Ch. 3 IoT Programming

Section 1: Intro, pins and A/D signals

COMPSCI 147

Internet-of-Things; Software and Systems

Birth of ESP

- One of the best options for “IoT Projects”
- Integrated Wi-Fi connectivity
- Different options

Birth of ESP (By Espressif Systems)

- One of the best options for “IoT Projects”
- Popular because of its Wi-Fi capability
- Different options available

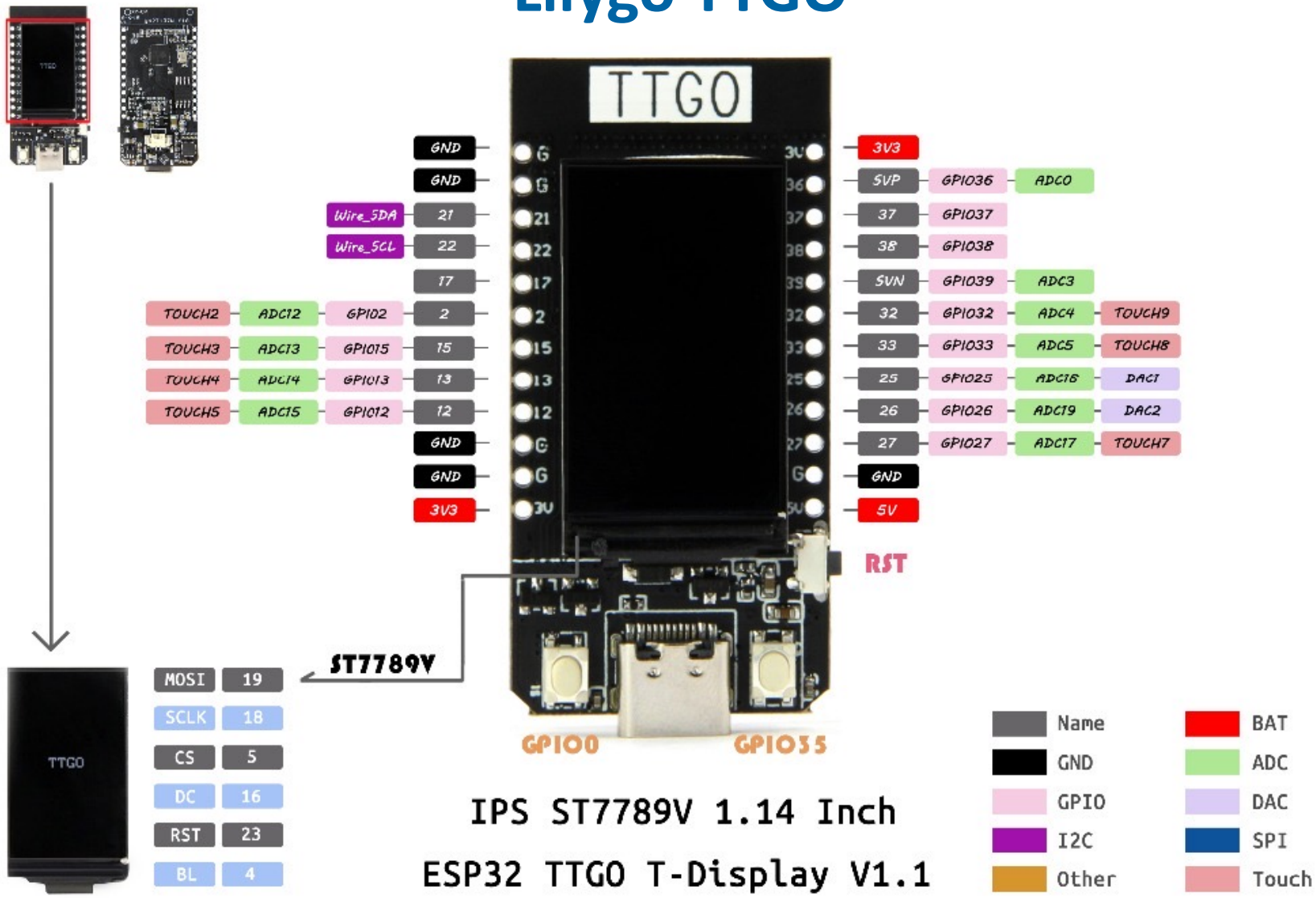
ESP-01	WEMOS D1 MINI	ESP-12E (NodeMCU)	ESP-32
Single-Core	Single-Core	Single-Core	Dual-Core + BT
4 GPIOs	11 GPIOs	17 GPIOs	36 GPIO
1 MB FLash	4 MB Flash	4 MB FLash	16MB Flash
$	$$	$$	$$$

Lots of ways to program!

- Arduino — A C++-based firmware. With this core, the ESP8266 CPU and its Wi-Fi components can be programmed like any other Arduino device. The ESP8266 Arduino Core is available through GitHub.
- ESP8266 BASIC — An open-source BASIC-like interpreter specifically tailored for the Internet of Things (IoT). Self-hosting browser-based development environment.
- ESP Easy — Developed by home automation enthusiasts.
- ESPHome — ESPHome is a system to control your ESP8266/ESP32 by simple yet powerful configuration files and control them remotely through home automation systems.
- Tasmota - open-source firmware, very popular with home automation enthusiasts.
- ESP-Open-RTOS — Open-source FreeRTOS-based ESP8266 software framework.
- ESP-Open-SDK — Free and open (as much as possible) integrated SDK for ESP8266/ESP8285 chips.
- Espruino — An actively maintained JavaScript SDK and firmware, closely emulating Node.js. Supports a few MCUs, including the ESP8266.
- ESPurna — Open-source ESP8285/ESP8266 firmware.
- Forthright — Port of Jones Forth to the ESP8266 microcontroller.
- MicroPython — A port of MicroPython (an implementation of Python for embedded devices) to the ESP8266 platform.
- Moddable SDK — includes JavaScript language and library support for the ESP8266
- Mongoose OS — An open-source operating system for connected products. Supports ESP8266 and ESP32. Develop in C or JavaScript.[14]
- NodeMCU — A Lua-based firmware.
- PlatformIO — A cross-platform IDE and unified debugger, which sits on top of Arduino code and libraries.
- Punyforth — Forth-inspired programming language for the ESP8266.
- Sming — An actively developed asynchronous C/C++ framework with superb performance and multiple network features.
- uLisp — A version of the Lisp programming language specifically designed to run on processors with a limited amount of RAM.
- ZBasic for ESP8266 — A subset of Microsoft's widely-used Visual Basic 6, which has been adapted as a control language for the ZX microcontroller family and the ESP8266.
- Zerynth — IoT framework for programming ESP8266[15] and other microcontrollers in Python.
- IOTBAH - is An operating system (OS) for Espressif ESP8266

https://en.wikipedia.org/wiki/ESP8266

Lilygo TTGO



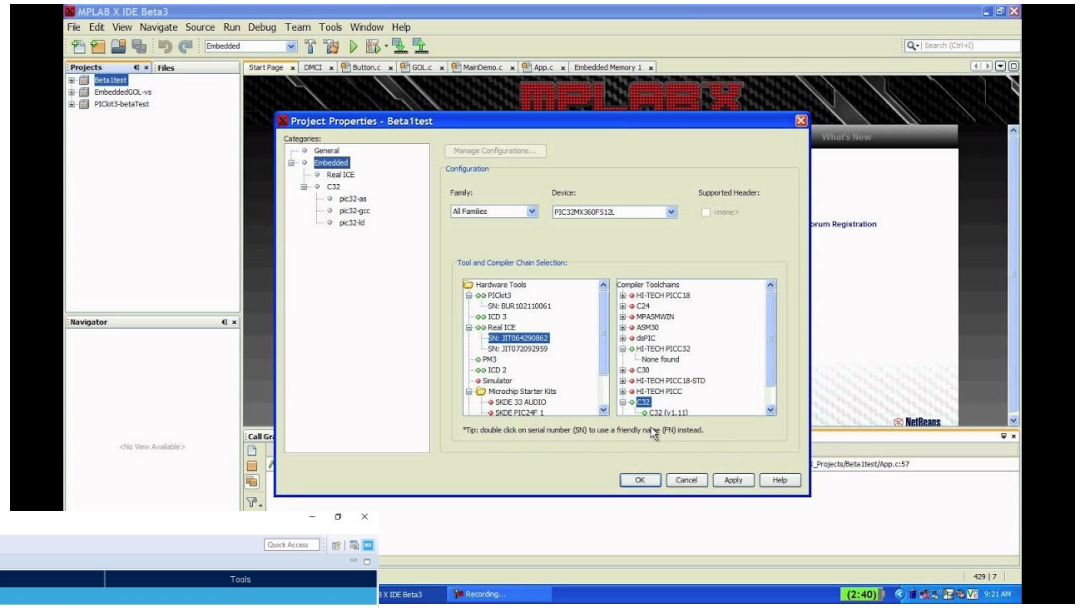
http://www.lilygo.cn/prod_view.aspx?Id=1126

Why TTGO?

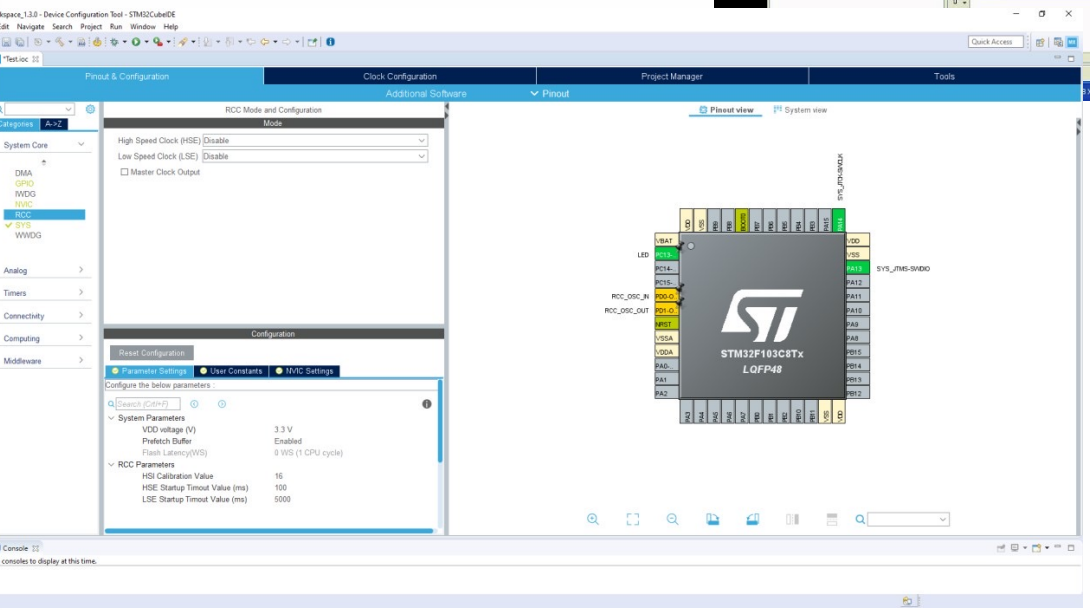
- ESP-32 based
- Built-in 1.14” RGB OLED !
- Built-in capacitive touch sensor !
- Battery support
- Everything is open-source
  https://github.com/Xinyuan-LilyGO/TTGO-T-Display

Before..

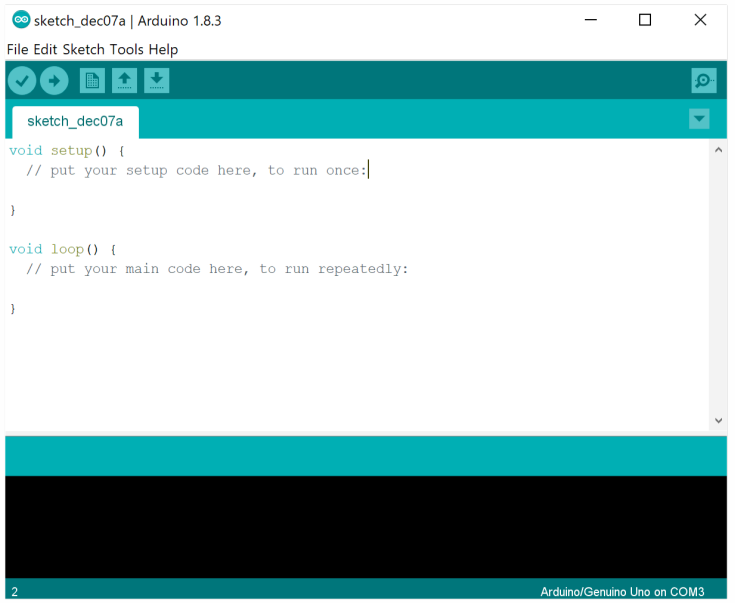
STM32 CubeIDE



ARDUINO IDE



MPLAB® X IDE



Before..


ARMmbed
ASR MICROELECTRONICS
Atmel
ESPRESSIF
freescale
GREENWAVES TECHNOLOGIES
Infineon
intel
LATTICE SEMICONDUCTOR
maxim integrated
MICROCHIP
NORDIC SEMICONDUCTOR
芯来科技 NUCLEI
NXP
PULP Parallel Ultra Low Power
RISC-V
SiFive
SAMSUNG ARTIK
SILICON LABS
life.augmented
Teensy
TEXAS INSTRUMENTS
WIZnet
ARDUINO
Energia
libOpenCM3

Now..

PlatformIO labs

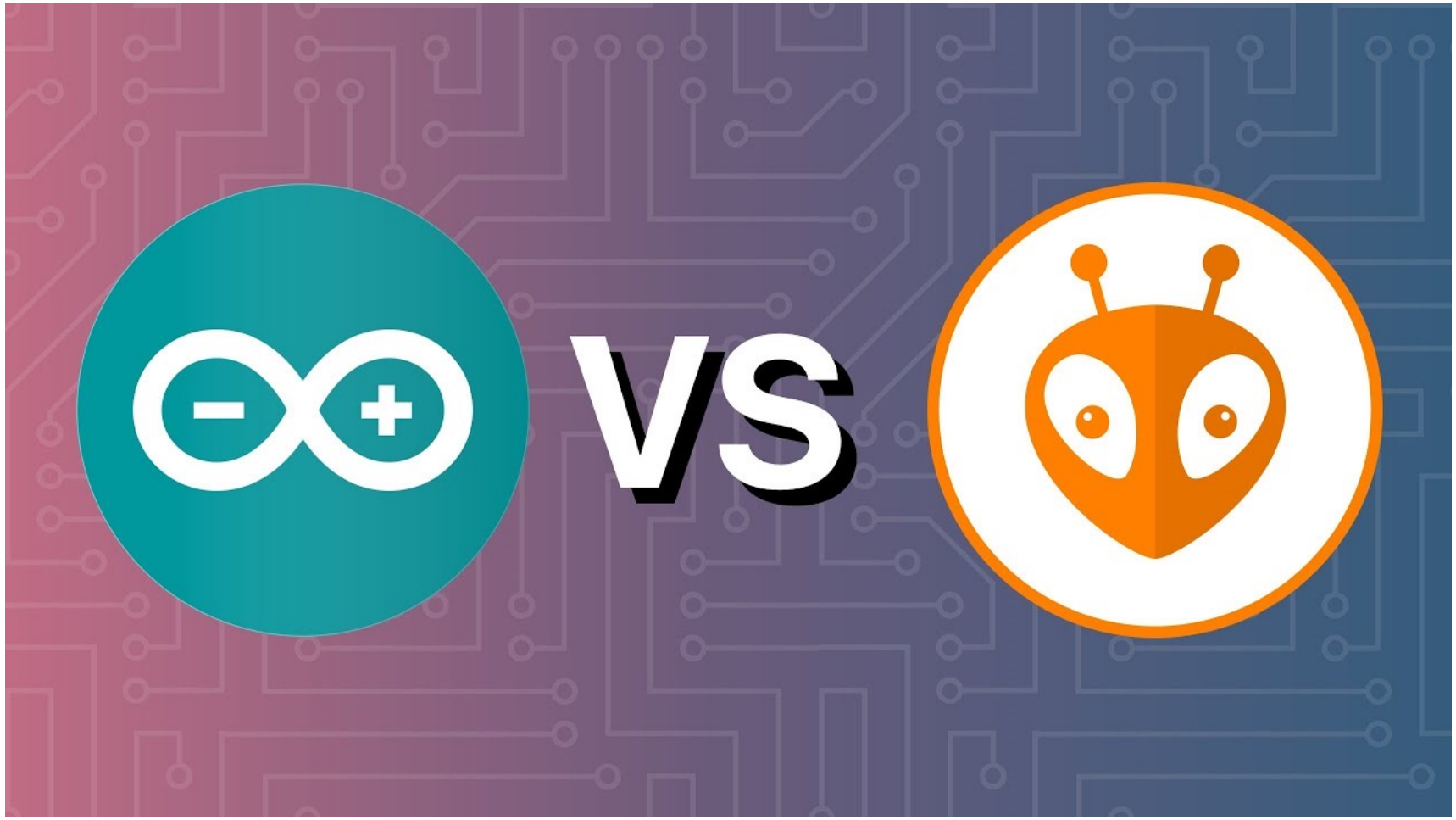
ARMmbed ASR MICROELECTRONICS Atmel ESPRESSIF freescale GREENWAVES TECHNOLOGIES

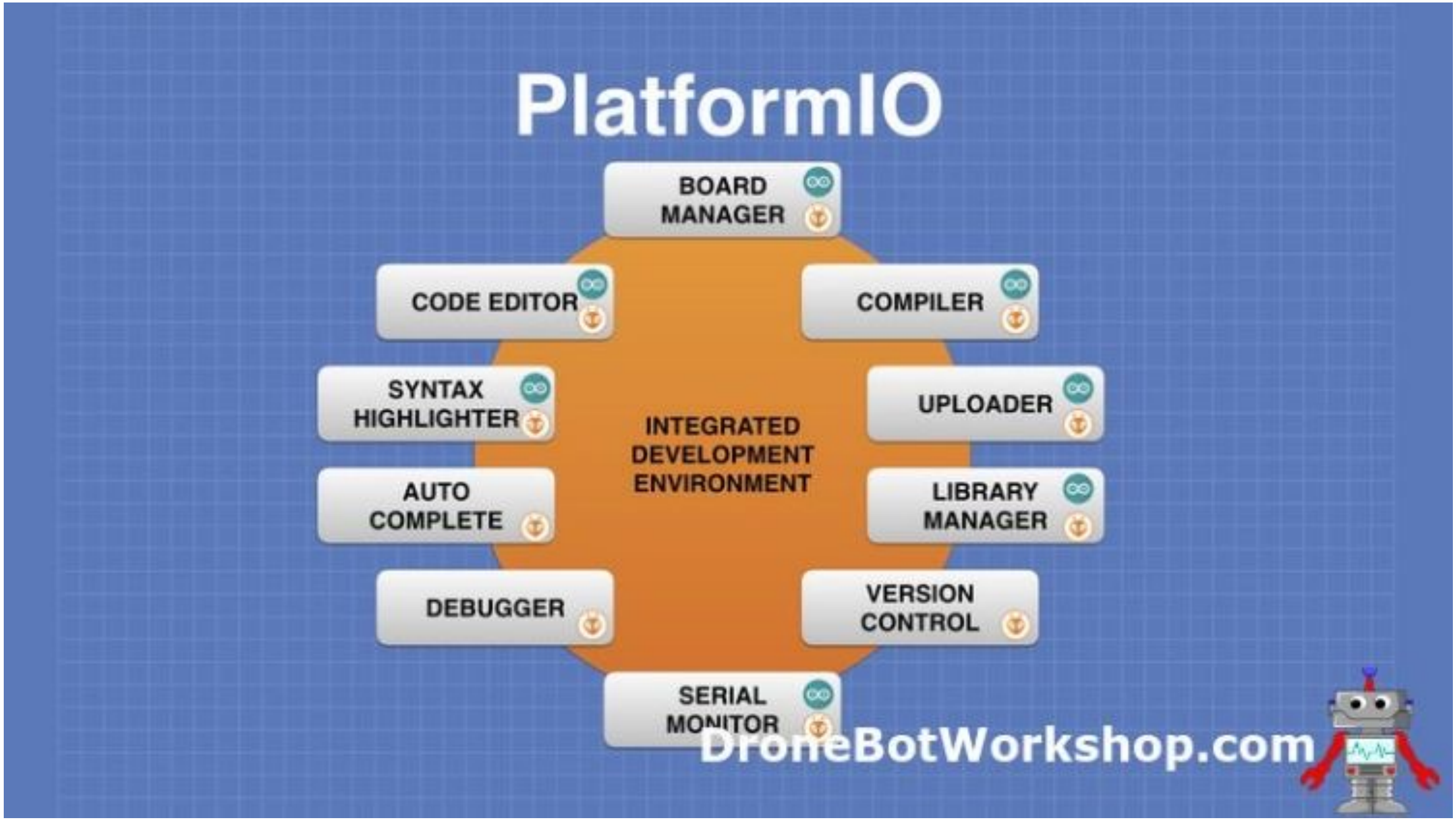
Infineon intel LATTICE SEMICONDUCTOR maxim integrated MICROCHIP NORDIC SEMICONDUCTOR 芯来科技 NUCLEI

NXP PULP Parallel Ultra Low Power RISC-V SiFive SAMSUNG ARTIK SILICON LABS

life.augmented Teensy Texas Instruments WIZnet ARDUINO Energia libOpenCM3

Two biggest embedded development environments




PlatformIO
BOARD MANAGER
CODE EDITOR
COMPILER
SYNTAX HIGHLIGHTER
INTEGRATED DEVELOPMENT ENVIRONMENT
UPLOADER
AUTO COMPLETE
LIBRARY MANAGER
DEBUGGER
VERSION CONTROL
SERIAL MONITOR
DroneBotWorkshop.com

Platform IO

- Different frameworks
  e.g., **Arduino**, ESP-IDF even for same microcontroller (ESP32)!

- Extensive library support

- Dependency manager

- Debugging support (May need external tools)

ARDUINO TERMINOLOGY

"*sketch*" – a program you write to run on an Arduino board

"*pin*" – an input or output connected to something.

e.g. output to an LED, input from a knob.

"*digital*" – value is either HIGH or LOW.

(aka on/off, one/zero) e.g. switch state

"*analog*" – value ranges, usually from 0-255.

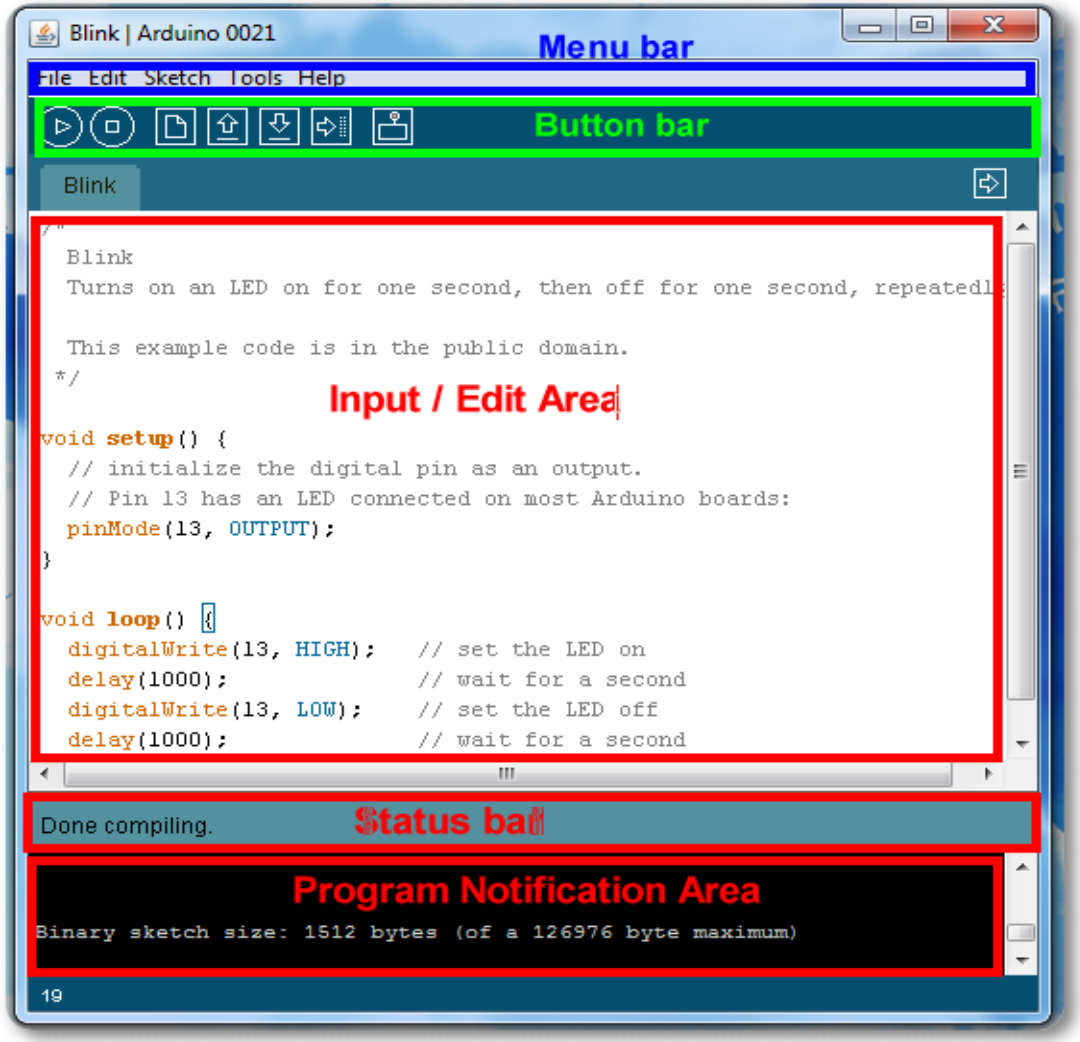
e.g. LED brightness, motor speed, etc.

Framework

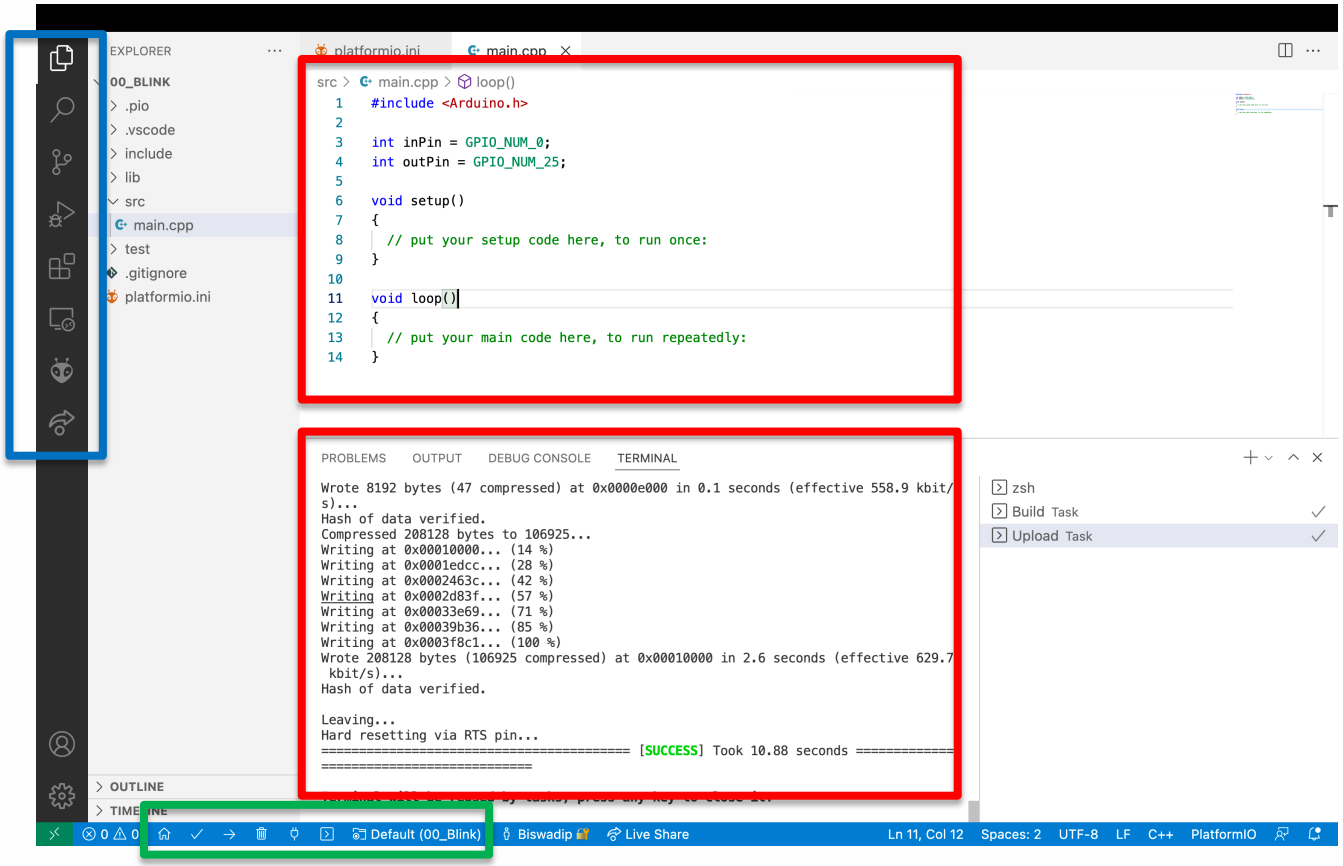
- **Arduino** programs are written in **C or C++**.
- Arduino
  - Written in C++, students normally use only small subset (looks a lot like C).
  - **Includes the Arduino IDE** and the **C/C++ software library**
  - It makes many common **input/output** operations much easier.
- Users only need to define **two functions** to make a runnable **cyclic executive program**:
  - **setup():** a function run **once** at the start of a program that can initialize settings.
  - **loop():** a function called **repeatedly** until the **board powers off.**

ENVIRONMENT

Arduino based local IDE (Guide)



PlatformIO+Arduino (Guide)



Program - Parts

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */
```

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
```

```
void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Comments / Explaining the game

Setup / Stretching or tying shoes

Loop / Playing the game

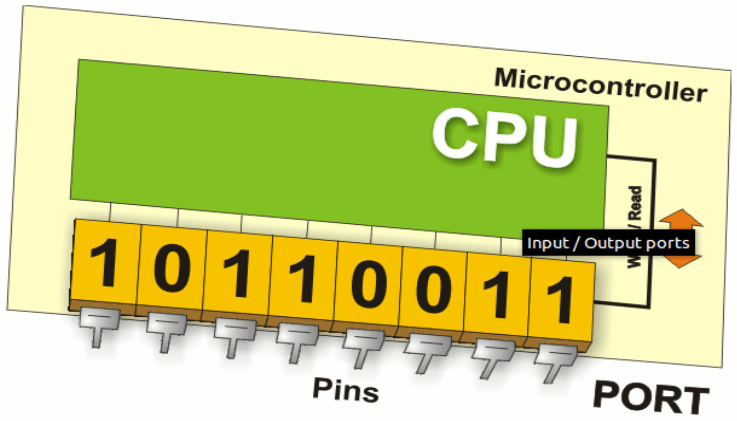
Comments can be anywhere (created with // or /* and */)

The setup function required for all Arduino sketches

The actual program for your board.

Pins - Digital I/0

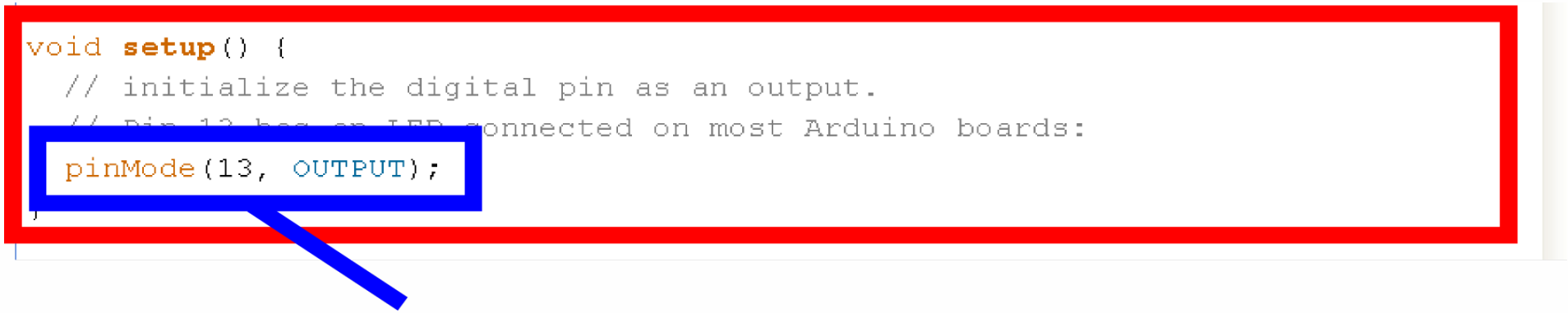
- `pinMode(pin, mode)`
  - Sets pin to either INPUT or OUTPUT
- `digitalRead(pin)`
  - Reads HIGH or LOW from a pin
- `digitalWrite(pin, value)`
  - Writes HIGH or LOW to a pin



- Electronic stuff
  - Output pins can provide 40 mA of current
  - Specials parameters INPUT_PULLUP / INPUT_PULLDOWN / ANALOG..

Program - Setup( )

- **Pin Outputs** are declared in setup
  - Done by using the `pinMode` function
  - This example declares digital pin # 13 as an output, remember to use CAPS

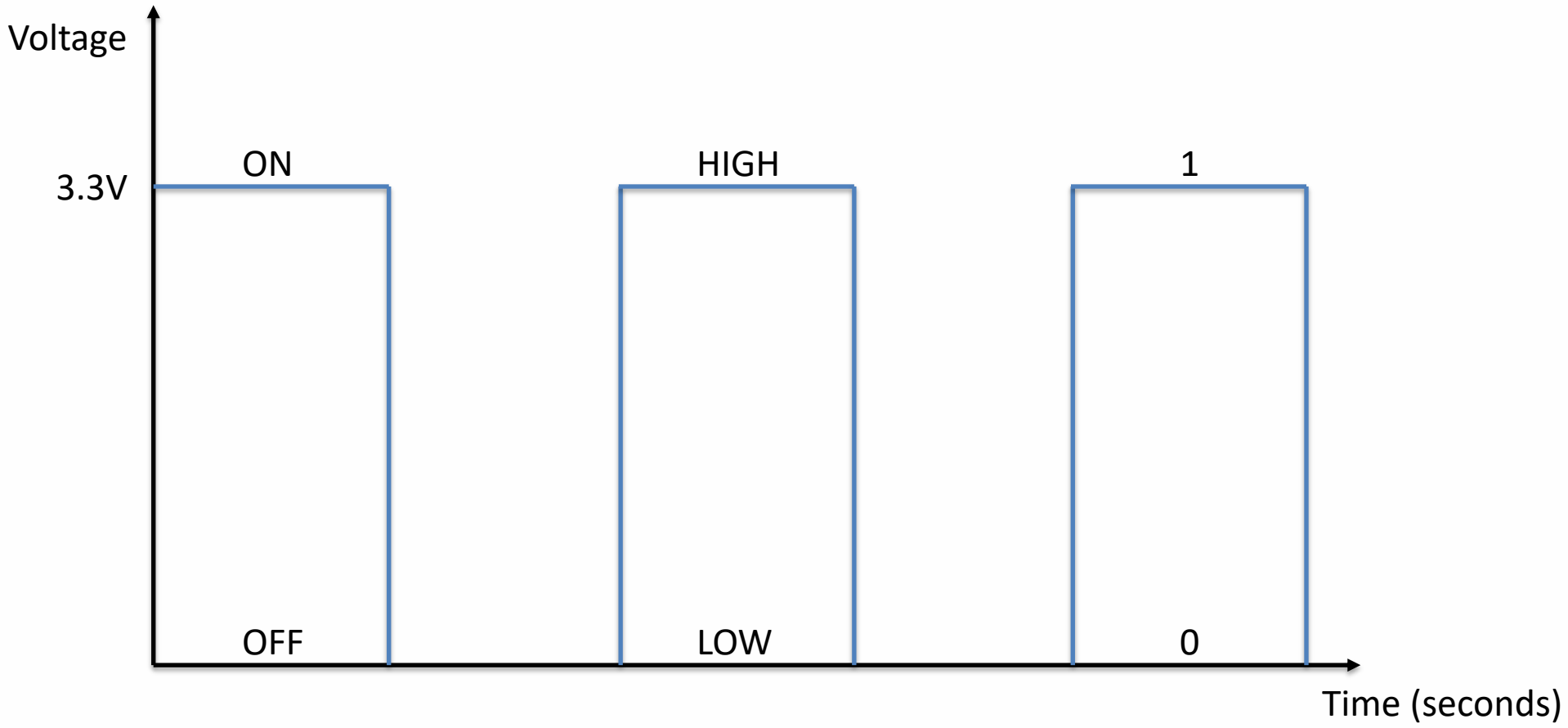


Example 1

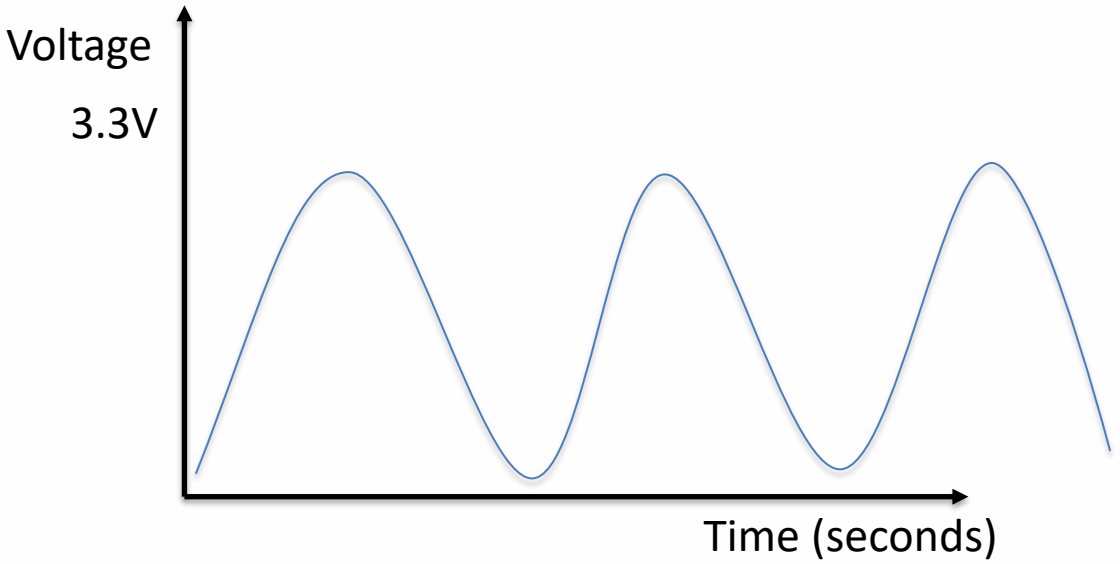
- HW: Schematic, SW: Platform I/O
- Use `digitalRead` to read a button state
- Use `digitalWrite` to turn on LED based on button

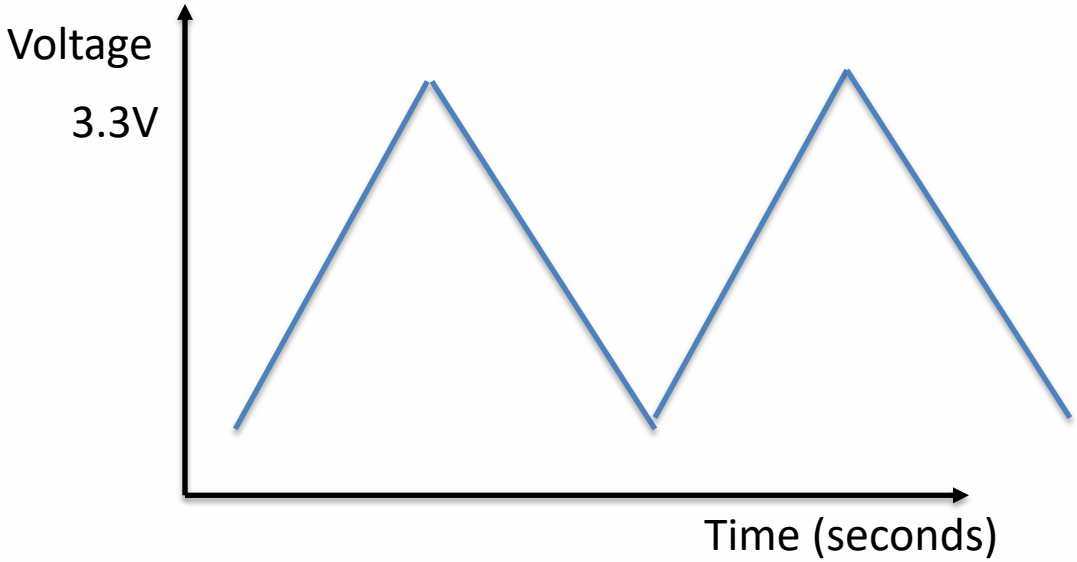
```
// define the pin numbers for the button and LED
const int buttonPin = 2;
const int ledPin = 3;

void setup() {
  // initialize the button pin as an input and the LED pin as an output
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the state of the button and store it in a variable
  int buttonState = digitalRead(buttonPin);

  // if the button is pressed, turn on the LED
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

Digital Values



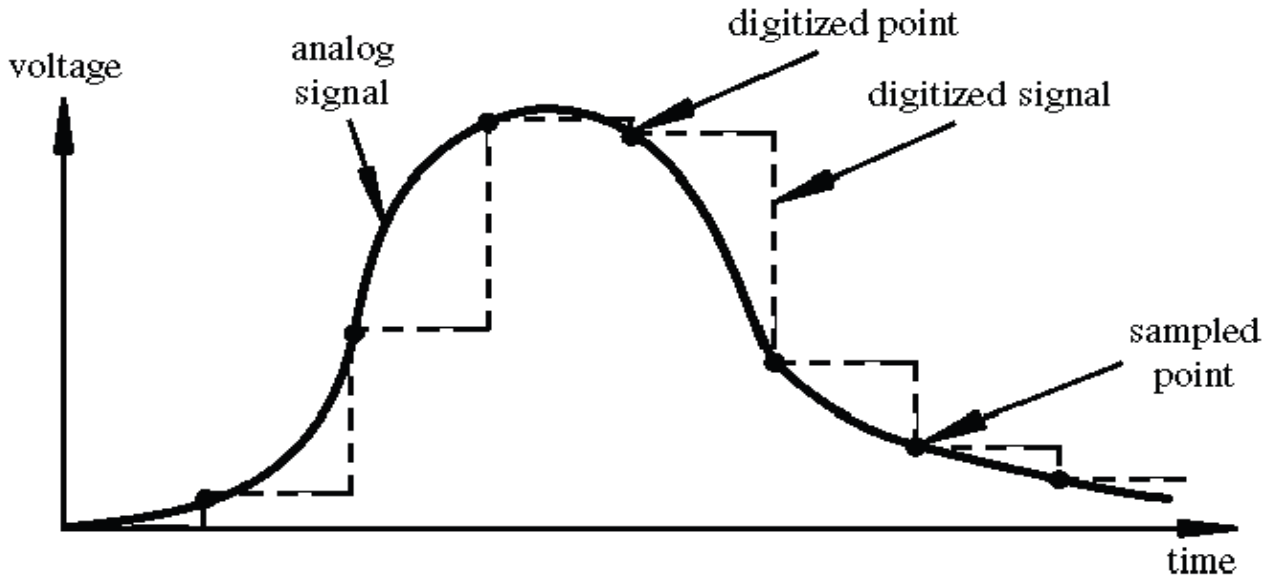
But what about..?



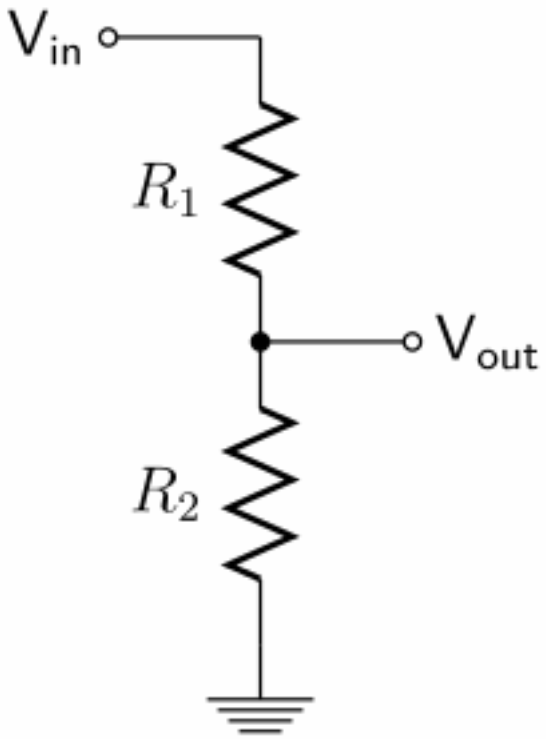


Analog vs Digital

- Digital has **two** values: **on** and **off**
- Analog has many (**infinite**) values
- Computers don't really do analog, they ***quantize***

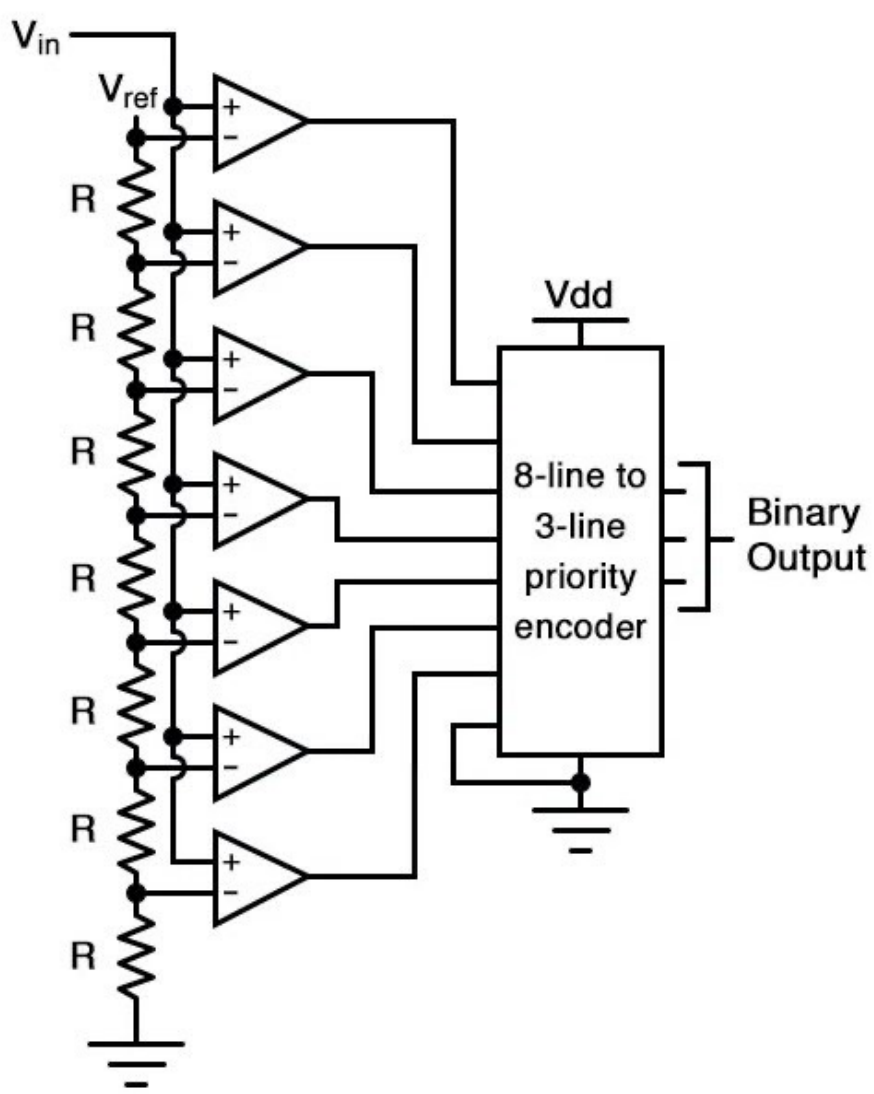


How to convert analog values to digital values

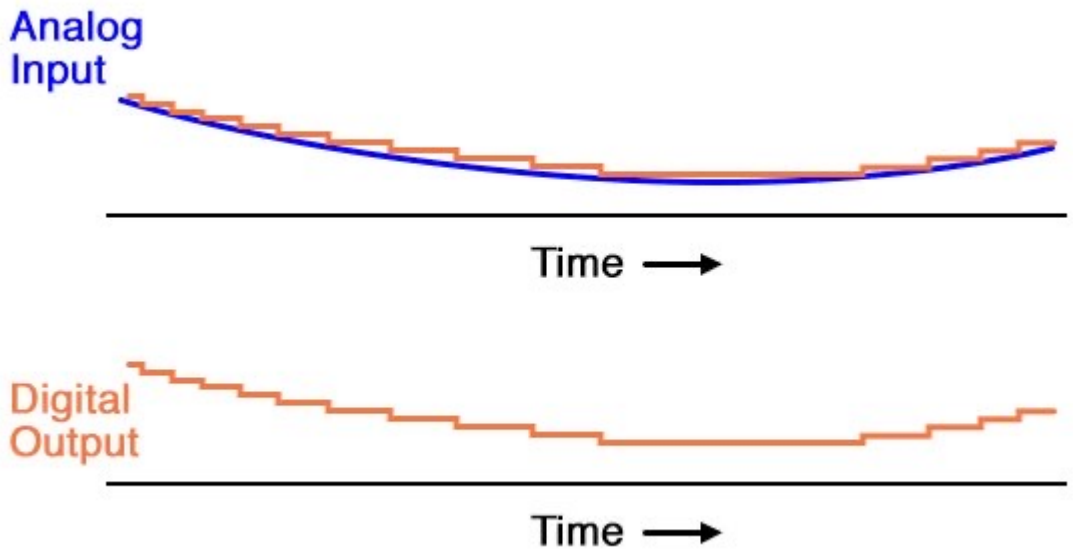


$$V_{\text{out}} = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}}$$

How to convert analog values to digital values

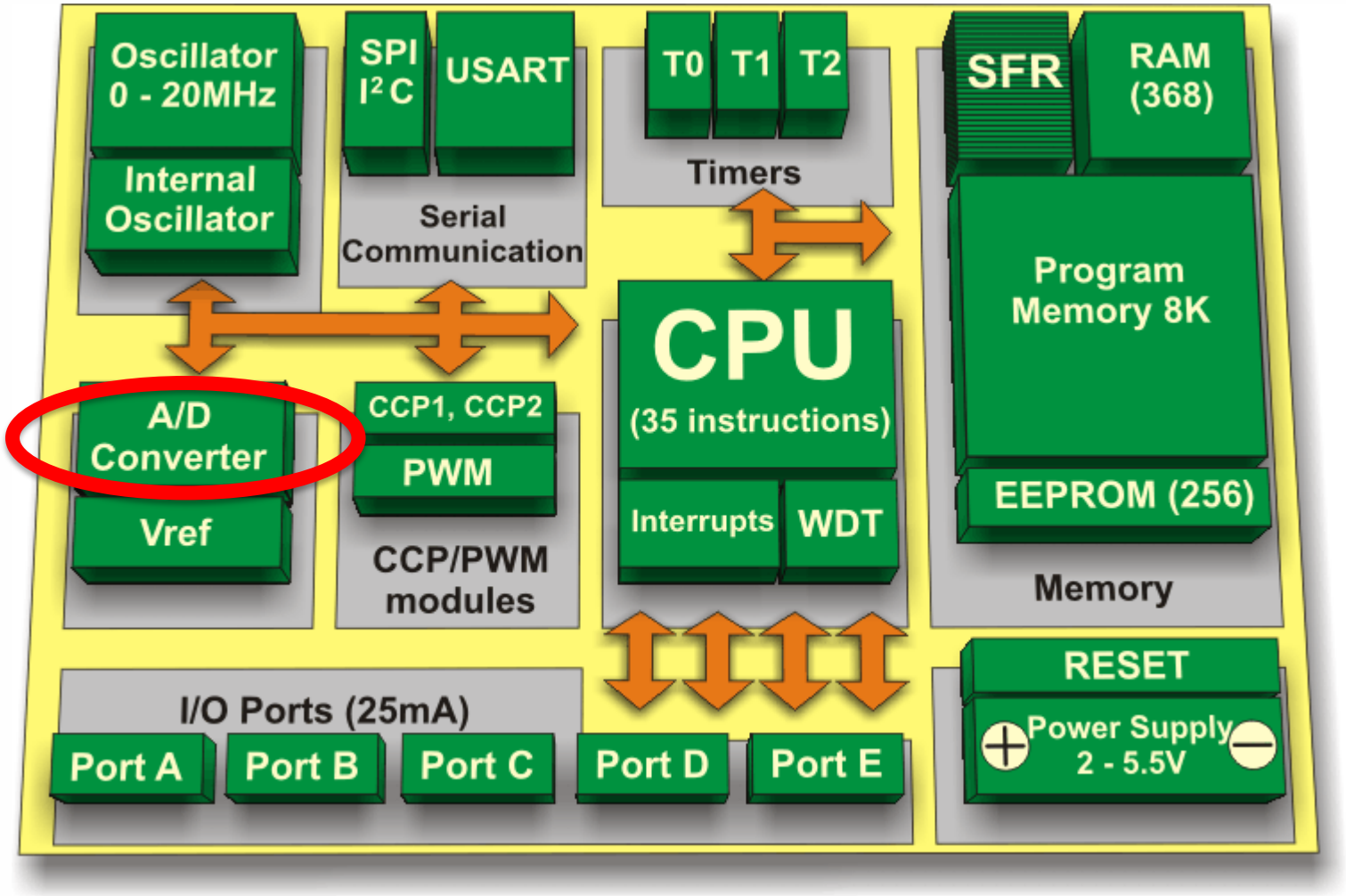


Flash ADC



ESP 32 uses
2 x SAR (Successive Approximation Register) ADCs

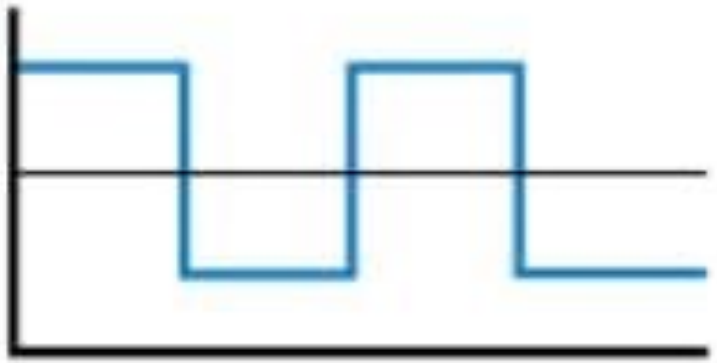
ESP32 already has similar circuit built-inside the chip



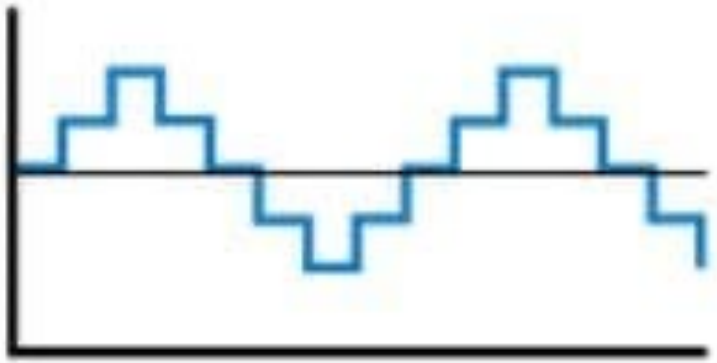
Example block diagram (NOT ESP32!)
https://www.lions-wing.net/lessons/microcontrollers/micros.html

- All we need to do is to configure the CPU and trigger the A/D convertor correctly.
- Arduino even abstracts it further by exposing very simple function. `analogRead()`

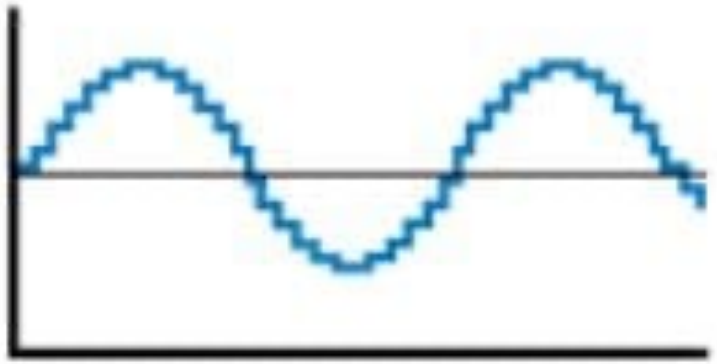
Effect of resolution



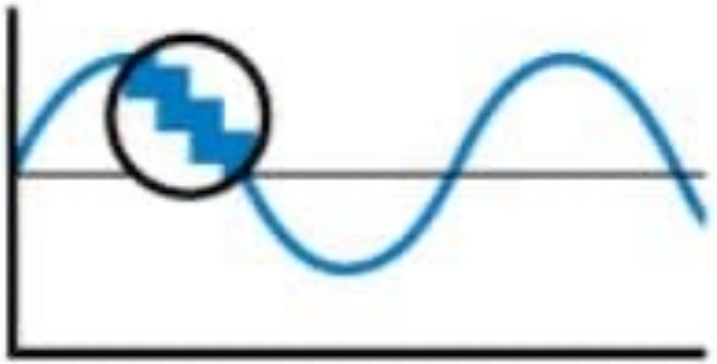
1-bit



2-bit



4-bit



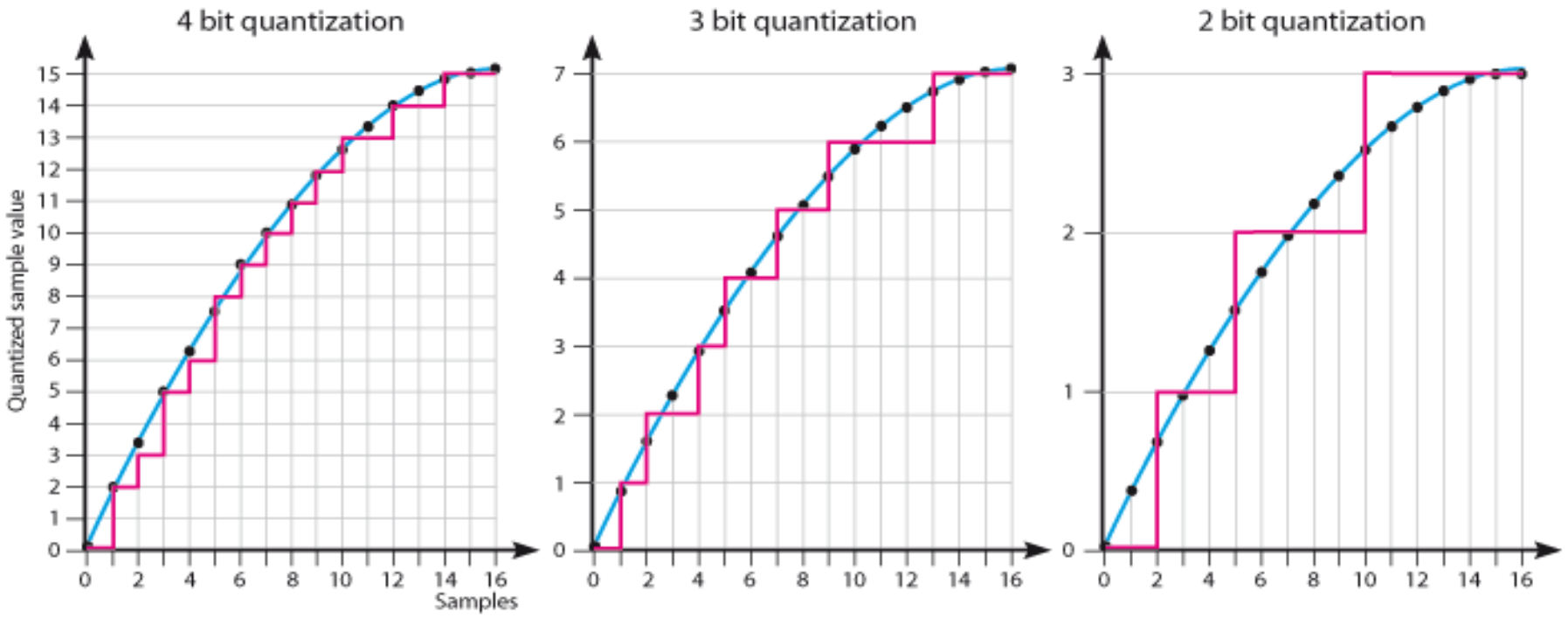
16-bit

Precision!

Pins - Analog I/0 (ADC)

`analogRead(pin)`

- Reads the voltage applied to an analog input pin and returns a number between **0 and 2^{adc_width}** (=resolution) that represents the voltages between 0 and 3.3 V.

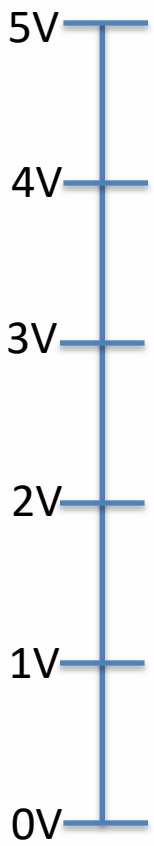


Pins - Analog I/0 (ADC)

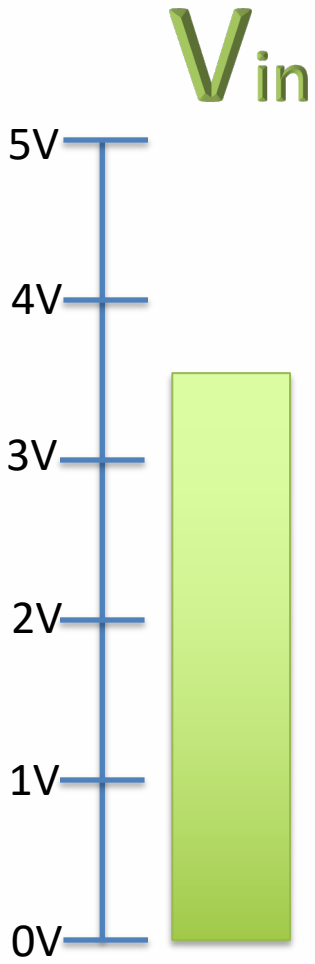
`analogRead(pin)`

- Resolution configurable to 9/10/11/12 bits
- Example: For 10bit resolution, reads the voltage applied to an analog input pin and returns a number between **0 and 1023** that represents the voltages between 0 and 3.3 V.
- ESP 32 supports a total of 18 measurement channels (analog enabled pins).
  - ADC1: 8 channels: GPIO32 - GPIO39
  - ADC2: 10 channels: GPIO0, GPIO2, GPIO4, GPIO12 - GPIO15, GOIO25 - GPIO27
- Not all pins may be exposed in TTGO (refer to pin diagram)

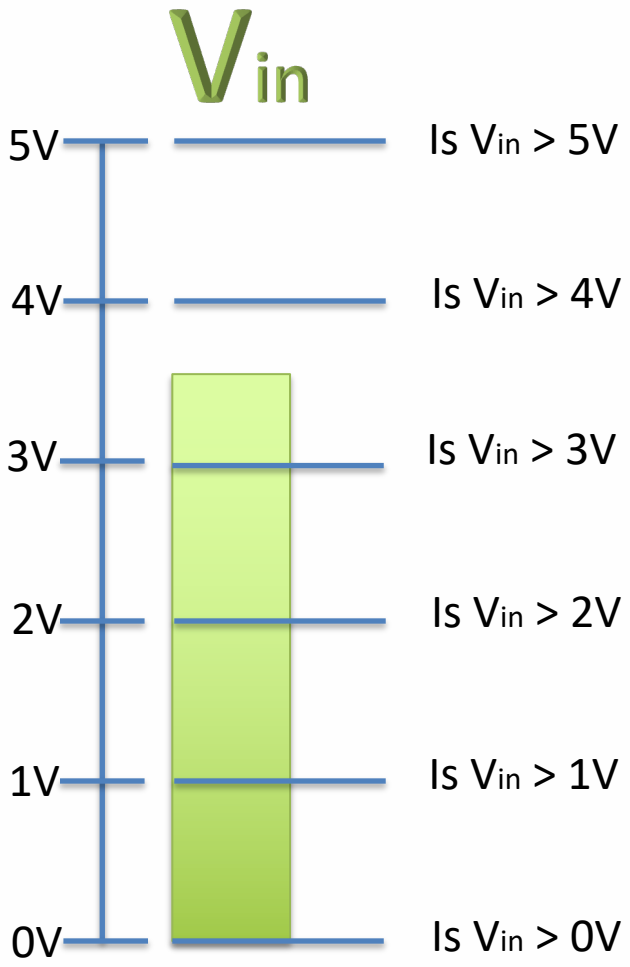
RECAP A2D



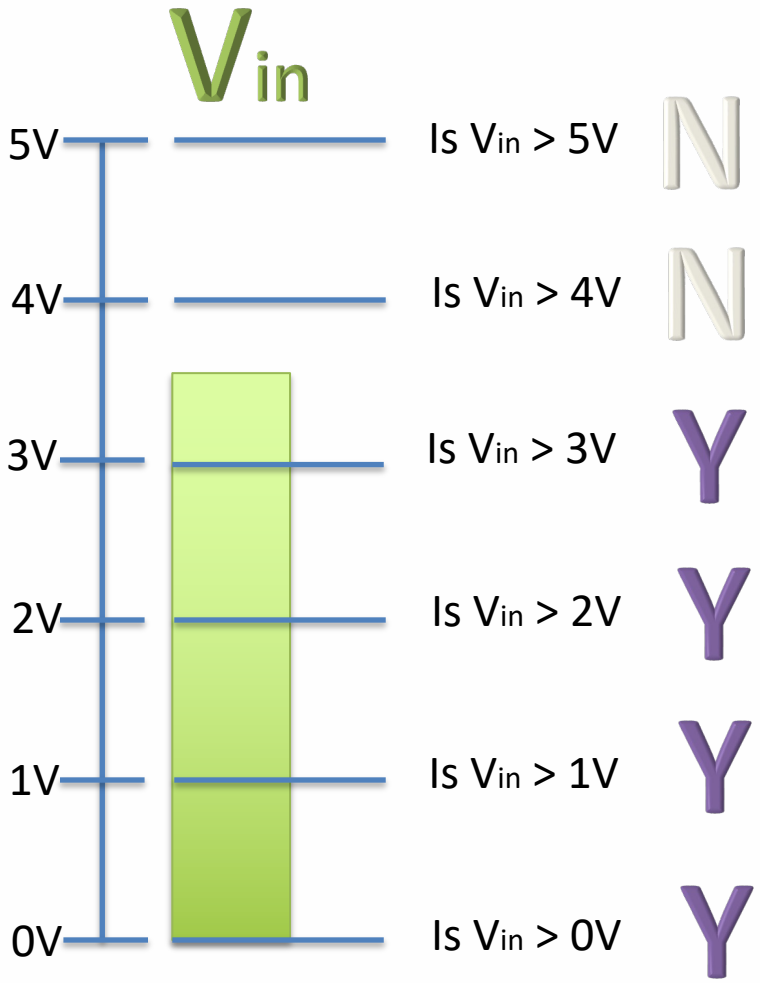
RECAP A2D



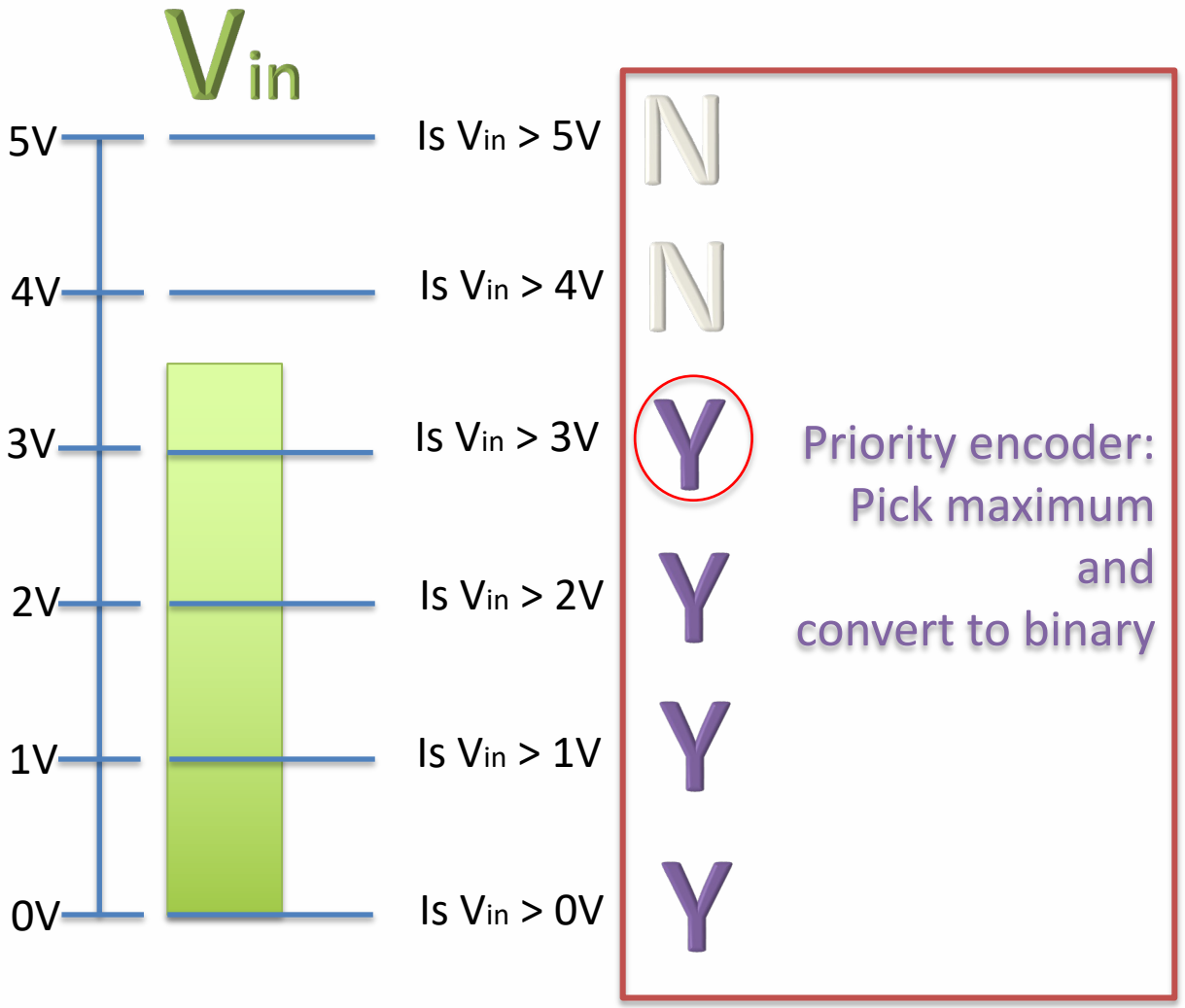
RECAP A2D


Vin
5V
4V
3V
2V
1V
0V
Is Vin > 5V
Is Vin > 4V
Is Vin > 3V
Is Vin > 2V
Is Vin > 1V
Is Vin > 0V

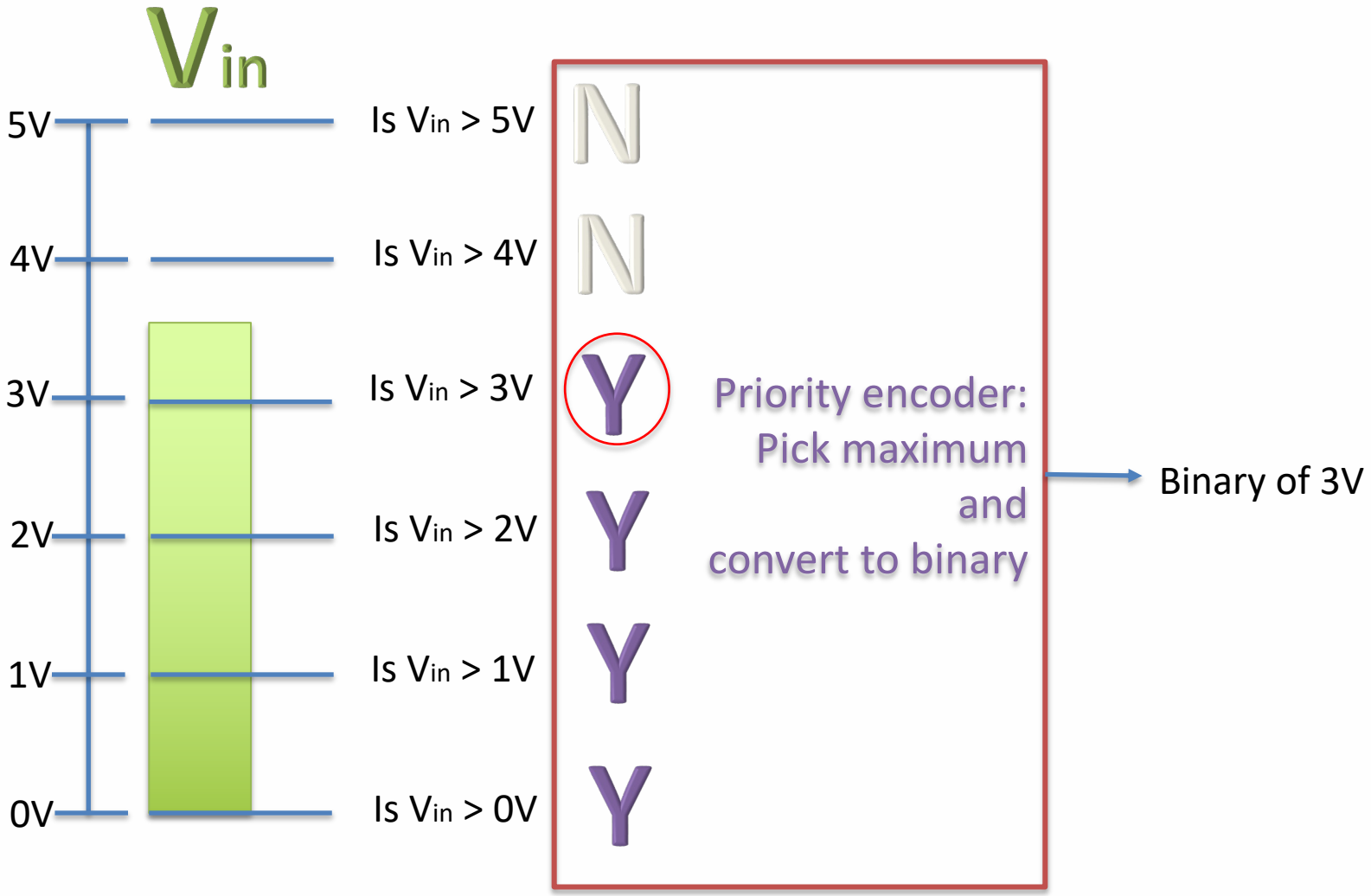
RECAP A2D


Vin
5V
4V
3V
2V
1V
0V
Is Vin > 5V
Is Vin > 4V
Is Vin > 3V
Is Vin > 2V
Is Vin > 1V
Is Vin > 0V
N
N
Y
Y
Y
Y

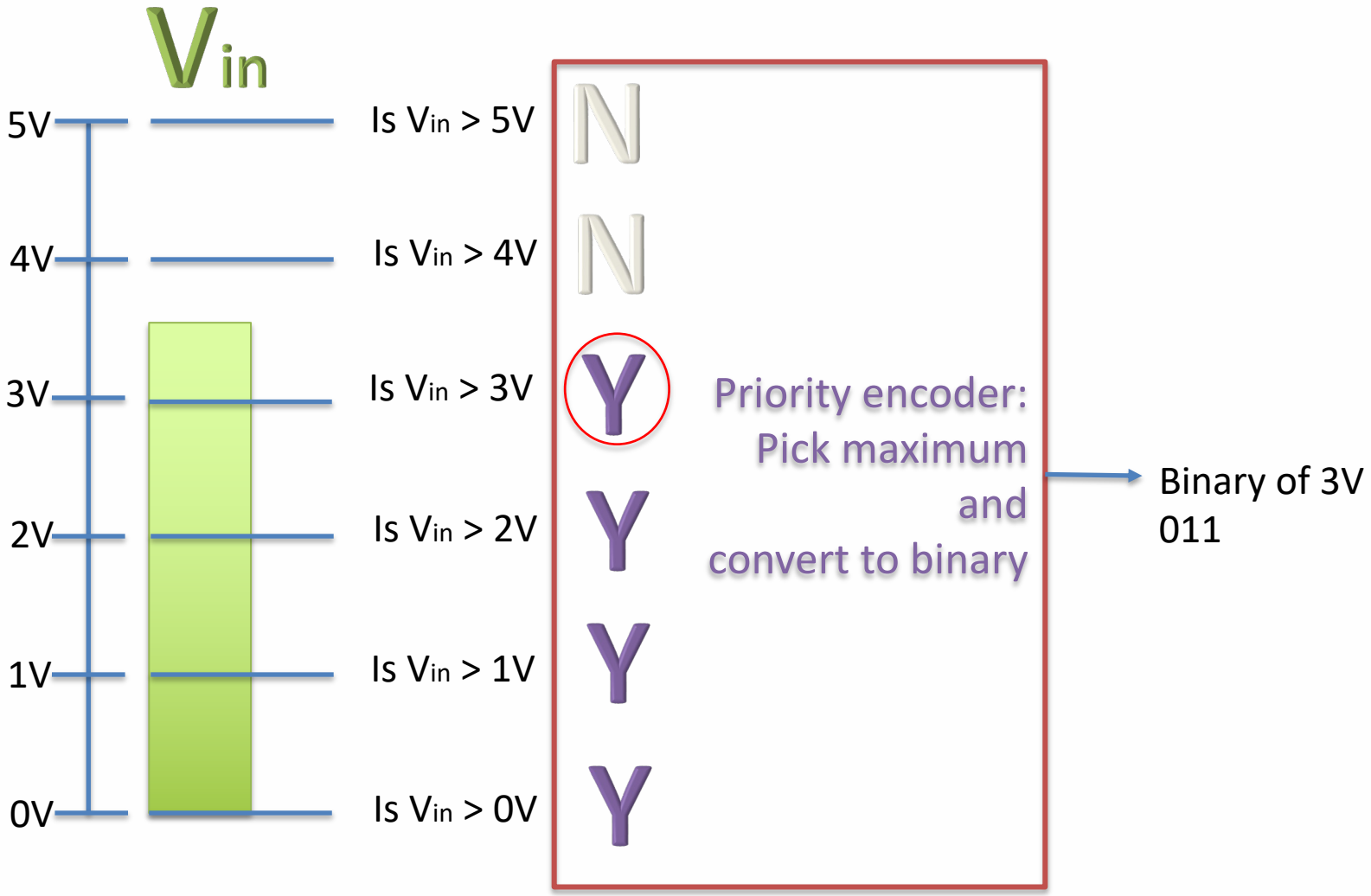
RECAP A2D

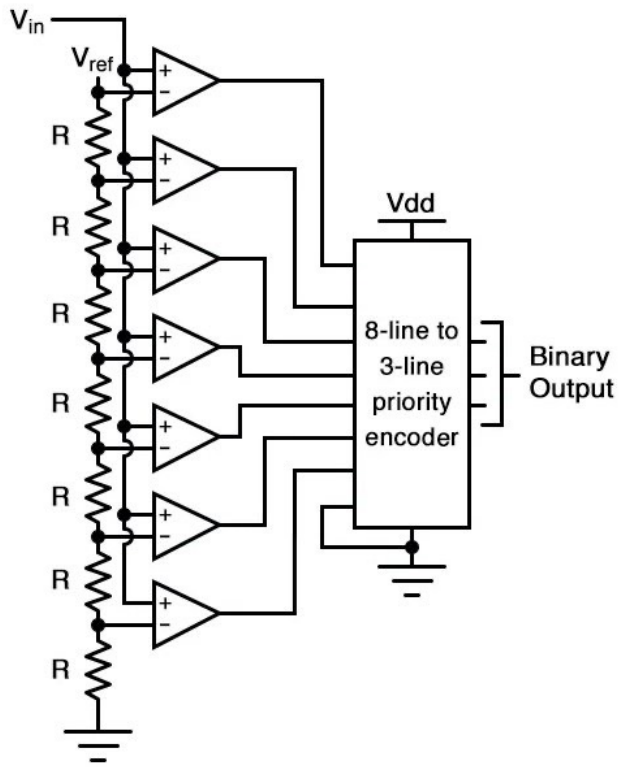

Vin
5V
4V
3V
2V
1V
0V
Is Vin > 5V
Is Vin > 4V
Is Vin > 3V
Is Vin > 2V
Is Vin > 1V
Is Vin > 0V
N
N
Y
Y
Y
Y
Priority encoder:
Pick maximum
and
convert to binary

RECAP A2D


Vin
5V
4V
3V
2V
1V
0V
Is Vin > 5V
Is Vin > 4V
Is Vin > 3V
Is Vin > 2V
Is Vin > 1V
Is Vin > 0V
N
N
Y
Y
Y
Y
Priority encoder:
Pick maximum
and
convert to binary
Binary of 3V

RECAP A2D


Vin
5V
4V
3V
2V
1V
0V
Is Vin > 5V
Is Vin > 4V
Is Vin > 3V
Is Vin > 2V
Is Vin > 1V
Is Vin > 0V
N
N
Y
Y
Y
Y
Priority encoder:
Pick maximum
and
convert to binary
Binary of 3V
011


Vin
Vref
R
R
R
R
R
R
R
Vdd
8-line to
3-line
priority
encoder
Binary
Output

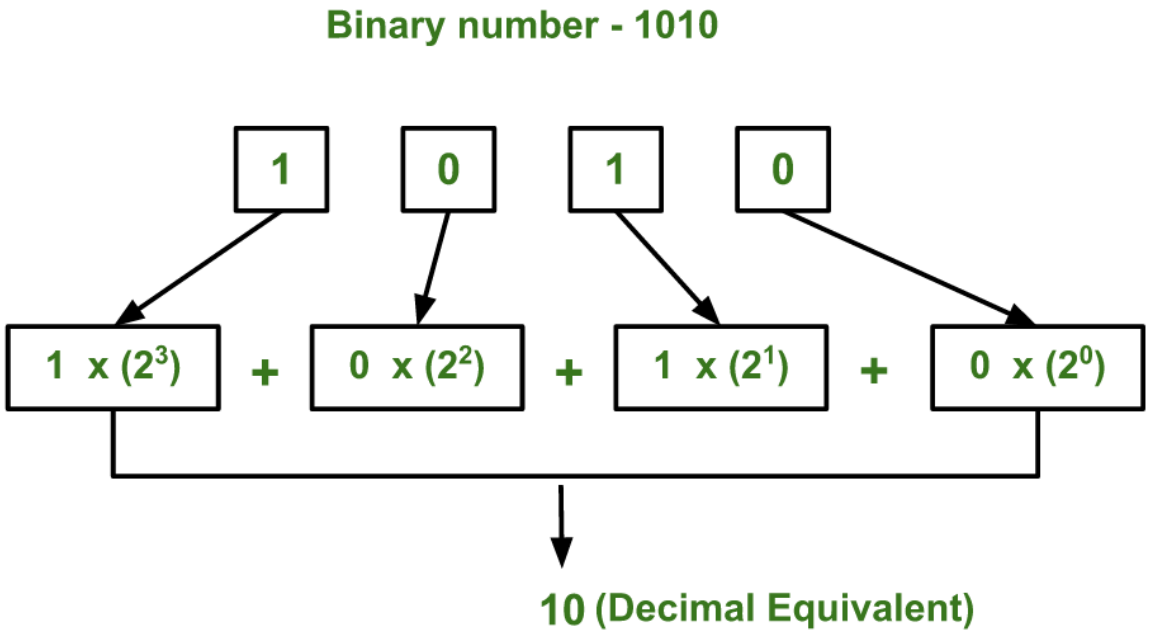
Digital to Analog conversion

Digital to Analog (DAC) Conversion

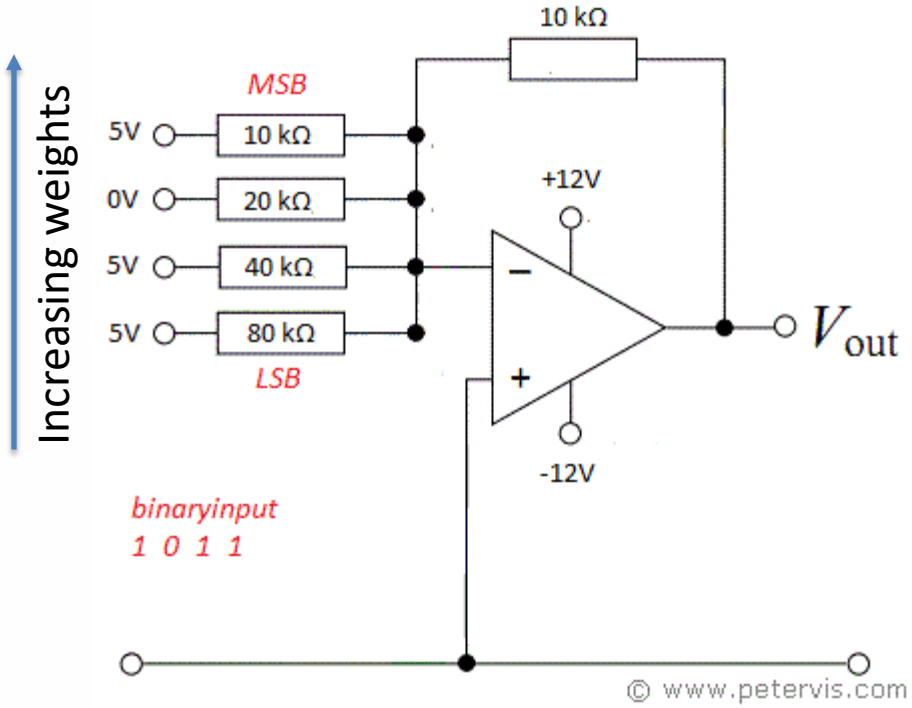
Pulse Width Modulation
(Simulate the effect of analog signal with digital pins)

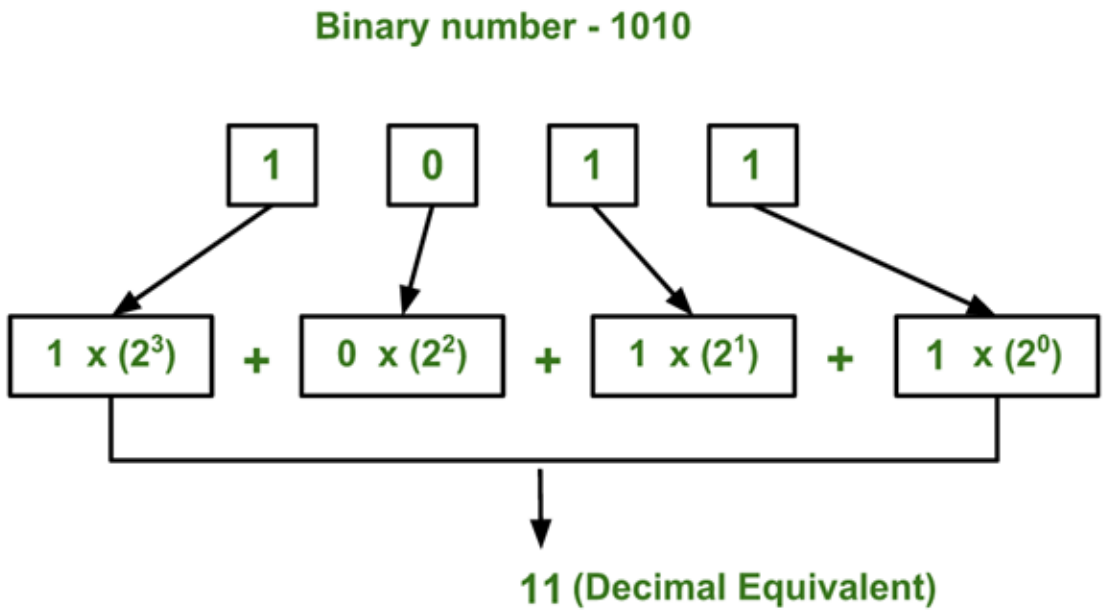
Digital to Analog conversion

Digital to Analog (DAC) Conversion



How to convert digital values to analog values





https://www.petervis.com/Education/summing-amplifier-inverting/4-bit-dac-using-op-amp-example-2.html

How to convert digital values to analog values

ESP32 has two 8-bit DAC (**digital to analog converter**) channels connected to **GPIO25 (Channel 1) and GPIO26 (Channel 2)**.

- Independent or simultaneous conversion in channels
- Configurable Voltage reference levels
- Cosine waveform (CW) generator!
  Can be used to generate a cosine / sine tone
- Direct Memory Access capability

..

Pins - Analog I/0 (DAC)

```
pinMode(pin, ANALOG);
```

```
dacWrite(pin, value)
```

- **Changes the analog value** on one of the pins marked DAC.
- Value may be a number between **0 and 255** that represents the scale between **0 and 3.3 V** output voltage.

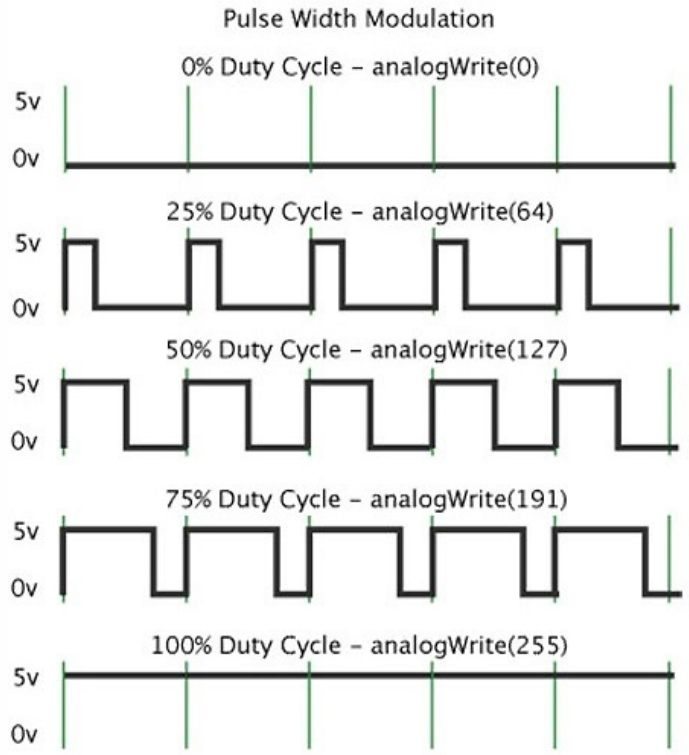
Example:
Assuming VDD of 3.3V, set GPIO25 to 2.59V:

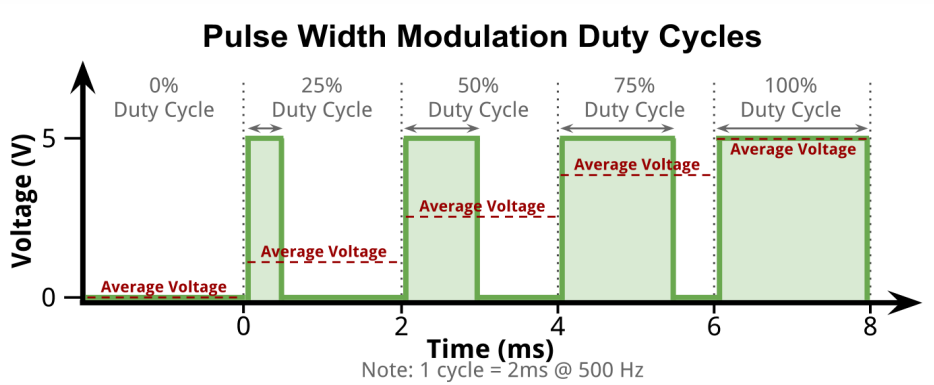
- 2.59V is approximately 0.78% of VDD voltage
- Or, 2.59V = (VDD * 200 / 255)
- `dacWrite(25, 200)`

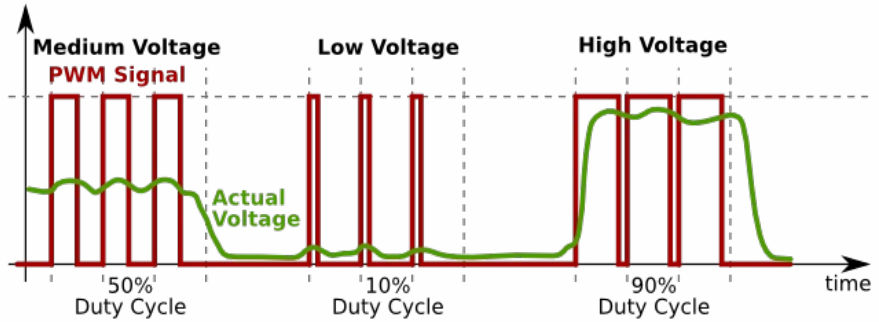
Pins - Analog I/0 (DAC)

`analogWrite(pin, value)`

- **Changes the PWM (pulse width modulation) rate** on one of the pins marked PWM.
- How do we simulate analog signals with digital signals
- Value may be a number between **0 and 255** that represents the scale between **0 and 3.3 V** output voltage.







Functions - Timing

- `sleep(s)`
  - Pauses for few seconds
- `delay(ms)`
  - Pauses for a few milliseconds
- `delayMicroseconds(us)`
  - Pauses for a few microseconds
- `millis()`
  - Returns the number of milliseconds that have **passed** since the program started.

```
duration = millis()-lastTime; // computes time elapsed since "lastTime"
```

Functions – Math and Operators

- The **equals** sign

  = is used to assign a value

  == is used to compare values

- **And** & **Or**

  && is “and”

  || is “or”

- **Math** Functions

```
min(x, y)
max(x, y)
abs(x)
sqrt(x)
map(value, fromLow, fromHigh, toLow, toHigh)
    val = map(analogRead(0),0,1023,100, 200);
 sin(rad)
    double sine = sin(2); // approximately 0.90929737091
 cos(rad)
 tan(rad)
 pow(base, exponent)
    double x = pow(y, 32); // sets x to y^32
constrain(x, a, b)
    val = constrain(analogRead(0), 0, 255);
                        // reject values bigger than 255
```

Functions - Random Numbers

- `void randomSeed(unsigned long seed)`
  - Resets Arduino's pseudorandom number generator. Although the **distribution** of the numbers returned by random() is essentially **random**, the **sequence** is **predictable**. So, you should reset the generator to some random value.
  - If you have an **unconnected analog pin**, it will pick up random noise from the surrounding environment (*radio waves, cosmic rays, electromagnetic interference from cell phones and fluorescent lights, and so on*).
  - randomSeed(analogRead(5)); // randomize using noise from pin 5

- `long random(long howsmall, long howbig)`
  - Returns a pseudorandom long integer value between min and max – 1.
  - long randnum = random(0, 100); // a number between 0 and 99

Variables – Declaring and Assigning

- Basic variable types:
  - Boolean:

```
boolean variableName;
variableName = true;
```

  - Integer:

```
int variableName;
variableName = 32767;
```

  - Character:

```
char variableName;
variableName = 'A';
```

  - String:

```
stringName[];
stringName = "SparkFun";
```

C++ KEYWORD	SIZE	DESCRIPTION
boolean	1 byte	Holds only two possible values, **true** or **false**, even though it occupies a byte in memory.
char	1 byte	Holds a number from -127 to 127. Because it is marked as a "char," the compiler will try to match it to a character from the ASCII table of characters.
byte	1 byte	Can hold numbers from 0 to 255.
int	2 byte	Can hold numbers from -32768 to 32767.
unsigned int	2 byte	Can hold numbers from 0-65535
word	2 byte	Same as the "unsigned int." People often use "word" for simplicity and clarity.
long	4 byte	Can hold numbers from -2,147,483,648 to 2,147,483,647.
unsigned long	4 byte	Can hold numbers from 0-4,294,967,295
float	4 byte	Can hold numbers from -3.4028235E38 to 3.4028235E38. Notice that this number contains a decimal point. Only use float if you have no other choice. The ATMEGA CPU does not have the hardware to deal with floats, so the compiler has to add a lot of code to make it possible for your sketch to use them, making your sketch larger and slower.
string - char array	-	A way to store multiple characters as an array of chars. C++ also offers a String object that you can use instead that provides more flexibility when working with strings in exchange for higher memory use.
array	-	A structure that can hold multiple data of the same type.

Size can be confusing when changing boards

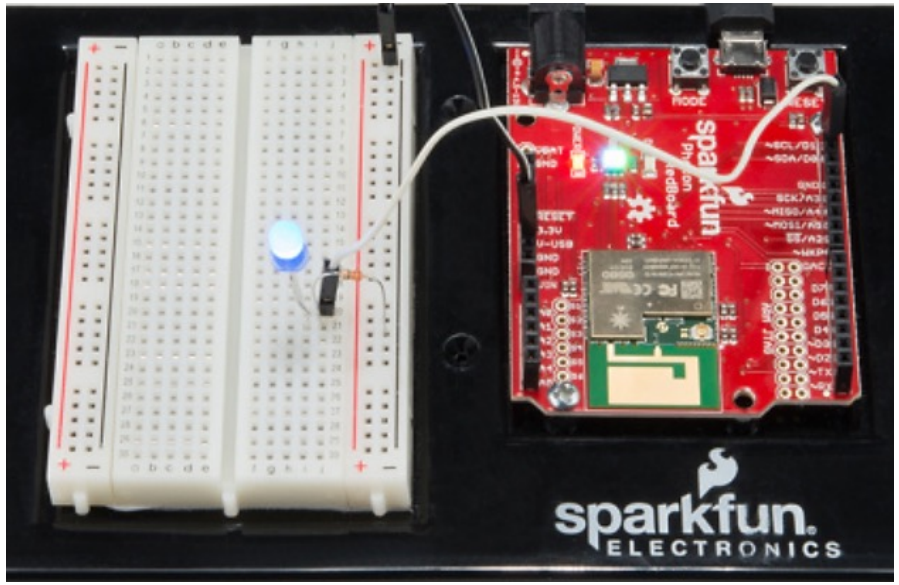
- Explicitly specify size:
- `int8_t ,uint8_t`
- `int16_t ,uint16_t`
- `int32_t ,uint32_t`
- `int64_t ,uint64_t`

- Floating point arithmetic:
- `float, double`

Variables – Declaring and Assigning

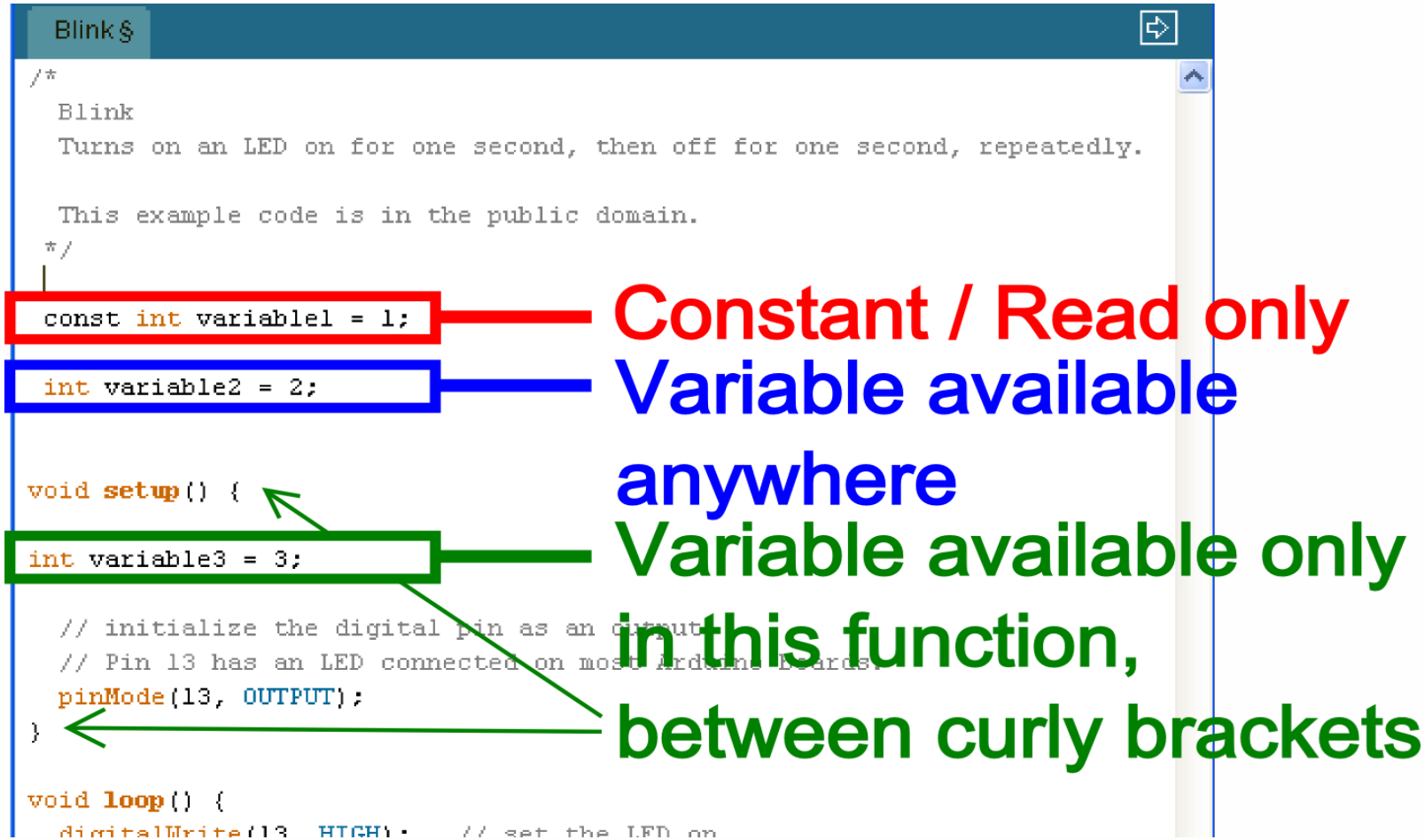
- Array:
  - A list of variables accessed via an index.
  - For example, storing different levels of brightness to be used when fading an LED.

```
int light[5] = {0, 25, 50, 75, 100};
```



Variables – Scope

- Where you declare your variables matters



Spot the bug!

```
void loop()
{
    for (uint8_t i = 0; i < 255; i++)
    {
        dacWrite(outPin, i);
        delay(10);
    }

    for (uint8_t i = 255; i >= 0; i--)
    {
        dacWrite(outPin, i);
        delay(10);
    }
}
```