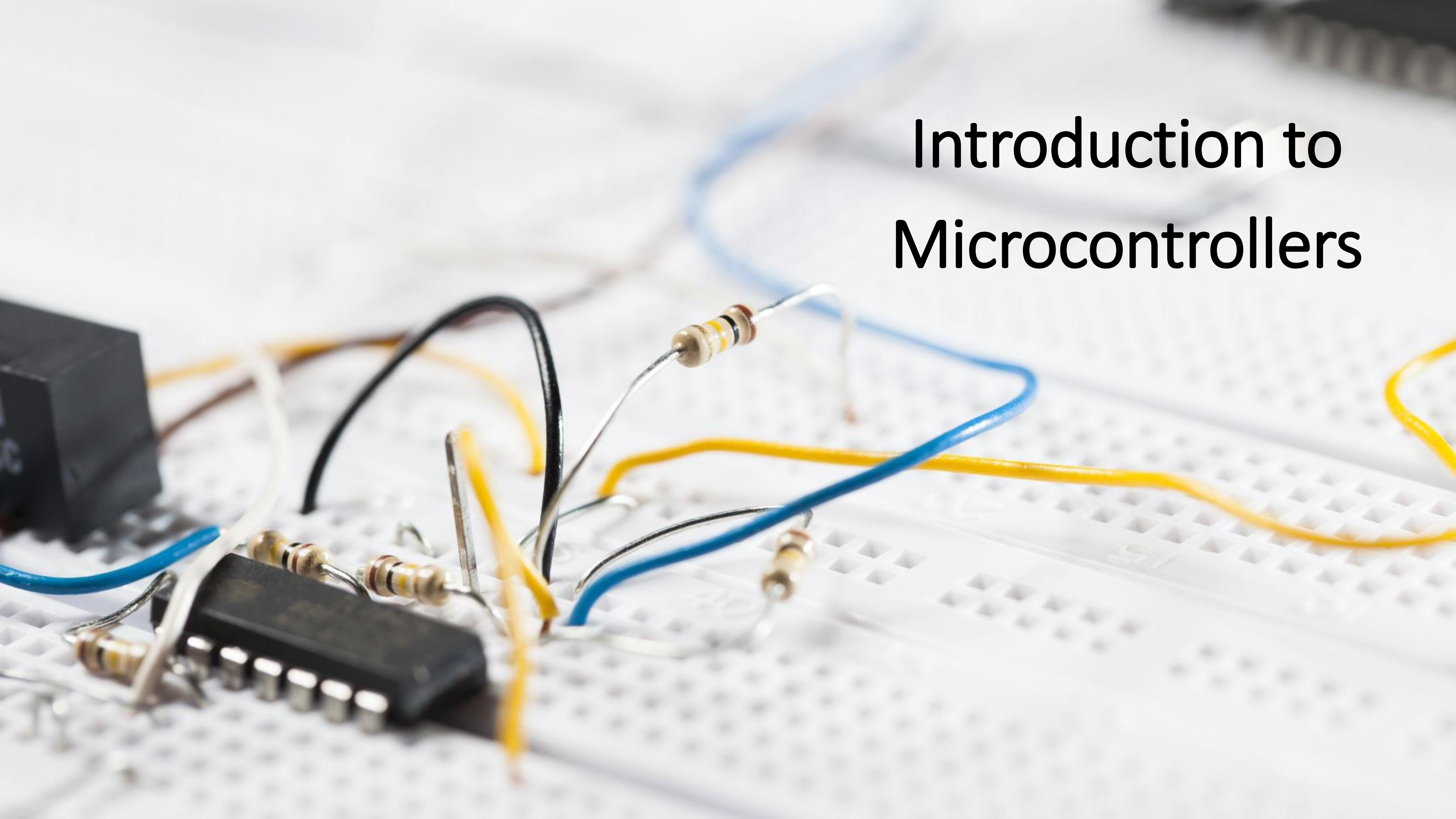
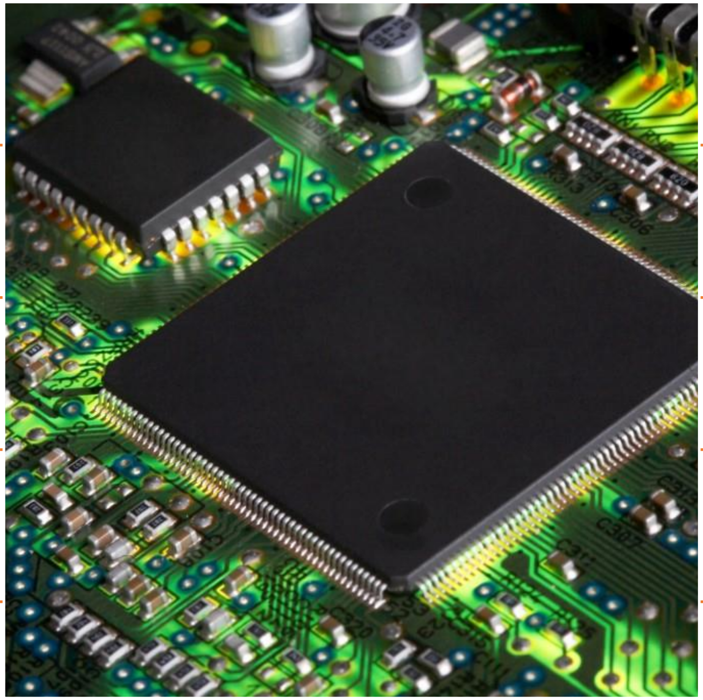
Introduction to Microcontrollers



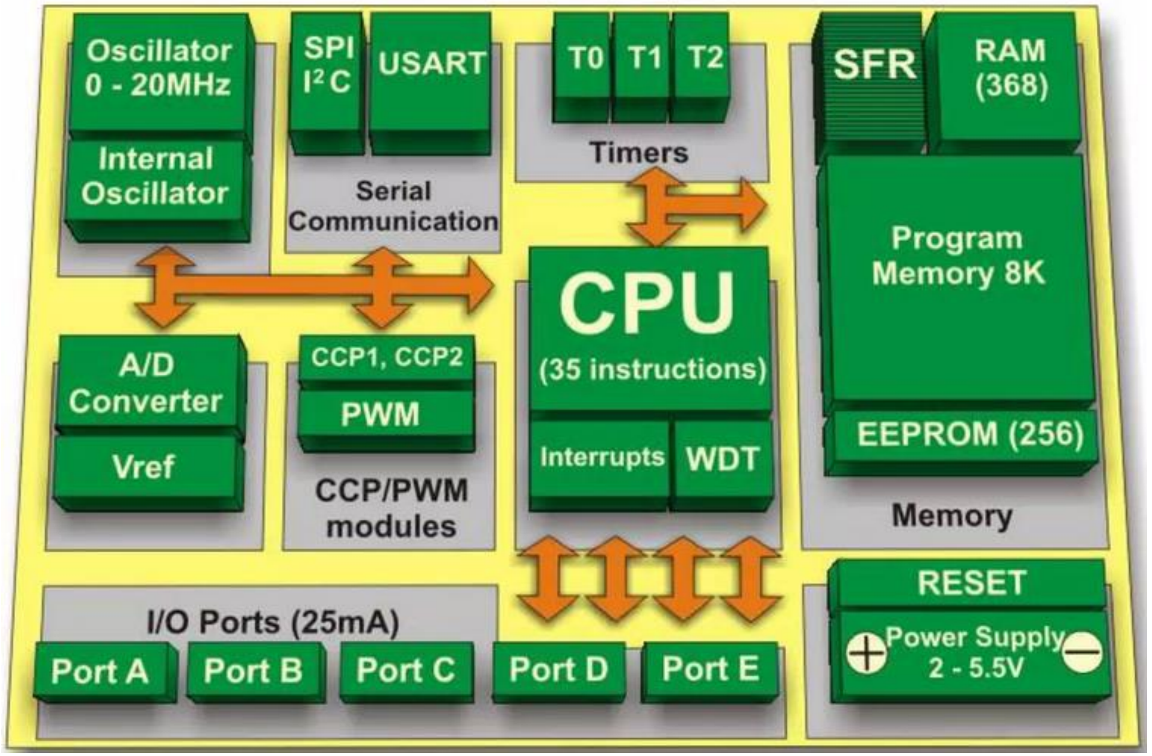
Content

Content



What is a microcontroller??

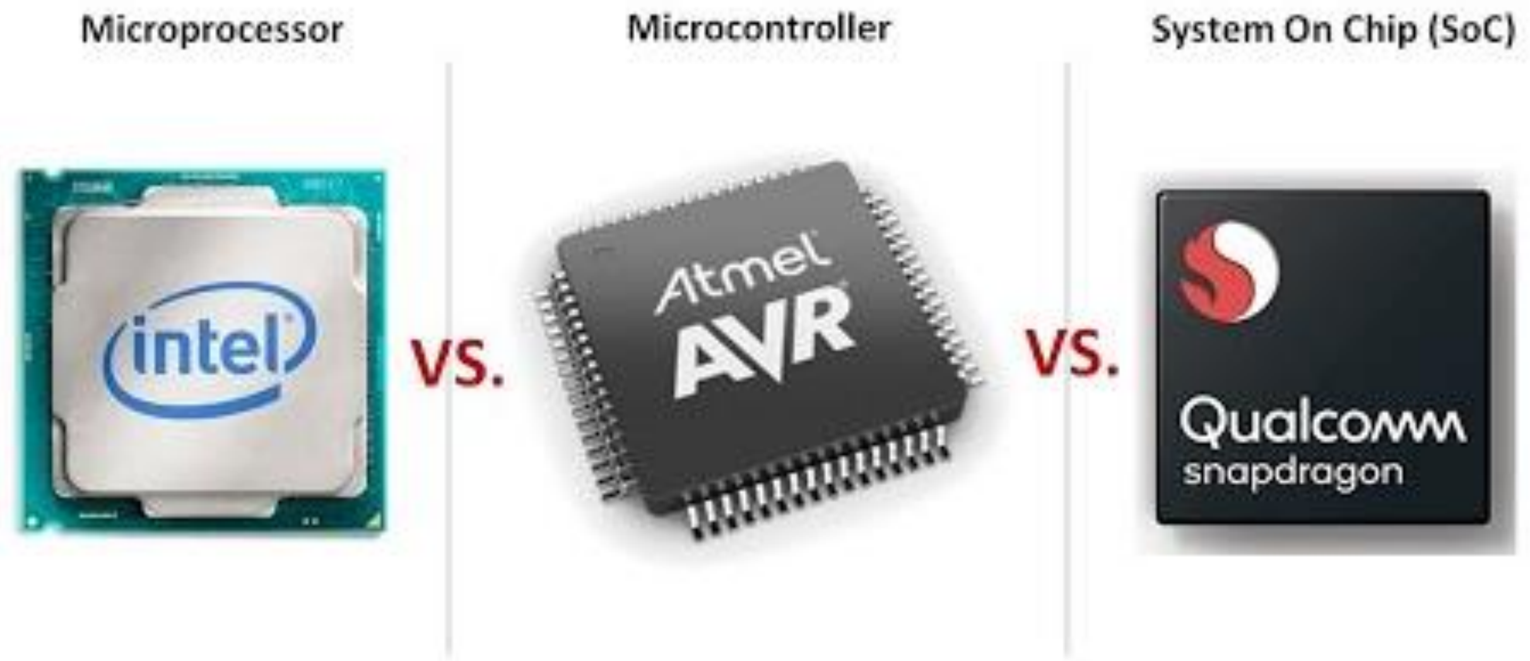
- A Microcontroller is a VLSI (Very Large Scale Integration) Integrated Circuit (IC) that contains electronic computing unit and logic unit (combinedly known as CPU), Memory (Program Memory and Data Memory), I/O Ports (Input / Output Ports) and few other components integrated on a single chip.



What is a microcontroller??

- An embedded system relies on a combination of hardware and software implementation to fulfill a specific function that imposes time constrains.

Microprocessor, Microcontroller, SoC



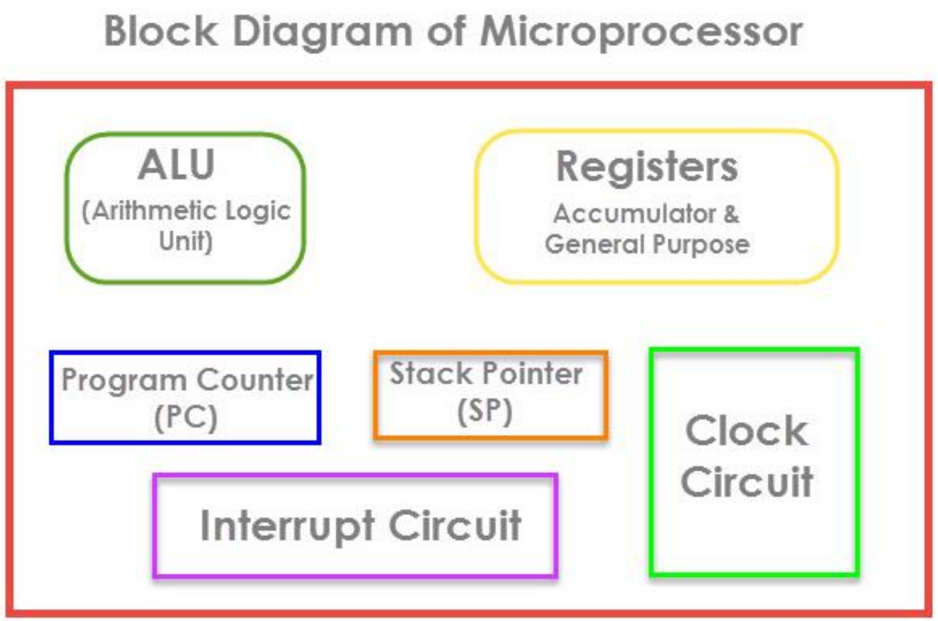
Microprocessor, Microcontroller, SoC

Microprocessor

- A Microprocessor is an Integrated Circuit (IC) that contains the Central Processing Unit (CPU).

Microprocessor, Microcontroller, SoC

- A Microprocessor is an Integrated Circuit (IC) that contains the Central Processing Unit (CPU).



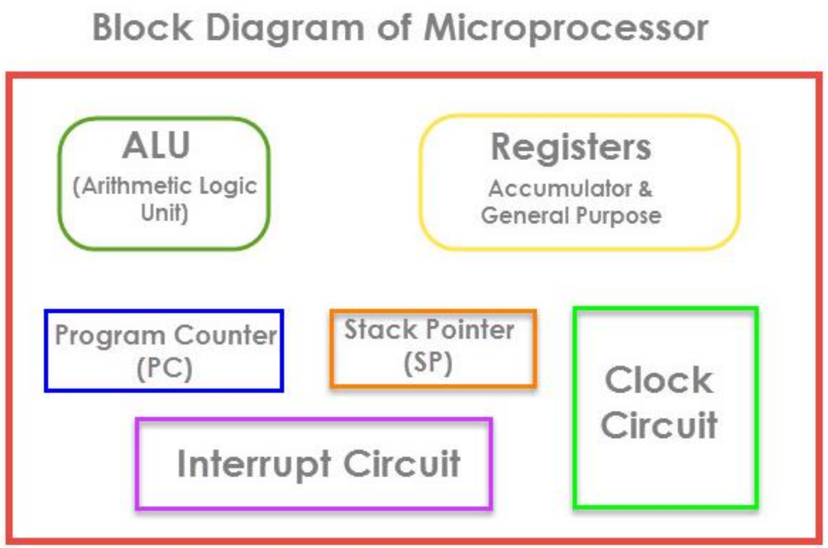
Microprocessor, Microcontroller, SoC

- It's a full computer system on a chip, even if its resources are far more limited than of a desktop personal computer.

Microcontroller

Microprocessor, Microcontroller, SoC

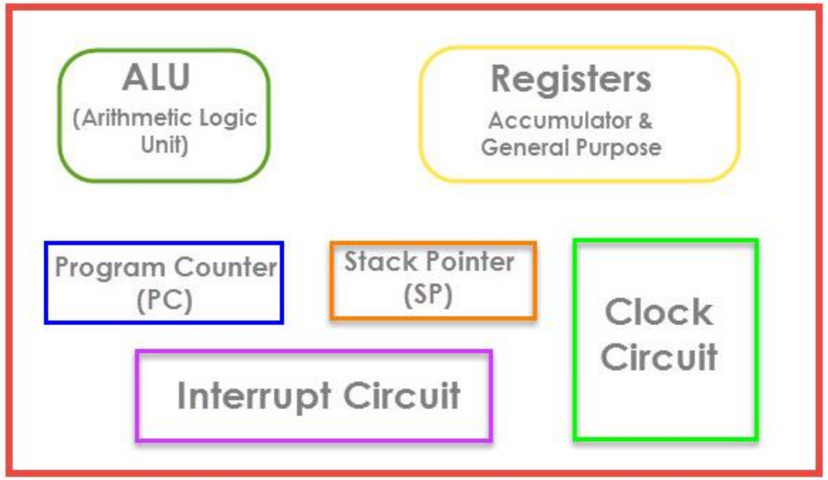
- It’s a full computer system on a chip, even if its resources are far more limited than of a desktop personal computer.



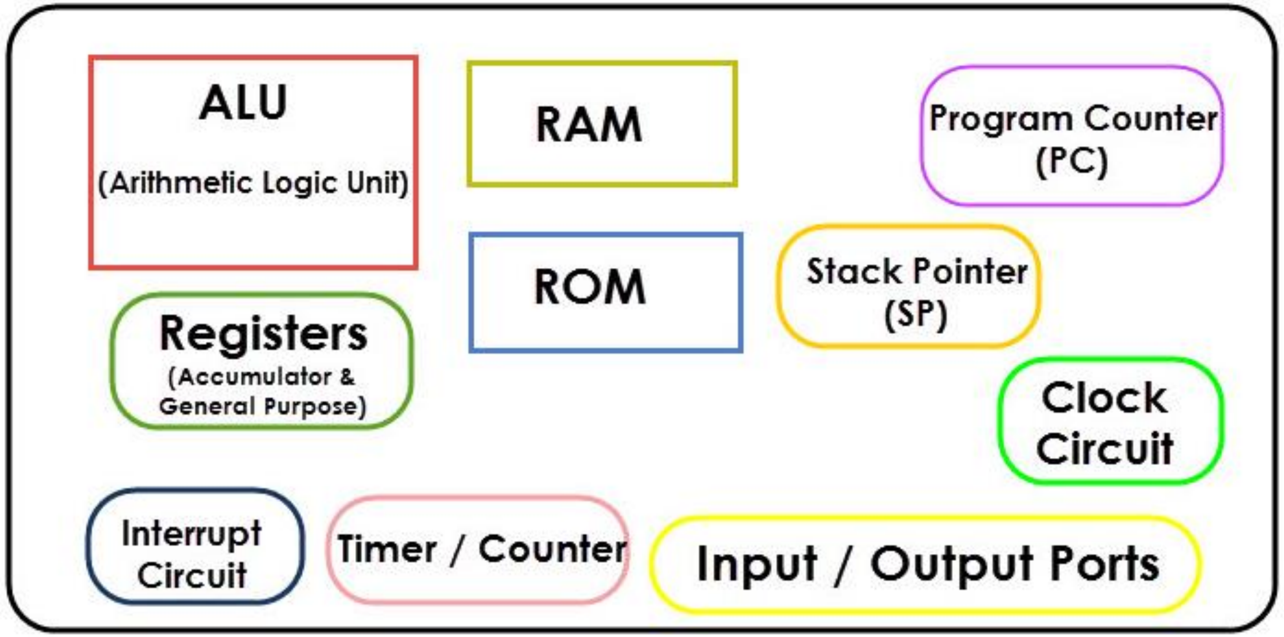
Microprocessor, Microcontroller, SoC

- It's a full computer system on a chip, even if its resources are far more limited than of a desktop personal computer.

Block Diagram of Microprocessor



Block Diagram of Microcontroller

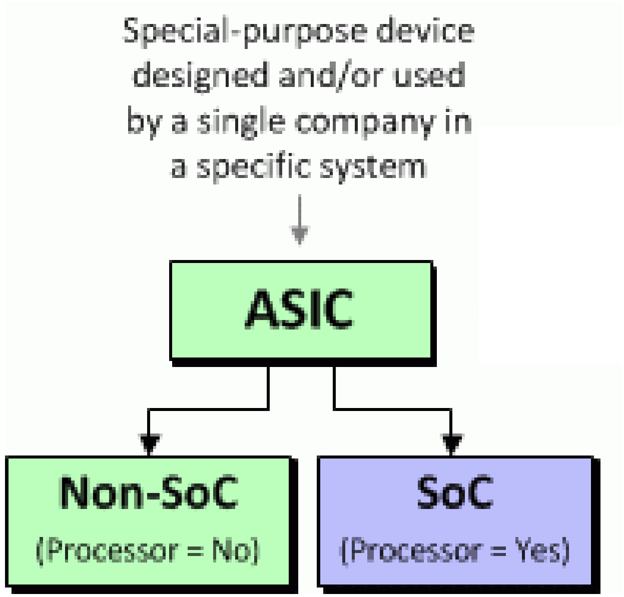


Microprocessor, Microcontroller, SoC

- A System-on-Chip (SoC) is a silicon chip that contains one or more processor cores — microprocessors (MPUs) and/or microcontrollers (MCUs) and/or digital signal processors (DSPs) — along with on-chip memory, hardware accelerator functions, peripheral functions, and (potentially) all sorts of other “stuff.”

Microprocessor, Microcontroller, SoC, ASIC

- ASIC (Application Specific Integrated Circuit) is a chip that is custom designed for a specific application. Usually designed by a company for a particular purpose or customer. This can be customized for a particular application, ensuring it meets the power and performance requirements of that specific application.



Content

What is a microcontroller?

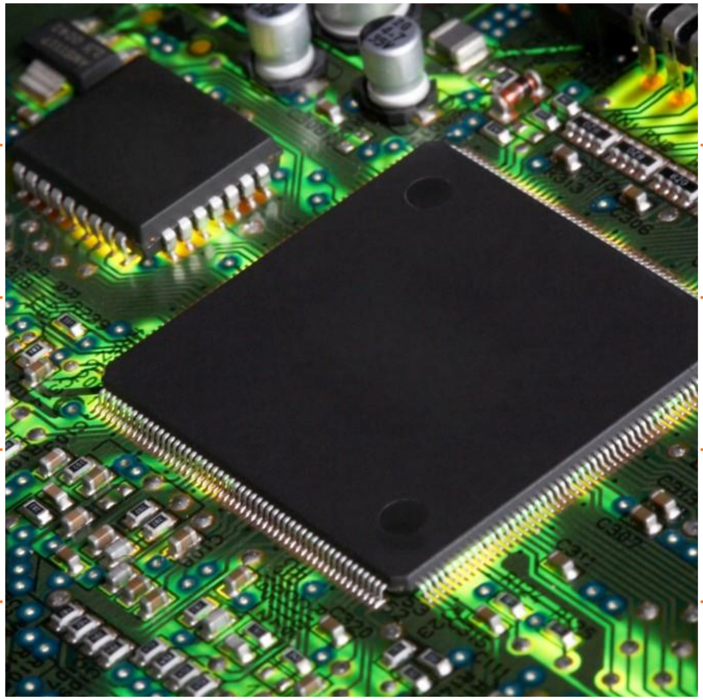
Powering up a microcontroller

Microcontroller memory

Microcontroller peripherals

Programming a microcontroller

Debugging a microcontroller code



Content

What is a microcontroller?

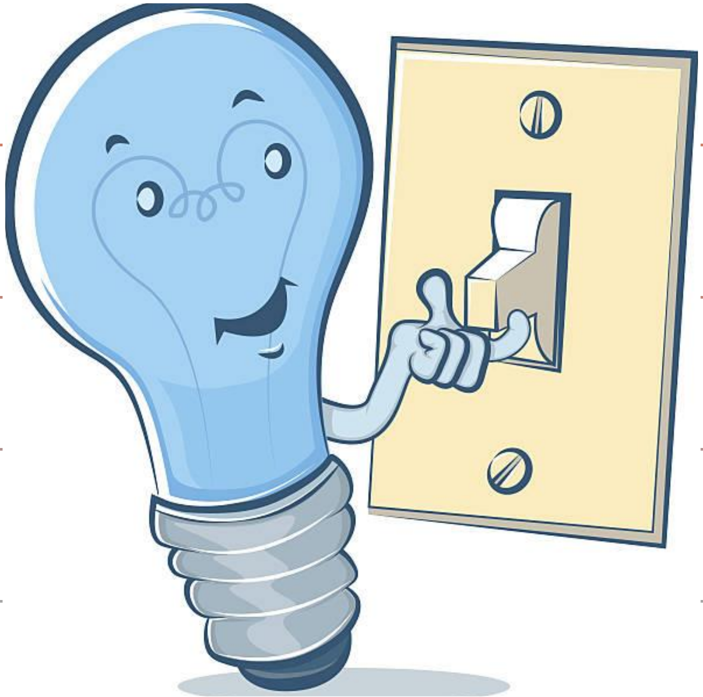
Powering up a microcontroller

Microcontroller memory

Microcontroller peripherals

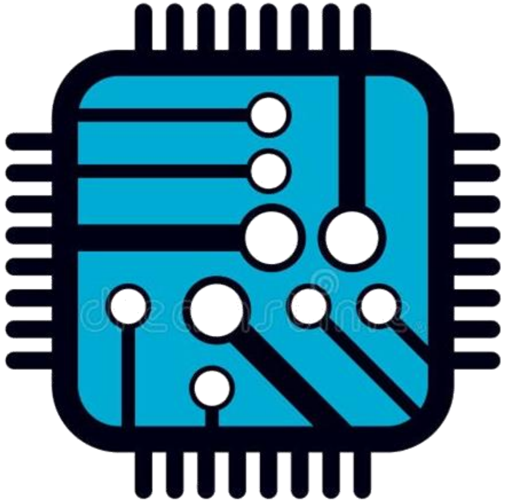
Programming a microcontroller

Debugging a microcontroller code



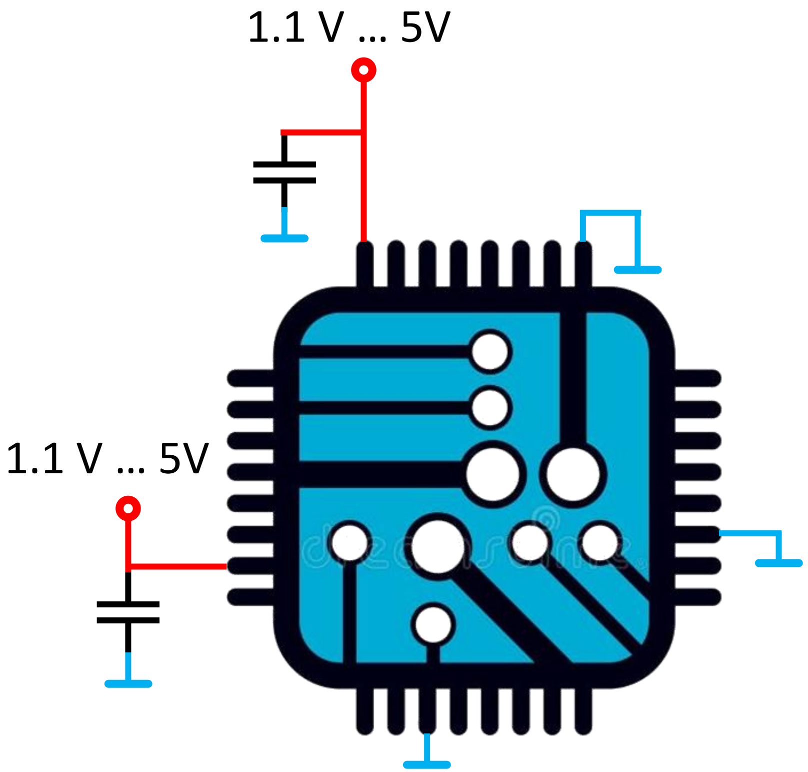
What is need to power up a microcontroller

- What is needed to start up a microcontroller?



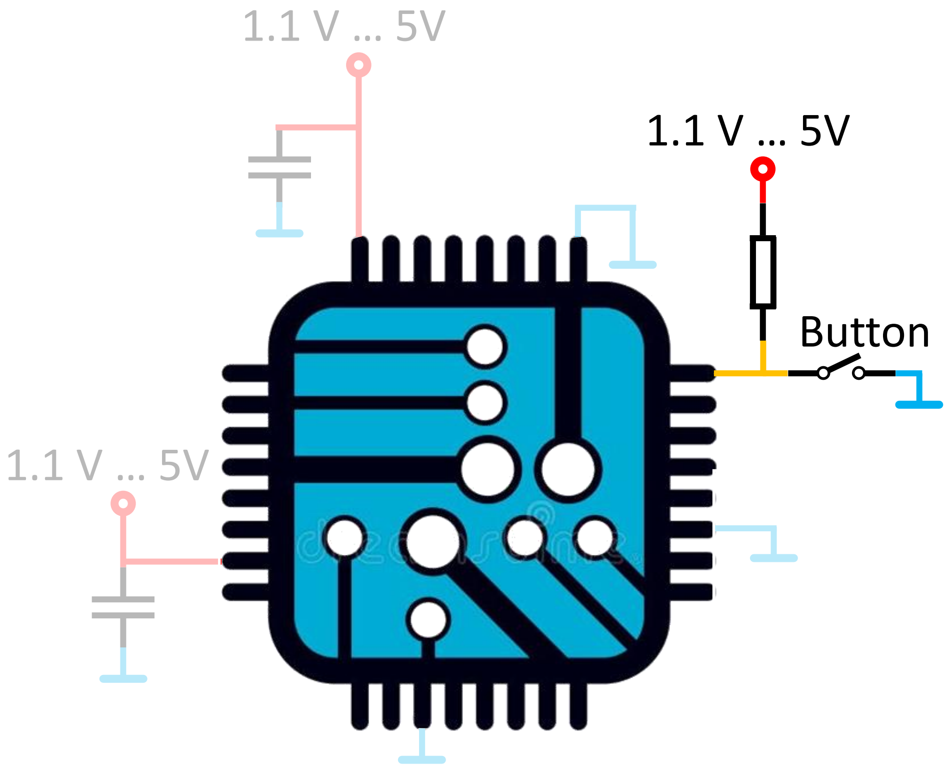
What is need to power up a microcontroller

- What is needed to start up a microcontroller?
- **Supply voltage**: depends on the microcontroller technology and is needed to power on the integrated electronics.



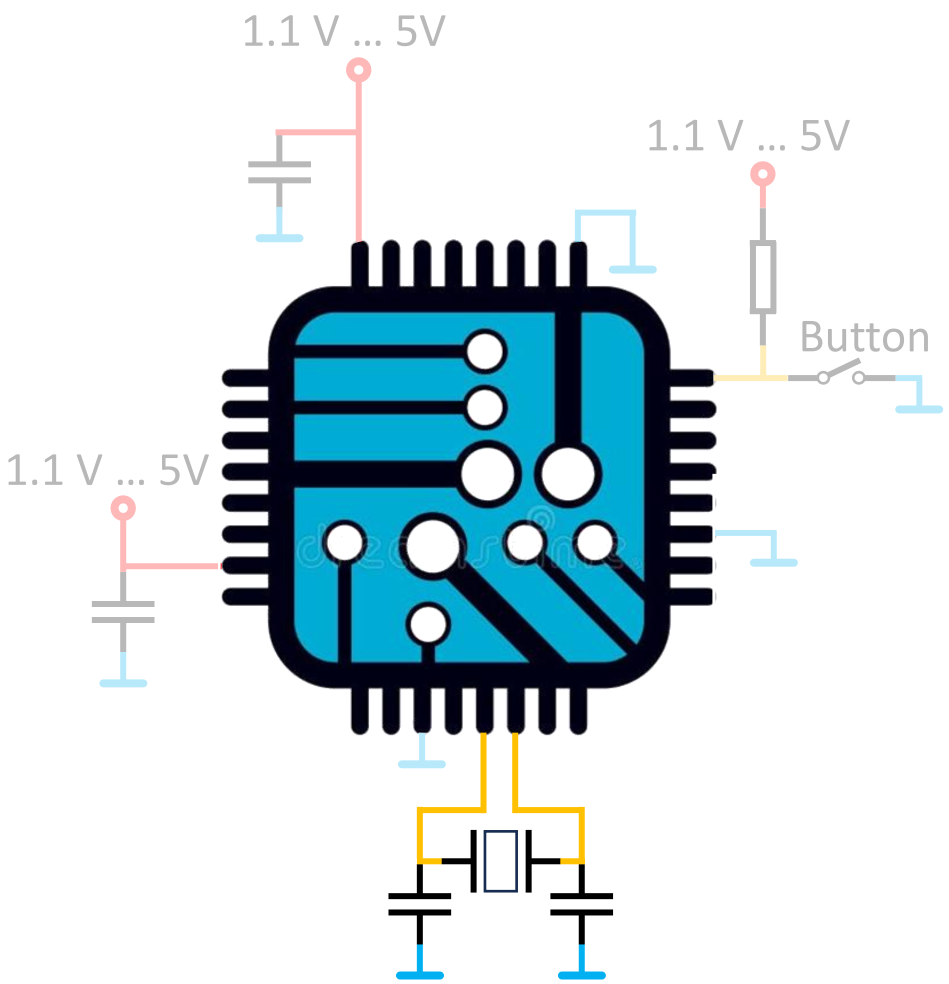
What is need to power up a microcontroller

- What is needed to start up a microcontroller?
- Supply voltage
- **Pull up the reset pin**: It is needed to wake up the microcontroller from the reset state.

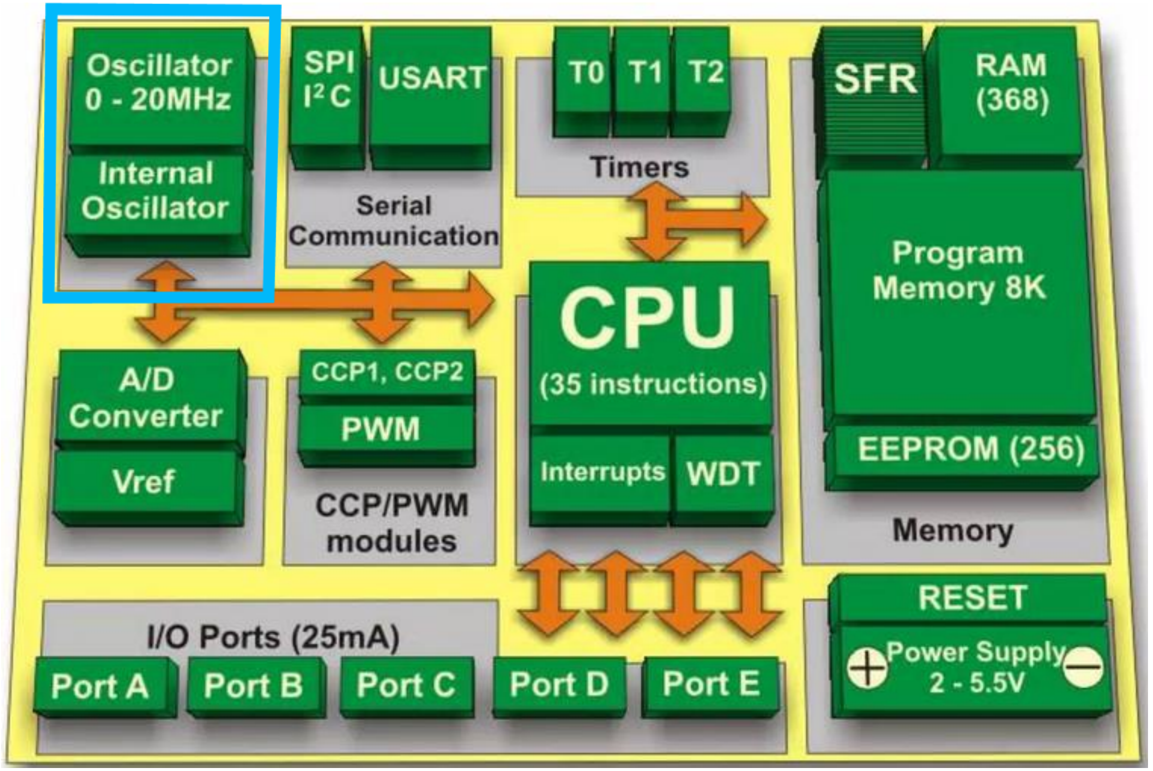


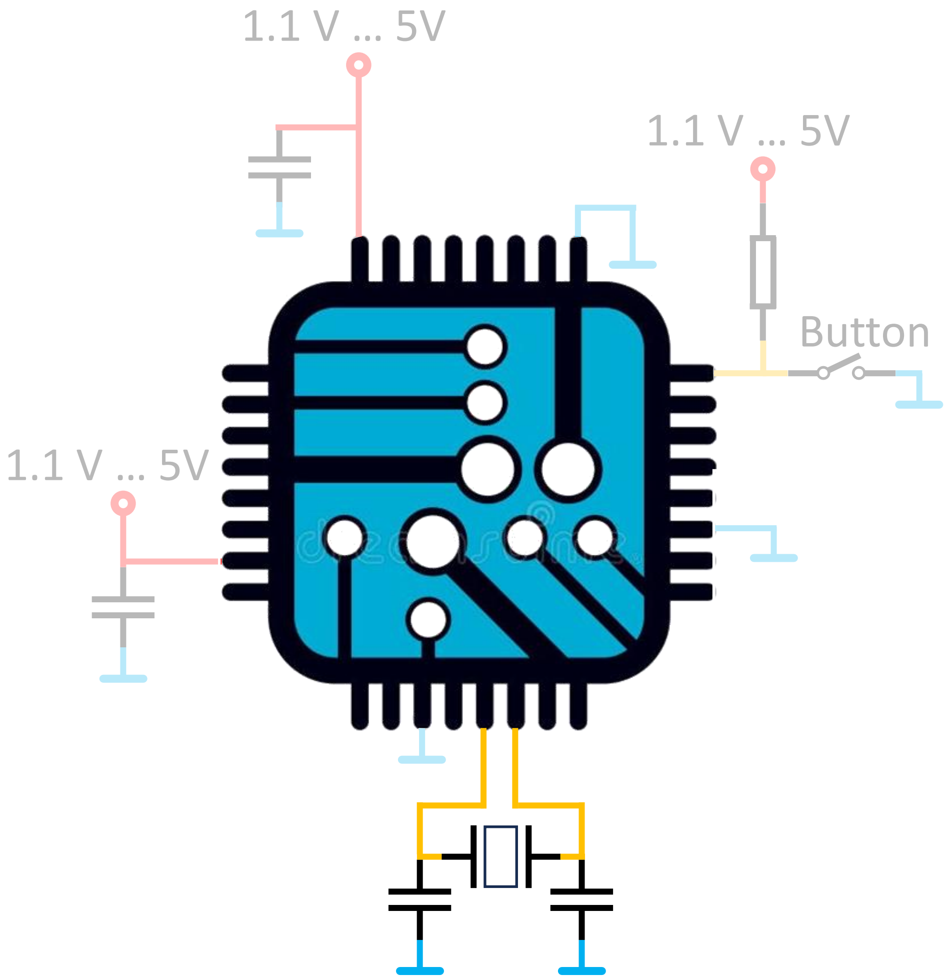
What is need to power up a microcontroller

- What is needed to start up a microcontroller?
- Supply voltage
- Pull up the reset pin
- **Microcontroller heartbeat**: Microcontrollers and microprocessors depend on oscillators for basic timing and control. Oscillators are responsible for supplying the clock signals in microcontrollers. All the instructions executed by microcontrollers are in synchronization with clock signals.



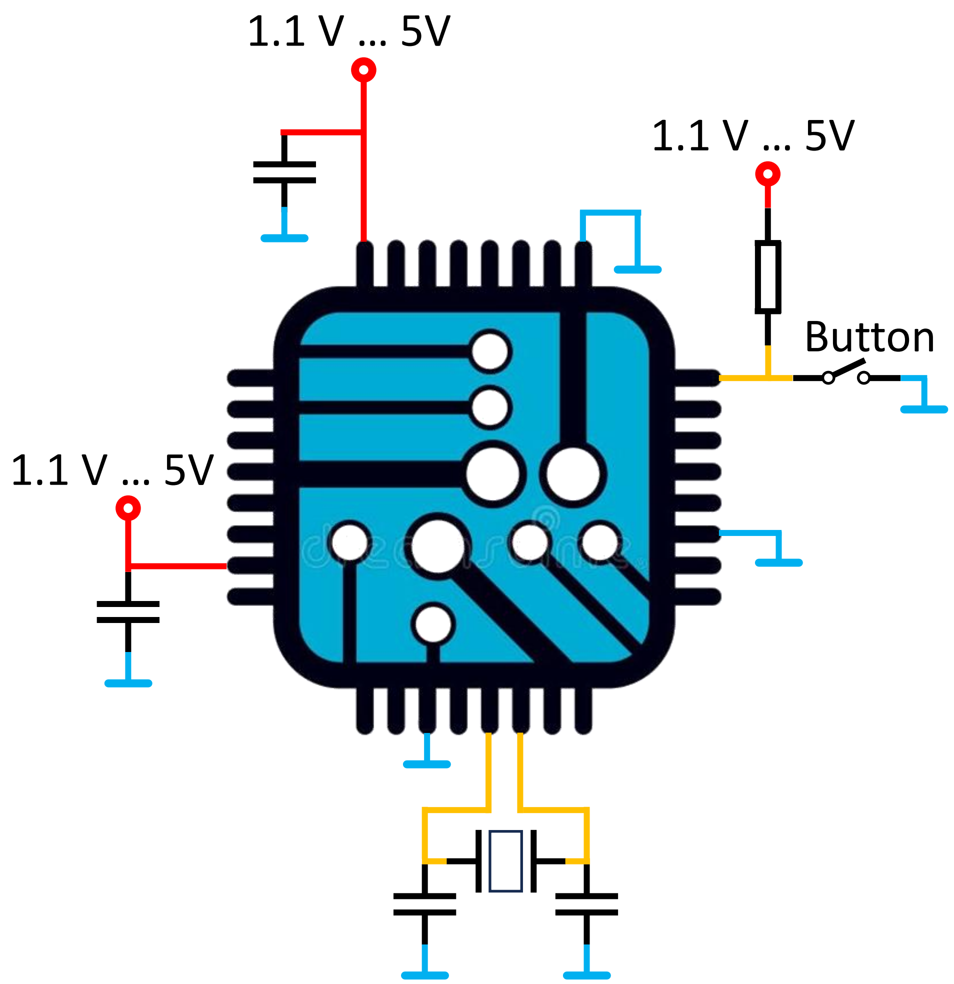
What is need to power up a microcontroller



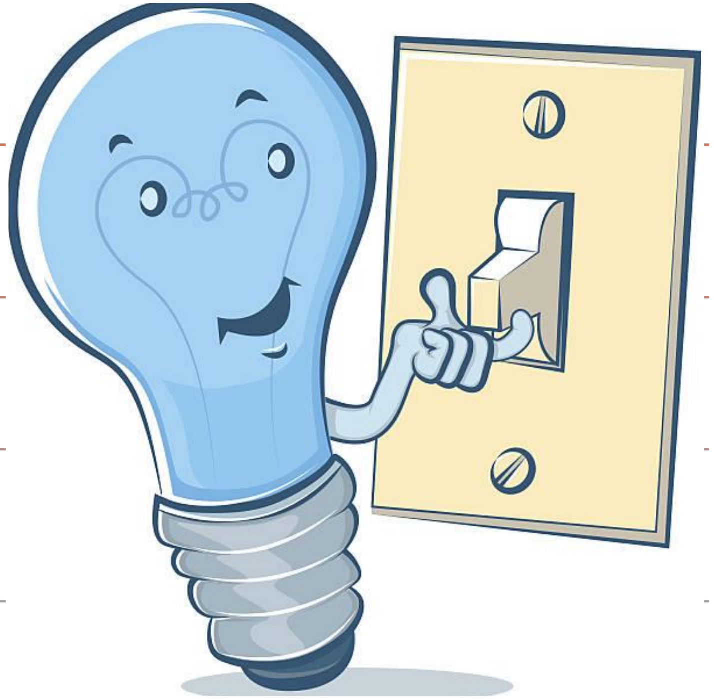


What is need to power up a microcontroller

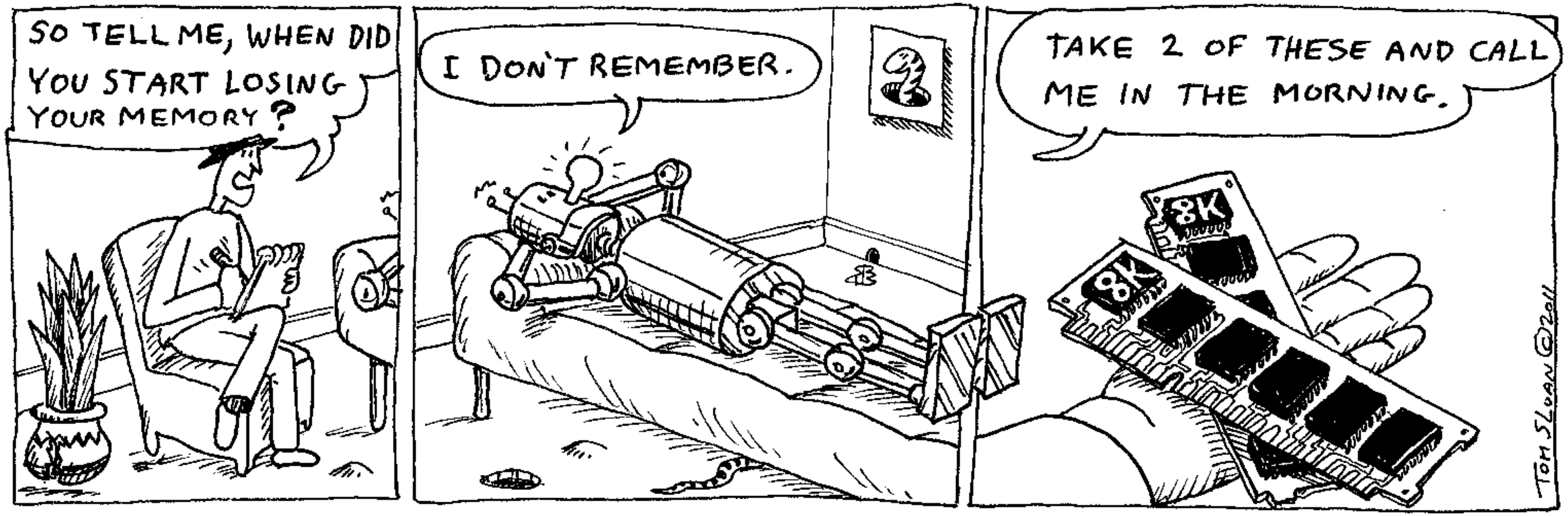
- What is needed to start up a microcontroller?
- Supply voltage
- Pull up the reset pin
- Microcontroller heartbeat
- Where is the feature that allows **it to think**?



Content

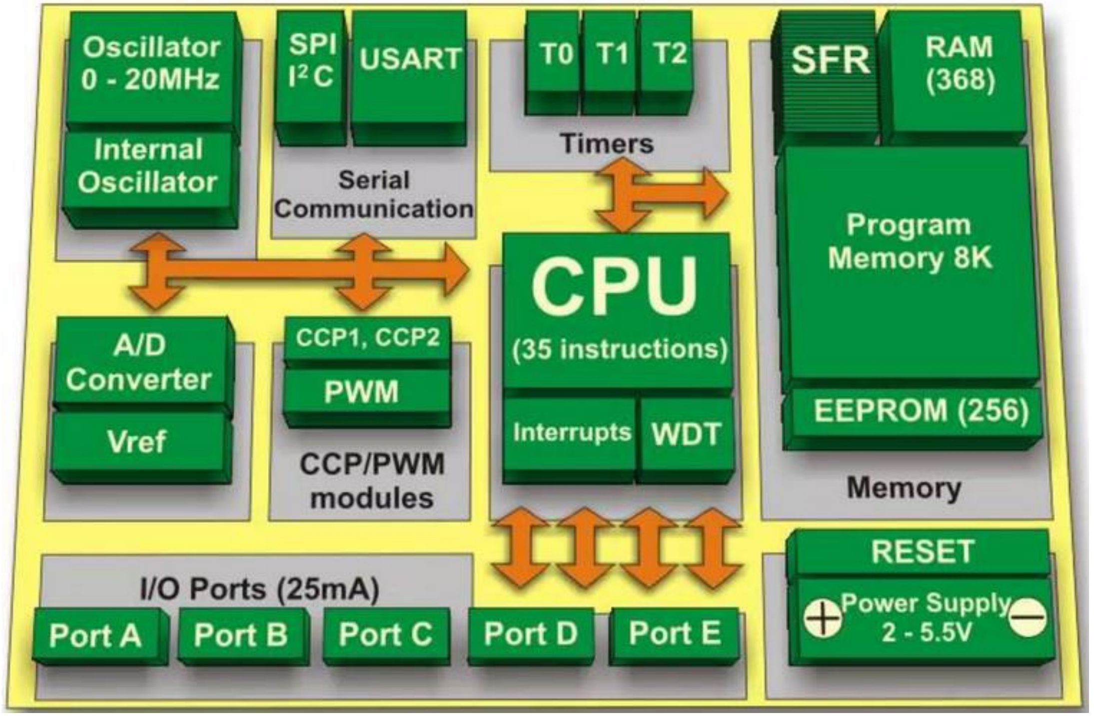


Content

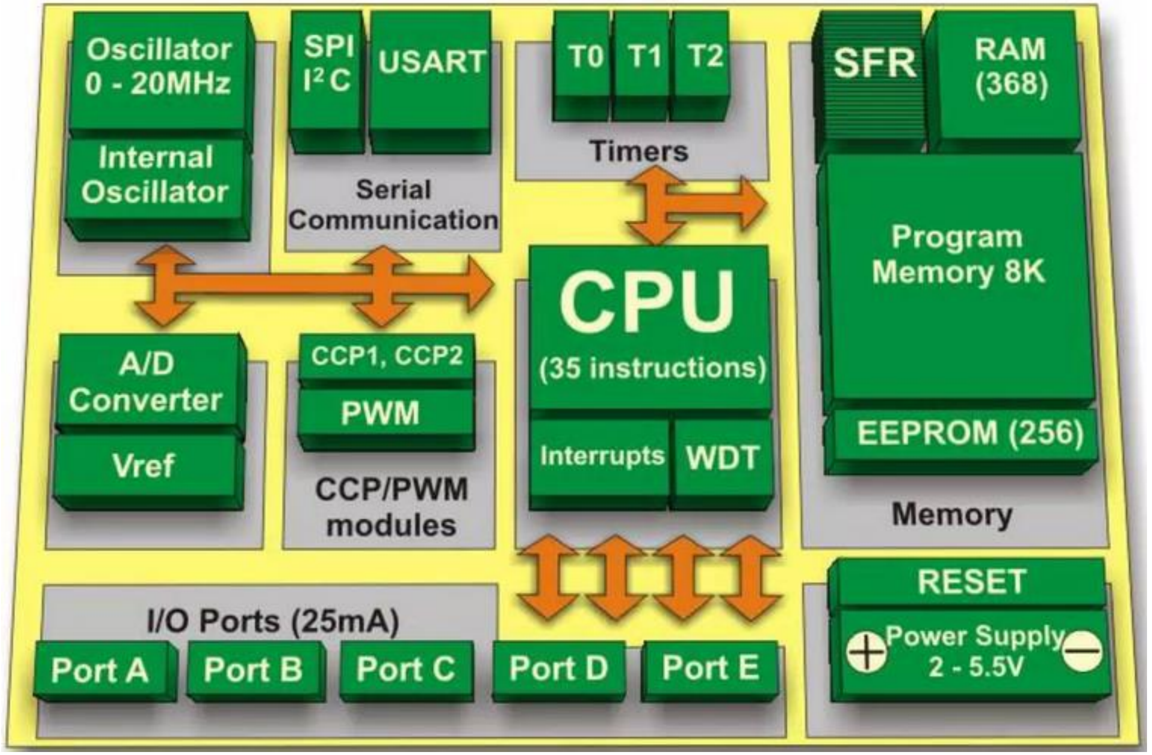

SO TELL ME, WHEN DID YOU START LOSING YOUR MEMORY?
I DON'T REMEMBER.
TAKE 2 OF THESE AND CALL ME IN THE MORNING.
8K
8K
TOM SLOAN ©2011

Microcontroller memory

	Volatile Memory	Non-Volatile Memory
Description	Loses all the data when power is lost	Retains all the data when power cycled
Device uses	Cache, Registers, Static RAM (SRAM), Dynamic RAM (DRAM)	Hard disk drives, EEPROM, Flash memory
When is used	Temporary retention of data	Permanent retention of information



Microcontroller memory - Flash

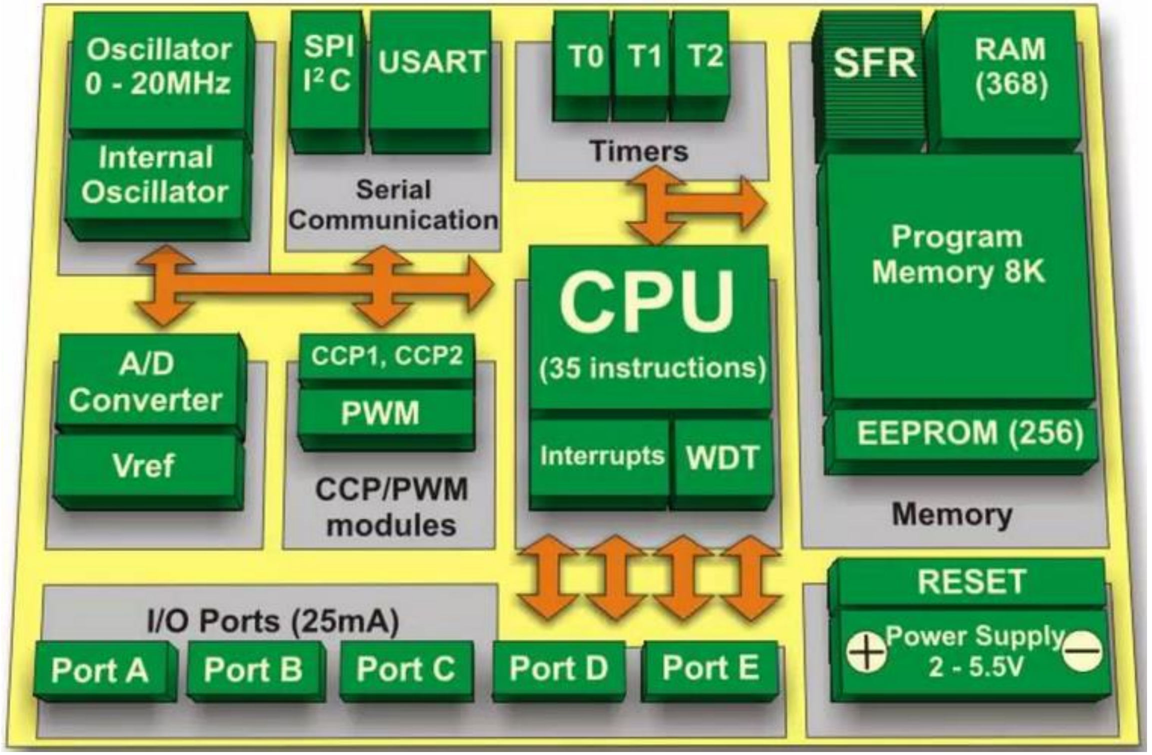


- Also know as Program Memory
- Flash memory is widely used in embedded systems due to its numerous advantages, such as its ability to maintain data without power and quick access to stored data.

Pro	Const
High-density storage	Erasing data is limited to one sector at a time
Low cost	
Fast read time	Slower write times compared to RAM
Non-volatile, retaining data without power	
Electrically reprogrammable	Finite number of write/erase cycles

Microcontroller memory - RAM

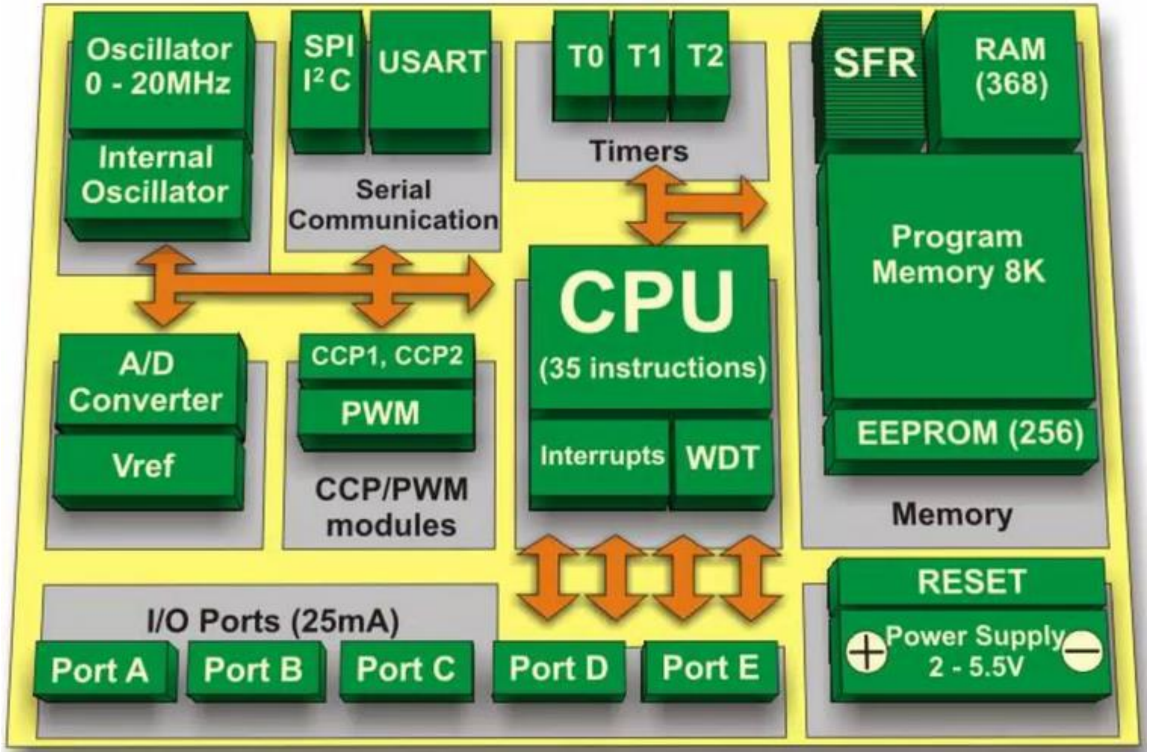
- Also know as data memory
- Static Random Access Memory (SRAM) is a type of volatile memory used in embedded systems.



Pro	Const
Greater number of write/erase cycles compared to Flash	Higher cost-per-byte compared to DRAM and FLASH
Fast access times (very fast read/write speed)	Consumes more power than DRAM, and even more then FLASH
No refresh cycles required, unlike DRAM	Requires more transistors per memory cell, resulting in a larger chip size
Smallest write/read size (byte level)	

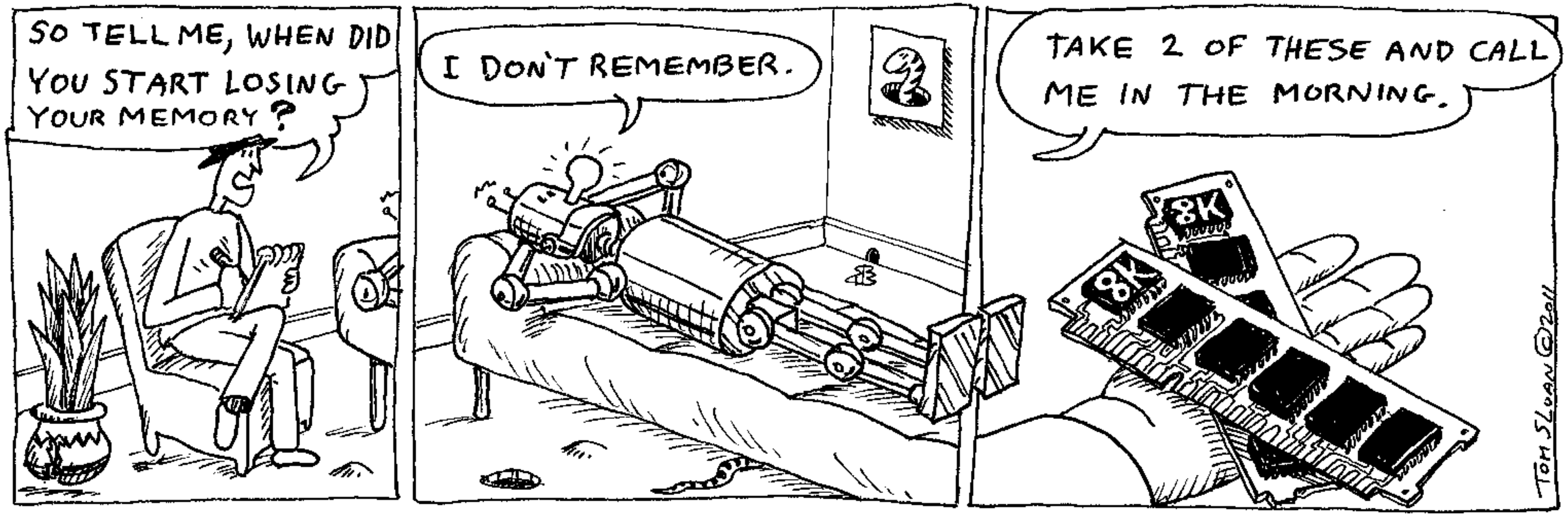
Microcontroller memory - EEPROM

- Electrically-Erasable-Programmable Read-Only Memory (EEPROM) is a hybrid memory device that combines features of both RAM and ROM (Read Only Memory).



Pro	Const
Greater number of write/erase cycles compared to Flash	Higher cost-per-byte compared to DRAM and FLASH
Fast access times (very fast read/write speed)	Consumes more power than DRAM, and even more then FLASH
No refresh cycles required, unlike DRAM	Requires more transistors per memory cell, resulting in a larger chip size
Smallest write/read size (byte level)	

Content



What is a micr

Powering up a

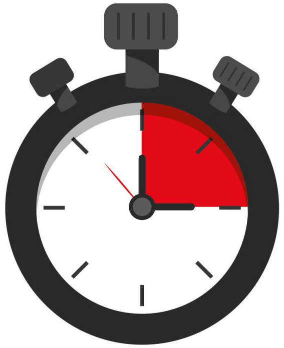
Microcontroller memory

Microcontroller peripherals

Programming a microcontroller

Debugging a microcontroller code

Content



What is a microcontroller?

Powering up a microcontroller

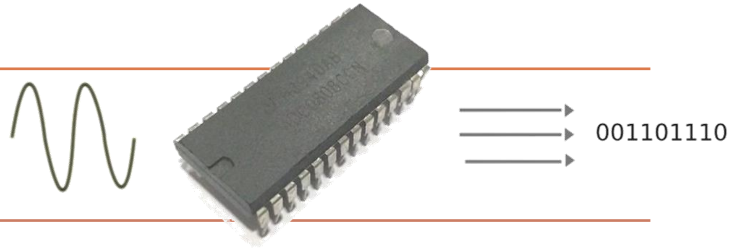
Microcontroller memory

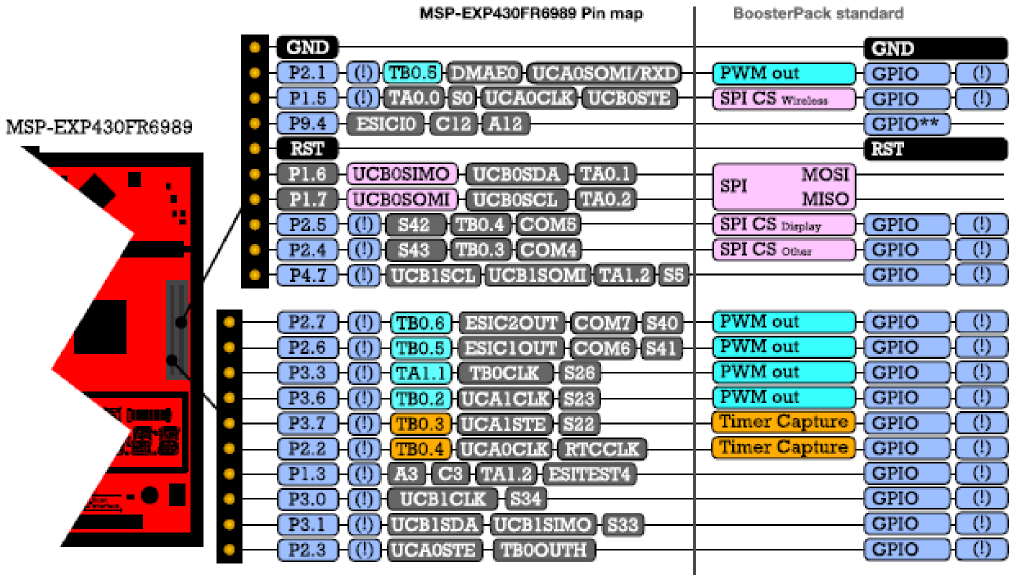
Microcontroller peripherals

Programming a microcontroller

Debugging a microcontroller code

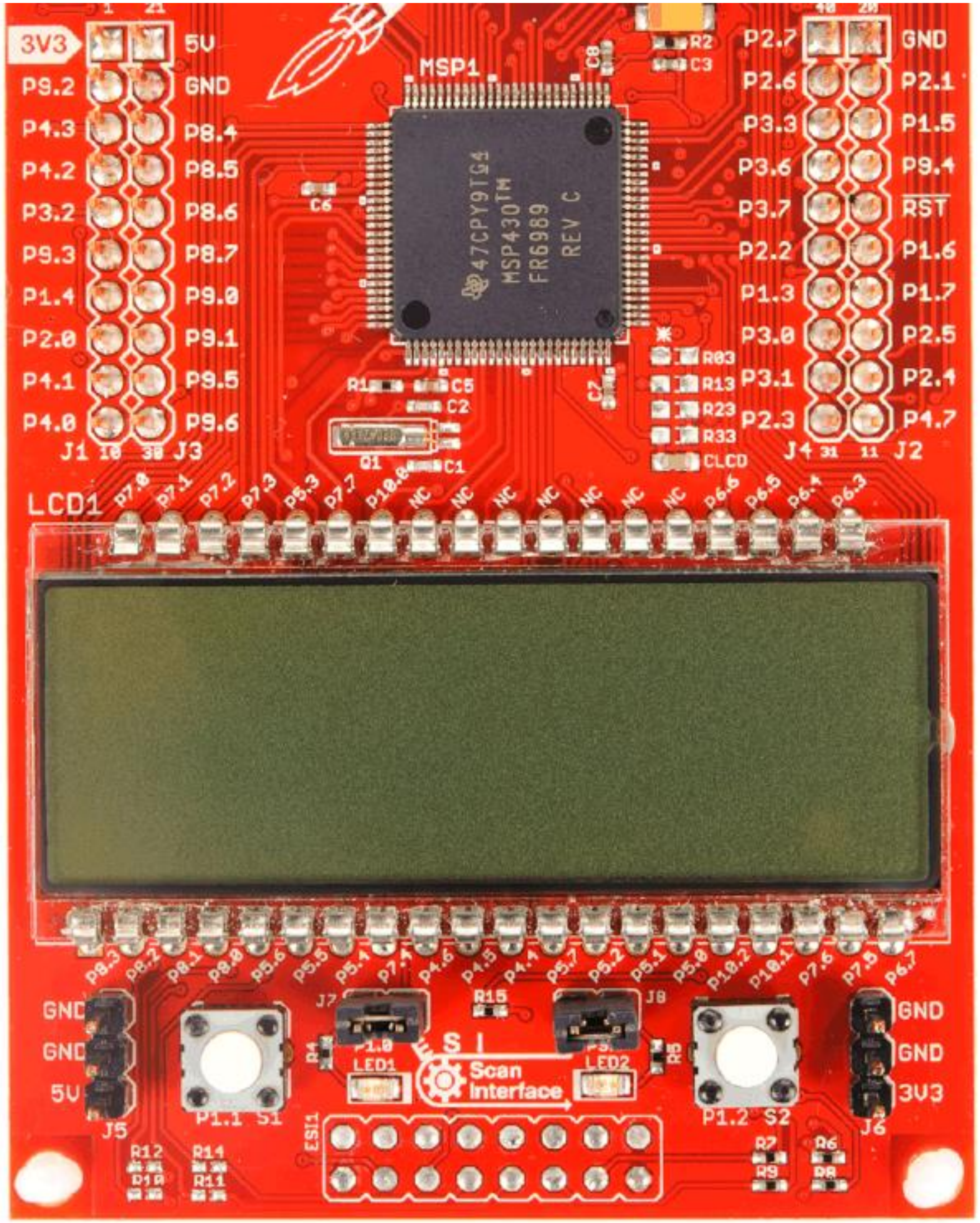
Analog to Digital Converters





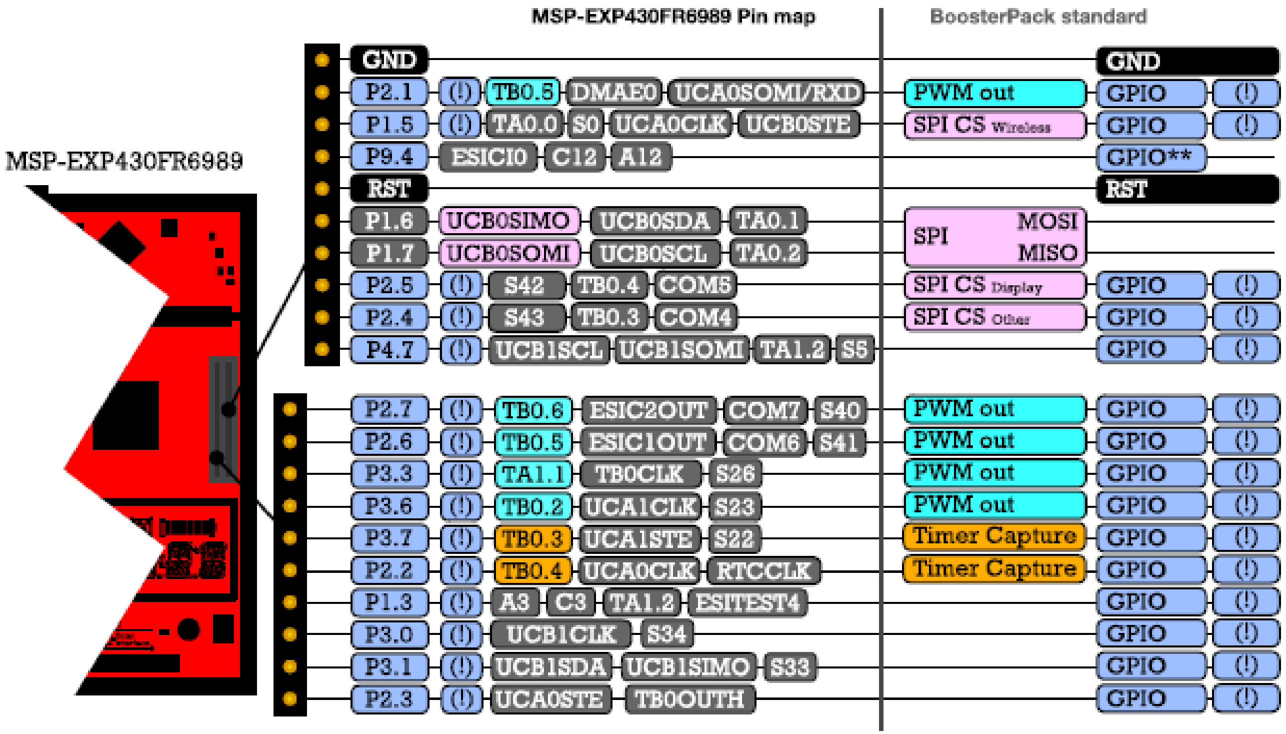
Microcontroller peripherals - GPIO

- GPIO stands for General Purpose Input/Output pin.
- General-Purpose Input/Output pins are used for simple “on”/”off” communication, such as reading a button or turning on an LED.
- As an input, a GPIO pin tells the microcontroller what voltage is present on the pin (high or low voltage).
- As an output, the microcontroller chooses to set the GPIO pin to output either a high or low voltage.



Microcontroller peripherals - GPIO

- GPIO stands for General Purpose Input/Output pin.
- General-Purpose Input/Output pins are used for simple “on”/”off” communication, such as reading a button or turning on an LED.
- As an input, a GPIO pin tells the microcontroller what voltage is present on the pin (high or low voltage).
- As an output, the microcontroller chooses to set the GPIO pin to output either a high or low voltage.



Microcontroller peripherals - Timers

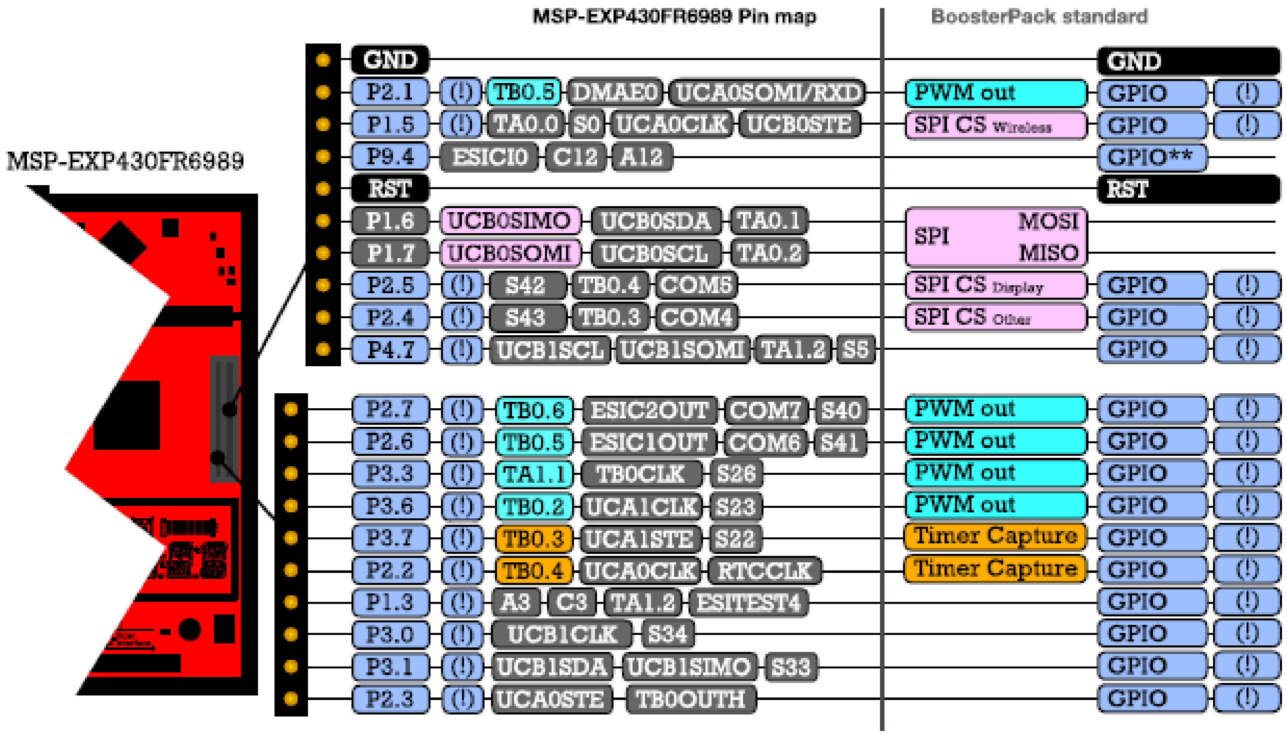
- Timing is a crucial part of any embedded system, be it controlling the blinking rate of the LEDs or controlling the sampling rate of the ADCs, or a simple delay on the source code.

- Timers can be used to keep track of time (a timer can be set to “tick” every 1ms for example), and counters can be used to count pulses on an external pin for example.

Microcontroller peripherals - Timers

- Timing is a crucial part of any embedded system, be it controlling the blinking rate of the LEDs or controlling the sampling rate of the ADCs, or a simple delay on the source code.

- Timers can be used to keep track of time (a timer can be set to “tick” every 1ms for example), and counters can be used to count pulses on an external pin for example.

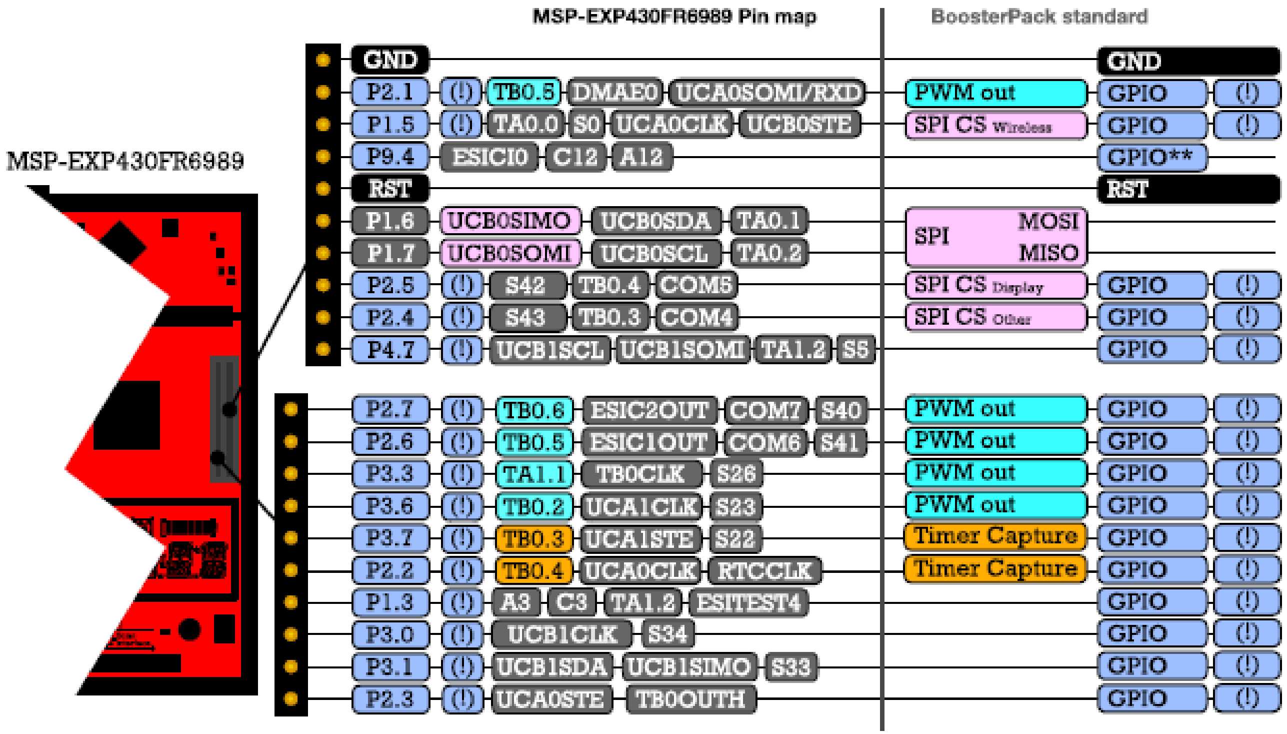


Microcontroller peripherals - ADC

- ADCs are used to read an analog voltage and convert it into a digital number which the microprocessor can understand.
- These ADCs are devices that can sense the voltage at a given GPIO pin. It takes an analog voltage and converts it to a digital number.

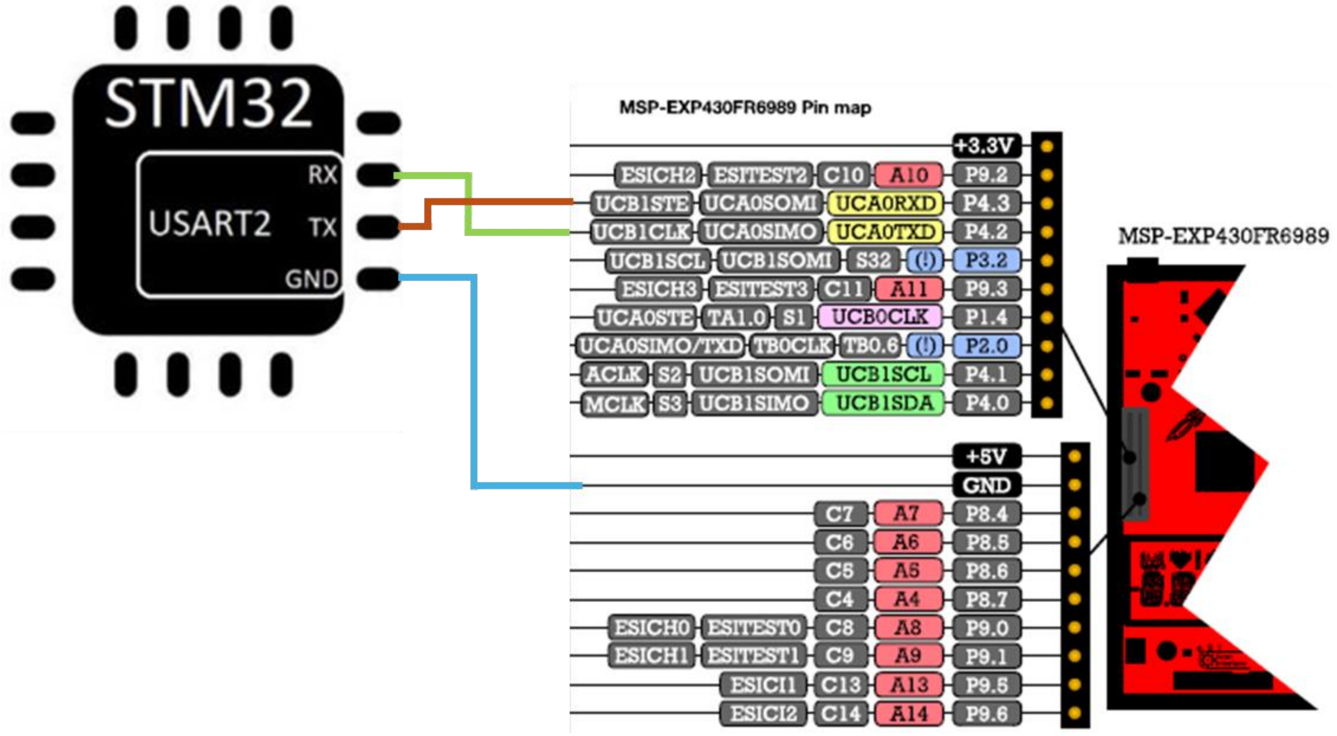
Microcontroller peripherals - ADC

- ADCs are used to read an analog voltage and convert it into a digital number which the microprocessor can understand.
- These ADCs are devices that can sense the voltage at a given GPIO pin. It takes an analog voltage and converts it to a digital number.



Microcontroller peripherals - UART

- To talk to the external peripherals, some sort of communication protocol is needed. This is taken care of using devices called serial communication controllers.
- One of the earliest communication protocols was UART (Universal Asynchronous Receiver and Transmitter).
- Peripherals are typically separate pieces of circuitry which offload work from the microprocessor.



Microcontroller peripherals – Interrupt controllers

- Interrupt controllers listen to the peripherals for events and reports to the processor once an event occurs.

- Examples of events that can produce interrupts include:
  - GPIO reads 1 or 0
  - Timer countdown reached 0
  - Serial communication received a packet of data
  - ADC conversion has ended.

Content

What is a microcontroller?

Powering up a microcontroller

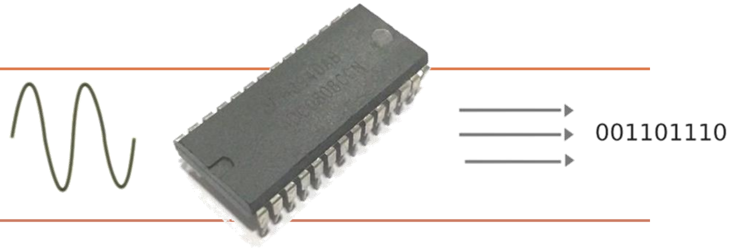
Microcontroller memory

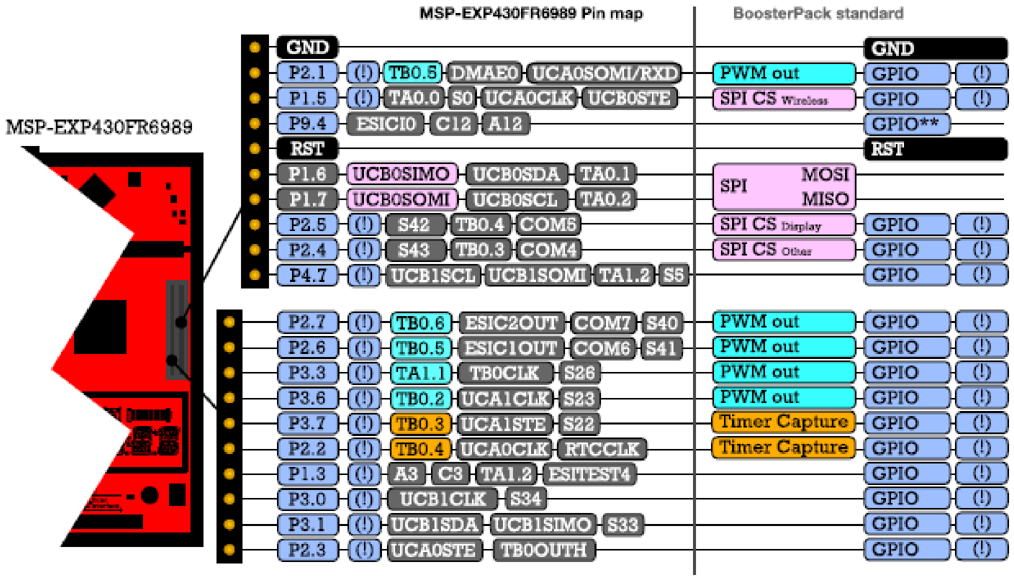
Microcontroller peripherals

Programming a microcontroller

Debugging a microcontroller code

Analog to Digital Converters





Content

What is a microcontroller?

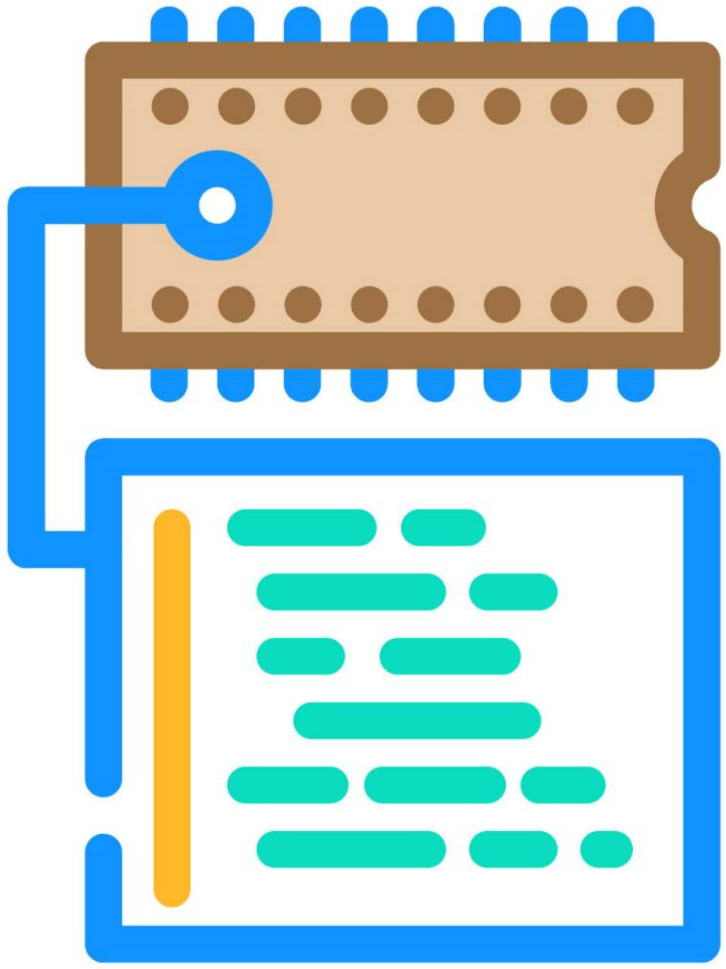
Powering up a microcontroller

Microcontroller memory

Microcontroller peripherals

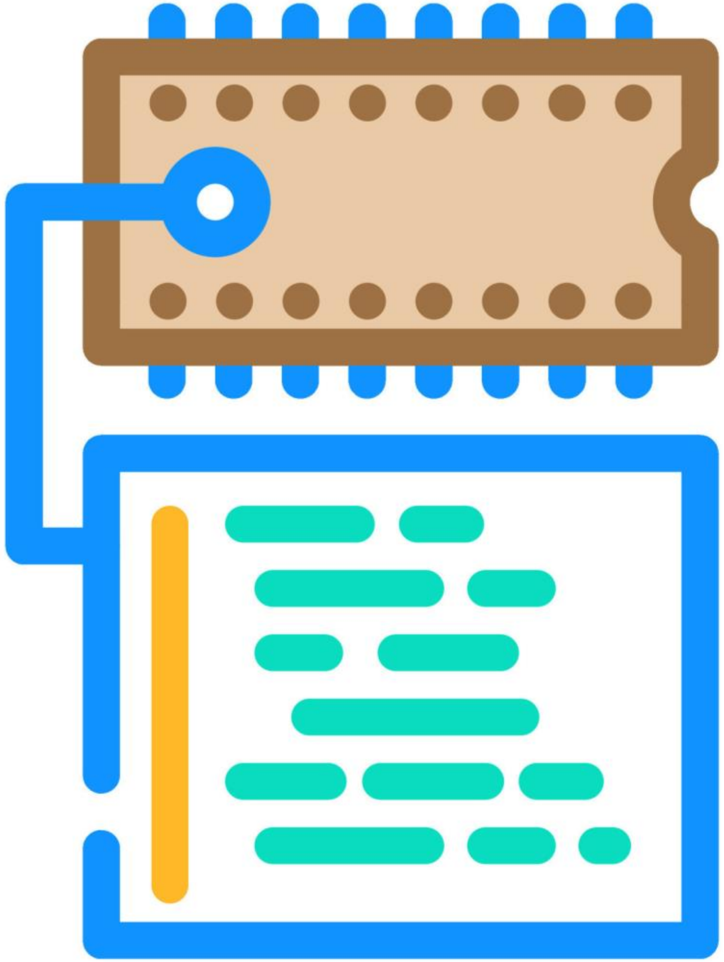
Programming the microcontroller

Debugging a microcontroller code



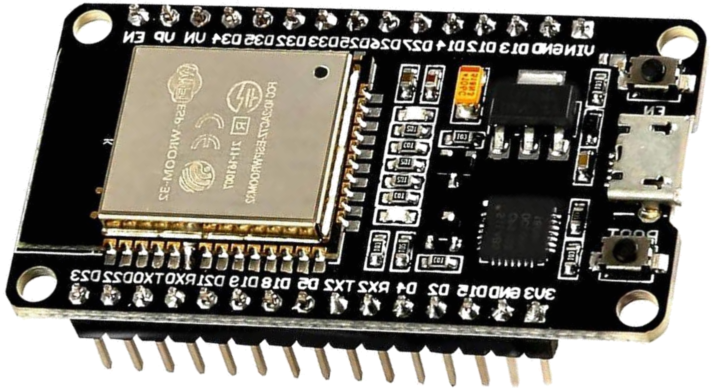
Programming the microcontroller

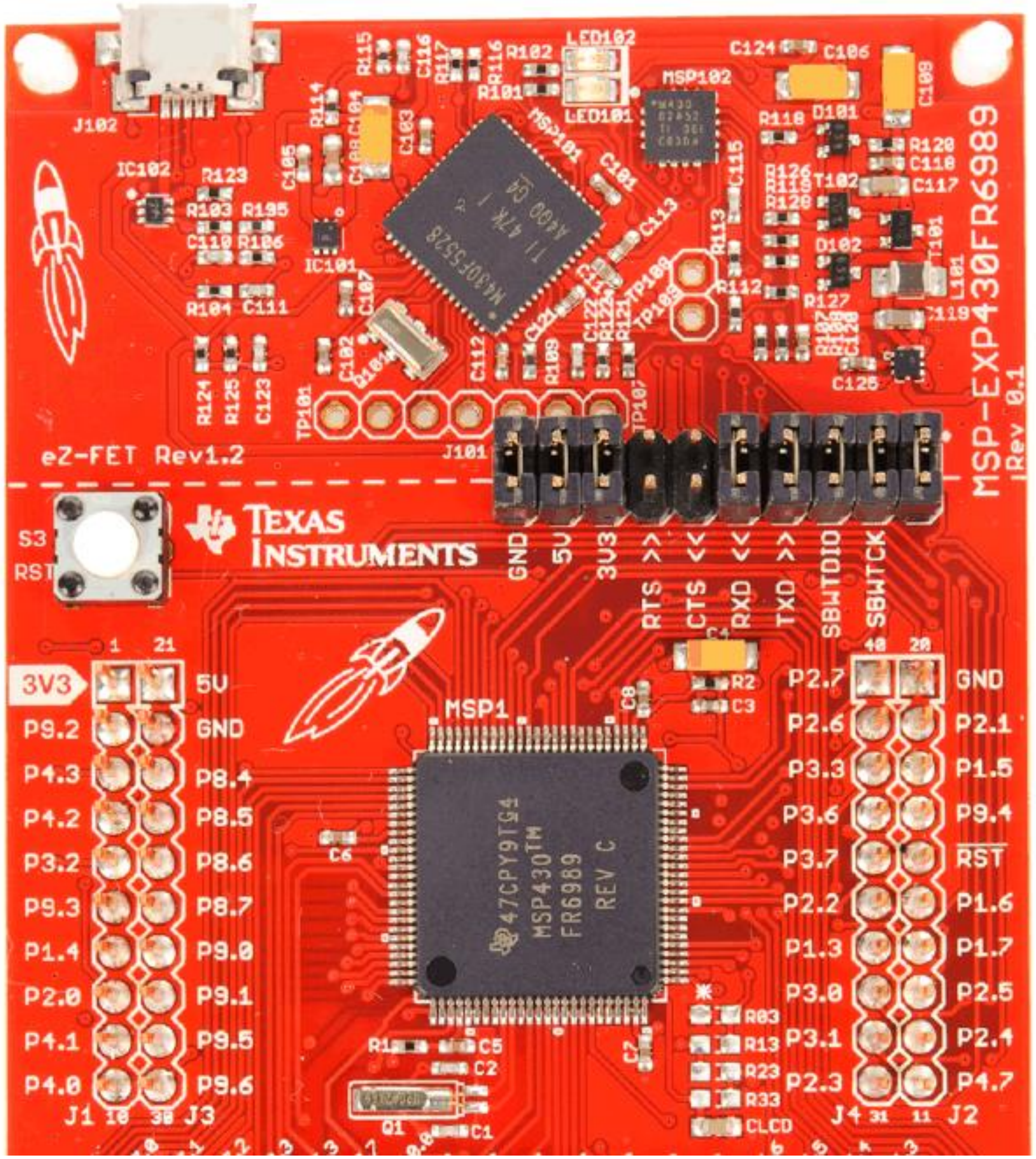
- An Integrated Development Environment (IDE) for microcontrollers is a software suite that provides a comprehensive set of tools and features to facilitate the development, programming, debugging, and testing of embedded software for microcontroller-based systems.



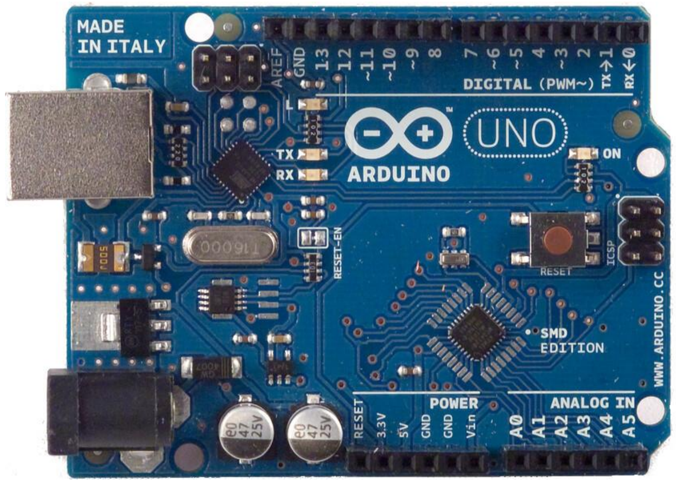
Programming the microcontroller

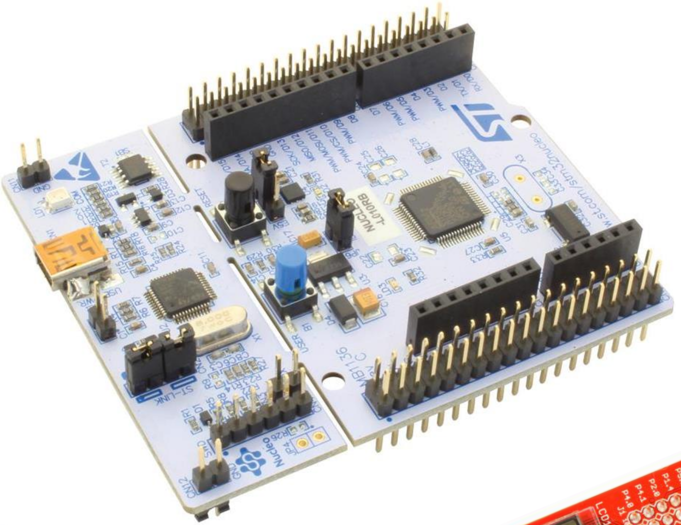
- IDEs often come with tools for **programming (flashing)** the microcontroller's memory with the compiled code. This is essential for loading the firmware onto the target microcontroller.

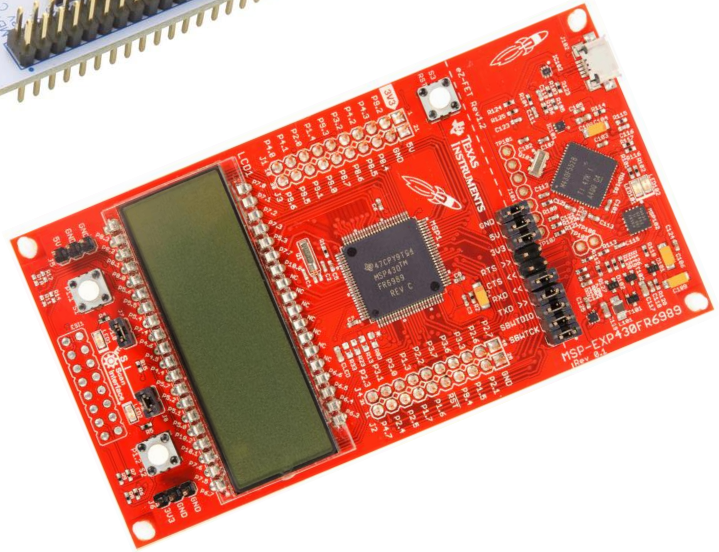


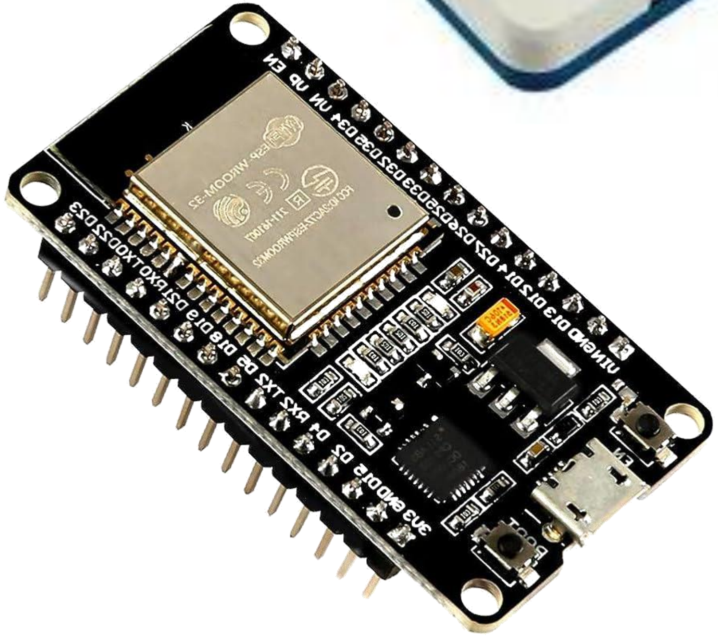


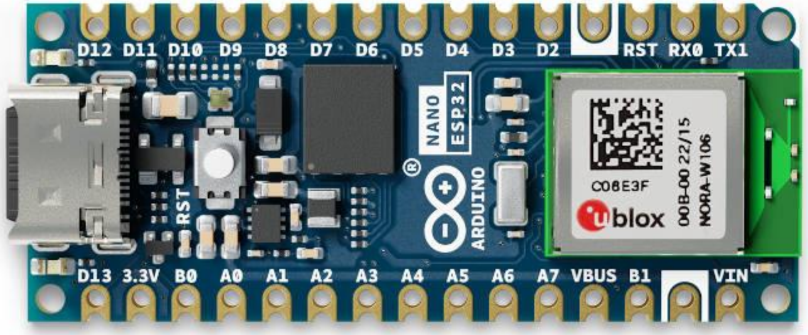
Programming the microcontroller











Content

What is a microcontroller?

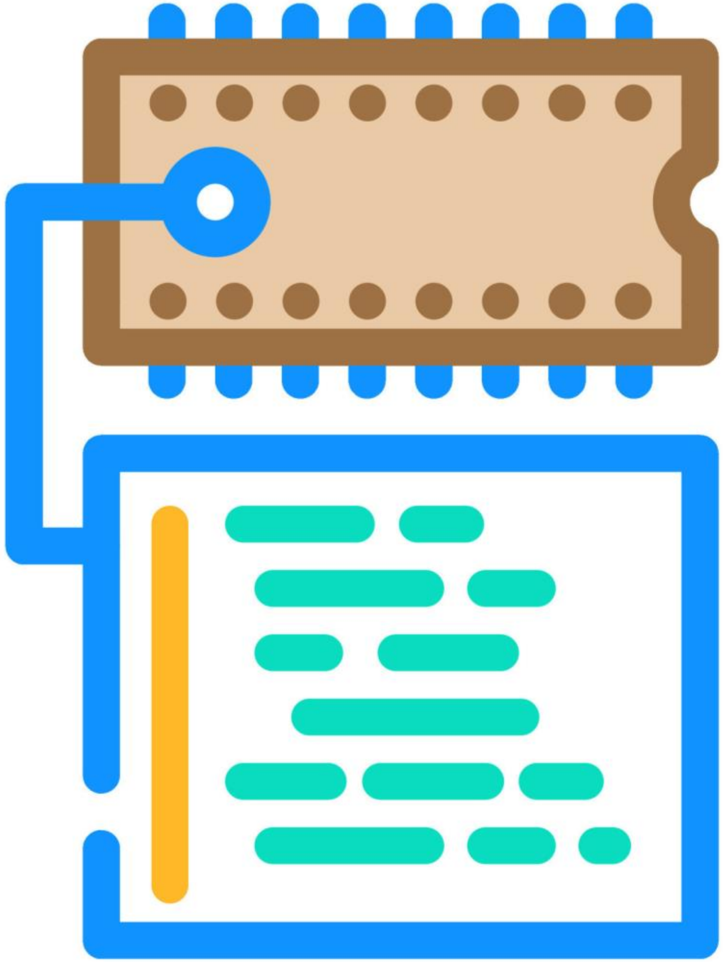
Powering up a microcontroller

Microcontroller memory

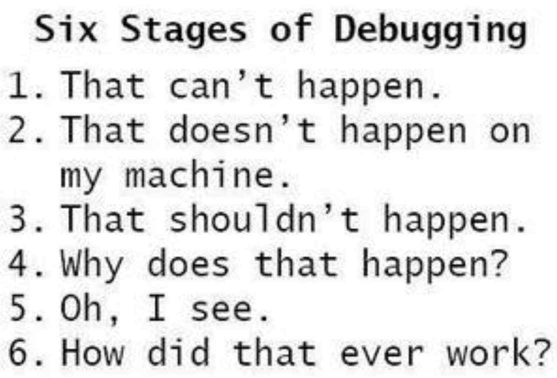
Microcontroller peripherals

Programming the microcontroller

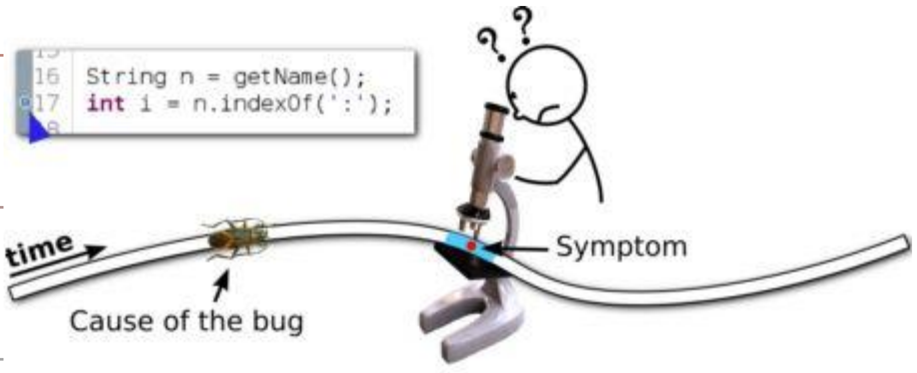
Debugging a microcontroller code



Content



Six Stages of Debugging
1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?



16 String n = getName();
17 int i = n.indexOf(':');
time
Cause of the bug
Symptom

Debugging a microcontroller code

- Debugging is the process of identifying, analyzing, and resolving issues within a software or hardware system.

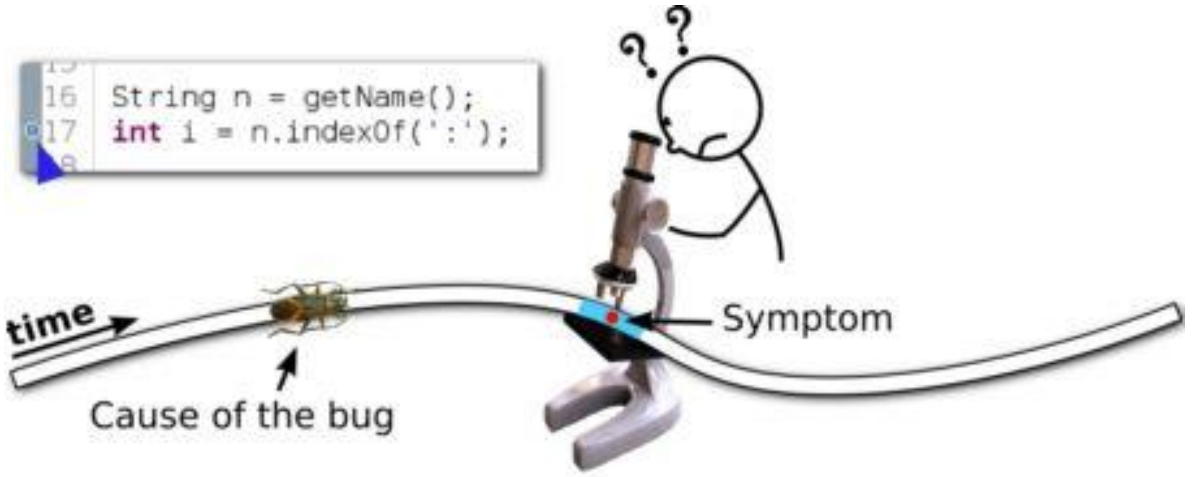
Debugging a microcontroller code

- Debugging is the process of identifying, analyzing, and resolving issues within a software or hardware system.

- Due to the specialized nature of embedded systems, errors can lead to severe consequences, such as equipment malfunction or even safety hazards.

Debugging a microcontroller code

- Common Debugging Challenges in Embedded Systems:
  - Limited Resources and Processing Power
  - Real-Time Constrains
  - Complex Hardware and Software Interactions
  - Concurrency Issues
  - Unique Platform-Specific Challenges

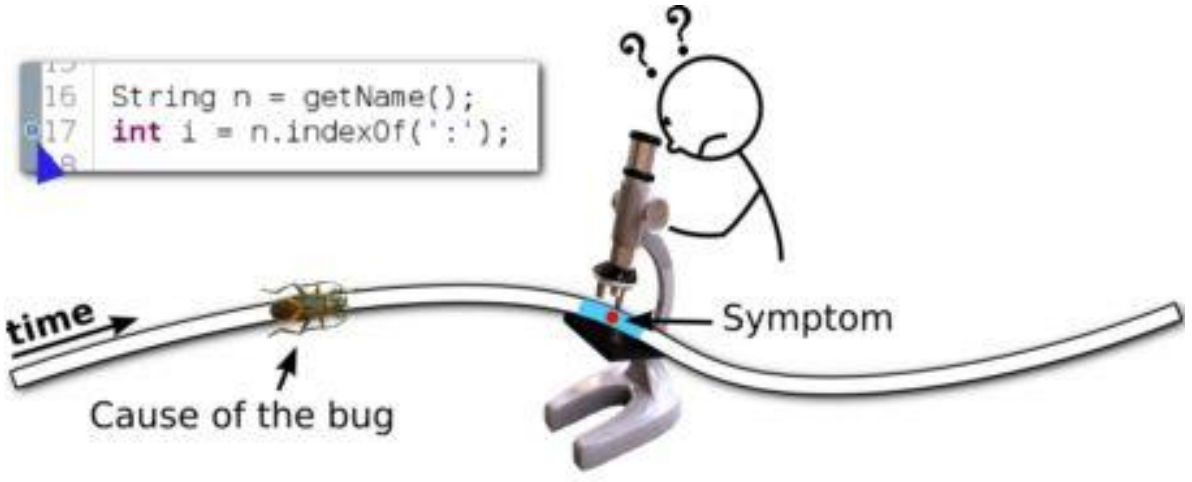


Debugging a microcontroller code - Techniques

- The Blinky LED: Using an LED as a microcontroller 'alive' indicator.

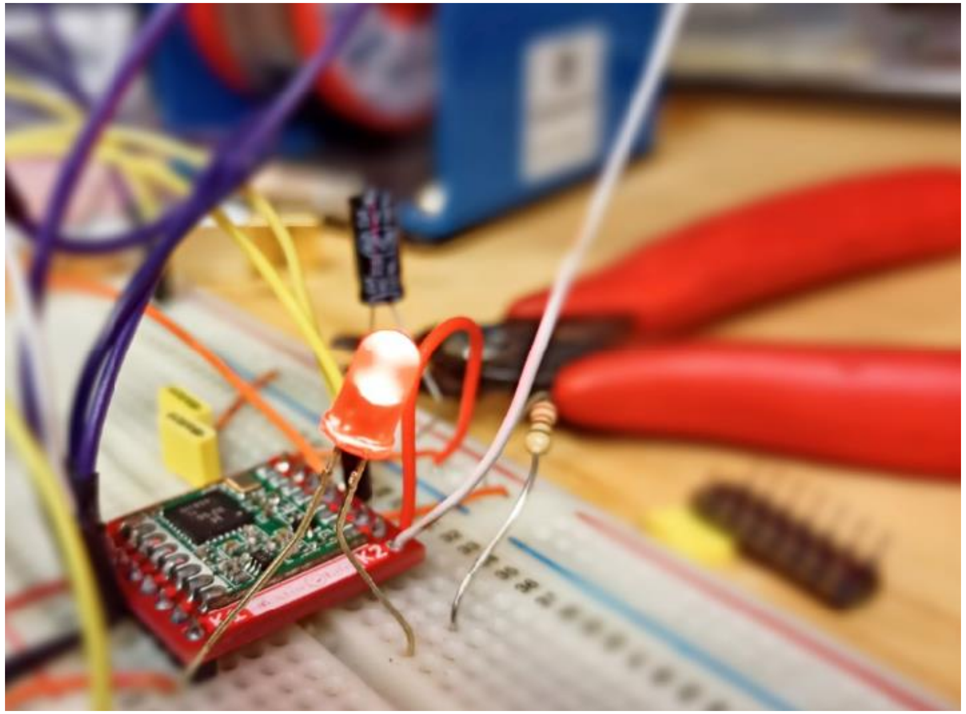
- By far the simplest debug tool is a resistor and an LED of your choosing.

- Connected to a spare general-purpose I/O pin (GPIO), it can be used like a latch at a strategic point in the code to leave an electronic breadcrumb.



Debugging a microcontroller code - Techniques

- **The Blinky LED**: Using an LED as a microcontroller 'alive' indicator.
- By far the simplest debug tool is a resistor and an LED of your choosing.
- Connected to a spare general-purpose I/O pin (GPIO), it can be used like a latch at a strategic point in the code to leave an electronic breadcrumb.



Debugging a microcontroller code - Techniques

- **Outputting Messages** through serial interfaces (UART) using *printf()*.
- The code behind this function is quite processor intensive.
- Assign different values to be written at different points in your code.

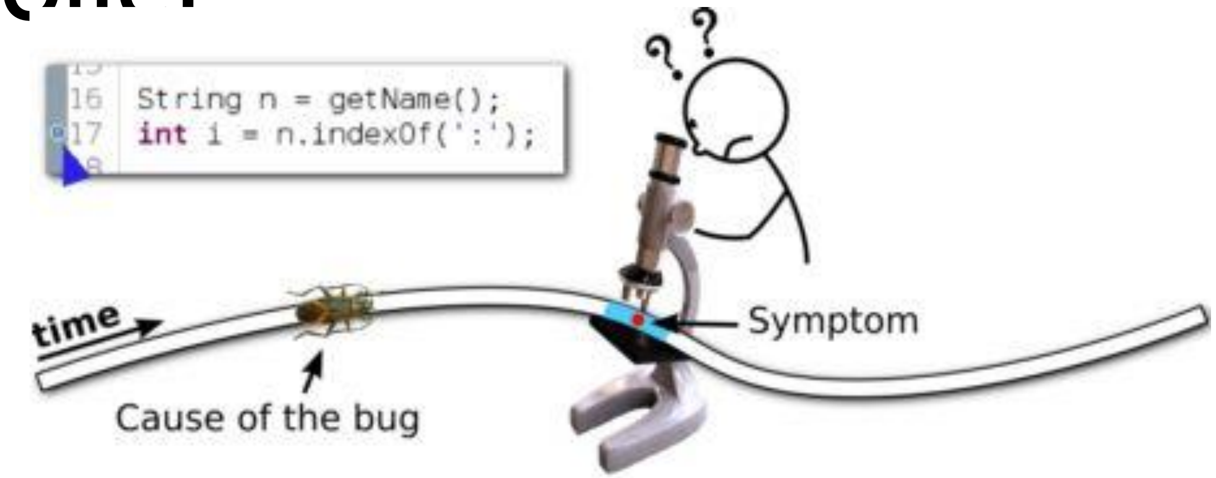
```
Serial.begin(115200);

…

Serial.println("Button pressed...");

…

Serial.println("Value: ");

Serial.write(dataByte);
```

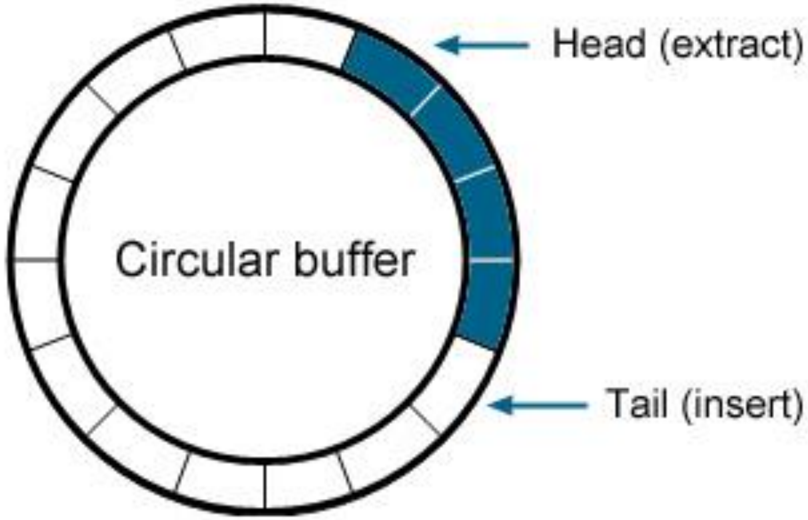
Debugging a microcontroller code - Techniques

- **Instrumentation:** place strategic information into an array without / with filtering.

- Observe the contents of the array at a later time.

- The first step when instrumenting a dump is to define a buffer in RAM to save the debugging measurements.

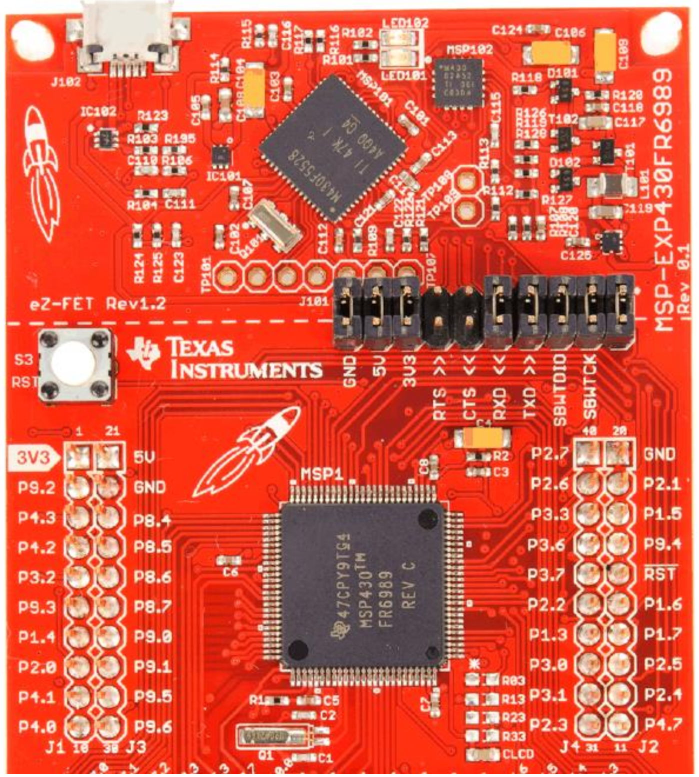


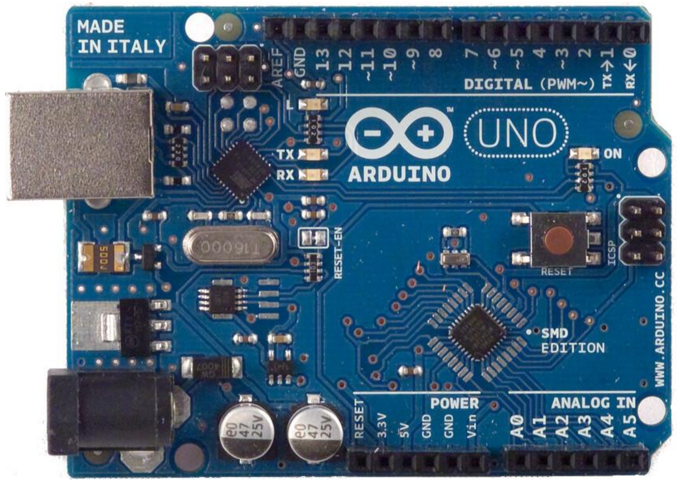


Debugging a microcontroller code - Techniques

- **Debugging Interfaces:** This opens access to all the internal circuitry, including memory, CPU registers, and all the peripherals.

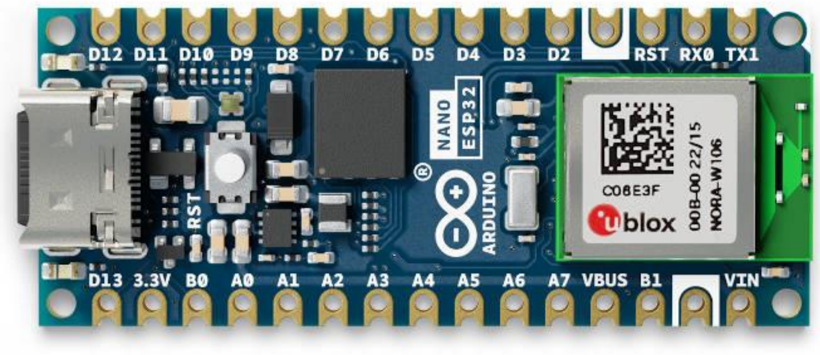
- While most microcontroller development boards come with an onboard debugger, **there are still plenty that don't**.



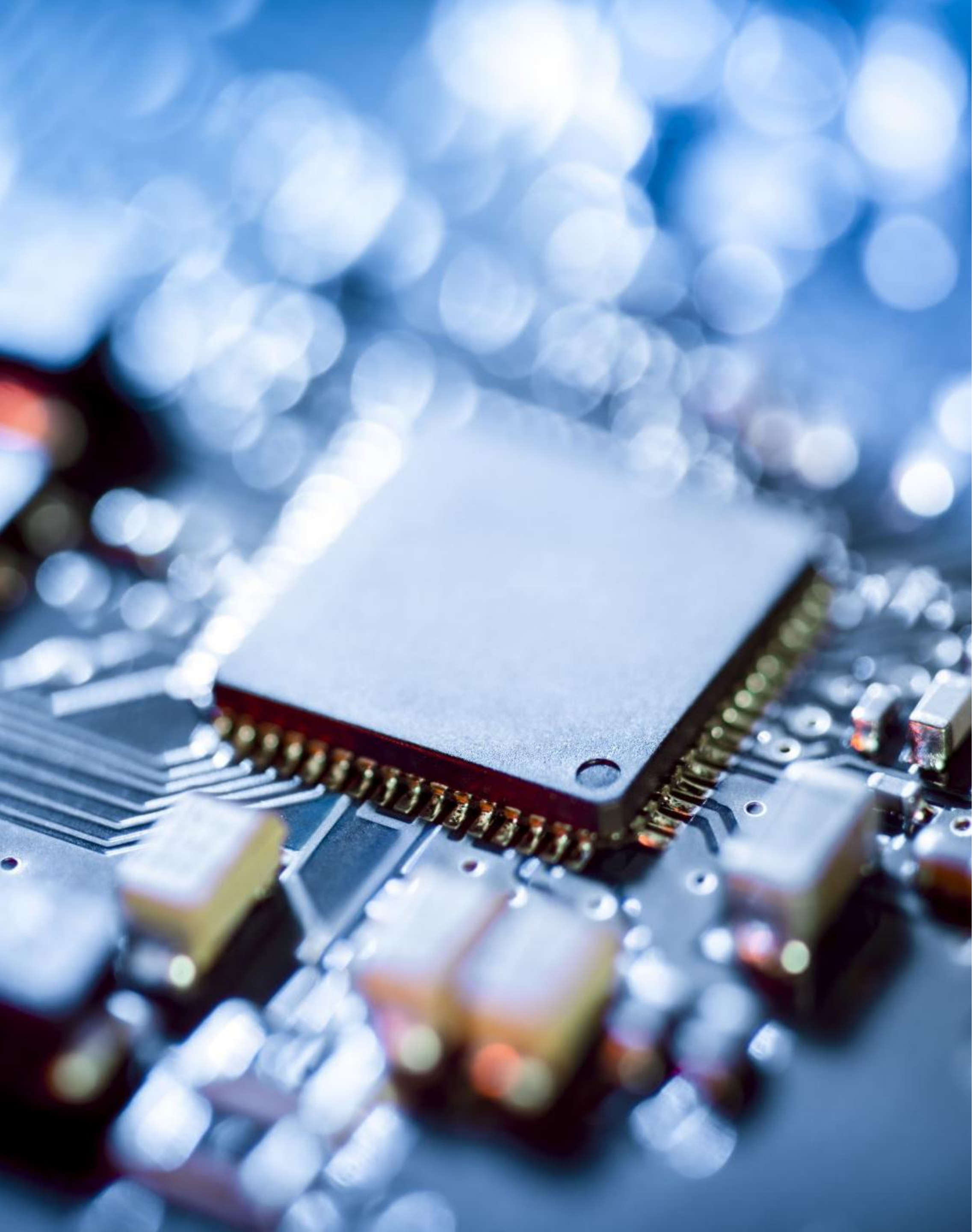


Code example

```
void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ;  // wait for serial port to connect. Needed for native USB port
only
  }

  Serial.println("Serial communication initialized.");

  // initialize digital pin LED_BUILTIN as an output.
  Serial.println("Initializing LED_BUILTIN as an output.");
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.println("Finished setting LED_BUILTIN as an output.");
}

// the loop function runs over and over again forever
void loop() {
  Serial.println("LED ON");
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the
voltage level)
  delay(1000);                       // wait for a second

  Serial.println("LED OFF");
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the
voltage LOW
  delay(1000);                       // wait for a second
}
```




```
Serial communication initialized.
Initializing LED_BUILTIN as an output.
Finished setting LED_BUILTIN as an output.
LED ON
LED OFF
LED ON
LED OFF
LED ON
LED OFF
LED ON
LED OFF
LED ON
```



Some Questions

- Where are global variables and constant values stored in a microcontroller?
- Is *printf()* suitable for debugging in an embedded system?
- Are microprocessors and microcontrollers the same thing?

Thank you for your attention!