
DTU

Networked Embedded Systems

Week 9: Embedded System Performance

Xenofon (Fontas) Fafoutis
Associate Professor
xefa@dtu.dk
www.compute.dtu.dk/~xefa

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

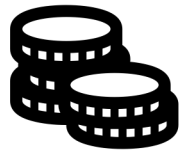
Measuring Reliability

- 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 $<10^{-5}$

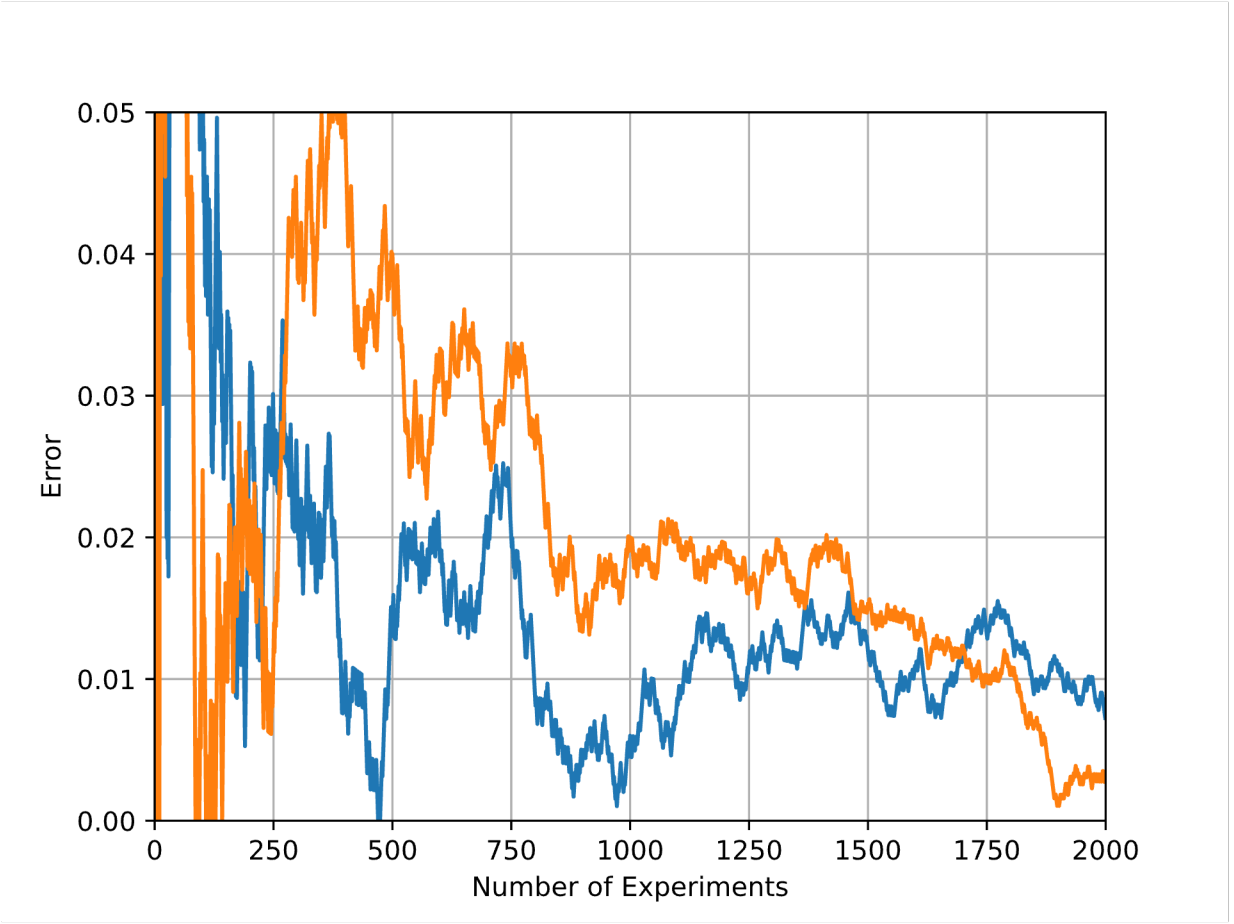
- Challenge
  - Measuring reliability very accurately needs a lot of samples

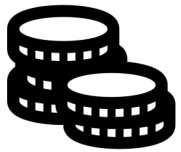
Example: Fair Coins



- 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?

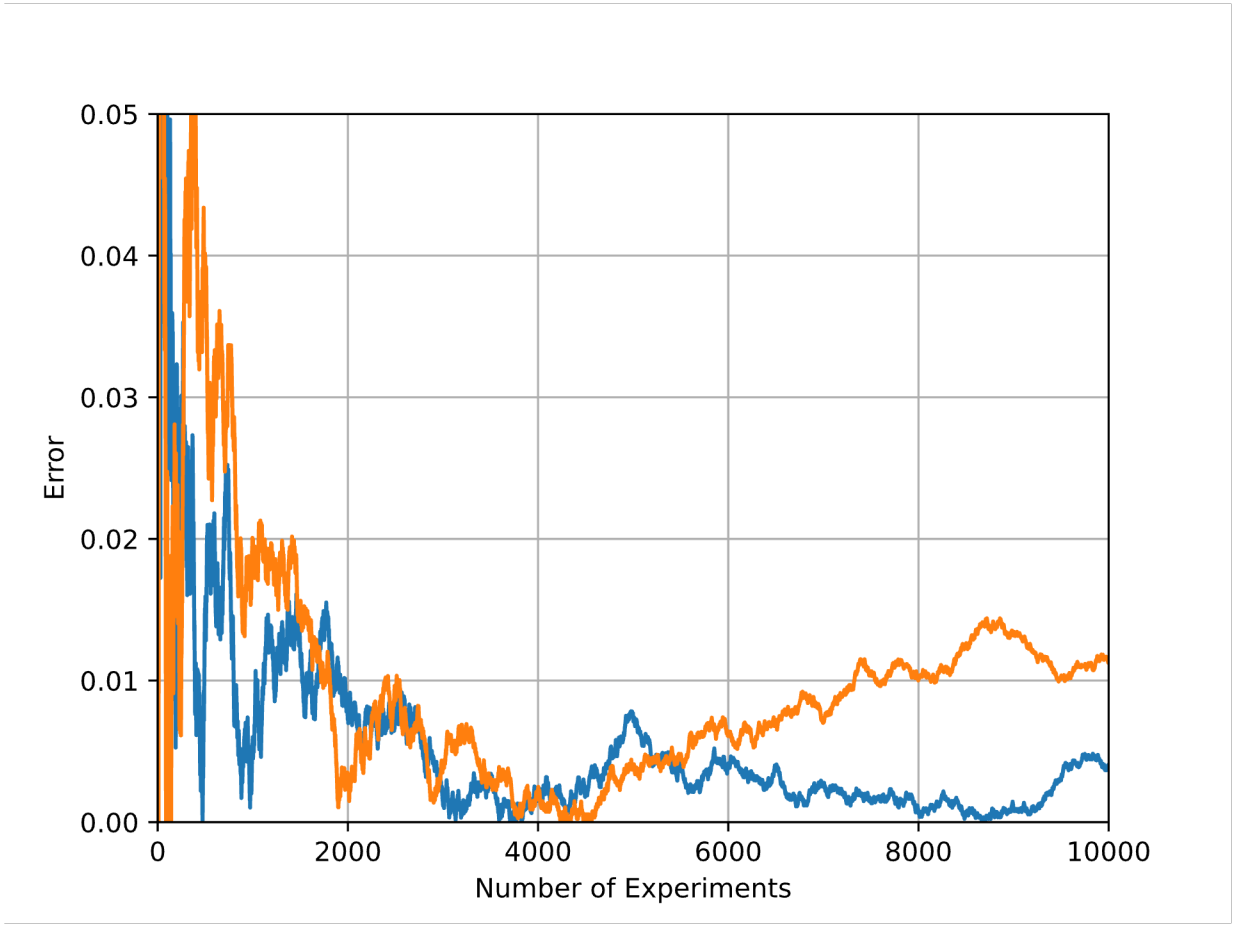
Example: Fair Coins (2000 tosses)

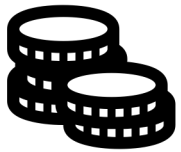




Example: Fair Coins (10000 tosses)

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

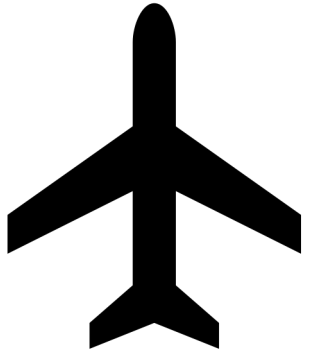




Airplane

- 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?

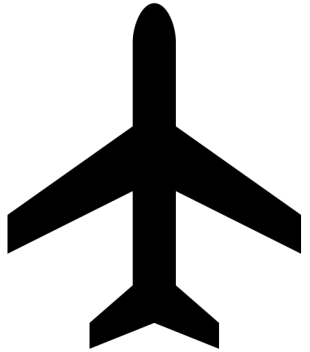


Airplane

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

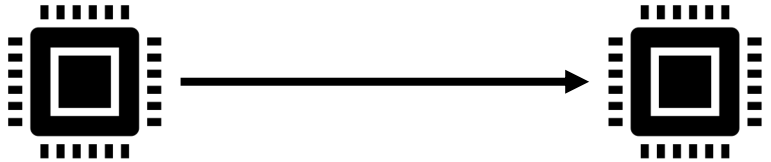
- Why do commercial airplanes have two engines, when one is sufficient to fly?

- For safety, if one pilot becomes unconscious or an engine fails, the plane can still be landed safely



Reliability After Countermeasures

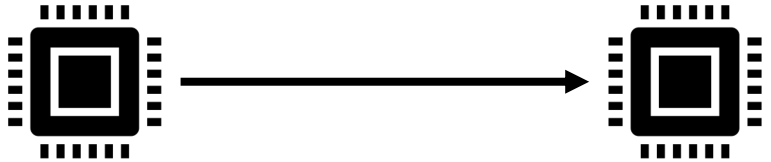
- The world is often unreliable, so we take actions to improve reliability
- The user/application experiences the reliability after all countermeasures have been applied



- 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

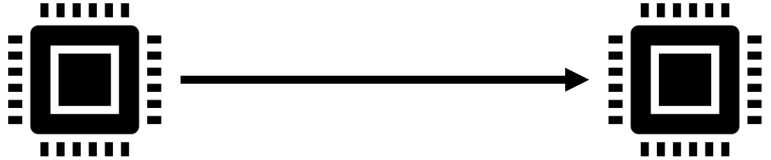


- The user/application experiences the reliability after all countermeasures have been applied

- 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 - $(1\text{-PRR})^4$ = 1 – 0.4^4 = 0.9744
    - If N=8, I have a failure if all 8 attempts fail, PDR = 1 - $(1\text{-PRR})^8$ = 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

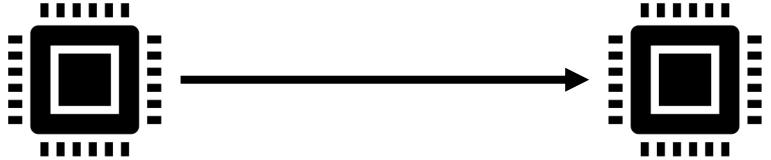
Let’s compare two reliability approaches…

- 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 = $1\text{-}0.4^8 = 0.99934$
- 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 = $1\text{-}0.4^8 = 0.99934$
- 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

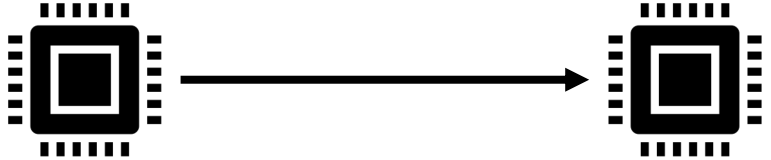
- Wired and (especially) Wireless Communication are particularly prone to errors

- 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)

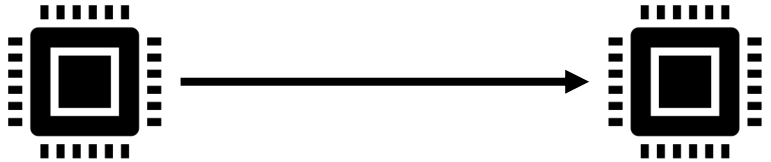
Delay (Latency)

- 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



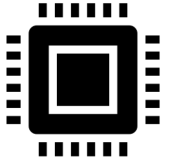
- Network
  - Transmission delay (time required to put bits on medium)
  - Propagation delay (time required for signal to travel through medium)
  - Duty cycle related delays (wait for receiver to be awake)
  - Congestion/error recovery (retransmissions, backoff time)
  - Multi-hop transmission (all 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 now 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)

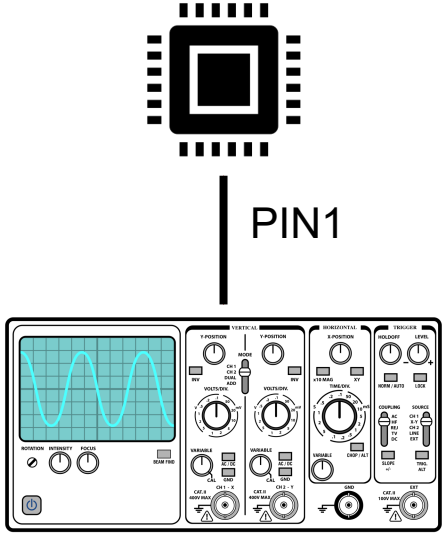
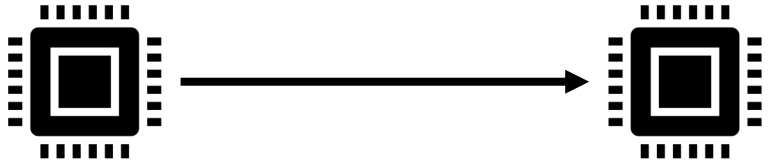
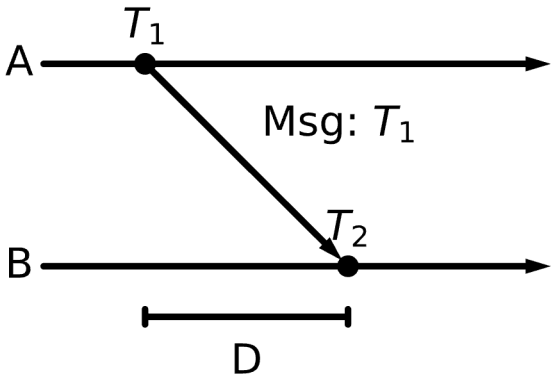


Image source: Wikipedia

Measuring Communication Delay

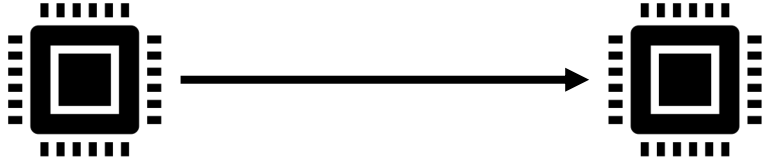
- <delay> = <time at destination> – <time at source>
  - $D = T_2 - T_1$
- Easy, right?

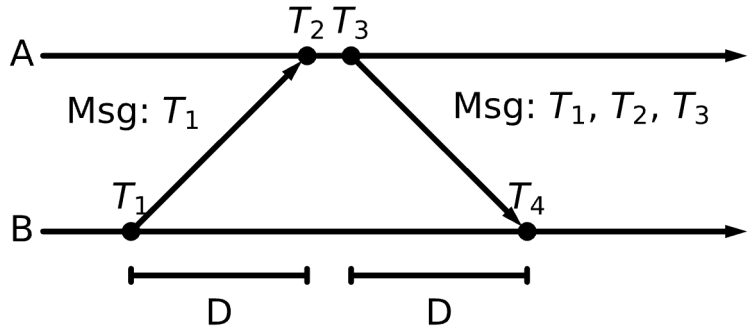




Measuring Communication Delay

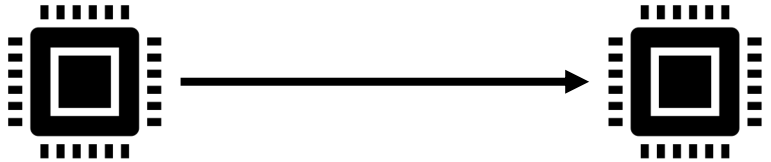
- <delay> = <time at destination> – <time at source>
- Easy, as long as source/destination are time sync'ed
- Expensive to add time synchronisation, if application does not needed it
- Alternatively, we can rely on round-trip delay
  - Exchange ping-pong messages
  - Minimise processing delays between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



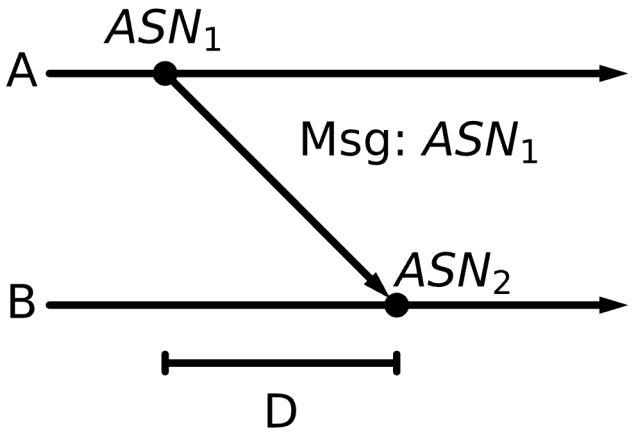


Taking Advantage of Existing Time Synchronisation

- 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 ~10-15ms)



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

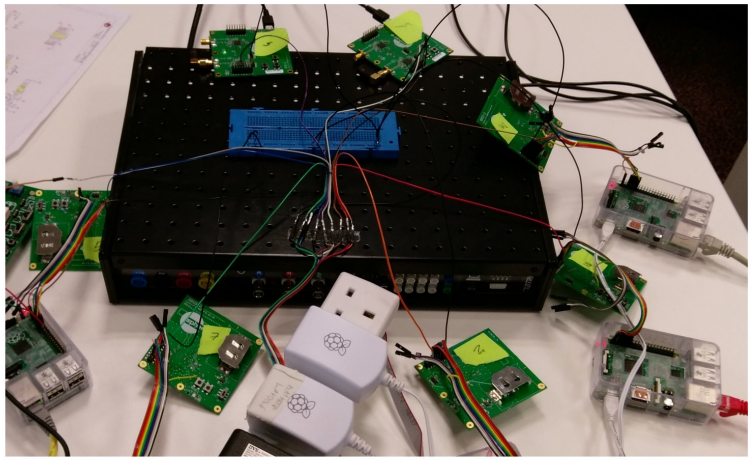


Fig. 9: Logic analyzer setup with seven nodes under test.

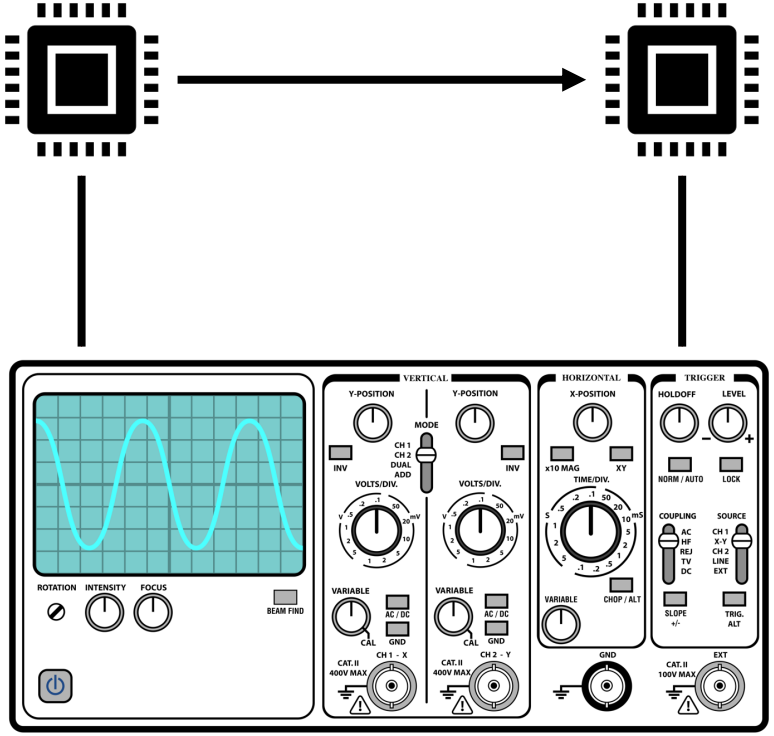
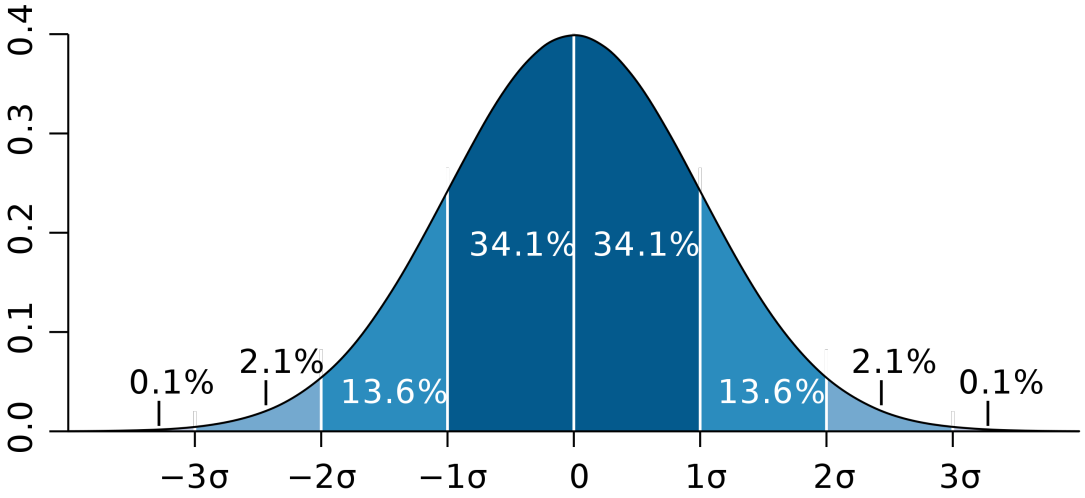


Image source: https://doi.org/10.1109/LCN.2016.042 (left) and Wikipedia (right)

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: $D_{MAX} = \max(\{D_1, D_2, \ldots, D_i\})$
- Box plot (min, max, median, first/third quartile)



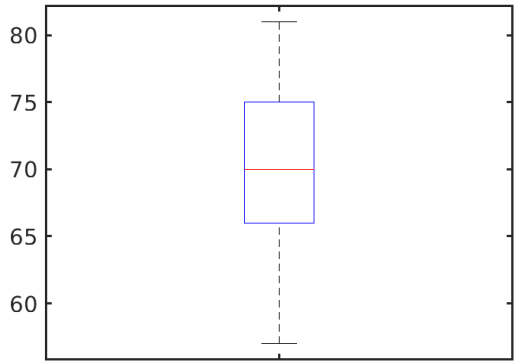


Image source: Wikipedia

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?

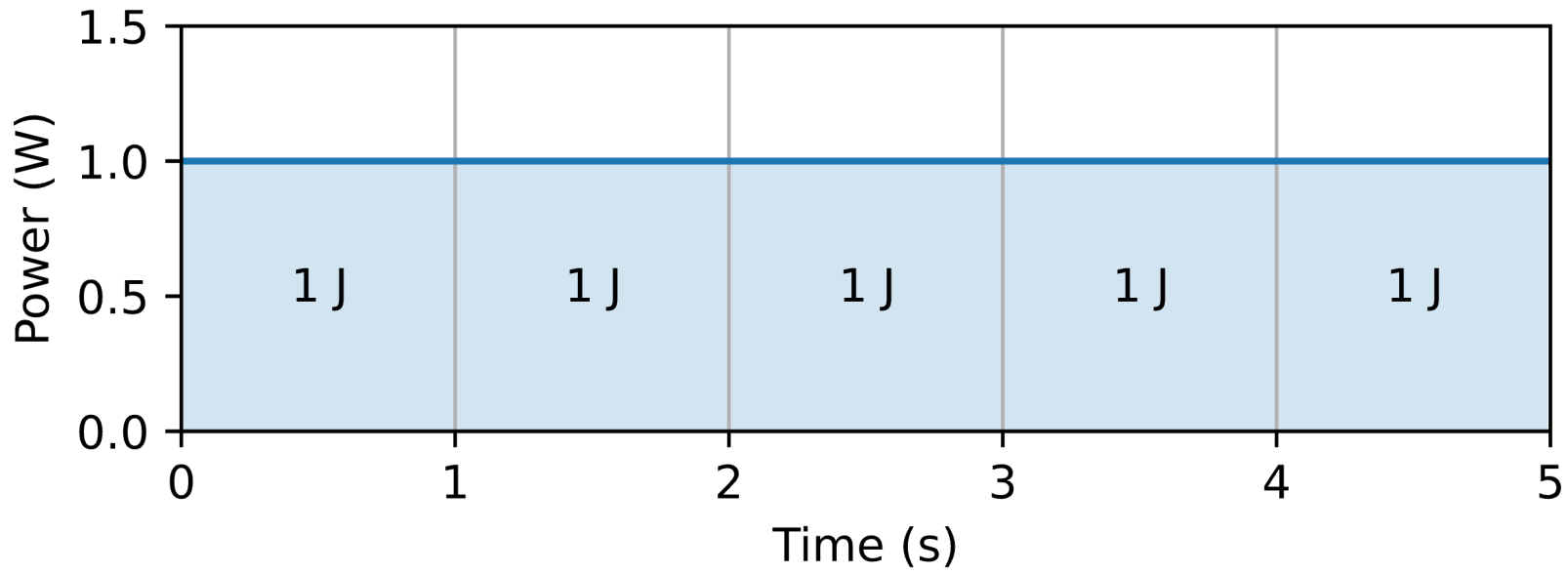
Energy Efficiency

- 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

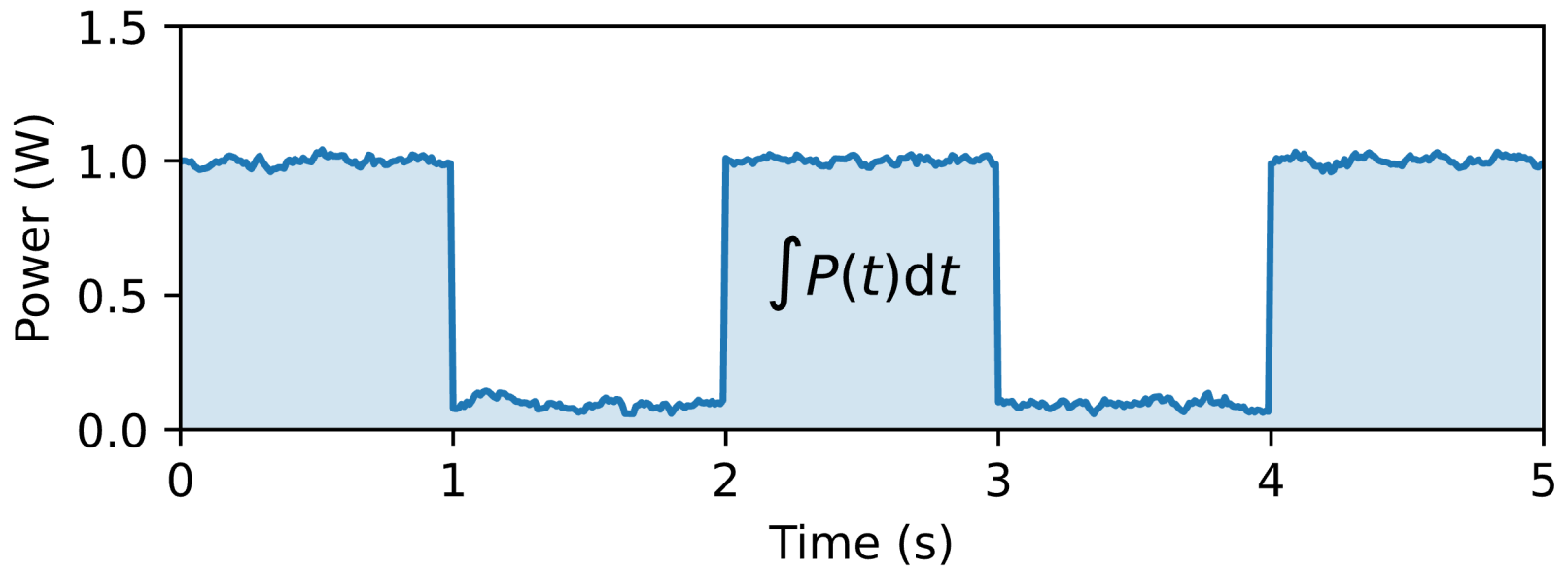
- Power is the rate of energy consumption
  - $E = P \times T$ (1 Joule = 1 Watt x 1 second)



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?

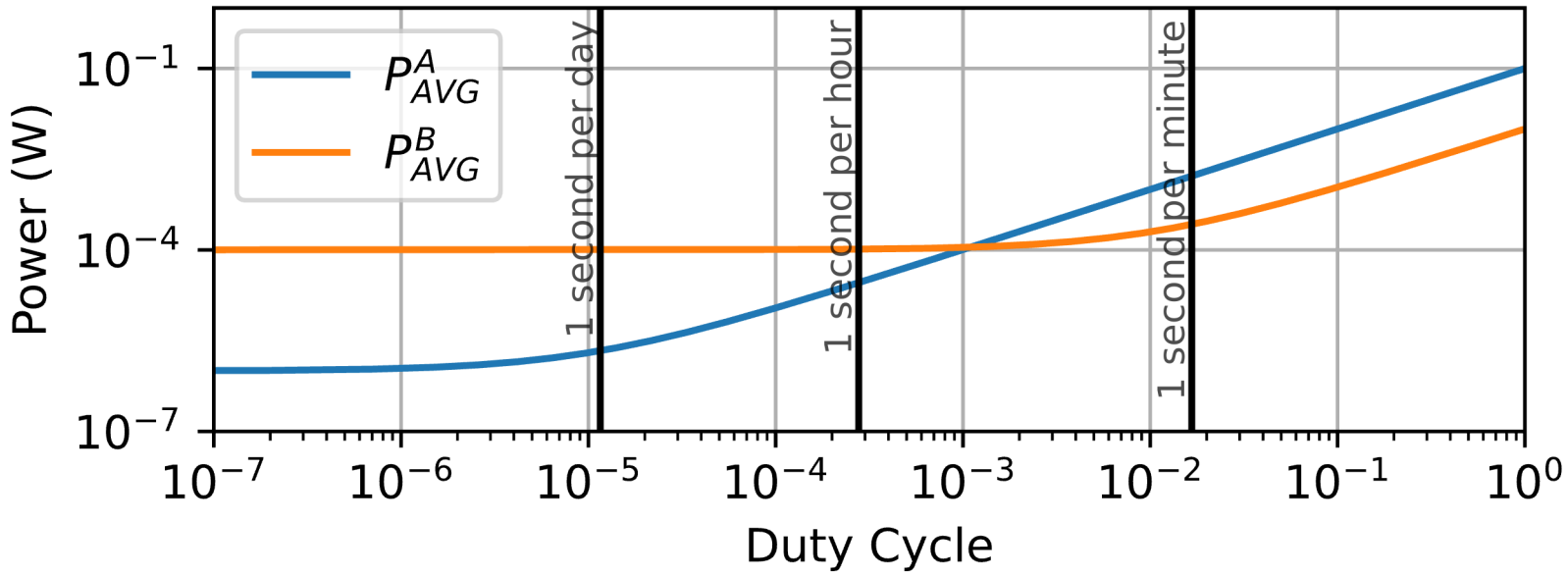
Quiz

- 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

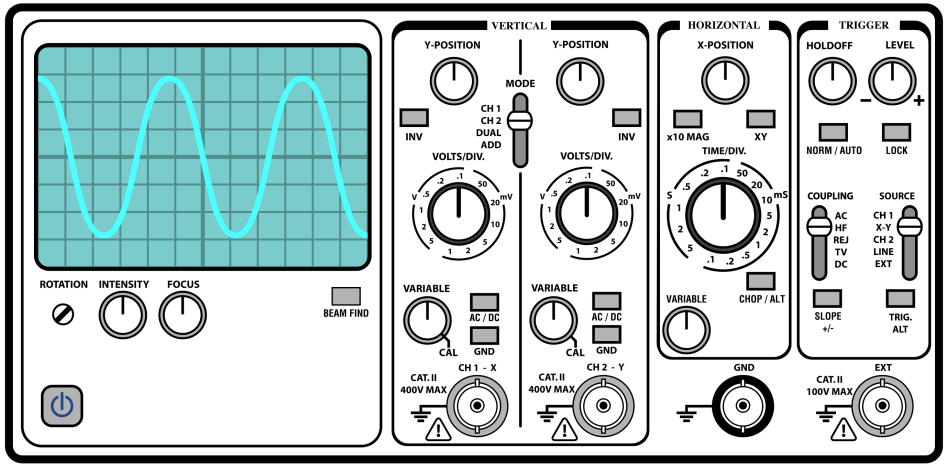
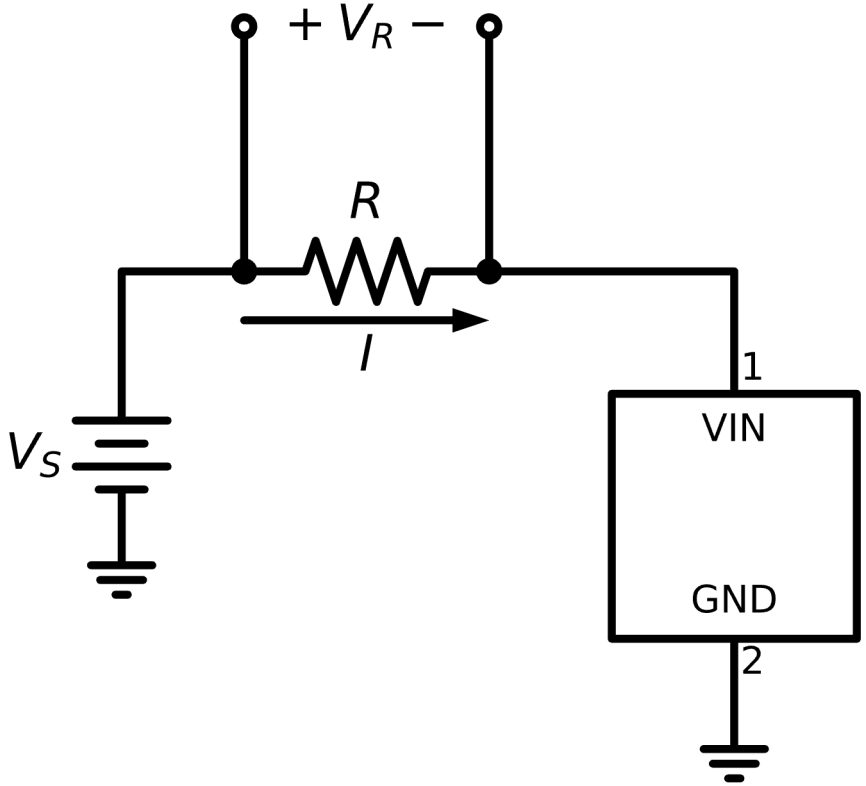


Image source: Wikipedia

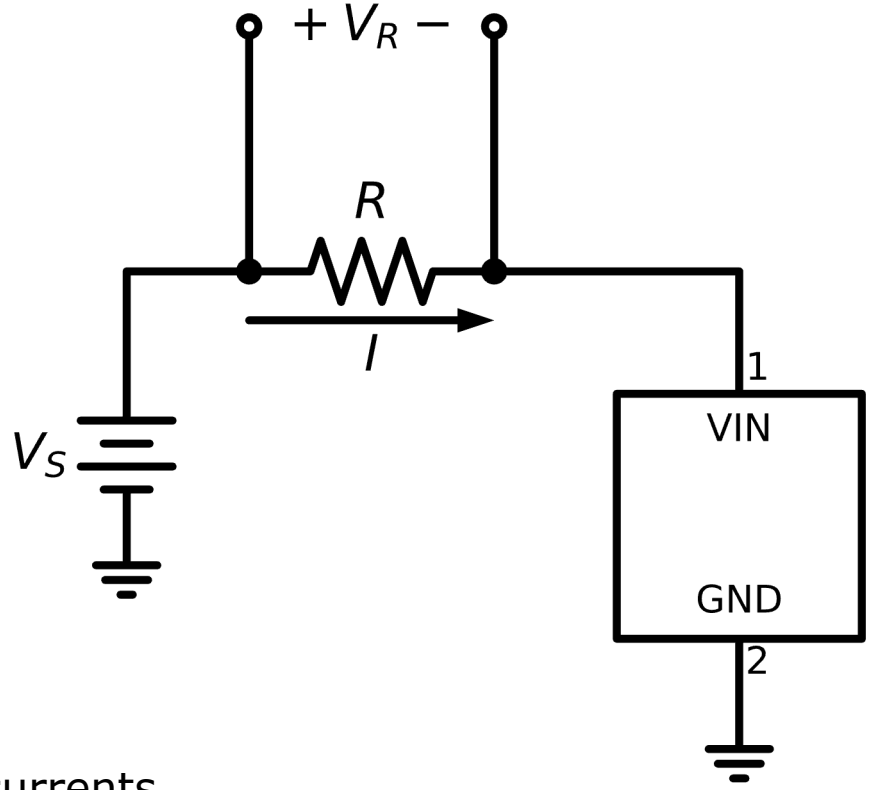
High-Side Shunt Resistor

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



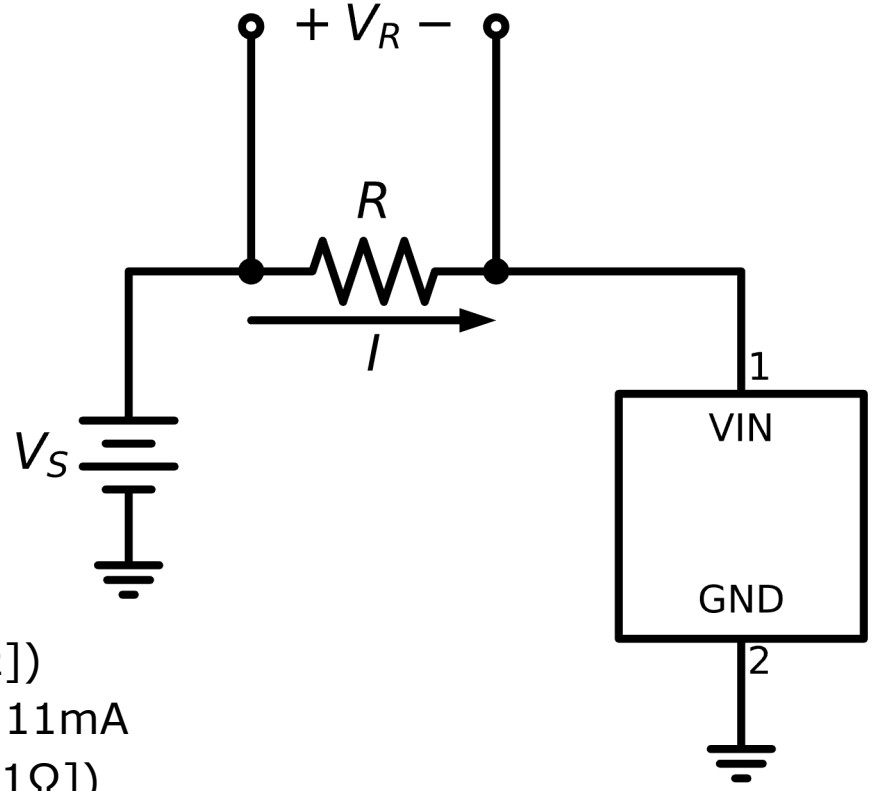
High-Side Shunt Resistor

- 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=100Ω) it can disrupt the circuit
  - Max voltage drop: 10mA x 100Ω = 1V, so V_{IN}=2.3V
- If too small (R=1Ω) it can be difficult to measure small currents
  - Max voltage drop: 10mA x 1Ω = 10mV, so V_{IN}=3.29V (OK)
  - Min voltage drop: 10uA x 1Ω = 10uV



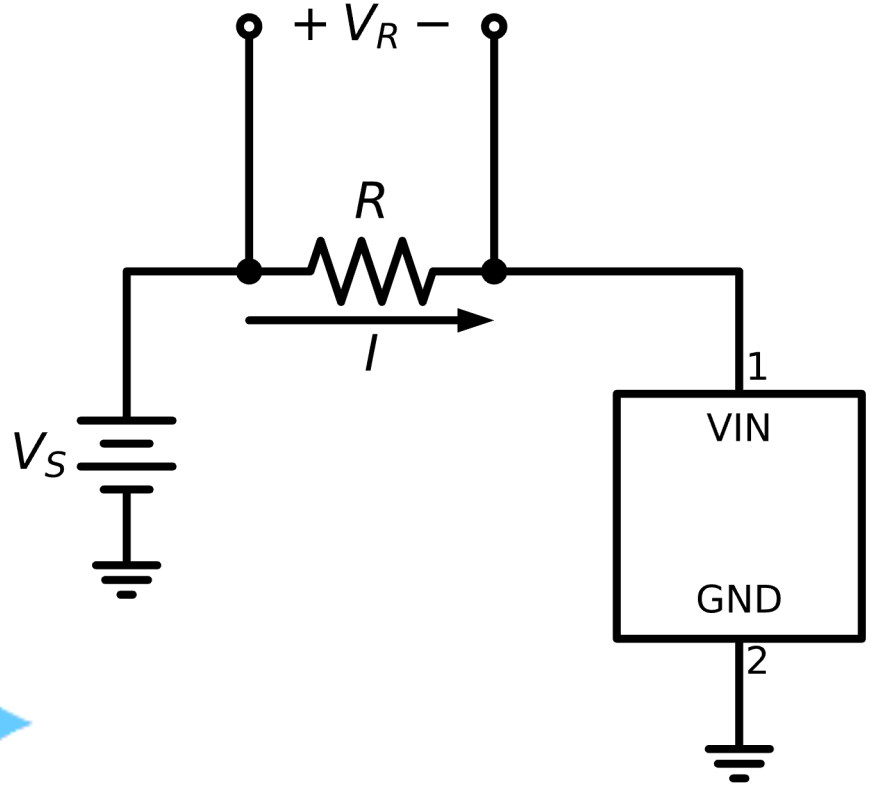
Resistor Tolerance

- 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=10mV
- If I used a 10Ω resistor with 10% tolerance (R=[9Ω,11Ω])
  - The current is between 10/11=0.91mA and 10/9=1.11mA
- If I used a 10Ω resistor with 1% tolerance (R=[9.9Ω,10.1Ω])
  - The current is between 10/10.1=0.99mA and 10/9.9=1.01mA
- 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
- $E = V_S \int_{t_0}^{t_1} I(t)dt = V_S \sum_{i=0}^{n} \frac{V_R^i}{R} \tau$



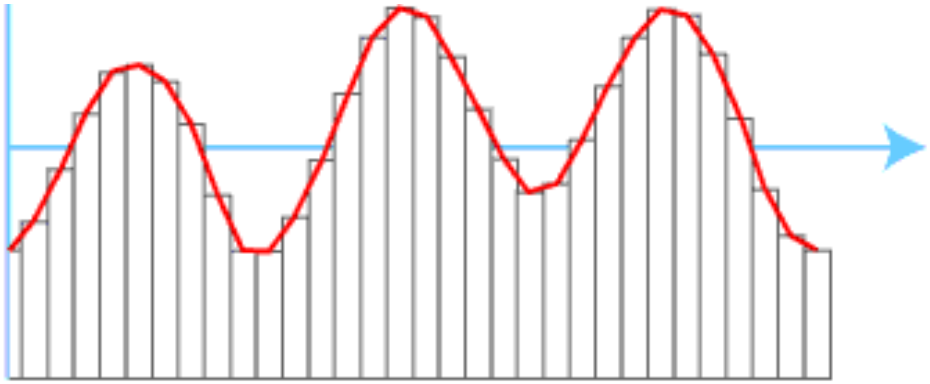


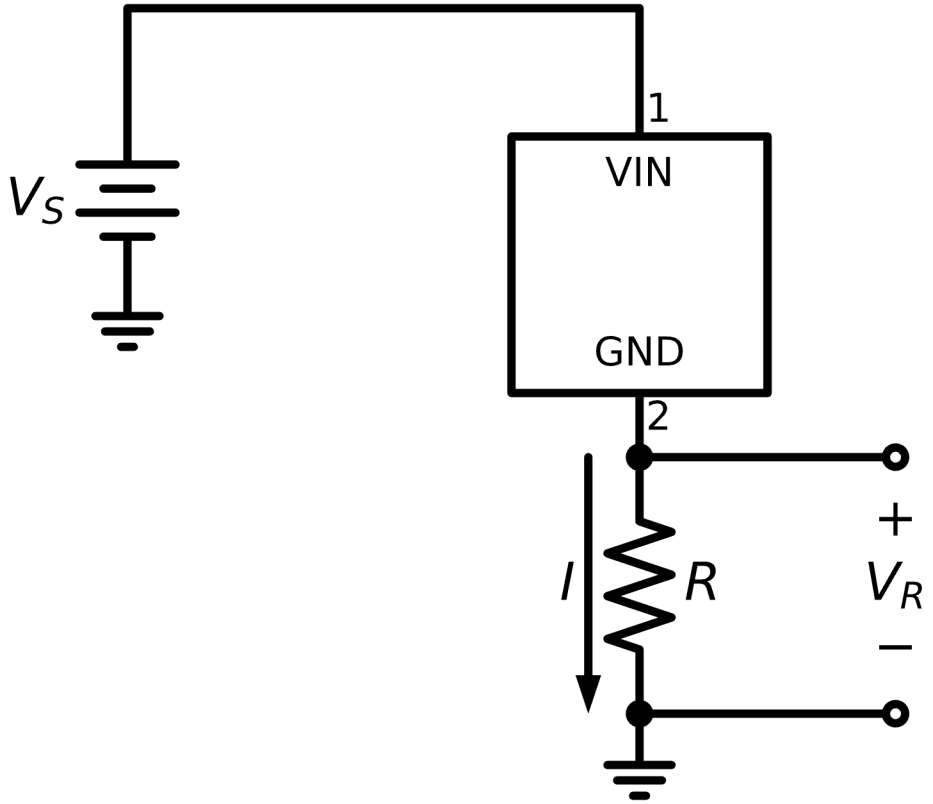
Image source: Wikipedia

Low-Side Shunt Resistor

- A single-ended probe measures voltage between a point and ground

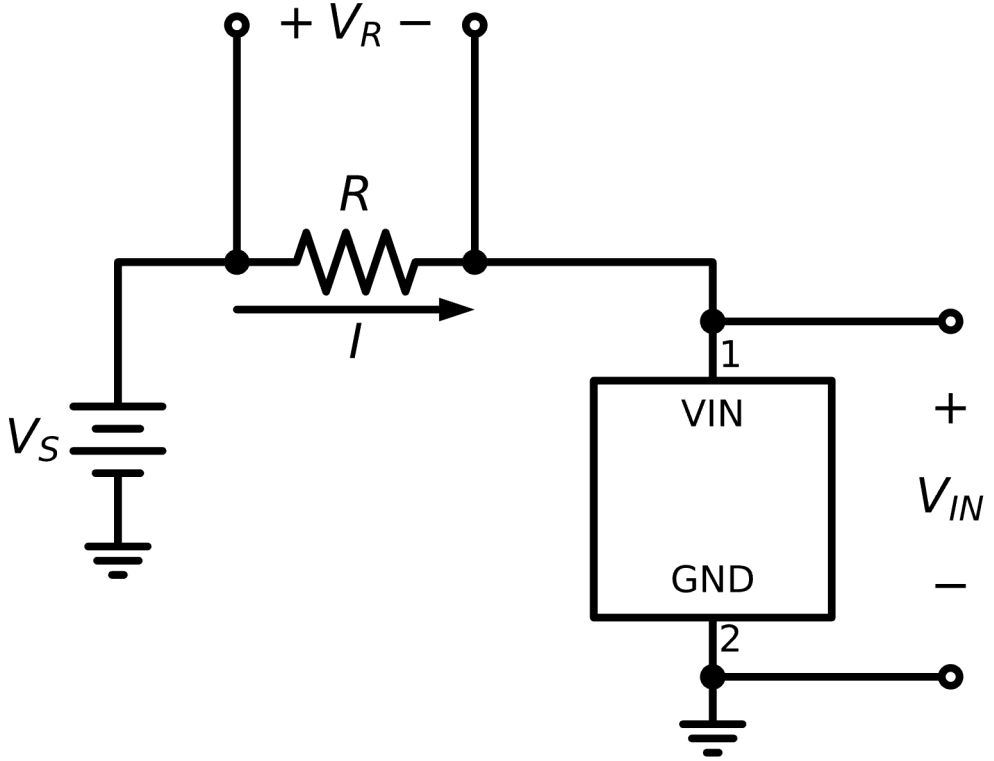
- The high-side shunt assumes that:
  - The measuring instrument does not have a common ground with measured system
  - We perform two single-ended measurements, and we subtract them
  - We use differential probes

- 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 some 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		µA
		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 µA/MHz		
		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

Example: Software-Based Estimation vs Real Measurements

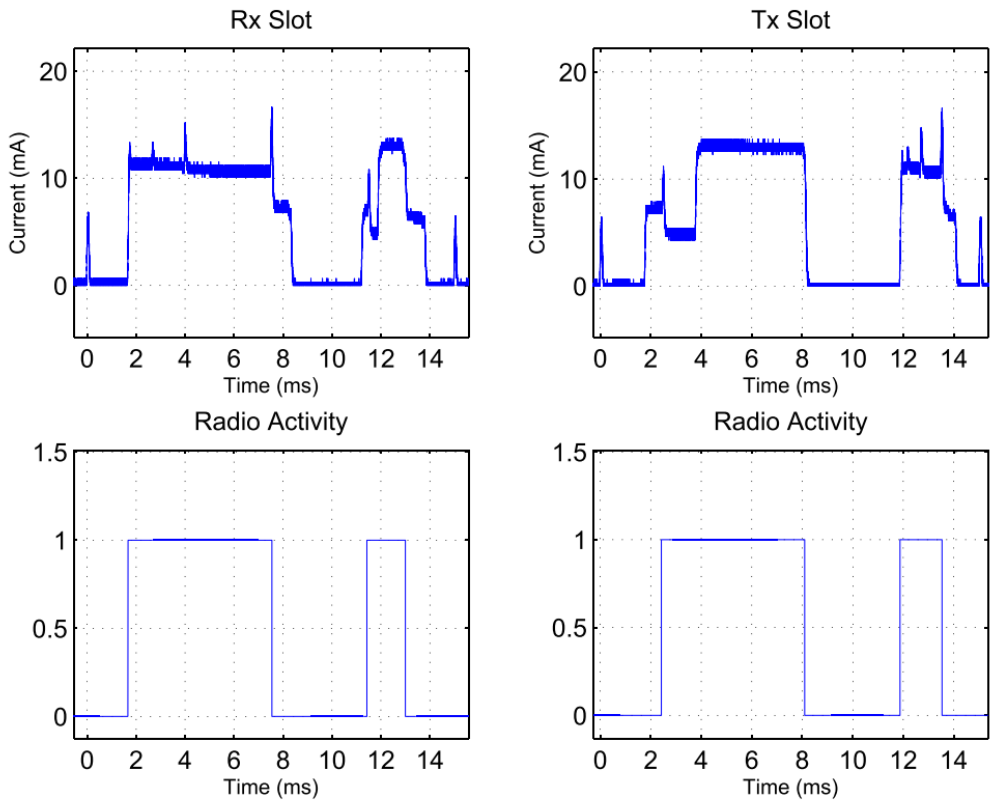


Image source: https://doi.org/10.1109/JSEN.2013.2285411