

Networked Embedded Systems

Week 1: Introduction to Networked Embedded Systems

Xenofon (Fontas) Fafoutis

Professor

xeafa@dtu.dk

www.compute.dtu.dk/~xeafa

Embedded Systems

- What is an Embedded System?

Embedded Systems

- A computer in a device whose principal mission is not computation
- A special-purpose computing system
 - Has a dedicated function within a larger system or product
 - Typically measures and controls physical operations
- As opposed to general-purpose computers
 - Laptops, Desktops, Servers
- Embedded systems are everywhere
 - In 2009, it was estimated that more than 98% of produced processors are used in embedded systems
 - Examples: washing machines, microwave ovens, calculators, digital watches, cruise missiles, GPS receivers, heart monitors, laser printers, engine controllers, digital cameras, traffic lights, remote controls, bread machines, fax machines, pagers, cash registers, treadmills, gas pumps, credit/debit card readers, thermostats, pacemakers



Image source: Wikipedia

Washing Machine as an Embedded System

- What are the different computer-controlled functions of a washing machine?



Image source: Wikipedia

Washing Machine as an Embedded System

- Input/Output
 - User selects programme, temperature, rinsing speed, duration, etc
 - User preferences are captured via buttons, switches, touch screens
 - Status of washing cycle, errors are shown in a screen or using LEDs
- Control functions (sensing and actuation)
 - Control valves for getting water in and out of tank
 - Control heater for heating the water in the correct temperature
 - Control power that goes to the rotation motor to get the right RPM
- Safety functions (errors and warnings)
 - Check the door is closed before the programme can start
 - Check the weight of the cloths
 - Check that the right amount of water is in before it starts heating it
- Time keeping
 - Keep the time passed between washing stages
 - Delayed start functionality



Image source: Wikipedia

Connecting the Digital to the Physical World

- Embedded Systems are close to the physical world
 - They employ sensors to measure/perceive the physical world
 - They employ actuators to alter/manipulate the physical world
- Sensors
 - Cameras, microphones, thermometers, humidity sensors, pressure sensors, light sensors, radars, lidars, motion sensors
- Actuators and other output devices
 - Screens, speakers, electric motors, fluid motors, switches, valves, heaters, vibration motors, lights

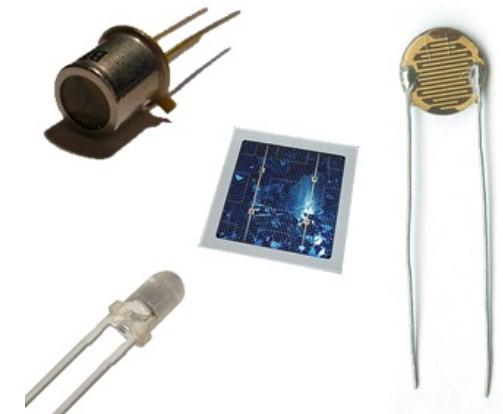
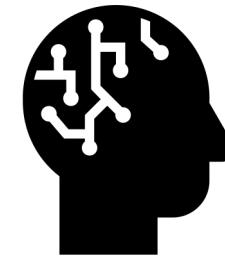
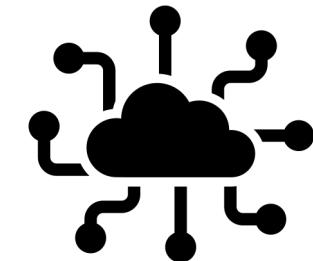


Image source: Wikipedia

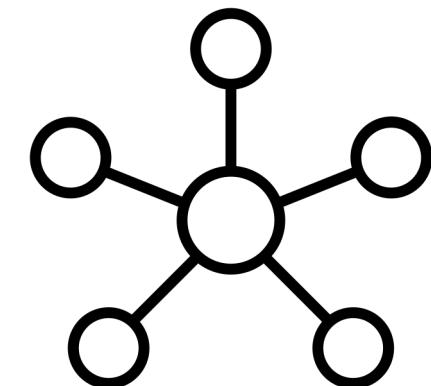
Two big trends in Embedded Systems

- **Connectivity**
 - Embedded Systems send/receive data to/from other Embedded Systems or larger systems
- **Intelligence**
 - Embedded Systems provide/consume data as part of Artificial Intelligence (AI) frameworks that extract knowledge and perform reasoning



Networked Embedded Systems

- Originally, Embedded Systems operated in isolation as part of a single machine or system
 - E.g. a thermostat inside a boiler measures the temperature and controls the power that goes to heater to reach the desired temperature
- Networked Embedded Systems
 - A group of embedded systems connected over a network and cooperate towards one goal or application
 - Also known as Distributed Embedded Systems
- The first Networked Embedded Systems were connected over wired networks



CAN Bus

- A wired networking solution designed for vehicles
- Originally developed by Bosch starting in 1983
- Officially, released in 1986
- Mercedes-Benz first introduced in the production vehicles in 1991
- Standardised by ISO in 1993

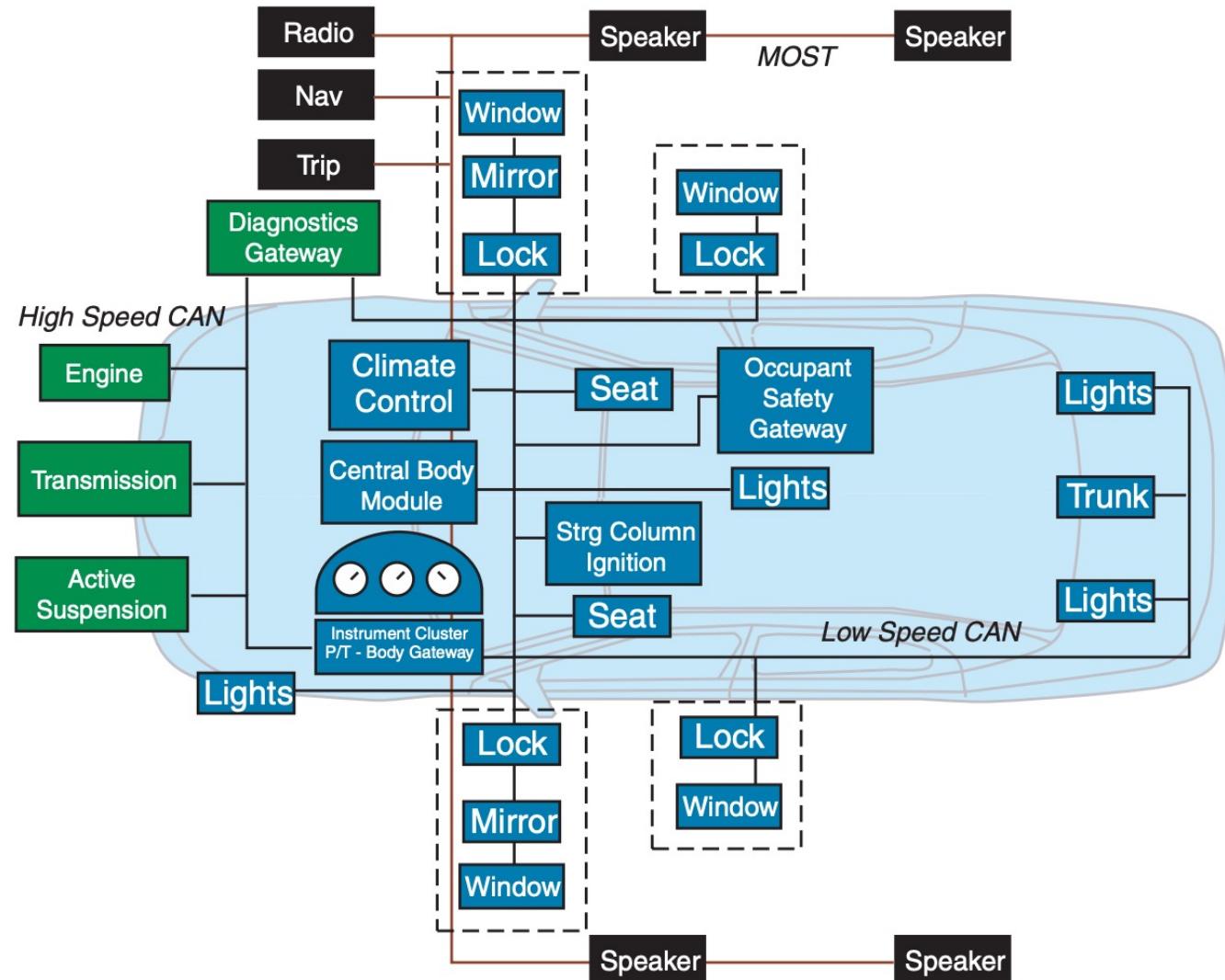
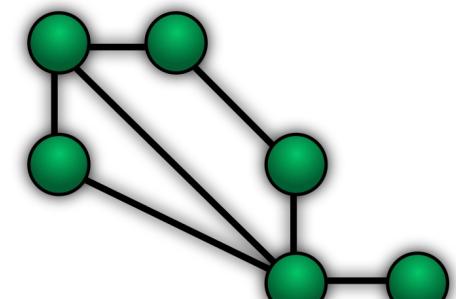


Image source: https://www.eecs.umich.edu/courses/eecs461/doc/CAN_notes.pdf

From Wired to Wireless Embedded Systems

- Wireless Networks are flexible and not require an infrastructure
 - An apartment can have wires embedded in the walls when built/renovated
 - But wireless networking makes it easier when this infrastructure does not exist
- Wireless Networks provide an easy way to extend the area of coverage
 - From star networks to multi-hop networks
- Wireless Networks enable mobility
 - E.g., wearable embedded systems, portable embedded systems
- Wireless Embedded Systems form:
 - Wireless Sensor Networks
 - Wireless Sensor and Actuator Networks



Mesh

Image source: Wikipedia

Wireless Embedded Systems in Agriculture

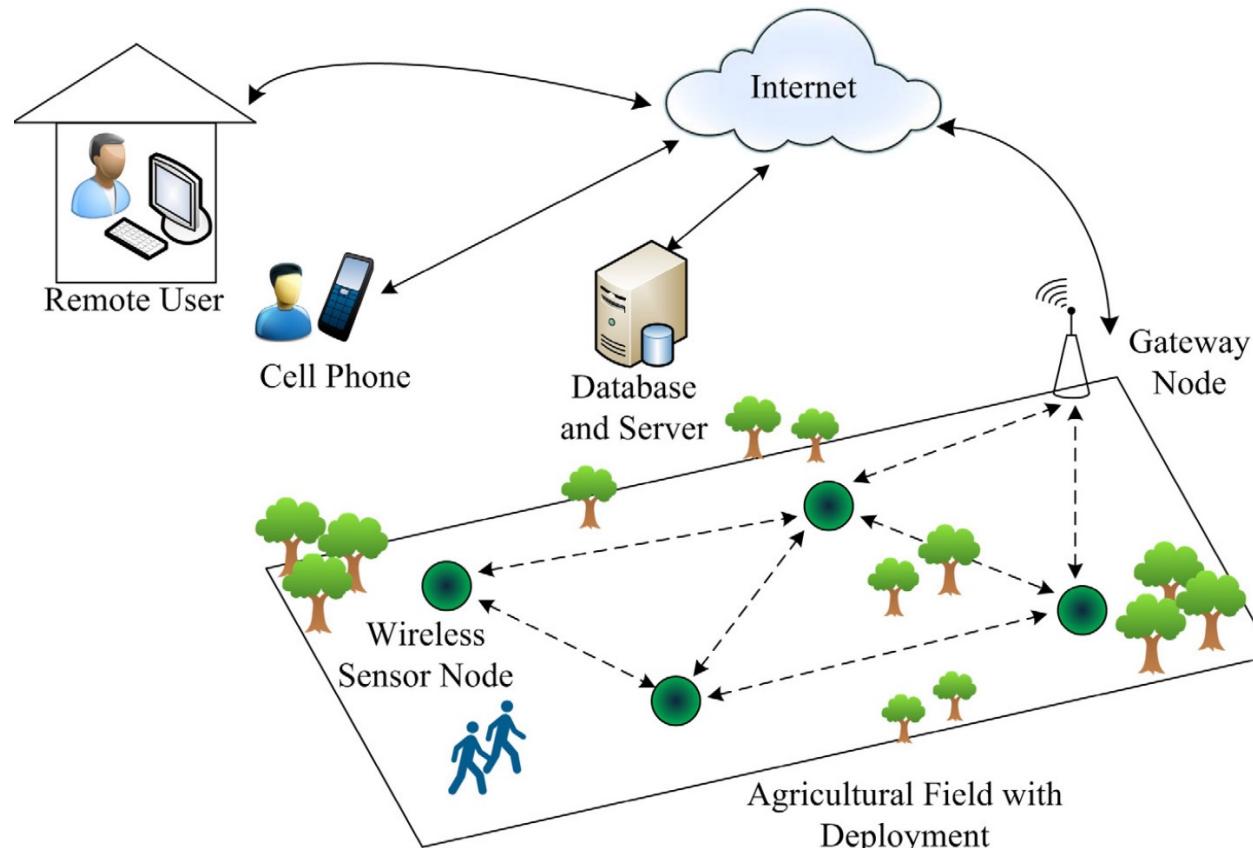


Image source: <https://doi.org/10.1016/j.compag.2015.08.011>

The Internet of Things

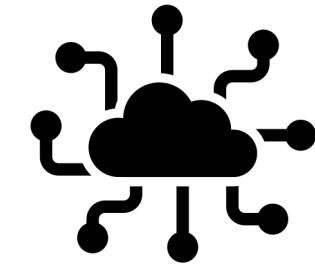
- The Internet of Things connects isolated networks of Embedded Systems in a “global” network
- Several application domains:
 - Smart Cities, transportation networks, logistics
 - Environmental monitoring, wildlife monitoring
 - Disaster response, earthquakes, wildfires
 - Industrial networks, manufacturing, factories
 - Agriculture, maritime
 - Energy management, smart grid
 - Health systems, personal health, fitness
 - Consumer electronics, entertainment
 - Building automation, smart homes
 - Military



Image source: Wikipedia

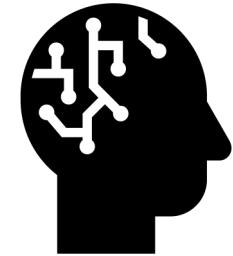
The Connectivity Trend

- Embedded Systems experience an increasing trend for connectivity and collaboration
- From isolated Embedded Systems...
 - To networks of Embedded Systems within the same machine...
 - To Networked Embedded Systems forming wired networks...
 - To Wireless Embedded Systems covering wider areas...
 - To the potentially global and beyond (e.g. satellites) connectivity of the Internet of Things



Intelligent Embedded Systems

- Artificial Intelligence
 - The process of finding patterns and extracting knowledge out of raw data
 - Using the data/knowledge to perform reasoning, decision making, planning, etc
- Cognitive Systems
 - Systems that imitate the human brain



"Cognition encompasses all aspects of intellectual functions and processes such as: perception, attention, thought, intelligence, the formation of knowledge, memory and working memory, judgment and evaluation, reasoning and computation, problem solving and decision making, comprehension and production of language." Wikipedia

- Autonomous Systems
 - Systems that operate with little or no human intervention

Control Systems

- Regulate a variable using sensing, actuation and a control algorithm
- The classic Embedded System application
- Involves perception (sensing) and decision making (actuation)

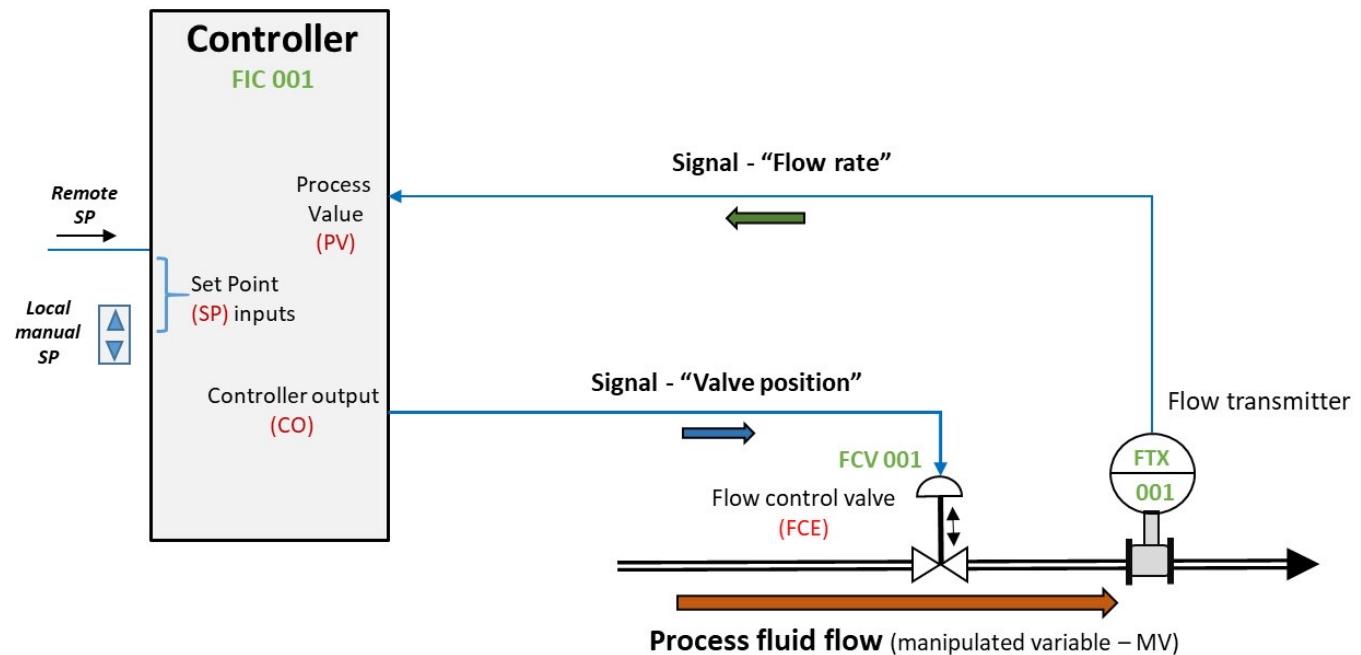
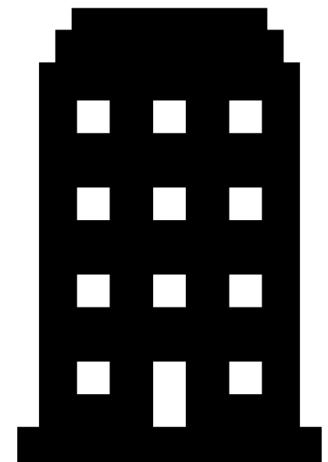


Image source: Wikipedia

Rule-Based Machine Intelligence

- Deductive Reasoning
 - Derive conclusions from rules and axioms
 - E.g. "All humans are mortal", "Alice is human", therefore "Alice is mortal"
- Can be formulated as "if-then" statements
 - E.g. "if X is human, then X is mortal"
- Intelligence and Automation in Networked Embedded Systems
 - E.g. Building Automation:
 - If "motion sensor inside room detects movement" AND
 - "time is between 9 am and 5 pm" AND
 - "room temperature is lower than 19 degrees" THEN
 - "turn on room heating"



Data-Driven Machine Intelligence

- Inductive Reasoning
 - Derive models from observations
 - E.g. “All swans we have seen are white”, therefore “All swans are white”, therefore “I predict that an unknown swan we haven’t seen is also white”
- In relatively “simple” problems, we can hand-craft models out of observations that have high predictive power
 - E.g. We can use Newton’s Law’s to predict the exact location of a planet in the future
- In more “complex” problems, we instruct machines to find a model that explains the input data and use it to predict future data
 - Machine Learning



Deep (Artificial) Neural Networks

- Very good at constructing complex models with high predictive power based on observations
- Require a lot of computing power
- Require a lot of observations (input data)
- Enabled by advances in Cloud Computing and Big Data
- In this context, Networked Embedded Systems play the role of (big) data generators

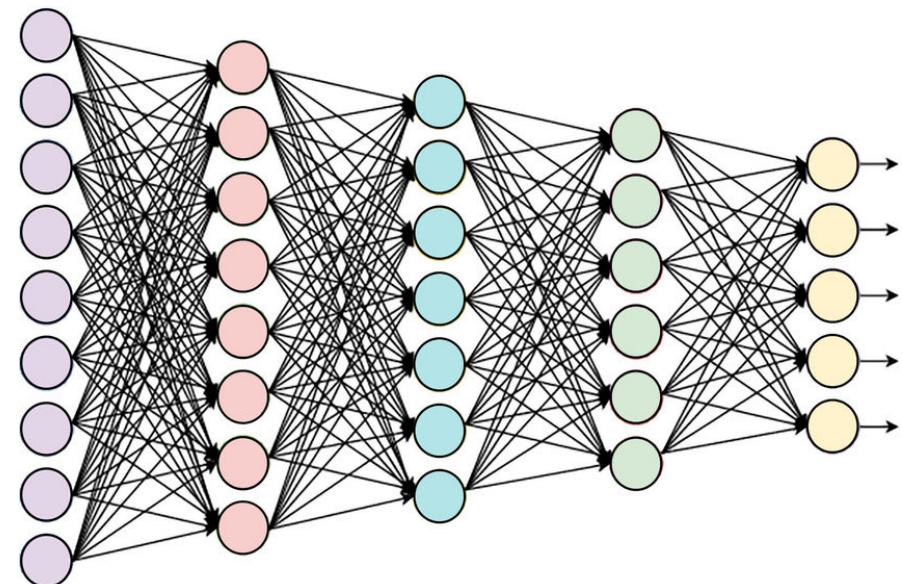


Image source: <http://dx.doi.org/10.1109/ACCESS.2020.3009819>

Cloud-based IoT Systems

- Advantages
 - Lots of computing power
 - Lots of storage
 - Possible to train really accurate but very resource-consuming deep learning models
- Disadvantages
 - High cost to transfer, store, process lots of data
 - Latency may be prohibiting for delay-sensitive use cases
 - Dependency to infrastructure (no coverage zones)
 - Privacy (potentially sensitive data goes to third parties)

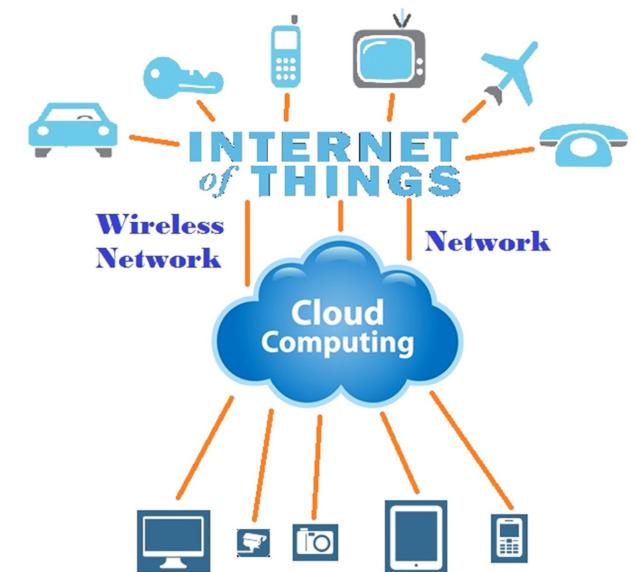


Image source: <https://doi.org/10.1016/j.future.2016.11.031>

Cloud-Fog-Edge Continuum

- Computing and AI is distributed in a cloud-fog-edge continuum as needed by the applications
- Embedded AI
 - AI comes to Embedded Systems (inference, training)
 - Embedded System become more autonomous (sensing, knowledge extraction, planning, decision making)
- Local Fog Servers are employed when more computing power or data fusion is needed
 - Privately-owned or offered as a service
- Cloud Data Centres used for really demanding tasks (e.g. training NLP models)

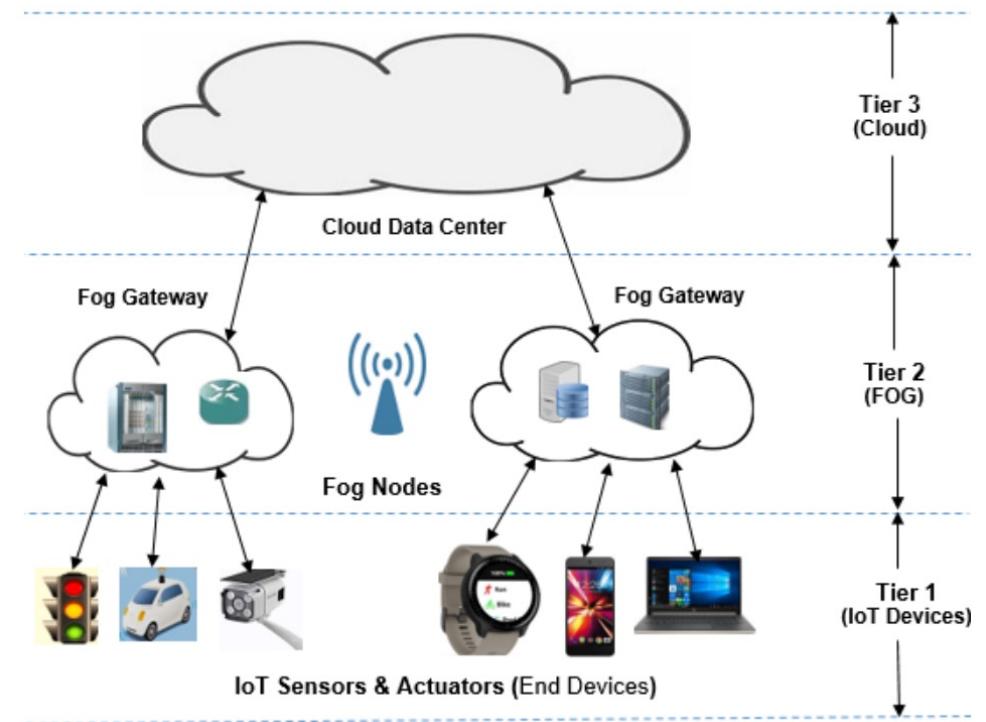
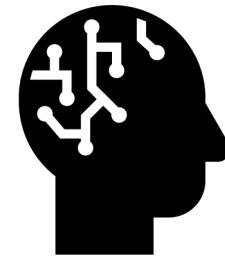
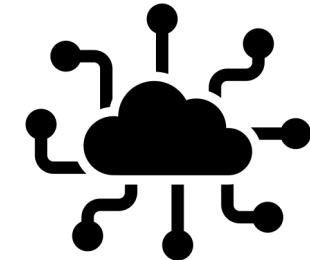


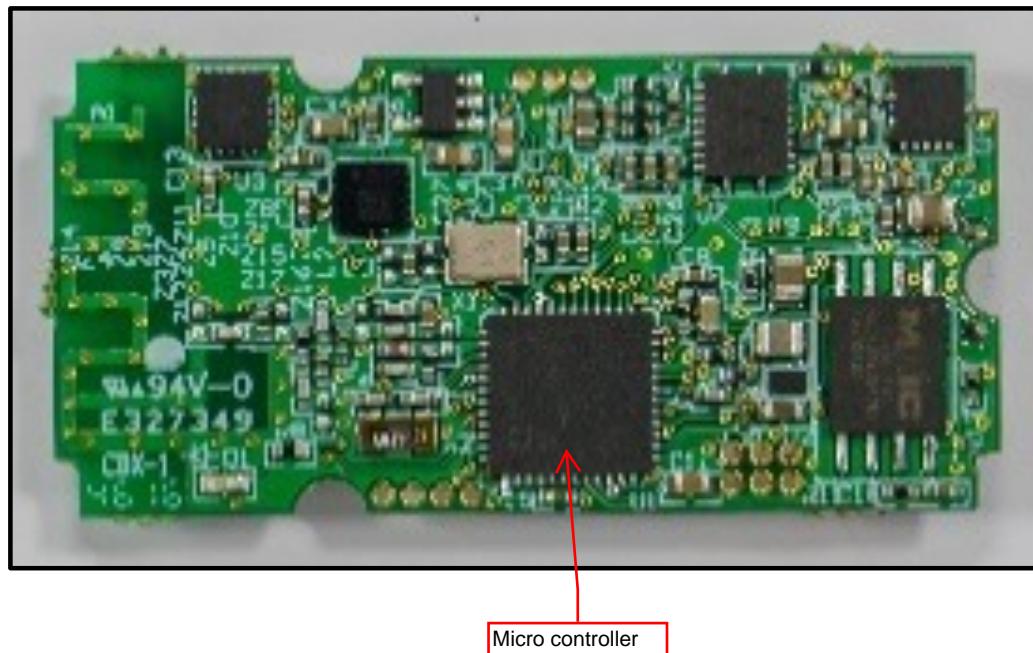
Image source: <https://doi.org/10.1109/ACCESS.2020.3032388>

Two big trends in Embedded Systems

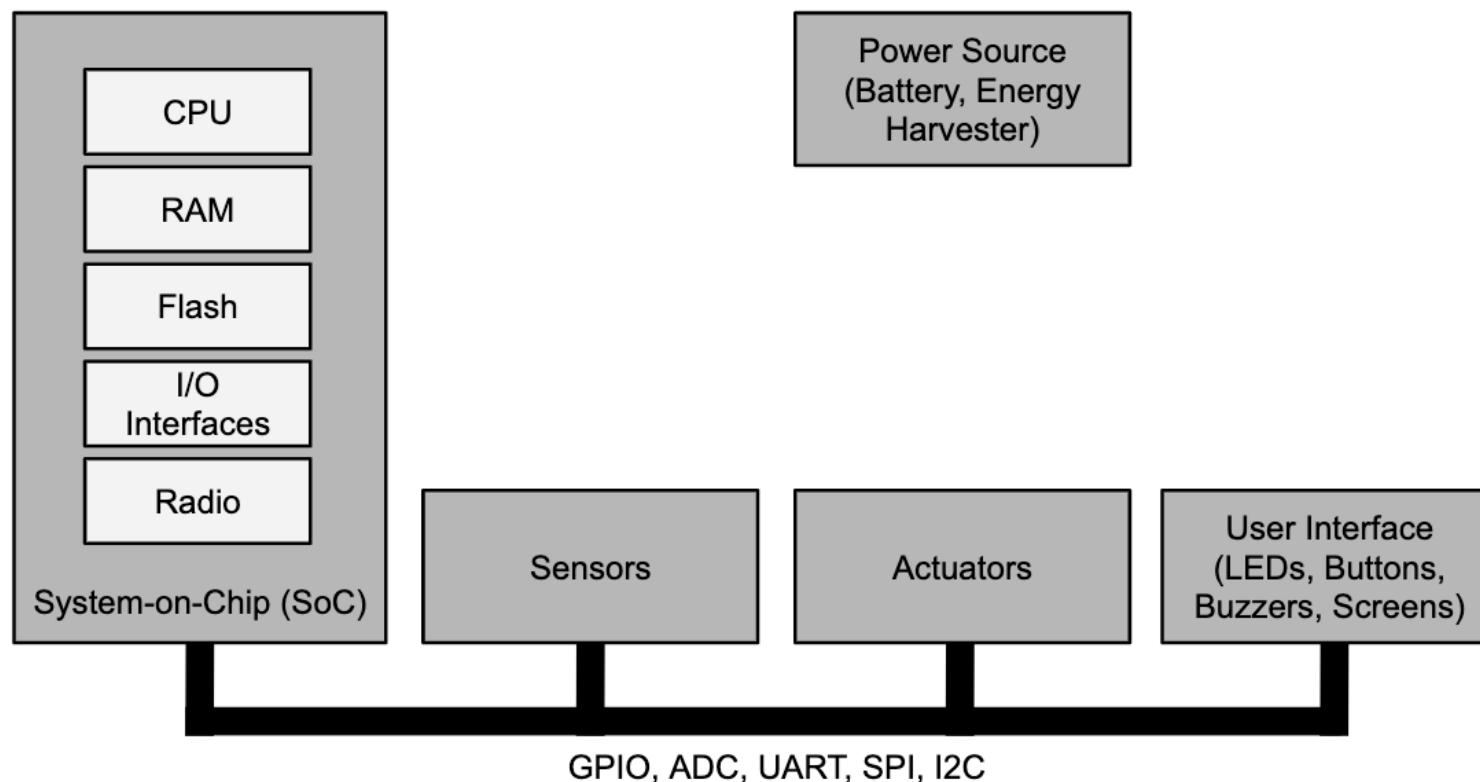
- **Connectivity**
 - Embedded Systems are getting connected, forming larger and wider (potentially global) collaboration networks
- **Intelligence**
 - Embedded Systems are becoming more intelligent, processing data locally, capable of embedded AI, and autonomous decision making



What are the main components of an Embedded System?



Anatomy of an Embedded System



Microcontrollers (MCU) and Systems-on-Chip (SoC)

- Small computer in a chip
 - Processor (CPU)
 - RAM
 - Input/Output Peripherals
- System-on-Chip (SoC)
 - MCU (processor, RAM, I/O)
 - Non-volatile storage (flash memory)
 - Network interfaces (wireless radio)
 - Other processing units (GPU, DSP, accelerators)
 - Integrated Sensors



Image source: Wikipedia

Example SoC

- CC2650 by Texas Instruments
 - ARM-based CPU
 - Memory (RAM, Cache, ROM, Flash)
 - Wireless Module with its dedicated CPU/RAM
 - Sensor Controller
 - Various controllers for interfaces
 - Various timers
 - Crypto module
 - Integrated temperature sensor
 - Integrated battery monitor
 - Direct Memory Access (DMA)

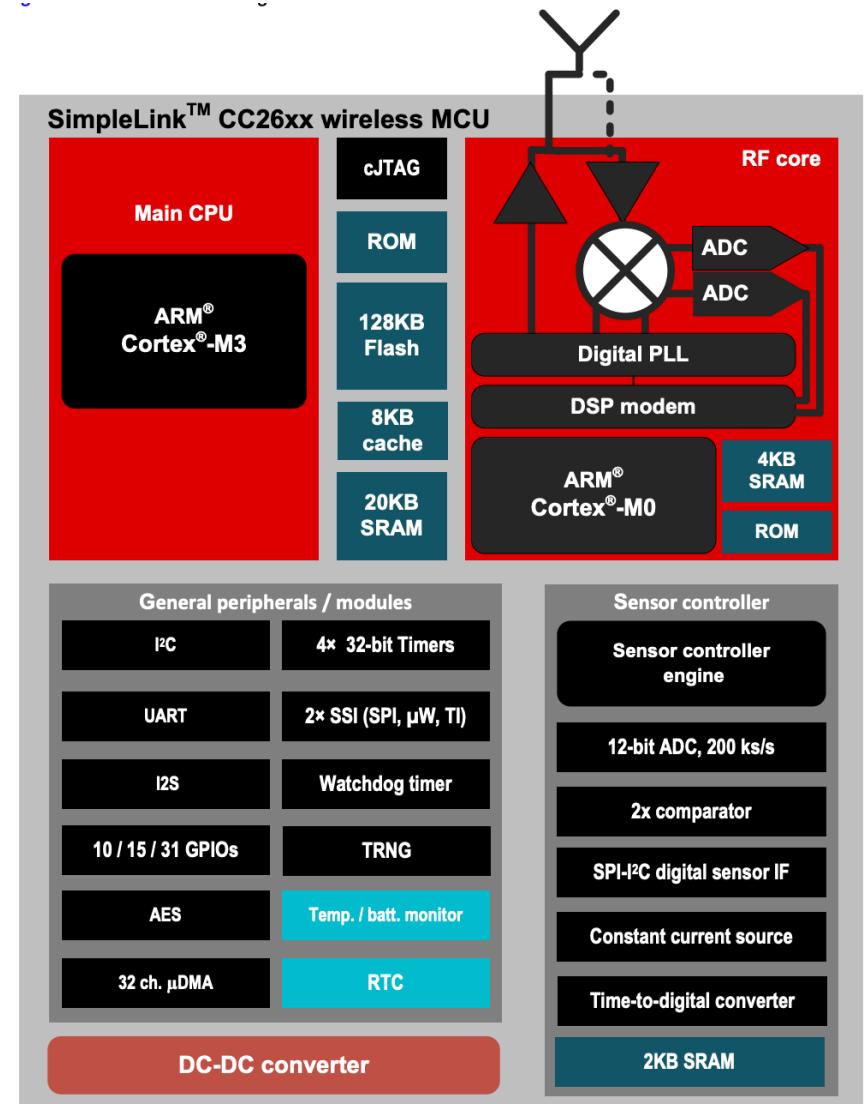


Image source: CC2650 Datasheet by Texas Instruments

Copyright © 2016, Texas Instruments Incorporated

Processors

- 8-bit, 16-bit, 32-bit, 64-bit architectures
- Popular architectures/vendors
 - AVR (Atmel)
 - PIC (Microchip)
 - MSP (Texas Instruments)
 - STM (ST Microelectronics)
- ARM (licenses CPUs to chip vendors)
 - ARM Cortex-A series (generic applications)
 - ARM Cortex-R series (real-time)
 - ARM Cortex-M series (embedded, low-power)

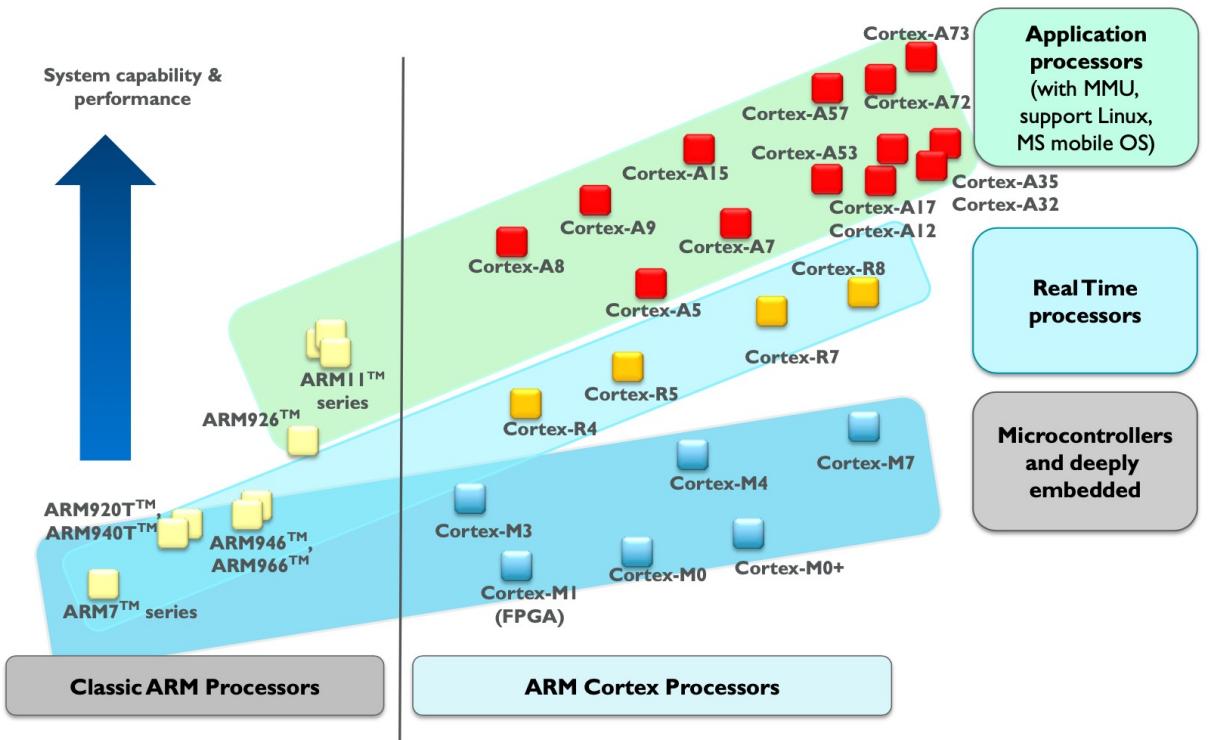
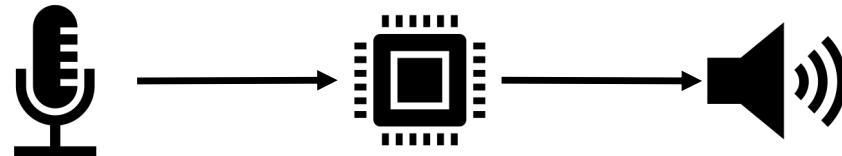


Image source: Cortex-M for Beginners by ARM

Transducers: Sensors and Actuators

- Sensors/Actuators are transducers that convert a signal from one form of energy to another
- Input Transducers convert a physical quantity into an electrical signal
 - Sensors (e.g. accelerometer, thermometer, microphone, cameras)
- Output Transducers convert an electrical signal into a physical quantity
 - Actuators and other output devices (e.g. motors, heaters, speakers, LEDs)
- Types of Transducers
 - Electromechanical
 - Electromagnetic
 - Electrochemical
 - Electroacoustic
 - Electro-optical
 - Thermoelectric



Sensors

- Analogue Sensors
 - Produce an electrical signal that correlates with a physical phenomenon
 - To be processed by a digital system, we need an Analogue-to-Digital Converter (ADC)
- Digital Sensors
 - Analogue sensors packed in a chip with an ADC and a controller

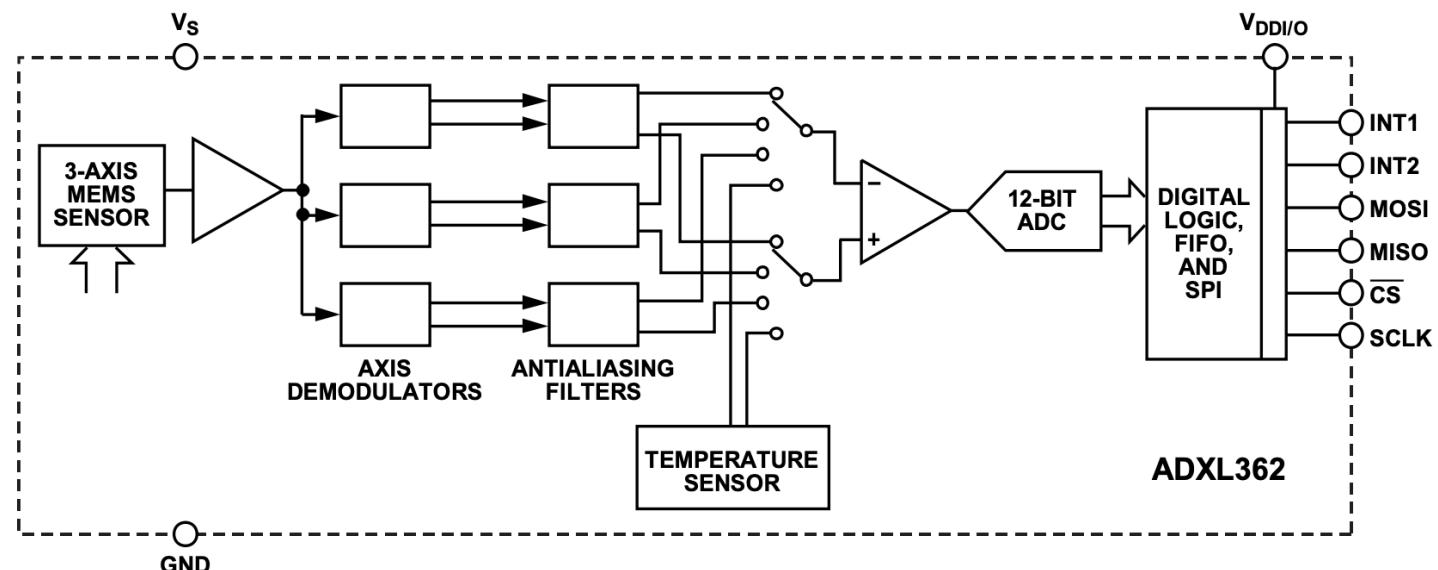


Image source: ADXL362 Datasheet by Analog Devices

Types of Sensors

- Electromechanical
 - Accelerometer, gyroscope, flow meters, air pressure
- Electromagnetic
 - Magnetometer, contact sensors, RADAR
- Electrochemical
 - pH, gas sensors, chemical detection
- Electroacoustic
 - Microphone, ultrasound, piezoelectric sensors
- Electro-optical
 - Camera, infrared camera, light sensor, PIR, LIDAR, heart rate (PPG)
- Thermoelectric
 - Temperature sensor, body temperature

Actuators/Output Devices

- Actuators/Output Devices
 - Control/manipulate an physical quantity with an electric signal
 - To be controlled by a digital system, we need a Switch, Digital-to-Analogue Converter (DAC), Signal Generator or Modulator
- Controller Chips
 - Implement a command interface to control the physical output

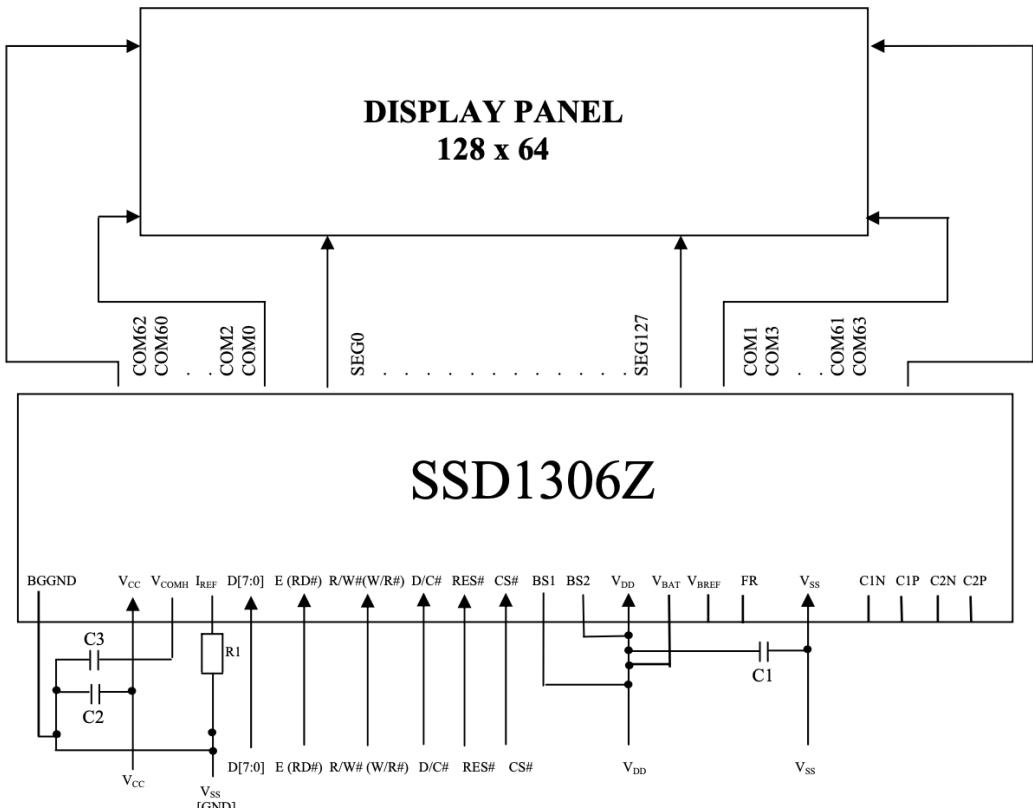


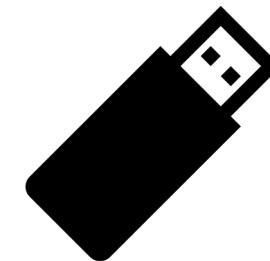
Image source: SSD1306 Datasheet by Solomon Systech

Types of Actuators/Output Devices

- Electromechanical
 - Motors, valves, vibration motor
- Electroacoustic
 - Loudspeaker
- Electro-optical
 - Screen, LEDs
- Thermoelectric
 - Heater

Other Input/Output Components

- External Storage
 - Flash memory chips
 - SD cards
- Connectors/Interfaces
 - Jacks, pin headers, sockets
 - Antenna connectors, printed antennas
 - Jumpers, solder bridges
 - Etc



Power Management

- Embedded Systems are typically powered by DC sources
 - Batteries or via the mains through rectification
 - Power source is typically noisy and imperfect
- Voltage Regulators
 - Automatically maintain constant output voltage
 - Linear Voltage Regulator (linear voltage adjustment, dissipate power)
 - Switching Regulators (rapid on/off switching, more efficient)
- Voltage Converters (DC-to-DC converter)
 - Converts a source from one voltage to another
 - Components may require to be powered at different voltage levels
 - Step-down Converter (Buck Converter): output voltage is lower than the input voltage
 - Step-up Converter (Boost Converter): output voltage is higher than the input voltage

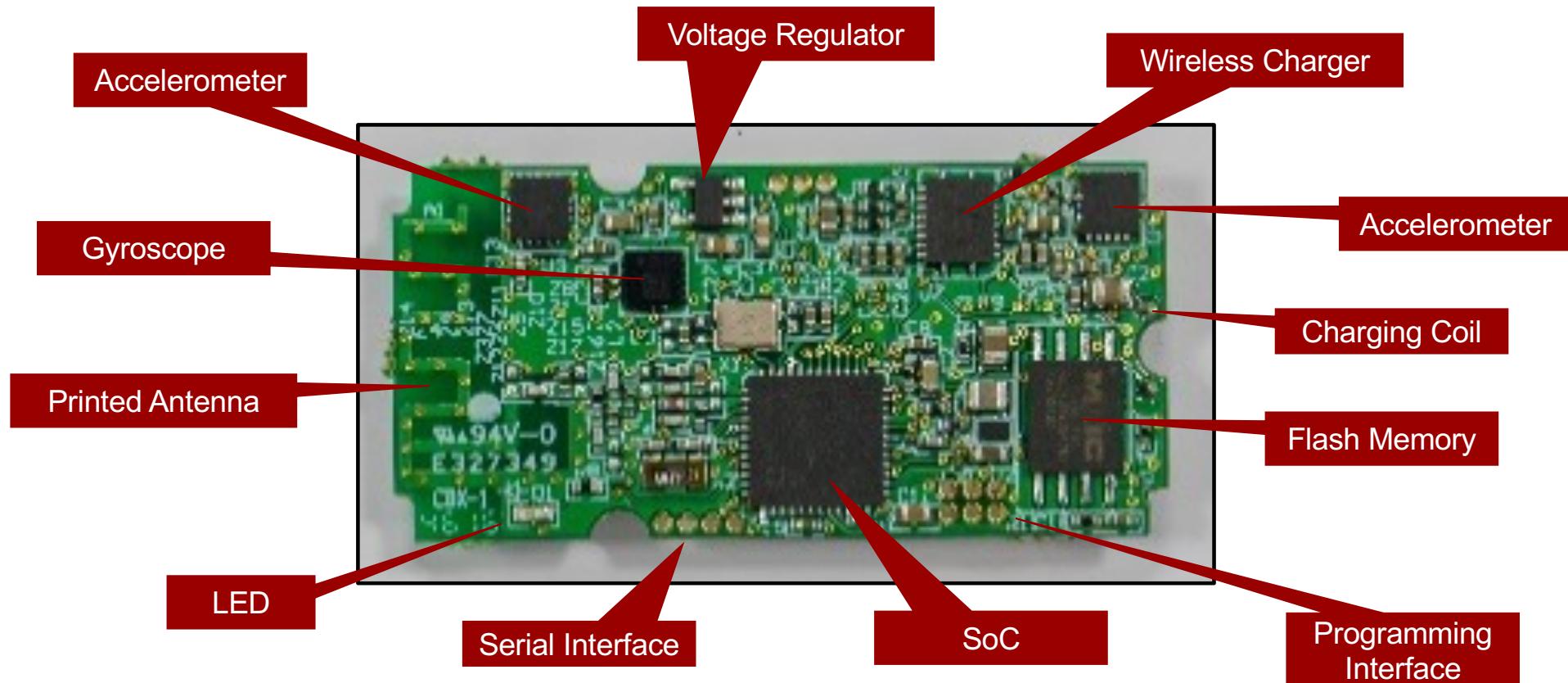


Power Management

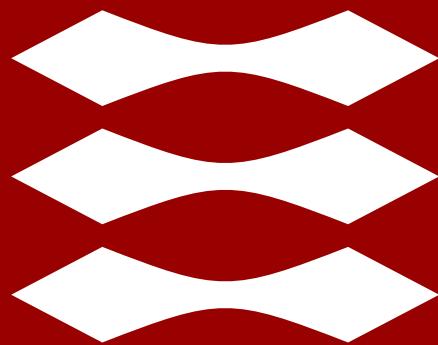
- Battery Charging
 - Battery Charger (e.g. via USB)
 - Inductive Wireless Power Receiver (e.g. Qi)
- Energy Harvesting
 - Solar Power Manager
 - RF Power Manager
 - Etc



The main components of an Embedded System



DTU



Networked Embedded Systems

Week 2: Practical Electronics for Embedded Systems

Xenofon (Fontas) Fafoutis

Professor

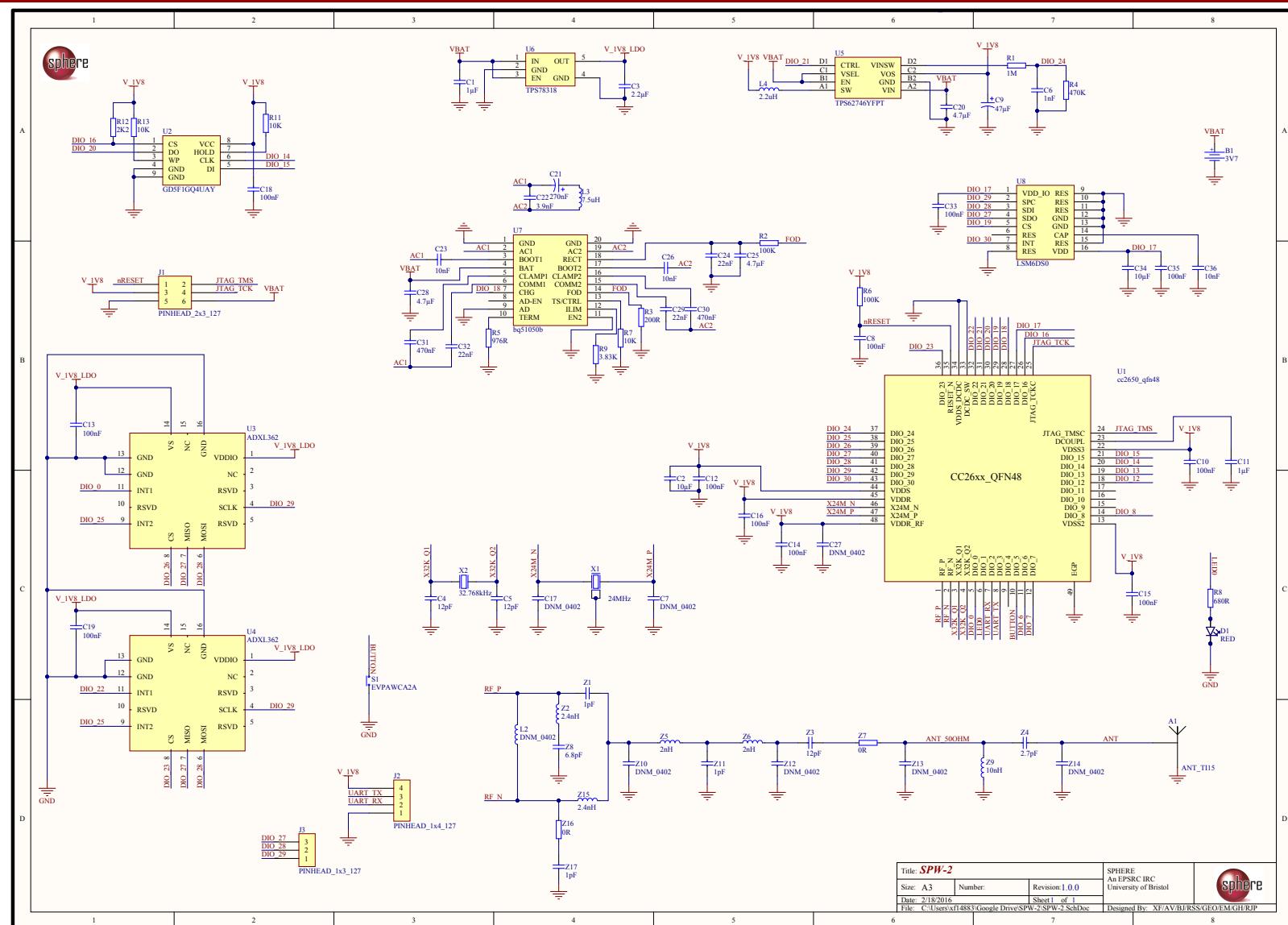
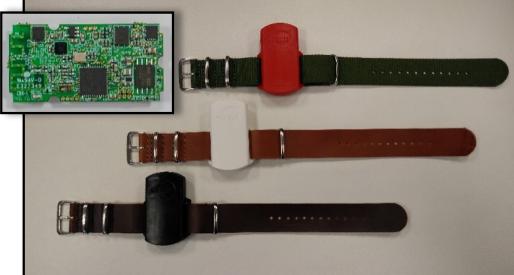
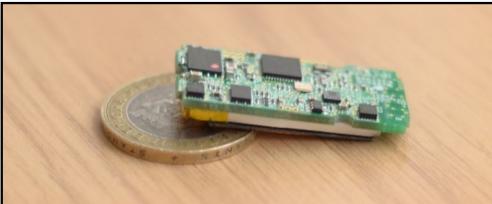
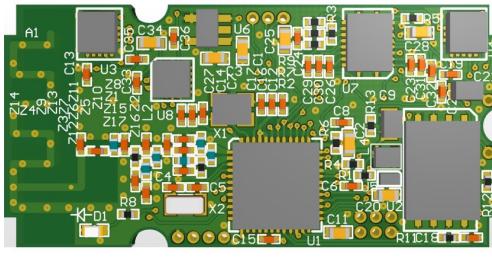
xeafa@dtu.dk

www.compute.dtu.dk/~xeafa

RF = Radio frekvens



A Wearable Embedded System



Schematic (or Circuit Diagram)

- Schematic is a model of an electronic circuit
 - Electronic circuit is electronic components interconnected in a closed loop
- Schematics make our lives easier
 - Abstracts electronic components to basic concepts
 - Easier to read and understand than a circuit

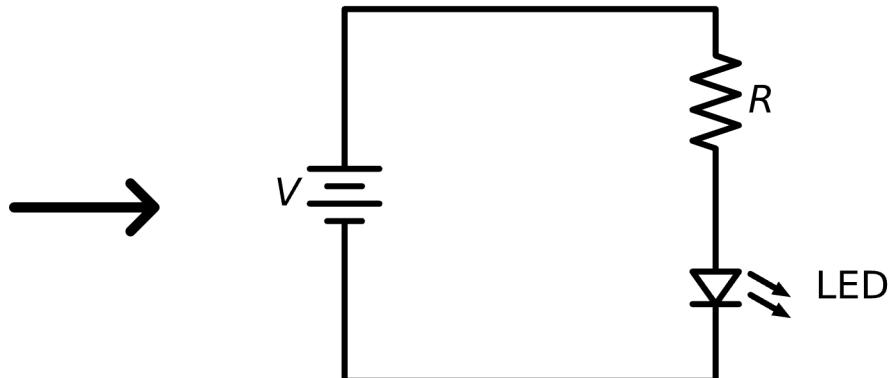
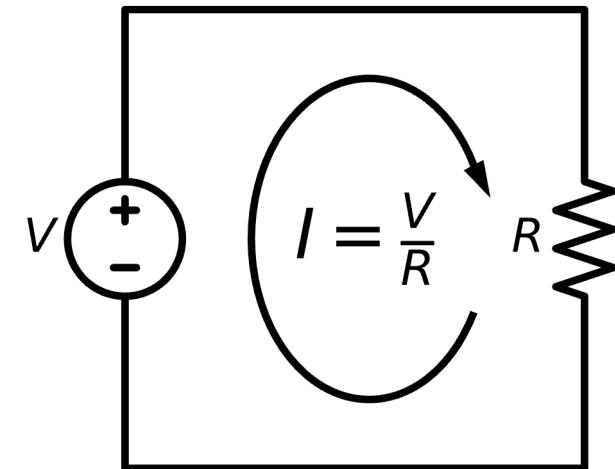


Image source: https://www.sciencebuddies.org/science-fair-projects/project-ideas/Elec_p077/electricity-electronics/led-dance-glove

Basic Concepts

- **Voltage (V, Volts):** Electric potential difference between two points
- **Current (I, Amperes):** Flow of electrically charged particles
- **Resistance (R, Ohm):** A measure of opposition to the flow of electric current
- **Ohm's Law (I=V/R):** The current through two points is directly proportional to the voltage across those points



Electric Power Source

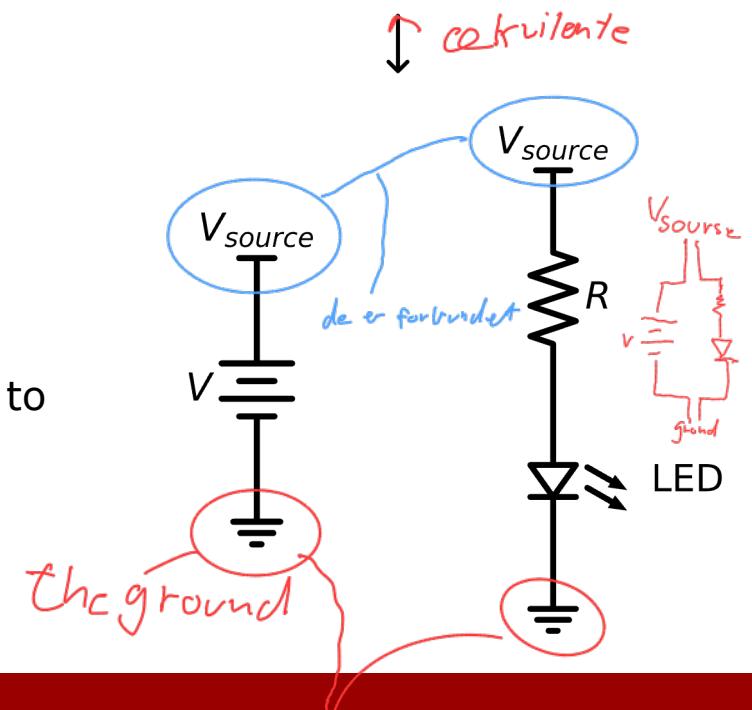
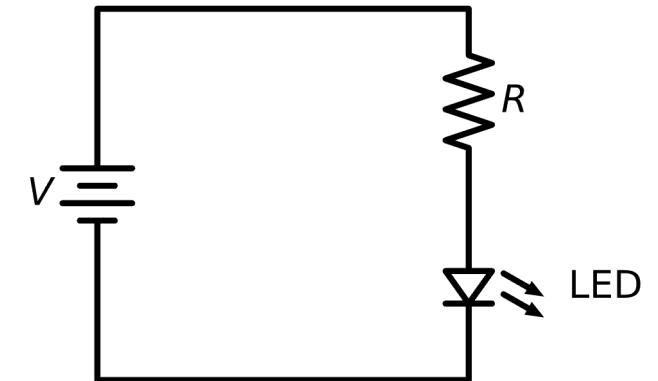
- **Direct current (DC)** is one-directional flow of electric charge (e.g. a battery)
- **Alternating current (AC)** is an electric current which periodically reverses direction (e.g. mains)
- **A rectifier** is an electrical device that converts AC to DC
- **Embedded Systems** typically require DC power sources



Image source: Wikipedia

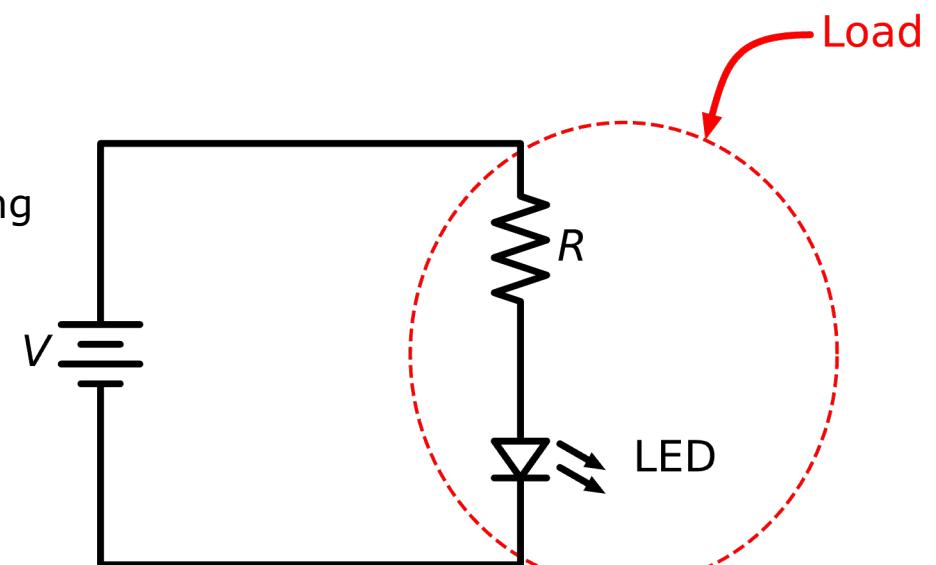
Voltage Sources

- DC sources can be **batteries** (or the mains via rectification)
- Voltage is the potential difference between two points
 - Yet, it is often convenient to measure the voltage from a common reference point, the ground
- **The ground** is a common reference point, sometimes physically connected to the earth
- As circuits grow in size and complexity, we use **labels** to represent the supply voltage and the ground
 - A label implies that all points with the same label are connected to each other



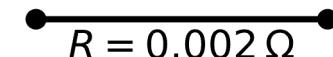
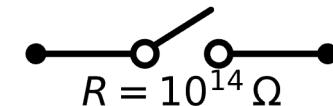
Electric Load (Power Consumers)

- **Electric Load** is the components or part of the circuit that consume electrical energy, converting it to light, kinetic energy, or heat
- **Electric Power (P, Watt):** Rate of electric transfer (consumption)
- $P = V I$
- The **Energy (E, Joule)** consumed is proportional to the time (t) of operation
- $E = P t = V I t$



Everything has resistance

- Resistance is opposition to current flow
 - Fixed resistance R from 0 Ohm to hundreds GOhms
- Insulators are assumed to have infinite resistance
 - In reality, an insulator like air has very high resistance
- Conducting wires are assumed to have zero resistance
 - In reality, a conductor like a copper wire has very low resistance

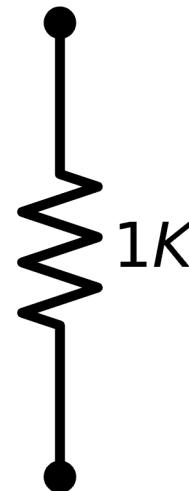


Quiz

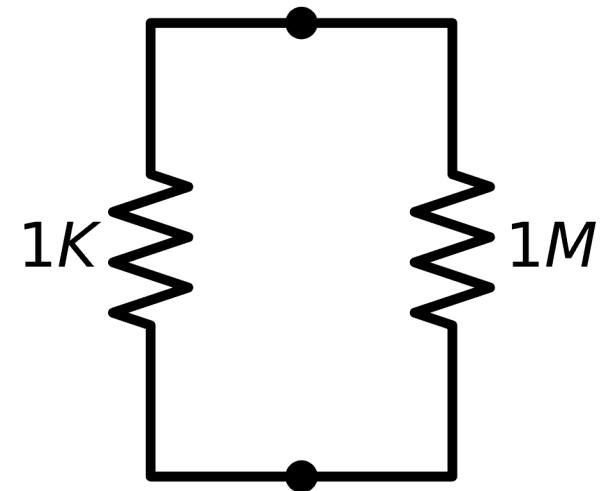
- If we apply the same voltage, which will have higher current?
- How much higher would that current be?
 - 0.1%
 - 1%
 - 10%

Serie

$$R = (R_1 \times R_2) / (R_1 + R_2)$$

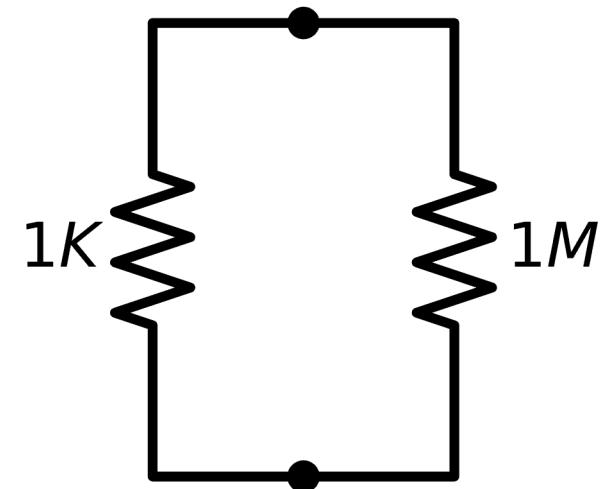
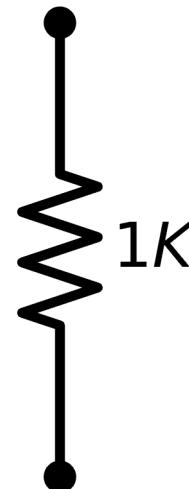


$$I = \frac{V}{R}$$



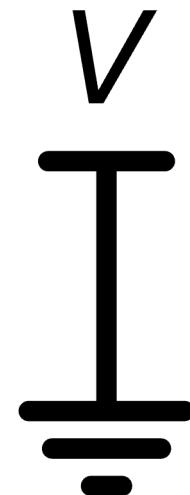
“Path of Least Resistance”

- Let's say we apply 1 V
- Left
 - $I = V / R = 1 / 1K = 1 \text{ mA}$
- Right 10^4 $10^3 + 10^4$
 - $R = (R_1 \times R_2) / (R_1 + R_2)$
 - $R = 999$
 - $I = V / R = 1 / 999 = 1.001 \text{ mA}$
- Practically, left and right are equivalent



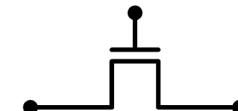
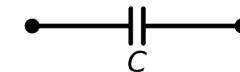
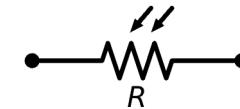
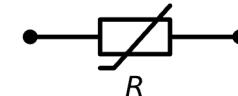
Short Circuit (or short)

- An often-unintentional path of “no resistance” from high voltage to ground
- Mathematically, it is an impossibility
 - $I = V/R = V/0 = \infty$
- Practically, any wire or conductive path has some very small resistance
 - Also, if power supply is a battery, batteries have internal resistance
- A short circuit practically leads to very high current, often limited by how much the power supply can provide
 - In high-power electronics this can be dangerous or lethal
 - In low-power electronics it can damage components or chips



Resistance can be variable

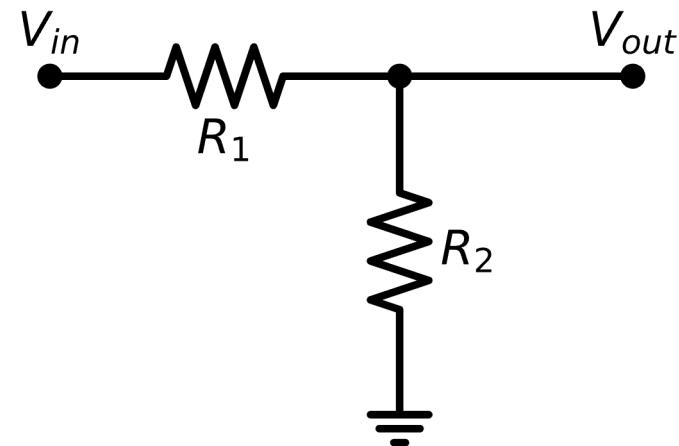
- Potentiometer: Mechanically-controlled resistance
- Thermistor: Temperature-dependent resistance
- Photoresistor: Luminosity-dependent resistance
- Capacitor/Inductor: Behave like frequency dependent resistors for AC (we call this reactance)
- Transistor: Electrically-controlled resistance



Voltage Divider

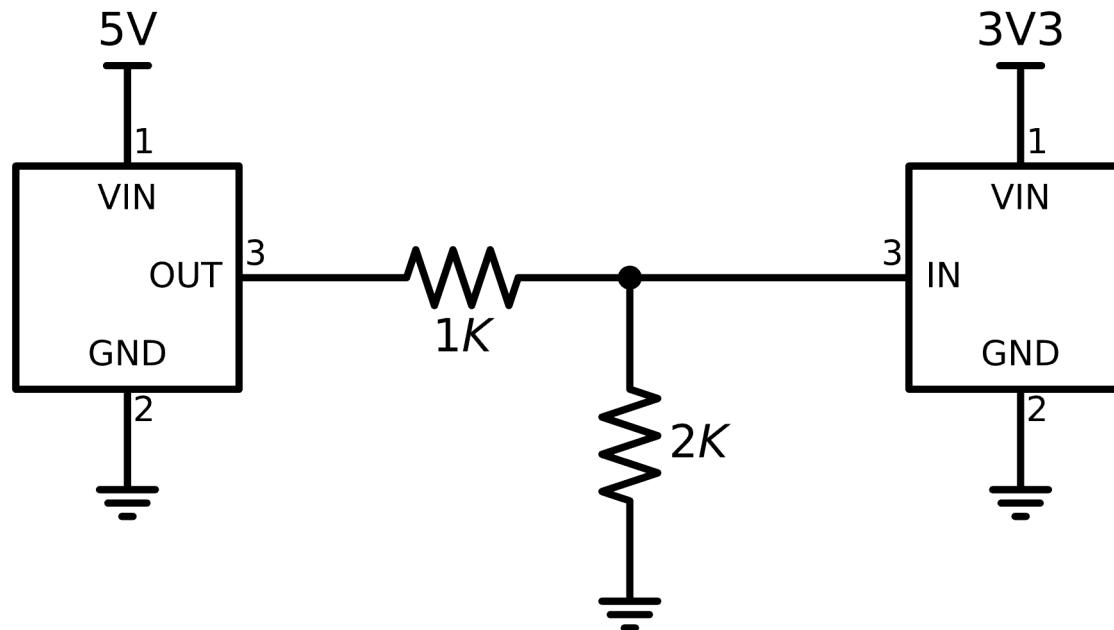
- **Signal Attenuator:** The output voltage is a fraction of the input voltage
- If R_1 is much smaller than R_2 , most of the input signal goes through (small attenuation)
- If R_1 is much larger than R_2 , most of the input signal does not go through (large attenuation)

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2}$$



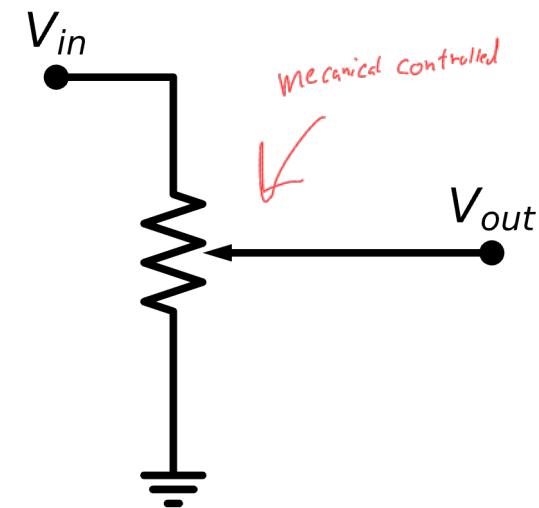
Level Shifter

- Interface the output of a device that encodes '1' as 5V to a device that expects input at maximum 3.3V without damaging the latter



Volume Control

- Mechanically-controlled signal attenuation
- A potentiometer has two terminals at the two ends of a resistor and one at the wiper arm
- The wiper can be moved, cutting the resistor in two parts with adjustable ratio
- This configuration can also be used to for user input
 - An Analog-to-Digital Converter (ADC) measures the output voltage and maps them to a set of numbers



Temperature Sensor

- The embedded systems can measure voltage not resistance
- A thermistor's resistance (R_2) depends on the temperature (T)
- The output voltage depends on the ratio of the two resistors
- Therefore, the output voltage depends on the temperature

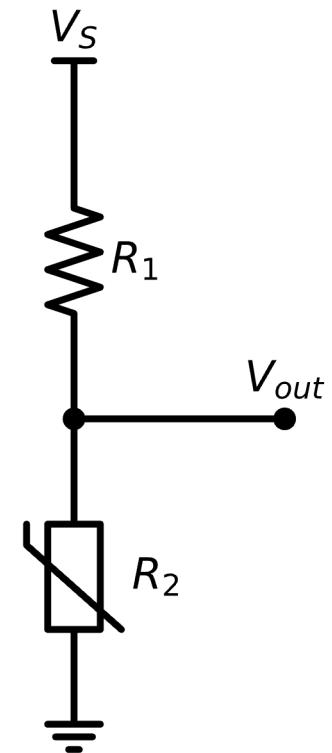
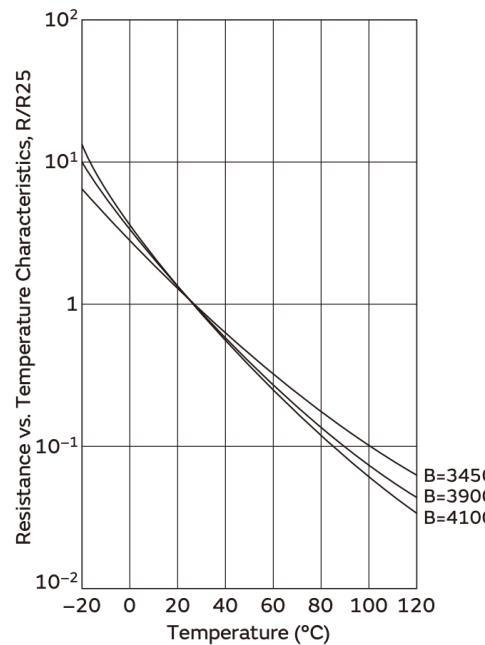
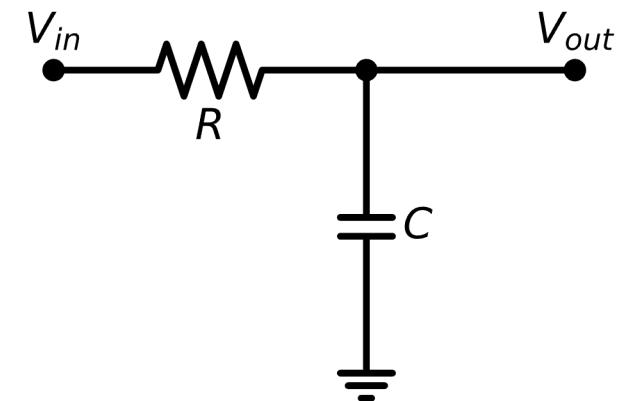
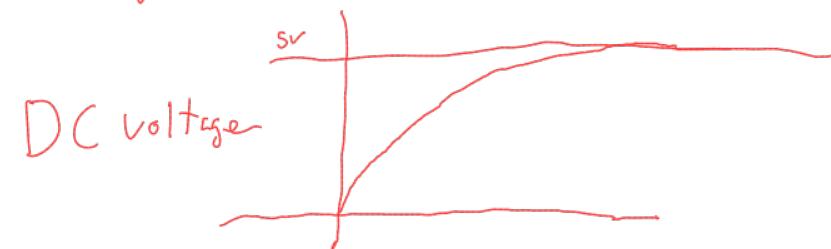


Image source: NTC Thermistors Datasheet by Murata

Capacitors

- Capacitors can store and provide electrical energy
 - They behave like tiny “power supplies”
- Capacitors do not let DC current to go through
- Capacitors let AC current go through
- They behave like frequency-dependent “resistors”
 - High frequencies will see low “resistance”
 - Low frequencies will see high “resistance”
- RC Low-Pass Filter
 - High frequencies are attenuated a lot
 - Low frequencies are attenuated a bit

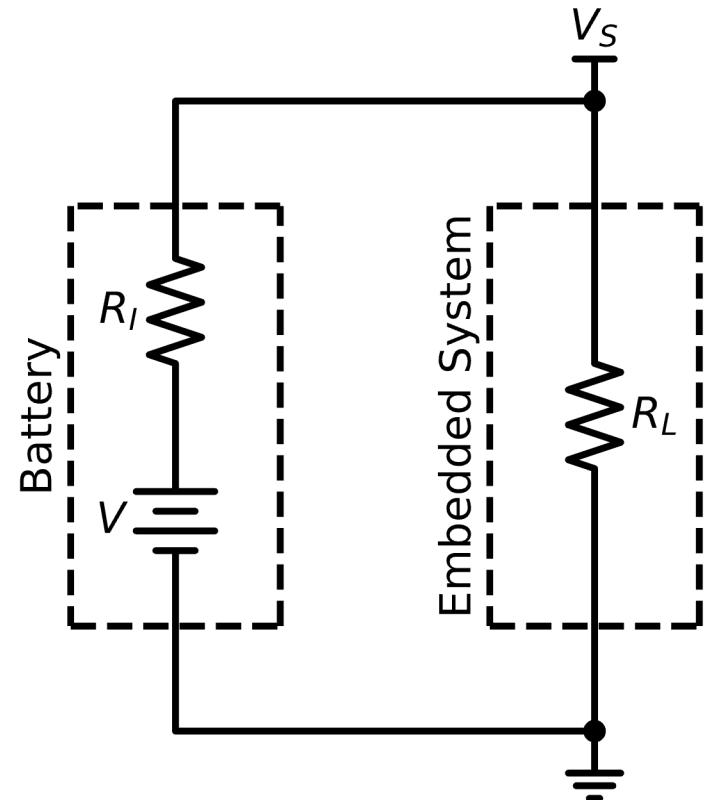
they have a charging period.





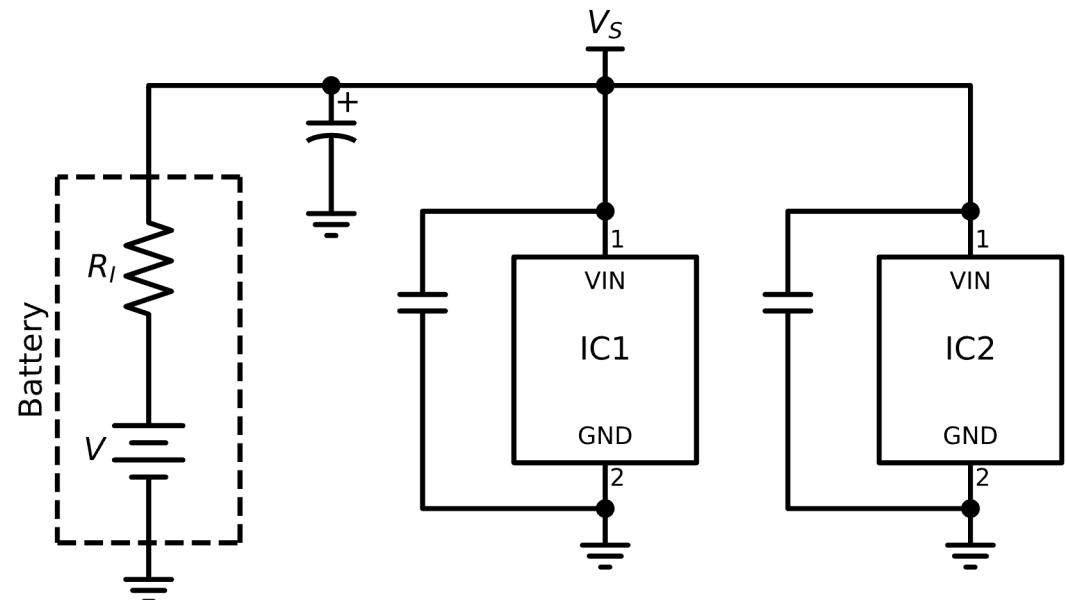
Batteries are imperfect power sources

- Batteries have internal resistance (R_I)
 - It increases as they're used
- The internal resistance creates a voltage drop proportional to the current that the system draws
- The supply voltage (V_S) will be less than the battery voltage (V)
- As the system goes through various states of operation it may need high current for a short period of time
 - Low current -> small voltage drop (no problem)
 - High current -> large voltage drop (dangerous)
- Large voltage drops can get the supply voltage below the minimum operating voltage of the embedded system!



Decoupling Capacitors

- Capacitors in parallel and physically very close to various components and integrated circuits
- Act as local power supply or local energy reservoir
- Filter out high-frequency noise in power supply signals
- Can support the battery in moments that require high peaks of current for a short time
- They oppose changes to supply voltage
 - If voltage goes up, they absorb excess energy
 - If voltage drops, they provide power to the IC



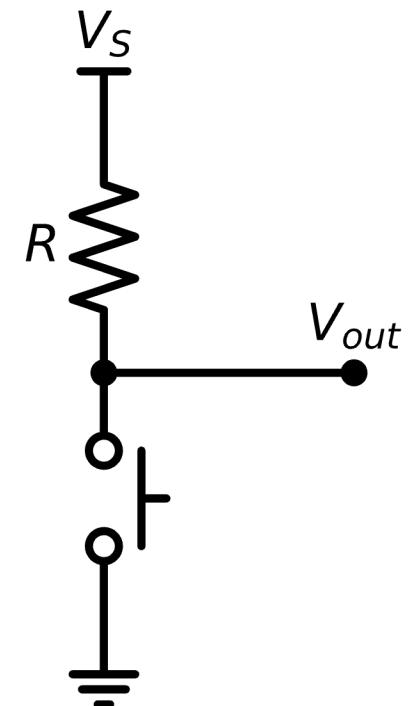
Switches and Buttons

- Mechanically open/close the circuit
- Switch: holds open/closed state
- Button: closed when pressed, open when released



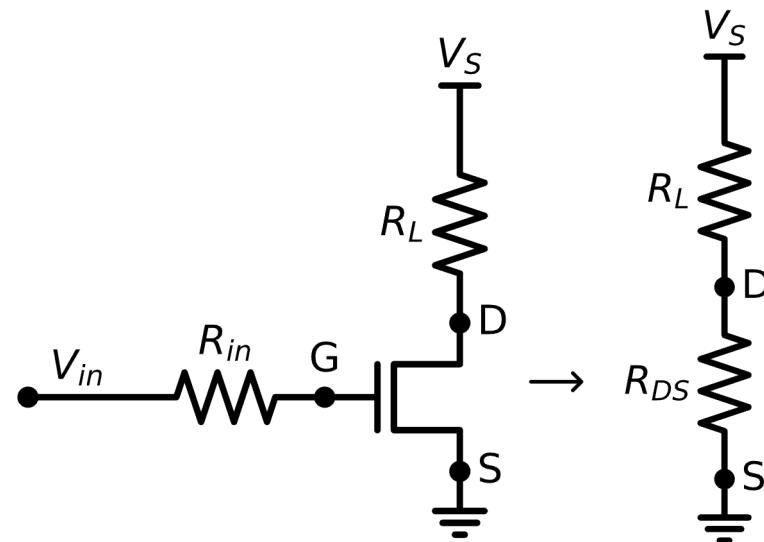
Detecting when a button is pushed

- Embedded Systems can measure voltage
 - We need to convert the push of the button in a change in voltage
- When button is not pushed, the circuit is open, as there is no path to the ground
 - Open circuit -> no current through R -> no voltage drop
 - Therefore, the output voltage is equal to V_s
- When button is pushed, the circuit is closed, as there is a path to the ground
 - Closed circuit -> current goes through R -> output (V_{out}) directly connected to ground
 - Therefore, the output voltage is equal to 0



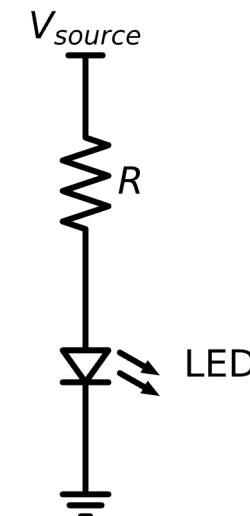
Transistor (MOSFET) as a Switch

- Controlling a switch with an electric signal
- MOSFETs are like electrically controlled resistors
- The resistance between two terminals depends on the voltage at the third terminal
- $V_{in} = V_S \rightarrow R_{DS}$ very very small \rightarrow switch closed \rightarrow current flows through $R_L \rightarrow$ load gets power
- $V_{in} = 0 \rightarrow R_{DS}$ very very big \rightarrow switch open \rightarrow no current flows through $R_L \rightarrow$ load gets no power



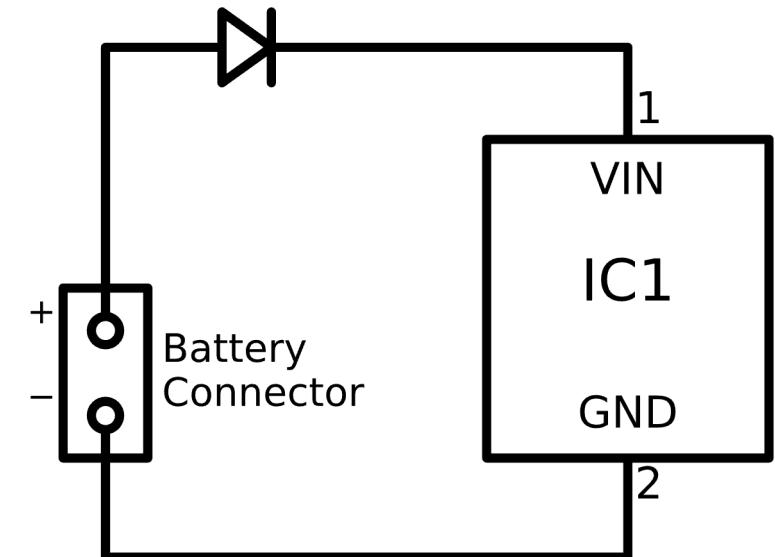
Diodes and LEDs

- One-way conductor
- Ideally zero resistance in one direction, infinite resistance in the other direction
- When current goes through there is a voltage drop and consumes power
- LEDs are diodes that emit light
- A series resistor (R) is needed to limit the current that goes through the LED to avoid damage



Reverse Voltage Protection

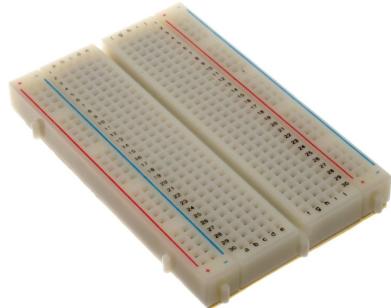
- Diode protects the circuit from reverse voltage
- When battery is connected correctly, current flows as expected, and the system is powered properly
- If the battery is connected reversely, the diode will block current flowing in the wrong direction, protecting the system from damage



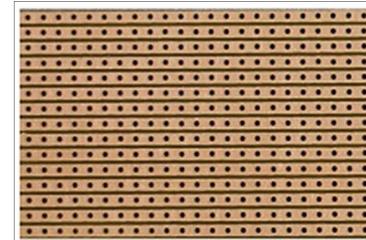
PCB Prototyping

- Before designing a PCB it is often a good idea to prototype the circuit or some parts of it
- Breakout boards allow easy access to the pins of small components

Breadboard



Veroboard

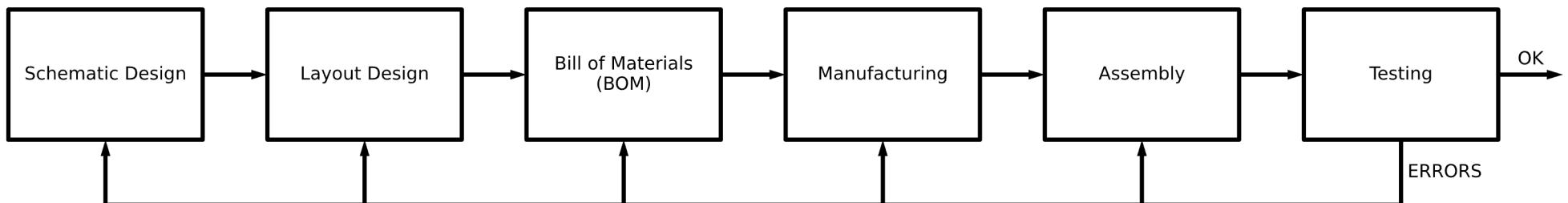
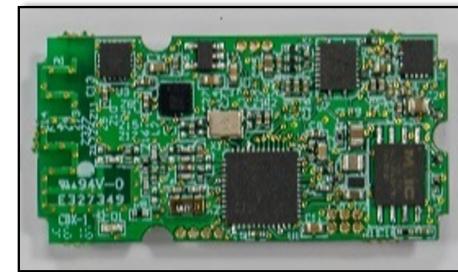


Breakout board



PCB Design

- Printed Circuit Board (PCB)
 - Copper tracks printed on a substrate
 - Exposed copper pads to mount components
- Designed through specialised software, named **Electronic Design Automation (EDA)**
 - Or Electronic Computer-Aided Design (ECAD)



PCB Design: Schematic

- The first step is the identification of key components and power source
 - Ensure compatibility, identify voltage level(s), etc
 - Keep it simple
- Often we don't have to select the values of passive components (resistors, capacitors, etc)
 - Components provide reference circuits (e.g. the decoupling capacitor should be 100nF)
 - There are lots of "open hardware" circuits that can be used as reference
- For each component, we need to make a schematic model based on the datasheet (matching pin numbers to functionality)

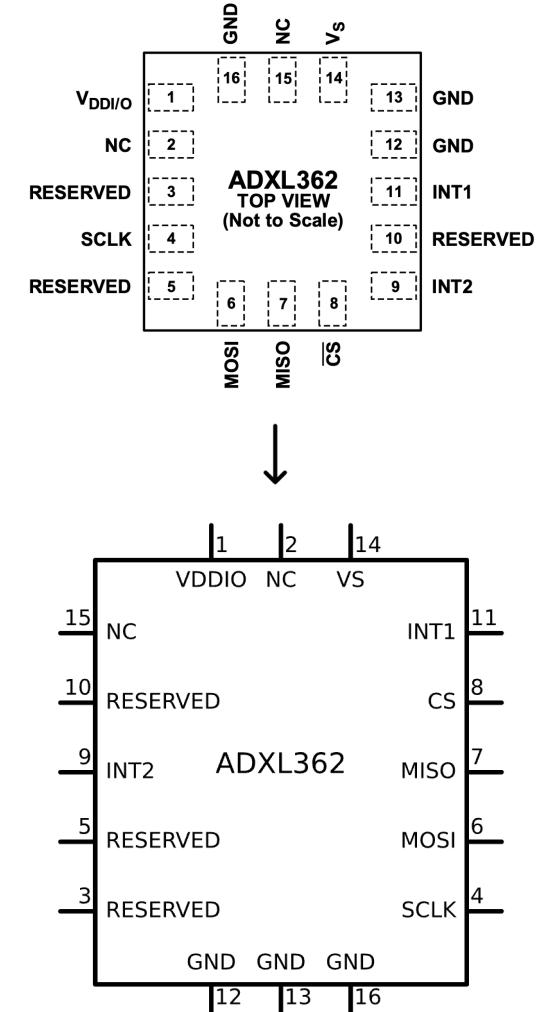


Image source: ADXL362 Datasheet by Analog Devices

PCB Design: Component Layout

- For each component, we need to create a footprint with exact dimensions, following the datasheet
- Attention: the bottom view is the mirror image of the top view
- Passive components come in standard sizes
- Imperial sizes (inches) and metric sizes (mm) are both used

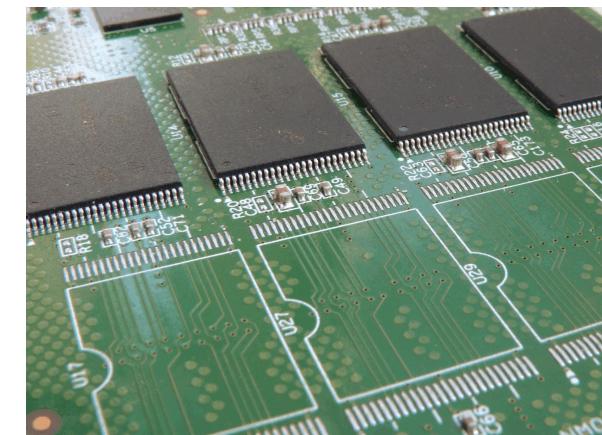
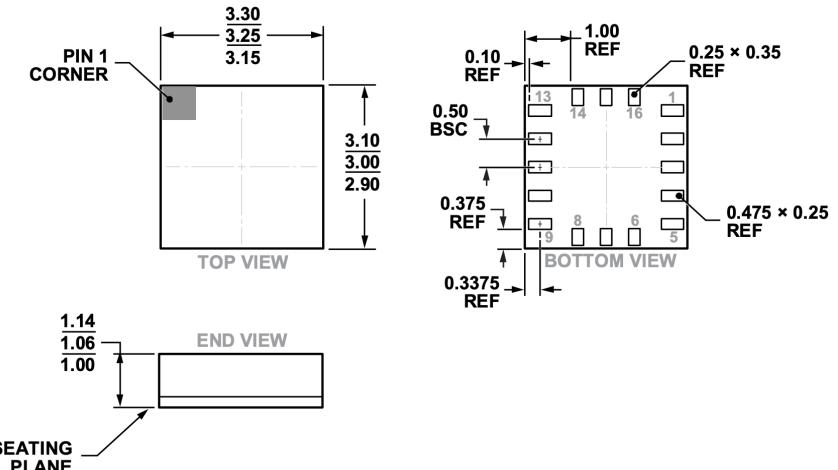


Image source: ADXL362 Datasheet by Analog Devices (top), Wikipedia (bottom)

PCB Design: Board Layout

- Place components on a board and connect them (routing)
- PCBs have 2 or more copper layers
 - Components can be placed at top and bottom layers
 - Internal layers have copper tracks for wiring
- Vias
 - Layers are connected vertically with vias
 1. Through hole via (top to bottom)
 2. Blind via (top/bottom to internal)
 3. Buried via (internal to internal)
- Ground Plane/Layer: Large copper areas connected to ground
 - Reduces noise and electromagnetic interference
 - Heat dissipation

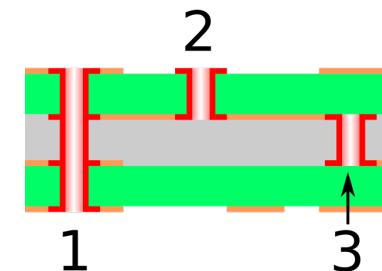
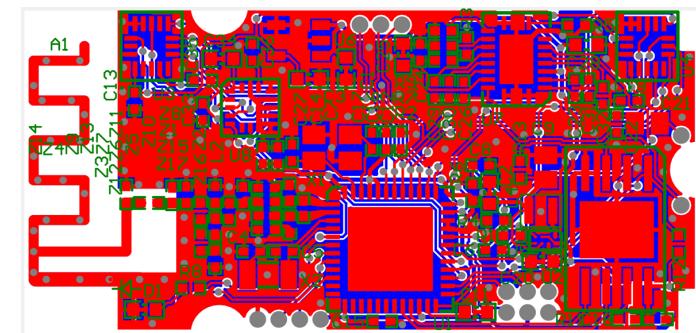


Image source: Wikipedia

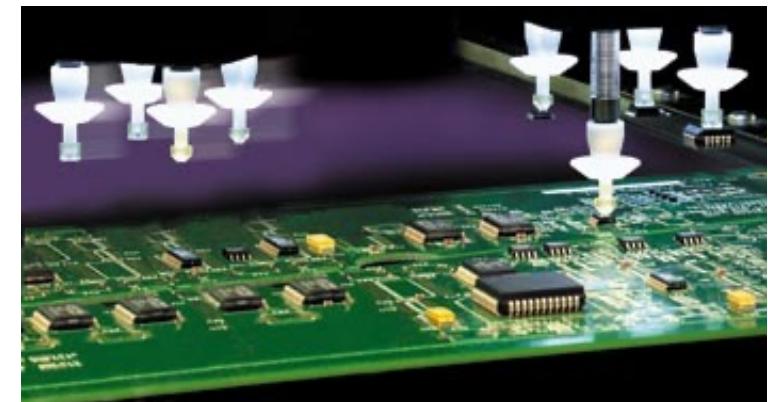
PCB Design: Bill of Materials (BOM)

- A shopping list of components (typically a spreadsheet)
 - Specifies manufacturer, supplier, price, etc

	A	B	C	D	E	F	G
1	SPW-2 (BOM v1.0)						
2							
3	Designator	Description	Comment	Manufacturer 1	Manufacturer Part Number 1	Supplier 1	Supplier Part Number 1
4							
5	SPW-2 PCB v1.0						
6	B1	LIPO261534	DNM				
7	C1	Capacitor, X5R, ±10%	1µF	TDK Corporation	C1608X7R1C105M080AC	Digi-Key	445-5131-1-ND
8	C2	Capacitor, X5R, ±10%	10µF	Murata Electronics North America	GRM188R60J106ME47D	Digi-Key	490-3896-1-ND
9	C3	Capacitor, X5R, ±10%	2.2µF	Taiyo Yuden	JMK107BJ225KA-T	Digi-Key	587-1254-1-ND
10	C4	Capacitor, NPO, ±2%	12pF	Murata Electronics	GRM1555C1H120JA01D	Mouser	81-GRM1555C1H120JA1D
11	C5	Capacitor, NPO, ±2%	12pF	Murata Electronics	GRM1555C1H120JA01D	Mouser	81-GRM1555C1H120JA1D
12	C6	Capacitor, X7R, ±10%	1nF	AVX	04025C102KAT2A	Mouser	581-04025C102K
13	C7		DNM_0402				
14	C8	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
15	C9	Capacitor, X5R, ±20%	47µF	Samsung Electro-Mechanics America, Inc.	CL21A476MQCLRNC	Digi-Key	1276-2420-1-ND
16	C10	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
17	C11	Capacitor, X5R, ±10%	1µF	TDK Corporation	C1608X7R1C105M080AC	Digi-Key	445-5131-1-ND
18	C12	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
19	C13	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
20	C14	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
21	C15	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
22	C16	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
23	C17		DNM_0402				
24	C18	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
25	C19	Capacitor, X7R, ±10%	100nF	Murata Electronics North America	GRM155R70J104KA01D	Digi-Key	490-6319-1-ND
26	C20	Capacitor, X5R, ±10%	4.7µF	Kemet	C0603C475K9PACTU	Digi-Key	399-3482-1-ND
27	C21	Capacitor, X5R, ±20%	270nF	Vishay / Vitramon	VJ0805Y274MXXTW1BC	Mouser	77-VJ0805Y274MXXTBC
28	C22	Capacitor, X7R, ±10%	3.9nF	Murata Electronics	GRM155R71H392KA01J	Mouser	81-GRM155R71H392KA1J
29	C23	Capacitor, X7R, ±10%	10nF	Kemet	C0402C103K5RACTU	Mouser	80-C0402C103K5R

PCB Design: Manufacturing and Assembly

- Manufacturing
 - Create the PCB itself
- Assembly
 - Place and solder the components on it
- These stages require machinery and often need to be outsourced to **PCB Fabrication Manufacturers** (fab)
- Cost increases with precision requirements
 - The smaller the costlier



A pick-and-place machine

Image source: Wikipedia

PCB Design: Testing

- PCBs like software may have bugs
 - Hardware revisions may be required
- PCB Fabrication Manufacturers would typically test their products
 - Errors can also be in the design stages
- Types of testing include:
 - Functional testing
 - Performance testing (power measurement)
 - Signal quality testing
 - Mechanical testing
 - Thermal testing
 - Specification verification
 - Safety certification

Electronic Testing Instruments: Multimeter

- Multimeter (Digital Multimeter, DMM)
 - Voltmeter (measures voltage, DC and/or AC)
 - Ammeter (measures current, DC and/or AC)
 - Ohmmeter (measures resistance, beeps when resistance is very low)
 - Other
- Operates at specific ranges
- Ideal for measuring constant values

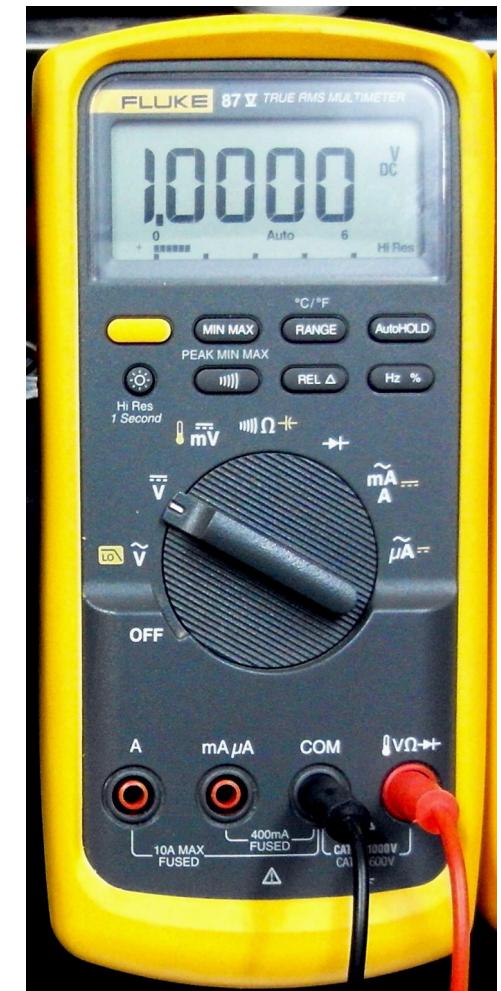


Image source: Wikipedia

Electronic Testing Instruments: DC Power Supply

- Provides DC electric power
- Output voltage is typically configurable
 - At a specified resolution
- Often come with multiple individually configurable outputs
- Output current can be limited to avoid accidental damage
- Operates at specific ranges

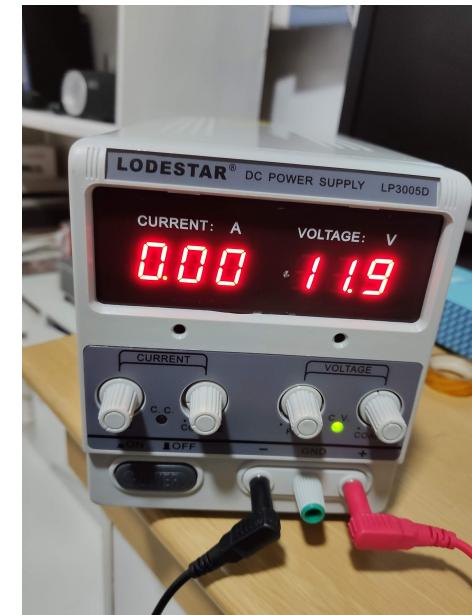


Image source: Wikipedia

Electronic Testing Instruments: Oscilloscope

- Measures and displays voltage signals over time
- Typically multiple voltage signals can be measured and displayed in parallel
- Possible to set triggers for the measurements
- Possible to calculate statistics (min, max), frequency, time duration
- Possible to extract data in a file

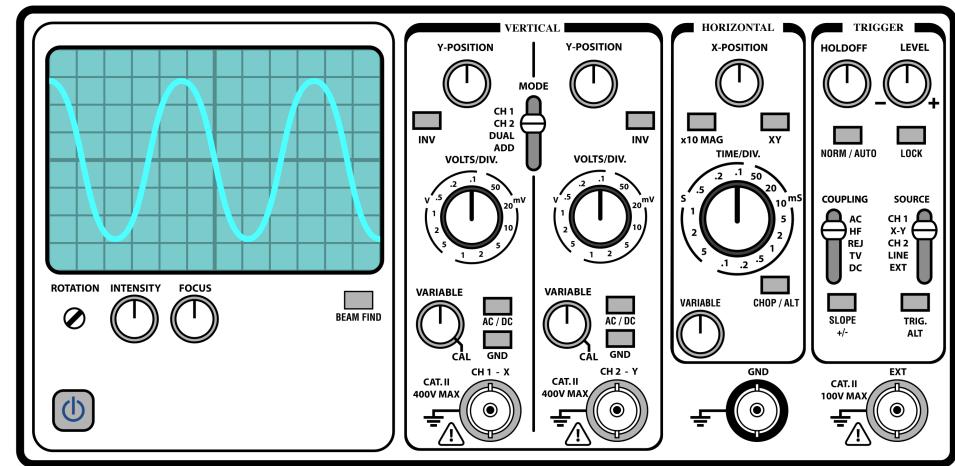


Image source: Wikipedia

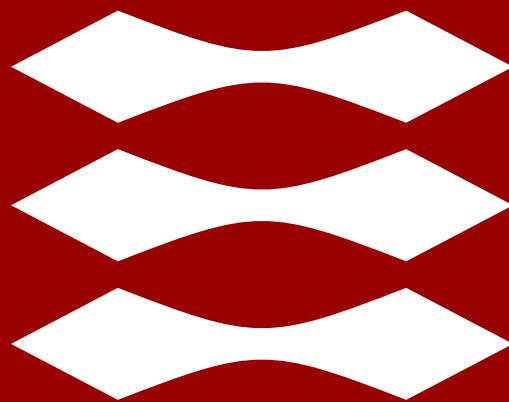
Electronic Testing Instruments

- Logic Analyser
 - Captures and displays multiple digital voltage signals
- Spectrum Analyser
 - Measures magnitude of input signal over a frequency range (spectrum)
- Signal/Function Generator
 - Generates signals, such as sine waves, square waves, etc
- Source Measurement Unit (SMU)
 - Supplies power and measure its voltage and current at the same time



Image source: Wikipedia

DTU



Networked Embedded Systems

Week 3: MCU Input/Output

Xenofon (Fontas) Fafoutis

Professor

xefa@dtu.dk

www.compute.dtu.dk/~xefa

Digital Logic

- Two distinguishable voltage levels are mapped in two a binary digit
- **High voltage** (typically, the supply voltage) is '1'
- **Low voltage** (typically, the ground) is '0'
- Real world is noisy, but noise can be tolerated
 - E.g. if $V > V_S/2$, then '1'

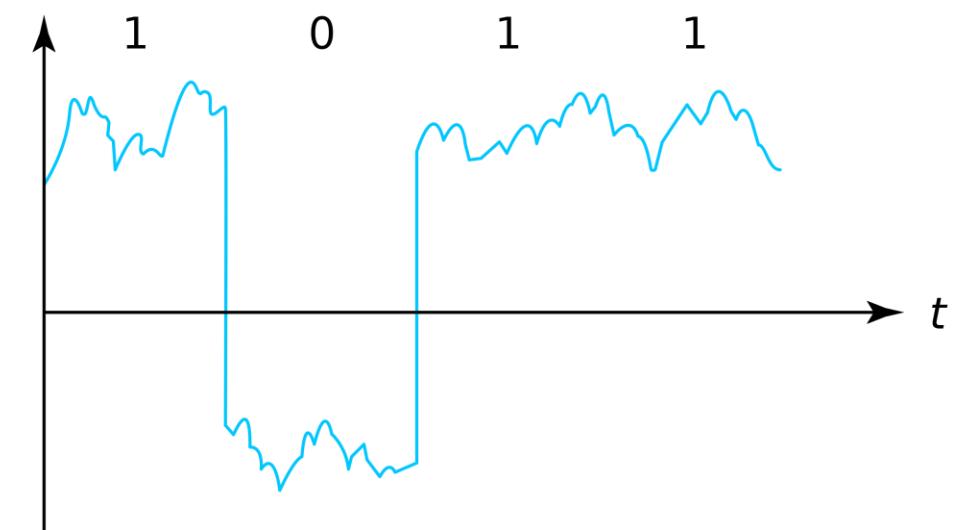
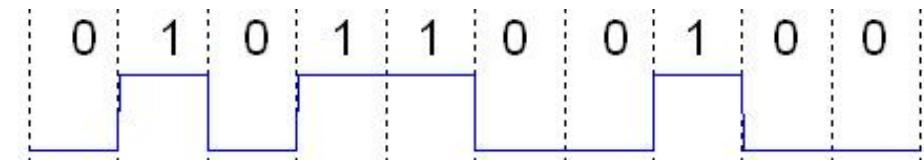


Image source: Wikipedia

General Purpose Input/Output (GPIO)

- A GPIO is controllable by software and can:
 - Set the logic level of a line to high or '1'
 - Set the logic level of a line to low or '0'
 - Read the logic level of the line
- Example: the nRF2832 has 32 GPIOs
 - Labelled as P0.00-P0.31 in the figure

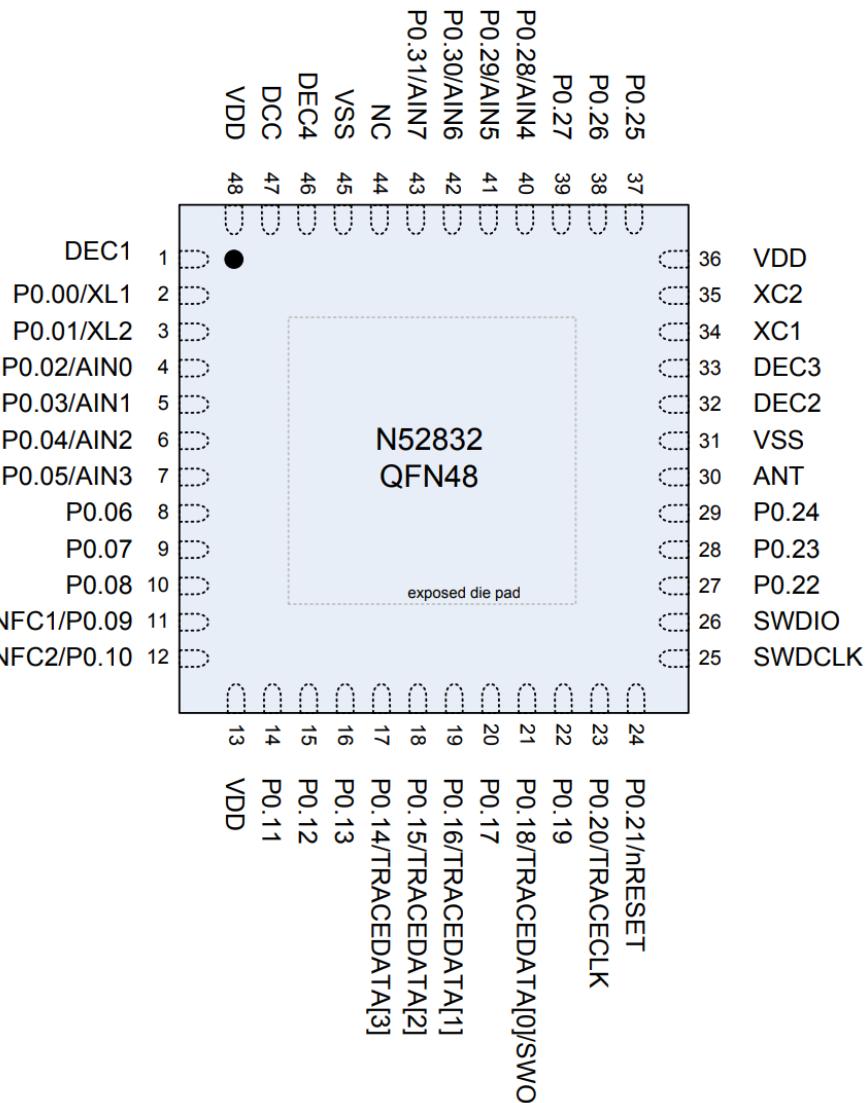
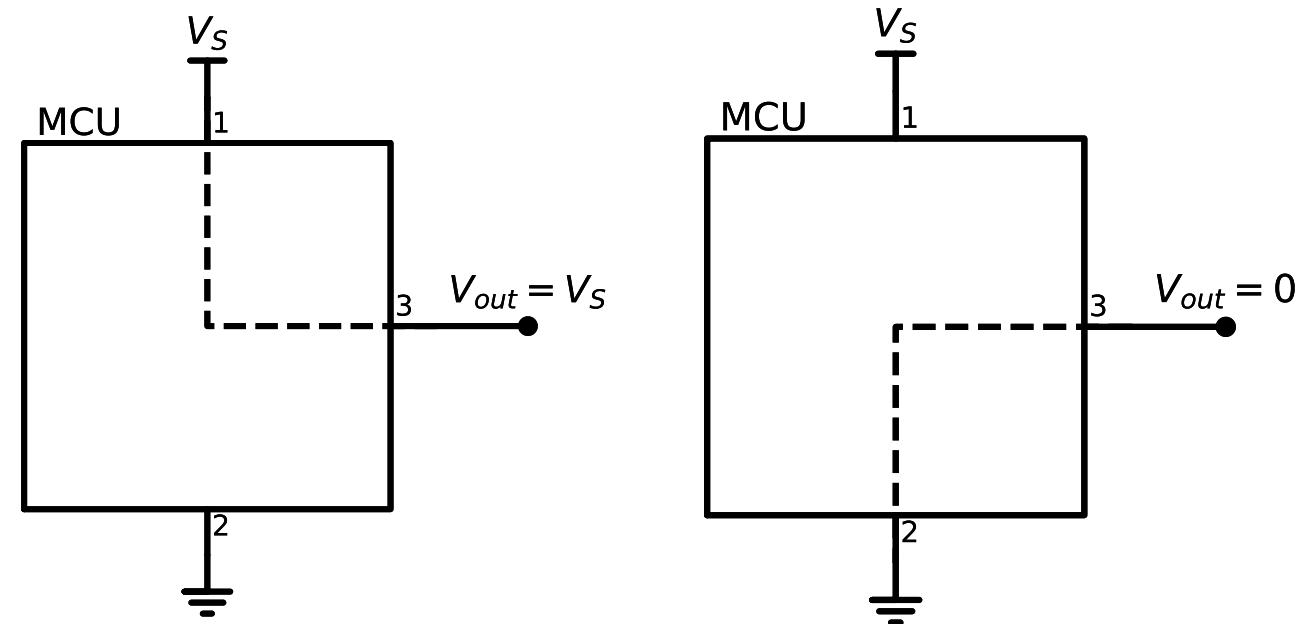


Image source: nRF52832 Datasheet by Nordic Semiconductors

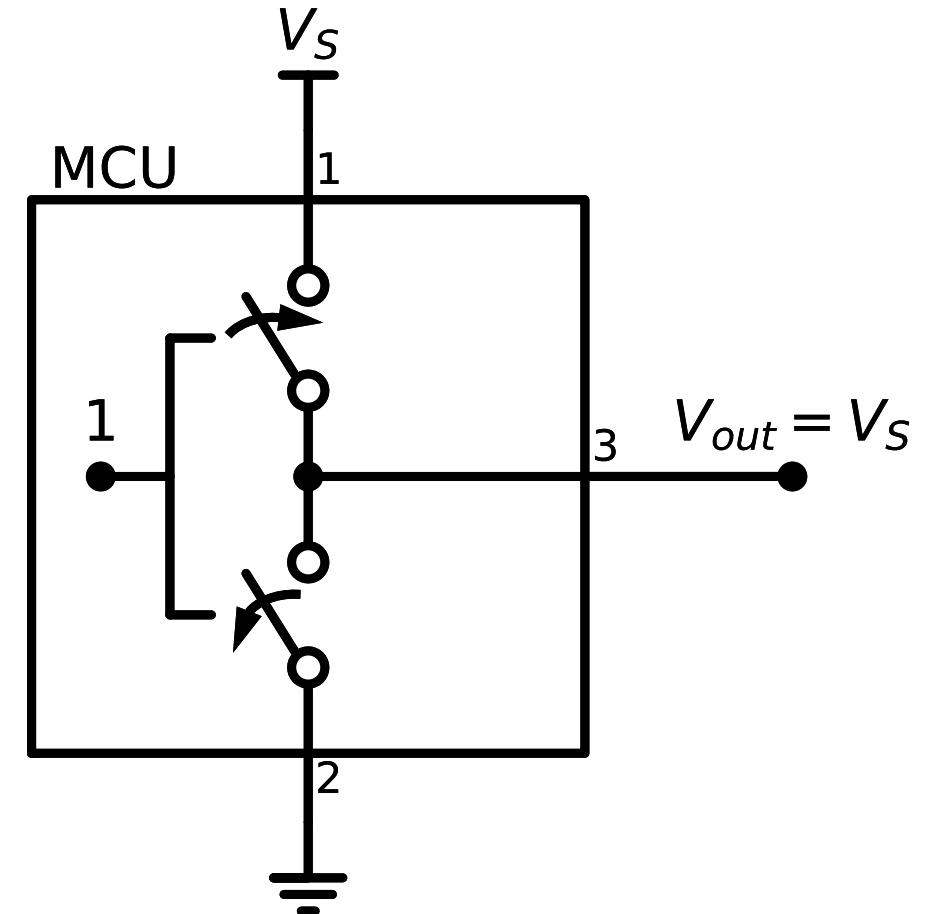
GPIO Output: Push-Pull Mode

- The MCU **drives** the output by actively connecting it to the supply voltage or ground
- When the output voltage is high, the MCU **pushes** the output to V_S
- When the output voltage is low, the MCU **pulls** the output to the ground
- Often push-pull mode is the default output state of the GPIO
- Controlled in **software**



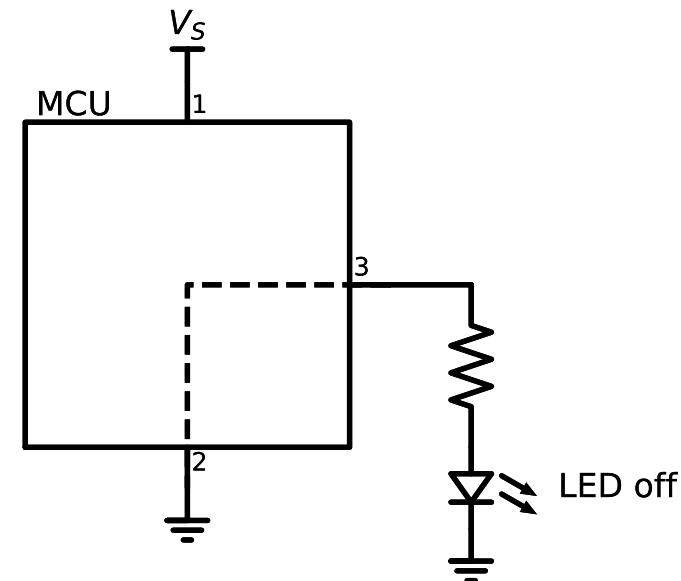
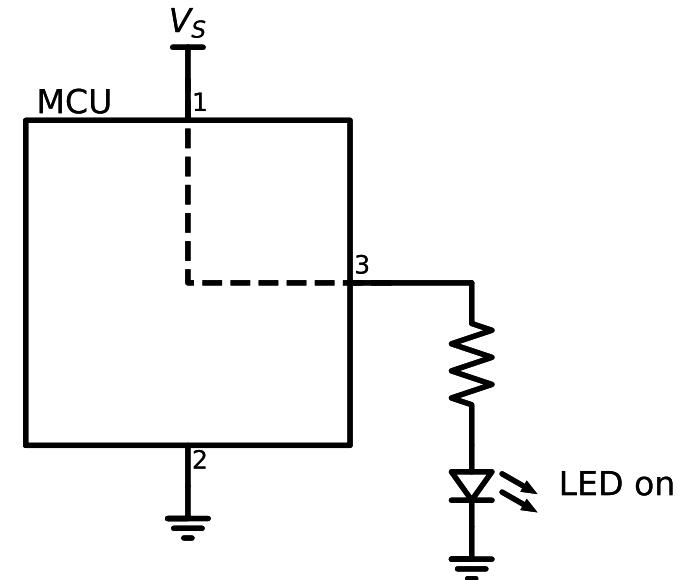
GPIO Output: Push-Pull Mode

- Implemented with MOSFET switches
- Input '1' connects output pin to V_S while simultaneously disconnects it from the ground



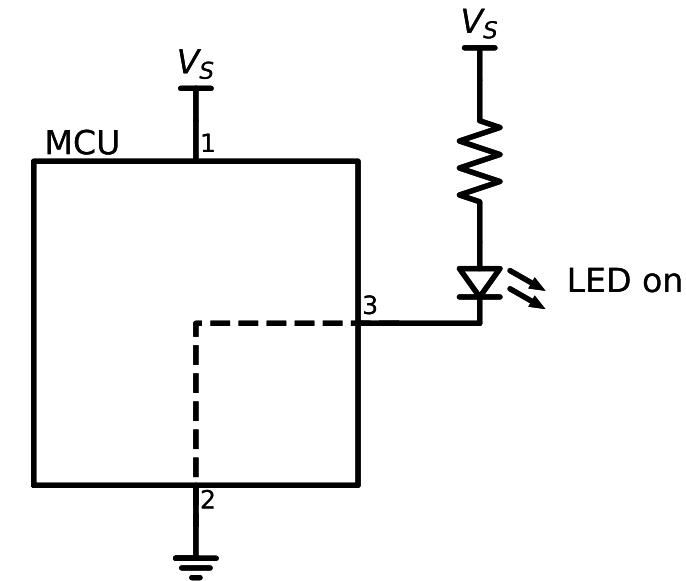
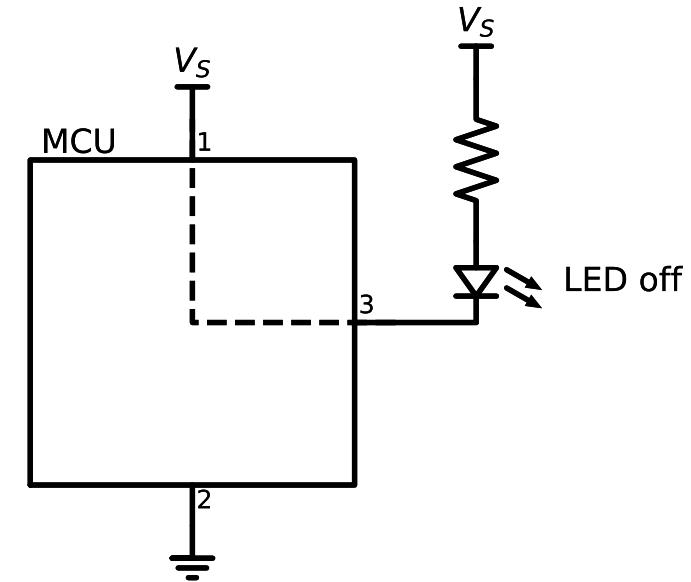
GPIO as a Current Source

- Communication signals are low current signals
- GPIOs can also **source current** to power external components
- MCU can use GPIO as an **on/off switch**
- However, the **output current** of a GPIO is limited (typically, 2-12mA)
- MCU might have some **high drive** GPIOs that can output more current (e.g. 40mA)
- High-power components need to be connected directly to the power source



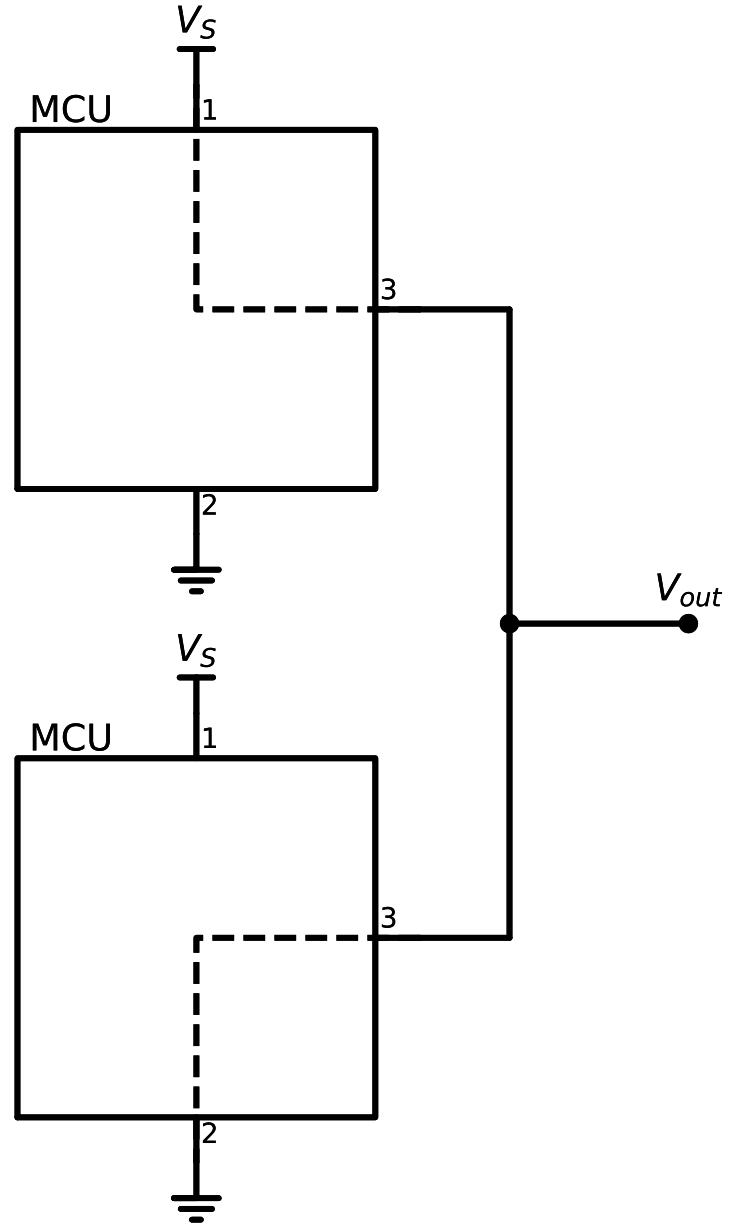
GPIO as a Current Sink

- GPIO can also **sink current**
- Current flows inwards
- Closes the circuit by connecting it to the ground
- Logic is reversed
 - ‘1’ turns off the LED
 - ‘0’ turns on the LED
- Sink current also limited



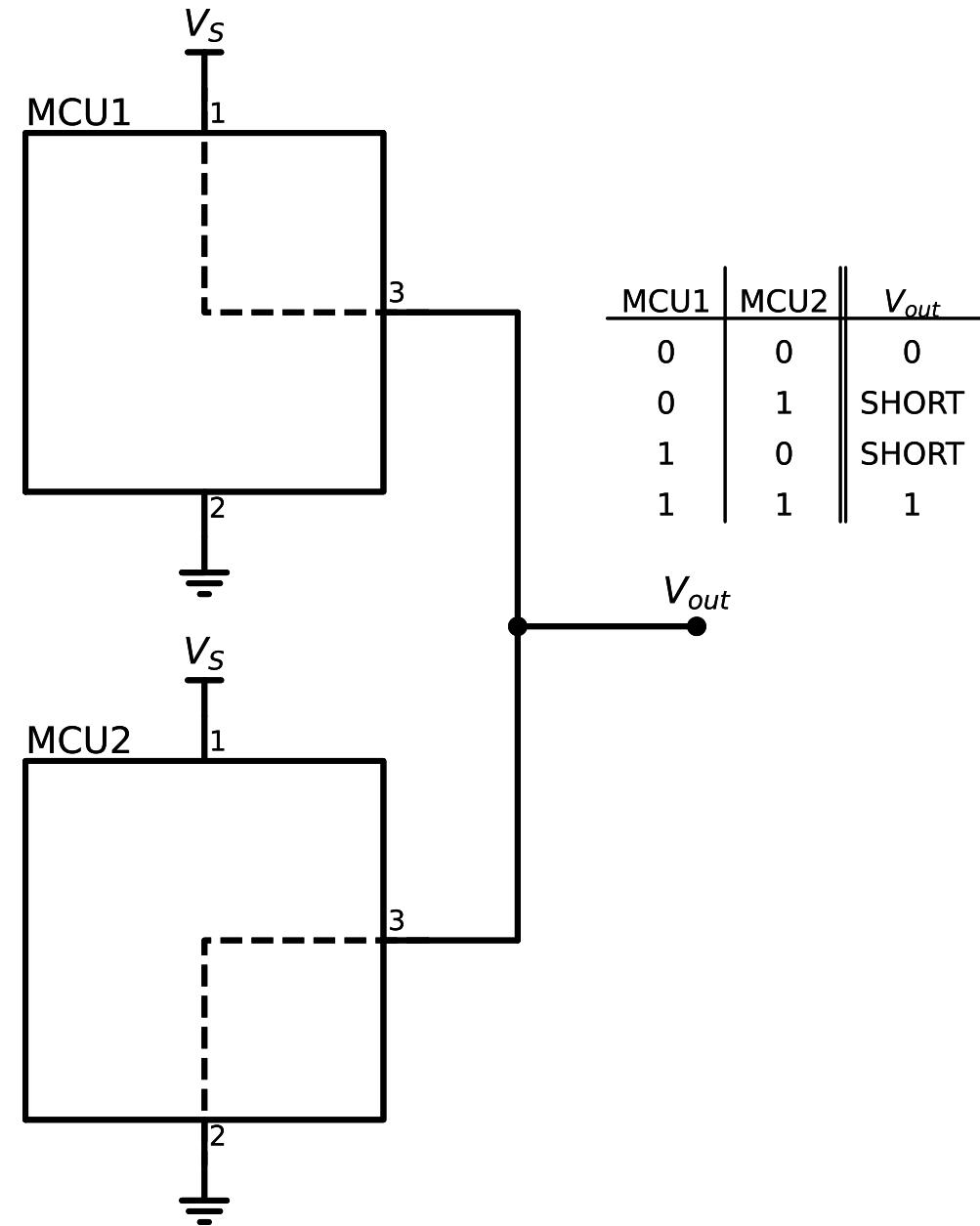
Two MCUs

- Multiple devices may share the same output line
- What happens if two MCUs drive the same output line?



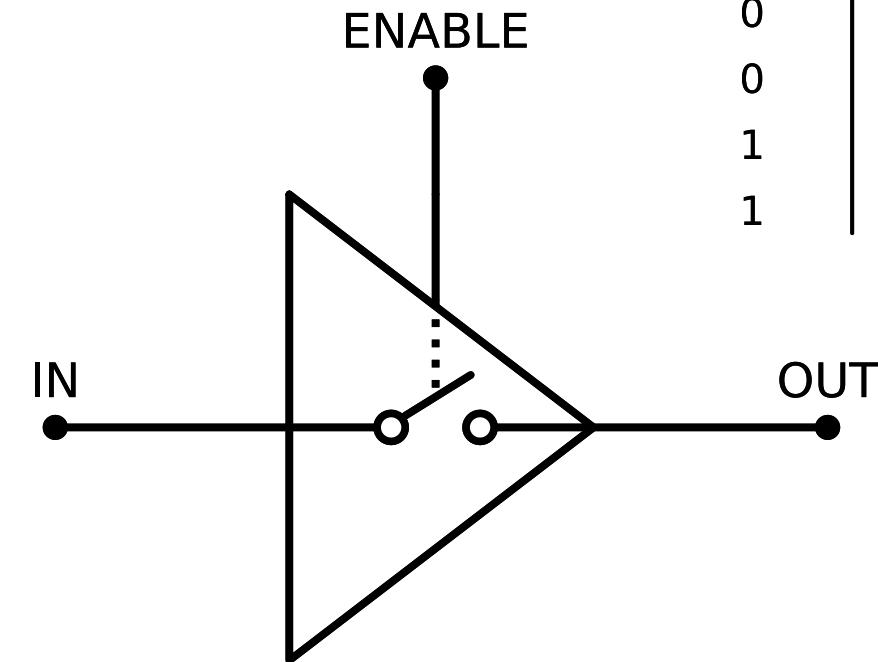
Two MCUs

- Multiple devices may share the same output line
- What happens if two MCUs drive the same output line?
- If one device pushes the output high while the other pulls it low, we have **a short circuit**
- In general, **only one** device should drive a line
- Two output states are not enough, we need to have the option to **disconnect** the output



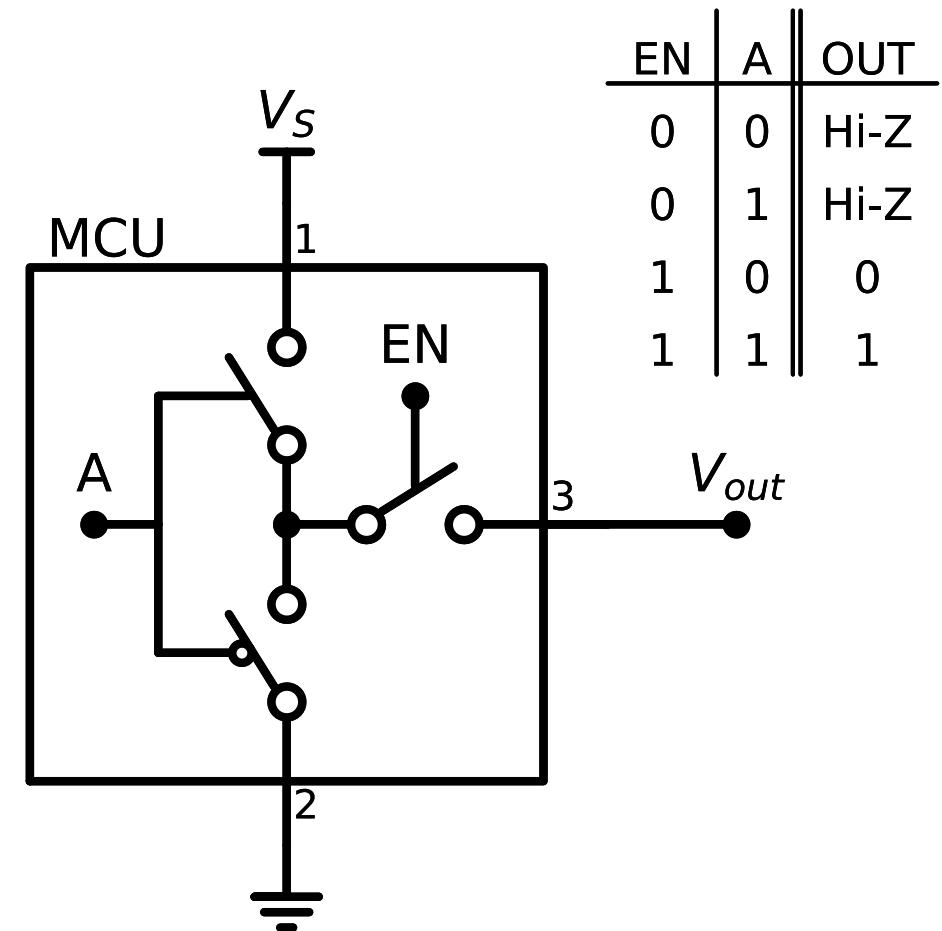
Tri-State Logic

- Three States
 - Logical '0' (i.e. low voltage)
 - Logical '1' (i.e. high voltage)
 - High Impedance or Hi-Z (disconnected)
- Tri-state Buffer
 - Implemented with MOSFET switches
 - At open state the resistance is so high that no current can go through
 - Equivalent to open circuit



GPIO Output States

- With two controls:
 - Output value (A)
 - Enable (EN)
- GPIO can be:
 - Actively pushed to '1'
 - Actively pulled to '0'
 - Disabled (Hi-Z)



GPIO Input and Input Hysteresis

- **Comparator** is a component that compares two (analogue) input signals and outputs '0' or '1' depending which is larger
- A GPIO input can be implemented by comparing the input signal to $V_s / 2$
- Noise close to the threshold creates quick changes successive changes between '0' and '1'
- **Input Hysteresis** mitigates this problem by applying two different thresholds
 - To go from '0' to '1', $V > (V_s / 2) + V_h$
 - To go from '1' to '0', $V < (V_s / 2) - V_h$

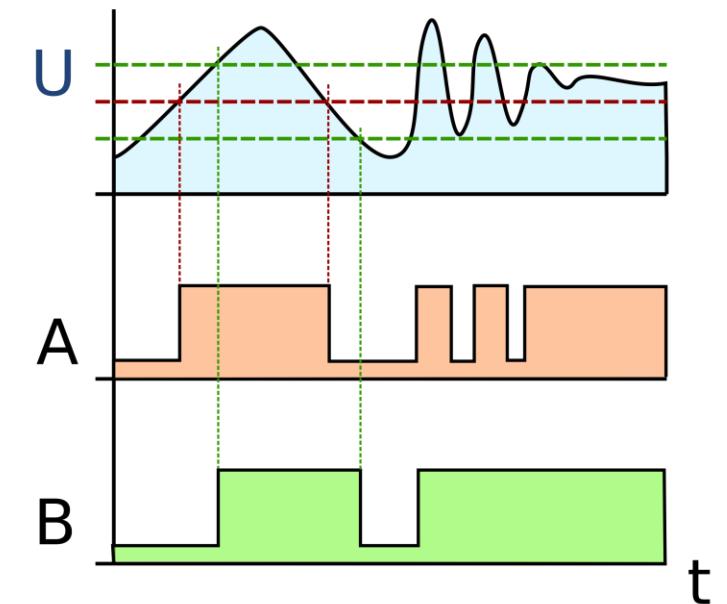
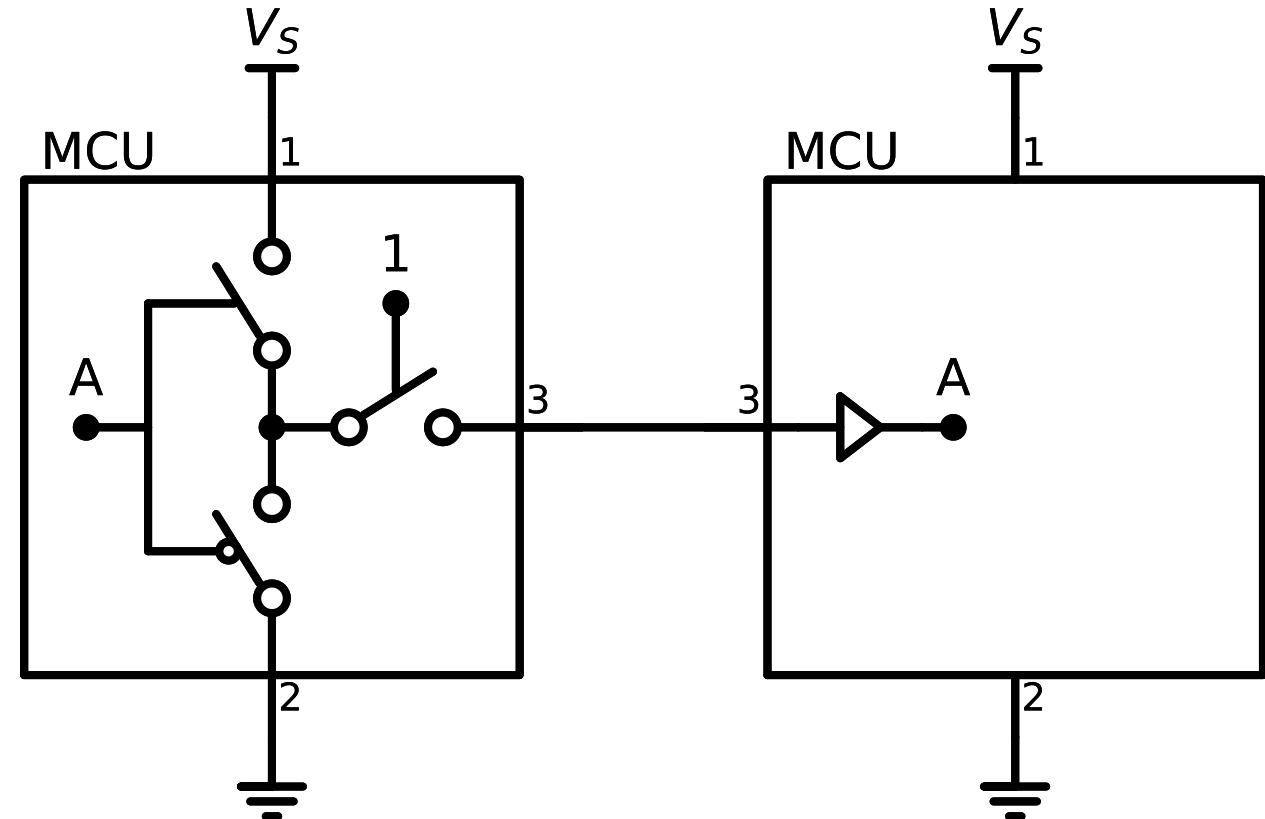


Image source: Wikipedia

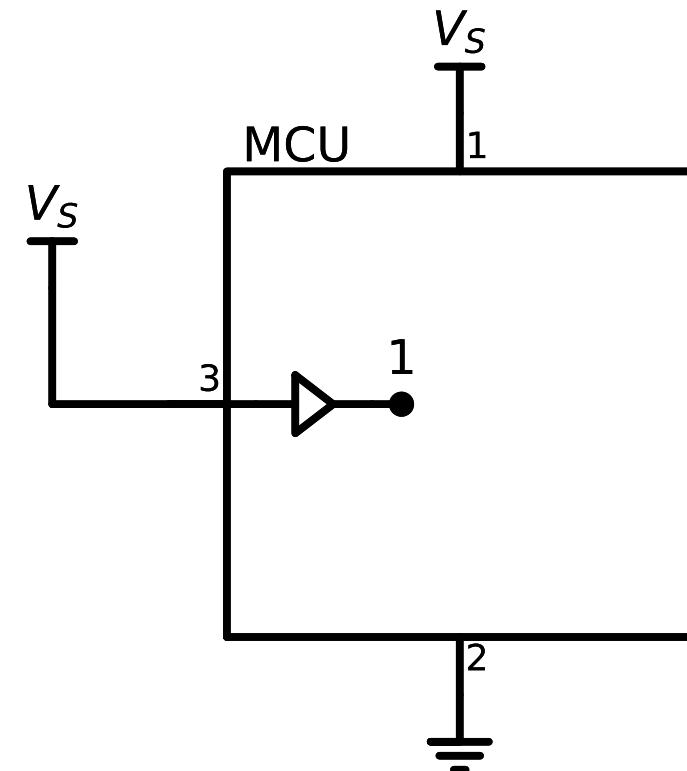
GPIO Input Mode

- When EN=1, the input IN equals A



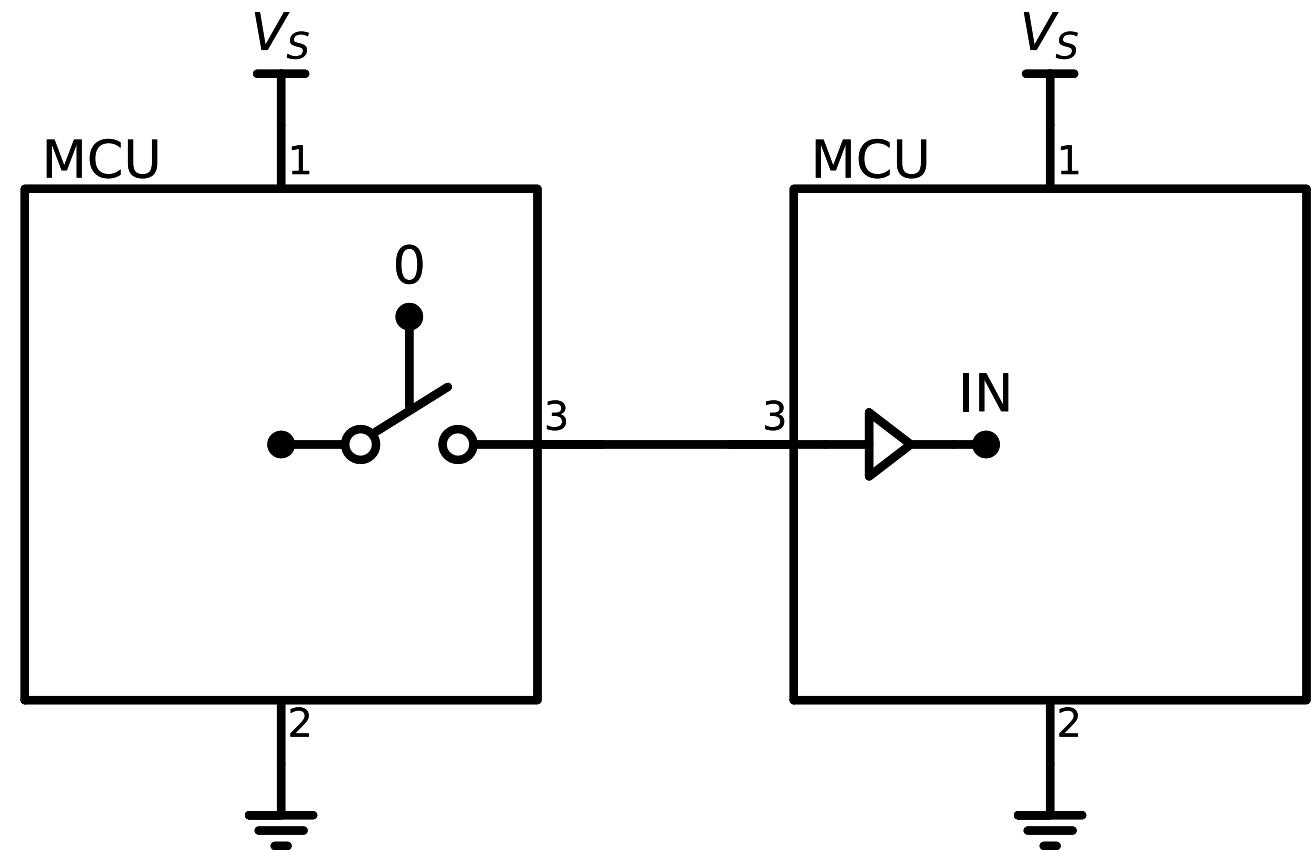
Hardwired Input

- Input can also be hardwired
 - Fixed to have **constant value** in the circuit
- Permanent configuration
 - Assuming input controls some configuration
- As a form of ID
 - Consider a circuit with two MCUs running the same software, but one has Pin3 hardwired to '0' and the other has Pin3 hardwired to '1'
 - Reading Pin3 is a form of self-identification



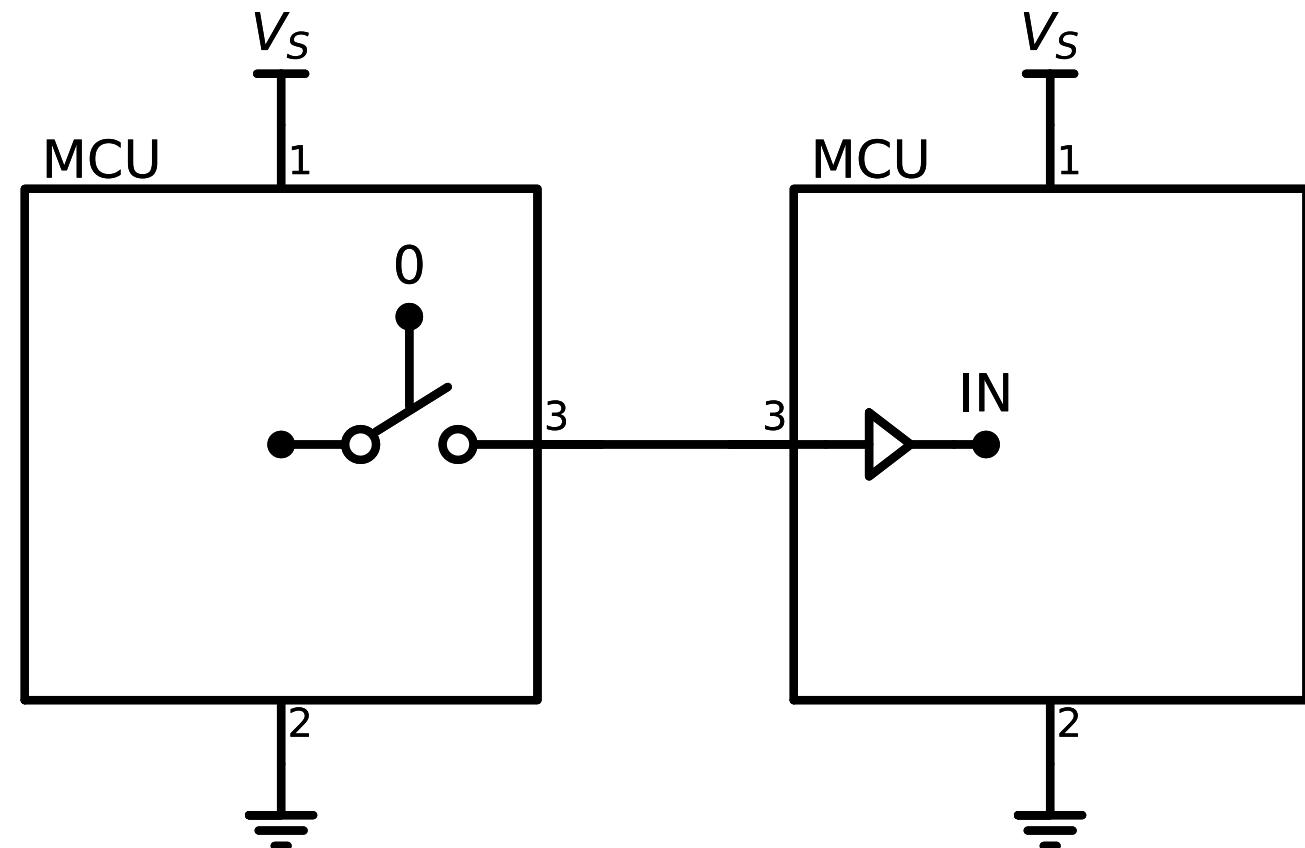
Reading a Hi-Z output

- What value will we get if we try to read a disabled output?



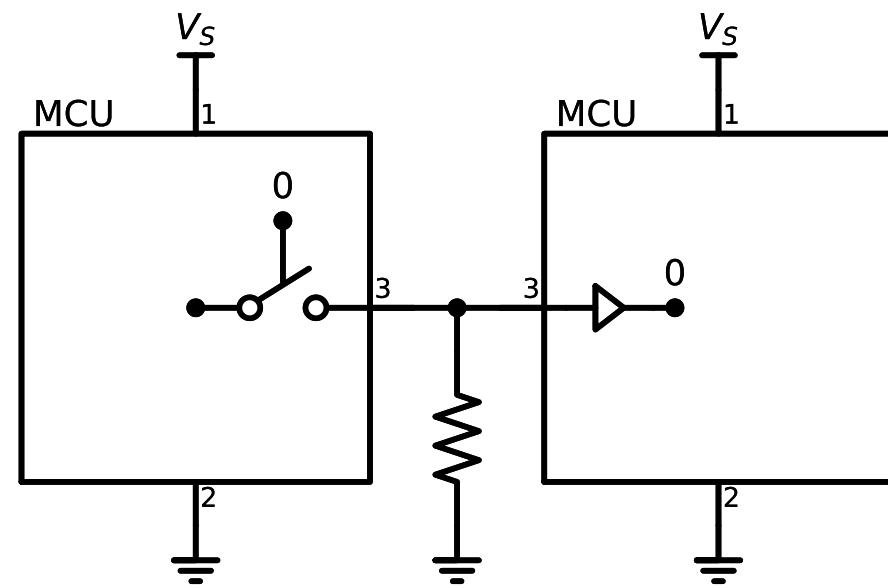
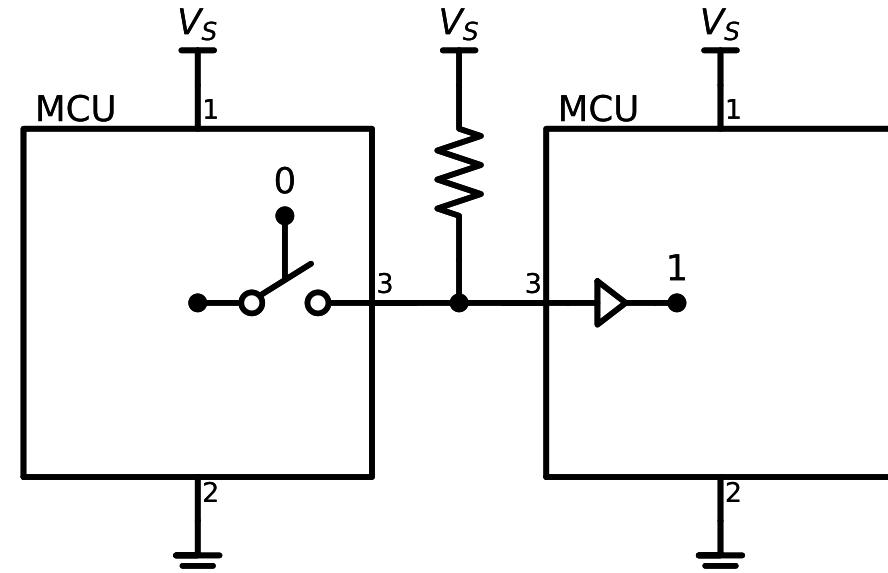
Reading a Hi-Z output

- What value will we get if we try to read a disabled output?
- The input is undefined
 - It can be 0 or 1 or can switch between them
- It essentially acts as a tiny antenna, capturing noise from its surroundings
- A Hi-Z pin is **floating**



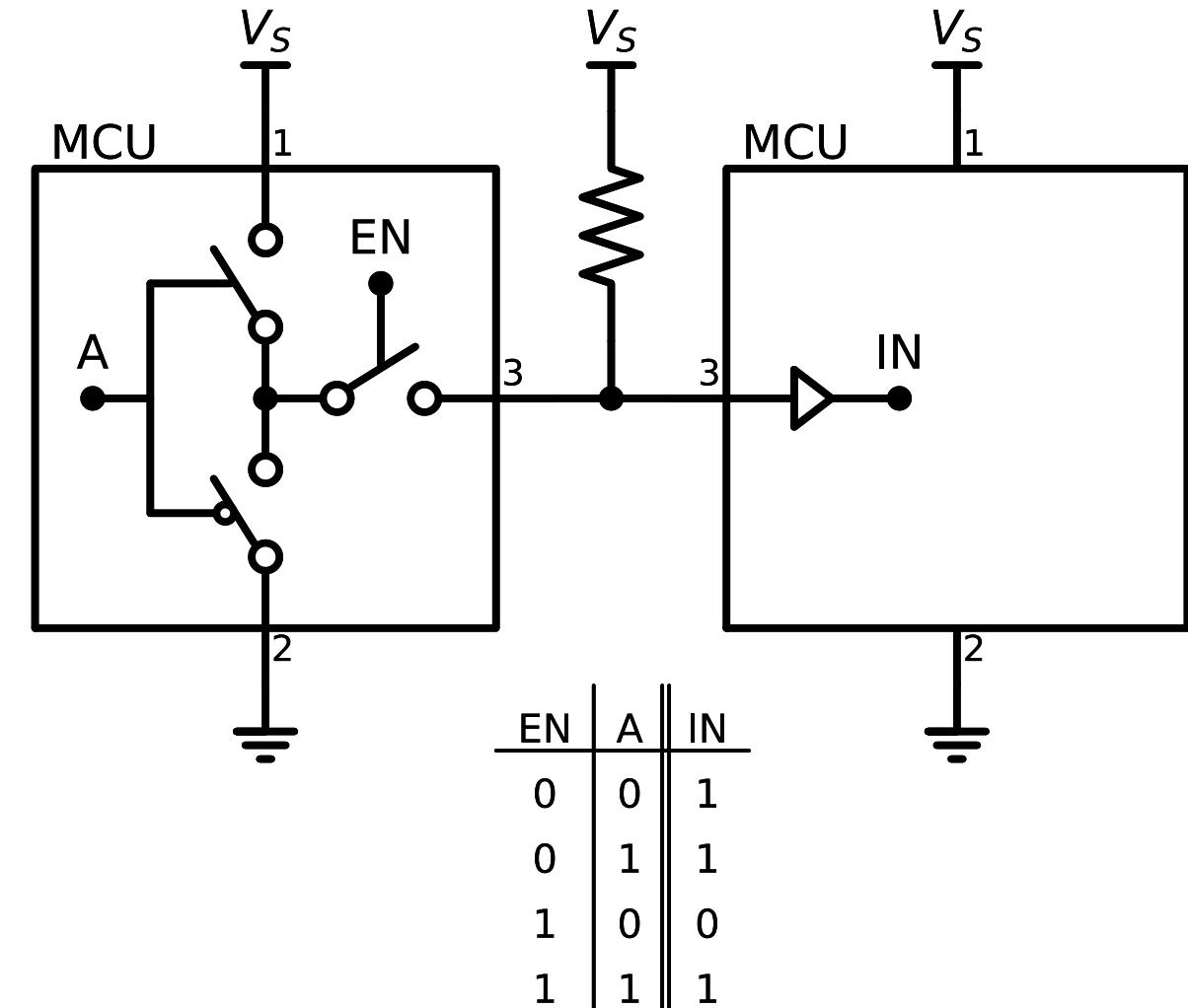
Pullup and Pulldown Resistors

- They provide **a default state**
- If nobody actively drives the line
 - The pullup resistor will raise the line to '1'
- If the line is actively pulled to '1'
 - No voltage difference across the pullup resistor (no current), line is '1'
- If the line is actively pushed to '0'
 - There is voltage difference across the pullup resistor (current goes through), line is '0'
- Similarly, the pulldown resistor defaults to '0'
- MCU typically supply internal pullup/pushdown resistors



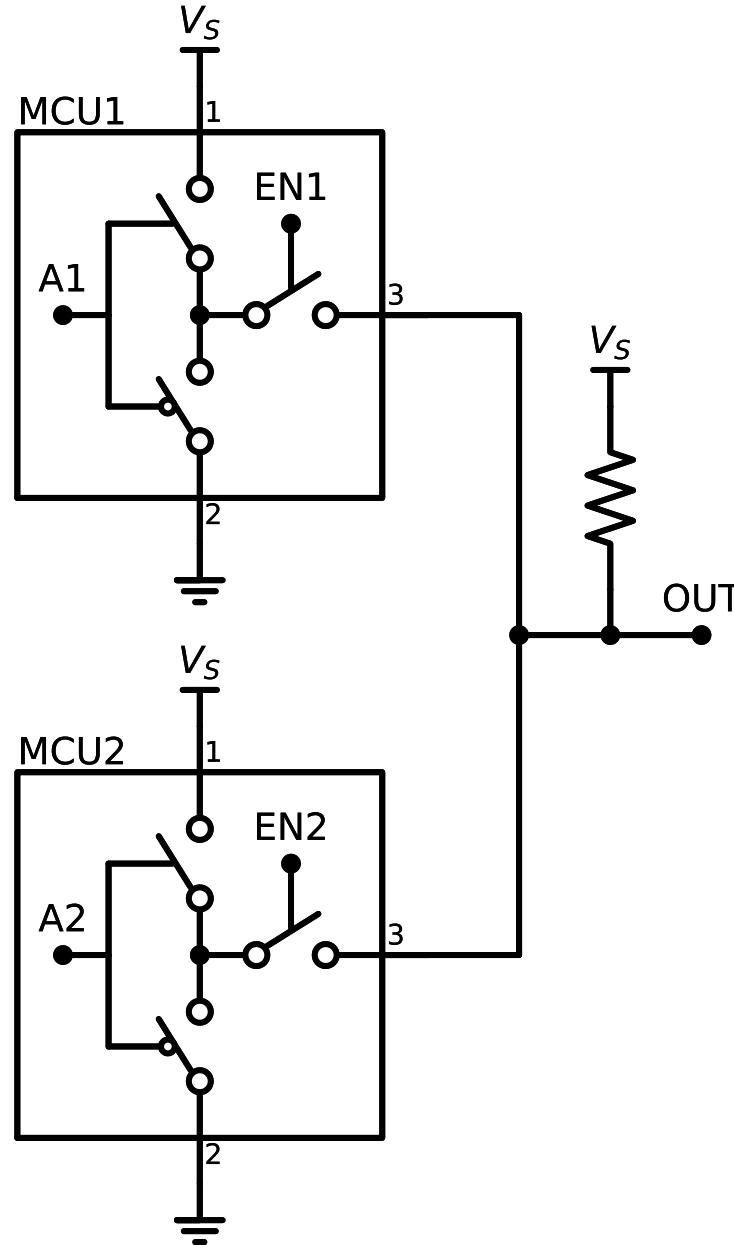
GPIO Output with Pullup

- If output is disabled ($EN='0'$)
 - Pullup resistor determines the input ($IN='1'$)
- If output is enabled ($EN='1'$)
 - Output determines the input ($IN=A$)



Multiple MCUs

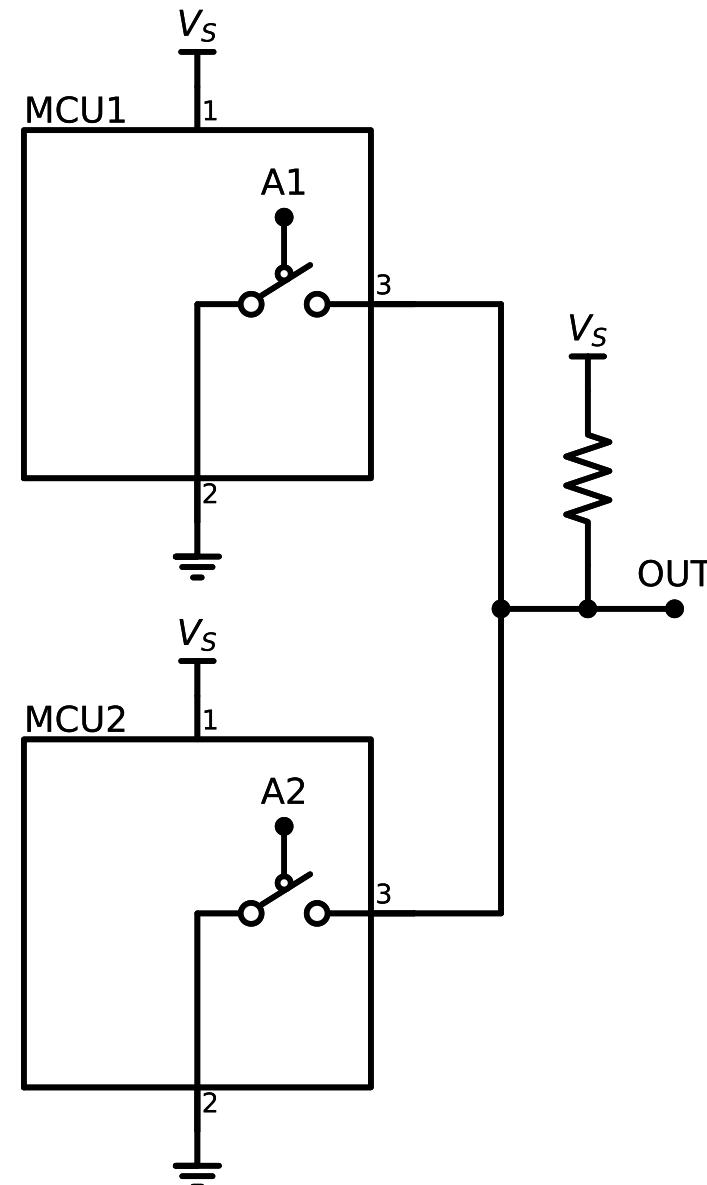
- OK if one output is enabled at a time
- Short circuits are still possible with push-pull outputs
- Is this safe enough?



A1	EN1	A2	EN2	OUT
0	0	0	0	0 (pullup)
0	0	0	1	0 (MCU2)
0	0	1	0	1 (pullup)
0	0	1	1	1 (MCU2)
0	1	0	0	0 (MCU1)
0	1	0	1	0 (both)
0	1	1	0	0 (MCU1)
0	1	1	1	1 (MCU1)
1	0	0	0	0 (pullup)
1	0	0	1	0 (MCU2)
1	0	1	0	1 (pullup)
1	0	1	1	1 (MCU2)
1	1	0	0	1 (MCU1)
1	1	0	1	1 (MCU1)
1	1	1	0	0 (MCU1)
1	1	1	1	1 (both)

GPIO Output: Open-Drain Mode

- Open-Drain mode has two states:
 - Logical '0' (i.e. low voltage)
 - Hi-Z (i.e. disconnected)
- Sets the line to '0' **actively**
- Sets the line to '1' **indirectly** by disconnecting the output and letting the pullup resistor raise it to '1'
- The pullup resistor can also be internal
- Open-Drain is used when multiple devices share the same line (safe from shorts)



A1	A2	OUT
0	0	0 (both)
0	1	0 (MCU1)
1	0	0 (MCU2)
1	1	1 (pullup)

Analog-to-Digital Converter (ADC)

- An ADC converts an analogue signal (voltage signal) into a digital signal (series of numbers)
- An ADC essentially measures the voltage of the input and maps it to a number
- The Bit Resolution (M) defines the number of discrete values the ADC can produce
- E.g.: A 12-bit ADC maps the input to a number in $[0, 4095]$, i.e., 2^{12} values

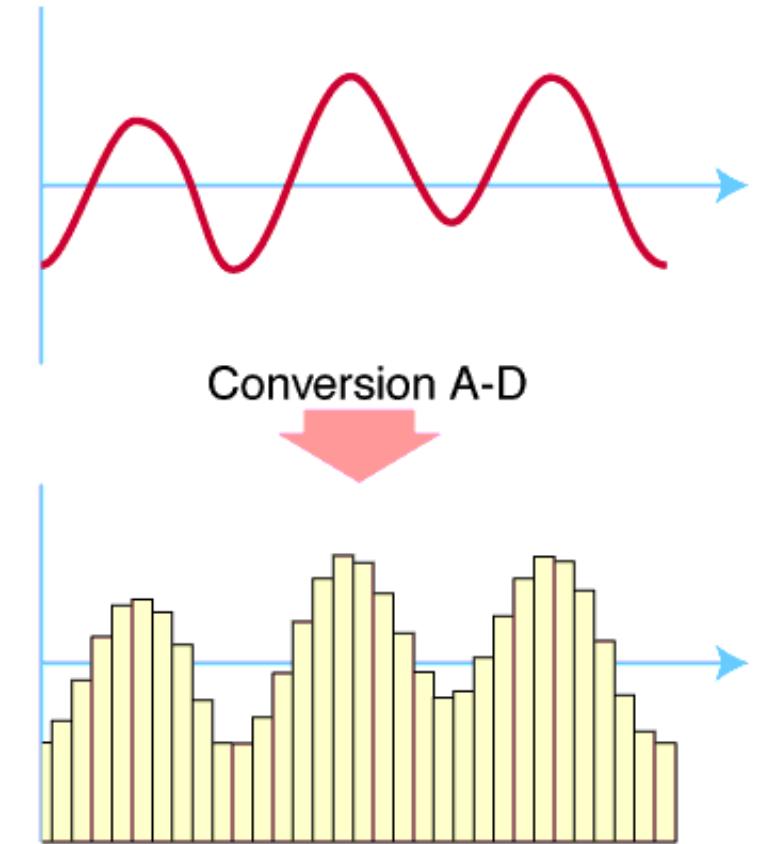
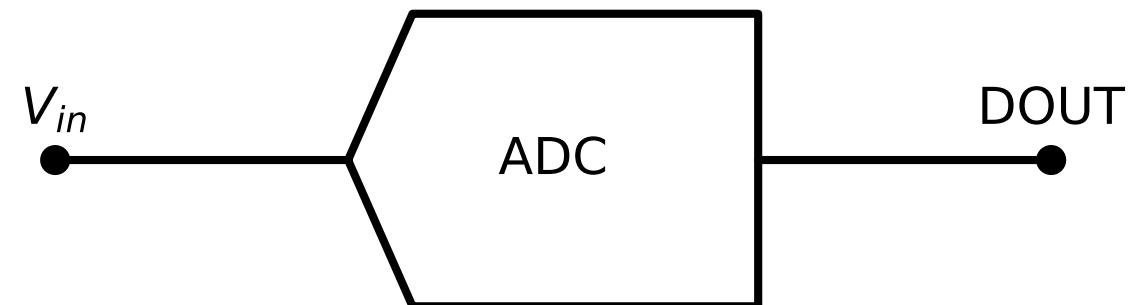


Image source: Wikipedia

Single-Ended ADC

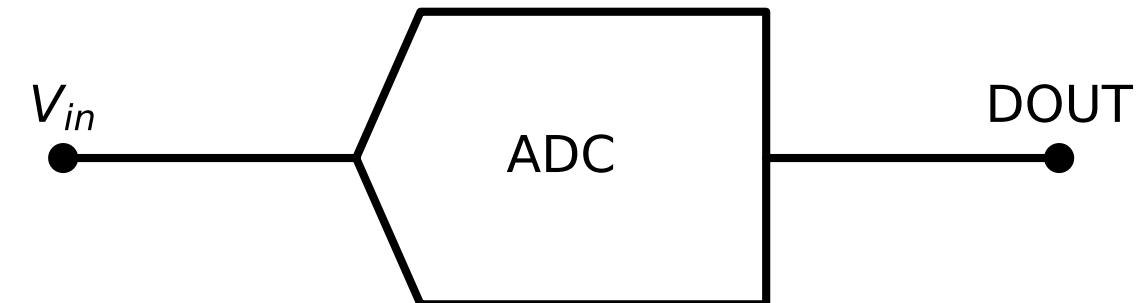
- Measures input voltage (V_{in}) referenced to the ground
 - Input and ADC have common ground
- The reference voltage (V_{ref}) defines the maximum value that can be measured
 - V_{ref} can be equal to the supply voltage or less
 - Voltages in $[0, V_{ref}]$ are linearly mapped in $[0, 2^M-1]$
 - If $V_{in} > V_{ref}$, $DOUT=2^M-1$ (saturation)
- Examples:
 - A 12-bit ADC with $V_{ref}=5V$ will map:
 - $V_{in}=0V$ to 0
 - $V_{in}=2.5V$ to 2048
 - $V_{in}=5V$ to 4095
 - $V_{in}=6V$ to 4095



$$DOUT \approx \frac{V_{in}}{V_{ref}} \cdot 2^M$$

Single-Ended ADC

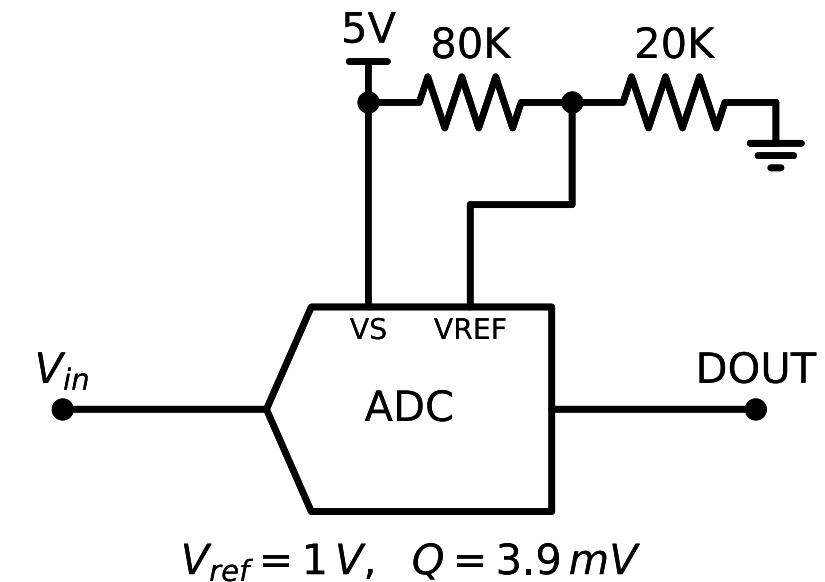
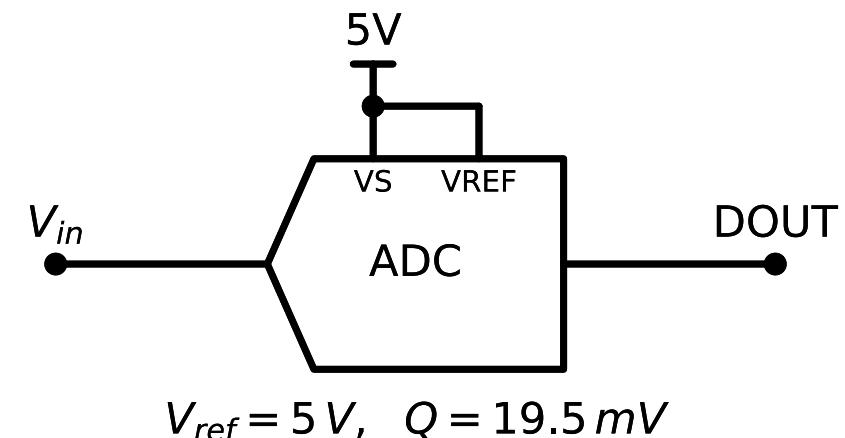
- Quantisation error
 - The continuous analogue values are mapped to the closest integer
 - The reference voltage (V_{ref}) together with the bit resolution (M) define the voltage resolution (Q)
 - In other words, measurements are in steps or quanta of $Q = V_{ref} / 2^M$
- Examples:
 - A 12-bit ADC with $V_{ref}=5$ V, $Q= 1.22$ mV
 - A 12-bit ADC with $V_{ref}=2.5$ V, $Q= 0.6$ mV
 - A 8-bit ADC with $V_{ref}=5$ V, $Q= 19.5$ mV
 - A 8-bit ADC with $V_{ref}=2.5$ V, $Q= 9.7$ mV
- Trade-off: range vs resolution



$$DOUT \approx \frac{V_{in}}{V_{ref}} \cdot 2^M$$

ADC: Range vs Resolution

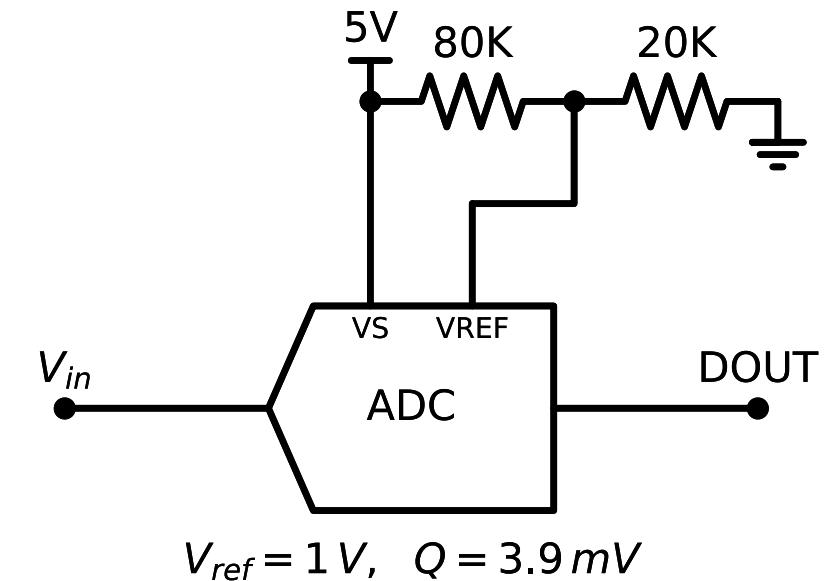
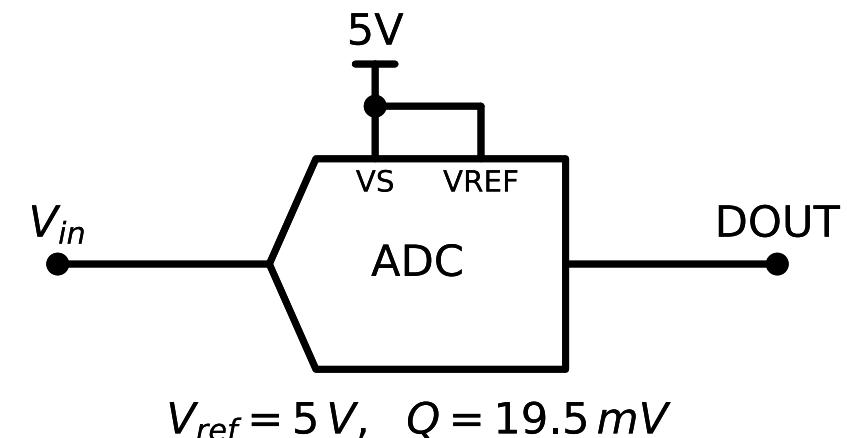
- Assuming an 8-bit ADC and supply voltage 5V
- If I want to measure a signal in [0, 5] Volts
 - I can do it with a 19.5 mV resolution
- If I want to measure a smaller signal in [0,1] Volts with more subtle variations
 - With $V_{ref}=5V$, 19.5 mV resolution may not be enough and DOUT values [52,255] will never be used!
 - Instead with $V_{ref}=1V$, I have 3.9 mV resolution and I use all DOUT values [0,255]



ADC: Range vs Resolution

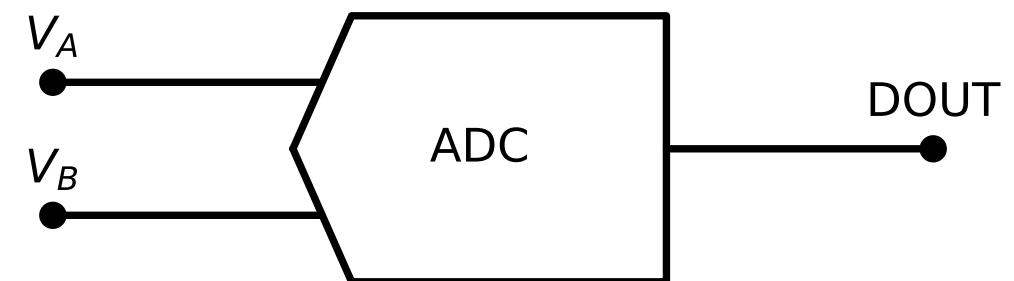
- Assuming an 8-bit ADC and supply voltage 5V
- If I want to measure a signal in [0, 5] Volts
 - I can do it with a 19.5 mV resolution
- If I want to measure a smaller signal in [0,1] Volts with more subtle variations
 - With $V_{ref}=5V$, 19.5 mV resolution may not be enough and DOUT values [52,255] will never be used!
 - Instead with $V_{ref}=1V$, I have 3.9 mV resolution and I use all DOUT values [0,255]

How can I measure a signal in [4,5] Volts with resolution 3.9mV?



Differential ADC

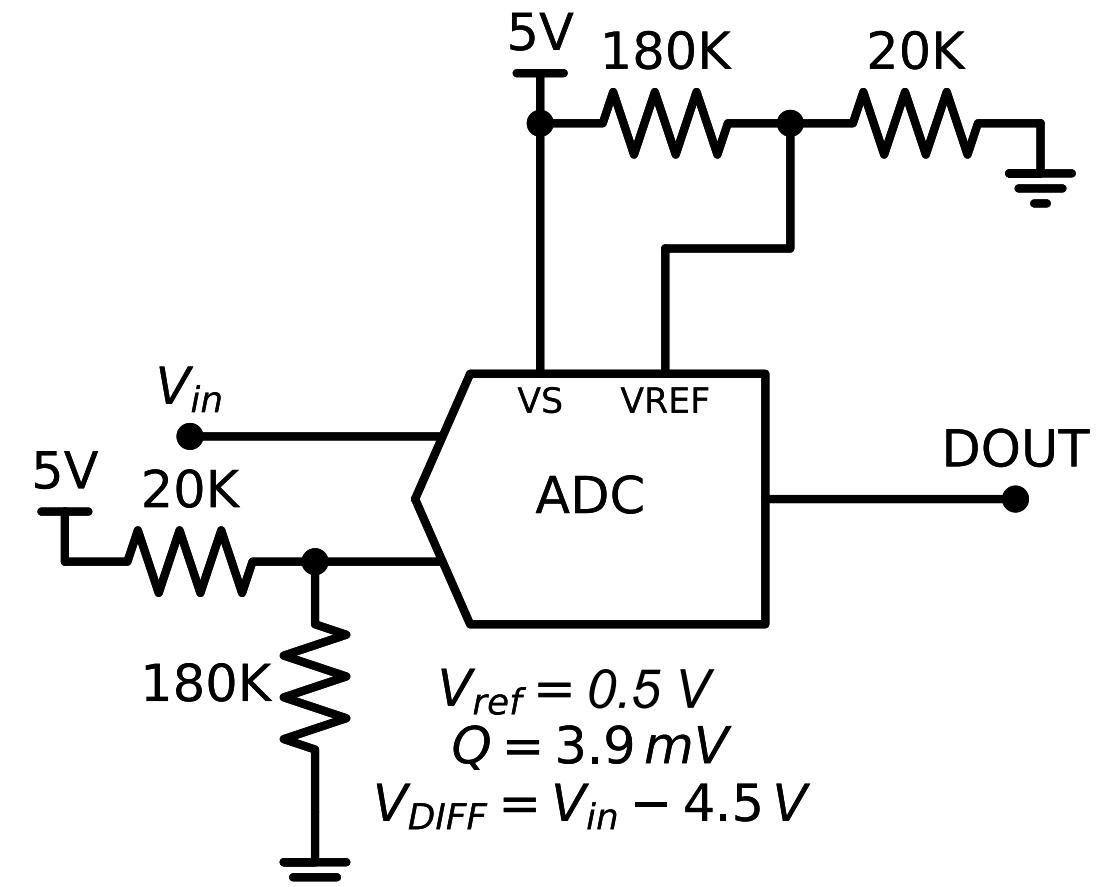
- Measures the voltage difference between two input signals ($V_A - V_B$)
 - A differential voltage can be positive or negative depending which signal is higher
 - Better at rejecting noise
- The reference voltage (V_{ref}) defines the maximum value that can be measured
 - V_{ref} can be equal to the supply voltage or less
 - Voltages in $[-V_{ref}, V_{ref}]$ are linearly mapped in $[-2^{M-1}, 2^{M-1}-1]$
 - If $(V_A - V_B) > V_{ref}$, $DOUT = 2^{M-1}-1$ (saturation)
 - If $(V_A - V_B) < -V_{ref}$, $DOUT = -2^{M-1}$ (saturation)
 - The resolution = $Q = V_{ref} / 2^{M-1}$
- Examples:
 - A 12-bit ADC with $V_{ref}=5V$ will map:
 - $V_A=0V$ and $V_B=5V$ to -2048
 - $V_A=5V$ and $V_B=0V$ to 2047
 - $V_A=5V$ and $V_B=5V$ to 0
 - $V_A=3V$ and $V_B=2V$ to 410



$$DOUT \approx \frac{V_A - V_B}{V_{ref}} \cdot 2^{M-1}$$

ADC: Removing a DC Offset

- Measure a small single-ended signal with a large DC-offset
- Assuming an 8-bit ADC and supply voltage 5V
- I want to measure a signal in [4, 5] Volts with 3.9mV resolution
- Apply to the negative input a fixed 4.5V signal using a voltage divider
 - The differential signal ($V_{DIFF} = V_{in} - 4.5V$) is in [-0.5, 0.5] Volts
- With $V_{ref}=0.5V$, I can measure the differential signal with $Q = V_{ref}/128 = 3.9 \text{ mV}$ resolution



ADC Channels

- MCUs typically have internal ADCs
- Typically, the input(s) of the ADC can be internally connected to multiple external pins
- Example: the nRF2832 has 8 input channels
 - Labelled as AIN0-AIN7 in the figure
- Only one channel can be used at a time

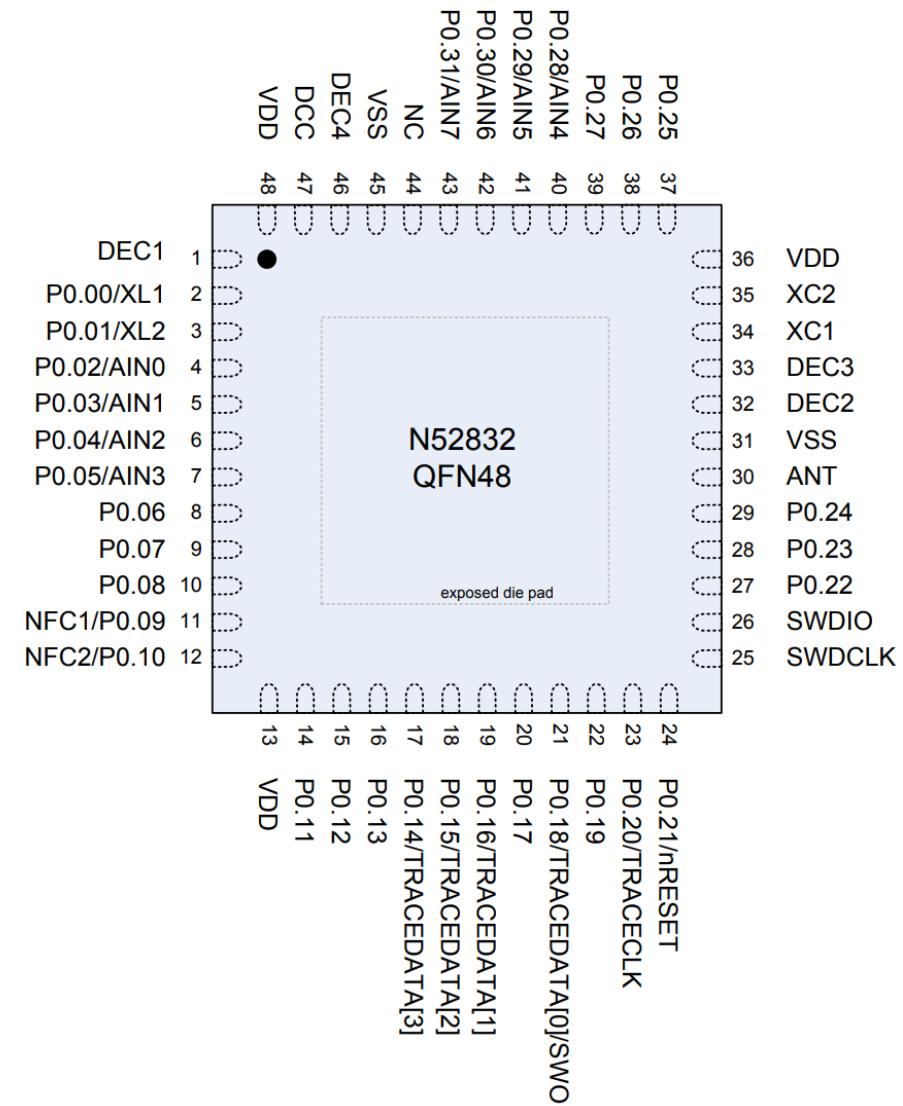


Image source: nRF52832 Datasheet by Nordic Semiconductors

ADC Modes of operation

- **One-shot mode:** a single sample is taken
- **Continuous mode:** samples are taken at a specified sampling frequency
 - Nyquist Theorem: to accurately represent a signal the sampling frequency should be double the highest frequency component of the input signal
- **Oversampling mode:** samples are taken at a higher than Nyquist frequency and then averaged out to reduce noise
- **Scan mode:** cycles through multiple input channels to measure multiple inputs in “parallel”

Digital-to-Analog Converter (DAC)

- A DAC converts a digital signal (series of numbers) into an analogue signal (voltage signal)
- An DAC essentially continuous signal out of discreet time data
- Applications
 - Audio
 - Video
 - Communications (wireless/optical)
 - Digitally-controlled potentiometer
 - ...

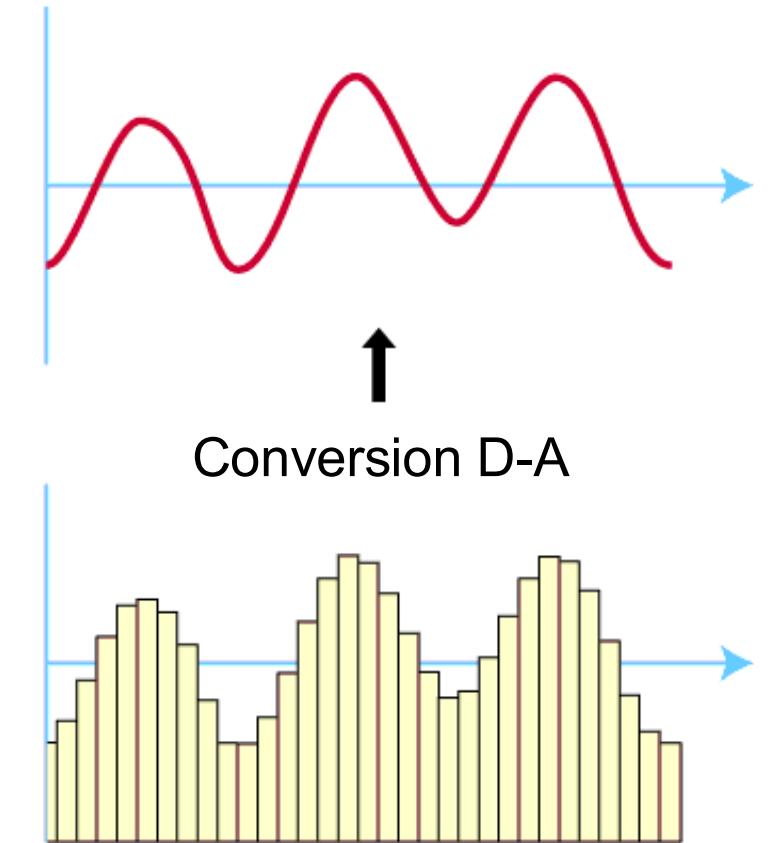
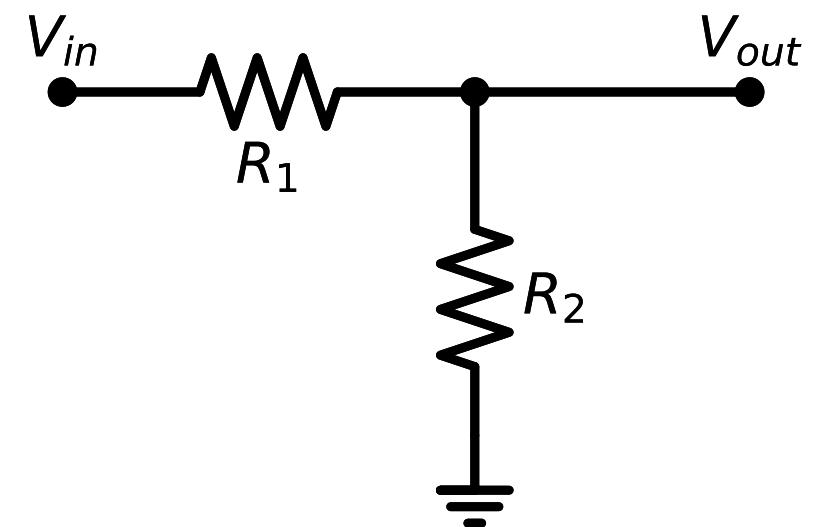


Image source: Wikipedia

Power Control

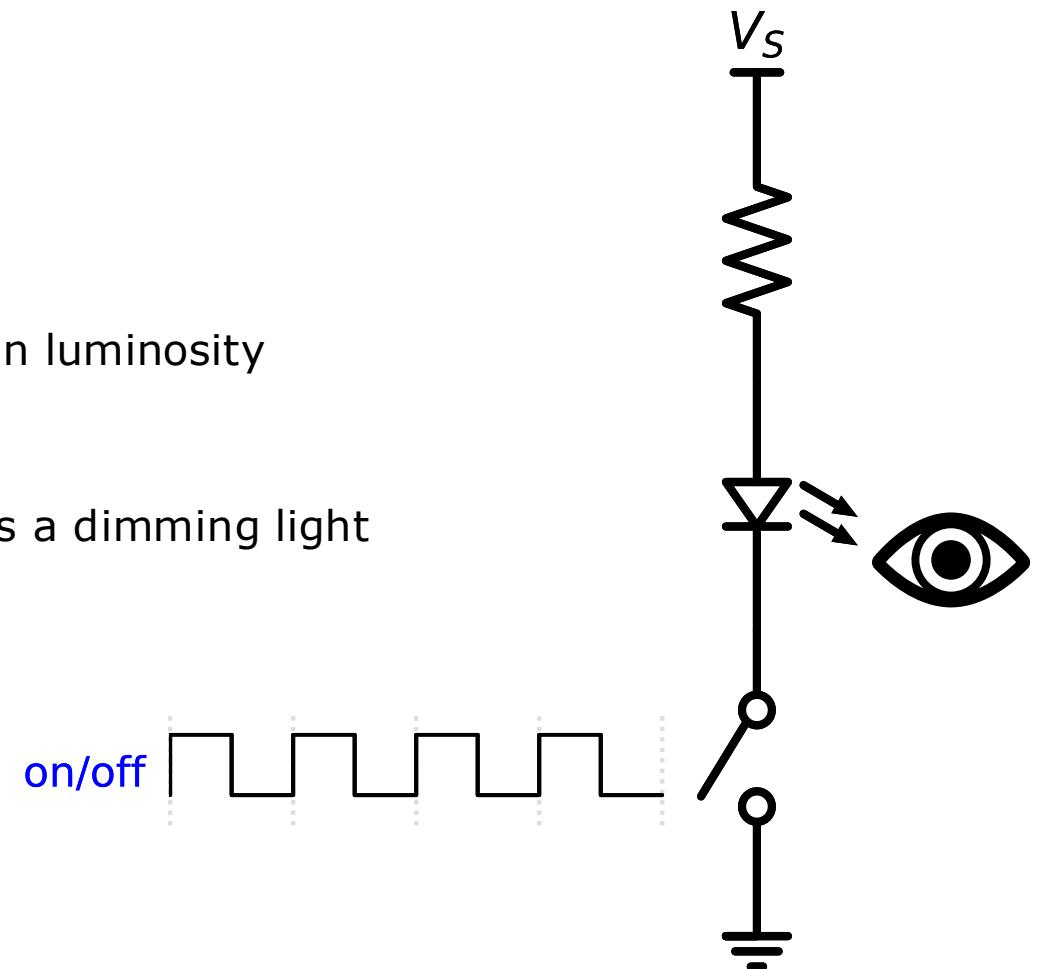
- Imagine R_2 is the load
- Imagine R_1 is a variable resistor that we use to control the power that goes to the load
- If $R_1 \ll R_2$, R_1 voltage drop negligible, power consumption of R_1 also negligible
- If $R_1 \gg R_2$, R_1 current negligible, power consumption of R_1 also negligible
- If $R_1 == R_2$, R_1 consumes as much as power as the load

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2}$$



Dimming a LED

- Turn LED on/off at a high frequency
- Human eye is not able to capture frequent changes in luminosity
 - Has inertia or short-term memory
- Instead, it averages out the luminosity and perceives a dimming light
 - This is a low pass filter!



Pulse Width Modulation (PWM)

- An on/off signal with two properties
 - Frequency: repetitions per second
 - Duty cycle: on time over period
- MCUs can generate PWM signals using the GPIOs

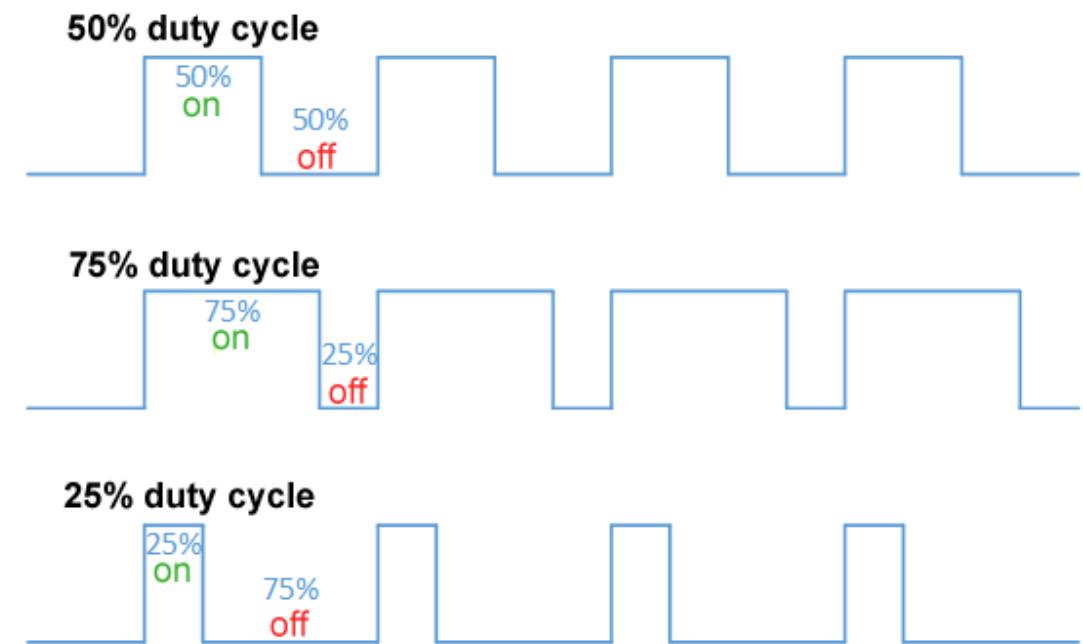


Image source: Wikipedia

PWM Motor Control

- The PWM duty cycle controls the power that goes to the motor
- Motors are inductive
 - They react slowly on changes in current
 - Current (and thus power) through is averaged out
- Why not power control with variable series resistor (transistor)?
 - Inefficient as resistor would consume energy
 - Less power would go to the motor
 - It is easier/cheaper to generate digital signals (PWM) than analogue signals

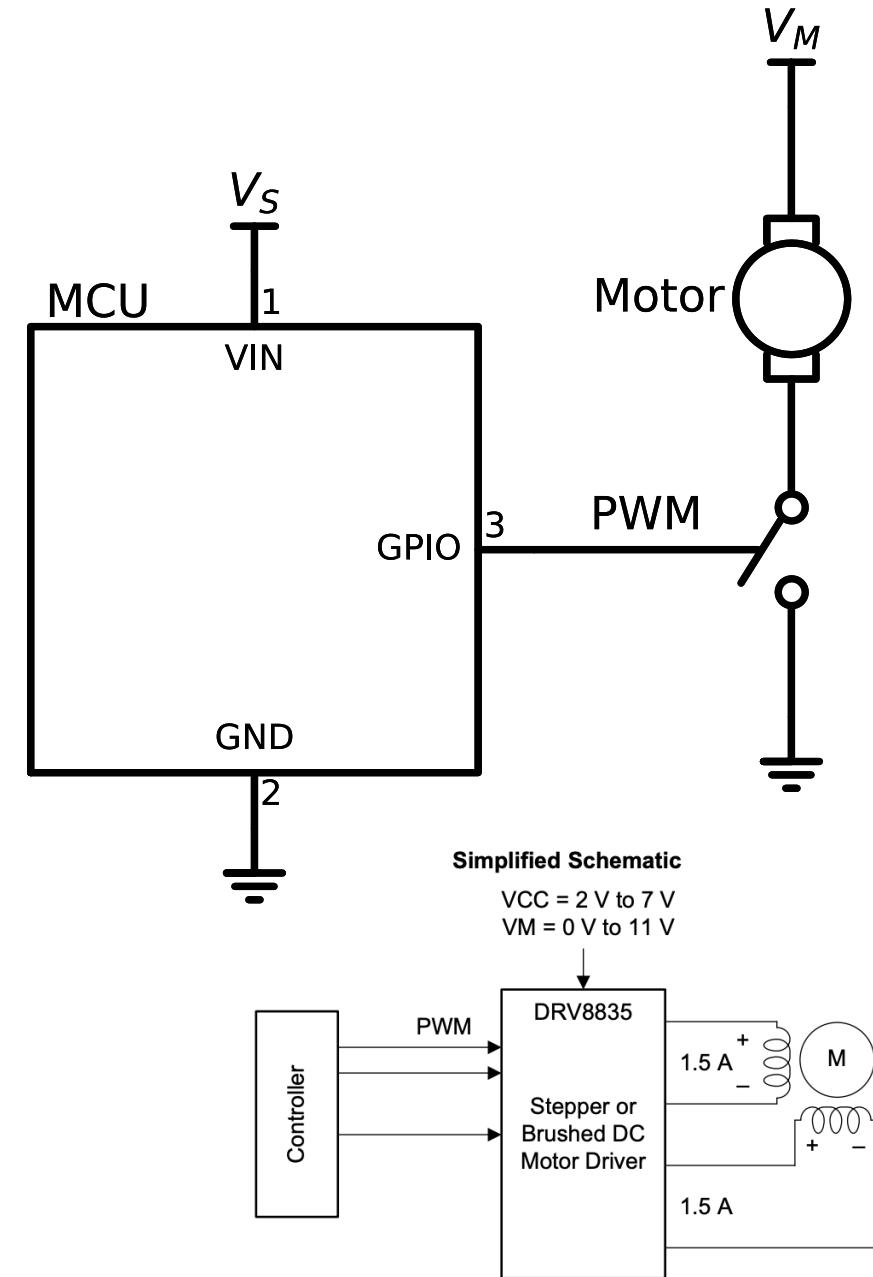
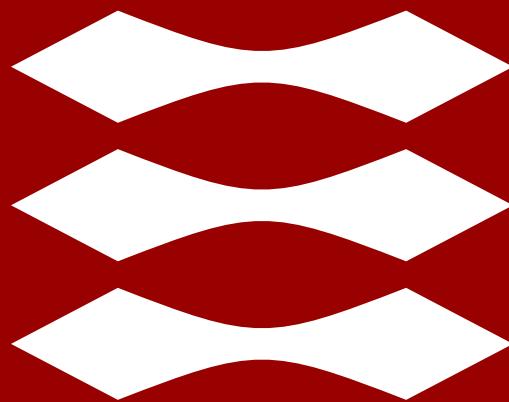


Image source: DRV8835 Datasheet from Texas Instruments

DTU



Networked Embedded Systems

Week 4: Embedded Software

Charalampos (Haris) Orfanidis

Assistant Professor

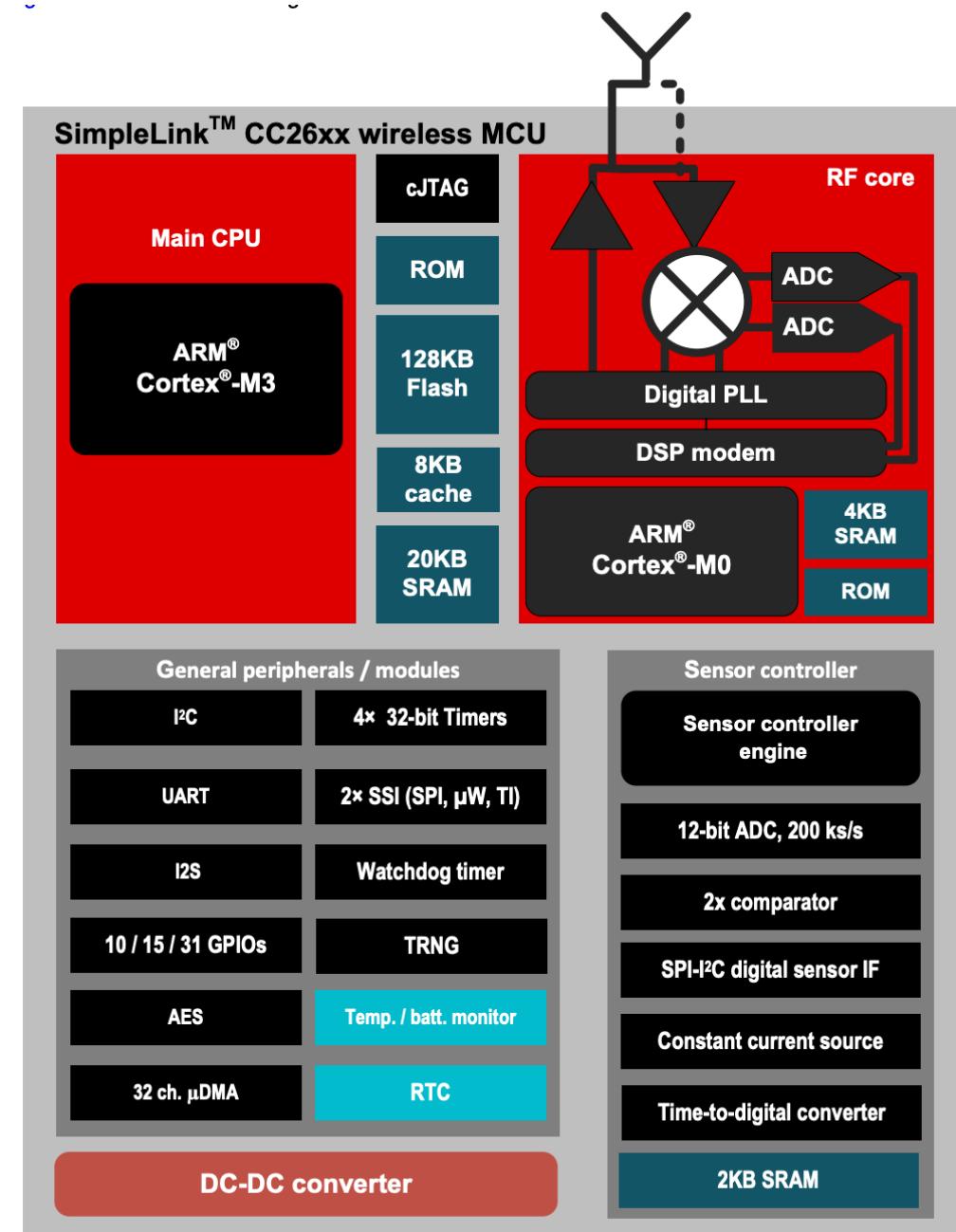
chaorf@dtu.dk

www.compute.dtu.dk/~chaorf

Slides by Xenofon (Fontas) Fafoutis

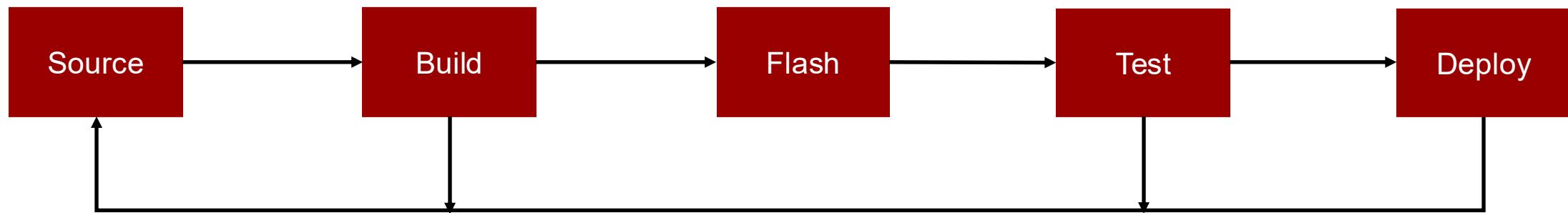
Embedded Software

- Embedded Software is software that runs on the processor of an embedded system
- Embedded software is typically closer to the hardware
 - Tied to particular hardware
 - Built around hardware constraints and limitations
 - Controlling sensors, actuators, peripherals, etc
- Also known as firmware
 - Typically, the software that runs on embedded systems does not change as easily or frequently as in general purpose computers
 - It is firmer!
- Most commonly written in C



Stages of Embedded Software Development

- The development environment is not the same as the execution environment
 - Development takes place at a normal computer
 - Executable binary is installed on the embedded device
 - aka programming or flashing the device
 - Debugging/testing takes place on the embedded device



Embedded Operating Systems

- What is the role of an operating system?

Embedded Operating Systems

- The Operating System (OS) manages and abstracts hardware resources
- However, traditional OS are often not suitable for embedded systems
- Embedded systems without an OS
 - Application talks directly to the hardware (bare metal)
 - Maximum efficiency, but no flexibility
- An Embedded OS provides
 - Hardware abstractions
 - APIs and drivers for common peripherals
 - Intuitive programming models

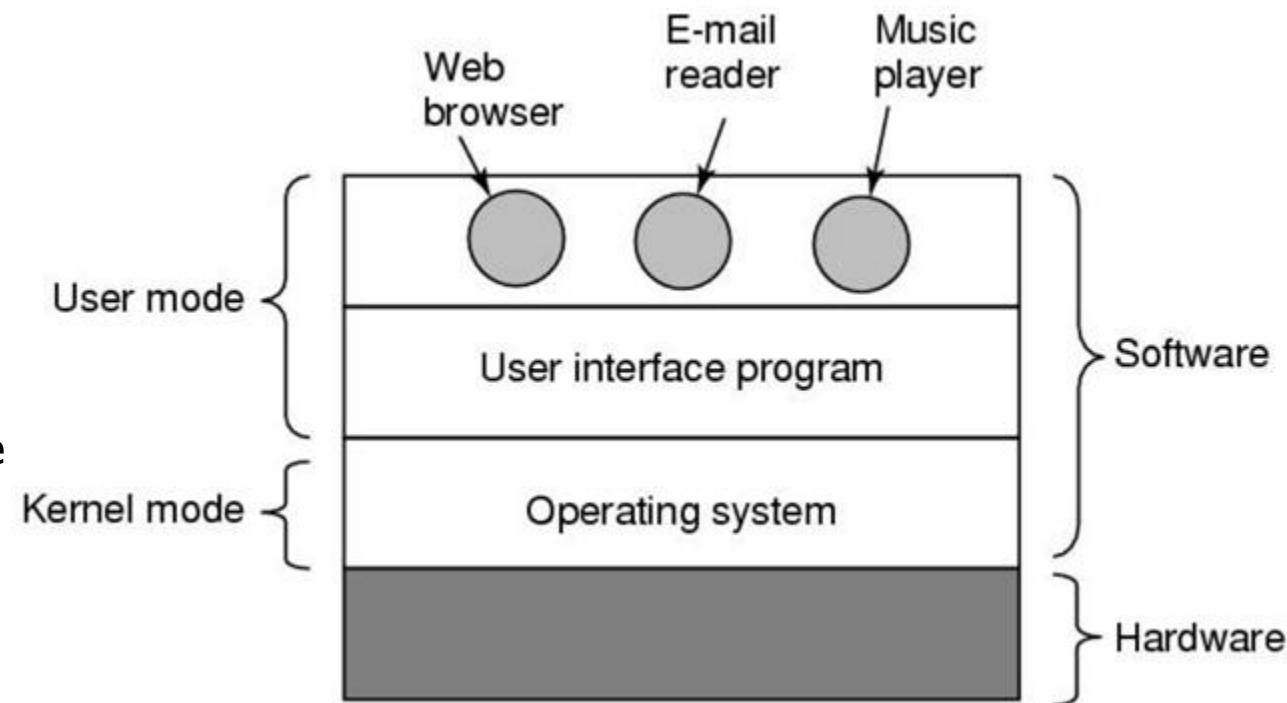
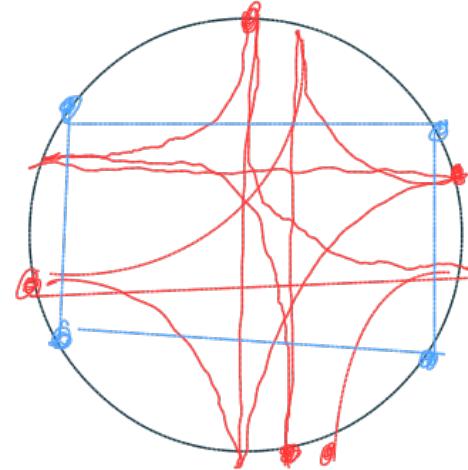


Image source: Modern Operating Systems by Tanenbaum and Bos

Requirements for Embedded OS

- Small memory footprint
 - As low as kbytes in some low-end embedded systems
- Support for heterogeneous hardware
 - Large variety of architectures (8-bit, 16-bit, 32-bit)
 - Variety of available RAM/flash memory
 - Variety of network interfaces (wired, wireless)
- Energy Efficiency
 - OS must duty cycle the CPU/radio/peripherals or provide API to application
- Real-time capabilities
 - Support for timely execution is crucial for various time-sensitive applications
 - An RTOS (Real-Time OS) is an OS that can guarantee worst-case execution times
- Security (confidentiality, authentication, data integrity, access control, etc)



Modules of an Embedded OS

- Embedded OS are designed to be modular
- The application is compiled together with the OS
- The embedded software developer decides which modules to include
 - Through the build system (make)
 - Pre-processor statements (#ifdef)
- The executable binary includes only the absolutely necessary code
- Supporting code for other CPUs, drivers for peripherals not present, protocols not used, etc, are not included in the binary

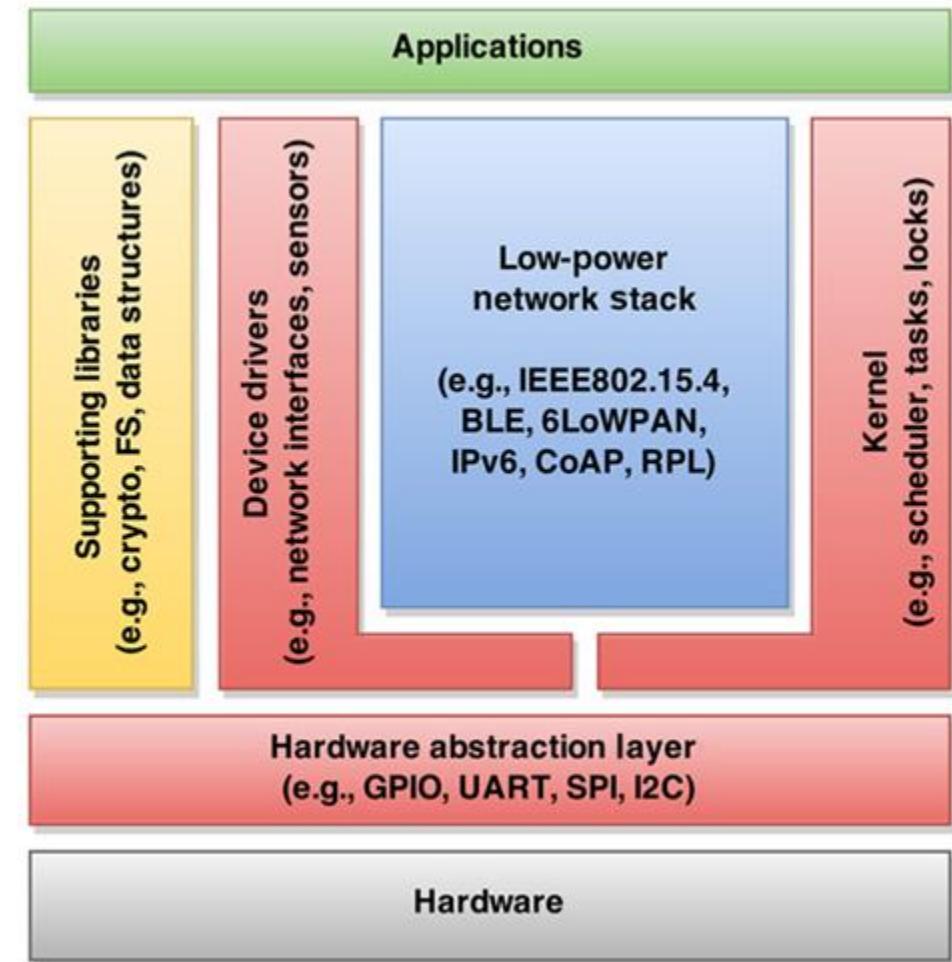


Image source: <https://doi.org/10.1109/JIOT.2015.2505901>

Event-Driven Embedded OS

- All processing triggered by an external event
 - Events can be generated by peripherals
 - CPU can schedule timer events
- Roughly equivalent to an infinite loop that handles events
- Everything (OS and apps) runs within the same context and address space
 - Like one single programme
- High efficiency in terms of memory and processing
 - Not all programmes can be easily expressed as state machines
- Example: Contiki-NG

Multi-Threading Operating Systems

- Each thread runs in its own context, and manages its own stack
- Scheduler has to perform context switching
- Traditional approach in modern OS (e.g. Linux)
- More natural programming model, but with overhead
 - Memory overhead, runtime overhead
- Example: RIOT

Real-Time Operating Systems (RTOS)

- Primary goal is to provide real-time guarantees
 - In terms of worst-case execution time
- Formal verification, certification, and standardisation
- Strict constraints for the developers
- OS is inflexible and difficult to port to new hardware
- Example: FreeRTOS

Scheduling

- The scheduler manages how the processor is shared among tasks
 - It decides when each task (process or thread) will be executed
- Preemptive Scheduling
 - Scheduler can interrupt the execution of a task to allow another task to use the CPU
 - Fundamental for fairness in UI-based OS
 - Less efficient, more context switches, timer prevents the system from going in sleep
- Non-preemptive Scheduling (cooperative scheduling)
 - Each task executes until it voluntarily yields the CPU
 - More efficient, but vulnerable to errors
 - Tasks must quickly yield the CPU or risk blocking important system functions
- Tickless Preemptive Scheduling
 - Preemption occurs on interrupts and when a task naturally blocks (not time-based)
 - Efficient solution in the middle

Priority Scheduling

- Tasks associated with a priority level
- The scheduler selects the task with the highest priority level
- Highest priority tasks (typically system-level tasks) always get priority over the lower priority tasks
- Lowest priority can suffer from starvation in busy systems
- Rarely the case in embedded systems that spend a lot of time idle/asleep

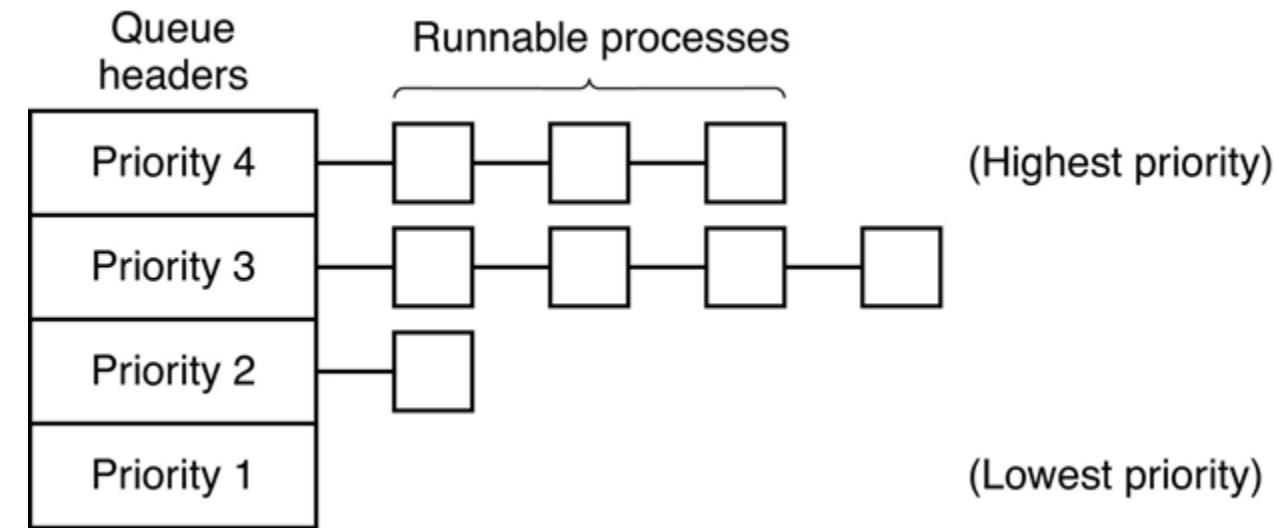


Image source: Modern Operating Systems by Tanenbaum and Bos

Multithreading and Thread Synchronisation

- Parallelism
 - Multicore CPUs
 - Preemptive scheduling enables pseudo-parallel thread execution with one CPU
 - Embedded systems often incorporate auxiliary processors and accelerators (true parallelism)
- Parallelism creates the risk of race conditions
 - Shared memory
 - Shared peripherals
- Thread synchronisation tools
 - Mutex
 - Condition variable
- Inter-Process Communication (IPC)
 - Messages

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

Image source: Wikipedia

Embedded Operating Systems

Name	Architecture	Scheduler	Programming model	Targeted device class ^a	Supported MCU families or vendors	Programming languages	License	Network stacks
Contiki	Monolithic	Cooperative	Event-driven, Protothreads	Class 0 + 1	AVR, MSP430, ARM7, ARM Cortex-M, PIC32, 6502	C ^b	BSD	uIP, RIME
RIOT	Microkernel RTOS	Preemptive, tickless	Multithreading	Class 1 + 2	AVR, MSP430, ARM7, ARM Cortex-M, x86	C, C++	LGPLv2	gnrc, OpenWSN, ccn-lite
FreeRTOS	Microkernel RTOS	Preemptive, optional tickless	Multithreading	Class 1 + 2	AVR, MSP430, ARM, x86, 8052, Renesas ^c	C	modified GPL ^d	None
TinyOS	Monolithic	Cooperative	Event-driven	Class 0	AVR, MSP430, px27ax	nesC	BSD	BLIP
OpenWSN	Monolithic	Cooperative ^e	Event-driven	Class 0 – 2	MSP430, ARM Cortex-M	C	BSD	OpenWSN
nuttX	Monolithic or microkernel	Preemptive (priority-based or round robin)	Multithreading	Class 1 + 2	AVR, MSP430, ARM7, ARM9, ARM Cortex-M, MIPS32, x86, 8052, Renesas	C	BSD	native
eCos	Monolithic RTOS	Preemptive	Multithreading	Class 1 + 2	ARM, IA-32, Motorola, MIPS ...	C	eCos License ^f	lwIP, BSD
uCLinux	Monolithic	Preemptive	Multithreading	>Class 2	Motorola, ARM7, ARM Cortex-M, Atari	C	GPLv2	Linux
ChibiOS/RT	Microkernel	Preemptive	Multithreading	Class 1 + 2	AVR, MSP430, ARM Cortex-M	C	Triple License ^g	None
CoOS	Microkernel RTOS	Preemptive	Multithreading	Class 2	ARM Cortex-M	C	BSD	None
nanorK	Monolithic (resource kernel)	Preemptive	Multithreading	Class 0	AVR, MSP430,	C	Dual License	None
Nut/OS	Monolithic	Cooperative	Multithreading	Class 0 + 1	AVR, ARM	C	BSD	native

Image source: <https://doi.org/10.1109/JIOT.2015.2505901>

Embedded Operating Systems

OS	Contiki	Contiki-NG	TinyOS	FreeRTOS	OpenWSN	RIOT	Zephyr
MCU	MSP430	MSP430	MSP430	MSP430	MSP430	MSP430	ARM
	AVR	Cortex-M	AVR	AVR	Cortex-M	ARM 7	x86
	Cortex-M	JN516x		Cortex-M		Cortex-M	Xtensa
	ARM 7			Cortex-A		x86	RISC-V
	8051			ARM7		AVR	ARC
	RL78			Cyclone V SOC		ESP8266	Nios II
	6502			ARM9		RISC-V	POSIX/NATIVE
	x86			PIC32			SPARC
				NIOS II			
				8051			
				x86			
				Microblaze			
				APS3			
				78K0R			
				TMS570			
RAM [KB]	10	10	10	4-8	-	1.5	8
Flash [KB]	30	~100	48	32-64	-	5	-
RPL	✓	✓	✓	✗	✓	✓	✓
UDP	✓	✓	✓	✓	✓	✓	✓
TCP	✓	✓	Experimental	✗	✓	✓	✓

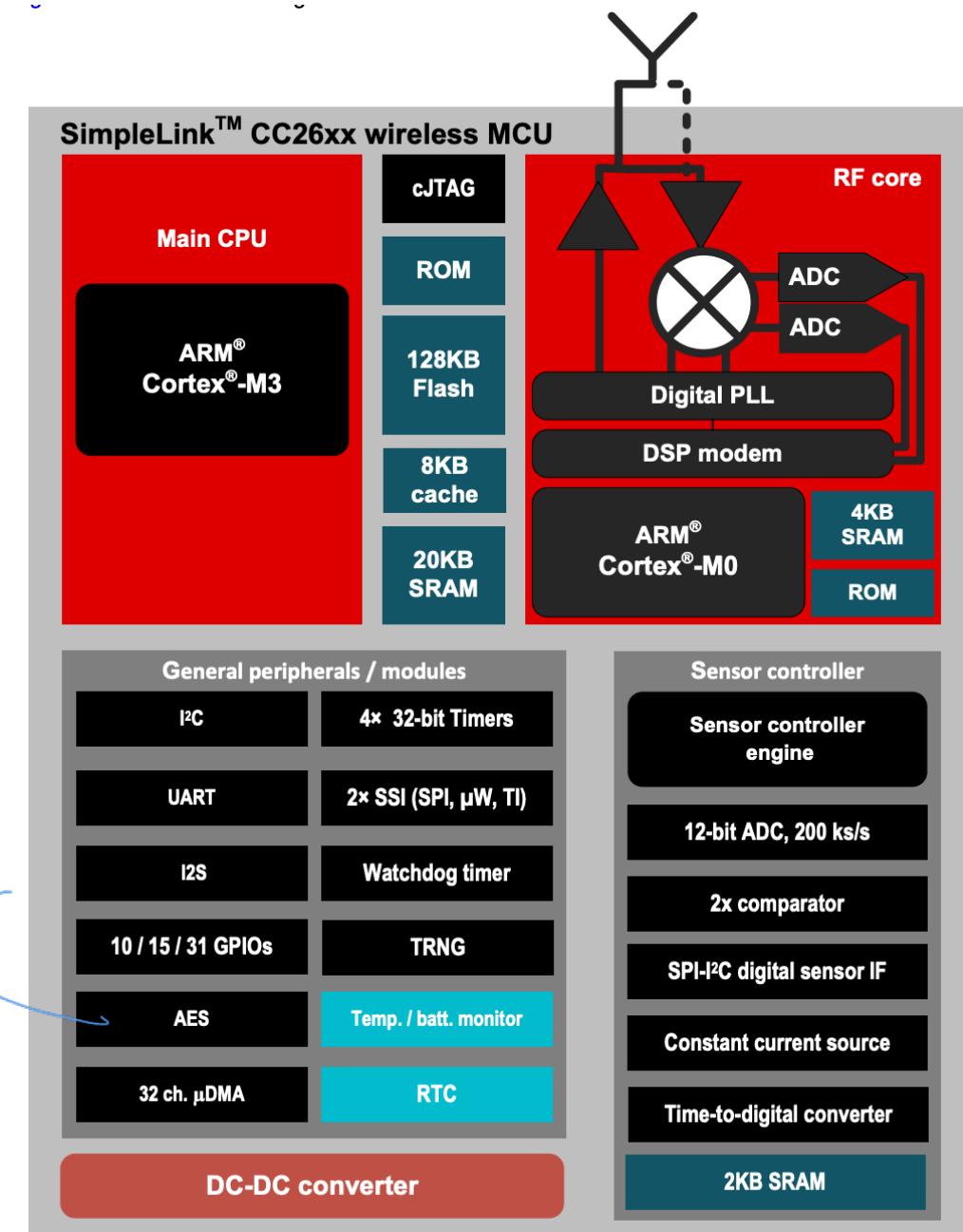
Image source: <https://doi.org/10.1109/ACCESS.2022.3153521>

How do we choose an Embedded OS?

- Features
 - Supported hardware (platforms, CPUs, peripherals, accelerators)
 - Supported software (communication protocols, file system, encryption libraries)
 - Tools (simulator, debugging tools)
- Portability: vendor-based vs generic OS, open standards (POSIX)
- Certification: real-time guarantees, networking interoperability
- License: proprietary, permissive, copyleft
- Documentation, Support, Community

Embedded Programming

- Typically involves communication with peripherals
 - Internal peripherals in the System-on-Chip (SoC)
 - External peripherals (same board)
- Peripheral controller has a number of registers
 - The CPU reads/writes the registers
 - Types of registers
 - Status registers (read-only)
 - Command registers (write-only)
 - Data/Configuration registers (read/write)
- Bitwise operations, fixed point operations
- Static memory allocation is preferred



Events and Interrupts

- Peripherals can be programmed to issue signals (interrupts) when specific events happen
 - GPIOs can detect changes in line to capture interrupts from external peripherals
 - The interrupt controller handles interrupt for multiple devices
- An Interrupt Request (IRQ) is then sent to the CPU
 - Interrupts normal execution and executes a specified interrupt handler

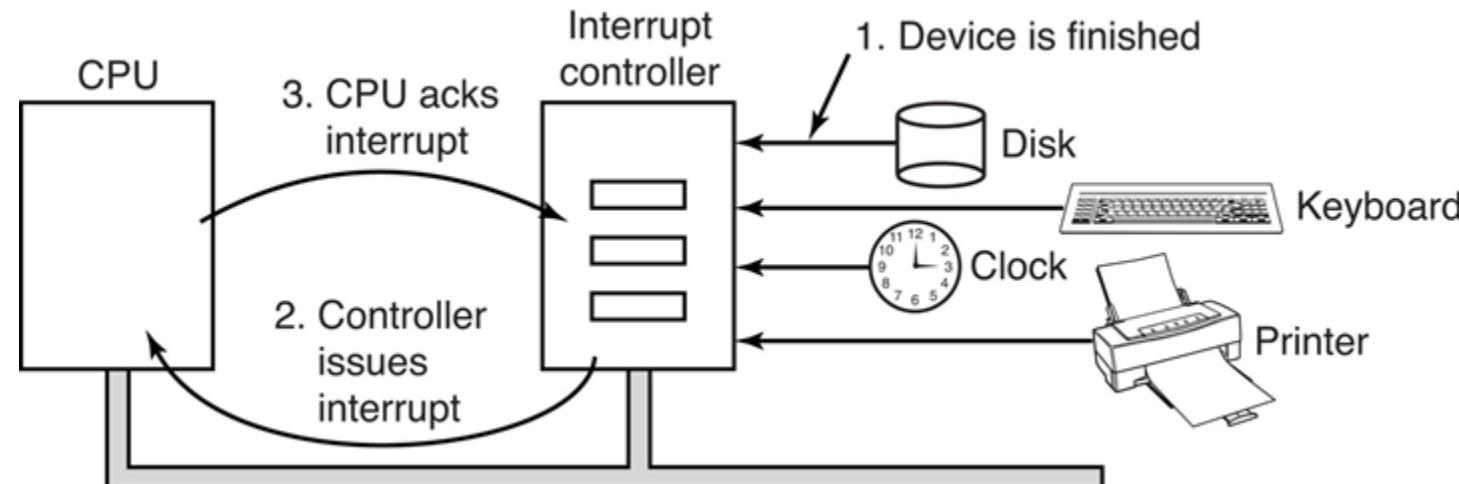


Image source: Modern Operating Systems by Tanenbaum and Bos

I/O Software with Busy Waiting

- Example: Send a string to peripheral (e.g. printer)
 - Copy first character from memory to printer's data register
 - Read printer's status register in a while loop until printer is done
 - Repeat until the end of the document

```
copy_from_user(buffer, p, count);           /* p is the kernel buffer */
for (i = 0; i < count; i++) {                /* loop on every character */
    while (*printer_status_reg != READY);   /* loop until ready */
    *printer_data_register = p[i];          /* output one character */
}
return_to_user();
```

- Disadvantages?

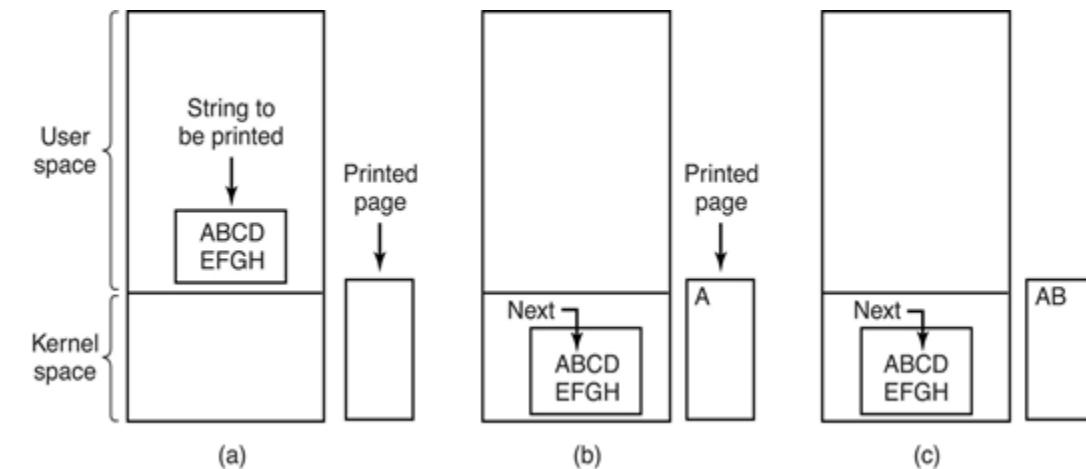


Image source: Modern Operating Systems by Tanenbaum and Bos

I/O Software with Busy Waiting

- Example: Send a string to peripheral (e.g. printer)
 - Copy first character from memory to printer's data register
 - Read printer's status register in a while loop until printer is done
 - Repeat until the end of the document

```
copy_from_user(buffer, p, count);           /* p is the kernel buffer */
for (i = 0; i < count; i++) {                /* loop on every character */
    while (*printer_status_reg != READY);   /* loop until ready */
    *printer_data_register = p[i];          /* output one character */
}
return_to_user();
```

- Disadvantages?
 - Occupies the CPU doing nothing
 - Wastes energy
 - Heats up the system

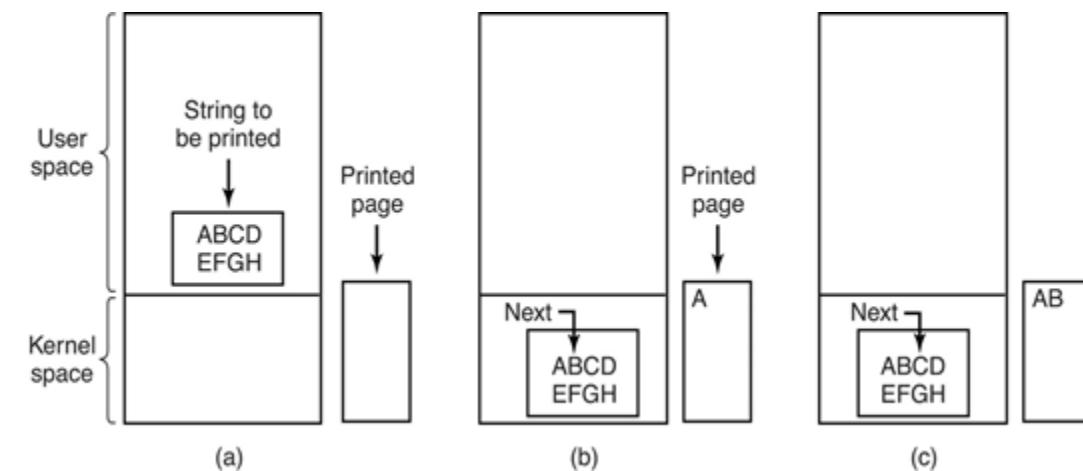


Image source: Modern Operating Systems by Tanenbaum and Bos

Interrupt-Driven I/O Software

- Example: Send a string to peripheral (e.g. printer)
 - Enable/configure interrupts
 - Copy first character from memory to printer's data register
 - Put CPU to sleep
 - When peripheral is done it issues an interrupt
 - The interrupt handler wakes up the CPU
 - Copy next character and repeat until the end of file

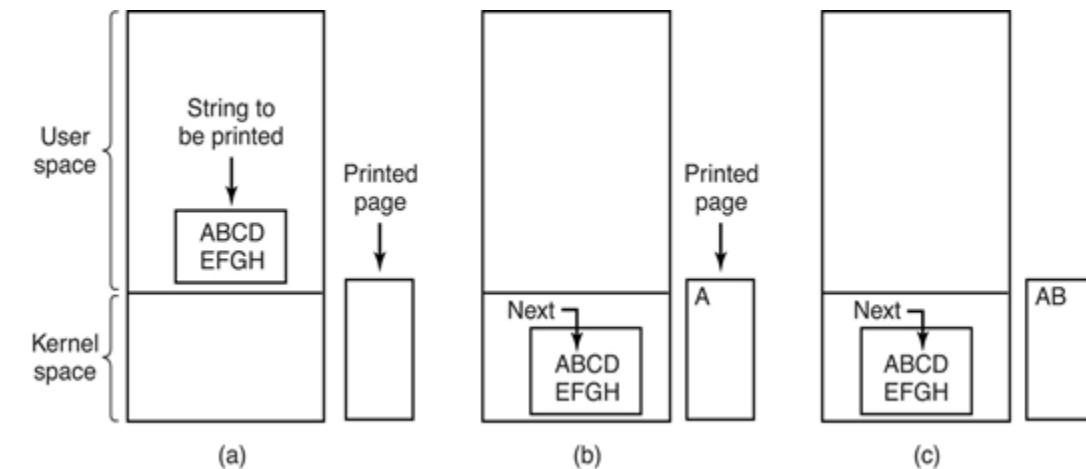
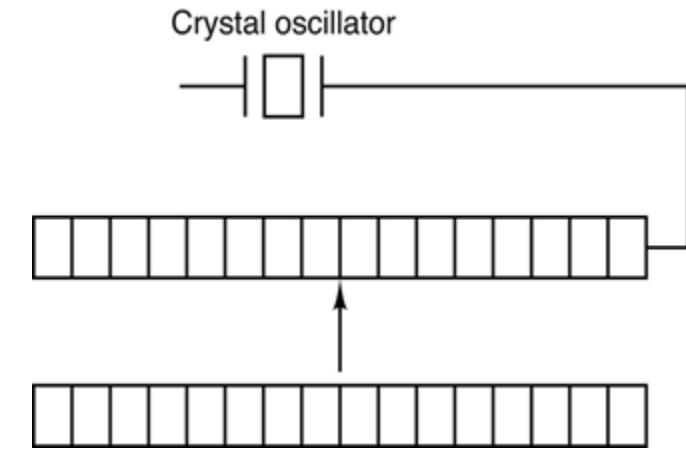


Image source: Modern Operating Systems by Tanenbaum and Bos

Timers

- An SoC would typically have a fixed number of hardware timers
 - High-frequency timers and low-frequency timers (RTC)
 - Example: CC2650 has 2 HF timers and 1 RTC
- A timer has a resolution that depends on its frequency ($1/f$)
 - A timer with $f=32\text{KHz}$ has resolution $\sim 30.5 \text{ us}$
 - A timer with $f=24\text{MHz}$ has resolution $\sim 41.7 \text{ ns}$
- A timer will overflow periodically according to its size and frequency
 - A 24-bit timer with $f=32\text{KHz}$ will overflow after 512 seconds
 - A 24-bit timer with $f=24\text{MHz}$ will overflow after ~ 0.7 seconds
- Each hardware timer needs to support multiple software timers
 - Timer increments at specific frequency and wraps when reaches maximum value
 - In each timer a COMPARE value can be set, to issue a clock interrupt when timer reaches it
 - The COMPARE value can be updated in software to reset the software timer



A Timer Challenge

M K

- Assuming my system has two oscillators 24MHz and 32KHz that drive two 24-bit timers
 - The LF timer with $f_{LF}=32\text{KHz}$ will overflow after 512 seconds
 - The HF timer with $f_{HF}=24\text{MHz}$ will overflow after ~ 0.7 seconds
- How do I schedule an event to occur in 5 minutes (300 seconds)?
 - Read the current value of LF timer (COUNTER)
 - Calculate the ticks the correspond to the interval ($\text{INTERVAL} = 300 \times f_{LF} = 9830400$)
 - Add ticks to current value, making sure I wrap around timer maximum value
 - $\text{COMPARE} = (\text{COUNTER} + \text{INTERVAL}) \bmod 2^{24}$
- How do I schedule an event to occur in 10 minutes (600 seconds)?



A Timer Challenge

- How do I schedule an event to occur in 10 minutes (600 seconds)?
 - The LF timer with $f_{LF} = 32768$ Hz will overflow after 512 seconds
- Approach #1: Generate more events than needed
 - Schedule an event every 300 seconds, ignore every odd event
- Approach #2: Trade resolution for overflow period
 - Set the PRESCALER=1 to generate a lower timer frequency ($f_{LF} = 16384$ Hz)
 - Calculate the interval based on reduced frequency (INTERVAL = $600 \times f_{LF} = 9830400$)
 - Set COMPARE = (COUNTER + INTERVAL) mod 2^{24}

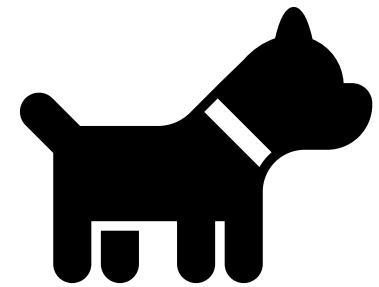
$$f_{RTC} [\text{kHz}] = 32.768 / (\text{PRESCALER} + 1)$$

Prescaler	Counter resolution	Overflow
0	30.517 μs	512 seconds
2^8-1	7812.5 μs	131072 seconds
$2^{12}-1$	125 ms	582.542 hours

Image source: nRF52832 Datasheet by Nordic Semiconductors

Watchdog Timer

- A safety mechanism against system crashes
- A timer that resets the system when it expires
- Under normal operation the system should reset the timer periodically making sure it never expires
- If the system hangs (deadlock, infinite loop, etc), the watchdog will reset the system
- Can be temporarily paused in long sleep mode or when the system is halted by a debugger



Direct Memory Access (DMA)

- Transfers data between the memory and peripherals without the involvement of CPU
- Support for multiple channels (multiple data transfers)
- Transfer modes:
 - Memory-to-memory
 - Memory-to-peripheral
 - Peripheral-to-memory
 - Peripheral-to-peripheral
- CPU gets notified with an interrupt when transfer done
- Slower but more energy efficient than CPU

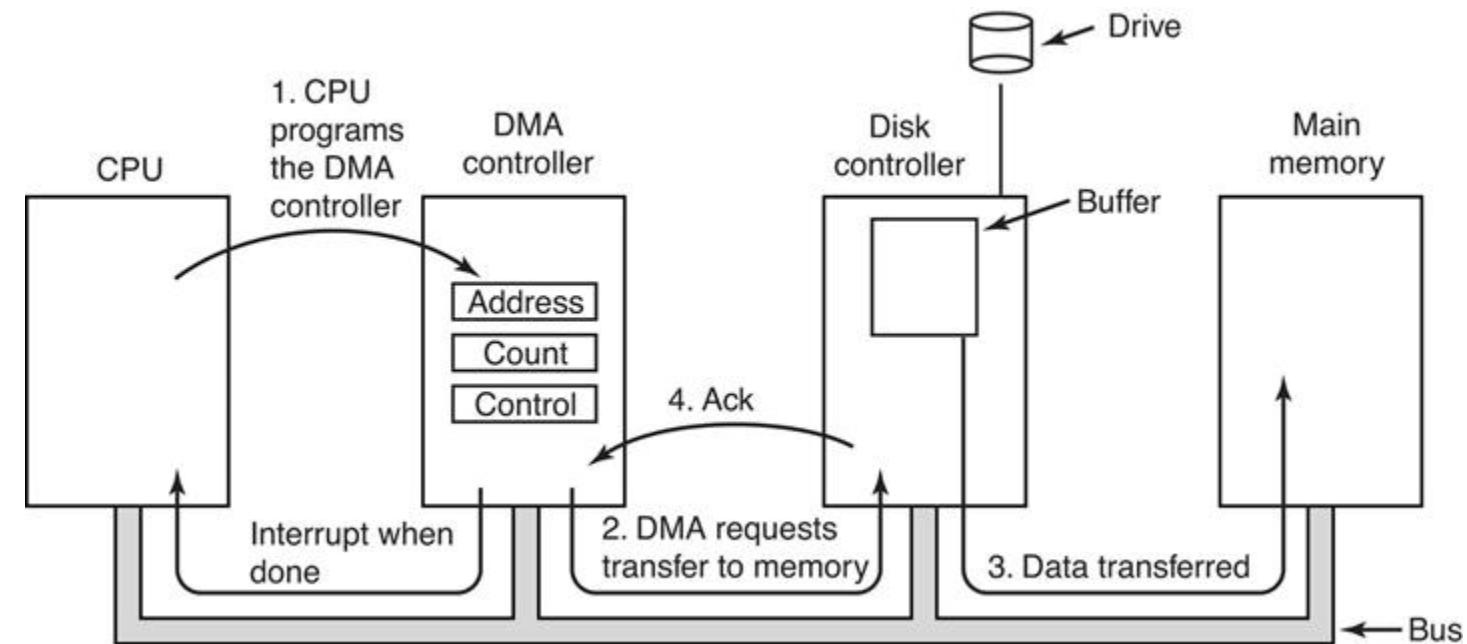


Image source: Modern Operating Systems by Tanenbaum and Bos

MCU Power Modes

- MCUs provide granular control on which sub-modules to have active
 - Energy consumption vs functionality
- Active Mode
 - CPU active, RAM on, HF clock on, internal peripherals active if needed
- Idle Mode
 - CPU off, RAM on, HF clock on, internal peripherals active if needed
- Sleep Mode (or Standby Mode)
 - CPU off, RAM retention, HF clock off, internal peripherals unavailable
 - LF clock on, time-scheduled wakeup possible
- Deep Sleep Mode (or Shutdown Mode)
 - CPU off, no RAM retention, HF clock off, LF clock off, internal peripherals unavailable
 - Wakeup on pin edge possible

Example: CC2650 Power Modes

Mode	Software Configurable Power Modes			
	Active	Idle	Standby	Shutdown
System CPU	Active	Off	Off	Off
System SRAM	On	On	Retained	Off
Register retention ⁽¹⁾	Full	Full	Partial	No
VIMS_PD (flash)	On	Available	Off	Off
RFCORE_PD (radio)	Available	Available	Off	Off
SERIAL_PD	Available	Available	Off	Off
PERIPH_PD	Available	Available	Off	Off
Sensor controller	Available	Available	Available	Off
Supply system	On	On	Duty-cycled	Off
High-speed clock	XOSC_HF or RCOSC_HF	XOSC_HF or RCOSC_HF	Off	Off
Low-speed clock	XOSC_LF or RCOSC_LF	XOSC_LF or RCOSC_LF	XOSC_LF or RCOSC_LF	Off
Wakeup on RTC	Available	Available	Available	Off
Wakeup on pin edge	Available	Available	Available	Available
Wakeup on reset pin	Available	Available	Available	Available

Image source: CC13xx, CC26xx SimpleLink Wireless MCU Technical Reference Manual by Texas Instruments

Example: CC2650 Consumption

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT	
I_{core}	Core current consumption	Reset. RESET_N pin asserted or VDDS below Power-on-Reset threshold	100			nA	
		Shutdown. No clocks running, no retention	150				
		Standby. With RTC, CPU, RAM and (partial) register retention. RCOSC_LF	1				
		Standby. With RTC, CPU, RAM and (partial) register retention. XOSC_LF	1.2				
		Standby. With Cache, RTC, CPU, RAM and (partial) register retention. RCOSC_LF	2.5			μ A	
		Standby. With Cache, RTC, CPU, RAM and (partial) register retention. XOSC_LF	2.7				
		Idle. Supply Systems and RAM powered.	550				
		Active. Core running CoreMark	1.45 mA + 31 μ A/MHz				
		Radio RX ⁽¹⁾	5.9				
		Radio RX ⁽²⁾	6.1				
		Radio TX, 0-dBm output power ⁽¹⁾	6.1				
		Radio TX, 5-dBm output power ⁽²⁾	9.1				

Image source: CC2650 Datasheet by Texas Instruments

Embedded Programming Tricks

- Two's Compliment Numbers for signed numbers
 - To take negative N-bit value, subtract from 2^N
 - Example: in an 8-bit integer, $-100 = 256 - 100 = 156 = 0x9C$
- Use explicit data types and minimal data types
 - How big is an int? in many embedded systems it is not 32 bits!
 - Use instead: int32_t, uint32_t, int16_t, uint16_t, etc
 - Don't use a uint32_t when a uint8_t is enough
- Static Variables (e.g. static uint8_t A;)
 - Puts variable A in statically allocated piece of global memory (not stack)
 - Variable survives between function calls
- Volatile Variables (e.g. volatile uint8_t A;)
 - Specifies that variable is expected to change by other entities
 - Compiler does not put it in register or cache

Embedded Programming Tricks: Inline Functions

- Inline functions
 - Omit overhead of calling function
 - Speeds up execution of basic functions
 - Copy/pasting code is bad practice for code maintenance

```
inline sum (int a, int b) {  
    int result;  
    result = a + b;  
    return result;  
}
```

- The two below are identical:
 - `c = sum(a,b);`
 - `c = a + b;`

Embedded Programming Tricks: Macros

- Avoid hardcoded numbers
 - `#define TICKS_IN_SECOND 32768`
- Remove functionality that you don't need at runtime
 - `#ifdef FEATURE ... #endif`
- Use parenthesis to avoid bugs
 - `#define ONE_PLUS_ONE 1+1 -> A = ONE_PLUS_ONE * 10 = 1+1*10 = 11`
 - `#define ONE_PLUS_ONE (1+1) -> A = ONE_PLUS_ONE * 10 = (1+1)*10 = 20`
- Same with function-like macros (use inline functions preferably)
 - `#define TIMES_TWO(x) x*2 -> A = TIMES_TWO(1+1) = 1+1*2 = 3`
 - `#define TIMES_TWO(x) (x)*2 -> A = TIMES_TWO(1+1) = (1+1)*2 = 4`

Embedded Programming Tricks: Efficient Math

- Shift left N to multiply by 2^N
 - $A = A << 2$ is equivalent to $A = A * 4$
- Shift right N to divide by 2^N (remainder gets lost)
 - $A = A >> 2$ is equivalent to $A = A / 4$
- Use fixed point operations instead of floating-point operations
 - Variable T is an int16 that holds temperature in 100ths of °C
 - T = 2535 means 25.35 °C
 - $T = T >> 1$ divides temperature by 2, so T = 1267 that is 12.67 °C

Specific Bits

- Often you need to read/set/clear/toggle a specific bit within a byte

[7] INT_LOW (RW)
Interrupt Active Low

[6] AWAKE (RW)
Awake Interrupt

[5] INACT (RW)
Inactivity Interrupt

[4] ACT (RW)
Activity Interrupt

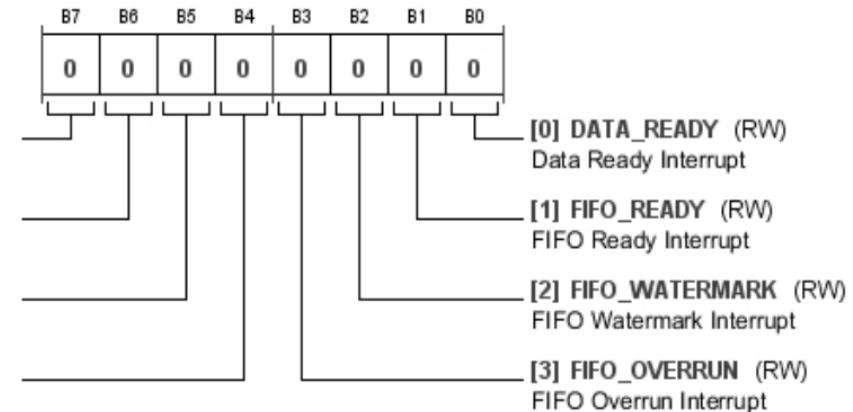


Table 15. Bit Descriptions for INTMAP1

Bits	Bit Name	Settings	Description	Reset	Access
7	INT_LOW		1 = INT1 pin is active low.	0x0	RW
6	AWAKE		1 = maps the awake status to INT1 pin.	0x0	RW
5	INACT		1 = maps the inactivity status to INT1 pin.	0x0	RW
4	ACT		1 = maps the activity status to INT1 pin.	0x0	RW
3	FIFO_OVERRUN		1 = maps the FIFO overrun status to INT1 pin.	0x0	RW
2	FIFO_WATERMARK		1 = maps the FIFO watermark status to INT1 pin.	0x0	RW
1	FIFO_READY		1 = maps the FIFO ready status to INT1 pin.	0x0	RW
0	DATA_READY		1 = maps the data ready status to INT1 pin.	0x0	RW

Image source: ADXL362 Datasheet from Analog Devices

Embedded Programming Tricks: Bitwise Operations

- Often you need to set/clear/toggle/read a specific bit within a byte (`uint8_t A`)
- Create a mask on a specific bit (left shift moves `0x01` to desired position)
 - `M = (0x1<<6); // mask on bit 6, that is 0b01000000`
- Setting a bit (logic OR with `1` sets the bit, OR with `0` keeps bit as is)
 - `A = A | (0x1<<6); // sets bit 6`
- Toggle a bit (logic XOR with `1` inverts the bit, XOR with `0` keeps bit as is)
 - `A = A ^ (0x1<<6); // toggles bit 6`
- Clear a bit (logic AND with `0` clears the bit, AND with `1` keeps bit as is)
 - `A = A & ((0x1<<6) ^ 0xFF); // clear bit 6`
- Branch if a bit is set (logic AND with `0` clears the bit, AND with `1` keeps bit as is)
 - `if(A & (0x1<<6)){} // branches if bit 6 is set`

Sets of Bits

- Often you need to read/write a set of bits within a byte

FILTER CONTROL REGISTER

Address: 0x2C, Reset: 0x13, Name: FILTER_CTL

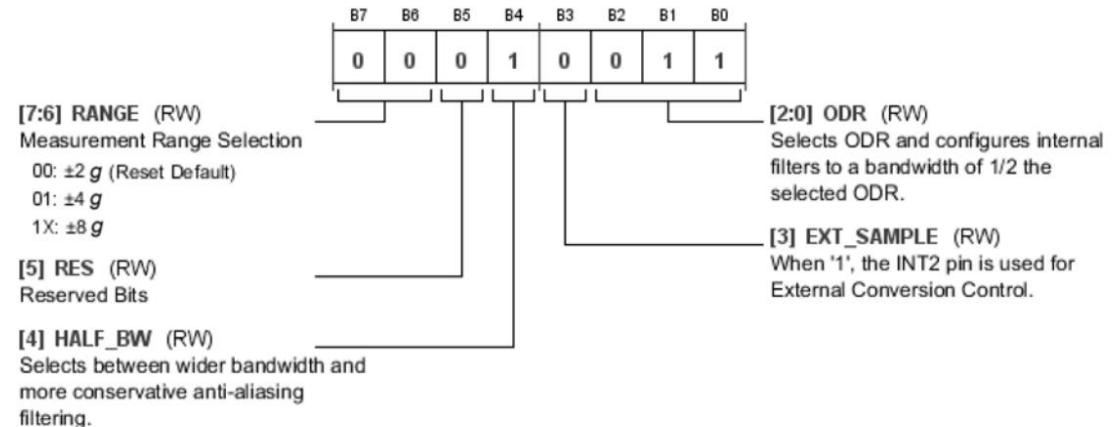


Table 17. Bit Descriptions for FILTER_CTL

Bits	Bit Name	Settings	Description	Reset	Access
[7:6]	RANGE	00 01 1X	Measurement Range Selection. $\pm 2 \text{ g}$ (reset default) $\pm 4 \text{ g}$ $\pm 8 \text{ g}$	0x0	RW
5	RES		Reserved.	0x0	RW
4	HALF_BW		Halved Bandwidth. Additional information is provided in the Antialiasing section. 1 = the bandwidth of the antialiasing filters is set to 1/4 the output data rate (ODR) for more conservative filtering. 0 = the bandwidth of the filters is set to 1/2 the ODR for a wider bandwidth.	0x1	
3	EXT_SAMPLE		External Sampling Trigger. 1 = the INT2 pin is used for external conversion timing control. Refer to the Using Synchronized Data Sampling section for more information.	0x0	RW
[2:0]	ODR	000 001 010 011 100 101...111	Output Data Rate. Selects ODR and configures internal filters to a bandwidth of 1/2 or 1/4 the selected ODR, depending on the HALF_BW bit setting. 000: 12.5 Hz 001: 25 Hz 010: 50 Hz 011: 100 Hz (reset default) 100: 200 Hz 101...111: 400 Hz	0x3	RW

Image source: ADXL362 Datasheet from Analog Devices

Embedded Programming Tricks: More Bitwise Operations

- Often you need to read/write a set of bits within a byte (uint8_t FILTER_CTRL)
- Create a mask on a set of bits
 - MASK3 = (0x1<<3)-1; // mask on first 3 bits, that is 0b00000111
 - MASK2 = (0x1<<2)-1; // mask on first 2 bits, that is 0b00000011
- Read a set of bits
 - ODR = FILTER_CTRL & MASK1; // keeps first 3 bits as is, clears all other
 - RANGE = (FILTER_CTRL>>6) & MASK2; // moves last 2 bits by 6 before masking
- Write a set of bits (without corrupting other bits!)
 - FILTER_CTRL = (FILTER_CTRL & (0xFF ^ MASK3)) | (new_odr & MASK3); // clears the 3 bits first, then OR the new value
 - FILTER_CTRL = (FILTER_CTRL & (0xFF ^ (MASK2<<6))) | ((new_range & MASK2)<<6); // clears the 2 bits first, then OR the new value

Programming Embedded Systems

- Programming or flashing the device
- The building toolchain outputs a binary file that needs to be uploaded on the flash memory of the embedded system
- Done a special programmer/debugger chip/board
 - Receives commands from USB/serial
 - Development boards often come with the programmer/debugger chip on board
- Alternatively, the MCU bootloader may be able to use the UART interface to receive firmware over serial

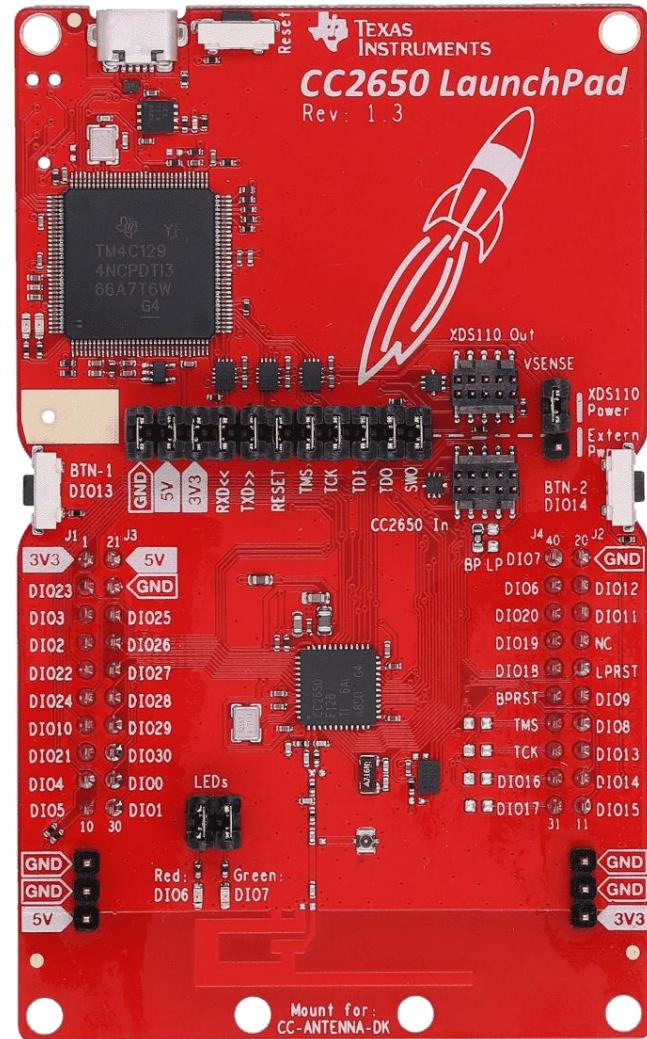


Image source: <https://www.ti.com/tool/LAUNCHXL-CC2650>

Programming/Debugging Interfaces

- Programming/Debugging Standards
 - Used for uploading binary code and as a debugger
- JTAG (Joint Test Action Group)
 - Generic standard (IEEE 1149.1)
 - 4 wires (TCK, TMS, TDI, TDO) + RESET (optional)
 - Can program multiple devices in a daisy chain
- cJTAG (IEEE 1149.7)
 - 2 wires (TMSC, TCKC) + RESET (optional)
 - Can program multiple devices in a star topology
- SWD (Serial Wire Debug)
 - Specific for ARM processors
 - 2 wires (SWDIO, SWCLK) + RESET (optional)
 - Can program multiple devices in a star topology

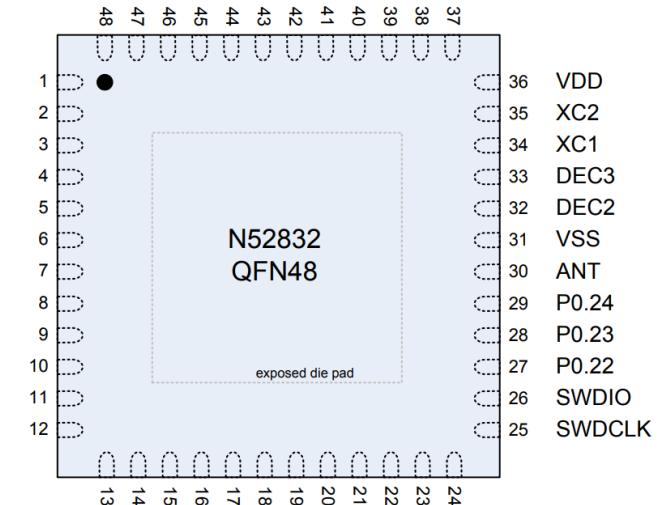
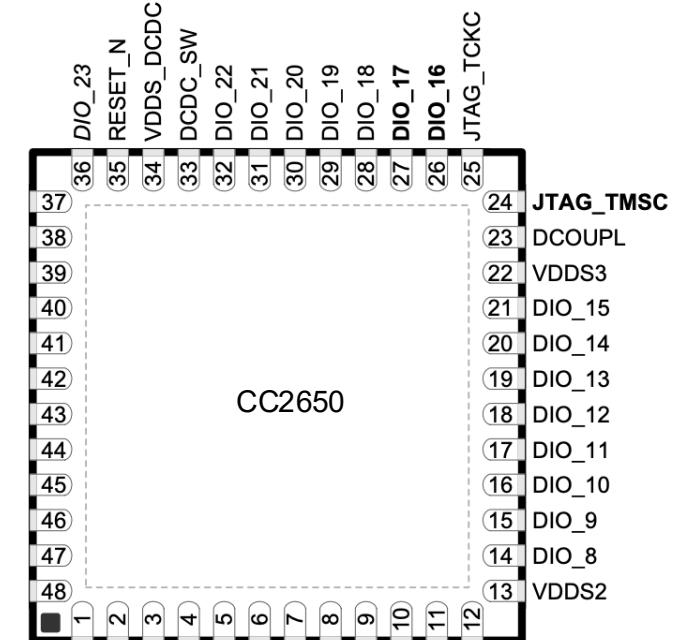
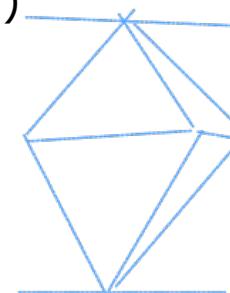


Image source: CC2650 Datasheet by Texas Instruments (top) and nRF52832 Datasheet by Nordic Semiconductors (bottom)

Debugging Embedded Software

- Debuggers are powerful (break points, read memory of embedded system, etc)
 - Break points not compatible with distributed software
 - Not always available in the wild
- LEDs as debugging tools
 - Blink when entering/exiting a code region
 - Turn on when entering an error state
- Printing messages and logs
 - Printing on serial if UART connected to a terminal
 - Printing on a file in flash memory
 - Sending log messages over wireless
- Oscilloscopes, Multi-meters, Logic Analysers
 - Test voltage levels and logic levels

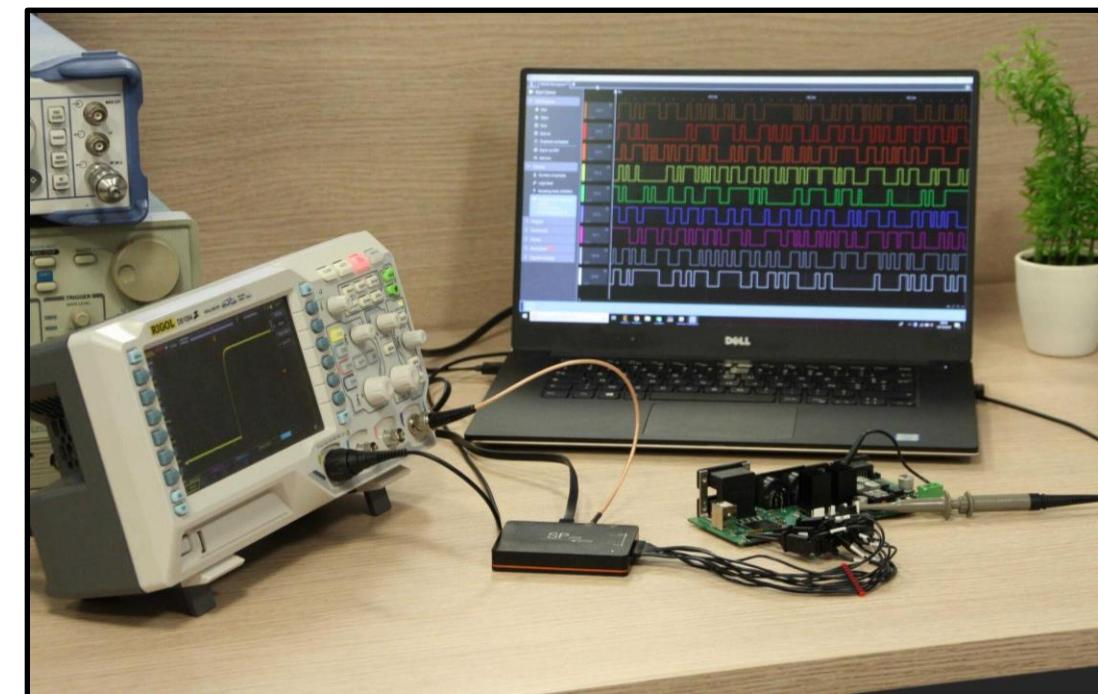
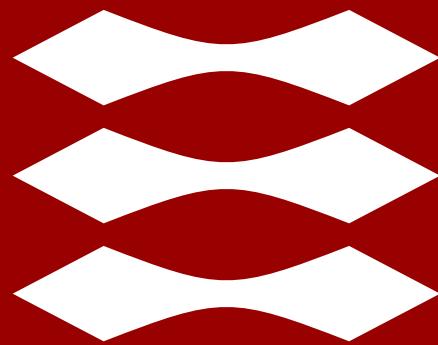


Image source: <https://www.ikalologic.com/sp209-logic-analyzer/>

DTU



Networked Embedded Systems

Week 5: Serial Communication

Xenofon (Fontas) Fafoutis

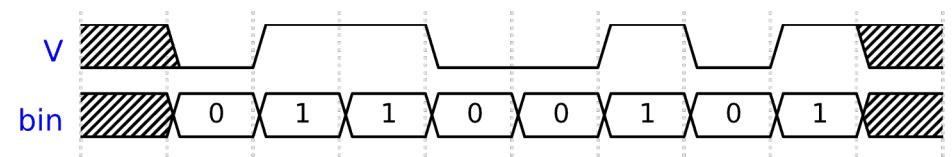
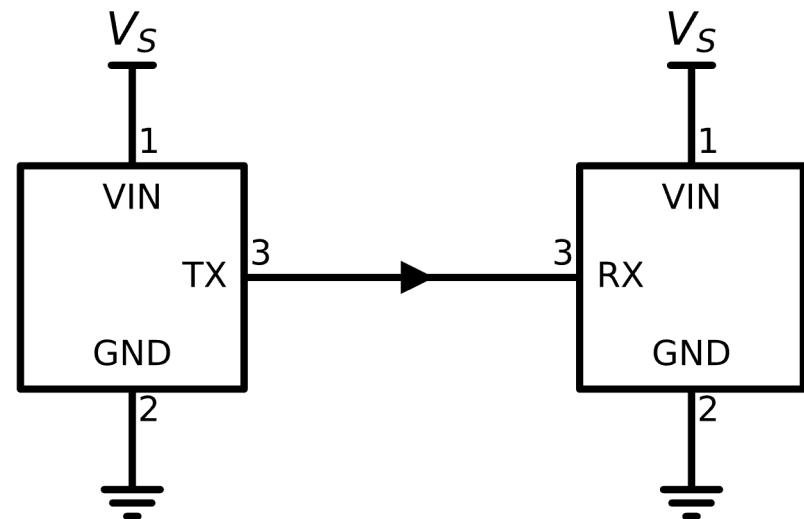
Professor

xefa@dtu.dk

www.compute.dtu.dk/~xefa

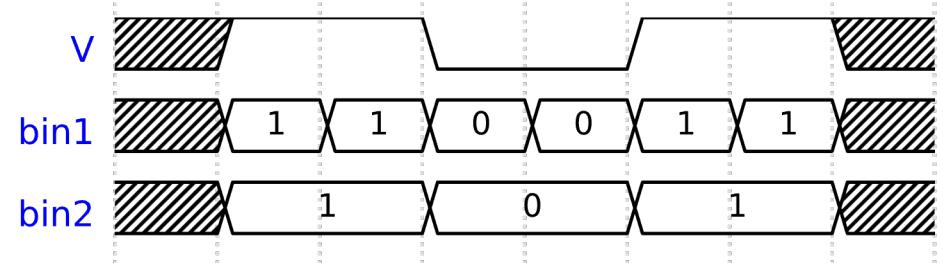
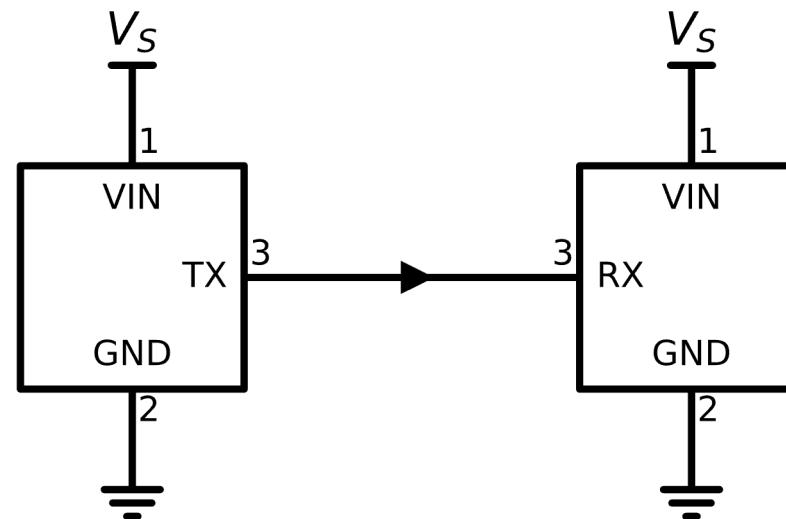
Serial Communication

- Bits are transmitted over a wire **in sequence**
 - One after the other
- Bits are encoded in the voltage level of the wire
 - E.g. high voltage is '1', low voltage is '0'



Timing the Receiver

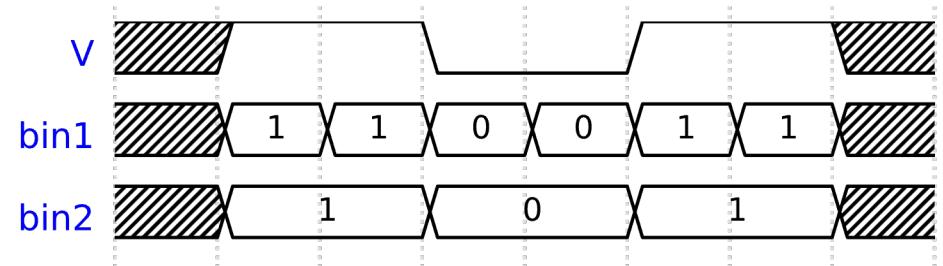
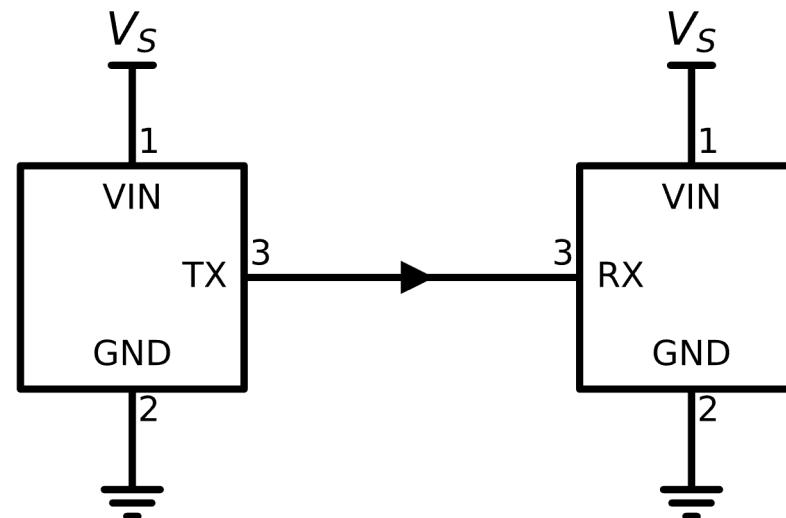
- The receiver can detect changes in the input
- How often shall the receiver read the input?
- What is the correct binary interpretation of the voltage V ? bin1 or bin2?



Timing the Receiver

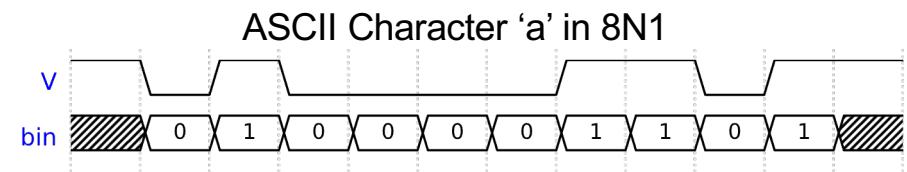
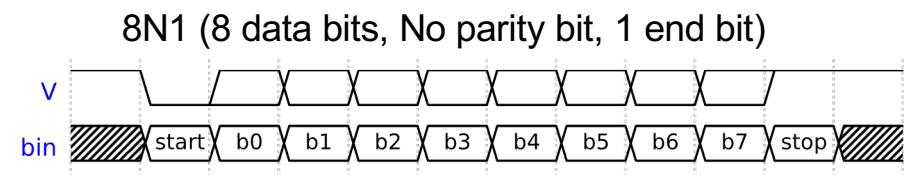
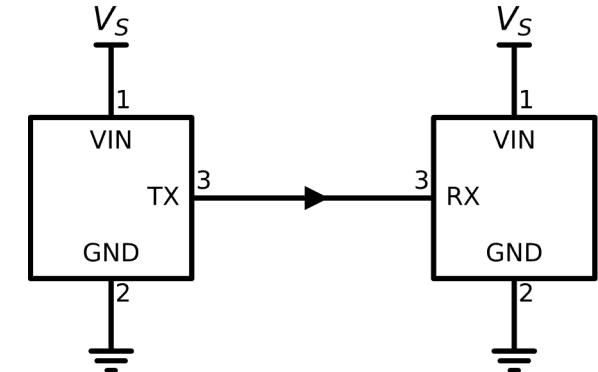
- The receiver can detect changes in the input
- How often shall the receiver read the input?
- What is the correct binary interpretation of the voltage V ? bin1 or bin2?

We can't know for sure! We need a way to synchronise the transmitter and the receiver!



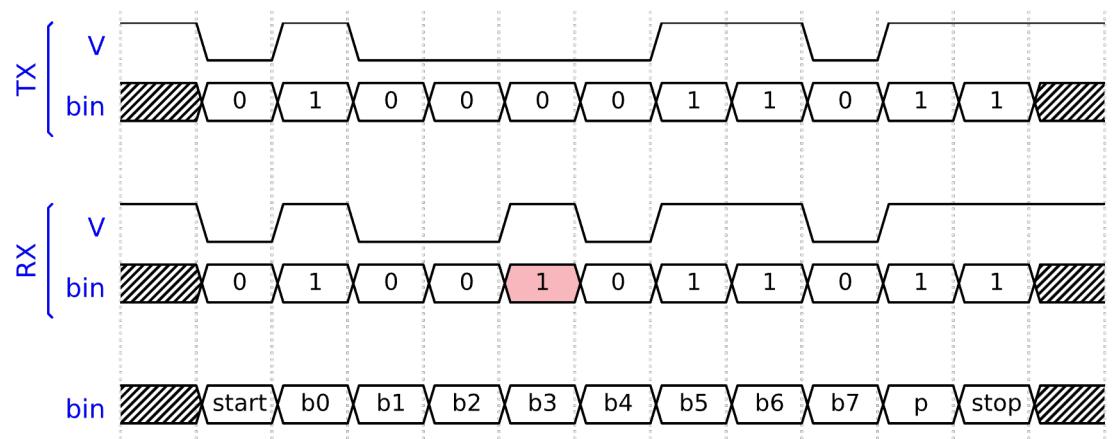
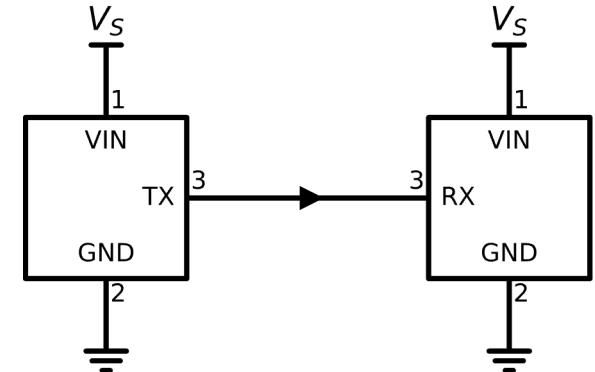
Asynchronous Serial Communication

- Transmitter and receiver do not share the same clock
- Synchronisation
 - When idle the line is at a high state, '1'
 - The beginning of each transmission is signalled by setting the line to '0' (start bit)
 - Bits transferred at a **pre-agreed baud rate** (\sim KHz)
 - The end of the transmission is signalled by setting the line to '1' and holding it (end bit)
 - There is one or two end bits
- Data are transmitted in a **pre-agreed format**
 - Data size (5-9 bits, typically 8 bits)
 - Bit order (typically, least significant bit first)
 - Parity (sum of all bits for error checking, optional)



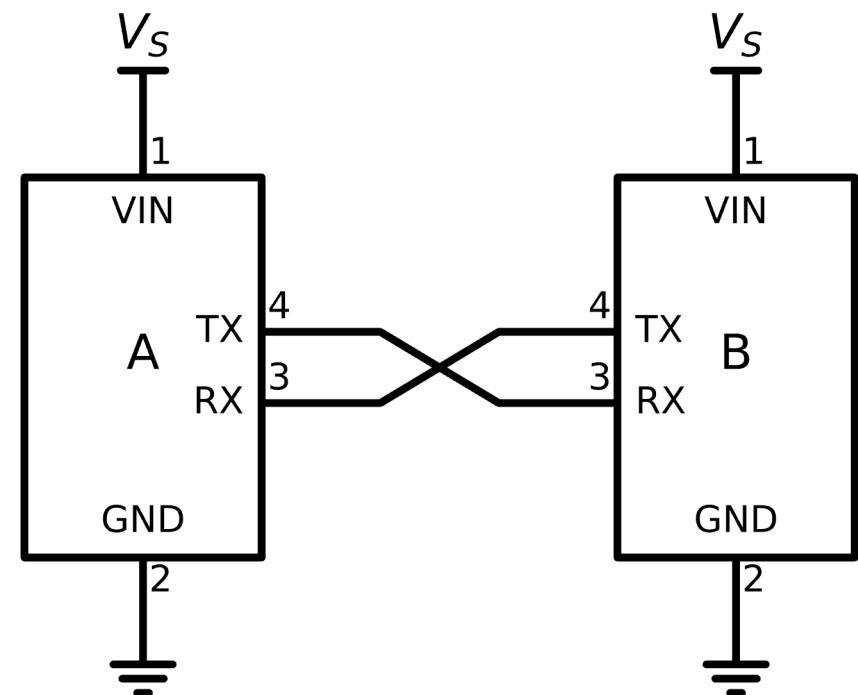
Error Detection: Parity Check

- Even Parity: The parity bit is set so that the total number of '1' transmitted is even
- Odd Parity: The parity bit is set so that the total number of '1' transmitted is odd
- If received total number of '1' is not as expected, an error occurred
 - One error (bit flip) can be detected
- Parity is optional
 - Helps in noisy mediums
 - Slows down data transfers



Bidirectional Asynchronous Serial

- For bidirectional communication we need 2 lines
 - The TX of A is connected to the RX of B
 - The RX of A is connected to the TX of B
- **Full-duplex**: both can transmit and receive simultaneously
- **Half-duplex**: devices take turns in transmitting and receiving
- **Simplex**: unidirectional only



Implementation of Asynchronous Serial

- **TTL Serial** (Transistor-Transistor Logic)
 - '1' is encoded as positive voltage, typically 5V or 3.3V
 - '0' is encoded as zero voltage (ground)
- **RS-232 Serial**
 - '1' is encoded as negative voltage, typically -12V
 - '0' is encoded as positive voltage, typically +12V
- **UART** (Universal Asynchronous Receiver-Transmitter)
 - Hardware implementation of asynchronous serial communication
 - Creates the frames and controls the physical lines
- **Software UART** (bit-banging)
 - Software implementation directly controlled by the processor
 - Inefficient, but an option if UART not available in MCU

Example of a UART Peripheral

- **CozIR-A:** A CO₂ Sensor
- Supports a 9600 baud rate, 8N1 serial interface
- Provides a ASCII-based command/control interface over serial



CONTROL INTERFACE TIMING - UART MODE

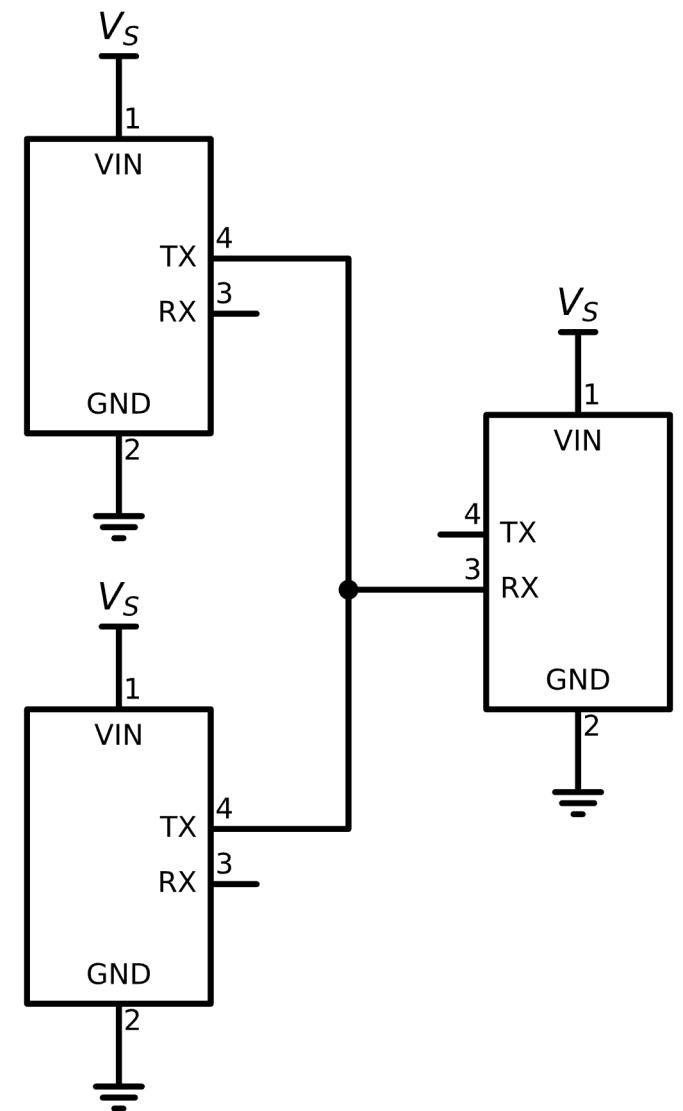
PARAMETER	SYMBOL	MIN	TYP	MAX	UNIT
Baud Rate			9600		Bits/s
Data Bits			8		
Parity			None		
Stop Bits			1		
Hardware Flow Control			None		

PIN-OUT DESCRIPTION: CozIR®-A (Both Types)

PIN	NAME	TYPE	DESCRIPTION
1	GND	Supply	Sensor ground
2	NC	Unused	Do Not Connect
3	VDD	Supply	Sensor supply voltage
4	GND	Supply	Sensor ground
5	Rx_In	Digital Input	UART Receive Input
6	GND	Supply	Sensor ground
7	Tx_Out	Digital Output	UART Transmit Output
8	NITROGEN_ZERO	Digital Input	Set low to initiate a Zero in Nitrogen Calibration Cycle
9	ANALOGUE_OUTPUT	Analogue Output	CO ₂ Level (Optional)
10	FRESH_AIR_ZERO	Digital Input	Set low to initiate a Zero in Fresh Air Calibration Cycle

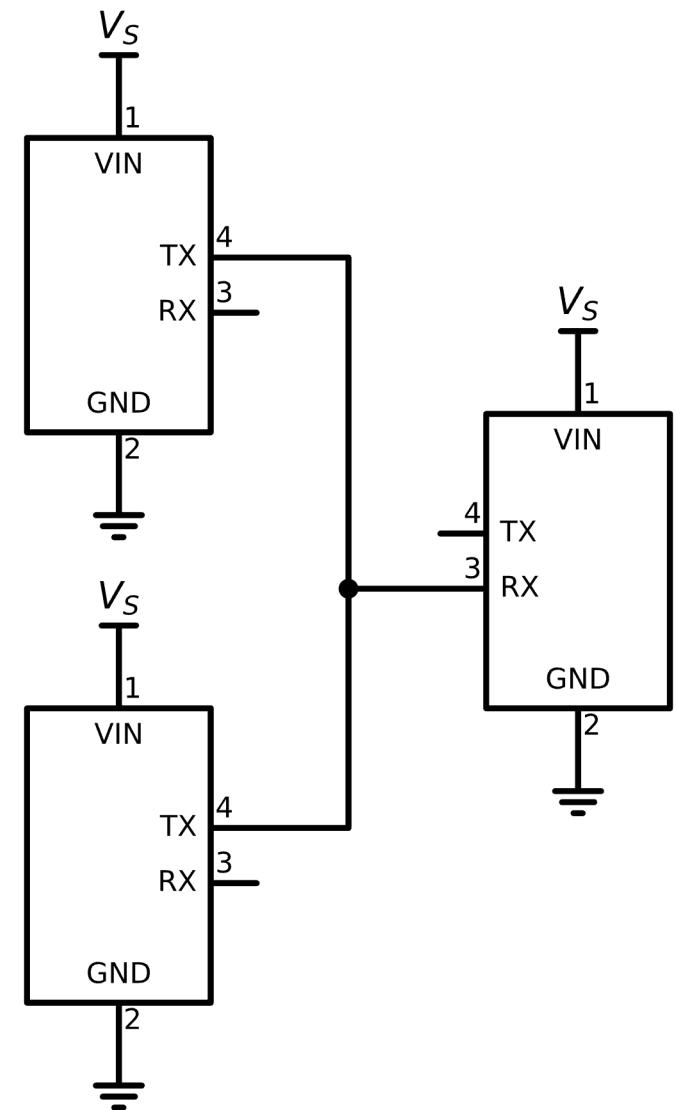
Image source: CoZIR Datasheet by GSS

Can UART be shared?



UART cannot be shared

- Asynchronous Serial (UART) does not support multiple devices
- It is designed for **two devices** to communicate
- Multiple UART transmitters must not share the same line
 - Output is push-pull
 - Shorts and hardware damage are possible
 - Communication will fail



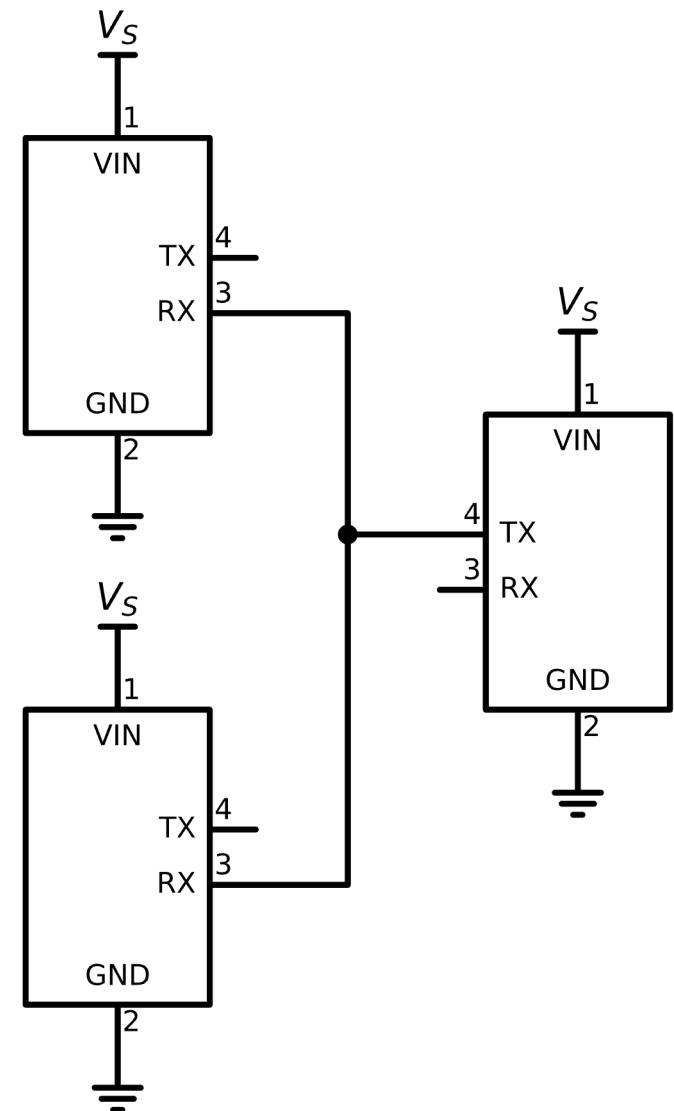
R:Reads

T:transmit (seeds)



UART cannot be shared

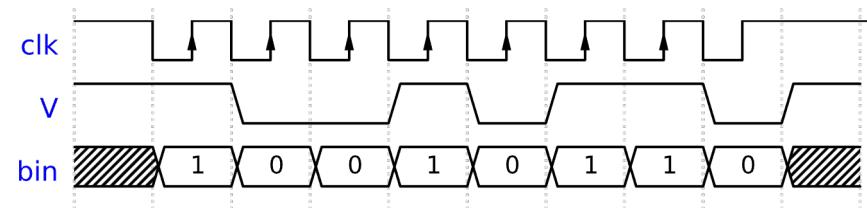
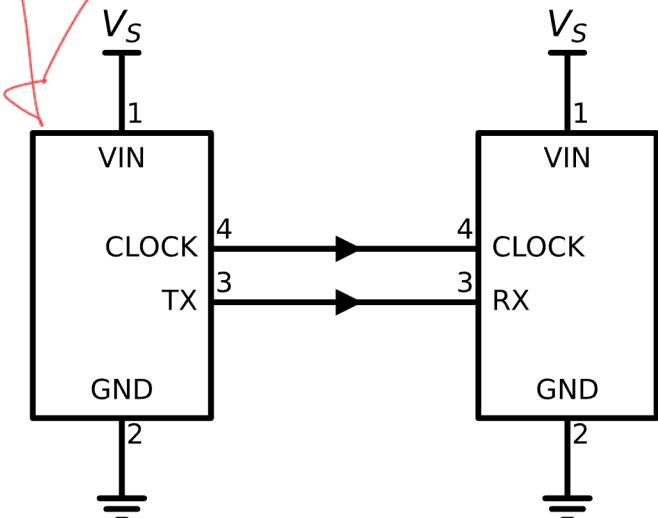
- Asynchronous Serial (UART) does not support multiple devices
- It is designed for **two devices** to communicate
- Multiple UART receivers is safe but...
 - It is impossible for the transmitter to select a receiver
 - All receivers will receive everything
 - Not proper use of the standard
 - **It may work on some applications**



Synchronous Serial Communications

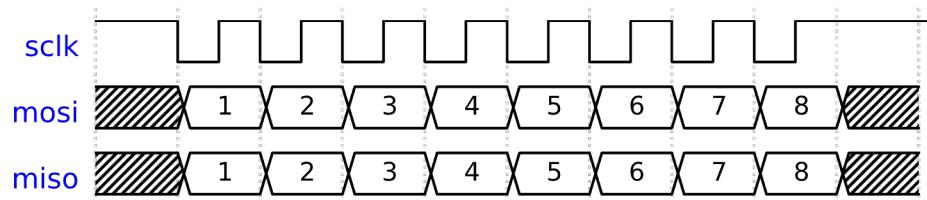
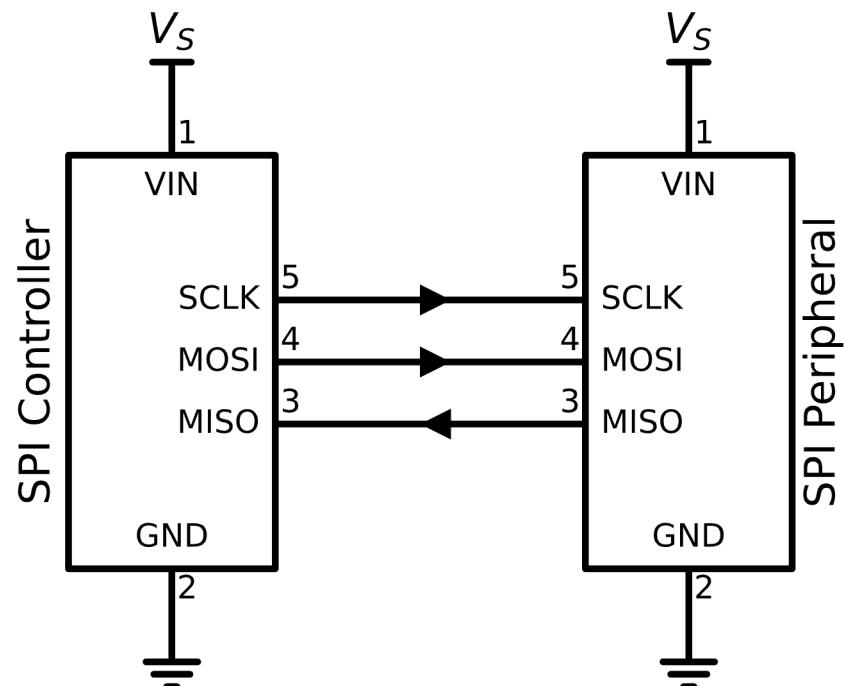
- One device, the controller, generates a clock signal
- All devices use this clock signal to synchronise their transmitters and receivers with the controller
 - Transmissions (i.e. driving the line) occur on the falling edge of the clock (high to low)
 - Receptions (i.e. reading the line) occur on the rising edge of the clock (low to high)
 - Or vice versa
- Advantages
 - No need to pre-agree the baud rate
 - No overhead of synchronisation bits
 - Requires cheaper hardware than UART

DSS / master device



SPI (Serial Peripheral Interface)

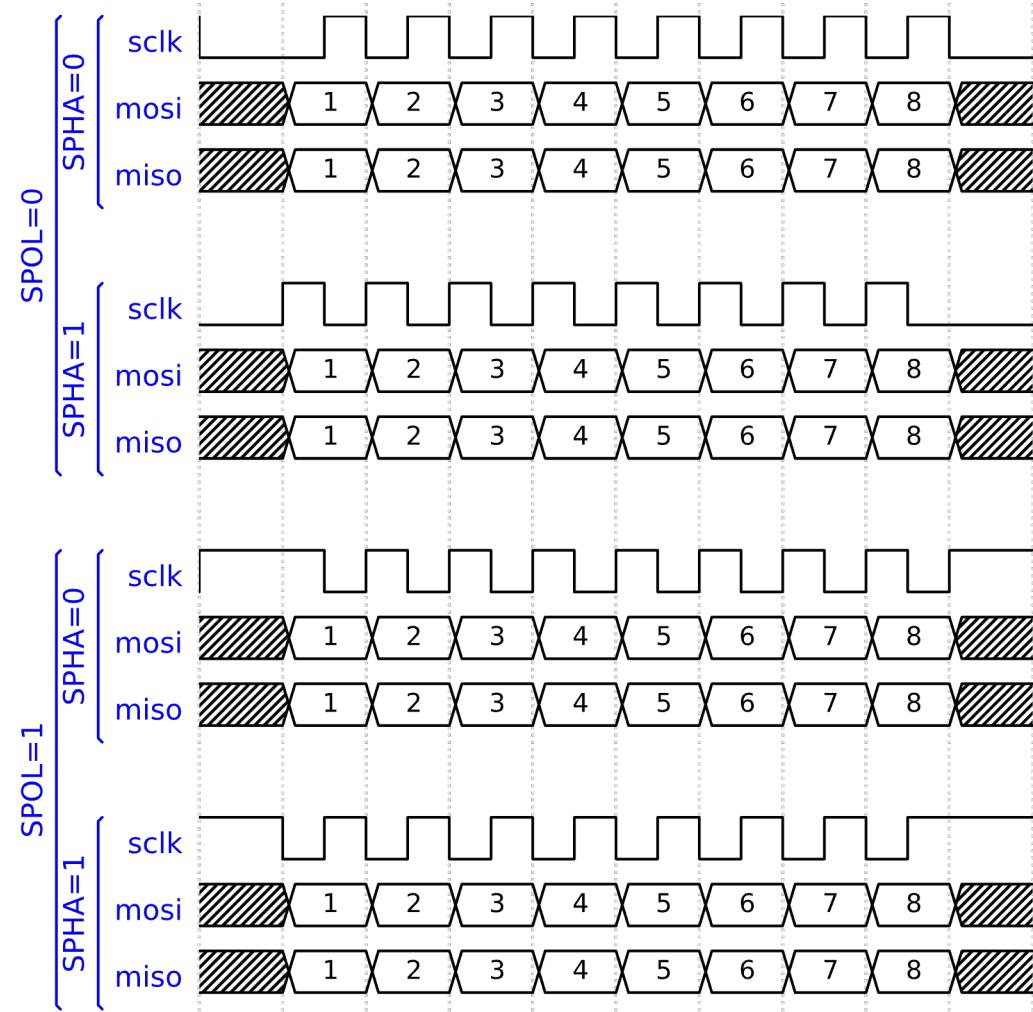
- SCLK (Serial Clock)
 - The SPI controller (or master) drives the clock
 - The controller selects the frequency of the clock dictating the rate of communication (~MHz)
 - Must be supported by SPI peripheral(s)
- MOSI (Master Out, Slave In)
 - Data from SPI controller to SPI peripheral(s)
- MISO (Master In, Slave Out)
 - Data from SPI peripheral(s) to SPI controller
- Full-duplex: On each clock cycle a bit is written/read on MOSI and a bit is written/read on MISO
 - This is maintained even when only one-directional data transfer is intended



SPI Modes

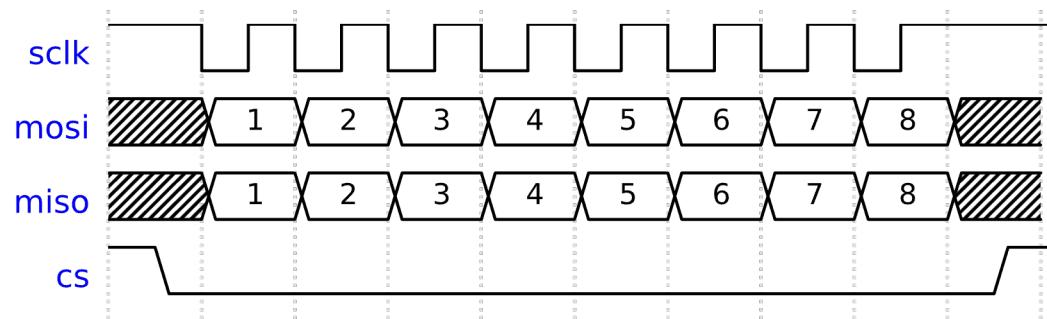
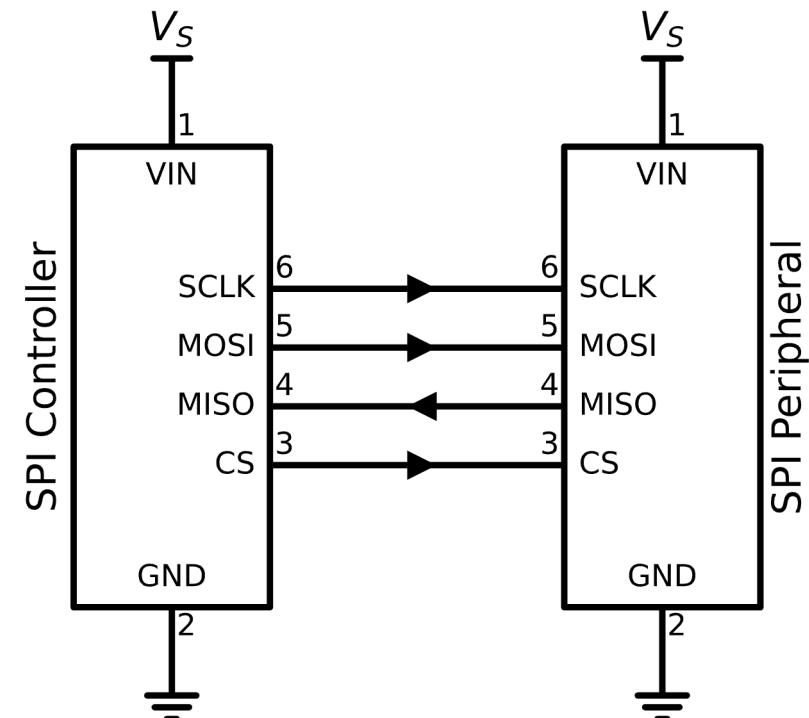
- CPOL=0: clock idles at 0, the leading edge is a rising edge, and the trailing edge is a falling edge
- CPOL=1: clock idles at 1, the leading edge is a falling edge, and the trailing edge is a rising edge
- CPHA=0: the data changes on the trailing edge, the data is read on the leading edge
- CPHA=1: the data changes on the leading edge, the data is read on the trailing edge

SPI Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



SPI: Chip Select

- Chip Select (CS): The controller uses it to select the peripheral to communicate with
- When the peripheral detects a CS transition from high to low, it activates itself and gets ready to transmit/receive
- When the peripheral detects a CS transition from low to high, it puts MISO in Hi-Z (deactivates the output)
- When deactivated, the peripheral ignores the SCLK and MOSI
- Recommended to have a pull-up resistor on CS for MCUs that boot with their GPIOs in Hi-Z

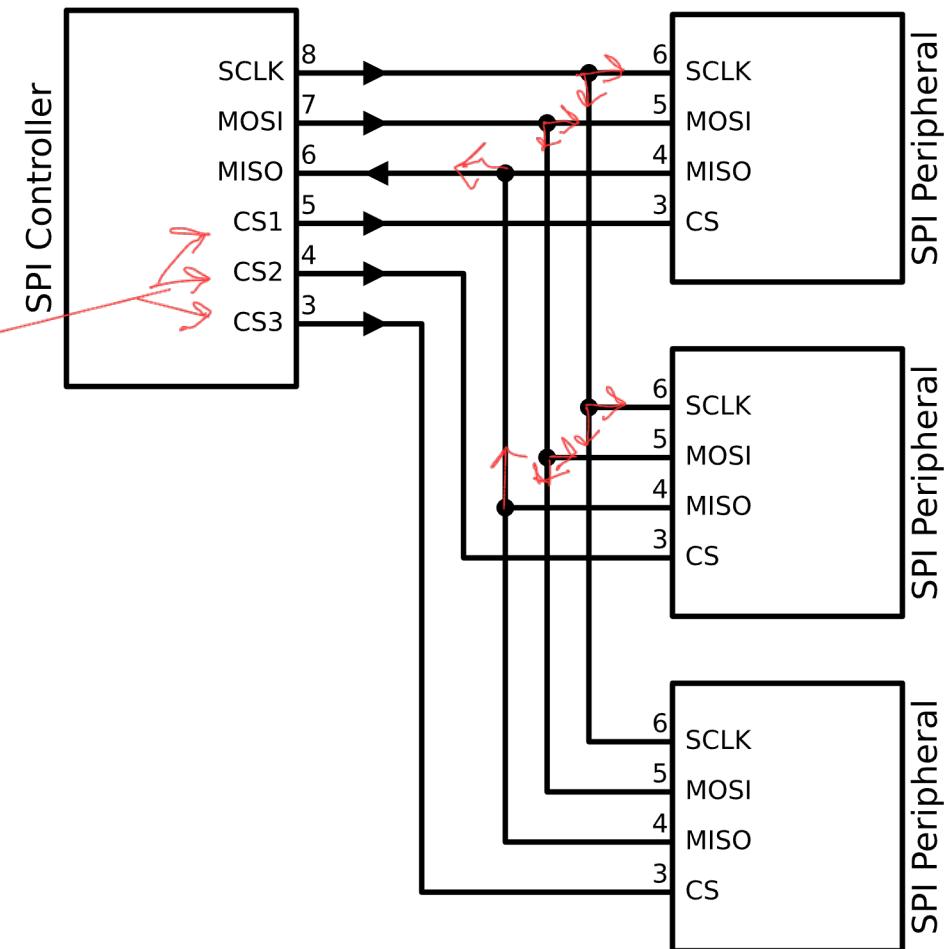


All clocks are connected.



SPI: Multiple Peripherals

- Multiple SPI peripherals are supported
- All are controlled by the SPI controller by selecting them via the respective CS output
- The SPI controller must make sure to have only one SPI peripheral enabled at a time
- Data transfers are only possible between the SPI controller and an SPI peripheral
- Requires 3 GPIOs + 1 GPIO per SPI peripheral
- Only one device can be the SPI controller



Not as common

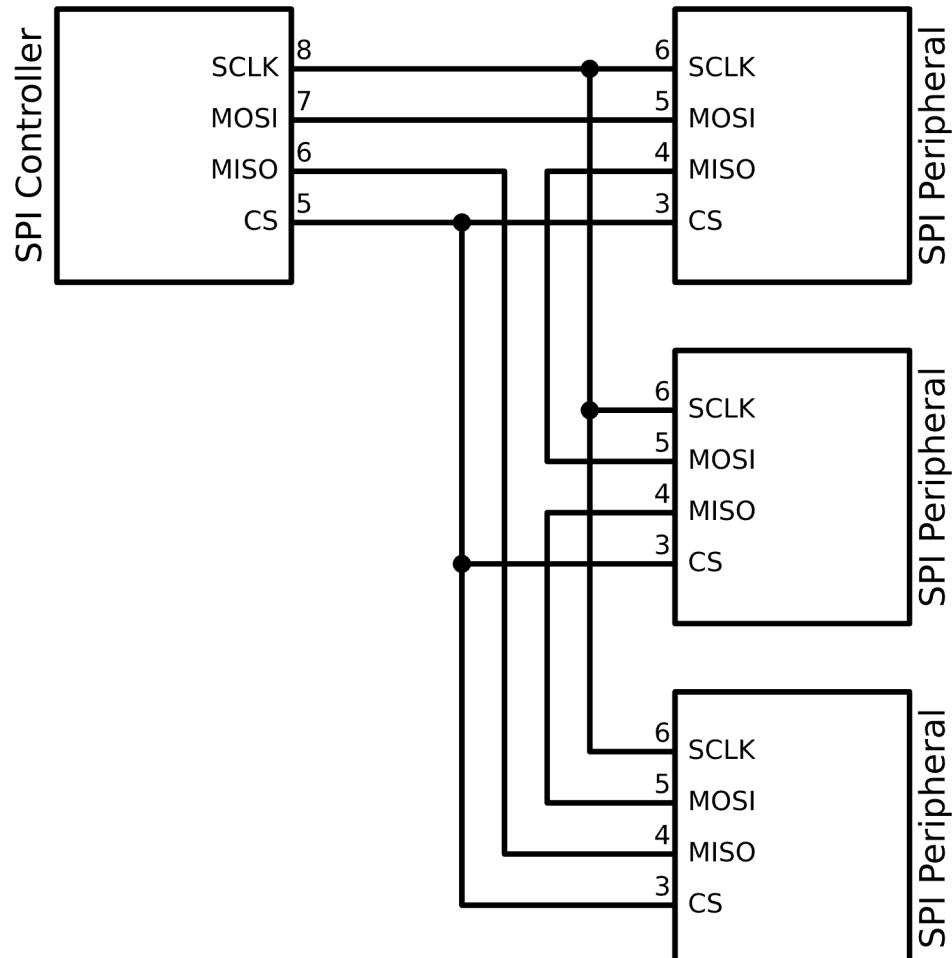


*de er linket sammen, de repeater
signalet.*

SPI: Daisy Chain

- SPI controller and peripherals form a circle
 - The output of the first peripheral goes at the input of the second, etc
 - The output of the last peripheral goes to the input of the controller
- Peripherals forward data to the next
- All peripherals are enabled together for a data transfer
- Less pins required but all peripherals need to be active, even if the controller needs to talk to one

delay med scale



Example of an SPI Peripheral

- ADXL362:** A 3-axes accelerometer
- Provides an SPI interface
- Provides a command/control interface over SPI

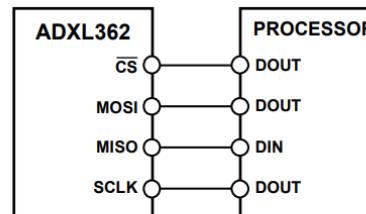


Figure 35. 4-Wire SPI Connection Diagram

10776-032

SPI COMMANDS

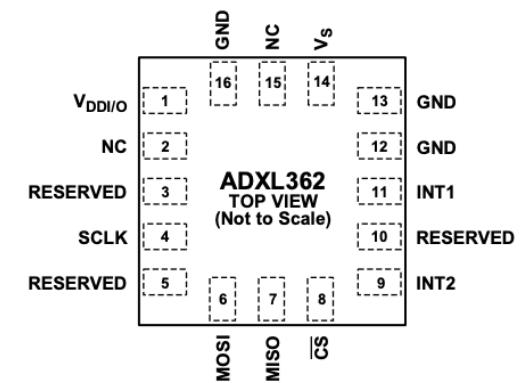
The SPI port uses a multibyte structure wherein the first byte is a command. The [ADXL362](#) command set is

- 0x0A: write register
- 0x0B: read register
- 0x0D: read FIFO

Table 11. Register Summary

Reg	Name	Bits	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset	RW
0x00	DEVID_AD	[7:0]									0xAD	R
0x01	DEVID_MST	[7:0]									0x1D	R
0x02	PARTID	[7:0]									0xF2	R
0x03	REVID	[7:0]									0x01	R
0x08	XDATA	[7:0]									0x00	R
0x09	YDATA	[7:0]									0x00	R
0x0A	ZDATA	[7:0]									0x00	R

Image source: ADXL362 Datasheet by Analog Devices



The SPI timing scheme follows CPHA = CPOL = 0.

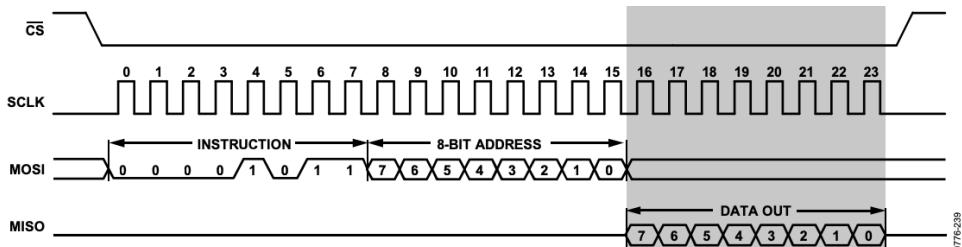


Figure 36. Register Read

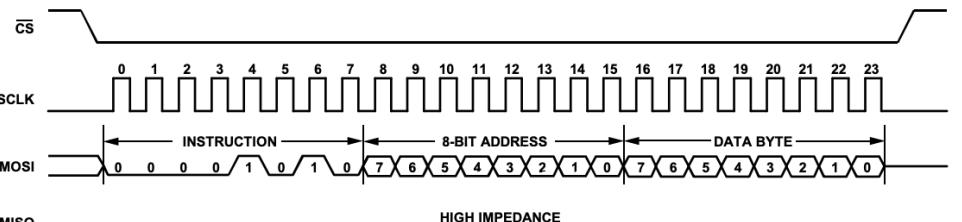


Figure 37. Register Write (Receive Instruction Only)

Synchronous Serial with less wires

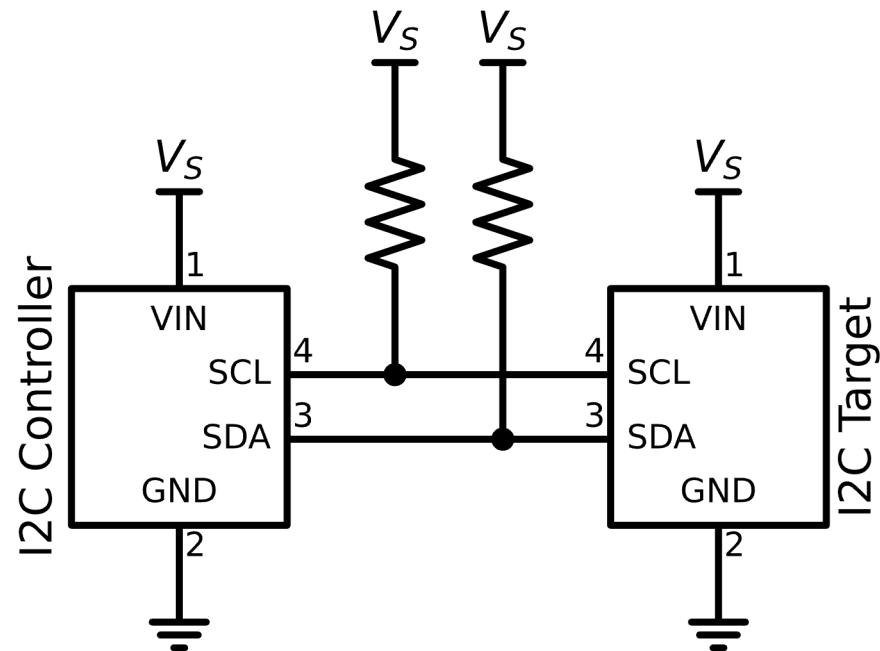
- SPI is faster and cheaper than UART, supports multiple peripherals
 - But requires a lot more GPIO pins, practical limit on the number of peripherals
- How can we reduce the number of required GPIOs?

Synchronous Serial with less wires

- SPI is faster and cheaper than UART, supports multiple peripherals
 - But requires a lot more GPIO pins, practical limit on the number of peripherals
- How can we reduce the number of required GPIOs?
- Eliminate the need for Chip Select (CS)
- Bi-directional data bus

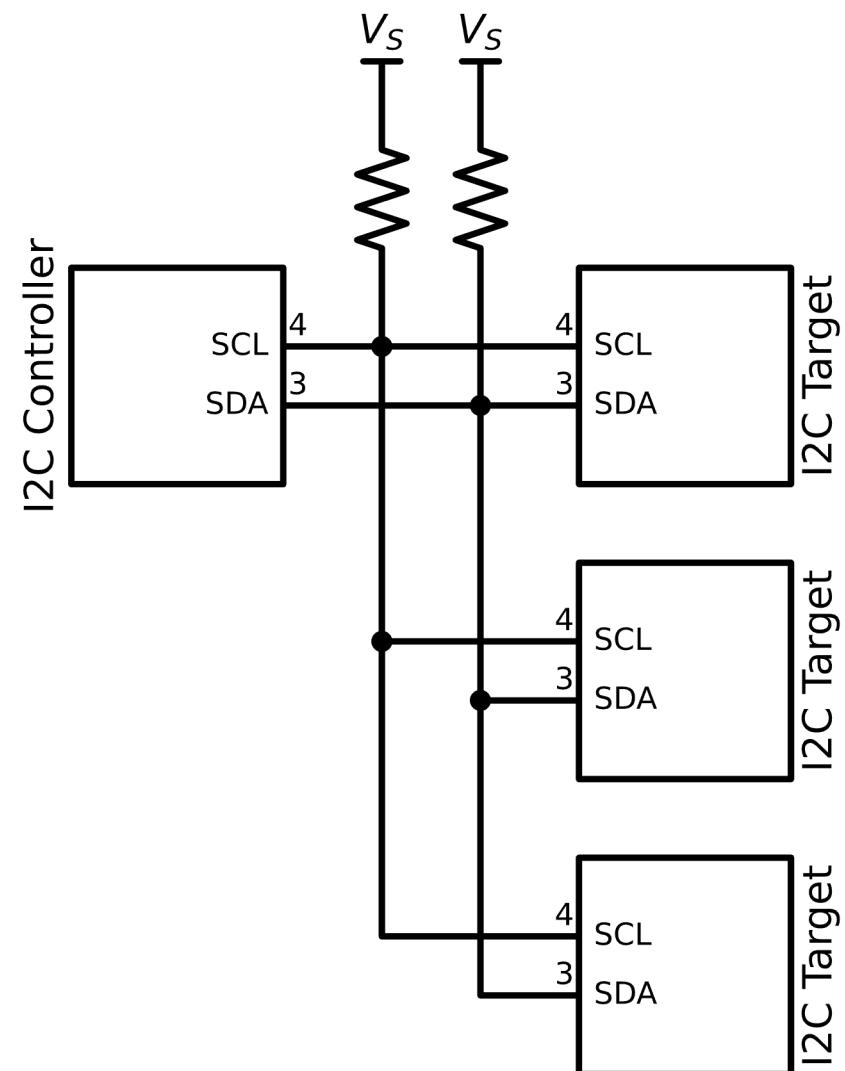
I2C (Inter-Integrated Circuit)

- Only 2 wires: a clock (SCL) and data bus (SDA)
- Outputs are configured as **open-drain**
 - Actively transmit '0' by pulling line to GND
 - Transmit '1' by disabling output (pullup)
 - Different to SPI and UART (push-pull)
- Multiple transmitters can safely share the same bus
- Clock is generated by the controller (100-400 KHz)
 - Faster than UART, slower than SPI
 - Open-drain is slower than push-pull
- Consumes energy when transmitting '0'
 - If $V_S=5V$, each pullup resistor (5K) consumes 5mW when transmitting '0'



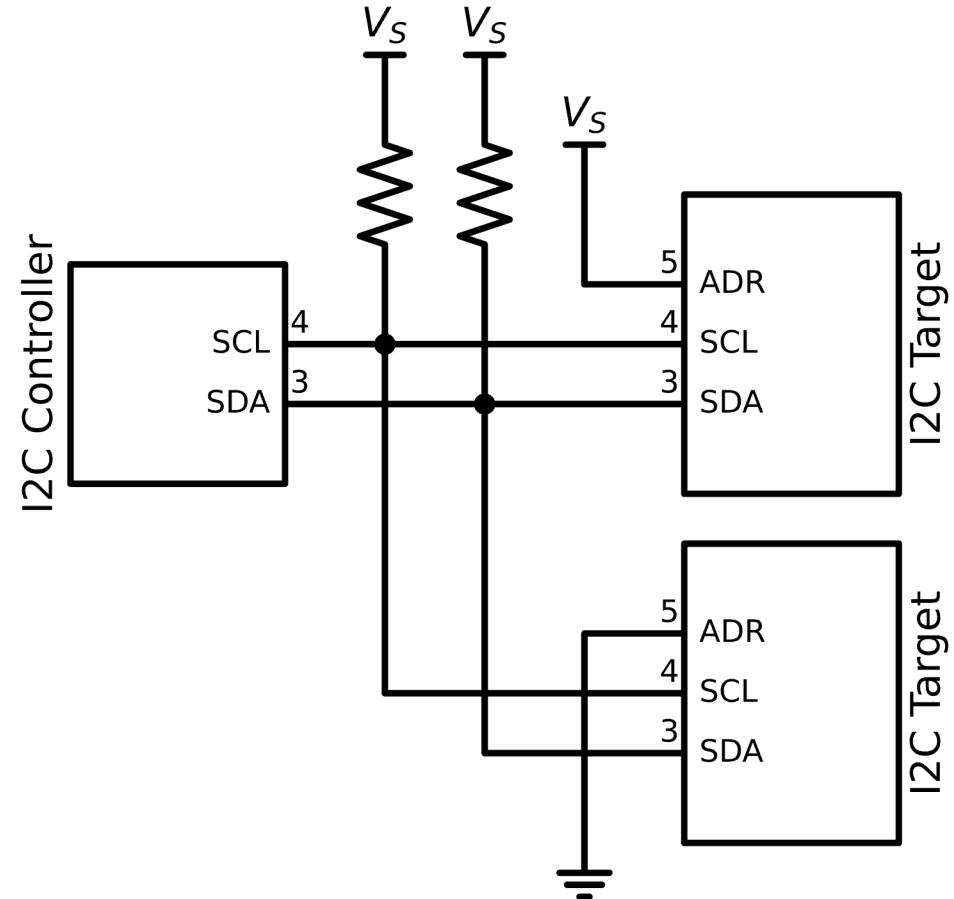
I2C: Multiple Peripherals

- Peripherals, known as I2C targets, simply connect to the SCL and SDA buses
- The CS functionality is replaced by **addresses**
- Each I2C target has a 7-bit address
 - Theoretically, supporting up to 107 peripherals (some addresses are reserved)
- Addresses of peripherals are often fixed
 - Sometimes they are configurable
- It is vital not to have two peripherals with the same address on the same I2C bus
- A variant of I2C with 10-bit addresses also exists



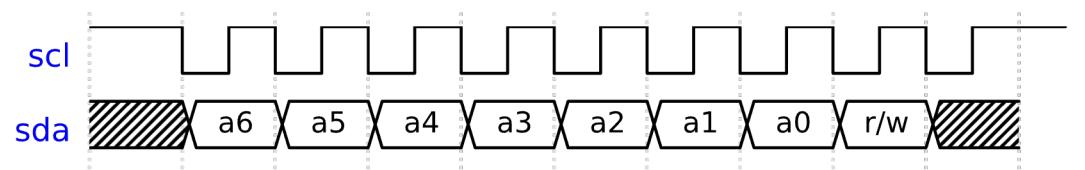
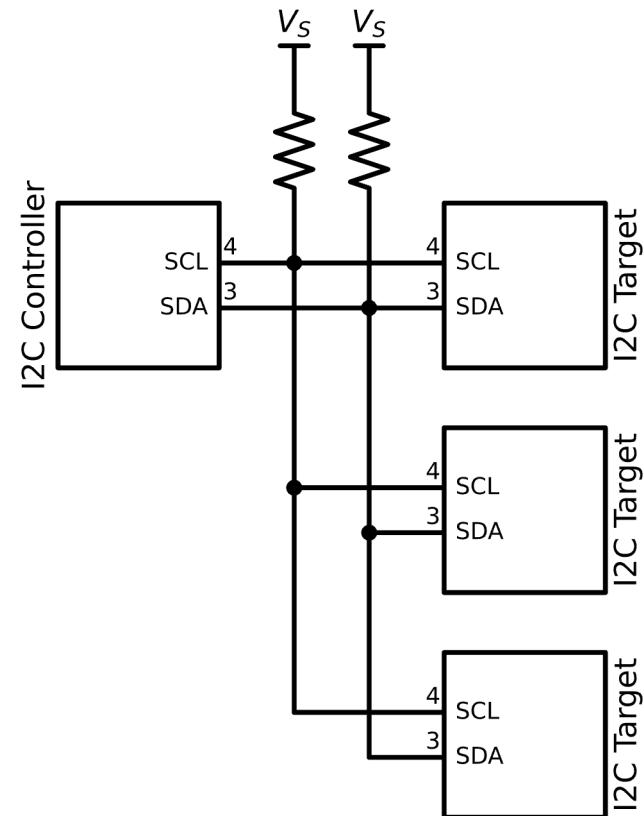
I2C: Configurable Addresses

- Consider that you want to connect two chips of the same sensor on the I2C bus
- The address of the sensor has 6 fixed bits but the 7th is read from an input pin
- For example:
 - A sensor has I2C address: 0b011010x
 - 'x' is read from the ADR pin
 - The top target has address 0b0110101
 - The bottom target has address 0b0110100
- A target may have zero, one, or more configurable address bits



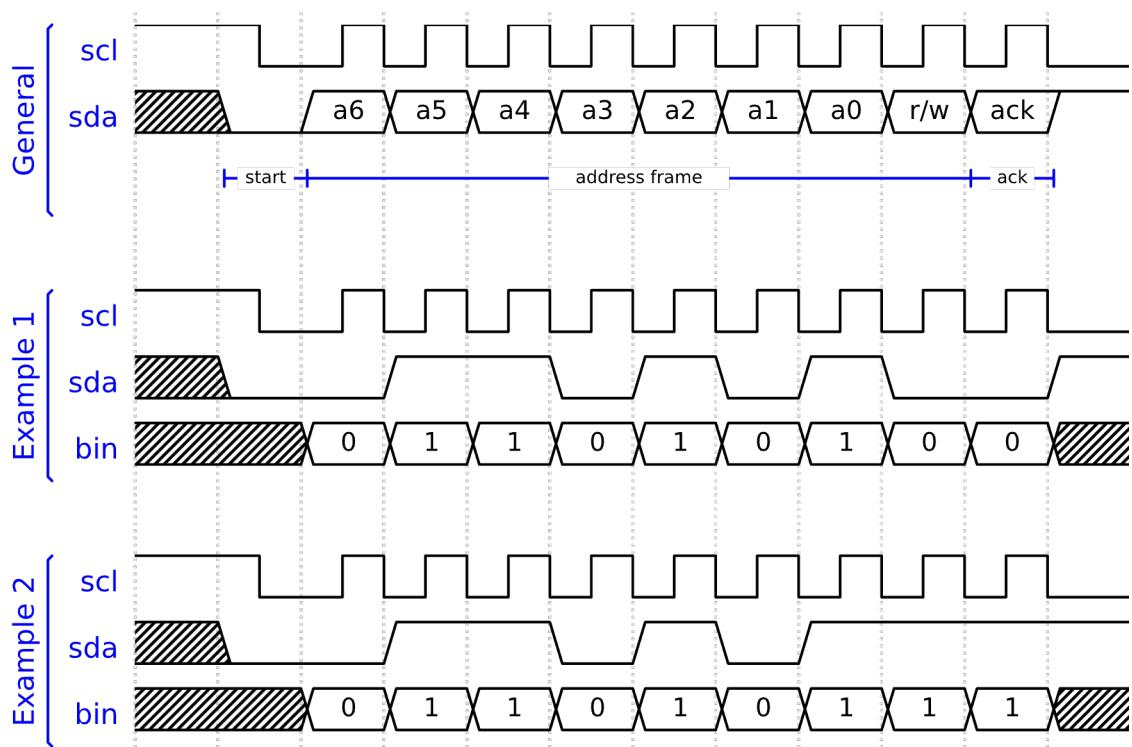
I2C: Address Frame

- All I2C data transfers are initiated by the controller
- The first byte that the controller puts on the line is composed of the **7-bit address** and a bit that indicates if the controller wants to **read (1)** from or **write (0)** to the peripheral
- Peripherals whose I2C address does not match will ignore it
- Example:
 - Peripheral 1 has I2C address: 0b0110101
 - To write to it the controller writes: 0b01101010 (0x6A)
 - To read from it the controller writes: 0b01101011 (0x6B)



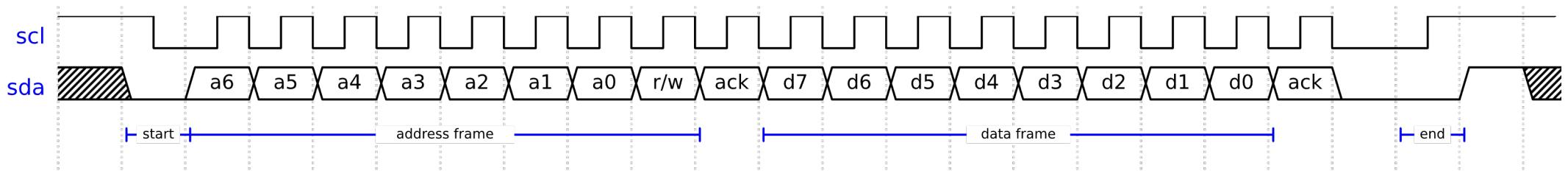
I2C Protocol: Start Condition and Acknowledgement

- **Start Condition:** to initiate a transaction, the controller device leaves SCL high and pulls SDA low
- All peripherals get ready to receive
- The controller writes the **address frame** byte indicating the target address and its intention to read or write
- The controller disables its output and gives control of SDA to the target
- **ACK Bit:** The target pulls SDA low to indicate a successful reception
- If error, the pullup will keep line high
- Example 1: target address 0x0110101, write operation, target acknowledges
- Example 2: target address 0x0110101, read operation, error



I2C Protocol: Data Frame and Stop Condition

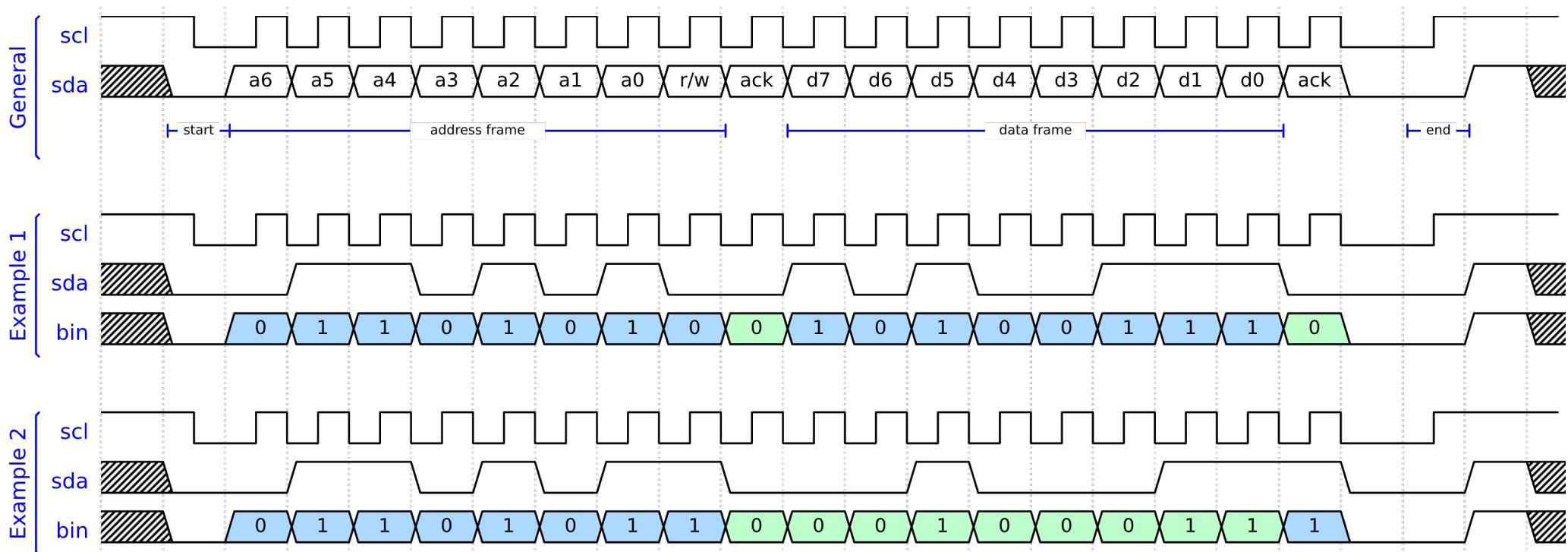
- After the successful transmission of the address frame, the controller continues generating clock pulses and the data transmission begins
- The SDA is controlled by either the controller or the target, according to the W/R bit
- Each **data frame** is a byte (8 bits) followed by an acknowledgement bit from the receiver
 - One or more data frames can be transmitted
- If the last data frame is a read, the controller sends a negative acknowledgement (NACK) to indicate it intends to terminate the transfer
- **Stop Condition:** The controller ends the transaction by low to high transition on the SDA line after a low to high transition on the SCL line



I2C: Transaction Examples

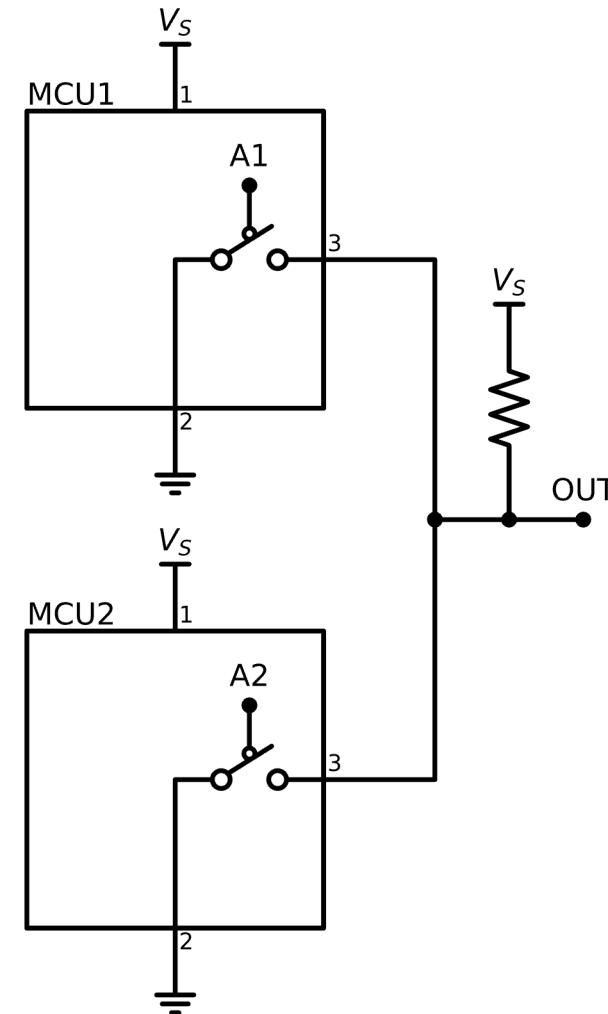
- Example 1: Controller writes a byte to target
- Example 2: Controller reads a byte from target

Blue: Controller controls the SDA line
Green: Target controls the SDA line



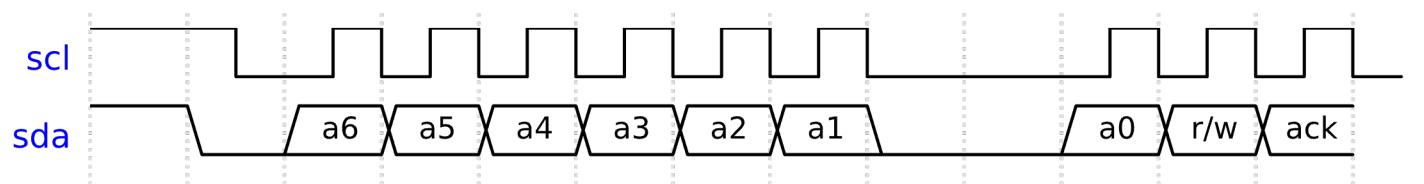
Open-Drain Output

- Open-Drain mode has two states:
 - Logical '0' (i.e., low voltage)
 - Hi-Z (i.e., disconnected)
- Sets the line to '0' **actively**
- Sets the line to '1' **indirectly** by disconnecting the output and letting the pullup resistor raise it to '1'
- The line is '1' only if nobody transmits '0'



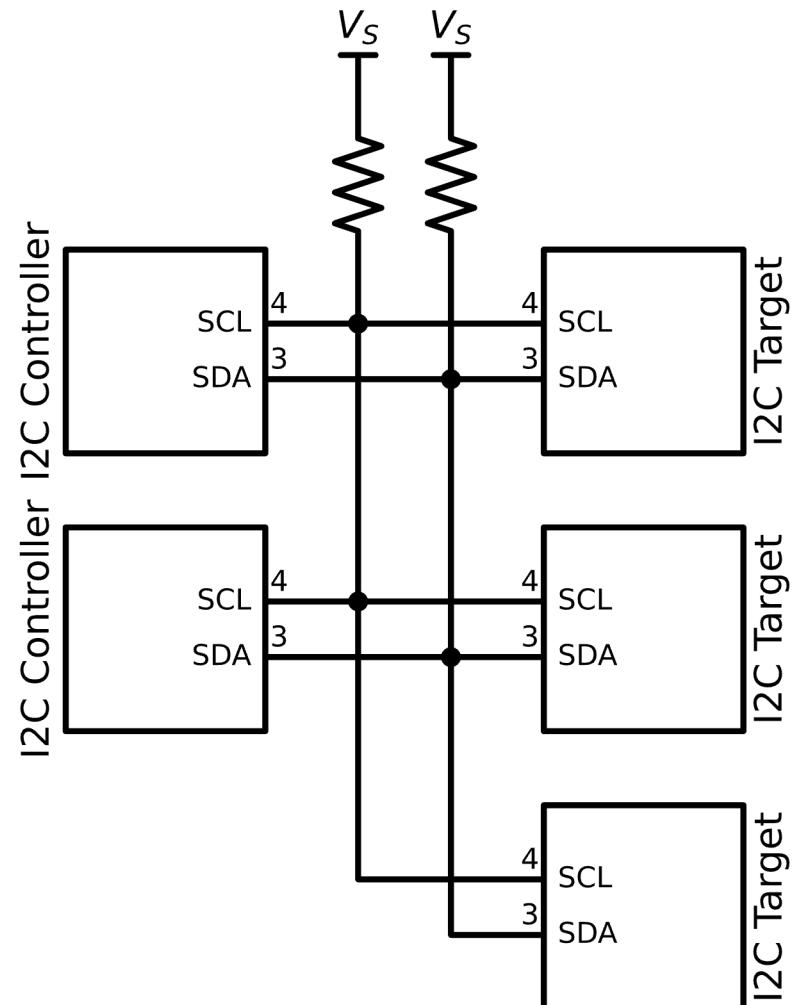
I2C: Clock Stretching

- In an open-drain bus, like I2C, if two devices control the line at the same time, '0' always wins
- A target that needs more time to process data, can pull the SCL line to '0'
- The controller cannot transmit '1' to generate a new clock pulse until SCL is released
- Essentially, the target temporarily stops time



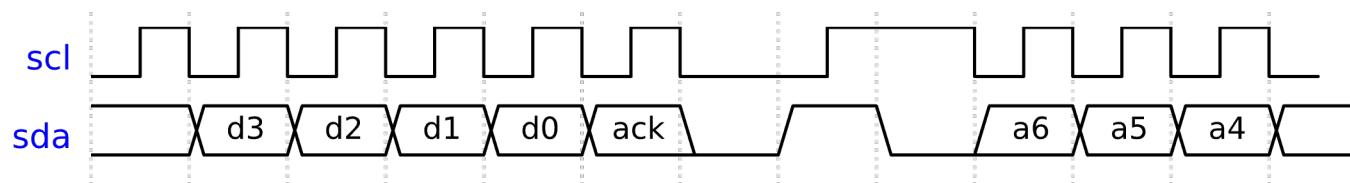
I2C: Multiple Controllers

- I2C supports multiple controllers due to the open-drain bus
 - Controllers monitor the bus for start/stop conditions and do not initiate a start condition
 - However, two controllers may start a transmission at the same time
- **Arbitration**
 - If two devices control the line at the same time, '0' always wins
 - When a device transmits '1' (indirectly by disabling its output), it can read the state of the line in parallel
 - If the line remains '0', it means somebody else is active at the same time
 - The first controller that detects an inconsistency backs off and lets the other finish
 - The last to transmit '1' wins



I2C: Repeated Start Condition

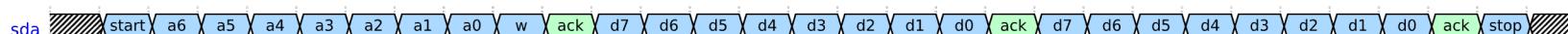
- The controller can make transactions back-to-back by generating a repeated start condition
- First, the controller raises both SDA and SCL high avoiding generating a stop condition
- Then, it generates a new start condition, by pulling SDA down before the SCL
- Other I2C controllers do not interfere waiting for a stop condition



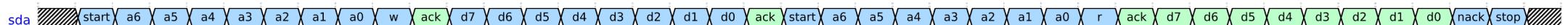
I2C: Write/Read Target Register

Blue: Controller controls the SDA line
Green: Target controls the SDA line

- Peripherals typically organise their internal memory in registers
- Each register has an address and a value
- To read/write a register, the controller needs to write the register address it wishes to read/write
- Write to target register (3 bytes):
 - Start -> Target Address (W) -> Register Address -> Register Value -> Stop



- Read register from target (4 bytes):
 - Start -> Target Address (W) -> Register Address -> Re-Start -> Target Address (R) -> Register Value -> Stop



Example of an I2C Peripheral

- TMP1075: A temperature sensor
- Provides an I2C interface
- Configurable I2C address: 10010xx
- Provides a command/control interface over I2C

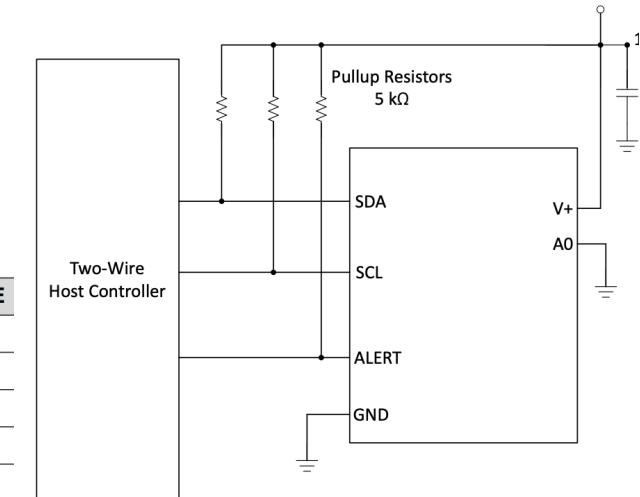


Table 9-5. TMP1075 Register Map

ADDRESS	TYPE	RESET	ACRONYM	REGISTER NAME
00h	R	0000h	TEMP	Temperature result register
01h	R/W	00FFh	CFGR	Configuration register
02h	R/W	4B00h	LLIM	Low limit register
03h	R/W	5000h	HLIM	High limit register
0Fh ⁽¹⁾	R	7500h	DIEID	Device ID register

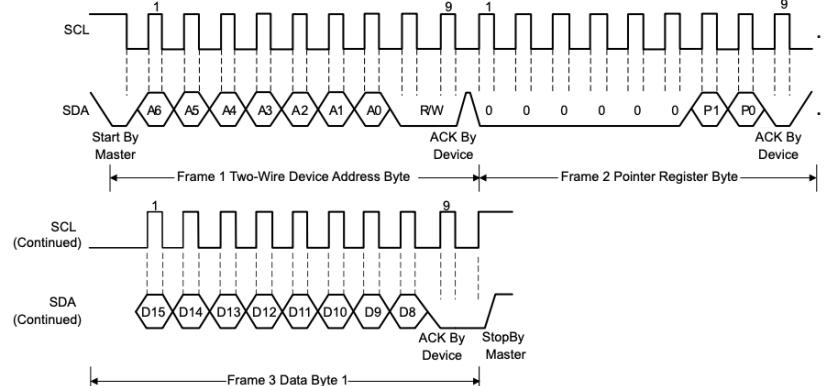


Figure 9-4. Two-Wire Timing Diagram for Write Single Byte Format

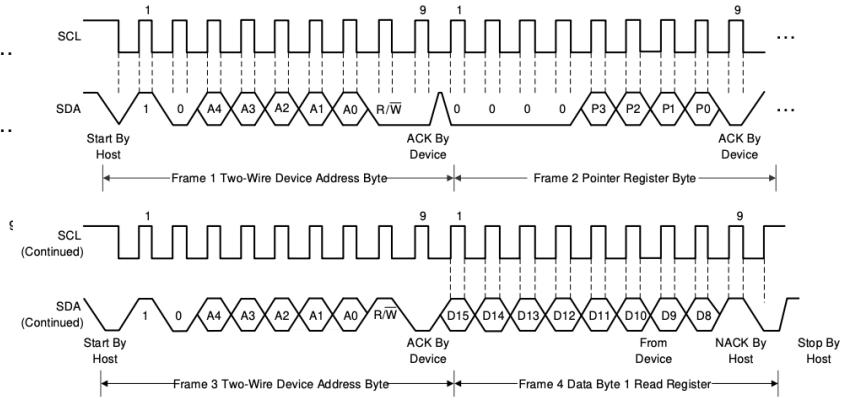


Figure 9-6. Two-Wire Timing Diagram for Read Single Byte Format

Image source: TMP1075N Datasheet by Texas Instruments

Summary

Property	UART	SPI	I2C
No. Peripherals	Point-to-Point	Many (GPIO)	Many (address)
Multi-Controller	No	No	Yes
Rate (typical)	10-100 Kbps	1-20 Mbps	100-400 Kbps
GPIOs Required	2	3 + 1 per peripheral	2
Simultaneous Tx/Rx	Depends on hardware	Full-duplex	Half-duplex
Overhead	2-4 bits per byte	0	11 + 1 per byte
Energy Requirements	++	+	+++

I3C

- Aim to be the best of both worlds
- Primary controller controls the clock
 - Secondary controller may exist
- Switches from open-drain to push-pull on the SDA whenever possible
 - After start condition controller switches the SDA to push-pull, allowing the clock to be increased up to 12.5 MHz
- Pullup resistors are provided by the I3C controller
- Clock stretching and 10-bit addresses not supported

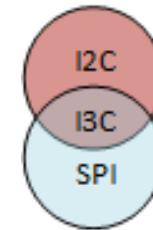
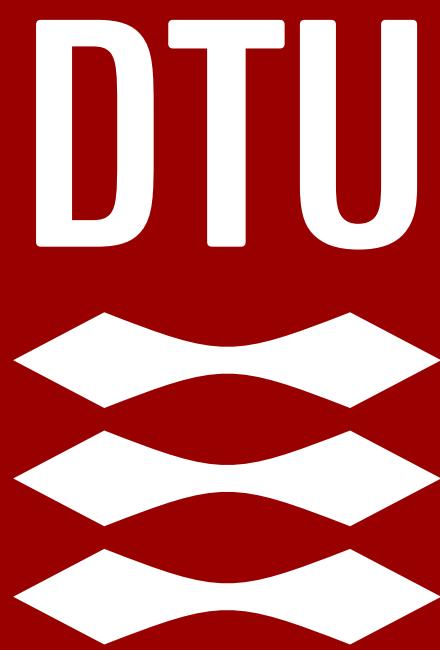


Image source: Wikipedia



Networked Embedded Systems

Week 7: Wired Networked Embedded Systems

Xenofon (Fontas) Fafoutis

Professor

xefa@dtu.dk

www.compute.dtu.dk/~xefa

From Intra-Board Communication to Local Networks

- Abstraction layers: The OSI reference model

Layer	Data Unit	Function
7	Application	Data
6	Presentation	Data
5	Session	Data
4	Transport	Segment
3	Network	Packet
2	Data Link	Frame
1	Physical	Bit

Physical Layer (PHY)

- The Physical Layer defines means for transmitting raw bits over a physical medium
- It is typically implemented in hardware (PHY chip)
- It abstracts the physical aspects of communication
 - Modulation, demodulation, bit synchronisation, line coding, multiplexing, filtering...
- All these procedures are abstracted to:
 - Bits transmitted on one end and received to the other end



RTL8201 Ethernet PHY chip

Image source: Wikipedia

Betyder bare der er mulig for flere måde at skelne mellem 0 og 1-
Det kan være at hvis en volt i x er = x-1 så er det 1 men hvis den
ikke er lig er det 0 (NRZ-S)

Line Coding

- Defines how the logical '0' and '1' are physically represented on the line
- NRZ (Non-Return-to-Zero): signal does not return to zero at the middle of the bit
 - NRZ-L: '0' is zero voltage, '1' is positive voltage
 - NRZ-M or NRZ-I: '0' is same voltage, '1' is a transition
 - NRZ-S: '1' is same voltage, '0' is a transition
- RZ (Return-to-Zero)
 - '0' is zero voltage, '1' is positive voltage returns to zero at the middle of the bit

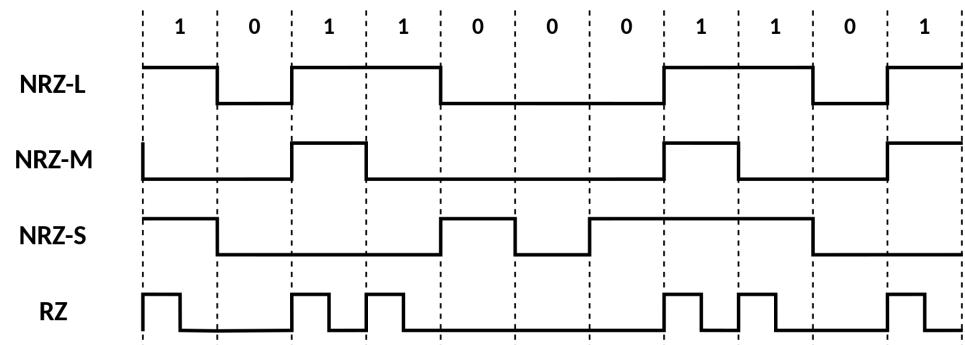


Image source: Wikipedia

Manchester Coding

- Encodes logical value as a transition in the middle of the bit
- Original:
 - '0' is a low-to-high transition
 - '1' is a high-to-low transition
- IEEE 802.3 (Ethernet)
 - '0' is a high-to-low transition
 - '1' is a low-to-high transition
- Self-clocking: Digital inputs can detect transitions without the need of a clock
- Requires double the bandwidth

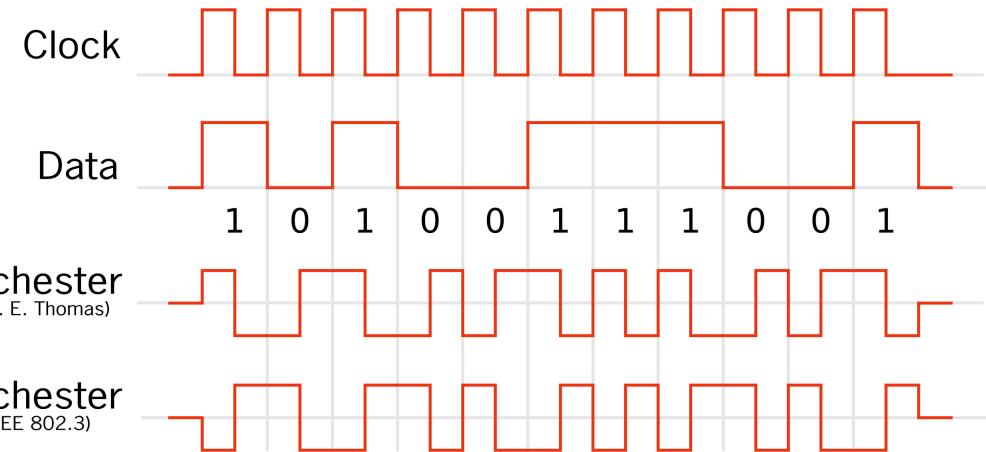
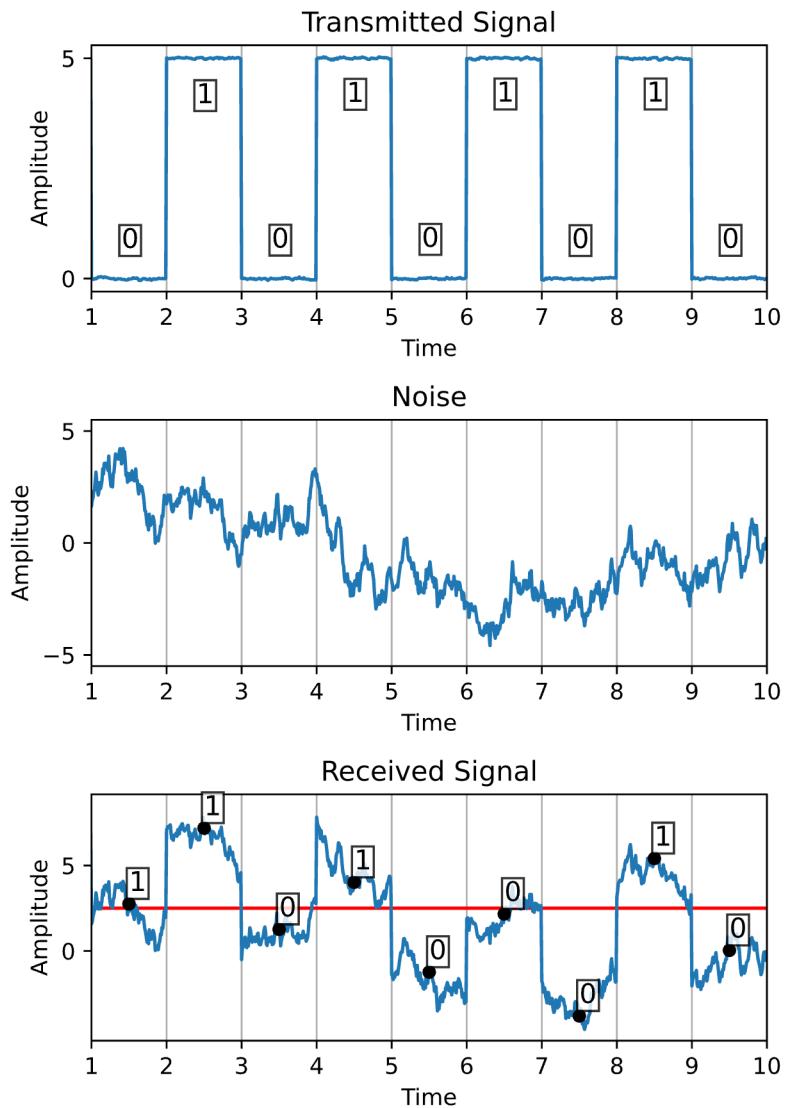
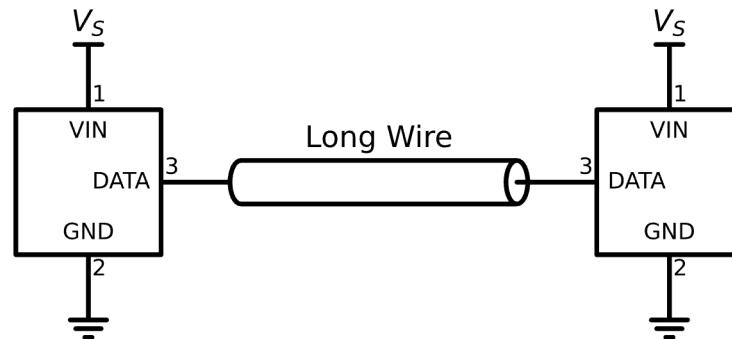


Image source: Wikipedia

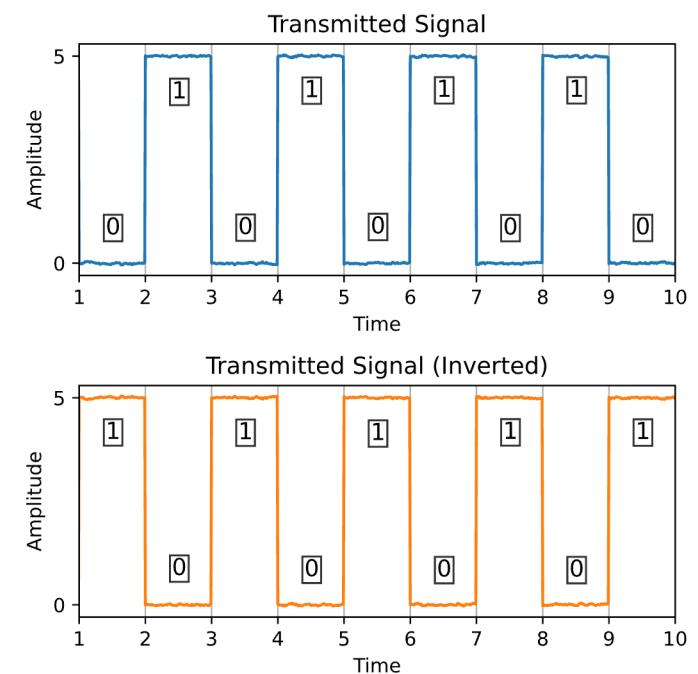
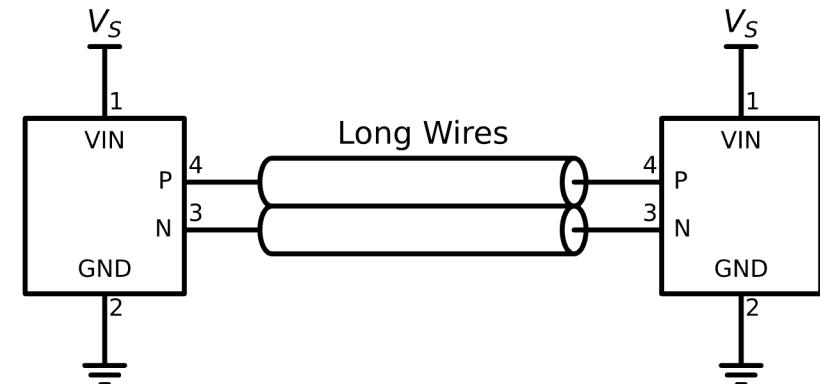
Transmission over Long Wires

- Wires are imperfect conductors
 - They have some small resistance (mOhm/m)
 - They have a small capacitance
- These imperfections are negligible in short wires but have side-effects when the wires are longer
 - Wires pick up electromagnetic interference (EMI)
 - Ground level at both sides not the same
 - High frequencies get attenuated
- Result: **Bit Errors**

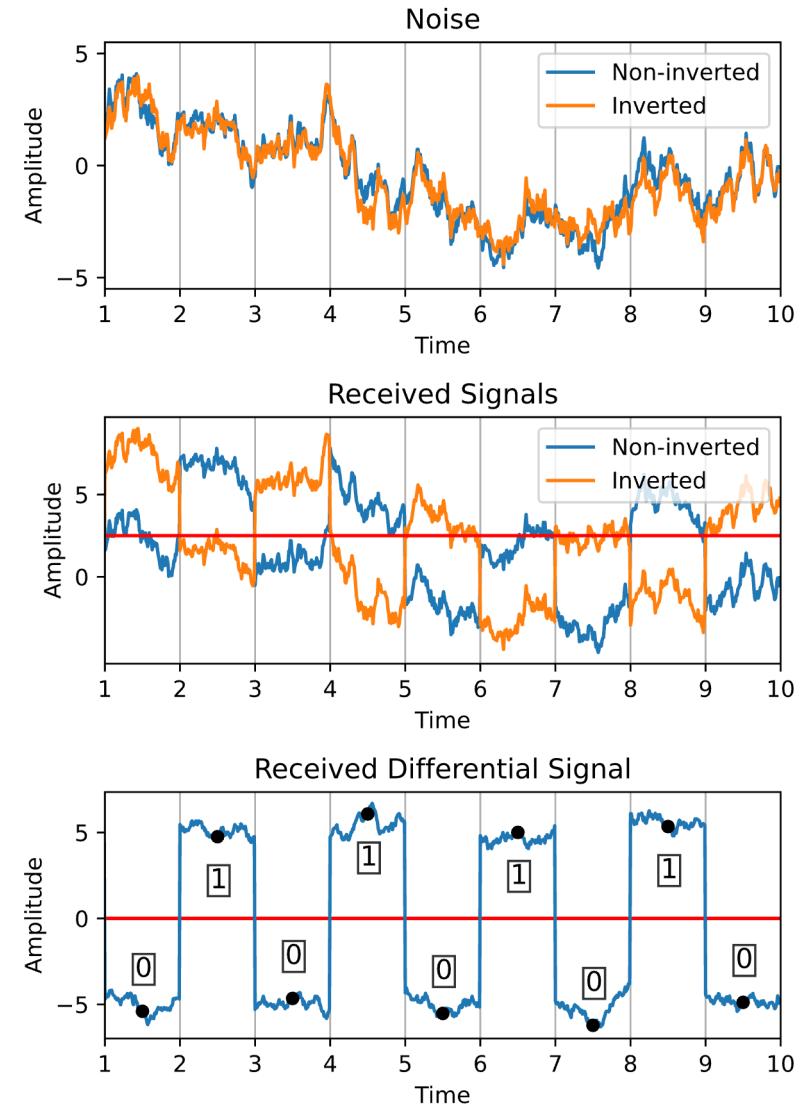
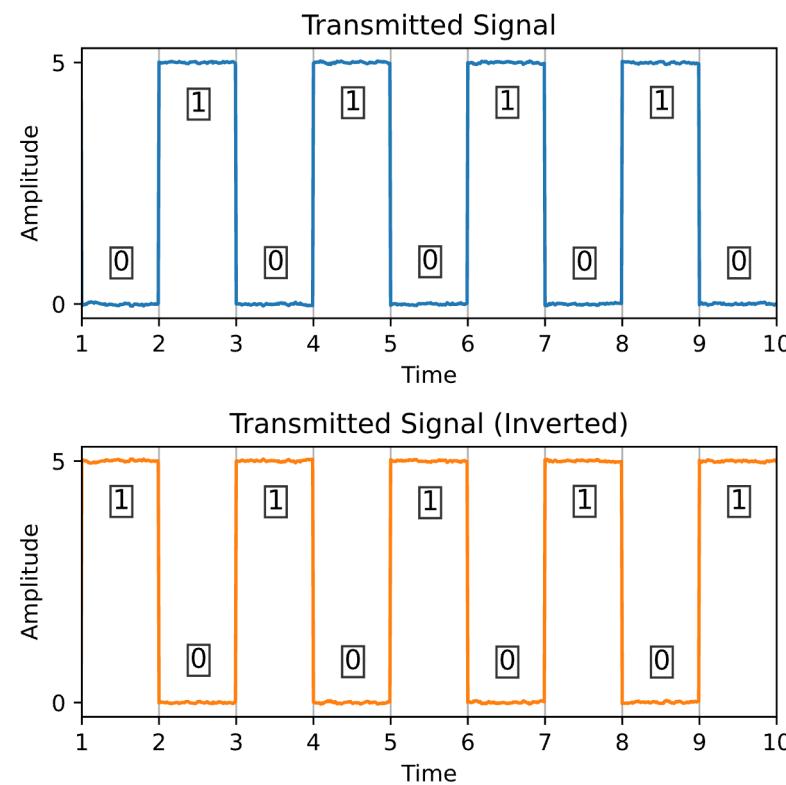


Differential Signalling with Balanced Lines

- We connect the two devices with two wires of the same type and length (balanced lines)
- We transmit the signal and the inverted data signal
 - Non-inverted Signal (P): ‘0’ is GND, ‘1’ is V_s
 - Inverted Signal (N): ‘0’ is V_s , ‘1’ is GND
- Similar to a differential ADC, the receiver measures the difference of the two (P-N)
 - A positive difference means ‘1’ and vice versa
- Advantages
 - Noise cancellation due to balanced lines
 - Additional noise resilience due to double signal voltage at the receiver
 - Enables low-voltage operation (lower consumption, higher speeds)

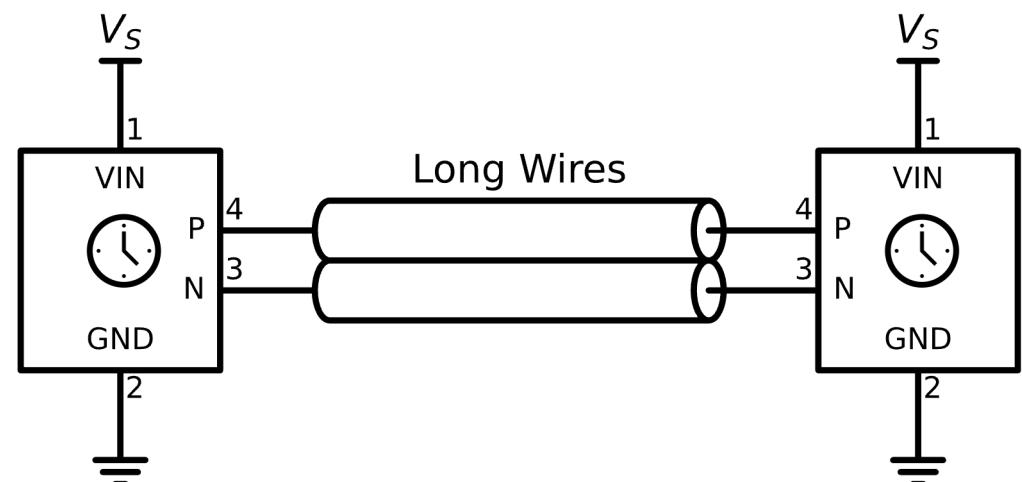


Differential Signal: An Example



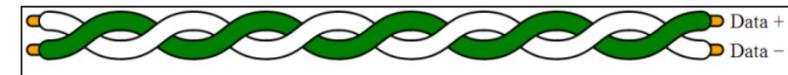
Synchronisation at longer distances

- Devices operate asynchronously using their own clocks (no clock line)
 - Clocks that drift relative to each other
- Transmission rates are pre-agreed and defined by the protocol
- Changes on the line (high-to-low, low-to-high) in predetermined patterns are used for synchronisation
- Long periods without a change on the line can cause de-synchronisation
 - Bit stuffing: adding bits to force a transition



USB 2.0 (Universal Serial Bus)

- Uses differential signalling with balanced lines
 - Lines are twisted to reduce noise
- Differential '1': D+ high, D- low
- Differential '0': D+ low, D- high
- Encodes data in NRZ-I with bit stuffing
 - Logic '1': no change in voltage level
 - Logic '0': change in voltage level
 - To maintain synchronisation, after 6 consecutive '1's, adds a '0' to ensure a transition (red in figure)



Name	Colour	Function
VBUS	Red	5V power supply
D-	White	Data (-), 3.3V
D+	Green	Data (+), 3.3V
GND	Black	Ground

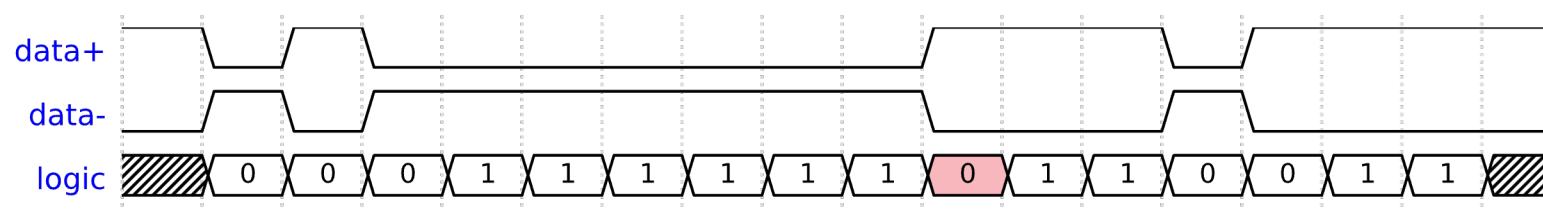
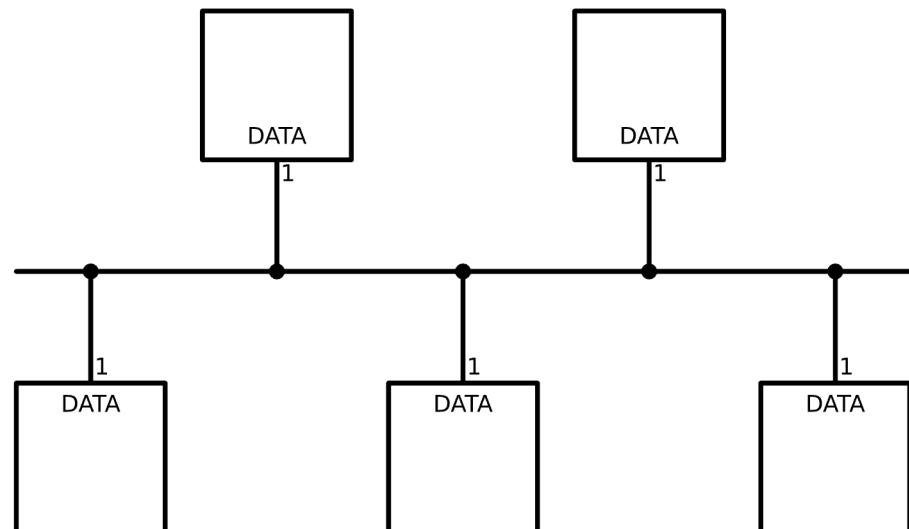


Image source: AN57294 by Cypress Semiconductor

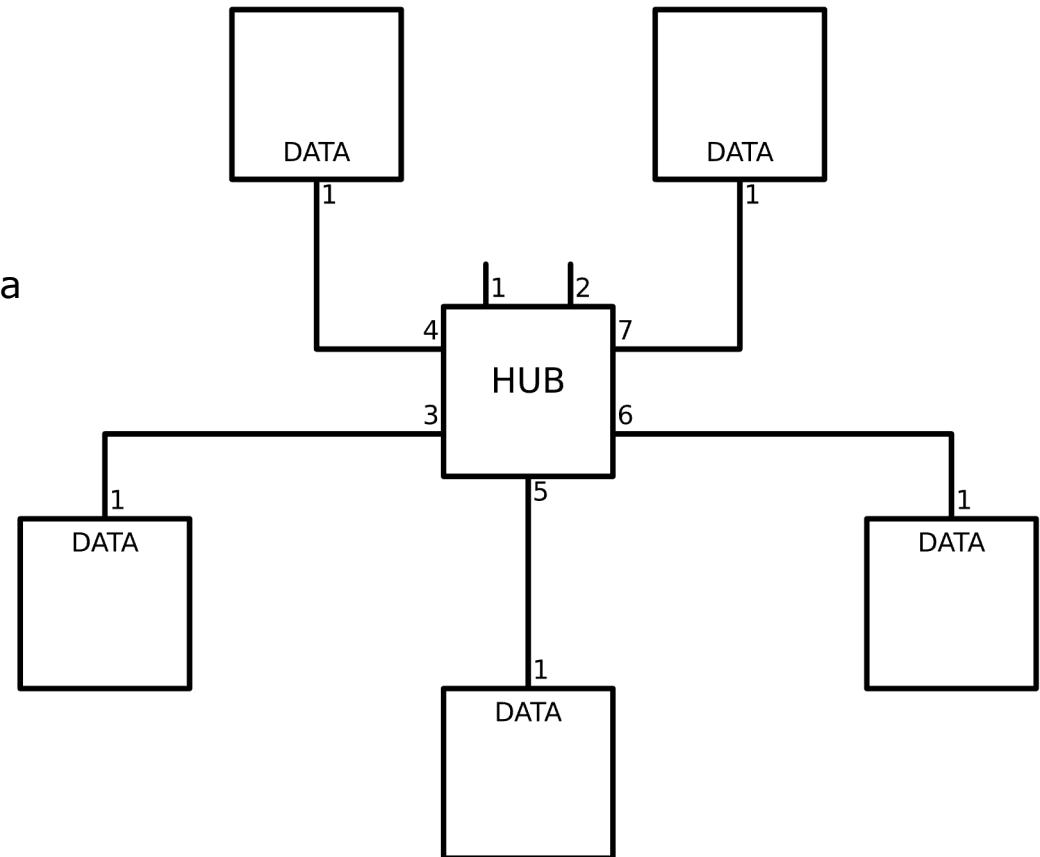
Bus Topology

- Devices tap on a shared cable
- Good fit for environments that are long
 - E.g. elevators, assembly lines, cars
- Easy to add devices
 - Anywhere on the cable
- Vulnerable to cable damage
 - Break in the cable splits bus in two
- Commonly found in embedded controller networks



Star Topology

- Devices connect to a hub with a dedicated cable
- In its simplest form, the hub retransmits the data received from one port to all others
 - Emulates a bus
- Hub is a centralised point that can implement more intelligent features
- Easy to detect and isolate failures
 - A broken wire affects only one device
- Hub is a single point of failure

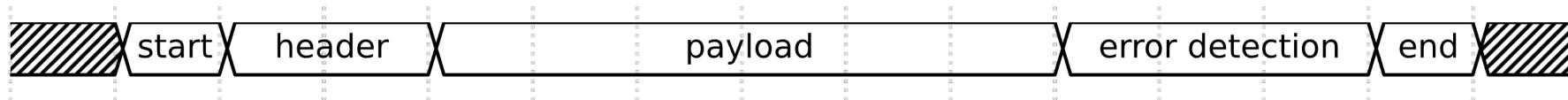


Data Link Layer

- Responsible for the transmission of data frames between physically-connected nodes (local networks)
- Logical Link Layer:
 - Common interface to protocols above
 - Error control (retransmissions, acks, etc)
 - Flow control
- Medium Access Control (MAC) Layer
 - Frame format and synchronisation
 - Physical addressing
 - Error detection
 - Coordinates access to a shared medium

Frames

- Defining the meaning of sequences of bits
- Typical fields
 - Start: Designates the beginning of frame, used by receiver to synchronise
 - Header: Protocol-specific information, such as configurations, addresses, etc
 - Payload: Higher-level data
 - Error Detection: A code used to detect errors (bit flips)
 - End: Designates the end of the frame

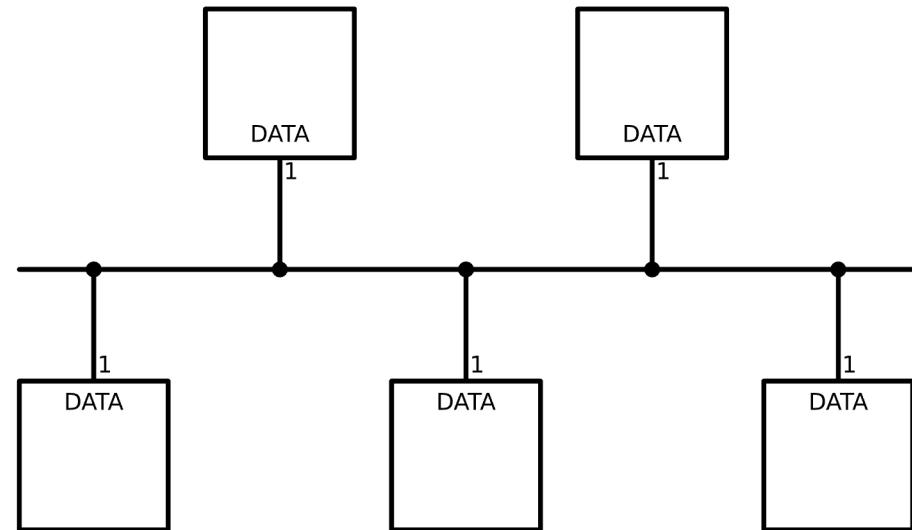


Error Detection

- Main idea:
 - Add a code to the frame that is generated from the header/payload of the frame
 - Receiver repeats the process and if it generates a different code, the frame is discarded
- Parity check
 - Adds one bit to ensure that the total number of '1' is odd or even as pre-agreed
 - Detects 1-bit errors
- Cyclic Redundancy Checks (CRC)
 - Adds a CRC code of N bits (e.g. CRC-15 adds 15 bits)
 - Detects more sophisticated errors

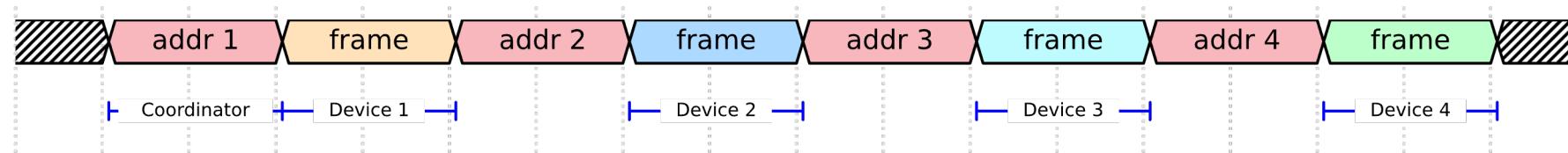
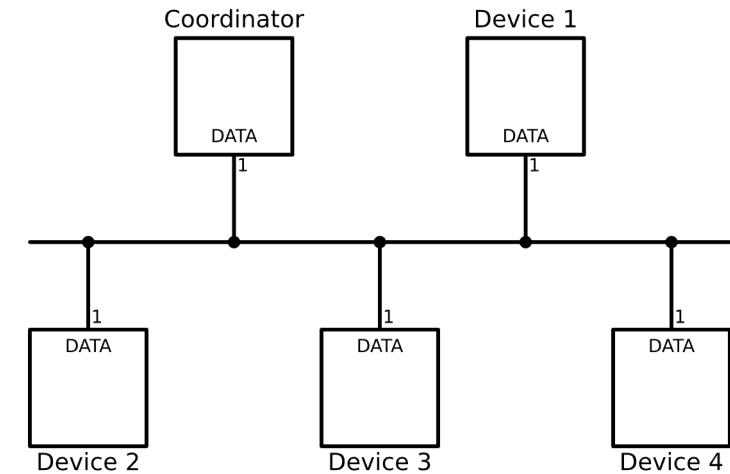
Sharing the medium

- Medium Access Control (MAC)
- Two devices cannot transmit at the same time on a shared bus
- How we can avoid such collisions?
- Who gets priority?



Polling

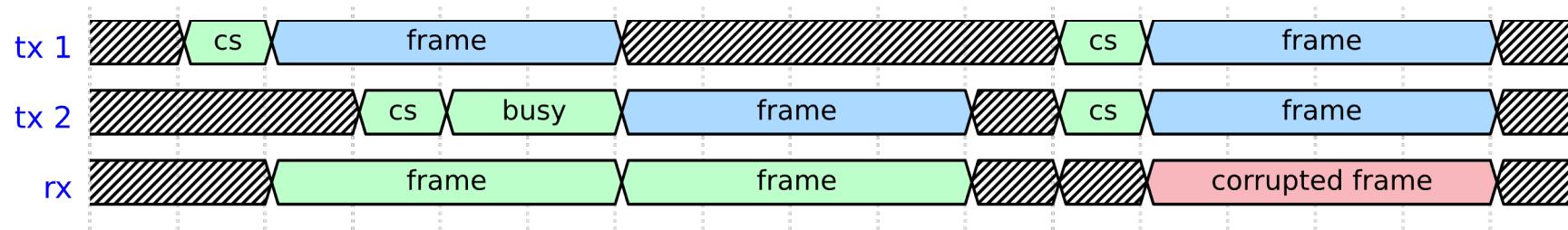
- Coordinator polls the other devices
 - Device does not transmit unless polled
- All communication goes through the coordinator
 - Device-to-device communication not possible
- Coordinator can give more transmission opportunities to devices that need
 - E.g. Device 1 → Device 2 → Device 3 → Device 3 → Device 4 → Device 1 ...
- Advantages: simple and bounded latency
- Disadvantages: polling overhead, device-to-device overhead, not easily extendable, single point of failure



CSMA (Carrier Sense Multiple Access)

- Listen-before-talk
- The transmitter monitors the wire for transmissions (carrier sense, cs)
 - If idle, it proceeds with the transmission
 - If busy, it waits and transmits when it becomes idle
- Collisions can happen if two devices transmit (almost) at the same time
 - Transmitters are not aware that collision happened

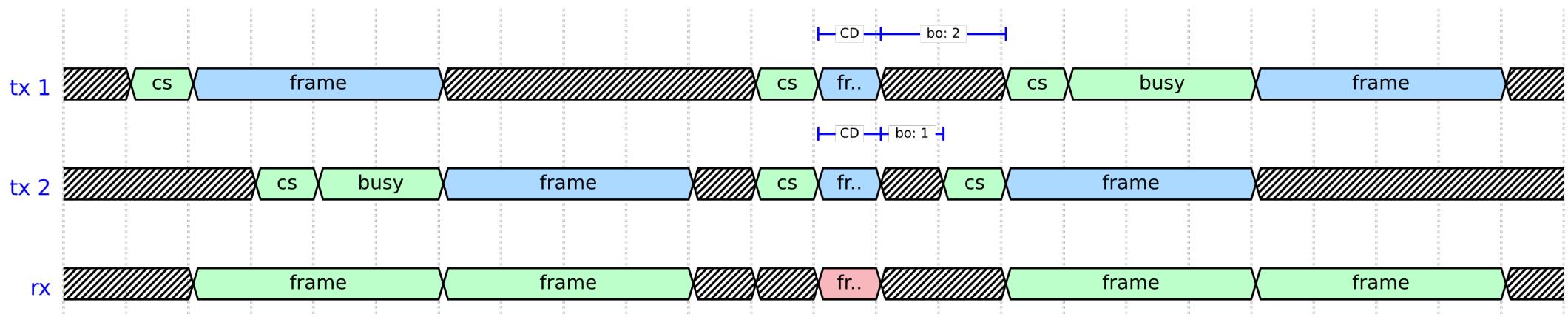
Blue: transmit
Green: receive
Red: receive with errors



CSMA/CD (Collision Detection)

- Collision Detection
 - While transmitting also sense medium for other transmissions
 - If collision detected, wait some time randomly and try again layer
- Advantages:
 - Collisions are stopped early without wasting a lot of bandwidth
 - Retransmissions

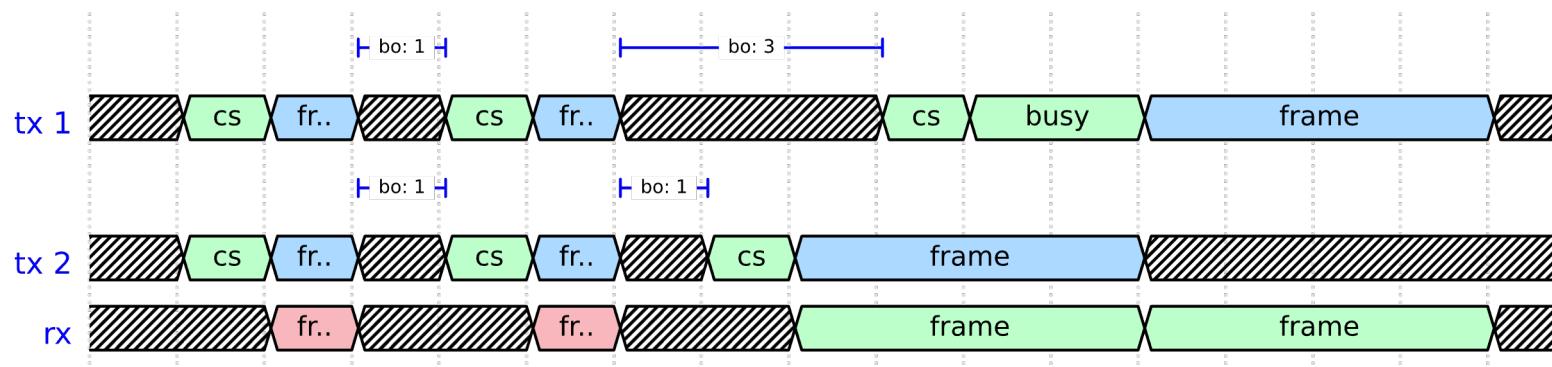
Blue: transmit and listen
Green: receive
Red: receive with errors



Binary Exponential Backoff

- After collision choose random number bo in $[0, CW]$ and back off for bo time units
- How large shall the contention window be?
 - Too small: high probability for another collision in high traffic
 - Too big: unnecessarily large idle time in low traffic
- Exponential backoff (initially CW=1)
 - After each collision CW doubles
 - After a successful transmission it resets to CW=1

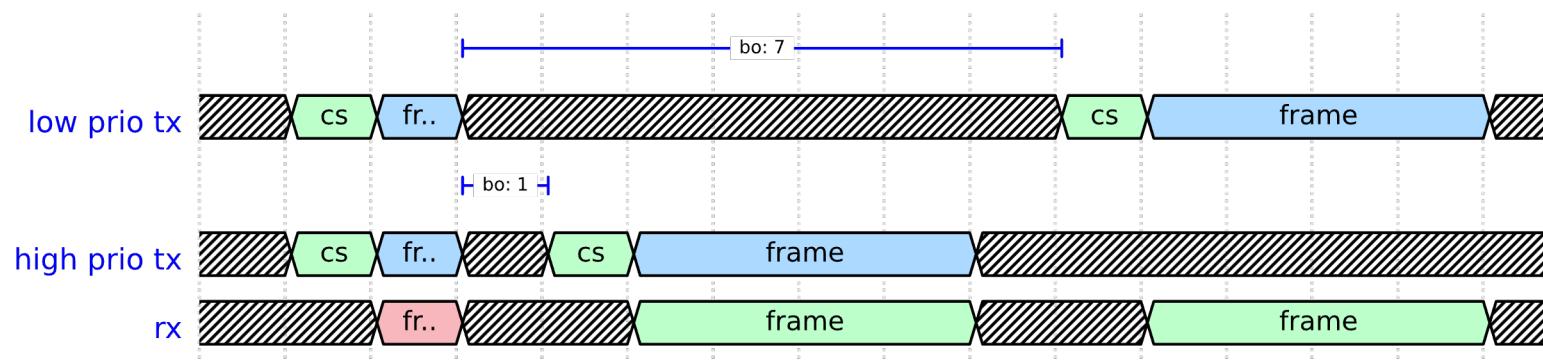
Blue: transmit and listen
Green: receive
Red: receive with errors



Probabilistic Prioritisation

- Device prioritisation via the Binary Exponential Backoff
- Low priority devices start with a large contention window (e.g. CW=8)
- High priority devices start with a small contention window (e.g. CW=1)
- Low priority devices can transmit before high priority devices if lucky!

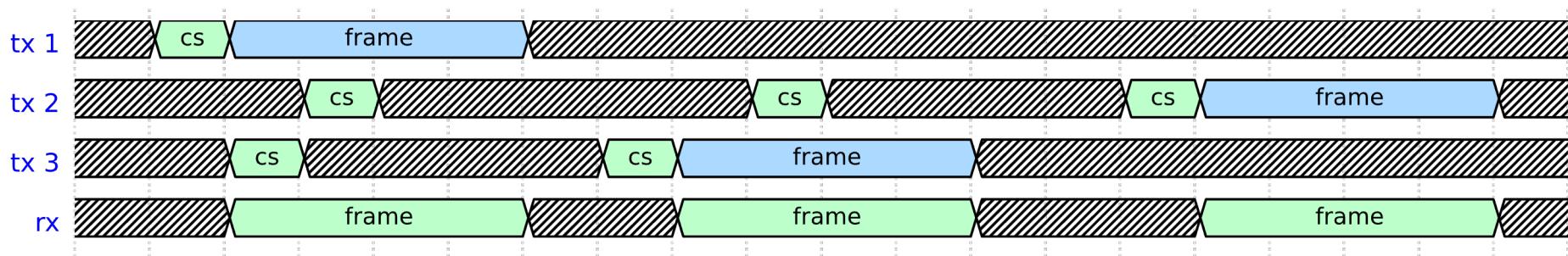
Blue: transmit and listen
Green: receive
Red: receive with errors



CSMA/CA (Collision Avoidance)

- Collision Avoidance
 - When not physically possible to sense medium for other transmissions while transmitting
 - Collisions are more costly as they cannot be stopped early
 - If channel busy, wait some time randomly before attempting first transmission
 - Binary exponential backoff
- Collisions are less likely to occur in the first place
 - But costs bandwidth

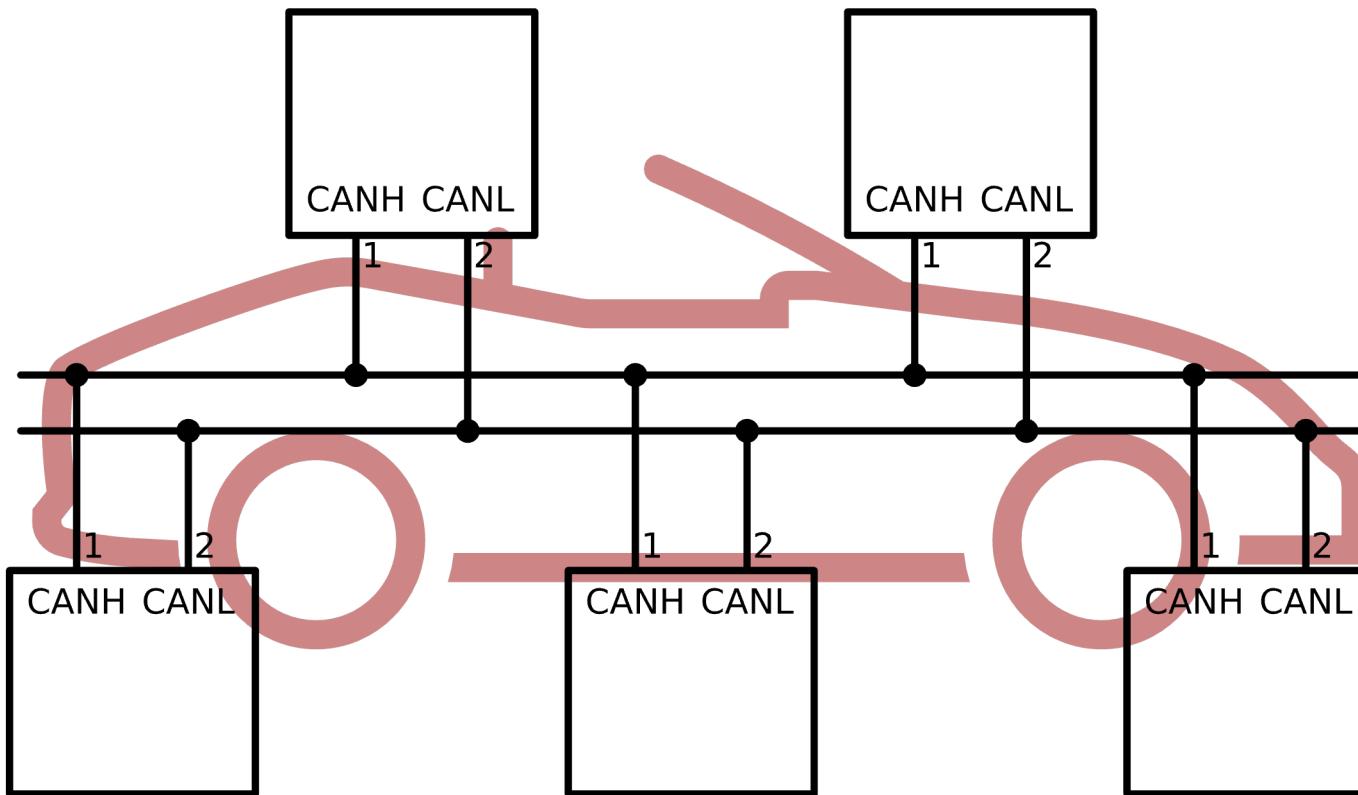
Blue: transmit
Green: receive



CSMA Trade-offs

- Pros
 - Low latency in low traffic conditions (no waiting for your turn)
 - Flexible to add/remove devices from the network
 - Probabilistic prioritisation possible (smaller CW for high priority devices)
- Cons
 - Unbounded latency for individual frames (can in theory wait forever for an idle medium)
 - Poor efficiency under heavy traffic conditions (collisions very likely)

CAN Bus



CAN Bus

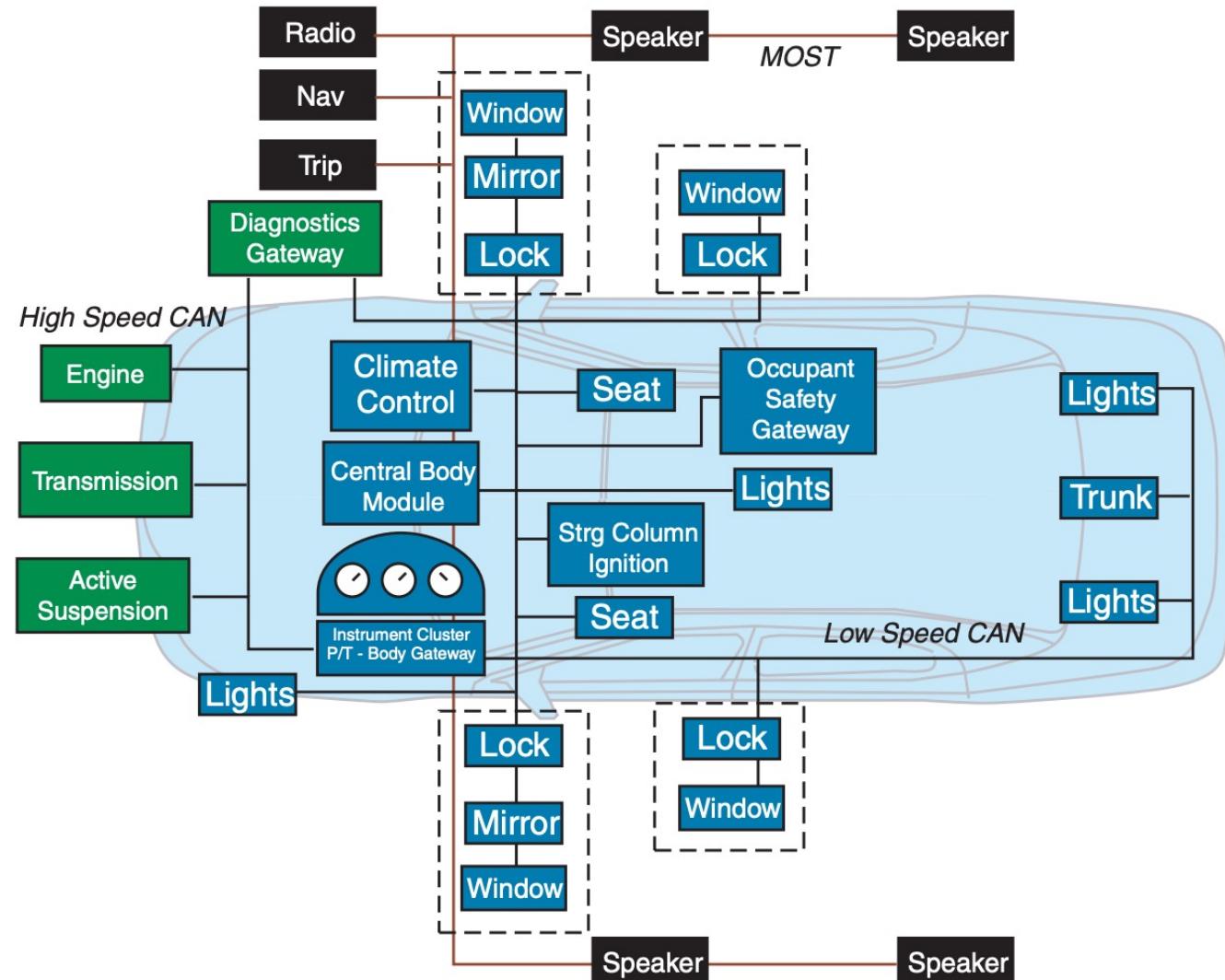
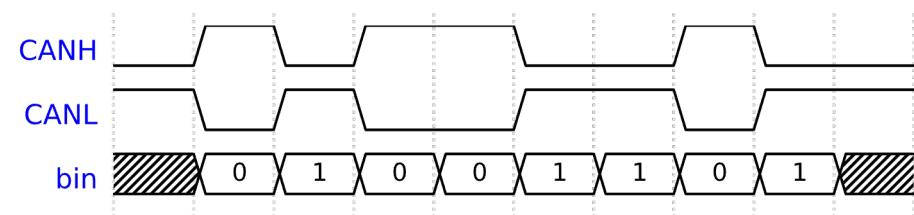
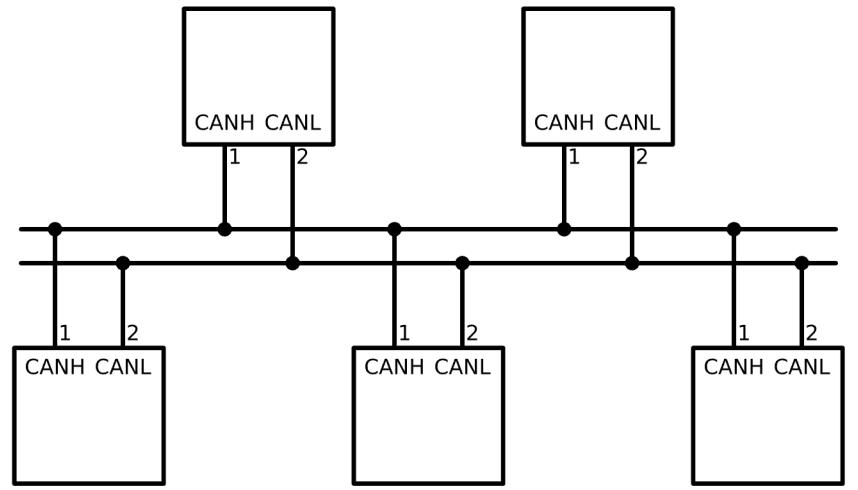


Image source: https://www.eecs.umich.edu/courses/eecs461/doc/CAN_notes.pdf

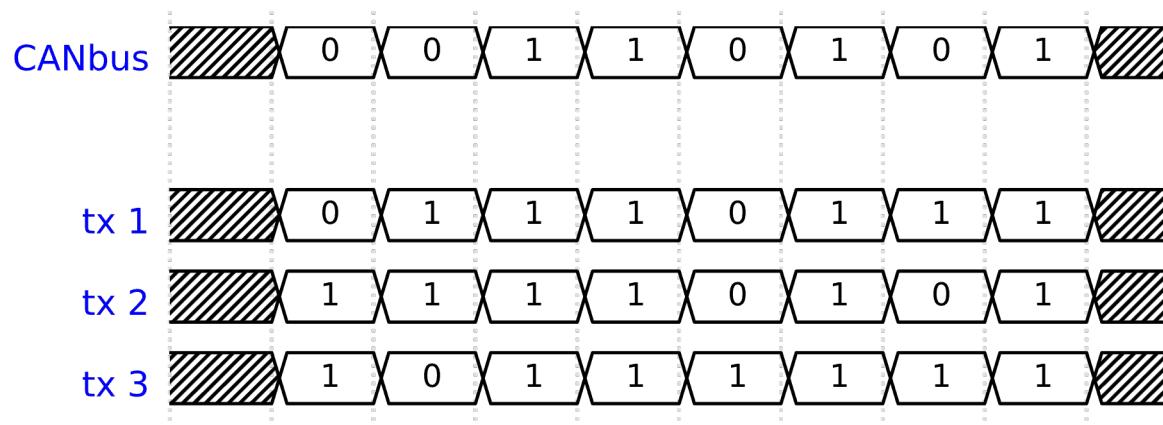
CAN Bus Signal

- Uses differential signalling with two balanced lines (CANH and CANL)
 - ‘1’ is transmitted passively (recessive state)
 - ‘0’ is actively driven (dominant state)
- Low-Speed CAN (up to 125 Kbps)
 - To transmit ‘1’: Resistors pull CANH to GND and CANL to 5V
 - To transmit ‘0’: CANH is actively driven to 5V, CANL is actively driven to GND
- High-Speed CAN (up to 1 Mbps)
 - To transmit ‘1’: Pullup resistors pull CANH and CANL to 2.5V
 - To transmit ‘0’: CANH is actively driven to 3.5V, CANL is actively driven to 1.5V



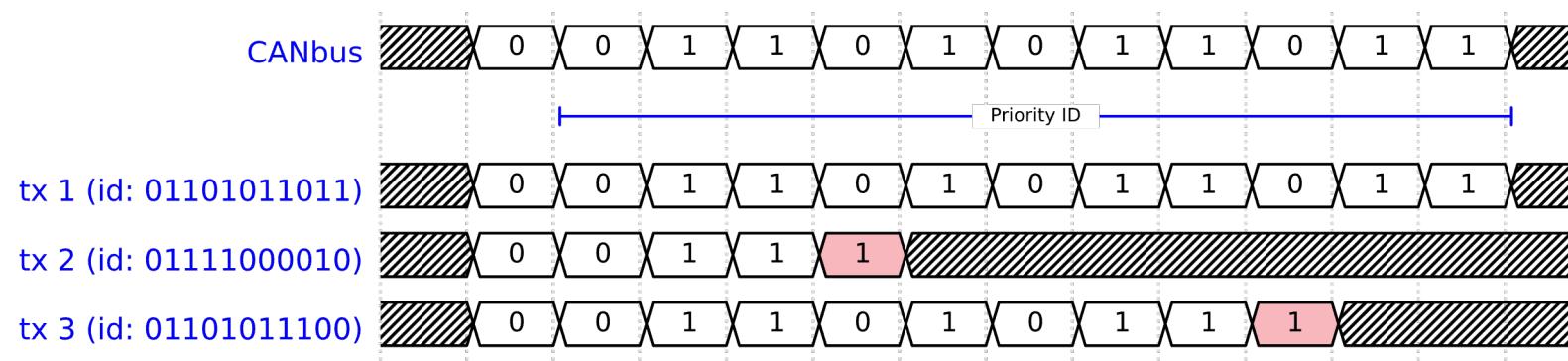
Dominant state wins!

- If **one** device transmits '0' and others transmit '1' at the same time, the CAN bus state is '0'
 - Because '0' is actively driven (dominant), while '1' is passive via pullup resistors (recessive)
 - Only if **all** nodes transmit '1', the bus is '1'
 - '0' always wins



CSMA/CD with Arbitration

- Each node gets a **unique** 11-bit ID
- The frame starts with a SOF (Start Of Frame) dominant bit '0' for synchronisation
- The first field after the SOF is an 11-bit node ID
 - The highest the priority of the node, the lower the ID
- Transmitters read the bus and if an inconsistency is observed (i.e. they transmit '1' but the bus is '0') they stop the transmission and try later
 - Collision is non-destructive, so the highest priority message goes through without errors

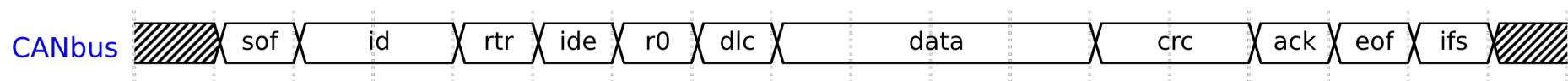


CAN Bus Frame Types

- Communication is sender-initiated and broadcast-based
 - All nodes in the bus receive all frames and higher-level protocols determine if they should be ignored or not
- Bit Stuffing: to maintain synchronisation, a bit of opposite polarity is added after 5 consecutive bits of the same polarity
- Two frame types:
 - Standard: 11-bit IDs (2048 unique identifiers)
 - Extended: 29-bit IDs (537 million unique identifiers)
- Four frame types:
 - Data frame: a frame with data by an ID
 - Remote frame: a frame requesting data from an ID
 - Error frame: a frame transmitted by any node that detects an error
 - Overload frame: a frame to inject a delay

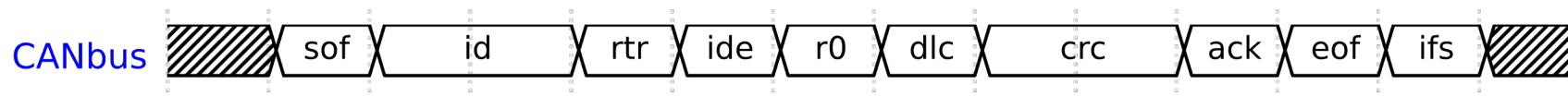
Standard Data Frame

- SOF (start of frame, 1 bit): denotes the beginning of a frame, must be dominant '0'
- ID (identifier, 11 bits): unique ID of sender, also representing message priority
- RTR (remote transmission request, 1 bit): must be dominant '0' for data frames
- IDE (identifier extension bit, 1 bit): must be dominant '0' for standard frames
- R0 (reserved bit, 1 bit): reserved for future extensions of CAN
- DLC (data length code, 4 bits): number of bytes the follow (0 to 8 bytes)
- DATA (data, 0-64 bits): data to be transmitted, length defined by DLC
- CRC (cyclic redundancy code, 16 bits): 15-bit error detection code, last bit recessive '1' as delimiter
- ACK (acknowledgement, 2 bits): transmitter transmits recessive '1', the receiver who received it without errors overwrites with a dominant '0', last bit recessive '1' as delimiter
- EOF (end of frame, 7 bits): denotes the end of frame, must all be recessive '1'
- IFS (inter-frame spacing, 3 bits): space between messages, must all be recessive '1'



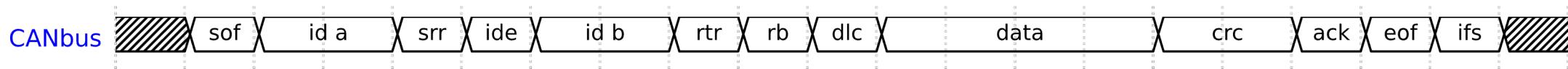
Standard Remote Frame

- SOF (start of frame, 1 bit): denotes the beginning of a frame, must be dominant '0'
- ID (identifier, 11 bits): unique ID of receiver, also representing message priority
- RTR (remote transmission request, 1 bit): must be dominant '1' for remote frames
- IDE (identifier extension bit, 1 bit): must be dominant '0' for standard frames
- R0 (reserved bit, 1 bit): reserved for future extensions of CAN
- DLC (data length code, 4 bits): number of expected bytes for requested message (0 to 8 bytes)
- CRC (cyclic redundancy code, 16 bits): 15-bit error detection code, last bit recessive '1' as delimiter
- ACK (acknowledgement, 2 bits): transmitter transmits recessive '1', the receiver who received it without errors overwrites with a dominant '0', last bit recessive '1' as delimiter
- EOF (end of frame, 7 bits): denotes the end of frame, must all be recessive '1'
- IFS (inter-frame spacing, 3 bits): space between messages, must all be recessive '1'



Extended Data Frame

- SOF (start of frame, 1 bit): denotes the beginning of a frame, must be dominant '0'
- ID A (identifier, 11 bits): most significant part of unique ID of sender
- SRR (substitute remote request, 1 bit): must be recessive '1'
- IDE (identifier extension bit, 1 bit): must be recessive '1' for extended frames
- ID B (identifier, 18 bits): least significant part of unique ID of sender
- RTR (remote transmission request, 1 bit): must be dominant '0' for data frames
- RB (reserved bits, 2 bit): reserved for future extensions of CAN
- DLC (data length code, 4 bits): number of bytes the follow (0 to 8 bytes)
- DATA (data, 0-64 bits): data to be transmitted, length defined by DLC
- CRC (cyclic redundancy code, 16 bits): 15-bit error detection code, last bit recessive '1' as delimiter
- ACK (acknowledgement, 2 bits): transmitter transmits recessive '1', the receiver who received it without errors overwrites with a dominant '0', last bit recessive '1' as delimiter
- EOF (end of frame, 7 bits): denotes the end of frame, must all be recessive '1'
- IFS (inter-frame spacing, 3 bits): space between messages, must all be recessive '1'



CAN Bus: Advantages and Disadvantages

- Advantages
 - Low latency in light traffic
 - Strong prioritisation for vital traffic
 - Robustness (abundant error checking procedures)
 - CRC and ACK, SOF, EOF, delimiters, IFS at message level
 - Bits transmitted are also read for inconsistencies
 - Bit stuffing rule of no more than 5 consecutive bits of the same logic level
- Disadvantages
 - Unfair to low-priority nodes, starvation in high traffic
 - Poor latency for low-priority nodes

IEEE 802.3: Ethernet

- A standard for wired local area networks
- A family of standards that evolves since 1983
 - Originally by Xerox in 1976 at 2.94 Mbps
 - 10BASE5: first standard (10 Mbps)
 - Now up to 400 Gbps (data centres)
- Up to 1500 Bytes of data per frame

Ethernet: MAC Addresses

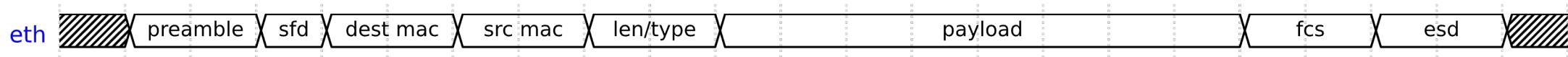
- A unique 48-bit identifier (6 bytes) for each Ethernet interface
 - Typically written in hex, for example AA:BB:CC:DD:EE:FF
- It is composed of two 24-bit parts
 - Organizationally Unique Identifier (OUI): identifies the manufacturer (assigned by IEEE)
 - Hardware identifier: identifies the specific hardware (assigned by the owner of the OUI)
- A MAC address of FF:FF:FF:FF:FF:FF is a broadcast address



Image source: Wikipedia

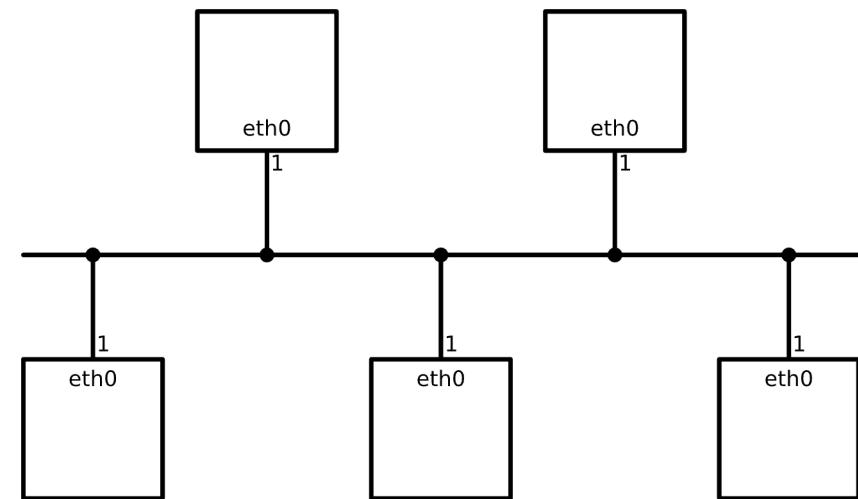
Ethernet Frame

- Preamble (7 bytes): Seven bytes of 0b10101010 for synchronisation
- SOF (start of frame delimiter, 1 byte): 0b10101011, marks the beginning of the frame
- Dest MAC (6 bytes): MAC address of the destination
- Src MAC (6 bytes): MAC address of the source
- Length/Type (2 bytes): Values <1500 indicate the payload length, values ≥ 1536 represent the type of packet inside the payload
- Payload (46-1500 bytes): Data, padding is added to reach the minimum 46 bytes
- FCS (frame check sequence, 4 bytes): Error detection code using CRC-32
- ESD (end of stream delimiter, 4 bytes): Marks the end of the frame
- Note: The first information in the packet is the destination address so receivers can quickly determine if the frame is for them (to process it) or not (to ignore it)



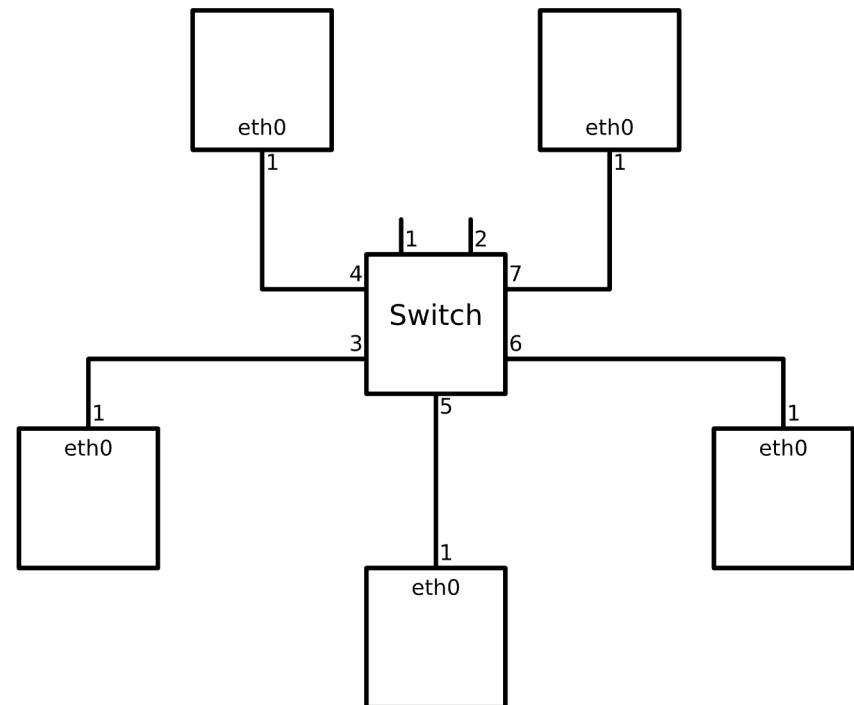
The Original Ethernet

- 10BASE2 and 10BASE5 (10Mbps)
 - Original implementations over a single coaxial cable (half-duplex)
 - Manchester encoding
 - '0' is a high-to-low transition
 - '1' is a low-to-high transition
 - Bus topology and repeater hubs
 - CSMA/CD



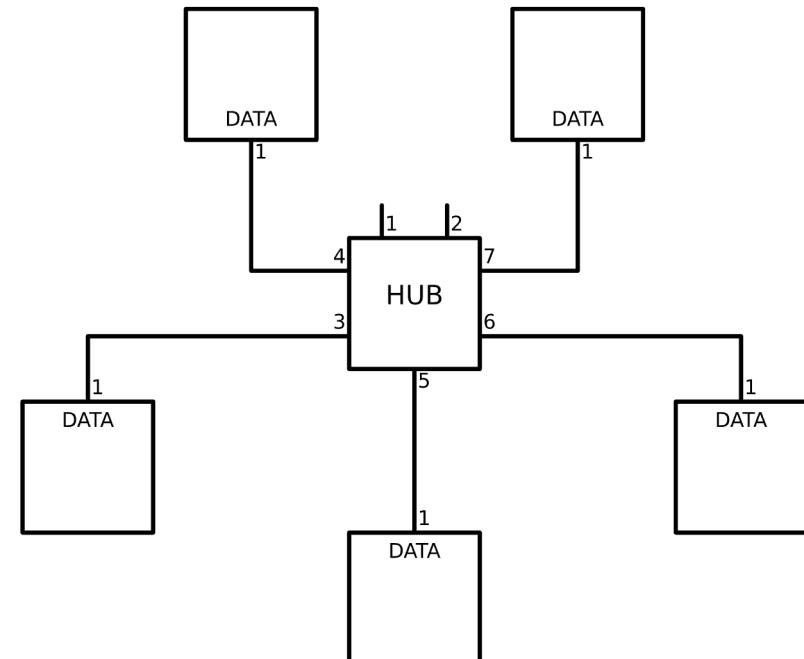
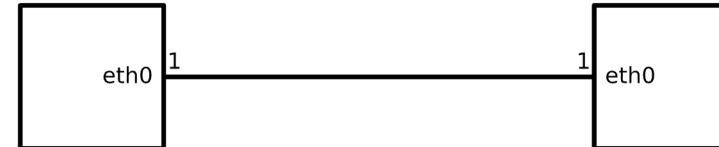
Ethernet Switch

- It learns where each device is connected
 - Maintains a table that maps MAC addresses to ports
 - Entries at the table have an expiration date
 - If destination is in the table, frame is forwarded to the correct port
 - Frame is repeated to all ports otherwise
- Unlike repeater hub that emulates a bus



The Common Ethernet

- 10BASE-T (10 Mbps) and 100BASE-TX (100 Mbps)
 - Peer-to-peer and switched topologies
 - Full-duplex: differential signalling over 2 twisted pair lines (TX+, TX-, RX+, RX-)
 - No collisions, throughput is doubled
- 10BASE-T keeps Manchester encoding
- 100BASE-TX uses:
 - Multi-Level Transition 3 (MLT3)
 - 4B/5B Encoding



Multi-Level Transition 3 (MLT3)

- Three voltage levels are defined: -1, 0, +1
- A transition order is defined: -1, 0, +1, 0, -1, ...
- Line Coding
 - A '0' is denoted as no transition
 - A '1' is denoted as a transition
- Advantage
 - Small transitions allow higher speeds
- Disadvantage
 - Many continuous zeros lead to no transitions and, eventually, loss of synchronisation

Data	1	1	1	0	1	1	0	1
Code	-1	0	1	1	0	-1	-1	0

4B/5B Encoding

- Encodes 4 bits of data into 5 bits
 - 5 bits are then transmitted using MLT-3
- 5 bits allow 32 values
 - 16 values of data (4 bits)
 - Signal code
- Ensures frequent transitions by avoiding 5 bit values that have less than two '1's
- Needs 125 Mbps transmission rate to achieve the 100 Mbps

TABLE 3: 4B/5B ENCODING

Code	Value	Definition
0	11110	Data 0
1	01001	Data 1
2	10100	Data 2
3	10101	Data 3
4	01010	Data 4
5	01011	Data 5
6	01110	Data 6
7	01111	Data 7
8	10010	Data 8
9	10011	Data 9
A	10110	Data A
B	10111	Data B
C	11010	Data C
D	11011	Data D
E	11100	Data E
F	11101	Data F
I	11111	Idle
J	11000	SSD (Part 1)
K	10001	SSD (Part 2)
T	01101	ESD (Part 1)
R	00111	ESD (Part 2)
H	00100	Transmit Error

Image source: AN1120 by Microchip

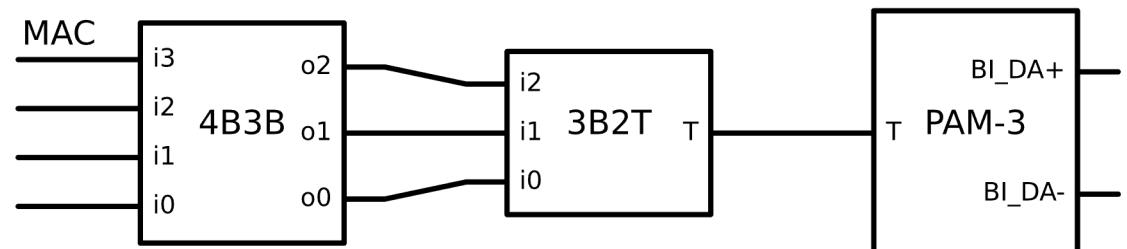
The Automotive Ethernet

- Requirements
 - Infotainment and new sensors require higher data rates in vehicles (than CAN)
 - Need for reduction of cable weight, cost, energy footprint, EMI
 - No need to support large cable lengths, 15m is enough
- 100BASE-T1
 - Uses only 1 bi-directional full-duplex twisted pair cable (echo cancellation)
 - 4B3B to 3B2T with PAM3 encoding
- Transparent to MAC layer

4B3B 3B2T PAM-3 Modulation

- 4B3B
 - Receives 4-bit blocks in parallel at 25 MHz (100 Mbps total)
 - Outputs 3-bit blocks in parallel at 33.33 MHz (100 Mbps total)
- 3B2T
 - Receives 3-bit blocks in parallel at 33.33 MHz (100 Mbps total)
 - Outputs 2 ternary bits, TA and TB, (i.e. 3-state bits) in sequence at 66.66 MHz (100 Mbps total)
- PAM-3
 - Modulates a ternary bit in 3 voltage levels: -1 V, 0 V, +1 V

3-bit data	TA	TB
000	-1	-1
001	-1	0
010	-1	1
011	0	-1
SSD/ESD	0	0
100	0	1
101	1	-1
110	1	0
111	1	1



Echo Cancellation

- The transmitted signal is removed from the received signal
- Enables parallel transmissions (full-duplex) over a single twisted pair cable

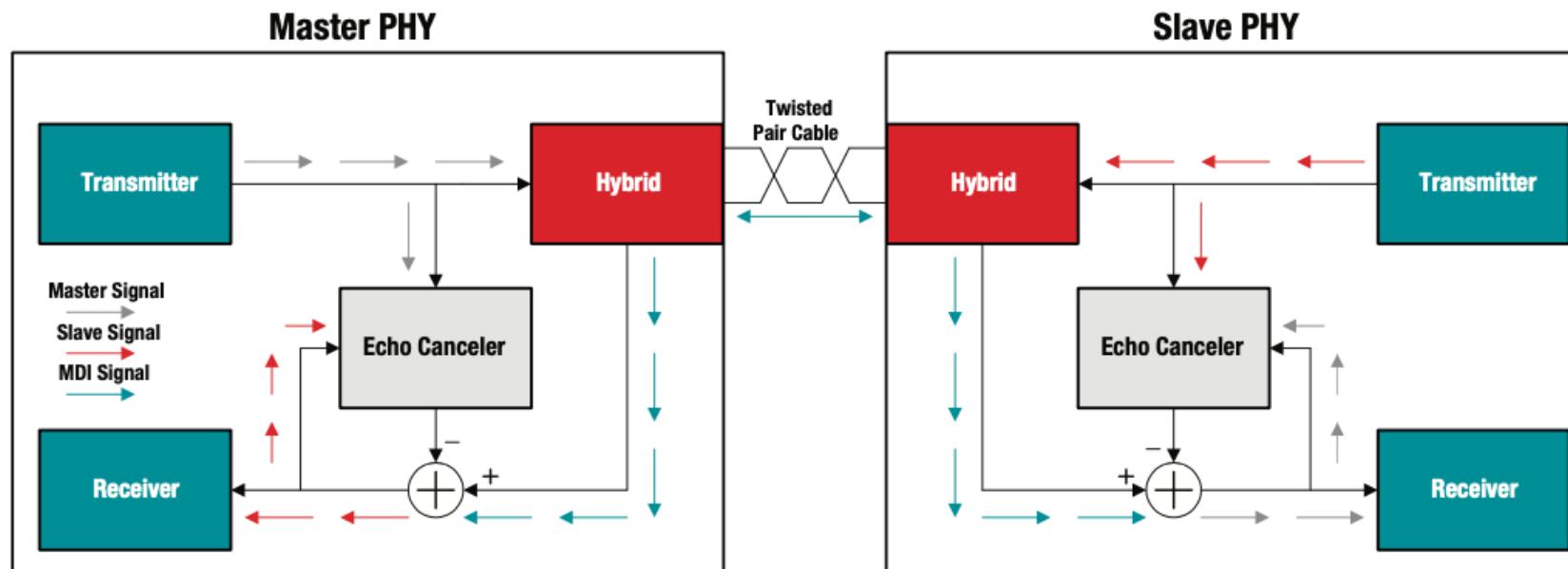


Image source: 100BASE-T1 Ethernet: the evolution of automotive networking by Texas Instruments

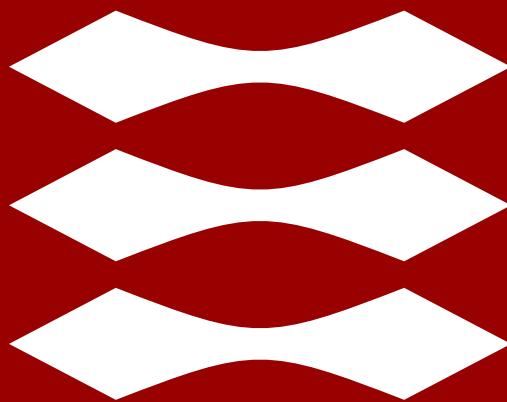
Comparison

Property	Classic Ethernet (100BASE-TX)	Automotive Ethernet (100BASE-T1)
Data rate	100 Mbps	100 Mbps
Baud rate / clock	125 MHz	66.66 MHz
No. Twisted Pairs	2	1
Line Code	4B5B MLT-3	4B3B PAM-3
Max Distance	100 m	15 m

Gigabit Ethernet

- 1000BASE-T (1000 Gbps)
 - Differential signalling over 4 bi-directional full-duplex twisted pair lines (echo cancelation)
 - Each operates at 125 MHz
 - 4D-PAM-5 Encoding
 - Encodes 2-bits:-2, -1, 1, 2
 - '0' for error correction
 - 8 bits are transmitted in parallel (4 lines, 2 bits per symbol, symbol rate 125 MHz)
- Gigabit Automotive Ethernet (1000Base-T1)

DTU



02116/02226 - Networked Embedded Systems

Week 8: Wireless Networked Embedded Systems

Charalampos (Haris) Orfanidis

Assistant Professor

chaorf@dtu.dk

www.compute.dtu.dk/~chaorf

Slides by Xenofon (Fontas) Fafoutis

Wireless vs Wired Embedded Systems

- Wireless Networked Embedded Systems
 - Necessary in mobile embedded systems (wearables, vehicles)
 - Necessary for embedded systems that are deployed in areas without easy access to infrastructure
 - Generally, cheaper, more flexible, and easier to install
 - They can be reliable and secure, but...
- Wired Networked Embedded Systems
 - More reliable (less vulnerable to interference and environmental noise)
 - More secure (less vulnerable to physical attacks)



Basics of Wireless Communications

- Transmission
 - The radio transmitter generates an electric current and supplies it to the antenna
 - The antenna radiates energy as an electromagnetic wave (radio wave)
 - Data are encoded in the physical properties of the radio wave (frequency, phase, etc)
- Reception
 - The antenna produces an electric current based on the electromagnetic waves that reach it
 - The radio receiver processes filters and amplifies the signal, and recovers the transmitted data via demodulation



Radio Waves and Decibels

- As a radio wave propagates through 3D space its power attenuates very fast (exponentially)
 - Analogy: An balloon becoming thinner as it expands
 - Example: signal is transmitted at 0.2 W and reaches the receiver with power 0.1 pW
 - Result: The power of radio signals has a huge dynamic range
- Power is typically expressed in Decibels (dB)
 - A relative unit in logarithmic scale
 - Expresses how big or small is something with respect to some reference
- dB arithmetic
 - Addition in logarithmic domain is multiplication in linear domain
 - Rules of thumb
 - +3 dB approximately equal to x2
 - -3 dB approximately equal to /2
 - +10 dB approximately equal to x10
 - -10 dB approximately equal to /10

$$P_{dB} = 10 \log_{10} \frac{P}{P_o}$$

Q: +17dB is approximately equal to?

Radio Waves and Decibels

- As a radio wave propagates through 3D space its power attenuates very fast (exponentially)
 - Analogy: A balloon becoming thinner as it expands
 - Example: signal is transmitted at 0.2 W and reaches the receiver with power 0.1 pW
 - Result: The power of radio signals has a huge dynamic range
- Power is typically expressed in Decibels (dB)
 - A relative unit in logarithmic scale
 - Expresses how big or small is something with respect to some reference
- dB arithmetic
 - Addition in logarithmic domain is multiplication in linear domain
 - Rules of thumb
 - +3 dB approximately equal to x2
 - -3 dB approximately equal to /2
 - +10 dB approximately equal to x10
 - -10 dB approximately equal to /10

$$P_{dB} = 10 \log_{10} \frac{P}{P_o}$$

Q: +17dB is approximately equal to?

A: $+17\text{dB} = +10\text{dB} + 10\text{dB} - 3\text{dB} = 10 \times 10 / 2 = \times 50$

Decibels: Typical Power References

- Amplification/Attenuation Processes
 - The output power is expressed with reference to the input power
 - Example: A 3dB power amplifier doubles the power of the signal it receives

$$P_{dB} = 10 \log_{10} \frac{P_{OUT}}{P_{IN}}$$

- Absolute Power
 - The power is expressed with reference to 1mW (dBm)
 - Example: A -10dBm signal has power that is $\sim 1/10$ of the power of 1mW, or $\sim 0.1\text{mW}$

$$P_{dB} = 10 \log_{10} \frac{P}{1\text{mW}}$$

- Combination
 - If I amplify a -10dBm signal using a 3dB amplifier, I get:
 $-10\text{dBm} + 3\text{dB} = -7\text{dBm} (\sim 0.2\text{mW})$

Power	Example
296 dBm	Power output of the sun
80 dBm	Transmission power of FM radio station
60 dBm	Radiated power of microwave ovens
30 dBm	Maximum transmission power of a Wifi router
27 dBm	Maximum transmission power of a cellular phone
0 dBm	Typical transmission power of small embedded systems
-20 dBm	Received signal power of WiFi at very short distance
-100 dBm	Minimal received signal power of WiFi
-174 dBm	Thermal noise

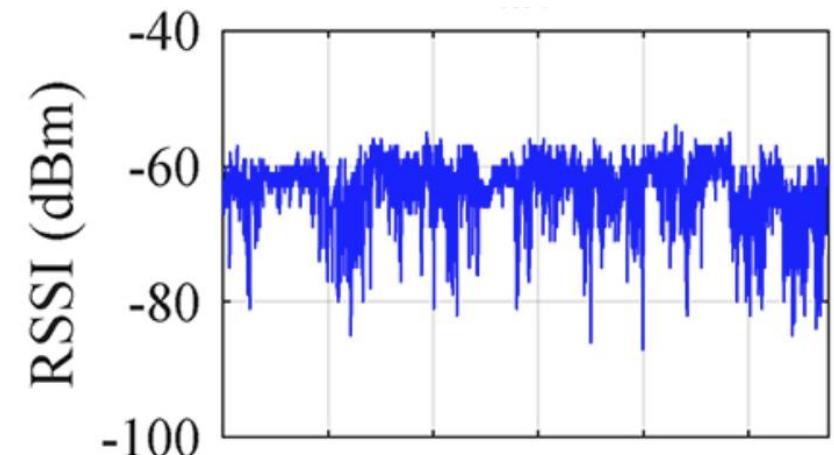
Link Budget

- A received signal power approximator: $P_{RX} = P_{TX} + G_{TX} - L_{TX} - L_P + G_{RX} - L_{RX}$ where:
 - P_{RX} is the received power (dBm)
 - P_{TX} is the transmission power (dBm)
 - G_{TX} is the transmitter antenna gain (dBi)
 - L_{TX} is the transmitter losses: cables, connectors, etc (dB)
 - L_P is the path loss (dB)
 - G_{RX} is the receiver antenna gain (dBi)
 - L_{RX} is the receiver losses: cables, connectors, etc (dB)
- The path loss (L_P) depends on:
 - The distance between transmitter and receiver
 - The frequency of the signal
 - The propagation environment
- The Received Signal Strength Indicator (RSSI) can be used to estimate the path loss



Path Loss

- Path Loss is very predictable in free space, but very dynamic down on Earth!
- Multipath fading
 - A wireless signal reflects on surfaces or the air
 - Multiple rays reach the receiver
 - Rays may arrive with small delays due to longer travel paths (phase shifts)
 - Rays may interfere constructively or destructively
- Shadowing
 - Obstacles in between can attenuate signals
 - Metallic objects block completely the signals
- Path loss is very dynamic even when the transmitter and receiver are static (constant distance)
 - Figure shows the RSSI of a static link in an office environment during working hours



Antenna Gain

- Antennas are passive components
 - They do not amplify the signal
 - They reshape the signal sending/receiving more energy in some directions than others
- Isotropic Antenna
 - A theoretical antenna that emits/collect energy equally in all 3D directions
 - Used as reference for directivity
- Directivity
 - Real-world antennas emit/collect more energy at some locations
 - Measured as peak power gain relative to isotropic antenna (dBi)
 - High directivity means high gain in one direction, but low gain in all others!
 - Low directivity is good when source location is unknown (e.g. mobility)
 - High directivity is good when source location is known (e.g. P2P links)
 - Antenna Gain is directivity minus losses

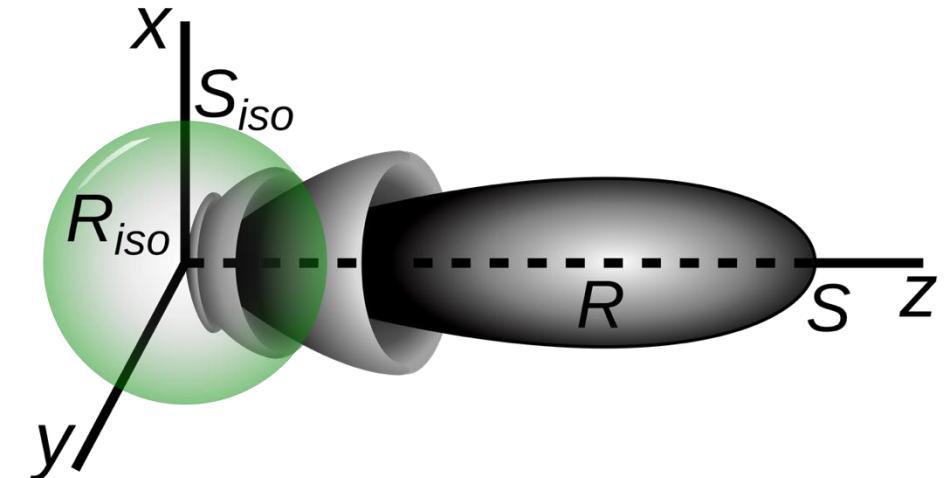


Image source: Wikipedia

Connecting Antennas to Radios

- External Antennas
 - Connect to the PCB via SMA or u.FL connectors
- Printed Antennas
 - Copper trace printed on the PCB
- Matching Circuit
 - Maximise power emitted
 - Minimise power reflected back
- Balun
 - Chip that converts the balanced output of a specific radio to unbalanced input for an antenna
 - With integrated matching circuit
- Some radios have integrated balun so you just connect the antenna

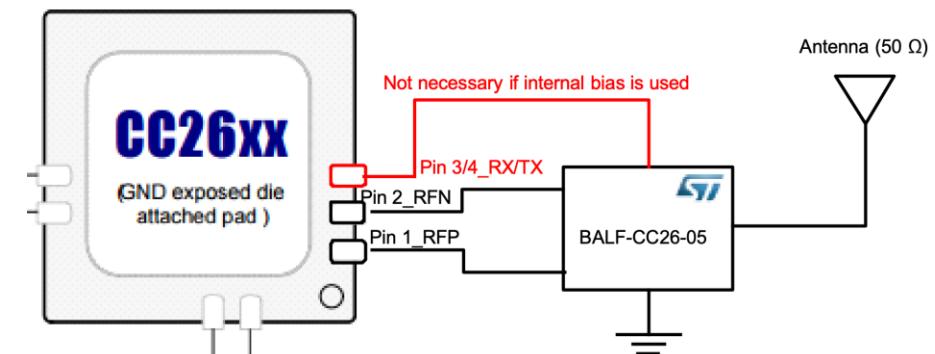
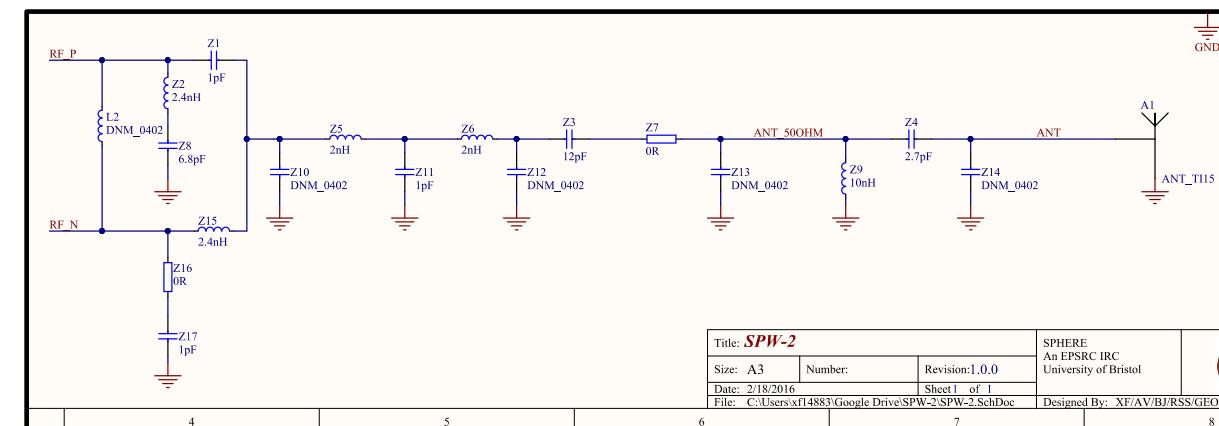
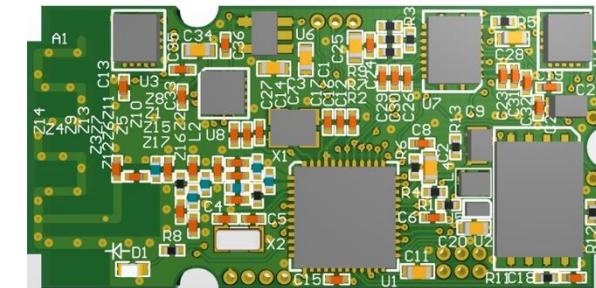


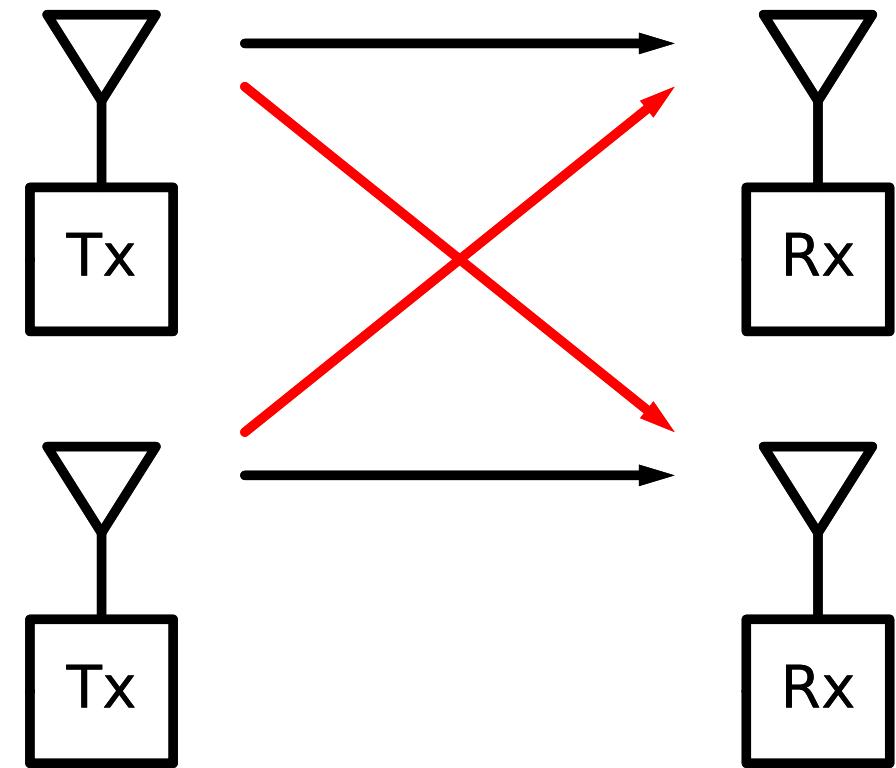
Image source: BALF-CC26-05D3 Datasheet by STMicroelectronics (bottom)

Requirements for a Successful Reception

- There are two requirements for successful packet reception
- The received signal needs to be sufficiently powerful to be perceived by the receiver
 - A radio receiver typically has a sensitivity threshold, e.g. -97dBm
 - It is the lowest signal power level from which the receiver can extract information
 - If the received signal is weaker, the receiver will perceive the signal as noise
- The received signal needs to be sufficiently more powerful than the sum of any external interference signals also captured by the receiver antenna
 - External interference can introduce bit errors in the decoding process
 - Wireless environments are exposed to interference so typically received power higher than the sensitivity is required for robust communication

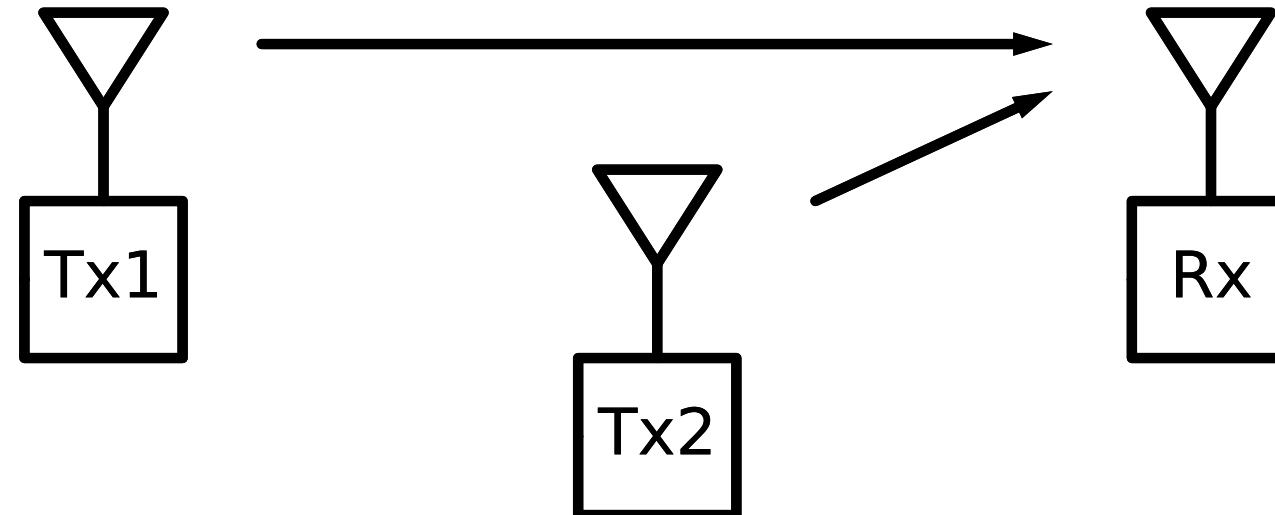
Interference

- Wireless Interference
 - All wireless signals are broadcasted in the environment
 - Receivers collect all of them
 - To successfully decode a signal, it must be sufficiently stronger than all others received
- Interference is subjective
 - A signal creates interference to others
 - Analogy: speaking/listening
- Sources of interference
 - Other transmitters of the same network
 - Transmitters from overlapping networks
 - Other electronics (microwaves, lights, etc)



Capture Effect

- Assume two simultaneous packet transmissions from Tx1 and Tx2 with equal Tx power
- The closest transmission from Tx2 can be received without errors
 - From the perspective of Tx2 the interference is low
 - From the perspective of Tx1 the interference is high



Wireless Coverage

- Often simplified as a circle or a cell
- However, real world is not so simple
 - RSSI is very dynamic due to multipath fading and shadowing
 - Bit Error Rate (BER) is very dynamic due to interference
- Packet Error Rate (PER), depends on:
 - Transmission power and antennas
 - Distance
 - Environment
 - Interference
 - Transmission rate
 - Packet size

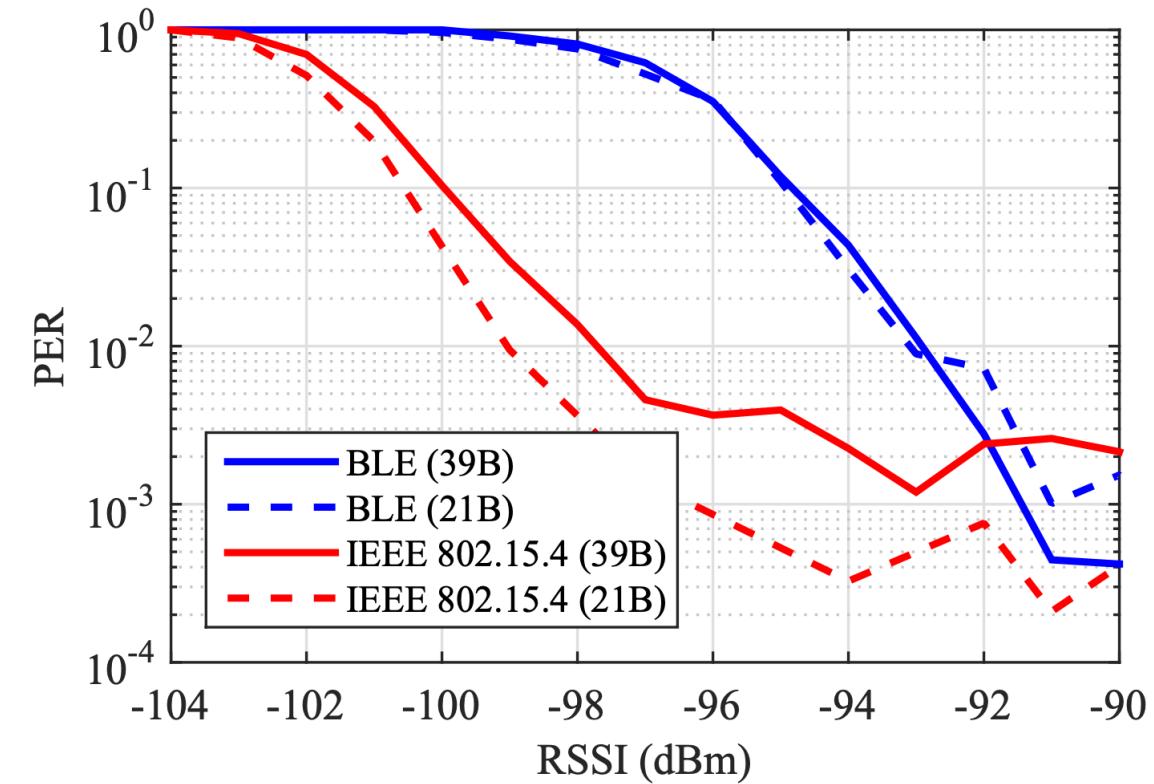


Image source: <http://dx.doi.org/10.4108/eai.1-12-2016.151713>

Carrier Wave

- Information is encoded in the physical properties of a radio wave (amplitude, frequency, phase, etc)
- The modifications that represent '0' and '1' are done on top of a carrier wave with frequency f_c
- Example: BFSK (Binary Frequency Shift Keying)
 - '0' is represented as f_c-f_0
 - '1' is represented as f_c+f_0
- Advantage:
 - Receiver can use a **bandpass filter** to filter out all signals that are not near the carrier wave
 - Limits interference to signals close to carrier frequency
 - Parallel interference-free transmissions in different frequencies

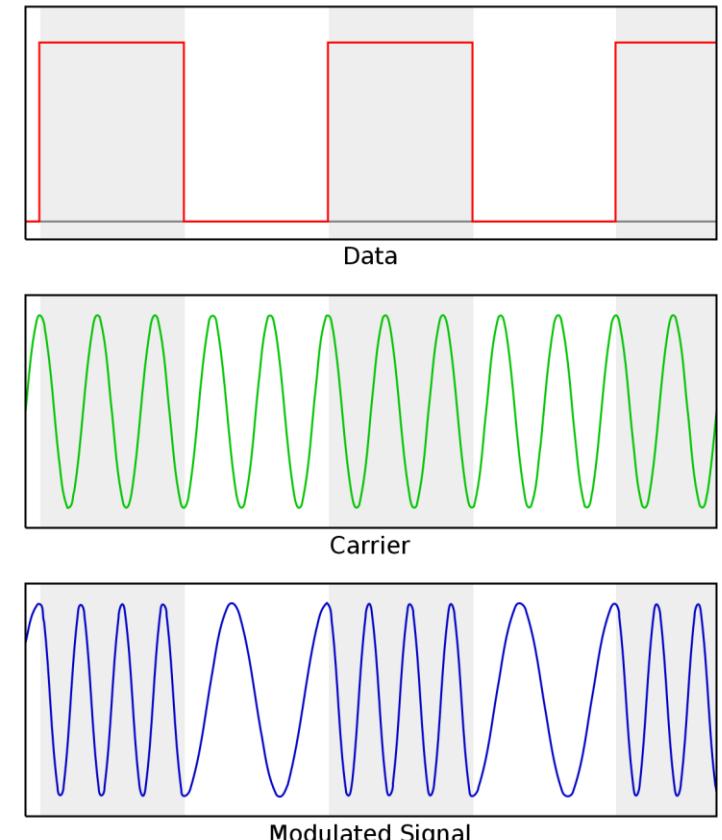


Image source: Wikipedia

Frequency Band: Trade-offs

- Lower Frequency Band
 - Lower path loss
 - More obstacle penetration
 - More coverage
 - More interference
 - Less capacity for data
 - Lower bitrate
 - Bigger antennas
- Above visible light EM radiation is dangerous
- Digital systems typically communicate with radio waves and microwaves (300 MHz to 300 GHz)
 - 300 MHz - 1 GHz (Sub-GHz): UHF radar band
 - 1 GHz - 30 GHz: UHF and SHF bands
 - 30 GHz - 300 GHz (millimetre wave): EHF band

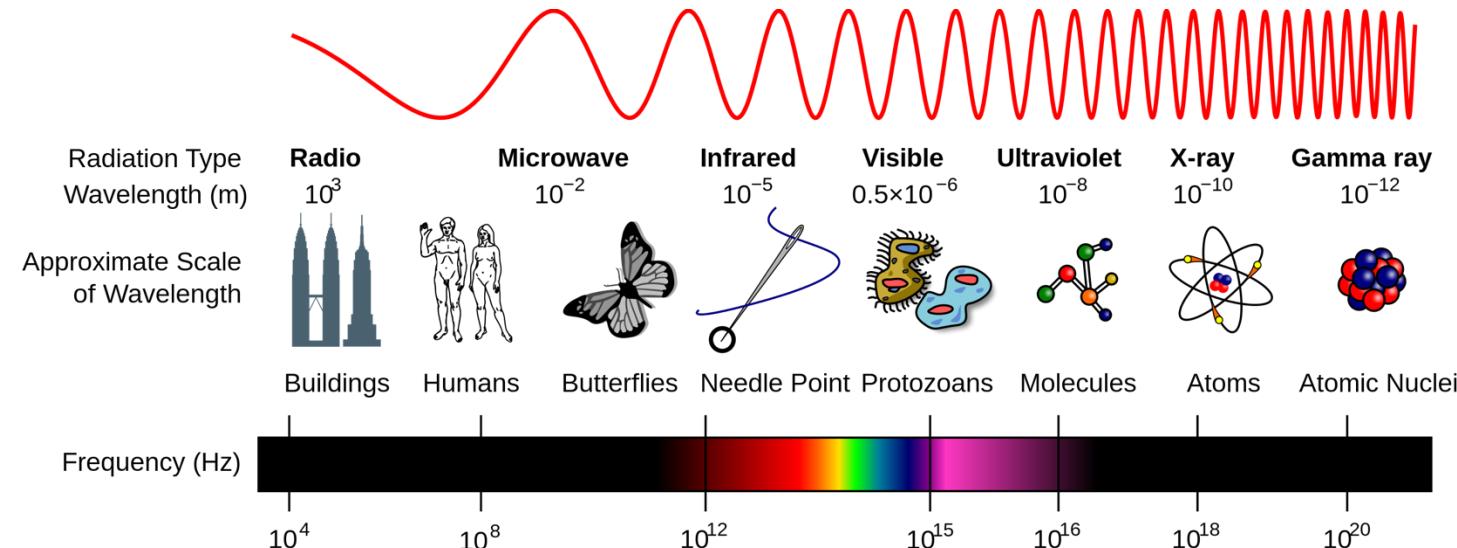


Image source: Wikipedia

Frequency Bands: Rules

- Frequency bands are limited, thus regulated by governments
 - Who can use them
 - How often they can be used
 - What is the maximum transmission power
- Licensed bands
 - Illegal to transmit unless you have a license
 - Government, aviation, military, etc
 - Rented to telecom providers (cellular)
- Unlicensed bands
 - OK to use without a license
 - ISM (industrial, scientific and medical) by ITU (International Telecommunication Union)
 - 863-870 MHz band in Europe

Frequency range	Availability	
6.765 MHz	6.795 MHz	Subject to local acceptance
13.553 MHz	13.567 MHz	Worldwide
26.957 MHz	27.283 MHz	Worldwide
40.66 MHz	40.7 MHz	Worldwide
433.05 MHz	434.79 MHz	Europe, Africa, Middle East
902 MHz	928 MHz	North/South America
2.4 GHz	2.5 GHz	Worldwide
5.725 GHz	5.875 GHz	Worldwide
24 GHz	24.25 GHz	Worldwide
61 GHz	61.5 GHz	Subject to local acceptance
122 GHz	123 GHz	Subject to local acceptance
244 GHz	246 GHz	Subject to local acceptance

Wireless Channels

- We can further split a band in channels
- Overlapping Channels
 - Interfere with each other
- Orthogonal Channels
 - Do not interfere with each other
- FDMA (Frequency Division Multiple Access)
 - Transmitters can transmit in parallel at different orthogonal channels without interfering with each other

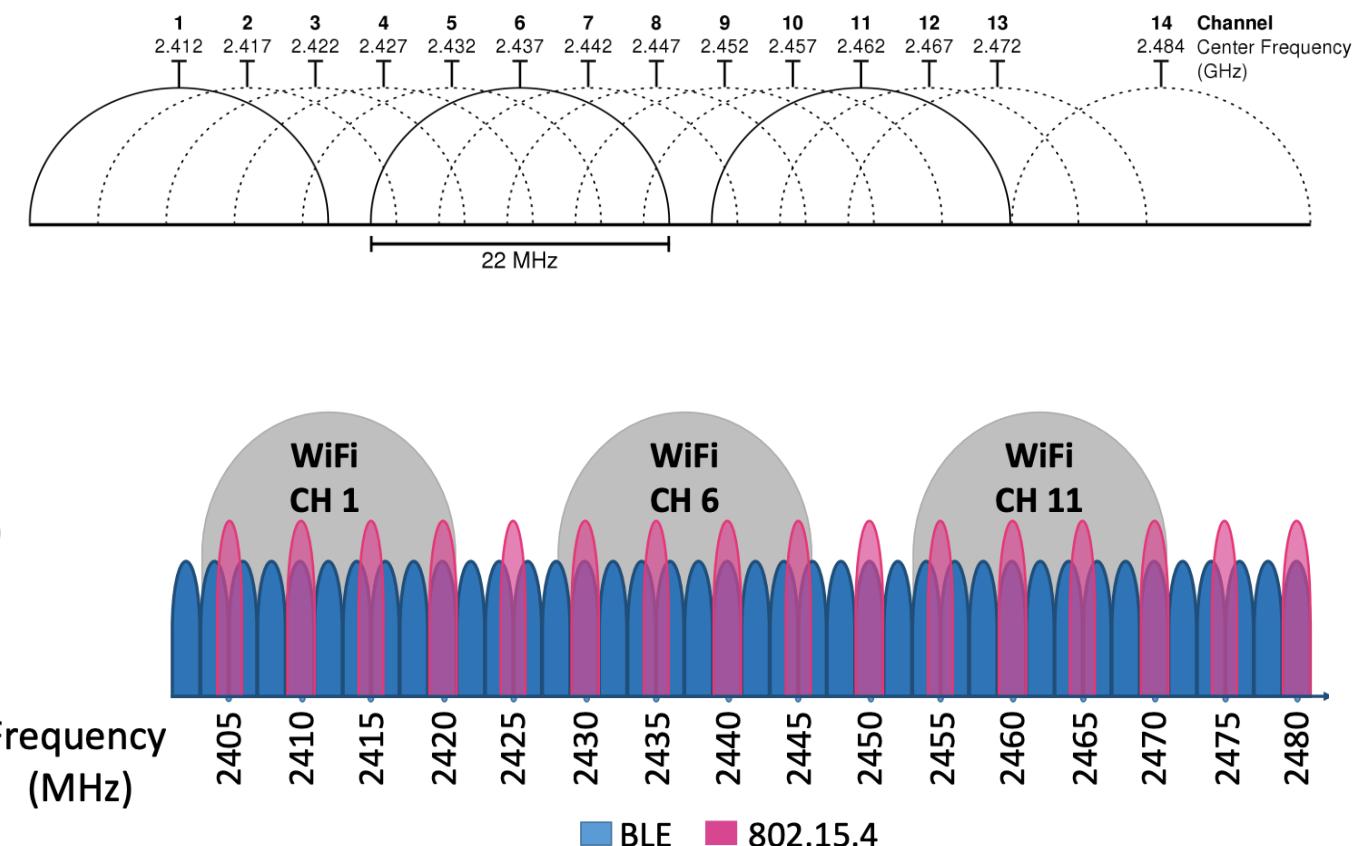


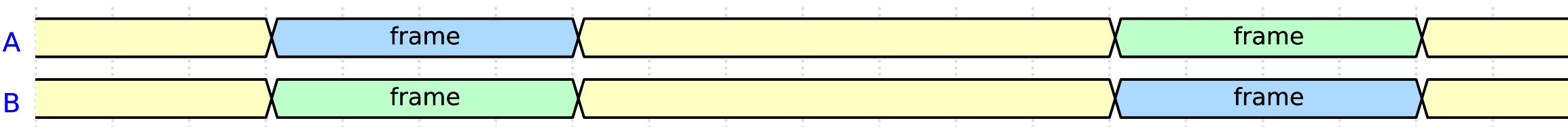
Image source: Wikipedia (top) and <https://doi.org/10.1109/PIMRC.2017.8292262> (bottom)

Radio States

- The receiver filters, amplifies and decodes an incoming signal
 - Each frame starts with a preamble (i.e., a synchronisation phrase)
 - Receive (receive data) and Listen (receive noise) are the same state
- Wireless radios have two modes: Transmit and Receive
 - Devices cannot receive while they transmit
 - Collisions cannot be detected as they happen
 - Devices need to be in receive mode when idle to detect incoming frames

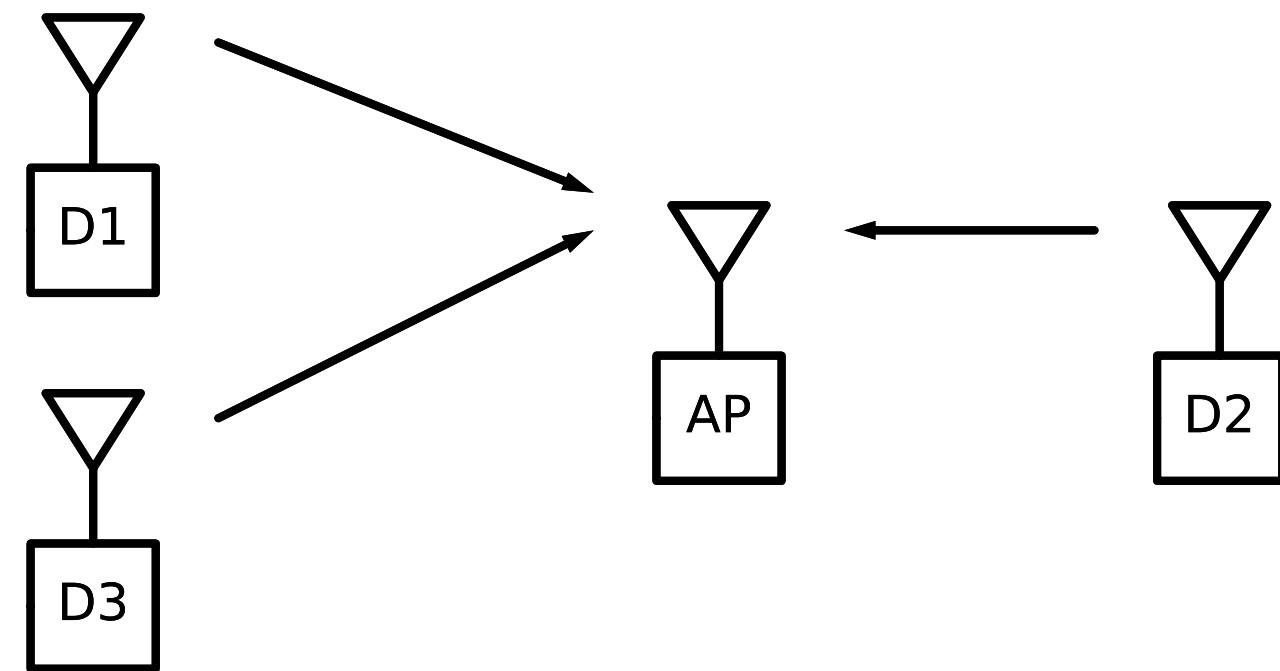


Blue: transmit
Green: receive
Yellow: listen

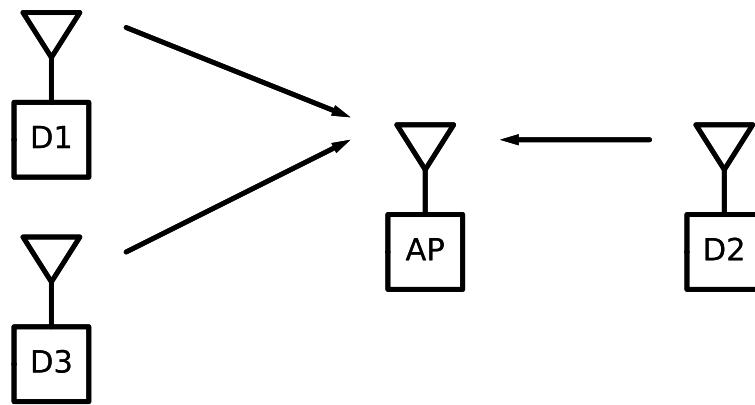


Medium Access Control: Sharing a Channel

- Star Topology
- Access Point (AP)
 - One node collects data or bridges the wireless network to the wired
 - Aka Sink or Gateway (GW)
- Wireless devices talk directly to the AP
- Vulnerable to collisions and interference
- Overlapping networks can use a different channel to mitigate interference

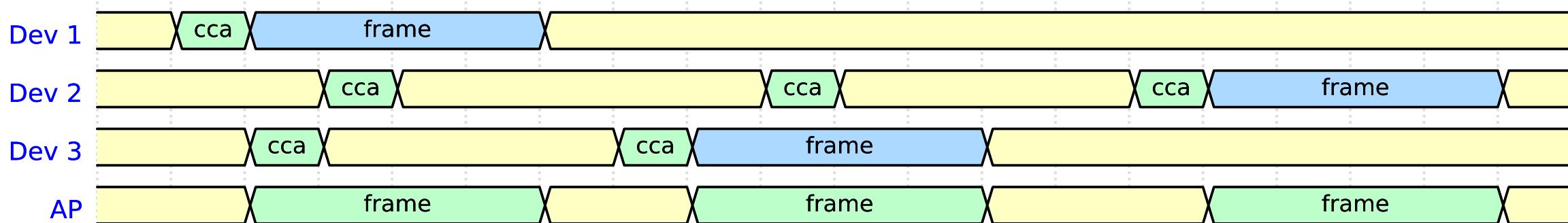


CSMA/CA

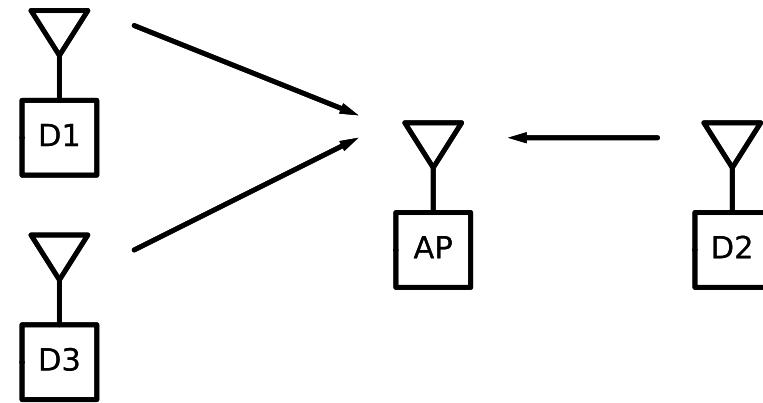


- CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)
 - Sense the channel before transmitting (Clear Channel Assessment, CCA)
 - If channel free, transmit
 - If channel busy, wait some time randomly before attempting first transmission
 - Binary exponential backoff
- Used by IEEE 802.11 (WiFi)

Blue: transmit
Green: receive
Yellow: listen

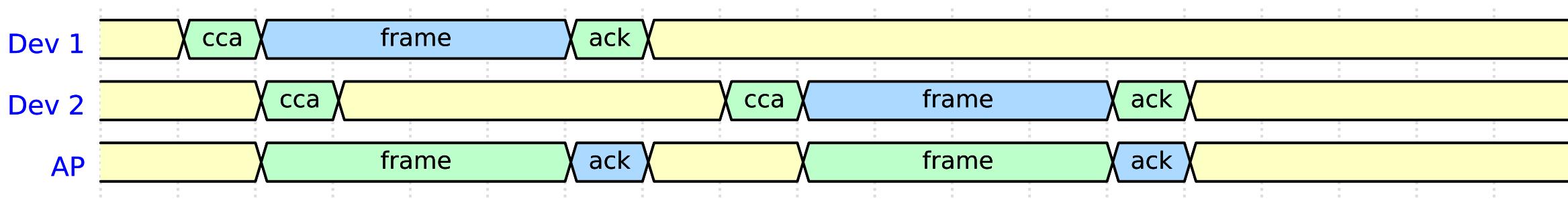


Acknowledgements



- Channel errors are more likely than wired networks
 - CA reduces the probability of collisions, but collisions can still happen
 - Interference from other nodes of the same network, other overlapping networks, etc
 - Signal might arrive weaker due to environmental issues, such as an obstacle
- Acknowledgements (ACK) at the end of each frame
 - If no ACK, retransmit
 - If maximum number of retransmission attempts reached, drop frame

Blue: transmit
Green: receive
Yellow: listen



Radio Duty Cycle

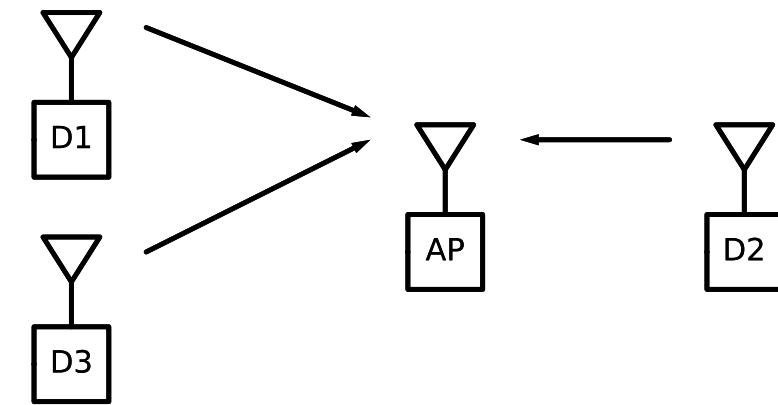
- Often devices have energy constraints
 - Battery-powered
 - Energy Harvesting
- The radio consumes a lot of energy when active (i.e. TX or RX mode)
 - Listening requires the same power as receiving
 - Idle listening wastes energy resources
- Solution: Radio Duty Cycle
 - Turn radio off when not used
 - Radio has three states (TX, RX, Sleep)
- Challenge
 - How does the receiver know when the transmitter is about to transmit, to wake up and put the radio in RX mode?

Radio RX ⁽¹⁾	5.9	mA
Radio RX ⁽²⁾	6.1	
Radio TX, 0-dBm output power ⁽¹⁾	6.1	
Radio TX, 5-dBm output power ⁽²⁾	9.1	

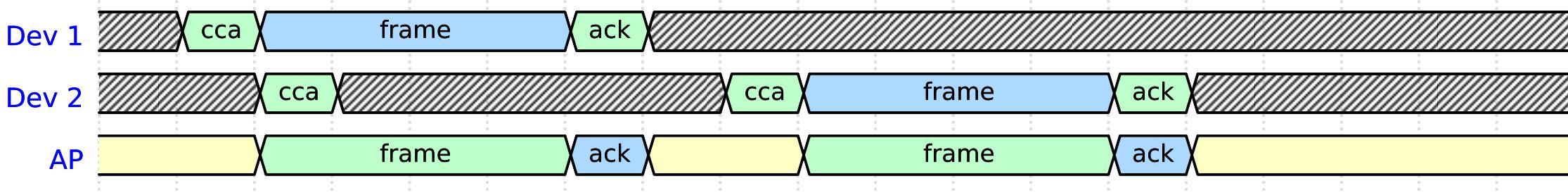
Image source: CC2650 Datasheet by Texas Instruments

Asymmetric Energy Availability

- Star Topology
 - The AP/GW is mains-powered (no energy limitations)
 - The wireless device are energy-constrained and need to be duty cycled
- MAC Protocol: Duty-Cycled CSMA
 - AP/GW always in RX mode unless it transmits (no duty cycle)
 - Nodes wake up when having data to transmit
 - If CCA succeeds, transmit packet and switch to RX mode waiting for the ACK
 - If CCA fails, go to sleep and try again later

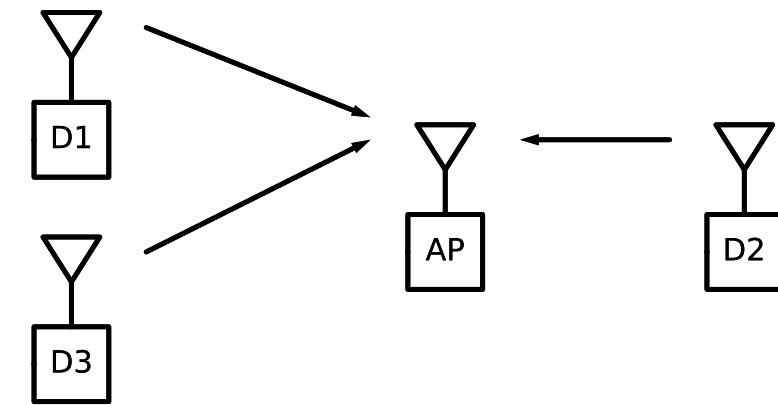


Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



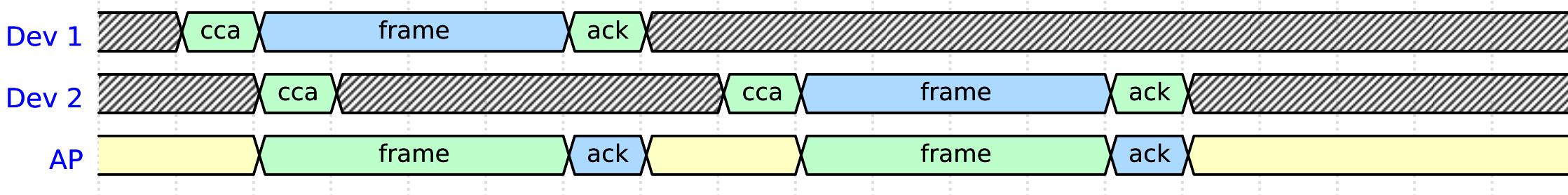
Asymmetric Energy Availability

- Star Topology
 - The AP/GW is mains-powered (no energy limitations)
 - The wireless device are energy-constrained and need to be duty cycled
- MAC Protocol: Duty-Cycled CSMA
 - AP/GW always in RX mode unless it transmits (no duty cycle)
 - Nodes wake up when having data to transmit
 - If CCA succeeds, transmit packet and switch to RX mode waiting for the ACK
 - If CCA fails, go to sleep and try again later



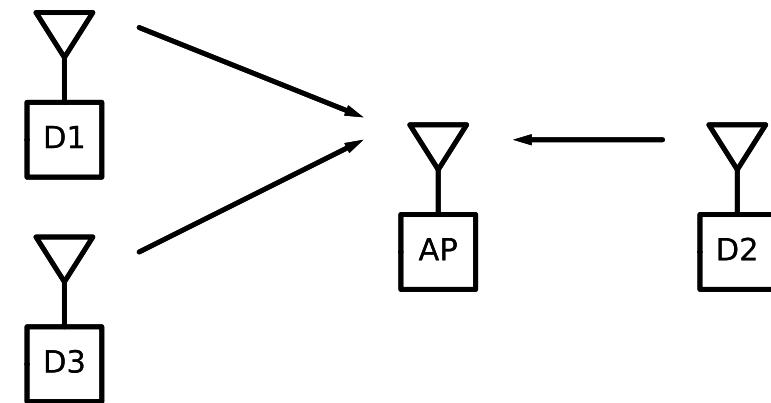
Blue: transmit
Green: receive
Yellow: listen
Gray: sleep

Challenge: OK for uplink, but how does the AP send data to the devices?

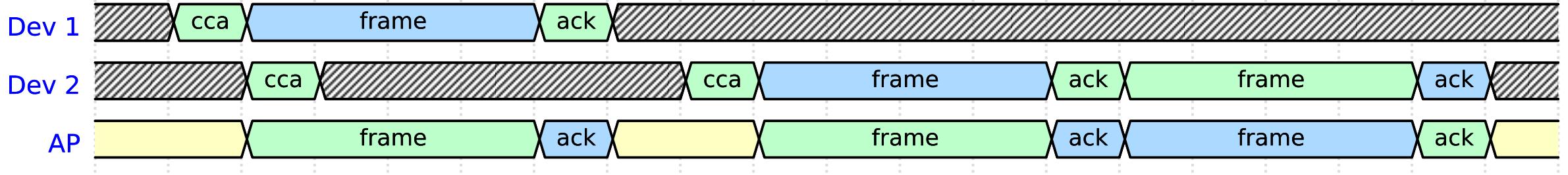


Downlink Traffic

- Star Topology
 - The AP/GW is mains-powered (no energy limitations)
 - The wireless device are energy-constrained and need to be duty cycled
- MAC Protocol: Duty-Cycled CSMA Downlink
 - AP/GW cannot initiate a data transfer to duty-cycled nodes
 - AP/GW can request in the ACK the node to remain awake in RX mode (receive window)



Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



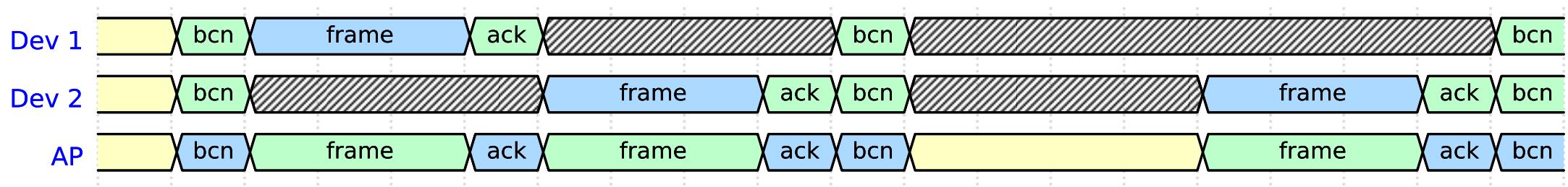
Contention-based vs Contention-free MAC

- CSMA/CA is a contention-based MAC: devices compete for the medium
 - Efficient at low-medium traffic conditions and sporadic traffic
 - No guarantees that important devices will find the medium free
 - Many collisions at high traffic conditions
- Contention-Free MAC: transmission slots are allocated to specific devices
 - Efficient at high traffic conditions
 - Important devices are guaranteed to find the medium free
 - Still vulnerable to external interference
 - Inefficient at unpredictable traffic
 - When nothing to transmit, the allocated slots are wasted
 - Less flexible to changes, overhead with making the slot allocation
- Analogy: Car lanes vs Bus lanes

Guaranteed Time Slots

- AP/GW transmits a beacon with a schedule indicating when each device can transmit
 - Devices transmit at their dedicated transmit slot
 - If nothing to transmit, the slot is idle
- TDMA (Time Division Multiple Access)
- When joining the network, a device needs to be in RX mode until it gets the first beacon

Blue: transmit
Green: receive
Yellow: listen
Gray: idle

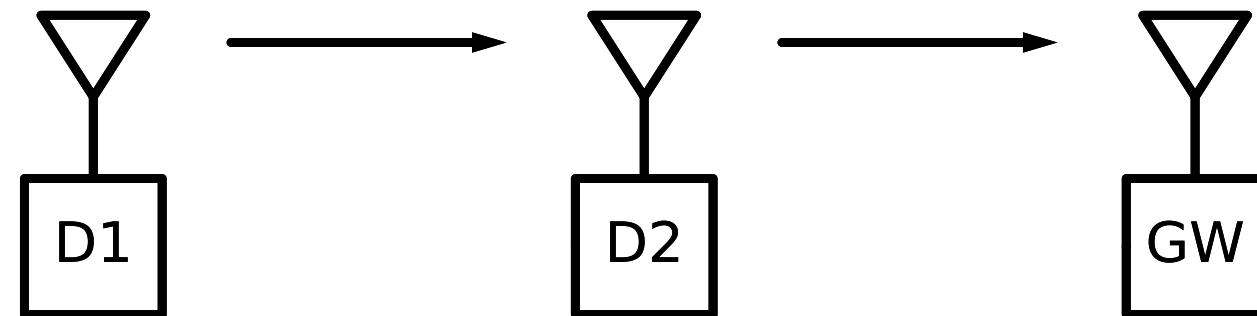


Radio Duty Cycling with Asymmetric Energy Availability

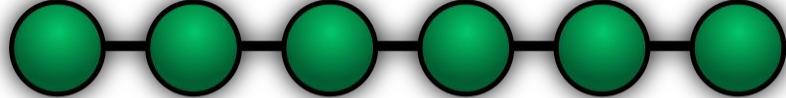
- Advantages
 - Simple to implement and maintain
 - Very energy-efficient for end devices
 - Overhead of synchronisation falls on AP/GW
- Disadvantages
 - Can only be used when receiver has no energy constraints
 - Cannot support device-to-device and multi-hop topologies (mesh networks)
 - Delay for downlink traffic
- Variations of the paradigm used:
 - IEEE 802.15.4 (Zigbee, Thread)
 - BLE (Bluetooth Low Energy)
 - LoRaWAN
 - Low-Power WiFi

Multi-hop Topology

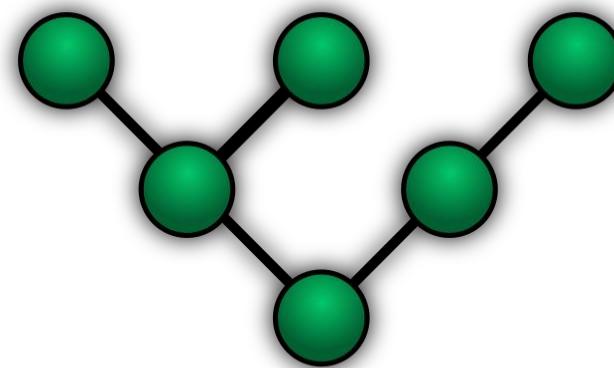
- A device may not have direct access to the gateway
 - Too far away
 - Too weak signal (e.g. basement)
- Coverage can be extended using multi-hop topologies
 - Normal nodes forward frames to the gateway on behalf of other nodes
 - Cost increase: A frame is transmitted/received multiple times to reach the GW
 - Cost decrease: Short links are more robust than long links (retransmissions are reduced)



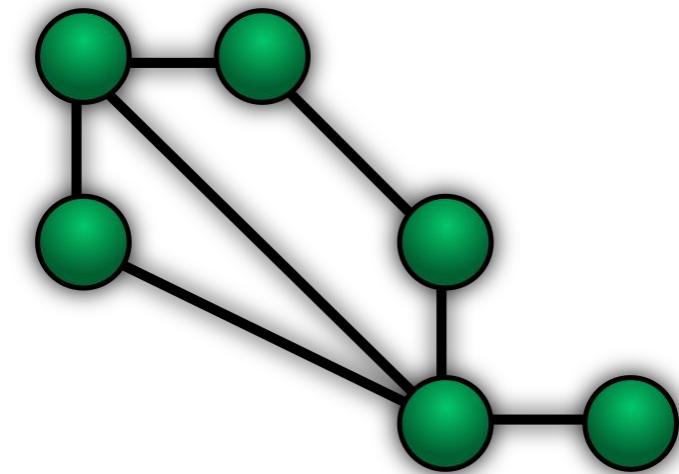
Types of Multi-Hop Topologies



Line



Tree



Mesh

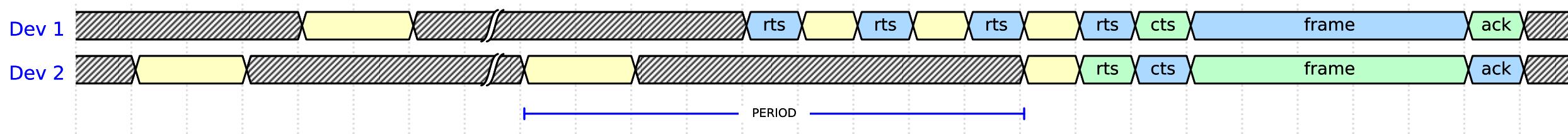
Symmetric Energy Limitations

- Both the transmitter and the receiver need to duty cycle their radios to preserve energy
- Typical in multi-hop topologies (line, tree or mesh topologies)
- Also in star topologies when the GW is battery-powered
- Example: Walkie-Talkie
 - Alice and Bob want to communicate with two battery-powered walkie-talkies
 - When powered on, the walkie-talkie is in listen mode, user presses button to speak
 - They both want to keep them off as much as possible to preserve their batteries
 - Before they separate, they have the opportunity to agree on a radio duty cycle strategy
 - How can they synchronise their transmissions/receptions?

Sender-Initiated Asynchronous MAC

- Also known as Preamble Sampling or Low-Power Listening (LPL)
- Protocol
 - Nodes do not synchronise their duty cycles (asynchronous)
 - When idle, nodes periodically turn their radios in RX mode and sample the channel
 - If nothing is detected, they go back to sleep
 - If preamble is detected, they send a CTS (clear to send) message and receive frame
 - Nodes with data to send, transmit a preamble (or RTS, ready to send) and listen for CTS
 - If CTS received, transmit the frame
 - If no CTS received, send preamble again and repeat
- The listening PERIOD bounds idle listening and defines senders delay/energy costs

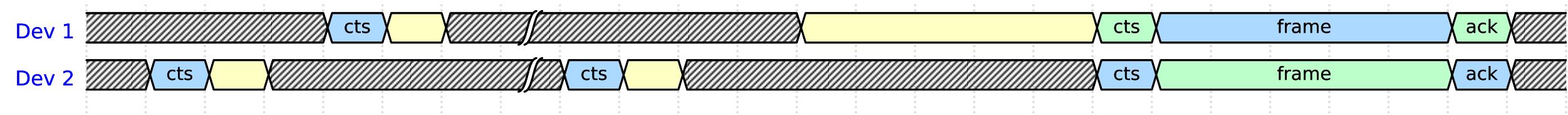
Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



Receiver-Initiated Asynchronous MAC

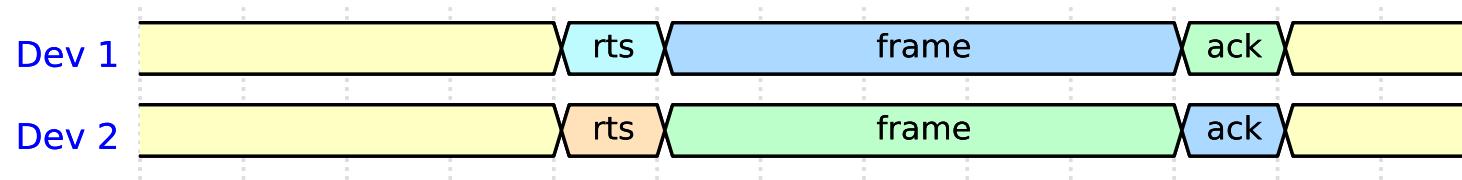
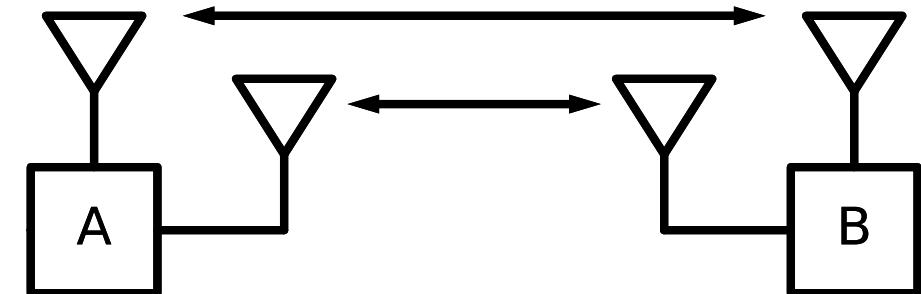
- Protocol
 - Nodes do not synchronise their duty cycles (asynchronous)
 - When idle, nodes periodically transmit a beacon (or CTS, clear to send) and put radios in RX mode to a while
 - If nothing is detected, they go back to sleep
 - If preamble is detected, they stay in RX mode and receive whole frame
 - Nodes with data to send, put their radio in RX mode and listen until they get the CTS
- The listening PERIOD bounds idle listening and defines senders delay/energy costs
- The performance of receiver-initiated and sender-initiated approaches is comparable

Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



Asynchronous MAC with Wakeup Radio

- Devices employ an extra radio: Wakeup Radio
 - Very low-power to keep in listen mode continuously
 - Very low data rates (just enough to send an ID)
 - Limited coverage (typically)
- Protocol
 - When idle, main radio is off, wakeup radio is in RX mode
 - Nodes with data to transmit, send a RTS message via the wakeup radio
 - The sender then puts the main radio in TX mode and sends the frame
 - The receiver of the RTS message, puts main radio in RX mode and waits for the frame

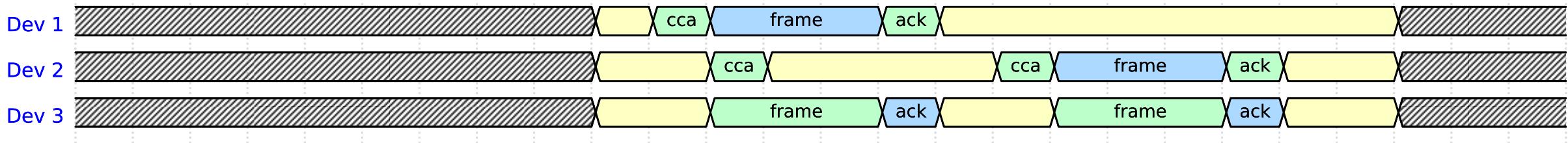


Blue: transmit (main)
Green: receive (main)
Light Blue: transmit (wakeup)
Orange: receive (wakeup)
Yellow: listen (wakeup)

Synchronous MAC

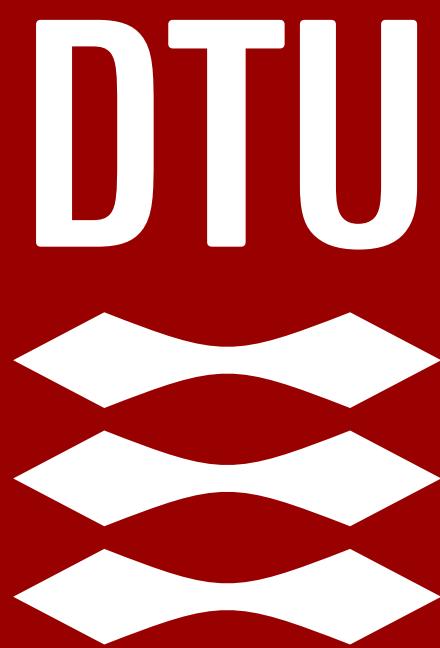
- Devices synchronise their duty cycles
 - Switch between active and sleep periods
 - During the active period, use any MAC scheme, like CSMA/CA or guaranteed time slots
- Duty cycle is controlled by the duration of active/sleep periods
- Idle listening is reduced but not eliminated
- Requires a common notion of time (periodic time synchronisation)

Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



Classification of MAC Protocols

- Duty Cycle
 - No Energy Constraints (no Duty Cycle)
 - Energy Constraints only on end devices (end devices Duty Cycle)
 - Energy Constraints on all devices (all devices Duty Cycle)
 - Synchronous Duty Cycles
 - Asynchronous Duty Cycles
 - Sender-Initiated
 - Receiver-Initiated
 - Wakeup Radio
- Contention Handling
 - Contention-free
 - Contention-based



Networked Embedded Systems

Week 9: Time Synchronisation

Xenofon (Fontas) Fafoutis

Professor

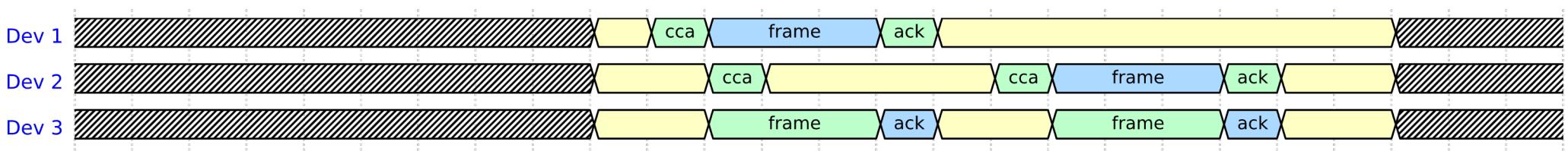
xefa@dtu.dk

www.compute.dtu.dk/~xefa

Synchronous MAC

- Devices synchronise their radio duty cycles
 - Switch between active and sleep periods
- How do devices know when to wake up?
- Requires a way to measure time (**clock**)
- Requires **clock synchronisation**

Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



Time Keeping in Distributed Embedded Systems

- When Distributed Embedded Systems need to keep the time?

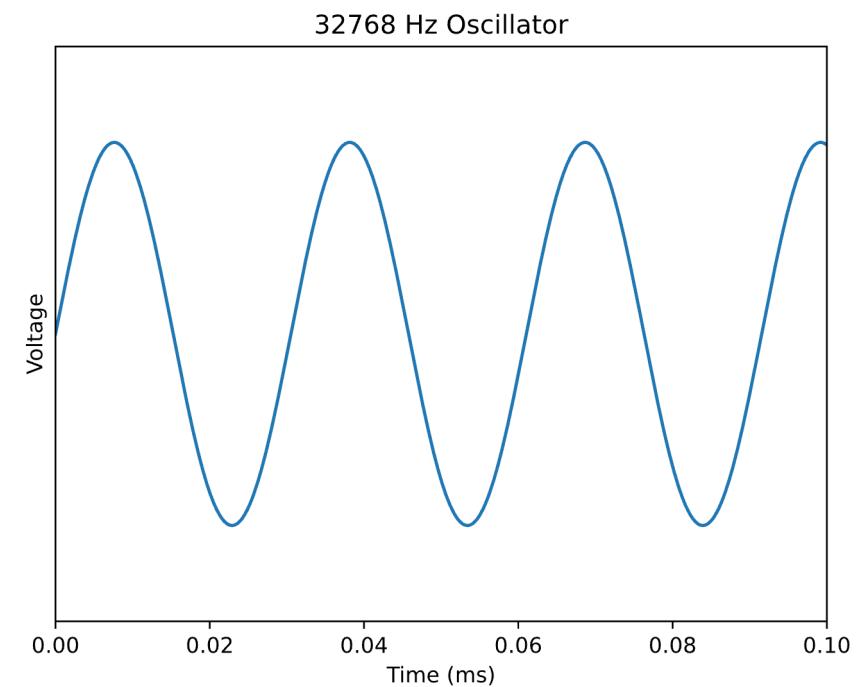
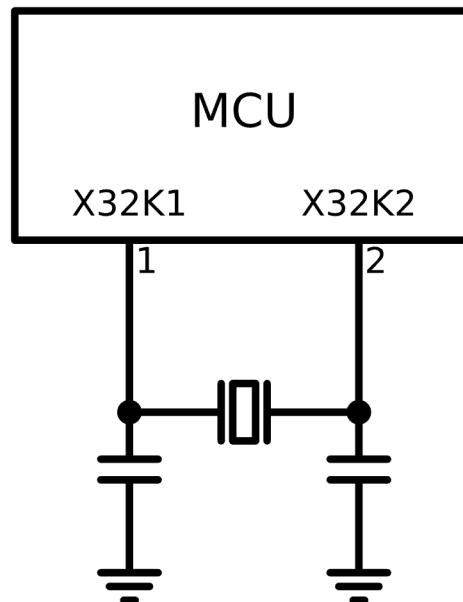
Time Keeping in Distributed Embedded Systems

- When Distributed Embedded Systems need to keep the time?
- System-Level Functions
 - Schedule periodic system events
 - Schedule timeouts
 - Schedule wakeup events
 - Generate clock signals
 - ...
- Application-Level Functions
 - Schedule periodic events
 - Timestamping distributed events or sensor data (data fusion)
 - Alarms
 - Real-world time for the user
 - ...

Oscillators



- Crystal Oscillator
 - A component that generates a signal at a specific frequency
 - From KHz to GHz
- Oscillators are used to keep track of time
- Oscillators can be internal or external



Programmable Timer

- Crystal Oscillator generates pulses at specific frequency
 - On each pulse a counter is decremented
 - When it reaches zero is generate an event (clock interrupt)
- The initial value of the counter controls the time until the event
 - Example: If initial value is 1000, the event will be generated in 1000 pulses or, assuming a $f=32768$ Hz crystal, in $1000/f = 30.5$ ms

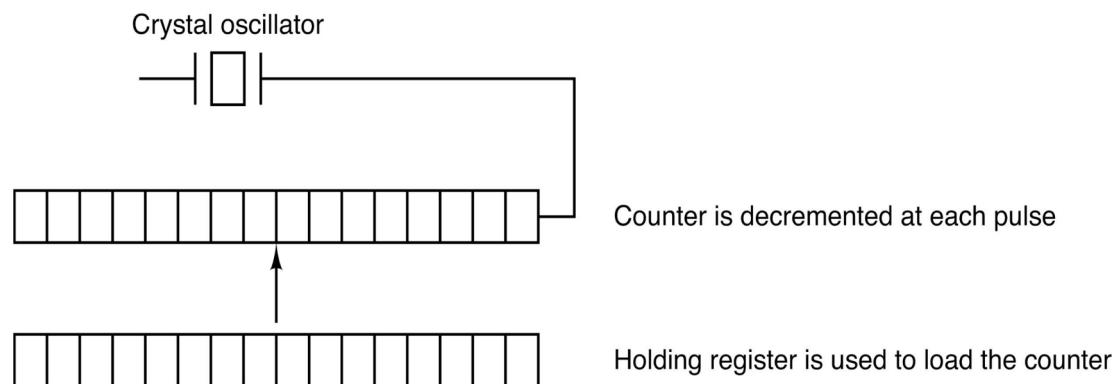


Image source: Modern Operating Systems by Tanenbaum and Bos

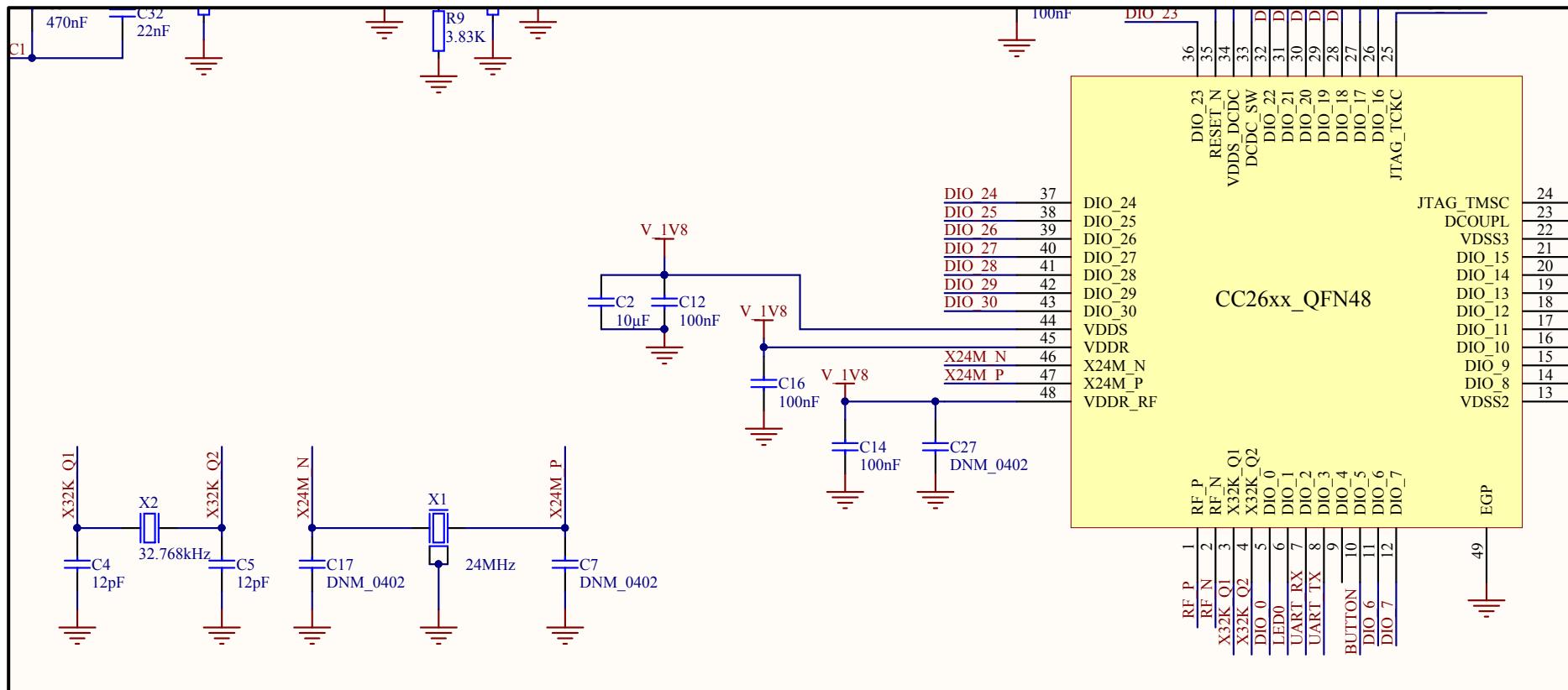
Timer Configuration and Modes

- Prescaler
 - Decreases the timer frequency
 - Decrement the counter every N oscillator pulses
- One-off mode
 - Clock starts with initial value and issues an interrupt when zero
 - Good for one-off timeouts, alarms
- Periodic mode (or square-wave mode)
 - Clock automatically restarts after interrupt is issued
 - Good for periodic events
 - The period is controlled by the timer frequency and the initial counter value

Time Resolution

- The resolution of timestamps, time intervals, etc depends on the crystal frequency
- Clocks measure essentially time in ticks (not in seconds)
 - A 32KHz crystal has a tick of 30.518 us
 - A 24MHz crystal has a tick of 41.667 ps
- Trade-off: higher frequency -> higher energy consumption
- Embedded devices often have two clocks, by incorporating two crystals:
 - A high frequency crystal that is used during active modes to clock the CPU and high-speed peripherals, measure time with high resolution, etc...
 - A low-power low-frequency crystal, typically 32KHz, used for keeping the real time even when the system is off and schedule wake-up events

Crystal Oscillators on CC2650



Example: Uptime Counter

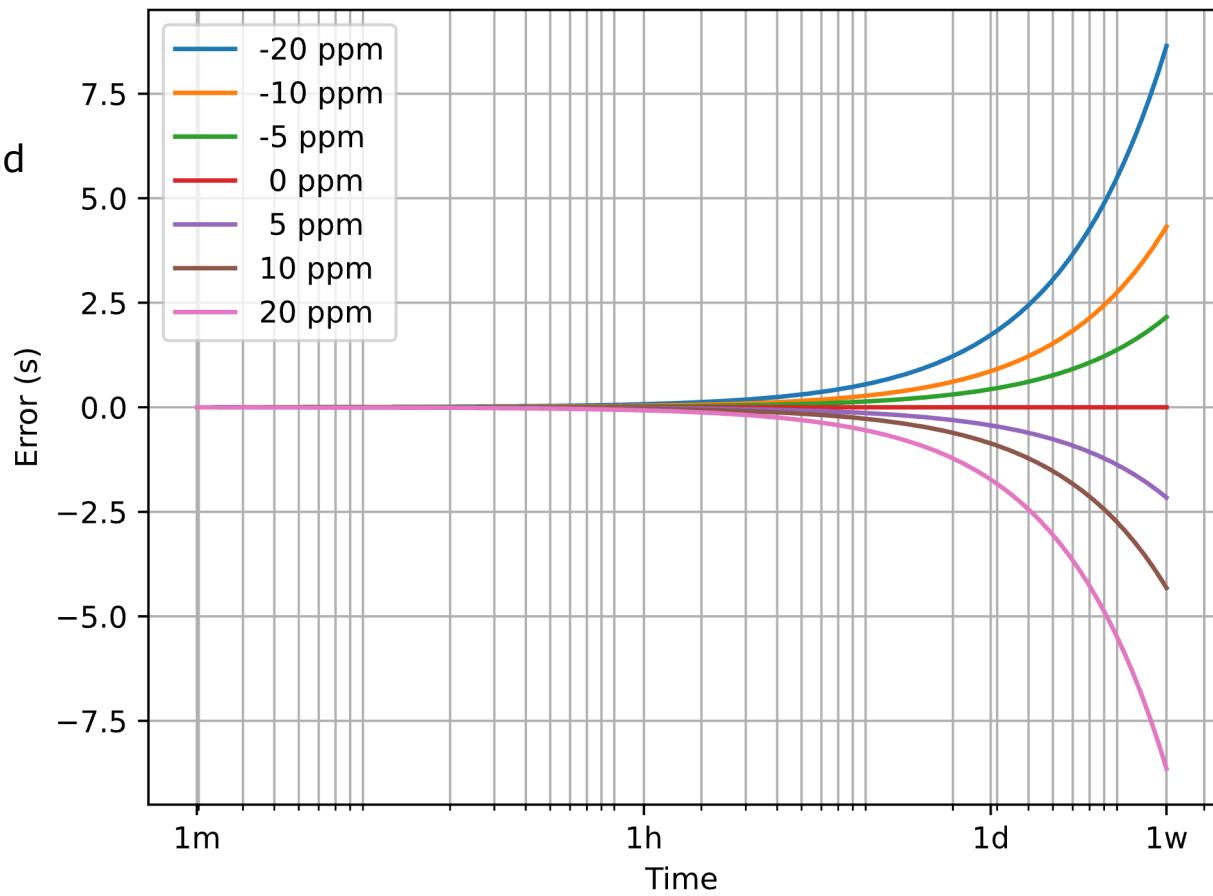
- We want a counter that counts how many seconds passed since the system booted
- Implementation
 - Our system has a timer that is based on a 32768 Hz crystal oscillator
 - On boot we initialise a uint32 UPTIME variable at 0 and...
 - We set a periodic timer with starting value 32768
 - On timeout event, we increment UPTIME
- Question:
 - After exactly one day (86400 seconds) what is the value of our counter?

Clock Drift

- Production Spread
 - All electronic components are manufactured with some error from their nominal value
 - The worst-case error is specified in the datasheet
- Crystal Oscillators are precise but imperfect
 - Crystal oscillators are produced with a frequency variance
 - Their actual oscillating frequency is slightly different than the nominal
 - The error is measured in ppm (parts per million) or ppb (parts per billion)
 - $1 \text{ ppm} = 10^{-6} = 0.0001\%$
 - E.g., FC-135 has max ± 20 ppm drift
- Example
 - A 32768 Hz crystal oscillator with ± 20 ppm maximum drift would have an actual frequency somewhere between 32767.34464 Hz and 32768.65536 Hz
 - In other words, the period is somewhere between 30516.97 ns and 30518.19 ns
 - It looks tiny, but it adds up!

Error

- Error is defined as the difference between the true time passed and the value of UPTIME
- Error can be positive or negative
- Time error adds up
- Negligible in the short term
- Significant in the long term



Other Sources of Clock Drift

- Ageing
 - Systematic changes in frequency with time due to internal changes in the oscillator
 - E.g. FC-135 ages max ± 3 ppm/year
- Temperature
 - Crystal oscillators are calibrated to yield the nominal frequency at room temperature
 - The slow down at very low and very high temperature following a parabolic pattern
 - E.g. FC-135 drifts max -0.04 ppm / $^{\circ}\text{C}^2$ difference to $25\text{ }^{\circ}\text{C}$
- Different sources of drift add up!

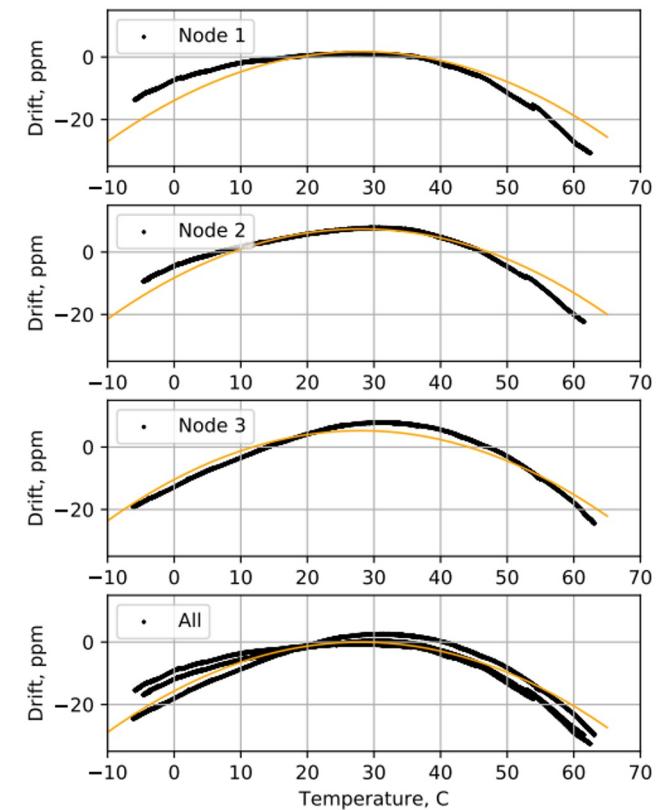
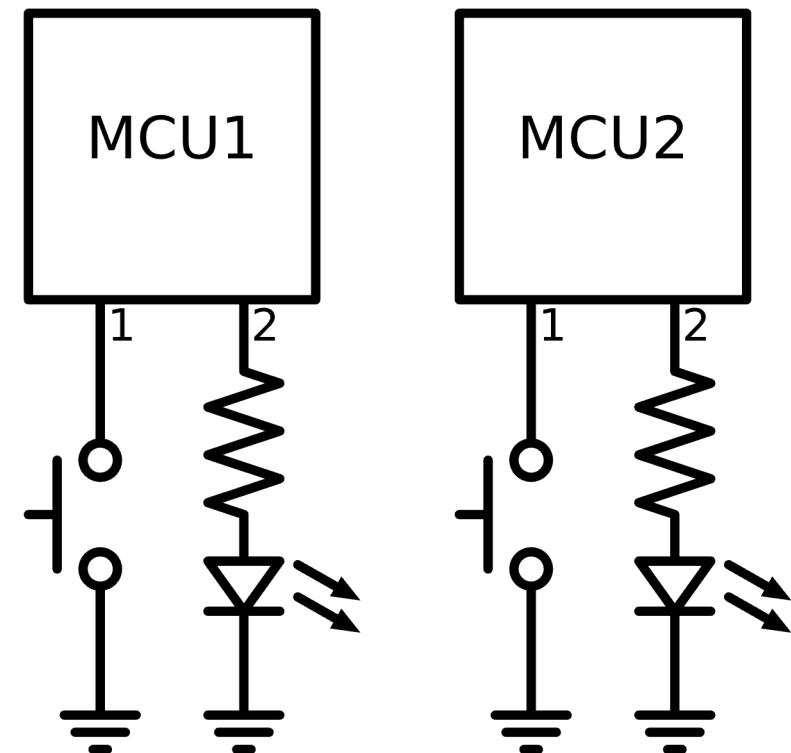


Image source: <https://doi.org/10.1109/TII.2017.2778746>

Example: Synchronised Blinking

- Start both MCUs at the same time by simultaneously push the button
- Blink LED with 1 second period using timer
- If we observe them 1 minute later will the blink in sync?
- If we observe them 1 hour later will the blink in sync?
- If we observe them 1 day later will they blink in sync?

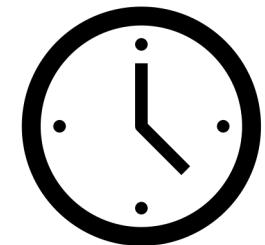


Clock Drift vs Relative Clock Drift

- Clock Drift
 - The rate at which a clock drifts from the true time
 - For the uptime application, we care about being in sync with the true time
- Relative Clock Drift
 - The rate at which two clocks drift from each other
 - To calculate the relative clock, we subtract the clock drifts ($\text{MCU1 drift} - \text{MCU2 drift}$)
 - For the blinking application, we care about the two devices being in sync with each other
- Two clocks can be in sync with each other but drift from true time
 - For example, both MCU1 and MCU2 drift is +10 ppm (relative drift = 0 ppm)
- Worst case relative drift is double the individual worst case clock drift
 - Assuming both MCU1 and MCU2 use an oscillator with ± 20 ppm maximum drift
 - Relative drift ranges from -40 ppm (MCU1 -20 ppm, MCU2 +20 ppm) to +40 ppm (MCU +20 ppm, MCU2 -20 ppm)

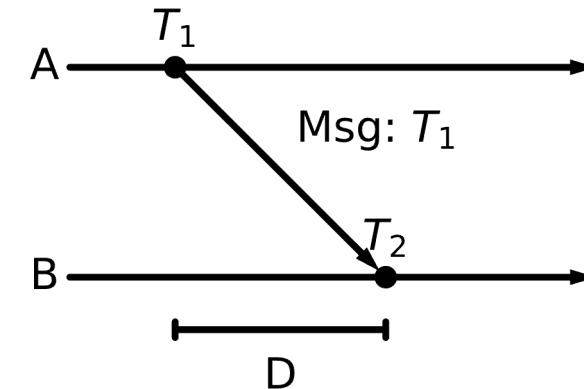
What is true time anyway?

- Atomic Clocks
 - A clock that measures time by monitoring the resonant frequency of atoms
 - Most accurate means we have to measure time
 - Expensive, e.g. \$30K+
- International Atomic Time (IAT)
 - Weighted average of over 450 atomic clocks at various laboratories worldwide
 - Continuous time without leap seconds
- Coordinated Universal Time (UTC)
 - Includes leap seconds due to earth's slowing rotation
 - As of 2022, UTC is 37 seconds ahead of IAT
 - Local time zones are based on UTC (e.g. CET is UTC+1)
- Expensive to have an atomic clock in each embedded system
 - So we have to periodically synchronise the imperfect clocks



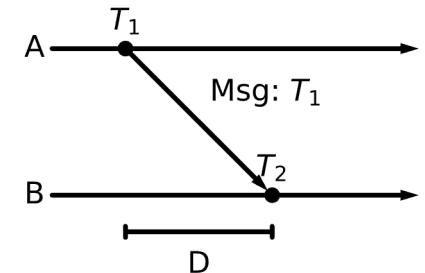
Unidirectional Synchronisation

- Alice takes a local timestamp (T_1) and immediately puts it in a packet and sends it to Bob
- Bob takes a local timestamp (T_2) the moment it receives the message
- $T_2 = T_1 + D + o$, where:
 - D is the message delay
 - o is the time offset
- If I figure out the message delay, I can calculate the offset and synchronise the clocks
 - It is impossible to *measure* the delay unless the two clocks are synchronised in the first place!
- If the delay is relatively stable, it might be possible to *calculate* or *estimate* it
 - The more accurate the estimation the more accurate the synchronisation



Components of Message Delay

- Operating System delays at Sender
 - Scheduling delay: wait for the operating system to run other processes
 - Processing delay: creating the message, copy it to the buffer of the network interface, etc
 - Queuing delay: wait for other messages to be sent first
 - Access delay: wait for medium to be clear, backoffs, receiver to wake-up
- Transmission delay: the time it takes to transmit all the bits
 - Easy to calculate: L/R where L is the packet size and R is the transmission rate
- Propagation delay: the time it takes for the signal to reach the receiver
 - Easy to calculate: d/s where d is the distance and s is the signal propagation speed ($\sim c$)
 - Negligible at short distances, considerable at satellite communications
- Operating System delays at Receiver
 - More scheduling and processing delay
- Multi-hop transmissions add up each link delay



Time Synchronisation with GPS

- Global Positioning System (GPS) or other equivalent systems (e.g. Galileo)
 - Known for localisation but time synchronisation is integral to the process
- 24 satellites
 - In predictable orbits so that at least 4 are visible from anywhere on earth
 - Each equipped with an **atomic clock**
 - Embedded systems dedicated for GPS operations
 - Broadcast periodically their time on a dedicated frequency
 - Delay is bound by the propagation delay
- GPS receiver receives the signal of 4 satellites and calculates its location (3D Trilateration)
 - In the process, it also synchronises with them



A 1D Example



- Let's assume unidimensional world (a line)
 - S1 is at location 0 and R is at location 200 m
 - Signals travel at $s=1$ m/s
 - The GPS Receiver (R) does not know its own location but knows the location of S1
- S1 sends the signal at $T_1=0$ and the signal arrives at $T_2 = 200$
 - R receives it and marks it with a local timestamp (T_2')
- If R is in sync with S1 ($T_2'=T_2$), then: $T_2' = T_2 = T_1 + d_{R-S1}/s$, $d_{R-S1} = 200$ m
 - Estimated Location: 200 m (correct)
- If R out of sync with S1 with offset 1 s ($T_2'=T_2+1$), then: $T_2' = T_1 + d_{R-S1}/s + 1$, $d_{R-S1} = 201$ m
 - Estimated Location 201 m (1 m error)

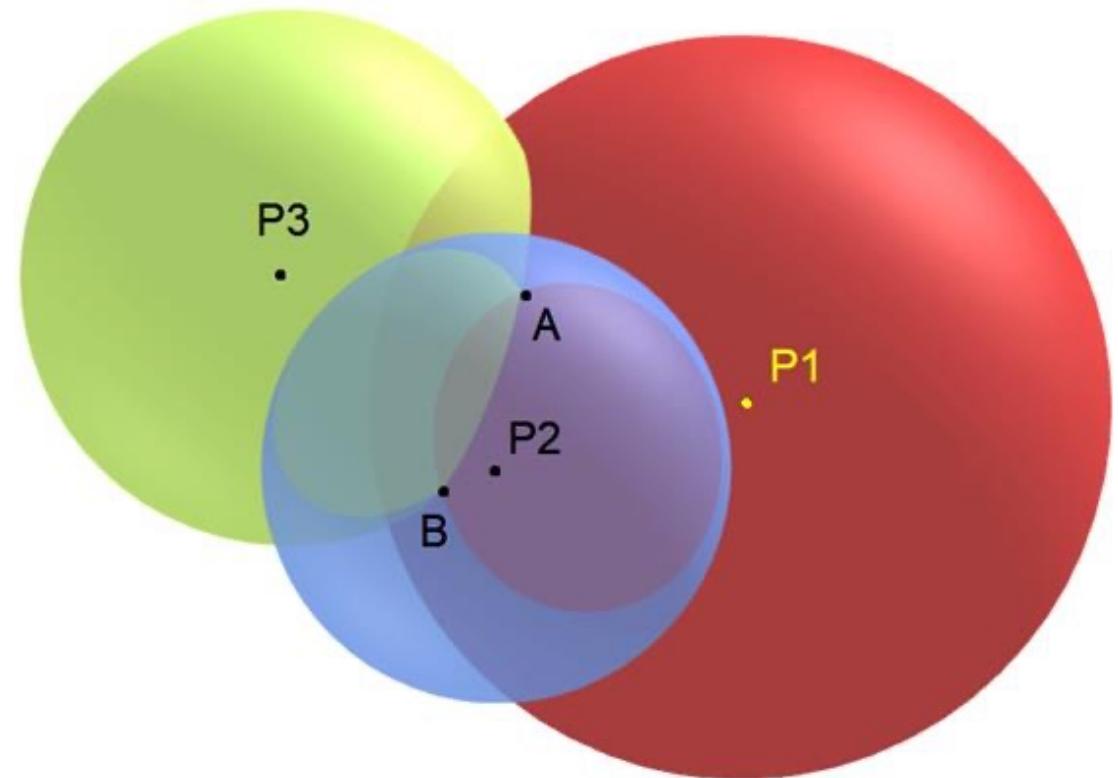
A 1D Example



- R consults a second satellite S2
 - S1 is at location 0, S2 is at location 300 m and R is at location 200 m
 - Signals travel at $s=1$ m/s and R knows the location of S1 and S2
- As before, S2 sends the signal at $T_3=0$ and the signal arrives at $T_4 = 100$ s
 - R receives it and marks it with a local timestamp (T_4')
- If R out of sync with S1/S2 with offset 1 s:
 - $T_2' = T_1 + d_{R-S1}/s + 1$, $d_{R-S1} = 201$ m (Estimated Location: 201 m)
 - $T_4' = T_3 + d_{R-S2}/s + 1$, $d_{R-S2} = 101$ m (Estimated Location: 199 m)
- R can then apply a correction to its timestamps until the two estimations are equal
 - This identifies the true location and the time offset from the satellite clocks

3D Trilateration

- In the 3D world, GPS Receivers need signals from 4 satellites to do the same calculations
- First signal localises the receiver on a sphere
- Second signal localises the receiver on a circle
- Third signal localises the receiver on either of two points (but only one is one earth)
- Fourth signal is used for time synchronisation and, thus reducing the localisation error



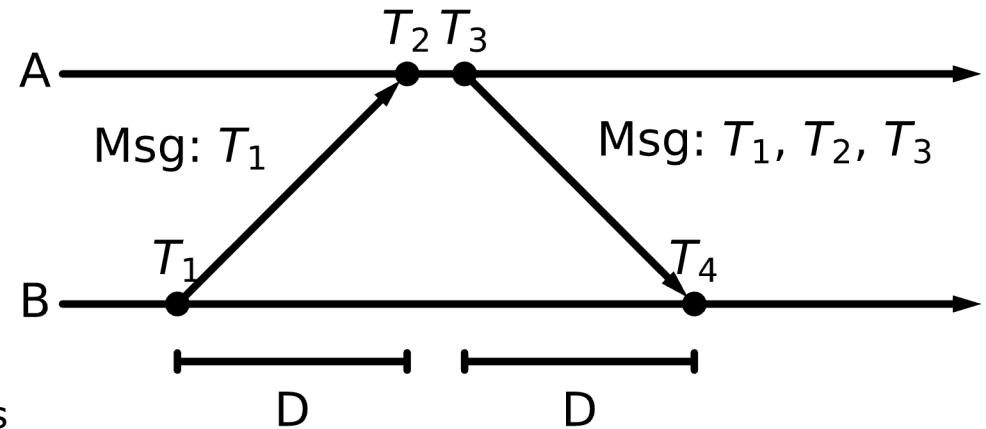


Time Synchronisation with GPS

- Advantages
 - Easy time synchronisation solution with high precision (1-hop access to an atomic clock)
 - GPS receiver hardware is relatively cheap (depending on the use case)
- Disadvantages
 - Requires line of sight (typically works only outdoors)
 - Consumes significant energy for battery-powered systems
 - GPS receiver hardware is required and is relatively costly (depending on the use case)
 - E.g. A modern car would have a GPS receiver but not every single sensor/subsystem of a car would have one

Bidirectional Synchronisation

- We cannot measure unidirectional delay unless sender and receiver are already synchronised
- But we can measure **round-trip delay**
- Assumptions
 - Delay is identical in both directions
 - Clock drift does not change between messages
- Simple system of linear equations
 - $T_2 = T_1 + D + \sigma$
 - $T_4 = T_3 + D - \sigma$
- Solution: $\sigma = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}$

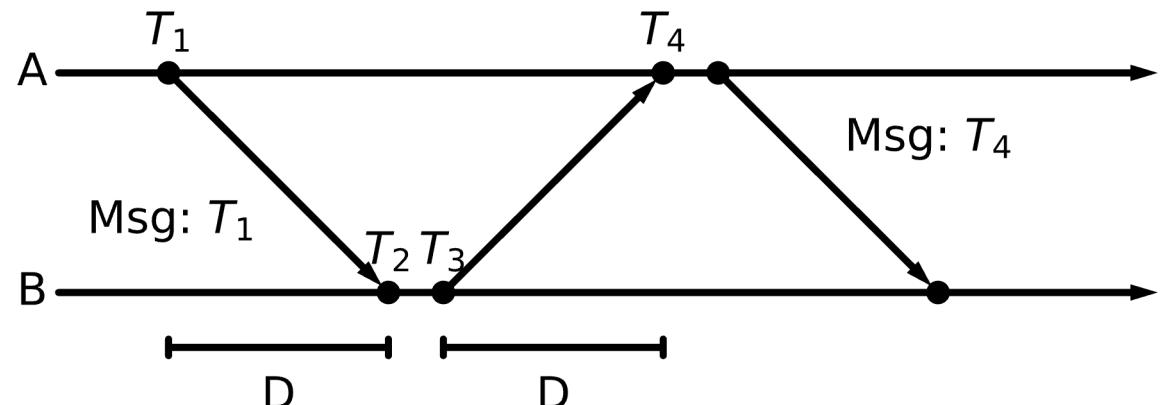


Network Time Protocol (NTP)

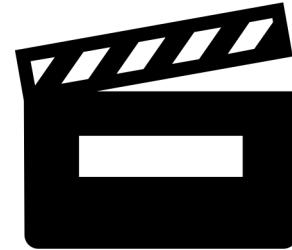
- Client/Server model (system queries a time server that has access to a good clock, e.g. GPS)
- Uses bidirectional synchronisation
- Can synchronise systems over variable-latency networks
- Achieves ms accuracy over the public internet and <ms accuracy over local networks
- Vulnerable to congestion, asymmetric routes, and asymmetry in general
- NTP queries multiple time servers, keeps history of clock drift, applies corrections
- Simple NTP (or SNTP) is a simplified client that makes one query to a single time server
 - Less accurate but more lightweight

Precision Time Protocol (PTP)

- Time server broadcast message with timestamp, devices send request to identify round-trip delay
 - Variation of the bidirectional synchronisation
- $$o = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}$$
- Same assumptions as NTP
 - Symmetric delay
 - Constant offset
- Better accuracy than NTP (ns or better)
 - Time master in local network mitigating queuing/routing delays
 - More accurate timestamps at hardware level mitigating OS-related delays



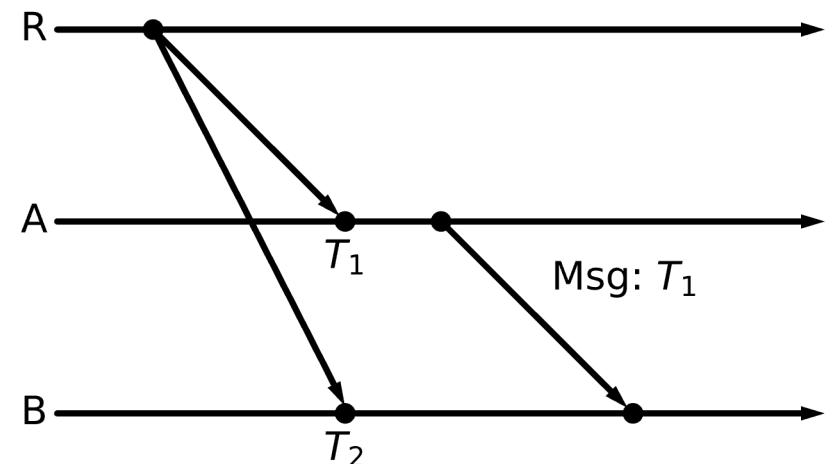
Reference-Based Synchronisation



- GPS, NTP, SNTP, PTP provide synchronisation with true time
- Relative synchronisation can be achieved by capturing a shared event
 - The event can be natural or induced
- Relative offset between clocks of two devices can be calculated by timestamp difference of capturing the event
- Example: Cinema Clacket
 - Creates a distinct audio-visual event that is used to synchronise the clocks of the camera and microphone
- A packet broadcast can serve as a synchronisation event

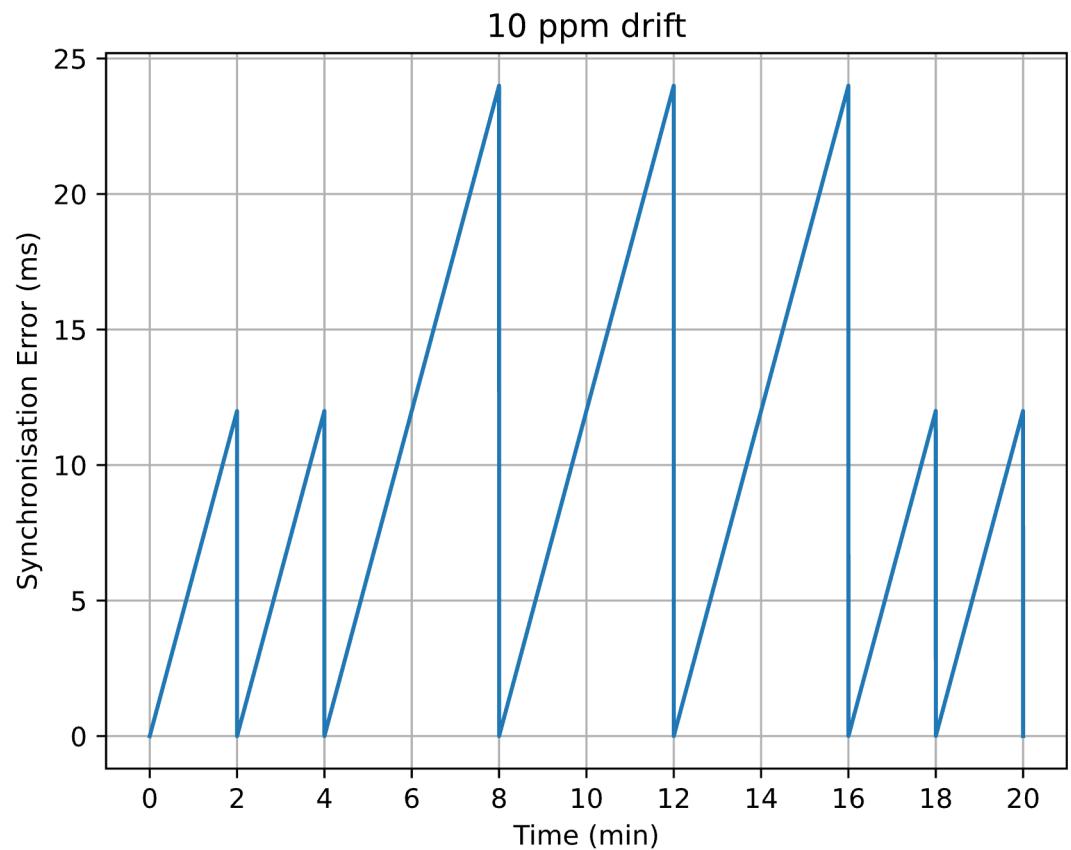
Reference-Broadcast Synchronisation

- A reference device (R) broadcasts a synchronisation beacon
- Device A and B take a local timestamp of the time they receive it (T_1 and T_2)
- Device A sends T_1 to Device B which calculates its relative offset to Device A as: $\delta = T_2 - T_1$
- Sender-related delays affect the beacon the same and propagation delay is almost the same at short distances
- Errors are limited to the differences in receiver-oriented delays



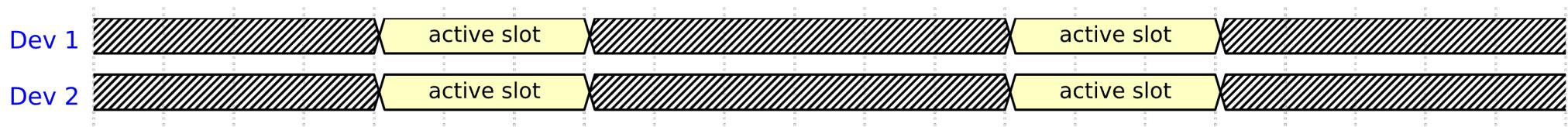
Periodic Synchronisation

- Time error (offset) accumulates over time
- A synchronisation event brings it (close to) zero
- Periodic synchronisation is required to maintain synchronicity
- Time error is contained by the synchronisation frequency
- Trade-off: Cost of synchronisation vs tolerance to synchronisation error



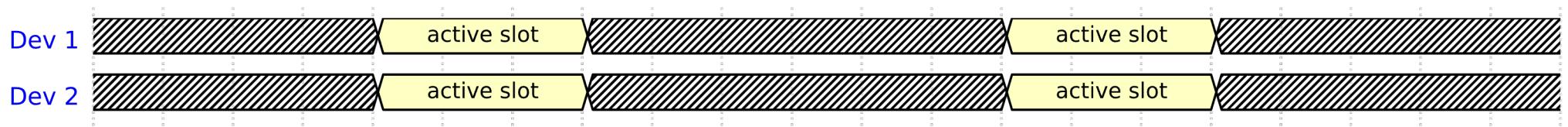
Synchronising the Radio Duty Cycle

- Back to the problem of Synchronous MAC
- Devices agree on a rendezvous point in time to turn on their radios to exchange data
- Example
 - Dev 1 and Dev 2 agree to exchange data in 1 minute
 - Dev 1 puts a timer for 1 minute, goes to sleep, when timer expires it starts transmitting
 - Dev 2 puts a timer for 1 minute, goes to sleep, when timer expires it starts listening
 - What will happen?



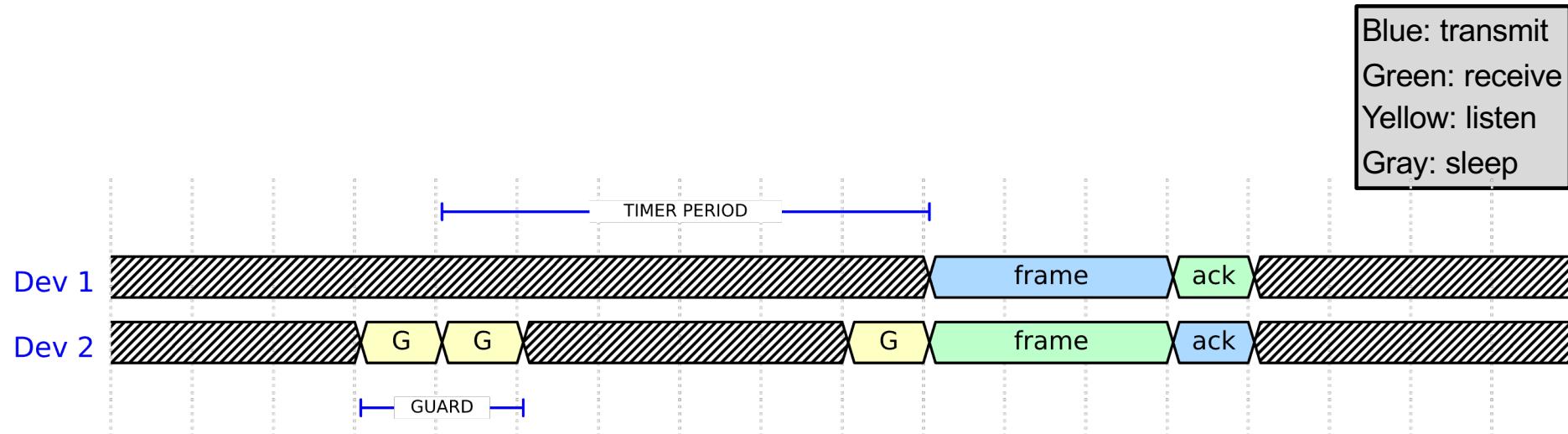
Synchronising the Radio Duty Cycle

- Back to the problem of Synchronous MAC
- Devices agree on a rendezvous point in time to turn on their radios to exchange data
- Example
 - Dev 1 and Dev 2 agree to exchange data in 1 minute
 - Dev 1 puts a timer for 1 minute, goes to sleep, when timer expires it starts transmitting
 - Dev 2 puts a timer for 1 minute, goes to sleep, when timer expires it starts listening
 - If Dev 1 and Dev 2 have no relative drift, it will work OK
 - If Dev 1's clock faster, Dev 2 will miss the transmission
 - If Dev 2's clock faster, Dev 1 will miss the listening window



Guard Time

- The receiver puts the radio in listen mode a bit before the rendezvous point and keeps it on for a while after to account for both positive and negative drift
- Trade-off
 - Large guard time improves robustness to time sync error
 - But it increases the idle listening (and occupies bandwidth)



Guard Time vs Max Sync Period

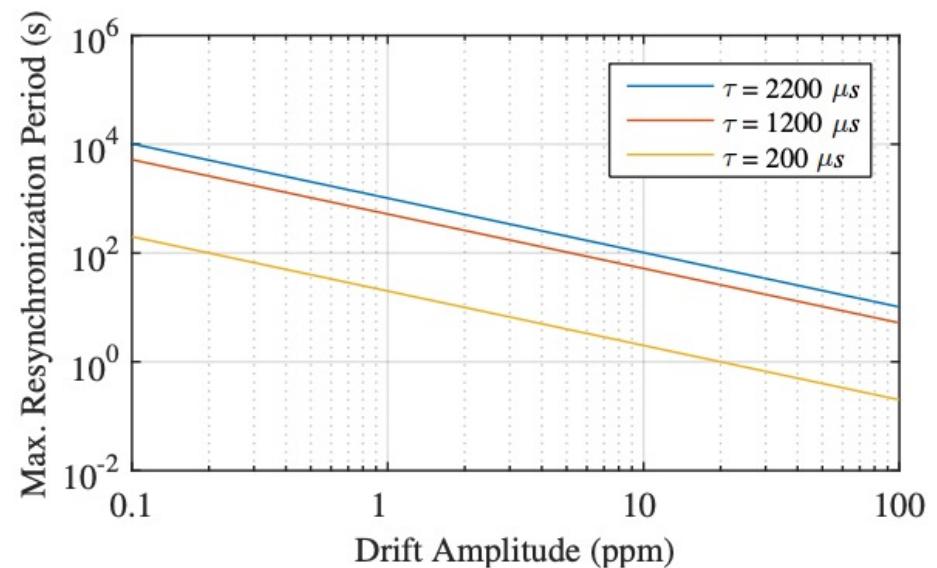
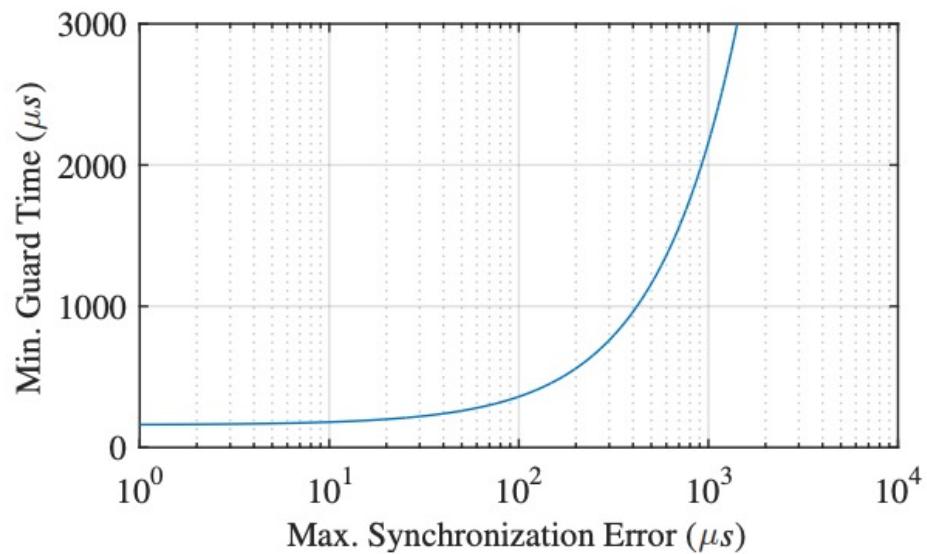
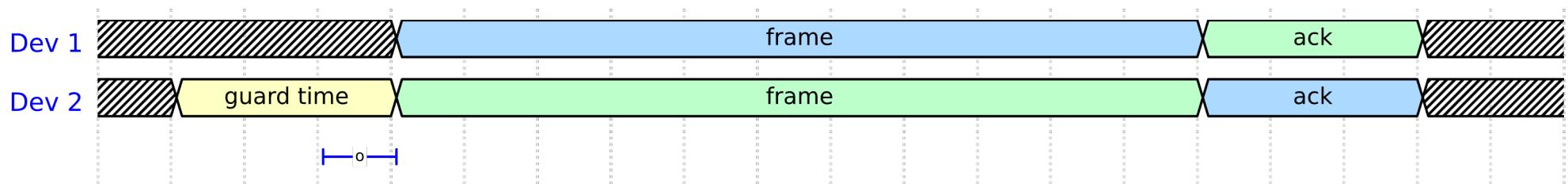


Image source: <https://doi.org/10.1109/LCN.2016.042>

Implicit Synchronisation

- All previous synchronisation mechanisms rely on explicit synchronisation frames (overhead)
- Rendezvous-based MAC can have implicit synchronisation using the data frames
 - Receiver expects the transmission at T_1 but it happens at T_2
 - It can then calculate the offset as $o = T_2 - T_1$
- “Free” resynchronisation on each data transmission
 - A data transmission must happen before the synchronisation error becomes larger than what the guard time can handle
- Runtime drift compensation
 - The drift is constant in the short term (assuming constant temperature)
 - The receiver can learn the drift by diving the offset to the time passed since last synchronisation

Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



Conclusion

- Time synchronisation might be necessary for applications and system-level processes
- Trade-offs
 - Real time synchronisation requires more resources than relative synchronisation
 - High precision synchronisation requires more resources than loose synchronisation
 - The frequency of synchronisation bounds the sync error but requires resources

Embedded Machine Learning

02226 Networked Embedded Systems

Riccardo Miccini

Technical University of Denmark

November 13, 2025

Before we start

- **ML primer:** what is it and what does it do
- **Wetware I:** human needs
- **Hardware:** from cloud to edge
- **Software:** from data to deployment
- **Wetware II:** human responsibilities

This slide deck is dense; we won't cover everything but you can use it to dig deeper if you want

Also: some of the references are missing, sorry! I will definitely might update them later on

Learning Objectives

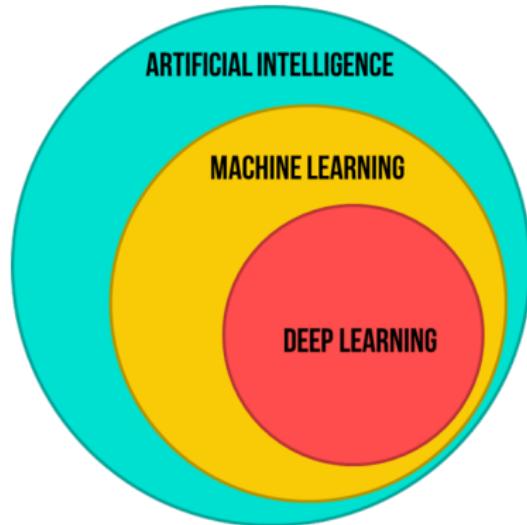
- **Explain** the motivation for performing ML on embedded devices
- **Compare** HW platforms for embedded ML in terms of their capabilities and constraints
- **Describe** model compression techniques and deployment steps
- **Evaluate** embedded ML systems with respect to efficiency, scalability, and societal implications
- **Critically assess** emerging trends and **propose** directions for responsible embedded AI development

Outline

1. Quick introduction to ML
2. Wetware I
 - Motivation and applications
3. Hardware
 - The computing continuum
 - Overview of hardware platforms
 - Neural network acceleration
 - Latest and niche topics
4. Software
 - The embedded ML pipeline
 - Model compression
 - Deployment tools
 - Latest and niche topics
5. Wetware II
 - Responsible embedded ML

Quick introduction to ML

What is even ML/AI/DL?



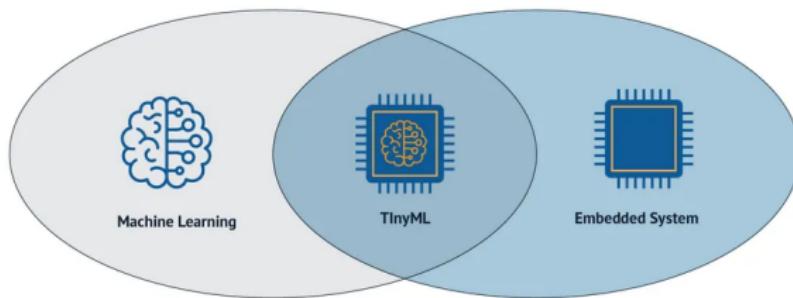
Some terminology

- **Artificial Intelligence**: attempting to perform complex tasks by mimicking human intelligence
- **Machine Learning**: subfield of AI focusing on learning from data rather than explicit rules
- **Deep Learning**: subset of ML techniques based on *deep neural networks* → contemporary AI is mostly deep learning

We will use the terms interchangeably but focus mostly on DL

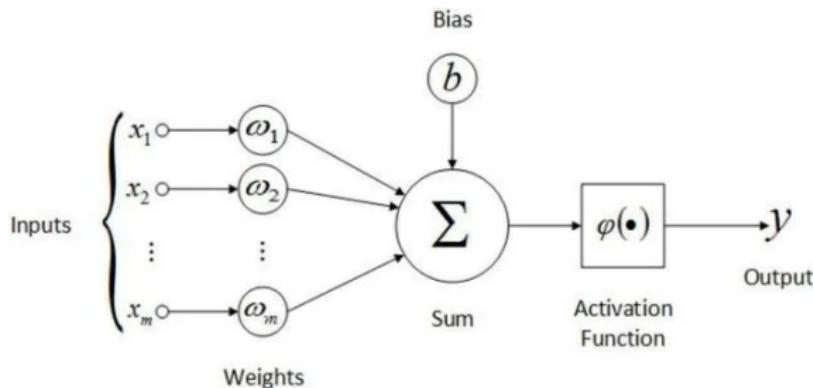
Other buzzwords I should know?

- **Edge AI:** running ML models directly on edge devices (PCs, smartphones, consumer electronics, IoT) rather than in the cloud
- **TinyML:** subfield of ML/DL dealing with the challenge of running model on *low-power, resource-constrained devices* ← our focus
- Less common aliases: AloT, edge computing, embedded ML, resource-constrained ML, resource-efficient ML...



The artificial neuron

- Compute linear combination (weighted sum) of inputs
- Activation: some non-linear operation (e.g., s-shaped function)
- The weights represent how important each input is

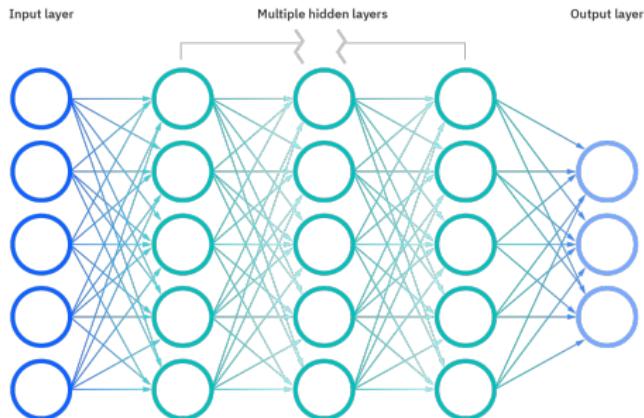


Pretty useless by itself, but if we stack many in series...

Deep neural networks

...we get a **Deep Neural Network**

a *Multilayer Perceptron*, specifically



- Interesting property: **can approximate any (continuous) function**
- But how do we approximate a very complex function
- Example: “*is this bunch of pixels showing a cat?*”

We need to **learn** from data

i.e., known instances of cats and non-cats

Training a DNN

- We start with random weights → model is pretty useless
- We feed some data into the model and get a *prediction* (output)
- We compute the *loss function* (error → how much is the model wrong?)
- We adjust the model weights based on how they contribute to the error
- How are weights actually adjusted? Backpropagation + gradient descent

With enough training examples, the model will learn the task

Is it really that simple?

Yes

- DL is proving effective at many tasks
- How? Larger models + lots of data and compute → *bitter lessons* [1] and *neural scaling laws* [2]

and

No

- Data labeling is mostly manual → expensive and time-consuming
- Training and running big models is (computationally) expensive

Beyond multilayer perceptrons

- Wait! Do we just treat a picture as a vector of pixels?
- Different types of data: text, images, audio, sensors...
- ...require different model architectures (convolutional neural networks, recurrent neural networks, transformers, etc.)

Wetware I

Motivation and applications

What can we use embedded ML for?
What do we *need*?



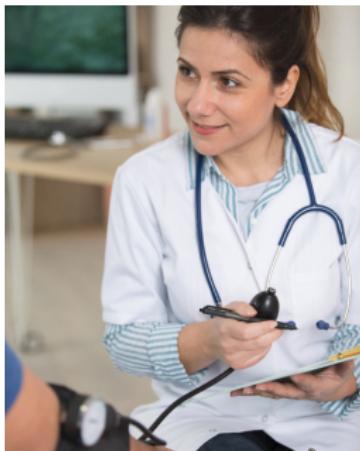
Current challenges

By 2050...

- the global population is expected to reach 9.7 billion
- 22 % of people will be older than 60
- 68 % of people will live in urban areas



Food security



Preventative healthcare



Efficient resource management

Where can embedded ML help?

Manufacturing and industrial



Fault detection,
predictive
maintenance

Healthcare



Wearable monitoring
devices, AI-assisted
diagnostics

Automotive and transportation



Autonomous and
assisted driving,
traffic management

Smart cities and infrastructure



Water, waste, and
power grid
management

Agriculture and food production



Precision farming,
pest detection, yield
prediction

Environmental monitoring



Real-time alerts
(fires, floods,
poaching...)

Why does it need to be *embedded*?

- **Bandwidth**: transmitting raw data from thousands of devices would clog infrastructure
- **Latency**: local processing lowers response time, enabling real-time applications
- **Economics**: maintaining a cloud infrastructure and transmitting data to it is expensive
- **Reliability**: more fault-tolerant systems due to fewer external dependencies
- **Privacy**: fewer opportunities for breaches and abuse when data stays on the device

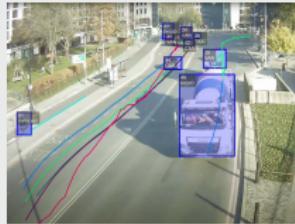
Some real-world examples

Early wildfire detection [4]



- Gas sensors (Bosch BME688) mounted on trees
- On-device ML model recognizes fire-related gases
- When fire is detected, node sends alert to gateway
- Ultra-low-power, always-on, LoRaWAN mesh

Traffic monitoring and management [5]



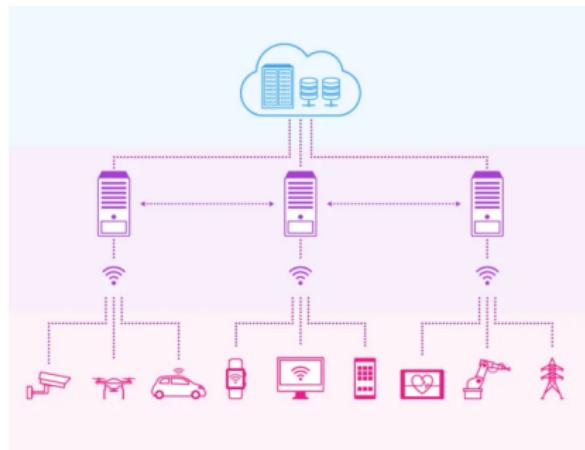
- Camera-based devices installed along roads
- Control traffic lights based on estimated demand
- Collect traffic data for data-driven policy making
- Multiple computer vision models running on-device

Hardware

The computing continuum

The computing continuum

- **Cloud:** centralized, large-scale compute and storage → high throughput, high latency
- **Fog:** intermediate gateways or local servers → pre-processing, caching, aggregation, filtering...
- **Edge:** end nodes (IoT, industrial or user devices) → real-time operation, limited resources



Real-world solutions often operate at multiple levels

Example: smart speaker

- Edge processing for wake word detection and noise suppression
- Cloud processing for speech recognition and voice assistant

What makes embedded ML different?

In the cloud...

- Virtually **unlimited compute**, memory, and storage
- Less latency-sensitive; data can be **processed in batches**
- Models can be retrained often and **deployed instantly**
- Relatively **general-purpose** HW; can run any model

On the edge...

- **Constrained resources** (e.g., due to cost or battery operation)
- **Always-on**, streaming, and **real-time** data processing/operation
- After deployment, **updating models is difficult** or impossible
- Application-dependent HW with **limited flexibility** and support
- Models must be **tailored** to the HW (or vice versa)

Hardware

Overview of hardware platforms

Overview of hardware platforms

Name	Characteristics	Applications	Examples
Microprocessor (CPU/MPU)	High freq, multi-core, sophisticated instructions and cache hierarchy	Robotics, automotive, home automation, medical, smartphones	x86, ARM Cortex-A
Microcontroller (MCU)	Low freq, low power, built-in memory and peripherals	Wide range; wearables, personal devices, IoT, industrial automation	ARM Cortex-M, RISC-V, ESP32
Graphics processing unit (GPU)	Single instruction, multiple threads; high parallelism, bandwidth, and throughput	Multiple heavy workloads, computer vision, smartphones	NVIDIA Jetson, Adreno
Neural processing unit (NPU)	Heterogeneous designs; low-precision math, multi-level local memory, static graph execution	Anything requiring accelerated neural network inference	Qualcomm HTA, ARM Ethos-U/N, Google EdgeTPU, Myriad X, NVIDIA DLA

Real-world solutions often combine multiple platforms

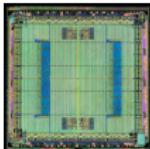
Other platforms and terms

DSP (digital signal processor)

- Optimized instructions (SIMD, fixed-point math, HW loops)
- Used for audio, radar, image, telecommunications...

FPGA (field-programmable gate array)

- Reconfigurable digital logic
- Prototyping or low-volume domain-specific tasks

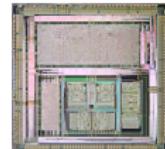
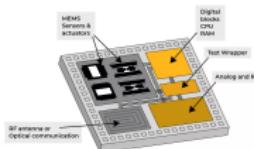


SoC (system-on-chip)

- Processing, memory, storage, and connectivity on one die
- Flexible/adaptable: from mobile phones to IoT

ASIC (application-specific integrated circuit)

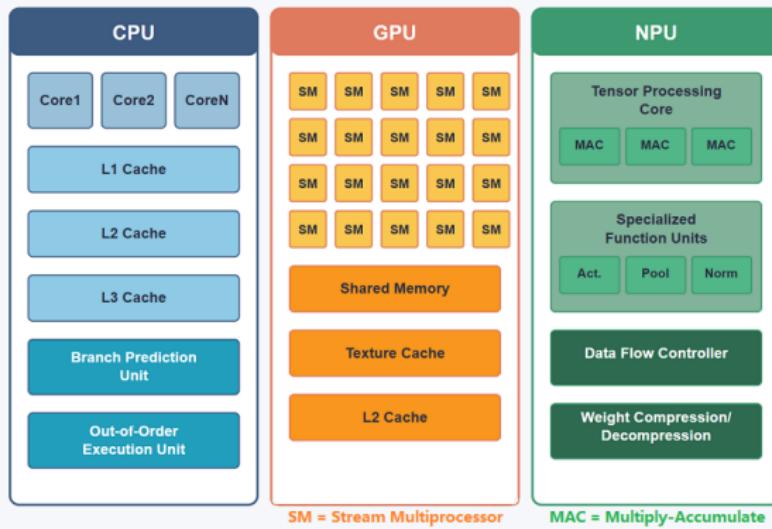
- High-efficiency, single-purpose chip (e.g., video encoding)
- Like FPGA but for large volumes



What changes in practice?

Different HW for different workloads

- CPU: few cores with many specialized instructions → **general-purpose**
- GPU: many simpler cores running same instruction → **parallel processing**
- NPU: fixed-function HW (MAC arrays, activations) → **neural networks**



Real-world examples

Reolink RLC-810A surveillance camera



- SoC: Novatek NT9852x (2x Cortex-A9 CPU + NPU)
- On-device: motion detection, object recognition (person, vehicle, animal)
- Fog (hub device): local storage, video captioning
- Cloud: remote storage, natural language querying

Butterfly Network iQ3 handheld ultrasound probe



- SoC: Microchip MPFS250T (4x RISC-V CPU + FPGA)
- On-device: image acquisition and preprocessing
- Smartphone: various predictive and diagnostic models (bladder volume, B-line count)
- Cloud: data storage, AI model marketplace

Hardware

Neural network acceleration

Why do we need custom accelerators?

Are neural networks any special?

- Pre-determined, **static computational graphs** (no conditional branching, fixed-length loops) → no need for control flow
- Inputs and model weights are **used multiple times** → keep data close to computation, avoid unnecessary memory transfers
- Mostly boil down to **multiply-accumulate (MAC)** operations
- Can often run with very **low-precision** arithmetics (INT8 or less)

$$\begin{matrix} \text{I} \\ (\text{inputs}) \end{matrix} \quad * \quad \begin{matrix} \text{W} \\ (\text{weights}) \end{matrix} \quad = \quad \begin{matrix} \text{O} \\ (\text{outputs}) \end{matrix}$$

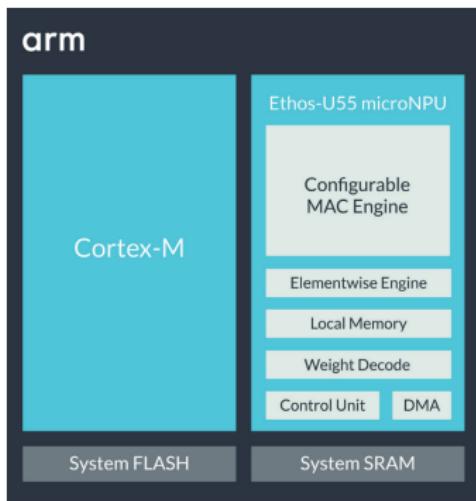
Design priorities for NPUs

- Energy efficiency over flexibility
- Minimal data movement
- High compute utilization
- **Maximize throughput per Watt**

Architecture overview

Case study:

ARM Ethos-U65 microNPU
(IoT, wearables, consumer...)



Control logic

- **Host MCU** requests an inference job
- **DMA** handles different streams (commands, input, output, weights...)
- **Control unit** parses command stream and orchestrates data transfer and computation

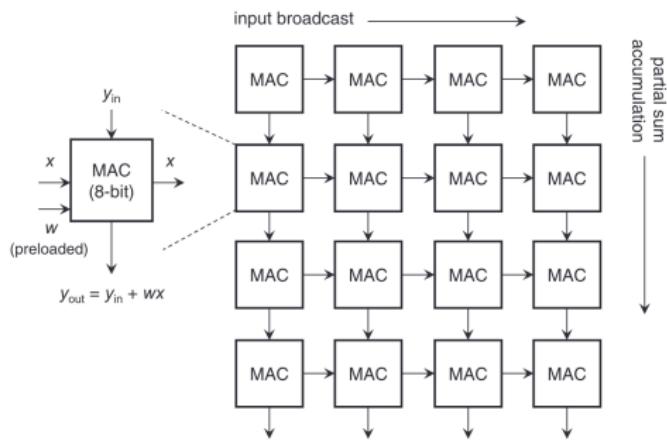
Compute units

- **MAC engine** accelerates dot products
- **Elementwise engine** parallelizes activation functions and other ops
- Weights for current layer and partial results are kept in **local memory**

Systolic arrays

Goal: perform many MACs with minimal overhead

- Weights are pre-loaded into each MAC unit
- Input data flows left to right →
- Partial results flow top to bottom ↓



Advantages

- Input/weights reuse
- Local data movement
- Minimal overhead
- High throughput

Hardware

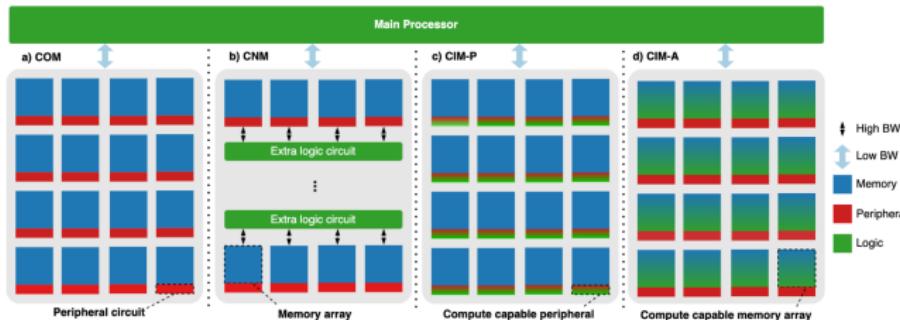
Latest and niche topics

Compute-in-memory

Problem: modern computational workloads are increasingly more data-intensive → computing systems are **bottlenecked by memory**

Compute-in-memory: a paradigm shift to...

- Perform computation where it makes sense (i.e., close to the data)
- Make computing architectures more data-centric

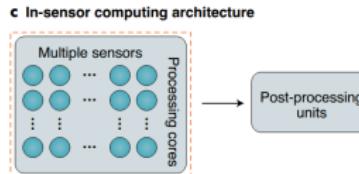
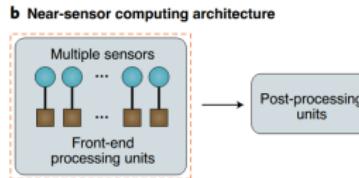
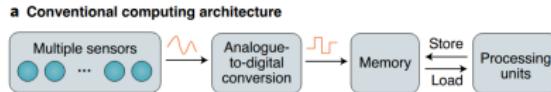


In/near-sensor computing

- **Problem:** data transfer from external peripherals is even more expensive and slow
- **Solution:** offload some computation (preprocessing, feature extraction) to the sensor itself → transfer less data

Examples

- **IMU** (accelerometer, gyroscope): step count, gesture recognition
- **Microphone**: voice activity detection, keyword spotting
- **Camera**: region-of-interest extraction



Neuromorphic computing

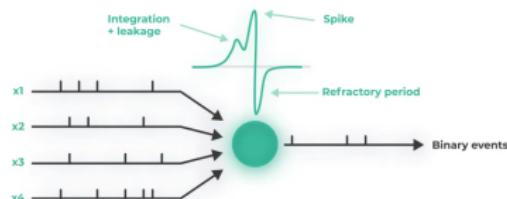
- **Conventional neural networks:** clock-driven, synchronous, and dense matrix operations → power-hungry
- **Neuromorphic systems:** event-driven, asynchronous, and sparse spiking neurons → efficient and biologically motivated

Main technologies

- **Spiking NNs:** mimic activation patterns of biological neurons
- **Specialized HW:** mixed-signal or digital neuromorphic accelerators, spike encoders/decoders
- **Event-based sensors:** asynchronous cameras/microphones emitting spikes only on intensity/amplitude changes

Advantages

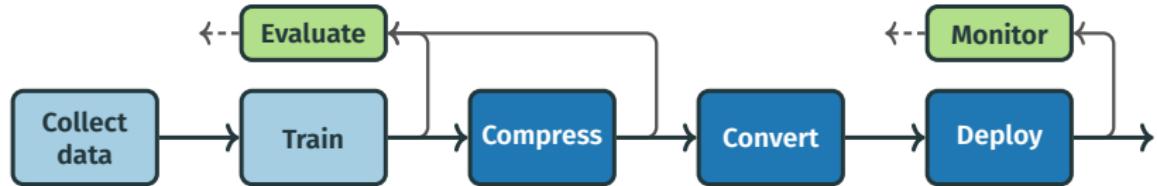
- Ultra-low power
- Continual learning



Software

The embedded ML pipeline

The embedded ML pipeline



- **Collect data:** acquire, label, and pre-process data
- **Train:** design model and train it, often on GPUs/cloud
- **Compress:** reduce size and computational complexity
- **Convert:** export to intermediate format
- **Deploy:** compile into executable code and integrate into firmware

Evaluate/monitor what, exactly?

- After compression: check for performance degradation using task-specific metrics (e.g., accuracy, mean squared error)
- In production: make sure model performs as expected in real-world conditions (and potentially acquire new data)

What embedded ML also cares about

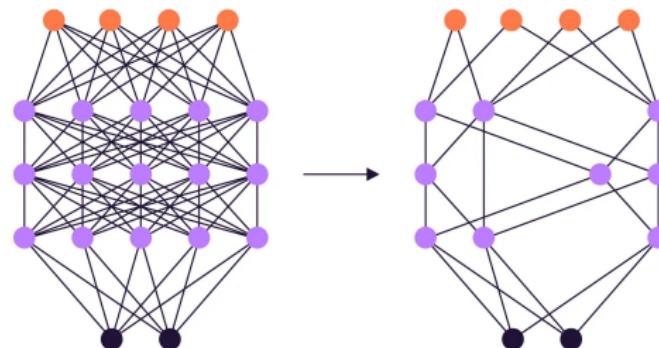
- **Model size** (kB, MB): determined by parameter count and precision
- **Operations** (FLOPs, MACs): related to model size and architecture
- **Inference time** (μ s, ms): duration of inference on single sample
- **Throughput** (FPS): amount of data processed per unit of time
- **Memory footprint** (kB, MB): working memory required to perform inference, including parameters and the intermediate activations
- **Energy consumption** (μ J): energy required to perform inference

Software

Model compression

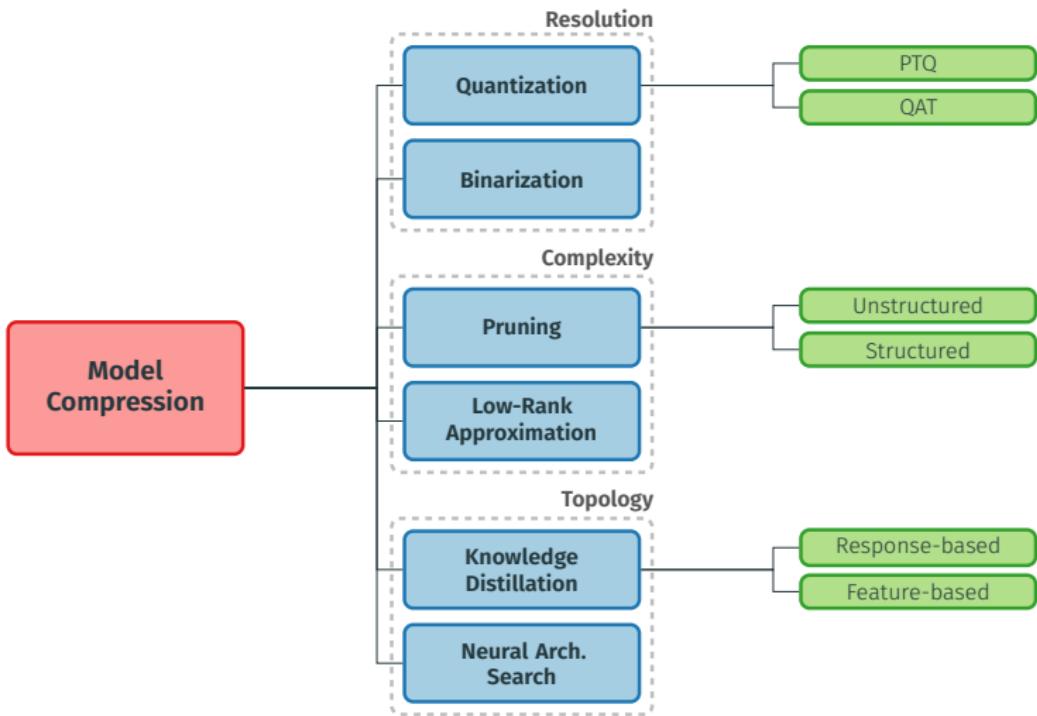
Model compression: what and why

Model compression reduces the resource requirements of a model while maintaining its performance



- DL models are often **overparameterized** and **redundant**
- Model compression helps us figure out what's worth keeping

Model compression: taxonomy

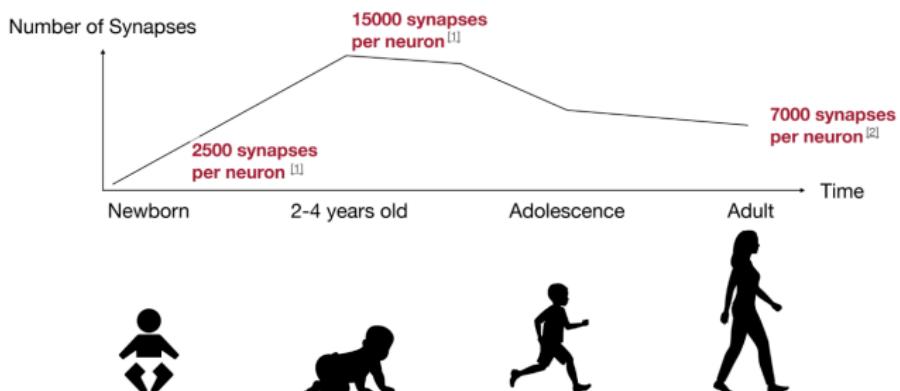


Model compression: classification of techniques

- **Quantization** – including **Binarization** – reduces the numerical precision, i.e., the number of bits per parameter or operation
- **Pruning** and **Low-Rank Approximation** reduce the computational complexity, i.e., the number of arithmetic operations
- **Knowledge Distillation** and **Neural Architecture Search** alter the model topology, resulting in smaller or fewer layers

Model compression: pruning (1)

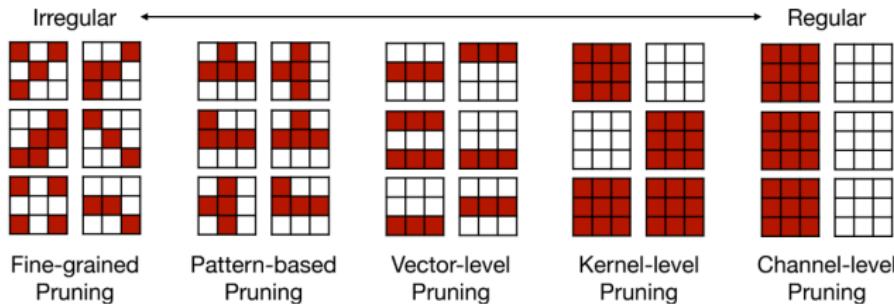
Pruning removes redundant or less important parameters from a neural network, reducing size and computation [6, 7]



Model compression: pruning (2)

Granularity: how should we prune?

- **Unstructured**: removes individual weights without considering position; results in sparse matrices → requires specialized HW
- **Structured**: removes entire groups (filters, channels, neurons, attention heads); lower compression but works with generic HW
- **Semi-structured**: removes blocks/patterns, balances compression ratio and HW support (easier to accelerate)



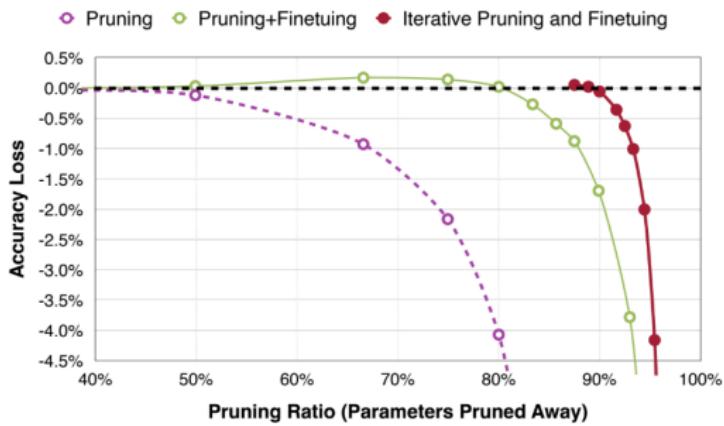
Model compression: pruning (3)

Pruning criteria: what should we remove?

- **Weight magnitude**: absolute value of weights
- ℓ_1/ℓ_2 **norm**: used for groups of weights in structured pruning
- **Sensitivity/Saliency**: remove weights that don't affect train loss
- **BatchNorm**: use learned scaling factors as channel importance

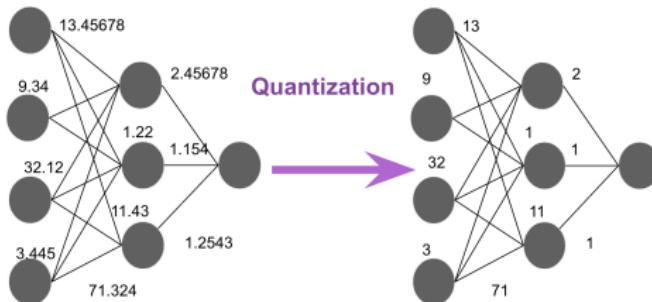
Other aspects

- When to prune?
During training,
after, or iteratively
- How much? Pruning
ratios, global vs.
per-layer



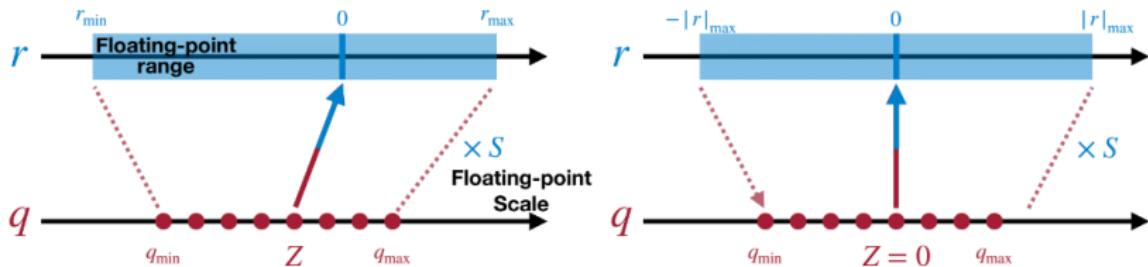
Model compression: quantization (1)

Quantization maps model weights and activations into lower-precision approximations (e.g., `int8` or `int4`) [7–9]



- Quantized models take up less memory and can run on hardware that doesn't support floating-point arithmetics (e.g., MCUs)
- **Uniform quantization** (most common method) is a linear mapping of integer values to floating-point values

Model compression: quantization (2)



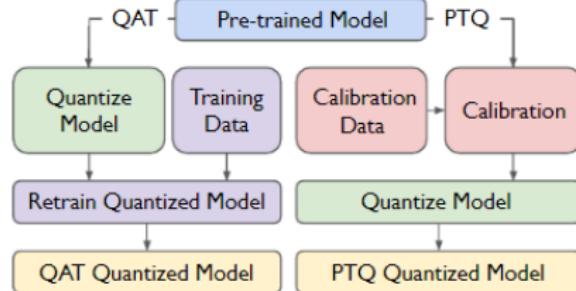
Main parameters

- **Scaling factor S** : distance between quantization levels
- **Zero-point Z** : offset to center the quantization range
- $[r_{\min}, r_{\max}]$: range of real-valued values
- $[q_{\min}, q_{\max}]$: range of quantized values

Model compression: quantization (3)

When to quantize?

- **Post-training quantization (PTQ)**: simple to implement, doesn't require labeled data or training steps, few hyperparameters
- **Quantization-aware training (QAT)**: simulates effect of quantization during training and adjusts weights and quantization parameters; more effective at low bit-widths

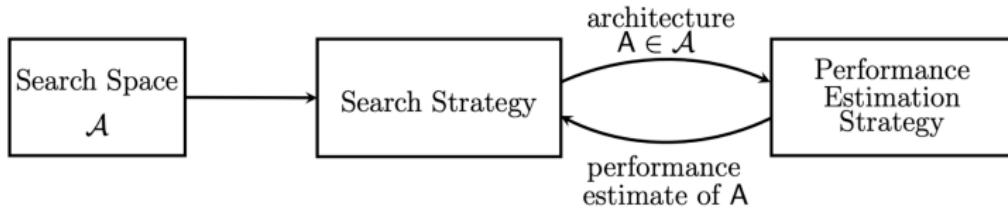


Additional considerations

- **Symmetric** ($Z = 0$, used for weights) vs. **asymmetric** (used for activations)
- **Per-tensor** (same S and Z for entire layer) vs. **per-channel**

Model compression: neural architecture search (1)

Neural Architecture Search (NAS) automates neural network design by searching for architectures that maximize an objective [10, 11]

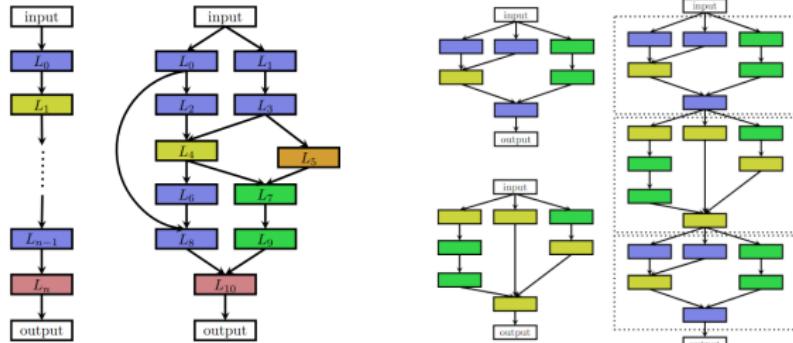


- Can address **multiple objectives** simultaneously (e.g., accuracy, latency, size, memory usage, etc.)
- Can include **hardware-aware** constraints or even **integrate HW platform design** as part of the search

Model compression: neural architecture search (2)

Search space

- Set of all possible architectures that can be discovered
- Defined by a set of **architectural choices** (number of layers, number of units per layer, type of operations, node connections)
- Trade-off between human insight, granularity, size of space, and computational budget



Model compression: neural architecture search (3)

Search strategies

- **Random search:** sample architectures randomly
- **Reinforcement learning:** controller network generates architecture based on *reward* signal
- **Genetic algorithms:** evolve populations of architectures through selection, crossover, and mutation
- **Bayesian optimization:** build *surrogate model* of performance; use *acquisition function* to pick next architecture

Performance estimation

- Problem: training and evaluating each model is very expensive
- Efficient alternatives: learning curve extrapolation, zero-cost proxies, dataset distillation, weight inheritance

Software

Deployment tools

How do we go from compressed model to executable binary?

- **Problem:** different HW targets (MCUs, SoCs, FPGAs) require different representations and compilation flows
- **“Solution”:** a few **model exchange formats** (ONNX, TFLite, NNEF) and many **application-specific tools** with different focuses

Overview of tools

Inference engines

- Convert model into internal representation (offline) and execute it using a runtime that's integrated into the firmware
- Most popular approach, with many runtimes available
- Examples: TFLite/TFLM, ExecuTorch, ONNX Runtime, Tract

Compilers and code generators

- Generate optimized, target-specific machine code or native code (e.g., C/C++, Rust) for direct execution
- Examples: TVM, XLA, IREE/MLIR, uTensor, onnx2c

Most tools are a mix of both approaches

Vendor-specific tools

Tool	Vendor	Target	Description
STM32Cube.AI	STMicro	MCU	Converter + code generator + runtime
eIQ Toolkit	NXP	MCU / MPU	Converter + runtime + quant.
MPLAB ML Suite	Microchip	MCU	End-to-end solution
Reality AI Tools	Renesas	MCU	End-to-end solution
Vela	Arm	NPU	Compiler + partitioner + quant.
ESP-NN	Espressif	MCU	Optimized kernels for TFLM
TensorRT	NVIDIA	MPU / GPU	Compiler + runtime
OpenVINO	Intel	CPU / NPU	Converter + optimizer + runtime + quant.
Vitis AI	AMD / Xilinx	FPGA / SoC	Quantizer + compiler + runtime
TIDL / Edge AI SDK	Texas Instruments	DSP / SoC	Converter + compiler + runtime
RKNN Toolkit	Rockchip	NPU / SoC	Converter + quantizer + runtime
DRP-AI TVM	Renesas	NPU / SoC	Compiler + runtime
SNPE / QNN SDK	Qualcomm	SoC	Converter + runtime + quant.
NeuroPilot	MediaTek	SoC	Converter + compiler + runtime
Edge TPU Compiler	Google	TPU	Compiler + runtime
HailoRT	Hailo	NPU	Compiler + runtime
Akida Toolsuite	BrainChip	SoC / NPU	Converter + quant. + runtime
GAP Flow	GreenWaves	DSP / SoC	Compiler + runtime + auto-tuning
Edge Impulse	Multi-vendor	Multi-target	End-to-end solution (web UI)

...plus many more smaller players

Software

Latest and niche topics

Continual learning enables models to adapt to new data after deployment [12]

Motivation

- In real-world, data changes over time (*concept drift*)
- Re-training (in the cloud) and re-deploying might be infeasible for many embedded or privacy-sensitive applications

Approaches

- **Progressive networks:** allocate new parameters for new tasks
- **Regularization:** penalize large weight changes
- **Memory-based:** store or generate samples from previous tasks

Federated learning

Federated learning enables distributed training without sharing raw data [13]

Advantages

- Improved **privacy** and **data ownership**
- **Reduced bandwidth usage** compared to raw data transmission
- Enables **personalization** and cross-device **collaboration**

Principle

- Clients (devices) train locally on their own data
- Only model updates or gradients are sent to the central server
- Server aggregates updates and broadcasts new global model

Dynamic inference

Dynamic neural networks can adapt their structures or parameters to the input during inference [14]

Downsides of static neural networks

- Fixed amount of computation regardless of difficulty
- Generalizing across all input conditions requires large models
- Fixed trade-off between performance and efficiency

Some examples of DynNNs

- **Early-exiting**: control how many layers are used
- **Mixture-of-Experts**: switch between different sub-networks
- **Slimming**: decide how many neurons or channels are used

Wetware II

Responsible embedded ML

Responsible embedded ML



How can we do it right?

No idea, but here's an example
of how it's been done **wrong**

Flock Safety

- Camera-based automated license plate readers (ALPR)
- On-device **licence plate recognition and vehicle fingerprinting**
- 90 000 devices across 6000 clients (including 5000 police depts)
- Claim: **10 % of US crimes solved** using Flock (700 000 cases/year)
- \$300 million sales in 2024, \$7.5 billion valuation [15]



Embedded ML success story?

Some concerning aspects [16]

- **Privacy:** no possibility to opt-out, data often shared with third parties
- **Data protection:** reports of hacked devices and leaked credentials
- **Fairness:** disproportionate deployment in marginalised communities
- **Transparency:** no publicly available data on accuracy or error rates
- **Accountability:** lack of oversight on how data is used
- **Public engagement:** deployed without consulting local community
- **Reporting:** official impact studies and figures widely disputed
- **Compliance:** devices installed without permit; long-term surveillance without warrant may constitute violation of Fourth Amendment



NEWS ELECTIONS LISTEN WATCH ARTS & CULTURE EDUCATION SCHEDULES SUPPORT ABOUT VPM OUR IMPACT MY ACCOUNT

Virginia surveillance network tapped thousands of times for immigration cases

WHRO | By Kaitlin Falay | VCU
Published October 6, 2020 at 12:03 PM EDT

A thumbnail image for a news story about a Virginia surveillance network being used for immigration cases. It shows a street sign for Main Street.

Additional resources

Online courses

- EdX course on TinyML from Harvard (self-paced, paid): [🔗](#)
- Course on TinyML from MIT (lecture recordings, free): [🔗](#)
- Course on Embedded ML from Edge Impulse (self-paced, free enrollment): [🔗](#)
- MEAD course on Embedded AI (Feb 2026, paid): [🔗](#)
- TinyML book (free preview, video tutorials): [🔗](#)
- Course on Deep Learning from Stanford (lecture recordings, free, iconic): [🔗](#)

Conferences and other events

- Edge AI/TinyML Summit: [🔗](#)
- Edge AI workshops at CVPR: [🔗](#) [🔗](#)
- ACACES Summer School: [🔗](#)

Curated lists

- Edge AI: [🔗](#)
- Awesome TinyML: [🔗](#)

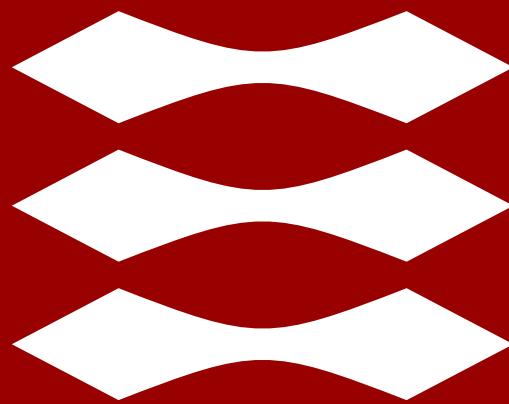
References (1)

- [1] R. Sutton, "The bitter lesson," Incomplete Ideas, [Online]. Available: <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>
- [2] Y. Bahri, E. Dyer, J. Kaplan, J. Lee, and U. Sharma, "Explaining neural scaling laws," *Proceedings of the National Academy of Sciences*, Jul. 2024, DOI: [10.1073/pnas.2311878121](https://doi.org/10.1073/pnas.2311878121)
- [3] J. Bier, "AI and vision at the edge," EE Times, [Online]. Available: <https://www.eetimes.com/ai-and-vision-at-the-edge/>
- [4] D. Networks, "Silvanet - AI wildfire detection in minutes," Dryad, [Online]. Available: <https://www.dryad.net/silvanet>
- [5] VivaCity, "Smart signal control solution," VivaCity, [Online]. Available: <https://vivacitylabs.com/products/smart-signal-control/>
- [6] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Dec. 2024, DOI: [10.1109/TPAMI.2024.3447085](https://doi.org/10.1109/TPAMI.2024.3447085)
- [7] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, *Pruning and quantization for deep neural network acceleration: A survey*, Jun. 2021, arXiv: [2101.09671\[cs\]](https://arxiv.org/abs/2101.09671).
- [8] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, *A survey of quantization methods for efficient neural network inference*, Jun. 2021, arXiv: [2103.13630\[cs\]](https://arxiv.org/abs/2103.13630).
- [9] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, *A white paper on neural network quantization*, Jun. 2021, arXiv: [2106.08295\[cs\]](https://arxiv.org/abs/2106.08295).

References (2)

- [10] C. White et al., *Neural architecture search: Insights from 1000 papers*, Jan. 2023, DOI: [10.48550/arXiv.2301.08727](https://doi.org/10.48550/arXiv.2301.08727)
- [11] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, 2019. [Online]. Available: <http://jmlr.org/papers/v20/18-598.html>
- [12] M. D. Lange et al., "A continual learning survey: Defying forgetting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021, DOI: [10.1109/TPAMI.2021.3057446](https://doi.org/10.1109/TPAMI.2021.3057446)
- [13] E. Gabrielli, G. Pica, and G. Tolomei, *A survey on decentralized federated learning*, Aug. 2023, DOI: [10.48550/arXiv.2308.04604](https://doi.org/10.48550/arXiv.2308.04604)
- [14] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Nov. 2022, DOI: [10.1109/tpami.2021.3117837](https://doi.org/10.1109/tpami.2021.3117837)
- [15] T. Brewster, "AI startup flock thinks it can eliminate all crime in america," Forbes, [Online]. Available: <https://www.forbes.com/sites/thomasbrewster/2025/09/03/ai-startup-flock-thinks-it-can-eliminate-all-crime-in-america/>
- [16] L. Daniel, "Privacy violated, warrantless surveillance alleges flock safety camera lawsuit," Forbes, [Online]. Available: <https://www.forbes.com/sites/larsdaniel/2024/10/22/warrantless-surveillance-federal-lawsuit-challenges-flock-safety-cameras/>

DTU



Networked Embedded Systems

Week 11: Embedded System Performance

Francesca Di Mella
PhD Student

Slides created by Xenofon Fafoutis.

What Makes an Embedded System Good?



Performance Objectives

- Speed
 - Short response time
 - High throughput/bandwidth
- Efficiency
 - Low utilisation of computing resources (cpu, memory)
 - Low power (green)
 - Low cost
- Dependability
 - Availability (readiness to be used, no downtimes, no dead battery)
 - Reliability (accuracy, correctness of service, no failures)
 - Maintainability (easy maintenance, repair)
 - Security (confidentiality, integrity)



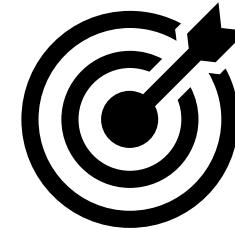
Performance Metrics

- Today we'll focus on some performance metrics
 - And how to measure them
- Reliability
 - Probability of success of an operation (higher is better)
- Delay (Latency)
 - Time it takes to complete an operation (lower is better)
- Energy Consumption
 - Energy it takes to complete an operation (lower is better)



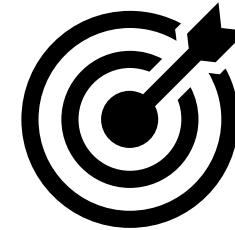
Reliability

- A measure of the probability of success over time
 - Or the opposite: failure probability
- Hardware reliability
 - Probability the hardware will fail
- Software reliability
 - Probability of software failures due to bugs
- Network reliability
 - Packets delivered successfully over total packets sent
- Accuracy
 - Prediction success rate



Measuring Reliability

- Repeat the process multiple times and count failures
 - Number between 0 and 1
- Safety-critical systems
 - Failures can lead to catastrophic damage or loss of life
 - Measures taken to keep reliability as high as possible
 - Example: 0.999 reliability is not good enough if 1 out of 1000 flights crash
 - Often expressed in orders of magnitude
 - Example: >0.99999 (five 9' of reliability) or error probability $<10^{-5}$
- Challenge
 - Measuring reliability very accurately needs a lot of samples

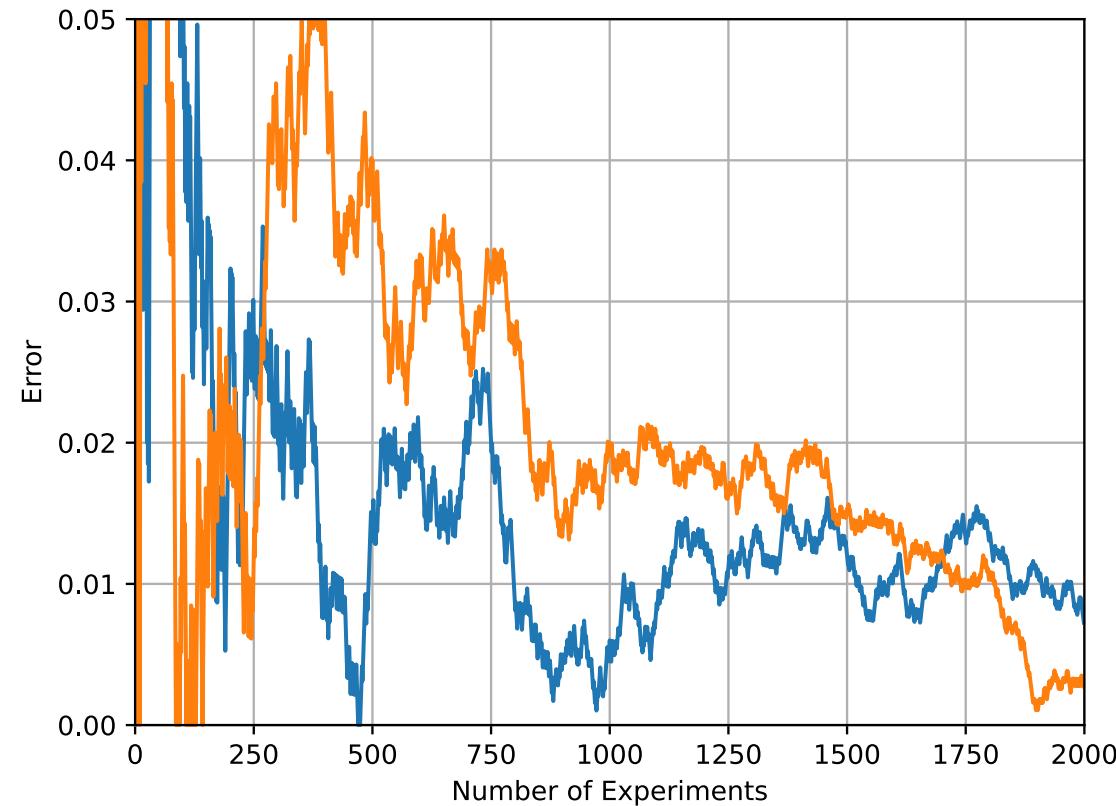


Example: Fair Coins

- A fair coin has $p_{IDEAL}=0.5$ probability to yield heads when tossed
- We have two coins, the blue coin and the orange coin
 - We expect their real probability to be fairly close to the ideal: $0.49 < p_{REAL} < 0.51$
- We want to compare which coin is more fair
- Let's design an experiment
 - We toss both coins and count how many times each yields heads so far
 - We subtract the measure probability of yielding heads from the ideal, error = $p_{REAL} - 0.5$
 - A perfect fair coin will have an error of 0
 - We repeat the experiment until we can confidently say which coin is more fair
- Question: how many times do we need to toss the coins to confidently decide which is more fair?

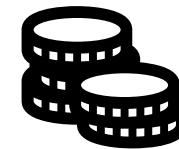
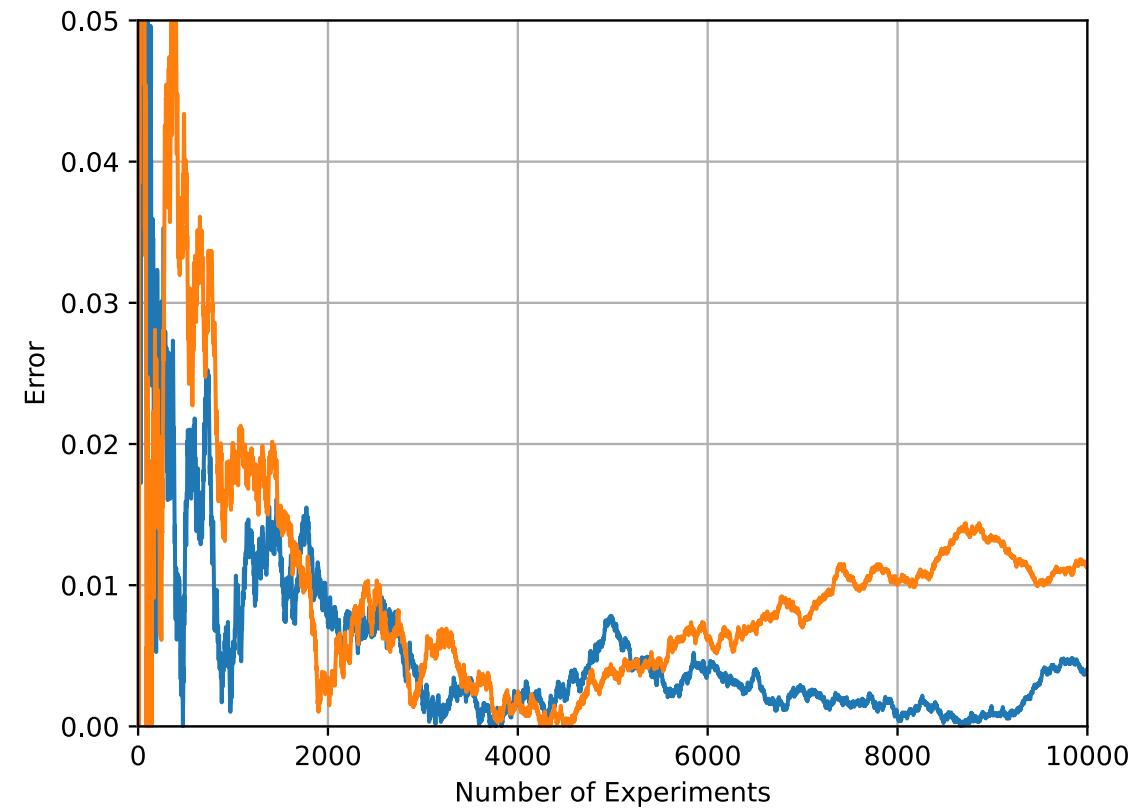


Example: Fair Coins (2000 tosses)



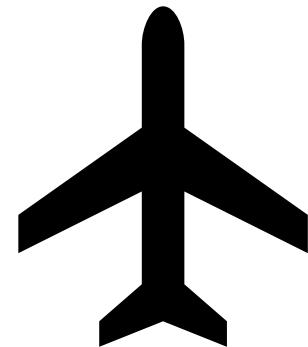
Example: Fair Coins (10000 tosses)

- $p_{\text{BLUE}} = 0.505$
- $p_{\text{ORANGE}} = 0.51$



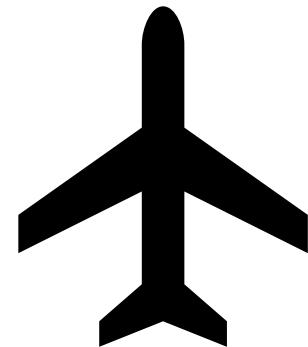
Airplane

- Why do commercial airplanes have two pilots, when one is perfectly capable to pilot?
- Why do commercial airplanes have two engines, when one is sufficient to fly?



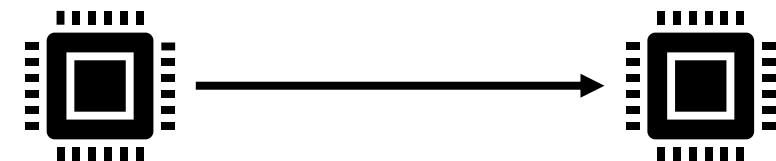
Airplane

- Why do commercial airplanes have two pilots, when one is perfectly capable to pilot?
- Why do commercial airplanes have two engines, when one is sufficient to fly?
- For safety, if one pilot becomes unconscious or an engine fails, the plane can still be landed safely



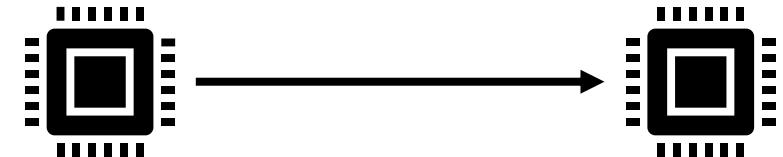
Reliability After Countermeasures

- The world is often unreliable, so we take actions to improve reliability
- The user/application experiences the reliability after all countermeasures have been applied
- Example: A wireless link has Packet Reception Rate, PRR=0.6
 - That's low, so employ a MAC protocol with ACKs and retransmissions
 - If failure, we retransmit; if we fail N times, we drop the packet
 - What is the reliability of the link after this countermeasure (Packet Delivery Rate, PDR)?



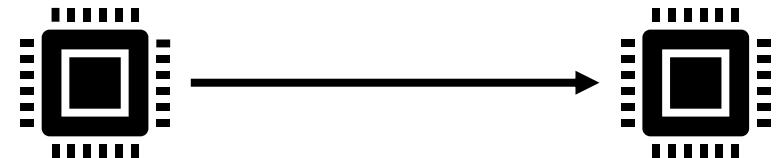
Reliability After Countermeasures

- The world is often unreliable, so we take actions to improve reliability
- The user/application experiences the reliability after all countermeasures have been applied
- Example: A wireless link has Packet Reception Rate, PRR=0.6
 - That's low, so employ a MAC protocol with ACKs and retransmissions
 - If failure, we retransmit; if we fail N times, we drop the packet
 - What is the reliability of the link after this countermeasure (Packet Delivery Rate, PDR)?
 - If N=2, I have a failure if both attempts fail, $PDR = 1 - (1-PRR)*(1-PRR) = 1 - 0.4*0.4 = 0.84$
 - If N=4, I have a failure if all 4 attempts fail, $PDR = 1 - (1-PRR)^4 = 1 - 0.4^4 = 0.9744$
 - If N=8, I have a failure if all 8 attempts fail, $PDR = 1 - (1-PRR)^8 = 1 - 0.4^8 = 0.99934$
 - Note: This logic assumes that each attempt is statistically independent; in reality, they are often not independent, for example, if there is a source of interference that affects multiple attempts



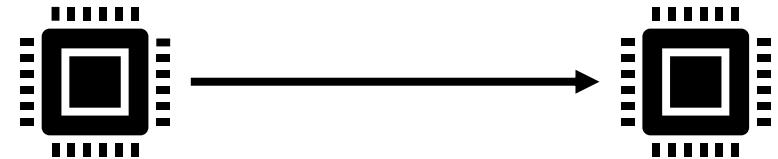
Let's compare two reliability approaches...

- A wireless link has Packet Reception Rate, PRR=0.6
- Approach #1: ACKs and retransmissions
 - If failure, we retransmit; if we fail 8 times, we drop the packet
- Approach #2: Repetition
 - We always transmit each packet 8 times back-to-back
- Assuming statistically independence, both approaches have same PDR = $1 - 0.4^8 = 0.99934$
- Which one would you choose to use?



Let's compare two reliability approaches...

- A wireless link has Packet Reception Rate, PRR=0.6
- Approach #1: ACKs and retransmissions
 - If failure, we retransmit; if we fail 8 times, we drop the packet
- Approach #2: Repetition
 - We always transmit each packet 8 times back-to-back
- Assuming statistically independence, both approaches have same PDR = $1 - 0.4^8 = 0.99934$
- Each approach has a different cost
 - Approach #1 would consume less energy and bandwidth (more efficient)
 - Approach #2 would result to less latency



Redundancy

- Intentional duplication of hardware, processes, or information to improve reliability
 - Not necessary in no-error conditions, but make the system fault-tolerant
 - Trade-off between efficiency and reliability
- Hardware redundancy (multiple instances of the same critical hardware)
- Information redundancy (add a code at a message that does not add information but helps with correcting errors)
- Time redundancy (perform action multiple times, send same data multiple times)
- Software redundancy (N-version Programming)
 - Separate teams are requested to develop functionally equivalent programs independently
 - The idea is that it becomes less likely that the different versions will have the same bugs
 - Applied to software in switching trains, flight control, electronic voting

Dissimilar Redundancy

- The problem with simply replicating processes or hardware is that often errors are correlated
 - So if an error happens, it is very likely to also happen in the replica
- Ideally, we want to add redundancy with uncorrelated failures
 - Do the same thing in multiple ways that fail differently, so one works when other fails
- Example: Use two different sensors made by different manufacturers
 - Protects us in case of failures caused by the manufacturing process
- Example: Use different types of sensors to observe the same phenomenon
 - Cameras become poor in low-light conditions
 - LIDAR becomes poor in rain/snow
 - Microphones are challenged by wind

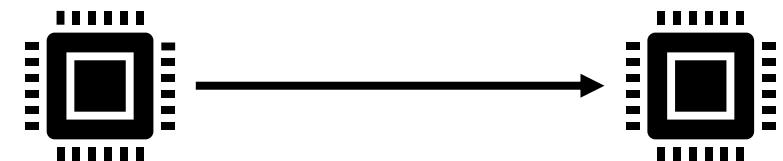
Diversity in (Wireless) Communications

- Wired and (especially) Wireless Communication are particularly prone to errors
- Diversity is essentially partial uncorrelated redundancy
- Time diversity (multiple versions of signal transmitted at different times, error correction codes)
- Frequency diversity (signal transmitted at different channels, channel hopping)
- Space diversity (signal send via different links, different wires, or different antennas)
- Path diversity (message send via different paths/routes, different intermediate hops)
- Diversity in wireless technology (e.g. message sent via WiFi and Cellular)



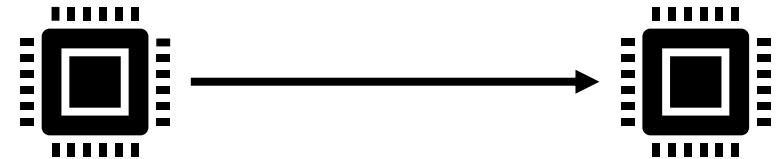
Delay (Latency)

- Response Time
 - Time between the input and a response
- Processing Delay (execution time)
 - Time between start and end of a processing block
- Communication Delay
 - Time between sending and receiving a message
- Age of Information
 - Time elapsed since generation of information (sampling delay + system delay)



Sources of Delay

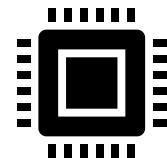
- Source
 - Transferring of data within the embedded system
 - Operating System delays (e.g. scheduling, interrupts)
 - Application related data processing
 - Message preparation (e.g. encryption, headers, routing)
 - Queuing delays
- Network
 - Transmission delay (time required to put bits on medium)
 - Propagation delay (time required for signal to travel through medium)
 - Duty cycle related delays (wait for receiver to be awake)
 - Congestion/error recovery (retransmissions, backoff time)
 - Multi-hop transmission (all above repeated multiple times)
- Destination
 - Same as source



Measuring Processing Delay

- Approach #1: Use timers to count ticks between events, then multiple by tick duration to convert ticks to time

```
T1=TIMERVALUE;  
processing();  
T2=TIMERVALUE;  
D = ((T2 - T1) % MAXTIMERVALUE) * TICKTIME;  
printf("%f\n", D);
```



- Attention: Timer should be configured so that it is impossible to overflow more than once in a single measurement event
- Example: If timer overflows every 10 seconds and the output of my measurement is 2 seconds, how do I know if the delay is 2, 12, 22, 32, ... seconds?

Measuring Processing Delay

- Approach #2: Generate output signals and use an external measurement device (e.g. oscilloscope) to measure time difference

```
gpio_set(PIN1);  
processing();  
gpio_clear(PIN1);
```

- Tip: Oscilloscopes can be set to start measuring based on a trigger event (e.g. voltages gets above set value)

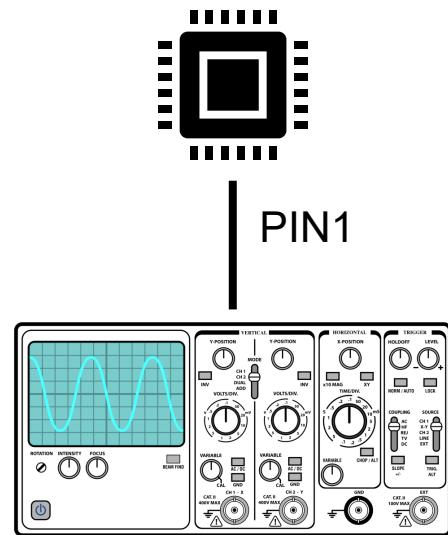
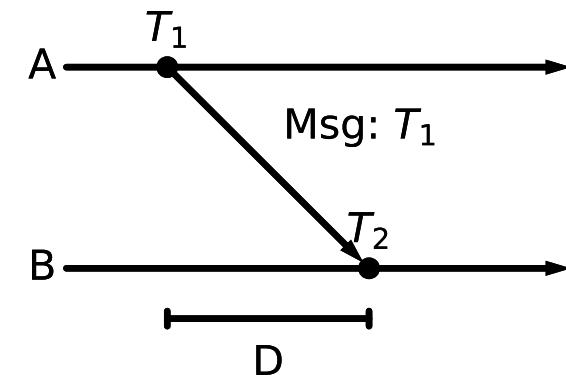
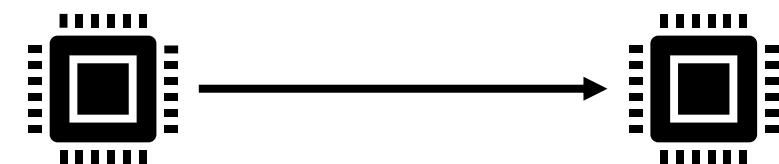


Image source: Wikipedia

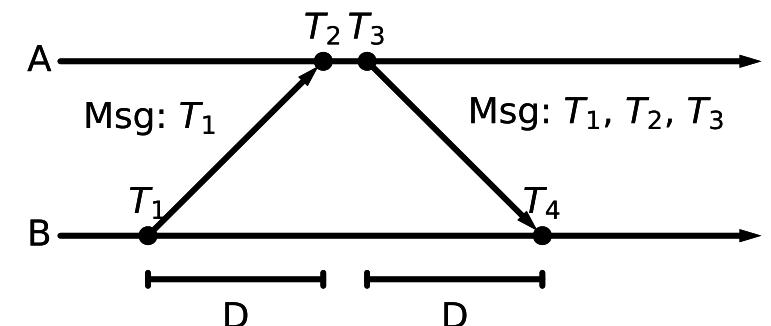
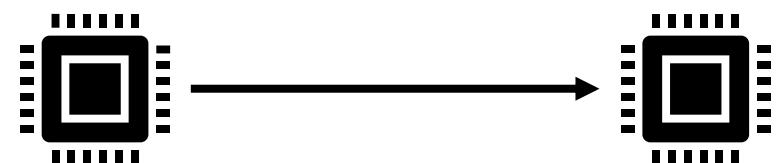
Measuring Communication Delay

- $\langle \text{delay} \rangle = \langle \text{time at destination} \rangle - \langle \text{time at source} \rangle$
 - $D = T_2 - T_1$
- Easy, right?



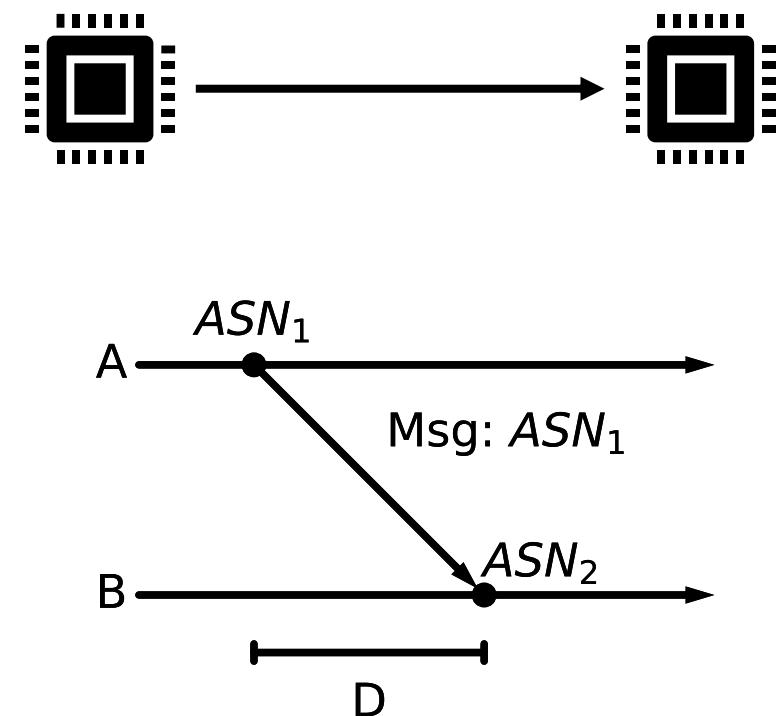
Measuring Communication Delay

- $\langle \text{delay} \rangle = \langle \text{time at destination} \rangle - \langle \text{time at source} \rangle$
- Easy, as long as source/destination are time sync'ed
- Expensive to add time synchronisation, if application does not need it
- Alternatively, we can rely on round-trip delay
 - Exchange ping-pong messages
 - Minimise processing delays between ping and pong
 - Take both timestamps at the source
 - Calculate delay as: $D = \frac{(T_4 - T_1) - (T_3 - T_2)}{2}$



Taking Advantage of Existing Time Synchronisation

- If a time synchronous protocol exists, we can take advantage of it without additional cost
- Example: TSCH Network
 - TSCH is a synchronous time-slotted protocol
 - All nodes keep a synchronised counter of timeslots
 - Absolute Slot Number (ASN)
 - Delay can be calculated as:
 - $D = (ASN_2 - ASN_1) * SLOTDURATION$
 - As accurate as TSCH synchronisation (slot is $\sim 10\text{-}15\text{ms}$)



Using External Measurement Devices

- Generate output signals on GPIOs when events occur
 - Capture these signals with an oscilloscope or logic analyser
- Disadvantage: Makes sense only when networked embedded systems are relatively close to each other

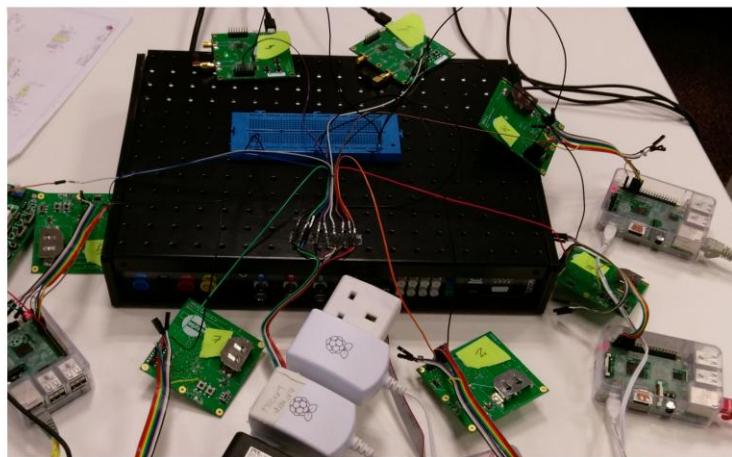


Fig. 9: Logic analyzer setup with seven nodes under test.

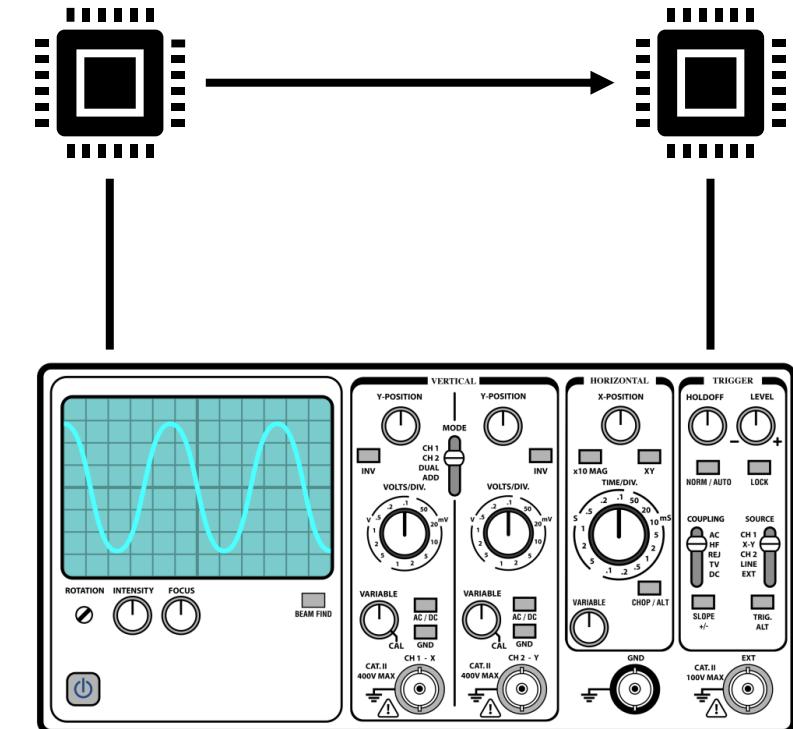


Image source: <https://doi.org/10.1109/LCN.2016.042> (left) and Wikipedia (right)

Delay Variability

- Delay is affected by random variables
 - We need to measure it multiple times
 - Report its statistical properties
- Average (mean) delay: $D_{AVG} = \frac{1}{N} \sum_{i=1}^N D_i$
- Jitter (standard deviation): $\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (D_i - D_{AVG})^2}$
- Max delay: $D_{MAX} = \max(\{D_1, D_2, \dots, D_i\})$
- Box plot (min, max, median, first/third quartile)

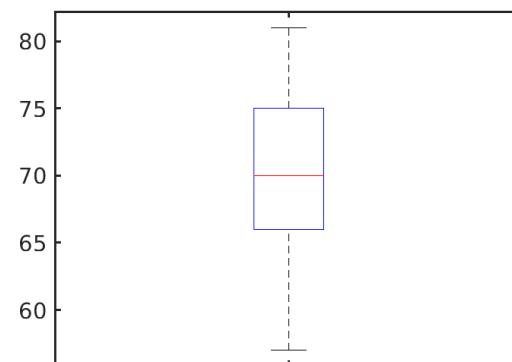
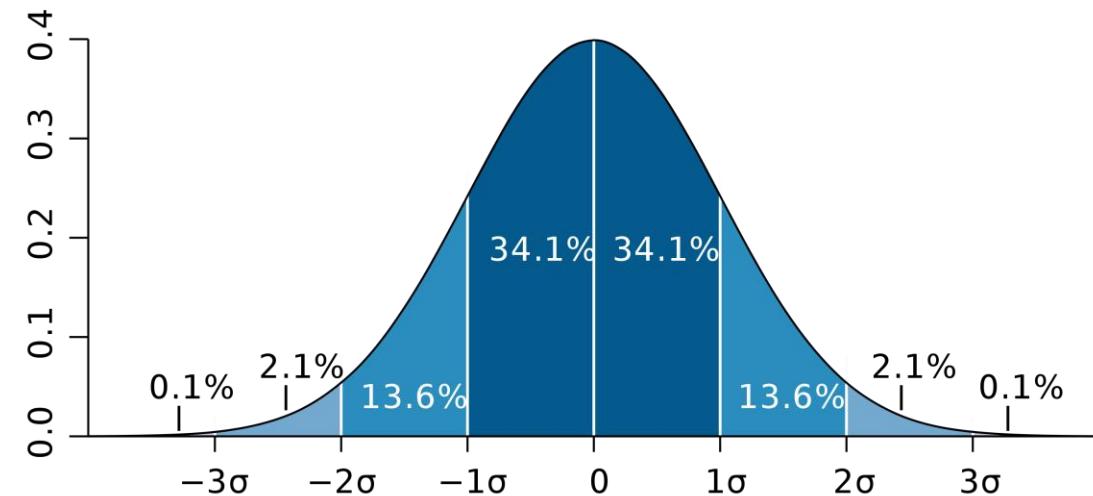


Image source: Wikipedia

Determinism (Predictability)

- Deterministic Networking (DetNet) by IETF DetNet Working Group
 - Extremely low data loss rates ($<10^{-9}$ for wired and $<10^{-5}$ for wireless links)
 - Extremely low delay variability (jitter)
 - Bounded Latency (guaranteed max delay)
- Applications
 - Audio/video streaming
 - Industrial automation
 - Vehicle control

Energy Efficiency

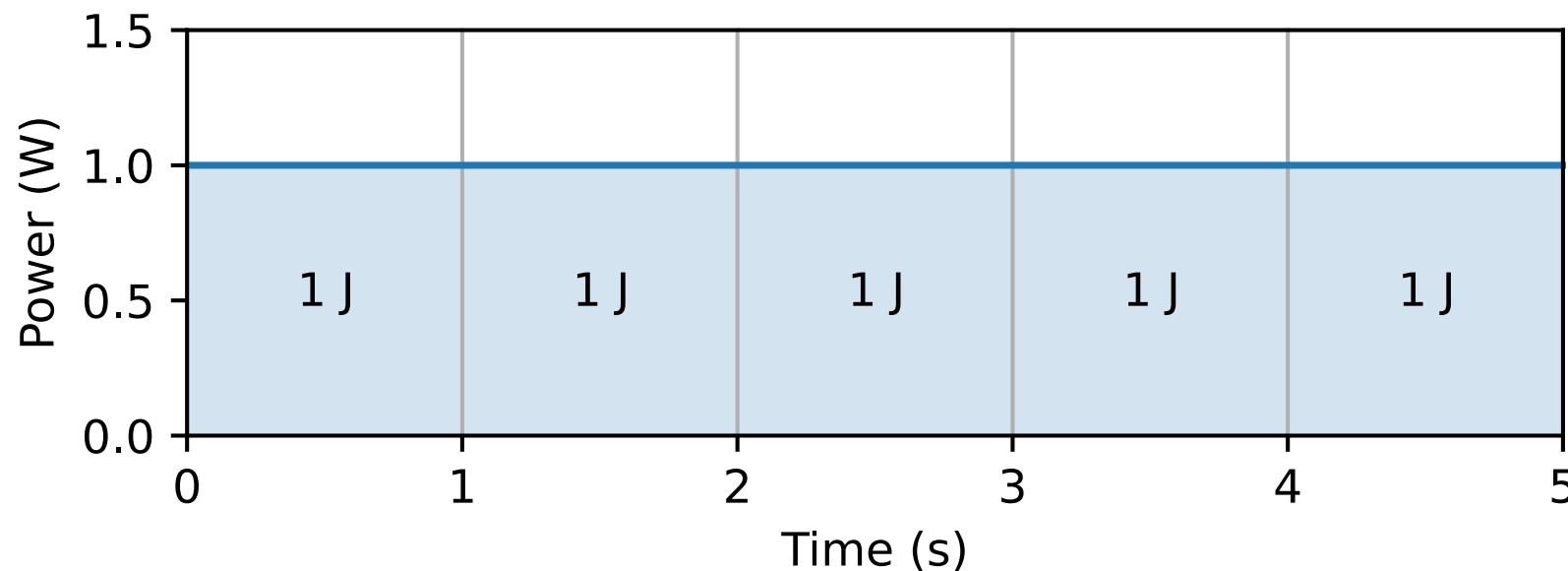
- When energy-efficiency matters?
- What affects energy consumption?

Energy Efficiency

- When energy-efficiency matters?
 - Energy-Constrained Embedded Systems (Battery-powered or energy harvesting)
 - A matter of availability (cannot operate without energy)
 - Green Embedded Systems
- What affects energy consumption?
 - Hardware (active/idle/sleep states, faults)
 - Software (duty cycling)

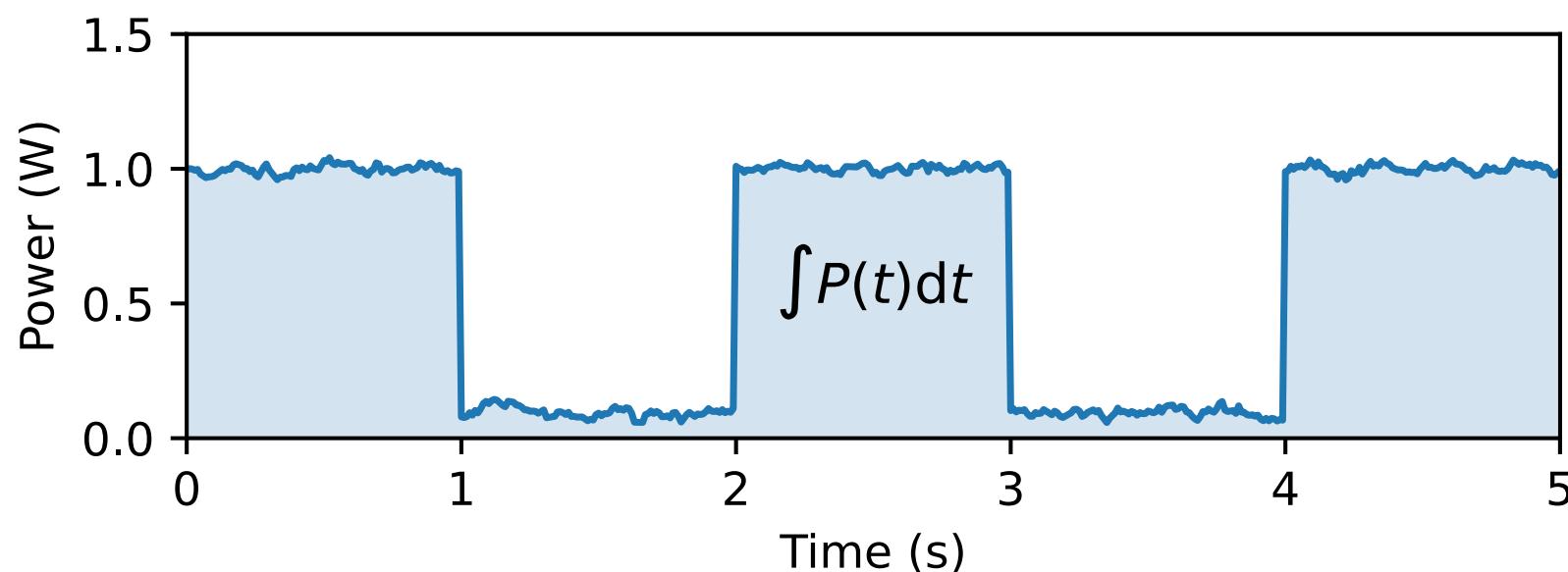
Energy Consumption Basics

- Power is the rate of energy consumption
 - $E = P \times T$ (1 Joule = 1 Watt x 1 second)



Variable Consumption

- Energy consumption depends on the state over system
 - E.g. A LED that blinks every second
- Energy is the time-integral of power



Quiz

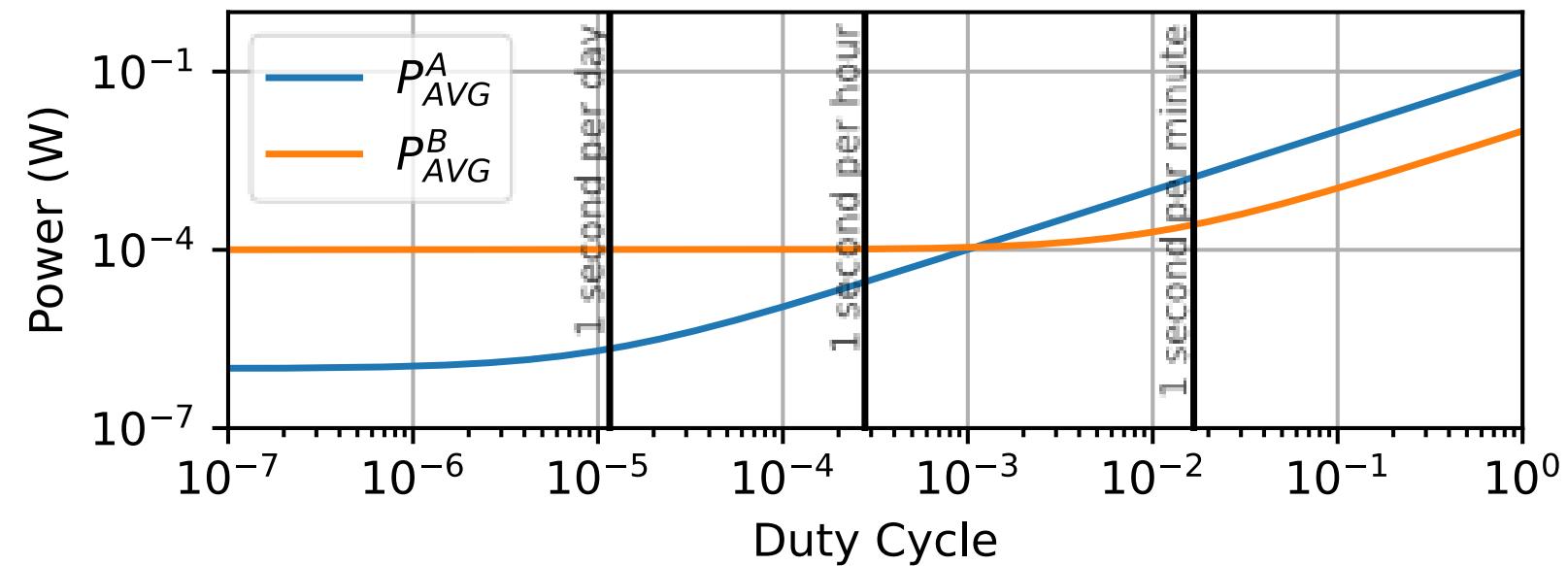
- Embedded System A and Embedded System B perform the same functionality
- System A consumes 1 uW in idle mode and 100 mW in active mode
- System B consumes 100 uW in idle mode and 10 mW in active mode
- Which one is more energy efficient?

Quiz

- Embedded System A and Embedded System B perform the same functionality
- System A consumes 1 uW in idle mode and 100 mW in active mode
- System B consumes 100 uW in idle mode and 10 mW in active mode
- Which one is more energy efficient?
- Depends on the duty cycle!

Long-Term Average Power Consumption

- Let DC be the duty cycle
 - 1% duty cycle means 1% of time in active mode and 99% in sleep mode
- $P_{AVG} = P_{ON} \times DC + P_{IDLE} \times (1 - DC)$
- In high DC active power consumption dominates
- In low DC idle power consumption dominates



How to Measure Energy Consumption?

- We know that:
 - $P = V \times I$ (1 Watt = 1 Volt x 1 Ampere)
 - So, $E = V \times I \times T$ (1 Joule = 1 Volt x 1 Ampere x 1 second)
- Let's assume that my voltage is constant (e.g. equal to the supply voltage of the battery)
 - $E = \int P(t)dt = V \int I(t)dt$
- I have to measure the current draw over time
 - I need a resistor and an oscilloscope

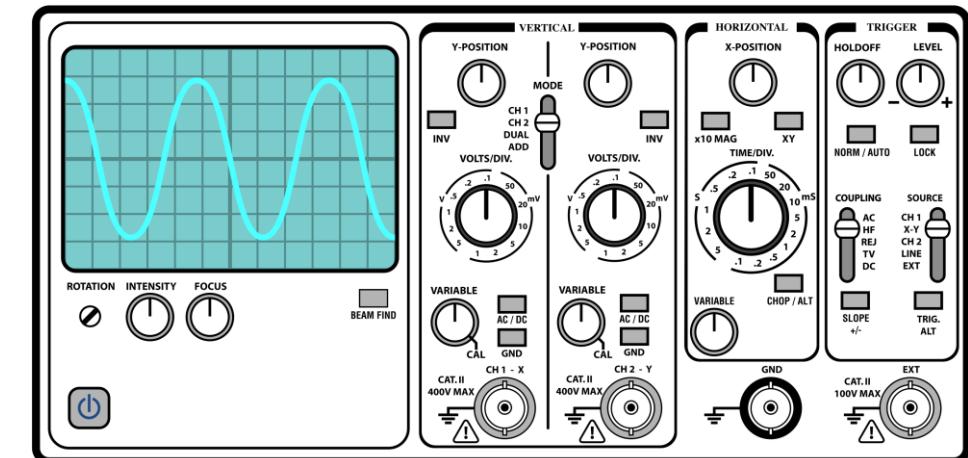
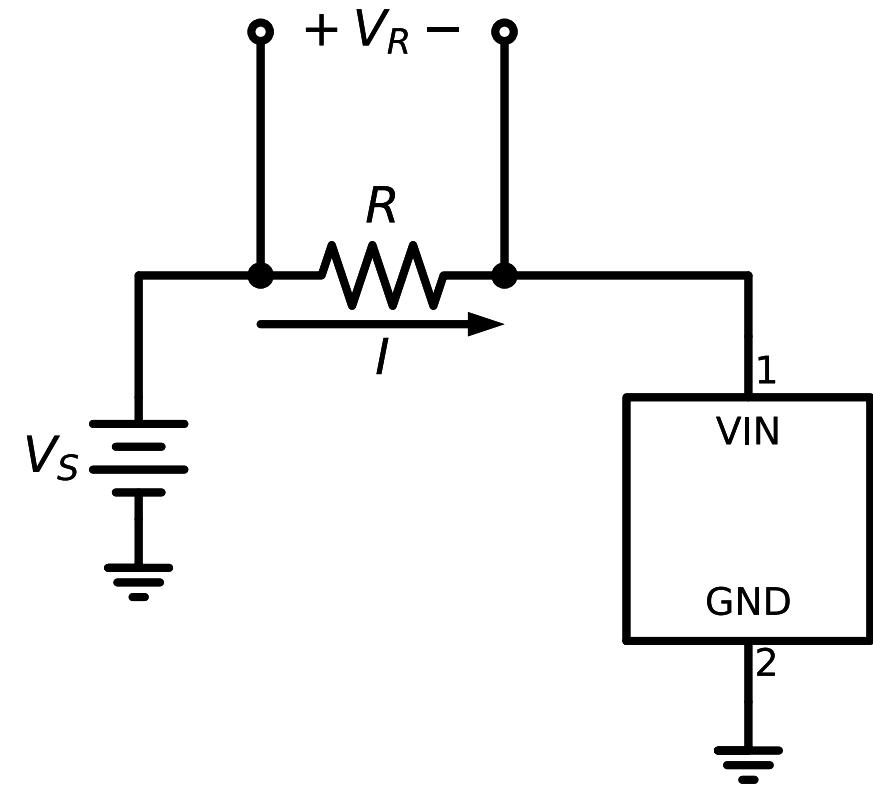


Image source: Wikipedia

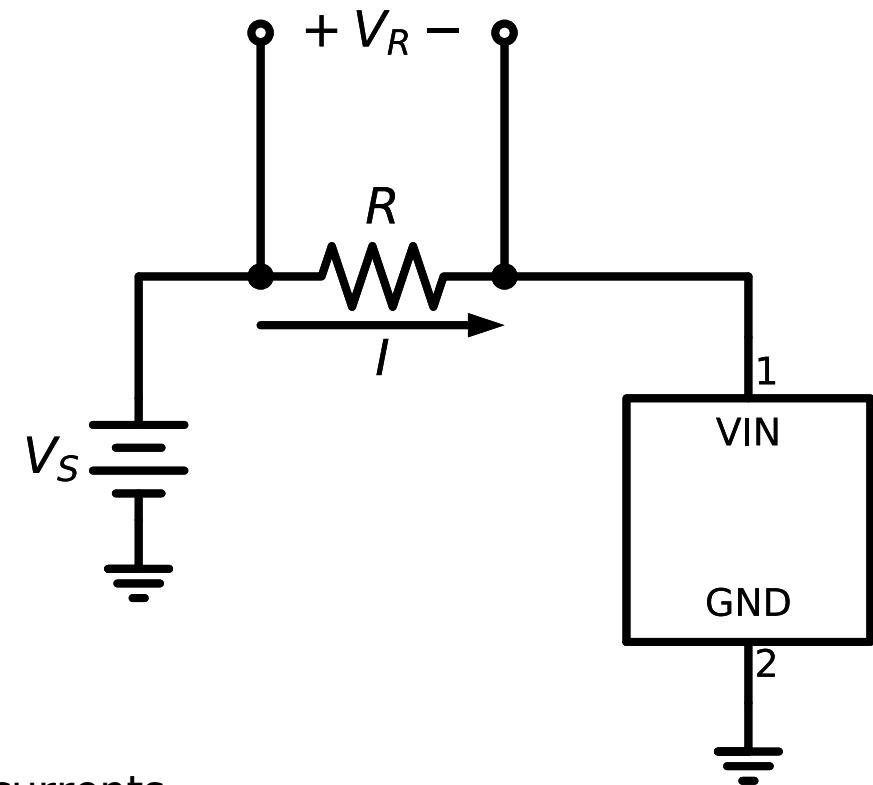
High-Side Shunt Resistor

- Measure the voltage drop across a series resistor
- $I(t) = \frac{V_R(t)}{R}$
- How large should R be?



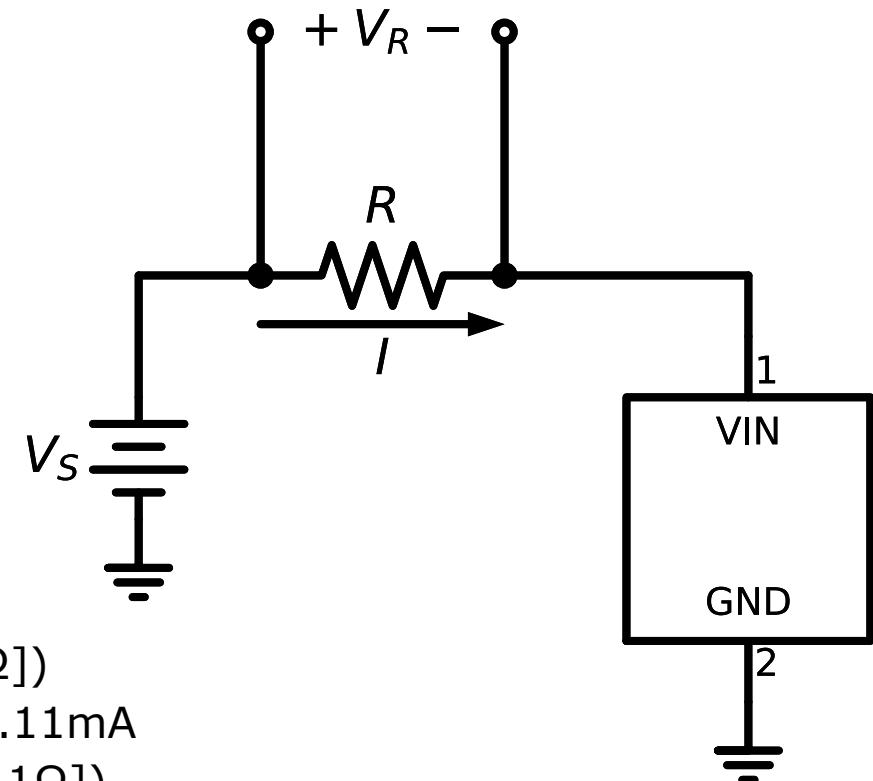
High-Side Shunt Resistor

- Measure the voltage drop across a series resistor
- $I(t) = \frac{V_R(t)}{R}$
- How large should R be?
- Example
 - $V_S=3.3V$ and IC needs $V_{IN}=3-5 V$
 - IC draws 10 uA in idle and 10 mA in active mode
- If too large ($R=100\Omega$) it can disrupt the circuit
 - Max voltage drop: $10mA \times 100\Omega = 1V$, so $V_{IN}=2.3V$
- If too small ($R=1\Omega$) it can be difficult to measure small currents
 - Max voltage drop: $10mA \times 1\Omega = 10mV$, so $V_{IN}=3.29V$ (OK)
 - Min voltage drop: $10uA \times 1\Omega = 10uV$



Resistor Tolerance

- Measure the voltage drop across a series resistor
- $I(t) = \frac{V_R(t)}{R}$
- Tolerance specifies the maximum difference between the nominal and actual resistance
- Example
 - I measure a voltage drop of $V_R=10\text{mV}$
- If I used a 10Ω resistor with 10% tolerance ($R=[9\Omega, 11\Omega]$)
 - The current is between $10/11=0.91\text{mA}$ and $10/9=1.11\text{mA}$
- If I used a 10Ω resistor with 1% tolerance ($R=[9.9\Omega, 10.1\Omega]$)
 - The current is between $10/10.1=0.99\text{mA}$ and $10/9.9=1.01\text{mA}$
- I shall use a low tolerance resistor or measure its actual resistance



Calculating the Energy

- The oscilloscope does analogue-to-digital conversion with frequency $f = 1/\tau$
- The energy is estimated as a sum of rectangles
- $E = V_S \int_{t_0}^{t_1} I(t)dt = V_S \sum_{i=0}^n \frac{V_R^i}{R} \tau$

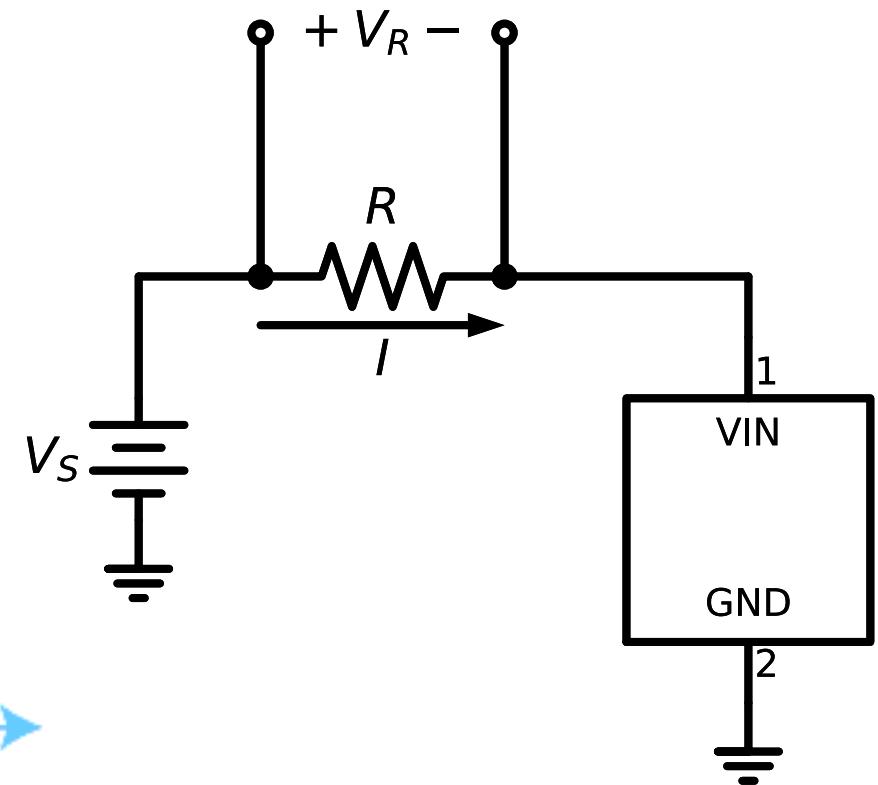
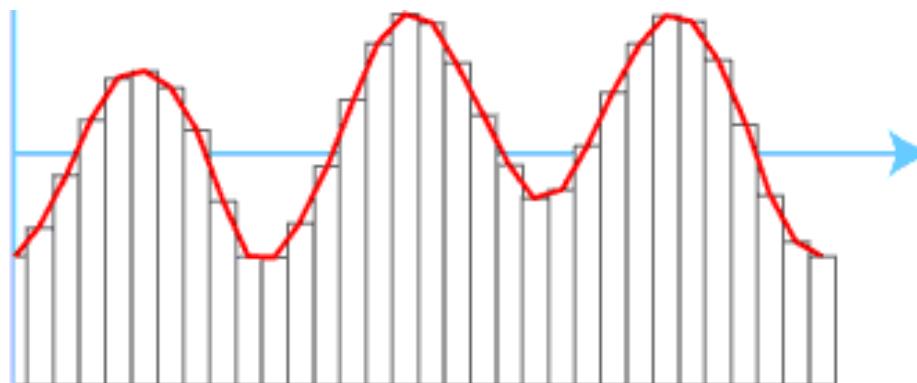
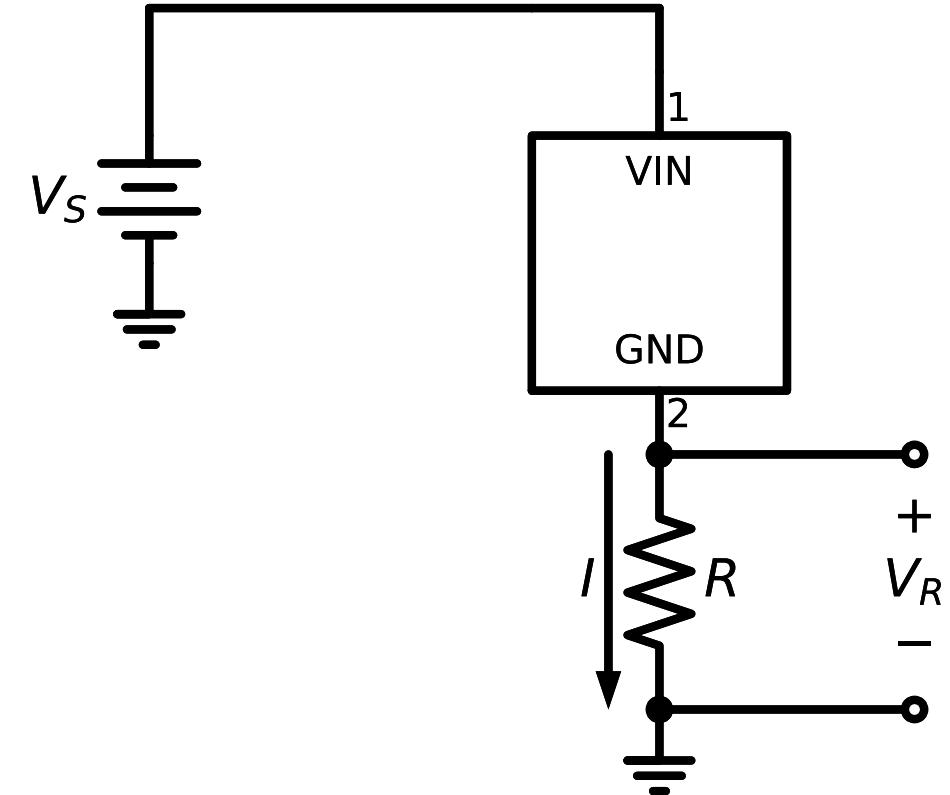


Image source: Wikipedia

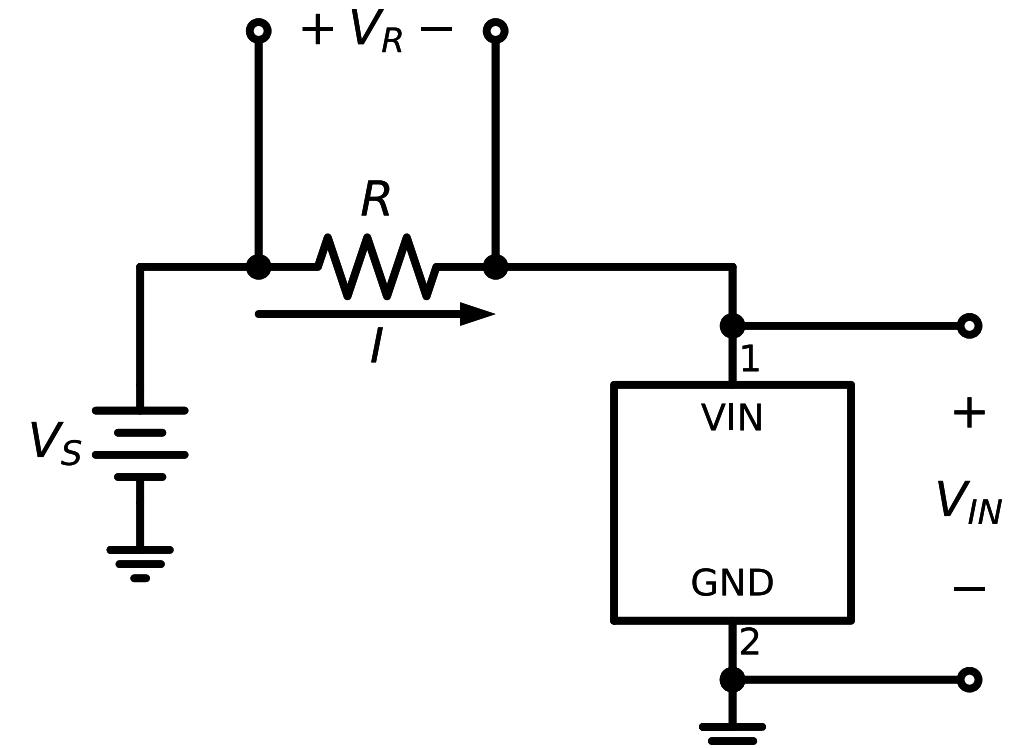
Low-Side Shunt Resistor

- A single-ended probe measures voltage between a point and ground
- The high-side shunt assumes that:
 - The measuring instrument does not have a common ground with measured system
 - We perform two single-ended measurements, and we subtract them
 - We use differential probes
- A low-side measurement is equivalent and possible with one single-ended measurement



Measuring the Supply Voltage

- It is common and often accurate enough to consider the supply voltage constant
- In reality this is not the case
 - Internal resistance that builds inside the battery
 - Noise from AC to DC rectification
 - Voltage drop of the shunt resistor itself
- For more accurate results I need to measure V_{IN} too
 - Measurements must be synchronised
- $E = \sum_{i=0}^n V_{IN}^i \frac{V_R^i}{R} \tau$



Power Measurement Instruments

- Higher-end equipment offer
 - More precise measurements
 - Higher dynamic range
- Power Analysers
 - Simultaneous voltage/current measurement
 - Automatic current range switching
- Source and Measurement Unit (SMU)
 - Provide power and measure it
 - Cleaner wiring
- Current Probes
 - Non-invasive, clamp around a wire
 - Measure magnetic field



Image source: N6705C DC Power Analyzer (top) and 1146B AC / DC Current Probe (bottom) by Keysight Technologies

Power Measurements in the wild

- Power measurement instruments are great for measuring one embedded system in the lab
 - Too expensive to scale
 - Too bulky and invasive to deploy in the wild
 - Require a lot of energy and storage for data
- How can I measure a distributed embedded system of 100 devices?
- How can I measure an embedded system deployed on the public streets?
- How can I measure an embedded system that moves like a smart vehicle?
- How can I measure an embedded system for 1 year?

Software-Based Estimation

- A less accurate, yet scalable way to measure energy consumption
 - Assumes constant supply voltage
 - Assumes the MCU goes through states where each state has constant current draw
- Procedure
 - MCU uses its clocks to measure how many ticks it spends in each state
 - Periodically it logs these measurements
 - Same can be applied to peripherals whose state the MCU controls

Period	MCU Active	MCU Idle	MCU Standby	Radio TX	Radio RX
1966080	32768	294912	1638400	540	10233
1966080	31232	296405	1638443	654	10323
1966080	34440	293106	1638534	434	11030

Software-Based Estimation

- The supply voltage (V_S) and current drawn in each state (I_i) can be measured or provided in the datasheet
- The energy consumed in each reporting period k is the weighted sum of each state
- Time spent in each state in ticks (t_i) is divided by ticks in a second (t_s) to be converted in time
- $$E_k = \sum_{i=0}^n \frac{t_i}{t_s} I_i V_S$$
- The average power consumption is E_k over the duration of reporting period

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{core}	Core current consumption	Reset. RESET_N pin asserted or VDDS below Power-on-Reset threshold	100			nA
		Shutdown. No clocks running, no retention	150			
		Standby. With RTC, CPU, RAM and (partial) register retention. RCOSC_LF	1			
		Standby. With RTC, CPU, RAM and (partial) register retention. XOSC_LF	1.2			
		Standby. With Cache, RTC, CPU, RAM and (partial) register retention. RCOSC_LF	2.5			
		Standby. With Cache, RTC, CPU, RAM and (partial) register retention. XOSC_LF	2.7			
		Idle. Supply Systems and RAM powered.	550			
		Active. Core running CoreMark	1.45 mA + 31 μ A/MHz			
		Radio RX ⁽¹⁾	5.9			
		Radio RX ⁽²⁾	6.1			
		Radio TX, 0-dBm output power ⁽¹⁾	6.1			
		Radio TX, 5-dBm output power ⁽²⁾	9.1			
						mA

Example: Software-Based Estimation vs Real Measurements

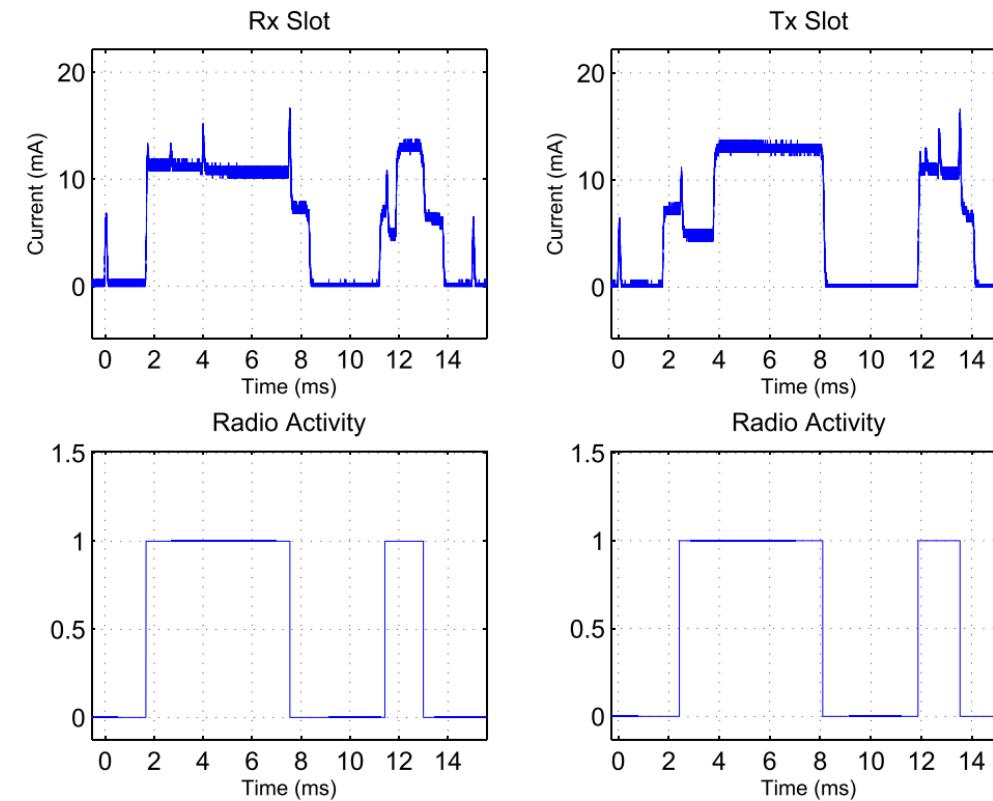
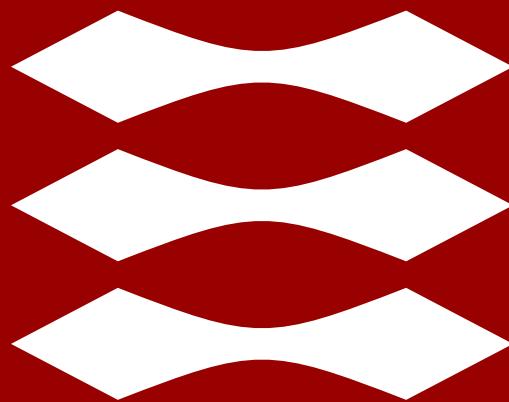


Image source: <https://doi.org/10.1109/JSEN.2013.2285411>

DTU



Networked Embedded Systems

Week 12: Industrial Embedded Networks

Xenofon (Fontas) Fafoutis

Professor

xefa@dtu.dk

www.compute.dtu.dk/~xefa

Requirements of (Industrial) Networked Embedded Systems

- Scalability
 - Supported number of embedded systems
- Throughput
 - Bytes received per second
- Reliability (Packet Delivery Ratio, PDR)
 - Number of packets received over packets sent
- Latency/Delay
 - Time interval from the time a packet is sent to the time packet received
- Jitter
 - Variability of latency/delay
- Time-sensitivity or determinism
 - Predictable/bounded latency and jitter
- Energy Efficiency and Consumption
 - All above need energy

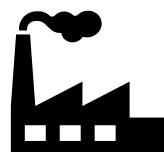
Requirements of (Industrial) Networked Embedded Systems

- Scalability
 - Supported number of embedded systems
- Throughput
 - Bytes received per second
- Reliability (Packet Delivery Ratio, PDR)
 - Number of packets received over packets sent
- Latency/Delay
 - Time interval from the time a packet is sent to the time packet received
- Jitter
 - Variability of latency/delay
- Time-sensitivity or determinism
 - Predictable/bounded latency and jitter
- Energy Efficiency and Consumption
 - All above need energy

Trade-offs: There is no solution that optimises all; we need figure out what we can afford to sacrifice

Industrial Embedded Networks

- Industrial Networks carry vital and often safety critical data
 - Tend to prioritise reliability and determinism over throughput and energy efficiency
 - Best-effort data may co-exist (e.g. vehicle sensor data vs entertainment)
- Common features
 - Schedule-based communication (avoid contention-based protocols)
 - Time synchronisation
 - Traffic prioritisation
 - Redundancy
 - Interference mitigation
- Examples
 - TSCH
 - TSN



Industrial Wireless Networks

- The 6TiSCH protocol stack
- A collection of IEEE/IETF standards
- Aiming at industrial wireless networks that require high reliability and deterministic delay/jitter
- Main components
 - RPL
 - IPv6
 - 6top
 - IEEE 802.15.4 TSCH

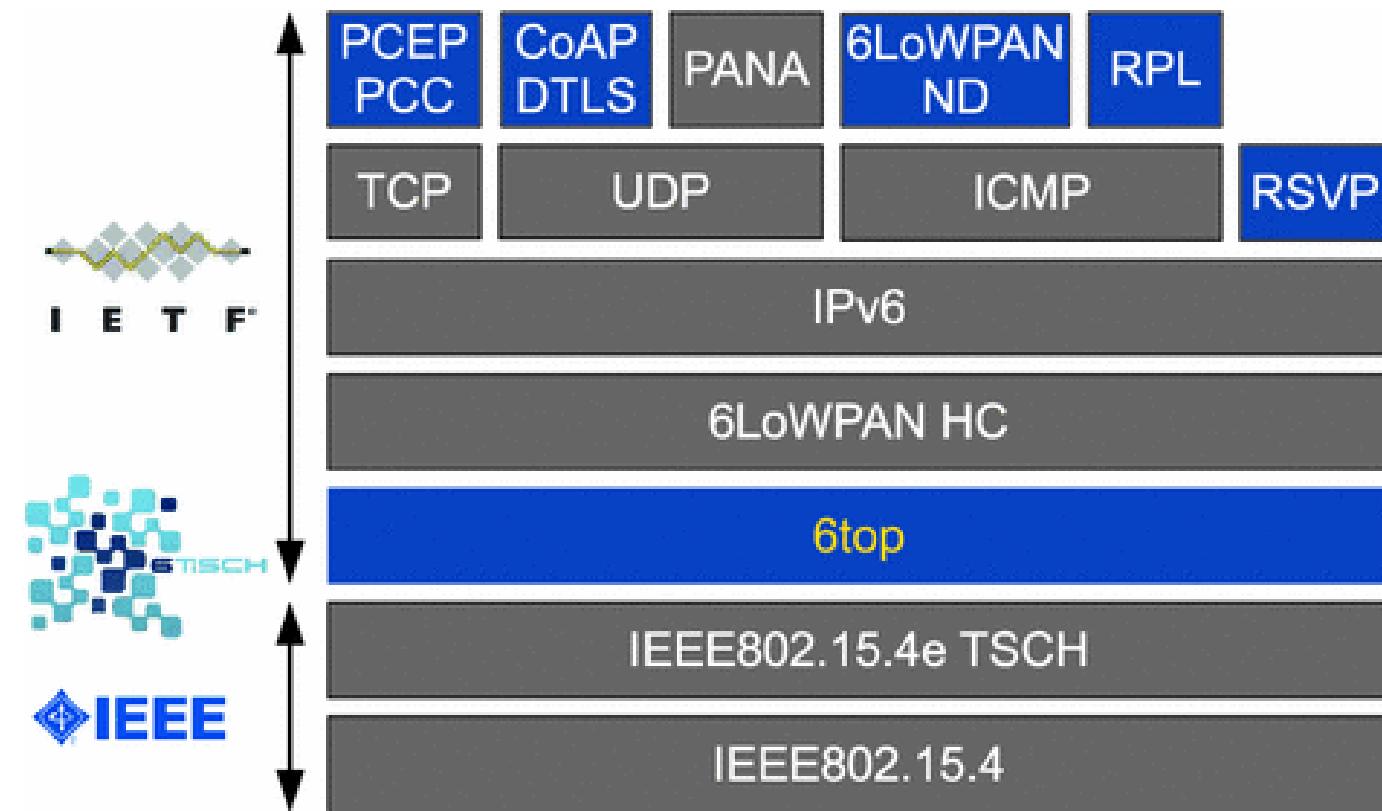


Image source: https://doi.org/10.1007/978-3-319-04223-7_5

IEEE 802.15.4 PHY

- A low-power wireless standard for Low-Rate Wireless Personal Area Networks (LR-WPANs)
 - Supports two ISM Bands (2.4 GHz, sub-GHz)
 - 250 kbps data rate @ 2.4 GHz
 - 127-byte packets
 - 16 channels @ 2.4 GHz
(+ 1 channel @ 868 MHz in Europe)
 - Supports encryption/authentication
- Used in smart home/city/* wireless networks (Thread, Zigbee)
- Used in industrial wireless networks (6TiSCH)

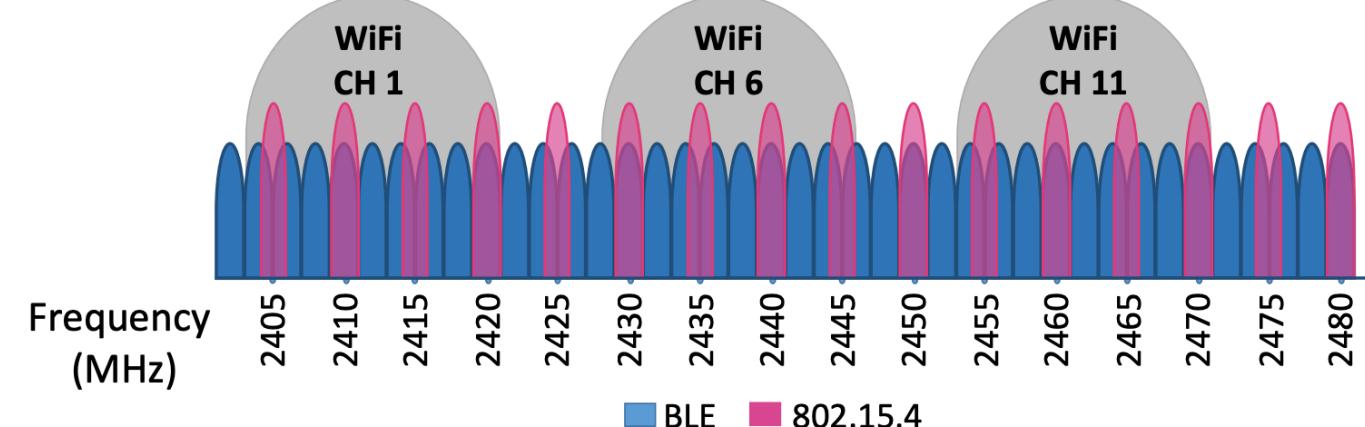


Image source: <https://doi.org/10.1109/PIMRC.2017.8292262>

IEEE 802.15.4 MAC

- Traditional IEEE 802.15.4-2011 MAC
 - Based on two classes of devices (asymmetric resources)
 - Full-Function Devices (FFD): Coordinator, frame forwarder
 - Reduced-Function Devices (RFD): Simple, energy-preserving
 - Supports contention-based (CSMA/CA) and contention-free (GTS) communication
- IEEE 802.15.4e MAC Enhancements
 - Enhanced MAC protocols for industrial applications
 - DSME (Deterministic and Synchronous Multi-channel Extension)
 - LLIN (Low Latency Deterministic Network)
 - TSCH (Time Slotted Channel Hopping)

IEEE 802.15.4 TSCH

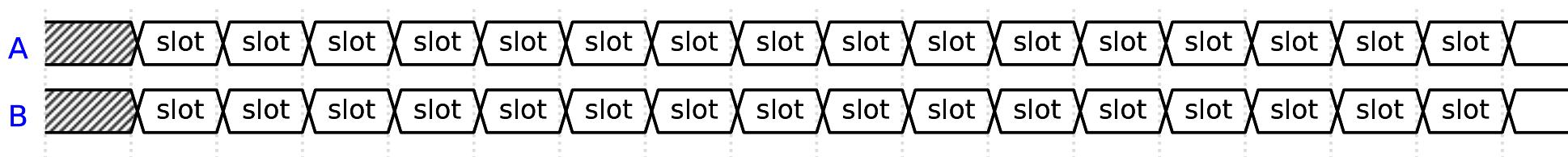
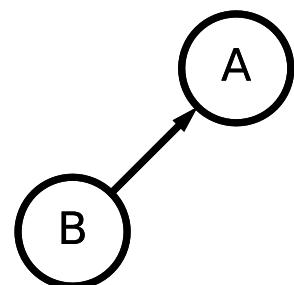
- TSCH (Time-Slotted, Channel Hopping)
 - A MAC protocol based on the IEEE 802.15.4 PHY
- TSCH supports
 - Multi-hop networks (line, tree, mesh) topologies
 - Radio duty cycling in all devices
- TSCH provides
 - Very high reliability (99.99%+ PDR)
 - Interference avoidance
 - Predictable delay/jitter/energy consumption
 - Time synchronisation (10ms)

TSCH Mechanics

- A combination of TDMA and FDMA
- Time-Slotted: Synchronous schedule-based channel access
 - Predictable channel access, possible to eliminate collisions
- Channel Hopping: Use of multiple frequency channel in a rotating fashion
 - Resilience to external interference
- TSCH traces its roots to older industrial wireless standards
 - Time Synchronized Mesh Protocol (TSMP, 2006) by Dust Networks (proprietary)
 - WirelessHART (2007) by the HART Communication Foundation
 - ISA100.11a (2009) by the International Society of Automation (ISA)

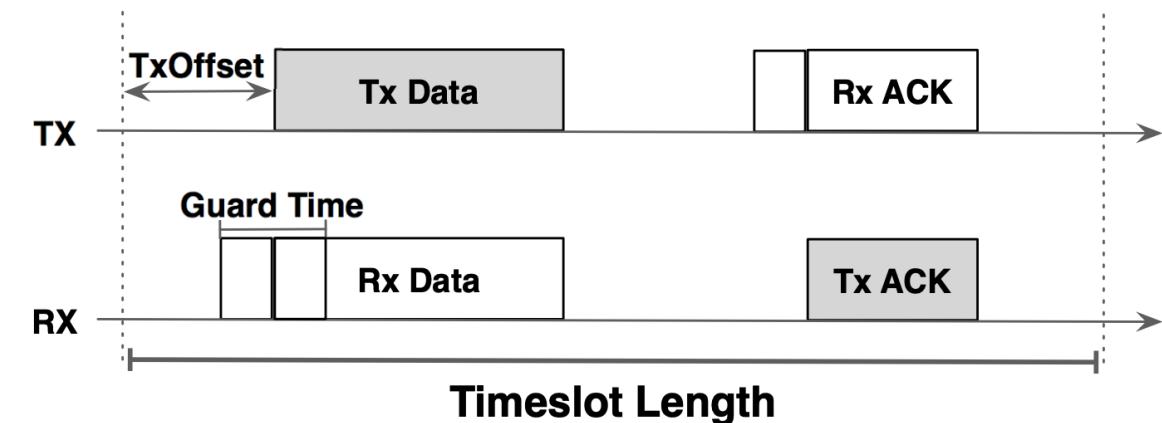
TSCH: Timeslots and Time Synchronisation

- Network-wide Time Synchronisation
 - All devices in a TSCH network are time synchronised with each other
 - Relative time synchronisation based on the implicit synchronisation technique
- Time is split into timeslots
 - Each slot is long enough to support:
 - Transmission of a frame of maximum size (~4 ms)
 - An acknowledgement (~0.35 ms)
 - The guard time (~2 ms)
 - Time required to switch from TX to RX mode, encryption/decryption
 - The duration of the timeslot is not imposed by the standard
 - Typical slot size is 10 ms
 - Some older radios are slow at switching from TX to RX mode and need 15ms



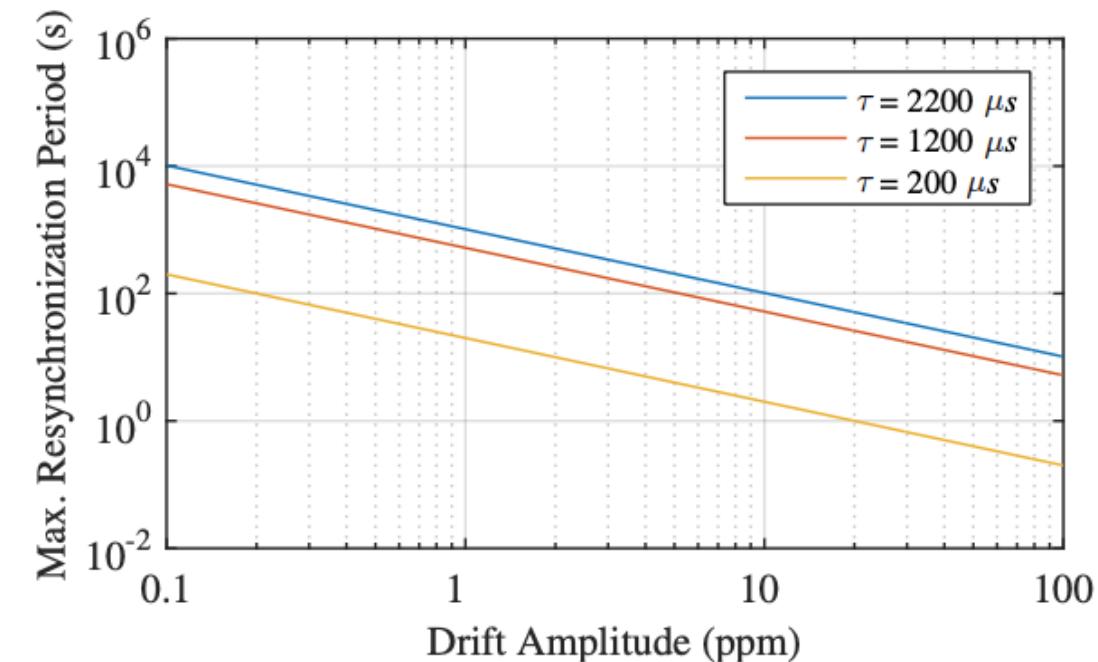
TSCH: Timeslot

- A timeslot defines a transmission opportunity
 - An actual frame transmission may not happen if there is nothing to transmit!
- The sender initiates the frame transmission at *TxOffset* after the beginning of the timeslot
- The receiver listens to the channel for the time defined by the *Guard Time*
 - The guard time is equally spaced around the expected transmission time to account for both positive and negative drift
 - If a preamble is captured within the guard time, the receiver stays on to capture the full frame
 - Otherwise, the receiver goes to sleep
- After a frame transmission, sender and receiver swap the modes of their radios for the ACK



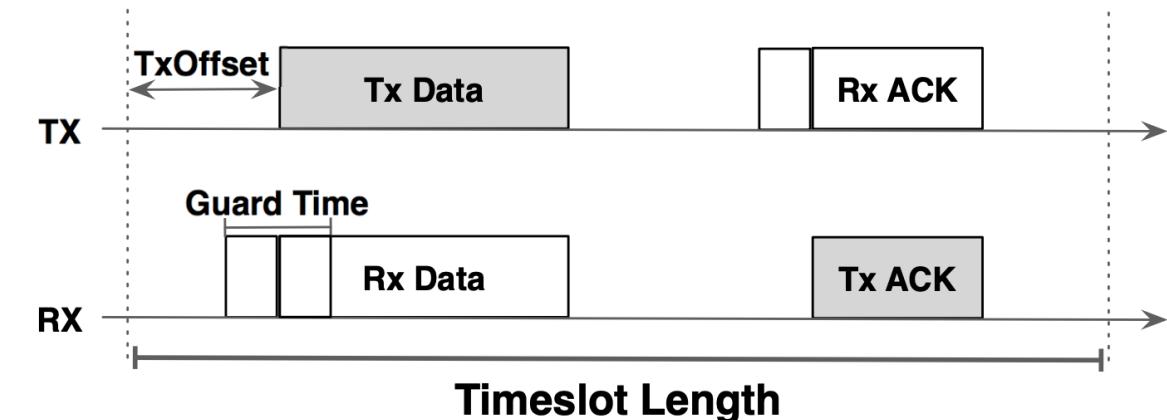
TSCH: Guard Time

- The guard time allows for a certain degree of synchronisation error
 - Periodic synchronisation is required
- Longer Guard Time
 - More robust to synchronisation errors
 - Less frequent synchronisation required
 - More idle listening, more energy consumption
 - Occupy the channel, bigger slots
 - Fewer slots per second, less data
- Ideally as short as possible, but long enough to maintain robust synchronisation



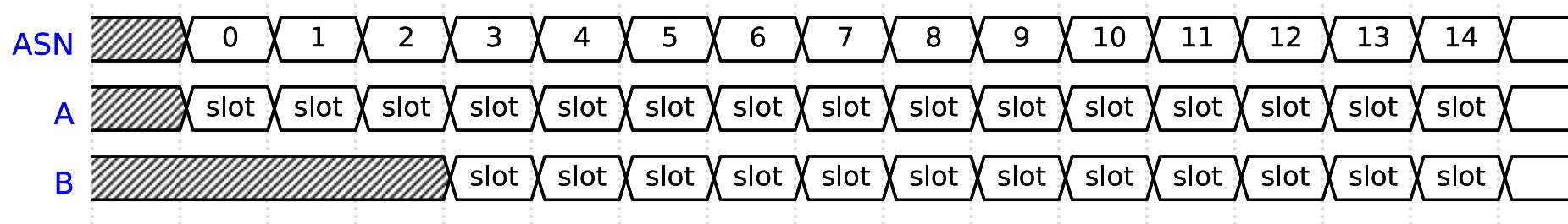
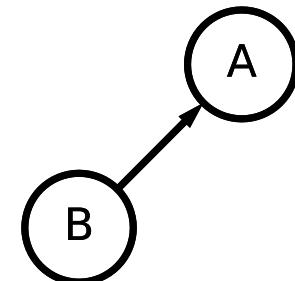
TSCH: Frame-Based and ACK-Based Synchronisation

- TSCH uses implicit synchronisation with a neighbour on each frame transmissions
- The receiver calculates the difference between the expected and actual time of frame arrival
- In frame-based synchronisation, the receiver uses this offset to synchronise to the clock of the sender
- In ACK-based synchronisation, the receiver adds this offset in the acknowledgement frame and the sender uses it to synchronise to the clock of the receiver
- In the absence of data, devices need to exchange an empty *Keep-Alive* frame to re-synchronise
 - Significant overhead in applications that transmit very infrequently, e.g. smart metering



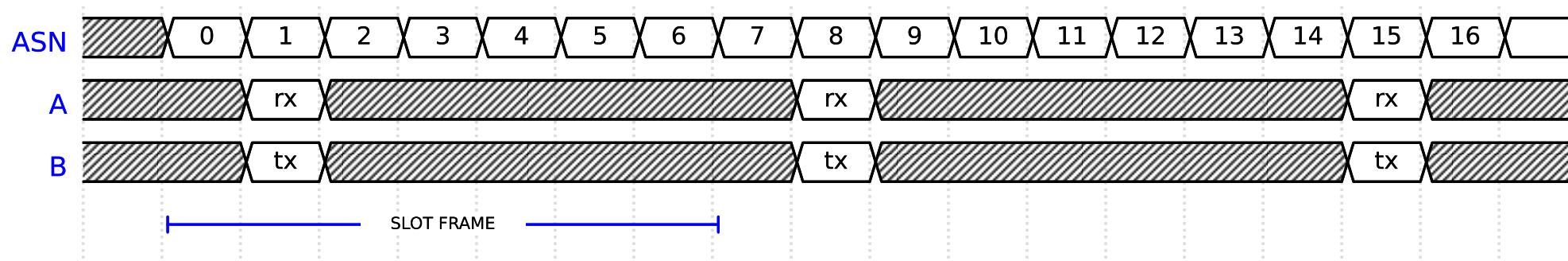
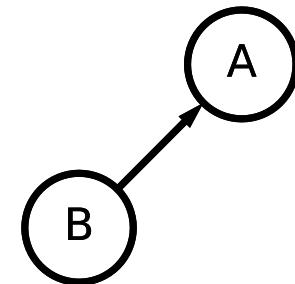
TSCH: Absolute Slot Number

- A number that counts slots since the formation of the network
 - 5 bytes so that it can increment for hundreds of years without wrapping
- ASN Synchronisation
 - ASN is initialised to 0 by the device that forms the network
 - Devices learn the current value of ASN when they join the network
 - Since they are synchronised, all devices know the current value of ASN at any time



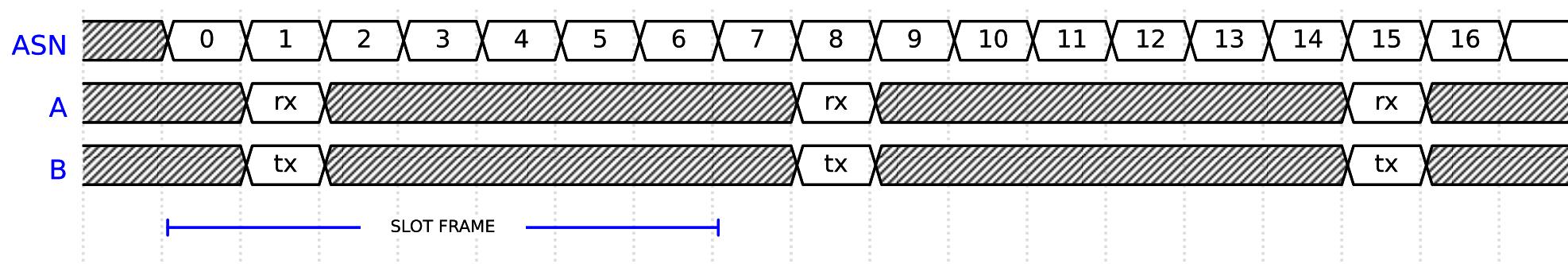
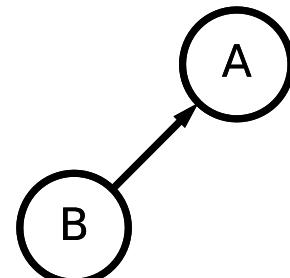
TSCH: Slot frame and Schedule

- Time slots are organised in a slot frame
 - A slot frame is a continuously repeating pattern of slots
 - Slot frame size (not defined by the standard, tailored to the application)
 - $[\text{slot offset}] = [\text{ASN}] \% [\text{slot frame size}]$
- Communication is based on a schedule, for example:
 - Slot frame size = 7
 - B transits to A in slot offset = 1



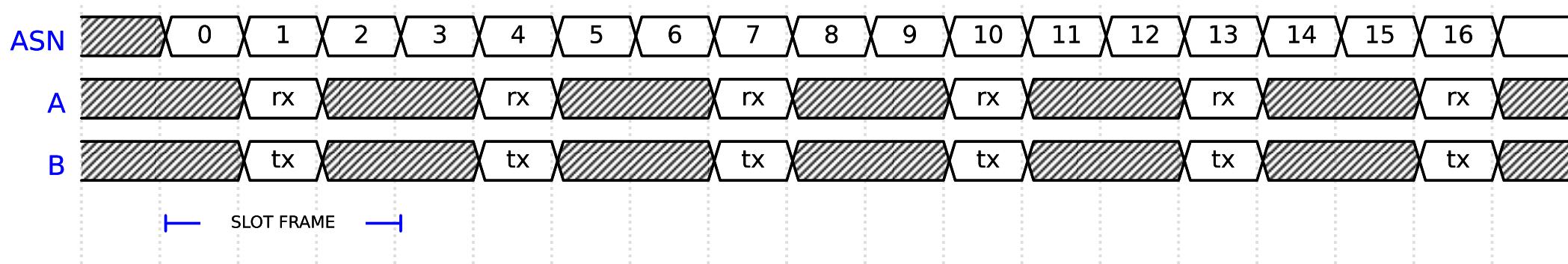
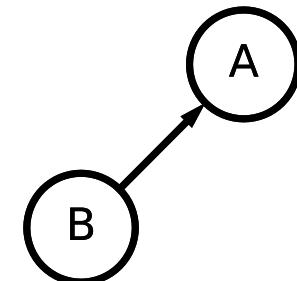
TSCH: The Schedule Controls Several Performance Metrics

- Assuming timeslot size 10 ms and frame size 7...
- Link Capacity: $100 / 7 = 14.28$ packets per second
- Energy Consumption: $1 / 7 = 14.28\%$ radio duty cycle
- Latency: up to $7 \times 10 \text{ ms} = 70 \text{ ms}$



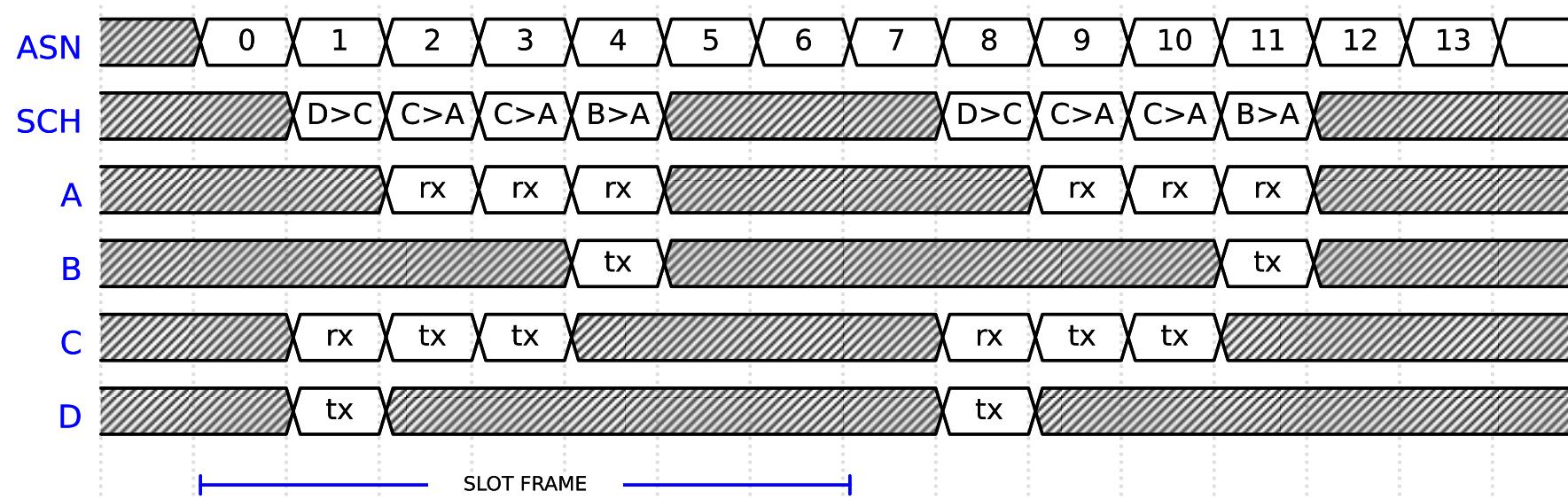
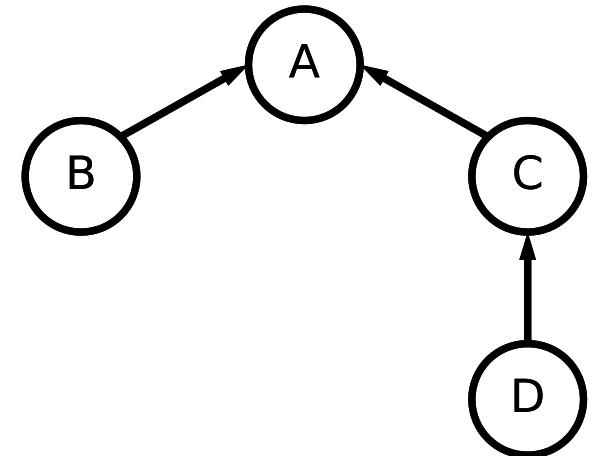
TSCH: Slot Frame Size

- Defines the number of idle slots in a frame
- Shorter slot frame size
 - More transmission opportunities -> higher capacity for data
 - More re-transmission opportunities -> higher reliability
 - Shorter average and worst-case latency
 - More unused slots -> worse energy-efficiency
 - Less room for other devices -> worse scalability



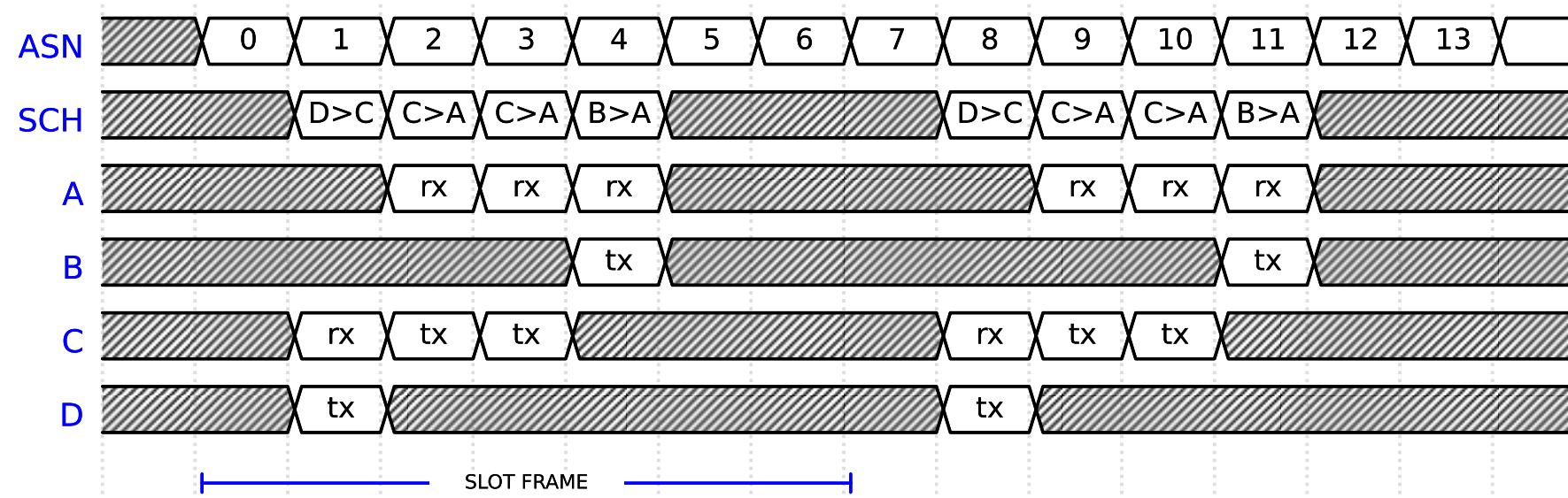
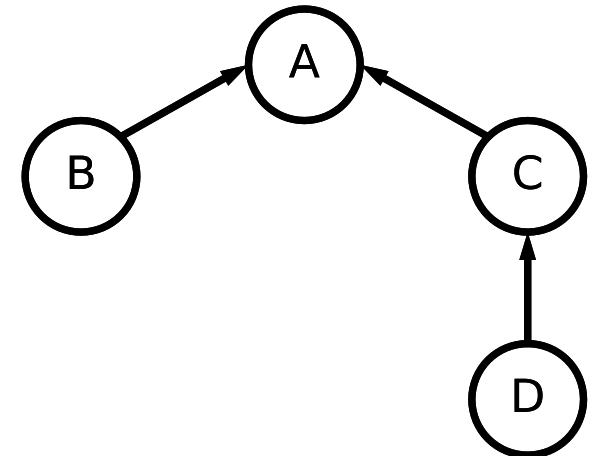
TSCH Schedule: Dedicated Slots

- In each slot, a device can transmit (Tx), receive (Rx), or sleep (idle)
- Dedicated slots (Contention-free)
 - Slot dedicated to a specific link
 - One assigned transmitter -> no collisions
 - Example: B>A, means B transmits (Tx) and A receives (Rx)



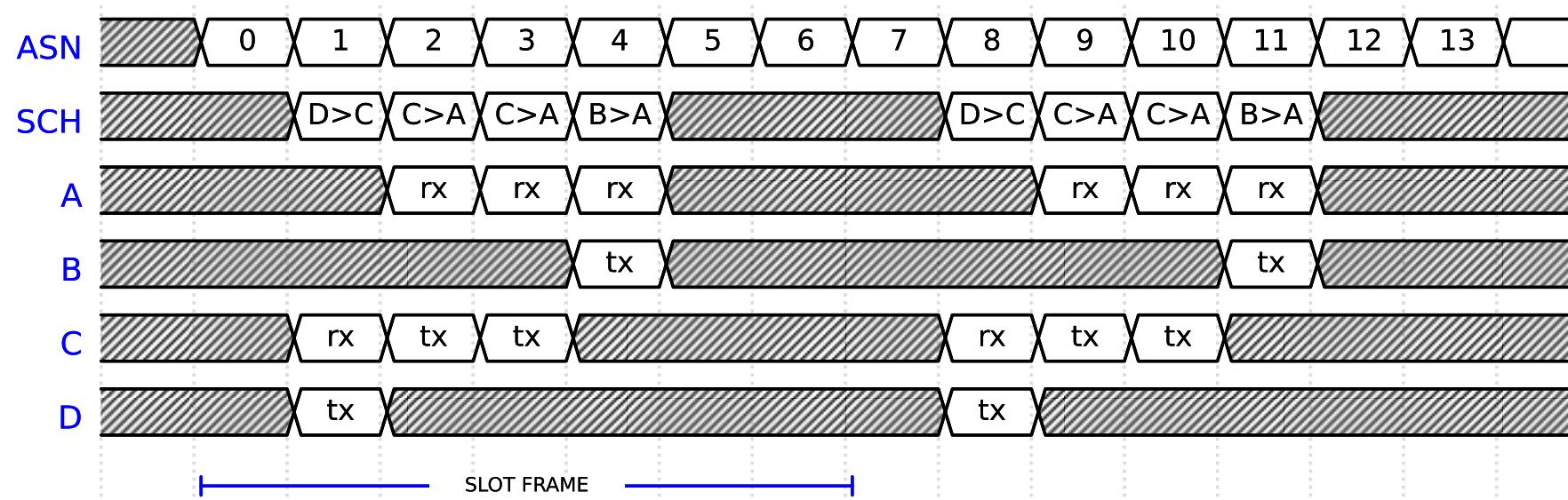
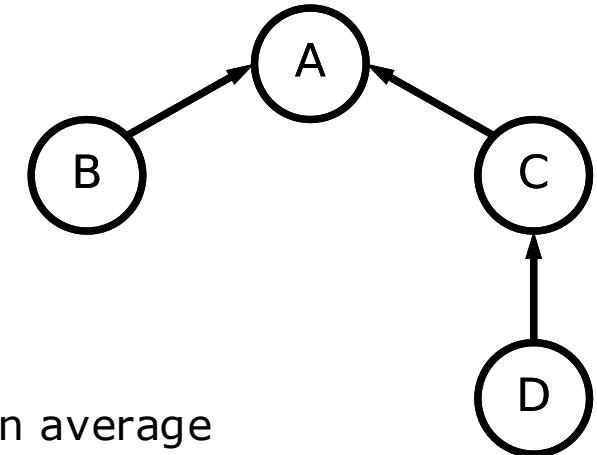
TSCH Schedule: How many slots?

- Why did I allocate 2 slots per frame for link C>A?
- How many slots per frame I need generally?



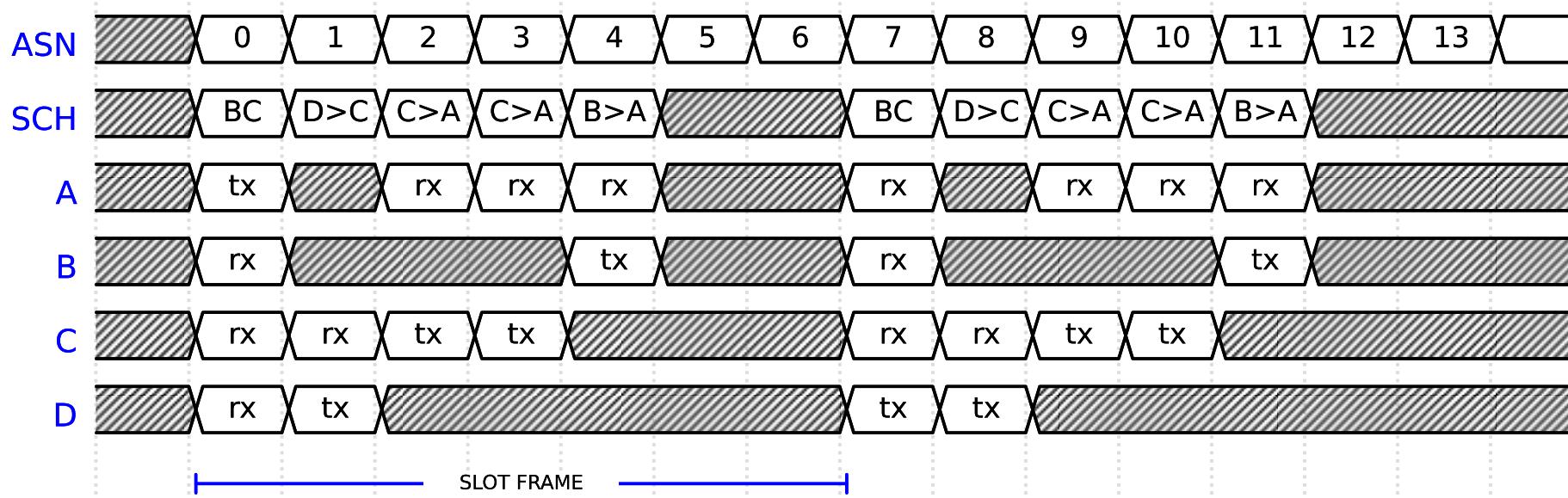
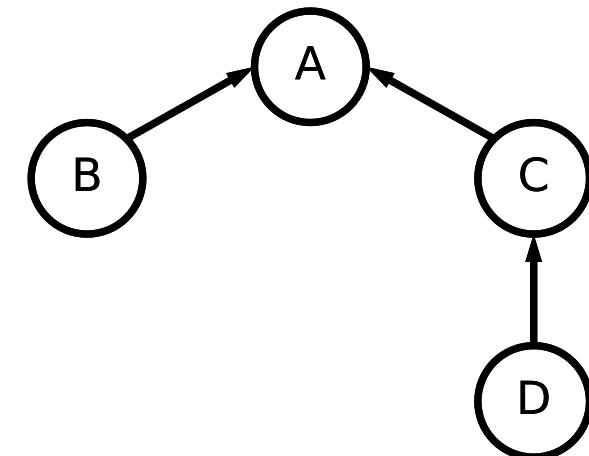
TSCH Schedule: How many slots?

- Slots in the schedule should be able to have capacity for:
 - Packets generation rate by the transmitter
 - Packets forwarding rate by the transmitter
 - Retransmissions (ETX, Expected Transmission Count)
 - $ETX = 1/p$ where p is packet reception probability
 - Example: if $p=20\%$, I have to retransmit each packet 5 times on average



TSCH Schedule: Broadcast Slots

- Broadcast slots (every node active)
 - If I have something to broadcast, I transmit
 - If I have nothing to broadcast, I listen/receive
 - Typically, the first slot of the frame
 - Collisions may happen
 - Transmitters cannot receive broadcasts of others

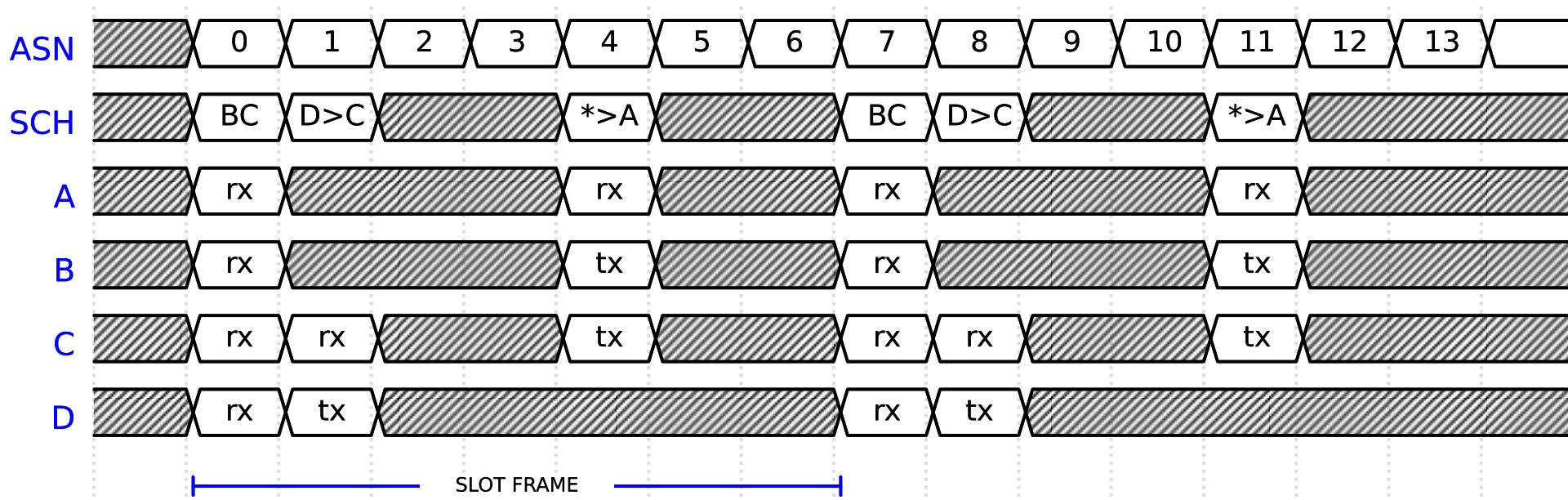
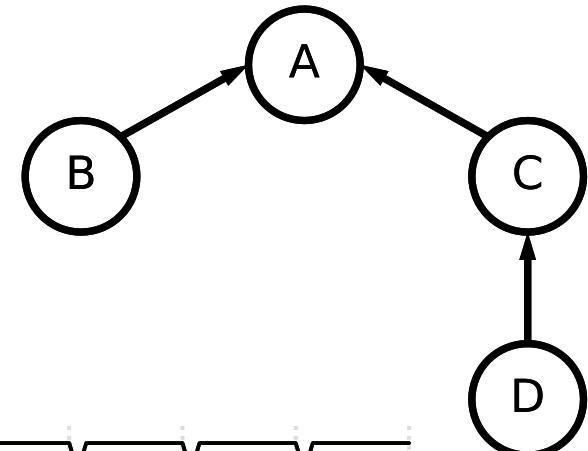


TSCH: Slots Are Limited

- Assuming a time slot of 10 ms, there are only 100 slots per second
 - Permit for maximum 100 packets per second
- What can do if we run out of slots, and we want to support more devices, traffic or retransmissions?

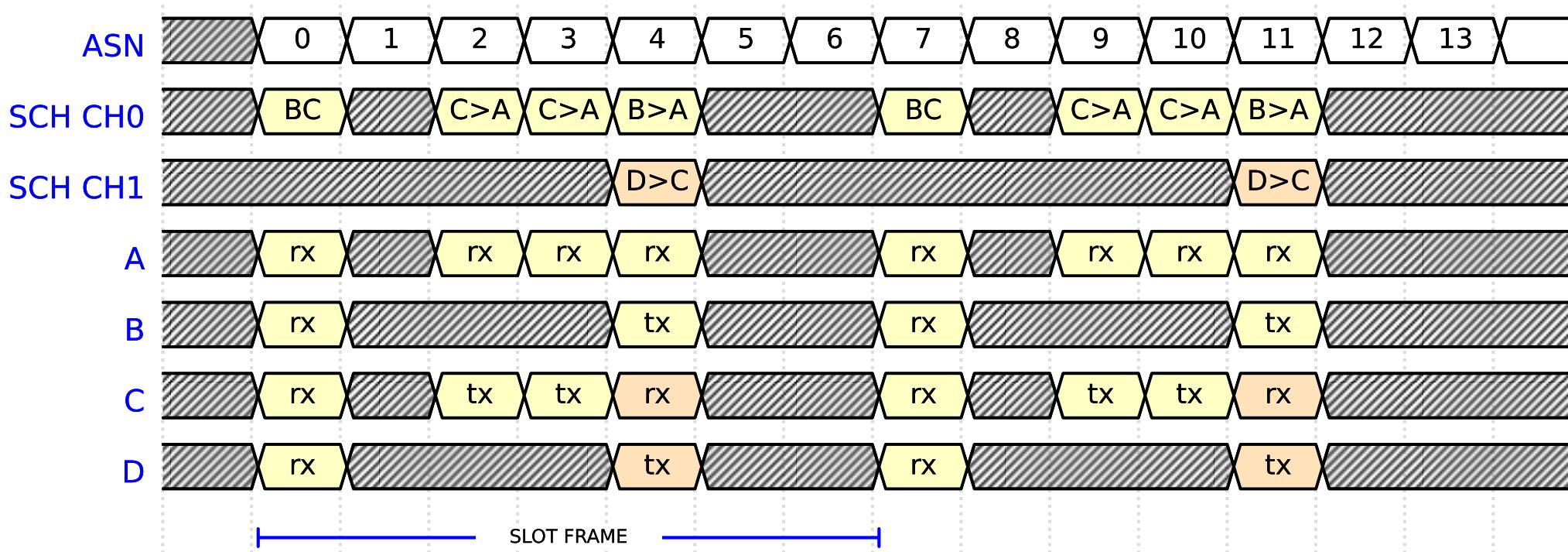
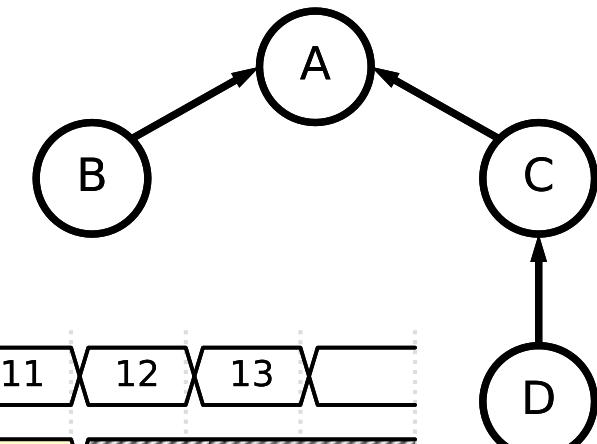
TSCH Schedule: Shared Slots

- Shared slots (Contention-based)
 - Slot assigned to multiple transmitters
 - Collisions are possible, need CSMA/CA



TSCH Schedule: Multiple Channels

- Parallel transmissions are possible in different channels
 - A transmitter and receiver can only tune to one specific channel

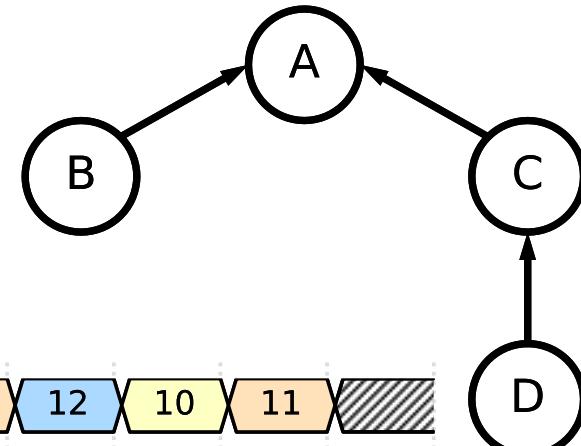
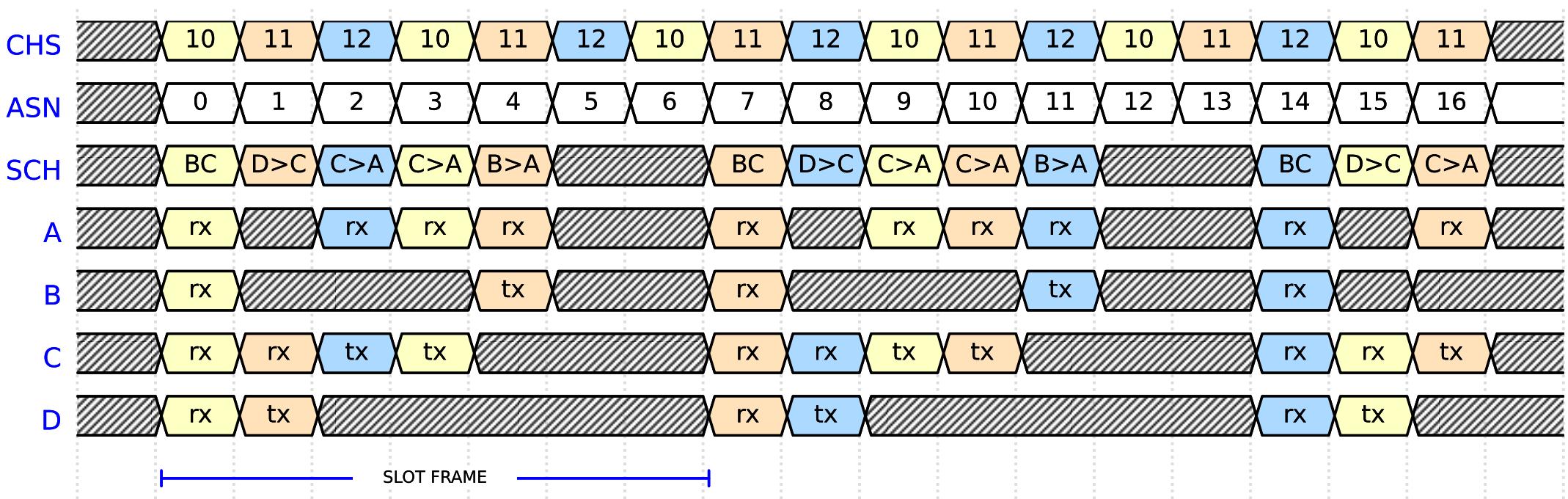


Some Channels are better than others

- IEEE 802.15.4 use retransmissions to combat channel errors
- Interference and fading errors are correlated in time
 - If a channel experiences a lot of interference now, it is very likely to experience a lot of interference in the next frame
- Yet, interference and fading of different channels are often uncorrelated
 - If a channel experiences a lot of interference now, it is very likely that another channel will not be affected by the same source of in the next frame
- Idea: Change channels for each retransmission to enhance reliability

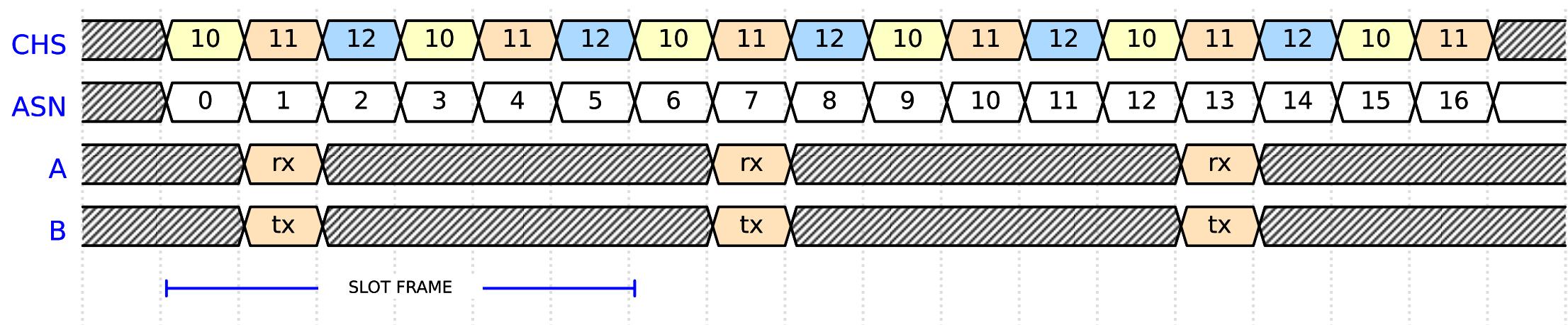
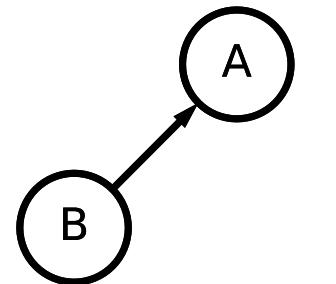
TSCH: Channel Hopping

- Channels change following a pre-agreed pseudo-random order
- All nodes agree on a Channel Hopping Sequence (CHS), e.g. 10-11-12
- Channel = CHS[index], where index = ASN mod len(CHS)



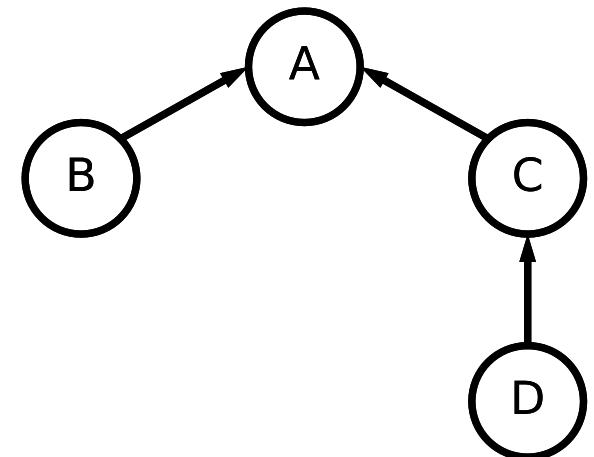
Slot Frame Size vs Channel Hopping Sequence Size

- The Slot Frame Size and CHS size must be relatively prime
 - Their highest common factor should be 1
- Otherwise, there is no channel hopping!
- Example: frame size is 6 and CHS size is 3



The TSCH Schedule

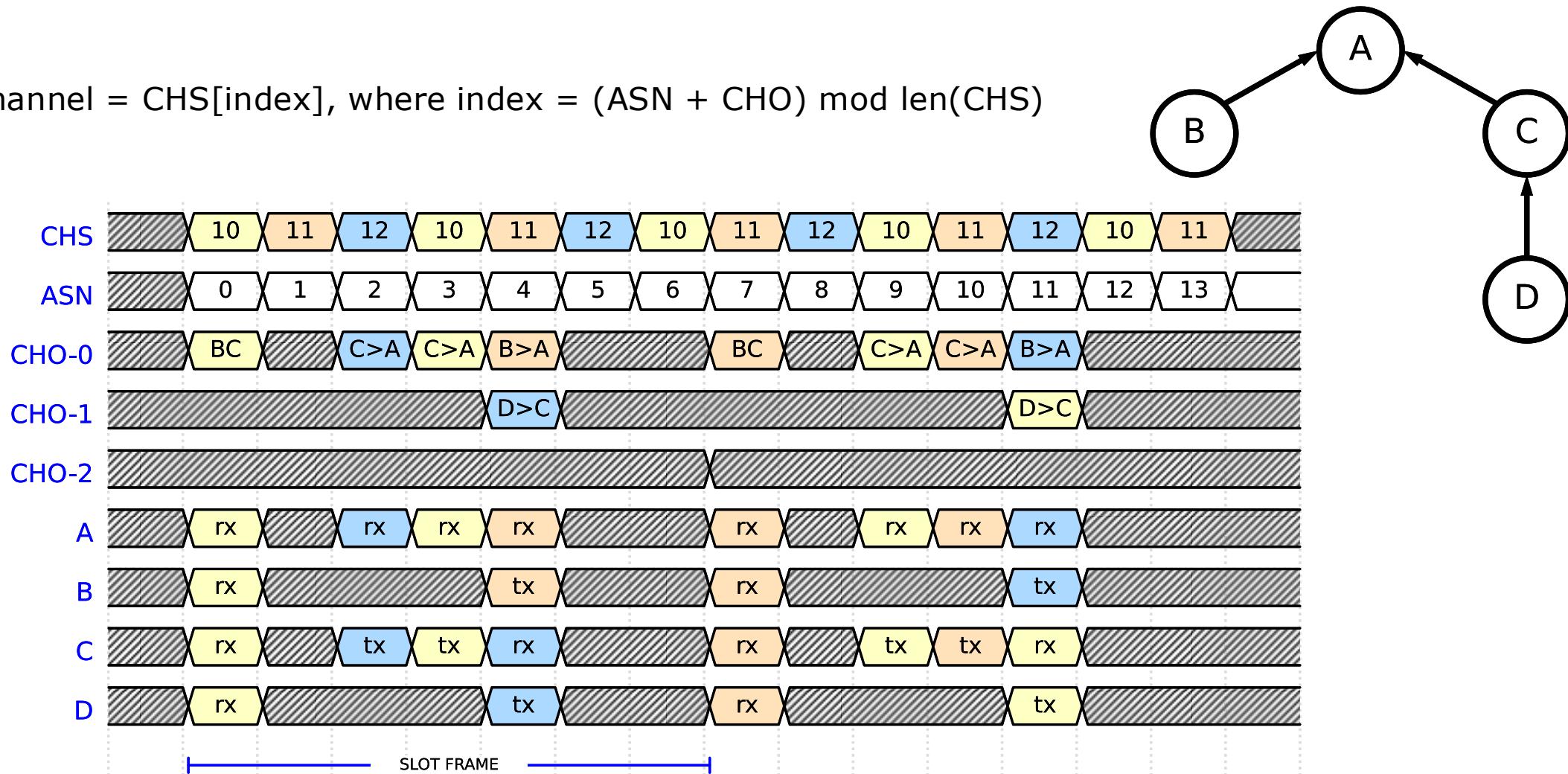
- A N-by-M table where N is the slot frame size and M is the CHS size
 - Each column represents a slot offset (SO)
 - Each row represents a channel offset (CHO)
- Channel = CHS[index], where index = $(ASN + CHO) \bmod \text{len(CHS)}$



CHO \ SO	0	1	2	3	4	5	6
0	BC		C > A	C > A	B > A		
1					D > C		
2							

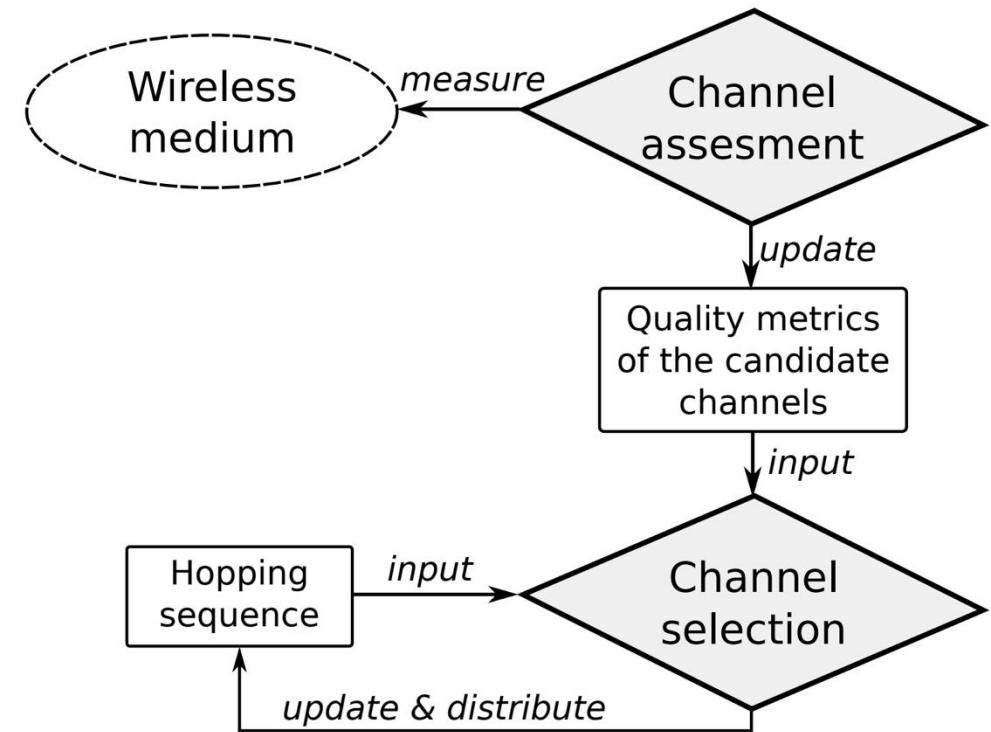
TSCH Schedule

- Channel = CHS[index], where index = (ASN + CHO) mod len(CHS)



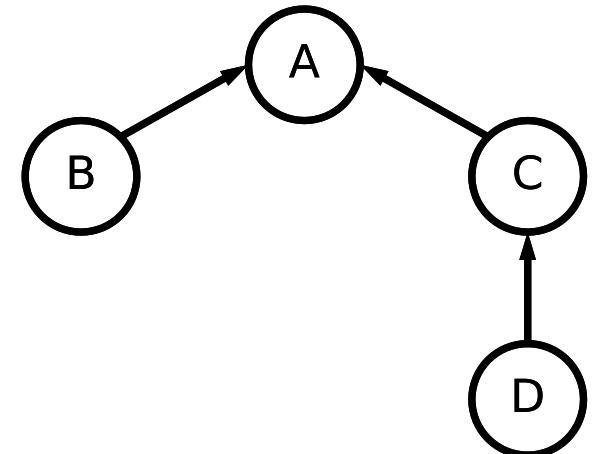
Channel Selection

- Consider a channel with a lot of interference
 - Using as the only single channel is very bad
 - Using it as part of the CHS is better, but inefficient
- How can we select which channels to use?
 - Some channels are more busy than others
 - This quality of the channels changes over time
- Adaptive Channel Selection
 - Measure the quality of channels
 - Update metrics with statistics
 - Select the best channels (blacklist/whitelist)
 - Distribute changes to the channel hopping sequence



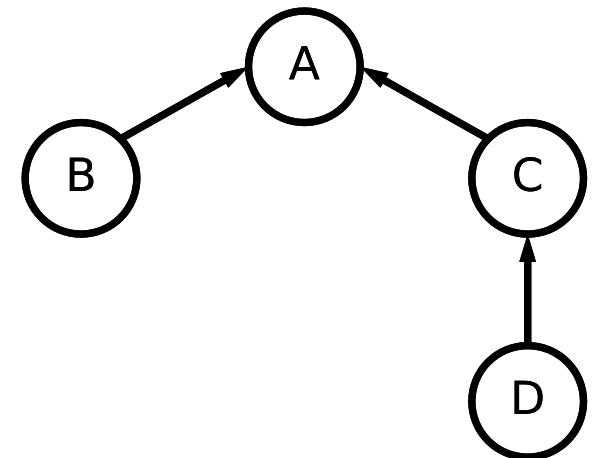
TSCH Scheduler

- A TSCH Scheduler is an algorithm that determines the TSCH schedule
- The Scheduler is not defined by the standard (flexible)
- Schedules are tailored to the use case
- Types of TSCH Scheduling
 - Static Scheduling
 - Dynamic Scheduling
 - Centralised Scheduling
 - Distributed Scheduling
 - Autonomous Scheduling



Static Scheduling

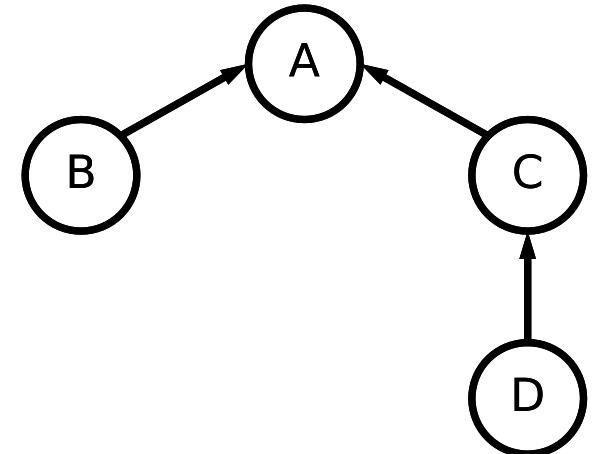
- Schedule is hardcoded on the firmware of the devices
- A static schedule can be handcrafted or algorithmically derived
- Makes sense only in relatively static TSCH networks
 - No. of devices, topology, traffic patterns, expected retransmissions



CHO \ SO	0	1	2	3	4	5	6
0	BC		C > A	C > A	B > A		
1					D > C		
2							

Dynamic Scheduling

- A minimal schedule with shared slots is defined
 - Hardcoded or installed when joining the network
- Dedicated slots are allocated dynamically
- Centralised Scheduling
 - The root node collects information about the nodes via these shared slots
 - Having the global picture of the network, it allocates dedicated slots to specific links
 - Such decisions can be determined using machine learning or reinforcement learning
- Distributed Scheduling
 - Neighbouring nodes exchange information and agree on which slots to use between them



IETF Minimal Schedule and 6top

- Minimal Schedule
 - One broadcast slot in offset (0,0)
 - The slot frame size and channel sequence size is undetermined
 - Hardcoded or installed when joining the network

CHO \ SO	0	1	2	3	4	5	...
0	BC						
1							
...							

THE COMMANDS SUPPORTED BY THE 6TOP PROTOCOL (6P)

Command	Code	Description
ADD	1	Add cell(s) between the two neighbors. The CellOptions bitmap indicates the type of cell(s) to add.
DELETE	2	Delete cell(s) from the schedule.
RELOCATE	3	Relocate cell(s) in the schedule. Used to handle schedule collisions.
COUNT	4	Count the cells with a particular CellOptions.
LIST	5	List the cells with a particular CellOptions.
SIGNAL	6	Placeholder for SF-specific commands.
CLEAR	7	Clears all cells between the two neighbors. Possibly used to handle schedule inconsistencies.

- 6top (6TiSCH Operation Sublayer Protocol)
 - A protocol to exchange messages for distributed scheduling

Image source: <https://doi.org/10.1109/COMST.2019.2939407>

Autonomous Scheduling

- Each node derives and changes its schedule autonomously
 - No need to exchange messages
- Shared slots are used for the routing protocol to determine parent nodes
 - Nodes autonomously determine when to transmit/receive using routing information
 - Devices hash the Node ID or MAC address (of sender or receiver) modulo frame size to determine which slot to use
- Orchestra is a widely used autonomous scheduler (albeit not a standard)

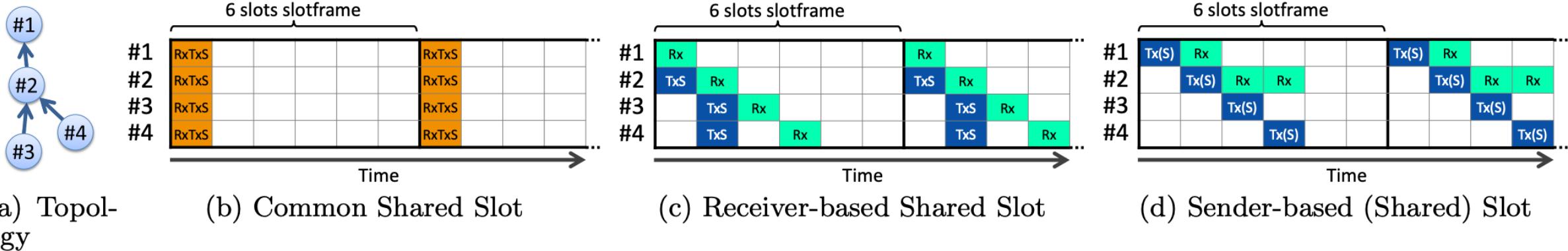


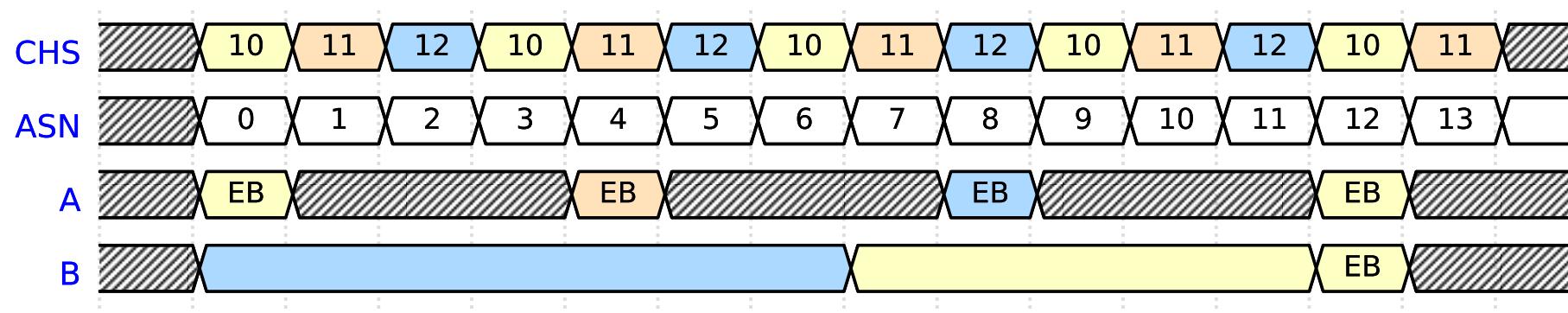
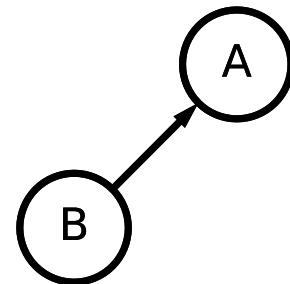
Image source: <https://doi.org/10.1145/2809695.2809714>

Joining a TSCH Network

- How does a new device joins a TSCH network?
- A new device that wants to join a TSCH network:
 - Does not know the schedule (unless static scheduling is used)
 - Does not know the current ASN
 - Does not know when the beginning of a frame/slot is (not in sync)
- Enhanced Beacon (EB)
 - Devices of the TSCH network periodically broadcast an EB to advertise the network
 - EB includes the current ASN and (part of) the schedule
- The joining device scans the wireless medium for an EB but without knowing the time and channel of the transmission

Searching for an EB packet

- The joining node scans sequentially all the channels until it receives an EB
 - If the CHS is fixed, the joining node needs to search only those channels
 - Otherwise, it needs to search all 16 channels
- Upon the reception of the first EB, the joining node keeps scanning for a potentially better parent until a timeout or a maximum number of candidate parents is reached
- The joining process is notoriously slow and inefficient
 - But often considered insignificant in static TSCH networks as it ideally happens just once



Industrial Wired Networks

- IEEE 802.1Q
 - Supports Virtual LANs (VLAN) over Ethernet (IEEE 802.3)
 - VLAN Tagging with 8 priority classes
- Time-Sensitive Networking (TSN)
 - A set of extensions on top of IEEE 802.1Q

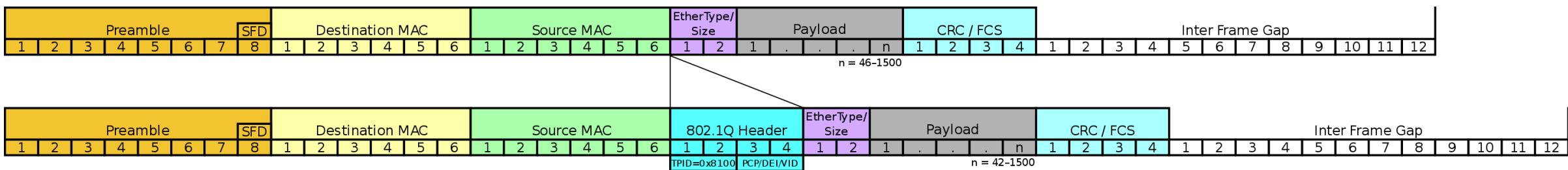


Image source: Wikipedia

Time Sensitive Networking (TSN)

- IEEE 802.1AS Timing and Synchronization for Time-Sensitive Applications
 - Precise time synchronisation based on PTP
- IEEE 802.1Qbv Enhancements to Traffic Scheduling: Time-Aware Shaper (TAS)
 - Synchronised TDMA-based medium access
 - Time is organised in repeating cycles of time slices
 - Time slices can be allocated to specific VLAN priorities for exclusive access
 - Time-critical traffic can be separated from non-critical background traffic

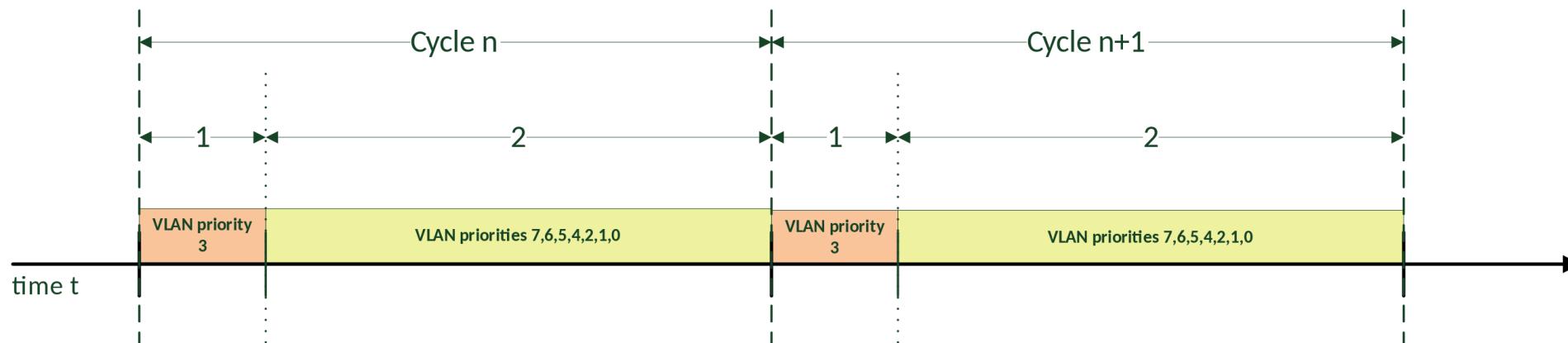


Image source: Wikipedia