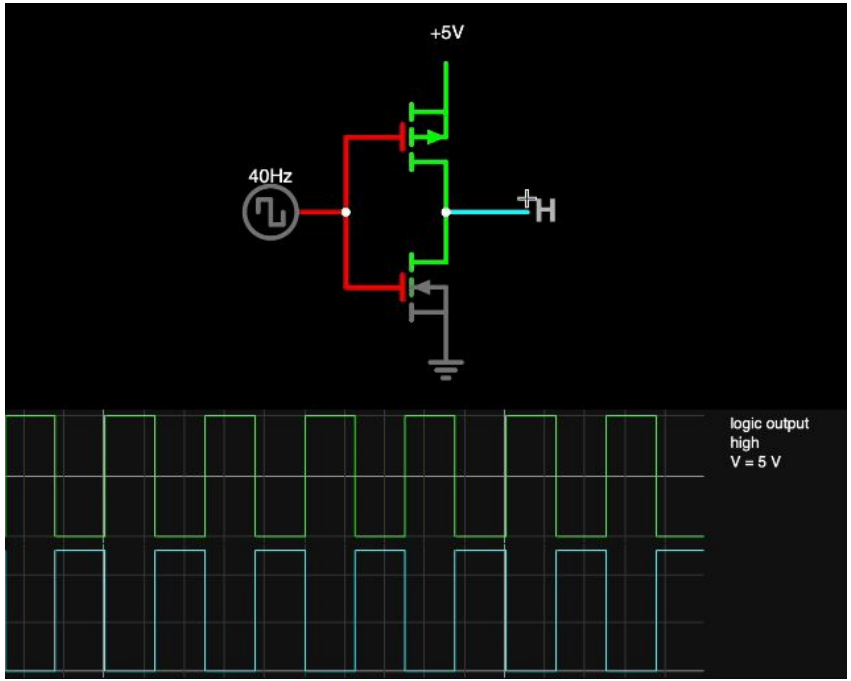




How does this work in hardware?



A	p-type	n-type	!A
0V (LOW)	active	off	5V (HIGH, “1”)
5V (HIGH)	off	active	0V (LOW, “0”)

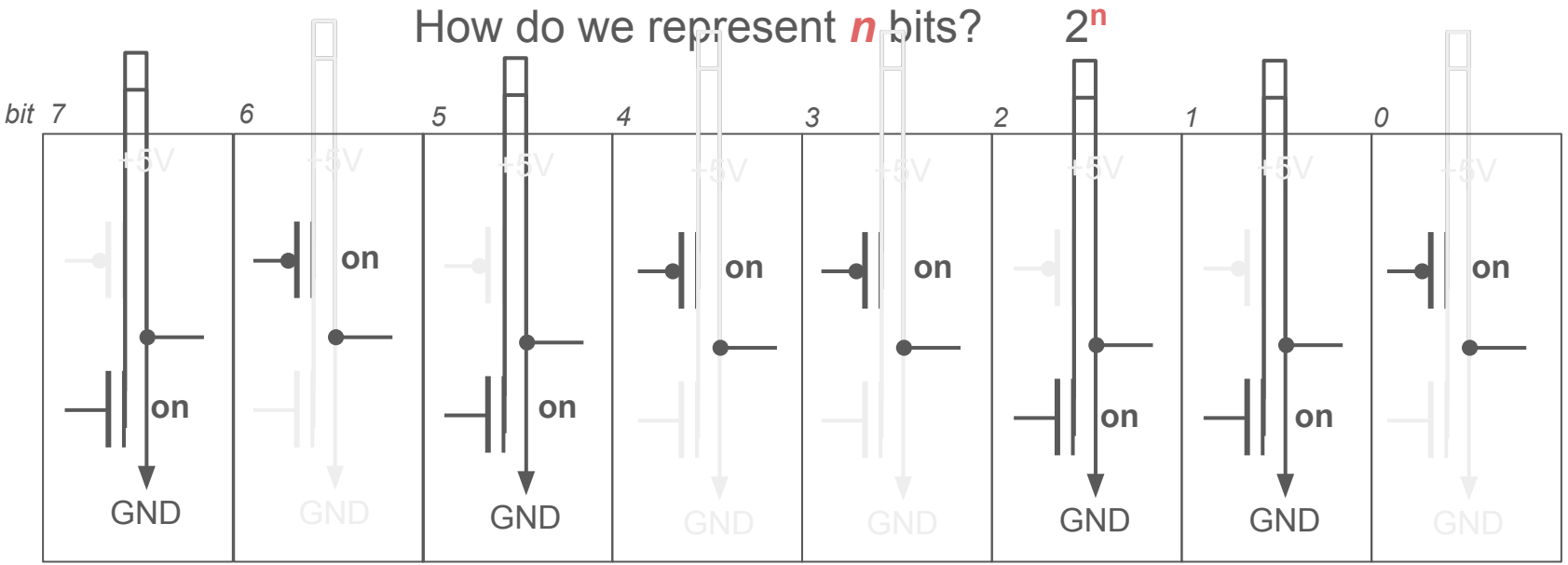


How does this work in hardware?

What if we have information that requires more than 1-bit representation?

2-bits = 4 possible states (00, 01, 10, 11)
3-bits = 8 possible states (000, 001, 010, 011, 100, 101, 111)

How do we represent ***n*** bits? 2^n



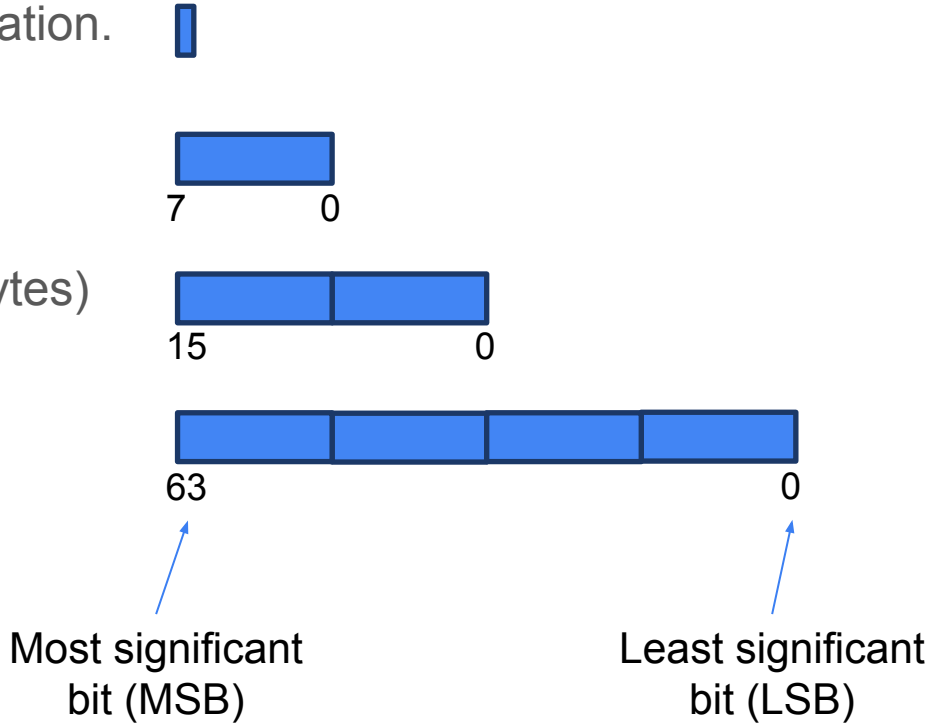
Data Representation Overview

A **bit** is the smallest quantity of information.

A **byte** is a group of 8 bits.

A **halfword** consists of 16 bits (or 2 bytes)

A **word** has 32 bits (or 4 bytes)



Binary, Octal, Decimal, and Hexadecimal Numbers

An ***n-digit*** integer ***N*** in the base ***b*** has the following form

$$a_{n-1}a_{n-2}a_{n-3}\ \cdots a_1a_0$$

Decimal (base 10) $2014_{10} = 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 4 \times 10^0$

We can calculate its equivalent decimal value ***N*** as follows:

$$N = a_{n-1} \times b^{n-1} + a_{n-2} \times b^{n-2} + a_{n-3} \times b^{n-3} + \cdots + a_1 \times \mathrm{b} + \mathrm{a}_0$$

Octal (base 8) $1375_8 = 1 \times 8^3 + 3 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 = 765_{10}$

Hexadecimal Numbers

Hexadecimal is a **base-16** number system (16 possible digits used to represent numbers)

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Example: Convert 0101111010100001

1 Hex Digit = 4 Binary Digits

0101 | 1110 | 1010 | 0001

5 **E** **A** **1**

Decimal	Hex	Binary...	Decimal	Hex	Binary
00	0	0000	08	8	1000
01	1	0001	09	9	1001
02	2	0010	10	A	1010
03	3	0011	11	B	1011
04	4	0100	12	C	1100
05	5	0101	13	D	1101
06	6	0110	14	E	1110
07	7	0111	15	F	1111

Signed Integers

Sign-and-magnitude uses the most significant bit to represent the sign, and the rest of the bits to represent the magnitude

One's Complement denotes a negative number by inverting every bit of its positive equivalent

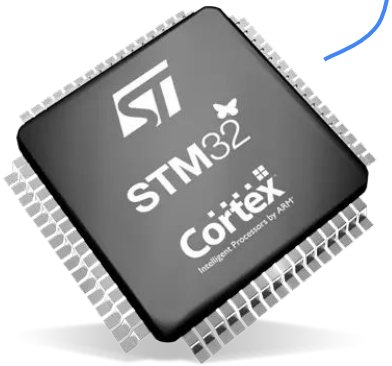
Two's Complement represents a negative number by adding one to the equivalent one's complement

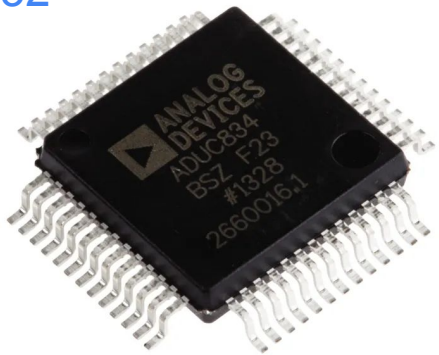
Binary Bit String	Sign-and-Magnitude	One's Complement	Two's Complement
0000	+0	+0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Microcontrollers

- A single chip that contains all of the components of a computer (processor, RAM, ROM, I/O ports)
- Often used in embedded systems because of their low cost, small footprint, and low power requirements.

in this class we will use the STM32





Why ARM Cortex-M?

Many manufacturers ship ARM products …

Frequently used in general embedded systems

Cheap to use

Flexible

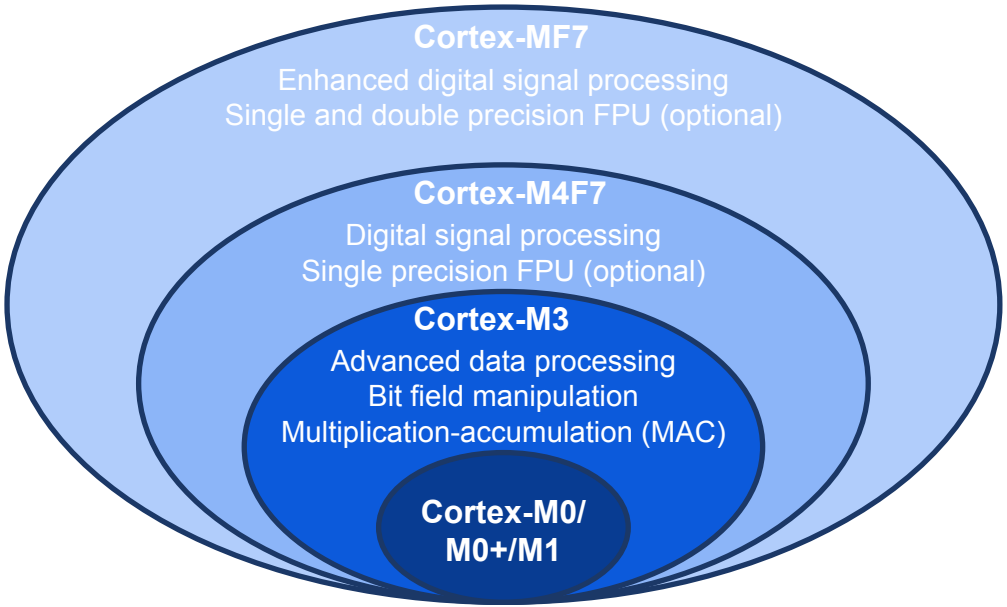
ARM-Cortex Processors

ARM Cortex-A: application processor cores for performance-intensive systems (e.g., smartphones)

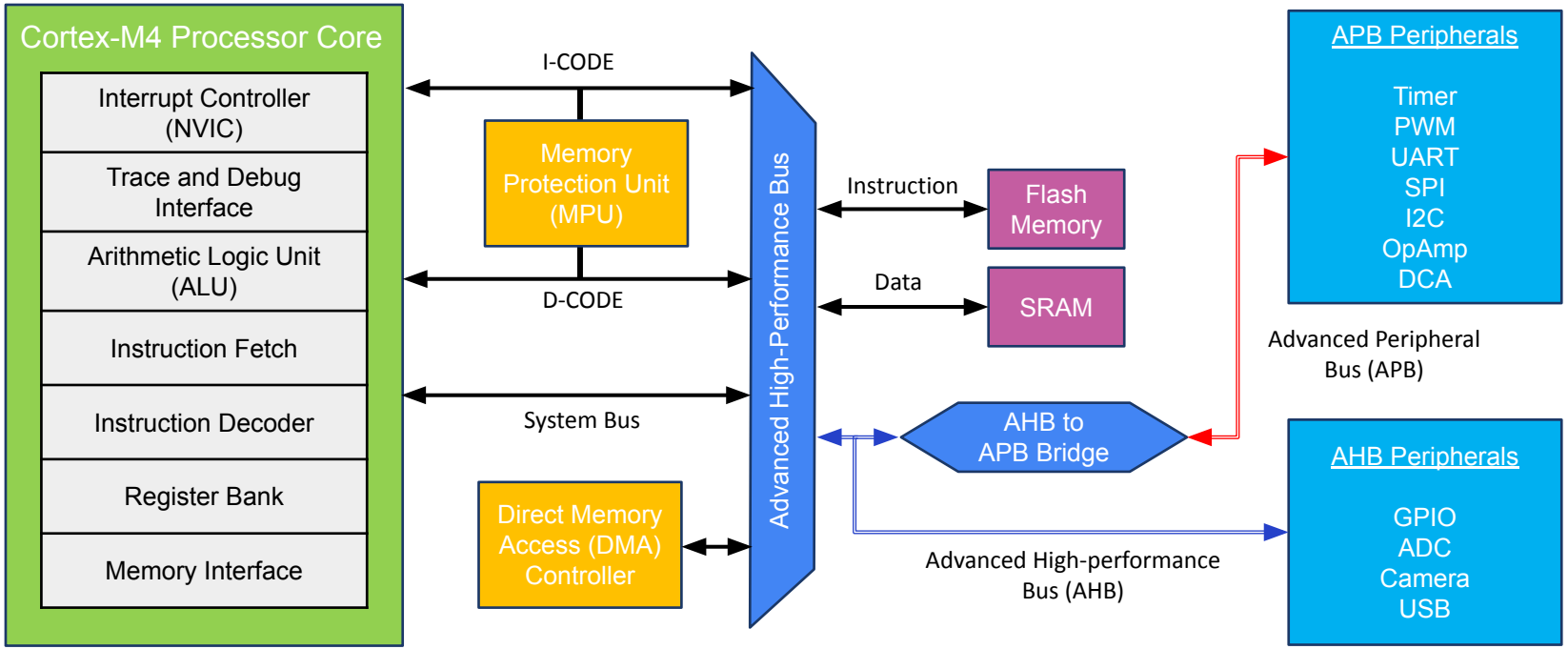
ARM Cortex-R: high-performance cores for real-time applications (e.g., cars)

ARM Cortex-M: microcontroller cores for embedded applications (e.g., IoT devices)

ARM Cortex-M Family



ARM Cortex M4 Microcontroller



Fundamental Components: ARM Cortex M4 Microcontroller

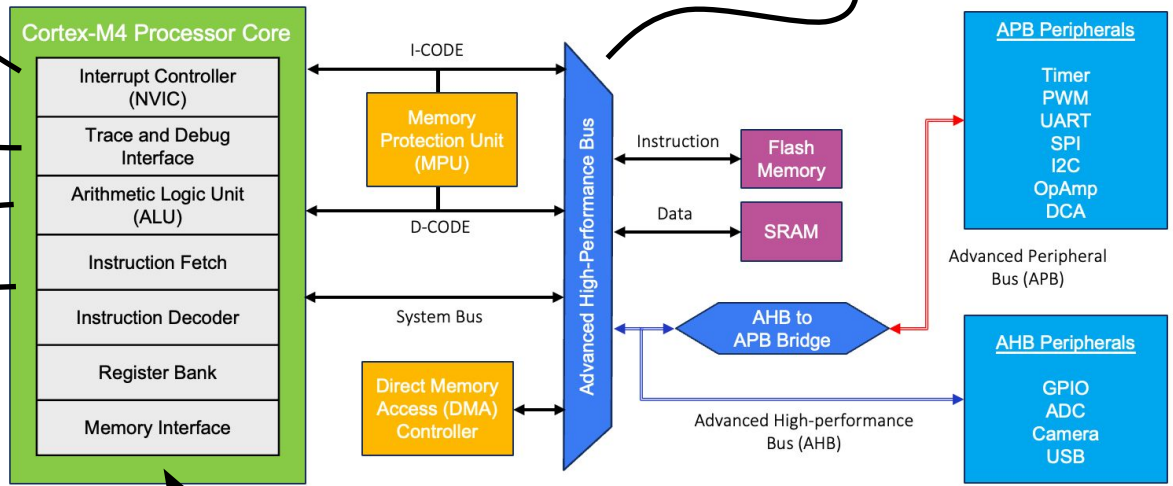
The interrupt controller (NVIC) allows the processor core to stop the execution of the current task and immediately respond to special events

The bus matrix is an interconnection scheme, that allows concurrent data streams between components

Allows the programmer to use a host computer to debug in real-time

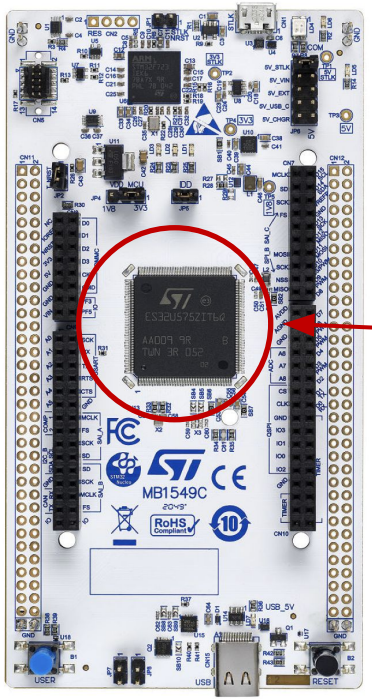
ALU carries out logical (e.g., AND) and arithmetic operations (e.g., add).

Instruction fetching/decoding unit reads and decodes machine instructions

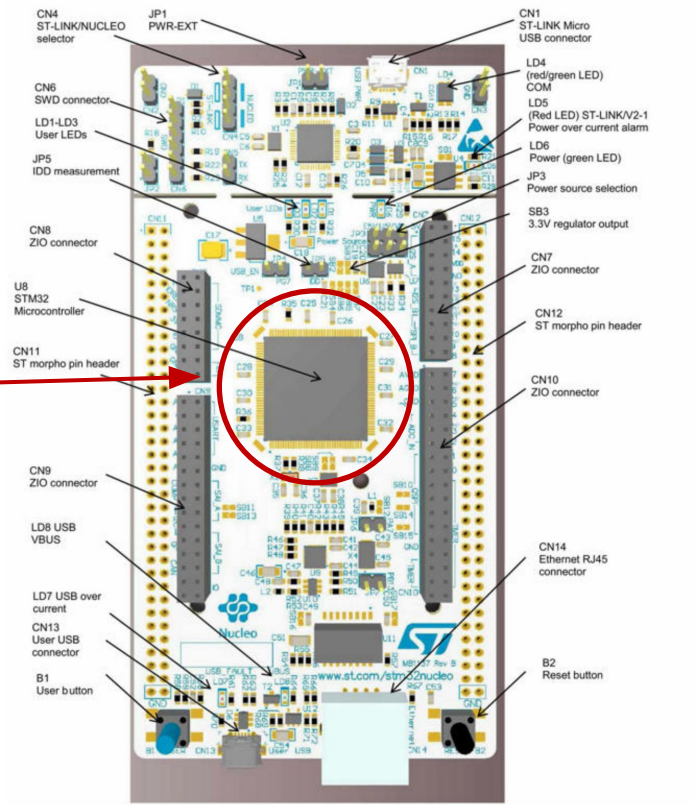


Memory interface supports the access to memory devices (e.g., SRAM, flash)

Your Development Kit



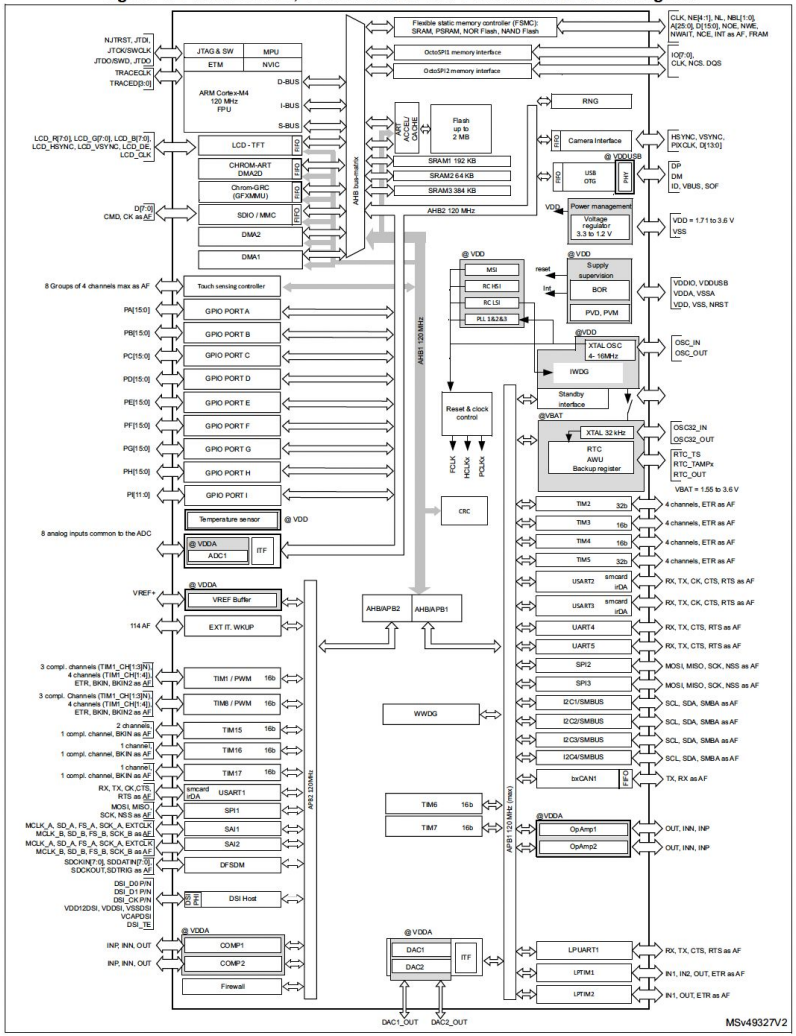
ARM Cortex-M
Microcontroller



STM32L4R5XX Block Diagram…

Don’t panic!

Figure 1. STM32L4R5xx, STM32L4R7xx and STM32L4R9xx block diagram



Zooming in…

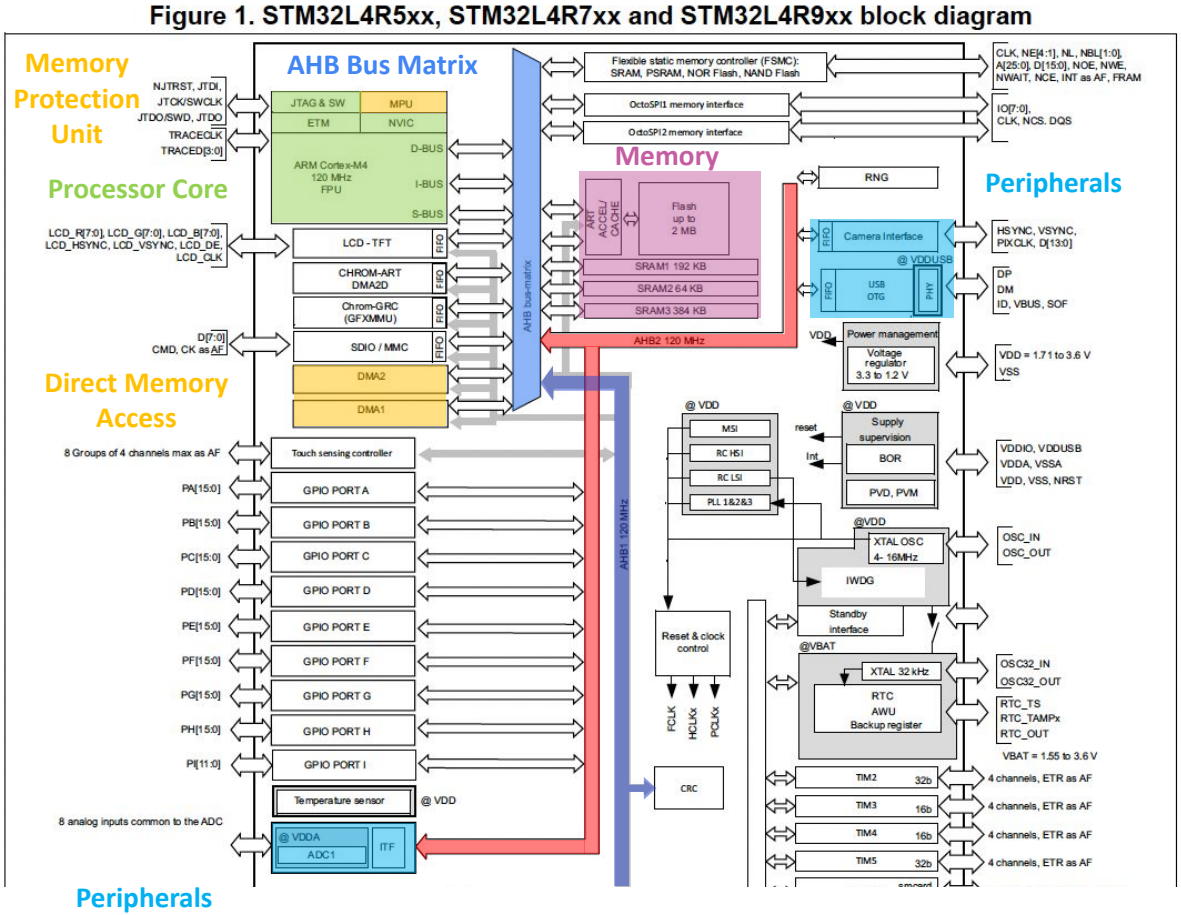
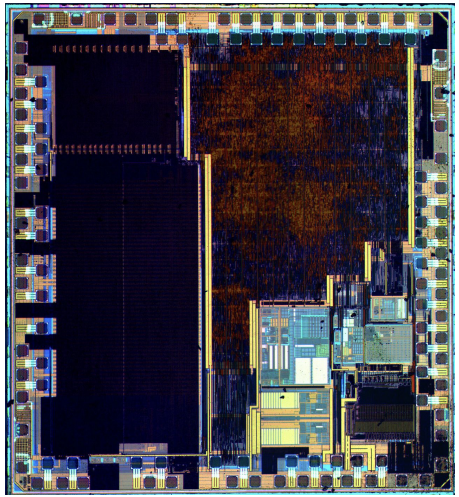


Figure 1. STM32L4R5xx, STM32L4R7xx and STM32L4R9xx block diagram
Memory Protection Unit
Processor Core
Direct Memory Access
Peripherals
AHB Bus Matrix
Memory
Peripherals
NJTRST, JTDI, JTCK/SWCLK JTDO/SWD, JTDO
TRACECLK
TRACED[3:0]
JTAG & SW
MPU
ETM
NVIC
ARM Cortex-M4 120 MHz FPU
D-BUS
I-BUS
S-BUS
AHB bus-matrix
ART ACCEL/ CACHE
Flash up to 2 MB
SRAM1 192 KB
SRAM2 64 KB
SRAM3 384 KB
AHB2 120 MHz
AHB1 120 MHz
Flexible static memory controller (FSMC): SRAM, PSRAM, NOR Flash, NAND Flash
CLK, NE[4:1], NL, NBL[1:0], A[25:0], D[15:0], NOE, NWE, NWAIT, NCE, INT as AF, FRAM
OctoSPI1 memory interface
OctoSPI2 memory interface
IO[7:0], CLK, NCS, DQS
LCD_R[7:0], LCD_G[7:0], LCD_B[7:0], LCD_HSYNC, LCD_VSYNC, LCD_DE, LCD_CLK
LCD - TFT
FIFO
CHROM-ART DMA2D
Chrom-GRC (GFXMMU)
D[7:0] CMD, CK as AF
SDIO / MMC
DMA2
DMA1
RNG
Camera Interface
@ VDDUSB
USB OTG
PHY
HSYNC, VSYNC, PIXCLK, D[13:0]
DP DM ID, VBUS, SOF
Power management
Voltage regulator 3.3 to 1.2 V
VDD
VDD = 1.71 to 3.6 V VSS
8 Groups of 4 channels max as AF
Touch sensing controller
PA[15:0]
GPIO PORT A
PB[15:0]
GPIO PORT B
PC[15:0]
GPIO PORT C
PD[15:0]
GPIO PORT D
PE[15:0]
GPIO PORT E
PF[15:0]
GPIO PORT F
PG[15:0]
GPIO PORT G
PH[15:0]
GPIO PORT H
PI[11:0]
GPIO PORT I
Temperature sensor
@ VDD
8 analog inputs common to the ADC
@ VDDA
ADC1
ITF
@ VDD
MSI
RC HSI
RC LSI
PLL 1&2&3
reset
Int
Supply supervision
BOR
PVD, PVM
VDDIO, VDDUSB VDDA, VSSA VDD, VSS, NRST
@VDD
XTAL OSC 4- 16MHz
IWDG
OSC_IN OSC_OUT
Standby interface
Reset & clock control
FCLK
HCLKx
PCLKx
CRC
@VBAT
XTAL 32 kHz
RTC AWU Backup register
OSC32_IN OSC32_OUT
RTC_TS RTC_TAMPx RTC_OUT
VBAT = 1.55 to 3.6 V
TIM2 32b 4 channels, ETR as AF
TIM3 16b 4 channels, ETR as AF
TIM4 16b 4 channels, ETR as AF
TIM5 32b 4 channels, ETR as AF

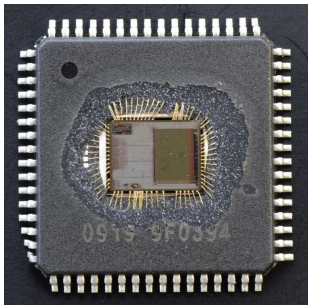
GPIO/Register/MUX Primer

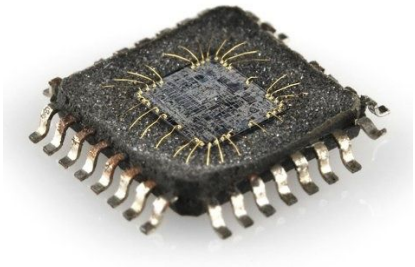
General Purpose Input Output (GPIO)

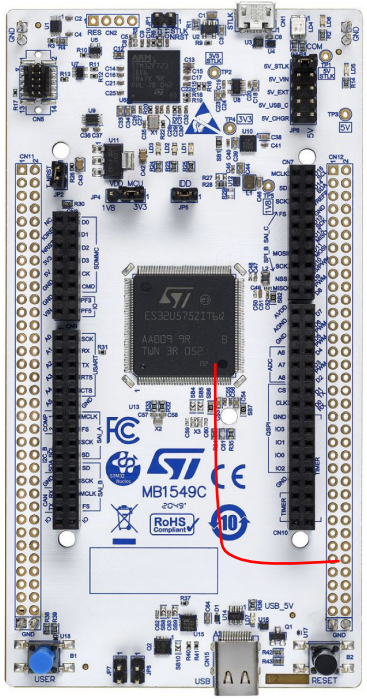
Get signals on and off chip



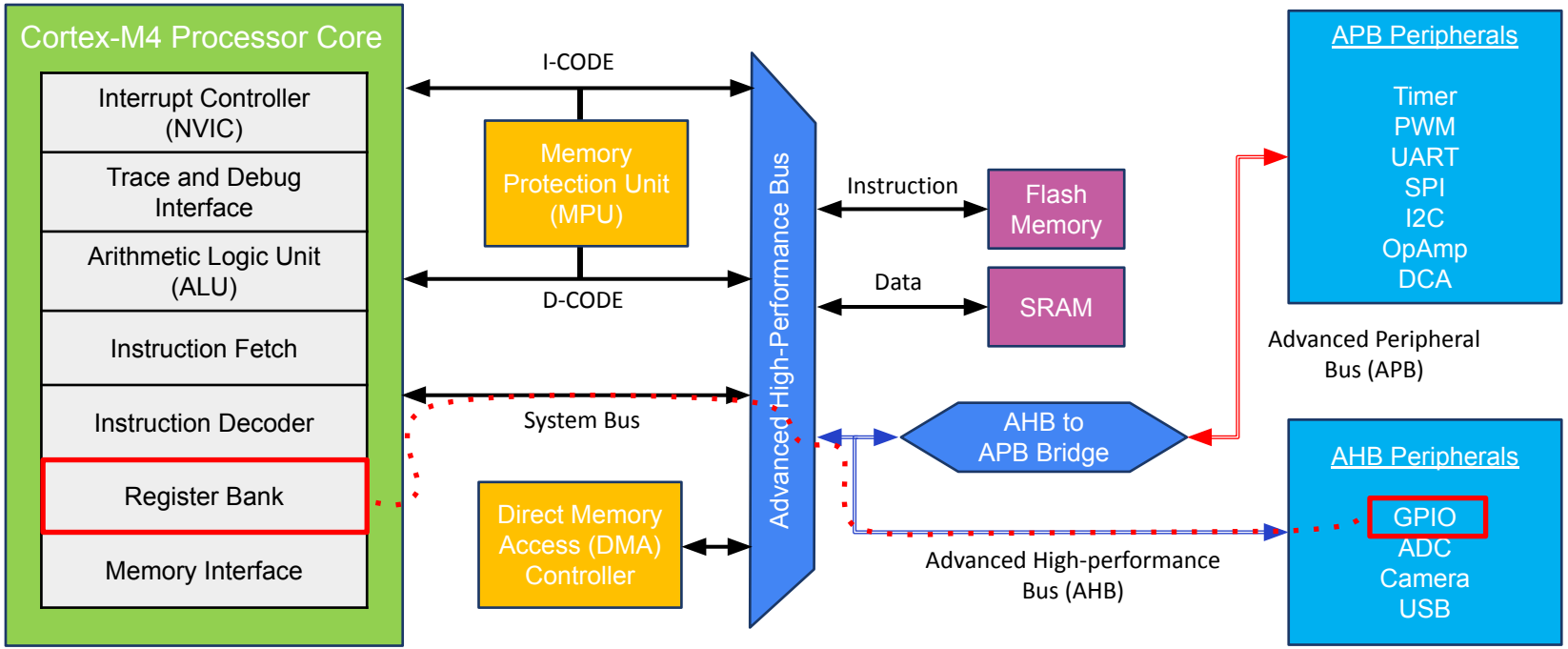
STM32 Integrated Circuit





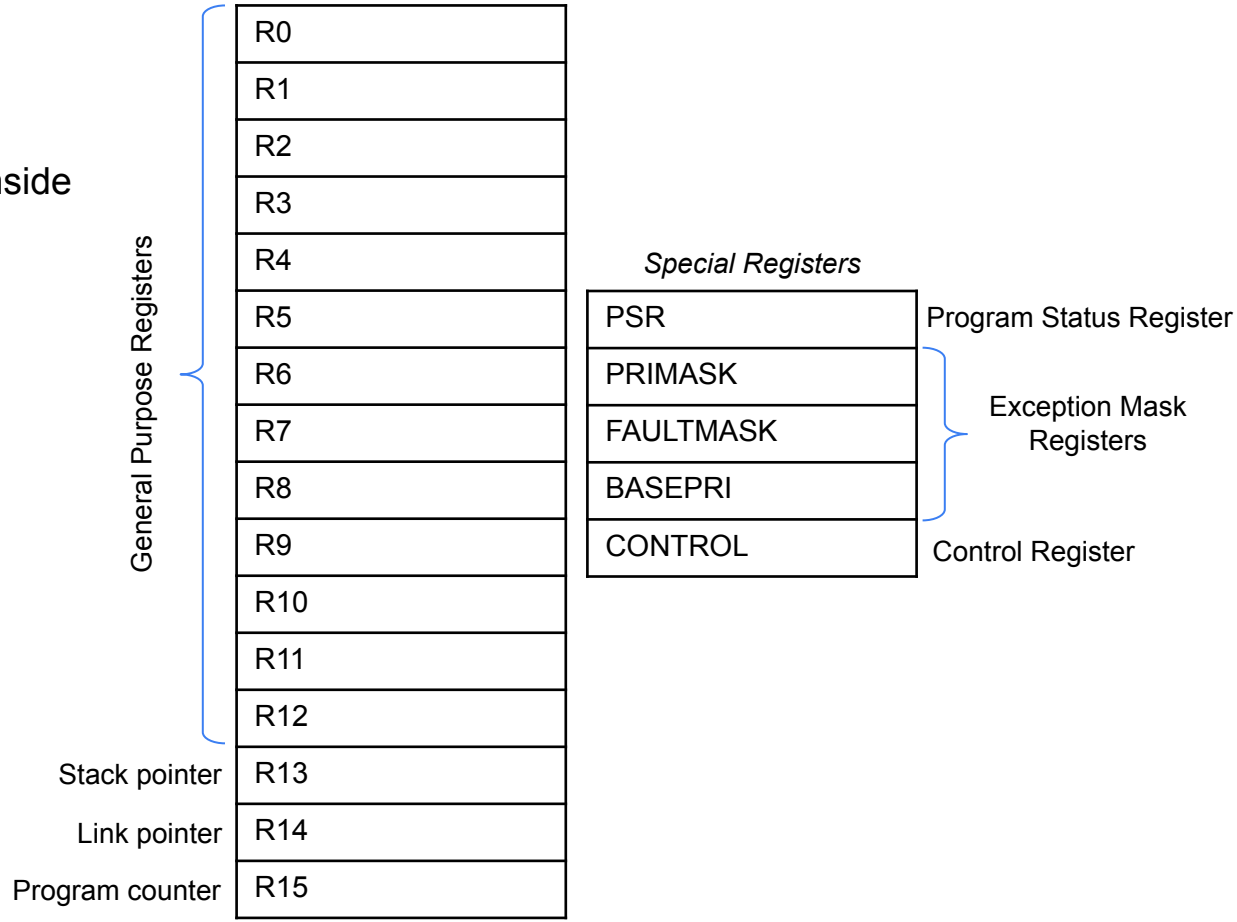


ARM Cortex M4 Microcontroller



What is a register?

Registers are high speed storage inside the processor

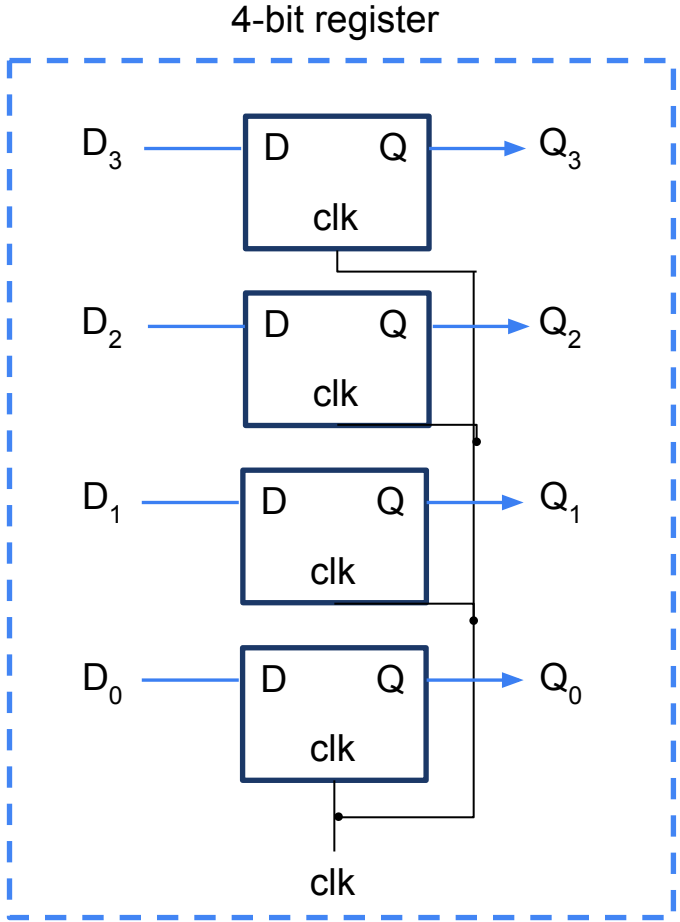


How does a register work?

- Storage unit. Can hold an ***n–bit value***
- Composed of a group of ***n flip-flops***

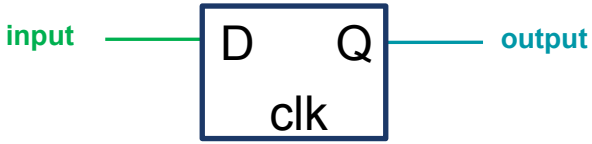
What is a D flip-flop (DFF)?

A digital circuit used for storing a single bit of data

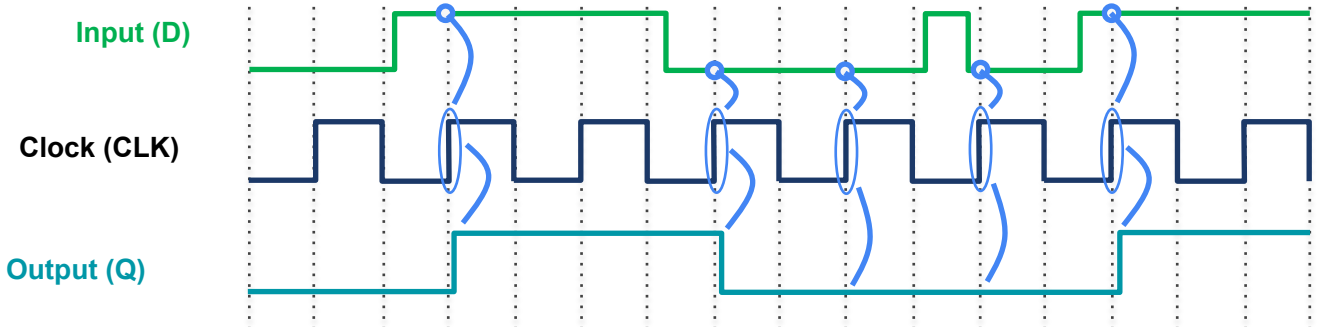


What is a D flip-flop (DFF)?

A digital circuit used for storing a single bit of data

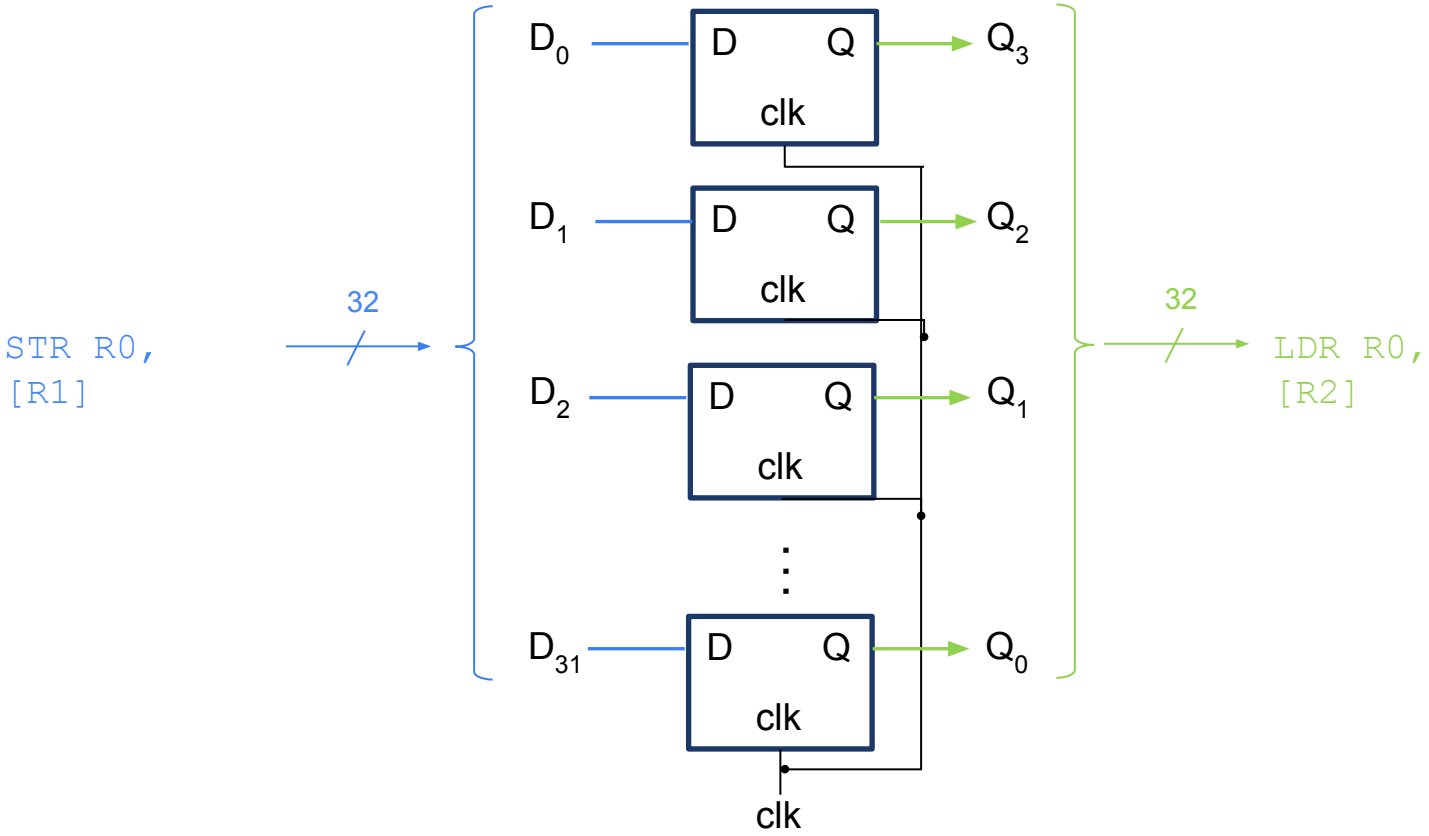


DFF Symbol

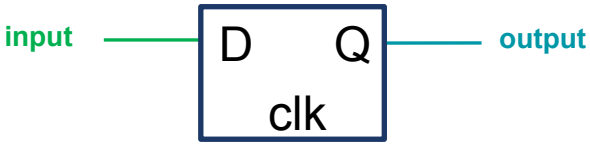


In this example, data is clocked in on the positive edge of the clock

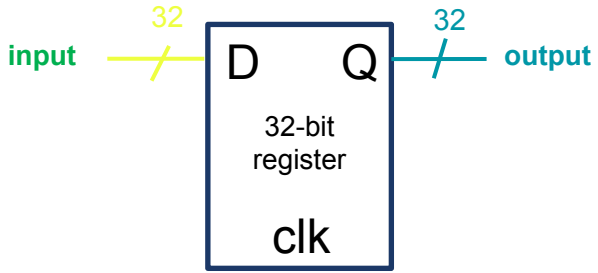
Load and **Store** data into **Registers**



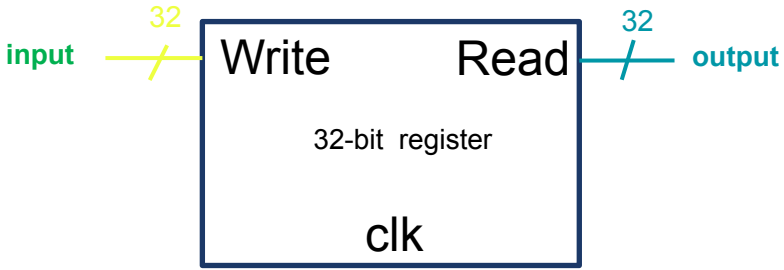
DFF and Register Symbols



DFF Symbol

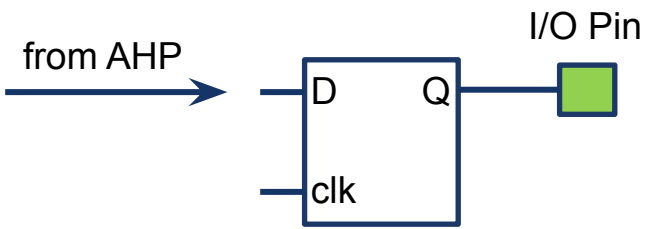


Another way to think about it

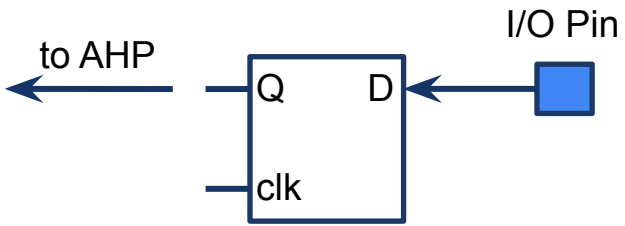


GPIO Pin Topology

Simplified **output** pin

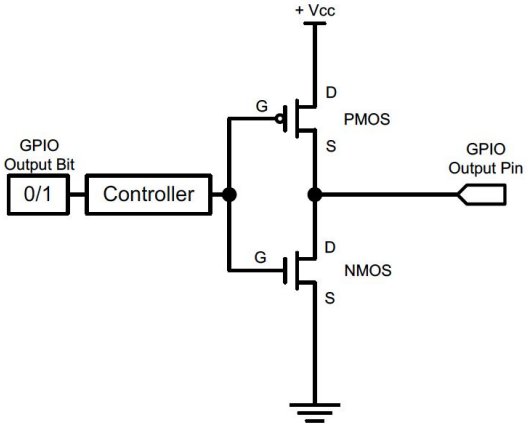


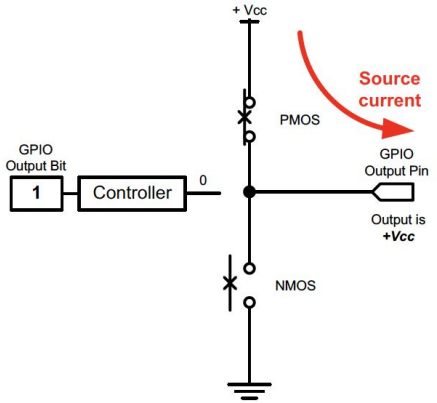
Simplified **input** pin



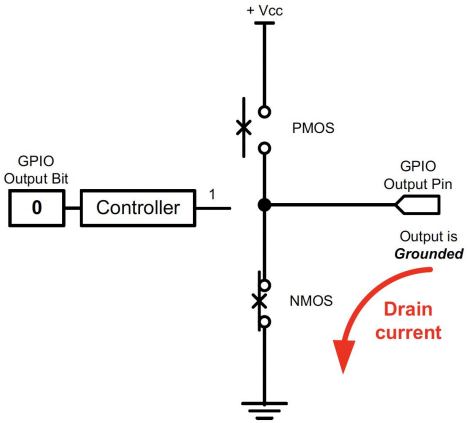
flipped pins

GPIO Output: Push-pull



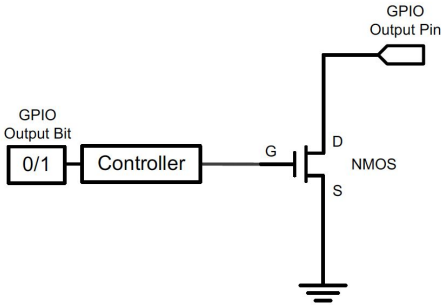


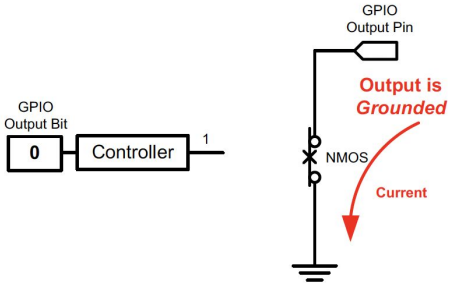
GPIO Output = 1
Source current to external circuit



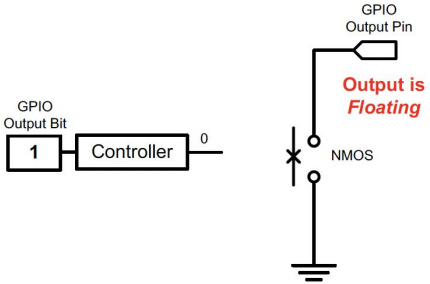
GPIO Output = 0
Drain current from external circuit

GPIO: Open Drain



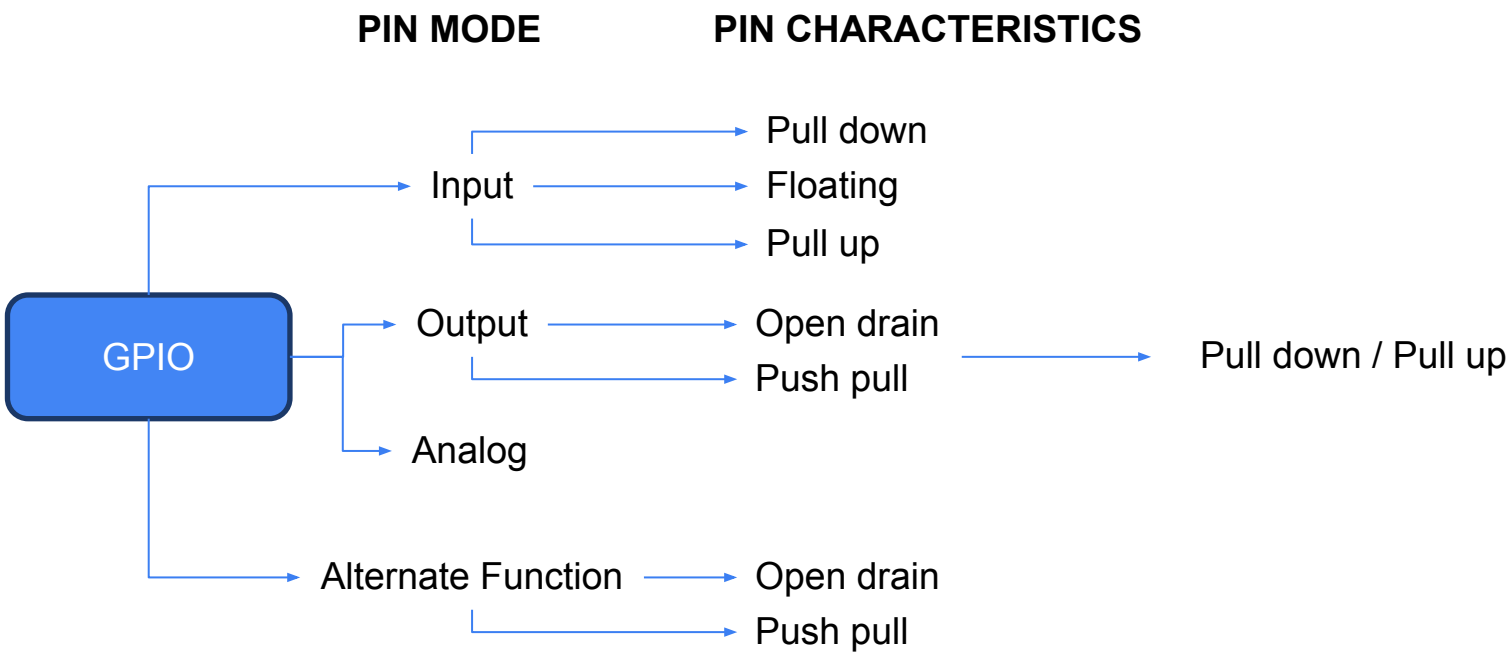


GPIO Output =0
Drain current from external circuit



GPIO Output = 1
GPIO Pin has high-impedance to external circuit

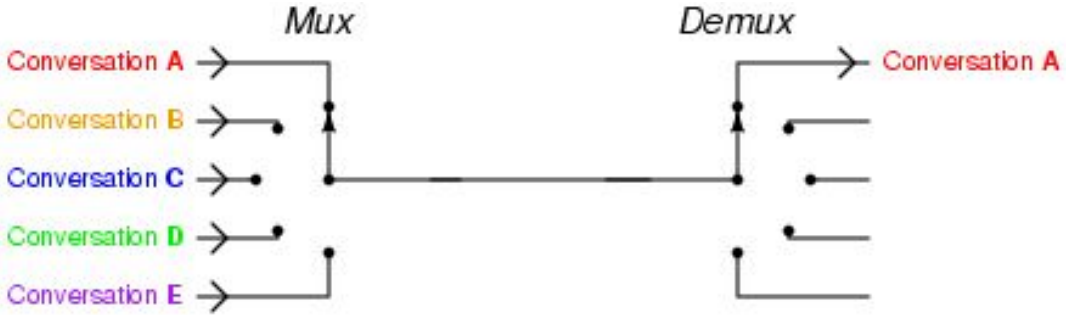
GPIO Configurations



What is a MUX?

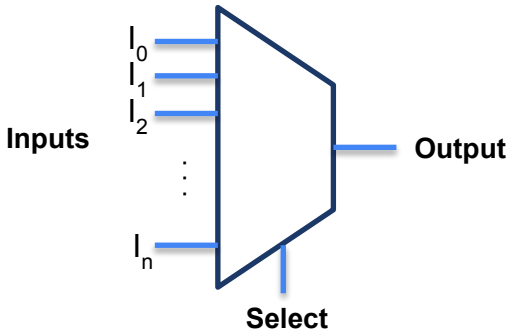
A device that **selects between multiple input signals** and forwards the selected input line to a single output

We see them in the STM32L4 manual (e.g., clock tree)!



Source: Wikipedia

MUX Symbol



EE40M

EE40M is a prerequisite to EE186. Review slides from the course if you need a refresher on:

- Number systems
- Boolean logic
- Logic Gates
- Basics of resistors, capacitors, diodes, inductors
- Basics of amplifiers
- Basics of filters
- Solving circuits

LINK:

https://web.stanford.edu/class/archive/engr/engr40m.1178/slides.html

Recap

- General course overview
  - See the course website for more details
- Overview of embedded systems
- High-level overview of ARM Cortex-M architecture
- High-level introduction to GPIO pins

Acknowledgements

- These lecture slides were adapted from the University of Michigan's EECS 373 course (many thanks to Alanson Sample!)

- These lecture slides also leverage materials from the following books:
  - Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C
    - Yifeng Zhu
  - The Definitive Guide to ARM Cortex – M3 and Cortex – M4 Processors
    - Joseph Yiu
  - Introduction to ARM Cortex – M Microcontrollers, Embedded Systems
    - Jonathan W. Valvano