# Design with Microprocessors

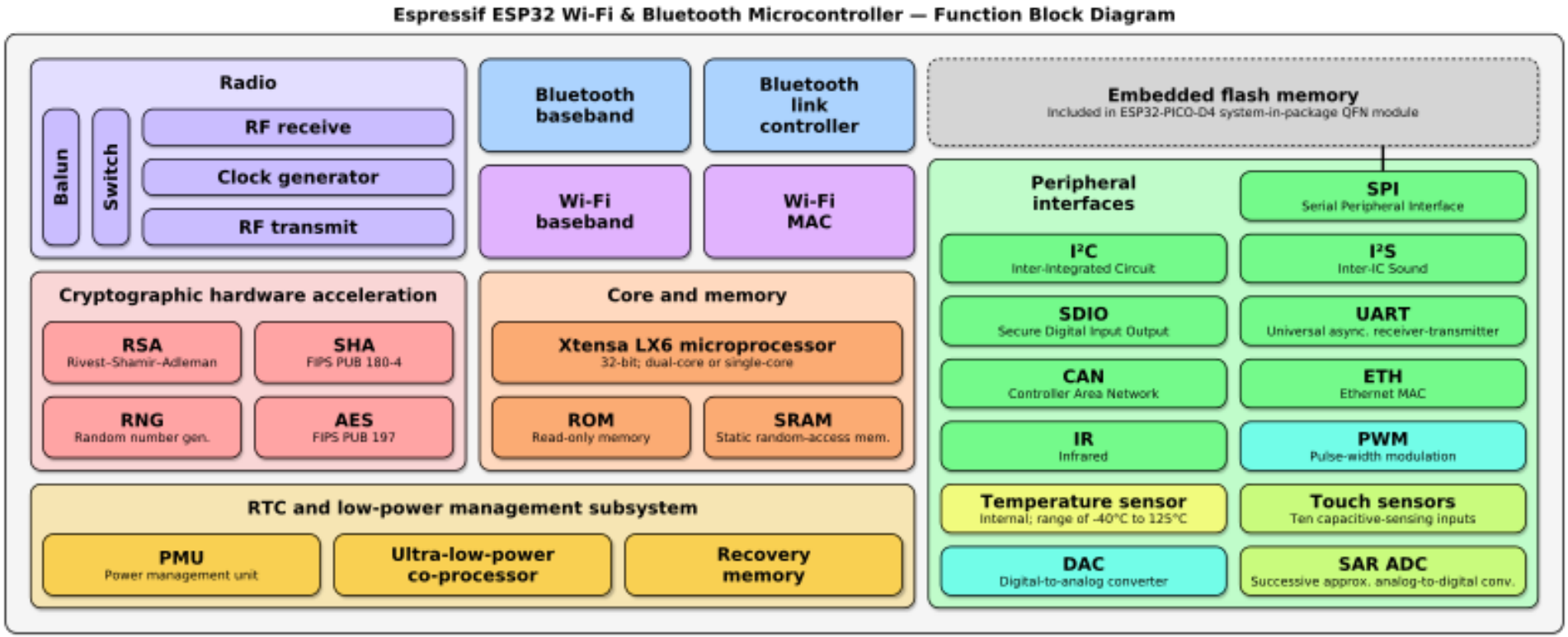
## Lecture 10

## The ESP32 Microcontroller

**Year 3 CS**
**Academic year 2023/2024**
**1st Semester**

**Lecturer: Radu Dănescu**

# ESP32 features




From:
https://en.m.wikipedia.org/wiki/File:Espressif_ESP32_Chip_Function_Block_Diagram.svg

# ESP32 vs Arduino

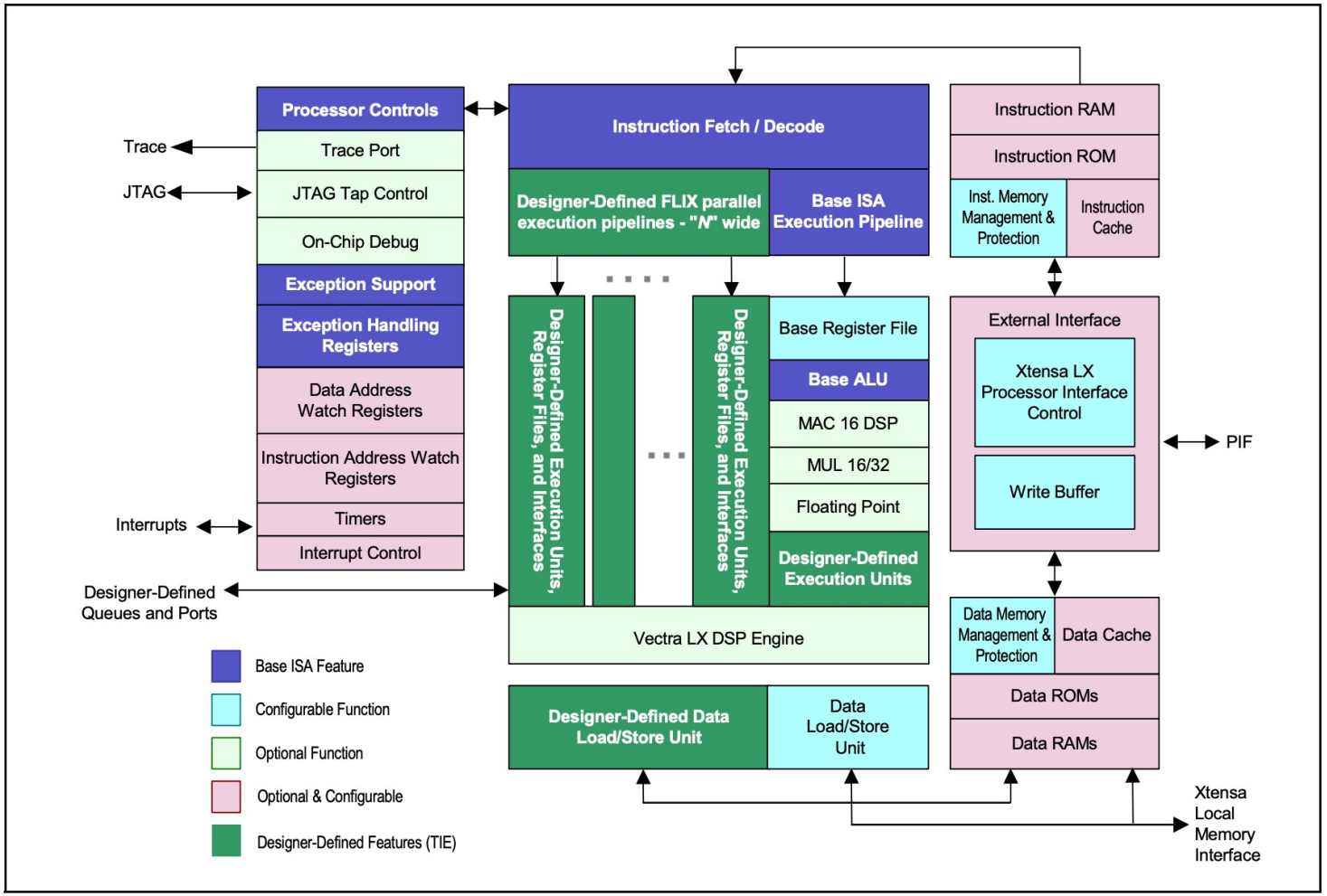
| Function | ESP32 | Arduino UNO | Arduino Mega2560 |
|---|---|---|---|
| Microcontroller | Xtensa Dual Core 32-bit LX6 microprocessor | ATMega328P | ATMega2560 |
| Flash Memory | 4MB | 32KB | 256 KB |
| SRAM | 520KB | 2KB | 8KB |
| EEPROM | Not Available | 1KB | 4KB |
| CLOCK Speed | Upto 240 MHz | 16MHZ | 16 MHz |
| Operating Voltage | 3.3V DC | 5V DC | 5V DC |
| Input Voltage | 3.3V DC | 6V-20V DC | 7-12V |
| Current consumption | 80 mA – 90mA | 45 mA – 80 mA | 150 uA |
| DC Current per I/O Pin | 40 mA | 20 mA | 20 mA |
| DC Current For 3.3V Pin | 50 mA | 50 mA | 50 mA |
| Digital IO Pins | 36 | 14 | 54 |
| Analog Input Pins | Up to 18 | 6 | 16 |
| UARTs | 3 | 1 | 4 |
| SPI | 4 | 1 | 1 |
| I2C | 2 | 1 | 1 |
| CAN | Yes | No | No |
| PWM | 16 | 6 | 14 |
| Wi-Fi | Yes,802.11 b/g/n | No | No |
| Bluetooth | Yes, Bluetooth v4.2 BR/EDR and BLE specification | No | No |

From: https://www.makerguides.com/esp32-vs-arduino-speed-comparison/

# ESP32 Processor and Memory

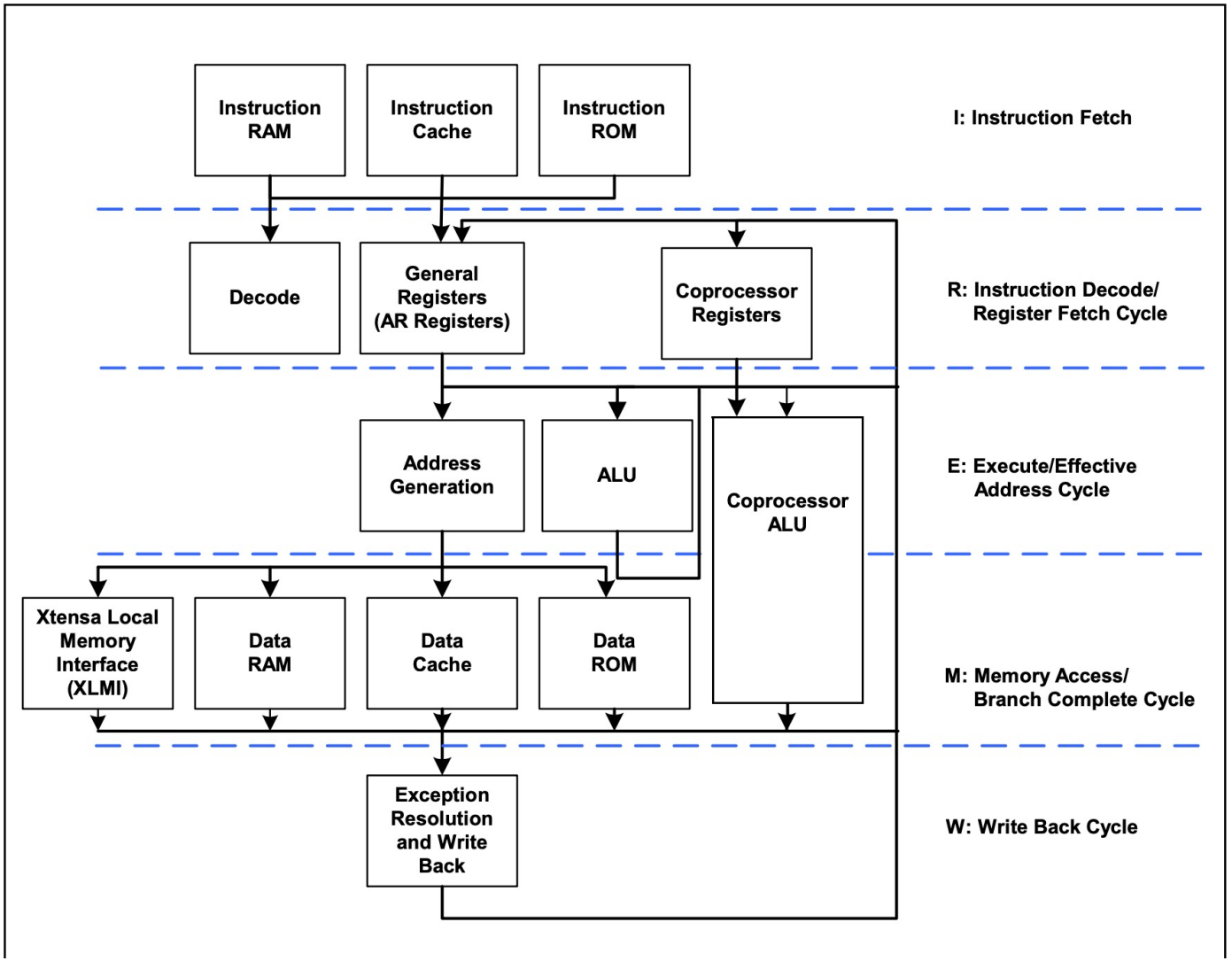
**Main processor:** Tensilica Xtensa 32-bit LX6 microprocessor
**Cores:** 2
**Clock frequency:** up to 240 MHz
**Ultra low power (ULP) co-processor:** allows some operations while in deep sleep.
**Internal memory:**
ROM: 448 Kb For booting and core functions.
SRAM: 520 Kb For data and instruction.
RTC slow SRAM: 8 Kb For co-processor accessing during deep-sleep mode.
RTC fast SRAM: 8 Kb For data storage and main CPU during RTC Boot from the deep-sleep mode.

# Generic Xtensa core architecture



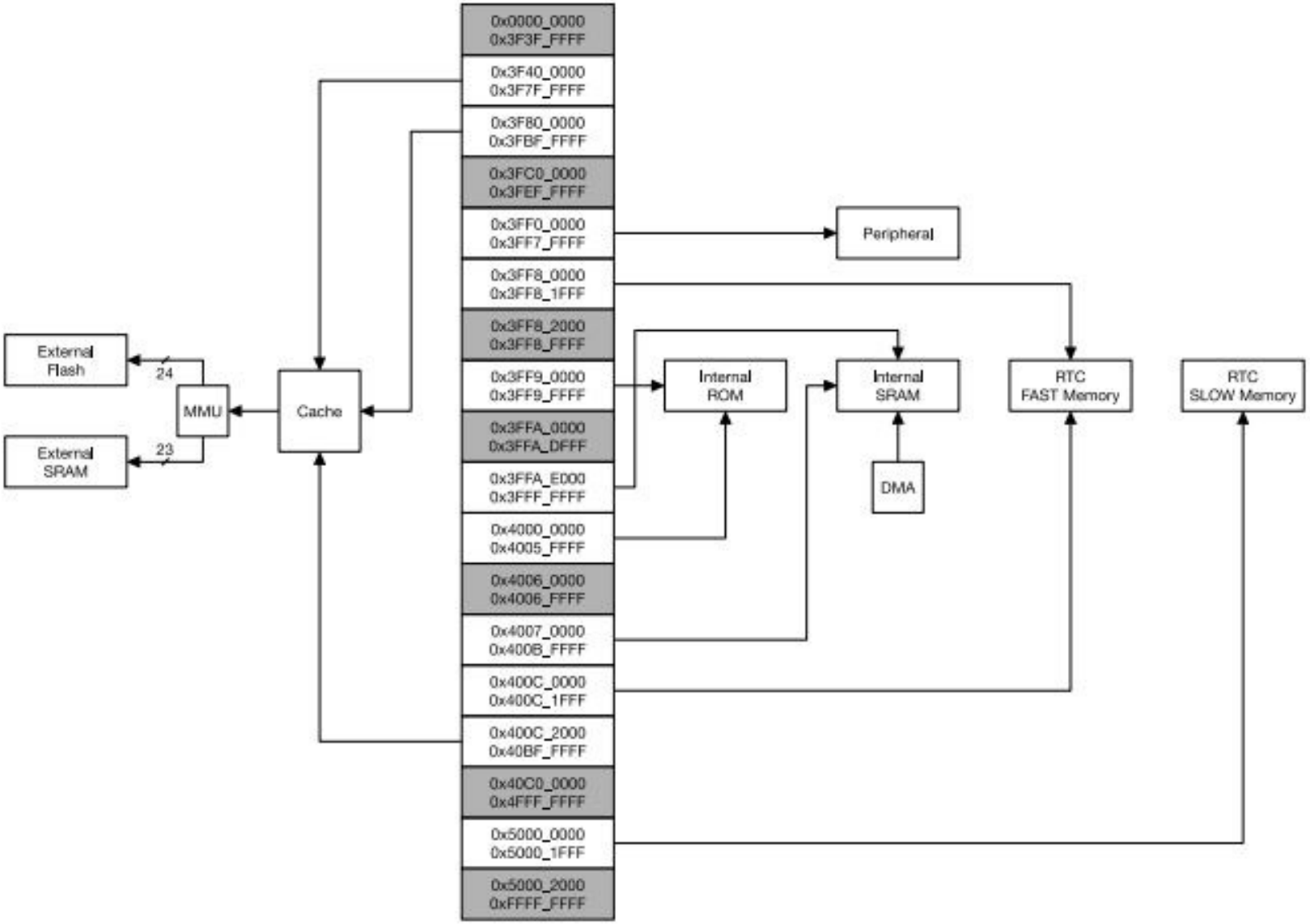

Xtensa® Instruction Set Architecture (ISA) Reference Manual
https://0x04.net/~mwk/doc/xtensa.pdf

# Xtensa instruction execution pipeline




Xtensa® Instruction Set Architecture (ISA) Reference Manual
https://0x04.net/~mwk/doc/xtensa.pdf

# Address Space - 32 bit addressing

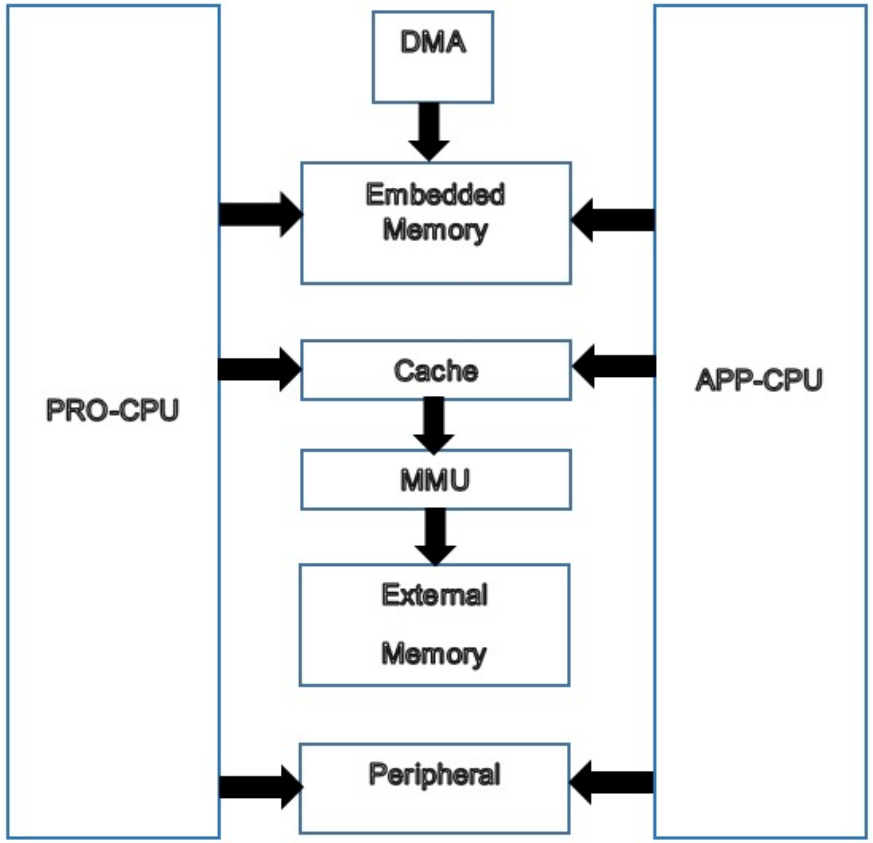



From: https://forum.arduino.cc/t/how-to-map-memory-in-esp32/964205/2

# Cores

2 Tensilica LX6 cores
PRO
APP

240 MHz maximum frequency




From: https://linuxhint.com/what-chip-esp32/

# Instruction Set Architecture

Load/Store architecture, with 24-bit and 16-bit instructions
Register file contains 16 registers named a0 through a15.

a0 holds the call return address.
a1 used as stack pointer.
a2 used for function arguments and return value.

Most instructions specify 3 registers: the first register is the destination, the remaining registers are operands.

| | | |
|---|---|---|
| **add** | a1, a2, a3 | //a1 = a2 + a3. |
| **addi** | a1, a2, N | // a1 = a2 + N. N is in the range [-127, 128]. |
| **sub** | a1, a2, a3 | // a1 = a2 - a3 |

# Instruction Set Architecture

Loads and stores are done using a base register and an 8 bit unsigned offset.

```
l32r    a2, 400011a0    // load from absolute address 400011a0
l32i.n  a3, a2, 0       // a3 = MEM [a2+0] - a2 is used as pointer
addi.n  a3, a3, 1       // a3 = a3 + 1
s32i.n  a3, a2, 0       // MEM[a2 + 0] = a3
```

The "mov" instruction can work register to register or to move a small immediate value into a register:

```
mov.n   a7, a2          // a7 = a2
movi.n  a3, 0           // a3 = 0
```

.n denotes a 16-bit instruction format

# Instruction Set Architecture

**Conditions and branches**

**Conditional moves**

```
moveqz  a7, a8, a9          // a7 = a8 if a9 == 0
movnez  a4, a3, a11         // a4 = a3 if a11 != 0
movgez  a2, a3, a4          // a2 = a3 if a4 >= 0
movltz  a6, a10, a11        // a6 = a10 if a11 < 0
```

**Branches on comparison between registers**

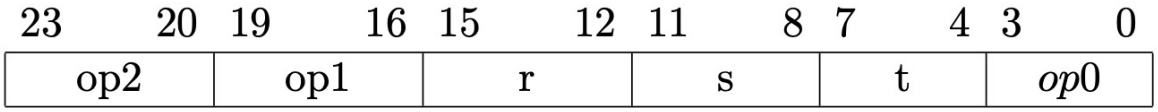
```
beq     a3, a6, 400000e0    // branch if equal
bne     a3, a4, 40000e74    // branch if not equal
bge     a3, a6, 400000e0    // branch if greater or equal
blt     a3, a4, 40000e74    // branch if less then
bgeu    a3, a6, 400000e0    // branch if greater or equal - Unsigned
bltu    a3, a4, 40000e74    // branch if less then - Unsigned
```

# Instruction Set Architecture

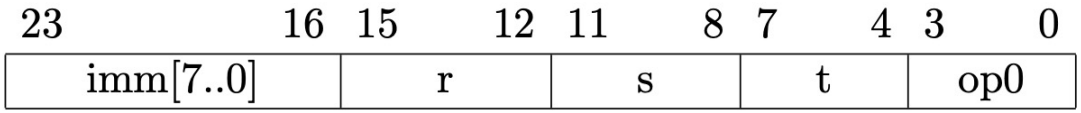
## Instruction formats

op0, op1, op2 4-bit opcode fields
r, s, t – 4-bit operand fields

**RRR Instruction Format**




**RRI8 Instruction Format**




**Arithmetic, logic instructions**

ADD, SUB, OR…

**Conditional moves**
MOVEQZ, …

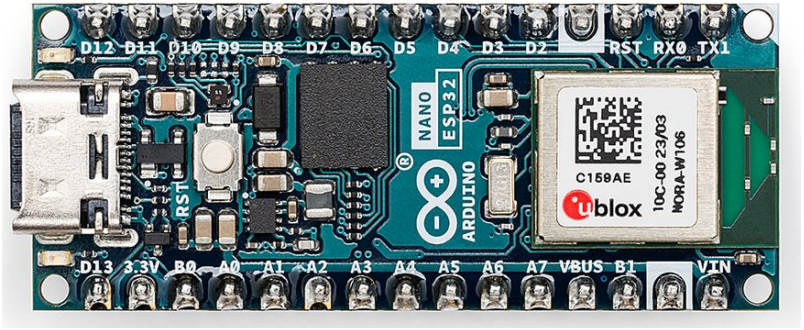
**Branching instructions**

BEQ, BGE, …

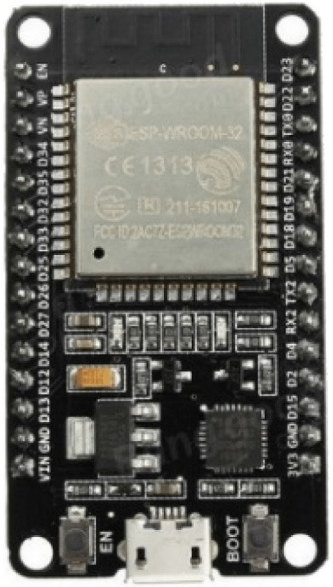
**Load/Store instructions**
L32I, S32I, …

"Overview of Xtensa ISA", Espressif Systems, https://dl.espressif.com/github_assets/espressif/xtensa-isa-doc/releases/download/latest/Xtensa.pdf

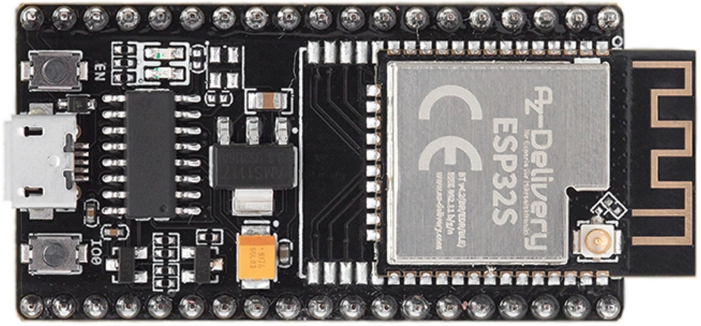
# ESP32 Boards



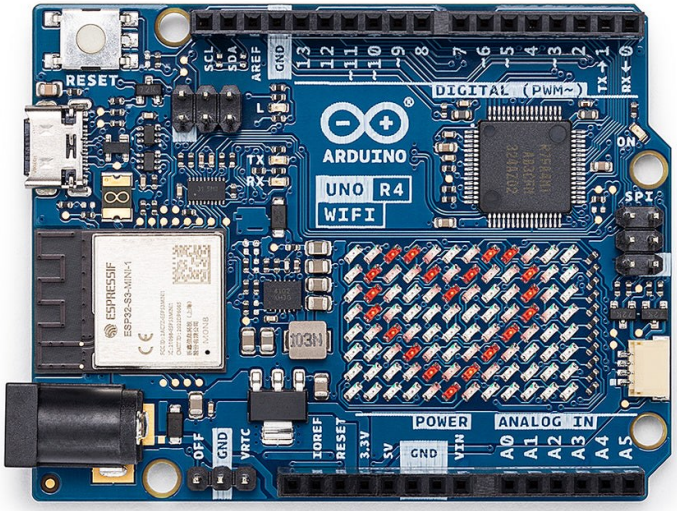

Arduino Nano ESP32




ESP32 DevKit V1

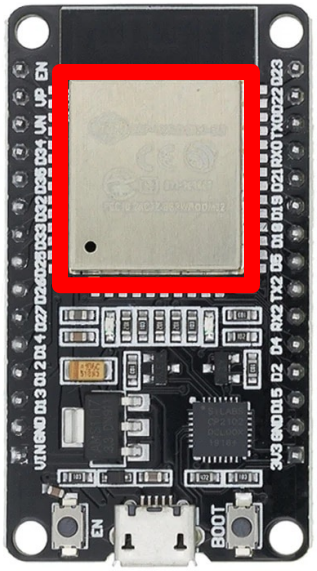


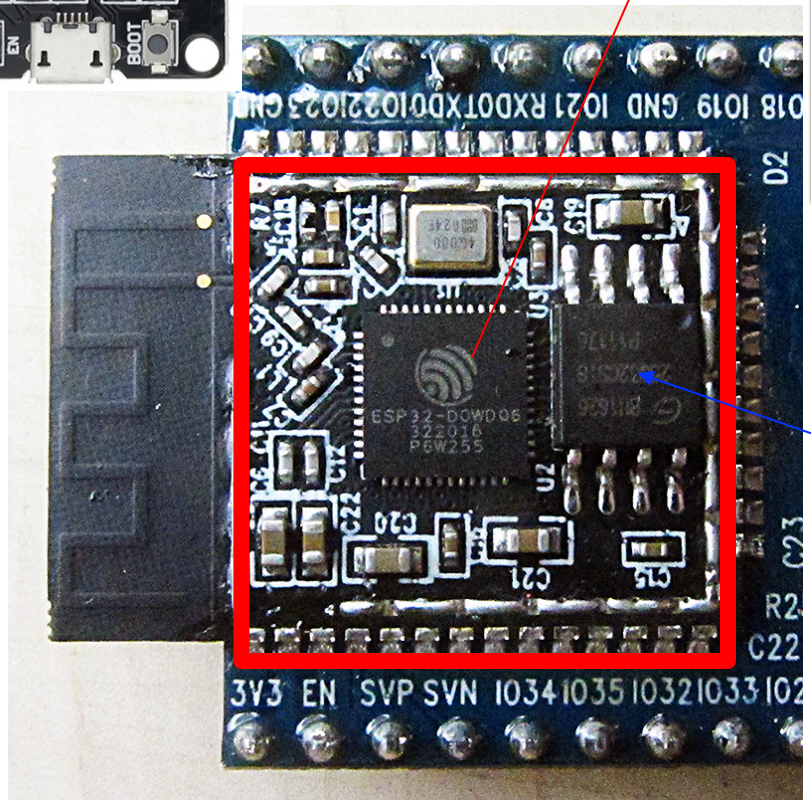

NodeMCU ESP32




Arduino Uno R4 WiFi

# ESP32 WROOM features





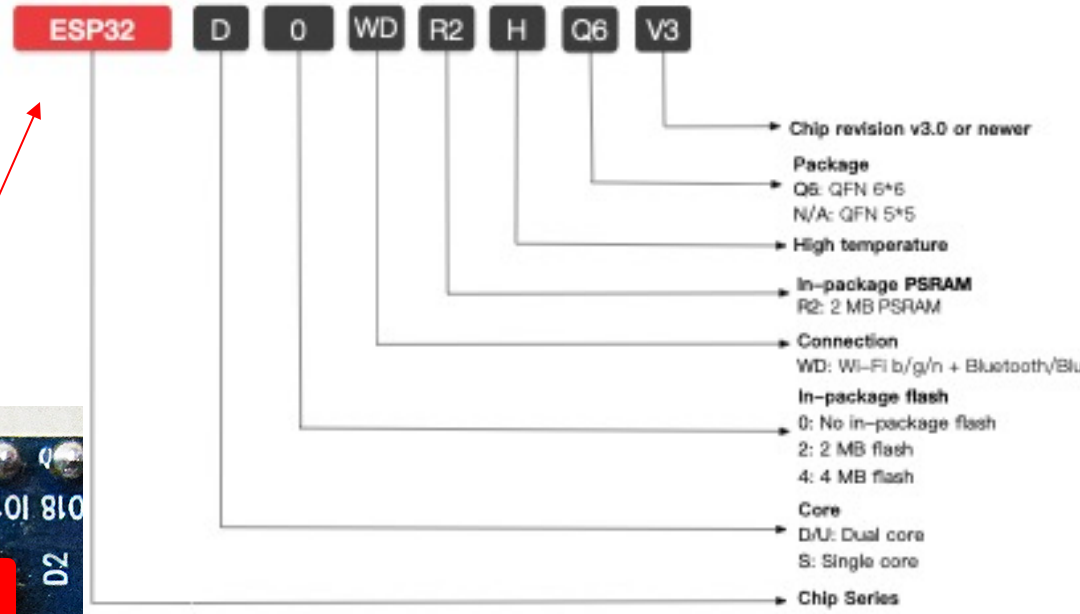




Figure 1-1. ESP32 Series Nomenclature

Flash memory - external!

# Development tools

## ESP IDF

```
#define BLINK_GPIO GPIO_NUM_2

uint8_t s_led_state = 0;

void configure_led(void)
{
  gpio_reset_pin(BLINK_GPIO);
  gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);
}

void blink_led(void)
{
  gpio_set_level(BLINK_GPIO, s_led_state);
}

void app_main(void)
{

   configure_led();

   while (1) {
        blink_led();
        s_led_state = !s_led_state;
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}
```

## ARDUINO

```
#define BLINK_GPIO GPIO_NUM_2

uint8_t s_led_state = 0;

void setup()
{
  pinMode(BLINK_GPIO, OUTPUT);
}

void blink_led(void)
{
  digitalWrite(BLINK_GPIO, s_led_state);
}

void loop()
{
  blink_led();
  s_led_state = !s_led_state;
  delay (1000);
}
```

**MICROPYTHON**

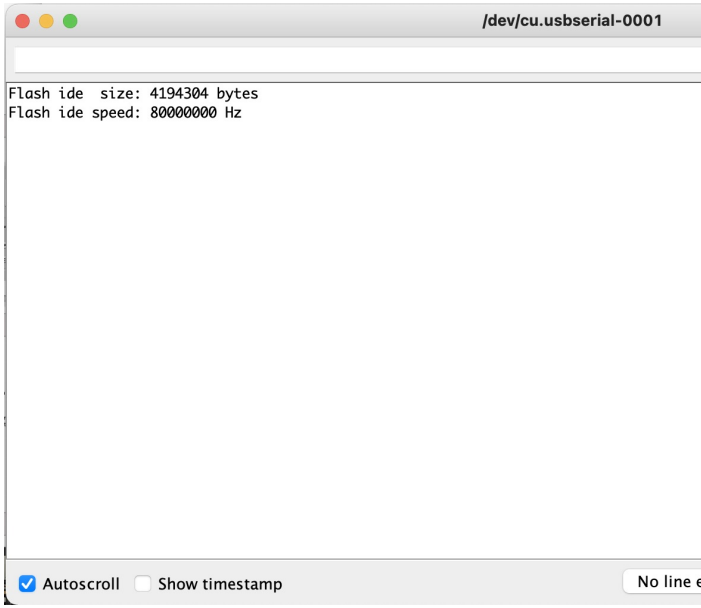
```
import machine
import time
led = machine.Pin(2, machine.Pin.OUT)
while True:
          led.value(1)
          time.sleep(1)
          led.value(0)
          time.sleep(1)
```

# ESP32 External flash

**ESP32 can support up to 16 MB external flash**

**What is the size of our flash?**

```
void setup(void) {
  Serial.begin(115200);
}

void loop() {

  uint32_t ideSize = ESP.getFlashChipSize();

  Serial.printf("Flash real size: %u bytes\n\n",
realSize);

  Serial.printf("Flash ide  size: %u bytes\n", ideSize);
  Serial.printf("Flash ide speed: %u Hz\n",
ESP.getFlashChipSpeed());

  delay(5000);
}
```



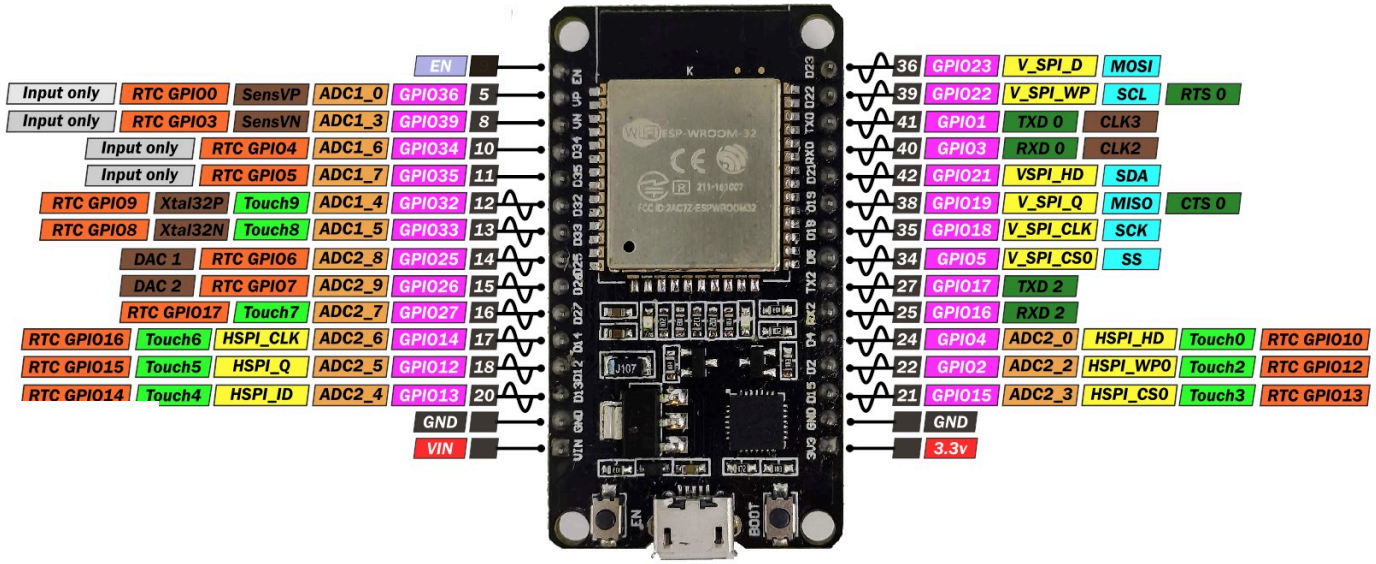

Adapted from
https://gist.github.com/walidamriou/4a3db3b36cb7fd401f9894a1b9d0b8b0

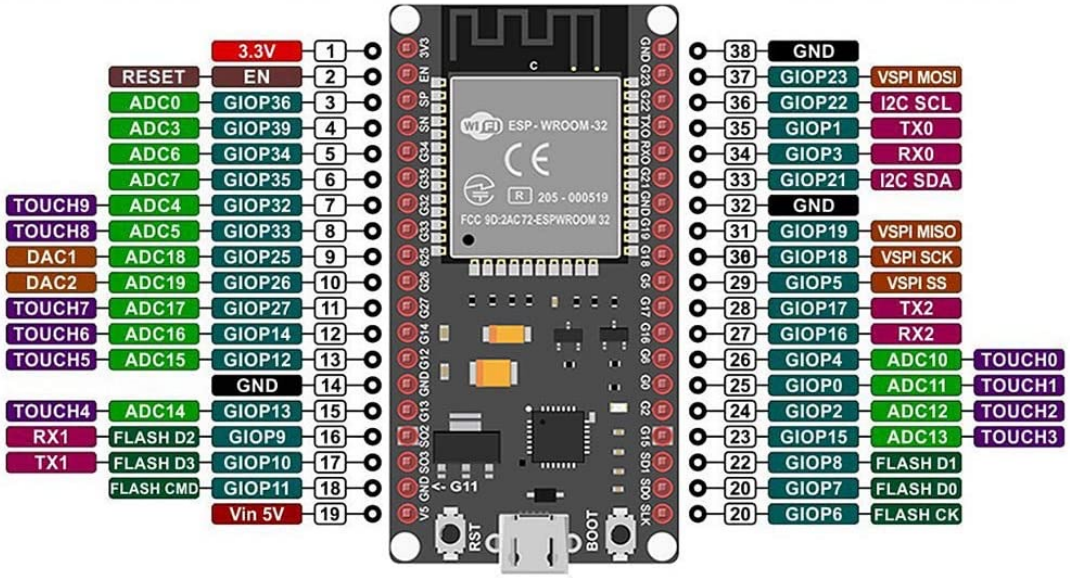
# ESP32 Peripherals

The ESP32 peripherals include:

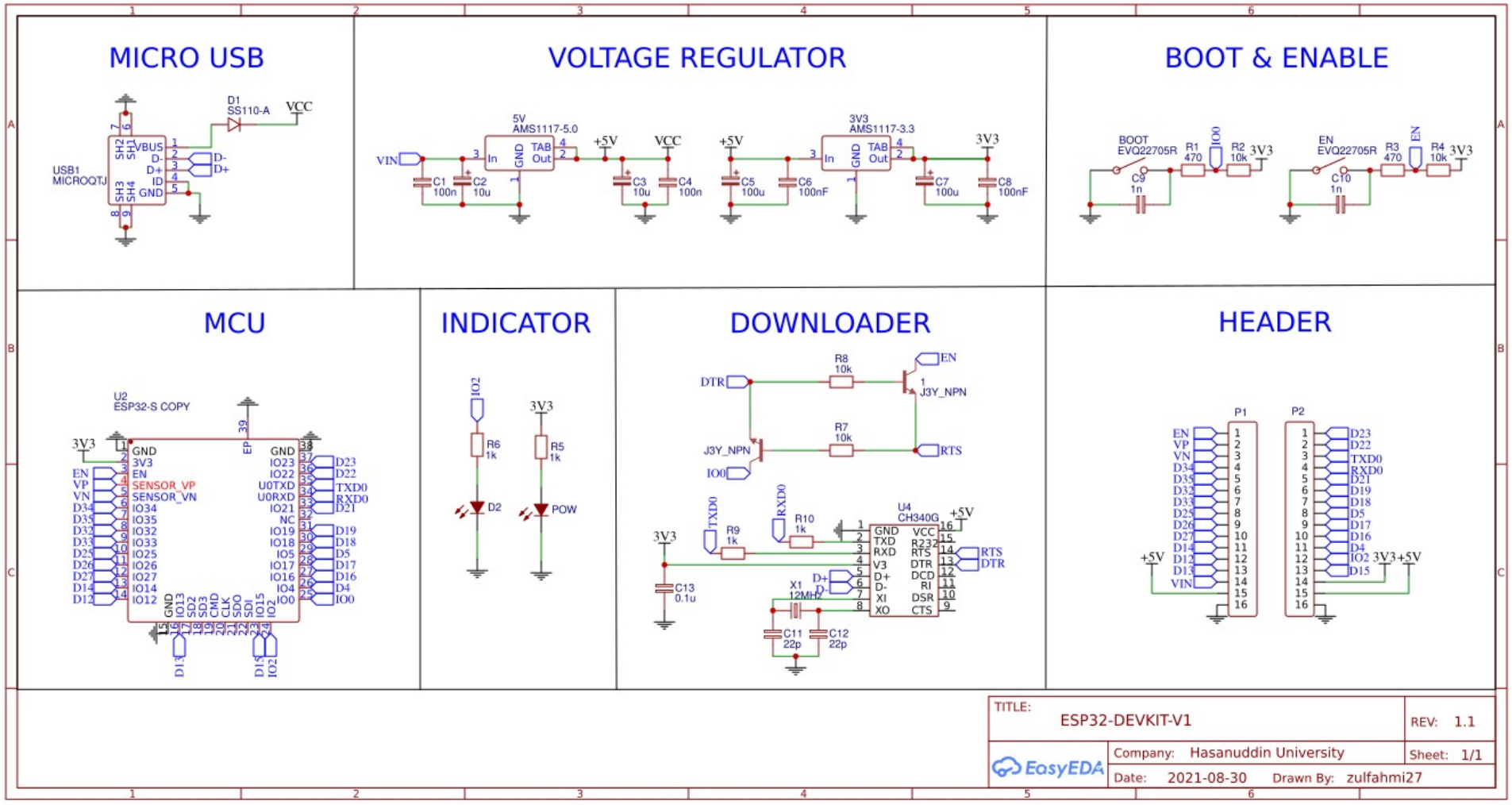
18 Analog-to-Digital Converter (ADC) channels
3 SPI interfaces
3 UART interfaces
2 I2C interfaces
16 PWM output channels
2 Digital-to-Analog Converters (DAC)
2 I2S interfaces
10 Capacitive sensing GPIOs

# ESP32 DevKit v1 Pins



# ESP32 DevKit V1 Schematic



# Voltage range

The ESP32 pins operate between **2.2 and 3.6 V**

- 5 V inputs are NOT ACCEPTABLE.

| Terminals | Symbol | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| Input logic level low | $V_{IL}$ | -0.3 | - | 0.25VDD | V |
| Input logic level high | $V_{IH}$ | 0.75VDD | - | VDD+0.3 | V |
| Output logic level low | $V_{OL}$ | N | - | 0.1VDD | V |
| Output logic level high | $V_{OH}$ | 0.8VDD | - | N | V |

Power input is typically **3.3 V** (VDD)

The ESP32 development boards may accept higher voltages.

**DevKit v1 power input**:

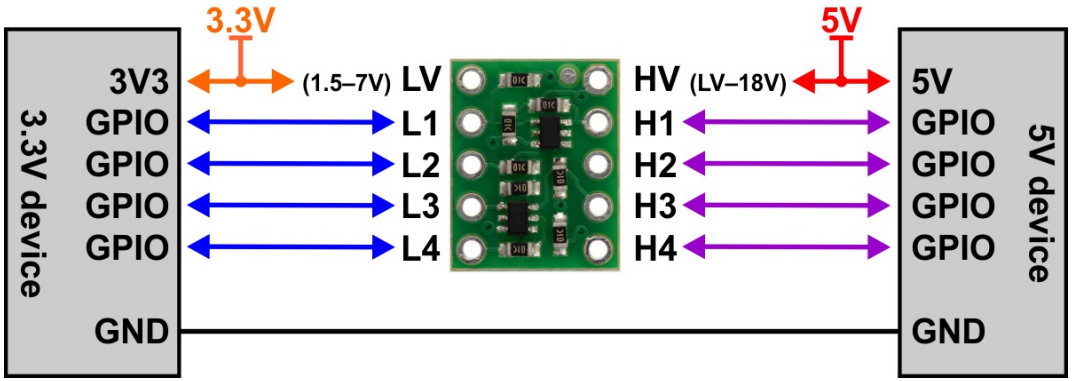
- USB voltage, 5 V
- Vin, from 6 (4?) to 20 V.

In practice, Vin should not exceed 7 V, as the excess is dissipated as heat!

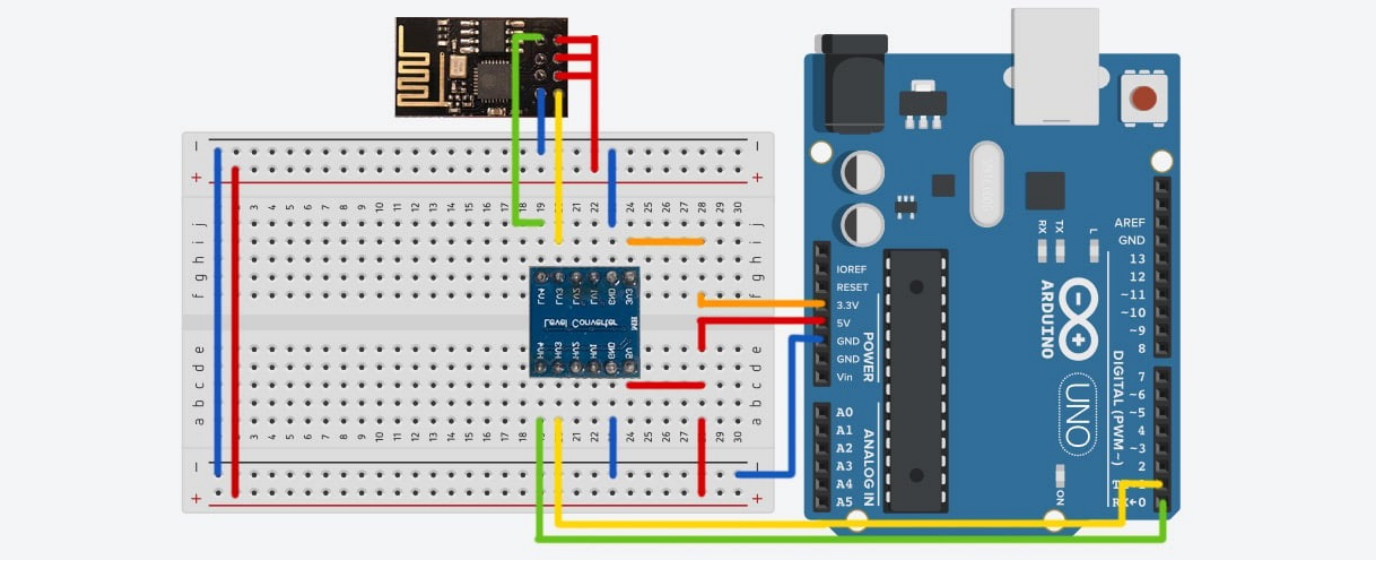
**When powered by USB, can Vin be used as a 5V output?**

# Shifting logic levels

Use of a **Logic Level Shifter**

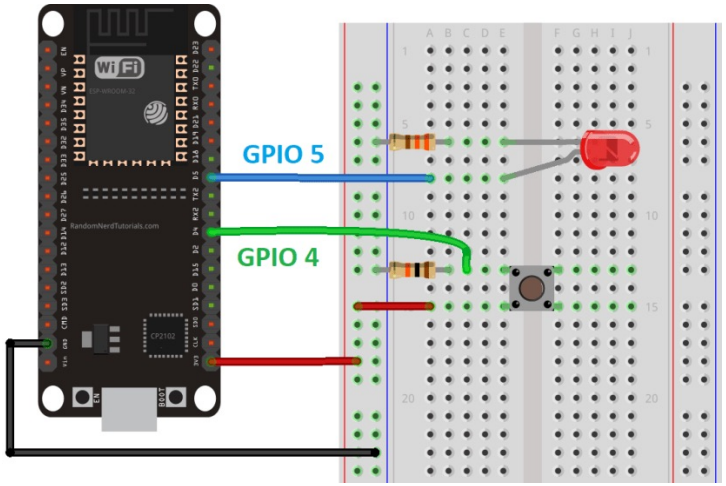



https://community.blynk.cc/t/using-logic-level-converter-to-connect-esp-and-l298/15323




https://maker.pro/arduino/tutorial/how-to-connect-an-arduino-to-modules-with-different-voltages

# Basic I/O Operations




```
const int buttonPin = 4;
const int ledPin = 5;

int buttonState = 0;

void setup() {
   Serial.begin(115200);
   pinMode(buttonPin, INPUT);
   pinMode(ledPin, OUTPUT);
}

void loop() {
 value buttonState = digitalRead(buttonPin);
 Serial.println(buttonState);
 if (buttonState == HIGH) { // turn LED on
        digitalWrite(ledPin, HIGH);
        }
      else {
        digitalWrite(ledPin, LOW);
        }
}
```

From https://randomnerdtutorials.com/esp32-digital-inputs-outputs-arduino/

# Using the two cores

- The Arduino IDE supports FreeRTOS for the ESP32 (Real Time Operating system)
- FreeRTOS allows the creation of multiple TASKS to be run in parallel
- Each Task can be assigned to a core, and will have a priority
- The Tasks can run in parallel wit the Loop function

```
TaskHandle_t Task1;
TaskHandle_t Task2;

// LED pins
const int led1 = 2;
const int led2 = 4;


//Task1code: blinks an LED every 1000 ms
void Task1code( void * pvParameters )
{
 Serial.print("Task1 running on core ");
 Serial.println(xPortGetCoreID());

for(;;) {
 digitalWrite(led1, HIGH);
delay(1000);
digitalWrite(led1, LOW);
delay(1000); }

}
```

```
//Task2code: blinks an LED every 700 ms

void Task2code ( void * pvParameters )
{
  Serial.print("Task2 running on core ");
  Serial.println(xPortGetCoreID());

for(;;) {
 digitalWrite(led2, HIGH);
 delay(700);
 digitalWrite(led2, LOW);
 delay(700); }

}
```

Adapted from
https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/

# Using the two cores

```
void setup() {

 Serial.begin(115200);

 pinMode(led1, OUTPUT);
 pinMode(led2, OUTPUT);
 //create a task that to execute Task1Core on core 0
 xTaskCreatePinnedToCore( Task1code, /* Task function. */
 "Task1", /* name of task. */
 10000, /* Stack size of task */
 NULL, /* parameter of the task */
 1, /* priority of the task */
 &Task1, /* Task handle to keep track of created task */
 0); /* pin task to core 0 */

delay(500);
 //create a task that to execute Task1Core on core 1
xTaskCreatePinnedToCore( Task2code, /* Task function. */
 "Task2", /* name of task. */
 10000, /* Stack size of task */
 NULL, /* parameter of the task */
 1, /* priority of the task */
 &Task2, /* Task handle to keep track of created task */
 1); /* pin task to core 1 */
delay(500);

}
```

```
// Loop can also be executed!
// It will run on CORE 1

void loop() {

   Serial.print("Loop running on core ");
   Serial.println(xPortGetCoreID());

   digitalWrite(led1, HIGH);
   delay(200);
   digitalWrite(led1, LOW);
   delay(200);

   digitalWrite(led1, HIGH);
   delay(200);
   digitalWrite(led1, LOW);
   delay(200);

   delay (5000);

}
```

# Using the two cores

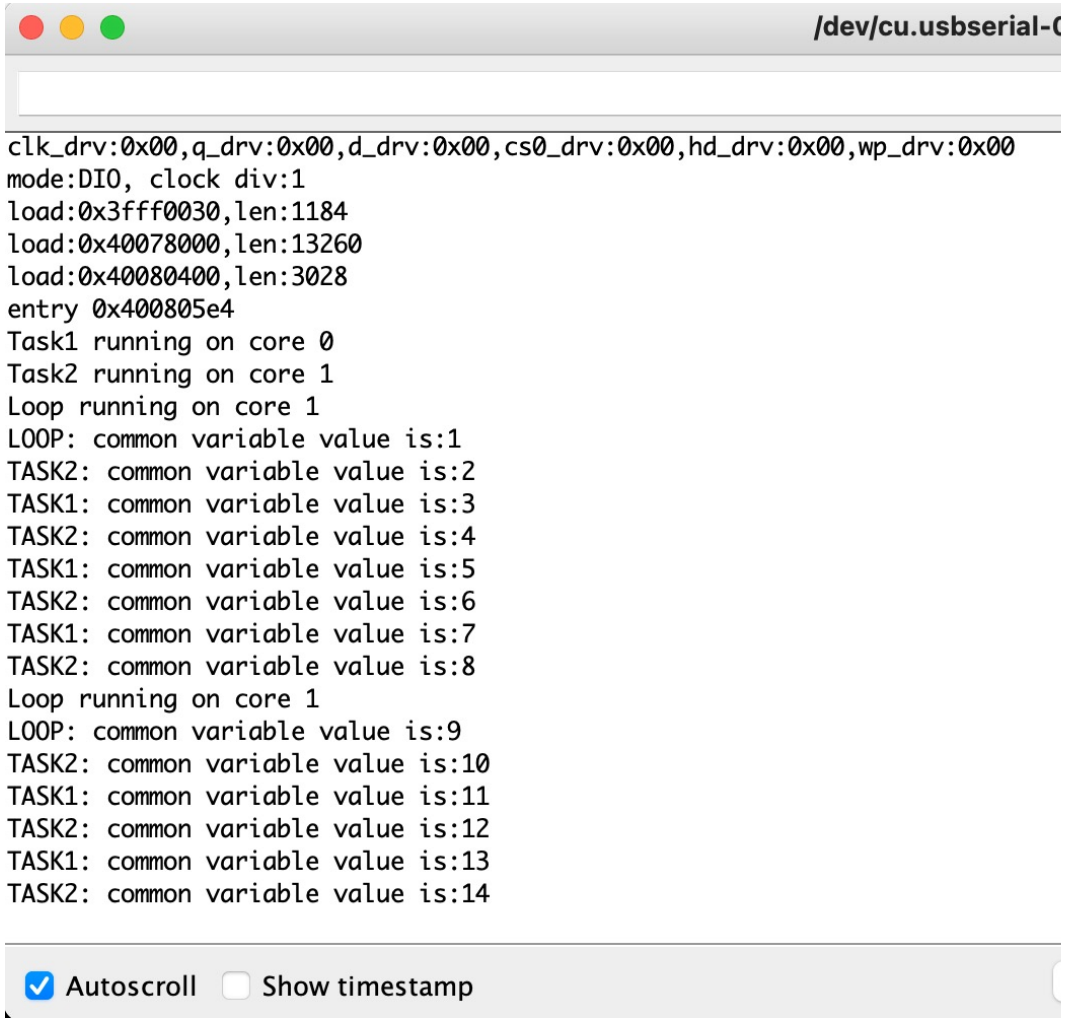
# Using the two cores

```
TaskHandle_t Task1;
TaskHandle_t Task2;

int cc;

// LED pins
const int led1 = 2;
const int led2 = 4;

void setup() {
  Serial.begin(115200);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);

xTaskCreatePinnedToCore(
                Task1code,   /* Task function. */
                "Task1",     /* name of task. */
                10000,       /* Stack size of task */
                NULL,        /* parameter of the task */
                1,           /* priority of the task */
                &Task1,      /* Task handle to keep track of created task */
                0);          /* pin task to core 0 */
  delay(500);

xTaskCreatePinnedToCore(
                Task2code,   /* Task function. */
                "Task2",     /* name of task. */
                10000,       /* Stack size of task */
                NULL,        /* parameter of the task */
                1,           /* priority of the task */
                &Task2,      /* Task handle to keep track of created task */
                1);          /* pin task to core 1 */
    delay(500);
}
```

```
//Task1code: blinks an LED every 1000 ms
void Task1code( void * pvParameters ){
  Serial.print("Task1 running on core ");
  Serial.println(xPortGetCoreID());

  for(;;){
    digitalWrite(led1, HIGH);
    delay(1000);
    digitalWrite(led1, LOW);
    delay(1000);
    cc++;
    Serial.print("TASK1: common variable value is:");
    Serial.println(cc);

  }
}

//Task2code: blinks an LED every 700 ms
void Task2code( void * pvParameters ){
  Serial.print("Task2 running on core ");
  Serial.println(xPortGetCoreID());

  for(;;){
    digitalWrite(led2, HIGH);
    delay(700);
    digitalWrite(led2, LOW);
    delay(700);
    cc++;
    Serial.print("TASK2: common variable value is:");
    Serial.println(cc);
  }
}
```

# Using the two cores

```
void loop() {

    Serial.print("Loop running on core ");
    Serial.println(xPortGetCoreID());

    cc++;
    Serial.print("LOOP: common variable value is:");
    Serial.println(cc);

    digitalWrite(led1, HIGH);
    delay(200);
    digitalWrite(led1, LOW);
    delay(200);

    digitalWrite(led1, HIGH);
    delay(200);
    digitalWrite(led1, LOW);
    delay(200);

    delay (5000);

}
```



# Pull Up / Pull Down Resistors

| Pin Number | Pin Name | Built-in Pull-Up/ Pull Down Resistors |
|---|---|---|
| 0 | GPIO0 | Yes |
| 2 | GPIO2 | Yes |
| 4 | GPIO4 | Yes |
| 5 | GPIO5 | Yes |
| 12 | GPIO12 | Yes |
| 13 | GPIO13 | Yes |
| 14 | GPIO14 | Yes |
| 15 | GPIO15 | Yes |
| 25 | GPIO25 | Yes |
| 26 | GPIO26 | Yes |
| 27 | GPIO27 | Yes |
| 32 | GPIO32 | Yes |
| 33 | GPIO33 | Yes |
| 34 | GPIO34 | No |
| 35 | GPIO35 | No |
| 36 | GPIO36 | No |
| 39 | GPIO39 | No |

```
const int buttonPin1 = 4;
const int buttonPin2 = 6;

void setup() {
   pinMode(buttonPin1, INPUT_PULLUP);
   pinMode(buttonPin2, INPUT_PULLDOWN);
}
```

# CPU and Bus frequency

**3 Types of frequencies:**

**Crystal frequency** - FIXED!
**CPU Frequency** - multiple of XTAL frequency by the use of a Phase Lock Loop circuit
**APB frequency** - bus frequency

```
#define GPIO_pin 5
uint32_t Freq = 0;

void setup() {
  pinMode(GPIO_pin, OUTPUT);
  Serial.begin(115200);
  Freq = getCpuFrequencyMhz();
  Serial.print("CPU Freq = ");
  Serial.print(Freq);
  Serial.println(" MHz");
  Freq = getXtalFrequencyMhz();
  Serial.print("XTAL Freq = ");
  Serial.print(Freq);
  Serial.println(" MHz");
  Freq = getApbFrequency();
  Serial.print("APB Freq = ");
  Serial.print(Freq);
  Serial.println(" Hz");
}
```

Serial Output for DevKit V1

```
CPU Freq = 240 MHz
XTAL Freq = 40 MHz
APB Freq = 80000000 Hz
```

```
void loop(){
 digitalWrite(GPIO_pin, 1);
 digitalWrite(GPIO_pin, 0);
}
```

Frequency of the generated signal

816.33 kHz

From https://deepbluembedded.com/esp32-change-cpu-speed-clock-frequency/

# CPU and Bus frequency

**Changing the frequency**

```
#define GPIO_pin 5
uint32_t Freq = 0;

void setup() {
// 240, 160, 80 <<< For all types of XTAL crystal
// 40, 20, 10 <<< For 40MHz XTAL
// 26, 13 <<< For 26MHz XTAL
// 24, 12 <<< For 24MHz XTAL
  setCpuFrequencyMhz(80);

  pinMode(GPIO_pin, OUTPUT);
  Serial.begin(115200);
  Freq = getCpuFrequencyMhz();
  Serial.print("CPU Freq = ");
  Serial.print(Freq);
  Serial.println(" MHz");
  Freq = getXtalFrequencyMhz();
  Serial.print("XTAL Freq = ");
  Serial.print(Freq);
  Serial.println(" MHz");
  Freq = getApbFrequency();
  Serial.print("APB Freq = ");
  Serial.print(Freq);
  Serial.println(" Hz");
}

void loop(){
 digitalWrite(GPIO_pin, 1);
 digitalWrite(GPIO_pin, 0);
}
```

Serial Output for DevKit V1

```
CPU Freq = 80 MHz
XTAL Freq = 40 MHz
APB Freq = 80000000 Hz
```

Frequency of the generated signal

272.11 kHz = 816.33 / 3

# Faster execution - Use another core!

```
void Task1code( void * pvParameters )
{
 Serial.print("Task1 running on core ");
 Serial.println(xPortGetCoreID());

for(;;) {
 digitalWrite(led1, HIGH);
 digitalWrite(led1, LOW);
}

}

void setup() {

 pinMode(led1, OUTPUT);
//create a task that to execute Task1Core on core 0
 xTaskCreatePinnedToCore( Task1code, /* Task function. */
 "Task1", /* name of task. */
 10000, /* Stack size of task */
 NULL, /* parameter of the task */
 1, /* priority of the task */
 &Task1, /* Task handle to keep track of created task */
 0); /* pin task to core 0 */

}
```

Frequency of the generated signal

1.49 MHz

# References


Xtensa® Instruction Set Architecture (ISA) Reference Manual
https://0x04.net/~mwk/doc/xtensa.pdf

“Overview of Xtensa ISA”, Espressif Systems,
https://dl.espressif.com/github_assets/espressif/xtensa-isa-doc/releases/download/latest/Xtensa.pdf

ESP32 Series Datasheet,
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

“160+ ESP32 Projects, Tutorials and Guides with Arduino IDE”, Random Nerd Tutorials
https://randomnerdtutorials.com/projects-esp32/

“ESP32 Programming Tutorials With Arduino”, DeepBlueMBedded
https://deepbluembedded.com/esp32-programming-tutorials/