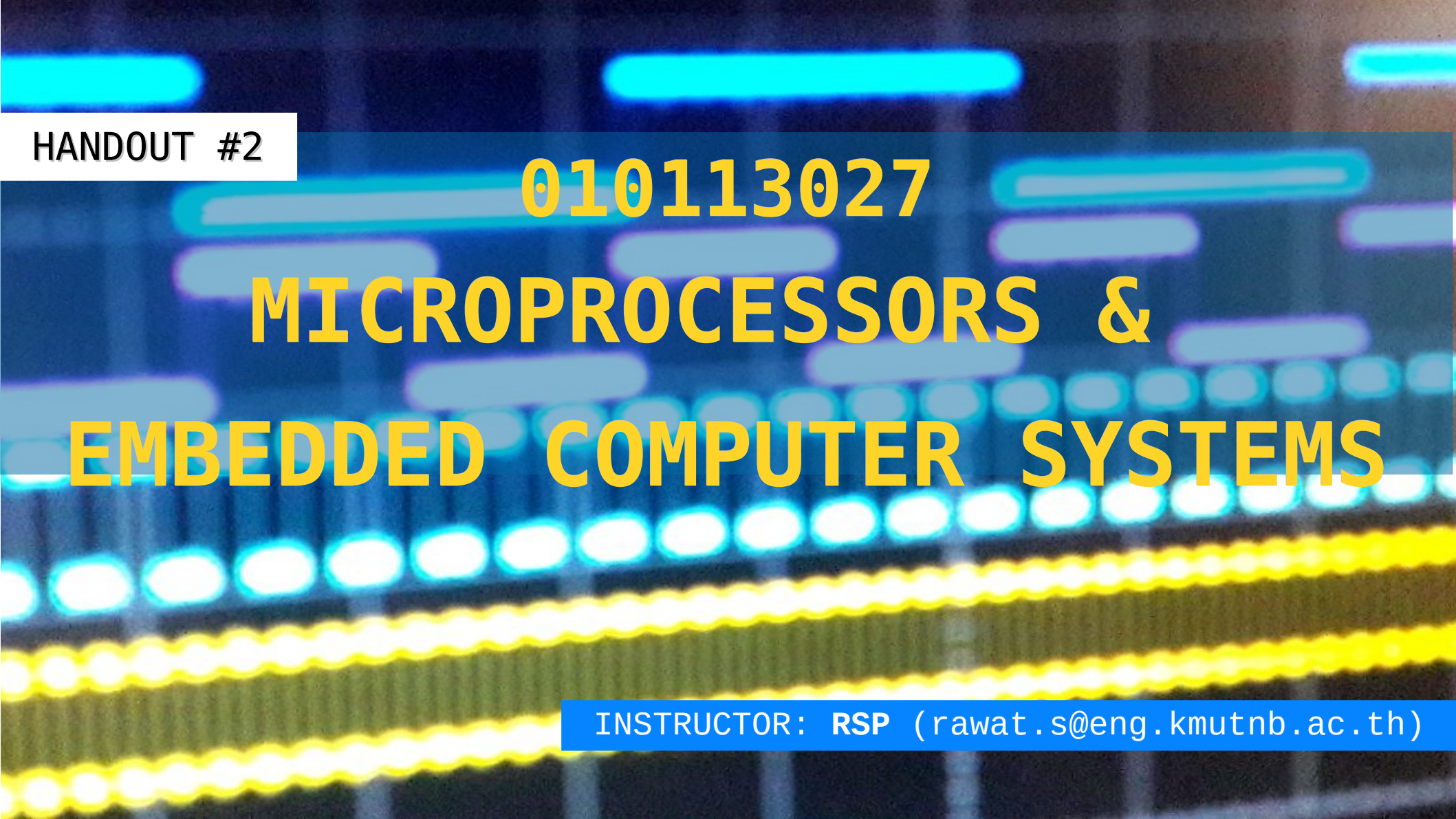


HANDOUT #2

010113027
MICROPROCESSORS & EMBEDDED COMPUTER SYSTEMS

INSTRUCTOR: **RSP** (rawat.s@eng.kmutnb.ac.th)

Key Topics

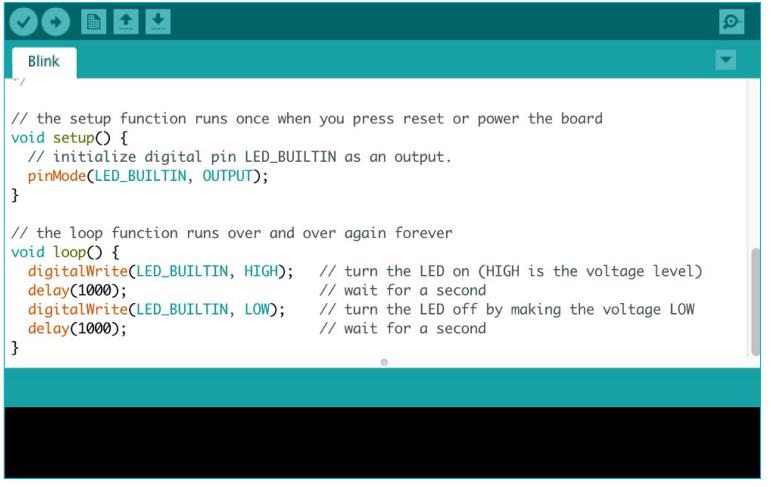
- Introduction to Arduino
- Advantages of Using Arduino
- Arduino Software and Alternatives
- History of Arduino
- Arduino Hardware Evolution
- Recent Arduino Boards
- Criteria for Choosing Arduino or MCU Boards
- Arduino API and Arduino Libraries
- Arduino Cores and Arduino Board Managers
- Arduino Bootloaders
- Arduino Software Flow

- History of Atmel MCU chips
- AVR Programmers / In-Circuit Debuggers
- ATmega328P-based Arduino Breadboard Prototyping
- AVR Programming Approaches
- Textbooks for AVR Programming
- ATmega32A vs. ATmega328P

Introduction to Arduino

- **Arduino** is an **open-source electronics platform** created in around **2003** for education to make microcontroller programming accessible to students, hobbyists, and makers.
- It consists of following key parts:
  - **Hardware**: Boards with 8-bit microcontrollers (e.g., Arduino Uno, Nano) and other supported boards with 32-bit MCUs like STM32, ESP32 and nRF52.
  - **Software**: Arduino IDE, Arduino CLI, and Arduino Cloud tools to write, compile, and upload programs easily.
  - **Services** / Ecosystem: Arduino Cloud for IoT, libraries, tutorials, and support for prototyping and connected projects.
  - **Community**: A large open-source community providing libraries, examples, and support.

Arduino Hardware (Arduino Uno R3)



Arduino IDE Software

Reference: https://en.wikipedia.org/wiki/Arduino

Introduction to Arduino

"Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects."

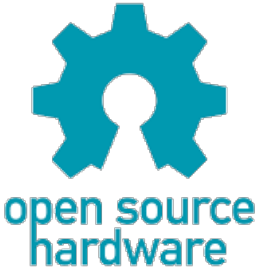
Source: https://www.arduino.cc/en/Guide/Introduction

The Arduino Ecosystem

- **Platform**: a type of computer (hardware + software)
- **Hardware**: microcontroller boards / shields
- **Software**: IDE, compiler, software libraries, ...
- **People**: a community of users, developers, vendors
- **Process**: development, maintenance, documentation support, blogs, education, training, ...

Note: **Arduino.cc**, the company that develops Arduino products, was acquired by **Qualcomm** in October 2025.

Advantages of Arduino



- **Why Arduino?**
  - **Easy-to-Use**: both software & hardware
  - **Free & Open Source Software (Cross-Platform)**
  - **Affordable Hardware** (e.g. microcontroller boards)
  - **Wide Variety of Products** (Arduino-compatible)
  - **Focusing more on Ideation & Exploration,**
    rather than mastering low-level technical details
  - **Large Community of Users / Developers**
  - **Large Number of Software Libraries for Arduino**
  - **Good documentations**: Online tutorials, blogs, youtube's videos, etc.

Arduino Software: IDE

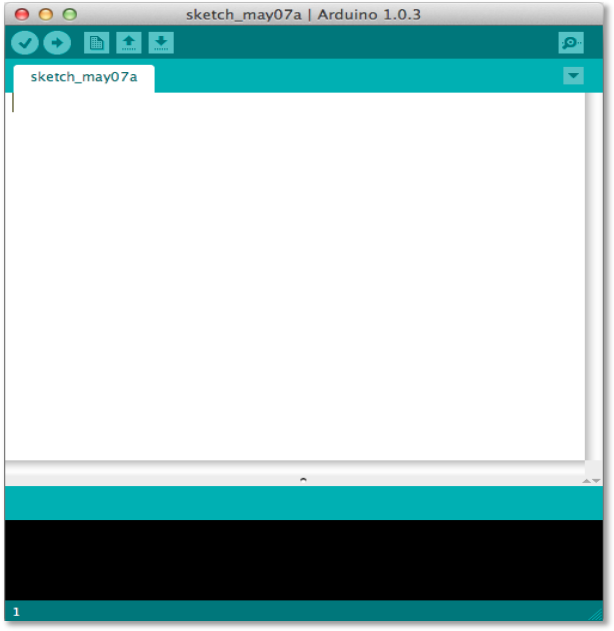
URL: https://www.arduino.cc/en/software/

- All Arduino software tools are open source and used to write, compile, and upload programs to Arduino boards.
- **Arduino IDE**: Cross-platform (Windows, macOS, Linux).
  - **Arduino IDE v1.x**: Legacy desktop IDE based on Java
  - **Arduino IDE v2.x**: Modern desktop IDE with GUI improvements, code autocompletion, debugger support, and faster compilation.
- **Arduino CLI**: Command-line tool, written in **Go (Golang)**, used for compilation, board management, and uploading sketches.
- **Arduino Cloud Editor / Web-based IDE**: Online IDE that allows programming and uploading sketches from a web browser.
- **Arduino Cloud**: A cloud-based platform by Arduino for developing, managing, and connecting IoT devices.
- **Arduino Lab for MicroPython**
- **Arduino App Labs for Arduino Uno Q board**
- **Arduino PLC IDE**

History of Arduino

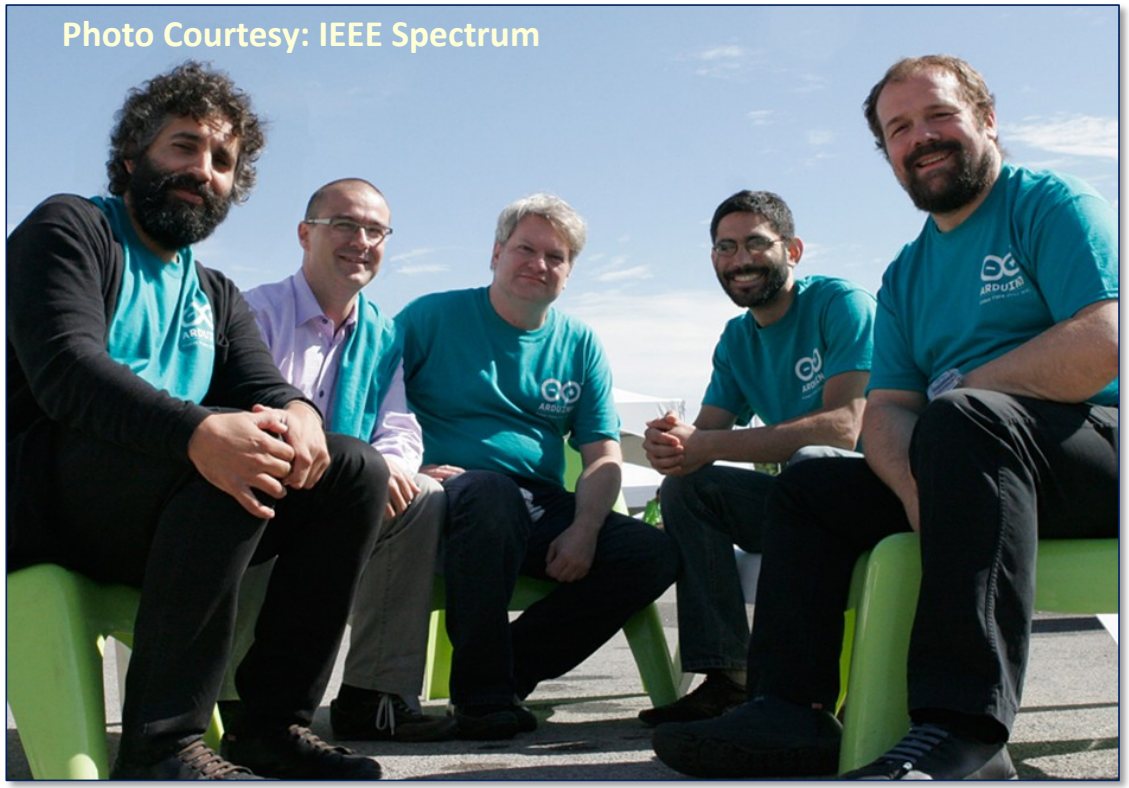
- Started around **2005** at **Interaction Design Institute**, Ivrea (**Italy**).
  - **Massimo Banzi** worked as an instructor.
  - **Banzi** with his student **Hernando Barragán** first developed / initiated the **Wiring Development Platform (IDE)** project; Inspired by **Processing IDE** (MIT Media Lab)

Processing IDE



Arduino IDE

Arduino Team



Arduino developer team:
- David Cuartielles
- Gianluca Martino
- Tom Igoe
- David Mellis
- Massimo Banzi

Source: "The Making of Arduino: How five friends engineered a small circuit board that's taking the DIY world by storm" by David Kushner (22 Oct. 2011), https://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino

First Prototyping of Arduino Hardware

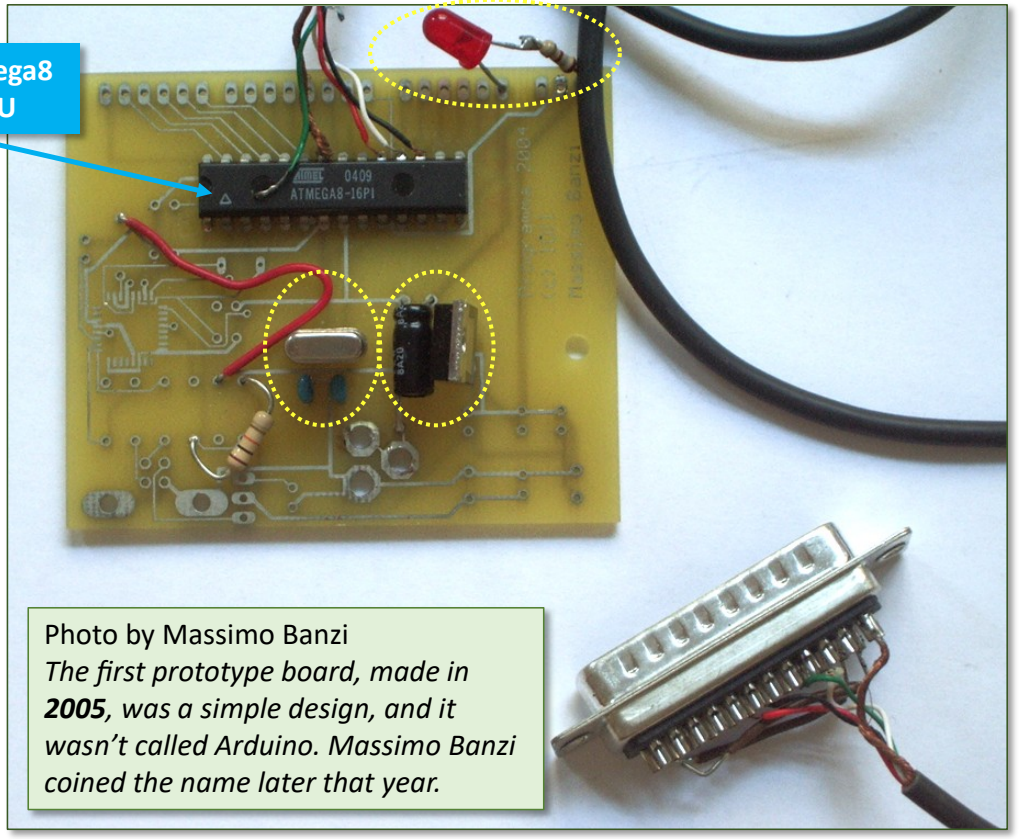
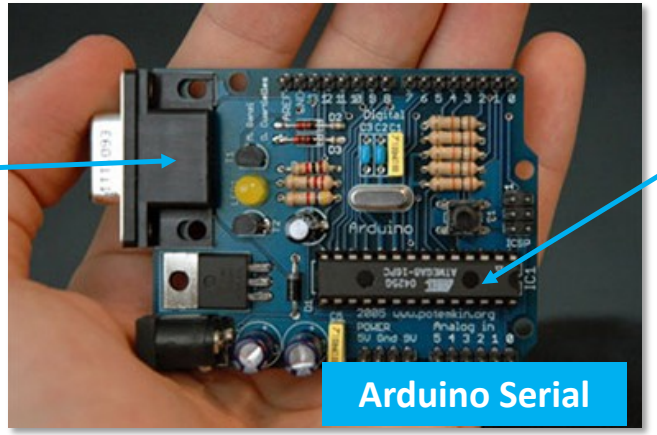
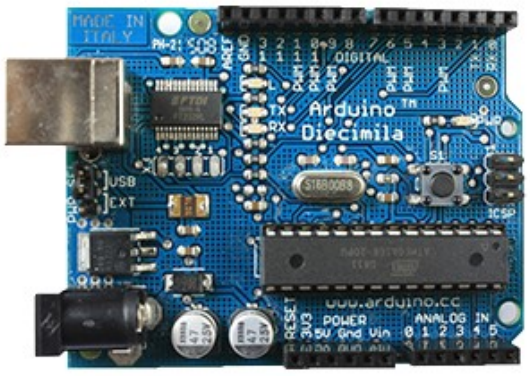


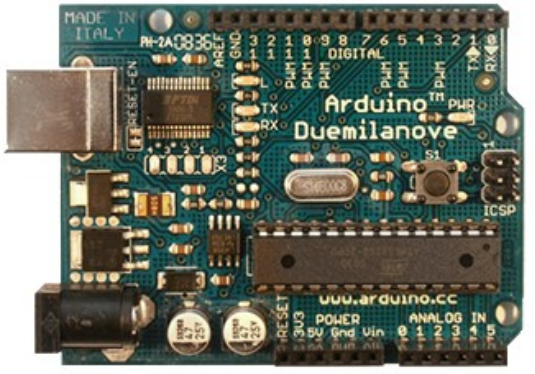
Photo by Massimo Banzi
*The first prototype board, made in **2005**, was a simple design, and it wasn't called Arduino. Massimo Banzi coined the name later that year.*

Source: https://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino

First Commercial Arduino Hardware







Source: Arduino.cc

Arduino Hardware Evolution & Trend

- **Shifting from 8-bit to 32-bit Solutions**
  - More **SRAM & Flash memory**
  - Higher **performance** (higher clock rate)
  - Capable of running **RTOS** (mbed OS, FreeRTOS, Zephyr RTOS)
  - Support **Machine Learning** on MCUs / Edge Devices (e.g. **TinyML**)
  - Various Options for **IoT Connectivity** (LAN/Ethernet, Wi-Fi, BLE, ZigBee / Matter, LoRa / LoRaWAN, 4G / 5G, NB-IoT)

RTOS = Real-Time Operating System

Criteria for Choosing Arduino Boards

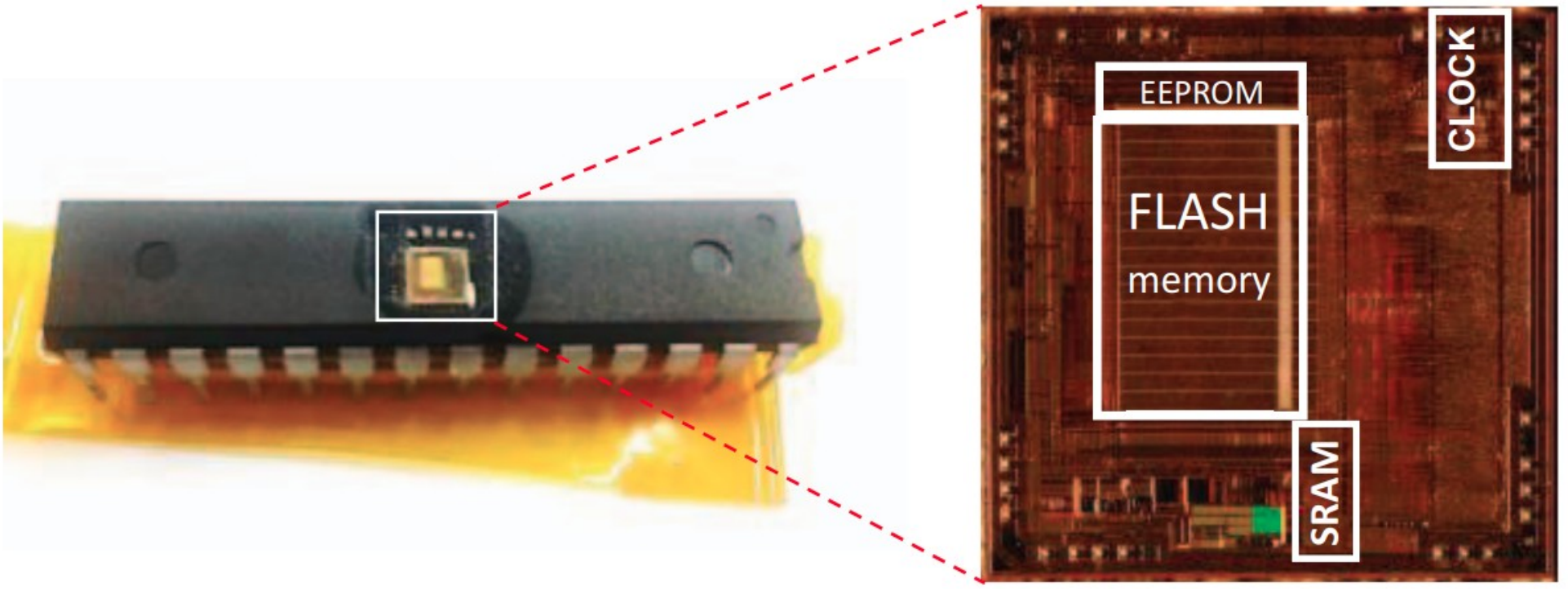
- **Performance**:
  - Low-speed vs. high-speed (Max. clock rate)
  - 8-bit vs. 32-bit MCUs
- **I/O Count & Interfacing / Types of On-chip Peripherals**
  - Number of **GPIO** pins / **PWM** pins
  - Number of **ADC** input pins / **DAC** output pins
  - Number of **I2C** / **SPI** / **I2S** / **UART** units
- **On-Board Modules (**e.g. sensors, communication modules)
- **On-Chip Memory Requirements**
  - **SRAM** size (main memory)
  - **Flash Memory** size (Program storage)
  - **EEPROM** size (Nonvolatile storage for user data)
- **Breadboard-Friendly** or not
  - Suitable for Breadboard-style Circuit Prototyping

Arduino Boards and Their Main MCUs

Arduino Board →	MCU
Arduino Uno (R3)	ATmega328P
Arduino Mega 2560	ATmega2560
Arduino Leonardo	ATmega32U4
Arduino Micro / Pro Micro	ATmega32U4
Arduino Nano (classic)	ATmega328P
Arduino Pro Mini	ATmega328P
Arduino Due	ATSAM3X8E
Arduino Uno WiFi Rev2	ATmega4809
Arduino Nano Every	ATmega4809

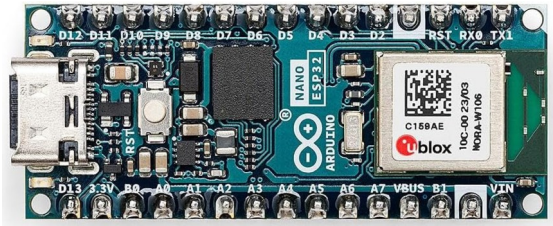
Arduino Board →	MCU
Arduino Zero	ATSAMD21G18
Arduino MKR Zero	ATSAMD21G18
MKR WiFi 1010	ATSAMD21G18
Nano 33 IoT	ATSAMD21G18
Nano 33 BLE / BLE Sense	nRF52840
Portenta H7	STM32H747
Portenta C33	Renesas RA6M5
Arduino GIGA R1	STM32H747
Uno R4 Minima	Renesas RA4M1
Uno R4 WiFi	Renesas RA4M1
Arduino Nano R4	Renesas RA4M1
Arduino Nano ESP32	ESP32-S3

ATmega328P Die Shot

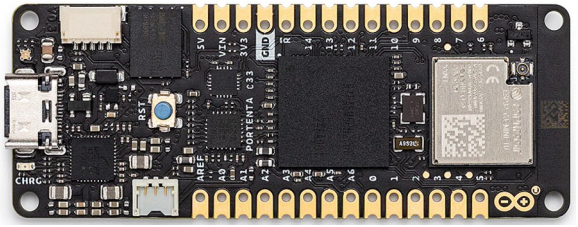


Source: Interceptive side channel attack on AES-128 wireless communications for IoT applications, October 2016 (DOI: 10.1109/APCCAS.2016.7804081)

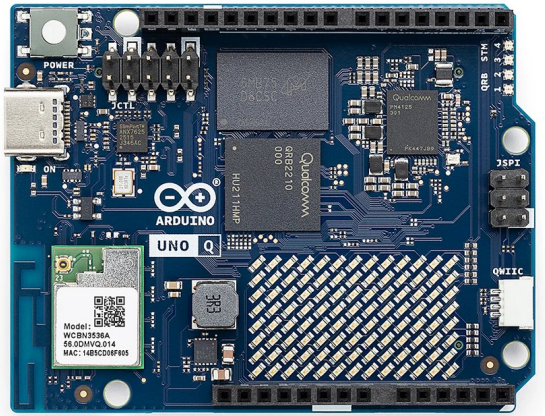
Most-Recent Arduino Boards: Examples



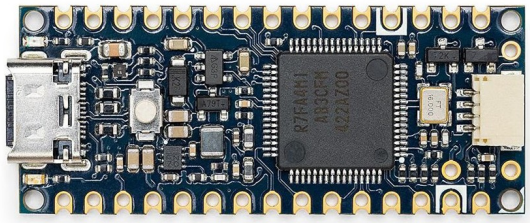
Nano ESP32



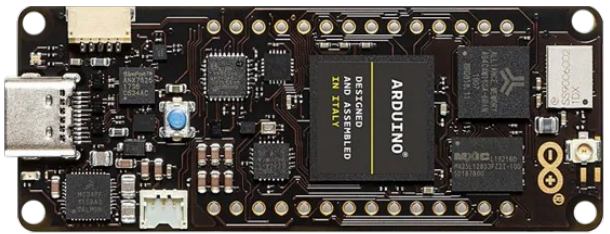
Portenta C33



Uno Q

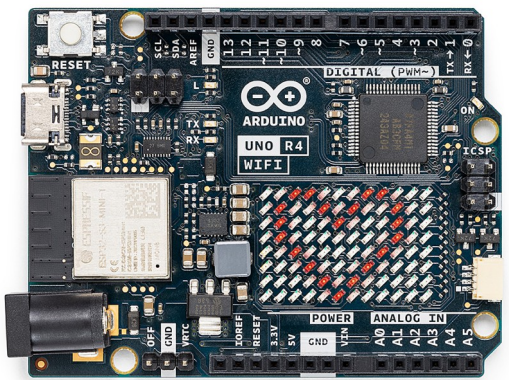


Portenta C33

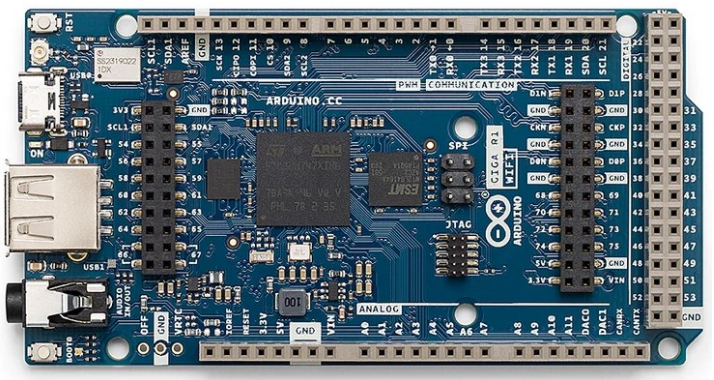


Portenta H7

Opta PLC



Uno R4 WiFi



Giga R1 WiFi

Criteria for Choosing Arduino Boards

Official Arduino & Clones / Arduino Compatible

- **Official Arduino Boards**: sold by arduino.cc or distributors
- **Clone Boards**: cheaper, lower-quality, mostly made in China
- **Arduino Compatible Boards or Alternatives (cheaper):** Well-known vendors: Sparkfun, Adafruit, SeeedStudio, …

Other Criteria

- **Cost Concern**
- **Local (domestic) vs. International Sellers**
  - Online shopping / website
  - Stock Availability / Delivery Time
- **Single Arduino Boards vs. Arduino Kits**
  - With various modules (e.g. sensors, actuators, ...)

Arduino C/C++ Programming

Arduino API (https://github.com/arduino/ArduinoCore-API)

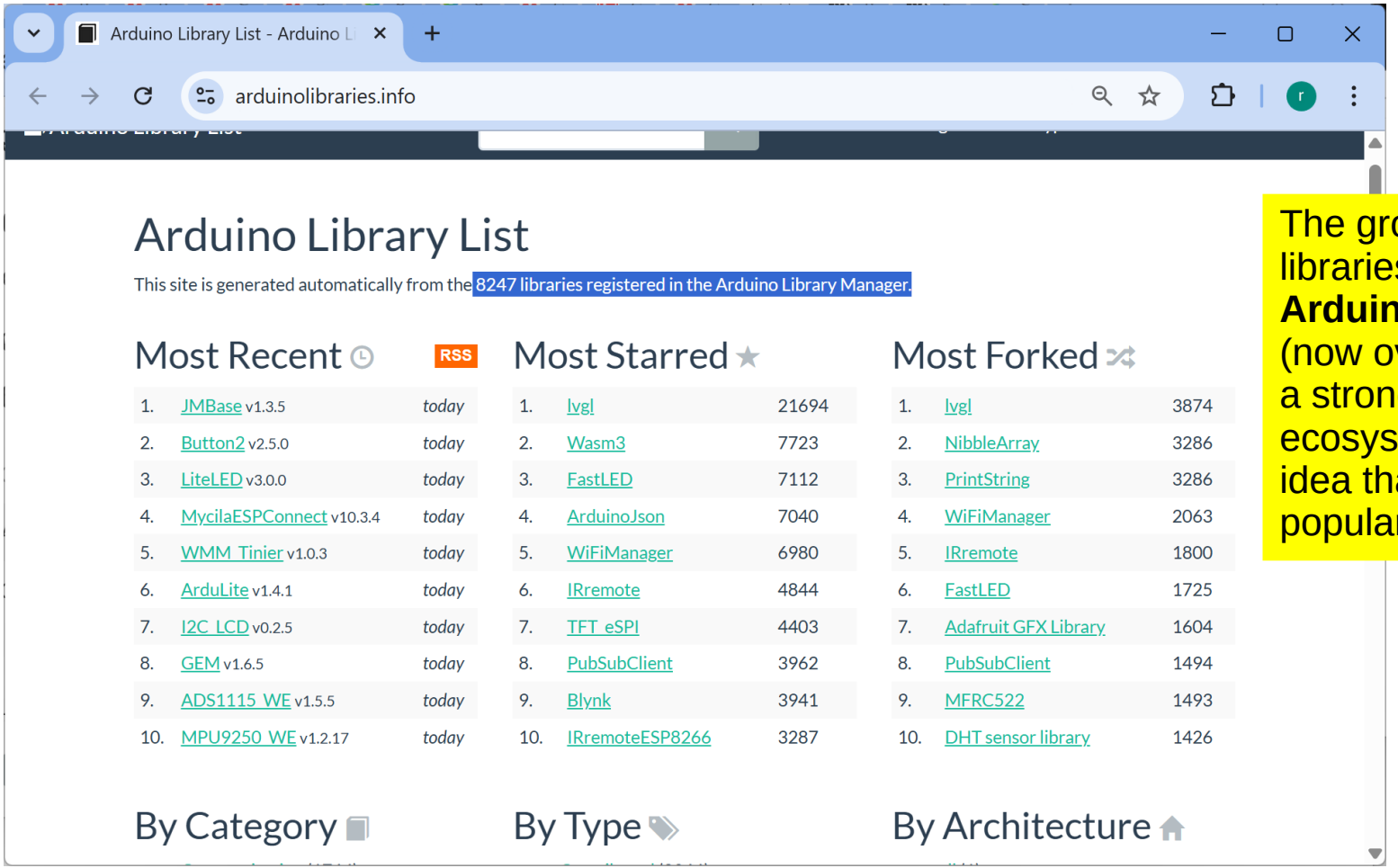
- It is written in C/C++ and provides a uniform set of functions that generally work across supported boards.
- Some functions only exist for certain hardware.
- **Arduino cores** are built on top of the vendor SDKs like **STM32 HAL**, **ESP-IDF (ESP32)**, or **nRF SDK (nRF52)**. These SDKs handle low-level hardware, while Arduino provides a **higher-level, consistent API**.

Examples of Arduino API functions

- `pinMode(pin, mode)` – Configure a pin as INPUT, OUTPUT, or INPUT_PULLUP
- `digitalWrite(pin, value)` – Set a digital pin HIGH or LOW
- `digitalRead(pin)` – Read a digital pin's state (HIGH or LOW)
- `analogRead(pin)` – Read an analog value (e.g., 0–1023 for 10-bit ADC)
- `analogWrite(pin, value)` – Output a PWM signal on a pin
- `analogReference(type)` – Set ADC reference voltage
- `millis()` – Return time in milliseconds since program started
- `micros()` – Return time in microseconds since program started
- `delay(ms)` – Pause execution for given milliseconds
- `delayMicroseconds(us)` – Pause execution for given microseconds

Arduino Libraries

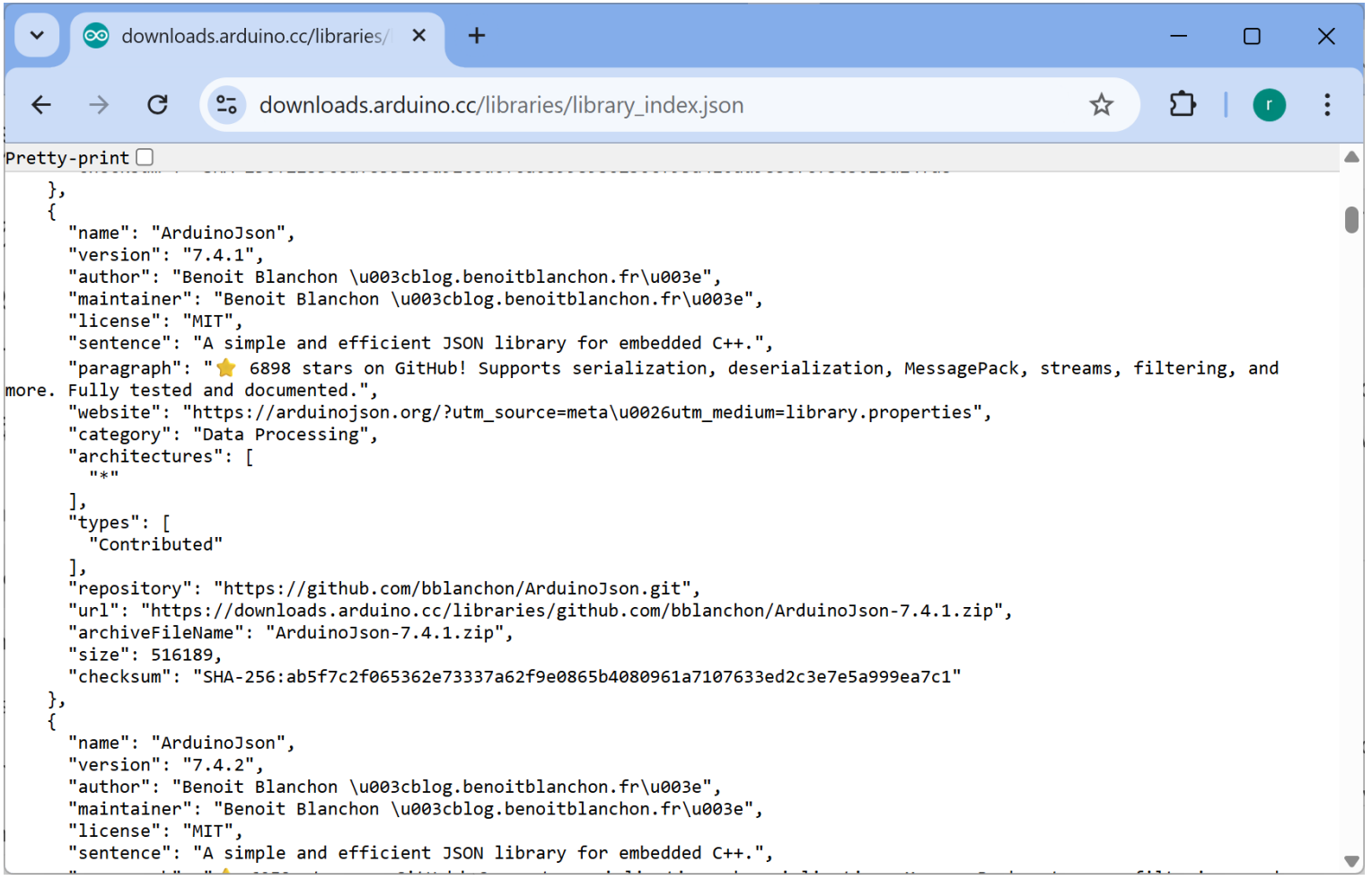
https://downloads.arduino.cc/libraries/library_index.json



The growth in the number of libraries registered in the **Arduino Library Manager** (now over 8,200) reflects a strong and active Arduino ecosystem, supporting the idea that Arduino remains popular and widely adopted.

Source: https://www.arduinolibraries.info/ [Last Access: 2025-11-15]

Arduino Libraries




```
    },
    {
      "name": "ArduinoJson",
      "version": "7.4.1",
      "author": "Benoit Blanchon \u003cblog.benoitblanchon.fr\u003e",
      "maintainer": "Benoit Blanchon \u003cblog.benoitblanchon.fr\u003e",
      "license": "MIT",
      "sentence": "A simple and efficient JSON library for embedded C++.",
      "paragraph": "⭐ 6898 stars on GitHub! Supports serialization, deserialization, MessagePack, streams, filtering, and
more. Fully tested and documented.",
      "website": "https://arduinojson.org/?utm_source=meta\u0026utm_medium=library.properties",
      "category": "Data Processing",
      "architectures": [
        "*"
      ],
      "types": [
        "Contributed"
      ],
      "repository": "https://github.com/bblanchon/ArduinoJson.git",
      "url": "https://downloads.arduino.cc/libraries/github.com/bblanchon/ArduinoJson-7.4.1.zip",
      "archiveFileName": "ArduinoJson-7.4.1.zip",
      "size": 516189,
      "checksum": "SHA-256:ab5f7c2f065362e73337a62f9e0865b4080961a7107633ed2c3e7e5a999ea7c1"
    },
    {
      "name": "ArduinoJson",
      "version": "7.4.2",
      "author": "Benoit Blanchon \u003cblog.benoitblanchon.fr\u003e",
      "maintainer": "Benoit Blanchon \u003cblog.benoitblanchon.fr\u003e",
      "license": "MIT",
      "sentence": "A simple and efficient JSON library for embedded C++.",
```


Source: https://downloads.arduino.cc/libraries/library_index.json

Arduino C/C++ Programming

What is Arduino Core?

- An Arduino core is a collection of software that allows the Arduino IDE or Arduino CLI to compile and upload code to a specific family of MCUs.
- It includes:
  - Compiler support for the MCU architecture (AVR, ARM, ESP32, etc.)
  - Low-level libraries (digital I/O, analog I/O, timers, SPI / I2C, etc.)
  - Board-specific files like bootloaders and pin definitions
  - Different boards use different microcontrollers, they require different Arduino cores.

What is Arduino Board Manager?

- A tool in the Arduino IDE used to install, update, or remove cores easily.
- It provides an online repository of cores (available on Github) from Arduino and third parties.

Examples of Arduino Cores

1) 8-bit ATmega (ATmega328P, ATmega32U4, ATmega2560)

- Boards: Uno, Nano, Mega, Leonardo, ...
- Git Repo: https://github.com/arduino/ArduinoCore-avr

2) 8-bit megaAVR (4809)

- Board: Arduino Nano Every, Arduino Uno WiFi Rev 2
- Git Repo: https://github.com/arduino/ArduinoCore-megaavr

3) 32-bit Arm Cortex-M3 (SAM3X8E)

- Board: Arduino Due
- Git Repo: https://github.com/arduino/ArduinoCore-sam

4) 32-bit Arm Cortex-M0+ (SAMD21)

- Board: Arduino Zero, MKR Zero, Nano 33 IoT
- Git Repo: https://github.com/arduino/ArduinoCore-samd

Examples of Arduino Cores

5) 32-bit Arm Cortex-M Series (M0+ / M4 / M7 / M33) + Arm Mbed OS

- Boards: Nano 33 BLE, Giga R1, Portenta H7, Opt, RP2040, ...
- Git Repo: https://github.com/arduino/ArduinoCore-mbed

6) 32-bit Arm Cortex-M Series + Zephyr RTOS

- Boards: Nano 33 IoT, Nano 33 BLE, Portenta C33, Portenta H7, Giga R1, Opta, ...
- Git Repo: https://github.com/arduino/ArduinoCore-zephyr

7) 32-bit Renesas Cortex-M33 (RA4M1 / RA6M5)

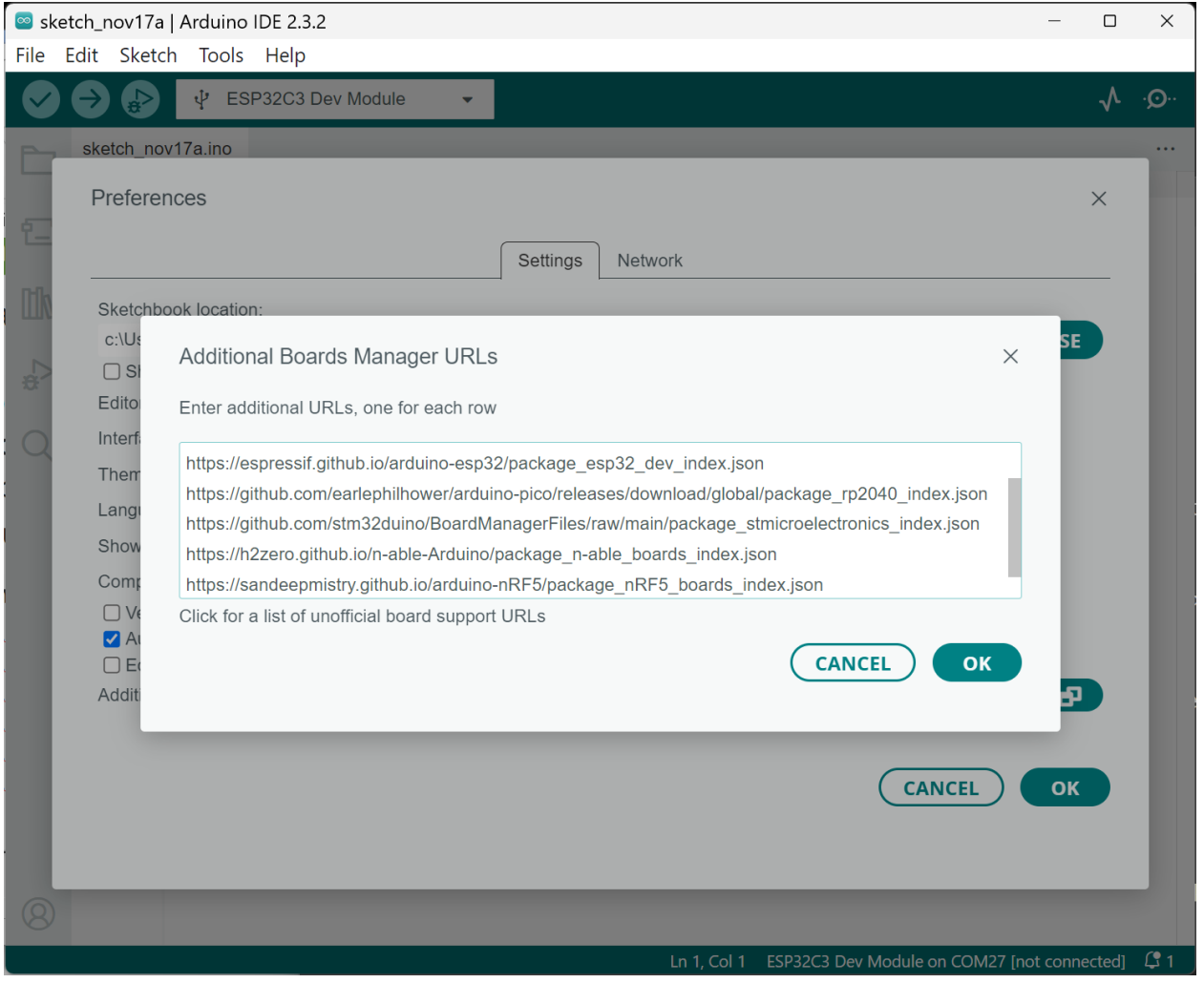
- Boards: Portenta C33, Uno R4, Nano R4, ...
- Git Repo: https://github.com/arduino/ArduinoCore-renesas

Arduino Board Managers

What is Arduino Board Manager?

- It is a **software tool built into the Arduino IDE** that allows users to download, install, and update **Arduino cores and board support packages**.
- It simplifies adding new boards from Arduino or third-party vendors.
- It can manage multiple MCU families (AVR, ESP8266, ESP32, RP2040, STM32, etc.) from a single interface.
- The **default and official JSON file** that the **Arduino Boards Manager** uses to list the **standard Arduino cores** (like AVR, SAMD, and newer official boards) is: https://downloads.arduino.cc/packages/package_index.json.
- For third-party vendors such as **ESP32**: https://espressif.github.io/arduino-esp32/package_esp32_index.json
- In Arduino IDE: Go to **File → Preferences → Additional Board Manager URLs**

Arduino Board Managers



Arduino CLI

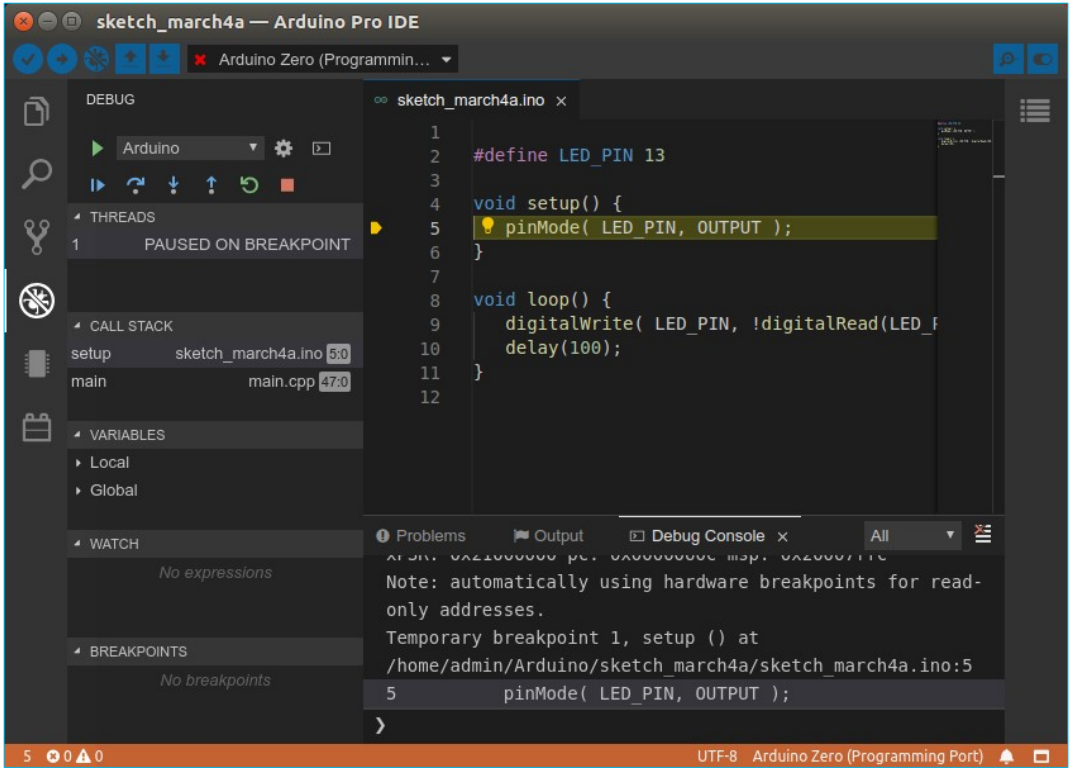
What is **Arduino CLI** ?

- **CLI** = **C**ommand **L**ine **I**nterface (No GUI)
- **Open-source** (code written in **Go** / **golang**)
- **Cross-platform** (OS: Windows, Linux, Mac OS)
- **Typical Usage**: Scripting / Automated Build Process, Cloud compilation
- **Git Repo**: https://github.com/arduino/arduino-cli

Command Usage:

```
arduino-cli board list
arduino-cli core update-index
arduino-cli core install arduino:avr
arduino-cli sketch new MySketch
arduino-cli compile --fqbn arduino:avr:nano MySketch
arduino-cli upload -p COM3 --fqbn arduino:avr:nano MySketch
```

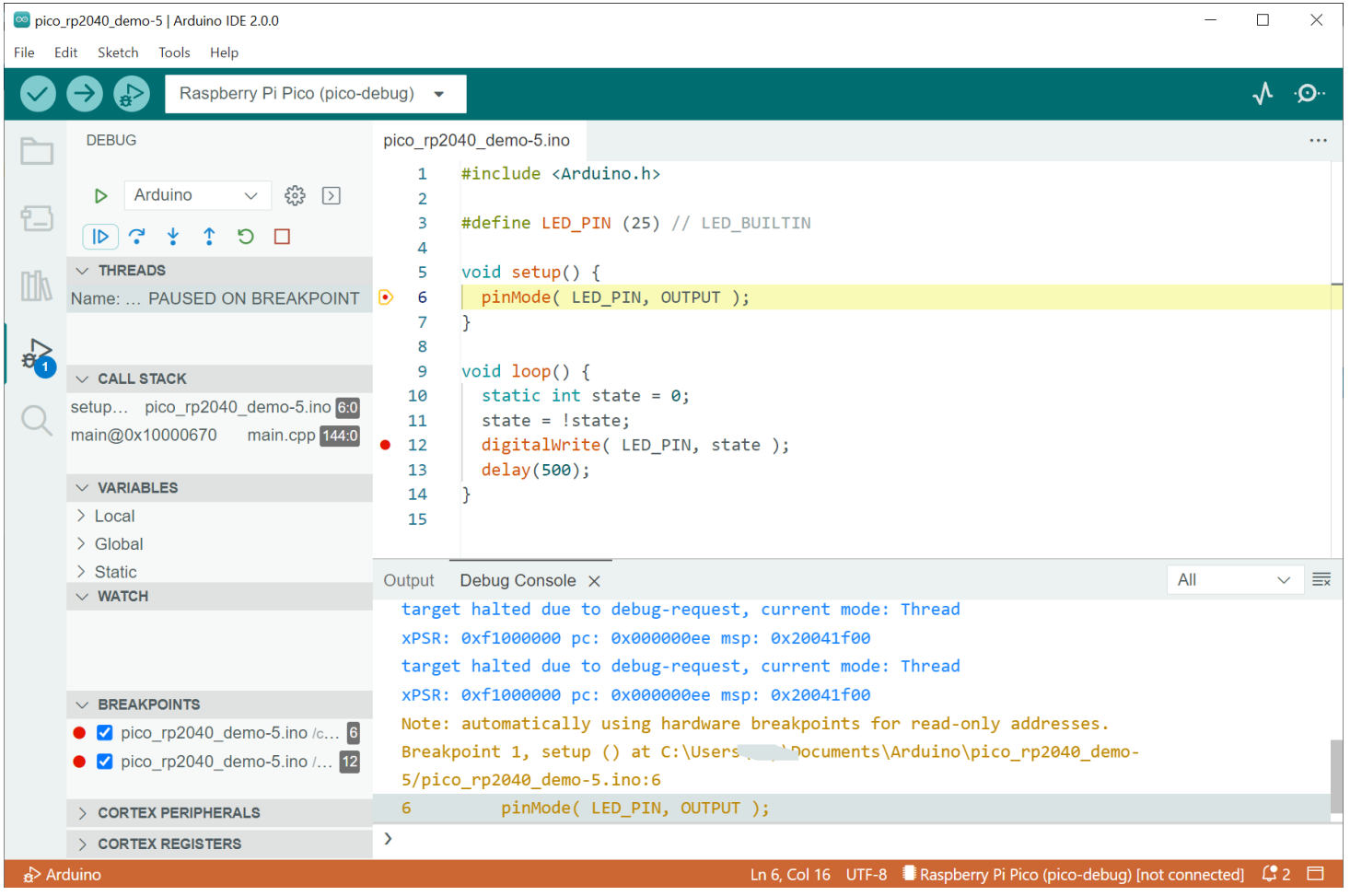
Arduino IDE with On-Chip Debug Support



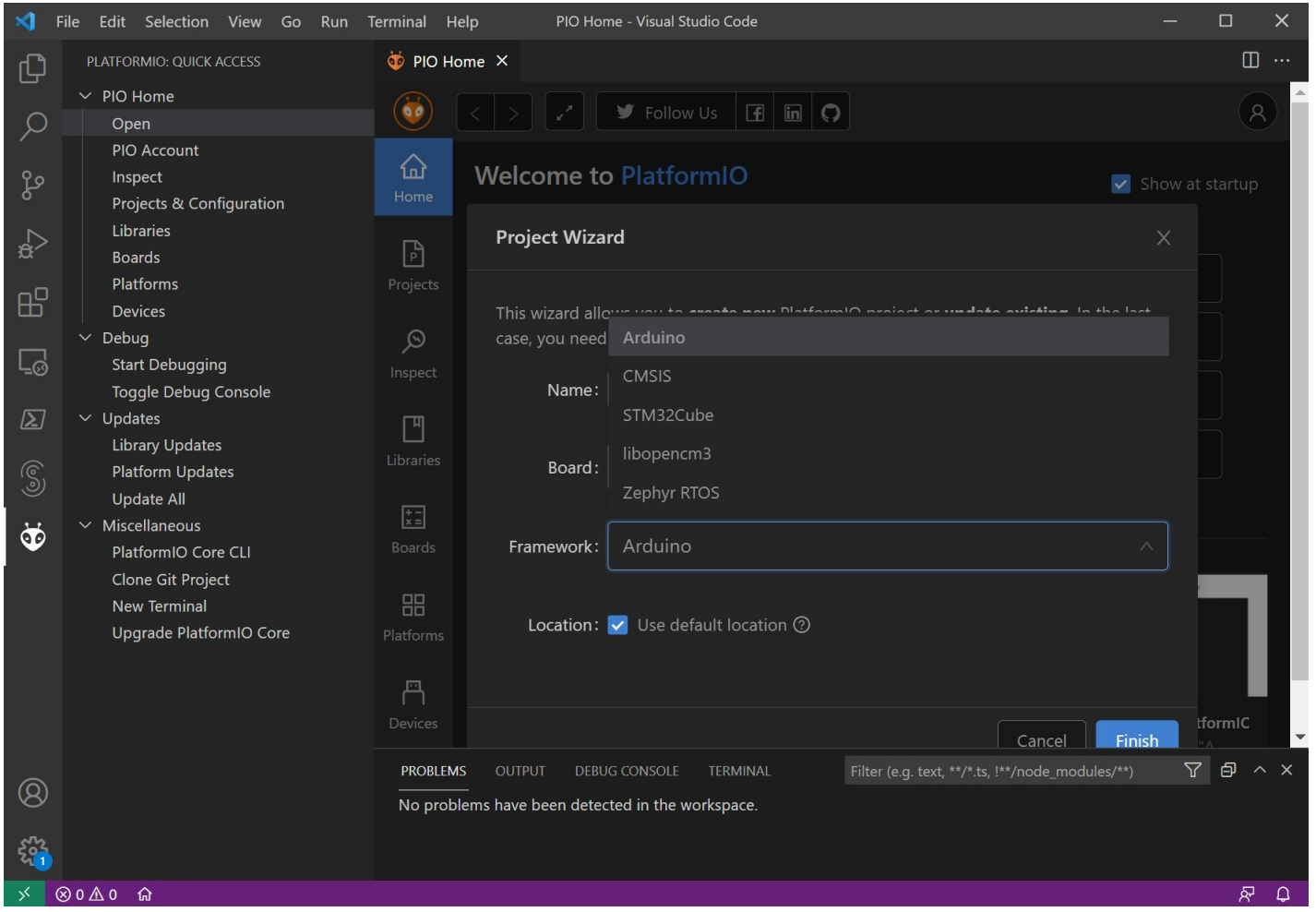
Arduino IDE With On-Chip Debug Support (Arduino IDE 2.x)

- Hardware breakpoints
- Step / run / stop / resume execution
- Watchpoints (view memory or variables)
- MCU with On-chip debug interface (SWD, JTAG, or vendor-specific)
- Official support is mainly for ARM Cortex-M based boards, including RP2040 (Cortex-M0+)

Arduino IDE with On-Chip Debug Support

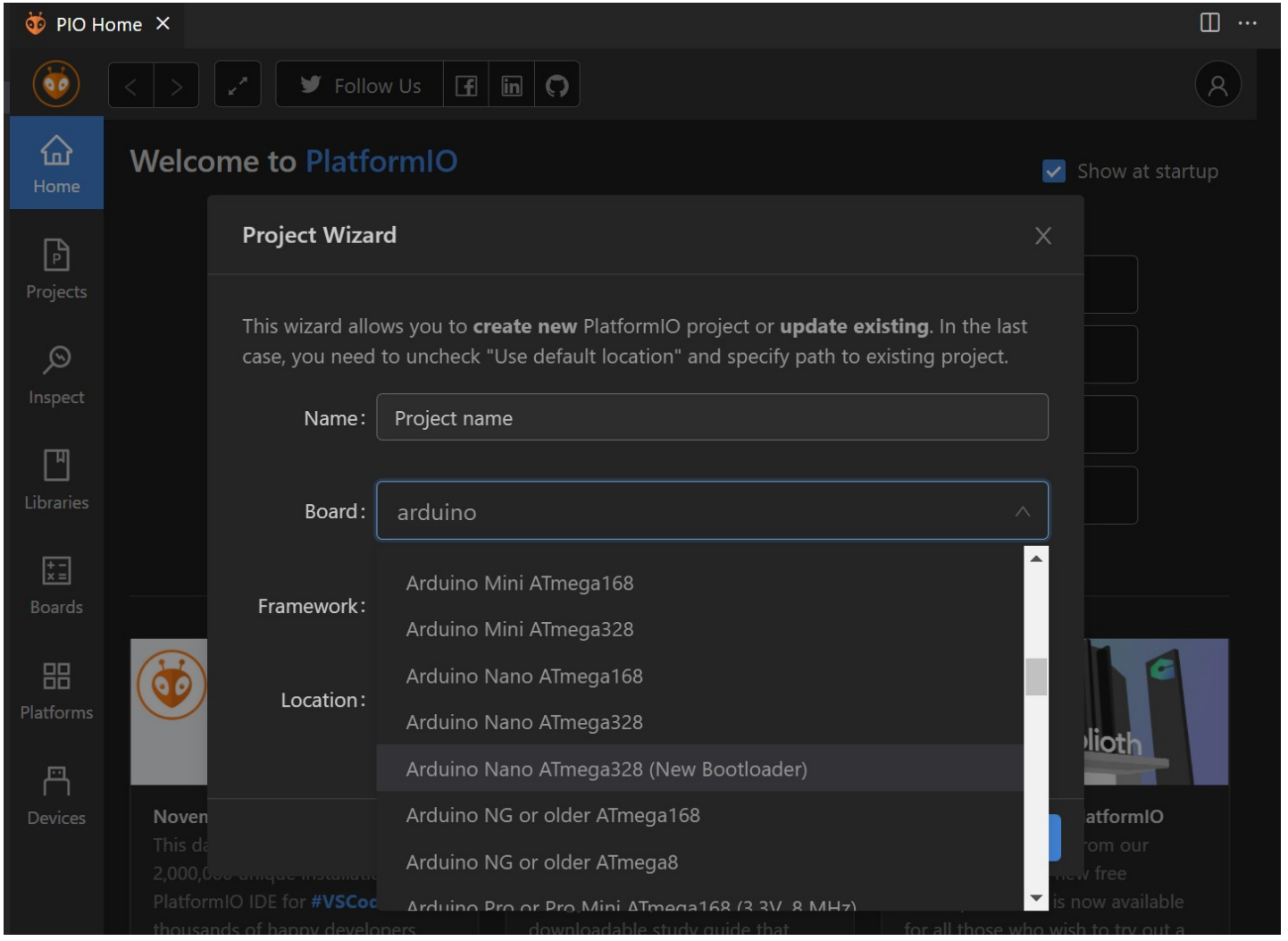


Alternative to Arduino IDE: VS Code IDE + PlatformIO

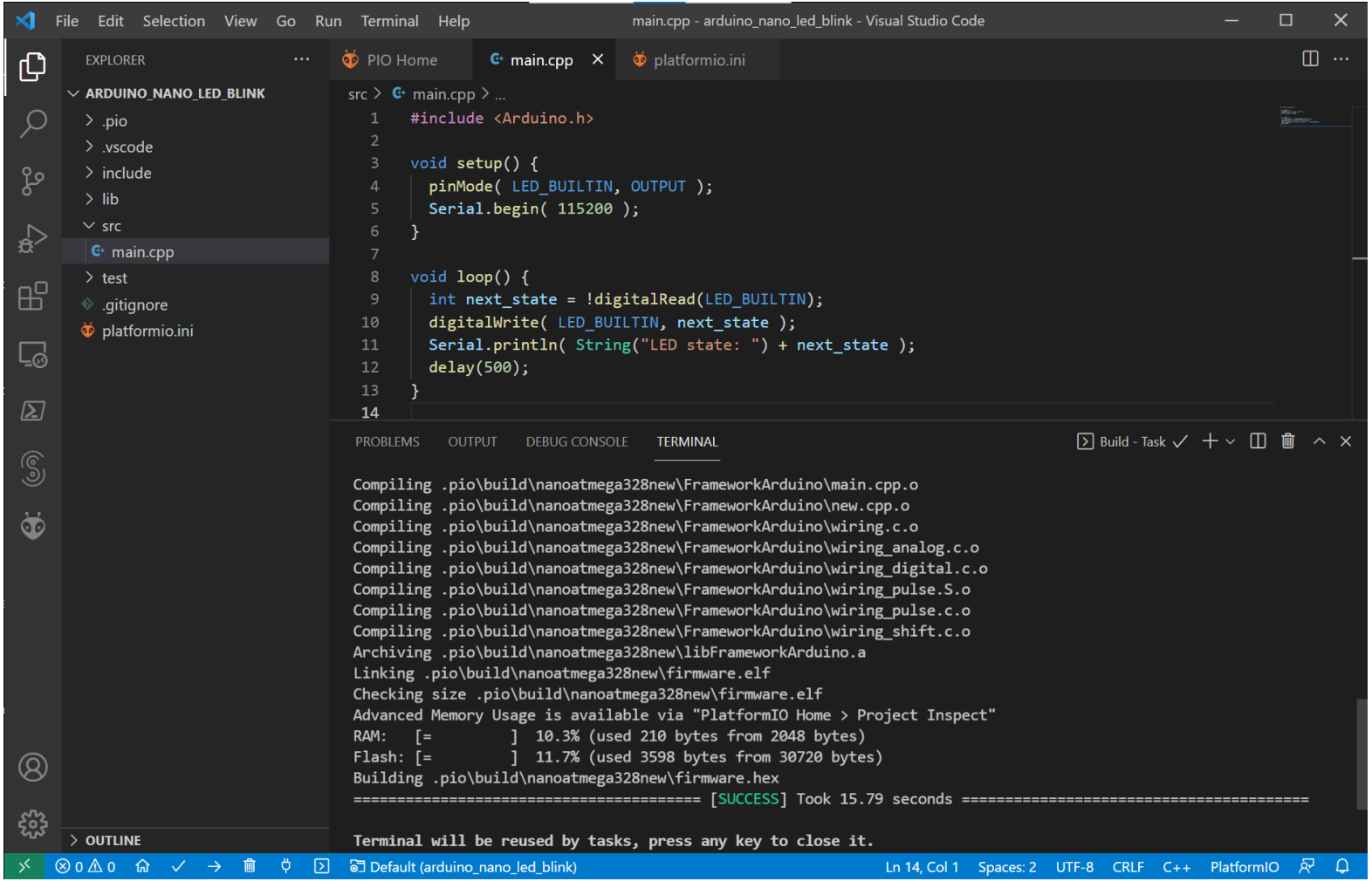


Source: https://iot-kmutnb.github.io/blogs/arduino/vscode_pio_arduino/

Alternative to Arduino IDE: VS Code IDE + PlatformIO



Alternative to Arduino IDE: VS Code IDE + PlatformIO



Families of Atmel AVR MCUs

1) tinyAVR

- Ultra-small, low-pin-count MCUs (8–24 pins).
- Targeted for cost-sensitive, simple applications
- Subseries: **tinyAVR 0/1/2-series (*new*)**, offering different memory sizes, and peripherals.

2) megaAVR

- Larger 8-bit MCUs with more I/O pins, Flash memory, and peripherals.
- Targeted for general-purpose embedded applications (Arduino Uno uses ATmega328P).
- Subseries: **0-series (*new*)**, optimized for energy efficiency and improved peripherals.

3) XMEGA

- High-performance 8-bit / 16-bit MCUs with advanced peripherals: DMA, and USB support.
- Designed for performance-demanding embedded applications.

4) AVR-Dx

- Latest AVR generation with modern peripherals and improved power efficiency, added features like Op-Amps and Multi-Voltage I/O (MVIO),
- Subseries: **AVR-DA / AVR-DB,** designed for automotive, industrial, and security applications.

History of Atmel & AVR

A brief history of the AVR microcontroller

- The basic AVR architecture was started by two students at the **Norwegian Institute of Technology** (NTH): **Alf-Egil Bogen** and **Vegard Wollan**.
- The original AVR MCU was developed at a local ASIC design company in Trondheim, Norway, called **Nordic VLSI** at the time, now Nordic Semiconductor, where Bogen and Wollan were working as students.
- When the technology was sold to Atmel from Nordic VLSI, the internal architecture was further developed by Bogen and Wollan at **Atmel Norway**, a subsidiary of Atmel.
- Atmel launched the first commercial microcontroller based on the AVR architecture, the AT90S family (8-bit), in 1996.

Source: https://en.wikipedia.org/wiki/AVR_microcontrollers

AVR® Microcontrollers Peripheral Integration

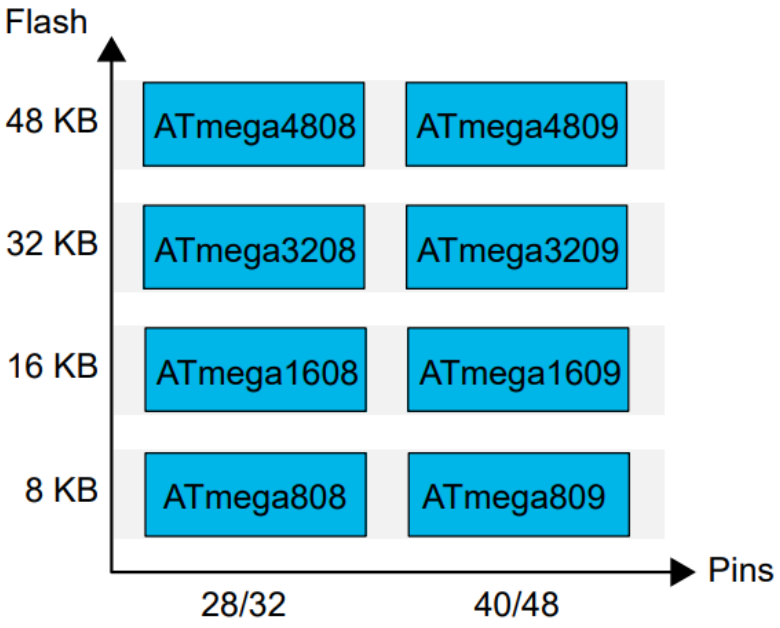
Quick Reference Guide

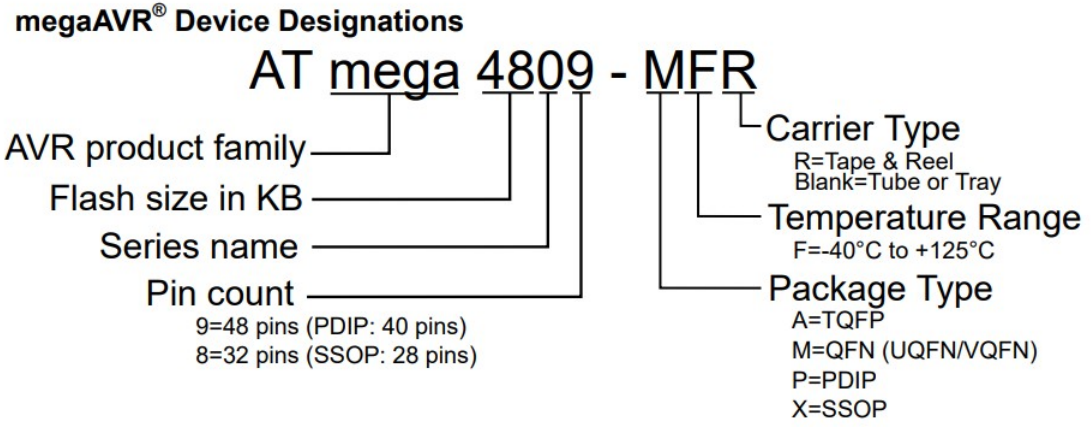
Product Family	Pin Count	Program Flash Memory (KB)	SRAM (KB)	Supply Voltage	Speed (MHz) Single Cycle Instruction: MHz = MIPS	Peripheral Function Focus																																		
						Intelligent Analog								Waveform Control				Timing and Measurements				Logic, Crypto and Math			Safety and Monitoring				Communications						User Interface			System Flexibility					
						ADC (# of bits)	ADC (# of channels)	Comparators	ADC Gain Stage	DAC (# of bits)	Temperature Sensor	Internal Voltage Reference	Zero Cross Detector (ZCD)	8-bit PWM	16-bit PWM	Quadrature Decoder	Waveform Extension (WeX)	Real-Time Counter	8-bit Timer/Counters	12-bit Timer Counter	16-bit Timer/Counter	CCL	MULT	Crypto (AES/DES)	CRC/SCAN	POR	BOD	WDT	USART	USB	I²C	SPI	IRCOM	Serial Number	QTouch® Technology	QTouch Technology with PTC[2]	LCD	External Bus Interface	DMA Channels	Event System	SleepWalking	Sleep Modes	picoPower® Technology
ATtiny4/5/9/10	6	0.5–1	0.032	1.8–5.5	12	10[3]	4[3]	✓							2					1					✓		✓							✓							4	
ATtiny102/104	8/14	1	0.032	1.8–5.5	12	10	5/8	✓				✓			2					2					✓		✓	1						✓							4	
ATtiny13A	8–20	1	0.064	1.8–5.5	20	10	4	✓							2										✓	✓	✓							✓							3	✓
ATtiny20/40	12–20	2/4	0.128/0.256	1.8–5.5	12	10	8/12	✓			✓			2	2				1		1					✓	✓	✓				1	1			✓						4	
ATtiny24A/44A/84A	14–20	2–8	Up to 0.512	1.8–5.5	20	10	8	✓	✓		✓	✓		2	2				1		1					✓	✓	✓				1	1			✓						4	✓
ATtiny48/88	28–32	4/8	Up to 0.512	1.8–5.5	16	10	8	✓			✓	✓		1	1				1		1					✓	✓	✓				1	1									3	✓
ATtiny87/167	20–32	8/16	0.512	1.8–5.5	16	10	11	✓			✓	✓		1	2				1		1					✓	✓	✓	1[8]			1	2									4	
ATtiny261A/461A/861A	20–32	2–8	Up to 0.512	1.8–5.5	20	10	11	✓	✓		✓	✓							1		1					✓	✓	✓				1	1			✓						4	✓
ATtiny20x/40x/80x/160x	8–24	2–16	Up to 1	1.8–5.5	20	10	12	✓			✓	✓			2			✓			1	✓	✓			✓	✓	✓	1[1]			1	1		✓					✓	✓	3	✓
ATtiny21x/41x/81x/161x/321x	8–24	2–32	Up to 2	1.8–5.5	20	10	12	✓		8	✓	✓			2			✓		1	1	✓	✓			✓	✓	✓	1[1]			1	1		✓					✓	✓	3	✓
ATtiny441/841	14–20	4/8	Up to 0.512	1.7–5.5	16	10	12	✓	✓		✓			1	2				1		2					✓	✓	✓	2			1	1									4	✓
ATtiny2313A	20	2	0.128	1.8–5.5	20	–	–	✓				✓		2	2				1		1					✓	✓	✓	1			1	2									3	✓
ATmega8A/16A/32A	28–44	8–32	1–2	2.7–5.5	16	10	8	✓						2	1			✓	2		1		✓			✓	✓	✓	1			1	1			✓						5	
ATmega8U2/16U2/32U2	32	8–32	0.5–1	2.7–5.5	16	–	–	✓			✓	✓		4	6			✓	2		3		✓			✓	✓	✓	2	✓		2	2									6	
ATmega16U4/32U4	32	16/32	1/2	2.7–5.5	16	10	12	✓			✓	✓		5					1		1		✓			✓	✓	✓	1	✓			1									6	
ATmega48PB/88PB/168PB/328PB	32	4–32	0.5–2	1.8–5.5	20	10	8	✓			✓	✓		4	2/6[6]			✓	2		1/3[6]		✓			✓	✓	✓	1/2[8]			1/2[8]	1/2[8]			✓	✓[4]					6	
ATmega80x/160x/320x/480x	28–48	8–48	1–6	1.8–5.5	20	10	16	✓			✓	✓		4	3			✓			5	✓	✓		✓	✓	✓	✓	4			1	1		✓					✓	✓	3	✓
ATmega64A/128A	64	64–128	4	2.7–5.5	16	10	8	✓	✓			✓		2	6				2		2		✓			✓	✓	✓	2			1	1		✓							6	
ATmega164PA/324PA/644PA/1284P	44	16–128	1–16	1.8–5.5	20	10	8	✓	✓			✓		4	2/2/4			✓	2		1/1/2		✓			✓	✓	✓	2			1	1		✓							6	✓
ATmega165PA/325PA/645P	44	16–64	1–4	1.8–5.5	16	10	8	✓				✓		4	6			✓	2		3		✓			✓	✓	✓	3			2	2									6	✓
ATmega169PA/329PA/649P	64	16–64	1–4	1.8–5.5	16	10	8	✓				✓		2	2			✓	2		1		✓			✓	✓	✓	1			1	1		✓		✓					5	
ATmega324PB	44	32	2	1.8–5.5	20	10	8	✓				✓		2	2			✓	2		1		✓			✓	✓	✓	1			1	1		✓	✓						5	
ATmega640/1280/2560/1281/2561	64–100	64–256	8	1.8–5.5	16	10	8/16	✓	✓			✓		4	6/12			✓	2		4		✓			✓	✓	✓	2/4			1	1		✓				✓[5]			6	
ATmega3290PA/6490P	100	32–64	2–4	1.8–5.5	20	10	8	✓	✓			✓		2	2			✓	2		1		✓			✓	✓	✓	1			1	1		✓		✓					5	
ATmega3250PA/6450P	100	32–64	2–4	1.8–5.5	20	10	8	✓	✓			✓		2	2			✓	2		1		✓			✓	✓	✓	1			1	1		✓							5	
AVR-DA Family	28-64	32-128	4–16	1.8–5.5	24	12	12	✓		10	✓	✓	1-3	9–17	3–6			✓		1	1–5	✓	✓		✓	✓	✓	✓	3–6		1–2	2	✓	✓		✓			✓	✓	3	✓
ATxmega A1U/A3U/A4U Family	44-100	16–128	2–8	1.6–3.6	32	12	12/16	✓	✓	12	✓	✓			5-8	✓	✓	✓			5-8		✓	✓		✓	✓	✓	5-8	✓	2-4	2-4	✓		✓			✓	4	✓		5	✓
ATxmega B1/B3 Family	64–100	64–128	4–8	1.6–3.6	32	12	8	✓	✓		✓	✓			2/3	✓	✓	✓			2/3		✓	✓		✓	✓	✓	1/2	✓	1	1	✓	✓	✓		✓		2	✓		5	✓
ATxmega C3/D3/C4/D4 Family	44-64	16–384	2–32	1.6–3.6	32	12	12/16	✓	✓		✓	✓			4/5	✓	✓	✓			4/5		✓	✓		✓	✓	✓	2/3	✓(7)	2	2	✓	✓	✓					✓		5	✓
ATxmega32E5 Family	32	8–32	1–4	1.6–3.6	32	12	16	✓	✓	12	✓	✓			3	✓	✓	✓			3	✓	✓	✓		✓	✓	✓	2		1	1	✓	✓	✓				4	✓		5	✓

1: LIN port also **2:** Peripheral Touch Controller **3:** Only on the ATtiny5/10 **4:** Not on the ATtiny212/214/412/414/416 **5:** Only on the ATmega1281/2561 **6:** Only on the ATmega328PB **7:** Only on the C3 and C4 **8:** UART only LIN Port also

Source: https://ww1.microchip.com/downloads/en/DeviceDoc/30010135E.pdf

megaAVR 0-Series Devices





Memory Type	ATmega80x	ATmega160x	ATmega320x	ATmega480x
Flash	8 KB	16 KB	32 KB	48 KB
SRAM	1 KB	2 KB	4 KB	6 KB
EEPROM	256B	256B	256B	256B
User row	32B	32B	64B	64B

Devices with different Flash memory size typically also have different SRAM and EEPROM.

megaAVR 0-Series Devices

New megaAVR MCUs

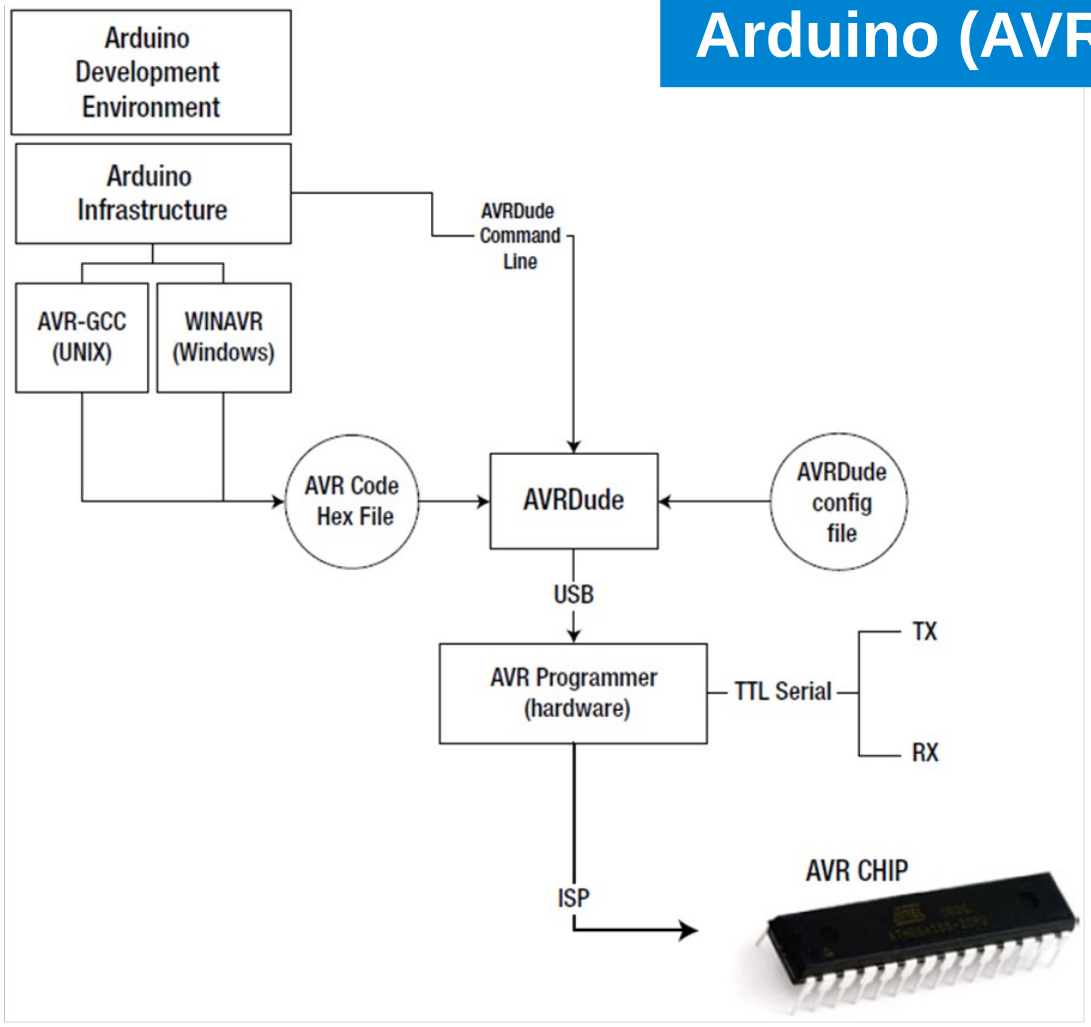
Part Number	Program Flash (KB)	EEPROM (B)	Data SRAM (KB)	I/O Pins	10-bit ADC (ch)	5-bit DAC	Comps	8-bit/16-bit Timers	Window Watchdog Timer	8-bit/16-bit PWM	Int RCO	CCL	Temp Sensor and Low Power	USART/I²C/SPI	Packages
ATMEGA4809	48	256	6	41	16	–	1	–/6	Y	4/3	32 KHz–20 MHz	1	Y	4/1/1	UQFN, TQFP
ATMEGA4808	48	256	6	27	12	–	1	–/5	Y	4/3	32 KHz–20 MHz	1	Y	3/1/1	VQFN, TQFP
ATMEGA4808	48	256	6	23	8	–	1	–/5	Y	4/3	32 KHz–20 MHz	1	Y	3/1/1	SSOP
ATMEGA3209	32	256	4	41	16	–	1	–/6	Y	4/3	32 KHz–20 MHz	1	Y	4/1/1	UQFN, TQFP
ATMEGA3208	32	256	4	27	12	–	1	–/5	Y	4/3	32 KHz–20 MHz	1	Y	3/1/1	VQFN, TQFP
ATMEGA3208	32	256	4	23	8	–	1	–/5	Y	4/3	32 KHz–20 MHz	1	Y	3/1/1	SSOP

Note: **Arduino Nano Every** and **Arduino Uno WiFi Rev 2** use the **ATmega4809** as the main MCU.

AVR MCUs: Loss of Popularity

- **AVR microcontrollers** became widely popular with the introduction of the **first-generation Arduino boards** (e.g. Uno, Nano, etc.).
  However, newer generations of Arduino hardware now use 32-bit MCUs instead of the original 8-bit AVR chips.
- After **Microchip acquired Atmel in 2016**, several factors contributed to AVR's reduced popularity:
  - **Microchip's PIC dominance:** Microchip already had a large product line of PIC MCUs and started favoring PIC promotion over AVR, especially in marketing and ecosystem support.
  - **Ecosystem integration:** Atmel software tools were replaced with Microchip tools; **Atmel Studio IDE** is now replaced with **MPLAB X IDE**.
  - **32-bit trend:** The embedded market is shifting toward 32-bit ARM Cortex-M MCUs, offering more performance, peripherals, and memory at similar cost.

Arduino (AVR) Build Process



Alan Trevennor, *Practical AVR Microcontrollers -- Games, Gadgets, and Home Automation with the Microcontroller Used in the Arduino*, 2012.

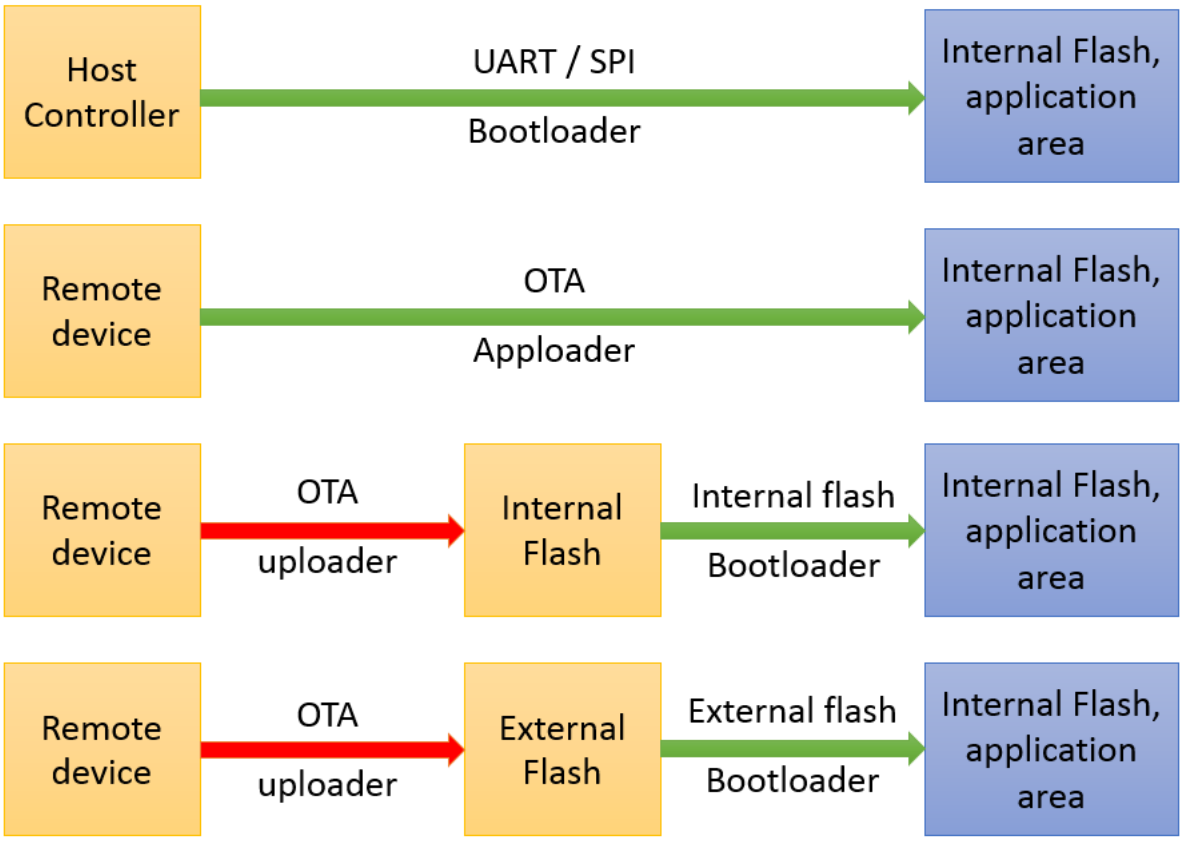
What is a Bootloader ?

- A **bootloader** is a small firmware program that runs right after reset and is mainly used to load, update, verify, or recover the main application firmware without external hardware programmers.
- Some MCUs include **factory ROM bootloaders** that are fixed and cannot be erased (e.g., built-in UART/USB bootloaders).
- Many MCUs allow **custom bootloaders**, letting developers implement their own update protocols, fail-safe logic, or multi-image selection.
- **Arduino bootloaders** are typical user-programmable bootloaders that can easily be reflashed or replaced.

What is a Bootloader ?

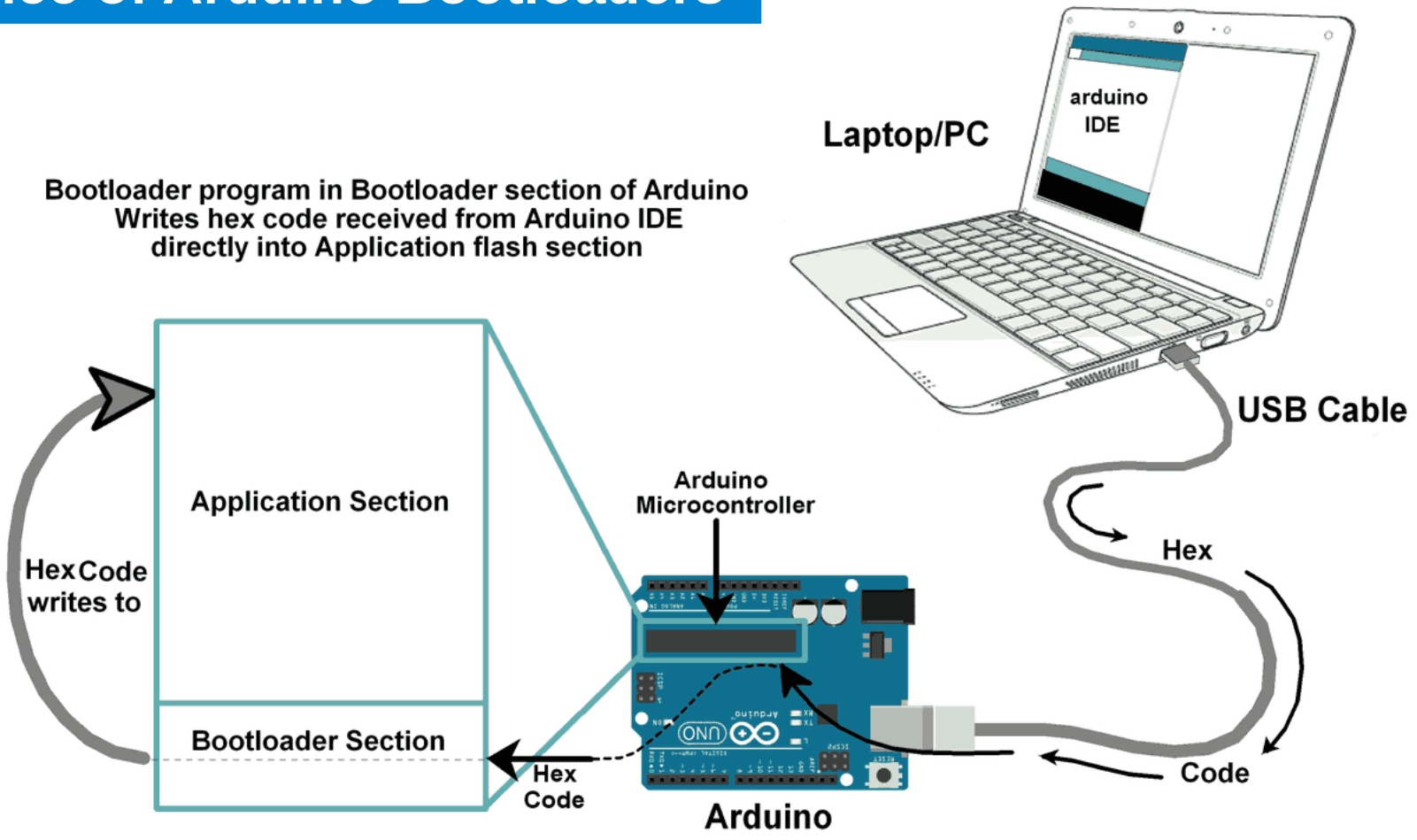
- Bootloaders can communicate through different **interfaces**:
  - UART / Serial
  - USB (such the DFU protocol)
  - SPI bus
  - CAN (Controller Area Network) bus
  - Ethernet / LAN or Wireless (OTA: Over-the-Air)
- Modern MCUs also support **secure boot**, such as in ESP32 and STM32 devices.
  - Secure boot verifies the authenticity and integrity of firmware before execution, usually using cryptographic signatures (RSA, ECC, or SHA-256 hashes).
  - It prevents booting unauthorized, modified, or malicious firmware—crucial for IoT security, preventing malware injection, and protecting intellectual property.
  - Often paired with flash encryption, making firmware unreadable to attackers even with physical access.

Bootloader Interfaces



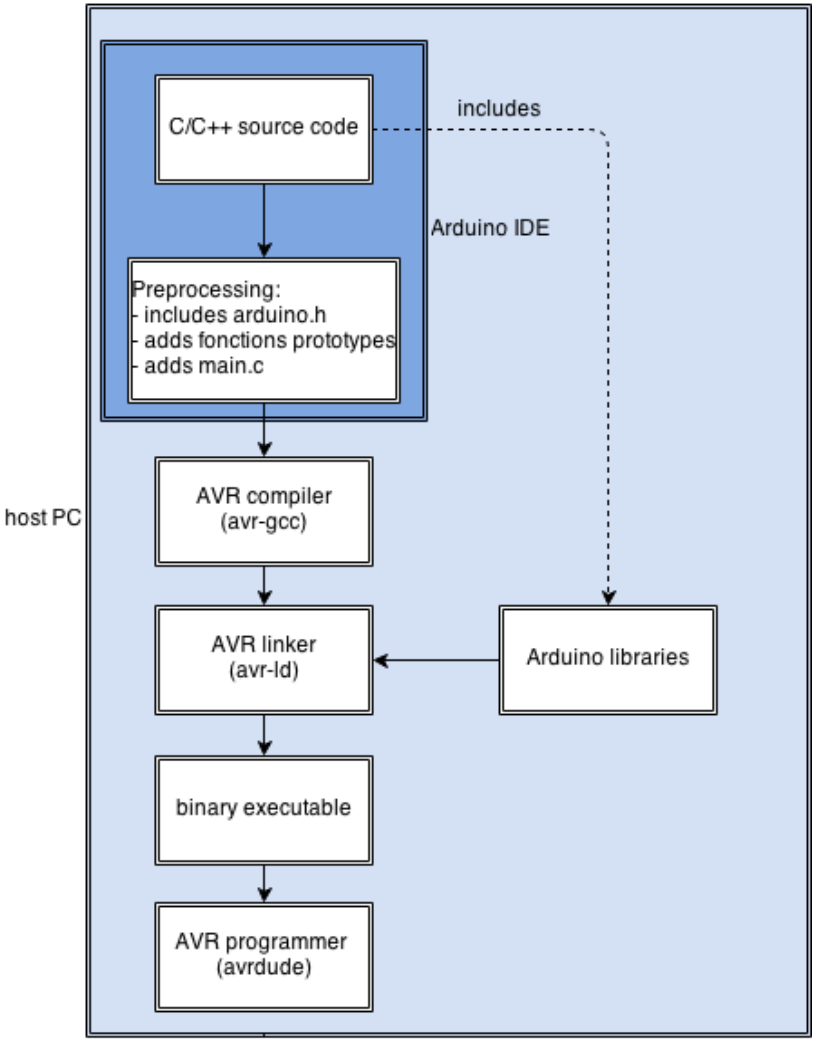
Source: Silicon Labs

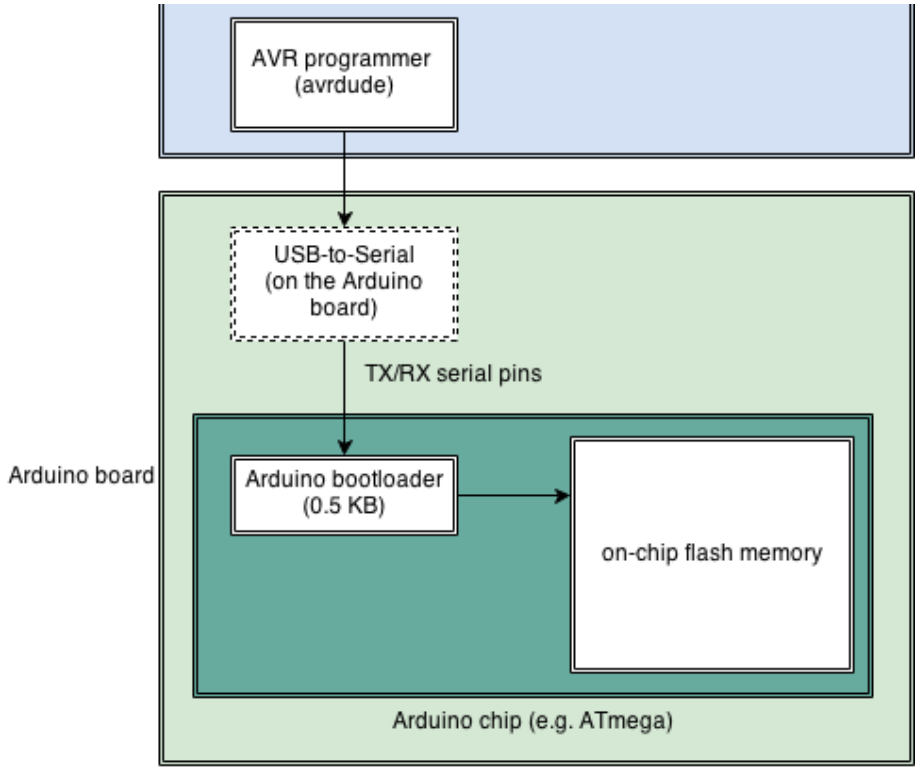
Importance of Arduino Bootloaders



https://www.electronicwings.com/arduino/basics-to-developing-bootloader-for-arduino

Arduino Software Flow



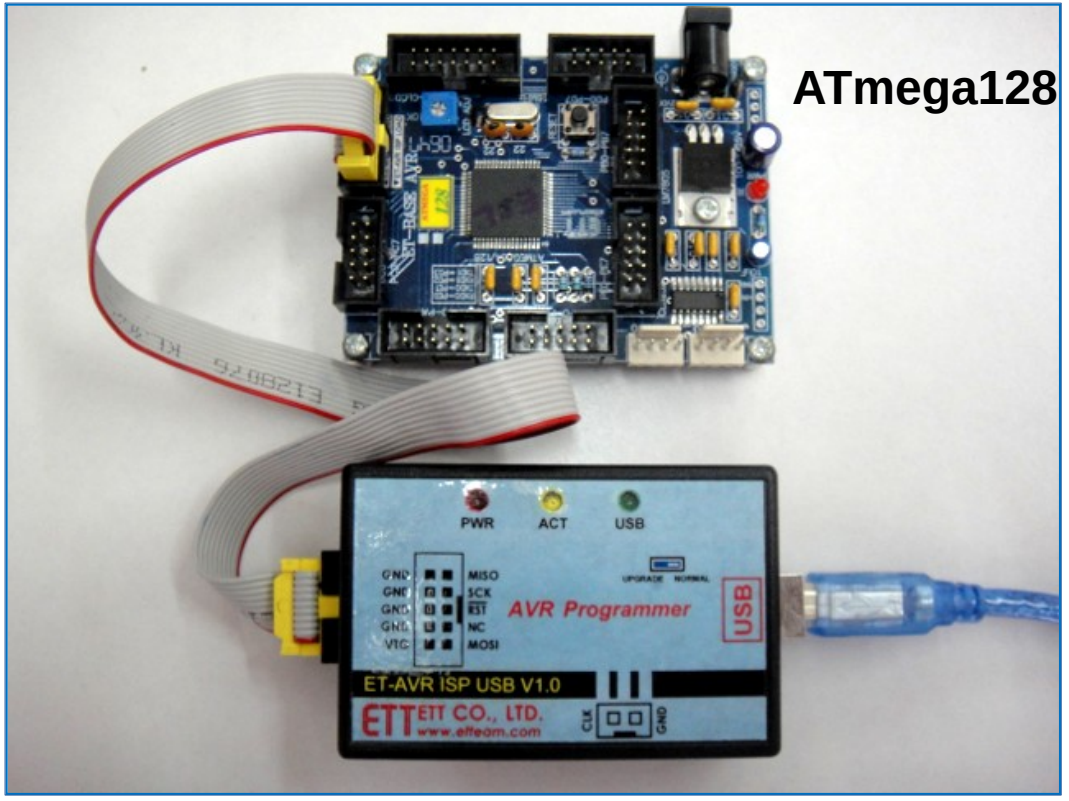


Source: https://jheyman.github.io/blog/pages/ArduinoUnderTheHood/

Importance of Arduino Bootloaders

- Without a bootloader, ATmega-based boards require an **external programmer** using the **ICSP (In-Circuit Serial Programming) interface**.
- Commercial **AVR programmers/debuggers** are often expensive and not convenient for beginners.
  - Examples include the **Atmel-ICE**, **AVRISP mkII / STK500**, and **JTAGICE3**, which are typically priced much higher than an entry-level Arduino board.
- The inclusion of the Arduino bootloader made Arduino boards very popular in their early days, because users could upload code directly via USB without needing a separate programmer.

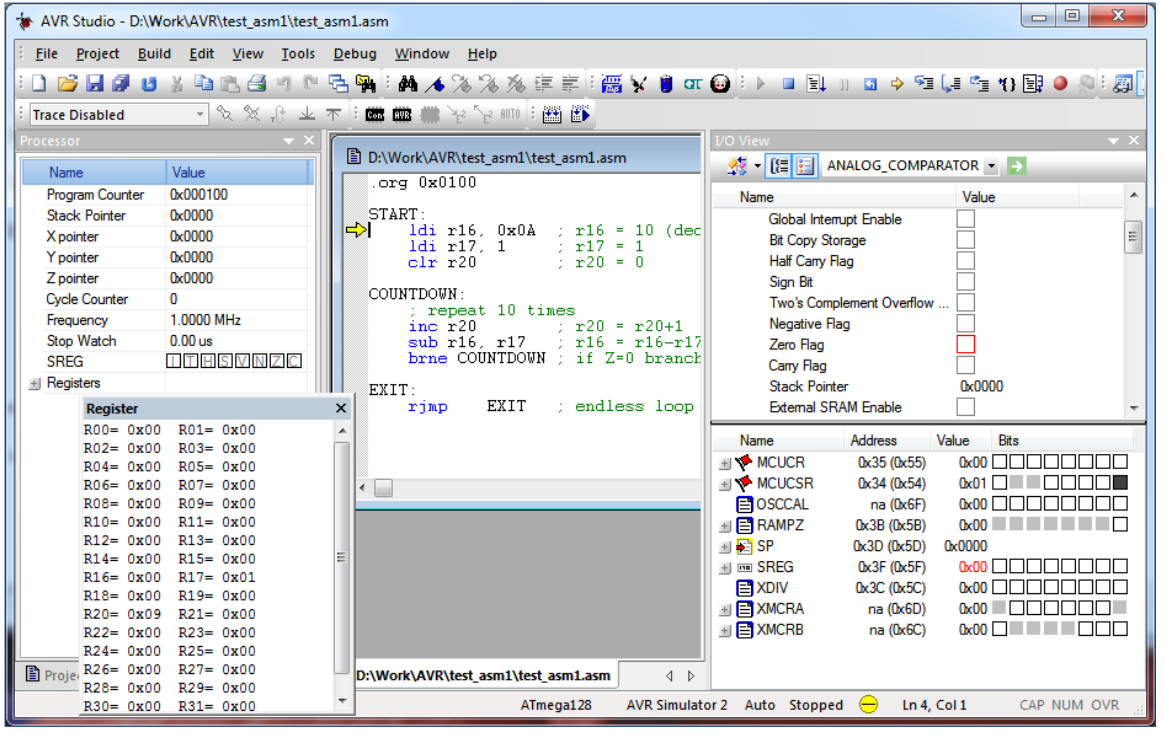
Examples of AVR Board + Programmer in Early Days



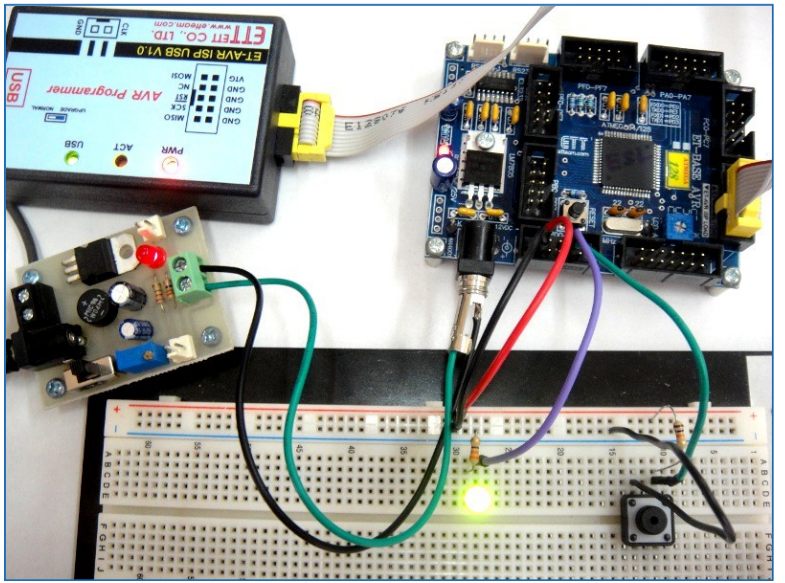
Before Arduino Era

- *In those early days, it was standard practice to use a separate dedicated programmer (often a device, like the **AVR ISP programmer**) to flash the code onto a bare AVR development board.*
- *Modern boards (like Arduino) have the programmer (a small USB-to-serial chip) built right onto the board, making the separate programmer setup a relic of the past.*

Examples of AVR Board + Programmer in Early Days



AVR Studio 4



Examples of AVR Programmers / Debuggers

Atmel-ICE

AVR MKII

USBasp

https://www.microchip.com/en-us/development-tool/ATATMEL-ICE

Examples of AVR Programmers / Debuggers

ATATMEL-ICE Microchip Techno

digikey.co.th/th/products/detail/microchip-technology/ATATMEL-ICE/4753379

Microchip Technology ATATMEL-ICE

ภาพที่ปรากฏเป็นเพียงภาพตัวอย่างเท่านั้น ควรขอข้อมูลจำเพาะที่แน่นอนจากเอกสารข้อมูลผลิตภัณฑ์

ATATMEL-ICE

หมายเลขผลิตภัณฑ์ DigiKey	ATATMEL-ICE-ND
ผู้ผลิต	Microchip Technology

ในสต็อก: 501

ตรวจสอบสินค้าที่กำลังจะเข้าคลังเพิ่มเติม

จำนวน

เพิ่มไปยังรายการ | เพิ่มไปยังรถเข็น

ราคาทั้งหมดแสดงเป็นสกุลเงิน THB

กล่อง

จำนวน	ราคาต่อหน่วย	ต่อราคา
1	฿3,314.68000	฿3,314.68

แพ็กเกจมาตรฐานผู้ผลิต

ความคิดเห็น

ต้องการความช่วยเหลือ?

Last access: 2025-11-15

Importance of Arduino Bootloaders

- **Arduino boards** with **ATmega microcontrollers** such as the **Arduino Uno, Nano, and Mega** require an Arduino bootloader to upload sketches through the USB-serial interface.
- The Arduino bootloader resides in a **protected boot section** of the ATmega's on-chip Flash memory.
  - It initializes the microcontroller after reset, communicates with the upload tool (such as **avrdude**), configures the upload protocol, and then jumps to the user application code.
  - It also sets **fuse bits** that specify memory layout, clock source, and bootloader behavior.
- There are **different versions of the Arduino bootloader** for different Arduino board models (e.g. **Optiboot for Arduino Uno**).
- Many Arduino boards—whether based on 8-bit AVR MCUs or 32-bit ARM/other MCUs — require a bootloader in order to work seamlessly with the Arduino IDE upload process.

Arduino Bootloader – Overview of Function and Operation

The main functions of the Arduino bootloader are as follows:

- Initialize the microcontroller after reset and configure the boot process.
- Communicate with the upload tool (e.g., **avrdude**) over the serial/USB interface.
- Receive the compiled sketch (a binary file) and write it into the application section of Flash memory.
- Set or rely on fuse settings that define the memory layout and bootloader size.
- Start the user program by jumping to **main()**, which eventually calls the **setup()** and **loop() functions** of the **Arduino Sketch**.

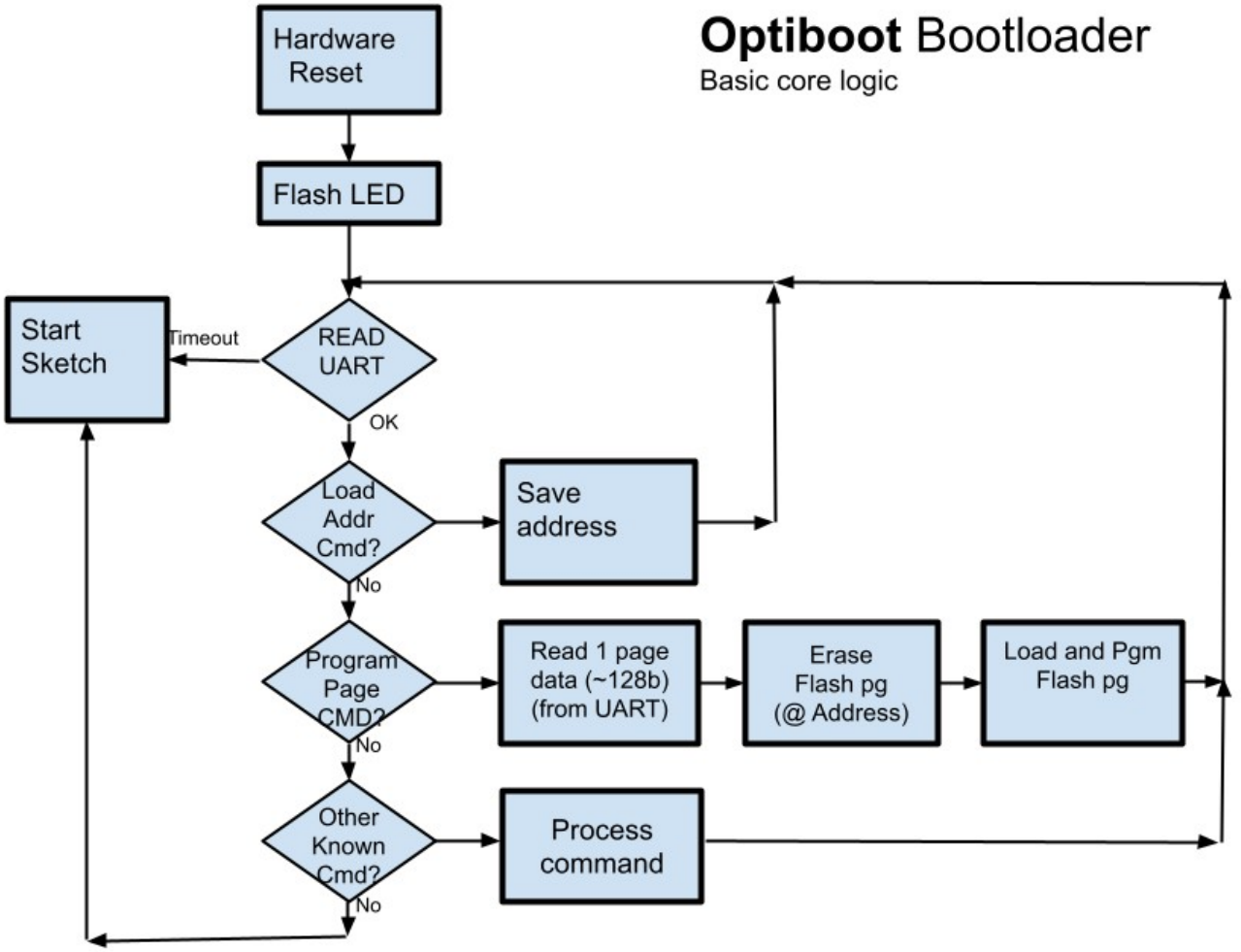
Optiboot Operation

- On reset, **Optiboot** checks the reset cause in the **MCUSR** register.
  If it is not an external reset, it immediately jumps to the **user application.**
- If it is an **external reset**, **Optiboot** runs the **bootloader sequence**:
  - Flashes the start LED (pin and number of flashes are configurable).
  - Configures the selected UART and enables the watchdog timer (1-second timeout).
  - Listens for valid commands on the UART; receiving valid data resets the watchdog and programs the application Flash.
- If no valid data is received — or once programming is complete — the **watchdog (WDT)** times out, forcing a reset.
- Because the **watchdog reset** is not an external reset, the MCU then starts the **user application** with all registers in their normal reset state (except the MCUSR and SP registers).

Optoboot source code (written in C with inline Assembly code:

https://github.com/arduino/ArduinoCore-avr/blob/master/bootloaders/optiboot/

Optiboot Bootloader - Flowchart



ATmega Fuse Bit Programming

- Managing **fuse bits** of the AVR MCU is a crucial part of bootloader programming.
- ATmega microcontrollers use **three fuse bytes** (low, high, and extended).
- Warning: An incorrect fuse configuration can brick the microcontroller by disabling essential functions such as the clock source or the reset pin.
- Fuse bits control several fundamental behaviors of the MCU, including:
  - Selecting the clock source and operating frequency
  - Configuring the brown-out detection threshold
  - Enabling or disabling the bootloader section
  - Defining the size of the bootloader memory area (from 256 to 2048 words / 512–4096 bytes)
  - Enabling or disabling serial programming (ISP)
  - Preserving EEPROM data during sketch upload

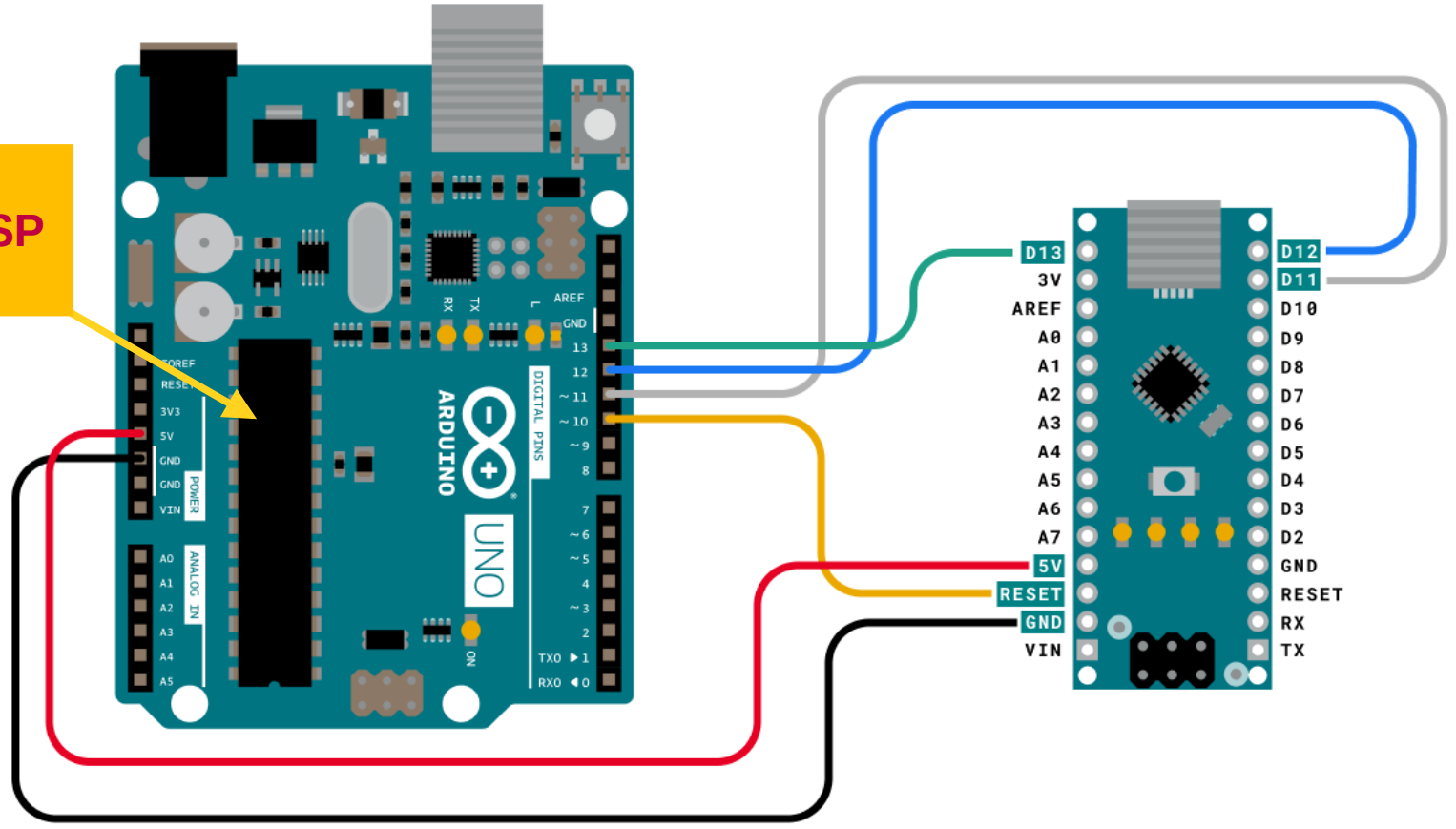
Detailed descriptions of all fuse bits can be found in the datasheet of each specific MCU.

Arduino Bootloader Installing

- If the Arduino bootloader is overwritten, it is easy to **reprogram** it.
- You can use one **Arduino board** to flash the bootloader onto another board. In this setup, the programming board runs the **ArduinoISP** sketch.
- Alternatively, you can use a **low-cost AVR programmer** such as **USBasp**, which provides a simple and inexpensive method for burning the bootloader.
- Arduino bootloaders source code and firmware files:
  - https://github.com/arduino/ArduinoCore-avr/tree/master/bootloaders

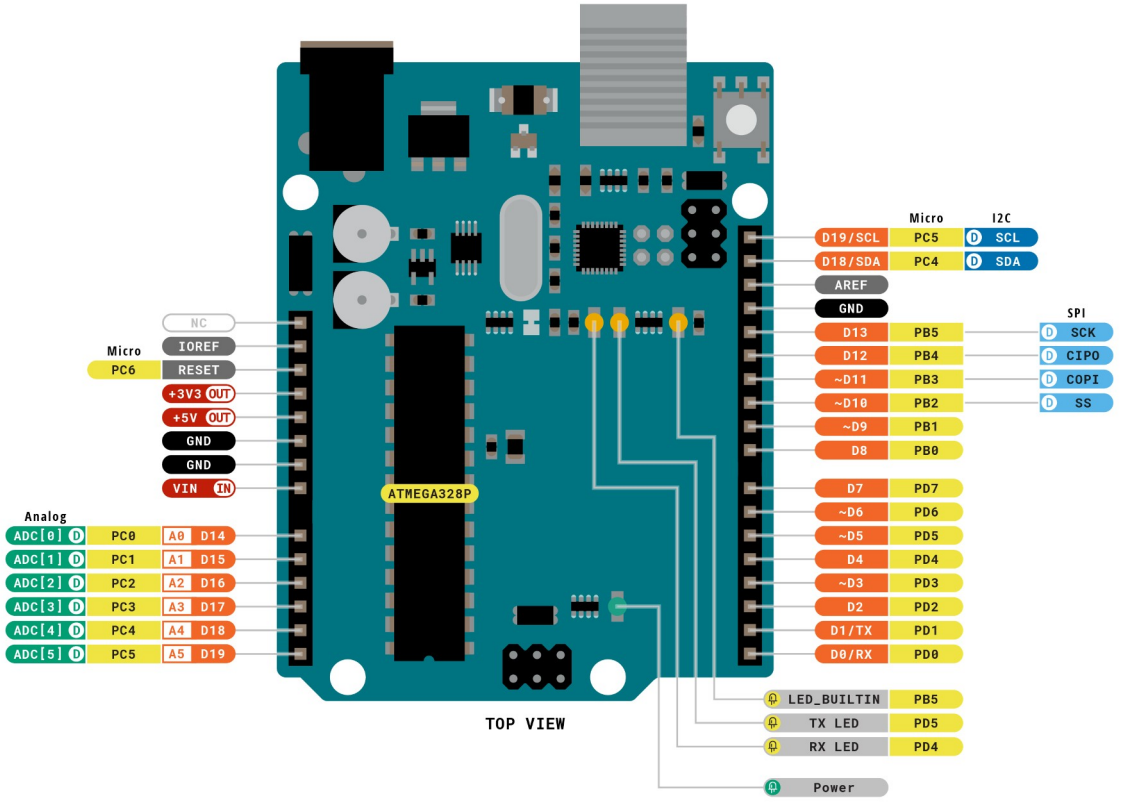
Using Arduino Uno as Programmer

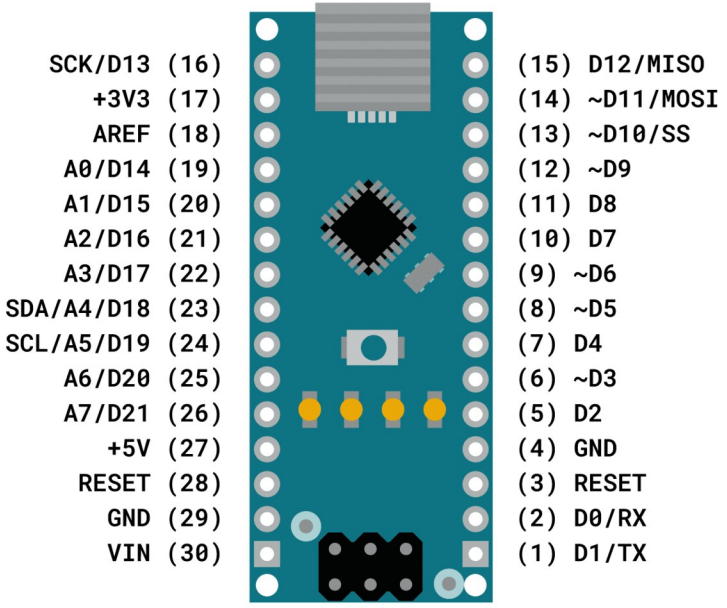
Install ArduinoISP sketch



Source: https://support.arduino.cc/hc/en-us/articles/4841602539164-Burn-the-bootloader-on-UNO-Mega-and-classic-Nano-using-another-Arduino

Arduino Uno and Nano: Pinout





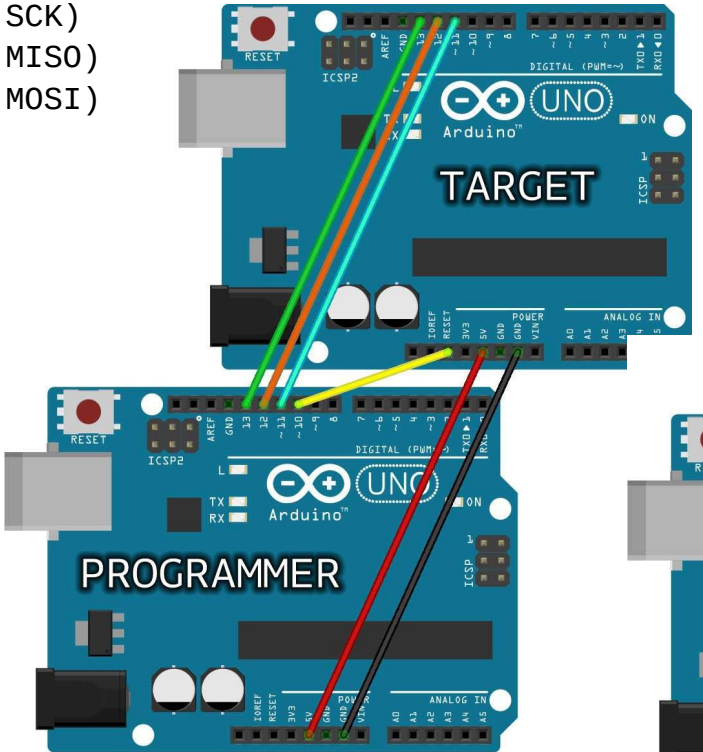
Legend:
Power
Ground
Digital
Analog
Main Part
I2C
SPI
Analog

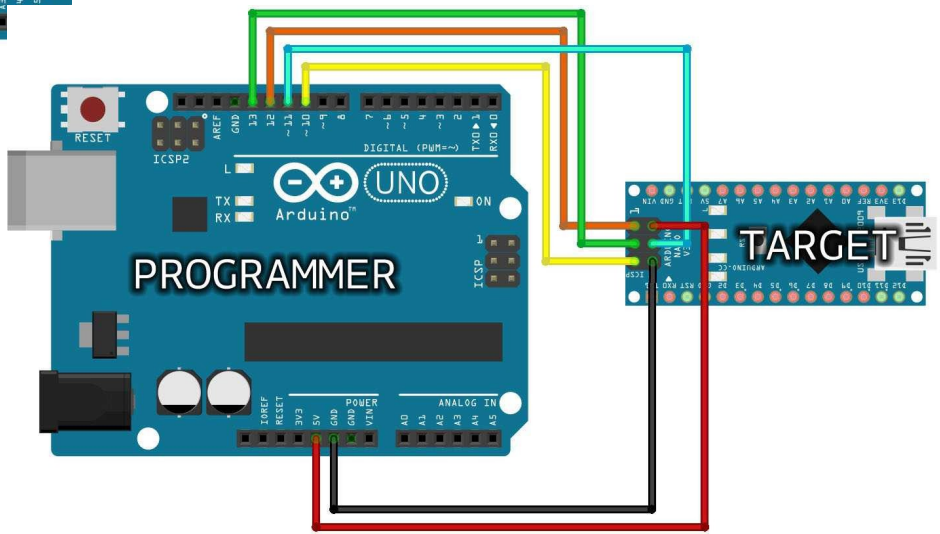
ARDUINO

ARDUINO UNO REV3
SKU code: A000066
Pinout
Last update: 6 Oct, 2022

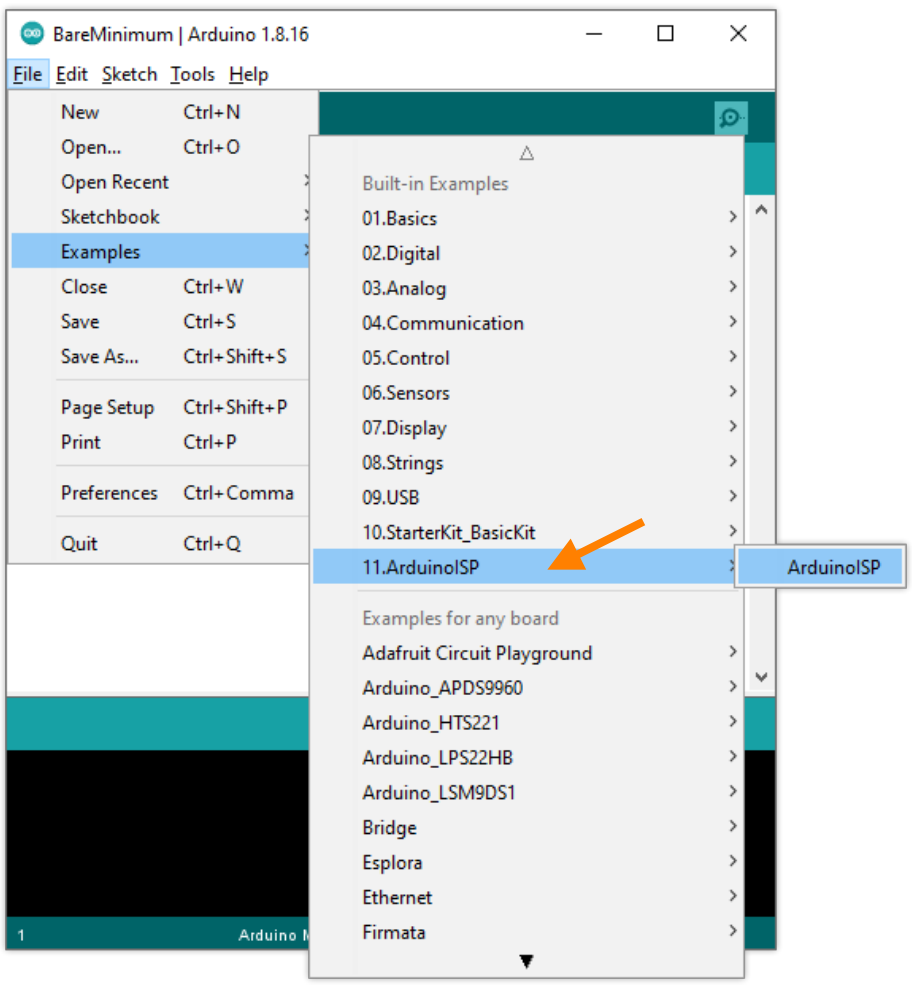
Using Arduino Uno as Programmer

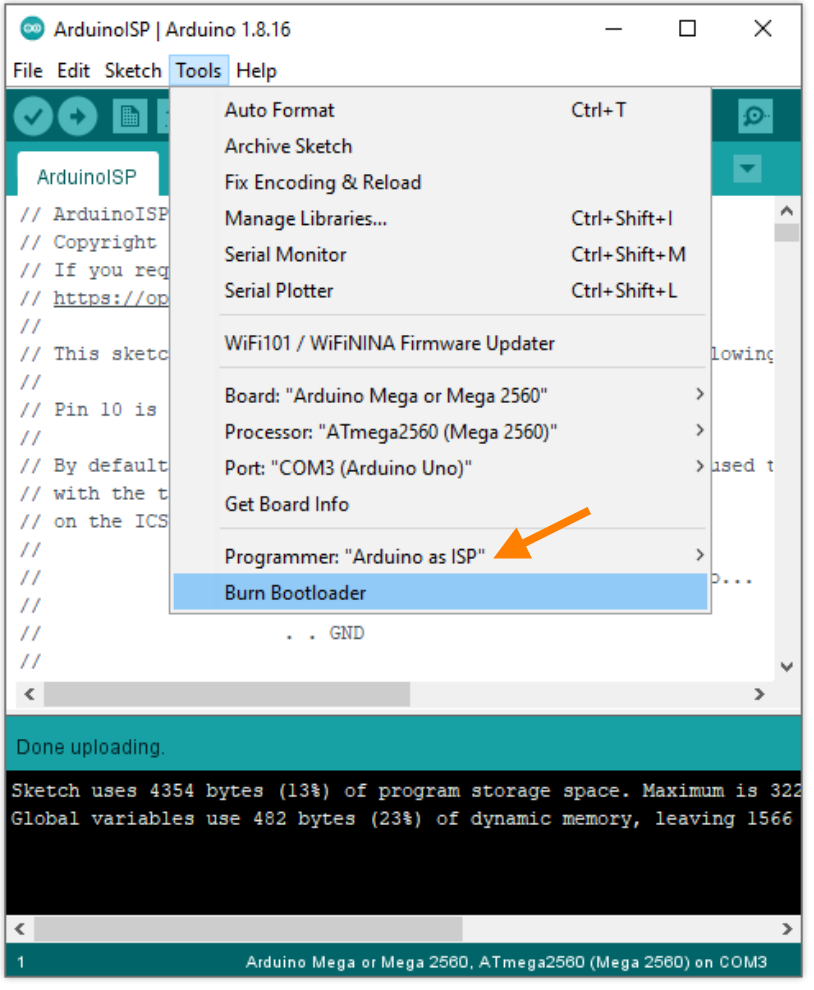
ArduinoISP	ArduinoTarget
D13	--> D13 (SPI SCK)
D12	<-- D12 (SPI MISO)
D11	--> D11 (SPI MOSI)
D10	--> /RESET
5V	--> 5V
GND	<-> GND





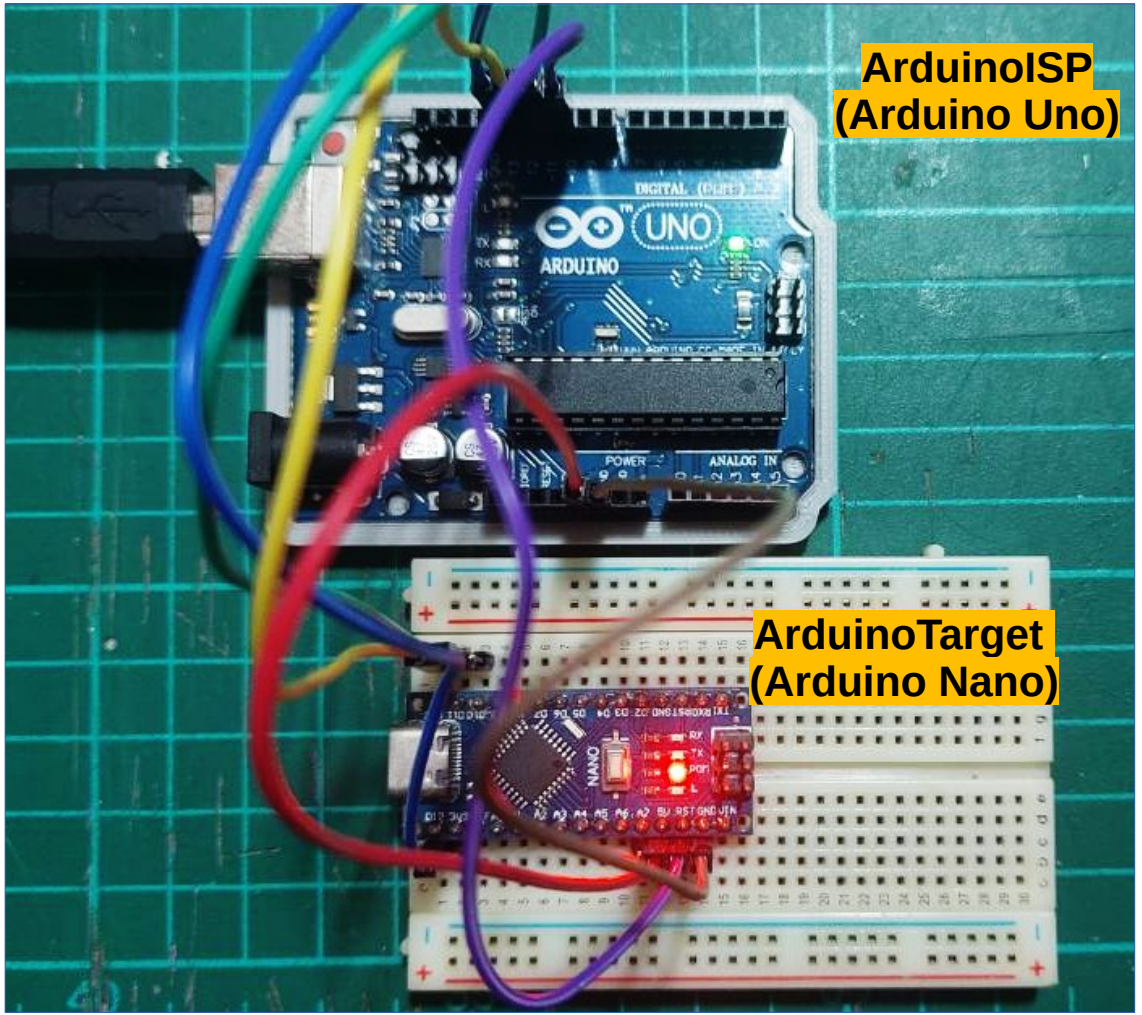
Using Arduino Uno as Programmer





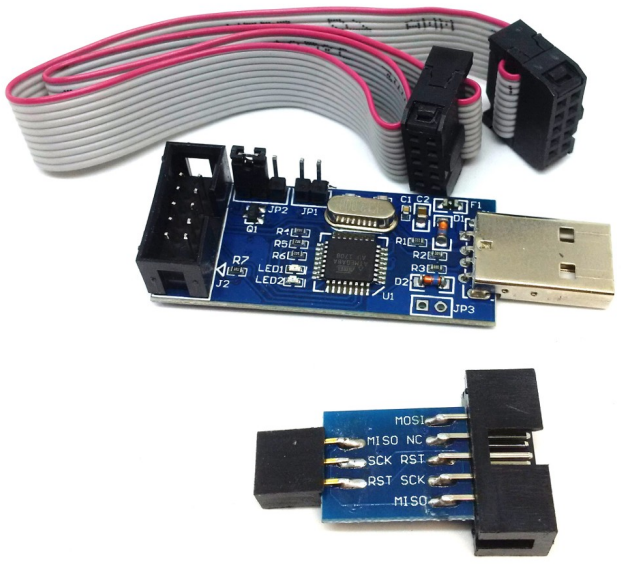
Using Arduino Uno as Programmer

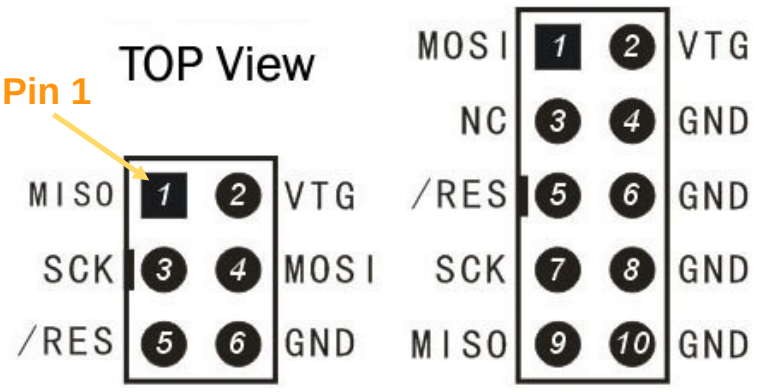
```
ArduinoISP   ArduinoTarget
-----------------------------
D13  --> D13 (SPI SCK)
D12  <-- D12 (SPI MISO)
D11  --> D11 (SPI MOSI)
D10  -->  RESET/
5V   --> 5V
GND  <-> GND
```



ICSP Connectors / Pin Layout

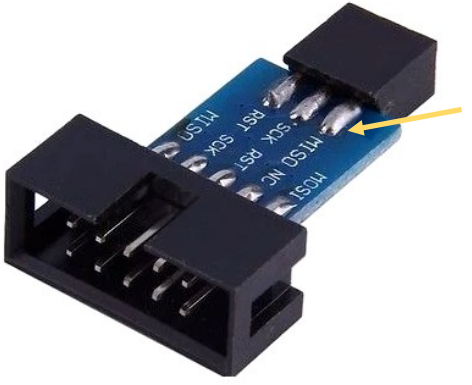
USBasp



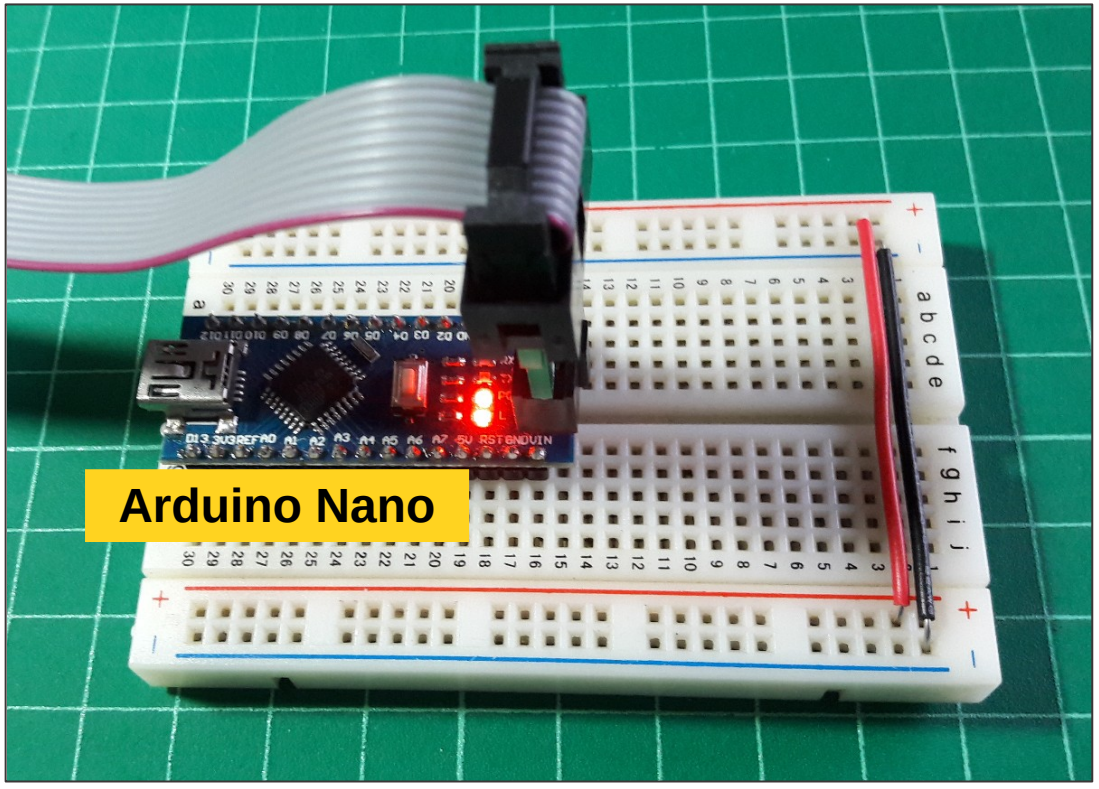


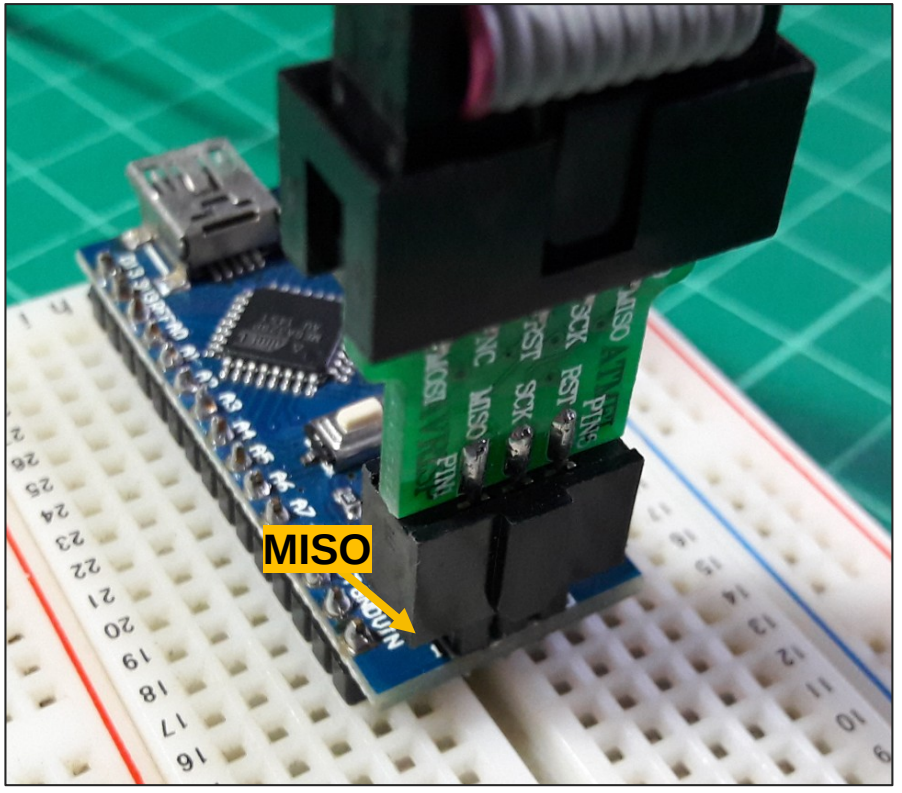
Arduino / AVR ICSP Header

Adapter

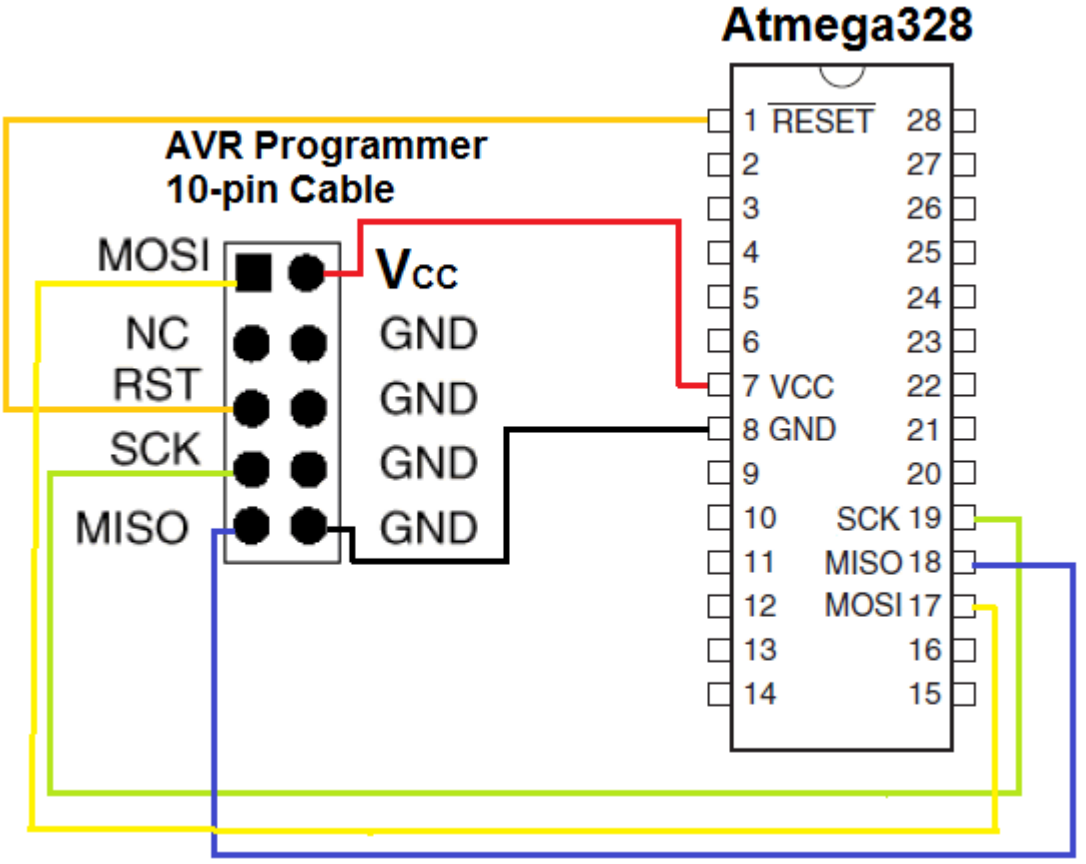


ATmega328P – Programming Wiring



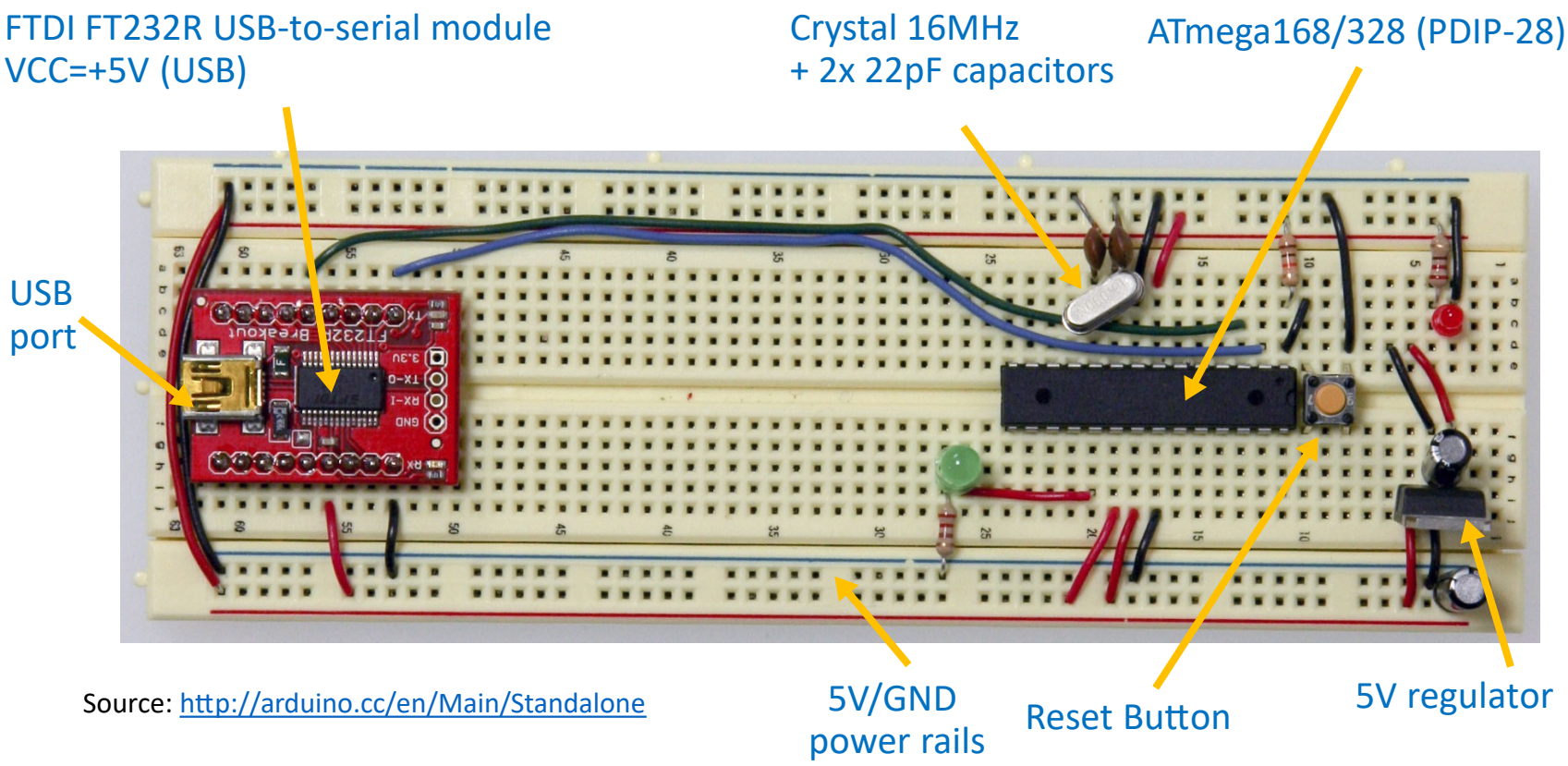


ATmega328P – Programming Wiring



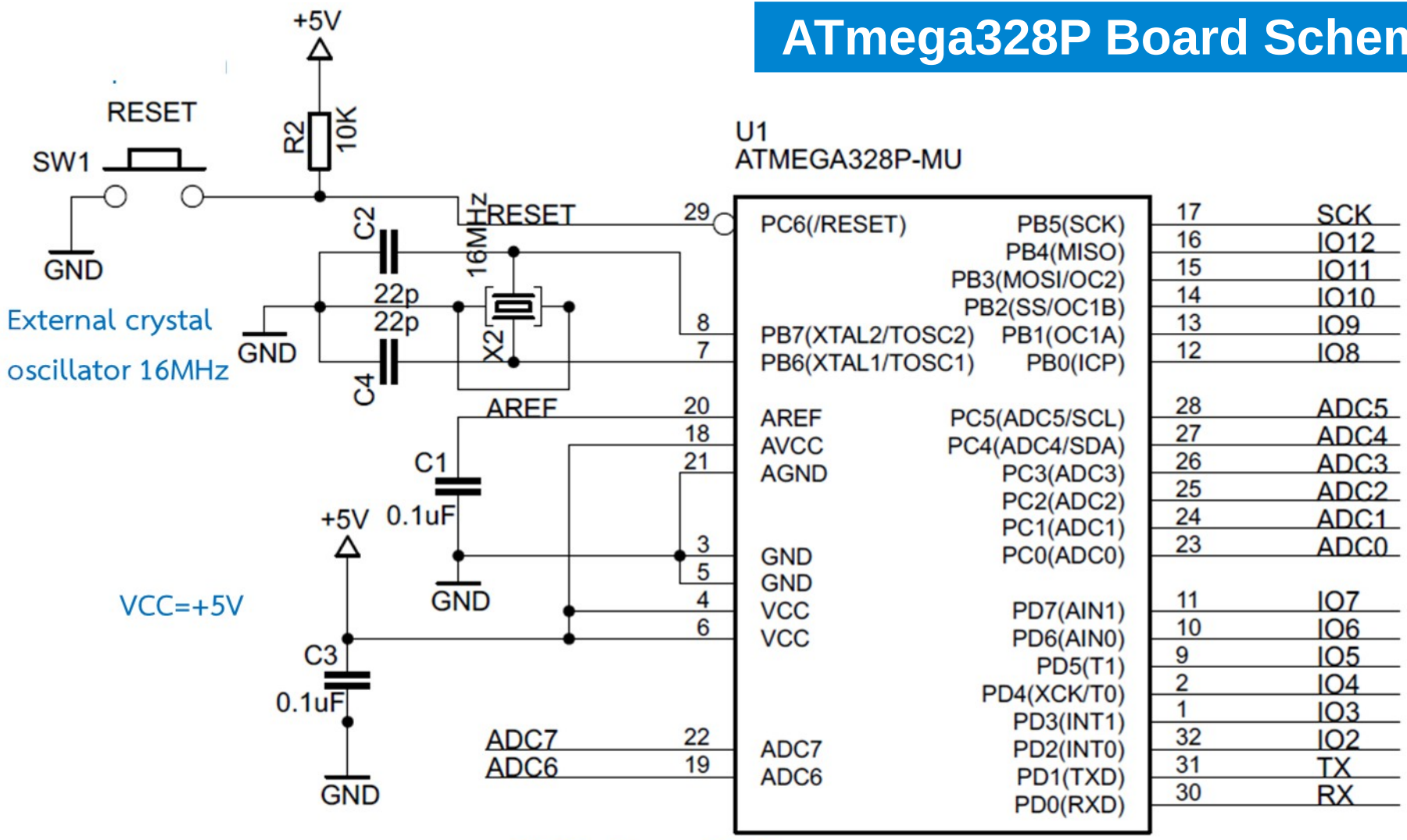
Source: http://www.datasheetcafe.com/usbasp-pinout-avr-programmer/

Arduino Breadboard Prototyping



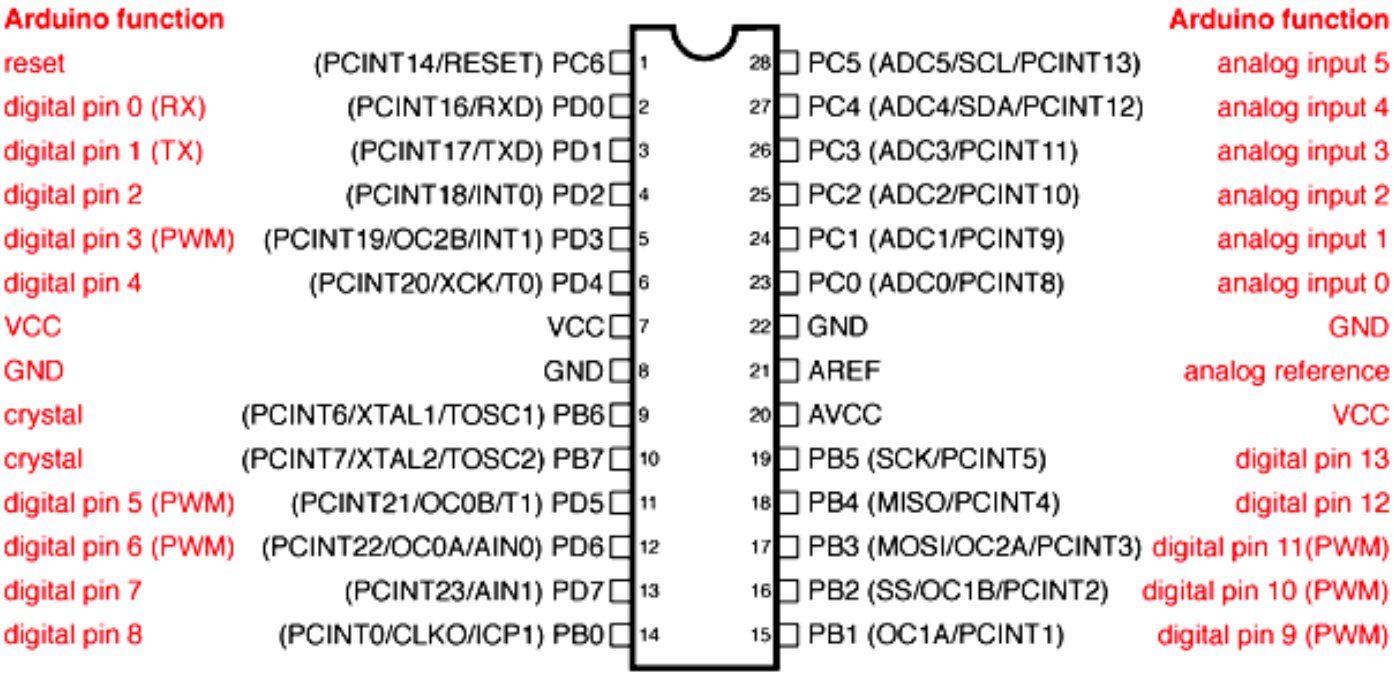
Source: http://arduino.cc/en/Main/Standalone

ATmega328P Board Schematic



MCU: ATmega328P

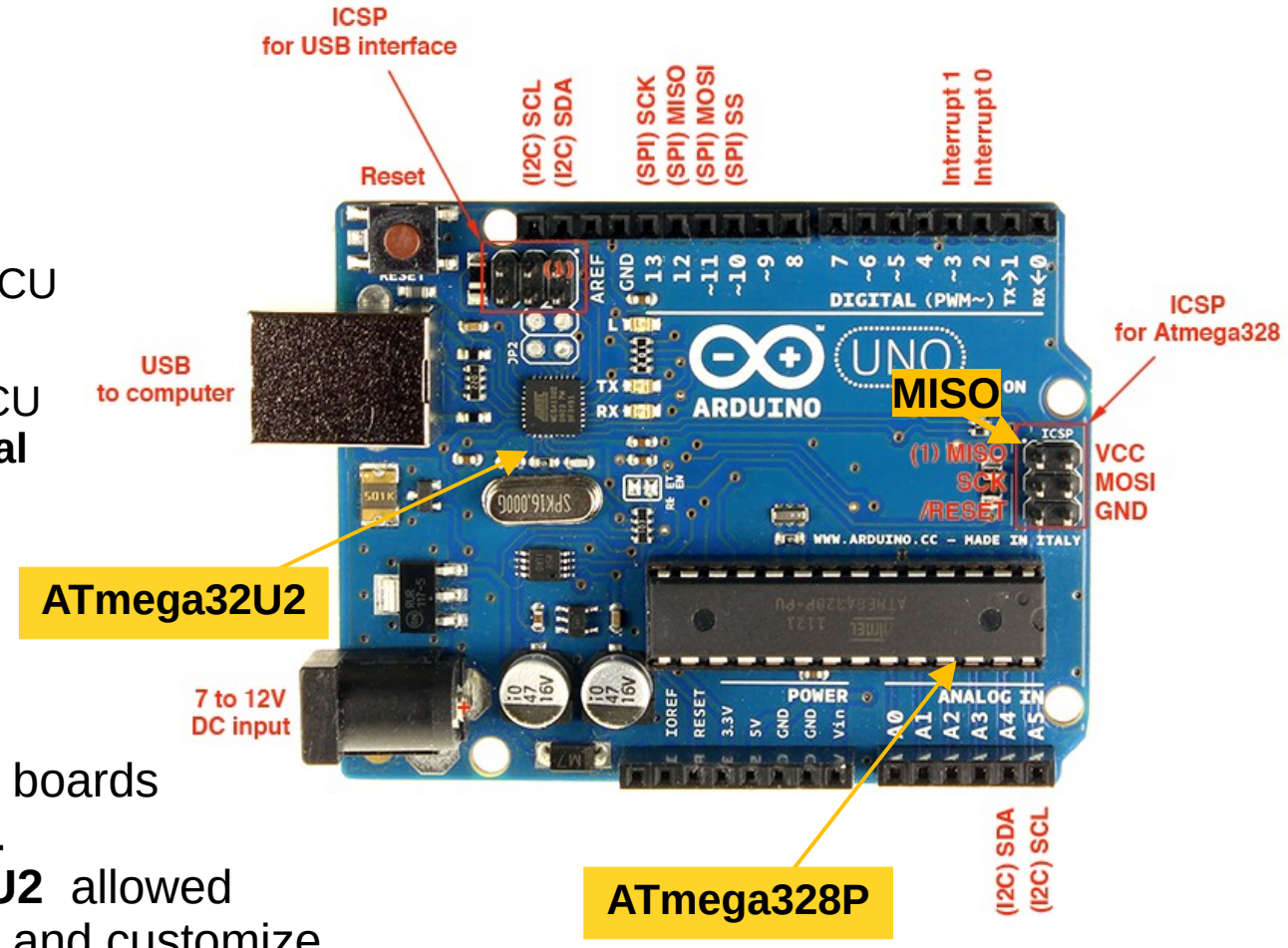
ATMega328P and Arduino Uno Pin Mapping



Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Arduino Uno R3

- **Arduino Uno Rev.3** has <u>two</u> 8-bit AVR-based MCU chips.
  - **ATmega328P** is the main MCU (no native-USB capability).
  - **ATmega32U2** is another MCU that provides a **USB-to-serial bridge** (USB CDC device) for the ATmega328P.



- Previous versions of Arduino boards used **FTDI's FT232RL** chips.
- Switching to the **ATmega16U2** allowed the Arduino team to program and customize the USB interface firmware.

AVR Programming Approaches

1) Using the Arduino API

- The Arduino environment provides the highest level of abstraction, making it the easiest entry point for beginners.
- **OOP** (Object-oriented programming) is supported by C++.
- The Arduino "language" is essentially C/C++ with a pre-setup environment (the IDE) and a specific library layer (**Arduino API**) that simplifies interaction with the AVR hardware (like the ATmega328P chip used in an Arduino Uno).
- This method hides low-level register access, making it ideal for beginners but less efficient and less flexible compared with bare-metal C or assembly
- **Toolchain**: The underlying compiler is the **AVR-GCC toolchain** when targeting AVR boards such as the ATmega328P (Arduino Uno).
- **IDEs**: Arduino IDE (offline / online), VS Code IDE + Platform IO

AVR Programming Approaches

2) Using Bare-Metal C

- This approach is about writing C code that interacts directly with the **AVR's hardware register**s, completely bypassing the Arduino API.
- **I/O Register Access**: This means using macros or variable names defined in header files (like <avr/iom328p.h>) to set or read specific memory locations (registers) that control peripherals.
- **Toolchain**:
  - **AVR-GCC toolchai**n is used (such as **avr-gcc** / **avr-g++**).
  - **AVR-Libc** is a **standard C library specific to AVR MCUs** that provides basic necessities like I/O functions and utility routines.
- **IDEs**:
  - Arduino IDE, VS Code IDE + Platform IO
  - Microchip Studio for AVR (based on Atmel Studio 7) or MPLAB-X + XC8 compiler

AVR Programming Approaches

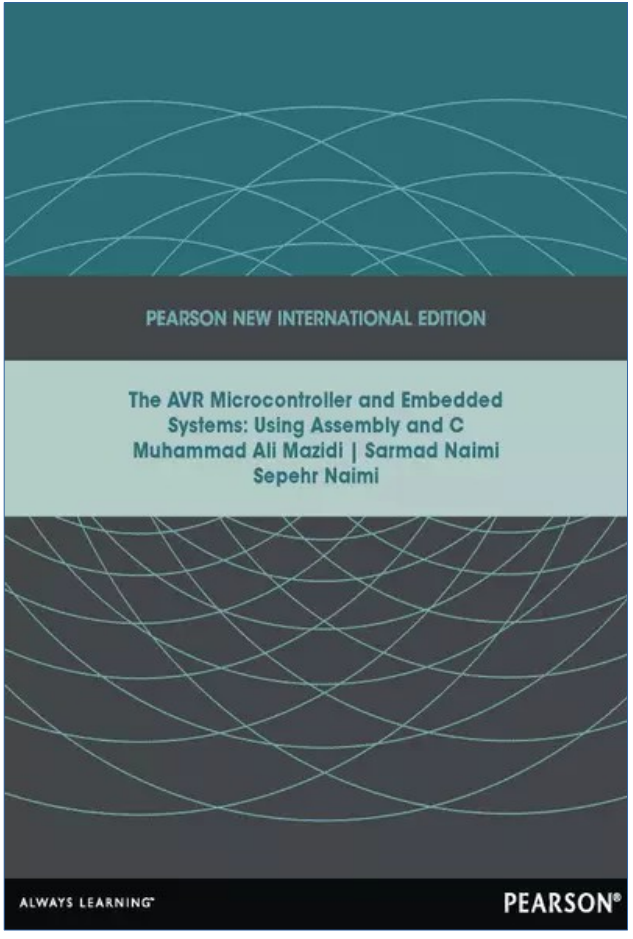
3) AVR Assembly

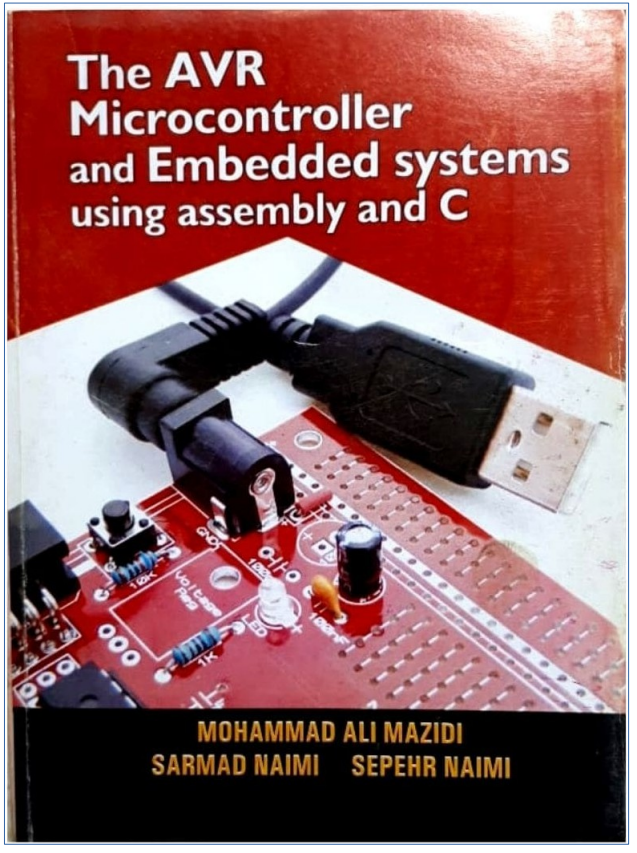
- This is the lowest level of programming.
- Assembly gives full control over registers, stack operations, timing, and instruction choices.
- Writing in Assembly requires the programmer to manage all aspects of the program, which is why it is considered tedious.
- **Assemblers** (CLI tools) for AVR Assembly code
  - **Atmel AVR Assembler 2** (**avrasm2**), included with the **Microchip Studio**.
  - Open-source tools: **AVR-GCC (GNU Assembler: avr-as)** and **AVRA**
  - **AVRA** is a popular, standalone, open-source assembler. It aims to be highly compatible with the syntax of the official Atmel assembler (**avrasm2**).

    ⚠ GNU Assembler (avr-as) uses syntax that differs from **AVRA** and **avrasm2** in some directives.

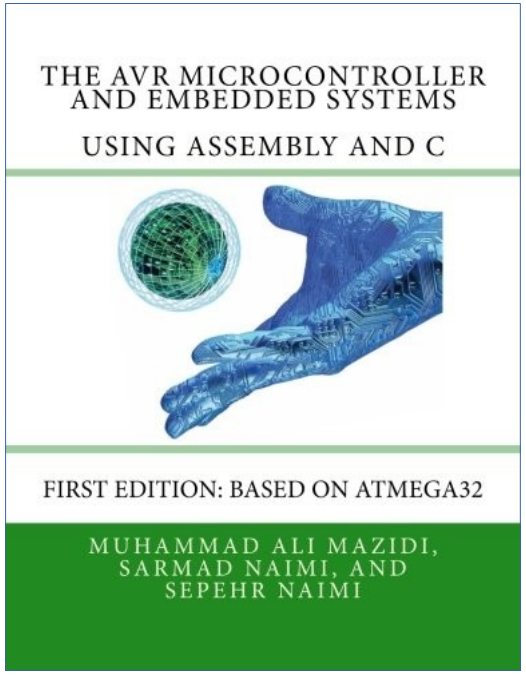
https://github.com/Ro5bert/avra

Textbooks for AVR Programming



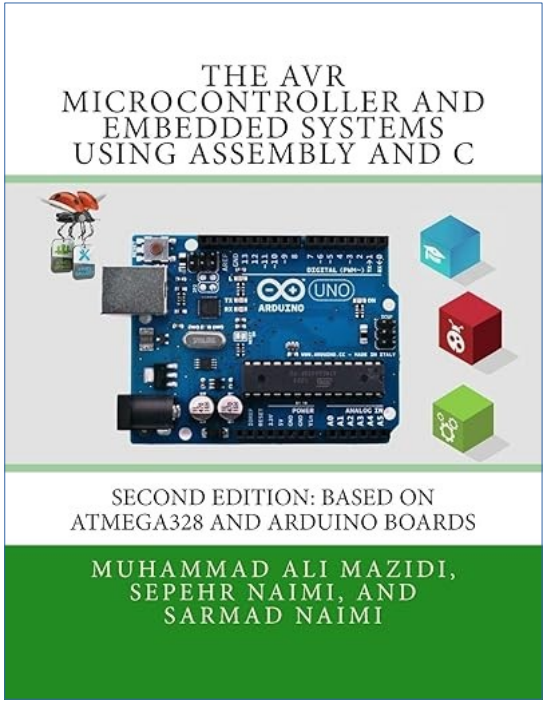


Textbooks for AVR Programming



ATmega32 Edition
Pub. Date: January 2011

https://nicerland.com/atmega32/



ATmega328 Edition
Pub. Date: August 2017

https://nicerland.com/avr/

CHAPTERS

APPENDICES

ATmega32A vs. ATmega328P

1) "*AVR Microcontroller and Embedded Systems: Using Assembly and C*"

2) "*The AVR Microcontroller and Embedded Systems Using Assembly and C: Using Arduino Uno and Atmel Studio*"

Based on these two books, it should be noted that each edition focuses on a **different AVR microcontroller**: the first book primarily uses the **ATmega32**, while the second focuses on the **ATmega328P** (used on Arduino Uno / Nano boards).

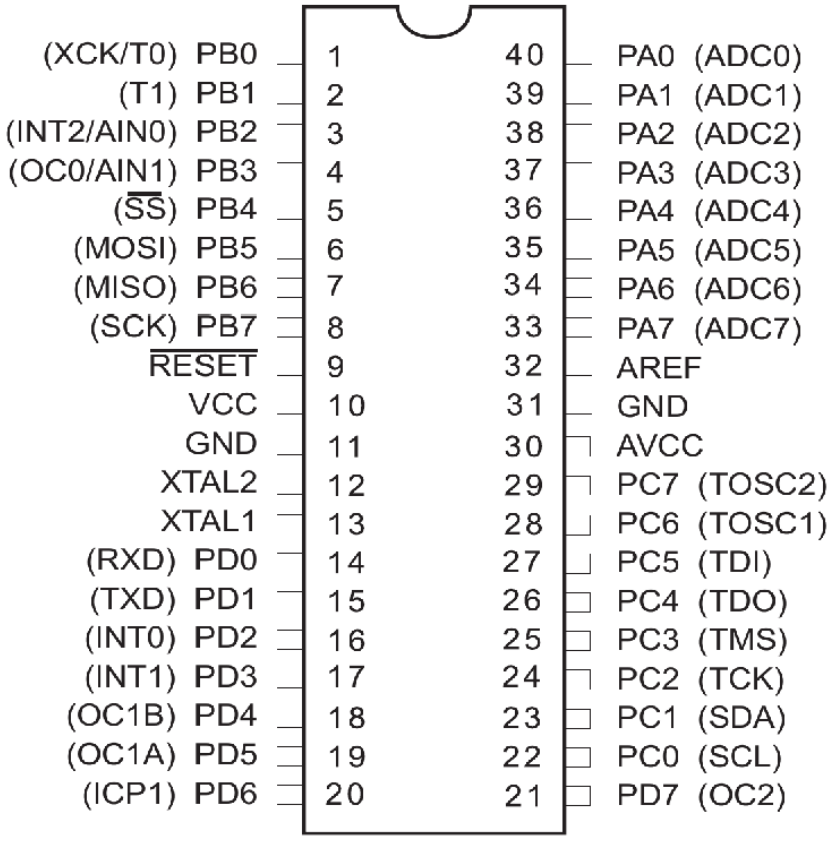
The **ATmega32** and **ATmega328P** share many architectural features, but they also differ in several important aspects.

ATmega32A vs. ATmega328P

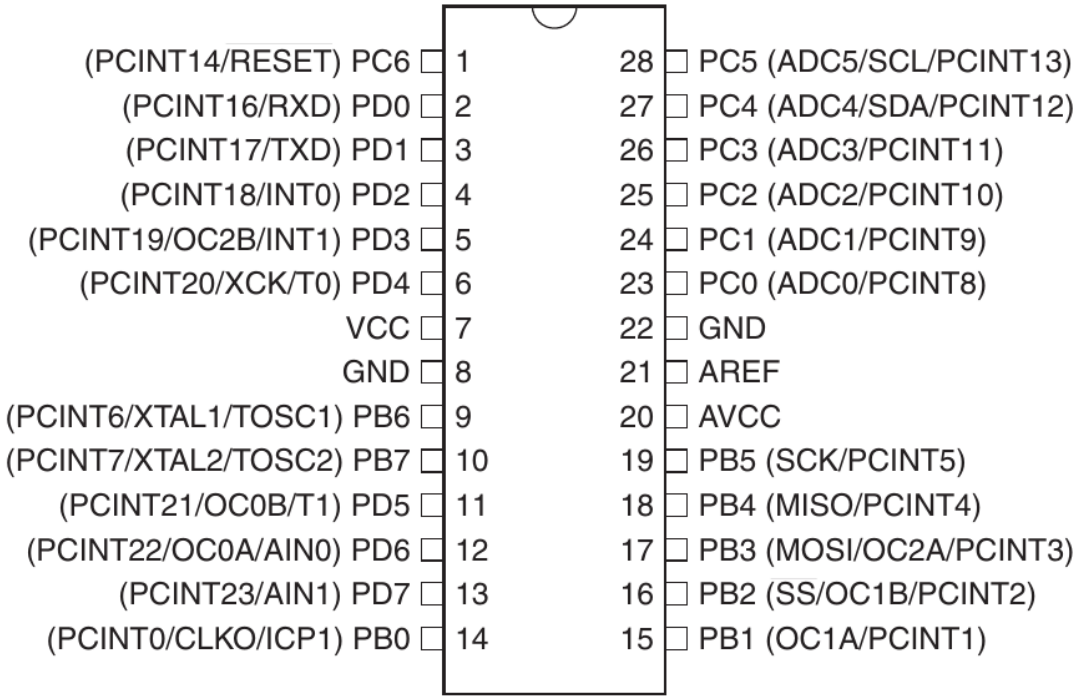
Common Features

- 8-bit AVR RISC architecture, same AVR instruction set / assembly model
- Harvard architecture with separate program and data memories
- On-chip memories:
  Flash (32,768 bytes), SRAM (2048 bytes), and EEPROM (1024 bytes)
- CPU frequency: 16MHz (max.) @VCC >= 4.5V
- On-chip 2-cycle Multiplier
- Six low power modes:
  Idle, ADC noise reduction, power-save, power-down, standby, extended standby
- ISP (In-System Programming)
- 1x Serial USART, 1x Master/Slave SPI, 1x TWI (I2C)
- 1x Programmable Watchdog Timer with Separate On-chip Oscillator
- 1x On-chip Analog Comparator
- 1x Internal Calibrated RC Oscillator
- PortA, PortB, PortC, and PortD

ATmega32A vs. ATmega328P



ATmega32A
40-pin PDIP



ATmega328P
28-pin PDIP

ATmega32A vs. ATmega328P

ATmega32 / 32A

- Older "classic megaAVR" family
- ATmega32A is a drop-in replacement for ATmega32, with lower power consumption
- Packages: 40-pin PDIP, 44-pin TQFP, 44-pad QFN/MLF
- Voltage range: 2.7V - 5.5V
- Four PWM Channels (from Timer0 / Timer1 / Timer2): OC0, OC1A/B, OC2 pins
- JTAG debug interface
- External interrupts: 3 (INT0, INT1, INT2)
- No Pin Change Interrupts (PCINTs)
- Internal reference voltage: 2.56V
- 8-channel (single-ended) 10-bit ADC, with differential input pairs and optional gain stages
- Programmable Internal RC: 1MHz / 2MHz / 4MHz / 8MHz
- External RC Oscillator support

ATmega32A vs. ATmega328P

ATmega328 / 328P

- Newer generation megaAVR
- P = picoPower (designed for lower power; lower power version of ATmega328)
- Voltage range: 1.8V – 5.5V
- Packages: 28-pin PDIP or 32-pin QFN/MILF
- Widely used on Arduino Uno and Nano boards
- DebugWIRE (for on-chip debugging via RESET pin), but no JTAG
- Six PWM Channels (from Timer0 / Timer1 / Timer2): OC0A/B, OC1A/B, OC2A/B pins
- Pin-Change Interrupt or PCINT (24 pins / 3 groups)
- 8-ch. Single-ended, 10-bit ADC, no programmable gain, no differential input pairs
- External interrupts: 2 (INT0, INT1)
- Internal reference voltage: 1.1V
- Internal RC: 8 MHz or 1MHz (DIV8)

ATmega32A vs. ATmega328P

Interrupt Vector Table Comparison

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

ATmega32A

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

ATmega328P