Computer Architecture and Embedded Systems
Semester Assignment

Vasileios Nastos
MSc in department of Informatics and Telecommunications
University of Ioannina
Arta, 47100, Greece
Id:137

Panagiotis Koromilias
MSc in department of Informatics and Telecommunications
University of Ioannina
Arta, 47100, Greece
Id:150

February 13, 2023

1 ESP32 microcontroller family

The ESP32 is a low-cost, low-power system-on-a-chip (SoC) microcontroller family developed by Espressif Systems. It is based on the Tensilica Xtensa LX6 microprocessor and features integrated Wi-Fi and Bluetooth capabilities. The ESP32 microcontroller family includes a variety of different modules and development boards, all of which are based on the same underlying technology and architecture. Some of the most common ESP32 modules and boards include:

- ESP32-S2: A highly integrated, low-power Wi-Fi microcontroller that is designed for low-cost IoT applications.
- ESP32-C3: A Wi-Fi microcontroller that is optimized for high performance and low power consumption.
- ESP32-WROOM: A Wi-Fi and Bluetooth module that is specifically designed for IoT applications.

In terms of hardware features, the ESP32 microcontroller family is equipped with a number of powerful peripherals, including a high-performance CPU, a large amount of RAM and flash memory, and a variety of communication interfaces. Additionally, the ESP32 supports a wide range of operating systems and development tools, making it easy for developers to create new applications and devices using this platform. Overall, the ESP32 microcontroller family provides a flexible, low-cost, and low-power solution for a wide range of IoT applications, making it a popular choice among developers, hobbyists, and engineers. For the assignment we use ESP32-C3 microcontroller which is described in Section 1.2.

1.1 Generations

The ESP32 is a series of low-cost, low-power system-on-a-chip (SoC) microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. There are several generations of ESP32 chips, each with different specifications and features. Here are some of the most commonly used ESP32 generations:

- **ESP32-S0WD:** This chip has a single core Xtensa LX6 microprocessor and supports dual-band Wi-Fi (2.4 GHz and 5 GHz) and dual-mode Bluetooth (Classic and BLE). It also has 4 MB of on-board flash memory and 520 KB of SRAM. The ESP32-S0WD is primarily used for IoT applications and provides a low-cost, low-power solution for wireless connectivity.
- **ESP32-S2:** This chip has a single core Xtensa LX7 microprocessor and also supports dual-band Wi-Fi (2.4 GHz and 5 GHz) and dual-mode Bluetooth (Classic and BLE). It has improved security features, including a secure boot and flash encryption, making it well suited for applications

that require secure data transmission. In addition, the ESP32-S2 is optimized for low-power applications and has a deep sleep current of just 20 uA.

- **ESP32-C3:** This chip has a single core Xtensa LX7 microprocessor and supports dual-band Wi-Fi (2.4 GHz and 5 GHz) and dual-mode Bluetooth (Classic and BLE). It has improved performance and power efficiency compared to previous generations, with a maximum clock speed of 160 MHz. The ESP32-C3 also has a new peripheral interface for enhanced functionality, including support for SPI, I2C, UART, and I2S.

It is worth noting that each generation of the ESP32 has its own unique set of features and specifications, so it's important to choose the right chip for your project based on your specific needs.

1.2 ESP32-C3

The ESP32-C3(Figure 1) is a low-cost, low-power microcontroller developed by Espressif Systems. It is a member of the ESP32 microcontroller family and is designed specifically for IoT applications. The ESP32-C3 is based on the Tensilica Xtensa LX7 microprocessor and features integrated Wi-Fi and Bluetooth capabilities. One of the key features of the ESP32-C3 is its high-performance CPU, which provides fast processing speeds and a high level of computational power. This allows the ESP32-C3 to handle a variety of complex IoT applications and tasks. Additionally, the ESP32-C3 has a large amount of RAM and flash memory, which provides ample storage space for program code and data. Another important feature of the ESP32-C3 is its integrated Wi-Fi and Bluetooth capabilities, which allow it to connect to the Internet and other devices wirelessly. This makes the ESP32-C3 ideal for a wide range of IoT applications, such as home automation, smart lighting, and wearable devices. The ESP32-C3 also supports a number of communication interfaces, including USB, UART, I2C, and SPI. This allows the microcontroller to interact with a variety of other devices and peripherals, making it a highly flexible solution for IoT applications. In terms of power consumption, the ESP32-C3 is designed for low-power operation, making it ideal for battery-powered applications. Additionally, the ESP32-C3 supports a number of power-saving modes and features, which can be used to further reduce power consumption and extend battery life. Overall, the ESP32-C3 is a highly versatile and capable microcontroller that is well-suited for a wide range of IoT applications. Its combination of high performance, low power consumption, and integrated Wi-Fi and Bluetooth capabilities make it an attractive option for developers and engineers looking to build new IoT devices.

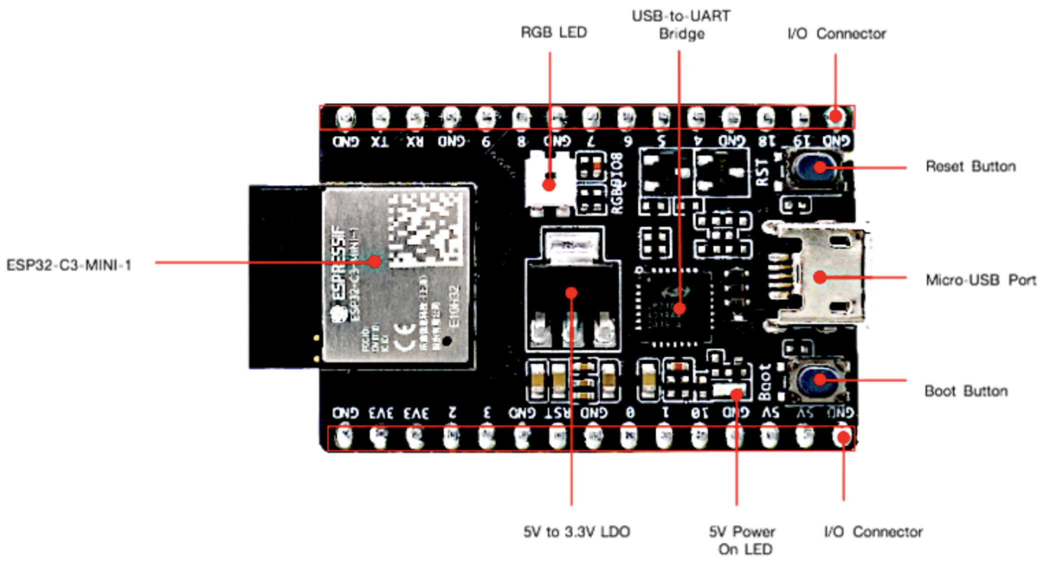


Figure 1: Esp32 Microcontroller[3]

1.3 Basic Competitors

There are several microcontroller chips that compete with the ESP32 in the market for Internet of Things (IoT) applications. Here are some of the most common competitors:

- **nRF52840:** This is a low-power, high-performance chip from Nordic Semiconductor that is based on an Arm Cortex-M4F microprocessor. It supports both Bluetooth 5.0 and Thread (a low-power, secure IoT networking protocol), and has a wide range of peripheral interfaces.
- **CC3220SF:** This is a low-power, high-performance chip from Texas Instruments that is based on an Arm Cortex-M4F microprocessor. It supports both Wi-Fi and Bluetooth 4.2, and has a wide range of peripheral interfaces.
- **Arduino MKR:** This is a low-power, high-performance platform from Arduino that is based on the SAM3X8E microcontroller from Atmel (now part of Microchip). It is designed for IoT and other electronics projects and offers a user-friendly, visual programming environment. The Arduino MKR is less powerful than the ESP32 in terms of processing power and peripheral interfaces, but is well suited for hobbyist and maker projects due to its ease of use.

2 Application development environment

2.1 Arduino IDE

The Arduino Integrated Development Environment (IDE) [4],[5],[6] is a free, open-source software platform for writing, uploading, and debugging code for microcontroller boards based on the Arduino platform. It is designed to provide a simple and user-friendly interface for beginners and experienced users alike. The Arduino IDE is a cross-platform software that can be installed on Windows, MacOS, and Linux operating systems. It includes a text editor, compiler, and uploader, making it easy to write, test, and upload code to the microcontroller board. The text editor supports syntax highlighting and auto-completion, helping users to write code more quickly and accurately. The Arduino IDE supports the C and C++ programming languages, which is the most commonly used languages for developing applications for the Arduino platform. The software includes a library manager that makes it easy to import and use libraries and other resources, such as example sketches, that are needed for a project. One of the key advantages of the Arduino IDE is its simplicity. The software is designed to be easy to use, even for those with no prior programming experience. The interface is straightforward and intuitive, and the built-in libraries and examples make it easy for users to get started on their projects quickly.

The Arduino IDE provides a number of additional features and tools that make it easier to develop applications for the ESP32. Some of the key features of the Arduino IDE include:

- **Simplified coding:** The Arduino IDE provides a simple and straightforward interface for writing, testing, and uploading code to the ESP32. The software includes a text editor that supports syntax highlighting and auto-completion, making it easy to write code.
- **Built-in libraries:** The Arduino IDE includes a large number of built-in libraries that provide additional functionality for the ESP32. For example, the WiFi library provides functions for connecting to Wi-Fi networks, while the Bluetooth library provides functions for connecting to Bluetooth devices.
- **Built-in libraries:** The Arduino IDE includes a large number of built-in libraries that provide additional functionality for the ESP32. For example, the WiFi library provides functions for connecting to Wi-Fi networks, while the Bluetooth library provides functions for connecting to Bluetooth devices.
- **Debugging tools:** The Arduino IDE includes a debugger that makes it easy to find and fix errors in your code. The debugger allows you to step through your code line by line, inspect variables, and set breakpoints to stop the execution of your code.

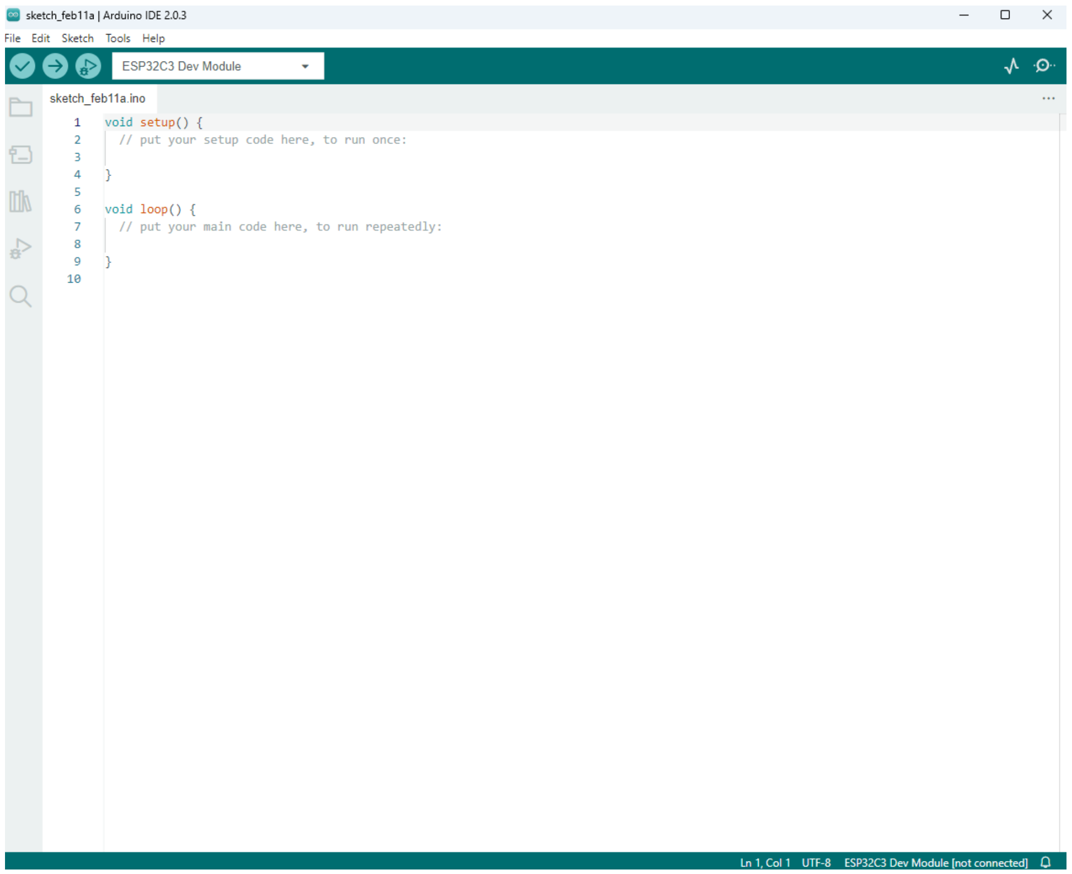
- **Easy library management:** The Arduino IDE includes a library manager that makes it easy to install and manage libraries for your projects. With the library manager, you can search for libraries, view documentation, and install libraries with just a few clicks.

- **Support for different boards:** The Arduino IDE supports a wide range of boards, including the ESP32. The software includes board definitions and libraries for each supported board, making it easy to select the right board for your project.

- **Cross-platform compatibility:** The Arduino IDE is available for Windows, MacOS, and Linux operating systems, making it easy to use on any computer. The software provides a consistent user experience across all platforms, making it easy to switch between computers if needed.

These are just some of the key features of the Arduino IDE for the ESP32. Whether you are a beginner or an experienced developer, the software provides a comprehensive and user-friendly environment for programming and developing applications for the ESP32.

Overall, the Arduino IDE is an excellent tool for developing applications for the ESP32 microcontroller. Whether you are a beginner or an experienced developer, the software provides a comprehensive and user-friendly environment for programming and developing applications for the ESP32.


sketch_feb11a | Arduino IDE 2.0.3
File Edit Sketch Tools Help
ESP32C3 Dev Module
sketch_feb11a.ino

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Ln 1, Col 1 UTF-8 ESP32C3 Dev Module [not connected]

Figure 2: Arduino IDE

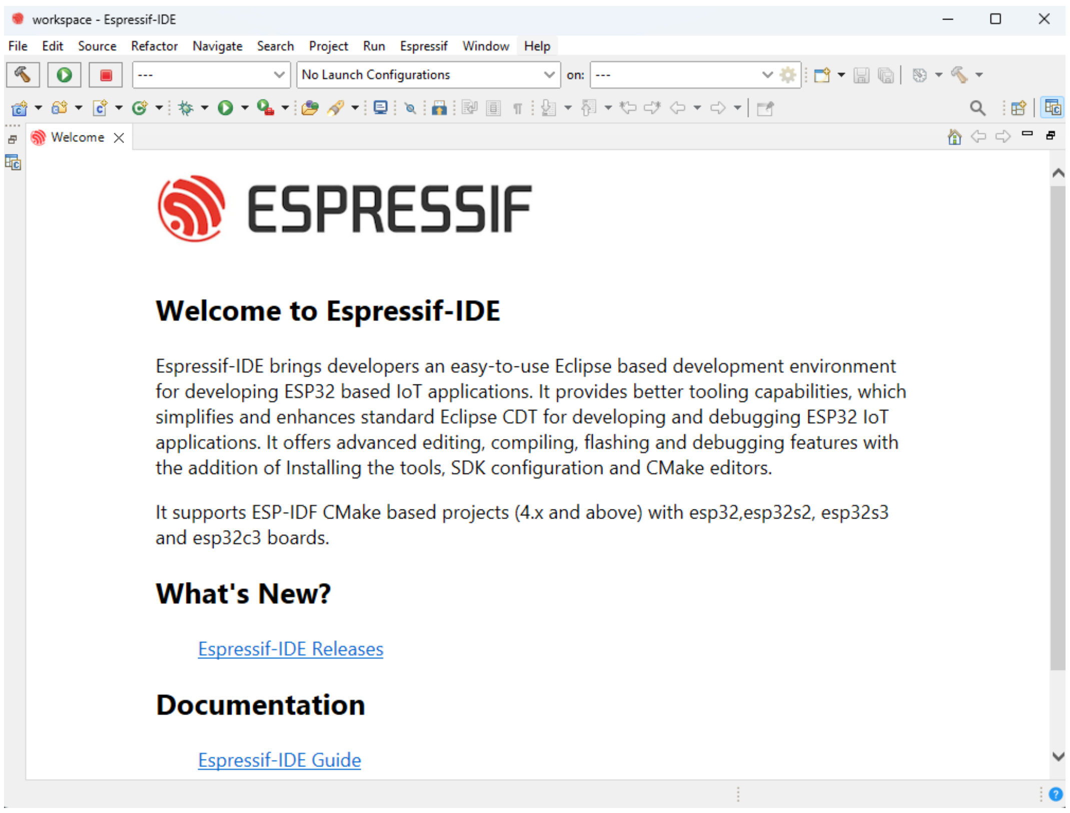
2.2 Espressif IDE

The Espressif Integrated Development Environment (IDE)(**??**) is a software development tool designed specifically for developing applications for the ESP32 microcontroller. It provides a comprehensive set of tools and resources for developing, debugging, and testing applications on the ESP32. The following is a brief overview of the key features and capabilities of the Espressif IDE:

- **Code Editor:** The Espressif IDE provides a built-in code editor with syntax highlighting, code completion, and other features to make writing code for the ESP32 easier.

- **Debugger:** The Espressif IDE includes a debugger that allows developers to step through their code, set breakpoints, and examine variables and memory while their application is running.
- **Library Manager:** The Espressif IDE provides a Library Manager that makes it easy to find and install libraries for the ESP32, as well as to manage library dependencies.
- **Serial Monitor:** The Espressif IDE includes a serial monitor that allows developers to send and receive data between the ESP32 and their computer, making it easier to debug and test applications.
- **Project Templates:** The Espressif IDE provides a range of project templates for different applications and functions, making it easier for developers to get started with a new project.
- **OTA (Over-the-Air) Updates:** The Espressif IDE includes support for OTA updates, allowing developers to easily upload new versions of their code to the ESP32 wirelessly, without having to physically connect it to their computer.
- **Documentation and Examples:** The Espressif IDE includes a range of documentation and examples that provide detailed information on the ESP32, as well as step-by-step guides for using the various features and tools of the IDE.

The Espressif IDE is an open-source tool, and its source code is available on Github. This makes it easy for developers to contribute to the development of the IDE, and to modify it to meet their specific needs. Overall, the Espressif IDE provides a powerful and versatile development environment for developing applications for the ESP32, making it an ideal platform for IoT and embedded applications.


workspace - Espressif-IDE
File Edit Source Refactor Navigate Search Project Run Espressif Window Help
No Launch Configurations
on:
Welcome
ESPRESSIF

Welcome to Espressif-IDE

Espressif-IDE brings developers an easy-to-use Eclipse based development environment for developing ESP32 based IoT applications. It provides better tooling capabilities, which simplifies and enhances standard Eclipse CDT for developing and debugging ESP32 IoT applications. It offers advanced editing, compiling, flashing and debugging features with the addition of Installing the tools, SDK configuration and CMake editors.

It supports ESP-IDF CMake based projects (4.x and above) with esp32,esp32s2, esp32s3 and esp32c3 boards.

What's New?

Espressif-IDE Releases

Documentation

Espressif-IDE Guide

Figure 3: Espressif IDE

2.3 Libraries

The ESP32 is a powerful and versatile microcontroller that supports a wide range of libraries for different applications and functions. These libraries simplify the process of writing code for the ESP32, by providing pre-written, tested, and optimized code that can be easily integrated into a project. The following is a brief overview of some of the most popular libraries available for the ESP32:

- **WiFi Library:** This library provides an easy-to-use interface for connecting the ESP32 to Wi-Fi networks, as well as for controlling and communicating with other devices over Wi-Fi. It includes support for both station mode and softAP mode, and supports a range of security protocols, including WPA2-PSK and WEP.
- **Bluetooth Library:** This library provides support for Bluetooth Low Energy (BLE) on the ESP32, allowing users to develop BLE-enabled applications. It supports a range of BLE profiles and services, and includes support for central and peripheral modes.
- **IoT Libraries:** A range of libraries are available for developing IoT applications on the ESP32, including the MQTT and HTTP libraries. The MQTT library provides support for the MQTT protocol, which is widely used for IoT communication, and the HTTP library provides support for HTTP communication.
- **Sensor Libraries:** A range of libraries are available for working with different types of sensors, such as the BME280 library for temperature, humidity, and pressure sensors, and the MPU6050 library for accelerometer and gyroscope sensors. These libraries provide pre-written code for reading and processing data from the sensors, making it easier to get started with developing IoT applications.
- **Display Libraries:** A number of libraries are available for working with displays, such as the TFT library for TFT displays, and the OLED library for OLED displays. These libraries provide support for displaying text, images, and graphics on the display, and make it easy to get started with creating interactive user interfaces for IoT applications.
- **Audio Libraries:** Libraries are available for working with audio, such as the I2S library for working with audio data, and the VS1053 library for working with MP3 audio files. These libraries provide support for audio input and output, making it easier to develop audio-based applications, such as voice-controlled IoT devices.

These are just a few examples of the many libraries available for the ESP32. The comprehensive range of libraries, together with the powerful development tools and resources available for the ESP32, make it an ideal platform for developing a wide range of IoT and embedded applications.

For the purposes of this assignment we use the Arduino IDE 2.0.3 version combined with the esp32 ESP32 board support package, which includes the necessary drivers, libraries, and tools for programming the chip and C as the programming language [12].

3 The ESP32-DEV-Kit-C embedded application development system

The ESP32-DEV-Kit-C(Figure 4) is an embedded application development system designed for the popular ESP32 microcontroller. This development kit provides a complete and user-friendly platform for developing a wide range of Internet of Things (IoT) and embedded applications. With its advanced features, ease of use, and comprehensive set of development tools and resources, the ESP32-DEV-Kit-C is a powerful and versatile platform for developing IoT applications.

The ESP32-DEV-Kit-C is equipped with the ESP32-C3 microcontroller, which is a low-power and high-performance Wi-Fi and Bluetooth enabled chip. The chip is based on the Xtensa® LX7 core and includes a number of advanced features, such as a dual-core processor, low-power sleep modes, and a high-speed Wi-Fi and Bluetooth radio. The ESP32-C3 microcontroller is designed to provide a highly efficient and flexible platform for developing IoT applications.

The ESP32-DEV-Kit-C includes a number of peripherals, such as a micro-USB port for power and programming, a micro-SD card slot for data storage, a 40-pin header for connecting external devices, and a built-in voltage regulator for powering the system. The kit also includes a number of on-board sensors and components, such as a temperature and humidity sensor, an ambient light sensor, and a microphone. These peripherals and components provide a convenient way for users to get started with developing IoT applications, and can be easily integrated into a wide range of projects.

The ESP32-DEV-Kit-C is designed to be easy to use, even for those with no prior experience in embedded application development. The kit includes a comprehensive set of development tools and resources, including the popular Arduino Integrated Development Environment (IDE). This allows users to quickly and easily write, upload, and debug code on the ESP32-DEV-Kit-C.

The Arduino IDE provides a user-friendly and feature-rich environment for writing code for the ESP32-DEV-Kit-C. The software includes a library manager that makes it easy to import and use libraries and other resources, such as example sketches, that are needed for a project. The text editor supports syntax highlighting and auto-completion, helping users to write code more quickly and accurately. The IDE also includes a number of debugging tools, such as the serial monitor, that make it easy to test and troubleshoot code.

In addition to the Arduino IDE, the ESP32-DEV-Kit-C also includes a number of other development tools and resources, such as the Espressif IoT Development Framework (IDF). The IDF is a comprehensive software platform for developing and deploying applications on the ESP32-DEV-Kit-C. It includes a range of libraries, examples, and tools for developing applications that are specifically designed for the ESP32-C3 microcontroller. The IDF provides a powerful and flexible environment for developing IoT applications, and includes support for a wide range of protocols and technologies, such as Wi-Fi, Bluetooth, BLE, and MQTT.

The ESP32-DEV-Kit-C also supports a range of connectivity options, including Wi-Fi, Bluetooth, and BLE. This allows users to connect to a wide range of IoT devices and networks, and to develop applications that make use of these technologies. With its high-speed Wi-Fi and Bluetooth radio, the ESP32-DEV-Kit-C provides a fast and reliable platform for connecting to other devices and networks. The kit also supports a number of other connectivity options, such as Ethernet and I2C, which can be used to connect to other.

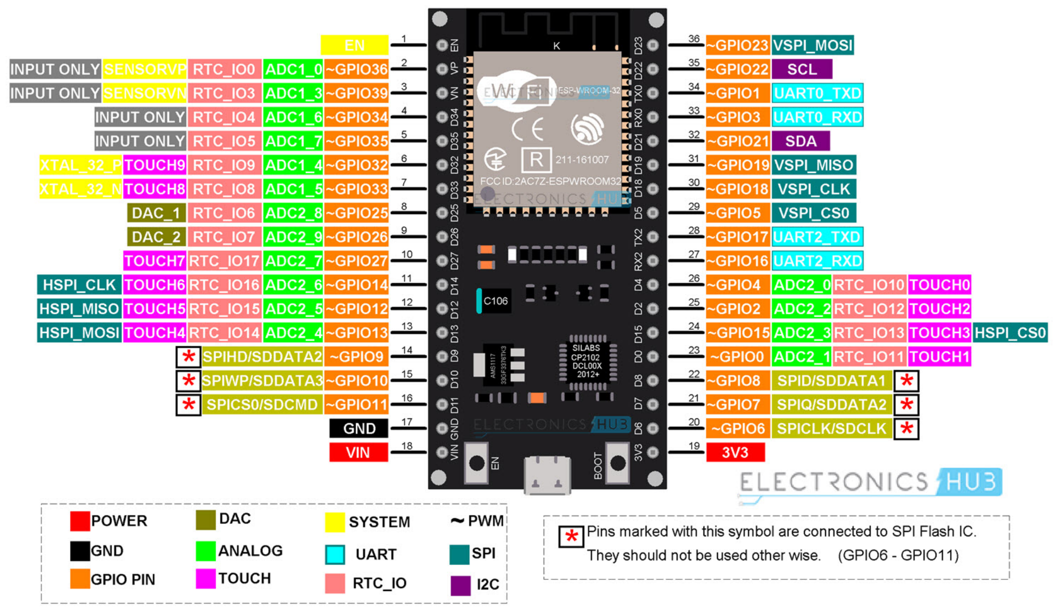

EN 1 EN
INPUT ONLY SENSORVP RTC_IO0 ADC1_0 ~GPIO36 2 VP
INPUT ONLY SENSORVN RTC_IO3 ADC1_3 ~GPIO39 3 VN
INPUT ONLY RTC_IO4 ADC1_6 ~GPIO34 4 D34
INPUT ONLY RTC_IO5 ADC1_7 ~GPIO35 5 D35
XTAL_32_P TOUCH9 RTC_IO9 ADC1_4 ~GPIO32 6 D32
XTAL_32_N TOUCH8 RTC_IO8 ADC1_5 ~GPIO33 7 D33
DAC_1 RTC_IO6 ADC2_8 ~GPIO25 8 D25
DAC_2 RTC_IO7 ADC2_9 ~GPIO26 9 D26
TOUCH7 RTC_IO17 ADC2_7 ~GPIO27 10 D27
HSPI_CLK TOUCH6 RTC_IO16 ADC2_6 ~GPIO14 11 D14
HSPI_MISO TOUCH5 RTC_IO15 ADC2_5 ~GPIO12 12 D12
HSPI_MOSI TOUCH4 RTC_IO14 ADC2_4 ~GPIO13 13 D13
* SPIHD/SDDATA2 ~GPIO9 14 D9
* SPIWP/SDDATA3 ~GPIO10 15 D10
* SPICS0/SDCMD ~GPIO11 16 D11
GND 17 GND
VIN 18 VIN
D23 36 ~GPIO23 VSPI_MOSI
D22 35 ~GPIO22 SCL
TX0 34 ~GPIO1 UART0_TXD
RX0 33 ~GPIO3 UART0_RXD
D21 32 ~GPIO21 SDA
D19 31 ~GPIO19 VSPI_MISO
D18 30 ~GPIO18 VSPI_CLK
D5 29 ~GPIO5 VSPI_CS0
TX2 28 ~GPIO17 UART2_TXD
RX2 27 ~GPIO16 UART2_RXD
D4 26 ~GPIO4 ADC2_0 RTC_IO10 TOUCH0
D2 25 ~GPIO2 ADC2_2 RTC_IO12 TOUCH2
D15 24 ~GPIO15 ADC2_3 RTC_IO13 TOUCH3 HSPI_CS0
D0 23 ~GPIO0 ADC2_1 RTC_IO11 TOUCH1
D8 22 ~GPIO8 SPID/SDDATA1 *
D7 21 ~GPIO7 SPIQ/SDDATA2 *
D6 20 ~GPIO6 SPICLK/SDCLK *
3V3 19 3V3
EN BOOT
ELECTRONICS HUB
POWER
GND
GPIO PIN
DAC
ANALOG
TOUCH
SYSTEM
UART
RTC_IO
~ PWM
SPI
I2C
* Pins marked with this symbol are connected to SPI Flash IC. They should not be used other wise. (GPIO6 - GPIO11)

Figure 4: Esp-32 development Kit

4 Implementation of the first application

The first application performs the dimming of two LEDs that are connected to the ESP32. The red LED's dimming is executed using button switch SW1, while the green led dimming is executed using button switch SW2. At the start of the application, both LEDs dim at 1MHZ. By double-clicking on SW2, the flashing frequency of the green led doubled. For the delay at each dimming, we use the delay function. The implementation of the first application is presented in Listing 1.

Listing 1: Implementation of the first application in C

```
// LEDS and BUTTONS
const int GREEN=4;
const int BLUE=5;
const int RED=6;
const int SW1=2;
const int SW2=3;


//  button and led variables
int buttonSwitch1 = 0;
int buttonSwitch2 = 0;
int button1_pressed = 0;
int button2_pressed = 0;
int green_state = 0;
int red_state = 0;
unsigned long firstClickTime;
bool firstClick = false;

// LED delays
int green_led_delay=500;
int red_led_delay=500;
int fdelay=-1;  // Detarmines the delay at each step on the turn_on_off
    function
int green_led_operation=0;  // if green led operation==1 then green_led_delay
    /2 else  green_led_delay*2

// Setup
void setup() {
  pinMode(GREEN, OUTPUT); //Green Led
  pinMode(BLUE, OUTPUT); //Blue Led
  pinMode(RED, OUTPUT);  //Red Led
  pinMode(SW1, INPUT_PULLUP); //SW1 as input
  pinMode(SW2, INPUT_PULLUP); //SW2 as input
  Serial.begin(9600);
}

void button1_click()
{
  if (buttonSwitch1 == LOW)
  {
    delay(10);
    button1_pressed++;
  }
}

void button2_click()
{
  if (buttonSwitch2 == LOW) {
    if (!firstClick) {
      firstClick = true;
      firstClickTime = millis();
    } else {
      // Detect single click
      if (millis() - firstClickTime < 500) {
        // Double click detected
        digitalWrite(BLUE, HIGH);  // turn the LED on (HIGH is the voltage
            level)
      } else {
        // Single click
        button2_pressed++;
        green_led_operation=!green_led_operation;
        green_led_delay=green_led_operation==1?green_led_delay/2:
            green_led_delay*2;
        digitalWrite(BLUE, LOW);  // turn the LED on (HIGH is the voltage
            level)
      }
      firstClick = false;
    }
  }
}


void turn_on_off(int LED)
{
  fdelay=LED==GREEN?green_led_delay:red_led_delay;
  digitalWrite(LED,HIGH);
  delay(fdelay);
  digitalWrite(LED,LOW);
  delay(fdelay);
}

void set_led_states()
{
  green_state=button1_pressed%2==0?1:0;
  red_state=button2_pressed%2==0?1:0;
}

void loop() {
  // put your main code here, to run repeatedly:
  buttonSwitch1 = digitalRead(SW1);
  buttonSwitch2 = digitalRead(SW2);

  button1_click();
  button2_click();
  set_led_states();


  if (green_state ==1 && red_state ==1)
  {
    digitalWrite(GREEN, HIGH);  // turn the LED on (HIGH is the voltage level)
    digitalWrite(RED, HIGH);  // turn the LED on (HIGH is the voltage level)
    delay(green_led_delay);
    digitalWrite(GREEN, LOW);  // turn the LED off
        if(green_led_delay==250)
        {
                delay(green_led_delay);
        }
    digitalWrite(RED, LOW);   // turn the LED off by making the voltage LOW
  }
  else if (green_state ==1 && red_state ==0)
  {
    turn_on_off(GREEN);
  }
  else if (green_state ==0 && red_state ==1)
  {
    turn_on_off(RED);
  }
  else
  {
    digitalWrite(GREEN, LOW);  // turn the LED on (HIGH is the voltage level)
    digitalWrite(RED, LOW);  // turn the LED on (HIGH is the voltage level)
  }
```

```
}
```

5 Implementation of the second application

In the second application, we expand the logic of the first application by adding timers and Interrupts to our code to implement an optimized version. For timers, we will use timer routines, while for interrupts, we use Interrupt service routines(ISR). The implementation of the second application is presented in Listing 3, while in Listring 2 an optimized version the code use only interrupt routines and perform led dimming is presented is presented.

Listing 2: Implementation of Led dimming using ISR

```
// LEDS and BUTTONS
const int GREEN = 4;
const int BLUE = 5;
const int RED = 6;
const int SW1 = 2;
const int SW2 = 3;

// button and led variables
int buttonSwitch1 = 0;
int buttonSwitch2 = 0;
bool greenBlink = true;
bool redBlink = true;
unsigned long green_led_delay = 500;
unsigned long red_led_delay = 500;

void setup() {
  pinMode(GREEN, OUTPUT); // Green LED
  pinMode(BLUE, OUTPUT);  // Blue LED
  pinMode(RED, OUTPUT);   // Red LED
  pinMode(SW1, INPUT_PULLUP); // SW1 as input
  pinMode(SW2, INPUT_PULLUP); // SW2 as input
  attachInterrupt(digitalPinToInterrupt(SW1), button1_ISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(SW2), button2_ISR, FALLING);
}

// Interrupt Service Routine (ISR) for button 1
void button1_ISR() {
  greenBlink = !greenBlink;
  if (!greenBlink) {
    digitalWrite(GREEN, LOW);
  }
}

// Interrupt Service Routine (ISR) for button 2
void button2_ISR() {
  redBlink = !redBlink;
  if (!redBlink) {
    digitalWrite(RED, LOW);
  }
}

void loop()
{
  buttonSwitch1 = digitalRead(SW1);
  buttonSwitch2 = digitalRead(SW2);
  if (greenBlink) {
    digitalWrite(GREEN, !digitalRead(GREEN));
    delay(green_led_delay);
  }
  if (redBlink) {
    digitalWrite(RED, !digitalRead(RED));
    delay(red_led_delay);
  }
}
```

Listing 3: Full implementation of application 2

```
#include <Arduino.h>

// LEDS and BUTTONS
const int GREEN=4;
const int BLUE=5;
const int RED=6;
const int SW1=2;
const int SW2=3;

// LED states
bool green_state = false;
bool red_state = false;

// LED delays
int green_led_delay=500;
int red_led_delay=500;

// button variables
int button1_presses = 0;
int button2_presses = 0;
bool double_click_detected = false;

// Timer variables
hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR onTimer() {
  portENTER_CRITICAL_ISR(&timerMux);

  static int led_toggle = 0;
  static bool led_on = false;

  // If both buttons are pressed
  if (green_state && red_state) {
    led_toggle = 0;
    led_on = true;
    digitalWrite(GREEN, led_on);
    digitalWrite(RED, led_on);
  }
  // If only green button is pressed
  else if (green_state && !red_state) {
    led_toggle = GREEN;
  }
  // If only red button is pressed
  else if (!green_state && red_state) {
    led_toggle = RED;
  }
  // If no buttons are pressed
  else {
    led_toggle = 0;
    led_on = false;
    digitalWrite(GREEN, led_on);
    digitalWrite(RED, led_on);
  }

  // Toggle the LED if necessary
  if (led_toggle != 0) {
    digitalWrite(led_toggle, led_on);
    led_on = !led_on;
  }

  portEXIT_CRITICAL_ISR(&timerMux);
}

void IRAM_ATTR onButton1Press() {
  button1_presses++;
  green_state = button1_presses % 2 == 0;
}

void IRAM_ATTR onButton2Press() {
  button2_presses++;
  red_state = button2_presses % 2 == 0;

  // Check if double click is detected
  if (!double_click_detected) {
    double_click_detected = true;
    digitalWrite(BLUE, HIGH);
    delay(500);
    digitalWrite(BLUE, LOW);
  } else {
    double_click_detected = false;
    green_led_delay = green_led_delay == 500 ? 250 : 500;
  }
}

void setup() {
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);
  pinMode(RED, OUTPUT);
  pinMode(SW1, INPUT_PULLUP);
  pinMode(SW2, INPUT_PULLUP);

  // Interrupts for buttons
  attachInterrupt(digitalPinToInterrupt(SW1), onButton1Press, FALLING);
  attachInterrupt(digitalPinToInterrupt(SW2), onButton2Press, FALLING);

  // Initialize timer
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &onTimer, true);
  timerAlarmWrite(timer, green_led_delay, true);
  timerAlarmEnable(timer);
}

void loop() {
  // Nothing needs to be done in the loop function
}
```

6 Consclusions

Summing up the two applications, the most important factors worth mentioning are the following:

- The code implements the functionality of a system with two buttons and three LEDs. The buttons are connected to the ESP32 and are used to control the state of the LEDs.

- The first button is used to control the state of the green LED. Each time the button is pressed, the state of the green LED is toggled between on and off.
- The second button is used to control the state of the red LED and also to detect double clicks. Each time the button is pressed, the state of the red LED is toggled between on and off. If the button is double-clicked, the blue LED will turn on for a brief moment and the green LED delay will change.
- The system uses hardware timer to implement the LED flashing behavior. The timer is set to fire at a fixed interval, and in each timer callback, the state of the LEDs is updated based on the button states.
- Interrupts are used to detect button presses and update the button press count and LED state variables. This ensures that the system is responsive to button presses and that the LED state is updated in real-time.
- The code is written in C and uses the Arduino framework, making it easy to understand and modify for those familiar with the Arduino environment. The code uses the ESP32's built-in support for interrupts and hardware timers, making it well-suited for low-power, real-time applications.
- The code uses the IRAM_ATTR attribute for the interrupt and timer callback functions, which ensures that these functions are executed in the fastest memory available on the ESP32, increasing the responsiveness of the system.
- Arduino.h library does contain timer and interrupt functionality. The Arduino platform provides a convenient way to use timers and interrupts in your sketches. Timers can be used for timekeeping, generating regular events, or for precise timing of actions. Interrupts, on the other hand, allow you to perform specific actions in response to external events, such as a button press or a sensor reading, without continuously checking for these events in the main loop.
- The use of C programming language provides the flexibility and control necessary to implement complex tasks on the ESP32, due to the lack of documantation and community in other languages such as Rust.

Overall, the implementation of the application highlights the versatility and potential of the ESP32 in developing advanced applications, and the importance of efficient programming and optimization techniques in enhancing their performance. Also The ESP32 is capable of handling multiple tasks simultaneously through the use of timers and interrupts.

References

[1] *(Esp32-C3,2021)*, https://www.espressif.com/en/products/socs/esp32-c3

[2] *(Eps32-Family, 2022)* https://www.circuitschools.com/what-is-esp32-how-it-works-and-what-you-can-do-with-esp32/

[3] *(Espressif-devkit, 2021)*, https://www.espressif.com/en/news/ESP32-C3-DevKitM-1

[4] *(Javapoint-arduino ,2022)*, https://www.javatpoint.com/arduino-ide

[5] *(AndProf-arduino, 2023)* https://andprof.com/tools/what-is-arduino-software-ide-and-how-use-it/

[6] *(Circuito Team-arduino ,2018)* https://www.circuito.io/blog/arduino-code/

[7] *(Grobotronics-devkitc, 2011)* https://grobotronics.com/esp32-development-board-esp32-s3-devkitc-1.html

[8] *(Digi-Key, 2023)* https://www.digikey.gr/en/products/detail/espressif-systems/ESP32-DEVKITC-32E/12091810

[9] *(Arduino Library List, 2023)* https://www.arduinolibraries.info/architectures/esp32

[10] *(Arduino-Esp, 2022)* https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/lib_builder.html

[11] *(SaaSHub, 2023)*https://www.saashub.com/esp32-alternatives

[12] *(Random Nerd Tutorials, 2016)* https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

[13] *(ElectroRules, 2023)* https://www.electrorules.com/esp32-timer-and-interrupts/

[14] *(Random Nerd Tutorials, 2018)* https://randomnerdtutorials.com/esp32-pir-motion-sensor-interrupts-timers/

[15] *(Circuit-Digest, 2022)* https://circuitdigest.com/microcontroller-projects/esp32-interrupt